

# **DISCIPLINA EA701**

## **Introdução aos Sistemas Embarcados**

**ROTEIRO 10: Sensores e Atuadores com SPI e I2C Integrada  
Transdutores, Sensores e Atuadores Programáveis, Interfaces de  
Comunicação Serial Síncrona SPI e I2C, EEPROM 24LC16,  
LSM6DSV32X, *Display* Nokia**

**Profs. Antonio A. F. Quevedo e Wu Shin-Ting**

**FEEC / UNICAMP**

**Revisado e modificado em maio de 2025 por Ting com auxílio do Chatgpt  
Revisado em outubro de 2024**



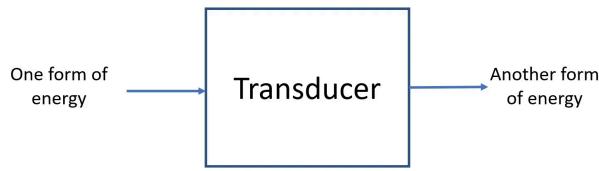
This work is licensed under Creative Commons  
Attribution-NonCommercial-ShareAlike 4.0 International. To view a copy of this  
license, visit <https://creativecommons.org/licenses/by-nc-sa/4.0/>

<b>INTRODUÇÃO</b>	<b>2</b>
<b>PROJETOS-EXEMPLO</b>	<b>4</b>
Projeto de comunicação com uma EEPROM I2C	5
Projeto de comunicação com um display SPI	11
Projeto de comunicação com um sensor de movimento I2C	19
<b>FUNDAMENTOS TEÓRICOS</b>	<b>22</b>
SENSORES E ATUADORES	23
Classificação de sensores quanto às grandezas físicas medidas	25
Classificação de sensores quanto aos princípios de operação	32
Classificação de sensores quanto à fonte de energia	35
Classificação de atuadores quanto às grandezas físicas geradas	36
Classificação de atuadores de movimento quanto à energia primária	38
Classificação de atuadores quanto ao consumo de energia	40
Calibração de transdutores	41
CONEXÃO DOS TRANSDUTORES COM MICROCONTROLADORES	41
Conexão direta	41
Conexão estruturada	42
Conexão estruturada e compatível	43
Programação de transdutores	44
COMUNICAÇÃO SERIAL SÍNCRONA	45
Interface serial SPI	46
Interface serial I2C	49
Exemplo: Comunicação I2C com memória EEPROM 24CXX	57
EXEMPLOS DE SENORES E ATUADORES PROGRAMÁVEIS	61
Display NOKIA 5110	61
Sensor inercial MPU6050	63
Unidade de medição inercial LSM6DSV32X	67
<b>STM32H7A3</b>	<b>70</b>
Módulo SPI	70
Módulo I2C	78

## INTRODUÇÃO

Nos roteiros anteriores, exploramos diversos sensores e atuadores. Utilizamos sensores digitais, como o botão azul e o *keypad* de membrana, e atuadores digitais, como os LEDs. No Roteiro 9, avançamos para sensores analógicos, como potenciômetros, sensores de temperatura e *joysticks*, além de atuadores analógicos, como *buzzers* e motores DC de 5V. Todos esses dispositivos, sejam digitais ou analógicos, são classificados como **transdutores**.

**Sensores e atuadores** são subclasses de transdutores, fundamentais para a comunicação entre sistemas elétricos e o mundo físico.



Transdutores permitem converter energia elétrica em formas como energia mecânica (motor), luminosa (lâmpada) ou sonora (*buzzer*), ou realizar o caminho inverso, transformando estímulos físicos em sinais elétricos que podem ser interpretados pelo microcontrolador. Essa capacidade de conversão viabiliza a interface entre o ambiente físico e o sistema digital: o microcontrolador capta informações por meio dos sensores e atua no ambiente por meio dos atuadores. Podemos comparar esse sistema ao corpo humano: o microcontrolador seria o cérebro, os sensores os órgãos dos sentidos e os atuadores os músculos. Para que essa interação ocorra de forma eficaz, é essencial que haja comunicação entre os transdutores e o microcontrolador. Em alguns casos, essa comunicação é simples, baseada na leitura direta de níveis lógicos ou sinais analógicos. No entanto, à medida que os sistemas se tornam mais complexos, a troca de informações passa a exigir protocolos seriais de comunicação, como I2C e SPI, que possibilitam o controle e a leitura de dados de dispositivos mais sofisticados.

Esses transdutores mais avançados vão além da simples conversão de energia: eles incorporam níveis variados de inteligência, sendo capazes de realizar processamento local, aplicar filtros, fornecer *feedback* ou até mesmo ajustar automaticamente seus parâmetros de operação. Em vez de apenas receberem ou enviarem sinais simples, esses dispositivos podem se comunicar com o microcontrolador por meio de comandos estruturados, como leitura de registradores internos ou envio de pacotes de dados. Muitos desses transdutores “mais autônomos” integram microcontroladores, condicionadores de sinal e interfaces de comunicação digital em um único *chip*. Essa arquitetura embutida reduz a complexidade de conexão física com o sistema principal e transfere parte do processamento para o próprio dispositivo. Assim, o desenvolvedor pode interagir com esses transdutores de maneira semelhante à programação de um periférico interno de um microcontrolador, configurando registradores e acessando informações através de protocolos como I2C ou SPI.

A preferência pelas interfaces I2C e SPI se deve a diversos fatores. Um dos principais é a redução no número de pinos, algo essencial em projetos de sistemas embarcados, onde o espaço e os recursos são limitados. Essas interfaces seriais síncronas permitem conectar múltiplos dispositivos com poucos fios, resultando em sistemas mais compactos e organizados. Além disso, sua robustez e flexibilidade tornam esses protocolos ideais para aplicações que exigem comunicação rápida e confiável. Em ambientes onde vários dispositivos compartilham o mesmo barramento, a capacidade de organizar e priorizar a comunicação serial se mostra extremamente vantajosa. A simplicidade e eficiência dessas

interfaces também contribuem para sua ampla adoção em sensores e atuadores considerados “inteligentes”.

Neste roteiro, vamos explorar não apenas o papel dos transdutores nos sistemas embarcados modernos, mas também como se dá sua comunicação com os microcontroladores, com ênfase no uso dos protocolos I2C e SPI. Ao longo deste Roteiro, analisaremos em detalhes o funcionamento desses protocolos, suas vantagens, limitações e critérios de escolha. Na parte prática, focaremos na programação de uma categoria específica de dispositivos: sensores e atuadores digitais que utilizam interfaces seriais síncronas para comunicação. Serão desenvolvidos três projetos-exemplo que demonstram como configurar e controlar periféricos por meio dos módulos I2C e SPI do microcontrolador STM32H7A. No primeiro projeto, aprenderemos a expandir a memória do microcontrolador conectado um *chip* de memória EEPROM a ele via um módulo I2C. No segundo projeto, exploraremos um atuador programável, o *display* Nokia, que converte sinais elétricos em imagens, demonstrando como operar esse periférico via módulo SPI. E, no último projeto, ampliaremos as funcionalidades do microcontrolador com um sensor MEMS, mais especificamente a IMU (em inglês *Inertial Measurement Unit*) LSM6DSV32X, que reúne um acelerômetro e um giroscópio, ambos acessados através de um módulo I2C. Ao longo dessas atividades, será possível perceber como o uso de bibliotecas de funções compartilhadas pela comunidade de sistemas embarcados facilita a configuração e operação de periféricos mais complexos, tornando sua programação mais acessível.

## PROJETOS-EXEMPLO

Nesta seção, exploraremos dois projetos práticos que ilustram o potencial da comunicação serial entre microcontroladores, sensores e atuadores programáveis. O primeiro projeto, focado em saída de dados, utiliza um *display* gráfico para apresentar informações em tempo real. O segundo, voltado para entrada de dados, emprega uma unidade de medição inercial (IMU) para captar informações do ambiente.

Antes de avançarmos para a programação desses dois periféricos, será apresentado um projeto introdutório com o uso de uma memória EEPROM externa, que servirá como exemplo didático de como programar uma interface de comunicação serial síncrona I2C. Esse projeto prepara o terreno para compreender, de forma mais estruturada, a troca de dados entre dispositivos por meio de um protocolo de comunicação serial síncrona.

Nos três projetos-exemplo, a complexidade dos protocolos de comunicação será abstraída por meio de funções dedicadas. Elas estão disponíveis na “BibliotecaRot10.zip”, um arquivo compactado que acompanha este roteiro.

Precisamos incluir dois desses arquivos, um de extensão .c e outro de extensão .h dentro de cada projeto. Eles podem ser incluídos em qualquer lugar dentro da pasta “Core”, mas aqui, para manter a organização dos arquivos, vamos colocar o “.c” na pasta “Src” e o “.h” na pasta “Inc”. Para isso, há três formas possíveis. A primeira é criar arquivos vazios dentro das pastas clicando com o botão **direito** na pasta e escolhendo a opção “New - Source File” para o “.c” e

“New - Header File” para o “.h”. Depois é necessário copiar o texto de cada arquivo original e colar no arquivo novo correspondente.

As outras duas formas são mais diretas. Em uma delas, abre-se a pasta onde se deseja colocar o arquivo, clicando na mesma com o botão direito, escolhendo “Properties”, e na linha “Location” clicar no botão no limite da linha à direita (*Show in System Explorer*). Em seguida cada arquivo da biblioteca é movido para a pasta correspondente. Por fim, seleciona-se no *Project Explorer* do IDE a raiz do arquivo, clica-se com o botão direito e seleciona-se a opção “Refresh” (ou usa-se a tecla F5). Assim, os arquivos colocados nas pastas serão listados no projeto.

A última forma é mais simples ainda. No Explorador de Arquivos, clica-se no arquivo e arrasta-se o mesmo diretamente para a pasta no *Project Explorer* do IDE. Neste caso, abre-se uma caixa de diálogo. Escolha a opção “Copy files” para que seja feita uma cópia completa do arquivo dentro do projeto, e não apenas um *link* para o arquivo original.

Com as bibliotecas incluídas, os arquivos podem ser abertos e editados. Abra os arquivos e veja a estrutura dos mesmos. Note que cada implementação de função apresenta um cabeçalho de comentários com explicações sobre o uso de cada uma. A sintaxe deste cabeçalho pode parecer estranha, mas ela é usada para permitir um aplicativo chamado **Doxxygen** gerar a documentação de bibliotecas automaticamente, a partir dos comentários das funções. Mesmo sem o Doxygen, os comentários são facilmente compreensíveis pelos programadores.

## Projeto de comunicação com uma EEPROM I2C

As EEPROMs com interface I2C são muito úteis quando se necessita armazenar uma quantidade pequena de dados de forma semi-permanente. Uma aplicação comum é no armazenamento de parâmetros de configuração do sistema, que podem ser alterados mas não mudam com frequência, e precisam permanecer armazenados mesmo quando o sistema se encontra desligado. Embora não se classifique como um sensor ou atuador — já que apenas dados são transferidos entre a memória EEPROM e o microcontrolador, sem resposta ativa — decidimos incluí-la para ilustrar a programação do módulo I2C, permitindo a leitura e escrita de dados em um CI EEPROM [24LC16](#) através do protocolo I2C. Sendo 24C16 o modelo de EEPROM implementado na [placa de Expansão](#), a capacidade de armazenamento dessa memória é 16 kilobits (ou 2 kilobytes), podendo ser escrita em blocos de no máximo 16 bytes. O *device address* de EEPROM é 0b1010xxx que deve ser alinhado à esquerda em conformidade com o protocolo I2C.

1. Crie um projeto chamado “I2C\_EEPROM”, sem inicializar os periféricos. Ative o *Debug* e mantenha o *clock* geral em 64MHz (configurações padrão). Gere o código.
2. Adicione os arquivos “EEPROM\_I2C.c” e “EEPROM\_I2C.h” do arquivo descompactado nas pastas “Src” e “Inc”.

3. Abra o arquivo “main.c”. Vamos incluir a biblioteca no projeto. Na seção `/* USER CODE BEGIN Includes */`, adicione a inclusão:

```
#include "EEPROM_I2C.h"
```

4. Vamos criar variáveis locais para a programação. Na seção `/* USER CODE BEGIN 1 */`, declare as variáveis para os *buffers* de dados, para os *loops* e para receber os códigos de erro:

```
uint8_t bw[16], br1[32], br2[16]; // Buffers de escrita e de leitura
uint8_t k; // Auxiliar de loops
uint8_t errno; // Variavel de erro
```

5. O programa de demonstração será executado apenas uma vez, antes do *loop* infinito. Vamos escrever e ler diferentes conjuntos de dados para demonstrar o funcionamento da EEPROM. Na seção `/* USER CODE BEGIN 2 */`, escreva o código:

```
I2C_Init();
errno = 0;
// Preenche os buffers de leitura com 0xFF
for(k = 0; k < 32; k++) {
    br1[k] = 0xFF;
}
for(k = 0; k < 16; k++) {
    br2[k] = 0xFF;
}

for(k = 0; k < 16; k++) {
    bw[k] = 0; // Escreve uma pagina com zeros
}
errno = Mem_Write(32, bw, 16); // Posicao de memoria 32-47 (pagina 2)
HAL_Delay(5); // Tempo maximo de escrita de uma pagina

for(k = 0; k < 16; k++) {
    bw[k] = 'A'+k; // Escreve sequencia de bytes 0-15
}
errno = Mem_Write(0, bw, 16); // Posicao de memoria 0-15 (pagina 0)
HAL_Delay(5);

for(k = 0; k < 16; k++) {
    bw[k] = k; // Escreve sequencia de bytes 16-31
}
errno = Mem_Write(272, bw, 16); // Posicao de memoria 16-31 (pagina 1)
HAL_Delay(5);

errno = Mem_Read(0, br1, 32); // Le paginas 0 e 1 em uma operacao
errno = Mem_Read(272, br2, 16); // Le pagina 2

HAL_Delay(10); // Apenas para poder colocar um breakpoint
```

Os três argumentos das funções `Mem_Write` e `Mem_Read` correspondem, respectivamente, ao endereço da memória **24LC16**, que vai de 0 a  $(2*1024)-1$  ( $16 \rightarrow 16Kbits = 2^4 2^7 2^3 = 2^4 2^7 bytes$ ), o endereço do vetor de *bytes* a serem escritos ou lidos e a quantidade de *bytes* a serem escritos ou lidos. As duas funções retornam com “1” em caso de erro e “0” se a execução for concluída com sucesso.

Foram realizados três acessos de escrita em blocos de 16 *bytes*. Primeiro, no intervalo de endereços de 32 a 47, foram escritos valores 0x00. Em seguida, no intervalo de 0 a 15, foram gravados os caracteres de 'A' a 'P'. Por último, no intervalo de 16 a 31, foram armazenados os valores de 0 a 15. Após essas operações, foram feitos dois acessos aleatórios para leitura de um bloco de 32 *bytes* a partir do endereço 0 e de 16 *bytes* a partir do endereço 32.

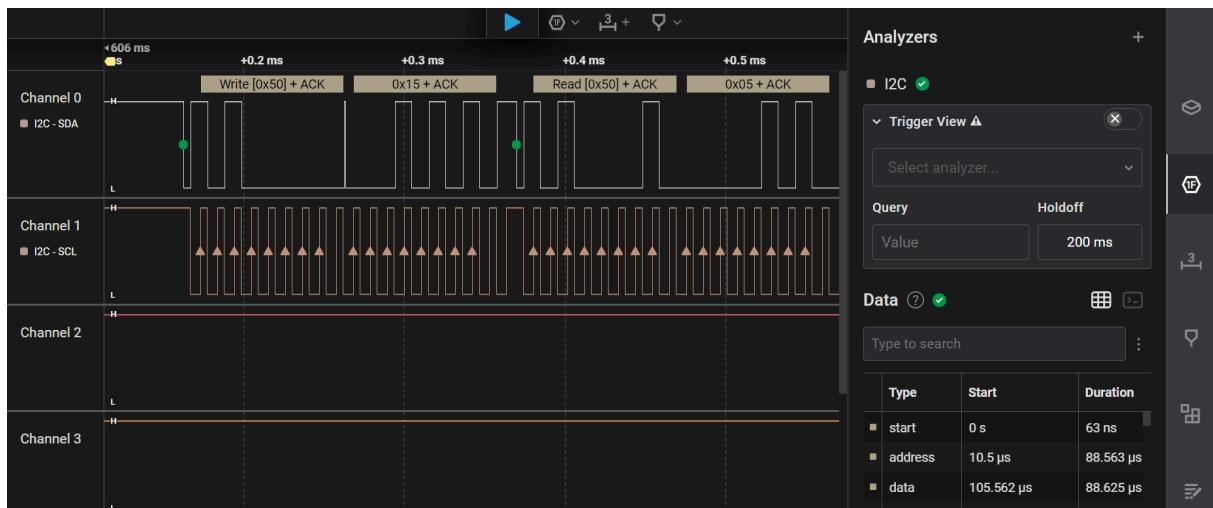
Note que toda operação de escrita só pode envolver um bloco de memória com alinhamento de 16 *bytes*, enquanto que operações de leitura podem passar os limites de blocos.

O preenchimento dos *buffers* de leitura com o valor 0xFF é para garantir que após a execução do programa os mesmos foram realmente preenchidos com os valores lidos na EEPROM. O valor 0xFF não é usado em nenhum *byte* gravado na EEPROM, pelo menos neste projeto.

6. Coloque um *breakpoint* na linha “`HAL_Delay(10);`”. Faça o *Build* e transfira o executável para o microcontrolador no modo *Debug*. Execute o código e observe o conteúdo dos vetores “`br1`” e “`br2`”.

7. Conecte os canais 0 e 1 do analisador lógico aos pinos PB8 (SCL) e PB9 (SDA). Configure a aquisição para 2s e o *trigger* para borda de descida no canal do SDA. Execute novamente (“*Terminate and Relaunch*”) o programa e faça a coleta ao chamar uma função de escrita e também uma de leitura. Configure o analisador para o modo I2C e compare os sinais lidos com o esperado para cada caso (leitura e escrita).

8. Note o uso dos 3 *bits* menos significativos do *device address* e do primeiro *byte* transmitido para definir o endereço de memória dentro da EEPROM a ser acessado, bem como o uso de uma escrita do endereço seguida de um “REPEATED START” para definir o endereço antes de uma operação de leitura. Com base na captura de um acesso de leitura abaixo, qual é o *device address* e qual é o endereço inicial de acesso na memória? Como a condição “REPEATED START” é sinalizada no diagrama de tempo?



9. Vamos investigar como o periférico I2C1 é configurado. Primeiro, adicione um *breakpoint* na função `InitI2C`. Depois, reinicie a execução do código ("Terminate and Relaunch"). Assim que o programa pausar em `InitI2C`, explore a função passo a passo para entender como o registrador `I2C1->TIMINGR` é programado para configurar a taxa de comunicação do I2C1 e como os pinos PB8 e PB9 devem ser configurados para servirem pinos SCL e SDA do periférico I2C1.

```

void I2C_Init(void) {
    RCC->AHB4ENR |= RCC_AHB4ENR_GPIOBEN;
    RCC->APB1LENR |= RCC_APB1LENR_I2C1EN;
    RCC->CDCCIP2R &= ~RCC_CDCCIP2R_I2C123SEL_Msk; //I2C123SEL = 0, PLL1
    I2C1->CR1 = 0;
    I2C1->CR2 = 0;
    // VALOR DE I2Cx_TIMINGR CALCULADO COM PROJETO HAL: 0x10707DBC
    // PRESC = 0x1, SCLL = 0xBC, SCLH = 0x7D, SDADEL = 0x0, SCLDEL = 0x7
    I2C1->TIMINGR = 0x10707DBC;
    I2C1->TIMEOUTR = 0;
    I2C1->CR1 |= I2C_CR1_PE; // Enable
    // Pinos PB8 (SCL) e PB9 (SDA) em AF4, com GPIO em open drain
    GPIOB->OTYPER |= GPIO_OTYPER_OT8 | GPIO_OTYPER_OT9; // Open drain
    GPIOB->AFR[1] &= ~(GPIO_AFRH_AFSEL8_Msk | GPIO_AFRH_AFSEL9_Msk);
    GPIOB->AFR[1] |= GPIO_AFRH_AFSEL8_2 | GPIO_AFRH_AFSEL9_2;
    GPIOB->MODER |= GPIO_MODER_MODE8_Msk | GPIO_MODER_MODE9_Msk;
    GPIOB->MODER &= ~(GPIO_MODER_MODE8_0 | GPIO_MODER_MODE9_0);
}

```

Gostaria de entender por que PB8 e PB9 são configurados como saída *open-drain*. Além disso, o [esquemático](#) mostra que as linhas SCL e SDA são puxadas para VDD por resistores *pull-up*; qual seria a função desses resistores? Não se preocupe se você ainda não tiver as respostas para essas perguntas, pois as explicações serão apresentadas neste Roteiro.

10. Vamos analisar a fundo a operação de escrita na memória. Adicione um *breakpoint* na chamada da função `Mem_Write` e, em seguida, retome a execução (“*Resume*”). Assim que o programa pausar dentro de `Mem_Write`, explore-a detalhadamente, passo a passo, para ver como os elementos são transmitidos na ordem correta: a condição START, o endereço do dispositivo (0b1010), o endereço inicial do bloco de *bytes*, os dados a serem gravados e a condição STOP. Observe atentamente como o processador confirma a recepção com um ACK a cada *byte* enviado.

```
uint8_t Mem_Write(uint16_t memaddr, uint8_t * buf, uint8_t n) {
    uint8_t i, devaddr, lsbaddr;
    lsbaddr = (uint8_t)(memaddr & 0x007F);
    devaddr = (uint8_t)((memaddr >> 7) & 0x000E) + 0xA0;
    I2C1->CR2 |= devaddr | ((n + 1) << I2C_CR2_NBYTES_Pos) |
        I2C_CR2_AUTOEND | I2C_CR2_START;
    while (!(I2C1->ISR & I2C_ISR_TXIS) && !(I2C1->ISR & I2C_ISR_NACKF)) {}
    if (I2C1->ISR & I2C_ISR_NACKF) {
        I2C1->ICR = I2C_ICR_STOPCF;
        I2C1->CR2 = 0;
        return 1;
    }
    I2C1->TXDR = lsbaddr;
    for (i = 0; i < n; i++) {
        while (!(I2C1->ISR & I2C_ISR_TXIS) && !(I2C1->ISR & I2C_ISR_NACKF)) {}
        if (I2C1->ISR & I2C_ISR_NACKF) {
            I2C1->ICR = I2C_ICR_STOPCF;
            I2C1->CR2 = 0;
            return 1;
        }
        I2C1->TXDR = *(buf + i);
    }
    while (!(I2C1->ISR & I2C_ISR_STOPF)) {}
    I2C1->ICR = I2C_ICR_STOPCF;
    I2C1->CR2 = 0;
    return 0;
}
```

Para enviar um vetor de *n bytes* a um 24LC16, começamos configurando o `I2C1_CR2`. Isso inclui o endereço do 24LC16 (0b1010), o número de *bytes* a transferir (*n* + 1, onde *o* + 1 é para o endereço de memória inicial) deslocado pela posição `I2C_CR2_NBYTES_Pos`, a opção `I2C_CR2_AUTOEND` para gerar um STOP automático e, finalmente, o comando para gerar o sinal START.

Em seguida, iniciamos a transferência sequencial, enviando primeiro o endereço de memória e depois os *n bytes* de dados. A cada *byte*, monitoramos o bit `I2C_ISR_TXIS` para confirmar o envio e esperamos pelo ACK. Caso um NACK seja recebido, gerando uma condição de STOP, a flag de parada é imediatamente limpa para restabelecer o estado do barramento I2C.

Através dessas instruções, o que você pode concluir sobre a geração das condições de START, STOP e STOP automático de uma transferência no barramento I2C?

11. Agora, analise a função Mem\_Read de maneira similar. Você notará que o processo de envio do endereço do dispositivo e do endereço de memória a ser lido é o mesmo da escrita. A diferença crucial para a leitura sequencial de dados é o envio de um REPEATED START. Este sinal precede a fase de leitura dos *bytes* de dados propriamente ditos. Ao final da transferência, a condição STOP é enviada. É importante destacar que, durante a leitura, um sinal de ACK é gerado pelo mestre após cada *byte* para indicar que ele foi recebido com sucesso.

```
uint8_t Mem_Read(uint16_t memaddr, uint8_t * buf, uint8_t n) {
    uint8_t i, devaddr, lsbaddr;
    i = I2C1->RXDR;
    lsbaddr = (uint8_t)(memaddr & 0x007F);
    devaddr = (uint8_t)((memaddr >> 7) & 0x000E) + 0xA0;
    i = 0;
    I2C1->CR2 |= devaddr | (1 << I2C_CR2_NBYTES_Pos) |
        I2C_CR2_START; // Escrita de 1 byte sem STOP
    while(!(I2C1->ISR & I2C_ISR_TXIS) && !(I2C1->ISR & I2C_ISR_NACKF)) {}
    if(I2C1->ISR & I2C_ISR_NACKF) {
        I2C1->ICR = I2C_ICR_STOPCF;
        I2C1->CR2 = 0;
        return 1;
    }
    I2C1->TXDR = lsbaddr; // Envia o lsbaddr
    while(!(I2C1->ISR & I2C_ISR_TC)) {}
    I2C1->CR2 |= devaddr | (n << I2C_CR2_NBYTES_Pos) |
        I2C_CR2_AUTOEND | I2C_CR2_RD_WRN | I2C_CR2_START;
    while(i < n) {
        while(!(I2C1->ISR & I2C_ISR_RXNE) && !(I2C1->ISR & I2C_ISR_NACKF)) {}
        if(I2C1->ISR & I2C_ISR_NACKF) {
            I2C1->ICR = I2C_ICR_STOPCF;
            I2C1->CR2 = 0;
            return 1;
        }
        *(buf + i) = I2C1->RXDR;
        i++;
    }
    while(!(I2C1->ISR & I2C_ISR_STOPF)) {}
    I2C1->ICR = I2C_ICR_STOPCF;
    I2C1->CR2 = 0;
    return 0;
}
```

Para ler um vetor de n *bytes* da memória, é realizada uma sequência específica de operações, começando com um endereçamento para escrita seguido de uma leitura:

1. Acesso Inicial (Escrita): O microcontrolador envia o endereço da memória (endereçamento para escrita) e, crucialmente, não gera uma condição de STOP. Isso mantém o barramento sob controle do mestre.
2. START Repetido (Leitura): Em seguida, a operação de leitura (I2C\_CR2\_RD\_WRN) é definida, com o bit de leitura/escrita (R/W) configurado para '1', e uma condição (repetida) de START é emitida (I2C\_CR2\_START). Ao mesmo tempo, o número de bytes a serem lidos (n) é definido em I2C\_CR2\_NBYTES\_Pos, e a opção I2C\_CR2\_AUTOEND é ativada para gerar um STOP automático ao final da leitura.
3. Leitura dos Dados: A coleta de cada byte recebido é feita monitorando-se o bit I2C\_ISR\_RXNE, que indica a disponibilidade de novos dados no buffer de recepção. O mestre gera o sinal ACK (I2C\_ISR\_NACKF em 1) para que o escravo enveja o próximo byte. Se um NACK for detectado (resultando em um STOP), a flag correspondente deve ser limpa imediatamente para redefinir o estado do barramento.

Através dessas instruções, o que você pode concluir sobre as condições de START e de REPEATED START de uma transferência no barramento I2C? Quando é necessário gerar a condição de REPEATED START?

## Projeto de comunicação com um *display* SPI

Os *displays* LCD de texto, como o nome diz, recebem códigos ASCII e apresentam diretamente os caracteres correspondentes. Em *displays* gráficos, a informação transferida ao dispositivo se refere ao estado de cada *pixel*. Assim, a quantidade de dados transmitida é muito maior que para um *display* de texto, exigindo interfaces com boa velocidade de comunicação. O *display* [Nokia 5110](#) possui 84x48 pixels monocromáticos (um bit por pixel), ou seja, um total de 4.032 bits por quadro. Por isso, ele utiliza a interface SPI, em modo unidirecional, funcionando como *slave* de um microcontrolador. Note que para escrever texto neste tipo de *display*, é necessário “desenhar” os caracteres, como *displays* LCD, definindo os pixels a serem ativados para cada um.

A questão que surge é como realizar esses desenhos em tempo real de maneira diversificada. Uma abordagem é enviar uma série de instruções apropriadas para os registradores do *display* Nokia, programando-o através do microcontrolador. Essas instruções são convertidas em sinais de controle que atuam diretamente sobre o *display*. Vamos demonstrar isso, implementando um projeto com o *display* Nokia5110, usando uma **biblioteca** para simplificar a implementação. A **biblioteca** nada mais é que um par de arquivos adicionados ao projeto, sendo um do tipo *header* (extensão “.h”, com macros e protótipos das funções) e um do tipo código-fonte C (extensão “.c”, com a implementação das funções).

1. Crie um projeto chamado “SPI\_Nokia”, sem inicializar os periféricos. Ative o *Debug* e mantenha o *clock* geral em 64MHz (configurações padrão). Gere o código.
2. Adicione os arquivos “Nokia\_5110.c” e “Nokia\_5110.h” do arquivo descompactado nas pastas “Src” e “Inc”.

3. Abra o arquivo “main.c”. Precisamos inicialmente incluir a biblioteca na compilação. Para isso, na seção `/* USER CODE BEGIN Includes */`, adicione as linhas:

```
#include "Nokia_5110.h"
```

Além disso, vamos definir macros para controlar o pino GPIO ligado ao controle da *backlight*. Na seção `/* USER CODE BEGIN PD */`, adicione as macros:

```
// Macros para backlight  
#define BKLT1()    GPIOD->BSRR = GPIO_BSRR_BS5  
#define BKLT0()    GPIOD->BSRR = GPIO_BSRR_BR5
```

4. Vamos também adicionar um vetor com o *bitmap* do logo da FEEC. Este vetor possui a sequência de *bytes* a serem transmitidos ao *display* para que desenhem o *bitmap*. Para isso, o logo foi submetido a um [app online](#) que gera a sequência de *bytes* no formato adequado. Este vetor deve residir na FLASH, e para isso ele é declarado como “static const”. Na seção `/* USER CODE BEGIN PV */`, declare o vetor:

```

0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01, 0x01,
0x03, 0x01, 0x01, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01,
0x01, 0x01, 0x03, 0x01, 0x01, 0x00, 0x00,
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x03, 0x07, 0x1d, 0x33, 0x4f,
0x7f, 0x7e, 0x1c, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};

5. Agora vamos inicializar o display e desenhar o logo da FEEC. Na seção /* USER CODE BEGIN 1 */ declare a variável com inicialização

uint8_t contraste = 52;

Na seção /* USER CODE BEGIN 2 */, faça a chamada das funções da biblioteca:

lcdBegin();
setContrast(contraste);
clearDisplay(WHITE);
SetBitMap(LogoFEEC);
updateDisplay();

Na primeira linha, estamos inicializando a interface SPI e enviando a configuração inicial para o display. Na segunda linha, o contraste é ajustado. Na terceira linha, o buffer do display é limpo e na quarta linha ele é preenchido com o bitmap. A quinta linha atualiza o display. Ou seja, todos os comandos da biblioteca alteram os bits de um buffer na RAM do microcontrolador. Após todos os comandos desejados serem executados, deve-se chamar a função “updateDisplay” para que os bits dos pixels sejam enviados do buffer para o display.

```

Note que o contraste varia entre módulos. O valor indicado costuma funcionar bem para a maioria dos módulos. Se a imagem aparecer muito “fraca”, deve-se aumentar o contraste. Ao contrário, se a imagem aparecer com as áreas claras muito escurecidas, deve-se diminuir o contraste.

6. Para fins de demonstração, vamos usar, além da carga do *bitmap*, o traçado de linhas, o uso de fundo preto e linhas brancas, e a apresentação de texto. Após a linha /\* USER CODE BEGIN 3 \*/, escreva o código do *loop*:

```

HAL_Delay(2000);
clearDisplay(BLACK);
// Desenha cubo com linhas claras em fundo escuro
setRect(25, 10, 45, 30, 0, WHITE);
setRect(35, 20, 55, 40, 0, WHITE);
setLine(25, 10, 35, 20, WHITE);
setLine(45, 30, 55, 40, WHITE);
setLine(25, 30, 35, 40, WHITE);
setLine(45, 10, 55, 20, WHITE);

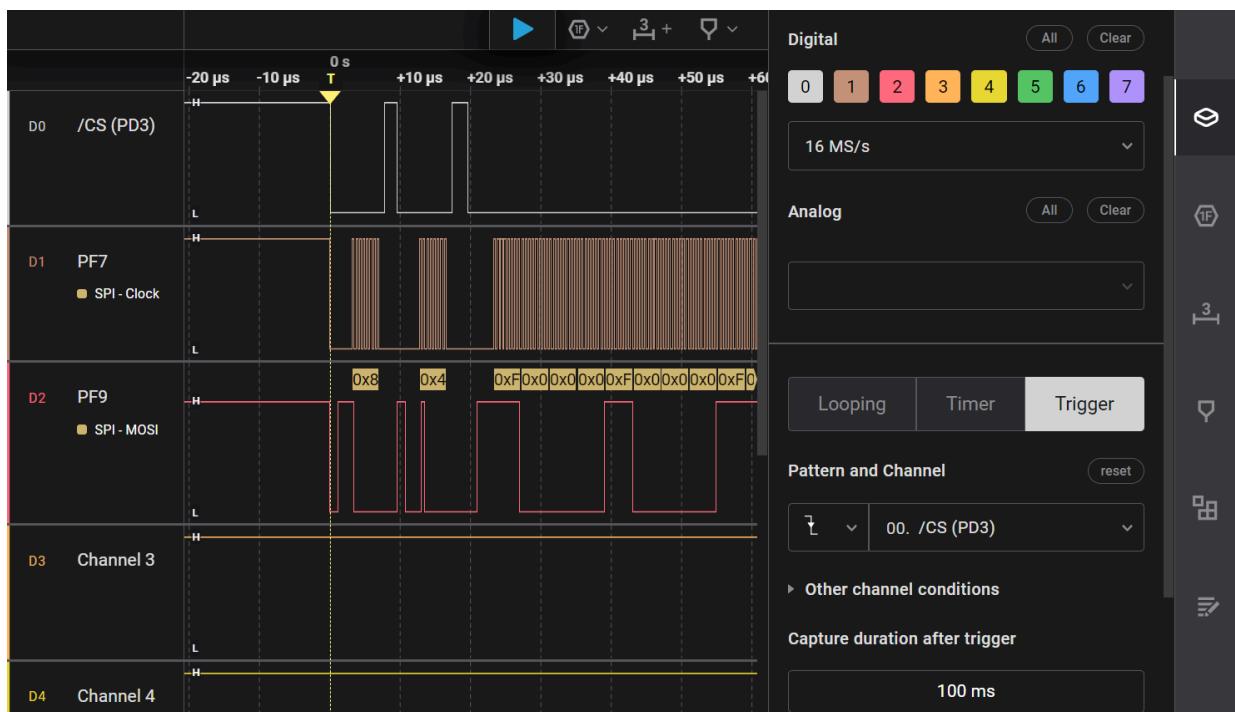
```

```

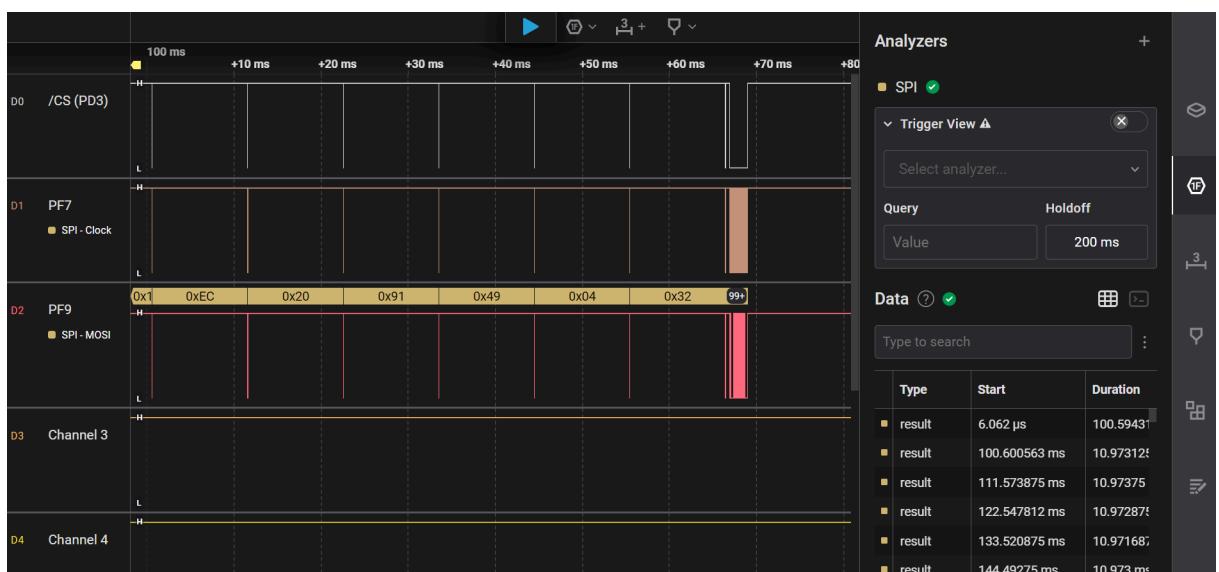
updateDisplay();
BKLT1();
HAL_Delay(2000);
clearDisplay(WHITE);
// Escreve texto escuro em fundo claro
setStr("EA701 FEEC", 0, 0, BLACK);
setStr("Profs.", 0, 8, BLACK);
setStr("Quevedo e Ting", 0, 16, BLACK);
updateDisplay();
HAL_Delay(2000);
clearDisplay(WHITE);
SetBitMap(LogoFEEC);
updateDisplay();
HAL_Delay(2000);
clearDisplay(WHITE);
updateDisplay();
BKLT0();

```

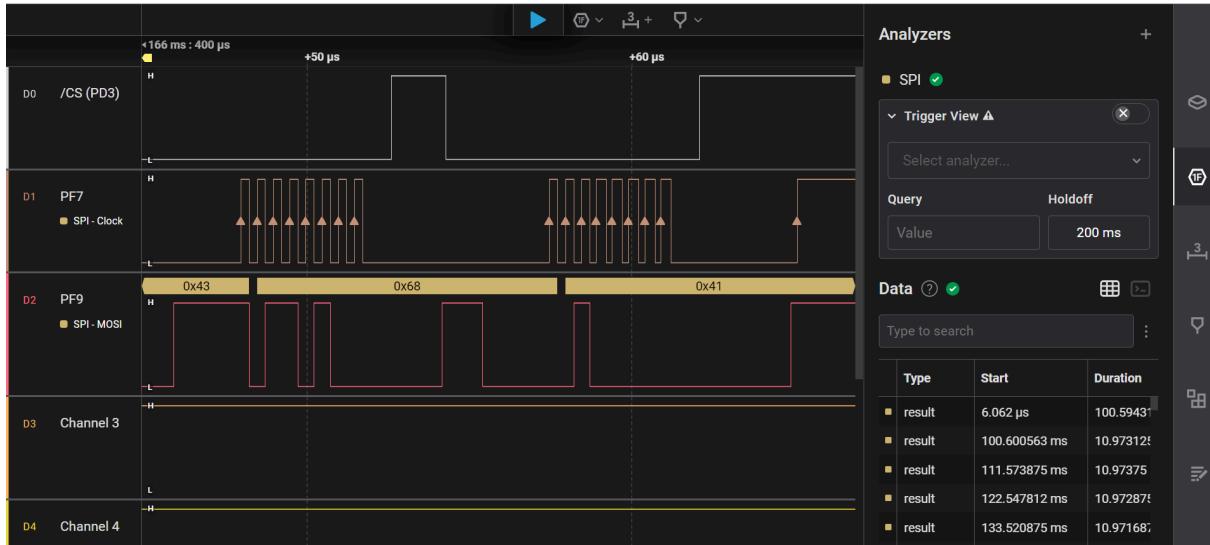
7. Realize o *Build* e transfira o código executável para o microcontrolador no modo *Debug*. Execute o programa e veja a demonstração no *display*.
8. Coloque um *breakpoint* na linha “`setContrast (contraste) ;`” da função `main`. Em seguida, reinicie o programa (“Terminate and Relaunch”). Quando a execução pausar nessa linha, altere o valor da variável `contraste` na aba “Variables”, aumentando seu valor. Depois, continue a execução (“Resume”). Repita o procedimento, desta vez reduzindo o valor da variável. Registre os efeitos observados em diferentes situações e configure o nível de contraste de sua preferência.
8. Remova o *breakpoint*. Ligue os 3 primeiros canais do analisador lógico nos pinos adequados do conector “Zio” (parte superior da placa NUCLEO). Veja [a figura 18, página 37, do manual da placa NUCLEO](#) para localizar os sinais corretos: canal 0 em PD3 (/CS), canal 1 em PF7 (SCK) e canal 2 em PF9 (MOSI). Configure a aquisição do “Logic” para um *trigger* na borda de descida do canal 0 (/CS) e um tempo de aquisição de 200ms.



9. Coloque um *breakpoint* na linha “LCDcommand(0x21);” da função “lcdBegin” do arquivo “Nokia\_5110.c”. Reinicie (“Terminate and Relaunch”) a execução até o *breakpoint*, inicie a aquisição no “Logic” e continue a execução do programa. Adicione o analisador SPI nos sinais, mantendo as configurações padrão e ajustando os canais para cada sinal. Veja a sequência de *bytes* de comando enviados e compare com o [datasheet do display](#) (Capítulo 8 - INSTRUCTIONS, e em especial a tabela 1 na página 14).



10. Faça um “zoom in” na vizinhança de um dos “palitos”. Interprete a relação entre os sinais visualizados, identificando o instante em que o dispositivo escravo (*display*) é selecionado e o instante em que os quadros de dados são enviados para o *display*.



11. Para iniciar a exploração da configuração do periférico SPI, o primeiro passo é adicionar um *breakpoint* na função `lcdBegin`. Em seguida, reinicie a execução do código utilizando a opção "Terminate and Relaunch". Assim que o programa pausar em `lcdBegin`, prossiga entrando na função e, posteriormente, na função `InitInterface`, que está aninhada dentro dela.

```
void InitInterface(void) {
    RCC->APB2ENR |= RCC_APB2ENR_SPI5EN; // Clock gating
    // Fonte de clock: HSI 64MHz, sem uso de FIFO
    // Selecionar HSI de 64MHz como fonte de clock
    RCC->CDCCIP1R &= ~RCC_CDCCIP1R_SPI45SEL_Msk;
    RCC->CDCCIP1R |= RCC_CDCCIP1R_SPI45SEL_1 | RCC_CDCCIP1R_SPI45SEL_0;
    SPI5->CFG1 &= ~SPI_CFG1_MBR_Msk; // divisor de baud rate, de 64MHz
    // para 2MHz, dividimos por 32
    SPI5->CFG1 |= SPI_CFG1_MBR_2; // MBR = 0b100, divide por 32
    SPI5->CFG1 &= ~(SPI_CFG1_CRCEN | // Sem calculo de CRC
                    SPI_CFG1_TXDMAEN | // Desabilita Tx DMA stream
                    SPI_CFG1_RXDMAEN | // Desabilita Rx DMA stream
                    SPI_CFG1_FTHLV_Msk | // FIFO de 1 dado (desativado)
                    SPI_CFG1_DSIZM_Msk); // Data frame size mask
    SPI5->CFG1 |= SPI_CFG1_DSIZM_2 | SPI_CFG1_DSIZM_1 | SPI_CFG1_DSIZM_0;
    // Data frame size = 8 bits
    SPI5->CFG1 |= SPI_CFG1_CRCSIZE_2 |
                    SPI_CFG1_CRCSIZE_1 | SPI_CFG1_CRCSIZE_0;
    SPI5->CFG2 &= ~(SPI_CFG2_AFCNTR | // Sem AF lock
                    SPI_CFG2_SSOE | // Nao usa CS em hardware
                    SPI_CFG2_CPOL | SPI_CFG2_CPHA | SPI_CFG2_LSBFRST | // MSB first
                    SPI_CFG2_SP_Msk |
                    SPI_CFG2_COMM_Msk | // Modo de comunicacao (mascara)
                    SPI_CFG2_IOSWP | // Nao troca pinos MOSI e MISO
                    SPI_CFG2_MIDI_Msk | // Sem delay entre frames de dados
                    SPI_CFG2_MSSI_Msk); // Sem delay entre SC e dados
```

```

    // Bits em 1: SPI MASTER, Simplex transmitter
    SPI5->CR1 = SPI_CR1_SSI; // Todos os bits de CR1 em 0 menos SSI
    // Se SSI = 0 e se configura CFG2 para MASTER, ele não passa para modo master
    SPI5->CFG2 |= SPI_CFG2_SSOM | SPI_CFG2_SSM; // SSM tambem precisa estar em 1 para entrar em MASTER
    SPI5->CFG2 |= SPI_CFG2_MASTER | SPI_CFG2_COMM_0;
    SPI5->CR2 = 0; // TSIZE = 0, transferencia sem FIFO
    //SPI5->IFCR |= SPI_IFCR_EOTC; // Limpa flag EOT
    // Configurando as saidas GPIO em PA4, PD3, PD4 e PD5
    RCC->AHB4ENR |= RCC_AHB4ENR_GPIOAEN | RCC_AHB4ENR_GPIODEN;
    GPIOA->MODER &= ~GPIO_MODER_MODE4_Msk;
    GPIOA->MODER |= GPIO_MODER_MODE4_0;
    GPIOA->OTYPER &= ~GPIO_OTYPER_OT4;
    GPIOD->MODER &= ~(GPIO_MODER_MODE3_Msk |
                       GPIO_MODER_MODE4_Msk | GPIO_MODER_MODE5_Msk);
    GPIOD->MODER |= GPIO_MODER_MODE3_0 |
                     GPIO_MODER_MODE4_0 | GPIO_MODER_MODE5_0;
    GPIOD->OTYPER &= ~(GPIO_OTYPER_OT3
                       | GPIO_OTYPER_OT4 | GPIO_OTYPER_OT5);
    GPIOD->BSRR = GPIO_BSRR_BS3;
    // Funcoes alternativas (AF5) para PF7 (SPI5_SCK) e PF9 (SPI5_MOSI)
    RCC->AHB4ENR |= RCC_AHB4ENR_GPIOFEN;
    GPIOF->MODER &= ~(GPIO_MODER_MODE7_Msk | GPIO_MODER_MODE9_Msk);
    GPIOF->MODER |= GPIO_MODER_MODE7_1 | GPIO_MODER_MODE9_1;
    GPIOF->AFR[0] &= ~GPIO_AFRL_AFSEL7_Msk;
    GPIOF->AFR[0] |= GPIO_AFRL_AFSEL7_0 | GPIO_AFRL_AFSEL7_2;
    GPIOF->AFR[1] &= ~GPIO_AFRH_AFSEL9_Msk;
    GPIOF->AFR[1] |= GPIO_AFRH_AFSEL9_0 | GPIO_AFRH_AFSEL9_2;
}

```

A partir desse ponto, explore a função passo a passo para compreender como os registradores SPI são programados. Observe como o registrador SPI15->CFG1 é utilizado para configurar a taxa de transferência, bem como o formato e o tamanho dos dados. Simultaneamente, analise como o registrador SPI15->CFG2 é programado para ajustar o formato do protocolo, o modo de operação e o papel do microcontrolador (mestre ou escravo) nas transferências.

Além da configuração do próprio periférico SPI, é fundamental entender como os pinos GPIO são definidos para essa comunicação. Verifique como os pinos PF7 e PF9 são configurados para suas funções alternativas de SCLK (do inglês *Serial Clock*) e MOSI (do inglês *Master Out Slave In*), respectivamente. Por fim, note que os pinos PA4, PD3 e PD4 são configurados como GPIOs padrão e desempenham papéis de controle: RST (*Reset*), CE (*Chip Enable*) e DC (seleção entre dados e comandos), respectivamente.

Embora existam muitas configurações não essenciais para este projeto, optamos por mantê-las. Isso serve para demonstrar como é possível adaptar códigos desenvolvidos e

compartilhados pela comunidade de sistemas embarcados às necessidades específicas de cada projeto.

Pelas instruções acima, qual é o formato de protocolo SPI adotado no *display* Nokia 5110? E o modo de operação? E a taxa de transmissão? O que acontecerá se errarmos numa dessas configurações, o escravo retornará um sinal de erro como no protocolo I2C? Se você tiver dúvidas, a melhor abordagem é realizar testes! Experimente diferentes valores de configuração para o SPI e observe o comportamento do *display* Nokia. Essa prática ajuda muito a entender a relação entre o código e o *hardware*.

Percebeu que o periférico SPI do STM32H7A possui FIFOs (do inglês *First-In, First-Out*) internas, separadas para transmissão (TxFIFO) e recepção (RxFIFO)? Tem uma ideia para que servem essas FIFOs? Não se preocupe se não tiver uma resposta ainda. Veremos isso mais adiante.

12. O *display* Nokia 5110, que utiliza o controlador PCD8544, é um dispositivo de escrita unicamente (em inglês, *write-only*). Isso significa que o microcontrolador (o mestre) envia comandos e dados de *pixel* para o *display*, mas o *display* não tem a capacidade de enviar nenhuma informação de volta para o microcontrolador. Por essa razão, ele tipicamente possui apenas a conexão MOSI, e não a MISO (do inglês *Master In Slave Out*), que seria para dados do escravo para o mestre.

A transmissão de dados e comandos para o *display* Nokia 5110 via MOSI é controlada pelo pino DC (conectado a PD4). O nível lógico de PD4 determina a interpretação do *byte* subsequente pelo controlador PCD8544: um nível específico indica um comando de configuração, enquanto o outro indica dados de imagem para exibição. Para facilitar essa distinção no código, foram definidas as macros que devem ser usadas antes de uma transferência:

```
#define LCD_DATA()  GPIOD->BSRR = GPIO_BSRR_BS4  
#define LCD_COMMAND() GPIOD->BSRR = GPIO_BSRR_BR4
```

Em sua essência, a função SPI\_Transmit sequencia as instruções a fim de gerar os sinais SPI de acordo com o modo de operação configurado. Para que você possa explorá-la passo a passo, entre na função setContrast depois de lcdBegin.

```
void SPI_Transmit(uint8_t * d, uint16_t n) {  
  
    uint16_t i;  
    i = n;  
    SPI5->CR2 = (uint32_t)(n & 0x0000FFFF); // TSIZE  
    SPI5->CR1 |= SPI_CR1_SPE; // Enable SPI  
    CS0();  
    SPI5->CR1 |= SPI_CR1_CSTART;  
    while(i > 0) {  
        if(SPI5->SR & SPI_SR_TXP) {
```

```

        SPI5->TXDR = *d;
        i--;
        d++;
    }
}

while (!(SPI5->SR & SPI_SR_EOT)) {}

SPI5->IFCR = SPI_IFCR_EOTC | SPI_IFCR_TXTFC;
SPI5->CR1 &= ~SPI_CR1_SPE;
CS1();
}

```

A configuração inicial para a transmissão SPI envolve definir o número de *bytes* (n) no campo SPI\_CR2\_TSIZE antes de habilitar o periférico com o bit SPI\_CR1\_SPE. Para iniciar a comunicação, a macro CS0() ativa o *Chip Enable* (pino PD3) do *display* Nokia, enquanto o bit SPI\_CR1\_CSTART inicia a transferência. Os n *bytes* são então enviados sequencialmente, com cada *byte* sendo escrito no registrador SPI5->TXDR apenas quando o bit SPI\_SR\_TXP sinaliza que o canal de transmissão está pronto. Ao final da transferência, quando o bit SPI\_SR\_EOT é setado para '1', as flags de fim de transferência e de FIFO cheio são limpas. Somente após essa limpeza, o periférico SPI5 é desabilitado (SPI5->CR1 &= ~SPI\_CR1\_SPE;) e o sinal *Chip Enable* é desativado (CS1).

Com base na análise da função SPI\_Transmit, o que acontecerá se enviarmos bytes com o sinal DC indefinido?

## Projeto de comunicação com um sensor de movimento I2C

Uma memória EEPROM apenas armazena *bytes* em endereços diversos. Um sensor “inteligente” usa a interface digital para receber parâmetros de configuração e enviar os dados coletados, já na forma digital. Isto é feito através de **registradores** internos ao sensor, sendo que cada um ocupa um endereço. A semelhança entre os acessos dos registradores do sensor e um acesso à memória EEPROM via I2C é como a semelhança entre um acesso à memória RAM e a um registrador de periférico interno em uma MCU. No caso do [IMU LSM6DSV32X](#), um sensor de movimentos em 6 eixos, os endereços de registradores são de 1 byte, e assim não são usados bits do *device address* para completar o endereço. **O device address de LSM6DSV32X é configurado em 0b1101010 (0x6A)** neste projeto.

1. Crie um projeto chamado “I2C\_IMU”, sem inicializar os periféricos. Ative o *Debug* e mantenha o *clock* geral em 64MHz (configurações padrão). Gere o código.
2. Adicione os arquivos “LSM6DSV32X.c” e “LSM6DSV32X.h” nas pastas “Src” e “Inc”, respectivamente, como foi feito nos projetos anteriores.
3. Abra o arquivo “main.c”. Vamos incluir a biblioteca no projeto. Na seção `/* USER CODE BEGIN Includes */`, adicione a inclusão:

```
#include "LSM6DSV32X.h"
```

4. Vamos criar variáveis para a programação. Na seção `/* USER CODE BEGIN 1 */`, declare as variáveis para funções auxiliares:

```
uint8_t r, id;
```

Na seção `/* USER CODE BEGIN PV */`, crie as variáveis para guardar os valores dos eixos de aceleração e velocidade angular. Estas variáveis devem ser **globais** para que o recurso a ser usado neste projeto possa funcionar.

```
int16_t ax, ay, az, gx, gy, gz;
int16_t buf[3];
```

5. Agora vamos configurar a interface I2C e o sensor MPU. Na seção `/* USER CODE BEGIN 2 */`, faça a chamada das funções:

```
I2C_Init();
id = LSM6DSV32X_Init();
r=0; // Apenas para poder adicionar um breakpoint
```

Note que a variável “id” recebe o *byte* de identificação do sensor, que deve ser 0x70.

6. Agora vamos adicionar o código executado em *loop*. Abaixo da linha `/* USER CODE BEGIN 3 */`, escreva o código:

```
// Espera o "Data ready" (Bit 0 de INT_STATUS em 1)
r = 0x00;
while(!r) {
    r = LSM6DSV32X_ReadStatus();
    r &= 0x03;
}
// Lê os valores de aceleração e de velocidade angular
LSM6DSV32X_ReadAccel(buf);
ax = buf[0];
ay = buf[1];
az = buf[2];
LSM6DSV32X_ReadGyro(buf);
gx = buf[0];
gy = buf[1];
gz = buf[2];
```

7. Faça o *Build*. Conecte o módulo LSM6DSV32X no *header* H4 (I2C1). Há uma marcação indicando o pino 1, que é o mais à esquerda no *header* (alinhado com o pino 1 do *header*). Este é o pino onde nenhum fio está conectado. Adicione um *breakpoint* na linha “`r=0;`”. Transfira o código executável para o microcontrolador no modo *Debug*, e inicie (“Resume”) o programa. Ao pausar, observe o valor da variável “id”, que deve conter o identificador do sensor retornado pela função `LSM6DSV32X_Init()`.

8. Dando continuidade, utilizaremos o *Live Expressions* do Eclipse para visualizar em tempo real as atualizações do valor de expressões correspondentes a variáveis globais. Na aba *Live Expressions*, adicione as variáveis globais ax, ay, az, gx, gy e gz. Ajuste a largura das colunas para uma visualização clara.

Expression	Type	Value
(x)= ax	int16_t	0
(x)= ay	int16_t	0
(x)= az	int16_t	0
(x)= gx	int16_t	0
(x)= gy	int16_t	0
(x)= gz	int16_t	0
<a href="#">+ Add new expression</a>		

9. Remova o *breakpoint* do item 7 e faça o *Resume*. Veja as variáveis sendo atualizadas continuamente. Movimente o sensor e veja os valores de “a” e “g” mudarem.

Expression	Type	Value
(x)= ax	int16_t	6278
(x)= ay	int16_t	-11574
(x)= az	int16_t	-9908
(x)= gx	int16_t	-486
(x)= gy	int16_t	-279
(x)= gz	int16_t	71

10. Os valores amostrados pelo acelerômetro são representados em complemento de 2, utilizando 16 bits, com um [fundo de escala de \[-4g, 4g\] \(CTRL8\\_FS\\_XL=0b00\)](#). Considerando os valores de sensibilidade linear de aceleração fornecidos pelo fabricante no [datasheet](#), quais são os fatores de g correspondentes aos valores mostrados em ax, ay e az no item 9?

11. Conecte o analisador lógico como no item 7 da EEPROM e visualize as leituras periódicas dos valores amostrados.

12. Tanto a comunicação entre uma IMU e um microcontrolador quanto a entre uma memória EEPROM e o microcontrolador são realizadas via protocolo I2C, o que confere a elas uma estrutura de transferência muito similar. Apesar de os dados enviados serem completamente diferentes, a inicialização e a sequência de sinais do I2C (como START, envio de endereço do dispositivo, e STOP) são as mesmas. Vamos analisar a função de leitura do LSM6DSV32X.

```
uint8_t Reg_Read(uint8_t end, uint8_t * d) {
```

```

    uint8_t i;
    i = I2C1->RXDR;
    I2C1->CR2 |= LSM6DSV32X_I2C_ADDRESS | (1 << I2C_CR2_NBYTES_Pos) |
        I2C_CR2_START; // Escrita de 1 byte sem STOP
    while(!(I2C1->ISR & I2C_ISR_TXIS) && !(I2C1->ISR & I2C_ISR_NACKF)) {}
    if(I2C1->ISR & I2C_ISR_NACKF) {
        I2C1->ICR = I2C_ICR_STOPCF;
        I2C1->CR2 = 0;
        return 1;
    }
    I2C1->TXDR = end; // Envia o endereço
    while(!(I2C1->ISR & I2C_ISR_TC)) {}
    I2C1->CR2 |= LSM6DSV32X_I2C_ADDRESS | (1 << I2C_CR2_NBYTES_Pos) |
        I2C_CR2_RD_WRN | I2C_CR2_START;
    while(!(I2C1->ISR & I2C_ISR_RXNE) && !(I2C1->ISR & I2C_ISR_NACKF)) {}
    if(I2C1->ISR & I2C_ISR_NACKF) {
        I2C1->ICR = I2C_ICR_STOPCF;
        I2C1->CR2 = 0;
        return 1;
    }
    *d = I2C1->RXDR;
    I2C1->CR2 = I2C_CR2_STOP; // Realiz ao STOP manualmente
    while(!(I2C1->ISR & I2C_ISR_STOPF)) {}
    I2C1->ICR = I2C_ICR_STOPCF;
    I2C1->CR2 = 0;
    return 0;
}

```

No arquivo `LSM6DSV32X.h`, notamos que o `LSM6DSV32X_I2C_ADDRESS` é definido como `0xD4`, embora o endereço do dispositivo seja `0x6A`. Considerando que o endereço de um periférico I2C é de *7 bits*, e que um *bit* de controle R\W para indicar se a operação é de leitura (R) ou escrita (\W) é adicionado como o *bit* menos significativo para formar o *byte* completo a ser transmitido, como você explica essa diferença?

Notou que o endereço do dispositivo escravo e, principalmente, o endereço interno do dispositivo é tratado de forma distinta pelo LSM6DSV32X (endereço interno de *8 bits*) e pela memória EEPROM (endereço interno é maior que *8 bits*). Considerando essa nuance, quais são as principais diferenças no esquema de endereçamento ao se comunicar com uma IMU versus uma EEPROM através do I2C?

## FUNDAMENTOS TEÓRICOS

Sensores e atuadores são a base da interação entre sistemas eletrônicos e o ambiente físico. Em transdutores mais avançados, que possuem diferentes níveis de capacidade de processamento, muitas vezes chamados de diferentes níveis de “inteligência”, a eficiência dessa interação é significativamente amplificada. Isso acontece graças à integração de recursos de processamento e comunicação diretamente no próprio dispositivo. Esta seção

detalha os princípios operacionais de sensores, que são essenciais para captar informações, e de atuadores, que permitem respostas controladas do sistema. Além disso, exploraremos dois protocolos de comunicação serial amplamente utilizados: I2C (Inter-Integrated Circuit) e SPI (Serial Peripheral Interface). Ambos são fundamentais para a troca de dados entre o transdutor “inteligente” e os demais componentes do sistema, facilitando o controle de operações mais complexas.

## SENSORES E ATUADORES

Os **sensores** são capazes de medir uma ampla gama de parâmetros físicos, como temperatura, pressão, luz, som e muito mais. Eles traduzem o mundo físico em dados digitais, fornecendo informações para que os microcontroladores tomem decisões. Por outro lado, os **atuadores** são capazes de executar ações com base nas decisões programadas nos microcontroladores. Eles podem girar um motor, acionar uma válvula, mover um braço de robô ou alterar a intensidade da luz emitida por um LED, como realizar outras tarefas. Os atuadores transformam as decisões tomadas por um sistema microcontrolado em ações, tornando as máquinas capazes de interagir com o ambiente.

Em sistemas eletrônicos, os sensores normalmente geram uma grandeza elétrica, como resistência, capacidade, tensão ou corrente, proporcional à quantidade física que se deseja medir. Quando a saída do sensor não está diretamente em forma de tensão, ela geralmente é convertida por meio de circuitos condicionadores. Essa tensão é então transformada em uma representação numérica digital por meio de um conversor analógico-digital (ADC), conforme discutido no Roteiro 9. Sensores que possuem um ADC integrado e que fornecem saídas digitais são chamados de **sensores digitais**. Já aqueles que fornecem uma saída analógica contínua são classificados como **sensores analógicos**.

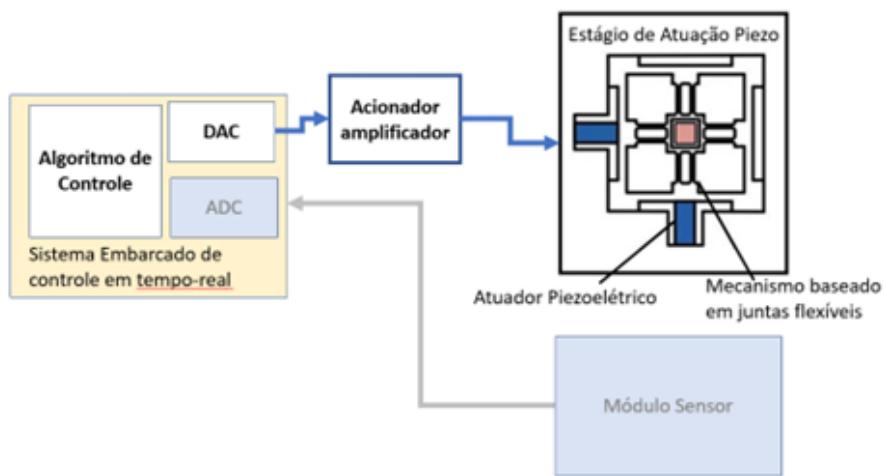
Por outro lado, os atuadores são dispositivos que realizam ações físicas em resposta a sinais de controle. Eles podem ser acionados por sinais analógicos (como tensão ou corrente) gerados por um conversor digital-analógico (DAC), também abordado no Roteiro 9. Além disso, muitos atuadores modernos são controlados por sinais digitais, como modulação por largura de pulso (PWM) ou comandos enviados por protocolos digitais. Atuadores que recebem diretamente comandos digitais e integram circuitos de controle são geralmente classificados como **atuadores digitais**, independentemente de possuírem internamente um DAC. Já os **atuadores analógicos** recebem sinais contínuos que variam dentro de um intervalo específico.

Nos últimos anos, surgiram no mercado *chips* altamente integrados que combinam sensores digitais ou atuadores digitais com microcontroladores embarcados, formando módulos compactos e inteligentes. Esses dispositivos são amplamente utilizados em sistemas embarcados e aplicações de Internet das Coisas (em inglês, *Internet of Things* – IoT), devido à sua eficiência, versatilidade e baixo consumo de energia. Com os avanços nas **tecnologias de materiais e microfabricação**, tornou-se possível medir uma ampla variedade de parâmetros físicos, como aceleração, pressão, temperatura e rotação, por meio de dispositivos miniaturizados. Também é possível gerar vibrações ou alterar a posição de componentes

mecânicos em escala microscópica. Essa inovação tecnológica impulsionou o desenvolvimento dos chamados **Sistemas Microeletromecânicos**, ou **MEMS** (do inglês *Micro-Electro-Mechanical Systems*).

Os dispositivos MEMS integram componentes mecânicos e elétricos em um único *chip*, permitindo a criação de sensores e atuadores extremamente compactos, com alta precisão, baixo custo e consumo de energia reduzido. Quando combinados com microcontroladores ou circuitos de processamento de sinais, esses sistemas dão origem aos chamados **sensores e atuadores inteligentes**. Além disso, quando conectados a interfaces de comunicação digital (como UART, I2C, SPI, CAN ou redes sem fio), esses módulos inteligentes podem ser integrados em **redes de sensores e atuadores**. Quando interligadas à internet, essas redes formam a base da IoT, possibilitando o monitoramento e controle remoto em tempo real de dispositivos físicos distribuídos em ambientes industriais, residenciais, urbanos e ambientais.

A figura a seguir representa um cenário típico de um sistema embarcado. Nesse contexto, o microcontrolador atua como unidade central de processamento, interpretando os sinais provenientes de um sensor e, com base nesses dados, gerando os sinais de controle apropriados para acionar um atuador. Como os atuadores geralmente operam com níveis de tensão e corrente mais elevados do que os sinais que o microcontrolador pode fornecer, é essencial o uso de **circuitos de acionamento**, também conhecidos como **drivers**, para realizar essa interface. Esses circuitos garantem que os sinais digitais do microcontrolador sejam amplificados ou adaptados de forma adequada para acionar o atuador com segurança e eficiência. Nesse processo, cabe ao projetista selecionar cuidadosamente os componentes eletrônicos complementares.



Além da distinção entre sensores analógicos e digitais, e da evolução tecnológica para módulos inteligentes e redes de sensores e atuadores, é igualmente essencial compreender a **classificação dos transdutores com base nos parâmetros físicos** que medem e nos **princípios físicos** que utilizam para operar. Essa categorização permite ao projetista selecionar o tipo de sensor ou atuador mais adequado a cada aplicação, considerando fatores como faixa de operação, sensibilidade, tempo de resposta, linearidade e condições ambientais. Por exemplo, sensores de temperatura podem operar com base em termorresistência, efeito

termoelétrico ou semicondutores, cada qual com características específicas quanto à precisão, estabilidade e tempo de resposta. Da mesma forma, sensores de pressão podem utilizar princípios piezoresistivos, capacitivos ou piezoelétricos, variando em robustez, custo e sensibilidade.

Compreender os princípios de transdução é fundamental para garantir a precisão, robustez e compatibilidade do sistema, especialmente em aplicações críticas nos setores industrial, médico ou automotivo, onde uma escolha inadequada pode comprometer o desempenho, a confiabilidade e até mesmo a segurança da aplicação. Por exemplo, usar um sensor de distância baseado em reflexões de ondas ultra-sônicas pode ser desastroso num ambiente com muito ruído acústico ou onde existam múltiplos objetos que geram reflexões confusas. Além disso, o conhecimento dos princípios físicos envolvidos também é essencial para o projeto dos circuitos de interface entre os sensores e os microcontroladores. Muitos sensores exigem **fontes de alimentação** externas, além de condicionamento de sinal ou amplificação, a fim de tornar detectáveis as variações nas grandezas físicas de interesse. De modo semelhante, a maioria dos atuadores depende de **fontes de energia** para fornecer a potência necessária às ações mecânicas ou eletromagnéticas que executam no ambiente físico.

## **Classificação de sensores quanto às grandezas físicas medidas**

Do ponto de vista das **grandezas físicas mensuráveis**, existe uma ampla variedade de sensores. A capacidade de transduzir variações dessas grandezas em sinais elétricos facilita a integração dos sensores a sistemas embarcados, possibilitando a implementação de soluções automatizadas e a coleta de dados em tempo real. Essa versatilidade e precisão tornam os sensores componentes fundamentais em diversas aplicações tecnológicas e de automação. Entre esses sensores, destacam-se:

- **Sensores de Entrada Pontual:** Este dispositivo é projetado para detectar a localização de um objeto, como a pressão de um botão ou a ativação de uma tecla em um teclado, convertendo essa informação em um sinal elétrico. A transdução ocorre por meio de diferentes princípios físicos, como a variação de resistência (em botões de pressão) ou a mudança na capacidade (em teclados de membrana). Nos **botões de pressão**, a ação de pressionar o botão provoca uma alteração na resistência elétrica, que é medida e convertida em um sinal digital, indicando que a posição “ativada” foi alcançada. Já os teclados de membrana, apresentados no Roteiro 8, podem funcionar com base na detecção da capacidade entre camadas condutoras; ao pressionar uma tecla, as camadas se conectam, gerando uma alteração na capacidade que pode ser interpretada como um evento. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, incluindo interfaces de usuário em dispositivos eletrônicos, painéis de controle e equipamentos de automação. Por exemplo, em painéis de controle de eletrodomésticos, botões de pressão permitem ao usuário selecionar funções, enquanto teclados de membrana são comuns em dispositivos portáteis, como calculadoras e controles remotos.

- **Sensores de Temperatura:** Esse dispositivo é responsável por medir a temperatura de um ambiente ou objeto e converte essa informação em um sinal elétrico, permitindo o monitoramento e o controle de processos térmicos. A transdução ocorre quando o sensor detecta mudanças de temperatura e utiliza efeitos físicos, como a variação da resistência elétrica (em termistores), o efeito termoelétrico (em termopares) ou a alteração na tensão gerada (em sensores de temperatura baseados em diodos). No caso dos **termistores**, a resistência do material varia com a temperatura, permitindo que pequenas mudanças de temperatura sejam convertidas em mudanças detectáveis na resistência elétrica. Já os **termopares** operam com base no [efeito Seebeck](#), onde dois metais diferentes, quando unidos, geram uma tensão elétrica proporcional à diferença de temperatura entre as junções. Os **sensores de temperatura baseados em diodos** aproveitam a relação entre a corrente elétrica e a temperatura para fornecer medições precisas. Esses sensores são amplamente utilizados em sistemas embarcados, como em dispositivos de controle de climatização, sistemas de automação residencial e aplicações industriais. Por exemplo, em aquecedores e sistemas de ar condicionado, sensores de temperatura monitoram continuamente as condições ambientais e ajustam a configuração do sistema para manter a temperatura desejada. Em ambientes médicos, sensores de temperatura são usados para o monitoramento de pacientes e o controle de equipamentos, garantindo a segurança e eficácia dos tratamentos.
- **Sensores de Pressão:** Este dispositivo é projetado para medir a pressão de gases ou líquidos e converte essa informação em um sinal elétrico, permitindo o monitoramento e o controle de processos industriais e ambientais. A transdução ocorre quando o sensor detecta variações na pressão e utiliza efeitos físicos, como a variação de resistência elétrica (em sensores piezoresistivos), a deformação de um diafragma (em sensores piezoelétricos) ou a variação de capacitância (em sensores capacitivos). Nos **sensores piezoresistivos**, a pressão aplicada causa uma mudança na resistência elétrica de um material semicondutor, que pode ser medida e convertida em um sinal elétrico. Os **sensores piezoelétricos** aproveitam o efeito piezoelétrico, onde a pressão aplicada a um material piezoelétrico gera uma tensão elétrica proporcional à força aplicada. Já os **sensores capacitivos** operam com base na variação da capacitância entre placas metálicas, cuja distância muda em resposta à pressão aplicada. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, como monitoramento de pressão em sistemas hidráulicos, controle de pressão em tanques de combustível, e medições em processos industriais. Por exemplo, em aplicações automotivas, sensores de pressão monitoram a pressão dos pneus para garantir a segurança e eficiência do veículo. Em sistemas de HVAC (do inglês *Heating, Ventilation, and Air Conditioning*), eles ajudam a regular a pressão do ar, otimizando o desempenho do sistema.
- **Sensores de Deslocamento:** Este dispositivo é projetado para medir a posição ou a movimentação de um objeto e converte essa informação em um sinal elétrico, permitindo o monitoramento e o controle de sistemas mecânicos e automatizados. A transdução ocorre quando o sensor detecta alterações na posição e utiliza efeitos

físicos, como a variação de resistência elétrica (em potenciômetros), a variação de capacidade (em sensores capacitivos), o efeito piezoelettrico ou a variação magnética ou óptica. Nos **potenciômetros**, o deslocamento do eixo gera uma mudança na resistência, que pode ser medida e convertida em um sinal elétrico proporcional à posição. Os **sensores capacitivos** operam com base na variação da capacidade entre placas metálicas, cuja distância muda conforme o deslocamento do objeto. Já os **sensores piezoelettricos** aproveitam o efeito piezoelettrico, onde a deformação mecânica causada pelo deslocamento gera uma tensão elétrica. Os **encoders lineares**, por sua vez, são utilizados para medir deslocamentos em linha reta, convertendo essa informação em sinais elétricos, conforme explicado no Roteiro 6. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, incluindo dispositivos de entrada, como *joysticks*, conforme apresentado no Roteiro 9. Eles são amplamente aplicados no controle de movimento em robótica, monitoramento de posição em atuadores e medição de deslocamentos em máquinas industriais. Por exemplo, em aplicações automotivas, sensores de deslocamento podem ser utilizados para monitorar a posição de componentes como aceleradores e freios, garantindo uma resposta precisa e segura.

- **Sensores de Vazão:** Este dispositivo é projetado para medir a quantidade de fluido que passa por um ponto específico em um determinado intervalo de tempo, convertendo essa informação em um sinal elétrico. A transdução ocorre quando o sensor detecta características do fluxo, como a velocidade ou o volume do fluido, e converte essas grandezas físicas em sinais elétricos. São aplicados princípios físicos, como a variação de pressão diferencial, resistência térmica, frequência de rotação de turbinas, ou até efeitos capacitivos, magnéticos ou ultrassônicos, dependendo da tecnologia adotada. Os **sensores de vazão por pressão diferencial** medem a diferença de pressão entre dois pontos de um tubo, permitindo calcular a vazão com base na lei de Bernoulli. Os **sensores capacitivos** utilizam a variação da capacidade causada pelo movimento do fluido para determinar a vazão. Já os **sensores de vazão eletromagnéticos** aproveitam o princípio de indução eletromagnética, onde o fluido condutor gera uma tensão proporcional à sua velocidade quando atravessa um campo magnético. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, como monitoramento de sistemas hidráulicos e pneumáticos, controle de processos industriais e medição de consumo em redes de abastecimento de água. Por exemplo, em aplicações agrícolas, sensores de vazão são essenciais para otimizar a irrigação, garantindo a eficiência do consumo de água.
- **Sensores de Movimento:** Este dispositivo é projetado para detectar a presença ou o deslocamento de objetos em uma determinada área e converte essa informação em um sinal elétrico. A transdução ocorre quando o sensor converte uma grandeza física do ambiente, como radiação infravermelha, distância ou proximidade, em um sinal elétrico que pode ser interpretado por circuitos eletrônicos. Os **sensores de movimento infravermelhos** detectam mudanças na radiação infravermelha emitida por objetos quentes, como seres humanos, ativando um sinal elétrico quando detectam

movimento. Os **sensores ultrassônicos** emitem ondas sonoras de alta frequência e medem o tempo que leva para essas ondas retornarem após refletirem em um objeto, permitindo calcular a distância e detectar movimento. Já os **sensores capacitivos** detectam mudanças na capacidade provocadas pelo movimento de objetos próximos. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, como sistemas de segurança, automação residencial e controle de iluminação. Por exemplo, em aplicações de segurança, sensores de movimento podem ativar alarmes ou câmeras de vigilância quando detectam movimento em áreas restritas.

- **Sensores de Nível:** Este dispositivo é projetado para medir a altura ou a quantidade de um líquido ou sólido em um recipiente, convertendo essa informação em um sinal elétrico. A transdução ocorre por meio da conversão da variação no nível de um fluido ou material em um sinal elétrico, utilizando diferentes princípios físicos, como flutuação mecânica, variação de capacidade, pressão hidrostática ou reflexão de ondas ultrassônicas. Os **sensores de nível por flutuador** utilizam um dispositivo flutuante que se desloca na superfície do líquido; quando a altura do líquido varia, o flutuador move-se, ativando um mecanismo que gera um sinal elétrico. Os **sensores capacitivos** medem a variação da capacidade entre duas placas, que muda conforme o nível do líquido altera a distância entre elas. Os **sensores de pressão** medem a pressão exercida pela coluna de líquido, convertendo essa pressão em um sinal correspondente ao nível do líquido. Já os **sensores ultrassônicos** emitem ondas sonoras e medem o tempo que leva para elas refletirem na superfície do líquido, permitindo calcular o nível com precisão. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, como monitoramento de reservatórios, controle de processos industriais e automação de sistemas de irrigação. Por exemplo, em aplicações de abastecimento de água, sensores de nível podem ser utilizados para garantir que tanques não transbordem ou fiquem vazios, ativando bombas conforme necessário.
- **Sensores de Luminosidade:** Este dispositivo é projetado para medir a intensidade da luz em um ambiente, convertendo essa informação em um sinal elétrico. A transdução em sensores de luminosidade ocorre com base em princípios fotoelétricos, nos quais a luz incidente sobre um material sensível provoca a geração de uma corrente elétrica ou altera sua resistência elétrica. Os **sensores fotográficos**, como os fotodiodos e fototransistores, utilizam materiais semicondutores que produzem uma corrente elétrica proporcional à intensidade da luz que os atinge. Quando a luz incide sobre esses sensores, a energia da luz excita os elétrons, resultando em uma corrente elétrica que pode ser medida. Já os **sensores de resistência dependente de luz** (em inglês, *Light Dependent Resistor – LDR*) variam sua resistência elétrica com base na quantidade de luz que recebem; em ambientes mais iluminados, sua resistência diminui, permitindo a passagem de mais corrente. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, como controle de iluminação automática, monitoramento ambiental e dispositivos de segurança. Por

exemplo, em sistemas de iluminação inteligente, sensores de luminosidade podem ajustar a intensidade das luzes com base na quantidade de luz natural disponível, economizando energia e melhorando a eficiência.

- **Sensores de Rotação:** Este dispositivo é projetado para medir a posição ou a velocidade angular de um objeto em movimento, convertendo essa informação em um sinal elétrico. Para medir a posição angular, a transdução ocorre por meio de diferentes princípios físicos, como o efeito Hall, que detecta variações em campos magnéticos; a variação de resistência em elementos sensíveis; ou a tecnologia óptica, que utiliza interrupções de feixes de luz para determinar a posição ou a velocidade angular de um eixo. Os **sensores de efeito Hall** utilizam um campo magnético para detectar a rotação. Quando um ímã é posicionado próximo ao sensor, a variação no campo magnético provoca uma alteração na tensão de saída, que pode ser interpretada como uma medida da rotação. Já os **encoders rotativos**, apresentados no Roteiro 6, que podem ser ópticos ou magnéticos, funcionam registrando a passagem de marcas em um disco rotativo, gerando pulsos elétricos que representam a posição angular e a velocidade de rotação. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, como controle de motores, robótica e automação industrial. Por exemplo, em sistemas de controle de movimento, sensores de rotação podem fornecer *feedback* preciso sobre a posição de um eixo, permitindo ajustes em tempo real para garantir a precisão do movimento.

A medição da velocidade angular é comumente realizada por **giroscópios**, cuja transdução também se baseia em diferentes fenômenos físicos. Nos **giroscópios MEMS**, o princípio mais comum é a detecção do efeito Coriolis sobre uma massa vibratória, sendo a força resultante convertida em sinal elétrico por meio de variações de capacidade ou piezoresistência. Em contrapartida, **giroscópios ópticos** utilizam o efeito Sagnac, que mede a diferença de fase entre feixes de luz propagando-se em direções opostas dentro de um caminho fechado, permitindo determinar a taxa de rotação com alta precisão.

- **Sensores de Aceleração:** Este dispositivo é projetado para medir a aceleração de um objeto em uma ou mais direções, convertendo essa informação em um sinal elétrico. A transdução ocorre quando o sensor detecta variações na força aplicada, utilizando efeitos físicos, como a variação de capacidade (em acelerômetros capacitivos) ou a variação de resistência (em acelerômetros piezoelétricos). A transdução ocorre por meio da detecção de variações na força resultante de movimentos acelerados gerados por diferentes efeitos físicos, como a variação de capacidade ou a variação de resistência elétrica gerada por deformações mecânicas. Nos **acelerômetros capacitivos**, a aceleração é medida por meio da mudança na capacidade entre placas que se deslocam conforme a força inercial, incluindo a aceleração gravitacional. Já os **acelerômetros piezoelétricos** aproveitam o efeito piezoelétrico, onde a deformação causada pela aceleração gera uma tensão elétrica proporcional à força aplicada. Os principais acelerômetros comerciais utilizam a aceleração da gravidade como

referência, o que permite a medição da orientação do sensor em relação ao vetor gravitacional. Isso é particularmente útil em aplicações onde a detecção da inclinação ou a orientação em relação ao solo é crítica, como em dispositivos móveis e sistemas de navegação. Essa referência também melhora a precisão das medições de movimento, uma vez que a gravidade fornece um ponto de partida consistente para as leituras. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, incluindo dispositivos móveis para detecção de movimento, sistemas de navegação em veículos e monitoramento de condições de funcionamento em equipamentos industriais. Por exemplo, em *smartphones*, os sensores de aceleração são usados para detectar a orientação do dispositivo e ajustar a tela automaticamente. Em aplicações automotivas, podem ser utilizados para monitorar a dinâmica do veículo, contribuindo para a segurança e o desempenho.

- **Sensores de Orientação Espacial:** Este dispositivo é projetado para determinar a posição e o movimento do corpo em relação a um sistema de coordenadas tridimensionais (espaço), convertendo essa informação em um sinal elétrico. A transdução em sensores de orientação espacial ocorre pela conversão de grandezas físicas, como aceleração e velocidade angular, em sinais elétricos, por meio de dispositivos como acelerômetros e giroscópios. Os **giroscópios** medem a taxa de rotação em torno de um ou mais eixos, enquanto os **acelerômetros** detectam a aceleração linear em múltiplos de gravidade ( $g$ ). A combinação permite que o sistema compense a gravidade, resultando em uma orientação mais exata, mesmo durante movimentos dinâmicos. Isso é fundamental em aplicações que exigem alta precisão, como em dispositivos móveis, sistemas de navegação e controle de estabilidade em veículos. Em *smartphones*, por exemplo, sensores de orientação espacial ajudam a ajustar a tela automaticamente e a detectar a posição em jogos de realidade aumentada. Em drones e robôs, eles são essenciais para a navegação e controle de movimento, garantindo que o dispositivo mantenha sua trajetória desejada e reaja adequadamente a mudanças de posição e orientação.
- **Sensores de Campo Magnético:** Este dispositivo é projetado para medir a intensidade e a direção de um campo magnético, convertendo essas grandezas físicas em sinais elétricos. A transdução ocorre por meio da detecção de variações no campo magnético, utilizando efeitos físicos como o efeito Hall, que gera uma tensão elétrica proporcional à densidade do fluxo magnético aplicado. Outros princípios, como a magnetorresistência ou o efeito magnetoindutivo, também podem ser empregados, dependendo da aplicação e da precisão desejada. Esses sensores são fundamentais em diversas aplicações, como navegação, posicionamento e detecção de objetos metálicos. Diferentemente dos sensores de aceleração e giroscópios, que medem movimentos e inclinações, os **magnetômetros** fornecem uma referência direcional fixa ao detectar a presença e a intensidade do campo magnético terrestre (norte magnético), sendo essenciais para determinar a orientação absoluta de dispositivos. Nos *smartphones*, sensores magnéticos são usados para orientar a bússola digital, melhorar a precisão em aplicativos de navegação e compensar movimentos

combinando dados com acelerômetros e giroscópios. Em aplicações automotivas, sensores de campo magnético são empregados em sistemas de posicionamento de rodas, controle de estabilidade e sensores de rotação. Em ambientes industriais e de Internet das Coisas (IoT), esses sensores servem para detectar metais, monitorar equipamentos e segurança de sistemas automatizados.

- **Sensores de Distância:** Este dispositivo é projetado para medir a distância entre o sensor e um objeto, convertendo essa informação em um sinal elétrico. A transdução ocorre por meio de diferentes princípios de operação, como ultrassom, infravermelho ou laser. Nos **sensores ultrassônicos**, uma onda sonora é emitida e, ao ser refletida pelo objeto, o tempo que leva para retornar é medido, permitindo calcular a distância com base na velocidade do som. Já os **sensores de infravermelho** utilizam luz infravermelha para medir a distância, geralmente através da [triangulação](#) ou do tempo de voo. **Sensores a laser**, por sua vez, emitem um feixe de luz que é refletido pelo objeto e mede o tempo que leva para retornar, proporcionando alta precisão e capacidade de medição em longas distâncias. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, incluindo robótica, automação industrial e sistemas de navegação. Por exemplo, em robôs, sensores de distância ajudam na detecção de obstáculos e na navegação autônoma, permitindo que o robô mapeie seu ambiente e evite colisões. Em aplicações de automação, como em linhas de produção, esses sensores podem ser utilizados para medir a posição de objetos e otimizar processos. Além disso, em sistemas de segurança, sensores de distância podem ser empregados para monitorar áreas e detectar intrusões. A versatilidade e a precisão na medição tornam os sensores de distância componentes valiosos em diversas tecnologias modernas.
- **Sensores de Corrente:** Este dispositivo é projetado para medir a intensidade da corrente elétrica que flui em um circuito, convertendo essa informação em um sinal elétrico. A transdução ocorre por meio de diferentes princípios de operação, como o efeito Hall, transformadores de corrente ou resistência shunt. No **caso do efeito Hall**, um campo magnético gerado pela corrente elétrica cria uma tensão que é proporcional à intensidade da corrente, permitindo a medição sem interrupção do circuito. **Transformadores de corrente** operam com base na indução eletromagnética, onde a corrente primária gera um campo magnético que induz uma corrente na bobina secundária, proporcional à corrente original. Já os **sensores de resistência shunt** utilizam uma resistência de precisão inserida no circuito, onde a tensão gerada pela passagem da corrente é medida e convertida em um valor de corrente. Esses sensores são amplamente utilizados em sistemas embarcados em diversas aplicações, incluindo monitoramento de consumo de energia, controle de motores e proteção de circuitos. Por exemplo, em sistemas de automação residencial, sensores de corrente ajudam a monitorar o consumo de energia de eletrodomésticos, permitindo a otimização do uso e a redução de custos. Em aplicações industriais, esses sensores são essenciais para o controle de motores, garantindo que funcionem dentro de limites seguros e eficientes. Além disso, em sistemas de proteção, sensores de corrente podem detectar sobrecargas

e curtos-circuitos, ativando mecanismos de segurança para evitar danos aos equipamentos. A precisão e a confiabilidade na medição da corrente elétrica fazem dos sensores de corrente componentes cruciais em diversas tecnologias modernas.

## Classificação de sensores quanto aos princípios de operação

A transdução de uma mesma grandeza física em um sinal elétrico pode ser realizada com base em diferentes efeitos físicos naturais. Isso significa que, dependendo do fenômeno físico explorado, os sensores podem ser classificados de diversas formas, segundo seu princípio de operação. Essa classificação permite que engenheiros e projetistas escolham a tecnologia mais adequada às exigências específicas de cada aplicação. A escolha do princípio de operação está diretamente ligada ao desempenho esperado do sensor em determinado contexto. Cada efeito físico apresenta características distintas, que influenciam fatores como precisão, sensibilidade, faixa de operação e resposta a condições ambientais adversas. Entre os principais princípios físicos utilizados na transdução, destacam-se:

- **Efeito Fotovoltaico:** O efeito fotovoltaico é o fenômeno pelo qual materiais semicondutores geram uma corrente elétrica quando expostos à luz. Ao absorver a radiação luminosa, os fótons excitam elétrons em um semicondutor, criando pares de elétron-lacuna que podem ser mobilizados para gerar eletricidade. Sensores baseados no efeito fotovoltaico são particularmente apropriados para aplicações em que a detecção de luz é essencial. Um exemplo são os sensores de luz, que são utilizados em sistemas de automação, como iluminação inteligente, onde a intensidade da luz é monitorada para ajustar o brilho das lâmpadas. Além disso, as células solares, embora não sejam sensores no sentido tradicional, empregam o efeito fotovoltaico para capturar luz e convertê-la em eletricidade, sendo amplamente utilizadas em sistemas de energia renovável. Outro uso importante é em sensores de radiação, que são aplicados em estações meteorológicas e equipamentos de pesquisa para medir a intensidade da radiação solar e monitorar condições ambientais. Por fim, os sensores de imagem em câmeras digitais também utilizam o efeito fotovoltaico para converter luz em sinais elétricos, permitindo a captura de fotografias.
- **Efeito Fotoelétrico (Efeito Hertz):** O efeito fotoelétrico é o fenômeno pelo qual elétrons são ejetados de um material quando este é iluminado por radiação eletromagnética, geralmente luz visível ou ultravioleta. Quando a energia dos fótons da luz incide sobre a superfície de um material, como um metal, ela pode ser suficiente para superar a função trabalho do material, liberando elétrons e gerando uma corrente elétrica. Sensores baseados no efeito fotoelétrico são adequados para várias aplicações. Um exemplo são as células fotográficas, que são usadas em dispositivos como fotômetros e sensores de luminosidade, detectando a intensidade da luz e ajustando sistemas de iluminação, como em painéis solares. Outro uso é em sensores de presença, empregados em sistemas de segurança e automação, onde a detecção de movimento ocorre através da interrupção de um feixe de luz, acionando alarmes ou sistemas de iluminação. Além disso, leitores de códigos de barras utilizam

o efeito fotoelétrico para detectar a reflexão da luz em diferentes intensidades ao passar sobre as barras do código, permitindo a leitura e interpretação de dados.

- **Efeito Piezoelétrico:** O efeito piezoelétrico é um fenômeno em que certos materiais, como cristais e cerâmicas piezoelétricas, geram uma tensão elétrica quando submetidos a pressão mecânica. Essa propriedade é resultado do rearranjo das cargas elétricas em sua estrutura cristalina, que ocorre sob estresse mecânico. Sensores baseados no efeito piezoelétrico são adequados para várias aplicações, incluindo sensores de pressão, que são utilizados em sistemas de monitoramento de pressão em diversas indústrias, permitindo medições precisas em ambientes variáveis. Os microfones piezoelétricos são amplamente usados em dispositivos de áudio, pois convertem ondas sonoras em sinais elétricos. Além disso, sensores piezoelétricos são empregados em impressoras 3D para controlar a movimentação e a deposição de material, garantindo precisão na impressão. Esses sensores também são aplicados em dispositivos de alerta e sinalização, onde a detecção de pressão ou vibração ativa um sinal de alerta, e em sistemas de controle de movimento, especialmente na robótica e automação, onde os atuadores piezoelétricos proporcionam movimentos precisos, essenciais em aplicações que exigem controle fino.
- **Efeito Resistivo:** O efeito resistivo é baseado na variação da resistência elétrica de um material em resposta a mudanças em fatores como pressão, temperatura ou deslocamento. Quando um material condutor ou semicondutor é submetido a uma força ou condição externa, sua resistência elétrica pode mudar, permitindo a conversão dessa variação em um sinal elétrico. Sensores que utilizam o efeito resistivo são amplamente empregados em diversas aplicações. Por exemplo, potenciômetros são utilizados em controles de volume e ajuste de intensidade, onde a posição do eixo altera a resistência e, consequentemente, o sinal elétrico. Sensores de pressão resistivos são usados em sistemas industriais para monitorar a pressão de fluidos, enquanto sensores de entrada pontual, como os utilizados em botões e teclados, detectam a ativação de comandos por meio da variação da resistência quando pressionados. Esse efeito é particularmente útil em aplicações onde medições precisas de força, posição ou pressão são necessárias, tornando os sensores resistivos componentes essenciais em sistemas de automação, controle e interface homem-máquina.
- **Efeito de Indução Mútua:** O efeito de indução mútua ocorre quando a corrente elétrica que flui através de um condutor cria um campo magnético, que, por sua vez, induz uma corrente em um segundo condutor próximo. Esse fenômeno é fundamental na operação de transformadores e em dispositivos de medição de corrente, onde a variação da corrente em um condutor influencia a corrente em outro, permitindo a detecção de mudanças elétricas. Sensores que utilizam o efeito de indução mútua são comumente empregados em diversas aplicações. Por exemplo, em transformadores, esse efeito é utilizado para transmitir energia elétrica entre circuitos, ajustando a tensão conforme necessário. Além disso, sensores de corrente que operam com

indução mútua são utilizados em sistemas de monitoramento de energia, permitindo medir a corrente em fios sem contato direto, o que proporciona segurança e precisão em aplicações industriais e residenciais. Também é possível encontrar esse efeito em transdutores de posição, onde a variação da posição de um objeto afeta o campo magnético, gerando sinais elétricos que podem ser medidos. Esses sensores são especialmente úteis em ambientes onde a medição não intrusiva é desejável, como em sistemas de automação industrial, medição de energia elétrica e monitoramento de equipamentos, garantindo eficiência e segurança.

- **Efeito Eletromagnético:** O efeito se refere à interação entre campos elétricos e magnéticos, resultando na geração de força eletromotriz (FEM) em condutores expostos a variações de campos magnéticos. Esse fenômeno é fundamental em dispositivos como geradores e motores elétricos, onde a energia elétrica é convertida em energia mecânica e vice-versa. Sensores que utilizam o efeito eletromagnético são amplamente aplicados em diversas áreas. Por exemplo, em sensores de proximidade, eles detectam a presença de objetos metálicos sem contato, sendo muito usados em sistemas de automação e controle industrial. Além disso, sensores de corrente baseados nesse efeito podem monitorar a intensidade da corrente elétrica em circuitos, sendo essenciais para sistemas de medição de energia. A principal diferença entre o efeito eletromagnético e o efeito de indução magnética é que, enquanto o efeito de indução magnética se concentra na indução de corrente em um condutor devido à variação de um campo magnético, o efeito eletromagnético engloba uma interação mais ampla entre campos elétricos e magnéticos que resulta em movimento ou geração de energia. Essa distinção é crucial para a aplicação adequada em diferentes tecnologias e dispositivos.
- **Efeito Hall:** O efeito é um fenômeno físico que ocorre quando um condutor ou semicondutor é exposto a um campo magnético perpendicular à direção da corrente elétrica que flui através dele. Esse efeito resulta na geração de uma tensão elétrica (tensão Hall) nas extremidades do material, perpendicular tanto à corrente quanto ao campo magnético aplicado. Essa tensão pode ser medida e utilizada para determinar a intensidade do campo magnético. Sensores baseados no efeito Hall são amplamente utilizados em várias aplicações, como em sistemas de detecção de corrente elétrica, onde permitem medir a intensidade da corrente sem necessidade de contato direto com o circuito. Também são utilizados em sensores de posição e velocidade, como em motores e acionamentos, além de serem comuns em dispositivos automotivos para detecção de posição de componentes como eixos e rotores. A diferença entre o efeito Hall e os efeitos de indução magnética e eletromagnético está em sua abordagem e aplicação. O efeito Hall é específico para a geração de uma tensão elétrica em resposta a um campo magnético aplicado a um condutor, enquanto o efeito de indução magnética se refere à geração de corrente em um circuito fechado devido à variação de um campo magnético. Já o efeito eletromagnético abrange uma interação mais ampla entre campos elétricos e magnéticos, sendo fundamental para a operação de motores e

geradores. Assim, o efeito Hall é particularmente útil para medições e detecções precisas em ambientes onde a interação magnética é necessária.

- **Efeito Capacitivo:** O efeito se refere à capacidade de um capacitor de armazenar carga elétrica, que varia com a distância entre as placas do capacitor ou com a área das placas. Em sensores capacitivos, essa variação é utilizada para detectar mudanças na presença ou na posição de objetos, bem como em medições de pressão e umidade. Quando um objeto se aproxima do sensor, a capacidade muda, permitindo que essa alteração seja convertida em um sinal elétrico. Sensores capacitivos são amplamente utilizados em diversas aplicações, como em telas sensíveis ao toque, detectores de proximidade, e medidores de nível de líquidos. Sua capacidade de funcionar sem contato físico os torna ideais para ambientes onde a higiene é importante, como em equipamentos médicos e em indústrias alimentícias. Além disso, são frequentemente empregados em sistemas automotivos e de automação industrial para detectar a presença ou movimento de objetos.
- **Efeitos Químicos:** O efeito se refere à interação entre substâncias químicas que resulta em mudanças físicas ou elétricas, frequentemente utilizadas em sensores para detectar a presença e a concentração de determinados compostos. Esses sensores funcionam com base em reações químicas que geram um sinal elétrico proporcional à quantidade de substância analisada. Sensores baseados no efeito químico são amplamente utilizados em aplicações como monitoramento da qualidade do ar, onde detectam poluentes e gases tóxicos, como dióxido de carbono e monóxido de carbono. Eles também são essenciais em análises químicas e bioquímicas, sendo utilizados em dispositivos de diagnóstico médico, como sensores de glicose, que monitoram os níveis de açúcar no sangue. Além disso, encontram aplicação em processos industriais para controle de reações químicas e em laboratórios para medições precisas de concentrações de substâncias. A capacidade de fornecer medições em tempo real torna esses sensores valiosos em diversas áreas, incluindo saúde, segurança e meio ambiente.

## Classificação de sensores quanto à fonte de energia

Baseada em sua fonte de energia, os sensores podem ser classificados em:

- **Sensores Ativos:** Requerem uma **fonte de energia externa** para operar. Eles injetam energia no ambiente (como luz, som ou campo magnético) e medem a resposta. Exemplos incluem sensores ultrassônicos (que emitem pulsos sonoros), sensores infravermelhos ativos (que emitem luz IR) e sensores de corrente baseados no efeito Hall. Embora consumam mais energia, geralmente oferecem maior precisão e capacidade de medição em ambientes variados.
- **Sensores Passivos:** **Não necessitam de energia externa** para funcionar; eles geram um sinal elétrico diretamente a partir da energia ou fenômeno que estão medindo. Exemplos clássicos são termopares (que geram tensão a partir de uma diferença de temperatura), fotodiódos (que convertem luz em corrente) e sensores de resistência

*shunt* (que geram uma queda de tensão). Sensores passivos são ideais para aplicações de baixa potência, pois não contribuem significativamente para o consumo geral do sistema.

## Classificação de atuadores quanto às grandezas físicas geradas

Atuadores são componentes essenciais em sistemas embarcados e automação, pois convertem sinais elétricos de controle em ações físicas, como movimento, força ou alteração de estado. Compreender a taxonomia dos atuadores permite que os desenvolvedores identifiquem rapidamente as opções disponíveis, considerando fatores como o tipo de transdução, a tecnologia empregada (elétrica, hidráulica, pneumática, etc.) e a fonte de alimentação. Essa organização simplifica a seleção do atuador mais adequado para cada aplicação.

Do ponto de vista do **tipo de transdução a partir de energia elétrica**, existe uma diversidade de transduções que os materiais naturais podem realizar, permitindo uma ampla gama de aplicações tecnológicas. Aqui estão algumas delas mais conhecidas:

- **Efeito de Campo Elétrico:** Este fenômeno ocorre quando a aplicação de um campo elétrico em certos cristais e materiais cerâmico, denominados piezoelétricos, resulta em deformação mecânica. A interação entre o campo elétrico e a estrutura interna do material provoca uma mudança em sua forma ou volume. Esse efeito é amplamente utilizado em atuadores e sensores, onde a precisão na movimentação ou na detecção de variações é crucial. Por meio do controle do campo elétrico aplicado, é possível ajustar com precisão a deformação do material, tornando-o adequado para aplicações em dispositivos que requerem movimentação controlada, como em sistemas de microeletrônica e tecnologia médica. Além disso, os materiais piezoelétricos têm a capacidade de gerar uma tensão elétrica quando submetidos a pressão mecânica, fenômeno conhecido como **efeito piezoelétrico**, que é utilizado em dispositivos como buzzers.
- **Efeito Eletroluminescente:** Este fenômeno consiste na emissão de radiações visíveis ou invisíveis em resposta a uma corrente elétrica ou a um forte campo elétrico. Ele é o inverso do **efeito fotoelétrico**, onde a luz incidente em um material gera uma corrente elétrica, sendo esse o princípio fundamental das células solares e fotodetectores. Os materiais que exibem o efeito eletroluminescente, como semicondutores, convertem a energia elétrica em luz. Um exemplo prático desse efeito é o LED (do inglês *Light Emitting Diode*), que transforma a energia elétrica em luz visível de maneira eficiente e é amplamente utilizado em aplicações que vão desde iluminação até *displays* eletrônicos. O controle preciso da corrente elétrica permite a modulação da intensidade da luz, tornando os LEDs versáteis para uma variedade de usos em tecnologia e comunicação.
- **Efeito Joule (ou Efeito Termoelétrico):** Este fenômeno estabelece a relação entre a produção de calor e a corrente elétrica que percorre um condutor ao longo do tempo. Quando uma corrente elétrica flui através de um resistor, parte da energia elétrica é

convertida em calor devido à resistência do material. Um exemplo comum é o uso de resistores feitos de ligas metálicas de alta resistividade, como a liga de nicromo, que contém de 15% a 25% de cromo e de 19% a 80% de níquel, com o restante composto por ferro. Esses materiais conseguem transformar quase integralmente a energia elétrica em calor, tornando-os ideais para aplicações de aquecimento.

- **Efeito de Arco Elétrico:** Este fenômeno ocorre quando uma corrente elétrica passa através de um gás, gerando arcos elétricos que emitem radiações visíveis, ou luz. Esse efeito é frequentemente observado em lâmpadas fluorescentes, onde a corrente elétrica ioniza o gás dentro do tubo, criando um arco elétrico que produz luz. A emissão luminosa resulta da excitação dos átomos do gás, que liberam energia na forma de radiação visível quando retornam ao seu estado fundamental. Esse efeito é utilizado em diversas aplicações, incluindo iluminação e soldagem, devido à sua capacidade de gerar intensas fontes de luz. Note que a luz é uma parte essencial do processo de soldagem, pois contribui tanto para o aquecimento necessário quanto para a limpeza da superfície ou a adição de materiais de enchimento, melhorando a qualidade da solda em alguns processos.
- **Efeito Eletromagnético:** Este fenômeno se refere à interação entre campos elétricos e magnéticos, onde a variação de um campo pode induzir o outro. Esse princípio é fundamental em uma variedade de dispositivos, como motores elétricos e geradores, onde a movimentação de cargas elétricas em um campo magnético gera força motriz ou, inversamente, a mudança de um campo elétrico pode criar um campo magnético. O efeito eletromagnético é utilizado em tecnologias como transformadores, onde a energia é transferida entre circuitos elétricos através de campos magnéticos, e em dispositivos de comunicação, como antenas, que convertem sinais elétricos em ondas eletromagnéticas. Esse efeito é essencial para a geração e transmissão de energia elétrica, além de ser a base para muitas inovações tecnológicas.
- **Efeito Magnetoresistivo ou Magnetoelétrico:** Este fenômeno ocorre em determinados materiais que apresentam uma variação na resistência elétrica quando expostos a um campo magnético. Essa mudança na resistência é decorrente da interação entre o campo magnético e os portadores de carga no material. O efeito magnetoresistivo é amplamente utilizado em sensores magnéticos e em dispositivos de armazenamento, como discos rígidos, onde a leitura dos dados é realizada através da detecção das alterações na resistência elétrica. Graças à sua capacidade de resposta a campos magnéticos, esse efeito permite a miniaturização e o aumento da capacidade de armazenamento em tecnologias modernas. Assim, materiais que exibem esse efeito são especialmente valiosos em aplicações que exigem precisão e sensibilidade, como em sistemas de navegação e monitoramento industrial.

## Classificação de atuadores de movimento quanto à energia primária

Especificamente, os **atuadores que geram movimentos físicos** podem ser classificados em diferentes categorias com base em seus **princípios de operação**. Esses dispositivos funcionam como transdutores, convertendo diversas formas de energia, como elétrica, hidráulica ou pneumática, em energia mecânica (movimento). Essa conversão é essencial em aplicações industriais e tecnológicas, onde o funcionamento de máquinas e sistemas automatizados é baseado em transformação eficiente de uma forma de energia em outra. É importante destacar que, embora os atuadores possam operar com várias fontes de energia, o controle dessas formas de energia é frequentemente realizado por meio de sistemas digitais, incluindo microcontroladores. Essa abordagem permite uma integração mais eficaz e precisa em sistemas eletrônicos, facilitando o desenvolvimento de soluções automatizadas que atendem a requisitos específicos de desempenho e eficiência. Assim, compreender os diferentes tipos de atuadores de movimento é fundamental para otimizar o funcionamento e a eficácia de sistemas automatizados. Cada classe de atuador possui características, vantagens e desvantagens específicas, que influenciam a escolha de acordo com as necessidades da aplicação. As principais classes incluem:

- **Atuadores Elétricos:** Esses dispositivos, que incluem motores e solenóides, convertem energia elétrica diretamente em movimento mecânico e são amplamente utilizados em sistemas microcontrolados devido à sua facilidade de controle. Os **motores elétricos** são projetados para converter energia elétrica em movimento rotativo. Eles são onipresentes em uma variedade de aplicações, como eletrodomésticos, veículos elétricos e máquinas-ferramenta. Sua capacidade de oferecer controle preciso de velocidade e torque os torna ideais para tarefas que exigem movimentação contínua. Os **solenóides**, por outro lado, transformam energia elétrica em movimento linear. Esse tipo de atuador é frequentemente utilizado em sistemas de travamento, válvulas e como componentes de controle em equipamentos automáticos. Sua operação simples e rápida os torna eficientes para aplicações que requerem movimentos curtos e precisos. Ambos os tipos de atuadores elétricos são valorizados pela sua versatilidade e facilidade de integração em sistemas eletrônicos, permitindo um controle eficaz em diversas aplicações industriais e comerciais.
- **Atuadores Pneumáticos:** Esses dispositivos operam com ar comprimido, oferecendo movimentos rápidos e uma excelente relação entre força e peso, sendo amplamente utilizados em diversas aplicações industriais. Os **cilindros pneumáticos** são um exemplo clássico, utilizando ar comprimido para criar movimento linear. Esses atuadores são comuns em sistemas de automação e robótica, onde a velocidade e a precisão são essenciais. Sua capacidade de gerar força considerável em relação ao seu peso os torna ideais para tarefas que requerem movimentação rápida e eficiente. Os **motores pneumáticos** geram movimento rotativo utilizando ar comprimido. Eles são utilizados em uma variedade de aplicações industriais, como máquinas de montagem e ferramentas pneumáticas. Esses motores são valorizados por sua durabilidade e alta potência, sendo uma escolha popular em ambientes que exigem operação contínua e

resistência. Ambos os tipos de atuadores pneumáticos são fundamentais em sistemas que exigem alta velocidade e resposta rápida, destacando-se pela sua eficiência e capacidade de manuseio em processos automatizados.

- **Atuadores Hidráulicos:** Esses dispositivos utilizam fluidos pressurizados para gerar movimento mecânico, oferecendo alta força e precisão, sendo essenciais em diversas aplicações industriais. Os **cilindros hidráulicos** são um exemplo notável, utilizando fluidos pressurizados, geralmente óleo ou água, para criar movimento linear. Eles são amplamente empregados em máquinas industriais, como prensas hidráulicas, onde a necessidade de força robusta e controle preciso é essencial. Esses cilindros são projetados para suportar cargas pesadas e operar em condições severas. Os **motores hidráulicos** convertem a energia hidráulica em movimento rotativo. Esses motores são comumente utilizados em veículos pesados, como escavadeiras e guindastes, onde a capacidade de gerar força significativa em espaços compactos é essencial. Sua operação eficiente e durabilidade os tornam ideais para aplicações que exigem movimentação constante e confiável. Ambos os tipos de atuadores hidráulicos são fundamentais em sistemas que necessitam de potência e controle precisos, destacando-se pela sua eficácia em aplicações industriais pesadas e em maquinário de grande porte.
- **Atuadores Mecânicos:** Esses dispositivos dependem de mecanismos físicos, como molas ou engrenagens, para gerar movimento. Eles operam com base em princípios mecânicos, aproveitando a força e a resistência de materiais para realizar tarefas específicas. Um exemplo comum de atuador mecânico é o **parafuso de avanço**, que converte movimento rotativo em movimento linear. Esse tipo de atuador é frequentemente utilizado em máquinas-ferramenta e dispositivos de posicionamento, onde a precisão e o controle são essenciais. Através do giro do parafuso, é possível mover uma carga de forma controlada, permitindo ajustes finos em diversas aplicações. Os atuadores mecânicos são valorizados pela sua simplicidade e robustez, sendo uma escolha ideal para sistemas onde a eletricidade não é desejada ou onde a simplicidade do *design* é prioritária. Sua operação é frequentemente direta e confiável, tornando-os adequados para uma ampla gama de aplicações, desde a automação industrial até dispositivos de uso cotidiano.
- **Atuadores Piezoelétricos:** Esses dispositivos utilizam o efeito piezoelétrico para converter tensão elétrica em deformação mecânica, possibilitando movimentos altamente precisos. Essa capacidade de gerar pequenas mudanças de forma em resposta a variações de tensão os torna ideais para aplicações que exigem controle fino. Um exemplo notável de atuador piezoelétrico é encontrado em **microfones**, onde a deformação mecânica gerada pelas ondas sonoras é convertida em sinal elétrico, permitindo a captura de som com alta fidelidade. Outro uso comum é nas **impressoras 3D**, onde os atuadores piezoelétricos controlam a extrusão do material, possibilitando a criação de objetos com detalhes complexos. Os atuadores piezoelétricos são altamente valorizados em áreas que requerem rapidez e precisão, como em

dispositivos médicos, sistemas de controle de movimento e tecnologia de áudio. Sua capacidade de resposta rápida e compacta os torna essenciais em aplicações modernas que demandam tecnologia avançada e desempenho confiável.

- **Atuadores Térmicos:** Esses dispositivos funcionam com base na expansão térmica de materiais quando aquecidos, gerando movimento. Essa propriedade de expansão permite que os atuadores térmicos realizem movimentos lineares ou rotativos em resposta a variações de temperatura. Um exemplo típico de atuador térmico é encontrado em **termostatos**, onde a expansão de um material bimetálico, composto por duas ligas metálicas diferentes, provoca o movimento de um interruptor. Esse mecanismo é amplamente utilizado em sistemas de controle de temperatura, como aquecedores e sistemas de refrigeração, permitindo a manutenção de temperaturas desejadas de forma automática. Os atuadores térmicos são valorizados por sua simplicidade e eficácia, sendo uma escolha confiável para aplicações que requerem resposta a mudanças de temperatura. Sua operação passiva e não eletrônica os torna ideais para ambientes onde a eletricidade pode não estar disponível ou ser desejada, destacando-se em sistemas de automação e controle ambiental.

### **Classificação de atuadores quanto ao consumo de energia**

Com base na quantidade de energia que consomem ou na **potência de entrada necessária** para sua operação, os atuadores podem ser classificados em duas categorias: atuadores de baixa potência e atuadores de alta potência. Essa classificação ajuda na identificação de quais atuadores são mais adequados para diferentes aplicações, especialmente em sistemas microcontrolados, onde o consumo de energia é uma consideração crítica. Os **atuadores de baixa potência** são projetados para operar com sinais elétricos de baixa potência, incluindo atuadores piezoeletricos e atuadores eletromagnéticos de baixa potência. Eles são ideais para aplicações onde a eficiência energética é fundamental, como em dispositivos portáteis, sensores alimentados por baterias e sistemas autônomos.

Em contraste, os **atuadores de alta potência** consomem mais energia e são alimentados por fontes de energia robustas, como motores elétricos, atuadores hidráulicos e pneumáticos. Esses atuadores são mais apropriados para aplicações que exigem movimentos de alta força ou torque, como máquinas industriais, veículos e sistemas de automação. Ao projetar um circuito de acionamento para interface com um microcontrolador, é essencial considerar as características específicas do atuador em questão. Atuadores de alta potência exigem um projeto de circuito cuidadoso devido às suas demandas de energia e controle. Eles frequentemente requerem correntes elétricas mais elevadas e tensões mais altas do que os dispositivos de baixa potência, além de circuitos de controle especializados para garantir um acionamento eficiente e seguro. Adicionalmente, é fundamental implementar medidas de proteção contra sobrecargas e curto-circuitos, bem como técnicas de controle de velocidade, direção e posicionamento, conforme as necessidades da aplicação.

## Calibração de transdutores

A calibração de sensores e atuadores é uma etapa fundamental no desenvolvimento e manutenção de sistemas de medição e controle de alta precisão. Ela envolve a comparação dos valores de saída dos sensores e ações dos atuadores com um padrão de referência de alta exatidão para determinar qualquer desvio ou erro na medição ou no controle. Essa análise fornece informações críticas sobre o desempenho real dos componentes e ajuda a garantir que suas saídas sejam confiáveis. Mesmo componentes de excelente qualidade podem sofrer pequenas variações ao longo do tempo ou devido a condições ambientais, o que pode afetar a **exatidão** ou a **repetibilidade das medições e das ações**. Portanto, é fundamental que os usuários considerem a necessidade de calibração e avaliem os métodos disponíveis para ajustar e otimizar seus sensores e atuadores, mesmo quando a função de calibração não está disponível diretamente no dispositivo.

A facilidade de calibração e a necessidade de ajustes periódicos são considerações importantes na escolha de um sensor ou atuador. A disponibilidade da função de calibração em sensores e atuadores pode variar significativamente com base no fabricante e na faixa de preço do componente. Enquanto sensores e atuadores mais sofisticados e caros tendem a incluir essa função, dispositivos mais acessíveis podem não oferecer a capacidade de calibração integrada. No entanto, **a importância da calibração é universal e se aplica a todos os tipos de sensores e atuadores, independentemente de seu custo.**

## CONEXÃO DOS TRANSDUTORES COM MICROCONTROLADORES

Em sistemas embarcados, sensores e atuadores são os principais meios de interação entre o mundo físico e os sistemas digitais. O microcontrolador atua como o núcleo de processamento e controle, recebendo informações do ambiente por meio dos sensores e enviando comandos aos atuadores para modificar esse ambiente. Essa comunicação pode ocorrer de formas diversas, dependendo da natureza do sinal, da complexidade do dispositivo e das exigências da aplicação.

### Conexão direta

A conexão mais básica é feita por meio de sinais analógicos e digitais diretos, como explorado nos roteiros anteriores. Os sinais digitais de propósito geral (GPIO) operam em dois estados binários (0 ou 1), correspondendo a tensões de terra e máxima, respectivamente. Por exemplo, uma saída digital controla o acionamento de um LED, enquanto entradas digitais, como as de botões ou teclados de membrana, têm seus pressionamentos detectados pela transição de seus dois estados. Já os sinais analógicos apresentam variação contínua. Um potenciômetro ou um *joystick*, por exemplo, utilizam entradas analógicas para que o microcontrolador possa ler suas variações de resistência conforme são manipulados. A implementação dessas funcionalidades requer a conexão de entradas e saídas digitais aos pinos [GPIO](#) e a manipulação via registradores específicos, conforme detalhado nos Roteiros 1

a 6 e 8. Por outro lado, as entradas e saídas analógicas são conectadas a pinos dedicados e configuradas através dos registradores dos módulos [ADC](#) (para entradas) e [DAC](#) (para saídas), como demonstrado no Roteiro 9.

## Conexão estruturada

Embora pareça uma solução simples, conectar sensores e atuadores diretamente aos pinos digitais (GPIO) ou analógicos (ADC/DAC) de um microcontrolador compromete a eficiência, escalabilidade e manutenibilidade do sistema. Essa abordagem direta é pouco flexível e se torna rapidamente inviável à medida que o número de dispositivos cresce, aumentando exponencialmente a complexidade do circuito e dificultando seu gerenciamento. Além disso, a ausência de uma estrutura padronizada de comunicação torna o sistema mais vulnerável a ruídos e interferências eletromagnéticas, problemas comuns em muitos ambientes operacionais. Isso pode causar leituras imprecisas e falhas na execução de comandos. Diante dessas limitações, o uso de uma comunicação estruturada passou a ser essencial para garantir a confiabilidade e a organização dos sistemas embarcados. Uma forma eficiente de estruturar a comunicação é o uso de protocolos **seriais assíncronos** com formato *Start-Stop*, como discutido no Roteiro 7. Esse formato permite, por exemplo, a inclusão de um *bit* de paridade para a detecção de erros de um *bit* durante a transmissão, aumentando a integridade dos dados e contribuindo para a robustez do sistema.

Historicamente, em ambientes industriais e de varejo, a comunicação ponto a ponto via RS-232 foi amplamente utilizada devido à sua simplicidade e robustez, apesar das limitações em alcance e taxa de transmissão. Mesmo com a popularização de interfaces modernas como USB e Ethernet, dispositivos RS-232 ainda são comuns em sistemas legados ou aplicações que não demandam alta velocidade de comunicação. Um exemplo típico é o controle de motores de passo, em que a interface RS-232 é usada para enviar comandos a partir de microcontroladores, computadores ou controladores lógicos programáveis (CLPs). Esses comandos são enviados como sequências de caracteres que o controlador interpreta para posicionar o motor conforme desejado. Para possibilitar essa comunicação, utilizamos o módulo [UART](#), presente na maioria dos microcontroladores, em conjunto com um **transceiver** RS-232, como o [MAX232](#) ou equivalente. Esse componente converte os níveis de tensão e lógica entre os padrões TTL/CMOS do UART e os níveis mais elevados e invertidos do RS-232. Como vimos no Roteiro 7, embora UART e RS-232 compartilhem o formato *Start-Stop*, diferem nos níveis elétricos, o que torna essa conversão indispensável.

Além de oferecer simplicidade e robustez, protocolos de comunicação estruturada permitem a implementação de mecanismos hierárquicos, como a distinção entre dispositivos mestres e escravos, o que organiza o tráfego de dados e evita colisões. Esse tipo de estrutura é especialmente útil em sistemas complexos e distribuídos, nos quais múltiplos sensores e atuadores devem operar de forma coordenada. Outro benefício importante é a versatilidade na integração de diferentes tipos de dispositivos e sinais, facilitando a adaptação do sistema a novas demandas ou tecnologias. Além disso, o uso de interfaces bem definidas favorece a modularidade, permitindo a substituição ou atualização de componentes sem grandes

alterações no restante da arquitetura. Isso reduz custos de manutenção e amplia a flexibilidade e a longevidade do sistema.

Portanto, embora a conexão direta entre sensores, atuadores e microcontroladores possa parecer uma solução rápida e simples, a adoção de uma comunicação estruturada é fundamental para garantir a eficiência, escalabilidade e confiabilidade dos sistemas embarcados. Microcontroladores modernos oferecem suporte a uma variedade de protocolos seriais, como os assíncronos que exploramos no Roteiro 7, e os **síncronos**, como [I2C](#), [SPI](#), e [CAN](#), que serão abordados neste e no Roteiro 10. Esses protocolos não apenas aumentam a confiabilidade na troca de dados, mas também oferecem a flexibilidade necessária para atender às demandas específicas de cada aplicação, facilitando o projeto e reduzindo o tempo de desenvolvimento.

## Cone&ccedil;óao estruturada e compatível

Com a evolução dos sistemas embarcados, os transdutores passaram a incorporar internamente circuitos que antes precisavam ser implementados externamente. Tradicionalmente, era necessário projetar **circuitos de interface** usando componentes discretos para adaptar os sinais físicos aos requisitos elétricos dos microcontroladores. Exemplos típicos incluem:

- filtros analógicos para suavizar sinais ruidosos;
- *transceivers* e isoladores para adequar e proteger os sinais de comunicação;
- amplificadores operacionais, usados para sinais analógicos de baixa amplitude;
- resistores de *pull-up* ou *pull-down* para manter níveis lógicos definidos nas entradas digitais;
- *drivers* de potência, como a ponte H (vista no Roteiro 6), usados no controle de motores;
- fontes externas de alimentação, necessárias para sensores ativos ou atuadores com alta demanda energética; e
- conversores analógico-digital (ADC) e digital-analógico (DAC), indispensáveis para captar sinais analógicos do ambiente e gerar saídas analógicas a partir de dados digitais, e vice-versa.

Os **transdutores modernos**, especialmente os baseados em tecnologia MEMS ou os chamados **transdutores inteligentes**, integram muitos desses circuitos diretamente no seu *chip*. Isso permite que eles se comuniquem com microcontroladores usando protocolos digitais padronizados, como I2C ou SPI, tornando seus sinais de entrada e saída diretamente compatíveis com os níveis lógicos dos microcontroladores. Dentre as diversas vantagens que essa integração traz, destacam-se:

- redução de componentes externos, simplificando o projeto de *hardware*;
- menor espaço físico ocupado, facilitando o uso em sistemas compactos;
- maior confiabilidade, devido à redução de conexões e fontes de ruído;

- facilidade de uso, pois o desenvolvedor interage com o sensor principalmente por meio da **programação de registradores** internos, configurando seu funcionamento conforme as demandas específicas da aplicação.

Apesar dessa alta integração, ainda existem aspectos que exigem atenção no projeto do *hardware*. Muitos desafios típicos de sinal analógico, interferência eletromagnética e ruídos de ambiente ainda não encontram soluções exclusivamente via *software* ou integração em *chip*. **Filtros passa-baixa** simples, formados por um resistor e um capacitor, por exemplo, continuam sendo necessários para suavizar sinais analógicos ruidosos e melhorar a precisão das medições. Esse tipo de solução física ainda é essencial em cenários onde os transdutores operam em ambientes eletricamente ruidosos ou com medições de alta sensibilidade. Além disso, boas práticas de projeto, como o uso de **resistores de pull-up ou pull-down** para manter níveis lógicos definidos nas entradas digitais, continuam indispensáveis para garantir estabilidade, confiabilidade e economia de processamento.

Portanto, o avanço dos transdutores em direção à maior integração funcional representa um passo importante na evolução dos sistemas embarcados. Mas é importante reconhecer que ainda não se atingiu um equilíbrio ideal (em inglês, *trade-off*) entre *hardware* e *software* capaz de eliminar completamente as necessidades de circuitos externos. A integração reduz a complexidade, mas não substitui totalmente o cuidado com os detalhes do projeto físico. Ainda é papel do desenvolvedor garantir, com soluções complementares, a integridade e o desempenho do sistema.

## Programação de transdutores

A principal característica dos sensores e atuadores modernos que incorporam internamente controladores ou microcontroladores é a presença de registradores internos, de controle, dados e estado, que podem ser configurados por meio de *software*, espelhando a flexibilidade encontrada na programação de microcontroladores convencionais. Nesses casos, “programar o transdutor” significa gerar uma sequência de *bytes* codificados no microcontrolador, segundo o protocolo de comunicação adotado (como I2C, SPI ou UART), e enviá-los para que o transdutor decodifique e execute a ação correspondente. Muitas vezes, essa flexibilidade permite ajustar uma variedade de parâmetros, tais como:

- frequência de amostragem;
- limiares de operação;
- modos de funcionamento (normal, *standby*, interrupção); e
- estratégias de resposta ou alarmes.

É também por meio dessa programação que se torna possível acessar as amostras coletadas pelos sensores ou enviar valores de atuação para os atuadores.

Por exemplo, ao configurar um sensor digital de temperatura, é possível definir a periodicidade das leituras, as condições que disparam alertas e até o formato de saída dos

dados. Para facilitar essa interação, muitos fabricantes disponibilizam APIs (do inglês *Application Programming Interface*) e bibliotecas específicas, compatíveis com plataformas populares. Exemplos incluem:

- a [biblioteca DHT](#), para sensores de temperatura e umidade;
- a [biblioteca Servo](#), para controle de motores servo.

Essas bibliotecas abstraem o envio dos *bytes* de controle e simplificam a configuração do dispositivo, permitindo que o desenvolvedor acesse suas funcionalidades com chamadas simples de função, o que acelera significativamente o desenvolvimento. Essa capacidade de configuração por *software* também viabiliza a execução de lógicas locais mais avançadas, como algoritmos de decisão em tempo real, pré-processamento de dados e, em alguns casos, até recursos de aprendizado de máquina. Além disso, em sistemas distribuídos, como redes de sensores em automação industrial ou aplicações de IoT, a possibilidade de personalizar os protocolos e o comportamento de comunicação permite integrar sensores e atuadores de forma eficiente e escalável, adaptando-se a diferentes topologias e exigências do sistema.

## COMUNICAÇÃO SERIAL SÍNCRONA

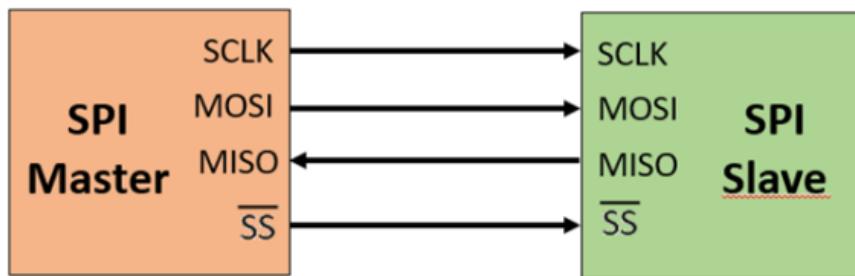
A comunicação serial síncrona é fundamental para a interação eficiente entre microcontroladores e periféricos em sistemas embarcados. No Roteiro 7, exploramos a comunicação serial assíncrona, amplamente utilizada em interfaces como RS-232, UART/USART e USB. Naquela ocasião, discutimos as diferenças fundamentais entre os modelos **assíncrono**, que não utiliza sinal de *clock* compartilhado, e **síncrono**, que depende de um sinal de *clock* comum para sincronizar a troca de dados entre os dispositivos. Embora a comunicação assíncrona, como a RS-232, tenha sido amplamente adotada por décadas, ela apresenta algumas limitações em sistemas embarcados modernos. Seu consumo elevado de pinos para conexões ponto a ponto via conectores [DB-9 ou DB-25](#), a limitação na taxa de transmissão, a dificuldade de conectar múltiplos dispositivos e a baixa imunidade a ruídos em ambientes ruidosos motivaram a busca por alternativas mais robustas e eficientes.

Nesse contexto, surgiram protocolos seriais síncronos como o I2C, o SPI e, para redes de maior porte e mais tolerantes a falhas, o CAN. Esses protocolos oferecem vantagens claras, como maior taxa de transmissão, redução no número de fios, facilidade na conexão de múltiplos dispositivos e maior confiabilidade na comunicação, especialmente em ambientes industriais ou automotivos. Nesta seção, aprofundaremos dois dos protocolos síncronos mais utilizados em sistemas embarcados: o SPI (do inglês *Serial Peripheral Interface*) e o I2C (do inglês *Inter-Integrated Circuit*). Ambos proporcionam soluções robustas para a troca de dados entre microcontroladores e periféricos, porém com características distintas que os tornam mais adequados para diferentes tipos de aplicações. O **SPI** se destaca por sua alta velocidade, baixa latência e simplicidade na implementação, sendo ideal para sistemas que exigem transferência rápida e comunicação ponto a ponto ou com poucos dispositivos. Por outro lado, o **I2C** oferece a vantagem de utilizar apenas dois fios para interligar múltiplos dispositivos,

tornando-o especialmente útil em sistemas com restrição de espaço e necessidade de conexão com diversos sensores ou atuadores.

## Interface serial SPI

A **Serial Peripheral Interface (SPI)** é um barramento de quatro fios desenvolvido pela Motorola. Sua arquitetura compreende uma linha de *clock* (SCLK), uma de saída do mestre/entrada do escravo (MOSI - *Master Output-Slave Input*), uma de entrada do mestre/saída do escravo (MISO - *Master Input-Slave Output*) e uma de seleção de dispositivo (SS - *Slave Select*), que pode ser chamada também de *Chip Select (CS)* ou *Chip Enable (CE)*. Variações que transferem múltiplos *bits* em paralelo (até 4 *bits*) aumentam ainda mais seu desempenho. Essa robustez o torna ideal para aplicações de alta demanda, como a conexão de memória *flash* serial que armazena o código de inicialização (*boot*) em plataformas Intel.

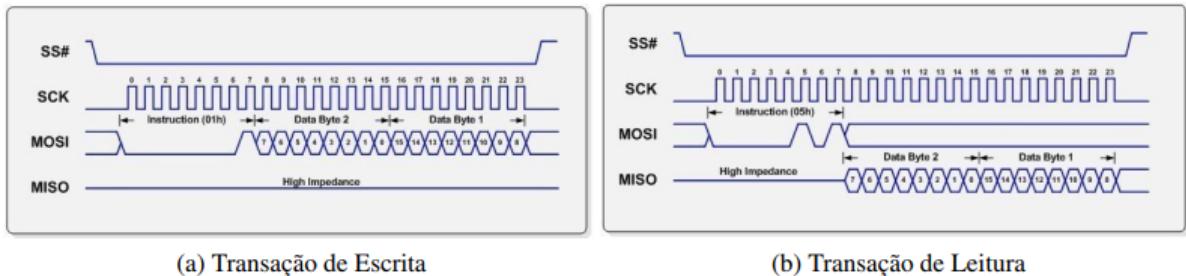


Fonte: [SurfVHDL](#)

O barramento SPI é considerado um barramento **full-duplex**. Ele utiliza um protocolo síncrono em que os dados são transmitidos e recebidos simultaneamente, em sincronia com o sinal de relógio SCLK gerado pelo mestre, usando dois fios distintos, MISO (*Master In Slave Out*) e MOSI (*Master Out Slave In*). Quando o mestre transmite um *bit* pela linha MOSI, ele simultaneamente recebe um *bit* correspondente pela linha MISO. Diferentemente de protocolos como o serial assíncrono ou o I2C, o SPI não inclui *bits* de *START*, *STOP* ou *ACK* específicos. O SPI usa um pino de seleção de escravo (em inglês, *Slave Select – SS*), também frequentemente chamado de pino de seleção de *chip* (em inglês, *Chip Select – CS*), para indicar o início e o término de uma transação.

A transferência começa quando o mestre ativa o pino de seleção de escravo (do inglês, *Chip Select - CS*) para o dispositivo com o qual deseja se comunicar, puxando a linha /CS correspondente a esse escravo para um nível lógico **baixo**. Após a ativação do /CS (geralmente em nível lógico baixo, por isso usaremos a notação com uma “/” na frente do nome do sinal CS para indicar ativo em nível baixo), o mestre envia os *bits* de dados para o dispositivo. A transição do pino /CS de alto para baixo sinaliza ao escravo que uma nova transação está prestes a começar e que ele deve começar a “escutar” os dados no pino MOSI e

a gerar dados no pino MISO, sincronizado com o *clock* (SCLK) do mestre. Os dados são transmitidos em pacotes de 8 *bits* (ou mais, dependendo da configuração). Em algumas configurações do SPI, os *bits* de endereço podem ser incluídos antes dos dados para selecionar um registrador ou função específica no dispositivo. No entanto, isso não é uma parte padrão do protocolo SPI e depende da implementação do dispositivo. A transmissão termina quando o mestre desativa o pino /CS (geralmente colocando-o em nível alto), indicando o fim da comunicação com o dispositivo.



Fonte:

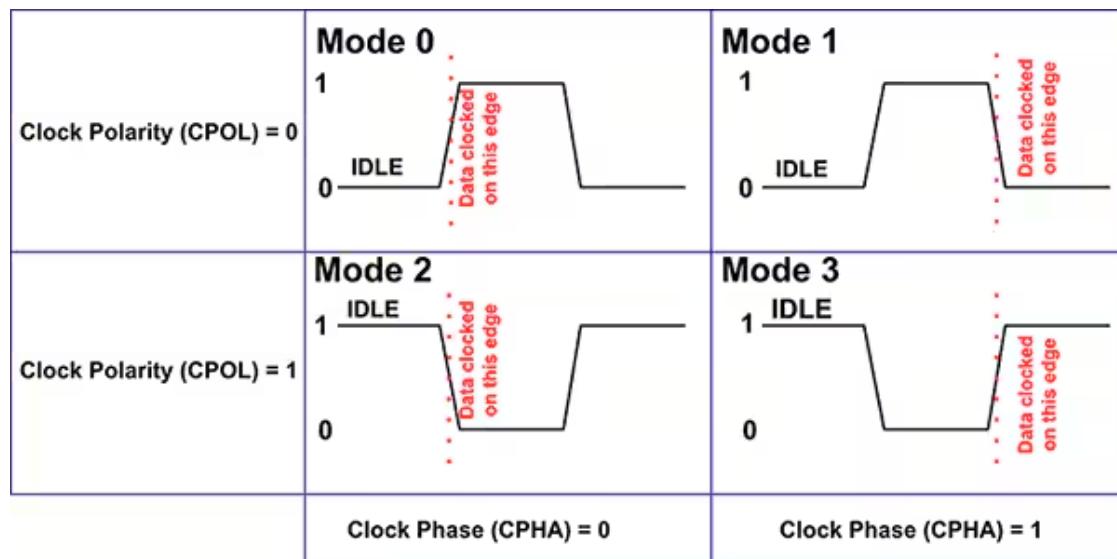
Em um barramento SPI, existe um mestre pré-definido, e um ou mais escravos, com a comunicação ocorrendo de forma **ponto-a-ponto** entre o mestre e cada escravo. O mestre sempre gera os sinais de /CS, SCLK e MOSI, e lê o sinal MISO. A seleção do escravo com quem o mestre troca informações é feita pelo sinal /CS. O mestre precisa produzir um sinal /CS distinto para cada escravo, e acionar apenas o sinal /CS correspondente ao escravo com quem se deseja comunicar. Assim, o barramento SPI usa  $3 + n$  fios, onde  $n$  é o número de escravos. Em algumas situações, alguns dos fios podem ser omitidos. Por exemplo, se o mestre apenas envia dados a todos os escravos, sem receber nada, pode-se omitir a linha MISO. Se o mestre apenas recebe dados dos escravos, pode-se omitir a linha MOSI. Se o barramento possui apenas um escravo e este não necessita das transições de /CS para controlar as transferências, pode-se omitir o sinal /CS.

O primeiro parâmetro importante da SPI é a **velocidade** de transmissão, em *bits* por segundo. Esta é definida no mestre, não precisando ser pré-definida nos escravos, pois os mesmos recebem o *clock* gerado pelo mestre. O segundo parâmetro é a **ordem dos bits**, que precisa ser estabelecida previamente entre mestre e escravo. O mais comum é enviar o *bit* mais significativo primeiro (*MSB first*). Outro parâmetro, também pré-estabelecido entre mestre e escravos, é o **formato da comunicação SPI**. Existem dois formatos principais: **Motorola**, o mais comum, e o formato **TI** (do inglês *Texas Instruments Synchronous Serial Frame Format*). Esses formatos diferem principalmente na maneira como o início e a sincronização dos dados são realizados, impactando diretamente o controle do clock e o momento de captura dos dados.

No formato **Motorola**, o mais amplamente adotado, a comunicação é delimitada pelo controle da linha /CS. Enquanto /CS está ativo (nível lógico baixo), o *clock* é gerado e a troca de dados ocorre. Esse formato é configurável por meio de dois parâmetros adicionais, a polaridade do

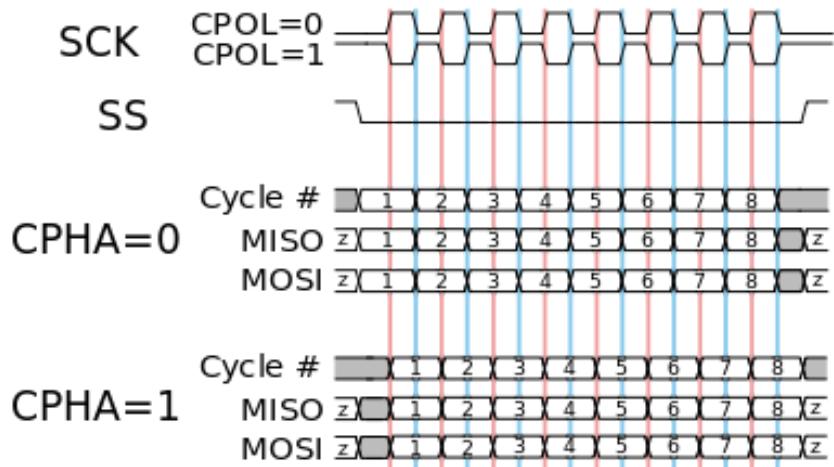
clock (CPOL) e a fase do clock (CPHA), que definem em qual borda do clock os dados são amostrados e transmitidos. Discutiremos esses parâmetros detalhadamente mais adiante. O formato **TI** possui uma diferença estrutural importante. Em vez de depender da linha /CS para indicar o início da comunicação, ele utiliza um **pulso de início** (em inglês, *start pulse*) embutido no próprio sinal de *clock*. Esse pulso ocorre como uma transição de nível alto para baixo do *clock* antes da transmissão dos dados, sinalizando claramente o início de um quadro de dados. Após o pulso de *start*, a comunicação segue de forma síncrona com o *clock*. Na prática, o formato TI permite que a linha /CS **não seja necessária** para sinalizar o início da transmissão, simplificando a conexão elétrica, embora alguns dispositivos possam ainda utilizar /CS para controle adicional ou habilitação geral. O dado é sempre transmitido logo após o pulso de início, no primeiro ciclo completo de *clock*. Essa estratégia oferece uma delimitação robusta dos quadros de dados, sendo útil especialmente em ambientes ruidosos ou em protocolos proprietários da Texas Instruments. Contudo, por ser menos flexível em topologias com múltiplos dispositivos e menos difundido na indústria, seu uso é bem mais restrito que o formato Motorola.

O último parâmetro importante, específico do **formato Motorola**, e também pré-estabelecido, é o **modo**, composto de dois parâmetros. A polaridade do *clock* (*bit CPOL*) determina o estado ocioso do sinal do *clock* (alto ou baixo). Ou seja, se CPOL = 0, o *clock* permanece em nível baixo entre transações SPI. A **fase do clock** (*bit CPHA*) determina quando os dados são capturados em relação ao ciclo do *clock*. Se CPHA = 0, a captura dos dados ocorre na **primeira borda**, enquanto se CPHA = 1, a captura ocorre na **segunda borda**. Note que o modo não define borda de subida ou descida. Se CPOL = 0, a primeira borda será de subida e se CPOL = 1, a primeira borda será de descida. Esses dois parâmetros permitem quatro configurações diferentes (modos SPI 0, 1, 2, 3), dependendo de como CPOL e CPHA são configurados (o número é definido em binário, com os *bits* na ordem CPOL,CPHA).



Fonte: [DigiKey](#)

A figura a seguir sintetiza os quatro modos de operação do formato Motorola, que se distinguem por suas configurações de temporização (em inglês, *clocking*) específicas.



Fonte: [Typhoon HIL Documentation](#)

Por permitir comunicação *full-duplex*, *clock* muito mais alto que I2C e não possuir *overhead* de arbitragem ou controle de endereço a cada transação, a velocidade do SPI é significativamente superior à do I2C ou da comunicação serial assíncrona. O barramento SPI tem taxas de dezenas de MHz (tipicamente 50 MHz, atingindo facilmente velocidades de até 80 MHz), limitado principalmente pela qualidade do cabeamento e pela capacitação de linha.

Note que o protocolo SPI define apenas a forma de transferência de *bytes*. O que cada dispositivo faz com os *bytes* recebidos e o formato dos dados nos *bytes* transmitidos depende das especificações do dispositivo. Por isso, é sempre importante estudar o *datasheet* do dispositivo específico usado no projeto.

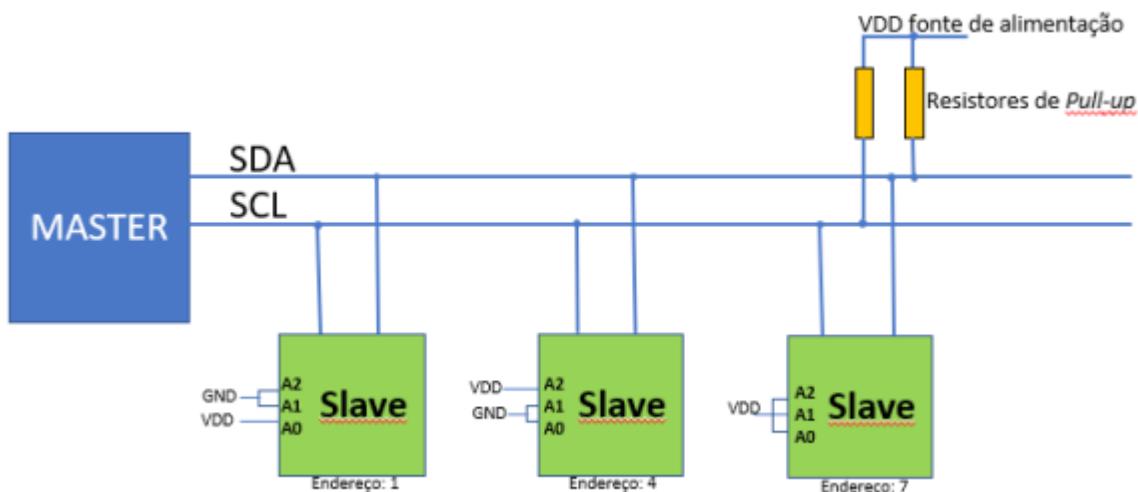
## Interface serial I2C

O barramento I2C foi inventado pela [Philips](#) (NXP) em 1982. É um padrão mundial *de facto* que hoje em dia é implementado em, aproximadamente, 1000 diferentes CI's fabricados por mais de 50 empresas. Adicionalmente, o barramento I2C inspirou várias arquiteturas de controle como *System Management Bus* (SMBus), *Power Management Bus* (PMBus), *Intelligent Platform Management Interface* (IPMI), *Display Data Channel* (DDC) e *Advanced Telecom Computing Architecture* (ATCA). O barramento I2C está presente na maioria dos sistemas embarcados.

O barramento I2C tem somente dois fios bidirecionais em *open collector* ou *open drain*, sendo uma linha de dados (SDA) e uma linha de relógio (SCL). Este é um sistema de comunicação multimestre, permitindo que múltiplos mestres (*Master*) envie dados para vários escravos (*Slaves*). Um dispositivo pode atuar tanto como mestre quanto como escravo. Isso significa que um único dispositivo pode iniciar a comunicação e controlar o barramento (modo mestre) em um momento, e em outro momento, pode responder a solicitações de um

outro dispositivo mestre (modo escravo). É importante lembrar que a capacidade de um dispositivo ser tanto mestre quanto escravo é uma capacidade do I2C, não implicando que todos os dispositivos I2C usem esta capacidade. Muitos dispositivos I2C (como os que veremos nos projetos) funcionam exclusivamente como escravo, e em muitos sistemas embarcados o microcontrolador atua exclusivamente como mestre.

A figura a seguir ilustra um barramento I2C, onde um mestre (em inglês, *Master*) interage com diversos escravos (em inglês, *Slaves*). Cada escravo possui um endereço de dispositivo (em inglês, *device address*) único, estabelecido pelo fabricante e previamente conhecido pelo mestre. Para contornar a limitação de endereços fixos e permitir a utilização de múltiplos dispositivos do mesmo tipo em um único barramento, os fabricantes frequentemente disponibilizam pinos de configuração (como A0, A1, A2, etc.) no encapsulamento do *chip*. Ao definir o nível lógico desses pinos (conectando-os ao terra, 0V, ou à tensão de alimentação, VCC/VDD), os *bits* menos significativos do endereço do dispositivo podem ser alterados. Essa flexibilidade é crucial: pense em um cenário com dois sensores de temperatura idênticos no mesmo barramento I2C; sem essa capacidade, o mestre não conseguiria diferenciá-los. Na figura, os três *bits* menos significativos são configurados pelas entradas A0-A2. Para iniciar a comunicação, o mestre endereça o escravo desejado, realiza a troca de dados (envio ou recebimento) e, então, finaliza a transferência.



Os CI's compatíveis com o barramento I2C permitem que um projeto de sistema progrida rapidamente diretamente do diagrama de blocos funcional para um protótipo. Além disso, como eles são conectados diretamente no barramento I2C sem nenhuma interface externa adicional, eles permitem que um sistema de protótipo seja modificado ou atualizado simplesmente “ligando” ou “soltando” CI's do, ou para o, barramento.

Abaixo estão listados alguns recursos dos CI's compatíveis com o barramento I2C que são particularmente atraentes para os projetistas:

- Os blocos funcionais no diagrama de blocos correspondem aos CI's reais; os projetos avançam rapidamente do diagrama de blocos para o esquema final.

- Não há necessidade de projetar interfaces de barramento, porque a interface de barramento I2C já está integrada no *chip*.
- O protocolo integrado de endereçamento e transferência de dados permite que os sistemas sejam completamente definidos por *software*.
- Os mesmos tipos de CI costumam ser usados em muitas aplicações diferentes.
- O tempo de projeto reduz à medida que os projetistas se familiarizam rapidamente com os blocos funcionais frequentemente representados pelos CI's compatíveis com barramento I2C.
- Os CIs podem ser adicionados ou removidos de um sistema sem afetar outros circuitos no barramento.
- O diagnóstico e a depuração de falhas são simples.
- O tempo de desenvolvimento de *software* pode ser reduzido montando uma biblioteca de módulos de *software* reutilizáveis.

Além dessas vantagens, os CIs CMOS compatíveis com o barramento I2C oferecem aos projetistas recursos especiais que são particularmente atraentes para equipamentos portáteis e sistemas alimentados por bateria. Todos eles têm:

- consumo de corrente extremamente baixo,
- alta imunidade a ruídos,
- ampla faixa de tensões de alimentação, e
- ampla faixa de temperaturas de operação.

O barramento I2C é amplamente popular por sua simplicidade e eficiência. Diferentemente do SPI, o I2C suporta múltiplos mestres, aumentando sua flexibilidade em sistemas complexos. A comunicação no I2C ocorre em pacotes de 8 bits. Embora as especificações iniciais definissem a frequência máxima do *clock* em 100 kHz (*Standard Mode*), o padrão evoluiu para incluir o ***Fast Mode*** (400 kHz), o ***Fast Mode Plus*** (1 MHz), o ***High Speed Mode*** (até 3,4 MHz) e o ***Ultra-Fast Mode*** (5 MHz), oferecendo maior velocidade conforme a necessidade da aplicação. Cada um desses modos possui requisitos temporais específicos para o tempo de subida (em inglês, *rise time*), tempo de descida (em inglês, *fall time*), período de alto e baixo do *clock* (THIGH, TLOW), e também o tempo de *setup* e *hold* dos dados (TSU:DATA, THD:DATA). A seguinte especificação da memória [EEPROM 24LC16](#), por exemplo, ilustra essas características detalhadas.

FIGURE 1-1:

BUS TIMING DATA

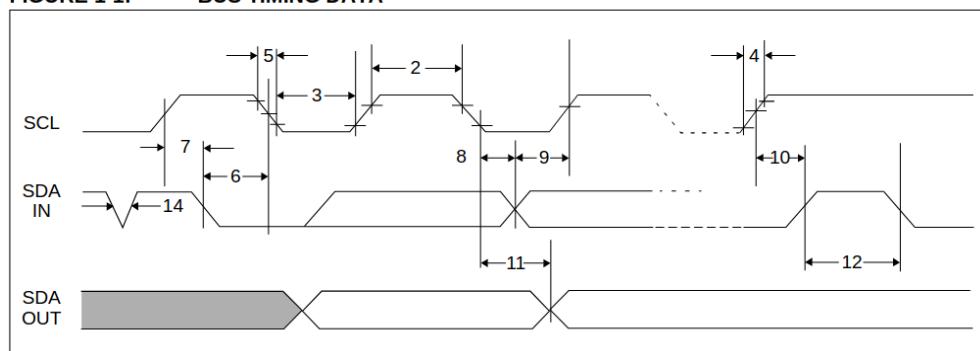


TABLE 1-2: AC CHARACTERISTICS

AC CHARACTERISTICS			Extended (M): TA = -55°C to +125°C, VCC = +2.5V to +5.5V			
Param. No.	Symbol	Characteristic	Min.	Max.	Units	Conditions
1	FCLK	Clock Frequency	—	400	kHz	
2	THIGH	Clock High Time	600	—	ns	
3	TLOW	Clock Low Time	1300	—	ns	
4	TR	SDA and SCL Rise Time <b>(Note 1)</b>	—	300	ns	<b>Note 1</b>
5	TF	SDA and SCL Fall Time	—	300	ns	<b>Note 1</b>
6	THD:STA	Start Condition Hold Time	600	—	ns	
			4000	—	ns	
7	TSU:STA	Start Condition Setup Time	600	—	ns	
8	THD:DAT	Data Input Hold Time	0	—	ns	<b>Note 2</b>
9	TSU:DAT	Data Input Setup Time	100	—	ns	
10	TSU:STO	Stop Condition Setup Time	600	—	ns	
11	TAA	Output Valid from Clock <b>(Note 2)</b>	—	900	ns	
12	TBUF	Bus Free Time: Bus time must be free before a new transmission can start	1300	—	ns	
13	TOF	Output Fall Time from VIH Minimum to VIL Maximum	20+0.1CB	250	ns	
14	TSP	Input Filter Spike Suppression (SDA and SCL pins)	—	50	ns	<b>Notes 1 and 3</b>
15	TWC	Write Cycle Time (byte or page)	—	5	ms	
16		Endurance	1M	—	cycles	Page mode, +25°C, 5.5V <b>(Note 4)</b>

**Note 1:** Not 100% tested. CB = total capacitance of one bus line in pF.

- 2:** As a transmitter, the device must provide an internal minimum delay time to bridge the undefined region (minimum 300 ns) of the falling edge of SCL to avoid unintended generation of Start or Stop conditions.
- 3:** The combined TSP and VHYS specifications are due to new Schmitt Trigger inputs which provide improved noise spike suppression. This eliminates the need for a TI specification for standard operation.
- 4:** This parameter is not tested but ensured by characterization. For endurance estimates in a specific application, please consult the Total Endurance™ Model which can be obtained from Microchip's website at [www.microchip.com](http://www.microchip.com).

Em microcontroladores, a configuração da frequência do I2C não é tão simples quanto selecionar um modo direto. A frequência final do SCL é derivada do *clock* principal do periférico I<sup>2</sup>C, que por sua vez vem do sistema de *clock* do microcontrolador e pode variar significativamente. A configuração desses valores é realizada através de registradores que permitem ao desenvolvedor ajustar finamente parâmetros como THIGH, TLOW, TSU:DATA e THD:DATA. As fórmulas para calcular esses valores levam em conta a frequência do *clock* de entrada do I2C e os tempos mínimos e máximos especificados pelo padrão I2C para cada modo, garantindo que todas as restrições temporais do protocolo sejam atendidas e otimizando a comunicação em diferentes condições de barramento (como comprimento do barramento e capacidade parasita) e requisitos específicos dos dispositivos.

Por fim, vale ressaltar que o protocolo I2C é *half-duplex*, o que significa que o mestre pode enviar ou receber dados, mas não ambos simultaneamente.

Para sua operação, são necessários apenas dois fios com resistores *pull-up*, permitindo a conexão de um grande número de dispositivos (até 127). A resistência típica desses resistores não é um valor fixo, mas sim uma faixa de valores que depende de vários fatores do barramento. No entanto, alguns valores são muito comuns e servem como ponto de partida:

- **10 kΩ (10.000 ohms):** Este é um dos valores mais frequentemente encontrados e funciona bem para barramentos I2C com baixa capacidade (poucos dispositivos, traços curtos) e taxas de comunicação mais lentas (100 kHz - *Standard Mode*).
- **4.7 kΩ (4.700 ohms):** Outro valor muito comum, especialmente quando se deseja um pouco mais de velocidade ou há uma capacidade de barramento um pouco maior. Funciona bem para 400 kHz (*Fast Mode*).
- **2.2 kΩ (2.200 ohms) ou valores ainda menores:** Podem ser usados para taxas de comunicação mais altas, como 1 MHz (*Fast Mode Plus*) ou 3.4 MHz (*High-Speed Mode*), ou em barramentos com alta capacidade, pois resistências menores ajudam a “puxar” a linha para o estado alto mais rapidamente.

A implementação do I2C é facilitada por sua baixa demanda em recursos do microcontrolador. Mesmo dispositivos de menor desempenho ou sem um controlador I2C de hardware dedicado podem utilizá-lo, necessitando apenas de dois pinos de E/S configuráveis como *open-collector* ou *open-drain*. Esse tipo de configuração permite que múltiplos dispositivos compartilhem as mesmas linhas de comunicação sem risco de conflito elétrico, graças ao **comportamento wired-AND**, no qual a linha só permanece em nível lógico alto quando todos os dispositivos estão inativos (liberando a linha). Se qualquer dispositivo “puxar” a linha para baixo (nível lógico zero), ela assumirá imediatamente esse estado. Assim, o microcontrolador precisa apenas gerar as condições de Início/Parada (em inglês, *Start/Stop*) e gerenciar a leitura e escrita de *bytes*, tudo isso com algumas rotinas de *software* simples.

### **Endereços I2C**

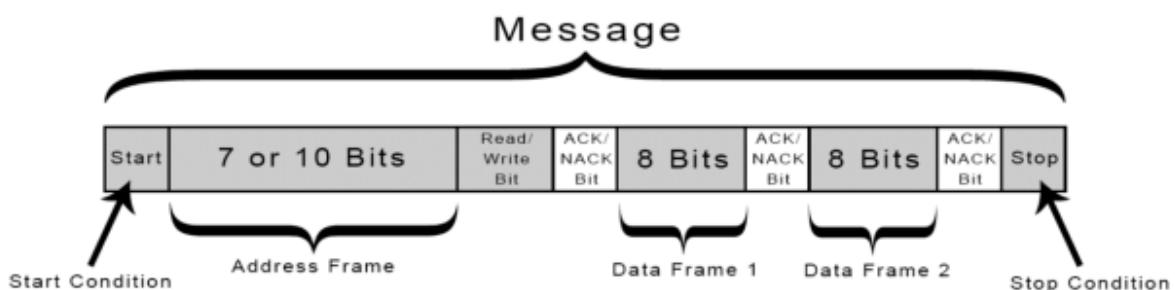
No barramento I2C, cada dispositivo escravo possui um **endereço de 7 bits (device address)**, que deve ser exclusivo em todo o barramento para evitar conflitos de comunicação. Como já comentado, esse endereço é, em parte, pré-definido pelo fabricante. Para flexibilizar o uso de múltiplos dispositivos idênticos, muitos *chips* permitem que os *bits* menos significativos do endereço sejam configurados externamente por meio dos níveis lógicos de pinos específicos. Vale notar que, além do endereçamento de 7 bits, existe também um modo I2C com endereços de 10 bits para sistemas com muitos dispositivos.

No esquema de 7 bits, os *bits* 7 a 1 do primeiro *byte* transmitido representam o endereço do escravo, enquanto o *bit* 0 é crucial para sinalizar a operação: se definido como “1”, indica que o mestre realizará uma leitura do escravo; se “0”, indica uma escrita. O dispositivo mestre, por sua vez, não necessita de um endereço próprio, pois é ele quem gera o *clock* (via SCL) e gerenciaativamente a comunicação, endereçando individualmente cada escravo.

É importante destacar que, diferentemente do I2C, o protocolo SPI não utiliza um sistema de endereçamento de dispositivos. Em vez disso, cada dispositivo escravo no barramento SPI requer uma linha de seleção de escravo (/CS ou /SS) dedicada, conectada diretamente ao mestre. O mestre ativa a linha /CS do escravo com o qual deseja se comunicar, selecionando-o de forma exclusiva para a transmissão de dados. Essa característica é uma das grandes distinções entre os dois protocolos.

### **Protocolo I2C**

No estado normal (*idle*), ambas as linhas (SCL e SDA) estão em nível alto. A comunicação é iniciada pelo dispositivo mestre através do sinal de **START**. É importante notar que qualquer dispositivo no barramento pode se tornar o mestre durante aquele ciclo de transferência (barramento *multi-master*), bastando produzir o sinal de START. No I2C, os dados são transferidos em mensagens. As mensagens são divididas em quadros de dados (em inglês, *data frame*). Cada mensagem possui um quadro de endereços, que contém o endereço binário do escravo, e um ou mais quadros de dados, que contêm os dados que estão sendo transferidos. É importante entender que esses “dados” de uma mensagem geralmente correspondem aos valores que devem ser lidos ou escritos em registradores específicos do dispositivo escravo. A mensagem também inclui condições de início e parada, *bits* de leitura/gravação e bits **ACK/NACK** (*Acknowledge / Non-Acknowledge*) entre cada quadro de dados.



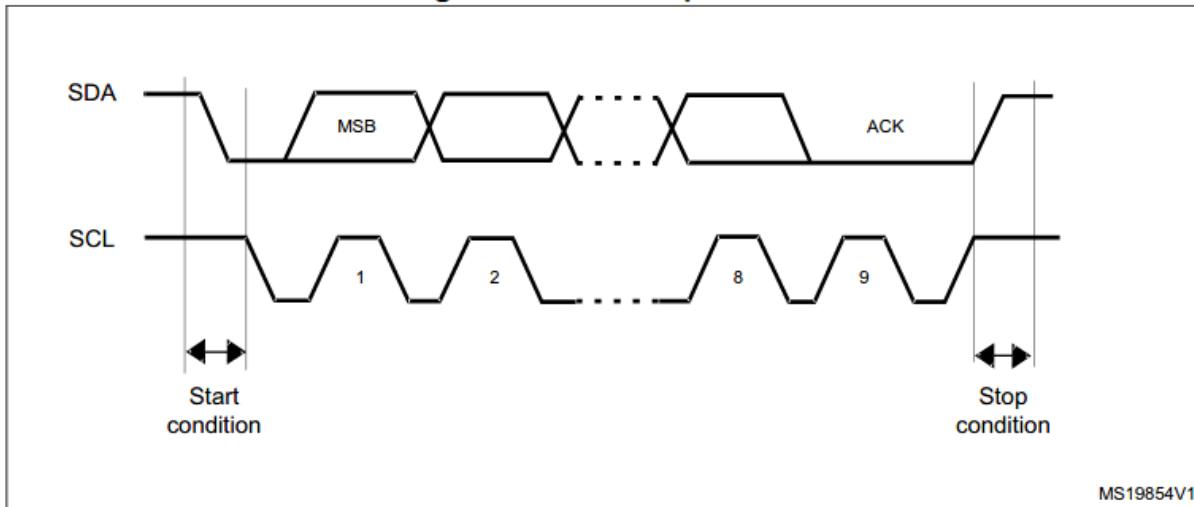
### **Passos da transmissão de dados no protocolo I2C**

Inicialmente, para entender a transmissão de dados no protocolo I2C, é importante considerar a restrição na alteração do nível lógico do sinal SDA. As transições de nível lógico em SDA devem ocorrer apenas enquanto o sinal SCL está em nível baixo, com apenas duas exceções. O sinal de **START** (início do ciclo de transmissão) é executado realizando a mudança no SDA do nível alto para o baixo com SCL em nível alto. O sinal de **STOP** (final do ciclo de transmissão) é executado realizando a mudança no SDA do nível baixo para o alto com SCL em nível alto.

1. O mestre envia uma condição de **START** para todos os escravos conectados. Ao executar o **START**, o mestre sinaliza que ele controla o barramento durante aquele ciclo.

2. O mestre envia aos escravos conectados os 7 ou 10 bits de endereço do escravo com o qual ele quer se comunicar, junto com o bit de leitura/escrita. O mestre muda o SDA, se necessário, enquanto SCL está em nível baixo e os escravos leem o valor em SDA na borda de subida do SCL. Assim, o mestre executa 8 (ou 11) ciclos de clock em SCL. Na sequência, o mestre executa ainda mais um ciclo de clock adicional.
3. Cada escravo compara o endereço enviado pelo mestre com seu próprio endereço. Se o endereço for o seu, o escravo retorna um bit de ACK (Acknowledge), fazendo que a linha SDA fique em nível baixo durante o ciclo adicional de clock gerado pelo mestre. Se o endereço recebido não coincidir com o seu endereço, o escravo deixa a linha SDA em nível alto.
4. Se nenhum dos escravos do barramento tiver seu endereço correspondendo ao endereço enviado pelo mestre, nenhum deles irá produzir o sinal de ACK. Desta forma, a linha SDA permanece em nível lógico alto durante o ciclo adicional de clock, e o mestre entende esta condição como dispositivo escravo ausente do barramento (NACK). Assim, o mestre aborta o ciclo gerando um sinal de STOP e liberando o barramento (linhas SDA e SCL em nível alto).
5. Caso contrário, o mestre envia ou recebe o quadro de dados, controlando o sinal SCL por nove ciclos. Os bits de dados são transferidos nos oito primeiros ciclos.
6. Após cada quadro de dados ser transferido, o dispositivo que o recebeu retorna um bit de ACK para quem o enviou, para reconhecer a recepção com sucesso do quadro. Se a operação é de leitura (o mestre recebe), quando o mestre recebe o último byte que deseja receber, ele retorna um bit de NACK para indicar que não deseja mais receber bytes.
7. Para parar a transmissão de dados, o mestre envia uma condição de STOP, liberando o barramento.

**Figure 512. I<sup>2</sup>C bus protocol**



O protocolo I<sup>2</sup>C possui detalhes importantes, como o sinal de *REPEATED START*. Este sinal consiste na emissão de uma nova condição de START sem que tenha ocorrido previamente a condição de *STOP*. O *REPEATED START* é amplamente utilizado quando o dispositivo

mestre (como os que possuem registradores internos mapeados por endereços e exigem uma escrita para definir o registrador-alvo antes da leitura dos dados) precisa mudar o tipo de operação sem liberar o barramento para outros dispositivos. Sem o sinal de *STOP*, o mestre continua a ter controle exclusivo do barramento. A nova condição de *START* apenas sinaliza que a comunicação continua, mas permite a mudança de direção da transmissão de dados. Um exemplo clássico de sua aplicação é a leitura de um registrador interno em muitos periféricos I2C. Para fazer isso, o mestre primeiro executa uma operação de escrita para enviar o endereço interno do registrador desejado. Em seguida, sem liberar o barramento (ou seja, sem um *STOP*), o mestre emite um *REPEATED START* e, então, inicia uma operação de leitura para obter os dados do registrador especificado.

No protocolo I2C, ACK (*Acknowledgment*) e NACK (*Not Acknowledgment*) são sinais enviados entre dispositivos para indicar a conclusão de uma operação de comunicação.

- ACK (*Acknowledgment*): Quando um dispositivo recebe um *byte* de dados, ele envia um sinal ACK para indicar que recebeu os dados com sucesso e que está pronto para receber mais. Esse sinal é crucial para a continuidade da comunicação, pois permite que o transmissor saiba que o receptor está pronto para o próximo byte.
- NACK (*Not Acknowledgment*): O NACK é enviado pelo receptor para indicar que ele não recebeu os dados corretamente, que não está pronto para receber mais *bytes*, ou que não compreendeu o comando/dado. Existem várias situações que podem gerar um NACK: (1) Ausência de Receptor: Uma das causas mais comuns de NACK, especialmente após o envio do endereço do escravo, é a inexistência de um dispositivo no barramento com o endereço especificado ou a falta de resposta de um dispositivo presente. Ou seja, se o mestre tenta se comunicar com um endereço para o qual não há nenhum escravo "escutando", ele receberá um NACK. (2) Receptor Não Pronto: O receptor pode estar ocupado com outra tarefa e, portanto, não está pronto para receber ou processar mais dados. (3) Dados Inválidos: O receptor pode ter recebido dados ou comandos que não compreende ou que são inválidos para sua operação atual. (4) Fim da Transmissão (pelo Mestre): Em operações de leitura, o mestre (que é o receptor nesses momentos) envia um NACK no último *byte* que deseja receber para indicar que a transmissão deve ser encerrada.

Existe ainda a possibilidade de arbitragem automática do barramento quando dois dispositivos tentam se tornar Mestres do barramento simultaneamente, mas o processo é relativamente complexo, fugindo do escopo deste texto.

Note que, da mesma forma que no protocolo SPI, o I2C define apenas a forma de transferência de *bytes* no barramento. A interpretação dos *bytes* transferidos depende do protocolo específico implementado pelo fabricante do dispositivo. Cada dispositivo I2C pode ter comandos, registradores e formatos de dados únicos que devem ser consultados no *datasheet*. O *datasheet* do dispositivo específico sempre deve ser consultado.

## **Exemplo: Comunicação I2C com memória EEPROM 24CXX**

A família de memórias EEPROM 24CXX consiste em dispositivos de armazenamento não volátil que utilizam o padrão I2C para comunicação. São amplamente usadas em projetos eletrônicos para armazenar dados de forma persistente, mesmo quando não há fornecimento de energia. A sigla "XX" no nome refere-se à capacidade de armazenamento de cada membro da família (dada em *kilobits*, e não em *kilobytes*), como 24C02 (2 *Kbits*), 24C16 (16 *Kbits*), e assim por diante.

As EEPROMs 24CXX destacam-se por diversas características:

1. Capacidades: Variam de 2 *Kbits* (24C02) até 512 *Kbits* (24C512) ou mais, organizados em diferentes números de endereços e páginas.
2. Interface de Comunicação: Utilizam o protocolo I2C, permitindo a comunicação simples e de baixa pinagem (usando apenas dois pinos: SDA e SCL).
3. Tensão de Operação: Geralmente de 1.7V a 5.5V, tornando-as compatíveis com diversos microcontroladores, como Arduino, ESP32, STM32, etc.
4. Endereçamento: Cada dispositivo pode ser configurado para ter um endereço único no barramento I2C, permitindo o uso de vários *chips* na mesma linha de comunicação (tipicamente até 8).
5. Ciclos de Gravação: Suportam tipicamente 1 milhão de ciclos de gravação/apagamento por célula de memória.
6. Retenção de Dados: Os dados armazenados são retidos por até 100 anos.
7. Tamanho das Páginas: A maioria das memórias dessa família permite a escrita de dados em blocos (páginas), com o tamanho de cada página variando entre 8 a 128 *bytes*, dependendo do modelo.

A combinação de simplicidade de uso e a compatibilidade com o barramento I2C faz da família 24CXX uma escolha popular para sistemas embarcados e projetos de *hobby*.

Todos os modelos, com suas diversas capacidades de memória, apresentam um encapsulamento com a mesma pinagem, sendo dois pinos para alimentação (Vcc e GND), dois para a interface (SCL e SDA), um para proteção contra gravação (WP, deve ser mantido em nível baixo para permitir a gravação), e três para definir o *device address* (A<sub>2</sub>, A<sub>1</sub> e A<sub>0</sub>). Os níveis lógicos destes três pinos definem os três *bits* menos significativos do *device address* (pino em nível baixo faz com que o *bit* correspondente no *device address* seja 0).

Em alguns modelos de menor capacidade (como 24C04, 24C08, 24C16), os pinos A<sub>0</sub>, A<sub>1</sub> e/ou A<sub>2</sub> **não são usados para o endereço do dispositivo I2C, mas sim para estender o endereço interno da EEPROM**. Isso é feito para evitar a necessidade de enviar múltiplos *bytes* para o endereço interno, economizando espaço na transmissão. Nesses casos, o dispositivo responde a uma faixa de endereços I2C que incluem esses *bits* como “don't care” no *device address*. Assim, em uma operação de escrita, o Mestre envia a seguinte sequência de sinais e *bytes*:

START - A<sub>6</sub>A<sub>5</sub>A<sub>4</sub>A<sub>3</sub>A<sub>2</sub>A<sub>1</sub>A<sub>0</sub> - E<sub>7</sub>E<sub>6</sub>E<sub>5</sub>E<sub>4</sub>E<sub>3</sub>E<sub>2</sub>E<sub>1</sub>E<sub>0</sub> - Dado1 - Dado2 - ... - Dadon - STOP,

onde A<sub>x</sub> é o *bit x* do *device address* (com A<sub>2..0</sub> definidos nos pinos externos), o *bit de Read / Write* é 0, E<sub>x</sub> é o *bit x* do endereço interno da EEPROM onde a escrita deve iniciar, e Dados são os *bytes* de dados a serem escritos. O *Byte de device address* vai ter a sequência binária 1010A<sub>2</sub>A<sub>1</sub>A<sub>0</sub>0.

Na operação de leitura na EEPROM, o endereço interno não é transmitido ao componente. A EEPROM retorna o dado presente no endereço indicado por um **ponteiro interno**, cujo valor é carregado com os *bits* E<sub>x</sub>. Assim, para se ler a partir de um endereço desejado, deve-se iniciar uma operação de escrita passando o *byte* de endereço e logo depois executar um REPEATED START (idêntico ao START, porém ocorrendo enquanto o barramento ainda está ocupado pelo mestre), mudando a operação para leitura. A sequência fica:

START - A<sub>6</sub>A<sub>5</sub>A<sub>4</sub>A<sub>3</sub>A<sub>2</sub>A<sub>1</sub>A<sub>0</sub> - E<sub>7</sub>E<sub>6</sub>E<sub>5</sub>E<sub>4</sub>E<sub>3</sub>E<sub>2</sub>E<sub>1</sub>E<sub>0</sub> - REPEATED START - A<sub>6</sub>A<sub>5</sub>A<sub>4</sub>A<sub>3</sub>A<sub>2</sub>A<sub>1</sub>A<sub>0</sub>1 - Dado1 - Dado2 - ... - Dadon - STOP (o Master responde cada Dado com um ACK, exceto no último dado, quando responde com um NACK).

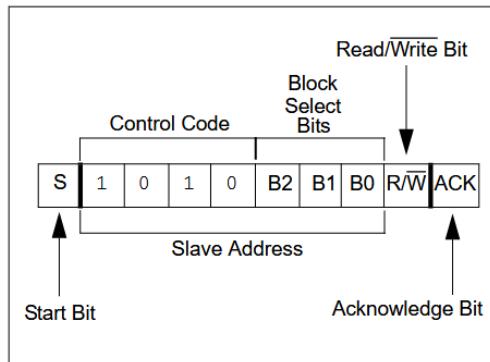
Se for usado um 24C04, teremos 512 *Bytes* ( $2^{2^7}2^3$  bits). Assim, precisamos de 9 *bits* para definir o endereço interno da EEPROM. Seriam necessários 2 *Bytes* de endereço interno para adicionar apenas um *bit*, o que é um desperdício. Neste caso, o *bit* A<sub>0</sub> passa a ter a função de E<sub>8</sub> e o pino A<sub>0</sub> no encapsulamento fica sem função. Ou seja, usa-se parte do *device address* para passar o *bit* mais significativo do endereço interno. Neste caso, o componente irá responder a *device addresses* da seguinte forma: 1010A<sub>2</sub>A<sub>1</sub>X0, onde X é um *bit don't care*.

Usando um 24C08 ( $2^32^72^3$  bits), A<sub>1</sub> e A<sub>0</sub> passam a ter a função de E<sub>9</sub> e E<sub>8</sub> respectivamente. Em um 24C16 (o componente presente na placa auxiliar), com  $2^42^72^3$  bits, A<sub>2</sub>, A<sub>1</sub> e A<sub>0</sub> passam a ter a função de E<sub>10</sub>, E<sub>9</sub> e E<sub>8</sub>, e os pinos A<sub>2</sub>, A<sub>1</sub> e A<sub>0</sub> ficam sem função. No *datasheet*, os 3 *bits* são chamados *block select bits*, como mostra na figura que segue.

Para modelos maiores (como 24C32 em diante), o endereço interno passa a ser transmitido em 2 *Bytes* e os pinos de definição do *device address* voltam a executar sua função original.

Operation	Control Code	Block Select	R/W
Read	1010	Block Address	1
Write	1010	Block Address	0

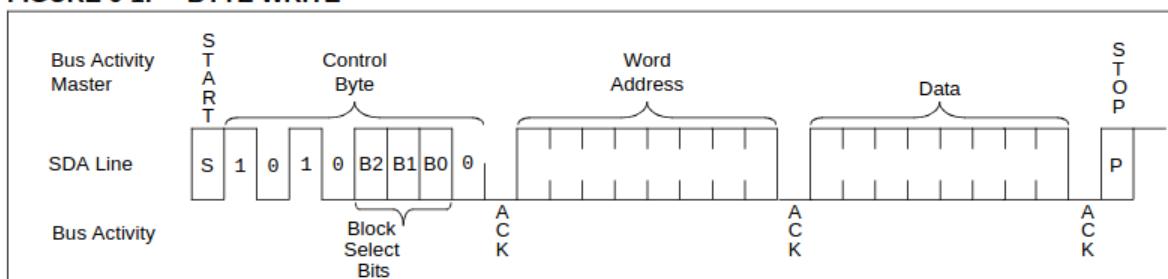
**FIGURE 5-1: CONTROL BYTE ALLOCATION**



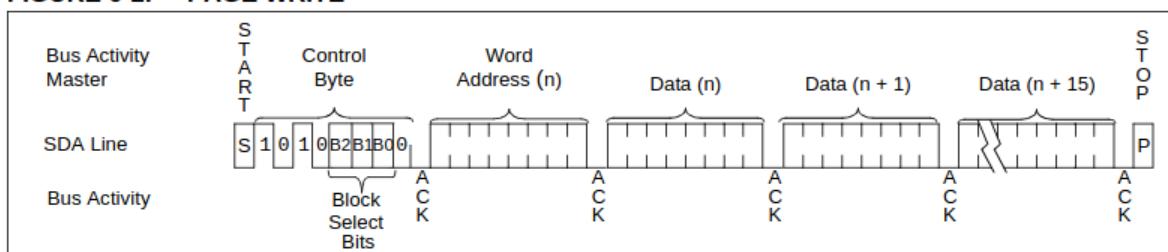
(Fonte)

É importante ter em mente que operações de escrita só podem acontecer dentro de uma mesma página da EEPROM. Se na transmissão os dados passarem do limite de uma página para outra, os dados serão escritos de forma errada dentro da mesma página. No caso da 24C16, o tamanho de página é de 16 Bytes. Assim, não se pode escrever uma sequência de dados que passe do endereço 15 para o 16, do 31 para o 32, etc. Para a leitura sequencial, não há restrições. O tempo máximo de escrita para uma página (em inglês, *Page Write Time*) é especificado como **5 ms máximo**.

**FIGURE 6-1: BYTE WRITE**



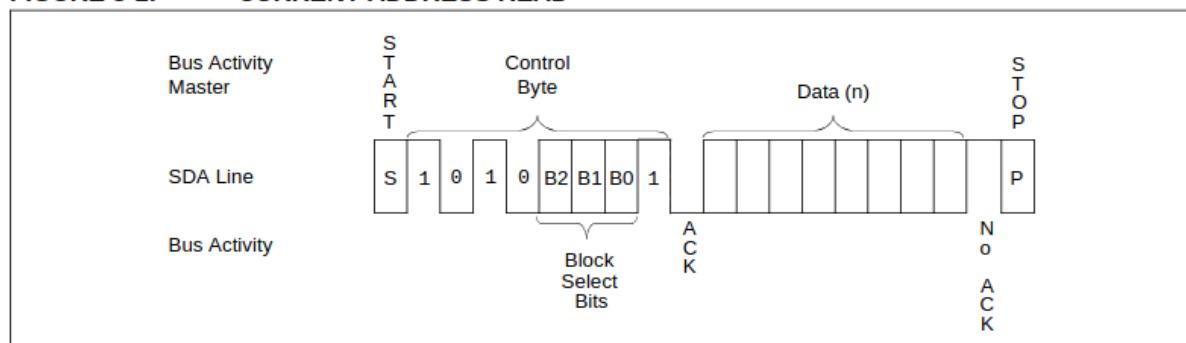
**FIGURE 6-2: PAGE WRITE**



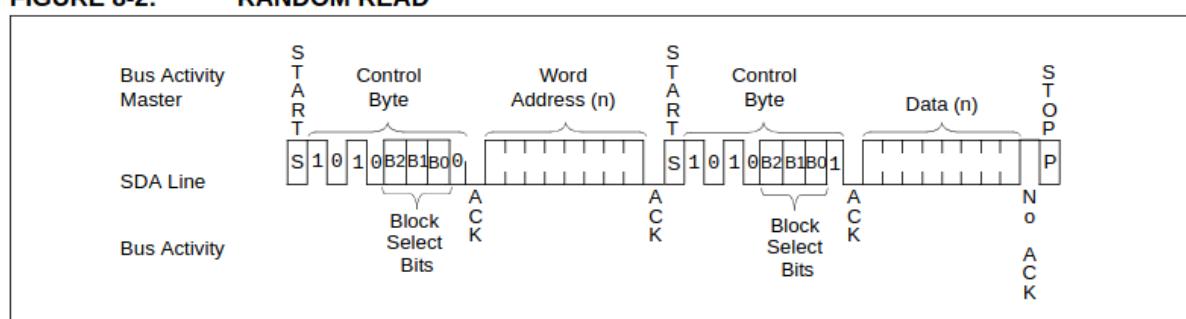
(Fonte)

Uma operação de leitura começa da mesma forma que uma de escrita: o mestre I2C gera uma condição de START no barramento. Logo após a condição de START, o mestre envia o endereço de 7 bits (ou 10 bits) do dispositivo escravo com o qual deseja se comunicar. A diferença está no 8º bit (ou 11º bit para endereçamento de 10 bits), o bit R/W, que segue imediatamente os bits do endereço do escravo, define a direção da transferência de dados subsequente. Se o bit R/W for '0', o mestre pretende escrever dados para o escravo; senão o mestre pretende ler dados do escravo. Após o mestre enviar o byte de endereço (incluindo o bit R/W), espera-se que o dispositivo escravo endereçado envie um bit de ACK (do inglês *Acknowledge*) para confirmar que recebeu seu endereço e está pronto para a comunicação. Em leituras sequenciais são iniciadas da mesma forma que uma leitura aleatória, exceto que, uma vez que a 24LC16B transmite o primeiro byte de dados, o mestre envia um ACK em vez de uma condição de Stop, como ocorreria em uma leitura aleatória. Isso instrui a 24LC16B a transmitir a próxima palavra de 8 bits endereçada sequencialmente.

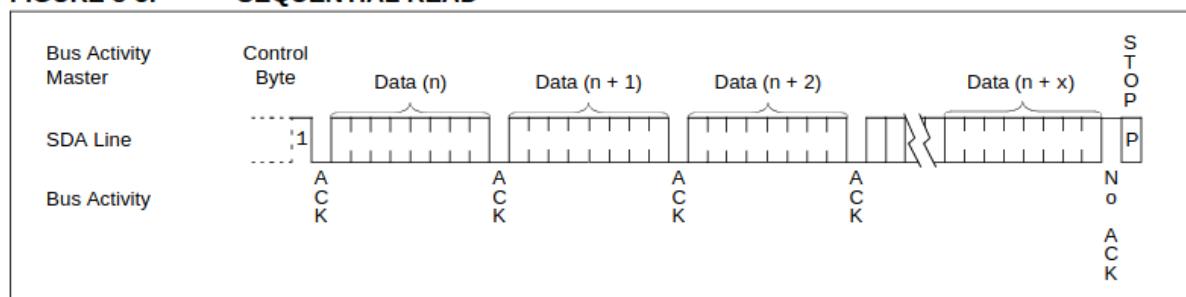
**FIGURE 8-1: CURRENT ADDRESS READ**



**FIGURE 8-2: RANDOM READ**



**FIGURE 8-3: SEQUENTIAL READ**

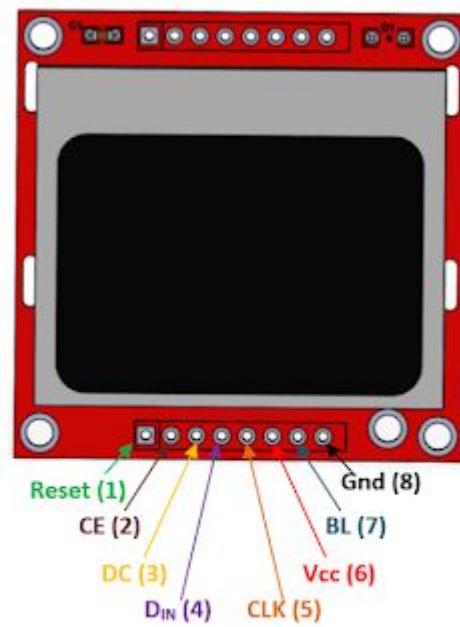


([Fonte](#))

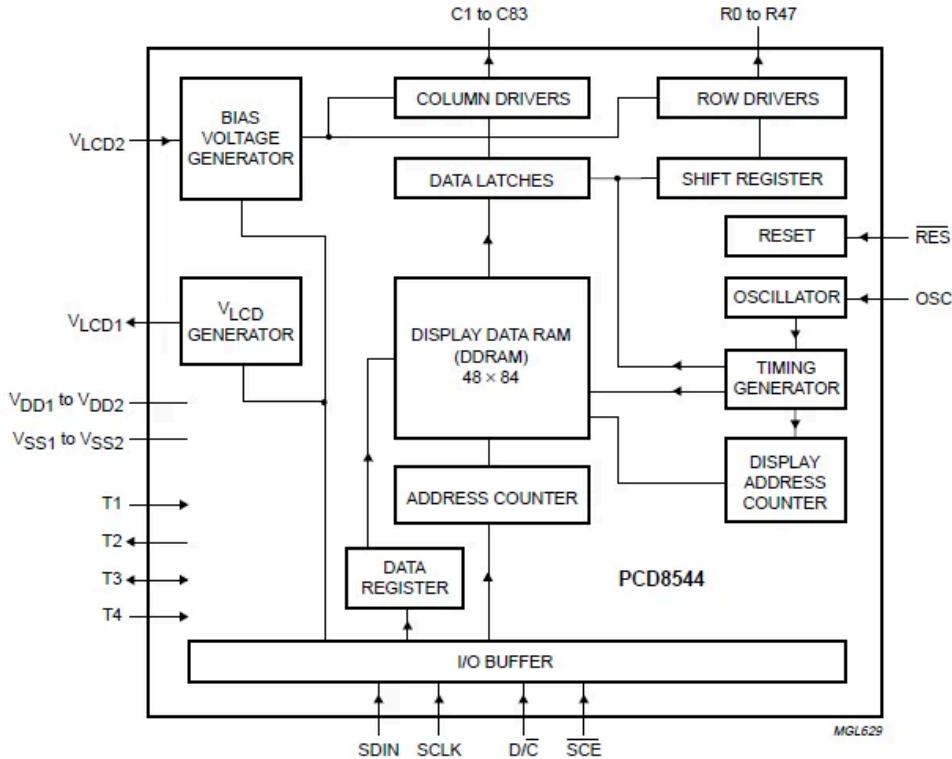
## EXEMPLOS DE SENsoRES E ATUAdoRES PROGRAMÁVEIS

Nesta seção, vamos explorar dois exemplos práticos de sensores e atuadores programáveis, destacando como as interfaces seriais síncronas, como SPI e I2C, facilitam a configuração e o controle desses dispositivos. Os sensores e atuadores escolhidos não apenas ilustram a versatilidade das interfaces seriais, mas também mostram como a integração desses componentes pode resultar em sistemas mais responsivos e precisos.

### Display NOKIA 5110



O **display Nokia 5110** é um módulo gráfico de baixa potência, originalmente usado em celulares Nokia, que se tornou popular em projetos com Arduino e outros microcontroladores. Ele possui uma tela LCD de 84x48 *pixels*, capaz de exibir gráficos e textos em formato monocromático (preto e branco). O controlador integrado é o [PCD8544](#), que se comunica com o microcontrolador via interface SPI.



As principais características deste *display* são:

- Resolução: 84x48 *pixels*.
- Controle: feito pelo controlador PCD8544.
- Interface: SPI (Serial Peripheral Interface).
- Tensão de operação: 3.3V (pode precisar de adaptadores de nível lógico para uso com placas Arduino de 5V).
- Baixo consumo de energia: 200 a 300  $\mu$ A quando ativo e 1 $\mu$ A em modo *sleep*, ideal para projetos com baterias.
- Tamanho compacto: aproximadamente 45mm x 45mm.

O *display* é amplamente utilizado em projetos que precisam de uma interface gráfica simples e de baixo custo, como monitores de sensores, relógios e pequenos jogos. Normalmente ele é vendido na forma de um módulo com o *display* montado e os pinos de seu controlador disponibilizados em uma fileira de pinos. Além disso, o módulo conta com uma luz de fundo (*backlight*) por LEDs, controlada por um dos pinos do módulo. Os LEDs são controlados por um transistor *drive*, portanto uma saída GPIO do microcontrolador pode controlar diretamente o acionamento do *backlight*, mantendo uma corrente de acionamento no pino muito baixa.

Além da interface padrão SPI (unidirecional *slave*, sem o sinal MISO), o módulo possui mais três pinos para sua configuração: um pino de RESET (ativo em nível baixo), um pino de *Data/Command* (**D/C**) e um pino para acender o *backlight* (ativo em nível alto ou baixo, dependendo do fabricante do módulo). O pino D/C determina se o *byte* recebido é um comando (nível baixo) ou um dado (nível alto). Para os comandos existe uma tabela

específica. Para os dados, cada *bit* dentro do *byte* especifica um *pixel* do *display* (1 para ativo, 0 para inativo). A posição do *pixel* depende da configuração realizada com os *bytes* de comando, mas geralmente cada *byte* especifica uma coluna vertical de 8 *pixels*, com o *bit* menos significativo correspondente ao *pixel* superior. Para escrever texto neste *display*, as letras precisam ser “desenhadas”, ou seja, deve-se estabelecer os *pixels* a serem ativados de acordo com a letra e sua posição.

O *display* conta com alguns recursos, como ajuste de contraste, rotação de tela a cada 90 graus e de inversão total da tela, tornando os *pixels* ativos transparentes e os inativos pretos (o oposto do modo normal). Além disso, apresenta um modo *sleep*, onde a tela é desligada mas a interface SPI e a memória com os valores dos *pixels* são mantidos ativos.

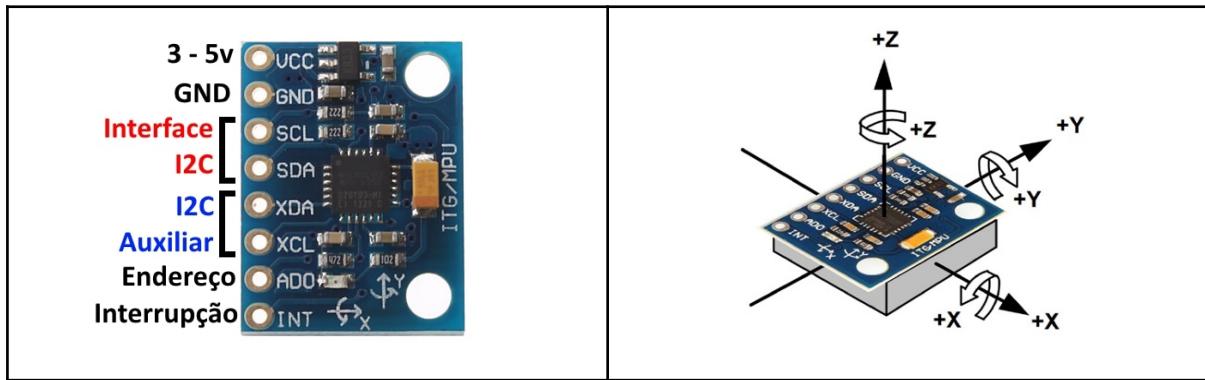
O PCD8544 é o “cérebro” por trás do *display*. Ele gerencia cada *pixel*, as operações de escrita, leitura e as configurações do *display*. Para que o *display* funcione como desejado, é necessário enviar comandos e dados específicos a este controlador através de uma interface serial (geralmente SPI). O repertório completo de comandos e a descrição detalhada de cada um estão disponíveis no [datasheet do PCD8544](#), um documento técnico essencial para qualquer desenvolvedor.

Para encontrar um comando como “limpar a tela” (em inglês *Clear Screen*), deve-se navegar pelo *datasheet* em busca da seção de “**Commands**” ou “**Instruction Set**”. No *datasheet* do PCD8544, encontrariamos comandos que permitem manipular o *display* a nível de *pixel*, como o controle de contraste, inversão e, crucialmente, comandos para apagar o conteúdo da memória de vídeo, que resultam em uma tela limpa. Por exemplo, o comando “Display Control” (0x08-0x0D) permite ligar/desligar o *display*. No entanto, para limpar efetivamente todos os *pixels* (setá-los para “apagado”), precisaríamos de uma rotina que escreva 0x00 em toda a memória de vídeo (DDRAM) do controlador. O comando “Set Y Address” (0x40-0x47) e “Set X Address” (0x80-0x87), combinados com a escrita de dados, permitiriam percorrer e apagar toda a memória do *display*.

Geralmente, bibliotecas prontas para o Nokia 5110 (como as usadas em Arduino) já implementam essas rotinas de baixo nível, oferecendo funções simples como `display.clear()` que executam a sequência de comandos e escritas na DDRAM necessária para limpar a tela.

## Sensor inercial MPU6050

O [MPU6050](#) é um sensor inercial de tecnologia MEMS muito popular, amplamente utilizado em sistemas de controle e navegação, dispositivos móveis, robótica e projetos de eletrônica. Ele integra dois sensores principais: um acelerômetro MEMS de 3 eixos e um giroscópio MEMS de 3 eixos, capazes de medir a aceleração linear e a velocidade angular em torno dos eixos X, Y e Z, respectivamente. Isso permite uma medição precisa de movimentos e inclinações em várias direções, tornando-o ideal para aplicações que exigem detecção de movimento em tempo real.



O acelerômetro, com fundos de escala configuráveis que variam de  $\pm 2g$  a  $\pm 16g$  onde “g” é a aceleração da gravidade terrestre, aproximadamente  $9,81 \text{ m/s}^2$ , é capaz de capturar a aceleração linear em cada eixo, fornecendo dados valiosos sobre inclinação, gravidade e vibrações. A escolha da faixa de fundo de escala define quantos “contagens” correspondem a uma aceleração de 1g. Para converter os valores obtidos do MPU6050, é necessário considerar os valores de fundo de escala configurados para o acelerômetro e para o giroscópio. Esses valores de fundo de escala determinam a sensibilidade do sensor e são especificados em unidades de aceleração e velocidade angular.

A fórmula para converter o valor bruto de 16 bits do acelerômetro (*Valor\_Bruto\_Acc*) em aceleração física (*a*) em metros por segundo ao quadrado ( $\text{m/s}^2$ ) é:

$$a = \frac{\text{Valor_Bruto_Acc}}{2^{15}} \times FS_{Acc} \times 9.8 \frac{\text{m}}{\text{s}^2}$$

Onde:

- *Valor\_Bruto\_Acc* é o valor obtido diretamente do acelerômetro (varia de -32768 a +32767).
- $2^{15}$  é o valor máximo positivo de 16 bits (magnitude = 32768).
- *FS\_Acc* é o fundo de escala do acelerômetro em “g” (2, 4, 8 ou 16).
- $9,8\text{m/s}^2$  é a aceleração da gravidade (1g).

Se, por exemplo, o fundo de escala for  $\pm 2g$ , temos *FS\_Acc* = 2. A fórmula ficaria assim:

$$a = \frac{\text{Valor_Bruto_Acc}}{2^{15}} \times 2 \times 9.8 \frac{\text{m}}{\text{s}^2}$$

Complementando essa funcionalidade, o giroscópio também opera em três eixos e pode medir a velocidade angular com escalas que vão de  $\pm 250$  a  $\pm 2000$  graus por segundo.

O giroscópio do MPU6050 mede a velocidade angular nos eixos X, Y e Z, e também retorna valores numéricos de 16 bits com sinal em complemento de 2. Esses valores devem ser

convertidos para unidades de graus por segundo ( $^{\circ}/s$ ). O fundo de escala do giroscópio pode ser configurado nas seguintes faixas:  $\pm 250^{\circ}/s$ ,  $\pm 500^{\circ}/s$ ,  $\pm 1000^{\circ}/s$  e  $\pm 2000^{\circ}/s$

O fundo de escala escolhido define quantos "contagens" correspondem a uma rotação de  $1^{\circ}/s$ .

A fórmula para converter o valor bruto de 16 bits do giroscópio (**Valor\_Bruto\_Gyro**) em velocidade angular ( $\omega$ ) em graus por segundo ( $^{\circ}/s$ ) é:

$$\omega = \frac{\text{Valor_Bruto_Gyro}}{2^{15}} \times FS_{Gyro}$$

Onde:

- **Valor\_Bruto\_Gyro** é o valor obtido diretamente do giroscópio(varia de -32768 a +32767).
- $2^{15}$  é o valor máximo positivo de 16 bits (magnitude = 32768).
- **FS\_Acc** é o fundo de escala do acelerômetro em “g” (250, 500, 1000, 2000).

Por exemplo, para um fundo de escala de  $\pm 500^{\circ}/s$  (**FS\_Gyro** = 500), a fórmula seria:

$$\omega = \frac{\text{Valor_Bruto_Gyro}}{2^{15}} \times 500^{\circ}/s$$

### **Exemplos:**

- a) Se o valor bruto do acelerômetro no eixo X for 16384 e o fundo de escala configurado for  $\pm 4g$ , a aceleração correspondente seria:

$$a = \frac{16384}{32768} \times 4 \times 9.8 \frac{m}{s^2} = 19,62 \frac{m}{s^2}$$

- b) Se o valor bruto do giroscópio no eixo Z for -16384 e o fundo de escala configurado for  $\pm 1000^{\circ}/s$ , a velocidade angular correspondente seria:

$$\omega = \frac{-16384}{32768} \times 1000^{\circ}/s = -500^{\circ}/s$$

Outro aspecto do MPU6050 é o seu *Digital Motion Processor* (DMP) integrado, que permite o processamento interno dos dados provenientes do acelerômetro e do giroscópio. Esse processador é capaz de realizar cálculos complexos, como a fusão de sensores, utilizando algoritmos avançados, como o filtro de Kalman ou o filtro complementar. Isso resulta em medições mais precisas dos ângulos de inclinação e orientação, como *roll*, *pitch* e *yaw*, sem sobrecarregar o microcontrolador responsável pela operação do sistema. Além disso, o MPU6050 inclui filtros digitais passa-baixa (DLPF) configuráveis, que suavizam ruídos em ambientes de alta vibração ou interferência, melhorando ainda mais a precisão das medições. O sensor também possui um sensor de temperatura embutido, que pode ser utilizado para

compensar variações térmicas que possam influenciar as medições, ou simplesmente para monitoramento térmico, garantindo uma performance otimizada em diversas condições operacionais.

O MPU6050 se destaca pela sua taxa de amostragem, que pode alcançar até 1 kHz, ou seja, ele é capaz de realizar até 1000 medições por segundo. Essa característica garante respostas rápidas e precisas, o que é fundamental em aplicações que exigem alta performance, como monitoramento de movimento e orientação. Além disso, o sensor utiliza uma interface de comunicação I2C, com um endereço padrão de 0x68, facilitando sua integração em plataformas populares como Arduino, Raspberry Pi e ESP32. Para aumentar a flexibilidade na comunicação, algumas versões do módulo, como o MPU6000, também suportam a interface SPI. Outro ponto positivo do MPU6050 é seu baixo consumo de energia, com uma média de apenas 3,9 mA. Isso o torna uma opção eficiente para dispositivos alimentados por bateria, como *wearables* e drones, que precisam operar por longos períodos sem recarga. Além de todas essas características, o *chip* tem um tamanho compacto de apenas 4x4 mm, facilitando sua integração em projetos de *hardware* onde o espaço é limitado.

Devido à sua versatilidade e conjunto abrangente de funcionalidades, o MPU6050 é amplamente utilizado em diversas aplicações. Uma de suas principais utilizações é na estabilização de drones e robôs, onde o sensor desempenha um papel crucial em manter a estabilidade e o equilíbrio, corrigindo rotações e inclinações em tempo real. Além disso, o MPU6050 é empregado em dispositivos de controle de gestos e movimentos, como consoles de videogame e interfaces de realidade aumentada, permitindo interações mais intuitivas e imersivas. Outra aplicação significativa é na navegação inercial, onde o sensor ajuda a rastrear a posição e a direção de um objeto em movimento, tornando-o essencial para sistemas de navegação em ambientes variados. O MPU6050 também é muito popular em plataformas de eletrônica, sendo frequentemente utilizado em projetos com Arduino, Raspberry Pi e ESP32, onde é necessária uma medição precisa de movimento. Por fim, em dispositivos *wearables* e *fitness trackers*, o MPU6050 é utilizado para rastrear atividades físicas e detectar movimentos corporais, contribuindo para um monitoramento eficaz da saúde e do desempenho físico. Essas diversas aplicações evidenciam a importância do MPU6050 em tecnologias modernas, abrangendo desde entretenimento até saúde e navegação.

O *chip* MPU6050 possui um extenso conjunto de registradores internos que controlam desde a sensibilidade do acelerômetro e giroscópio até o gerenciamento de energia e a leitura dos dados de movimento. Para configurar e obter informações do sensor, seu microcontrolador (mestre) se comunica com o MPU6050 (escravo) via I2C, enviando comandos que consistem em escrever ou ler os valores desses registradores. O repertório completo desses comandos e a descrição detalhada de cada registrador, incluindo seus endereços e as funcionalidades de cada *bit*, estão especificados no [datasheet Register Map do MPU-6000/MPU-6050](#).

Para configurar o acelerômetro e o giroscópio no MPU6050, precisaríamos consultar a seção de registradores relevantes no *datasheet*.

1. Configuração do Giroscópio (em inglês, *Gyroscope Configuration*): Na página 14, encontraremos o registrador “GYRO\_CONFIG” mapeado no endereço 0x1B. Neste registrador, encontraremos os *bits* “FS\_SEL” (*bits* 4:3), que permitem selecionar a escala de medição do giroscópio (por exemplo, ±250 dps, ±500 dps, ±1000 dps, ±2000 dps). Para configurar, escreveríamos um valor específico neste registrador para definir a escala desejada.
2. Configuração do Acelerômetro (em inglês, *Accelerometer Configuration*): Na mesma página 14, procure pelo registrador “ACCEL\_CONFIG”. Ele está no endereço 0x1C. Similarmente, os *bits* “AFS\_SEL” (*bits* 4:3) neste registrador são usados para selecionar a escala de medição do acelerômetro (por exemplo, ±2g, ±4g, ±8g, ±16g). Para configurar o acelerômetro, escreveríamos o valor correspondente a esses *bits* no registrador 0x1C.

Programar o MPU6050, portanto, envolve consultar este mapa de registradores para identificar os endereços e os valores binários corretos a serem escritos para alcançar a funcionalidade desejada, e então realizar as operações de leitura/escrita via I2C.

### **Unidade de medição inercial LSM6DSV32X**



O [LSM6DSV32X](#) é fabricado utilizando a tecnologia MEMS como MPU6065. Ele representa um sistema integrado de sensoriamento inercial, combinando em um único encapsulamento compacto um acelerômetro linear triaxial e um giroscópio para detecção de velocidade angular em três eixos ortogonais. Este dispositivo, fabricado pela STMicroelectronics e

amplamente compatível com a plataforma de microcontroladores STM32, oferece uma solução de baixo consumo de energia e alto desempenho para a medição precisa de movimento. Sua arquitetura de seis graus de liberdade permite a captura detalhada das dinâmicas de movimento em diversas aplicações, sendo superior ao MPU6050 em termos de desempenho, eficiência energética e funcionalidades inteligentes embarcadas. Enquanto o MPU6050 ainda é uma excelente opção para aprendizado e projetos simples devido à sua maturidade e comunidade, o LSM6DSV32X representa o que há de mais moderno em IMUs, permitindo aplicações muito mais sofisticadas e eficientes.

O acelerômetro integrado no LSM6DSV32X oferece faixas de escala completa selecionáveis, atingindo até  $\pm 32$  g, onde  $1\text{ g} = 9.8\text{ m/s}^2$ , o que o torna adequado tanto para aplicações de detecção de movimento sutil quanto para aquelas que envolvem acelerações mais intensas. Adicionalmente, dispõe de largura de banda configurável e apresenta um baixo nível de ruído, contribuindo para a qualidade dos dados de aceleração. Recursos como a detecção de eventos específicos, incluindo contagem de passos de uma pessoa (pedômetro), queda livre, despertar e reconhecimento de orientação espacial, ampliam a sua funcionalidade em sistemas inteligentes.

O giroscópio triaxial do LSM6DSV32X complementa o acelerômetro, fornecendo medições de velocidade angular com faixas de escala que podem alcançar  $\pm 4000$  dps (graus por segundo). Similarmente ao acelerômetro, sua largura de banda é configurável, e o dispositivo se destaca pela estabilidade do *bias* e pela sensibilidade apurada. A combinação dessas características garante a obtenção de dados de rotação precisos, essenciais para aplicações que demandam rastreamento de orientação e movimento rotacional.

Para que os dados brutos de 16 bits (Amostra) lidos do acelerômetro e giroscópio sejam convertidos em suas grandezas físicas correspondentes (aceleração em g e velocidade angular em dps, respectivamente), usa-se a sensibilidade linear do acelerômetro e a sensibilidade angular do giroscópio fornecidas pelo fabricante. A **sensibilidade linear do acelerômetro** expressa, em mg/LSB (mili-g por unidade de bit), quanto a saída do sensor (em LSB) representa de aceleração em g (gravidade terrestre). Por exemplo, se a sensibilidade for 0,976 mg/LSB em  $\pm 32$  g, então cada unidade digital (LSB) corresponde a 0,976 mg. A **sensibilidade angular do giroscópio** (em mdps/LSB ou dps/LSB), por sua vez, dá quanto a saída digital representa de velocidade angular em graus por segundo (dps). Por exemplo, se a sensibilidade for 70 mdps/LSB, cada LSB equivale a 0,07 dps.

Para facilitar a integração com sistemas de processamento, o LSM6DSV32X incorpora interfaces de comunicação digital I2C e SPI, oferecendo flexibilidade na escolha da conexão com o microcontrolador hospedeiro. Além disso, a presença da interface MIPI I3C® v1.1 possibilita uma comunicação de alta velocidade e um gerenciamento de energia mais eficiente. A unidade também integra um *buffer* FIFO inteligente de até 4.5 KB, capaz de armazenar dados dos sensores e reduzir a frequência de acesso ao processador principal, otimizando o consumo energético do sistema como um todo.

Um aspecto distintivo do LSM6DSV32X reside em suas capacidades de processamento de dados embarcado. A Unidade de Processamento de Movimento (MPU) com sua *Finite State Machine* (FSM) programável permite o reconhecimento de padrões de movimento complexos com baixo consumo de energia. Adicionalmente, o *Machine Learning Core* (MLC) integrado

possibilita a execução de algoritmos de aprendizado de máquina diretamente no sensor, abrindo caminho para aplicações avançadas de reconhecimento de atividades e detecção de contexto. A inclusão do algoritmo *Sensor Fusion Low Power* (SFLP) para a geração de informações de orientação 3D e o recurso *Adaptive Self-Configuration* (ASC) demonstram o nível de sofisticação e inteligência embarcada neste IMU.

A integração do sensor eletrostático Qvar<sup>TM</sup> para detecção de toques e gestos, um *hub* analógico para conexão de sensores externos e diversas fontes de interrupção programáveis enriquecem ainda mais as funcionalidades deste componente. Os sensores eletrostáticos Qvar<sup>TM</sup> são primariamente sensores de toque e proximidade, baseados na detecção de variações no campo eletrostático. Eles funcionam detectando mudanças na carga elétrica induzidas pela aproximação ou contato de um objeto condutor, como um dedo humano.

Programar o LSM6DSV32X significa, essencialmente, configurar seu controlador interno. É este controlador que gerencia e interpreta todos os sinais do acelerômetro e do giroscópio, transformando dados brutos em informações úteis. Para entender como interagir com ele, precisamos consultar o [datasheet oficial](#). Este documento é o guia completo, contendo o repertório de comandos e os registradores específicos que devemos escrever para configurar o dispositivo.

Para configurar o acelerômetro e o giroscópio, precisaremos navegar até a seção de “Registers” no *datasheet*. Nela, encontraremos tabelas detalhadas que descrevem a função de cada registrador (endereço de memória em que está mapeado) e os valores binários (*bits*) que podemos escrever para definir diferentes parâmetros. Por exemplo, para configurar a escala de medição (*full-scale*) e a taxa de dados de saída (*output data rate - ODR*) do acelerômetro e do giroscópio, devemos procurar por registradores como:

- CTRL1\_XL (endereço 0x10): Este registrador é frequentemente usado para configurar o acelerômetro. No *datasheet*, encontraremos a descrição dos *bits* neste registrador. Por exemplo, os *bits* 7-4 (ODR\_XL) podem controlar a taxa de dados de saída, e os *bits* 3-2 (FS\_XL) podem definir a escala de medição do acelerômetro.
- CTRL2\_G (endereço 0x11): Similarmente, este registrador é usado para configurar o giroscópio. Os *bits* 7-4 (ODR\_G) podem controlar a taxa de dados de saída, e os *bits* 3-0 (FS\_G) podem definir a escala de medição do giroscópio.

Para configurar, por exemplo, o acelerômetro para uma ODR de 104 Hz e uma escala de  $\pm 2g$ , precisaríamos identificar os códigos binários correspondentes no *datasheet* para esses parâmetros nos *bits* apropriados do registrador CTRL1\_XL e então escrever esse *byte* completo para o endereço 0x10 via sua interface de comunicação (I2C ou SPI). O mesmo princípio se aplica ao giroscópio, utilizando o registrador CTRL2\_G e outros registradores relevantes detalhados no *datasheet*.

Em sua essência, programar o LSM6DSV32X resume-se a localizar o registrador (endereço) responsável pela função desejada e, então, definir o código binário correto para os *bits* dentro desse registrador, conforme as especificações detalhadas no *datasheet*. Esse processo é bastante similar ao que já fazemos na programação de microcontroladores.

## STM32H7A3

O STM32H7A3 oferece uma variedade de módulos de comunicação serial, incluindo interfaces UART, I2C, SPI e CAN (do inglês, *Controller Area Network*), o que possibilita uma ampla gama de aplicações em projetos eletrônicos. No [Roteiro 7](#), aprofundamos nossa exploração do módulo UART, enquanto neste Roteiro focaremos nas funções disponibilizadas pelos módulos SPI e I2C.

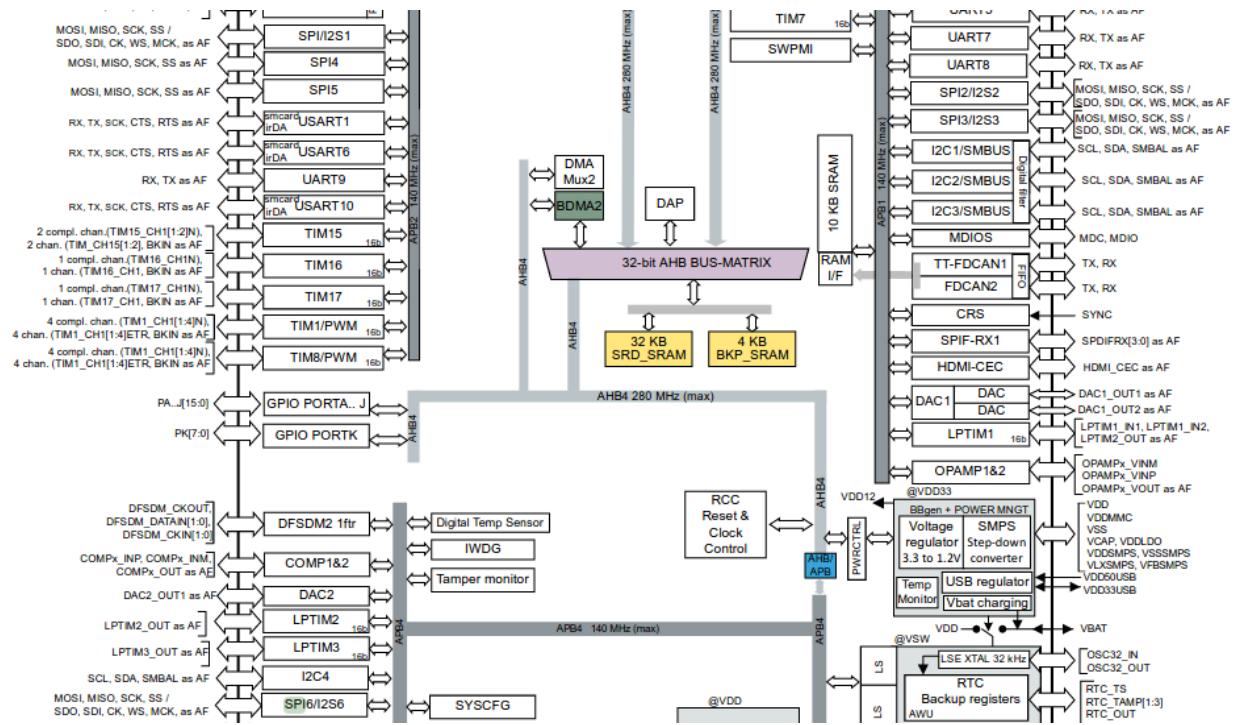
### Módulo SPI

O microcontrolador STM32H7A3 é equipado com 6 módulos SPI que facilitam a comunicação serial periférica:

- SPI1, SPI2, SPI3: Esses módulos são geralmente utilizados para comunicação serial de uso geral, adequados para diversas aplicações.
- SPI4, SPI5: Oferecem flexibilidade adicional, permitindo a operação com um *clock* distinto do barramento de interface.
- SPI6: Este módulo também pode operar com um *clock* independente e apresenta um *bit* de ativação adicional, o SPI6AMEN, que controla o *clock* do DFSDM2.

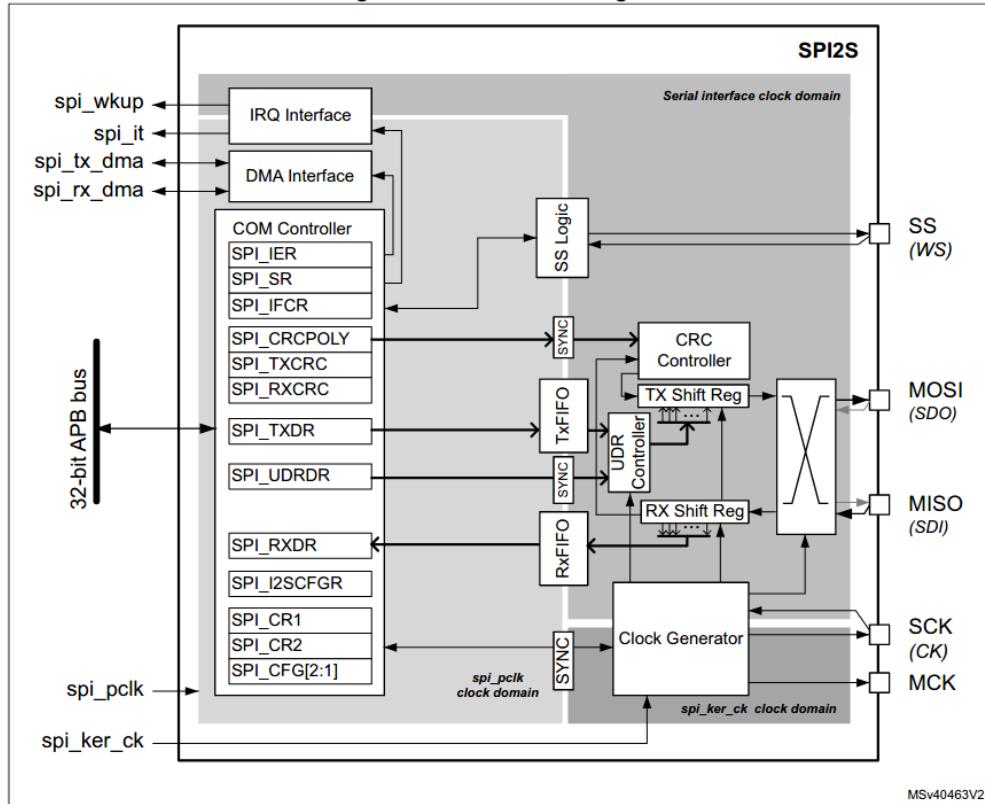
Esses módulos utilizam um mecanismo conhecido como *clock gating* para gerenciar a ativação dos sinais de relógio, contribuindo para a otimização do consumo de energia.

Para ativar o *clock* de um módulo SPI, é necessário configurar os *bits* [RCC\\_APB2ENR\\_SPI1EN](#), [RCC\\_APB1LENR\\_SPI2EN](#), [RCC\\_APB1LENR\\_SPI3EN](#), [RCC\\_APB2ENR\\_SPI4EN](#), [RCC\\_APB2ENR\\_SPI5EN](#) e [RCC\\_APB4ENR\\_SPI6EN](#), conforme o módulo SPI em questão. Quando o periférico não é mais necessário, o *clock* pode ser desativado desabilitando o *bit* SPIxEN correspondente. Os *bits* [RCC\\_CDCCIP1R\\_SPI123SEL](#), [RCC\\_CDCCIP1R\\_SPI45SEL](#) e [RCC\\_SRDCCIPR\\_SPI6SEL](#) permitem escolher entre várias fontes de *clock*, como PLLs, HSI, CSI, HSE ou um *clock* externo (I2S\_CKIN) para o módulo SPI correspondente. É necessário ativar o *clock* do barramento de interface AHB4 através dos *bits* GPIOxEN correspondentes ao GPIO utilizado pelo módulo SPI. Por exemplo, se o SPI1 utiliza pinos do GPIOA, o *bit* [RCC\\_AHB4ENR\\_GPIOAEN](#) deve ser habilitado.



A comunicação SPI envolve diferentes domínios de *clock*, incluindo o *clock kernel* (`spi_ker_ck`), o *clock* do barramento de interface (32-bit APB bus) e o *clock* serial do periférico (`spi_pclk`). É fundamental garantir a sincronização adequada entre esses *clocks* para evitar problemas de comunicação.

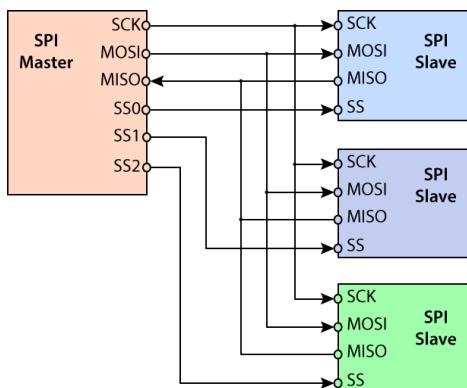
Figure 583. SPI2S block diagram



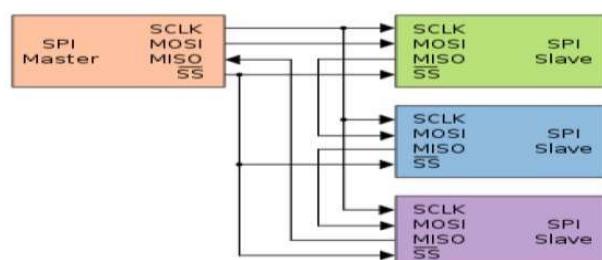
O protocolo SPI utiliza quatro sinais principais: o MOSI (*Master Out Slave In*), que é utilizado pelo mestre para enviar dados ao escravo; o MISO (*Master In Slave Out*), que permite ao escravo enviar dados de volta ao mestre; o SCK (*Serial Clock*), um sinal de *clock* gerado pelo mestre que sincroniza a transferência de dados; e o SS (*Slave Select*), que serve para selecionar qual escravo o mestre deseja comunicar.

Quando a comunicação é no **modo mestre e escravo**, um único mestre se comunica com um único escravo, iniciando a comunicação e controlando o sinal de *clock* e o sinal de seleção do escravo. O escravo responde aos comandos do mestre e transfere dados conforme o *clock* fornecido. Este modo é determinado pela configuração do registrador [GPIOx\\_MODER](#) para definir os pinos de saídas para o mestre (MOSI, SCK, SS) e de entrada para o escravo (MISO) com a função alternativa (AF). Em seguida, os registradores [GPIOx\\_AFRL](#) e [GPIOx\\_AFRH](#) devem ser configurados para selecionar a função alternativa SPI correspondente aos pinos GPIO utilizados. No que diz respeito ao *software*, **o mestre é responsável por gerenciar o sinal SS, gerar o clock (SCK) e iniciar a transferência de dados**.

Quando se fala em **comunicação multi-escravos**, um único mestre pode se comunicar com vários escravos, cada um possuindo um sinal SS dedicado. O mestre ativa o sinal SS correspondente para selecionar qual escravo deseja comunicar. Existem duas principais topologias para essa comunicação: a **topologia em estrela**, onde cada escravo tem uma linha SS dedicada conectada a um pino GPIO, SS0, SS1 ou SS2, controlado pelo mestre, e a **topologia em cadeia Daisy (Daisy chain)**, na qual os escravos são conectados em série, compartilhando os sinais SCK e MOSI, com cada um tendo um sinal SS dedicado ao escravo anterior na cadeia. A topologia em estrela é simples de implementar, mas requer um número maior de pinos GPIO para o mestre comunicar com escravos, enquanto a topologia em cadeia reduz o número de pinos GPIO necessários, mas exige um gerenciamento mais complexo do fluxo de dados.



Topologia em estrela



Topologia em cadeia Daisy

A configuração do SPI é realizada principalmente por meio de quatro registradores essenciais: [SPI\\_CR1](#), [SPI\\_CR2](#), [SPI\\_CFG1](#) e [SPI\\_CFG2](#). O registrador SPI\_CR1 é responsável por controlar a habilitação do módulo (SPE), a polaridade do *clock* (CPOL), a fase do *clock* (CPHA) e outros parâmetros relevantes. Por sua vez, o SPI\_CR2 permite definir a quantidade de dados

(TSIZE) a serem transferidos em uma operação do registrador [SPI\\_TXDR](#) ou para o registrador [SPI\\_RXDR](#) e o número de transições (TSER) dessa transação. O registrador SPI\_CFG1 é fundamental para definir a taxa de transmissão (MBR), o número de *bits* por quadro de dados (DSIZE), além de controlar a habilitação do código de verificação de redundância cíclica (CRCEN) e o gerenciamento de DMA (TXDMAEN e RXDMAEN). Já o SPI\_CFG2 especifica se o SPI opera em modo mestre ou escravo (MASTER), se o periférico mantém o controle dos pinos GPIO associados mesmo quando o módulo SPI está desabilitado (AFCNTR) e permite a seleção do protocolo serial entre Motorola e TI (SP). Durante a configuração, é crucial garantir que o bit [SPI\\_CR1\\_SPE](#) esteja resetado para 0, assegurando que o módulo não esteja habilitado antes que todas as configurações necessárias sejam realizadas.

O *baud rate* do módulo SPI é determinado pela frequência do *clock kernel* do módulo SPI no modo mestre e o divisor configurado em [SPI\\_CFG1\\_MBR](#):

$$\text{baud\_rate} = \text{spi\_ker\_ck}/2^{\text{SPI\_CFG1\_MBR}+1}$$

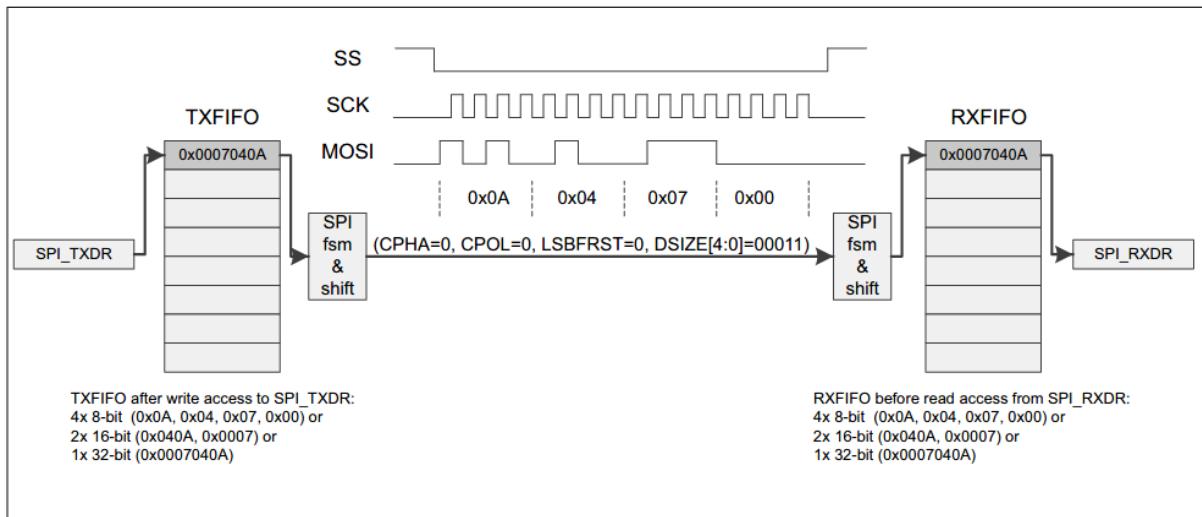
Quanto ao modo de comunicação, o SPI opera predominantemente em modo *full-duplex*, onde os dados são transmitidos e recebidos simultaneamente. Entretanto, também é possível configurar a comunicação em modo *simplex*, onde a transmissão ou recepção ocorre em apenas uma direção; isso pode ser feito através de um campo específico SPI\_CFG2\_COMM do registrador [SPI\\_CFG2](#). Além disso, este mesmo registrador é responsável pela configuração do formato dos dados SPI, que inclui a polaridade (SPI\_CFG2\_CPOL) e a fase (SPI\_CFG2\_CPHA) do *clock*. O modo *half-duplex*, que alterna entre transmissão e recepção, não é diretamente suportado pelo *hardware* SPI, mas pode ser implementado em *software*, gerenciando a direção da transferência através do bit [SPI\\_CR1\\_HDDIR](#) e os pinos de comunicação. A correta configuração desses parâmetros é fundamental para assegurar a sincronização entre o mestre e o escravo durante a comunicação SPI. Se a polaridade e a fase do *clock* não forem configuradas adequadamente, pode haver amostragem ou transmissão de dados em momentos inadequados, resultando na corrupção das informações transmitidas. Portanto, uma atenção cuidadosa a esses detalhes é crucial para o sucesso da comunicação.

Para informações detalhadas sobre a configuração do SPI, consulte o Manual de Referência. As seções relevantes incluem: configuração do SPI, que é semelhante para os modos mestre e escravo ([Seção 55.4.9](#)); habilitação do SPI ([Seção 55.4.10](#)); transmissão e recepção de dados ([Seção 55.4.11](#)); e desabilitação do SPI ([Seção 55.4.12](#)).

Para otimizar a transferência de dados nos módulos SPI, tanto a estrutura de dados FIFO (do inglês, *First-In, First-Out*) quanto a DMA (*Direct Memory Access*) podem ser usados junto com o *hardware* de serialização dos *bytes* em conformidade com o protocolo SPI implementado nos módulos SPI. O módulo SPI possui duas **FIFOs internas**, uma no receptor e outra no transmissor. Essas FIFOs atuam *buffers* temporários, armazenando dados para garantir um fluxo de transferência mais contínuo. O campo [SPI\\_CFG1\\_FTHLV](#) permite configurar a quantidade de quadros de dados por pacote. É importante ressaltar a restrição quanto ao tamanho do pacote de dados, que não pode ultrapassar metade do espaço disponível na FIFO. Essa limitação é crucial para assegurar que haja espaço suficiente para acomodar os dados

recebidos enquanto um pacote está sendo transmitido, prevenindo situações de *overflow* que poderiam comprometer a integridade da transferência. Além disso, a eficiência da interface SPI é otimizada quando os tamanhos dos pacotes configurados estão alinhados com o paralelismo de acesso aos registradores de dados SPI. Isso significa que, para maximizar a eficácia da transferência, o tamanho do pacote deve ser ajustado em conformidade com a largura de dados do registrador SPI. Esse alinhamento não apenas melhora a performance geral, mas também assegura um fluxo contínuo e eficiente de dados durante as operações de comunicação.

**Figure 595. Packing data in FIFO for transmission and reception**



- DSIZE[3:0] is configured to 4-bit, data is right aligned, valid bits are performed only on the bus, their order depends on LSBFRST, if it is set, the order is reversed at all the data frames.

Mostramos no [Roteiro 9](#) que o DMA é uma funcionalidade que permite a transferência de dados entre periféricos e memória, ou entre diferentes áreas da memória, sem a necessidade de intervenção da CPU. Essa abordagem libera a CPU para realizar outras tarefas durante as transferências de dados, resultando em um aumento significativo da eficiência do sistema. A configuração do DMA para operar com periféricos envolve algumas etapas cruciais. Primeiramente, é necessário habilitar o *clock* do controlador DMA desejado, utilizando os bits [RCC\\_AHB1ENR\\_DMA1EN](#) ou [RCC\\_AHB1ENR\\_DMA2EN](#) nos registradores. Em seguida, recomenda-se resetar o controlador DMA, acionando os bits [RCC\\_AHB1RSTR\\_DMA1RST](#) ou [RCC\\_AHB1RSTR\\_DMA2RST](#) no registrador [RCC\\_AHB1RSTR](#), e o periférico SPI, setando o bit [SPIxRST](#) do registrador [RCC\\_APB1LRSTR](#), [RCC\\_APB2RSTR](#) ou [RCC\\_APB4RSTR](#), garantindo que os módulos estejam prontos para configuração. Uma vez realizado o *reset*, a próxima etapa é configurar o canal DMA, utilizando registradores específicos, como [DMA\\_SxCR](#), [DMA\\_SxNDTR](#), [DMA\\_SxPAR](#) e [DMA\\_SxM0AR](#). Esses registradores permitem definir a direção da transferência, o número de dados a serem transferidos, além dos endereços de origem e destino, juntamente com outras opções relevantes. Deve-se, então, selecionar o canal DMAMUX que corresponde ao canal DMA configurado. Em seguida, é importante configurar o registrador de configuração do canal, o [DMAMUX\\_CxCR](#). Nesse registrador, é necessário definir o [DMAREQ\\_ID](#), que especifica a fonte de requisição do DMA. Para isso, é aconselhável consultar [a tabela de mapeamento do DMAMUX](#), onde se pode identificar o código correspondente ao periférico SPI desejado, seja para transmissão (SPIx\_TX) ou recepção (SPIx\_RX).

**Table 101. DMAMUX1: assignment of multiplexer inputs to resources (continued)**

DMA request MUX input	Resource	DMA request MUX input	Resource	DMA request MUX input	Resource
37	SPI1_RX	79	UART7_RX	121	Reserved
38	SPI1_TX	80	UART7_TX	122	Reserved
39	SPI2_RX	81	UART8_RX	123	Reserved
40	SPI2_TX	82	UART8_TX	124	Reserved
41	USART1_RX	83	SPI4_RX	125	Reserved
42	USART1_TX	84	SPI4_TX	127	Reserved

**Table 104. DMAMUX2: assignment of multiplexer inputs to resources (continued)**

DMA request MUX input	Resource	DMA request MUX input	Resource
10	LP UART1_TX	26	Reserved
11	SPI6_RX	27	Reserved
12	SPI6_TX	28	Reserved
13	I2C4_RX	29	Reserved
14	I2C4_TX	30	Reserved
15	Reserved	31	Reserved
16	Reserved	-	-

DMA request MUX input	Resource	DMA request MUX input	Resource
43	USART2_RX	85	SPI5_RX
44	USART2_TX	86	SPI5_TX
45	USART3_RX	87	SAI1_A
46	USART3_TX	88	SAI1_B
47	TIM8_CH1	89	SAI2_A
48	TIM8_CH2	90	SAI2_B
49	TIM8_CH3	91	SWPPI_RX
50	TIM8_CH4	92	SWPPI_TX
51	TIM8_UP	93	SPDIFRX_DT
52	TIM8_TRIG	94	SPDIFRX_CS
53	TIM8_COM	95	Reserved
54	Reserved	96	Reserved
55	TIM5_CH1	97	Reserved
56	TIM5_CH2	98	Reserved
57	TIM5_CH3	99	Reserved
58	TIM5_CH4	100	Reserved
59	TIM5_UP	101	DFSDM1_dma0
60	TIM5_TRIG	102	DFSDM1_dma1
61	SPI3_RX	103	DFSDM1_dma2
62	SPI3_TX	104	DFSDM1_dma3

Após realizar essas configurações, o próximo passo é habilitar o canal DMA. Isso é feito configurando o *bit* DMA\_SxCR\_EN no registrador [DMA\\_SxCR](#), ativando assim o canal DMA e permitindo que ele inicie as transferências de dados conforme configurado.

O módulo SPI possui suporte para **CRC** (do inglês *Cyclic Redundancy Check*), uma importante funcionalidade detetora de erros que aumenta a integridade dos dados transmitidos. O CRC é um código de detecção de erros que assegura a confiabilidade das informações em diversos protocolos de comunicação, incluindo o SPI. Ao transmitir dados, um valor de CRC é calculado matematicamente com base nos dados originais e anexado à transmissão. Quando os dados chegam ao receptor, este recalcula o CRC utilizando os dados recebidos e o compara com o valor de CRC que foi enviado. Se ambos os valores coincidirem, é altamente provável que os dados tenham sido transmitidos sem erros.

Para configurar o módulo SPI para utilizar o CRC, alguns registradores e *bits* específicos são utilizados. O registrador [SPI\\_CFG1](#) contém o *bit* SPI\_CFG1\_CRCEN, que deve ser habilitado para ativar o cálculo e a verificação do CRC. Além disso, o campo SPI\_CFG1\_CRCSIZE neste registrador permite definir o tamanho do polinômio CRC, que pode ser de 16 *bits* (padrão) ou 32 *bits*. O registrador [SPI\\_CRCPOLY](#) é onde se armazena o valor do polinômio CRC a ser usado, enquanto o [SPI\\_TXCRC](#) contém o valor CRC calculado para os dados que estão sendo transmitidos, e o [SPI\\_RXCRC](#) armazena o valor CRC recebido juntamente com os dados.

O procedimento para utilizar o CRC no SPI é bastante direto. Primeiro, é necessário habilitar o CRC, configurando o *bit* SPI\_CFG1\_CRCEN no registrador SPI\_CFG1. Em seguida, o polinômio CRC desejado deve ser escrito no registrador SPI\_CRCPOLY. Os dados podem então ser transmitidos normalmente através do registrador de dados SPI. Durante a transmissão, o módulo SPI calcula automaticamente o CRC, que é armazenado em SPI\_TXCRC, e o anexa aos dados. Ao receber os dados, o módulo SPI também recebe o CRC, que é armazenado no registrador SPI\_RXCRC. Por fim, o receptor recalcula o CRC com base nos dados recebidos e compara esse valor ao que está armazenado em SPI\_RXCRC. Se os dois valores coincidirem, a transmissão é considerada íntegra; caso contrário, um erro de CRC é detectado, garantindo assim a confiabilidade da comunicação.

Por fim, o gerenciamento das interrupções é uma responsabilidade do *software*, que deve configurar, habilitar e tratar as interrupções geradas pelos periféricos SPI. Para isso, é necessário implementar rotinas de serviço de interrupção (ISRs) para as IRQs, que executarão as ações apropriadas em resposta a cada evento. Os periféricos SPI oferecem a possibilidade de habilitar interrupções para uma variedade de situações, utilizando os *bits* de controle presentes no registrador SPI\_IER. Entre os eventos que podem gerar interrupções, destacam-se a conclusão de uma transferência de dados (EOTIE), a detecção de erros de comunicação, como os erros de CRC (CRCEIE) e erros de quadro de dados (TIFREIE), e eventos relacionados à FIFO. Por exemplo, interrupções podem ser geradas quando a FIFO está cheia (OVRIE) ou vazia (UDRIE), atingiu-se o limiar de TXFIFO (TXPIE) ou o limiar de RXFIFO (RXPIE). Além disso, o *software* tem a capacidade de consultar quais interrupções ocorreram por meio do registrador de estado SPI\_SR. Após o reconhecimento dos eventos, as *flags* podem ser limpadas utilizando o registrador SPI\_IFCR. Esse processo garante que o sistema esteja sempre pronto para detectar e responder a novos eventos de forma eficaz, mantendo a comunicação SPI robusta e eficiente.

spi3_it	58	51	SPI3	SPI3 global interrupt	0x0000 010C
exti_spi3_wkup					

spi1_it	42	35	SPI1	SPI1 global interrupt	0x0000 00CC
exti_spi1_it					
spi2_it	43	36	SPI2	SPI2 global interrupt	0x0000 00D0
exti_spi2_it					

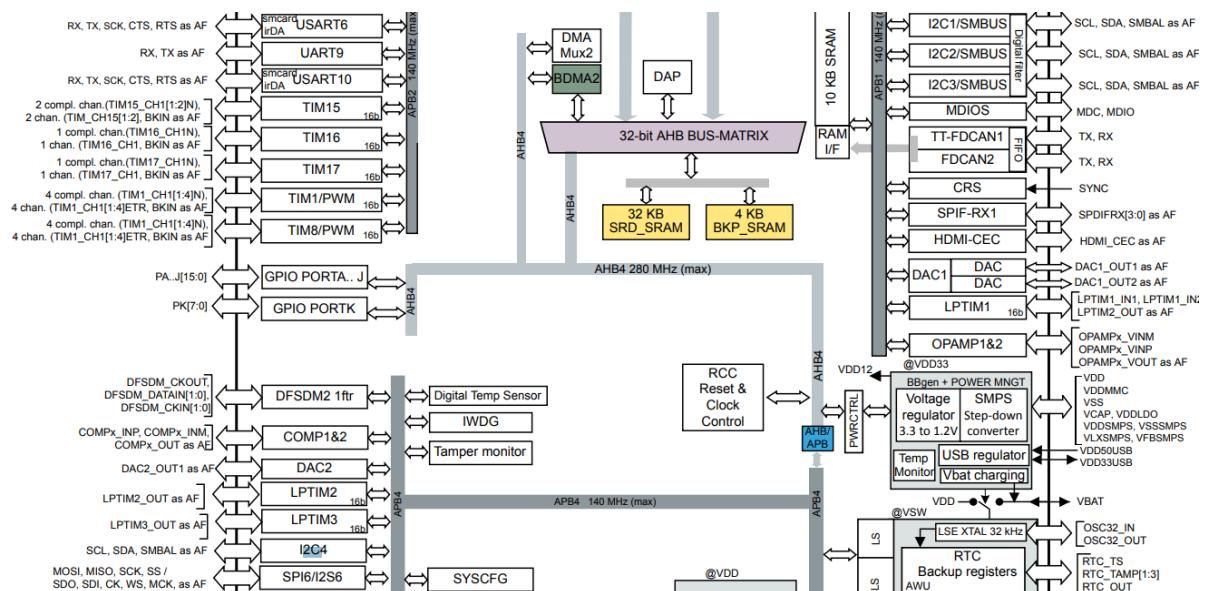
spi4_it	91	84	SPI4	SPI4 global interrupt	0x0000 0190
exti_spi4_wkup					
spi5_it	92	85	SPI5	SPI5 global interrupt	0x0000 0194
exti_spi5_wkup					
spi6_it	93	86	SPI6	SPI6 global interrupt	0x0000 0198
exti_spi6_wkup					

No Manual de Referência são apresentados [2 exemplos de aplicação](#) que envolvem o uso do DMA.

## Módulo I2C

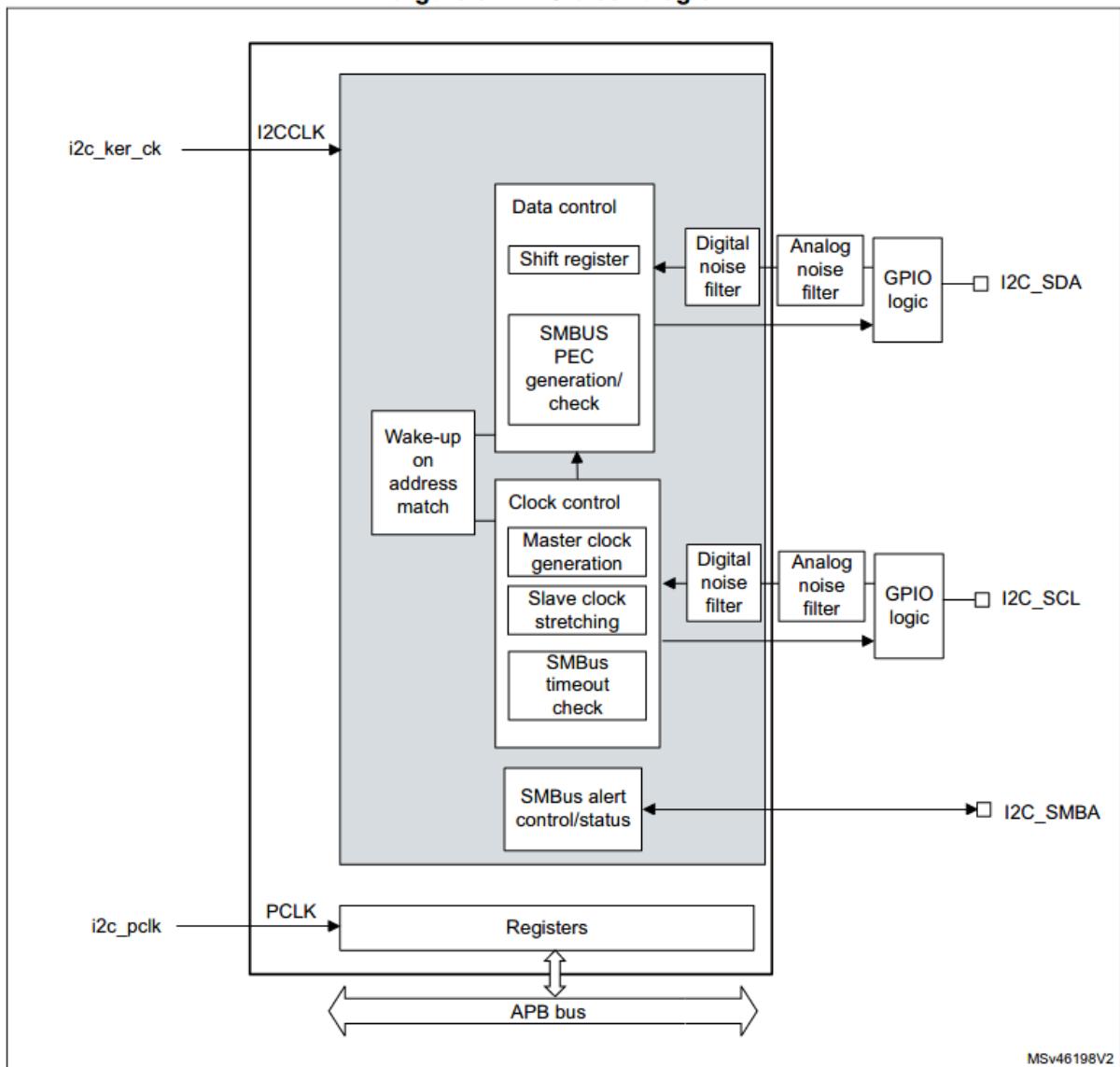
O STM32H7A2 suporta [quatro periféricos I2C](#): I2C1, I2C2, I2C3 e I2C4. Os recursos disponíveis nesses periféricos são compatíveis com a especificação SMBus versão 3.0, que é um protocolo de comunicação usado em dispositivos que requerem monitoramento e controle, como sensores e dispositivos de gerenciamento de energia.

A ativação do *clock gating* para os periféricos I2C envolve uma série de configurações nos registradores RCC (do inglês, *Reset and Clock Control*). O procedimento para habilitar esses periféricos pode ser resumido nas seguintes etapas. Primeiramente, é necessário habilitar o *clock* do periférico I2C (`i2c_pcclk`) desejado. Isso é feito configurando o *bit* correspondente no registrador [`RCC\_APB1LENR1`](#) para I2C1, I2C2 ou I2C3, ou utilizando o *bit* [`RCC\_APB4ENR\_I2C4`](#) para o I2C4. Após habilitar o *clock*, é importante garantir que ele permaneça ativo durante o modo de suspensão (CSleep). Para isso, devem ser configurados os *bits* apropriados nos registradores [`RCC\_APB1LLPENR`](#) e [`RCC\_APB4LPENR`](#). Outra etapa crucial é a configuração do *clock kernel* (`i2c_ker_ck`). Os periféricos I2C requerem um *clock* rápido o suficiente para atingir a taxa de transmissão desejada. Cada periférico I2C possui um conjunto específico de *bits* nos registradores RCC que controla sua fonte de *clock*. Para selecionar a fonte do *clock kernel* para o I2C1, I2C2 e I2C3, utiliza-se o campo [`RCC\_CDCCIP2R\_I2C123SEL`](#), enquanto para o I2C4, a seleção é feita pelos *bits* no campo [`RCC\_SRDCCIPR\_I2C4SEL`](#). Esses *bits* permitem escolher entre várias fontes, como RCC, PLLs, HSI e CSI, garantindo que a configuração atenda aos requisitos de desempenho e consumo de energia da aplicação. Por fim, o periférico I2C4 possui um modo autônomo que permite que continue operando mesmo quando a CPU está em modo de baixo consumo, como CStop. Para habilitar esse modo autônomo, deve-se setar o *bit* [`RCC\_SRDAMR\_I2C4AMEN`](#) em ‘1’. No modo autônomo, o periférico é capaz de gerar uma requisição de *clock kernel* quando necessário, otimizando assim o consumo de energia.



Para garantir transmissões no protocolo I2C, deve-se ativar o *clock gating* dos pinos GPIO multiplexáveis, configurando-os adequadamente para o modo alternativo. A definição correta da função alternativa nos registradores [GPIOx\\_AFRL](#) e [GPIOx\\_AFRH](#) permite estabelecer a conexão adequada entre os pinos e os periféricos I2C. Na [Figura 511 do Manual de Referência](#), é possível observar que os pinos I2C\_SDA e I2C\_SCL estão conectados a um circuito de lógica GPIO, que oferece opções para a lógica de saída dos pinos configurados. É importante configurar ambos os pinos como “*open drain*” usando registradores [GPIOx\\_OTYPER](#) e ter resistores de *pull-up* ativados para SDA e SCL, externamente ou por meio da configuração dos registradores [GPIOx\\_PUPDR](#). Essa configuração é necessária devido à natureza *multi-master* do protocolo I2C, que permite que vários dispositivos compartilhem o mesmo barramento. Com a configuração *open drain*, apenas um dispositivo pode conduzir a linha para baixo de cada vez, enquanto os demais permanecem em alta impedância, deixando a linha flutuante. Isso evita conflitos na condução da linha e possíveis curtos-circuitos que poderiam danificar os dispositivos, caso pinos configurados como *push-pull* tentassem conduzir a linha para níveis lógicos diferentes. Dessa forma, a comunicação no barramento I2C se torna mais confiável e segura.

**Figure 511. I2C block diagram**



As [tabelas de funções alternativas](#) dos pinos fornecem uma lista detalhada dos pinos GPIO disponíveis para cada função de periférico. Com base nessas tabelas, é possível selecionar os pinos multiplexáveis para as linhas SDA e SCL em cada periférico I2C. Por exemplo, o par de pinos ([PB8, PB9](#)) pode ser configurado para a função alternativa 4 (AF4), que corresponde a I2C1\_SCL e I2C1\_SDA. Da mesma forma, o par ([PB10, PB11](#)) também pode ser configurado para a função alternativa 4 (AF4), correspondente a I2C2\_SCL e I2C2\_SDA. Além disso, é importante destacar que o par (PB8, PB9) pode ser multiplexado para as funções I2C4\_SCL e I2C4\_SDA ao definir a função alternativa como 6 (AF6). Essa flexibilidade na configuração

O procedimento básico para garantir uma comunicação I2C confiável e eficiente envolve várias etapas cruciais. O primeiro passo é habilitar o *clock* do periférico I2C desejado, bem como o *clock* do módulo GPIO responsável pelos pinos do periférico I2C. Isso deve ser feito utilizando os registradores RCC apropriados, garantindo que ambos os componentes estejam prontos para operar de forma integrada. Em seguida, recomenda-se realizar um *reset* de *software* no periférico I2C. Isso é importante para assegurar que o dispositivo inicie em um estado conhecido. O *reset* pode ser efetuado definindo o *bit* de *reset* correspondente no registrador [RCC\\_APB1LRSTR](#) para os módulos I2C1, I2C2 e I2C3 e [RCC\\_APB4RSTR](#) para I2C4. A configuração adequada dos pinos GPIO utilizados na comunicação I2C, como SDA e SCL, é outro passo crítico. Para isso, é necessário definir a função alternativa desses pinos utilizando os registradores [GPIOx\\_AFRL](#) ou [GPIOx\\_AFRH](#) e configurar os pinos em modo *open-drain* com resistores de *pull-up* ativados.

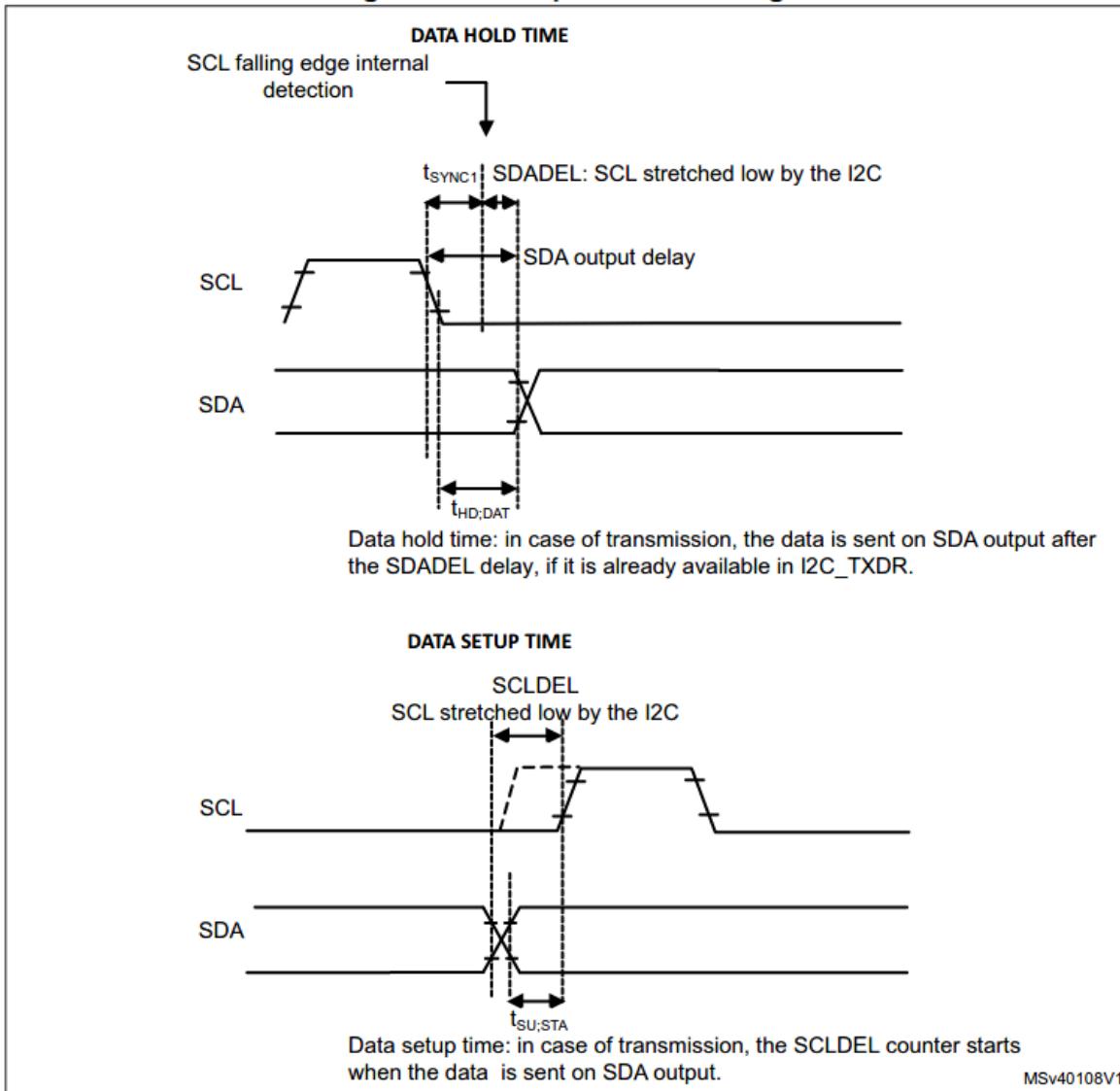
Após a configuração dos pinos, é necessário ajustar os parâmetros de comunicação I2C utilizando dois registradores de configuração: [I2C\\_CR1](#) e [I2C\\_CR2](#). O registrador I2C\_CR1 permite habilitar diversas funcionalidades, como o filtro analógico (ANFOFF), o filtro digital para os pinos SCL e SDA (DNF), e a configuração do modo DMA (RXDMAEN, TXDMAEN). Por sua vez, o registrador I2C\_CR2 é responsável por configurar o modo de endereçamento, que pode ser de 7 *bits* ou 10 *bits* (ADD10). O I2C\_CR2 permite definir a quantidade de *bytes* a serem transferidos (NBYTES), gerar condições de início (START) e fim (STOP), determinar o sentido da transferência (RD\_WRN) e especificar o endereço do escravo (SADD). [Por padrão, o periférico I2C opera em modo escravo.](#) Muda-se automaticamente de escravo para mestre quando gera uma condição de START e de mestre para escravo se ocorrer uma perda de arbitragem ou a geração de um STOP.

Os registradores [I2C\\_OAR1](#) e [I2C\\_OAR2](#) são essenciais na configuração dos endereços de [dispositivos que operam no modo escravo](#) do barramento I2C. Os endereços definidos nesses registradores são utilizados para identificar o periférico escravo durante a comunicação. Quando um mestre I2C inicia uma transmissão, ele envia o endereço do escravo desejado, configurado no campo I2C\_CR2\_SADD. Se o *bit* I2C\_OAR1\_OA1EN estiver definido como “1”, o escravo compara o endereço recebido com o endereço configurado no campo I2C\_OAR1\_OA1, adotando o modo de endereçamento (7 ou 10 *bits*) especificado no *bit* I2C\_OAR1\_OA1MODE. Se houver correspondência, o escravo responde, estabelecendo a comunicação. Caso contrário, se endereços adicionais estiverem configurados (I2C\_OAR1\_OA2EN = 1), o periférico escravo poderá responder ao endereço definido em I2C\_OAR2\_OA2 e as até sete variantes geradas pela máscara definida no campo

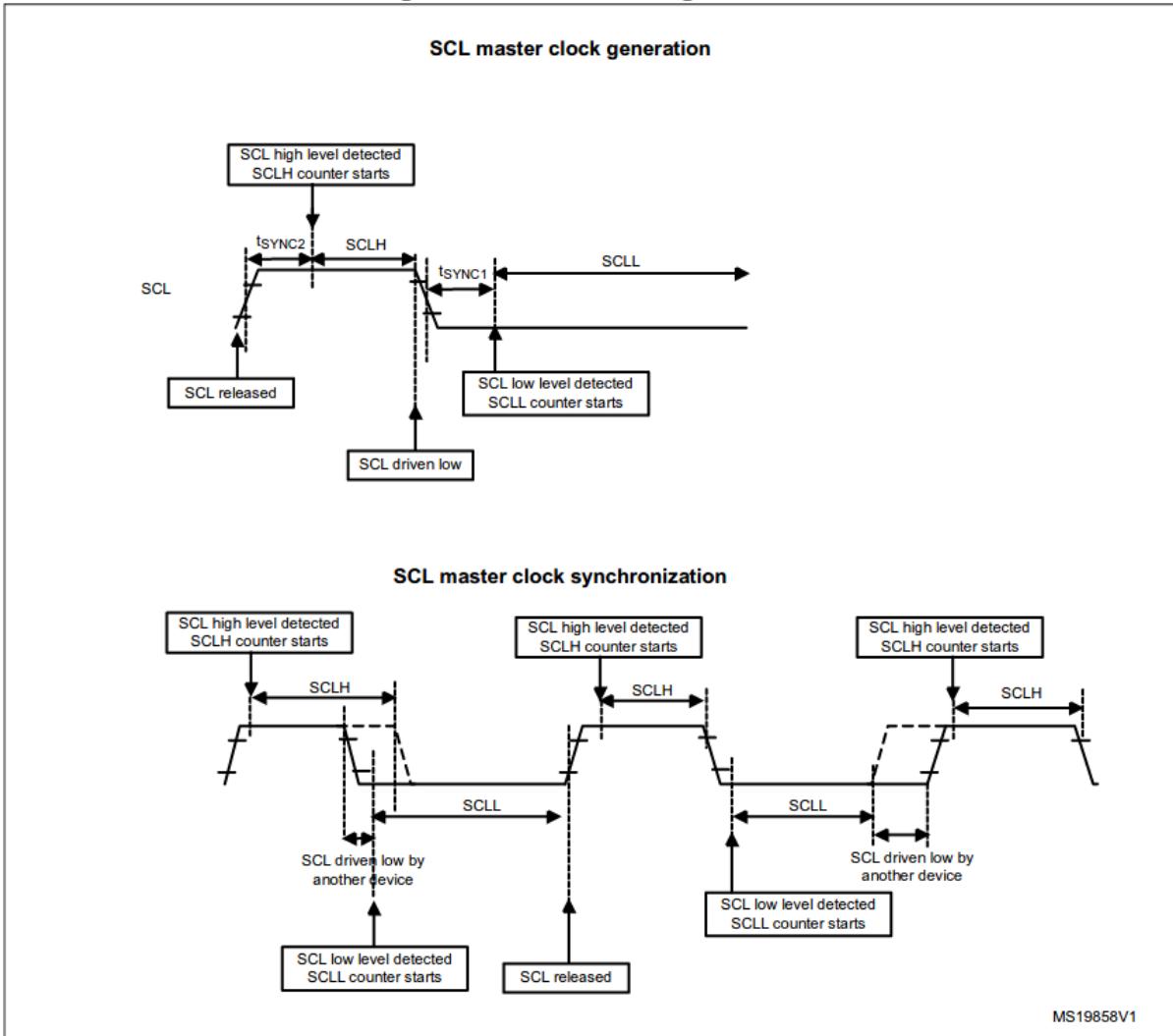
`I2C_OAR2_OA2MSK`, possibilitando a comunicação. A correta configuração dos endereços nos registradores `I2C_OAR1` e `I2C_OAR2` é fundamental para o funcionamento adequado da comunicação I2C. Um endereço configurado incorretamente impede que o escravo responda ao mestre, impossibilitando a troca de dados.

O registrador `I2C_TIMINGR` é usado para configurar a temporização do I2C, com campos que especificam tempos de *setup* (`SCLDEL`) e *hold* (`SDADEL`) de dados, largura de nível alto de SCL (`SCLH`) largura de nível baixo de SCL (`SCLL`), e um prescaler (`PRESC`). Esses tempos são ilustrados nos diagramas de tempo a seguir.

**Figure 513. Setup and hold timings**



**Figure 524. Master clock generation**



A partir da frequência do *clock kernel*, `i2c_ker_ck` e dos parâmetros configurados no registrador `I2C_TIMINGR`, podemos derivar as seguintes métricas de tempo:

$$t_{PRESC} = \frac{PRESC+1}{i2c\_ker\_ck} = (PRESC+1) \times t_{I2CCLK}$$

$$t_{SCLDEL} = (SCLDEL+1) \times t_{PRESC}$$

$$t_{SDADEL} = SDADEL \times t_{PRESC}$$

$$t_{SCLH} = (SCLH+1) \times t_{PRESC}$$

$$t_{SCLL} = (SCLL+1) \times t_{PRESC}$$

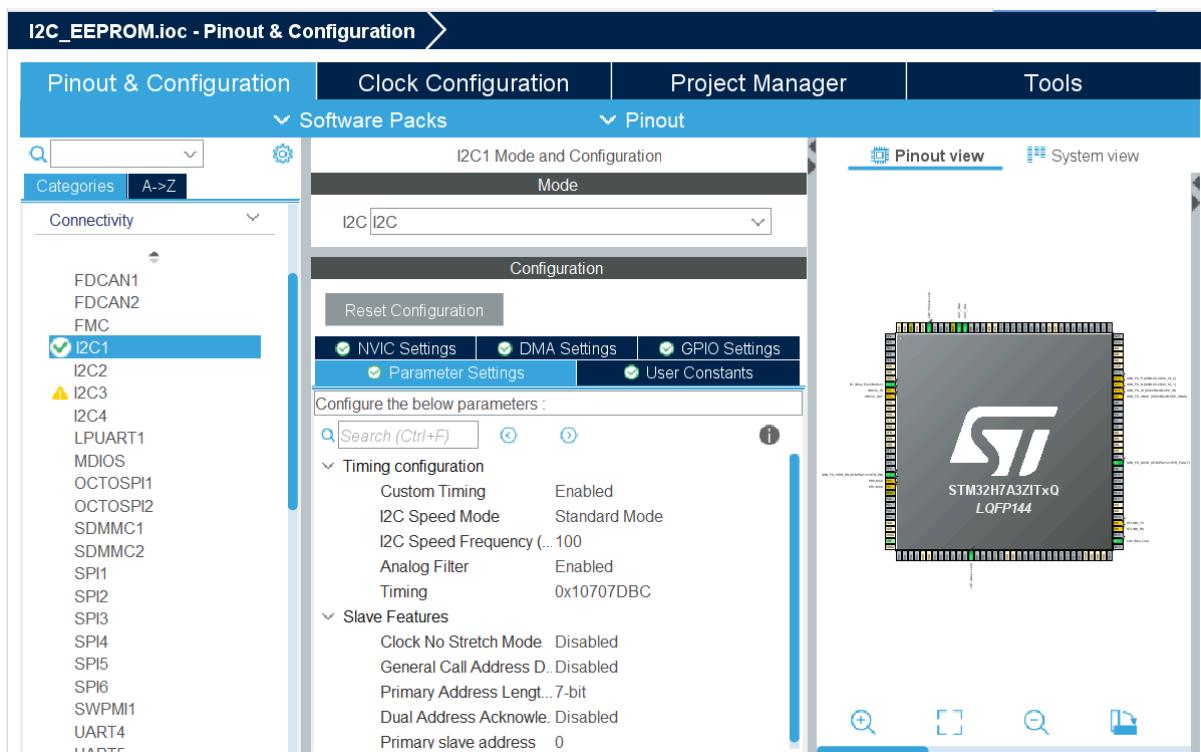
Essas medidas devem satisfazer as restrições temporais estabelecidas pelo protocolo I2C e SMBus.

**Table 376. I<sup>2</sup>C-SMBus specification clock timings**

Symbol	Parameter	Standard-mode (Sm)		Fast-mode (Fm)		Fast-mode Plus (Fm+)		SMBus		Unit
		Min	Max	Min	Max	Min	Max	Min	Max	
f <sub>SCL</sub>	SCL clock frequency	-	100	-	400	-	1000	-	100	kHz
t <sub>HD:STA</sub>	Hold time (repeated) START condition	4.0	-	0.6	-	0.26	-	4.0	-	μs
t <sub>SU:STA</sub>	Set-up time for a repeated START condition	4.7	-	0.6	-	0.26	-	4.7	-	
t <sub>SU:STO</sub>	Set-up time for STOP condition	4.0	-	0.6	-	0.26	-	4.0	-	
t <sub>BUF</sub>	Bus free time between a STOP and START condition	4.7	-	1.3	-	0.5	-	4.7	-	
t <sub>LOW</sub>	Low period of the SCL clock	4.7	-	1.3	-	0.5	-	4.7	-	
t <sub>HIGH</sub>	Period of the SCL clock	4.0	-	0.6	-	0.26	-	4.0	50	ns
t <sub>r</sub>	Rise time of both SDA and SCL signals	-	1000	-	300	-	120	-	1000	
t <sub>f</sub>	Fall time of both SDA and SCL signals	-	300	-	300	-	120	-	300	

A [Seção 52.4.10 do Manual de Referência](#) apresenta alguns exemplos de configuração dos campos do registrador I<sup>2</sup>C\_TIMINGR para diferentes frequências de *clock kernel* e demandas

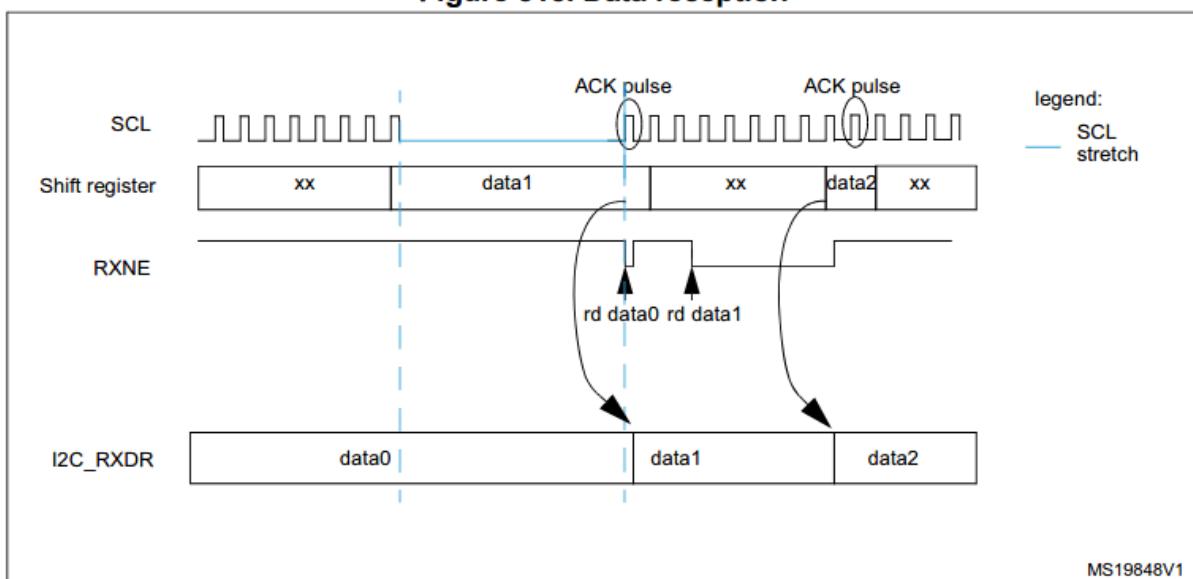
Ferramentas como o STM32CubeMX simplificam a configuração, abstraindo a complexidade do cálculo manual dos valores dos registradores em vez de expor diretamente esses parâmetros complexos ao desenvolvedor. O desenvolvedor insere a frequência SCL desejada (ex: 100 kHz, 400 kHz) na interface gráfica, e o CubeMX, por trás dos panos, calcula e gera os valores corretos para o I<sup>2</sup>Cx\_TIMINGR com base no *clock* do sistema e nas especificações do I<sup>2</sup>C. A seguinte figura ilustra a interface de configuração do STM32CubeMX.



Por fim, para que o periférico I2C comece a operar, é necessário habilitá-lo definindo o *bit* I2C\_CR1\_PE no registrador I2C\_CR1. Com essa etapa concluída, o periférico estará pronto para iniciar a comunicação I2C. Para informações detalhadas sobre a configuração do I2C, consulte o Manual de Referência. As seções relevantes incluem: configuração do modo escravo e processamentos no modo escravo ([Seção 52.4.8](#)); e configuração do modo mestre e processamentos no modo mestre ([Seção 52.4.9](#)).

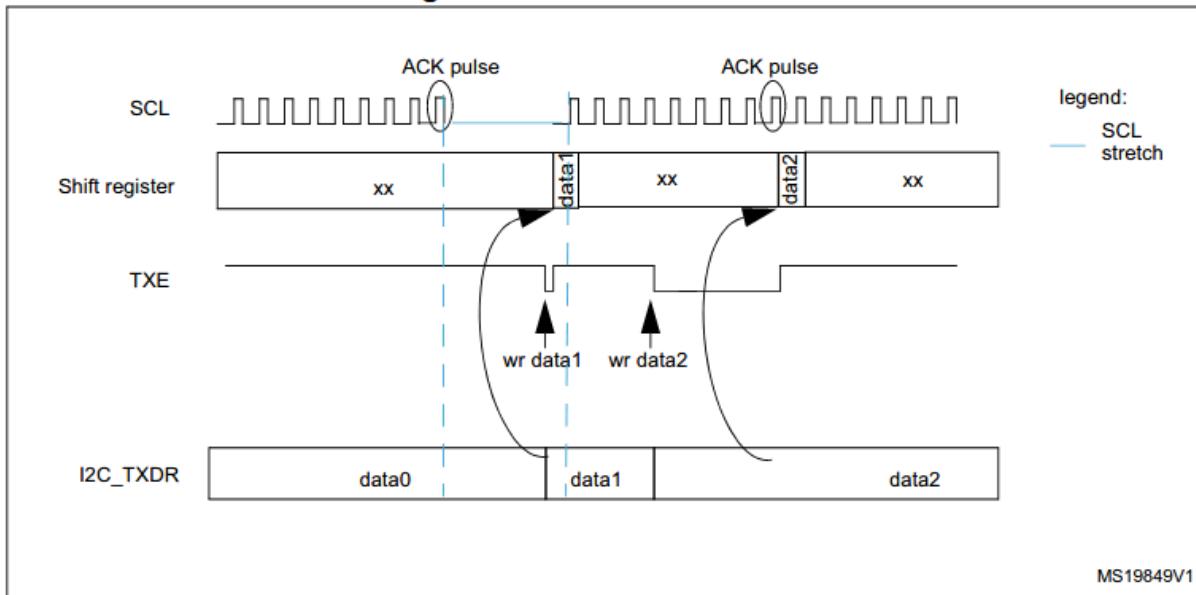
[Figura 515 do Manual de Referência](#) ilustra a dinâmica de recepção de um dado. Os dados de entrada SDA são armazenados em um registrador de deslocamento e transferidos para o registrador I2C\_RXDR, caso este esteja vazio (I2C\_ISR\_RXNE = 0). Se I2C\_ISR\_RXNE = 1, significando que o *byte* de dados recebido anteriormente ainda não foi lido, a linha SCL é estendida até que o I2C\_RXDR seja lido.

**Figure 515. Data reception**



[Figura 516 do Manual de Referência](#) ilustra a transmissão dos dados armazenados no registrador I2C\_TXDR. Se o registrador I2C\_TXDR não estiver vazio (I2C\_ISR\_TXE = 0), seu conteúdo é copiado para o registrador de deslocamento após o nono pulso SCL (o pulso de reconhecimento). Em seguida, o conteúdo do registrador de deslocamento é deslocado para fora na linha SDA. Se I2C\_ISR\_TXE = 1, significando que nenhum dado foi escrito ainda no I2C\_TXDR, a linha SCL é estendida até que o I2C\_TXDR seja escrito.

**Figure 516. Data transmission**



MS19849V1

O fluxo de dados entre o periférico I2C e a memória pode ser otimizado por meio do uso do DMA. Sem o DMA, a CPU é responsável por gerenciar cada *byte* transferido, o que consome tempo e pode comprometer o desempenho, especialmente em aplicações que lidam com grandes volumes de dados. Com o DMA, a transferência de dados é delegada ao controlador DMA e ao multiplexador DMAMUX, permitindo que ambos operem sem intervenção da CPU, liberando-a para outras tarefas. A configuração do controlador DMA e do DMAMUX é semelhante àquela descrita para a transferência de dados entre os periféricos SPI e a memória via DMA. Basta consultar a [tabela de mapeamento do DMAMUX](#) e substituir os códigos (*DMA request MUX input*) correspondentes aos periféricos SPI pelos respectivos códigos dos periféricos I2C e habilitar transferências via DMA sentando os bits `I2C_CR1_RXDMAEN` e `I2C_CR1_TXDMAEN` em '1'. Os I2C1, I2C2 e I2C3 são mapeados no DMAMUX1, enquanto o I2C4 é mapeado no DMAMUX2.

13	I2C4_RX	29	Reserved
14	I2C4_TX	30	Reserved

31	TIM4_CH3	73	I2C3_RX	115	Reserved
32	TIM4_UP	74	I2C3_TX	116	UART9_RX
33	I2C1_RX	75	DCMI_PSSI	117	UART9_TX
34	I2C1_TX	76	CRYP_IN	118	USART10_RX
35	I2C2_RX	77	CRYP_OUT	119	USART10_TX
36	I2C2_TX	78	HASH_IN	120	Reserved

Os periféricos I2C geram diversos eventos de interrupção que sinalizam diferentes estados e ocorrências durante a comunicação. Esses eventos podem ser habilitados e configurados para gerar interrupções (IRQs) no processador, permitindo que o *software* reaja a situações específicas. A [Tabela 386 do Manual de Referência](#) lista os eventos de interrupção disponíveis

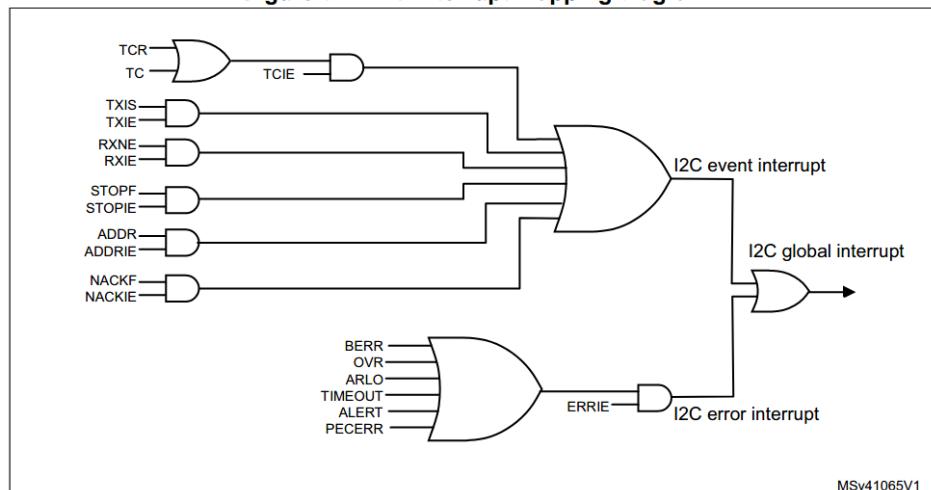
para os periféricos I2C. A segunda coluna da tabela apresenta o *bit* que indica o estado do evento correspondente no registrador `I2C_ISR`. A terceira coluna descreve as ações necessárias para limpar cada *bit* de estado, além do uso do registrador de limpeza `I2C_ICR`. Por fim, a quarta coluna indica o *bit* de habilitação de interrupção no registrador `I2C_CR1`.

**Table 386. I2C interrupt requests**

Interrupt event	Event flag	Event flag/interrupt clearing method	Interrupt enable control bit	Interrupt/wake-up activated		
				i2c_event_it	i2c_error_it	i2c_wkup
Receive buffer not empty	RXNE	Read I2C_RXDR register	RXIE	Yes	No	No
Transmit buffer interrupt status	TXIS	Write I2C_TXDR register				
Stop detection interrupt flag	STOPF	Write STOPCF=1				
Transfer Complete Reload	TCR	Write I2C_CR2 with NBYTES[7:0] ≠ 0				
Transfer complete	TC	Write START=1 or STOP=1				
Address matched	ADDR	Write ADDRCF=1				Yes <sup>(1)</sup>
NACK reception	NACKF	Write NACKCF=1				
Bus error	BERR	Write BERRCF=1	ERRIE	No	Yes	No
Arbitration loss	ARLO	Write ARLOCF=1				
Overrun/underrun	OVR	Write OVRCF=1				
PEC error	PECERR	Write PECERRCF=1				
Timeout/t <sub>LOW</sub> error	TIMEOUT	Write TIMEOUTCF=1				
SMBus Alert	ALERT	Write ALERTCF=1				

Quando uma interrupção é gerada pelo periférico I2C e a interrupção correspondente está habilitada, é gerada uma solicitação de interrupção por evento (em inglês, *I2C event interrupt*) ou uma solicitação de interrupção por erro (em inglês, *I2C error interrupt*) ao controlador NVIC (do inglês, *Nested Vector Interrupt Controller*).

**Figure 541. I2C interrupt mapping diagram**



Segue-se o mapeamento das 2 IRQs de cada periférico I2C nos números de vetores de interrupção.

<b>Signal</b>	<b>Priority</b>	<b>NVIC position</b>	<b>Acronym</b>	<b>Description</b>	<b>Address offset</b>
i2c2_ev_it					
exti_i2c2_ev_wkup	40	33	I2C2_EV	I2C2 event interrupt	0x0000 00C4
i2c2_err_it	41	34	I2C2_ER	I2C2 error interrupt	0x0000 00C8
i2c1_ev_it					
exti_i2c1_ev_wkup	38	31	I2C1_EV	I2C1 event interrupt	0x0000 00BC
i2c1_err_it	39	32	I2C1_ER	I2C1 error interrupt	0x0000 00C0
i2c3_ev_it					
exti_i2c3_ev_wkup	79	72	I2C3_EV	I2C3 event interrupt	0x0000 0160
i2c3_err_it	80	73	I2C3_ER	I2C3 error interrupt	0x0000 0164
i2c4_ev_it					
exti_i2c4_ev_it	102	95	I2C4_EV	I2C4 event interrupt	0x0000 01BC
i2c4_err_it	103	96	I2C4_ER	I2C4 error interrupt	0x0000 01C0