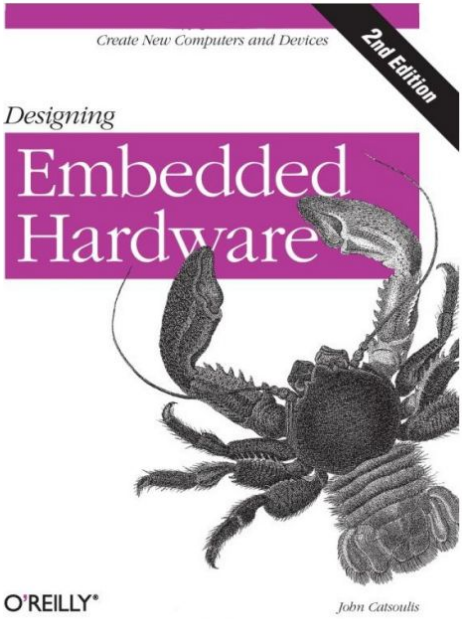
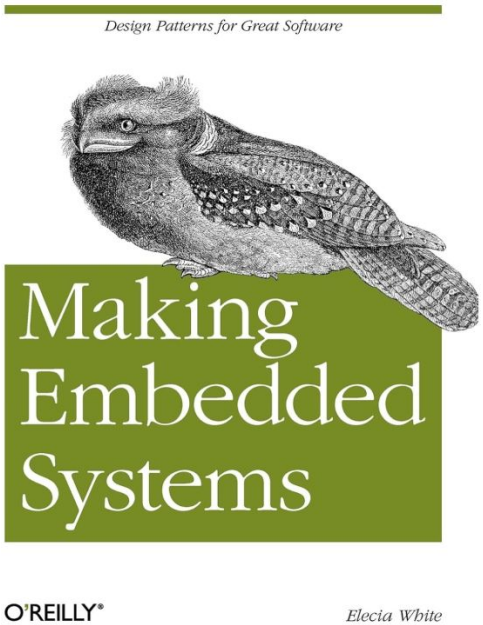
Comunicação Serial

UART I²C SPI

João Felipe Amaral Santiago





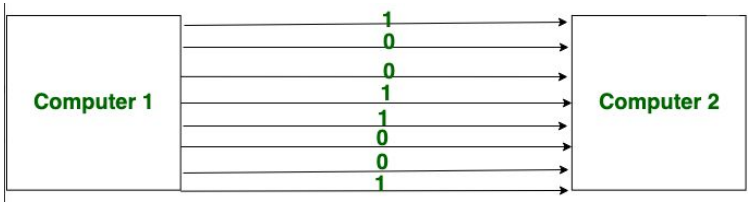
Slides das aulas

https://github.com/JaoIndio/Docencia_Microcontroladores

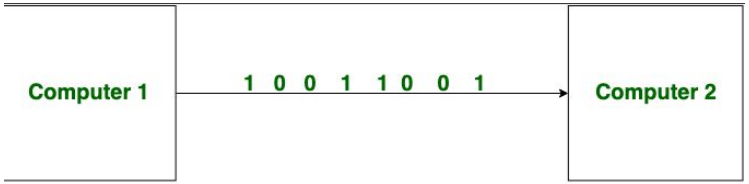
Fundamentos - Serial

Trata-se de um termo extremamente amplo, pois diz-se que uma comunicação é **serial** quando ocorre a transferência de bits ao longo do tempo por meio de um canal ou barramento.

Em contraposição há a comunicação **paralela.** Nela, todos os bits são enviados de uma única vez atrávez de multiplos barramentos ou canais.



Paralelo



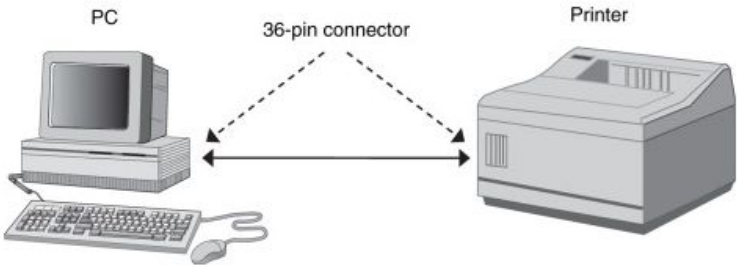
Serial

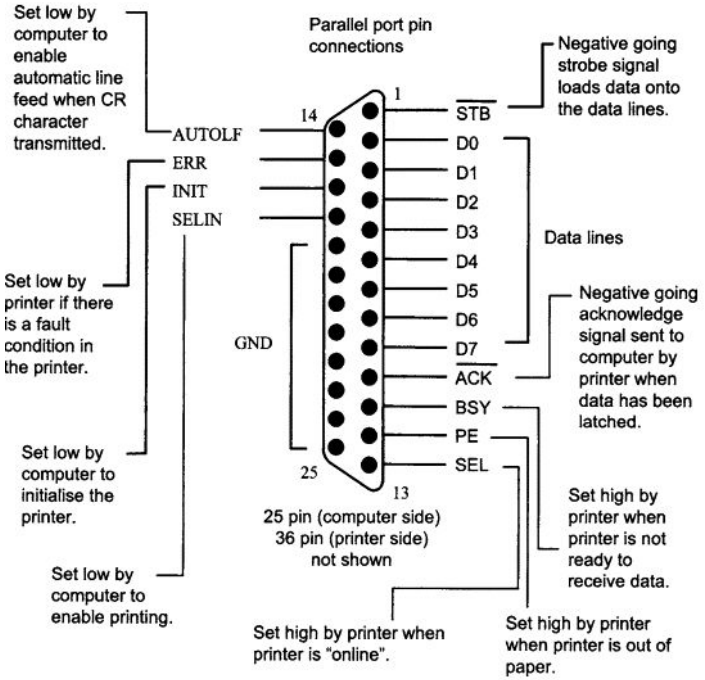
Fundamentos - Serial

Em essência a totalidade das comunicações entre computadores sempre será serial, uma vez que é inviável enviar todas as informações necessárias em um único barramento em um **único ciclo**.

A comunicação entre a CPU e memória é um exemplo de comunicação puramente paralela.

A comunicação com impressoras mais antigas também.



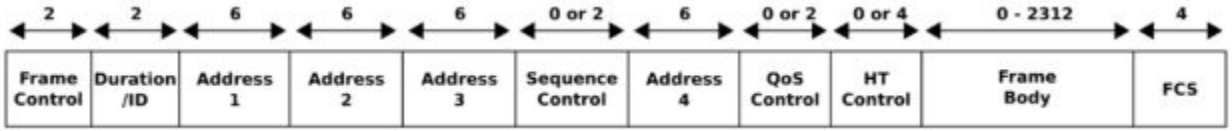


Fundamentos - Serial

Desenvolvida nos anos 80 pela Western Digital, Compaq e IBM, o padrão ATA (*Advanced Technology Attachment*) foi implementado com o propósito de integrar os discos rígidos e demais dispositivos de memória não volátil da época.

Suas variações possuíam de 40 a 80 pinos. No início dos anos 2000 foi substituído pelo padrão SATA (*Serial Advanced Technology Attachment)*.

O protocolo **WiFi** também transmite seus frames serialmente, **apesar** de explorar técnicas de envio muito sofisticadas as quais envolvem por exemplo, a divisão de seu frame em símbolos os quais são enviados em diferentes frequências. Esses símbolos são enviados **simultaneamente,** porém é necessário uma batelada de envios simultâneos para se ter o envio completo de um frame.



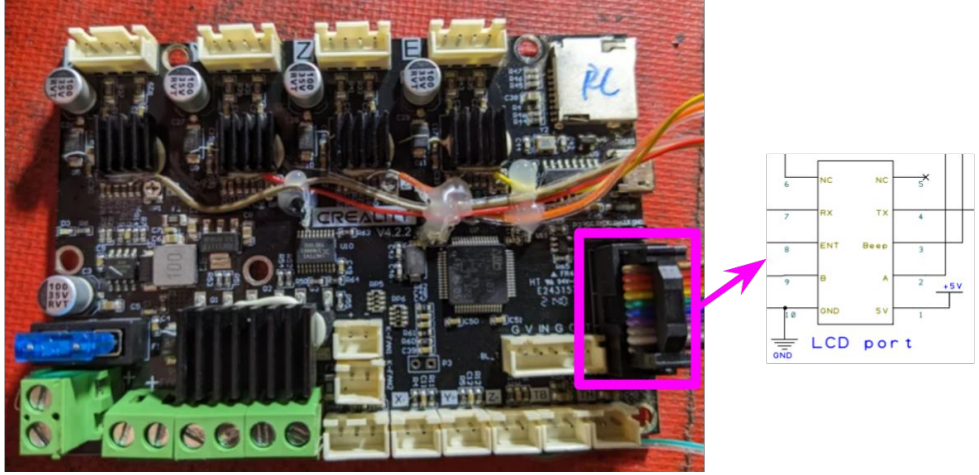
Campos do frame do protocolo IEEE 802.1 (padrão WiFi lançado em 1997)

Fundamentos - Serial

Dentro do escopo de **microcontroladores** e por consequência, sistemas embarcados, os protocolos comumente utilizados nas aplicações são o **UART**, **I²C** e **SPI**.

Quando comparados com outros protocolos cabeados seriais (CAN, ModBus, USB, PCIe, SATA, HDMI, Ethernet, etc), esses três são os mais simples, possuem menor complexidade portanto, sua implementação tende a ter custo reduzido.

São fortemente empregados para conectar e configurar RTCs (Real-Time Clock), memórias não voláteis (flash, EEPROM, etc), interfacemanto com sensores, atuadores e vários outros tipos de periféricos.

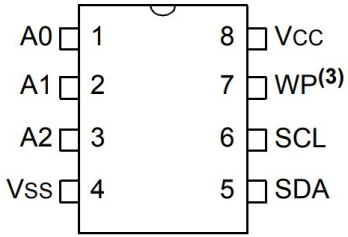


MICROCHIP

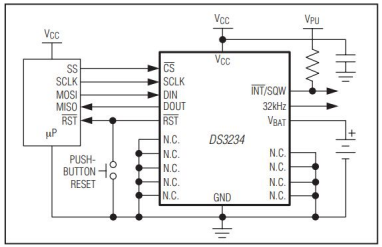
24AA00/24LC00/24C00 24AA01/24LC01B
24AA014/24LC014 24C01C/24C02C
24AA02/24LC02B 24AA025/24LC025
24AA024/24LC024 24AA08/24LC08B
24AA04/24LC04B 24AA32A/24LC32A
24AA16/24LC16B 24AA128/24LC128/24FC128
24AA64/24LC64/24FC64 24AA512/24LC512/24FC512
24AA256/24LC256/24FC256 24AA1025/24LC1025/24FC1025

I²C™ Serial EEPROM Family Data Sheet

PDIP/SOIC



Typical Operating Circuit



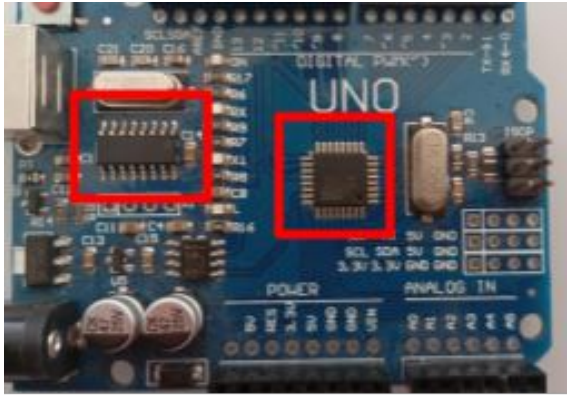
maxim integrated

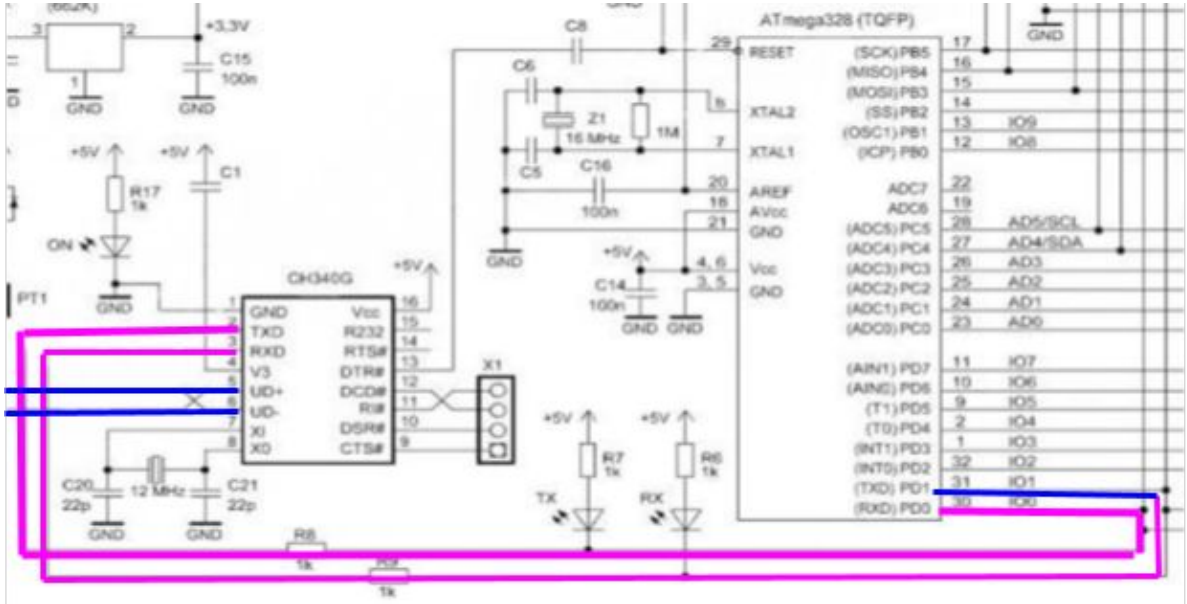
Extremely Accurate SPI Bus RTC with Integrated Crystal and SRAM

UART

Conecta o sistema embarcado ao mundo externo. Realiza a transferência de dados a partir do uso de algum outro protocolo físico (RS-232, RS-232C, RS-422, RS-485, USB, etc).

No Arduino Uno o micro ATmega328P liga os pinos TX e RX ao ci CH340G, que por sua vez é um conversor UART/USB.





UART - Universal Asynchronous Receiver Transmitter

É um dos protocolos mais simples. O termo assíncrono é atribuído a ele pois não há nenhum barramento responsável por ser o clock. Nesse sentido o recebedor deve ser capaz de detectar os bits individualmente sem o auxílio de sinais para sincronização.

Além dos barramentos TX/RX o UART fornece informações relacionadas ao status das informações, tais como se o buffer do receptor está cheio, ou o buffer de transmissão está vazio.

UARTs actually predate semiconductor-based computers. In the early days of electrical communication, UARTs were mechanical devices with cogs, relays, and electromechanical shift registers. To adjust a UART's settings, you first picked up a wrench!

UART

Uma de suas grandes desvantagens é a dificuldade na reconstrução dos dados recebidos. Uma dos maiores desafios de sua operação é a detecção da separação entre um bit e outro.

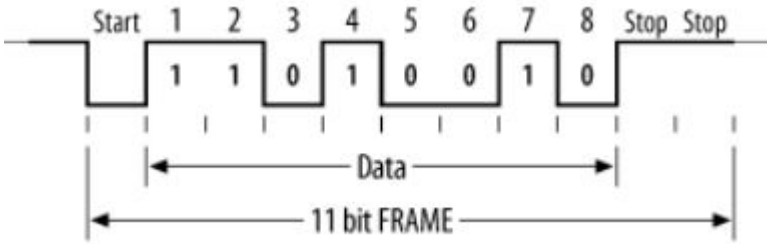
Por exemplo, se um barramento permanece em nível baixo por um dado período de tempo, o dispositivo receptor deve ser capaz de identificar se conjunto representa ‘00’ ou ‘000’. Ou seja, é necessário saber onde um bit termina e o próximo começa.

A solução mais prática para esse problema é o uso de um clock comum. Portanto em comunicações seriais **síncronas**, transmissor e receptor compartilham o mesmo clock. No caso de protocolos **assíncronos** os participantes da comunicação possuem seus próprios clocks. Consequentemente os dispositivos precisam necessariamente estarem configurados para operar na exata mesma frequência

UART

O formato de sua transmissão usa um bit de início no começo e um ou dois bits para indicar o fim. Uma vez identificado o bit de início, o receptor amostra 7 ou 8 bits (depende de sua configuração).

Uma vez recebido o *stop* bit, caso a sequência correta tenha sido enviada, o receptor irá supor que um caracter válido foi enviado e irá disponibilizado no buffer RX.



UART

Slides das aulas

https://github.com/JaoIndio/Docencia_Microcontroladores

Atividade - Material Complementar.

Implementar um echo entre dois Arduino Mega

Serial Monitor (COM1) -> Arduino 1 -> Arduino 2 -> Serial Monitor (COM2)

https://www.youtube.com/watch?v=P0O9sVRe15I
https://docs.arduino.cc/language-reference/en/functions/communication/serial/

https://docs.arduino.cc/micropython/communication/uart/

https://docs.arduino.cc/learn/communication/uart/ ←

https://docs.arduino.cc/built-in-examples/communication/ASCIITable/

https://docs.arduino.cc/built-in-examples/communication/SerialCallResponseASCII/

I²C - Intra-Integrated Circuit

Foi desenvolvido nos anos 80 pela Phillips (hoje NXP). Significa *Intra-Integrated Circuit*.

É classificado como um protocolo de baixa velocidade para os padrões atuais e é utilizado fortemente para conectar dispositivos periféricos ao microcontrolador. Desde 2006 não exige nenhum tipo de licensa para ser utilizado, desde então uma gama variada de fabricantes desenvolvem suas interfaces de integração para esse protocolo.

Seu barramento é composto por apenas dois fios.

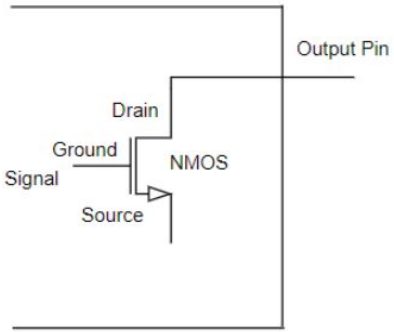
Possibilita a operção, desde que devidamente gerenciada, de multiplos clientes e multiplos servidores.

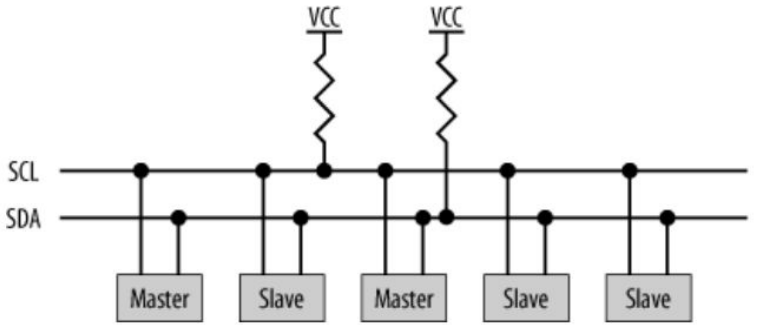
Suas velocidades mais tradicionais são 100 kbps (padrão) e 400 kbps (*fast mode*).

I²C

Internamente, os pinos de sua comunicação SDA (serial data) e SCL (serial clock) **sempre** estarão eletronicamente configurados a um **open-drain,** portanto para permitir o funcionamento adequado, um resistor de **pull up** deve ser adicionado ao circuito. Alguns dispositivos já possuem o pull up soldado na placa, ou internamente ao ci.

Cada dispositivo conectado à rede deve possuir seu próprio **endereço**.

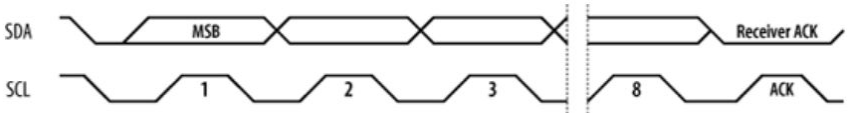


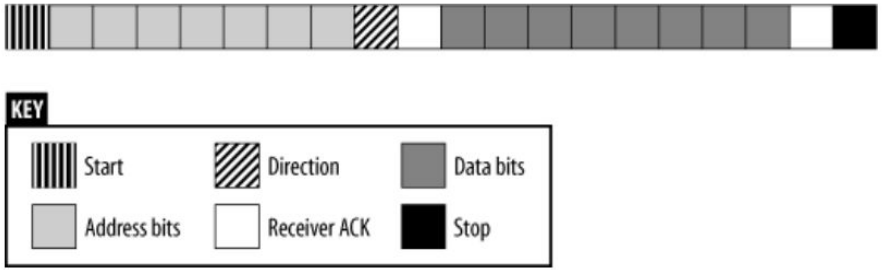


I²C

Ao contrário do UART (tipicamente realiza transferências em bytes), o I²C é capaz de transmitir qualquer número de bytes durante um envio.

A cada byte enviado um sinal adicional chamado de bit de confirmação (**ACK**, acknowledged) deve ser mandado pelo receptor.





I²C

Slides das aulas

https://github.com/JaoIndio/Docencia_Microcontroladores

Atividade.

Enviar requisições de leitura dos dois sensores presentes no ci MPU 9250

https://docs.arduino.cc/learn/communication/wire/

```
// Requisição de Escrita
Wire.begin();
Wire.beginTransmission(ADDR);
Wire.write(REG_ADDR);
Wire.write(VALUE_TO_WRITE);
byte error = Wire.endTransmission();

if (error == 0) {
  Serial.println("Write successful");
} else {
  Serial.print("Error during write: ");
  Serial.println(error);
}
```

```
// Requisição de Leitura
Wire.beginTransmission(ADDR);
Wire.write(REG_ADDR);
Wire.endTransmission(false);

// Request bytes from device
Wire.requestFrom(ADDR, bytesToRead_amount);

if (Wire.available() == bytesToRead) {
  byte firstByte = Wire.read();
  ...
}
```

SPI - Serial Peripheral Interface

Foi concebido pela Motorola nos anos 80. Assim como o I²C é utilizado para integrar dispositivos periféricos ao microcontrolador.

Não possui velocidades padronizadas. Cada fabricante fornece em seus *datasheets* a taxa de transferência máxima.

Trata-se de um protocolo síncrono cujo clock é gerado pelo **master** (microcontrolador). Tipicamente opera dentro da topologia 1 mestre, muitos escravos.

Seu barramento é formado por 4 sinais. **MOSI** (*Master Out Slave In*), **MISO** (*Master In Slave Out*), **SCK** (*Serial Clock*) e **CS** (*Chip Select*).

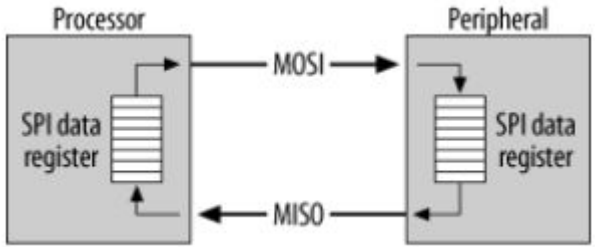
Todos os periféricos podem compartilhar os sinais **MOSI**, **MISO** e **SCK**, porém cada **CS** de cada escravo deve estar conectado a um pino diferente do mestre.

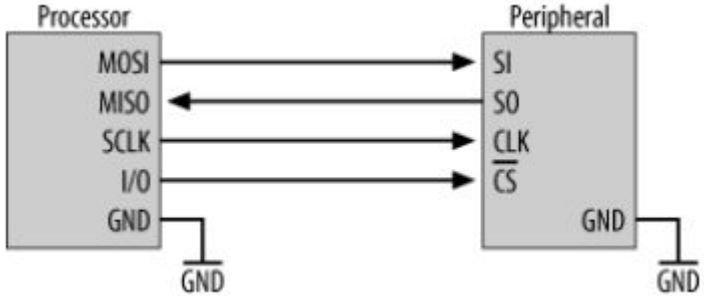
SPI

Tem como caracteristica ser um protocolo **full-duplex**, ou seja há transferência de bytes no barramento MOSI e MISO simultaneamente.

Na prática isso permite que uma operação de leitura e escrita seja efetuada em uma única transferência.

No caso em que há o interesse apenas de escrever em algum registrador, o master ignora a resposta do slave. Consequentemente, caso deseja-se realizar apenas uma leitura, o master primeiro envia a solicitação e na sequência realiza o envio de um byte “lixo” para forçar a transferência da resposta por parte do slave.





SPI

Existem 4 modos de operção. Todos são condicionados pelas configurações da **polaridade** e **fase** do clock.

Polaridade

Pode ser **High** ou **Low.** Quando em nível alto, o **SCK** permanece em nível alto quando em **IDLE,** quando em nível baixo, o contrário.

Fase

- Clock Phase Zero: MISO e MOSI são válidos durante **rising edges** do SCK, se a polaridade é **low.**

  MISO e MOSI são válidos durante **falling edges** do SCK, se a polaridade é **high**.

- Clock Phase One: MISO e MOSI são válidos durante **rising edges** do SCK, se a polaridade é **high.**

  MISO e MOSI são válidos durante **falling edges** do SCK, se a polaridade é **low**.

SPI

Figure 7-4. SPI timing with clock polarity low and clock phase zero

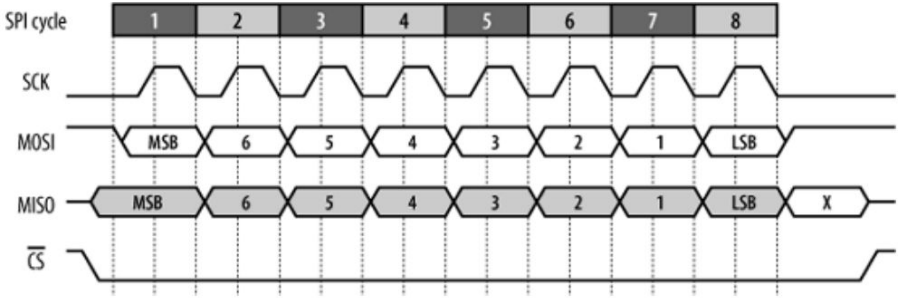


Figure 7-5. SPI timing with clock polarity high and clock phase zero

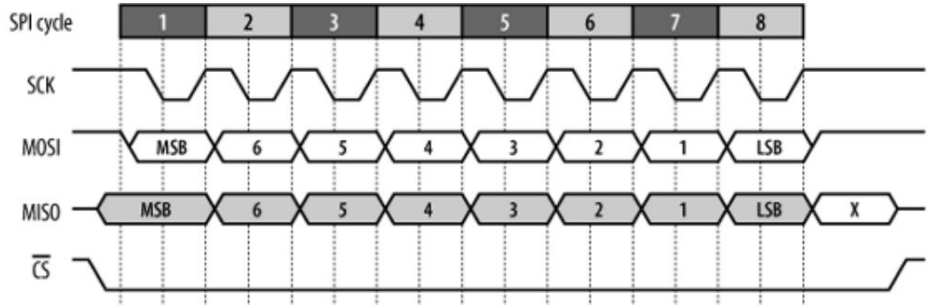


Figure 7-6. SPI timing with clock polarity low and clock phase one

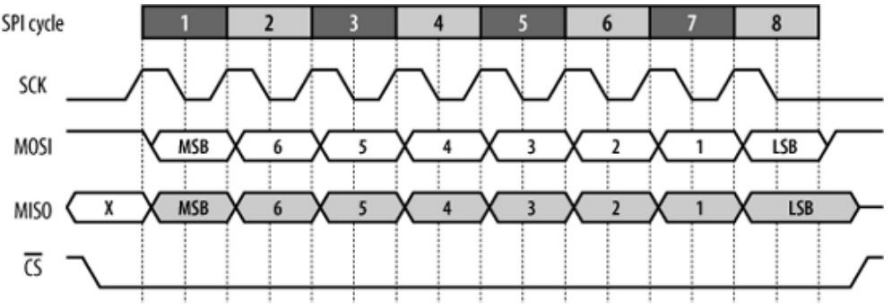
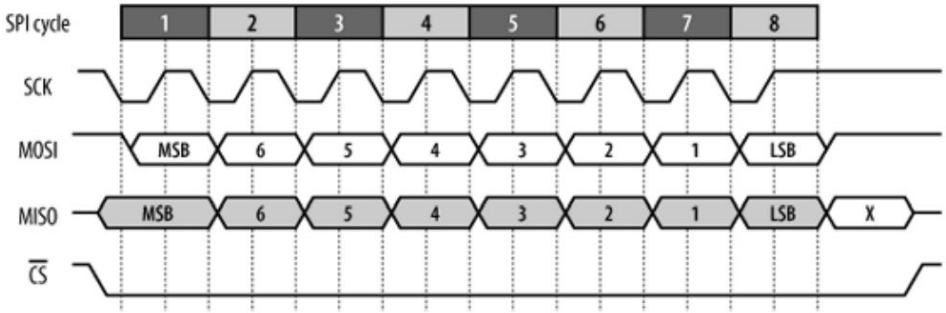


Figure 7-7. SPI timing with clock polarity high and clock phase one



Slides das aulas

https://github.com/JaoIndio/Docencia_Microcontroladores

SPI

Atividade.

Enviar requisições de leitura dos dois sensores presentes no ci MPU 9250

https://docs.arduino.cc/learn/communication/spi/

```
// Requisição de Escrita
#include <SPI.h>

SPISettings spi_Settings(1000000, MSBFIRST, SPI_MODE3);
SPI.begin();

SPI.beginTransaction(spi_Settings);
digitalWrite(CS_PIN, LOW);
SPI.transfer(REG);
SPI.transfer(REG_VALUE);
SPI.endTransaction();
digitalWrite(CS_PIN, HIGH);
```

```
// Requisição de Leitura
#include <SPI.h>

SPISettings spi_Settings(1000000, MSBFIRST, SPI_MODE3);
SPI.begin();

SPI.beginTransaction(spi_Settings);
digitalWrite(CS_PIN, LOW);

// Set MSB for read operation
SPI.transfer(reg | 0x80);
uint8_t data = SPI.transfer(0x00);

SPI.endTransaction();
digitalWrite(CS_PIN, HIGH);
```