5197 - Sistema Digitais

Bacharelado de Informática

UEM – DIN - Prof. Elvio

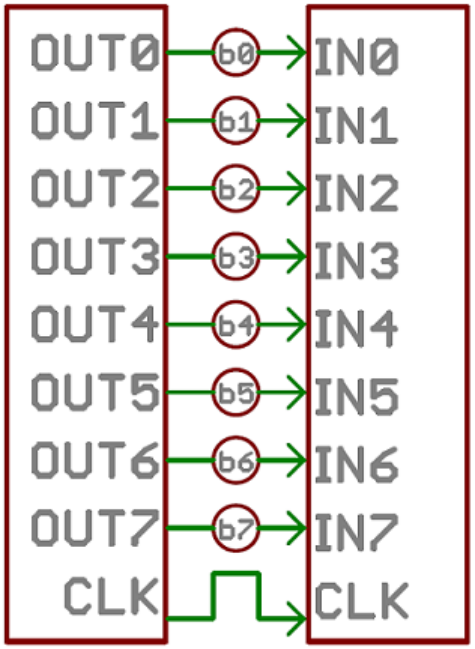
2017

Roteiro

- ATmega328 (SPI)
- ATmega328 (USART)

Introdução

- Interfaces de comunicação



Comunicação

Paralela

LPT (IEEE1284)

IDE

PCI

Serial

Assíncrona

RS232

RS485

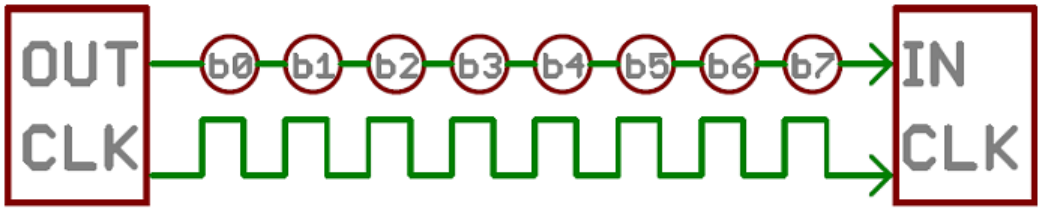
OneWire

Síncrona

SPI

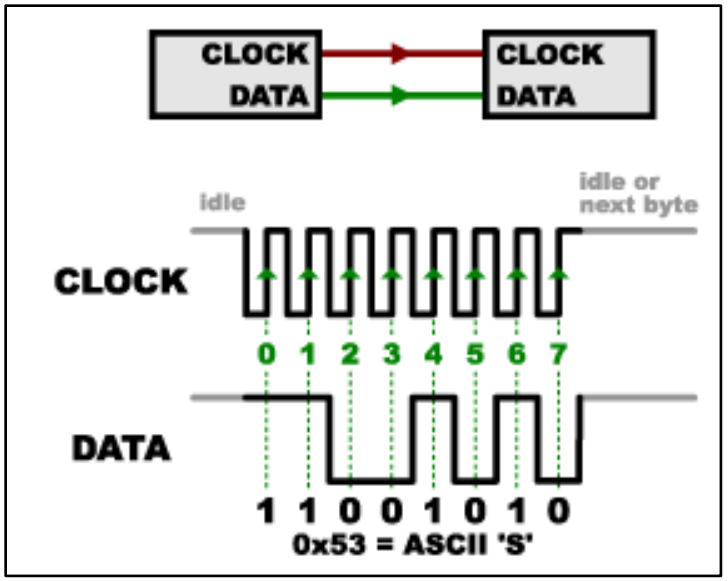
I2C

I2S



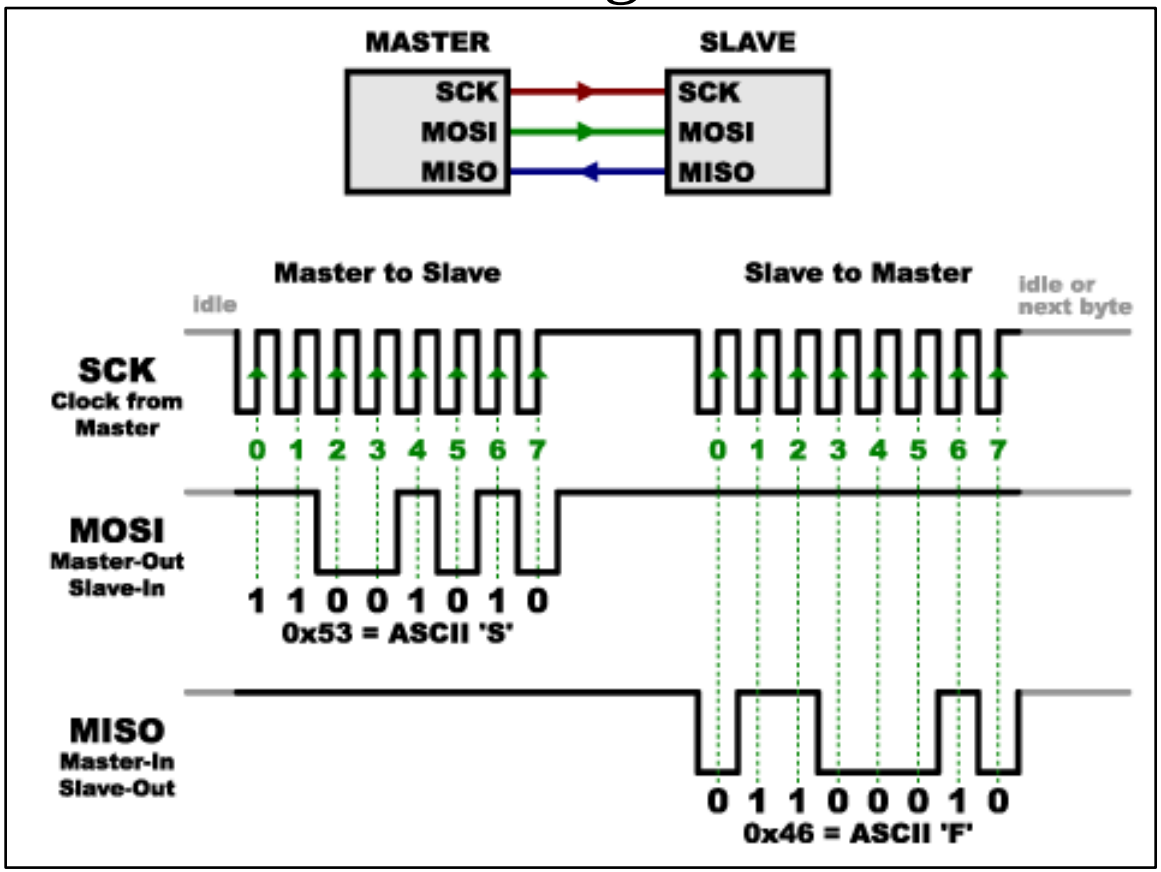
Interface Síncrona

- HW mais simples que interface assíncrona
- Usa linhas de dados e relógio separadas
- Dados são amostrados na borda (subida ou descida) do sinal de relógio
- Como relógio e dado seguem juntos, a especificação da velocidade não é importante
  - Deve-se especificar apenas a velocidade máxima



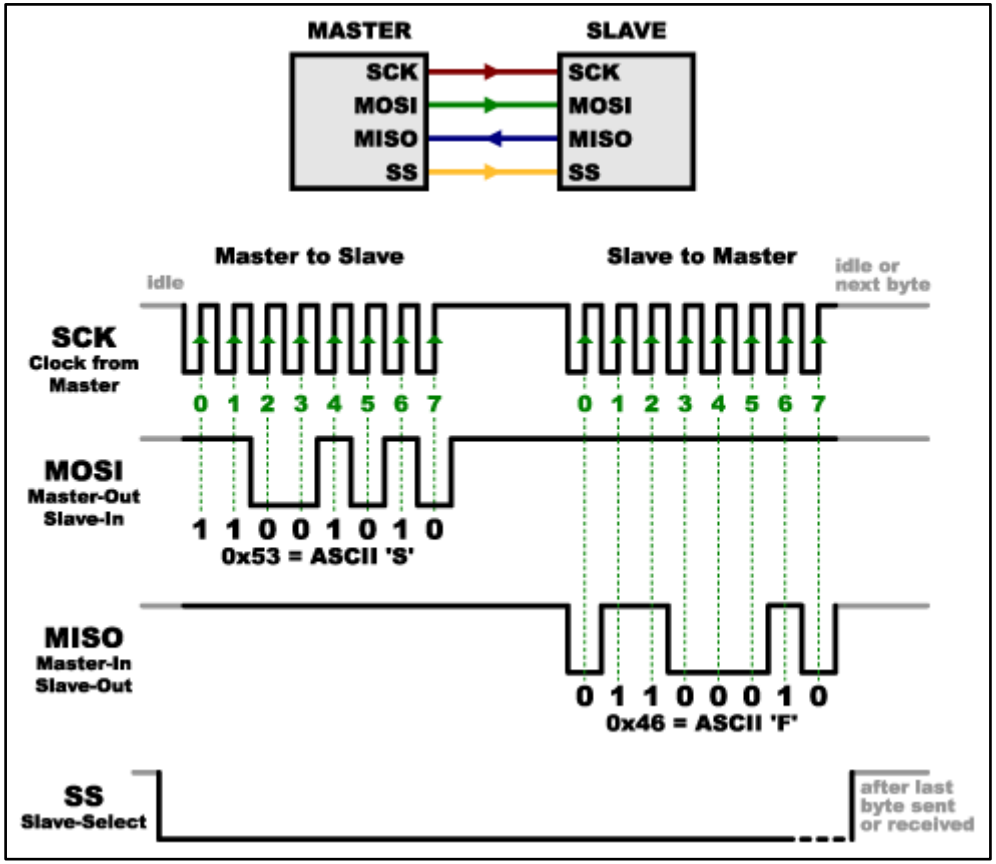
SPI

- Terminais **mestre** e **escravo**
  - Mestre fornece o relógio



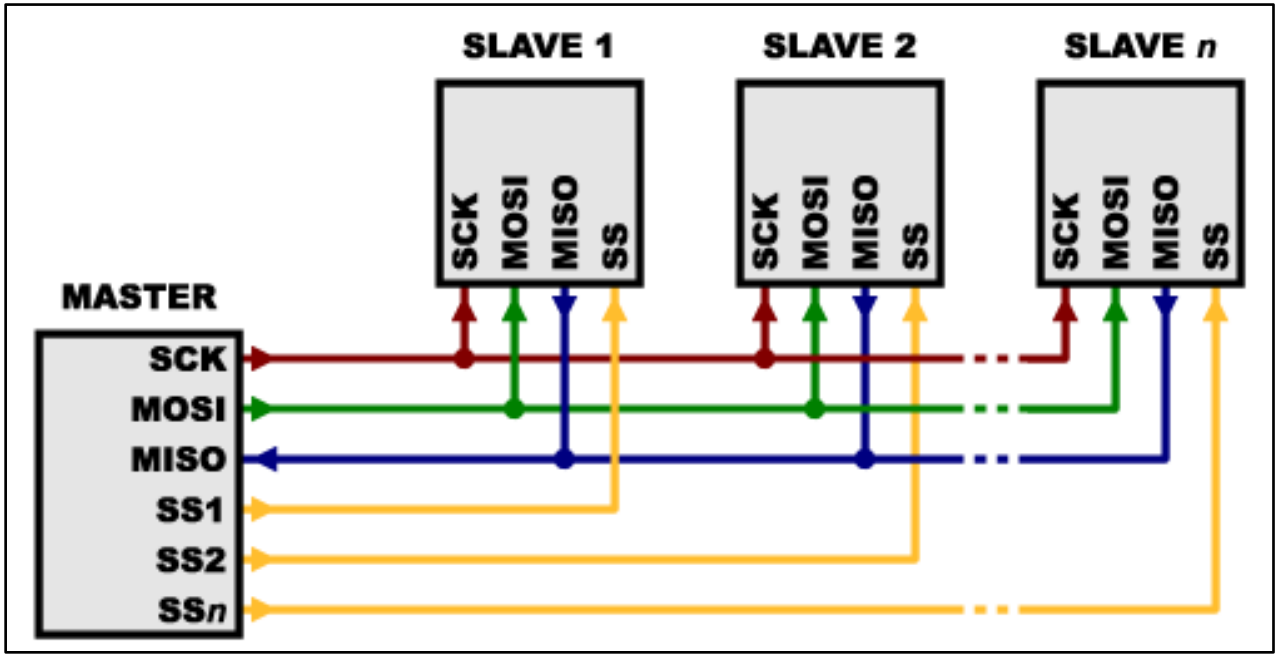
SPI

- Sinal **<u>Slave Select (SS)</u>**
  - Fornecido pelo mestre, habilita o escravo



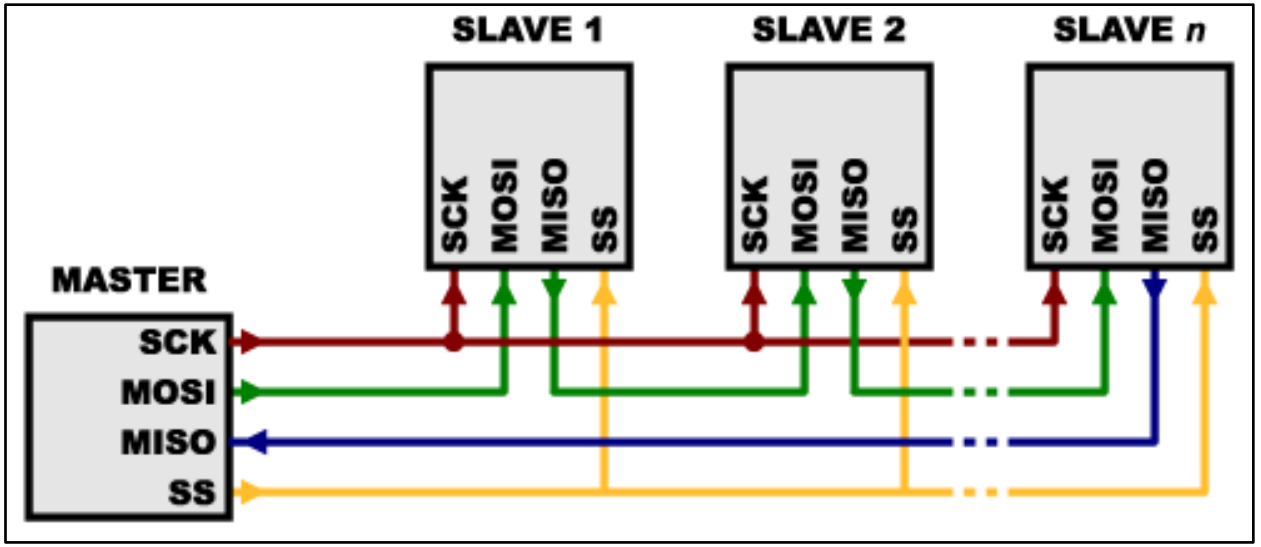
SPI

- Múltiplos escravos
  - Com múltiplos sinais de habilitação (SS) e operação independente



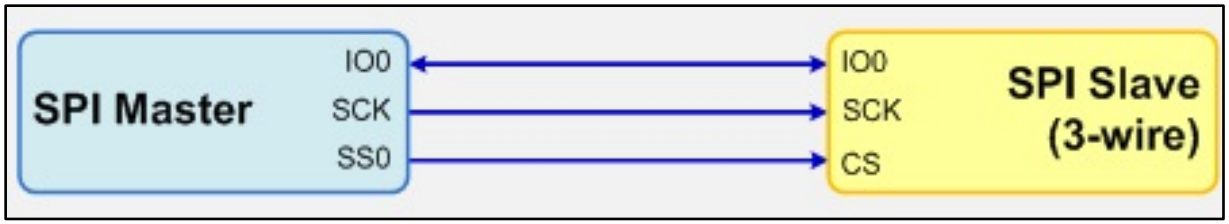
SPI

- Múltiplos escravos (cont.)
  - Apenas um sinal de habilitação (SS) e operação em *daisy-chain*

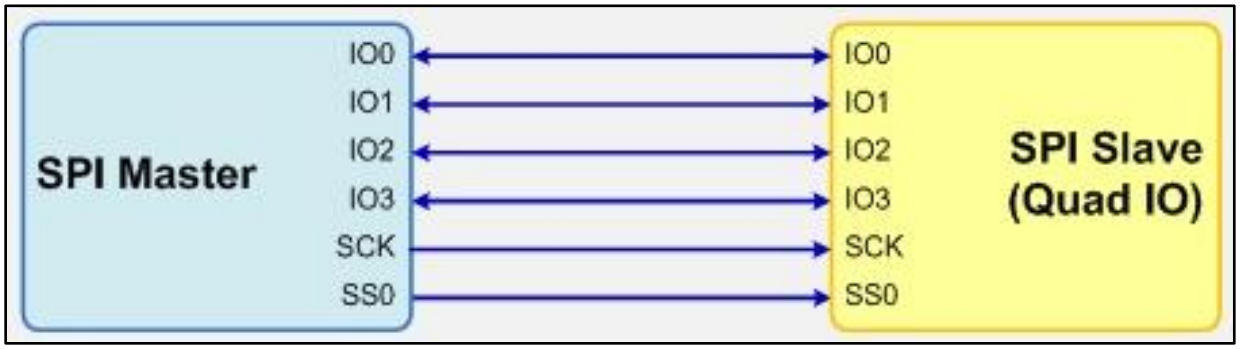


SPI

- Ligações de I/O Alternativas
  - Apenas 3 fios



  - Múltiplas linhas de I/O



SPI

- Vantagens:
  - Mais rápida que comunicação assíncrona
  - Hardware é simples (apenas um registrador de deslocamento)
  - Permite múltiplos escravos
- Desvantagens:
  - Usa várias linhas de sinal
  - Comunicação tem que ser definida a priori, dependente do periférico
  - Gerência centralizada no mestre (escravos não podem conversar entre si)
  - Usualmente exige uma linha de habilitação (SS) para cara escravo

SPI no ATmega328

- Características principais:
  - Interface Síncrona
  - Modo *Full-duplex*
  - Opera como mestre ou escravo
  - Transferência de dados com LSB ou MSB primeiro
  - Taxa de dados configurável (7 opções)
  - *Flag* de final de transmissão
  - *Flag* de conflito de escrita
  - Pode ser usado para despertar de *Idle Mode*

SPI no ATmega328

- Configuração automática dos pinos:
  - Pino PB2: sinal SS
  - Pino PB3: sinal MOSI
  - Pino PB4: sinal MISO
  - Pino PB5: sinal SCK

Pin	Direction, Master SPI	Direction, Slave SPI
MOSI	User Defined	Input
MISO	Input	User Defined
SCK	User Defined	Input
$\overline{SS}$	User Defined	Input

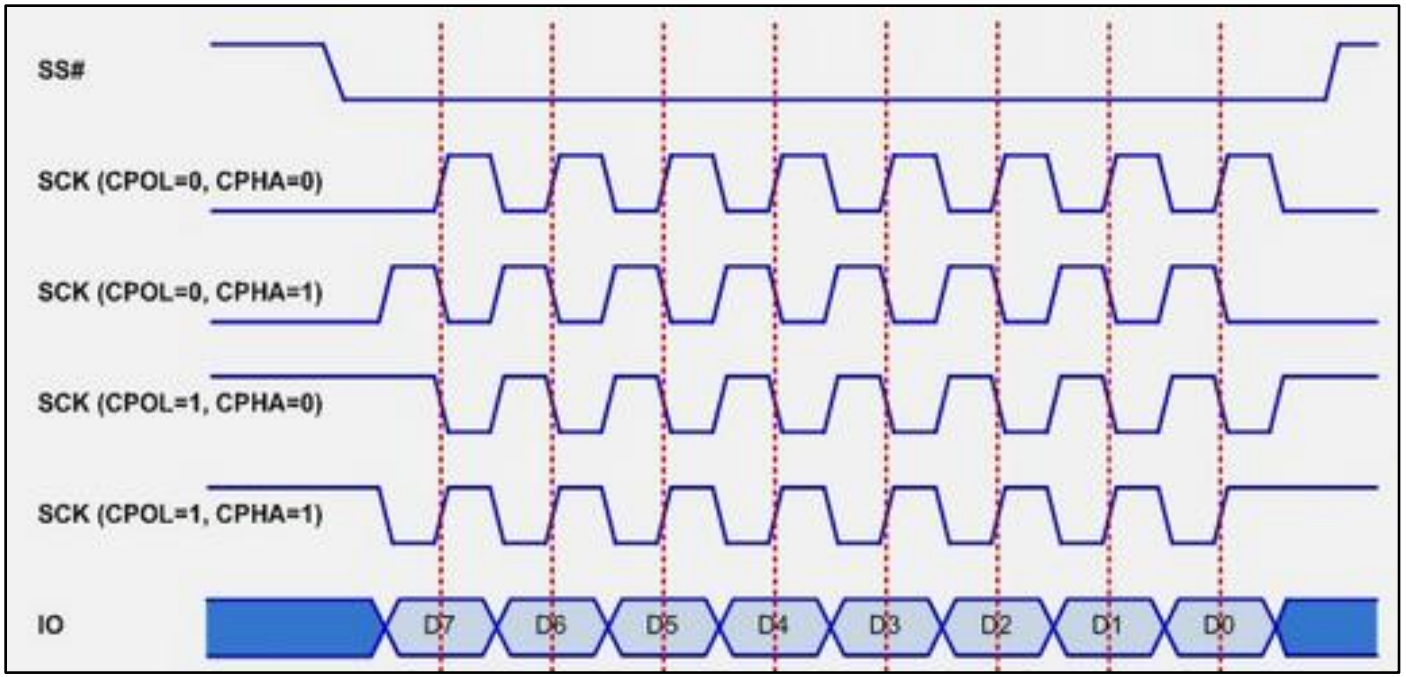
SPI no ATmega328

- Reg. SPI Control (SPCR)
  - SPI Interrupt Enable (SPIE): habilita interrupção
  - SPI Enable (SPE): habilita interface SPI
  - Data Order (DORD): indica qual é transmitido primeiro, LSB ou MSB
  - Master/Slave Select (MSTR): seleciona entre mestre e escravo
  - Clock Polarity (CPOL) e Clock Phase (CPHA): determinam o modo de operação (detalhes adiante)
  - SPI Clock Rate Select 1, 0 (SPR[1:0]) e Double SPI Speed Bit (SPI2X): determinam o relógio SCK (detalhes adiante)

SPI no ATmega328

- Modos de operação:

Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1



SPI no ATmega328

- Frequência do relógio:

SPI2X	SPR1	SPR0	SCK Frequency
0	0	0	$f_{osc}/4$
0	0	1	$f_{osc}/16$
0	1	0	$f_{osc}/64$
0	1	1	$f_{osc}/128$
1	0	0	$f_{osc}/2$
1	0	1	$f_{osc}/8$
1	1	0	$f_{osc}/32$
1	1	1	$f_{osc}/64$

SPI no ATmega328

- Reg. SPI Status (SPSR)
  - SPI Interrupt Flag (SPIF): indica quando uma transferência foi concluída
  - Write Collision Flag (WCOL): indica quando o registrador de dado (SPDR) foi alterado durante uma transferência
  - Double SPI Speed Bit (SPI2X) fica neste registrador
- Reg. SPI Data Register (SPDR)

SPI no ATmega328

- Exemplo de código

```
void SPI_MasterInit(void)
{
  /* Set MOSI and SCK output, all others input */
  DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);
  /* Enable SPI, Master, set clock rate fck/16 */
  SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}

void SPI_MasterTransmit(char cData)
{
  /* Start transmission */
  SPDR = cData;
  /* Wait for transmission complete */
  while(!(SPSR & (1<<SPIF)))
    ;
}
```

SPI no ATmega328

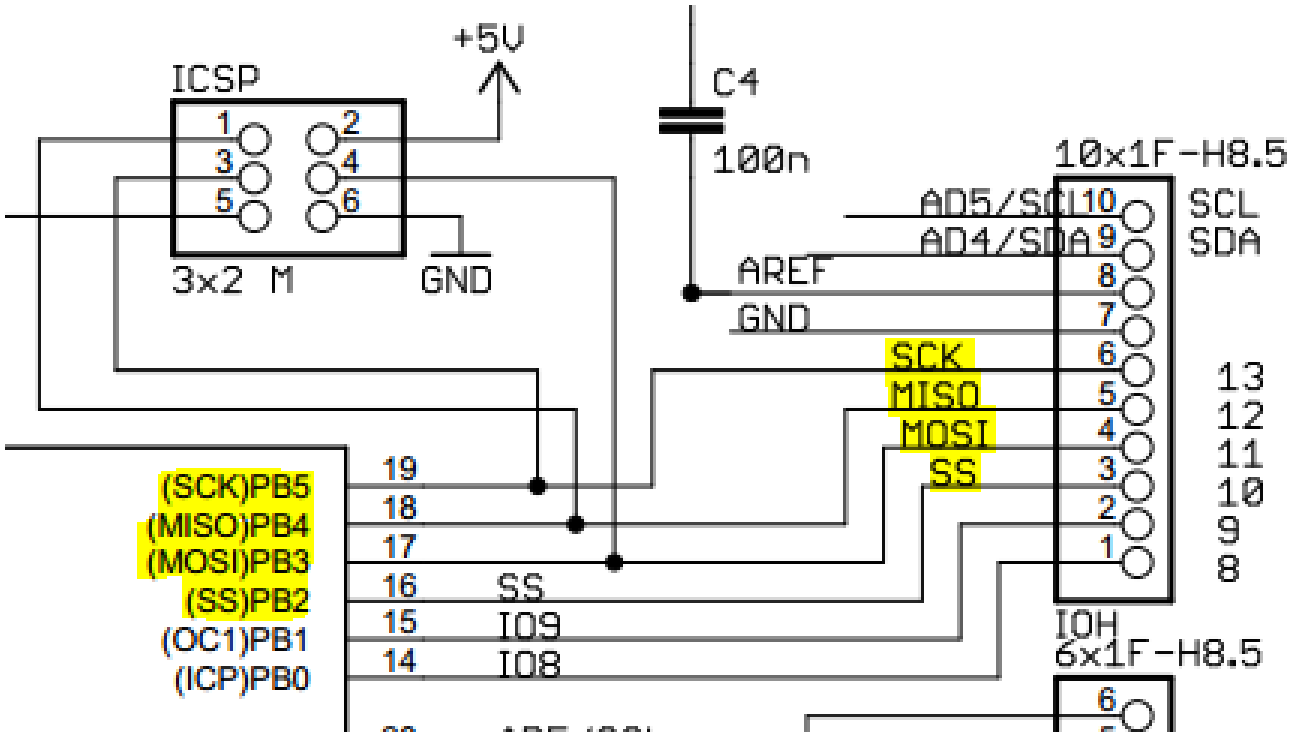
- Exemplo de código

```
void SPI_SlaveInit(void)
{
  /* Set MISO output, all others input */
  DDR_SPI = (1<<DD_MISO);
  /* Enable SPI */
  SPCR = (1<<SPE);
}

char SPI_SlaveReceive(void)
{
  /* Wait for reception complete */
  while(!(SPSR & (1<<SPIF)))
    ;
  /* Return Data Register */
  return SPDR;
}
```

SPI no Arduino Uno R3

- Sinais disponíveis nos pinos 10 a 13
- Sinal com nível de 5 V



SPI no Arduino IDE

- Biblioteca SPI
  - begin(): inicia interface usando configuração padrão
  - end(): desabilita interface
  - SPISettings(): configura interface
  - beginTransaction(SPISettings()): inicia interface usando configuração de SPISettings()
  - endTransaction(): desabilita interface
  - transfer() e transfer16(): realiza transferência (transmite o valor passado e retorna o valor recebido)
  - usingInterrupt(): informa uso de interrupção

SPI no Arduino IDE

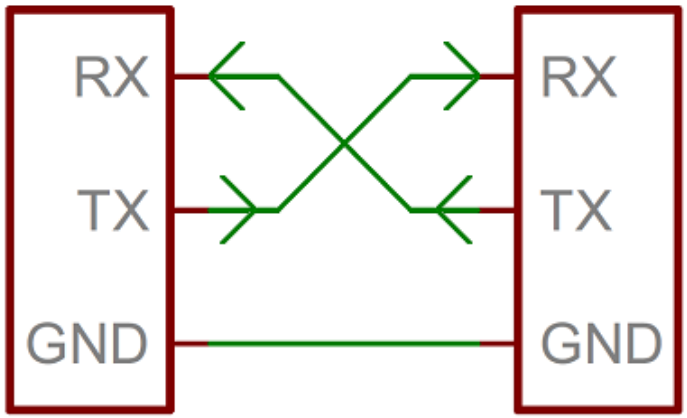
- Exemplo de código

```
int readStuff(void) {
  SPI.beginTransaction(SPISettings(12000000, MSBFIRST, SPI_MODE0));  // gain control of SPI bus
  digitalWrite(10, LOW);         // assert chip select
  SPI.transfer(0x74);            // send 16 bit command
  SPI.transfer(0xA2);
  byte b1 = SPI.transfer(0);     // read 16 bits of data
  byte b2 = SPI.transfer(0);
  digitalWrite(10, HIGH);        // deassert chip select
  SPI.endTransaction();          // release the SPI bus
  return (int16_t)((b1 << 8) | b2);
}
```

USART

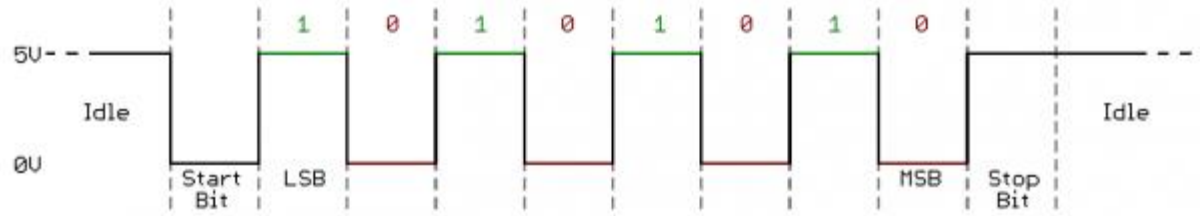
- Comunicação consiste de:

- *Baud rate*: velocidade da comunicação (refere-se a símbolos por segundo)
- Conexão:

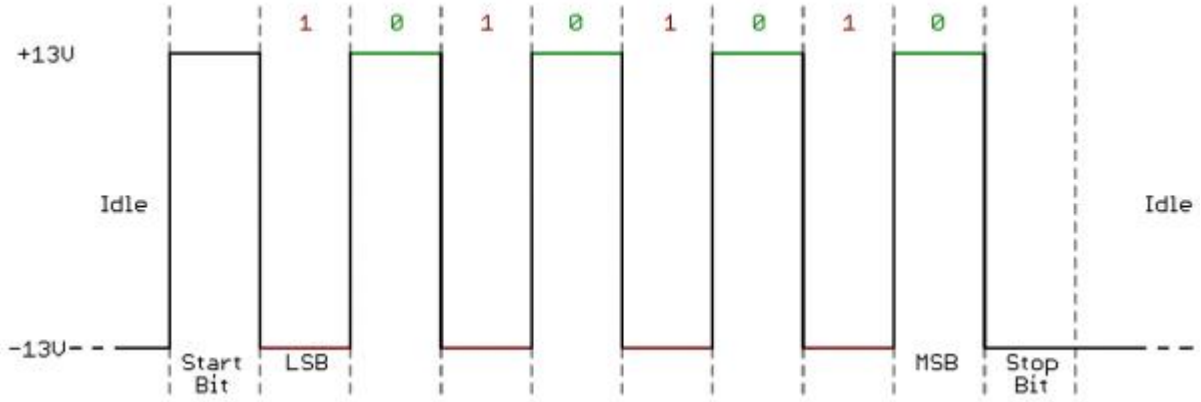


USART

- Nível TTL



- RS-232



USART no ATmega328

- Operação *full duplex* (registradores independentes de recepção e transmissão)
- Operação síncrona ou assíncrona
- Operação síncrona fornecendo relógio (mestre) ou recebendo relógio (escravo)
- Permite quadros com 5 a 9 bits de dados e 1 ou 2 bits de parada
- Gerador de paridade par ou ímpar e verificação de paridade por *hardware*
- Detecção de colisão de dados e erros de frames
- Interrupção em: *(i)* transmissão completa, *(ii)* recepção completa, e *(iii)* esvaziamento do buffer de transmissão

USART no ATmega328

- Velocidade da interface

Modo de operação	Equação para o cálculo da taxa de transmissão	Equação para o cálculo do valor de UBRR0
Modo Normal Assíncrono (U2X0 = 0)	$TAXA = \frac{f_{osc}}{16(UBRR0+1)}$	$UBRR0 = \frac{f_{osc}}{16.TAXA} - 1$
Modo de Velocidade Dupla Assíncrono (U2X0 = 1)	$TAXA = \frac{f_{osc}}{8(UBRR0+1)}$	$UBRR0 = \frac{f_{osc}}{8.TAXA} - 1$
Modo Mestre Síncrono	$TAXA = \frac{f_{osc}}{2(UBRR0+1)}$	$UBRR0 = \frac{f_{osc}}{2.TAXA} - 1$

- Reg. USART I/O Data 0 (UDR0)
  - Escrita: dado para transmissão
  - Leitura: dado recebido

USART no ATmega328

- Reg. USART *Control and Status* A (UCSR0A)
  - *Receive Complete* (RXC0): dado disponível no buffer de recepção
  - *Transmit Complete* (TXC0): último bit do dado foi transmitido
  - *Data Register Empty* (UDRE0): *buffer* de transmissão está vazio
  - *Frame Error* (FE0): erro no quadro recebido
  - *Data Over Run* (DOR0): *buffer* de recepção está cheio e novo dado foi recebido (está aguardando no registrador de deslocamento)
  - *Parity Error* (UPE0): erro de paridade
  - *Double* TX *Speed* (U2X0): velocidade de transmissão dobrada
  - *Multi-processor Communication Mode* (MPCM0): modo "multi-processador“
    - quadros recebidos sem endereço são descartados

USART no ATmega328

- Reg. USART *Control and Status* B (UCSR0B)
  - RX *Complete Interrupt Enable* (RXCIE0): habilita interrupção com RXC0
  - TX *Complete Interrupt Enable* (TXCIE0): habilita interrupção com TXC0
  - *Data Register Empty Interrupt Enable* (UDRIE0): habilita interrupção com UDRE0
  - *Receiver Enable* (RXEN0): habilita receptor da USART
  - *Transmitter Enable* (TXEN0): habilita transmissor da USART
  - *Character Size* (UCSZ0[2]): combinado com UCSZ0[1,0] determina tamanho do quadro de dados
  - *Receive Data Bit* 8 (RXB80): nono *bit* recebido (para quadro com 9 *bits* de dados)
  - *Transmit Data Bit* 8 (TXB80): nono *bit* transmitido (para quadro com 9 *bits* de dados)

USART no ATmega328

- Reg. USART *Control and Status* C (UCSR0C)
  - *Mode Select* (UMSEL0[1:0])
  - *Parity Mode* (UPM0[1:0]): veja abaixo
  - *Stop Bit Select* (USBS0): seleciona *stop bit* para 1 *bit* (USBS0 = 0) ou 2 *bits* (USBS0 = 1)

UMSEL01	UMSEL00	Modo de operação
0	0	assíncrono
0	1	síncrono
1	0	reservado
1	1	SPI mestre

UPM01	UPM00	Modo de Paridade
0	0	Desabilitado
0	1	Reservado
1	0	Habilitado, paridade par
1	1	Habilitado, paridade ímpar

USART no ATmega328

- Reg. USART *Control and Status* C (UCSR0C)
  - *Character Size* (UCSZ0[1:0]): veja abaixo
  - *Clock Polarity* (UCPOL0): veja abaixo

UCSZ02	UCSZ01	UCSZ00	Tamanho do Caractere
0	0	0	5 bits
0	0	1	6 bits
0	1	0	7 bits
0	1	1	8 bits
1	0	0	reservado
1	0	1	reservado
1	1	0	reservado
1	1	1	9 bits

UCPOL0	Mudança do Dado Transmitido (saída do pino TxD0)	Amostragem do Dado Recebido (entrada do pino RxD0)
0	borda de subida de XCK	borda de descida de XCK
1	borda de descida de XCK	borda de subida de XCK

USART no ATmega328

- Reg. Baud Rate (UBRR0L e UBRR0H)
  - Baud Rate (UBRR[11:0]): usado no cálculo do *baud rate* da interface

Baud Rate (bps)	f_{osc} = 16.0000MHz			
	U2Xn = 0		U2Xn = 1	
	UBRRn	Error	UBRRn	Error
2400	416	-0.1%	832	0.0%
4800	207	0.2%	416	-0.1%
9600	103	0.2%	207	0.2%
14.4k	68	0.6%	138	-0.1%
19.2k	51	0.2%	103	0.2%
28.8k	34	-0.8%	68	0.6%
38.4k	25	0.2%	51	0.2%
57.6k	16	2.1%	34	-0.8%
76.8k	12	0.2%	25	0.2%
115.2k	8	-3.5%	16	2.1%
230.4k	3	8.5%	8	-3.5%
250k	3	0.0%	7	0.0%
0.5M	1	0.0%	3	0.0%
1M	0	0.0%	1	0.0%
Max. (1)	1Mbps		2Mbps	

USART no ATmega328

```
//=============================================================================== //
//        INICIALIZANDO A USART                                                   //
//=============================================================================== //
#define FOSC    1843200         //Frequência de trabalho da CPU
#define BAUD    9600
#define MYUBRR FOSC/16/BAUD-1
//------------------------------------------------------------------------------
void main(void)
{
     ...
     USART_Init(MYUBRR);
     ...
}
//------------------------------------------------------------------------------
void USART_Init(unsigned int ubrr)
{
     UBRR0H = (unsigned char)(ubrr>>8);  //Ajusta a taxa de transmissão
     UBRR0L = (unsigned char)ubrr;
     UCSR0B = (1<<RXEN0)|(1<<TXEN0);     //Habilita o transmissor e o receptor
     UCSR0C = (1<<USBS0)|(3<<UCSZ00);    //Ajusta o formato do frame:
                                         //8 bits de dados e 2 de parada
}
//==============================================================================
```

USART no ATmega328

```
//=========================================================================== //
//        ENVIANDO FRAMES COM 5 A 8 BITS                                      //
//=========================================================================== //
void USART_Transmit(unsigned char data)
{
     while(!( UCSR0A & (1<<UDRE0)));//Espera a limpeza do registr. de transmissão
     UDR0 = data;                    //Coloca o dado no registrador e o envia
}
//===========================================================================


//=========================================================================== //
//        ENVIANDO FRAMES COM 9 BITS                                          //
//=========================================================================== //
void USART_Transmit(unsigned int data)
{
     while(!(UCSR0A &(1<<UDRE0)));//Espera a limpeza do registr. de transmissão

     UCSR0B &= ~(1<<TXB80);        //Copia o 9° bit para o TXB8

     if(data & 0x0100)
          UCSR0B |= (1<<TXB80);

     UDR0 = data;                 //Coloca o dado no registrador e o envia
}
//===========================================================================
```

USART no ATmega328

```
//=========================================================================== //
//        RECEBENDO FRAMES COM 5 A 8 BITS                                     //
//=========================================================================== //
unsigned char USART_Receive(void)
{
     while(!(UCSR0A & (1<<RXC0)));   //Espera o dado ser recebido
     return UDR0;                    //Lê o dado recebido e retorna
}
//===========================================================================

//=========================================================================== //
//        RECEBENDO FRAMES COM 9 BITS                                         //
//=========================================================================== //
unsigned int USART_Receive(void)
{
     unsigned char status, resh, resl;    //Espera o dado ser recebido
     while(!(UCSR0A &(1<<RXC0)));//Obtêm o status do 9° bit, então, o dado do registr.
     status = UCSR0A;
     resh = UCSR0B;
     resl = UDR0;

     if(status & (1<<FE0)|(1<<DOR0)|(1<<UPE0)) //Se ocorrer um erro retorna -1
          return -1;

     resh = (resh >> 1) & 0x01;               //Filtra o 9° bit, então, retorna
     return ((resh << 8) | resl);
}
//===========================================================================
```

USART no ATmega328

```
//=========================================================================== //
//  LIMPANDO O REGISTRADOR DE ENTRADA (quando ocorre um erro p. ex.)          //
//=========================================================================== //
void USART_Flush(void)
{
     unsigned char dummy;
     while(UCSR0A & (1<<RXC0)) dummy = UDR0;
}
//===========================================================================
```

USART no Arduino IDE

- if(Serial)
- available()
- availableForWrite()
- begin()
- end()
- find()
- findUntil()
- flush()
- parseFloat()
- parseInt()

USART no Arduino IDE

- peek()
- print()
- println()
- read()
- readBytes()
- readBytesUntil()
- readString()
- readStringUntil()
- setTimeout()
- write()
- serialEvent()