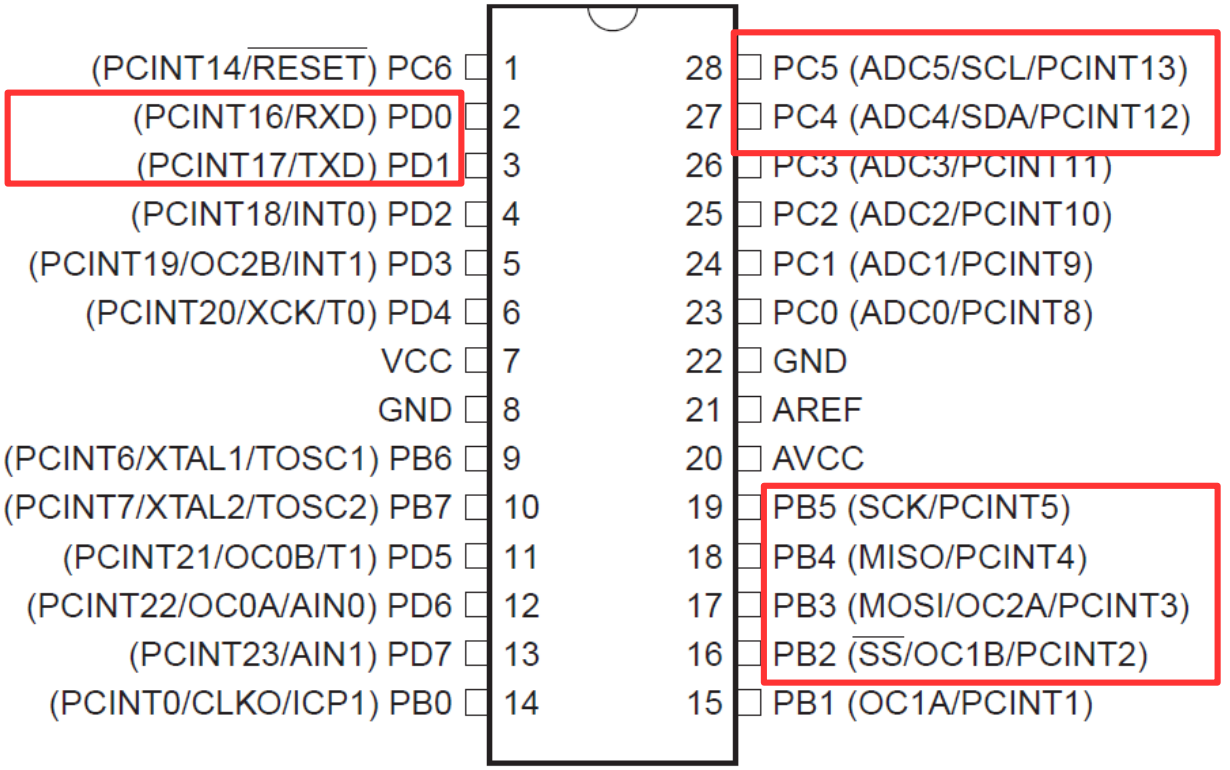
Sistemas embarcados

Comunicações seriais:
UART, SPI, IIC (I²C)

Periférico USART
(Universal Synchronous and Asynchronous serial Receiver and Transmitter)

Sistemas embarcados

UART



IIC

SPI

Nos microcontroladores é comum se ter periféricos para comunicação de dados serial, que pode ser síncrona ou assíncrona

Comunicação serial

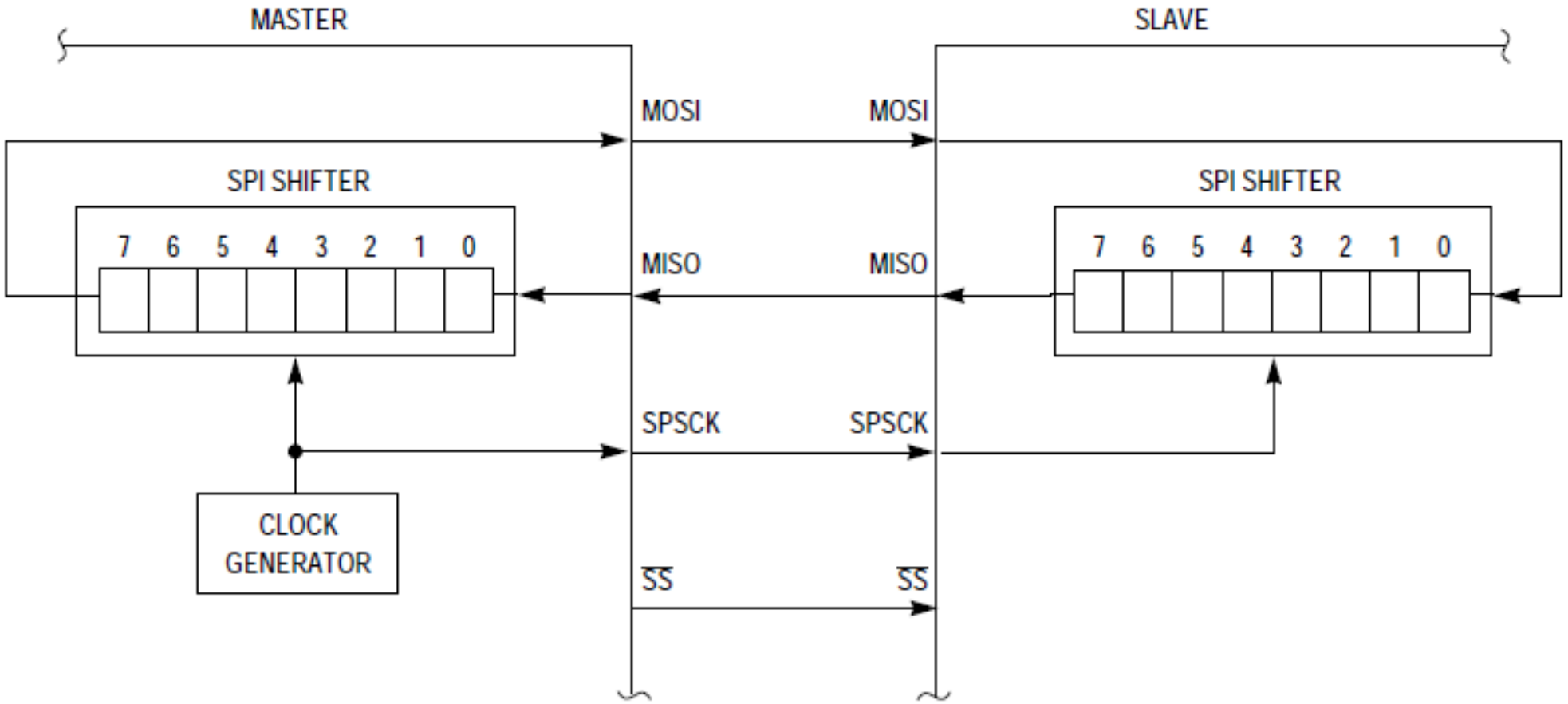
- Comunicações seriais podem ser:
  - Assíncronas (ex.: UART)
  - Síncronas (ex.: SPI e I2C)

A diferença está no uso de um sinal adicional de sincronização (clock)

Programando a SPI

- A comunicação SPI (Serial Peripherial Interface) funciona em modo mestre/escravo.
  - Mestre inicia comunicação (sempre) através do sinal de clock (comunicação síncrona).
- A comunicação SPI utiliza 4 linhas (MOSI, MISO, SCK, SS ou CS):
  - MOSI (Saída de dados do mestre e entrada de dados do escravo)
  - MISO (Entrada de dados do mestre e saída de dados do escravo)
  - SCK (Sinal de clock)
  - SS (Seleção do escravo, ou seleção de chip (CS))

Programando a SPI



Programando a SPI

- O modo de operação e a taxa de dados (baudrate) da SPI são configurados usando o registrador SPCR (reg. de controle)

SPCR – SPI Control Register

Bit	7	6	5	4	3	2	1	0	
0x2C (0x4C)	**SPIE**	**SPE**	**DORD**	**MSTR**	**CPOL**	**CPHA**	**SPR1**	**SPR0**	**SPCR**
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

- O estado da comunicação SPI é indicada pelo registrador SPSR (status)

SPSR – SPI Status Register

Bit	7	6	5	4	3	2	1	0	
0x2D (0x4D)	**SPIF**	**WCOL**	–	–	–	–	–	**SPI2X**	**SPSR**
Read/Write	R	R	R	R	R	R	R	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Programando a SPI

- O clock da SPI (taxa de dados) é um multiplo do clock do barramento.

Table 19-5. Relationship Between SCK and the Oscillator Frequency

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	$f_{osc}/64$

Programando a SPI

- O dado a transmitir ou recebido fica armazenado no registrador de 8 bits SPDR.

- O sinal de clock é configurado no mestre a partir da divisão do clock de barramento. O valor do divisor é configurado pelo registrador SPCR.

- A comunicação SPI é full-duplex, portanto ao mesmo tempo que o mestre envia um *byte* ao escravo (linha MOSI), o escravo envia um *byte* ao mestre (MISO).

- Ao escrever o byte no registrador SPDR (do mestre), o sinal de clock é iniciado e a comunicação começa. Ao mesmo tempo, o escravo envia um *byte* na linha MISO, se a linha SS está baixa (nível lógico zero). Nível alto desabilita a comunicação com o escravo.

Inicialização da SPI

- Exemplo de inicialização no modo mestre, com clock de SPI igual a clock de barramento / 16 .

```
void SPI_MasterInit(void)
{

/* Coloca MOSI e SCK como saída, outros pinos como
entrada, usando registradores de direção dos pinos */
    DDR_SPI = (1<<DD_MOSI)|(1<<DD_SCK);

 /* Habilita SPI, Mestre, com clock igual a fck/16 */
    SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);
}
```

Operação da SPI

Após a inicialização (que pode ser colocada em uma função. ex.: SPI_init()), pode-se escrever as funções para operar a comunicação.

Pode ser uma única função, pois a SPI é full-duplex (Tx e Rx simultâneos).

Ex.: void SPI_transmite_recebe (uint8_t *dado)

Operação da SPI

```
void SPI_Master_Transmite_Recebe(uint8_t *dado)
{
    /* Inicia transmissão */
    SPDR = *dado;

    /* Aguarda completar a transmissão */
    while(!(SPSR & (1<<SPIF)));

    /* Retorna dado recebido */
    *dado = SPDR;
}
```

Operação da SPI

No modo escravo, a transmissão deve ser feita na recepção.

```
void SPI_Slave_Init(void)
{
   /* Coloca MISO como saída, outros pinos como entrada */
   DDR_SPI = (1<<DD_MISO);

   /* Habilita SPI */
   SPCR = (1<<SPE);
}

void SPI_Slave_RecebeTransmite(uint8_t* dado)
{

    /* Armazena dado para transmissão */
    SPDR = *dado;

    /* Aguarda terminar repceção */
    while(!(SPSR & (1<<SPIF)));

    /* Retorna dado recebido */
    *dado = SPDR;
}
```

Programando a I²C

- A comunicação I²C (ou IIC, Inter Integrated Circuit) funciona em modo mestre/escravo, assim como a SPI.
  - Mestre sempre inicia a comunicação.
- Porém a comunicação I²C utiliza 2 linhas apenas (SDA e SCL):
  - SDA (Linha de dados bidirecional)
  - SCL (Linha de clock)
- Foi inventada pela Philips (hoje NXP), para comunicar vários CIs com menos conexões físicas.

Protocolo I²C

- A comunicação I²C é dividida em **quatro** etapas:
  - Sinal de INÍCIO (START)
  - Transmissão do endereço do escravo e direção dos dados
  - Transferência dos dados (escrita ou leitura)
  - Sinal de PARADA (STOP)

Protocolo I²C

- Regras gerais:
  - As linhas SCL e SDA ficam em **estado alto** (lógico 1) quando **não há comunicação** no barramento.
  - Os **dados (SDA)** no I²C só podem ser **alterados** (0 para 1, 1 para 0) quando o **SCL** está em nível **baixo**.
  - Os **dados** são **validados** (lidos na SDA) sempre na **transição do SCL de baixo para cima** (0 para 1).

Protocolo I²C

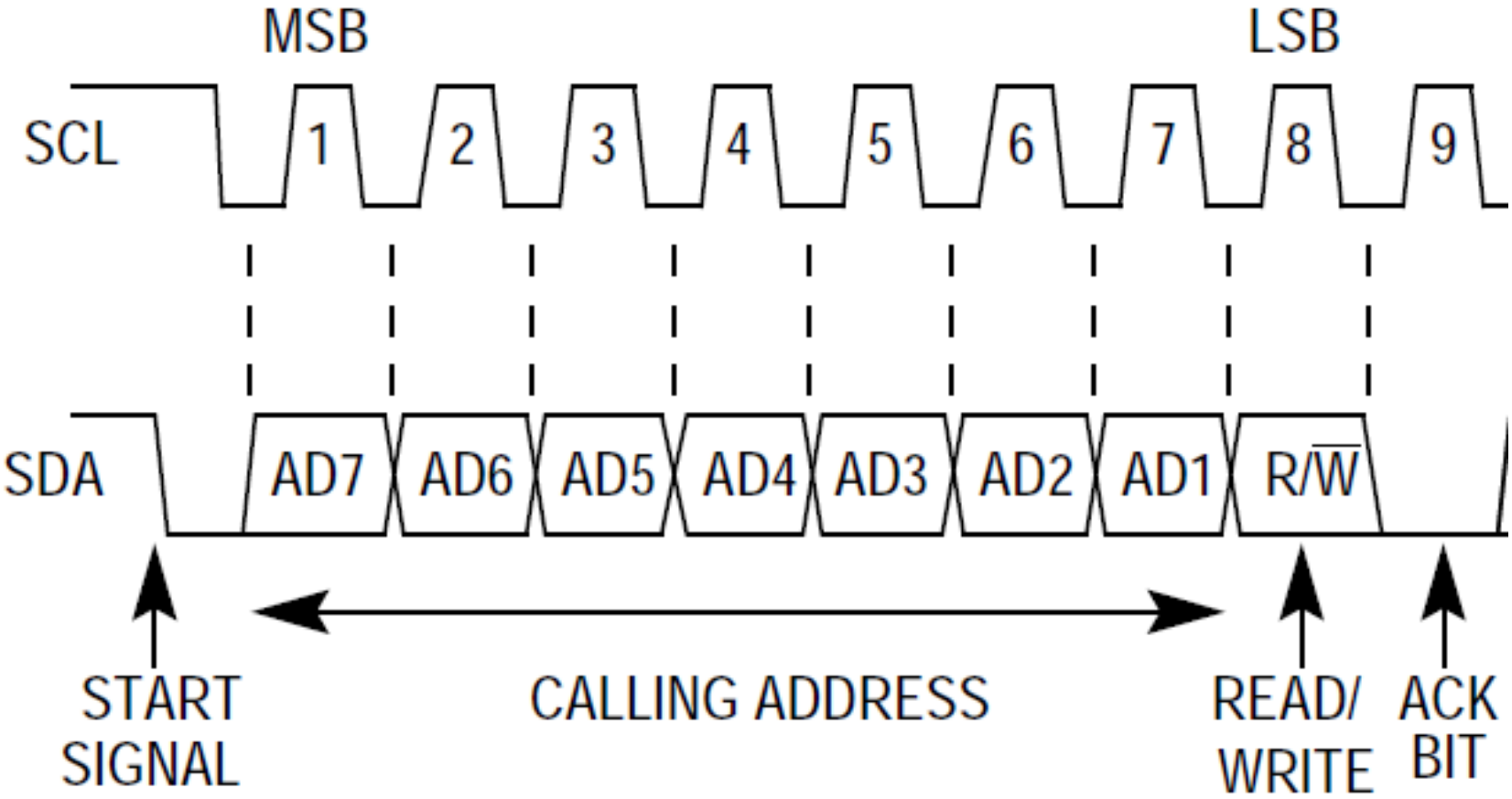
- Sinal de INÍCIO (**START**)
  - Se o SDA é baixado (1 para 0) pelo mestre enquanto a SCL está alta, o(s) escravo(s) interpreta(m) como um sinal de START.

Protocolo I²C

- Transmissão do **endereço do escravo** e **direção dos dados**
  - Na próxima etapa da comunicação, o mestre envia o **endereço de 7 bits** (pode ser 10 bits, no modo extendido) do escravo desejado. Cada escravo tem um **endereço único**.
  - O 8º bit é usado para indicar a direção dos dados:
    - Leitura (R) é bit 1 e escrita (W) é bit 0.
  - O 9º bit é denominado ACK (acknowledge, ou reconhecimento). O mestre dá um sinal de clock e o escravo deve escrever o bit 0 na SDA (baixar a linha).

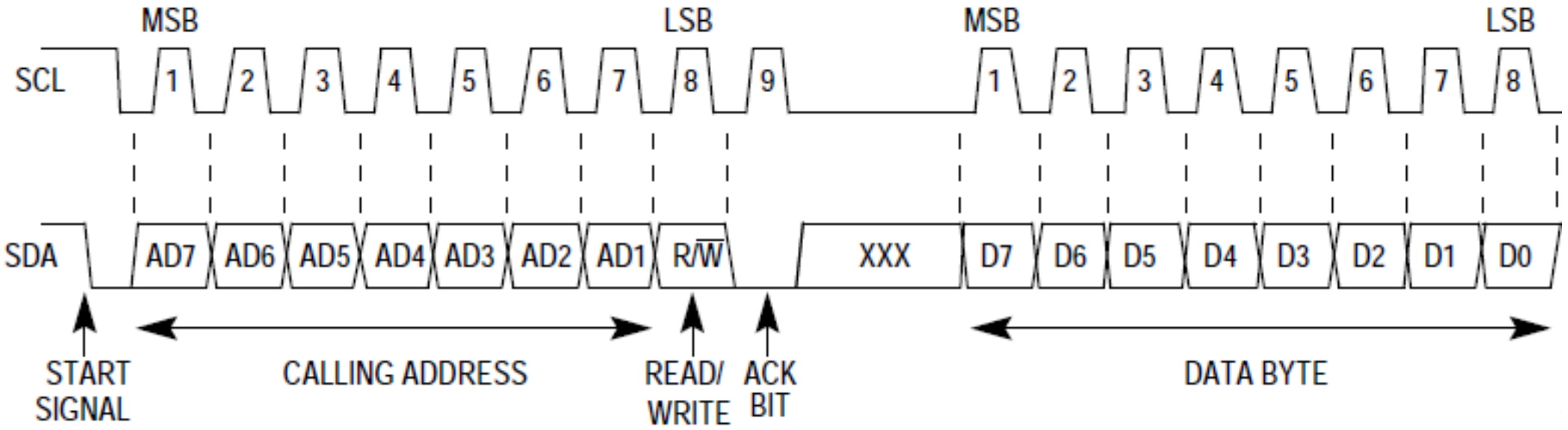
Protocolo I²C

- Transmissão do **endereço do escravo** e **direção dos dados**



Protocolo I²C

- Transferência dos dados (escrita ou leitura)
  - Na próxima etapa da comunicação, o mestre **envia ou recebe um byte** de dados. A direção (R ou W) é definida pelo bit enviado anteriormente.

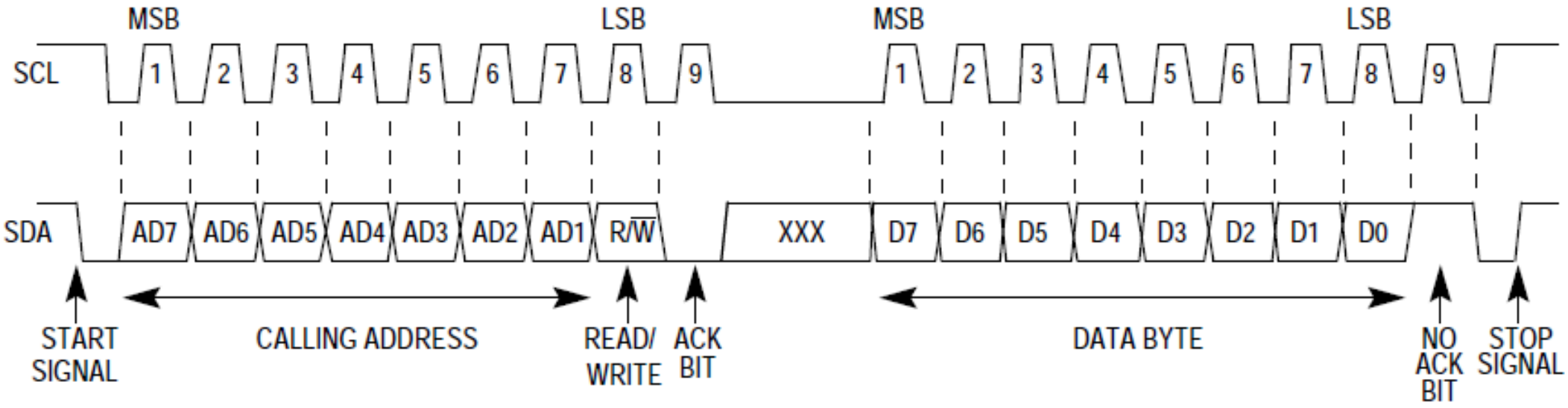


Protocolo I²C

- Transferência dos dados (escrita ou leitura)
  - O mestre pode **enviar ou receber vários bytes de dados**.
  - Cada byte é seguido do respectivo **bit de ACK** (9º bit), enviado pelo lado receptor.
  - Se **não houver o bit de ACK** (a linha SDA for deixada alta), o mestre interpreta como **falha** de transmissão, e o escravo interpreta como **fim da transferência** de dados.
  - Então, o mestre faz uma das seguintes opções:
    - Gera um sinal de STOP.
    - Gera um sinal repetido de START.

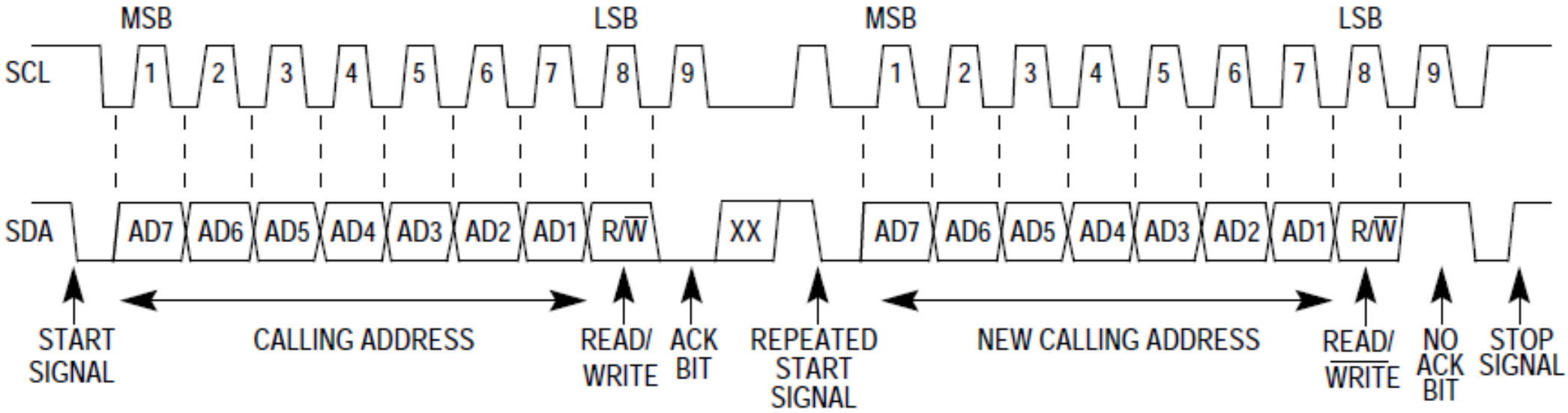
Protocolo I²C

- Sinal de PARADA (STOP)
  - Transição de baixo para cima da linha SDA com SCL alto.



Protocolo I²C

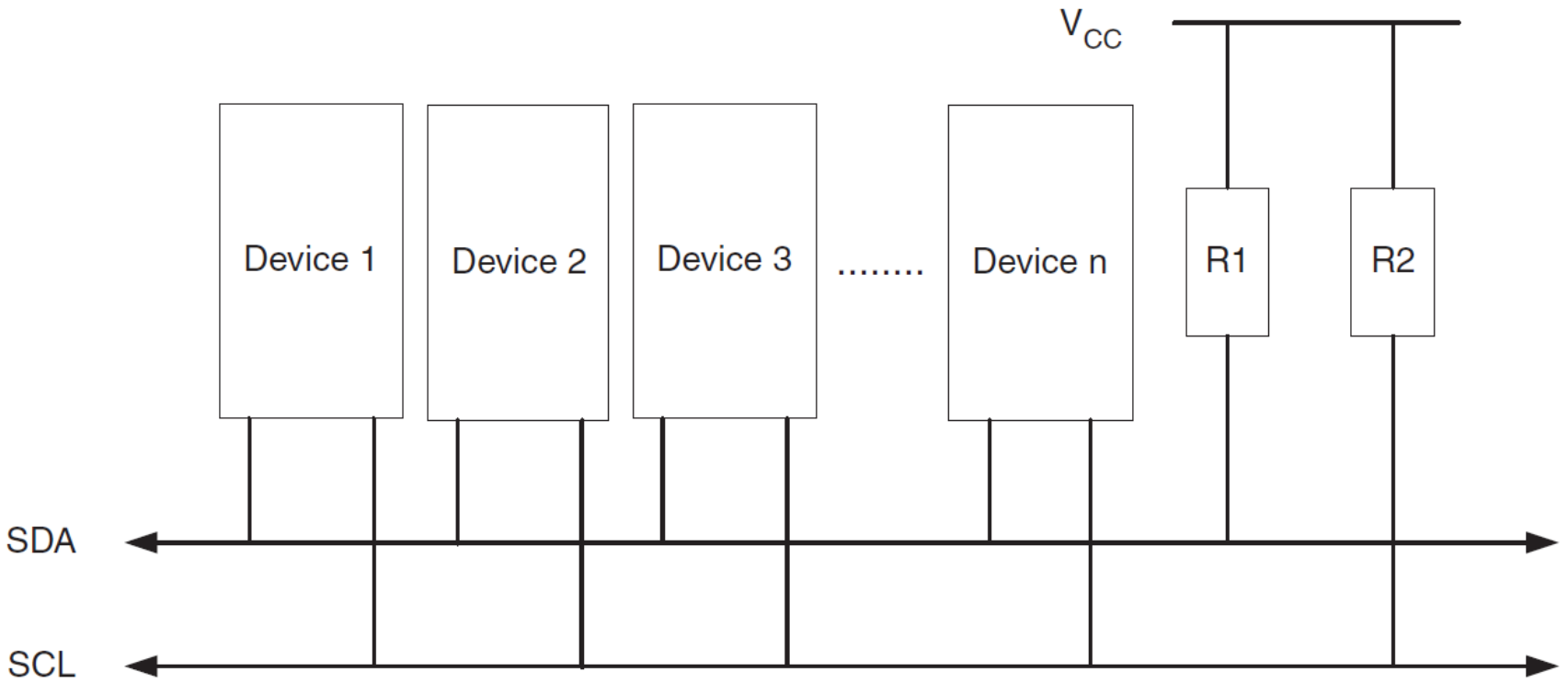
- Sinal repetido de START
  - O mestre também pode optar por repetir o START. Isto é, gerar um START sem gerar um STOP (p. ex.: para trocar a direção de dados com o mesmo escravo, ou comunicar com outro escravo)



Programando a I²C

No AVR, a comunicação IIC é feita pelo periférico TWI (2-wire Serial Interface)

Figure 22-1. TWI Bus Interconnection



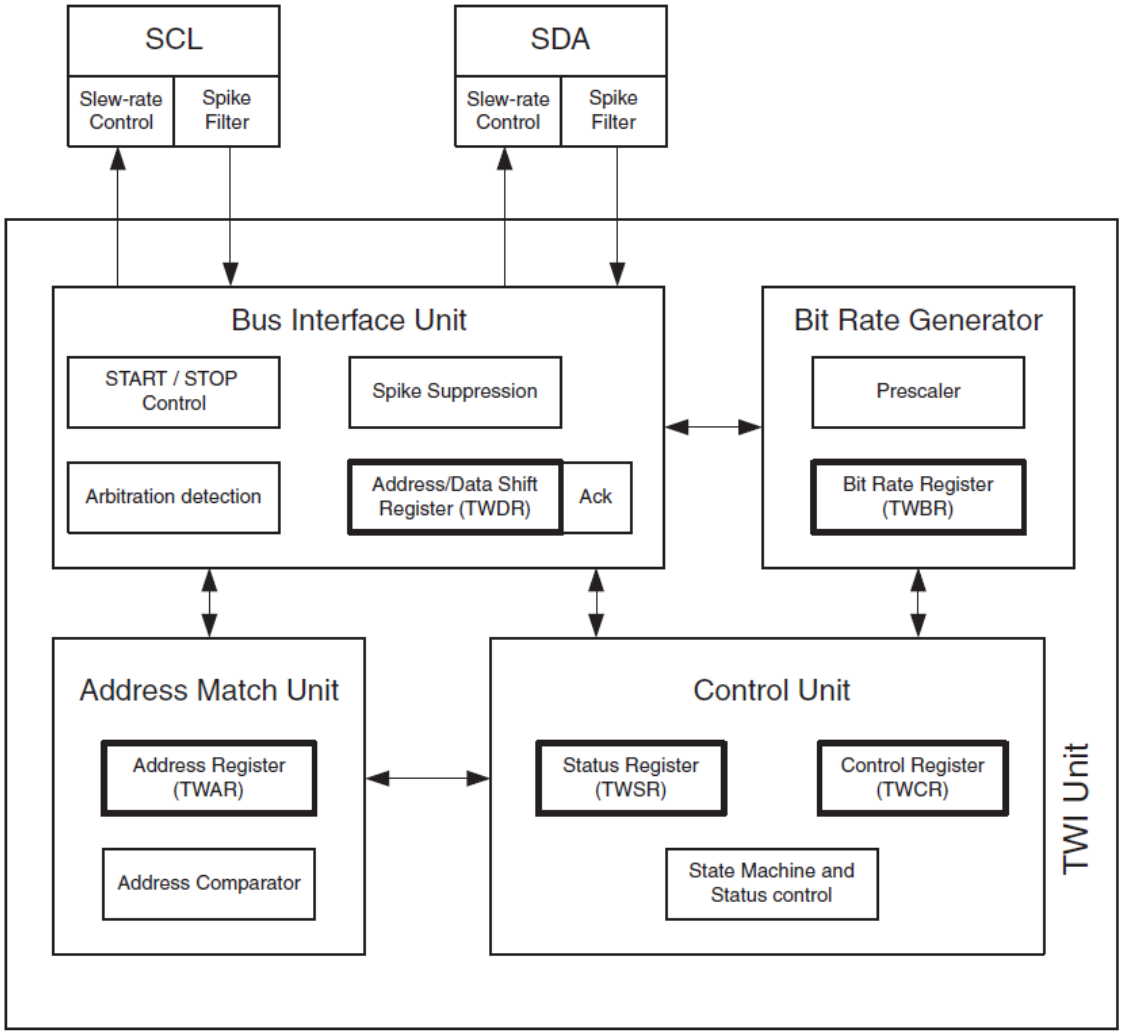
Programando a I²C

- Todo dispositivo I²C (mestre ou escravo) deve ter um endereço único.
- No AVR este endereço é programado no registrador TWAR.
- O dispositivo pode operar como mestre ou escravo (bit MST do registrador TWCR.
- Se operar como mestre, deve-se configurar a taxa de clock (baud rate), através do registrador TWBR.

Programando a I²C

- Visão geral do periférico TWI

Overview of the TWI Module



Inicialização da I²C

- Exemplo de inicialização da I²C no modo mestre, com clock de barramento a 1MHz para operar 100 kbps.

$$\text{SCL frequency} = \frac{\text{CPU Clock frequency}}{16 + 2(\text{TWBR}) \cdot (PrescalerValue)}$$

Table 22-7. TWI Bit Rate Prescaler

TWPS1	TWPS0	Prescaler Value
0	0	1
0	1	4
1	0	16
1	1	64

Inicialização da I²C

- Exemplo de inicialização da I²C no modo mestre, com clock de barramento a 1MHz para operar 100 kbps.

```
#define FOSC        1000000
#define BITRATE     10000
#define PRESCALER   1
void iic_init(void)
{
    TWBR = ((FOSC/BITRATE) - 16)/2/PRESCALER;
}
```

Operação da I²C

```
void iic_transmite(uint8_t end_escravo, uint8_t *dado)
{

    TWCR = (1<<TWINT)|(1<<TWSTA)|(1<<TWEN);   /* Habilita IIC e gera um START bit */
    while (!(TWCR &    (1<<TWINT))); /* Aguarda a geração do START bit */
    if ((TWSR & 0xF8) != 0x08) return;    /* Verifica se gerou START bit */

    TWDR = end_escravo;
    TWCR = (1<<TWINT) |(1<<TWEN);  /* Copia endereço para buffer e inicia a transmissão */
    while (!(TWCR &    (1<<TWINT))); /* Aguarda a transmissão */
    if ((TWSR & 0xF8) != 0x18) return;    /* Aguarda ACK da transmissão */

    TWDR = *dado;
    TWCR = (1<<TWINT) |(1<<TWEN);     /* Copia dado para buffer e inicia a transmissão */
    while (!(TWCR &    (1<<TWINT)));   /* Aguarda a transmissão */
    if ((TWSR & 0xF8) != 0x28) return;      /* Aguarda ACK da transmissão */
    TWCR = (1<<TWINT)|(1<<TWEN) | (1<<TWSTO); /* Gera um STOP bit */

    if ((TWSR & 0xF8) != 0x28) return;  /* Aguarda ACK da transmissão */
    TWCR = (1<<TWINT)|(1<<TWEN) | (1<<TWSTO);  /* gera um STOP bit */

}
```