Sistemas embarcados

Comunicações seriais:
SCI (UART), SPI, IIC (I^2C)

Programando a SCI

- Os módulos SCI do HCS08 possuem 8 registradores.
  - 2 reg. para configurar *baudrate* (SCIxBDH, SCIxBDL)
  - 3 reg. de configuração de opções (ex.: interrupções, erros, ...) (SCIxC1, SCIxC2)
  - 2 reg. de estado (ex.: flags) (SCIxS1, SCIxS2)
  - 1 reg. de dados (SCIxD) (escrita p/ TX e leitura p/ RX)

Inicialização da SCI

- Primeiro passo: calcular e escrever o valor dos registradores de taxa de baud (baud rate - BR). Ex.: serial 1 a 9600bps, clock de 20MHZ, BR=130.

  SCI1BDH = 0x00 e SCI1BDL=0x82 (130)

$$BR = \frac{(Clock\ de\ Barramento)}{(16\ x\ baud\ rate)}$$

SCIxBDH	LBKDIE	RXEDGIE		SBR12	SBR11	SBR10	SBR9	SBR8
SCIxBDL	SBR7	SBR6	SBR5	SBR4	SBR3	SBR2	SBR1	SBR0

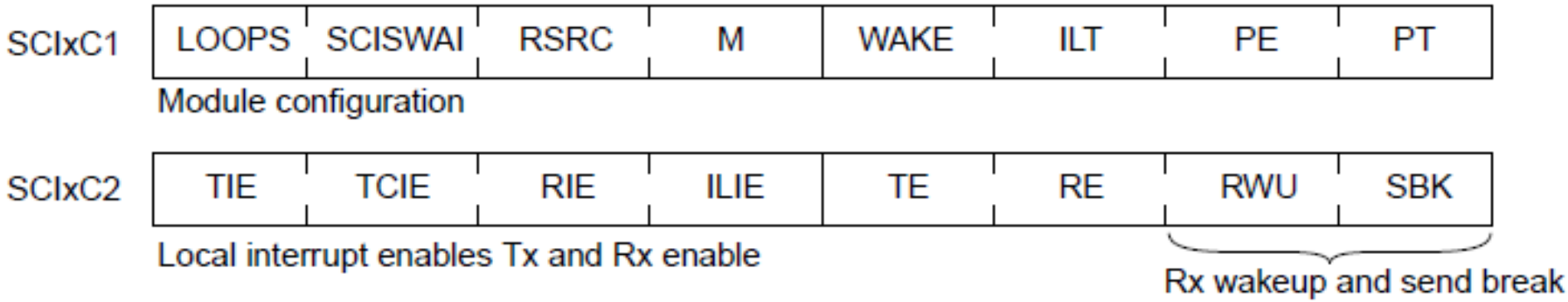
Baud rate = BUSCLK / (16 x SBR12:SBR0)

Inicialização da SCI

Segundo passo: **configurar modo de funcionamento** (8 ou 9 bits de dado, paridade, habilitar transmissor (Tx) e receptor (Rx), habilitar interrupções,...)

Ex.: SCI1C1 = 0x00 (porta serial 1, dado de 8 bits, sem paridade)

SCI1C2 = 0x0C (Tx e Rx habilitados, interrupções desabilitadas)



Operação da SCI

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

Ex.: void SCI_transmite (tipo_byte dado_a_enviar)

Ex.: void SCI_recebe (tipo_byte * dado_a_receber)

Operação da SCI

```
void SCI_transmite ( tipo_byte dado_a_enviar){

   // Espera o fim da transmissão do caractere
  anterior para transmitir o próximo
    while (!SCI1S1_TC);

   /* Armazena o caractere a ser transmitido no
  registrador de transmissão */
    SCI1D = dado_a_enviar;
}
```

Operação da SCI

```
interrupt void serial_rx(void){
  /* Leitura do registrador SCI1S1 para analisar o estado
  da transmissão */
  (void)SCI1S1;
  /* Leitura do registrador SCI1C3 para limpar o bit de
  paridade */
  (void)SCI1C3;
  // valor é uma variável do tipo byte declarada como
  global
   valor = SCI1D; /* Leitura dos dados recebidos */
 /* Usa uma “flag” para avisar a recepção. A “flag” é
  declarada como global e volatile */
   valor_recebido = 1;
}
```

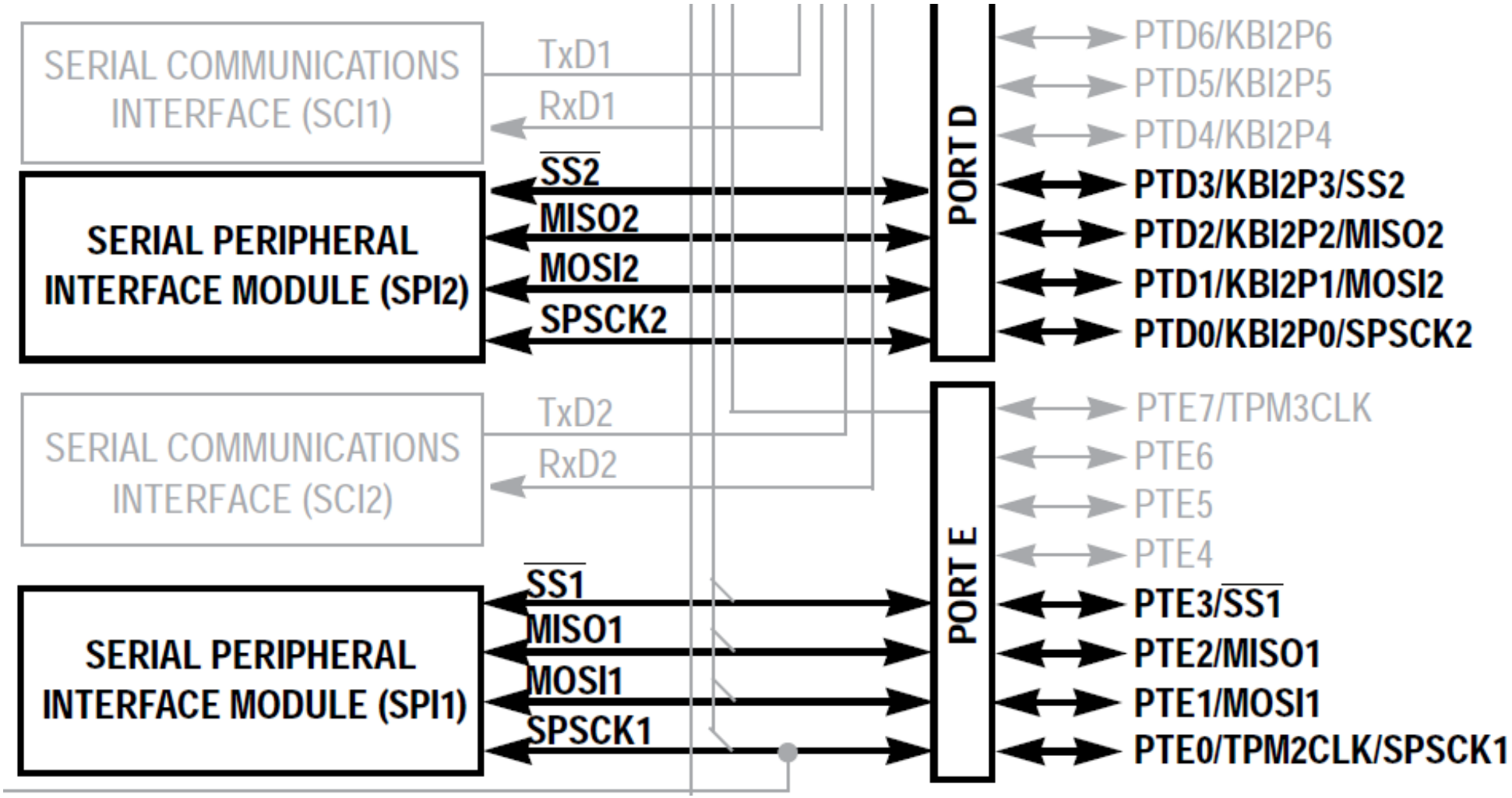
Operação da SCI

```
void SCI_recebe(tipo_byte *dado_a_receber){
  /* Usa uma “flag” para avisar a recepção. A “flag” é
  declarada como global e volatile */
  valor_recebido = 0;
  /* Habilita interrupção de Rx */
  SCI1C2 |= 0x20;
  while (!valor_recebido);
  // valor é do tipo byte declarada como global
  SCI1C2 &= ~0x20; /* Desabilita interrupção de Rx */
  *dado_a_receber = valor;
}
```

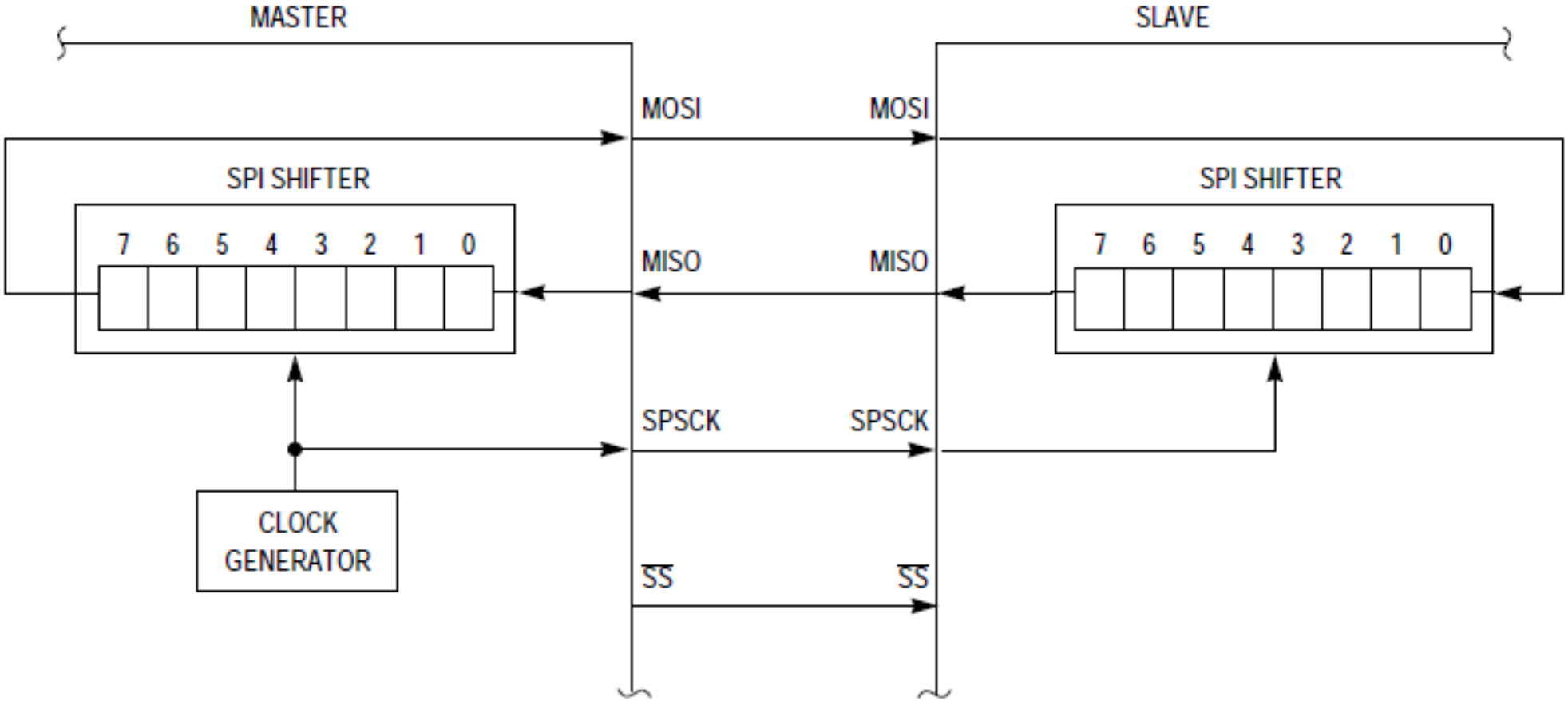
Programando a SPI

- 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



Programando a SPI

- A SPI é configurada usando os registradores de controle SPIxC1 e SPIxC2

	7	6	5	4	3	2	1	0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset	0	0	0	0	0	1	0	0

	7	6	5	4	3	2	1	0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W								
Reset	0	0	0	0	0	0	0	0

Programando a SPI

- A taxa de dados (baudrate) da SPI é configurada usando o registrador SPIxBR

	7	6	5	4	3	2	1	0
R	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
W								
Reset	0	0	0	0	0	0	0	0

- O estado da comunicação SPI é indicada pelo registrador SPIxS

	7	6	5	4	3	2	1	0
R	SPRF	0	SPTEF	MODF	0	0	0	0
W								
Reset	0	0	1	0	0	0	0	0

= Unimplemented or Reserved

Programando a SPI

- O dado a transmitir ou recebido fica armazenado no registrador de 8 bits SPIxD.
- O sinal de clock é configurado no mestre a partir da divisão do clock de barramento. O valor do divisor é configurado pelo registrador SPIxBR.
- 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 SPIxD (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 barramento de 24MHz, e clock de SPI igual a 6 MHz.

```
void SPI_init (void){
    SPIC1 = 0x00; // SPI desabilitada
    SPIC2 = 0x00; // SPI full-duplex, SS não usado
    SPIBR = 0x10; // pré-escala 2, divisor 2, total = 4
    (void) SPIS ; // limpa flags
    SPIC1 = 0x14; // SPI em modo mestre
    SPIC1 |= 0x40; // SPI habilitada
}
```

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 (tipo_byte *dado)

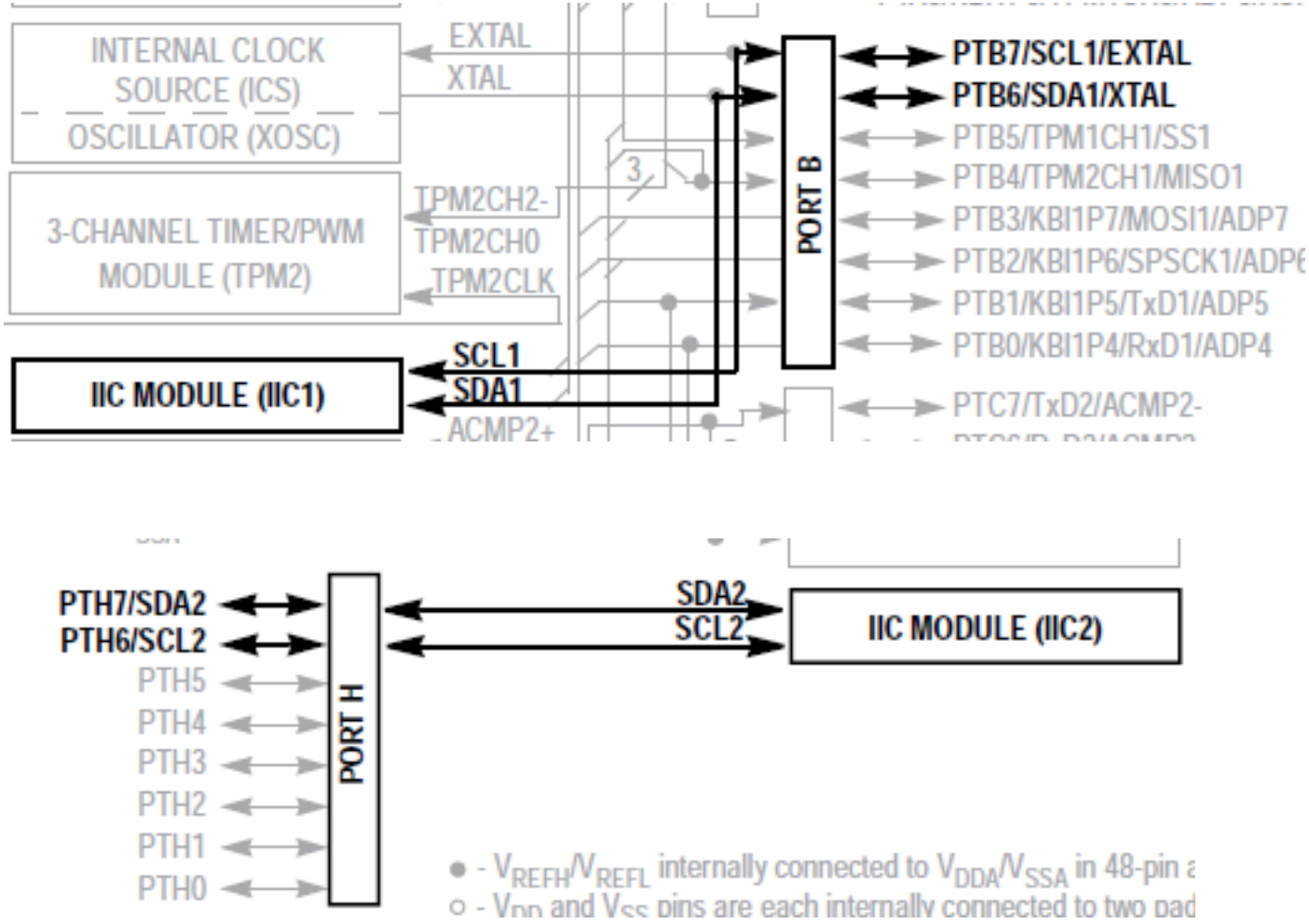
Operação da SPI

```
void SPI_transmite_recebe(tipo_byte *dado){
    while (!SPIS_SPTEF); // aguarda buffer de Tx vazio
    (void) SPIS ; // limpa flags
    SPID = *dado; // copia dado para buffer de Tx
    while (!SPIS_SPRF); // aguarda buffer de Rx cheio
    (void) SPIS ; // limpa flags
    *dado = SPID; // retorna dado recebido
}
```

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.

Programando a I²C



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

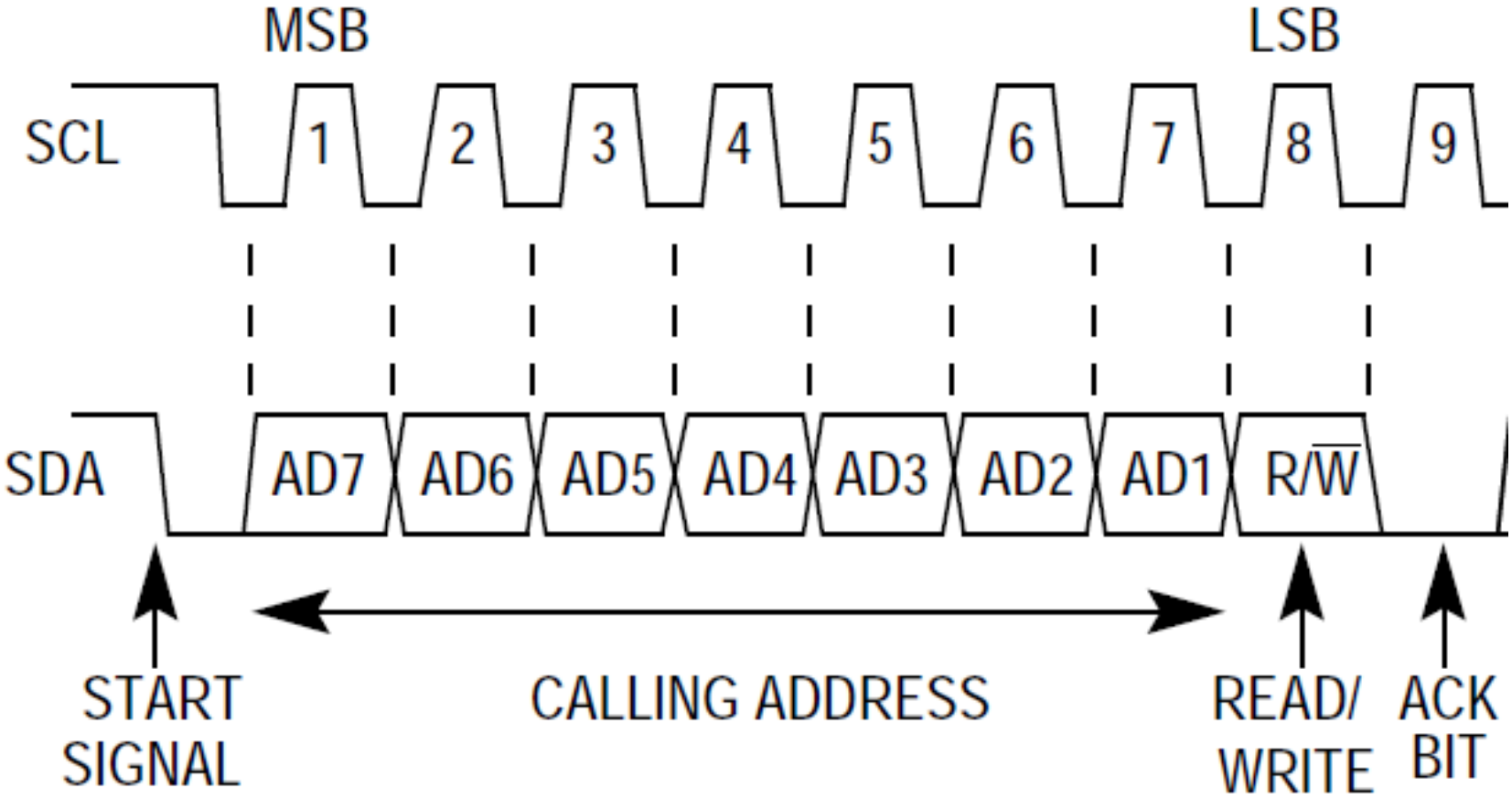
- Sinal de INÍCIO (**START**)
  - Se o SDA é baixado (1 para 0) pelo mestre enquanto a SCL está alta, o(s) escravo(s) interpreta(m) como um sinal de START.

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).

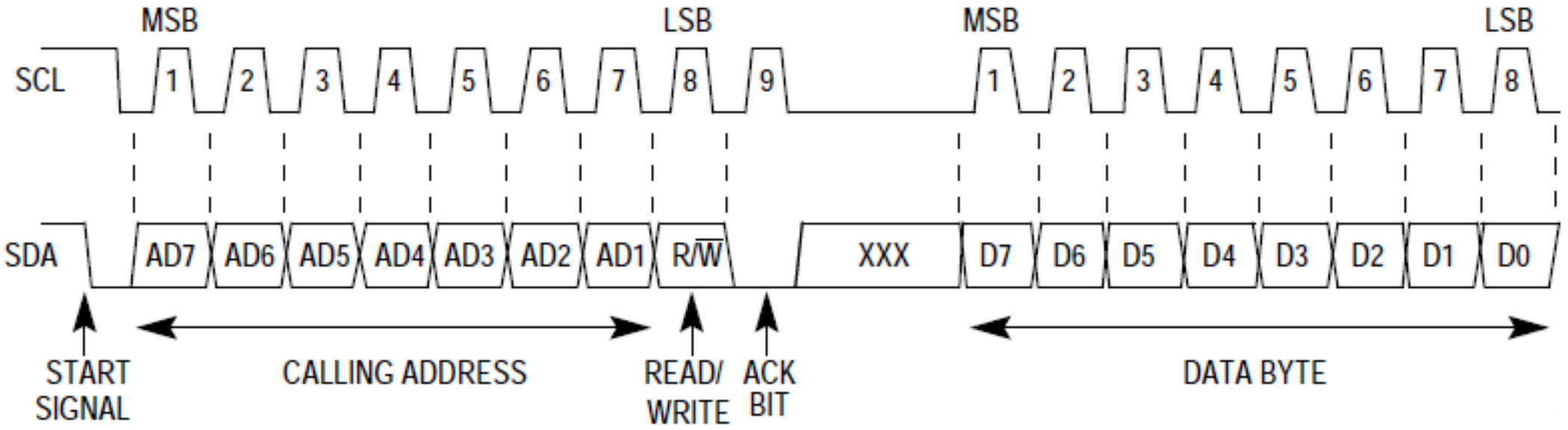
Protocolo I²C

- 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.

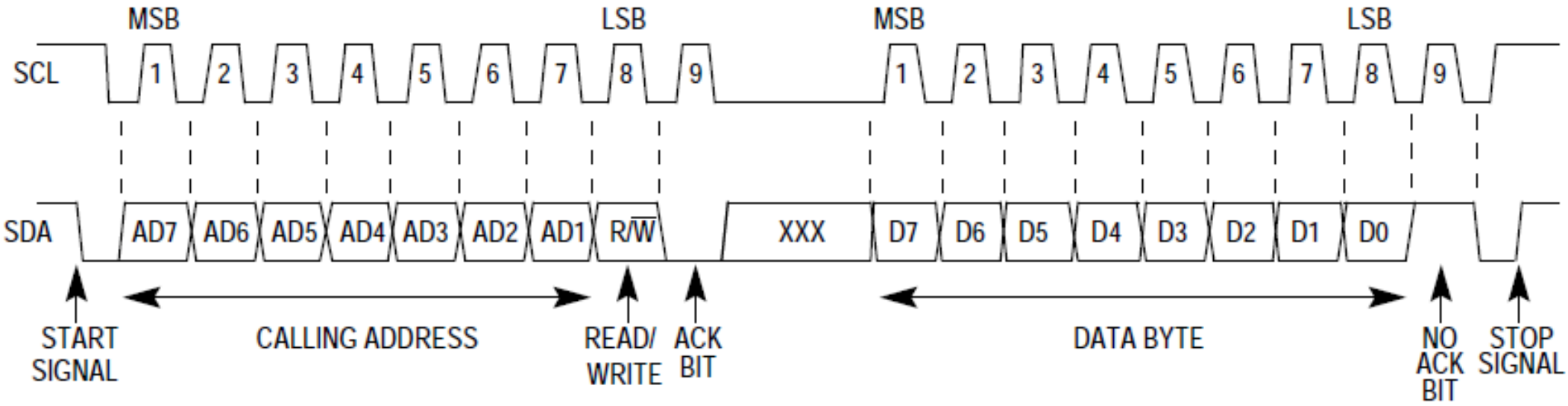


Protocolo I²C

- 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.

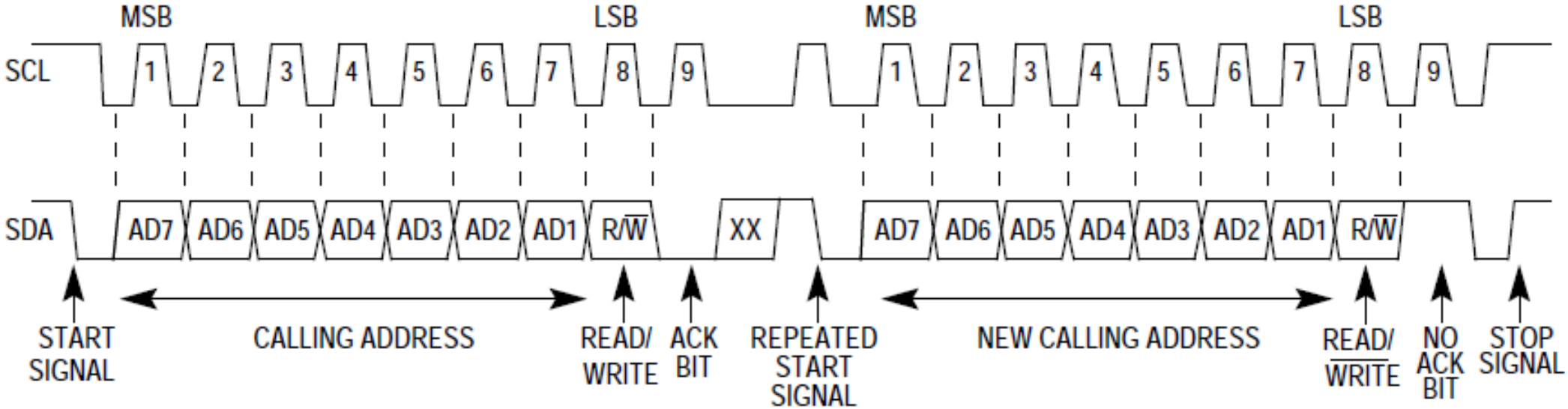
Protocolo I²C

- 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

- Todo dispositivo I²C (mestre ou escravo) deve ter um endereço único.
- No HCS08 este endereço é programado no registrador IICxA.
- O dispositivo pode operar como mestre ou escravo (bit MST do registrador IICxC1.
- Se operar como mestre, deve-se configurar a taxa de clock (baud rate), através do registrador IICxF.

Programando a I²C

- Resumo da configuração

IICA	AD[7:1]							0
Address to which the module will respond when addressed as a slave (in slave mode)								
IICF	MULT		ICR					
Baud rate = BUSCLK / (2 x MULT x (SCL DIVIDER))								
IICC1	IICEN	IICIE	MST	TX	TXAK	RSTA	0	0
Module configuration								
IICS	TCF	IAAS	BUSY	ARBL	0	SRW	IICIF	RXAK
Module status flags								
IICD	DATA							
Data register; Write to transmit IIC data read to read IIC data								
IICC2	GCAEN	ADEXT	0	0	0	AD10	AD9	AD8
Address configuration								

Inicialização da I²C

- Exemplo de inicialização da I²C 1 no modo mestre, com clock de barramento de 24MHz, e clock de I²C igual a 100 kbps.

```
void IIC_init (void){
    IIC1F = 0x1F; // SCL divisor = 240, mul = 01
    IIC1C1 = 0x80; // habilita IIC
    IIC1C1 |= 0x18; // habilita Tx sem ACK automático
    (void) IIC1S ; // limpa flags
    IIC1C1 |= 0x20; // IIC em modo mestre
    (void) IIC1S ; // limpa flags
}
```

Operação da I²C

```
void IIC_transmite(tipo_byte end_escravo,tipo_byte *dado){
   while (IIC1S_BUSY); // aguarda liberar barramento
   do{
      while (!IIC1S_TCF); // aguarda completar Tx
      IIC1D = (end_escravo & 0xFE); // Tx endereço do escravo - W
    }while(IIC1S_RXAK); // aguarda bit de ack
   do{
      while (!IIC1S_TCF); // aguarda completar Tx
      IIC1D = (*dado); // Tx dado
   }while(IIC1S_RXAK); // aguarda bit de ack
}
```

Operação da I²C

```
void IIC_recebe(tipo_byte end_escravo,tipo_byte *dado){
   while (IIC1S_BUSY); // aguarda liberar barramento
   do{
      while (!IIC1S_TCF); // aguarda completar Tx
      IIC1D = (end_escravo | 1); // Tx endereço do escravo - R
    }while(IIC1S_RXAK); // aguarda bit de ack

   IIC1C1_TX = 0; // modo Rx
   while (!IIC1S_TCF); // aguarda completar Tx
   *dado = IIC1D; // Tx dado
    IIC1C1_TX = 1; // modo Tx
}
```