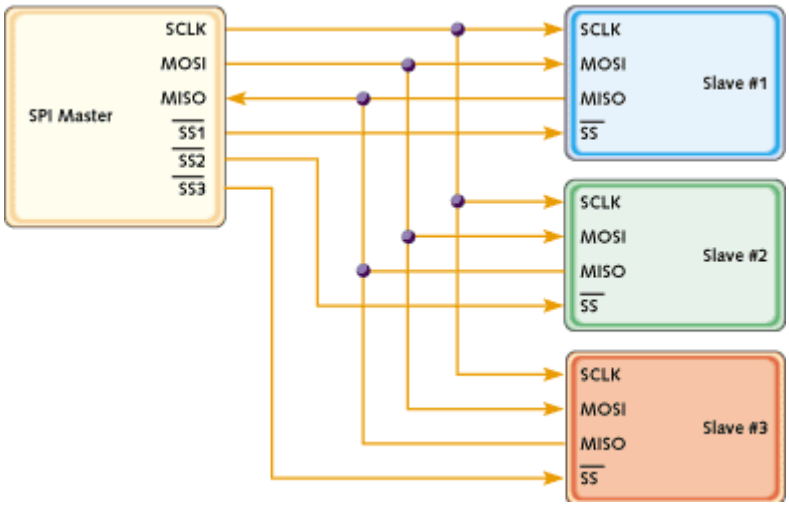
Protocolo SPI

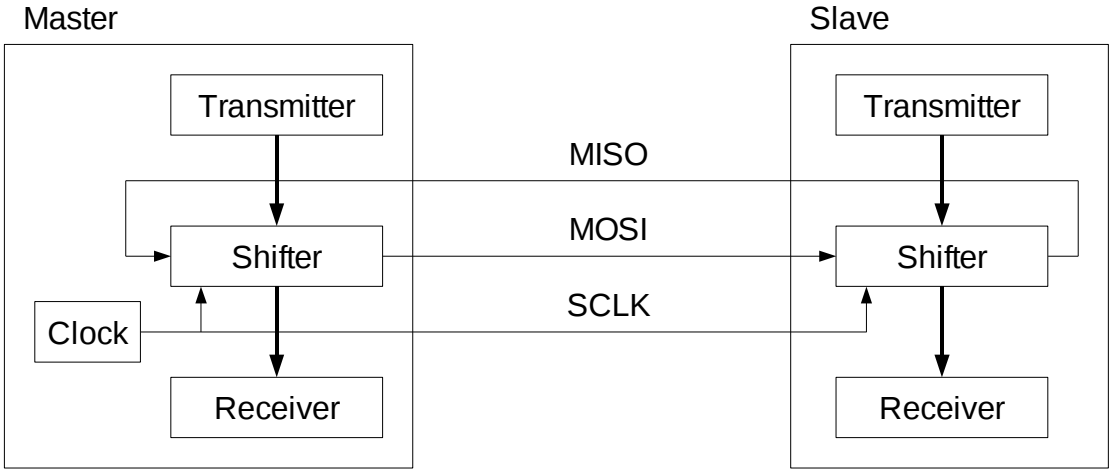
O protocolo SPI (**S**erial **P**eripheral **I**nterface bus) foi definido pela Motorola e é utilizado para ligar o processador a periféricos no âmbito de uma placa electrónica. Cobre o mesmo campo de aplicação que o protocolo I2C e suporta ritmos superiores.

A interface física é composta pelos sinais MOSI, MISO e SCLK que ligam, em barramento, a todos os intervenientes, e pelos sinais de selecção SSn, um por cada periférico.

A transferência de dados processa-se em *full-duplex* sobre os sinais MOSI (*Master Output Slave Input* MISO) e MISO (*Master Input Slave Output*). O ritmo de transferência é definido pelo processador, no papel de master, através do sinal SCLK. A selecção do periférico é feita através do sinal SSn.



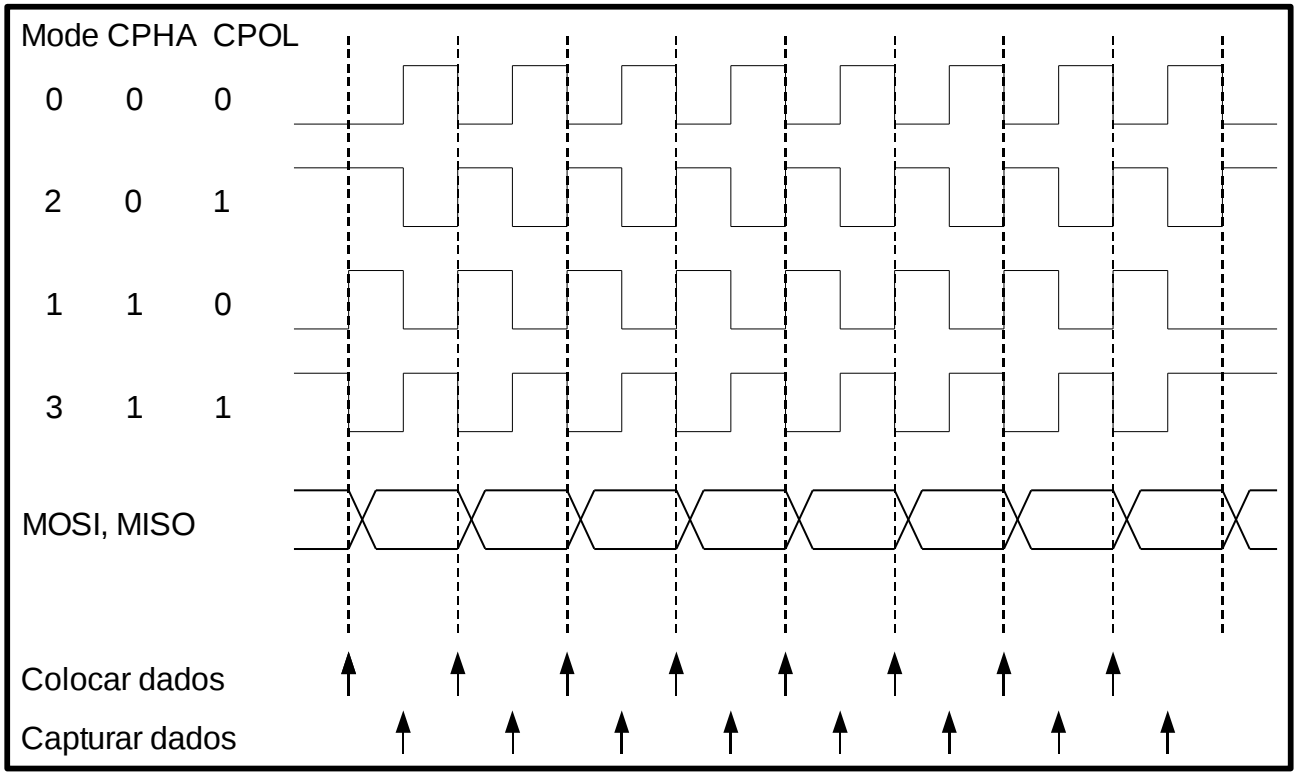
A figura seguinte ilustra o principio de funcionamento, quanto ao modo como se processa a transferência dos dados. Em ambos os lados existe um registador deslizante, onde são colocados, em paralelo, os dados a transmitir em série e de onde são recolhidos, em paralelo, os dados recebido em série. Por cada impulso de relógio o *master* recolhe um bit de dados no sinal MISO e coloca um bit de dados no sinal MOSI.



O fluxo é controlado pelo *master* através do sinal de relógio SCLK. Só há transferência enquanto o *master* pulsar o sinal SCLK. Em em repouso o sinal SCLK encontra-se estável com o valor lógico definido por CPOL.

A transferência de dados é sempre realizada em *full-duplex*, por isso o SPI é um protocolo muito eficiente. Se se pretender transferir dados num único sentido, o dispositivo no papel de receptor terá que transmitir dados fictícios e o emissor simplesmente ignorar os dados recebidos.

Existem quatro variantes quanto ao posicionamento relativo do sinal de relógio em relação aos sinais de dados. Cada uma destas variantes designa-se por modo e corresponde às quatro combinações dos parâmetros CPHA e CPOL. Quando o parâmetro CPHA é zero, os dados são capturados na primeira transição de relógio, quando CPHA é um, os dados são capturados na segunda transição do relógio. O parâmetro CPOL define a polaridade do sinal em repouso e como consequência o sentido das transições durante a transferência. Em todos os casos aplica-se o seguinte princípio: os dados são capturados pelo receptor na transição de relógio oposta à que são colocados pelo transmissor. Dispõe-se assim de meio período de relógio para estabilização dos dados.



Muitos microcontroladores possuem controladores SPI integrados que podem operar no papel de *master* ou *slave*. Os controladores SPI permitem, geralmente, definir os seguintes parâmetros:

- o número de bits que formam as palavras (ex: 8 a 16);
- o ritmo do clock;
- os sinais de selecção dos periféricos - nSS;
- as pausas nas mudanças de periféricos;
- o modo de operação – mestre ou escravo;
- a fase (CPHA) e a polaridade (CPOL).

Uma transferência é sempre realizada pela iniciativa do *master,* tem uma duração previsível e sucede sempre sem erros. Se o periférico não existir a transacção processa-se na mesma e terão que ser as camadas superiores do software a detectar.

Uma consequência das transferências serem da iniciativa do *master* é não existirem acontecimentos assíncronos, como por exemplo a a recepção de dados.

Interface de programação

Para representar e comunicar com periféricos por SPI é definida a interface de programação apresentada à frente.

A **`struct Spi_device`** representa um periférico e contém a informação necessária para interactuar com ele.

```
typedef struct {
     int device;
     int clock;
     int mode;
     int nbits;
} Spi_device;
```

O campo **`device`** define a identificação do periférico.

O campo **`clock`** define o ritmo do sinal de relógio.

O campo **`modo`** definem o modo SPI (CPHA, CPOL).

O campo **`nbits`** define o número de bits de uma palavra.

A interface de programação é composta por funções de transferência de dados, para palavras individuais e para blocos, e pelas funções de manipulação da interface.

A função **`init`** deve ser chamada apenas uma vez, no início do programa, para preparar a infraestrutura de comunicação com dispositivos SPI.

A função **`spi_transaction_begin`** além de activar o sinal de selecção do periférico, programa o ritmo do relógio e o modo do protocolo (CPOL e CPHA). No mesmo sistema podem coabitar periféricos com diferentes modos e diferentes velocidades. A função **`spi_transaction_end`** desactiva o sinal de selecção.

A função de transferência de dados **`spi_transaction_transfer`** pode ser chamada sucessivamente entre **`spi_transaction_begin`** e **`spi_transaction_end`**. para formar uma mensagem completa. A formatação da mensagem depende do periférico em causa. Em caso de comunicação unidireccional, um mesmo *buffer* pode ser usado nos parâmetros **`tx_data`** e **`rx_buffer`**, sendo o seu conteúdo sempre modificado.

```
void spi_init(Spi_device *, int);
void spi_transaction_begin(Spi_device *);
void spi_transaction_end(Spi_device *);
void spi_transaction_transfer(Spi_device *, size_t size, const U8 * tx_data, U8 * rx_buffer);
```

Implementação por software no LPC2106

A iniciação consiste em desactivar os sinais de selecção dos periféricos e programar, no GPIO, os pinos usados para MOSI e SCLK como saídas e o usado para MISO como entrada.

```
void spi_init(Spi_device * spis, int n_spis) {
     int i;
     for (i = 0; i < n_spis; ++i) {
          io_write_u32(LPC210X_GPIO + LPC2XXX_IOSET, 1 << spis[i].device);
          U32 aux = io_read_u32(LPC210X_GPIO + LPC2XXX_IODIR);
          io_write_u32(LPC210X_GPIO + LPC2XXX_IODIR, aux | (1 << spis[i].device));
```

```
    }
    U32 aux = io_read_u32(LPC210X_GPIO + LPC2XXX_IODIR) & ~(1 << MISO_PIN);
    io_write_u32(LPC210X_GPIO + LPC2XXX_IODIR, aux | 1 << MOSI_PIN | 1 << SCK_PIN);
}
```

Na função de inicio de transação – **spi_transaction_begin** – são executadas as seguintes operações:

- colocar o sinal SCLK com a polaridade e correcta (CPOL);
- activar o sinal de selecção;

```
void spi_transaction_begin(Spi_device * spi) {
    io_write_u32(LPC210X_GPIO +
        ((spi->mode & 2) ? LPC2XXX_IOSET : LPC2XXX_IOCLR), 1 << SCK_PIN);
    io_write_u32(LPC210X_GPIO + LPC2XXX_IOCLR, 1 << spi->device);
}
```

Na função **spi_transaction_end** desactiva-se apenas o sinal de selecção do periférico.

```
void spi_transaction_end(Spi_device * spi) {
    io_write_u32(LPC210X_GPIO + LPC2XXX_IOSET, 1 << spi->device);
}
```

Na função de transferência – **spi_transaction_transfer** – há a preocupação de minimizar o processamento executado no ciclo **for** para maximizar a velocidade de transferência.

Ao colocar nas variáveis **clk_start** e **clk_middle** os endereços **LPC2XXX_IOCLR** ou **LPC2XXX_IOSET**, determina-se o sentido da transição do sinal SCLK, para o início e para o meio do bit time.

Em cada ciclo são executadas as seguintes operações:

- efectuar a transição inicial do relógio;
- colocar os dados a transmitir no sinal MOSI;
- efectuar a transição intermédia do relógio;
- recolher os dados a receber do sinal MISO.

Se os portos de saída fossem baseados em *flip-flops* do tipo D, em vez de quatro operações de escrita poderiam ser realizadas apenas duas. Na primeira activava-se a transição inicial e colocavam-se os dados e na segunda activava-se apenas a transição intermédia.

Num dispositivo hardware os dados a receber são capturados pela transição intermédia do sinal de relógio. Em software, como não é possível executar a transição intermédia em simultâneo com a captura dos dados, optou-se por capturar os dados após a transição intermédia.

```
void spi_transaction_transfer(Spi_device * spi, size_t size, const U8 * tx_data, U8 * rx_buffer) {
    int i, nbits = spi->nbits;
    U32 clk_start, clk_middle, data = 0;
    if ((spi->mode == 0) || spi->mode == 3) {
```

```
            clk_start  = LPC210X_GPIO + LPC2XXX_IOCLR;
            clk_middle  = LPC210X_GPIO + LPC2XXX_IOSET;
      } else {
            clk_start  = LPC210X_GPIO + LPC2XXX_IOSET;
            clk_middle = LPC210X_GPIO + LPC2XXX_IOCLR;
      }
      while (size-- > 0) {
            U8 value = * tx_data++;
            for (i = 0; i < nbits; ++i) {
                  U32 mosi = (value >> (nbits - i - 1)) & 1;
                  io_write_u32(clk_start, 1 << SCK_PIN);
                  io_write_u32(LPC210X_GPIO + LPC2XXX_IOSET,  mosi << MOSI_PIN);
                  io_write_u32(LPC210X_GPIO + LPC2XXX_IOCLR, (~mosi & 1) << MOSI_PIN);
                  io_write_u32(clk_middle, 1 << SCK_PIN);
                  data = (data << 1) | (io_read_u32(LPC210X_GPIO + LPC2XXX_IOPIN) >> MISO_PIN & 1);
            }
            *rx_buffer++ = data;
      }
      io_write_u32(clk_start, 1 << SCK_PIN);
}
```

Referências:

Designing Embedded Hardware, Capítulo 7

http://embedded.com/columns/beginerscorner/9900483?_requestid=608596

LPC2104/2105/2106 User Manual, Capítulo 12