Prática 8:

Comunicação SPI

8.1 – Introdução e objetivos

O protocolo SPI (*serial peripheral interface*) é um método de comunicação utilizado para interconectar dispositivos ou chips de tal forma que estes troquem informações entre si. Inicialmente, seu propósito era ser um protocolo de comunicação entre periféricos (câmeras, impressoras, scanners, etc) e o computador, por exemplo. Contudo, este nicho de aplicação foi largamente tomado por outra tecnologia (a USB - *universal serial bus*, vista em práticas posteriores). Mesmo assim, o SPI é ainda amplamente utilizado para troca de dados entre chips ou CI´s (circuitos integrados) sendo um importante método de comunicação para sistemas digitais de alguma complexidade que envolvam, principalmente, conversores AD e DA, memórias EEPROM e flash, *real time clocks* (RTC), sensores com interface de comunicação, cartões de memória, controladores LCD e unidades de transmissão e recepção de dados UART.

O SPI adota um esquema de comunicação do tipo mestre/escravo e pode ser executado em modo "full duplex" (e.g., cada componente executando o SPI pode receber e transmitir dados ao mesmo tempo). Vale destacar que mesmo sendo um protocolo largamente difundido, ele não é padronizado por nenhuma entidade e por isto pode haver variações de uso dependendo do fabricante que emprega este protocolo. Em relação ao seu grande "rival", o protocolo I2C (também visto em práticas posteriores), o SPI tem a vantagem de se comunicar com maior rapidez e quando temos vários dispositivos intercomunicando-se em um barramento compartilhado, o SPI não requer um esquema de endereçamento de dispositivos tão elaborado quanto o I2C.

8.2 – Princípios básicos do protocolo SPI

O SPI requer 4 fios como descrito na Tabela 7.1. O dispositivo mestre, como sugere o nome, controla a comunicação. Através do controle do clock, ele decide quando o dado é enviado dele ou é recebido por ele. Dentro de um mesmo ciclo de clock, uma comunicação full duplex pode ser realizada com o mestre que pode enviar dados para um escravo e receber dados dele ou de outro dispositivo simultaneamente. Usando o sinal SS, o mestre pode escolher com qual dos escravos se comunicar. A Figura 7.1 ilustra a arquitetura de ligação entre dispositivos que emprega este protocolo.

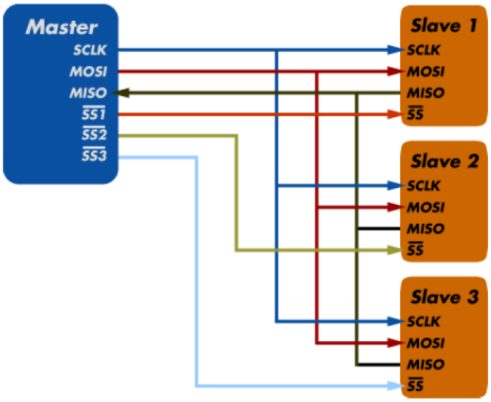


Figura 7.1 - Esquema de ligação entre dispositivos mestre e escravo para o protocolo SPI.

Tabela 8.1 - Fios do barramento SPI

Linha	Nome	Descrição
SCLK	Serial clock	É o clock gerado pelo mestre (os escravos apenas leem esta sinal) utilizado para fazer a sincronização dos dados.
MOSI ou DO	Master output, slave input ou data output	É o pino reservado para enviar dados seja o chip mestre ou escravo.
MISO ou DI	Master input, slave output ou data input	É o pino reservado para receber dados seja o chip mestre ou escravo.
SS ou CS	Slave select ou select device ou chip selec	Geralmente é ativo baixa e indica no barramento qual dispositivo escravo tem permissão para enviar/receber dados no barramento de dados. Este fio é opcional em um sistema com um único escravo

O protocolo estipula ainda quatro métodos de transmissão e sincronização de dados entre transmissor e receptor usando o sinal de clock. Assim, o sinal de clock pode ter sua fase e polaridade configuradas de formas diferentes para gerar os 4 modos de transmissão. A polaridade dizer respeito ao sinal CPOL mostrado na Figura 7.2 e a fase a CPHA mostrada no mesmo modo. A Tabela 7.2 mostra como são definidas as quatro combinações destes modos a partir de CPOL e CPHA.

Tabela 7.1 - Modos de transmissão SPI

Modo	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

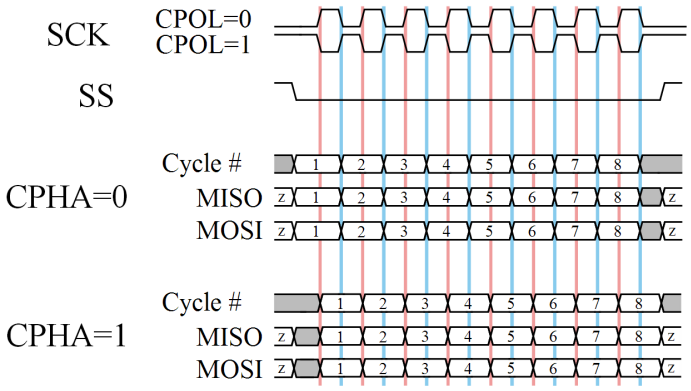


Figura 7.2 - Diagrama ilustrando os diferentes modos de sincronização entre mestre e escravo para transmissão de dados.

O diagrama de tempo da Figura 7.2 ajuda a ilustrar como funcionam estes modos.Quando CPOL=0, temos um sinal de clock similar ao ilustrado na figura do exemplo. Assim, se CPHA=0, os dados são lidos na borda de subida do clock e os dados são atualizados no barramento na borda de descida. Já para CPHA=1 (ainda considerando CPOL=0), os dados são lidos na borda de descida e a saída é atualizada na borda de subida. Por fim, quando CPOL=1, a referência do clock é invertida como mostra a Figura 7.2 gerando uma diferença de fase. CPHA age do mesmo modo como no caso anterior.

8.3 – Exemplo de aplicação

Para ilustrar um exemplo versátil de SPI usando as bibliotecas SPI do CCS e também o hardware SPI embutido no 18F4550, será produzida uma aplicação onde:

- são usados 2 PICs 18F4550 sendo um o dispositivo mestre e o outro escravo;

- o PIC mestre lê de três chaves (S2, S3 e S4) um valor de 3 bits. Estes dados são enviados via SPI para o PIC escravo.
- o estado de uma outra chave S1 também é enviada em um outro pacote de dados para o PIC escravo. Esta chave S1 indica um comando onde: S1=0 deve-se ligar um LARANJA do PIC escravo ou, quando S1=1, o escravo deve pegar o valor de S2S3S4 para um conjunto de 3 LEDs.
- uma chave S8 é usada para indicar ao mestre que existe um dado ou comando novo que deve ser lido pelo mestre e enviado por SPI ao escravo que irá ler o comando e realizar a operação indicada pelo comando.

As próxima seções descrevem esquematicamente o circuito e um exemplo de código fonte para esta aplicação. A ideia é demonstrar como acontece a comunicação SPI entre dois dispositivos. O PIC escravo poderia ser substituído por qualquer outro chip capaz de realizar a comunicação SPI. Mesmo assim foi preferido um emprego de um PIC como escravo para mostrar com mais detalhes como funciona a recepção de dados em um dispositivo escravo.

8.3.1 - O circuito

A Figura 7.3 mostra o diagrama elétrico da aplicação. Dados e comandos são lidos de uma chave DIP (*dual in package*) *switch* de 8 vias, enviados pelo PIC mestre e interpretados pelo PIC escravo segundo a sequência vista na figura (S1 indica comando; S2S3S4 indica um valor de 3 bits e S8 aciona uma interrupção externa no mestre para que ele possa ler os novos valores de S1 a S4 e enviá-los ao escravo).

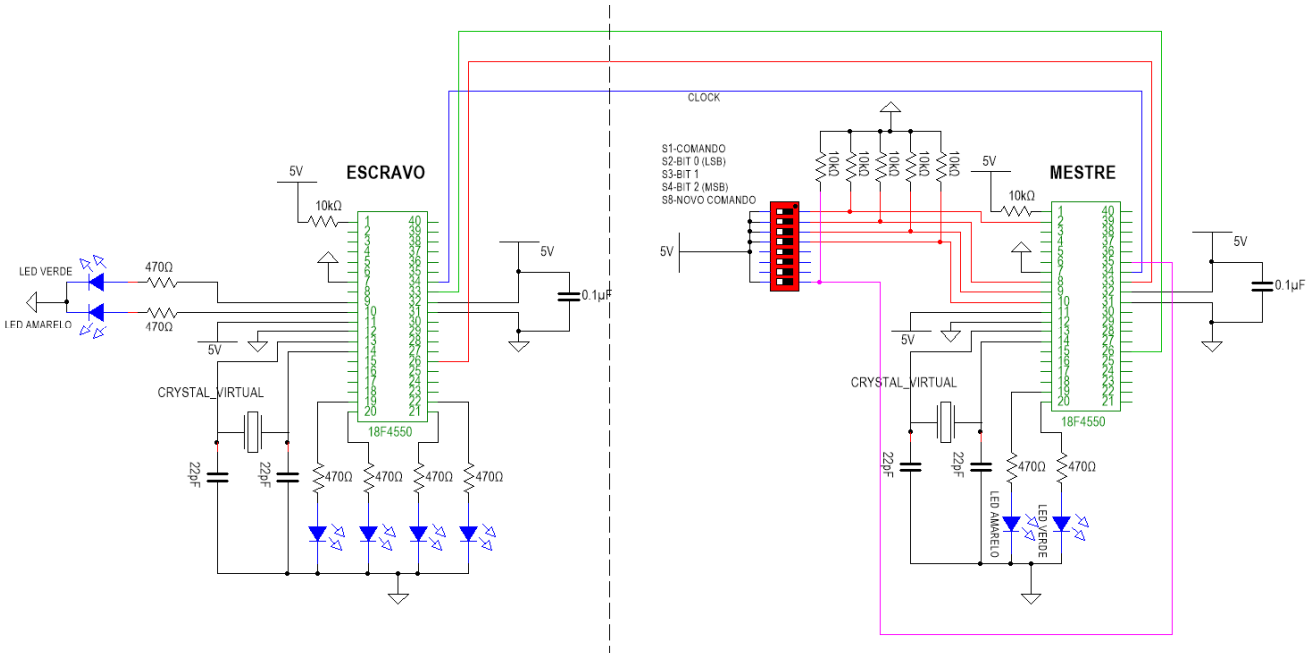


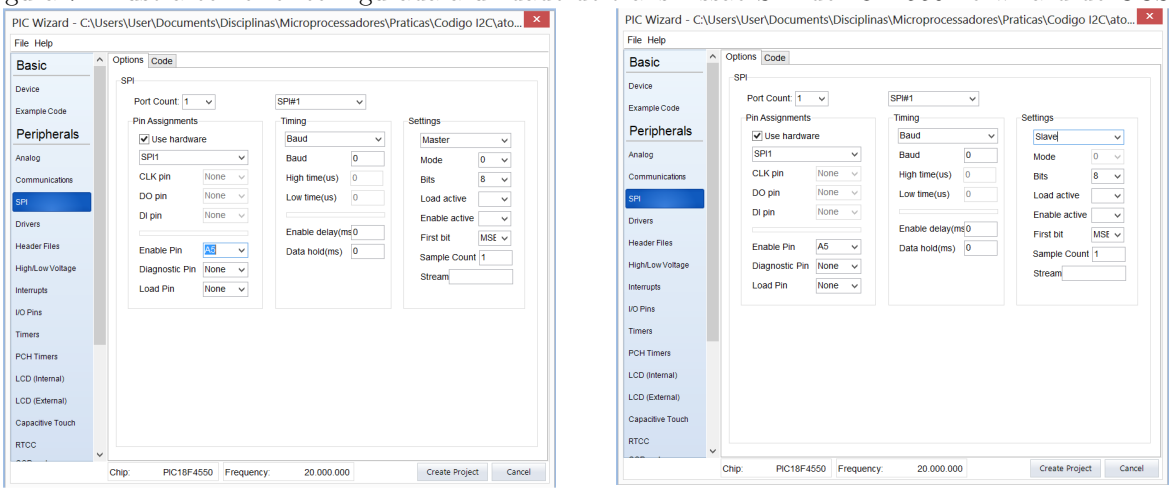
Figura 7.3 - Diagrama esquemático de montagem da transmissão SPI entre um PIC mestre e um escravo conforme descrição da aplicação feita.

O mestre se decida exclusivamente à leitura das chaves e envio de dados SPI. O escravo monitora - por interrupção de recepção de novos dados - a chegada de novos dados e faz a interpretação do dois possíveis comandos definidos pela chave S1. Este exemplo pode ser facilmente

estendido para um maior número de comandos e funcionalidades incluindo uma maior transmissão de dados. A escolha dos pinos deve ser feita segundo indicações do datasheet e os pinos escolhidos são exclusivos para transmissão SPI quando este recurso é habilitado no wizard do CCS.

8.3.2 - O código

Os comandos mostrados no Código 7.1 definem o comportamento do dispositivo mestre. A Figura 7.4 ilustra como foi configurada a unidade de transmissão SPI do 18F4550 no wizard do CCS.



(a) (b)

Figura 7.4 - Configurando o uso do SPI no CCS 5. (a) Configuração do dispositivo mestre e (b) do escravo.

Além das configurações indicadas na Figura 7.4, também habilite no dispositivo escravo a interrupção "SPI or 12C activity" que será acionada toda vez que o escravo receber um novo dado na porta de entrada SPI. Para o mestre, selecione a interrupção ("external interrupt #2") que será executada toda vez que o botão S8 for ativado.

Código 8.1 - Código fonte do dispositivo mestre (compilado para a versão CCS 5)

```
1   #include <mestre.h>
2   #USE SPI (MASTER, SPI1, ENABLE=PIN_A5, BAUD=5000, MODE=0, BITS=8, STREAM=ALAN1,
3   MSB_FIRST)
4   ////////////////////////////////////////////////////////////////////////////
5   #INT_EXT2 //interupcao externa
6   void  EXT2_isr(void)
7   {
8        int dado, comando;
9        disable_interrupts(GLOBAL);
10       dado = input_e();
11       comando = input(PIN_A0);
12       spi_write(comando);
13       spi_write(dado);
14       output_high(PIN_D0);  delay_ms(100);  output_low(PIN_D0);
15       enable_interrupts(GLOBAL);
16  }
17  ////////////////////////////////////////////////////////////////////////////
18  void main()
19  {
20     enable_interrupts(INT_EXT2);
21     enable_interrupts(GLOBAL);
22
23     while(TRUE) //pisca led verde
24     {
25       output_high(PIN_D1);
26       delay_ms(50);
27       output_low(PIN_D1);
```

```
28       delay_ms(500);
29    }
30 }
```

Código 8.2 - Código fonte do dispositivo escravo

```
1  #include <escravo.h>
2  #USE SPI (SLAVE, SPI1, ENABLE=PIN_A5, BITS=8, STREAM=ALAN1, MSB_FIRST, MODE=0)
3  #DEFINE COMANDO_LIGAR_LED_LARANJA 0
4  #DEFINE COMANDO_MOSTRAR_VALOR_LIDO 1
5  int conta_numero_recepcoes, dados_recebidos[2];
6  ////////////////////////////////////////////////////////////////////////////
7  #INT_SSP  //a funcao abaixo tratara a interupcao de recebimento dado SPI
8  void  SSP_isr(void)
9  {
10    if( spi_data_is_in() ) //confirma se tem dado novo no buffer recpecao SPI
11    {
12          dados_recebidos[conta_numero_recepcoes] = spi_read();
13          conta_numero_recepcoes++;
14          if (conta_numero_recepcoes > 1)//completou as 2 leituras (COMANDO+DADO)?
15          {
16             conta_numero_recepcoes = 0;
17             if(dados_recebidos[0] == COMANDO_LIGAR_LED_LARANJA)
18             {
19                output_d(0);
20                output_high(PIN_E1);
21             }
22             if(dados_recebidos[0] == COMANDO_MOSTRAR_VALOR_LIDO)
23             {
24                output_d(dados_recebidos[1]);
25                output_low(PIN_E1);
26             }
27          }
28    }
29 }
30 ////////////////////////////////////////////////////////////////////////////
31 void main()
32 {
33    conta_numero_recepcoes = 0;
34    enable_interrupts(INT_SSP);
35    enable_interrupts(GLOBAL);
36    while(TRUE)
37    {
38      //apenas pisca led
39      output_high(PIN_E2);
40      delay_ms(50);
41      output_low(PIN_E2);
42      delay_ms(500);
43    }
44 }
```