



Projeto de Sistemas RF

Aula 10 - Transmissão de dados utilizando transceptor SPI



Material Didático do Instituto Metrópole Digital - IMD
Versão 5.3 - Todos os Direitos reservados

Apresentação

Na aula prática anterior, você viu como fazer dois PIC se comunicarem utilizando o protocolo SPI. Na aula de hoje, vamos utilizar o conhecimento que você já adquiriu na aula passada para criar um link de rádio entre dois PIC e realizar uma transmissão de dados sem fio, utilizando um dispositivo transceptor que se comunica com o microcontrolador por meio do protocolo SPI. Esta aula também vai ser prática e vai se basear nas atividades realizadas na prática anterior. Preparado? Então, vamos nessa!

Objetivos

- Realizar a comunicação entre dois microcontroladores utilizando um link de rádio e comunicação pelo protocolo SPI.

O *Transceiver*

Creio que você já saiba o que é um transceptor, ou, em inglês, *transceiver*. Mas se não souber, sem problemas. Aqui vai uma explicação: o *transceiver* é um dispositivo eletrônico que pode tanto transmitir dados quanto receber dados. Inclusive o nome dele é formado a partir das palavras "transmissor" e "receptor" (ou "*transmitter*" e "*receiver*", em inglês). Existem *transceivers* para diversas aplicações, como, por exemplo, para telefonia ou redes de computadores, mas estamos interessados nos de rádio frequência.

Se liga!

Os *transceivers* têm a vantagem de ter as capacidades de transmissão e recepção em um único dispositivo, ou seja, um circuito com tal dispositivo já tem a capacidade de receber e enviar dados dependendo da configuração desejada. Mas você pode adquirir também pares de transmissor-receptor para utilizar em aplicações onde se deseja transmitir informação em apenas um sentido.

Os *transceivers*, assim como os microcontroladores, são dispositivos semicondutores que apresentam alguma forma de comunicação com os demais dispositivos do circuito. Eles necessitam ser alimentados e devidamente acoplados aos outros componentes. O melhor lugar para você encontrar toda (ou quase toda) informação sobre o seu *transceiver* é no datasheet do componente. Por isso, procure sempre ler e entender bem o seu dispositivo antes de usá-lo. Esses dispositivos, geralmente, são sensíveis e um mau uso pode danificá-los facilmente.

Na prática desta aula, nós usaremos o *transceiver* nRF24L01+ da Nordic Semiconductor. Esse Transmissor trabalha na faixa de frequência aberta para aplicações industriais, científicas e médicas, e vai de 2.4GHz até 2.4835GHz. Ele precisa ser devidamente alimentado e conectado com algum dispositivo através de um protocolo SPI. Além disso, ele apresenta várias tecnologias para melhorar a

transmissão dos dados, como um sistema de “*auto acknowledgement*”, que confirma automaticamente ao transmissor que um pacote foi recebido e tenta reenviar o pacote caso a transmissão falhe, ou outras que permitem que um *transceiver* configurado como receptor se comunique ao mesmo tempo com até seis *transceivers* transmissores. Mas esses são apenas alguns dos recursos disponíveis e estão além do escopo desta aula. Para saber mais, você pode consultar o datasheet. No momento, estamos apenas preocupados em como configurar o transmissor e o receptor corretamente para realizar uma simples transmissão, o que basicamente se resume em como é realizada a troca de mensagens entre o PIC e o nRF24L01+.

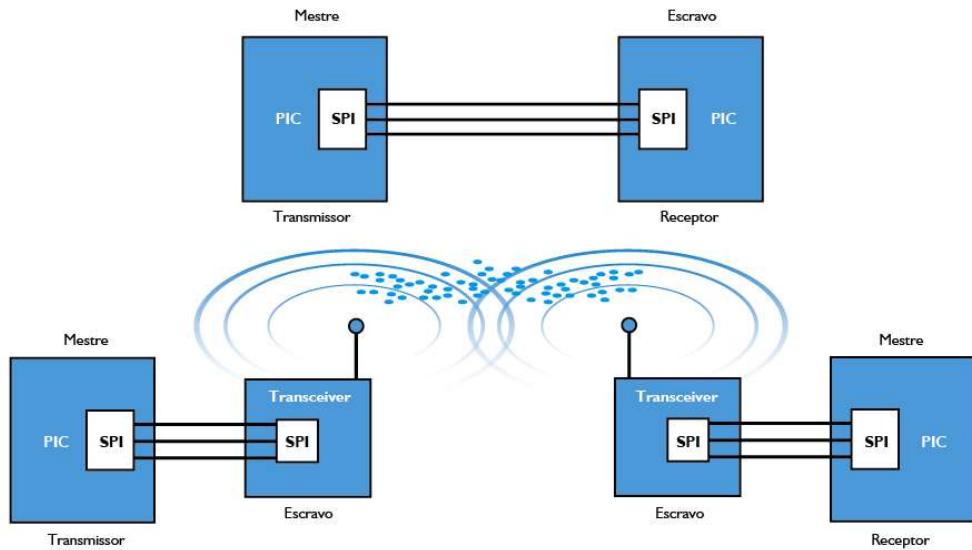
Montando o circuito com o *transceiver*

Você lembra-se de como é a estrutura de comunicação pelo protocolo SPI? Vamos relembrar:

- O SPI é um protocolo baseado em mestre e escravo.
- O dispositivo mestre vai gerar o trem de pulsos (clock) da transmissão.
- É preciso conectar o pino de saída de dados do mestre no pino de entrada de dados do escravo, assim como o pino de saída de dados do escravo na entrada de dados do mestre.
- O protocolo funciona com troca de mensagens entre um mestre e um escravo.
- Se tivermos mais de um escravo na rede de comunicação, usamos um pino extra de seleção de escravo (*slave select*) para decidir com qual escravo o mestre vai se comunicar.

No experimento anterior colocamos um dos PIC como mestre e o outro como escravo, e assim realizamos a troca de mensagens. Porém, os *transceivers* que vamos utilizar agora só podem se comunicar na rede SPI como escravos. Ou seja, ambos os PIC serão mestres de uma comunicação SPI que vai ter o seu *transceiver* como escravo. Perceba que essas comunicações SPI serão distintas, porém ligadas por meio do link de rádio formado pelo par de *transceivers*, como você pode ver na **Figura 1**.

Figura 01 - Transmissão de dados somente pela SPI e utilizando SPI + transceivers



Uma vantagem do esquema atual é que como várias configurações têm de ser iguais em ambos os transceivers para a comunicação acontecer, o trecho do código que realiza a configuração desses dispositivos vai ser bem parecida, facilitando, assim, o seu trabalho na hora de projetar o sistema.

O circuito que vai ser montado será bem semelhante ao da prática passada, mas, de fato, serão dois circuitos completamente separados. A principal diferença são as ligações da SPI entre os PIC, que agora serão feitas com os seus respectivos *transceivers*. Observe bem os circuitos do transmissor e do receptor nas **Figuras 2 e 3**. Com exceção do botão e suas resistências presentes no transmissor, os dois circuitos são essencialmente iguais. Como mais uma vez vamos utilizar o gravador PICKit 3 da Microchip, tenha um especial cuidado para marcar qual é o “pino 1” da barra de pinos do gravador, evitando que ele seja conectado de forma errada e na ordem dos fios na conexão com o transmissor.

Figura 02 - Circuito transmissor

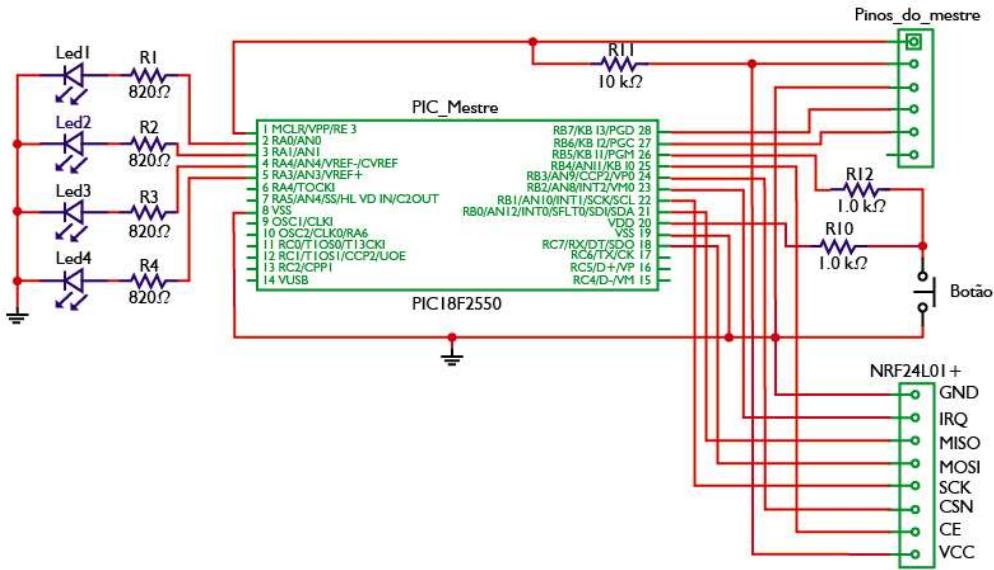
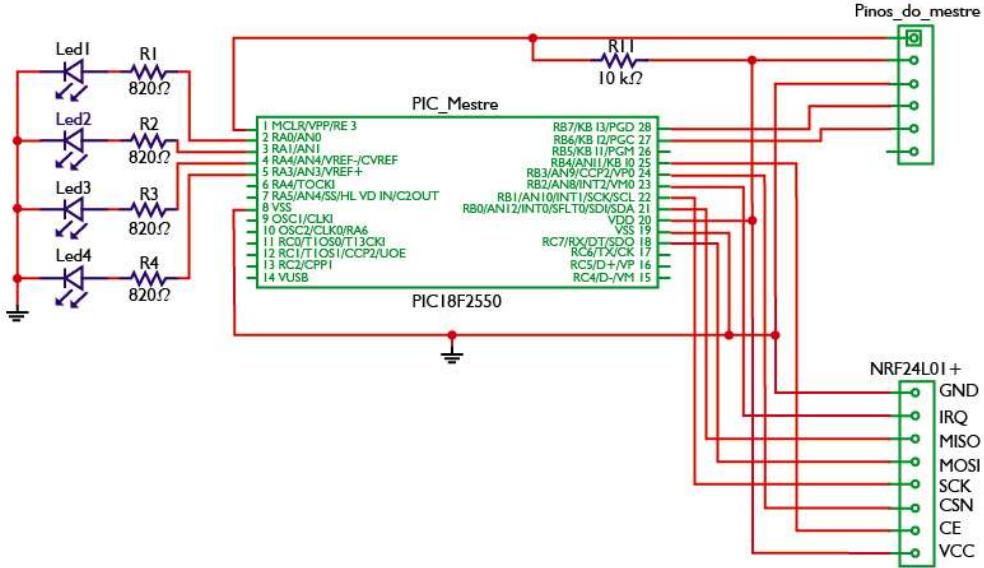


Figura 03 - Circuito receptor



Os circuitos das Figuras 2 e 3 utilizam o microcontrolador PIC18F2550 como exemplo. Porém, algumas configurações detalhadas nesta aula utilizam o microcontrolador PIC18F45K20.

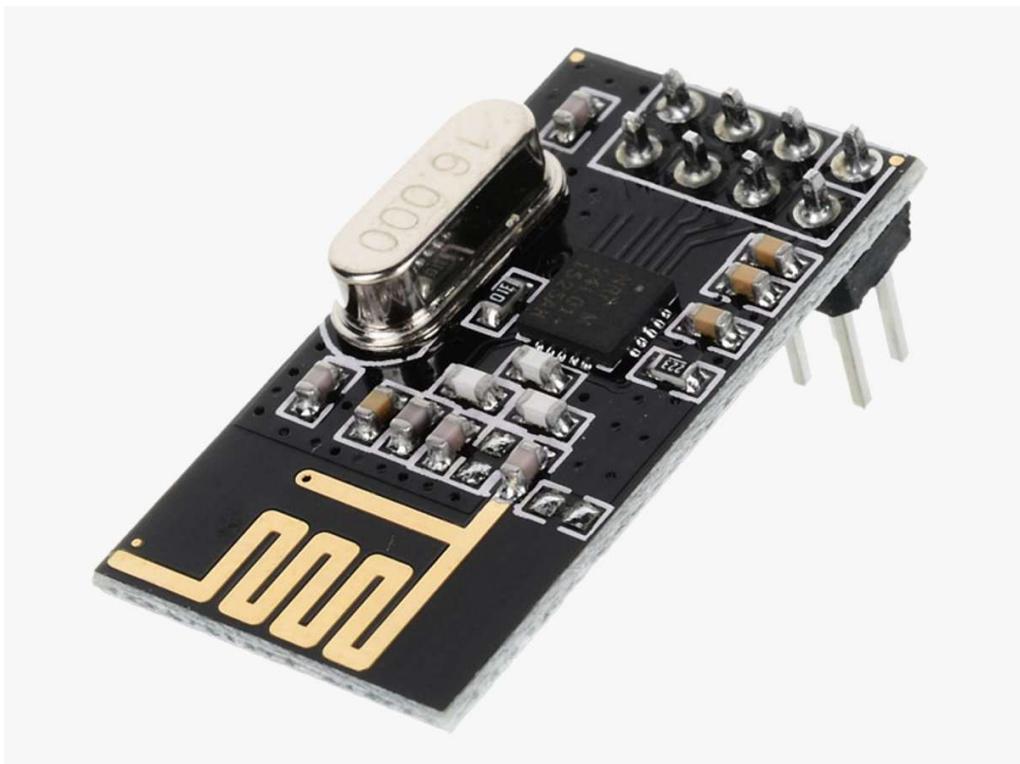
O *transceiver* que vamos utilizar, mostrado na **Figura 4**, contém 8 pinos:

- GND: deve ser ligado no “terra”, ou seja, na mesma tensão que é aplicada ao Vss do microcontrolador.
- IRQ: Conhecido como “*Interrupt Request*”, esse pino é utilizado para interromper o PIC quando o transceiver precisar se comunicar com

ele. Essa interrupção acontece, geralmente, quando o receptor acabou de receber um novo dado e está pronto para passá-lo para o PIC, para avisar que um dado foi enviado com sucesso ou que ocorreu algum erro de transmissão. Esse pino deve ser conectado a uma entrada do PIC que aceite interrupções externas e tem como estado “ativo” o nível lógico 0. No PIC18F45K20, temos as entradas 0, 1 e 2 da porta B. Vamos utilizar o pino 2.

- MISO: “*Master Input, Slave Output*”, ou seja, é o meio de comunicação entre o escravo e o mestre. Deve ser conectado no pino SDI (*Serial Data Input*) do PIC.
- MOSI: De forma análoga, o “*Master Output, Slave Input*” é por onde o *transceiver* vai receber os dados do PIC pela SPI. Deve ser conectado no pino SDO do PIC.
- SCK: o pino “*Serial Clock*” vai receber o trem de pulsos gerado pelo mestre para sincronizar a comunicação entre eles. Deve ser conectado no pino de mesmo nome no PIC.
- CSN: Esse pino funciona como o “seletor de escravo” da SPI, apesar de cada PIC ter apenas um escravo conectado. Ele é utilizado para iniciar uma troca de mensagens via SPI entre o microcontrolador e o *transceiver*. A comunicação é ativada ao aplicar um nível lógico baixo, e deve ser mantido baixo durante toda a comunicação com o *transceiver*. Ele deve ser ligado em uma saída digital qualquer do PIC. No experimento atual, conecte-o no pino 3 da porta B.
- CE: O pino CE é conhecido como “*Chip Enable*” e é por onde o PIC vai informar ao *transceiver* se ele deve habilitar o sistema de rádio frequência, ou seja, entrar efetivamente em funcionamento. Se ele estiver configurado como transmissor e existir algum dado no buffer para ser enviado, o dispositivo vai enviá-lo. Já se a configuração for para receber dados, o *transceiver* vai ficar ouvindo o meio, esperando por alguma mensagem e avisando ao PIC através do pino IRQ, caso alguma chegue. Também deve ser conectada a alguma saída digital do PIC. Conecte-o no pino 4 da porta B.
- VCC: Tensão de alimentação do dispositivo. Aceita tensões até 7V. Pode ligá-lo na mesma tensão que você está aplicando ao Vdd do microcontrolador.

Figura 04 - Transceiver nRF24L01+



Se liga!

Os pinos CSN e CE podem ser ligados em qualquer outra saída digital e o pino IRQ pode ser conectado em qualquer outra entrada que gere interrupções, ou mesmo em qualquer entrada digital se você implementar no seu PIC alguma função que verifique com frequência qual o valor dessa porta e tome as medidas necessárias caso seja detectado um sinal baixo. Não houve nenhum critério para escolher essas entradas/saídas. Sinta-se a vontade para mudá-las se quiser, desde que você modifique o código de acordo para usar os novos pinos.

Configurando o *transceiver*

Para configurar e utilizar o dispositivo, é preciso fazê-lo “dialogar com o PIC”. Pode parecer esquisito, mas é bem isso que acontece entre os dois: uma conversa. Através do protocolo SPI, você vai dizer ao *transceiver* se ele vai funcionar como transmissor ou como receptor, qual a frequência que ele vai trabalhar, qual a

potência da transmissão, qual a taxa de transmissão, qual o tamanho do pacote a ser transmitido, qual o dado a ser transmitido etc. O que você precisa saber é como realizar essa conversa, ou seja: o SPI provê uma forma de trocar mensagens, mas as mensagens têm que fazer sentido para o transmissor. Você vai precisar fazer com que o PIC, ao executar o seu programa, mande mensagens que o dispositivo consiga entender e, assim, funcionar da forma desejada. Você pode estar se perguntando: "Mas onde eu posso aprender quais são os comandos que eu devo enviar?". A resposta é simples: no datasheet do componente. Mas não se preocupe: nesta aula, vamos explicar como são essas mensagens que o *transceiver* espera receber do PIC e quais enviar para que ele funcione no modo mais básico.

A experiência que vamos realizar é semelhante à realizada na aula passada: ao pressionar um botão no circuito transmissor, vamos atualizar o valor de uma variável que é usada para acender alguns LED e, ao mesmo tempo, vamos transmitir esse valor atualizado para o dispositivo receptor, que, por sua vez, vai receber esses dados e exibi-los em seus LED. A diferença agora é que a transmissão vai ser por meio de um link de rádio.

Para facilitar, utilizamos os códigos da aula anterior como esqueletos para os códigos da aula de hoje. Porém, dessa vez você vai apenas precisar adicionar os trechos de código para configuração e uso do transmissor. Os métodos de atualização dos LED ao pressionar o botão já estarão implementados. Vamos primeiro falar do código do programa transmissor.

Código do transmissor

Você se lembra da função "UserInit()"? Ela era a responsável por configurar todos os módulos do PIC, antes dele entrar no laço principal de execução. Pois bem, o *transceiver* precisa ser configurado **antes** de o PIC entrar no seu laço principal, pois queremos utilizar o transmissor ou receptor durante a execução desse laço. Portanto, toda a configuração do *transceiver* vai ser feita dentro do UserInit(). Lembre-se que essa função já vai ter outras linhas de código para configuração de outras coisas do microcontrolador. Os locais onde você deve adicionar o seu código estarão devidamente sinalizadas em comentários no código.

Para fazer o *transceiver* funcionar como transmissor, você deve:

1. Definir a direção dos dados e o valor inicial dos pinos conectados ao CSN,

▼ CE, IRQ do *transceiver*

Esses pinos do *transceiver* são conectados em portas digitais normais do PIC, e, portanto, elas devem ser configuradas de acordo com tais normas. Altere o bit do registrador TRIS da porta B para que os pinos conectados ao CSN e ao CE funcionem como saídas digitais, e o pino conectado ao IRQ funcione como entrada digital. Além disso, sete o valor inicial da saída CE para 0 e da saída CSN para 1. Experimente utilizar as definições encontradas no código (na seção onde as linhas começam com "#define").

▼ 2. Configurar e habilitar a comunicação pelo protocolo SPI

O *transceiver* vai conversar com o PIC através da SPI, mas para isso acontecer a SPI deve estar configurada e funcionando. Para configurar o SPI utilize o conhecimento que você adquiriu na aula passada. Lembre-se que o *transceiver* vai ser o seu escravo, portanto configure o PIC para ser o mestre do barramento. Experimente utilizar a função *OpenSPI* da biblioteca do C18 para facilitar o seu trabalho.

3. Esperar um tempo necessário para que o transmissor esteja pronto para

▼ se comunicar com o PIC

Assim como o microcontrolador, o *transceiver* tem uma rotina de inicialização para executar e demora certo tempo até que ele esteja pronto para conversar com o PIC. O datasheet do nRF24L01+ diz que esse tempo é de 150ms, logo você deve esperar esse tempo antes de enviar qualquer dado para ele através do SPI. De fato, quando o PIC chegar nessa linha de código durante a sua execução já vai ter se passado algum tempo, mas você não tem como ter certeza de quanto tempo se passou. Por segurança, espere pelos 150ms mesmo.

▼ 4. Enviar os comandos de configuração

É aqui que toda a mágica acontece. Você vai ter que enviar dados para configurar o *transceiver* como transmissor, antes de dizer para ele entrar em funcionamento. Existem muitas coisas a serem ditas aqui, portanto, vamos explicá-la em detalhes na próxima seção.



Atenção

Para fazer o PIC esperar por algum tempo, você pode utilizar as funções da biblioteca `<delay.h>` do C18. Ela fornece várias funções para fazer o PIC esperar por um número de ciclos até continuar para a linha seguinte. O que é importante aqui é saber quanto tempo dura um ciclo de processamento do PIC. Nas configurações do código, o PIC está usando o oscilador interno e logo deve executar os ciclos em uma frequência de 250KHz. Para saber o tempo de um ciclo, use a equação

$$\text{Time} = \frac{1}{f}$$

onde f é a frequência em que os ciclos estão sendo executados. O tempo total T_E gasto para executar N ciclos pode ser obtido multiplicando N pelo tempo T de execução de um ciclo, ou seja:

$$T_E = N \times T$$

Porém, o que queremos mesmo é saber qual o valor de N que vai gerar o tempo T_E desejado, portanto podemos usar as duas equações acima para gerar uma única equação do valor de N . Por exemplo, o número de ciclos para fazer o PIC esperar por 150ms é:

$$N = \frac{T_E}{T} = \frac{150 \times 10^{-3}}{\frac{1}{250 \times 10^6}} = 37500$$

Para implementar essa espera, você pode, por exemplo, utilizar a função `Delay10KTCYx(N)` que faz o PIC esperar por ciclos antes de continuar com a execução do código. Verifique na leitura complementar a documentação das funções da biblioteca do C18 para ver as outras funções de espera disponíveis.

Trocando mensagens com o *transceiver*

A conversa com o *transceiver* acontece da seguinte forma: primeiro você tem que informar ao *transceiver* que você quer se comunicar com ele. Isso é feito colocando a saída conectada no pino CSN do *transceiver* para um nível lógico 0. Dessa forma o *transceiver* vai ficar pronto para trocar mensagens pela SPI com o PIC. Essa saída deve ficar nível baixo durante toda a conversa. Se o *transceiver* perceber que o CSN voltou para um nível alto, mesmo enquanto ele estiver recebendo dados pela SPI, ele vai considerar que a conversa com o PIC terminou e não vai receber mais nada dele até que ele perceba um valor baixo novamente no pino CSN.

Após aplicar 0 no CSN, você vai ter que enviar um comando para o *transceiver*. O *transceiver* espera receber alguns comandos específicos e predeterminados. Esses comandos podem indicar, por exemplo, que você quer escrever algum dado em um dos registradores internos do *transceiver*, ou que você quer dizer ao *transceiver* qual o pacote que ele deve enviar pela rede sem fio, ou que você quer ler o *status* atual do *transceiver* etc. Os comandos são de fato apenas sequências de bits e o formato dessas sequências e o que cada uma delas significa está descrito em detalhes no datasheet do dispositivo. Porém, você não precisa decorar todas as sequências: elas estão todas definidas no arquivo nRF24L01.h que você pode encontrar na pastas do Mestre e do Escravo nessa aula. Logo, você pode acessar esses valores usando apenas o nome do comando, o que é muito mais fácil e menos suscetível a erros.

Alguns dos comandos que você envia para o *transceiver* exigem que algum dado seja enviado na sequência, ou que você leia um dado do *transceiver*. Outros não exigem nenhum dado adicional. Se você precisar enviar algum dado na sequência ou ler algum dado do PIC após o comando, lembre-se de manter o CSN em nível baixo até terminar de enviar ou ler por completo os dados. Por fim, retorne o CSN para o nível alto para informar ao *transceiver* que a conversa terminou.

Dito isso, vamos a um exemplo mais prático. Suponha que você quer informar ao transmissor que os próximos 3 bytes que você deseja enviar têm os valores 10, 15 e 27. O comando usado para informar isso ao transceiver é W_TX_PAYLOAD (que é, de certa forma, intuitivo: W de write, TX significa *transmitter* e PAYLOAD é a carga

de transmissão, ou seja, esse comando pode ser lido como “escrever a carga a ser transmitida”). Assim sendo, um trecho de código que executaria corretamente o comando seria o seguinte:

```
1 CSN = 0.  
2 WriteSPI(W_TX_PAYLOAD).  
3 WriteSPI(10).  
4 WriteSPI(15).  
5 WriteSPI(27).  
6 CSN = 1.
```

Observe no código acima aquilo que estávamos discutindo nos parágrafos anteriores: o uso do CSN para indicar a comunicação, o envio do comando W_TX_PAYLOAD, o envio da “carga” a ser transmitida (10, 15 e 27) e, finalizando a conversa, colocando novamente o CSN para o nível alto. Perceba que estamos usando o comando *WriteSPI* da biblioteca spi.h do C18, que recebe um byte (ou seja, um valor de 0 a 255) como argumento e o transfere pela SPI. O uso dessa função facilita bastante a comunicação com *transceiver*. Observe que fazemos a comunicação byte a byte, como era de se esperar de uma comunicação SPI.

Se quiséssemos ler dois dados do *transceiver*, poderíamos fazer da seguinte forma:

```
1 CSN = 0.  
2 WriteSPI(R_RX_PAYLOAD).  
3 dado1 = ReadSPI().  
4 dado2 = ReadSPI().  
5 CSN = 1.
```

Perceba que aqui o comando é R_RX_PAYLOAD (R de read e RX indicando receptor, ou seja, “ler a carga recebida”), e estamos usando a função ReadSPI da biblioteca spi.h. O primeiro dado lido está sendo colocado na variável dado1 e o segundo dado na variável dado2. Se o comando não exigir uma escrita ou leitura adicional, como o comando FLUSH_TX, nós podemos enviar somente ele, como mostrado no código abaixo.

```
1 CSN = 0.  
2 WriteSPI(FLUSH_TX).  
3 CSN = 1.
```

Na prática desta aula, vamos precisar manter os seguintes tipos de comunicação com o transceiver: escrever e ler um dado em um registrador do transceiver; ler um dado recebido e escrever um dado a ser transmitido; limpar o buffer de transmissão e de recepção.

Configurando registradores do transceiver

- Escrever e ler um dado em um registrador

O transceiver contém alguns registradores que definem o seu funcionamento. A configuração inicial dele que é feita na função UserInit significa, basicamente, setar os valores dos bits desses registradores para os valores corretos que fazem com que ele funcione da forma que queremos.

O comando de escrita é W_REGISTER e o de leitura é R_REGISTER, mas eles não funcionam se enviados sozinhos: você precisa informar qual é o endereço do registrador que você quer ler ou escrever. A diferença é que não informamos o endereço em um byte separado, e sim dentro do próprio byte de comando!

Parece complicado, mas, na verdade, não é. Os comandos W_REGISTER e R_REGISTER são formados por 1 Byte, porém somente os 3 bits mais significativos que definem o comando. Os 5 menos significativos são variáveis e definem o endereço do registrador. Por exemplo, W_REGISTER tem o valor binário "001AAAAA" onde somente "001" define o comando de escrita, e "AAAAA" é o endereço do registrador. Se você quisesse escrever o valor 15 no registrador RF_CH cujo endereço é "00101" em binário, o comando de "escrever no registrador 00101" ficaria:

- | | |
|---|-----------------------|
| 1 | CSN = 0. |
| 2 | WriteSPI(0b00100101). |
| 3 | WriteSPI(15). |
| 4 | CSN = 1. |

Onde o "0b" indica que o número é um valor binário.

Mas agora imagine como seria difícil ter que decorar todos os endereços de todos os registradores do *transceiver* que você quer utilizar. Ainda bem que temos uma alternativa pra isso: o arquivo nRF24L01.h define também os endereços de todos os registradores do *transceiver*. O que você precisa fazer é utilizar o nome do comando de escrita ou leitura fazendo um OU binário com o endereço que você quer ler ou escrever. Ou seja, para escrever 15 no registrador RF_CH podemos alterar o código anterior para:

```
1 CSN = 0.  
2 WriteSPI(W_REGISTER | RF_CH).  
3 WriteSPI(15).  
4 CSN = 1.
```

Esse código é bem mais fácil de entender e programar. Uma leitura no mesmo registrador ficaria mais ou menos assim:

```
1 CSN = 0.  
2 WriteSPI(R_REGISTER | RF_CH).  
3 dado = ReadSPI().  
4 CSN = 1.
```

E o valor do registrador ficaria armazenado na variável *dado*. Observe o uso do comando “|” (barra vertical) que em C significa a operação “OU-binário”.

Agora, vamos a uma última observação sobre as operações de escrita nos registradores. Muitas vezes você vai querer apenas setar um bit específico de um registrador ou alguns bits específicos. Assim como acontece no PIC, cada bit de um registrador do *transceiver* tem funções específicas, e o arquivo nRF24L01.h armazena as posições que os bits ocupam nos seus registradores através dos nomes nos bits.

Observe que isso é diferente do acesso aos bits de um registrador do PIC. No caso do *transceiver*, você sabe apenas qual a posição que um determinado bit ocupa em um registrador, e não qual é o registrador que contém esse bit. É como se nós soubéssemos que o bit RA1 pertence ao registrador PORTA, mas o compilador não soubesse, portanto não poderíamos alterar o valor do bit usando PORTAbits.RBA = 1. Teríamos que utilizar outra estratégia.

Os nomes dos bits do *transceiver* são vistos pelo compilador como simples números. Por exemplo, o bit PRIM_RX é o bit número 0 do registrador CONFIG (lembre-se que os bits em um Byte são ordenados de 0 a 7). Para setar esse bit teríamos que dizer ao *transceiver* “coloque o bit número 0 do registrador CONFIG para o valor 1”. Isso de ser feito pelas seguintes linhas de código:

```
1 CSN = 0.  
2 WriteSPI(W_REGISTER | CONFIG).  
3 WriteSPI( (1 << PRIM_RX) ).  
4 CSN = 1.
```

Note o comando “<<” no código acima. O que ele faz é deslocar o valor 1 um número PRIM_RX de bits para a esquerda. Por exemplo, o código “1 << 3” iria gerar o valor binário “0000 1000”, ou seja, somente o bit de número 3 está setado.

Agora imagine que você quer setar também o bit PWR_UP do registrador CONFIG. Você pode pensar que é só utilizar as linhas de código acima e adicionar as linhas abaixo depois:

```
1 CSN = 0.  
2 WriteSPI(W_REGISTER | CONFIG).  
3 WriteSPI( (1 << PWR_UP) ).  
4 CSN = 1.
```

Infelizmente, **isso não vai funcionar!**

O motivo é que a escrita de um registrador do *transceiver* reescreve **todo** o registrador. Ou seja, apesar de você ter setado o bit PRIM_RX primeiro, a segunda escrita vai setar somente o bit PWR_UP, limpando o bit PRIM_RX que você tinha setado anteriormente.

Para resolver isso, você tem que montar todo o registrador da forma que você quer no código e enviar tudo isso de uma vez para o *transceiver*. Isso pode ser feito da seguinte forma:

```

1 CSN = 0.
2 WriteSPI(W_REGISTER | CONFIG).
3 WriteSPI( (1 << PRIM_RX) | (1 << PWR_UP) ).  

4 CSN = 1.

```

Agora sim estamos setando tanto o bit PRIM_RX quanto o bit PWR_UP ao mesmo tempo. Somente esses bits do registrador CONFIG estarão com valor 1, os demais ficarão com valor zero. O uso dessa notação pode gerar alguns avisos na compilação (*warnings*) de que a expressão não tem nenhum efeito, mas nesse caso isso não é um problema.

Comandos para o transceiver

- Escrever um dado para ser transmitido e receber um dado lido.

Para esse tipo de diálogo usamos os comandos W_TX_PAYLOAD e R_RX_PAYLOAD. Esses comandos são utilizados para colocar dados no buffer de transmissão ou para ler os dados do buffer de recepção. Esses buffers têm 32 Bytes de tamanho. Ou seja, 32 Bytes é o tamanho máximo do pacote que você vai poder transmitir de uma só vez. A leitura ou escrita dos dados vai em sequência ao comando, como mostramos no exemplo que usa o W_TX_PAYLOAD. Porém, para dados maiores você pode querer armazenar os dados em um vetor e utilizar um laço para percorrer o vetor de dados ao invés de colocar uma linha para cada byte. Assim, seu código deve ficar mais ou menos assim:

```

1 CSN = 0.
2 WriteSPI(W_TX_PAYLOAD).
3 for(contador=0; contador < tamanho_do_dado; contador = contador + 1){
4   WriteSPI(dado[contador]).  

5 }
6 CSN = 1.

```

Onde *tamanho_do_dado* é uma variável que armazena o tamanho do dado que você quer enviar, *dado* é o vetor onde você colocou o dado que você quer transmitir, e *contador* é a variável usada para percorrer esse vetor de dados. Uma leitura usando a mesma ideia poderia ser feita da seguinte forma:

```
1 CSN = 0.  
2 WriteSPI(R_RX_PAYLOAD).  
3 for(contador=0; contador < tamanho_do_dado; contador = contador + 1){  
4   dado[contador] = ReadSPI().  
5 }  
6 CSN = 1.
```

- Limpar o buffer de transmissão e o de recepção.

Às vezes, vamos precisar limpar os buffers de transmissão ou de recepção. Geralmente fazemos isso para garantir que não existe nenhum lixo nesses buffers, evitando confundir esse lixo com dados que realmente importam. Para limpar o buffer de transmissão, use o comando FLUSH_TX. Para limpar o buffer de recepção, use o comando FLUSH_RX.

Pronto. Agora já sabemos como comunicar o PIC com o transceiver. Agora vamos voltar e ver o que precisamos informá-lo durante a configuração.

Nós paramos na configuração do *transceiver*. Vamos continuar vendo o que você tem que configurar nele. Lembre-se que essas configurações são feitas enviando os comandos e dados para o transmissor pela SPI. Consulte a seção 9.1 do datasheet do *transceiver* para verificar qual valor tem que ser colocado em cada bit para atingir o efeito desejado do *transceiver*.

▼ 4.1. Colocar o transmissor no modo "Power Down"

Para configurar o transceiver, ele tem que estar no modo “desligado”. Porém o seu programa não tem como saber se o transmissor foi ligado agora junto com o PIC ou se ele já estava ligado e somente o PIC que foi resetado. Para garantir que o transmissor vai estar no modo “desligado” antes de ser configurado, independente da forma que o PIC foi iniciado, vamos primeiro mandar o transmissor desligar.

Para fazer isso, basta resetar o bit PWR_UP do registrador CONFIG.

▼ 4.2. Configurar a potência e taxa de transmissão

Você pode configurar a potência do transmissor alterando os bits RF_PWR do registrador RF_SETUP e a taxa de transmissão através dos bits RF_DR_LOW e RF_DR_HIGH do mesmo registrador. Note que RF_PWR representa na verdade dois bits, logo você vai ter que usar a notação "(0b11 << RF_PWR)" se quiser setar ambos os bits do RF_PWR, ou "(0b01 << RF_PWR)" se quiser setar somente o bit menos significativo, e assim por diante. Lembre-se tanto o transmissor quanto o receptor tem que estar configurados com a mesma taxa de transmissão. Consulte o datasheet para ver os valores de potência e taxa disponíveis para o transceiver.

▼ 4.3. Configurar o canal de comunicação

O canal de comunicação é a frequência que o *transceiver* vai utilizar para transmitir ou receber dados. Você define o canal de comunicação escrevendo no registrador RF_CH. O *transceiver* aceita valores para o RF_CH de 0 a 127.

A frequência de trabalho do *transceiver* é definida através da seguinte fórmula:

$$\$ F_0 = 2400 + RF_CH[\text{MHz}] \$$$$

Lembre-se de definir o mesmo canal de comunicação no receptor ou eles não vão ser capazes de se comunicar.

▼ 4.4. Desabilitar o envio do ACK automático

O nRF24L01+ tem um sistema automático que faz o receptor avisar ao transmissor se o pacote transmitido foi recebido corretamente. Porém para utilizar corretamente esse recurso é preciso ajustar também outras características do transmissor, que foge do escopo desta aula. Logo, o que vamos fazer aqui é permitir que o transmissor envie um pacote que ele não quer saber se chegou ou não no receptor. Isso torna o sistema mais sujeito a falhas, mas ao mesmo tempo torna as coisas mais simples.

Para fazer isso, você precisa setar o bit EN_DYN_ACK no registrador FEATURE.

4.5. Configurar o transceiver como transmissor e colocá-lo no modo "Power

▼ Up"

Agora já configuramos quase tudo que queríamos configurar. Só está faltando agora dizer ao transceiver que ele irá funcionar como um transmissor e colocá-lo em funcionamento. Faça isso ao mesmo tempo, resetando o bit PRIM_RX e setando o bit PWR_UP do registrador CONFIG.

▼ **4.6. Esperar por 200us para o transmissor ser habilitado.**

O *transceiver* demora, aproximadamente, 200us para entrar em funcionamento depois de um "Power Up". Use uma das funções de espera para fazer o PIC esperar por esse tempo.

▼ **4.7. Esvaziar o buffer de transmissão**

Por fim, esvazie o buffer de transmissão para garantir que o transmissor não vai ter lixo nos seus registradores. Para isso, utilize o comando FLUSH_TX.

Isso deve finalizar a configuração do *transceiver*. Agora só precisamos fazer mais uma coisa.

▼ **5. Habilite as interrupções na porta RB1 (INT1)**

Quando o *transceiver* precisar informar qualquer coisa ao PIC, ele vai fazer colocando o sinal no pino IRQ para zero. O PIC deve ser configurado para capturar essas interrupções. Como conectamos o pino IRQ na interrupção INT1, ela deve ser configurada corretamente. Experimente utilizar a função *OpenRB1/INT* da biblioteca portb.h. Lembre de também configurar a interrupção para a transição de alto para baixo (*FALLING_EDGE_INT*) e habilitar os *pull ups* na porta B (*PORTB_PULLUPS_ON*). Verifique no manual das funções das bibliotecas do C18 como fazer isso utilizando o *OpenRB1/INT*. Não se esqueça de habilitar o bit GIEH do registrador INTCON do microcontrolador depois de chamar a função *OpenRB1/INT*.

Pronto. Isso finaliza a configuração do transmissor.

Enviando um dado pelo *transceiver*

Na prática desta aula, um dado vai ser enviado sempre que pressionarmos o botão no circuito. A rotina que atualiza a variável LED_Display e que acende os LED já está implementada, portanto você não precisa se preocupar com isso. O que você vai precisar fazer é adicionar o código para realizar a transmissão sempre que a variável LED_Display for atualizada.

Para realizar a transmissão, você usaria normalmente o comando W_TX_PAYLOAD, mas como estamos querendo transmitir um pacote que não vai ter confirmação de recebimento, vamos utilizar o comando W_TX_PAYLOAD_NO_ACK. Para nossa sorte, o funcionamento deste comando é o mesmo do outro. Você vai informar o comando para o *transceiver* e depois vai enviar os bytes que deseja transmitir. No nosso exemplo, queremos transmitir somente 1 Byte, que é a variável LED_Display.

Depois de transmitir o dado para o *transceiver*, é chegada, finalmente, a hora de dizer para ele mandar esse dado por radio frequência. Para isso, você deve colocar a saída do PIC conectada ao pino CE para 1. Assim o *transceiver* vai transmitir tudo que estiver no seu buffer de transmissão. Se existirem muitos bytes a serem transmitidos, isso pode demorar certo tempo. Mas o PIC não precisa esperar os dados serem transmitidos. Basta deixar o CE em nível 1 por no mínimo 20us que o *transceiver* vai fazer o resto. Use uma das funções de *delay* esperar o tempo necessário antes de colocar o pino CE para o nível zero. Como estamos trabalhando com uma frequência de oscilação relativamente baixa, pode ser difícil fazê-lo esperar por exatamente 20us. Se achar mais fácil, você pode usar o tempo de 40us.

O *transceiver* vai enviar o pacote e assim que terminar de transmitir ele deve informar o PIC a sua conclusão. Isso é feito através da rotina de interrupção. Nessa rotina, você deve ler o registrador STATUS do *transceiver* e ver por que o PIC foi interrompido. Cada bit desse registrador informa alguma coisa ao PIC. No modo de transmissão, o *transceiver* vai interromper para: 1) avisar se a transmissão foi concluída, através do bit TX_DS; ou 2) avisar que o máximo de retransmissões foi alcançado sem obter a confirmação que o dado foi recebido, através do bit MAX_RT. Porém, essa segunda opção nunca deve acontecer, já que desabilitamos a confirmação do recebimento.

Após perceber o motivo da interrupção do *transceiver*, é extremamente importante que você avise ao *transceiver* que já tratou a interrupção e que ele pode voltar a funcionar normalmente. Isso é feito escrevendo 1 no bit do registrador

STATUS que gerou a interrupção.

Esse tratamento para os bits MAX_RT e TX_DS já está feito no transmissor, portanto você não vai precisar mexer na função de tratamento da interrupção no código do transmissor. Apesar disso, é importante que você veja o código e entenda como ele funciona, pois você vai precisar implementar algo semelhante no código do receptor.

Pronto. Experimente agora compilar seu código e ver se ele está compilando corretamente. Vamos agora passar para o código que você deve desenvolver para o circuito receptor.

Código do receptor

No PIC receptor, as coisas vão funcionar de forma muito semelhante ao que foi feito no PIC transmissor. Como ambos transceivers são escravos, o modo de funcionamento na SPI vai ser o mesmo. Assim como os valores iniciais das saídas conectadas aos pinos CE e CSN.

O código de configuração do *transceiver* vai ser bem parecido, por uma questão bem lógica: eles têm que estar funcionando na mesma frequência e taxa de transmissão. Apesar disso, existirão, sim, algumas pequenas diferenças na configuração. Comece seguindo os passos 1 ao 4.3 da configuração do transmissor, e então siga os passos a seguir:

▼ 4.4. Informe o número de bytes dos pacotes a serem recebidos.

Como quem define se vai querer receber ou não a confirmação de recebimento é o transmissor, não há nada que você possa fazer aqui em relação a isso. Porém uma coisa que deve ser feita somente no receptor é informar quantos bytes o *transceiver* deve esperar receber pela rede sem fio antes de avisar ao PIC que existe um novo dado. Ou seja, você tem que informar ao *transceiver* o número de bytes do seu pacote. Para isso, você deve escrever esse número no registrador RX_PW_P0 do *transceiver*. Lembrando que no nosso experimento o pacote que estamos transmitindo tem apenas 1 Byte.

4.5. Configurar o *transceiver* como receptor e colocá-lo no modo "Power Up"

▼ Up"

A configuração aqui é bem semelhante ao realizado no transmissor, com a diferença que você vai setar o bit PRIM_RX para 1 ao invés de resetá-lo. Isso vai informar ao *transceiver* que ele está funcionando como receptor, ao invés de transmissor. O bit PWR_UP continua sendo setado em 1.

▼ 4.6. Esperar por 200us para o transmissor ser habilitado

O *transceiver* demora o mesmo tempo para ser inicializado no modo de recepção, portanto faça o PIC esperar pela mesma quantidade de tempo.

▼ 4.7. Esvaziar o buffer de recepção

Semelhante ao realizado no transmissor, mas agora para o receptor. Utilize o comando FLUSH_RX.

▼ 4.8. Habilite a recepção de dados

Diferente do transmissor, onde queremos que ele acesse o meio sem fio somente quando estiver transmitindo algo, o receptor precisa ficar acessando o meio o tempo todo, pois assim vai poder perceber se algum dado está sendo transmitido e assim receber esse dado. Para habilitar a recepção, mude a saída conectada ao pino CE para 1. Não precisa colocar essa saída de volta para 0 depois: queremos que o receptor sempre ativado.

▼ 5. Habilite as interrupções na porta RB1 (INT1)

As interrupções no receptor devem ser configuradas da mesma forma que o transmissor.

Isso é o suficiente para configurar o receptor. Vamos, finalmente, ao que você deve fazer para pegar no PIC o dado que o *transceiver* recebe pela rede sem fio.

Recebendo o dado do *transceiver*

Estando tudo configurado de acordo com os passos anteriores, o que vai acontecer é o seguinte: assim que o *transceiver* receber um byte pela rede sem fio, ele vai armazená-lo em uma memória própria e vai informar ao PIC que “algo aconteceu” através da interrupção INT1. O PIC, ao entrar na sua rotina de interrupção, vai perguntar ao *transceiver* o que aconteceu, lendo o registrador STATUS. Daí ele deve verificar no valor lido o que aconteceu e tratar corretamente o ocorrido.

Nós já vimos duas possíveis situações em que o PIC seria interrompido: quando o bit MAX_RT ou o bit TX_DS fosse colocado em nível alto. Nenhuma dessas duas interrupções deve acontecer no receptor, já que ele somente recebe dados. O bit que devemos checar é se o bit RX_DR é 1. Se for, quer dizer que o receptor acabou de receber um novo dado, logo devemos copiá-lo para o PIC e colocá-lo na variável LED_Display, para que os LED no circuito sejam acesos de acordo com o que o transmissor enviar. É esse o procedimento que você vai ter que implementar na rotina de interrupção.

A rotina de recepção seria bem mais simples se não fosse o seguinte fato: dependendo da velocidade da SPI, da taxa de transmissão e do tamanho do pacote, pode acontecer que um pacote seja recebido pelo *transceiver* enquanto o pacote anterior ainda estiver sendo copiado para o PIC. O nRF24L01+ tem três buffers de recepção e transmissão, o que significa que eles podem armazenar até três mensagens, antes de começar a perder mensagens, caso o PIC demore muito para responder a interrupção. Sendo assim, é importante que você não receba apenas um pacote, mas que verifique se existem outros pacotes para serem recebidos, antes de voltar para a execução normal do seu código. Para fazer isso, siga os seguintes passos.

1. Crie um laço de recepção

O objetivo aqui é receber todos os pacotes do *transceiver* até que ele não tenha mais nenhum novo pacote. Isso garante que sempre vamos ter o pacote mais recente em mãos. Esse laço deve ser infinito (usando um *while(1)*, por exemplo). O laço será quebrado com uma instrução *break* assim que tivermos certeza que não tem mais nenhum dado nos buffers. Todos os passos a seguir devem ser executados dentro do laço.

1.1. Verifique se existem dados nos buffers de recepção.

Isso pode ser feito lendo o registrador FIFO_STATUS do *transceiver*. Ao receber o estado desse registrador, verifique se o bit RX_EMPTY é 1. Se ele for, significa que não existem novos pacotes a serem recebidos, e logo você pode quebrar o laço com um comando *break*. Caso contrário, vamos copiar o novo dado para o PIC.

1.2. Leia um novo pacote do *transceiver*.

Para ler o pacote do *transceiver*, utilize o comando R_RX_PAYLOAD e logo em seguida chame a função *ReadSPI* quantas vezes forem necessárias para ler todos os dados do pacote. Como nosso pacote só tem 1 byte, só precisa chamá-la uma vez. Lembre-se de colocar o valor lido na variável LED_Display, que é onde fica o valor utilizado para acender os LEDs.

1.3. Sinalize ao *transceiver* que você leu um pacote.

Para sinalizar, basta escrever 1 no bit RX_DR do registrador STATUS, de forma semelhante ao que é feito nos casos de interrupção por TX_DS ou MAX_RT no transmissor. Se não existir mais dados para serem lidos pelo PIC nos buffers de recepção do *transceiver*, o bit RX_EMPTY do registrador FIFO_STATUS vai ser setado para 1, permitindo sair do laço no próximo passo, quando testarmos novamente o valor desse bit.

Pronto. Se você chegou até aqui, então estamos com tudo pronto para testar o experimento.

Realizando a experiência

Para testar os seus programas, você vai precisar montar os circuitos do transmissor e do receptor. Não se esqueça de ter todas as mesmas precauções que você teve na prática passada. Não queremos nenhum PIC, gravador ou computador queimado. Grave o código do transmissor no PIC conectado ao circuito transmissor e o código do gravador no PIC do circuito receptor.

Agora vamos fazer uma coisa ligeiramente diferente. Desconecte o gravador dos circuitos e em cada um deles conecte o pino 1 do PIC na linha de alimentação do circuito (Vcc). Você vai então pegar duas fontes de alimentação e regulá-las para 5V e, então, ligar um circuito em cada uma das fontes. Assim teremos realmente dois circuitos completamente separados e o único link entre eles vai ser a transmissão sem fio.

Com ambos os circuitos alimentados, experimente pressionar o botão. Os LED do circuito transmissor devem mudar pra uma nova configuração, assim como os LED do circuito receptor. Se tudo der certo, parabéns! Você conseguiu realizar a transmissão com sucesso. Tente agora colocar os transmissores mais distantes e tentar enviar novamente o dado e veja qual a distância máxima que você consegue fazer os seus circuitos se comunicarem. Caso o experimento não funcione, verifique a montagem do circuito e o código.



Atenção

Um detalhe bastante importante: provavelmente, você vai realizar essa experiência em um laboratório onde outras pessoas também estarão realizando o mesmo experimento que você e isso pode te atrapalhar: se você estiver usando o mesmo canal de comunicação que outra pessoa ou grupo estiver usando, vocês vão causar interferência um no outro. Portanto, antes de gravar os códigos, decidam entre vocês quais canais cada um vai utilizar. De preferência, mantenha um número razoável de distância entre os canais: enquanto um usa o canal 0, outro usa o 5, outro usa o 12, outro usa o 16 etc. E mesmo que nenhuma outra pessoa esteja usando o mesmo canal que você, lembre-se que a frequência de operação do *transceiver* é aberta. Ou seja, pode existir alguém ou algum aparelho perto de você que esteja usando o seu canal e assim prejudicando o seu experimento. Tente trocar de canal por outro livre caso as coisas não funcionem como esperado.

Então é isso.

E com isso chegamos ao fim da disciplina de Projetos de Sistemas RF. Com essa última experiência, utilizamos bastante conhecimento de programação em microcontroladores e comunicação com dispositivos periféricos ao PIC. O nRF24L01+ é um dispositivo com muitos recursos e não utilizamos todos os recursos que poderíamos. Sinta-se livre para estudar as outras coisas que ele pode fazer, assim como ver outros dispositivos de comunicação sem fio. O funcionamento deles vai ser de alguma forma similar a esses que vimos na disciplina e a comunicação deve utilizar algum dos protocolos que citamos durante o curso (UART, USART, SPI, I2C etc...).

Por fim, tente dar uma olhada ao seu redor. Procure ver a quantidade de dispositivos que utilizam transmissão de dados sem fio. Com certeza você vai encontrar muitos deles. E muitos utilizam formas de comunicação muito parecidas com as que você aprendeu aqui. Além disso, pense naquilo que ainda não usa comunicação sem fio e você poderia melhorá-la adicionando um link de rádio. O conhecimento que você adquiriu nessa disciplina tem um potencial imenso, basta você ter dedicação para estudar mais e explorar as possibilidades ao seu redor. Pense nisso e bons estudos!

Atividade 01

- 1. Como atividade para esta aula, implemente o código completo para comunicar dois microcontroladores, conforme as Figuras 2 e 3**

[Clique aqui](#) para verificar suas respostas.

Respostas

- 1. Como atividade para esta aula, implemente o código completo para comunicar dois microcontroladores, conforme as Figuras 2 e 3**

Não há resposta. Atividade prática para o aluno.

Leitura complementar

- NORDIC. Datasheet do nRF24L01+. Disponível em <<http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01>>. Acesso em: 20 ago. 2012.

Aqui estão todos os dados e informações sobre o *transceiver* utilizado na implementação do sistema SPI.

Resumo

Nesta aula, nós implementamos um sistema de comunicação utilizando o protocolo SPI. Diferente da implementação utilizando RS232, vimos a necessidade de utilizar um dispositivo *transceiver*. A vantagem está no fato de um dispositivo apenas servir como transmissor e receptor ao mesmo tempo. Vimos, também, a utilização de bibliotecas (C18) para realizar esta comunicação.

Autoavaliação

- 1. Qual a diferença (em termos de *hardware*) entre comunicar dispositivos utilizando o protocolo serial RS232 e SPI?**
- 2. Qual a diferença (em termos de *software*) entre comunicar dispositivos utilizando o protocolo serial RS232 e SPI?**
- 3. O que é o nRF24L01+?**
- 4. Qual a diferença entre comunicação serial normal e comunicação mestre-escravo?**
- 5. Cite 3 registradores utilizados na configuração de uma comunicação SPI.**

[Clique aqui](#) para verificar suas respostas.

Respostas

1. Qual a diferença (em termos de *hardware*) entre comunicar dispositivos utilizando o protocolo serial RS232 e SPI?

O hardware utilizado para comunicação via RS232 é bem mais simples que o utilizado na comunicação SPI. No RS232 basicamente se necessita da linha TX e RX para realizar uma comunicação bidirecional. No Caso do SPI, necessita-se de uma linha de clock, uma para comunicação entre o mestre e o escravo (MOSI) e outra entre o escravo e o mestre (MISO), além de uma linha de ativação do escravo (SS, que no caso de existir apenas 1 escravo, pode ser dispensada). A Vantagem do SPI é a possibilidade de utilizar um barramento (um mestre e vários escravos), diferente do RS232 que é ponto a ponto.

2. Qual a diferença (em termos de *software*) entre comunicar dispositivos utilizando o protocolo serial RS232 e SPI?

Na questão de software ambos são parecidos, mas o RS232 ainda é mais simples que o SPI. No RS232 é necessário apenas dizer ao microcontrolador que será utilizado o hardware da comunicação USART, a frequência, bit de paridade e quantidade de bits utilizados (em ambos, transmissor e receptor). Após isso, para transmitir basta escrever no registrador específico e habilitar a transmissão por um bit do registrador de configuração. Na recepção basta monitorar o bit do registrador de configuração que indica uma recepção e ler os dados do registrador de recepção. No caso do SPI, além das configurações do hardware e da velocidade no mestre e da configuração do hardware do escravo, em cada transmissão, o mestre é quem deve iniciar a comunicação. Logo, se o escravo deseja transmitir algo, deve colocar o buffer o dado e aguardar o mestre realizar alguma transmissão. Assim que um dado é recebido, o escravo deve removê-lo do buffer e colocar algo no lugar (se for um dado a ser enviado ou um dado lixo para indicar que não tem nada ao mestre). Por sua vez o mestre deve sempre verificar se o escravo deseja lhe enviar algo. Para isso, deve enviar algum lixo (que não será aproveitado pelo escravo) e ler do buffer algo que o escravo lhe enviou.

3. O que é o nRF24L01+?

É um transceiver (um transmissor e receptor em um único chip) que trabalha com a frequência de 2.4 GHz (faixa ISM) e possui diversas funcionalidades como o auto acknowledgement para confirmar automaticamente ao transmissor a recepção de um pacote corretamente ou para que o transmissor reenvie o pacote no caso de falha e a possibilidade de um receptor se comunique com até 6 transmissores.

4. Qual a diferença entre comunicação serial normal e comunicação mestre-escravo?

Na comunicação serial normal, tanto o transmissor quanto o receptor podem iniciar comunicações. Ambos tem linhas de dados de transmissão e recepção e devem ficar monitorando tais linhas para verificar se há alguma transmissão em algum sentido. No caso da comunicação mestre-escravo apenas o mestre inicia as transmissões, cabendo aos escravos apenas aguardar tais transmissões. Sendo assim, o mestre deve continuamente fazer requisições aos escravos, caso contrário, é possível que algum escravo tenha alguma informação relevante e que só será repassada ao mestre em um momento que esse último tenha algo a transmitir ao escravo em questão.

5. Cite 3 registradores utilizados na configuração de uma comunicação SPI.

SSPSTAT, SSPCON, SSPBUF.

Referências

C18 getting Started. (disponível no pacote do KITPic).

NORDIC. Datasheet do nRF24L01+. Disponível em:
<<http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01>>. Acesso em: 20 ago. 2012.