Projeto de Sistemas RF

Aula 06 - Programação para Comunicação Serial

Apresentação

Já vimos como funcionam as transmissões seriais e paralelas, e como utilizar transmissões seriais USART junto com dispositivos transmissores para transmitir dados através de uma comunicação sem fio. Nesta aula, vamos passar para o aspecto mais prático, que é como colocar a USART para funcionar em um microcontrolador. Como transmitir um dado pela USART de um microcontrolador? Quais registradores preciso conhecer e alterar? Como setar a taxa de transmissão? O que é o "bit de paridade"? E como posso receber esse dado? Essas são algumas das perguntas que serão respondidas nesta aula.

Objetivos

Ao final desta aula, você deverá ser capaz de:

- Entender como ocorre a transmissão e recepção de dados no PIC pela USART.
- Aprender como configurar a USART: quais são os registradores utilizados e qual a função de seus bits.
- Aprender como calcular corretamente a taxa de transmissão a ser utilizada.
- Entender como funcionam as interrupções e como utilizá-las na transmissão e recepção.
- Analisar pseudocódigos utilizados na transmissão e recepção de dados pela USART.

USART em Microcontroladores

Nós estudamos nas aulas anteriores o que é a USART. Vimos que ela é utilizada para transmitir e receber dados de forma serial, de maneira síncrona ou assíncrona. A comunicação USART é usada, principalmente,

em comunicações pelo protocolo RS232, utilizando alguns circuitos (como o popular MAX232) para converter os níveis de tensão utilizados em circuitos lógicos para os usados pelo RS232. Para fazer a comunicação de um microcontrolador com um computador através da porta serial (e logo do protocolo RS232), ou com outro dispositivo que utilize comunicação serial, é preciso mais do que conectar os fios entre os dispositivos. É preciso também saber gerar corretamente o sinal em uma das pernas do microcontrolador.

A boa notícia é que os microcontroladores atuais geralmente vêm com um módulo de comunicação USART. Tudo que precisamos fazer é configurar esse módulo para que ele funcione do jeito que a gente quer. Ou seja, configurar o tipo de comunicação (síncrona ou assíncrona), taxa de transmissão, quantidade de bits por pacote etc. Nós ajustamos essas configurações escrevendo nos registradores específicos do PIC que queremos utilizar. O datasheet do PIC contém a lista de registradores relativos ao módulo USART e qual a função de cada um. Nós veremos a seguir como configurar a transmissão e recepção pela USART em um microcontrolador.

Se liga!

Os exemplos desta aula serão baseados no microcontrolador PIC 18F2550. Outros PICs podem ter diferentes endereços ou configurações de seus módulos USART. Nunca se esqueça de verificar o datasheet do dispositivo que você está utilizando.

Como Ocorre a Transmissão e Recepção em um PIC

Sabemos que o sinal sai do PIC por um de seus pinos, mas existem alguns processos internos que são realizados antes do sinal realmente ser transmitido pelo fio: o que ocorre basicamente é um joguinho de

passagem de dados através de alguns registradores do microcontrolador. Podemos ver uma forma simplificada dessa transmissão assíncrona na **Figura 1**.

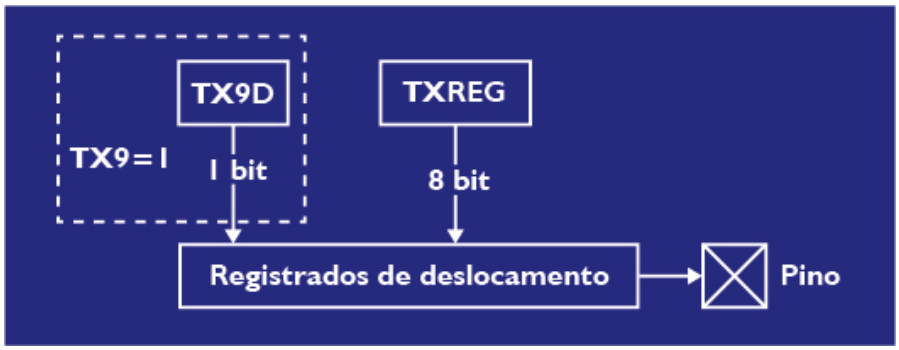

Figura 1 - Diagrama simplificado de transmissão pela USART.

Para transmitir 1 byte (8 bits) usando o módulo USART, escrevemos o byte no registrador TXREG. O módulo então se encarrega de transferir o byte para um registrador de deslocamento, e então transmitir cada bit pelo pino de transmissão. A USART normalmente é utilizada para transmitir pacotes de 8 bits, mas pode também ser utilizada para transmitir 9 bits. Para transmitir 9 bits, é necessário setar para '1' o bit TX9 do registrador TXSTA e armazenar o nono bit no bit TX9D do registrador TXSTA antes de colocar os 8 bits restantes no registrador TXREG.

Na recepção, após o "Start Bit" ser detectado, os bits são recebidos do pino de recepção um a um e vão sendo armazenados em um registrador de deslocamento, até serem armazenados 8 ou 9 bits, dependendo de como o receptor estiver configurado. Para receber 9 bits, o bit RX9 do registrador RCSTA tem que estar setado em '1'. Após serem recebidos todos os bits, o módulo verifica o "Stop Bit", e estando tudo OK, o dado recebido é colocado em um buffer, para depois ser passado para o registrador RCREG. Se existir um nono bit no dado, ele é passado para o bit RX9D do registrador RCSTA. O buffer e os registradores formam uma fila de duas posições. Isso é importante porque caso os dados sejam transmitidos mais rápido do que processados pelo dispositivo receptor, é possível ter 2 dados ainda não processados, um nos registradores RCREG (e RX9D, para o caso de 9 bits de dados) e outro no buffer, enquanto um terceiro está sendo recebido no registrador de deslocamento, diminuindo a chance de ocorrer perda de dados. Como podemos verificar na **Figura 2**

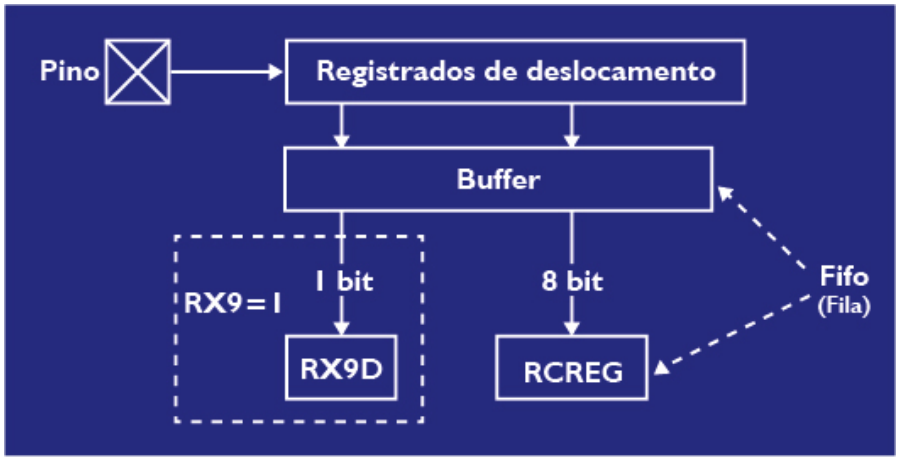

Figura 2 - Recepção assíncrona.

Se liga!

Parece estranho a USART permitir a transmissão de dados em 9 bits, mas isso tem um motivo. Normalmente, o nono bit é um "bit de paridade", gerado pelo transmissor e utilizado pelo receptor para verificar se os outros 8 bits foram transmitidos corretamente. Ele ainda pode ser utilizado como bit de paridade, ou então como um bit de dado a mais do pacote. Uma aplicação interessante é utilizar o bit de paridade como "indicador de endereço": quando existem mais de um receptor, podemos enviar primeiro um dado de endereço (com o nono bit setado em '1') e, logo depois, o dado que queremos enviar (com o nono bit em '0'). Todos recebem o dado de endereço, mas somente o dispositivo com o endereço informado é que recebe o segundo dado.

Configurando a USART – Registradores

Entenderam como acontece a transmissão e recepção pela USART em um PIC? Como eu falei, é um "jogo de registradores". Da mesma forma, configuramos o modo de operação do módulo USART, escrevendo em alguns registradores do PIC. Os principais registradores que influenciam no módulo são:

- **SPBRG** – Conhecido como *Baud Rate Generator*. É alterando aqui que decidimos a velocidade de

transmissão da USART.

- **TXSTA e RCSTA** – são respectivamente *Transmit Status and Control* e *Receive Status and Control*. São usados para setar outras configurações da transmissão e recepção. As configurações no lado do receptor têm que ser compatíveis com os do lado do transmissor para que ocorra a transmissão.

- T**XREG e RCREG** – São conhecidos como *Transmit Data Register* e *Receive Data Register*. Eles são os registradores em que são escritos os dados para transmissão e de onde são lidos os dados quando recebidos da USART. Lembra das figuras que vocês acabaram de ver? São exatamente aqueles registradores.

- **PIR1 e PIE1** – São respectivamente o *Peripheral Interrupt Flag Register* e o *Peripheral Interrupt Enable Register*. Esses registradores permitem que USART gere interrupções no PIC. As interrupções são usadas quando o PIC está executando um código e dados precisam ser transmitidos ou recebidos em *background*. Eles também podem ser lidos para verificar se um dado foi recebido ou pode ser transmitido.

Esses são os registradores que devem ser alterados na configuração. Lembre-se sempre de consultar o datasheet do PIC que está sendo utilizado para ter certeza de como setar corretamente a configuração que você deseja. Os registradores TXSTA e RCSTA definem várias configurações da USART, sendo que geralmente cada bit é responsável por setar alguma coisa. Eles serão detalhados mais adiante. Por enquanto, vamos ver como ajustamos a velocidade de transmissão.

Se liga!

Os registradores no PIC são nada mais do que locais na memória que devem ser configurados na inicialização do seu programa, porém ficar utilizando o endereço do registrador sempre que desejar alterar algo é bem improdutivo, além de abrir brechas para erros. Para a maioria dos microcontroladores existe um cabeçalho (geralmente um ".h" para a linguagem C) com definições dos endereços dos registradores que utilizam o próprio nome do registrador, facilitando assim a vida do programador.

Registradores de Status e Controle

Os dois principais registradores de controle da transmissão e recepção USART são o TXSTA e o RCSTA. É interessante entender os nomes: 'STA' vem de *status*, 'TX' se relaciona com *transmitter* e 'RC' com *receiver*. Como você pode ver, o nome do registrador já diz muito sobre o que ele é. Alguns dos bits dos registradores TXSTA e RCSTA são usados para a configuração, enquanto outros são utilizados para verificar o status atual dos módulos (se ocorreram erros, se houve interrupções etc.). Vamos dar uma olhada mais a fundo nos bits do registrador TXSTA.

Atenção!

Esses bits podem mudar dependendo do microcontrolador que está sendo utilizado! Portanto, mais uma vez, não se esqueça de checar o datasheet do seu microcontrolador!

- **CSRC** – Esse bit não tem importância quando utilizado em comunicações assíncronas. Nas comunicações síncronas, ele informa se o PIC vai funcionar no modo Mestre (bit setado em '1'), que significa que o sinal clock vai ser gerado por ele a partir do BRG, ou se ele vai funcionar no

modo Escravo (bit setado em '0') que significa que o clock vai ser gerado por uma fonte externa.

- TX9 – Informa se a transmissão é de 9 bits (TX9 setado em '1') ou 8 bits (TX9 setado em '0'). Faz com que o bit armazenado em TX9D seja transmitido após os 8 bits de dados que foram escritos no TXREG.

- TXEN – Bit responsável por habilitar as transmissões. Uma vez setado em '1', o registrador TXREG vai resultar em uma transmissão sendo iniciada, se a porta serial estiver habilitada.

- SYNC – Bit utilizado para escolher entre o modo síncrono (SYNC em '1') ou assíncrono (SYNC em '0'). Deve ser modificado para o valor desejado tanto quando estiver configurando um transmissor como quando estiver configurando um receptor.

- SENDB (Send Break Character bit) - Envio do caractere 'Break' (Somente modo assíncrono). Quando setado em "1" envia o caractere de sincronismo após o envio do caractere break; Já, quando setado como "0" significa que a transmissão do caractere de sincronismo foi completada.

- BRGH – Define se vai ser utilizada uma alta taxa de transmissão (valor em '1') ou baixa taxa de transmissão (valor do bit em '0'). Não é utilizado no modo síncrono.

- TRMT – Bit para indicar que existem dados no registrador de deslocamento de transmissão, ou seja, tem uma transmissão em andamento (utiliza valor '0' para isso). Esse bit é usado internamente pelo PIC e logo não pode ser lido pelo programa que você criar.

- TX9D – É onde fica armazenado o nono bit utilizado em uma comunicação que utilize 9 bits por pacote transmitido.

Agora, vamos dar uma olhada nos bits do registrador RCSTA. Ele é principalmente usado para configurar a recepção, mas tem também outras funções como, por exemplo, a do bit SPEN.

- **SPEN** – Setar esse bit para '1' configura todos os pinos do PIC associados à USART para as suas funções. Ele é utilizado para habilitar a porta serial do PIC, tanto no transmissor quanto no receptor.

- **RX9** – Habilita a recepção do nono bit. Faz com que o nono bit recebido seja armazenado no bit RX9D do registrador RCSTA.

- **SREN** – Esse bit habilita a recepção de um dado ('SR' de *single reception*) quando funcionando no modo mestre em uma transmissão síncrona. Não é utilizado no modo escravo ou em uma transmissão assíncrona.

- **CREN** – Habilita a recepção contínua ('CR' de *continuous reception*) quando setado e desabilita a recepção quando resetado.

- **ADDEN** – Habilita o uso do nono bit como um bit indicador de endereço. Se ADDEN = 1, os 8 bits recebidos só serão colocados no buffer de recepção se o nono bit for igual a 1. Caso ADDEN = 0, todos os bits recebidos podem utilizar o nono bit como bit de paridade. O valor de ADDEN não importa se RX9 = 0.

- **FERR** – Bit que indica um erro de *framming*, que significa que o *stop bit* não foi detectado. Esses erros são geralmente causados por taxas de transmissão incorretas.

- **OERR** – Bit que indica um erro de *overrun*, que significa que um novo dado foi recebido enquanto a fila de recepção estava cheia. O dado sendo recebido vai ser perdido e o PIC não vai receber mais nenhum dado até que o CREN seja resetado e setado pelo seu programa.

- **RX9D** – Espaço que vai armazenar o nono bit recebido em transmissões que usem um pacote com tamanho de 9 bits.

Ufa! Parece muita coisa, mas não precisa se preocupar: assim que você começar a programar vai perceber que é fácil configurar a USART do jeito que você quiser. A boa notícia é que você só precisa fazer isso mesmo uma vez no transmissor e uma no receptor e pronto. Caso precise fazer outra comunicação, pode usar o mesmo trecho de código para

configuração, fazendo as adaptações necessárias. Além disso, muitos compiladores vêm com códigos exemplos de configuração da USART. Ou seja, você só vai precisar modificar os valores, ao invés de digitar todo o código novamente.

Configurando a Velocidade de Transmissão

A taxa de transmissão é controlada pelo módulo chamado BRG – *Baud Rate Generator*. Para configurar a velocidade, vamos mexer no registrador SPBRG e nos bits BRGH e SYNC no registrador TXSTA. O bit SYNC define se a transmissão vai ser síncrona (nível lógico '1') ou assíncrona (nível lógico '0'). O bit BRGH define se queremos utilizar uma alta velocidade de transmissão (nível lógico '1') ou uma baixa velocidade de transmissão (nível lógico '0'), permitindo uma maior flexibilidade na hora de configurar a velocidade.

A velocidade de transmissão é definida a partir do valor do registrador SPBRG e do bit BRGH, através das fórmulas:

$$Taxa\ de\ Transmissão = \frac{F_{osc}}{(16(SPBRG+1))}, se\ SYNC = 0,\ BRGH = 1$$

$$Taxa\ de\ Transmissão = \frac{F_{osc}}{(4(SPBRG+1))}, se\ SYNC = 0,\ BRGH = 0$$

$$Taxa\ de\ Transmissão = \frac{F_{osc}}{(4(SPBRG+1))}, se\ SYNC = 1,\ BRGH = X$$

Nas equações anteriores F_{osc} é a frequência de oscilação do PIC. Note que em uma transmissão assíncrona (SYNC = 0), para um mesmo valor de F_{osc} e SPBRG, temos uma taxa de transmissão maior se o BRGH estiver em '1', ou seja, se o módulo estiver configurado para alta velocidade de transmissão. As transmissões síncronas só têm uma fórmula de cálculo da taxa de transmissão, independente do valor de BRGH. Essas fórmulas são importantes para o projetista, mas normalmente você vai querer saber qual valor colocar no registrador SPBRG para conseguir uma determinada taxa de transmissão. Para isso, você pode utilizar essas fórmulas:

$$SPBRG = \frac{F_{osc}}{16 \times Taxa\ de\ Transmissão} - 1, se\ SYNC = 0, BRGH = 1$$

$$SPBRG = \frac{F_{osc}}{64 \times Taxa\ de\ Transmissão} - 1, se\ SYNC = 0, BRGH = 0$$

$$SPBRG = \frac{F_{osc}}{4 \times Taxa\ de\ Transmissão} - 1, se\ SYNC = 1, BRGH = X$$

O registrador SPBRG pode ter valores de 0 a 255 e deve sempre ser um valor inteiro. Quando o valor calculado de SPBRG não for inteiro, deve-se utilizar o valor inteiro mais próximo e logo vai haver uma diferença entre a taxa de transmissão desejada e a que realmente vai ser utilizada. Você pode calcular a taxa de transmissão real utilizando uma das duas primeiras fórmulas, e encontrando o erro da taxa de transmissão, e então verificar se o erro é aceitável para a sua aplicação. Vamos a um exemplo para deixar as coisas mais claras.

Exemplo

Considere um microcontrolador trabalhando a 4MHz que precisa se comunicar a uma taxa de transmissão de 9600 bps com uma porta serial de um computador. A USART teria que ser utilizada no modo assíncrono, logo o valor do bit SYNC seria '0'. Vamos então aos cálculos.

Para $BRGH = 1$:

$$SPBRG = \frac{4000000}{(16 \times 9600)} - 1 = 25.04$$

Para $BRGH = 0$:

$$SPBRG = \frac{4000000}{(64 \times 9600)} - 1 = 5.51$$

Vemos que para BRGH em zero, o valor calculado é de 5.51, sendo que o valor inteiro mais próximo é 6. Por outro lado, se BRGH = 1, temos que o valor calculado é 25.04, que é bem mais próximo de um valor inteiro, causando assim um erro menor na taxa de transmissão. Logo, deve-se utilizar BRGH = 1 e SPBRG = 25. A taxa de transmissão será de 9615, que dá um erro menor que 2% da taxa de transmissão desejada.

Se liga!

Alguns microcontroladores utilizam até 16 bits para armazenar o valor de SPBRG. O valor então vai ser dividido em 2 registradores, cada um de 1 byte. Os registradores com os 8 bits menos significantes são chamados normalmente de SPBRG, enquanto o com 8 bits mais significantes é chamado de SPBRGH ('H' do inglês *high*). A configuração entre utilizar 8 ou 16 bits é feita pelo bit BRG16 do registrador BAUDCON.

Transmitindo Dados

Então, agora você já sabe como configurar a USART para funcionar do modo que você quiser, basta setar os bits de controle corretamente e pronto! Lembre-se de configurar a velocidade corretamente no registrador SPBRG, de setar o bit TXEN no transmissor e o CREN no receptor (para uma recepção contínua). E também se lembre de setar o SPEN para habilitar a porta serial. Os demais bits são dependentes da sua aplicação e você deve setá-los com o devido cuidado. Agora, vamos ver como realizar efetivamente uma transmissão e uma recepção pela USART.

Para transmitir um dado após as configurações e inicializações do módulo, basta escrever o dado no registrador TXREG que ele vai ser transmitido. Se você configurou para transmitir 9 bits, lembre-se de colocar o nono bit no TX9D antes de colocar o restante no TXREG.

Porém, antes de escrever é importante verificar se o último dado que foi escrito já foi transmitido. E por que isso? O que acontece é o seguinte: um dado escrito no TXREG só é transferido para o registrador de deslocamento quando ele está vazio, ou seja, quando o dado anterior for transmitido completamente pela porta serial. Enquanto um dado estiver sendo transmitido, o próximo dado a ser transmitido vai esperar no registrador TXREG. Se tentarmos enviar outro dado nesse momento,

vamos sobrescrever o valor que está em TXREG antes de ele ser transferido para o registrador de deslocamento e transmitido. Ou seja, perderemos um dado!

Podemos evitar esse problema consultando o bit TXIF do registrador PIR1. O bit TXIF é setado sempre que um dado no TXREG é transferido para o registrador de deslocamento. Ou seja, se ele estiver em '0', isso significa que existe algum dado em TXREG esperando para ser transferido. Assim que ele mudar para '1', então significa que TXREG está liberado para ser escrito novamente. Escrever no TXREG vai fazer com que TXIF mude para '0' novamente, até que esse novo dado seja transferido para transmissão. Uma forma simples (e pouco eficiente) de fazer esse teste pode ser visto no pseudocódigo a seguir.

```
1 repita para sempre {
2    Monte um novo DADO;
3    enquanto TXIF for igual a zero {
4          //Faça absolutamente nada.
5    }
6    //Se chegou até aqui, então TXIF é '1'.
7    escreva DADO no registrador TXREG; //Isso vai iniciar a transmissão
8 }
```

Esse procedimento é chamado de "espera ocupada", porque o PIC está literalmente esperando pela mudança em TXIF para continuar o processamento do programa. Essa solução funciona, mas existem maneiras mais eficientes como, por exemplo, utilizar as interrupções do PIC para isso. Mais uma vez, é comum encontrar bibliotecas em alto nível, que já implementem esses testes para você. Portanto, é importante que você tenha conhecimento do procedimento. A não ser que você esteja criando um código em um nível mais baixo, você provavelmente não vai precisar se preocupar com isso.

Recebendo Dados

A recepção acontece de forma semelhante à transmissão. Depois de configurado um receptor, ele deve ficar esperando um dado chegar, para então processá-lo. Quando o novo dado recebido é colocado no registrador RCREG, o bit RCIF no registrador PIR1 é setado, indicando que

existe um novo dado para ser processado. Logo, um código que implementa uma "espera ocupada" pode ser utilizado para receber esse dado, assim como foi utilizado para transmitir. Observe o pseudocódigo a seguir.

```
1 repita para sempre {
2    enquanto RCIF for igual a zero {
3          //Faça absolutamente nada.
4    }
5    //Se chegou até aqui, então RCIF é '1' e existe um novo dado
6    Leia o valor em RCREG e coloque na variável DADO;
7    Processe o DADO recebido;
8 }
```

Utilizar algum código semelhante ao pseudocódigo anterior vai funcionar, porém existe um problema: se o receptor tiver outras tarefas para processar como, por exemplo, acionar motores e ler sensores, pode acontecer de ele nunca receber um dado e nunca sair da "espera ocupada", e logo nunca vai realizar os outros processamentos. Outro pseudocódigo que não teria esse problema seria o seguinte:

```
1 repita para sempre {
2    se RCIF for igual a '1' {
3          Leia o valor em RCREG e coloque na variável DADO;
4          Processe o DADO recebido;
5    }
6    Realize os outros processamentos;
7 }
```

Você consegue ver a diferença? Nesse código, estamos testando se RCIF é igual a '1', ou seja, se tem novo dado, e só fazemos alguma coisa quando existir esse novo dado. Caso não exista novo dado a processar, o programa simplesmente passa para as outras tarefas, ao contrário do programa anterior, que "espera enquanto não tiver um dado para receber". Mesmo assim, essa não é a melhor maneira de receber um dado. Se os "outros processamentos" do programa consumirem muito tempo, pode acontecer de os dados chegarem pela USART mais rapidamente do que o programa vai conseguir processar, e em algum tempo vai ocorrer um erro de *overrun*. A maneira mais correta de tratar a recepção dos dados é utilizando as interrupções do PIC, como vamos ver a seguir.

Utilizando Interrupções na USART

Ambas as formas de receber dados que vimos anteriormente apresentaram possíveis problemas. Uma forma de evitar tais problemas é utilizar as interrupções do PIC para receber os dados. Mas o que é uma interrupção? Como o próprio nome diz, é uma forma de "interromper" o processamento do PIC, e de fato é isso que acontece. Ao utilizar interrupções, o PIC não se preocupa em verificar a cada momento se chegaram novos dados. Quando os dados chegam, o sistema de interrupções avisa ao PIC e ele automaticamente desvia o processamento atual para o processamento da USART, que no caso da recepção é copiar o dado do RCREG para outra variável e, logo depois, ele volta para o que estava fazendo antes de ser interrompido. Ou seja, o dado vai ser processado sempre que o PIC chegar, diminuindo assim a chance de um erro de *overrun* e, ao mesmo tempo, ele não vai parar as demais tarefas para ficar esperando um dado.

Mas, como você faz para utilizar as interrupções? Bem, primeiramente, você vai ter que configurar o PIC para tal. As interrupções da USART são controladas por 3 registradores: o INTCON, o PIE1 e o PIR1.

- **INTCON** – Nesse registrador, estamos interessados nos bits GIE (de *Global Interrupt Enable*) e PEIE (de *Peripheral Interrupt Enable*). Ambos devem estar setados (valores em '1') para que as interrupções da USART aconteçam.

- **PIE1** – Esse registrador contém os bits TXIE e RCIE, que permitem habilitar independentemente as interrupções para a transmissão e recepção. Caso você deseje trabalhar somente com interrupções de transmissão, sete o TXIE e resete RCIE. Se deseja só interrupções de recepção, faça o contrário: resete o TXIE e sete o RCIE. Para trabalhar com ambas interrupções, sete ambos os bits

- **PIR1** – Nesse registrador ficam os bits que efetivamente acionam as interrupções: o TXIF para transmissão e RCIF para a recepção. Quando um desses bits for setado enquanto os bits relativos do INTCON e do PIE1 estiverem setados, uma interrupção vai ocorrer.

- O bit RCIF é setado sempre que tem dados novos no registrador RCREG e é resetado quando não tiver mais dados na Fila de recepção. Ou seja, se você ler o RCREG, o bit RCIF vai ser automaticamente resetado se não existir nenhum outro dado para ser recebido na fila.

- O bit TXIF é resetado sempre que um dado é escrito no TXREG e é setado quando o dado é movido para o registrador de deslocamento. Portanto, uma interrupção vai ocorrer sempre que um novo dado puder ser transmitido. Logo, se não existirem mais dados para serem escritos no TXREG (ou seja, nada mais para ser transmitido), você deve resetar o bit TXIF para evitar que novas interrupções aconteçam. As interrupções de transmissão podem ser habilitadas novamente quando existirem novos dados para serem transmitidos. Para tal, basta setar o TXIF e isso vai imediatamente causar uma interrupção.

Como você pode ver, também é possível utilizar interrupções para a transmissão de dados, apesar de ser bem mais comum utilizá-las na recepção.

Em termos de código, como você vai implementar a interrupção depende da linguagem que está utilizando para programar seu PIC. Geralmente, existem códigos exemplos que você pode utilizar como base para seu código. Utilize-os para entender o que está sendo feito e, então, alterar para a sua aplicação. O que vale a pena citar é que existem dois níveis de interrupção: de alta prioridade e de baixa prioridade. A diferença entre essas interrupções é que as de baixa prioridade podem ser interrompidas pelas interrupções de alta prioridade, caso as últimas ocorram enquanto o PIC está processando o código da de baixa prioridade. Caso você esteja programando um dispositivo que utilize o mesmo nível de interrupção para várias interrupções diferentes (como,

por exemplo, tratar um dado de um sensor de distância) você vai precisar verificar as várias "flags" de interrupção para saber qual delas foi a geradora da interrupção.

Leitura Complementar

Neste link existem várias informações sobre como funcionam e quais são as interrupções do PIC. Understanding Interrupts in PIC Microcontrollers <http://www.microautomate.com/PIC/pic-interrupts.php

Nesta seção do datasheet, existem detalhes funcionais de como funcionam as interrupções no PIC 18F2550. Seção 20.0 do datasheet do PIC 18F2550:
<http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf

Resumo

Nesta aula, você viu como programar um PIC para se comunicar com outro através da USART. Você aprendeu como configurar os modos de operação e a taxa de transmissão, assim como analisou alguns pseudocódigos utilizados para fazer seu PIC transmitir e receber dados, bem como utilizar as interrupções para transmissão e recepção de dados. Vale ressaltar que a programação da USART é importante, pois vários dispositivos transmissores e receptores RF utilizam essa comunicação para a troca de dados com o PIC, e logo entendê-la é muito importante também para a comunicação sem fio.

Autoavaliação

1. Descreva com suas palavras como acontece a transmissão e a recepção de dados em um PIC pela USART.
2. Qual o melhor valor do registrador SPBRG e do bit BRGH para fazer uma transmissão assíncrona com uma taxa de transferência de 9600 bits por segundo, utilizando uma frequência de oscilação de 16MHz? E se fosse síncrona? Qual são os erros nessas taxas de transmissão?

3. Quais os passos que devem ser seguidos para configurar uma transmissão assíncrona em que sejam transmitidos 9 bits por pacote, em um microcontrolador trabalhando a 16MHz e com taxa de transmissão de 9600 bits por segundo? Descreva tanto os passos do PIC transmissor quanto os do receptor. (Dica: procure no datasheet mostrado nas Leituras Complementares).

4. Quais os passos que devem ser seguidos para configurar uma transmissão síncrona de 8 bits no modo master, em um microcontrolador trabalhando a 16MHz e com taxa de transmissão de 9600 bits por segundo? Descreva tanto os passos do PIC transmissor quanto os do receptor. (Dica: procure no datasheet mostrado nas Leituras Complementares).

5. Como configurar um microcontrolador para receber um dado utilizando interrupções?

Referências

MICHOCHIP. **USART**: using the USART in Asynchronous mode. Disponível em: <http://www.eti.pg.gda.pl/katedry/ksg/dydaktyka/dla_studentow//usart.pdf>. Acesso em: 17 maio 2012a

_____. **Datasheet do PIC 18F2550**. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>. Acesso em: 17 maio 2012b.