Trabalho prático N.º 8

Objetivos

- Compreender os mecanismos básicos que envolvem a comunicação série assíncrona.
- Implementar funções básicas de comunicação série através de uma UART, usando as técnicas de *polling* e de interrupção.

Introdução

A UART (acrónimo que significa *Universal Asynchronous Receiver Transmitter*) é um canal de comunicação série assíncrona *full-duplex* e é um dos módulos disponíveis no PIC32 que permitem comunicação série. Implementa o protocolo RS232 o que permite, por exemplo, a ligação a um PC (diretamente através de uma porta RS232 ou indiretamente usando uma ligação física USB). O PIC32, na versão usada na placa DETPIC32, disponibiliza 6 UARTs, das quais a UART1 assegura, através de um conversor USB - RS232, a comunicação com o PC.

A UART é constituída, essencialmente, por 3 módulos fundamentais: um módulo de transmissão (TX), um módulo de receção (RX) e um gerador do sinal de relógio comum para a transmissão e receção, vulgarmente designado por gerador de *baudrate* (BRG – *BaudRate Generator*).

Módulo de transmissão

Na Figura 1 apresenta-se o diagrama de blocos simplificado do módulo de transmissão que inclui, para possibilitar uma visão mais completa do sistema, a ligação ao módulo BRG.

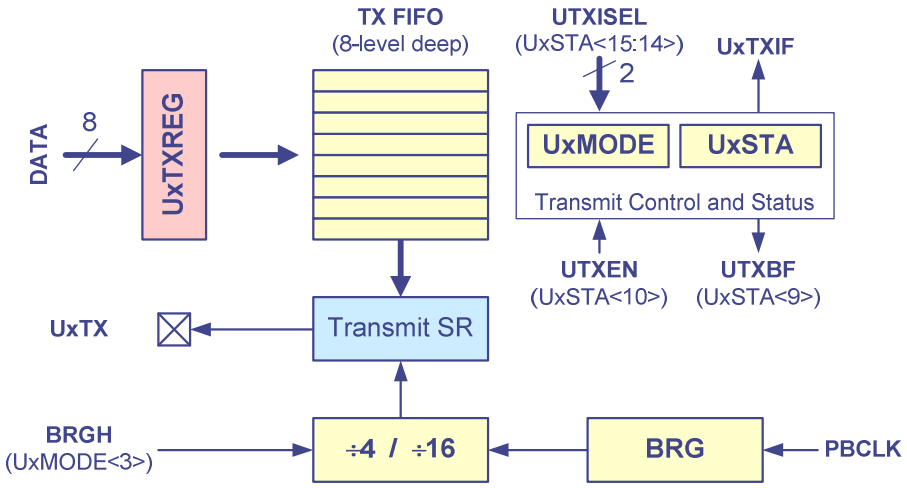

Figura 1. Diagrama de blocos simplificado do módulo de transmissão da UART (perspetiva do programador).

O bloco-base do módulo de transmissão é um *shift register* (*Transmit SR*) que faz a conversão paralelo-série, enviando sucessivamente para a linha série os carateres[1] armazenados no *buffer* de transmissão. Este *buffer* é um FIFO (*First In First Out*) de 8 posições, gerido de forma automática pelo *hardware* e a que o programador acede, indiretamente, através do registo **UxTXREG**[2]. Assim, o envio de informação é feito escrevendo, sucessivamente, os carateres a transmitir no registo **UxTXREG**, que funciona como a cauda do FIFO ("TX FIFO"). Os carateres armazenados no FIFO são copiados sucessivamente para o *shift-register* de transmissão.

[1] A UART permite a comunicação com palavras de 8 e 9 bits. Nestas aulas consideramos apenas a comunicação com palavras de 8 bits, a que chamamos, carateres.

[2] A letra **x** minúscula que aparece no nome de todos os registos deve ser substituída pelo número da UART pretendida (entre 1 e 6). Por exemplo, para a UART1 o registo de transmissão é U1TXREG.

O relógio que determina a taxa de transmissão é obtido através de uma divisão por 4 ou por 16 (bit **BRGH** do registo **UxMODE**) do relógio gerado pelo módulo BRG (f_{BRG} na Figura 3).

A ativação deste módulo é feita através do bit **UTXEN** do registo **UxSTA** – após *reset* ou *power-on* todas as 6 UART estão desligadas e os módulos de transmissão e receção de cada uma delas não estão ativos.

Módulo de receção

A Figura 2 apresenta o diagrama de blocos simplificado do módulo de receção onde se inclui também o módulo BRG. Como se pode observar, a estrutura do módulo de receção é semelhante ao módulo de transmissão, tendo como elemento central um *shift-register* que faz, neste caso, a conversão série-paralelo. Os sucessivos carateres recebidos da linha série são colocados no *buffer* de receção que é, tal como no módulo de transmissão, um FIFO de 8 posições ("RX FIFO"). A gestão do "RX FIFO" é, igualmente, realizada pelo *hardware* de forma automática e transparente para o programador que interage com esta estrutura de dados indiretamente através do registo **UxRXREG**.

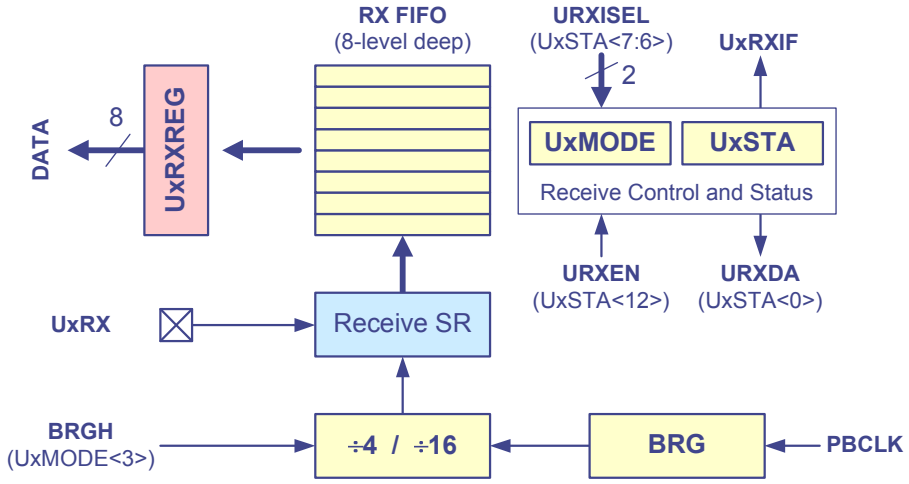

Figura 2. Diagrama de blocos simplificado do módulo de receção da UART (perspetiva do programador).

O relógio que determina a taxa de receção é o mesmo que é usado no módulo de transmissão, e é obtido através de uma divisão por 4 ou por 16 (bit **BRGH** do registo **UxMODE**) do relógio gerado pelo módulo BRG.

O bit **URXEN** do registo **UxSTA** ativa este módulo que, tal como o de transmissão, fica inativo após *reset* ou *power-on*.

Gerador de *baudrate*

O gerador de *baudrate* apresenta uma estrutura semelhante aos *timers* de utilização geral (sem o módulo *prescaler*)) já utilizados anteriormente (Figura 3).

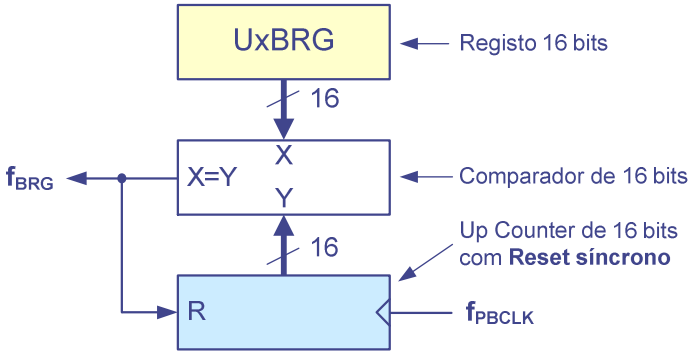

Figura 3. Diagrama de blocos do módulo gerador de *baudrate*.

O sinal de relógio à entrada deste módulo é o *Peripheral Bus Clock* que tem, na placa DETPIC32, uma frequência de 20 MHz.

A frequência à saída deste módulo é, então,

$$f_{BRG} = f_{PBCLK} / (UxBRG + 1)$$

em que **UxBRG** representa a constante armazenada no registo com o mesmo nome. O sinal à saída deste módulo é posteriormente dividido por 4 ou por 16, em função da configuração do bit **BRGH** do registo **UxMODE**, representando a frequência do sinal obtido a taxa de transmissão/receção (*baudrate*) da UARTx.

No caso em que **BRGH** (**UxMODE<3>**) é configurado com o valor 1, o fator de divisão referido anteriormente é 4, pelo que a taxa de transmissão/receção é dada por:

$$baudrate = f_{PBCLK} / (4 * (UxBRG + 1))$$

No caso em que **BRGH** (**UxMODE<3>**) é configurado com o valor 0, o fator de divisão é 16, sendo então a taxa de transmissão/receção dada por:

$$baudrate = f_{PBCLK} / (16 * (UxBRG + 1))$$

ou

$$UxBRG = (f_{PBCLK} / (16 * baudrate)) - 1$$

Ou ainda, e de modo a evitar o erro de truncatura que se terá ao realizar as operações com inteiros:

$$UxBRG = ((f_{PBCLK} + 8 * baudrate) / (16 * baudrate)) - 1$$

O módulo BRG é comum aos módulos de transmissão e receção, pelo que a taxa de transmissão é igual à taxa de receção.

Configuração da UART

A configuração completa da UART, sem interrupções, é efetuada nos seguintes passos:

1) Configurar o gerador de *baudrate* de acordo com a taxa de transmissão/receção pretendida (registo **UxBRG** e bit **BRGH** do registo **UxMODE**).
2) Configurar os parâmetros da trama: dimensão da palavra a transmitir (número de *data bits*) e tipo de paridade (bits **PDSEL<1:0>** do registo **UxMODE**); número de *stop bits* (bit **STSEL** do registo **UxMODE**).
3) Ativar os módulos de transmissão e receção (bits **UTXEN** e **URXEN** do registo **UxSTA**).
4) Ativar a **UART** (bit **ON** do registo **UxMODE**).

A ativação de uma dada UARTx e dos correspondentes módulos de transmissão e receção configura automaticamente o porto de transmissão (**UxTX**) como saída e o porto de receção (**UxRX**) como entrada, sobrepondo-se esta configuração à efetuada através do(s) registo(s) **TRISx**.

Procedimento de transmissão – *polling*

O procedimento de transmissão envolve sempre a verificação da existência de espaço no respetivo FIFO. O bit **UTXBF** (*Transmit Buffer Full*) do registo **UxSTA** é um bit de *status* que, quando ativo, indica que o FIFO de transmissão está cheio. É, assim, necessário esperar que o bit **UTXBF** fique inativo para colocar nova informação no FIFO. Uma função para transmitir 1 carater deverá então ser estruturada do seguinte modo:

```
void putc(char byte2send)
{
   // wait while UTXBF == 1
   // Copy byte2send to the UxTXREG register
}
```

Procedimento de receção – *polling*

No caso da receção, o bit **URXDA** (*Receive Data Available*) do registo **UxSTA** indica, quando ativo, que está disponível para ser lido pelo menos 1 carater no FIFO de receção. O procedimento genérico de leitura envolve, assim, o *polling* desse bit, esperando que ele fique ativo para proceder depois à leitura do carater recebido. A estrutura de uma função bloqueante de leitura de 1 carater poderá então ser a seguinte:

```
char getc(void)
{
   // Wait while URXDA == 0
   // Return U1RXREG
}
```

Na situação em que o FIFO de receção está cheio e um novo carater foi lido da linha série, ocorre um erro de *overrun*, sinalizado no bit **OERR** do registo **UxSTA**. Nesta situação, o último carater lido da linha série é descartado e, enquanto o bit **OERR** estiver ativo, a UART não recebe mais nenhum carater. Assim, a função de receção de um carater deve incluir o teste da *flag* **OERR** de modo a efetuar o respetivo *reset* no caso de ocorrência de erro de *overrun* (o *reset* dessa *flag* elimina toda a informação existente no *buffer* de receção). A função anterior poderá então ser re-escrita do seguinte modo:

```
char getc(void)
{
   // If OERR == 1 then reset OERR
   // Wait while URXDA == 0
   // Return U1RXREG
}
```

Configuração da UART com interrupções

O modo como as interrupções são geradas é configurável através dos bits **UTXISEL<1:0>** e **URXISEL<1:0>** do registo **UxSTA**. Configurando esses bits com a combinação binária "**00**", são geradas interrupções nas seguintes situações:

- Receção: uma interrupção é gerada quando o FIFO de receção tem, pelo menos, um novo carater para ser lido; a rotina de serviço à interrupção pode efetuar a respetiva leitura do registo **UxRXREG**.
- Transmissão: uma interrupção é gerada quando o FIFO de transmissão tem, pelo menos, uma posição livre; a rotina de serviço à interrupção pode colocar no registo **UxTXREG** um novo carater para ser transmitido;

Para além destas duas configurações específicas, é ainda necessário configurar, como para todas as outras fontes de interrupção, os bits de *enable* das interrupções e os bits que definem a prioridade. Assim, para a ativação da interrupção de receção é necessário configurar o bit **UxRXIE** (*receive interrupt enable*) e para a interrupção de transmissão o bit **UxTXIE** (*transmit interrupt enable*). Para a definição da prioridade devem ser configurados os 3 bits **UxIP** (a configuração de prioridade é comum a todas as fontes de interrupção de uma UART).

Cada UART pode ainda gerar uma interrupção quando é detetada uma situação de erro na receção de um carater. Os erros detetados são de três tipos: erro de paridade, erro de *framing*

e erro de *overrun*. Se se pretender fazer a deteção destes erros por interrupção, então é também necessário ativar essa fonte de interrupção, isto é, ativar o bit **UxEIE**.

Finalmente, é importante referir que a cada UART está atribuído um único vetor para as 3 possíveis fontes de interrupção. Essa situação obriga a que a identificação da fonte de interrupção tenha que ser feita por *software* na rotina de serviço à interrupção (rotina associada ao vetor atribuído à UART que está a ser usada). A identificação é feita através do teste dos *interrupt flag bits* de cada uma das três fontes possíveis de interrupção, isto é, **UxRXIF**, **UxTXIF** e **UxEIF**.

Trabalho a realizar

Parte I

1. Configure a UART1 (sem interrupções), com os seguintes parâmetros de comunicação: 115200 bps, sem paridade, 8 *data bits*, 1 *stop bit* (consulte os registos **UxMODE** e **UxSTA** no manual da UART). No cálculo da constante de configuração do gerador de *baudrate* considere um fator de divisão do relógio de 16.

```
int main(void)
{
   // Configure UART1:
   // 1 - Configure BaudRate Generator
   // 2 – Configure number of data bits, parity and number of stop bits
   //     (see U1MODE register)
   // 3 – Enable the trasmitter and receiver modules (see register U1STA)
   // 4 – Enable UART1 (see register U1MODE)
}
```

2. Acrescente ao código que escreveu no exercício anterior uma função para o envio de um carater para a porta série. No programa principal envie, usando essa função, o carater '**+**' a uma frequência de 1 Hz.

```
int main(void)
{
   // Configure UART1 (115200, N, 8, 1)
   while(1)
   {
      putc('+');
      // wait 1 s
   }
   return 0;
}

void putc(char byte2send)
{
   // wait while UTXBF == 1
   // Copy byte2send to the UxTXREG register
}
```

3. Escreva e teste uma função para enviar para a porta série uma *string* (*array* de carateres terminado com o carater 0). Utilize a função **`putc()`** para enviar cada um dos carateres da *string*.

```
int main(void)
{
   // Configure UART1 (115200, N, 8, 1)
   while(1)
   {
      puts("String de teste\n");
      // wait 1 s
   }
   return 0;
}

void puts(char *str)
{
   ... // use putc() function to send each charater ('\0' should not
       //  be sent)
}
```

4. Generalize o código de configuração da UART1 que escreveu no ponto 1, implementando uma função que aceite como argumentos o *baudrate*, o tipo de paridade e o número de *stop bits* (o número de *data bits* deverá ser fixo e igual a 8). Para a configuração do *baudrate* utilize um fator de divisão do relógio de 16.

   Valores possíveis para os argumentos de entrada da função:

   - Baudrate: 600 a 115200
   - Paridade: 'N', 'E', 'O' (sem paridade, paridade par (*even*), paridade ímpar (*odd*))
   - Stop bits: 1 ou 2

   Valores a considerar por defeito, isto é, sempre que os argumentos de entrada não estejam nas gamas definidas anteriormente:

   - Baudrate: 115200
   - Paridade: 'N'
   - Stop bits: 1

```
void configUart(unsigned int baud, char parity, unsigned int stopbits)
{
   // Configure BaudRate Generator
   // Configure number of data bits (8), parity and number of stop bits
   // Enable the trasmitter and receiver modules
   // Enable UART1
}
```

5. Teste a função anterior com o programa que escreveu no ponto 3, com diferentes valores de entrada. Exemplo: **`600,'N',1`**; **`1200,'O',2`**; **`9600,'E',1`**; **`19200,'N',2`**; **`115200,'E',1`**. No PC execute o programa **`pterm`** com os mesmos parâmetros com que configurou a UART1 (por exemplo: **`pterm 1200,O,8,2`**).

6. Neste exercício pretende-se avaliar a forma de funcionamento da UART na transmissão. Para isso vamos medir o tempo que a UART demora a transmitir *strings* com diferente dimensão, partindo sempre de uma situação de repouso, isto é, em que garantidamente a UART não tem nenhuma informação pendente para ser transmitida. Para garantir essa condição de início vamos usar o bit **`TRMT`** do registo **`UxSTA`** (**`UxSTA<8>`**) que, quando a 1, indica que o TX FIFO e o *transmit shift register* estão ambos vazios.

```
int main(void)
{
   configUart(115200,'N',1);   //default "pterm" parameters (8 data bits)
   // config RB6 as output
   while(1)
   {
      // Wait until TRMT == 1
      // Set RB6
      puts("12345");
      // Reset RB6
   }
   return 0;
}
```

Meça, com o osciloscópio, os tempos a 1 e a 0 do sinal no porto RB6 e anote os valores obtidos. Repita o teste, sequencialmente, com as seguintes *strings*: "123456789", "123456789A", "123456789AB", anotando todos os valores. Compare os valores obtidos e tire conclusões. Determine o tempo necessário para transmitir cada carater.

7. Repita o exercício anterior configurando a UART com *baudrates* de 57600 e 19200 bps.
8. Retire, do programa anterior, a condição de espera e envie a *string* inicial i.e. "12345". Meça novamente o tempo a 1 do sinal no porto RB6, compare esse valor com o que obteve anteriormente para a mesma *string* e tire conclusões.

```
...
while(1)
{
   // Set RB6
   puts("12345");
   // Reset RB6
}
...
```

9. Acrescente agora a função para ler um carater da linha série e teste-a. Para isso, configure a UART1 com os parâmetros por defeito do **`pterm`** e, em ciclo infinito, reenvie para a linha série todos os carateres recebidos (procedimento geralmente referido como "eco dos carateres"). Altere também a função **`getc()`** para detetar erros na receção de carateres (paridade, *framing* e overrun).

```
int main(void)
{
   configUart(115200,'N',1);    // default "pterm" parameters
   while(1)
   {
      putc( getc() );
   }
   return 0;
}

char getc(void)
{
   // If OERR == 1 then reset OERR
   // Wait while URXDA == 0
   // If FERR or PERR then
   //   read UxRXREG (to discard the character) and return 0
   // else
   //   Return U1RXREG
}
```

Parte II

1. Configure a UART para gerar uma interrupção na receção de um carater. Altere o programa anterior para fazer o eco do carater recebido na respetiva rotina de serviço.

```
int main(void)
{
   configUart(115200,'N',1);    // default "pterm" parameters
                                //  with RX interrupt enabled
   EnableInterrupts();
   while(1);
   return 0;
}

void _int_(VECTOR_UART1) isr_uart1(void)
{
   putc(U1RXREG);
   // Clear UART1 rx interrupt flag
}
```

2. Retome o programa que escreveu no último exercício do trabalho prático n.º 7 e inclua a função de configuração da UART1 e de receção por interrupção, que implementou no exercício anterior. Faça ainda as seguintes alterações:

   a) Envie o valor da tensão medida (i.e. o valor que é enviado para os *displays*) para o PC a um ritmo de 1 Hz. Para isso implemente um contador módulo 100 na rotina de serviço à interrupção do Timer T3 (que é evocada, recorde-se, a cada 10 ms) e envie para a porta série os dois dígitos da variável "**`voltage`**", codificados em ASCII.

   b) Quando for lida da porta série a tecla "**L**" envie o valor mínimo e o valor máximo da tensão que o sistema mediu desde o seu arranque. Para isso declare duas variáveis globais, "**`voltMin`**" e "**`voltMax`**", inicialize-as adequadamente e atualize-as na rotina de serviço à interrupção do módulo A/D, isto é, na rotina que calcula um novo valor de tensão.

```
void _int_(VECTOR_ADC) isr_adc(void)
{
   (...)
   // Convert voltage amplitude to decimal. Assign it to "value2display"
   // Update variables "voltMin" and "voltMax"
   IFS1bits.AD1IF = 0;               // Reset AD1IF flag
}

void _int_(VECTOR_TIMER3) isr_T3(void)
{
   static int counter = 0;

   // Send "value2display" global variable to displays
   if(++counter == 100)
   {
      counter = 0;
      // send voltage to the serial port UART1
   }
   // Clear T3 interrupt flag
}

void _int_(VECTOR_UART1) isr_uart1(void)
{
   if(U1RXREG == 'L')
   {
      // Send "voltMin" and "voltMax" to the serial port UART1
   }
   // Clear UART1 rx interrupt flag
}
```

3. Pretende-se agora detetar, por interrupção, a ocorrência de erros de comunicação na receção (paridade, *framing* e *overrun*). Altere adequadamente a função de configuração da UART1 (veja a introdução para mais detalhes) e acrescente na rotina de serviço à interrupção o tratamento da interrupção de erro.

```
void _int_(VECTOR_UART1) isr_uart1(void)
{
   // If U1EIF set then
   //    if overrun error then clear OERR flag
   //    else read U1RXREG to a dummy variable
   //    clear UART1 error interrupt flag
   //
   // If U1RXIF set then
   //    if(U1RXREG == 'L')
   //       (...)
   // Clear UART1 rx interrupt flag
}
```

Elementos de apoio

- Slides das aulas teóricas.
- PIC32 Family Reference Manual, Section 08 – Interrupts.
- PIC32 Family Reference Manual, Section 21 – UART.
- PIC32MX5XX/6XX/7XX, Family Data Sheet, Pág. 74 a 76.