Aula prática N.º 10

Objetivos

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

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 UART2 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, designado por gerador de *baudrate* (BRG – *BaudRate Generator*).

Módulo de transmissão

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

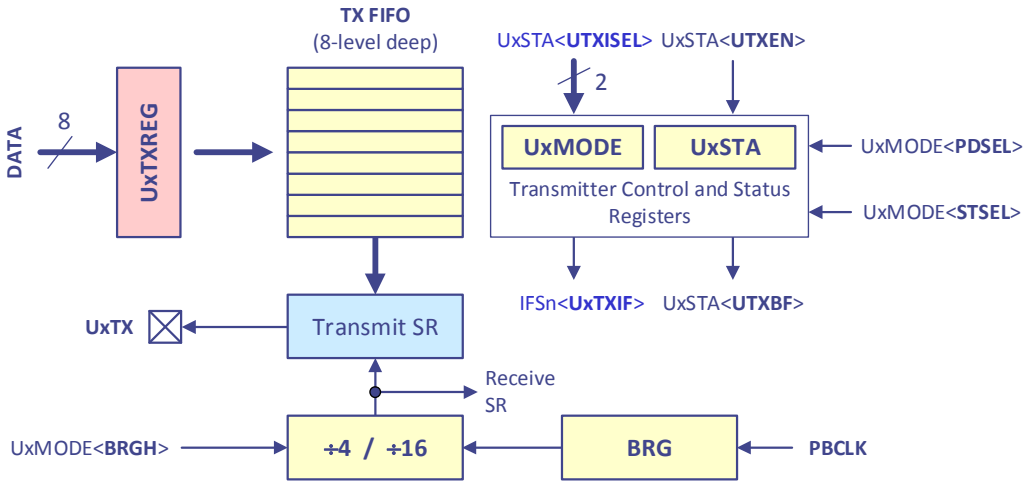

Figura 1. Diagrama de blocos simplificado do módulo de transmissão da UART (perspetiva do programador).

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.

A ativação do módulo de transmissão 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 estão inativos.

[1] A UART permite a comunicação com palavras de 8 e 9 bits. Nestas aulas consideramos apenas a comunicação com palavras de 8 bits (que designamos por carateres).

[2] A letra **`x`** minúscula no nome dos registos (e nos bits **`UxTXIF`** e **`UxRXIF`**) deve ser substituída pelo número da UART pretendida (entre 1 e 6). Por exemplo, para a UART2 o registo de transmissão é **`U2TXREG`**.

A configuração dos parâmetros da trama (comum aos módulos de transmissão e receção) é feita no registo **UxMODE**, nomeadamente: a dimensão da palavra a transmitir (número de bits de dados) e o tipo de paridade, definidos pelo campo **PDSEL** (2 bits); o número de bits de stop, configurado através do bit **STSEL** do mesmo registo.

O relógio que determina a taxa de transmissão (e receção) 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**).

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**.

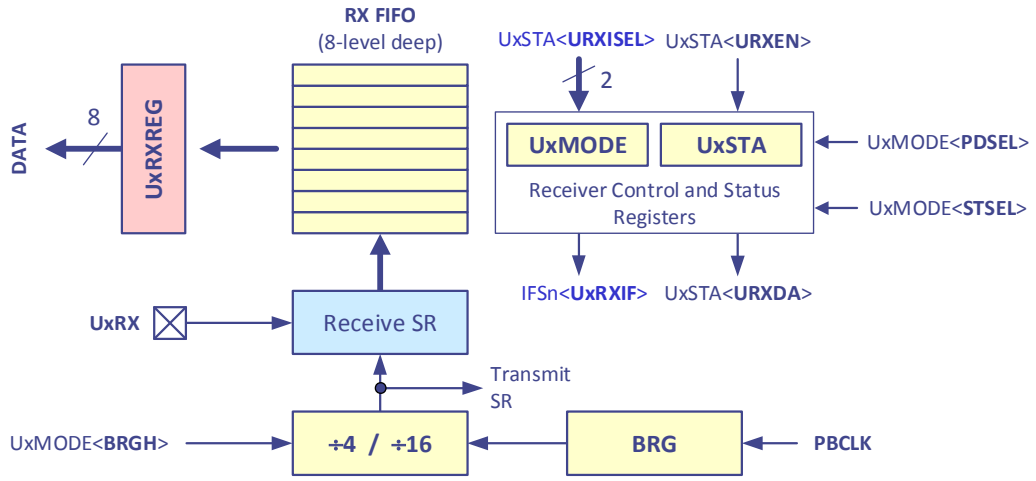

Figura 2. Diagrama de blocos simplificado do módulo de receção da UART (perspetiva do programador).

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

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.

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**).

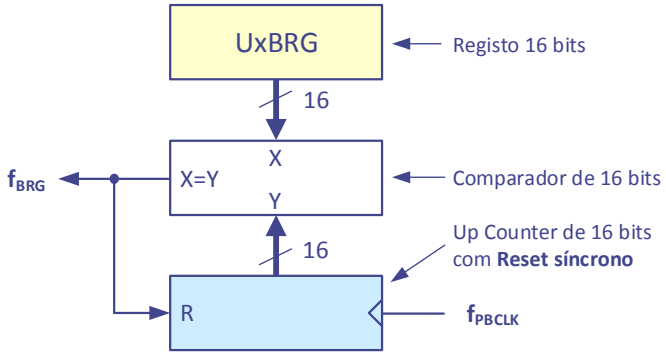

Figura 3. Diagrama de blocos do módulo gerador de *baudrate*.

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,

$$f_{BRG} = \frac{f_{PBCLK}}{UxBRG + 1}$$

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:

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

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:

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

Ou seja, a constante de configuração do módulo BRG, para um dado *baudrate*, é calculada como:

$$UxBRG = \frac{f_{PBCLK}}{16 * baudrate} - 1$$

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

$$UxBRG = \frac{f_{PBCLK} + 8 * baudrate}{16 * baudrate} - 1$$

O módulo BRG é comum aos módulos de transmissão e receção, pelo que a taxa de transmissão é igual à taxa de receção. Na expressão anterior note que **8*baudrate/(16*baudrate)=0.5**.

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** 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 (UxSTA register)
   // 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 (UxSTA register)
   // Return UxRXREG
}
```

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, de uma forma completa, 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 U2RXREG
}
```

É importante notar que o registo **UxRXREG** não pode, em muitas situações, ser usado como se de uma variável se tratasse. Isso acontece porque a cada nova leitura do registo **UxRXREG** o FIFO de receção é atualizado e, consequentemente, em N leituras consecutivas são devolvidos os N carateres residentes nas N posições mais antigas desse mesmo FIFO. No caso particular de o FIFO conter apenas um carater, uma segunda leitura devolve um valor indeterminado.

Por exemplo, usar o registo **UxRXREG** diretamente numa estrutura de decisão pode conduzir a testes sobre valores diferentes, consoante o conteúdo do FIFO no momento da leitura. O valor lido do registo **UxRXREG** deve então ser armazenado numa variável e só depois deve ser usado, como no exemplo seguinte (para a UART2):

```
Utilização ERRADA:
   if(U2RXREG == 'x')
      ...
   else if(U2RXREG == 'y')
      ...
```

```
Utilização correta:
   char c;
   c = U2RXREG;
   if(c == 'x')
      ...
   else if(c == 'y')
   ...
```

Trabalho a realizar

Parte I

1. Configure a UART2, com os seguintes parâmetros de comunicação: 115200 bps, sem paridade, 8 *data bits*, 1 *stop bit* (**115200,N,8,1**) - 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 (fator de sobreamostragem) de 16.

```
int main(void)
{
   // Configure UART2:
   // 1 - Configure BaudRate Generator
   // 2 - Configure number of data bits, parity and number of stop bits
   //     (see U2MODE register)
   // 3 - Enable the trasmitter and receiver modules (see register U2STA)
   // 4 - Enable UART2 (see register U2MODE)
}
```

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 UART2 (115200, N, 8, 1)
   while(1)
   {
      putc('+');
      // wait 1 s
   }
   return 0;
}

void putc(char byte)
{
   // wait while UART2 UTXBF == 1
   // Copy "byte" to the U2TXREG register
}
```

3. Repita o exercício anterior, considerando um fator de sobreamostragem de 4.

4. 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 UART2 (115200, N, 8, 1)
   while(1)
   {
      putstr("String de teste\n");
      // wait 1 s
   }
   return 0;
}

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

5. Teste o programa anterior com diferentes valores de configuração da UART2: **600,N,8,1**; **1200,O,8,2**; **9600,E,8,1**; **19200,N,8,2**; **115200,E,8,1**; **230400,E,8,2**; **460800,O,8,1**; **576000,N,8,1**.
   Notas:
   - Para os valores de *baudrate* **230400**, **460800** e **576000**, terá que usar, obrigatoriamente, um fator de sobreamostragem de 4.
   - No PC o programa **pterm**[3] tem que ser executado com os mesmos parâmetros com que configurou a UART2 (por exemplo: **pterm 600,N,8,1**).

6. Acrescente agora a função para ler um carater da linha série e teste-a. Para isso, configure a UART2 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").

```
int main(void)
{
   // Configure UART2 (115200, N, 8, 1)
   while(1)
   {
      // Read character using getc()
      // Send character using putc()
   }
   return 0;
}

char getc(void)
{
   // Wait while URXDA == 0
   // Return U2RXREG
}
```

7. Altere a função **main()** do exercício anterior de modo a implementar um contador crescente módulo 10, incrementado a uma frequência de 5 Hz. O valor do contador deve ser transmitido em binário pela UART2, ao mesmo ritmo. Para isso deve usar, exclusivamente, as funções **putstr()** e/ou **putc()**.

Parte II

1. Altere a função **main()** de modo a configurar a **UART1** com os mesmos parâmetros que usou inicialmente para a UART2 (**115200,E,8,1**) e altere a função **putc()** de modo a enviar um carater para a UART1 (designe a nova função por **putc1()**).
2. Na função **main()** transmita, em ciclo infinito a cada 10 ms, o valor **0x5A**, usando a função **putc1()**. Com o osciloscópio observe o sinal na linha **U1TX** (disponível no ponto de teste **OC4**). Em particular, identifique o *start* bit, o bit de paridade, o *stop* bit e cada um dos bits de dados da palavra transmitida. Meça ainda o tempo de bit e o tempo total de transmissão de um carater.
3. Repita o exercício anterior com outros valores (por exemplo, **0xA5**, **0xF0**, **0x0F**, **0xFF**, **0x00**) e com outros valores de *baudrate*.

[3] Os parâmetros de comunicação por defeito do programa **pterm** são: **115200,N,8,1**

Parte III

1. 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)
{
   // Configure UART2 (115200, N, 8, 1)
   // config RD11 as output
   while(1)
   {
      // Wait while TRMT == 0
      // Set RD11
      putstr("12345");
      // Reset RD11
   }
   return 0;
}
```

   Meça, com o osciloscópio, os tempos a 1 e a 0 do sinal no porto **RD11** (disponível no ponto de teste **INT4**) 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.

2. Repita o exercício anterior configurando a UART com um *baudrate* de 19200 bps.
3. 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 **RD11**, compare esse valor com o que obteve anteriormente para a mesma *string* e tire conclusões.

```
...
while(1)
{
   // Set RD11
   putstr("12345");
   // Reset RD11
}
...
```

Exercício adicional

1. Generalize o código de configuração da UART2 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 **`576000`** (para valores de *baudrate* superiores a **`115200`** tem que usar um fator de sobreamostragem de 4)
   - 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 configUart2(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 UART2
}
```

Elementos de apoio

- Slides das aulas teóricas (aula 18).
- PIC32 Family Reference Manual, Section 08 – Interrupts.
- PIC32 Family Reference Manual, Section 21 – UART.
- PIC32MX5XX/6XX/7XX, Family Data Sheet, Pág. 74 a 76.

PDF criado em 14/04/2025