

UART (Universal Asynchronous Receiver-Transmitter)

- Es un protocolo de comunicación serie. Con este protocolo se pueden implementar los siguientes tipos de comunicación:

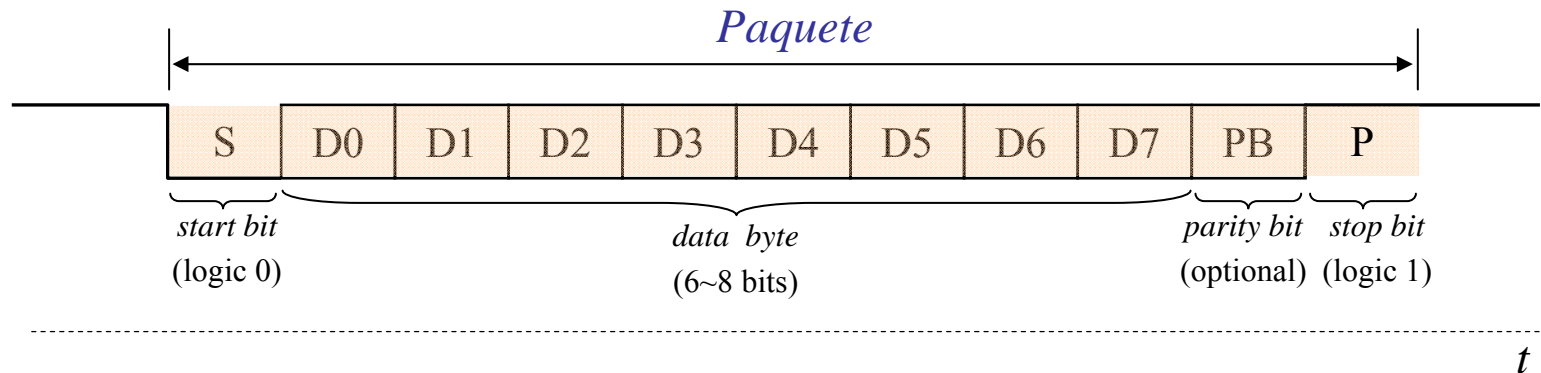
Simplex: transmisión en un único sentido, sin que haya confirmación alguna por parte del receptor de que ha recibido el dato.

Half duplex: transmisión en ambos sentidos, pero no de forma simultánea.

Full duplex: transmisión en ambos sentidos de forma simultánea.

- Cualquier nodo puede iniciar la comunicación.
- La comunicación es asíncrona (\equiv no utiliza una señal de reloj)
- *Baud rate*: es el número de bits transmitidos por segundo

- El formato de un paquete básico es el siguiente: 1 *start bit* (logic 0), 8 *data bits*, 1 *parity bit*, 1 *stop bit* (logic 1).



Nota: cuando no se está transmitiendo un paquete, la línea está a nivel alto.

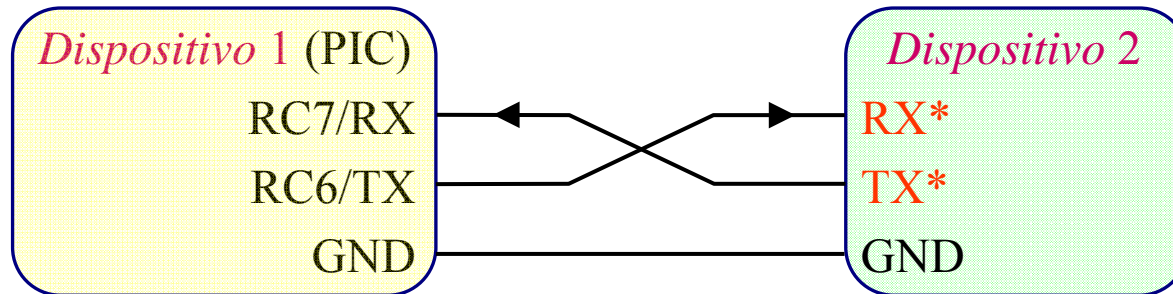
- Los niveles de tensión asociados a los niveles lógicos no están establecidos. En la práctica se utilizan diferentes normas para establecer dichos niveles de tensión, como son: RS-232, RS-422 y RS-485 (ver estándares EIA: *Electronic Industries Alliance*)

▪ Conexiones***:

nomenclatura:

RX \equiv RXD

TX \equiv TXD



▪ La velocidad de transmisión en serie de los bits se mide en *bauds* y debe ser la misma en ambos dispositivos. Los valores habituales son: 0.3k, 1.2k, 2.4k, 9.6k, 19.2k, 57.6k, 115.2k

▪ El protocolo *Uart* utiliza un *bit de paridad* para detectar errores en la transmisión (es opcional... el PIC18F452 no lo tiene implementado)

A continuación se indican las funciones que tiene el compilador MikroC PRO para manejar el módulo Uart de los microcontroladores PIC

▪ *void UARTx_Init*(const unsigned long *baud_rate*); esta función configura e inicializa el módulo *Uart*, para enviar y recibir paquetes con las siguientes características:

_ 1 bit de *start*

_ 8 bits de *datos*

x: número del módulo *Uart* a utilizar

_ 1 bit de *stop*

_ sin bit de *paridad*

_ transmisión *asíncrona*

Nota1: *baud_rate* tiene que ser una *constante* (no puede ser una *variable*)

Nota2: esta función debe ejecutarse antes de ejecutar cualquier otra función relativa al módulo *Uart*

Ejemplo:

```
UART1_Init(9600);
```

```
delay_ms(300); // desde que se ejecuta la función uart1_init() hay que esperar al menos 100mseg  
               // antes de utilizar el módulo uart.
```

- *char* UARTx_Tx_Idle(); devuelve un 1 si el registro en el que se guarda el dato a transmitir está vacío (≡ si el dato anterior ha sido transmitido ≡ si el módulo *Uart* está disponible para enviar un nuevo carácter). Devuelve un 0 en caso contrario.

Ejemplo:

```
if (UART1_Tx_Idle() == 1)
{
    UART1_Write(aux); // envía por Uart el valor guardado por la variable aux
}
```

- *void* UARTx_Write(char *aux*); // envía por Uart el valor guardado por la variable *aux*

Ejemplo:

```
unsigned char dato = 0x47;
...
```

```
UART1_Write(dato); // también se puede poner 0x47 ó 'G' ó 71
```

- *void* UARTx_Write_Text(char *aux); esta función envía por *Uart* los caracteres guardados por el array *aux*. El contenido debe estar limitado a 255 caracteres, siendo el último carácter un 0.

Ejemplo:

UARTx_Write_Text (“*Hola Orense*”);

- *char* UARTx_Data_Ready(); devuelve un 1 si hay un dato disponible para ser leído (devuelve un 0 en caso contrario)

Ejemplo:

```
if (UART1_Data_Ready() == 1)
{
    aux = UART1_Read(); // se lee el dato recibido y se guarda en la variable aux
}
```

- *char* UARTx_Read(); devuelve el dato recibido

Ejemplo:

```
if (UART1_Data_Ready() == 1)
{
    aux = UART1_Read(); //se guarda el valor recibido en la variable aux
}
```

Nota: se puede transmitir el valor guardado por una *variable* y recuperarlo como tal

Ejemplo:

```
int aux = -12;
```

```
...
```

```
while(UARTx_Tx_Idle() == 0);
```

```
UART1_Writet(aux); //se envía el byte menos significativo de aux
```

```
UART1_Write(aux >> 8); //se envía el byte más significativo de aux
```


- Configuración de la interrupción RCIE en un PIC18F452 (comunicación asíncrona):

Cada vez que el módulo Uart recibe un byte de datos (y está disponible en el registro RCREG para ser leído) se puede crear una interrupción.

```
void interrupt() //rutina de servicio de interrupciones (MikroC)
{
    .....
    PIR1.RCIF = 0; // se borra el flag de la interrupción RCIE
}
```

```
void main()
{
    .....
    RCON.IPEN = 0; //se deshabilitan las interrupciones con prioridad (MikroC)
    PIR1.RCIF = 0; //se pone a cero el flag de la interrupción RCIE
    PIE1.RCIE = 1; // se habilita la interrupción RCIE
    INTCON.PEIE = 1; //es de tipo peripheral
    INTCON.GIE = 1; // se habilitan las interrupciones en general
    .....
}
```

- Terminal virtual (ISIS) :

_ El terminal virtual de ISIS permite ver los caracteres enviados por *Uart*. Para ello sólo hay que conectar el terminal TX del transmisor al terminal RXD del terminal virtual.

_ Hay que configurar el terminal con los datos de la transmisión a observar: *baud rate*, *data bits*, *parity*, *stop bits*, etc. (haz doble click sobre el terminal virtual para configurarlo)

Funciones de conversión a una cadena de caracteres	Tipos a los que se aplica	Tamaño array
void ByteToStr(unsigned short input, char *output);	(unsigned) char, unsigned short (int)	[4]
void ShortToStr(short input, char *output);	signed char, (signed) short (int)	[5]
void WordToStr(unsigned input, char *output);	unsigned (int)	[6]
void IntToStr(int input, char *output);	(signed) int	[7]
void LongToStr(long input, char *output);	(signed) long (int)	[12]
void LongWordToStr(unsigned long input, char *output);	unsigned long (int)	[11]
unsigned char FloatToStr(float fnum, unsigned char *str);	float	[14]

Tipo de variable	Tamaño	Rango de valores	Tamaño array
(unsigned) <i>char</i>	8 bits	[0, 255]	[4]
signed <i>char</i>	8 bits	[−128, +127]	[5]
unsigned short (<i>int</i>)	8 bits	[0, 255]	[4]
(signed) short (<i>int</i>)	8 bits	[−128, +127]	[5]
(signed) <i>int</i>	16 bits	[−32768, +32767]	[7]
unsigned (<i>int</i>)	16 bits	[0, 65535]	[6]
(signed) long (<i>int</i>)	32 bits	[−2147483648, +2147483647]	[12]
unsigned long (<i>int</i>)	32 bits	[0, 4294967295]	[11]
<i>float</i> , double, long double	32 bits	[−1.5·10 ^{−45} , +3.4·10 ³⁸]	[14]

Nota: en el array se tienen que poder guardar los caracteres correspondientes a cada dígito, el signo (si lo hay) y el carácter nulo ‘/0’ que indica el fin del array.

PIC18F452

