UART (*Univ*ersal 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.

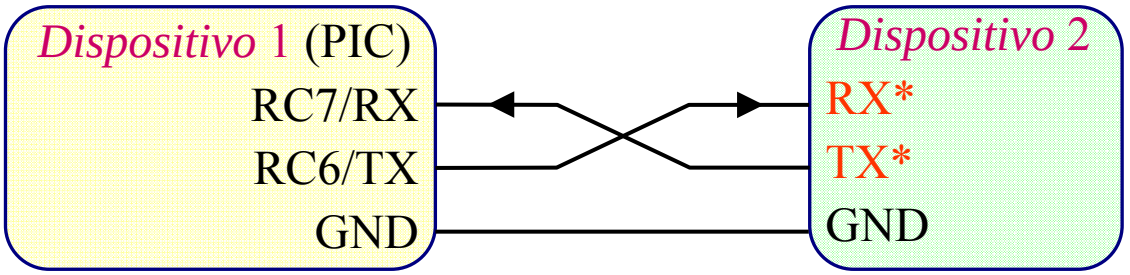
  Half duplex: transmisión en ambos sentidos, pero no de forma simultánea.

  Full duplex: transmisión en ambos sentidos de forma simultánea.

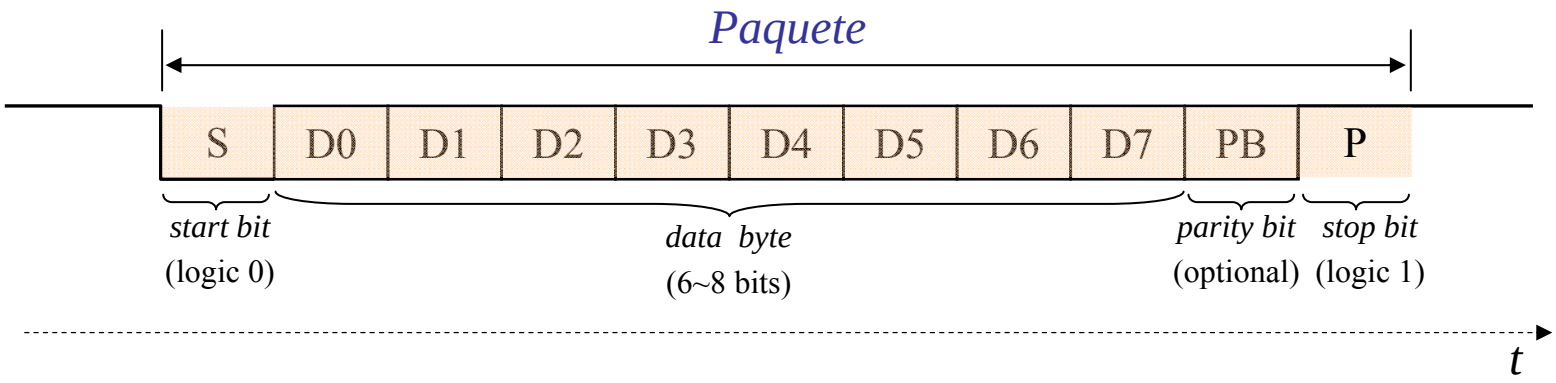
- Cualquier nodo puede iniciar la comunicación.

- La comunicación es asíncrona (≡ 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 *TX* está a nivel alto.

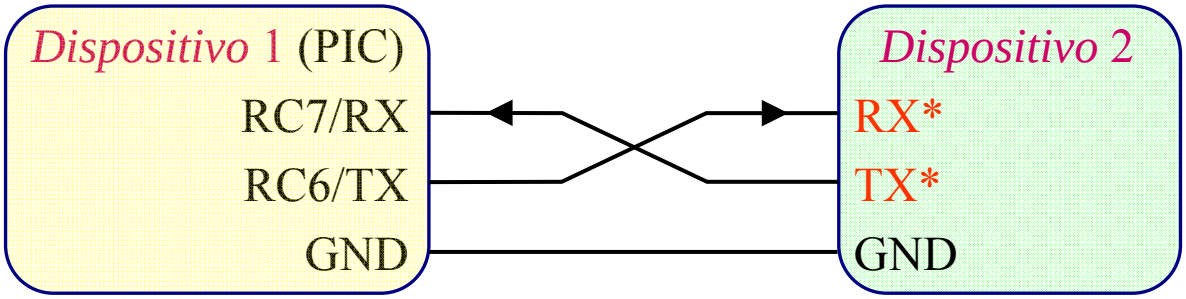
· Los niveles de tensión asociados a los niveles lógicos no están establecidos. En la práctica se utilizan diferentes normas para establecer dichos niveles de tensión, como son: RS-232, RS-422 y RS-485 (ver estándares EIA: *Electronic Industries Alliance*)

nomenclatura:

RX ≡ RXD

TX ≡ TXD

- Conexiones***:



- 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*

_ 1 bit de *stop*

_ sin bit de *paridad*

_ transmisión *asíncrona*

x: número del módulo *Uart* a utilizar

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_T*x*_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())
{
  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
```

Nota: si realizas dos escrituras consecutivas, por un retardo de 10mseg entre una escritura y otra o bien utiliza UARTx_T*x*_Idle();

- *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())
{
  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())
{
  aux = UART1_Read(); //se guarda el valor recibido en la variable aux
}
```

Importante: se puede transmitir el valor guardado por una *variable* de cualquier tipo y recuperarlo como tal

Ejemplo:

```
int aux = –12; // variable de 16 bits
…
while(UARTx_Tx_Idle() == 0);
UART1_Writet(aux); //se envía el byte menos significativo de aux
while(UARTx_Tx_Idle() == 0);  //delay_ms(10);
UART1_Write(aux >> 8); //se envía el byte más significativo de aux
```

Acceso a los bytes en memoria de un valor *float*: (1º método)

```
float valor = 1.5;
char *puntero; //puntero es una variable especial que guarda la dirección de una variable (normal) en memoria
unsigned short int aux1, aux2, aux3, aux4;

puntero = &valor; //puntero guarda la dirección en memoria de la variable valor

aux1 = *puntero;  //aux1 guarda el primer byte de los 4 que forman el valor guardado por valor
aux2 = *(puntero+1);  //aux2 guarda el segundo byte de los 4 que forman el valor guardado por valor
aux3 = *(puntero+2);  //aux3 guarda el tercer byte de los 4 que forman el valor guardado por valor
aux4 = *(puntero+3);  //aux4 guarda el cuarto byte de los 4 que forman el valor guardado por valor

// Nota en cada posición de memoria se guarda 1 byte
```

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

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

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

void main()
{
 …..
 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.

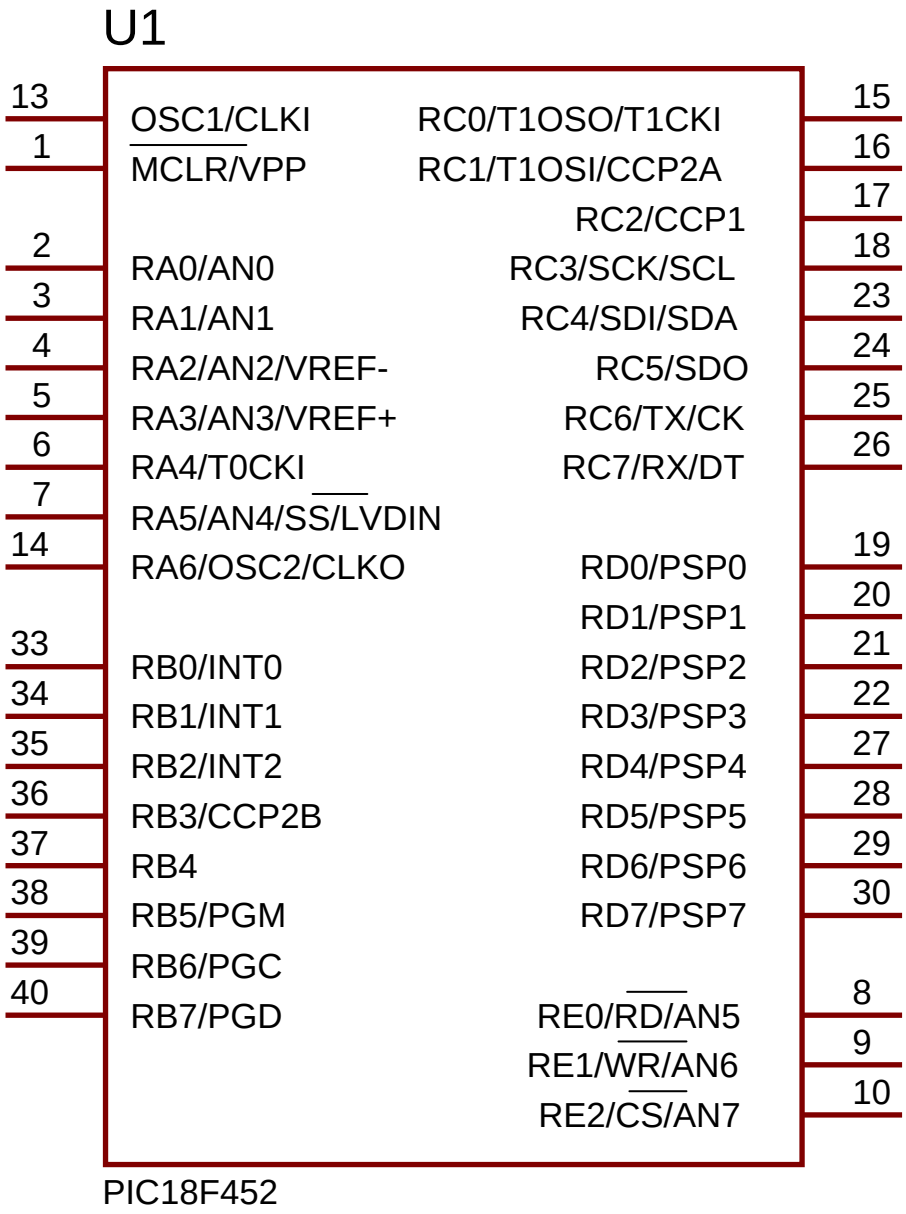
_ Hay que configurar el terminal con los datos de la transmisión a observar: *baud rate*, *data bits*, *parity*, *stop bits*, etc. (haz doble click sobre el terminal virtual para configurarlo)

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) ***char***	8 bits	[0, 255]	[4]
signed ***char***	8 bits	[–128, +127]	[5]
unsigned short (***int***)	8 bits	[0, 255]	[4]
(signed) short (***int***)	8 bits	[–128, +127]	[5]
(signed) ***int***	16 bits	[–32768, +32767]	[7]
unsigned (***int***)	16 bits	[0, 65535]	[6]
(signed) long (***int***)	32 bits	[–2147483648, +2147483647]	[12]
unsigned long (***int***)	32 bits	[0, 4294967295]	[11]
float, double, long double	32 bits	$[-1.5 \cdot 10^{-45}, +3.4 \cdot 10^{38}]$	[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.

*PIC*18*F*452


U1
13
1
2
3
4
5
6
7
14
33
34
35
36
37
38
39
40
OSC1/CLKI
MCLR/VPP
RA0/AN0
RA1/AN1
RA2/AN2/VREF-
RA3/AN3/VREF+
RA4/T0CKI
RA5/AN4/SS/LVDIN
RA6/OSC2/CLKO
RB0/INT0
RB1/INT1
RB2/INT2
RB3/CCP2B
RB4
RB5/PGM
RB6/PGC
RB7/PGD
RC0/T1OSO/T1CKI
RC1/T1OSI/CCP2A
RC2/CCP1
RC3/SCK/SCL
RC4/SDI/SDA
RC5/SDO
RC6/TX/CK
RC7/RX/DT
RD0/PSP0
RD1/PSP1
RD2/PSP2
RD3/PSP3
RD4/PSP4
RD5/PSP5
RD6/PSP6
RD7/PSP7
RE0/RD/AN5
RE1/WR/AN6
RE2/CS/AN7
8
9
10
15
16
17
18
23
24
25
26
19
20
21
22
27
28
29
30
PIC18F452