MCBtec

Mas información en
www.mcbtec.com

CONEXIÓN DEL PC A MICROCONTROLADOR POR RS232

INDICE:

1. Introducción.

El objetivo de este texto es facilitar la información básica para conectar un PC con un microcontrolador. Estudiaremos la conexión física entre el PC y el microcontrolador, el software del microcontrolador (PIC16F628 y COP8CCR) y el software del PC utilizando el API32.

2. Estructura física y características básicas de la RS232.

El objetivo de este texto no es hacer un estudio en profundidad de las especificaciones técnicas del estándar RS23, por lo que nos limitaremos a describir solo las características técnicas basicas que necesitamos para la conexión.

Desde el punto de vista de la RS232 existen dos tipos de dispositivos:

- DTE (Data terminal equipment): El PC que controla la transmisión.
- DCE (Data communications equipment): El modem, impresora, etc…

Las especificaciones máximas eléctricas para la transmisión de datos del EIA para la RS232C son:

- Lógica 0: de +3v a +25v.
- Lógica 1: de –3v a –25v.
- De –3v a +3v es indefinida.

- En circuito abierto, la tensión no puede exceder de los 25v.
- En cortocircuito la corriente no puede superar los 500 mA.

La transmisión de datos vía RS232C es asíncrona, es decir, la señal de reloj no se transmite, por lo que tanto el transmisor como el receptor tienen que funcionar a la misma velocidad (por ejemplo 9600 Kb/s). Los bytes transmitidos tienen el siguiente formato:

Formato de transmisión de un byte por RS232

El bit de Start sincroniza los dos dispositivos y el bit de stop finaliza la transmisión.

La velocidad de transmisión se da en baudios. Realmente este termino indica la cantidad de cambios que hay en la línea, pero para nosotros va a ser lo mismo que bits/s, las velocidades más habituales son 38400, 19200, 9600, 7200, 4800, 3600, 2400, 1800, 1200, 600, 300, 150.

Existe una limitación física de la longitud del cable, según la norma RS232, a 15 metros.

3. Conexión física y flujo de datos.

El conector utilizado en el puerto RS232 es el llamado SUB-D y en el PC lo podemos encontrar en dos tamaños; de 9 pines (el más habitual) y de 25 pines. La descripción de los pines es la siguiente:

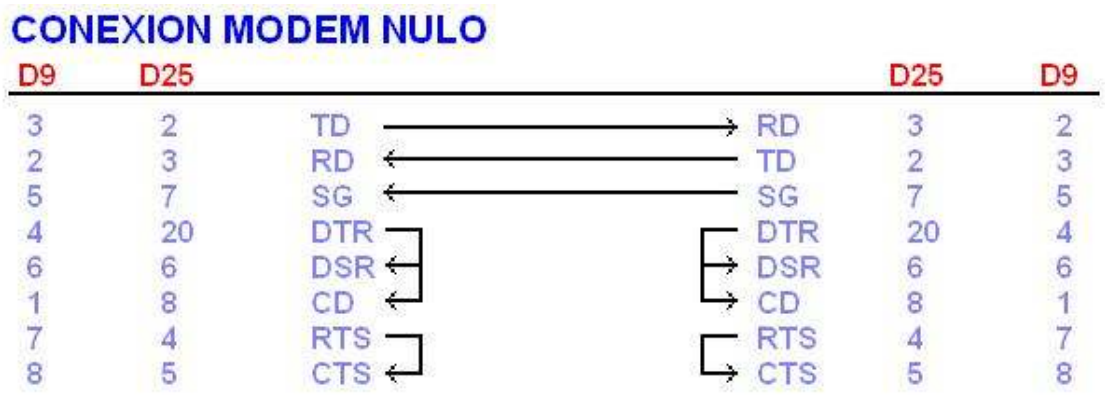
Name	Sigla	9 Pines	25 Pines
Transmit Data	TD	Pin 3	Pin 2
Receive Data	RD	Pin 2	Pin 3
Request To Send	RTS	Pin 7	Pin 4
Clear To Send	CTS	Pin 8	Pin 5
Data Set Ready	DSR	Pin 6	Pin 6
Signal Ground	SG	Pin 5	Pin 7
Carrier Detect	CD	Pin 1	Pin 8
Data Terminal Ready	DTR	Pin 4	Pin 20

Función de los Pines:

Sigla	Función
TD	Salida Data Serie (TXD)
RD	Entrada Data Serie (RXD)
CD	Cuando el MODEM (DCE) detecta una portadora desde el modem al otro lado del teléfono, esta línea se activa para avisar al DTE(PC).

DSR	Similar a CTS, pero no se suele usar para protocolo.
DTR	Similar a RTS, pero no se suele usar para protocolo.
CTS	El MODEM pone esta línea en mark (1), para indicar al PC esta preparado para recibir datos, y en space (0) si el buffer esta lleno.
RTS	El PC pone esta línea en mark (1) para indicar que esta preparado para recibir los datos del MODEM, en espace (0) si el buffer esta lleno.
RI	Se activa cuando se detecta señal de llamada el PSTN.

El tipo de conexión RS232 más sencilla es la de **MODEM NULO**, la cual solo utiliza 3 líneas (RD, TD y SG). El siguiente diagrama muestra una conexión de este tipo que nos va a permitir conectar el PC a los microcontroladores (el PC y el microcontrolador son de tipo DTE:



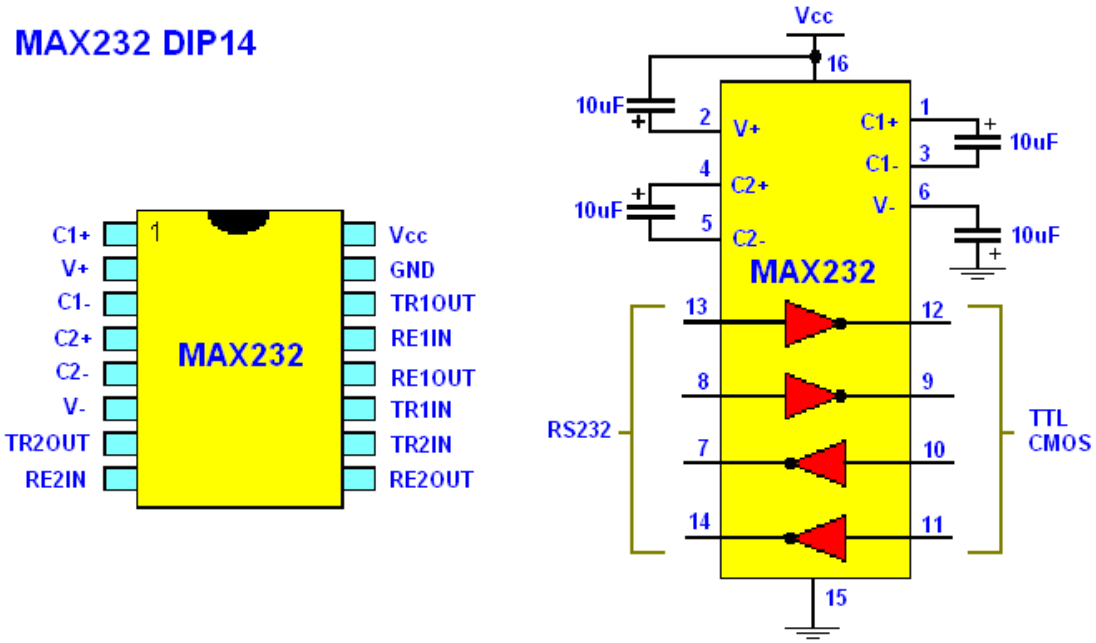
Si queremos hacer pruebas con el puerto serie podemos hacer un puente entre las líneas 2 y 3 para que lo que salga por el puerto vuelva a entrar en el mismo puerto.

Si queremos activar el control de flujo de datos mediante hardware, uniremos RTS del PC con CTS del microcontrolador y CTS del PC con RTS del microcontrolador.

Si alguno de los dispositivos DCE o DTE, tiene una velocidad de transmisión/ recepción muy diferente al otro, es posible que se pierdan datos. Para evitar esto existen dos formas de controlar el flujo de datos:

- **Por software (control Xon/Xoff):** Esta utiliza dos caracteres Xon (ASCII 17) y Xoff (ASCII 19). Cuando el microcontrolador tiene el buffer lleno, envía Xoff para que el PC pare de transmitir y luego envía Xon para reactivar la transmisión. En la configuración del puerto serie del PC existe una opción para activarlo. En el microcontrolador habrá que implementarlo en el software.
- **Por Hardware (control RTS/CTS):** Utiliza las líneas RTS y CTS para controlar el flujo de datos. Cuando el PC quiere transmitir datos activa RTS, si el microcontrolador esta preparado para recibir activa CTS, si el buffer esta lleno no lo activa. Esta forma no funciona en conexión MODEM NULO.

Como hemos visto anteriormente las especificaciones eléctricas del puerto serie tiene unos niveles de tensión para el 1 lógico (-3v a –25v) y el 0 lógico (+3v a +25v) muy distintas de los niveles utilizados por los microcontroladores (5v y 0v), para poder interconectar el PC con el microcontrolador tenemos que utilizar un driver que adapte estos dos niveles lógicos. Un driver muy utilizado es el MAX232 (datasheet en www.maxim-ic.com), que es capaz de generar las tensiones necesarias para los 0 y 1 lógicos (+10v y -10v) a partir de los 5v de alimentación y, además, tiene dos buffers de entrada y dos de salida, lo que permite el control total del puerto (además de TD y RD podemos conectar RTS y CTS). El pin out y el esquema de utilización básico, es el siguiente:

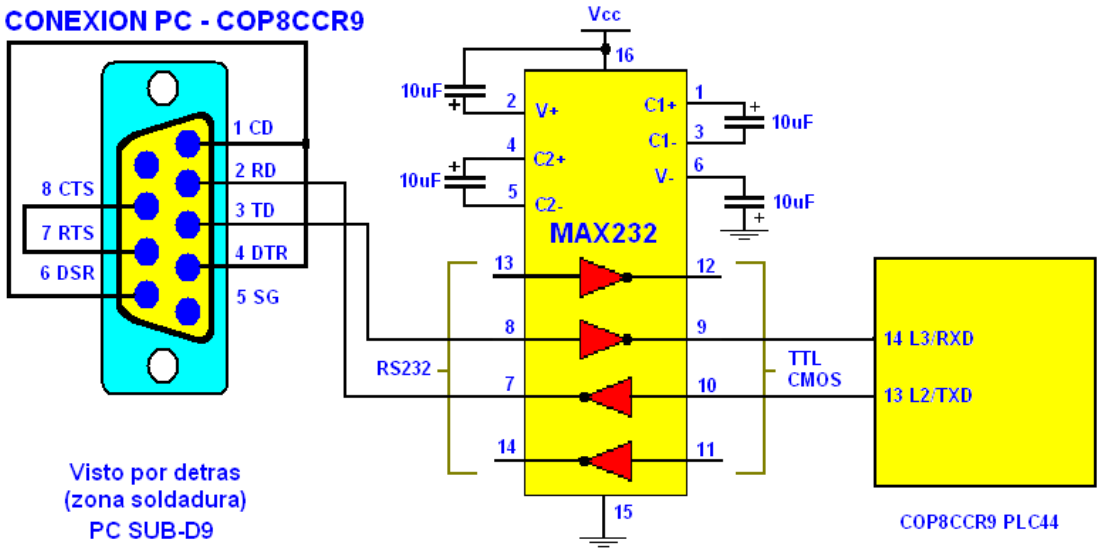


4. Control RS232 por COP8CCR.

Para conectar el PC al COP8CCR9 utilizo un cable de 2 hilos más malla, los colores de los hilos con blanco y marrón y los conecto según la tabla siguiente en el SUB-D de 9 pines hembra que luego conectaremos al PC y a los pines del COP8CCR:

Pin Sub-D 9 PC	Color Cable	Pin COP8CCR
2 (RD)	Marrón	13 (L2/TXD)
3 (TD)	Blanco	14 (L3/RXD)
5 (SG)	Malla	GND
4(DTR)+ 6 (DSR) + 1(CD) PUENTEADOS		
7(RTS)+ 8(CTS) PUENTEADOS		

El esquema de las conexiones es el siguiente:



En el COP8C se utilizan tres líneas para la gestión del USART, estas líneas son:

- **L2/TDX:** Se utiliza para salida de datos del COP8CCR. Hay que configurarla como salida y poner a 1 ETDX de ENUI para que funcione.
- **L3/RDX:** Se utiliza para leer datos. No hay que configurar nada para que funcione.
- **L1/CKX:** Si usamos el modo síncrono y la configuramos como entrada, por aquí introduciremos la frecuencia en baudios para la transmisión. Si esta configurada como salida, sacara la frecuencia en baudios seleccionada.

Aunque puede funcionar en modo síncrono (utilizando CKX) nos centraremos en el modo asíncrono. Veamos el proceso de trasmisión y recepción:

- **Transmisión:** Para transmitir hay dos registros; TSFT, que es el registro de desplazamiento, y TBUF que es el buffer de transmisión. Inicialmente el buffer esta vacío (TBMT de ENU=1), al cargar TBUF con el byte a transmitir TBMT se pone a 0 (indicando que el buffer esta lleno) y el dato almacenado en TBUF es transferido a TSFT para su transmisión vaciándose TBUF y poniéndose a 1 TBMT.
- **Recepción:** Para recibir hay dos registros; RSFT, que es el registro de desplazamiento, y RBUF que es el buffer de recepción. Los bits llegan por L3/RDX y se van introduciendo en RSFT, cuando un dato se ha completado, este se carga en RBUF y se pone a 1 RBFL de ENU, este se pondrá a cero cuando se lea RBUF. Si ocurre un error se detectará por ERR del registro ENU o RCVG del registro ENUR, para borrar el error leemos el registro ENUR.
- **Interrupciones:** Se pueden activar interrupciones tanto en la recepción como en la transmisión. Veámoslas:
    - **Recepción:** Esta se activa poniendo a 1 ERI del registro ENUI. Cada vez que se llene el buffer de recepción RBFL=1, se activará la interrupción. Para detectarla haremos IFBIT RBFL,ENU y para desactivarla haremos X A, RBUF.
    - **Transmisión:** Esta se activa poniendo a 1 ETI del registro ENUI. Cada vez que se vacíe el buffer de transmisión TBMT=1, se activará la

interrupción. Para detectarla haremos IFBIT TBMT,ENU y para desactivarla haremos LD TBUF,#dato.

Para calcular los baudios de transmisión el sistema dispone de una preescala y de un divisor de 11 bits. El calculo de los valores de N (divisor de 11 bits) y P (preescala), se calculan con la siguiente formula:

$$BR=(Fc*2)/(16*N*P)$$

Por ejemplo, para una frecuencia de Xtal (Fc) de 10 MHz, queremos obtener BR=9600. Primero despejamos la formula anterior y la transformamos en:

$$N*P=(Fc*2)/(16*BR)=(10000000*2)/(16*9600)=130,208$$

Como N tiene que ser un entero para que la frecuencia no tenga error, intentamos buscar un P tal que 130,208/P sea lo más parecido a un entero. Si cogemos P=13 tenemos N=10,016, que es casi un entero. En valor a meter en el registro divisor en N-1 es decir, para el caso, meteremos 9.

Preescala	Factor	Preescala	Factor	Preescala	Factor
00000	NOCLOCK	01011	6	10110	11,5
00001	1	01100	6,5	10111	12
00010	1,5	01101	7	11000	12,5
00011	2	01110	7,5	11001	13
00100	2,5	01111	8	11010	13,5
00101	3	10000	8,5	11011	14
00110	3,5	10001	9	11100	14,5
00111	4	10010	9,5	11101	15
01000	4,5	10011	10	11110	15,5
01001	5	10100	10,5	11111	16
01010	5,5	10101	11		

El COP8CCR tiene 2 registros para guardar esta información. Estos son PSR y BAUD, en PSR se introducen en la parte alta D7..D3 el valor de la Preecala, y en D2..D0 los bits D10..D8 del divisor. En el registro BAUD, se introduce D7..D0 del divisor.

A continuación se describen una serie de rutinas para controlar el puerto serie, suponiendo que el USART del PC esta configurado como 9600, 8 bits dato, no paridad y 1 bits de stop.

```
; Instrucciones para la inicialización del puerto serie
INICIO:     LD PORTLC, #XXXXX01XX     ; L3= Entrada, L2= Salida
            LD BAUD, #009             ;9600b a 10MHz
            LD ENU, #000              ;No paridad, 8 bits.
            LD ENUR, #000             ; ATTN= 0
            LD ENUI, #001000RT        ;TDX Activo e int Re y Tras
            LD PSR, #0C8
```

```
;Transmite #dato por el puerto serie.
TRANS:    IFBIT TBMT,ENU          ;TBMT=1 buffer vacio
          LD TBUF,#dato
          RET

;Almacena en A el valor del byte ( si lo hubiera ) recibido por el puerto serie
RECI:     IFBIT RBFL, ENU         ;RBFL=1 buffer lleno
          JP LEEDAT
          RET
LEEDAT:   IFBIT ERR, ENU          ;ERR=1 hay error
          JP ERROR
          LD A, RBUF
          RET
ERROR:    LD A, ENUR              ;Borra el error
          RET
```

Lo lógico es transmitir cuando haga falta (sin interrupción) y activar la interrupción para la recepción. La ruina para la recepción mediante interrupción seria esta:

```
          .=0FF
          ...                     ;Otras interrupciones
          IFBIT RBFL, ENU         ;Int Recepción
          JP RECIBE
               ...                ;Otras interrupciones
RECIBE:   PUSH A                  ;DATRS Contiene dato.
          IFBIT ERR, ENU          ;ERR=1 hay error
          JP ERROR
          LD A, RBUF
          X A, DATRS
          JP FINRECI
ERROR:    LD A, ENUR              ;Borra el error
          LD A, RBUF
FINRECI:  RBIT ERI, ENUI
          POP A
          RETI
```

4. Control RS232 por PIC16F628.

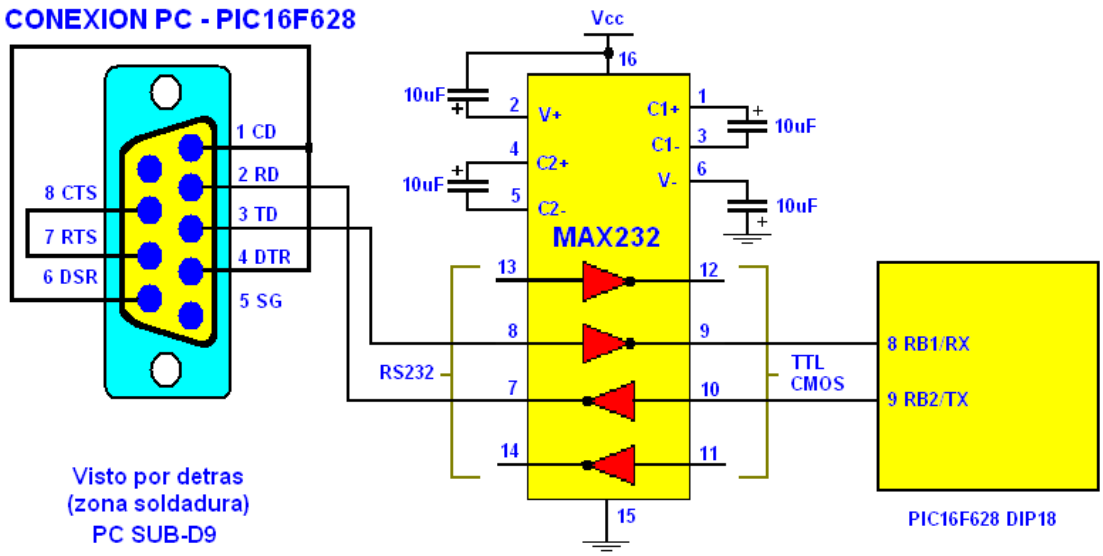
El funcionamiento del puerto serie en todos los PIC de la serie 16F es muy similar, por lo que todo lo escrito a continuación para el PIC16F628, sirve para cualquier otro PIC16FXXX.

Para conectar el PC al PIC16F628 utilizo un cable de 2 hilos más malla, los colores de los hilos son blanco y marrón y los conecto según la tabla siguiente en el SUB-D de 9 pines hembra que luego conectaremos al PC y a los pines del PIC16F628:

Pin Sub-D 9 PC	Color Cable	Pin PIC
2 (RD)	Marrón	9 (RB2/TX)

3 (TD)	Blanco	8 (RB1/RX)
5 (SG)	Malla	GND
4(DTR)+ 6 (DSR) + 1(CD) PUENTEADOS		
7(RTS)+ 8(CTS) PUENTEADOS		

El esquema de las conexiones es el siguiente:



En el PIC16F628 se utilizan dos líneas para la gestión del USART. Para que estas líneas se conviertan en RX y TX, ponemos el bit SPEN (RCSTA<7>) a 1. estas líneas son:

- **RB2/TX:** Se utiliza para salida de datos del PIC. Hay que configurarla como entrada TRISB<2>=1. Para Permitir la transmisión hay que poner a 1 TXEN (TXSTA<5>).
- **RB1/RX:** Se utiliza para leer datos. Hay que configurarla como entrada TRISB<1>=1. Para permitir la recepción hay que poner a 1 CREN (RCSTA<4>).

Aunque puede funcionar en modo síncrono nos centraremos en el modo asíncrono. Veamos el proceso de trasmisión y recepción:

- **Transmisión:** Para transmitir hay dos registros; TSR, que es el registro de desplazamiento, y TXREG que es el buffer de transmisión. Inicialmente el buffer esta vacío (TXIF de PIR1<4>=1), al cargar TXREG con el bit a transmitir TXIF se pone a 0 (indicando que el buffer esta lleno) y el dato almacenado en TXREG es transferido a TSR para su transmisión vaciándose TXREG y poniéndose a 1 TXIF.
- **Recepción:** Para recibir hay dos registros; RSR, que es el registro de desplazamiento, y RCREG que es el buffer de recepción. Los bits llegan por RB1/RX y se van introduciendo en RSR, cuando un dato se ha completado, este se carga en RCREG y se pone a 1 RCIF de PIR1<5>, este de pondrá a cero cuando se lea RCREG.

- **Interrupciones:** Se pueden activar interrupciones tanto en la recepción como en la transmisión. Veámoslas:
  - **Recepción:** Esta se activa poniendo a 1 RCIE del registro PIE1<5>. Para detectarla miraremos PIR1,RCIF y para desactivarla leeremos RCREG.
  - **Transmisión:** Esta se activa poniendo a 1 TXIE del registro PIE1<4>. Para detectarla miraremos PIR1,TXIF y para desactivarla escribiremos en TXREG.

Para calcular los baudios de transmisión el sistema dispone de un registro de preescala SPBRG. El calculo del valor de este registro se hace teniendo en cuenta el valor de del bit BRGH(TXSTA<2>). Las formulas son las siguientes:

BRGH=0	BRGH=1
BD=Fosc/(64(X+1))	BD=Fosc/(16(X+1))

Por ejemplo, para una frecuencia de Xtal (Fosc) de 4 MHz, queremos obtener BR=9600. Primero pondremos BRGH=1, despejamos la formula anterior y la transformamos en:

X=(Fosc/16*BD)-1=(4000000/16*9600)-1=25,04

Como N tiene que ser un entero haremos X=25. El valor de SPBGR=25=19H.

Una vez visto esto pasemos a realizar algunas rutinas, suponiendo que el USART del PC esta configurado como 9600, 8 bits dato, no paridad y 1 bits de stop.

```
;Configuración del puerto serie.
Inicio:     Banco1          ;Selecciona Banco1
            movlw 0x06      ;Pone RX,TX como entradas
            iorwf TRISB
            movlw 0x19      ;Pone La velocidad a 9615 (25 decimal)
            movwf SPBRG
            movlw 0x24      ;Activa TXEN y BRGH (high baud rate)
            movwf TXSTA
            bsf PIE1,RCIE   ;Habilita interrupción de recepción.
            Banco0          ;select bank 0
            movlw 0x90      ;Habilita Recepcion de bytes
            movwf RCSTA
            .....
;Recibe dato por puerto serie y lo almacena en DatoRed
ReciRS232:  bcf ControlRed,0
            btfsc RCSTA,OERR  ;Si hay error overrun
            goto ErrRSOverr
            btfsc RCSTA,FERR  ;Si hay error de framing
            goto ErrRSFrame
            movf RCREG,w      ;Coge dato
            movwf DatoRed     ;Introduce dato en buffer
            bsf ControlRed,0  ;hay dato correcto
```

```
FinReciRS:    return
ErrRSOverr:   bcf RCSTA,CREN    ;Inicializa
              bsf RCSTA,CREN    ;Activa otra vez recepción
              return
ErrRSFrame:   movf RCREG,W      ;Elimina el dato erróneo
              return

;Transmite el dato almacenado en DirRed.
TransRS232:   Banco0
BucleTrans:   btfss PIR1,TXIF   ;Mira si se puede transmitir
              goto BucleTrans
              movf DirRed,w     ;Envia dato en DirRed.
              movwf TXREG
              return
```

6. WIN API32 para controlar RS232.

Desde el punto de vista del PC el control del puerto serie lo realizamos utilizando las funciones y estructuras que el WIN API32 tiene implementadas para realizar esta función.

Desde el punto de vista del PC el puerto serie es un fichero y por tanto, para utilizarlo, usaremos las mismas funciones que se utilizan para manejar ficheros. Básicamente para comunicar un microcontrolador con el PC necesitamos realizar 4 funciones:

Abrir Puerto: Para estudiar el proceso de apertura de un puerto vamos a analizar la siguiente función en C++ que realiza esta función:

```
// Control puerto serie C++
// Variables globales

DCB DCBdata;              // Contiene la información del puerto serie
HANDLE ComHandle;         // Manipular del puerto

 // Status =  0  No hay error.
 // Status = -1  No existe el puerto.
 // Status = -5  Error leer o configuración.
 // Status = -6  Error abrir el puerto serie.

int AbrirPuertoSerie(char *Puerto, char *Configuracion)
{
 int Status;
 COMMTIMEOUTS time_out;

Status=0;
 ComHandle=CreateFile(Puerto,GENERIC_READ | GENERIC_WRITE ,0,NULL,
OPEN_EXISTING, 0, 0);
```

```
if (ComHandle==INVALID_HANDLE_VALUE) {CloseHandle(ComHandle);Status= -
1;}
 else
  if (GetCommState(ComHandle,&DCBdata)==true)
    {
     BuildCommDCB(Configuracion,&DCBdata);
     if (SetCommState(ComHandle,&DCBdata)==true)
      {
       time_out.ReadIntervalTimeout = 20;
       time_out.ReadTotalTimeoutMultiplier = 2;
       time_out.ReadTotalTimeoutConstant = 50;
       time_out.WriteTotalTimeoutMultiplier = 2;
       time_out.WriteTotalTimeoutConstant = 50;
       if (SetCommTimeouts(ComHandle, &time_out)==true)  Status=0; else Status=-6;
      }
     else Status = -5;
    }
   else Status = -4;
 return Status;
}
```

La función *CreateFile* abre un fichero o un puerto de comunicaciones, esta tiene varios parámetros (si te interesa más información sobre esta función, puede visitar www.winapi.conclase.net -> Curso de API32 -> Barra de Menú "Funciones"), para nuestro caso los parámetros que nos interesan son:

- *Puerto:* Cadena de caracteres en la que indicaremos el nombre del puerto que queremos abrir, por ejemplo "COM1".
- *GENERIC_READ | GENERIC_WRITE*: Aquí indicamos si queremos leer y/o escribir. En este caso queremos lee y escribir.
- *OPEN_EXISTING*: Si el puerto no existe la función falla.

La función nos devolverá un manipulador para el puerto serie. En el caso de que la función fallara, por ejemplo, si el puerto no existe, la función devuelve el valor *INVALID_HANDLE_VALUE,* que mediante un if podremos tratar.

Si hemos conseguido abrir el puerto, lo primero que haremos será, mediante la función GetCommState, cargar en una variable de tipo DCB, la información con las características por defecto de ese puerto, luego con la función BuildCommDCB, cambiamos los valores almacenados en la estructura de datos *DCBdata*, para ello utilizamos la cadena de caracteres *Configuración* cuyo formato es el siguiente "b,p,d,s", en donde:

- b: Son los baudios a los que queremos que trabaje el puerto (110, 150, 300, 600, 1200, 2400, 4800, 9600, etc...)
- p: La paridad (n=ninguna, o = odd, e = event, m = mak y s = space).
- d: Es el tamaño del dato en bits (8,7).
- s: Número de bits de stop (1,1.5 o 2).

Por ejemplo, si queremos configurar nuestro puerto serie a 9600 baudios, sin paridad, con 8 bits y 1 bit de stop haremos *configuracion*=”9600,n,8,1”. La función SetCommState actualiza la configuración del puerto con la estructura *DCBdata* modificada. Si hasta aquí todo es correcto entonces pasaremos a poner los timeout de la llamadas al puerto. Estos timeout están almacenados en una estructura de tipo *COMMTIMEOUTS* que mediante la función *SetCommTimeouts* se actualiza para el puerto, los timeouts almacenan los tiempos que queremos que el puerto este a la espera de un dato o a la espera de que salga un dato antes de indicar que no hay datos o que hay un error en la transmisión.

Si no ha habido errores en el proceso la función devolverá 0 y ya dispondremos de nuestro puerto serie.

En pascal esta función queda como:

```
// Pascal Delphi.

// Status =  0  No hay error
 // Status = -1  No existe el puerto
 // Status = -5  Error leyendo configuracion
 // Status = -6  Error abriendo el puerto serie

Function AbrirPuertoSerie(Puerto,Configuracion:String):Integer;
var
  status:Integer;
  time_out:COMMTIMEOUTS;
begin
  status:=0;
  //FILE_FLAG_OVERLAPPED
  serial_handle:=CreateFile(PChar(Puerto),GENERIC_READ OR GENERIC_WRITE
,0, 0, OPEN_EXISTING, 0, 0);

If (serial_handle=INVALID_HANDLE_VALUE) then
 begin closehandle(serial_handle);status := -1;end
else
   If GetCommState(serial_handle,DCBdata) then
     begin
      BuildCommDCB(PChar(Configuracion),DCBdata);
      if SetCommState(serial_handle,DCBdata) then
       begin
        time_out.readintervaltimeout := 20;
        time_out.readtotaltimeoutmultiplier := 2;
        time_out.readtotaltimeoutconstant := 50;
        time_out.writetotaltimeoutmultiplier := 2;
        time_out.writetotaltimeoutconstant := 50;
        If SetCommTimeouts(serial_handle, time_out)  then status := 0
                                        else status := -6;
       end
      else status := -5;
```

```
    end
   else status := -4;
 AbrirPuertoSerie:=Status;
end;
```

Cerrar Puerto: Para cerrar puerto utilizamos la función CloseHandle que elimina el manipulador del puerto serie. El código para C y pascal seria:

```
// Pascal Delphi
Function CerrarPuertoSerie:Boolean;
begin
  CerrarPuertoSerie:=CloseHandle(serial_handle);
end;

// c++
bool CerrarPuertoSerie()
{
  return CloseHandle(ComHandle);
}
```

Leer Puerto: La función ReadFile realiza una operación de lectura. A continuación tenemos un código en Pascal (delphi) y C++ que realiza una operación de lectura del puerto.

```
// Pascal Delphi
Function LeerPuertoSerie(cantidad:Integer;var Buffer:String):Integer;
var
  LBread:Boolean;
  leido:DWord;
begin
  result:=0;cadena:='';
  LBread:=ReadFile(ComHandle, Buffer, cantidad,leido,nil);
  If Not(LBRead) then result:=-1;
  If (leido=0) then result:=-2;
end;

// c++
int LeerPuertoSerie(int NumBytes, char *Buffer)
{
  bool LBread;
  DWORD leido;
  bool Resultado;

  Resultado=0;
  Buffer="";
  LBread=ReadFile(ComHandle, Buffer, NumBytes,&leido,NULL);
  if (LBread==0) Resultado=-1;
  if (leido==0) Resultado=-2;
  return Resultado;
}
```

Escribir Puerto: Transmitir datos por el puerto serie es muy similar a leerlos, para esto utilizamos la función WriteFile. Vemos un ejemplo de uso.

```
// Pascal Delphi
Function EscribirPuertoSerie(Buffer:String):Boolean;
var
  llongitud:Integer;
  lenviado:DWord;
  LBWrite:Boolean;
begin
  longitud := Length(Buffer);
  result := WriteFile(ComHandle, Buffer, longitud, enviado, nil);
  If (longitud <> lenviado) Then result:=False;
end;

// c++
bool EscribirPuertoSerie(char *Buffer)
{
  DWORD longitud;
  DWORD enviado;
  bool Resultado;

  longitud = strlen(Buffer);
  Resultado=WriteFile(ComHandle, Buffer, longitud, &enviado,NULL);
  if (longitud != enviado) Resultado=false;
  return Resultado;
}
```

El uso de estas funciones en el código del programa es el siguiente:

- Al iniciar el programa, si por ejemplo, queremos abrir el "COM1" a 9600 baudios, sin paridad, con 8 bits de dato y 1 bit de stop, pondremos la siguiente línea de código *AbrirPuertoSerie("COM1","9600,n,8,1").*
- Si queremos enviar "Hola mundo" por el puerto serie utilizaremos la siguiente instrucción *EscribirPuertoSerie("Hola mundo").*
- Si queremos leer 1 char del puerto serie haremos *LeerPuertoSerie(1, Buffer).*
- Antes de salir del programa haremos *CerrarPuertoSerie().*

7. Conclusiones.

En este texto hemos estudiado la conexión básica entre un PC y un microcontrolador. Si quieres ampliar conocimientos en www.mcbtec.com encontraras algunos links de utilidad.