LO STANDARD RS232

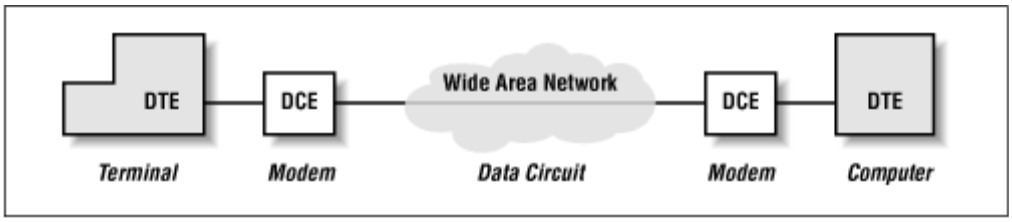
Lo standard RS232 definisce una serie di specifiche per la trasmissione seriale di dati tra due dispositivi denominati **DTE** (Data Terminal Equipment) e **DCE** (Data Communication Equipment).

Il Data Communication Equipment è un dispositivo che si occupa di gestire una comunicazione dati mentre il Data Terminal Equipment è un dispositivo che si occupa di generare o ricevere dati.

In pratica l'RS232 è stata creata per connettere tra loro un terminale dati (nel nostro caso un computer) con un modem, ma successivamente questo tipo di trasmissione è stata utilizzata per connettere altri dispositivi.

Per avere una connessione tra due computer è quindi necessario disporre di quattro dispositivi come visibile in figura: un computer (DTE) collegato al suo modem (DCE) ed un altro modem (DCE) collegato al suo computer (DTE). In questo modo qualsiasi dato generato dal primo computer e trasmesso tramite RS232 al relativo modem verrà trasmesso da questo al modem remoto che a sua volta provvederà ad inviarlo al suo computer tramite RS232. Lo stesso vale per il

percorso a ritroso.



Per usare la RS232 per collegare tra loro due computer vicini senza interporre tra loro alcun modem dobbiamo simulare in qualche modo le connessioni intermedie realizzando un cavo **NULL MODEM** o cavo invertente, ovvero un cavo in grado di far scambiare direttamente tra loro i segnali dai due DTE come se tra loro ci fossero effettivamente i DCE.

Per connettere il PC al nostro circuito simuleremo invece direttamente un DCE facendo credere al PC di essere collegato ad un modem. Prima di fare questo diamo uno sguardo in dettaglio al principio di funzionamento di una comunicazione seriale.

La comunicazione seriale asincrona

Per consentire la trasmissione di dati tra il PC ed il modem, lo standard RS232 definisce una serie di specifiche elettriche e meccaniche. Una di queste riguarda il tipo di comunicazione seriale che si vuole implementare che può essere sincrona o asincrona.

Nel nostro caso analizzeremo solo la comunicazione seriale asincrona ignorando completamente quella sincrona in quanto più complessa e non disponibile sui normali PC.

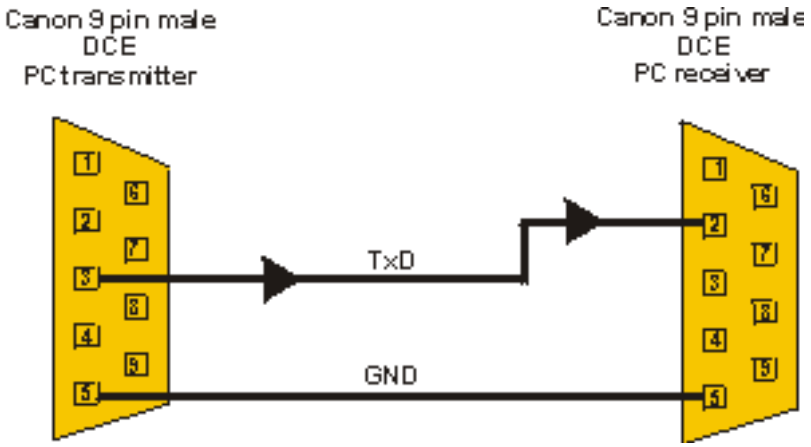
Una comunicazione seriale consiste in genere nella trasmissione e ricezione di dati da un punto ad un altro usando una sola linea elettrica. In pratica se desideriamo trasmettere un intero byte dobbiamo prendere ogni singolo bit in esso contenuto ed inviarlo in sequenza sulla stessa linea elettrica, un pò come avviene per la trasmissione in codice morse. La differenza sostanziale stà nel fatto che a generare e ricevere dati non c'é il telegrafista ma un computer per cui le velocità

di trasmissione raggiungibili sono molto superiori.

Facciamo subito un esempio pratico e vediamo come fa un PC a trasmettere, ad esempio, il carattere **A** usando la RS232.

Non è necessario ovviamente realizzare gli esempi riportati di seguito in quanto presuppongono l'uso di una coppia di PC ed un oscilloscopio non sempre disponibili nei nostri mini-laboratori da hobbysta. Per comprendere il funzionamento di quanto esposto è sufficiente fare riferimento alle figure a corredo.

Se prendiamo una coppia di fili e colleghiamo tra loro le porte seriali di due PC (che denomineremo PC trasmittente e PC ricevente) secondo lo schema riportato in figura, otterremo la più semplice delle connessioni in RS232:



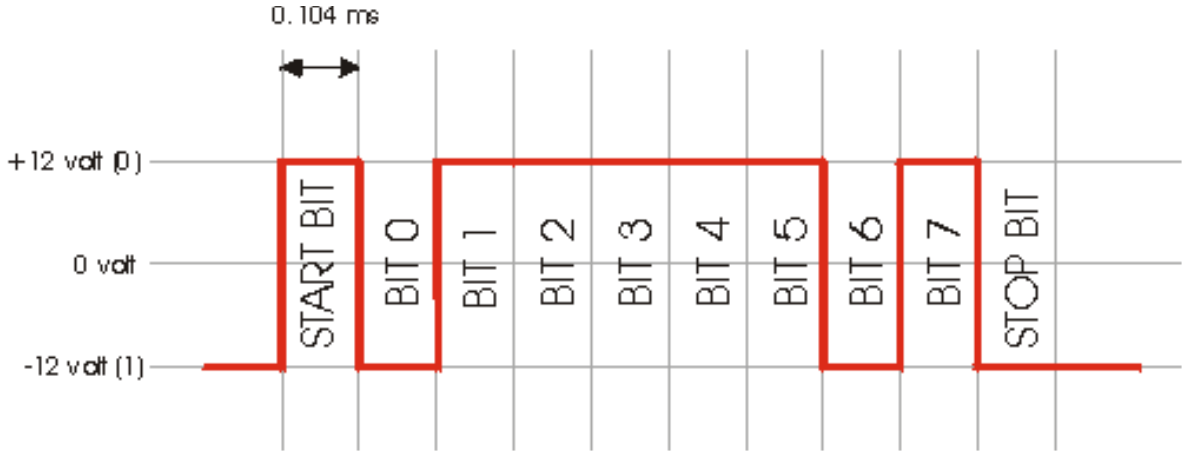
La linea **Transmit Data** (TXD) presente sul pin 3 del connettore DB9 maschio di cui il vostro PC è dotato è connessa alla linea **Receive Data** (RXD) presente sul pin 2 del secondo PC. Le masse (GND) presenti sul pin 5 di entrambe i PC sono connesse tra loro.

Quando non c'é nessuna trasmissione in corso la tensione sulla linea TXD è di -12 volt corrispondente alla condizione logica 1. Per indicare al PC ricevente che la trasmissione ha inizio, il PC trasmittente porta a +12 volt la linea TXD per un tempo pari all'inverso della frequenza di trasmissione ovvero al tempo di trasmissione di un singolo bit.

Nel nostro caso, avendo scelto di trasmettere a 9600 bit per secondo, la tensione di alimentazione rimarà a +12 volt per: 1/9600=0.104 mS.

Questo segnale viene denominato **START BIT** ed è sempre presente all'inizio di trasmissione di ogni singolo byte.

Dopo lo start bit vengono trasmessi in sequenza gli otto bit componenti il codice ASCII del carattere trasmesso partendo dal bit meno significativo. Nel nostro caso la lettera A maiuscola corrisponde al valore binario **01000001** per cui la sequenza di trasmissione sarà la seguente:



Una volta trasmesso l'ottavo bit (bit 7), il PC aggiunge automaticamente un ultimo bit a 1 denominato **STOP BIT** ad indicare l'avvenuta trasmissione dell'intero byte. La stessa sequenza viene ripetuta per ogni byte trasmesso sulla linea.

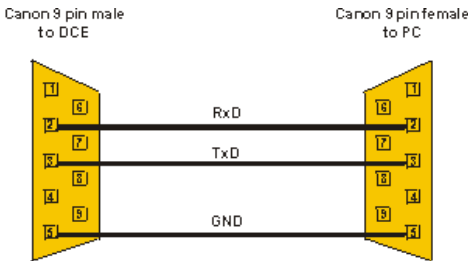
Aggiungendo al nostro cavo seriale una connessione tra il pin TXD (pin 3) del PC ricevente con il pin RXD (pin 2) del PC trasmittente, potremo effettuare una trasmissione RS232 bidirezionale. Il cavo che abbiamo ottenuto è il più semplice cavo NULL MODEM in grado di mettere in collegamento tra loro due DTE.

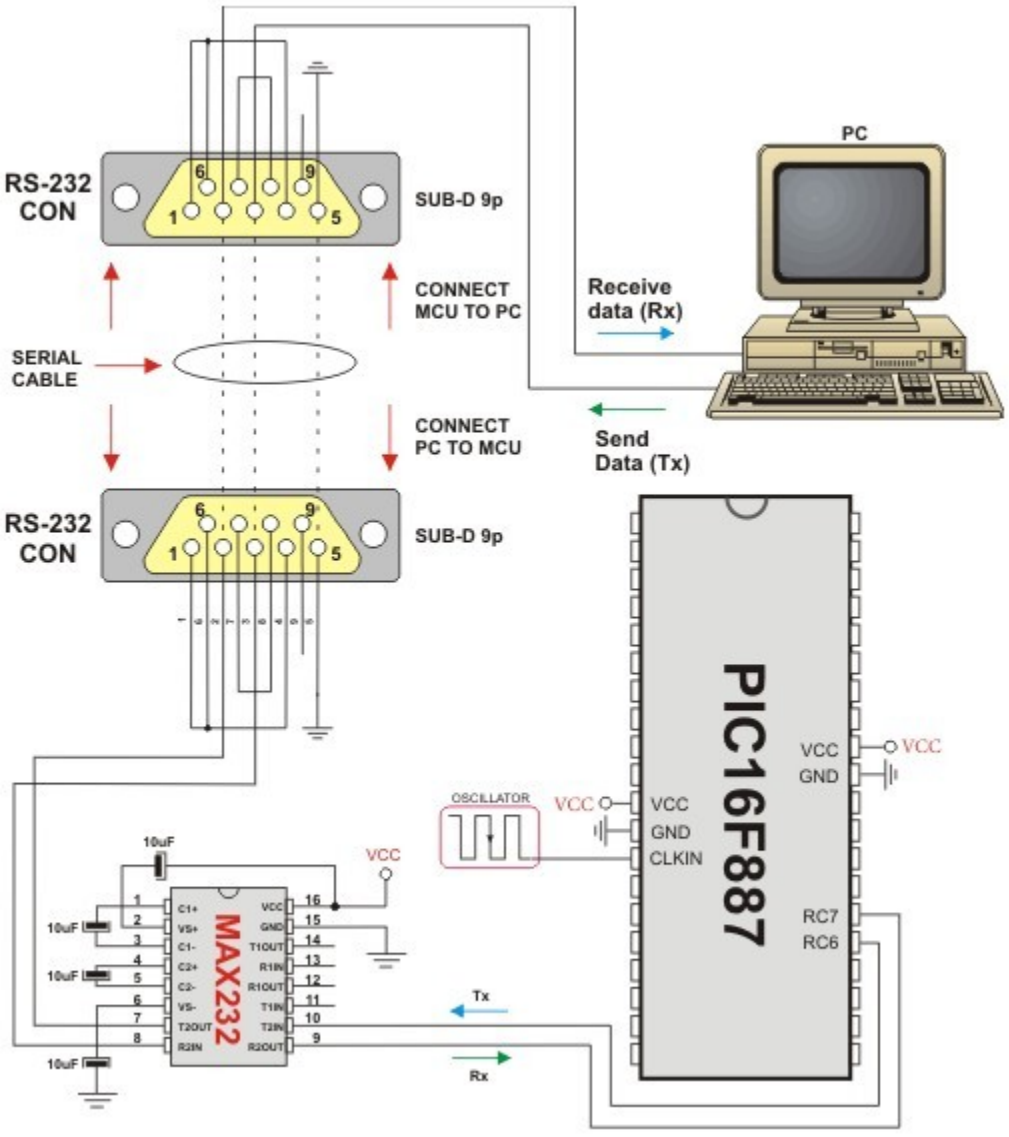
Come collegare il nostro circuito con microcontrollore

Come accennato prima il nostro circuito d'esempio simula un dispositivo DCE. Questo significa che il cavo che dovremo realizzare non dovrà essere di tipo NULL MODEM o INVERTENTE ma **DRITTO** ovvero con i pin numerati allo stesso modo connessi tra loro. Questo tipo di cavo è identico a quelli che vengono usati per connettere al PC un modem esterno.

Dato che i dispositivi DTE sono sempre dotati di connettore DB9 maschio, il nostro circuito, essendo un DCE, avrà un connettore DB9 femmina. In alcuni casi i PC sono dotati di connettori DB25 anziché DB9 per cui per le equivalenze occorre consultare la piedinatura dei connettori RS232.

Il cavo di collegamento tra il PC ed il nostro circuito dovrà essere intestato a sua volta con un connettore femmina da un lato per poter essere inserito nella seriale del PC ed un connettore maschio dall'altro per poter essere inserito nel connettore del nostro circuito di prova. I collegamenti interni al cavo da usare sono riportati nella seguente figura.

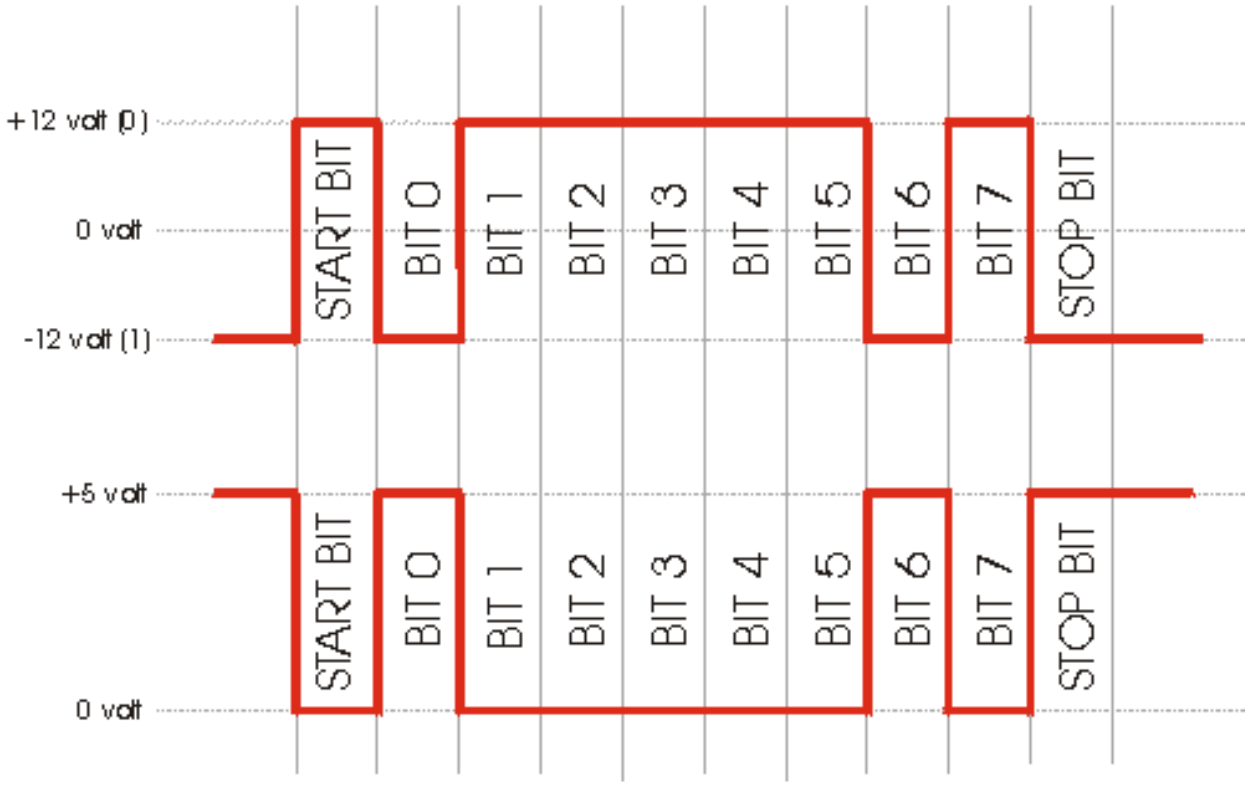




Funzionamento del MAX232

Come accennato prima, nel nostro circuito d'esempio useremo un driver RS232, ovvero un integrato in grado di convertire i segnali a +/- 12 volt tipici della RS232 in segnali a 0/5 volt gestibili dalle porte del PIC.

Il segnale perciò verrà convertito in modo che il livello 0 di un bit corrisponde a 0 Volt ed il livello 1 di un bit corrisponde a +5 Volt. La conversione avviene sia in ricezione che in trasmissione, pertanto se sulla linea RX c'è , un valore di 12 Volt dopo l'integrato MAX232 sul pin del microcontrollore ci saranno 0 Volt. Se sul pin del microcontrollore (cioè all'ingresso dell'integrato MAX232) viene messa una tensione di 0 Volt all'uscita dell'integrato e cioè sulla linea ci saranno 12 Volt.



Sinteticamente possiamo riassumere tutto il discorso dicendo che lo standard di comunicazione seriale RS232 prevede la trasmissione e ricezione su due piedini definiti RX e TX.

La comunicazione avviene pertanto in maniera seriale, inviando un bit alla volta, ed in maniera asincrona definendo la velocità di trasmissione in bit per secondo (bps) es, 1200bps, 2400 bps, 4800bps, 9600 bps, 14400 bps, 19200 bps, 38400 bps, 115200bps, sono delle velocità di trasmissione spesso utilizzate nella comunicazione seriale. Se ad esempio decidessimo di trasmettere e ricevere alla velocità di 9600bps ogni singolo bit avrà una durata di 104us.

La trasmissione avviene un byte alla volta, e la trasmissione del byte è composta da un bit di start, uno o più bit di stop (a seconda delle impostazioni) e dagli 8 bit che compongono il carattere da inviare, il primo bit è quello meno significativo LSB.

GESTIONE SERIALE CON PIC

In determinati tipi di microcontrollori PIC, è presente un circuito interno al componente che gestisce la comunicazione seriale, e riceve o invia automaticamente un byte passando per due piedini ben definiti (RC6 e RC7 nel caso del PIC18F458) un registro interno del PIC.

RC6/TX/CK	17	25	44	27			
RC6					I/O	ST	Digital I/O.
TX					O	—	USART asynchronous transmit.
CK					I/O	ST	USART synchronous clock (see RX/DT).
RC7/RX/DT	18	26	1	29			
RC7					I/O	ST	Digital I/O.
RX					I	ST	USART asynchronous receive.
DT					I/O	ST	USART synchronous data (see TX/CK).

Legend: TTL = TTL compatible input; CMOS = CMOS compatible input or output; ST = Schmitt Trigger input with CMOS levels; Analog = Analog input; I = Input; O = Output

La gestione di questo circuito interno al PIC avviene tramite dei registri interni, dove troviamo anche il valore da inviare o quello ricevuto. Di seguito tutti i registri coinvolti nella gestione della trasmissione seriale.

TABLE 18-10: REGISTERS ASSOCIATED WITH SYNCHRONOUS SLAVE TRANSMISSION

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	1111 1111	1111 1111
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000u
TXREG	USART Transmit Register								0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for synchronous slave transmission.
Note 1: These registers or register bits are not implemented on the PIC18F248 and PIC18F258 and read as '0's.

TABLE 18-11: REGISTERS ASSOCIATED WITH SYNCHRONOUS SLAVE RECEPTION

Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on POR, BOR	Value on all other Resets
INTCON	GIE/GIEH	PEIE/GIEL	TMR0IE	INT0IE	RBIE	TMR0IF	INT0IF	RBIF	0000 000x	0000 000u
PIR1	PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF	0000 0000	0000 0000
PIE1	PSPIE(1)	ADIE	RCIE	TXIE	SSPIE	CCP1IE	TMR2IE	TMR1IE	0000 0000	0000 0000
IPR1	PSPIP(1)	ADIP	RCIP	TXIP	SSPIP	CCP1IP	TMR2IP	TMR1IP	1111 1111	1111 1111
RCSTA	SPEN	RX9	SREN	CREN	ADDEN	FERR	OERR	RX9D	0000 000x	0000 000u
RCREG	USART Receive Register								0000 0000	0000 0000
TXSTA	CSRC	TX9	TXEN	SYNC	—	BRGH	TRMT	TX9D	0000 -010	0000 -010
SPBRG	Baud Rate Generator Register								0000 0000	0000 0000

Legend: x = unknown, - = unimplemented, read as '0'. Shaded cells are not used for synchronous slave reception.
Note 1: These registers or register bits are not implemented on the PIC18F248 and PIC18F258 and read as '0's.

La trasmissione seriale può pertanto essere gestita in assembler utilizzando i registri sopra indicati, ma grazie all'ambiente di sviluppo mikroC esistono delle librerie già realizzate, oltre che la possibilità di gestire la trasmissione in maniera software.

L'ambiente di sviluppo mikroC prevede al suo interno due tipi di librerie una software che consente di effettuare trasmissione e ricezione di byte su qualsiasi pedino, ed una hardware che invece utilizza i registri ed i piedini descritti precedentemente.

Ovviamente è preferibile utilizzare la seconda libreria quella hardware, che impiega sicuramente meno risorse del processore.

Le due librerie sono ampiamente descritte nella manualistica del compilatore, a cui si può facilmente accedere mediante il tasto F1.

Di seguito approfondiremo solo la libreria hardware e vedremo un esempio di funzione per gestire una comunicazione seriale.

SVILUPPO DI FUNZIONI PER TRASMISSIONE E RICEZIONE SERIALE

Funzione trasmetti

```
//DEFINIZIONE SEGNALI PER RS232
//solo per le funzioni software trasmetti e ricevi
sbit TX_RS232  at RB2_bit;
sbit RX_RS232  at RB1_bit;
```

```
//TRASMISSIONE BYTE VIA RS232  9600 bps
//va passato il byte da inviare
void trasmetti(char dato_tx){
     char cont;   //variabile per conteggio bit

     cont=8;
     //inizio la trasmissione
     TX_RS232=0;              //start bit
     delay_us(100);           //ritardo di 104 microsecondi
     //invio il byte
     while(cont>0)
      {
        if(dato_tx.B0==1)   TX_RS232=1;   //testo il bit meno significativo LSB di dato_tx
        else                TX_RS232=0;
        dato_tx=dato_tx>>1;   //shit di dato_tx a destra di una posizione
        cont--;               //decremento il contatore dei bit
        delay_us(100);        //durata di un bit
      }//while(cont)
     TX_RS232=1;     //stop bit
     delay_us(104);  //ritardo d 104 microsecondi durata stop bit
}//trasmetti
```

L'esempio sopra mostra una funzione per trasmettere un byte, analizziamone il funzionamento.
In primo luogo la funzione è di tipo void, perciò non restituisce alcun valore, ma tra le parentesi viene dichiarata la variabile **dato_tx** di tipo char (8 bit) che dovremo passare quando chiamiamo questa funzione, ad esempio per inviare il carattere A (corrispondente al byte 41 in esadecimale o 65 in decimale) scriverò:

trasmetti(65); *oppure* *trasmetti(0x41);* *oppure* *trasmetti('A');*

Inizialmente occorre dichiarare su quale pin verrà inviato e su quale pin verrà ricevuto il dato.
Nel nostro caso al pin RB2 viene associata la label TX_RS232, ed al pin RB1 la label RX_RS232.

Bisogna ricordarsi all'inizio del programma dopo il main, di porre il pin TX_RS232 ad un livello alto, come prevede l standard RS232, ovviamente prima va configurato il registro TRISB per definire quale pin è in ingresso e quale in uscita, nel nostro caso RX è un ingresso e TX è un uscita.

```
void main(){
     CMCON=0x07; //disabilito i comparatori all'interno del PIC
     TRISB=0x02; //porta RB2=OUT RB1=IN
     TX_RS232=1;  //metto TX in stato di IDLE  solo se uso la funzione trasmetti
```

All’inizio della funzione viene dichiarata e poi inizializzata al valore 8 la variabile cont, che servirà per contare i bit inviati sulla seriale.

Successivamente occorre creare lo start bit perciò mettere il pin TX_RS232 ad un livello basso per una durata definita dalla velocità di trasmissione che nell’esempio è di 9600 bps. Ogni bit in questo caso dovrà durare 104 microsecondi.

Poi ci sarà il ciclo while dove vengono inviati gli 8 bit che compongono il dato da trasmettere contenuto nella variabile **dato_tx**.

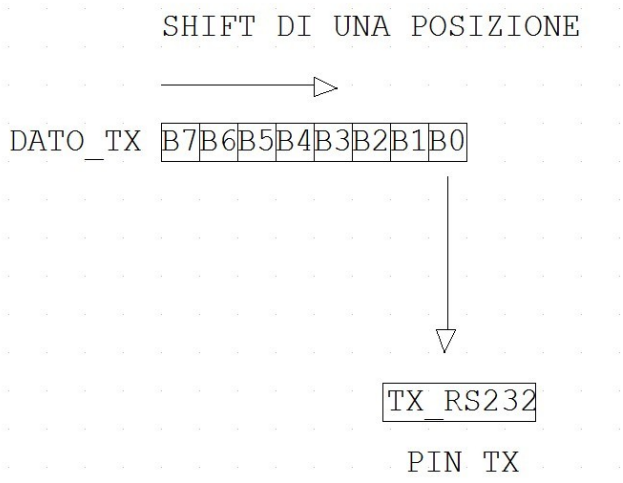
Il ciclo while viene ripetuto per 8 volte, infatti la condizione per rimanere nel ciclo è che la variabile cont sia maggiore di 0.

All’interno del ciclo while, viene testato il bit meno significativo del dato da trasmettere e viene messo a 0 o ad 1 il pin di uscita TX_RS232;

```
if(dato_tx.B0==1)       TX_RS232=1;   //testo il bit meno significativo LSB di dato_tx
else                    TX_RS232=0;
```

Scrivendo dato_tx.B0 si intende il bit 0 della variabile dato_tx.

Successivamente con l’istruzione; *dato_tx=dato_tx>>1* si effettua un shift a sinistra di una posizione, in modo da portare il bit 1 nella posizione del bit 0, in modo che al ciclo successivo testando B0 si legge il valore del secondo bit, e così via per tutti gli 8 bit. Tra un test e l’altro occorre aspettare sempre un ritardo di 104 microsecondi che nel nostro caso è impostato a 100 microsecondi, per considerare il tempo impiegato dal microcontrollore per eseguire le varie istruzioni.



Alla fine dopo il ciclo while viene messo ad uno il pin TX_RS232 per 104 microsecondi, per indicare il bit di stop.

Funzione ricevi

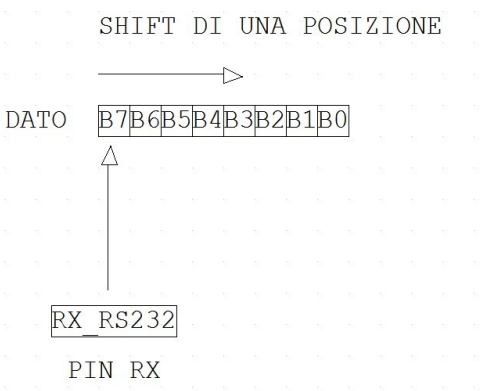
```
//RICEZIONE DA SERIAL A 9600bps
//la funzione va chiamata se il bit RX è andato a 0 (start bit)
//la funzione restituisce il byte ricevuto
char ricevi(){
     char dato_rx=0,cont=8;

     delay_us(150); //ritardo di 1,5 bit per posizionamento al centro del 1°bit

     while(cont)
      {
        if(RX_RS232==1) dato_rx.B7=1;  //test del bit RX, modifica bit 7 di dato
        else            dato_rx.B7=0;
        if(cont>1)      dato_rx=dato_rx>>1; //shit di dato_tx a destra
        cont--;                             //decremento del contatore dei bit
        delay_us(100);
      }//while(cont)

     delay_us(104);  //attesa stop bit, ritardo d 104 microsecondi
     return(dato_rx);
}//ricevi
```

La funzione di ricezione è diversa in quanto non è di tipo void, perché deve restituire il byte ricevuto.
Viene chiamata all'interno del programma utilizzando una variabile di tipo char che conterrà il valore restituito dalla funzione. Ad esempio dopo aver dichiarato una variabile di tipo char chiamata dato_rx, posso chiamare la funzione nel seguente modo:

```
char dato_rx;
.
.
if (RX_RS232==0)  dato_rx=ricevi();  //se il pin RX è 0 inizia la ricezione
```

All'interno della funzione vengono dichiarate due variabili una che contiene il valore ricevuto, ed una per il conteggio dei bit ricevuti. Inizialmente c'è un tempo di attesa pari a 150microsecondi cioè la durata di un bit + quella di mezzo bit. In questo modo mi posiziono al centro del primo bit da ricevere.

Poi faccio un test del pin RX e metto il risultato nel bit più significativo della variabile dato (dato.B7) in modo che successivamente shiftando per 8 volte la variabile dato, questo bit andrà a collocarsi alla posizione B0 del bit meno significativo.



Successivamente nel ciclo while viene fatta la stessa cosa ripetuta per altre 7 volte, sempre con un tempo di ritardo, che in teoria dovrebbe essere di 104 microsecondi, in pratica, per tener conto dei ritardi delle istruzioni si mette un tempo più piccolo di 100microsecondi, anche se sperimentalmente abbiamo visto essere necessario un tempo ancora inferiore

Al termine si attende la durata del bit d stop e si restituisce il byte contenente il dato ricevuto, *return(dato)*.
Le due funzioni sopra descritte presentano delle criticità sul dimensionamento dei tempi, sarebbe perciò preferibile utilizzare le funzioni che utilizzano la parte hardware interna del PIC.
In questo caso il compilatore mette a disposizione le seguenti librerie:

Al posto della “x” di ogni libreria va messo nel nostro caso il numero 1, perché abbiamo una sola seriale hardware nel PIC, in caso contrario bisogna specificare quale seriale si sta utilizzando.

- UART1_Init — es. UART1_Init(9600) questa funzione inizializza la velocità di trasmissione e va chiamata subito dopo il main.
- UART1_Data_Ready — es. if (UART1_Data_Ready()==1) …….

questa funzione restituisce il valore 1 se c'è un dato presente nel registro interno del PIC, pronto per essere letto con la funzione UART1_Read()

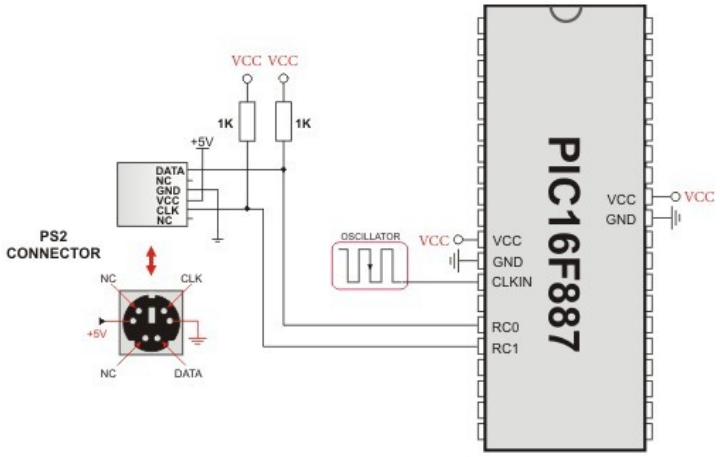
- UART1_Tx_Idle — es. if (UART1_Tx_Idle==1) UART_Write('a')
  questa funzione restituisce 1 se la trasmissione precedente del dato è stata completata e perciò posso inviare altri dati0
- UART1_Read — restituisce il valore letto dalla seriale es, dato=UART1_Read()
- UART1_Read_Text(..) — Legge fino ad una sequenza di caratteri definita da un delimitatore (vedi help online del compilatore)
- UART1_Write() — Invia un carattere es. UART1_Write('a')
- UART1_Write_Text(..) — Invia una sequenza di caratteri (vedi help online del compilatore)
- UART_Set_Active — Per i microcontrollori che hanno più moduli interni per la gestione seriale, questa funzione definisce quale modulo è attivo.

GESTIONE TASTIERA PS2 E DISPLAY LCD

Tra le varie librerie disponibili nell'ambiente di sviluppo mikroC ci sono quelle relative alla gestione di una tastiera tramite interfaccia PS2 e di un display tramite interfaccia parallela.

Lo standard PS2 prevede due soli segnali, dove i dati transitano in maniera seriale sincrona.

Un esempio di collegamento può essere l seguente:



In questo caso le librerie messe a disposizione sono due:

Library Routines

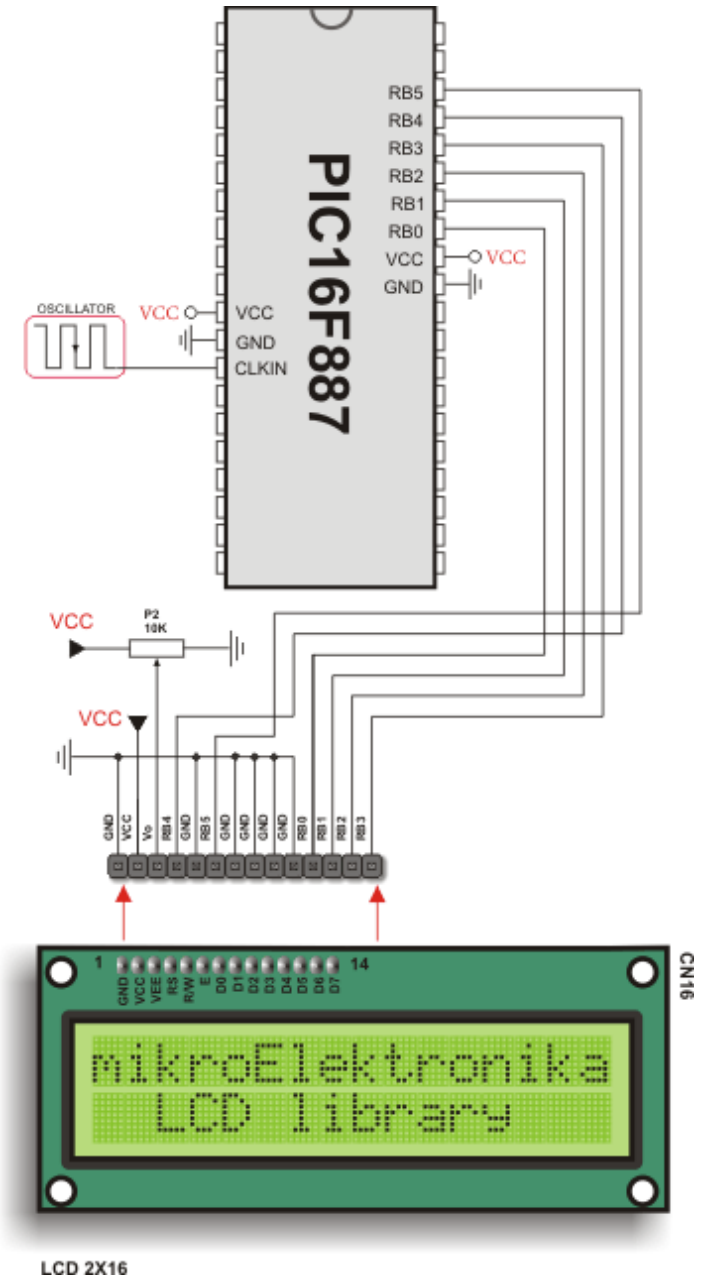
- Ps2_Config
- Ps2_Key_Read

Ps2_Config, predispone la gestione della PS2 tramite i pin indicati all'inizio del programma. Ps2_Key_Read, restituisce su 3 variabili le informazioni essenziali e cioè; il codice del carattere ricevuto, se è un carattere speciale (invio, F1, Esc ecc…) e se il tasto è ancora premuto.

Un esempio di programma può essere il seguente:

```
//DEFINIZIONI SEGNALI PER TASTIERA PS2
//da utilizzare con le funzioni:
//PS2_CONFIG,  PS2_KEY_READ
sbit PS2_Data at RC0_bit;
sbit PS2_Clock at RC1_bit;
//imposto la direzione di ogni bit
sbit PS2_Data_Direction at TRISC0_bit;
sbit PS2_Clock_Direction at TRISC1_bit;

//variabii utilizzate per la gestione della tastiera PS2
unsigned short keydata = 0, special = 0, down = 0;

void main(){
     CMCON=0x07; //disabilito i comparatori all'interno del PIC

     UART1_Init(9600);  //imposto velocità di trasmissione  9600bps
     Ps2_Config();  //configuro la PS2 secondo i piedini descritti inizalmente

     while (1)
     {
       if (Ps2_Key_Read(&keydata, &special, &down)) {
                if(special==0) UART1_Write(keydata);  //se non è un carattere speciale invio il carattere sulla seriale
              }
        while(down==1)   //attendo il rilascio
          {
           Ps2_Key_Read(&keydata, &special, &down)
          }//while
     }//while (1);

}//main
```

Le funzioni disponibili per la gestione di un display LCD, prevedono una gestione parallela e funzionano solo se su display c'è un controller compatibile HD44780 (una degli standard più diffusi in commercio)
Uno schema di esempio è il seguente:



LCD 2X16

Nello schema il display viene controllato con 4 linee dati D4,D5,D6 e D7 oltre che con i due segnali RS e Enable. Questi segnali sono collegati a 6 pin della porta B. Il display LCD Dotmatrix alfanumerico, è strutturato in righe e colonne, nella figura il display ha due righe per 16 colonne. Anche in questo caso le librerie hanno bisogno della definizione dei pin collegati al display:

```
sbit LCD_RS at RB2_bit;
sbit LCD_EN at RB3_bit;
sbit LCD_D4 at RB4_bit;
sbit LCD_D5 at RB5_bit;
sbit LCD_D6 at RB6_bit;
sbit LCD_D7 at RB7_bit;
sbit LCD_RS_Direction at TRISB2_bit;
sbit LCD_EN_Direction at TRISB3_bit;
sbit LCD_D4_Direction at TRISB4_bit;
```

```
sbit LCD_D5_Direction at TRISB5_bit;
sbit LCD_D6_Direction at TRISB6_bit;
sbit LCD_D7_Direction at TRISB7_bit;
```

Le librerie messe a disposizione sono le seguenti:

Si rimanda al manuale disponibile premendo F1 sul compilatore mikroC per le caratteristiche delle varie librerie, possiamo sinteticamente elencare le loro funzioni:

- Lcd_Init Predispone il PIC ad operare con il display collegato ai pin indicati
- Lcd_Out Visualizza sul display un stringa di caratteri contenuta tra le virgolette “” stringa “”

nella riga e colonna indicata. Es Lcd_Out(1,2,”ciao”) visualizza sulla prima riga seconda colonna la stringa ciao.

- Lcd_Out_cp Come la precedente ma non vanno indicate riga e colonna, in quando la stringa viene visualizzata sulla corrente posizione del cursore.
- Lcd_Chr Visualizza sul display un carattere contenuto tra gli apici 'a'

nella riga e colonna indicata. Es Lcd_Chr(1,2,'a') visualizza sulla prima riga seconda colonna il carattere a.

- Lcd_Chr_cp Come la precedente ma non vanno indicate riga e colonna, in quando il carattere viene visualizzato sulla corrente posizione del cursore.
- Lcd_Cmd Invia un comando al display come ad esempio cancella il display, fa lampeggiare il cursore ecc... .L'elenco dei comandi disponibili è sul manuale (F1 sul compilatore) i più usati sono comunque i seguenti:
  - Lcd_Cmd(_LCD_CLEAR) cancella il display
  - Lcd_Cmd(_LCD_CURSOR_OFF) non visualizza il cursore
  - Lcd_Cmd(_LCD_BLINK_CURSOR_ON) fa lampeggiare il cursore

Di seguito l'elenco completo.

Available Lcd Commands

Lcd Command	Purpose
_LCD_FIRST_ROW	Move cursor to the 1st row
_LCD_SECOND_ROW	Move cursor to the 2nd row
_LCD_THIRD_ROW	Move cursor to the 3rd row
_LCD_FOURTH_ROW	Move cursor to the 4th row
_LCD_CLEAR	Clear display
_LCD_RETURN_HOME	Return cursor to home position, returns a shifted display to its original position. Display data RAM is unaffected.
_LCD_CURSOR_OFF	Turn off cursor
_LCD_UNDERLINE_ON	Underline cursor on
_LCD_BLINK_CURSOR_ON	Blink cursor on
_LCD_MOVE_CURSOR_LEFT	Move cursor left without changing display data RAM
_LCD_MOVE_CURSOR_RIGHT	Move cursor right without changing display data RAM
_LCD_TURN_ON	Turn Lcd display on
_LCD_TURN_OFF	Turn Lcd display off
_LCD_SHIFT_LEFT	Shift display left without changing display data RAM
_LCD_SHIFT_RIGHT	Shift display right without changing display data RAM