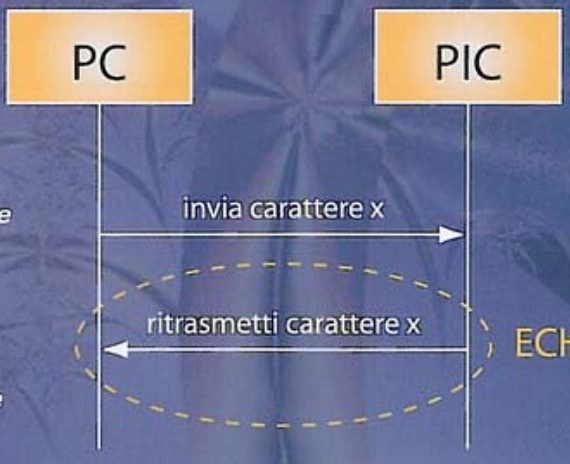
IL PRIMO FIRMWARE PER LA SERIALE

In questo Workshop proseguiamo lo studio dell'utilizzo della porta seriale con i PIC, dedicandoci agli aspetti software della sua programmazione con l'ambiente di sviluppo mikroC.

Nell'articolo precedente siamo giunti a presentare le quattro funzioni di mikroC dedicate alla gestione della porta seriale dei PIC (che puoi vedere riassunte nella tabella in basso a pag. 6 assieme ai rispettivi prototipi). Nel corso dell'articolo vedremo come impiegarle per scrivere un semplici programma dimostrativo per l'uso della seriale e per implementare una nuova funzione ausiliaria di maggiore complessità.

IL PROGRAMMA DI ESEMPIO

Il primo programma che realizzeremo sarà un semplice firmware che implementerà le **funzionalità di 'echo'** e che ci permetterà di **controllare una coppia di LED collegati al PIC.** Iniziamo chiarendo il significato del concetto di echo. Abbiamo visto che il collegamento seriale tra PIC e PC è possibile grazie a due linee monodirezionali indipendenti che permettono lo scambio di informazioni tra i due dispositivi. **L'effetto di echo non è altro che un meccanismo automatico di ritrasmissione al mittente dei dati ricevuti durante il processo di comunicazione.** Come esempio pratico, possiamo far riferimento allo schema sottostante. In questo schema, nel quale vedi rappresentato un ipotetico collegamento tra un PC e un PIC, il personal computer invia un generico carattere 'x' al microcontrollore attraverso la propria linea di trasmissione TX. Non appena il carattere viene ricevuto dal PIC, esso lo ritrasmetterà al PC, generando, appunto, l'effetto di echo. Nel nostro programma il carattere dovrà, comunque, essere memorizzato dal PIC, in modo da poter essere **interpretato**, in quanto il firmware di esempio prevede la gestione di due LED pilotati da altrettanti piedini del PIC (useremo i pin 0 e 1 della porta A). **L'accensione e lo spegnimento dei due dispositivi saranno controllati attraverso appositi caratteri**, che per convenzione vediamo riassunti nella tabella in alto a pagina 6.



Parte dell'esperimento di oggi riguarda l'implementazione di un effetto di 'echo', ossia di ritrasmissione, dei dati ricevuti dal PIC durante la comunicazione con il personal computer.

CARATTERE	FUNZIONE
a	Spegni LED numero 1
A	Accendi LED numero 1
b	Spegni LED numero 2
B	Accendi LED numero 2
0	Spegni tutti i LED
1	Accendi tutti i LED

LA FUNZIONE USART_WRITE_STRING ▸▸▸

Prima di dedicarci all'implementazione del firmware per l'esperimento vero e proprio, creiamo una nuova funzione che ci sarà utile per scrivere il nostro programma: la funzione **'Usart_Write_String'**. Scopo di tale funzione sarà quello di **inviare sulla porta seriale una intera stringa di testo, terminante con i due caratteri '\r'** (corrispondente al carattere 'carriage return') **e '\n'** (carattere 'new line'). Dalla tabella sotto, infatti, puoi osservare come mikroC non offra procedure per l'invio di un intero blocco di byte, ma si limiti alla trasmissione di elementi di un singolo byte. La **chiamata iterativa della funzione 'Usart_Write'**, tuttavia, verrà incontro alle nostre necessità permettendoci di automatizzare l'invio delle stringhe di testo. Nel box in alto a pag. 7 vediamo il codice, che dovrà essere riportato nel sorgente mikroC prima della funzione 'main'.

LE FUNZIONI DI MIKROC PER LA GESTIONE DEL MODULO USART

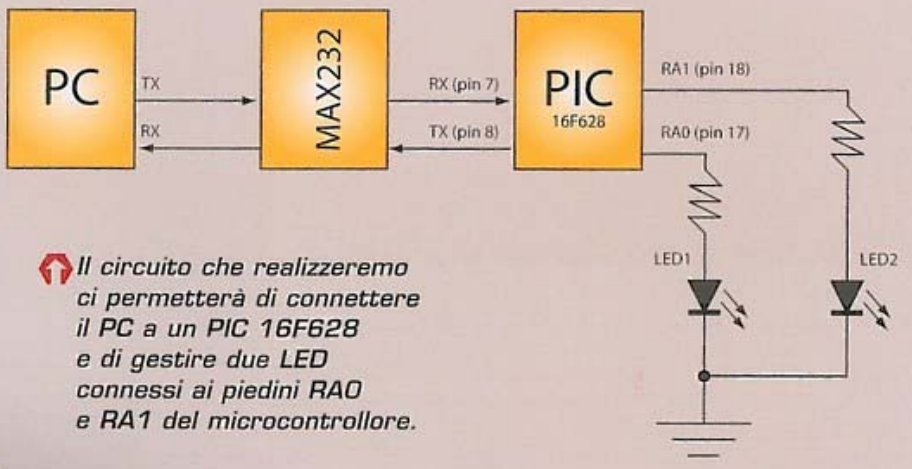
NOME DELLA FUNZIONE	DESCRIZIONE
void Usart_Init (const unsigned long baud_rate);	Inizializza il modulo USART impostando la velocità di trasmissione della porta. I valori più comuni sono 9600, 14400, 28800 e 57600 baud. La funzione deve essere chiamata prima dell'uso del modulo USART.
unsigned short Usart_Data_Ready (void);	Permette di verificare la presenza di dati nel buffer di ricezione del PIC. Se restituisce 1 il buffer contiene dati che possono essere letti.
unsigned short Usart_Read(void);	Permette di leggere un byte dal buffer di ricezione (se presente, informazione ottenibile tramite la funzione 'Usart_Data_Ready'). Una volta chiamata, restituisce il byte letto.
void Usart_Write (unsigned short data);	Invia al modulo USART il byte passato come parametro da trasferire sulla linea seriale.

La funzione Usart_Write_String

```
void  Usart_Write_String (char * stringa)
{
  int i; /* indice di scorrimento della stringa */

   /* utilizziamo la struttura ciclica 'for' per scorrere i caratteri della
    stringa fino al raggiungimento del carattere di terminazione '\0' */
   for(i=0 ; stringa[i] != '\0' ; i++ )
   {
         /* Per ogni carattere della stringa diverso dal carattere di terminazione,
            eseguiamo l'invio sulla linea seriale tramite la funzione Usart_Write*/
         Usart_Write(stringa[i]);
   }

   /* terminato l'invio della stringa ricevuta come parametro,
      trasmettiamo i due caratteri di chiusura '\r' e '\n' */
   Usart_Write('\r');
   Usart_Write('\n');
}
```



Il circuito che realizzeremo ci permetterà di connettere il PC a un PIC 16F628 e di gestire due LED connessi ai piedini RA0 e RA1 del microcontrollore.

IL SORGENTE DEL FIRMWARE ›››

Veniamo ora all'analisi del codice sorgente del primo programma di comunicazione seriale, per la scrittura del quale faremo riferimento alla struttura a blocchi del circuito mostrata nello schema a sinistra (in particolare per i LED).

Il sorgente

```
/* definizioni utili */
/* definizione delle linee di controllo dei LED */
#define LED1 PORTA.F0
#define LED2 PROTA.F1

/* definizione degli stati dei LED */
#define ACCESO 1
#define SPENTO 0

/* definizione dei comandi interpretabili (associamo ad ogni carattere
interpretabile un simbolo che ne descrive esplicitamente la sua funzione) */
#define ACCENDI_LED1 'A'
#define SPEGNI_LED1 'a'
```

```
#define ACCENDI_LED2 'B'
#define SPEGNI_LED2 'b'

#define ACCENDI_TUTTO '1'
#define SPEGNI_TUTTO '0'


/* definizione della funzione  Usart_Write_String. Rispetto al codice visto nella pagina
precedente, utilizziamo come parametro una stringa 'const' in quanto i
messaggi saranno definiti come 'const char []' */
void Usart_Write_String (const char * stringa)
{
  int i; /* indice di scorrimento della stringa */

   /* utilizziamo la struttura ciclica 'for' per scorrere i caratteri della stringa
   fino al raggiungimento del carattere di terminazione '\0' */
   for(i=0 ; stringa[i] != '\0' ; i++ )
   {
         /* Per ogni carattere della stringa diverso dal carattere di terminazione,
eseguiamo l'invio sulla linea seriale tramite la funzione Usart_Write*/
         Usart_Write( stringa[i]);
   }

   /* terminato l'invio della stringa ricevuta come parametro, trasmettiamo i due
     caratteri di chiusura '\r' e '\n' */
   Usart_Write('\r');
   Usart_Write('\n');
}



/* funzione principale 'main' */
void main()
{
  /* variabile per memorizzare i byte ricevuti */
  unsigned short byte_ricevuto;

  /* definizione dei messaggi che verranno trasmessi al PC. Le variabili che li con
     tengono sono dichiarati utilizzando il tipo 'const char []' per non consumare
     la memoria RAM del PIC. Le stringhe dichiarate char, infatti, sono utilizzate
     direttamente dalla memoria flash del microcontrollore e non occupano
     inutilmente memoria di lavoro (tuttavia, essendo costanti, non possono essere
     modificate). */
  const char messaggio_benvenuto [] = "Primo esempio di comunicazione seriale.\r\n";
  const char messaggio_LED1_ON [] = "LED1 Acceso\r\n";
  const char messaggio_LED1_OFF [] = "LED1 Spento\r\n";
  const char messaggio_LED2_ON [] = "LED2 Acceso\r\n";
  const char messaggio_LED2_OFF [] = "LED2 Spento\r\n";
  const char messaggio_TUTTO_ON [] = "LED Accesi\r\n";
  const char messaggio_TUTTO_OFF [] = "LED Spenti\r\n";
  const char messaggio_comando_sconosciuto [] = "Comando sconosciuto\r\n";

  /* inizializzazione delle porte A e B */
  TRISA = 0;  /* porta A interamente in output*/
  CMCON = 0x7; /* configuriamo la porta A come I/O digitale */
  PORTA = 0; /* tutti i pin della porta A a 0 */
  TRISB = 0b00000010; /* forziamo il pin TX (pin 2) in out e il pin RX (pin 1)
                      in input */

  /* inizializziamo la comunicazione seriale a 9600 baud */
  Usart_Init(9600);
  Delay_ms(10); /* Applichiamo una breve pausa */
```

```
  /* stampiamo il messaggio di benvenuto */
  Usart_Write_String (messaggio_benvenuto);

  /* ciclo principale del software */
  while(1)
  {
     /*controlliamo se è presente un dato in arrivo */
     if (Usart_Data_Ready())
     {
        /* se è stato ricevuto un byte, lo memorizziamo e lo interpretiamo */
        byte_ricevuto = Usart_Read();

        /* ritrasmissione del carattere ('echo') con l'aggiunta
           dei caratteri '\n' e '\r' */
        Usart_Write(byte_ricevuto);
        Usart_Write('\r');
        Usart_Write('\n');

        /* Applichiamo il costrutto 'switch/case' per eseguire il comando
          appropriato, in base al carattere ricevuto */
        switch(byte_ricevuto)
        {
            /* ricevuto il carattere 'A', accendiamo il LED1 */
            case  ACCENDI_LED1: LED1 = ACCESO;
                                Usart_Write_String (messaggio_LED1_ON);
                                break;

             /* ricevuto il carattere 'a', spegniamo il LED1 */
            case  SPEGNI_LED1: LED1 = SPENTO;
                               Usart_Write_String (messaggio_LED1_OFF);
                               break;

             /* ricevuto il carattere 'B', accendiamo il LED2 */
            case  ACCENDI_LED2: LED2 = ACCESO;
                                Usart_Write_String (messaggio_LED2_ON);
                                break;

             /* ricevuto il carattere 'a', spegniamo il LED2 */
            case  SPEGNI_LED2: LED2 = SPENTO;
                               Usart_Write_String (messaggio_LED2_OFF);
                               break;

            /* ricevuto il carattere '1', accendiamo entrambi i LED */
            case  ACCENDI_TUTTI: LED1 = ACCESO;
                                 LED2 = ACCESO;
                                 Usart_Write_String (messaggio_TUTTO_ON);
                                 break;

            /* ricevuto il carattere '0', spegniamo entrambi i LED */
            case  SPEGNI_TUTTI: LED1 = SPENTO;
                                LED2 = SPENTO;
                                Usart_Write_String (messaggio_TUTTO_OFF);
                                break;

             /* se il carattere è diverso da quelli sopraelencati,
                il comando è sconosciuto */
            default: Usart_Write_String (messaggio_comando_sconosciuto);
                     break;

        } /* chiusura switch */
     } /* chiusura  if (Usart_Data_Ready()) */
  } /* chiusura  while(1) */
}
```