UNA SEMPLICE SCHEDA SERIALE PER SERVO

Da questo fascicolo iniziamo a osservare un esempio di firmware che ci permetterà di costruire una semplice scheda di gestione per servocomandi analogici, dotata di interfaccia di controllo seriale.

In questo nuovo Workshop iniziamo a dedicarci a un nuovo progetto di una certa complessità, che ci permetterà di realizzare una **scheda di controllo per servocomandi analogici a quattro canali**. Data la mole del codice necessario a implementare il funzionamento della scheda di esempio, codice che non può essere contenuto per intero in un singolo Workshop, inizieremo suddividendo il sorgente mikroC tra questo fascicolo e il successivo, dedicandoci poi all'analisi del circuito elettrico della scheda e della struttura logica del programma di esempio. A differenza dei listati mostrati nei numeri precedenti, noterai che **accanto a ogni linea di codice è stato aggiunto un numero di tre cifre**: tale elemento rappresenta semplicemente il **numero di linea**, un riferimento che ci tornerà utile nel momento in cui andremo a commentare il codice e ci permetterà di facilitare il concatenamento delle due parti del programma.

La semplice scheda di controllo su cui potrà essere caricato il firmware presentato in questi Workshop.

L'USO DELL'ASSEMBLY

Leggendo il codice di esempio presentato in questo fascicolo ti accorgerai dell'utilizzo di un costrutto di programmazione che non è mai stato impiegato prima d'ora: il costrutto **asm{...}**. Questa particolare istruzione permette di **definire un blocco di programma contenente una sequenza di istruzioni in linguaggio assembly di basso livello**. Non avendo mai affrontato la programmazione del 16F628 in assembly è più che normale che tali parti del sorgente non ti siano chiare. Tuttavia, non ti preoccupare, in quanto tutto sarà chiarito. La scelta di utilizzare l'assembly all'interno di un programma C è legata alla necessità di creare una procedura controllata di ritardo, con una durata impostabile dall'utente. Ma iniziamo a presentare la prima parte del codice sorgente del progetto di esempio.

La prima parte del firmware

```
000 #define CONTROLLO_SERVO0 PORTA.F0
001 #define CONTROLLO_SERVO1 PORTA.F1
002 #define CONTROLLO_SERVO2 PORTA.F2
003 #define CONTROLLO_SERVO3 PORTA.F3
004
005 define ACCENSIONE_SERVO0 PORTB.F4
006 #define ACCENSIONE_SERVO1 PORTB.F5
007 #define ACCENSIONE_SERVO2 PORTB.F6
008 #define ACCENSIONE_SERVO3 PORTB.F7
009
010
011 #define IMPULSO_MINIMO_us 500
012
013 #define SI 1
014 #define NO 0
015
016 #define NUMERO_SERVO 4
017
018 #define MAX_POS 255
019 #define MIN_POS 0
020
021 #define DIM_MASSIMA_BUFFER 6
022
022 /* elenco dei messaggi di output firmware */
023 const char stringa_Prompt [] = "\r\nSERVO>";
024 const char stringa_Comando_Sconosciuto [] = "Comando sconosciuto!\r\n";
025 const char stringa_Posizione_cambiata [] = "Servo azionato";
026 const char stringa_Nuova_linea [] = "\r\n";
027 const char stringa_Sintassierrata [] = "Sintassi Errata";
028 const char stringa_StatoCambiato [] = "Stato del servo modificato";
029
030 unsigned TEMP_REG;
031 unsigned ASM_DELAY;
032
033 char car_letto;                          /* Carattere ricevuto tramite la linea seriale */
034 char buffer[DIM_MASSIMA_BUFFER+1];       /* Vettore di memorizzazione dei dati ricevuti */
035 int cursore_buffer;                      /* Cursore di scansione del buffer */
036 int posizioni_servo [NUMERO_SERVO];      /* vettore di interi per memorizzare le posizioni
037                                             dei servocomandi la posizione vale da 0 a MAX_POS
038                                             (ne useremo 256) */
039
040
041 /* La funzione 'isInteger' verifica se la stringa passata è un numero intero o meno.
042  Nel caso in cui il parametro fosse un numero intero restituisce il valore simbolico 'SI';
043 'NO' altrimenti. */
044
045 int isInteger (char * stringa_numerica)
046 {
047   int i; /* variabile cursore */
048
049   /* come prima cosa testiamo se si tratta di una stringa vuota o meno verificando
050      il primo carattere. Se il primo carattere corrisponde al terminatore '\0' significa
051      che la stringa è vuota e non può essere un numero. */
```

```
052    if(stringa_numerica[0]=='\0')
053    {
054       return NO;
055    }
056
057    /* Iniziamo la scansione della stringa */
058    /* La scansione procede finché si raggiunge il carattere di terminazione '\0' */
059    for(i=0; stringa_numerica[i] != '\0'  ; i++)
060    {
061      /* Se il carattere corrente della stringa non è un numero (ossia non è compreso nella
062         tavola caratteri tra i caratteri 0 e 9 inclusi) allora è un carattere alfabetico o
063         un segno di punteggiatura e quindi la stringa non rappresenta un numero intero.
064         La funzione termina restituendo 'NO' */
065      if(stringa_numerica[i]<'0' && stringa_numerica[i]>'9')
066       {
067          return NO;
068       }
069    }
070    /* Se l'elaborazione è arrivata a questo punto, è stato raggiunto il carattere di
071       terminazione '\0' incontrando solo caratteri numerici. La stringa è un numero intero
072      positivo. La funzione termina restituendo il valore SI. */
073      return SI;
074 }
075
076
077
078 /* la funzione_Write_String invia tramite la linea seriale una stringa di caratteri
079   di tipo const char []. Rispetto alla versione presentata nel fascicolo precedente,
080   questa funzione non termina con l'invio dei caratteri '\r' e '\n' */
081
082 void Usart_Write_String(const char * stringa)
083 {
084    int i; /* variabile cursore */
084
085    for(i=0; stringa[i]!='\0'; i++)
086    {
087       Usart_Write(stringa[i]);
088    }
089 }
090
091
092
093 /* Invia tramite la seriale la stringa di prompt testuale del firmware */
094 void printPrompt()
095 {
096    Usart_Write_String(stringa_Prompt);
097 }
098
099
100 void interrupt()
101 {
102    /* Finché sono presenti dati sul modulo USART */
103    if(Usart_Data_Ready())
104     {
105      car_letto = Usart_Read(); /* leggiamo il carattere ricevuto */
106
```

```
107     /* se il carattere è diverso da INVIO (\r) e il buffer non è pieno */
108     if(car_letto != 0x0D && car_letto != 0x0A && cursore_buffer<DIM_MASSIMA_BUFFER)
109     {
110         /* Memorizziamo il carattere nel buffer e incrementiamo l'indice */
111         Usart_Write(car_letto); /* echo del carattere ricevuto */
112         buffer[cursore_buffer] = car_letto;
113         cursore_buffer++;
114     }
115     /* se il buffer è pieno o è stato ricevuto l'invio elaboriamo il comando */
116     else if(strlen(buffer)>0)
117     {
118         /* chiudiamo la stringa, impostando l'ultimo carattere con '\0' */
119         buffer[cursore_buffer]='\0';
120
121         /* Azzeriamo il cursore del buffer */
122         cursore_buffer = 0;
123
124         /* Se il primo carattere è 'M' e la lunghezza della
125            stringa di comando è 5 processiamo il comando di movimento */
126         if(buffer[0]=='M' && strlen(buffer)==5)
127         {
128           char s_posizione [4];   /* stringa in cui copieremo la posizione del servo */
129           char s_numero_servo [2]; /* stringa in cui copieremo il numero del servo */
130           int posizione;     /* nuova posizione del servo */
131           int numero_servo; /* numero servo da muovere */
132
133           /* copiamo il carattere indicativo del numero servo
134              nella stringa dedicata */
135           s_numero_servo[0] = buffer[1];
136           s_numero_servo[1] = '\0';
137
138           /* copiamo i caratteri indicativi della posizione
139              nella stringa dedicata */
140           s_posizione[0] = buffer[2];
141           s_posizione[1] = buffer[3];
142           s_posizione[2] = buffer[4];
143           s_posizione[3] = '\0';
144
145           /* se entrambe le stringhe sono numeriche eseguiamo il comando */
146           if(isInteger(s_numero_servo) && isInteger(s_posizione))
147           {
148              /* convertiamo le stringhe in numeri interi */
149              posizione = atoi(s_posizione);
150              numero_servo = atoi(s_numero_servo);
151              if(posizione<MIN_POS) posizione = MIN_POS;
152              if(posizione>MAX_POS) posizione = MAX_POS;
153              if(numero_servo<0) numero_servo = 0;
154              if(numero_servo>3) numero_servo = 3;
155
156              /* reinizializziamo la posizione del servo */
157              posizioni_servo[numero_servo]=posizione;
158           }
159           else
160           {
161              Usart_Write_String(stringa_Nuova_linea);
162                 Usart_Write_String(stringa_Sintassierrata);
```

```
163           }
164
165           /* Messaggio di risposta */
166           Usart_Write_String(stringa_Nuova_linea);
167           Usart_Write_String(stringa_Posizione_cambiata);
168           printPrompt();
169         } /* FINE if(buffer[0]=='M' && strlen(buffer)==5) */
170
171
172         /* altrimenti, se la prima lettera è 'O' (ON) o 'o' (off) e la lunghezza
173         della stringa è 2 elabora il comando di accensione del servo */
174         else if ((buffer[0]=='O' || buffer[0]=='o') && strlen(buffer)==2)
175         {
176           char s_numero_servo [2]; /* stringa in cui copieremo il numero del servo */
177           int  numero_servo;  /* numero servo da accendere */
178
179           /* se la stringa è numerica */
180           if(isInteger(s_numero_servo))
181           {
182              /* Convertiamo la stringa in valore numerico */
183              numero_servo = atoi(s_numero_servo);
184
185              /* Se si vuole accendere il servo */
186              if(buffer[0]=='O')
187              {
188                   /* elaboriamo il comando */
189                   switch (numero_servo)
190                      {
191                      case 0:  ACCENSIONE_SERVO0=SI;
192                               break;
193                      case 1:  ACCENSIONE_SERVO1=SI;
194                               break;
195                      case 2:  ACCENSIONE_SERVO2=SI;
196                               break;
197                      case 3:  ACCENSIONE_SERVO3=SI;
198                               break;
199                      default: Usart_Write_String(stringa_Sintassierrata);
200                               break;
201                      }
202              } /* FINE */
203
204              /* Altrimenti, se si vuole spegnere il servo */
205              else if(buffer[0]=='o')
206              {
207                   /* elaboriamo il comando */
208                   switch (numero_servo)
209                   {
210                      case 0:  ACCENSIONE_SERVO0=NO;
211                               break;
212                      case 1:  ACCENSIONE_SERVO1=NO;
213                               break;
214                      case 2:  ACCENSIONE_SERVO2=NO;
215                               break;
216                      case 3:  ACCENSIONE_SERVO3=NO;
217                               break;
218                      default: Usart_Write_String(stringa_Sintassierrata);
```

```
                                break;
                    }
                } /* FINE */

                Usart_Write_String(stringa_Nuova_linea);
                Usart_Write_String(stringa_StatoCambiato);
                printPrompt();
              }
            } /* FINE else if ((buffer[0]=='O' || buffer[0]=='o') && strlen(buffer)==2) */

            /* alternativamente il comando è sconosciuto */
            else
            {
                Usart_Write_String(stringa_Nuova_linea);
                Usart_Write_String(stringa_Comando_Sconosciuto);
                printPrompt();
            }
         }
        /* FINE elaborazione dei comandi seriali */
        else //se la stringa è vuota annulla
        {
          cursore_buffer = 0;
        }
      } /* FINE while(Usart_Data_Ready()) */

    }


/* La funzione principale del firmware */
void main()
{
    int servo_corrente;      /* variabile cursore per la scansione dei servocomandi */
    int STEP;                /* La variabile che conterrà l'incremento di impulso per step */

    /* configurazione del microcontrollore */
    /* impostiamo la porta A e la portra B come porta di output */
    TRISA = 0;
    TRISB = 0;
    /* disabilitiamo i comparatori */
    CMCON = 0x07;

    /* inizializzazione di due variabili che verranno impiegate nel blocco assembly */
    TEMP_REG=0;
    ASM_DELAY=0;

    /* abilitiamo gli interrupt generali e l'interrupt di ricezione di un byte
       sulla linea seriale */
    INTCON.GIE = 1;
    PIE1.RCIE = 1;
    INTCON.PEIE = 1;

    /* Inizializzazione del modulo USART */
    Usart_Init(9600);

    /* centriamo i servocomandi sulla base degli impulsi massimi e minimi */
```