LE FUNZIONI BASE (prima parte)

In questo fascicolo introduciamo la programmazione di RZB-1p, presentando la prima parte delle funzioni di base che potrai utilizzare per la scrittura dei firmware dedicati al tuo robot.

In chiusura del precedente Workshop abbiamo presentato l'elenco delle procedure elementari che regoleranno il funzionamento del robot, dividendole concettualmente in **tre distinte categorie** (funzioni di sistema, di movimento e di I/O). Passiamo ora ad analizzare i sorgenti che definiscono tali funzioni e che ci permetteranno di creare un template di base per la programmazione di RZB-1p. Nelle pagine che seguono proponiamo il codice mikroC, intervallando tratti di programma a commenti e note di utilizzo.

LA DEFINIZIONE BASE ›››

La prima parte del template di programmazione è costituita dalla **definizione dei valori simbolici** che potranno essere impiegati all'interno del firmware. Come già avvenuto con esempi contenuti in alcuni Workshop precedenti, **tale sezione del sorgente sarà ottenuta sfruttando una sequenza di direttive '#define'**, che ci permetterà di associare ai vari simboli mnemonici una serie di corrispondenti valori numerici. L'impiego di tali simboli ci consentirà di facilitare la lettura e la comprensione dei programmi.

Le definizioni

```
/* Definizione dei simboli legati alla direzione di movimento dei motori (definiti
in modo concorde alla direzione di movimento del robot). */
#define AVANTI 0
#define INDIETRO 1


/* Definizione dei pin di controllo del motore destro. La linea di attivazione è
collegata al pin 0 della porta B, quella di controllo al pin 1 della medesima porta. In fase
di compilazione i simboli verranno sostituiti con i rispettivi collegamenti
ai bit dei registri di porta. */
#define LINEA_ATTIVAZIONE_DX PORTB.F0
#define LINEA_CONTROLLO_DX PORTB.F1

/* Definizione dei pin di controllo del motore sinistro. La linea di attivazione è collegata
al pin 2 della porta B, quella di controllo al pin 3 della medesima porta. In fase di compi-
lazione i simboli verranno sostituiti con i rispettivi collegamenti
ai bit dei registri di porta. */
#define LINEA_ATTIVAZIONE_SX PORTB.F2
#define LINEA_CONTROLLO_SX PORTB.F3

/* Definizione del valore di inizializzazione del timer 0. Con il valore 115 e
il prescaler impostato ad un fattore di riduzione di 1:256 otteniamo una frequenza
di chiamata degli interrupt di circa 55 Hz (adatta al controllo dei servocomandi). */
#define TIMER0_START 115
```

```
/* Definizione della durata degli impulsi di movimento dei servocomandi, espressa in micro-
secondi. Poiché i servocomandi assumono la posizione di centratura con impulsi di circa
1500 microsecondi di durata, un impulso maggiore di tale valore manterrà.  */
#define IMPULSO_LUNGO 1800
#define IMPULSO_BREVE 1200

/* Definizioni simboliche dei pin di collegamento ai cinque sensori di posizione. */
#define SENSORE_EST PORTA.F0
#define SENSORE_NORDEST PORTA.F1
#define SENSORE_NORD PORTA.F2
#define SENSORE_NORDOVEST PORTA.F3
#define SENSORE_OVEST PORTA.F4


/* Definizioni simboliche dei pin di input del PIC connessi al connettore flat
a 10 poli. Tale definizione è analoga alla precedente, cambiano solo i nomi di
riferimento, ora più generici. */
#define INPUT_1 PORTA.F0
#define INPUT_2 PORTA.F1
#define INPUT_3 PORTA.F2
#define INPUT_4 PORTA.F3
#define INPUT_5 PORTA.F4

/* Definizioni dei simboli ON e OFF. */
#define ON 1
#define OFF 0

/* Definizioni dei simboli SI e NO. */
#define SI 1
#define NO 0

/* definizione dei simboli ACCESO e SPENTO. */
#define ACCESO 1
#define SPENTO 0

/* definizione delle porte di collegamento alle linee di controllo del LED rosso e del LED
verde. */
#define LED_ROSSO PORTB.F4
#define LED_VERDE PORTB.F5

/* definizione dei simboli DESTRO, SINISTRO e ENTRAMBI. */
#define DESTRO 0
#define SINISTRO 1
#define ENTRAMBI 2
```

Le definizioni mostrate costituiscono le prime righe del sorgente del nostro firmware e hanno la funzione di definire alcuni elementi simbolici che garantiscono una maggiore leggibilità al programma. Per avere un'idea della loro efficacia è sufficiente confrontare l'istruzione 'LED_ROSSO = ACCESO;' con il comando equivalente 'PORTB.F4 = 1;'. Mentre nel primo caso è chiaro l'effetto dell'operazione anche a 'livello utente', nel secondo l'istruzione non dà informazioni 'concrete' sui suoi effetti, al di là del fatto che venga portato 'alto' il bit 4 della porta B (per capire cosa accade realmente dovremmo mettere in relazione il pin RB4 al circuito in cui è inserito il microcontrollore).

LE VARIABILI DI STATO

Per il corretto funzionamento del robot utilizzeremo una coppia di variabili intere (di tipo int) chiamate 'direzione_DX' e 'direzione_SX' in cui memorizzeremo, di volta in volta, lo stato di movimento

dei due servocomandi.

```
int direzione_DX;
int direzione_SX;
```

LA FUNZIONE DI RISPOSTA ALL'INTERRUPT ›››

Per poter pilotare correttamente i servocomandi il firmware del nostro robot deve generare due sequenze approssimativamente periodiche di impulsi sulle linee di controllo dei motori (pin RB1 e RB3). Tale operazione avviene sfruttando opportunamente, come fatto negli esperimenti precedenti, le chiamate di interrupt. È necessario, quindi, implementare la funzione 'interrupt'. Come puoi notare dal codice sorgente presentato nel box sottostante, il funzionamento dei blocchi che generano l'impulso di controllo dei motori destro e sinistro è complementare (nel destro l'avanzamento è associato all'impulso lungo, mentre nel sinistro viene usato l'impulso breve). Ciò dipende dal fatto che i due motori sono montati sul telaio del robot in modo opposto; per avere versi di rotazione concordi rispetto al robot, quindi, dobbiamo imporre movimenti reciproci dei due alberi, che si ottengono inviando sulle linee di controllo due treni di impulsi di durata differente.

La funzione interrupt

```
/* Funzione di risposta agli interrupt */
void interrupt()
{

 /* se il bit T0IF del registro INTCON è '1', la richiesta di interrupt
 proviene dal timer 0 */
  if (INTCON.T0IF == 1)
  {
        /* blocco di codice che genera l'impulso di controllo del motore destro */
        /* Se la direzione di movimento del motore destro è 'AVANTI' */
        if (direzione_DX == AVANTI)
        {
            /* Genera un impulso lungo sulla linea di controllo del motore destro */
            LINEA_CONTROLLO_DX = 1;
            Delay_us(IMPULSO_BREVE);
            LINEA_CONTROLLO_DX = 0;
        }
        else
        {
            /* Altrimenti, genera un impulso breve sulla linea di controllo del
             motore destro */
            LINEA_CONTROLLO_DX = 1;
            Delay_us(IMPULSO_LUNGO);
            LINEA_CONTROLLO_DX = 0;
        }
      /* blocco di codice che genera l'impulso di controllo del motore destro */
      /* Se la direzione di movimento del motore sinistro è 'AVANTI' */
       if (direzione_SX == AVANTI)
        {
            /* Genera un impulso breve sulla linea di controllo del motore
             sinistro */
            LINEA_CONTROLLO_SX = 1;
            Delay_us(IMPULSO_LUNGO);
            LINEA_CONTROLLO_SX = 0;
        }
        else
        {
            /* Genera un impulso lungo sulla linea di controllo del motore
             sinistro */
            LINEA_CONTROLLO_SX = 1;
```

```
            Delay_us(IMPULSO_BREVE);
            LINEA_CONTROLLO_SX = 0;
        }

    /* Termina la risposta all'interrupt del timer 0, resettando il bit T0IF
       e il reimpostando il timer0 */
       INTCON.T0IF = 0;
       TMR0 = TIMER0_START;
       }
}
```

LE FUNZIONI DI SISTEMA ››› Vediamo ora le due **funzioni di sistema**, che agiscono sulla configurazione del PIC. Le prime delle due è la funzione **'InizializzaRobot()'**, che permette di racchiudere le istruzioni necessarie a **configurare il microcontrollore del robot** (impostazione del timer 0, abilitazione degli interrupt ecc.). **La seconda funzione di sistema si occupa, invece, di resettare il robot.** Il reset avviene in modo molto semplice per mezzo della chiamata alla funzione di sistema InizializzaRobot().

Le funzioni di sistema

```
/* La funzione InizializzaRobot() predispone il PIC per il funzionamento del robot.
Da chiamare all'avvio della funzione 'main'. */
void InizializzaRobot()
{
    /* impostiamo il TIMER0 a 255 in modo da generare immediatamente il primo impulso
       di controllo */
    TMR0 = 255;

    /* inizializzazione della porta B come output */
    TRISB = 0x0;

    /* inizializzazione della porta A come input e disattivazione dei comparatori*/
    TRISA = 0b11111111;
    CMCON = 0x07;

    /* Impostiamo il prescaler all'uso con il timer 0 con un fattore di riduzione
       di 1:256 */
    OPTION_REG = 0b10000111;

    /* Attiviamo gli interrupt globali e l'interrupt di overflow del timer 0*/
    INTCON.GIE = 1;
    INTCON.T0IE = 1;

    /* Disattiviamo i motori del robot e i LED e resettiamo le variabili di stato */
    LINEA_ATTIVAZIONE_DX = SPENTO;
    LINEA_ATTIVAZIONE_SX = SPENTO;
    LED_ROSSO = SPENTO;
    LED_VERDE = SPENTO;
    direzione_DX = AVANTI;
    direzione_SX = AVANTI;
}

/* La funzione Reset() riavvia il firmware rieseguendo l'inizializzazione */
void Reset()
{
   InizializzaRobot();
}
```

LE FUNZIONI DI MOVIMENTO ›››

Nel box sottostante trovi descritte e implementate le funzioni elementari utili alla gestione dei motori del robot. In questa porzione di codice, in particolare, puoi trovare le funzioni necessarie ad accendere e spegnere i singoli servocomandi e a impostare la loro direzione di movimento. Da notare che **l'attuazione dei motori non avviene agendo sulle linee di controllo dei servo, ma semplicemente sulle due variabili di stato del firmware.**

La funzioni di movimento

```
/* Le funzioni AccendiMotori e SpegniMotori consentono di accendere o spegnere il motore
indicato come parametro. Il parametro può valere DESTRO, SINISTRO o ENTRAMBI, in base al
servo che si vuole attivare o disattivare */

/* Accende il motore indicato come parametro */
void AccendiMotori(int motore)
{
      if (motore == DESTRO)
       {
          LINEA_ATTIVAZIONE_DX = ACCESO;
       }
      else if   (motore == SINISTRO)
       {
          LINEA_ATTIVAZIONE_SX = ACCESO;
        }
      else if (motore == ENTRAMBI)
        {
          LINEA_ATTIVAZIONE_DX = ACCESO;
          LINEA_ATTIVAZIONE_SX = ACCESO;
         }
}


/* Spegne il motore indicato come parametro */
void SpegniMotori(int motore)
{
      if (motore == DESTRO)
       {
          LINEA_ATTIVAZIONE_DX = SPENTO;
       }
      else if   (motore == SINISTRO)
       {
          LINEA_ATTIVAZIONE_SX = SPENTO;
       }
      else if (motore == ENTRAMBI)
       {
          LINEA_ATTIVAZIONE_DX = SPENTO;
          LINEA_ATTIVAZIONE_SX = SPENTO;
       }
}



/* la funzione DirezioneMotore permette di impostare la direzione di rotazione (AVANTI o
INDIETRO) del motore indicato nei parametri (DESTRO, SINISTRO) */

void DirezioneMotore(int motore, int direzione)
{
     /* impostazione della direzione del motore destro */
      if (motore == DESTRO)
      {
```

```
            /* se la direzione è valida */
            if(direzione == AVANTI || direzione == INDIETRO)
            {
                direzione_DX = direzione;
            }
        }
        else
        /* impostazione della direzione del motore sinistro */
        if (motore == SINISTRO)
        {
            /* se la direzione è valida */
            if(direzione == AVANTI || direzione == INDIETRO)
            {
                direzione_SX = direzione;
            }
        }
}
```

Passiamo infine alle **funzioni di movimento di livello più alto** (mostrate nel box di codice sottostante), che **agiscono su entrambi i motori per controllare il movimento del robot**. Nel prossimo fascicolo proseguiremo l'analisi delle funzioni di base per la programmazione di RZB-1p, analizzando il sorgente delle funzioni di I/O e un primo semplice esempio scritto in mikroC, con cui testeremo il funzionamento della scheda PIC. Ti ricordiamo, in ultimo, che **tutto il codice presentato fa parte di un unico sorgente.**

Le funzioni di movimento avanzate

```
/* La funzione Avanza configura il sistema per l'avanzamento del robot */
void Avanza(void)
{
    direzione_DX = AVANZA;
    direzione_SX = AVANZA;
}

/* La funzione Retrocedi configura il sistema per fare indietreggiare il robot */
void Retrocedi(void)
{
    direzione_DX = INDIETRO;
    direzione_SX = INDIETRO;
}

/* La funzione RuotaDestra configura il sistema per far ruotare a destra il robot */
void RuotaDestra(void)
{
    direzione_DX = INDIETRO;
    direzione_SX = AVANTI;
}


/* La funzione RuotaSinistra configura il sistema per far ruotare a sinistra
   il robot */
void RuotaSinistra(void)
{
    direzione_DX = AVANTI;
    direzione_SX = INDIETRO;
}
```