ANALIZZIAMO IL FIRMWARE

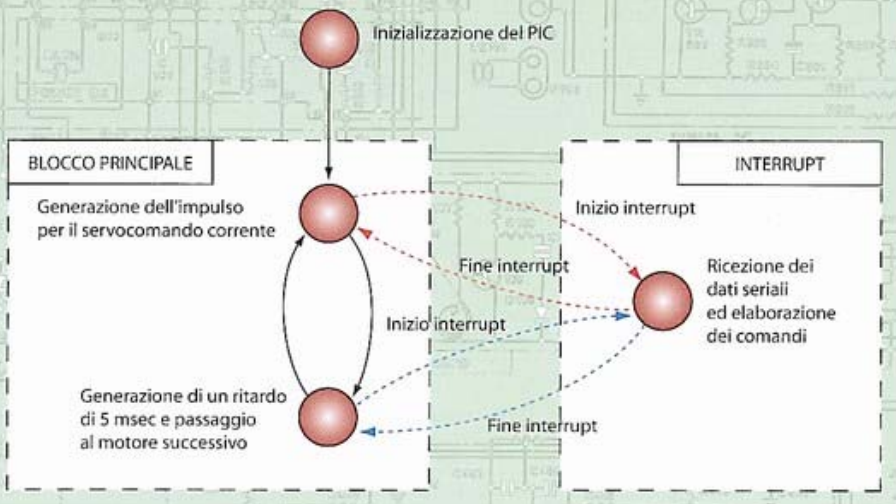
Ora che abbiamo analizzato la struttura hardware della mini scheda di controllo per servocomandi che stiamo progettando, è giunto il momento di capire come opera il firmware di test, di cui abbiamo a disposizione l'intero sorgente.

Quello mostrato nei Workshop 77 e 78 è il primo firmware di una certa complessità che abbiamo presentato in questa raccolta (ben 10 pagine di sorgente). Come è facile immaginare, quando si creano firmware per applicazioni concrete la mole di codice richiesta è quasi sempre notevole ed è quindi normale ricorrere a una grande quantità di righe di programma (basti pensare che il sorgente presentato offre solo le operazioni essenziali per la gestione dei servocomandi). Ma torniamo al nostro firmware e soprattutto iniziamo a capire i suoi principi di funzionamento. Esso può essere, innanzitutto, suddiviso in quattro blocchi (vedi schema sotto), che possiamo chiamare 'Inizializzazione', 'Generazione dell'impulso', 'Generazione del ritardo' e 'Interrupt'.

LA FASE DI INIZIALIZZAZIONE

Questa prima porzione del programma, si occupa, ovviamente, di configurare l'hardware del PIC e le variabili di sistema (righe #250-290). La configurazione inizia, come già accaduto in alcuni miniprogetti precedenti, con l'impostazione della 'direzione' delle porte A e B (in uscita) e la disattivazione dei componenti analogici della porta A (righe

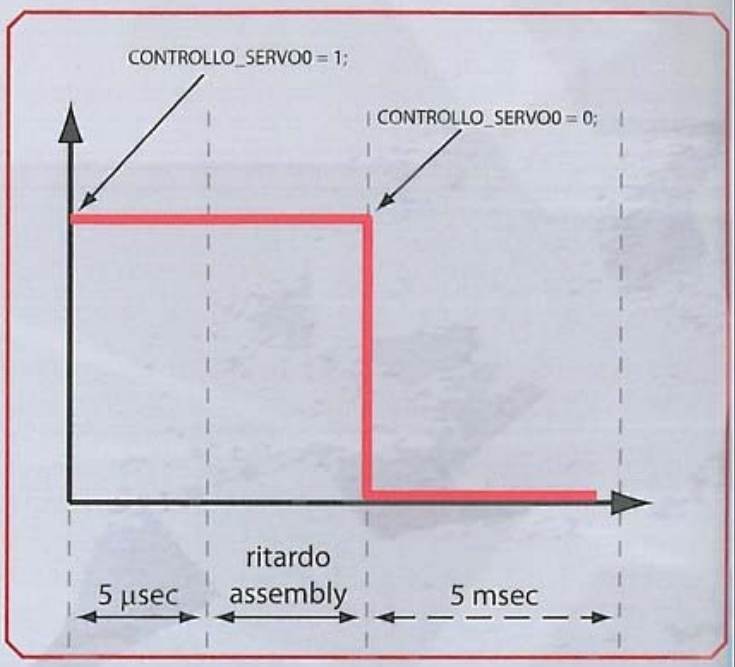
STRUTTURA DEL FIRMWARE



Il programma può essere suddiviso in quattro fasi specifiche: dopo la prima fase di inizializzazione del microcontrollore e delle variabili di sistema, il firmware entra in un ciclo infinito nel quale vi è un susseguirsi sequenziale di generazioni di impulsi e di pause (in modo da produrre una frequenza di invio degli impulsi che si aggira attorno ai 20 Hz). Ogni iterazione genera un impulso per un singolo servocomando: per inviare impulsi a tutti e quattro i servo, quindi, sono necessarie quattro iterazioni distinte. Nel caso in cui il modulo USART del PIC riceva dati dalla linea seriale, infine, viene invocato un interrupt, che congela lo stato del firmware e procede all'elaborazione dei byte ricevuti. Terminata l'interruzione, il programma riprende la sua esecuzione da dove era stata sospesa.

#255-258). In tal modo i piedini associati alle due porte potranno essere utilizzati per **emettere i segnali logici necessari al controllo e all'attivazione dei servocomandi.** Successivamente, dopo aver inizializzato le variabili TEMP_REG e ASM_DELAY (#261, #262), si procede con **l'abilitazione degli interrupt** (#266-268). In particolare, in queste tre righe vengono **impostati altri tre bit specifici.** Il primo è, ovviamente, il bit **GIE (Global Interrupt Enable) del registro INTCON,** che come sappiamo costituisce una sorta di 'interruttore generale' degli interrupt del microcontrollore. La novità è costituita, invece, dalle due righe successive (#267 e #268) nelle quali vengono **portati a 1 i bit PEIE (registro INTCON) e RCIE (registro PIE1).** Qual è la loro funzione? **Il bit PEIE** è un altro bit di attivazione, che **abilita la risposta agli interrupt generati dalle periferiche interne del microcontrollore. Il bit RCIE** (Receive Enable), invece è il bit che **consente di attivare la generazione di chiamate di interrupt da parte del modulo USART ogniqualvolta venga ricevuto un byte** attraverso la linea seriale. Grazie a questo interrupt, potremo rispondere in tempo reale alla trasmissione di comandi da parte dell'utente, ma di questo ci occuperemo nei paragrafi successivi. Terminata la configurazione delle interruzioni si passa alla **inizializzazione del modulo USART tramite la funzione Usart_Init di mikroC** (#271). Il ciclo 'for' che segue (#274-277) ha la scopo di **inizializzare le posizioni iniziali dei quattro servocomandi,** assegnando il valore 128 (corrispondente alla posizione centrale) alle quattro celle dell'**array 'posizioni_servo'** (#276). Nelle righe successive si **passa all'attivazione dei motori** (#280-283), ossia all'impostazione di uscite alte sulle linee che portano ai pin di gate dei transistor di accensione. Nel caso in cui volessi i motori disattivati nel momento in cui viene alimentata la scheda, ti sarà sufficiente agire sulle quattro righe appena indicate, modificando le istruzioni ACCENSIONE_SERVOx=SI; in ACCENSIONE_SERVOx=NO;. Le istruzioni successive (#286 e #290), infine, concludono la fase di inizializzazione del programma.

La generazione dell'impulso avviene sfruttando un ritardo variabile prodotto con un blocco di codice assembly.



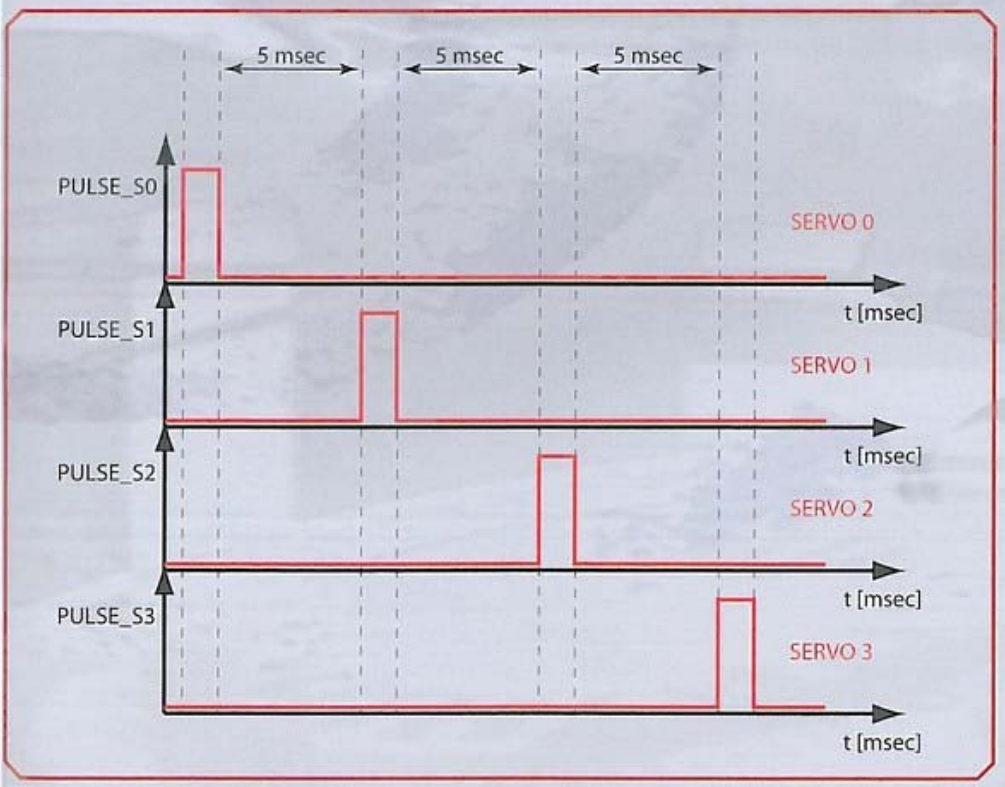
LA GENERAZIONE DEGLI IMPULSI

Passiamo ora al ciclo di vita del programma vero e proprio, che ha inizio dalla riga #293 con la dichiarazione del costrutto ciclico infinito 'while(1)'. All'interno di esso trovi una ramificazione di blocchi di codice 'if/else' effettuata rispetto al valore della **variabile 'servo_corrente'.** Questo dipende dal fatto che **a ogni esecuzione del blocco 'while' viene inviato verso l'esterno un solo impulso sulla linea dedicata,** impulso che viene seguito da una **pausa di circa 5 ms** (vedi schema in basso nella pagina precedente). **Ogni ciclo produce, di conseguenza, un singolo impulso di controllo per un ben preciso motore,** ottenendo in questo modo una serie di segnali 'sfasati' l'uno rispetto all'altro con un ritardo praticamente costante (vedi schema nella pagina successiva). Ma passiamo a osservare in che modo avviene la **generazione dell'impulso** di durata prescelta. Abbiamo già avuto modo di discutere in merito all'utilizzo della funzione **'Delay_us'** di mikroC e abbiamo già avuto modo di definirne i limiti pratici (deve ricevere un parametro costante e non è adatta per generare impulsi

di durata variabile). Serve quindi una 'strategia' alternativa, che ci consenta di **generare treni di impulsi con durate variabili tra gli 0,5 e i 2,5 msec.** La tecnica che utilizzeremo è in sé molto semplice (prenderemo come riferimento la generazione dell'impulso per il servocomando 0, descritta dalla riga #301 del codice e schematizzata nella pagina precedente). Innanzitutto **viene innalzato il livello elettrico della linea** (istruzione CONTROLLO_SERVO0=1;). Successivamente si **genera un ritardo di 5 microsecondi,** che corrispondono alla **durata minima** assumibile dall'impulso, dopo il quale **subentra il blocco di codice assembly compreso nella struttura asm{...}.**

Il blocco assembly implementa un **semplice ciclo,** che si svolge tra **l'etichetta 'INIZIO0'** e **l'istruzione di salto 'GOTO INIZIO0',** all'interno del quale vengono eseguite **istruzioni 'nulle' 'NOP',** utili a 'bruciare' cicli di clock in modo controllato. Il controllo sul ciclo avviene per mezzo dell'istruzione **'SUBWF ASM_DELAY'** (che decrementa la variabile 'ASM_DELAY' del valore contenuto nel registro W del PIC, precedentemente inizializzato a 1 con il comando 'MOVLW 0x01') e dall'istruzione di salto condizionato **'BTFSS'.** **Ogni iterazione del ciclo assembly presentato, equivale a un ritardo che si aggira attorno ai 5 µs, che viene ripetuto per 'ASM_DELAY' volte.** La programmazione in linguaggio assembly, come già detto, non è parte di quest'opera; tuttavia è utile poter vedere almeno un esempio pratico che ne illustri la struttura. Per una maggiore comprensione del significato delle istruzioni assembly (che sono fortemente legate alla struttura hardware del microcontrollore di riferimento), ti rimandiamo ai datasheet ufficiali. Terminato il blocco assembly, l'impulso viene abbassato con l'istruzione CONTROLLO_SERVO0=0;.

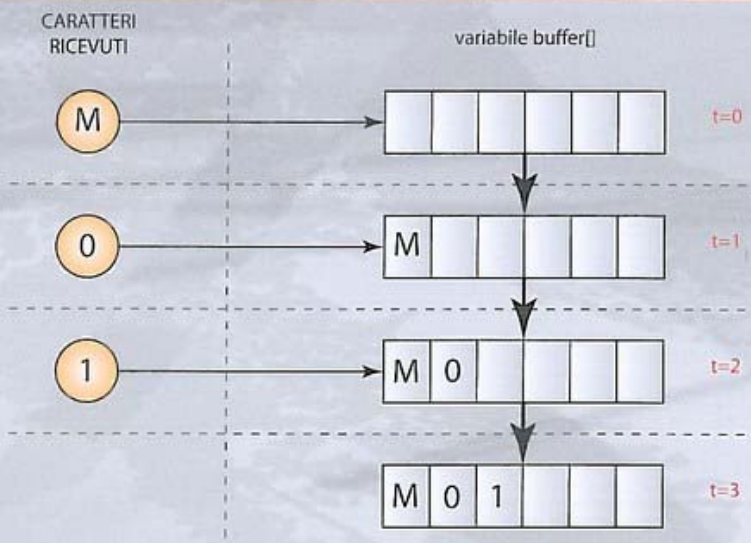
Il firmware caricato sul PIC 16F628 produce treni di impulsi sulle quattro linee dei pin RA0, RA1, RA2 e RA3. La struttura ciclica del programma fa sì che tra la fine di un impulso e l'inizio dell'impulso sulla linea successiva intercorrano circa 5 millisecondi.



L'ELABORAZIONE DEI COMANDI SERIALI

Passiamo ora a capire **in che modo la scheda interpreta i comandi ricevuti attraverso la linea seriale.** Innanzitutto premettiamo che la tecnica di elaborazione dei dati seriali utilizzata è estremamente semplice e non ha nulla a che vedere con gli algoritmi realmente impiegati all'interno dei software di interpretazione o compilazione delle istruzioni. Come sempre, comunque, l'obiettivo più importante è quello di offrire una base 'didattica' da cui partire per poter comprendere il funzionamento di dispositivi e firmware simili. Abbiamo già visto come nella porzione di sorgente legata alla configurazione del microcontrollore vengano abilitati gli interrupt della periferica seriale. Tutta **la gestione dei comandi avviene, di conseguenza, all'interno della funzione 'interrupt'** (righe #100-244). Essa inizia con la **memorizzazione del**

byte ricevuto nella variabile 'car_letto' (riga #105), in base al quale viene successivamente definita la risposta. Se il byte è diverso dal 'carriage return' (0x0D) e dal 'line feed' (0x0A), significa che l'utente sta digitando un comando. In tal caso viene prima di tutto generato l'effetto di echo (#111) e, successivamente, copiato il carattere nell'array di caratteri 'buffer', nella cella di posizione 'cursore_buffer', con successivo incremento. In questo modo non facciamo altro che 'comporre' una stringa con i caratteri ricevuti (vedi schema in basso a destra), archiviandoli in un vettore dedicato. Nel caso in cui venisse ricevuto uno dei due byte di terminazione elencati in precedenza si procederà all'elaborazione del comando, che ha inizio con la 'chiusura della stringa' con l'inserimento del carattere '\0' (riga #119) e con il 'reset' della variabile che funge da 'cursore' della stringa (#122). Si passa poi all'analisi della stringa di comando, che viene svolta in modo molto specifico e schematico, proprio per garantire la massima semplicità concettuale dell'algoritmo. Come visto nei fascicoli precedenti, i comandi supportati sono tutti costituiti da un numero ben preciso di caratteri disposti secondo sequenze schematiche. Ad esempio, il comando di movimento deve essere sempre formato da 5 caratteri, il primo dei quali è 'M' (maiuscolo), seguito da 4 cifre (la prima delle quali costituisce il servo da azionare, mentre la seconda rappresenta la posizione desiderata). Allo stesso modo, i comandi di accensione e spegnimento dei motori sono di due caratteri, il primo dei quali è 'O' (maiuscolo per l'accensione) o 'o' (minuscolo per lo spegnimento) seguito da una cifra che indica il servo su cui si vuole agire. Il processo di esecuzione dei comandi deve prevedere, quindi, prima di tutto la validazione del comando (per stabilire se è formattato correttamente) e solo successivamente l'esecuzione. Così, ad esempio, da riga #135 vengono copiati nei due vettori temporanei 's_numero_servo' e 's_posizione' i caratteri che devono indicare il numero del servocomando e la posizione finale, ottenendo così due stringhe. Le stringhe sono poi validate a riga #146 con l'uso della funzione 'isInteger', che è stata creata per stabilire se la stringa ricevuta per parametro rappresenta un numero intero positivo o meno. Nel caso in cui entrambe le variabili rispettino i vincoli strutturali, si procede con la chiamata della funzione 'atoi' (già definita in mikroC, righe #149 e #150), che converte una stringa contenente un numero intero in un valore di tipo 'int'. Stabilita la nuova posizione e identificato il motore, l'esecuzione del comando si conclude con l'assegnamento del nuovo valore al vettore 'posizioni_servo' (riga #137). Nel caso in cui il comando sia mal formattato a causa di valori numerici errati (righe #159-163), infine, il programma invierà un messaggio di errore. Dalla riga numero 174 ha inizio il blocco di risposta ai due comandi 'o' e 'O', molto simile a quanto appena visto. Per questa ragione non ci soffermeremo a commentare anche tali righe di codice. Non ci resta che collaudare la nostra scheda.

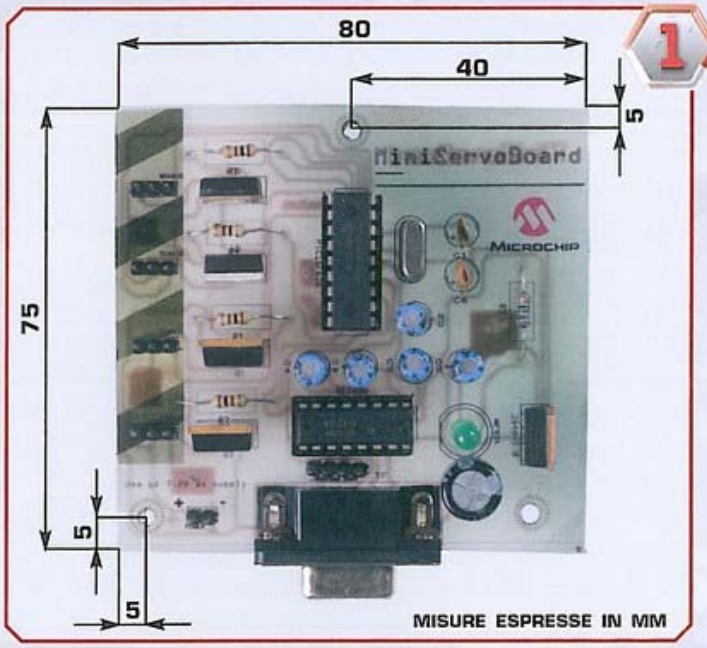


Ogni volta che il modulo USART riceve un byte tramite linea seriale, questo byte viene memorizzato in modo incrementale nelle celle dell'array 'buffer'. Se il carattere ricevuto è un carattere di 'chiusura' (nuova linea o carriage return) il sistema considera 'concluso' il comando e passa all'interpretazione.

STEPbySTEP

COLLAUDIAMO LA MINI SCHEDA DI CONTROLLO PER I SERVO

Vediamo come collaudare il funzionamento della mini scheda di controllo per servocomandi presentata nei fascicoli precedenti.

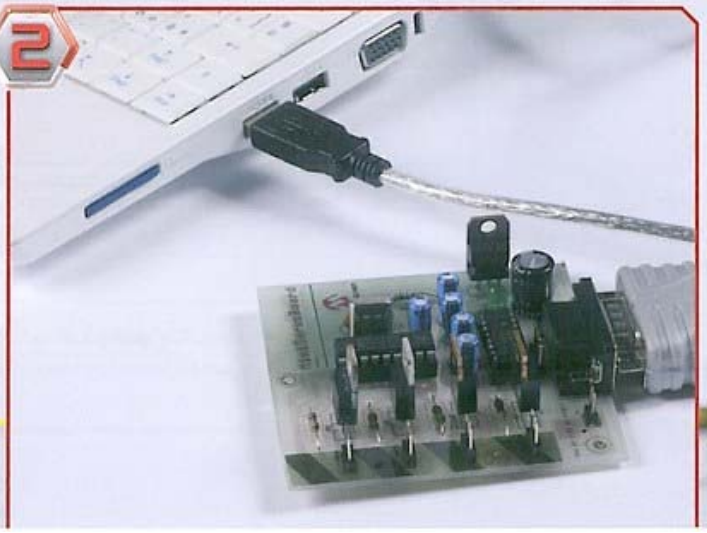


1 Inizia realizzando la scheda di controllo, per la quale avrai bisogno dei componenti elencati qui di seguito. Se realizzerai la scheda secondo le misure mostrate nella foto, potrai installare questo nuovo hardware sul telaio di RZB-1, per espanderne ulteriormente le funzioni.

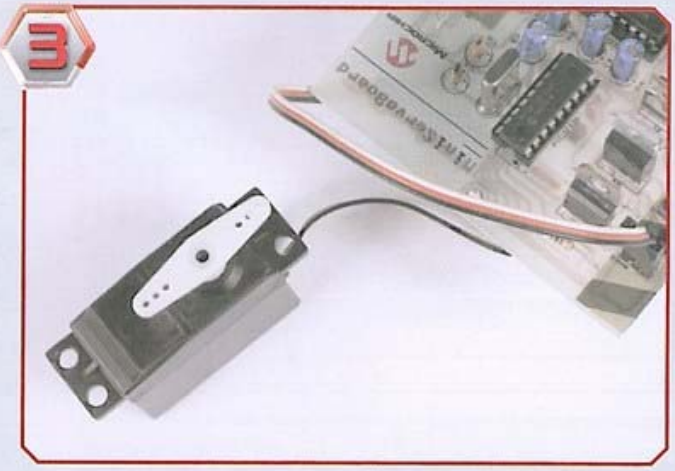
Elenco componenti richiesti:

- 1 PIC 16F628 con precaricato il firmware mostrato nei fascicoli precedenti + SOCKET 18 pin
- 1 MAX232 + SOCKET 16 pin
- 5 Condensatori 1 μF
- 1 Condensatore 220 μF
- 1 2940-5,0
- 4 IRF540
- 4 Resistori 100 W
- 1 Resistore 220 W
- 1 LED
- 2 Condensatori da 22 pF
- 1 Quarzo 20 MHz
- 1 Connettore Cannon DB-9
- Strip line maschi

2 Una volta realizzata la scheda, collegala alla porta seriale del computer. Nel caso in cui il tuo computer non disponesse di una porta seriale puoi ricorrere a un adattatore USB-RS232.

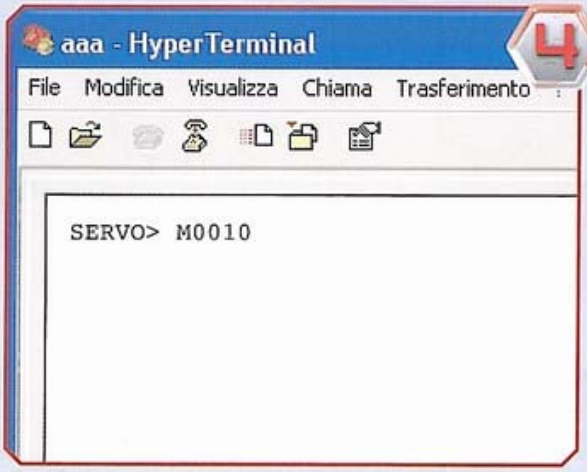


3



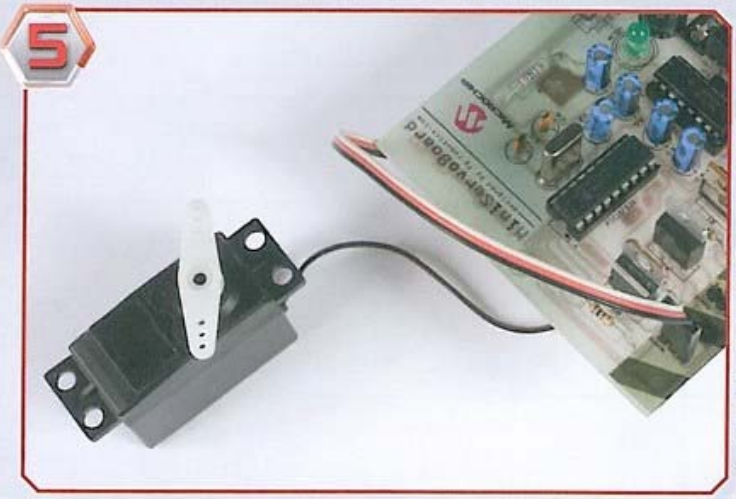
Collega i servocomandi ai connettori della scheda. Ricorda di rispettare il verso di inserimento degli spinotti dei motori. Avvia il programma di emulazione del terminale seriale (come Hyperterminal o PuTty) e attiva la connessione con velocità 9600, come visto nei Workshop precedenti. Alimenta la scheda con un pacco batterie con voltaggio di almeno 6 V.

4



Una volta alimentata la scheda, il servocomando si porterà in posizione centrata per effetto degli impulsi generati dal PIC. Puoi ora provare ad azionare i servocomandi digitando nel software di emulazione terminale il comando di movimento, mostrato nella prima pagina del fascicolo 78. Digita, ad esempio, il comando M0010 (ossia 'muovi il servo 0 in posizione 10') e vedrai il servo collegato al connettore numero 0 ruotare fino a raggiungere la posizione associata al valore 10. Ricorda che il comando di movimento ha una struttura fissa formata dal carattere 'M' seguito da una cifra indicante il servo da azionare e da altre tre cifre corrispondenti alla posizione desiderata.

5



Prova ora a utilizzare i comandi per l'accensione e lo spegnimento del motore. Per spegnere il servo 0, ad esempio, invia il comando 'o0' ('o' minuscola seguita dalla cifra '0') e il motore 0 si spegnerà (te ne accorgerai in quanto il suo albero non opporrà resistenza a una torsione manuale). Invia successivamente 'O0' ('o' maiuscola seguita dalla cifra '0') per riattivare il motore (lo vedrai raggiungere nuovamente la posizione impostata prima dello spegnimento).