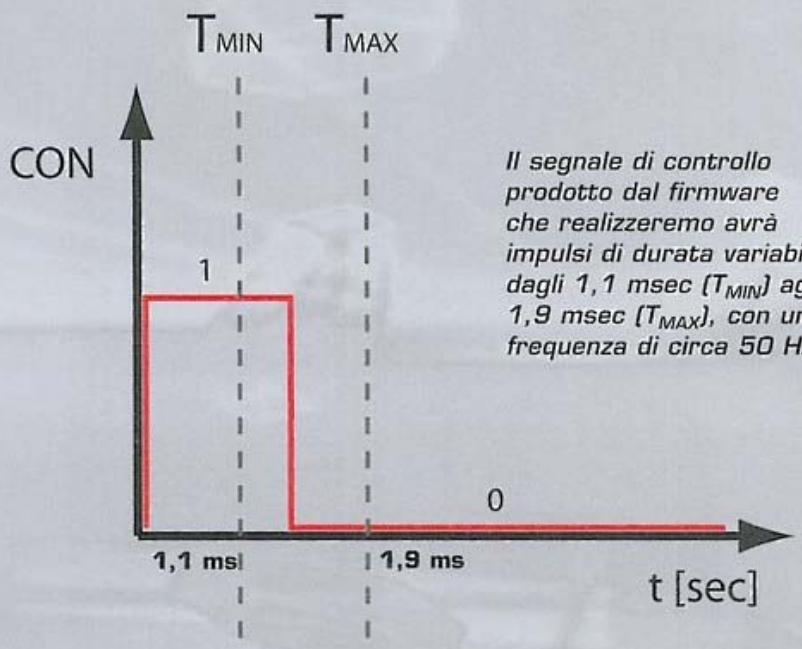
# CONTROLLARE UN SERVO CON UN PIC

*In questo Workshop realizziamo una semplice console digitale di controllo per servocomandi analogici, utile per testare i servo.*

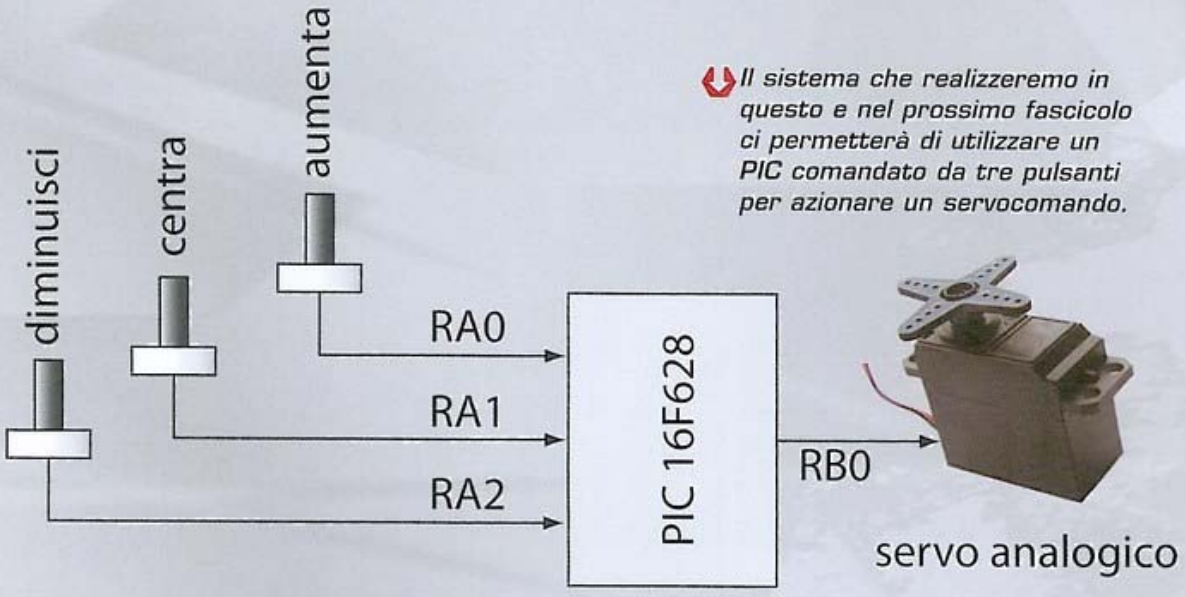
Il funzionamento teorico dei servocomandi è già stato introdotto nel corso del fascicolo 11, dove abbiamo visto a grandi linee la struttura interna di questi attuatori. In quel Workshop abbiamo anche visto come la posizione di tali sistemi elettromeccanici sia controllabile attraverso un treno di impulsi la cui durata è associata all'angolo desiderato. In questo Workshop realizzeremo **un semplice firmware che ti permetterà di controllare un comune servo analogico da modellismo.** Tieni comunque presente che l'esempio proposto è un sistema di generazione degli impulsi molto semplificato e non rappresenta certamente la soluzione ideale per realizzare sistemi multiservo, simili alla scheda di controllo di RoboZak.




*Il segnale di controllo prodotto dal firmware che realizzeremo avrà impulsi di durata variabile dagli 1,1 msec ($T_{MIN}$) agli 1,9 msec ($T_{MAX}$), con una frequenza di circa 50 Hz.*

## CONTROLLARE IL SERVOCOMANDO ›››

Prima di passare ad analizzare il circuito che utilizzeremo nell'esperimento, dedichiamo qualche riga a ricordare come operano i servocomandi. Come abbiamo già visto, i servo sono dotati di un cavo a tre poli, due dei quali utilizzati per **l'alimentazione del motore** e uno impiegato come **linea di controllo** su cui vengono trasmessi gli impulsi. Ricordiamo, in particolare, che **il treno di impulsi deve essere 'quasi periodico'**, nel senso che, sebbene non venga richiesta la regolarità temporale tipica della codifica PWM, affinché il motore mantenga costantemente la sua posizione senza 'perdere rigidità' **è indispensabile che due impulsi consecutivi non siano né troppo distanziati, né troppo ravvicinati (parlando in termini temporali).** Normalmente **si considera accettabile una frequenza di circa 50 Hz,** ossia un segnale composto da circa 50 impulsi al secondo. Proprio tale frequenza sarà il riferimento per la realizzazione del nostro progetto, mentre considereremo




*Il sistema che realizzeremo in questo e nel prossimo fascicolo ci permetterà di utilizzare un PIC comandato da tre pulsanti per azionare un servocomando.*

validi **impulsi di durata compresa tra gli 1,1 msec e gli 1,9 msec** (vedi schema nella pagina precedente). **La durata degli impulsi potrà essere variata dall'utente per mezzo di semplici pulsanti che avranno funzione di dispositivi di input per il circuito.**

## IL CIRCUITO ELETTRONICO ›››

Il circuito elettronico di questo Workshop ti permetterà di costruire un servotester con cui potrai collaudare semplici servo analogici. Analizziamo il funzionamento dell'elettronica che dovremo realizzare. **Il cuore di tutto il sistema è, come nei fascicoli precedenti, il PIC 16F628,** configurato in modo da fornire **tre ingressi logici** (nel nostro caso i pin 0, 1 e 2 della porta A) e **un'uscita digitale** (pin 0 della porta B). **I tre ingressi saranno collegati ad altrettanti pulsanti di input** attraverso i quali potrai effettuare tre operazioni di controllo del motore, ossia **l'avanzamento, l'arretramento e il reset in posizione centrata.** Più nel dettaglio utilizzeremo il pin **RA0 per aumentare** la durata dell'impulso di 100 µsec, il pin **RA2 per diminuire** la durata dell'impulso di 100 µs e il pin **RA1 per centrare il motore** (tale operazione equivale a impostare la durata dell'impulso al valore intermedio tra la durata massima e quella minima previste). **L'uscita digitale RB0, invece, dovrà essere collegata al pin di controllo del servocomando** e verrà utilizzata per **inviare il treno di impulsi** generato dal PIC (vedi schema in alto). Come fatto finora, infine, l'elettronica verrà alimentata facendo ricorso al **regolatore di tensione 2940.** In questo esperimento vedrai un classico esempio di **alimentazione del circuito con differenti valori di tensione:** mentre il PIC opererà sfruttando i 5 V regolati dal 2940, il servocomando verrà alimentato direttamente dal pacco batterie (tensione > 6 V).

## LA STRUTTURA DEL FIRMWARE ›››

Il firmware del nostro progetto potrà essere strutturato **in tre fasi sequenziali distinte.** La prima sarà dedicata alla **generazione dell'impulso di controllo** per il servo, **la seconda all'elaborazione dello stato dei pulsanti** e **la terza all'elaborazione di un ritardo** che preceda l'emissione dell'impulso successivo. Gli impulsi saranno prodotti in maniera molto semplice, ricorrendo all'uso della **funzione 'Delay_us',** messa a disposizione dall'ambiente mikroC, che consente di **generare un ritardo nell'esecuzione del codice di una quantità di microsecondi pari al valore intero passato come parametro.** Attenzione, però, a un particolare: la Delay_us non è una funzione come quelle viste finora, ma è di tipo **'inline'.** Ma in cosa differiscono le procedure inline da quelle viste finora? Quando abbiamo introdotto il concetto di 'funzione' nel corso del

Workshop abbiamo visto come la definizione di questi costrutti permetta di 'compattare il codice' mediante la definizione di istruzioni via via più complesse. Abbiamo anche visto che, affinché una funzione possa essere eseguita, essa deve essere **'chiamata'**, generando una serie di meccanismi elettronici che pongono il programma principale in una condizione di **'congelamento'** e che richiedono il salvataggio dei cosiddetti **punti di ritorno. Ogni volta che utilizziamo una funzione, di conseguenza, introduciamo in modo più o meno inconsapevole un ritardo di esecuzione, che può comportare inconvenienti non sempre controllabili in modo semplice,** specialmente in **applicazioni sensibili alla temporizzazione.** La definizione di blocchi di codice inline, invece, è una tecnica particolare che consente di **godere dei vantaggi di leggibilità offerti dall'uso delle funzioni uniti all'efficienza del codice sequenziale.** La tecnica è molto semplice: in fase di scrittura le procedure inline possono essere impiegate esattamente come normalissime funzioni, ma **nel momento in cui il sorgente viene compilato, al posto della chiamata di funzione viene inserito direttamente il codice macchina corrispondente alla funzione stessa** (il codice viene, quindi, replicato con il conseguente aumento delle dimensioni dell'eseguibile). La funzione 'Delay_us', inoltre, ha un'altra caratteristica: **richiede un parametro intero costante.** Il nostro ritardo verrà quindi prodotto con un uso ripetuto di tale funzione, che impiegheremo per generare **singoli passi di 100 µsec.** Vediamo come utilizzare questa funzione per il nostro scopo. La fase di generazione dell'impulso sarà composta da **tre step successivi.** Come prima cosa **viene innalzato il livello logico del pin RB0 per dare inizio all'impulso.** Successivamente è necessario **ripetere mediante un ciclo 'for' la funzione inline Delay_us(100); in modo da mantenere alto l'impulso per il periodo desiderato.** Terminata tale fase, **riporteremo bassa l'uscita RB0 per terminare l'invio dell'impulso.** Sebbene tale soluzione non sia la migliore da utilizzare in applicazioni 'reali', per il momento è quella più semplice e adatta ai nostri scopi. Un discorso più particolareggiato deve essere invece fatto per ciò che riguarda la gestione dei pulsanti di controllo del servocomando.

## LA GESTIONE DEI PULSANTI

Nell'esperimento precedente abbiamo visto un metodo di rilevamento del click basato su una struttura 'if' e un ciclo 'while' nidificati, nel quale il condizionale **'if' era impiegato per rilevare la pressione del tasto, mentre il ciclo 'while' vuoto permetteva di rimanere in attesa del suo rilascio.** Tale soluzione algoritmica crea, tuttavia, evidenti problemi nel caso in esame in questo fascicolo. Nei paragrafi

*La struttura ciclica del firmware che realizzeremo in questo fascicolo può essere suddivisa in tre macroblocchi sequenziali. Nel primo viene generato l'impulso di controllo del servo, seguito dall'elaborazione della pressione dei pulsanti e dalla generazione del ritardo di chiusura che permette di mantenere una frequenza di emissione degli impulsi di circa 50 Hz.*

precedenti, infatti, abbiamo detto che il segnale di controllo dei servocomandi deve mantenere una frequenza impulsiva di circa 50 Hz, ciò significa che **il PIC deve emettere un impulso più o meno ogni 20 msec**. Il problema sorge dal fatto che, utilizzando il metodo visto nell'esperimento precedente, **l'esecuzione del programma viene 'sospesa' (tramite il ciclo while) per tutto il tempo intercorrente tra la pressione e il rilascio dei pulsanti di controllo, bloccando con esso anche la generazione del treno di impulsi, con la conseguente perdita di rigidità del servocomando.** Di conseguenza, in questo esempio useremo un'altra strategia, **basata sull'impiego di variabili di stato dei pulsanti.** La funzione di tali variabili è quella di **memorizzare gli stati dei tre pulsanti a ogni ciclo di elaborazione del firmware**, confrontandoli con gli stati dei cicli successivi. Nel momento in cui si verifica che le variabili di stato hanno registrato un pulsante premuto, mentre i pulsanti sono stati rilasciati, significa che **è stato eseguito un click** ed è necessario elaborare l'azione associata. Nella fattispecie **il click su RA0 aumenterà la durata dell'impulso, RA2 la diminuirà e RA1 la reimposterà con un valore di centratura.** Le azioni di incremento e decremento non sono completamente 'libere', ma devono essere sottoposte a un **controllo di validità, in modo che la durata degli impulsi rimanga compresa tra i 1100 µs e i 1900 µs.** Per far ciò **definiamo due macro simboliche attraverso la direttiva #define** (box sotto), chiamate **T_MIN** e **T_MAX**, che ci consentiranno di impostare a priori i valori minimi e massimi dell'impulso. Considera inoltre che, per quanto detto nei paragrafi precedenti, i parametri dell'impulso non saranno espressi in microsecondi, ma bensì **in step da 100 µs ciascuno (in modo da utilizzare ciclicamente la chiamata Delay_us(100);)**. I valori di T_MIN e T_MAX saranno, di conseguenza, 11 e 19 (ossia 11 step da 100 µs = 1100 µs e 19 step da 100 µs = 1900 µs). La parte finale si occuperà **della generazione del ritardo** utile per l'impulso successivo.

### LA DIRETTIVA #DEFINE›››

**La direttiva di precompilazione '#define' è una speciale istruzione per il precompilatore che permette al programmatore di definire macroistruzioni e macrovalori. La sua particolarità consiste nel fatto di permettere agli sviluppatori di sostituire numeri costanti o blocchi di istruzioni con nomi simbolici, senza però dover definire variabili costanti o funzioni. Prima della reale compilazione, infatti, avviene un processo di sostituzioni fisica dei nomi simbolici '#define' con i loro 'contenuti'. Per avere un'idea di ciò che accade, puoi osservare l'esempio sottostante.**

| SORGENTE | DOPO LA PRECOMPILAZIONE |
|---|---|
| `#define LIMITE 10` | `......` |
| `.....` | `if(variabile < 10)` |
| `if (variabile < LIMITE)` | `....` |

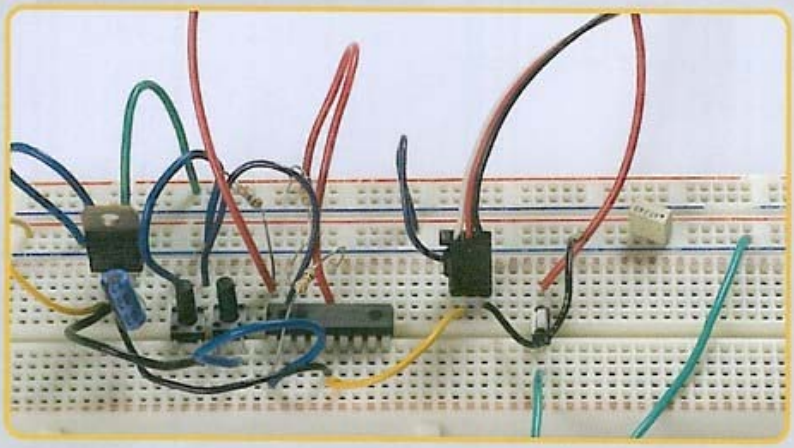
**Le direttive '#define', quindi, sono molto utili ogni volta che si vogliono definire parametri di programma che possono essere modificati nel corso dello sviluppo. Il vantaggio rispetto alla dichiarazione di variabili const consiste nel fatto che il ricorso alla direttiva '#define' non richiede l'allocazione di nuove entità.**

**IL CODICE ›››**
Prima di passare all'analisi del circuito elettronico dell'esperimento di questo Workshop, che affronteremo nel fascicolo successivo, vediamo il codice sorgente che implementa quanto descritto nelle pagine precedenti. Come puoi vedere, il sorgente inizia con la definizione dei parametri simbolici di funzionamento per mezzo di una sequenza di direttive di precompilazione #define. Tra i simboli definiti vi sono anche le macro PREMUTO e RILASCIATO, associate ai valori logici di ingresso corrispondenti a questi due stati. Nel prossimo fascicolo completeremo il progetto.

*Controllare un servo analogico*

```
/* definizione dei parametri del programma */
#define CENTRATURA 15
#define T_MIN 11
#define T_MAX 19
#define PREMUTO 0
#define RILASCIATO 1

/* funzione principale */
void main()
{

  /*indice per la gestione dei cicli */
  int i;

  /* durata dell'impulso */
  int durata_us = CENTRATURA;


  /* stato dei tre pulsanti di ingresso */
  int stato_RA0 =  RILASCIATO;
  int stato_RA1 =  RILASCIATO;
  int stato_RA2 =  RILASCIATO;


  /*configurazione delle porte del PIC. Porta A con i tre pin meno significativi
    in input e porta B configurata con tutti i pin in uscita. */
  TRISA = 0b00000111;
  TRISB = 0b00000000;

  /* disattivazione dei comparatori del 16F628 */
  CMCON = 0b00000111;


  while(1)
  {

    /*inizio dell'impulso con l'innalzamento dell'output RB0 */
    PORTB.F0 = 1;


    /*generazione del ritardo di impulso */
    for(i=0; i<durata_us; i++) Delay_us(100);


    /*termine dell'impulso con l'abbassamento dell'output RB0 */
    PORTB.F0 = 0;
```

```
    /*inizio gestione dei pulsanti */

    /*rilevamento dei rilasci se la variabile di stato dei pulsanti ha registrato la
      pressione, ma il bit corrispondente segnala il pulsante rilasciato, È stato
      ultimato il click */


    /* pulsante di decremento */
    if(stato_RA0 == PREMUTO && PORTA.F0==RILASCIATO)
    {
       durata_us--; /* decremento la durata */
       /* se la durata dell'impulso ha superato il limite minimo, la correggo */
       if (durata_us<T_MIN) durata_us = T_MIN;
    }


    /* pulsante di centratura */
    if(stato_RA1 == PREMUTO && PORTA.F1==RILASCIATO)
    {
       durata_us = CENTRATURA; /* decremento la durata */
    }


    /* pulsante di incremento */
    if(stato_RA2 == PREMUTO && PORTA.F2==RILASCIATO)
    {
       durata_us++; /* decremento la durata */
    }

    /* se la durata dell'impulso ha superato il limite massimo, la correggo */
    if (durata_us>T_MAX) durata_us = T_MAX;
    }


    /*rilevamento della pressione dei pulsanti e memorizzazione degli stati */
    stato_RA0 = PORTA.F0;
    stato_RA1 = PORTA.F1;
    stato_RA2 = PORTA.F2;


     /* generazione del ritardo di chiusura */
     for(i=0; i<RITARDO; i++) Delay_us(100);

 } /* si chiude il ciclo di funzionamento del firmware */

}
```



Una foto del circuito di controllo per servocomandi analogici che potrai realizzare nel prossimo Workshop. Come nei precedenti Workshop, anche in questo caso utilizzeremo il PIC 16F628 come elettronica centrale del nostro sistema.