

In questo tutorial verrà spiegato come utilizzare PIC Genius per realizzare un firmware in grado di pilotare fino a 4 servocomandi standard tramite 4 potenziometri, sfruttando il convertitore analogico digitale presente nel microcontrollore PIC.

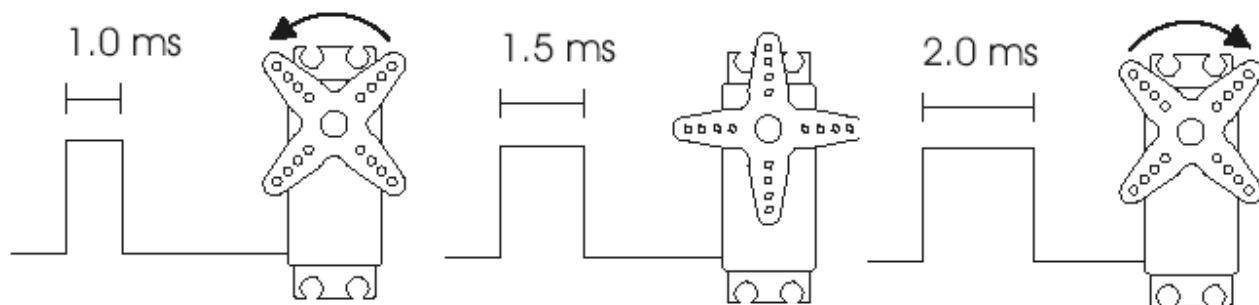
Per questo esperimento è stato utilizzato un PIC 16F676 a 4 Mhz tramite l'oscillatore interno.

Non è stato realizzato nessun circuito stampato del progetto perché al momento esso non ha una valenza pratica.

Lo scopo infatti è quello di spiegare passo passo come utilizzare il linguaggio di PIC Genius insieme agli strumenti di calcolo e simulazione.

Prima di iniziare ad utilizzare PIC Genius è opportuno spiegare il funzionamento di un servocomando standard in modo da capire cosa si deve realizzare con il microcontrollore.

Un servo dispone solitamente di tre fili, due dei quali usati per l'alimentazione ed un terzo per il segnale di pilotaggio.



La figura mostra che con un segnale di 1,5 millisecondi il servo si posizionerà al centro della sua corsa (90 gradi). Per ruotare a sinistra il perno il segnale dovrà diminuire di frequenza, mentre per ruotare a destra dovrà aumentare.

Anche se la figura mostra che i limiti del segnale vanno da 1 millisecondo a 2, in realtà esistono molti servo che vanno da 0,5 fino a 2,5 millisecondi.

I 4 servocomandi che ho utilizzato nel progetto hanno queste ultime caratteristiche, pertanto il firmware è stato scritto in modo da generare segnali di frequenza variabile da 0.5 a 2.5 ms.

Cominciamo quindi caricando il tool PIC Genius versione 5.4 e scegliendo la voce NUOVO PROGETTO.

Dare un nome al file di progetto che si vuole realizzare (servo) e scegliere il micro da utilizzare : PIC 16F676.

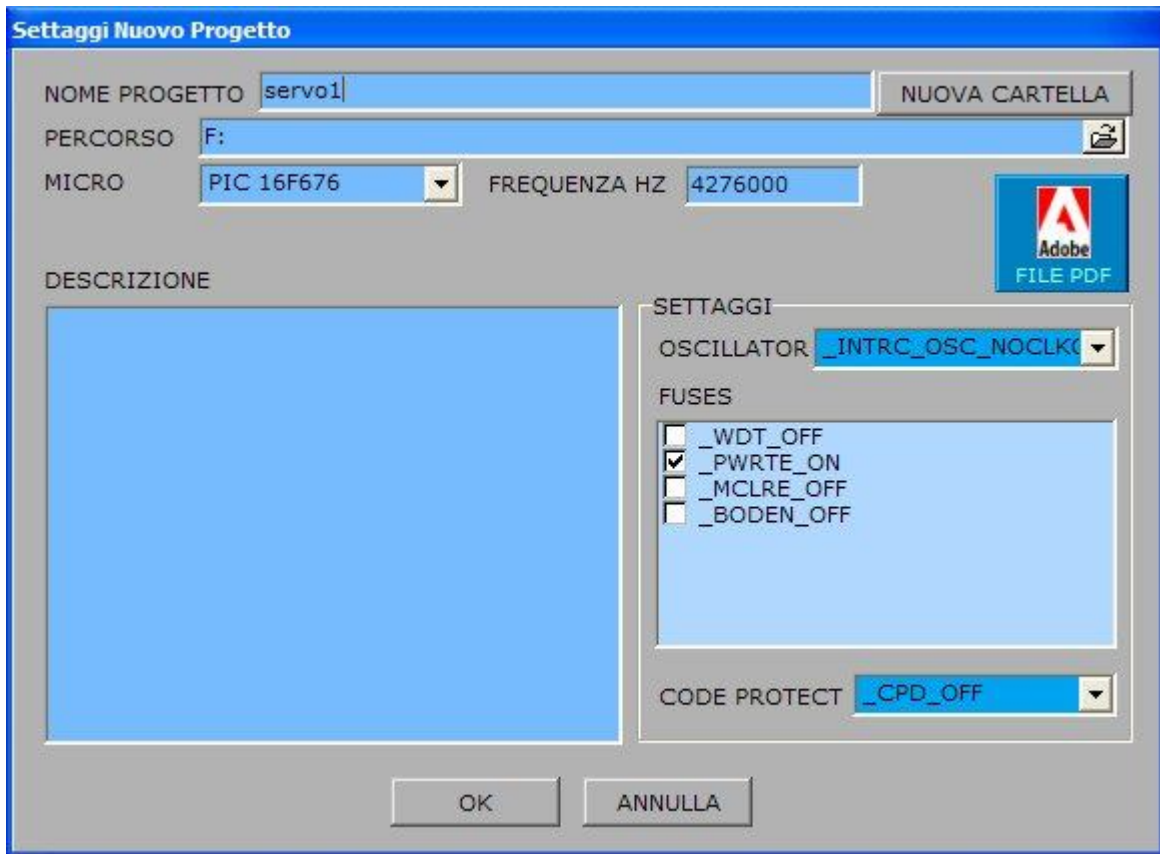


Figura 1 - Settaggi del nuovo progetto

Anche se la figura 1 mostra un valore di frequenza un po' strano, selezionare **4 MHz o 4.000.000 Hz**. Come oscillatore scegliere la voce **INTERNAL NO CLOCK OUT** mentre come **FUSES** quello evidenziato nella figura. Per la spiegazione dettagliata di questi parametri, è opportuno riferirsi al PDF del microcontrollore scelto. Nessuna protezione per quanto riguarda il codice.

Dopo aver premuto il pulsante OK ed aver accettato tutti i settaggi inseriti siamo pronti per cominciare a scrivere il codice.

Di solito, nei microcontrollori dove è presente un convertitore analogico digitale chiamato anche ADC se si volesse utilizzare una porta ad esso collegata come digitale, è necessario settare dei registri che controllano il convertitore stesso. Infatti di default tutti gli ingressi del **ADC** sono settati in modalità analogica.

**Pic Genius** permette di settare graficamente le porte senza dover scrivere del codice.

Per effettuare questi settaggi, aprire la finestra del **MODULO ADC** premendo il pulsante **PROGETTO** nella schermata di lavoro di PIC Genius.



Figura 2 - Settaggi Modulo ADC

Dalla figura è facile capire che degli **otto pin collegati all ADC** , 4 sono stati settati in modalità digitale.

A questo punto è necessario configurare i pin del micro come **INGRESSO** o **USCITA** in base al circuito che si vuole realizzare. Personalmente ho scelto di utilizzare la **PORT A** per collegare gli ingressi ( trimmer o potenziometri) e la **PORT C** (da 0 a 3) come uscite per pilotare i **4 servocomandi**. Prima quindi di scrivere una sola riga di codice è sempre opportuno cominciare a collegare i vari dispositivi che si vuole utilizzare e PIC Genius in fase di compilazione penserà in modo automatico a configurare i vari registri per settare gli ingressi e le uscite. I vari dispositivi potranno essere configurati anche durante la stesura del codice.

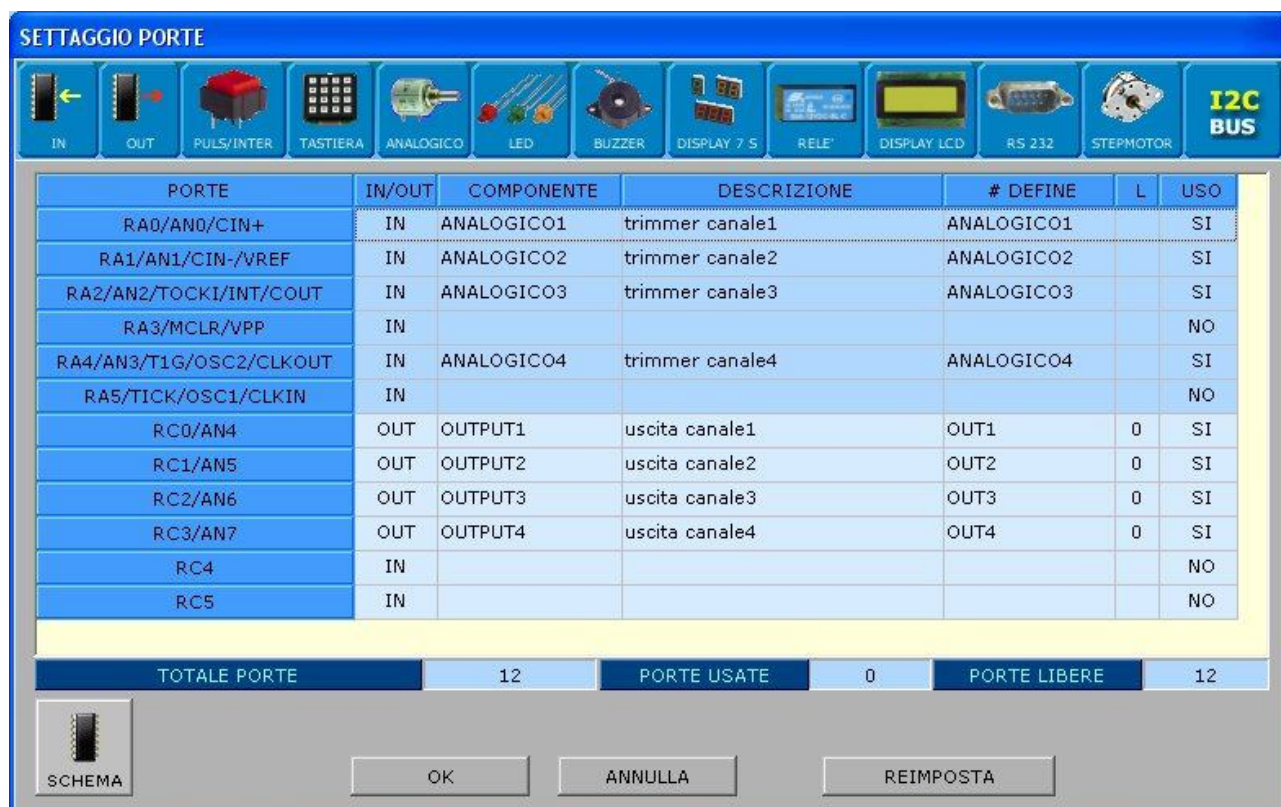


Figura 3 - Settaggio delle porte

Nella figura 3 è possibile vedere un riassunto dei dispositivi connessi alle varie porte del PIC micro.

Nella parte alta della schermata, si trovano tutti i possibili componenti o dispositivi che PIC Genius gestisce nella sua ultima versione : 5.4.

Qualora si voglia collegare un dispositivo non presente in elenco si potrà utilizzare il componente standard **IN** come **ingresso** e **OUT** come **uscita**.

Nel nostro progetto sono stati inseriti quattro componenti di tipo **analogico** (trimmer) e quattro **uscite standard**. In fase di setup di ciascun componente è possibile settare un **DEFINE** ovvero un nome simbolico del componente. Se durante la programmazione, si farà riferimento ai componenti tramite questo nome simbolico non sarà necessario modificare il codice scritto qualora in una seconda fase di stesura del codice si voglia cambiare disposizione ai componenti collegati. Ad esempio per una migliore **sbrogliatura** del circuito potrebbe essere necessario collegare il trimmer del canale 2 alla porta C1 e viceversa.



Figura 4 - Settaggio componente analogico

La figura 4 mostra i settaggi di un componente analogico mentre la 5 quelli relativi ad una uscita standard.

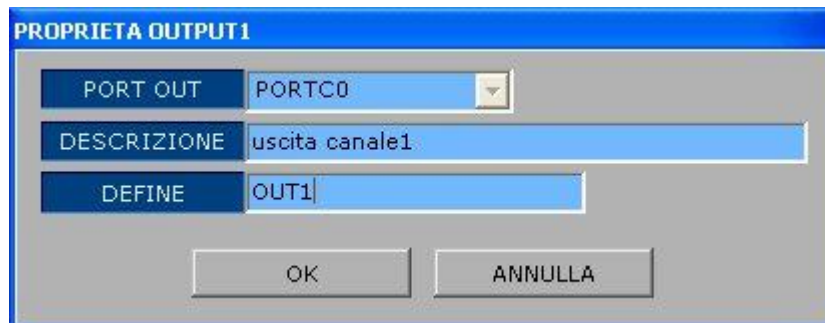


Figura 5 - Settaggio uscita standard

## TEST PRELIMINARE

Prima di iniziare con la stesura del codice, visto che si utilizzerà un **PIC micro senza il quarzo** che assicura una certa stabilità all'oscillatore, è opportuno effettuare un test per analizzare ed eventualmente correggere la **frequenza di riferimento** in modo da programmare onde quadre molto precise.

Dato che non è possibile stabilire la frequenza interna del micro è opportuno tarare la frequenza di PIC Genius in funzione del segnale generato dalla scheda hardware.

Sfruttando il pin OUT1 come uscita è possibile, scrivendo questo semplice codice, rilevare tramite un oscilloscopio o meglio un frequenzimetro la frequenza in uscita.

PROGETTO		MAINLOOP
PORTA		;
N.U.	RA0/AN0/CIN+	;
N.U.	RA1/AN1/CIN-/VREF	;
N.U.	RA2/AN2/TOCKI/INT1	;
N.U.	RA3/MCLR/VPP	
N.U.	RA4/AN3/T1G/OSC2	
N.U.	RA5/TICK/OSC1/CLK	main:
PORTC		<b>delay_precision_us</b> (500)
OUT	RC0/AN4	OUT1=1
N.U.	RC1/AN5	<b>delay_precision_us</b> (500)
N.U.	RC2/AN6	OUT1=0
N.U.	RC3/AN7	
N.U.	RC4	
N.U.	RC5	<b>goto</b> main

Figura 6 - Onda quadra 1000 Hz

Questo codice genera una onda quadra di circa 1000 Hz direttamente sul pin **RC0 (OUT1)** del micro. Tramite un frequenzimetro sarà possibile leggere tale frequenza ed eventualmente valutare il margine di errore che potrebbe avere l'oscillatore principale del micro.

Il chip da me utilizzato genera una frequenza di **1069 hz**. Questo è dovuto alla non precisione dell'oscillatore principale che ricordo non è quarzato. Con Pic Genius possiamo modificare però la frequenza dell'oscillatore in modo da far quadrare tutti i valori. Per calcolare la nuova frequenza basterà eseguire questa operazione  $(4000000 * 1069) / 1000 = 4276000$ . Inserendo questo nuovo valore nella frequenza di riferimento di **PIC Genius**, si avrà la certezza di generare forme d'onda molto precise.

## MUOVERE UN SERVO

Dopo aver terminato il test ed aver tarato l'oscillatore principale, è arrivato il momento di cominciare a scrivere del codice per muovere per il momento un solo servocomando. Scriviamo quindi un semplice codice che ci permetterà di generare un impulso positivo della durata di **1.5** millisecondi ed uno negativo della durata di **18,5** millisecondi in modo continuo.

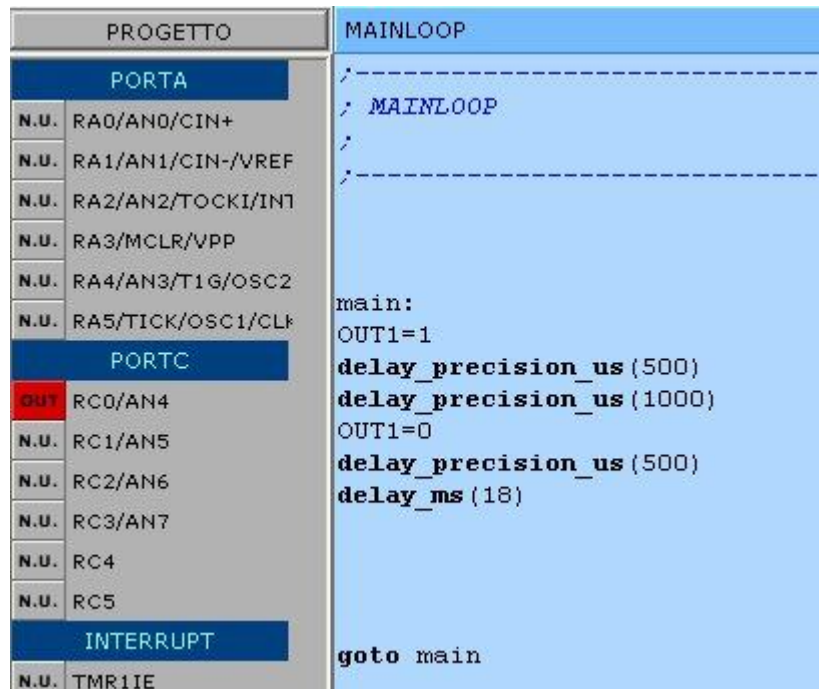


Figura 7 -Mettere il servo al centro

Tramite questo semplice codice sull' uscita **RC0 (OUT1)** vi sarà un segnale in grado di posizionare un servocomando nella sua posizione centrale. L'istruzione **delay\_precision\_us()** è una istruzione **statica di PIC Genius**, nel senso che come parametro potrà essere indicato solamente un **numero certo e non una variabile**. Questo perché è molto difficile assicurare tramite una routine la precisione di un ritardo espresso in **microsecondi** in quanto a 4 mhz ogni singola istruzione in ASM perde un microsecondo ad eccezione di qualcuna che ne perde due. **In un linguaggio ad alto livello come quello implementato in PIC Genius è impossibile stabilire prima i tempi di lavoro di una porzione di listato.**

Vediamo adesso come si fa ,sfruttando PIC Genius a calcolare quanti microsecondi impiega una determinata istruzione o una porzione di codice.

Come esempio, ma anche perché verrà poi utilizzata nel nostro codice ho scelto il **ciclo for**.

La sintassi del ciclo **FOR** è la seguente :

**FOR** *variabile* = *valoreinizio* **TO** *valorefine*

**ENDFOR**

Questa istruzione presente in tutti i linguaggi ad alto livello, (cambia la sintassi) permette di eseguire una o più istruzioni (una porzione di codice) per un numero definito di volte.

Esempio se **valoreinizio=1** e **valorefine=100** il codice scritto tra FOR e ENDFOR verrebbe eseguito **100 volte**.

Ipotizziamo di scrivere adesso all'interno del blocco una istruzione di ritardo :

**FOR** ciclo = 1 **TO** 100

**DELAY\_PRECISION\_US(10)**

**ENDFOR**

Riferendoci al discorso di prima, in teoria il tempo totale di questo **loop** dovrebbe essere  $100 * 10 = 1000$  microsecondi. In realtà esso non accade perché l'istruzione **FOR ENDFOR** perde già una serie di microsecondi in base anche al tipo di variabile usata.

L'unico modo per sapere con esattezza quanti microsecondi occorrono per eseguire il loop è quello di posizionare **due breakpoint** e lanciare la simulazione con PIC Genius. Per prima cosa dichiariamo la variabile ciclo di tipo **INT** che potrà assumere un valore da 0 a 4.294.967.295 (32 bit) In realtà potrebbe essere utilizzata anche una var di tipo **BYTE da 0 a 255** ma quando si usa un ciclo **FOR con almeno 2 variabili PIC Genius per sicurezza richiede che esse siano dello stesso tipo.**



Figura 8 -Dichiarazione variabile

Per aprire questa finestra è necessario premere il pulsante che rappresenta un segno di addizione posto in basso a destra della schermata principale di PIC Genius. La variabile inserita comparirà nella lista delle variabili usate.

Adesso scriviamo questo semplice codice ricordandosi di settare almeno una porta del microcontrollore poiché in caso contrario PIC Genius **non eseguirà la compilazione del codice scritto.**

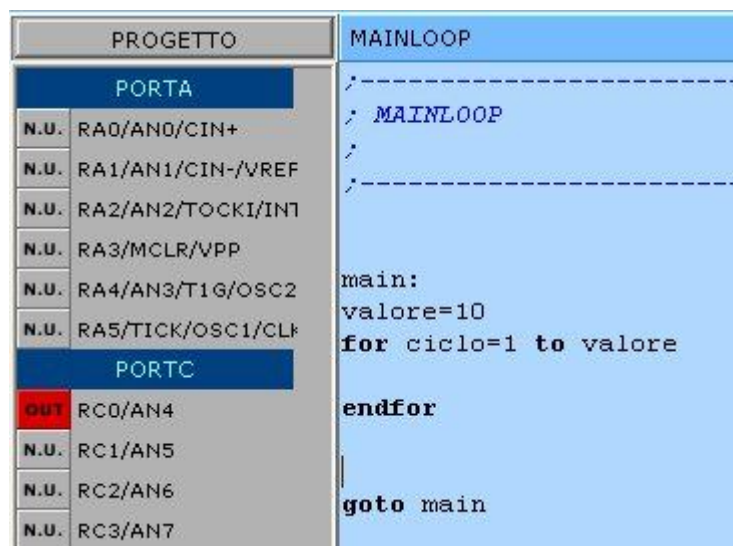


Figura 9 - Codice di un ciclo FOR vuoto

Dopo aver editato questo semplicissimo codice, eseguiamo la compilazione che genera il codice assembler e successivamente la trasformazione del **asm in hex** tramite il tool aggiuntivo **MPASM** che deve naturalmente risultare installato e i suoi percorsi devono essere conosciuti da PIC Genius. Per ottenere il file hex e poter avviare la simulazione tramite PIC Genius vi sono due diversi modi. **Il primo più veloce è una prerogativa della versione registrata di PIC Genius.** Basterà premere il pulsante che avvia il **SIMULATORE** nella barra dei pulsanti di PIC Genius posta in alto nella schermata principale. Per chi utilizza la versione **FREE** invece la procedura è un pò più lunga e consiste nel generare dapprima il file **ASM** premendo il relativo pulsante posto nella barra dei pulsanti, e successivamente avviare il simulatore dopo aver assemblato il file **asm**

```

PIC
Listato ASSEMBLER del progetto: cicloFOR

ASSEMBLA  DEBUG E SIMULATORE  PROGRAMMA  STAMPA

MICROCONTROLLORE  PIC 16F676  STATISTICHE

;
;          MOVLW      0          ;00000000
;          BANKSEL   CMCON
;          MOVWF     CMCON
;
;-----
; ; SETTAGGIO ANSEL
;-----
;
;          MOVLW      255       ;11111111
;          BANKSEL   ANSEL
;          MOVWF     ANSEL
;          BCF       STATUS,RPO
;          GOTO      MAINLOOP
MAINLOOP
;main:
main
;for ciclo=1 to 10
;          MOVLW      1
;          MOVWF     ciclo
;_FOR1LOOP
;          BCF       STATUS,RPO
;          MOVF      ciclo,W
;          SUBLW     10
;          BTFSC    STATUS,C
;          GOTO      _FOR1BODY
;          GOTO      _ENDFOR1
;_FOR1BODY
;endfor
;          BCF       STATUS,RPO
;          INCFSZ   ciclo,F
;          GOTO      _FOR1LOOP
;_ENDFOR1
;goto main
;          GOTO      main
;          ORG      0x2100

```

Figura 10 - Finestra dell'assembler

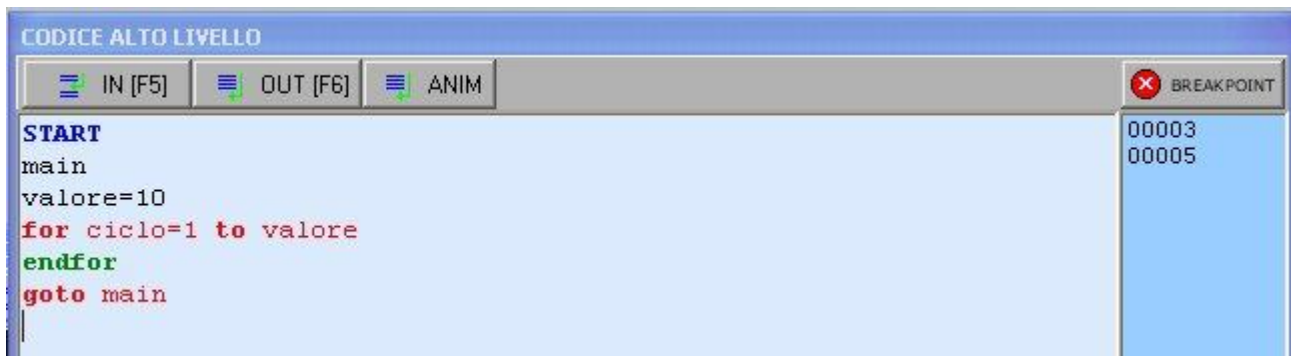
La figura mostra il codice ASM del ciclo FOR scritto in linguaggio ad alto livello.



## LA PRIMA SIMULAZIONE

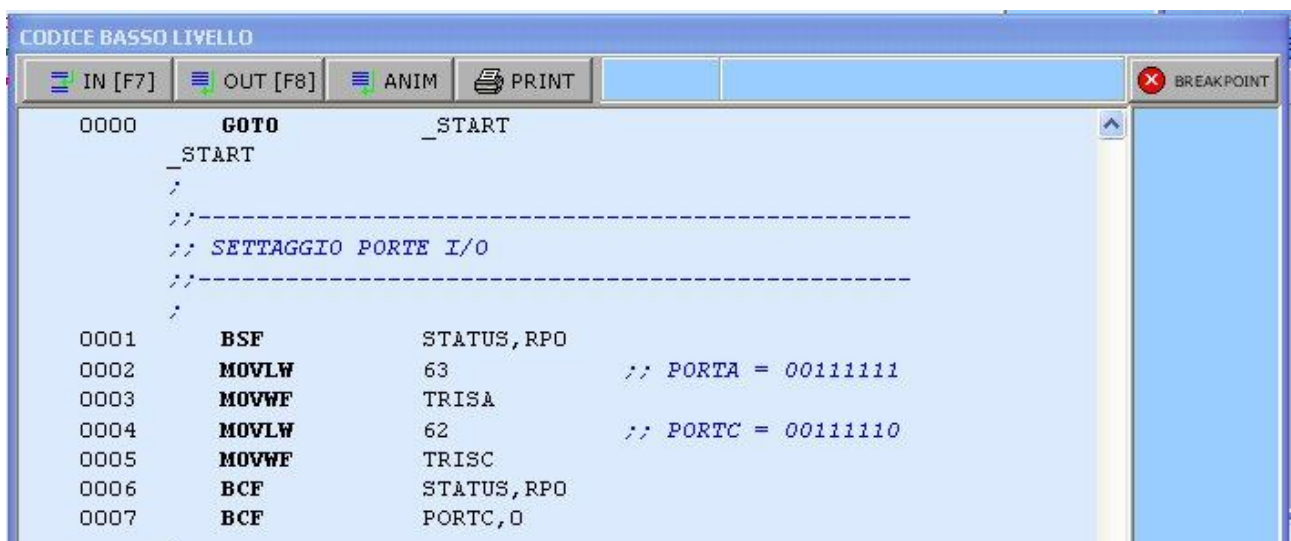
Dopo aver avviato il simulatore di PIC Genius si aprirà una finestra contenitore all'interno della quale si possono posizionare diverse finestre di controllo e di calcolo per la visualizzazione dei **registri e delle variabili, dei valori della eeprom interna ecc.ecc.**

Vi sono anche due finestre che riportano la prima il codice ad alto livello scritto, mentre la seconda il corrispettivo codice **ASM** generato. In queste due finestre il programmatore potrà posizionare dei **breakpoint** (punti in cui si ferma la simulazione del codice) al fine di valutare lo stato di registri, bit ecc.ecc in quel dato momento.



```
CODICE ALTO LIVELLO
IN [F5]  OUT [F6]  ANIM  BREAKPOINT
START
main
valore=10
for ciclo=1 to valore
endfor
goto main
00003
00005
```

Figura 11 - Codice ad alto livello



```
CODICE BASSO LIVELLO
IN [F7]  OUT [F8]  ANIM  PRINT  BREAKPOINT
0000  GOTO      _START
      _START
      ;
      ;-----
      ;; SETTAGGIO PORTE I/O
      ;-----
      ;
0001  BSF      STATUS,RPO
0002  MOVLW   63      ;; PORTA = 00111111
0003  MOVWF   TRISA
0004  MOVLW   62      ;; PORTC = 00111110
0005  MOVWF   TRISC
0006  BCF      STATUS,RPO
0007  BCF      PORTC,0
```

Figura 12 - Codice basso livello

Nella figura che rappresenta il codice ad alto livello è possibile notare che due righe risultano **colorate in rosso**. Questo colore identifica che in queste due righe è stato inserito un punto di **break** o **breakpoint**. Quando si avvierà la simulazione (tramite il pulsante **RUN**) il simulatore eseguirà il codice scritto ad alta velocità e si fermerà sul primo **breakpoint** trovato. (in questo caso alla riga 2).

Al successivo comando di **RUN** la simulazione riprenderà ad alta velocità per arrestarsi nuovamente ad un altro eventuale punto di break. **Per inserire o togliere un breakpoint** basta posizionarsi sulla riga voluta e tramite il tasto **destra del mouse scegliere la relativa voce**. I punti di break possono essere inseriti indipendentemente nella finestra del **codice ad alto livello** o in quella dove vi è

**l'assembler.** Sarà possibile altresì proseguire la simulazione del codice in modalità passo passo **IN** o **OUT** o in modalità animata.

Per proseguire con il nostro esempio dopo aver posizionato i due **breakpoint all'inizio e alla fine del ciclo FOR** avviamo la simulazione. Come detto prima l'esecuzione si arresterà nel primo breakpoint. Riprendendo il pulsante di **RUN** proseguirà fino al secondo breakpoint. A questo punto basterà consultare la **finestra dei contatori** per rilevare che tra i due punti sono intercorsi circa **2000 microsecondi**.. Il valore **77** da assegnare alla variabile valore, è stato trovato dopo qualche tentativo.

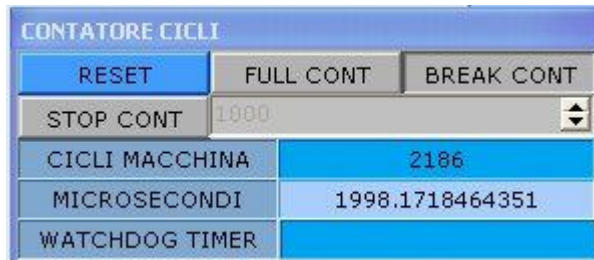


Figura 13 - Finestra contatore cicli

Questa finestra permette di operare diverse tipologie di calcolo.

Inserendo il valore 0 nel ciclo **FOR** il tempo utilizzato dalla routine è pari a **27 microsecondi** quindi variando il valore della variabile da **0 a 77** si potrà gestire un ritardo da **27 a 1998** microsecondi. E' proprio il tempo per pilotare il nostro servocomando. Adesso bisogna calcolare i valori in **funzione della posizione del nostro potenziometro** collegato ad un canale del convertitore analogico digitale (adc) del nostro microcontrollore.

L'**adc** del PIC micro ha una risoluzione a **10 bit quindi ritornerà un valore da 0 a 1023**.

Tramite una semplice espressione matematica si potrà ottenere l'intervallo dei valori utili per ottenere i tempi di pilotaggio dei servocomandi.

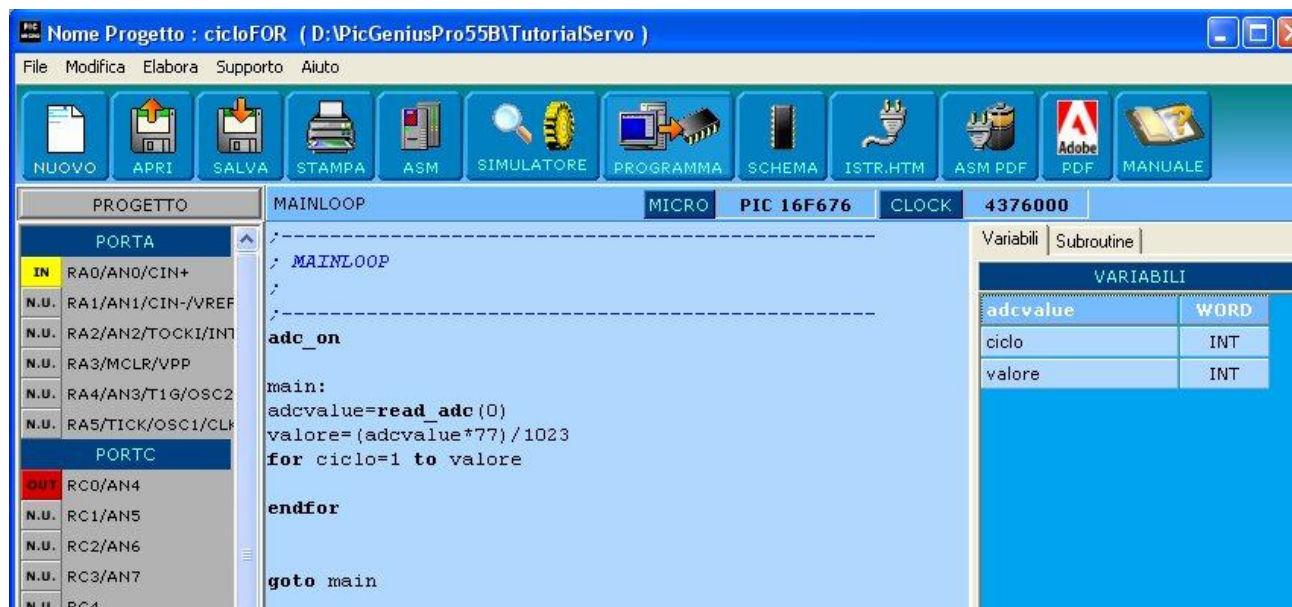


Figura 14 - Esempio calcolo valori

Per prima cosa viene acceso il convertitore con l'istruzione **adc\_on**

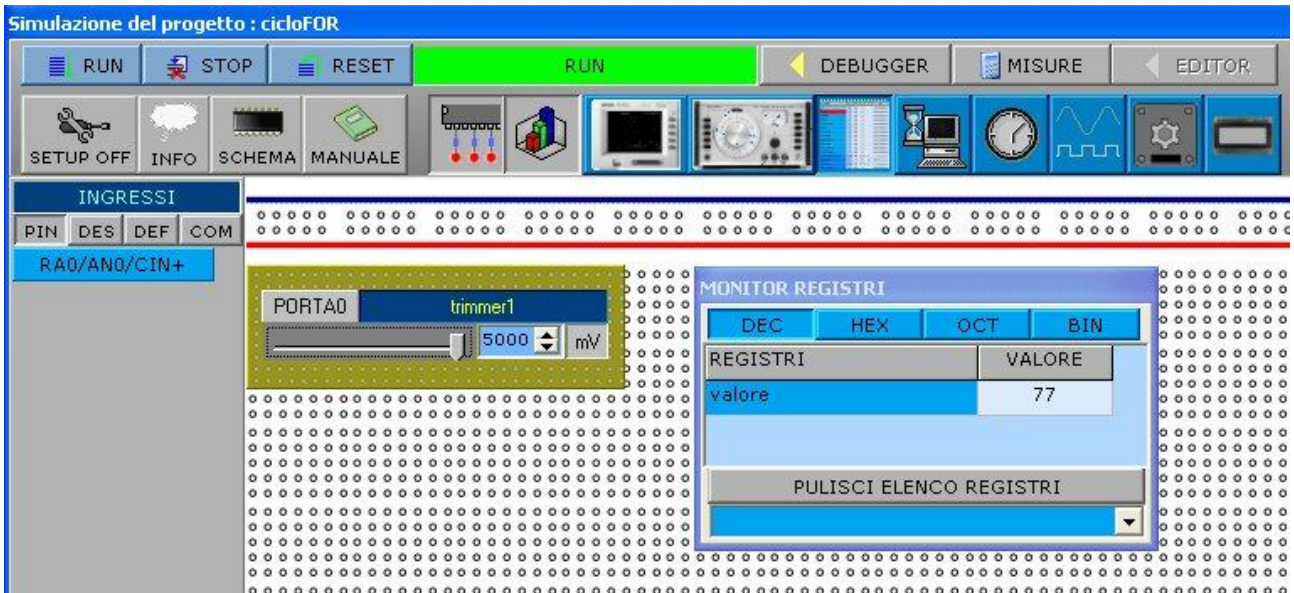
Successivamente dopo la **label main** viene letto il valore del convertitore relativo al canale 0 e memorizzato nella variabile **adcvalue**.

La variabile **valore** è una variabile di **tipo INT** in grado di immagazzinare un numero a 32 bit.

Il valore ricavato dall'ADC viene quindi moltiplicato per il numero fisso 77 (che rappresenta il massimo per il nostro valore di ritardo) e successivamente diviso per il numero fisso che

corrisponde al valore massimo dell'ADC. Quando **adcvalue** sarà uguale a 1023 (cloche a destra) il risultato sarà uguale a 77 e si perderà un tempo di 1998 microsecondi, mentre quando **adcvalue** sarà uguale a zero il risultato sarà zero e si perderà un **tempo di soli 27 microsecondi**.

Questa parte teorica fin qui espressa potrà essere simulata passo passo tramite **PIC Genius o meglio ancora dinamicamente ad alta velocità leggendo in tempo reale il valore che assumerà la variabile valore in base allo spostamento del potenziometro**.

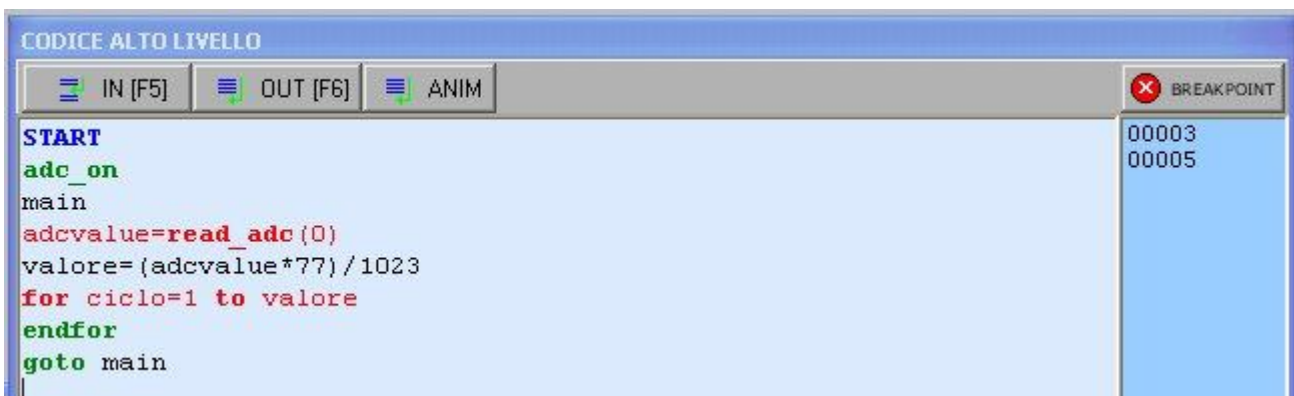


**Figura 15 Simulazione alta velocità**

La figura 15 mostra la simulazione ad alta velocità del codice appena descritto. Dalla schermata non è possibile naturalmente apprezzare le variazioni della variabile **valore** in base allo spostamento del potenziometro della cloche ma si ha la conferma che il codice finora editato è funzionante.

Adesso per continuare nel nostro progetto, si deve calcolare il tempo utilizzato sia per effettuare la lettura del convertitore, sia per calcolare il numero da assegnare alla variabile **valore**.

Avviamo la simulazione e posizioniamo i due breakpoint per effettuare il calcolo :



**Figura 16 - Calcolo dei tempi**

Dalla simulazione emerge che il tempo necessario per effettuare questi calcoli è di 2268 microsecondi. Riassumiamo per un attimo il funzionamento del servocomando :

- 1) il pin di controllo va a 1 per un tempo di 500 microsecondi
- 2) il pin di controllo rimane a 1 per il tempo necessario a far sì che il rotore del servo si posizioni nella **posizione voluta** (att da non confondere con il tempo per raggiungere la posizione)

3) il pin di controllo **va a zero per la rimanenza dei 20 millisecondi.**

Analizzando quanto detto e quello che abbiamo costruito con il codice è evidente che c'è una grossa discordanza.

Dato che i **2268 microsecondi** non possono essere modificati in quanto non dipendono dal software ma dall'hardware, si deve trovare un'altra strategia o soluzione, ovvero spostare la decodifica e l'elaborazione della posizione del potenziometro nella parte non attiva del treno d'impulsi ovvero nei restanti 20 millisecondi. Per fare un esempio chiarificatore si potrebbe dire :

- 1) il micro è avviato
- 2) il pin di controllo viene messo a **1** per il tempo di **500 microsecondi.**
- 3) La variabile valore assume il valore **0** come inizializzazione pertanto il posizionamento teorico del servo dovrebbe essere a sinistra. (in realtà non arriverebbe neanche a muoversi anche se fosse tutto a destra)
- 4) Il pin di controllo viene messo a 0 e da questo momento ha inizio la decodifica e l'elaborazione della posizione del potenziometro per una durata di **2268 microsecondi**
- 5) Il pin di controllo rimane a **0** per il tempo necessario affinché siano passati **20 millisecondi** dall'inizio del punto 2
- 6) Il **loop ricomincia**, questa volta però la variabile valore contiene il numero esatto che determinerà il posizionamento del servocomando.

E' questa la tecnica che è stata utilizzata per gestire fino a 4 servocomandi tramite un solo treno di 20 millisecondi.

```

PROGETTO MAINLOOP MICRO PIC 16F676 CLOCK 4276000 HZ
PORTA
IN RA0/AN0/CIN+
N.U. RA1/AN1/CIN-/VREF
N.U. RA2/AN2/TOCKI/INT1
N.U. RA3/MCLR/VPP
N.U. RA4/AN3/T1G/OSC2
N.U. RA5/TICK/OSC1/CLK
PORTC
OUT RC0/AN4
N.U. RC1/AN5
N.U. RC2/AN6
N.U. RC3/AN7
N.U. RC4
N.U. RC5
INTERRUPT
N.U. TMR1IE
N.U. CMIE
N.U. ADIE
N.U. EEIE
N.U. RAIE
N.U. INTE
N.U. TOIE
N.U. PEIE
N.U. GIE

;-----
; MAINLOOP
;-----
adc_on
valore=2
valoremix=77
main:

OUT1=1 ; il pin di controllo viene messo a 1
delay_precision_us(473) ; si perdono 473 us
for ciclo=1 to valore ; qui si perderanno da 27 a 1998 microsecondi

endfor
OUT1=0 ; il pin di controllo è messo a zero
; adesso finiamo il conteggio
valoreout=valore
if valoreout<77 then inc(valoreout) endif
for ciclo=valoreout to valoremix

endfor

adcvalue=read_adc(0) ; si decodifica la posizione del potenziometro
valore=(adcvalue*77)/1023 ; la si elabora
; adesso la variabile valore contiene la nuova posizione utile da
; usare nel prossimo treno d'impulsi
; tutto il processo perde 2840 microsecondi quindi adesso si deve
; generare la rimanenza per 20 millisecondi
delay_ms(15)
delay_precision_us(75)
goto main

```

VARIABILI	
adcvalue	WORD
ciclo	INT
valore	INT
valoremix	INT
valoreout	INT

Figura 17 - Codice controllo servocomando

La figura mostra il codice utile per gestire un servocomando tramite un potenziometro o trimmer collegato nel canale 0 del convertitore.

## IL CODICE PER CONTROLLARE UN SOLO SERVO

```
-----  
; MAINLOOP  
;  
-----  
adc_on  
valore=2  
valoremax=77  
main:  
  
OUT1=1 ; il pin di controllo viene messo a 1  
delay_precision_us(473) ; si perdono 473 us  
for ciclo=1 to valore ; qui si perderanno da 27 a 1998 microsecondi  
  
endfor  
OUT1=0 ; il pin di controllo è messo a zero  
; adesso finiamo il conteggio  
  
valoreout=valore  
if valoreout<77 then inc(valoreout) endif  
for ciclo=valoreout to valoremax  
  
endfor  
; in questo punto sono già stati utilizzati circa 2500 microsecondi +98  
; per alcuni calcoli  
  
adcvalue=read_adc(0) ; si decodifica la posizione del potenziometro  
valore=(adcvalue*77)/1023 ; la si elabora  
; adesso la variabile valore contiene la nuova posizione utile da  
; usare nel prossimo treno d'impulsi  
; tutto il processo perde 2316 microsecondi quindi adesso si deve  
; generare la rimanenza per 20 millisecondi  
; 2598+2340=4938 microsecondi quindi 20000-4840 = 15062 circa  
delay_ms(15)  
delay_precision_us(75)  
; con queste due istruzioni si conclude la temporizzazione che supera di poco  
; i 20 millisecondi  
goto main  
; il ciclo si ripete  
Il segnale di controllo verrà ripetuto ogni 20 millisecondi (50 volte al  
secondo) e il servo sarà in questo caso sempre sotto tiro
```

Adesso è arrivato il momento di scrivere il codice che invece controlla fino a 4 servocomandi. Naturalmente l'acquisizione dei valori dei potenziometri deve avvenire nei tempi morti e cioè durante lo stato di off del segnale.

La logica del programma è:

- 1) Viene generato il primo segnale relativo al primo servo
- 2) Viene generato un segnale off fino a coprire i 2,5 millisecondi qualora ve ne sia bisogno.
- 3) I punti 1 e 2 vengono ripetuti per gli altri 3 servo per un totale di 10 millisecondi.
- 4) Nei restanti 10 millisecondi, avverrà l'acquisizione ed il calcolo dei nuovi valori di posizionamento.

## SORGENTE PER GESTIRE 4 SERVOCOMANDI

La posizione angolare dei 4 servocomandi sarà sempre aggiornata (circa 50 volte al secondo)

```
-----  
; MAINLOOP  
;  
-----  
adc_on  
valore1=0  
valore2=0  
valore3=0  
valore4=0  
  
valoremax=77  
main:  
  
; -----controllo del servo 1-----  
OUT1=1 ; il pin di controllo viene messo a 1  
CALL RITARDO  
for ciclo=1 to valore1 ; qui si perderanno da 27 a 1998 microsecondi  
endfor  
OUT1=0 ; il pin di controllo è messo a zero  
; adesso finiamo il conteggio  
valoreout=valore1  
if valoreout<77 then inc(valoreout) endif  
for ciclo=valoreout to valoremax  
endfor  
; -----controllo del servo 2-----  
OUT2=1 ; il pin di controllo viene messo a 1  
CALL RITARDO  
for ciclo=1 to valore2 ; qui si perderanno da 27 a 1998 microsecondi  
endfor  
OUT2=0 ; il pin di controllo è messo a zero  
; adesso finiamo il conteggio  
valoreout=valore2  
if valoreout<77 then inc(valoreout) endif  
for ciclo=valoreout to valoremax  
endfor  
; -----controllo del servo 3-----  
OUT3=1 ; il pin di controllo viene messo a 1  
CALL RITARDO  
for ciclo=1 to valore3 ; qui si perderanno da 27 a 1998 microsecondi  
endfor  
OUT3=0 ; il pin di controllo è messo a zero  
; adesso finiamo il conteggio  
valoreout=valore3  
if valoreout<77 then inc(valoreout) endif  
for ciclo=valoreout to valoremax  
endfor  
; -----controllo del servo 4-----  
OUT4=1 ; il pin di controllo viene messo a 1  
CALL RITARDO  
for ciclo=1 to valore4 ; qui si perderanno da 27 a 1998 microsecondi  
endfor  
OUT4=0 ; il pin di controllo è messo a zero  
; adesso finiamo il conteggio  
valoreout=valore4  
if valoreout<77 then inc(valoreout) endif  
for ciclo=valoreout to valoremax  
endfor
```

```

adcvalue=read_adc(0) ; si decodifica la posizione del potenziometro
valore1=(adcvalue*77)/1023 ; la si elabora
adcvalue=read_adc(1) ; si decodifica la posizione del potenziometro
valore2=(adcvalue*77)/1023 ; la si elabora
adcvalue=read_adc(2) ; si decodifica la posizione del potenziometro
valore3=(adcvalue*77)/1023 ; la si elabora
adcvalue=read_adc(3) ; si decodifica la posizione del potenziometro
valore4=(adcvalue*77)/1023 ; la si elabora

delay_precision_us(470)
; con questa istruzione si conclude la temporizzazione che supera di poco
; i 20 millisecondi
goto main
; il ciclo si ripete

```

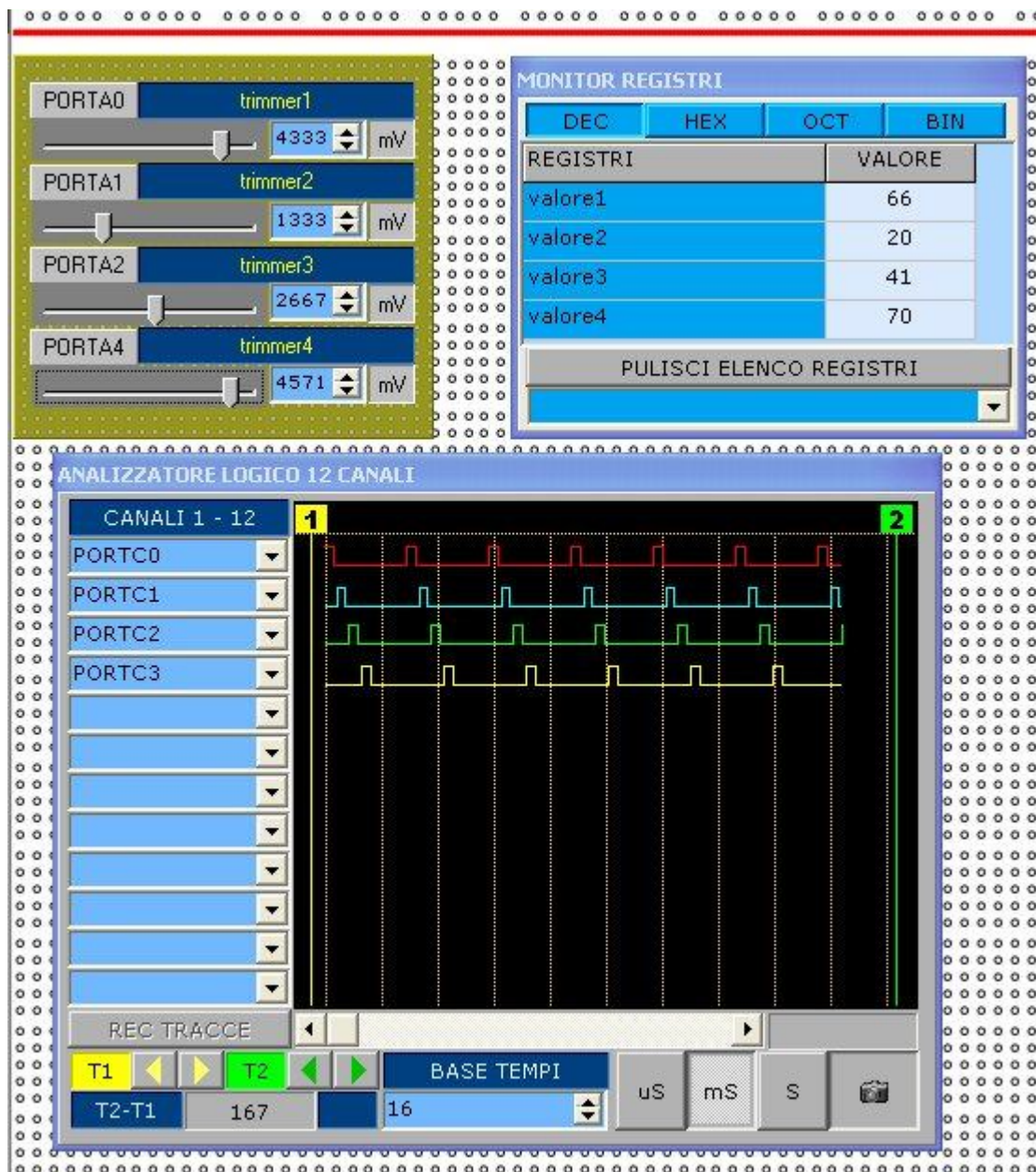


Figura 18 - Simulazione alta velocità

Nella figura è possibile vedere Pic Genius in azione durante la simulazione del firmware appena descritto.

Il **MONITOR REGISTRI** permette di analizzare in tempo reale il contenuto delle variabili e il loro cambiamento in base alla regolazione dei 4 potenziometri (ingressi)  
L'analizzatore logico invece permette di vedere le 4 forme d'onda generate a cascata.