Università degli studi di Roma
"La Sapienza"

DIS
Dipartimento di Informatica e Sistemistica

Laboratorio di Automatica

Implementazione di un controllore PID digitale per il controllo di posizione di un motore DC

A.A. 2004/05

Controllo di posizione di un motore DC

L'oggetto in esame riguarda la progettazione di un controllore per il pilotaggio di un motore in continua. In particolare il motore è corredato di un encoder incrementale con una risoluzione pari a 500 impulsi/giro, un riduttore di giri con rapporto n=19,7 e una lancetta collegata all'albero motore come si può vedere nella figura 1.

figura 1 – Motore con lancetta, riduttore ed encoder incrementale

Il controllore PID (Proporzionale Integrale Derivativo) viene implementato su un PIC16F876 prodotto dalla Microchip mediante un codice assembler che esegue tutte le procedure atte a realizzarlo. Nella figura 2 viene riportato il microcontrollore in questione che, come si può vedere, è 14+14 pin con frequenza di lavoro massima pari a 20 MHz.

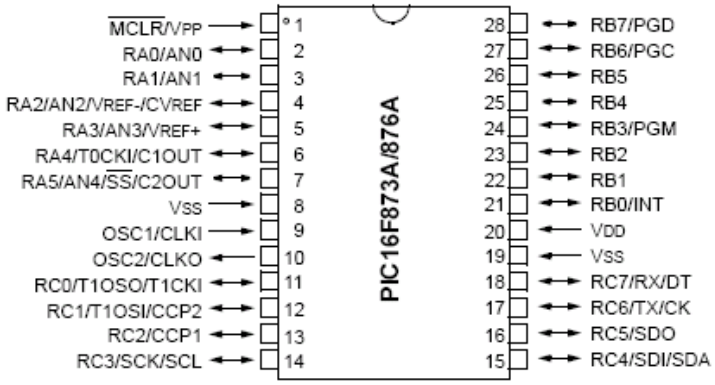


figura 2 – Piedinatura del microcontrollore PIC16F876

Le caratteristiche principali di questo processore sono qui riportate:

- Frequenza di clock massima: *20 MHz*
- Set di 35 istruzioni (*ISA – Instruction Set Architetture*)
- *8kx14* parole di memoria flash
- *368x8* byte di *RAM*
- *256x8* byte di *EEPROM*
- Gestione di interrupt
- *200 nsec* per ogni ciclo istruzione

figura 3 – Aspetto esterno del microcontrollore PIC16F876

Le linee di *I/O* (*Input – Output*) del processore in esame sono 22 divise su tre porte: *porta A* con 6 linee, *porta B* con 8 linee e *porta C* con 8 linee. La memoria flash del *PIC* risulta mappata in quattro pagine come in figura 4, mentre i registri operativi e general purpose risultano divisi in quattro banchi selezionabili attraverso i bit *RP0,RP1* del registro *STATUS* (figura 5).

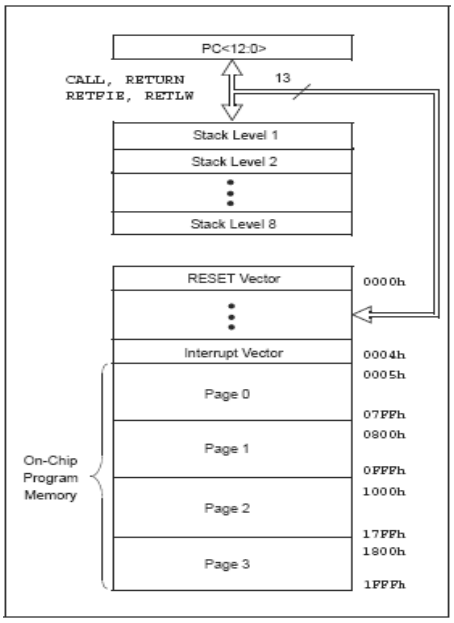


figura 4 – Organizzazione della memoria del *PIC16F876*

Il registro *STATUS*, è un registro operativo presente in tutti i banchi e l'indirizzo relativo al primo banco è $(03)_H$. La configurazione delle linee delle tre porte come ingressi o uscite avviene attraverso i registri *TRISA,TRISB,TRISC* i cui indirizzi sono rispettivamente $(85)_H$,$(86)_H$,$(87)_H$ (secondo banco – *Bank 1*), mentre il livello logico delle linee è controllato dai registri *Port_A, Port_B, Port_C* individuati dagli indirizzi $(05)_H$,$(06)_H$,$(07)_H$ (primo banco – *Bank 0*).

R/W-0	R/W-0	R/W-0	R-1	R-1	R/W-x	R/W-x	R/W-x
IRP	RP1	RP0	$\overline{TO}$	$\overline{PD}$	Z	DC	C
bit 7							bit 0

bit 7 **IRP**: Register Bank Select bit (used for indirect addressing)
1 = Bank 2, 3 (100h - 1FFh)
0 = Bank 0, 1 (00h - FFh)

bit 6-5 **RP1:RP0**: Register Bank Select bits (used for direct addressing)
11 = Bank 3 (180h - 1FFh)
10 = Bank 2 (100h - 17Fh)
01 = Bank 1 (80h - FFh)
00 = Bank 0 (00h - 7Fh)
Each bank is 128 bytes

bit 4 $\overline{TO}$: Time-out bit
1 = After power-up, `CLRWDT` instruction, or `SLEEP` instruction
0 = A WDT time-out occurred

bit 3 $\overline{PD}$: Power-down bit
1 = After power-up or by the `CLRWDT` instruction
0 = By execution of the `SLEEP` instruction

bit 2 Z: Zero bit
1 = The result of an arithmetic or logic operation is zero
0 = The result of an arithmetic or logic operation is not zero

bit 1 DC: Digit carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
(for $\overline{borrow}$, the polarity is reversed)
1 = A carry-out from the 4th low order bit of the result occurred
0 = No carry-out from the 4th low order bit of the result

bit 0 C: Carry/$\overline{borrow}$ bit (`ADDWF`, `ADDLW`, `SUBLW`, `SUBWF` instructions)
1 = A carry-out from the Most Significant bit of the result occurred
0 = No carry-out from the Most Significant bit of the result occurred

Note: For $\overline{borrow}$, the polarity is reversed. A subtraction is executed by adding the two's complement of the second operand. For rotate (`RRF`, `RLF`) instructions, this bit is loaded with either the high, or low order bit of the source register.

figura 5 – Struttura del registro *STATUS*

L'inizializzazione di una porta (per esempio la *porta A*) avviene nel seguente modo:

1. Si associano gli indirizzi dei registri a degli identificatori (*Id - Identificator*) mediante la direttiva *EQU*[1]:

```
STATUS   EQU 0x03
Port_A   EQU 0x05
TRISA    EQU 0x85
```

2. Una linea viene configurata come uscita se il livello logico del bit corrispondente nel registro *TRISA* è basso, viceversa se alto.

[1] E' importante non confondere una istruzione con una direttiva in quanto questa è relativa all'assemblatore, indica cioè un'operazione interna all'assemblatore.

```
Movlw    B'00101011'    ;Carico valore nel registro di lavoro W
Bsf      STATUS,5       ;Passaggio al bank 1
Movwf    TRISA          ;Linee RA5,RA3,RA1,RA0 configurate come ingressi
Bcf      STATUS,5       ;Passaggio al bank 0
```

La prima istruzione carica il valore binario *(00101011)* nel registro di lavoro *W*, dopodiché viene settato al livello logico alto il bit 5 del registro *STATUS* (*RP0*) per passare ai registri presenti nel *Bank 1*. Qui si trova il registro *TRISA* il quale viene caricato con il valore presente in *W*, cioè $(00101011)_2$. Infine, l'ultima istruzione consente di ritornare al *Bank 0* portando nuovamente al livello logico basso il bit 5 del registro *STATUS*.
Dopo aver eseguito la precedente sequenza di istruzioni, la *porta A* sarà configurata nel seguente modo:

Configurazione Porta A							
Non disponibile	Non disponibile	Linea RA5	Linea RA4	Linea RA3	Linea RA2	Linea RA1	Linea RA0
-	-	Ingresso	uscita	Ingresso	uscita	Ingresso	ingresso
Valore del registro TRISA							
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
0	0	1	0	1	0	1	1

La gestione delle interrupt viene controllata mediante il registro *Intcon* presente in tutti i banchi con indirizzo $(0B)_H$ nel *Bank 0* come visibile nella figura seguente:

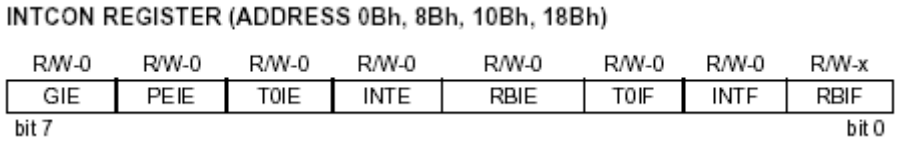


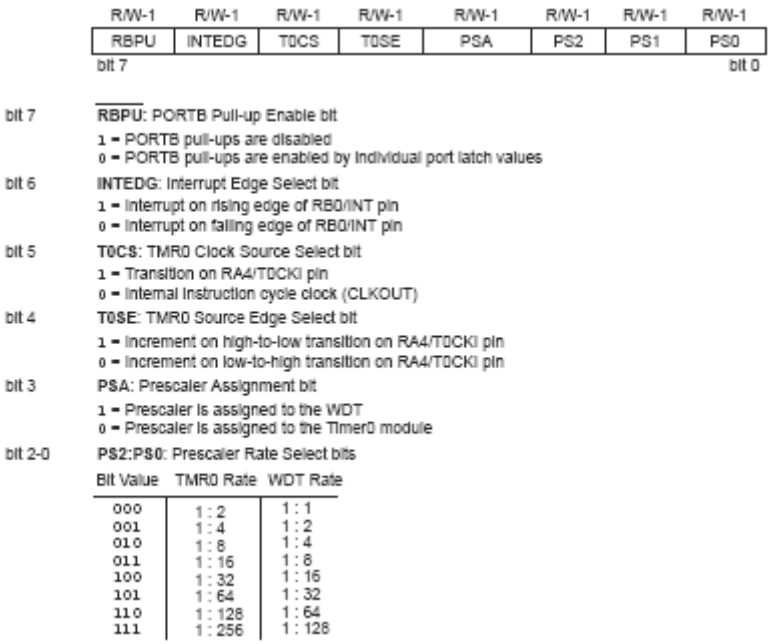
figura 6 – Struttura del registro *Intcon*

I bit che interessano il sistema in esame sono il bit 7 (*GIE – Global Interrupt Enable bit*), il bit 4 (*INTE – RB0/INT External Interrupt Enable bit*) e il bit 5 (*TOIE – TMR0 Overflow Interrupt Enable bit*). Il bit 7 consente di abilitare o disabilitare tutte le interrupt presenti nel sistema settandolo rispettivamente a 1 o 0 logico, il bit 4 abilita l'interrupt relativa alla linea RB0 per la lettura dell'encoder incrementale, il bit 5 infine, consente di abilitare l'interrupt relativa al timer interno chiamato TMR0.
Quindi l'inizializzazione delle interrupt avviene nel seguente modo:

```
STATUS   EQU 0x03          ;Stringa STATUS associata all'indirizzo (03)H
Intcon   EQU 0x0B          ;Stringa Intcon associata all'indirizzo (0B)H
OPT      EQU 0x81          ;Stringa OPT associata all'indirizzo (81)H
Init     Bsf STATUS,5      ;Passaggio al Bank 1
         Movlw B'11010111' ;Scrittura valore (11010111)2 nel registro di lavoro (prescaler 1/256
                           TMR0 rate)
         Movwf OPT         ;Copia del valore di W in OPT
         Bcf STATUS,5      ;Ritorno al Bank 0
         Movlw B'11100000' ;Scrittura valore (11010000)2 nel registro di lavoro
         Movwf Intcon      ;Copia del valore di W in Intcon (abilitazione interrupt)
```

Il registro *OPT* serve per impostare il *prescaler* del *TMR0* e altre funzioni come verrà illustrato più dettagliatamente in seguito.

Nella figura 7 viene riportata la struttura interna del registro *Option*.



R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
RBPU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
bit 7							bit 0

bit 7 **RBPU**: PORTB Pull-up Enable bit
1 = PORTB pull-ups are disabled
0 = PORTB pull-ups are enabled by individual port latch values

bit 6 **INTEDG**: Interrupt Edge Select bit
1 = Interrupt on rising edge of RB0/INT pin
0 = Interrupt on falling edge of RB0/INT pin

bit 5 **T0CS**: TMR0 Clock Source Select bit
1 = Transition on RA4/T0CKI pin
0 = Internal instruction cycle clock (CLKOUT)

bit 4 **T0SE**: TMR0 Source Edge Select bit
1 = Increment on high-to-low transition on RA4/T0CKI pin
0 = Increment on low-to-high transition on RA4/T0CKI pin

bit 3 **PSA**: Prescaler Assignment bit
1 = Prescaler is assigned to the WDT
0 = Prescaler is assigned to the Timer0 module

bit 2-0 **PS2:PS0**: Prescaler Rate Select bits

Bit Value	TMR0 Rate	WDT Rate
000	1 : 2	1 : 1
001	1 : 4	1 : 2
010	1 : 8	1 : 4
011	1 : 16	1 : 8
100	1 : 32	1 : 16
101	1 : 64	1 : 32
110	1 : 128	1 : 64
111	1 : 256	1 : 128

figura 7 – Struttura del registro *Option*

Il settaggio del pin 7 del registro *OPT* a 1 logico, comporta la disabilitazione dei circuiti di *pull-up* sulla *porta B*, mentre con i bit 0,1,2 viene settato il *prescaler* relativo al *TMR0*.

Quando viene generata una interrupt l'esecuzione del programma viene bloccata e il *PC* (*Program Counter*) passa alla locazione $(0004)_H$ dove viene prelevata la prima istruzione e

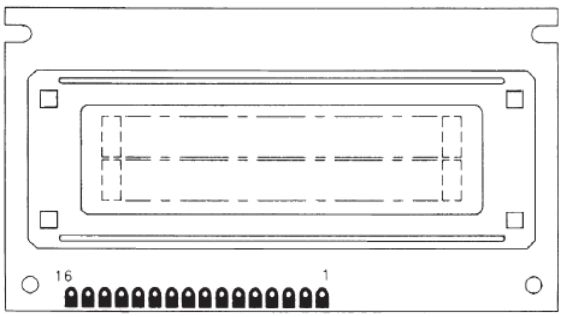
caricata nel registro *IR* (*Instruction Register*) al fine di eseguire il corpo di programma che parte da codesta locazione. La locazione in esame viene identificata dall'etichetta *Int* nel seguente modo:

```
Int     Org 0004h
        Movwf W_Tmp         ;Salvataggio registro di lavoro W
        Swapf  STATUS,0     ;Inversione nibble STATUS con risulato in W
        Clrf   STATUS       ;Passaggio al Bank 0
        Movwf ST_Tmp        ;Salvataggio registro STATUS in ST_Tmp
        Movf   PCLATH,0     ;Trasferimento di PCLATH in W
        Movwf PCL_Tmp       ;Salvataggio registro PCLATH in PCL_Tmp
        Istruzione 1        ;Inizio procedura gestione interrupt
        Istruzione 2
        ……
        Istruzione n        ;Fine procedura gestione interrupt
        Movf   PCL_Tmp,0    ;Trasferimento di  PCL_Tmp in W
        Movwf PCLATH        ;Ripristino registro PCLATH
        Movf   RCREG,0      ;Reset bit interrupt relativo all'interfaccia seriale
        Swapf  ST_Tmp,0     ;Inversione nibble ST_Tmp con risultato in W
        Movwf STATUS        ;Ripristino registro STATUS
        Swapf   W_Tmp,1     ;Inversione nibble con risultato in W_Tmp
        Swapf   W_Tmp,0     ;Ripristino registro di lavoro W
        Bsf     Intcon,7    ;Abilita nuovamente interrupt globale
        Retfie              ;Ritorno da una interrupt
```

dove *Org* non è un'istruzione, ma è una direttiva la quale dice all'assemblatore che la compilazione del codice ha inizio a partire dalla locazione $(0004)_H$. Le istruzioni che precedono e seguono il corpo istruzioni relativo alla procedura di gestione delle interrupt, servono per salvare lo stato attuale dei registri *W*, *STATUS* e *PCLATH*. Questo perché la procedura di gestione delle interrupt modifica certamente lo stato di questi registri e quindi al ritorno dalle interrupt i tre registri risulterebbero corrotti e potrebbero creare problemi per l'esecuzione delle altre procedure. Supponendo, per esempio, che sia in esecuzione una procedura che si trova nella terza pagina della *flash memory* e che utilizza i registri presenti nel *bank 0*. Se ad un certo istante viene generata una interrupt, l'esecuzione del programma passa alla locazione $(0004)_H$ dove si trova la procedura relativa alla gestione della interrupt. Se tale procedura si trovasse per esempio nella

pagina 1 della *flash memory* e utilizzasse i registri del *bank 3*, al ritorno dalla interrupt, la procedura che era stata interrotta in precedenza si troverebbe a lavorare nelle stesse condizioni della procedura di gestione della interrupt (pagina 1 *flash memory* e registri del *bank 3*) creando naturalmente errori e in particolare problemi con la gestione della *stack*[2] (*stack* fuori flusso - *stack overflow*). L'ultima istruzione, serve a far ritornare l'esecuzione del programma nel punto in cui era stato temporaneamente bloccato, mentre l'istruzione *Bsf Intcon,7* abilita tutte le interrupt, le quali vengono disabilitate nel momento in cui ne viene generata una. Per ogni byte ricevuto viene quindi generata un'interrupt al fine di leggerlo, memorizzarlo ed eseguire le operazioni necessarie.

E' presente inoltre, un display LCD 16 caratteri per 2 righe al fine di visualizzare eventuali letture dell'encoder e per l'impostazione dei parametri del controllore PID. Questo è costituito da una serie di pin (esattamente 16) visibili nella figura 8, attraverso i quali è possibile gestirlo. Osservando le caratteristiche tecniche, notiamo che la tensione tipica di funzionamento è di 5 volts mentre, per quanto riguarda i livelli logici, un segnale viene riconosciuto come livello alto se è compreso tra 2,2V e V_{DD} (nel caso in esame $V_{DD} = 5V$), come livello basso se è compreso tra 0V e 0,6V. La temperatura di funzionamento varia tra 0 e 50°C per cui è necessario fare attenzione agli sbalzi di temperatura sul display del dispositivo esterno.



Caratteristiche tecniche

Tensione di Alimentazione	VDD	minima 4,75V
		tipica 5,0V
		massima 5,25V
Tensione di ingresso livello alto	VIH	minima 2,2V
		VDD
Tensione di ingresso livello basso	VIL	minima 0V
		massima 0,6V
Temperatura di funziona-mento	Topr	da 0° a 50° C

figura 8 – Caratteristiche tecniche del display *LCD* (*Liquid Crystal Display*)

Nella figura 9 sono visibili le funzioni dei vari pin. I pin relativi alla alimentazione non hanno bisogno di ulteriori precisazioni, mentre è bene porre l'attenzione sui pin 6÷16. In

[2] Nella *stack* vengono memorizzati tutti gli indirizzi e quindi i punti di ritorno delle chiamate a sub-routine (call …). Quando il registro *IR* (*Instruction Register*) del *PIC* incontra l'istruzione *Return* della sub-routine chiamata, l'esecuzione del programma ritorna all'indirizzo memorizzato dalla *stack* e cioè nel punto in cui era stata chiamata la sub-routine (call …).

particolare il pin 7 (*R/W*) permette di scrivere i dati presenti sulla linea del *BUS*[3] dati se è al livello logico basso oppure di leggerli quando viene posto al livello logico alto.

Il pin 6 (*RS*) invece, se alto –con *R/W* basso- permette di inviare codici di istruzione necessari alla gestione dello stesso (come la cancellazione del display, la visualizzazione del cursore, lo spostamento del cursore in una posizione specifica, lo shift del display, l'impostazione dell'I/O a 4 o 8 bit, ecc.), se basso – con *R/W* basso- consente di inviare una parola dalla memoria dati (*DR – Data RAM*[4]) alla memoria per la generazione dei caratteri (*CGRAM*) al fine di visualizzare sul display i simboli *ASCII* (*American Standard Code for Information Interchange*).

Assegnazione dei segnali sul connettore

Pin-N.	Segnali	Funzione
1	BL+	Terminale di alimentazione LED (+)
2	BL-	Terminale di alimentazione LED (-)
3	GND	Alimentazione (0V)
4	V_{DD}	Alimentazione (5V)
5	V_O	Drive LCD (0V rispetto V_{DD})
6	RS	(Alto) ingresso codici di istruzione (Basso) ingresso dati
7	R/W	(Alto) lettura dati (Basso) scrittura dati
8	E	Seganle di abilitazione
9	DB_0	Linea di bus dati
10	DB_1	
11	DB_2	
12	DB_3	
13	DB_4	
14	DB_5	
15	DB_6	
16	DB_7	

Selezione dei registri

RS	R/W	Operazioni
0	0	IR scrive operazioni interne (come cancella display ecc.)
0	1	Legge il flag occupato (DB_7) e indirizza il counter (DB_0 - DB_6)
1	0	DR scrive operazioni interne (da DR a DD oppure CG RAM)
1	1	DR legge operazioni interne (da DR oppure CG RAM a DD)

figura 9 – Assegnazione dei segnali sul connettore e selezione dei registri.

Se *RS* è basso e *R/W* è alto, è possibile leggere lo stato del display attraverso la linea *DB7*: se è alta vuol dire che il display sta eseguendo operazioni interne e quindi non è pronto a ricevere ulteriori istruzioni o dati, mentre se è bassa è pronto a ricevere informazioni. Se *RS* e *R/W* sono entrambi alti è possibile leggere il dato- istruzione presente nella memoria dati (*DR*) oppure il codice binario associato ad un carattere visualizzato sul display

[3] Insieme di linee che trasportano le informazioni tra due o più dispositivi.
[4] *Random Access Memory* - Memoria ad accesso casuale.

(*DDRAM - Display Data RAM*).

L'invio di una istruzione o di un dato deve essere eseguita rispettando la seguente sequenza:

1. Impostazione dei pin 6 (*RS*) e 7 (*R/W*) per la lettura/scrittura di dati o istruzioni;
2. Presentazione del dato o del codice istruzione sulla linea *BUS* dati;
3. Invio di un impulso con una durata minima di *450ns* sul pin 8 (*Enable* - segnale di abilitazione del display).

L'intervallo di tempo minimo tra due abilitazioni successive del pin 8 è di 1 µs, tenendo presente che tale ciclo di abilitazione dipende anche dal tempo di esecuzione delle operazioni interne al display. L'interfaccia utilizzata nel controllo del motore è a 4 bit al fine di risparmiare linee di *I/O* del *PIC* per altre funzioni che andrò ad analizzare in seguito. L'inizializzazione del display per il funzionamento a 4 bit avviene nel seguente modo:

RS	R/W	DB7	DB6	DB5	*DB4*
0	*0*	*0*	*0*	*1*	*0*

Invio segnale di enable (pin 8)

La pulitura del display (display clear) mediante la sequenza:

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	0	0

Invio segnale di enable (pin 8)

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	0	1

Invio segnale di enable (pin 8)

Inoltre è possibile incrementare o decrementare una posizione nella *DDRAM*, quando viene scritto o letto un codice carattere dalla stessa:

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	0	0	0

Invio segnale di enable (pin 8)

RS	R/W	DB7	DB6	DB5	DB4
0	0	0	1	I/D	0

Invio segnale di enable (pin 8)

Se *I/D* = 1 si ha un incremento, se *I/D* = 0 si ha un decremento del cursore. Per inviare un dato è dunque necessario dividere il codice carattere a 8 bit in due *nibble* da 4 bit ed inviarli in sequenza, uno dopo l'altro nel seguente modo:

RS	R/W	DB7	DB6	DB5	DB4
1	0	Bit7	Bit6	Bit5	Bit4

Invio segnale di enable (pin 8)

RS	R/W	DB7	DB6	DB5	DB4
1	0	Bit3	Bit2	Bit1	Bit0

Invio segnale di enable (pin 8)

dove i *bit7, bit6, bit5, bit4*, rappresentano il *nibble* superiore, mentre i *bit3, bit2, bit1, bit0* rappresentano quello inferiore. Si noti che in questo caso *RS* è settato ad 1 in quanto si trasmette un dato e non una istruzione, mentre *R/W* rimane a 0 logico in quanto sto scrivendo nella memoria del display.

Per fare un esempio numerico suppongo di dover inviare il codice *ASCII* $(61)_H$ che corrisponde alla vocale "a": la conversione in binario di 61 (esadecimale) si ottiene eseguendo la conversione delle singole cifre, raggruppando poi i gruppi di 4 bit (*nibble*):

6h				1h			
0	1	1	0	0	0	0	1
nibble superiore				*nibble inferiore*			

Per cui la sequenza di codici da inviare, sarà il seguente:

RS	R/W	DB7	DB6	DB5	DB4
1	0	0	1	1	0

Invio segnale di enable (pin 8)

RS	R/W	DB7	DB6	DB5	DB4
1	0	0	0	0	1

Invio segnale di enable (pin 8)

Le linee *DB7, DB6, DB5, DB4* sono controllate dalle linee *RB7, RB6, RB5, RB4* del *PIC* che saranno configurate, come illustrato in precedenza, come linee di uscita. La regolazione del contrasto avviene attraverso un trimmer da 4.7 KΩ collegato come in figura 10.

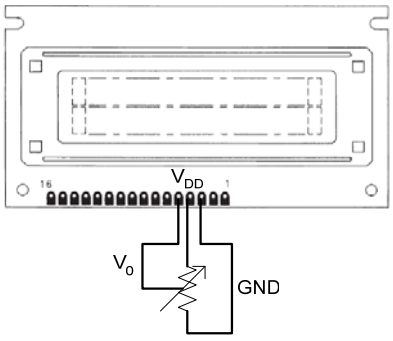


figura 10 – Collegamento del trimmer resistivo per la regolazione del contrasto.

Per quanto riguarda lo stadio di uscita, ho fatto uso di un integrato prodotto dalla National Semiconductor e cioè un LMD18200 in grado di pilotare carichi fino a 3 A. Per quanto riguarda il motore, si tratta di un motore DC prodotto dalla PITTMAN, i cui dati di targa sono riportati nella figura 11.

Line No.	Parameter	Symbol	Units	9X34			
58	Reference Voltage	E	V	12.0	19.1	24.0	30.3
59	Torque Constant	K_T	oz.in/A	(18.2 X 10^{-3})	(28.7 X 10^{-3})	(36.5 X 10^{-3})	(45.9 X 10^{-3})
60	Back-EMF Constant	K_E	V/krpm (V/rad/s)	1.91 (18.2 X 10^{-3})	3.01 (28.7 X 10^{-3})	3.82 (36.5 X 10^{-3})	4.81 (45.9 X 10^{-3})
61	Resistance	R_T	Ω	0.83	1.89	2.96	4.62
62	Inductance	L	mH	0.63	1.56	2.51	3.97
63	No-Load Current	I_{NL}	A	0.33	0.21	0.16	0.13
64	Peak Current (Stall)	I_P	A	14.5	10.1	8.11	6.55

figura 11 – Dati di targa del motore PITTMAN modello GM9434

I dati riportati nella figura 11 sono stati utilizzati per le simulazioni in ambiente MATLAB e SIMULINK e in particolare sono stati utilizzati la costante di coppia, la costante elettrica, l'induttanza, la resistenza e la tensione di alimentazione. Lo schema SIMULINK per il controllore in questione è riportato nella figura 12.

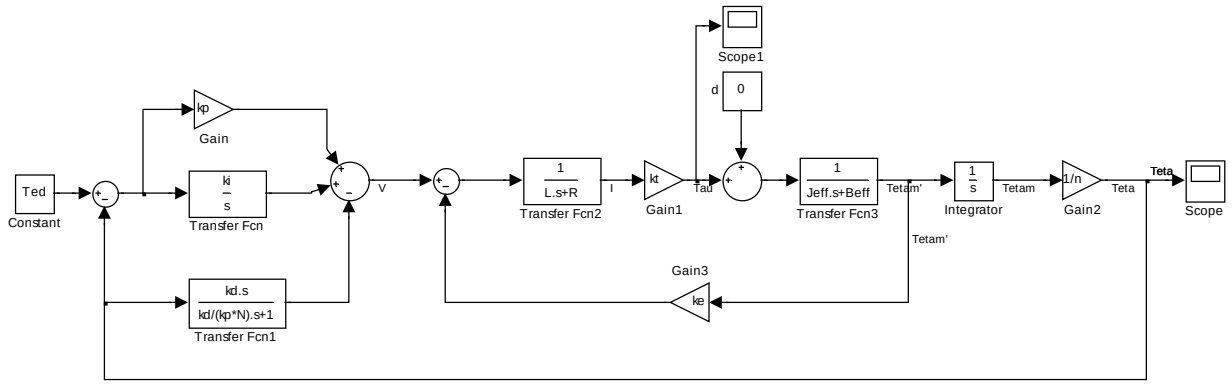


figura 12 – Modello Simulink del controllore PID e del motore DC

Come si può notare dalla figura 12, il disturbo sulla coppia fornita dal motore viene assunto pari a zero in quanto la lancetta, essendo di massa trascurabile, non influisce in modo considerevole sul controllore dal punto di vista gravitazionale. I valori limiti delle costanti del controllore sono stati calcolati con il metodo di Ziegler-Nichols. Il metodo in questione consiste nel porre a zero le costanti dei controllori integrale e derivativo e

aumentare il valore della costante del controllore proporzionale fino a quando nel sistema si innesca un'oscillazione permanente. A quel punto, si ricava il periodo di oscillazione a partire dalla conoscenza della pulsazione di attraversamento e il valore della costante Kp_max per il quale è avvenuta l'oscillazione. Da questi ultimi valori si ricavano, mediante l'uso della tabella di Ziegler-Nichols, i valori delle costanti del controllore integrale e derivativo. Dai dati di targa del motore, mediante l'utilizzo di MATLAB, si ottiene una pulsazione di attraversamento pari a:

$$\omega_t = 18.3776 \text{ rad/sec}$$

e quindi il periodo di oscillazione è dato da:

$$T_0 = (2 * \pi) / \omega_t = 0.3419 \text{ sec}$$

$$Kp\_max = 2.1488e+003$$

Dalla tabella di Ziegler-Nichols si ottengono quindi i valori delle costanti del controllore PID:

$$K\_pro = Kp\_max * 0.6 = 1.2893e+003$$

$$T\_d = 0.125 * T\_0 = 0.0427 \text{ sec}$$

$$K\_der = T\_d * K\_pro = 55.0998$$

$$T\_i = 0.5 * T\_0 = 0.1709 \text{ sec}$$

$$K\_int = K\_pro/T\_i = 7.5422e+003$$

Con questi parametri la risposta del sistema è riportata nella figura 13 e come si può vedere il comportamento è oscillatorio.

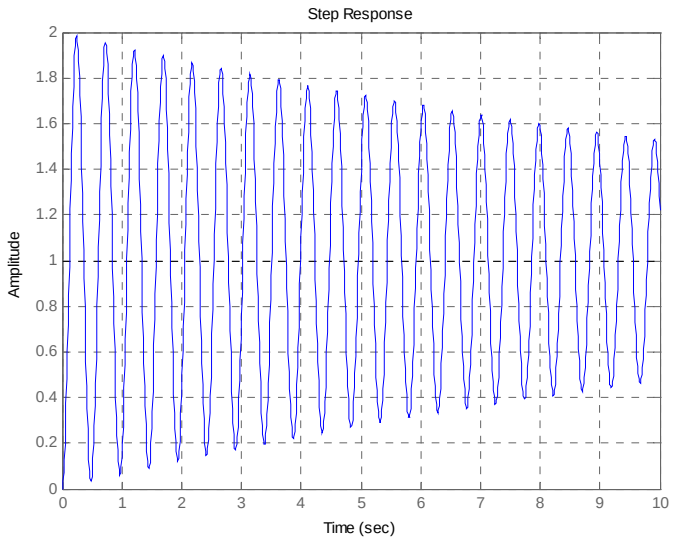


figura 13 – Comportamento oscillatorio del sistema

Il file con l'inizializzazione dei parametri del modello Simulink è riportato qui di seguito:

```
%Controllo motore DC
%Assegnazione parametri
L = 0.63*(10.^-3) %induttanza del circuito di armatura
R = 0.83 %resistenza del circuito di armatura
kt = 18.2*(10.^-3)* 0.2780139 * 2.54*(10.^-2) %costante di coppia del motore (conversione oz.in-->N.m)
ke = 21.3*(10.^-3) %costante di velocità del motore
n = 19.7 %rapporto di riduzione
Ted = -10 %angolo desiderato
kp = 255 %costante controllore P
ki = 6 %costante controllore I
kd = 70 %costante controllore D
N = 100 %costante polo lontano
Jm = 5e-5 %momento d'inerzia del motore
l=0.01;     %lunghezza della lancetta
ml=0.01;    %massa della lancetta
Jl=(ml*l.^2)/3 %momemnto di inerzia della lancetta sull'asse del motore
Bm = 1e-5 %coefficiente di attrito viscoso del motore
Bl = 0 %coefficiente di attrito viscoso del carico
Jeff=Jm+Jl/n.^2 %momento di inerzia complessivo del sistema
Beff=Bm         %coefficiente di attrito viscoso complessivo del sistema
```

In realtà, poiché l'angolo desiderato varia in un range che va da -359° a + 359°, il valore di Kp è stato ricavato mediante simulazioni Simulink con i seguenti range di valori che mantengono il sistema, e quindi la coppia del motore, sotto i limiti consentiti.

Impostazione controllore P(ID) in funzione dell'angolo desiderato		
Kp_{min}	θ_d	Kp_{MAX}
140	1°	255
70	2°÷9°	190
15	10°÷14°	120
10	15°÷24°	70
6	25°÷49°	35
3	50°÷74°	24
1	75°÷99°	18
1	100°÷149°	12
1	150°÷224°	8
1	225°÷359°	5

Riferendomi, per esempio al primo range, con un angolo di 5° e Kp = 160 (con Ki e Kd al limite massimo; Ki = 6, Kd = 70) si ha la risposta riportata in figura 14.

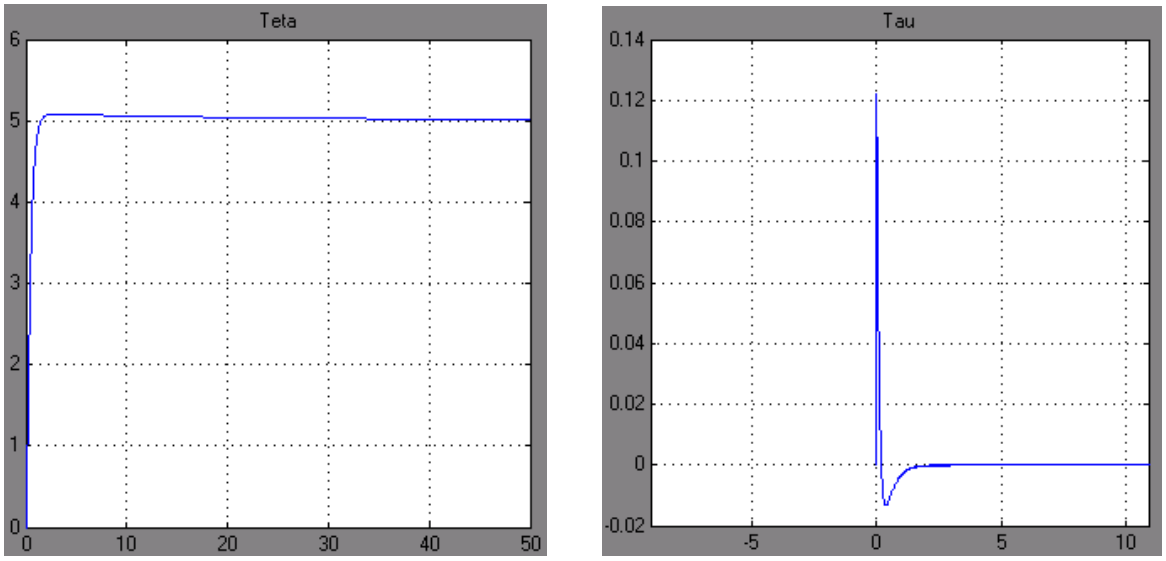


figura 14 – Risposta del sistema con costanti nel range di stabilità per $\theta_d = 5$

Imponendo invece, Kp = 20 (con i valori degli altri parametri uguali a quelli del caso precedente) e quindi un valore fuori range si ottiene la risposta di figura 15.

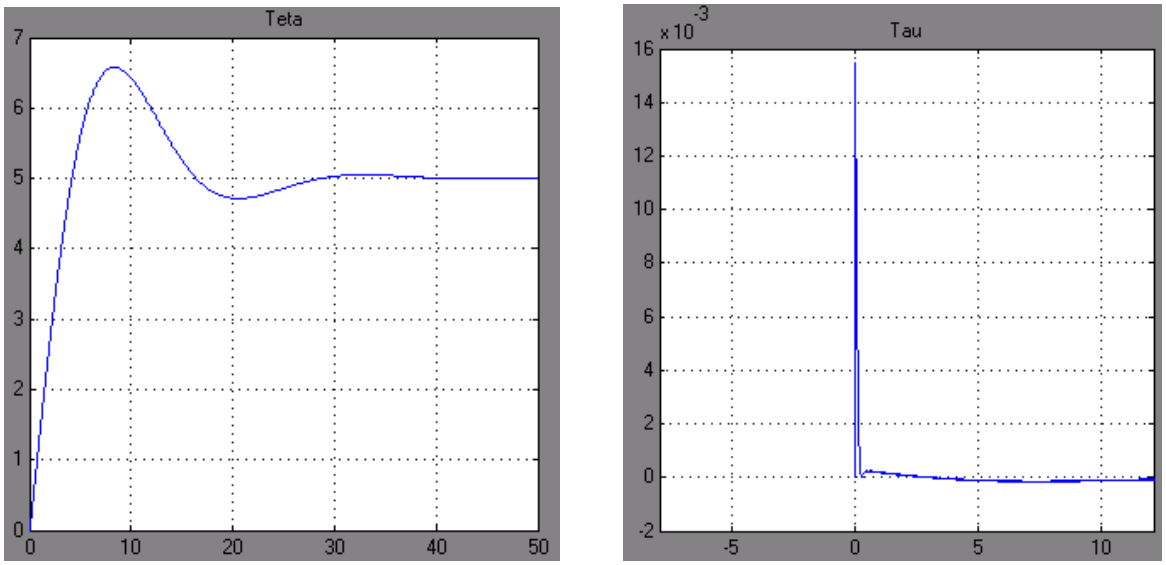


figura 15 – Comportamento del sistema per valori della costante Kp fuori range

Per quanto riguarda l'implementazione del controllore PID digitale, ho utilizzato la forma discretizzata con derivate all'indietro:

$$PID(n) = Kp * E(n) + Ki * Ts * \sum_{0}^{N} E(n) + \{Kd * [E(n) - E(n-1)]\} / Ts$$

La lettura dell'encoder avviene mediante il PIC sulla linea B0 della porta B. In particolare ad ogni impulso inviato dall'encoder viene generata un'interrupt che sposta l'esecuzione del codice presente nel PIC alla locazione 0x04h. Qui è presente la procedura per l'incremento e/o il decremento della variabile relativa alla posizione (angolo) della lancetta. Poiché la risoluzione dell'encoder è di 500 impulsi/giro e poiché il rapporto di riduzione è pari a 19.7, ogni impulso corrisponde ad un angolo pari a:

$$\theta m = 360/(500*19.7) = 0{,}0365482233°$$

E' chiaro che per rappresentare dei numeri così piccoli i registri a 8 bit presenti nel PIC non sono sufficienti. A tal proposito ho utilizzato diversi registri insieme al fine di avere "virtualmente" registri più grandi di 8 bit e realizzare così una rappresentazione binaria in virgola fissa con 1 bit di segno, 9 bit per la parte intera e 22 bit per la parte frazionaria.

Ad ogni impulso che viene generato sulla linea RB0 del PIC, viene controllato il livello presente sulla linea A0, al fine di verificare se si tratta di una rotazione oraria o antioraria come si può vedere nella figura 16. Quello che avviene internamente al PIC è che le variabili di aggiornamento dell'encoder non vengono incrementate ad ogni impulso proveniente sulle linee RB0 e RA0, ma si utilizzano altre due variabili interne in modo tale da aggiornare la posizione ogni 28 impulsi, cioè:

$$0{,}0365482233° * 28 = 1{,}0233502524°$$

che rappresenta il passo di incremento o decremento della posizione.

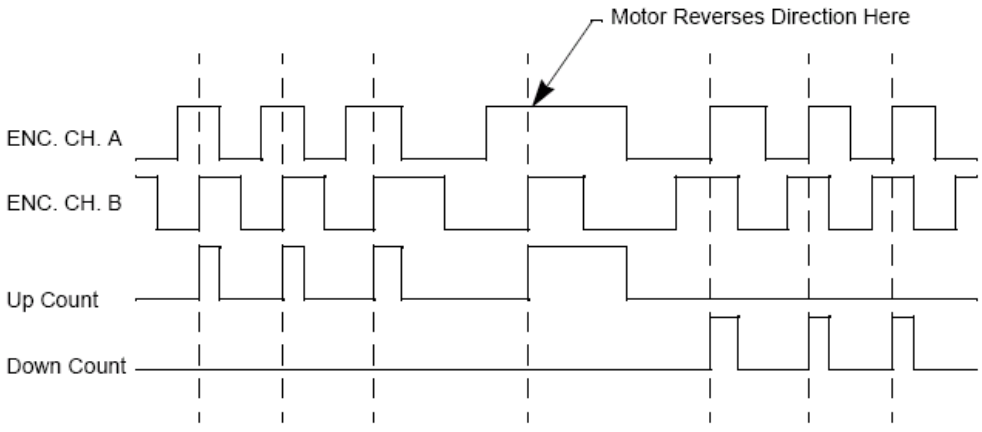


figura 16 – Schema temporale dei segnali che arrivano dall'encoder incrementale

Se si tratta di un incremento, alla variabile di memorizzazione dell'encoder (4 registri da 8 bit – rappresentazione 10.22 bit) vengono aggiunti sequenzialmente i seguenti valori:

$$1 + 2^{(-6)} + 2^{(-7)} - 2^{(-14)} - 2^{(-15)}$$

che corrisponde al valore decimale frazionario 1,0233459 che rappresenta una buona approssimazione del valore reale precedentemente ottenuto. Se invece, si tratta di un decremento, vengono sommati i valori complementari del caso precedente e cioè:

$$-1 - 2^{(-6)} - 2^{(-7)} + 2^{(-14)} + 2^{(-15)}$$

Praticamente il metodo utilizzato per convertire il numero frazionario decimale in binario è quello dei moltiplicatori. Quello che ho fatto è prendere la parte frazionaria del numero da rappresentare ed eseguire i seguenti passi:

1. Moltiplica il numero frazionario per 2
2. Salva la cifra intera del risultato ottenuto (cifre binarie rappresentative)
3. Se il risultato è maggiore di uno sottrai uno e ritorna al punto 1, altrimenti ritorna al punto 1.

Riporto l'esempio per i primi n bit del numero relativo al singolo impulso dell'encoder, cioè 0,03654822335:

0,03654822335 * 2 = 0,0730964467

0,0730964467 * 2 = 0,1461928934

0,1461928934 * 2 = 0,2923857868

0,2923857868 * 2 = 0,5847715736

0,5847715736 * 2 = 1,1695431472

(1,1695431472-1) * 2 = 1,1695431472

e così via per 22 volte (rappresentazione 10.22 bit)

Le cifre evidenziate in rosso rappresentano il numero in base 2 (00011…) e come si può notare, il numero 0,03654822335 si ottiene come somma di potenze di due mentre in precedenza ho considerato anche sottrazioni di potenze di due. In realtà, eseguendo solo somme per incrementare la variabile, durante il decremento dovrei eseguire solo sottrazioni e, poiché la sottrazione è sensibilmente più lenta della somma dal punto di vista hardware, ho diviso le operazioni di somma e sottrazione tra l'incremento e il decremento della variabile di misura della posizione.

Quindi ritornando alla forma discretizzata del controllore PID, ciò che fa il PIC durante il funzionamento è quello di calcolare il valore dell'errore attuale E(n) come differenza tra il valore impostato dall'utente mediante input da tastiera sul display e il valore letto dall'encoder incrementale. A questo punto viene sommato l'errore attuale E(n) all'errore

precedente E(n-1) e il risultato viene moltiplicato per la costante del controllore integrale e per la frequenza di campionamento (Fs = 1/Ts). Calcolato il valore del controllore integrale, si passa al calcolo del controllore derivativo eseguendo in un primo momento la differenza tra l'errore attuale E(n) e l'errore precedente E(n-1). Il valore così ottenuto viene moltiplicato per il valore della costante Kd e diviso per la frequenza di campionamento. Infine viene calcolato il controllore proporzionale mediante il prodotto della costante Kp e dell'errore attuale. Quando tutti i valori intermedi sono pronti, questi vengono sommati al fine di calcolare il controllore PID complessivo. A questo punto viene salvato l'errore attuale nelle variabili dell'errore precedente al fine di eseguire un nuovo calcolo e i 10 bit più significativi della parte intera del controllore PID (risultato a 40 bit dei quali gli ultimi 22 bit rappresentano la parte frazionaria) vengono utilizzati per pilotare il modulo PWM (Pulse Width Modulation) presente all'interno del PIC stesso. La risoluzione del PWM è appunto di 10 bit e questi vengono utilizzati per impostare il duty cicle degli impulsi in uscita al PWM mentre la frequenza del PWM è impostata tramite software (e quindi fissa) a 50 KHz. Nella figura 17 viene riportato l'algoritmo che, tradotto in assembler, viene eseguito dal PIC durante il funzionamento. E' chiaro che se l'errore calcolato all'inizio della procedura è nullo, non è necessaria alcuna azione da parte del controllore e la parte di codice relativa al calcolo del PID viene tralasciata. Per quanto riguarda il periodo e quindi la frequenza di campionamento, questa viene impostata via software in fase di programmazione e gestita mediante un timer a 8 bit presente nel PIC (TMR0).

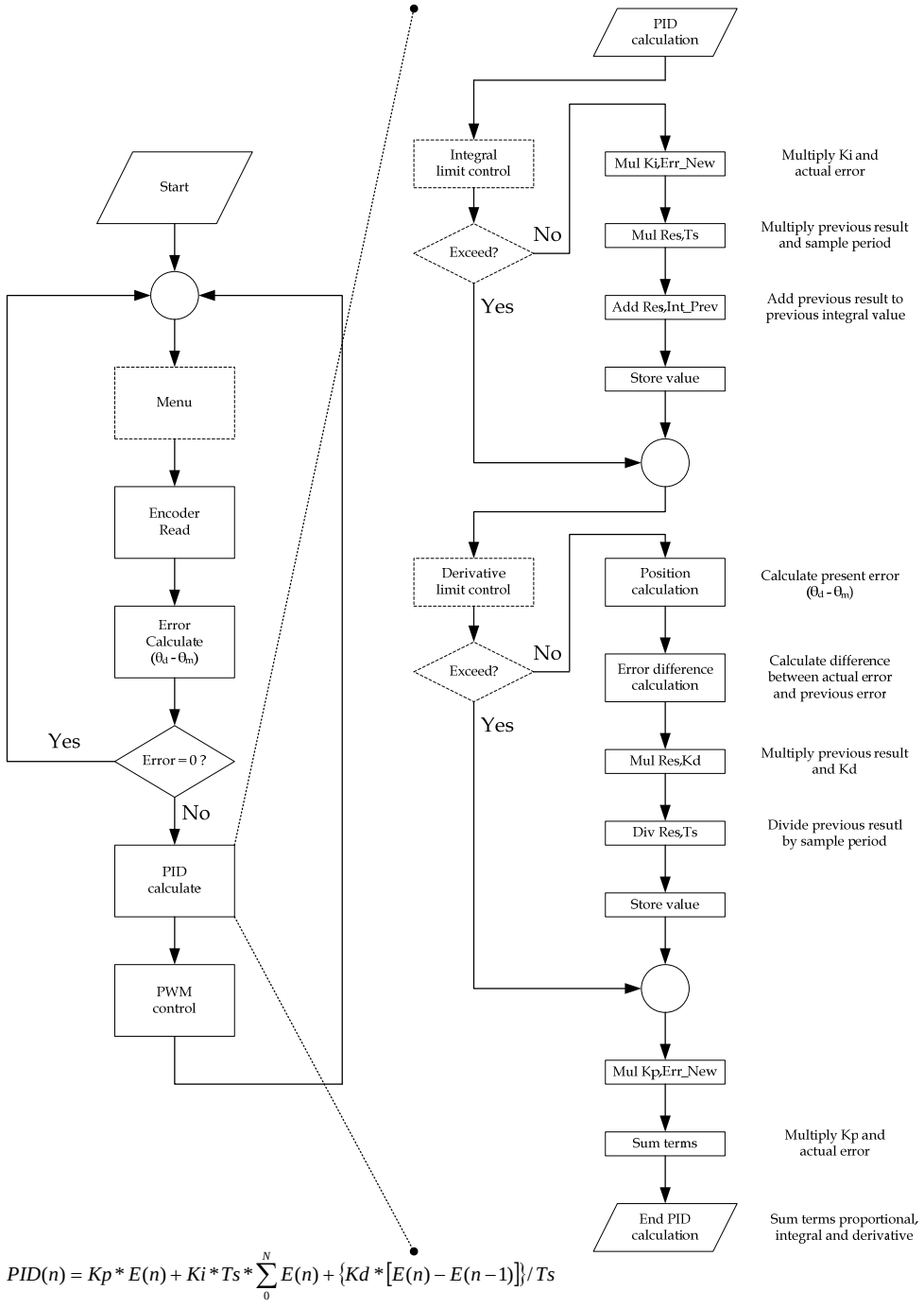


figura 17 – Algoritmo di esecuzione per il calcolo del controllore PID

In particolare, al TMR0, viene associato un prescaler che appunto consente di scalare la frequenza, secondo rapporti definiti su potenze di 2, rispetto al periodo dell'oscillatore principale di sistema (quarzo esterno). Quindi in base all'impostazione del prescaler, il TMR0, di volta in volta viene incrementato. Quando si ha overflow, cioè quando il TMR0 passa dal valore 0xFFh al valore 0x00h viene generata un'interrupt che fa porta nuovamente l'esecuzione del codice alla locazione 0x04h. Qui viene settato un bit di

controllo che viene utilizzato, in un secondo momento, per l'avvio della procedura relativa al calcolo del controllore PID. Per quanto riguarda l'interfaccia utente, questa è composta essenzialmente da tre micro switch e da un display LCD retroilluminato. Mediante gli switch è possibile accedere al menu al fine di impostare le costanti del controllore PID, impostare l'angolo desiderato di posizionamento e infine leggere la posizione attuale della lancetta. Per fare questo in particolare, essendo i numeri rappresentati in virgola fissa nella forma 10.22 bit, è stato necessario convertire tale formato binario nel formato BCD. Per quanto riguarda la parte intera il problema è stato superato facilmente mediante una semplice procedura assembler. Per la parte frazionaria invece, ho deciso di visualizzare solo sei cifre dopo la virgola. Per fare questo, moltiplico il numero a 32 bit (10.22 bit) per 10^6, dopodichè considero solo la parte intera del risultato. Il valore così ottenuto viene diviso per 10, per 6 cicli e ad ogni divisione il resto rappresenta il codice BCD della cifra da rappresentare sul display. Nella figura 18 viene riportata la struttura ad albero del menu per l'impostazione delle costanti del controllore PID, dell'angolo desiderato e per la lettura della posizione della lancetta.

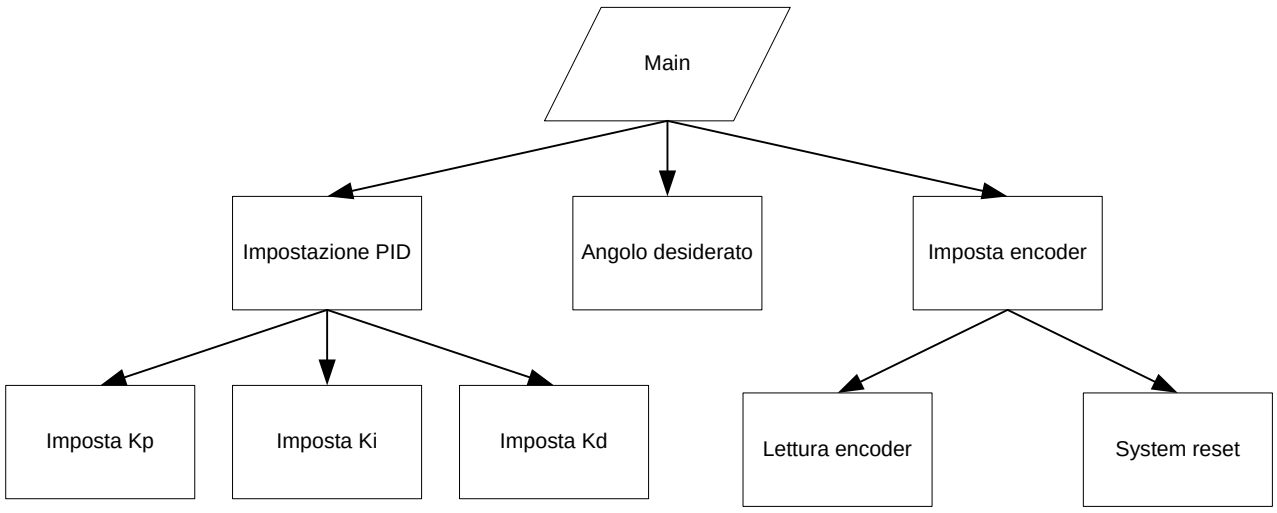


figura 18 – Struttura del menu per l'impostazione e la lettura dei parametri del controllore

L'angolo desiderato può essere impostato tramite gli switch in un range che va da -359° a +359°. Per quanto riguarda lo stadio di potenza, questo è realizzato mediante l'integrato LMD18200 il cui schema interno e la relativa piedinatura sono riportati nella figura 19.

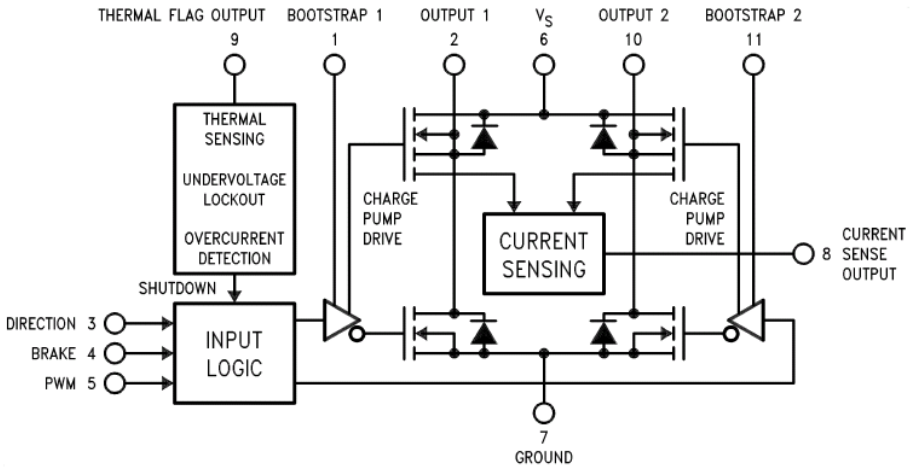


figura 19 – Struttura interna dello stadio di potenza

Come si può osservare è presente uno stadio di potenza a ponte H, un circuito di protezione per il surriscaldamento eccessivo e una logica di controllo del ponte H. L'integrato in questione può pilotare il motore DC secondo due modalità:

1. In base al solo pin DIRECTION (con PWM alto), cioè valutando l'ampiezza del duty cicle; se questo è maggiore del 50% il motore si muove in una direzione con una velocità che dipende sempre dall'ampiezza del duty cicle; se è minore del 50% il motore si muove nella direzione opposta e se è pari al 50% il motore resta fermo.
2. Facendo uso dei pin "DIRECTION" e "PWM" come avviene nel controllore PID in questione. La direzione di rotazione viene dunque impostata mediante il pin DIRECTION, mentre la velocità dall'ampiezza del duty cicle.

Il pin "BRAKE" consente di bloccare il motore in una posizione fissa facendo si che si attivino o i transistor di source (transistor in alto al ponte H) o quelli di sink (transistor in basso al ponte H) contemporaneamente. Nella figura 20 viene riportata la tavola logica per il controllo dello stadio di potenza.

PWM	Dir	Brake	Active Output Drivers
H	H	L	Source 1, Sink 2
H	L	L	Sink 1, Source 2
L	X	L	Source 1, Source 2
H	H	H	Source 1, Source 2
H	L	H	Sink 1, Sink 2
L	X	H	NONE

figura 20 – Livelli logici dei segnali per il controllo dello stadio di potenza

Nella figura 21 viene introdotto lo schema di principio del circuito di potenza, il cosiddetto ponte ad H:

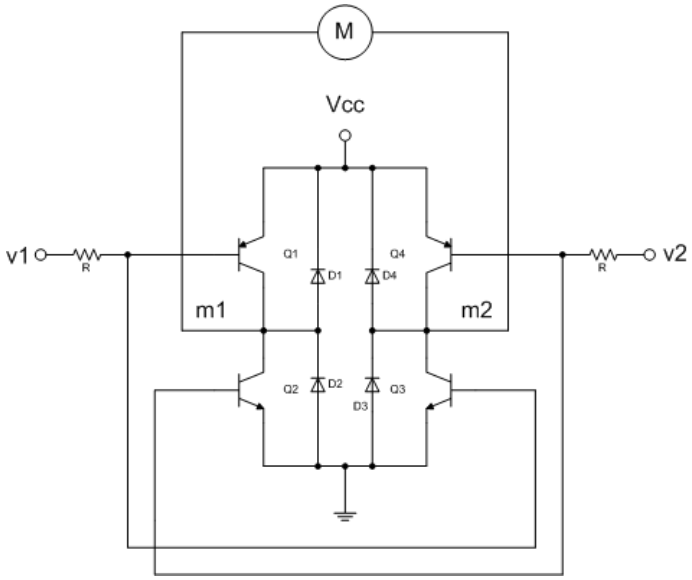


figura 21 – Schema elettrico del ponte H

Come si può vedere dallo schema, se v1 = 5 V e v2 = 0, i transistor Q1 (source 1),Q3 (sink 2) sono in conduzione, mentre Q2 (sink 1), Q4 (source 2) sono interdetti per cui la fase del motore risulta polarizzata con un certo verso. Se invece, v1 = 0 e v2 = 5 V, i transistor Q1,Q3 sono interdetti, mentre Q2,Q3 sono in conduzione e la polarizzazione sulla fase risulta invertita. Le resistenze R le ho utilizzate per limitare la corrente nella giunzione base-emettitore dei transistor. Nella figura 22 infine, viene riportato il layout del circuito stampato comprendente il PIC16F876, lo stadio di potenza LMD18200, il display LCD e tutti i componenti al contorno.

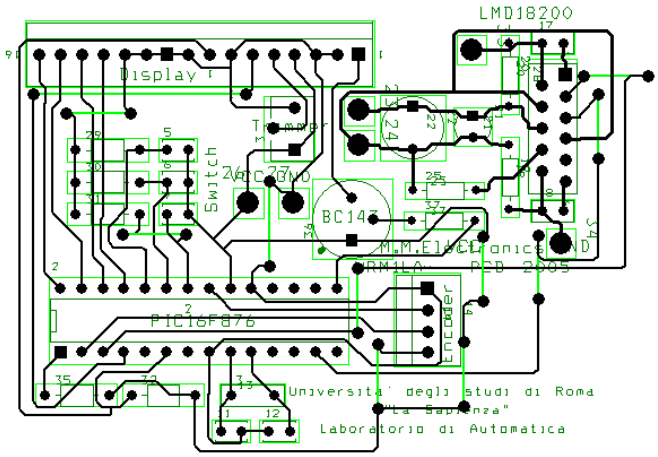


figura 22 – Circuito stampato del controllore digitale