REALIZZAZIONE DI UN CONTROLLO PID SU MICROCONTROLLORE PIC 18F2520

Laboratorio di Strumentazione elettronica di misura 2006
Toss Viviana

Il programma realizzato permette di controllare la velocità di un motorino tramite un algoritmo PID implementato su un microcontrollore, nello specifico il PIC18F2520 della Microchip. Per realizzarlo ci si è avvalse del compilatore PCW della medesima casa produttrice compreso nel pacchetto MPAB IDE versione 6. Le funzioni utilizzate sono tutte disponibili nella libreria del compilatore. Il microcontrollore è stato montato sulla propria demoboard PICDEM 2 PLUS alla quale è stato collegato il motorino. Il programma si prefigge di stabilire la velocità alla quale gira il motorino e di controllarla tramite un PWM che influisce sull'alimentazione del motore stesso. In questo modo è possibile agire sul PWM per raggiungere la velocità stabilita dall'utente (setpoint). Nella versione realizzata è possibile variare il valore del setpoint, il valore delle costanti presenti nell'algoritmo e il tipo di algoritmo utilizzato per il controllo. Sono proposti quattro algoritmi diversi, selezionabili tramite un pulsante esterno: proporzionale-integrativo-derivativo, proporzionale, integrativo, derivativo. Nei seguenti capitoli è stato riportato:

- diagramma di flusso del programma
- codice commentato del programma (per ulteriori specifiche sulle funzioni utilizzate fare riferimento alla libreria PCW)
- sezione dedicata a come è acquisita la frequenza del motore
- sezione esplicativa su come si è ottenuto un PID a tempo costante, ossia che agisce sul motore con una frequenza indipendente dalla velocità alla quale il motore gira
- sezione sull'attuazione del processo PWM

COLLEGAMENTI E PORTE UTILIZZATE DAL PIC18F2520

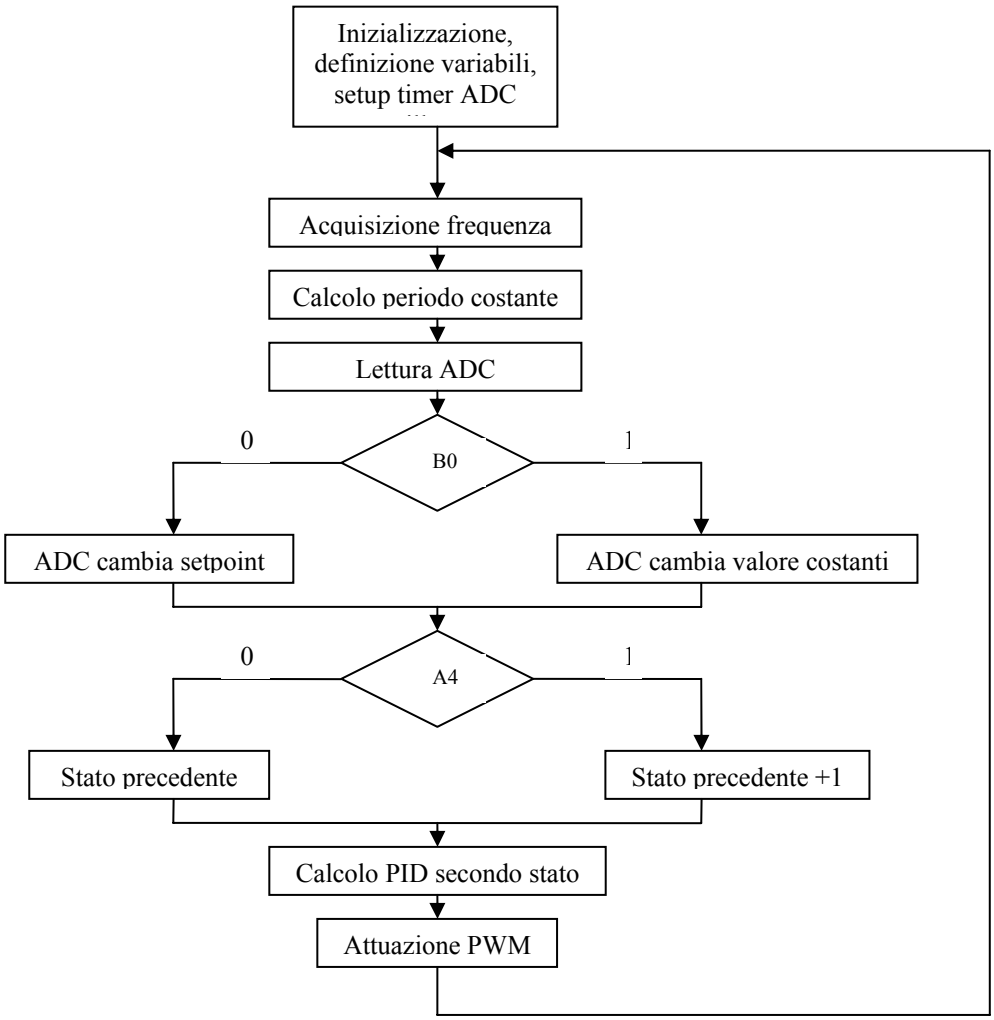
PIN A0	ingresso:	porta analogica (valore setpoint e costanti algoritmi)
PIN A4	ingresso:	pulsante scelta modalità (PID-P-I-D)
PIN B0	ingresso:	pulsante cambio valore costanti
PIN B1	uscita:	led stato
PIN B2	uscita:	led stato
PIN B3	uscita:	led accensione
PIN C0	ingresso:	segnale encoder proveniente da motore
PIN C2	uscita:	segnale PWM per controllare alimentazione motore

DIAGRAMMA DI FLUSSO

Il programma è costituito da una parte di inizializzazione e da cinque blocchi essenziali:

- acquisizione della frequenza alla quale il motore sta girando
- routine di delay per il mantenimento del periodo costante
- lettura ADC e controllo dello stato dei pulsanti
- calcolo dell'algoritmo selezionato
- attuazione PWM

Il periodo con il quale i blocchi ciclano è determinato all'interno del programma ed è facilmente variabile agendo sulla costante PID_TIME_CYCLE.



CODICE COMMENTATO

Parte di inizializzazione

```
#FUSES NOWRT            //Program memory not write protected
#FUSES NOWRTD           //Data EEPROM not write protected
#FUSES IESO             //Internal External Switch Over mode enabled
#FUSES FCMEN            //Fail-safe clock monitor enabled
#FUSES PBADEN           //PORTB pins are configured as analog input channels on RESET
#FUSES NOWRTC           //configuration not registers write protected
#FUSES NOWRTB           //Boot block not write protected
#FUSES NOEBTR           //Memory not protected from table reads
#FUSES NOEBTRB          //Boot block not protected from table reads
#FUSES NOCPB            //No Boot Block code protection
#FUSES LPT1OSC          //Timer1 configured for low-power operation
#FUSES MCLR             //Master Clear pin enabled
#FUSES NOXINST          //Extended set extension and Indexed Addressing mode disabled (Legacy mode)

#use delay(clock=16000000)
```

È abilitata la porta rs232 per poter comunicare con il PC.

```
#use rs232(baud=57600,parity=N,xmit=PIN_C6,rcv=PIN_C7,bits=9)
```

Il bit TMR0ON del registro T0CON serve per abilitare e disabilitare il timer0.

```
#byte T0CON = 0xFD5
```

Il PIN_C0 rappresenta l'uscita dell'encoder del motore.

```
#define IN_CLK PIN_C0
```

Definizione delle variabili.

```
#define TIMEOUT_RELOAD 3  // 60Hz ovvero 13ms
int16 timeout = TIMEOUT_RELOAD;
int16 tempo_ciclo_ms;
int 16 delay_todo_ms;
int32 tempo;
int16 tempoprec;
int8 ciclipassati=0;
```

} variabili per gestire il periodo costante

```
int16 setpoint,last_setpoint, valoreADC;
```

variabili per gestire il setpoint e la conversione A/D;

```
int16 tics,velocita;
float duty = 0.0;
float last_duty = 0.0;
float fvel =0.0;
```

} definizioni delle variabile per la velocità e il dutycycle

```
int stato=0;
```

indice dello stato in cui si trova il programma

```
float err=0.0;
```

errore tra setpoint e velocità misurata dall'encoder

```
float err1=0.0;
```

errore tra setpoint e velocità misurata dall'encoder alla precedente iterazione

```
float err2=0.0;
```

errore tra setpoint e velocità misurata dall'encoder due iterazioni prima

```
float K_P =0.01;
```

valore della costante proporzionale

```
float K_I =0.001;
```

valore della costante integrativa

```
float K_D =0.001;
```

valore della costante derivativa

Routine di interrupt del Timer1, è eseguita solo quando il Timer1 lancia un interrupt (il conteggio è arrivato a 0). Decrementa la variabile timeout, incrementa la variabile ciclipassati e inverte lo stato del led B3.

```
#int_TIMER1
TIMER1_isr(){
        ciclipassati ++;
        output_toggle(PIN_B3);

       if(timeout)
              timeout--;
}
```

Inizio del programma e inizializzazioni del microcontrollore

```
void main()
{

  setup_adc_ports(NO_ANALOGS|VSS_VDD);
  setup_adc(ADC_OFF|ADC_TAD_MUL_0);
  setup_spi(FALSE);
  setup_wdt(WDT_OFF);
  setup_timer_0(RTCC_INTERNAL);
  setup_timer_1(T1_DISABLED);
  setup_timer_2(T2_DISABLED,0,1);
  setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  setup_low_volt_detect(FALSE);
```

Setup oscillatore interno a 4MHz con opzione PLL (moltiplicatore di frequenza x4). Si ottiene una frequenza del processore di 16MHz.

```
  setup_oscillator(OSC_4MHZ|OSC_NORMAL|OSC_31250|OSC_PLL_ON);  //clock CPU a 16Mhz
```

Abilitazione porta analogica AN0

```
  setup_adc_ports(AN0|VSS_VDD);
  setup_adc(ADC_CLOCK_DIV_64|ADC_TAD_MUL_0);
  setup_spi(FALSE);
  setup_wdt(WDT_OFF);
```

Setup del Timer0, utilizzato per misurare il tempo trascorso tra due fronti di salita del segnale di encoder. Si basa sull'oscillatore interno.

```
  setup_timer_0(RTCC_INTERNAL);
```

Setup del Timer1, utilizzato per mantenere il periodo tra due misure costante. È impostato in modo da ottenere un interrupt di fine conteggio ogni 16ms.

```
  setup_timer_1(T1_INTERNAL|T1_DIV_BY_1);      //T1 16ms interrput (60Hz)
```

Setup del Timer2, utilizzato nella generazione del segnale PWM che controlla la velocità del motore. È impostata la risoluzione a 10bit che permette di variare il dutycycle tra 0 e 1023.

```
  setup_timer_2(T2_DIV_BY_16,255,1);          //PWM 977Hz 10bit res
```

Setup di altre opzioni

```
  setup_timer_3(T3_DISABLED|T3_DIV_BY_1);
  setup_ccp1(CCP_PWM);
  setup_comparator(NC_NC_NC_NC);
  setup_vref(FALSE);
  setup_low_volt_detect(FALSE);
  input(IN_CLK);
  setup_uart(57600);
```

Messaggio di partenza del PID

```
  printf("\r\n\r\n");
    printf("HELLO TEST PID MOTOR CONTROL\r\n");
```

Abilitazione degli interrupts

```
enable_interrupts(INT_TIMER1);
enable_interrupts(GLOBAL);
```

Lettura e conversione ADC per impostare il valore di setpoint. La funzione read_adc() restituisce un intero nel range 0-1023, viene moltiplicato per ottenere un range di variazione maggiore.

```
setpoint = read_adc() * 3;
last_setpoint = setpoint;
```

Vengono spenti i led B1 e B2.

```
output_bit(PIN_B2, 1);
output_bit(PIN_B1, 1);
```

Inizializzazione del valore di partenza del Timer1

```
set_timer1(0);
tempo=tempoprec=0;
```

Ciclo perenne

```
while(1)
{
```

Si ricarica la variabile timeout.

```
    timeout = TIMEOUT_RELOAD;
```

Si aspetta il fronte di discesa sull'ingresso encoder.

```
    while(input(IN_CLK) && timeout);      //wait for H2L
```

Se il timeout non è avvenuto viene abilitato il timer per il conteggio del periodo tra due eventi sull'ingresso encoder. Per dettagli vedere la sezione dedicata *Acquisizione Frequenza Motore*.

```
    if(timeout)
    {
      T0CON &= 0x7F;      //timer OFF
      set_timer0(0);
      while(!input(IN_CLK) && timeout);   //wait for L2H    1st positive edge
      T0CON |= 0x80;    //timer ON
    }
    if(timeout)
      while(input(IN_CLK) && timeout);      //wait for H2L

    if(timeout)
      while(!input(IN_CLK) && timeout);     //wait for L2H   2nd positive edge
    T0CON &= 0x7F;      //timer OFF
    tics = get_timer0();
```

Se il timeout non è avvenuto la velocità è posta pari al valore del timer0, ossia al conteggio del periodo tra due tacche della ghiera, altrimenti è posta pari a zero.

```
    if(timeout)
      velocita = tics;
    else
      velocita = 0;
```

Viene convertita la velocità in giri del motore al minuto.

```
//          fvel = 60*(1*1000*1000)/((float)velocita * 16.0);//rpm
        fvel = (2*1000*1000)/((float)velocita);//rpm
        fvel /=16;
        fvel *=60;
```

Durante il conteggio del tempo trascorso dal ciclo precedente vengono disabilitati gli interrupt, onde evitare che un interrupt possa avvenire nel mentre delle operazioni ed ottenere un risultato sbagliato.

```
        disable_interrupts(INT_TIMER1);
```

Calcolo del tempo trascorso dalla misura precedente. Per dettagli vedere la sezione dedicata *Pid a Periodo Costante.*

```
        tempo = (int32)get_timer1() + (int32)ciclipassati * 65536;


        tempo_ciclo_ms = tempo / (2*1000);  //tempo ms

        printf("t=%lu-tp=%lu-OWF=%u-tcy=%lu ",tempo,tempoprec,ciclipassati,tempo_ciclo_ms);



        enable_interrupts(INT_TIMER1);
```

È possibile definire la durata del periodo tra due misure cambiando il valore della costante PID_TIME_CYCLE. Il massimo valore impostabile è 255.

```
#define PID_TIME_CYCLE (100) // 100 ms //attenzione max val 255
```

Se il tempo trascorso è minore del periodo impostato vengono eseguiti dei cicli di delay per azzerare questa differenza.

```
        if(PID_TIME_CYCLE > tempo_ciclo_ms)
        {
          delay_todo_ms = (PID_TIME_CYCLE - tempo_ciclo_ms);
          printf("d=%lu ",delay_todo_ms);
          while (delay_todo_ms > 255)
          {
            delay_ms(255);
            delay_todo_ms -=255;
          }
          delay_ms((int8)(delay_todo_ms));
        }
```

Si risetta a zero sia il timer1 sia la variabile ciclipassati.

```
        disable_interrupts(INT_TIMER1);

        set_timer1(0);

        ciclipassati=0;

        enable_interrupts(INT_TIMER1);
```

Viene letto e convertito il valore presente sulla porta analogica.

```
valoreADC = read_adc()*3;
```

Se il tasto B0 è premuto il valore delle costanti proporzionale, integrativo e derivativo è impostato in base alla lettura avvenuta sulla porta analogica. In particolare la costante proporzionale può variare tra 1 e 0.1, le costanti integrativa e derivativa varrano invece un decimo della costante proporzionale.

```
if(input(PIN_B0)) { setpoint=valoreADC;}

else{
    float a;

     a=(valoreADC/30);

    K_P=a/100;
    //printf("a=%f\r\n",a);

    K_D=K_I=K_P/10;

    printf("P=%f-D=%f-I=%f\r\n",K_P,K_D,K_I);

}
```

Le variabili di errore vengono scalate e alla variabile err è assegnata la differenza tra il setpoint e il valore attuale di giri al minuto del motore.

```
err2=err1;
err1=err;
err=(float)setpoint-fvel;

#define tempo_pid      PID_TIME_CYCLE
```

È possibile impostare la modalità con la quale viene effettuato l'inseguimento del setpoint. Con il tasto A4 è possibile impostare in sequenza 4 stati, ognuno dei quali corrisponde ad una modalità: proporzionale-integrativa-derivativa, proporzionale, integrativa, derivativa. Lo stato nel quale ci si trova è evidenziato dai led B1 e B2.

```
if(!input(PIN_A4)) {

      stato++;
      if(stato>3) stato=0;
}
```

Stato 0: il dutycycle varia secondo un procedimento PID. I due led risultano entrambi spenti. Per il calcolo del dutycycle ci si avvale dell'algoritmo PID in forma discreta.

```
if(stato==0) {

  output_bit(PIN_B2, 0);
  output_bit(PIN_B1, 0);
  duty = duty + K_P * err +  tempo_pid * K_I * (err-err1) + K_D * (err - 2*err1 + err2) / tempo_pid;
  printf("Proporzionale-integrativo-derivativo\r\n");

}
```

Stato 1: il dutycycle varia secondo un procedimento proporzionale. Il led B1 risulta spento mentre risulta acceso il led B2.

```
if(stato==1){

    output_bit(PIN_B1, 0);
    output_bit(PIN_B2, 1);
    duty = duty + K_P * err
    printf("Proporzionale\r\n");

}
```

Stato 2: il dutycycle varia secondo un procedimento integrativo. Il led B2 risulta spento mentre risulta acceso il led B1.

```
if(stato==2){

    output_bit(PIN_B2, 0);
    output_bit(PIN_B1, 1);
    duty = duty + tempo_pid * K_I * (err-err1);
    printf("Integrativo\r\n");

}
```

Stato 3: il dutycycle varia secondo un procedimento derivativo. I due led risultano entrambi accesi.

```
if(stato==3){

    output_bit(PIN_B2, 1);
    output_bit(PIN_B1, 1);
    duty = duty + K_D * (err - 2*err1 + err2) / tempo_pid;
    printf("Derivativo\r\n");

}
```

Il range del dutycycle varia tra i valori 0 e 1023. I valori eccedenti vengono sogliati.

```
if (duty > 1023)
  duty = 1023;

if (duty < 0.0)
  duty = 0;
```

Per evitare di dover riscrivere in continuazione i registri dedicati all'attuazione PWM, se il dutycycle non varia rispetto a quello precedente non vengono effettuati cambiamenti.

```
if (duty != last_duty)
{
  last_duty = duty;
  set_pwm1_duty((int16)duty);
}
```

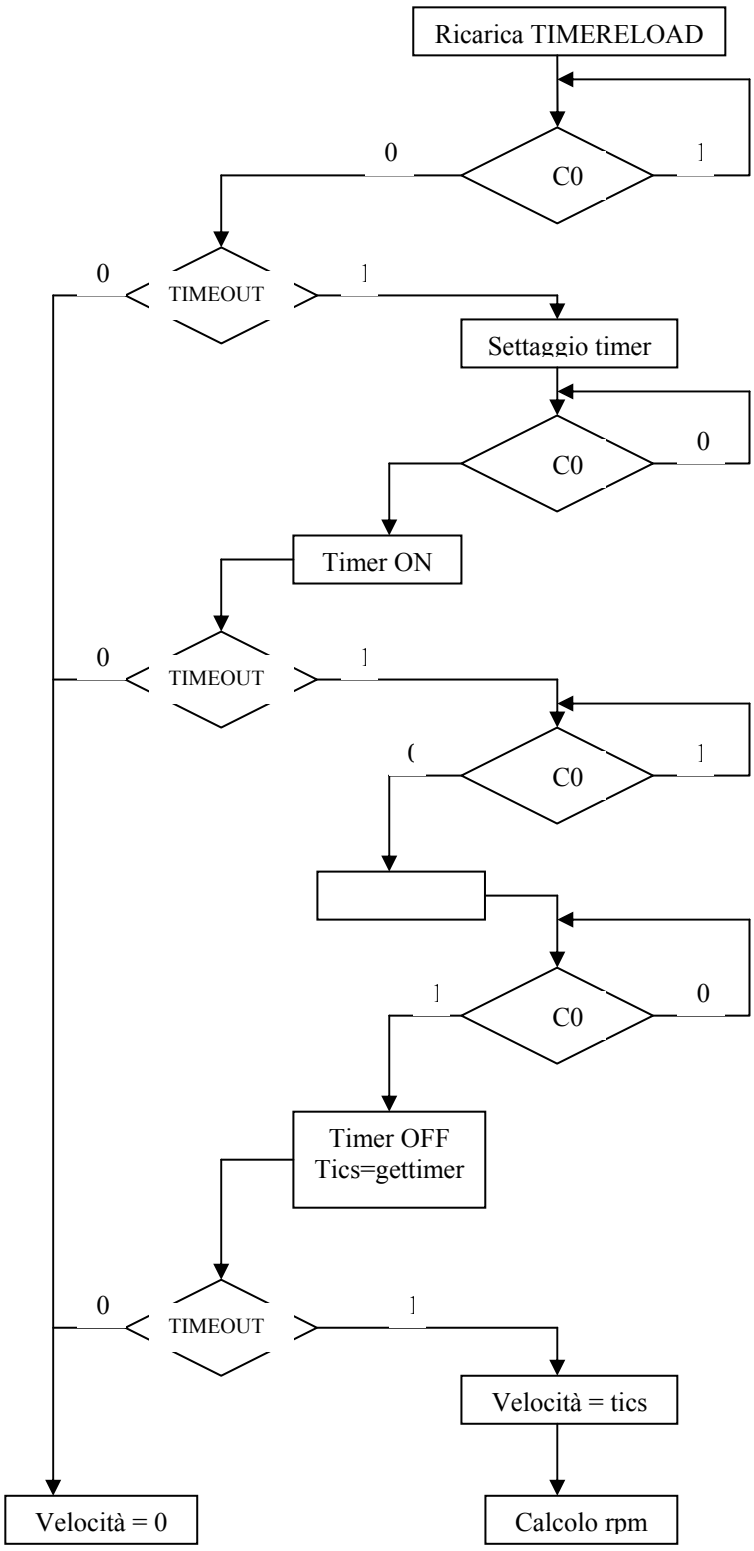
Viene effettuata una printf tramite porta seriale per poter visualizzare su PC i dati di velocità, setpoint, e dutycycle.

```
printf("velocita=%f-setpoint=%lu-dutycycle=%f-tempo=%lu\r\n",fvel,setpoint,duty,tempo);


  }
}
```

Acquisizione Frequenza Motore

Per acquisire la frequenza con la quale gira il motore ci si avvale di un encoder ottico formato da un led che funge da fonte di luce infrarossa e un fotorilevatore integrato. La parte meccanica dell'encoder è costituita da un disco con 16 fenditure fissato all'asse di rotazione. Le fenditure permettono il passaggio di luce infrarossa tra led e fonorivelatore, mentre il metallo lo blocca. Rotando il disco crea ai capi del fotorivelatore un'onda quadra dove il valore logico alto corrisponde al passaggio di una fenditura. Per calcolare la frequenza con la quale ruota il disco si può quindi calcolare il periodo tra due fronti di salita del segnale proveniente dall'encoder che corrisponde al periodo tra due fenditure. Si attende che il segnale sia basso (non si è in corrispondenza di un tacca) e si azzera il Timer0. Il timer è stato precedentemente impostato senza prescaler, basato sull'oscillatore interno e con una risoluzione a 16bit. Appena arriva un fronte di salita sull'ingresso encoder il Timer viene attivato e viene stoppato al successivo fronte di salita. In questo modo il valore nei registri TMR0H e TMR0L corrisponde al conteggio effettuato con la frequenza dell'oscillatore interno tra i due fronti di salita e viene assegnato ad una variabile (tics). La frequenza dell'oscillatore interno è di 4MHz. Per ricavare il tempo che intercorre tra i due eventi si può moltiplicare il numero di occorrenze (tics) per la durata di ciascuna.



$$Periodo = \frac{n°occorrenze}{frequenza\_osc}$$

Moltiplicando il numero delle occorrenze per 16 (in quanto sono presenti 16 fenditure sul disco) si ottiene il tempo che ci impiega una fenditura a ripassare davanti all'encoder. L'inverso di questo sarà quindi la frequenza espressa in giri al secondo del motore. Moltiplicando per 60 questa misura si ottengono i giri al minuto. In sintesi:

$$Fmotore_{RPM} = \frac{frequenza\_osc * 60}{n°occorrenze * 16}$$

Questa frequenza esprime la velocità alla quale gira il motore e sarà confrontata con il setpoint impostato per il calcolo del PWM. Una volta terminato l'algoritmo sarà inviato il nuovo valore PWM che modificherà la velocità del motore influendo sulla modulazione dell'alimentazione dello stesso.

Pid a Periodo Costante.

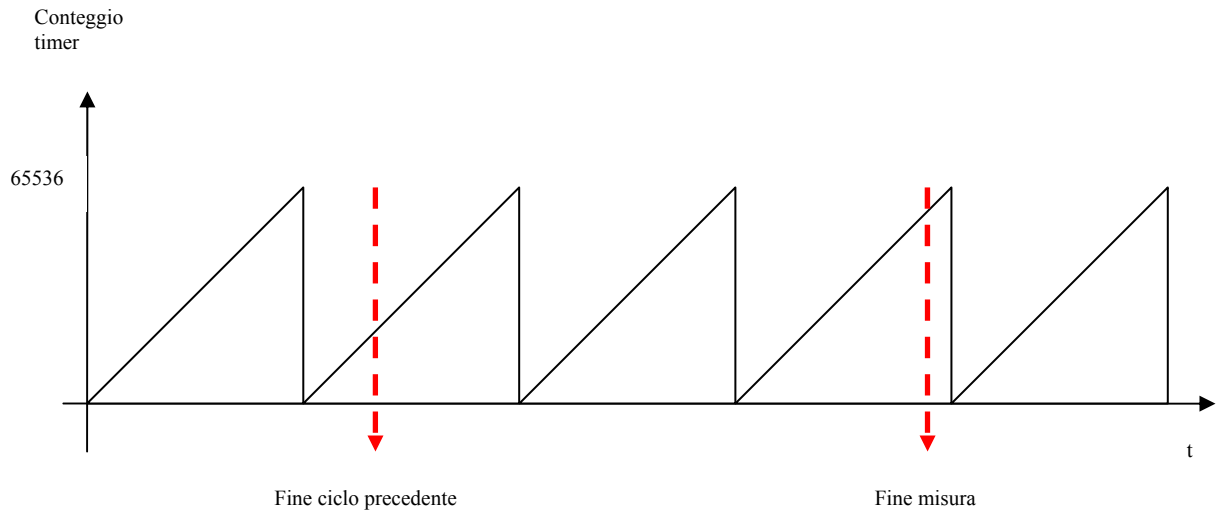
Nella realizzazione del programma è stata data particolare importanza alla temporizzazione. Si è cercato di ottenere che il sistema effettui una misura ogni 100millisecondi. Questo aspetto risulta importante per poter garantire delle prestazioni ottimali del motore. Per misurare il tempo è stato impiegato il Timer1 impostato in modo da ottenere un interrupt per fine conteggio ogni 16ms. Il contatore del timer è a 16 bit e si basa su un oscillatore interno alla frequenza di 4MHz. La frequenza con la quale il registro del timer va in overflow è dato da:

$$f_{overflow} = \frac{4MHz}{2^{16}} \cong 61Hz$$

Il periodo sarà quindi pari a:

$$T_{overflow} = \frac{1}{f_{overflow}} \cong 16ms$$

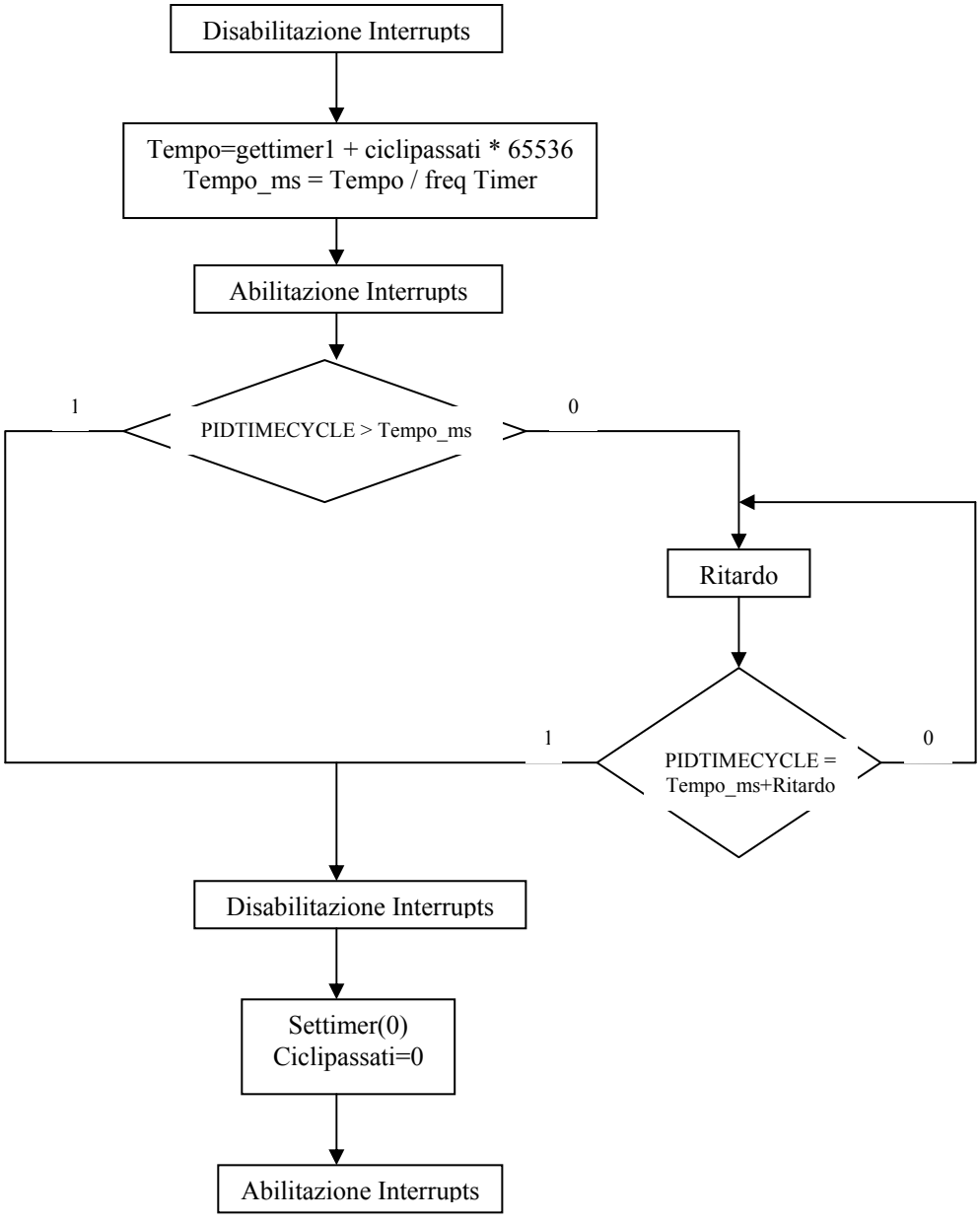
Per calcolare il tempo trascorso tra una misura e la successiva serve tenere in considerazione la possibilità che il registro vada in overflow e si resetti. Perché rimanga traccia di questo nella routine di interrupt del Timer1 viene incrementata una variabile (ciclipassati) che memorizza quante volte il timer ha terminato il ciclo.



In figura è mostrato lo schema di principio per ottenere la misura temporale tra la fine del ciclo precedente e la fine della misura in considerazione. Il tempo è calcolato secondo la formula seguente:

$$tempo = valoreTimer_{PRESENTE} + 65536 * ciclipassati - valoreTimer_{PRECEDENTE}$$

Per esprimere il tempo in millisecondi è sufficiente dividere questo risultato per la frequenza utilizzata dal timer, 4MHz.
Confrontato il tempo calcolato con il tempo di durata del ciclo impostato si predispone un delay in millisecondi peri alla differenza tra i due. A questo punto si resettano sia i cicli passati che il registro di timer e si lancia una nuova operazione. In questo modo ogni ciclo inizia dal valore 0 e ogni misura dista dalla precedente del tempo scelto.
Grazie a questa metodologia è possibile mantenere sotto controllo la possibilità che il motore sia fermo. Infatti in questo caso il processo di acquisizione aspetterebbe i fronti di salita per un tempo infinito. Per aggirare il problema viene settata una variabile di timeout che è decrementata nella routine di interrupt del Timer1. Se la variabile raggiunge il valore zero significa che è passato un tempo massimo e che la velocità del motore è pressoché nulla. Nell'esempio mostrato la variabile di timeout viene caricata all'inizio di ogni ciclo con una costante pari a 3. Questo significa che il timeout sarà individuato dopo 3 interrupt, ossia 48ms. La minima velocità rilevabile è 76giri al minuto.



Attuazione PWM

Tramite l'algoritmo PID in forma discreta è possibile calcolare un valore che rappresenta l'energia che servirebbe fornire al motore per permettergli di raggiungere il setpoint stabilito. È facilmente realizzabile modulare la potenza fornita da un microcontrollore tramite una modulazione di impulso. Questo segnale ad onda quadra ha un dutycycle variabile e il motore viene alimentato solo quando il valore logico di questo segnale è alto. Quindi quando il dutycycle è al 100% (valore logico sempre alto) il motore è alimentato al massimo mentre quando è allo 0% (valore logico sempre basso) il motore non è alimentato e gira per inerzia finché non si ferma per attrito, in quanto non è possibile impostare una velocità negativa. Con il PIC 18F2520 si può settare un'uscita PWM utilizzando il Timer2 in questa modalità. Il range nel quale è possibile far variare il dutycycle è compreso tra 0 e 1023 (8bit). Se il valore calcolato dovesse eccedere questi estremi è opportuno sogliarli a 0 per i minori e a 1023 per i maggiori.