```c
/* ////////////////////////////////////////////////////////////////////////////
** File:      Dino.c
*/
#define  Ver "Dino 18 ver 2.51\r       by Guiott"
/* Autore:    Guido Ottaviani-->g.ottaviani@mediaprogetti.it<--
** Descrizione:Gestione Robot Dino, comportamento tipo explorer
**             questo e' un laboratorio per la prova di utilizzo di diversi sensori
**             e il movimento in un abiente sconosciuto
**
**
** LA DESCRIZIONE PROSEGUE SU "settings.h"
////////////////////////////////////////////////////////////////////////////*/


// include standard
#include <p18f4525.h>
#include <stdlib.h>
// #include <math.h>
#include <pwm.h>
#include <i2c.h>

#include "settings.h" // contiene: definizioni, prototipi funzioni, settings, descrizioni
#include "terminal.h" // contiene routine gestione tastiera e display
#include "myi2c.h"    // contiene routine gestione i2c a basso ed alto livello
#include "sensor.h"   // contiene routine gestione sensori peer explorer
#include "movement.h" // contiene routine gestione movimento

/* MAIN ==========================================================================*/
void main (void)
{
INTCONbits.GIEH=0;   // disabilitati tutti gli interrupt high
INTCONbits.GIEL=0;   // disabilitati tutti gli interrupt low


Settings(); // imposta porte e registri (settings.h)


// ---- Inizializzazione flag e variabili
MotEnable = 0;       // motori disabilitati
PidTick=0;

SetDCPWM2(0x0100);   // motori fermi
SetDCPWM1(0x0100);

Buzzer=0;            // beeper spento
FlagTargetLight = 1; // i sensori di luce sono abilitati
LedRossoOFF;         // led spenti
LedGialloOFF;
LedVerdeOFF;

```

```
51   // ---- Lettura costanti da EEPROM
52   Light1=ReadEeprom(0);// prima soglia di intensita' luminosa, il bot si gira verso la luce (37-0x25)
53   Light2=ReadEeprom(1);// seconda soglia, il bot ha raggiunto il target e si ferma a Xcm (48-0x30)
54   GasOn=ReadEeprom(2); // soglia di "gas rivelato" (130 - 0x82)
55   kp=ReadEeprom(3);    // costante errore proporzionale (fattore P) moltiplicato 16 (40 - 0x28)
56   ki=ReadEeprom(4);    // costante errore integrale (fattore I) moltiplicato 16 (24 - 0x18)
57   kd=ReadEeprom(5);    // costante errore derivativo (fattore D) moltiplicato 16 (24 - 0x18)
58   En=ReadEeprom(6);    // abilitazione motori, = 0 per debug o misure
59
60   // ---- init I2C
61   I2cBusCollFlag = 0;
62   I2cEventFlag = 0;
63   I2cBusyFlag = 0;
64   for (i=0; i<I2cMaxDev; i++)
65   {
66      I2c[i].Flag.Tx = 0;
67      I2c[i].Flag.Rx = 0;
68      for (j=0; j<=1; j++)
69      {
70         I2c[i].RxBuff[j] = 0;
71      }
72
73      for (j=0; j<=3; j++)
74      {
75         I2c[i].TxBuff[j] = 0;
76      }
77   }
78
79
80   // ---- init LCD
81   InitFlag=1;          // avvia l'inizializzazione dell'LCD
82   LcdInitStatus=0;     // azzera contatore stati init LCD
83   DisplayStatus=1;     // abilita la gestione del display all'avvio
84   UserInterfaceFlag=1; // abilita la gestione delle routine manuali all'avvio
85   TimerLcd=LcdCycle;   // azzera timer per routine init LCD
86                        // ritardo di 150mS per far stabilizzare tutti i device all'avvio
87   LcdPutCharFlag=0;    // azzera trasmissione byte su LCD
88   LcdStringPtr=0xFF;   // disabilita LcdStringPut
89
90   LcdVar[1][4]=0;              // menu 1 non si modifica
91   LcdVar[2][4]=0;              // menu 2 non si modifica
92   LcdVar[3][4]=0;              // menu 3 non si modifica
93   LcdVar[4][4]=1;              // menu 4 si modifica
94   LcdVar[5][4]=1;              // menu 5 si modifica
95
96   // non usate
97   LcdVar[4][2]=0;
98   LcdVar[2][1]=0;
99   LcdVar[2][2]=0;
100  LcdVar[2][3]=0;
```

```
101
102  // ---- init Keypad
103  FlagKbdIntr=0;       // inizializza flag Keypad
104  KeypadInt=0;         // inizializza flag Interrupt
105  FlagKbdStartRead=0;  // inizializza flag Lettura keypad
106  CursorStatus=0;      // posizione iniziale del cursore sui menu con modifica
107  FlagMenuMod=0;       // a default i menu non sono modificabili
108
109  // ---- init Bussola
110  TimerCmp=CmpCycle;   // azzera timer lettura bussola
111  FlagCmp=0;           // flag lettura bussola
112  FlagCmpCal=0;        // flag avvio procedura calibrazione bussola
113  FlagCmpReg15=0;      // flag calibrazione bussola
114
115  // ==== enable interrupt in questo momento parte il timer di sistema e quindi l'RTOS
116  INTCONbits.GIEH=1;   // abilitati tutti gli interrupt high
117  INTCONbits.GIEL=1;   // abilitati tutti gli interrupt low
118
119
120  while (1)  // main loop
121  {
122
123  //TimeKeeping-------------------------------------
124  if (FlagTime)        // e' passato un altro millisecondo
125  {
126     TimeKeeping();
127  }
128
129
130  //UserInterface-----------------------------------
131  if (UserInterfaceFlag)
132  /* flag di abilitazione di tutte le routine gestite manualmente: Keypad, LCD, EEPROM.
133     Dal momento che sono eseguite molto raramente sono tutte ragruppate sotto un unico
134     semaforo. Questo permette di risparmiare diversi controlli nel funzionamento normale.
135  */
136  {
137     UserInterface();     // (terminal.h)
138  }
139
140
141  //MotSpeed----------------------------------------
142     if (PidTick)
143     {
144        MotSpeed();       // (movement.h)
145        /*
146           Regola la velocita' dei motori
147           questa routine e' eseguita solo su abilitazione della ISR, quest'ultima calcola
148           la velocita' ogni x mSec e solo dopo averla calcolata ne permette la correzione.
149           Il calcolo PID e' quindi eseguito piu' o meno con la stessa periodicita'
150        */
```

```
151     }
152
153
154  //Space2Run-------------------------------------
155     if (Space2RunFlag)
156     {
157        Space2Run();      // (movement.h)
158        /*
159           Controlla lo spazio da percorrere
160           qualche routine ha abilitato il controllo dello spazio percorso,
161           ha impostato la velocita' (DesSpeedX), lo spazio da percorrere (Space2RunX)
162           e la posizione di partenza (Space2RunXstart).
163           Questa routine sara' eseguita continuamente fino al raggiungimento della
164           posizione desiderata, dopodiche' fermera' i motori e disabilitera' il flag
165        */
166     }
167
168
169  //Path------------------------------------------
170     if (Space2RunFlag == 0  && PathSeq[PathSeqPointer] != 0 && FlagStop == 0)
171     {
172        Path();          // (movement.h)
173        /*
174           Esegue, in sequenza, le manovre impostate
175           se Space2RunFlag = 1, e' ancora in esecuzione una singola routine di movimento
176           se PathSeq[PathSeqPointer] = 0 -> fine sequenza routines di movimento (EOL)
177           se FlagStop = 1 -> la routine Stop ha fermato i motori
178
179            La routine che deve usare Path
180              -Imposta DesSpeedX = 0; // ferma motori
181              -imposta PathSeq con passi da fare in ordine (fine seq = 0)
182              -imposta PathSeqPointer = 0; // inizializza sequenza passi
183              -imposta Space2RunFlag = 0; // reset di qualsiasi routine di movimento
184
185           Se deve avanzare a velocita' costante non c'e' bisogno di chiamare Path(),
186           si imposta semplicemente la velocita'
187        */
188     }
189
190
191  //Bumpers---------------------------------------
192     Bumpers();        // (sensor.h)
193     /*
194        Controllo collisioni
195        questi sensori hanno priorita' massima su tutti gli altri
196        e resettano qualsiasi manovra eventualmente in esecuzione
197     */
198
199
200  //Stop------------------------------------------
```

```
201     if (FlagStop)
202     {
203        Stop();         // (movement.h)
204        /*
205           se qualche routine ha abilitato lo stop i motori si fermano per X mSec
206        */
207     }
208
209
210  //ReadAD-----------------------------------------
211     if ((TimerAD) >= 7)
212     {
213        ReadAD();       // (sensor.h)
214        /*
215           legge le porte analogiche e mette i valori letti nelle rispettive variabili.
216           TimerAD viene incrementato ogni 1ms dalla ISR
217           La routine di conversione viene  chiamata solo dopo 7ms, in attesa Tacq,
218           anche se per il Tempo di acquisizione bastano 10-20us con il PIC 16F877 o il
219           PIC 18F452 e 2-3us con il PIC 18F4525, non servono piu' di 20 letture al
220           secondo per ognuno dei sensori.
221           Tutte le porte AD sono utilizzate:
222           EyeRl, EyeLl, EyeRf, EyeLf, FrR, FrL, FrC, Gas
223           Ogni 7ms legge una porta, quindi ritorna a leggere la stessa porta dopo:
224           7x7=49ms ,  1000/49=20 letture al secondo
225           Se aumentano le porte A/D da leggere, diminuire il tempo di attesa.
226           Il  Timer viene  resettato alla fine del ciclo di conversione dalla routine  stessa
227        */
228     }
229
230
231  //Eye--------------------------------------------
232     if (Space2RunFlag == 0  && PathSeq[PathSeqPointer] == 0 && FlagStop == 0 && FlagEye)
233     {
234        Eye();          // (sensor.h)
235        /*
236           Controllo sensori di prossimita' ad IR
237           Questa routine ha priorita' bassa rispetto alle altre routine di movimento
238           viene quindi eseguita solo se non ci sono manovre in atto, anche se innescate
239           dalla routine stessa
240           se Space2RunFlag = 1, e' ancora in esecuzione una singola routine di movimento
241           se PathSeq[PathSeqPointer] = 0 -> fine sequenza routines di movimento (EOL)
242           se FlagStop = 1 -> la routine Stop ha fermato i motori
243           se FlagEye = 0 -> non c'e' ancora stata una nuova lettura dei sensori, e' inutile
244              eseguire questa routine, si risparmiano molti cicli macchina
245        */
246     }
247
248
249  //Light------------------------------------------
250     if (FlagLight)
```

```
251     {
252        if (!FlagTargetLight)
253        {
254           if (TimerSensor <= 0)
255           {
256              FlagTargetLight = 1; // riabilita i sensori dopo "obiettivo raggiunto"
257           }
258        }
259        else
260        {
261           Light();    // (sensor.h)
262           /*
263           Controllo sensori di luce
264           se FlagLight = 0 -> non c'e' ancora stata una nuova lettura delle fotoresistenze,
265           e' inutile eseguire questa routine, si risparmiano molti cicli macchina.
266           Se la routine e' stata disabilitata perche' raggiunto obiettivo di luce:
267           (FlagTargetLight=0) controlla se e' scaduto il timer di disabilitazione.
268           Se e' scaduto, riabilita la routine e non controlla piu' il timer
269           */
270        }
271     }
272
273
274  //Gas-----------------------------------------
275     if (FlagGas)
276     {
277        if (!FlagTargetGas)
278        {
279           if (TimerSensor <= 0)
280           {
281              FlagTargetGas = 1;   // riabilita i sensori dopo "obiettivo raggiunto"
282           }
283        }
284        else
285        {
286           Gas();      // (sensor.h)
287           /*
288           Controllo sensori di gas
289           se FlagGas = 0 -> non c'e' ancora stata una nuova lettura dei TGS822,
290           e' inutile eseguire questa routine, si risparmiano molti cicli macchina.
291           Se la routine e' stata disabilitata perche' raggiunto sorgente di gas:
292           (FlagTargetGas=0) controlla se e' scaduto il timer di disabilitazione.
293           Se e' scaduto, riabilita la routine e non controlla piu' il timer
294           */
295        }
296     }
297
298
299  //Sound---------------------------------------
300     if (FlagSound && FlagTargetLight && !Buzzer)
```

```
301     {/* disabilita sensore suono mentre e' fermo per rivelazione luce altrimenti
302         potrebbe capitare in un loop infinito tra delay di rivelazione luce e suono
303         in caso di echi vicino a sorgenti luminose.
304         Disabilitato anche mentre suona il buzzer per evitare che riveli il suo tono
305         come segnale a 4KHz
306      */
307        if (!FlagTargetSound)
308        {
309           if (TimerSensor <= 0)
310           {
311              FlagTargetSound = 1; // riabilita i sensori dopo "obiettivo raggiunto"
312           }
313        }
314        else
315        {
316           Sound();    // (sensor.h)
317           /*
318           Controllo sensore di suono
319           Viene abilitata dalla routine ReadAD con la stessa temporizzazione della
320           routine Gas.
321           Se la routine e' stata disabilitata perche' raggiunto obiettivo di suono:
322           (FlagTargetSound=0) controlla se e' scaduto il timer di disabilitazione.
323           Se e' scaduto, riabilita la routine e non controlla piu' il timer
324           */
325        }
326     }
327
328
329  //Beep-------------------------------------------
330     if ((BeepCount>0) && (TimerBeep >= BeepTime))
331     {
332           Beep();
333           /*
334              Suona il buzzer
335           */
336     }
337
338
339  //DeadCorner-------------------------------------
340     if (TimerDeadCorner >= DeadCornerTime)
341     {
342        /*
343        controlla se il bot e' entrato in un angolo morto, ad ogni manovra si incrementa
344        un contatore. Se il contatore supera un valore prefissato in una precisa
345        finestra di tempo, il bot fa retromarcia.
346        In ogni caso si resettano contatore e timer dopo il timeout
347        */
348           if (DeadCorner >= MaxDeadCorner)
349           {
350              // inverte la marcia
```

```
351             PathSeq [0] = 2;   // indietro n cm
352             PathSeq [1] = 51;  // gira di 135° o -135°  randomicamente
353             PathSeq [2] = 255; // cammina avanti a velocita' costante
354             PathSeq [3] = 0;   // fine sequenza
355             PathSeqPointer = 0;// inizializza sequenza
356          }
357          DeadCorner = 0; // reset contatore e timer
358          TimerDeadCorner = 0;
359    }
360
361
362 //I2C-----------------------------------------
363    if (I2cEventFlag)
364    /* la  porta SSP ha comunicato, tramite ISR, che e' terminata la precedente azione I2c
365       LA DESCRIZIONE DELLE PROCEDURE I2c PROSEGUE IN DETTAGLIO SU "settings.h"
366    */
367    {
368      I2cLowService();      // si esegue l'azione successiva (myi2c.h)
369      I2cEventFlag=0;       // pronti per il prossimo interrupt
370       // exit
371    }
372    else
373    {
374       if (I2cBusyFlag)     // I2cHighService ha avviato una sequenza di
375                            // comunicazione I2c gestita tramite  ISR
376                            // solo alla fine la I2cLowService liberera' il bus
377       {
378       // exit
379       }
380
381       else
382       {
383          if ((I2c[I2cDevPtr].Flag.Rx > 0) || (I2c[I2cDevPtr].Flag.Tx > 0))
384          // si controlla un altro record del buffer per vedere se le routine a
385          // livello piu' alto hanno inserito byte da scambiare nel proprio buffer
386
387          {
388             I2cHighService();    // Inizializza una nuova sequenza di comunicazione
389                                  // e impegna il bus  (myi2c.h)
390             // exit
391          }
392
393          else
394          {
395             if(I2cDevPtr < (I2cMaxDev-1)) // ci sono ancora record da controllare nel buffer
396             {
397                I2cDevPtr ++;        // al prossimo giro controllera' il record successivo
398                // exit
399             }
400
```

```
401                 else                        // ha controllato tutti i record del buffer
402                 {
403                     I2cDevPtr = 0;          // al prossimo giro ricomincera' dal primo device
404                 }
405             }
406         }
407     }
408
409 //CmpRead--------------------------------------
410     if ((TimerCmp) <= 0)
411     {// e' tempo di leggere la bussola
412        if (!I2c[CmpPtr].Flag.Tx && !I2c[CmpPtr].Flag.Rx)
413        {// se i buffer TX e RX I2C sono liberi si puo' avviare una nuova lettura
414          CmpRead();              // (sensor.h)
415        }
416        /*
417           avvia la procedura di lettura o calibrazione della bussola tamite I2C
418        */
419     }
420
421 }  // end while
422 } // main
423 /*=============================================================================*/
424
425
426
427 /* Funzioni ***********************************************************************/
428
429
430 /*WriteEeprom ********************************************************************
431 scrive nell'Eeprom interna "Data" all'indirizzo "Adr"
432 */
433
434 void WriteEeprom(unsigned char Data, unsigned char Adr)
435 {
436     EEADR=Adr;
437     EEDATA=Data;
438     EECON1bits.WREN=1;     // abilita scrittura
439     EECON2=0x55;
440     EECON2=0xAA;
441     EECON1bits.WR=1;       // avvia scrittura
442 } // WriteEeprom
443 /*********************************************************************************/
444
445
446 /*ReadEeprom *********************************************************************
447 ritorna il dato scritto nell'Eeprom interna all'indirizzo Adr
448 */
449
450 unsigned char ReadEeprom (unsigned char Adr)
```

```
451  {
452     EEADR=Adr;
453     EECON1bits.EEPGD=0; // Point to DATA memory
454     EECON1bits.CFGS=0;  // Access program FLASH or Data EEPROM memory
455     EECON1bits.RD=1;    // EEPROM Read
456     return EEDATA;
457
458  } // ReadEeprom
459  /*****************************************************************************/
460
461
462  /*TimeKeeping ****************************************************************
463  gestione temporizzazioni e calcolo velocita'
464  */
465
466  void TimeKeeping (void)
467   {// e' passato un'altro milliSecondo
468      TimerLcd --;      // timer fino a  30  Sec per visualizzazione display
469      TimerBumpers ++; // timer fino a  30  Sec per routine bumpers
470      TimerStop --;     // timer fino a  30  Sec per routine Stop Motori
471      TimerBeep ++;     // timer fino a  30  Sec per routine Beep
472      TimerSensor --;   // timer fino a  30  Sec per disabilitazione sensori
473      TimerAD ++;       // timer fino a 255 mSec per routine conversione AD
474      TimerDeadCorner ++;//timer fino a 30  Sec per routine rivelazione angoli morti
475      TimerKeypad ++;   // timer fino a 255 mSec per debounce tasti
476      TimerCmp --;      // timer fino a  30  Sec per routine lettura bussola
477      RandomBit = RandomBit ^ 1; // ogni mSec viene invertito, usato come variabile random
478
479     if (DeltaT >= 64)   //ogni 64 mSec campiona la velocita'
480     {
481        // velocita' = spazio/Tempo. Dividendo millesimi di mm con millisecondi il risultato e' in mm/Sec
482        SpeedR = DivIntS64((EncoderRcount - EncoderRcountPrev) * stepR); // divide per 64 (DeltaT)
483        EncoderRcountPrev = EncoderRcount;   // reset contatore
484
485        SpeedL = DivIntS64((EncoderLcount - EncoderLcountPrev) * stepL); // ripete per l'altra ruota
486        EncoderLcountPrev = EncoderLcount;
487
488        DeltaT=0;
489        PidTick = 1;   // eseguito il calcolo della velocita' può essere eseguito il calcolo del PID
490     }
491     else
492     {
493        DeltaT++;
494     }
495
496     FlagTime=0;    // azzera flag in attesa del prossimo millisecondo
497
498  } // TimeKeeping
499  /*****************************************************************************/
500
```

```
501
502 /*SequenzaStart ***************************************************************************
503 Esce dallo stato "init" e inizializza le sequenze di movimento
504 */
505
506 void SequenzaStart (void)
507 {
508    InitFlag = 0;        // esce dallo stato init
509
510    // ---- init movimento
511    FlagBumpers = 1;  // disattiva bumpers
512    TimerBumpers = 0; // per 1Sec
513    Space2RunFlag = 0;// reset di qualsiasi routine di movimento
514    // fermo per 1 sec poi parte a velocita' normale
515         PathSeq [0] = 21;  // fermo 1 sec
516         PathSeq [1] = 255; // cammina avanti a velocita' costante
517         PathSeq [2] = 0;   // fine sequenza
518         PathSeqPointer = 0;// inizializza sequenza
519    BeepCount = 0;           // suona 1 beep, per debug=0, altrimenti = 2 *=*=*=*=*=*
520    TimerLcd=LcdCycle;       // tempo tra le visualizzazioni del display
521    LcdInitStatus=0;         // azzera contatore stati init LCD
522    LcdPutCharFlag=0;        // azzera trasmissione byte su LCD
523    LcdStringPtr=0xFF;       // disabilita LcdStringPut
524    LcdStatus=0;             // parte dal primo passo del ciclo
525    DisplayStatus=90;        // disabilita display
526
527    LedRossoOFF;             // led spenti
528    LedGialloOFF;
529    LedVerdeOFF;
530    LcdPutChar (0x01, 0, 4);// clear display
531
532 } // SequenzaStart
533 /*****************************************************************************************/
534
535
536 /*Beep ************************************************************************************
537 esegue un numero di beep uguali al contenuto della variabile BeepCount/2
538 la routine che deve far suonare il beep, carica nella variabile BeepCount il numero
539 di beep da eseguire moltiplicato per 2.
540 Ogni BeepTime ms viene chiamata Beep che inverte Buzzer e decrementa BeepCount.
541 Il buzzer sara' quindi per BeepTime ms ON e per BeepTime ms OFF (duty cycle 50%)
542 */
543
544 void Beep (void)
545 {
546    TimerBeep = 0; // azzera il contatore per il prossimo ciclo
547    BeepCount--;   // decrementa il contatore di semicicli eseguiti
548    Buzzer = Buzzer ^ 1; // se e' accceso lo spegne e viceversa
549
550 } // Beep
```

```
551 /*****************************************************************************/
552
553
554
555 /*===========================================================================*/
556
557 // Low priority interrupt vector
558
559 #pragma code LowVector = 0x18
560 void InterruptVectorLow (void)
561 {
562    asm
563     goto InterruptHandlerLow //jump to interrupt routine
564   _endasm
565 }
566
567 //---------------------------------------------------------------------------
568 // Low priority interrupt routine
569
570 #pragma code
571 #pragma interruptlow InterruptHandlerLow
572
573 /*===========================================================================*
574
575 /*IntServiceRoutine*************************************************************/
576 void InterruptHandlerLow (void)
577
578 {
579     if (!EncoderRint && !EncoderLint && !PIR1bits.SSPIF && !PIR2bits.BCLIF)
580     //se non e' interrupt previsto c'e' un errore
581     {
582          while (1)      // blocca l'esecuzione del programma
583          {
584          MotEnable = 0; // ferma i motori
585          Buzzer=1;      // suona il beep
586          LedRossoOFF;
587          LedGialloON;
588          LedVerdeON;
589          }
590     }
591
592     if (EncoderLint) // impulso dall'encoder SX?
593     {
594     /* encoder in quadratura, se al cambiamento di stato i due segnali non sono in fase
595        la direzione e' = FWD, altrimenti e' REW
596     */
597       if (EncoderLpulse != EncoderLdir)
598       {
599           EncoderLcount ++;   // allora incrementa contatore
600       }
```

```
601         else
602         {
603             EncoderLcount --;   // altrimenti decrementa contatore
604         }
605         EncoderLint = 0;
606       }
607 
608 
609       if (EncoderRint) // ripete per l'altro encoder
610       {
611         if (EncoderRpulse != EncoderRdir)
612         {
613             EncoderRcount ++;   // allora incrementa contatore
614         }
615         else
616         {
617             EncoderRcount --;   // altrimenti decrementa contatore
618         }
619         EncoderRint = 0;   // reset interrupt flag
620       }
621 
622 
623       if (PIR1bits.SSPIF)             // un evento I2c e' stato completato
624       {
625            PIR1bits.SSPIF = 0;        // reset dell'interrupt I2c
626            I2cEventFlag = 1;          // sara' eseguita la I2cLowService
627       }
628 
629 
630       if (PIR2bits.BCLIF)             // c'e' stata una collisione sul bus I2c
631       {
632            PIR2bits.BCLIF = 0;        // reset dell'interrupt I2c
633            I2cEventFlag = 1;          // sara' eseguita la I2cLowService
634            I2cBusCollFlag = 1;        // avverte della collisione
635       }
636 
637   }   // Low Priority IntServiceRoutine
638   /*****************************************************************************/
639 
640 
641   /*===========================================================================*/
642   // High priority interrupt vector
643 
644   #pragma code HighVector = 0x08
645   void
646   InterruptVectorHigh (void)
647   {
648      asm
649       goto InterruptHandlerHigh //jump to interrupt routine
650     _endasm
```

```
651  }
652
653  //---------------------------------------------------------------------------
654  // High priority interrupt routine
655
656  #pragma code
657  #pragma interrupt InterruptHandlerHigh
658
659  /*===========================================================================*/
660
661
662  /*IntServiceRoutine*********************************************************/
663  void InterruptHandlerHigh (void)
664
665  {
666   if (INTCONbits.TMR0IF)   // l'interrupt e' stato generato dall'overflow del timer 0 ?
667   {
668      TMR0L -= 156;// ricarica il timer
669      /*interrupt ogni 1mSec
670       periodo = CLKOUT * prescaler * TMR0
671       1mSec ~= 1/(40.000.000/4) * 64 * 156
672      */
673
674      FlagTime=1;          // abilita aggiornamento temporizzazioni
675
676      INTCONbits.TMR0IF = 0;// reset flag di interrupt
677   }
678
679
680   if  (KeypadInt)         // e' stato premuto un tasto
681    {
682        FlagKbdIntr= 1;    // abilita la routine Keypad
683        UserInterfaceFlag=1;// abilita la routine user interface
684        KeypadInt = 0;     // reset dell'interrupt
685    }
686
687  }   // High Priority IntServiceRoutine
688  /*****************************************************************************/
689
690
691  /* Fine funzioni ************************************************************/
692
```