Progetto DIM-BOT – parte 7

REALIZZAZIONE DEL SOFTWARE

La logica di funzionamento di ogni sistema “intelligente” risiede in un programma, che nel nostro caso verrà eseguito dal microcontrollore presente sulla scheda.

Per gestire il manipolatore abbiamo deciso di progettare una scheda con 3 microcontrollori, pertanto i pogrammi saranno 3, due per il controllo del movimento dei singoli motori ed uno per la gestione della logica del manipolatore.

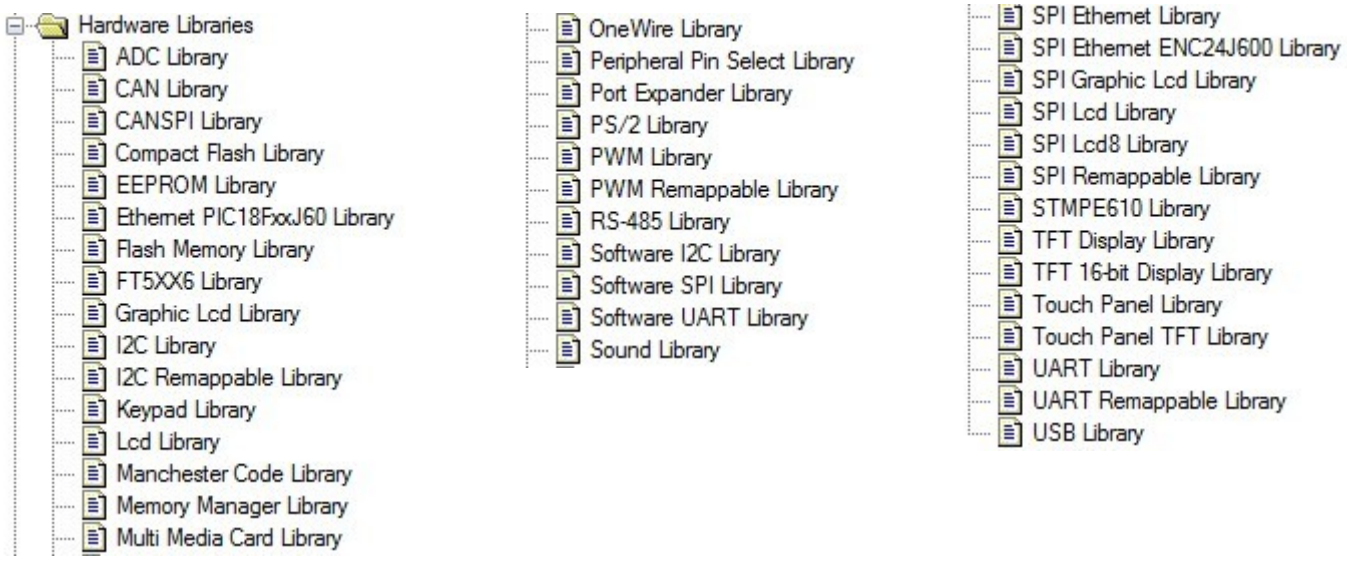
Il microcontrollore principale è un PIC18F4620, un microprocessore ad 8bit, scelto soprattutto per la sua elevata quantità di RAM circa 4 kbytes, anche se può sembrare poco, per un microcontrollore è un valore più che soddisfacente. Esso funziona con il clock interno ad 8MHz, una velocità più che sufficiente per svolgere le funzioni richieste. Il microcontrollore è dotato di diversi dispositivi hardware interni, tra cui una seriale RS232, ingressi analogici e a svariati GPIO, cioe i piedini di ingresso ed uscita digitali.

I due microcontrollori che gestiscono i motori, sono dei DSPIC30F4011, due microcontrollori a 16 bit, scelti per la loro velocità 30 MIPS e per la possibilità di gestire, qualora necessario, un encoder incrementale. I due DSPIC funzionano con un clock esterno a 20Mhz, e si occupano solamente di gestire il movimento dei due motori passo-passo.

Tutti 3 i micro sono stati programmati utilizzando il linguaggio di programmazione C, tramite la suite di programmazione fornita dalla ditta Mikroelektronika. Per chi fosse interessato sul sito www.mikroe.com, sono disponibili tutti i compilatori e tutti i dispositivi (programmatori, schede di sviluppo ecc…).

I compilatori utilizzati sono il mikroCpro per PIC e mikroCpro per DSPIC, entrambi forniscono delle librerie molto utili, tra cui quella per la comunicazione seriale, o quella per la gestione del display LCD.

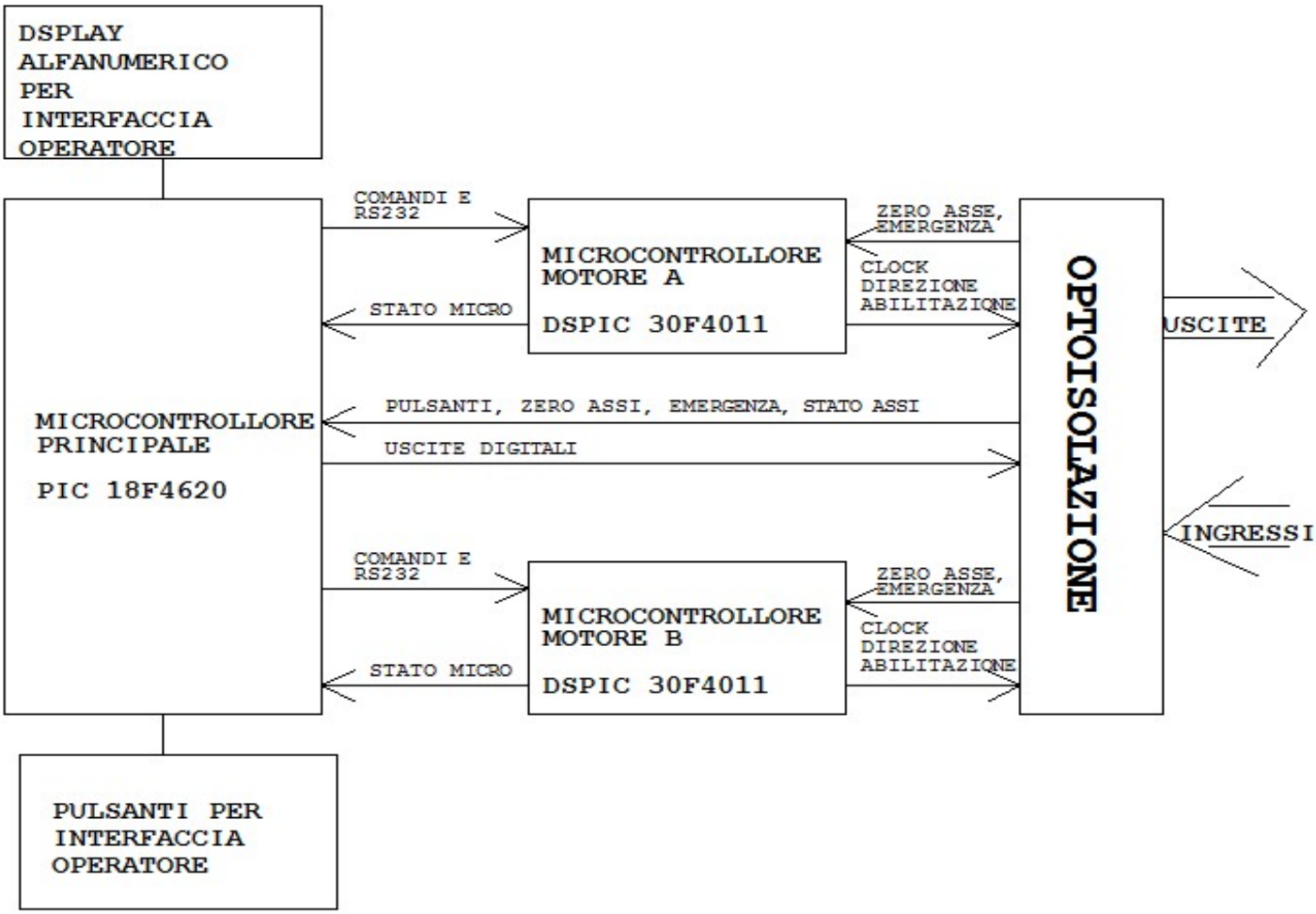
Di seguito l’elenco delle librerie messe a disposizione da questi compilatori.



COMUNICAZIONE

Come abbiamo già detto nella parte relativa alla progettazione della scheda, i 3 microcontrollori dovranno dialogare tra di loro, pertanto la parte più importante è sicuramente quella dello sviluppo di un protocollo di comunicazione tra i 3 dispositivi.

Riprendiamo lo schema a blocchi visto precedentemente:



Si è scelto di sviluppare una comunicazione basata sul classico ed ormai più che solido protocollo di comunicazione RS232, perché semplice, stabile e realizzabile senza componenti aggiuntivi. Solo che questo tipo di protocollo è adatto per fa comunicare solo due dispositivi, nel nostro caso sarebbe stato più adatto un bus RS485 o un altro metodo che possa far comunicare più dispositivi sullo stesso bus di collegamento, ma con qualche piccolo adattamento possiamo far comunicare i 3 dispositivi anche con l'RS232.

Il microcontrollore principale, avvierà sempre la comunicazione con gli altri due, scegliendo prima con chi comunicare mediante un bit digitale, il Chip Select. Quando questo bit è attivo si instaurerà la comunicazione tra il micro principale ed il DSPIC scelto, il micro principale potrà in questo modo inviare i seguenti comandi descritti in tabella.

COMANDO DA PIC A DSPIC	SIGNIFICATO	RISPOSTA DA DSPIC A PIC
VX#	INVIO VELOCITA' DI LAVORO *X=VALORE IN ESADECIMALE DA 1 A 90*	ECO DEI CARATTERI RICEVUTI
DXX#	INVIO DELAY IN uSEC PER POSIZIONAMENTO IN START STOP,. *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
PXX#	INVIO POSIZIONE ASSOLUTA IN IMPULSI *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
A#	LEGGI IL NUMERO ASSE	ECO DEI CARATTERI RICEVUTI BYTE CONTENENTE IL NUMERO DELL'ASSE
C#	LEGGI STATO ASSE	ECO DEI CARATTERI RICEVUTI BYTE CONTENENTE 8 BIT CHE IDENTIFICANO LO STATO DELL'ASSE, AZZERATO IN ERRORE, ECC..
S#	LEGGI CODICE ERRORE	ECO DEI CARATTERI RICEVUTI BYTE CONTENENTE UN NUMERO CHE RAPPRESENTA IL TIPO DI ERRORE DEL'ASSE
s#	RESETTA ERRORE	ECO DEI CARATTERI RICEVUTI
OXX#	INVIA OFFSET IN IMPULSI *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
o#	LEGGI OFFSET IN IMPULSI	ECO DEI CARATTERI RICEVUTI DUE BYTE CONTENENTI LA PARTE BASSA ED ALTA DEL VALORE RICHIESTO
v#	LEGGI LA VELOCITA' IMPOSTATA	ECO DEI CARATTERI RICEVUTI UN BYTE CONTENENTE IL VALORE RICHIESTO
d#	LEGGI I DELAY IMPOSTATO	ECO DEI CARATTERI RICEVUTI DUE BYTE CONTENENTI LA PARTE BASSA ED ALTA DEL VALORE RICHIESTO
p#	LEGGI LA POSIZIONE IMPOSTATA	ECO DEI CARATTERI RICEVUTI DUE BYTE CONTENENTI LA PARTE BASSA ED ALTA DEL VALORE RICHIESTO
L#	LEGGI POSIZIONE ASSOLUTA IN IMPULSI	ECO DEI CARATTERI RICEVUTI DUE BYTE CONTENENTI LA PARTE BASSA ED ALTA DEL VALORE RICHIESTO
M1XX#	INVIO POSIZIONAMENTO ASSOLUTO IN IMPULSI STEP1 *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
M2XX#	INVIO POSIZIONAMENTO ASSOLUTO IN IMPULSI STEP2 *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
M3XX#	INVIO POSIZIONAMENTO ASSOLUTO IN IMPULSI STEP3 *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
M4XX#	INVIO POSIZIONAMENTO ASSOLUTO IN IMPULSI STEP4 *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI
M5XX#	INVIO POSIZIONAMENTO ASSOLUTO IN IMPULSI STEP5 *XX= DUE BYTE PARTE LOW E PARTE HIGH DEL VALORE IN ESADECIMALE*	ECO DEI CARATTERI RICEVUTI

Sempre per semplificare al massimo il software, non è stato previsto alcun controllo di errore, CRC o simili, ma si utilizza il semplice controllo dell'eco del carattere ricevuto.

Al termine di ogni comando inoltre viene inviato il carattere #.

Per realizzare la comunicazione sono state sviluppate le seguenti routine:

Microcontrollore principale.

Questa routine cancella il vettore utilizzato per inviare il comando sulla seriale

```
//*******************************************************
//CANCELLA GLI ARRAY PER LA COMUNICAZIONE SERIALE
//*******************************************************
void vuota_array_seriale()
```

Questa routine invia il comando scelto sulla seriale. Viene restituito il valore 0 se non ci sono errori.

```
//*******************************************************
//INVIO COMANDI SULLA SERIALE
//*******************************************************
char invia_dati()
```

Questa routine legge il valore intero restituito dal DSPIC, composto da due byte, in base al valore della variabile “tipo”, il valore viene memorizzato nell’apposita variabile.

```
//*******************************************************
//LEGGI UN VALORE INTERO DUE BYTE  XX  POSIZIONE ASSE
//TORNA 1 SE ERRORE
//tipo=0 valore in quota attuale
//tipo=1 valore in quota richiesta
//tipo=2 offset impostato
//tipo=3 delay Start Stop impostato
//*******************************************************
char lettura_intero(char tipo)
```

Routine analoga alla precedente ma utilizzata per la lettura di un valore composto da un solo byte.

```
//*******************************************************
//LEGGI UN BYTE - VELOCITA' IMPOSTATA
//TORNA 1 SE ERRORE
//TIPO 0 VELOCITA' ASSE
//TIPO 1 NUMERO ASSE
//TIPO 2 CODICE ERRORE
//TIPO 3 STATO ASSE
//*******************************************************
char lettura_byte(char tipo)
```

Microcontrollori DSPIC.

Le 4 routine per i due DSPIC sono quelle di lato, e permettono rispettivamente di:
leggere il comando, leggere un byte inviare un byte ed inviare un valore intero su due byte.
A differenza del microcontrollore principale, nel DSPIC non viene utilizzato l’hardware interno, ma vengono utilizzati due pin digitali.

```
//*******************************************************
//LETTURA COMANDO DA SERIALE
//*******************************************************
char lettura_seriale()
```

```
//*******************************************************
//LEGGI SINGOLO BYTE
//*******************************************************
char leggi_byte()
```

```
//*******************************************************
//INVIA SINGOLO BYTE
//*******************************************************
void invia_byte(char valore)
```

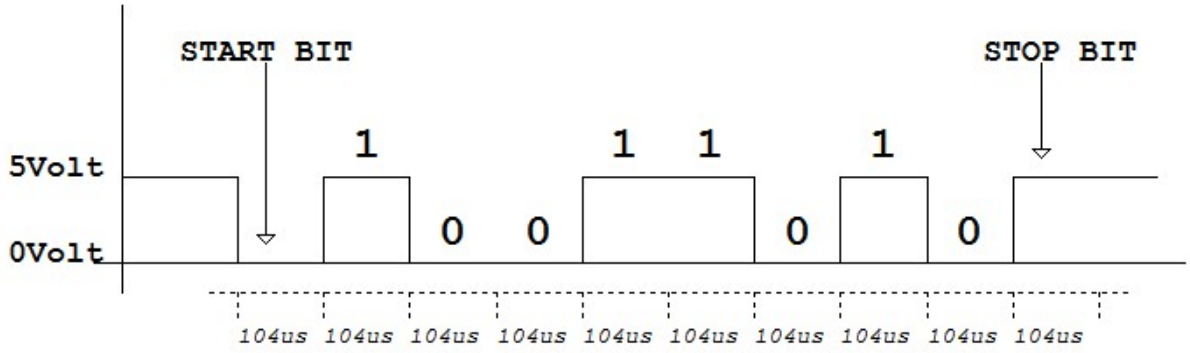
```
//*******************************************************
//INVIA VALORE INTERO XX#
//torna 1 se ERRORE tramissione
//*******************************************************
char invia valore(int valore)
```

Analizziamo il funzionamento della prima routine che consente di leggere il comando senza utilizzare l'hardware interno.

```
char lettura_seriale()
 {
  char cont_char,cont_bit;
  unsigned int cont_time;

  //attesa risposta
  cont_char=0;
  while(cont_char<5)
   {
    dato_rx[cont_char]=leggi_byte();
    delay_us(100);
    invia_byte(dato_rx[cont_char]); //invio eco carattere ricevuto
    if(dato_rx[cont_char]=='#') return(0);
    cont_char++;
    cont_time=30000;
    while(RX_232)
     {
      delay_us(1);
      cont_time--;
      if(!cont_time) return(1);
     }//while(cont_time)
   }//while(cont_char<5)
  return(1);
}//lettura_seriale
```

Innanzitutto possiamo vedere che la routine "lettura_seriale" restituisce il valore 0 se non ci sono errori.
I byte ricevuti vengono memorizzati nel vettore dato_rx[5] contenente 5 elementi.
Il pin di ricezione è il pin 3 della porta F, definito nel programma RX_232.
All'interno della routine c'è un ciclo while, che viene ripetuto fino a quando non vengono ricevuti 5 caratteri, o fino a quando non viene ricevuto il carattere di chiusura #.
C'è inoltre un controllo per verificare se non arrivano caratteri, tramite il decremento della variabile cont_time, inizializzata a 30000. Se non c'è il bit di start sul pin RX_232 (cioè un livello basso) per un ciclo ripetuto 30000 volte con un delay di 1us, allora viene restituito il valore 1, corrispondente ad un errore.

All'interno di questa routine vediamo la chiamata ad altre due routine e cioè, leggi_byte ed invia_byte.
In pratica viene letto il byte con la prima delle due, salvato nel vettore dato_rx ed inviato come eco con invia_byte.
Analizziamo di seguito le due routine menzionate. Prima di farlo però dobbiamo ricordare come avviene l'invio di un byte sulla seriale alla velocità di 9600bps (bit per secondo).



La routine leggi_byte viene avviata se c'è un livello basso sul pin RX_232, pertanto la prima cosa da fare è attendere un tempo di 100us, per far passare il bit di start.
Successivamente si aspettano 50us, per posizionarsi al centro del primo bit che è il meno significativo.
A questo punto viene avviato un ciclo while che si ripete 8 volte (cont_bit=8) in questo ciclo si testa il valore del pin RX_232, ed il suo valore, 0 o 1 viene dato al bit 7 della variabile dato_leggi.
Ogni volta che viene letto il bit la variabile dato_leggi, viene shiftata a destra di una posizione in modo da avere al termine delle 8 letture il byte nella variabile.


```
char leggi_byte()
 {
  char cont_bit,dato_leggi;

  cont_bit=8;
  //attesa start bit
  Delay_us(100);      //attesa start bit
  //ricezione byte
  Delay_us(50);       //attesa metà bit
  dato_leggi=0;
  while(cont_bit)
   {
     dato_leggi=dato_leggi>>1;
     if(RX_232)    {dato_leggi.F7=1;}
     else          {dato_leggi.F7=0;}
     Delay_us(100);
     cont_bit--;
   }//while(cont_bit)
  Delay_us(150);  //stop bit + mezzo bit
  return(dato_leggi);
}//leggi_byte()
```

Nella routine invia_byte, avviene il meccanismo inverso, in pratica alla routine viene passato un valore da inviare, il byte viene inviato bit per bit dopo 104us (start bit) mediante un ciclo while eseguito per 8 volte (cont_bit=8).
All'interno del ciclo viene letto il bit 0 della variabile che contiene il byte da inviare, ed a seconda del suo valore viene settato o resettato il pin TX_232 che nel nostro caso corrisponde a pin 2 della porta F.
Anche in questo caso la variabile viene shiftata a destra dopo ogni invio. Tra un invio e l'altro c'è un tempo di 104us.


```
void invia_byte(char valore)
{
  char cont_bit;

  //INVIO BYTE
  cont_bit=8;
  TX_232=0;   //start bit
  Delay_us(104);
  while(cont_bit)
    {
      if(valore.F0)     TX_232=1;
      else              TX_232=0;
      Delay_us(104);
      valore=valore>>1;
      cont_bit--;
    }//while(cont_bit)
  TX_232=1;
  delay_us(104);
}//invia_byte(char valore)
```

Può sembrare un metodo un po' artigianale, ma una volta regolati i tempi (da notare che nella routine di lettura non c'è un'attesa di 104 ma di 100us) si raggiunge un adeguato livello di sicurezza.
Bisogna comunque considerare il fatto che l'invio dell'eco del carattere, il controllo del numero di byte e l'invio del carattere di chiusura, offrono un'adeguato controllo sulla correttezza dei valori inviati.
Restano sicuramente più affidabili i più standard controlli di errore, che magari verranno implementati successivamente con un maggiore tempo a disposizione.

Con la seriale RS232 però non vengono inviati tutte le istruzioni, infatti come si può notare dallo schema della scheda o anche dallo schema a blocchi, ci sono delle linee di comando che tramite dei segnali digitali consentono l'invio immediato dei seguenti comandi.

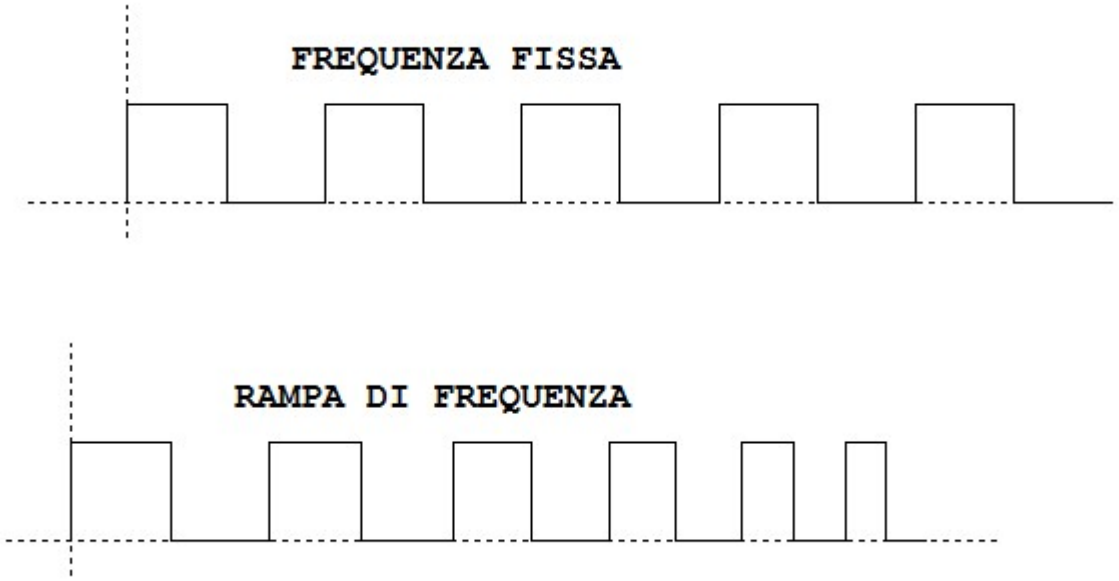
```
RD3-RD2-RD1-RD0
0   0   0   0            Nessuna operazione

RD3-RD2-RD1-RD0
Con gestione CS azione singolo asse
0   0   0   1            Avvia posizionamento in start stop
0   0   1   0            Avvia posizionamento singolo con rampa
0   0   1   1            Avvia posizionamento multiplo con rampa
0   1   0   0            Muovi in meno
0   1   0   1            Muovi in più
0   1   1   0            Azzeramento
0   1   1   1            ----------

RD3-RD2-RD1-RD0
senza gestione CS azione simultanea entrambi gli assi
1   0   0   0            Avvia posizionamento in start stop
1   0   0   1            Avvia posizionamento singolo con rampa
1   0   1   0            Avvia posizionamento multiplo con rampa
1   0   1   1            Azzeramento
1   1   0   0            ----------
1   1   1   0            ----------
1   1   1   1            ----------
```

In pratica con i 4 bit digitali del PIC principale RD3,RD2,RD1 ed RD0, si gestiscono le richieste di avvio del posizionaento, movimento in più o in meno o azzeramento.
Sono previste due modalità, la prima con l'utilizzo del Chip Select, perciò il comando è indirizzato ad un solo asse, la seconda invece avviene senza l'utilizzo del Chip Select, perciò il comando è indirizzato ad entrambi i DSPIC che lo eseguiranno simulaneamente.
In questo modo si uò avere una notevole flessibilità potendo così soddisfare diverse esigenze.

CONTROLLO MOTORI

Il controllo del movimento del motore è affidato interamente ad ogni DSPIC, che dovrà generare un treno di impulsi sul pin CLOCK, oltre che impostare correttamente gli altri due bit, ENABLE e DIREZIONE.

Un motore passo-passo può essere gestito tramite un CLOCK a frequenza fissa, solo se inviamo una frequenza corrispondente a circa un giro al secondo. Se dobbiamo inviare frequenza superiori, occorre realizzare una rampa di frequenza sia in salita che in discesa.



La routine che realizza il CLOCK a frequenza fissa è relativamente facile e la si può realizzare con un semplice ciclo while nel seguente modo:

nella variabile passi_totali ci sono i passi da fare e successivamente nel ciclo while viene messo ad 1 ed a 0 il pin di CLOCK, intervallando le due operazioni con il ritardo necessario per ottenere una velocità non superiore al giro al secondo.

```
conteggio_passi=0;
while(conteggio_passi<passi_totali)
 {
  CLOCK=1;
  delay_us(1000);
  CLOCK=0;
  delay_us(1000);
  conteggio_passi++;
  pos_attuale=pos_attuale+valore;
 }//while(conteggio_passi<passi_totali)
```

Invece per realizzare una rampa in frequenza, il meccanismo è più complesso. Aiutandosi con un foglio elettronico, sono stati fatti dei calcoli per determinare i valori da inserire in due vettori.

```
unsigned short const rampa_passi [96] = {
1,1,1,1,1,1,1,1,1,1,1,2,2,2,2,2,2,2,2,2,3,3,3,3,3,3,3,3,3,4,4,4,4,4,4,4,
4,4,5,5,5,5,5,5,5,5,6,6,6,6,6,6,6,6,7,7,7,7,7,7,8,8,8,8,8,8,8,9,9,9,9,9,9,
10,10,10,10,10,10,10,11,11,11,11,11,11,11,11,11,12};
//dt=4us
unsigned short const rampa_time [96] = {
250,216,189,169,152,139,128,118,110,102,96,91,86,81,77,74,70,67,64,62,60,57,55,
53,52,50,48,47,46,44,43,42,41,40,39,38,37,36,35,35,34,33,32,32,31,30,30,29,29,
28,28,27,27,26,26,26,25,25,24,24,24,23,23,23,22,22,22,21,21,21,20,20,20,20,19,19,
19,19,19,18,18,18,18,18,17,17,17,17,17,16,16,16,16,16,16,15};
```

I due vettori vengono utilizzati nella routine posiziona_asse.

```
//--------------------------------
//ESEGUO IL POSIZIONAMENTO
//torna 0 se ok 99 se emergenza
char posiziona_asse()
```

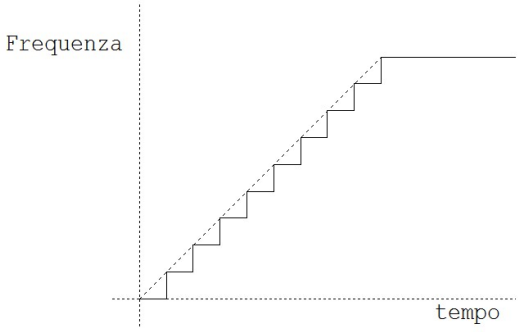
Di questa routine analizziamo la sola rampa di salita.

In pratica nel vettore rampa_time, ci sono i valori da dare alla variabile cont_del generando così un CLOCK ad una determinata frequenza.

Nel vettore rampa_passi ci sono i passi da fare prima di variare ancora la frequenza di CLOCK.

```
while(cont_pos<cont_vel)
 {
  temp_rampa=rampa_passi[cont_pos];
  while(temp_rampa)
   {
    CLOCK=1;
    conteggio_passi++;
    pos_attuale=pos_attuale+valore;
    cont_del=rampa_time[cont_pos];
    while(cont_del)
     {
      Delay_us(10);
      cont_del--;
     }//while
    CLOCK=0;
    cont_del=rampa_time[cont_pos];
    while(cont_del)
     {
      Delay_us(10);
      cont_del--;
     }//while
    temp_rampa--;
   }//while
  cont_pos++;
  if(EMERGENZA)   {IN_MOTO=0;return(99);}  //emergenza
  if(ALLARME)    {IN_MOTO=0;return(1);}
 }//while
```

L'idea è quella di eseguire determinati cicli di CLOCK ad una certa frequenza, aumentandola gradualmente, in modo da ottenere la seguente rampa di frequenza.



Il numero di passi ed il tempo, sono stati calcolati in modo da ottenere dei gradini come in figura.

Ovviamente nei primi gradini, ci sono meno passi con tempi più lunghi (frequenze più basse).

Non è facile descrivere nei dettagli il funzionamento di tutte le routine, ma il programma allegato e commentato potrebbe aiutare maggiormente.
Il DSPIC inoltre si occupa di controllare l'azzeramento dell'asse, e di eseguire tutti i comandi impartiti dal microcontrollore principale. I comandi vengono letti e decodificati nel main principale.

CALCOLO VALORI

Il compito di applicare tutte le formule viste precedentemente nel capitolo relativo allo studio della cinematica, è affidato al microcontrollore principale tramite le seguentuìi routine.

Il calcolo dei valori viene effettuato all'avvio ed il risultato viene salvato in due vettori dati contenenti angoli ed impulsi.

```
//*******************************************************
//ARROTONDA UN NUMERO REALE
//*******************************************************
int arrotonda(float numero_real)
```

La routine calcola_angoli, consente di calcolare gli angoli dei due motori, partendo dalle coordinate cartesiane.

```
//*******************************************************
//CALCOLO ANGOLI
//*******************************************************
//calcola gli angoli partendo dalle coordinate
//formula Kinematics inversa
//ritorna 0 s ok 1 se posizione non possibile
char calcola_ang(float Xc,float Yc)
```

La routine calcola coordinate consente di ottenere le coordinate partendo dal numero della casella dell'elemento.

```
//*******************************************************
//CALCOLA COORDINATE
//*******************************************************
//trova le coordinate dal numero della casella
//ritorna 0 s ok 1 se posizione non possibile
char calcola_coord(char numero)
```

DISPLAY E GESTIONE MENU

Anche questa parte viene gestita dal microcontrollore principale, in pratica con i 4 pulsanti e con il display LCD, è stata realizzata una semplice ed essenziale interfaccia operatore, per svolgere le seguenti funzioni:

- azzeramento manipolatore
- lettura codici di errore
- azzeramento errori
- avvio ciclo rapido con rampe di frequenza (i due motori possono arrivare al termine del proprio spostamento in tempi differenti)
- avvio ciclo lento senza rampe di frequenza (i due motori arrivano al termine del proprio spostamento insieme)
- Movimenti manuali
- Ipostazione degli offset di zero dei due assi
- Visualizzazione degli angoli e degli impulsi calcolati.

Le routine utilizzate per la gestione del display e dell'interfaccia operatore sono le seguenti:

Con i 2 pulsanti up e down si può scorrere nei vari menu, con i pulsanti esc e invio, invece si può uscire o dare conferma della scelta.

Il menù iniziale viene visualizzato all'avvio, mentre invece il menù manuale viene visualizzato se viene scelto nel menù iniziale di eseguire i movimenti manuali.

Negli altri casi occorre seguire le indicazioni date nel display.

```
//*****************************************************
//VISUALIZZA IL MENU' INIZIALE
//RITORNA DA 1 A 8 SECONDO L'OPZIONE RICHIESTA
//char txt_montani="ITT MONTANI";
//char txt_azzera="AZZERA";          return(1)
//char txt_errori="ERRORI";          return(2)
//char txt_reset="RESET ALLARMI";    return(3)
//char txt_ciclo_rapido=CICLO RAPIDO";      return(4)
//char txt_ciclo_lento="CICLO_LENTO";        return(5)
//char txt_manuale="MANUALE";        return(6)
//char txt_manuale="OFFSET ZERO";    return(7)
//char txt_manuale="VALORI";         return(8)
//*****************************************************
char menu_iniziale()
```

```
//*****************************************************
//VISUALIZZA IL MENU' MANUALE
//RITORNA DA 1 A 5 SECONDO L'OPZIONE RICHIESTA
//*****************************************************
char menu_manuale()
```

```
//*****************************************************
//MOVIMENTI MANUALI
//RITORNA 0 SE ASSI AZZERATI SENZA ERRORE
//*****************************************************
char movimenti manuali()
```

E' molto lungo e complesso descrivere tutte le funzioni e le routine, ci sono ad esempio quelle relative al controllo del ciclo automatico, e molte altre non menzionate, ma lo scopo era dare un'indicazione di massima del funzionamento del software.
Ovviamente chi vuole, può di seguito analizzare ogni riga di codice dei due software, allegati nelle pagine successive.

Come si può leggere nella parte iniziale del codice di seguito allegato, ci sono ancora delle parti di programma da sviluppare, al momento della stesura di questo documento infatti manca ad esempio, la parte del controllo del ciclo tramite la pulsantiera esterna.
Queste verranno sviluppate appena avremo a disposizione il manipolatore funzionante, che al momento è in fase di realizzazione meccanica.

Questa è comunque una prima versione del software, sicuramente con il prototipo funzionante, potremo aggiungere altre funzioni, come ad esempio la possibilità di interagire con il manipolatore tramite un'app, cosa questa che richiede una piccola scheda aggiuntiva, ma che è sicuramente realizzabile.