A.S.
2008/2009

Armatronick

Progetto Di Un Braccio Robotizzato

Andrea Aglietti

I.T.I.S. Leonardo Da Vinci Firenze

A.S. 2008/2009

Sommario

Introduzione

Il progetto nasce con l'idea di combinare meccanica elettronica e informatica per costruire un automa in grado di emulare un braccio impiegato in un processo produttivo industriale, nel quale vengono scartati dei pezzi, che prelevati da esso sono collocati in un'altra locazione decisa dall'utente.

Questo punto di partenza è stato pian piano rimodellato procedendo per passi fino ad ottenere il prototipo finale; di seguito è riportata la descrizione del braccio allo stato attuale e di alcuni passaggi intermedi.

Meccanica

- ### L'idea iniziale

  Seguendo l'idea di partenza il progetto è partito pensando a un possibile processo produttivo strutturato in questo modo:

  una catena di montaggio in cui vengono costruiti pezzi in modo automatico; i pezzi sono movimentati da un nastro trasportatore che li fa scorrere attraverso le varie macchine utensili fino ad arrivare a un'unità di controllo in grado di determinare se il pezzo è a norma o occorrono ulteriori lavorazioni in quanto è stato rilevato un difetto di produzione.

  L'unità di controllo in questione è costituita dal braccio che deve essere in grado di determinare la presenza di un pezzo anomalo sul nastro trasportatore e di prelevarlo spostandolo in un'altra locazione, per ovvi motivi la sua azione non potrà essere immediata per cui questo tipo di controllo può essere impiegato in catene di produzione relativamente lente (ad. es. un pezzo al minuto) per dare tempo all'automatismo di svolgere la sua funzione.

  Sulla base di questo ipotetico processo il primo passo verso la realizzazione è stato quello di reperire i materiali e stabilire le modalità di simulazione: non sarebbe stato possibile realizzare l'intero sistema (nastro trasportatore e braccio), per cui si è deciso di realizzare il braccio integralmente ma di simulare la presenza di un nastro trasportatore con dei sensori fotosensibili (fotocellule) che rilevano un pezzo inserito dall'utente su un'apposita base.

Il passo successivo all'impostazione generale del progetto è stata la realizzazione meccanica delle varie parti; essa ha richiesto un notevole sforzo in termini di tempo e di risoluzione dei problemi inizialmente non previsti; l'approccio alla costruzione è stato di tipo sperimentale senza aver realizzato preventivamente un progetto disegnato e senza l'ausilio di particolari macchine utensili in quanto il tutto è stato realizzato in casa, si è proceduto quindi per continui miglioramenti sulla base dell'esperienza via via accumulata fino ad arrivare al prototipo finale.

- Reperimento materiali

Prima di poter iniziare la realizzazione si è dovuto reperire il materiale opportuno seguendo la filosofia del risparmio ovvero andando a cercare i vari pezzi in apparecchi in disuso o da rottamare; si sono utilizzati due motori passo passo ricavati da una vecchia stampante, una puleggia di una vecchia radio a valvole termoioniche (dove veniva usata per movimentare un condensatore variabile in aria per variare la frequenza di ricezione), il mozzo cuscinettato di una bicicletta e pezzi di alluminio provenienti da surplus industriale; le uniche parti che si sono dovute acquistare sono stati dei trafilati in allluminio, un motore con riduttore in corrente continua per la movimentazione della pinza e del compensato e altri pezzi di legno per la costruzione della base.

Utilizzando questi pezzi recuperati si è iniziato la costruzione di un primo prototipo dotato di una base in legno su cui era fissato il mozzo e sul quale si trovava una base in grado di ruotare, la base era movimentata da un motorino passo passo con una trasmissione a cinghia piana e su di essa era posta una torretta con il braccio vero e proprio infulcrato e in grado di ruotare verso l'alto e verso il basso tramite una puleggia, movimentata anch'essa, grazie a un motorino passo passo e una cinghia piana: la foto riportata di seguito rende molto meglio l'idea.

Questo primo "modello" è stato utilizzato per effettuare varie prove a livello elettronico ma è servito anche per avere una base meccanica su cui poter elaborare il modello finale con tutte le sue componenti aggiuntive.

- Il passaggio dal primo prototipo al modello finale

Il modello finale si differenzia molto dal primo prototipo ma in linea di massima conserva lo stesso principio di funzionamento per quanto riguarda la rotazione della base e lo spostamento verso l'alto o verso il basso del braccio; le parti che sono state aggiunte e fanno ora parte del prototipo sono le seguenti:

La Pinza

Essa ha la funzione di afferrare il pezzo e rilasciarlo al momento opportuno, anch'essa non è nata in modo definitivo ma ha subito un

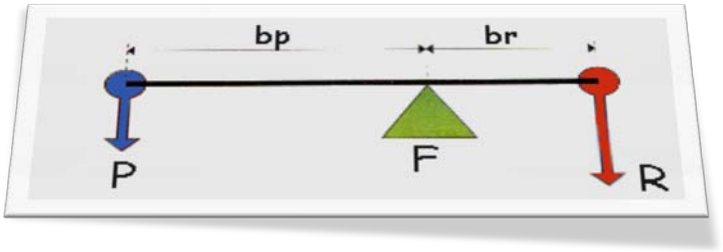
progressivo miglioramento; un primo modello è stata costruito secondo uno schema iniziale di prova per poi passare a quello riportato nella foto.
Il secondo modello (quello attuale) utilizza un motorino in corrente continua per movimentare i due bracci della pinza, essa infatti è dotata di due estensioni di alluminio infulcrate su di una base e imperniate come si vede nella foto a un supporto libero di scorrere sulla filettatura di una vite quando questa gira, la rotazione è innescata dal motorino che è collegato meccanicamente ad essa tramite un rapporto di riduzione a ingranaggi permettendo di avere una buona forza di presa.

Il contrappeso

La prima realizzazione era formata da un braccio in grado di ruotare verso il basso o verso l'alto infulcrato a circa metà della sua lunghezza, in realtà il braccio era ed è tuttora costituito da due trafilati posti uno sopra l'altro parallelamente e infulcrati allo stesso punto (la metà), utilizzando questo accorgimento si è riusciti a mantenere il piano della pinza parallelo al terreno indipendentemente dall'altezza del braccio da terra.
Sfruttando questa particolarità si è posto un contrappeso sulla parte opposta della leva per dar vita a un momento antagonista a quello dato dal peso della pinza per la lunghezza del braccio da essa al fulcro, esso è stato posizionato a una distanza molto minore dal fulcro rispetto alla pinza in modo da rendere la meccanica più compatta e meno antiestetica.
Dal lato tecnico questo accorgimento è servito a ridurre la forza necessaria al motore passo passo per sollevare la leva, essa si riduce, in condizione di equilibrio, alla sola forza di attrito tra le parti meccaniche in movimento; inoltre si ha una migliore distribuzione delle masse intorno all'asse di rotazione, utilizzare un contrappeso a una distanza considerevole dall'asse avrebbe comportato un aumento del momento dinamico di eccessivo valore.
Il momento dinamico è proporzionale al quadrato delle masse e alla distanza che esse presentano rispetto all'asse di rotazione ed esprime l'inerzia legata al sistema per cui è preferibile avere una bassa inerzia in quanto il motore passo passo che aziona la base facendola ruotare, imprime ad essa variazioni di velocità anche repentine che porterebbero a oscillazioni e a perdite del passo nel caso in cui il momento dinamico fosse particolarmente elevato.

Il contrappeso è stato realizzato fondendo del piombo all'interno di una piccola bomboletta di alluminio e la sua massa non è stata calcolata ma è stata ricavata per tentativi fino ad ottenere il giusto valore tale che alla distanza scelta dall'asse fosse in grado di equilibrare la leva; volendolo calcolare le formule relative alla leva sono riportate di seguito.

Leva Del Primo Tipo P = Pinza – R = Contrappeso



$mr = massa\ del\ contrappeso$ [1.0]

$mp = massa\ della\ pinza$ [1.1]

$P = mp * 9{,}8\frac{m}{s^2}; R = mr * 9{,}8\frac{m}{s^2}$ [1.2]

$P * bp = R * br$ [1.3]

$R = \frac{P*bp}{br} \rightarrow mr * 9{,}8\frac{m}{s^2} = \frac{P*bp}{br}$ [1.4]

$mr = \frac{P*bp}{br*9{,}8\frac{m}{s^2}}$ [1.5]

Come evidenziato dalle formule è possibile calcolare la massa del contrappeso necessaria a mantenere in equilibrio la leva conoscendo la distanza della pinza e del contrappeso dal fulcro.

- **La base**

Un altro elemento che è stato drasticamente rivisto è la base su cui poggia l'intero meccanismo, inizialmente infatti era costituita da un semplice supporto di legno multistrato di spessore di circa 1 cm mentre nel modello finale è un vero e proprio supporto tridimensionale a tronco di piramide al cui interno trovano alloggiamento il mozzo per la rotazione, il motore che aziona la base e le varie parti elettriche ed elettroniche necessarie al funzionamento.
Tutta la meccanica è visibile nella seguente foto del prototipo finale del braccio, come si può notare dall'immagine non sono presenti soltanto parti prettamente meccaniche ma anche dispositivi elettronici e collegamenti; tutto ciò è descritto nella sezione elettronica del progetto riportata di seguito.

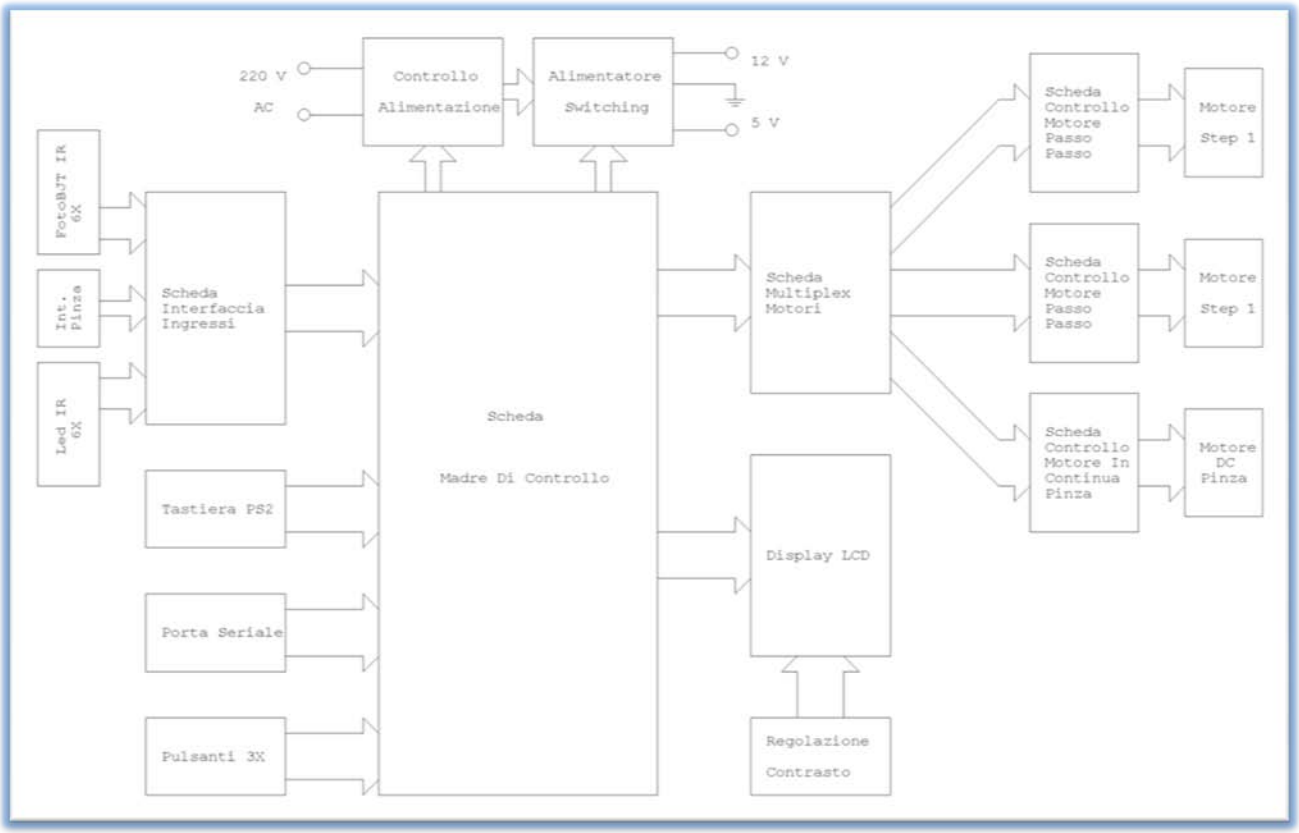
Elettronica

- Prefazione

Anche l'elettronica è stata sviluppata procedendo per passi anche se per essa era stato impostato uno schema a blocchi generale di funzionamento sul quale si sono poi modellate le varie schede in funzione delle problematiche da risolvere incontrate via via nella realizzazione del progetto.

Come accennato nella sezione "meccanica" il braccio è dotato di una serie di dispositivi elettronici alloggiati su di esso per poter funzionare, tutto ciò si combina poi con una consolle di controllo sulla quale si trovano le schede elettroniche grazie alle quali è possibile gestire la meccanica del braccio; di seguito è illustrato il funzionamento di ogni scheda e sono riportate le caratteristiche dei componenti utilizzati per ciascuna di esse, è inoltre riportato uno schema generale e a blocchi delle unità che si trovano sulla consolle.

Schema a blocchi sistema



- Introduzione al funzionamento del sistema

Come si può vedere dallo schema il sistema è strutturato a moduli e ogni scheda praticamente realizzata corrisponde a uno di essi, operando in questo modo il sistema può essere aggiornato in un secondo tempo con la possibilità di aggiungere altri moduli dato che ognuno di essi è realizzato su una scheda differente collegata alle altre tramite dei connettori di facile rimozione.

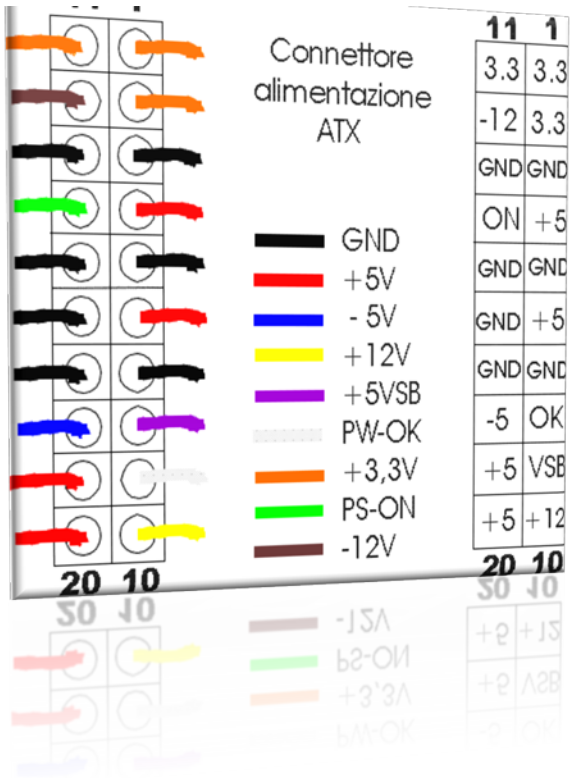
Lo schema può essere suddiviso in quattro grandi blocchi,: periferiche di ingresso, unità di controllo costituita da una scheda a microcontrollore PIC, periferiche di uscita e il blocco di alimentazione costituito dall'alimentatore e da un'opportuna scheda di controllo; di seguito sono analizzati in dettaglio i singoli blocchi.

- Alimentazione

Scheda Controllo Alimentazione e Alimentatore

La sezione dell'alimentatore è riportata nel primo foglio degli schematici ed è costituita dai due blocchi "scheda controllo alimentazione" e "alimentatore".
Si è scelto di utilizzare un vecchio alimentatore ATX per personal computer avente le seguenti caratteristiche: esso fornisce varie tensioni stabilizzate di 12 V, 5 V, 3,3 V, -12 V +12 V, -5 V ed è in grado di erogare una corrente di svariati ampere su ogni linea, per il progetto si sono utilizzate soltanto le linee a + 12 e 5 volt (ampiamente sovradimensionate riguardo alla potenza massima erogabile) impiegate per alimentare rispettivamente la parte di potenza costituita dai motori e la sezione logica del circuito (microcontrollore, scheda di controllo sensori, scheda di multiplexing motori).
L'alimentatore non ha richiesto quindi particolari lavorazioni ma è stato sufficiente cortocircuitare due pin interni ad esso per farlo funzionare correttamente, essendo utilizzato per i PC può essere acceso cortocircuitando verso massa un opportuno piedino (pilotato dalla scheda madre in cui una parte di essa viene tenuta costantemente alimentata da un piccolo trasformatore interno all'alimentatore sempre attivo anche nel caso in cui questo sia spento) del computer sul connettore ATX, per questo motivo il piedino PS-ON è stato cortocircuitato a massa trasformando così l'alimentatore da PC in un comune alimentatore switching stabilizzato senza alcun tipo di controllo in bassa tensione sull'accensione.

Pinout Connettore ATX

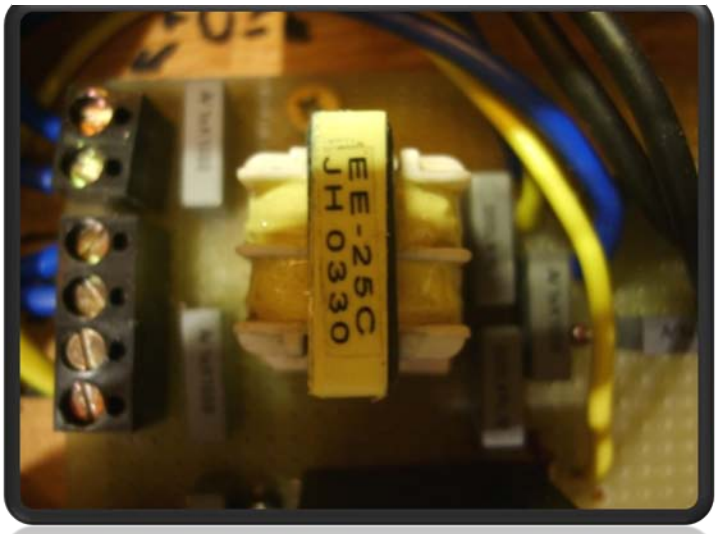


Si è effettuato tale modifica, non per eliminare il sistema di accensione e spegnimento da una bassa tensione, ma per portare questo esternamente all'alimentatore ed essere sicuri di non avere alcun circuito alimentato in caso di sistema spento.
Il sistema di controllo dello spegnimento è infatti integrato nella scheda controllo alimentazione, come visibile dallo schema entro di essa si trovano circuiti di potenza formati da relais e un filtro EMI.

Il filtro EMI è costituito dalle due induttanze L_1 e L_2 e dai condensatori C_1 C_2 C_4 C_5 e grazie ad esso è possibile ridurre le interferenze elettromagnetiche generate all'accensione (

spikes) e allo spegnimento del circuito o in caso di forti variazioni di assorbimento di corrente dalla rete; esso si comporta infatti come un filtro passa basso e impedisce alle alte frequenze di arrivare sulla rete e viceversa; le due induttanze sono avvolte sullo stesso nucleo per attenuare le interferenze di modo comune mentre i condensatori bypassano le alte frequenze verso terra.

Filtro EMI



I filtri EMI trovano applicazione in praticamente tutti i dispositivi elettrici ed elettronici connessi alla rete, molto spesso serve per far rientrare nella norma l'emissione elettromagnetica di tali circuiti verso la rete e impedire ad essi che possano essere danneggiati da disturbi presenti su di essa.

La parte restante del circuito in cui sono utilizzati i due relais è il vero e proprio sistema di controllo per lo spegnimento e funziona in questo modo; il pulsante Sw_1 normalmente aperto, che viene premuto ogni volta che si vuole accendere il circuito, nel momento della pressione fa scorrere corrente nella bobina di RL_1 eccitandolo e portando tensione sull'ingresso dell'alimentatore; in un lasso di tempo molto breve (circa 50 ms) l'alimentatore si porta a regime e alimenta i vari circuiti tra cui anche la scheda madre, dalla quale proviene un segnale alto a 5 V che porta in saturazione Q_1, esso permette a RL_2 di eccitarsi e bypassare il pulsante mantenendo RL_1 eccitato fintanto che la scheda madre non "richiede" lo spegnimento portando basso nuovamente il segnale Off.

Scheda alimentazione – relais 220 V

Si è realizzato così un controllo sullo spegnimento che permette alla scheda madre di interagire con l'alimentatore determinando lo spegnimento del circuito senza avere collegamento elettrico tra parte ad alta tensione e a bassa tensione: si è preferito infatti l'utilizzo di relais a quello di TRIAC o SCR per garantire un certo livello di sicurezza e isolare galvanicamente il circuito a bassa tensione da quello ad alta; inoltre per avere il sistema alimentato occorre che i relais siano eccitati entrambi per cui in caso di malfunzionamento, ad es. un cortocircuito su una linea di alimentazione, uno dei due relais interrompe i suoi contatti ed evita ulteriori danni.

I circuiti formati da una resistenza e un condensatore presenti su ogni interruzione del circuito sono le così dette reti snubber, esse evitano lo scintillio sui contatti dei relais (prolungandone la vita) e permettono di avere commutazioni con minori emissioni di interferenze ad alta frequenza, la rete R_4 C_7 anche se utilizzata per il transistor ha la stessa funzione e si completa con il diodo di ricircolo D_1; esso è un diodo veloce che cortocircuita le tensioni inverse generate dalla bobina del relais durante le commutazioni impedendo la distruzione del transistor che altrimenti potrebbe danneggiarsi.

Scheda controllo alimentazione

Infine si può vedere nello schema un diodo LED polarizzato dalla resistenza R_5, esso si accende ogni volta che viene alimentato il circuito ed è posizionato sul frontale della consolle come visibile dalle foto.
Questa scheda a differenza delle altre è l'unica realizzata su basetta millefori, si è utilizzato questo supporto data la semplicità del circuito per cui la realizzazione di un circuito stampato avrebbe comportato soltanto una perdita di tempo per disegnarlo e realizzarlo praticamente.

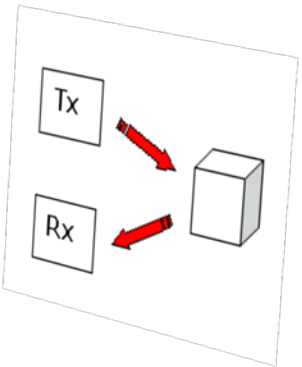
- Scheda controllo sensori

Questa scheda gestisce tutti gli ingressi provenienti da sensori alloggiati sul braccio: esso è dotato di cinque sensori infrarossi e uno di contatto.

Tipi di sensore utilizzati e loro alloggiamento

Per quanto riguarda i tipi essi sono tre: sensore di prossimità a infrarossi, barriera infrarossa e microswitch come sensore di contatto; analizziamo nel dettaglio il funzionamento di ogni tipologia:

- **Sensore di prossimità**



Questa tipologia è utilizzata per rilevare la presenza di un pezzo ed è posizionato al di sotto della pinza; il funzionamento è il seguente: un emettitore di luce infrarossa (LED IR) emette un

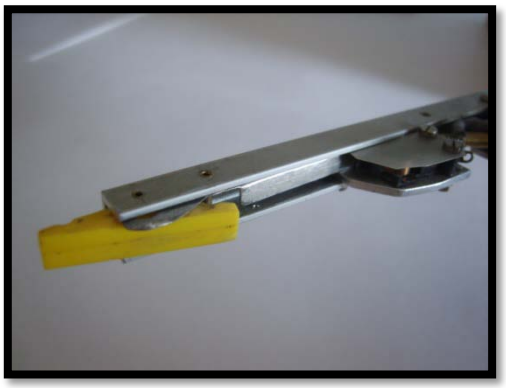
fascio luminoso che può essere riflesso o meno; se il pezzo è presente ed è sufficientemente riflettente una parte considerevole della radiazione viene riflessa e può essere rilevata da un fotodiodo posizionato accanto al led emettitore.

- **Barriera infrarossa**

La barriera infrarossa non è altro che una fotocellula: un LED IR emette radiazione luminosa che viene captata da un ricevitore posto di fronte ad esso (fotodiodo IR),

se il fascio viene interrotto da un opportuna bandierina funzionante come finecorsa si ha un indicazione sulla posizione di una parte meccanica del pezzo ad esempio le due leve.
Questa tipologia è utilizzata come finecorsa per evitare movimenti non consentiti al braccio (ad es. spostamenti oltre il limite superiore o inferiore); essi sono impiegati per la rotazione della base dove si ha la continuità del fascio soltanto nel caso in cui essa si trovi in posizione 0 ovvero di partenza e quindi il braccio sia posizionato al di sopra del supporto per il pezzo.
Altre due barriere infrarosse vengono utilizzate come finecorsa superiore e inferiore per fornire un punto di riferimento al microcontrollore sulla posizione e per evitare spostamenti oltre tale limite; infine l'ultima è usata per rilevare la presenza di un pezzo sulla base che simula il nastro trasportatore.

- **Microswitch sensore di contatto**



Data l'impossibilità di utilizzare un sensore infrarosso per rilevare l'effettiva presa del pezzo da parte della pinza, si è utilizzato un microswitch che si chiude, venendo azionato da un sistema a leva, quando la pinza ha afferrato il pezzo.
Unitamente alla resistenza di pull up, visibile nello schema, fornisce uno 0 logico se il pezzo è stato afferrato e un 1 in condizioni di normalità.

Per migliorare le caratteristiche dei sensori si è scelto di modulare il fascio luminoso dei led infrarossi pilotandoli con un'onda quadra alla frequenza di circa 11,4 Khz; grazie a questo accorgimento si sono resi immuni i ricevitori alla luce esterna utilizzando un opportuno circuito ricevitore di filtraggio di tipo passa banda sensibile quindi alla sola frequenza utilizzata in trasmissione; per poter realizzare ciò si è reso necessario l'utilizzo del timer 555 come trasmettitore e di circuiti ricevitori a operazionali e tone decoder a PLL.

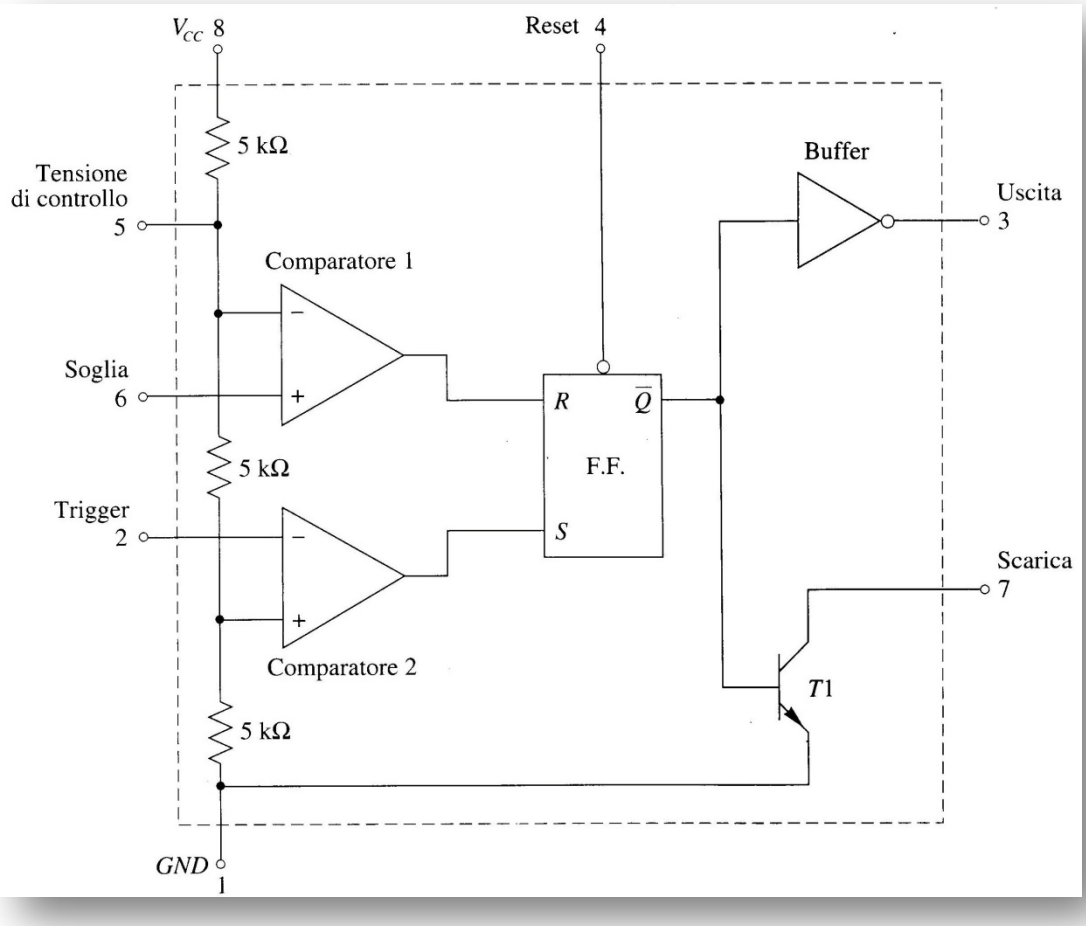
Il timer 555

Il timer 555 è un integrato multiuso introdotto a partire dagli anni '70, grazie ad esso è possibile realizzare vari circuiti multivibratori (monostabili, bistabili, astabili); in questo caso è utilizzato come astabile per generare un'onda quadra di frequenza variabile (per centrare finemente la frequenza di centro banda del filtro passa banda usato in ricezione) e ampiezza 5 volt; di seguito è riportato il principio di funzionamento e il calcolo dei componenti.

È stato scelto di utilizzare questo circuito integrato data la sua larga diffusione e flessibilità, altrimenti si sarebbero potuti utilizzare altri tipi di multivibratori astabili realizzati ad esempio con amplificatori operazionali o porte logiche.

Funzionamento nella configurazione astabile

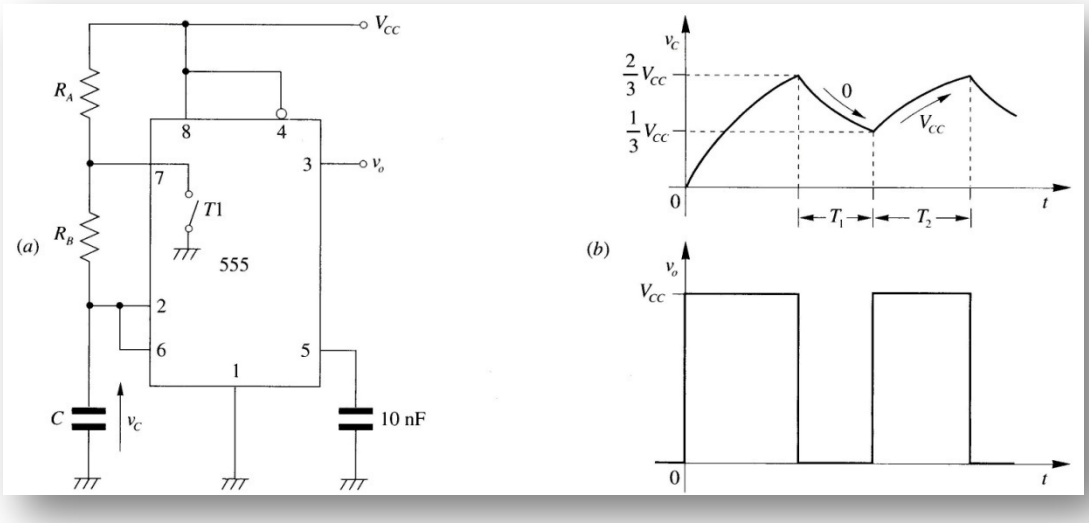
Schema Interno Del Timer 555



Come intuibile dallo schema interno il 555 presenta tre stati stabili dell'uscita in relazione al livello di tensione presente sulla soglia e sul trigger; le combinazioni che determinano uno stato basso o uno stato alto sono riportate nella tabella: oltre a queste esiste una terza combinazione valida quando $v_2 > \frac{1}{3}Vcc$ e $v_6 < \frac{2}{3}Vcc$ corrispondente allo stato di memoria del flip flop per cui esso mantiene lo stato precedente fino alla reimpostazione di uno 0 o di un 1.

	Trigger v_2	*Soglia* v_6	*Uscita* v_3	*T1*
Trigger attivo	$< 1/3\ V_{CC}$	$< 2/3\ V_{CC}$	Alta (H)	OFF
Soglia attiva	$> 1/3\ V_{CC}$	$> 2/3\ V_{CC}$	Bassa (L)	ON

Per quanto riguarda il funzionamento come astabile lo schema e l'andamento delle tensioni sono riportate sotto, come si può vedere tramite il ramo di temporizzazione Ra Rb C si introduce un ritardo nella carica del condensatore C che è costretto a caricarsi e scaricarsi tra le due soglie 1/3 Vcc e 2/3 Vcc determinando il passaggio tra i tre stati del flip flop in modo da generare l'onda quadra in uscita.



Il periodo di scarica di C dipende in modo direttamente proporzionale dalla somma dei valori di Ra e Rb mentre il periodo di carica dipende soltanto dalla resistenza Rb la quale deve essere necessariamente maggiore o uguale a 1 kohm per evitare la distruzione del transistor interno all'integrato per eccessiva corrente.

Utilizzando questo schema quindi, il duty cicle non potrà mai essere minore del 50 per cento in quanto Rb dovrebbe valere 0 e ciò non è possibile per il motivo esposto prima, per ottenere tale condizione è sufficiente aggiungere un diodo e porre Ra=Rb in modo da cortocircuitare la resistenza Rb durante il periodo di carica tramite il diodo e eguagliare i due tempi ponendo le resistenze uguali; nello schema realizzato praticamente si sono aggiunti in serie alle resistenze dei trimmer in modo da far variare la frequenza per tararla sull'esatto valore necessario.

Di seguito sono riportate le formule utilizzate per il calcolo dei componenti

$$T1 = RbCln(\frac{0-\frac{2}{3}Vcc}{0-\frac{1}{3}Vcc}) = 0{,}693RbC \quad [2.0]$$

$$T2 = RaCln(\frac{Vcc-\frac{1}{3}Vcc}{Vcc-\frac{2}{3}Vcc}) = 0{,}693RaC \quad [2.1]$$

Si è scelto prima di tutto il valore della capacità C di 1 nF (un valore non troppo basso per evitare errori dovuti alle capacità parassite del 555), dopodiché tramite le formule inverse riportate sotto si è calcolato la resistenza minima e massima per avere rispettivamente il periodo massimo e minimo.

$$T = \frac{1}{f} \quad [2.2]$$

$$T1 = T2 = \frac{T}{2} = \frac{1}{2f} \quad [2.3]$$

$$fmin = 10\ Khz = 10 * 10^3\ Hz \quad [2.4]$$

$$fmax = 60\ Khz = 60 * 10^3 Hz \quad [2.5]$$

$$\frac{Tmin}{2} = \frac{1}{2fmax} = \frac{1}{2*60*10^3} = 8{,}3 * 10^{-6}s = 8{,}3\ \mu s \quad [2.6]$$

$$\frac{Tmax}{2} = \frac{1}{2fmin} = \frac{1}{2*10*10^3} = 5 * 10^{-5}s = 50\ \mu s \quad [2.7]$$

$$C = 1\ \eta F \quad [2.8]$$

$$R1, R3 = \frac{\frac{Tmin}{2}}{0{,}693C} = \frac{8{,}3*10^{-6}}{0{,}693*1*10^{-9}} = 11{,}98 * 10^3\Omega \cong 12\ K\Omega \quad [2.9]$$

$$R2, R4 = \frac{\frac{Tmax}{2}-\frac{Tmin}{2}}{0{,}693C} = \frac{(5*10^{-5}-8{,}3*10^{-6})}{0{,}693*10^{-9}} = 60{,}2 * 10^3\Omega \cong 60\ K\Omega \quad [2.10]$$

Per R_2 e R_4 si è utilizzato un trimmer da 100 Kohm in modo da esser sicuri di raggiungere la frequenza più bassa e spingersi ancora più in basso per evitare problemi legati ad eventuali capacità - resistenze parassite e tolleranze dei componenti.

I led sono connessi direttamente all'uscita del 555 tramite le rispettive resistenze di limitazione di corrente: esse sono calcolate per far scorrere una corrente di 100 mA nel led del sensore di prossimità (necessita di una corrente maggiore in quanto si rileva solo la luce riflessa) e 20 mA negli altri LED utilizzati per le barriere infrarosse (il 555 riesce da solo senza bisogno di circuiti buffer ad alimentare tutti i led essendo capace di erogare 200 mA max).

I ricevitori

Per quanto riguarda la parte di ricezione ovvero il trattamento del segnale proveniente dai fotodiodi si sono trovate due soluzioni differenti per il sensore di prossimità e per le barriere infrarosse, di seguito è descritto in modo dettagliato il funzionamento dei due tipi di ricevitore.

Ricevitori per barriera infrarossi con tone decoder NE567

Per realizzare il ricevitore per la foto barriera si è utilizzato un circuito integrato che basa il suo principio di funzionamento sul PLL: l'NE567.

Il PLL (Phased Locked Loop) ovvero anello ad aggancio di fase è un dispositivo elettronico che permette di sincronizzare la sua uscita con un segnale comunque variabile in ingresso mantenendo una certa relazione di fase con esso e la stessa frequenza o in certi casi multipla o sottomultipla.

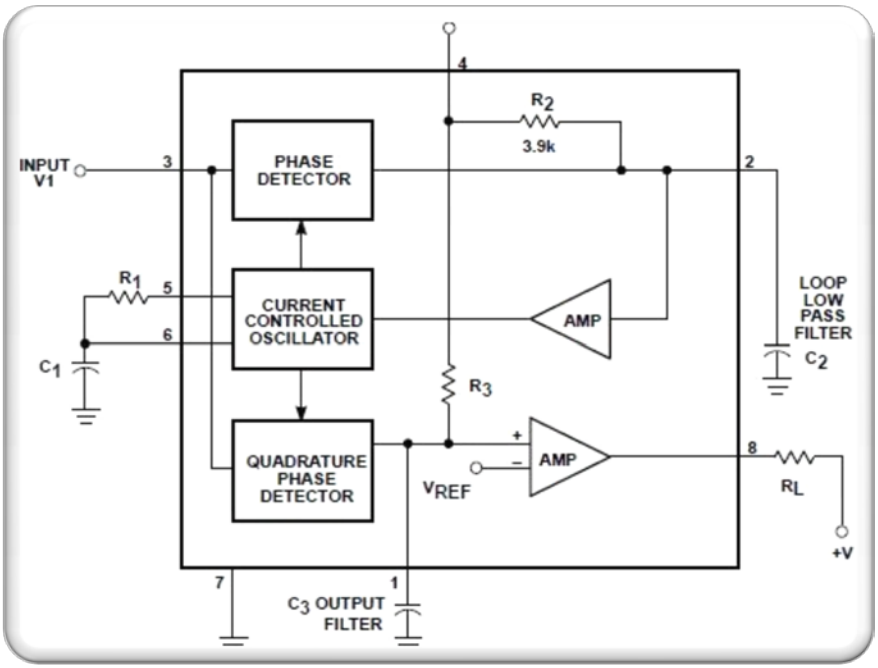
Esso è costituito da tre blocchi fondamentali, il primo, un comparatore di fase che fornisce in uscita un'onda quadra con duty cicle proporzionale alla differenza di fase; sarà minimo se le due fasi sono le stesse e massimo in corrispondenza del massimo scostamento.

Al primo blocco segue un filtro passa basso che ha il compito di estrarre la componente continua del primo blocco determinando una tensione di errore direttamente proporzionale alla differenza di fase tra i due segnali applicati in ingresso.

L'ultimo blocco, un VCO o CCO (Voltage/Current Controlled Oscillator) è un oscillatore controllato in tensione libero di oscillare in un range di frequenze compreso tra una minima e una massima.

Esso riceve sul piedino di controllo la tensione di errore uscente dal secondo blocco e la sua uscita è riportata in ingresso al comparatore di fase che confronta l'uscita con il segnale da agganciare: inizialmente il PLL oscilla ad una frequenza propria definita da alcuni elementi del circuito (resistenze e condensatori) e in questo momento la differenza di fase tra l'onda da lui generata e il segnale da agganciare da origine a un segnale errore che sommandosi alla tensione di riposo presente sul pin di controllo va a modificare la frequenza del segnale generato dal VCO fintanto che questa non diventa uguale a quella del segnale di ingresso e sfasata rispetto ad esso di 90 gradi.

Schema interno NE567



È proprio sulla seconda condizione di aggancio che basa il suo funzionamento il tone decoder, in aggiunta al PLL ha infatti al suo interno il quadrature phase detector che rileva l'avvenuto aggancio, in questo caso la frequenza dell'oscillatore locale (il VCO; per l'NE567 CCO Current Controlled Oscillator) rimane costante e ciò che può variare è il segnale di ingresso: soltanto quando questo ha la stessa frequenza di quella generata internamente dal CCO si troverà sul piedino 8 uno zero logico altrimenti un 1; si avrà quindi un livello logico basso quando le fotocellule non sono interrotte e viceversa in condizione di interruzione del fascio.

Di seguito sono riportate le formule utilizzate per il calcolo dei componenti

$$T = 1{,}1RC \qquad [2.11]$$

Si è scelto un valore per C di 1 nF

$$C = 1\eta F \qquad [2.12]$$

$$fmin = 10\ Khz \qquad [2.13]$$

$$fmax = 60\ Khz \qquad [2.14]$$

$$Tmin = \frac{1}{fmax} = \frac{1}{60*10^3} = 1{,}6 * 10^{-5}s = 16\ \mu s \qquad [2.15]$$

$$Tmin = \frac{1}{fmin} = \frac{1}{10*10^3} = 1 * 10^{-4}s = 100\ \mu s \qquad [2.16]$$

$$Rmin = \frac{Tmin}{1{,}1C} = \frac{1{,}6*10^{-5}}{1{,}1*1*10^{-9}} = 14{,}5 * 10^3\ \Omega \cong 15\ \mathrm{K}\Omega \qquad [2.17]$$

$$Rmax = \frac{Tmax}{1{,}1C} = \frac{1*10^{-4}}{1{,}1*1*10^{-9}} = 90{,}9 * 10^3\ \Omega \cong 100\ \mathrm{K}\Omega \qquad [2.18]$$

Il segnale proveniente dai fotodiodi non è applicato direttamente all'ingresso del tone decoder ma tramite un condensatore di disaccoppiamento che permette alla sola componente alternata di arrivare all'integrato bloccando invece la continua; per quanto riguarda invece gli altri condensatori presenti nel circuito essi fanno parte di due filtri: uno è il passa basso del PLL mentre l'altro il filtro in uscita al phase quadrature detector.

È stato fondamentale per il funzionamento corretto dei tone decoder aggiungere dei condensatori tra il piedino di alimentazione e massa vicino agli integrati in modo da eliminare i disturbi presenti sulle linee di alimentazione; esse infatti erano responsabili di malfunzionamenti, il tone decoder non era in grado di fornire uno stato determinato quando il fascio era interrotto e l'uscita oscillava continuamente passando da stato alto a stato basso; i condensatori aggiunti successivamente sono visibili sulla scheda e hanno un valore di 470 uF.

Ricevitori per sensore di prossimità a operazionali

Il ricevitore realizzato per il sensore di prossimità funziona in modo differente rispetto a quello a tone decoder; esso utilizza dei circuiti a operazionale e ha un funzionamento più complesso.

Il circuito può essere diviso in blocchi:

- Blocco 1 - Buffer di corrente
- Blocco 2 – Filtro passa banda
- Blocco 3 – Rivelatore di inviluppo
- Blocco 4 – Filtro passa basso
- Blocco 5 – Comparatore di tensione
- Blocco 6 – Adattatore di livelli

Il fotodiodo fornisce un segnale alternato di ampiezza proporzionale alla distanza del pezzo da esso e di frequenza pari a quella impostata dal trasmettitore, il circuito ricevitore amplifica in corrente il segnale tramite il buffer, seleziona poi soltanto la frequenza di trasmissione rendendo immune il sistema a fonti di luce esterne tramite il filtro passa banda e infine con un rivelatore di inviluppo e un successivo filtro passa basso fornisce una tensione continua direttamente proporzionale alla distanza del pezzo dal sensore.

Questa tensione viene poi applicata a un comparatore dove viene confrontata con una tensione di riferimento fissa impostabile con un trimmer, questa tensione determina a che distanza si ha la commutazione dell'uscita e quindi il cambiamento dello stato logico, l'uscita viene poi adattata in tensione, per renderla compatibile con il microcontrollore, utilizzando un diodo zener a 5 volt opportunamente polarizzato; il risultato è che in uscita si ha uno zero logico in presenza di un pezzo di fronte al sensore e viceversa un uno.

Descrizione dettagliata dei blocchi del ricevitore

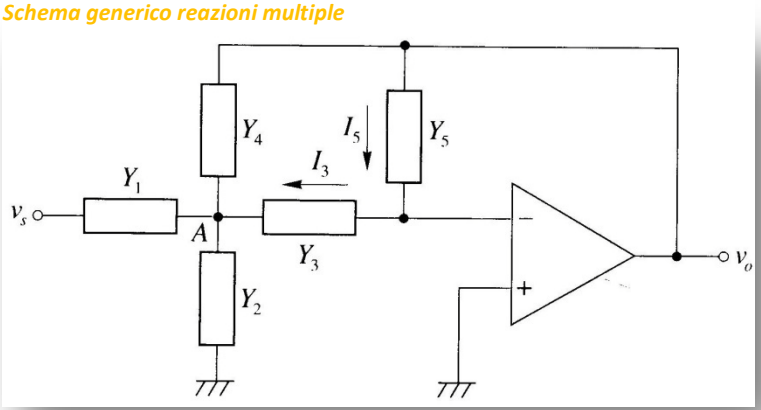
Blocco 1

Il primo blocco è un circuito buffer realizzato con un operazionale, ha un guadagno unitario e grazie ad esso è possibile amplificare in corrente il segnale proveniente dal fototransistor, l'impedenza di ingresso è infatti molto alta e idealmente infinita mentre quella di uscita zero, per questo motivo non si ha assorbimento di corrente in ingresso mentre in uscita il circuito è in grado di erogare una corrente anche di qualche centinaio di mA; in questo modo è possibile riportare la tensione presente sul fotodiodo all'ingresso del filtro passa banda senza introdurre attenuazioni.

Blocco 2

Per il secondo blocco si è scelto di utilizzare un filtro attivo passa banda a reazioni multiple del secondo ordine. Grazie ad esso si è ottenuto un filtro con un buon fattore di qualità e si è riusciti a introdurre anche un'amplificazione sul segnale proveniente dal fotodiodo, esso infatti ha un'ampiezza dell'ordine del mV e per renderlo meglio utilizzabile si è preferito

Schema generico reazioni multiple



introdurre un guadagno direttamente tramite il filtro anziché utilizzare un circuito amplificatore dedicato; in questo modo si è sicuri di amplificare soltanto la frequenza desiderata, nel caso in cui si fosse amplificato prima di inviarlo al filtro tutta la banda passante dell'amplificatore sarebbe stata amplificata e ci sarebbe stato un aumento della rumorosità.

I filtri a reazione multipla impiegano un amplificatore operazionale in configurazione invertente con due anelli di reazione, se si scrive l'equazione al nodo A e si ritiene ideale l'operazionale si ricava la funzione di trasferimento caratteristica dei filtri a reazione multipla riportata di seguito.

$$A(s) = \frac{Vo}{Vs} = \frac{-Y1Y3}{Y5(Y1+Y2+Y3+Y4)+Y3Y4} \qquad [2.19]$$

Nel caso del filtro passa banda realizzato per il progetto la funzione di trasferimento assume questa forma:

$$A(s) = \frac{-\frac{s}{R11C5}}{s^2+\frac{1}{R13}\left(\frac{1}{C5}+\frac{1}{C4}\right)s+\frac{1}{R13C5C4}\left(\frac{1}{R11}+\frac{1}{R12}\right)} \qquad [2.20]$$

Confrontando questa equazione con la funzione di trasferimento generalizzata per i filtri passa banda $G(s) = \frac{\frac{\varpi o}{Qo}Aos}{s^2+\frac{\varpi o}{Qo}s+\varpi o^2}$ [2.21] si ottengono, per il principio di identità dei polinomi, le relazioni riportate sotto.

$$\frac{\varpi o}{Qo}Ao = -\frac{1}{R11C5} \qquad [2.22]$$

$$\frac{\varpi o}{Qo} = \frac{1}{R13}\left(\frac{1}{C5}+\frac{1}{C4}\right) \qquad [2.23]$$

$$\varpi o^2 = \frac{1}{R13C5C4}\left(\frac{1}{R11}+\frac{1}{R12}\right) \qquad [2.24]$$

La pulsazione di risonanza è pari a:

$$\varpi o = \sqrt{\frac{1}{R13C5C4}\left(\frac{1}{R11}+\frac{1}{R12}\right)} \qquad [2.25]$$

Da cui la frequenza di risonanza:

$$fo = \frac{\varpi o}{2\pi} \qquad [2.26]$$

Per quanto riguarda il guadagno questo è invece pari a:

$$Av = -\frac{R13}{R11} * \left(\frac{1}{1+\frac{C5}{C4}}\right) \qquad [2.27]$$

Per il corretto funzionamento del circuito è necessario rispettare: $Q^2\left(1+\frac{C5}{C4}\right) \geq |Ao|$, per cui utilizzando C_4=C_5 si semplificano i calcoli e risulta $Q \geq \frac{|Ao|}{2}$.

I valori dei componenti calcolati per il progetto sono per una frequenza di risonanza di 11Khz e un amplificazione di 10 unità; si è scelto questa frequenza non troppo alta data la relativa lentezza nella risposta dei fototransistor in modo da avere un segnale su di esso di ampiezza maggiore e di forma più vicina all'onda quadra invece che a quella triangolare.

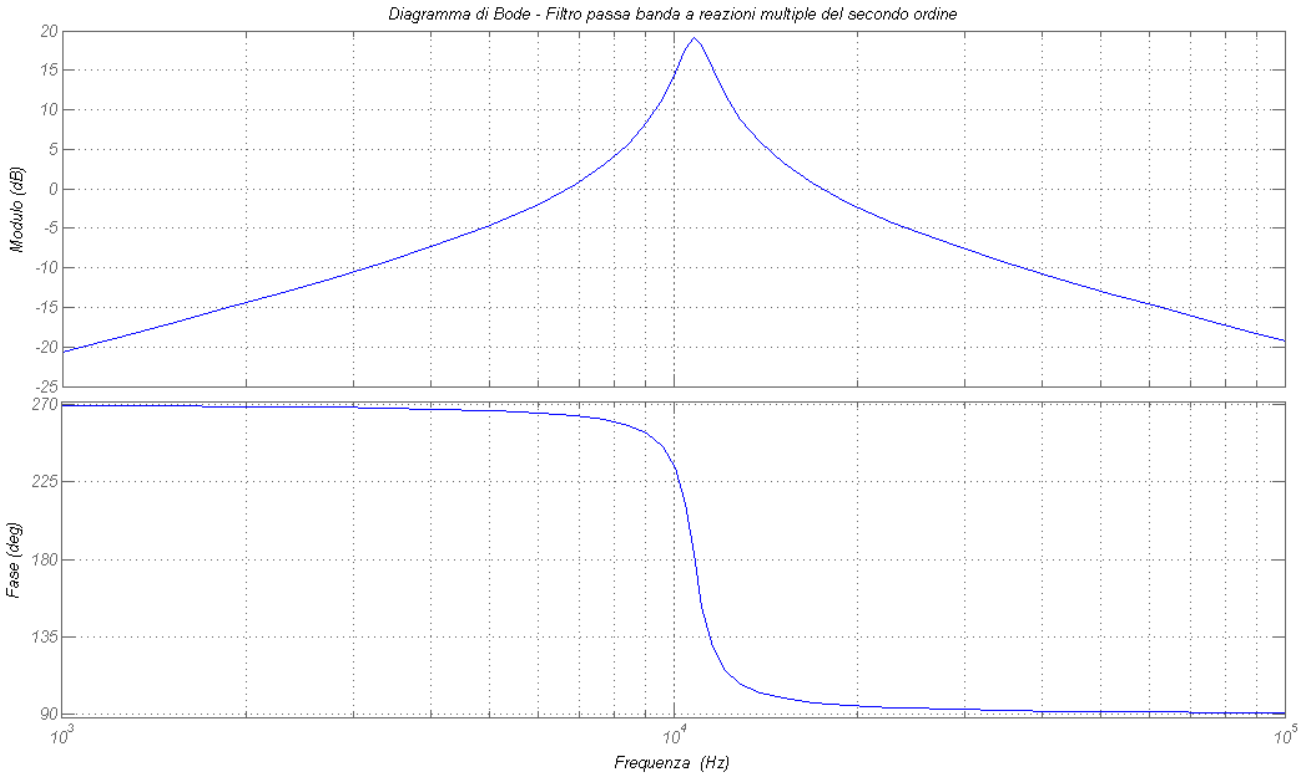
Il valore ottenuto realmente per la frequenza di risonanza e per l'amplificazione differisce leggermente a causa delle tolleranze dei componenti e la frequenza di risonanza effettiva è di 11,39 Khz mentre l'amplificazione leggermente minore di 10.

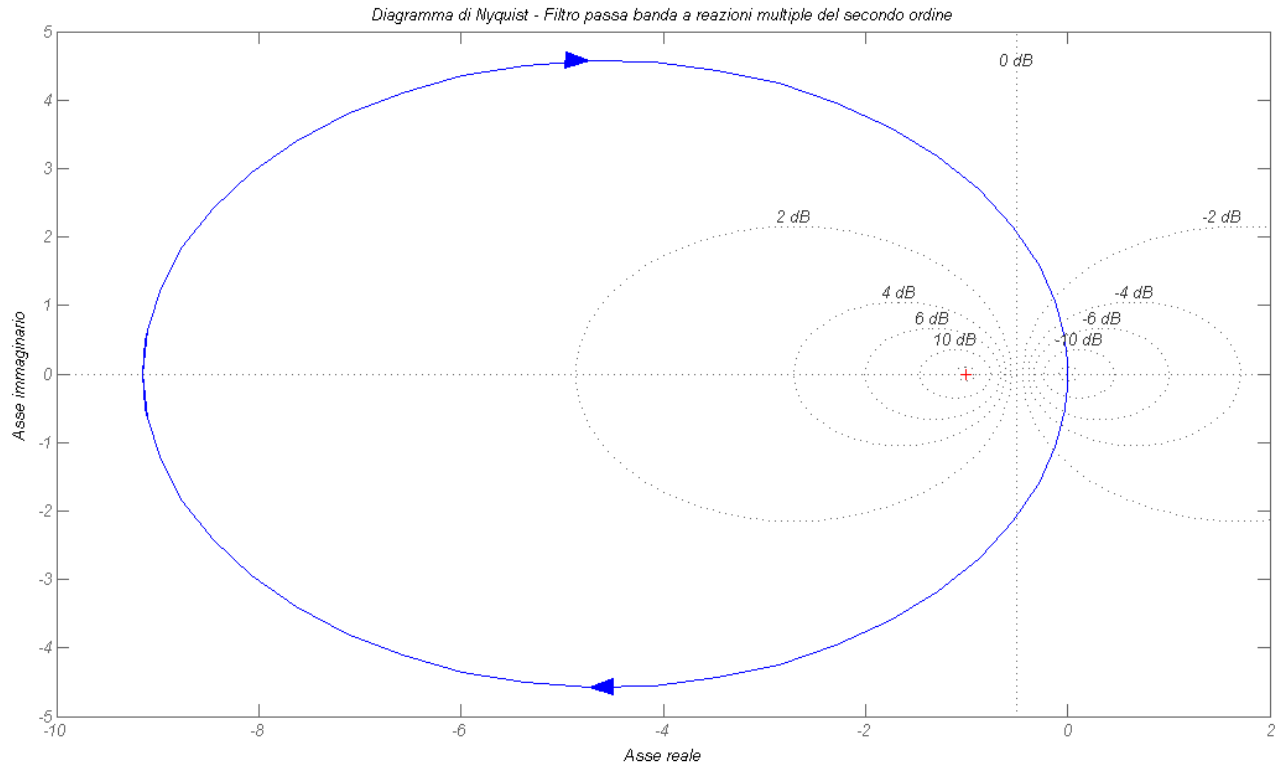
I calcoli per ricavare i valori dei componenti del progetto non sono riportati in quanto sono stati effettuati con un software online di simulazione, tuttavia possono essere calcolati anche manualmente utilizzando le formule inverse derivate da quelle viste sopra.

Caratteristiche del filtro passa banda secondo il calcolo iniziale

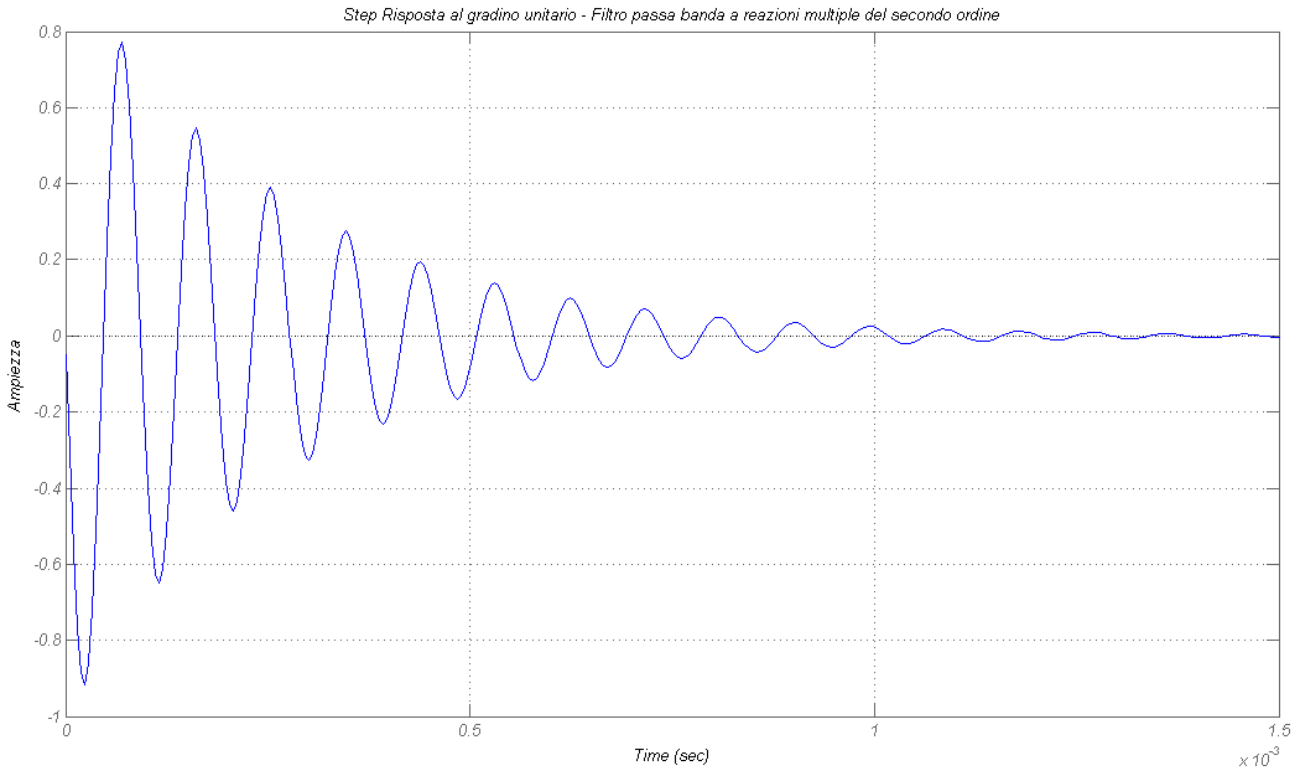
Sostituendo i valori reali dei componenti all'interno dell'equazione [2.20] si ottiene la funzione di trasferimento del filtro utilizzato nel progetto ed è possibile ricavare i diagrammi di Bode e di Nyquist; essi sono riportati sotto e sono stati disegnati con il software di simulazione MATLAB.

$$A(s) = \frac{-67{,}75*10^3 s}{s^2+7407s+1{,}6*10^9} \qquad [2.28]$$





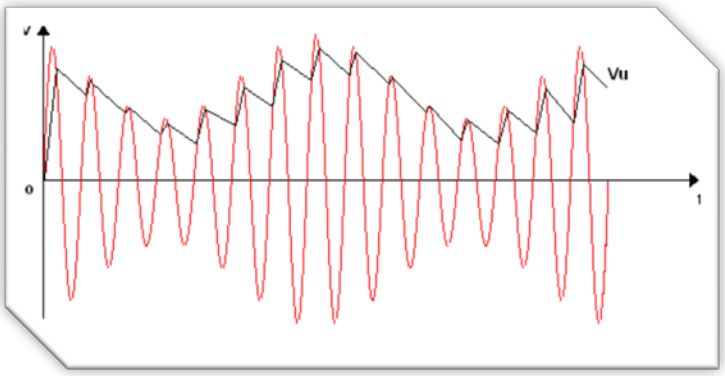
Di seguito è riportato anche il grafico della risposta al gradino del sistema.



Blocco 3

Il blocco successivo al filtro passa banda è un rivelatore di inviluppo simile a quelli utilizzati per la rivelazione in AM, esso è costituito da un diodo (D_2) e una cella RC (C_7, R_{16}) parallelo; il condensatore si carica tramite il diodo al valore di picco dell'onda in ingresso e si scarica attraverso R_{16} in un tempo maggiore del periodo di modo tale da mantenere il valore di picco e ottenere in uscita dal blocco una tensione pressoché continua che segue l'andamento dell'inviluppo (con dei residui alla frequenza con cui si pilotano i sensori).

Vo – rivelatore di inviluppo



Il valore di tensione è direttamente proporzionale all'ampiezza dell'onda uscente dal filtro passa banda e di conseguenza anche all'intensità della radiazione incidente sul fotodiodo del sensore di prossimità e alla distanza del pezzo dal sensore: minore è la distanza, maggiore sarà la luce riflessa e la tensione continua in uscita dal rivelatore e viceversa; l'uscita può variare tra $\frac{1}{2}Vcc$ e circa Vcc; il valore minimo è $\frac{1}{2}Vcc$ dato che il filtro passa banda è polarizzato per il funzionamento in continua e in uscita si ritrova sempre una componente continua a $\frac{1}{2}Vcc$.

Costante di tempo del rivelatore

$$\tau = RC = R16 * C7 = 100 * 10^3 * 1 * 10^{-6} = 0{,}1\ s \qquad [2.29]$$

Come si può vedere la costante di tempo τ è maggiore del periodo del segnale ($T = 8{,}78\ \mu s$) per cui il rivelatore di inviluppo è insensibile a eventuali disturbi in alta frequenza e per poter avere una variazione della tensione continua in uscita essa deve presentarsi in ingresso e durare per numerosi periodi dell'onda.

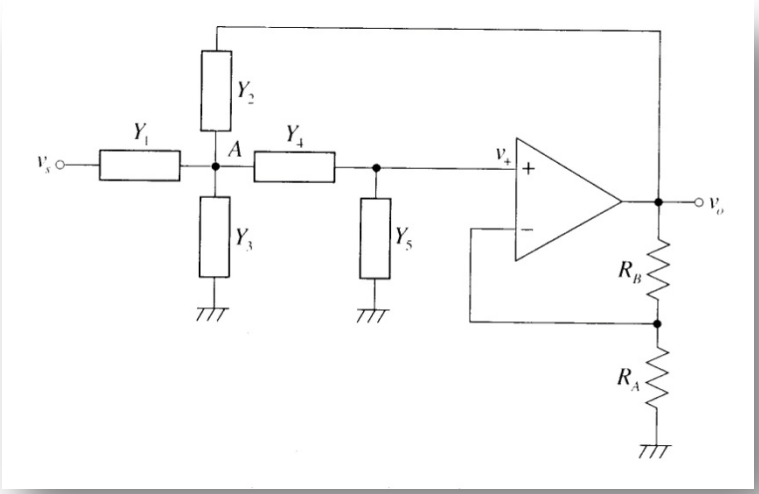
Blocco 4

Prima di poter inviare il segnale uscente dal blocco 3 a un comparatore, per determinare se il pezzo è effettivamente presente, esso va filtrato definitivamente per eliminare gli ultimi residui ad alta frequenza ottenendo così in uscita una tensione continua con bassissimo ripple; questo passaggio è necessario per evitare false commutazioni del circuito comparatore che potrebbe cambiare stato in uscita in seguito a variazioni della tensione da confrontare dovute al ripple presente su di essa.

A tale scopo si è utilizzato un filtro passa basso del secondo ordine VCVS realizzato con un amplificatore operazionale, con questa soluzione si è ottenuto un circuito in grado di eliminare efficacemente i residui in alternata, l'operazionale funziona da buffer in quanto l'amplificazione è pari a 1 ovvero la tensione in entrata viene riportata (esclusa la parte filtrata) in uscita senza introdurre attenuazioni.

I filtri del tipo a reazione positiva semplice o VCVS (voltage – controller voltage source) impiegano un amplificatore operazionale in configurazione non invertente con un anello di reazione negativo e uno positivo composti rispettivamente da due resistenze per determinare il fattore K (il guadagno nella banda passante) e varie celle filtranti con una propria ammettenza visibili nello schema generale del filtro.

Schema generico filtri VCVS



Se si pone $K = \frac{Vo}{V+} = 1 + \frac{Rb}{Ra}$ [2.30] e si scrive l'equazione al nodo A si ricava la funzione di trasferimento caratteristica dei filtri VCVS riportata di seguito.

$$A(s) = \frac{Vo}{Vs} = \frac{KY1Y4}{Y5(Y1+Y2+Y3+Y4)+Y4[Y1+Y2(1-K)+Y3]} \qquad [2.31]$$

Nel caso del filtro passa basso utilizzato per il progetto la funzione di trasferimento assume questa forma:

$$A(s) = \frac{K\left(\frac{1}{R17R18C9C8}\right)}{s^2+s\left(\frac{1}{R17C9}+\frac{1}{R18C8}+\frac{1-K}{R18C8}\right)+\frac{1}{R17R18C9C8}} \qquad [2.32]$$

Confrontando questa equazione con la funzione di trasferimento generalizzata per i filtri passa basso $G(s) = \frac{\varpi o^2 Ao}{s^2+\frac{\varpi o}{Qo}s+\varpi o^2}$ [2.33] si ottengono, per il principio di identità dei polinomi, le relazioni riportate sotto.

$$\varpi o^2 Ao = K\left(\frac{1}{R17R18C9C8}\right) \qquad [2.34]$$

$$\frac{\varpi o}{Qo} = \left(\frac{1}{R17C9}+\frac{1}{R18C8}+\frac{1-K}{R18C8}\right) \qquad [2.35]$$

$$\varpi o^2 = \frac{1}{R17R18C9C8} \qquad [2.36]$$

La pulsazione di taglio è pari a:

$$\varpi o = \frac{1}{\sqrt{R17R18C9C8}} \qquad [2.37]$$

Da cui la frequenza di taglio:

$$fo = \frac{\varpi o}{2\pi} \qquad [2.38]$$

Per quanto riguarda il guadagno questo è invece pari a:

$$Av = K = 1 + \frac{Rb}{Ra} \qquad [2.39]$$

I valori dei componenti calcolati per il progetto sono per una frequenza di taglio di 10 Hz e un amplificazione di 1 (l'operazionale è collegato come buffer $Rb = 0, Ra = \infty$); si è scelto questa frequenza così bassa in modo da attenuare fortemente i residui alla "frequenza portante" di 11,39 Khz che essendo superiore di 3 decadi a quella di taglio viene attenuata di 120 dB dato che il filtro è del secondo ordine e introduce un'attenuazione fuori banda di – 40 Db/decade.

Dalla componente continua fino a 10 Hz si rientra nella banda passante del filtro per cui la componente continua può tranquillamente arrivare al circuito comparatore senza subire attenuazioni.

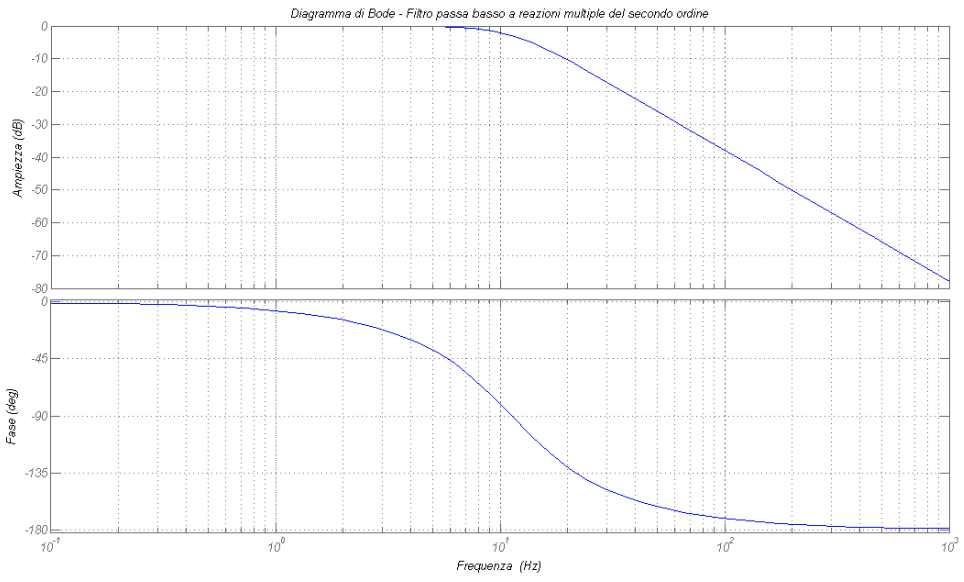
Il valore ottenuto realmente per la frequenza di taglio differisce leggermente a causa delle tolleranze dei componenti e la frequenza di taglio effettiva è di circa 6 Hz.

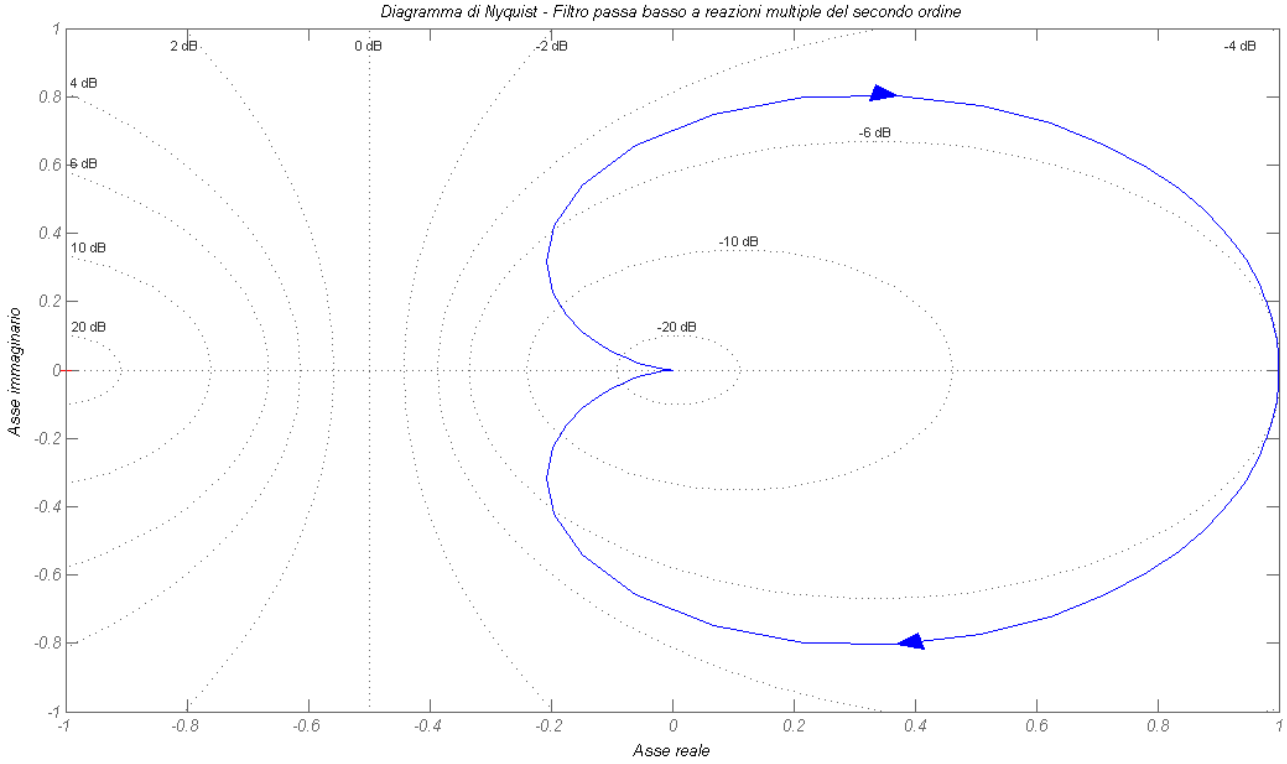
I calcoli per ricavare i valori dei componenti del progetto non sono riportati in quanto sono stati fatti con un software online di simulazione, tuttavia possono essere calcolati anche manualmente utilizzando le formule inverse derivate da quelle viste sopra.

Caratteristiche del filtro passa basso

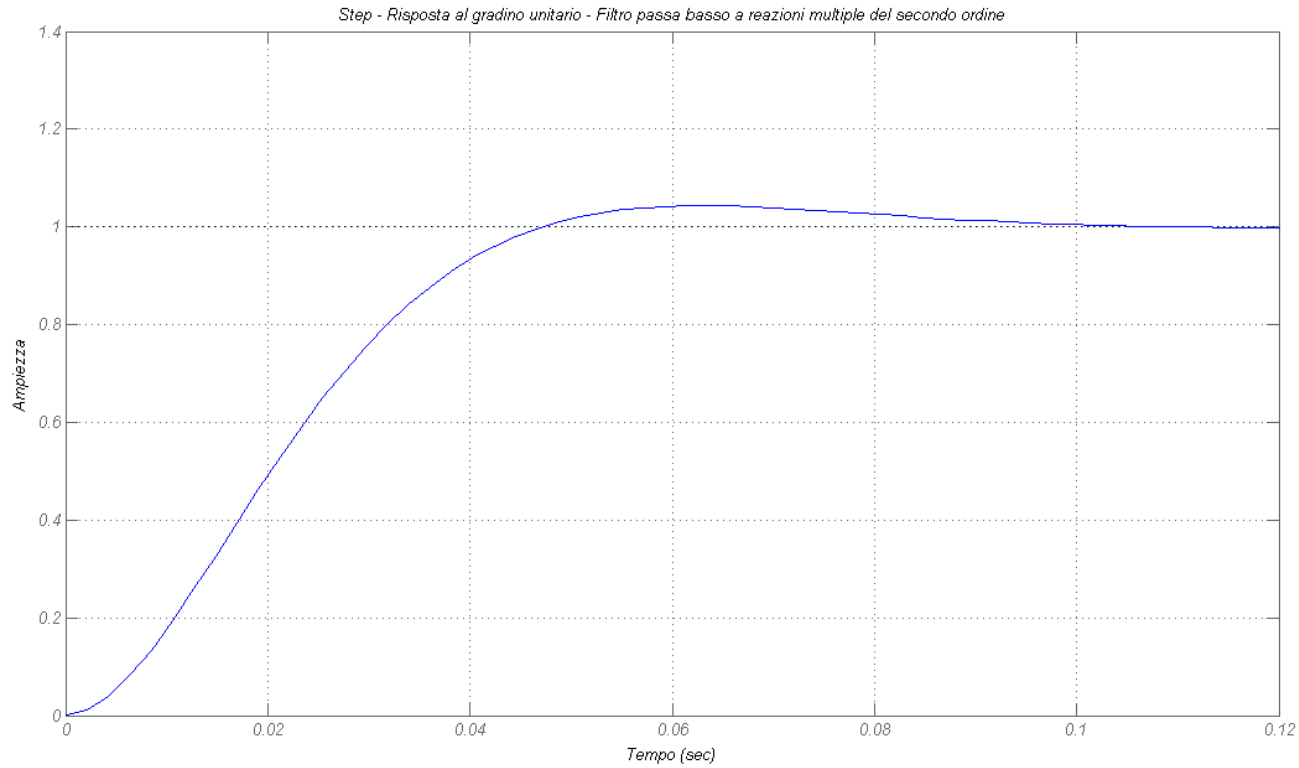
Sostituendo i valori reali dei componenti all'interno dell'equazione [2.32] si ottiene la funzione di trasferimento del filtro utilizzato nel progetto ed è possibile ricavare i diagrammi di Bode e di Nyquist; essi sono riportati sotto e sono stati disegnati con il software di simulazione MATLAB.

$$A(s) = \frac{5000}{s^2+100s+5000} \qquad [2.40]$$





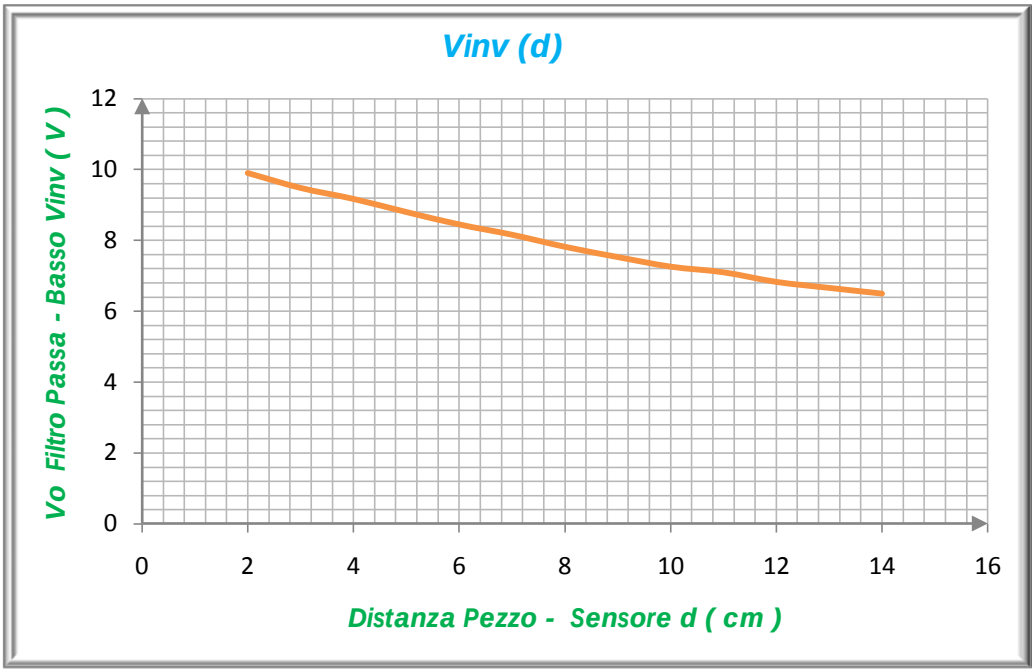
Di seguito è riportato anche il grafico della risposta al gradino del sistema



Blocco 5

Il penultimo blocco del circuito dedicato al sensore di prossimità è un comparatore; esso riceve sull'ingresso invertente la tensione uscente dal filtro passa basso mentre su quello non invertente è presente una tensione di soglia stabilizzata dal condensatore C_{10} e regolabile agendo sul trimmer R_{20} da 6 a 12 V, così facendo l'uscita del comparatore è normalmente a livello alto se non è presente un pezzo davanti al sensore mentre se il pezzo è abbastanza vicino da far salire la tensione sul morsetto invertente oltre quella di soglia l'uscita cambia stato andando bassa e segnalando al microcontrollore la presenza di un pezzo; grazie al trimmer è possibile regolare la soglia determinando a quale distanza il pezzo viene considerato nella giusta posizione e rilevato (il circuito è tarato per rilevare la presenza di un pezzo a circa 10 cm o meno di distanza).

La tensione che arriva sul morsetto invertente è il risultato dei passaggi dai blocchi precedenti e per essa si può determinare una caratteristica di trasferimento che la mette in relazione con la distanza del pezzo dal sensore ovvero, Vinv (tensione sul morsetto invertente del comparatore) in funzione di d (distanza del pezzo dal sensore), di seguito è riportato il grafico dal quale emerge una caratteristica pressoché lineare nell'intervallo di distanze 2 – 14 cm dal sensore utilizzando come riferimento un pezzo bianco (massima riflessione).



Blocco 6

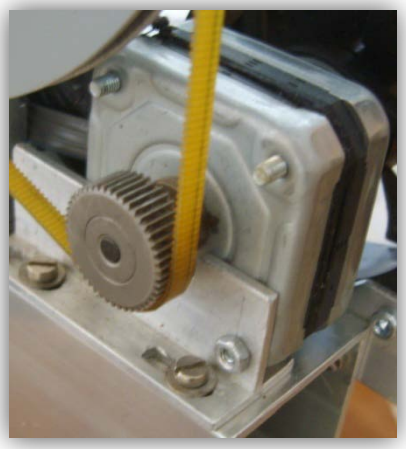
L'ultimo blocco posto in uscita al comparatore è un adattatore di livelli realizzato con una resistenza di limitazione (R_{21}) e un diodo zener a 5 volt, esso ha la funzione di adattare il livello alto uscente dal comparatore ai livelli TTL accettati in ingresso dal microcontrollore, dagli 0 - 12 Volt in entrata si passa agli 0 - 5 V in uscita; la corrente che scorre nello zener, quando svolge la sua funzione limitando a 5 V la tensione in uscita, è di circa 7 mA.

- Schede controllo motori

Le schede di controllo di potenza dei motori rivestono il ruolo di maggiore importanza per quanto riguarda le uscite del sistema ovvero tutte quelle parti che possono essere viste come periferiche di uscita della scheda a microcontrollore a cui sono connesse tramite delle interfacce e permettono ad essa di pilotare le parti di potenza come i motori indirettamente.
Per il progetto sono stati utilizzati due tipi di schede: una per il controllo dei motori passo passo e una per il controllo del motore in corrente continua della pinza; prima di iniziare a descrivere i circuiti veri e propri è riportata una parte dedicata ai tipi di motore utilizzati.

- **I motori passo passo**

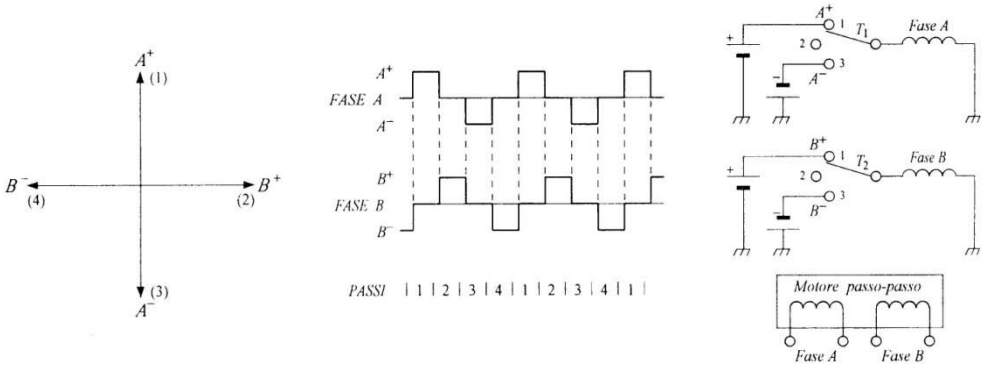
Questo tipo di motore è stato utilizzato per la movimentazione della base e delle leve per lo spostamento verso l'alto e verso il basso; esistono vari tipi di motori passo passo e la distinzione principale può essere fatta tra quelli bipolari e unipolari; per il progetto si è utilizzato il primo tipo ovvero bipolare (a due fasi).
Si è scelto di utilizzare motori passo passo in quanto presentano la particolarità di ruotare per passi successivi con spostamenti precisi e definiti per ogni passo per cui è possibile in ogni momento conoscere la posizione dell'asse contando i passi compiuti nei due versi senza utilizzare alcun tipo di sensore (encoder o potenziometri sugli assi come per i motori in continua).



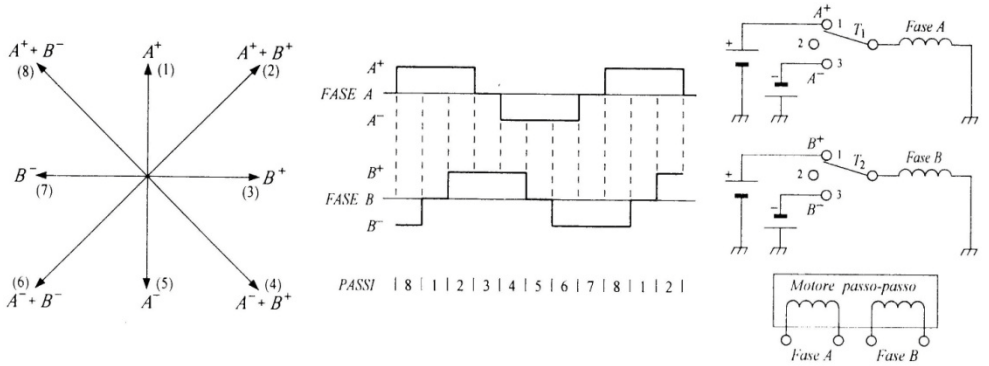
Il tipo bipolare presenta all'esterno 4 connessioni corrispondenti a 2 a 2 ai capi degli avvolgimenti interni del motore (2 avvolgimenti), agendo opportunamente con segnali ad onda quadra sui 2 avvolgimenti si possono far compiere al motorino dei passi in un determinato senso e si può scegliere anche con che metodo pilotarli ovvero a mezzo passo o a passo intero.
Ogni motore ha un numero intrinseco di passi per ruotare di 360 gradi ed è questa una delle caratteristiche che viene fornita dal costruttore assieme al grafico della coppia in funzione del numero di giri; i motori utilizzati per il progetto, essendo stati recuperati non erano completi della documentazione e sia per i passi sia per la tensione corretta di alimentazione si sono effettuate delle prove preliminari su di essi per determinarne rispettivamente il numero e il valore appropriato per avere una buona coppia e un riscaldamento ridotto, con un alimentazione a 12 V e un sistema di

raffreddamento a ventola la temperatura raggiunta dai motori si aggira intorno ai 30 – 40 gradi).

Nel metodo di pilotaggio a passo intero la minima rotazione che si può far compiere al motore è di un passo mentre nell'altra modalità mezzo; di seguito sono riportate le sequenze per il pilotaggio in entrambe le modalità: 1 - intero, 2 - mezzo.



Passi	T_1	T_2	A^+	A^-	B^+	B^-
1	1	2	ON	OFF	OFF	OFF
2	2	1	OFF	OFF	ON	OFF
3	3	2	OFF	ON	OFF	OFF
4	2	3	OFF	OFF	OFF	ON



Passi	T_1	T_2	A^+	A^-	B^+	B^-
1	1	2	ON	OFF	OFF	OFF
2	1	1	ON	ON	OFF	OFF
3	1	2	OFF	OFF	ON	OFF
4	3	1	OFF	ON	ON	OFF
5	3	2	OFF	ON	OFF	OFF
6	3	3	OFF	ON	OFF	ON
7	2	3	OFF	OFF	OFF	ON
8	2	3	ON	OFF	OFF	ON

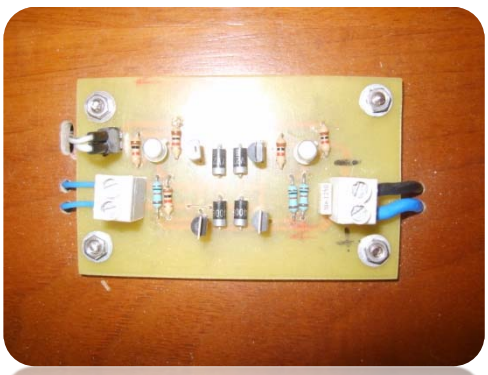
- **Il motore DC della pinza**

Per la pinza si è scelto di utilizzare un motore DC ovvero in continua: questo tipo di motore è adatto a questo utilizzo in quanto ha una buona coppia e riesce a garantire assieme alle riduzioni meccaniche una buona stretta da parte della pinza.

Questo motore è molto più semplice da pilotare rispetto ad uno di tipo passo passo ma non si hanno riferimenti sulla sua posizione in base a come viene pilotato come avveniva invece per i motori step.
La rotazione avviene quando si applica una tensione sui due terminali del motore, dalla polarità dipende il verso di rotazione mentre la velocità è direttamente proporzionale alla tensione con cui si alimenta; in questo caso la velocità è costante e per rilevare la posizione della pinza si effettua un conteggio via software che calcola il tempo impiegato dalla pinza per chiudersi per poi riaprirsi per il medesimo tempo e tornare al punto di partenza.
Il tipo utilizzato per il progetto è alimentato a 12 volt e ha un assorbimento di circa 30 mA, inoltre è dotato internamente di riduttori meccanici a ingranaggi per aumentare la coppia.

- Scheda controllo motore DC

La scheda per il controllo del motore DC è un ponte ad H realizzato a transistor, si è scelto di utilizzare una soluzione a componenti discreti dato che i circuiti integrati come l'L298 sono in grado di pilotare motori con assorbimenti fino a 2 ampere mentre per il progetto bastano 50 mA facilmente trattabili con transistor di media potenza relativamente piccoli.



Il funzionamento del circuito è il seguente: i transistor Q_2 – Q_6 che costituiscono il ponte possono essere portati in saturazione soltanto due alla volta ovvero Q_2 e Q_6 o Q_3 e Q_5; attivando una delle due possibili coppie la corrente è forzata a scorrere in un certo verso all'interno dell'avvolgimento del motore (connesso sui collettori dei transistor) o in quello opposto determinando la rotazione destrorsa o sinistrorsa.
Il verso di rotazione può essere selezionato azionando una sola delle due coppie di transistor, la parte restante dei componenti serve a rendere in grado il

microcontrollore a pilotare il circuito in logica positiva ed evitare problemi legati al picco di lenz per i transistor; queste due funzioni sono svolte rispettivamente dai transistor Q_1 e Q_4 che funzionano da porta NOT e dai diodi veloci di ricircolo $D_1 – D_4$.
Le porte NOT realizzate con i transistor portano in saturazione i transistor PNP della parte superiore del ponte quando sui loro ingressi è presente un uno mentre in presenza di uno zero Q_2 e Q_3 rimangono interdetti.
I segnali di controllo non sono quattro come ci si potrebbe aspettare cioè uno per ogni base ma uno per ogni coppia e quindi 2 e la coppia viene attivata da un livello logico alto mentre se è basso i transistor rimangono interdetti.
L'azione svolta dal motore in relazione agli stati degli ingressi è riportata nella seguente tabella.

Clock_MC	Verso_MC	Condizione Motore
0	0	Fermo
0	1	Ruota sinistrorso
1	0	Ruota destrorso
1	1	Condizione non ammessa

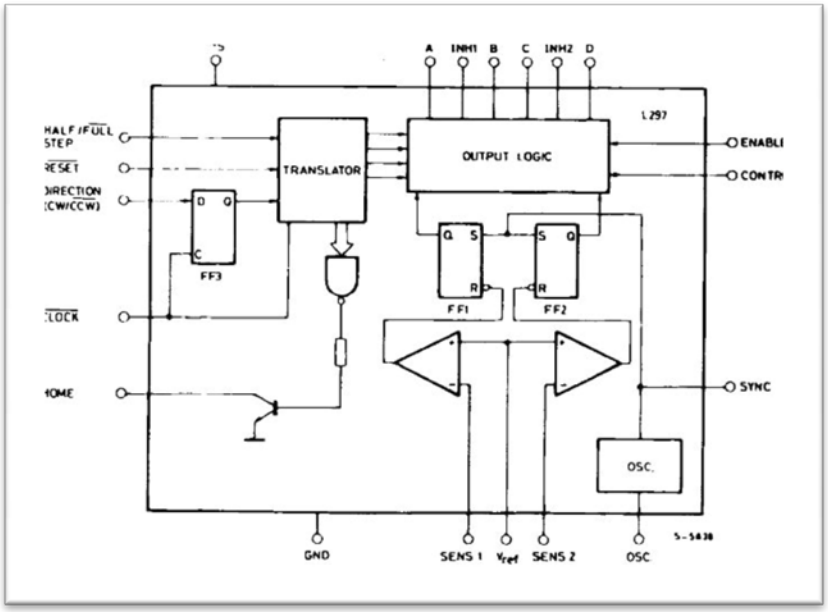
Come si può vedere dalla tabella la condizione in cui entrambi gli ingressi sono a livello alto è proibita, questo perché così facendo si porterebbero in saturazione tutti i transistor del ponte cortocircuitando Vcc verso massa e distruggendo i transistor per eccessivo scorrimento di corrente; questa condizione è evitata via software.

- Scheda controllo motori passo passo

Per la realizzazione di questa scheda si è fatto riferimento al circuito realizzato da Vincenzo Villa utilizzando i circuiti integrati L297 e L298, la documentazione è reperibile al sito http://www.vincenzov.net/progetti/l297.htm.
Il circuito è l'applicazione da datasheet degli integrati L297 e L298 della National: il primo è un circuito integrato di decodifica che riceve in ingresso vari segnali e fornisce in uscita i segnali per un pilotaggio a passo intero o a mezzo passo interfacciandosi direttamente con il ponte ad H integrato L298.
Tramite le linee di ingresso CW/CCW e clock si selezionano rispettivamente il verso di rotazione del motore passo passo e la velocità, ad ogni fronte di salita dell'onda quadra utilizzata come clock il motore avanza di un passo se nella modalità passo intero o di mezzo passo se nella modalità mezzo passo; le due modalità sono selezionate dall'ingresso Half/Full e in questo caso è impostata la modalità mezzo passo essendo il pin posto a 1 tramite una resistenza di pull up, dalla frequenza dell'onda quadra inviata sul clock dipende la velocità di rotazione dei motori.
La velocità di rotazione deve essere regolata in modo opportuno, essa infatti non può essere troppo elevata in quanto il motore tende a perdere coppia

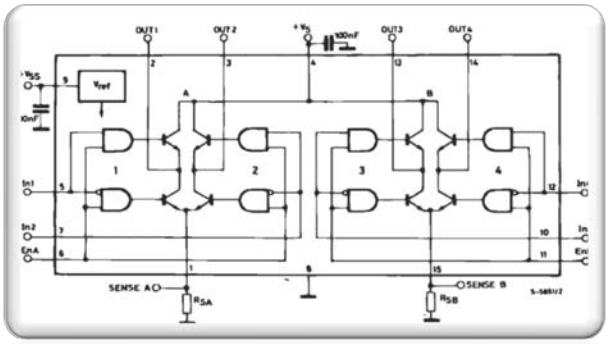
all'aumentare della velocità perdendo anche più facilmente il passo senza seguire più gli impulsi pilota e fermandosi.
Gli ingressi Reset ed Enable permettono rispettivamente di resettare le uscite riportandole allo stato iniziale e di abilitare o meno il funzionamento dell'integrato per cui sono posti a 1 per garantire il funzionamento continuo fintanto che il circuito è alimentato; per quanto riguarda l'ingresso Vref su di esso si riporta una tensione di riferimento per effettuare un controllo sulla massima corrente erogabile dal ponte ad H ai motori.

Schema interno L297



Come si può vedere dallo schema interno dell'integrato sono presenti due ingressi SENSE $_1$ e SENSE $_2$ collegati a due resistenze di sensing che monitorano la corrente che scorre negli avvolgimenti dei motori, quando la tensione ai loro capi (direttamente proporzionale alla corrente assorbita dai motori) supera quella impostata su Vref vengono resettati i latch e il circuito entra in modalità chopper per prevenire danni dovuti a circolazione di extra correnti.
Nella modalità chopper i motori vengono alimentati in modo PWM a una tensione tale da farvi scorrere la massima corrente impostata in base a Vref, per determinare la frequenza di modulazione PWM è presente il blocco OSC. (visibile nello schema interno) completato dalla rete esterna RC che determina la frequenza di oscillazione; sul piedino sync è possibile prelevare quest'onda quadra per sincronizzare il funzionamento di due o più schede nella modalità chopper; con i valori delle resistenze utilizzate nello schema la massima corrente erogabile dal circuito è di circa 2 ampere.
Infine il piedino control determina su quali linee viene azionata la modalità chopper ovvero se è posto alto questo avviene sulle linee ABCD che pilotano direttamente il ponte ad H, altrimenti sulle uscite di inibizione di ciascun avvolgimento ovvero INH 1 e INH 2; il piedino home non è utilizzato per il progetto e pertanto rimane scollegato, esso è un uscita open collector e il transistor a cui è collegato internamente è normalmente in saturazione e si interdice quando sulle uscite ABCD è presente la combinazione di partenza pari a 0101.

Schema interno L298

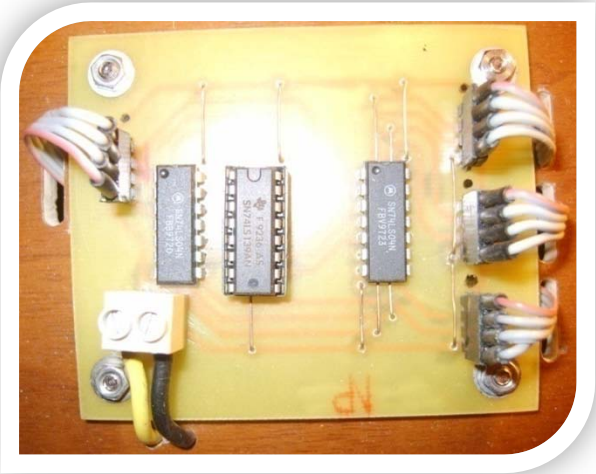


L'L297 si interfaccia direttamente con il ponte ad H integrato L298 di cui è riportato lo schema interno, esso non fa altro che amplificare in corrente i segnali pilota provenienti dall'L297 per applicarli al motore, gli unici componenti discreti aggiunti in più rispetto al ponte ad H sono otto diodi veloci di ricircolo che come per il circuito pilota del motore DC, servono a evitare la circolazione di correnti inverse nei transistor del ponte durante le commutazioni.

Anche l'L298 come il 297 prevede due ingressi per il controllo della corrente che disabilitano il funzionamento dell'integrato quando questa supera un certo valore limite che dipende dal valore delle due resistenze, per comodità si sono utilizzate le stesse resistenze di sensing impiegate per l'altro circuito integrato.

Il circuito necessita di una doppia alimentazione di 5 V per il funzionamento della logica interna degli integrati e 12 V per l'azionamento di potenza dei motori.

- Scheda demux motori

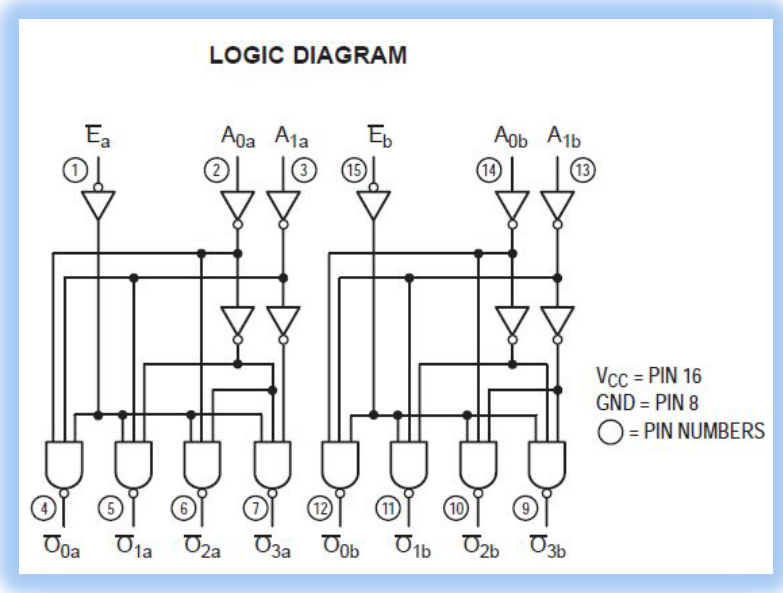


Come visto prima le schede per il controllo di potenza ricevono in ingresso due segnali, per il clock e per il verso di rotazione, per questo motivo servono in totale 6 linee per pilotare direttamente tutte e tre le schede.

Per ovviare a questo inconveniente e ridurre il numero di pin necessari da parte del microcontrollore per interfacciarsi a queste periferiche si è utilizzato un sistema multiplexato gestito da un'apposita scheda, essa riceve in ingresso soltanto 4 segnali ovvero 4 bit provenienti da altrettanti pin del microcontrollore: due di essi sono linee di indirizzo (in realtà sono 4, 2 per ogni demux ma vengono collegate assieme le linee dei due demux per indirizzare lo stesso motore sia per il verso sia per il clock) e selezionano a quale scheda verranno connessi fisicamente gli altri due bit rimanenti relativi al clock e al verso; si è ottenuto così un risparmio di 2 pin sul microcontrollore e si è in grado di utilizzare al meglio l'hardware disponibile.

Per realizzare la funzione voluta si è utilizzato un circuito integrato digitale de multiplexer 74LS139, esso come si può vedere è dotato internamente di due de multiplexer da 1 a 4 e questo permette di indirizzare contemporaneamente il bit del clock e quello del verso inviandone uno in ingresso a un demux e uno in ingresso all'altro; all'ingresso enable A è applicato il segnale del verso mentre all'enable B quello di clock dopodiché l'uscita O_{0a} è connessa all'ingresso Verso della scheda di

Schema interno demultiplexer 74LS139



controllo del motore A e l'uscita O_{0b} è connessa all'ingresso di clock della scheda di controllo del motore A e così via fino al terzo motore; esiste anche una quarta uscita selezionabile dal multiplexer ma rimane inutilizzata per il progetto.
Sotto è riportata una tabella riassuntiva che fa vedere la corrispondenza tra l'indirizzo e il motore selezionato.

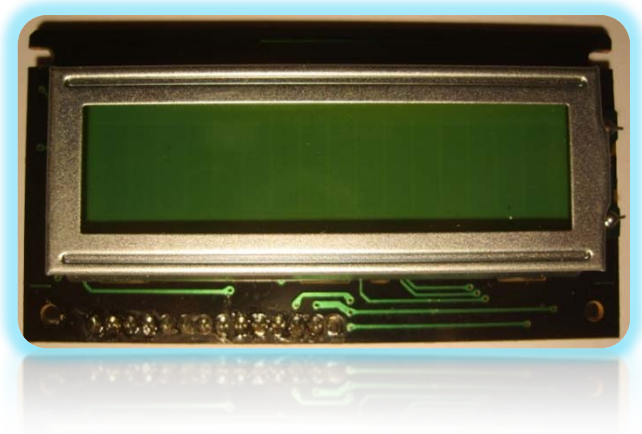
A_{1a}, A_{1b}	A_{0a}, A_{0b}	Motore	Funzione
O	0	Motore A	Motore Pinza
O	1	Motore B	Motore Rot. Base
1	0	Motore C	Motore Verticale
1	1	Non utilizzato	Non utilizzato

Come si può vedere dallo schema interno del de multiplexer questo funziona in logica negativa ovvero gli ingressi enable sono attivi bassi e le uscite sono normalmente nello stato alto e si portano basse quando sono selezionate dalle due linee di indirizzo e l'ingresso enable è a livello basso, per risolvere questo problema e far lavorare il circuito in logica positiva gli ingressi enable sono stati fatti precedere da delle porte not mentre le uscite sono state applicate agli ingressi di altre porte not prelevando l'uscita negata; si sono resi necessari due integrati 74LS04 per realizzare tale funzione dato che in uno sono presenti soltanto 6 porte not non sufficienti per l'operazione da svolgere; in questo modo le uscite sono normalmente a livello basso e si portano alte quando sono selezionate dagli ingressi di indirizzamento e sul piedino enable è presente un livello logico alto.

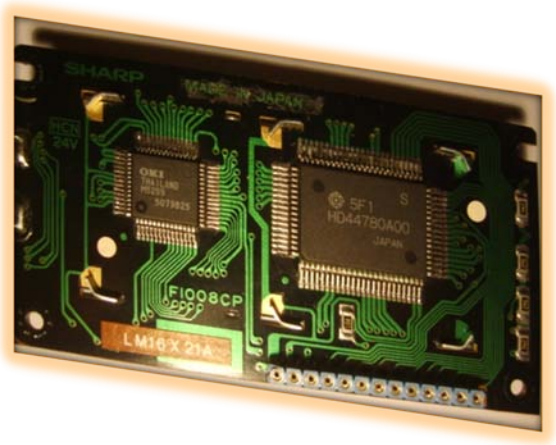
Per quanto riguarda gli integrati utilizzati questi sono tutti della famiglia TTL e del tipo low power schottky, si è scelto di usare la famiglia TTL invece che la CMOS per comodità e per uniformità dato che anche il microcontrollore lavora con tensioni tra 0 e 5 volt, inoltre l'utilizzo della famiglia ad alta velocità e basso consumo consente di aumentare le prestazioni e avere commutazioni sicure e bassi assorbimenti di corrente.

Dopo aver analizzato il funzionamento delle periferiche di ingresso e di uscita del sistema strettamente necessarie al funzionamento automatico del braccio passiamo alle unità di interfaccia con l'utente ovvero la tastiera PS/2 il display e i pulsanti per arrivare infine all'unita centrale di elaborazione: la scheda madre a microcontrollore.

- Display LCD

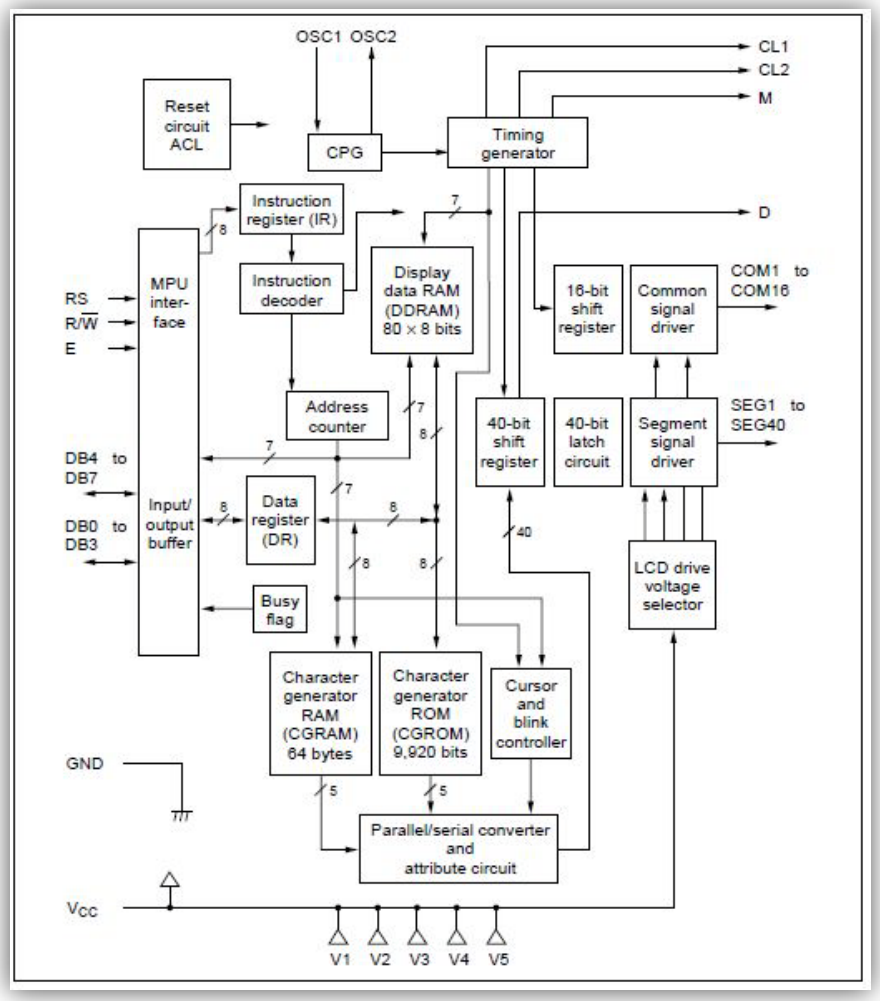


Una delle unità di interfaccia principali verso l'utente è il display LCD, è su di esso infatti che sono visualizzate le operazioni in corso di svolgimento e i messaggi di errore del sistema.
Si è utilizzato il display nelle foto, esso è del tipo LCD (liquid cristal display) a matrice di punti dotato di un led per la retroilluminazione (il display ha un assorbimento di 100 mA dovuto alla retroilluminazione se alimentato a 5 V), possiede due righe di sedici caratteri ciascuna ed è pilotato da un apposito circuito integrato di controllo saldato direttamente sul suo retro (il chip è un HD44780A00 a montaggio superficiale SMD), verso l'esterno si hanno quindi una serie di piedini di controllo facilmente interfacciabili in modo parallelo al microcontrollore e facilmente individuabili sulla scheda (la fila di contatti strip sulla scheda del display).



L'integrato di controllo fornisce autonomamente l'alimentazione per il display e può essere alimentato con tensioni comprese tra 2,7 V e 5,5 V, per il progetto si è alimentato a 5 V come i circuiti logici; come si nota dallo schema interno esso possiede una serie di uscite che sono connesse direttamente ai pannelli del display LCD mentre una parte è dedicata all'interfaccia con il microcontrollore e sono queste le linee utilizzate per il progetto (MPU interface).

Schema interno Hitachi HD44780A00

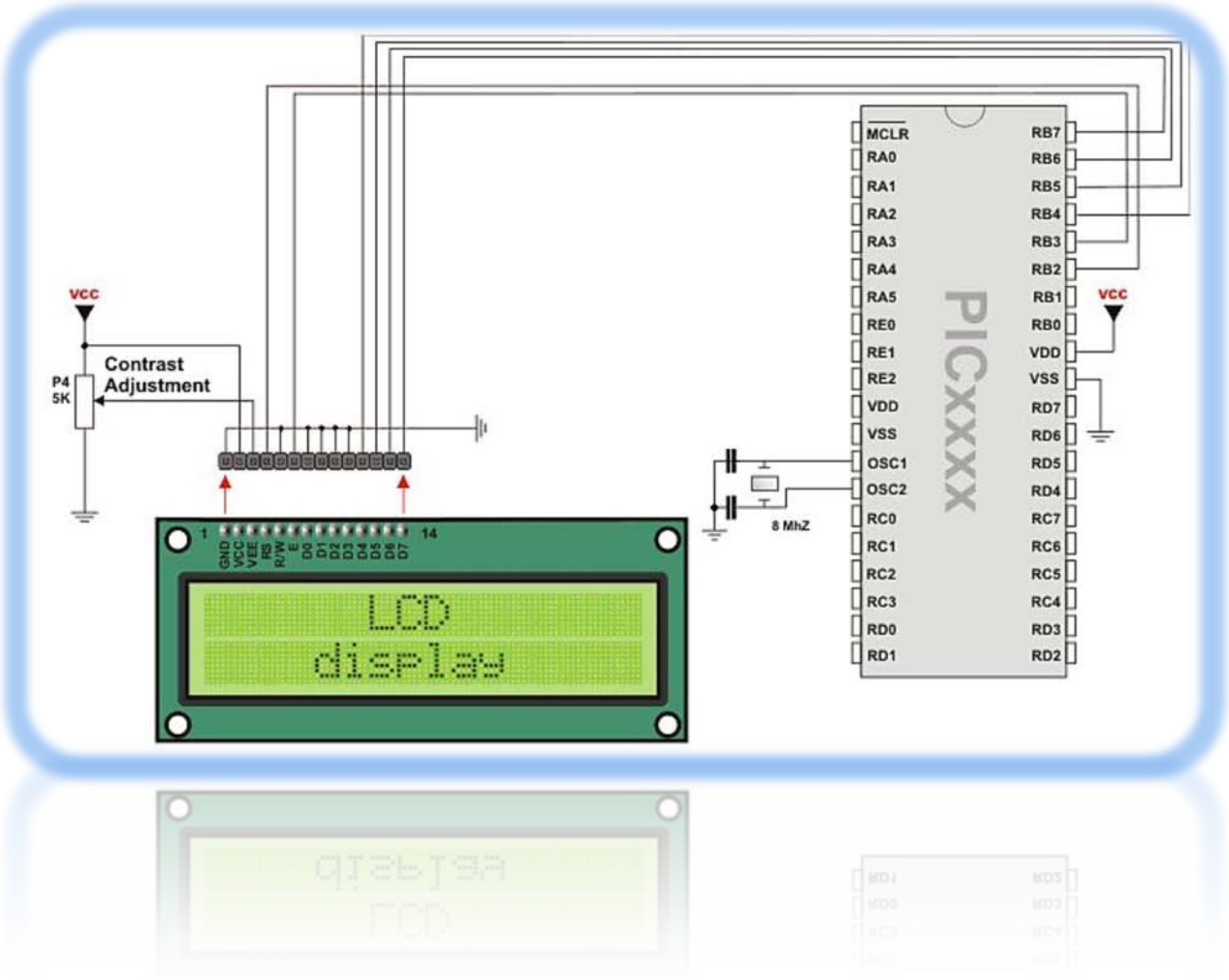


Analizziamo ora la funzione di ogni linea connessa al microcontrollore:

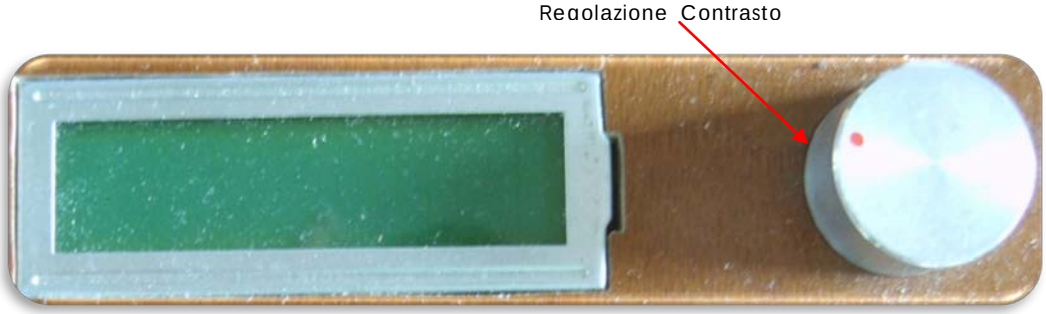
- ***RS - Selects registers***
  Questa linea permette di selezionare i registri interni al chip; se è posto a 0 si selezionano i registri di istruzione mentre se posto a 1 quelli dati.
- ***R/W – Read Write***
  Imposta la scrittura di dati sui registri interni se posto a 1 e viceversa la lettura dei dati contenuti nei registri se posto a zero.
- ***E – Enable***
  Da lo start per le operazioni di lettura scrittura.
- ***DB0 – DB3 - I 4 bit meno significativi del bus dati***
  Queste sono le 4 linee meno significative del bus dati a 8 bit complessivi, esse sono del tipo three state e possono essere settate in ingresso o in uscita e su di esse sono veicolati i 4 bit meno significativi del carattere da scrivere sul display in codice ASCII.
- ***DB4 – DB7 – I 4 bit più significativi del bus dati***
  Queste sono le 4 linee più significative del bus dati, come le 4 precedenti sono impostabili come ingresso uscita e con esse sono veicolati i 4 bit più significativi del carattere se il display è pilotato in

modalità parallela a un byte, per il progetto infatti il display è pilotato in modalità nibble trasferendo il carattere in due fasi in due metà utilizzando soltanto le 4 linee più significative per avere un risparmio sulle porte utilizzate sul microcontrollore.

Di seguito è riportato lo schema di collegamento tra il display e il microcontrollore.



È presente, oltre alle linee già menzionate, un ingresso per il controllo del contrasto del display, su di esso è riportata una tensione variabile tramite un potenziometro che va a variare in modo direttamente proporzionale al suo livello il contrasto del display, inoltre nello schema il display è connesso alla PORTB mente in realtà per il progetto si è utilizzato per comodità la PORTD del PIC.



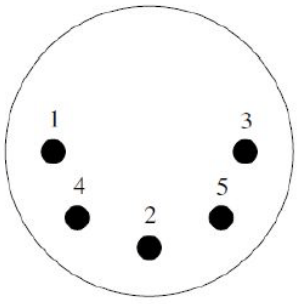
- Tastiera PS2

Per permettere all'utente di interagire direttamente con il sistema in una modalità di telecomando si è utilizzato una tastiera PS/2 utilizzata anche per i PC, una qualsiasi tastiera QWERTY con protocollo di comunicazione PS/2 può essere utilizzata come interfaccia per il progetto.
La comunicazione può avvenire solo in un verso (dalla tastiera al microcontrollore) e non in ambedue le direzioni come avviene normalmente sui PC, per quanto riguarda l'hardware la tastiera è interfacciata direttamente al microcontrollore senza bisogno di particolari accorgimenti tranne due resistenze di pull up necessarie al funzionamento; alla gestione della comunicazione pensano poi delle funzioni integrate nel compilatore utilizzato per realizzare il programma (esse sono responsabili della comunicazione in un solo verso, non riescono a gestire una comunicazione bidirezionale).
Sono necessarie due linee per connettere la tastiera alla scheda a microcontrollore ovvero una per i dati e una per il clock; la trasmissione è di tipo seriale e avviene in modo sincrono scandita dal segnale di clock sulla seconda linea; vediamo ora il funzionamento del protocollo PS2 per la trasmissione da tastiera (host) a microcontrollore.

Il Protocollo PS/2

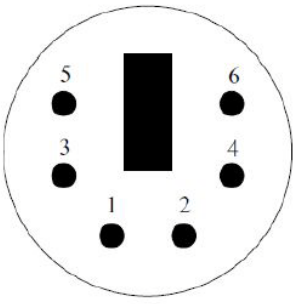
Il protocollo PS/2 è un protocollo di tipo seriale, sincrono e bidirezionale ideato dall'IBM e utilizzato per connettere periferiche come tastiere e mouse ai computer, di seguito sono riportati i pinout dei connettori standard utilizzati per questo tipo di comunicazione, per il progetto si è utilizzato lo standard a 6 pin mini-DIN.

5 pin DIN (AT/XT)



1. clock
2. data
3. non implementato
4. massa
5. +5V

6 pin mini–DIN (PS2)



1. data
2. non implementat
3. massa
4. +5V
5. clock
6. non implementat

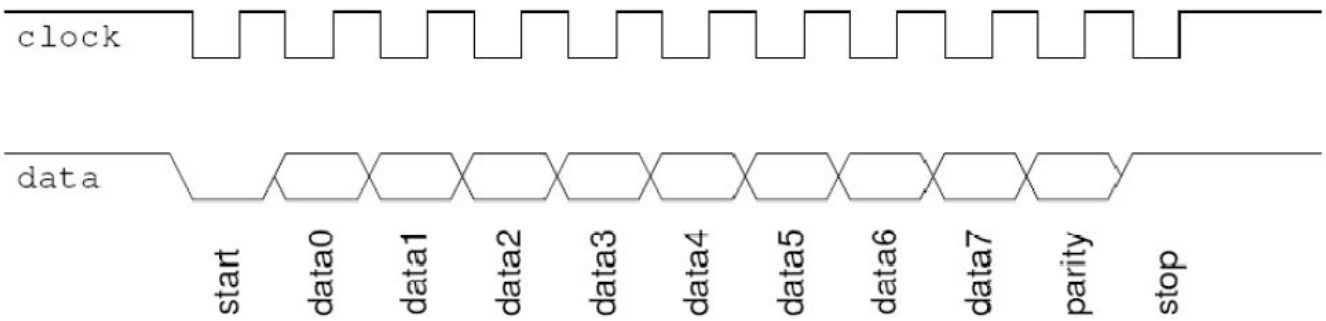
La comunicazione si basa su un bus formato da due linee bidirezionali, una di clock e una per i dati; il bus è in uno stato di attesa quando le due linee sono a livello logico alto ed è questo l'unico stato nel quale la tastiera può trasmettere dati verso il microcontrollore.

L'host, ovvero il sistema a cui è connessa la tastiera, ha sempre il controllo finale sul bus e può in ogni momento inibire la comunicazione mettendo a massa la linea di clock mentre alla tastiera spetta il compito di generare il clock.

I dati spediti dalla tastiera all'host sono letti sul fronte di discesa del clock mentre nella direzione opposta vengono letti sul fronte di salita; per quanto riguarda la frequenza di clock questa rimane nel range 10,0 ÷ 16,7 kHz.

Quando la tastiera vuole spedire un dato, controlla prima di tutto la linea di clock per assicurarsi che sia in uno stato logico alto; se non lo è significa che l'host sta inibendo la comunicazione e la tastiera deve bufferizzare i dati da spedire finché la linea di clock non viene rilasciata per almeno 50 µs, quindi può spedire di nuovo, per trasmettere i dati la tastiera usa il seguente protocollo a undici bit; i dati vengono trasmessi in modo seriale un byte alla volta e ogni byte viene inserito in un frame di undici bit:

- 1 bit di start – Posto sempre a 0
- 8 bit di dati - Ordinati dal meno significativo
- 1 bit di parità
- 1 bit di stop – Posto sempre a 1



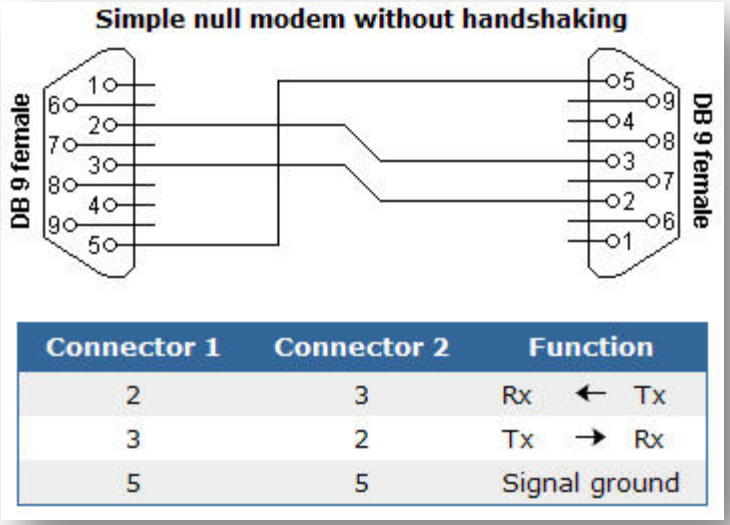
Per quanto riguarda il livello superiore a quello hardware legato alla trasmissione fisica si passa a analizzare i pacchetti che la tastiera invia al microcontrollore; essa non fa altro che generare dei cosiddetti scan codes relativi al tasto che è stato premuto: esistono due tipi di scan code ovvero make code e break code che vengono trasmessi rispettivamente alla pressione e al rilascio di un tasto e assieme costituiscono lo scan code set.
Esistono tre tipi di scan code set che utilizzano tre tipi diversi di codifica per i tasti premuti, è infatti compito dell'host convertire lo scan code in codice ASCII per poterlo elaborare in quanto la tastiera non invia in modo autonomo i dati in questo formato; inoltre la tastiera supporta la cosiddetta modalità typematic ovvero quando un tasto viene tenuto premuto essa continua a inviare il make code a intervalli di tempo regolari fintanto che il tasto è premuto per poi inviare un break code quando viene rilasciato.
Al momento dell'accensione la tastiera esegue un test auto diagnostico (BAT Basic Assurance Test) e setta il tempo di ripetizione per il typematic e il delay: il primo indica la frequenza a cui si ripete il rinvio del make code nella modalità typematic mentre il secondo indica il tempo dopo il quale, se il tasto è ancora premuto, si attiva la funzionalità typematic; i valori di default sono 500 ms per il delay e circa 10 Hz per il typematic.

- Scheda Madre e Seriale

L'unità centrale di elaborazione è proprio la scheda madre, su di essa sono alloggiati, oltre alle interfacce seriale e PS/2, il microcontrollore e tutte le connessioni verso le periferiche.
Prima di scendere nel dettaglio sul microcontrollore prendiamo in considerazione l'interfaccia seriale sia a livello hardware sia a livello software ovvero per quanto riguarda il protocollo utilizzato per il trasferimento dati e il controllo dell'errore.

L'interfaccia seriale RS232

Questo tipo di interfaccia nacque attorno agli anni 60 ad opera della Electronic Industries Association (EIA) per rendere possibile e agevole la comunicazione tra mainframe e terminali attraverso modem su linea telefonica, nel corso degli anni ha subito vari sviluppi e modifiche che possono essere individuate dalla lettera posta dopo la sigla RS232, esistono varie versioni del protocollo in base alla lettera che si trova dopo di esso; il protocollo è piuttosto vecchio ma nonostante ciò viene ancora largamente utilizzato in applicazioni didattiche e per piccoli progetti o apparecchiature industriali dato che è semplice da implementare e ha un costo relativamente basso.
L'interfaccia utilizza un protocollo seriale asincrono il cui segnale elettrico non è bilanciato e la connessione è del tipo point to point; i bit infatti vengono trasmessi in successione su una sola linea con un notevole risparmio sull'hardware e maggiore immunità al rumore rispetto a una trasmissione di tipo parallelo, si ha un solo riferimento di massa (non bilanciato – single ended) e infine la comunicazione avviene tra due soli blocchi ricetrasmittenti.
Come per tutte le comunicazioni si possono individuare dei parametri caratteristici che danno un riferimento sulla velocità di trasmissione ovvero il **bit rate** e il **baud rate**, questi due parametri indicano rispettivamente la quantità di bit trasmessi al secondo e i simboli trasmessi al secondo; dato che la codifica è a due soli livelli ovvero 1 = livello basso, 0 = livello alto, i due parametri coincidono ed esistono delle velocità standard di comunicazione come 1200 bps 2400 bps ecc., la velocità utilizzata per il progetto è di 9600 bps per un tempo di bit di 104 uS,.
Per quanto riguarda i modi di collegamento con l'RS232 si possono operare tutte e tre le modalità standard ovvero simplex, half duplex e full duplex per una comunicazione rispettivamente unidirezionale, bidirezionale alternata con una la linea condivisa commutata e bidirezionale contemporanea; per il progetto si è utilizzato il modo full duplex dato che è necessario uno scambio di dati bidirezionale per il controllo dell'errore.
I collegamenti utilizzati nel caso della comunicazione bidirezionale sono 2 (più la massa) ovvero ognuno per ogni direzione di trasmissione e per questo motivo è stato realizzato un cavo incrociato (tx microcontrollore – rx PC, rx microcontrollore – tx PC) per il collegamento come quello riportato in figura; di seguito sono riportati anche i connettori standard a 25 e 9 pin con la descrizione delle funzionalità di ciascun pin; per il progetto si è utilizzato il connettore a 9 pin dato che è meno obsoleto e più compatto.



Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	Signal ground

Connettore DB9 – 9 poli

Connettore DB25 – 25 poli

Descrizione linee

Sigla	25pin	9pin	In/Out	Nome
TxD o Tx	2	3	O	Dati trasmessi
RxD o Rx	3	2	I	Dati ricevuti
RTS	4	7	O	Request To Send
CTS	5	8	I	Clear To Send
DTR	20	4	O	Data Terminal Ready
DSR	6	6	I	Data Set Ready
RI	22	9	I	Ring Indicator
DCD	8	1	I	Data Carrier Detect
GND	7	5	-	Massa
-	1	-	-	Terra

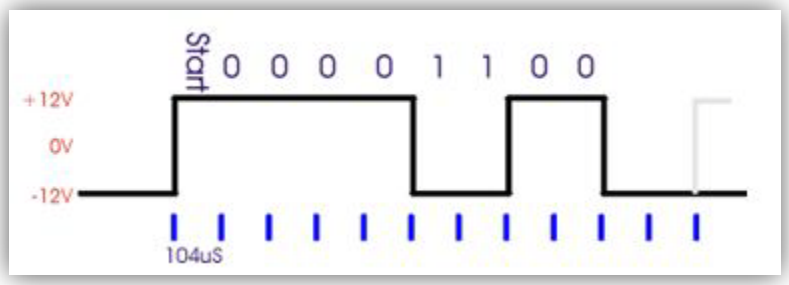
Molti pin previsti per questo tipo di interfaccia servono per il collegamento di un DTE (data terminal equipment) con un DCE (data communication equipment) ovvero un modem e non servono per il progetto per cui non sono collegati; il cavo realizzato per il braccio

prende il nome di null modem in quanto permette di collegare due DTE direttamente simulando la presenza di un modem tra di essi.

Il protocollo start/stop

Come detto i dati viaggiano sulle linee di comunicazione in modo seriale e i due dispositivi si sincronizzano senza l'utilizzo di un clock comune ma soltanto avendo le stesse impostazioni ovvero la stessa velocità di trasmissione (baud rate) e lo stesso tipo di controllo per il bit di parità; il bit di parità come si vede dall'impacchettamento effettuato per la trasmissione è posto in coda ai bit utili e serve per verificare eventuali errori durante la trasmissione, può essere impostato per la parità pari o dispari ma non viene utilizzato per il progetto in quanto è implementato un controllo dell'errore più complesso e affidabile descritto successivamente.

Per il resto la comunicazione avviene secondo il seguente schema trasmettendo un byte alla volta e il pacchetto è preceduto da un bit di start mentre in coda si trova un bit di stop, il pacchetto può essere composto da 7 8 o 9 bit e comunemente si utilizza la trasmissione a 8 bit – 1 byte come in questo caso; le linee sono normalmente alte e i livelli di tensione sono + 12 volt per il livello logico zero e -12 volt per il livello logico 1.



Nella figura è riportato un esempio in cui viene trasmesso il byte 00110000 a una velocità di trasmissione di 9600 baud/sec 8 bit dati e nessuna parità.

Come detto prima i livelli presenti sulla linea di trasmissione seriale non sono compatibili TTL e devono essere adattati per poter interfacciare la linea al microcontrollore che può trattare solo tensioni tra 0 e 5 volt; a questo scopo si è utilizzato l'adattatore di livelli integrato MAX232, esso necessita di una sola linea di alimentazione a 5 V ed è autonomamente in grado di fornire i + 12 – 12 volt necessari con un circuito a pompa di carica sfruttando la carica alternata di alcuni condensatori posti esternamente ad esso (visibili sulla scheda accanto all'integrato), inoltre inverte il segnale in entrambe le direzioni per far gestire la comunicazione dal micro in logica positiva mentre sulla linea i livelli sono invertiti ovvero 1 = livello basso = -12 Volt e 0 = livello alto = + 12 V.

Dato che la seriale viene utilizzata nel progetto per fornire una modalità di telecomando da computer del braccio è necessario trasmettere una serie di comandi più lunghi di un byte (delle stringhe) e avere un collegamento affidabile ovvero non affetto da errori, per questi motivi si è implementato via software il seguente protocollo di trasmissione e controllo dell'errore a più byte.

Protocollo di trasmissione a più byte con controllo dell'errore

Per quanto riguarda la trasmissione dei singoli byte componenti il pacchetto finale previsto dal protocollo per essi si utilizza sempre il protocollo start stop per cui il protocollo per il controllo dell'errore a più pacchetti può essere visto come uno strato superiore di astrazione in un ipotetica organizzazione a layer delle regole di comunicazione.

Il pacchetto completo (telegramma o datagram) si compone dei seguenti elementi

STX	PAYLOAD	ETX

STX – Carattere di inizio trasmissione pacchetto

PAYLOAD – Carico Utile, esso contiene la stringa (il comando) inviata dal PC al micro

ETX – Carattere di fine trasmissione pacchetto

Può succedere che un byte presente nel payload coincida con l'STX o con l'ETX e in tal caso le funzioni per la ricezione riconoscerebbero un nuovo pacchetto o la fine del pacchetto precedente commettendo un errore; per evitare questa situazione si ricorre all'operazione di **escaping,** tramite un'apposita funzione in trasmissione viene verificata la presenza di un byte uguale a STX o ETX nel payload e in tal caso i due caratteri di start e di stop subiscono la seguente trasformazione.

STX diventano 2 caratteri → **ESC A**

ETX → **ESC B**

ESC → **ESC ESC**

ESC diventa un carattere speciale proibito

Per quanto riguarda invece la ricezione se viene ricevuto un carattere pari a ESC viene analizzato anche il carattere successivo e se è A la coppia viene convertita in STX, se è B in ETX, se è ESC in ESC e non è nessuno di questi si è verificato un errore di trasmissione.

Controllo dell'errore

In qualsiasi sistema di telecomunicazione è inevitabile la presenza di rumore e quindi di distorsione dei segnali trasmessi il che potrebbe causare l'insorgere di errori in quanto alcuni bit trasmessi (comunicazioni digitali) potrebbero non essere più riconoscibili, inoltre un altro fattore che determina errori in ricezione è l'interferenza intersimbolica ISI che si verifica quando si hanno problemi di sincronizzazione tra trasmettitore e ricevitore o semplicemente se lungo la linea si verificano sovrapposizioni tra i vari bit trasmessi serialmente uno dopo l'altro e questi non possono più essere discriminati correttamente.

Il parametro che esprime la probabilità di errore valutando la qualità della trasmissione è il BER (bit error rate), esso viene misurato in ricezione ed è espresso dalla seguente formula.

$$BER = \frac{Numero\ bit\ errati}{Numero\ totale\ di\ bit\ ricevuti} \qquad [2.41]$$

Per correggere eventuali errori si implementano vari tipi di controllo e la distinzione principale tra i sistemi di correzione può essere fatta tra gli ARQ e i FEC, il primo tipo a richiesta automatica di ripetizione utilizza un controllo dell'errore in ricezione e nel caso in cui venga rilevato un errore il ricevitore richiede la trasmissione mentre per il FEC ovvero a correzione diretta degli errori vengono implementati degli algoritmi via software che rendono in grado il ricevitore di correggere autonomamente senza ulteriori richieste di invio l'errore.

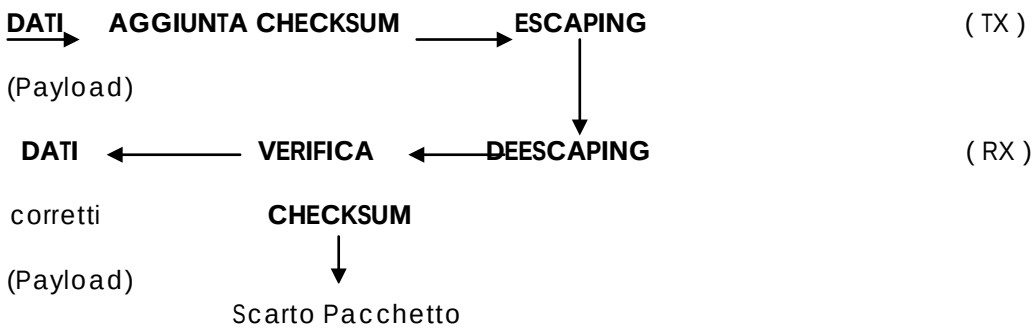
Il secondo tipo è più difficile da realizzare ma garantisce prestazioni migliori soprattutto per le alte velocità in quanto non richiedendo nessun rinvio diminuiscono i tempi necessari altrimenti per questa operazione, per il progetto si è utilizzato la tipologia ARQ con un controllo di tipo Checksum.

Esistono per la tipologia ARQ vari metodi di controllo come quello di parità o il CRC e il checksum: il più versatile e allo stesso tempo affidabile è il checksum, esso consiste nell'aggiungere in trasmissione al pacchetto utile un carattere o più contenente la somma in modulo 2 byte a byte di tutti i byte componenti la stringa trasmessa, dopodiché in ricezione viene ricalcolato il checksum sui byte ricevuti e se è diverso si è verificato un errore altrimenti il dato è corretto e può essere trattato dal programma; di seguito è riportata una schematizzazione di tutto il processo.

CHECKSUM a N BIT (8, 16, 32)

Si sommano tutti i byte del payload, modulo 2^{NBIT}.

STX	PAYLOAD	CKS	ETX



Il programma in ricezione passa a analizzare la stringa ricevuta per eseguire il relativo comando nel caso in cui non ci siano errori mentre in caso contrario crea un pacchetto da inviare al PC e invia un messaggio di errore che si visualizza sul programma di gestione sul PC in modo da essere a conoscenza che si è verificato un errore e rispedire manualmente il pacchetto.

Passiamo ora a vedere il funzionamento del microcontrollore in generale e in modo più dettagliato per le periferiche interne ad esso utilizzate nel progetto.

Il Microcontrollore

Il microcontrollore è il cuore del sistema e è da esso che parte ogni segnale di controllo verso le periferiche, inoltre tutti i sensori sono interfacciati con esso, le schede periferiche sono connesse alla madre tramite dei contatti strip e dei collegamenti realizzati con cavo flat, si è scelto di operare in questo modo invece di realizzare tutto su un'unica scheda per rendere il sistema modulare e facilmente visibile in tutte le sue parti a scopo didattico.

Oltre alle periferiche già descritte sono connessi al microcontrollore i tre pulsanti rossi sulla consolle, essi hanno la funzione di reset del programma, di spegnimento e di cambio modalità (le modalità sono illustrate nella sezione software informatica).

Per il microcontrollore la scelta è caduta su un integrato della microchip ovvero un PIC18F452, inizialmente il progetto era partito con un modello inferiore (PIC16F877A) che si è rivelato nel corso della programmazione insufficiente a ospitare al suo interno tutto il programma per cui si è deciso fare un upgrade con il modello pin-compatibile attuale.

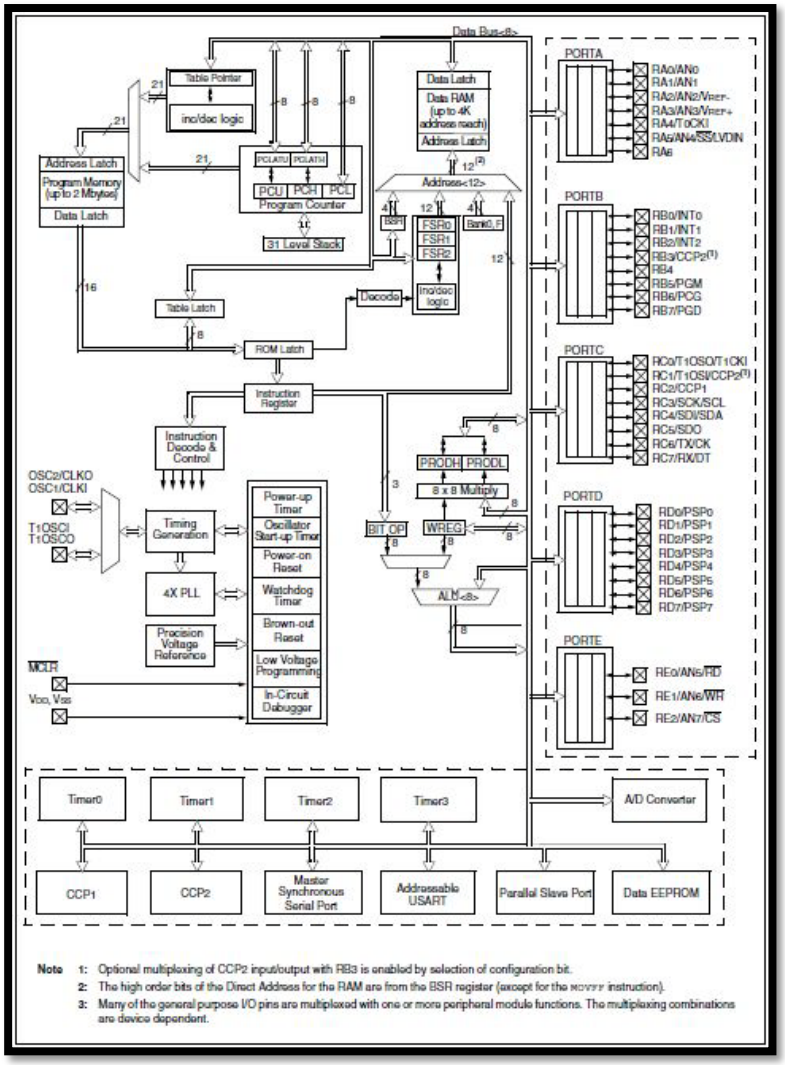
Le caratteristiche principali del microcontrollore sono di avere un'architettura di tipo RISC (reduce instruction set computer), e di possedere 32 Kb di memoria di programma di tipo flash e 1500 Byte di RAM più una memoria EEPROM di 256 byte, il clock è ottenuto da un oscillatore interno stabilizzato tramite un quarzo e due condensatori per realizzare il tipico circuito a pigreco per far oscillare la porta NOT CMOS interna al micro, esso ha una frequenza di 8 Mhz e si può arrivare a far funzionare il circuito fino a 40 Mhz attivando il PLL interno (moltiplicatore x 4) e utilizzando un quarzo esterno a 10 Mhz, tuttavia non sono richieste elevate velocità per il controllo del braccio per cui si è utilizzato un quarzo a 8 Mhz (con il PLL escluso) pienamente sufficiente per la gestione della tastiera PS2, le funzioni per il controllo della tastiera necessitano infatti un clock di almeno 6 Mhz; nonostante il clock sia a 8 Mhz un ciclo macchina dura 4 cicli di clock per cui la frequenza effettiva di funzionamento è di 2 Mhz.

Caratteristiche principali PIC18FXXX

Features	PIC18F242	PIC18F252	PIC18F442	PIC18F452
Operating Frequency	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz	DC - 40 MHz
Program Memory (Bytes)	16K	32K	16K	32K
Program Memory (Instructions)	8192	16384	8192	16384
Data Memory (Bytes)	768	1536	768	1536
Data EEPROM Memory (Bytes)	256	256	256	256
Interrupt Sources	17	17	18	18
I/O Ports	Ports A, B, C	Ports A, B, C	Ports A, B, C, D, E	Ports A, B, C, D, E
Timers	4	4	4	4
Capture/Compare/PWM Modules	2	2	2	2
Serial Communications	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART	MSSP, Addressable USART
Parallel Communications	—	—	PSP	PSP
10-bit Analog-to-Digital Module	5 input channels	5 input channels	8 input channels	8 input channels
RESETS (and Delays)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)	POR, BOR, RESET Instruction, Stack Full, Stack Underflow (PWRT, OST)
Programmable Low Voltage Detect	Yes	Yes	Yes	Yes
Programmable Brown-out Reset	Yes	Yes	Yes	Yes
Instruction Set	75 Instructions	75 Instructions	75 Instructions	75 Instructions
Packages	28-pin DIP 28-pin SOIC	28-pin DIP 28-pin SOIC	40-pin DIP 44-pin PLCC 44-pin TQFP	40-pin DIP 44-pin PLCC 44-pin TQFP

Il dispositivo è dotato internamente di vari moduli, molti dei quali non sono utilizzati per il progetto, di seguito è riportato lo schema interno e la descrizione dei moduli utilizzati.

Schema interno PIC18F452



Di tutti i moduli presenti quelli che interessano particolarmente sono le porte di interfaccia verso l'esterno e la USART, le prime sono svariate e ad ognuna di esse è associata una lettera per identificarla PORTA, PORTB, PORTC, PORTD, PORTE, hanno diversi numeri di bit e possono essere impostate come ingresso o come uscita e alcuni bit di esse possono essere settate come ingressi analogici per il convertitore analogico digitale interno, tuttavia per il progetto non si è utilizzato l'ADC per cui tutte le porte sono impostate tramite appositi registri per funzionare in modalità digitale o come ingresso o come uscita.

Per quanto riguarda l'USART essa gestisce la comunicazione seriale a livello del protocollo start stop mentre il controllo dell'errore è implementato nel firmware a un livello superiore, essa viene impostata tramite appositi registri per una comunicazione a 8 bit senza parità e 9600 bps ed è in grado di gestire la ricezione in modo asincrono anche rispetto alle operazioni che sta svolgendo il microcontrollore, essa infatti ogni volta che riceve un byte genera un interrupt per cui il processore è libero di svolgere le sue operazioni mentre vengono inviati dei comandi sulla seriale senza dover dedicare il suo tempo interamente al controllo dell'USART con il metodo del polling.

Sempre per quanto riguarda l'hardware, esternamente al micro sulla scheda madre è presente un connettore per il pulsante di reset situato nella parte superiore della consolle, tramite questo pulsante si porta a zero il pin MCLR determinando il reset del programma e il riavvio da zero; per resettare correttamente il sistema va tenuto premuto il tasto di accensione assieme al pulsante del reset per evitare il distacco del releis nella scheda di controllo alimentazione e lo spegnimento del sistema.

Informatica

Il vero e proprio cuore del sistema è il software di gestione; esso si divide in due parti: una sul computer e una sul microcontrollore (firmware); per quanto riguarda il PC si è utilizzato il linguaggio di programmazione visual basic mentre per il firmware il C, di seguito sono riportate alcune indicazioni sul compilatore utilizzato per il microcontrollore e l'interfaccia hardware per la programmazione: il programmatore USB.

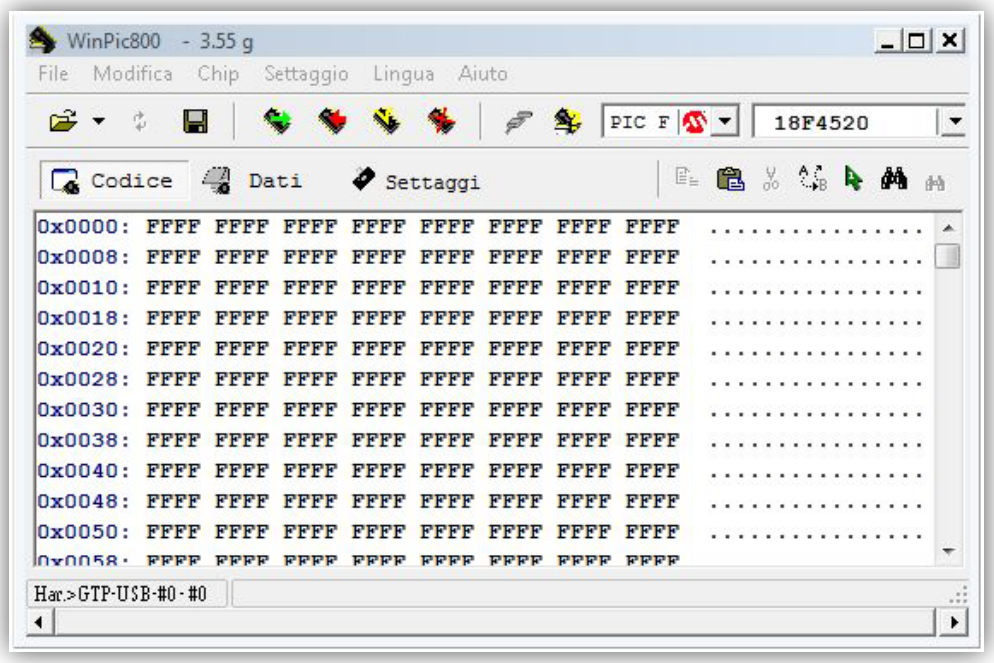
Un'altra sezione che ha richiesto l'utilizzo di software sul PC è stata la progettazione dei PCB e il disegno dei circuiti elettronici effettuato con la suite di ORCAD 16 e il programma freeware FIDOCAD.

- Compilatore MikroC e programmatore USB

IL Programmatore

Per quanto riguarda la programmazione a livello hardware i microcontrollori PIC supportano il cosiddetto ICSP ovvero In Circuit Serial Programming; grazie a questa funzionalità è possibile programmare direttamente il dispositivo senza estrarlo dalla scheda; sulla scheda madre è visibile un connettore a 6 pin posto al di sopra dello zoccolo, su di esso sono veicolati i segnali per la programmazione ICSP ed è qui che si connette il programmatore con un apposito cavo flat, è inoltre presente su uno dei pin un'alimentazione a 12 Volt necessaria durante la fase di programmazione.

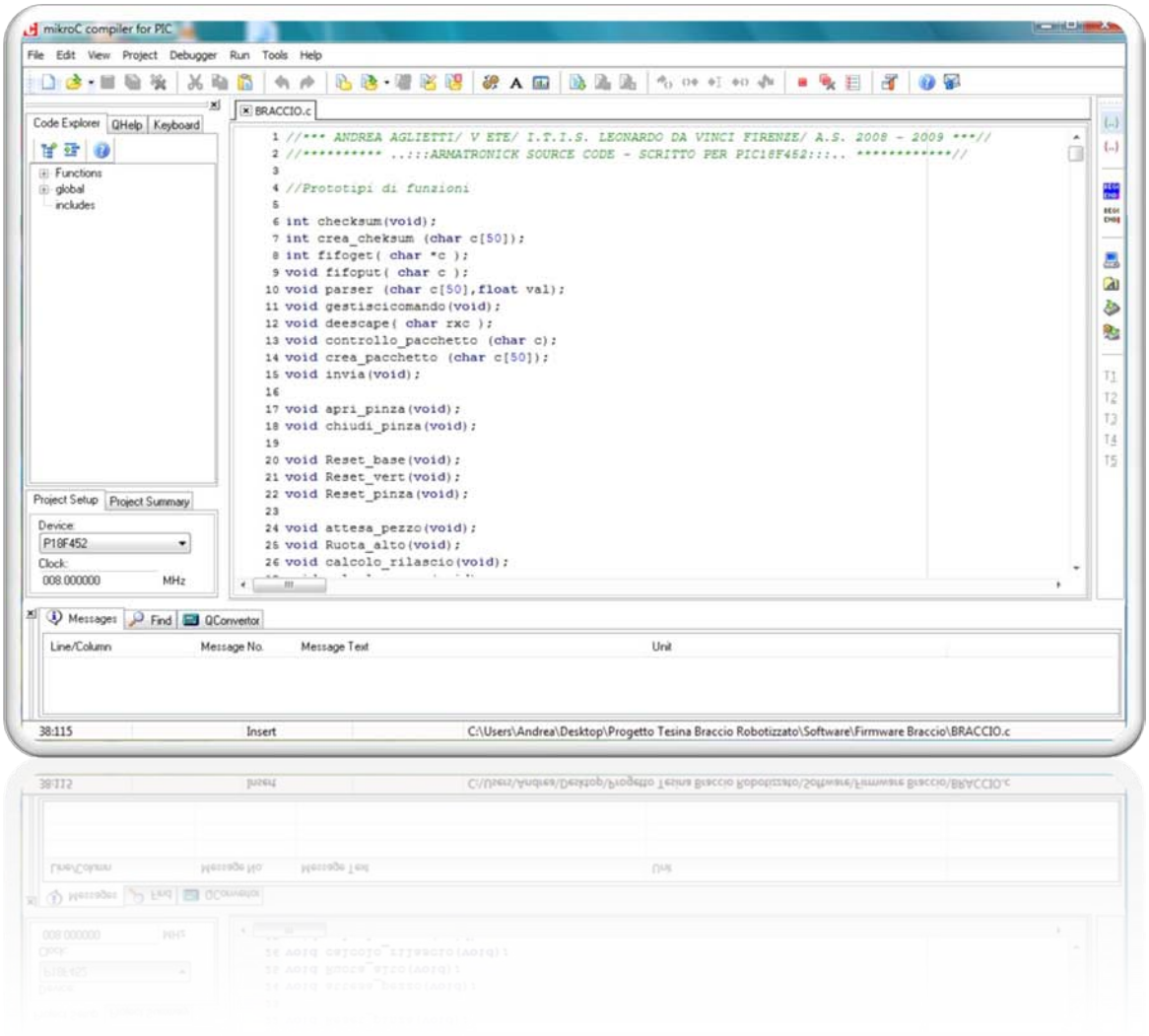
Il programmatore USB è stato anch'esso realizzato in casa ed è in grado di programmare, in abbinamento a un programma apposito di gestione sul PC (WinPIC800), tutti i PIC della serie 16 e 18, è dotato di uno zoccolo ZIF e dell'interfaccia per l'ICSP, si interfaccia al computer con una connessione di tipo USB e per il funzionamento utilizza anch'esso un PIC che è stato preventivamente programmato utilizzando un secondo programmatore di tipo seriale; di seguito è riportata una videata del software WinPIC800 mentre gli schemi del programmatore sono allegati assieme a quelli delle altre schede.



Il compilatore

Dato che il microcontrollore non è stato programmato in linguaggio assembly si è reso necessario l'utilizzo di un compilatore in linguaggio C dato che questo offre molte possibilità anche a livello piuttosto basso, si può agire sui singoli bit dei registri del microcontrollore ma svolgere anche operazioni più complesse grazie alle funzioni integrate del C.

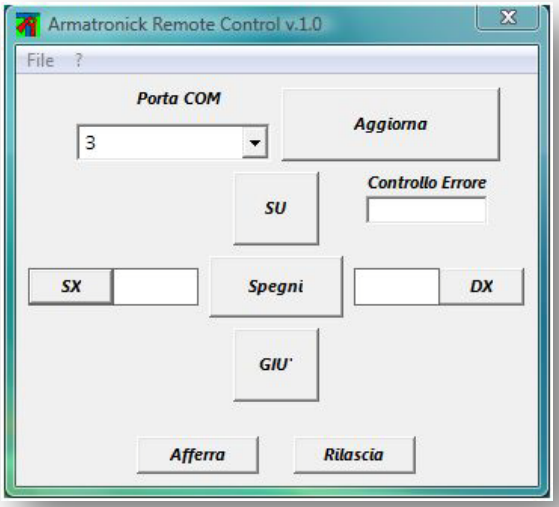
Il compilatore impiegato è il mikroC della Mikroelettronica (di seguito è riportata una videata del programma), con esso si crea un file .hex (formato esadecimale) pronto per essere trasferito sulla memoria del microcontrollore, dopodiché si apre questo file con WinPIC800 e si trasferisce sulla flash del PIC tramite il programmatore ICSP; l'operazione di trasferimento è molto veloce anche se il programma impiega quasi tutta la memoria a disposizione (circa l'80 % di memoria di programma e 70 % di ram).



Il programma offre varie funzionalità tra cui un tool di debug, tra le caratteristiche più interessanti è da tenere presente l'esistenza di alcune funzioni proprie del compilatore per la gestione del display, dell'USART e della comunicazione con la tastiera PS/2.

- Programma di controllo remoto da PC

Esso è stato realizzato con il software visual basic 6 e il relativo linguaggio di programmazione orientato agli oggetti, grazie ad esso si è in grado di realizzare un controllo remoto da computer del braccio (la gestione non avviene sul PC, da esso si richiamano soltanto delle funzioni presenti sul micro), come si vede nella videata si possono dare vari comandi direttamente cliccando sui pulsanti e inviare così su seriale la stringa corrispondente a quel comando utilizzando il protocollo visto in precedenza per la trasmissione a più byte con il controllo dell'errore; oltre a quanto già visto il programma trasmette in coda alla stringa vera e propria contenente il comando un valore a 4 cifre che contiene un eventuale parametro necessario per l'esecuzione del comando.



Nel caso dei comandi per la rotazione si inserisce infatti il numero di gradi di cui si vuol far ruotare il braccio e sarà questo il valore assunto dalle 4 cifre spedite nel pacchetto.
Le funzioni che è possibile richiamare sono molteplici e non necessitano di particolari spiegazioni data l'esplicità del programma, esso permette, oltre ad effettuare rotazioni, abbassamenti elevamenti, prese e rilasci di pezzi, di far eseguire un reset della meccanica totale o selettivo oppure un semplice riposizionamento selettivo con cui il braccio andrà a riposizionare solo le parti che effettivamente hanno bisogno di essere riportate in posizione di partenza; un'altra funzione che è possibile richiamare è il cambio di velocità dei motorini, esistono infatti tre velocità (di default è selezionata quella intermedia) ed è possibile selezionare quella che più si adatta al tipo di operazione da effettuare.

Infine le ultime funzionalità del programma sono la selezione della porta COM tra quelle fisicamente presenti sul computer e la visualizzazione del messaggio di conferma ricevuto dal micro per il controllo dell'errore: in condizioni normali (nessun errore) dopo l'invio di un comando il micro risponde sempre con un OK mentre se si verifica un errore sul cheksum CK, altrimenti se il micro non rileva errori ma non riconosce il comando invia un terzo messaggio di ritorno ovvero EC.

- Il Firmware

Dato che sarebbe dispersivo descrivere tutto il programma nei minimi particolari prestando attenzione a tutte le funzioni e istruzioni si prende come riferimento il programma principale all'interno del main per dare una panoramica sulle capacità e sulle funzioni più importanti rimandando ai commenti presenti sul codice per eventuali chiarimenti e dettagli.
Il programma principale ovvero la funzione main ha un'organizzazione principale legata allo switch visibile all'inizio di esso; il programma infatti dopo aver settato i vari registri (porte in entrata uscita, accensione led relais, visualizzazione messaggi sul display ...) entra in un ciclo infinito (while(1)) all'interno del quale è presente uno switch case che regola il funzionamento di tutto il sistema.
Esso determina il case in base al valore di select: select è una variabile che indica la modalità nella quale si vuole far lavorare il braccio, è impostata di default a 0 per entrare nella prima modalità ovvero quella di reset; all'interno di questa modalità il braccio esegue una serie di funzioni di reset e si posiziona per essere pronto a effettuare le varie operazioni, dopodiché passa automaticamente al case 1 ovvero entra nella modalità automatica, le altre modalità sono case 2 – controllo da PC, case 3 – controllo da tastiera, case 4 gestione degli errori e case 5 – routine di spegnimento.
Si può passare da una modalità all'altra agendo sul pulsante per il cambio di modalità, esso funziona tramite un interrupt e nella funzione dell'interrupt sono gestite, oltre alla pressione di questo pulsante, la comunicazione seriale e la pressione del pulsante per lo spegnimento; passiamo ora ad analizzare il funzionamento in ciascuna modalità.
Se alla richiesta di cambio modalità il sistema sta eseguendo delle operazioni queste non vengono interrotte ma viene messo in attesa la richiesta che viene eseguita solo alla fine di tali operazioni.

Modalità di reset

In questa modalità, che viene attivata all'accensione, sono eseguite le funzioni per il posizionamento alla posizione di partenza zero delle varie parti meccaniche del braccio per poi impostare alla fine del ciclo il select a 1 e entrare nella modalità automatica selezionata di default.

Modalità automatica

Nella modalità automatica il sistema rileva autonomamente la presenza di un pezzo quando questo interrompe la fotocellula posta sulla base e dopodiché esegue una serie di operazioni per afferrarlo e spostarlo di 90 gradi verso destra o verso sinistra; la funzione per la rotazione della base riceve come parametri il numero di passi di cui deve ruotare e il verso di rotazione, ad ogni nuovo pezzo il verso di rotazione si inverte per cui si ottiene il rilascio di un pezzo 90 gradi a destra e del successivo 90 gradi a sinistra della base e così via, una volta che il braccio ha posato il pezzo si riposiziona automaticamente ed è pronto per prenderne di nuovi.
La funzione per l'abbassamento permette di far afferrare al braccio qualsiasi pezzo (entro i limiti di altezza e larghezza massimi) a ¾ della sua altezza per poi rilasciarlo sempre a pari sul piano nonostante che la base sia leggermente sopraelevata rispetto ad esso.

Modalità controllo da PC

In questa modalità è previsto un controllo remoto dal PC tramite il programma di gestione visto precedentemente, tutto si basa sulla comunicazione seriale, il firmware infatti riceve il comando e attraverso la funzione parser (interprete dei comandi da seriale) richiama la funzione relativa al comando inviato da PC.
La gestione della seriale avviene su interrupt di modo che il programma sia in grado di ricevere fino a 50 byte, anche in corso di svolgimento di un'operazione, memorizzandoli in una FIFO per poi andare a estrarre il comando dalla FIFO quando effettivamente può effettuare l'operazione per eseguire la funzione richiesta, non si ha quindi perdita di informazioni dato che si utilizza l'interrupt.

Modalità controllo da tastiera

Per quanto riguarda il controllo da tastiera questo assomiglia molto al telecomando da PC, attraverso la tastiera infatti si possono inserire una serie di comandi che vengono visualizzati durante la scrittura sul display e vengono riconosciuti da una funzione interprete dei comandi apposita per la tastiera ovvero parser PS2; allo stesso modo della seriale una volta inserito il comando e premuto invio il firmware va ad analizzarlo e lo esegue andando a richiamare le relative funzioni in caso si sia inserito un comando esistente altrimenti restituisce sul display un messaggio di errore "comando errato".
I comandi riconosciuti sono i seguenti:

ruotagiu	ruota verso il basso
ruotasu	ruota verso l'alto
ruotadx	ruota a destra (richiede l'inserimento dei gradi)
ruotasx	ruota a sinistra (richiede l'inserimento dei gradi)
apri	apre la pinza
chiudi	chiude la pinza
riposiziona	riposiziona la meccanica selettivamente
resetall	resetta tutta la meccanica
resetbase	resetta la base rotante
resetvert	resetta le leve per la rotazione verso l'alto o il basso
reset pinza	resetta la pinza
spegni	entra nella modalità di spegnimento del sistema
velvert	entra nel menù di selezione velocità verticale
velbase	entra nel menù di selezione velocità di rotazione della base

Dalle modalità tastiera e PC si possono effettuare operazioni che non sono necessariamente legate al posizionamento del pezzo sulla base ovvero è possibile prendere un pezzo entro + - 90 gradi dalla base per rilasciarlo in qualsiasi posizione entro questo angolo; il programma gestisce automaticamente la rotazione e impedisce di far ruotare il braccio oltre questo angolo predefinito, inoltre le funzioni ruota_basso e ruota_alto sono in grado di gestire il rilascio del pezzo indipendentemente dallo scalino costituito dalla base (un pezzo può essere preso fuori di essa e rilasciato sopra o viceversa senza problemi di rilascio, il pezzo viene posato sempre a pari).

Modalità Gestione errori

Durante il funzionamento può capitare che si verifichino degli errori dovuti a cadute del pezzo o errori dell'utente o quant'altro per cui è necessario introdurre un controllo su questo tipo di errori: gli errori gestiti sono riportati di seguito assieme alla modalità con cui viene individuato e risolto ciascuno di essi.

Errore	**Codice Errore**
Finecorsa Su	1
Finecorsa giù	2
Posizione base errata	3
Pezzo caduto o rimosso	4
Errore pinza	5

All'interno di quasi tutte le funzioni per la movimentazione sono presenti dei controlli per rilevare la situazione di errore e nel caso in cui si verifichi, il controllo provvede ad uscire dalla funzione e imposta select a 4 per entrare nella modalità di gestione errori, assegna il codice relativo all'errore rilevato alla variabile error_code e memorizza il select di provenienza per tornare, una volta corretto l'errore, nella modalità precedente.
Una volta entrato nella gestione dell'errore il programma visualizza sul display l'errore e fa lampeggiare il led rosso per circa 5 secondi, dopodichè rilascia il pezzo se è presente e passa a effettuare un reset selettivo ovvero un riposizionamento delle parti sposizionate a causa dell'errore, dopodiché reimposta il vecchio select e ritorna nella modalità di esecuzione precedente all'errore.

Modalità Spegnimento

L'entrata nella modalità di spegnimento può essere effettuata premendo il pulsante relativo oppure direttamente tramite il comando spegni da PC o da tastiera ed è regolata dallo stato della variabile spegni; se viene posta a 1 il programma è abilitato a entrare nella funzione di spegnimento altrimenti no.

Per quanto riguarda la pressione del pulsante questa viene gestita da un interrupt e in tal caso viene posta a 1 la variabile spegni, se si è in modalità automatica inoltre il programma non entra immediatamente nella funzione di spegnimento ma finisce il ciclo del pezzo presente ovvero lo rilascia nella posizione finale e si riposiziona, in tutti gli altri casi invece alla richiesta dello spegnimento il programma finisce l'esecuzione della funzione in corso per poi spegnersi.

La funzione di spegnimento vera e propria funziona in questo modo, una volta entrati nel case 5 viene effettuato un riposizionamento con il rilascio del pezzo se ancora presente e dopodiché viene tolta l'alimentazione al relais della scheda per il controllo alimentazione consentendo quindi lo spegnimento del sistema, inoltre sul display si visualizzano le operazioni in corso e in questo caso il seguente messaggio "Arresto Sistema In Corso".

I messaggi verso l'utente

L'utente è informato costantemente riguardo alle operazioni in corso di svolgimento tramite delle scritte sul display e inoltre alcune informazioni vengono date anche dai led che possono accendersi secondo varie combinazioni: durante il funzionamento normale è acceso soltanto il led verde a indicare il corretto funzionamento del sistema, se si richiede un cambio di modalità si accende anche il led rosso fintanto che questo non avviene e infine sulla richiesta di spegnimento si spegne il led verde e si accende quello rosso per far vedere che una volta finite le operazioni il sistema si spegnerà, è previsto anche un lampeggio del led rosso nel caso in cui si verifichi un errore.
Per quanto riguarda il passaggio tra le modalità, tutte le volte che questo avviene, il programma controlla che non ci siano pezzi in movimentazione e che il braccio sia in posizione di partenza e se non lo fosse o ci fosse un pezzo lo rilascia e si riposiziona effettuando un reset selettivo delle sole parti da riportare a zero prima di passare alla modalità successiva.

Tutte le funzioni per la movimentazione hanno un controllo interno che evita l'esecuzione nel caso in cui questa sia impossibile, ad esempio se si cerca di chiudere la pinza quando questa è già chiusa si visualizza il messaggio "impossibile chiudere pinza" e la pinza non viene chiusa, lo stesso vale per le altre funzioni che regolano la movimentazione meccanica del braccio.

Conclusioni

Nonostante i vari problemi riscontrati durante le varie fasi di sviluppo si è riusciti a portare a termine anche la realizzazione pratica del progetto riuscendo nell'intento di avere un'applicazione tangibile dei metodi teorici studiati per il sistema.

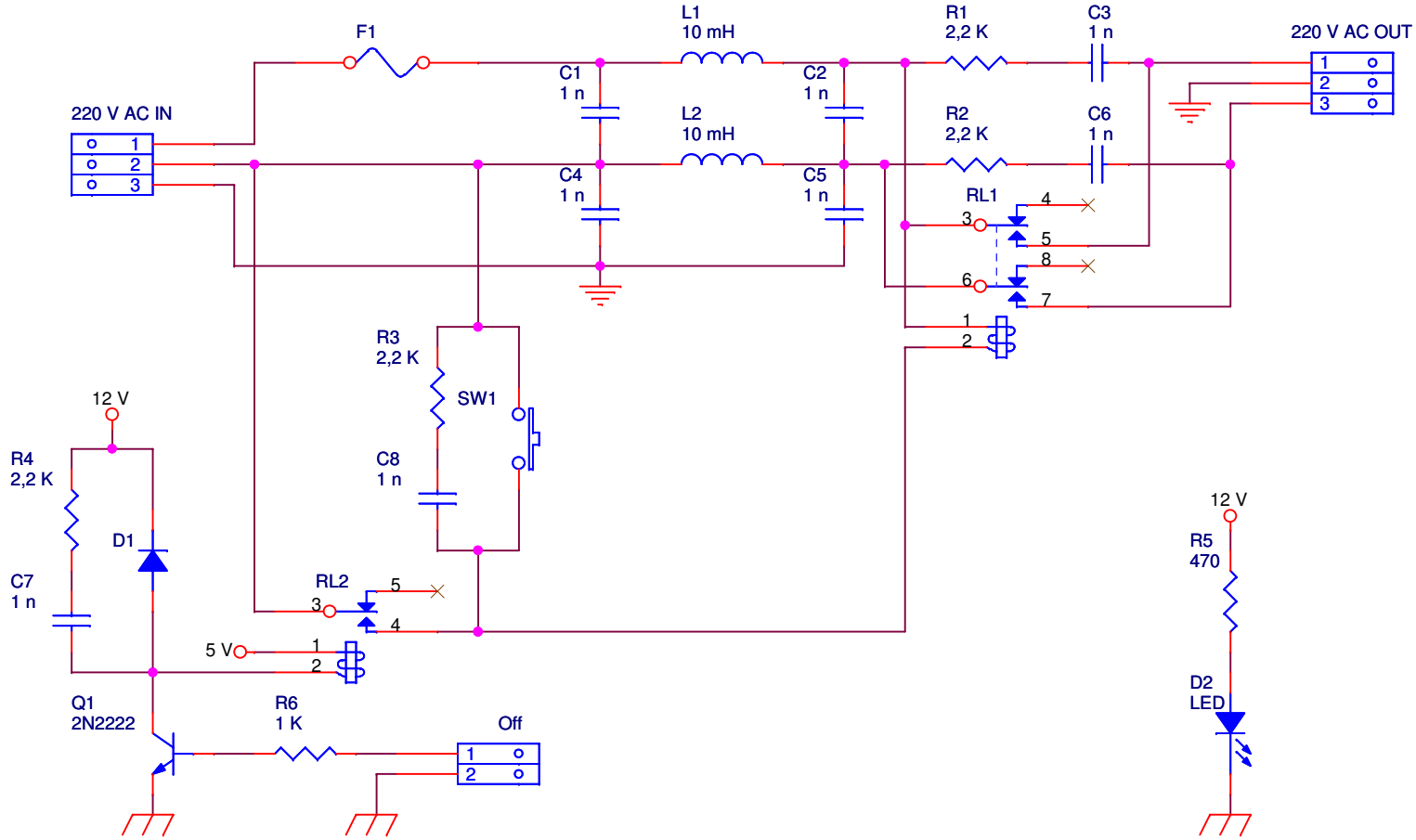
Ringraziamenti

Si ringraziano per gli insegnamenti ricevuti, utili alla realizzazione del progetto, tutti i professori della specializzazione elettronica e telecomunicazioni, inoltre un ringraziamento particolare va allo studente di V Informatica Gabriele Posca per la collaborazione prestata al progetto riguardo alla comunicazione seriale.

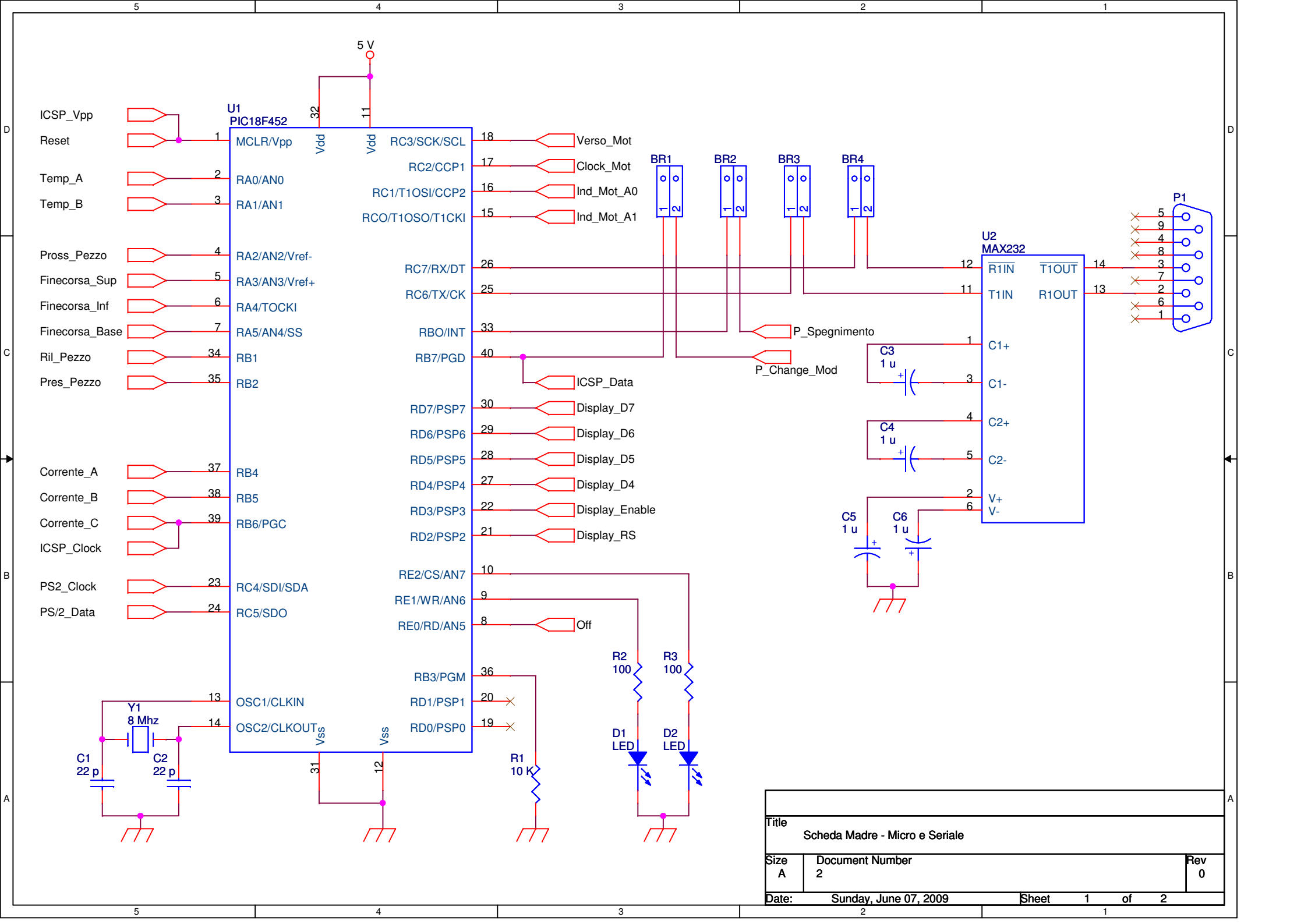
Schematici

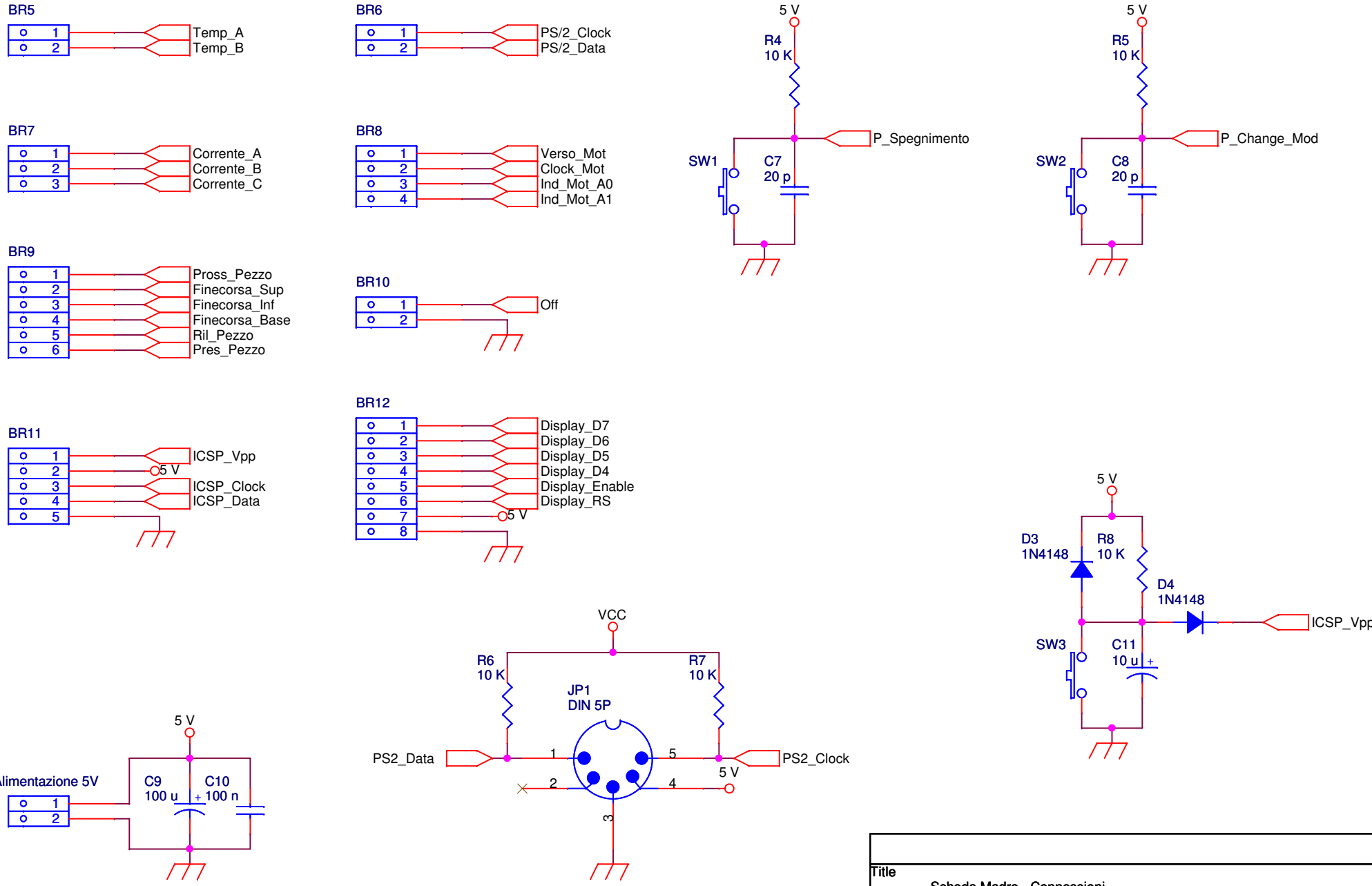
Liste Componenti

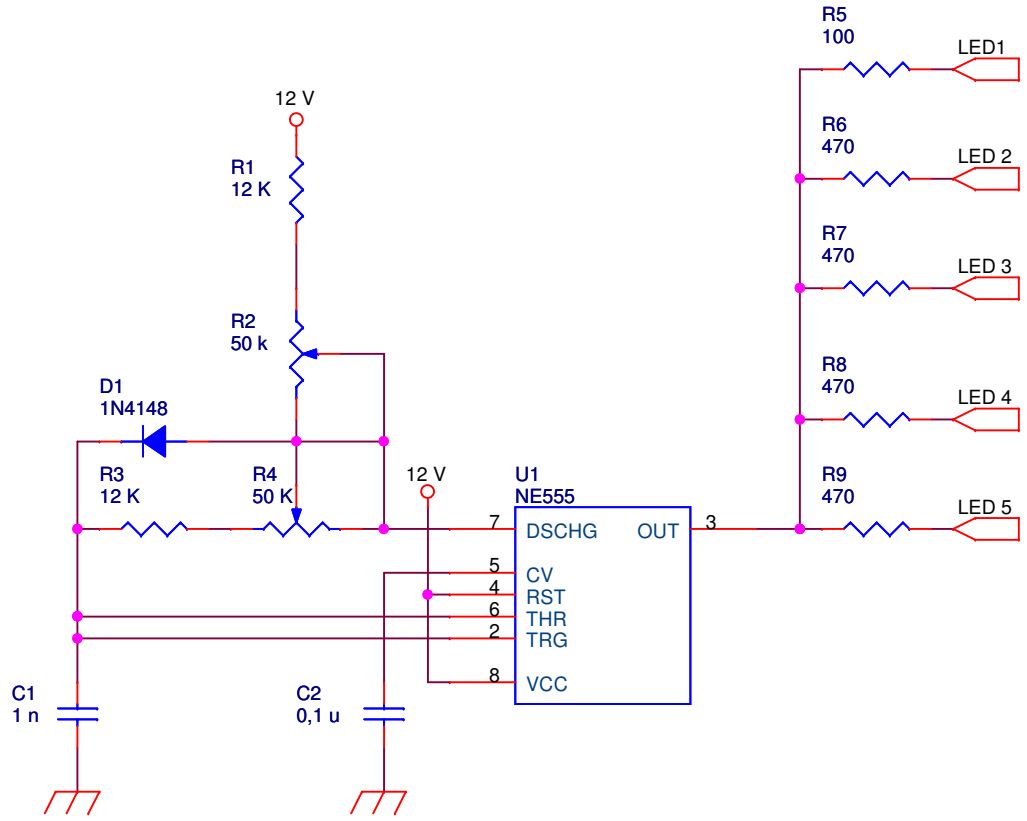
PCB



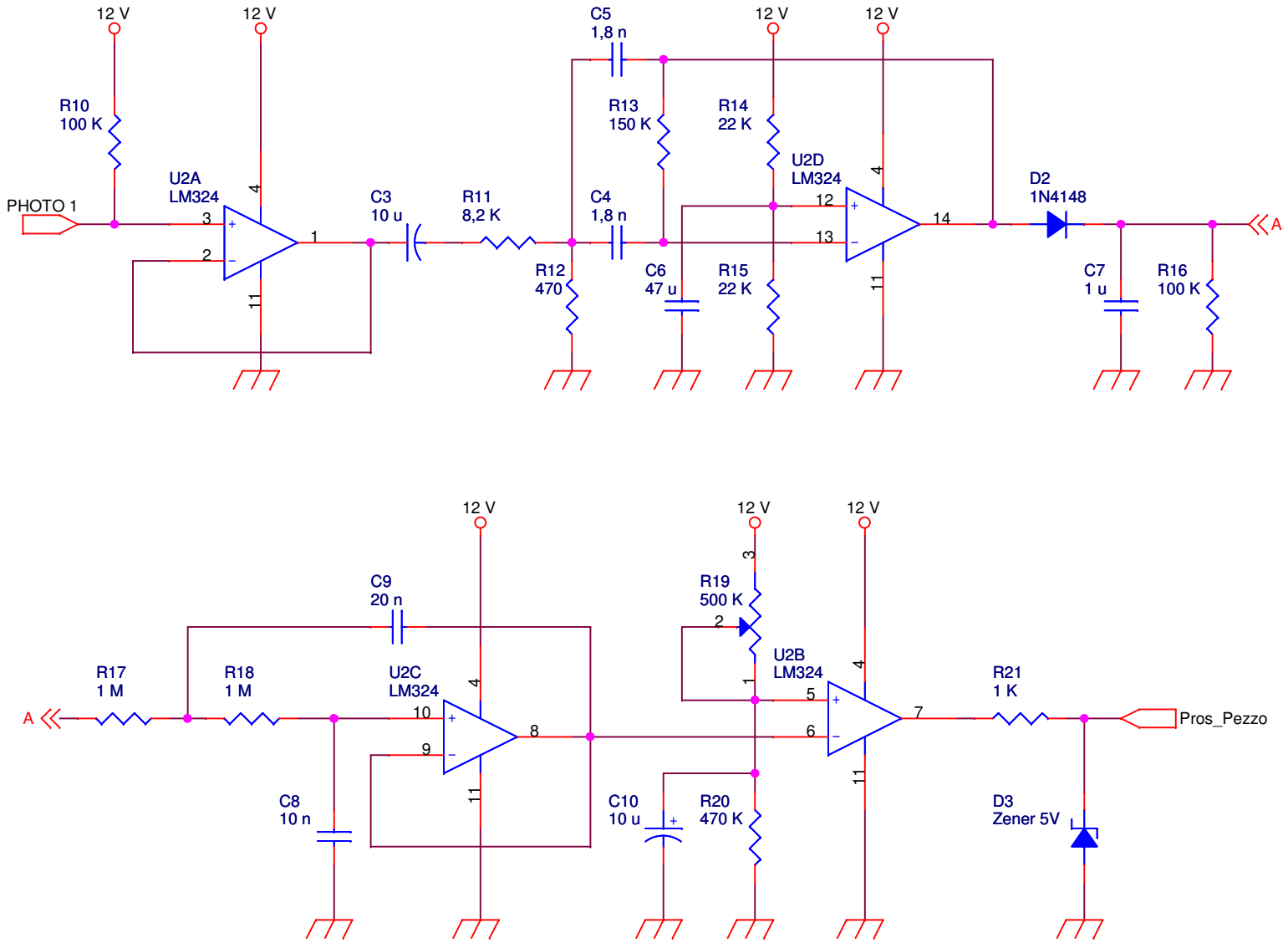
Title	Scheda Controllo Alimentazione	
Size A	Document Number 1	Rev 0
Date: Sunday, June 07, 2009	Sheet 1 of 1	


5 V
U1
PIC18F452
ICSP_Vpp
Reset
Temp_A
Temp_B
Pross_Pezzo
Finecorsa_Sup
Finecorsa_Inf
Finecorsa_Base
Ril_Pezzo
Pres_Pezzo
Corrente_A
Corrente_B
Corrente_C
ICSP_Clock
PS2_Clock
PS/2_Data
MCLR/Vpp
RA0/AN0
RA1/AN1
RA2/AN2/Vref-
RA3/AN3/Vref+
RA4/TOCKI
RA5/AN4/SS
RB1
RB2
RB4
RB5
RB6/PGC
RC4/SDI/SDA
RC5/SDO
OSC1/CLKIN
OSC2/CLKOUT
Vdd
Vss
RC3/SCK/SCL
RC2/CCP1
RC1/T1OSI/CCP2
RCO/T1OSO/T1CKI
RC7/RX/DT
RC6/TX/CK
RBO/INT
RB7/PGD
RD7/PSP7
RD6/PSP6
RD5/PSP5
RD4/PSP4
RD3/PSP3
RD2/PSP2
RE2/CS/AN7
RE1/WR/AN6
RE0/RD/AN5
RB3/PGM
RD1/PSP1
RD0/PSP0
Verso_Mot
Clock_Mot
Ind_Mot_A0
Ind_Mot_A1
P_Spegnimento
P_Change_Mod
ICSP_Data
Display_D7
Display_D6
Display_D5
Display_D4
Display_Enable
Display_RS
Off
BR1
BR2
BR3
BR4
U2
MAX232
R1IN
T1IN
T1OUT
R1OUT
C1+
C1-
C2+
C2-
V+
V-
C3
1 u
C4
1 u
C5
1 u
C6
1 u
P1
Y1
8 Mhz
C1
22 p
C2
22 p
R1
10 K
R2
100
R3
100
D1
LED
D2
LED
Title
Scheda Madre - Micro e Seriale
Size
A
Document Number
2
Rev
0
Date:
Sunday, June 07, 2009
Sheet
1
of
2

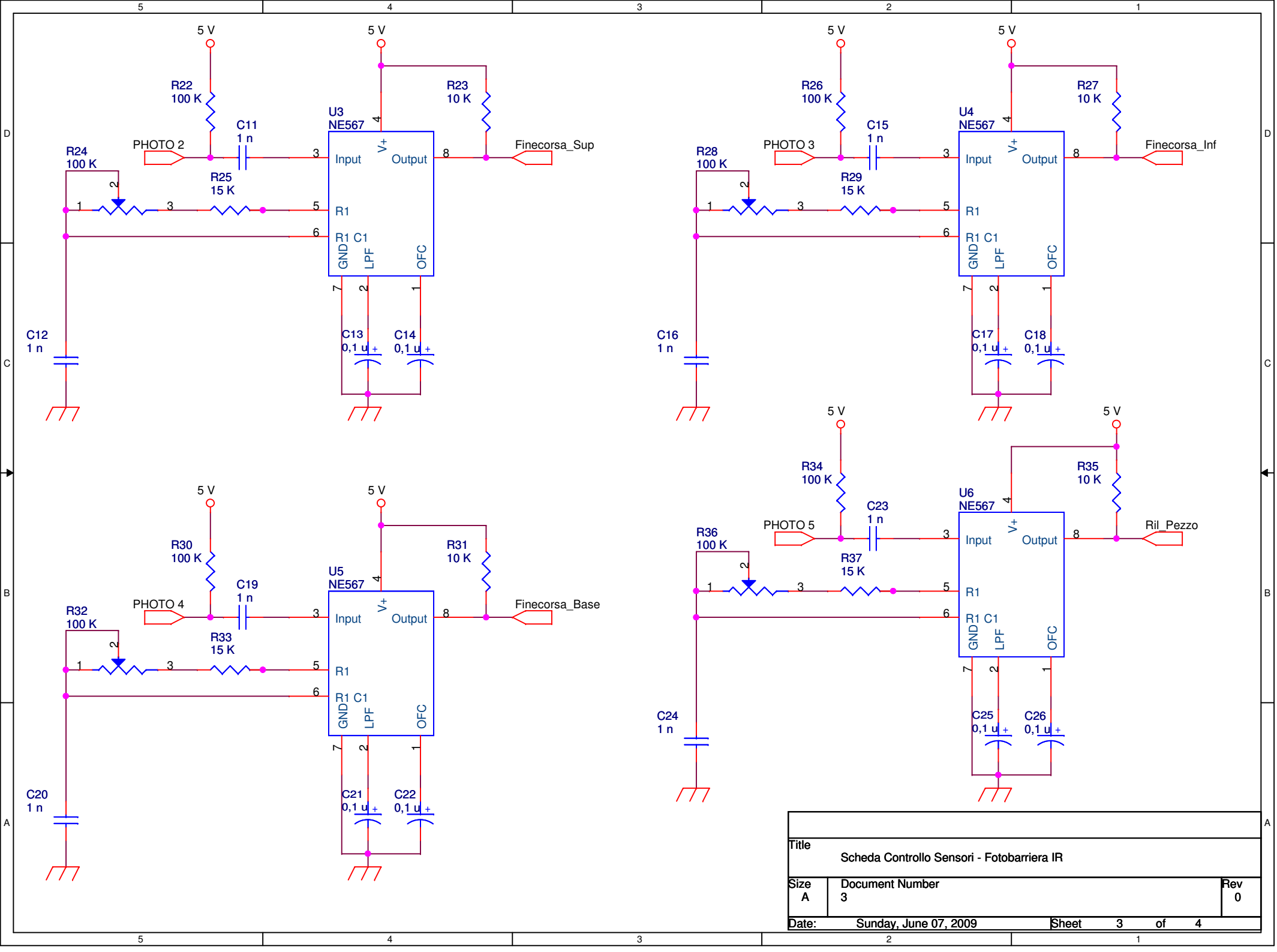

BR5
1
2
Temp_A
Temp_B
BR6
1
2
PS/2_Clock
PS/2_Data
5 V
R4
10 K
P_Spegnimento
SW1
C7
20 p
5 V
R5
10 K
P_Change_Mod
SW2
C8
20 p
BR7
1
2
3
Corrente_A
Corrente_B
Corrente_C
BR8
1
2
3
4
Verso_Mot
Clock_Mot
Ind_Mot_A0
Ind_Mot_A1
BR9
1
2
3
4
5
6
Pross_Pezzo
Finecorsa_Sup
Finecorsa_Inf
Finecorsa_Base
Ril_Pezzo
Pres_Pezzo
BR10
1
2
Off
BR11
1
2
3
4
5
ICSP_Vpp
5 V
ICSP_Clock
ICSP_Data
BR12
1
2
3
4
5
6
7
8
Display_D7
Display_D6
Display_D5
Display_D4
Display_Enable
Display_RS
5 V
5 V
D3
1N4148
R8
10 K
D4
1N4148
ICSP_Vpp
SW3
C11
10 u
VCC
R6
10 K
R7
10 K
JP1
DIN 5P
PS2_Data
1
2
3
4
5
5 V
PS2_Clock
5 V
Alimentazione 5V
1
2
C9
100 u
C10
100 n
Title
Scheda Madre - Connessioni
Size
A
Document Number
2
Rev
0
Date:
Sunday, June 07, 2009
Sheet
2
of
2



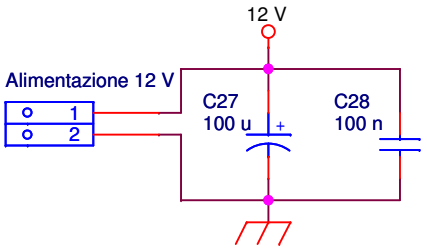
Title	Scheda Controllo Sensori - Trasmettitore IR 555	
Size A	Document Number 3	Rev 0
Date: Sunday, June 07, 2009	Sheet 1 of 4	

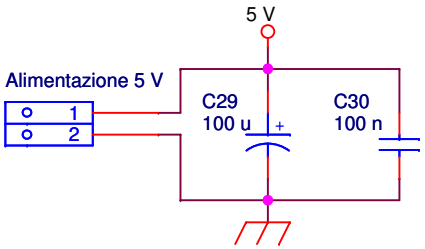


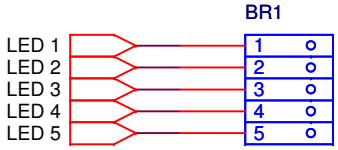
Title	Scheda Controllo Sensori - Sensore Di Prossimità Infrarosso	
Size A	Document Number 3	Rev 0
Date: Sunday, June 07, 2009	Sheet 2 of 4	

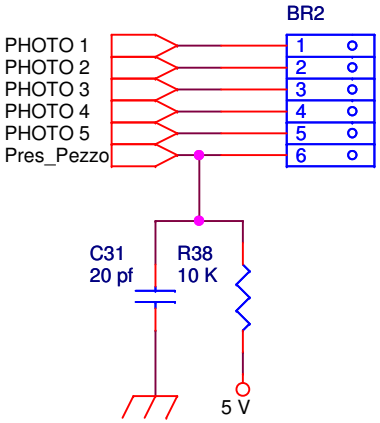


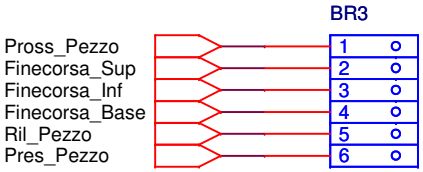
Title	Scheda Controllo Sensori - Fotobarriera IR	
Size A	Document Number 3	Rev 0
Date: Sunday, June 07, 2009	Sheet 3 of 4	



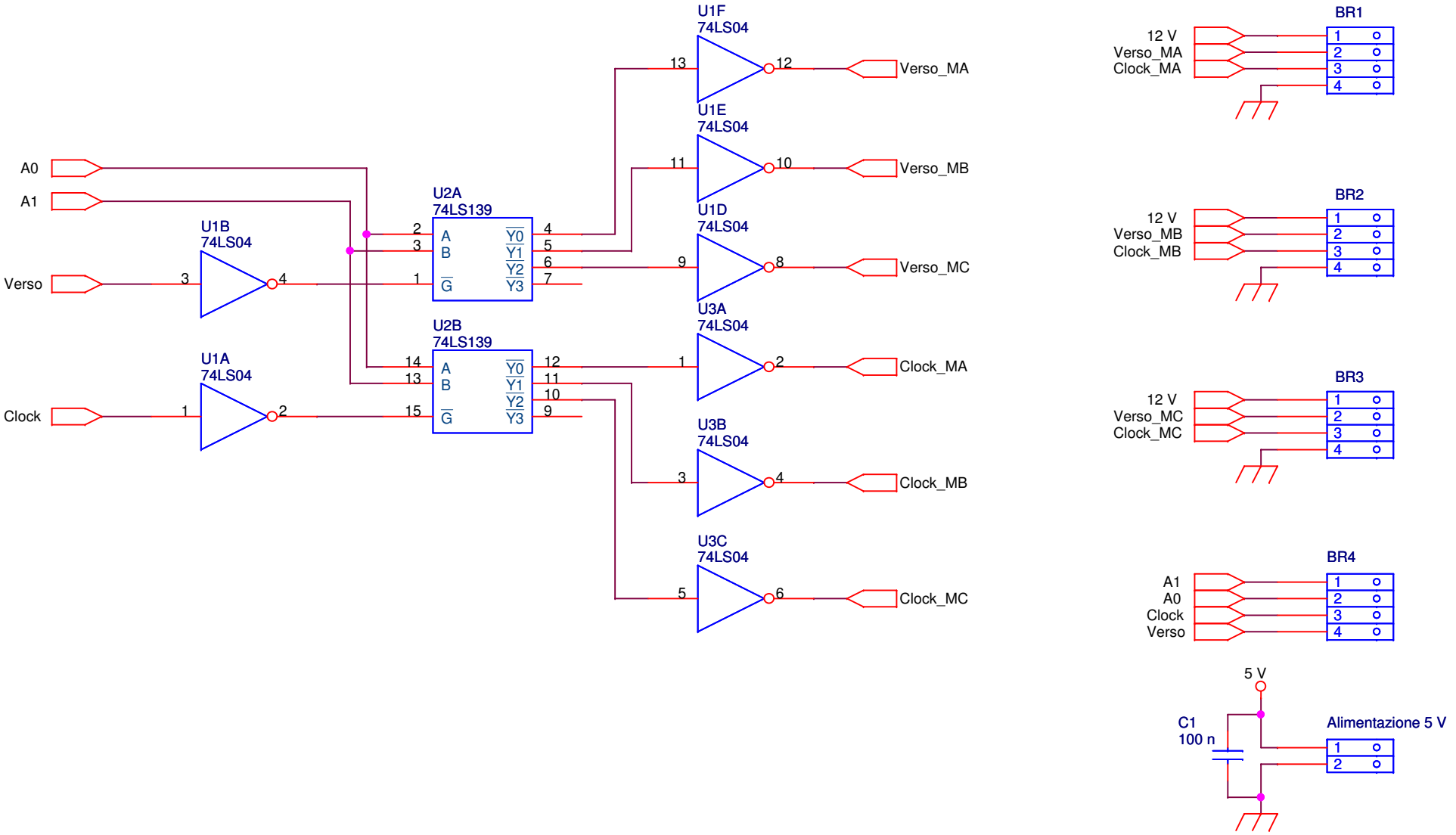




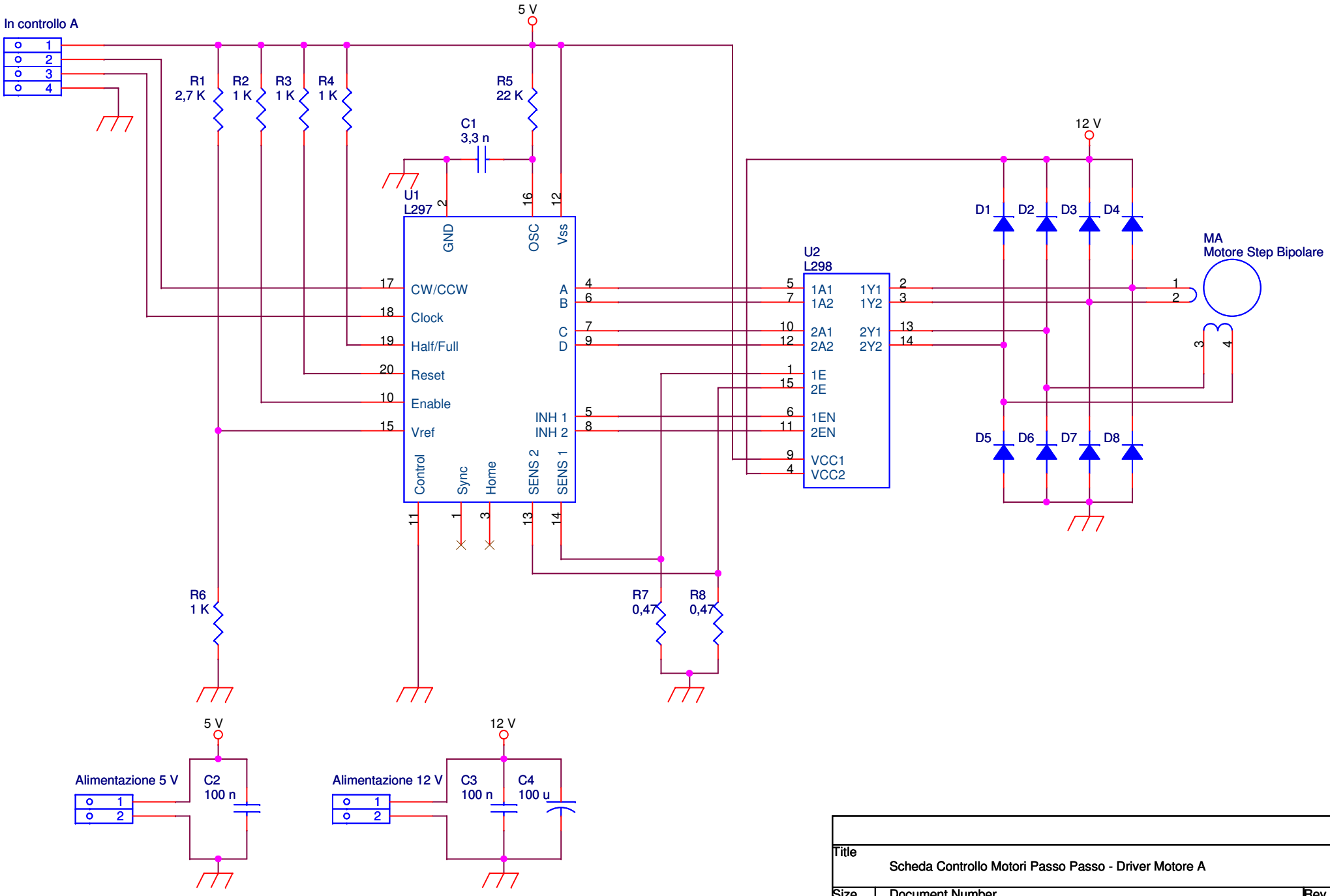




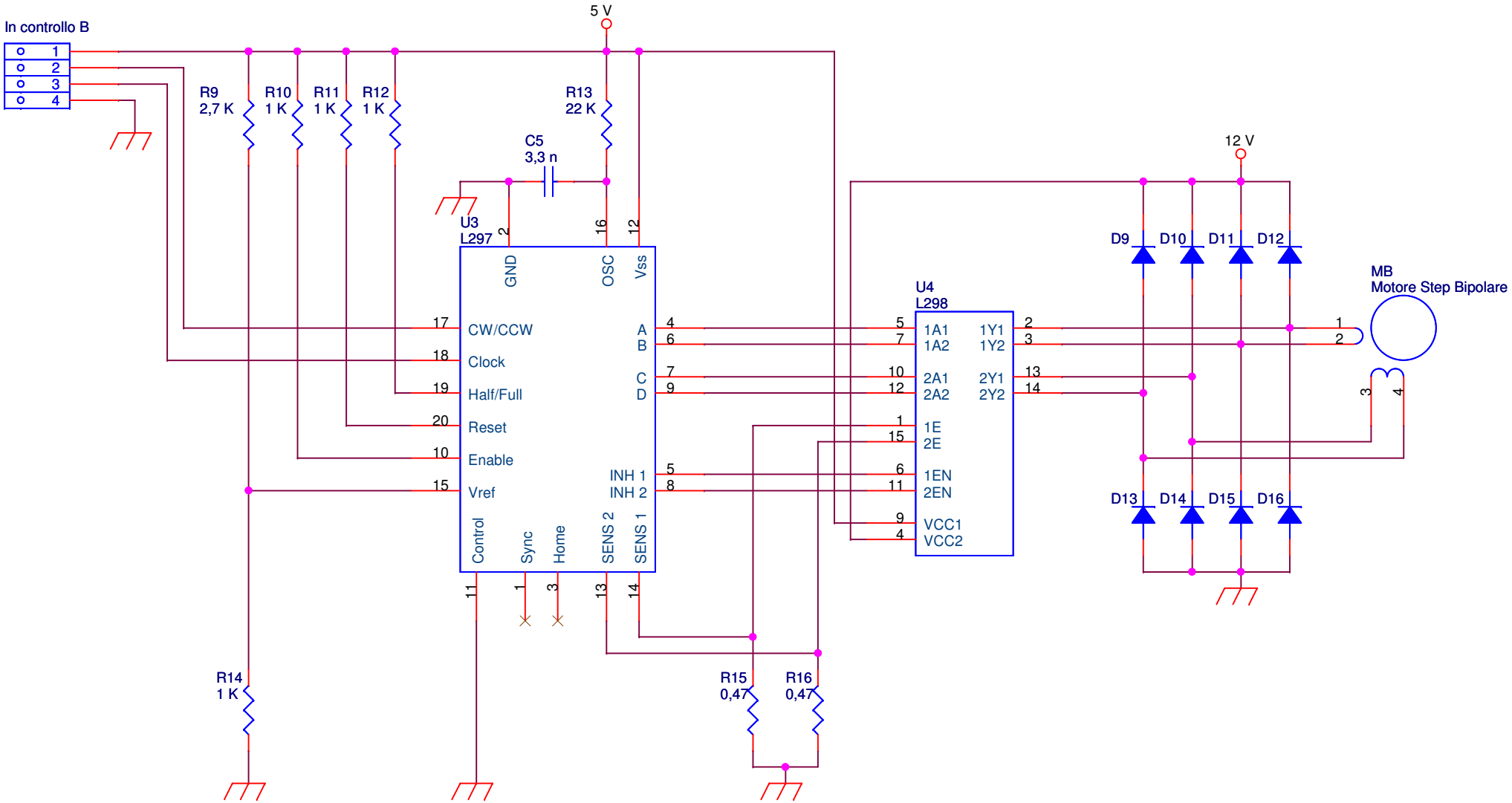
Title: Scheda Controllo Sensori - Connessioni		
Size: A	Document Number: 3	Rev: 0
Date: Sunday, June 07, 2009	Sheet 4 of 4	



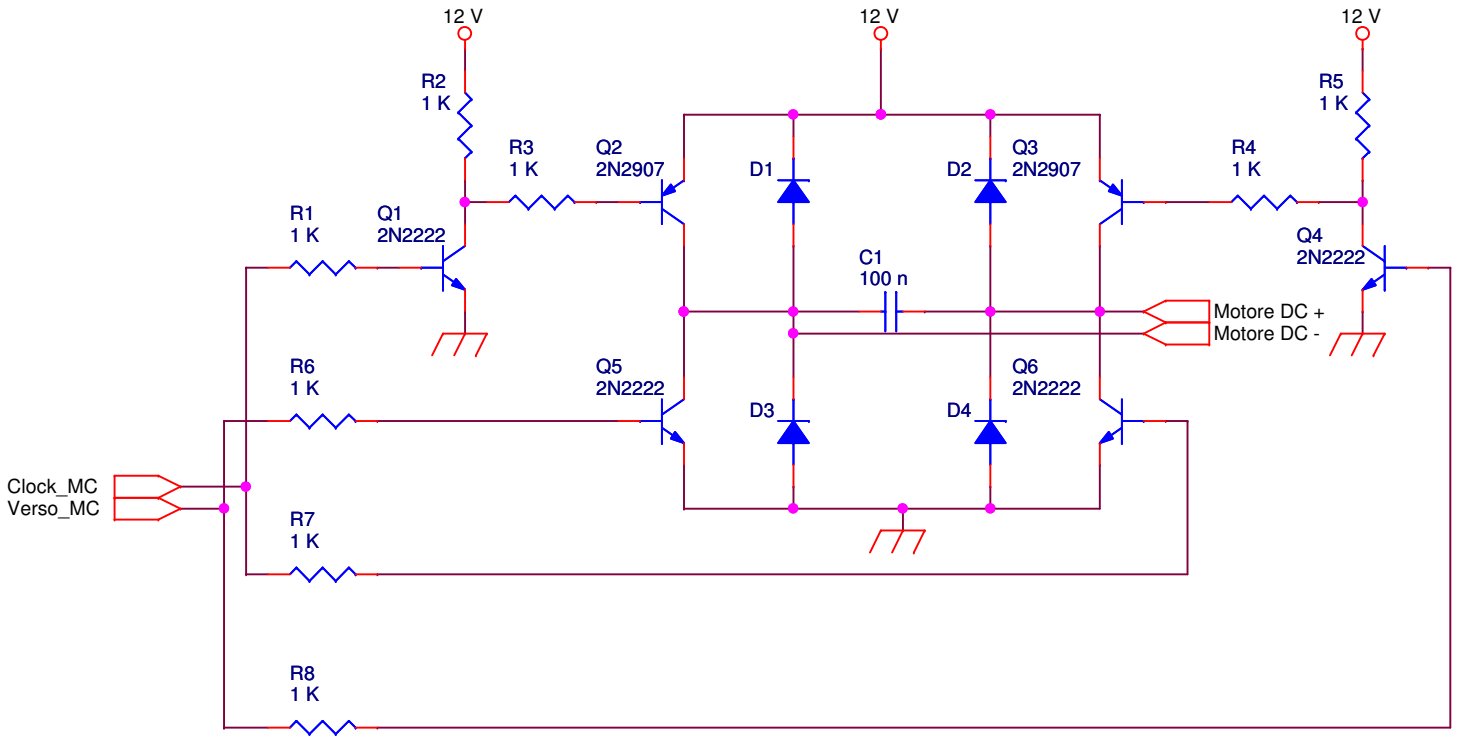
Title	Scheda Multiplexing Motori - Demultiplexer Per Segnali Di Controllo Motori	
Size A	Document Number 4	Rev 0
Date: Sunday, June 07, 2009	Sheet 1 of 1	

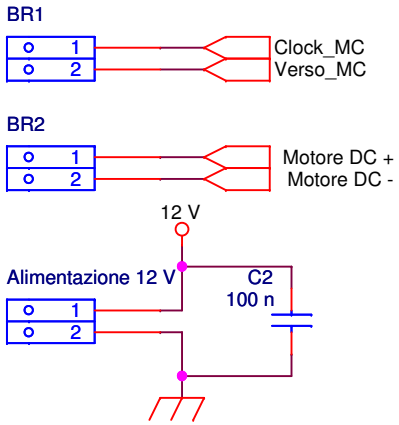


Title	Scheda Controllo Motori Passo Passo - Driver Motore A	
Size A	Document Number 5	Rev 0
Date: Sunday, June 07, 2009	Sheet 1 of 2	



Title	Scheda Controllo Motori Passo Passo - Driver B	
Size: A	Document Number: 5	Rev: 0
Date: Sunday, June 07, 2009	Sheet 2 of 2	





Title	Scheda Controllo Motore DC Pinza - Ponte Ad H	
Size A	Document Number 6	Rev 0
Date: Sunday, June 07, 2009	Sheet 1 of 1	

intro: GTP USB PIC PROGRAMMER (Open Source)

This work includes, GTP USB (not plus or lite) .

The schematic, photos and PCB have been developed by PICMASTERS based on some valuable works done before.

This programmer supports pic10F, 12F, 16C, 16F, 18F,24Cxx Eeprom.

Unfortunately, it works with only Winpic800 v.355.

We have succesfully tried it with some pics; PIC18F252, 18F2455, 18F2550, 18F2520, 16F84, 16F628 and 24C32 eeprom.

The best and fastest method of pic programming.

All you need (hex file, winpic800, schematic, PCB board in ARES) are included.

You must program pic18F2550 with hex file by a classical programmer.

Just leave a good fedback for us.

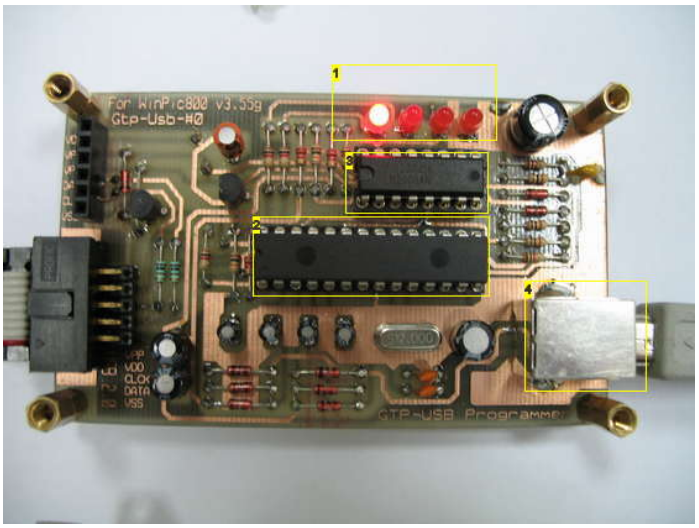


Image Notes
1. leds
2. pic 18f2550
3. ULN2003ANE
4. usb connector

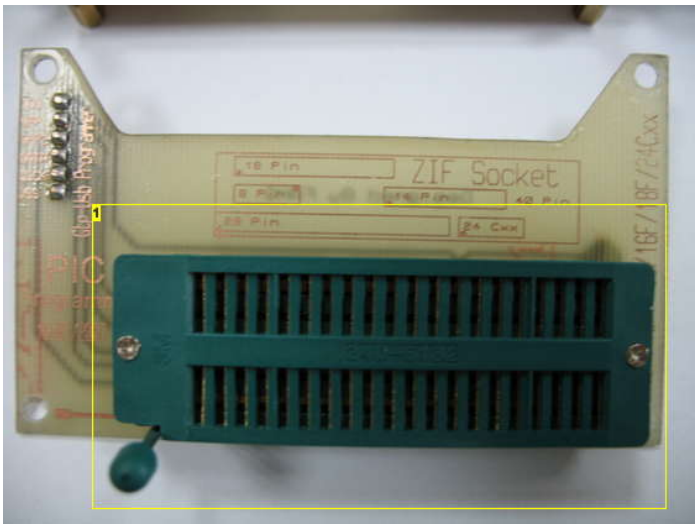
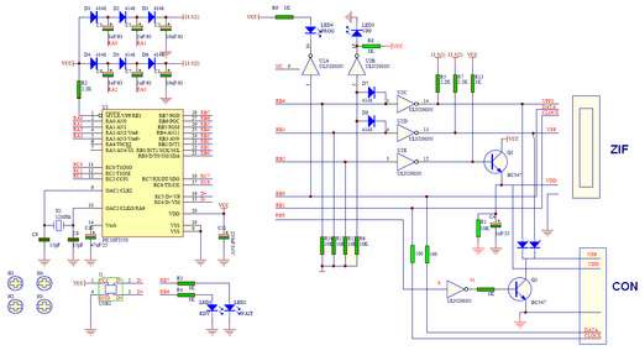
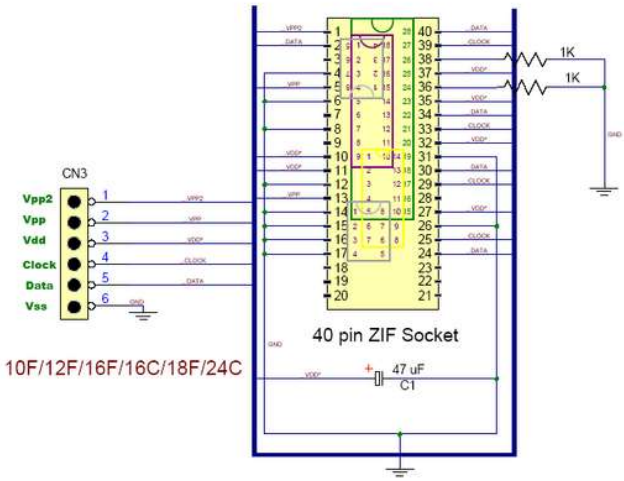


Image Notes
1. ZIF socket

step 1: circuit scheamatic

circuit schematic. You can see the type and value of the companents.

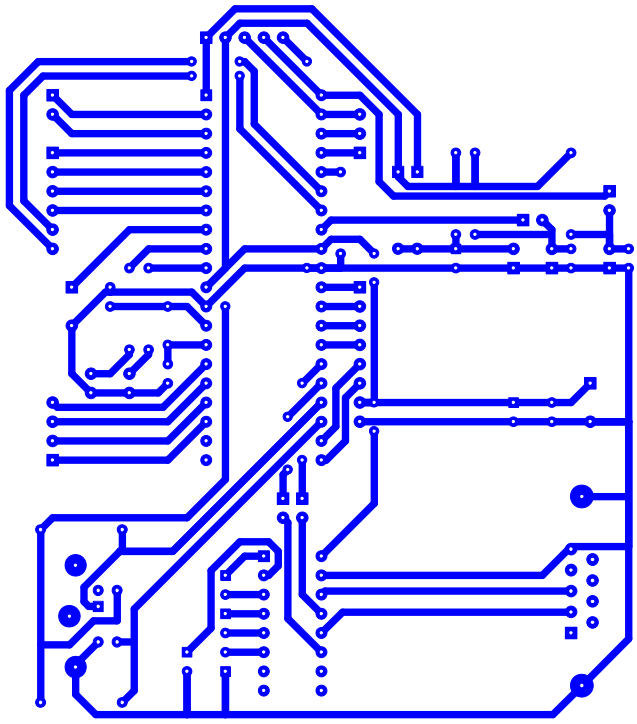


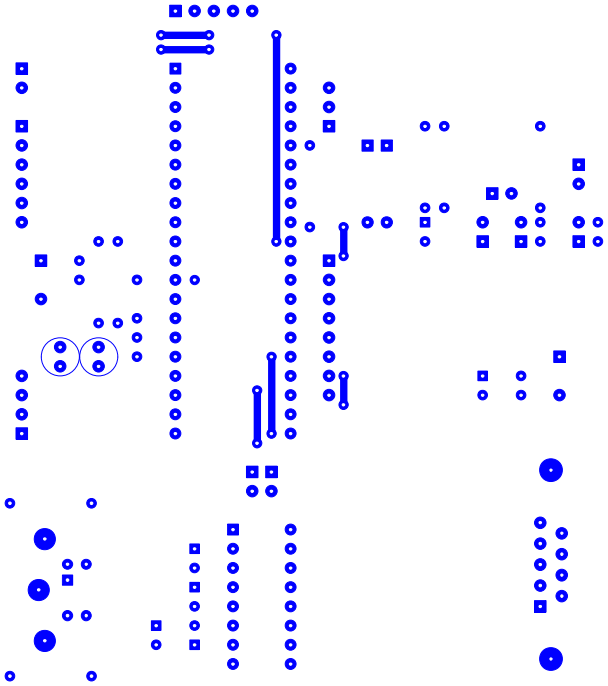


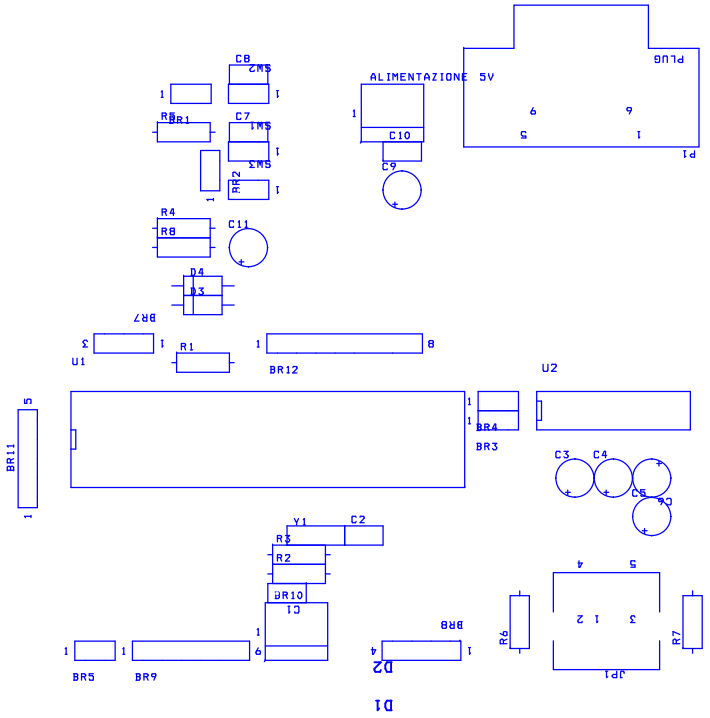
Schema Collegamenti Cavo Consolle-Braccio

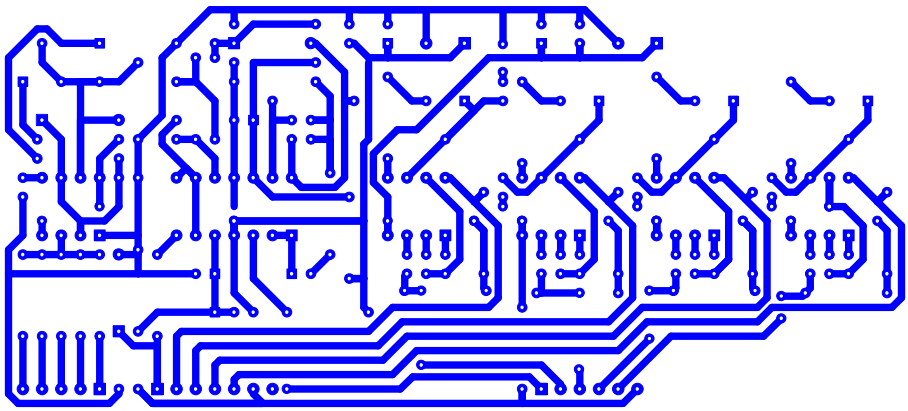
Connettore 25 Poli

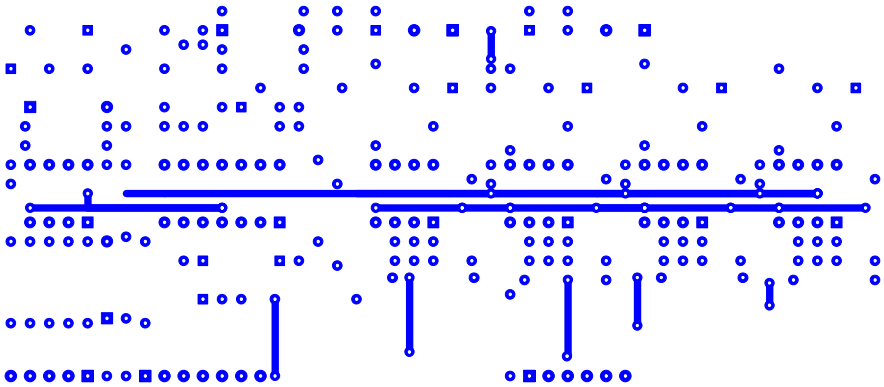
1	Uscita Motore DC Pinza a
2	Uscita Motore Dc Pinza b
3	Motore Step A - Motore Base
4	Motore Step A - Motore Base
5	Motore Step A - Motore Base
6	Motore Step A - Motore Base
7	Motore Step B - Motore Verticale
8	Motore Step B - Motore Verticale
9	Motore Step B - Motore Verticale
10	Motore Step B - Motore Verticale
11	Negativo
12	Negativo
13	+ 12 V
14	+ 12 V
15	Pres_Pezzo
16	Photo5 Ril_Pezzo
17	Photo4 Finecorsa_Base
18	Photo3 Finecorsa_Inf
19	Photo2 Finecorsa_Sup
20	Photo1 Sens.Prossimità IR Pinza (Rx)
21	Led1 Sens. Prossimità IR Pinza (Tx)
22	Led2 Finecorsa_Sup
23	Led3 Finecorsa_Inf
24	Led4 Finecorsa_Base
25	Led5 Ril_Pezzo
Carcassa	Cavo Di Terra

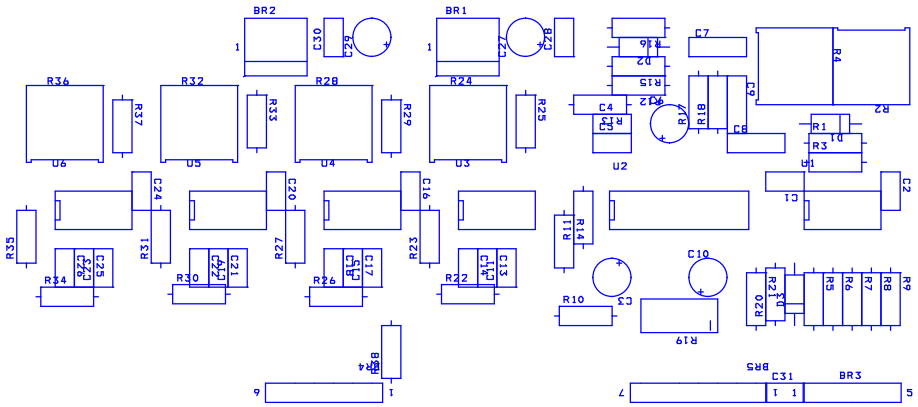


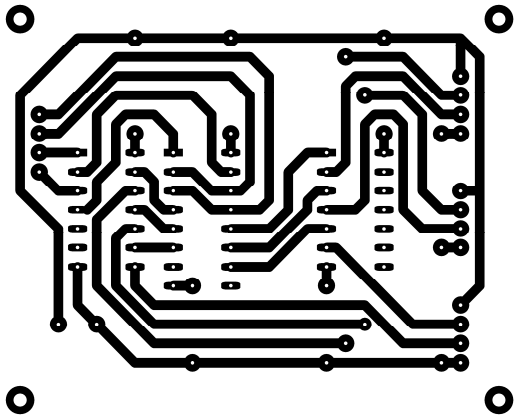


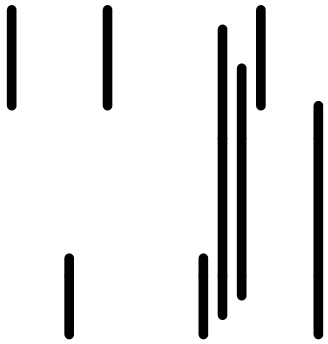

ALIMENTAZIONE 5V
PLUG
P1
C8
SW2
C7
SW1
SW3
BR1
BR2
R8
R4
C11
D4
D3
BR7
U1
R1
BR12
U2
BR4
BR3
BR11
C3
C4
C5
C6
Y1
C2
R3
R2
BR10
C1
BR8
BR5
BR9
R6
R7
JP1
D2
D1

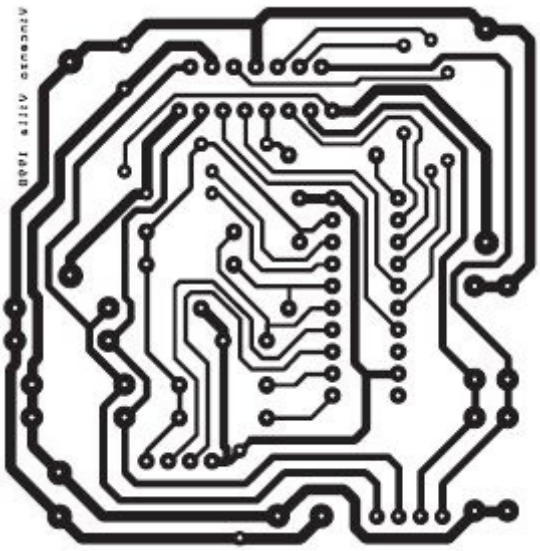


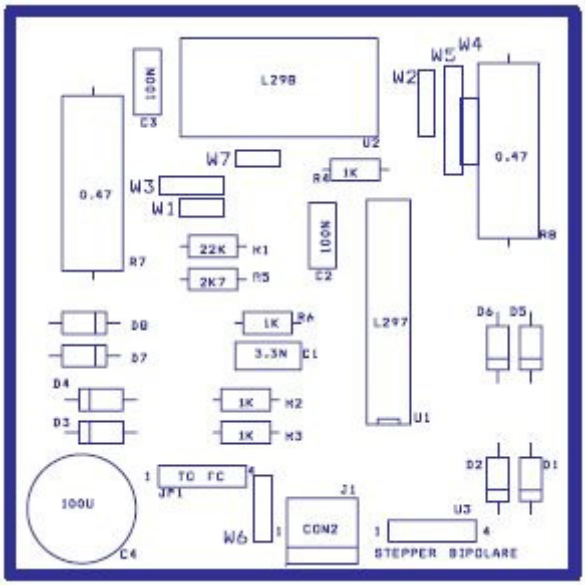


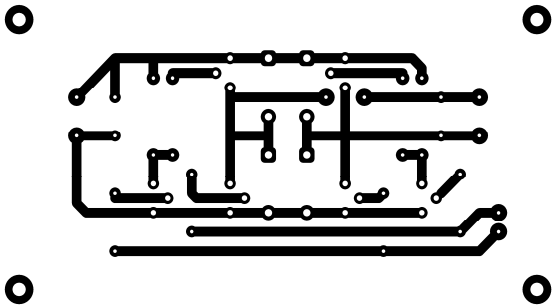

BR2
BR1
C30
C29
C27
C28
C7
R16
D2
R15
R14
R4
R36
R32
R28
R24
R37
R33
R29
R25
C4
C5
R17
R18
C9
C8
R2
R1
D1
R3
U6
U5
U4
U3
U2
C24
C20
C16
C1
C2
R35
R31
R27
R23
R11
R12
C10
C25
C21
C17
C13
C14
C18
C22
C26
C23
C19
C15
R34
R30
R26
R22
C3
R10
R19
R20
R21
R5
R6
R7
R8
R9
BR4
BR5
C31
BR3
6
1
7
5

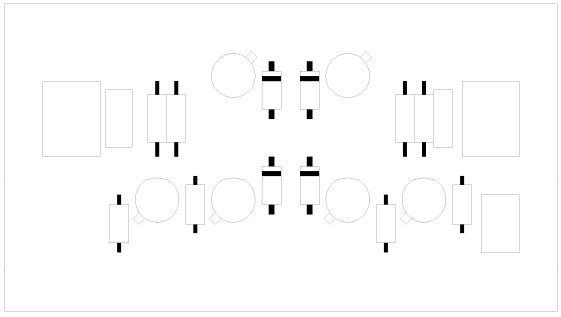







L298
100N
C3
0.47
R7
W7
W3
W1
U2
1K
R6
W2
W5
W4
0.47
R8
22K
R1
2K7
R5
100N
C2
L297
U1
D6
D5
D8
D7
1K
R4
3.3N
C1
D4
D3
1K
R2
1K
R3
D2
D1
100U
C4
TO FC
JP1
W6
J1
CON2
U3
STEPPER BIPOLARE





C:\Users\Andrea\Desktop\Progetto Tesina Braccio Robotizzato\Hardware\Schemi Orcad\Scheda Controllo Alimentazione\controllo alimentazione.bom

Scheda Controllo Alimentazione

Bill Of Materials

Item	Quantity	Reference	Part
1	4	C1,C2,C4,C5	1 n
2	4	C3,C6,C7,C8	100 n
3	1	D1	Diodo Veloce 1 ampere
4	1	D2	LED
5	1	F1	Fusibile X Ampere
6	2	L1,L2	10 mH
7	1	Off	Morsettiera 2 Posti
8	1	Q1	2N2222
9	1	RL1	Relay 200 V 2 Scambi
10	1	RL2	Relay 12 V 1 Scambio
11	4	R1,R2,R3,R4	2,2 K
12	1	R5	470
13	1	R6	1 K
14	1	SW1	Pulsante N.A. 220 V
15	2	220 V AC OUT,220 V AC IN	Morsettiera 3 Posti

C:\Users\Andrea\Desktop\Progetto Tesina Braccio Robotizzato\Hardware\Schemi Orcad\Scheda Madre PIC16F877\Schematico\scheda madre.bom

Scheda Madre

Bill Of Materials

Item	Quantity	Reference	Part
1	2	Alimentazione 5V,BR10	Morsettiera 2 Posti
2	6	BR1,BR2,BR3,BR4,BR5,BR6	Contatti Strip 2
3	1	BR7	Contatti Strip 3
4	1	BR8	Contatti Strip 4
5	1	BR9	Contatti Strip 6
6	1	BR11	Contatti Strip 5
7	1	BR12	Contatti Strip 8
8	2	C1,C2	22 p
9	4	C3,C4,C5,C6	1 u
10	2	C7,C8	20 p
11	1	C9	100 u
12	1	C10	100 n
13	1	C11	10 u
14	2	D1,D2	LED
15	2	D3,D4	1N4148
16	1	JP1	DIN 6P
17	1	P1	Connettore Maschio DB9
18	6	R1,R4,R5,R6,R7,R8	10 K
19	2	R2,R3	100
20	3	SW1,SW2,SW3	Pulsante N.A.
21	1	U1	PIC18F452
22	1	U2	MAX232
23	1	Y1	8 Mhz

Scheda Controllo Sensori

Bill Of Materials

Item	Quantity	Reference	Part
1	2	Alimentazione 5 V, Alimentazione 12 V	Morsettiera 2 Posti
2	2	BR2,BR3	Contatti Strip 6
3	1	BR1	Contatti Strip 5
4	10	C1,C11,C12,C15,C16, C19,C20,C23,C24	1 n
5	9	C2,C13,C14,C17,C18,C21, C22,C25,C26	0,1 u
6	2	C3,C10	10 u
7	1	C4,C5	1,8 n
8	1	C6	47 u
9		C7	1 u
10	2	C8	10 n
11	1	C9	20 n
12	2	C27,C29	100 u
13	2	C28,C30	100 n
14	1	C31	20 pf
15	2	D1,D2	1N4148
16	1	D3	Zener 5V
17	2	R1,R3	12 K
18	2	R2,R4	50 K
19	1	R5	100
20	5	R6,R7,R8,R9,R12	470
21	10	R10,R16,R22,R24,R26,R28, R30,R32,R34,R36	100 K
22	1	R11	8,2 K
23	1	R13	150 K
24	2	R14,R15	22 K
25	2	R17,R18	1 M
26	1	R19	500 K
27	1	R20	470 K
28	1	R21	1 K
29	5	R23,R27,R31,R35,R38	10 K
30	4	R25,R29,R33,R37	15 K
31	1	U1	NE555
32	1	U2	LM324
33	4	U3,U4,U5,U6	NE567

C:\Users\Andrea\Desktop\Progetto Tesina Braccio Robotizzato\Hardware\Schemi Orcad\Scheda Multiplexing Motori\Schematico\SCHEDA MULTIPLEXING MOTORI.BOM

Scheda Multiplexing Motori

Bill Of Materials

Item	Quantity	Reference	Part
1	1	Alimentazione 5 V	Morsettiera 2 Posti
2	4	BR1,BR2,BR3,BR4	Morsettiera 4 Posti
3	1	C1	100 n
4	2	U1,U3	74LS04
5	1	U2	74LS139

Scheda Controllo Motori Passo Passo

Bill Of Materials

Item	Quantity	Reference	Part
1	2	R5, R13	22 K
2	2	Alimentazione 5 V, Alimentazione 12 V	Morsettiera 2 Posti
3	2	C1,C5	3,3 n
4	2	C2,C3	100 n
5	1	C4	100 u
6	16	D1,D2,D3,D4,D5,D6,D7,D8, D9,D10,D11,D12,D13,D14, D15,D16	Diodo Veloce 2 Ampere
7	2	In controllo B, In controllo A	Contatti Strip 4
8	2	MB,MA	Motore Step Bipolare
9	2	R1,R9	2,7 K
10	8	R2,R3,R4,R6,R10,R11,R12, R14	1 K
11	4	R7,R8,R15,R16	0,47
12	2	U1,U3	L297
13	2	U2,U4	L298

Scheda Controllo Motore

Bill Of Materials

Item	Quantity	Reference	Part
1	3	BR1,BR2, Alimentazione 12 V	Morsettiera 2 Posti
2	2	C1,C2	100 n
3	4	D1,D2,D3,D4	Diodo veloce 1 ampere
4	4	Q1,Q4,Q5,Q6	2N2222
5	2	Q2,Q3	2N2907
6	8	R1,R2,R3,R4,R5,R6,R7,R8	1 K

Firmware

Su

Microcontrollore

```
1: //*** ANDREA AGLIETTI/ V ETE/ I.T.I.S. LEONARDO DA VINCI FIRENZE/ A.S. 2008 -
   2009 ***//
2: //********** ..:::ARMATRONICK SOURCE CODE - SCRITTO PER PIC18F452:::..
   ***********//
3:
4: //Prototipi di funzioni
5:
6: int checksum(void);
7: int crea_cheksum (char c[50]);
8: int fifoget( char *c );
9: void fifoput( char c );
10: void parser (char c[50],float val);
11: void gestiscicomando(void);
12: void deescape( char rxc );
13: void controllo_pacchetto (char c);
14: void crea_pacchetto (char c[50]);
15: void invia(void);
16:
17: void apri_pinza(void);
18: void chiudi_pinza(void);
19:
20: void Reset_base(void);
21: void Reset_vert(void);
22: void Reset_pinza(void);
23:
24: void attesa_pezzo(void);
25: void Ruota_alto(void);
26: void calcolo_rilascio(void);
27: void calcolo_presa(void);
28: void switch_presa1(void);
29: void switch_presa2(void);
30: void switch_presa3(void);
31: void switch_presa4(void);
32: void Ruota_basso(void);
33: void ruota_base (char verso, int passi_base);
34: void repos(void);
35:
36: void tastiera(void);
37: void parserps2(void);
38: void gradi(void);
39: void velvert(void);
40: void velbase(void);
41:
42: void display(char clear, char riga, char colonna, char dato[16]);
43: void gestione_errori(void);
44: void Arresto(void);
45:
46: void setta_registri(void);
47: void Reset_variabili(void);
48: void interrupt(void);
49:
50: //Definizione define
51:
52: #define STX 2                                   //Carattere STX - seriale
53: #define ETX 3                                   //Carattere ETX - seriale
54: #define ESC 27                                  //Carattere ESC - seriale
55: #define COMMAND_MAX_SIZE 50                     //Dimensione Max comando - seriale
56: #define Corr_Pezzo 100                          //Fattore Correttivo
57: #define Correttivo_pinza 200                    //Fattore correttivo per la
   riapertura della pinza
58:
59: //Definizione Variabili Globali
```

```
60: //Variabili formato int
61:
62: int wasesc = 0;
63: int ricezione = 0;
64: int ncr = 0;
65: int idx_r = 0;                              //Indice lettura fifo_rx
66: int idx_w = 0;                              //Indice scrittura  fifo rx
67: int dim = 0;
68: volatile int  txsize = 0;                   //Dimensione buffer trasmissione
69: volatile int  txpos = 0;                    //Indice buffer trasmissione
70:
71: int Finecorsa = 0;                          //Variabile che memorizza la
    posizione di finecorsa verticale verso il basso
72: int Finecorsa_mem = 0;                      //Memorizza il valore iniziale di
    Finecorsa
73: int Posizione_vert = 0;                     //Posizione del motore verticale
74: int Limiteinf = 290;                        //Numero di passi del motore
    verticale per arrivare alla posizione limite verso il basso
75: int Posizione_base = 0;                     //Posizione della base rotante del
    braccio in passi
76: int Altezza_Pezzo = 0;                      //Altezza del pezzo in passi
77: int i = 0;                                  //Variabile usata come indice nei
    cicli for
78: int Width_pezzo = 0;                        //Larghezza del pezzo in ms
79: int passibase = 0;                          //Passi di cui deve ruotare la base
80: int contpinza = 0;                          //Variabile contatore utilizzata
    per l'apertura della pinza
81:
82: //Variabili formato char
83:
84: char rxfifo[COMMAND_MAX_SIZE];              //Fifo in ricezione
85: char comando[COMMAND_MAX_SIZE];             //Comando ricevuto da seriale
86: char txbuff[COMMAND_MAX_SIZE];             //Buffer trasmissione seriale
87: char flag = 0;                              //Variabile flag
88: char datops2[16] = "";                      //Dato ricevuto dalla tastiera PS/2
89: char cont = 0;                              //Variabile indice
90: char nopezzo = 0;                           //Variabile Flag - E' posta a 1 se
    il braccio dopo aver ruotato in basso non ha trovato il pezzo
91: char repos_flag = 0;                        //Variabile Flag - Se posta a 1
    indica che si è eseguito almeno un riposizionamento nella funzione repos
92: char Ril_pezzo = 0;                         //Variabile Flag - Viene posta a 1
    quando viene stretto un pezzo nella pinza e di nuovo a 0 quando questa viene
    riaperta
93: char vel_sel_vert = 2;                      //Serve a selezionare quale
    velocità verticale utilizzare (1-Lento, 2-Medio, 3-Veloce), di default è
    selezionata la 2-media
94: char vel_sel_base = 2;                      //Serve a selezionare quale
    velocità per la base utilizzare (1-Lento, 2-Medio, 3-Veloce) di default è
    selezionata la 2-media
95: char vel_vert = 4;                          //Velocità verticale (indica la
    durata di un passo in ms)
96: char vel_base = 4;                          //Velocità rotazione base (indica
    la durata di un passo in ms)
97: char ivel = 0;                              //Variabile di conteggio utilizzata
    nei cicli for per determinare il clock dei motori passo passo
98: char key, special, down;                    //Variabili per la gestione PS/2
99: char error_code = 0;                        //Codice errore
100: char select = 0;                           //Seleziona la modalità (0=Auto,
    2=Tastiera, 3=PC)
101: char sel_savestate = 1;                    //Salva la modalità in caso di
    errore
102: char txt[4];                               //Variabile per la visualizzazione
    del codice errore sul display
```

```
103: char Spegni = 0;                              //Entra nella modalità di
     spegnimento
104: char Tornatosu = 0;                           //Variabile flag indica che il
     braccio è tornato su dopo aver preso il pezzo
105: char Versog = 0;                               //Setta il verso di rotazione
     della base
106: char Esci = 0;                                //Serve per uscire da un ciclo
     all'interno della funzione setta_base
107: char errore = 0;                              //Indica che si è verificato un
     errore se posta a 1
108: char flag_preso_base = 0;                     //Indica che il pezzo è stato preso
     sulla base se posta a 1
109: char Out_forzato = 0;                         //Variabile flag - consente di
     uscire immediatamente dalla modalità auto in caso di richiesta di spegnimento
     all'avvio
110: char Out_funz = 0;                            //Variabile flag - consente di
     uscire dalle funzioni se posta a 1
111: char Flag_Return = 0;                         //Variabile flag -
112: char outint = 1;                              //Variabile flag - viene posta a 1
     se si è richiesto un cambio di modalità
113: char Correttivo_rilascio = 7;                 //Fattore correttivo per rilascio
     pezzo fuori dalla base
114: char Corr_mecc = 0;
115: char Corr_ril_sx = 0;
116:
117: //*************************************************************************
     *****//
118: //***************************** GESTIONE SERIALE RS 232
     ****************************//
119: //*************************************************************************
     *****//
120:
121: void fifoput( char c )                         //Scrittura fifo
122: {
123: if (Out_funz==0)
124:  {                                             //Trova il prossimo byte libero, se
     c'è'
125:   char idx_w_tmp = idx_w;
126:   idx_w_tmp++;
127:   if( idx_w_tmp==COMMAND_MAX_SIZE ) idx_w_tmp=0;          //Sono in fondo, devo
     tornare all'inizio
128:   if( idx_w_tmp==idx_r ) return;              //La fifo e' piena, non scrivo
     niente
129:   idx_w = idx_w_tmp;                          //Aggiorna l'indice e scrivi davvero
130:   rxfifo[idx_w] = c;
131:  }
132: }
133:
134: int fifoget( char *c )                         //Lettura fifo
135: {
136: if (Out_funz==0)
137:  {                                             //Trova il prossimo byte scritto,
     se c'è'
138:   if( idx_r==idx_w ) return 0;                //La fifo e' vuota, ritorno 0
139:   idx_r++;
140:   if( idx_r==COMMAND_MAX_SIZE ) idx_r=0;                  //Sono in fondo, devo
     tornare all'inizio
141:   *c = rxfifo[idx_r];                         //Leggi
142:   return 1;                                   //Letto con successo
143:  }
144: }
145:
```

```
146: void controllo_pacchetto (char c)           //Gestione del byte ricevuto dalla
     seriale
147: {
148: if (Out_funz==0)
149:  {
150:   switch( ricezione )
151:   {
152:     case 0:                                 //Aspetto STX
153:       if (c==STX)
154:       {
155:         ricezione = 1;                      //Aspetto ETX
156:         ncr = 0;                            //Riazzero indice comando (nuovo
     comando)
157:       }
158:     break;
159:     case 1:                                 //Aspetto ETX
160:       if (c==ETX)
161:       {
162:         ricezione=0;
163:         if( checksum() )                    //Se il checksum torno, vai col
     comando
164:         {
165:           gestiscicomando();
166:         }
167:         else
168:         {
169:           crea_pacchetto ("ck");           //Errore cheksum
170:           invia();
171:         }
172:       }
173:       else
174:       {
175:         deescape( c );                      //Deescape e mette nel comando il
     nuovo carattere
176:       }
177:     break;
178:   }
179:  }
180: }
181:
182: void deescape( char rxc )                   //Viene eseguita l'operazione di
     deecape sul byte ricevuto dalla seriale
183: {
184: if (Out_funz==0)
185:  {
186:   if( rxc==ESC )
187:   {
188:     wasesc = 1;
189:   }
190:   else
191:   {
192:     if( wasesc )
193:     {
194:       rxc -= 64;
195:       wasesc = 0;
196:     }
197:     comando[ncr] = rxc;
198:     ncr++;
199:     if( ncr >= COMMAND_MAX_SIZE )
200:     {
201:       ncr = 0;
202:     }
```

```
203:    }
204:   }
205: }
206:
207: void gestiscicomando(void)                    //Estrae il valore contenuto nel
     pacchetto (val) e il comando da eseguire
208: {                                             //Invia i valori estratti al parser
209: if (Out_funz==0)
210:  {
211:   char appoggio [5];
212:   float val=0;
213:   appoggio[0]=comando[ncr-5];
214:   appoggio[1]=comando[ncr-4];
215:   appoggio[2]=comando[ncr-3];
216:   appoggio[3]=comando[ncr-2];
217:   appoggio[4]='\0';
218:   val = atof (appoggio);
219:   comando [ncr-5]='\0';
220:   parser (comando,val);
221:   ncr = 0;
222:  }
223: }
224:
225: void parser (char c[COMMAND_MAX_SIZE],float val)          //Interprete dei
     comandi ricevuti dalla seriale
226: {
227: if (Out_funz==0)
228:  {
229:   if (strcmp (c,"SU")==0)                     //Richiama la funzione ruotasu() e
     il braccio si porta in posizione_vert 0
230:   {
231:    crea_pacchetto ("Ok");
232:    invia();
233:    PORTE.F2=1;
234:    delay_ms(500);
235:    PORTE.F2=0;
236:    ruota_alto();
237:   }
238:   else if (strcmp (c,"GIU")==0)               //Richiama la funziona rutoagiu() e
     il braccio si porta giu
239:   {
240:    crea_pacchetto ("Ok");
241:    invia();
242:    PORTE.F2=1;
243:    delay_ms(500);
244:    PORTE.F2=0;
245:    ruota_basso();
246:   }
247:   else if (strcmp (c,"DX")==0)               //Richiama la funzione ruotabase() e
     ruota a DX di passibase=val
248:   {                                           //val viene inviato dal PC e contine
     il numero di gradi di cui ruotare
249:    crea_pacchetto ("Ok");
250:    invia();
251:    PORTE.F2=1;
252:    delay_ms(500);
253:    PORTE.F2=0;
254:    passibase=((val*85)/9);
255:    ruota_base(1,passibase);
256:   }
257:   else if (strcmp (c,"SX")==0)               //Richiama la funzione ruotabase() e
     ruota a SX di passibase=val
```

```
258:   {                                        //val viene inviato dal PC e contine
     il numero di gradi di cui ruotare
259:    crea_pacchetto ("Ok");
260:    invia();
261:    PORTE.F2=1;
262:    delay_ms(500);
263:    PORTE.F2=0;
264:    passibase=((val*85)/9);
265:    ruota_base(0,passibase);
266:   }
267:   else if (strcmp (c,"Chiudi")==0)         //Richiama la funzione
     chiudi_pinza() e chiude la pinza
268:   {
269:    crea_pacchetto ("Ok");
270:    invia();
271:    PORTE.F2=1;
272:    delay_ms(500);
273:    PORTE.F2=0;
274:    chiudi_pinza();
275:   }
276:   else if (strcmp (c,"Apri")==0)           //Richiama la funzione apri_pinza()
     e apre la pinza
277:   {
278:    crea_pacchetto ("Ok");
279:    invia();
280:    PORTE.F2=1;
281:    delay_ms(500);
282:    PORTE.F2=0;
283:    apri_pinza();
284:   }
285:   else if (strcmp (c,"Velvert")==0)        //Imposta la velocità per il motore
     verticale
286:   {
287:    crea_pacchetto ("Ok");
288:    invia();
289:    vel_sel_vert=val;                       //vel_sel_vert è uguale a val
     inviato dal pc e contiene il numero di velocità da selezionare (1,2,3)
290:    PORTE.F2=1;
291:    delay_ms(500);
292:    PORTE.F2=0;
293:    switch (vel_sel_vert)                   //Switch in base a vel_sel_vert,
     seleziona la velocità desiderata in base a vel_sel_vert
294:    {
295:     case 1:                                //Imposta velocità minima e
     visualizza la velocità selezionata
296:      vel_vert=7;
297:      PORTE.F2=1;
298:      delay_ms(500);
299:      PORTE.F2=0;
300:      display(0,2,1,"Vel Vert Minima");
301:      delay_ms(1500);
302:     break;
303:
304:     case 2:                                //Imposta velocità media e
     visualizza la velocità selezionata
305:      vel_vert=5;
306:      PORTE.F2=1;
307:      delay_ms(500);
308:      PORTE.F2=0;
309:      display(0,2,1,"Vel Vert Media");
310:      delay_ms(1500);
311:     break;
```

```
312:
313:     case 3:                                  //Imposta velocità massima e
     visualizza la velocità selezionata
314:      vel_vert=3;
315:      PORTE.F2=1;
316:      delay_ms(500);
317:      PORTE.F2=0;
318:      display(0,2,1,"Vel Vert Max");
319:      delay_ms(1500);
320:     break;
321:    }
322:   }
323:   else if (strcmp (c,"Velbase")==0)        //Imposta la velocità per il motore
     della base
324:   {
325:     crea_pacchetto ("Ok");
326:     invia();
327:     vel_sel_base=val;                      //vel_sel_base è uguale a val
     inviato dal pc e contiene il numero di velocità da selezionare (1,2,3)
328:     switch (vel_sel_base)                  //Switch in base a vel_sel_base,
     seleziona la velocità desiderata in base a vel_sel_base
329:     {
330:      case 1:                               //Imposta velocità minima e
     visualizza la velocità selezionata
331:       vel_base=8;
332:       PORTE.F2=1;
333:       delay_ms(500);
334:       PORTE.F2=0;
335:      display(0,2,1,"Vel Base Minima");
336:      delay_ms(1500);
337:      break;
338:
339:      case 2:                               //Imposta velocità media e
     visualizza la velocità selezionata
340:       vel_base=4;
341:       PORTE.F2=1;
342:       delay_ms(500);
343:       PORTE.F2=0;
344:      display(0,2,1,"Vel Base Media");
345:      delay_ms(1500);
346:      break;
347:
348:      case 3:                               //Imposta velocità massima e
     visualizza la velocità selezionata
349:       vel_base=2;
350:       PORTE.F2=1;
351:       delay_ms(500);
352:       PORTE.F2=0;
353:      display(0,2,1,"Vel Base Max");
354:      delay_ms(1500);
355:      break;
356:     }
357:    }
358:    else if (strcmp (c,"repos")==0)         //Richiama la funzione repos() e
     riposiziona selettivamente la meccanica
359:    {
360:     crea_pacchetto ("Ok");
361:     invia();
362:     PORTE.F2=1;
363:     delay_ms(500);
364:     PORTE.F2=0;
365:     outint=1;
```

```
366:     repos();
367:     outint=0;
368:    }
369:    else if (strcmp (c,"setall")==0)       //Richiama tutte le funzioni per il
    reset e resetta tutto (togliere manualmente pezzo)
370:    {
371:     crea_pacchetto ("Ok");
372:     invia();
373:     PORTE.F2=1;
374:     delay_ms(500);
375:     PORTE.F2=0;
376:     reset_vert();
377:     reset_base();
378:     reset_pinza();
379:     reset_variabili();
380:    }
381:    else if (strcmp (c,"setbase")==0)      //Richiama la funzione per il reset
    della base (togliere manualmente pezzo)
382:    {
383:     crea_pacchetto ("Ok");
384:     invia();
385:     PORTE.F2=1;
386:     delay_ms(500);
387:     PORTE.F2=0;
388:     reset_base();
389:     reset_variabili();
390:    }
391:    else if (strcmp (c,"setvert")==0)      //Richiama la funzione per il reset
    verticale (togliere manualmente pezzo)
392:    {
393:     crea_pacchetto ("Ok");
394:     invia();
395:     PORTE.F2=1;
396:     delay_ms(500);
397:     PORTE.F2=0;
398:     reset_vert();
399:     reset_variabili();
400:    }
401:    else if (strcmp (c,"setpinza")==0)     //Richiama la funzione per il reset
    della pinza (togliere manualmente pezzo)
402:    {
403:     crea_pacchetto ("Ok");
404:     invia();
405:     PORTE.F2=1;
406:     delay_ms(500);
407:     PORTE.F2=0;
408:     reset_pinza();
409:     reset_variabili();
410:    }
411:    else if (strcmp (c,"Spegni")==0)      //Imposta le variabili flag Spegni e
    Out_funz a 1
412:    {                                     //consentendo di uscire dalle funzioni
    e entrare nella modalità di spegnimento
413:     crea_pacchetto ("Ok");
414:     invia();
415:     Spegni=1;
416:     Out_Funz=1;
417:     PORTE.F2=1;
418:     PORTE.F1=0;
419:    }
420:
421:   else
```

```
422:   {
423:    crea_pacchetto ("EC");                //Comando non esistente
424:    invia();
425:   }
426:  }
427: }
428:
429: int checksum(void)                        //Calcola il checksum del pacchettto
     ricevuto
430: {
431: if (Out_funz==0)
432:  {
433:   int sr;                                //Somma ricevuta
434:   int sc;                                //Somma calcolata
435:   int i;
436:   sr = comando[ncr-1];                   //L'ultimo byte del comando è il
     checksum
437:                                          //Calcolo il checksum su tutti, tranne
     l'ultimo
438:   sc = 0;
439:   for(i=0;i<ncr-1;i++)
440:   {
441:     sc = sc + comando[i];
442:   }
443:
444:   sc = sc % 256;                         //Se la somma calcolta è = alla somma
     ricevuta il pacchetto è integro
445:   if (sc == sr )
446:     return (1);
447:   else
448:     return (0);
449:  }
450: }
451:
452: void crea_pacchetto (char c[COMMAND_MAX_SIZE])        //viene creato il
     pacchetto da inviare tramite seriale
453: {
454:   int ck;
455:   char ck_tx,appoggio[2];
456:   ck = crea_cheksum (c);
457:   ck_tx = ck;
458:   txbuff[1]='\0';
459:   txbuff[0]=STX;
460:   strcat(txbuff,c);
461:   appoggio[0]=ck_tx;
462:   appoggio[1]=ETX;
463:   appoggio[2]='\0';
464:   strcat(txbuff,appoggio);
465: }
466:
467: int crea_cheksum (char c[COMMAND_MAX_SIZE])           //Calcola il cheksum del
     pacchetto da inviare tramite seriale
468: {
469:   int sc,i,size;
470:   sc = 0;
471:   size= strlen (c);
472:   for (i=0;i<size;i++)
473:   {
474:     sc = sc + c[i];
475:   }
476:   sc = sc % 256;
477:   return sc;
```

```
478: }
479:
480: void invia(void)
481: {
482:  PIE1.RCIE = 0;                          //Attiva l'interrupt di ricezione USART
483:  dim=strlen(txbuff);
484:  for (i=0;i<dim;i++)
485:  {
486:   Usart_Write(txbuff[i]);
487:  }
488:  PIE1.RCIE = 1;                           //Attiva l'interrupt di ricezione USART
489: }
490:
491: //*************************************************************************
     *****//
492: //***************************** GESTIONE TASTIERA PS/2
     ******************************//
493: //*************************************************************************
     *****//
494:
495: void tastiera(void)                              //Funzione per la gestione della
     tastiera PS/2
496: {
497: if (Out_funz==0)
498: {
499:  if (Ps2_Key_Read(&key, &special, &down))     //E' stato premuto un tasto
500:  {
501:   if (down && special && key==13)             //Se è stato premuto INVIO
     richiama il parserps2 e resetta le variabili
502:   {
503:    parserps2();
504:    display(1,1,1,"Modalita PS/2");
505:    display(0,1,1,"");
506:    Lcd_Cmd(LCD_SECOND_ROW);
507:    Lcd_Cmd(LCD_BLINK_CURSOR_ON);
508:    for (i=0; i<16;i++)
509:    {
510:     datops2[i]=0;
511:    }
512:    cont=0;
513:    down=0;
514:   }
515:   else if (down && special && key==16)        //Se è stato premuto backspace
     cancella un carattere
516:   {                                           //e aggiorna il datops2 e il
     cursore
517:    if (cont==0)
518:    {
519:     cont=0;
520:    }
521:    else
522:    {
523:     cont--;
524:     Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);
525:     Lcd_Out(2,cont+1," ");
526:     Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);
527:    }
528:    datops2[cont]=0;
529:   }
530:   if (down && special && key==18)             //Se è stato premuto CANC cancella
     tutto ciò che è stato inserito
531:   {                                           //e resetta datops2 e cursore
     riportandolo al primo carattere
```

```
532:    display(1,1,1,"Modalita PS/2");           //della seconda riga
533:    Lcd_Cmd(LCD_SECOND_ROW);
534:    Lcd_Cmd(LCD_BLINK_CURSOR_ON);
535:    for (i=0; i<16;i++)
536:    {
537:     datops2[i]=0;
538:    }
539:    cont=0;
540:   }
541:   if (down && special && key==30)           //Se è stato premuto il tasto
    freccia sinistra scorre il cursore
542:   {                                         //di un posto a sinistra senza
    modificare datops2
543:    if (cont==0)
544:    {
545:     cont=0;
546:    }
547:    else
548:    {
549:     cont--;
550:     Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);
551:    }
552:   }
553:   if (down && special && key==31)           //Se è stato premuto il tasto
    freccia destra scorre il cursore
554:   {                                         //di un posto a destra senza
    modificare datops2
555:    if (cont==15)
556:    {
557:     cont=15;
558:    }
559:    else
560:    {
561:     cont++;
562:     Lcd_Cmd(LCD_MOVE_CURSOR_RIGHT);
563:    }
564:   }
565:   if (down && !special && key > 64 && key < 123) //Se il tasto premuto è una
    lettera visualizza key e memorizza
566:   {                                             //carattere in datops2 alla
    locazione cont
567:    datops2[cont]=key;
568:    cont++;
569:    Lcd_Chr(2,cont,key);
570:   }
571:   }
572:  delay_ms(50);
573: }
574: }
575:
576: void parserps2(void)                        //Interprete dei comandi da
    tastiera PS/2
577: {
578: if (Out_funz==0)
579: {
580:         Lcd_Cmd(LCD_CURSOR_OFF);           //Disattiva il cursore del display
    LCD
581:         if (strcmp (datops2,"ruotagiu")==0)  //Richiama la funziona rutoagiu()
    e il braccio si porta giu
582:         {
583:          PORTE.F2=1;
584:          delay_ms(500);
```

```
585:          PORTE.F2=0;
586:          ruota_basso();
587:         }
588:         else if (strcmp (datops2,"ruotasu")==0) //Richiama la funziona
     rutoasu() e il braccio si porta su
589:         {
590:          PORTE.F2=1;
591:          delay_ms(500);
592:          PORTE.F2=0;
593:          ruota_alto();
594:         }
595:         else if (strcmp (datops2,"ruotadx")==0) //Richiama la funzione gradi()
     per inserire il parametro e ruota a destra dei gradi inseriti
596:         {
597:          PORTE.F2=1;
598:          delay_ms(500);
599:          PORTE.F2=0;
600:          gradi();
601:          ruota_base(1,passibase);
602:         }
603:         else if (strcmp (datops2,"ruotasx")==0) //Richiama la funzione gradi()
     per inserire il parametro e ruota a sinistra dei gradi inseriti
604:         {
605:          PORTE.F2=1;
606:          delay_ms(500);
607:          PORTE.F2=0;
608:          gradi();
609:          ruota_base(0,passibase);
610:         }
611:         else if (strcmp (datops2,"apri")==0) //Richiama la funzione
     apri_pinza() e apre la pinza
612:         {
613:          PORTE.F2=1;
614:          delay_ms(500);
615:          PORTE.F2=0;
616:          apri_pinza();
617:         }
618:         else if (strcmp (datops2,"chiudi")==0) //Richiama la funzione
     chiudi_pinza() e chiude la pinza
619:         {
620:          PORTE.F2=1;
621:          delay_ms(500);
622:          PORTE.F2=0;
623:          chiudi_pinza();
624:         }
625:         else if (strcmp (datops2,"riposiziona")==0) //Richiama la funzione
     repos() e riposiziona la meccanica
626:         {
627:          PORTE.F2=1;
628:          delay_ms(500);
629:          PORTE.F2=0;
630:          outint=1;
631:          repos();
632:          outint=0;
633:         }
634:         else if (strcmp (datops2,"resetall")==0) //Richiama tutte le funzioni
     per il reset e resetta tutto (togliere manualmente pezzo)
635:         {
636:          PORTE.F2=1;
637:          delay_ms(500);
638:          PORTE.F2=0;
639:          reset_vert();
```

```
640:          reset_base();
641:          reset_pinza();
642:          reset_variabili();
643:         }
644:         else if (strcmp (datops2,"resetbase")==0) //Richiama la funzione per il
     reset della base (togliere manualmente pezzo)
645:         {
646:          PORTE.F2=1;
647:          delay_ms(500);
648:          PORTE.F2=0;
649:          reset_base();
650:          reset_variabili();
651:         }
652:         else if (strcmp (datops2,"resetvert")==0) //Richiama la funzione per il
     reset verticale (togliere manualmente pezzo)
653:         {
654:          PORTE.F2=1;
655:          delay_ms(500);
656:          PORTE.F2=0;
657:          reset_vert();
658:          reset_variabili();
659:         }
660:         else if (strcmp (datops2,"resetpinza")==0) //Richiama la funzione per
     il reset della pinza (togliere manualmente pezzo)
661:         {
662:          PORTE.F2=1;
663:          delay_ms(500);
664:          PORTE.F2=0;
665:          reset_pinza();
666:          reset_variabili();
667:         }
668:         else if (strcmp (datops2,"spegni")==0) //Imposta le variabili flag
     Spegni e Out_funz a 1
669:         {                                      //consentendo di uscire dalle
     funzioni e entrare nella modalità di spegnimento
670:          PORTE.F2=1;
671:          delay_ms(500);
672:          PORTE.F2=0;
673:          Spegni=1;
674:          Out_funz=1;
675:         }
676:         else if (strcmp (datops2,"velvert")==0) //Imposta la velocità per il
     motore verticale
677:         {
678:          PORTE.F2=1;
679:          delay_ms(500);
680:          PORTE.F2=0;
681:          velvert();
682:         }
683:         else if (strcmp (datops2,"velbase")==0) //Imposta la velocità per il
     motore della base
684:         {
685:          PORTE.F2=1;
686:          delay_ms(500);
687:          PORTE.F2=0;
688:          velbase();
689:         }
690:         else
691:         {
692:          display(0,2,1, "Comando Errato");   //Il comando inserito non è valido
     visualizza errore
693:          delay_ms(2000);
```

```
694:         }
695: }
696: }
697:
698: void gradi(void)                                  //Funzione per l'inserimento da
     tastiera dei gradi di rotazione base
699: {
700: char ind=0;                                      //Indice per il vettore gradi e
     l'aggiornamento della posizione sul display
701: int  gradi[3];                                   //Vettore contenente i gradi di
     cui ruotare
702: char esc=0;                                      //Variabile Flag - se posta a 1
     attiva rilevamento pressione pulsante INVIO
703: gradi[0]=0;                                     //Locazione 0 di gradi - centinaia
704: gradi[1]=0;                                     //Locazione 1 di gradi - decine
705: gradi[2]=0;                                     //Locazione 2 di gradi - unità
706: gradi[3]=0;                                     //Locazione 3 di gradi - non
     utilizzata
707:
708: if (Out_funz==0)
709: {
710:  Lcd_Cmd(LCD_CURSOR_OFF);                       //Spegne il cursore sul display
     LCD
711:  display(1,1,1,"Modalita PS/2");                //Visualizza modalità selezionata
712:  Lcd_Out(2,1,"Gradi.. ");                       //Richiede inserimento gradi
713:  Lcd_Cmd(LCD_BLINK_CURSOR_ON);                 //Accende il cursore sul display
     LCD
714:  while (Out_funz==0)
715:  {
716:   if (Ps2_Key_Read(&key, &special, &down))     //E' stato premuto un tasto
717:   {
718:    if (down && special && key==13 && esc==1)   //Se il tasto premuto è INVIO
     esce dalla funzione
719:    {
720:     PORTE.F2=1;                                //Lampeggio led
721:     delay_ms(500);
722:     PORTE.F2=0;
723:     Lcd_Cmd(LCD_CURSOR_OFF);
724:     esc=0;
725:     break;
726:    }
727:    if (down && !special && key>47 && key<58)   //Se il tasto premuto è un numero
     salva numero e visualizza sul display
728:    {
729:     ind++;                                     //Incrementa indice
730:     gradi[ind-1]=(key-48);                     //Converte da ASCII a decimale e
     salva numero in gradi[ind-1]
731:     Lcd_Chr(2,ind+8,key);                      //Visualizza il numero inserito
732:     if (ind-1==0) esc=0;                       //Se viene riscritto il primo
     numero si disattiva il tasto invio fino al completo inserimento
733:     if (ind==3)                                //Se il numero è completo (3
     cifre) si può premere invio per andare avanti
734:     {
735:      Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);            //Il cursore torna al primo
     numero per reinserire una nuova cifra
736:      Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);
737:      Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);
738:      ind=0;                                    //Resetta indice
739:      esc=1;                                    //Attiva il tasto invio
740:     }
741:    }
742:   }
```

```
743:   delay_ms(50);
744:  }
745:  passibase=(100*gradi[0])+(10*gradi[1])+gradi[2]; //Assegna a passibase il
      numero di gradi inserito da tastiera
746:  passibase=(passibase*85)/9;                    //Converte da gradi in passi con
      il fattore di conversione 85/9
747: }
748: }
749:
750: void velbase(void)                               //Funzione per l'inserimento
      della velocità di rot. verticale da tastiera
751: {
752: char esc=0;
753:
754: if (Out_funz==0)
755: {
756:  Lcd_Cmd(LCD_CURSOR_OFF);                        //Spegne il cursore sul display
      LCD
757:  display(1,1,1,"Modalita PS/2");                 //Visualizza Modalità selezionata
758:  Lcd_Out(2,1,"Vel.. ");                          //Richiede inserimento velocità
759:  Lcd_Cmd(LCD_BLINK_CURSOR_ON);                  //Attiva il lampeggio del cursore
760:  while (Out_funz==0)                            //Esegue finchè non viene
      richiesta l'uscita dalle funzioni (Out_funz=1)
761:  {
762:   if (Ps2_Key_Read(&key, &special, &down))     //E' stato premuto un tasto
763:   {
764:    if (down && special && key==13 && esc==1)   //E' stato premuto INVIO per cui
      imposta velocità ed esce dalla funzione
765:    {
766:     Lcd_Cmd(LCD_CURSOR_OFF);                   //Disattiva il cursore sul
      display LCD
767:     switch (vel_sel_base)                      //Switch in base a vel_sel_base,
      seleziona la velocità desiderata in base a vel_sel_vert
768:     {
769:      case 1:                                  //Imposta velocità minima e
      visualizza la velocità selezionata
770:           vel_base=8;
771:           PORTE.F2=1;
772:           delay_ms(500);
773:           PORTE.F2=0;
774:           display(0,2,1,"Vel Base Minima");
775:           delay_ms(1500);
776:      break;
777:
778:      case 2:                                  //Imposta velocità media e
      visualizza la velocità selezionata
779:           vel_base=4;
780:           PORTE.F2=1;
781:           delay_ms(500);
782:           PORTE.F2=0;
783:           display(0,2,1,"Vel Base Media");
784:           delay_ms(1500);
785:      break;
786:
787:      case 3:                                  //Imposta velocità massima e
      visualizza la velocità selezionata
788:           vel_base=2;
789:           PORTE.F2=1;
790:           delay_ms(500);
791:           PORTE.F2=0;
792:           display(0,2,1,"Vel Base Max");
793:           delay_ms(1500);
```

```
794:      break;
795:
796:      default:                                  //Valore inserito non ammesso,
     visualizza messaggio di errore
797:           display(0,2,1,"Valore");
798:           delay_ms(1500);
799:           display(0,2,1,"Non Ammesso");
800:           delay_ms(1500);
801:      break;
802:     }
803:     esc=0;
804:     break;
805:    }
806:    if (down && !special && key>47 && key<58) //Se il tasto premuto è un numero
     aggiorna vel_sel_base e visualizza valore inserito
807:    {
808:     vel_sel_base=(key-48);                    //Converte key da ASCII a decimale
     e memorizza in vel_sel_base
809:     Lcd_Chr(2,7,key);                         //Visualizza valore inserito
810:     Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);            //Il cursore torna dietro di 1
     carattere per reinserire un nuovo valore
811:     esc=1;                                    //Attiva il rilevamento della
     pressione del tasto INVIO per uscire
812:    }
813:    }
814:    }
815:   delay_ms(50);
816:  }
817: }
818:
819: void velvert(void)                             //Funzione per l'inserimento della
     velocità di rot. verticale da tastiera
820: {                                             //Funziona nello stesso modo di
     velbase()
821: char esc=0;
822:
823: if (Out_funz==0)
824: {
825:  Lcd_Cmd(LCD_CURSOR_OFF);
826:  display(1,1,1,"Modalita PS/2");
827:  Lcd_Out(2,1,"Vel.. ");
828:  Lcd_Cmd(LCD_BLINK_CURSOR_ON);
829:  while (Out_funz==0)
830:  {
831:   if (Ps2_Key_Read(&key, &special, &down))
832:   {
833:    if (down && special && key==13 && esc==1)
834:    {
835:     Lcd_Cmd(LCD_CURSOR_OFF);
836:     switch (vel_sel_vert)
837:     {
838:      case 1:
839:           vel_vert=7;
840:           PORTE.F2=1;
841:           delay_ms(500);
842:           PORTE.F2=0;
843:           display(0,2,1,"Vel Vert Minima");
844:           delay_ms(1500);
845:      break;
846:
847:      case 2:
848:           vel_vert=5;
```

```
849:           PORTE.F2=1;
850:           delay_ms(500);
851:           PORTE.F2=0;
852:           display(0,2,1,"Vel Vert Media");
853:           delay_ms(1500);
854:      break;
855:
856:      case 3:
857:           vel_vert=3;
858:           PORTE.F2=1;
859:           delay_ms(500);
860:           PORTE.F2=0;
861:           display(0,2,1,"Vel Vert Max");
862:           delay_ms(1500);
863:      break;
864:      default:
865:           Lcd_Cmd(LCD_CURSOR_OFF);
866:           display(0,2,1,"Valore");
867:           delay_ms(1500);
868:           display(0,2,1,"Non Ammesso");
869:           delay_ms(1500);
870:      break;
871:     }
872:     esc=0;
873:     break;
874:    }
875:    if (down && !special && key>47 && key<58)
876:    {
877:     vel_sel_vert=(key-48);
878:     Lcd_Chr(2,7,key);
879:     Lcd_Cmd(LCD_MOVE_CURSOR_LEFT);
880:     esc=1;
881:    }
882:    }
883:    }
884:   delay_ms(50);
885:  }
886: }
887:
888: //*************************************************************************
     *****//
889: //****************************** FUNZIONI DI RESET
     ******************************//
890: //*************************************************************************
     *****//
891:
892: void repos(void)                                  //Funzione per il
     riposizionamento selettivo della meccanica
893: {
894:  repos_flag=0;                                   //Variabile Flag - se posta a 1
     indica che la funzione
895:  if (errore==1)                                  //ha effettuato almeno un
     riposizionamento
896:  {
897:   display(1,1,1,"Riposizionamento");             //Visualizza operazione in corso
     se è in modalità di errore
898:   display(0,2,1,"Meccanica");
899:  }
900:  if (Out_funz==0 && outint==1)                   //Esegue in caso di nuova
     modalità selezionata o errore o spegnimento
901:  {
902:   delay_ms(1000);
```

```
903:   if (PORTB.F2==0)                              //Se è presente un pezzo lo porta
     giu se è alto e apre la pinza
904:   {
905:    repos_flag = 1;
906:    if (Posizione_vert < 1)                      //Se il pezzo è alto ruota giu
     prima di aprire la pinza
907:    {
908:     ruota_basso();
909:    }
910:    apri_pinza();
911:   }
912:   if (Posizione_vert > 0)                       //Se il braccio non è in
     posizione 0 verticale si riposiziona verticalmente
913:   {
914:    repos_flag = 1;
915:    reset_vert();
916:   }
917:   if (Posizione_base !=850 || error_code==3) //Se la base è fuori centro la
     riposiziona
918:   {
919:    repos_flag=1;
920:    if (Posizione_base > 870)                   //Se il braccio si trova a destra
     dell'area coperta dal sensore base
921:    {                                           //Imposta flag_return a 1 per
     ruotare a sinistra durante il reset_base
922:     Flag_return = 1;
923:    }
924:    reset_base();
925:   }
926:   reset_variabili();                           //Resetta le variabili
927:  }
928: }
929:
930: void Reset_variabili(void)               //Funzione per il reset delle variabili
931: {
932:  Esci = 0;
933:  errore = 0;
934:  nopezzo = 0;
935:  Out_funz = 0;
936:  Tornatosu = 0;
937:  Finecorsa = 0;
938:  contpinza = 0;
939:  Ril_pezzo = 0;
940:  Width_pezzo = 0;
941:  Limiteinf = 290;
942:  Finecorsa_mem = 0;
943:  Altezza_Pezzo = 0;
944:  Posizione_vert = 0;
945:  Posizione_base = 850;
946: }
947:
948: void Reset_pinza(void)                   //Funzione per il reset della pinza
949: {
950: if (Out_funz==0)                         //Esegue solo se Out_funz=0
951: {
952:      display(0,2,1,"Reset pinza");
953:      PORTC.F0=0;                         //Seleziona tramite indirizzamento il
     motore della pinza
954:      PORTC.F1=0;
955:      PORTC.F2=1;                         //Setta i bit per la chiusura della
     pinza
956:      PORTC.F3=0;
```

```
957:      while (PORTB.F2==1)                //Ruota finchè non si è arrivati al
      finecorsa
958:      {
959:      }
960:      PORTC.F2=0;                        //Setta i bit per l'apertura della pinza
961:      PORTC.F3=1;
962:      delay_ms(5000);                    //Apre per 5 secondi
963:      PORTC.F3=0;                        //Ferma pinza
964:      Esci = 0;
965:      Ril_pezzo=0;                       //Resetta variabili
966:      contpinza=0;
967:      Width_pezzo=0;
968:      delay_ms(500);                     //Attende 500 ms
969: }
970: }
971:
972: void Reset_base(void)                   //Funzione per il reset della base
973: {                                       //Dichiarazione variabili della funzione
974: char  Flag_Partenza = 0;                //Variabile flag - se posta a 1 indica
      che il braccio è partito da sopra la base
975: char  Flag_Finestra = 0;                //Variabile flag - se posta a 1 indica
      che il braccio è all'interno dell'area del sensore della base
976: char  uno=0;                           //Variabile flag per l'antirimbalzo
      software
977: char  outif=0;                         //Variabile flag - serve per uscire da
      un if
978: int   Passibase = 850;                 //Passi per far ruotare la base di 90
      gradi
979: int   Finestra_pos = 0;                //Conteggio larghezza finestra del
      sensore base in passi
980:
981: if (Out_funz==0)                       //Esegue solo se Out_funz=0
982:    {
983:      display(0,2,1,"Reset base");      //Visualizza l'operazione in corso
984:      Esci = 0;                         //Resetta variabili
985:      Posizione_base = 0;
986:      PORTC.F0=0;                       //Seleziona tramite indirizzamento il
      motore della base
987:      PORTC.F1=1;
988:      PORTC.F3=1;                       //Imposta il verso di rotazione a destra
989:
990:      if (Flag_return==1)               //Se si era verificato un errore a
      destra della base imposta la rotazione verso sinistra
991:      {
992:       PORTC.F3=0;
993:       Flag_return=0;
994:      }
995:
996:      if (PORTA.F5==0)                  //Se il braccio parte dalla base porta
      a 1 Flag_Partenza
997:      {
998:       Flag_Partenza=1;
999:      }
1000:
1001:     while (Esci==0)                   //Ruota finche Esci=0
1002:     {
1003:      PORTC.F2=0;
1004:      if (PORTA.F5==0)                 //Se trova l'area coperta dal sensore
      gestisce l'antirimbalzo
1005:      {
1006:       uno=0;
1007:       for (i=0; i<40; i++)             //Antirimbalzo software - ripete le
      operazioni per 40 volte in 4 ms durante un passo del motore
```

```
1008:        {
1009:         if (PORTA.F5==1)                //Se il sensore "rimbalza" e torna a
      uno anche una sola volta viene incrementata la variabile 1
1010:         {
1011:          uno++;
1012:         }
1013:          delay_us(100);
1014:        }
1015:         if (uno==0)                     //Se uno=0 allora ci si trova nell'area
      del sensore e il dato uscente dal sensore è stabile
1016:         {
1017:          Posizione_base=851;            //Setta posizione_base a 851 per poter
      ruotare di qualsiasi angolo
1018:          Finestra_pos++;                //Incrementa il conteggio della
      larghezza finestra
1019:          Flag_Finestra=1;               //Porta a 1 Flag_Finestra indicando che
      si trova dentro alla finestra coperta dal sensore base
1020:          outif=0;
1021:         }
1022:       }
1023:       if (PORTA.F5==1)                  //Se non ci si trova all'interno
      dell'area coperta dal sensore inserisce un ritardo
1024:       {                                 //per la giusta temporizzazione (clock
      del motorino) al posto del ritardo
1025:        delay_ms(5);                     //altrimenti introdotto
      dall'antirimbalzo software
1026:       }
1027:       PORTC.F2=1;
1028:       delay_ms(5);                      //Determina il clock del motore
1029:       Posizione_base++;                 //Incrementa la posizione della base
1030:
1031:       if (Flag_Finestra==1 && PORTA.F5==1 && outif==0)
1032:       {                                 //Se il braccio era dentro alla finestra
      e arriva al limite o destro o sinistro dell'area
1033:        outif=1;                         //Esegue delle operazioni per portarsi
      al centro della finestra
1034:        if (1==PORTC.F3)                 //Inverte il senso di rotazione del
      motore base
1035:        {
1036:         PORTC.F3=0;
1037:        }
1038:         else
1039:        {
1040:         PORTC.F3=1;
1041:        }
1042:        if (Flag_Partenza==1)            //Se il braccio era partito dall'interno
      dell'area coperta dal sensore
1043:        {                                //resetta la larghezza della finestra e
      il flag_partenza e attende 500ms
1044:         Flag_Finestra=0;
1045:         Flag_Partenza=0;
1046:         Finestra_pos=0;
1047:         delay_ms(500);
1048:        }
1049:        else
1050:        {
1051:         Esci=1;                         //Se il braccio era partito all'esterno
      dell'area coperta dal sensore
1052:        }                                //esce dal ciclo while
1053:        }
1054:       if (Posizione_base==850)          //Se il braccio ha ruotato a DX per 90
      gradi senza trovare l'area coperta dal sensore
```

```
1055:       {
1056:        Posizione_base=851;             //Toglie limiti su quanti gradi ruotare
      impostando a 851 passibase
1057:        PORTC.F3=0;                     //Inverte il verso di rotazione del
      motore
1058:        delay_ms(500);
1059:       }
1060:      }
1061:      delay_ms(500);
1062:      Finestra_pos=(Finestra_pos/2);    //Determina il punto centrale della
      finestra effettuando delle correzioni software
1063:      if (PORTC.F3==0)
1064:      {
1065:       Finestra_pos=Finestra_pos-5;     //Corregge via software la metà di
      Finestra_pos per un posizionamento corretto
1066:      }                                 //Se il braccio ruota a SX toglie 5
      passi altrimenti ne aggiunge 5
1067:      else
1068:      {
1069:       Finestra_pos=Finestra_pos+5;
1070:      }
1071:      for(i=0; i<Finestra_pos; i++)     //Ruota di Finestra_pos e si porta al
      centro dell'area coperta dal sensore
1072:       {
1073:        PORTC.F2=1;
1074:        delay_ms(5);
1075:        PORTC.F2=0;
1076:        delay_ms(5);
1077:       }
1078:      Posizione_base=850;               //Resetta Posizione_base al valore di
      partenza 850 (850=la base è in centro)
1079:      delay_ms(500);
1080:      }
1081: }
1082:
1083: void Reset_vert(void)                  //Funzione per il reset verticale
1084: {
1085: if (Out_funz==0)
1086:    {
1087:      display(0,2,1,"Reset Vert.");     //Visualizza l'operazione in corso
1088:      PORTC.F0=1;                       //Seleziona tramite indirizzamento il
      motore della base
1089:      PORTC.F1=0;
1090:      PORTC.F3=1;                       //Imposta il verso di rotazione verso
      l'alto
1091:      while (PORTA.F3==0)               //Ruota in alto finchè non trova il
      finecorsa superiore
1092:      {
1093:       PORTC.F2=1;
1094:       delay_ms(10);
1095:       PORTC.F2=0;
1096:       delay_ms(10);
1097:      }
1098:      delay_ms(500);
1099:      PORTC.F3=0;
1100:      for(i = 0; i < 180; i++)          //Dopo aver trovato il finecorsa
      superiore ruota di 180 passi per portarsi
1101:       {                                //parallelo al terreno
1102:        PORTC.F2=1;
1103:        delay_ms(10);
1104:        PORTC.F2=0;
1105:        delay_ms(10);
```

```
1106:       }
1107:       Posizione_vert=0;                //Resetta la posizione_vert al valore di
      partenza =0
1108:       delay_ms(500);
1109:     }
1110: }
1111:
1112: void setta_registri(void)               //Funzione per settare il giusto valore
      dei registri di configurazione (porte, interrupt, ADC)
1113: {
1114:      ADCON1 = 0x0E;                    //Setta il registro di configurazione
      dell'ADC
1115:      TRISA = 0x3C;                     //Setta la PORTA
1116:      TRISB = 0x8F;                     //Setta la PORTB
1117:      TRISC = 0xB0;                     //Setta la PORTC
1118:      TRISD = 0x00;                     //Setta la PORTD
1119:      TRISE = 0x00;                     //Setta la PORTE
1120:      PORTA = 0x24;                     //Setta PORTA
1121:      PORTB = 0x81;                     //Setta PORTB
1122:      PORTC = 0x00;                     //Setta PORTC
1123:      PORTD = 0x00;                     //Setta PORTD
1124:      PORTE = 0x03;                     //Setta PORTE
1125:      USART_Init(9600);                 //Inizializza il modulo USART del PIC:
      no parità, 8 bit, 9600 baud.
1126:      Ps2_Config(&PORTC, 4, 5);         //Inizializza i piedini per la
      comunicazione con la tastiera PS/2
1127:      Delay_ms(100);                    //Attende che la tastiera sia
      inizializzata correttamente
1128:      Lcd_Init(&PORTD);                 //Inizializza il Display
1129:      Lcd_Cmd(LCD_CLEAR);               //Cancella Il Display
1130:      Lcd_Cmd(LCD_CURSOR_OFF);          //Spegne il cursore
1131:      INTCON = 0xD8;                    //Setta il registro INTCON (Attiva gli
      interrupt)
1132:      PIE1.RCIE = 1;                    //Attiva l'interrupt di ricezione USART
1133: }
1134:
1135: //*************************************************************************
      *****//
1136: //****************************** GESTIONE INTERRUPT
      ******************************//
1137: //*************************************************************************
      *****//
1138:
1139: void interrupt (void)                      //Routine di interrupt
1140: {
1141: if (PIR1.RCIF==1)                         //Interrupt Seriale
1142: {
1143:    char c=0;                              //Carattere ricevuto dalla seriale
1144:    c = RCREG;                             //Riceve carattere e lo mette nella
      fifo
1145:    fifoput(c);
1146:    RCREG=0x00;                            //Resetta i registri della USART
1147:    RCSTA.F4=0;
1148:    RCSTA.F4=1;
1149:    PIR1.RCIF=0;                           //Resetta il bit flag dell'interrupt
      seriale
1150: }
1151: if (INTCON.F1==1 && PORTB.F0==0)           //Interrupt Spegnimento
1152: {
1153:  Spegni=1;                                //Imposta a 1 la variabile flag Spegni
1154:  PORTE.F1=0;                              //Spegne led verde
1155:  PORTE.F2=1;                              //Acende led rosso
```

```
1156:  if (select!=1 && errore==0)             //Se non è nella modalità automatica
       porta a 1 Out_funz
1157:  {
1158:   Out_funz=1;
1159:  }
1160:  if (errore==1 && sel_savestate!=1)
1161:  {
1162:    Out_funz=1;
1163:  }
1164: }
1165: if (INTCON.F0==1 && PORTB.F7==0 && outint==0) //Interrupt Cambio Modalità
1166: {
1167:  if (select==!1)                           //Se non è nella modalità automatica
       porta a 1 Out_funz
1168:  {
1169:   Out_funz=1;
1170:  }
1171:  outint=1;                                 //Disattiva interrupt cambio modalità
       momentaneamente
1172:  PORTE.F2=1;                               //Accende led rosso
1173:  if (select==4)                            //Se è nella gestione errore
       incremente sel_savestate al posto di select
1174:  {                                         //Per permettere di cambiare
       correttamente la modalità
1175:   sel_savestate++;
1176:   if (sel_savestate==4)                    //Se è arrivato in fondo imposta la
       prima modalità
1177:   {
1178:    sel_savestate=1;
1179:   }
1180:  }
1181:  select++;                                 //Incrementa select per il cambio di
       modalità
1182:  if (select==4)                           //Se è arrivato in fondo imposta la
       prima modalità
1183:  {
1184:   select=1;
1185:  }
1186: }
1187:  INTCON.F0=0;                              //Resetta bit flag di interrupt
1188:  INTCON.F1=0;
1189: }
1190:
1191: //*************************************************************************
       *****//
1192: //******************FUNZIONI PER LA MOVIMENTAZIONE
       MECCANICA************************//
1193: //*************************************************************************
       *****//
1194:
1195: void attesa_pezzo(void)                   //Funzione per attendere la presenza
       del pezzo
1196: {
1197: char outpezzo=0;                          //Variabile Flag - se posta a 1 esce
       dal ciclo while
1198: if (Out_funz==0)
1199: {
1200:   while (PORTB.F1==0 && outpezzo==0)      //Ciclo while - esegue finchè non c'è
       il pezzo o outpezzo=0
1201:   {
1202:    if (Spegni==1)                         //Se durante il ciclo e in assenza di
       pezzo mette outpezzo a 1 e esce dal ciclo
```

```
1203:    {
1204:     outpezzo=1;
1205:    }
1206:    if (outint==1)                          //Se si verifica un interrupt di
      cambio modalità in assenza di pezzo esce dal ciclo
1207:    {
1208:     outpezzo=1;
1209:    }
1210:   }
1211:   if (outpezzo==1)                         //Se è uscito forzatamente dal ciclo
      while porta a 1 out_forzato per non eseguire alcun ciclo automatico
1212:   {
1213:    Out_forzato=1;
1214:   }
1215:   else
1216:   {
1217:    display(0,2,1,"Rilevato Pezzo");        //Visualizza l'avvenuto rilevamento
      del pezzo
1218:    delay_ms(2000);                         //Attende 2 sec per la visualizzazione
1219:   }
1220: }
1221: }
1222:
1223: void ruota_base (char verso, int passi_base) //Funzione per far ruotare la base
      di "passibase" e nel verso "verso"
1224: {
1225: if (Out_funz==0)
1226: {
1227:   if (Posizione_vert > 0)                    //Se il braccio è in basso
      visualizza messaggio di errore
1228:   {
1229:    if (errore==0)                           //Se si è verificato un errore non
      visualizza niente
1230:    {
1231:     display(0,2,1,"Impossibile");           //Visualizza che la base non può
      ruotare
1232:     delay_ms(1500);
1233:     display(0,2,1,"Ruotare Base");
1234:     delay_ms(1500);
1235:    }
1236:    goto escibase;                           //Salta alla fine della funzione
      senza ruotare
1237:   }
1238:   display(0,2,1,"Rotazione");               //Visualizza l'operazione in corso
1239:   PORTC.F0 = 0;                            //Setta i pin relaviti
      all'indirizzo del motore da utilizzare
1240:   PORTC.F1 = 1;
1241:            if (verso==0)                   //Se verso=0 ruota verso Sinistra
1242:            {
1243:             PORTC.F3=0;                    //Porta a 0 il bit relativo alla
      rotazione (verso sinistra)
1244:             if (passi_base > Posizione_base)  //Se passi_base è maggiore del
      massimo di cui può ruotare verso sinistra
1245:             {
1246:              passi_base=posizione_base;    //Imposta passi_base al valore
      massimo possibile vero sx
1247:             }
1248:            }
1249:            else                                    //Se verso=1 ruota verso
      destra
1250:            {
1251:             PORTC.F3=1;                    //Porta a 1 il bit relativo alla
      rotazione (verso destra)
```

```
1252:             if (passi_base > (1700-posizione_base))      //Se passi_base è
      maggiore del massimo di cui può ruotare verso destra
1253:             {
1254:              passi_base = (1700-posizione_base);       //Imposta passi_base al
      valore massimo possibile verso dx
1255:             }
1256:            }
1257:            for(i = 0; i < passi_base; i++)             //Ciclo for produce
      "passi_base" impulsi e fa ruotare la base di "passi_base" passi
1258:            {
1259:
1260:             PORTC.F2=1;                                //Porta a 1 il bit 2
      della PORTC (clock bit)
1261:             for (ivel=0; ivel<=vel_base;ivel++)        //Determina un
      semiperiodo di clock in base alla velocità impostata
1262:             {
1263:              delay_ms(1);
1264:             }
1265:             PORTC.F2=0;                                //Porta a 0 il bit 2
      della PORTC (clock bit)
1266:             for (ivel=0; ivel<=vel_base;ivel++)        //Determina un
      semiperiodo di clock in base alla velocità impostata
1267:             {
1268:              delay_ms(1);
1269:             }
1270:             if (verso==0)                              //Se il braccio ruota
      verso destra aumenta la variabile relativa al numero dei passi altrimenti la
      decrementa
1271:             {
1272:              Posizione_base--;                         //Incrementa ad ogni
      passo verso destra la variabile posizione_base
1273:             }
1274:             else
1275:             {
1276:              Posizione_base++;                         //Decrementa ad ogni
      passo verso sinsitra la variabile posizione_base
1277:             }
1278:             if (Posizione_base > 840 && Posizione_base < 860 && PORTA.F5==1)
      //Se la base si trova nella posizione di partenza controlla che il sensore sia a
1279:             {
1280:               i = passi_base;                          //se il sensore è a
      livello alto i = passi_base per uscire dal ciclo for
1281:               Out_funz=1;                              //imposta a 1 Out_funz
      per uscire dalle funzioni
1282:               error_code = 3;                          //imposta il codice
      errore a 3 (errore posizionamento base)
1283:               errore = 1;                              //porta a 1 il flag di
      avvenuto errore
1284:               sel_savestate=select;                    //memorizza il select
      corrente
1285:               select = 4;                              //imposta select a 4 per
      entrare nella modalità gestione errore
1286:             }
1287:             if (Ril_pezzo==1 && PORTA.F2==1)           //Se il pezzo viene
      rimosso gestisce errore 4
1288:             {
1289:               i = passi_base;                          //imposta i=passi_base
      per uscire dal ciclo for
1290:               Out_funz=1;                              //imposta a 1 Out_funz
      per uscire dalle funzioni
1291:               error_code = 4;                          //imposta il codice
      errore a 4 (pezzo rimosso)
```

```
1292:               errore = 1;                                 //porta a 1 il flag di
      avvenuto errore
1293:               sel_savestate=select;                       //memorizza il select
      corrente
1294:               select = 4;                                 //imposta select a 4 per
      entrare nella modalità gestione errore
1295:              }
1296:            }
1297:   delay_ms(500);
1298:   escibase:;
1299:  }
1300: }
1301:
1302: void calcolo_rilascio(void)
1303: {
1304: if (Tornatosu==1)                       //Se il braccio è tornato su con il pezzo
      torna giù fino all'altezza del pezzo rilevato
1305:         {
1306:          Finecorsa = Finecorsa_mem;          //Carica in Finecorsa il valore
      iniziale o precedente (valore reale di finecorsa=finecorsa_mem)
1307:          if ( Posizione_base > 840 && Posizione_base < 860) //Controlla se il
      pezzo viene rilasciato sulla base con la fotocellula o fuori da essa
1308:          {
1309:           if (flag_preso_base==1)           //Se il pezzo era stato preso sulla
      base e viene rilasciato sulla base non aggiunge fattori correttivi al finecorsa
1310:           {
1311:            Limiteinf = Finecorsa;
1312:           }
1313:           else
1314:           {
1315:            Finecorsa=Finecorsa-Correttivo_rilascio-Corr_ril_sx; //Se il pezzo
      era stato preso fuori dalla base e viene rilasciato sulla base elimina un
      fattore correttivo a finecorsa
1316:            Limiteinf=Finecorsa;
1317:           }
1318:          }
1319:          else
1320:          {
1321:          if (flag_preso_base==1)           //Se il pezzo era stato preso sulla
      base e viene rilasciato fuori dalla base aggiunge un fattore correttivo a
      finecorsa
1322:           {
1323:            if (Posizione_base < 95)         //Correzione dovuta a imperfezioni
      della perpendicolarità dell'asse rispetto a terra
1324:            {
1325:             Correttivo_rilascio=14;
1326:            }
1327:            else if (Posizione_base > 94 && Posizione_base < 283)
1328:            {
1329:             Correttivo_rilascio=13;
1330:            }
1331:            else if (Posizione_base > 282 && Posizione_base < 475)
1332:            {
1333:             Correttivo_rilascio=10;
1334:            }
1335:            else if (Posizione_base > 474 && Posizione_base < 519)
1336:            {
1337:             Correttivo_rilascio=8;
1338:            }
1339:            Limiteinf = Finecorsa + Correttivo_rilascio; //Imposta il numero di
      passi corretto per tornare giù precisamente fino all'altezza del pezzo
1340:            Finecorsa = Limiteinf;
```

```
1341:            Correttivo_rilascio=7;
1342:           }
1343:           else
1344:           {
1345:            if (Posizione_base < 95)        //Correzione dovuta a imperfezioni
      della perpendicolarità dell'asse rispetto a terra
1346:            {
1347:             switch_presa1();
1348:            }
1349:            else if (Posizione_base > 94 && Posizione_base < 283)
1350:            {
1351:             switch_presa2();
1352:            }
1353:            else if (Posizione_base > 282 && Posizione_base < 475)
1354:            {
1355:             switch_presa3();
1356:            }
1357:            else if (Posizione_base > 474 && Posizione_base < 519)
1358:            {
1359:             switch_presa4();
1360:            }
1361:            else if (Posizione_base > 518)
1362:            {
1363:             Correttivo_rilascio=Corr_ril_sx;
1364:             Finecorsa=Finecorsa-Correttivo_rilascio;
1365:             Correttivo_rilascio=7;
1366:            }
1367:            Limiteinf = Finecorsa;           //Se il pezzo era stato preso sulla
      base e viene rilasciato sulla base finecorsa rimane invariato
1368:           }
1369:          }
1370:          }
1371: }
1372:
1373: void calcolo_presa(void)
1374: {
1375: if (Posizione_base > 840 && Posizione_base < 860)        //Stabilisce se il
      pezzo è stato preso sulla base con la fotocellula o esternamente ad essa
1376: {
1377:  flag_preso_base=1;               //Se viene preso sulla base porta
      flag_preso_base a 1
1378: }
1379: else
1380: {
1381: flag_preso_base=0;               //Se viene preso fuori dalla base porta
      flag_preso_base a 0
1382: if (Posizione_base < 95)          //Correzione dovuta a imperfezioni della
      perpendicolarità dell'asse rispetto a terra
1383: {
1384:  Corr_mecc=1;
1385:  Corr_ril_sx=7;
1386: }
1387: else if (Posizione_base > 94 && Posizione_base < 283)
1388: {
1389:  Corr_mecc=2;
1390:  Corr_ril_sx=6;
1391: }
1392: else if (Posizione_base > 282 && Posizione_base < 475)
1393: {
1394:  Corr_mecc=3;
1395:  Corr_ril_sx=3;
1396: }
```

```
1397: else if (Posizione_base > 474 && Posizione_base < 519)
1398: {
1399:  Corr_mecc=4;
1400:  Corr_ril_sx=1;
1401: }
1402: else if (Posizione_base > 518)
1403: {
1404:  Corr_mecc=5;
1405:  Corr_ril_sx=0;
1406: }
1407: }
1408: }
1409:
1410: void switch_presa1(void)
1411: {
1412: switch (Corr_mecc)
1413: {
1414: case 1:          //Correzione dovuta a imperfezioni della perpendicolarità
      dell'asse rispetto a terra
1415:  Limiteinf=Finecorsa;
1416:  break;
1417: case 2:
1418:  Finecorsa=Finecorsa+1;
1419:  break;
1420: case 3:
1421:  Finecorsa=Finecorsa+4;
1422:  break;
1423: case 4:
1424:  Finecorsa=Finecorsa+6;
1425:  break;
1426: case 2:
1427:  Finecorsa=Finecorsa+7;
1428:  break;
1429: }
1430: }
1431:
1432: void switch_presa2(void)
1433: {
1434: switch (Corr_mecc)
1435: {
1436: case 1:           //Correzione dovuta a imperfezioni della perpendicolarità
      dell'asse rispetto a terra
1437:  Finecorsa=Finecorsa-1;
1438:  break;
1439: case 2:
1440:  Limiteinf=Finecorsa;
1441:  break;
1442: case 3:
1443:  Finecorsa=Finecorsa+3;
1444:  break;
1445: case 4:
1446:  Finecorsa=Finecorsa+5;
1447:  break;
1448: case 5:
1449:  Finecorsa=Finecorsa+6;
1450:  break;
1451: }
1452: }
1453:
1454: void switch_presa3(void)
1455: {
1456: switch (Corr_mecc)
```

```
1457: {
1458: case 1:           //Correzione dovuta a imperfezioni della perpendicolarità
      dell'asse rispetto a terra
1459:  Finecorsa=Finecorsa-4;
1460:  break;
1461: case 2:
1462:  Finecorsa=Finecorsa-3;
1463:  break;
1464: case 3:
1465:  Limiteinf=Finecorsa;
1466:  break;
1467: case 4:
1468:  Finecorsa=Finecorsa+2;
1469:  break;
1470: case 5:
1471:  Finecorsa=Finecorsa+3;
1472:  break;
1473: }
1474: }
1475:
1476: void switch_presa4(void)
1477: {
1478: switch (Corr_mecc)
1479: {
1480: case 1:          //Correzione dovuta a imperfezioni della perpendicolarità
      dell'asse rispetto a terra
1481:  Finecorsa=Finecorsa-6;
1482:  break;
1483: case 2:
1484:  Finecorsa=Finecorsa-5;
1485:  break;
1486: case 3:
1487:  Finecorsa=Finecorsa-2;
1488:  break;
1489: case 4:
1490:  Limiteinf=Finecorsa;
1491:  break;
1492: case 5:
1493:  Finecorsa=Finecorsa+1;
1494:  break;
1495: }
1496: }
1497:
1498: void ruota_basso (void)                    //Funzione per far ruotare il
      braccio verso il basso
1499: {
1500: int uno = 0;                               //
1501: char ant_rimb = 0;                         //Variabile utilizzata per
      l'antirimbalzo software
1502: int i1 = 0;                                //Variabile contatore
1503: char iout=0;
1504: nopezzo=0;                                 //Resetta la variabile flag
      nopezzo
1505:
1506: if (Out_funz==0)
1507: {
1508:       if (Posizione_vert > 0)              //Se il braccio è in basso
      visualizza messaggio di errore
1509:       {
1510:         if (errore==0)                     //Se si è verificato un errore non
      visualizza niente
1511:         {
```

```
1512:          display(0,2,1,"Impossibile");      //Visualizza che il braccio non può
       ruotare verso il basso
1513:          delay_ms(1500);
1514:          display(0,2,1,"Ruotare Giu");
1515:          delay_ms(1500);
1516:         }
1517:        goto outbasso;                       //Salta alla fine della funzione
       senza ruotare
1518:       }
1519:       if (errore==1 || select==5)           //Se ruota in basso nella modalità
       errore o di spegnimento visualizza "Rilascio Pezzo"
1520:       {
1521:        display(0,2,1,"Rilascio Pezzo");     //Visualizza "Rilascio Pezzo"
1522:       }
1523:       else
1524:       {
1525:        display(0,2,1,"Abbassamento");       //Se non è nella modalità di errore
       o spegnimento visualizza "Abbassamento"
1526:       }
1527:       PORTC.F0 = 1;                         //Setta i pin relaviti
       all'indirizzo del motore da utilizzare
1528:       PORTC.F1 = 0;
1529:       PORTC.F3 = 0;                         //Porta a 0 il bit relativo al
       verso di rotazione del motore verticale (verso il basso)
1530:       calcolo_rilascio();
1531:         for (i=0; i<=Limiteinf; i++)        //Ciclo for per determinare il
       clock del motore passo passo
1532:         {
1533:          if (ant_rimb==0)                   //Se non è stato eseguito
       l'antirimbalzo introduce un ritardo per determinare il periodo di clock in base
       alla velocità selezionata
1534:          {
1535:           for (ivel=0; ivel<=vel_vert;ivel++)
1536:           {
1537:            delay_ms(1);
1538:           }
1539:          }
1540:          else                               //Se era stato eseguito
       l'antirimbalzo resetta la variabile flag che indica l'esecuzione
       dell'antirimbalzo (ant_rimb)
1541:          {
1542:           ant_rimb=0;
1543:          }
1544:          if (Tornatosu==0 && PORTA.F2==0)   //Se il sensore di prossimità
       cambia stato entra nella gestione dell'antirimbalzo
1545:          {
1546:           ant_rimb=1;                       //Porta a 1 la variabile flag
       relativa all'esecuzione dell'antirimbalzo
1547:           uno=0;
1548:           for (i1=0; i1<(vel_vert*10); i1++)//Antirimbalzo software - ripete le
       operazioni per (vel_vert*10) volte in "vel_vert" ms durante un passo del motore
1549:           {
1550:            if (PORTA.F2==1)                //Se il sensore "rimbalza" e torna a
       uno anche una sola volta viene incrementata la variabile uno
1551:            {
1552:             uno++;
1553:            }
1554:          delay_us(100);
1555:           }
1556:           }
1557:           if (PORTA.F2==0 && uno==0 && Esci==0) //Esegue se è la prima volta
       che il braccio si abbassa e il dato sul sensore di prossimità è stabile a zero
       (uno=0, il pezzo è stato rilevato correttamente)
```

```
1558:           {
      //Calcola i tre quarti dell'altezza del pezzo e imposta l'indice i per ruotare d
1559:               Altezza_Pezzo = Limiteinf + Corr_Pezzo - Posizione_vert;
      //Calcola l'altezza del pezzo in passi
1560:               i = Limiteinf - (Altezza_Pezzo/4);
      //Imposta i per arrivare a 3/4 di altezza pezzo
1561:               Finecorsa = Limiteinf + Corr_Pezzo - 1 - (3*(Altezza_Pezzo/4)) ;
      //Imposta finecorsa per tornare su alla posizione 0
1562:               Finecorsa_mem = Finecorsa;
      //Aggiorna Finecorsa_mem con il nuovo valore di Finecorsa
1563:               calcolo_presa();
1564:               Esci=1;                          //Imposta esci a 1 per non
      rientrare nel calcolo dei 3/4 altezza pezzo alla prossima rotazione basso
1565:           }
1566:           PORTC.F2=1;                          //Porta a 1 il bit 2 della
      PORTC (clock bit)
1567:           for (ivel=0; ivel<=vel_vert;ivel++)  //Attende vel_vert ms per
      determinare il clock del motore
1568:           {
1569:            delay_ms(1);
1570:           }
1571:           PORTC.F2=0;                          //Porta a 0 il bit 2 della
      PORTC (clock bit)
1572:           Posizione_vert++;                    //Aggiorna la variabile
      contenente la posizione in passi del braccio
1573:           if (PORTA.F4==1)                     //Se viene attivato il
      finecorsa inferiore gestisce errore 2
1574:           {
1575:            i = Limiteinf+1;                    //Imposta i=Limiteinf+1 per
      uscire dal ciclo for
1576:            Out_funz=1;                         //Imposta a 1 Out_funz per
      uscire dalle funzioni
1577:            error_code = 2;                     //Imposta il codice errore a 2
      (troppo giu)
1578:            errore = 1;                         //porta a 1 il flag di avvenuto
      errore
1579:            sel_savestate=select;               //memorizza il select corrente
1580:            select = 4;                         //imposta select a 4 per
      entrare nella modalità gestione errore
1581:           }
1582:           if (PORTA.F2==1 && Ril_pezzo==1)     //Se il pezzo viene rimosso
      gestisce errore 4
1583:           {
1584:               i = Limiteinf+1;                 //Imposta i=Limiteinf+1 per
      uscire dal ciclo for
1585:               Out_funz=1;                      //Imposta a 1 Out_funz per
      uscire dalle funzioni
1586:               error_code = 4;                  //Imposta il codice errore a 4
      (Pezzo rimosso)
1587:               errore = 1;                      //porta a 1 il flag di avvenuto
      errore
1588:               sel_savestate=select;            //memorizza il select corrente
1589:               select = 4;                      //imposta select a 4 per
      entrare nella modalità gestione errore
1590:           }
1591:         }
1592:         if (PORTA.F2==1 && i==limiteinf+1 && errore==0) //Se il braccio ha
      ruotato verso il basso e non ha trovato il pezzo torna su
1593:         {
1594:          nopezzo=1;                           //Porta a 1 il flag di
      segnalazione pezzo non rilevato
```

```
1595:          Finecorsa=290;                          //Setta Finecorsa=Limiteinf per
       tornare su alla posizione 0
1596:          display(0,2,1,"Nessun Pezzo");          //Visualiza "Nessun Pezzo sul
       display"
1597:          delay_ms(1000);                         //Attende 1 secondo per la
       visualizzazione
1598:          ruota_alto();                           //Torna su
1599:          Finecorsa=0;                            //Resetta le variabili
1600:          Finecorsa_mem=0;
1601:          Tornatosu=0;
1602:         }
1603:  delay_ms(500);
1604:  }
1605:  outbasso:;
1606: }
1607:
1608: void ruota_alto (void)                           //Funzione per far ruotare il
       braccio verso l'alto
1609: {
1610: if (Out_funz==0)
1611: {
1612:       if (Posizione_vert < 1)                   //Se il braccio è in posizione
       0 non può ruotare su
1613:       {
1614:         if (errore==0)                          //Se non si sono verificati
       errori visualizza che è impossibile ruotare verso l'alto
1615:        {
1616:         display(0,2,1,"Impossibile");           //Visualizza "Impossibile
       Ruotare Su"
1617:         delay_ms(1500);
1618:         display(0,2,1,"Ruotare Su");
1619:         delay_ms(1500);
1620:        }
1621:        goto outalto;                            //Salta alla fine della
       funzione senza ruotare su
1622:       }
1623:       if (nopezzo==0)                           //Se il braccio è sceso ha
       rilevato correttamente un pezzo visualizza "Elevamento"
1624:       {
1625:        display(0,2,1,"Elevamento");
1626:       }
1627:       PORTC.F0 = 1;                             //Setta i pin relaviti
       all'indirizzo del motore da utilizzare
1628:       PORTC.F1 = 0;
1629:       PORTC.F3 = 1;                             //Porta a 0 il bit relativo al
       verso di rotazione del motore verticale (verso l'alto=
1630:       for(i = 0; i <= Finecorsa; i++)           //Ciclo for per determinare il
       clock del motore passo passo
1631:       {
1632:          PORTC.F2=1;                            //Porta a 1 il bit 2 della
       PORTC (clock bit)
1633:          for (ivel=0; ivel<=vel_vert;ivel++)    //Attende vel_vert ms per
       determinare il clok del motore in base alla velocità selezionata
1634:          {
1635:           delay_ms(1);
1636:          }
1637:          PORTC.F2=0;                            //Porta a 0 il bit 2 della
       PORTC (clock bit)
1638:          for (ivel=0; ivel<=vel_vert;ivel++)    //Attende vel_vert ms per
       determinare il clok del motore in base alla velocità selezionata
1639:          {
1640:           delay_ms(1);
```

```
1641:          }
1642:          Posizione_vert=(Finecorsa-i);
1643:          if (PORTA.F2==1 && nopezzo==0 && Ril_pezzo==1) //Se il pezzo viene
       rimosso gestisce errore 4
1644:          {
1645:            i = Finecorsa+1;                       //Imposta i=Finecorsa+1 per
       uscire dal ciclo for
1646:            Out_funz = 1;                           //Imposta a 1 Out_funz per
       uscire dalle funzioni
1647:            error_code = 4;                        //Imposta il codice errore a 4
       (Pezzo rimosso)
1648:            errore = 1;                            //porta a 1 il flag di avvenuto
       errore
1649:            sel_savestate = select;                //memorizza il select corrente
1650:            select = 4;                            //imposta select a 4 per
       entrare nella modalità gestione errore
1651:          }
1652:          if (PORTA.F3==1)                         //Se il braccio è salito oltre
       il finecorsa superiore gestisce errore 1
1653:          {
1654:           i = Finecorsa+1;                        //Imposta i=Finecorsa+1 per
       uscire dal ciclo for
1655:           Out_funz=1;                             //Imposta a 1 Out_funz per
       uscire dalle funzioni
1656:           error_code = 1;                         //Imposta il codice errore a 1
       (Troppo Su)
1657:           errore = 1;                             //porta a 1 il flag di avvenuto
       errore
1658:           sel_savestate=select;                   //memorizza il select corrente
1659:           select = 4;                             //imposta select a 4 per
       entrare nella modalità gestione errore
1660:          }
1661:       }
1662:       if (0==Tornatosu)                           //Se il braccio non è ancora
       risalito con il pezzo attiva il flag tornatosu=1
1663:       {
1664:        Tornatosu=1;
1665:        if (Ril_pezzo==0)                          //Se non è stato preso alcun
       pezzo durante la rotazione giu resetta Tornatosu e limiteinf
1666:        {
1667:         Tornatosu = 0;
1668:         Limiteinf = 290;
1669:         Esci=0;
1670:        }
1671:       }
1672:       else
1673:       {
1674:        if (Ril_pezzo==0)                          //Se il braccio è già tornato
       su una volta con il pezzo ma non è ancora stato rilasciato resetta
1675:        {                                          //tornato su, esci, limiteinf,
       finecorsa e finecorsa_mem
1676:         Tornatosu = 0;
1677:         Esci=0;
1678:         Limiteinf = 290;
1679:         Finecorsa = 0;
1680:         Finecorsa_mem = 0;
1681:        }
1682:       }
1683:   delay_ms(500);
1684:   outalto:;
1685:   }
1686: }
```

```
1687:
1688: void chiudi_pinza(void)                          //Funzione per la chiusura
      della pinza
1689: {
1690: char escipinza=0;                                //Variabile Flag - se impostata
      a 1 esce dal ciclo while per la chiusura
1691:
1692: if (Out_funz==0)
1693: {
1694:  if (PORTB.F2==0)                                //Se la pinza è già chiusa è
      impossibile chiuderla
1695: {
1696:  if (errore==0)                                  //Se non si è verificato un
      errore visualizza messaggio altrimenti va a outpinza:
1697:  {
1698:   display(0,2,1,"Impossibile");                  //Visualizza che è impossibile
      chiudere la pinza
1699:   delay_ms(1500);
1700:   display(0,2,1,"Chiudere Pinza");
1701:   delay_ms(1500);
1702:  }
1703:  goto outpinza;                                  //Se si verifica un errore non
      esegue la funzione saltando a outpinza:
1704: }
1705:       display(0,2,1,"Chiusura Pinza");           //Visualizza l'operazione in
      corso
1706:       PORTC.F0 = 0;                              //Setta i pin relaviti
      all'indirizzo del motore da utilizzare
1707:       PORTC.F1 = 0;
1708:       PORTC.F2 = 1;                              //Setta i due pin PORTC.F2,
      PORTC.F3 per la chiusura della pinza
1709:       PORTC.F3 = 0;
1710:       Width_pezzo = 0;                           //Azzera larghezza pezzo
1711:       while (escipinza==0)                       //Esegue finchè escipinza=0
1712:       {
1713:        delay_ms(1);                              //Attende 1 ms a ogni ciclo
1714:        Width_pezzo++;                            //Incremeta widht pezzo di 1
      (widht pezzo contiene la larghezza del pezzo in ms)
1715:        if (PORTB.F2==0)                          //Se si attiva lo switch è
      stato afferrato il pezzo
1716:        {
1717:         Ril_pezzo=1;                             //Setta a 1 variabile flag
      Ril_pezzo (il pezzo è stato afferrato)
1718:         escipinza=1;                             //Setta a 1 variabile flag
      escipinza per uscire dal ciclo di chiusura
1719:        }
1720:        if (Width_pezzo>5000)                     //Controllo Timeout, se la
      pinza ha chiuso per troppo tempo senza trovare un pezzo
1721:        {                                         //gestisce errore 5
1722:         escipinza = 1;                           //Imposta escipinza per uscire
      dal ciclo for
1723:         Out_funz=1;                              //Imposta a 1 Out_funz per
      uscire dalle funzioni
1724:         error_code = 5;                          //Imposta il codice errore a 5
      (Problema Pinza)
1725:         errore = 1;                              //porta a 1 il flag di avvenuto
      errore
1726:         sel_savestate = select;                  //memorizza il select corrente
1727:         select = 4;                              //imposta select a 4 per
      entrare nella modalità gestione errore
1728:        }
1729:       }
```

```
1730:       delay_ms(100);
1731:       PORTC.F2=0;                                  //Spegne il motore della pinza
1732:       delay_ms(500);
1733:  }
1734:  outpinza:;
1735: }
1736:
1737: void apri_pinza(void)                              //Funzione per l'apertura della
      pinza
1738: {
1739: if (Out_funz==0)
1740: {
1741: if (contpinza==Width_pezzo || posizione_vert==0 && ril_pezzo==1) //Se la pinza
      è già aperta o è sul con il pezzo non è possibile aprirla
1742: {
1743:  if (errore==0)                                   //Se non si è verificato un
      errore visualizza che è impossibile aprire la pinza
1744:  {
1745:   display(0,2,1,"Impossibile");                   //Visualizza messaggio di errore
      "Impossibile Aprire Pinza"
1746:   delay_ms(1500);
1747:   display(0,2,1,"Aprire Pinza");
1748:   delay_ms(1500);
1749:  }
1750:  contpinza=0;                                     //Azzera contpinza
1751:  goto outpinza1;                                  //Salta alla fine della funzione
      senza aprire la pinza
1752: }
1753:       display(0,2,1,"Apertura Pinza");            //Visualizza operazione in corso
1754:       PORTC.F0=0;                                 //Seleziona tramite
      indirizzamento il motore della pinza
1755:       PORTC.F1=0;
1756:       PORTC.F2=0;                                 //Imposta i bit per l'apertura
      della pinza
1757:       PORTC.F3=1;
1758:       Width_pezzo = Width_pezzo - Correttivo_pinza;          //Aggiorna la
      larghezza del pezzo Width_Pezzo
1759:       for (contpinza=0; contpinza < Width_pezzo; contpinza++) //Apre la pinza
      per Width_pezzo ms
1760:       {
1761:        delay_ms(1);
1762:       }
1763:       PORTC.F3=0;                                 //Spegne il motore della pinza
1764:       Ril_pezzo=0;                                //Azzera variabile flag Ril_pezzo
1765:   delay_ms(500);
1766:  }
1767:  outpinza1:;
1768: }
1769:
1770: //*************************************************************************
      *****//
1771: //********** FUNZIONI PER LA GESTIONE DELLO SPEGNIMENTO, ERRORE e DISPLAY
      **********//
1772: //*************************************************************************
      *****//
1773:
1774: void Arresto(void)                              //Funzione per arrestare il sistema
1775: {
1776:  Lcd_Cmd(LCD_CURSOR_OFF);                       //Spegne il cursore del display
1777:  PORTE.F1=0;                                    //Spegne led verde
1778:  PORTE.F2=1;                                    //Accende led rosso
1779:  display(1,1,1,"Arresto Sistema");              //Visualizza che si sta per spegnere
```

```
1780:  display(0,2,1,"In Corso");
1781:  Out_funz=0;                          //Porta a 0 Out_funz per permettere
       di eseguire repos()
1782:  outint=1;                            //Porta a 1 outint per permettere di
       eseguire repos()
1783:  repos();                             //Richiama la funzione repos() per
       rilasciare il pezzo e riposizionare
1784:  if (repos_flag==0)                   //Se repos() non ha eseguito nemmeno
       un riposizionamento attende 5 sec prima di spegnere
1785:  {
1786:   delay_ms(5000);
1787:  }
1788:  else                                 //Se repos ha eseguito almeno un
       posizionamento attende 1 sec prima di spegnere
1789:  {
1790:   delay_ms(1000);
1791:  }
1792:  PORTE=0x00;                          //Porta a 0 tutta la PORTE e spegne
       il relais di alimentazione
1793: }
1794:
1795: void gestione_errori(void)            //Funzione per la gestione degli
       errori del braccio (es. pezzo rimosso)
1796: {
1797:  Lcd_Cmd(LCD_CURSOR_OFF);             //Spegne il cursore del display
1798:  ByteToStr(error_code, txt);          //Converte in stringa error_code per
       la visualizzazione
1799:  display(1,1,1,"Errore");             //Visualizza che si è verificato un
       errore
1800:  Lcd_Out_Cp(txt);                     //Visualizza il numero relativo
       all'errore
1801:  switch (error_code)                  //Visualizza il codice dell'errore in
       base al numero
1802:  {
1803:   case 1:
1804:   display(0,2,1,"Troppo Su");         //Il braccio ha ruotato fino al
       finecorsa superiore (1 - troppo su)
1805:   break;
1806:   case 2:
1807:   display(0,2,1,"Troppo Giu");        //Il braccio ha ruotato fino al
       finecorsa inferiore (2 - troppo giu)
1808:   break;
1809:   case 3:
1810:   display(0,2,1,"Base Out Centro");   //La base si è sposizionata (3 - base
       out centro)
1811:   break;
1812:   case 4:
1813:   display(0,2,1,"Pezzo Rimosso");     //Il pezzo è stato rimosso (4 - pezzo
       rimosso)
1814:   break;
1815:   case 5:
1816:   display(0,2,1,"Problema Pinza");    //La pinza si è chiusa senza
       stringere alcun pezzo o lo switch non ha funzionato (Problema Pinza)
1817:   break;
1818:   default:
1819:   break;
1820:  }
1821:  for (i=0; i<5; i++)                  //Il led rosso lampeggia per 5
       secondi con un periodo di 1 sec
1822:  {
1823:   PORTE.F2=1;
1824:   delay_ms(500);
```

```
1825:   PORTE.F2=0;
1826:   delay_ms(500);
1827:  }
1828:  for (i=0; i<8; i++)                        //Il led rosso lampeggia per gli
       ultimi 1,8 secondi con un periodo di 400 msec
1829:  {
1830:   PORTE.F2=1;
1831:   delay_ms(200);
1832:   PORTE.F2=0;
1833:   delay_ms(200);
1834:  }
1835:  outint=1;                                 //Porta a 1 outint per permettere
       l'esecuzione di repos()
1836:  Out_funz=0;                               //Resetta Out_funz per permettere di
       nuovo l'esecuzione delle funzioni
1837: }
1838:
1839: void display(char clear, char riga, char center, char dato[16]) //Funzione per
       la visualizzazione sul display
1840: {
1841:  char colonna=1;                           //Contiene la posizione sul
       display colonna (1 di default)
1842:  if (clear==1)                            //Se clear = 1 pulisce il display
1843:  {
1844:   Lcd_Cmd(LCD_CLEAR);
1845:  }
1846:  if (riga==2)                              //Se è selezionata la riga 2
       pulisce la seconda riga per prepararla alla scrittura
1847:  {
1848:   Lcd_Out(2,1,"                ");
1849:  }
1850:  if (center==1)                            //Se center = 1 centra la scritta
       sul display
1851:  {
1852:   colonna=((16-strlen(dato))/2)+1;         //Calcola la lunghezza del dato da
       scrivere e in base al valore risultante
1853:  }                                         //Imposta colonna per centrare la
       scritta sul display
1854:  else
1855:  {
1856:   colonna=1;                               //Se center = 0 imposta colonna a
       1 di default senza centrare (allineamento a sinistra)
1857:  }
1858:  Lcd_Out(riga,colonna,dato);               //Visualizza il dato sulla riga
       "riga" a partire dal carattere "colonna"
1859: }
1860:
1861: //*************************************************************************
       *****//
1862: //****************************** PROGRAMMA PRINCIPALE
       ******************************//
1863: //*************************************************************************
       *****//
1864:
1865: void main (void)                           //Programma principale
1866: {
1867:      char rxc;                             //Variabile ricezione seriale
1868:      setta_registri();                     //Setta i registri
1869:      display(0,1,1,"Armatronick");         //Visualizza titolo di avvio
1870:      display(0,2,1,"Andrea Aglietti");
1871:      delay_ms(2000);                       //Attende 2 secondi
1872: while (1)
```

```
1873: {
1874:      switch (select)                          //Switch in base a select per la
      selezione della modalità
1875:      {                                        //(0 - Inizializzazione, 1 -
      Automatica, 2 - Da PC, 3 - Da Tastiera, 4 - Gestione errore, 5 - Arresto del
      sistema)
1876:      case 0:                                  //Select = 0 - Inizializzazione
1877:      Lcd_Cmd(LCD_CURSOR_OFF);                 //Disattiva il cursore sul display
1878:      display(1,1,1,"Inizializzazione");       //Visualizza modalità selezionata
1879:      delay_ms(1000);
1880:      Reset_vert();                            //Resetta tutta la meccanica e
      inizializza le variabili
1881:      Reset_base();
1882:      Reset_pinza();
1883:      select = sel_savestate;                  //Seleziona select=1 di default (
      modalità automatica )
1884:      outint=0;                                //Attiva interrupt cambio modalità
1885:      break;
1886:
1887:      case 1:                                  //Select = 1 - Modalità Automatica
1888:      Lcd_Cmd(LCD_CURSOR_OFF);                 //Disattiva il cursore sul display
1889:      repos();                                 //Riposiziona in caso di meccanica
      fuori posizione
1890:      while (select==1)                        //Esegue Fintanto che select = 1
1891:       {
1892:        nopezzo=0;                             //Resetta nopezzo
1893:        display(1,1,1,"Modalita Auto");        //Visualizza Modalità Selezionata
1894:        display(0,2,1,"Selezionata");
1895:        if (outint==1)                         //Se era stata cambiata la
      modalità attende 200 ms per antirimbalzo
1896:        {                                      //prima di riattivare l'interrupt
      di cambio modalità (outint)
1897:         PORTE.F2=0;
1898:         delay_ms(200);
1899:         outint=0;
1900:         Out_forzato=0;
1901:        }
1902:        attesa_pezzo();                        //Sta fermo finchè non trova il
      pezzo
1903:       if (Out_forzato==0)                     //Esegue solo se out_forzato = 0
1904:       {
1905:        Esci=0;                                //Resetta Esci
1906:        ruota_basso();                         //Ruota in basso
1907:        if (nopezzo==0)                        //Se non viene trovato il pezzo
      non esegue tutta questa parte
1908:         {                                     //ma torna all'inizio del ciclo
      automatico (attesa_pezzo)
1909:          chiudi_pinza();                      //Afferra il pezzo
1910:          ruota_alto();                        //Ruota in alto
1911:          ruota_base(Versog,850);               //Ruota di 90 gradi nel verso
      "Verso"
1912:          ruota_basso();                       //Si abbassa per rilasciare il
      pezzo
1913:          apri_pinza();                        //Rilascia il pezzo
1914:          ruota_alto();                        //Torna Su
1915:          if (0==Versog)                        //Inverte il verso per lasciare
      un pezzo a +90 gradi
1916:          {                                    //e il successivo a -90
1917:           Versog=1;
1918:          }
1919:          else
1920:          {
```

```
1921:           Versog=0;
1922:          }
1923:         ruota_base(Versog,850);               //Ruota di 90 gradi nel verso
      "Verso"
1924:         }
1925:        }
1926:        if (Spegni==1)                         //Se spegni è stato portato a 1
      porta select a 5 per entrare nella modalità di spegnimento
1927:        {
1928:         select=5;
1929:        }
1930:       }
1931:      break;
1932:
1933:      case 2:                                  //Select=2 - Modalità Da PC
1934:      repos();                                 //Riposiziona in caso di meccanica
      fuori posizione
1935:      Lcd_Cmd(LCD_CURSOR_OFF);                 //Disattiva il cursore sul display
      LCD
1936:      display(1,1,1,"Modalita PC");            //Visualizza Modalità Selezionata
1937:      display(0,2,1,"Selezionata");
1938:      while (select==2)                        //Esegue fintanto che select = 2
1939:      {
1940:       if (outint==1)                          //Se era stata cambiata la
      modalità attende 200 ms per antirimbalzo
1941:       {                                       //prima di riattivare l'interrupt
      di cambio modalità (outint)
1942:        PORTE.F2=0;
1943:        delay_ms(200);
1944:        outint=0;
1945:       }
1946:       if( fifoget( &rxc ) )                   //Se è presente un nuovo comando
      nella fifo controlla il pacchetto
1947:       {                                       //ed esegue istruzione ricevuta
1948:        controllo_pacchetto( rxc );
1949:        display(0,2,1,"Selezionata");
1950:       }
1951:       if (Spegni==1)                          //Se spegni è stato portato a 1
      porta select a 5 per entrare nella modalità di spegnimento
1952:       {
1953:        select=5;
1954:       }
1955:      }
1956:      break;
1957:
1958:      case 3:                                  //Select = 3 - Modalità Da
      Tastiera PS/2
1959:      repos();                                 //Riposiziona in caso di meccanica
      fuori posizione
1960:      cont=0;                                  //Resetta contatore "cont"
1961:      display(1,1,1,"Modalita PS/2");          //Visualizza Modalità Selezionata
1962:      Lcd_Cmd(LCD_SECOND_ROW);                 //Sposta il cursore sulla seconda
      riga
1963:      Lcd_Cmd(LCD_BLINK_CURSOR_ON);            //Attiva il lampeggio del cursore
      sul display LCD
1964:      while (select==3)                        //Esegue fintanto che select = 3
1965:      {
1966:        if (outint==1)                         //Se era stata cambiata la
      modalità attende 200 ms per antirimbalzo
1967:        {                                      //prima di riattivare l'interrupt
      di cambio modalità (outint)
1968:         PORTE.F2=0;
```

```
1969:          delay_ms(200);
1970:          outint=0;
1971:         }
1972:         tastiera();                          //Richiama la funzione di gestione
      della tastiera
1973:         if (Spegni==1)                       //Se spegni è stato portato a 1
      porta select a 5 per entrare nella modalità di spegnimento
1974:         {
1975:          select=5;
1976:         }
1977:       }
1978:       break;
1979:
1980:       case 4:                                //Select = 4 - Modalità gestione
      errori
1981:       gestione_errori();                     //Richiama la funzione per la
      gestione degli errori
1982:       if (PORTB.F2==1)                       //Se la pinza è mezza aperta o
      aperta la resetta
1983:       {
1984:        Reset_pinza();
1985:       }
1986:       repos();                               //Riposiziona selettivamente la
      meccanica
1987:       reset_variabili();                     //Resetta le variabili
1988:       error_code=0;                          //Resetta l'error_code
1989:       select=sel_savestate;                  //Ripristina select per tornare
      alla modalità in cui si è verificato l'errore
1990:       if (Spegni==1)                         //Se spegni è stato portato a 1
      porta select a 5 per entrare nella modalità di spegnimento
1991:       {
1992:        select=5;
1993:       }
1994:       break;
1995:
1996:       case 5:                                //Select = 5 - Modalità Spegnimento
1997:       Arresto();                             //Richiama la funzione per
      l'arresto del sistema
1998:       break;
1999:
2000:       default:
2001:       break;
2002:       }
2003: }
2004: }
```

Codice Sorgente

Software

Visual Basic

Form 1 - 1

```
Dim Buffer, Instring As String ' Imposta la porta e la apre
Option Explicit
Public Function cheksum(stringa As String) As Integer
    Dim i, l, s As Integer
    l = Len(stringa)
    s = 0
    For i = 1 To l
        s = s + Asc(Mid(stringa, i, 1))
    Next
    s = s Mod 256
    cheksum = s
End Function
Public Sub invia(ByVal payload As String)
   Dim ck As Integer
   If MSComm1.PortOpen = True Then
       MSComm1.PortOpen = False
       Instring = ""
   End If
   If CInt(Combo1.Text) > 0 Then
    MSComm1.CommPort = CInt(Combo1.Text)
   End If
   MSComm1.Settings = "9600,N,8,1"
   MSComm1.Handshaking = comNone
   MSComm1.RThreshold = 1
   MSComm1.PortOpen = True
   ck = cheksum(payload)
   Buffer = Chr(2) & payload & Chr(ck) & Chr(3)
   MSComm1.Output = Buffer
End Sub

Private Sub Combo1_Change()
If MSComm1.PortOpen = True Then
    MSComm1.PortOpen = False
End If
   MSComm1.CommPort = CInt(Combo1.Text)
   MSComm1.Settings = "9600,N,8,1"
   MSComm1.Handshaking = comNone
   MSComm1.RThreshold = 1
   MSComm1.PortOpen = True
End Sub

Private Sub Command1_Click()
    invia ("Spegni" & "0000")
End Sub

Private Sub Command2_Click()
    Dim n, appoggio, comando As String
    If Val(Text4.Text) < 0 Or Val(Text4.Text) > 180 Then
        MsgBox "Valore non ammesso", vbCritical, "Attenzione"
    Else
        n = Text4.Text
        If n = "" Then
            Text4.Text = "0"
        End If
        If Len(n) = 1 Then
            appoggio = "000" & n
        End If
        If Len(n) = 2 Then
            appoggio = "00" & n
        End If
        If Len(n) = 3 Then
            appoggio = "0" & n
        End If
        If Len(n) = 4 Then
            appoggio = n
        End If
        comando = "SX" & appoggio
```

```
        invia (comando)
    End If
End Sub

Private Sub Command3_Click()
    Dim n, appoggio, comando As String
    If Val(Text2.Text) < 0 Or Val(Text2.Text) > 180 Then
        MsgBox "Valore non ammesso", vbCritical, "Attenzione"
    Else
        n = Text2.Text
        If n = "" Then
            Text2.Text = "0"
        End If
        If Len(n) = 1 Then
            appoggio = "000" & n
        End If
        If Len(n) = 2 Then
            appoggio = "00" & n
        End If
        If Len(n) = 3 Then
            appoggio = "0" & n
        End If
        If Len(n) = 4 Then
            appoggio = n
        End If
        comando = "DX" & appoggio
        invia (comando)
    End If
End Sub

Private Sub Command4_Click()
    invia ("SU" & "0000")
End Sub

Private Sub Command5_Click()
    invia ("GIU" & "0000")
End Sub

Private Sub Command6_Click()
    invia ("Chiudi" & "0000")
End Sub

Private Sub Command7_Click()
    invia ("Apri" & "0000")
End Sub
Private Sub Aggiorna_COM()
    Dim s As Byte
    Dim out As Byte
        Combo1.Clear
        With MSComm1
          For s = 1 To 255
            Err.Clear
            If .PortOpen = True Then .PortOpen = False
            On Error Resume Next
            .CommPort = s
            .PortOpen = True
            If Err.Number = 0 Then
                Combo1.AddItem Trim(CStr(s))
                If out = 0 Then
                    Combo1.Text = s
                    out = 1
                End If
            End If
            .PortOpen = False
          Next s
        End With
```

```
End Sub

Private Sub Command8_Click()
   Aggiorna_COM
End Sub

Private Sub esci_Click()
   End
End Sub

Private Sub Form_Load()
   Aggiorna_COM
End Sub

Private Sub informazioni_Click()
   Form3.Show
End Sub

Private Sub MSComm1_OnComm()
  If (MSComm1.CommEvent = comEvReceive) Then
      Instring = Instring & MSComm1.Input
      estrai_comando
  End If
End Sub

Public Sub estrai_comando()
   Dim ck As Integer
   ck = controlla_cheksum()
   If ck = 1 Then
      Instring = Mid(Instring, 2, 2)
      Text5.Text = Instring
   End If
End Sub

Public Function controlla_cheksum() As Integer
   Dim i, l, s, ck As Integer
   l = Len(Instring)
   s = 0
   ck = Asc(Mid(Instring, 4, 1))
   For i = 2 To l - 2
      s = s + Asc(Mid(Instring, i, 1))
   Next
   s = s Mod 256
   If ck = s Then
      controlla_cheksum = 1
   Else
      controlla_cheksum = 0
   End If
End Function

Private Sub opzioni_Click()
   Form2.Show
End Sub
```

```
Private Sub Combo1_Change()

End Sub

Private Sub Command1_Click()
   Unload Me
End Sub

Private Sub Command2_Click()
   Dim n, appoggio, comando As String
   If Combo2.Text = "Minima" Then
      n = 1
   End If
   If Combo2.Text = "Media" Then
      n = 2
   End If
   If Combo2.Text = "Massima" Then
      n = 3
   End If
      If Len(n) = 1 Then
         appoggio = "000" & n
      End If
      If Len(n) = 2 Then
         appoggio = "00" & n
      End If
      If Len(n) = 3 Then
         appoggio = "0" & n
      End If
      If Len(n) = 4 Then
         appoggio = n
      End If
      comando = "Velvert" & appoggio
      Form1.invia (comando)
End Sub

Private Sub Command3_Click()
   Form1.invia ("setbase" & "0000")
End Sub

Private Sub Command4_Click()
   Dim n, appoggio, comando As String
   If Combo3.Text = "Minima" Then
      n = 1
   End If
   If Combo3.Text = "Media" Then
      n = 2
   End If
   If Combo3.Text = "Massima" Then
      n = 3
   End If
      If Len(n) = 1 Then
         appoggio = "000" & n
      End If
      If Len(n) = 2 Then
         appoggio = "00" & n
      End If
      If Len(n) = 3 Then
         appoggio = "0" & n
      End If
      If Len(n) = 4 Then
         appoggio = n
      End If
      comando = "Velbase" & appoggio
      Form1.invia (comando)
End Sub

Private Sub Command5_Click()
```

```
    Form1.invia ("setpinza" & "0000")
End Sub

Private Sub Command6_Click()
    Form1.invia ("repos" & "0000")
End Sub

Private Sub Command7_Click()
    Form1.invia ("setall" & "0000")
End Sub

Private Sub Command8_Click()
    Form1.invia ("setvert" & "0000")
End Sub

Private Sub List1_Click()

End Sub

Private Sub Form_Load()
    Dim minima As String
    Dim media As String
    Dim massima As String
    minima = "Minima"
    media = "Media"
    massima = "Massima"
    Combo2.AddItem (minima)
    Combo2.AddItem (media)
    Combo2.AddItem (massima)
    Combo3.AddItem (minima)
    Combo3.AddItem (media)
    Combo3.AddItem (massima)
End Sub
```

Form 3 - 1

```
Private Sub Command1_Click()
    Unload Me
End Sub

Private Sub Text1_Change()

End Sub
```