UNIVERSITÀ DI BRESCIA
FACOLTÀ DI INGEGNERIA
Dipartimento di Elettronica per l'Automazione

Laboratorio di Robotica Avanzata
Advanced Robotics Laboratory

Corso di Robotica
(Prof. Riccardo Cassinis)

Movimentazione del robot MARMOT

Elaborato d'esame di: **Stefano Rosa, Sergio Sigala**

Consegnato il: **13 luglio 2001**

Sommario

This work implements all the logic and driving circuitry necessary to drive and control one to three bipolar stepper motors. From the user's point of view, these motors are controlled by a series of simple data packets sent to our boards over a standard EIA/TIA-232 serial interface. The physical connection chosen is the classic two wires, plus ground.

In the driving section, we used a Toshiba TA8435 integrated circuit for each motor; this constitutes an interesting improvement over the classic L297-L298 dual-chip alternative. This integrated circuit alone gives us the same control and driving capabilities the L297-L298 couple provides, plus a novel micro-stepping driving feature.

The high-level control board is built around an 8-bit microcontroller, a proud member of the well-known Motorola 68HC11 family. This device communicates to the host system and takes care to generate the velocity ramps necessary to correctly operate the stepper motors.

Our entire assembly will be part of the MARMOT robot, currently under development at the ARL

1. Introduzione

Questo elaborato ha il fine ultimo di donare la capacità di movimento al nuovo robot MARMOT, correntemente in fase di sviluppo e integrazione presso il Laboratorio di Robotica Avanzata. Ci siamo occupati di tutti gli aspetti progettuali e realizzativi coinvolti nell'impresa, esclusa la sola parte meccanica, già abilmente risolta da due ragazzi dell'ambito del loro elaborato dell'anno passato.

2. Il problema affrontato

Consiste nel *rendere mobile il robot MARMOT e controllare la sua velocità di spostamento tramite comandi passati via porta seriale*; a tal fine si sono dovute esaminare e risolvere questioni sia hardware che software.

Gli aspetti hardware coinvolti sono riassumibili essenzialmente nella realizzazione del pilotaggio elettrico dei tre motori a passo di cui MARMOT dispone: è necessario scegliere dei circuiti integrati adatti a questo compito, definire il loro punto di lavoro e infine montarli su opportuni circuiti stampati, assieme ad altri componenti indispensabili al loro funzionamento.

La parte software, invece, dovrà realizzare il controllo a basso livello dei motori, azionandoli secondo le indicazioni ricevute da un sistema di più alto livello; è indispensabile individuare il tipo di microprocessore adatto al compito e le periferiche di I/O richieste nonché, infine, scrivere un programma di governo adatto. Nel variare le velocità dei motori, il programma dovrà seguire delle opportune curve di accelerazione e decelerazione, altrimenti gli alberi dei motori potrebbero perdere passi.

Per concludere, abbiamo dovuto esaminare anche alcuni problemi pratici di connessione e alimentazione dei vari dispositivi coinvolti (motori, schede driver, batteria, scheda di controllo e joystick per comando manuale).

3. La soluzione adottata

Per quanto riguarda gli aspetti hardware di pilotaggio dei motori, la disponibilità di circuiti stampati già realizzati e discretamente adatti al nostro compito ha determinato chiaramente il tipo di circuito integrato da utilizzare, il Toshiba TA8435.

Per risolvere la seconda parte del problema abbiamo adottato come sistema intelligente di controllo una scheda basata sul Motorola 68HC11, la 6811LABE. La scelta di questa particolare scheda e di questa famiglia di microcontrollori è giustificata dalle seguenti osservazioni:

- è stata sviluppata all'interno della nostra università e dunque risulta immediatamente disponibile, compresa la sua documentazione;
- nonostante il gran numero di periferiche integrate e di linee di I/O disponibili è compatta e facilmente adattabile alle nostre esigenze;
- esiste moltissima documentazione di buona qualità sulla famiglia di microcontrollori impiegata e sono reperibili gratuitamente dal sito Motorola sia l'assemblatore che il software di monitor-debugging;
- impiegando pochi componenti alquanto diffusi risulta essere molto economica.

Abbiamo organizzato logicamente la relazione in tre parti, per permettere di analizzare in modo specifico i vari argomenti trattati e per cercare di evitare confusione. Nella prima parte parliamo degli aspetti hardware legati agli azionamenti dei motori e delle connessioni tra le varie schede. Nella seconda diamo una breve panoramica della scheda con il 68HC11; nell'ultima presentiamo il programma scritto per girare sul microcontrollore e spieghiamo in dettaglio il particolare protocollo a pacchetti utilizzato per scambiare dati con il resto del mondo.

4. Note al lettore

> "The time has come," the Walrus said, "to talk of many things."
>
> L. Carroll

Prima di continuare esponiamo alcune definizioni e convenzioni che utilizzeremo estensivamente nel corso di questo elaborato:

- Per quanto riguarda l'identificazione dei segnali attivi bassi abbiamo utilizzato la convenzione Intel, che prevede che essi siano seguiti da un cancelletto, piuttosto che essere soprasegnati da una barra. Esempio: ENABLE# sarà l'equivalente del classico segnale ENABLE negato.
- Il termine *CCCU* l acronimo usato per indicare la Central Communication and Control Unit, vale a dire il mini PC installato a bordo del MARMOT con lo scopo di controllarne il funzionamento ad alto livello.

- La sigla *MCU*, Micro Controller Unit, è spesso usata nella documentazione Motorola per alludere al 68HC11; noi la prenderemo in prestito. A volte ci riferiremo al medesimo dispositivo utilizzando anche i termini processore e microcontrollore. Nel nostro contesto, la sigla MCU può anche essere vantaggiosamente interpretata come Motor Controller Unit.

Per concludere elenchiamo alcuni prerequisiti che è necessario soddisfare per proseguire con profitto la lettura:

- necessario che si conosca un minimo dell architettura della famiglia di processori 68HC11;
- serve qualche conoscenza di carattere generale sui motori a passo e sul mondo delle comunicazioni seriali.

5. Struttura del MARMOT

Questa sezione riassume brevemente le caratteristiche fisiche principali del robot, in modo che il lettore possa acquisire un minimo di confidenza con esso. In particolare, si pone l'accento sull'identificazione dei vari blocchi che compongono il suo sottosistema di movimento.

5.1. Le ruote svedesi

Il robot MARMOT (Mobile Advanced Robot for Multiple Office Tasks) è costruito attorno ad un telaio d'alluminio a forma di prisma a base triangolare con gli angoli appiattiti per ospitare gli organi di movimento. La mobilità è assicurata da tre motori a passo, fissati nella parte superiore del robot, che trasmettono il moto a tre ruote *svedesi* tramite delle cinghie e alcune puleggie dentate; lo schema della struttura vista dall'alto è visibile in Figura 1.

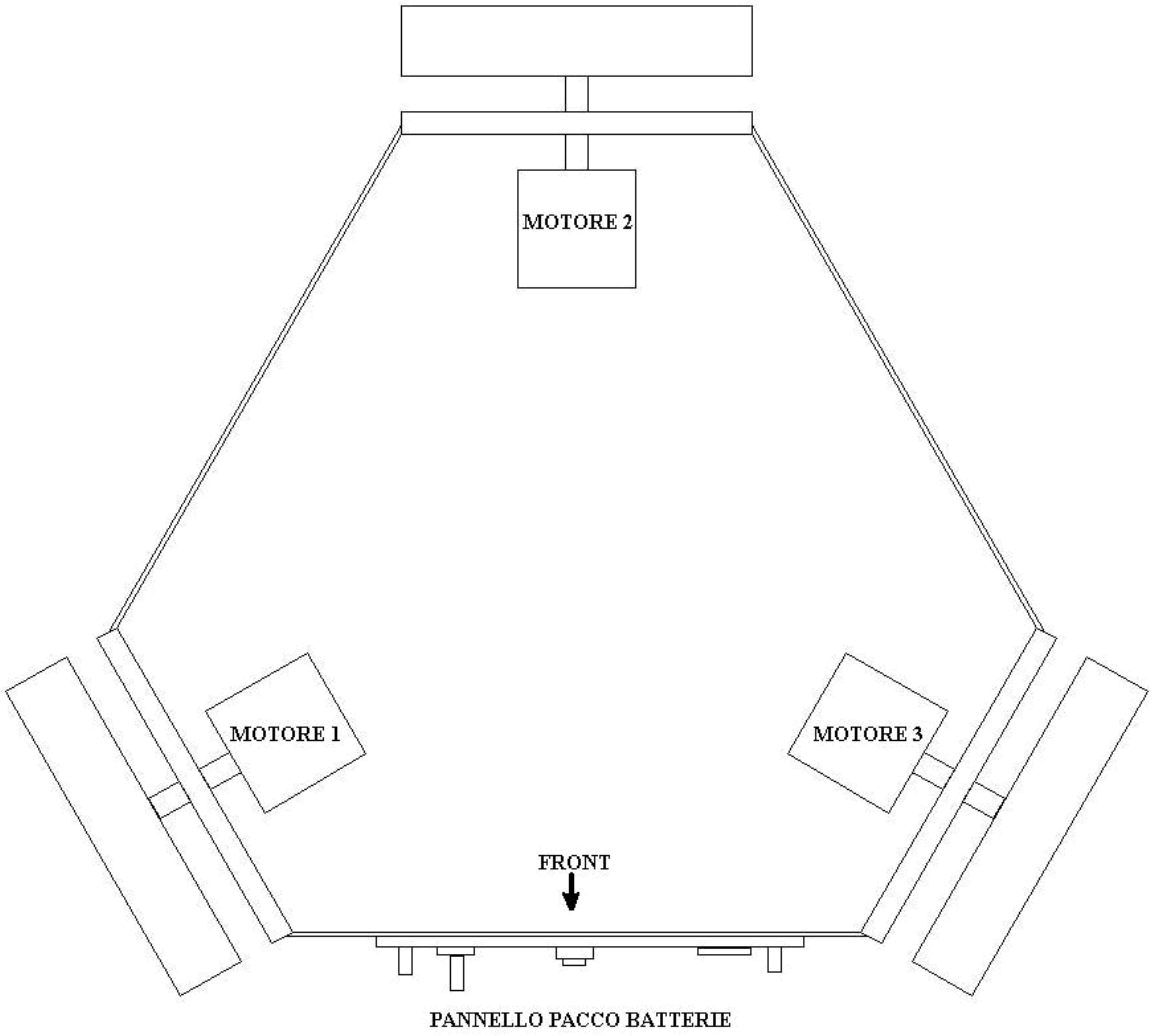

Figura 1: vista dall'alto del MARMOT

Le ruote svedesi consentono contemporaneamente la trazione sul terreno nella direzione perpendicolare al loro asse e la traslazione nella direzione parallela: questo permette ad un robot come il MARMOT, le cui tre ruote sono disposte con angoli di 60° tra gli assi, di potersi muovere agilmente anche in direzioni oblique senza dover effettuare cambi di direzione sul posto, come sarebbe necessario per esempio con altre soluzioni quali differential-drive oppure sistemi a quattro ruote. Questa particolare tecnologia consente quindi una maggiore agilità, soprattutto negli spazi ristretti in cui pareti o altri ostacoli renderebbero difficoltose le manovre d'inversione per altri tipi di robot.

5.2. Individuazione dei motori

Vista la simmetria del MARMOT, si è deciso convenzionalmente di indicare come parte anteriore, o front, il lato che ospita il pacco batterie e di numerare i motori come descritto nella figura. Stabiliamo che le etichette M1, M2 e M3 identificheranno il primo, il secondo ed il terzo motore, rispettivamente. Inoltre, i segnali CW/CCW# di direzione dei motori si riferiranno all'effettivo verso di rotazione dei rispettivi alberi se visti dall'interno del robot verso l'esterno. Ricordiamo che il significato di CW è clock wise, mentre CCW vuole dire counter clock wise; dunque se il segnale ha livello logico

1 il motore collegato ruoterà in senso orario, viceversa, se è 0, l'albero dovrà ruotare in senso antiorario.

5.3. L'alimentazione

L'alimentazione a 24Vcc è fornita al robot attraverso due accumulatori da 12V connessi in serie, alloggiati all'interno del suo corpo e facilmente estraibili mediante una slitta. Sul pannello della slitta, coincidente con la parte frontale del robot, sono presenti l'interruttore generale d'alimentazione con relativo led verde d'indicazione, un porta fusibile e un connettore AXR/XLR femmina a 3 poli per il collegamento del carica batteria; accanto vi è un led rosso di segnalazione carica in corso. I classici 5V, necessari per alimentare le logiche della MCU e della CCCU, sono ricavati dai 24V attraverso l'impiego di un convertitore DC/DC di tipo switching, montato su un fianco del robot.

La Figura 2 illustra il circuito del pacco batterie e la morsettiera principale di distribuzione dell'alimentazione, montata accanto al convertitore DC/DC e alla quale fanno capo le alimentazioni di tutti i sottosistemi. Nella parte superiore del robot sono montati due ripiani per alloggiare la strumentazione elettronica di controllo e vari connettori. La Figura 3 riporta il cablaggio della scatola che contiene la CCCU.

5.4. La scatola Power Control

Lo schema elettrico riportato nella Figura 4 mostra il circuito contenuto nella scatoletta nera denominata Power Control, che si occupa di sezionare le alimentazioni che giungono ai driver dei tre motori, alla MCU e alla CCCU. Sono presenti due interruttori, due pulsanti e tre led; i loro compiti sono illustrati di seguito.

- L'interruttore MAIN attiva l'alimentazione generale a 5V denominata +5V MAIN, prelevandola direttamente dall'uscita dell'alimentatore switching che, come conseguenza, è stata chiamata +5V UNSWITCHED. A quest'interruttore è associato il led verde Main.

- L'interruttore AUX fornisce tensione alla linea +5V AUX che alimenta il microcontrollore, prelevandola dalla linea +5V MAIN; ad esso è associato il led verde Aux.

- I pulsanti ON (nero) e OFF (rosso), collegati alla bobina del relè, ne governano il funzionamento in modalità autoritenuta. Il relè, attraverso il parallelo di tre dei suoi quattro contatti, alimenta i driver dei motori e le quattro ventole di raffreddamento. Durante il funzionamento s'illumina un led giallo lampeggiante. Il pulsante OFF è d'emergenza e diseccita il relè, togliendo l'alimentazione sia ai motori che alle ventole.

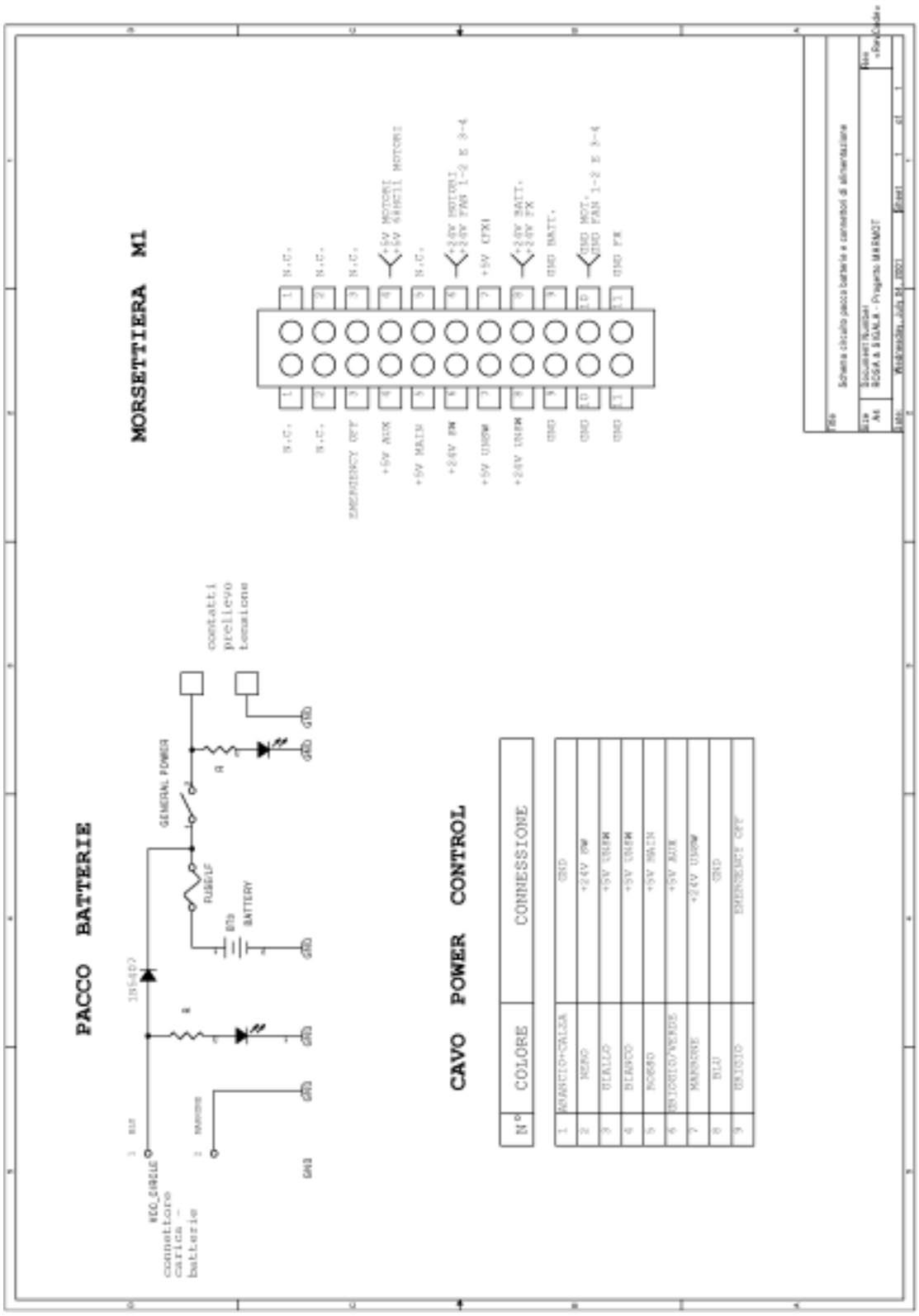

Figura 2: pacco batterie e morsettiera principale d'alimentazione

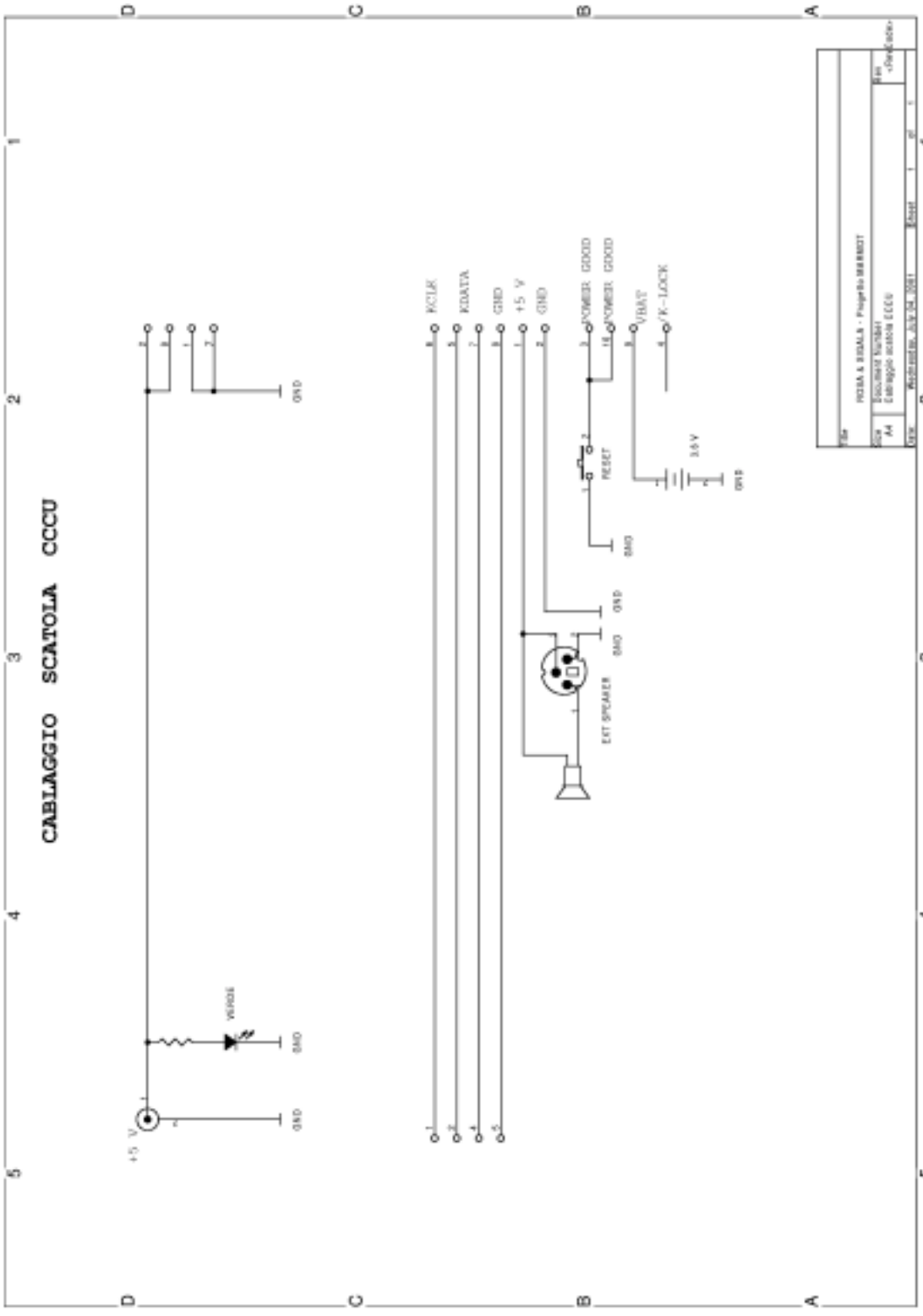

Figura 3: cablaggio della scatola che contiene la CCCU

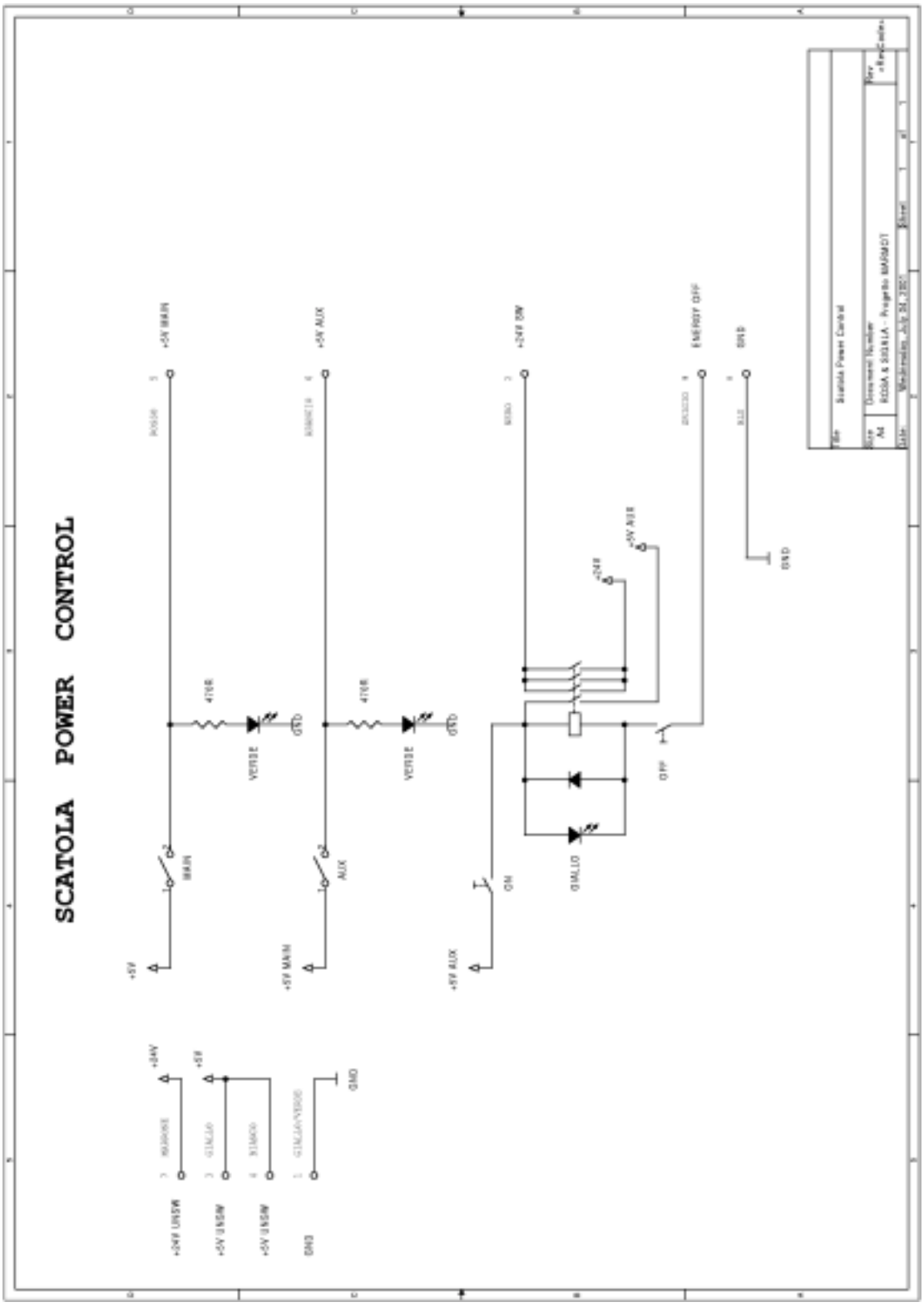

Figura 4: circuito della scatola Power Control

6. Le schede driver

Si tratta dei tre circuiti stampati che alloggiano i driver di potenza per i motori a passo, facilmente individuabili data la loro vicinanza ai rispettivi motori controllati.

Ogni scheda contiene un driver di potenza in grado di alimentare un motore a passo bipolare. Integra la logica di controllo necessaria a generare le corrette forme d'onda di pilotaggio degli avvolgimenti del motore, partendo da segnali d'ingresso logici quali clock e senso di rotazione. Pu funzionare anche in modalit *micro-stepping*.

6.1. Descrizione

Le parti di controllo e potenza sono entrambe contenute nel circuito integrato Toshiba TA8435, che svolge le stesse funzioni della nota coppia L297-L298; inoltre consente il pilotaggio del motore in modalit micro-stepping, tecnica che il L297 non supporta.

- Il dip-switch doppio permette di impostare i livelli logici dei segnali M1 e M2, selezionando quindi la modalit di funzionamento tra quattro possibili, come descritto nel data sheet. La modalit pu essere cambiata anche a run-time, sotto certe condizioni.
- Lo LM555 configurato come multivibratore astabile, abilitato nel funzionamento quando l'ingresso RUN alto. E' una possibilit che evita di pilotare il motore mandando direttamente gli impulsi di clock alla scheda; utile per solo negli impieghi in cui non richiesta precisione di posizionamento dell'albero motore.
- Gli ingressi CK1, CK2 e CW/CCW# determinano congiuntamente la rotazione e il senso di marcia del motore.
- L'ingresso ENABLE# disabilita i driver, se alto.

L'ingresso logico REFIN seleziona la corrente d intervento dei chopper interni al driver, *tra due valori possibili*. Le correnti effettive sono determinate dai due gruppi di tre resistenze in parallelo collegate ai pin NFA e NFB del TA8435. L'ingresso REFIN utile per diminuire la corrente di pilotaggio del motore, quando questo fermo.

Lo TA8435 integra otto diodi di protezione dalle correnti di ritorno, indotte sugli avvolgimenti del motore quando i driver di potenza commutano. I quattro diodi esterni risultano in parallelo coi quattro collegati all'interno del dispositivo verso massa, poich questi ultimi sono ricavati sul substrato del chip e pertanto sono poco robusti rispetto agli altri quattro interni collegati verso l'alimentazione positiva.

Le reti LC all'uscita costituiscono dei filtri passa basso che dovrebbero sopprimere le componenti ad alta frequenza presenti sui segnali di pilotaggio delle fasi, migliorando la compatibilit elettromagnetica del sistema. Per approfondire il funzionamento del TA8435 rimandiamo il lettore alla consultazione del suo datasheet.

La scheda prevedeva un'alimentazione in corrente alternata ai morsetti VA e VB, con circuiti di raddrizzamento e filtraggio a bordo. Il BJT BD241C, montato assieme al diodo Zener in configurazione generatore di tensione costante, alimenta la logica con una tensione di 5V circa. La Figura 5 illustra lo schema elettrico della scheda gi montata che avevamo a disposizione.

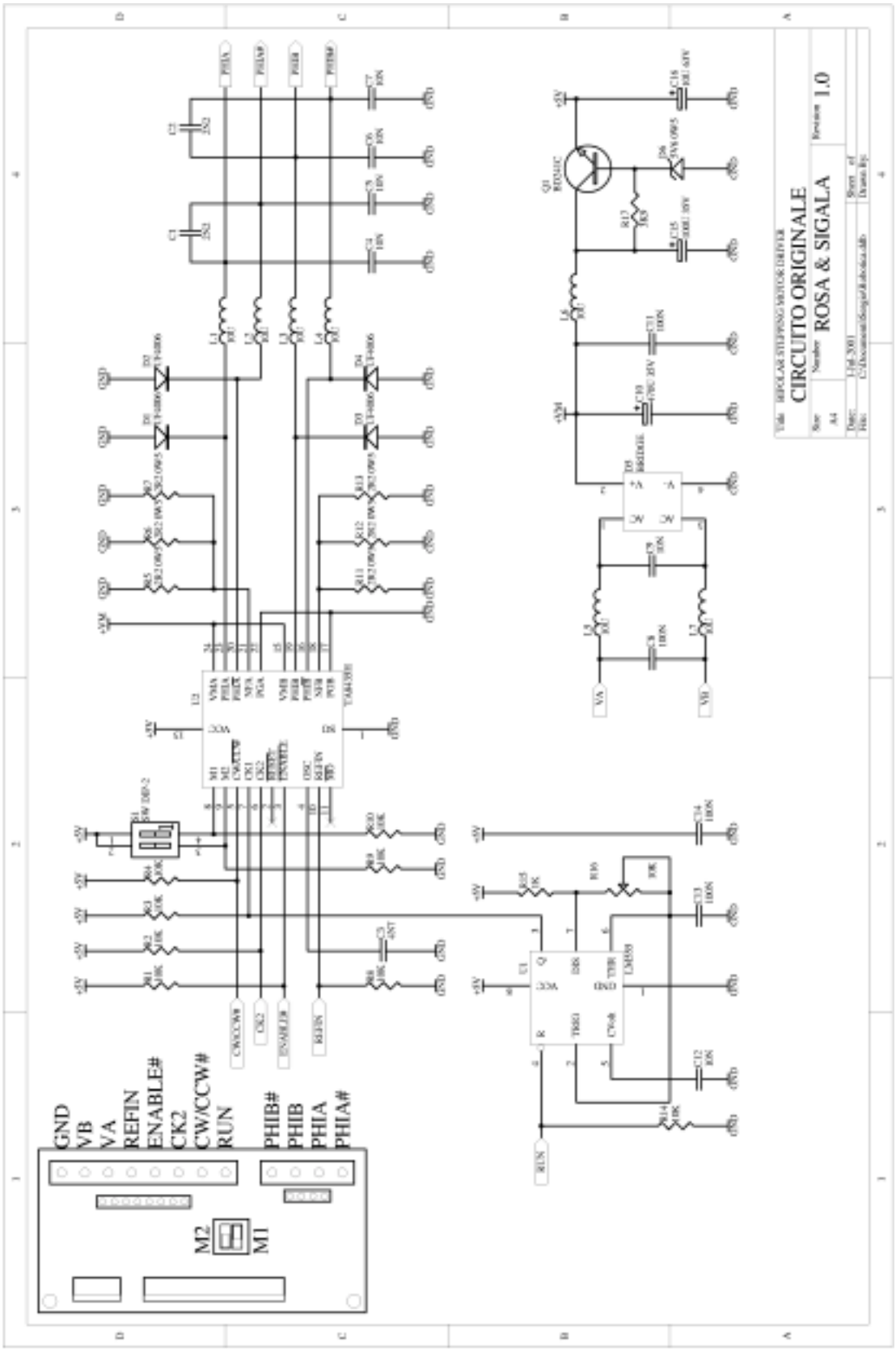

Figura 5: circuito originale delle schede driver

6.2. Il nostro uso

Sono state apportate diverse modifiche al circuito originario, descritte in dettaglio di seguito.

- Poich introduciamo il segnale CK1 dall'esterno, lo LM555 non pi necessario e dunque non montato. Quindi prevediamo un collegamento tra i suoi pin 3 e 4, in modo da garantire che il segnale CK1 d ingresso al TA8435 sia fisicamente accessibile dalla morsettiera.
- Noi disponiamo di un'alimentazione in corrente continua a 24V circa, perci il ponte raddrizzatore ed il doppio filtro passa basso a pigreco posti sull'ingresso dell'alimentazione sono stati rimossi. L'alimentazione positiva a 24V deve arrivare al morsetto VA.
- Al fine di minimizzare la potenza dissipata in calore dalle schede, stato eliminato il regolatore a 5V, portando dall'esterno la tensione a 5V necessaria per l'alimentazione della logica. Questa tensione ricavata in modo particolarmente efficiente dai 24V utilizzando il modulo convertitore DC/DC di tipo switching gi accennato in precedenza. L'alimentazione a 5V deve pervenire al morsetto VB.
- Sono state rimosse le quattro reti LC all uscita; i segnali che alimentano i motori raggiungono direttamente la morsettiera attraverso dei ponticelli.

6.3. La morsettiera

In definitiva, applicate le nostre modifiche, la configurazione finale delle due morsettiere quella illustrata nella Figura 6 (la prima morsettiera occupa i primi quattro contatti, la seconda i restanti otto). Il circuito elettrico risultante è mostrato nella Figura 7.

$\phi\overline{A}$	ϕA	ϕB	$\phi\overline{B}$	CK1	CW/CCW#	CK2	ENABLE #	REFIN	+24V	+5V	GND

Figura 6: assegnamento morsettiere delle schede driver

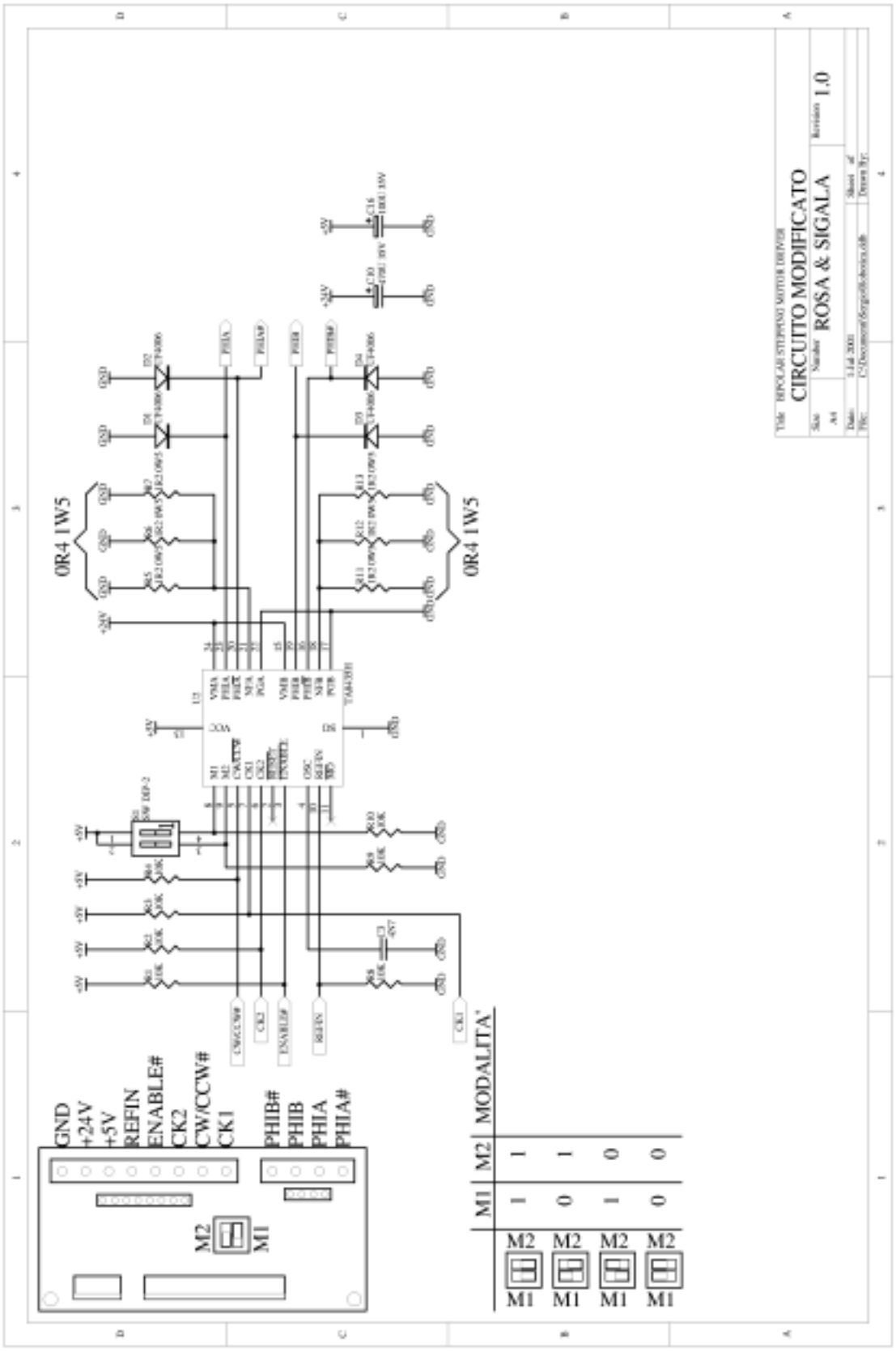

Figura 7: circuito modificato delle schede driver

7. La scheda 6811LABE

Le velocità angolari dei motori a passo sono controllate variando le frequenze di tre segnali indipendenti ad onda quadra prodotti internamente al 68HC11, ognuno dei quali raggiunge il rispettivo driver TA8435. La direzione di ciascun motore si determina assegnando opportunamente il livello logico della corrispondente linea CW/CCW#. Tutti questi segnali sono generati dalla MCU della scheda 6811LABE che contemporaneamente colloquia con la CCCU tramite la sua interfaccia seriale.

Come spiegato nell'introduzione, la scelta è caduta su questa particolare scheda piuttosto che su altre più performanti per vari motivi di convenienza riconducibili sia al suo basso costo che al fatto che è facilmente reperibile.

7.1. Caratteristiche

La serigrafia della scheda è visibile in Figura 8. La scheda ha dimensioni standard Eurocard (160mm x 100 mm) e la metà destra è una millefori disponibile per realizzare montaggi sperimentali.

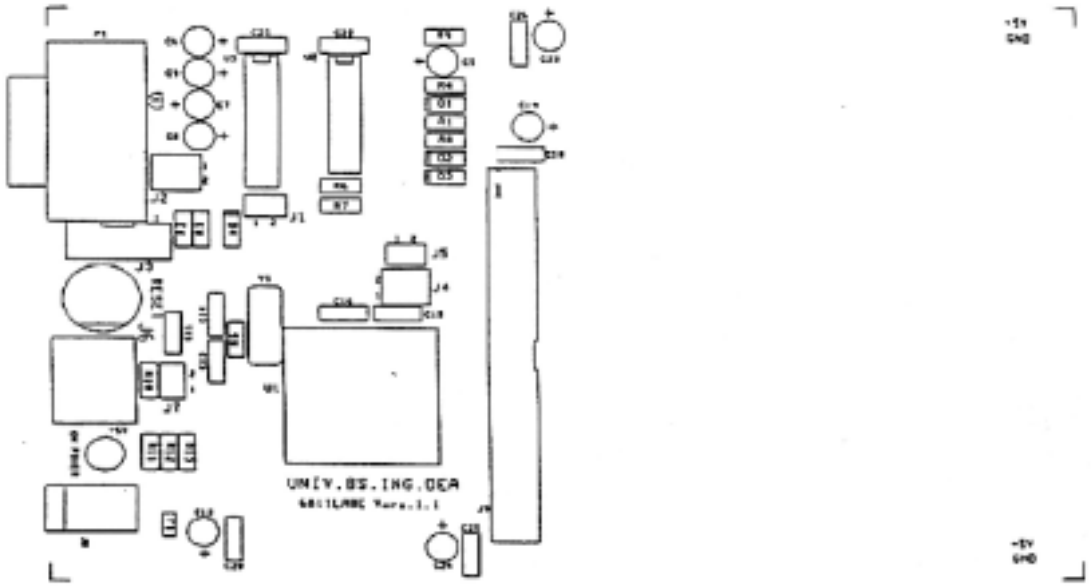

Figura 8: serigrafia della 6811LABE

Sulla scheda è presente la circuiteria con quarzo a 8MHz necessaria a produrre il segnale di clock, un driver MAX232 per la conversione tra livelli logici HCMOS e livelli per linee seriali standard (U3), alcuni condensatori di filtro sulle alimentazioni per eliminare i disturbi, diversi jumper di configurazione ed il circuito di reset con relativo pulsante d'attivazione manuale. La scheda è predisposta per ospitare un connettore a 50 poli per prelevare i segnali dai pin della MCU (J9 in figura), possibilità da noi effettivamente sfruttata.

Per comodità del lettore, nella Tabella 1 è riportato l'elenco completo dei componenti presenti sulla scheda.

Item number	Quantity	Reference	Value
1	5	C5, C6, C7, C8, C9	10µF
2	2	C11, C15	1µF
3	4	C12, C19, C23, C24	22µF
4	2	C13, C14	22pF
5	7	C16, C18, C20, C21, C22, C25, C26	100Nf
6	4	D1, D2, D3, D4	1N4148
7	1	D5	LED
8	1	F1	FUSE
9	3	J1, J5, J7	CON2AP
10	2	J2, J4	CON4AP
11	1	J3	CON10AP
12	1	J6	CON3
13	1	J8	BNC
14	1	J9	CON50A
15	1	P1	CONNECTOR DB9
16	6	R1, R4, R7, R9, R12, R13	10K
17	2	R2, R8	1K
18	1	R3	100K
19	1	R5	10M
20	1	R6	10
21	1	R10	100
22	1	R11	470
23	1	S1	RESET
24	1	U1	68HC811E2
25	1	U2	74HC132
26	1	U3	MAX232
27	1	Y1	8MHz

Tabella 1: lista componenti della 6811LABE

7.2. Configurazione della scheda

Nella Tabella 2 è riportato l'elenco dei jumper presenti sulla scheda e la corrispondente funzione. Per l'individuazione sulla scheda si faccia riferimento alla serigrafia riportata nella Figura 8 all'inizio di questa sezione.

J1	Se chiuso consente in modo bootstrap di attivare automaticamente il programma memorizzato in EEPROM inibendo la seriale per 100ms dopo il reset (solo MC68HC11A1). Stato: aperto
J2	1-2 chiuso e 3-4 chiuso → cavo seriale RS232 dritto (DCE-DTE) 1-3 chiuso e 2-4 chiuso → cavo seriale RS232 null modem (DTE-DTE) Altre configurazioni sono proibite. Stato: 1-3 e 2-4 chiusi
J3	Consente di predisporre dei consensi automatici sul canale seriale RS232. Stato: 1-2 chiuso, 3-4 chiuso, 7-8 chiuso e 9-10 chiuso
J4	Predispone le tensioni di riferimento per il convertitore A/D (VRH=+5V, VRL=GND). 1-2 chiuso → connette VRL e GND 3-4 chiuso → connette VRH a +5V attraverso una resistenza da 10Ω Stato: 1-2 chiuso e 3-4 chiuso
J5	**Se chiuso configura il modo di funzionamento *special bootstrap*; se aperto configura il modo di funzionamento *single chip* (non consentito per MC68HC11A1).** **Stato: chiuso in programmazione, aperto in funzionamento (run)**
J6	Se chiuso connette il pin XIRQ al morsetto 3 (+10V) attraverso una resistenza da 100Ω (solo per programmazione EPROM del dispositivo MC68HC711E9). Stato: aperto

Tabella 2: descrizione dei jumper della 6811LABE

Il jumper più importante è il J5, che seleziona la modalità di funzionamento del processore tra special bootstrap e single chip; vedere il capitolo successivo oppure la documentazione Motorola citata nella bibliografia per altre informazioni in merito. **Ricordarsi di predisporre opportunamente questo jumper prima di operare**. Le posizioni degli altri jumper non dovrebbero avere bisogno di modifiche.

7.3. Tipi di MCU installabili sulla 6811LABE

Come visibile nella serigrafia della scheda (Figura 8), il microcontrollore U1 deve essere in package PLCC, plastic leaded chip carrier. La scheda sperimentale è compatibile con le versioni PLCC a 52 pin dei tre microcontrollori riportati in Tabella 3.

MC68HC11A1	dispositivo con 512 byte di EEPROM e 256 byte di RAM; modello base della famiglia 68HC11, può essere usato solo in modo special bootstrap
MC68HC711E9	dispositivo con 512 byte di EEPROM, 256 byte di RAM e 12kB di EPROM; necessita per la programmazione di 10V sul pin XIRQ, collegato alla morsettiera di alimentazione della scheda 6811LABE
MC68HC811E2	dispositivo con 2kB di EEPROM e 256 byte di RAM

Tabella 3: MCU installabili sulla 6811LABE

I tre dispositivi si differenziano soltanto per i vari tipi di memoria integrati e le rispettive dimensioni; la dotazione di periferiche interne e linee di I/O è la medesima. Con un normale clock a 8MHz il tempo di ciclo è di 500ns, poiché viene ricavato internamente da una divisione per quattro del segnale di clock.

Tra i tre processori abbiamo scelto il 68HC811E2, data la necessità di inserirvi un programma di oltre 1kB e la possibilità di riscritture frequenti offerta dalla sua EEPROM, estremamente utile durante la fase di sviluppo del programma. Il 68HC11A1 non può essere usato perché poco capiente; il 68HC711E9 potrebbe essere utilizzato, ma solo quando non vi saranno più modifiche da apportare al programma oppure diverranno molto rare.

7.4. Dove reperire altre informazioni

Come spiegato nella parte iniziale della relazione, si suppone che il lettore abbia qualche conoscenza della famiglia di microcontrollori Motorola 68HC11, sia per quanto riguarda il set di istruzioni del loro linguaggio di programmazione che dal punto di vista dei dispositivi di I/O che integrano.

Per acquisire queste conoscenze consigliabile procurarsi almeno il 68HC11 Reference Manual; sono altres molto utili anche le varie *application note*, i *data sheet* e gli *engineering bullettin* che hanno in qualche modo a che fare con questi processori. Tutti questi manuali e documenti sono reperibili dopo qualche ricerca al sito della Motorola www.mcu.motsps.com, dedicato esclusivamente ai microcontrollori prodotti dall azienda come pure agli strumenti gratuiti di sviluppo software. Nella bibliografia sono elencati quelli che sono stati consultati da noi.

Pu anche essere utile avere sotto mano la documentazione che la Prof.ssa Alessandra Flammini ha redatto per la scheda 6811LABE, sviluppata presso il Laboratorio di Elettronica del Dipartimento di Elettronica per l'Automazione e della quale come detto impiegamo un esemplare nel nostro sistema. Parte della documentazione riprodotta senza permesso qui.

8. La MCU 68HC811E2

Membro interessante della famiglia dei 68HC11, ha la caratteristica peculiare di possedere 2kB di EEPROM per contenere il programma utente, contro i 512 byte usuali. Lo schema a blocchi interno del chip è riportato nella Figura 9.

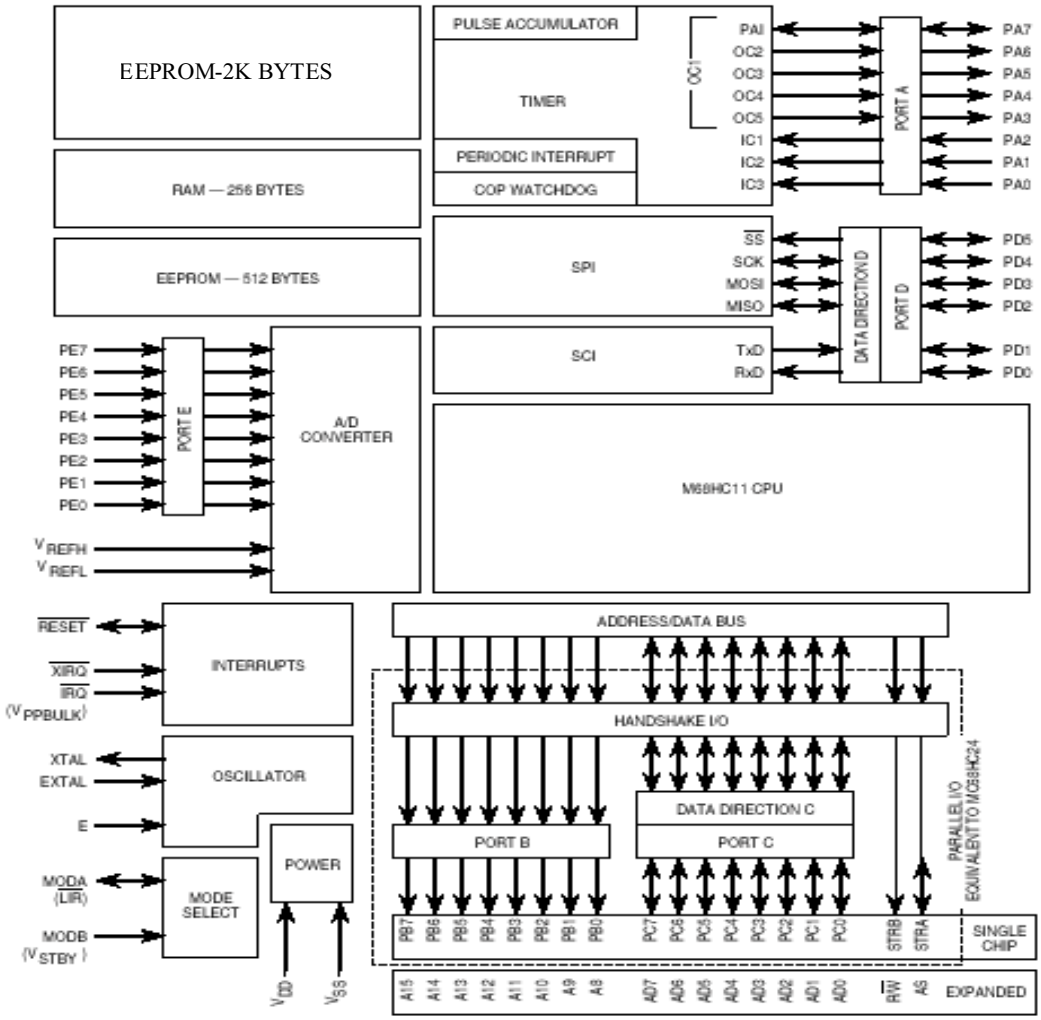

Figura 9: schema a blocchi del 68HC811E2

Il microcontrollore possiede queste caratteristiche:

- 256 byte di RAM alle locazioni **0000h-00FFh** e 2kB di EEPROM alle locazioni **F800h-FFFFh.**
- 1 timer a 16 bit free running pilotato a 2MHz (se il clock è a 8MHz), con linee di input capture (ICx) e output compare (OCx) accessibili dall'esterno. Si ha input capture quando, in corrispondenza della commutazione di un ingresso logico, il valore del timer viene memorizzato in un registro senza alterazione del program counter. Si ha output compare quando è possibile programmare la variazione di un'uscita logica in corrispondenza di un valore (programmabile) assunto dal timer senza alterazioni del program counter.
- 1 timer/counter a 8 bit (counter = pulse accumulator).
- circuiteria di watchdog (COP, Computer Operating Properly) e di sorveglianza del segnale di clock. Un circuito di watchdog consiste in un sistema retriggerabile: se il meccanismo non viene retriggerato (tramite scrittura in un opportuno registro interno) per un periodo superiore al tempo prestabilito di timeout si attiva automaticamente il circuito di reset.

- 1 interfaccia seriale asincrona (SCI, Serial Communication Interface).
- 1 interfaccia seriale sincrona (SPI, Serial Peripheral Interface) dedicata all'interfacciamento di periferiche ad accesso seriale (EEPROM, convertitori A/D o D/A, display ed altro ancora).
- 1 convertitore A/D a 8 bit con 8 canali di ingresso multiplexati con alimentazione separata. Il tempo di conversione per canale è di 16µs.
- 1 linea esterna di interrupt mascherabile.
- 1 linea esterna di interrupt non mascherabile.
- Cinque port paralleli configurabili come segue:
  - PORTA: 8 bit di ingresso/uscita, con 4 linee di uscita, 3 di ingresso e una bidirezionale; le linee possono essere collegate ai timer interni ed utilizzate in modalità input capture o output compare;
  - PORTB: 8 bit in sola uscita;
  - PORTC: 8 bit configurabili individualmente come ingresso o come uscita;
  - PORTD: raggruppa le 6 linee dedicate alle unità seriali SCI (asincrona) e SPI (sincrona) che, se non utilizzate, diventano 6 normali linee di I/O indipendenti;
  - PORTE: raggruppa 8 ingressi analogici che fanno capo al convertitore analogico/digitale ad 8 bit integrato nel chip.

Il microcontrollore richiede un clock esterno a 8MHz per il funzionamento ma le operazioni vengono sincronizzate con il segnale E a 2MHz, generato internamente a partire dal clock tramite divisione per quattro; l'alimentazione del chip può variare tra 0.5 e 7V mentre l'assorbimento è di circa 20mA.

8.1. Modi di funzionamento

Vi sono quattro possibili modi di funzionamento, selezionabili tramite gli ingressi MODA e MODB del microcontrollore, come illustrato nella Tabella 4.

MODA	MODB	Modo di funzionamento
0	0	**special bootstrap**
0	1	single chip (disabilitato nella versione A1 della MCU)
1	0	**single chip**
1	1	expanded multiplexed

Tabella 4: modi di funzionamento della MCU

Nel modo **special bootstrap** al reset viene caricato un programma monitor chiamato *talker* residente in ROM che consente la comunicazione tramite linea seriale ad un programma di debugging in esecuzione su un PC, solitamente il PCbug11. In questo modo è possibile scrivere agevolmente la EEPROM ed effettuare il debugging online dei programmi.

Nel modo **single chip** si ha funzionamento embedded; al reset viene caricato nel program counter il contenuto delle locazioni FFFE-FFFFh, che nella versione 68HC811E2 della MCU sono le ultime due celle della EEPROM. In tali locazioni va precedentemente memorizzato il vettore di reset, in altre parole l'indirizzo della prima cella di EEPROM in cui ha inizio il programma utente.

In modo expanded multiplexed il microcontrollore lavora allo stesso modo di un microprocessore che necessita di memoria esterna per il suo funzionamento; le memorie interne divengono infatti invisibili. In questa situazione i bus indirizzi e dati vengono connessi con alcune linee di I/O: il PORTB rappresenta la parte alta del bus indirizzi mentre il PORTC multiplexa la parte bassa del bus indirizzi e il bus dati.

8.2. Note importanti

Poiché il programma che gira sulla MCU comprende quattro routine che lavorano sotto interrupt, *è necessario selezionare la modalità di funzionamento single chip prima di eseguirlo*. La modalità special bootstrap non può essere utilizzata perché vi sarebbero conflitti di allocazione dei quattro vettori di interrupt; in particolare il vettore che gestisce la porta di comunicazione seriale sarebbe conteso dal programma monitor.

Dal punto di vista del programmatore, questa situazione di appropriamento dell'interfaccia seriale della MCU da parte del programma sotto test si manifesta nell'*impossibilità di svolgere il debugging online* del programma utilizzando il classico software PCbug11.

Il caricamento del programma nel microcontrollore deve continuare ad avvenire attraverso l'uso del PCbug11 in modalità special bootstrap.

Per maggiori dettagli riguardo alle modalità di funzionamento si rimanda alla consultazione del M68HC11 Reference Manual.

9. Connessioni e cablaggi

Dopo aver assemblato le schede driver ed averle sistemate sul robot, si è provveduto ad assegnare ai vari segnali di controllo delle stesse una precisa collocazione sul connettore J1 e, soprattutto, né è stata definita una chiara corrispondenza con i pin del 68HC11, specificata di seguito.

9.1. Assegnamento delle porte della MCU

Come spiegato nel capitolo di descrizione delle schede driver per i motori a passo, ogni TA8435 viene controllato da un segnale a onda quadra sulla linea CK1 e dai quattro segnali logici CK2, CW/CCW#, REFIN e ENABLE#. I segnali a onda quadra vengono generati sfruttando gli output compare OC2, OC3 e OC4 abbinati rispettivamente ai bit PA6, PA5 e PA4 del PORTA della MCU; gli altri segnali di controllo vengono prodotti sui port paralleli PORTB e PORTC, quest'ultimo configurato come uscita tramite il registro DDRC del microcontrollore.

Il segnale REFIN è comune a tutti i driver e seleziona l'intensità di corrente che percorre le fasi di ciascun motore. *Quando almeno un motore è in rotazione viene scelta la massima corrente*; per contro, *si richiede la minima corrente quando tutti i motori sono fermi*, diminuendo così il consumo di energia.

Il fatto di comandare i segnali ENABLE# (attivi bassi) tramite la MCU assicura che i motori siano alimentati solamente se è in funzione anche il microcontrollore. Inoltre, attraverso comandi opportuni, la CCCU può governare lo stato delle singole linee ENABLE#, *fornendo o togliendo alimentazione ai motori in modo selettivo*.

Sono stati riservati gli ingressi analogici PE0 e PE1 per collegarvi un joystick analogico, attraverso il quale è possibile governare manualmente i movimenti del robot. La linea PE0 acquisisce la tensione sul cursore del potenziometro che trasduce il movimento orizzontale, mentre PE1 quella sul cursore del potenziometro relativo al movimento verticale. Il programma dimostrativo che gira sul PC passa in "modalità joystick" se viene lanciato senza parametri sulla linea di comando: l'utente potrà quindi manovrare il robot manipolando il joystick; vedere gli ultimi capitoli per ulteriori informazioni in proposito.

L'assegnamento delle linee di I/O general purpose del microcontrollore è riassunto schematicamente nella Tabella 5.

Segnale logico	**Assegnamento sulla MCU**
CK1 M1	PA6
CK1 M2	PA5
CK1 M3	PA4
CK2 M1	PC3
CK2 M2	PC4
CK2 M3	PC5
ENABLE# M1	PC0
ENABLE# M2	PC1
ENABLE# M3	PC2
REFIN M1,M2,M3	PC6
CW/CCW# M1	PB0
CW/CCW# M2	PB1
CW/CCW# M3	PB2
Joystick asse orizzontale	PE0
Joystick asse verticale	PE1

Tabella 5: assegnamenti pin di I/O della MCU

9.2. Linee di I/O non utilizzate

Rimangono libere per assolvere altri compiti queste linee di I/O general purpose:

- 4 linee (PA0, PA1, PA2 e PA7) sulla PORTA;
- 5 linee (PB3, PB4, PB5, PB6 e PB7) sulla PORTB;
- 1 linea (PC7) sulla PORTC;

- 4 linee (PD2, PD3, PD4 e PD5) sulla PORTD;
- 6 canali A/D (PE2, PE3, PE4, PE5, PE6, PE7) sulla PORTE, eventualmente anche PE0 e PE1 se il joystick non viene utilizzato.

9.3. Dettaglio dei connettori

I due schemi riportati nella Figura 10 e nella Figura 11 illustrano con precisione gli assegnamenti sul connettore per flat cable a 50 poli della 6811LABE e sui connettori a vaschetta J1 e J2, da 25 e 9 pin rispettivamente.

Dal connettore a 50 poli montato sulla 6811LABE parte un cavo piatto che trasporta tutti i segnali general purpose della MCU verso i connettori J1 e J2. Attraverso J1 e J2 i segnali vengono smistati ai vari dispositivi.

Al connettore J1 sono connessi tre cavi identici provenienti dai tre driver dei motori a passo; ognuno possiede otto conduttori più la calza di schermo. Di ciascun cavo solo cinque fili sono utilizzati per pilotare i segnali CK1, CK2, CW/CCW#, REFIN ed ENABLE# del rispettivo driver; gli altri tre sono inutilizzati e collegati a massa assieme allo schermo, ma solo dalla parte di J1 per non creare inutili anelli di massa.

Al connettore J2 sono invece collegati i quattro canali analogici PE0, PE1, PE2 e PE3, oltre che l'alimentazione +5V AUX e GND. Il joystick utilizza solamente i segnali PE0 e PE1, corrispondenti ai convertitori ADC1 e ADC2; gli altri sono resi disponibili in ogni caso per eventuali utilizzi futuri.

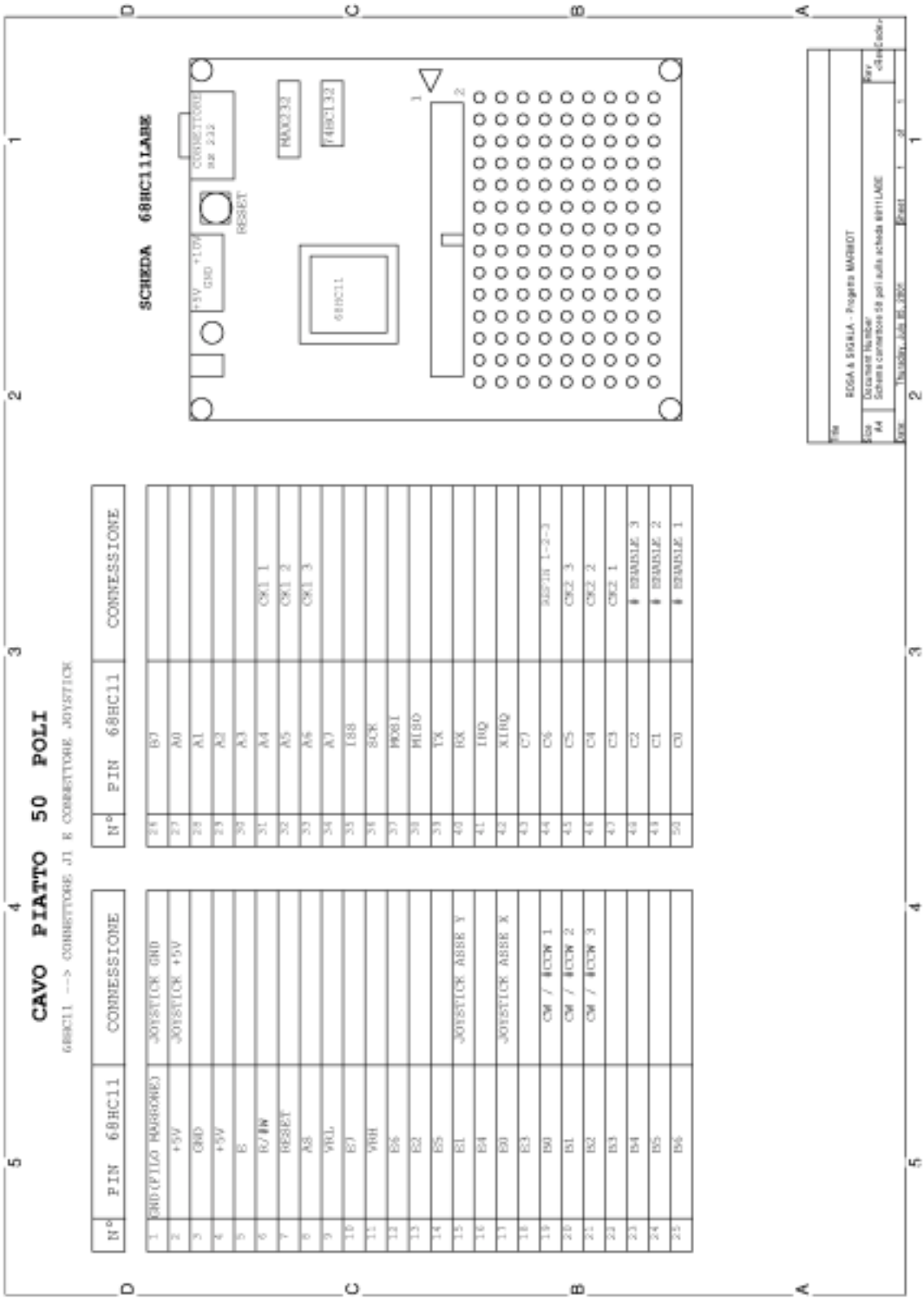

Figura 10: assegnamenti sul connettore a 50 poli

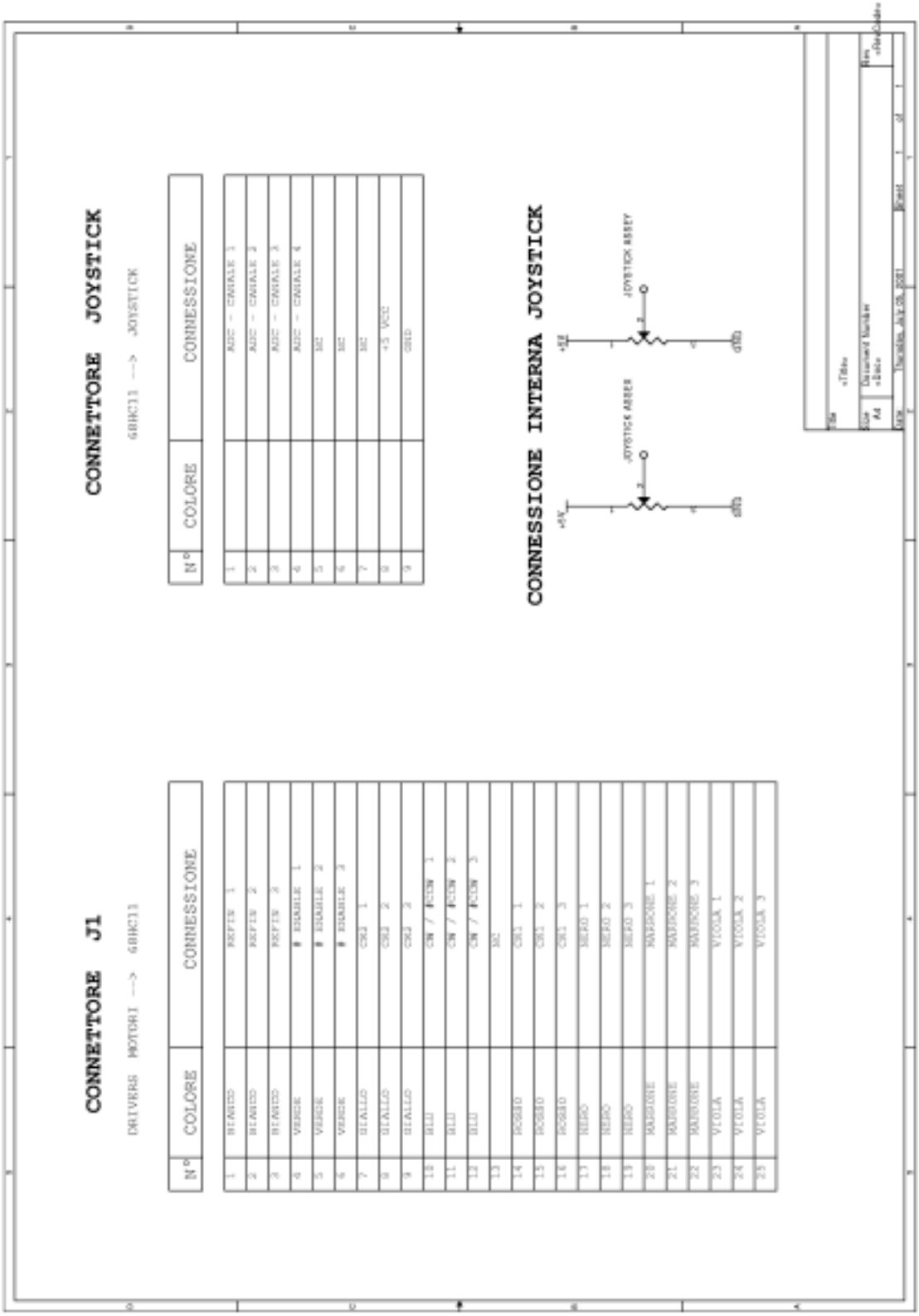

Figura 11: assegnamenti sui connettori J1 e J2

9.4. Disposizione fisica dei connettori

Sulla sommità del robot è stato fissato un ripiano circolare bianco per ospitare le unità di controllo e la strumentazione elettronica che verrà aggiunta in seguito. Nella Figura 12 è riportato uno schizzo del ripiano con indicati la scheda 6811LABE con a bordo la MCU, i connettori J1 e J2, la scatola power control nonché le posizioni delle schede driver dei tre motori a passo.

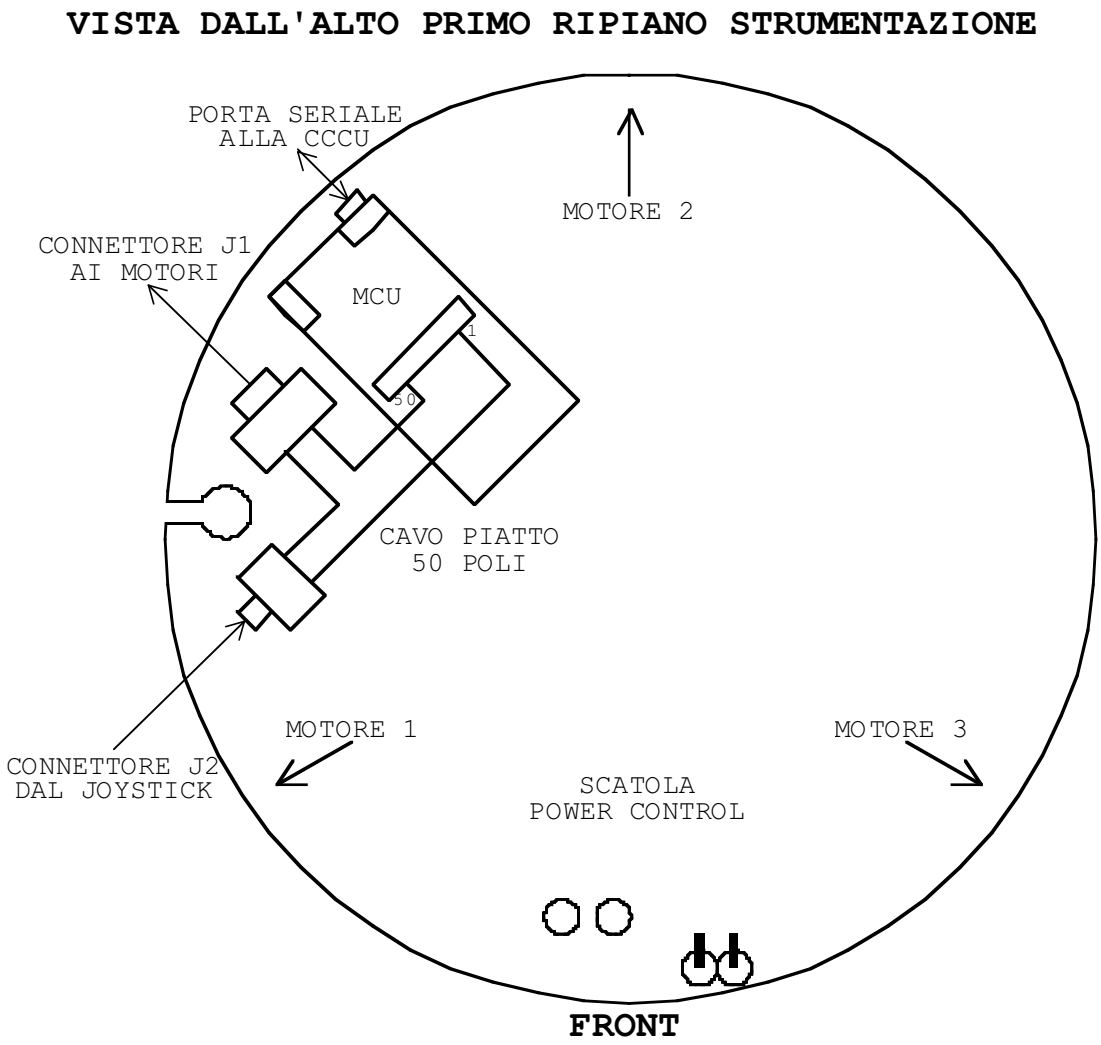

Figura 12: disposizione dei connettori

10. Programma per il MARMOT

> "È vero che il software non potrebbe esercitare i poteri della sua leggerezza se non mediante la pesantezza dell'hardware; ma è il software che comanda, che agisce sul mondo esterno e sulle macchine, le quali esistono solo in funzione del software, si evolvono in modo da elaborare programmi sempre più complessi."
>
> Italo Calvino
>
> *Lezioni Americane*, Edizioni Einaudi

Del programma da installare nella MCU viene fornito il solo sorgente in linguaggio assembly; sarà cura dell'utilizzatore il suo assemblaggio e la conseguente programmazione della EEPROM del processore. Riteniamo che il programma sia ben strutturato e discretamente commentato, perciò in questa sede esamineremo solo le parti più critiche o meno ovvie. Il programma nella sua attuale versione 1.0 occupa circa 1kB di EEPROM.

10.1. Risorse della MCU utilizzate

Come immaginabile, il programma fa un uso piuttosto esteso e intenso delle risorse messe a disposizione dalla MCU. In particolare, vengono utilizzate diverse porte general purpose per I/O parallelo, già descritte precedentemente nelle sezioni dedicate alla scheda 6811LABE e alle connessioni. Inoltre si sfrutta l'interfaccia di comunicazione seriale asincrona e quattro dei cinque timer che il microcontrollore mette a disposizione; i timer utilizzati sono OC2, OC3, OC4 e OC5.

La gestione del canale seriale è completamente realizzata via interrupt; le routine della seriale che lavorano sotto interrupt e quelle chiamate dal main program operano in modo asincrono. Lo scambio di dati tra loro avviene tramite una coppia di buffer circolari, ciascuno dotato di un gruppo di variabili dedicate alla sua amministrazione.

I timer OC2, OC3 e OC4 vengono impiegati sotto interrupt per il controllo in frequenza dei segnali di clock che giungono alle schede driver; tale tipologia di pilotaggio si traduce fisicamente nella capacità di controllare facilmente in velocità i tre motori a passo, come richiesto nelle specifiche.

Il quarto timer, OC5, viene utilizzato in modalità polling per rilevare situazioni di timeout durante la ricezione di dati dalla CCCU, per identificare condizioni in cui il collegamento con la CCCU può considerarsi caduto siccome non si ricevono comandi da parecchio tempo e, infine, per realizzare le curve di velocità.

10.2. Struttura del programma

> Premature optimization is the root of all evil.
>
> D. Knuth
>
> On the other hand, we cannot ignore efficiency.
>
> Jon Bentley

Da un punto di vista ad alto livello il programma è un semplice ciclo infinito che svolge queste due attività:

1. Controllare, tramite una chiamata alla routine **chkpkt,** se sono stati ricevuti dati dal canale seriale. In caso positivo questi verranno accumulati in memoria fino a completare un pacchetto, particolare struttura dati descritta meglio in seguito. Alla conclusione di un pacchetto, questo sarà esaminato per verificarne la validità. Se il test viene superato le richieste che contiene saranno esaudite, con eventuale ritrasmissione alla CCCU di un pacchetto di notifica.
2. Verificare, tramite una chiamata alla routine **checkoc5**, se il quarto timer ha esaurito il tempo impostato. In caso affermativo saranno trascorsi almeno **5ms** dal precedente caricamento del timer ed è necessario eseguire questi tre compiti:
   2.1. Decrementare il contatore utilizzato per rilevare eventuali timeout durante la ricezione di pacchetti dalla seriale. In caso il contatore raggiunga lo zero, la routine **chkpkt** si incaricherà di gestire l'eventuale situazione di timeout invalidando il pacchetto e informando la CCCU della circostanza.
   2.2. Aggiornare, tramite tre chiamate consecutive alla routine parametrica **aggiornarampa**, la velocità attuale di rotazione di ciascuno dei tre motori. A questo proposito ricordiamo che in generale il passaggio dalle velocità attuali a quelle desiderate *non può essere istantaneo*, pena il pericolo di perdita di passi da parte dei motori. Per scongiurare tale situazione questa routine è stata codificata in modo da creare un profilo di velocità adatto a raggiungere gradualmente la velocità finale desiderata. Le frequenze dei segnali di clock vengono quindi aggiornate ogni 5ms. La routine **checkperiodi** si incarica di rendere effettive le modifiche, trasmettendole ai vari dispositivi di I/O coinvolti e alterando i registri dei timer.
   2.3. Controllare lo stato della connessione con la CCCU, dichiarandola persa se non si ricevono dati da troppo tempo. Questo intervallo è stato fissato da noi in **5s** attraverso la costante **DEADTIME**, definita al principio del programma. Per ragioni di sicurezza i motori vengono fermati se questa situazione si verifica.

Organizzando il programma in questo modo siamo riusciti a soddisfare le specifiche del problema senza avere necessità di introdurre alcun ciclo di ritardo. Le routine sono state scritte per la massima velocità operativa e non per minimizzare l'occupazione di memoria.

10.3. Sottosistema di comunicazione

Don't interrupt me while I'm interrupting.

Winston S. Churchill

Come già accennato, la comunicazione tra la MCU e la CCCU che lo comanda si realizza tramite un collegamento seriale asincrono convenzionale; il dispositivo che fisicamente realizza il trasferimento dal lato del controllore è la *SCI*, Serial Communications Interface, già integrata all'interno del chip.

Dopo una serie di esperimenti abbiamo fissato la velocità del canale a 9600 bps, modificabile solo mediante alterazione del programma che gira sulla MCU. Questa particolare velocità coniuga due esigenze:

1. risultare elevata, in modo che l'intervallo intercorrente tra l'invio di un comando alla MCU e l'istante in cui viene effettivamente eseguito sia il più breve possibile;
2. essere abbastanza ridotta da non sovraccaricare eccessivamente la MCU.

A proposito di questo secondo punto, è fondamentale rammentare il peculiare sistema di produzione di segnali ad onda quadra con frequenza arbitraria implementato in questa famiglia di microcontrollori. Sfruttiamo questi segnali per il pilotaggio degli ingressi CK2 delle tre schede driver dei motori a passo che controllano M1, M2 e M3. Poiché pilotiamo tre motori indipendenti necessitiamo di altrettanti timer hardware; a questo scopo abbiamo selezionato OC2, OC3 e OC4.

Questi timer sono detti *output compare* e hanno la singolare caratteristica di dover essere ricaricati via software allo scadere dell'intervallo impostato; la condizione di tempo scaduto, detta *successful output compare*, è segnalata tramite l'invocazione dell'interrupt relativo al timer che ha prodotto l'evento. Ciascun gestore di interrupt dovrà aggiornare il contatore relativo al proprio timer.

Fisicamente, dal punto di vista dell'utilizzatore dei segnali generati, la situazione di tempo scaduto coincide con una transizione del segnale prodotto, cioè un suo fronte di salita o di discesa, secondo il valore assunto dal segnale negli istanti precedenti l'evento. Dunque *il periodo del segnale prodotto da ciascun oscillatore è il doppio dell'intervallo impostato nel rispettivo timer*.

A causa di questa gestione mista dei timer, contemporaneamente hardware e software via interrupt, *è assolutamente necessario che tutti gli interrupt siano gestiti nel minor tempo possibile*, pena la non costanza delle frequenze generate, che nei casi più gravi potrebbe degenerare in perdita di interrupt e quindi mancate transizioni sui segnali prodotti oppure smarrimento di caratteri sull'interfaccia seriale.

Proprio con l'obiettivo di eliminare l'incombenza di questo fenomeno, abbiamo organizzato il programma che gira sulla MCU in modo che le routine di servizio agli interrupt abbiano i tempi d'esecuzione più brevi possibili. A tal fine abbiamo spostato al loro esterno tutto il codice possibile, ma non indispensabile al servizio dell'interrupt. Lo stato attuale del programma dovrebbe minimizzare il tempo speso nella gestione degli interrupt.

Il dover ricaricare periodicamente i tre contatori si traduce anche in un overhead del software, che si manifesta nella diminuzione del tempo di calcolo allocabile ad altri compiti.

10.3.1. Buffer di comunicazione

> The reason that data structures and algorithms can work together seamlessy is that they do not know anything about each other.
>
> Alex Stepanov

Nell'intento di generalizzare il più possibile il sottosistema di comunicazione seriale, senza perdere in efficienza, abbiamo utilizzato due buffer software, implementati nella parte di codice che gestisce a basso livello il trasferimento tramite accesso diretto ai registri della SCI.

Entrambi i buffer possiedono struttura circolare, uno è per la ricezione e l'altro è riservato alla trasmissione; hanno dimensioni fisse di **32** byte ciascuno, definite dalle costanti **rxsize** e **txsize** all'inizio del programma. Il loro stato è memorizzato nelle variabili **rxin**, **rxout**, **rxlen**, **txin**, **txout** e **txlen**; alla loro gestione provvede un gruppo di routine, alcune riservate al sistema, altre disponibili all'utente.

10.3.2. Accesso al canale

L'inizializzazione dell'interfaccia seriale si realizza chiamando **initsci**, mentre la routine **isr_sci** è il gestore delle sue interruzioni. Le routine **sciget**, **scipeek**, **sciput**, **sciputlen** e **sciputchksum** sono liberamente chiamabili dall'utente e operano in modo asincrono rispetto alla **isr_sci**. Ampia documentazione è fornita nel programma.

10.3.3. Protocollo di comunicazione

Il bit rate è stato dunque fissato a **9600 bps**, **1 stop bit**, **no parity**. Non esiste alcun controllo di flusso, né software né hardware. I livelli dei segnali scambiati tra CCCU e MCU, che a questo punto sono solo gli indispensabili TXD e RXD, rientrano nei limiti imposti dallo standard. Se la comunicazione dal punto di vista fisico avviene scambiando bit organizzati in caratteri, dal punto di vista logico abbiamo preferito trasferire blocchi strutturati di byte, che denomineremo pacchetti.

10.4. Sistema a pacchetti

I comandi che dalla CCCU giungono alla MCU, e che da tale dispositivo tornano indietro, hanno una struttura ben definita chiamata *pacchetto*. Un pacchetto è una sequenza ordinata di byte, composta rispettando un certo schema generale e trasmessa in modo che tra un carattere ed il successivo esista un *breve intervallo*, in modo che la comunicazione non venga considerata interrotta, cioè in timeout, da chi la riceve. I pacchetti sono a lunghezza variabile e ne sono stati definiti molti tipi, per consentire una codifica rapida e compatta di molti comandi e varie segnalazioni; tutti però si basano sulla medesima architettura.

10.4.1. Importanza del tempo di timeout

L'intervallo massimo che può trascorrere tra la ricezione di un carattere ed il successivo, prima che la MCU dichiari il pacchetto in timeout, è stato fissato da noi in **15ms** dalla ricezione dell'ultimo carattere. Considerando che alla velocità di 9600 bps il tempo speso per la trasmissione di un singolo carattere è di circa 1ms, questo intervallo di sicurezza risulta essere ben quindici volte maggiore.

Vedremo più avanti che, in caso di timeout, la MCU trasmetterà alla CCCU un pacchetto di tipo TIMEOUT per segnalare la situazione. Sempre nella medesima situazione, la MCU svuoterà il suo buffer di ricezione dalla porta seriale, perché considerato contenere dati oramai inconsistenti.

10.4.2. Funzionamento del DEADTIME

Esiste un altro tempo di timeout, da non confondere con il precedente, che tiene conto del tempo trascorso dalla ricezione dell'ultimo pacchetto. Se è troppo elevato, la connessione con la CCCU viene dichiarata persa dalla MCU; come risultato di questa situazione i motori vengono saggiamente fermati. Il tempo di intervento è fissato in **5 s** dalla costante **DEADTIME**, definita all'inizio del programma.

NOTA: togliendo alcuni commenti al programma, è possibile fare in modo che al verificarsi di questa situazione la MCU mandi un pacchetto di notifica di tipo DEADCONN alla CCCU (vedere più avanti la descrizione di questo pacchetto).

10.4.3. Struttura dei pacchetti

Nella Figura 13 è illustrata la struttura di un pacchetto; ogni riga rappresenta un byte e ne spiega brevemente il significato. Ad occhio è possibile individuare tre campi predefiniti, sempre presenti ed il cui significato è noto a priori; essi sono *lunghezza*, *identificatore* e *checksum*. Vi è inoltre un blocco di byte di dimensioni e significato variabili, che noi chiameremo genericamente *blocco di dati*. Segue una descrizione più dettagliata dei vari campi.

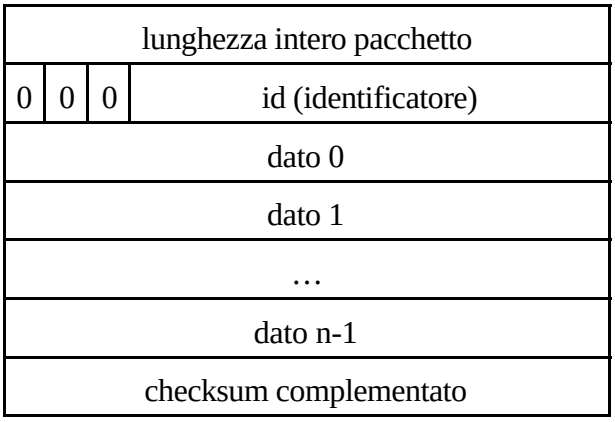

Figura 13: struttura generale di un pacchetto

10.4.4. Lunghezza del pacchetto

Si tratta di un campo di tipo byte contenente un numero, da interpretarsi senza segno, che indica la lunghezza complessiva del pacchetto espressa in byte, compresi i campi predefiniti. Questo valore è utilizzato dalle routine di ricezione, sia sulla MCU che sulla CCCU, per individuare quando un pacchetto è da considerarsi concluso e quindi ricevuto completamente. Se questo numero vale tre significa che il pacchetto non contiene dati, ovvero il blocco di byte è assente.

10.4.5. Checksum complementato

Si tratta di un singolo byte, scelto opportunamente dal trasmettitore affinché un pacchetto ricevuto correttamente abbia la somma di tutti i byte che lo compongono nulla. Molto semplicemente, il trasmettitore calcola questo numero eseguendo il *complemento a due della somma dei byte precedenti*; questo campo va perciò interpretato come intero con segno.

Il controllo che la somma di tutti i byte sia nulla è uno dei metodi che usa il ricevitore per verificare se il pacchetto acquisito è valido o meno; altri metodi controllano se vi è stato timeout oppure se i dati trasmessi hanno valori accettabili.

10.4.6. Blocco di dati

Si tratta di una successione di byte contigui, senza alcun significato particolare predefinito. Sono inseriti nel pacchetto allo scopo di integrare l'informazione recata dall'identificatore; usualmente contengono byte che possono rappresentare i valori di alcune grandezze oppure le configurazioni di certi registri.

10.4.7. Identificatore del pacchetto

E' un byte, da interpretarsi senza segno, che specifica il tipo di pacchetto tra i vari che sono stati definiti. A questo proposito è possibile utilizzare *solo i cinque bit meno significativi*; i tre bit più significativi sono riservati e per ora vengono fissati a zero. In una reimplementazione futura, può darsi che questi bit vengano utilizzati per memorizzare il nome del destinatario del pacchetto, rendendo dunque fattibile la realizzazione di un collegamento multipunto, con più di due dispositivi collegati contemporaneamente sul canale seriale.

La Figura 14 illustra i tipi di pacchetto già definiti, caratterizzati da un nome mnemonico univoco e da un identificatore, quest'ultimo accompagnato dal rispettivo valore.

Nome	Identificatore	Valore
SETSPEED	**ID_SETSPEED**	0
GETSPEED	**ID_GETSPEED**	1
GETADC	**ID_GETADC**	2
BADPKT	**ID_BADPKT**	3
BADCHKSUM	**ID_BADCHKSUM**	4
TIMEOUT	**ID_TIMEOUT**	5
SETMOTORS	**ID_SETMOTORS**	6
GETMOTORS	**ID_GETMOTORS**	7
DEADCONN	**ID_DEADCONN**	8
BADPARAMS	**ID_BADPARAMS**	9

Figura 14: identificatori dei pacchetti già definiti

All'interno dell'insieme di pacchetti è possibile introdurre una suddivisione in *pacchetti di comando* e *pacchetti di segnalazione*. Appartengono alla prima categoria SETSPEED, GETSPEED, GETADC, SETMOTORS e GETMOTORS. Questi pacchetti provengono prevalentemente dalla CCCU e sono diretti alla MCU, che deve risolverli in un cambiamento delle sue uscite o nella trasmissione verso la CCCU di alcuni valori, presi da uno o più dei suoi registri. Durante la trasmissione verso la CCCU, la MCU può utilizzare gli stessi tipi di pacchetto per contenere dati, oppure semplicemente per indicare alla CCCU che il comando relativo è stato ricevuto correttamente ed eseguito con successo.

I rimanenti pacchetti, BADPKT, BADCHKSUM, TIMEOUT, DEADCONN e BADPARAMS, sono di pura segnalazione: vengono inviati soltanto dalla CCCU alla MCU per notificarle particolari situazioni.

Segue ora una descrizione dettagliata di ciascun tipo di pacchetto.

10.5. Pacchetto SETSPEED

Trasmesso dalla CCCU alla MCU quando si intende cambiare la velocità e il senso di rotazione dei tre motori. E' composto da dieci byte, il cui significato è illustrato nella Figura 15. Le sigle M1, M2 e M3 si riferiscono al primo, al secondo e al terzo motore, rispettivamente.

<table>
<tr><td colspan="8">10</td></tr>
<tr><td>0</td><td>0</td><td>0</td><td colspan="5">ID_SETSPEED</td></tr>
<tr><td colspan="8">MSB periodo M1</td></tr>
<tr><td colspan="8">LSB periodo M1</td></tr>
<tr><td colspan="8">MSB periodo M2</td></tr>
<tr><td colspan="8">LSB periodo M2</td></tr>
<tr><td colspan="8">MSB periodo M3</td></tr>
<tr><td colspan="8">LSB periodo M3</td></tr>
<tr><td colspan="5">0</td><td>dirM3</td><td>dirM2</td><td>dirM1</td></tr>
<tr><td colspan="8">checksum complementato</td></tr>
</table>

Figura 15: struttura del pacchetto SETSPEED

E' fondamentale sottolineare che, per motivi legati alla limitata capacità di calcolo dalla MCU, v*engono trasmessi i periodi dei segnali* che pilotano i tre motori, piuttosto che le rispettive velocità. I periodi rappresentano multipli interi di 1µs, valore corrispondente alla minima risoluzione possibile per i tre timer, alla frequenza di clock di 8MHz. Esempio di settaggio: per generare un segnale a 2.5KHz è necessario richiedere un periodo di 1/2.5KHz = 400µs perciò, tenendo presente la risoluzione di 1us, il valore da impostare risulta essere 400µs / 1µs = 400.

La MCU esegue un controllo di sicurezza sui periodi ricevuti: essi vengono *silenziosamente forzati* ad assumere valori nell'intervallo [300, 16001], i cui estremi corrispondono alla frequenza di 3.33KHz ed alla situazione di motore fermo, rispettivamente. Un valore di 16000 produce una frequenza di 62.5Hz, che quindi risulta essere la minima possibile.

Per quanto riguarda i tre bit dirM1, dirM2, dirM3: essi hanno lo stesso significato dei segnali CW/CCW# che pilotano i rispettivi motori. Precisamente, devono valere 1 se si richiede una rotazione oraria; valgono 0 se si esige una rotazione antioraria. I versi si riconoscono guardando dall'interno del robot verso l'esterno. I cinque bit più significativi del campo che li contiene sono riservati e devono valere zero.

In caso di successo, la MCU risponde alla CCCU mandandole un pacchetto di tipo SETSPEED, *ma senza dati*, come mostrato nella Figura 16.

3			
0	0	0	**ID_SETSPEED**
checksum complementato			

Figura 16: risposta al pacchetto SETSPEED

NOTA: questa risposta è facilmente sopprimibile commentando opportunamente alcune righe del sorgente del programma che gira sulla MCU. In questo caso una mancata risposta significa che tutto procede bene.

In caso di problemi, dovuti al cattivo checksum oppure a situazione di timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrati nella Figura 17 e nella Figura 18.

3			
0	0	0	**ID_BADCHKSUM**
checksum complementato			

Figura 17: risposta in caso di cattivo checksum

3			
0	0	0	**ID_TIMEOUT**
checksum complementato			

Figura 18: risposta in caso di timeout

10.6. Pacchetto GETSPEED

Viene trasmesso dalla CCCU alla MCU quando si vuole conoscere i *periodi correnti* dei timer relativi ai tre motori. E' composto da 3 byte, il cui significato è illustrato nella Figura 19.

3			
0	0	0	**ID_GETSPEED**
checksum complementato			

Figura 19: struttura del pacchetto GETSPEED

In caso di successo, la MCU risponde mandando un pacchetto dello stesso tipo GETSPEED, *ma con i dati* illustrati nella Figura 20. Il pacchetto di risposta è composto da 10 byte; le sigle M1, M2 e M3 si riferiscono al primo, al secondo ed al terzo motore, rispettivamente. Il significato del blocco di dati è il medesimo di quello visto per il pacchetto di tipo SETSPEED.

10			
0	0	0	**ID_GETSPEED**
MSB periodo M1			
LSB periodo M1			
MSB periodo M2			
LSB periodo M2			
MSB periodo M3			
LSB periodo M3			
0		dirM3	dirM2	dirM1
checksum complementato			

Figura 20: risposta al pacchetto GETSPEED

In caso di problemi, dovuti a cattivo checksum oppure a situazione di timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrati nella descrizione del tipo di pacchetto precedente.

10.7. Pacchetto GETADC

Trasmettendo questo pacchetto, la CCCU richiede alla MCU il campionamento delle tensioni presenti sui quattro ingressi analogici che la MCU possiede. La struttura del pacchetto di domanda è mostrata in Figura 21.

3			
0	0	0	**ID_GETADC**
checksum complementato			

Figura 21: struttura del pacchetto GETADC

In caso di successo, la MCU risponde mandando un pacchetto dello stesso tipo GETADC, *ma con i dati* illustrati nella Figura 22. I campi ADC1, ADC2, ADC3 e ADC4 contengono i risultati della conversione A/D ad 8 bit delle quattro tensioni agli ingressi.

7			
0	0	0	**ID_GETADC**
ADC1			
ADC2			
ADC3			
ADC4			
checksum complementato			

Figura 22: risposta al pacchetto GETADC

Se il joystick analogico è collegato, i campi ADC1 e ADC2 riportano le letture degli assi orizzontale e verticale, rispettivamente. In caso di problemi, dovuti a cattivo checksum oppure a situazione di timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrati precedentemente nella descrizione del pacchetto di tipo SETSPEED.

10.8. Pacchetto SETMOTORS

Trasmesso dalla CCCU alla MCU per controllare l'alimentazione di ciascuno dei tre motori. E' composto da 4 byte, il cui significato è illustrato nella Figura 23.

4					
0	0	0	**ID_SETMOTORS**		
0			enaM3#	enaM2#	enaM1#
checksum complementato					

Figura 23: struttura del pacchetto SETMOTORS

I tre bit enaM1#, enaM2# ed enaM3# controllano lo stato dei motori M1, M2 e M3, rispettivamente. Essi hanno lo stesso significato dei segnali ENABLE# che pilotano i tre motori: se valgono 0 il motore è abilitato, se valgono 1 il motore è spento. I cinque bit più significativi del campo che li contiene sono riservati e devono valere zero.

In caso di successo, la MCU risponde alla CCCU mandandole un pacchetto di tipo SETMOTORS, *ma senza dati*, come mostrato nella Figura 24.

3			
0	0	0	**ID_SETMOTORS**
checksum complementato			

Figura 24: risposta al pacchetto SETMOTORS

NOTA: la risposta è facilmente sopprimibile commentando opportunamente alcune righe del sorgente del programma che gira sulla MCU. In questo caso una mancata risposta significa che tutto procede bene.

In caso di problemi, dovuti a cattivo checksum oppure a situazione di timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrati precedentemente nella descrizione del pacchetto di tipo SETSPEED.

10.9. Pacchetto GETMOTORS

Viene trasmesso dalla CCCU alla MCU quando si vuole conoscere lo *stato corrente* dell'alimentazione di ciascuno dei tre motori. E' composto da 3 byte, il cui significato è illustrato nella Figura 25.

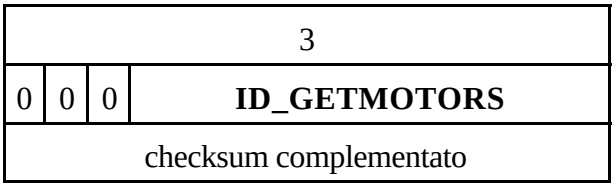

Figura 25: struttura del pacchetto GETMOTORS

In caso di successo, la MCU risponde mandando un pacchetto dello stesso tipo GETMOTORS, *ma con i dati* illustrati nella Figura 26. Il pacchetto di risposta è composto da 4 byte. Il significato del blocco di dati è il medesimo di quello visto per il pacchetto di tipo SETMOTORS.

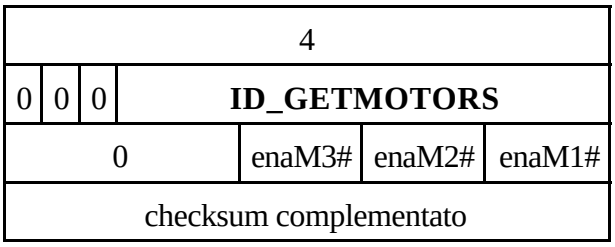

Figura 26: risposta al pacchetto GETMOTORS

In caso di problemi, dovuti a cattivo checksum oppure a situazione di timeout del pacchetto ricevuto, la MCU risponde rispettivamente con un pacchetto di tipo BADCHKSUM oppure TIMEOUT, come illustrati precedentemente nella descrizione del pacchetto di tipo SETSPEED.

10.10. Pacchetto BADPKT

Viene trasmesso dalla MCU verso la CCCU ogni volta che il pacchetto appena ricevuto contiene un identificatore non riconosciuto, ovvero per il quale non è stata definita alcuna routine di gestione. E' composto da 3 byte, il cui significato è illustrato nella Figura 27. La MCU non si aspetta alcuna risposta dalla CCCU.

3			
0	0	0	**ID_BADPKT**
checksum complementato			

Figura 27: struttura del pacchetto BADPKT

10.11. Pacchetto BADCHKSUM

Viene trasmesso dalla MCU verso la CCCU ogni volta che il pacchetto appena ricevuto presenta un errore di checksum, ovvero la somma di tutti i byte di cui è costituito non risulta essere nulla. E' composto da 3 byte, il cui significato è illustrato nella Figura 28. La MCU non si aspetta alcuna risposta dalla CCCU.

3			
0	0	0	**ID_BADCHKSUM**
checksum complementato			

Figura 28: struttura del pacchetto BADCHKSUM

10.12. Pacchetto TIMEOUT

Viene trasmesso dalla MCU verso la CCCU ogni volta che si è presentata una situazione di timeout durante la ricezione di un pacchetto. Il pacchetto incriminato verrà definitivamente scartato dalla MCU, che svuoterà anche il suo buffer di ricezione dalla porta seriale, perché considerato contenere dati inconsistenti. E' composto da 3 byte, il cui significato è illustrato nella Figura 29. La MCU non si aspetta alcuna risposta dalla CCCU.

3			
0	0	0	**ID_TIMEOUT**
checksum complementato			

Figura 29: struttura del pacchetto TIMEOUT

10.13. Pacchetto DEADCONN

Viene trasmesso dalla MCU verso la CCCU *quando è trascorso troppo tempo dalla ricezione dell'ultimo pacchetto*. Il suo scopo è segnalare alla CCCU che la MCU considera caduto il collegamento, poiché apparentemente non vi è più alcuna attività su di esso. Se si verifica questa situazione i motori verranno fermati, rimanendo comunque alimentati alla minima corrente. L'intervallo di tempo per l'intervento di questo meccanismo di sicurezza si definisce attraverso la costante **DEADTIME** all'interno del programma che gira sulla MCU; noi lo abbiamo fissato in **5s**. Il pacchetto è composto da 3 byte, il cui significato è illustrato nella Figura 30. La MCU non si aspetta alcuna risposta dalla CCCU.

NOTA: il meccanismo funziona ma attualmente la spedizione del pacchetto non avviene; per abilitarla è sufficiente decommentare alcune righe del programma che gira sulla MCU.

3			
0	0	0	**ID_DEADCONN**
checksum complementato			

Figura 30: struttura del pacchetto DEADCONN

10.14. Pacchetto BADPARAMS

NOTA: definito ma non implementato.

Il suo scopo dovrebbe essere quello di segnalare alla CCCU che il pacchetto appena ricevuto dalla MCU *contiene valori non accettabili* nel blocco dati. Ad esempio, un buon uso potrebbe essere quello di risposta ad un pacchetto di tipo SETSPEED, se i periodi richiesti sono fuori dall'intervallo consentito. Il pacchetto è composto da 3 byte, il cui significato è illustrato nella Figura 31. La MCU non si aspetta alcuna risposta dalla CCCU.

3			
0	0	0	**ID_BADPARAMS**
checksum complementato			

Figura 31: struttura del pacchetto BADPARAMS

11. Modalità operative

In questa sezione vengono descritte in maniera esauriente le modalità di installazione del software di controllo scritto per la MCU del MARMOT. Prima di procedere è opportuno riassumere gli strumenti software utilizzati per il suo sviluppo: l'assemblatore ASMHC11 ed il programma monitor-debugger PCBUG11**,** entrambi reperibili sul sito Motorola assieme alla documentazione che li riguarda.

11.1. L'assemblatore ASMHC11

Partendo dal file sorgente ASCII (estensione .ASC), redatto tramite un qualsiasi editor di testo, genera un *file listing* (estensione .LST) ed un *file oggetto* assoluto in formato S-record (estensione .S19). Il file listing viene utilizzato per la ricerca e l'identificazione degli errori di assemblaggio e contiene gli indirizzi di allocazione di codice e dati nonché il codice macchina delle istruzioni assemblate. Il file oggetto contiene il codice macchina assemblato; il formato S-record utilizzato è un file ASCII che viene riconosciuto dal PCBUG11 e dai programmatori di circuiti integrati programmabili, fra quali i programmatori di EPROM. Questo formato possiede una struttura a record: è composto da un record d'intestazione (S0), una successione di record di istruzioni (S1) e un record di conclusione (S9).

Per assemblare un programma è sufficiente digitare **asmhc11 nomefile.asc** dalla linea di comando; al termine del processo verrà visualizzato a video il numero di errori individuati nel programma sorgente. Per comodità del lettore, nella Tabella 6 è riportata una breve descrizione delle principali direttive usate nel programma per il MARMOT; si rimanda ai manuali Motorola per una lista esaustiva e per approfondire la grammatica del linguaggio assembly del 68HC11.

EQU	assegna un nome simbolico ad una costante
FCB	introduce un byte all'indirizzo di assemblaggio corrente
FCC	introduce una stringa all'indirizzo di assemblaggio corrente
FDB	introduce una word (due byte) all'indirizzo di assemblaggio corrente
RMB	consente di dichiarare le variabili, riservando spazio in memoria
ORG	assegna l'indirizzo a cui allocare codice oppure variabili
END	delimitatore finale del programma, è obbligatorio

Tabella 6: principali direttive dell'assemblatore

11.2. Il monitor-debugger PCBUG11

Questo software permette di caricare programmi da PC nella EEPROM della MCU in modalità special bootstrap. Può anche mandarli in esecuzione ed effettuarne il debugging online, ma solo *a patto che il programma da testare non utilizzi la porta di comunicazione seriale*, in quanto già impegnata dal programma di monitor, che non si faccia uso di alcuni interrupt già sfruttati dal monitor e che non ci sia necessità di molte locazioni di memoria RAM, in quanto i 256 byte disponibili sono quasi totalmente occupati dal programma talker.

Proprio a causa di queste forti limitazioni non è stato possibile utilizzare PCBUG11 per svolgere un regolare e rapido debugging del programma; l'applicazione è stata sfruttata solamente per caricare il programma in EEPROM.

Al reset della MCU in modalità special bootstrap viene automaticamente eseguito un programma, detto BOOTLOADER, residente in ROM sul 68HC11. Il bootloader è la prima parte del programma di monitor ed il suo compito consiste nello spedire un dato sulla SCI e, in base alla risposta ricevuta, passare il controllo al codice memorizzato in EEPROM oppure caricare del codice in RAM ed eseguirlo.

Nella seconda situazione il PCBUG11 trasferirà il programma utente dal disco del sistema di sviluppo alla RAM del 68HC11; in questo caso la RAM risulterà quasi completamente occupata e alcuni vettori di interrupt (SCI, ILLOP, COP, XIRQ e SWI) non potranno essere utilizzati dal programma utente.

11.2.1. Comandi principali

Per agevolare l'utente i principali comandi sono riportati nella Tabella 7; si raccomanda la consultazione del manuale del PCBUG11 per dettagli. Il tasto ESC termina il comando correntemente in esecuzione; in caso di segnalazioni di tipo **Communication fault,** dovute alla mancata acquisizione oppure alla perdita di sincronismo tra MCU e CCCU, si consiglia di resettare la scheda 6811LABE e riattivare la comunicazione con il comando RESTART.

ASM	visualizza l'istruzione assembly contenuta ad un certo indirizzo di memoria; può anche scrivere in memoria il codice relativo ad una istruzione
BF	scrive un dato in un'area di memoria
BL	visualizza i breakpoint impostati
BR	imposta i breakpoint
CLS	cancella lo schermo e riporta alla configurazione di default
DASM	disassembla
DOS	consente l'uso di comandi DOS
EEPROM	definisce o visualizza l'area di applicazione dell'algoritmo di accesso alla EEPROM (scrittura e attesa ritardo)
EEPROM DELAY	imposta il ritardo (in ms) delle scritture all'area di indirizzamento EEPROM (il valore minimo è 120/baud rate)
EEPROM ERASE	cancella la EEPROM (da farsi prima della scrittura) e abilita o disabilita la funzione erase-before-write (EBW)
FIND	trova un dato o un'istruzione simbolica in un range di indirizzi
G	esegue il codice utente
HELP	accede al menu di aiuto
LOADS	carica il programma utente (un file in formato S-record)
MD	visualizza il contenuto di aree di memoria
MM	modifica il contenuto della memoria
MS	modifica il contenuto della memoria
NOBR	rimuove tutti i breakpoint
QUIT	esce dal programma
RD	visualizza lo stato dei registri principali della MCU (ACCA, ACCB, X, Y, CCR, SP e PC)
RESTART	provoca il riavvio di PCBUG11
RM	modifica lo stato dei registri principali (il cambiamento diviene effettivo premendo ENTER)
RS	imposta i valori dei registri principali
S	ferma l'esecuzione del codice utente
T	traccia il codice utente

Tabella 7: principali comandi del debugger

11.2.2. Supporto delle macro

Vale la pena, prima di terminare la descrizione del PCBUG11, annunciare la possibilità di utilizzare macro in esso; le macro permettono di automatizzare sequenze di operazioni, una capacità che si è rilevata molto utile durante lo sviluppo del programma.

Ad ogni avvio del PCBUG11 è infatti necessario comunicare all'applicazione l'intervallo di indirizzi a cui la EEPROM risponde con il comando EEPROM F800 FFFF e, prima di trasferirvi il programma, è anche indispensabile azzerare alcuni bit di protezione della memoria nel registro BPROT, block protect register, mappato all'indirizzo $1035. La macro che è stata utilizzata è riportata di seguito.

```
RESTART
EEPROM 0
EEPROM F800 FFFF
MS 1035 00
LOADS nomefile
```

11.3. Installazione del software di sviluppo

I programmi assemblatore e debugger non necessitano di alcuna vera procedura d'installazione, basta decomprimere i file del pacchetto in una qualsiasi directory. Nella directory devono essere presenti, oltre agli eseguibili, anche gli altri file distribuiti dalla Motorola quali, per esempio, il programma talker per quanto riguarda il monitor PCBUG11 oppure il file delle regole per l'assemblatore.

Per quanto riguarda l'uso: basta lanciare da linea di comando DOS gli eseguibili ASMHC11.EXE oppure PCBUG11.EXE con i relativi parametri. Rimandiamo alla documentazione Motorola per ulteriori informazioni.

11.4. Osservazione sul testing

A causa dell'impossibilità di effettuare un debugging online del programma, per le ragioni viste precedentemente, il testing dello stesso è stato svolto indirettamente, tramite una serie di esperimenti di validazione sulle routine appena scritte che coinvolgevano l'uso di un oscilloscopio.

Utilizzando questo strumento si sono verificate sia la correttezza del colloquio sul canale seriale tra MCU e CCCU che la qualità dei segnali generati dai tre timer, in termini di forma d'onda, stabilità della frequenza e jitter. Nelle prime fasi dello sviluppo sono state utilizzate anche varie linee della PORTC per segnalare al mondo esterno il verificarsi di particolari situazioni, ritenute indicatori importanti del buon funzionamento del programma.

11.5. Accensione del robot

Per accendere il robot, una volta inserito il pacco delle batterie, azionare l'interruttore sul pannello frontale per fornire i 24 V al convertitore switching. A questo punto, prima di alimentare i motori premendo il pulsante ON, è preferibile attivare l'interruttore MAIN seguito dall'interruttore AUX, in modo da far partire il programma di controllo nella MCU che abilita i motori ma li mantiene fermi. Solo a questo punto è consigliabile fornire ai driver i 24V premendo il pulsante nero.

All'accensione della scheda 6811LABE un circuito, formato da due porte NAND e una rete RC, provvede a generare un impulso di reset che permane per circa 100ms dopo l'accensione, in modo da dare tempo alle alimentazioni di stabilizzarsi. Se la scheda è configurata in modalità single chip il programma utente parte immediatamente e quindi i driver vengono abilitati subito, ma i motori rimangono fermi fino alla ricezione di un opportuno pacchetto di comando dalla porta seriale.

11.6. Avvertenze

Nel caso siano necessarie modifiche al programma che gira sulla MCU, ricordarsi di predisporre opportunamente il jumper J5 sulla scheda 6811LABE, in modo da selezionare la modalità special bootstrap o single chip, come necessario. In ogni caso assicurarsi che la scheda sia alimentata, prima di ipotizzare interruzioni sul collegamento con il PC.

12. Conclusioni e sviluppi futuri

Siamo riusciti nell'impresa, che ci eravamo prefissata, di controllare in velocità i motori del MARMOT, soddisfacendo in pieno le specifiche. Durante il lavoro abbiamo immaginato alcune innovazioni, brevemente riportate di seguito, che potrebbero essere introdotte in una ipotetica versione successiva del firmware della MCU. Le proposte dovrebbero essere in ordine crescente di difficoltà realizzativa:

1. utilizzare uno dei canali A/D della MCU rimasti liberi per leggere la tensione della batteria, dopo averla opportunamente attenuata mediante un partitore resistivo in modo da farla rientrare nel range da 0V a 5V;
2. implementare il tipo di pacchetto BADPARAMS, già definito;
3. rendere la MCU un attore attivo rispetto al canale seriale, in modo che possa prendere l'iniziativa mandando alla CCCU pacchetti del tipo "batteria scarica", "dissipatore troppo caldo" oppure "connessione caduta";
4. sfruttare una lookup table per genere in modo più raffinato le curve di velocità dei motori, una volta nota con sufficiente precisione la configurazione finale del robot;
5. introdurre la possibilità di riprogrammare blocchi della EEPROM via canale seriale durante l'esecuzione del programma, comprese eventuali lookup table, come descritto nell'engineering bullettin EB301 riportato nella bibliografia;
6. aggiungere la possibilità di conoscere la posizione assoluta dei tre motori del robot;
7. implementare la capacità di controllare la posizione del robot, oltre che la velocità;
8. realizzare un collegamento multipunto tra tre o più dispositivi seriali, sfruttando il sistema d'indirizzamento abbozzato nella sezione di descrizione del protocollo a pacchetti.

Bibliografia

[1] Prof.ssa Alessandra Flammini, *Dispensa per il corso di Elettronica Dei Sistemi Digitali*, CLUB Brescia

[2] *TA8435 data sheet*, Toshiba

[3] *M68HC11 Reference Manual*, Motorola

[4] *MC68HC811E2 device information B19C mask set*, Motorola

[5] AN997, *CONFIG Register Issues Concerning the M68HC11 Family*, Motorola

[6] AN1010, *MC68HC11 EEPROM Programming from a Personal Computer*, Motorola

[7] AN1060, *M68HC11 Bootstrap Mode*, Motorola

[8] AN1064, *Use of Stack Simplifies M68HC11 Programming*, Motorola

[9] AN1752, *Data Structures for 8-Bit Microcontrollers*, Motorola

[10] EB188, *Enabling the Security Feature on M68HC811E2 Devices with PCbug11 on the M68HC711E9PGMR*, Motorola

[11] EB189, *Programming MC68HC811E2 Devices with PBbug11 and the M68HC711E9PGMR*, Motorola

[12] EB291, *Programming MC68HC811E2 Devices with PBbug11 and the M68HC11EVBU*, Motorola

[13] EB295, *Programming the EEPROM on the MC68HC811E2 with the M68HC11EVM Board*, Motorola

[14] EB301, *Programming EEPROM on the MC68HC811E2 during Program Execution*, Motorola

[15] EB378, *CONFIG Register Programming for EEPROM-Based M68HC11 Microcontrollers*, Motorola

[16] *M68HC11 PCbug11 USER'S MANUAL*, Motorola

Figure

Tabelle

Indice