Alma Mater Studiorum
Università di Bologna

II Facoltà di Ingegneria – Sede di Cesena

Corso di Laurea in Ingegneria Elettronica e delle Telecomunicazioni

---

Progetto di un sistema a microcontrollore per il controllo a distanza di macchine automatiche con interfaccia CANopen®

Elaborato in:

Elettronica dei Sistemi Digitali

Relatore:
Prof. Ing. Aldo Romani

Correlatore:
Ing. Antonio Valli

Presentato da:
Andrea Culiani

III Sessione
Anno Accademico 2011–2012

Indice

Introduzione

Oggetto di questo elaborato è, data una serie di specifiche, la progettazione di un sistema digitale a microcontrollore, in grado di rendere controllabile a distanza il movimento di un macchinario industriale.

Tale sistema, rappresentato in modo semplificato in Figura 1, è composto da un radiocomando (*consolle*), in grado di interfacciarsi con l'operatore del macchinario per mezzo di pulsanti, joystick proporzionali (manipolatori), ed un display LCD, e da una scheda posta a bordo del macchinario stesso (*macchina*), che deve essere capace di interfacciarsi con un'elettronica di controllo a PLC preesistente, rendendo disponibili per quest'ultima i dati ricevuti dal radiocomando, opportunamente elaborati, mediante protocollo CANopen®. Per la scheda *macchina* si richiede inoltre la presenza di ingressi e uscite digitali, e di ingressi analogici, al fine di espandere le funzionalità dell'elettronica di bordo.

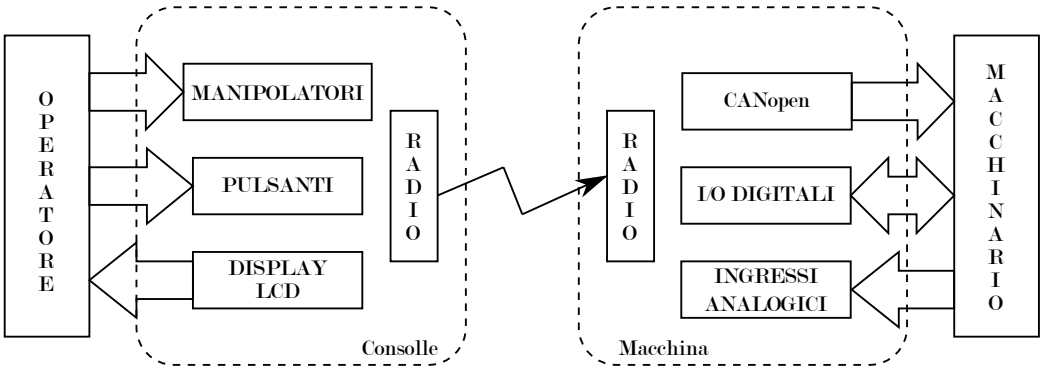


Figura 1: Schema a blocchi del sistema in esame

Per lo scambio di informazioni tra radiocomando e macchinario, il sistema deve instaurare una comunicazione numerica di tipo punto-punto su mezzo

radio, garantendo affidabilità nella comunicazione anche in presenza di disturbi esterni, diagnosticando anomalie nel collegamento, e riconfigurandosi autonomamente in caso di eccessiva degradazione del segnale, limitando al tempo stesso l'occupazione del mezzo radio, ed ottimizzando l'autonomia del radiocomando.

Nella prima parte dell'elaborato, verranno valutati i requisiti funzionali del sistema (Capitolo 1), per poi descrivere, per ciascuna porzione dello stesso, le soluzioni hardware adottate (Capitolo 2), in base ai requisiti di prestazioni richiesti, in rapporto ad aspetti quali economicità e semplicità di collaudo. Viene poi presentata una descrizione dei protocolli di comunicazione utilizzati (Capitolo 3), e la descrizione delle fasi riguardanti la progettazione e la realizzazione dell'hardware (Capitolo 4). Infine, vengono descritte la fase di sviluppo del firmware (Capitolo 5), e le relative funzionalità.

Capitolo 1

Specifiche di progetto

Le specifiche di progetto rappresentano il primo input per il processo di sviluppo di un sistema elettronico. La definizione di tali elementi è dipendente dall'applicazione. Nel caso in esame, si richiede di progettare un sistema costituito da un radiocomando (in seguito *scheda consolle*), e da un secondo circuito (*scheda macchina*) in grado di scambiare dati con il radiocomando su mezzo radio. Il radiocomando rappresenta l'interfaccia con l'operatore del macchinario, e i comandi impartiti a distanza devono essere resi disponibili su interfaccia CANopen®, per permettere l'integrazione con un sistema PLC già presente a bordo del macchinario. L'operatore deve poter controllare il movimento del macchinario, e deve ricevere informazioni di base sullo stato del sistema, mentre gli devono essere nascosti tutti gli altri aspetti, che devono essere gestiti autonomamente dal sistema. Sul radiocomando deve inoltre essere presente un pulsante di sicurezza, che se premuto blocca immediatamente il movimento del macchinario. La comunicazione deve risultare sicura anche in caso di interferenza da parte di altri apparati dello stesso tipo, o di qualunque altro. Ad ogni scheda *macchina* deve essere associato uno ed un solo radiocomando.

1.1 Specifiche generali

- Interventi necessari da parte dell'operatore limitati al minimo
- Possibilità di comunicazione bidirezionale tra radiocomando e macchina
- Tempo di fuori servizio ridotto ad un massimo di 250 ms consecutivi

- Portata radio minima di 50 m

1.2 Specifiche della scheda consolle

- Alimentazione mediante batteria ricaricabile agli ioni di litio da 2 celle (7.4 V nominali, 8.4 V a batteria totalmente carica, e 6.0V a batteria totalmente scarica) e 1800 mAh di capacità
- Autonomia di almeno una intera giornata di lavoro
- Possibilità di collegare fino a 4 manipolatori proporzionali
- Ingressi digitali per pulsanti
- Display LCD alfanumerico, per visualizzazione dello stato della batteria, della qualità del segnale radio, e la ripetizione di segnalazioni dal macchinario al radiocomando

1.3 Specifiche della scheda macchina

- Alimentazione da 10 a 28 V
- Interfaccia CANopen®
- 16 ingressi digitali
- 12 uscite digitali, capaci di pilotare carichi fino a 200 mA, e tensione massima pari a quella di alimentazione
- 4 ingressi analogici $0 \div 10V$
- Contenitore compatibile con guida DIN
- Connettori estraibili per permettere una agevole installazione e rimozione del circuito

Capitolo 2

Selezione dell'hardware

Le specifiche elencate al Capitolo 1 costituiscono la base per la selezione dell'hardware, che sarà oggetto del presente capitolo.

Lo schema a blocchi di una scheda *consolle* è visibile in Figura 2.1, quello di una scheda *macchina* in Figura 2.2. Osservando i due schemi, è possibile notare che la maggior parte dei blocchi sono comuni ai due tipi di hardware. La scheda *consolle*, che non deve interfacciarsi con il macchinario, non è dotata di interfaccia CANopen, mentre la scheda *macchina*, che non deve interfacciarsi direttamente con l'operatore, è priva del display LCD e dei manipolatori.

Date le similitudini nelle funzionalità richieste tra il circuito posto all'interno del radiocomando (scheda *consolle*) e quello a bordo del macchinario (scheda *macchina*), i due dispositivi sono stati realizzati sfruttando lo stesso circuito stampato, in cui alcuni spazi appositamente predisposti, vengono o meno popolati in fase di montaggio, a seconda dell'applicazione. Questa scelta comporta numerosi vantaggi:

- Minore tempo di sviluppo: il progetto dell'hardware ha richiesto lo studio di un solo circuito. Buona parte del software è costituita dai driver delle periferiche, e le similitudini tra l'hardware delle due schede ne permettono il riuso

- Costi di montaggio ridotti: la maggior parte delle fasi di montaggio sono comuni alle due schede, e permettono di essere eseguite su tutte le schede contemporaneamente

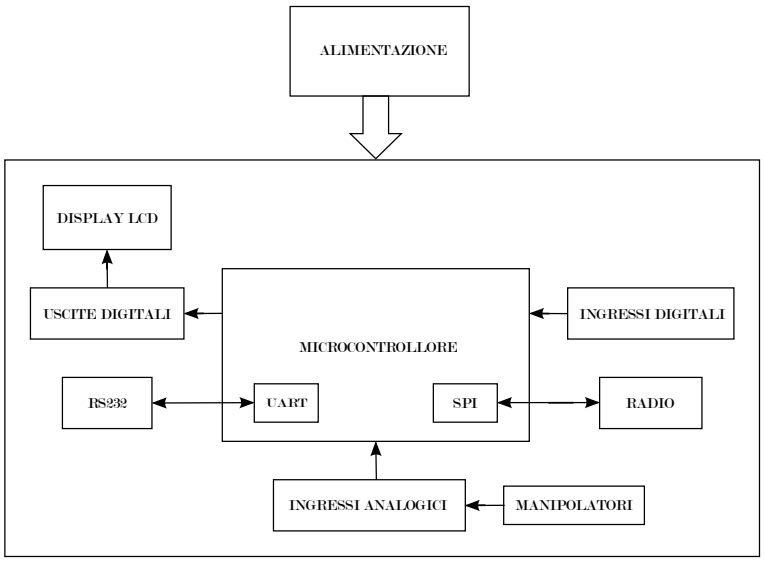


Figura 2.1: Schema a blocchi di una scheda *consolle*

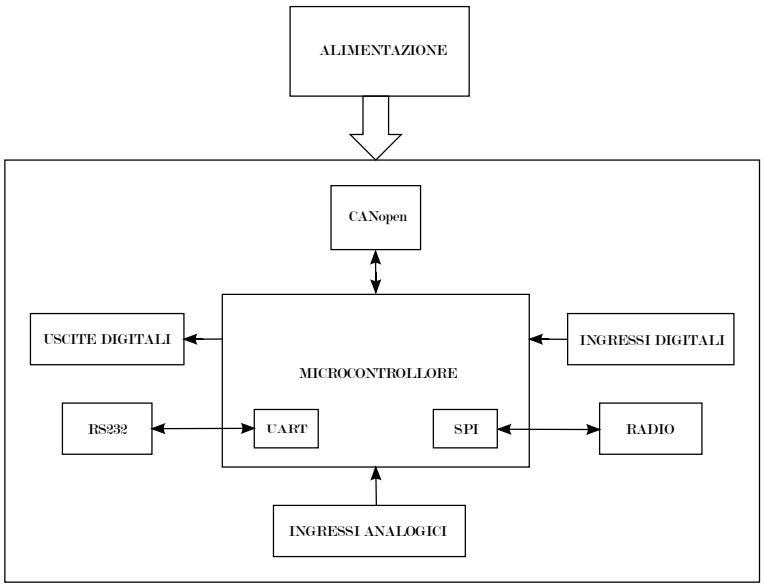


Figura 2.2: Schema a blocchi di una scheda *macchina*

- Semplificazione della fase di collaudo, che richiede solo minime differenze nell'esecuzione tra i due tipi di circuito
- Minore costo dei componenti in fase di approvvigionamento: i componenti comuni alle due schede vengono acquistati in quantità maggiori

Nel seguito verranno descritte le varie parti che compongono il sistema, seguendo la suddivisione utilizzata per la costruzione degli schemi a blocchi.

2.1 Microcontrollore

Il sistema è incentrato sulla presenza di un microcontrollore, che ad una grande flessibilità, garantita dalla possibilità di essere programmato, unisce la presenza di periferiche integrate, con la possibilità di realizzare un sistema completo aggiungendovi un numero relativamente basso di componenti esterni. La selezione del microcontrollore è stata effettuata valutando la presenza di periferiche utili nell'applicazione in esame, in particolare interfacce di comunicazione quali CAN, SPI, UART, un convertitore analogico/digitale dotato di almeno 4 ingressi, almeno un timer disponibile per le temporizzazioni di sistema, una buona quantità di memoria RAM e di memoria Flash per il codice, data la complessità dell'applicazione. Il microcontrollore scelto è il $PIC18F4680$, prodotto da Microchip, che presenta le seguenti caratteristiche principali[1]:

- architettura RISC[1] ad 8 bit, modello di memoria Harvard, e bus istruzioni a 16 bit
- 64 kB di memoria Flash
- 3328 byte di SRAM
- Frequenza di clock fino a 40 MHz (10 MIPS[2]) con quarzo esterno
- 36 linee di I/O nel package scelto
- Modulo MSSP (Master Synchronous Serial Port), con supporto SPI

[1]Reduced Instruction Set Computer
[2]Million Instruction Per Second, milioni di istruzioni eseguite in un secondo

- Modulo EUSART (Enhanced Universal Synchronous Asynchronous Receiver Transmittercon), con supporto UART
- Modulo CAN, con bit rate programmabile fino ad 1 Mbps, supporto per identificatori a 11 e 29 bit, buffer di ricezione e trasmissione dedicati, filtri e maschere per l'accettazione dei pacchetti, gestione degli stati di errore
- Un timer ad 8 bit, 3 a 16 bit
- Convertitore analogico/digitale ad approssimazioni successive con risoluzione di 10 bit e fino ad 11 canali
- Possibilità di uscite PWM (non utilizzate in questo progetto)
- Comparatori analogici (non utilizzati per questo progetto)
- Linee di uscita capaci di pilotare fino a 25 mA ciascuna
- Moltiplicatore hardware 8x8 bit

Il package scelto è il TQFP a 44 pin, per ottenere dimensioni ridotte, mantenendo la possibilità di un montaggio manuale per la realizzazione dei prototipi.

2.2 Ingressi e uscite digitali

Dato il numero elevato di ingressi ed uscite digitali che si richiede di gestire, al fine di ottimizzare l'occupazione delle linee di I/O del microcontrollore, le linee digitali sono state suddivise in gruppi, denominati *blocchi*, formati da 8 ingressi o uscite ciascuno. I blocchi condividono un bus ad 8 bit tra di loro e con il microcontrollore, il quale ha la possibilità di abilitare i singoli blocchi per leggerne (se di ingresso) o modificarne (se di uscita) lo stato. La condivisione del bus tra tutti i blocchi, obbliga all'accesso ad un solo blocco per volta per evitare situazioni di conflitto.

Questo tipo di architettura, a fronte di una notevole riduzione delle linee di I/O del microcontrollore impegnate, richiede operazioni aggiuntive da parte dello stesso per l'abilitazione dei blocchi, con un aumento del tempo di esecuzione di una operazione completa di lettura/scrittura rispetto ad un semplice accesso ad un registro. Il maggiore impegno computazionale è stato comunque ritenuto accettabile in rapporto ai benefici ottenuti.

2.2.1 Blocchi di ingresso

I blocchi di ingresso, che quando interrogati forzano un livello logico sulle linee del bus, devono essere in grado di configurarsi come linee ad alta impedenza quando non in uso, permettendo agli altri blocchi di operare senza creare situazioni di conflitto. Il componente selezionato per questo compito è il transceiver $74HC245$[2], il quale permette, pilotando a livello alto la linea *Output Enable* ($\overline{OE}$), di porre le uscite ad un livello di alta impedenza, come riportato in Tabella 2.1. La linea DIR, che comanda il verso di funzionamento del dispositivo, è posta a livello costante alto, in cui le linee A_n rappresentano gli ingressi, il cui stato viene copiato, se il dispositivo è abilitato, sulle linee B_n.

INPUT		**INPUT/OUTPUT**	
$\overline{\mathbf{OE}}$	**DIR**	$\mathbf{A_n}$	$\mathbf{B_n}$
L	L	$A_n = B_n$	*input*
L	H	*input*	$B_n = A_n$
H	X	Z	Z

Tabella 2.1: Tabella di verità 74HC245

Figura 2.3 riporta il metodo di pilotaggio di un blocco di ingressi digitali, dove $\overline{OE}$ rappresenta il segnala di abilitazione del blocco, e D_n e Q_n rappresentano rispettivamente uno degli 8 ingressi e una delle 8 uscite. Quando è necessaria la lettura di un blocco di ingressi, il microcontrollore imposta le linee corrispondenti al bus come ingressi, abilita il blocco imponendo un livello basso sulla corrispondente linea di abilitazione $\overline{OE}$, ed attende il tempo di propagazione massimo[2], aumentato di un margine di sicurezza, quindi legge legge lo stato del bus e disabilita nuovamente il blocco, che torna in condizione di alta impedenza.

2.2.2 Blocchi di uscita

Per riuscire a mantenere lo stato imposto anche durante successive operazioni sul bus, i blocchi di uscita necessitano di un elemento di memorizzazione. Per questo motivo, è stato selezionato un componente con logica flip-flop, il $74HC273$, che integra 8 flip-flop di tipo D, con possibilità di reset[3]. Il reset dei $74HC273$ (linea $\overline{MR}$), è connesso al reset di sistema, e riporta quindi

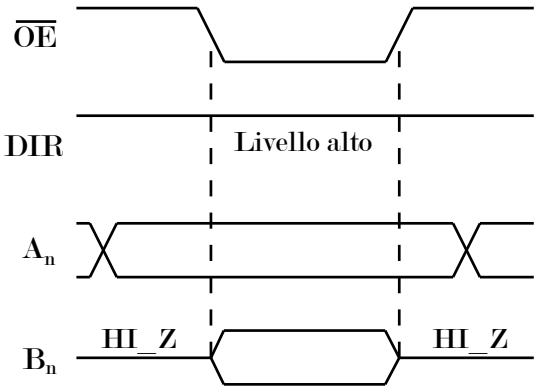


Figura 2.3: Pilotaggio di un blocco di ingressi

tutte le uscite a livello basso in caso di reset globale, come mostrato dalla tabella caratteristica del dispositivo (Tabella 2.2).

Operazione	**INPUT**		**OUTPUT**	
	$\overline{\mathbf{MR}}$	**CP**	$\mathbf{D_n}$	$\mathbf{Q_n}$
reset	*L*	*X*	*X*	*L*
load 1	*H*	↑	*H*	*H*
load 0	*H*	↑	*L*	*L*

Tabella 2.2: Tabella caratteristica 74HC273

Figura 2.4 mostra il metodo di pilotaggio di un blocco di uscite digitali: le linee del bus vengono impostate dal microcontrollore come uscite digitali, e sulla linea CP, normalmente a livello basso, viene generato un impulso a livello alto, il cui fronte di salita innesca la copia sulle uscite dei livelli logici a cui si trovano le linee del bus. Trascorso il tempo di hold massimo del dispositivo[3], l'operazione di scrittura è conclusa, e il bus è disponibile per una nuova operazione.

La necessità di funzionamento delle uscite digitali a tensioni anche superiori a 5 V, e con carichi maggiori di quelli ammessi dai $74HC273$, ha portato all'utilizzo per le uscite digitali, di driver a transistor ULN2003A[4], costituiti da 7 canali a Darlington NPN (Figura 2.5), capaci di pilotare carichi fino a 500 mA ciascuno, e di sopportare tensioni fino a 50 V, superiori quindi alla massima tensione presente a bordo del macchinario, pari a 28 V. Ciascun

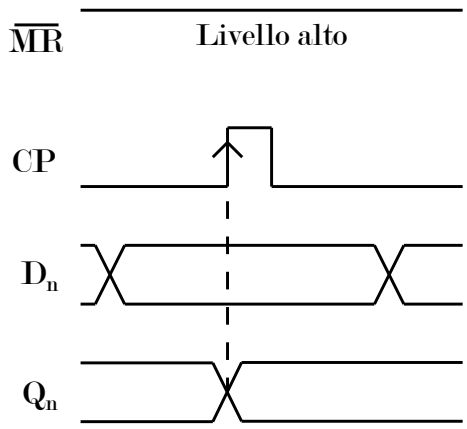


Figura 2.4: Pilotaggio di un blocco di uscite

canale è inoltre dotato di un diodo di ricircolo, permettendo il pilotaggio di carichi induttivi quali relè, valvole, piccoli motori, ecc.

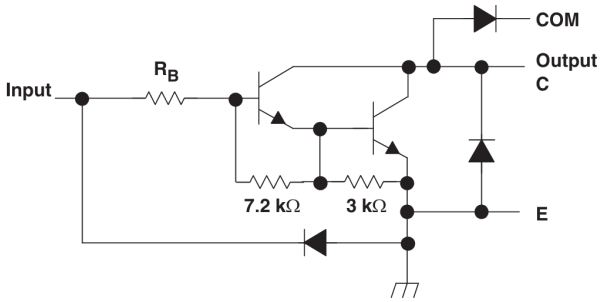


Figura 2.5: Struttura di uno dei canali di un ULN2003A

2.3 Ingressi analogici

Entrambe le schede necessitano di ingressi analogici, che sono però a tensione $0 \div 5\ V$ per la *consolle*, e $0 \div 10\ V$ per la *macchina*. Per permettere la compatibilità con entrambe le tensioni con l'impiego del minimo numero di componenti possibile, è stato utilizzato per ciascun ingresso analogico il sistema visibile in Figura 2.6, dove $SJ1$ connette o esclude il resistore $R22$, che assieme ad $R20$ realizza un partitore resistivo, dimensionato in modo da riportare le tensioni dal range $0 \div 10\ V$ a quello $0 \div 5\ V$ compatibile con

il convertitore analogico/digitale integrato nel microcontrollore. In fase di montaggio, sulle schede *macchina* $SJ1$ viene chiuso in modo permanente. I diodi $D3$ e $D5$ proteggono il circuito da sovratensioni e tensioni negative, mentre il condensatore $C16$, forma assieme al resistore $R20$ un filtro, che limita superiormente la banda del segnale analogico. Un buffer ad amplificatore operazionale, disaccoppia lo stadio di condizionamento e filtraggio dall'ingresso del convertitore analogico/digitale del microcontrollore; l'op-amp dovrà essere del tipo rail-to-rail per permettere di ottenere tensioni di uscita prossime alla tensione di alimentazione. Il componente utilizzato è l'$LMV358$.

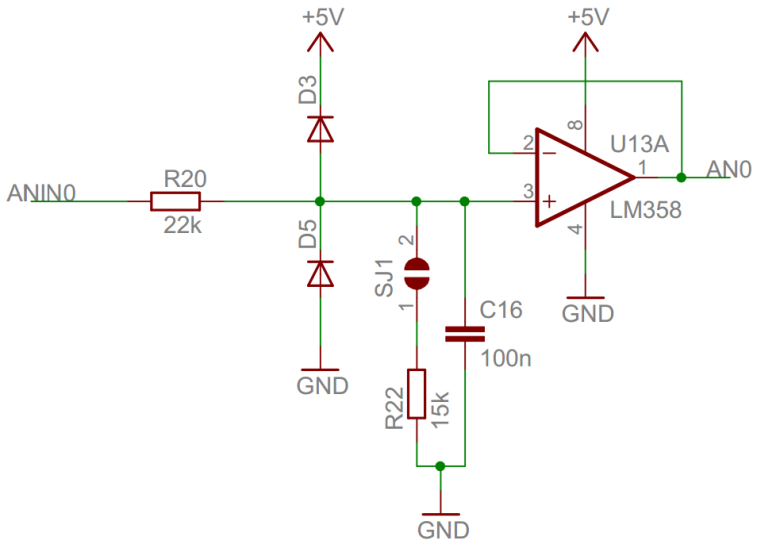


Figura 2.6: Schema elettrico di un ingresso analogico

2.4 Alimentazione

La sezione di alimentazione richiede di poter funzionare in un range di tensioni che va da un minimo di 6 V per la scheda *consolle* a batteria scarica, fino a 28 V a bordo del macchinario. La regolazione di tensione, è realizzata ponendo in cascata un primo stadio (Paragrafo 2.4.1), che dalla tensione di alimentazione principale, produce un'uscita a 5 V, ed un secondo (Paragrafo 2.4.2) che riduce ulteriormente la tensione fino a 3.3 V. L'alimentazione del circuito è protetta dai sovraccarichi da un fusibile ripristinabile.

Un sistema di autoritenuta, visibile in Figura 2.7, permette l'accensione del circuito con un pulsante, portando a massa la linea $PWRON$, e lo spegnimento comandato dal microcontrollore in caso di inattività, che avviene quando la linea PS_KEEP smette di essere pilotata a livello alto. Tale sistema è utilizzato solamente sulla scheda *consolle*, e viene bypassato con la chiusura di $SJ5$ nelle schede *macchina*.

Il partitore formato da $R31$ ed $R34$, anch'esso presente nella sola variante *consolle*, permette al microcontrollore una stima della carica della batteria, utile per la protezione della batteria agli ioni di litio, che può danneggiarsi se scaricata oltre i limiti previsti dal costruttore, e per fornire all'operatore indicazioni sulla carica residua, visualizzate sul display LCD del radiocomando. Il posizionamento del partitore a valle del BJT, permette di evitare che il collegamento resistivo tra il positivo di alimentazione e massa provochi un assorbimento di corrente a circuito spento, pur introducendo un errore nella lettura della tensione, che è stato ritenuto accettabile in relazione alla precisione richiesta.

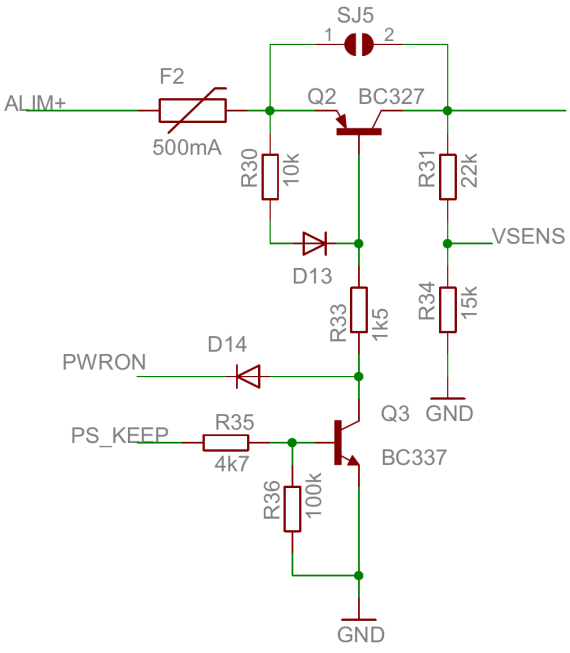


Figura 2.7: Schema del sistema di autoritenuta

2.4.1 Sezione a 5 V

La regolazione a 5 V si effettua a partire della tensione di alimentazione principale; per la scheda *consolle* sarebbe stato possibile utilizzare un regolatore lineare di tipo LDO[3], che nella *macchina*, avrebbe invece comportato una efficienza limitata, e la necessità di un elemento di raffreddamento, incompatibile con lo spazio ridotto imposto dal contenitore. Si è deciso dunque di realizzare, almeno per la scheda *macchina*, la sezione di alimentazione con un regolatore switching di tipo step-down. In seguito, la ricerca del regolatore ha portato alla selezione del componente LT1933, prodotto da Linear Technology, compatibile sia con la tensione minima con alimentazione a batteria, considerando un assorbimento massimo stimato di 100 mA (Figura 2.8), che con la massima tensione a bordo del macchinario (il funzionamento è garantito fino a 36 V[5]). Per il collegamento del regolatore (Appendice A, Figura A.3 sono state seguite le indicazioni del costruttore, riportate sul datasheet del componente[5].

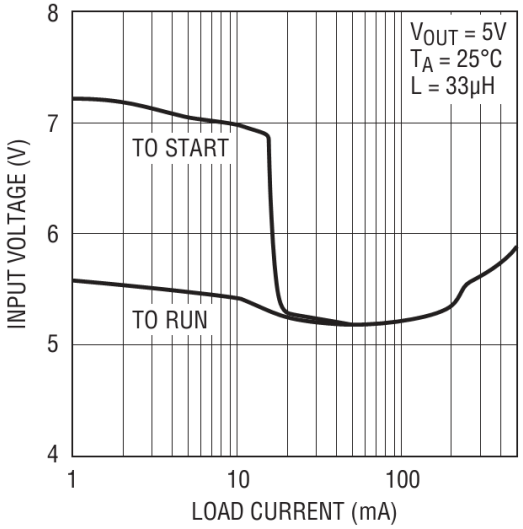


Figura 2.8: Tensione di ingresso minima in funzione della corrente di carico

[3]Low-Dropout

2.4.2 Sezione a 3.3 V

La regolazione a 3.3 V non presenta problematiche particolari, se non la necessità di un regolatore di tipo LDO, dovuta alla ridotta differenza tra la tensione di ingresso e quella di uscita di questo secondo stadio di regolazione. Il componente selezionato è l'$LM1117$[6], in formato SOT-223.

2.5 Interfaccia CAN

Il protocollo CAN è implementato in buona parte in hardware dal controller integrato nel microcontrollore; è richiesta però l'aggiunta di un *transceiver* che permetta il passaggio da una forma di comunicazione a due linee monodirezionali CANTX e CANRX con livelli logici TTL, ad una comunicazione differenziale bidirezionale con livelli compatibili con il bus CAN, oltre ad implementare meccanismi di protezione per il componente stesso ed il resto del circuito da sovratensioni o cortocircuiti sul bus. Il transceiver scelto è l'$MCP2551$[7] prodotto da Microchip che rispetta gli standard ISO, e permette una bit rate massima di 1Mbit/s, oltre ad implementare una protezione che disabilita il pilotaggio del bus se viene determinata l'imposizione da parte del microcontrollore, di un livello dominante per un tempo troppo prolungato, segno di un malfunzionamento che impedirebbe, in assenza di tale protezione, l'accesso al bus a tutti i nodi della rete. Il connettore utilizzato è una morsettiera a 3 poli, che rappresenta una delle possibili scelte previste dalle specifiche CANopen[17].

2.6 Interfaccia RS-232

Entrambe le schede sono dotate di interfaccia EIA RS-232, utile per l'invio di comandi in fase di collaudo, come porta di diagnosi, e per la modifica dell'indirizzo univoco associato a ciascuna coppia consolle/macchina, in caso di necessità di sostituzione di una delle due schede. Il protocollo è gestito in hardware dal modulo EUSART del microcontrollore, che comunica però con livelli elettrici $0 - 5\ V$ compatibili TTL, e necessita di componentistica aggiuntiva per la traslazione nei livelli bipolari previsti dalle specifiche del protocollo. La traslazione è affidata al circuito integrato MAX232, collegato come in Figura 2.9.

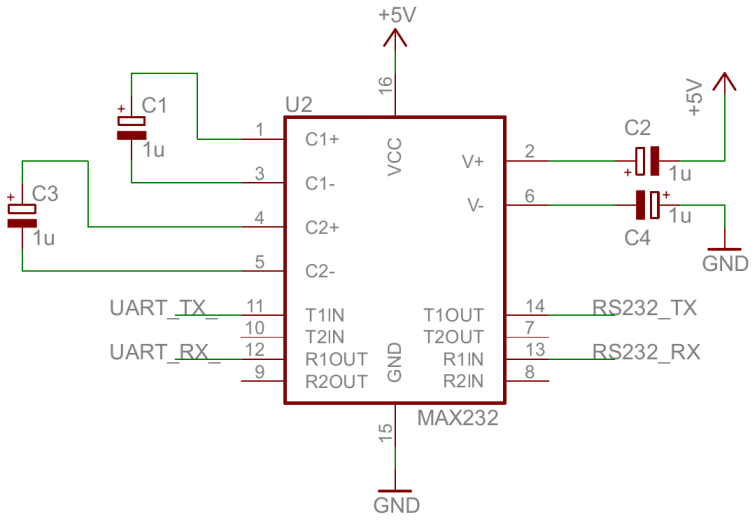


Figura 2.9: Schema di collegamento del circuito integrato MAX232

2.7 Modulo radio

Per la scelta del modulo radio, si è dapprima scelta la banda di frequenze di funzionamento; le normative vigenti[8], impongono limitazioni di potenza e, per alcune bande, limitazioni temporali (duty cycle). La banda scelta è la ISM[4] dei 2.4 GHz, per la quale non sono presenti limiti di duty cycle, e il limite massimo per la potenza ERP[5] è pari a 10 mW. Per riuscire a rispettare la portata radio imposta dalle specifiche, è stato scelto il modulo radio XTR VF 2.4 PA-LNA[9] prodotto da Aurel, che integra il ricetrasmettitore CYRF6936 di Cypress, un'antenna, ed un amplificatore di segnale (Figura 2.10a), in grado di portare la potenza di trasmissione del CYRF6936, che può essere impostata su 8 diversi livelli da -35 ad un massimo di +4 dBm, corrispondenti a circa 2.5 mW, al range $-10 \div +18\ dBm$, in cui il quinto step di potenza corrisponde alla massima potenza ammessa dalla normativa[8], pari a 10 mW (10 dBm).

Il modulo scambia dati e comandi con il microcontrollore per mezzo di una interfaccia SPI (Paragrafo 3.1). L'accesso a registri di configurazione e

[4]Industrial, Scientific and Medical

[5]Effective Radiated Power, rappresenta la potenza che dovrebbe irradiare un'antenna a dipolo per produrre un campo della stessa intensità, nella direzione di massimo guadagno

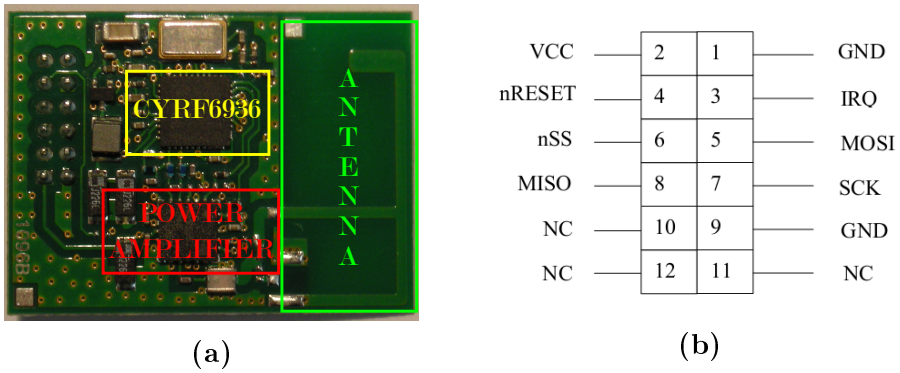


Figura 2.10: Modulo radio XTR VF 2.4 PA-LNA

di stato, e buffer di ricezione e trasmissione si effettua per mezzo di indirizzi, ciascuno dei quali caratterizza un registro od un buffer.

É inoltre presente una linea IRQ, per mezzo della quale il modulo genera interruzioni in corrispondenza di eventi di ricezione, fine trasmissione, ecc.

Il ricetrasmettitore CYRF6936 fa uso della tecnologia Direct Sequence Spread Spectrum (DSSS), che consiste nel codificare ciascun bit di informazione, caratterizzato da una durata pari al tempo di bit T_b, sotto forma di sequenza di simboli binari (nel caso in esame 32 o 64), detti *chip*, come mostrato in Figura 2.11, trasmessi ad intervalli T_c. I chip codificati vengono modulati GFSK (Gaussian Frequency Shift Keying) e trasmessi sul canale radio, con una chip rate fissa pari ad 1 Mbps. La bit rate, dipendente dal tipo di codifica selezionato tra quelli disponibili[10], può variare da un minimo di 16.125 kbps, ad un massimo di 1 Mbps disabilitando il sistema spread spectrum (trasmissione di un bit per chip senza codifica).

I bit vengono codificati sulla base di sequenze, dette Pseudo-Noise (PN) che devono essere note in trasmissione per la costruzione del segnale, ed in ricezione per la corretta decodifica. La decodifica in ricezione, viene effettuata sulla base della correlazione delle sequenze ricevute con la sequenza PN nota; grazie a questo meccanismo, due o più apparati DSSS funzionanti nello stesso istante temporale sul medesimo canale, ma con due codici PN non correlati, o poco correlati tra loro, riescono ad operare contemporaneamente con moderata interferenza tra loro.

Un altro notevole vantaggio di questo sistema, è rappresentato dalla robustezza: se uno o più chip vengono corrotti da interferenza durante il loro trasferimento, è comunque possibile, entro certi limiti, la ricostruzione della

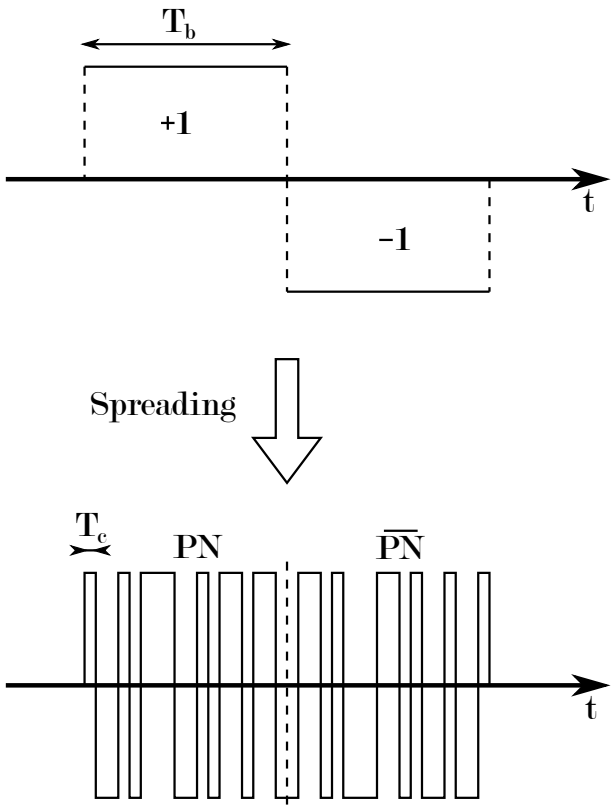


Figura 2.11: Effetti della codifica DSSS nel dominio del tempo

sequenza di bit originale; se la correlazione tra la sequenza di chip ricevuti e la PN supera una soglia stabilita, il bit viene riconosciuto, a prescindere dalla presenza o meno di errori nella sequenza, consentendo il raggiungimento di un tasso di errore per bit molto basso rispetto ad altre tecnologie.

La codifica DSSS, data la validità della relazione $T_c << T_b$, rende la banda del segnale trasmesso, formato dai chip, più ampia rispetto a quella della sola sequenza di bit non codificati (Figura 2.12), aumentando l'immunità da disturbi a banda stretta, e rendendo possibile la coesistenza con sistemi di altro genere operanti sulla stessa banda di frequenze (sitemi WiFi, Bluetooth, ecc.), molto importante su una banda di libero accesso, sulla quale sarà molto probabile incontrare interferenze.

I dati scambiati vengono incapsulati in *pacchetti*, la cui struttura è programmabile in aspetti quali la presenza o meno del campo SoP (Start of Packet), del campo LEN, che indica la lunghezza in byte del successivo cam-

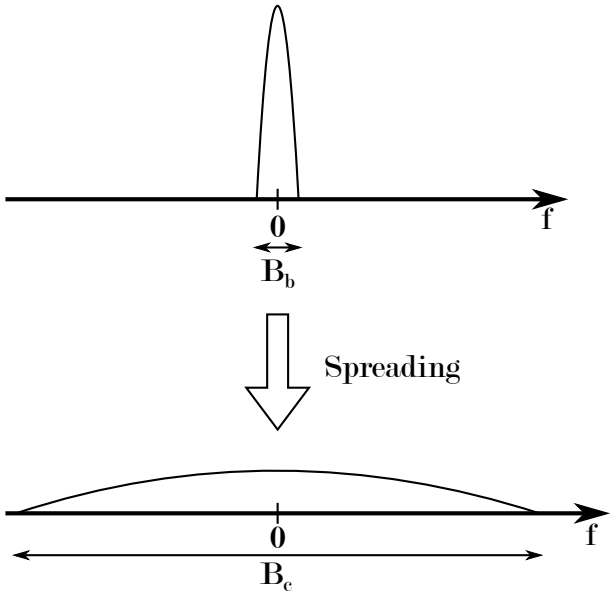


Figura 2.12: Effetti della codifica DSSS nel dominio delle frequenze

po dati, e del campo CRC16 per la rivelazione degli errori. La struttura utilizzata nel caso in esame è riportata in Figura 2.13. Ciascun pacchetto è caratterizzato da un preambolo, una sequenza fissa di 16 chip trasmessa prima di tutti gli altri campi, che ha lo scopo di sincronizzare il tempo di chip tra trasmettitore e ricevitore.

Figura 2.13: Struttura di un pacchetto inviato dal modulo radio

2.8 Display LCD

Il visualizzatore utilizzato è a cristalli liquidi (LCD), di tipo alfanumerico a matrice di punti, pilotato dal chip $HD44780$, divenuto ormai uno standard per moduli di questo tipo. L'interfacciamento avviene mediante un connettore a 16 pin (Tabella 2.3), cui fanno capo le 8 linee dati, le linee di controllo,

e quelle necessarie per l'alimentazione del modulo e della retroilluminazione a LED.

Numero pin	**Denominazione**
1	VSS
2	VDD
3	VEE
4	RS
5	R/W
6	E
7-14	D[0..7]
15	A
16	K

Tabella 2.3: Pinout del modulo LCD

La comunicazione è di tipo parallelo ad 8 bit sulle linee D0-D7; è inoltre disponibile una modalità di trasferimento a 4 bit[11], utilizzata in questo progetto, che permette un risparmio di 4 linee di I/O per il pilotaggio del modulo, a fronte di un interfacciamento lievemente più complesso, in cui il trasferimento di informazioni avviene in due tempi, un *nibble* per volta, per ciascun dato/comando. Un'altra linea di I/O può essere risparmiata evitando l'utilizzo del pin R/W, che se settato permanentemente a livello logico basso, permette l'utilizzo del display in sola scrittura.

Le restanti linee di controllo RS ed E, assumono rispettivamente il ruolo di selezione tra l'invio di dati o comandi, e di avvio delle operazioni di lettura/scrittura. I caratteri da visualizzare sono inviati al modulo in modalità dati (RS=0), e la corrispondenza tra byte inviato e carattere visualizzato è definita da una mappa caratteri predefinita[11], alla quale, per mezzo di appositi comandi[11], è possibile affiancare fino ad 8 caratteri personalizzati. I comandi, inviati in modalità comando (RS=1), forniscono funzionalità utili alla configurazione del modulo, allo spostamento del cursore nella matrice di caratteri, alla pulizia del display, ecc.

Sono inoltre presenti pin aggiuntivi per il controllo del contrasto (VEE), e per l'alimentazione della retroilluminazione a LED del display (A, K).

Al fine di diminuire ulteriormente l'impatto del modulo LCD sull'occupazione delle linee di I/O del microcontrollore, il pilotaggio delle linee di comunicazione, di controllo, e della retroilluminazione (linee LCD_x in Fi-

gura 2.14, 7 in totale), è stato affidato ad un blocco di uscite digitali, descritto al Paragrafo 2.2.2, privo però della sezione di potenza ad ULN2003A.

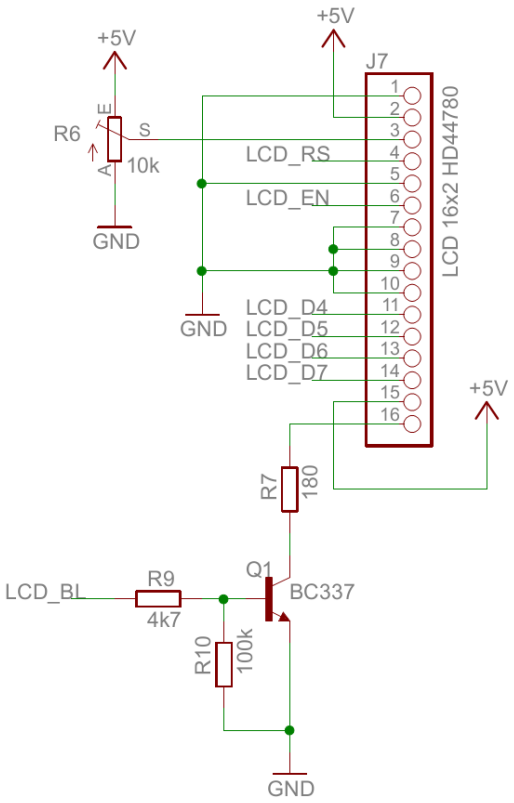


Figura 2.14: Schema di collegamento del modulo LCD

2.9 Manipolatori

I manipolatori costituiscono l'interfaccia tra l'operatore e la scheda *consolle*, e rendono possibile un controllo proporzionale del movimento del macchinario. L'utente è in grado di regolare la velocità variando l'inclinazione dell'elemento mobile (Figura 2.15a) del manipolatore rispetto alla posizione centrale di riposo, in un verso o nell'altro; è quindi disponibile un controllo avanti/indietro per ciascun canale.

La struttura interna di ciascun manipolatore è costituita da un potenziometro e da due switch (Figura 2.15b). Il potenziometro è connesso in

configurazione di partitore di tensione, e presenta un terminale collegato con il centro dell'elemento resistivo (pin 7). Questa configurazione permette di fissare la tensione al centro dell'elemento resistivo, dando la possibilità di ottenere una escursione simmetrica in tensione per entrambi i versi di inclinazione dell'elemento mobile.

L'aggiunta degli switch è utile nel discriminare il verso in cui si sta muovendo il manipolatore, oltre a svolgere l'importante funzione di protezione contro eventuali malfunzionamenti del potenziometro: ciascuno switch si attiva (chiudendo il contatto) solamente quando l'elemento mobile viene inclinato di oltre 4^o rispetto alla posizione di riposo, nella direzione corrispondente. In questo modo, se un potenziometro dovesse presentare malfunzionamenti, ci sarebbe comunque la possibilità di determinare se il manipolatore si trova in posizione centrale o meno, permettendo di interrompere tempestivamente il movimento del macchinario.

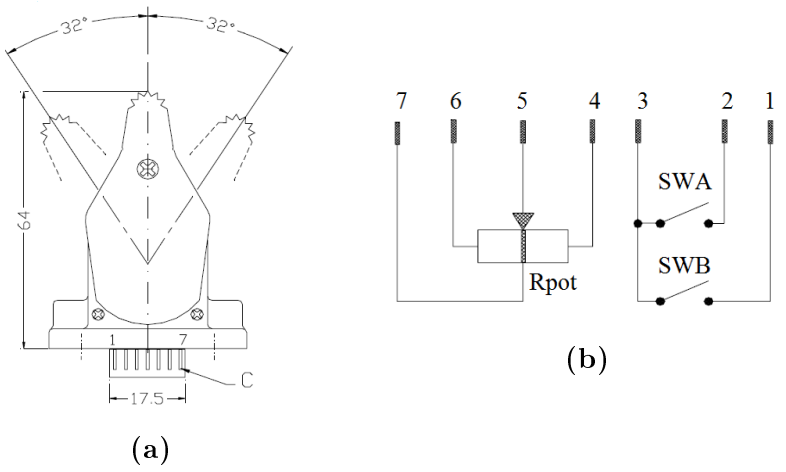


Figura 2.15: Manipolatore Tecnord JLP-L2S

Capitolo 3

Protocolli di comunicazione

3.1 SPI

Il *Serial Peripheral Interface* (*SPI*), è uno protocollo di comunicazione seriale di tipo sincrono (è presente un segnale di clock per la sincronizzazione dei singoli bit) e full-duplex[1], che permette il trasferimento di dati tra un *master* e uno o più *slave*, connessi mediante una linea di clock, due linee dati, ed una di selezione; un esempio di connessione tra un master ed un solo slave, che rappresenta il caso in esame, è rappresentato in Figura 3.1.

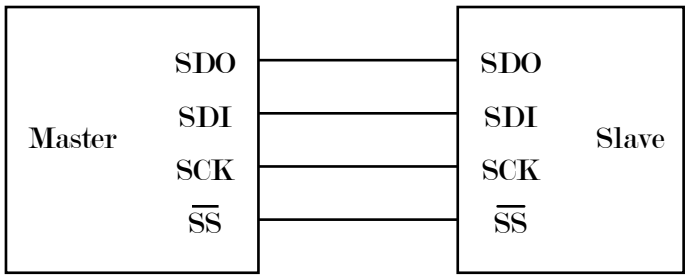


Figura 3.1: Connessione SPI tra master ed un solo slave

In questo progetto l'intefaccia *SPI* è sfruttata per lo scambio tra microcontrollore ed il modulo radio di comandi di configurazione, e di dati ricevuti o da trasmettere. La temporizzazione della comunicazione è affidata ad una linea di clock (SCK) pilotata dal *master*; i dati vengono scritti sulle

[1]Si dice full-duplex una comunicazione che rende possibile lo scambio di informazioni simultaneo in trasmissione e ricezione

due linee dati SDO ed SDI in corrispondenza dei fronti di discesa, mentre il campionamento avviene sui fronti di salita (Figura 3.2).

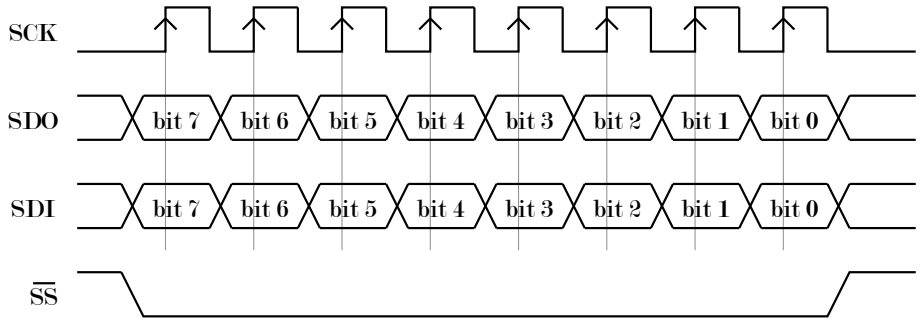


Figura 3.2: Esempio di trasferimento su interfaccia SPI

Sono possibili implementazioni che differiscono da quella descritta per la polarità del segnale (e quindi dei fronti) di clock, o per la dimensione delle unità di informazione trasferite. Il trasferimento di dati su *SPI* avviene sempre contemporaneamente in trasmissione e ricezione, anche se spesso (come nel caso in esame) il protocollo viene sfruttato solamente in una direzione per volta, impostando la linea inutilizzata su un livello qualsiasi, e scartando i dati campionati sulla stessa. I bit vengono trasmessi a partire dal più significativo (*MSB*) fino al meno significativo (*LSB*). Si osserva inoltre l'attivazione (livello basso) di una linea *Slave Select* all'inizio della comunicazione, e la sua disattivazione a trasferimento completato. Tale linea, pilotata dal *master*, permette l'indirizzamento di più *slave* con la condivisione delle linee dati e del clock. Data la scarsa immunità ai disturbi, solitamente il protocollo SPI è utilizzato tra dispositivi collocati sulla stessa scheda, o comunque a distanze molto ridotte. L'assenza di un limite massimo alla velocità di trasmissione, ed il minimo overhead, consentono un elevato throughput rispetto ad altri protocolli.

3.2 CAN

Il protocollo Controller Area Network, noto anche con il nome di CAN bus, nasce negli anni Ottanta, al fine di trasferire ridotte quantità di informazioni tra più unità di controllo in campo automotive, riducendo la quantità di cablaggio necessaria. Attualmente il CAN è molto diffuso anche in ambito industriale, grazie alle caratteristiche di elevata flessibilità, affidabilità ed immunità ai disturbi, ed è standardizzato come ISO 11898. É un protocollo seriale in cui i dispositivi comunicano mediante lo scambio di messaggi broadcast su un bus condiviso, con una bitrate massima di 1 Mbit/s, e su distanze anche superiori al chilometro a bitrate inferiori. L'elevato overhead lo rende inadatto per flussi di dati ad elevato throughput.

Il CAN si posiziona nel modello ISO/OSI[2] ai primi due livelli[12] (Figura 3.3).

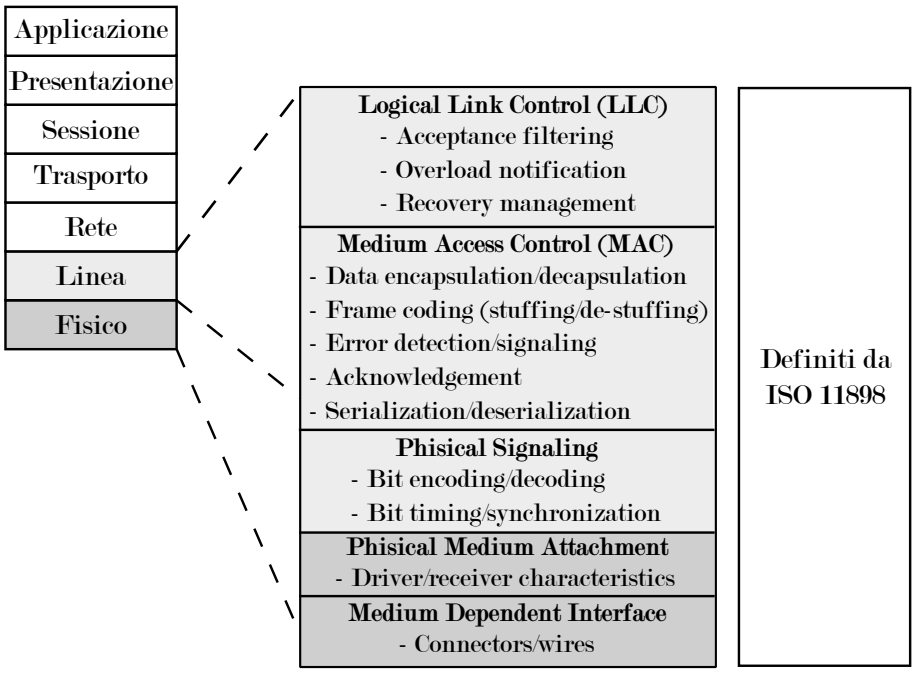


Figura 3.3: CAN nel modello ISO/OSI

[2]ISO Open Systems Interconnection

3.2.1 Struttura di una rete CAN

Il mezzo di trasmissione è costituito da una coppia di cavi (linee CAN_H e CAN_L) paralleli, schermati o meno[13], detta bus, che rappresenta una risorsa condivisa tra tutti i nodi (Figura 3.4), i quali possono quindi accedervi anche contemporaneamente, determinando una condizione di *collisione*: per questo il protocollo è di tipo *CSMA/CD*[3], e il metodo di pilotaggio del bus da parte dei driver permette collisioni non distruttive, e la rilevazione e gestione delle stesse secondo un sistema di priorità (Paragrafo 3.2.3). Il bus deve essere chiuso ad entrambe le estremità su circuiti di terminazione, solitamente costituiti da un resistore del valore nominale di 120Ω, per evitare che le discontinuità inneschino riflessioni del segnale.

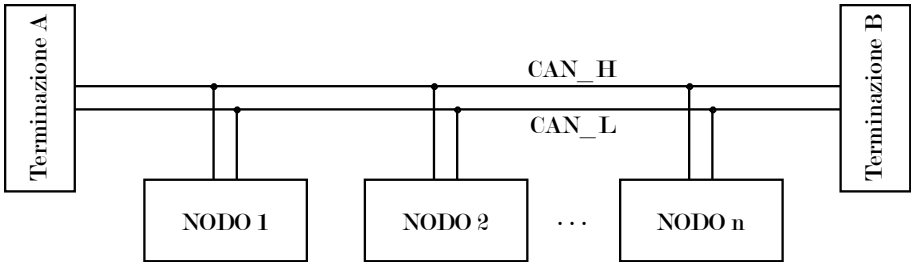


Figura 3.4: Struttura di una rete CAN

3.2.2 Segnali elettrici

I simboli trasmessi sul bus, corrispondenti ai bit, sono di tipo NRZ: viene rilevato un livello recessivo (livello logico 1) se la tensione sulla linea CAN_H non è superiore a quella sulla linea CAN_L più 0.5 V. Se invece la tensione sulla linea CAN_H supera di almeno 0.9 V quella della linea CAN_L, il livello imposto sul bus è dominante (livello logico 0). Tipicamente la tensione a livello recessivo è di 2.5 V su entrambe le linee, e il segnale sulle linee CAN_H e CAN_L segue andamenti simmetrici attorno a tale valore. Il livello logico rilevato dipende comunque non dalle singole tensioni, ma dalla differenza tra le stesse; questo conferisce al CAN Bus ottime caratteristiche di immunità ai disturbi di modo comune. Un esempio dell'andamento nel tempo delle tensioni sulle linee CAN_H e CAN_L è rappresentato in Figura 3.5.

[3] Carrier Sense Multiple Acces with Collision Detection

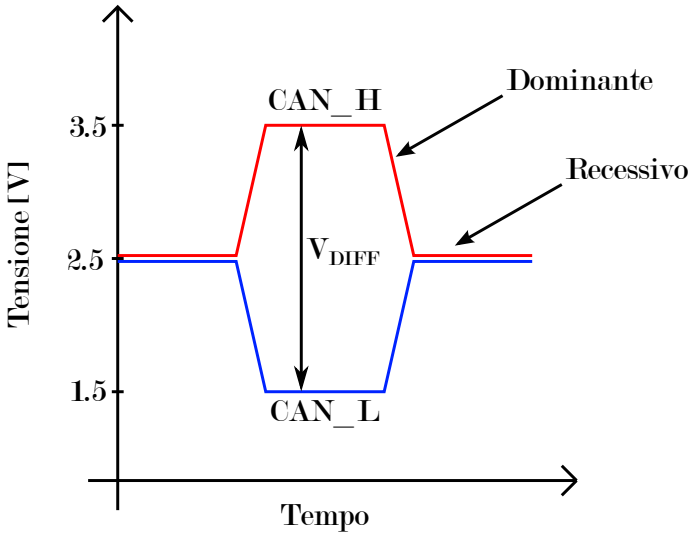


Figura 3.5: Esempio di segnali su bus CAN

3.2.3 Bit dominanti e recessivi

Il CAN utilizza un modello a priorità: i messaggi sono costituiti da una sequenza di bit dominanti e recessivi, in cui i bit dominanti sono quelli a livello logico basso (0), che risultano più prioritari rispetto a quelli a livello alto (1). I driver CAN sono costruiti in modo che il livello effettivamente imposto sul bus, sia l'AND logico tra tutti i valori che i singoli nodi tentano di imporre concorrentemente.

Per rilevare condizioni di collisione, ciascun nodo resta in ascolto durante la trasmissione Se viene rilevata una differenza tra il valore che si sta tentando di imporre e quello ricevuto (imposizione di un bit recessivo e ricezione un bit dominante), il nodo interrompe immediatamente l'operazione e si pone in ricezione, in modo da essere in grado di acquisire il pacchetto correttamente e poterlo eventualmente processare. Per il pacchetto bloccato, viene tentata automaticamente la ritrasmissione appena il bus torna libero. Il nodo che al momento della collisione stava trasmettendo il bit dominante, continua la trasmissione normalmente.

3.2.4 Struttura dei pacchetti

Ciascun pacchetto presenta una struttura fissa, rappresentata in Figura 3.6, in cui, con riferimento al solo protocollo standard con identificatori ad 11 bit utilizzato nel presente progetto:

- **SOF:** Start Of Frame, indica a tutti i nodi della rete che una trasmissione ha avuto inizio, e permette la sincronizzazione del tempo di bit di tutti i nodi. É costituito da un singolo bit dominante.
- **Identificatore:** indica la tipologia di messaggio, e permette ai nodi, mediante opportuni filtri, di riuscire a selezionare i messaggi rilevanti e di scartare quelli che non lo sono. Ciascun messaggio può essere accettato da un numero qualsiasi di nodi, anche nessuno. Per il meccanismo descritto al Paragrafo 3.2.3, il trasferimento di messaggi aventi un identificatore con valore più basso, risulta più prioritario.
- **RTR:** Remote Transmission Request, è utilizzato per richiedere ad un altro nodo la trasmissione di un determinato dato, caratterizzato dal medesimo identificatore della richiesta.
- **IDE:** Identifier Extension, è un bit che indica il tipo di identificatore, ad 11 bit (identificatore standard) se si tratta di uno 0 logico, a 29 bit (identificatore esteso) altrimenti.
- **r0:** bit riservato per espansioni future.
- **DLC:** Campo a 4 bit, indica il numero di byte di dati presenti nel pacchetto. Tutti i valori rappresentabili (da 0 a 15) sono validi, ma valori maggiori di 8 devono essere interpretati dai nodi riceventi come 8.
- **Dati:** Campo di lunghezza variabile da 0 ad un massimo di 8 byte. La dimensione del campo dati, in byte, è indicata dal campo DLC.
- **CRC:** Cyclic Redundancy Check, valore a 15 bit, calcolato sulla base dei valori presenti nei campi precedenti, da SOF all'ultimo bit di dati, seguito da un sedicesimo bit di valore recessivo. Il campo CRC è sfruttato nell'attuazione di uno dei meccanismi di rivelazione degli errori, che in questo caso avviene confrontando il valore CRC inviato dal nodo trasmittente con quello determinato dal nodo ricevente utilizzando i dati acquisiti.

- **ACK:** Il nodo che trasmette il messaggio inserisce un bit recessivo, che viene sovrascritto da uno dominante inserito dai nodi in cui la verifica mediante CRC dei dati ricevuti sia andata a buon fine. Questo primo bit è seguito da un secondo di valore sempre recessivo, detto delimitatore.

- **EOF:** Ogni pacchetto termina con un End Of Frame, costituito da una sequenza di 7 bit recessivi.

Figura 3.6: Struttura di un pacchetto CAN

I campi Identificatore e RTR (12 bit in tutto) costituiscono il campo di arbitraggio, che determina quale tra più messaggi trasmessi concorrentemente gode priorità maggiore. In particolare, essendo il valore 0 dominante, i messaggi caratterizzati da un identificatore di valore più basso, risultano i più prioritari.

3.2.5 Bit stuffing

Il meccanismo *bit stuffing* prevede l'aggiunta, dopo 5 bit aventi lo stesso valore trasmessi in sequenza, di un bit di valore opposto. Questa tecnica, oltre a permettere di mantenere un buon sincronismo del tempo di bit tra i nodi della rete, rappresenta una forma di rivelazione degli errori: nel caso in cui vengano ricevuti 6 o più bit consecutivi dello stesso tipo, è sicuramente presente un errore o una condizione di guasto. I bit aggiunti in fase di codifica vengono rimossi in fase di decodifica dal nodo che riceve il pacchetto.

3.2.6 Spaziatura tra i pacchetti

Tra la trasmissione di due pacchetti è stabilita una spaziatura: al termine del pacchetto il bus deve rimanere nello stato recessivo per 3 tempi di bit, detti di interruzione, a cui si aggiungono, se il nodo trasmittente è nello stato di errore passivo (Paragrafo 3.2.9), 8 tempi di bit di sospensione della trasmissione. Trascorso questo tempo, il bus assume lo stato detto IDLE, in cui ogni nodo può prenderne il controllo tentando di trasmettere un messaggio. La durata

dello stato IDLE è indefinita, e dura finché il bus non viene nuovamente impegnato.

3.2.7 Condizioni di errore

Il CAN prevede 5 diverse condizioni di errore (non mutuamente esclusive)[12]:

- **Errore di bit:** Si verifica quando in trasmissione, il nodo trasmittente rileva sul bus un valore diverso da quello imposto. Non si applica per i campi ACK e per le segnalazioni passive di errore.
- **Errore di stuffing:** Viene generato in caso di violazione della regola del bit stuffing (Paragrafo 3.2.5).
- **Errore CRC:** Si ha un errore CRC se il valore del campo CRC ricevuto non coincide con quello calcolato localmente.
- **Errore ACK:** Se il primo bit del campo ACK resta recessivo, nessun nodo ha ricevuto il pacchetto correttamente.
- **Errore di forma:** Generato quando si rileva un bit dominante in una posizione in cui dovrebbe essere obbligatoriamente presente un bit recessivo (EOF, delimitatore ACK, ecc.).

Quando uno dei nodi rileva uno dei tipi di errore descritti, lo stato di errore viene segnalato agli altri nodi, mediante un una trama costituita da un flag di errore attivo o passivo, a seconda dello stato in cui si trova il nodo (Paragrafo 3.2.9), al quale deve seguire l'invio di un delimitatore, costituito da 8 bit recessivi. La trama di errore segue il pacchetto che ha generato la condizione di errore, come da Figura 3.7. Alla ricezione di un flag di errore, ciascun nodo deve inviare la medesima segnalazione sul bus: questo comporta un prolungamento del flag, che diviene la sovrapposizione delle segnalazioni inviate dai diversi nodi.

3.2.8 Overload frame

In seguito alla trasmissione di un pacchetto, un nodo può richiedere di ritardare la trasmissione del pacchetto successivo mediante una trama di tipo *overload*, rappresentata in Figura 3.8, e costruita allo stesso modo di quella di errore appena descritta. La trama di *overload* e viene trasmessa immediatamente dopo quella di errore, se quest'ultima è presente.

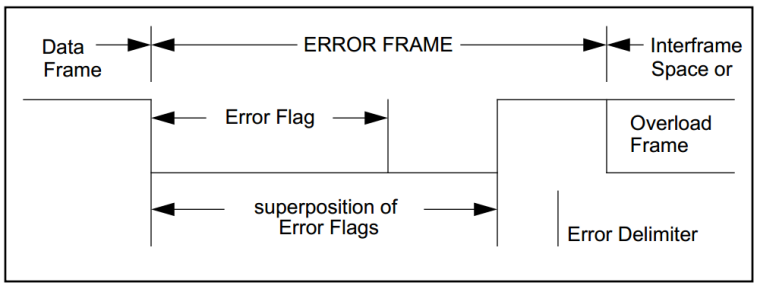


Figura 3.7: Trama di errore CAN

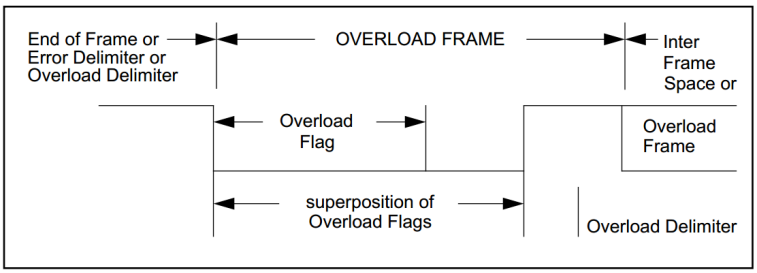


Figura 3.8: Trama di *overload* CAN

3.2.9 Stati di errore

Al fine aumentare la probabilità che il bus resti fruibile anche in caso di problemi ad uno dei nodi, il protocollo CAN prevede un differente comportamento degli stessi in funzione della frequenza con cui vengono rilevati errori sul bus. Tale meccanismo è realizzato mediante due contatori: TEC per gli errori di trasmissione, e REC per gli errori di ricezione. Il valore del contatore TEC è incrementato quando si verificano errori di trasmissione, REC viene invece incrementato quando si verificano errori in ricezione; l'entità dell'incremento è proporzionale alla gravità dell'errore riscontrato. Quando invece un pacchetto viene trasmesso o ricevuto correttamente, il valore del rispettivo contatore viene decrementato di 1.

Sulla base del valore raggiunto dai due contatori, il comportamento di un nodo può assumere uno dei seguenti 3 stati:

- **Errore attivo:** Quando entrambi i contatori hanno valori inferiori a 127. Le condizioni di errore vengono segnalate mediante un flag di errore attivo, costituito da sei bit dominanti consecutivi (viola la regola del bit stuffing). I nodi che rilevano sul bus un flag di errore attivo,

devono (se in stato di errore attivo) a loro volta trasmettere lo stesso flag.

- **Errore passivo:** Un nodo entra in errore passivo quando il valore di uno dei due contatori di errore supera 127. Il nodo può prendere parte normalmente alla comunicazione sul bus, ma non è autorizzato all'invio di flag di errore attivo, che vengono sostituiti dal flag di errore passivo, costituito da 6 bit recessivi consecutivi. Il nodo può tornare allo stato di errore attivo se entrambi i contatori tornano ad assumere valori inferiori a 127.

- **Bus off:** Se il contatore degli errori di trasmissione supera 255, viene riconosciuto il nodo come disconnesso dal bus. Il nodo non può quindi trasmettere né ricevere messaggi, non può inviare flag di segnalazione di errori né ACK. É possibile una transizione allo stato di errore attivo, se il nodo riceve correttamente 128 sequenze di 11 bit recessivi consecutivi.

Un riassunto delle modalità di transizione tra gli stati è mostrato in Figura 3.9.

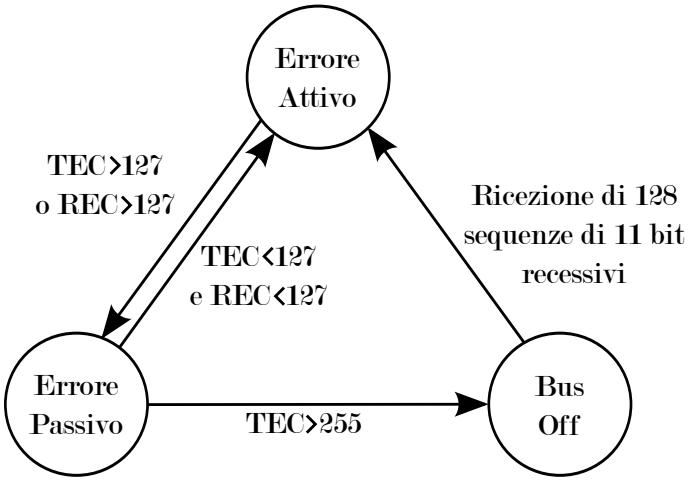


Figura 3.9: Riassunto delle transizioni tra gli stati di errore CAN

3.3 CANopen

Il protocollo CANopen®, descritto da una serie di specifiche sviluppate e supportate dall'organizzazione *CAN in Automation (CiA)*, fornisce un modello di comunicazione standardizzato, che permette compatibilità ed interoperabilità tra dispositivi anche di diversi costruttori, e capacità "plug-and-play" dei nodi sulla rete.

Si posiziona nel modello ISO/OSI, ai livelli superiori rispetto al CAN, per il quale definisce un livello applicativo (livello 7), come mostrato in Figura 3.10.

In una rete CANopen, è prevista la presenza di un solo master e più slave, e l'identificazione di ciascun nodo per mezzo di un node-ID, che deve essere univoco sulla rete. La comunicazione CAN deve avvenire utilizzando una delle bit rate previste dalle specifiche[14].

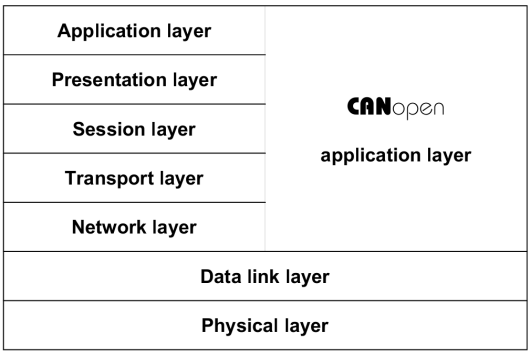


Figura 3.10: CANopen nel modello ISO/OSI

É possibile effettuare una suddivisione logica di un dispositivo CANopen in tre parti, come indicato in Figura 3.11:

- L'applicazione, che gestisce gli I/O ed i dati di processo (nel caso in esame i blocchi di I/O digitali, gli input analogici, e l'interfacciamento con il radiocomando), e le funzionalità di elaborazione necessarie per rendere disponibili i dati al dizionario degli oggetti.
- L'insieme degli oggetti di comunicazione (Paragrafo 3.3.3), che gestiscono le comunicazioni sulla rete CAN
- Il dizionario degli oggetti (Paragrafo 3.3.1), che rappresenta l'interfaccia tra i primi due elementi

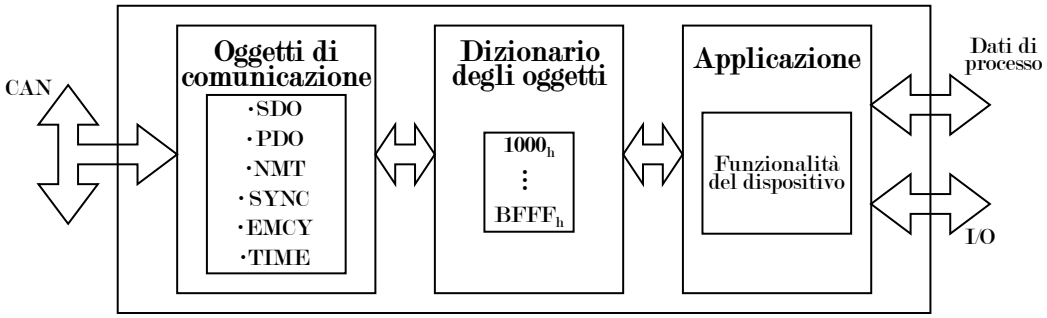


Figura 3.11: Struttura logica di un dispositivo CANopen

3.3.1 Dizionario degli oggetti

Il dizionario degli oggetti rappresenta il concetto alla base del protocollo CANopen: si tratta di un insieme di parametri accessibili dalla rete CANopen, in cui ciascun elemento (*oggetto*) è indirizzabile per mezzo di un *indice* a 16 bit ed un *sotto-indice* ad 8 bit. Alcuni degli oggetti presenti nel dizionario sono definiti dalle specifiche *communication profile*[14] o *device profile* (nel caso in esame è stato utilizzato il device profile DS-401[15]), mentre altri possono essere definiti liberamente dal costruttore. Il dizionario contiene oggetti necessari per la configurazione del dispositivo, l'accesso ai dati di processo, e la comunicazione, ed è costruito come da Tabella 3.1.

Ogni oggetto nel dizionario è caratterizzato dai seguenti parametri:

- **Indice:** campo a 16 bit per l'indirizzamento di un oggetto all'interno del dizionario
- **Sotto-indice:** campo ad 8 bit per l'indirizzamento degli oggetti, tra quelli appartenenti ad un dato indice
- **Nome:** descrizione testuale dell'oggetto, che ne facilita l'identificazione
- **Tipo:** indica il tipo dell'oggetto tra variabile, array, funzione, ecc.
- **Attributi:** indicano se l'oggetto è di sola lettura, sola scrittura, lettura/scrittura, costante
- **Obbligatorietà o meno di implementazione:** l'implementazione di ciascun oggetto può essere obbligatoria o opzionale per un dato tipo di dispositivo

Indici	Descrizione
0000h	non utilizzato
00001h - 025Fh	Tipi di dato
0260h - 0FFFh	riservati
1000h - 1FFFh	Oggetti definiti dal *communication profile*
2000h - 5FFFh	Oggetti definiti dal costruttore
6000h - 9FFFh	Oggetti definiti dal *device profile*
A000h - AFFFh	Variabili di rete
B000h - BFFFh	Variabili di sistema
C000h - FFFFh	riservati

Tabella 3.1: Struttura del dizionario degli oggetti

3.3.1.1 Electronic Data Sheet (EDS)

Ogni dispositivo CANopen deve essere accompagnato da un file di testo, denominato *Electronic Data Sheet* (EDS), riportante informazioni sul contenuto del dizionario degli oggetti in formato standardizzato[16] e con una precisa sintassi, al fine di facilitare il lavoro di configurazione necessario per l'integrazione del dispositivo nella rete.

3.3.2 Identificatori CAN

Le specifiche CANopen definiscono un'allocazione predefinita degli identificatori CAN ai servizi CANopen, in base al tipo di servizio ed al node-ID; in particolare, per gli identificatori CAN ad 11 bit, viene utilizzata la suddivisione mostrata in Figura 3.12, dove:

- Il campo *Node-ID*, costituito da 7 bit, identifica il nodo sorgente o destinazione del messaggio: è dunque possibile l'indirizzamento di un nodo master (Node-ID = 0) ed un massimo di 127 nodi slave.
- Il campo *function code* è necessario per distinguere il tipo di oggetto di comunicazione al quale appartiene il messaggio; codici più bassi sono assegnati ai protocolli di maggiore importanza (ad esempio il protocollo *Network Management*, che ha codice 0), in modo tale che i relativi identificatori CAN (CAN-ID) risultanti abbiano valore più basso, equivalente a maggiore priorità sulla rete CAN in caso di collisione (Paragrafo 3.2).

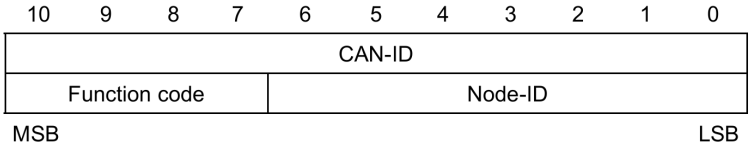


Figura 3.12: Struttura identificatore CAN in un sistema CANopen

L'allocazione dei *function code* ai servizi CANopen, e dei relativi CAN-ID risultanti in base ai possibili Node-ID, è riassunta in Figura 3.13.

COB	Function code	Resulting CAN-IDs
EMCY	0001_b	129 (081_h) – 255 ($0FF_h$)
PDO1 (tx)	0011_b	385 (181_h) – 511 ($1FF_h$)
PDO1 (rx)	0100_b	513 (201_h) – 639 ($27F_h$)
PDO2 (tx)	0101_b	641 (281_h) – 767 ($2FF_h$)
PDO2 (rx)	0110_b	769 (301_h) – 895 ($37F_h$)
PDO3 (tx)	0111_b	897 (381_h) – 1023 ($3FF_h$)
PDO3 (rx)	1000_b	1025 (401_h) – 1151 ($47F_h$)
PDO4 (tx)	1001_b	1153 (481_h) – 1279 ($4FF_h$)
PDO4 (rx)	1010_b	1281 (501_h) – 1407 ($57F_h$)
SDO (tx)	1011_b	1409 (581_h) – 1535 ($5FF_h$)
SDO (rx)	1100_b	1537 (601_h) – 1663 ($67F_h$)
NMT error control	1110_b	1793 (701_h) – 1919 ($77F_h$)

Figura 3.13: Allocazione dei *function code* e degli identificatori CAN ai servizi CANopen

3.3.3 Oggetti di comunicazione (COB)

Il protocollo CANopen prevede diversi oggetti di comunicazione (Communication Objects - COB), ciascuno dei quali realizza una specifica funzionalità, fornendo una serie di servizi, e comunica sulla rete CAN mediante protocolli. Il servizi, con le relative primitive, ed i protocolli, sono definiti dal *communication profile*[14].

3.3.3.1 Process Data Object (PDO)

I *Process Data Object (PDO)* sono utilizzati per trasferire ridotte quantità di dati (fino ad 8 byte) ad alta priorità, senza conferma, ma senza overhead aggiuntivo, tra un nodo produttore e zero o più nodi consumatori. Ciascun PDO implementato viene mappato ad uno specifico oggetto del dizionario degli oggetti (mediante indice e sotto-indice); tale mappatura rappresenta a sua volta un oggetto del dizionario, e può essere, se il dispositivo supporta tale operazione, modificato in fase di configurazione.

I PDO possono essere suddivisi in due tipi, in base alla direzione con la quale avviene lo scambio dei dati:

- **Transmission-PDO (TPDO)** se il dispositivo trasmette dati, divenendo il produttore del PDO
- **Receive-PDO (RPDO)** se il dispositivo riceve dati, assumendo il ruolo di consumatore del PDO

La trasmissione di un PDO può avvenire, a seconda della configurazione, secondo tre differenti modalità:

- **Trasmissione ad eventi:** la trasmissione avviene in corrispondenza di eventi esterni, o generati da timer
- **Richiesta remota:** la trasmissione del messaggio viene richiesta da un altro nodo mediante trama RTR (Paragrafo 3.2.4)
- **Trasmissione sincrona:** la trasmissione avviene in risposta alla ricezione di un messaggio di tipo SYNC (descritto più avanti)

3.3.3.2 Service Data Object (SDO)

Il protocollo *Service Data Object (SDO)* fornisce accesso completo al dizionario degli oggetti mediante indice e sotto-indice, con trasferimento di dati di qualunque dimensione, che avviene, a differenza di quanto avviene per i PDO, con conferma: per ogni richiesta di trasferimento, e per ogni trasferimento di dati, viene generato in risposta un messaggio che ne conferma l'avvenuta ricezione.

La comunicazione con SDO segue un modello client/server, in cui il nodo che fornisce l'accesso al proprio dizionario è definito *server*, mentre il nodo

che vi accede è detto *client*. Al fine di garantire l'accesso al proprio dizionario, ciascun dispositivo CANopen deve implementare almeno un SDO.

Per gli SDO, sono previsti tre tipi di trasferimento, ciascuno dei quali fornisce servizi per la lettura di un oggetto del dizionario (operazione denominata *upload*), la sua scrittura (*download*), l'inizializzazione della comunicazione (*initiate*), e l'annullamento di un trasferimento in corso (*abort*):

- **Expedited transfer:** trasferimento rapido a ridotto overhead, di ridotte quantità di dati, fino a 4 byte. Il trasferimento avviene mediante il solo servizio *initiate*, con l'incapsulamento nei dati all'interno del medesimo messaggio che ne richiede l'inizializzazione (*initiate*) se si tratta di una operazione di *download*, o nella relativa risposta per operazioni di *upload*. In Figura 3.14a e 3.14b è riassunta la modalità di trasferimento di un SDO mediante *expedited transfer*, rispettivamente per una operazione di *download* e di *upload*.

- **Segmented transfer:** trasferimento di quantità di dati qualsiasi, con suddivisione in segmenti di dimensione massima pari a 7 byte ciascuno. Ciascun trasferimento segmentato, come mostrato rispettivamente per operazioni di *download* e *upload* in Figura 3.15a e 3.15b, richiede un messaggio di inizializzazione della comunicazione (*initiate*), e la segnalazione del termine della comunicazione, che avviene contrassegnando l'ultimo segmento con un bit di segnalazione (c=1).

  É prevista una numerazione dei messaggi ad un solo bit (toggle, t), che cambia di stato tra un messaggio ed il successivo; la conferma deve possedere la medesima numerazione.

- **Block transfer:** per il trasferimento di grandi quantità di dati mediante suddivisione in blocchi composti a loro volta da un numero variabile (fino a 127) di segmenti ciascuno. L'implementazione di questo tipo di trasferimento è opzionale, e non presente nel progetto in esame, pertanto esso non verrà descritto in dettaglio.

La suddivisione in segmenti o blocchi, ed il limite nella quantità di dati che è possibile incapsulare in ciascun segmento, sono dovuti alla massima dimensione ammissibile per il campo dati dei pacchetti CAN, pari ad 8 byte (Paragrafo 3.2.4).

La comunicazione SDO utilizza due differenti CAN-ID, uno per la trasmissione dei messaggi, ed uno per la ricezione: questo è dovuto alla necessità

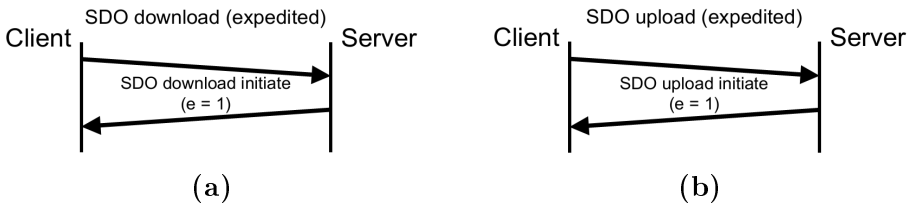


Figura 3.14: Modalità di trasferimento *SDO expedited transfer*

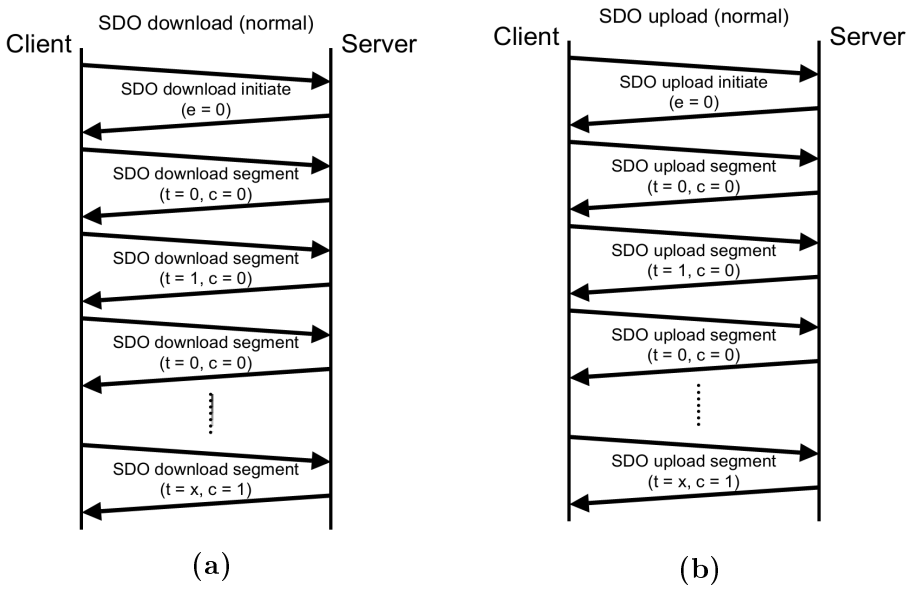


Figura 3.15: Modalità di trasferimento *SDO segmented transfer*

di poter differenziare i due flussi nella rete, che utilizzando lo stesso formato sarebbero altrimenti indistinguibili.

3.3.3.3 Synchronization object (SYNC)

Il protocollo SYNC, consiste nell'invio periodico di pacchetti broadcast, solitamente da parte del master, con lo scopo di generare eventi sincroni nei nodi della rete (ad esempio la trasmissione sincrona dei PDO). Il periodo che intercorre tra l'invio di due messaggi successivi è definito da un oggetto del dizionario, e può essere modificato durante la configurazione del dispositivo.

Il messaggio SYNC può essere affetto da jitter, nel caso in cui al momento di trasmissione stabilito sia già in corso un'altro trasferimento sul bus CAN,

o in caso di collisione al momento della trasmissione, con un pacchetto avente CAN-ID più prioritario (messaggi NMT).

3.3.3.4 Time stamp object (TIME)

Il protocollo TIME, pemette una sincronizzazione del timer di sistema tra i nodi che lo supportano, e prevede la trasmissione del tempo in formato assoluto, con un campo indicante il tempo trascorso dalla mezzanotte, espresso in millisecondi, e da un secondo campo contenente il numero di giorni trascorsi dal 1 gennaio 1984.

Anche in questo caso il messaggio può essere affetto da jitter, nel caso in cui al momento di trasmissione stabilito sia già in corso un'altro trasferimento sul bus CAN, o in caso di collisione al momento della trasmissione, con un pacchetto avente CAN-ID più prioritario (messaggi NMT o SYNC).

3.3.3.5 Emergency object (EMCY)

I messaggi di tipo EMCY vengono prodotti in seguito ad errori interni al dispositivo CANopen, e contengono indicazioni riguardo il tipo di errore, costituite da un codice a 16 bit standardizzato dal *communication profile* o dal *device profile*, più un campo la cui definizione è lasciata al costruttore, utile per una individuazione più specifica del tipo di errore.

Se il protocollo è supportato dal dispositivo, questo deve implementare almeno i due codici di errore corrispondenti ad un errore generico (0x1000), e alla risoluzione di una condizione di errore (0x0000).

La segnalazione di errore viene inviata una sola volta per ogni evento di errore, e può essere ricevuta da zero o più consumatori.

Un oggetto del dizionario tiene traccia, in forma di array, di tutti gli errori attivi sul dispositivo, nell'ordine cronologico in cui si sono verificati.

3.3.3.6 Network management (NMT)

Gli oggetti *network management* (NMT), rispondono ad un modello di comunicazione master/slave, in cui è obbligatoreio che uno dei nodi della rete assuma la funzione di master. Essi offrono due tipi di servizi:

- **Servizi di controllo dei nodi**, mediante i quali il master della comunicazione NMT è in grado di controllare lo stato degli slave, ed in particolare:

  - Avviare un nodo
  - Fermare un nodo
  - Resettare un nodo

- **Servizi per il controllo dei guasti**, basati sulla trasmissione periodica di messaggi, che permettono di identificare avarie nel funzionamento dei nodi:
  - **Node guarding:** consiste nell'invio periodico di una interrogazione da parte del master a ciascuno degli slave; se lo slave non risponde nel tempo stabilito, il livello applicativo del nodo master riceve una indicazione di errore
  - **Hearbeat:** consiste nell'invio periodico al master di un messaggio da parte di ciascuno slave, che ne segnala il corretto funzionamento. La mancata ricezione del messaggio heartbeat nel tempo stabilito comporta una indicazione di errore nel livello applicativo del nodo master

É prevista l'implementazione, all'interno dell'oggetto di comunicazione NMT di ciascun nodo, di una macchina a stati, le cui transizioni sono controllate dal master, mediante i servizi di controllo dei nodi. Sono previsti i seguenti stati:

- **Initialisation:** É lo stato in cui il nodo entra automaticamente all'avvio o quando viene resettato, ed è costituito dall'inizializzazione del dispositivo e della comunicazione CANopen. Al termine di tali operazioni, il nodo entra autonomamente nello stato *Pre-operational*
- **Pre-operational:** In questo stato solitamente vengono configurati i parametri del nodo (tra cui la mappatura dei PDO), per poi avviare il nodo, facendolo passare allo stato *operational*. Non è disponibile la comunicazione mediante oggetti PDO, che devono ancora essere creati.
- **Operational:** In questo stato tutti gli oggetti di comunicazione sono attivi. Può essere limitato l'accesso in scrittura ad alcuni oggetti del dizionario.
- **Stopped:** La transizione allo stato *Stopped*, forza la disattivazione di tutti gli oggetti di comunicazione, ad eccezione dei protocolli *Node Guarding* e *Heartbeat*, se utilizzati.

Le possibili transizioni tra gli stati sono riassunte in Figura 3.16, in cui è possibile notare come da qualunque stato, mediante i servizi di reset, sia possibile far tornare il nodo allo stato di inizializzazione.

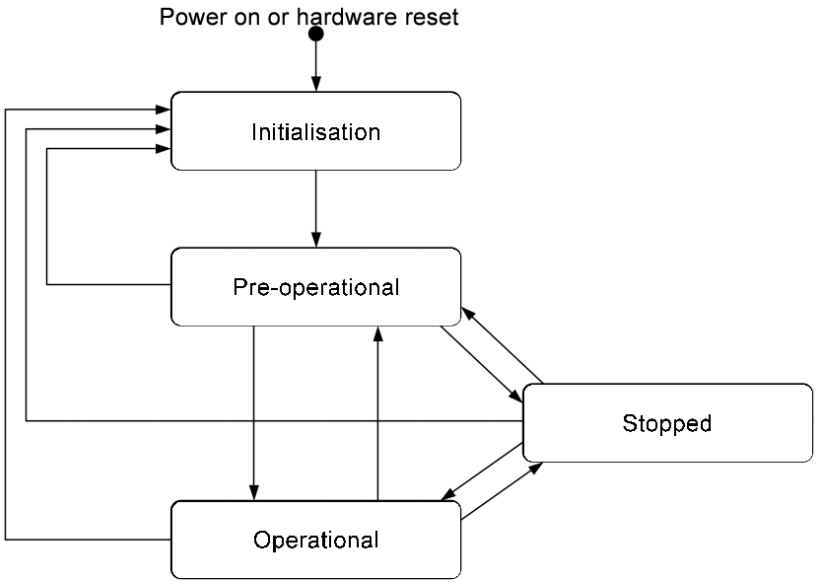


Figura 3.16: Transizioni della macchina a stati NMT

3.3.4 Connettori

La scelta del connettore CAN, vengono fornite diverse alternative, tra le quali è stata selezionata per il presente progetto quella denominata "open style connector"[17]; si tratta di un connettore estraibile a 3 pin, come mostrato in Figura 3.17, che presenta la piedinatura riportata in Tabella 3.2.

Pin	**Funzione**
1	CAN_L
2	Schermatura (opzionale)
3	CAN_H

Tabella 3.2: Piedinatura del connettore CANopen

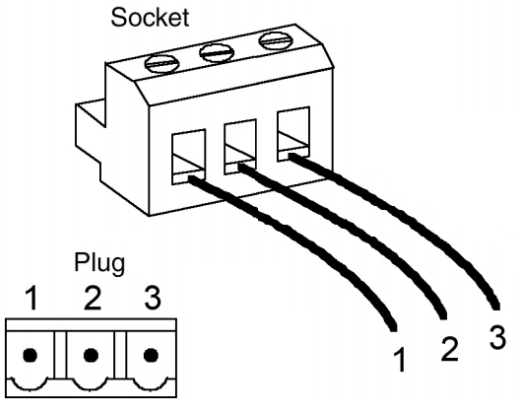


Figura 3.17: Connettore CANopen

3.3.5 Indicatori

Le specifiche CiA[18] definiscono la presenza di due indicatori a LED di colore rosso e verde (o di un unico indicatore bicolore), utilizzati rispettivamente per segnalare errori nello strato fisico CAN, e lo stato della macchina a stati relativa all'oggetto di comunicazione NMT.

Per ciascun indicatore vengono definiti i tipi di lampeggio associati a ciascuna condizione; questo tipo di standardizzazione è utile per permettere una rapida diagnosi dei problemi di comunicazione, senza la necessità di accedere ai manuali dei singoli dispositivi.

3.4 Protocollo radio

Per la comunicazione radio tra le due schede, è stato predisposto un protocollo proprietario che prevede la trasmissione dei dati tra due agenti, uno master ed uno slave, mediante incapsulamento in pacchetti di dimensioni fisse. Il protocollo è in grado di selezionare automaticamente un canale libero e di instaurare una comunicazione su di esso, e di ristabilire il collegamento nel caso in cui questo venga a mancare o risulti inaffidabile.

3.4.1 Modalità di funzionamento

Durante il normale funzionamento il master, ovvero la scheda *consolle* invia pacchetti allo slave (scheda *macchina*) con un periodo T_{pkt} fisso, pari a $10ms$. Se un pacchetto viene ricevuto correttamente, la *macchina* genera una risposta di tipo *acknowledge* (pacchetto di tipo *ACK*, descritto al Paragrafo 3.4.3.2), che deve giungere al master entro un tempo $T_{ack} < T_{pkt}$. In caso di mancata ricezione del pacchetto *ACK* entro T_{ack}, la *consolle* suppone che sia stato perso un pacchetto o la corrispondente risposta, e provvede all'invio del pacchetto successivo, fino ad un massimo di 3 tentativi in assenza di risposta (in Figura 3.18 è mostrato il comportamento del sistema in risposta alla perdita di un pacchetto *dati*).

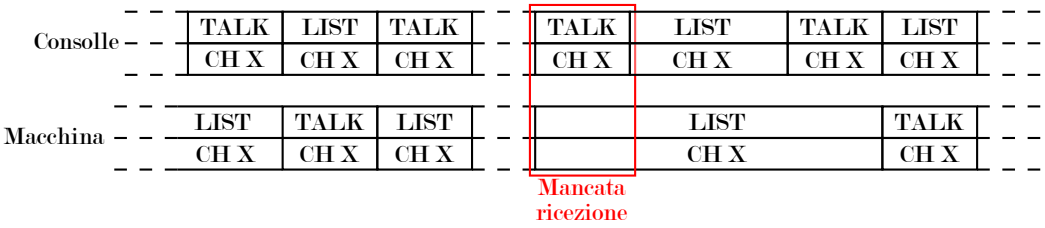


Figura 3.18: Esempio di perdita di un pacchetto

Superato tale massimo, si considera il collegamento caduto definitivamente, ed ha inizio una fase detta di *aggancio* (Paragrafo 3.4.2), in cui si tenta di ristabilire il collegamento su un canale differente, nel minore tempo possibile. La *macchina* controlla invece il tempo che intercorre tra la ricezione di due pacchetti consecutivi: noto il tempo che intercorre tra l'invio di due pacchetti, sempre pari a T_{pkt}, ed il numero di tentativi effettuati, se dopo un tempo pari a $3 \cdot T_{pkt}$ non viene ricevuto alcun pacchetto, anche la *macchina* attiva la procedura di aggancio. Se invece il numero di pacchetti persi non

supera il limite massimo, la comunicazione riprende normalmente sul canale già in uso.

3.4.2 Procedura di aggancio

La procedura di *aggancio* rappresenta la serie di operazioni necessarie a stabilire (o ripristinare) la comunicazione tra le due schede; viene attivata all'accensione, oppure ogni volta che durante il funzionamento, il link di comunicazione viene interrotto da cause esterne (distanza superiore alla portata, interferenze, ecc.). L'algoritmo di aggancio è stato costruito in modo tale da rendere veloce il ripristino della comunicazione, senza tuttavia disturbare eventuali altre apparecchiature radio dello stesso tipo o di altro genere, funzionanti nella stessa banda di frequenza. Nel caso di utilizzo della macchina senza il controllo remoto inoltre, si evita di impegnare il mezzo radio inutilmente: se non interrogata dal radiocomando (*consolle*), la *macchina* resta continuamente in ascolto, senza però entrare mai in trasmissione. Il sistema è inoltre in grado di verificare quali siano i canali più liberi, sui quali sarà più probabile instaurare una comunicazione con successo. A tale scopo si utilizza la funzionalità del modulo radio, detta *RSSI* (Received Signal Strength Indication), che permette di ricavare un valore numerico proporzionale alla potenza del segnale ricevuto. Quando il link viene perso, la *consolle* effettua la lettura del valore RSSI di tutti i canali, e sceglie quello che restituisce il valore minore, iniziando a trasmettere sul canale scelto, e rimanendo in ascolto dopo ogni trasmissione per rilevare l'eventuale risposta da parte della *macchina*.

La *macchina* resta invece in ascolto su tutti i canali in sequenza, con un tempo di permanenza su ciascun canale T_{list}, almeno pari a quello necessario a ricevere un pacchetto completo nel caso peggiore, ovvero $2 \cdot T_{pkt}$. Alla corretta ricezione di un pacchetto su uno dei canali, viene generato un pacchetto ACK (Paragrafo 3.4.3.2) di risposta sullo stesso canale; se anche l'ACK generato viene ricevuto correttamente dalla *consolle*, si considera l'aggancio completato, e la comunicazione inizia (o riprende) sul nuovo canale. Se invece l'ACK non viene ricevuto entro un tempo pari a quello necessario alla *macchina* per l'ascolto su tutti i canali, equivalente ad un numero di tentativi $(T_{list} * N)/T_{pkt}$ (con N numero di canali disponibili), la *consolle* seleziona un altro canale libero su cui tentare un nuovo aggancio secondo le modalità appena descritte. In Figura 3.19 è visibile un esempio di aggancio portato a termine con successo nel tempo massimo possibile (supponendo che le due

schede si trovino in condizioni tali da permettere la comunicazione sul canale scelto).

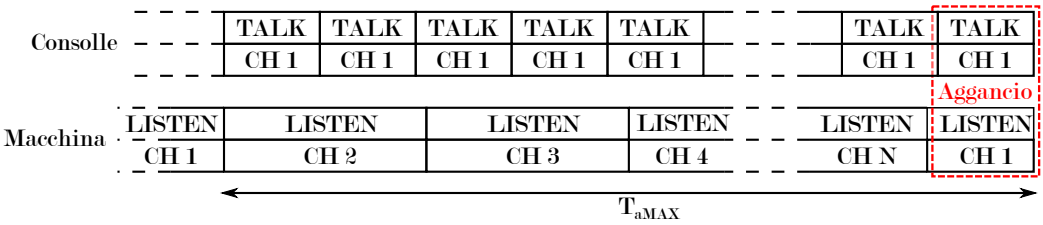


Figura 3.19: Aggancio, caso peggiore

3.4.3 Struttura dei pacchetti

La comunicazione radio consiste nello scambio di pacchetti di dimensioni fisse tra gli agenti *consolle* e *macchina*. La comunicazione è di tipo asimmetrico, per la maggiore quantità di dati che deve essere spostata dalla *consolle* alla *macchina* e half-duplex per le caratteristiche dei moduli radio. I pacchetti scambiati sono di due tipi: i pacchetti *dati*, trasmessi dalla *consolle* e contenenti lo stato dei manipolatori e i comandi, e i pacchetti *ACK*, trasmessi dalla macchina come conferma. Per entrambi i tipi di pacchetto, si è cercato di rendere minima la quantità di dati trasmessa, in modo da massimizzare la probabilità che un intero pacchetto sia ricevuto correttamente.

3.4.3.1 Pacchetto dati

Questo tipo di pacchetti, trasmessi dalla *consolle* alla *macchina*, contiene informazioni sulla posizione dei manipolatori e sullo stato degli switch integrati negli stessi (Paragrafo 2.9). Il pacchetto è composto come da Tabella 3.3, dove:

- **ADD:** Indirizzo univoco a 16 bit associato a ciascuna *consolle* e a ciascuna *macchina* in fase di collaudo, e non modificabile dall'utente; la comunicazione può avvenire solo tra due schede a cui sia associato il medesimo indirizzo.
- **TYPE:** Campo disponibile per una futura implementazione di diversi tipi di pacchetto, ad esempio per segnalare la pressione del pulsante di emergenza (attualmente non utilizzato e posto di default a 0).

Byte	Contenuto
0	ADD [15..8]
1	ADD [7..0]
2	TYPE
3	SW [7..0]
4	AN0
5	AN1
6	AN2
7	AN3

Tabella 3.3: Struttura di un pacchetto di tipo *dati*

- **SW:** Stato degli switch presenti sui manipolatori (2 switch per ogni manipolatore), che permettono una verifica ridondante dell'attivazione o meno dei manipolatori. Il bit si trova a 1 quando lo switch è chiuso.
- **ANx:** Valore ad 8 bit che indica la posizione del manipolatore; un valore pari a 0 indica che il manipolatore è in posizione di riposo, un valore vicino a 255 indica che il manipolatore è spostato a fine corsa in una delle due direzioni possibili.

3.4.3.2 Pacchetto ACK

Il pacchetto *ACK* rappresenta la conferma di avvenuta ricezione di un pacchetto *dati* da parte della *macchina*, e contiene un solo byte di dati, utile per la ripetizione di segnalazioni dal macchinario al radiocomando. Allo stato attuale tale meccanismo non è stato ancora implementato, e il byte è trasmesso con valore costante e pari a 0.

Le strutture presentate non prevedono l'introduzione di meccanismi di rivelazione o correzione degli errori; a garanzia dell'integrità dei dati scambiati, il modulo radio integra un controllo CRC16.

Capitolo 4

Progetto e realizzazione dell'hardware

I dispositivi selezionati al Capitolo 2 per poter funzionare, hanno bisogno di essere interconnessi tra loro e con l'esterno, alimentati se necessario, e fissati su di un supporto meccanico; oggetto del presente capitolo sarà dapprima la creazione di una rappresentazione astratta dei componenti e delle interconnessioni, lo *schema elettrico*, e a partire da tale rappresentazione, il disegno del circuito stampato su cui saranno, sul quale saranno posizionati fisicamente i componenti.

4.1 Disegno dello schema elettrico

Facendo riferimento al datasheet dei singoli componenti per i parametri di funzionamento necessari, è stata creata una rappresentazione circuitale del sistema, utilizzando la funzionalità *Schematic Editor* offerta dal software *EAGLE*, di cui Figura 4.1 mostra l'aspetto durante una fase del lavoro.

Tale rappresentazione è costituita da simboli convenzionali associati a ciascun componente, connessi da linee che schematizzano i collegamenti elettrici tra i dispositivi. Il software dispone di librerie da cui prelevare i componenti, e nel caso in cui i componenti necessari non fossero presenti, permette di crearne di nuovi.

Lo schema completo è presentato in Appendice A.

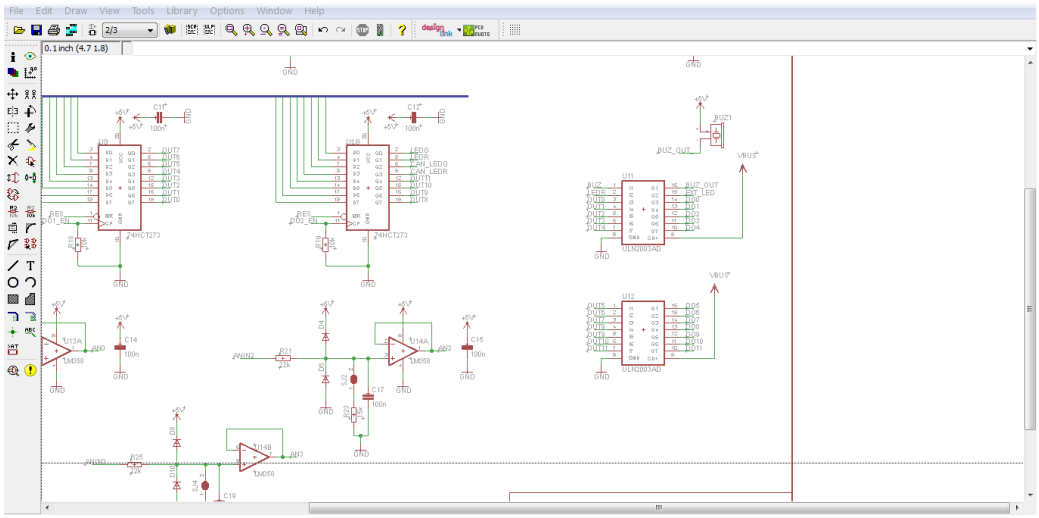


Figura 4.1: Ambiente *Schematic Editor* di EAGLE

4.2 Disegno del circuito stampato

Scopo di questa fase della progettazione, è il disegno dei collegamenti fisici tra i componenti del circuito, che nella successiva fase di produzione verranno realizzati sotto forma di circuito stampato. Il costo della produzione è fortemente dipendente dal numero di layer utilizzati, che dovrà dunque essere limitato al minimo necessario. Per questo progetto sono stati realizzati circuiti stampati a due layer, con componentistica in parte a montaggio superficiale (Surface Mounting Device - SMD) ed in parte a foro passante (through-hole technology - THT).

Il software *Layout Editor* di *EAGLE*, permette l'importazione dei collegamenti e dei componenti utilizzati dallo schema elettrico; in seguito all'importazione è stato comunque necessario un controllo della corrispondenza dei footprint inseriti automaticamente dal software, spesso differenti da quelli dei componenti effettivamente utilizzati, e talvolta la creazione di un nuovo footprint per i componenti non presenti nelle librerie del software.

É stato poi disegnato un modello dello spazio interno del contenitore per guida DIN, prendendo come riferimento le quote indicate dal costruttore[19]. Noto lo spazio a disposizione, si è potuto procedere con un primo posizionamento dei componenti, che è stato eseguito partendo da quelli la cui posizione è obbligata e non modificabile:

- LED di segnalazione, che devono essere visibili anche quando il circuito è installato
- Connettori estraibili, posizionati obbligatoriamente sui lati lunghi del contenitore, sul quale sono predisposte delle aperture per l'accesso agli stessi
- Connettore RS232, predisposto per la diagnosi e per eventuali aggiornamenti del firmware

Il posizionamento dei componenti deve inoltre tenere conto dei fori necessari per il fissaggio del circuito al contenitore.

Le dimensioni del contenitore e la quantità di componenti presenti, hanno costretto alla suddivisione del circuito in due schede, unite poi elettricamente da un connettore, e meccanicamente da una serie di distanziali.

Si è infine completato il posizionamento dei componenti in modo da renderne possibile il collegamento sfruttando i due layer a disposizione, ottenendo il risultato visibile in Figura 4.2, in cui sono riportati con delle linee gialle i collegamenti non ancora eseguiti.

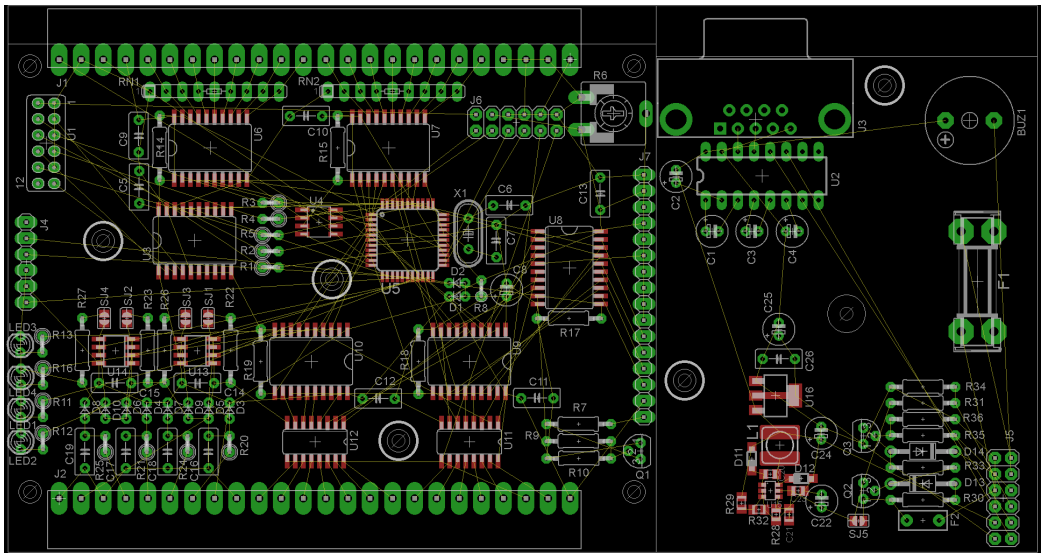


Figura 4.2: Posizionamento dei componenti

Terminato il posizionamento, sono stati disegnati i collegamenti, fino a raggiungere il layout definitivo, presentato in Appendice B. Un esempio che

mostra il software *Layout Editor* di *EAGLE* durante questa fase, è mostrato in Figura 4.3.

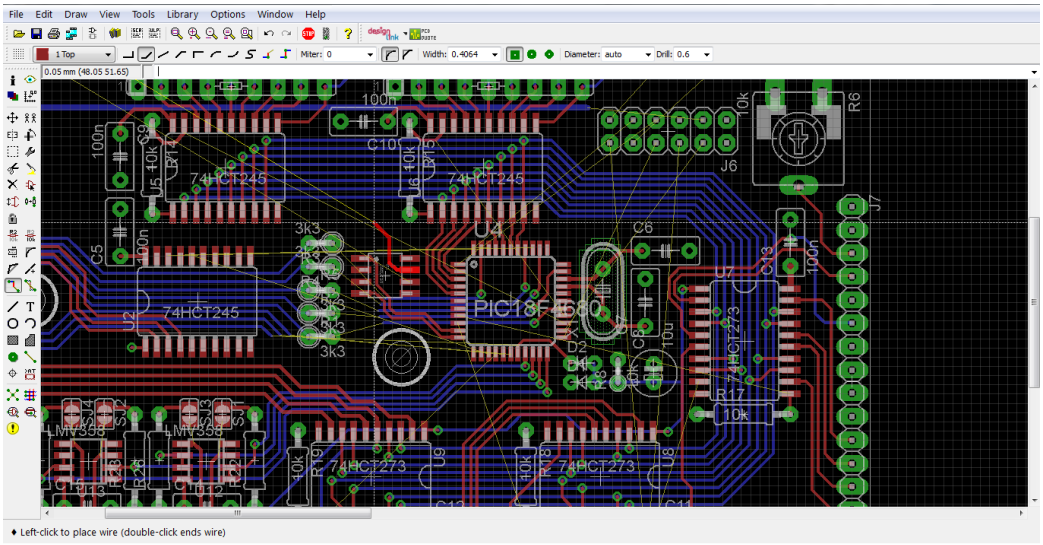


Figura 4.3: Ambiente *Layout Editor* di *EAGLE*

4.3 Realizzazione dei prototipi

Sulla base dei file di lavorazione generati dal software di disegno, un fornitore esterno ha prodotto i circuiti stampati, che si presentano, come da Figura 4.4, con i collegamenti in rame rivestiti da un materiale isolante, detto *solder mask*, in questo caso di colore verde, e i punti che necessitano di saldatura di colore argentato, per la presenza di un rivestimento superficiale che ha lo scopo di facilitare la saldatura. Sono inoltre presenti in bianco le serigrafie che riportanti i nomi e la sagoma dei componenti, per facilitarne un posizionamento esente da errori.

Ai circuiti stampati sono stati saldati manualmente i componenti, completando così la realizzazione dei prototipi (Figura 4.5).

In seguito i prototipi sono stati posti in condizioni di funzionamento che riproducono quelle reali:

- La scheda *consolle* è stata collegata a due manipolatori ed un display LCD, alimentata con la batteria prevista dalle specifiche, ed inserita all'interno di un contenitore.

- La scheda *macchina* è stata collegata ad un analizzatore CANopen, a dei pulsanti e dei LED per verificare il funzionamento dei blocchi di input e output digitali, e a dei potenziometri per simulare il comportamento degli ingressi analogici. L'alimentazione è stata fornita tramite un alimentatore variabile da laboratorio.

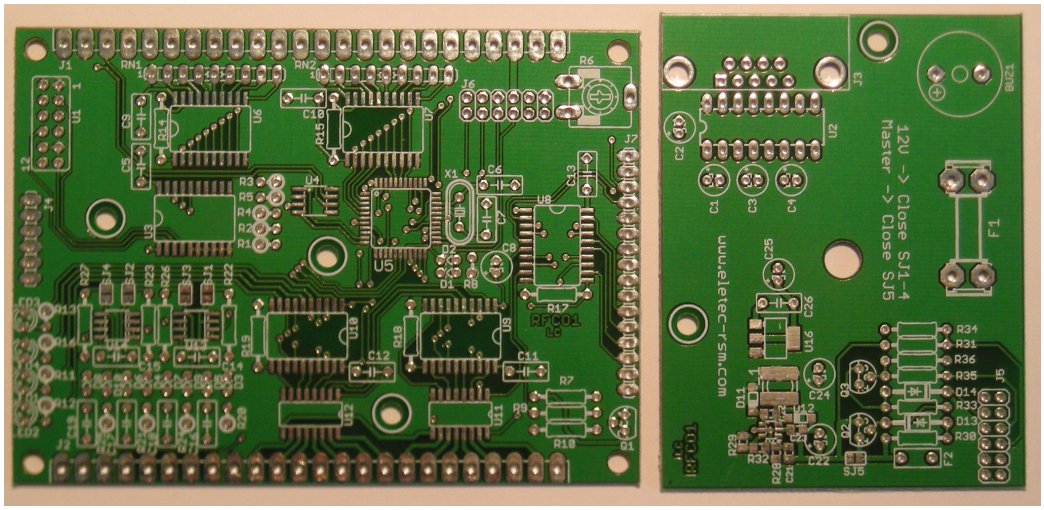


Figura 4.4: Foto del circuito stampato

4.4 Collaudo

Il collaudo dei prototipi è stato effettuato facendo eseguire al microcontrollore di ciascuna scheda, semplici porzioni di codice, atte a testarne le funzionalità. I prototipi si sono dimostrati funzionanti, permettendo di proseguire con la successiva fase di sviluppo del firmware vero e proprio.

É prevista, ma non ancora implementata, la possibilità di collaudare le schede finite eseguendo una apposita routine, presente nella memoria del microcontrollore, la cui esecuzione è richiesta dall'operatore che esegue il collaudo, con l'invio di comandi sulla porta di diagnosi. Nella stessa fase dovrebbe anche essere associato a ciascuna coppia *consolle/macchina* il numero di identificazione univoco per la comunicazione radio (procedura di *bind*).

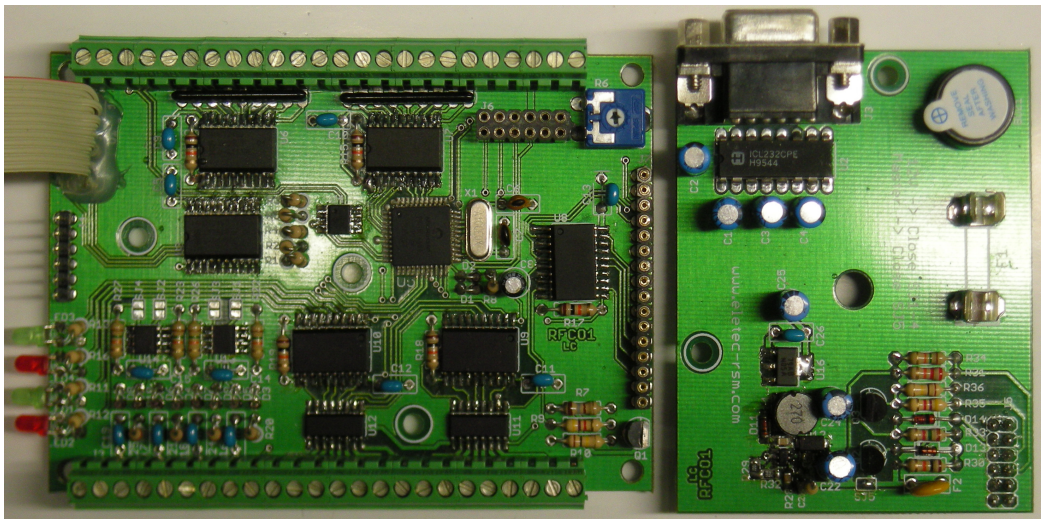


Figura 4.5: Circuito stampato in seguito al montaggio dei componenti

Capitolo 5

Sviluppo del firmware

Il firmware per il microcontrollore PIC18F4680, è stato scritto principalmente in linguaggio C, con alcune parti in Assembly. Il compilatore utilizzato è MPLAB® C18, compatibile con la famiglia PIC18 di Microchip, ed integrato nell'ambiente di sviluppo MPLAB® IDE.

Le funzionalità dei dispositivi *macchina* e *consolle* sono state suddivise in moduli, ciascuno dei quali esegue un compito (task) specifico; ciascun modulo è stato sviluppato in modo da risultare autonomo dagli altri, al fine di semplificare il debug dell'applicazione, che in questo modo può essere eseguito testando le funzionalità una per volta, escludendo le altre.

I task sono eseguiti in modo sequenziale, virtualmente parallelo; si tratta di un multitasking di tipo cooperativo, in cui ad ogni ciclo del programma principale ciascun modulo esegue operazioni semplici, per poi lasciare la CPU del microcontrollore libera per l'esecuzione del task successivo; per permettere la gestione di tutte le funzionalità evitando il rischio di *starvation* per alcune di esse, con conseguenti ritardi nella risposta agli eventi, è necessaria l'assenza di porzioni di codice bloccante, e che il tempo di esecuzione di ciascun task sia limitato.

In particolare ciascun modulo, appena entra in esecuzione, verifica la necessità o meno di eseguire una nuova operazione sull'hardware, per poi eventualmente richiederne l'avvio, se necessario inizializzare la periferica affinché ne segnali il termine, e liberare quindi la CPU. In nessun caso il programma deve bloccarsi nell'attesa del termine dell'operazione. Se invece non ci sono operazioni da avviare, o si è già in attesa del termine di un'operazione gestita in hardware, viene ceduta immediatamente la CPU al task successivo.

Il meccanismo descritto permette, di raggiungere tempi di esecuzione del ciclo principale ridotti (stimati tra i 300 ed i 400 μs) su entrambe le versioni di firmware (*consolle* e *macchina*), ed una latenza che è stata giudicata accettabile in rapporto all'applicazione.

Altro obbiettivo è stato lo sfruttamento delle similitudini dell'hardware dei due dispositivi realizzati, al fine del massimo riuso del codice tra le applicazioni dedicate a *consolle* e *macchina*, minimizzando il tempo di sviluppo e di debug. In particolare, è stato possibile il riuso di tutti i driver, sia delle periferiche integrate nel microcontrollore (ADC, SPI, timer), che di quelle esterne ad esso (blocchi di I/O digitali, modulo radio).

5.1 Timer

Viene utilizzato uno dei timer integrati nel microcontrollore, configurato per ottenere una interruzione ad intervalli di 1 ms. In corrispondenza di tali interruzioni, viene incrementato un contatore a 16 bit, che rappresenta il timer di sistema, ed il cui valore viene reso disponibile ai diversi task, che lo possono utilizzare per gestire temporizzazioni in modo non bloccante: eseguendo una operazione di differenza tra il valore del timer in due istanti differenti, è possibile la misura del tempo in millisecondi che intercorre tra i due istanti, fino ad un massimo di 65,535 s.

Il timer viene utilizzato principalmente per rilevare condizioni di *timeout*: contestualmente all'avvio di una operazione, viene salvato il valore del timer di sistema. Ad ogni successiva esecuzione, viene misurato il tempo trascorso dall'avvio dell'operazione, e se questo supera il valore stabilito, viene segnalata la condizione di *timeout*, che comporta l'annullamento dell'operazione, ed il ripristino della relativa periferica in uno stato libero e pronto ad accettare nuove richieste.

5.2 Driver

I driver rappresentano porzioni di codice che permettono il pilotaggio da parte del microcontrollore, dei componenti hardware presenti a bordo dei dispositivi o integrati nel microcontrollore stesso.

Ciascun driver offre la funzionalità `module_Init()` (dove *module* è il nome del modulo), che è necessario chiamare all'avvio del dispositivo; tale pro-

cedura esegue l'inizializzazione dell'hardware e dei parametri necessari al funzionamento del modulo stesso.

5.2.1 SPI

Il driver SPI configura il modulo MSSP integrato nel microcontrollore (Paragrafo 2.1) per la comunicazione in modalità SPI con il modulo radio. Sono fornite primitive per la trasmissione e la ricezione di un byte.

5.2.2 I/O digitali

Il driver degli input e output digitali, realizza il pilotaggio dei blocchi di ingressi ed uscite digitali, secondo quanto descritto al Paragrafo 2.2. Sono fornite primitive per la lettura/scrittura di interi blocchi di ingressi/uscite, e per la lettura/scrittura di singoli ingressi/uscite.

Viene tenuta traccia nella memoria dati del microcontrollore dello stato delle uscite, in modo da poter risalire all'ultimo stato imposto.

5.2.3 ADC

Il driver configura ed utilizza il modulo ADC integrato nel microcontrollore: ad ogni ciclo del programma principale, se il modulo ADC è libero, viene lanciata automaticamente la conversione su uno dei canali, coprendo in sequenza tutti quelli utilizzati. I dati relativi all'ultima conversione effettuata su ciascun canale sono raccolti in un buffer, a cui l'applicazione può accedere in qualunque momento.

5.2.4 Modulo radio

Il driver per il modulo radio si basa, per il trasferimento di dati e comandi, a sua volta sul driver SPI, descritto al Paragrafo 5.2.1. Fornisce le primitive necessarie alla configurazione del modulo mediante lettura/scrittura dei registri, alla ricezione/trasmissione di dati, e all'utilizzo delle funzioni di risparmio energetico del modulo.

Il driver implementa inoltre una macchina a stati per tenere traccia delle operazioni in corso da parte del modulo, e restituire codici di errore in caso si tenti di eseguire un'operazione non compatibile con lo stato corrente. La

transizione tra gli stati si ha in corrispondenza della richiesta di interruzione relativa all'operazione in corso, generata dal modulo radio mediante la linea IRQ (Paragrafo 2.7). Le operazioni di trasmissione ricezione comportano segnalazioni di *timeout* se non completate nei tempi stabiliti: in caso di *timeout* in ricezione, viene semplicemente richiesto l'annullamento dell'operazione, mentre in caso di blocco in trasmissione il modulo viene resettato con relativa nuova inizializzazione.

5.2.5 LCD

Il driver del display LCD permette l'invio di dati e comandi a tale modulo; essendo il display pilotato da un blocco di output digitali, il driver si appoggia a sua volta sul driver degli I/O digitali (Paragrafo 5.2.2).

Sono supportati display di qualunque dimensione, e l'indirizzamento mediante i parametri *riga* e *colonna* per il posizionamento del cursore.

Le primitive di scrittura di variabili o stringhe sul display, non agiscono direttamente su di esso, ma solo su un buffer nella memoria dati del microcontrollore, che rappresenta una copia dei caratteri visualizzati. Ad ogni ciclo del programma principale, viene avviata la scrittura di un solo carattere sul display, coprendo in sequenza tutti i caratteri: in questo modo si ottiene, rispetto alla scrittura diretta di dati e variabili sul display, una diminuzione del tempo necessario alla gestione del display, che per il display da 20 x 4 caratteri utilizzato nel presente lavoro è stato stimato stimato in quasi due ordini di grandezza. Questo metodo di gestione comporta però una diminuzione della frequenza di aggiornamento del display, il quale però non ha fatto riscontrare effetti visibili durante i test.

5.3 Consolle

L'applicazione relativa alla *consolle*, descritta sinteticamente dal diagramma in Figura 5.1a implementa, sfruttando le primitive fornite dal driver del modulo radio, la parte relativa alla *consolle* del protocollo di comunicazione radio descritto al Paragrafo 3.4.

É inoltre gestita la visualizzazione sul display LCD come interfaccia tra il radiocomando e l'operatore, la gestione dei manipolatori, mediante le funzionalità degli I/O digitali per la determinazione dello stato degli switch integrati negli stessi, e dell'ADC per il controllo proporzionale. I dati re-

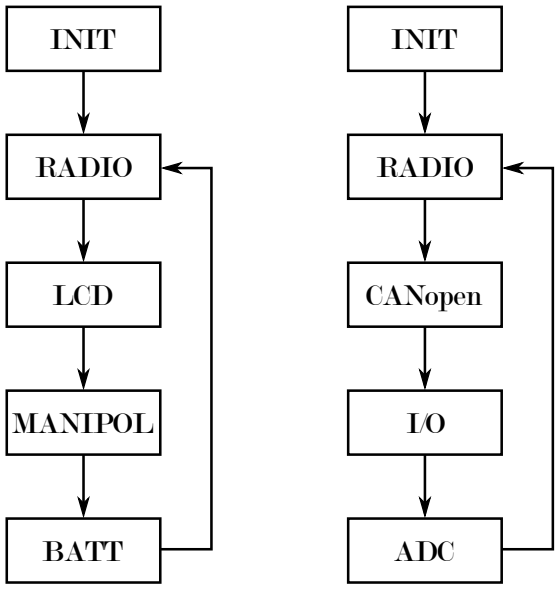
lativi ai manipolatori, in seguito all'acquisizione, sono trasferiti al buffer di trasmissione radio.

Viene infine gestito lo stato della batteria, con le relative segnalazioni sullo stato di carica ed un avviso testuale sul display in prossimità della scarica totale.

5.4 Macchina

L'applicazione *macchina*, descritta in Figura 5.1b, implementa la parte l'algoritmo descritto al Paragrafo 3.4 relativa alla *macchina*, la gestione dei blocchi di input e output digitali, degli ingressi analogici, e dei servizi CANopen per i quali è stata presa come base l'application note *AN945*[20] fornita da Microchip.

Al fine di rendere i dati, riguardanti l'azionamento dei manipolatori da parte dell'operatore, disponibili sulla rete CANopen come richiesto dalle specifiche, tali dati sono stati mappati su oggetti assegnati dal *device profile*[15] ad ingressi digitali (per gli switch), e ad ingressi analogici (per la tensione del potenziometro). Tali oggetti sono poi stati mappati sui PDO predefiniti.



(a) *consolle* **(b)** *macchina*

Figura 5.1: Diagramma di flusso del firmware

Conclusioni

Il sistema progettato soddisfa, e talvolta supera, le specifiche imposte al Capitolo 1.

I test in laboratorio ed in campo aperto hanno verificato la bontà dell'algoritmo sviluppato per la negoziazione del canale di trasmissione tra *consolle* e *macchina*: l'aggancio è sempre avvenuto al primo tentativo, in tempi mediamente molto più brevi rispetto al caso peggiore.

É stato inoltre verificato il basso tasso di errore per bit del modulo radio, permesso dalla tecnica DSSS, ed in particolare, testando il collegamento con la trasmissione di dati noti in ricezione, anche con il controllo CRC16 disabilitato non sono si sono mai verificati errori nei dati scambiati.

In campo aperto, all'aumentare della distanza di trasmissione fino ad avvicinarsi alla portata, si nota che i pacchetti vengono persi con maggiore frequenza, senza tuttavia corruzione dei dati in quelli che giungono a destinazione. Aumentano ulteriormente la distanza, sopraggiunge piuttosto bruscamente la perdita totale del collegamento. Questo comportamento è spiegabile con il fatto che, nonostante all'aumentare della distanza i chip vengano corrotti in misura via via maggiore, una sequenza contenente chip errati ha una probabilità molto ridotta di divenire sufficientemente correlata con il codice corrispondente ad un bit di segno opposto, ma diviene piuttosto non correlata a nessuna delle sequenze, causando la perdita del pacchetto.

In ambiente urbano invece, è stato verificato che il sistema riesce, mediante funzionalità RSSI, ad identificare ed evitare i canali occupati da reti Wi-Fi.

Una possibile criticità individuabile nella comunicazione wireless è, nonostante la relativa difficoltà di decodifica delle sequenze codificate DSSS in mancanza del codice PN utilizzato in fase di codifica, la possibilità di manipolazioni dei dati da parte di soggetti malintenzionati. Al fine di evitare tale eventualità, sarebbe dunque opportuna l'introduzione di tecniche crittografiche.

Appendice A

Schema elettrico

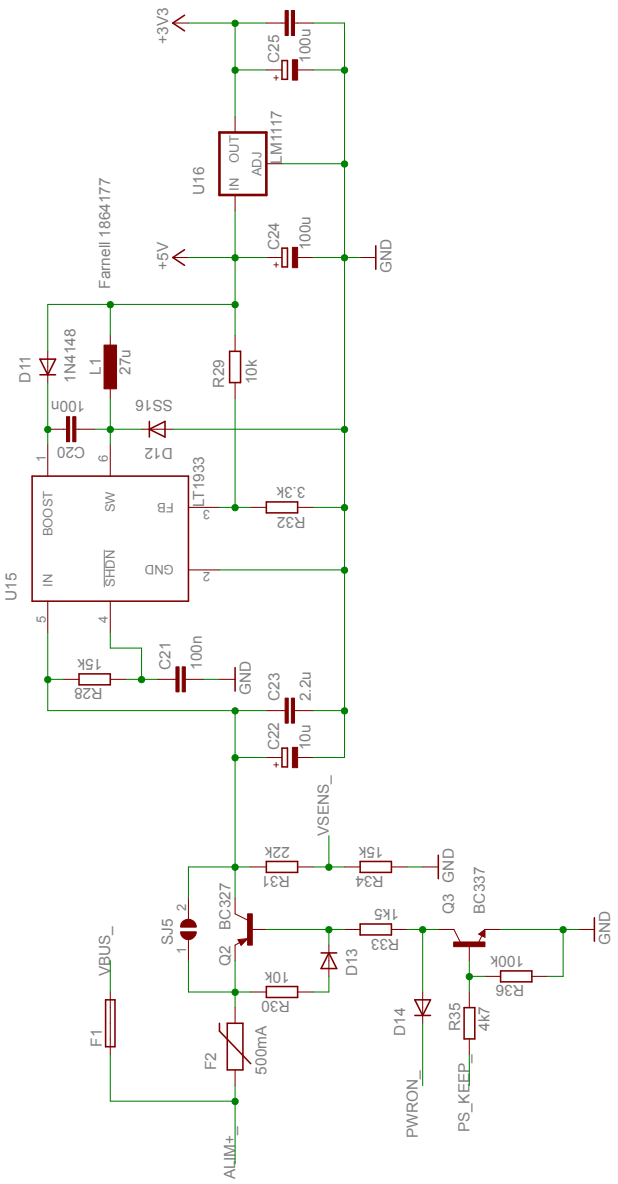


Figura A.1: Schema elettrico, sezione di alimentazione

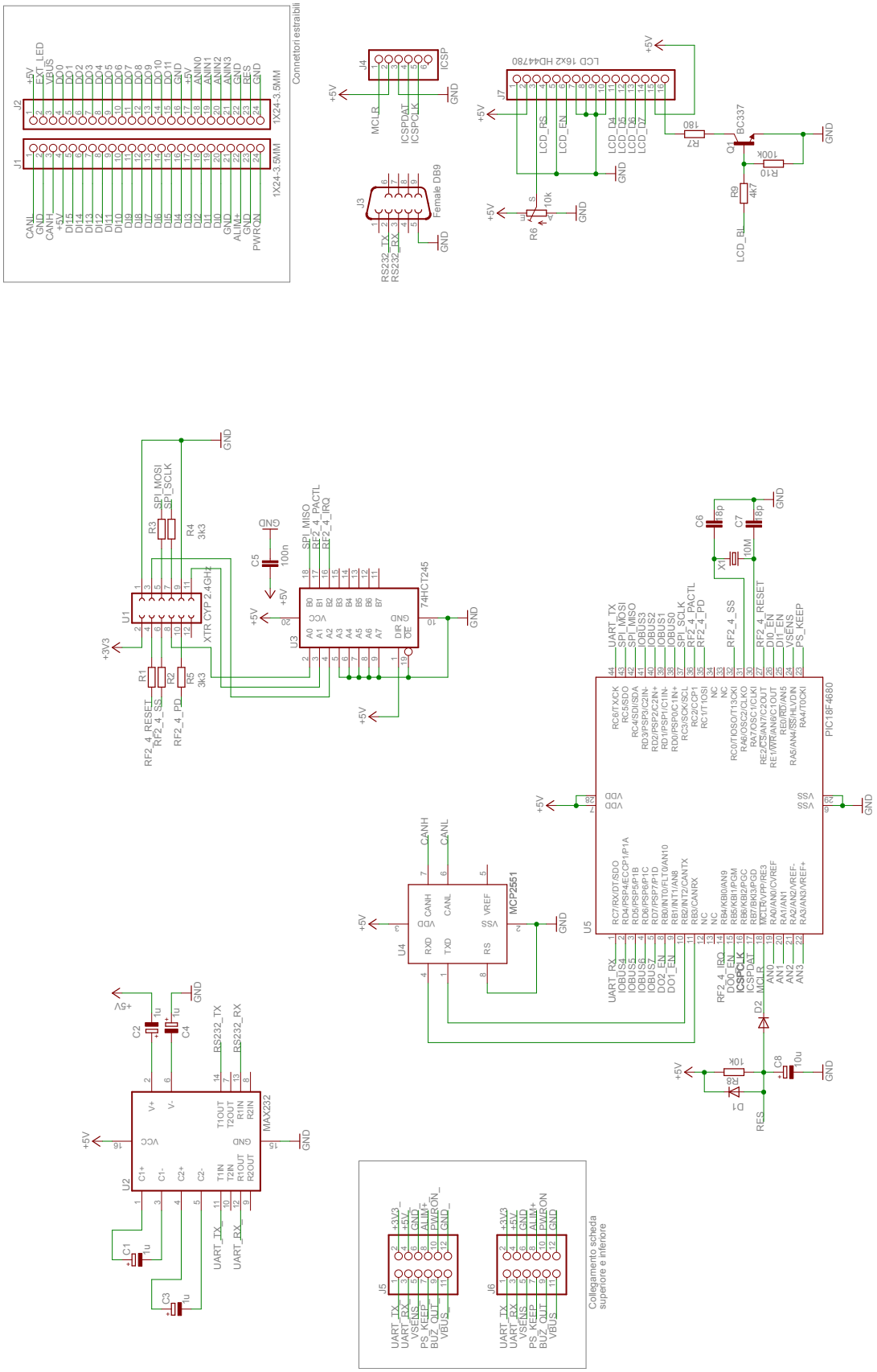


Figura A.2: Schema elettrico, microcontrollore e interfacce di comunicazione

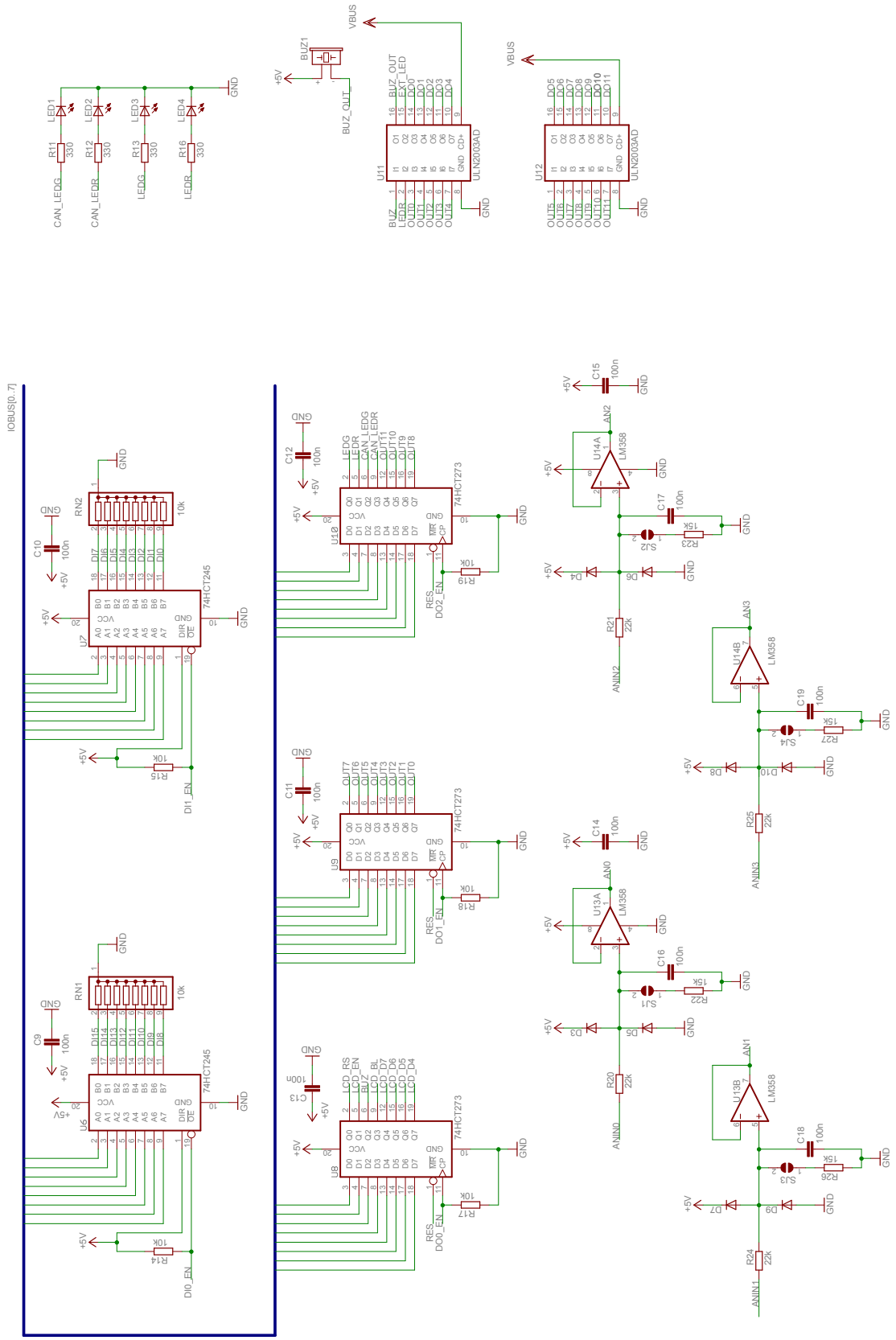


Figura A.3: Schema elettrico, input/output digitali ed analogici

Appendice B

Circuito stampato

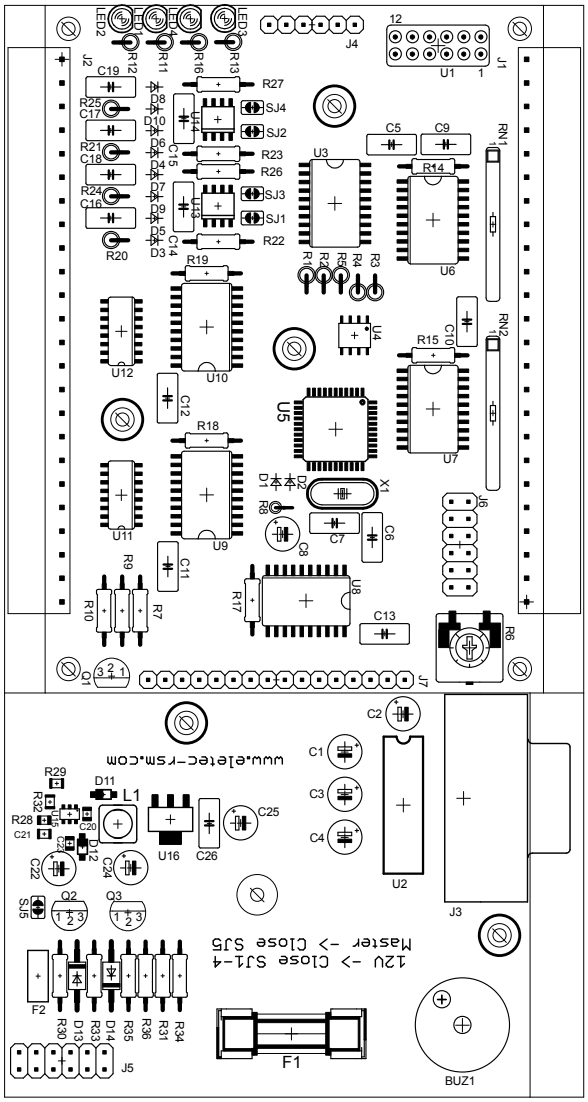


Figura B.1: Circuito stampato, silkscreen

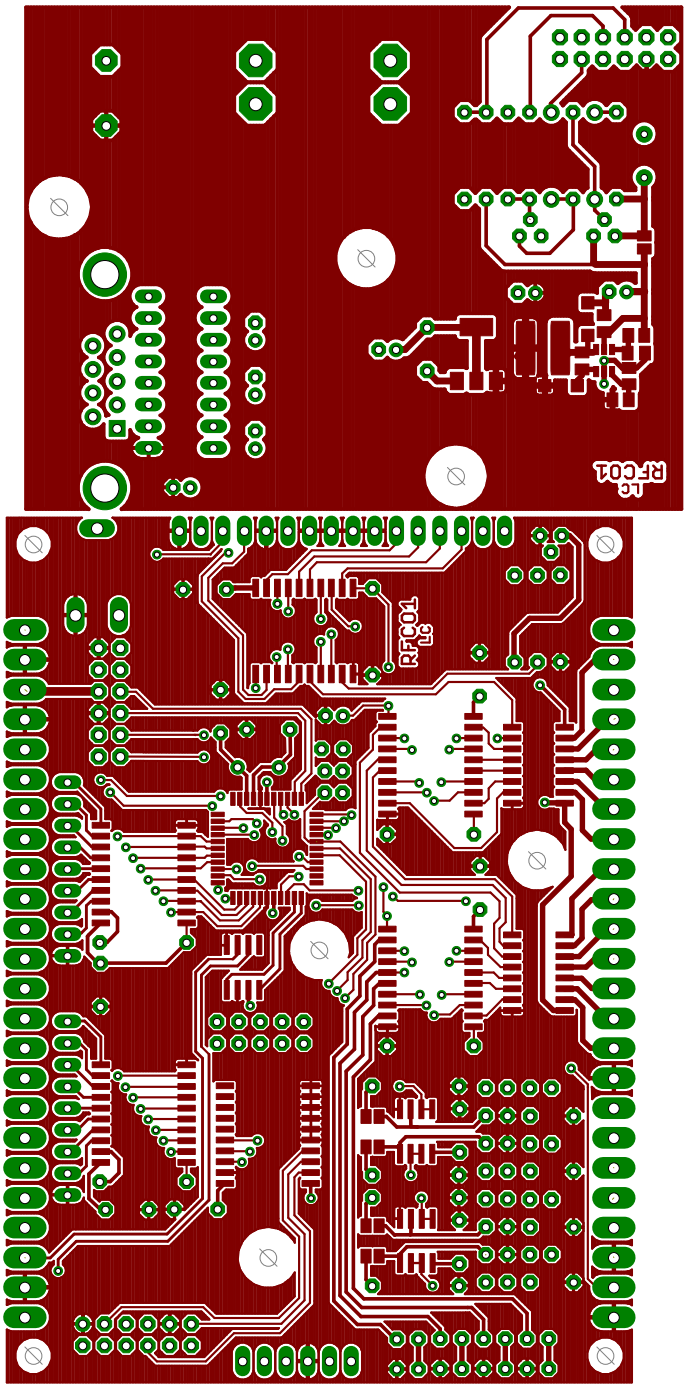


Figura B.2: Circuito stampato, layout lato componenti

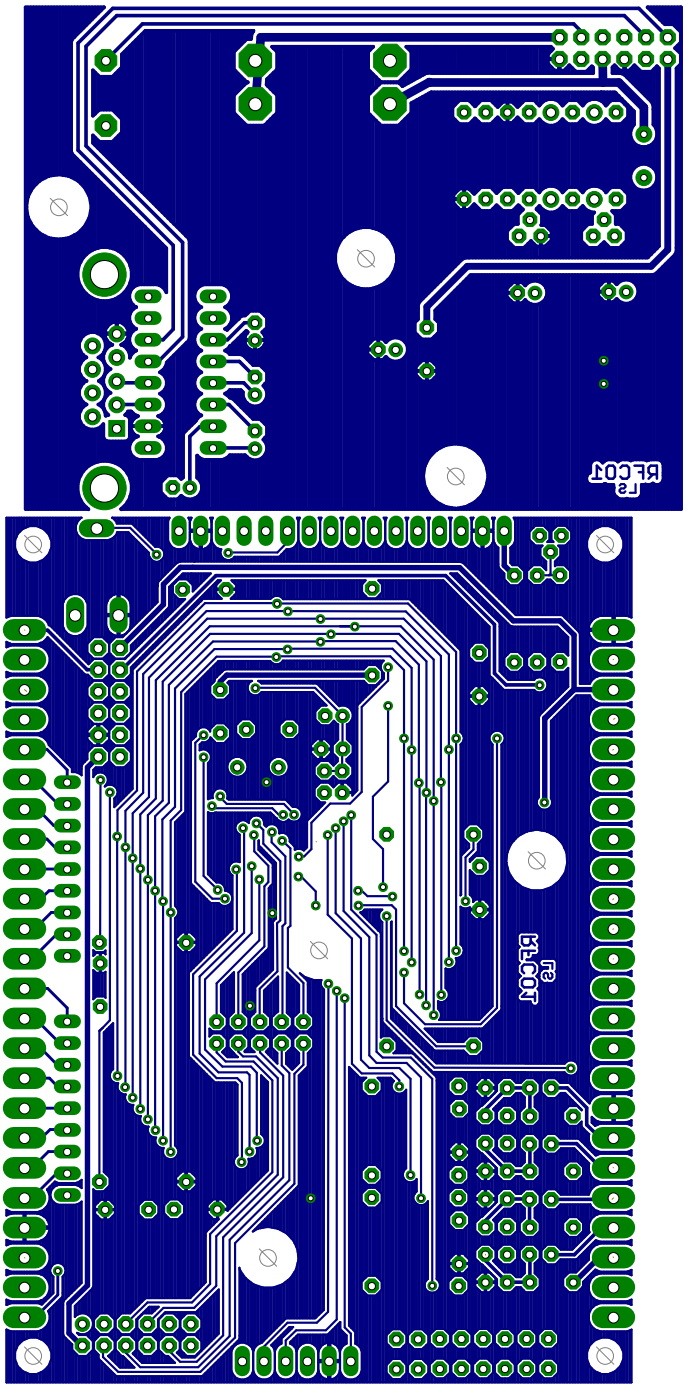


Figura B.3: Circuito stampato, layout lato saldature

Elenco delle figure

Elenco delle tabelle

Bibliografia

[1] Microchip Technology Inc., *PIC18F2585/2680/4585/4680 Data Sheet (DS69325B)*, 2004

[2] Philips Semiconductors, *74HC/HCT245 - Octal bus transceiver; 3-state*, 1993

[3] Philips Semiconductors, *74HC/HCT273 - Octal D-type flip-flop with reset; positive-edge trigger*, 1993

[4] Texas Instruments Incorporated, *ULN2002A, ULN2003A, ULN2003AI, ULN2004A ULQ2003A, ULQ2004A - High-voltage, high-current darlington transistor arrays (SLRS027K)*, 2011

[5] Linear Technology Corporation, *LT1933 - 600mA, 500kHz Step-Down Switching Regulator in SOT-23 and DFN Packages (1933fe)*

[6] National Semiconductor Corporation, *LM1117/LM1117I - 800mA Low-Dropout Linear Regulator (DS100919)*, 2004

[7] Microchip Technology Inc., *MCP2551 - High-Speed CAN Transceiver (DS21667D)*, 2003

[8] European Conference of Postal and Telecommunications Administrations - CEPT, *ERC Recommendation 70-03 - Relating to the use of Short Range Devices (SRD)*, 2012

[9] AUREL S.p.A *XTR VF 2.4 PA-LNA - Very Fast Low Power Multichannel Transceiver equipped with power amplifier, rev. B*, 2011

[10] Cypress Semiconductor Corporation, *CYRF6936 - WirelessUSB^{TM}LP 2.4 GHz Radio SoC (38-16015 Rev. J)*, 2011

[11] Hitachi, *HD44780U (LCD-II) Dot Matrix Liquid Crystal Display Controller/Driver (ADE-207-272(Z))*, 1999

[12] International Organization for Standardization, *ISO 11898-1:2003, Road vehicles - Controller area network (CAN) - Part 1: Data link layer and physical signalling*, 2003

[13] International Organization for Standardization, *ISO 11898-2:2003, Road vehicles - Controller area network (CAN) - Part 2: High-speed medium access unit*, 2003

[14] CAN in Automation, *CiA 301 V4.2.0 - CANopen® application layer and communication profile*, 2011

[15] CAN in Automation, *CiA 401 V3.0.0 - CANopen® device profile for generic I/O modules*, 2008

[16] CAN in Automation, *CiA 306 V1.3.0 - CANopen® electronic data sheet specification*, 2005

[17] CAN in Automation, *CiA 303-1 V1.7.0 - CANopen® recommendation - Part 1: Cabling and connector pin assignment*, 2009

[18] CAN in Automation, *CiA 303-3 V1.3.0 - CANopen® recommendation - Part 3: Indicator specification*, 2006

[19] *Cabur S.R.L*, http://www.cabur.it

[20] Ross M. Fosler, Microchip Technology Inc., *AN945 - A CANopen Stack for PIC18 ECAN^{TM}Microcontrollers (DS00945A)*, 2004