Corso di Laurea Magistrale in Ingegneria Meccanica

Sistemi Elettronici per la Meccatronica - Rita Asquini

“Realizzazione di un sistema di comunicazione radio bidirezionale, basato sull’utilizzo dell modulo radio nRF24L01+, per il controllo ed il monitoraggio in remoto di un autoveicolo"

Andrea Patrizi: 1742937

ABSTRACT:

Lo scopo di quest'elaborato è descrivere, nei suoi elementi principali, la realizzazione di un sistema di tipo master-slave per il controllo in radiofrequenza di un sistema meccatronico.
L'intero sistema è basato sull'utilizzo di due moduli radio di tipo nRF24L01+, equipaggiati con PA/LNA ("Power amplifier" e "Low noise amplifier", rispettivamente). Il sistema master, che può essere pensato come il vero e proprio controller del sistema, comunica ed interagisce con lo slave; lo slave può a sua volta essere pensato solidale ad un autoveicolo o ad un qualsiasi sistema meccatronico che richieda un controllo e/o monitoraggio in remoto. Il master e lo slave utilizzano, quindi, la capacità di comunicazione bidirezionale del chip nRF24L01+ per scambiarsi informazioni e segnali di controllo.
Si partirà dall'analisi dei componenti che costituiscono il sistema meccatronico nella sua interezza, con particolare attenzione dedicata alla descrizione del funzionamento ed alla discussione delle caratteristiche del modulo radio; ques'ultimo rappresenta, infatti, un componente elettronico che richiede una gestione particolarmente delicata e complessa, sia dal punto di vista hardware, che dal punto di vista software.
Si passerà successivamente alla presentazione della configurazione hardware del sistema, mostrata attraverso gli schemi di montaggio e circuitali dello slave e del master. Dopo aver discusso l'implementazione hardware del sistema, si effetteuerà un'analisi della parte software del sistema, discutendo con particulare attenzione la libreria NRF24, che si occupa della gestione dei moduli radio. Sarà riportata anche una breve descrizione delle ulteriori librerie di cui fa uso il programma. Verranno inoltre discusse tutte le funzioni definite all'interno dello script di gestione del sistema, come anche la struttura del preambolo, del setup e del loop del programma. Verranno infine riportati un diagramma di flusso esplicativo del funzionamento dello script e lo script stesso.

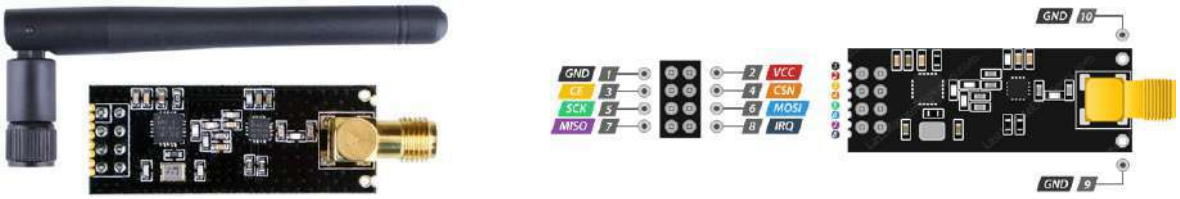
INDICE

LISTA DELLE IMMAGINI

1. Analisi componenti del sistema

Questa sezione è dedicata all'analisi dei singoli componenti che compongono il sistema nella sua interezza. Per qualsiasi informazione aggiuntiva circa i componenti è possibile far riferimento ai datasheet allegati al report, presenti anche sui siti dei produttori. Come già scritto all'interno dell'abstract, particolare attenzione è dedicata alla descrizione dei moduli radio.

1.1. Modulo radio



(a) Foto dell'intero modulo radio

(b) Pinout

Immagine 1.1: Modulo radio

Il modulo radio scelto per l'implementazione del sistema master-slave rappresenta, secondo l'opinione dell'autore di questa tesina, una delle alternative più efficaci per la comunicazione wireless a media e lunga distanza. Rispetto ad altri moduli di comunicazione wireless, il chip nRF24L01+ lavora a frequenze più alte; ciò consente, in linea di principio, una comunicazione a distanze maggiori rispetto a tecnologie alternative (Bluetooth, Wifi, etc...). In particolare, il modulo ricetrasmittente nRF24L01+ è progettato per operare alla frequenza ISM ("industrial, scientific, medical") di $2.4GHz$.
Il modulo può operare ad una tensione di alimentazione variabile nell'intervallo $1.9V - 3.6V$. Nonostante la ridotta tensione di alimentazione, i pin logici del modulo tollerano anche segnali di $5V$, il che rende il modulo adatto anche a microcontrollori con tensioni di lavoro maggiori.
La figura 1.1 riporta una foto del modulo ed una rappresentazione che ne mostra anche il pinout: i due pin di alimentazione, cioè Vcc e GND, ed i pin logici. Tra i pin logici, due sono possono essere collegati a due qualsiasi pin digitali del microcontrollore; essi sono CE (attivo alto, "chip enable"), che attiva la modalità RX o TX, e CSN (attivo basso, normalmente tenuto basso; "chip select not" o "SPI chip select"). Quando CSN effettua la transizione a basso, l'nRF24L01+ inizia l'ascolto sulla sua porta SPI, per predisporsi alla ricezione e processazione di eventuali dati. Il pin SCK ("serial clock") accetta un segnale di clock fornito dal bus SPI. Il pin MOSI ("master out, slave in") rappresenta l'input al chip, mentre il pin MISO ("master in, slave out") conduce l'output. Infine, l'IRQ è un pin di interrupt che può allertare il microcontrollore della disponibilità di nuovi dati da processare.
Guardando la figura 1.1 è possibile indentificare due IC montati sulla scheda di silicio del modulo: a sinistra è presente l'nRF24L01+, mentre il chip visibile sulla destra è l'RFX2401, che implementa funzionalità di amplificazione del segnale di output (PA, "power amplifier") ed il filtraggio del rumore in lettura (LN, "low noise amplifier"). Quest'ultimo chip permette di estendere considerevolmente il range di trasmissione (ed anche ricezione) fino ad $800m$; in realtà, con particolari accorgimenti, è possibile raggiungere distanze di trasmissione anche di decine di chilometri.

1.1.1. Il chip nRF24L01+

In questa sezione si esplorano ed approfondiscono alcune caratteristiche peculiari del chip utilizzato per la comunicazione radio. Nelle figure seguenti sono mostrati un diagramma a blocchi del modulo ed il relativo pinout. Sono ben distinguibili l'interfaccia SPI, il registro, i 6 registri FIFO (3 per ogni modalità), il modulatore, il demodulatore ed il PA/LNA.

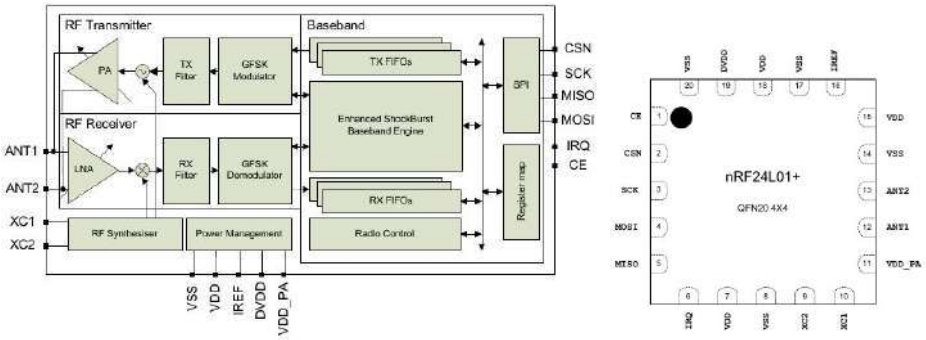


Immagine 1.2: Diagramma a blocchi e pinout del chip

Come visibile nelle figure 1.3 e 1.4, il modulo trasmette su specifiche frequenze, chiamate "canali", nel range $2400 - 2525MHz$ ed ogni canale occupa circa $1MHz$. Ciò fornisce un totale di 125 canali indipendenti su cui è possibile operare senza mutue interferenze.
In realtà, come visibile negli estratti dei datasheet, la spaziatura dei canali varia a seconda della velocità di trasmissione dei dati. In particolare, il modulo può operare a tre velocità: $250kbps$, $1Mbps$, , $2Mbps$, settabili tramite l'interaccia SPI. Durante il setup della comunicazione bisogna prendere attentamente in considerazione il datarate a cui operare: come visibile in figura 1.3, la sensibilà di ricezione dipende dalla velocità di trasmissione ed, in particolare, diminuisce all'aumentare della seconda. Di conseguenza, se la mole di dati non è eccessivamente grande, è preferibile lavorare a $250kbps$.
Per quanto riguarda il consumo di corrente, il modulo richiede un massimo di $13.5mA$ durante l'operazione di ricezione e di $11.3mA$ durante la trasmissione; durante lo standby, consuma circa $26\mu A$.
L'nrf24L01+ è munito di una modalità di comunicazione multipla chiamata "Multiciever" (figura 1.5). È un'abbreviazione di "Multiple Transitters, Single Receiver": il master può ricevere contemporaneamente da 6 nodi trasmettitore attraverso le 6 data pipe attivabili alla ricezione. Una data pipe è di fatto un canale logico all'interno della radiofrequenza fisica di comunicazione.

Features of the nRF24L01+ include:

- Radio
  - Worldwide 2.4GHz ISM band operation
  - 126 RF channels
  - Common RX and TX interface
  - GFSK modulation
  - 250kbps, 1 and 2Mbps air data rate
  - 1MHz non-overlapping channel spacing at 1Mbps
  - 2MHz non-overlapping channel spacing at 2Mbps
- Transmitter
  - Programmable output power: 0, -6, -12 or -18dBm
  - 11.3mA at 0dBm output power
- Receiver
  - Fast AGC for improved dynamic range
  - Integrated channel filters
  - 13.5mA at 2Mbps
  - -82dBm sensitivity at 2Mbps
  - -85dBm sensitivity at 1Mbps
  - -94dBm sensitivity at 250kbps
- RF Synthesizer
  - Fully integrated synthesizer
  - No external loop filer, VCO varactor diode or resonator
  - Accepts low cost ±60ppm 16MHz crystal
- Enhanced ShockBurst™
  - 1 to 32 bytes dynamic payload length
  - Automatic packet handling
  - Auto packet transaction handling
  - 6 data pipe MultiCeiver™ for 1:6 star networks
- Power Management
  - Integrated voltage regulator
  - 1.9 to 3.6V supply range
  - Idle modes with fast start-up times for advanced power management
  - 26µA Standby-I mode, 900nA power down mode
  - Max 1.5ms start-up from power down mode
  - Max 130us start-up from standby-I mode
- Host Interface
  - 4-pin hardware SPI
  - Max 10Mbps
  - 3 separate 32 bytes TX and RX FIFOs
  - 5V tolerant inputs
- Compact 20-pin 4x4mm QFN package

Immagine 1.3: Principali caratteristiche

Symbol	Parameter (condition)	Notes	Min.	Typ.	Max.	Units
f_{OP}	Operating frequency	a	2400		2525	MHz
PLL_{res}	PLL Programming resolution			1		MHz
f_{XTAL}	Crystal frequency			16		MHz
Δf_{250}	Frequency deviation @ 250kbps			±160		kHz
Δf_{1M}	Frequency deviation @ 1Mbps			±160		kHz
Δf_{2M}	Frequency deviation @ 2Mbps			±320		kHz
R_{GFSK}	Air Data rate	b	250		2000	kbps
$F_{CHANNEL\ 1M}$	Non-overlapping channel spacing @ 250kbps/ 1Mbps	c		1		MHz
$F_{CHANNEL\ 2M}$	Non-overlapping channel spacing @ 2Mbps	c		2		MHz

a. Regulatory standards determine the band range you can use.
b. Data rate in each burst on-air
c. The minimum channel spacing is 1MHz

Immagine 1.4: Ulteriori informazioni

MultiCeiver™ is a feature used in RX mode that contains a set of six parallel data pipes with unique addresses. A data pipe is a logical channel in the physical RF channel. Each data pipe has its own physical address (data pipe address) decoding in the nRF24L01+.

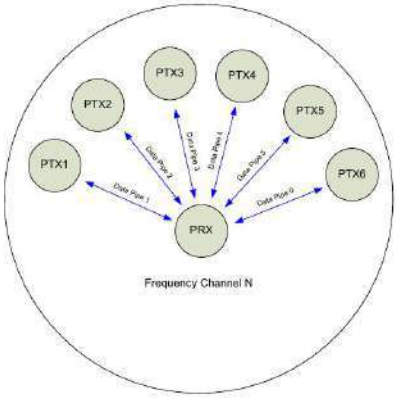


Immagine 1.5: Modalità "Multiciever"

Ad ogni nodo e pipe di comunicazione del sistema viene associato un indirizzo univoco, in modo che il master possa distinguere i messaggi provenienti dai diversi slave . La figura 1.6 mostra la struttura degli indirizzi utilizzati per la comunicazione. Essi sono specificati da una parola di 5 byte (dimensione di default), sulla quale sono però imposti alcuni vincoli. Come visibile in figura, la data pipe 0 ha un indirizzo unico, mentre le pipe dalla 1 alla 6 devono condividere i 4 MSbytes (devono invece possedere diverso LSbyte).

Each pipe can have up to a 5 byte configurable address. Data pipe 0 has a unique 5 byte address. Data pipes 1-5 share the four most significant address bytes. The LSByte must be unique for all six pipes. Figure 11 is an example of how data pipes 0-5 are addressed.

	Byte 4	Byte 3	Byte 2	Byte 1	Byte 0
Data pipe 0 (RX_ADDR_P0)	0xE7	0xD3	0xF0	0x35	0x77
Data pipe 1 (RX_ADDR_P1)	0xC2	0xC2	0xC2	0xC2	**0xC2**
Data pipe 2 (RX_ADDR_P2)	0xC2	0xC2	0xC2	0xC2	**0xC3**
Data pipe 3 (RX_ADDR_P3)	0xC2	0xC2	0xC2	0xC2	**0xC4**
Data pipe 4 (RX_ADDR_P4)	0xC2	0xC2	0xC2	0xC2	**0xC5**
Data pipe 5 (RX_ADDR_P5)	0xC2	0xC2	0xC2	0xC2	**0xC6**

Immagine 1.6: Struttura indirizzi delle DATA PIPES (0-5)

Un'ulteriore aspetto fondamentale da considerare è la struttura dei pacchetti dati che vengono scambiati tra i moduli radio.
In particolare, l'nRF24L01+ implementa un protocollo di comunicazione chiamato Enhanced ShockBurst, il quale prevede la struttura mostrata in figura 1.6. Il pacchetto contiene un preambolo, seguito dall'indirizzo; successivamente è inserito una parola di controllo (9 byte), che contiene informazioni circa la lunghezza del payload, un identificativo univoco del pacchetto ed un byte che indica l'eventuale richiesta del mittente di inviare un'acknowledgement (una parola di conferma di avvenuta ricezione). Dopo il Packet control viene inserito il payload vero e proprio ed infine è presente il CRC ("cyclic redundancy check").
Questo nuovo tipo di struttura sostituisce il vecchio Shockburst ed introduce diverse migliorie:

- La possibilità di utilizzare payload di lunghezza variabile, evitando di doverla sovradimensionare.
- Fornisce ad ogni pacchetto dato inviato un codice identificativo univoco, il che consente alla ricevente di capire se si tratta di un nuovo messaggio o di una ritrasmissione.
- Ogni trasmissione può richiedere che venga inviato indietro un acknowledgment in risposta all'avvenuta ricezione di un messaggio.

Con questo tipo di setup, possono verificarsi tre diversi casi durante la comunicazione:

1. La trasmittente invia un segnale verso la ricevente e rimane in attesa, per un determinato intervallo di tempo, dell'ack da parte della ricevente. Quando l'ack è ricevuta, la trasmittente genera un interrupt sul pin IRQ.
2. Se la trasmittente non riceve l'ack all'interno dell'intervallo prefissato, tenta il reinvio (ritardi e reinvii possono essere impostate via SPI). Se l'ack è ricevuta, la trasmittente genera un interrupt.
3. Nel caso la ricevente riceva il pacchetto e per qualche motivo l'ack vada perso, la trasmittente tenterà la ritrasmissione. Quando la ricevente riceverà nuovamente il pacchetto, potrà ignorarlo grazie al codice identificativo univoco.

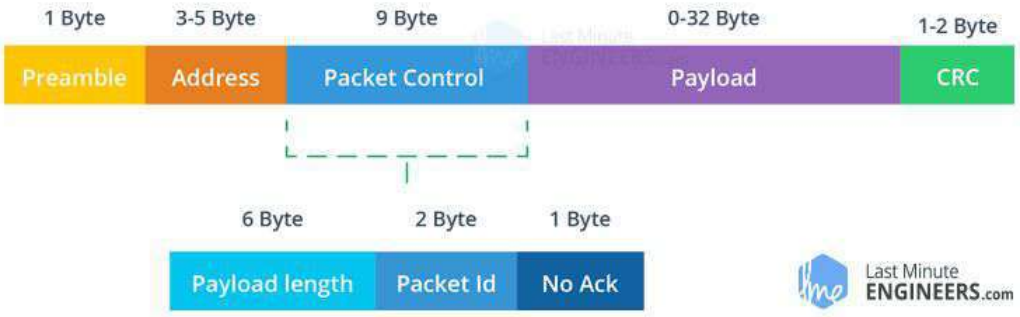


Immagine 1.7: Struttura del payload Enhanced Shockburst

Nell'immagine seguente, sono mostrati i 6 registri FIFO, di cui 3 per la modalità RX e 3 per la modalità TX, ed i relativi flussi di dati dati provenienti e verso l'interfaccia SPI. Come visibile, sono presenti due controller dedicati comandati via SPI ed ogni singolo registro può contenere fino a 32 byte di dati.

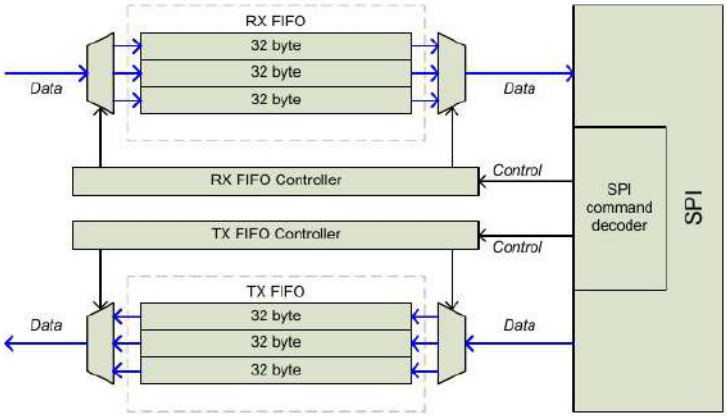


Immagine 1.8: Diagramma a blocchi FIFO

Infine, di seguito (figura 1.9) è riportato un estratto del datasheet che descrive sinteticamente la temporizzazione delle operazioni di lettura e scrittura sulla porta SPI.

Abbreviation	Description
Cn	SPI command bit
Sn	STATUS register bit
Dn	Data Bit (**Note:** LSByte to MSByte, MSBit in each byte first)

Table 20. Abbreviations used in Figure 23. to Figure 25.

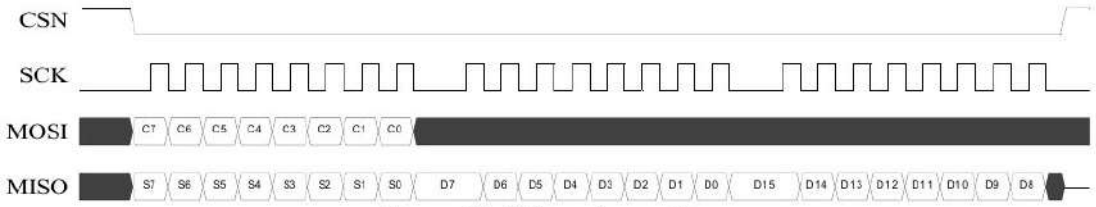


Figure 23. SPI read operation

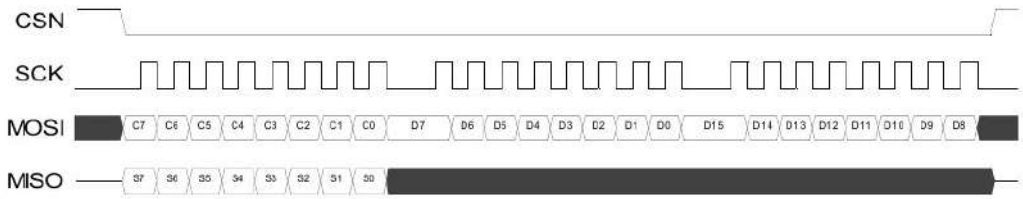


Figure 24. SPI write operation

Immagine 1.9: Temporizzazione operazioni di lettura e scrittura SPI

1.1.2. PA/LNA (power amplifier/low noise amplifier): RFX2401C

Come accennato nella sezione precedente, il modulo fa uso di un IC di PA/LNA, accoppiato all'nRF24L01+. Ciò permette di estendere notevolmente la portata della radiotrasmittente. La figura 1.10 riporta la pagina iniziale del datasheet. In sostanza il chip realizza, attraverso uno switch (duplexer), l'amplificazione del segnale di uscita ed l'amplificazione a basso rumore del segnale in ingresso.

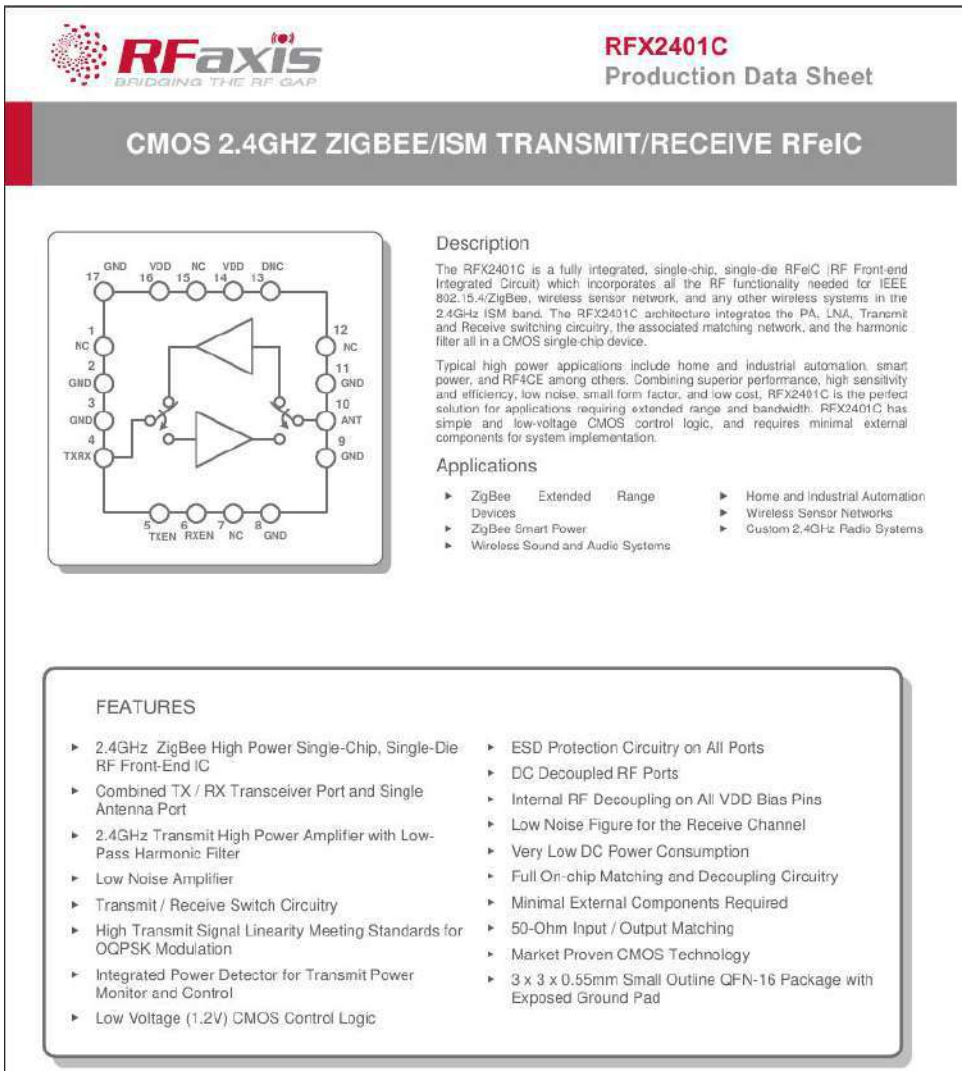


Immagine 1.10: Power amplifier e Low noise amplifier: l'RFX24101C

1.2. Periferiche hardware

1.2.1. Componenti elettromeccanici: Servomotore, motore BLDC, ESC

Immagine 1.11: Il motore BLDC utilizzato (sinistra), l'ESC commerciale (centro) ed un servomotore (destra)

Le figure di sopra mostrano i principali componenti elettromeccanici utilizzati dallo slave. Il motore BLDC è di tipo SPM (surface permanent magnet), provvisto di sensorizzazione opzionale. La struttura del rotore è ben visibile in figura; questo tipo di rotore, date le sue caratteristiche geometriche, produce una emf sulle fasi dello statore di tipo sinusoidale (ciò è stato verificato tramite l'utilizzo di un oscilloscopio digitale di tipo DSO138; si faccia riferimento alla sezione "Strumenti utilizzati"). Sul retro dello statore è presente un encoder, il quale fornisce all'ESC la posizione del campo di induzione magnetica del rotore. Tale informazione viene utilizzata per stimare le correnti di statore ed esercitare le opportune azioni di controllo. Si noti che, visto il carattere "hobbistico" dei componenti e la presenza del solo encoder, la strategia di controllo del motore non potrà essere di tipo vettoriale ("Vector Control").
L'ESC si occupa di pilotare le singole fasi del motore in base alla velocità settata dal segnale di comando ricevuto (di tipo PWM, si veda "Configurazione del sistema" per ulteriori informazioni). Presenta un totale di 5 cavi, di cui 2 di alimentazione (Vcc e GND) e 3 di pilotaggio del motore. L'esc presenta un ulteriore ingresso per l'eventuale feedback proveniente dall'encoder del motore.

Specifiche principali motore (vedasi la figura in basso per maggiori dettagli):

- 2700 KV (rpm/V)
- 130 A (massimo assorbimento di corrente)
- 2 coppie polari
- 2.4 kW

Specifiche principali ESC (vedasi la figura in basso per maggiori dettagli):

- 120 A (massima erogazione continua di corrente; leggermente sottodimensionata rispetto al motore, a sfavore di sicurezza)
- 760 A (massima corrente di picco)
- Supporta motori sensorizzati e non
- Alimentabile con lipo di tipo 2s o 3s (numero di celle)
- BEC (battery eliminator circuit) da 6V/2A. Utile nel caso si voglia alimentare una trasmittente utilizzando la stessa batteria dell' ESC.

Infine, lo slave utilizza un comune servomotore di tipo hobbystico, il quale funziona regolarmente a 5V, ma può essere anche alimentato a 6V (vedasi il BEC dell'ESC). Per maggiori informazioni sul suo controllo, si faccia riferimento alla sezione "Configurazione del sistema".

Parameter:

Product	Watts	Max voltage	Max Amps	Rotor Poles	IO	Resistance	Kv (RPM/Volt)	Max RPM	Length (mm)	Diameters (mm)	Weight (g)	Shaft (mm)	Length of extend shaft
4268/2Y	2400W	<18V	130A	4	3.2A	0.0053	2700	50000	68	Ø41/Ø42 with fans	320	Ø5	18mm
4268/4D	2400W	<22V	110A	4	3.0A	0.0080	2350	50000	68	Ø41/Ø42 with fans	320	Ø5	18mm
4268/5D	2400W	<25V	95A	4	2.5A	0.0090	2000	50000	68	Ø41/Ø42 with fans	320	Ø5	18mm
4268/3Y	2400W	<27V	82A	4	2.2A	0.0130	1850	50000	68	Ø41/Ø42 with fans	320	Ø5	18mm
4268/6D	2400W	<32V	75A	4	1.8A	0.0160	1550	50000	68	Ø41/Ø42 with fans	320	Ø5	18mm

4268

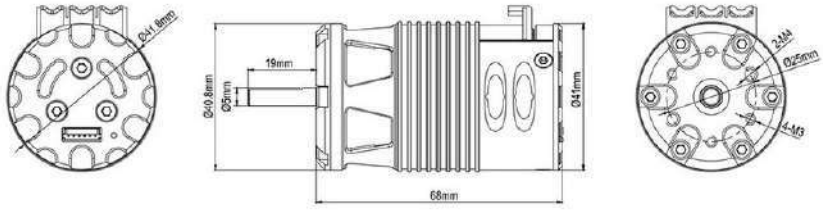
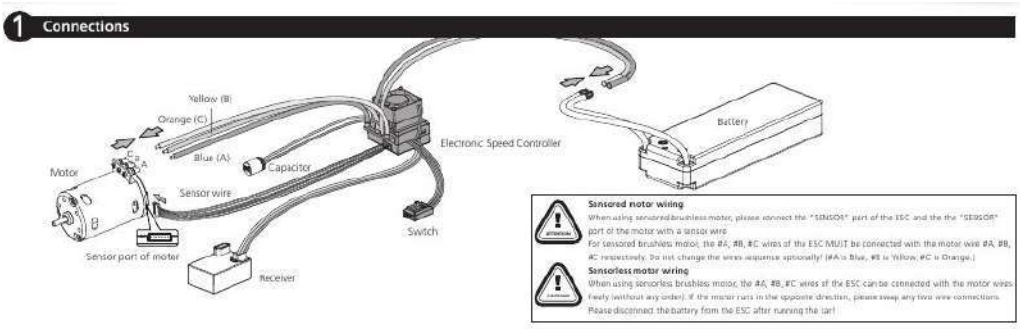


Immagine 1.12: Caratteristiche tecniche dell'ESC utilizzato



Specifications

Model		QUICRUN 10BL60 SENSORED	QUICRUN 10BL120 SENSORED
Cont. / Peak Current		60A / 380A	120A / 760A
Motor Type Supported		Sensored / Sensorless Brushless Motor	Sensored / Sensorless Brushless Motor
Car Applicable		1/10th Touring Car/Buggy/F1/Drift Car & 1/12th On-road	1/10th Touring Car/Buggy/F1/Drift Car, and etc.
Motor Limit (Note)	4-6 cells NiMH / 2S LiPo	>8.5T (1/10 On-road), >13.5T (1/10 Off-road)	>3.5T (1/10 On-road), >4.5T (1/10 Off-road)
	8-9 cells NiMH / 3S LiPo	>11.5T (1/10 On-road), >17.5T (1/10 Off-road)	>8.5T (1/10 On-road), >10.5T (1/10 Off-road)
Battery		4-9 cells NiMH, 2-3S LiPo	4-9 cells NiMH, 2-3S LiPo
BEC Output		6V@2A, Linear Mode	6V@3A, Linear Mode
Cooling Fan		6V (The cooling fan gets its power supply from the built-in BEC)	Powered by the battery
Dimensions / Weight		33 x 28 x 31.5mm (w/ Fan) / 59g (w/ Wires, w/o Fan)	43 x 36 x 33mm (w/ Fan) / 105g (w/ Wires & Fan)

Immagine 1.13: Caratteristiche tecniche del motore utilizzato

1.2.2. Ulteriori componenti

- LCD con espansore per bus I2C:
  Di seguito si riporta una foto dell'LCD utilizzato dal master per la visualizzazione dei dati di interesse. Il suddetto LCD è equipaggiato con un espansore I/O via I2C (mostrato in basso), il quale consente il pilotaggio del dispositivo utilizzando solamente 4 cavi (Vcc, Gnd, SDA, SCL). Di seguito è riportato anche un estratto del datasheet relativo all'espansore, da cui si evince come tale dispositivo effettui un'operazione di parallelizzazione della linea I2C (two-line bidirectional bus communication).

Immagine 1.14: LCD e relativo espansore I2C

Remote 8-bit I/O expander for I²C-bus PCF8574

1 FEATURES

- Operating supply voltage 2.5 to 6 V
- Low standby current consumption of 10 µA maximum
- I²C to parallel port expander
- Open-drain interrupt output
- 8-bit remote I/O port for the I²C-bus
- Compatible with most microcontrollers
- Latched outputs with high current drive capability for directly driving LEDs
- Address by 3 hardware address pins for use of up to 8 devices (up to 16 with PCF8574A)
- DIP16, or space-saving SO16 or SSOP20 packages.

2 GENERAL DESCRIPTION

The PCF8574 is a silicon CMOS circuit. It provides general purpose remote I/O expansion for most microcontroller families via the two-line bidirectional bus (I²C).

The device consists of an 8-bit quasi-bidirectional port and an I²C-bus interface. The PCF8574 has a low current consumption and includes latched outputs with high current drive capability for directly driving LEDs. It also possesses an interrupt line ($\overline{INT}$) which can be connected to the interrupt logic of the microcontroller. By sending an interrupt signal on this line, the remote I/O can inform the microcontroller if there is incoming data on its ports without having to communicate via the I²C-bus. This means that the PCF8574 can remain a simple slave device.

The PCF8574 and PCF8574A versions differ only in their slave address as shown in Fig.9.

Immagine 1.15: Estratto datasheet espansore per bus I2C

- n.2 adattatori/regolatori di tensione per i due moduli radio nRF24L01+:

Immagine 1.16: Il regolatore di tensione/adattatore
per il modulo radio (datasheet allegato al report)

Questo componente presenta un IC di tipo AMS1117 per la regolazione della tensione ed una serie di capacitori per la stabilizzazione dell'alimentazione. Ciò è fondamentale sia per garantire un funzionamento più regolare del modulo radio, sia perchè i pin del modulo radio tollerano tensioni di $5V$, ma il modulo dev'essere comunque alimentato a $3.3V$.

- Sensore di temperatura ed umidità digitale DHT11:

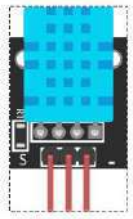


Immagine 1.17: DHT11

In figura sono mostrati i tre pin (Vcc, Ground ed il pin di Segnale). Il sensore presenta in output direttamente un segnale digitale calibrato. Si noti che, secondo il relativo datasheet, è richiesta la connessione di un resistore di pullup da $10K\Omega$ tra i pin di segnale e Vcc.

- n.2 batterie LiPo $2s$ da $6200\,mAh$: batterie litio-polimero a due celle ($2s$), con capacità di scarica pari a $50C$. Ogni cella ha un voltaggio nominale di $3.7V$, per un totale di $7.4V$.

Immagine 1.18: Batteria utilizzata

- n.2 led (rosso e blu):

Immagine 1.19: LED

- Joystick per il pilotaggio del servomotore e dell'ESC:

Immagine 1.20: Thumb joystick

- Cavi $AWG12$ e connettori Deans per il collegamento delle batterie ai microcontrollori ed all'ESC:

Immagine 1.21: Cavi utilizzati per realizzare le connessioni
non effettuabili tramite semplici "jumper wires"

- n.2 schede di prototipazione per la MEGA 2560, utilizzate per rendere più stabili ed ordinate le connessioni:

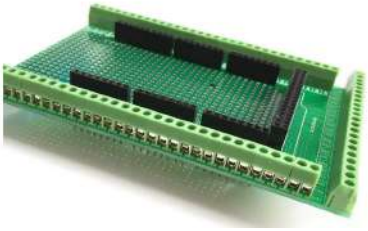


Immagine 1.22: schede di prototipazione per la MEGA 2560

- Cavi di tipo "jumper", utilizzati per realizzare i collegamenti diretti con i due microcontrollori:

Immagine 1.23: "jumper wires"

- n.6 resistori ad ossido metallico, di cui due da $1M\Omega$ e due da $100K\Omega$ sono utilizzati nel partitore per la misurazione della tensione di alimentazione delle batterie, mentre i restanti resistori sono da 220Ω e sono utilizzati per i due led:

Immagine 1.24: Resistori assiali ad ossido metallico
(con tolleranze tra lo 0.1% ed il 5%)

- Guaine termorestringenti, utilizzate per mettere in sicurezza le eventuali saldature effettuate sui cavi:

Immagine 1.25: Guaine termorestringenti

1.3. MCU: Arduino Mega 2560 rev3

L' Arduino Mega 2560 è una scheda a microcontrollore costruita intorno al chip ATmega2560 (datasheet allegato), programmabile attraverso l'IDE di Arduino. Possiede un totale di 54 pin digitali di input/output (di cui 15 utilizzabili in PWM), 16 pin analogici, 4 UART ("Universal Asynchronous Receiver-Transmitter") ed un oscillatore a 16MHz.
Di seguito sono riportate le principali specifiche tecniche della scheda ed il pinout della stessa. Particolarmente importante è l'immagine di destra del pinout, che contiene la posizione dei pin SPI (MISO. MOSI, SCK), fondamentali per utilizzare il modulo radio. Essi sono, rispettivamente, i pin 50, 51, 52. Un altro dettaglio fondamentale è che la scheda opera a $5V$; questo è fondamentale per evitare di danneggiare i moduli radio (che operano a $3.3V$). Si noti inoltre che la tensione di alimentazione raccomandata applicabile al pin V_{in} è nell'intervallo $7V - 12V$; le batterie LiPo selezionate sono quindi idonee all'alimentazione della scheda. Infine, la risoluzione standard dei pin analogici è di 10 bit (1024 possibili rivelli rilevabili); ques'informazione è necessaria per poter sapere l'accuratezza con la quale può operare il partitore di tensione utilizzato per la misurazione della tensione di alimentazione.

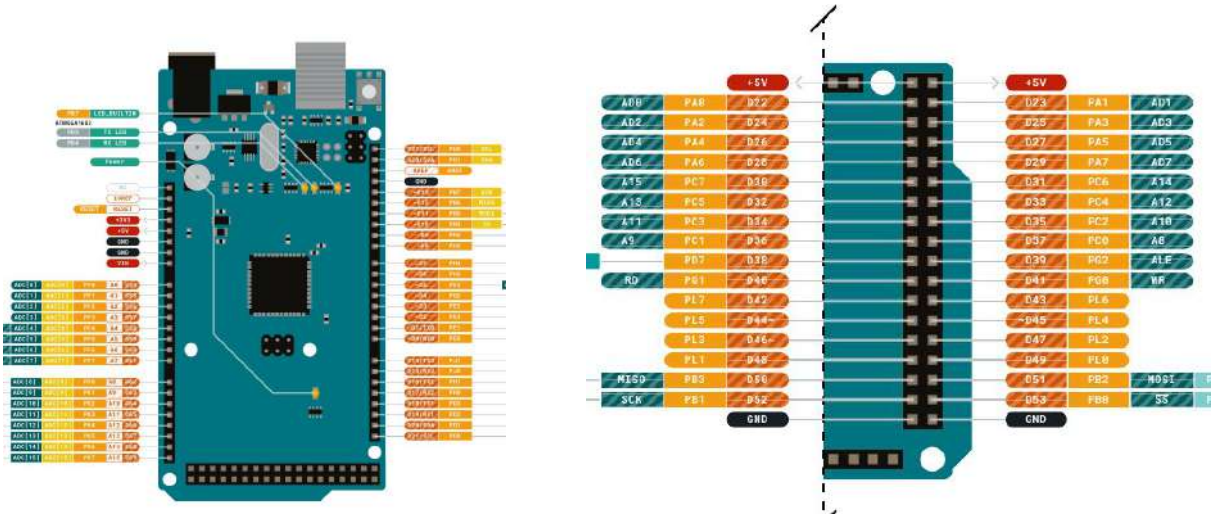


Immagine 1.26: Pinout Arduino MEGA 2560 Rev3
(per il pinout completo si faccia riferimento al datasheet)

Microcontroller	ATmega2560
Operating Voltage	5V
Input Voltage (recommended)	7-12V
Input Voltage (limit)	6-20V
Digital I/O Pins	54 (of which 15 provide PWM output)
Analog Input Pins	16
DC Current per I/O Pin	20 mA
DC Current for 3.3V Pin	50 mA
Flash Memory	256 KB of which 8 KB used by bootloader
SRAM	8 KB
EEPROM	4 KB
Clock Speed	16 MHz
LED_BUILTIN	13
Length	101.52 mm
Width	53.3 mm
Weight	37 g

Immagine 1.27: Principali specifiche Arduino MEGA 2560 Rev3

1.3.1. Strumenti utilizzati

- Saldatore a stagno, utilizzato per effettuare i collegamenti alle batterie, ai partitori di tensione e per sostituire le connessioni con i jumper, laddove la stabilità di connessione fosse cruciale:

Immagine 1.28: Saldatore a stagno

- Multimetro digitale, utilizzato per controllare le tensioni in punti chiave del sistema e per calibrare i partitori di tensione (che sono inevitabilmente soggetti ad errori dovuti a resistenze parassite, incertezze sui valori delle stesse e risoluzione limitata del microcontrollore).
- Oscilloscopio digitale DS138, utilizzato in una fase preliminare del progetto; in particolare si è rivelato essere fondamentale per verificare i segnali di pilotaggio dell'ESC e del servomotore, come anche per visualizzare le forme d'onda prodotte dall'ESC nel pilotaggio delle tre fasi del motore BLDC. Si noti che, data la non professionalità dell'oscilloscopio, si è potuta visualizzare solo la tensione tra due qualsiasi delle fasi del motore, mentre la singola tensione di fase resta non misurabile (connessione a stella delle fasi).
  Per informazioni circa le caratteristiche del suddetto DSO138, si faccia riferimento al relativo datasheet, allegato al report.

Immagine 1.29: Multimetro digitale (sinistra) ed oscilloscopio digitale DSO138

2. Configurazione del sistema

Questa sezione è dedicata alla discussione della configurazione e del montaggio del sistema. Viene presentato prima il master, con annessi schemi di montaggio e circuitali e, di seguito, lo slave.
Gli schemi riportati sono stati realizzati con l'utilizzo del software Fritzing. I componenti non presenti tra le parti standard sono stati reperiti online o implementati direttamente da zero.

2.1. Master

Il master è costituito dai seguenti componenti:

- Scheda Arduino MEGA 2560 rev3:
  —> è il cuore del sistema master e gestisce tutte le periferiche hardware associate.

- Batteria LiPo:
  —> è connessa direttamente tra il pin V_{in} della scheda e GND; fornisce la tensione di alimentazione della scheda (se la batteria è carica, erogherà circa $8.5V$; altrimenti, nello stato di "storage" erogherà circa $7.4V$).

- Partitore di tensione:
  —> Realizzato con due resistori da $100K\Omega$ e $1M\Omega$; è connesso direttamente alla batteria e presenta il pin di lettura connesso ad uno dei pin analogici della scheda. È necessario per adattare la tensione di alimentazione ai pin analogici della MEGA, che consentono una tensione massima di $5V$. Con questa configurazione del partitore è teoricamente possibile misurare una tensione di ingresso fino a $55V$.

- Joystic:
  —> Viene utilizzato nel processo di avvio del master (la pressione sul bottone inizializza la comunicazione radio) e, durante la comunicazione, per inviare i segnali di comando allo slave. Utilizza un totale di 5 pin: Vcc, GND, VER (asse verticale), HOR (asse orizzontale) e SEL (pulsante). VER ed HOR sono connessi a due pin analogici, mentre SEL ad un pin digitale.

- LCD con interfaccia I2C:
  —> L'espansore I2C è direttamente saldato ai pin dell'LCD; per il pilotaggio dell'LCD tramite l'espansore sono richiesti 4 pin: Vcc, GND, SDA (data line), SCL(clock line).

- Modulo radio nRF24L01+ e relativo adattatore/regolatore di tensione:
  —> L'adattatore è collegato al modulo radio tramite 8 pin: Vcc (3.3V!), GND, IRQ (interrupt), SCK (clock), MISO (master in, slave out), MOSI (master out, slave in), CE (chip enable), CSN (chip select not). A sua volta, l'adattatore è connesso alla scheda tramite altri 8 pin, con il medesimo ruolo; l'unica diferenza è che l'adattatore può essere alimentato a 5V. Si precisa che, mentre i pin CE, CSN, IRQ (non utilizzato) possono essere connessi ad un qualsiasi pin digitale della scheda, i pin MISO, MOSI ed SCK devono essere necessariamente connessi ai pin SPI dedicati (vedasi gli schemi riportati di sotto).

2.1.1. Schema di montaggio

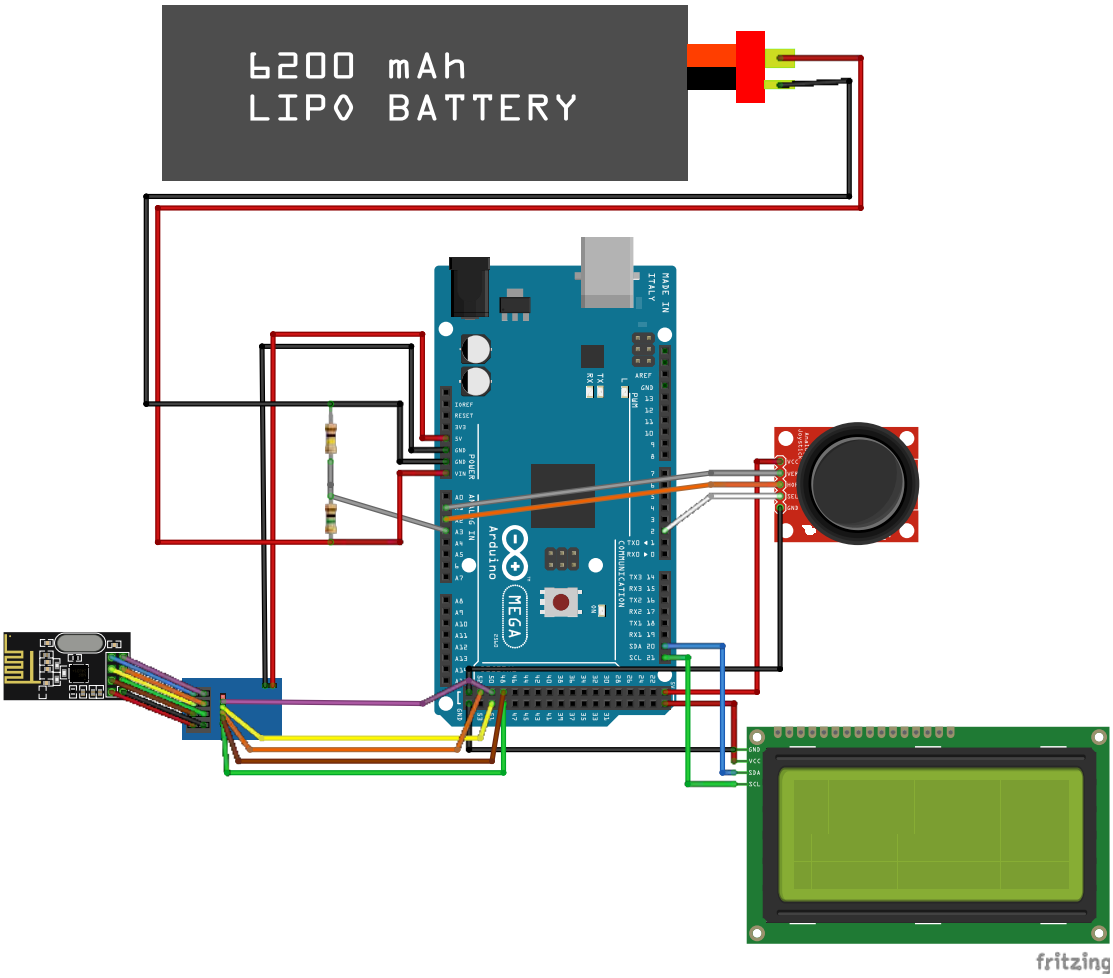


Immagine 2.1: Schema di montaggio del master

2.1.2. Schema circuitale

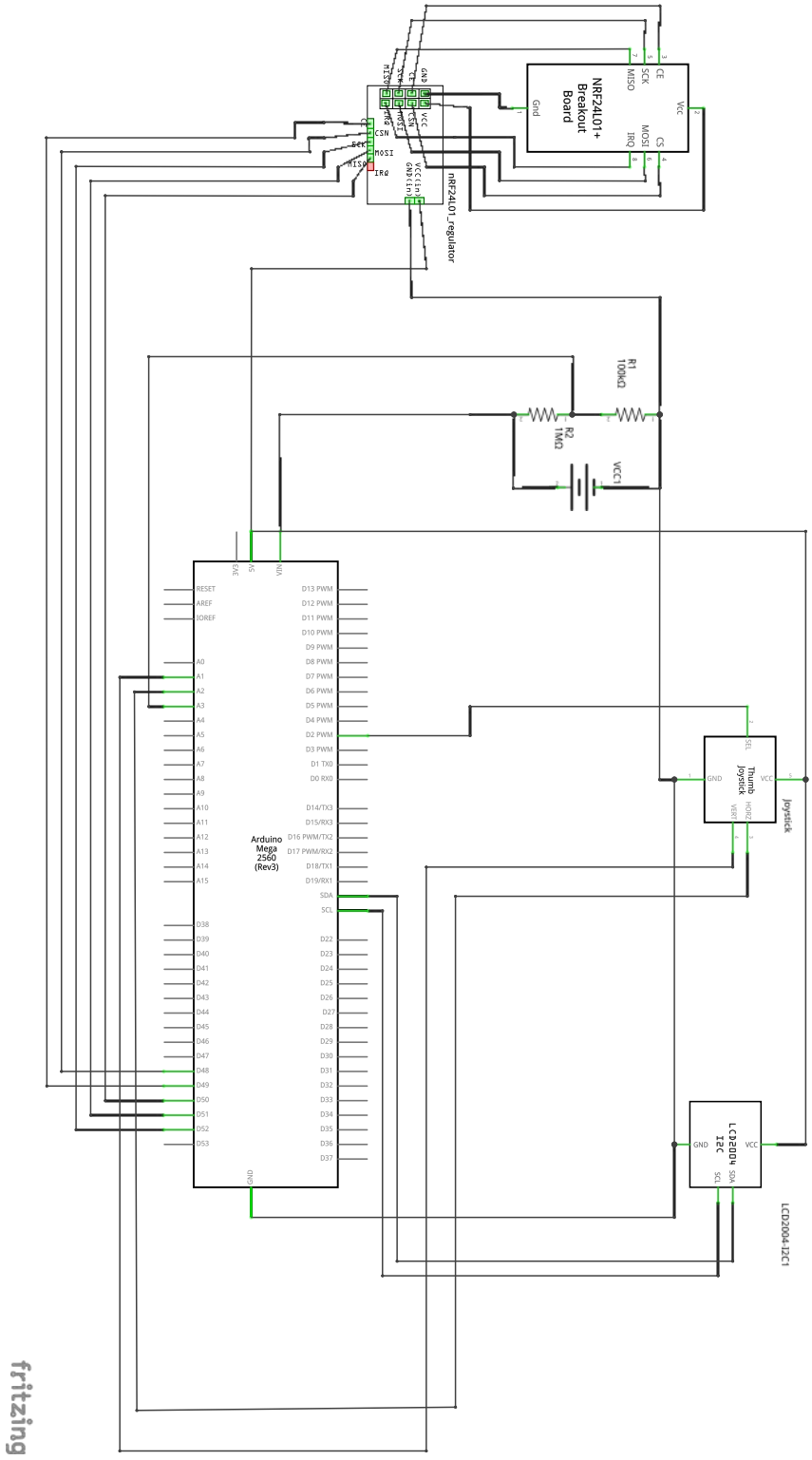


Immagine 2.2: Schema circuitale master

2.2. Slave

Lo slave è costituito dai seguenti componenti:

- Scheda Arduino MEGA 2560 rev3:
  —> è il cuore dello slave e gestisce tutte le periferiche hardware associate.

- Servomotore ed ESC:
  —> Utilizzano tre pin di input: Vcc, GND ed il pin di segnale (PWM). In entrambi i casi il segnale di comando è un onda quadra con periodo $20ms$ ed ampiezza variabile da $1ms$ a $2ms$ (mostrato sotto). È possibile comandare il servomotore direttamente dalla scheda, mentre il motore BLDC necessita di un controllore dedicato, cioè l'ESC, il quale si occupa di generare i segnali di comando per le tre fasi del motore. Si noti che l'ESC presenta anche due pin (Vcc e GND) che forniscono una tensione di $6V$, eventualmente utile per alimentare esternamente il servo, in caso di carichi da pilotare elevati (e.g. sterzo).

- Motore BLDC:
  —> Direttamente comandato dall'ESC, che si occupa di generare gli opportuni segnali di fase. È inoltre presente un feedback (opzionale) dal motore all'ESC: un encoder misura la posizione del rotore (e quindi del campo d'induzione magnetica associato) e consentono un buon controllo del motore anche a basse velocità.

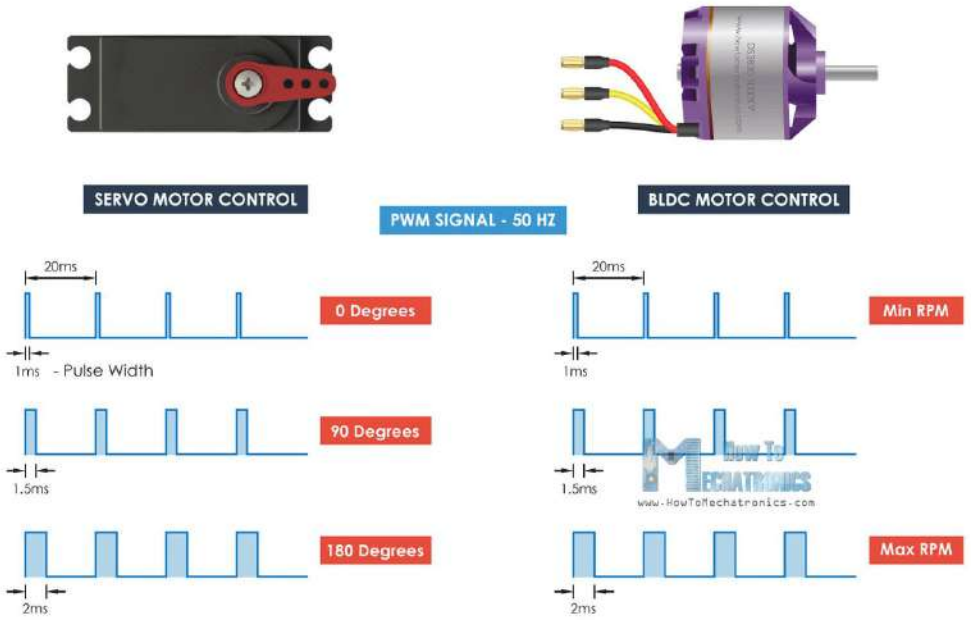


Immagine 2.3: Segnale di comando del servomotore e dell'ESC

- Batteria LiPo:
  —> è connessa direttamente tra il pin V_{in} della scheda e GND; fornisce la tensione di alimentazione della scheda (se la batteria è carica, erogherà circa $8.5V$).

- Partitore di tensione:
  —> Realizzato con due resistori da $100K\Omega$ e $1M\Omega$; è connesso direttamente alla batteria e presenta il pin di lettura connesso ad uno dei pin analogici della scheda. È necessario per adattare la tensione di alimentazione ai pin analogici della MEGA, che consentono una tensione massima di $5V$. Con questa configurazione del partitore è teoricamente possibile misurare una tensione di ingresso fino a $55V$.

- Una coppia di LED (blue e rosso):
  —> Connessi a GND tramite due resistori da 220Ω, verranno utilizzati dallo slave per mostrare lo stato della connessione radio.

- Modulo radio nRF24L01+ e relativo adattatore/regolatore di tensione:
  —> L'adattatore è collegato al modulo radio tramite 8 pin: Vcc (3.3V!), GND, IRQ (interrupt), SCK (clock), MISO (master in, slave out), MOSI (master out, slave in), CE (chip enable), CSN (chip select not). A sua volta, l'adattatore è connesso alla scheda tramite altri 8 pin, con il medesimo ruolo; l'unica diferenza è che l'adattatore può essere alimentato a 5V. Si precisa che, mentre i pin CE, CSN, IRQ (non utilizzato) possono essere connessi ad un qualsiasi pin digitale della scheda, i pin MISO, MOSI ed SCK devono essere necessariamente connessi ai pin SPI dedicati (vedasi gli schemi riportati di sotto).

- Sensore digitale di temperatura ed umidità relativa DHT11:
  —> Utilizza tre pin: Vcc, GND e SIGNAL. SIGNAL è connesso ad un pin digitale della scheda e presenta un resistore di pullup da $10k\Omega$ direttamente collegato al pin di segnale.

2.2.1. Schema di montaggio

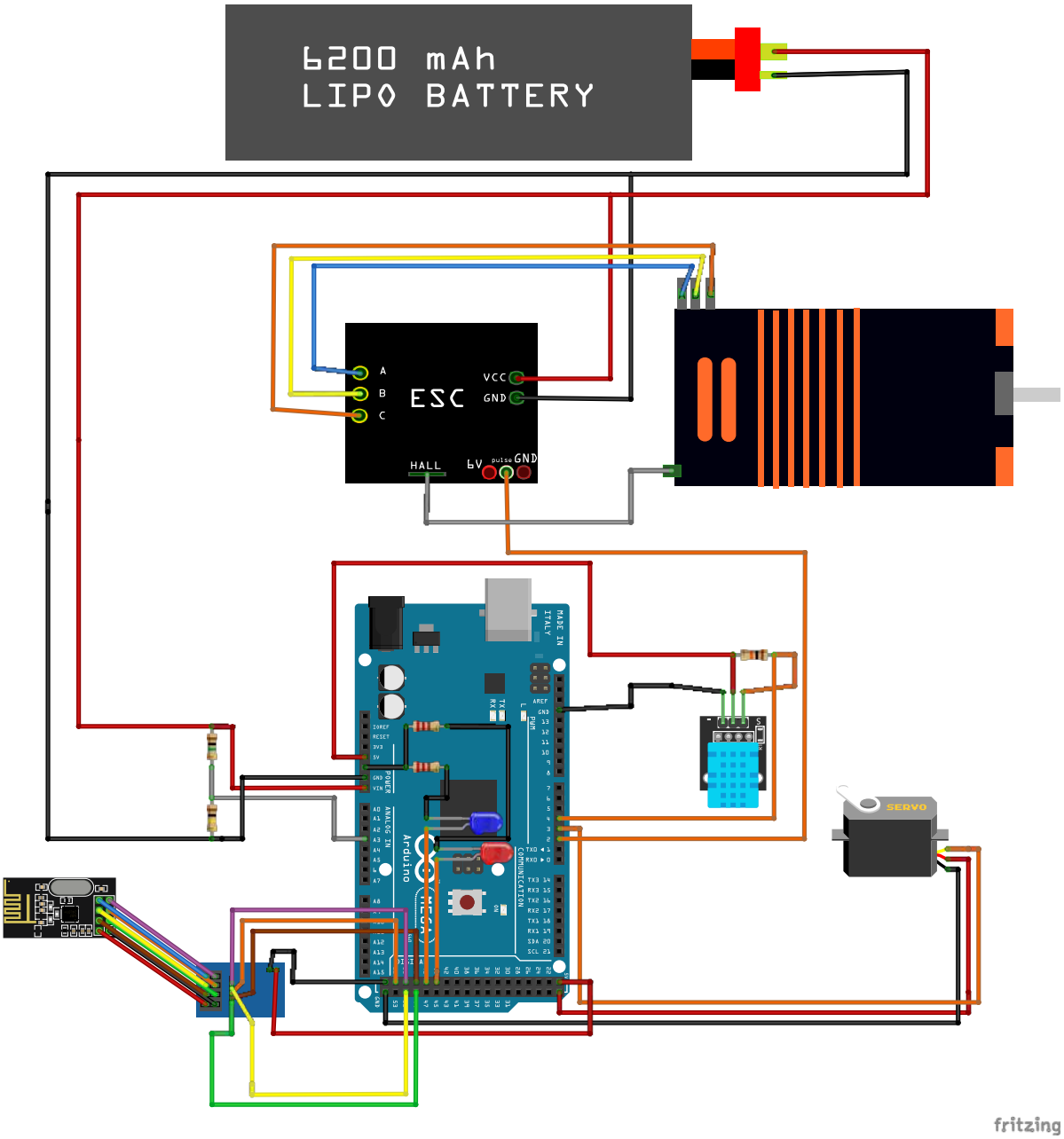


Immagine 2.4: Schema di montaggio slave

2.2.2. Schema circuitale

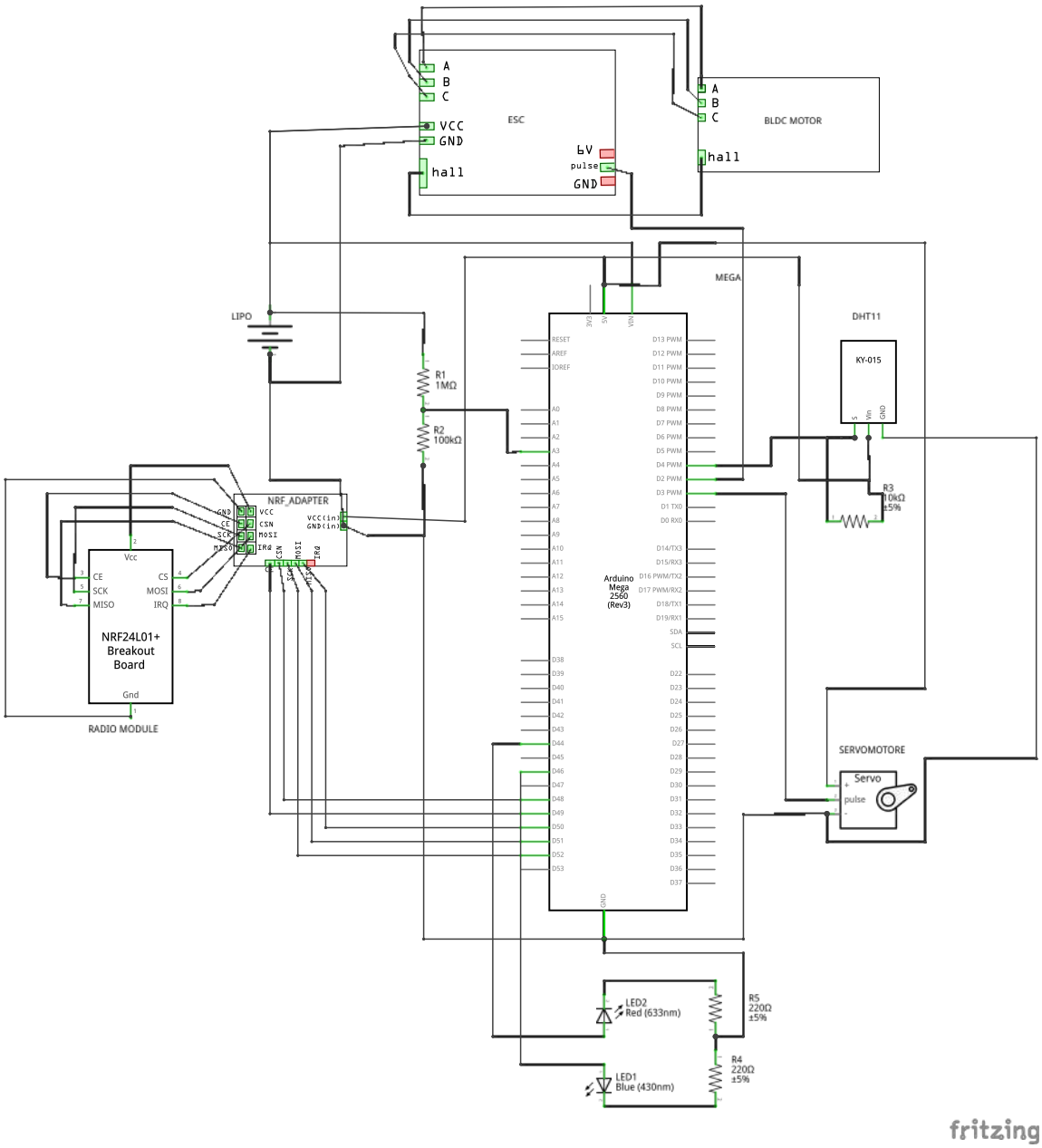


Immagine 2.5: Schema circuitale slave

2.3. Setup sperimentale

In questa sezione vengono riportate delle fotografie di come i due sistemi si presentano una volta assemblati secondo quanto specificato nelle sezioni precedenti.

- Setup del Master:

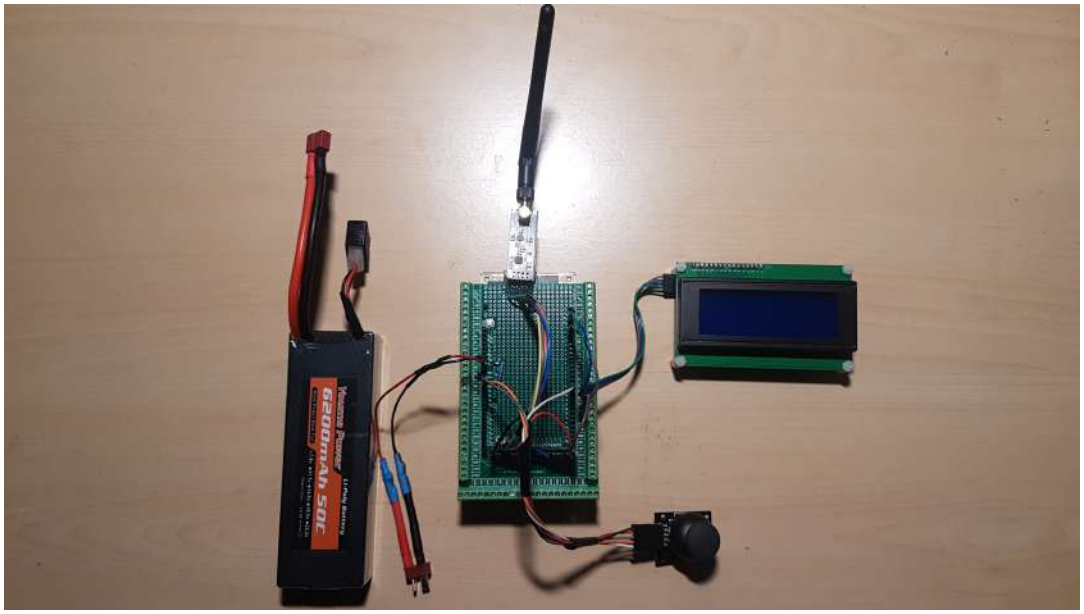


Immagine 2.6: Setup del Master

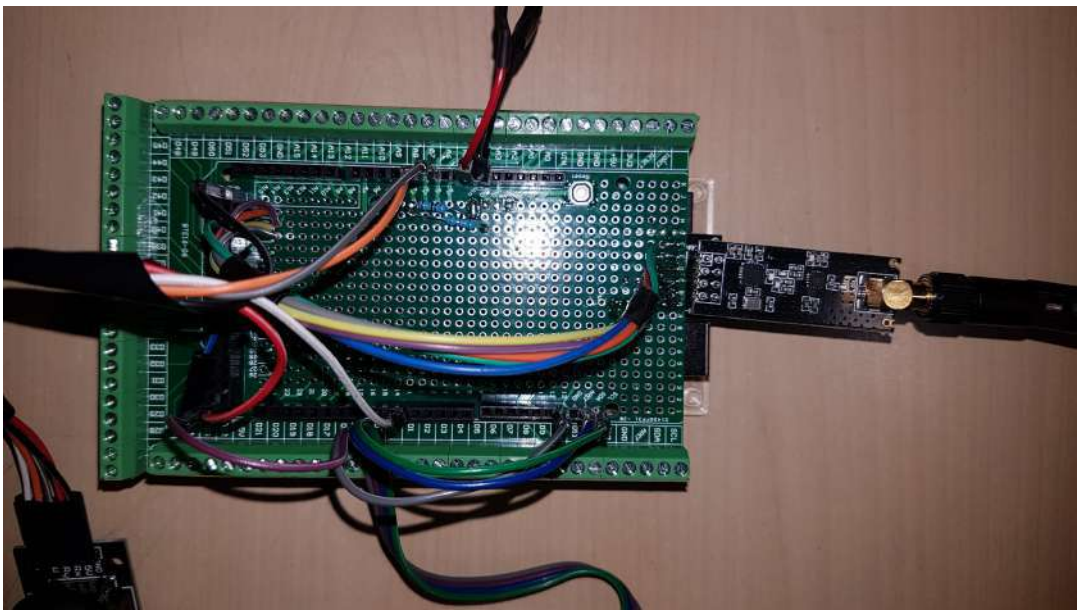


Immagine 2.7: Dettaglio del Master

- Setup dello Slave:

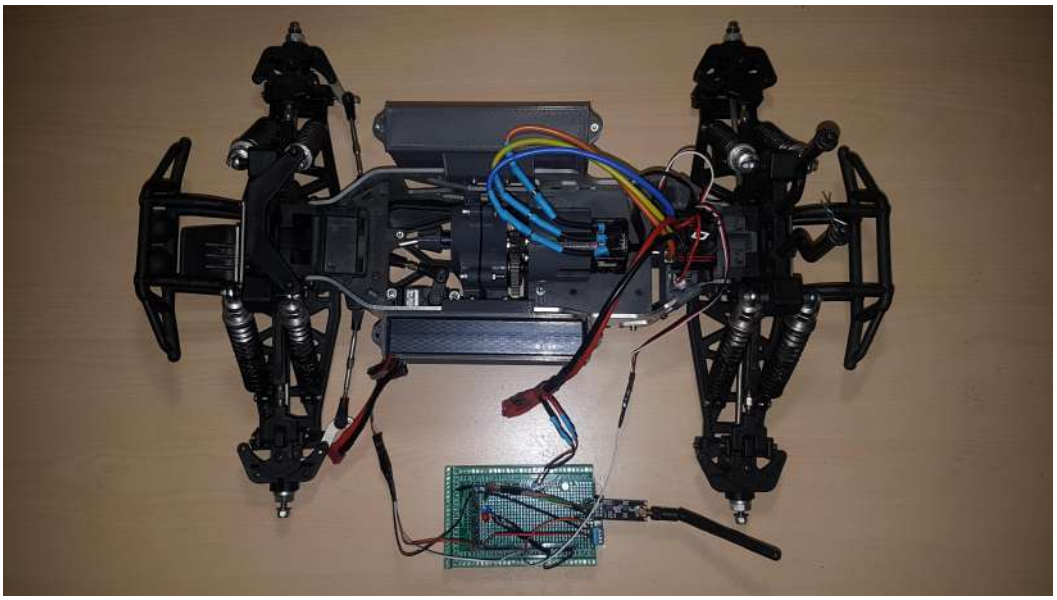


Immagine 2.8: Setup dello Slave

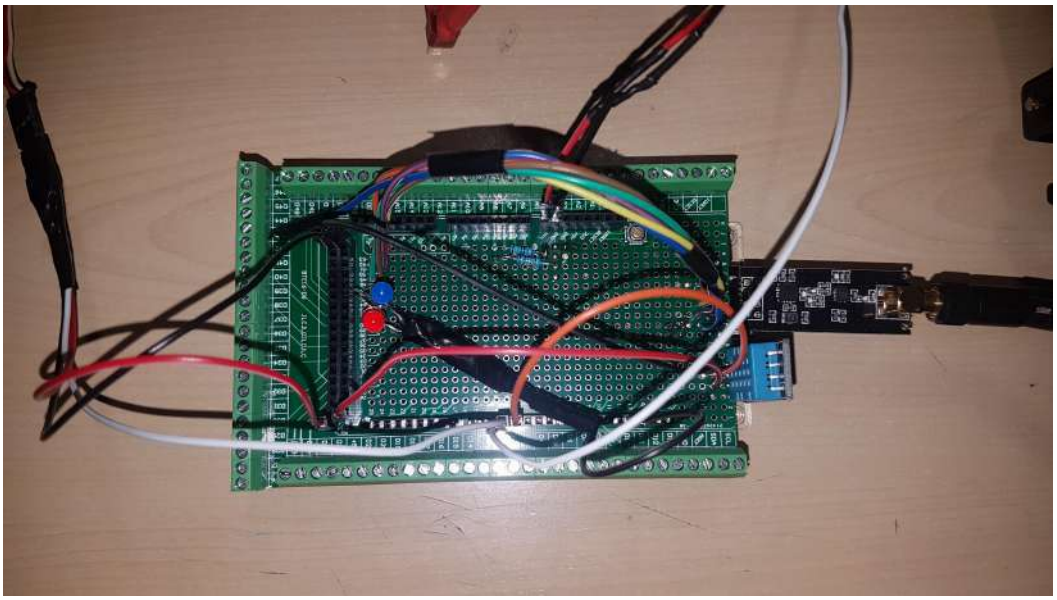


Immagine 2.9: Dettaglio dello Slave

Si noti, in particolare, che il complesso motore-ESC-servomotore è montato su di un veicolo in scala e che le due MEGA 2560 sono nascoste dalle due schede di prototipazione. Dalle figure sono chiaramente visibili i collegamenti relativi ai moduli radio, i due partitori di tensione e le varie periferiche hardware del sistema (LCD, sensori, LED, batterie, etc..).

3. Implementazione software

Questa sezione è dedicata alla presentazione e discussione dello script di controllo (realizzato e testato tramite l'IDE di Arduino).
Si noti che lo script è stato pensato in modo da poter essere utilizzato sia dal master, che dallo slave. È infatti sufficiente cambiare il valore della variabile "radioNumber" in base al ruolo del microcontrollore su cui ci si appresta a caricare lo sketch.
Ciò non rappresenta l'ottimo dal punto di vista del consumo di memoria; tuttavia, visto l'ingombro di memoria contenuto dello script, i vantaggi di utilizzare un solo script sono decisamente superiori a quelli dell'utilizzarne due separati.
Di seguito si riporta una sezione dedicata ad una lista sintetica delle funzioni della libreria utilizzate per il controllo e monitoraggio radio del sistema.

3.1. Funzioni della classe RF24 utilizzate nello script

La libreria RF24 si occupa di tutto ciò che riguarda il setup ed il funzionamento dei moduli radio. Una documentazione esaustiva della libreria è presente al seguente link:
https://nrf24.github.io/RF24/classRF24.html.
Le funzioni della libreria utilizzate dallo script sono:

- bool begin(void)
  —> inizializza il chip.
- bool isChipConnected()
  —> verifica che tutte gli 8 pin necessari ad operare il modulo radio siano connessi.
- void startListening(void)
  —> inizia l'ascolto sulle pipe abilitate all'ascolto.
- void stopListening(void)
  —> interrompe l'ascolto ed effettua la transizione alla modalità di trasmissione
- bool available(void)
  —> verifica la disponibilità di byte da leggere (nei buffer FIFO)
- void read(void *buf, uint8_t len)
  —> legge gli eventuali pacchetti di dati disponibili (len rappresenta il numero massimo di byte da leggere nel buffer), salvandoli all'indirizzo indicato dal puntatore buf.
- bool write(const void *buf, uint8_t len)
  —> invia i dati puntati da buf, finchè un'ack non è ricevuta, o finchè non si raggiunge un ritardo preimpostabile (max 60/70 ms).
- void openWritingPipe(const uint8_t *address)
  —> apre una pipe per la scrittura via byte array, con indirizzo specificato da address (di default specificato da 5 byte).
- void openReadingPipe(uint8_t number, const uint8_t *address)
  —> apre una pipe di lettura con indirizzo specifico; possono essere aperte fino a 6 pipe per la lettura simultanea da più nodi.

- void enableAckPayload(void)
—> abilità payload personalizzati di riconoscimento.

- void enableDynamicPayloads(void)
—> abilità payload di dimensione variabile.

- void writeAckPayload(uint8_t pipe, const void *buf, uint8_t len)
—> Scrive un pacchetto di riconoscimento (ack) sulla pipe specificata. La prossima volta che un messaggio è ricevuto su pipe, i dati puntati da buf verranno mandati al master come acknowledgement. Questa tecnica è utilizzata dallo script per evitare che il master debba effettuare transizioni tra la modalità di lettura e scrittura. Possono pendere massimo tre ack per volta, essendo 3 i buffer FIFO. Si noti, inoltre, che è necessario abilitare i payload dinamici per poter utilizzare questa funzione.

- bool isAckPayloadAvailable(void)
—> Determina se un'ack è stata ricevuta nella chiamata più recente a write; l'ack è recuperabile semplicemente chiamando read()

- void setRetries(uint8_t delay, uint8_t count)
—> Imposta il ritardo tra ogni prova di scrittura ($0 - 15$, in multipli di 250 us), ed il numero di tentativi (max 15) .

- void setChannel(uint8_t, channel)
—> Imposta il canale di comunicazione radio ($0 - 125$) .

- void setPALevel(uint8_t level, bool lnaEnable=1)
—> Imposta il livello dell'amplificatore di potenza (PA) ad uno di quattro possibili: RF24_PA_MIN, RF24_PA_LOW, RF24_PA_HIGH and RF24_PA_MAX .

- void setDataRate(rf24_datarate_e speed)
—> Imposta la frequenza di trasmissione ad uno di tre possibili livelli: RF24_250KBPS per 250kbs, RF24_1MBPS per 1Mbps, or RF24_2MBPS per 2Mbps .

3.2. Funzioni definite all'interno dello script

Le funzioni definite all'interno dello script sono:

- void rf_setupMS():

  Inizializza la radio del master, il canale di comunicazione, abilita l'autoack (ed anche i payload a lunghezza variabile), imposta le pipe di lettura e scrittura e tutta una serie di altri parametri (si rimanda allo script per ulteriori dettagli). Dato che il master non farà altro che inviare e leggere l'ack ricevuta dello slave, in questa funzione viene anche settata permanentemente la modalità di scrittura.

```
void rf_setupMS(){
  //setup modulo radio master
  //l'RF24L01+ può trasmettere un massimo di 32 byte in una singola
  //trasmissione
  radio.begin();
  radio.setChannel(125);//0-125/2.400 -2.525 GHz
  // (> freq-> < interferenze da disp wifi)
  radio.setDataRate(RF24_250KBPS);//250KBPS-1KBPS-2KBPS(la sensibilità
  // scende con freq. di trasmissione)
  radio.setPALevel(RF24_PA_HIGH,1);// RF24_PA_MIN, RF24_PA_LOW,
  // RF24_PA_HIGH, RF24_PA_MAX;lna_enable
  radio.setRetries(10,10);//ritardo(multipli di 250 us;da 0 a 15);
  //numero di prove (max 15)
  radio.setAutoAck(true);// non necessario, di default già true
  radio.enableAckPayload();// consenti i payload ack opzionali
  radio.enableDynamicPayloads();
  radio.openWritingPipe(master_address);//pipe 0 di default scrittura
  radio.openReadingPipe(1,slave_address);//pipe 1-5 disponibili per la
  // lettura eventualmente attivabili contemporaneamente; possibile la
  // comunicazione con altri 6 dispositivi
  radio.stopListening();
}
```

- void rf_setupSL():

  Inizializza la radio dello slave, il canale di comunicazione, abilita l'autoack (ed anche i payload a lunghezza variabile), imposta le pipe di lettura e scrittura e tutta una serie di altri parametri (si rimanda allo script per ulteriori dettagli). In questo caso lo slave non viene settato permanentemente in ascolto, perchè altrimenti non si riesce a fare in modo che esso invii al master un ack contenente sempre e solamente le letture dei sensori.

```
void rf_setupSL(){
  //setup modulo radio slave
  //l'RF24L01+ può trasmettere un massimo di 32 byte in una singola
  //trasmissione
  radio.begin();
```

```
  radio.setChannel(125);//0-125/2.400--2.525 GHz (> freq-->,
  // < interferenze da disp wifi)
  radio.setDataRate(RF24_250KBPS);//250KBPS-1KBPS-2KBPS (la sensibilità
  // scende se freq. di trasmissione cresce)
  radio.setPALevel(RF24_PA_HIGH,1);// RF24_PA_MIN, RF24_PA_LOW,
  // RF24_PA_HIGH, RF24_PA_MAX;lna_enable
  radio.setRetries(10,10);//ritardo(multipli di 250 us;da 0 a 15);
  //numero di prove (max 15)
  radio.setAutoAck(true);// non necessario, di default già true
  radio.enableAckPayload();// consenti i payload ack opzionali
  radio.enableDynamicPayloads();
  radio.openWritingPipe(slave_address);//pipe 0 di default scrittura
  radio.openReadingPipe(1,master_address);//pipe 1-5 disponibili per la
  // lettura eventualmente attivabili contemporaneamente; possibile la
  // comunicazione con altri 6 dispositivi
}
```

- float rdBattery():

  Effettua la lettura dal pin analogico deputato alla misurazione della tensione delle batterie, converte il valore in volt ed effettua una correzione definita da una calibrazione preliminare del partitore.

```
float rdBattery(){//monitoraggio batteria di alimentazione
  float voltage;
  if (!radioNumber){
    voltage= (R1+R2)/R2*analogRead(divider)/1023.0*5.0*volt_corr_ms;
  //tensione della batteria, ottenuta dal valore sul partitore
  //relazione poi calibrata con multimetro
  // risoluzione default sui pin 10 bit
  }
  else{
    voltage= (R1+R2)/R2*analogRead(divider)/1023.0*5.0*volt_corr_sl;
  }
  return voltage;
}
```

- void rdJoy():

  Legge i valori analogici provenienti dai due potenziometri angolari del joystic, senza occuparsi della conversione della lettura.

```
void rdJoy(){//funzione di lettura del joystick
  joy[0]=analogRead(joy_x);
  joy[1]=analogRead(joy_y);
}
```

- void do_master():

  Funzione di loop del master.
  Effettua la lettura del joystick e tenta la trasmissione dei due valori verso lo slave. In caso di ricezione dell'ack dallo slave, legge l'ack (conterrà tutti dati del parco sensori dello slave), legge la tensione sulla sua batteria, procede al salvataggio dei dati, per poi stampare a schermo. Si noti che la stampa a schermo non avviene ad ogni iterazione del loop, ma solamente ad intervalli di tempo prefissati. Nel caso l'ack non fosse disponibile, il master procederà alla lettura della tensione sulla sua batteria ed effettuerà la stampa a schermo della sola tensione.

```
void do_master(){
  unsigned long currentMillis = millis();//in questo modo l'lcd
  // verrà aggiornato con la frequenza desiderata;
  //contatore per evitare di stampare con troppa
  // frequenza a schermo
  rdJoy();//legge il joystick e sovrascrive i
  //valori globali dell'array
  radio.write(&joy,sizeof(joy));
  is_ack_ok=radio.isAckPayloadAvailable();
  if (is_ack_ok){
    radio.read(&sens,sizeof(sens));
    batt[0]=rdBattery();
    batt[1]=sens[0];
    if ((unsigned long)(currentMillis - previousMillis)>=lcd_rfrsh_rate){
    wrt2LCD(is_ack_ok,batt,sens[1],sens[2]);
    previousMillis = currentMillis;
    }
  }
  else {// se non ci sono dati da leggere, stampa a schermo la sola
    // tensione della batteria del master
    batt[0]=rdBattery();
    if ((unsigned long)(currentMillis - previousMillis)>=lcd_rfrsh_rate){
    wrt2LCD(is_ack_ok,batt,sens[1],sens[2]);
    previousMillis = currentMillis;
    }
  }
}
```

- void do_slave():

  Funzione di loop dello slave.
  La prima operazione effettuata ad ogni iterazione è il passaggio alla modalità di ricezione ed la scrittura immediata della ack personalizzata. Se il chip risulta connesso e sono presenti pacchetti dati da leggere, lo slave procede subito alla lettura, accende il led blu e smette immediatamente di ascoltare. Ciò è necessario per evitare che durante l'esecuzione del resto del programma il master tenti di comunicare con lo slave e quest'ultimo invii al master un'ack generata automaticamente; quest'ack non conterrebbe i dati dei sensori ed, in più, provocherebbe il blocco per overflow dell'lcd (ciò renderebbe necessario un reset manuale del master).
  Se non sono presenti dati da leggere, spegne entrambi i led. Se invece il chip non risulta connesso, accende il led rosso e spegne il led blu (ques'ultima operazione è necessaria per evitare che il led blu rimanga acceso in seguito ad un'eventuale precedente comunicazione andata a buon fine).

```
void do_slave(){
  radio.startListening();
  radio.writeAckPayload(1,&sens,sizeof(sens));//carica nel buffer FIFO
  // (max 3 in attesa)dello slave il payload dei dati dei sensori, che
  //verrà automaticamente inviato al Master in seguito alla prossima
  // ricezione sulla pipe selezionata (ogni float occupa 4 byte)
  unsigned long currentMillis = millis();
  //contatore per evitare di leggere i dati con troppa
  // frequenza e così pregiudicare la rapidità del programma
  if (radio.isChipConnected()){
      if(radio.available()){
      radio.read(&joy,sizeof(joy));
      radio.stopListening();//smette di ascoltare immediatamente
      //altrimenti potrebbe ricevere nel frattempo la scrittura dal
      // master e mandare indietro un acknowledge che non coincide con
      // i dati dei sensori. Questo produce l'overflow dell'lcd ed
      // il blocco del Master
      digitalWrite(ledb,HIGH);
      ESC.writeMicroseconds((int)((float)min_micros+
      (float)(max_micros-min_micros)*(joy[1]/1023.0)));
      servo.writeMicroseconds((int)((float)min_micros+(float)
      (max_micros-min_micros)*(joy[0]/1023.0)));
    }
    else{
      digitalWrite(ledb,LOW);
    }
    digitalWrite(ledr,LOW);
  }
  else{
     digitalWrite(ledr,HIGH);
     digitalWrite(ledb,LOW);
  }
  sens[0]= rdBattery();
```

```
  if ((unsigned long)(currentMillis - previousMillis)>=dht11_rfrsh_rate){
    sens[1]=dht.readTemperature();// la lettura è lenta;
    // per evitare di rallentare la ricezione dei comandi dal Master
    // si legge solo ogni tot millisecondi; quando non si legge
    // vengono iniviati al master le letture vecchie della temp ed um
    sens[2]=dht.readHumidity();
    previousMillis = currentMillis;
  }
}
```

- void lcd_startup():

  Questa funzione si occupa dell'inizializzazione dell'lcd e della visualizzazione a schermo della procedura di avvio del master. Ciò comprende anche la lettura del terzo output del joystick, cioè il bottone, utilizzato come comando di prosecuzione nell'esecuzione del resto dello script.

```
void lcd_startup(){
  //inizializzazione lcd
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0,0);//colonna,riga
  lcd.print(F("INIZ. MASTER..... "));
  delay(1000);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(F("  PER INIZIARE LA   "));
  lcd.setCursor(0,1);
  lcd.print(F("COMUN. CON LO SLAVE "));
  lcd.setCursor(0,2);
  lcd.print(F("PREMERE IL JOYSTICK "));
  while (!butt){//bottone in logica negativa
    butt=!(bool)digitalRead(button);
  }
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print(F("   RICEVUTO CAPO,   "));
  lcd.setCursor(0,1);
  lcd.print(F("INIZIALIZZAZIONE IN "));
  lcd.setCursor(0,2);
  lcd.print(F("       CORSO        "));
  delay(1500);
}
```

- void wrt2LCD(bool is_link_ok, float batt[2], float temp, float rel_hum):

  Questa funzione gestisce la stampa a schermo dei valori dei diversi sensori; in particolare, utilizza una variabile booleana di controllo che determina la visualizzazione di tutti i dati dei sensori se "true", oppure solamente della tensione della batteria del master in caso contrario.

```
void wrt2LCD(bool is_link_ok,float batt[2],float temp,float rel_hum){
  //funzione per la stampa a schermo delle letture dei vari sensori;
  //se i dati dallo slave sono disponibili (e.g. interruzione link rf),
  //vengono stampati a schermo, altrimenti sono visualizzati come **.**
  //ad indicare che non si è ricevuto alcun aggiornamento
  if (is_link_ok){
    lcd.setCursor(0,0);
    lcd.print(F("DATI SENSORI:"));
    lcd.setCursor(0,1);
    lcd.print(F("TEMP.:"));
    lcd.setCursor(6,1);
    lcd.print(String(temp,2));
    lcd.setCursor(11,1);
    lcd.print((char)223);
    lcd.print(F("C"));
    lcd.setCursor(0,2);
    lcd.print(F("REL. HUM:"));
    lcd.setCursor(9,2);
    lcd.print(String(rel_hum,2));
    lcd.setCursor(14,2);
    lcd.print(F("%"));
    lcd.setCursor(0,3);
    lcd.print(F("SL:"));
    lcd.setCursor(3,3);
    lcd.print(String(batt[1],2));
    lcd.setCursor(9,3);
    lcd.print(F("V"));
    lcd.setCursor(10,3);
    lcd.print(F(" MS:"));
    lcd.setCursor(14,3);
    lcd.print(String(batt[0],2));
    lcd.setCursor(19,3);
    lcd.print(F("V"));
  }
  else{
    lcd.setCursor(0,0);
    lcd.print(F("DATI SENSORI:"));
    lcd.setCursor(0,1);
    lcd.print(F("TEMP.:"));
    lcd.setCursor(6,1);
```

```
    lcd.print(F("**.**"));
    lcd.setCursor(11,1);
    lcd.print((char)223);
    lcd.print(F("C"));
    lcd.setCursor(0,2);
    lcd.print(F("REL.HUM:"));
    lcd.setCursor(9,2);
    lcd.print(F("**.**"));
    lcd.setCursor(14,2);
    lcd.print(F("%"));
    lcd.setCursor(0,3);
    lcd.print(F("SL:"));
    lcd.setCursor(3,3);
    lcd.print(F("**.*"));
    lcd.setCursor(9,3);
    lcd.print(F("V"));
    lcd.setCursor(10,3);
    lcd.print(F(" MS:"));
    lcd.setCursor(14,3);
    lcd.print(String(batt[0],2));
    lcd.setCursor(19,3);
    lcd.print(F("V"));
  }
}
```

3.3. Preambolo

Nel preambolo dello script viene innanzitutto effettuato il caricamento di tutte le librerie necessarie:

- SPI ("Serial peripheral interface")
- nRF24L01 (comunicazione rf)
- RF24 (comunicazione rf)
- Wire (comunicazione I2C)
- LiquidCrystal_I2C (gestione LCD, utilizza Wire)
- DHT (lettura sensore di umidità e temperatura DHT11)

Successivamente, si passa alla definizione di tutte le variabili globali utilizzate nello script (costanti e non), sia per il master, che lo slave. Infine, si procede all'inizializzazione degli oggetti che utilizzano le librerie: "servo" (eventualmente ESC, se utilizzata), "lcd" e "dht".

```
//DIRETTIVE AL PREPROCESSORE
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <Servo.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"// per leggere um. e temp dal DHT11
#define DHTPIN 4 // pin digitale connesso al DHT11
#define DHTTYPE DHT11 //tipologia sensore
//////////////////////////////////////////////////////
bool radioNumber = 0;//0 MASTER, 1 SLAVE (oppure
// qualsiasi numero diverso da 0)
//////////////////////////////////////////////////////
const byte slave_address[6] = "00001";//indirizzo di comunicazione;
// numero max di canali attivabili contemporaneamente: 6
const byte master_address[6] = "10000";
const float R1=10.0; // resitore 1 MOhm
const float R2=1.0;// resistore 100 kOhm
const char divider=A3;// pin analogico di lettura della tensione della batt.
float sens[3];// conterrà le letture dai sensori (array usato dallo
//Slave in scrittura, dal Master in lettura)
int joy[2];// posizione angolare assi joystick(array usato dal
//Master in scrittura, dallo Slave in lettura)
bool butt=false;//(variabile bool usata dallo
//Master in scrittura, dallo Slave in lettura (eventualmente))
unsigned long previousMillis=0;//utilizzato per temporizzare
// l'esecuzione di alcune righe di codice
//////////////////////////////////////////////////////
//MASTER
const float volt_corr_ms=1;//fattore di correzione lettura tensione
```

```
bool is_ack_ok=0;//var. booleana che controlla la presenzabutton
// di un acknowledgement
const int button=2;//pin del bottone del joystick
const char joy_x=A1;//pin asse x joystick
const char joy_y=A2;//pin asse y joystick
float batt[2];//var globale per memorizzare le tensioni su
//entrambe le batterie
const int lcd_rfrsh_rate=500;//il master stampa i dati sull'LCD
// ogni sens_rfrsh_rate millisecondi
//SLAVE
const float volt_corr_sl=1.044;//fattore di correzione lettura tensione
const int esc_pin=2;//pin PWM per il pilotaggio dell'ESC
const int servo_pin=3;//pin PWM per il pilotaggio del servo
const int ledb=46;//led usato per mostrare
// lo status della trasmissione
const int ledr=44;//led usato per mostrare
// lo status della connessione del chip radio
const int min_micros=1000;// il motore ed il servo sono pilotati con
// un segnale PWM con periodo 20 ms ed ampiezza variabile
//intorno ad 1-2 micros (legato al duty cycle)
const int max_micros=2000;
const int dht11_rfrsh_rate=1000;//lo slave legge i dati di temp ed um
// ogni sens_rfrsh_rate millisecondi
////////////////////////////////////////////////////
RF24 radio(49,48);//CE ("chip enable") e CSN ("chip select not")
Servo ESC;//oggetto Servo che indica l'ESC
Servo servo;//oggetto Servo che indica il servomotore
LiquidCrystal_I2C lcd(0x27,20,4);// inizializzazione lcd
DHT dht(DHTPIN, DHTTYPE);//inizializzazione DHT11
```

3.4. setup()

Il setup viene differenziato automaticamente in base al valore assegnato alla variabile booleana globale "radioNumber". In particolare, se 0, viene effettuato il setup del master, il quale prevede l'utilizzo di lcd_startup(), rf_setupMS() ed infine la stampa a schermo di alcuni messaggi. Altrimenti, se "radioNumber" è diversa da 0, si procede con il setup dello slave, il quale prevede la dichiarazione in output dei pin che comanderanno i led, l'assegnazione dell'oggetto servo (eventualmente anche ESC) al relativo pin PWM, l'inizializzazione del DHT11 ed, infine, la chiamata a rf_setupSL ().

```
void setup() {
  if (!radioNumber){
    digitalWrite(button, LOW);
    ////////////////////setup del master///////////
    lcd_startup(); // setup LCD
```

```
    rf_setupMS();//setup modulo radio master
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print(F("  INIZIALIZZAZIONE  "));
    lcd.setCursor(0,2);
    lcd.print(F("  RADIO TERMINATA   "));
    delay(2000);
    lcd.clear();
    lcd.setCursor(0,1);
    lcd.print(F(" IN ATTESA DEL LINK "));
    lcd.setCursor(0,2);
    lcd.print(F("CON IL DISPOSITIVO.."));
    delay(2000);
    lcd.clear();
  }
  else {
    ////////setup dello slave (ogni valore diverso da 0 è trattato
    // come slave)///////////////
    pinMode(ledr,OUTPUT);
    pinMode(ledb,OUTPUT);
    ESC.attach(esc_pin,min_micros,max_micros);
    servo.attach(servo_pin);
    dht.begin();// inizializza il sensore di temp DTH11
    rf_setupSL();//setup modulo radio slave
  }
}
```

3.5. loop()

Come il setup, il loop è differenziato in base al valore di "radioNumber". I loop del master e dello slave si limitano semplicemente a chiamare, rispettivamente, do_master() e do_slave().

```
void loop() {
  if (!radioNumber){//master loop
    do_master();
  }
  else {//slave loop
    do_slave();
  }
}
```

3.6. Diagramma di flusso esplicativo dello script

Di seguito si riporta un diagramma di flusso che descrive sinteticamente l'esecuzione dello script (per semplicità, sono stati omessi tutti quei dettagli/operazioni ritenute non strettamente necessarie alla comprensione, ponendo l'accento sull'utilizzo del modulo radio).

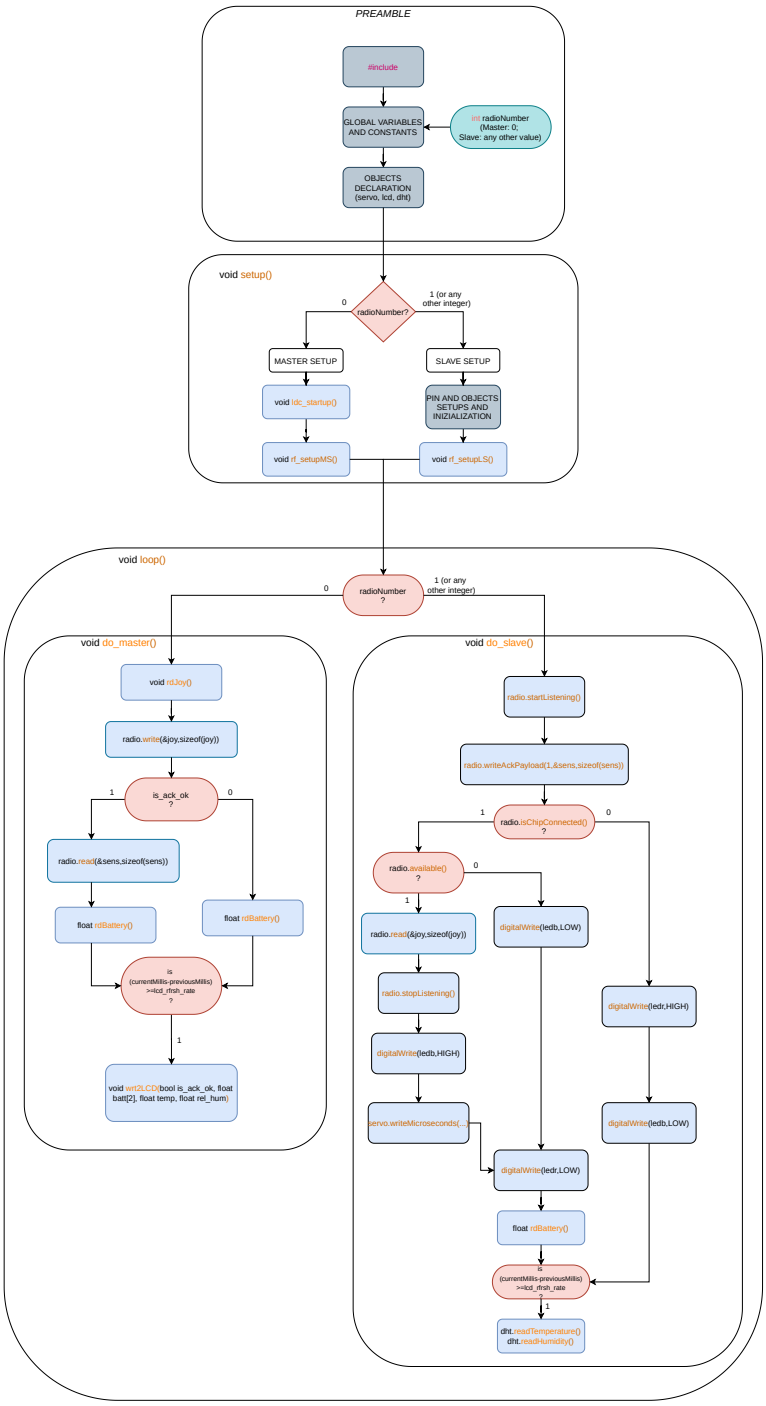


Immagine 3.1: Diagramma di flusso sintetico del programma

3.7. Script completo

```
//DIRETTIVE AL PREPROCESSORE
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
#include <Wire.h>
#include <Servo.h>
#include <LiquidCrystal_I2C.h>
#include "DHT.h"// per leggere um. e temp dal DHT11
#define DHTPIN 4 // pin digitale connesso al DHT11
#define DHTTYPE DHT11 //tipologia sensore
//////////////////////////////////////////////////////
bool radioNumber = 0;//0 MASTER, 1 SLAVE (oppure
// qualsiasi numero diverso da 0)
//////////////////////////////////////////////////////
const byte slave_address[6] = "00001";//indirizzo di comunicazione;
// numero max di canali attivabili contemporaneamente: 6
const byte master_address[6] = "10000";
const float R1=10.0; // resitore 1 MOhm
const float R2=1.0;// resistore 100 kOhm
const char divider=A3;// pin analogico di lettura della tensione della batt.
float sens[3];// conterrà le letture dai sensori (array usato dallo
//Slave in scrittura, dal Master in lettura)
int joy[2];// posizione angolare assi joystick(array usato dal
//Master in scrittura, dallo Slave in lettura)
bool butt=false;//(variabile bool usata dallo
//Master in scrittura, dallo Slave in lettura (eventualmente))
unsigned long previousMillis=0;//utilizzato per temporizare
// l'esecuzione di alcune righe di codice
//////////////////////////////////////////////////////
//MASTER
const float volt_corr_ms=1;//fattore di correzione lettura tensione
bool is_ack_ok=0;//var. booleana che controlla la presenzabutton
// di un acknowledgement
const int button=2;//pin del bottone del joystick
const char joy_x=A1;//pin asse x joystick
const char joy_y=A2;//pin asse y joystick
float batt[2];//var globale per memorizzare le tensioni su
//entrambe le batterie
const int lcd_rfrsh_rate=500;//il master stampa i dati sull'LCD
// ogni sens_rfrsh_rate millisecondi
//SLAVE
const float volt_corr_sl=1.044;//fattore di correzione lettura tensione
const int esc_pin=2;//pin PWM per il pilotaggio dell'ESC
const int servo_pin=3;//pin PWM per il pilotaggio del servo
```

```
45 const int ledb=46;//led usato per mostrare
46 // lo status della trasmissione
47 const int ledr=44;//led usato per mostrare
48 // lo status della connessione del chip radio
49 const int min_micros=1000;// il motore ed il servo sono pilotati con
50 // un segnale PWM con periodo 20 ms ed ampiezza variabile
51 //intorno ad 1-2 micros (legato al duty cycle)
52 const int max_micros=2000;
53 const int min_micros_esc=900;// calibrazione per il motore
54 const int max_micros_esc=1990;// calibrazione per il motore
55 const int dht11_rfrsh_rate=1000;//lo slave legge i dati di temp ed um
56 // ogni sens_rfrsh_rate millisecondi
57 ////////////////////////////////////////////////////
58 RF24 radio(49,48);//CE ("chip enable") e CSN ("chip select not")
59 Servo ESC;//oggetto Servo che indica l'ESC
60 Servo servo;//oggetto Servo che indica il servomotore
61 LiquidCrystal_I2C lcd(0x27,20,4);// inizializzazione lcd
62 DHT dht(DHTPIN, DHTTYPE);//inizializzazione DHT11
63 ////////////////////////////////////////////////////
64 void setup() {
65   if (!radioNumber){
66     digitalWrite(button, HIGH);
67     ////////////////////setup del master///////////
68     lcd_startup(); // setup LCD
69     rf_setupMS();//setup modulo radio master
70     lcd.clear();
71     lcd.setCursor(0,1);
72     lcd.print(F("  INIZIALIZZAZIONE  "));
73     lcd.setCursor(0,2);
74     lcd.print(F("  RADIO TERMINATA   "));
75     delay(2000);
76     lcd.clear();
77     lcd.setCursor(0,1);
78     lcd.print(F(" IN ATTESA DEL LINK "));
79     lcd.setCursor(0,2);
80     lcd.print(F("CON IL DISPOSITIVO.."));
81     delay(2000);
82     lcd.clear();
83   }
84   else {
85     ////////setup dello slave (ogni valore diverso da 0 è trattato
86     // come slave)//////////////
87     pinMode(ledr,OUTPUT);
88     pinMode(ledb,OUTPUT);
89     ESC.attach(esc_pin,min_micros_esc,max_micros_esc);
90     servo.attach(servo_pin,min_micros,max_micros);
```

```
    dht.begin();// inizializza il sensore di temp DTH11
    rf_setupSL();//setup modulo radio slave
  }
}
void loop() {
  if (!radioNumber){//master loop
    do_master();
  }
  else {//slave loop
    do_slave();
  }
}
//////////////////FUNZIONI/////////////////////////////////////////
// SETUP RADIO //
void rf_setupMS(){
  //setup modulo radio master
  //l'RF24L01+ può trasmettere un massimo di 32 byte in una singola
  //trasmissione
  radio.begin();
  radio.setChannel(125);//0-125/2.400 -2.525 GHz
  // (> freq-> < interferenze da disp wifi)
  radio.setDataRate(RF24_250KBPS);//250KBPS-1KBPS-2KBPS(la sensibilità
  // scende con freq. di trasmissione)
  radio.setPALevel(RF24_PA_HIGH,1);// RF24_PA_MIN, RF24_PA_LOW,
  // RF24_PA_HIGH, RF24_PA_MAX;lna_enable
  radio.setRetries(10,10);//ritardo(multipli di 250 us;da 0 a 15);
  //numero di prove (max 15)
  radio.setAutoAck(true);// non necessario, di default già true
  radio.enableAckPayload();// consenti i payload ack opzionali
  radio.enableDynamicPayloads();
  radio.openWritingPipe(master_address);//pipe 0 di default scrittura
  radio.openReadingPipe(1,slave_address);//pipe 1-5 disponibili per la
  // lettura eventualmente attivabili contemporaneamente; possibile la
  // comunicazione con altri 6 dispositivi
  radio.stopListening();
}
void rf_setupSL(){
  //setup modulo radio slave
  //l'RF24L01+ può trasmettere un massimo di 32 byte in una singola
  //trasmissione
  radio.begin();
  radio.setChannel(125);//0-125/2.400--2.525 GHz (> freq-->, < interferenze
  // da disp wifi)
  radio.setDataRate(RF24_250KBPS);//250KBPS-1KBPS-2KBPS (la sensibilità
  // scende se freq. di trasmissione cresce)
  radio.setPALevel(RF24_PA_HIGH,1);// RF24_PA_MIN, RF24_PA_LOW,
```

```
137   // RF24_PA_HIGH, RF24_PA_MAX;lna_enable
138   radio.setRetries(10,10);//ritardo(multipli di 250 us;da 0 a 15);
139   //numero di prove (max 15)
140   radio.setAutoAck(true);// non necessario, di default già true
141   radio.enableAckPayload();// consenti i payload ack opzionali
142   radio.enableDynamicPayloads();
143   radio.openWritingPipe(slave_address);//pipe 0 di default scrittura
144   radio.openReadingPipe(1,master_address);//pipe 1-5 disponibili per la
145   // lettura eventualmente attivabili contemporaneamente; possibile la
146   // comunicazione con altri 6 dispositivi
147 }
148 // LETTURA SENSORI/ATTUATORI //
149 float rdBattery(){//monitoraggio batteria di alimentazione
150   float voltage;
151   if (!radioNumber){
152     voltage= (R1+R2)/R2*analogRead(divider)/1023.0*5.0*volt_corr_ms;
153   //tensione della batteria, ottenuta dal valore sul partitore
154   //relazione poi calibrata con multimetro
155   // risoluzione default sui pin 10 bit
156   }
157   else{
158     voltage= (R1+R2)/R2*analogRead(divider)/1023.0*5.0*volt_corr_sl;
159   }
160   return voltage;
161 }
162 void rdJoy(){//funzione di lettura del joystick
163   joy[0]=analogRead(joy_x);
164   joy[1]=analogRead(joy_y);
165 }
166 // FUNZIONI LOOP del MASTER E SLAVE //
167 void do_master(){
168   unsigned long currentMillis = millis();//in questo modo l'lcd
169   // verrà aggiornato con la frequenza desiderata;
170   //contatore per evitare di stampare con troppa
171   // frequenza a schermo
172   rdJoy();//legge il joystick e sovrascrive i
173   //valori globali dell'array
174   radio.write(&joy,sizeof(joy));
175   is_ack_ok=radio.isAckPayloadAvailable();
176   if (is_ack_ok){
177     radio.read(&sens,sizeof(sens));
178     batt[0]=rdBattery();
179     batt[1]=sens[0];
180     if ((unsigned long)(currentMillis - previousMillis)>=lcd_rfrsh_rate){
181     wrt2LCD(is_ack_ok,batt,sens[1],sens[2]);
182     previousMillis = currentMillis;
```

```
183       }
184     }
185     else {// se non ci sono dati da leggere, stampa a schermo la sola tensione
186       // della batteria del master
187       batt[0]=rdBattery();
188       if ((unsigned long)(currentMillis - previousMillis)>=lcd_rfrsh_rate){
189       wrt2LCD(is_ack_ok,batt,sens[1],sens[2]);
190       previousMillis = currentMillis;
191       }
192     }
193   }
194   void do_slave(){
195     radio.startListening();
196     radio.writeAckPayload(1,&sens,sizeof(sens));//carica nel buffer FIFO
197     // (max 3 in attesa)dello slave il payload dei dati dei sensori, che
198     //verrà automaticamente inviato al Master in seguito alla prossima
199     // ricezione sulla pipe selezionata (ogni float occupa 4 byte)
200     unsigned long currentMillis = millis();
201     //contatore per evitare di leggere i dati con troppa
202     // frequenza e così pregiudicare la rapidità del programma
203     if (radio.isChipConnected()){
204         if(radio.available()){
205         radio.read(&joy,sizeof(joy));
206         radio.stopListening();//smette di ascoltare immediatamente
207         //altrimenti potrebbe ricevere nel frattempo la scrittura dal master
208         // e mandare indietro un acknowledge che non coincide con i dati dei
209         // sensori. Questo produce l'overflow dell'lcd ed il blocco del Master
210         digitalWrite(ledb,HIGH);
211         ESC.writeMicroseconds((int)((float)min_micros+
212         (float)(max_micros-min_micros)*(joy[1]/1023.0)));
213         servo.writeMicroseconds((int)((float)min_micros+(float)
214         (max_micros-min_micros)*(joy[0]/1023.0)));
215       }
216       else{
217         digitalWrite(ledb,LOW);
218       }
219       digitalWrite(ledr,LOW);
220     }
221     else{
222        digitalWrite(ledr,HIGH);
223        digitalWrite(ledb,LOW);
224     }
225     sens[0]= rdBattery();
226     if ((unsigned long)(currentMillis - previousMillis)>=dht11_rfrsh_rate){
227       sens[1]=dht.readTemperature();// la lettura è lenta;
228       // per evitare di rallentare la ricezione dei comandi dal Master
```

```
229       // si legge solo ogni tot millisecondi; quando non si legge
230       // vengono iniviati al master le letture vecchie della temp ed um
231       sens[2]=dht.readHumidity();
232       previousMillis = currentMillis;
233     }
234   }
235   // LCD //
236   void lcd_startup(){
237     //inizializzazione lcd
238     lcd.init();
239     lcd.backlight();
240     lcd.clear();
241     lcd.setCursor(0,0);//colonna,riga
242     lcd.print(F("INIZ. MASTER..... "));
243     delay(1000);
244     lcd.clear();
245     lcd.setCursor(0,0);
246     lcd.print(F("  PER INIZIARE LA   "));
247     lcd.setCursor(0,1);
248     lcd.print(F("COMUN. CON LO SLAVE "));
249     lcd.setCursor(0,2);
250     lcd.print(F("PREMERE IL JOYSTICK "));
251     while (!butt){//bottone in logica negativa
252       butt=!(bool)digitalRead(button);
253     }
254     lcd.clear();
255     lcd.setCursor(0,0);
256     lcd.print(F("   RICEVUTO CAPO,   "));
257     lcd.setCursor(0,1);
258     lcd.print(F("INIZIALIZZAZIONE IN "));
259     lcd.setCursor(0,2);
260     lcd.print(F("       CORSO        "));
261     delay(1500);
262   }
263   void wrt2LCD(bool is_link_ok,float batt[2],float temp,float rel_hum){
264     //funzione per la stampa a schermo delle letture dei vari sensori;
265     //se i dati dallo slave sono disponibili (e.g. interruzione link rf),
266     //vengono stampati a schermo, altrimenti sono visualizzati come **.**
267     //ad indicare che non si è ricevuto alcun aggiornamento
268     if (is_link_ok){
269       lcd.setCursor(0,0);
270       lcd.print(F("DATI SENSORI:"));
271       lcd.setCursor(0,1);
272       lcd.print(F("TEMP.:"));
273       lcd.setCursor(6,1);
274       lcd.print(String(temp,2));
```

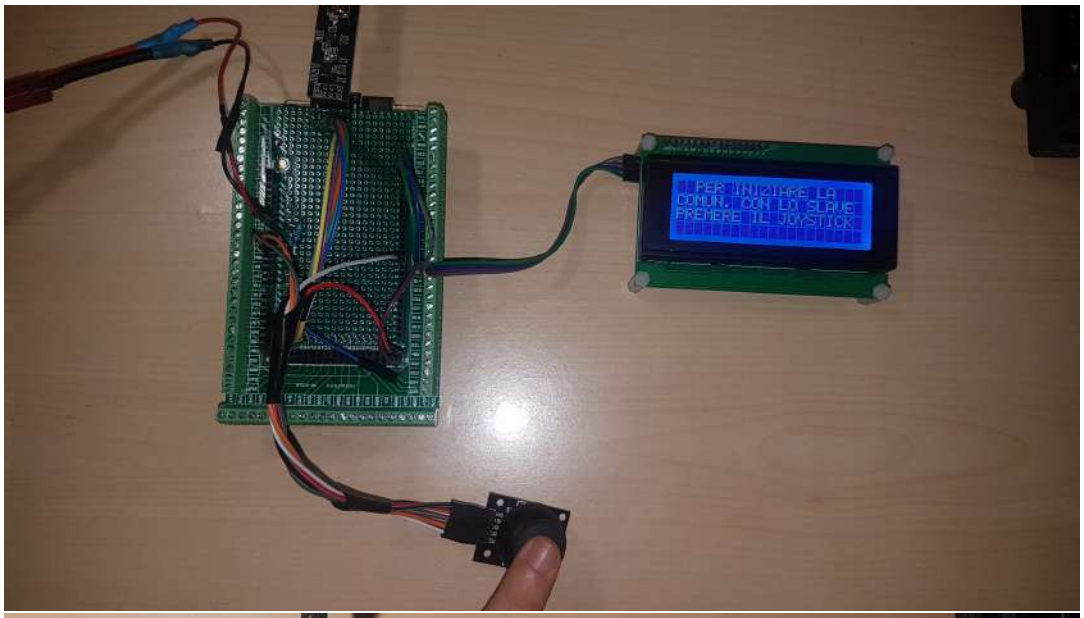
```
275       lcd.setCursor(11,1);
276       lcd.print((char)223);
277       lcd.print(F("C"));
278       lcd.setCursor(0,2);
279       lcd.print(F("REL. HUM:"));
280       lcd.setCursor(9,2);
281       lcd.print(String(rel_hum,2));
282       lcd.setCursor(14,2);
283       lcd.print(F("%"));
284       lcd.setCursor(0,3);
285       lcd.print(F("SL:"));
286       lcd.setCursor(3,3);
287       lcd.print(String(batt[1],2));
288       lcd.setCursor(9,3);
289       lcd.print(F("V"));
290       lcd.setCursor(10,3);
291       lcd.print(F(" MS:"));
292       lcd.setCursor(14,3);
293       lcd.print(String(batt[0],2));
294       lcd.setCursor(19,3);
295       lcd.print(F("V"));
296     }
297     else{
298       lcd.setCursor(0,0);
299       lcd.print(F("DATI SENSORI:"));
300       lcd.setCursor(0,1);
301       lcd.print(F("TEMP.:"));
302       lcd.setCursor(6,1);
303       lcd.print(F("**.**"));
304       lcd.setCursor(11,1);
305       lcd.print((char)223);
306       lcd.print(F("C"));
307       lcd.setCursor(0,2);
308       lcd.print(F("REL.HUM:"));
309       lcd.setCursor(9,2);
310       lcd.print(F("**.**"));
311       lcd.setCursor(14,2);
312       lcd.print(F("%"));
313       lcd.setCursor(0,3);
314       lcd.print(F("SL:"));
315       lcd.setCursor(3,3);
316       lcd.print(F("**.*"));
317       lcd.setCursor(9,3);
318       lcd.print(F("V"));
319       lcd.setCursor(10,3);
320       lcd.print(F(" MS:"));
```

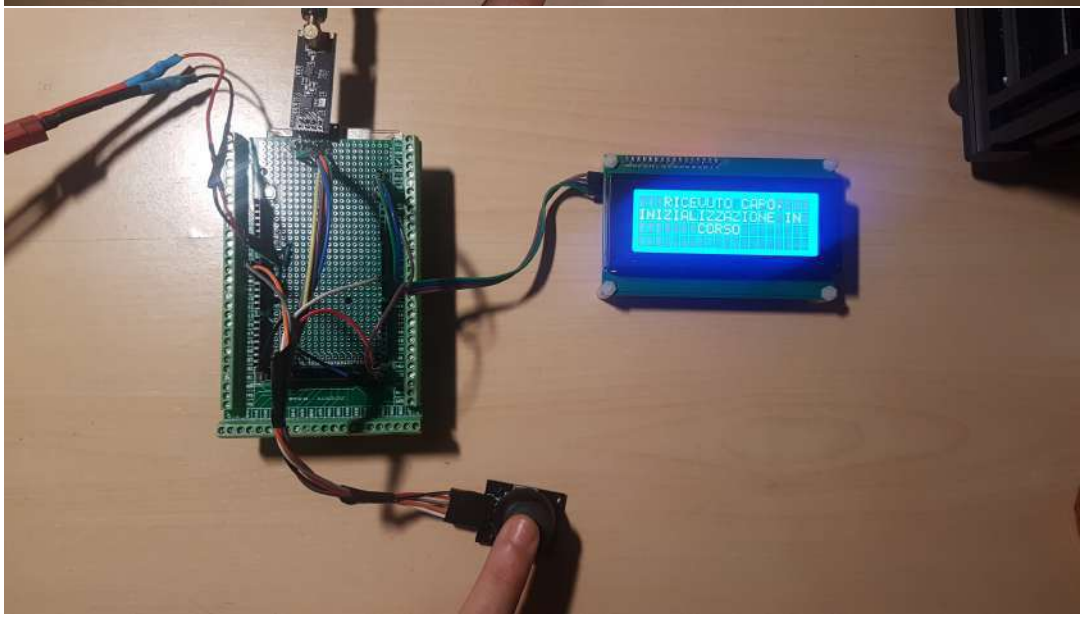
```
321     lcd.setCursor(14,3);
322     lcd.print(String(batt[0],2));
323     lcd.setCursor(19,3);
324     lcd.print(F("V"));
325   }
326 }
```

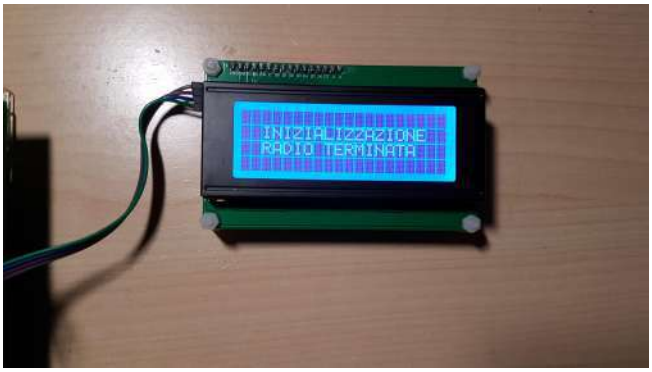
4. Funzionamento

4.1. Fase di avvio del Master

Una volta connessa la batteria di alimentazione, il Master effettua una fase di inizializzazione che consiste nella stampa a schermo di alcuni messaggi e nell'attesa della pressione, da parte dell'utente, del pulsante digitale del joystick.


PER INIZIARE LA
COMUN. CON LO SLAVE
PREMERE IL JOYSTICK


RICEVUTO CAFO.
INIZIALIZZAZIONE IN
CORSO



Una volta premuto il pulsante, il Master effettua l'intera procedura di setup della radio ed entra nel loop, mettendosi in attesa dei dati provenienti dallo slave. Di seguito è riportata un'immagine di come i dati vengono presentati a schermo se lo Slave non stabilisce una connessione con il Master: in particolare, la sola tensione di alimentazione del Master è stampata a schermo.

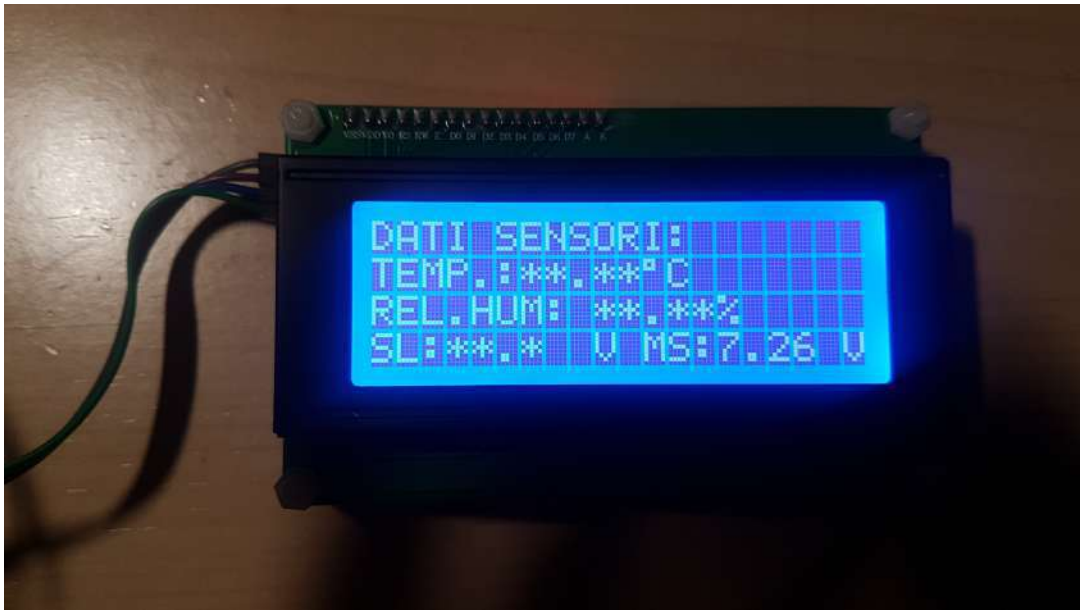


Immagine 4.1: Presentazione dati sull'lcd del Master nel caso di connessione con lo Slave assente

4.2. Funzionamento normale del Master e dello Slave, con ritorno dei dati del parco sensori dello Slave

Una volta connesso lo Slave all'alimentazione, se la radio risulta ben collegata e se il link con il Master ha successo, il led blu si accende, segnalando che la comunicazione bidirezionale avviene con successo. Nel caso le connessioni della radio dello slave siano corrette, ma il Master non sia disponibile per qualche motivo, il led blu si mantiene spento.

Immagine 4.2: Sistema completo in funzione, con successo del link Master-Slave

Immagine 4.3: Esempio di lettura e presentazione dei dati sull'lcd

Infine, nel caso qualcosa andasse storto ed i collegamenti fisici della radio dello Slave si iterrompessero, allora il led rosso si accenderebbe, come mostrato nella seguente figura (si noti che il modulo radio è stato scollegato per simulare un malfunzionamento):

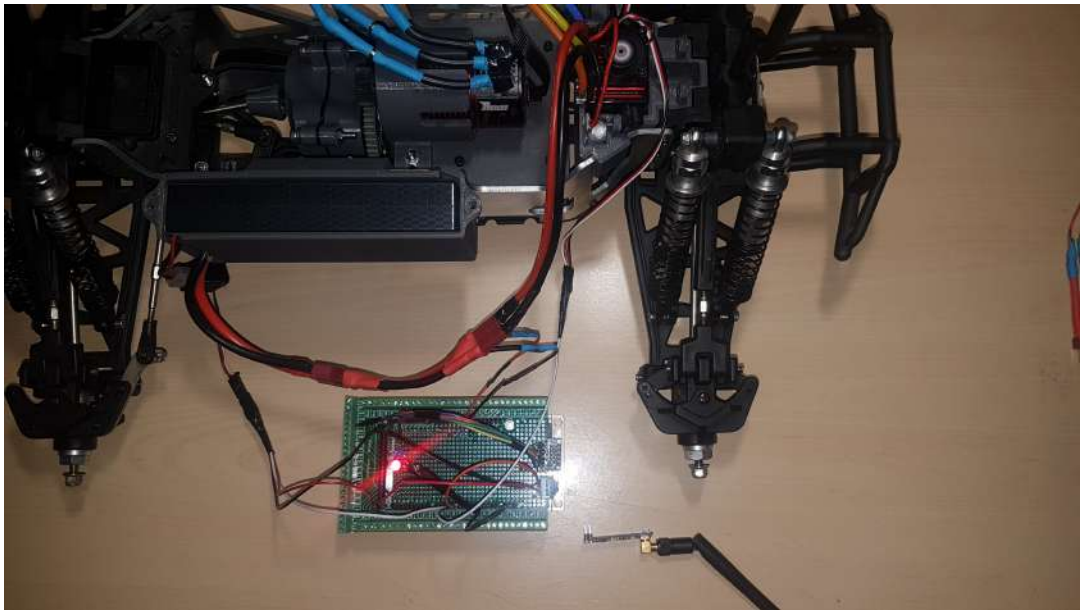


Immagine 4.4: Segnale di malfunzionamento della radio dello Slave

4.3. Controllo del motore e del servomotore

Utilizzando i due assi del joystick, è possibile regolare la velocità del motore (per il tramite dell'ESC) e la posizione del servomotore (che nello specifico controlla lo sterzo del veicolo).

Di seguito si riporta un'immagine che mostra lo spostamento del servomotore, in seguito al comando ricevuto dal Master.

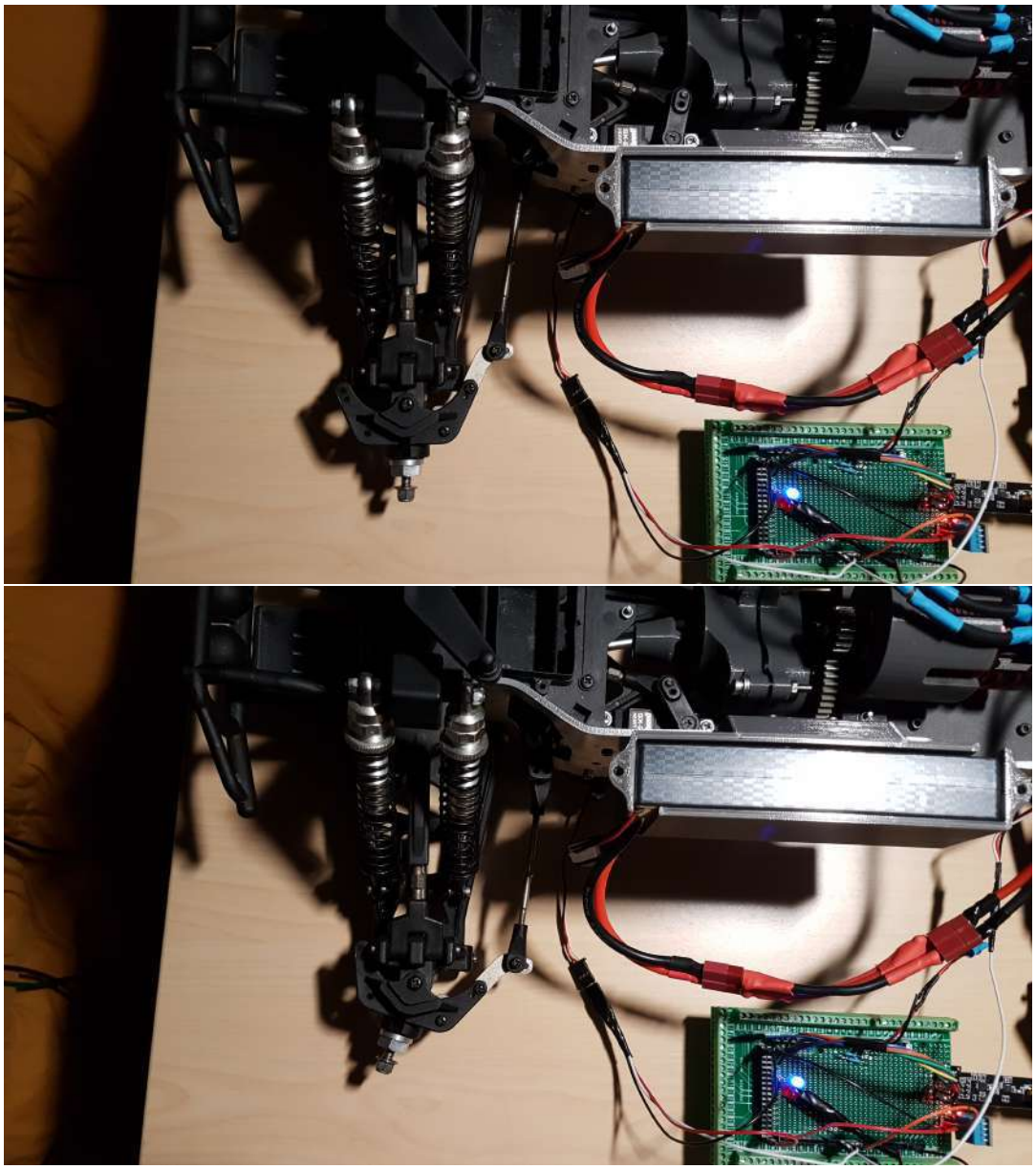


Immagine 4.5: Controllo in remoto del servomotore

Nella figura è chiaramente visibile il led blu acceso ed anche il servomotore, parzialmente nascosto dalla batteria.

Nella seguente immagine, invece, si riporta un'istantanea che cattura il controllo del motore da parte del Master. Con un po' di attenzione, si può apprezzare la rotazione della corona della trasmissione.

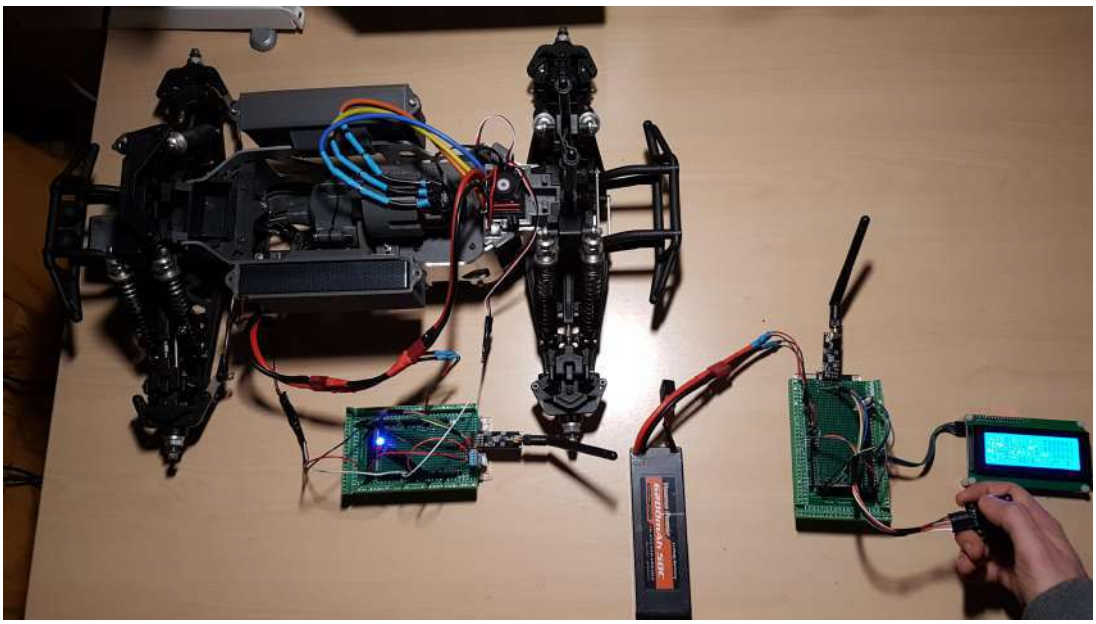


Immagine 4.6: Controllo in remoto del motore brushless

Nelle precedenti figure, le azioni di controllo sono state mostrate singolarmente. È tuttavia chiaro che, durante il funzionamento normale, tali azioni possano essere inviate e comandate contemporaneamente (a meno del ritardo dovuto all'esecuzione dei programmi da parte dei microcontrollori).

5. Commenti finali

Il sistema master-slave implementato rappresenta un sistema meccatronico a tutti gli effetti, nel quale le componenti meccanica ed elettronica convivono ed interagiscono. Il cuore del circuito, fatta eccezione per i microcontrollori, è costituito dai moduli radio, componenti di gestione particolarmente ostica e delicata, sia dal punto di vista puramente circuitale, che software. Per essere in grado di mettere efficacemente in funzione i moduli, è stato necessario un ingente lavoro di ricerca e documentazione.
Una volta archiviato il setup delle radio, il sistema in sè presenta infinite possibilità di adattamento/estensione e rappresenta un'ottima piattaforma di apprendimento/sviluppo. Questo tipo di impianto può essere applicato al controllo e monitoraggio remoto di una grande varietà di sistemi.

6. Ringraziamenti

Questa relazione è dedicata alla bellissima Nano 33 BLE sense dell'autore; nonostante la giovane età, il microcontrollore è venuto a mancare in seguito ad uno sfortunato incidente avvenuto durante gli svariati tentativi di setup dei moduli radio.
Una menzione particolare va anche alla Nano IoT dell'autore, che come la Nano BLE si è sacrificata per la causa. Attualmente versa in stato comatoso ed è in lista d'attesa per un trapianto di bootloader.