

Università degli Studi di Modena e Reggio Emilia

Corso di Progettazione di Sistemi Embedded

a.a. 2016/2017

UniMoRover: veicolo a controllo remoto con riconoscimento degli ostacoli

Enrico Mazzocchi^{1,*}, Giacomo Mellone^{2,*}, Luca Pistoni^{3,*},

Samuele Truffellini^{4,*}, Aldo Vidoni^{5,*}

**Studente, Università di Modena and Reggio Emilia*

¹88249@studenti.unimore.it

²210730@studenti.unimore.it

³84859@studenti.unimore.it

⁴214837@studenti.unimore.it

⁵85058@studenti.unimore.it

INDICE

1 ARCHITETTURA HARDWARE	5
1.1 MICROCONTROLLORI	6
1.1.1 PIC#1	6
1.1.2 PIC#2	6
1.1.3 PIC#3 E PIC#3BIS	7
1.2 FPGA (FIELD PROGRAMMABLE GATE ARRAY)	11
1.3 DRIVER MOTORI DC	14
1.3.1 SPECIFICHE DEL MODULO	14
1.3.2 SCELTA DEL MODULO	14
1.3.3 CIRCUITO E REALIZZAZIONE	15
1.3.4 TEST DEL CIRCUITO	19
1.3.5 COSTANTE DI PROPORZIONALITÀ DEL CURRENT SENSE	20
1.4 MODULO WIRELESS: NORDIC NRF24L01	21
1.5 SENSORI AD ULTRASUONI: HC-SR04	23
2 FIRMWARE	25
2.1 FPGA	26
2.1.1 VIDEO GRAPHICS ARRAY (VGA)	26
2.1.2 UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART)	30
2.1.2.1 TRASMISSIONE: TX_COUNTER.VHD	31
2.1.2.2 RICEZIONE: RX_FSM.VHD	32
2.1.3 SCHEMA FINALE E CONSIDERAZIONI	33
2.2 NORDIC NRF24L01	34
2.2.1 SERIAL PERIPHERAL INTERFACE (SPI)	34
2.2.1.1 PROGRAMMAZIONE INIZIALE E DEBUGGING	36
2.2.2 COMUNICAZIONE E PROGRAMMAZIONE DI FUNZIONI AVANZATE	41
2.3 SENSORI HC-SR04	44
2.4 CONTROLLER AREA NETWORK (CAN)	46
2.5 DRIVER MOTORI	50
2.5.1 CONTROLLO E GESTIONE DEI MOTORI	51
2.5.2 CURRENT SENSE	52
3 CONCLUSIONI E SVILUPPI FUTURI	52

INTRODUZIONE

UniMoRover è un progetto realizzato durante il corso di “Progettazione di sistemi Embedded”.

I vincoli unici del progetto sono:

- comunicazione CAN tra due o più microcontrollori;
- comunicazione UART tra microcontrollore e FPGA.

Basandosi su questi vincoli si è deciso di sviluppare un sistema per la frenata assistita su un’auto elettrica per bambini, prendendo spunto dai sistemi di sicurezza introdotti negli ultimi anni in ambito automotive.

La prima fase del progetto è incentrata su come poter utilizzare l’FPGA all’interno della nostra applicazione. Dopo un breve periodo di apprendimento nel quale sono state studiate le capacità di questo dispositivo si è deciso di sfruttarla per visualizzare a schermo lo stato della vettura. In un secondo momento, si è pensato inoltre di comandare il Rover tramite la pulsantiera integrata nell’FPGA stessa. I segnali relativi ai comandi generati dall’FPGA sono inviati ad un microcontrollore utilizzando la comunicazione UART, rispettando uno dei due vincoli del progetto.

Il passo successivo è stato la scelta dei sensori per il rilevamento delle distanze dagli ostacoli; tali sensori sono posizionati nella parte anteriore e posteriore della vettura e sono gestiti da due microcontrollori, connessi a loro volta ad un’unità centrale tramite bus CAN, altro vincolo del progetto.

I segnali che circolano sul bus CAN provenienti dai sensori vengono utilizzati dall’unità centrale per pilotare i driver dei motori.

La parte di progetto più dispendiosa a livello temporale è stata la trasmissione e la ricezione dei comandi e dello stato dei sensori tra il microcontrollore lato FPGA e il microcontrollore presente all’interno della vettura. Dopo una prima fase nel quale si è decisa la tecnica di trasmissione (wireless), si è passati alla scelta e allo studio del modulo più adatto per ottenere questo tipo di comunicazione. Sono state effettuate varie prove e vari test per comprendere al meglio il comportamento del modulo scelto ottenendo alla fine una trasmissione efficace e robusta dei segnali.

Nei capitoli successivi verranno spiegate più in dettaglio le fasi che hanno portato alla realizzazione del sistema. Si partirà da una descrizione generale dell’architettura; si descriveranno tutti i componenti utilizzati dal punto di vista hardware, le specifiche di ciascun componente e il perché della loro scelta. Dopodiché si passerà alla descrizione del firmware integrato nei vari componenti elettronici utilizzando principalmente schemi a blocchi, diagrammi di flusso e macchine a stati finiti. Per ciascun componente si spiegheranno inoltre le strategie di fault-tolerance utilizzate al fine di ottenere un sistema affidabile. Infine verranno elencati i possibili sviluppi futuri del sistema realizzato.

1 ARCHITETTURA HARDWARE

L'architettura del sistema è rappresentata nella Figura 1.1:

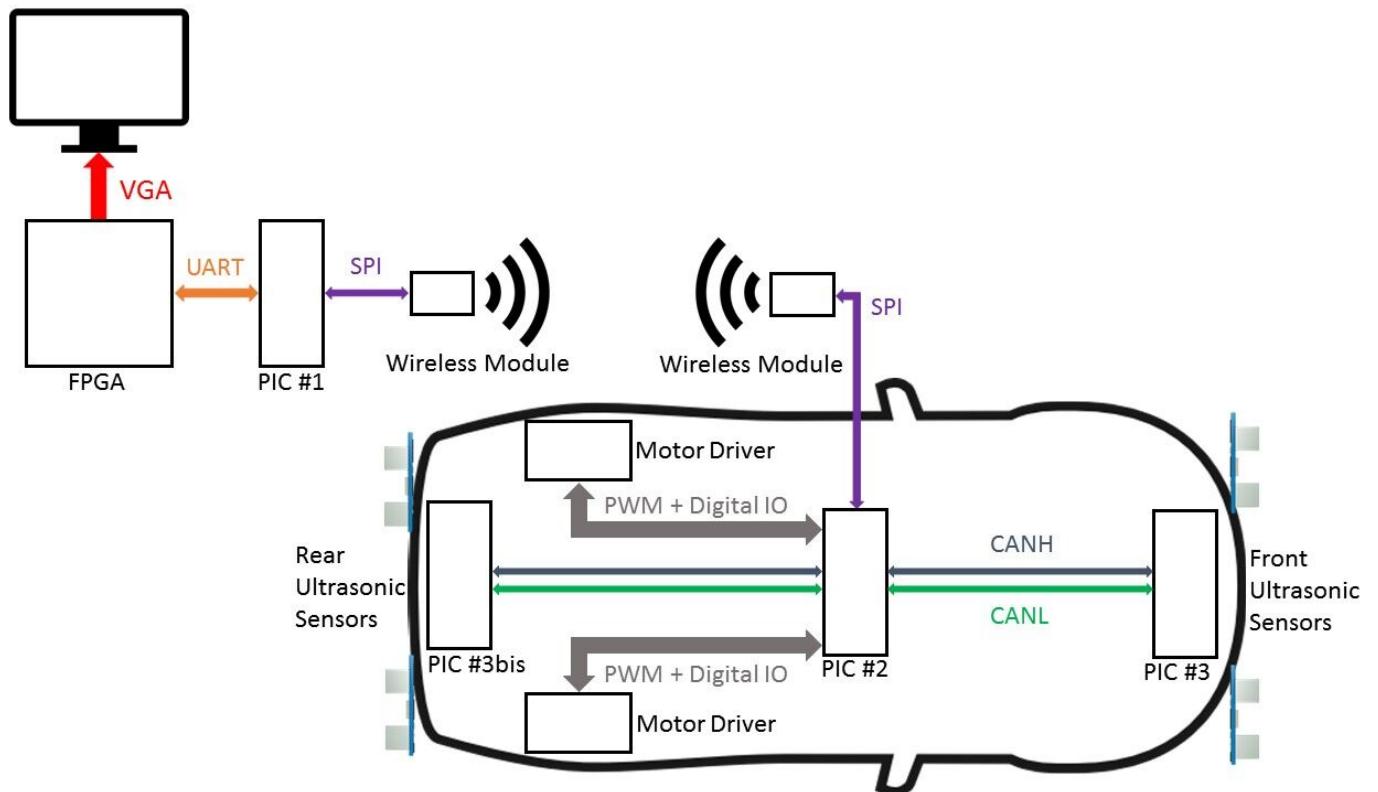


Figura 1.1: architettura del sistema UniMoRover.

Nella sezione successiva verranno descritti i ruoli principali di ogni microcontrollore.

1.1 MICROCONTROLLORI

UniMoRover sfrutta diversi microcontrollori, ognuno dei quali è responsabile di determinate funzioni scelte a seconda del carico computazionale e delle tempistiche necessarie per la risposta. I dispositivi sono stati scelti in modo da rispettare le specifiche elettroniche e di progetto. In seguito si entrerà nel dettaglio per ogni singolo PIC utilizzato.

1.1.1 PIC#1

Questo microcontrollore deve svolgere i seguenti task:

- scambio di pacchetti tramite UART (*Universal Asynchronous Receiver-Transmitter*) con l’FPGA;
- invio e ricezione di pacchetti con il veicolo, utilizzando il modulo nRF24L01. La programmazione e lo scambio di dati avviene tramite protocollo SPI (*Serial Peripheral Interface*).

Si è deciso di sfruttare la tensione di lavoro della FPGA (3.3V) per alimentare questo PIC. La scelta è ricaduta sul *PIC18LF25K80*: la versione LF permette al microcontrollore di lavorare ad una tensione inferiore ai 5V. Questo PIC presenta 28 pin, i quali risultano sufficienti per implementare le funzioni precedentemente descritte.

Lo schema elettrico della scheda del PIC#1 è riportato in Figura 1.2. Si è scelto di inserire il connettore JP2 in modo da permettere la programmazione on-board. Il connettore JP4 viene utilizzato per connettere il modulo wireless Nordic nRF24L01 alla scheda.

1.1.2 PIC#2

Questo microcontrollore deve svolgere i seguenti task:

- gestione della logica per la ricezione dei messaggi sul bus CAN (*Controlled Area Network*) da parte di due diversi PIC;
- gestione della logica per invio e ricezione dei messaggi con l’FPGA utilizzando il Nordic nRF24L01 pilotato tramite protocollo SPI;
- controllo dei motori tramite i messaggi provenienti dall’FPGA e tramite lo stato dei sensori ad ultrasuoni HC-SR04.

Viste le numerose funzioni da implementare su questo microcontrollore, la scelta è ricaduta sul *PIC18F45K80*: questo PIC presenta 40 pin ed è alimentato a 5V.

La presenza di un numero maggiore di I/O è dovuta alle numerose connessioni presenti su questa scheda, la quale svolge il ruolo di unità centrale del sistema UniMoRover.

Lo schema elettrico della scheda del PIC#2 è riportato in Figura 1.3. I driver dei motori richiedono l'utilizzo di 6 porte: una per la modulazione PWM (*Pulse Width Modulation*), necessaria per controllare la velocità del motore; un ingresso per ADC e 4 I/O per la gestione della rotazione.

I segnali entranti negli ADC sono precedentemente condizionati tramite un amplificatore operazionale (*LM358*). Anche in questa scheda sono presenti i connettori per la programmazione on-board (JP7) e per la connessione con il modulo wireless (JP4).

La linea CAN è implementata grazie all'utilizzo del transceiver *MCP2551*: questo componente permette la conversione dei segnali da single-ended a differenziale e viceversa, adattandone i livelli di tensione. Un buzzer (SG1) è implementato in modo da fornire un avviso acustico in caso di presenza di ostacoli e/o retromarcia.

1.1.3 PIC#3 e PIC 3#bis

Questi microcontrollori devono svolgere il seguente task:

- pilotaggio dei sensori ad ultrasuoni *HC-SR04* anteriori per il rilevamento della distanza e invio di byte di messaggio riguardanti lo stato dei sensori sul bus CAN.

Si è scelto di utilizzare il *PIC18F25K80*: questo PIC presenta 28 pin ed è alimentato a 5V.

Lo schema elettrico della scheda dei PIC#3 e PIC#3bis è riportato in Figura 1.4.

Ognuna di queste schede presenta due sensori di prossimità, utilizzati per la rilevazione della distanza degli ostacoli: la connessione è effettuata mediante i connettori JP3 e JP2.

I dati provenienti dai sensori sono analizzati dal PIC e vengono trasferiti alla centralina tramite bus CAN: per questo motivo è stato inserito il transceiver *MCP2551*.

La programmazione on-board è mantenuta tramite il connettore JP4.

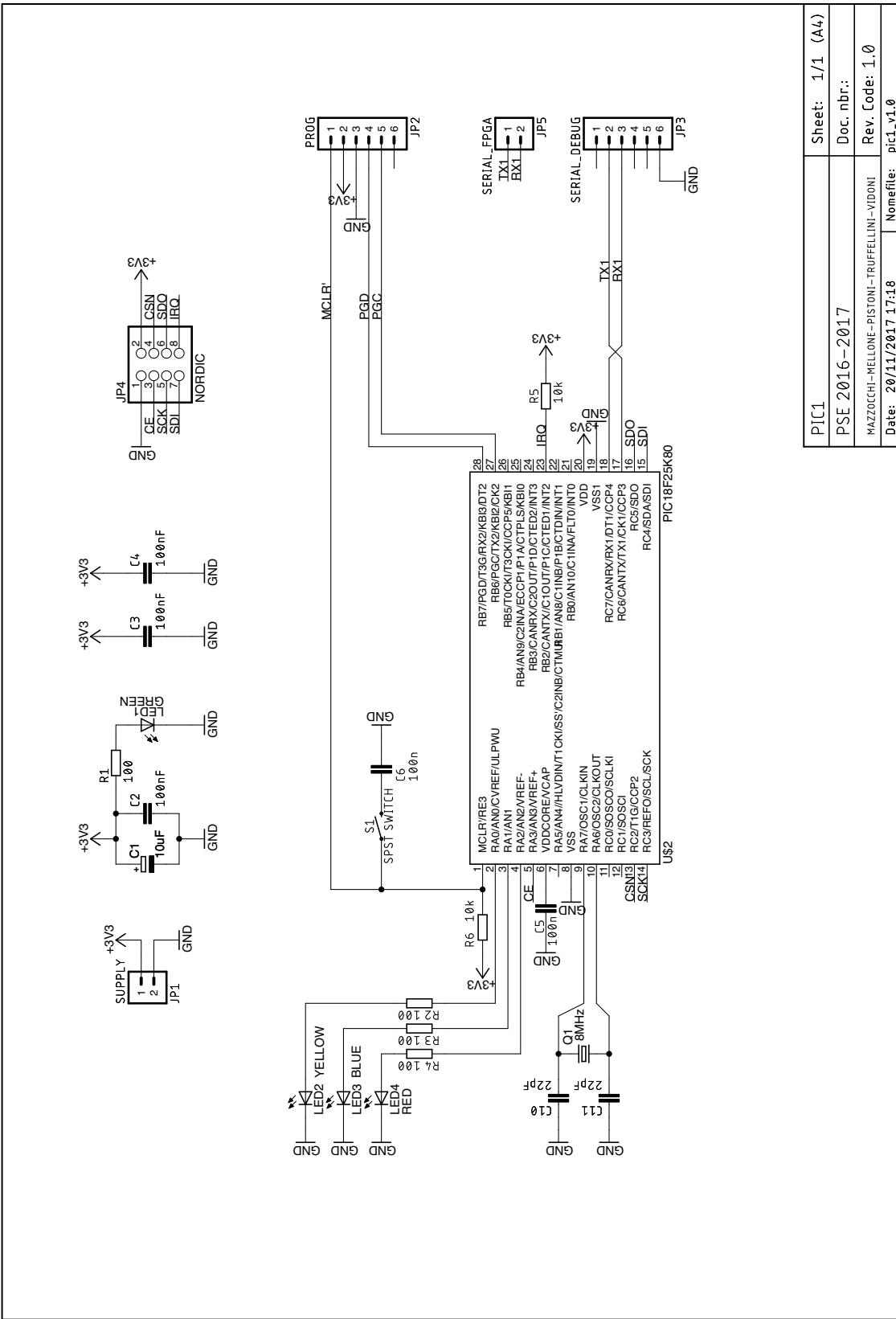


Figura 1.2: schema elettrico della board PIC#1.

PIC1	Sheet: 1/1 (A4)
PSE 2016-2017	Doc. nbr.:
MATZOCCHI - MELLONE - PISTONI - TRUFFELINI - VIDONI	Rev. Code: 1.0
Date: 20/11/2017 17:18	Nomefile: pic1_v1.0

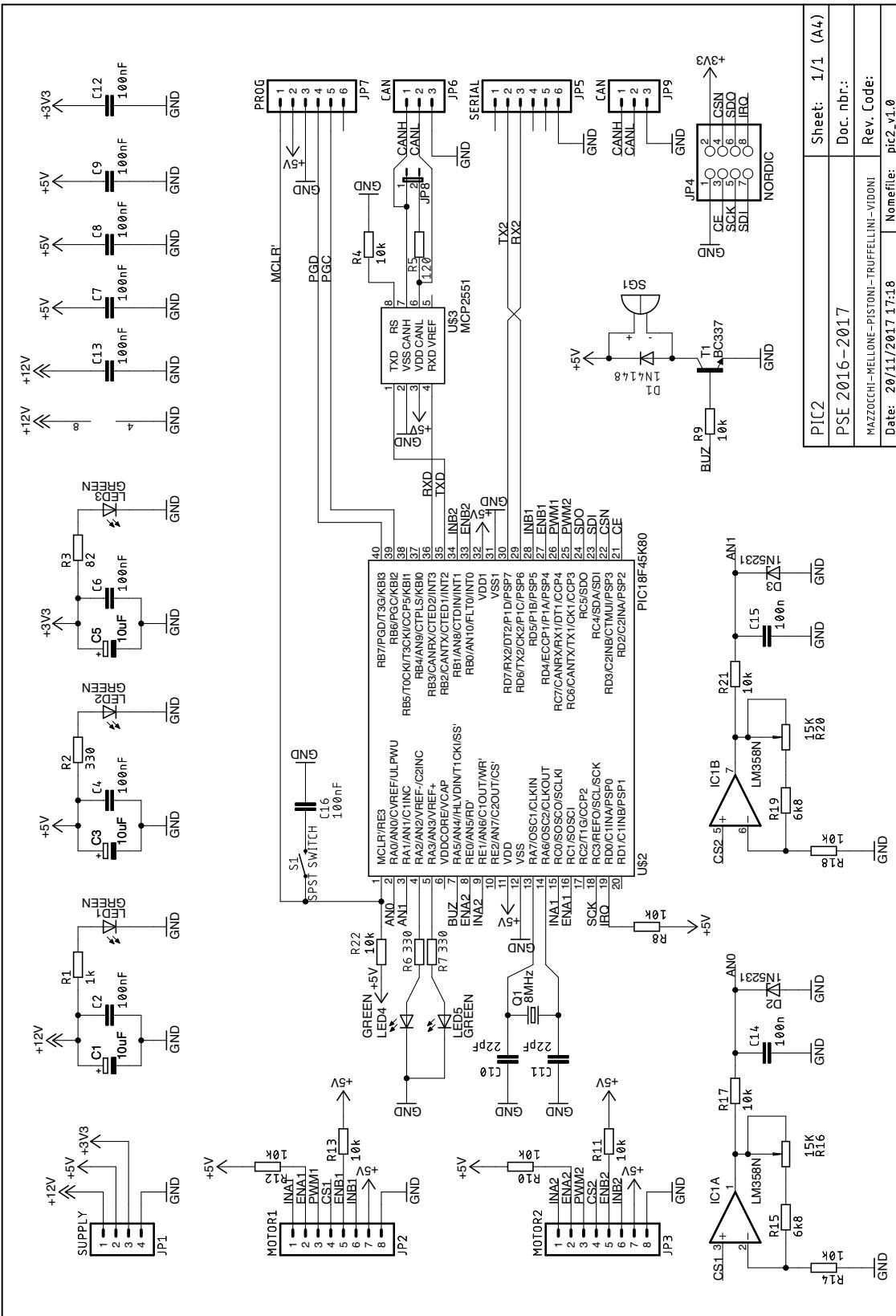


Figura 1.3: schema elettrico della board PIC#2.

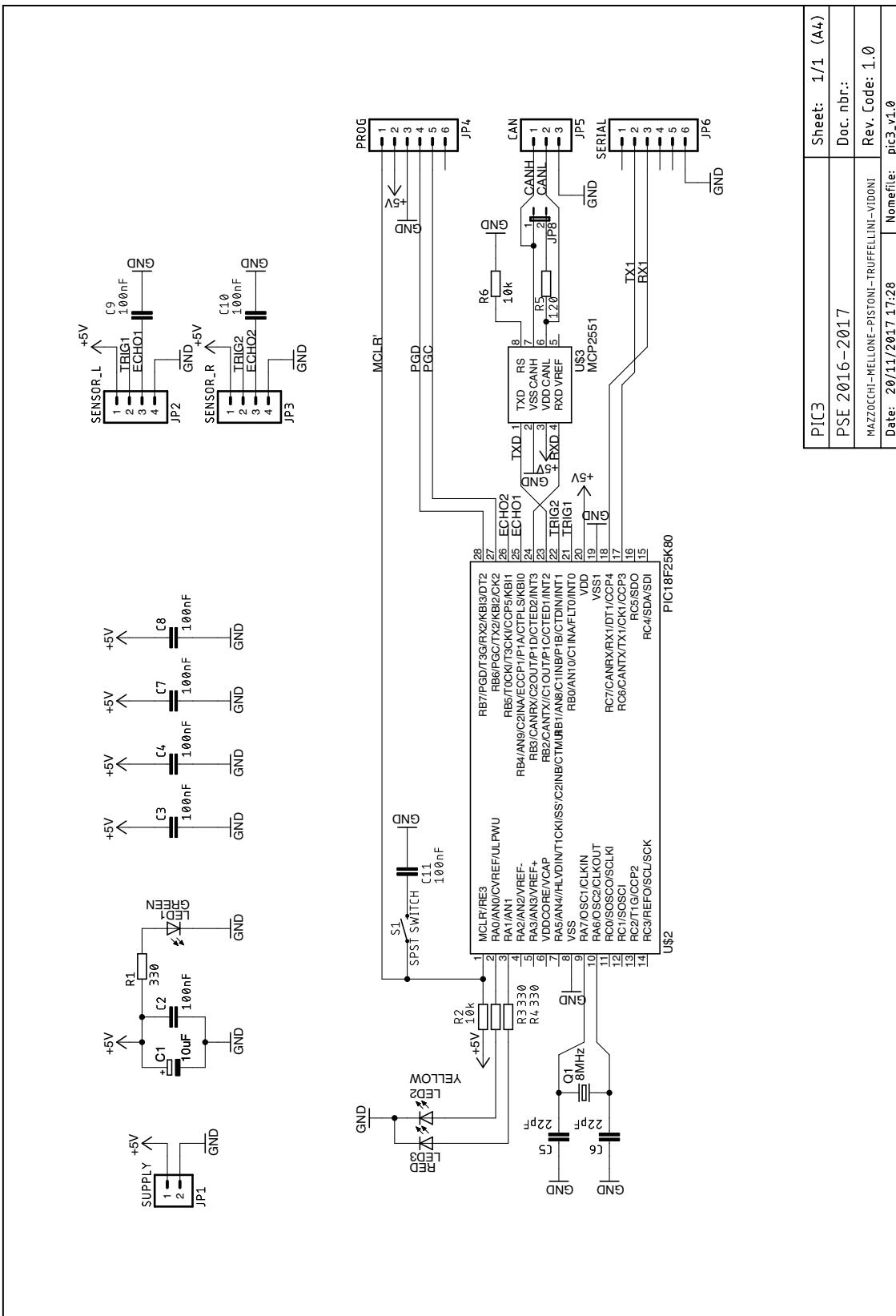


Figura 1.4: schema elettrico della board PIC#3/PIC#3bis.

PLC3	Sheet: 1/1 (A4)
PSF 2016-2017	Doc. nbr.:
MATZOCCHI-MELLONE-PISTONI-TRUFFELLINI-VIDONI	Rev. Code: 1.0
Date: 20/11/2017 17:28	Namefile: pic3_v1.0

1.2 FPGA (FIELD PROGRAMMABLE GATE ARRAY)

Le FPGA (*Field Programmable Gate Array*) sono dispositivi programmabili direttamente dall'utente costituiti da un array di componenti logici collegabili tra loro.

Le FPGA mettono a disposizione:

- Componenti logici: costituiti da porte logiche, Flip-Flop, Buffer ecc. che offrono differenti prestazioni a seconda del numero di connessioni e di complessità dei componenti scelti.
- Linee di connessione locale e distribuite: la cui lunghezza modella parametri di ritardo e potenza.

Queste schede sono così utilizzate per i vantaggi offerti in termini di compromessi rispetto alle altre attuali tecnologie, essendo la migliore scelta per costi, flessibilità e prestazioni.

Un flusso di sviluppo tradizionale prevede che la simulazione del comportamento del circuito avvenga tramite software, permettendo di realizzare e simulare dispositivi in modo rapido (fast prototyping) e di emulare i dispositivi realizzati come insieme di logica generica.

Nel lavoro presente è stata adottata una scheda Nexys3 equipaggiata con una FPGA *Xilinx Spartan-6 LX16* e un ambiente di sviluppo *ISE Design Suite 14.7*.

I file di configurazione dell'FPGA sono originati da tool integranti tutte le risorse necessarie per progettare, modificare e generare configurazioni di FPGA utilizzando le più diverse metodologie di progetto (linguaggio VHDL e Verilog, schemi circuitali, metodi di tipo grafico o basati su blocchi).

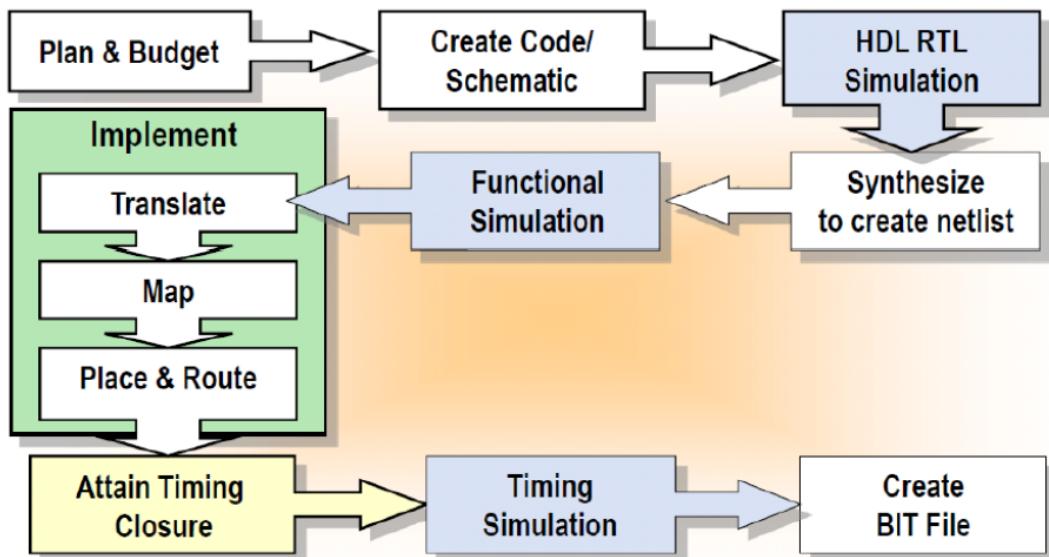


Figura 1.5: sequenza temporale di programmazione su FPGA.

Nella Figura 1.5 viene rappresentata la sequenza temporale e le varie fasi a cui va sottoposto il progetto prima di poter essere effettivamente caricato ed utilizzato sulla scheda.

Partendo dalle specifiche che sono richieste e dal budget messo a disposizione, si decide che tipo di approccio avere e la complessità del progetto che si vuole realizzare. I compromessi sono cruciali: più aumenta la complessità del progetto, maggiori saranno i ritardi e i costi.

Successivamente si crea uno schema a blocchi per capire quali sono le componenti principali che si dovranno andare a creare per implementare il progetto e si scrive il codice che corrisponde ai vari blocchi, connettendoli infine tra loro.

Terminata la fase progettuale si passa alla sintesi e alla creazione di una prima rete di connessioni. Si simula quindi la logica del progetto per verificare che il comportamento che esso assume sia corretto e si passa all'implementazione. Si ottiene ora un modello utile a simulare il reale comportamento della scheda, verificando ritardi e specifiche di progetto; se si è soddisfatti si può infine generare il file *.bit che servirà a programmare la scheda.

Al termine della fase di progetto si otterrà un file contenente uno “stream” (flusso) di bit che andrà poi caricato direttamente sull’FPGA o nella PCM della scheda e che servirà a programmare tutte le strutture contenute nel FPGA.

Il setup hardware è riportato in figura:

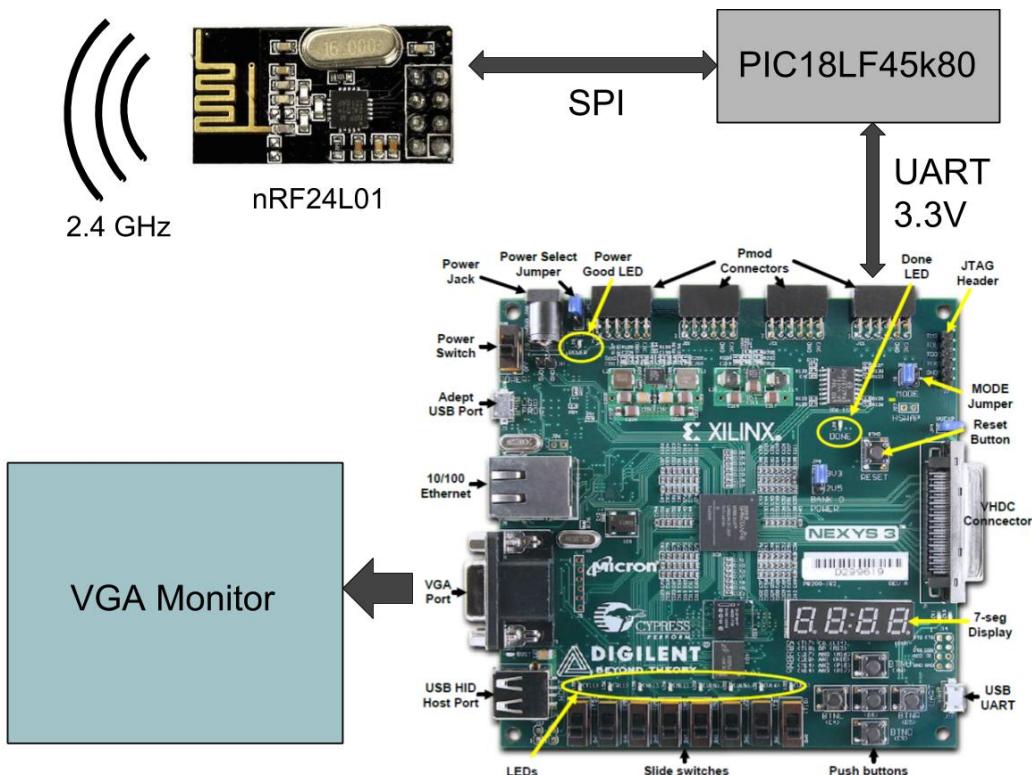


Figura 1.6: Xilinx Nexys3 setup hardware.

L'idea è di sfruttare l'FPGA come unità di controllo del rover: i comandi vengono inviati premendo la pulsantiera a 5 tasti (pulsante centrale usato come clacson), in contemporanea vengono ricevuti pacchetti usati per segnalare lo stato dei sensori e dei motori a bordo e la loro condizione viene visualizzata a schermo sfruttando l'uscita VGA (*Video Graphics Array*) che aggiorna dinamicamente una semplice interfaccia grafica rendendo cosciente l'utente dello stato del rover.

Nel dettaglio, la FPGA dovrà effettuare le seguenti operazioni:

- stampare a monitor lo stato dei sensori del veicolo e della pulsantiera dei comandi, utilizzando l'uscita VGA;
- comunicare con il PIC #1 tramite l'interfaccia UART per trasmettere i comandi e ricevere byte ascii di status dei sensori di prossimità.

1.3 DRIVER MOTORI DC

Il pilotaggio di due motori brushed posti sulle ruote posteriori permette il movimento del veicolo.

In principio essi venivano attivati simultaneamente tramite una corrente continua, proveniente direttamente dalla batteria: non era permessa la variazione della velocità dei singoli motori, requisito fondamentale per rispettare le specifiche del nostro progetto.

Per ovviare a questo problema si è deciso di optare per un controllo tramite modulazione PWM (*Pulse Width Modulation*). La corrente che eccita i motori viene generata tramite un'onda quadra con duty cycle variabile: la componente continua che fornisce energia ai motori varia a seconda del valore D impostato, permettendo l'accelerazione del veicolo.

Questa tecnica richiede l'implementazione di un modulo elettronico, gestito tramite il PIC scelto precedentemente, in grado di supportare alti livelli di corrente in uscita.

1.3.1 SPECIFICHE DEL MODULO

Le specifiche del modulo riguardano principalmente la tensione di alimentazione e la massima corrente disponibile in uscita; quest'ultima è stata valutata inserendo un multimetro in serie al cablaggio di ognuno dei due motori e fornendo una corrente continua tramite la batteria del veicolo: senza considerare i transitori, dove si verificavano picchi elevati di corrente dovuti alla necessità del motore di vincere gli attriti e l'inerzia, il valore massimo di corrente valutato era circa 6A. L'alimentazione è ottenuta da una batteria al piombo a 12V.

I requisiti di questo modulo possono essere così riassunti:

- $I_{out} \text{ max} > 10\text{A};$
- $V_{cc} = 12\text{V};$
- Comando tramite segnali PIC18F25K80;
- Protezione contro cortocircuito e inversione di polarità.

1.3.2 SCELTA DEL MODULO

Si sono valutati alcuni moduli plug-and-play presenti sul mercato, ma essi sono risultati sconvenienti dal punto di vista del rapporto prestazioni/costi. Si è quindi optato per la realizzazione home-made della circuiteria necessaria, composta principalmente da quattro switch in configurazione full-bridge e la conseguente elettronica di controllo.

Inizialmente si sono cercati MOSFET di potenza adatti a rispettare le specifiche di output richieste. Nonostante questo approccio garantisse un'elevata flessibilità nella scelta dei componenti, l'elettronica necessaria per il controllo dei singoli transistor introduceva alcuni problemi, tra i quali la minima carica necessaria fornita dal gate driver per la commutazione.

La scelta finale è quindi ricaduta sull'utilizzo di un chip, integrante direttamente la parte di potenza e il suo controllore. Sono state analizzate principalmente tre alternative presenti sul listino *STMicroelectronics*, le cui caratteristiche sono riportate nella Tabella 1.1.

	VNH3SP30	VNH2SP30	VNH5019
Operating voltage	5.5 – 16 V	5.5 – 16 V	5.5 – 24 V
MOS R_{on} (per leg)	34 mΩ typ.	19 mΩ typ.	18 mΩ typ.
Max PWM frequency	10 kHz	20 kHz	20 kHz
Current sense	n/a	0.13 V/A typ.	0.14 V/A typ.
Logic input threshold	3.25 V min.	3.25 V min.	2.1 V min.
Current for infinite run-time	9 A	14 A	12 A
Time to overheat at 15A	30 s	150 s	90 s
Time to overheat at 20A	8 s	35 s	20 s

Tabella 1.1: confronto tra i diversi integrati considerati.

Come si può notare dai dati riportati, il componente *VNH5019* è la scelta migliore tra le proposte in quanto compatibile con segnali logici a 3.3V, inoltre la sua R_{on} è la minore ed è capace di lavorare a frequenze molto alte per quanto riguarda il segnale PWM in ingresso.

1.3.3 CIRCUITO E REALIZZAZIONE

La scelta di un chip integrato ha semplificato notevolmente la realizzazione del modulo.

Nella Figura 1.7 si può notare il circuito finale. Al centro è presente il chip *VNH5019*: questo componente necessita di una tensione d'alimentazione per il ponte di MOSFET (chiamata *VBATT*), fornita direttamente dalla batteria del veicolo, dalla quale viene creata la tensione d'alimentazione per l'elettronica interna (chiamata *VCC*) tramite un MOS di protezione (*NCV7808DT*). I condensatori sono inseriti per filtrare eventuali disturbi sulla linea dei 12V.

Tra le due uscite si possono notare due Led, utili in fase di debugging per visualizzare il ramo attivo del ponte, più alcuni condensatori al poliestere, necessari per il corretto funzionamento dei motori elettrici.

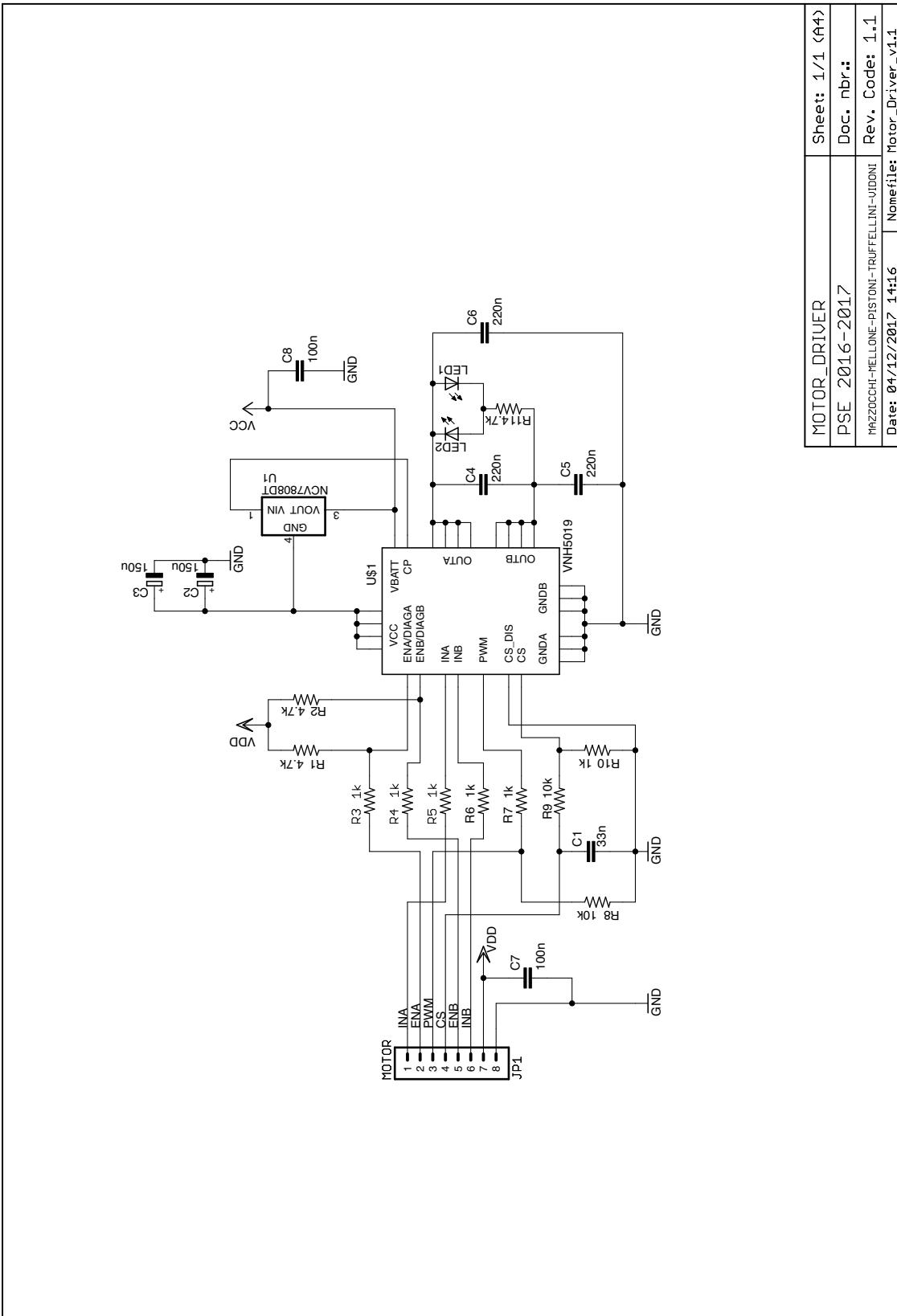


Figura 1.7: schema elettrico della board driver motore DC.

Gli ingressi *INA/INB* vengono attivati per indicare il verso di rotazione voluto (vedi Tabella 1.2).

IN_A	IN_B	EN_{A/DIAG_A}	EN_{B/DIAG_B}	OUT_A	OUT_B	CS	Operating Mode
0	0	1		L	L	High imp.	Brake to GND
0	1	1		L	H	$I_{sense} = I_{out}/K$	Clockwise (CW)
1	0	1		H	L	$I_{sense} = I_{out}/K$	Counterclockwise (CCW)
1	1	1		H	H	High imp.	Brake to V _{CC}

Tabella 1.2: segnali di controllo del chip VNH5019

Quando la loro polarità è uguale, il ponte non permette il passaggio di corrente all'interno degli avvolgimenti del motore e quindi il rotore sarà fermo.

I due segnali di Enable (*ENA/ENB*) sono considerati come input in condizioni operative normali: qualora si presentasse un guasto, *VNH5019* li utilizza come output di diagnosi e li pone a massa. Per questo motivo questi due segnali possiedono un pull-up esterno. Il pin *CS* è utilizzato come output: esso fornisce un valore di tensione continua proporzionale alla corrente che il chip sta erogando in quell'istante in uno dei suoi rami. Questa funzione, detta *Current Sense*, può essere sfruttata per monitorare le correnti nei due motori e attuare politiche di fault-tolerance.

Si è quindi realizzato il circuito stampato tramite fotoincisione. Il circuito integrato utilizzato viene fornito con package *MultiPowerSO-30*, la cui struttura è mostrata in Figura 1.8: in blu sono evidenziate le connessioni (*pad*) di potenza degli output e di alimentazione, posti sotto il chip.

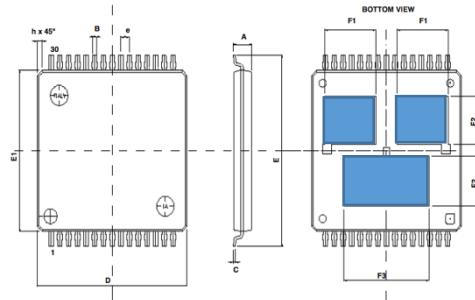
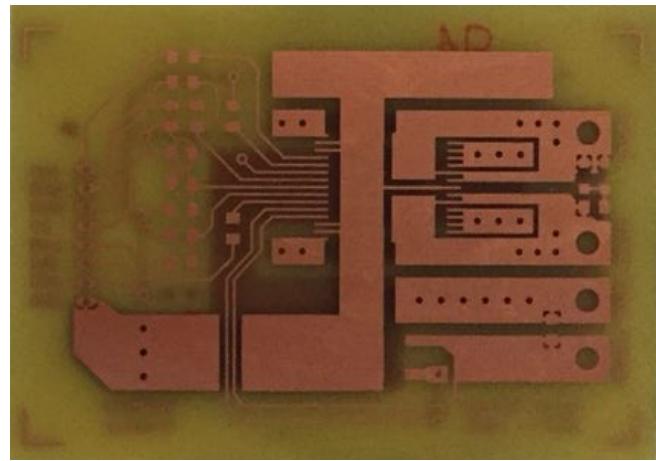


Figura 1.8: board driver motore DC e connessioni dell'integrato utilizzato.

Il package del VNH5019 non rende possibile la saldatura diretta sulla board realizzata: questo problema è stato risolto mediante l'utilizzo di un apposito macchinario per saldature ad aria calda, seguendo i gradienti di temperatura forniti dal datasheet del componente (Figura 1.9).

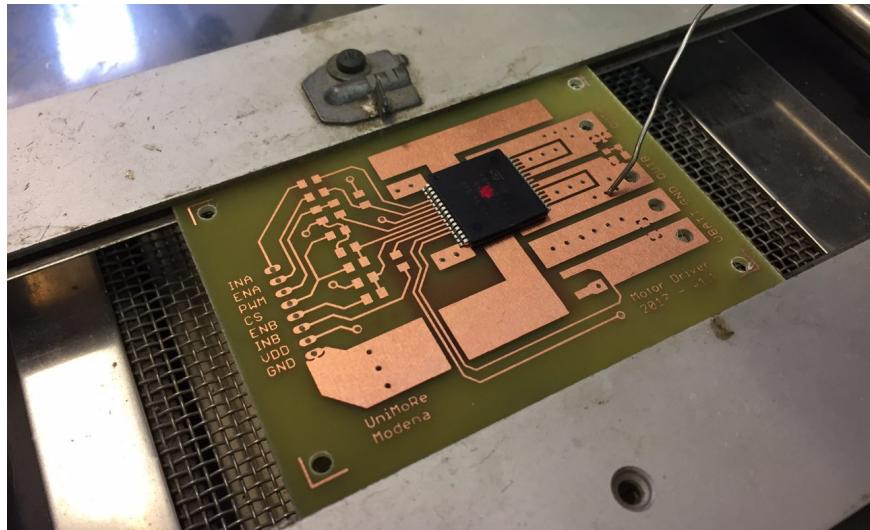


Figura 1.9: fase di saldatura dell'integrato su PCB.

Ogni saldatura è stata controllata al fine di ridurre spiacevoli falsi contatti e/o cortocircuiti (Figura 1.10)

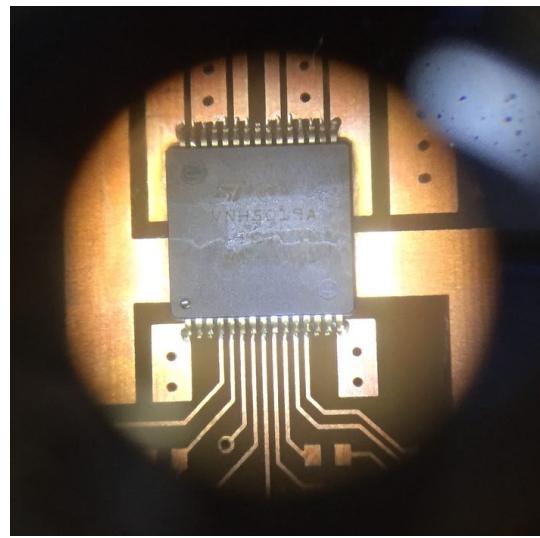


Figura 1.10: controllo delle connessioni su PCB tramite microscopio.

1.3.4 TEST DEL CIRCUITO

Il circuito stampato è stato sottoposto ai seguenti test prima di essere definito completamente funzionante:

- Test prolungato a carico e duty cycle variabili;
- Test over-temperature sotto carico.

Nella prima prova si sono verificate le funzionalità del modulo sottponendolo ad un segnale con duty cycle variabile, in presenza di diverse configurazioni di carico resistivo, formato da alcune lampadine a incandescenza collegate in parallelo (Figura 1.11). L'alimentazione è stata fornita da un alimentatore da computer per evitare di scaricare la batteria del veicolo.

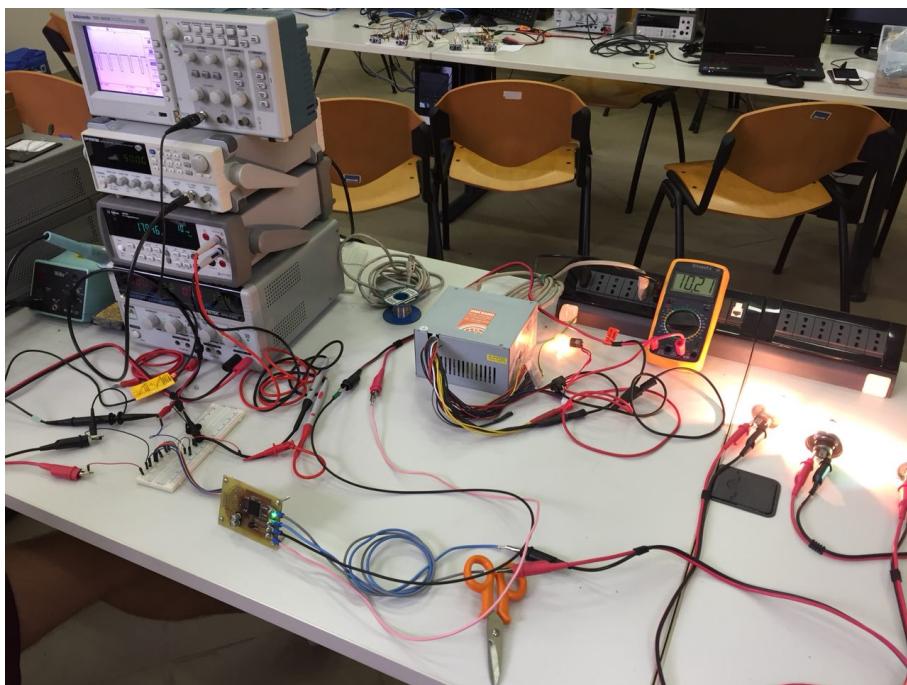


Figura 1.11: setup test a carico e duty cycle variabile.

Durante il test si sono raggiunti valori prossimi ai 14A di corrente continua massima, con duty cycle D=80%, utilizzando come carico 3 lampadine. Anche in queste condizioni il modulo ha funzionato correttamente, senza attuare nessuna politica di protezione nonostante le temperature in gioco.

Nel secondo test si è verificata la capacità del modulo di lavorare per tempi prolungati con correnti elevate, situazione analoga ad un percorso rettilineo da effettuare a velocità massima. In questa prova il circuito è stato alimentato per cinque minuti e gli è stato fornito un duty cycle fisso con valore D=0.8. Lo stress test è stato superato senza riscontrare anomalie e il modulo non è mai entrato in *Over-temperature protection*.

1.3.5 COSTANTE DI PROPORZIONALITÀ DEL CURRENT SENSE

La tensione fornita dal modulo nel suo output CS, proporzionale alla corrente circolante nel ponte di transistor, è filtrata tramite un filtro passa basso in modo da eliminare possibili disturbi ad alta frequenza e ridurne il ripple. La costante di proporzionalità K è stata valutata tramite test sperimentali: i valori di corrente e delle rispettive tensioni CS sono stati acquisiti per tre diverse configurazioni di carico (singola lampadina, a due o tre lampadine).

Il valore di K è stato estrapolato eseguendo la media sui valori di proporzionalità nelle tre configurazioni. Il risultato ottenuto è il seguente:

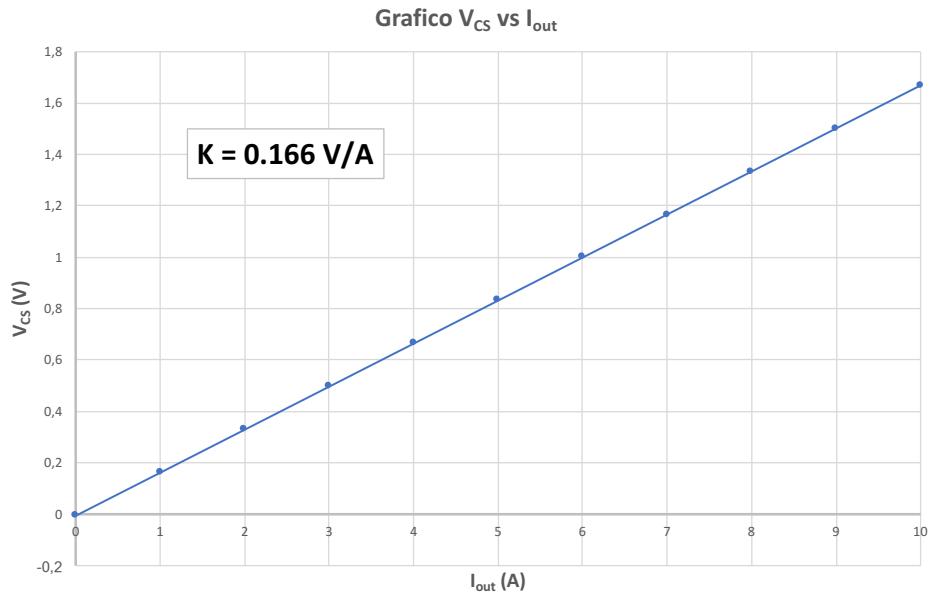


Figura 1.12: caratterizzazione della costante di proporzionalità del current sense.

1.4 MODULO WIRELESS: NORDIC nRF24L01

La comunicazione FPGA – Rover, per invio e ricezione dei comandi e dello stato dei sensori, deve necessariamente avvenire via wireless.

Sono state valutate diverse opzioni a seconda dei costi, della complessità e del numero di protocolli e livelli di reti coinvolti. La scelta è ricaduta sul Nordic *nRF24L01* sia per i costi ridotti sia perché adotta un protocollo di comunicazione proprietario “*Enhanced ShockBurst*” che lavora a 2.4GHz e non ha bisogno di utilizzare protocolli Wi-Fi di livello superiore che renderebbero tutto molto più complesso e macchinoso.

Le caratteristiche chiave sono elencate di seguito:

- 2.4GHz ISM banda operativa;
- Fino a 2Mbps di velocità di trasferimento dei dati via aria;
- Consumi ridotti;
- 11.3mA TX a 0dBm di potenza;
- 12.3mA RX a 2Mbps di velocità di trasferimento;
- 900nA quando è in modalità “power down”;
- 22µA quando è in modalità “Standby”;
- On chip voltage regulator;
- Possibilità di essere alimentato da 1.9 a 3.6V;
- Impacchettamento automatico dei dati ed invio tramite Enhanced ShockBurst;
- Gestione automatica dei pacchetti in ricezione;
- Comunicazione multipla fino a 6 moduli;
- Input 5V tolerant.

Tra i numerosi vantaggi c’è quindi l’impacchettamento e la lettura automatica dei dati una volta che TX e RX sono stati programmati correttamente. Il programmatore non deve così preoccuparsi di come avviene la trasmissione via aria né di questioni come CRC, tutto viene gestito in maniera automatica una volta programmato.

Di contro l’utilizzo di questi moduli ha suscitato non poche complicazioni dovute alla difficoltà nell’eseguire il debugging.

I moduli non sono infatti dotati di alcun led che potrebbe indicare la corretta alimentazione, programmazione, trasmissione o ricezione dei dati. Dovendo inoltre lavorare in coppia TX e RX è necessario che entrambi i moduli siano funzionanti e programmati specularmente perché avvenga lo scambio dei dati. Ultimo problema, ma non per questo meno importante, i moduli possono essere programmati solo tramite SPI (*Serial Peripheral Interface*) di cui, come verrà spiegato brevemente nel firmware, è stato necessario comprendere quali impostazioni adottare per fare in modo di scrivere sulla memoria del *nRF24L01* in modo corretto. Ciò è stato possibile sia tramite un processo di reverse engineering a partire dalle librerie dello stesso modulo trovate per Arduino sia andando “a tentativi”, dato che le informazioni sul datasheet si sono poi rilevate equivoche.

Come precedentemente menzionato, l'SPI è standardizzato ma ci sono diverse configurazioni disponibili, quindi è stato necessario comprendere quali impostazioni fossero necessarie perché avvenisse la comunicazione e scrittura tra PIC e *nRF24L01*.

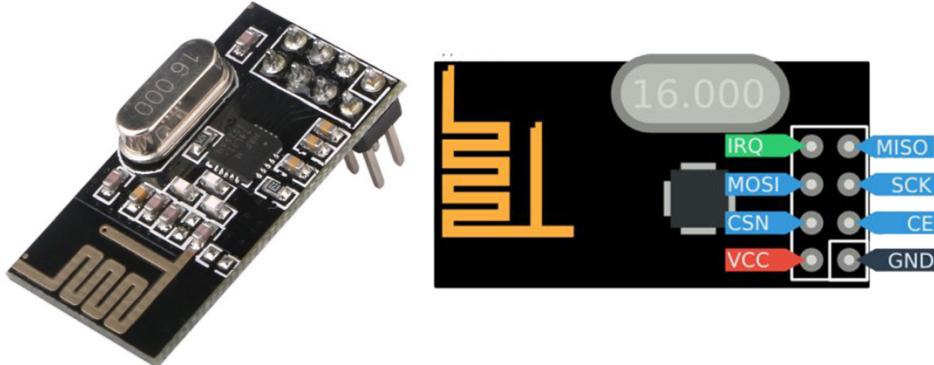


Figura 1.13: Nordic nRF24L01.

In Figura 1.13 è possibile vedere come si presenta il modulo. In Figura 1.14 è riportato lo schema a blocchi preso dal datasheet. Ci sono un totale di 8 pin:

- **IRQ**: Interrupt pin, attivo basso. Serve a segnalare l'avvenuta ricezione/trasmissione di un pacchetto. Programmabile a seconda della funzione che si vuole avere;
- **MISO**: Master Input Slave Output, equivalente del SDI (Serial Data Input);
- **MOSI**: Master Output Slave Input, equivalente del SDO (Serial Data Output);
- **SCK**: Clock;
- **CSN**: SPI chip select, deve essere messo basso prima di ogni operazione di scrittura sulla memoria del modulo per poi essere rimesso a 1 quando finito.

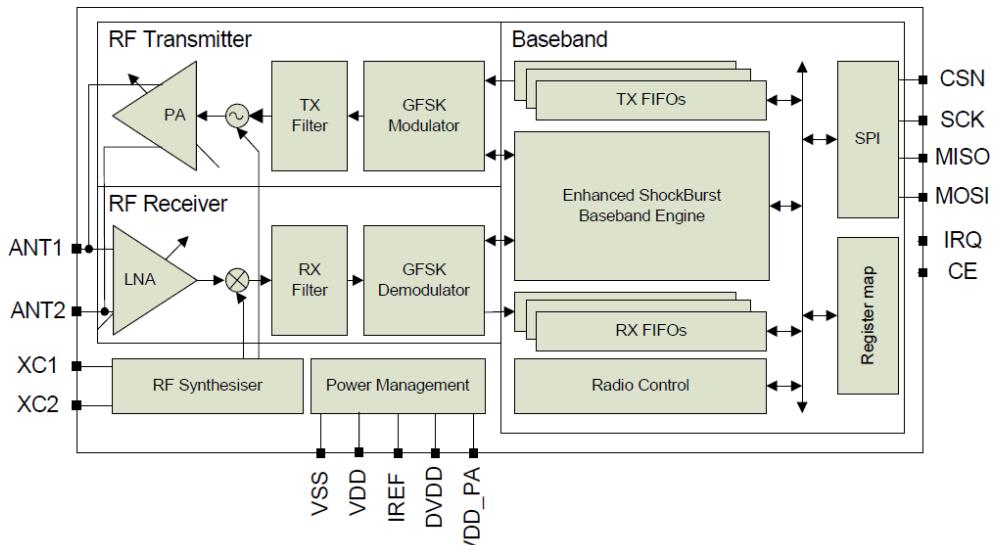


Figura 1.14: schema a blocchi del modulo nRF24L01.

1.5 SENSORI AD ULTRASUONI: HC-SR04

L'HC-SR04 è un sensore ad ultrasuoni utilizzato per il rilevamento delle distanze (Figura 1.15) ed è dotato di 4 pin:

- Vcc, PIN di alimentazione collegato a 5V
- Trig, PIN di input del Trigger;
- Echo, PIN di output dell'Eco;
- GND.



Figura 1.15: sensore HC-SR04.

Le specifiche del modulo sono riportate nella seguente tabella:

Item	Value
Tensione di lavoro	5V
Corrente di lavoro	15mA
Frequenza di lavoro	40kHz
Distanza massima	4m
Distanza minima	2cm
Angolo massimo di misurazione	15°

Tabella 1.3: specifiche del modulo HC-SR04.

Il sensore è dotato di un trasmettitore, il quale invia un segnale di trigger verso l'ostacolo, e da un ricevitore a cui ritorna l'eco (Figura 1.16). Nel frattempo, viene prodotto un impulso che corrisponde al tempo richiesto dall'eco per ritornare al sensore. Misurando l'ampiezza dell'impulso dell'eco è possibile calcolare la distanza dall'ostacolo. Trig ed Echo sono i pin utilizzati per interfacciare il modulo con il microcontrollore.

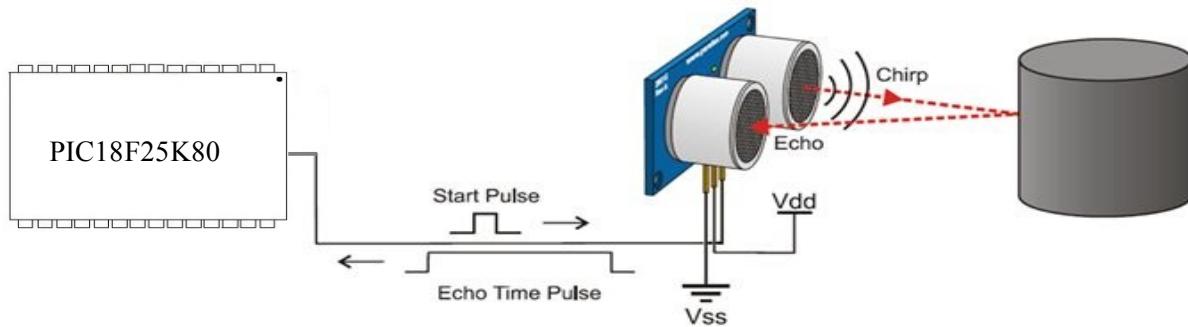


Figura 1.16: principio di funzionamento.

Il principio di funzionamento è riportato in Figura 1.16.

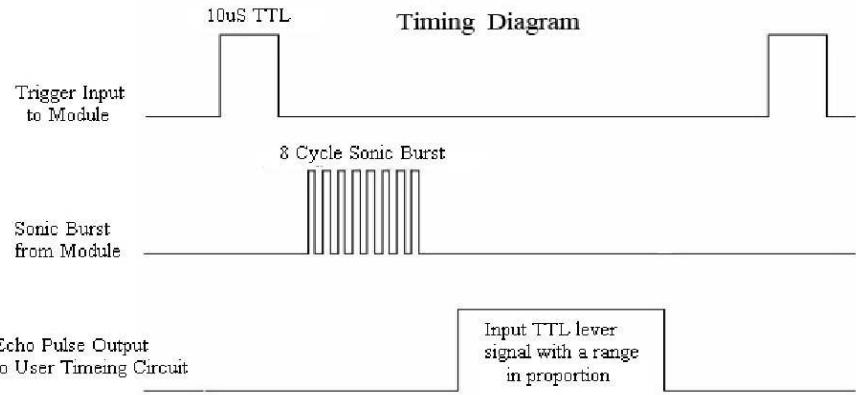


Figura 1.17: diagramma temporale dei segnali.

Per iniziare il rilevamento, si fornisce all'input TRIGGER un breve impulso di 10us (Figura 1.17), il modulo spedirà automaticamente un burst di ultrasuoni composto da 8 cicli a 40kHz e rileverà se è presente un segnale di ritorno, ovvero il segnale di eco.

Se è presente un segnale di ritorno, l'output Echo del sensore si troverà in uno stato alto (5V). Sarà quindi possibile calcolare la distanza dall'ostacolo attraverso l'intervallo di tempo che intercorre tra l'invio del segnale di trigger e la ricezione del segnale di eco tramite la seguente formula:

$$range = \frac{high\ level\ time * velocity\ of\ sound}{2}$$

In questo caso, la velocità del suono viene approssimata a 340 m/s.

2 FIRMWARE

Dopo aver scelto i vari componenti necessari per soddisfare le specifiche di progetto, è stato sviluppato il firmware che permette di sfruttare le funzionalità dell'hardware.

Per quanto riguarda l'FPGA, inizialmente è stata disegnata la sagoma del rover per la visualizzazione a schermo, mentre i segnali di allerta dei sensori sono stati simulati tramite gli interruttori integrati nella scheda. I comandi di movimento del veicolo vengono azionati tramite la pulsantiera e trasmessi all'unità centrale presente nel rover attraverso una comunicazione wireless.

Una volta stabilito il funzionamento dell'FPGA, l'attenzione si è spostata verso i sensori ad ultrasuoni; per una maggiore precisione, è stato assegnato un valore esadecimale in base alla distanza e alla posizione, creando così un alfabeto univoco utilizzato per tutto il progetto.

I dati provenienti dai sensori vengono comunicati all'unità centrale da un microcontrollore collegato ad essa tramite bus CAN. Poiché non esistono messaggi più importanti di altri, tutti i pacchetti dati avranno la stessa priorità. La fase iniziale della progettazione del bus CAN si è basata sulla trasmissione dei dati da parte del microcontrollore ad un CAN Sniffer. Verificato il corretto funzionamento, si è passati alla progettazione del firmware per la ricezione dei dati su un microcontrollore esterno, implementato successivamente nell'unità centrale.

I valori ricevuti dai sensori, uniti ai comandi inviati dall'FPGA, vengono usati dall'unità centrale per comandare il moto del rover; nel caso in cui i sensori avvertano la presenza di un ostacolo nelle vicinanze, l'unità centrale invierà un comando di fermata ai driver motori per permettere l'arresto del veicolo.

Nei sottocapitoli successivi verrà descritto nei dettagli il firmware di ciascun componente del sistema.

2.1 FPGA

Il software è stato diviso in due macro blocchi: il primo inerente il driver della VGA, con al suo interno anche il timing necessario per visualizzare sul display a 7 segmenti i comandi inviati, usando gli switch come strumenti di debug per simulare gli alert dei sensori e dell'avarie motore; il secondo macro blocco consiste nella gestione del pacchetto fornito “uart.vhd” usando due macchine a stati finiti, una per la ricezione e l'altra per la trasmissione dei pacchetti con il PIC#1, collegato a sua volta con il Nordic *nRF24L01* usato per la comunicazione wireless.

Successivamente si andranno ad analizzare singolarmente e in maggior dettaglio i due macro blocchi.

2.1.1 VIDEO GRAPHICS ARRAY (VGA)

VGA è uno standard utilizzato per la trasmissione di questi segnali sullo schermo, inventato da IBM nel 1987.

La scheda grafica integrata nell’FPGA permette di visualizzare immagini su schermo. Il monitor è rappresentato da una memoria costituita da un determinato numero di pixel. Ogni pixel ha una componente rossa, verde e blu e la scheda grafica è la responsabile della trasmissione di questi pixel attraverso segnali analogici. Inizialmente lo standard prevedeva una risoluzione di 640x480 pixel, a cui si farà riferimento anche nelle immagini a seguire. Nel presente lavoro è stato invece adottato un monitor avente risoluzione 1368x768: nonostante cambino i valori per la sincronizzazione, il principio di funzionamento rimane invariato.

Nella Tabella 2.1 si riportano i valori di timing dei due monitor, uno 640x480 preso come esempio e l’altro 1368x768 effettivamente implementato nella scheda FPGA.

640 x 480 @ 60 Hz		1368 x 768 @ 60 Hz	
General Timing		General Timing	
Screen refresh rate	60 Hz	Screen refresh rate	60 Hz
Vertical refresh	31.4 kHz	Vertical refresh	47.7 kHz
Pixel freq.	25.1 MHz	Pixel freq.	85.86 MHz
Horizontal Timing (line)		Horizontal Timing (line)	
Scanline part	Pixels	Scanline part	Pixels
Visible area	640	Visible area	1368
Front porch	16	Front porch	72
Sync pulse	96	Sync pulse	144
Back porch	48	Back porch	216
Whole line	800	Whole line	1800
Vertical Timing (frame)		Vertical Timing (frame)	
Frame part	Lines	Frame part	Lines
Visible area	480	Visible area	768
Front porch	10	Front porch	1
Sync pulse	2	Sync pulse	3
Back porch	33	Back porch	23
Whole frame	525	Whole frame	795

Tabella 2.1: valori di sincronizzazione per monitor con due diverse risoluzioni.

Il controller VGA (integrato nel monitor) genera due rampe di comando (verticale e orizzontale) partendo dai segnali HSync e VSync che sono forniti alla porta dal dispositivo che la comanda (Figura 2.1), assieme ai tre segnali RGB analogici che di volta in volta danno le informazioni di colore del singolo pixel.

Si definiscono tutti i pixel uno ad uno partendo dall'angolo in alto a sinistra (0,0) fino a quello in basso a destra (479, 639), intendendo lo schermo come una matrice di pixel.

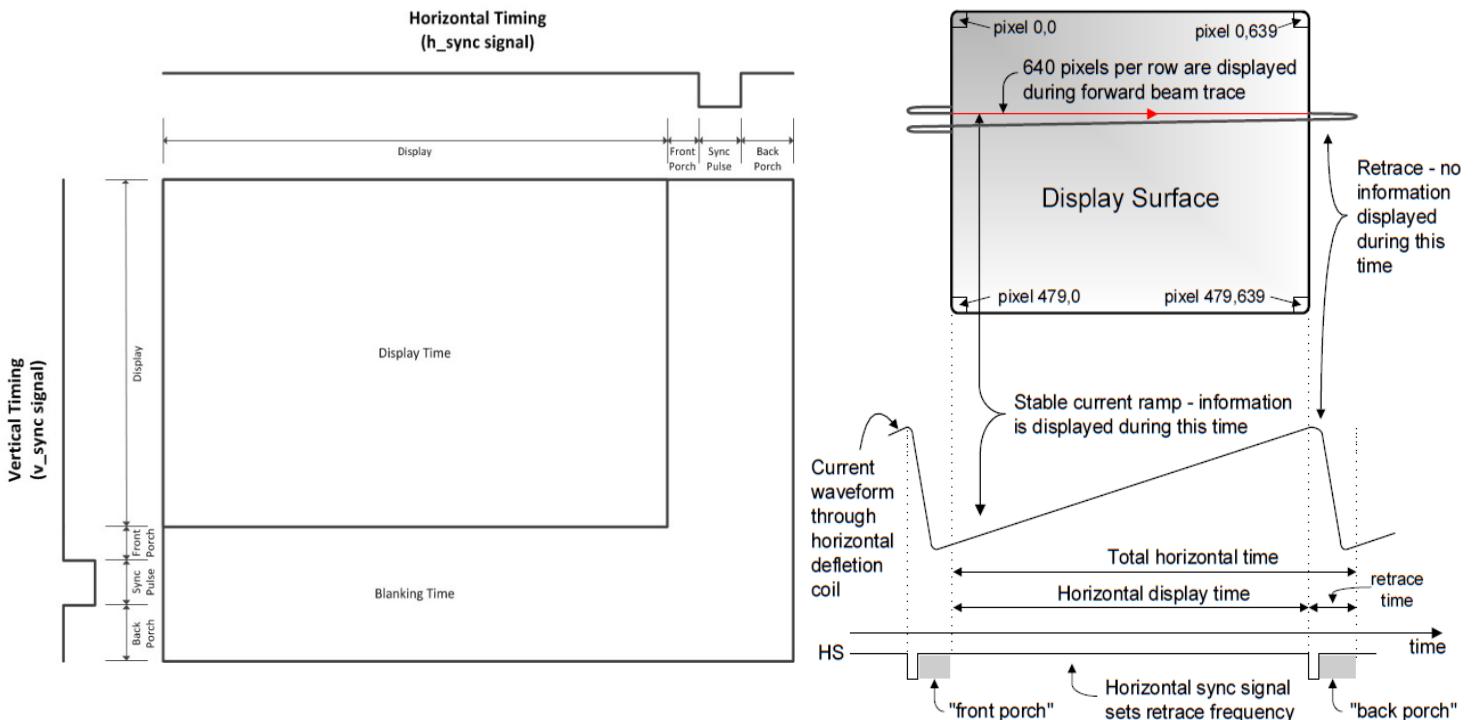


Figura 2.1: temporizzazione del display.

Quando si è “fuori schermo” i segnali RGB (*Red-Green-Blue*) devono essere a 0. I dati dei pixel si susseguono alla frequenza di clock di 25.1 MHz. Per ottenere questa esatta frequenza di aggiornamento dei pixel è stato utilizzato il “Clocking Wizard”, un tool interno all’ISE Design suite che permette di ottenere una frequenza entro un certo range a partire dal clock interno a 100 MHz dell’FPGA.

Come mostrato nella Figura 1.3 oltre ai segnali RGB sono presenti i due segnali di sincronismo HSync e VSync i quali devono essere in uno stato logico alto nell’area in cui vengono disegnati i pixel (*active region*), e poi andare allo stato logico basso (e rimanervi per un tempo detto “*sync time*”) nella regione in cui i pixel non vengono disegnati (*blinking region*) dopo un tempo detto “*front porch*”; i segnali di sincronismo tornano allo stato logico alto entro un tempo chiamato “*back porch*” prima di uscire dalla blinding region.

La durata dei tempi “*sync time*”, “*front porch*” e “*back porch*” è calcolata a partire dalla risoluzione dello schermo e fornita nella Tabella 2.1.

Per meglio comprendere le scelte in fase progettuale è conveniente analizzare il firmware dividendolo in parti elementari per poi giustificare il modo in cui sono stati collegati i vari blocchi.

Inizialmente ci si focalizzerà sul codice “VGA.vhd”, contenente l’intero driver per il monitor VGA e il display a 7 segmenti, utilizzato per leggere i comandi che si stanno inviando al rover premendo la pulsantiera.

Come è possibile vedere in Figura 2.2, l’entità *VGA.vhd* è composta da un totale di tre oggetti: *Sync.vhd*, *PLL.xco* e *Display7.vhd*.

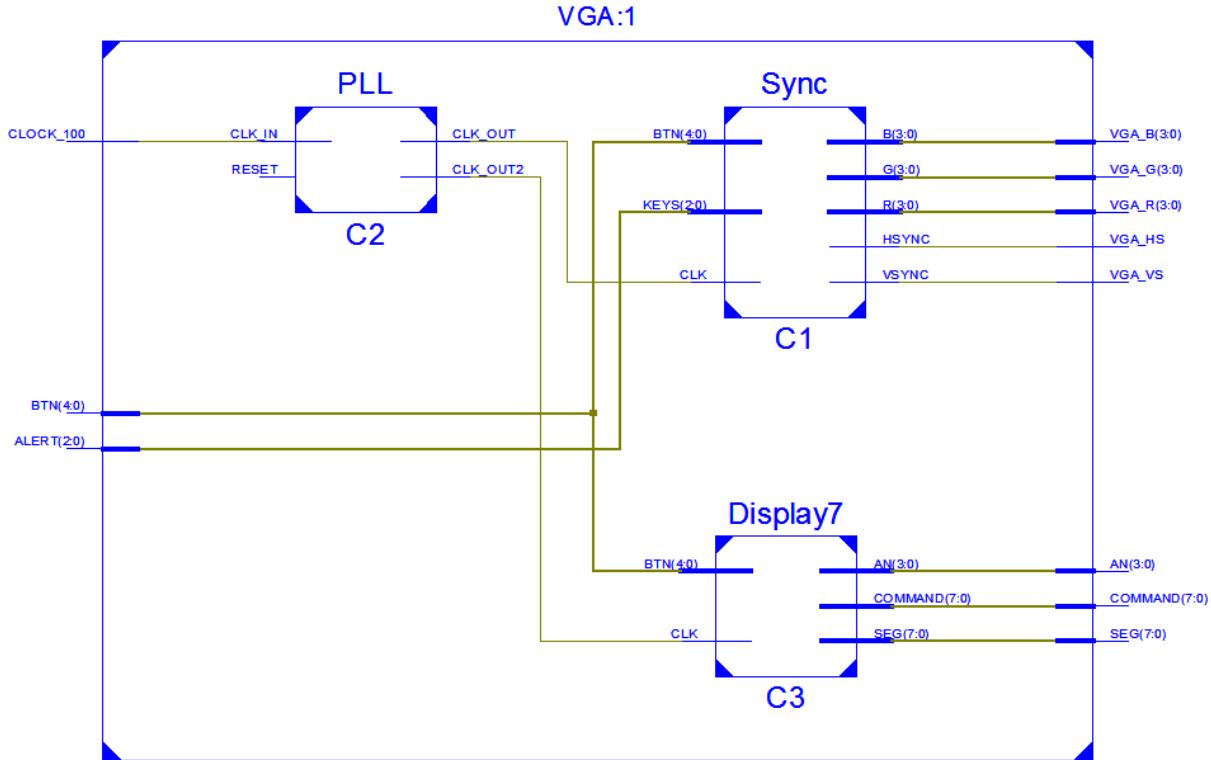


Figura 2.2: schema RTL di *VGA.vhd*.

- **Sync.vhd**: Il vero core necessario per la sincronizzazione dei segnali VGA. Utilizza un clock variante a seconda della risoluzione del monitor che viene generato dal file *PLL.xco*. Combinato con l’uso di HSYNC e VSYNC si riesce ad eseguire nei tempi previsti lo scanning di tutti i pixel del monitor, inclusi quelli di blanking time, permettendo così l’illuminazione dei pixel utili. Diverse librerie sono poi utilizzate per il vero e proprio disegno della macchina, dei segnali di allarme e dei bottoni della pulsantiera. Sono stati scelti dei colori accesi per meglio contrastare con il nero del monitor. L’idea alla base dell’illuminazione dei pixel giusti è basato sull’uso di un segnale “DRAW” interno a tutte le librerie per il disegno: quando i segnali HSYNC e VSYNC si trovano entrambi in corrispondenza di un pixel che deve essere illuminato, il segnale DRAW (diverso per ogni oggetto) viene settato alto ed il pixel viene colorato. Diversamente se nessun segnale DRAW è alto, il pixel rimane spento. In questa maniera è facile anche combinare questi segnali con azioni come pressione di pulsanti o avvisi

provenienti dal rover. L'ultima parte del codice è invece dedicata alla sincronizzazione dei segnali per rispettare i tempi di *back porch*, *sync pulse* e *front porch*, come spiegato in precedenza.

- **PLL.xco**: Questo è un file auto generato in seguito all'impiego del *Clocking Wizard*, tool che prende come input il clock a 100 MHz dell'FPGA e genera come output il clock a 85.8 MHZ usato dal “Sync.vhd” e quello a 5 MHz usato per il timing del Display a 7 segmenti.
- **Display7.vhd**: Lo scopo di questo codice è visualizzare un breve comando utilizzando 4 digit che possa rendere cosciente l'utente del messaggio che si sta inviando al rover.

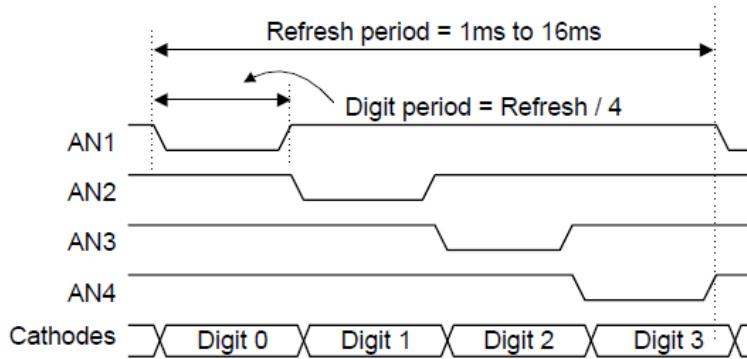


Figura 2.3: sincronizzazione anodi e catodi.

Ogni digit è composta da 7 led cui anodo è comune, ogni led all'interno della stessa cifra può essere illuminato singolarmente. La difficoltà risiede nell'illuminare le altre cifre senza far notare lo spegnimento delle precedenti, cambiando l'anodo in questione in un tempo ragionevolmente alto così da non permettere all'occhio umano di accorgersi della variazione (Figura 2.3). Ciò è stato ottenuto con un clock a 5 MHz, che tramite una catena di “if” conta un determinato numero di cicli di clock prima di cambiare anodo da mettere ad un livello logico alto. Una volta selezionato l'anodo corretto vengono illuminati i catodi per costruire la scritta. Duplice scopo di “Display7.vhd” è anche quello di inviare i byte di ascii per pilotare il rover, una volta infatti associata la scritta al bottone che si sta premendo è immediato avere come output un segnale di comando del mezzo. In caso non si stiano premendo pulsanti e quindi si voglia la macchina ferma, verrà inviato un comando “S” di stop, senza visualizzare nulla nel display a 7 segmenti.

Il risultato finale, ciò che viene visualizzato sul monitor VGA, è rappresentato in Figura 2.4.

Pulsantiera e segnale acustico vengono pilotati lato FPGA, mentre i segnali di allarme per ostacoli e guasto motore provengono dal rover e dipendono dalle soglie impostate all'interno dei firmware a bordo.

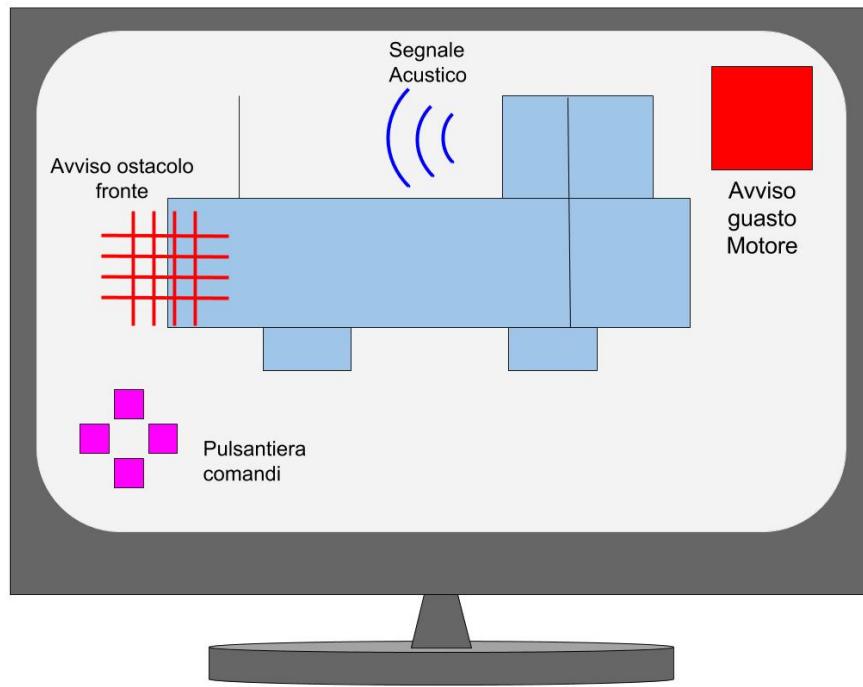


Figura 2.4: monitor VGA.

2.1.2 UNIVERSAL ASYNCHRONOUS RECEIVER TRANSMITTER (UART)

Il secondo e ultimo macro blocco riguarda la comunicazione seriale e come questa viene gestita per avere trasmissioni real time.

Affinché si possa avere uno scambio di dati rapido sia per aggiornare la schermata VGA e avere un quadro sulla condizione del rover sia per inviare rapidamente comandi ai motori, è stato scelto di inviare/trasmettere dati ogni 50 ms in modo continuativo, mandando *STOP* nel caso in cui non si stia premendo alcun pulsante; ad ogni pacchetto di comandi inviato si ha quindi indietro un *ack* contenente lo stato dei sensori e motori, cosicché si possa avere un'idea della situazione anche con il mezzo fermo. Tutto questo avviene con il compromesso di un maggior uso di risorse.

Il file “uart.vhd” viene già fornito funzionante, il lavoro è consistito nel programmare due differenti macchine a stati per regolare la trasmissione e ricezione dei pacchetti (un byte alla volta) tramite trasmissione seriale con il PIC18LF25k80. Il debug è stato effettuato prima utilizzando gli switch e led dell’FPGA per verificare l’invio e ricezione corretta dei pacchetti. In seguito è stato adoperato il software *RealTerm* per intercettare i dati scambiati tra FPGA e PIC.

2.1.2.1 TRASMISSIONE: TX_COUNTER.VHD

Compito di questa FSM è abilitare la trasmissione non appena il contatore interno arriva a 50 ms da quando è stato attivato.

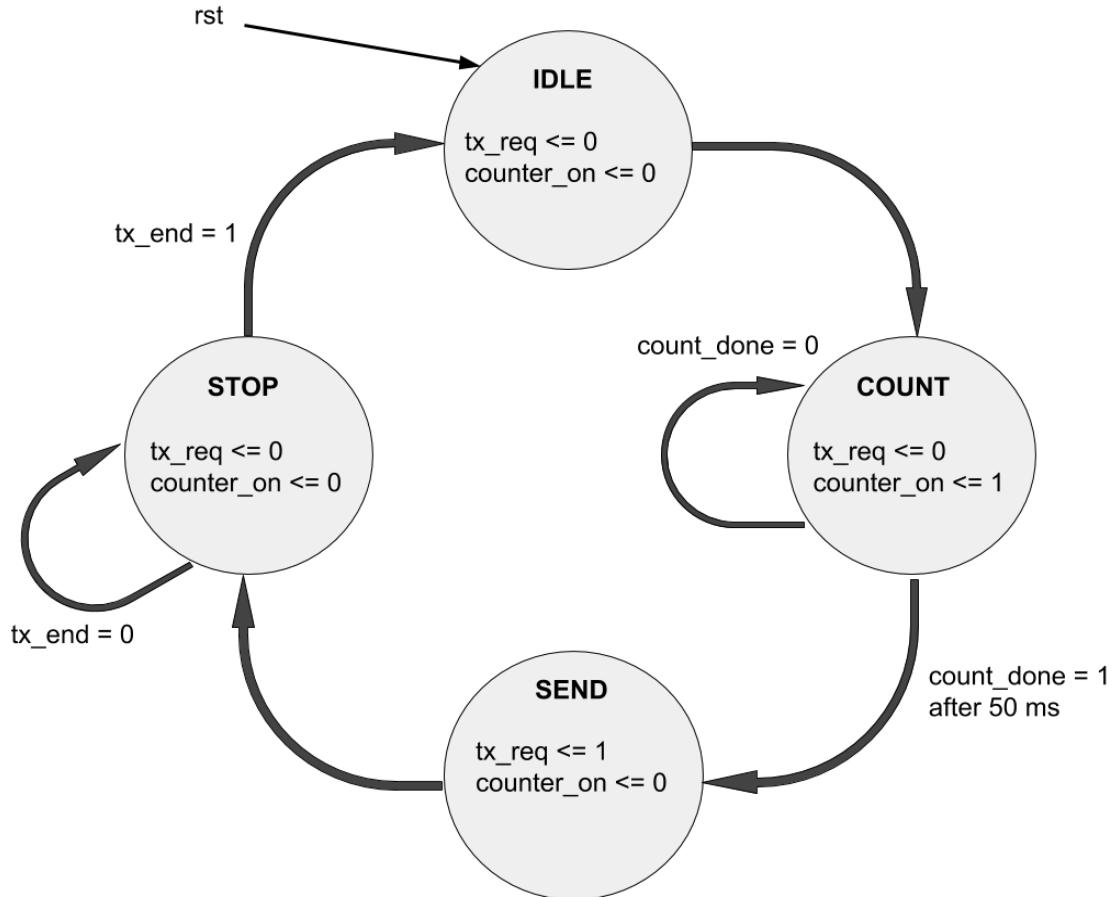


Figura 2.5: macchina a stati finiti per la trasmissione uart.

A quel punto il segnale *tx_req* viene posto al livello logico alto e il segnale *COMMAND* uscente dal blocco VGA viene campionato e inviato sotto forma di byte dal blocco *uart*. Fino a quando il segnale *tx_end* rimane basso la FSM non torna allo stato di IDLE, facendo ripartire il conteggio. Il processo è schematizzato nella Figura 2.5.

2.1.2.2 RICEZIONE: RX_FSM.VHD

Similmente al caso precedente anche la ricezione avviene dopo circa 50 ms, visto che questo periodo coincide con la ricezione di un *ack*.

La FSM viene attivata e si sveglia dallo stato di IDLE non appena il segnale *rx_ready* va alto, indicando che il pacchetto seriale è pronto per essere letto. A quel punto il dato viene salvato su un vettore *DATA* per evitare che venga modificato prima di essere correttamente elaborato e nello stato *READ_DATA* viene assegnato il valore di *ALERT* a seconda del pacchetto ricevuto, questo indica lo stato dei sensori e dei motori permettendo così l'aggiornamento dell'interfaccia VGA.

Non appena il dato ricevuto viene letto il segnale *rx_ready* torna automaticamente basso e la macchina a stati ritorna nello stato di attesa in IDLE. Nella Figura 2.6 viene fornito lo schema della macchina a stati per la ricezione.

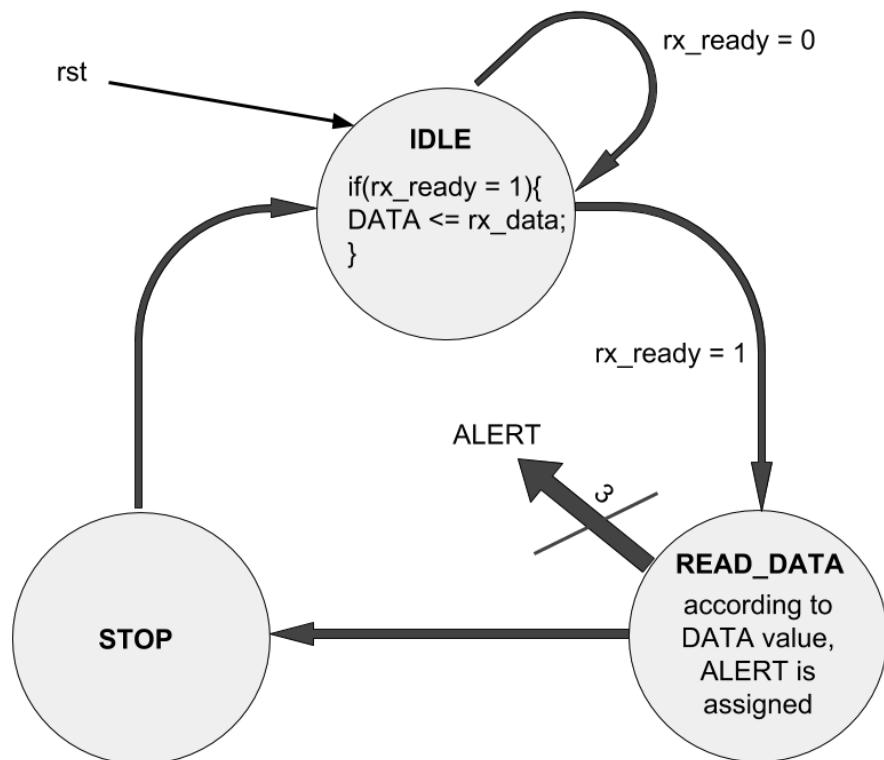
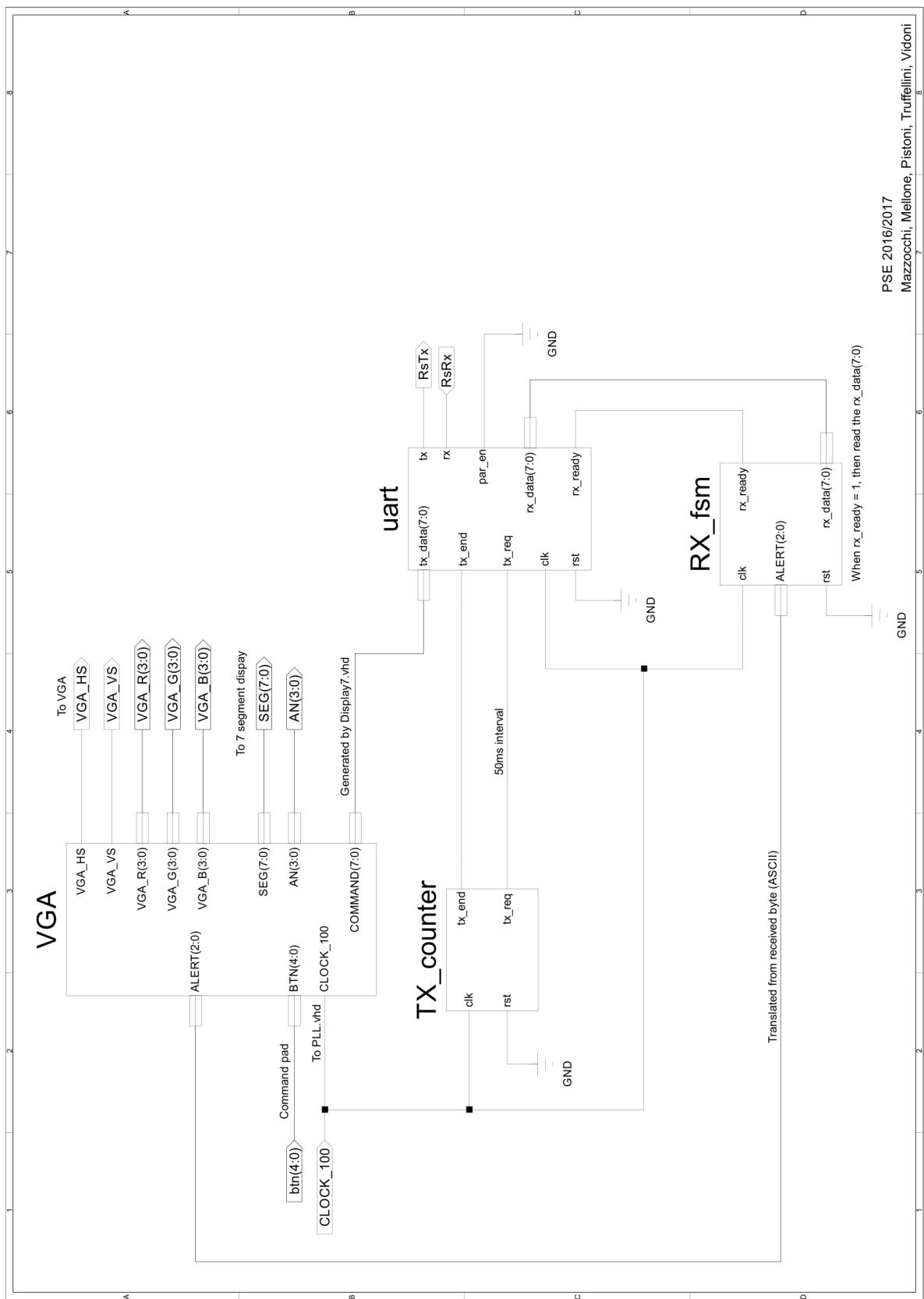


Figura 2.6: macchina a stati finiti per la ricezione uart.



L'unione dei due macro blocchi avviene in modo naturale: come si vede dalla **Errore. L'origine riferimento non è stata trovata.** il segnale di comando *COMMAND* (un byte) si collega direttamente al blocco *uart.vhd* pronto per essere inviato non appena il contatore di *TX_counter* segna i 50 ms.

Il byte ricevuto, diversamente, viene prima elaborato dalla macchina a stati finiti *RX_fsm* e ciò che entra nel blocco “*VGA.vhd*” è il segnale *ALERT* (3 bit) che a seconda della posizione degli uni accende gli avvisi sul monitor VGA.

2.2 NORDIC nRF24L01

I dati provenienti dalla FPGA tramite comunicazione seriale sono gestiti dal PIC#1: questo microcontrollore gestisce il modulo per la comunicazione wireless (Nordic *nRF24L01*) utilizzando il bus SPI.

2.2.1 SERIAL PERIPHERAL INTERFACE (SPI)

Il bus SPI è un bus di comunicazione seriale sincrono sviluppato dalla Motorola del genere Master-Slave.

Esso si definisce seriale in quanto al trasmissione-ricezione dei dati avviene bit a bit, sincrono per la presenza di un clock che coordina le operazioni di trasmissione e ricezione dei bit e determina la velocità di trasmissione e si definisce inoltre full-duplex poiché il processo di trasmissione e ricezione può avvenire contemporaneamente e in maniera simultanea.

Il master è il dispositivo che controlla il sistema, nel nostro caso un microcontrollore PIC, emettendo il segnale di clock e stabilendo l'inizio e la fine della comunicazione, mentre lo slave è un dispositivo periferico che può ricevere e inviare dati, ma non può inviare comandi, né iniziare una sessione di trasmissione, il modulo radio *nRF24L01* nel presente progetto. Il clock con cui trasmette e riceve i dati è in ogni caso fornito dal master e lo slave non ha alcun controllo su questa linea.

Il sistema è comunemente definito a quattro linee di comunicazione i cui nomi possono differire dal tipo di dispositivi utilizzati, infatti, sebbene l'SPI sia considerato un bus di comunicazione standard, manca di un protocollo di comunicazione univocamente definito, tuttavia, vengono solitamente indicate con le seguenti sigle:

- **MOSI (SDO)**: Master Output Slave Input (canale per la trasmissione dati dal master allo slave);
- **MISO (SDI)**: Master Input Slave Output (canale per la trasmissione dati dallo slave al master);
- **SCLK**: Serial Clock (canale del clock che scandisce gli istanti di trasmissione e ricezione dei bit sulle apposite linee);
- **SS**: Slave Select (canale attraverso cui si sceglie lo slave con cui s'intende comunicare, solitamente attivo basso).

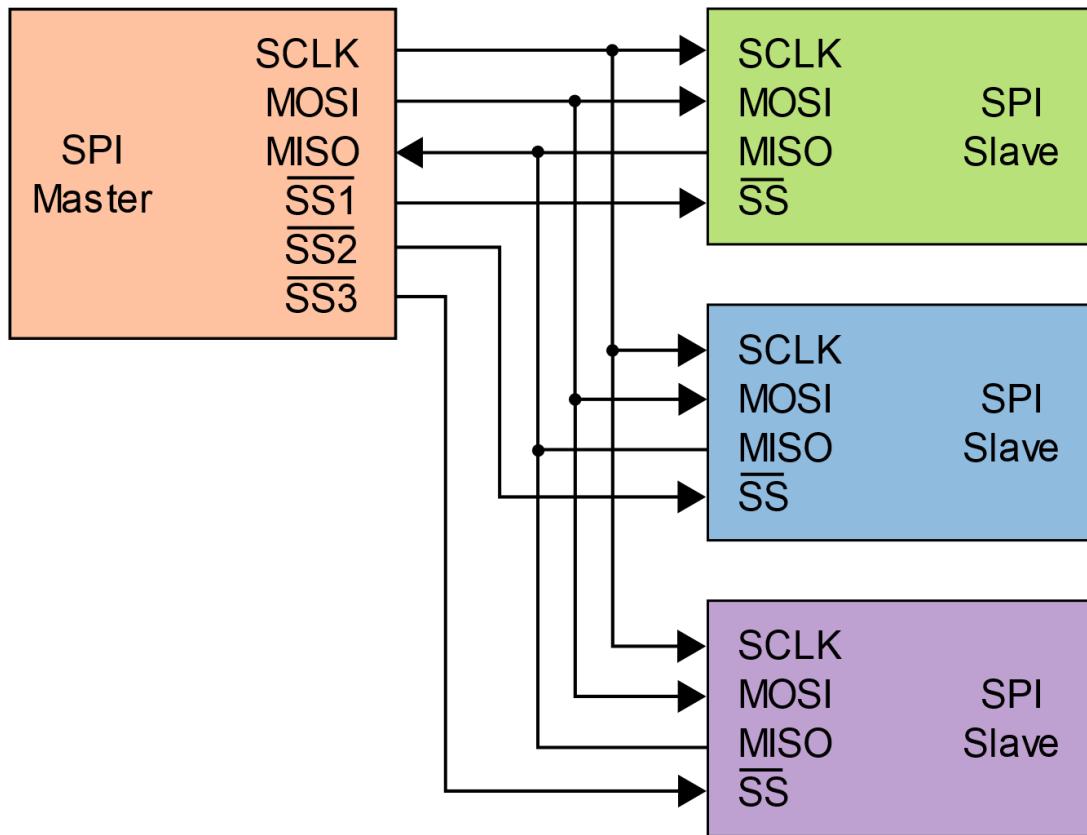


Figura 2.7: esempio di connessione diretta tra un master e tre slave controllati singolarmente.

Il protocollo SPI non è stato studiato per essere implementato all'interno di un sistema multi-master, ma è possibile collegare più slave ad un unico master come in Figura 2.7, cambiando quindi lo SS è possibile selezionare lo slave a cui si è interessati. Nel presente lavoro la comunicazione avviene tra un PIC che fa da master ed un solo modulo wireless che fa da slave, quindi il ruolo dello SS non è cruciale.

Per quanto concerne la struttura interna del sistema di comunicazione, ogni dispositivo SPI possiede al suo interno un registro a scorrimento utilizzato per inviare e ricevere i dati serializzati (Figura 2.8).

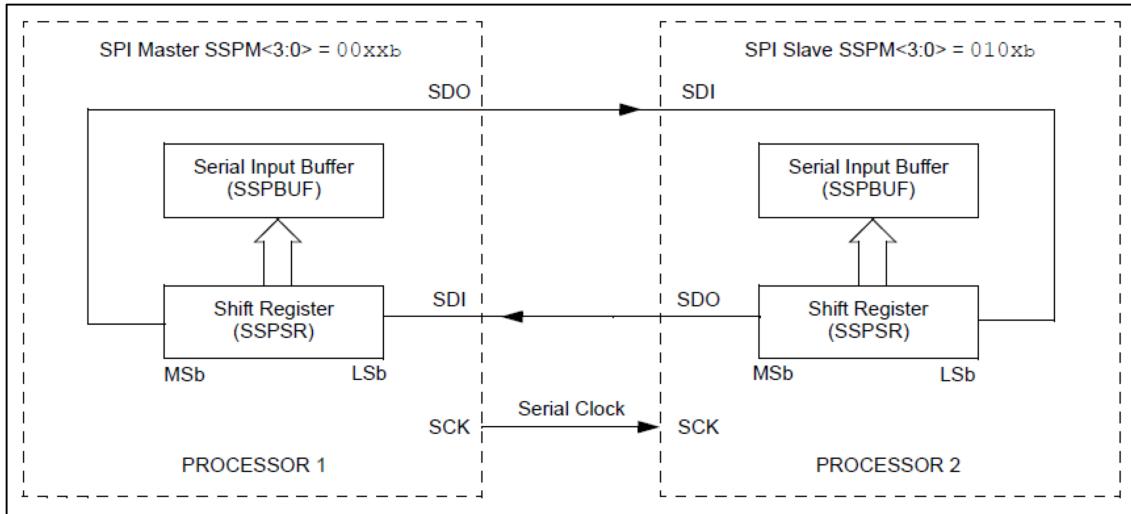


Figura 2.8: registro a scorrimento SPI nel PIC microcontroller.

In SPI usualmente tale registro ha una dimensione di 8 bit. Se si suppone un trasferimento da parte del master verso lo slave, il dato contenuto nell'ultimo flip-flop verrà trasmesso sul canale MOSI, a sua volta, nello slave, connesso al primo flip-flop del suo registro a scorrimento. In maniera analoga, l'ultimo flip-flop del registro a scorrimento dello slave è collegato al canale in uscita MISO, a sua volta, nel master, legato al primo flip-flop del proprio registro a scorrimento costituendo un architettura ad anello chiuso.

I bit che avanzano nei due registri a scorrimento utilizzano la medesima sincronizzazione proveniente dal clock generato dal master e seguendo un ordine che va dal bit più significativo al meno significativo o viceversa, a seconda di quanto impostato nel modulo SPI del master.

2.2.1.1 PROGRAMMAZIONE INIZIALE E DEBUGGING

Il PIC presenta già delle librerie per l'utilizzo dell'SPI e tutto risiede nel settaggio dei registri `SSPCON1` e `SSPSTAT`, che sono i registri di controllo e di stato nella modalità SPI. I registri invece contenenti il dato da inviare e il dato ricevuto in contemporanea sono chiamati `SSPBUF` e `SSPSR`. La funzione di scrittura e di ricezione del dato è sintatticamente molto semplice e riportata in seguito:

```

unsigned char spi_write(unsigned char data) {
    SSPSTATbits.BF = 0;
    SSPBUF = data;
    while(SSPSTATbits.BF == 0){}
    return SSPBUF;
}

```

Il dato di ritorno è il byte che viene ricevuto dal buffer, essendo un protocollo a scorrimento infatti ogni volta che si invia un dato se ne riceve un altro dallo slave.

Come illustrato precedentemente, l'SPI è un protocollo con diverse configurazioni disponibili, quindi è stato necessario comprendere quali impostazioni fossero necessarie perché avvenisse la scrittura tra PIC e *nRF24L01*.

Vedendo che i tentativi iniziali di programmazione al fine di far comunicare usando due PIC i moduli erano vani si è deciso per un altro approccio, aiutandosi con delle librerie per Arduino scritte appositamente per il modulo in questione. Queste sono state utili a trovare quali impostazioni ottimali per l'SPI fossero necessarie per far avvenire il dialogo tra PIC e Nordic ed in particolar modo come settare i bit *SMP*, *CKE* e *CKP*. Di seguito viene riportata la maniera corretta per il settaggio dei bit ed in Figura 2.9 le corrispondenti forme d'onda SPI per tutte le possibili combinazioni.

<code>SMP = 0; // Sample bit</code>	<code>1 = Input data sampled at end of data output time</code>
	<code>//0 = Input data sampled at middle of data output time</code>
<code>CKE = 1; // Clock Edge Select</code>	<code>1 = Transmit occurs on transition from active to Idle clock state</code>
	<code>//0 = Transmit occurs on transition from Idle to active clock state</code>
<code>CKP = 0; // Clock Polarity</code>	<code>1 = Idle state for clock is a high level</code>
	<code>//0 = Idle state for clock is a low level</code>

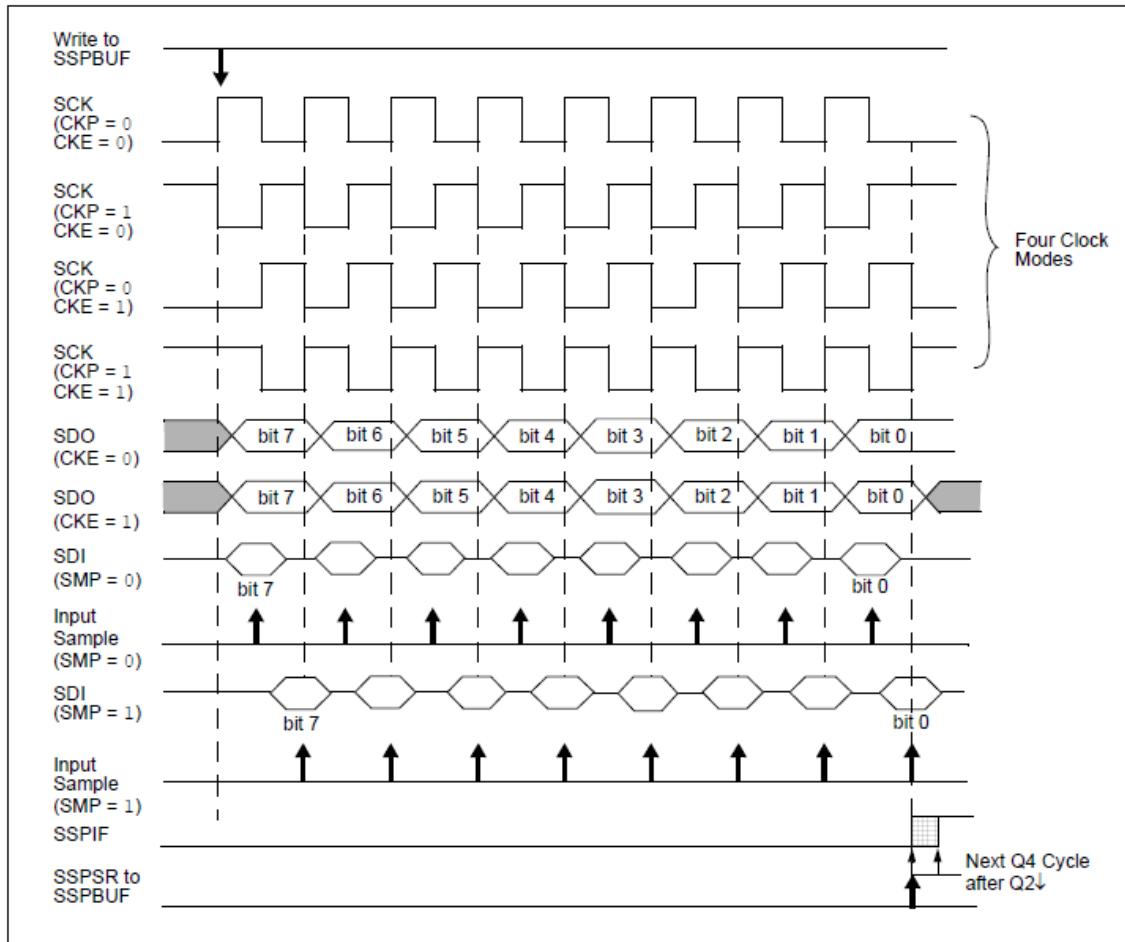


Figura 2.9: forme d'onda per diversi settaggi dei bit di configurazione riguardanti l'SPI.

Il primo strumento utilizzato per il debug è stato l'oscilloscopio: visualizzando clock (*SCK*) e *MOSI* (*SDO*) in caso di corretto funzionamento ci si aspetta una corrispondenza tra i segnali trasmessi e come le onde quadre dell'SPI si sovrappongono tra clock e *SDO*.

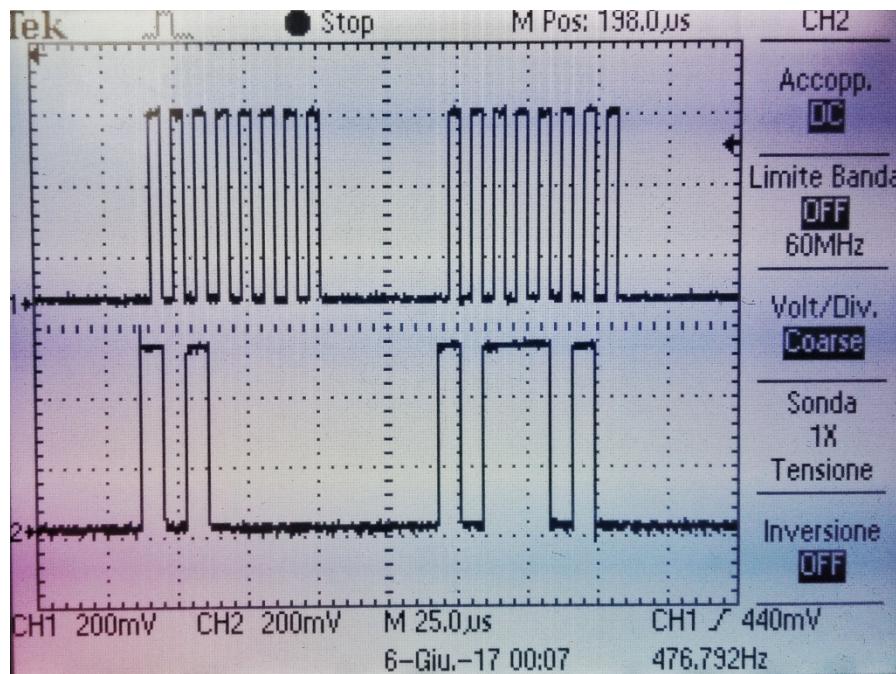


Figura 2.10: segnale SPI uscente dal nRF24L01: sopra CLK, sotto MOSI.

Osservando la Figura 2.10 si riesce a comprendere come funziona la scrittura sulla memoria del modulo: in alto si ha il segnale di clock, con 8 colpi (uno per bit). Sotto si hanno invece due segnali diversi: il primo a sinistra indica 0xA0 e vuol dire che si vuole scrivere sul *W_TX_PAYLOAD* inviando quindi il byte che si vuole trasmettere via aria, il segnale a destra invece indica il dato da trasmettere che in questo caso è 0xBA.

Una volta constatato che la trasmissione SPI funziona come dovrebbe è stato quindi necessario capire se si stesse effettivamente scrivendo sulla memoria del modulo qualcosa che lui fosse in grado di comprendere ed elaborare.

L'idea è stata quindi quella di abilitare la trasmissione continua dei pacchetti, senza quindi aspettare un ACK dalla controparte, ed utilizzare un analizzatore di reti.

Il problema era che non si aveva la certezza che si stessero effettivamente trasmettendo dei pacchetti, per cui l'idea è stata utilizzata un'antenna patch con frequenza centrale a circa 2.4GHz collegata all'analizzatore di rete Anritsu per captare eventuali pacchetti in aria.

La frequenza di lavoro del Nordic è stata impostata a 2.51GHz per evitare i disturbi del Wi-Fi a 2.4GHz. Ponendosi in ascolto sulla stessa frequenza si è registrato il segnale riportato in Figura 2.11.

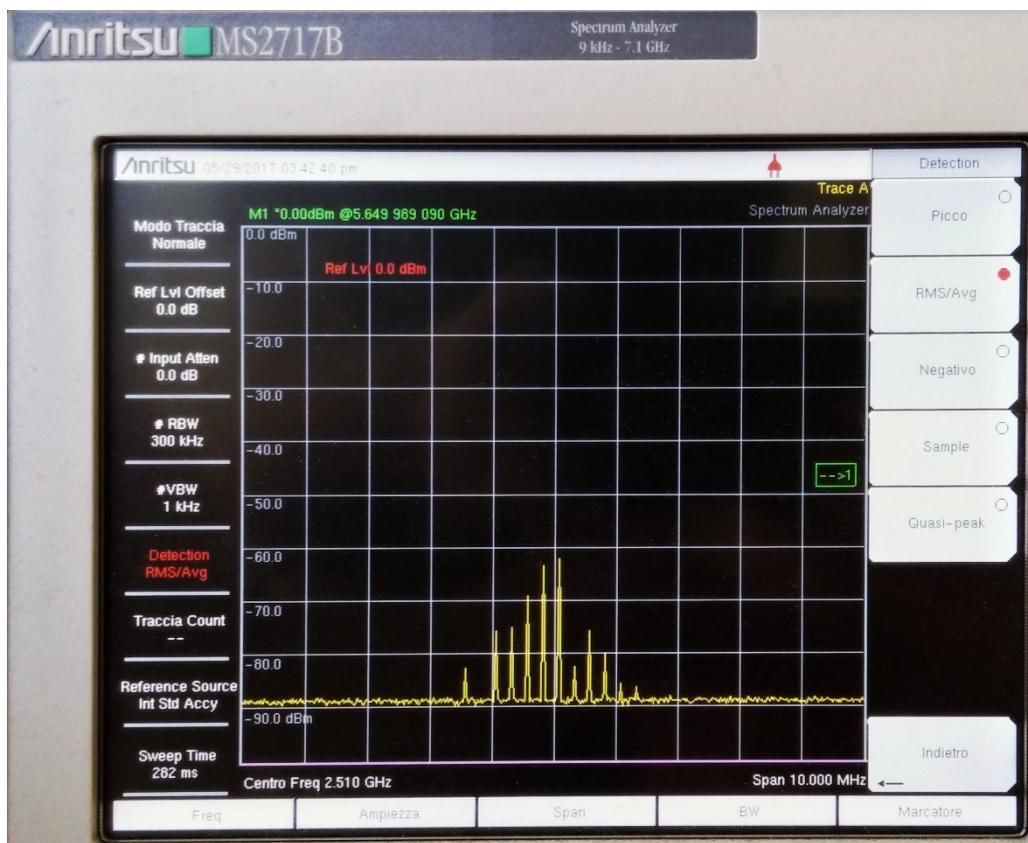


Figura 2.11: schermata analizzatore di reti con antenna patch centrata a 2.51GHz.

Lo spettro ottenuto è una chiara prova che si stesse realmente trasmettendo. Cambiando la frequenza di trasmissione, la potenza ed altri parametri infatti il segnale muta e scompare allo spegnimento del modulo.

Avendo così la certezza del funzionamento del *nRF24L01* è stato possibile proseguire con la programmazione delle funzioni desiderate.

2.2.2 COMUNICAZIONE E PROGRAMMAZIONE DI FUNZIONI AVANZATE

Con la sicurezza di poter scrivere sulla memoria del modulo in modo corretto sono state quindi implementate tutte le funzioni necessarie alla comunicazione tra i Nordic, tra cui potenza, canale e indirizzi. Un occhio di riguardo è stato dedicato al fault tolerance.

La trasmissione dei dati deve essere robusta e veloce ed è stata scelta una frequenza di lavoro di 2.51GHz onde evitare interferenze con dispositivi che lavorano a 2.4GHz come WiFi.

Preamble 1 byte	Address 3-5 byte	Packet Control Field 9 bit	Payload 0 - 32 byte	CRC 1-2 byte
-----------------	------------------	----------------------------	---------------------	--------------

Figura 2.12: pacchetto di trasmissione Enhanced ShockBurst.

Nel momento in cui, cercando di trasmettere un dato, non si riceve risposta dall'altra parte, è stato deciso di effettuare 15 ritrasmissioni ad una distanza di 1 ms l'una dall'altra. Nel caso in cui si raggiunga il massimo numero di ritrasmissioni un bit nel registro di status cambia e questo viene rilevato dalla funzione *check_MAX_RT()*, a quel punto per tentare nuovamente di avere la comunicazione è necessario eseguire il reset del bit *MAX_RT* nel registro di *STATUS* e si può ricominciare la ritrasmissione. Questa operazione viene eseguita finché non si riaggancia il segnale dall'altra parte.

Inoltre è stato fatto in modo che nel momento in cui si perde il segnale venga incrementata ad ogni giro di main una variabile contatore *counter_no_data*; usando un “if” viene eseguito il polling ad ogni giro sulla variabile e quando il valore supera una certa soglia la variabile che contiene il comando inviato dalla FPGA viene messo su *STOP*. In questa maniera si evita che al momento della perdita del segnale, la variabile che pilota i motori rimanga salvata in una posizione senza possibilità di essere cambiata se non togliendo l'alimentazione o facendo in modo di riagganciare il segnale.

Il sistema di gestione dei pacchetti del Nordic “Enhanced Shockburst” (Figura 2.12) offre anche la possibilità di avere CRC (*Cyclic Redundancy Check*) per rendere più robusta la trasmissione. Nel presente lavoro è stato deciso di optare per un CRC di 1 byte essendo i dati da trasmettere pacchetti molto leggeri di al più 1 byte.

Una volta accertato che la comunicazione avviene in maniera corretta anche dopo l'implementazione delle varie funzioni per scrittura e lettura dei pacchetti è stato necessario trovare un modo per rendere la comunicazione full-duplex. I dati dati devono essere inviati da entrambe le parti e non avendo una connessione cablata è difficile gestire a distanza lo scambio di ruolo tra trasmettitore e ricevitore.

La soluzione è stata quella di utilizzare l'ACK come pacchetto per comunicare, scrivendo dentro lo stesso un altro payload. In questo modo si ha una netta distinzione tra *Primary RX* (PIC2 = PRX) e *Primary TX* (PIC1 = PTX). Vengono chiamati “primary” poiché sono i principali ricevitori/trasmettitori ma quando il *PTX* ha inviato il pacchetto si mette immediatamente in attesa dell'ACK e diventa quindi RX, viceversa quando il *PRX* riceve il pacchetto correttamente e rimanda indietro l'ACK con il payload sullo stato dei sensori esso cambia in TX. Il funzionamento è spiegato in Figura 2.13.

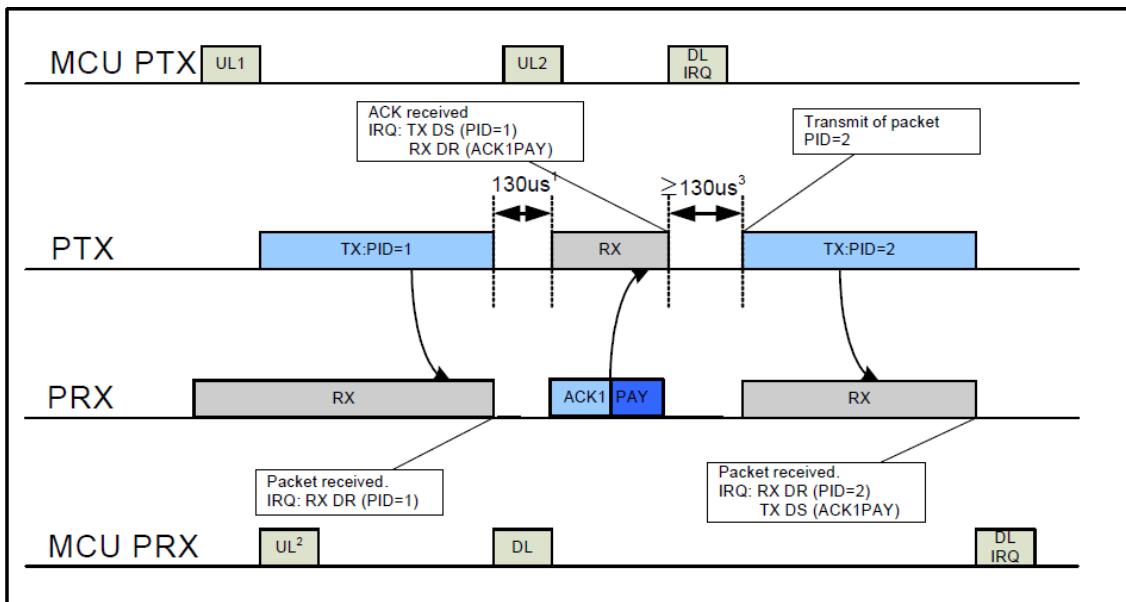


Figura 2.13: comunicazione tra nRF24L01 avendo ACK + payload.

Tutto il funzionamento e la gestione dei tempi e delle priorità è gestito da un complesso sistema di macchine a stati per ricevitore e trasmettitore. In Figura 2.14 si riporta una macchina a stati semplificata che viene eseguita per il controllo della parte radio tra settaggio iniziale e invio/ricezione dei pacchetti.

La trasmissione e la ricezione dei pacchetti vengono poi gestiti da FSM più complesse specialmente se, come nel caso del presente lavoro, è stato implementato il payload dinamico per la gestione della scrittura del payload nell'ACK.

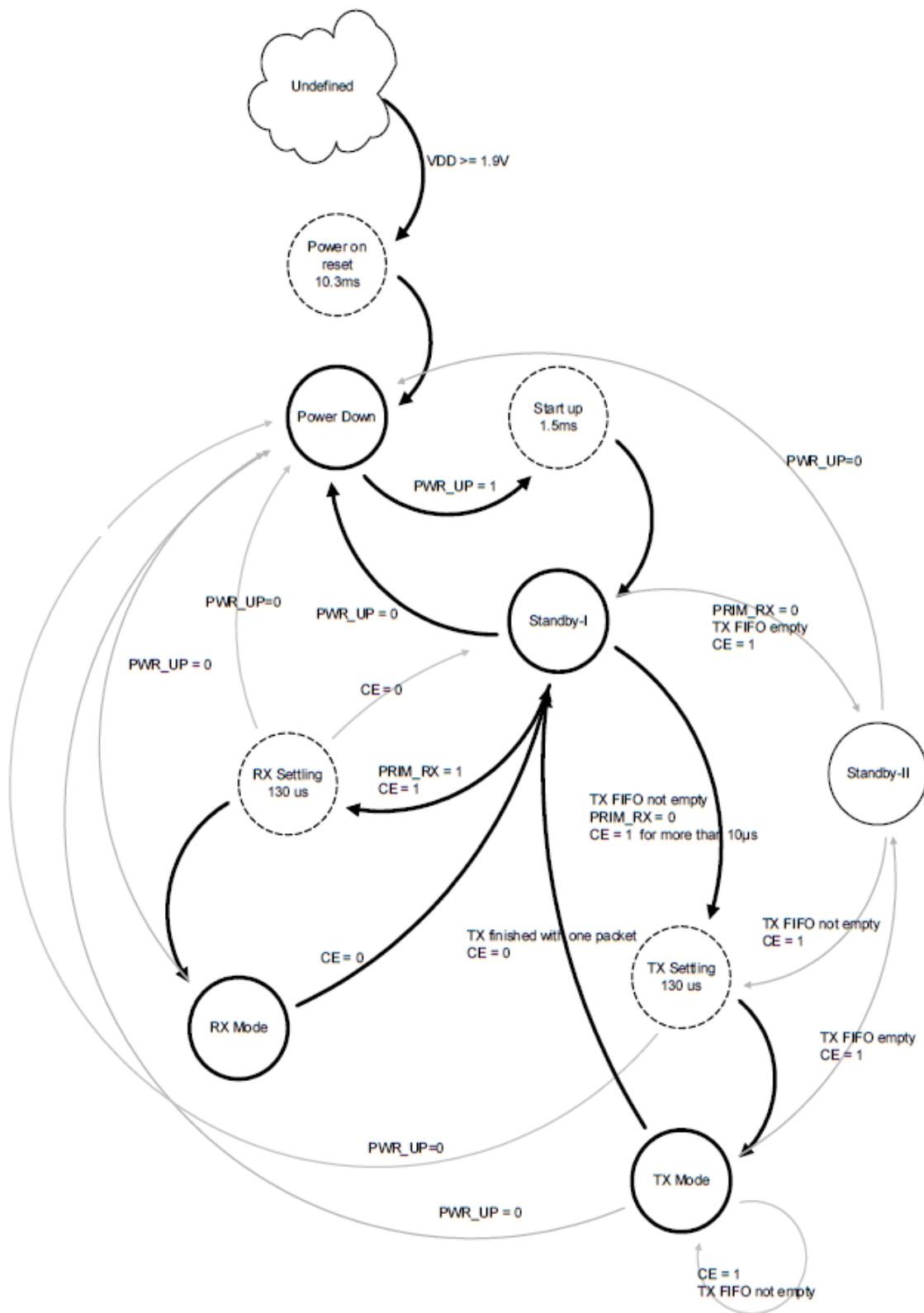


Figura 2.14: macchina a stati finiti inherente la gestione radio TX e RX.

2.3 SENSORI HC-SR04

Il calcolo della distanza viene eseguito dal PIC#3-3bis tramite una routine di interrupt, dove il programma entra ogni volta che si rileva un segnale di ritorno.

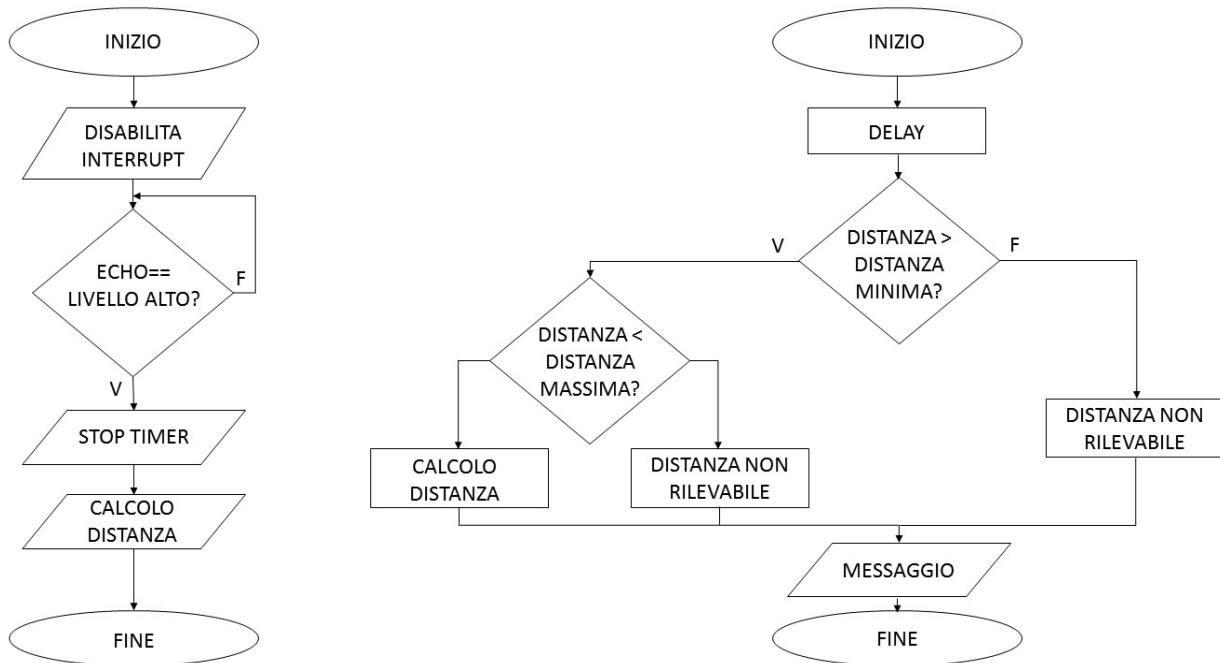


Figura 2.15: diagramma di flusso per il calcolo della distanza (a sx) e per la trasmissione del messaggio (a dx).

Nell'implementazione sono state adoperate alcune accortezze per aumentare la robustezza e il fault tolerance del firmware. Sono presenti due sensori per ogni PIC3, per un totale di due sensori davanti e due sensori dietro.

Nel momento in cui uno solo dei sensori della coppia rileva un ostacolo questo viene segnalato, lo scambio di ruoli tra i sensori associati ad ogni PIC viene gestito da una variabile *token* il cui valore viene incrementato ad ogni giro di main e il valore massimo è minore del numero di sensori utilizzati; a seconda del valore del token viene abilitato l'interrupt on change solo sul piedino dove ci si aspetta di ricevere l'eco, così facendo si evita che eventuali disturbi causino l'entrata nella routine di interrupt senza aver acquisito un valore buono.

Inoltre è stato scelto di utilizzare due soglie: la prima soglia minima abilita solo la visualizzazione dell'allarme sul monitor VGA; se invece viene rilevato che la distanza rover-ostacolo supera anche la seconda soglia in questo caso si invia tramite CAN il segnale di arresto dei motori al PIC2 collegato, oltre che all'allarme visivo, anche all'allarme sonoro del clacson.

Il vantaggio di questa soluzione è la possibilità da parte dell'utente di decidere la politica da intraprendere pur sapendo della presenza di un eventuale ostacolo ad una distanza non critica.

Nel caso in cui la distanza diminuisca ancora, raggiungendo la distanza critica, a quel punto l'auto si arresta in modo autonomo per evitare incidenti.

Data la sensibilità dei sensori wireless e dei frequenti falsi valori che vengono ottenuti, sia per problemi con i disturbi del canale di trasmissione sia per la natura stessa dei sensori, è stato scelto di implementare un sistema a contatori per irrobustire il calcolo del segnale prima di inviare e mandare in circolo il byte di stato dei sensori.

L'idea è stata quella di inviare il dato solo dopo averlo validato in seguito a 2-3 giri di main. Sono state usate quindi 2 variabili contatore: una per il conteggio della distanza critica per cui si visualizza soltanto l'allarme e un'altra per il conteggio della distanza critica per cui si ha anche il blocco del motore e il segnale acustico. Nel momento in cui si passa dalla validazione di una distanza all'altra, si azzera la variabile contatore della distanza precedente e si incrementa quella della nuova distanza.

Con diversi cicli *if* viene poi eseguito il polling sulle variabili contatori e in caso di esito positivo si invia il segnale riportante lo stato dei sensori.

Questo processo avviene al prezzo di un aumento della complessità del codice e di un maggior numero di segnali da gestire sui canali di comunicazione, ma in questa maniera si aumenta la robustezza evitando falsi allarmi e possibile arresto (e non arresto) del veicolo dovuto ad essi.

La funzionalità di real-time non viene persa grazie alla velocità di aggiornamento dei dati: il vantaggio di avere i sensori collegati a dei PIC separati permette di sfruttare appieno i microcontrollori senza doversi preoccupare dei ritardi causati dalla gestione dei motori e dei moduli wireless come avviene nel PIC2. L'unico collo di bottiglia è l'attesa dei 50 ms necessari per attendere il ritorno dell'eco, in questo lasso di tempo infatti non è possibile svolgere alcuna operazione per evitare di alterare il segnale ricevuto.

Nonostante ciò è stato stimato che il dato sulla CAN viene aggiornato ogni 120/180ms in caso di rilevamento di un ostacolo, soddisfacendo quindi le caratteristiche di real-time per il presente progetto, tenendo in considerazione la velocità di movimento del rover.

2.4 CONTROLLER AREA NETWORK (CAN)

Il *Controller Area Network(CAN)* è un protocollo di comunicazione seriale utilizzato principalmente in ambiente automotive e in molte applicazioni industriali di tipo embedded. Questo è un sistema standardizzato basato solo sui primi due livelli del modello *ISO/OSI*.

Le caratteristiche principali del bus sono:

- Diminuzione dei costi di progettazione ed implementazione, dovuta al basso costo dei moduli di controllo e dell'hardware necessario per realizzare di un nodo CAN;
- Tecnica di accesso al bus non distruttiva (CSMA/CA - *Carrier Sense Multiple Access/Collision Avoidance*), per far fronte alla corruzione dei dati;
- Ritrasmissione automatica dei dati in caso di errore;
- Classificazione del contenuto dei messaggi tramite il campo identificativo: il bus non indirizza i nodi individuali, ma i messaggi che hanno inviato. Un nodo può così inviare un messaggio sul bus che verrà ricevuto da tutti gli altri nodi e filtrato dagli stessi in modo che ognuno utilizzi solo i messaggi che ritiene utili;
- Comunicazione *event-driven*, ossia la trasmissione viene inizializzata solo quando necessario. Tutti i componenti collegati al bus ascoltano in “parallelo” tutti i messaggi, sono sempre pronti a ricevere e, quando necessario, cominciano a trasmettere se nessun altro nodo sta inviando messaggi sul bus;
- Isolamento di eventuali nodi guasti senza che essi interferiscano con il resto del sistema.

La specifica utilizzata nel progetto è la *CAN 2.0A*, caratterizzata dalla lunghezza del campo di identificazione dei messaggi a 11 bit, come si può osservare in figura:

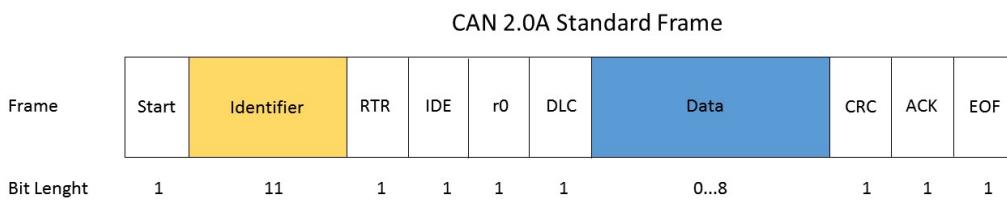


Figura 2.16: frame standard CAN 2.0A.

Una rete CAN nella sua versione più tradizionale è composta da un bus realizzato con una coppia di fili (*CAN_H* e *CAN_L*) e da una serie di interfacce(nodi) collegate tra loro tramite questo mezzo trasmittivo.

Nel nostro caso la struttura della rete è rappresentata dalla seguente figura:

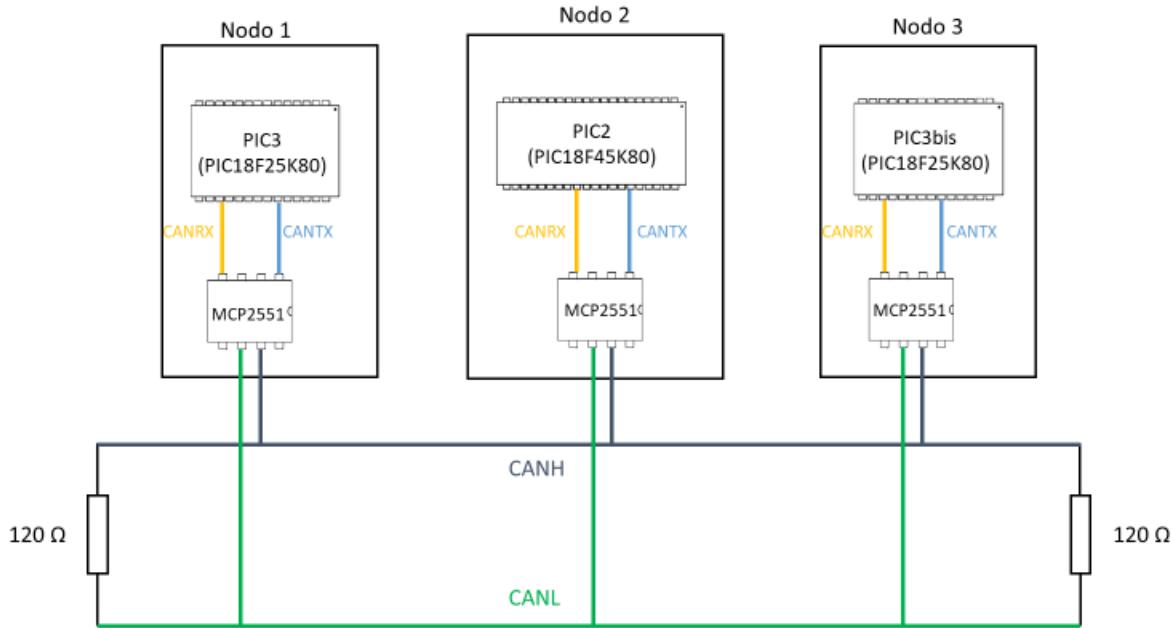


Figura 2.17: bus CAN UniMoRover.

Nel progetto il bus CAN viene utilizzato all'interno della vettura al fine di ottenere una comunicazione flessibile, robusta ed affidabile tra i vari microcontrollori. Nello specifico i dati corrispondenti alle distanze rilevati dai sensori vengono inviati sul bus CAN dai PIC #3 e #3bis. Il PIC#2 legge tali dati, in base al campo identificativo del messaggio determina la provenienza (sensori anteriori o posteriori) e li salva nei buffer appropriati. Due resistenze da 120Ω, utilizzate come terminazioni del bus, sono necessarie per minimizzare le riflessioni dei segnali.

Per quanto riguarda il lato software, sono necessari alcuni passaggi prima che il modulo CAN possa essere usato per trasmettere o ricevere messaggi:

- Inizializzazione dei bit *LAT* e *TRIS* per *CANRX* e *CANTX*.
- Assicurarsi che il modulo CAN sia in *Configuration Mode* e successivamente selezionare il modo operativo (*Operational Mode*):

```

CANCON = 0x80;                                // Configuration mode      Setting
                                                // REQOP<2:0>=100
CANSTATbits.OPMODE2 = 0x00;                      // Setting of CANSTAT in Normal mode
CANSTATbits.OPMODE1 = 0x00;
CANSTATbits.OPMODE0 = 0x00;

while(!(CANSTATbits.OPMODE ==0x04));           // Wait until CANSTAT isn't in Configuration mode
ECANCON = 0x00;                                // Operation mode 0 (Legacy mode)

```

- Settare i registri della baud rate. Nel nostro caso specifico, al fine di ottenere un valore di 100 Kbps con una frequenza di oscillazione di 8 MHz, occorre utilizzare le seguenti formule, necessarie per calcolare la durata del tempo di bit nominale e il time quanta (T_Q):

$$\text{Nominal Bit Time} = T_Q * (\text{Sync_Seg} + \text{Prop_Seg} + \text{Phase_Seg1} + \text{Phase_Seg2})$$

$$T_Q(\mu\text{s}) = (2 * (\text{BRP} + 1)) / \text{Fosc (MHz)}$$

Nella tabella sono riportati i risultati ottenuti:

Item	Value	Item	Value
BRP	2	SJW	1
TQ (us)	500	Phase_Seg1	8
N. Time Quanta	20	Phase_Seg2	8
Prop_Delay	3		

Tabella 2.2: valori dei registri di configurazione della CAN.

Grazie a questi valori, è stato possibile configurare i registri *BRGCON*:

```
BRGCON1 = 0x01; // Baud Rate Settings (100 Kbps)
BRGCON2 = 0xBA;
BRGCON3 = 0x07;
```

- Settare le maschere di ricezione solamente per messaggi standard:

```
// Mask 0 (M0) is configured only for standard messages
RXM0SIDH = 0xFF; // Standard Address receive acceptance mask,
                  // high byte
RXM0SIDL = 0xE0; // Standard Address receive acceptance mask,
                  // low byte
// Mask 1 (M1) is configured only for standard messages
RXM1SIDH = 0xFF; // Standard Address receive acceptance mask,
                  // high byte
RXM1SIDL = 0xE0; // Standard Address receive acceptance mask,
                  // low byte
```

- Settare il modulo CAN in *Normal Mode*:

```
//Normal mode
CANCON = 0b00000000; // Set CANCON in Normal mode
CANSTATbits.OPMODE2 = 0x00; // Setting of CANSTAT in normal mode
CANSTATbits.OPMODE1 = 0x00;
CANSTATbits.OPMODE0 = 0x00;

while(! (CANSTATbits.OPMODE == 0x04)); // Wait until CANSTAT isn't in normal mode
```

Come si può osservare dal codice, le maschere sono settate in modo tale da non accettare tutti i tipi di messaggio, ma solo quelli che combaciano con i relativi filtri. Il buffer 0 di ricezione accetterà solo i messaggi CAN con identificativo 0x05; mentre il buffer 1 solo quelli con identificativo 0x02.

I messaggi con identificativo “2” corrispondono ai dati relativi alle distanze rilevati dai sensori anteriori; mentre quelli con identificativo “5” si riferiscono ai dati dei sensori posteriori.

Per iniziare la trasmissione del messaggio, il bit TXREQ deve essere settato per ogni buffer che sarà utilizzato. La trasmissione inizierà quando il dispositivo rileverà che il bus è libero e sarà completata con successo se nella rete sarà presente un nodo con la stessa baud rate. Per verificare la corretta trasmissione del messaggio il bit TXREQ sarà settato automaticamente a 0. Se la trasmissione del messaggio non è andata a buon termine, TXREQ rimarrà settato a 1 così come il bit TXERR, in tal modo si entra nel ciclo *if* che fa accendere il led rosso, rendendo cosciente l’utente dell’errore appena avvenuto.

```
//Filter settings for the two RX Buffers
RXF0SIDH = 0x00;                                //Standard Address Filter-0 high byte set to
                                                    //0x00
RXF0SIDL = 0xA0;                                //Standard Address Filter-0 low byte set to 0xA0
                                                    // (ID= 0x05)
RXF2SIDH = 0x00;                                //Standard Address Filter-2 high byte set to
                                                    //0x00
RXF2SIDL = 0x40;                                //Standard Address Filter-2 low byte set to 0x40
                                                    // (ID= 0x02)

TXB0DLC = 0x01;                                // Length = 1 byte
TXB0DLCbits.TXRTR = 0x00;                         // Setting of RTR bit at 1
TXB0SIDLbits.EXIDE = 0x00;                         // Setting of transmission bits
TXB0SIDH = 0x00;                                // Identifier = 5
TXB0SIDL = 0xA0;                                // if TXB0SIDL=0x40 -> identifier = 2

TXB0CONbits.TXPRI0 = 0x01;                         // Priority bit
TXB0CONbits.TXPRI1 = 0x01
TXB0D0 = value;                                 // Filling the first register of TX buffer 0

TXB0CONbits.TXREQ = 1;                            // Set the bit flag for the transmission
                                                    // request
while (TXB0CONbits.TXREQ != 0) {                  // Wait for the CAN TX buffer to be ready
    if (TXB0CONbits.TXERR == 1){                   // Check if there is a bus error in the
        LED2 = 1;                                // transmission of the CAN message
    }
}
```

Nella foto successiva si possono osservare i due messaggi di lunghezza pari ad un byte ($DLC = 1$) con diverso identificativo che sono presenti sul bus CAN e visualizzati tramite un programma dedicato del CAN sniffer (Figura 2.18).

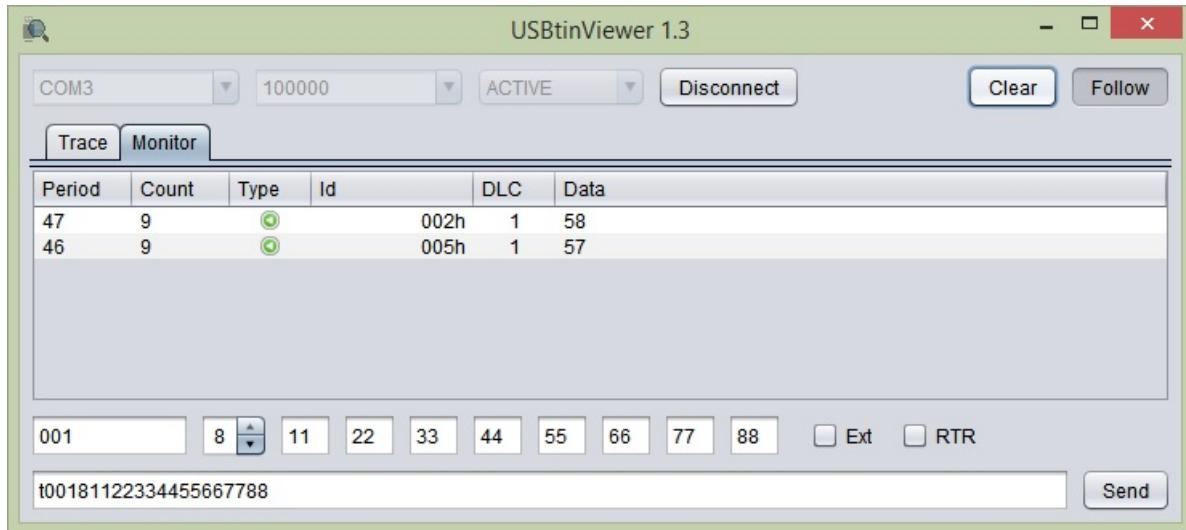


Figura 2.18: visualizzazione messaggi tramite CAN Sniffer.

2.5 DRIVER MOTORI

I driver utilizzati nell'UniMoRover sono gestiti dal microcontrollore PIC#2, l'unità principale di elaborazione dei dati. Il firmware è diviso in due blocchi: il primo gestisce la generazione dei segnali PWM e dei segnali di controllo necessari al controllo dei driver mentre il secondo gestisce l'acquisizione del valore di corrente istantaneo generato dal driver.

2.5.1 CONTROLLO E GESTIONE DEI MOTORI

I driver necessitano di due segnali PWM utilizzati per modulare la velocità di rotazione del motore e creati dai moduli CCP (*Capture, Compare and PWM*) del microcontrollore. Questi moduli sfruttano un timer interno per generare un'onda quadra con periodo costante e duty cycle predefinito. Ogni modulo CCPx va configurato nella modalità “PWM” mediante il registro *CCPxCON*.

Per questo progetto si è scelto di sfruttare i moduli CCP3 e CCP4 per ottenere due onde sincronizzate sullo stesso contatore. Quest'ultimo viene definito nel registro *CCPTMRS*, contenente un bit per ognuno dei moduli CCP presenti: in questo modo si può selezionare il timer voluto. La scelta è ricaduta sull'utilizzo del timer 4 poiché il timer 2 è utilizzato dalla centralina per la misurazione della distanza dagli ostacoli.

La frequenza di oscillazione è quindi definita tramite la seguente formula:

$$T_{PWM} = \frac{[(PR4) + 1] \cdot 4 \cdot (TMR4 Prescaler Value)}{f_{osc}} \quad [ms]$$

Questi registri vengono settati nella funzione “*set_PWM.h*” in modo da ottenere un segnale con frequenza f=500Hz. I valori utilizzati sono definiti nella Tabella 2.3.

Item	Value
PR4	249
TMR4 Prescaler	16
fosc	8Mhz

Tabella 2.3: valori per la configurazione del PWM.

Il duty cycle è definito dalla seguente relazione:

$$D_{PWM} = \frac{[CCPRxL : CCPxCON < 5:4 >] \cdot (TMR4 Prescaler Value)}{f_{osc}} \quad [ms]$$

Si è scelto di non considerare i bit meno significativi nella selezione del duty cycle, ovvero $CCPxCON<5:4>$. In questo modo si può modificare il valore D semplicemente variando il registro $CCPRxL$.

La gestione del duty cycle è demandata alla funzione “`set_duty.h`”, la quale utilizza come ingresso il valore voluto di D, lo trasforma in un valore binario a 8 bit e lo sovrascrive nel registro $CCPRxL$.

I due driver vengono comandati con duty cycle variabile: si è scelto di generare una rampa lineare, partendo da un valore minimo ed incrementandolo ad ogni ciclo di main fino al raggiungimento di un valore massimo.

La rotazione dei motori viene gestita da ogni driver mediante due segnali di controllo, *INA* e *INB*. Il PIC#2 è in grado di fornire i valori logici idonei (Tabella 1.2) in modo da ottenere il corretto movimento dell’UniMoRover. In caso di moto rettilineo, entrambi i motori sono pilotati nello stesso verso di rotazione, permettendo sia il movimento di avanzamento che la retromarcia.

La frenata inizialmente veniva effettuata impedendo la circolazione della corrente nei motori, i quali si fermavano per inerzia scaricando la corrente immagazzinata attraverso i low side mosfet del driver. I primi test sperimentali hanno reso necessario l’implementazione di una frenata “assistita”, in modo da ridurre i tempi di arresto del rover. Questa funzione è stata ottenuta forzando una breve rotazione nella direzione contraria a quella di marcia.

In caso di cambiamenti di direzione, essendo le ruote anteriori fissate, la centralina attua un controllo simile a quello utilizzato nelle macchine agricole cingolate: la ruota interna viene movimentata con una rotazione contraria rispetto alla ruota esterna. In questo modo UniMoRover è in grado di girare su sé stesso e di evitare gli ostacoli presenti sul percorso.

I valori massimi di duty cycle sono stati determinati tramite test empirici. La scelta è stata effettuata tenendo sotto controllo la corrente circolante nei motori e valutando la velocità massima raggiungibile dal rover.

2.5.2 CURRENT SENSE

I driver utilizzati consentono il monitoraggio istantaneo della corrente circolante nei motori ad essi collegati. Questi segnali, opportunamente condizionati, sono collegati ad ingressi analogici e vengono convertiti in un valore digitale dall’ADC integrato a 10 bit. Il convertitore è configurato tramite la funzione “`set_ADC.h`” ed effettua un interrupt ogni volta che il dato è completamente convertito.

PIC#2 contiene al suo interno solamente un convertitore analogico/digitale: dovendo campionare due segnali, si è deciso di sfruttare una gestione a token. La scelta dell’input analogico da convertire è cambiata ad ogni interrupt: questo avviene modificando il registro *ADCON0*. Così facendo si convertono i valori di corrente di entrambi i driver sfruttando un singolo ADC.

Questi valori sono molto importanti in ottica Fault Tolerance poiché, insieme ai segnali di diagnosi *DIAGA/DIAGB* dei singoli driver, permettono di valutare lo stato in cui lavorano i motori. In caso di errori temporanei o correnti troppo elevate il software taglia l’alimentazione ai due motori e si invia il segnale di errore motore all’FPGA per essere visualizzato sul monitor VGA rendendo cosciente l’utente dell’errore avvenuto. Il rover rimane in questo stato fino a che l’errore non è stato ripristinato.

3 CONCLUSIONI E SVILUPPI FUTURI

Molto spesso non si pensa a tutto quello che c'è dietro un semplice gioco per bambini.

Ripercorrendo lo sviluppo e la progettazione di UniMoRover, si possono definire parecchie motivazioni per cui questo progetto è risultato utile in ottica futura per la nostra carriera da Ingegneri.

Tra le principali possiamo citare l'approccio prettamente pratico voluto dai docenti e la possibilità di poter sviluppare un nostro sistema, consolidando le conoscenze acquisite in questi anni in ambito elettronico ed arricchite da quanto di nuovo appreso in questi mesi, come ad esempio la programmazione di FPGA e microcontrollori, l'implementazione dei protocolli di comunicazione CAN e la realizzazione di una comunicazione wireless utilizzando il transceiver Nordic *nRF24L01* comunicante tramite SPI.

Il progetto è servito inoltre a migliorare le nostre capacità di teamwork, un aspetto molto importante per il nostro futuro lavorativo in azienda.

La complessità del progetto ed i problemi incontrati in fase di realizzazione non ci hanno consentito di implementare alcune idee scaturite nella nostra mente durante lo sviluppo, che quindi verranno elencate di seguito come possibili miglioramenti del progetto:

- sostituire la pulsantiera integrata nell'FPGA con un joystick al fine di pilotare il rover in maniera più efficace;
- sviluppare una comunicazione diretta tra il modulo Nordic *nRF24L01* e FPGA, senza utilizzare un microcontrollore posto tra i due dispositivi;
- implementare un controllo della direzione del rover avanzato, inserendo un motore elettrico sullo sterzo e controllandolo tramite sensori ed elettronica ad hoc: in questo modo si avrebbe una gestione del moto più completa ed efficiente.

Un doveroso ringraziamento va ai Professori Bertacchini, Santinelli e Vezzani per il supporto fornito in questi mesi nonché all'Ing. Dott. Moreno Maini per la sua completa disponibilità durante la realizzazione del progetto.