TRANSISTORs TEAM

CLASSE V°C

I.T.C.S E. da ROTTERDAM
BOLLATE (MI)

Laboratorio Permanente di Robotica Didattica

“ La scienza, qualsivoglia siano i suoi ultimi sviluppi, ha le sue origini nelle tecniche, nelle arti e nei mestieri… la scienza emerge nel contatto con le cose, dipende dall’evidenza dei sensi, e, per quanto sembri allontanarsi da essi, sempre ad essi deve tornare”
[Farrington, 1949]

RESCUE TEAM

CAP.1 DATI GENERALI

1.1 La squadra e i suoi componenti

Studenti:

- Bonetti Michael
- Costantini Steven
- Ignazzi Marco
- Marani Simone
- Mazzara Simone

Docente accompagnatore :

Ing. Prof. Giulio Vitale

Appartenenti alla classe V ° C dell'indirizzo Informatica e Telecomunicazioni
dell'Istituto I.T.C.S. Erasmo da Rotterdam, Bollate (MI)

CAP.2 – DATI DI CONTESTO E MOTIVAZIONE

L'idea di partecipare alla Robocup Jr Italia ha iniziato a prendere vita fin dal primo anno in cui questa manifestazione si è tenuta in Italia, ovvero la RCJ svoltasi a Torino nel 2009.
In quell'anno il nostro Istituto poté fruire di un finanziamento all'interno di un programma generale denominato Scuole Aperte, che si proponeva lo scopo di estendere l'offerta formativa della scuole con attività integrative pomeridiane. Tra le varie attività finanziate erano presenti quelle che proponevano interventi mirati ad approfondire l'interesse nell'ambito delle discipline tecniche e scientifiche.

Fu proprio nel 2009 che il Professor Giulio Vitale pensò di istituire all'interno dell'ITCS Erasmo da Rotterdam un "Laboratorio Permanente di Robotica Didattica" con lo scopo di creare uno spazio attrezzato nel quale si potesse sperimentare una nuova forma didattica, scelta volontariamente, in cui ribaltare il classico rapporto docente-studente. In questo laboratorio, gli studenti diventano i veri sperimentatori, ovvero gli attori principali che decidono le linee generali di ricerca e di sperimentazione, si organizzano in "team" per essere in grado di affrontare e risolvere i problemi legati agli obiettivi scelti. Il rapporto con il docente diventa simile a quello normalmente instaurato con un consulente esperto, in grado di offrire le tecnologie e gli strumenti di base necessari a perseguire gli scopi della ricerca e con il quale discutere e individuare le linee guide.
In quest'ambito il Prof. Vitale propose ad alcuni giovani di terza e di quarta dell'indirizzo di informatica e telecomunicazioni di provare a realizzare un robot auto-costruito capace di seguire una linea ed individuare eventuali vittime lungo il suo percorso, per aderire alla prima manifestazione della Robocup, che in quell'anno si sarebbe svolta a Torino.
Ebbe inizio così una lunga serie di lavori, dei quali, forse, all'inizio, non si aveva neppure il lontano pensiero della loro esistenza!
Un'impresa quasi "titanica" che accompagnò questa squadra a Torino e che la portò a presentarsi come uno dei pochissimi team che partecipavano con un robot auto-costruito.

L'idea portante fu quella di proseguire un'esperienza nata nel corso di Elettronica Digitale, cioè di usare le logiche programmabili come tecnologie fondanti per la realizzazione della piattaforma di base, proprio per avere a disposizione un ambiente flessibile, da modellare alle nostre esigenze, in cui ogni singolo organo d'interfaccia, fosse discusso e determinato collettivamente, prima di essere implementato. Il docente si sarebbe offerto di fornire al "team" la sua esperienza per la definizione del microcomputer principale, pensato e realizzato internamente, in maniera analoga a quella che si sarebbe avuto prendendo un microcomputer già realizzato esternamente, ma con in più il vantaggio di vederlo nascere contestualmente al proprio lavoro.
La prima esperienza del 2009 non ebbe un successo eclatante, anzi il robot non fu in grado nemmeno d'iniziare il percorso: tanti erano stati le questioni da affrontare e cercare di risolvere che si arrivò al momento della gara con un sistema solo abbozzato, non ancora pronto a competere pubblicamente.
Ma quello fu solo l'inizio e il dato era tratto: i robot della Lego sarebbero stati usati come primo approccio alla robotica didattica e la squadra dei più giovani si sarebbe cimentata con quelli. Rimaneva comunque l'idea di continuare a far evolvere il nostro progetto iniziale e di renderlo man mano più competitivo.

In questa esperienza ormai triennale risiede tutto il senso del nome che la squadra scelse prima di partire: Transistors Team.
Il transistor come ben sappiamo è un piccolo oggetto capace di tanto, difficile da usare, detta le condizioni per ottenere il giusto lavoro, insomma un oggetto dalle tante difficoltà di impiego ma che vuole lasciare il segno indelebile della sua presenza.
Così è stato scelto di valorizzare questo incredibile componente elettronico e renderlo la piccola mascotte del gruppo, nell'impegno ad un vero lavoro di TEAM.

Dopo la clamorosa uscita di gara agli inizi, il robot è divenuto modello di esperienza per la nuova versione che ha gareggiato a Vicenza nel 2010, questa volta però da due ruote motrici si è passati ad un cingolato, anch'esso interamente auto-costruito.

Il tempo ha insegnato molto alla squadra e il 2010 ha portato un grande successo, seppur non qualificandosi, alla seconda versione del robot che a Vicenza riuscì a superare bene le prime due parti dell'arena ma fermandosi in cima alla rampa per difetto di potenza meccanica e di equilibrio costruttivo. Senza scoraggiamento, il 2011 apre alla nuova gara a Catania, con *NESSY*, nome di battesimo del nuovo cingolato.

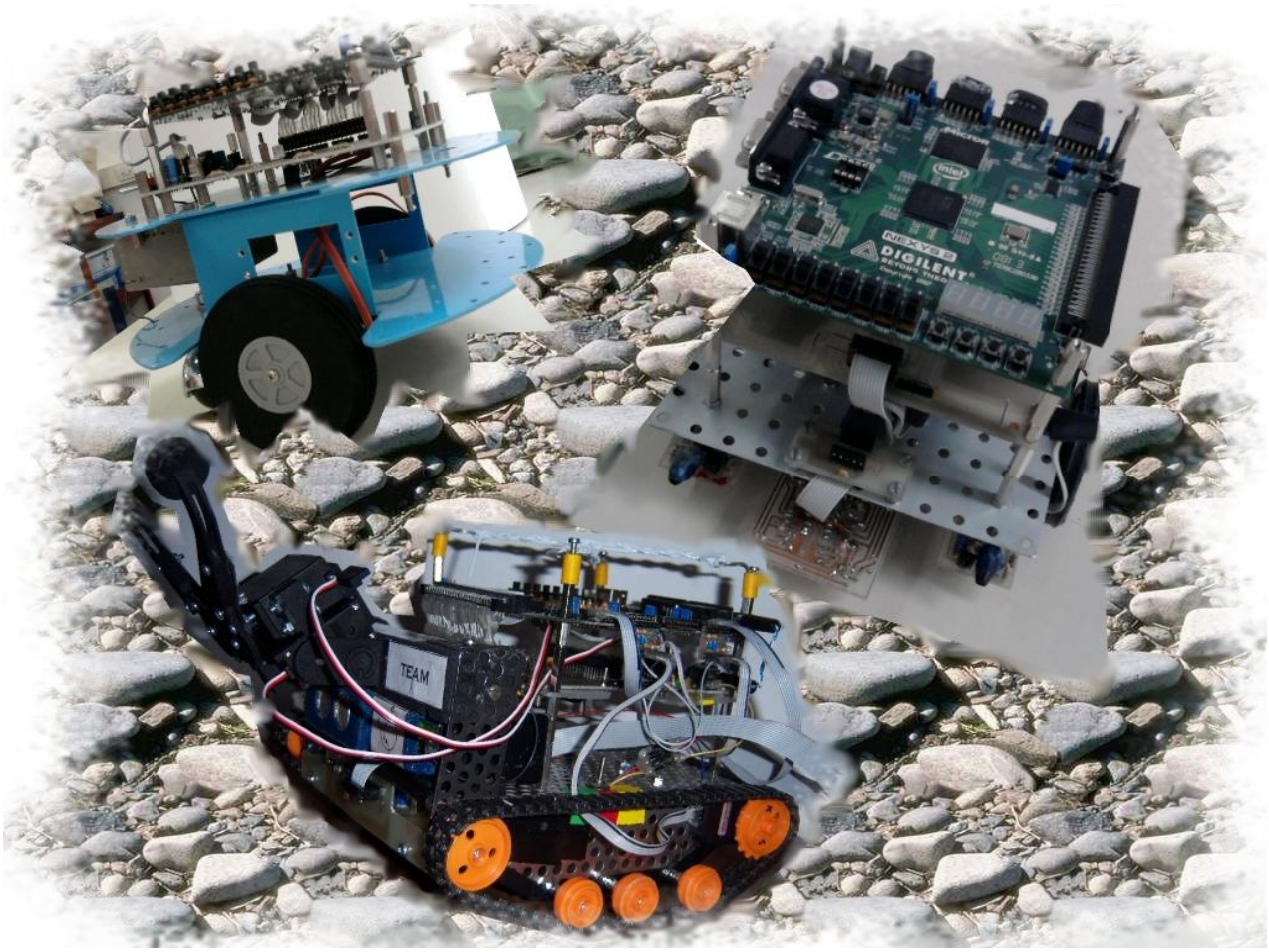
I passi che sono stati fatti, riflettono tutto l'impegno e la costanza che la sinergia tra il Professor Vitale e i suoi studenti hanno avuto gli anni scorsi, ma, proprio perché l'esperienza resta un fattore particolarmente importante, quest'anno, il Transistors Team ha scelto di chiedere, per la parte meccanica (poiché la scuola non ha la strumentazione sufficiente), aiuto ad alcune aziende del territorio. La società Futura Elettronica ha risposto all'appello, fornendo materiali e risorse tecniche e umane che si sono messe a disposizione per favorire un lavoro che vuole allargare gli orizzonti e condividere le proprie conoscenze con gli altri ragazzi che concorrono per crescere ed imparare dalla robotica.

Come in ogni team, si evidenzia la difficoltà del lavoro di gruppo, ma anche la tenacia e la collaborazione che ogni membro ha per comprendere il robot in ogni sua parte, ma soprattutto per migliorare il proprio modo di porsi nei confronti del “collega”.

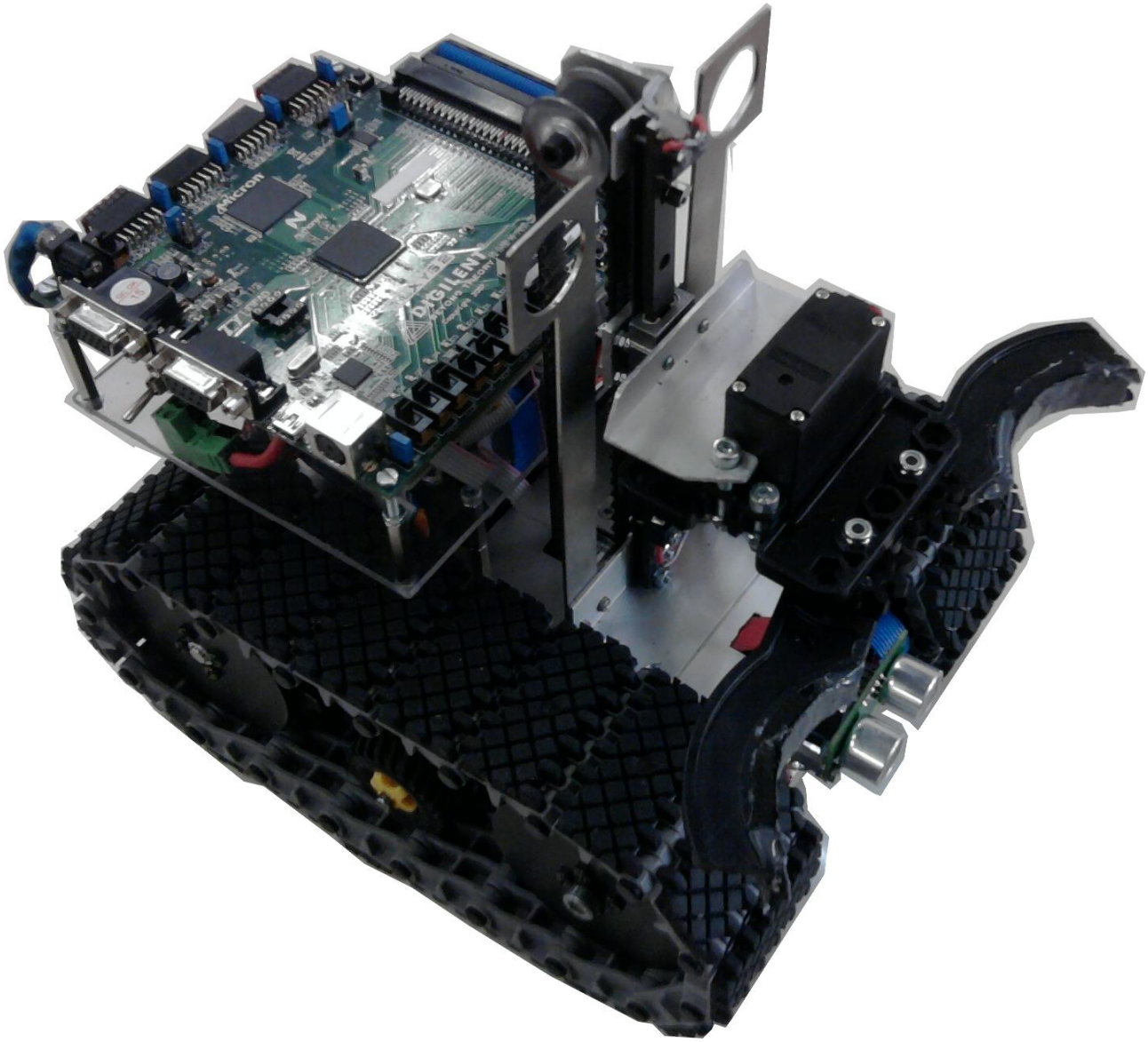
A questo progetto, inoltre, hanno supportato la squadra il professor Filippo Bilardo, docente dell'attuale V°C, tutti i professori che chiedendo e informandosi sull'andamento dei lavori spronano sempre il Team a proseguire nel suo progetto e la scuola, che, nonostante qualche difficoltà amministrativa, ha permesso al Gruppo di lavorare.

Il Team vuole ringraziare tutti coloro che hanno aderito con passione, sostegno, impiego di tempo e denaro, per la loro disponibilità e presenza necessaria che hanno fatto giungere il Transistors Team a questo livello.

L'EVOLUZIONE DELLA SPECIE...



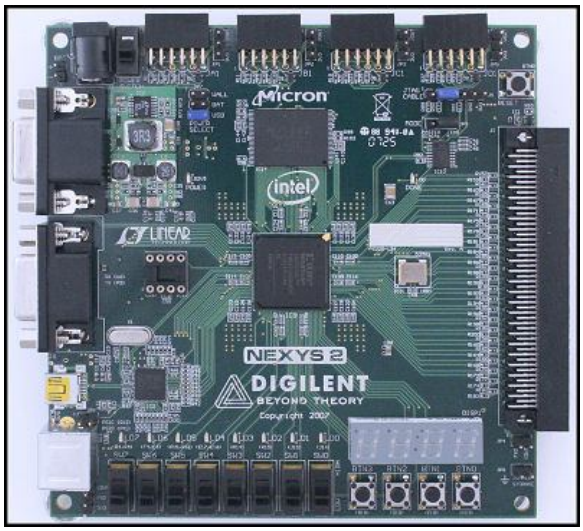
...L'ULTIMO
IL SAPIENS SAPIENS...



CAP.3 – NOME E STRUTTURA DEL ROBOT

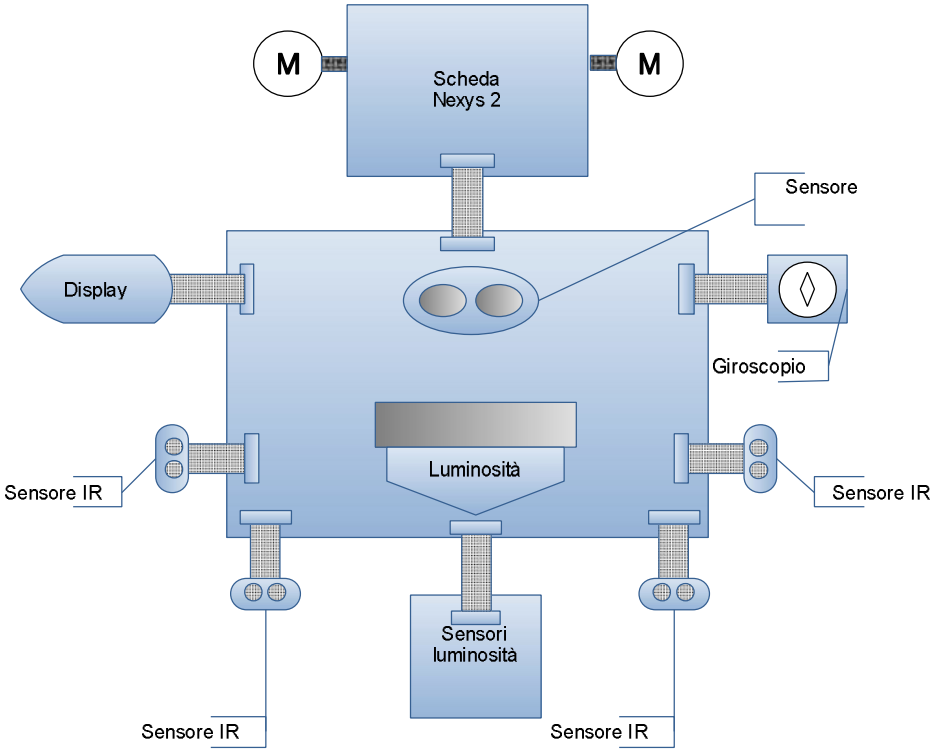
Il nome del nostro robot è Nessie nome facilmente riconducibile al leggendario mostro di Loch Ness, il quale infesterebbe un lago della Scozia.
La scelta è stata presa dal gruppo in quanto nella sua fase iniziale, il nostro robot era costituito solo dallo chassis e dalla pinza e ciò lo faceva rassomigliare al suddetto mostro.
Quest'anno, infatti, siamo partiti subito con la soluzione meccanica adatta al salvataggio della vittima per prevenire i problemi di assemblaggio dell'anno precedente, quando cercammo di inserire una pinza su una struttura non pensata a riceverla.

Nessie, comunque, mantiene la nostra architettura iniziale, scelta tre anni fa, e che ha garantito la possibilità di far evolvere la nostra macchina, sia dal punto di vista della tipologia di sensori da connettere sia dal punto di vista della loro gestione, lasciando immutata la piattaforma di base, formata dalla CPU e da tutti i circuiti d'interfacciamento tra questa e le periferiche di input/output.
L'idea iniziale di scegliere un'architettura flessibile, basata sull'hardware riconfigurabile di una FPGA (Field Programmable Gate Array), ha dato i suoi frutti, fornendo la possibilità di adattarsi ai cambiamenti successivi.
L'architettura che caratterizza Nessie, si basa su una scheda madre Nexys, fornita dalla società Digilent Inc., sulla quale è presente una FPGA Spartan3E – 1200, insieme a un primo insieme di risorse di base di tipo general purpose:



- 8 dipswitch;
- 4 pulsanti;
- 8 led;
- 4 display a 7 segmenti;
- 16 MB di Flash;
- 16 MB di RAM;
- 1 porta USB, utile alla configurazione della FPGA e per scambiare dati con un host;
- 1 porta RS232;
- 1 porta PS2;
- 1 interfaccia VGA;
- 16 IO disponibili su connettori separati Pmod e 43 su connettore Hirose;
- un generatore di clock a 50 Mhz.

Alla scheda madre viene collegata una scheda generale d'interfaccia che funge di canale d'interconnessione tra la scheda madre e le varie schede di gestione dei singoli sensori. La struttura complessiva appare come nel seguente schema a blocchi generale, in cui sono indicate le singole parti fisiche del sistema, così come inizialmente concepite.



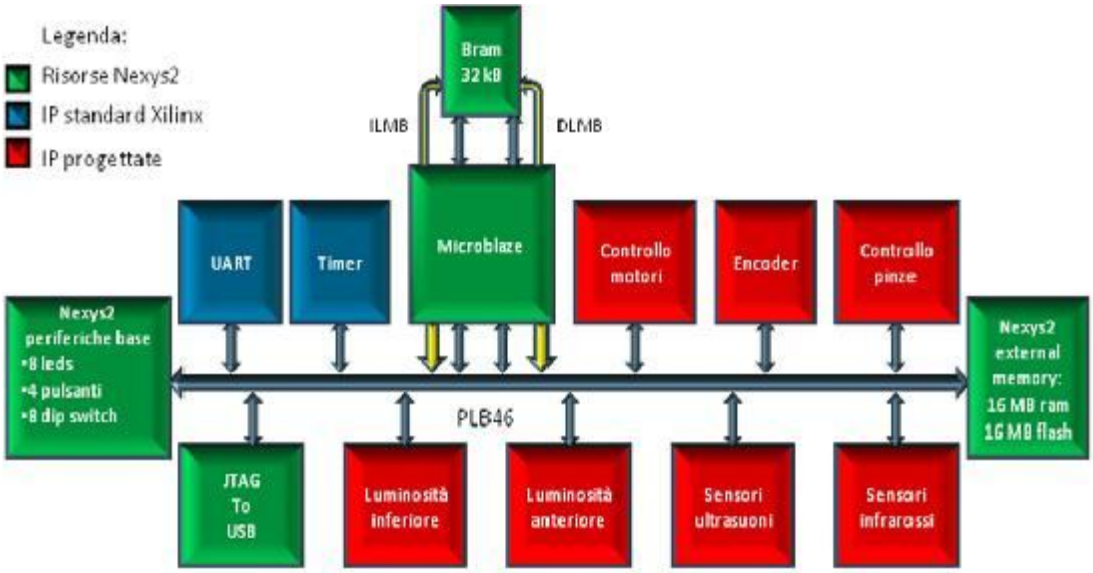
La versione attuale di Nessie, per necessità di tempo, mostra alcune differenze:

- Il giroscopio è stato sostituito con un circuito di Tilt realizzato con un interruttore al mercurio che è risultato molto più semplice da realizzare e integrare;
- i sensori a infrarossi sono stati momentaneamente omessi, per questioni legate alla loro sistemazione nello chassis attuale;
- il display è stato previsto, acquistato, provato ma non integrato nel software attuale perché non si è dimostrato necessario nella fase attuale; sarà previsto il suo montaggio nella prossima versione.

3.1 Struttura del microcomputer interno alla FPGA

L'intero sistema di elaborazione dei dati è contenuto all'interno della FPGA ed è basato su una CPU RISC a 32 bit, denominata Microblaze, fornita da Xilinx come “soft core” riconfigurabile secondo le esigenze dell'utente a cui poter connettere sia delle periferiche standard, offerte dal costruttore (memorie, timer, UART, porte parallele di I/O, interrupt controller, …), sia delle periferiche progettate ad hoc in funzione dei sensori specifici.

Il sistema di elaborazione, assume così la seguente architettura logica.



Per Catania abbiamo deciso di attrezzare Nessie con i seguenti sensori:

- sensore di luminosità inferiore, per l'inseguimento della traccia;
- sensore di colore, per identificare condizioni di riflessione diverse ed eventuali vittime di colore diverso dallo sfondo;
- sensore a ultrasuoni, per l'identificazione degli ostacoli e della vittima;
- sensore “tilting”, per l'identificazione della rampa e del suo completamento.

Gli attuatori da noi controllati sono quattro:

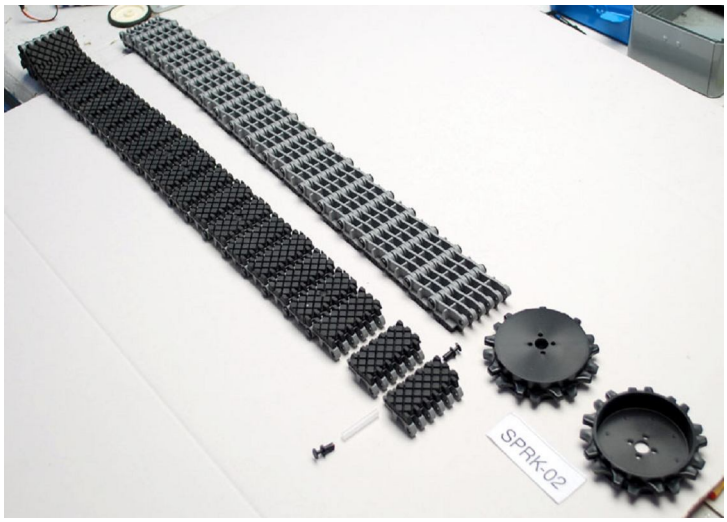
- due motori in corrente continua per l'avanzamento del robot;
- un motore in corrente continua per il sollevamento e il deposito della vittima;
- un servo motore per l'apertura e la chiusura delle pinza.

CAP.4 MECCANICA

Per quanto riguarda la meccanica, il nostro robot NESSIE, è stato interamente progettato internamente, anche se con il supporto della Futura Elettronica che ci ha aiutato sia nella fase di definizione di alcuni organi meccanici, come la pinza, sia nel procurarci gli strumenti di lavorazione che mancano del tutto nella nostra Scuola.
Di seguito sono esposti i principali componenti meccanici del robot e la sequenza delle fasi salienti dell'assemblaggio delle varie parti.

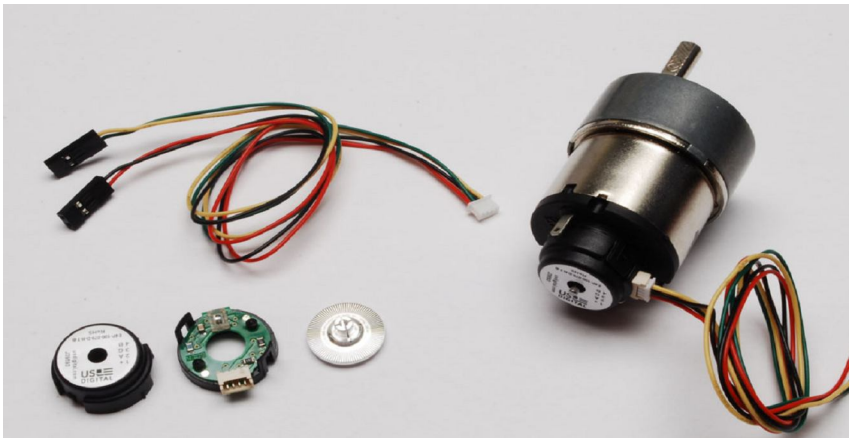
4.1 La trazione

Dopo l'esperienza dello scorso anno, non del tutto soddisfacente rispetto alle prestazioni meccaniche, si è deciso di cercare componenti più professionali che ci garantissero un comportamento più sicura sulla rampa. Abbiamo trovato la soluzione attuale presso la società Lynx Motion che produce componenti e kit per robot didattici di qualità.
Il sistema di trazione del nostro robot è composto da dei cingoli in plastica ricoperti da della gomma tassellata nella parte esterna per evitare che il robot abbia problemi di scivolamento soprattutto nei tratti più tortuosi. I cingoli sono supportati da una ruota di diametro inferiore, interposta tra le due ruote principali dei cingoli alle quali sono collegati i due motori, per aumentare la rigidità dei cingoli nella parte centrale.

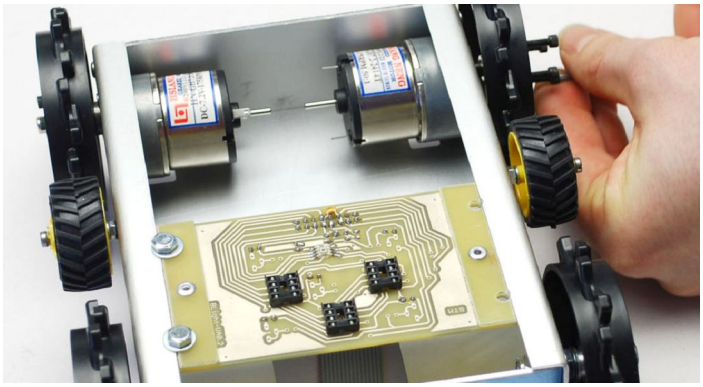


4.2 I motori

Anche la scelta dei motori è stata influenzata dalle vicende precedenti e dalla scarse prestazioni che erano state riscontrate nel precedente modello che utilizzava una coppia di motoriduttori della Tamiya, che si sono rivelati non adeguati alle dimensioni e al peso del nostro robot. Per Nessie abbiamo pensato a qualcosa di più solido e la soluzione ci è stata fornita ancora dalla stessa Linx Motion con la seguente coppia di motori.



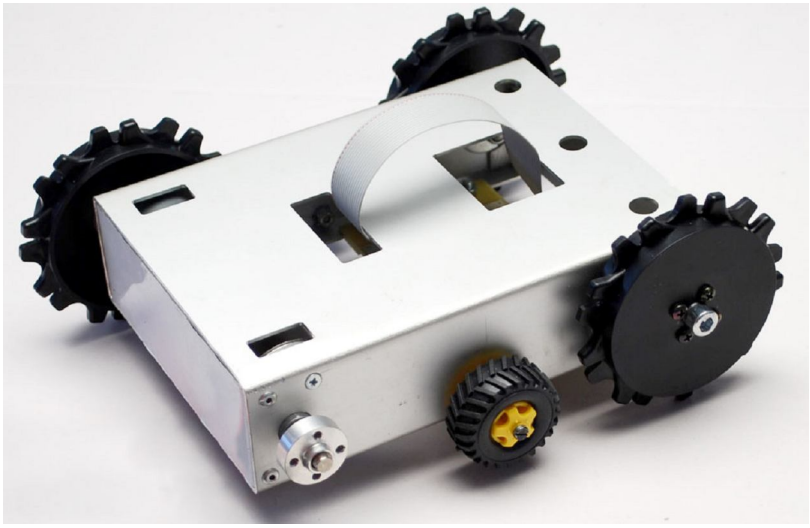
I motori sono stati ubicati all’interno dello chassis progettato interamente da noi del TRANSISTORs TEAM. Sono stati posti nella parte posteriore del robot pertanto la trazione e sulle due ruote posteriori.



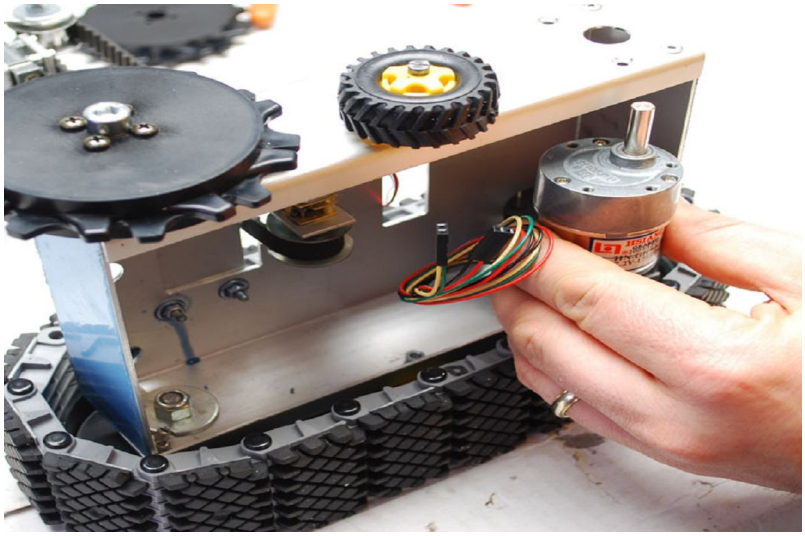
Particolare del fissaggio degli encoder ai motori

4.3 Lo chassis

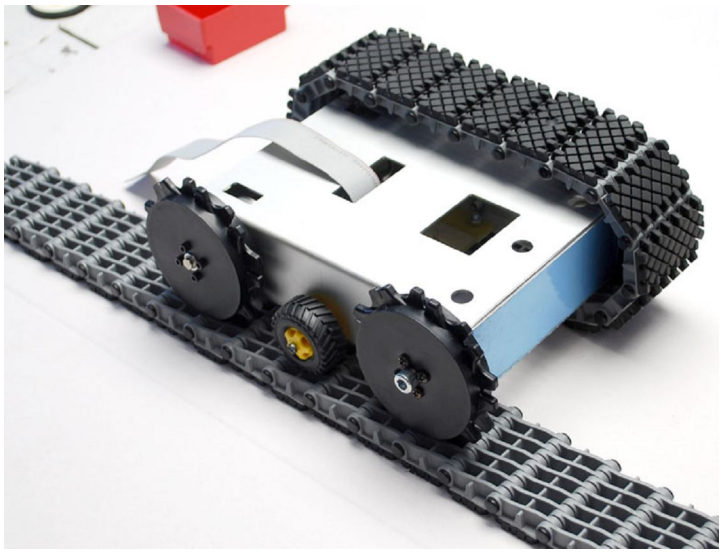
Per quanto concerne lo chassis come già detto è stato disegnato da noi in ogni sua parte, infatti sono state pensate tutte le misure più opportune alle nostre esigenze. Sono stati pensati anche tutti i fori che erano fondamentali per poter collegare i motori alle ruote e per far passare tutti i fili da collegare ai connettori delle varie schede.
Lo chassis è in alluminio e per la sua realizzazione fisica abbiamo richiesto il supporto della Futura Elettronica che ha sopperito alle nostre carenze di strumentazione meccanica e ha trasformato il disegno del nostro chassis nell'oggetto che appare in figura.



Il fissaggio dei motori allo chassis.



Lo chassis completo montato sui cingoli



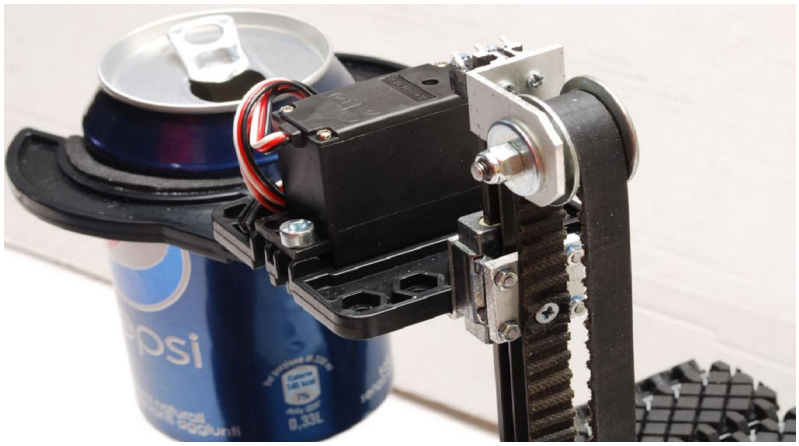
4.4 La pinza

Per il sistema di recupero della “vittima” è stata scelta una pinza prodotto dalla POB che per conformazione sembrava adatta al recupero della vittima, ma il meccanismo di elevazione era completamente fuori misura con le specifiche del percorso di gara.

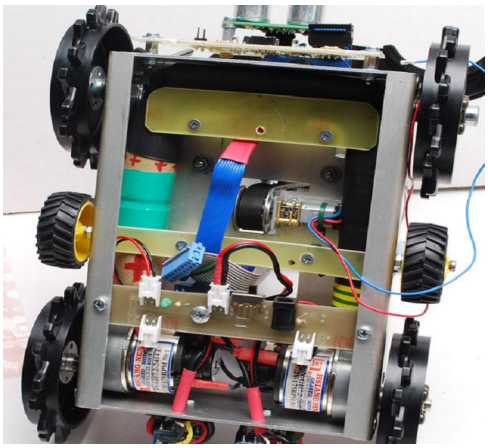
Anche in questo caso abbiamo dovuto far ricorso alle risorse della Futura Elettronica, insieme alla quale abbiamo pensato di realizzare una struttura basata su una guida lineare DIN della Drylin da 9 mm di larghezza, trovata nel catalogo della RS Components che sembrava adatta al nostro caso. Sulla slitta è stata fissata la pinza e tramite una cinghia dentata, che scorre su due pulegge poste agli estremi, è

stata collegata ad un piccolo motore in corrente continua, alimentato a 6V e dotato di un rapporto di riduzione di 280:1.

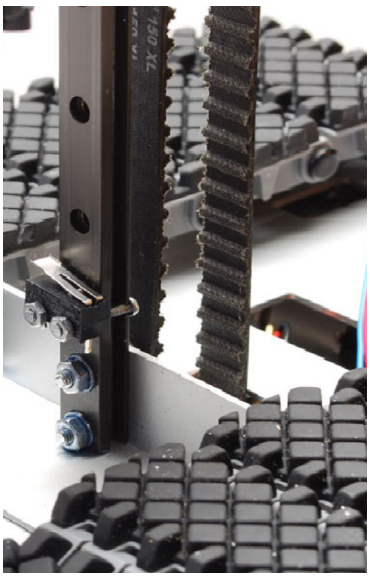
Particolare del fissaggio della pinza alla slitta e di questa alla cinghia.



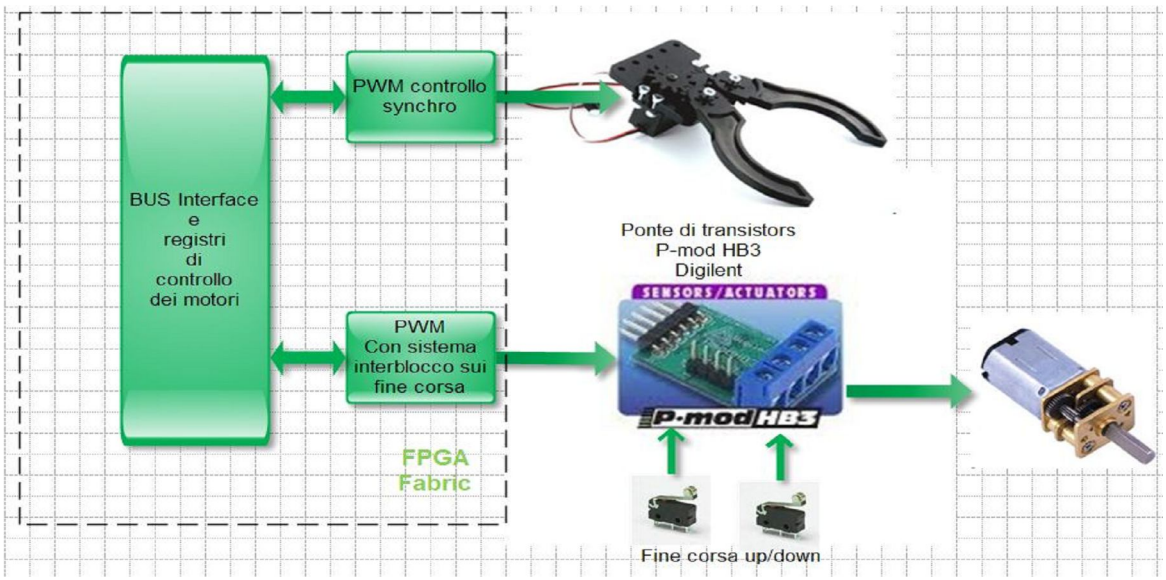
Particolare della connessione della cinghia al motore di sollevamento.



Il sistema di sollevamento è stato completato con due fine corsa realizzati con micro switch fissati agli estremi



Schema a blocchi complessivo del controllo della pinza.

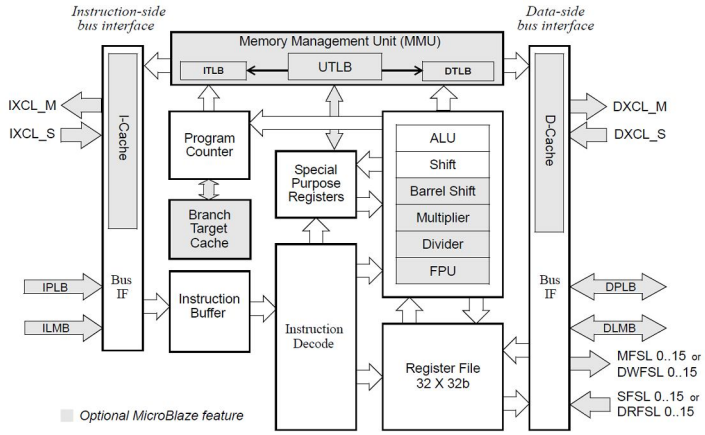


CAP.5 UNITÁ DI CONTROLLO

Il Microblaze, si presenta come una architettura Harward dotata di quattro bus principali separati, un set d’istruzioni Risc molto snello, con istruzioni uniformi e ortogonali da 32 bit ognuna, un BUS dati a 32 bit e 32 registri della medesima dimensione. Come si può vedere in figura, partendo dai blocchi fondamentali, indicati in bianco, e in funzione delle specifiche esigenze, il “core” si espande con i blocchi, indicati in grigio nella figura, che possono man mano essere aggiunti per adattarlo alle particolari prestazioni richieste, senza dover ritoccare l’intero sistema.
In particolare i moduli che possono essere successivamente connessi sono i seguenti:

- una MMU derivata dal modello Power PC 405, che può supportare “kernel”, come quello di Linux, che ne esigono la presenza per il controllo della memoria virtuale,
- una cache dedicata ai salti, associata a un sistema di previsione, per limitare il calo delle prestazioni dovute alle possibili condizioni di stallo della pipeline nei “branch”,
- due memorie cache, una per le istruzioni e una per i dati, dimensionabili, ognuna, da 64 a 64K byte a scelta del progettista, e connesse con un BUS separato,
- fino a 16 canali di interfacciamento in input/output, gestibili tramite link seriali (FSL), con un meccanismo simile a una FIFO, accessibili direttamente con istruzioni dedicate, per la realizzazione di coprocessori dedicati, con bassa latenza nel trasferimento dei dati,
- moltiplicazione e supporto alla divisione implementati in hardware,
- Floating Point Unit a singola precisione compatibile con standard IEEE 754,
- Barrell shifter, per la realizzazione di shift multipli in un solo ciclo macchina.

Nell’applicazione che presentiamo, sono state selezionate soltanto le seguenti risorse basilari:



- i due doppi bus, istuzioni/dati, per l’accesso simultaneo alla memoria locale interna e alle periferiche, ovvero il BUS LMB, con 32kB di memoria interna accessibile con un solo ciclo di clock, e il bus PLB per l’accesso alla memoria esterna e alle periferiche,
- Floating Point Unit, moltiplicatore e divisore hardware, barrel shifter.

5.1 Le motivazioni alla scelta fatta

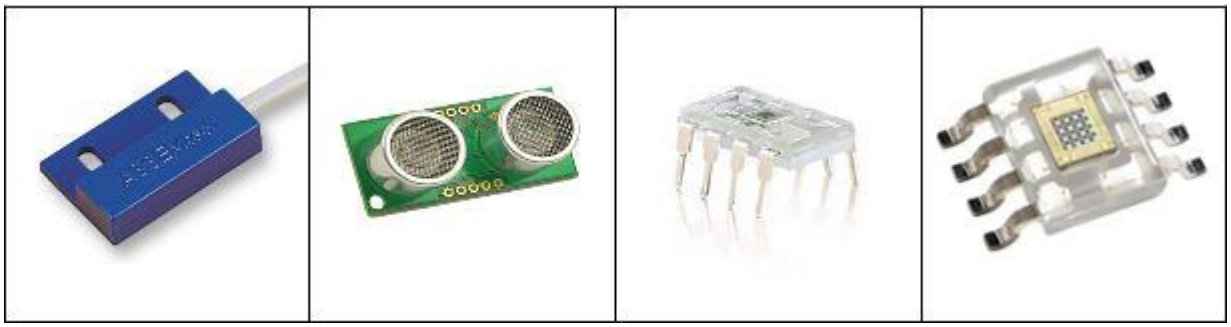
Quali sono i vantaggi dell’impiego di una simile architettura?

Innanzitutto il robot nasce per uno scopo didattico, quindi deve essere aperto, in modo che gruppi di studenti diversi possono pensare simultaneamente alla realizzazione delle diverse periferiche di controllo di sensori e attuatori, controllandone completamente le funzionalità, il progetto e la verifica;

Nel dimensionamento delle prestazioni, una FPGA offre costantemente la possibilità di scegliere la strada ottimale, in termini di efficienza di elaborazione, tra la realizzazione in hardware e quella in software di ogni funzione, eliminando tutti gli eventuali colli di bottiglia. Per esempio nel nostro caso è stato scelto un “tick” base di 10 millisecondi e, al massimo, in questo intervallo di tempo tutta la ricca dotazione di sensori deve essere stata letta e tutti gli attuattori devono essere aggiornati con i valori coerenti alle condizioni di controllo decise dal programma.

Lo stesso hardware deve essere riadattabile alle specifiche condizioni sperimentali che si vogliono affrontare di anno in anno in eventuali corsi didattici differenti, recuperando interamente le parti più costose per riutilizzarle in differenti esperienze.

CAP. 6 I SENSORI



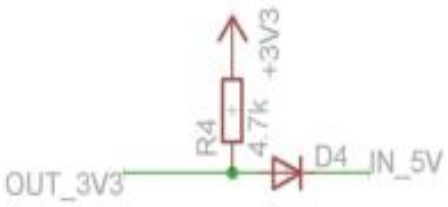
tilt switch | sensore ultrasuoni | sensore luminosità | sensore colore

Nessie è stato progettato in modo tale da poter affrontare tutte le difficoltà imposte dalla gara “Rescue”; quindi deve essere in grado di seguire un percorso più o meno impegnativo, con curve di qualsiasi angolazione, con brevi interruzioni della linea, con impedimenti lungo il tragitto quali macerie o veri e propri ostacoli, e deve poter essere in grado di affrontare salite impegnative.
Oltre al percorso deve essere capace di riconoscere la “vittima” in una stanza senza alcuna traccia o riferimento da seguire, di prenderla e di metterla in salvo in un'area sopraelevata della stessa stanza (area di salvataggio).
I sensori che noi abbiamo utilizzato per fornire all'unità centrale e quindi all'automa, tutte le informazioni che gli consentano di capire come muoversi e soprattutto quando fare determinate azioni, dipendono un po' dalla nostra esperienza e un po' dalla endemica mancanza di tempo che non ci ha permesso d'impiegare nel modello attuale di Nessie un sistema di rilevamento con quattro trasduttori d'infrarossi che avrebbe senz'altro completato e arricchito la dotazione attuale.
La versione di Nessie che ora presentiamo è provvista dei seguenti trasduttori:

- 1 rivelatore di rampa con un tilt Switch TSM 40/110 (rimpiazzabile con TSW 30/60)
- 1 sensore ad ultrasuoni (SRF05)
- 4 sensori di luminosità TSL230
- 1 sensore RGB di colore con package “SOIC8”

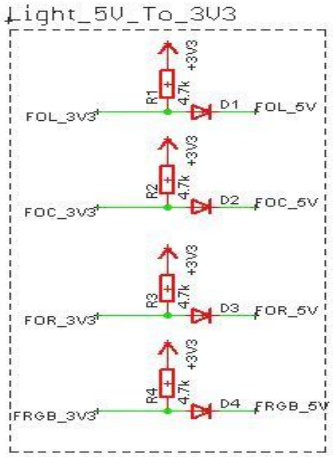
6.1 Collegamento con l'unità centrale

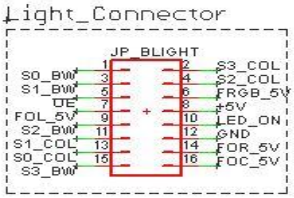
Prima di entrare nella spiegazione di ogni singolo sensore è importante spiegare come questi circuiti auto costruiti siano stati collegati all'unità centrale. Tutti i sensori, prima di giungere alla FPGA contenuta nel modulo centrale “Nexys 2”, passano per una scheda d'interfaccia che, oltre ad avere il compito di alimentare tutte le altre schede presenti sul robot, svolge l'importante funzione di mantenere uguale la frequenza del segnale che riceve, ma di convertirne l'ampiezza. Ad esempio se la scheda dei sensori di luminosità vuole trasmettere un segnale all'unità centrale, il segnale passa prima nella scheda d'interfaccia dove la frequenza di tale segnale verrà mantenuta integra, mentre l'ampiezza verrà portata dai 5 Volt ai 3,3 Volt; questo è dovuto al fatto che le schede dei sensori hanno bisogno di una alimentazione pari a 5 Volt mentre la Nexys 2 viene alimentata a 3,3 Volt.
Per evitare il danneggiamento delle celle d'input/output della FPGA, è stato opportuno utilizzare un semplice circuito “resistenza + diodo schottky (BAT46)” come indicato in figura 6.2 .
La necessità di usare un diodo schottky è derivata dalla tensione di saturazione molto bassa che li caratterizza.
Quando IN_5V, che rappresenta il segnale d'uscita di un sensore qualsiasi, è a un potenziale pari a 5V, il diodo si interdice, ovvero si polarizza inversamente, e il segnale OUT_3V3, input per la FPGA, risulta ancorato alla tensione di 3,3V.
Invece, quando IN_5V è a 0V, il diodo si polarizza direttamente, producendo una caduta di tensione diretta V_γ, alla quale risulterà ora vincolato il segnale OUT_3V3. Il basso valore della V_γ del diodo BAT46 garantisce così uno zero logico abbastanza vicino al potenziale di massa.



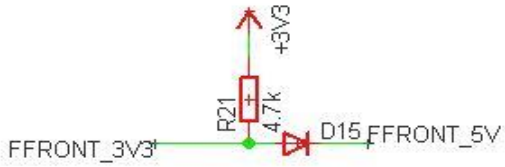
6.2 Scheda d'Interfaccia

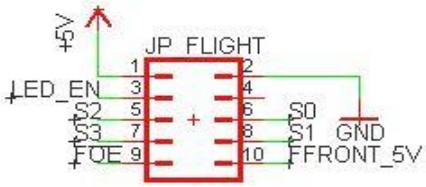
In questo paragrafo viene riportato lo schema elettrico della scheda d'interfaccia ma in diversi frammenti per rendere ogni singola maggiormente visibile, vista la grandezza dello schema.



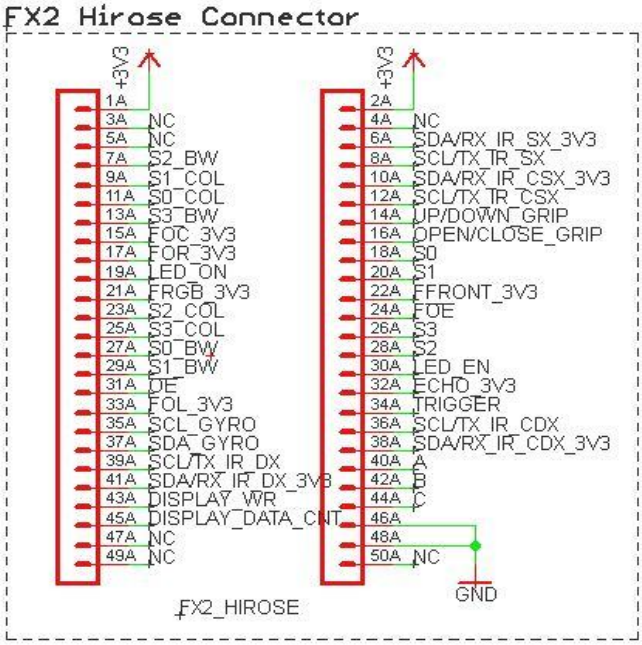


Nel primo riquadro a sinistra, sono presenti quattro convertitori 5V – 3,3V che collegano i quattro sensori presenti nella scheda bottom di luminosità alla FPGA; si tratta di 3 sensori TSL230 di luminosità ed un sensore RGB. A destra invece è riportato il connettore che si connette realmente i sensori con i vari convertitori.

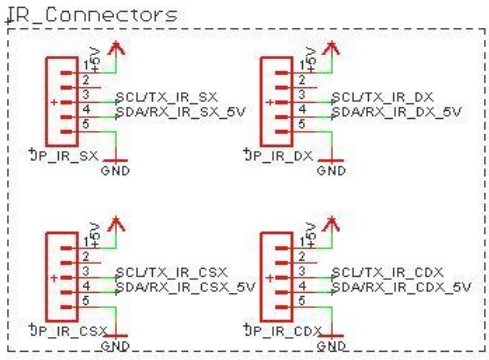


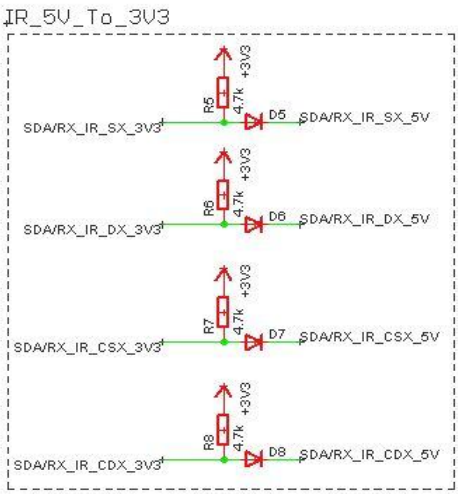


Stessa configurazione della figura 6.3, con l'unica differenza che questa volta si tratta del sensore di luminosità situato nella scheda frontale dell'automa.

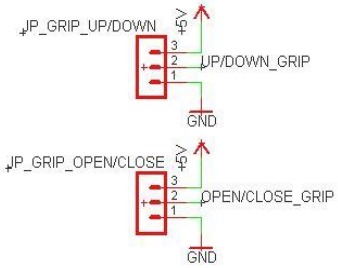


FX2 Hirose Connector è uno dei connettori più importanti del robot, perché attraverso questo connettore passano tutti i segnali dei vari sensori, ed è proprio grazie a questo componente che la FPGA può interagire con le varie periferiche.



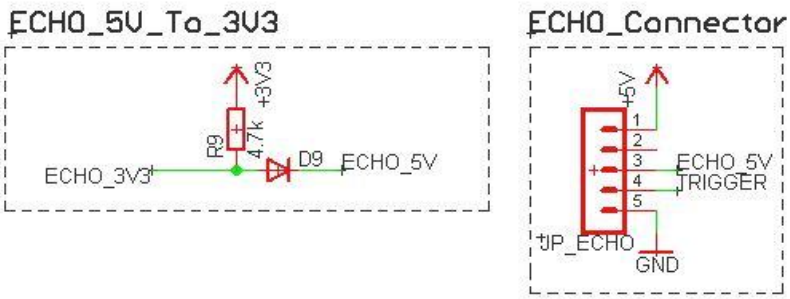


Questa parte della scheda sarebbe servita alla conversione di tensione e alla connessione con i sensori ad infrarossi. Per scelte del team quest'anno non sono stati inseriti nel robot, tuttavia uno di questi connettori è stato già impiegato nella connessione della FPGA con il rivelatore di rampa.



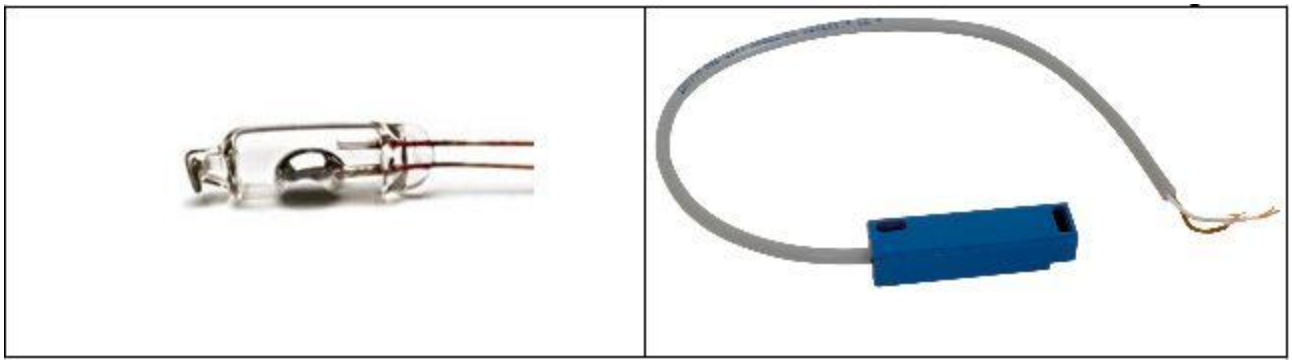
I due jumper sopra raffigurati connettono la Nexys 2 con i due motori della pinza: uno per far salire e scendere la pinza dal supporto su cui è stata montata e l'altro per far aprire e chiudere le “chele” della pinza.

Convertitore e jumper di connessione del sensore SRF05 ultrasuoni:



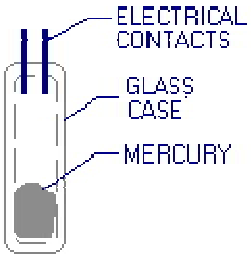
6.3 Rivelatore di Rampa

6.3.1 Circuito di tilting per la rilevazione della rampa



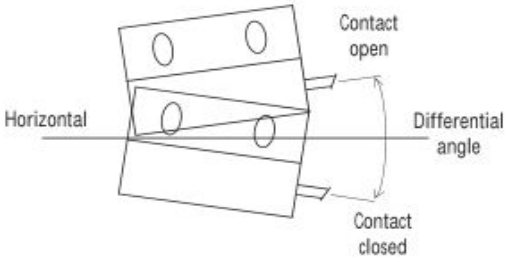
mercury contact

non-mercury contact



Il rivelatore di rampa è un circuito che, come dice il nome, serve per capire quando il robot si trova sulla rampa dell'arena di gara. Il circuito sfrutta un interruttore, solidale al robot che, in relazione alla sua inclinazione, chiude o apre un contatto, che può essere utilizzato
per segnalare l'evento alla CPU.
Attualmente il rivelatore di rampa ha sopra installato il tilt switch TSM 40/110, che è di forma identica all'interruttore di figura 6.9, ma con due differenze:

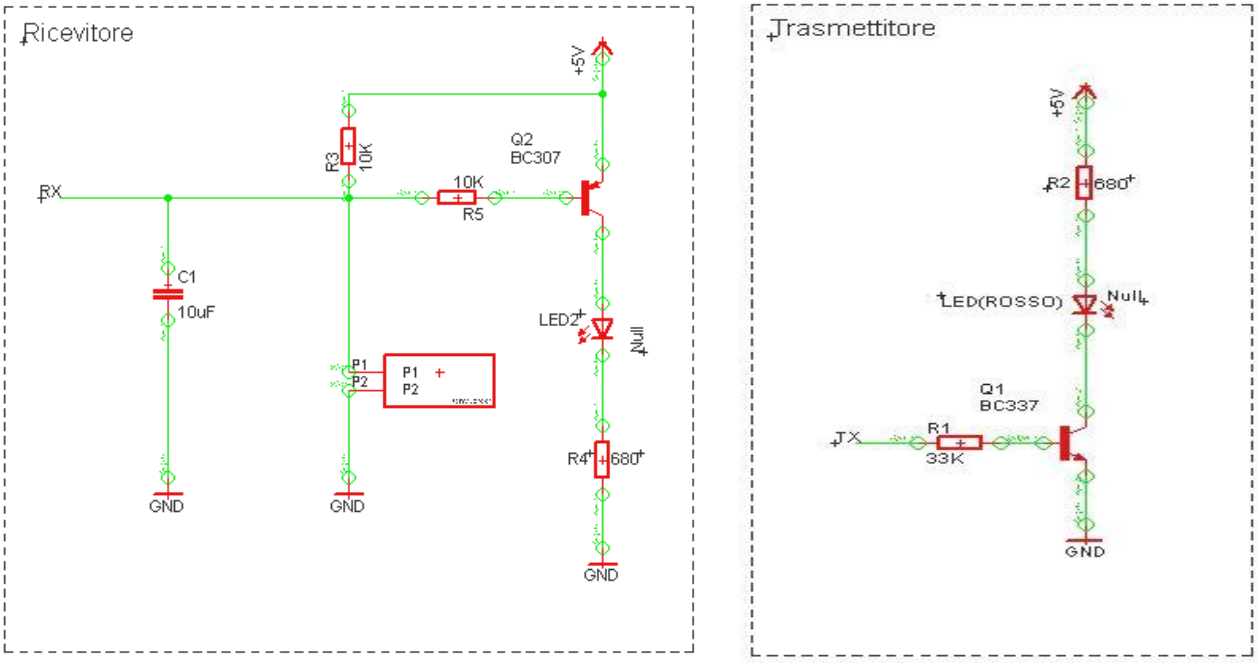
- la prima è che, mentre il TSM 40/110 è un tilt switch al mercurio, quello raffigurato (TSW 30/60) non lo è.
- La seconda differenza, ed è quella di maggior interesse perché riguarda la gara, è l'angolo di inclinazione che i due interruttori permettono per un funzionamento utile. Lo switch installato sul robot ha un angolo di inclinazione pari a 10 o 40 gradi, a seconda del verso in cui viene posizionato (orizzonatale o verticale). L'altro switch invece, ha il vincolo di essere posizionato orizzontalmente alla superficie, perché "percepisce" l'inclinazione solo in un verso; l'angolo di inclinazione che serve all'interruttore per cambiare di stato è di 16 gradi.



6.3.2 Schema elettrico

In sintesi, il funzionamento del ricevitore è il seguente:

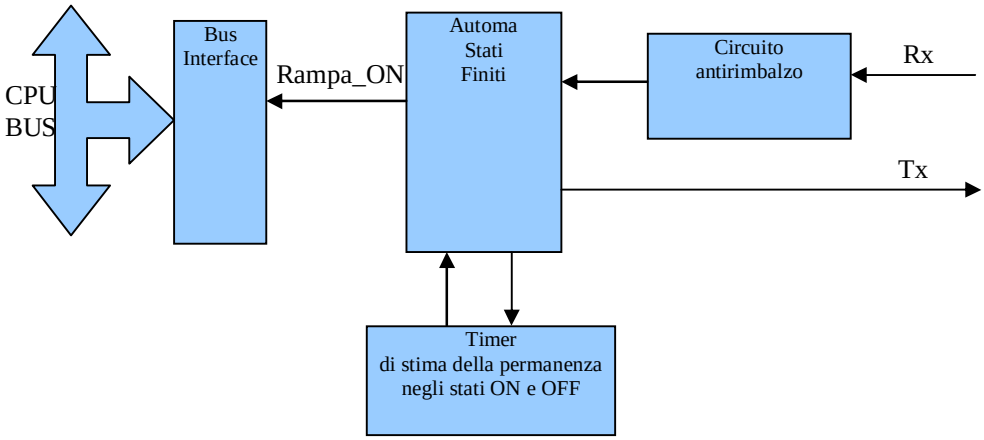
- attraverso la resistenza R3, i 5V di alimentazione, caricano il condensatore C1, che mantiene alto il segnale Rx per tutto il periodo di tempo in cui l'interruttore si mantiene aperto;
- con C1 caricato al massimo la giunzione base-emettitore di Q2 si polarizza inversamente evitando che giunga corrente al collettore e il diodo led "LED2" rimane spento e la FPGA leggerà un valore logico alto;
- quando invece il robot si trova sulla rampa, l'interruttore si chiude, il condensatore C1 si scarica verso massa, polarizzando direttamente la giunzione base-emettitore di Q2 e consentendo alla corrente di circolare nel circuito di collettore accendendo il led; in questo stato il segnale Rx verso la FPGA sarà a livello logico basso, indicando che Nessie è sulla rampa.



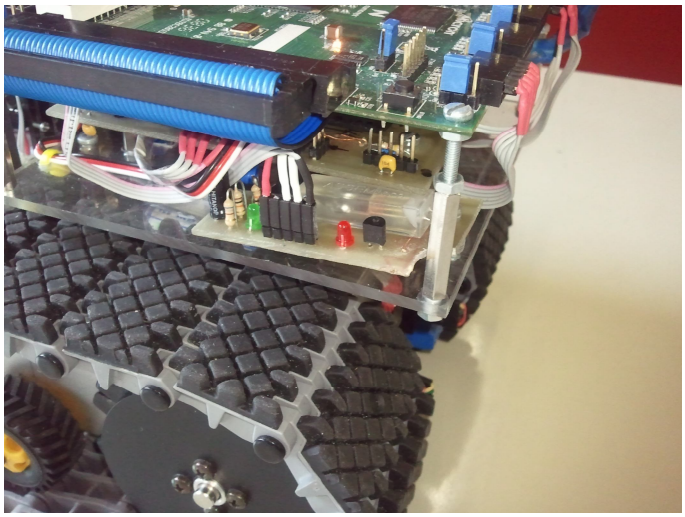
Il circuito trasmettitore permette di visualizzare il segnale di riconoscimento della rampa da parte della FPGA sul led rosso segnalando l'evento all'esterno, tramite il led rosso.

6.3.3 Considerazioni e problemi riscontrati

I due interruttori sono stati acquistati con lo stesso package in modo tale da poterli utilizzare sullo stesso circuito, e di intercambiarli in base alle esigenze del robot e della pista, la quale dovrà affrontare.
Il TSW 30/60 è molto sensibile alla pendenza, tenendo conto che ha un range di inclinazione piccolo (16°); il TSM 40/110 invece è meno sensibile e ciò sarebbe una buona cosa ma, per converso, presenta l'inconveniete che esso "sente" le variazioni d'inclinazione sia lungo il piano di beccheggio, sia lungo quello orizzontale; pertanto, durante il movimento del robot, le vibrazioni orizzontali producono rimbalzi sui contatti, che ingenerano instabilità sul segnale Rx.
Il problema è stato risolto via hardware sfruttando, appunto, la flessibilità della logica programmabile e filtrando il segnale Rx, diretto verso la CPU, con il circuito digitale indicato nel seguente schema a blocchi.

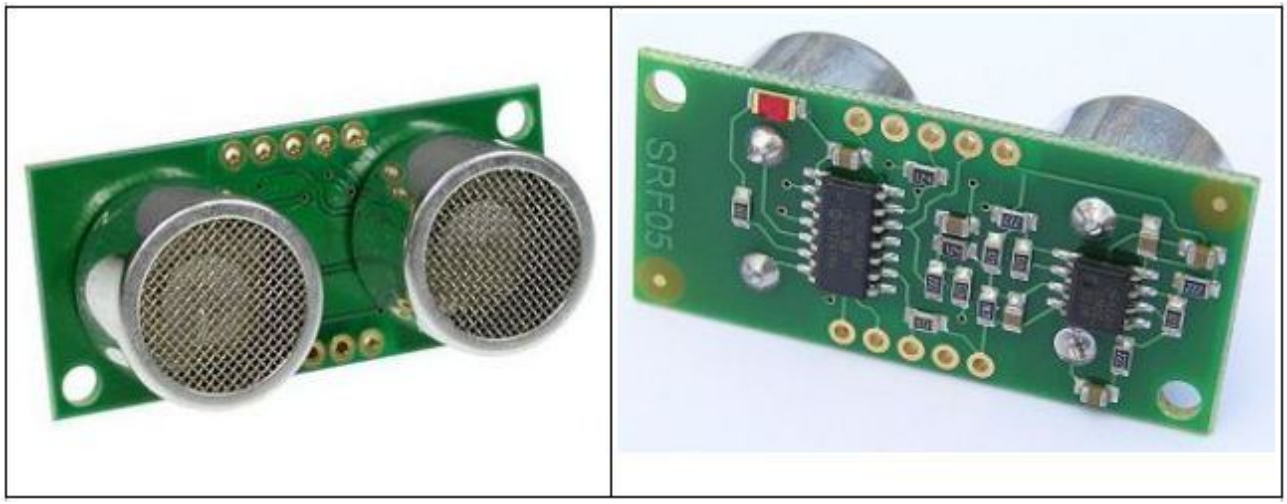


6.3.4 Scheda montata sul robot



6.4 Sensore ad Ultrasuoni

SRF05 – Ultra Sonic



L'SRF05 è l'acronimo di “Sonic Range Finder” versione 05 ed è il sensore ad ultrasuoni che è stato aggiunto al robot per affrontare diverse problematiche : evitare lo o gli ostacoli presenti sul percorso e, trovare e salvare la vittima; insomma due compiti opposti, ma che hanno bisogno entrambi di questo importante dispositivo.
Questo sensore è una sorta di sonar che sfrutta gli ultrasuoni per determinare la distanza da un oggetto che ha di fronte. E' composto da due elementi, un trasmettitore e un ricevitore, che potremmo definire gli “occhi” dell'automa, il cui compito sarà rispettivamente di trasmettere un pacchetto di ultrasuoni e di misurare il tempo di ritorno dell'eco rimandata dall'eventuale ostacolo.

6.4.1 Funzionamento

I passi fondamentali del funzionamento di questo sensore sono:

- il circuito d'interfaccia realizzato sulla FPGA produce un impulso elettrico della durata minima di 10 ms che viene convertito dal controllo presente sul sensore in un treno d'impulsi elettrici alla frequenza di 40 kHz;

- il trasduttore di trasmissione trasforma, per effetto piezolelettrico, il treno d'impulsi elettrici in un "burst" di onde sonore di pari frequenza;

- il sensore, al completamento del "burst", pone alto in uscita un segnale, che viene letto dalla

FPGA, e che rimarrà in tale stato fino a quando il traduttore di ricezione, che effettua l'operazione inversa del trasmettitore, riceve l'eco dall'ostacolo. Appena ciò accade il segnale verrà riportato al livello basso e sarà proprio la distanza temporale ΔT tra i due fronti di quest'impulso e che sarà misurata dal circuito d'interfaccia;

- ora utilizzando la formula V(velocità) = S(spazio) / ΔT (tempo) e conoscendo sia la velocità del suono nell'aria (340m/s circa), sia l'intervallo di tempo tra la trasmissione e la ricezione dell'ultra-suono (ΔT) , si può ottenere la distanza del robot da un ostacolo, in modo abbastanza preciso. Infatti bisogna tener conto che al ricevitore non arriva solamente il rimbalzo perpendicolare, ma arrivano una serie di rimbalzi che possono a volte, diminuire la precisione del dispositivo.

- l'interfaccia da noi realizzata, consente alla CPU di leggere direttamente la distanza dal circuito d'interfaccia, senza alcuna ulteriore necessità di rielaborazione.

-

6.4.2 Caratteristiche tecniche

Le caratteristiche tecniche dell'SRF05 sono riportate nella tabella seguente:

Caratteristiche Tecniche:	
Tensione Operativa	5V
Corrente Operativa Tipica	4mA
Frequenza	40 Khz
Portata	1cm - 4mt
Impulso di ritorno	Segnale TTL positivo, di durata proporzionale alla distanza rilevata.
Trigger di Input	Impulso TTL di durata minima di 10 uS.
Modalita' di funzionamento	Pin singolo per trig/echo o 2 Pin SRF04 compatibile.
Dimensioni	43 x 20 x H 17 mm

L'immagine sottostante raffigura il bottom del sensore e la configurazione dei pin che abbiamo deciso di adottare.

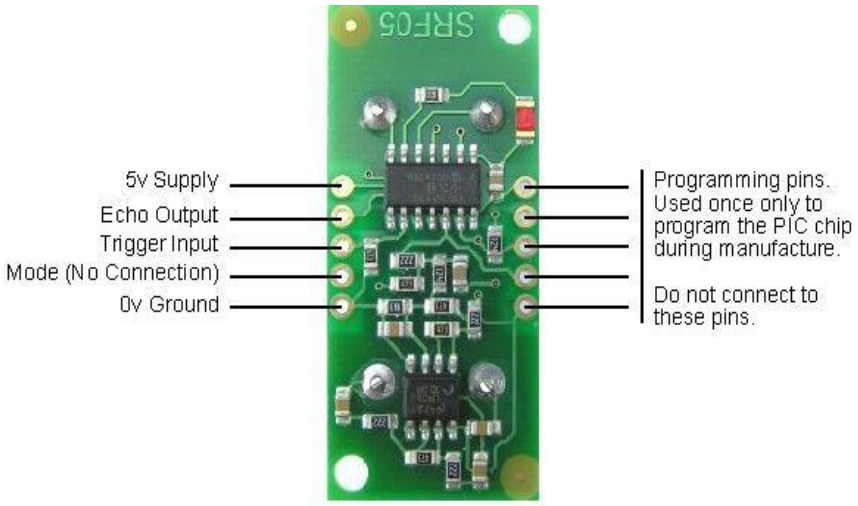
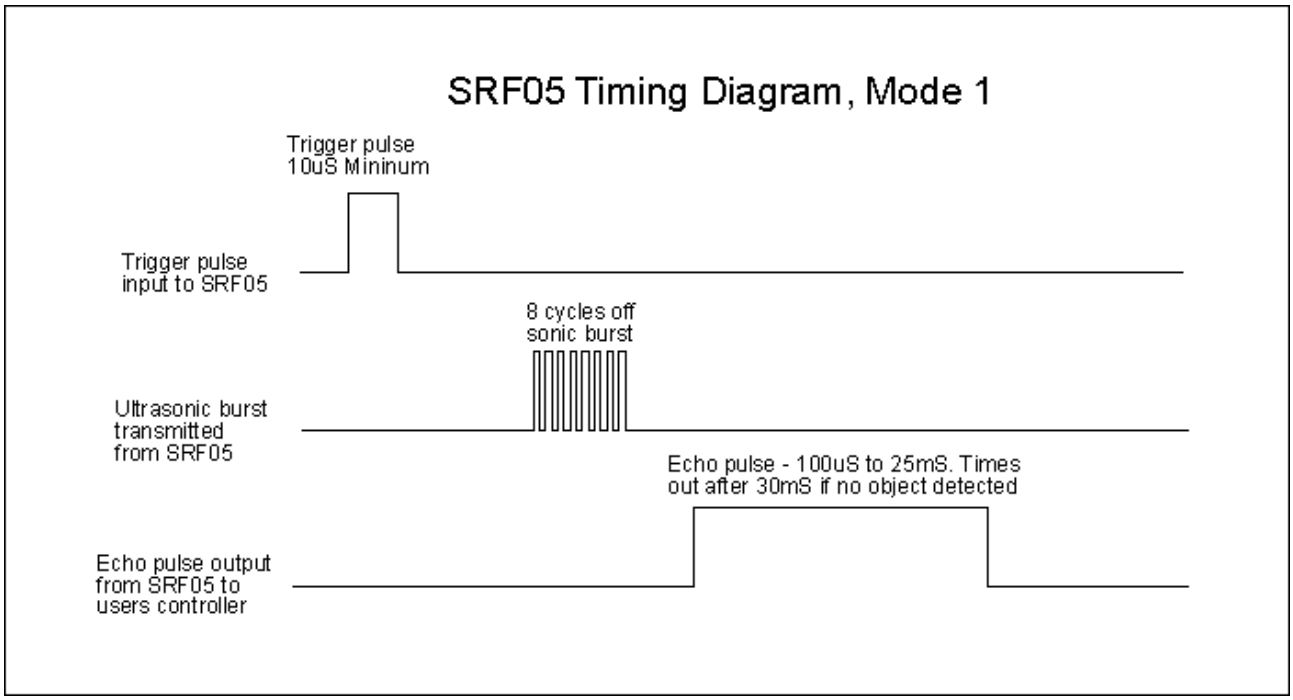
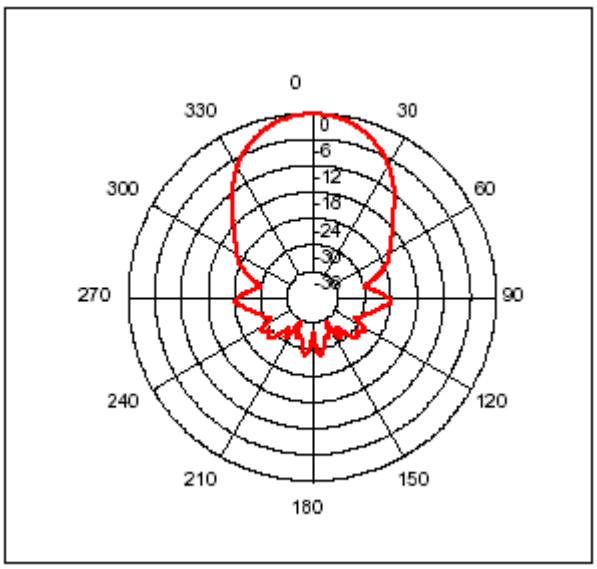


Figura 6.20

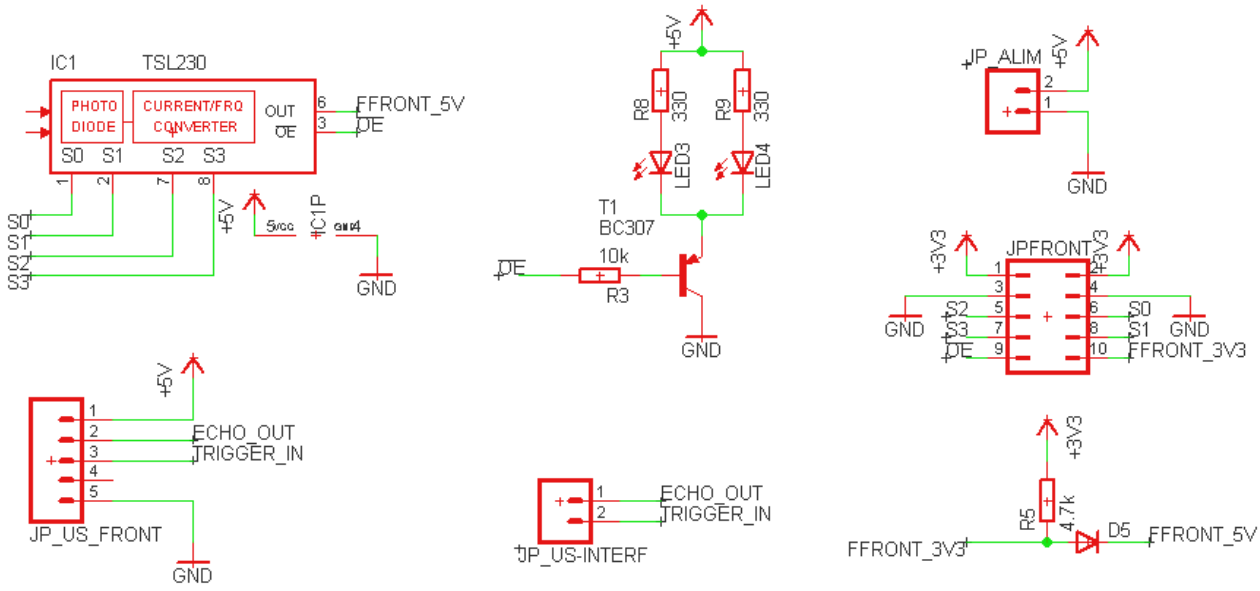
Per questa configurazione il datasheet fornisce un grafico che mostra le varie fasi del suo lavoro



L'immagine qui a destra invece rappresenta la "vista" del sensore.
Il suo range di localizzazione è molto preciso ed ha una buonissima portata per l'utilizzo che ne fà il robot. Il campo di rilevazione frontale di circa 60° che il grafico fornisce, risulta senz'altro adatto al rilevamento di un ostacolo frontale. Per la ricerca della vittima esso può risultare molto stretto. L'unica soluzione è quella di mettere in atto una strategia di "brandeggio" che sia in grado di avvicinarsi frontalmente alla vittima per approssimazioni successive.



6.4.3 Schema elettrico scheda luminosità frontale

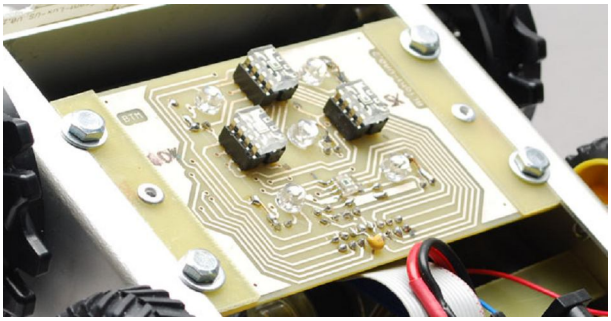


Questo schema elettrico oltre a racchiudere in sè il sensore ad ultrasuoni SRF05, possiede al suo interno un sensore di luminosità TSL230, lo stesso che viene utilizzato nella scheda bottom del robot, e verrà utilizzato per il riconoscimento della "vittima" (rappresentata da una lattina ricoperta di stagnola riflettente).

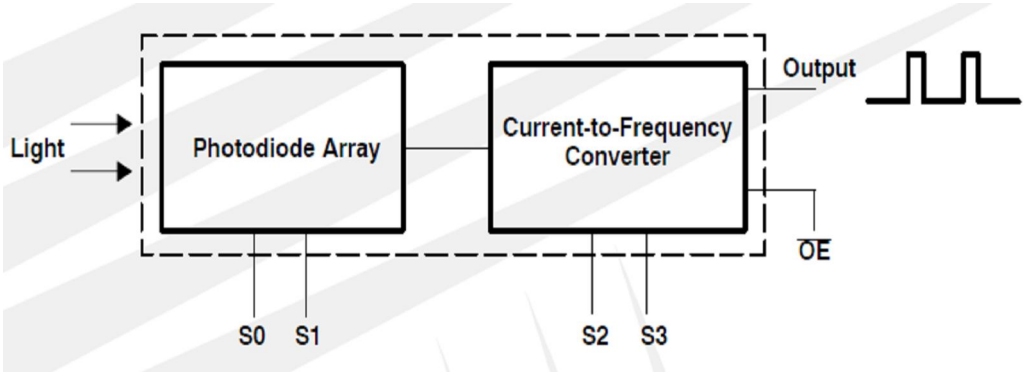
6.5 Sensore di Luminosità

6.5.1 Sensori di luminosità inferiori

Questo è l'organo sensoriale più importante per Nessie, perchè gli consente di seguire la traccia che la condurrà verso la vittima da salvare.
Il sensore attuale è l'evoluzione dei quello utilizzato nei modelli precedenti ed è basato su quattro sensori della Taos, tre dei quali sono sensibili all'intensità luminosa monocromatica, mentre il quarto è realizzato con fotodiodi che sono capaci di scomporre la luce nelle sue componenti primarie rosso, verde e blu.
La disposizione geometrica dei sensori è stata dettata dall'esperienza e risulta abbastanza efficace nella guida del robot.
Tutti i sensori scelti danno in uscita un segnale binario la cui frequenza è proporzionale all'intensità della radiazione luminosa incidente o a una sua componente cromatica.



Lo schema a blocchi dei due sensori, quello di luminosità con la sigla TSL230-LF e quello di colore con la sigla TCS3200, si presentano con uno schema a blocchi simile.



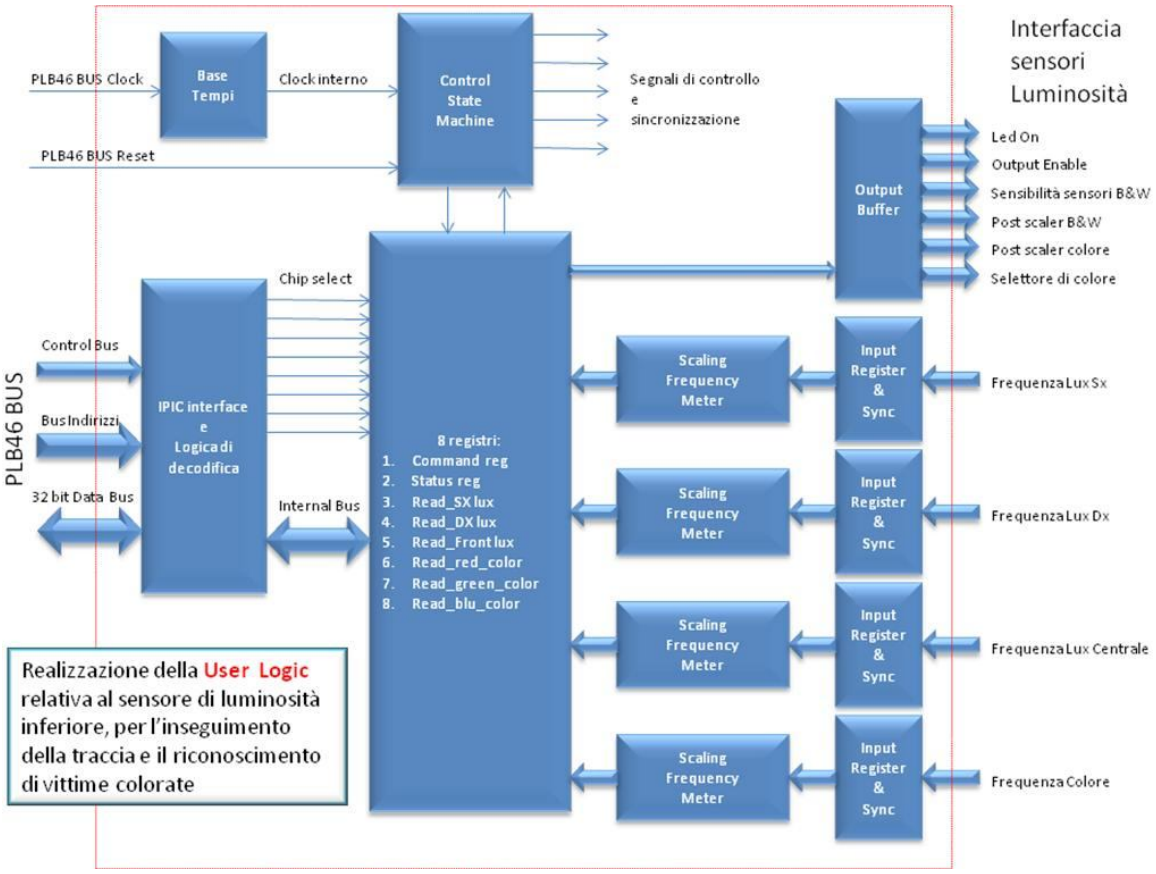
La differenza tra i due consiste nella diversa funzione svolta dai segnali di controllo dell'array di fotodiodi:
- nel sensore di luminosità servono a impostare la sensibilità dei fotodiodi;
- in quello di colore consentono la scelta della specifica componente cromatica che si vuole misurare.
Entrambi presentano in uscita un convertitore tensione-frequenza dotato di un “post-scaler” per selezionare uno di quattro range di frequenza disponibili.

6.5.2 Circuito d'interfaccia tra sensori e CPU

Anche in questo caso la flessibilità della FPGA ci ha consentito di realizzare un interfacciamento tra sensori e CPU flessibile ed efficiente.
Viste che le misure di luminosità devono essere simultanee abbiamo realizzato quattro circuiti paralleli che misurano in modo sincrono i risultati della conversione dei sensori. In più, in funzione delle condizioni di luminosità e della velocità di conversione richiesta, le misure possono effettuate scegliendo una base tempi programmabile da 1 millisecondo a un secondo.

6.5.3 Un esempio di modulo d'interfaccia progettato

Possiamo approfittare del fatto il sensore di luminosità è il sensore più articolato di cui Nessie dispone per dare un esempio di come sia possibile realizzare dei moduli personalizzati sfruttando la riconfigurabilità delle FPGA e loro capacità di plasmarsi ai bisogni specifici dell'utente.



Come si vede dallo schema ognuno dei sensori è connesso a un frequenzimetro separato, di tipo scalabile, per adattarlo alle differenti base tempi richieste.
La presenza di una macchina a stati finiti di controllo, permette di impostare i valori di configurazione dei sensori e di avere differenti modi di misura. Ad esempio, si può scegliere, per ognuna delle due tipologie di sensori, se effettuare misure in modo continuo o in modo spot; si decidere se si desidera scandire in sequenza le singole componenti cromatiche oppure di misurarne una in particolare.
La CPU non deve far altro che impostare la modalità di misura, avviare la misura stessa, attendere che sia stata completata e leggere quella desiderata da uno dei sei registri di lettura.

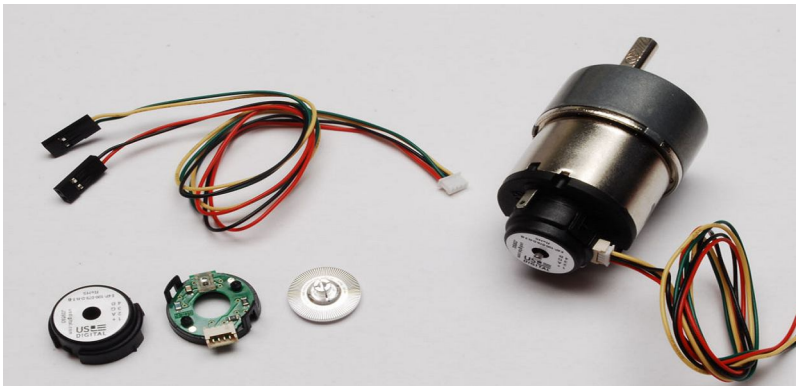
CAP.7 GLI ATTUATORI

Nessie utilizza quattro attuatori con le seguenti funzioni:

- Due motori in corrente continua per l'avanzamento;
- Un motore in corrente continua per il sollevanento/abbassamento della pinza;
- Un servo per apertura/chiusura della pinza.

7.1 Motori d'avanzamento

La scelta dei motori è stata influenzata dalle vicende precedenti e dalla scarse prestazioni che erano state riscontrate nel precedente modello che utilizzava una coppia di motoriduttori della Tamiya, che si sono rivelati non adeguati alle dimensioni e al peso del nostro robot. Per Nessie abbiamo pensato a qualcosa di più solido e la soluzione ci è stata fornita ancora dalla Linx Motion con la seguente coppia di motori.
I motori possono essere dotati di un encoder incrementale predisposto per un montaggio sul loro asse, che fornisce due segnali in quadratura da 100 impulsi/giro, da cui si possono dedurre: la direzione, la distanza percorsa e la velocità.

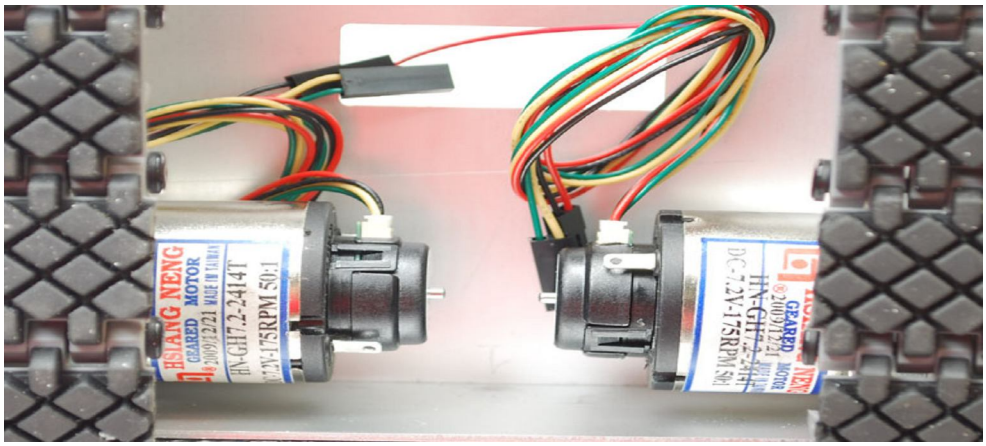


I dati di targa dei due motori sono i seguenti:

Tensione di alimentazione massima	7.2 Volt
Velocità di rotazione a vuoto	175 giri/minuto
Rapporto di trasmissione	50 : 1
Coppia massima di stallo	7,1 Kg * cm
Peso	Circa 122g

Particolare del fissaggio degli encoder ai motori.

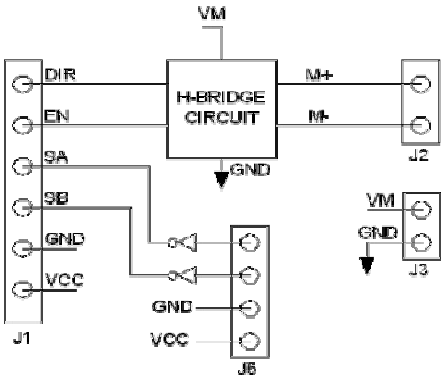
I motori montati su Nessie.



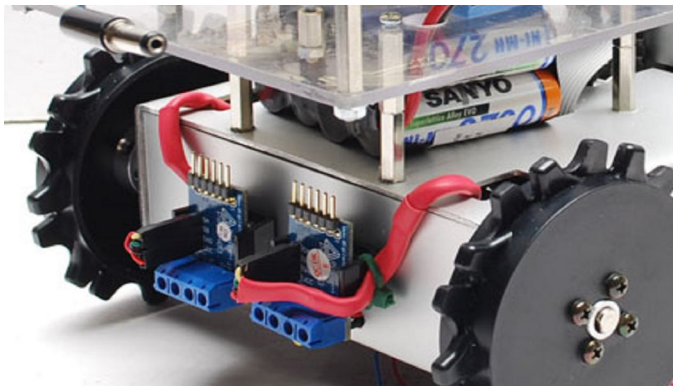
7.2 Ponte di transistors per il pilotaggio dei motori in c.c.

Per comandare i motori in corrente continua, Nessie usa dei moduli della serie Pmod della Digilent Inc. In particolare la nostra scelta è caduta sul ponte di transistor PmodHB3 2A , H-Bridge Module, perchè sono risultati facilmente collegabili alla nostra scheda madre e avevano le caratteristiche di pilotaggio adeguate ai nostri motori.
Questi driver funzionano con tensioni di alimentazione da 2,5 V a 5V, ma, sono normalmente gestiti con segnali d'ingresso a 3.3V, la tensione di alimentazione che usa la scheda Nexys2.
Possono pilotare alimentazioni di motori fino a 12V con correnti massime di 2 A, senz'altro sufficienti per il nostro caso con tensioni massime di 7,2 V e 6V.

Lo schema di principio dei dispositivi è il seguente, con accanto il circuito.

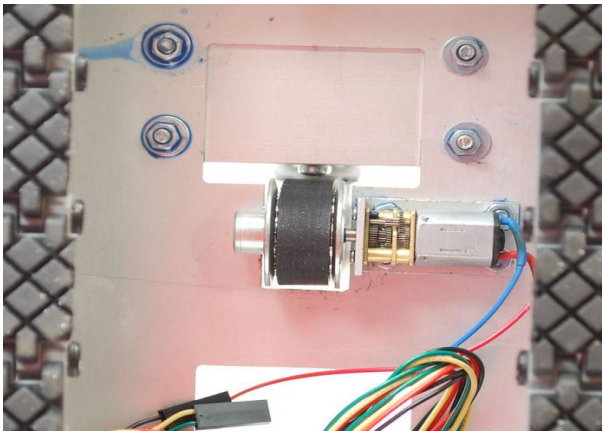


I segnali DIR ed EN, rappresentano i segnali di comando provenienti dalla FPGA e rappresentano, rispettivamente, il segnale modulato in PWM, generato dal nostro circuito d'interfaccia, il controllo della direzione in avanti o indietro.
I moduli dispongono anche gli adattatori di livello per ricevere i segnali provenienti dagli encoder incrementali montati sul motore.



7.3 Motore sollevamento pinza

Il sollevamento/abbassmento della pinza è stato ottunuto con un minuscolo, ma prestante, motore in c.c., dotato di un rapporto di riduzione di 1 a 298, che con pochi mA di corrente e 6 volt massimi di tensione di alimentazione, consente un agevole sollevamento della vittima da salvare. Il motore occupa uno spazio di solo 26x10x12 millimetri e pesa in tutto circa 10g.
Anche questo motore è stato comandato con lo stesso ponte PmodHB3.



7.4 Servo motori di apertura chiusura

Il servo presente sulla pinza POB, si pilota direttamente con il segnale logico PWM così come è prodotto dalla FPGA e non richiedono nessun circuito intermedio di pilotaggio.
I dati del servo usato (Futaba S3003) sono i seguenti:

- Segnale di controllo: PWM di ampiezza 1520 microsecondi per la posizione neutra;
- Caratteristica dell'impulso di comando: onda quadra da 3 a 5 V picco-picco;
- Tensione di alimentazione operativa: da 4.8 a 6 Volt;
- Coppia di stallo: 3,2 Kg*cm (@ 4.8V), 4,1 Kg*cm (@ 6V);
- Velocità di rotazione: 0,23 sec/grado (@ 4.8V), 0,19 sec/grado (@ 6V);
- Direzione di rotazione: in verso antiorario da 1520 a 1900 microsecondi;
- Corrente assorbita in stallo: 7,2 mA (@ 4.8V), 8 mA (@ 6V).

CAP.8 L’AMBIENTE DI SVILUPPO

8.1 Xilinx Platform Studio(XPS)

L'ambiente di sviluppo nel quale è stato sviluppato il progetto ci è stato offerto dalla Xilinx; esso va sotto il nome di Xilinx Platform Studio (XPS) ed è lo strumento che consente ai progettisti di sistemi embedded di creare interi sistemi hardware/software personalizzati e d'integrarli all'interno di logiche FPGA della Xilix.
XPS è composto da vari moduli che consentono di progettare utilizzando un ambiente grafico molto intuitivo e sofisticato, aiutando l'utente con “wizard” e procedure guidate per accompagnarlo durante tutti i passi necessari a creare, in poco tempo, sistemi di elaborazione dati interamente definiti dall'utente in tutte le sue parti: dalla scelta della CPU e dei moduli standard disponibili a catalogo, alla realizzazione delle periferiche particolari, definite sulle specifiche tipiche di ogni applicazione particolare, per giungere fino allo sviluppo dell'applicazione software e della successiva integrazione delle parti.
Disponibile come parte di un sistema integrato che include, oltre a tutti gli strumenti di sintesi e di implementazione del progetto, i programmi di simulazione, sia hardware che software, tutto ciò che è necessario per il “debug” integrato dell'intero progetto.
I moduli fondamentali di XPS:

- il BSB (Base System Builder) che, grazie alla libreria offerta dal costruttore della scheda, nel nostro caso la società americana Digilent Inc, include la procedura che consente con un semplice processo di “pick and place” da un menu di generare il microcomputer di base con tutte le periferiche messe a disposizione sulla scheda stessa (memorie e dispositivi di I/O);
- un ampio catalogo di dispositivi standard interfacciabili sul BUS del microprocessore messi a disposizione dalla Xilinx;
- la procedura guidata per la produzione di “template” relativi alle periferiche “custom” pensate e definite dall'utente in un linguaggio descrittivo dell'hardware ad alto livello di astrazione, come il VHDL o il Verilog;
- l'integrazione con l'ambiente di sintesi, “mapping”, “placing” e “routing” ISE di Xilinx;
- l'esportazione del modello hardware generato verso ambienti IDE più standard, come SDK (Software Development Kit), offerta da Xilinx stessa, che impiega la nota e diffusa interfaccia Eclipse per lo sviluppo del software;
- la creazione automatica di documentazione.

8.2 Piattaforma ISE (Integrated System Environment) Design Suite

ISE (Integrated System Environment), di cui Xilinx mette a disposizione una versione WEB gratuita, è la suite dei moduli necessari a trasformare le descrizioni ad alto livello dell'hardware in una stringa di bit necessari a configurare la FPGA in modo da implementare l'intero progetto, hardware e software, così come è stato sviluppato dall'utente. La stringa di bit, una volta prodotta viene scritta su un memoria flash che la FPGA, all'atto dell'accensione, leggerà per configurarsi secondo le specifiche di progetto.
Tra le varie versione disponibili noi abbiamo impiegato quella cosiddetta “Embedded”, che include infatti XPS e SDK, insieme al Core Generator e al programma di debug hardware ChipScope.
Il Core Generator, serve a generare i componenti standard configurabili dall'utente, mentre il ChipScope serve a produrre della logica da inserire all'interno della FPGA da usare come un analizzatore di stati logici da impiegare per “tracciare” l'evoluzione dei segnali presenti all'interno di nodi significativi del circuito, adatti a diagnosticare problemi e risolvere situazioni di funzionamento non corretti.
La versione da noi utilizzata per il progetto è la non recentissima 10.3, offerta gratuitamente, ma ancora più che valida per lo sviluppo dei progetti a livello didattico, anche di elevata complessità.

8.3 Il linguaggio di programmazione: C/C++

Il linguaggio di programmazione da noi utilizzato è il C/C++.
Le motivazioni de questa scelta sono molteplici e includono considerazioni di vario genere.
Innanzitutto è il linguaggio di programmazione che viene insegnato fin dal terzo anno nella nostra scuola; poi è quello che viene offerto come linguaggio di base nella piattaforma che abbiamo impiegato per progettare Nessie.
Ma ci sono anche altre ragioni alla base della scelta fatta.
Il C è un linguaggio di alto livello con un insieme ristretto di costrutti di controllo e parole chiave, ma possiede una struttura che lo rende adatto nelle applicazioni “real time”, dove il software è molto vicino all'hardware; ovvero, interagisce molto strettamente con esso per il controllo di eventi fisici, provenienti da sensori e trasduttori, e deve produrre risposte che controllano attuatori, come i motori di Nessie, a cui bisogna dare indicazioni in tempi molto ristretti.
Il C, infatti, e' diventato un linguaggio professionale ampiamente utilizzato perché è dotato, appunto, di strutture di alto livello ma puo' maneggiare attività di basso livello con l'efficienza in un linguaggio Assemler. Il linguaggio produce un codice oggetto molto efficiente e puo' essere compilato su un'ampia gamma di sistemi operativi.
La struttura logica del C consente una progettazione procedurale ben strutturata e ogni programmatore puo' aggiungere nuove librerie rendendo la programmazione estremamente versatile e flessibile.
Per i prossimi anni, con un hardware più stabilizzato, si può pensare di estendere il linguaggio alla programmazione ad oggetti, magari concentrandoci di più sul software e passando al C++.
Il compilatore da noi usato, incluso in XPS, permette entrambe le versioni del linguaggio.

8.3.1 Il compilatore

Il compilatore usato è la versione attuale del GCC (GNU Compiler Collection, in origine GNU C Compiler). GCC è un compilatore C e C++ multi-target, creato inizialmente dal fondatore della Free Software Foundation, Richard Matthew Stallman, come parte del progetto GNU.
Esso è successivamente diventato il compilatore per eccellenza utilizzato nel comunità Open Software, che vede in Linux il mattone fondamentale su cui basare i sistemi di informatica aperta allo scambio del codice e alla condivisione dei sorgenti.
Si sta pensando ad una nuova edizione di Nessie (Nessie II), in cui inserire un Kernel Linux, ritagliato per i sistemi embedded, come supporto alle prossime applicazione. Ma questo è un discorso da riprendere il prossimo anno.

8.4 La scheda madre e la FPGA utilizzata

La famiglia Spartan3E delle FPGA Xilinx, utilizzata nella nostra scheda Nexys2 della Digilent, offre la versatilità e la ricchezza di una vasta gamma di componenti, che includono dalle cinquanta mila ai cinque milioni di porte logiche, un costo basso e facilmente accessibile ai sempre più scarni budget a disposizione della scuola.
Noi stiamo lavorando con il componente Spartan 3E – 1200-5 da 384 pin e tutto l'intero progetto di Nessie occupa in tutto circa 5288 slices su un totale di 8672 disponibili. Ovvero, tutto il progetto da noi ritagliato, prende circa il 60% delle risorse di logica programmabile disponibili e impiega solo il 60% delle porte presenti sul componente.
Riguardo alle ulteriori risorse impiegate da Nessie sono due includere:

- i 32 Kbyte di Ram interna utilizzata; circa il 57% di quella disponibile.
- I 128 pin di Input/Output utilizzati di un totale di 250 presenti nel package usato, quindi circa il 51%.
- I 16 moltiplicatori hardware sui 28 presenti, utilizzati per realizzare le periferiche da noi progettate; ovvero il 57% del totale a disposizione.

8.5 Il software

Per le gestione e organizzazione del software abbiamo utilizzato il classico modo di operare nel linguaggio C. Ci siamo organizzati dividendo il programma in varie routine le quali verranno chiamate nel main. Il main(o programma principale) deve essere ordinato e comprensibile per una buona riuscita del lavoro.
Il main è strutturato in diverse fasi:

- **Fase di POST** (Power On Self Test) in cui si effettua un test delle risorse fondamentali (Dip Switch, LED, Display, Timer, UART, Memoria) per avere una indicazione sulla corretta configurazione della FPGA tramite la bitstream contenuta nella memoria flash e di una altrettanto buona partenza del main;
- **Fase di inizializzazione**: nella fase di inizializzazione(oppure fase di INIT) vengono inserite tutte le inizializzazione dell'hardware: inizializzazione delle porte di IO standard presenti sulla Nexys2; inizializzazione del controllo dei motori; assegna i valori di default al controllo di luminosità;
- **Fase di taratura**: determina le soglie di riferimento dei livelli di riflessione dei sensori di luminosità inferiori e frontali. In questa fase si determina anche la sensibilità, il divisore di frequenza del post-scaler e la base tempi di misura da impostare sui sensori;

- **Loop principale**: nel loop principale vengono chiamate ed eseguite le routine inserite in un ciclo infinito. Nel nostro caso,abbiamo preferito suddividere il loop principale in un "Grande" CASE(istruzione del linguaggio C) nel quale vengono rappresentate le situazioni in cui si potrà trovare il robot durante lo svolgimento del tragitto.

Il "Grande Case" è strutturato in quattro stati differenti e rappresenta un automa a stati finiti, le cui transizioni di stato indicano i diversi contesti operativi di Nessie:

Percorso con inseguimento della traccia: caso in cui il robot sta seguendo la linea nera tracciata sulla pista. Dalla partenza, il robot dovrà superare varie stanze seguendo solamente una linea nera tracciata sulla superficie utilizzando i sensori di luminosità posti nella parte inferiore del robot.

Ostacolo: durante il percorso, è necessaria una routine che permetta il superamento di un eventuale ostacolo presente sul tracciato. Siccome l'ostacolo, in base al regolamento, si potrà trovare solamente nella "Seconda stanza" mentre la vittima sarà sicuramente nella "Terza", in questa fase non sarà necessario tentare di identificare l'ostacolo per discernerlo dalla vittima.

Rampa: all'uscita della Seconda stanza,il robot, per accedere alla successiva, dovrà necessariamente superare una rampa con inclinazione nota. Nel nostro caso, per identificare la presenza della rampa, utilizziamo un circuito di "tilting" in grado di rilevare l'inclinazione del piano di avanzamento del robot. Alla fine della rampa, sulla superficie sarà posta una strisca trasversale di materiale riflettente che identificherà l'inizio della Terza e ultima stanza e, quindi, il passaggio allo stato finale.

Cerca vittima: fase di ricerca della vittima nell'ultima stanza. Il robot dovrà localizzare la "vittima" all'interno del piano senza nessun tracciato da seguire; dovrà, attraverso la pinza, sollevare la vittima e portarla "al sicuro" su una specifica piattaforma identificata da un triangolo colorato di nero rialzato dalla superficie.

Alla fine di ogni blocco di istruzioni del Case , cioè al termine di ognuno dei singoli segmenti di codice che gestiscono i vari stati, viene utilizzata una "Condizione di transizione" la quale segna il passaggio da

uno stato all'altro del robot e determina i parametri di comportamento da tenere in conto all'interno di ogni specifco segmento di programma.
Nel nostro caso,le condizione di transizione tra uno stato e l'altro sono le seguenti:

- da **"Stato percorso" a " Stato ostacolo"**: la condizione di transizione è stabilita utilizzando i sensori a ultrasuoni posti nella parte frontale del robot. Infatti, tale condizione sarà vera solo se il robot troverà un ostacolo entro una determinata misura minima in centimetri;
- da **"Stato ostacolo" a " Stato percorso"**: il superamento dell'ostacolo riporta sempre il sistema nello stato precedente;
- da **"Stato ostacolo" a " Stato rampa"**: come detto precedentemente, utilizziamo un sensore di "tilting" per rilevare le inclinazioni del robot. Quindi, una volta iniziata la rampa e quindi rilevata la diversa inclinazione del robot, siamo in grado di stabilire che la Seconda stanza è terminata e quindi ogni ostacolo che troverà successivamente sarà da analizzare, in relazione alla sua riflettenza, in quanto non costituira più un elemento da superare bensì un possibile oggetto che rappresenterà la vittima;
- da **"Stato rampa" a "Stato cerca vittima"**: la condizione di fine rampa sarà data dal fatto che, il robot, dopo un certo periodo di permanenza su una detrminata pendenza, tornerà in posizione orizzontale per predisporsi all'accesso dell'ultima stanza.

Oltre al main, il programma è composto da due file "header" chiamati rispettivamente TypeDef.h e DefAlloc.h.

8.5.1 Header TypeDef.h

Questo file definisce tutte le strutture dati che rappresentano i registri d'interfaccia tra la CPU e ognuno dei circuiti di controllo e condizionamento dei singoli sensori.
Ogni registro, infatti, è diviso in campi e svolge funzioni che possono essere di lettura e/o scrittura.
In generale, ogni periferica viene vista dal software come un banco di registri, in cui le funzioni di base, svolte tipicamente da ognuno di essi, sono:

7. COMMAND;
8. STATUS;
9. DATA.

Solitamente viene scelto il registro "0", quello a indirizzo più basso, come registro di comando, verso il quale inviare tutti i singoli segnali che attivano/disattivano ogni specifica funzione del sensore.
Il successivo registro, registro "1" è tipicamente impiegato per leggere lo stato del controllo del sensore per verificare il corretto funzionamento dell'hardware e per stabilire lo stato effettivo in cui esso si trova.
Tutti i registri successivi sono utilizzati come registri di scambio dei dati, in lettura e/o in scrittura, in funzione della periferica. Di solito, si legge da sensori e si scrive sugli attuatori
In questo file, ogni registro che riguarda il sensore viene descritto come un "type" definito dall'utente i cui campi strutturali possono essere raggiunti tramite un puntatore ancora da allocare.

Si riporta qui di seguito un esempio per tutti.
Visto che in precedenza era stato mostrato lo schema a blocchi dell'hardware di controllo dei sensori di luminosità inferiori, si illustrano le parti dell'"header" file che li rigurdano.

Si inizia dal "COMMAND REGISTER" ovvero dal registro "0".

In esso ogni bit è tipicamente di scrittura ma può, per verifica, essere anche rileggibile.

```
typedef struct LuxReg0_st
{
 Xuint32 LED_ON      :1;
 Xuint32 OE_n:1;
 Xuint32 S1_S0_BW          :2;
 Xuint32 S3_S2_BW          :2;
 Xuint32 S0_S1_COL  :2;
 Xuint32 S2_S3_COL :2;
 Xuint32 Continous_BW        :1;
 Xuint32 Continous_COLOR    :1;
 Xuint32 scale_select_for_measure_BW      :2;
 Xuint32 scale_select_for_measure_COLOR :2;
 Xuint32 Lux_NonConnessi           :11;                  //NON CONNESSI
 Xuint32 Reg0_SetTimer             :1;                   //WRITE ONLY
 Xuint32 start_single_color        :1;                   //WRITE ONLY
 Xuint32 start_all_color           :1;
 Xuint32 start_BW_only             :1;
  Xuint32 start_all          :1;
}LUX_REG_0, *LUX_REG_0_PTR;

//(Significato di ognuno dei bit del registro "0":

//bit0: LedOn: WriteOnly=> accende i Led

 //bit1:OE:      WriteOnly=> abilita frequenza uscita sensori

 //bit2-3:      S1-S0,Sensitive_Selector_BW,WriteOnly=> selezione sensibilità fotodiodi BW
 //"00"=spenti; da 01 a 11 tre livelli crescenti di sensibilità

//bit4-5:S3-S2 Frequency_Divider_BW,WriteOnly=>divisore frequenza sensori BW
 //"00":nessuna divisione,"01":divide per 2,"10":per dieci,"11"div per cento

 //bit 6-7:S0-S1 Frequency_Divider_COL,WriteOnly=> Divisore Frequenza Sensori colore
 //"00":nessuna divisione,"01":divide per 2,"10":per dieci,"11"div per cento

//bit8-9:S2-S3 COL_Selector:WriteOnly:Forza selettore singolo colore
 //"00":Red;"01":Blu;"10":bianco e nero;"11":Green

//bit10:Continous_BW,WriteOnly=>Forza scansione continua sensori BW

//bit11:Continous_COL,WriteOnly=>Forza Scansione continua sensore Colore

//bit12-13:Scale_select_BW,WriteOnly=>Selettori range misura frequenza sensori BW:
 //"00":base tempi 1 sec;"01"=100ms;"10":10ms;"11":1ms;

//bit14-15: Scale_Select_BW, WriteOnly=> selettori range misura frequenza sensori BW:
 //"00":base tempi 1 sec;"01"=100ms;"10":10ms;"11":1ms;
```

//bit27: Reg0_SetTimer, WriteOnly=> Fa partire il timer da 100us di base tempi

//bit28: Start_Single_Color, WriteOnly=> attiva misura per il solo colore selezionato dai bit 6-7

//bit29: Start_All_Color, WriteOnly=> attiva misura per tutti i colori

//bit30: Start_BW_Only,WriteOnly=> attiva misura solo sensori BW

//bit31: Start_All,WriteOnly=> scansione continua di tutti i sensori

8.5.2 Header DefAlloc.h

Questo file header è utilizzato per allocare in memoria le strutture dati definite in senso generale nel file TypeDef.h e, attraverso l'inizializzazione di puntatori alle strutture, astrarre l'hardware.
Con il termine astrazione dell'hardware si intende individuare una metodologia di riferimento all'hardware specifico, da parte del software, che risulti quanto più indipendente è possibile da due fattori rilevanti nel caso di impiego di FPGA:
a) modalità d'implementazione: nelle FPGA lo stesso sensore può essere interfacciato usato più hardware di tipo diverso, in funzione delle informazioni da estrarre dal sensore o delle prestazione attese.
b) allocazione in memoria: nelle FPGA, ogni qualvolta si effettua una modifica dell'hardware, tutti il sistema viene riallocato fisicamente in memoria, in quanto tutte le periferiche sono del tipo "memory mapped"; pertanto, tutti gli indirizzi possono essere stati modificati ad ogni "run" del processo d'implementazione necessario a generare la bitstream di configurazione.
Questa tecnica consente di scrivere le routine di utilizzo dei sensori, prima ancora che il progetto hardware sia completato oppure modificato, purchè ogni campo dei registri mantenga lo stesso significato, senza che ciò richieda necessariamente una modifica del software.

Esempio
//Iinizializza il pointer al registro Reg0
#define Lux_Reg_0 XPAR_INSEGUITORE_TRACCIA_BASEADDR

Il parametro "XPAR_INSEGUITORE_TRACCIA_BASEADDR" che si vede nella precdente dichiarazione è un parametro formale che rappresenta in modo simbolico l'indirizzo fisico a cui è stato allocato il banco di registri d'interfaccia riguardanti quel determinato sensore.

LUX_REG_0_PTR ControlloLux = Lux_Reg_0;
Con questa seconda assegnazione è stato inizializzato un pointer a questi registri ai quali si può, ora, accedere per leggere e scrivere i segnali d'intercomunicazione con la CPU.
Una volta inizializzato il pointer alla struttura effettiva, ora non più astratta ma fisicamente presente in memoria, si può accedere ad ogni singolo elemento.
Per facilitare ulteriormente questo accesso, si predispongono un certo numero di "define" che hanno ora il compito di rendere simbolica la lettura/scrittura di ogni campo della struttura, evitando la necessità di dover maneggiare direttamente il puntatore alla struttura stessa.

//Definizione macro dei comandi
#define LedOn ControlloLux->LED_ON = 1

```
#define LedOff ControlloLux->LED_ON = 0
#define AbilitaSensori ControlloLux->OE_n = 1
#define DisabilitaSensori ControlloLux->OE_n = 0
#define Sensibilita_BW ControlloLux->S1_S0_BW
#define DivisoreFrequenza_BW ControlloLux->S3_S2_BW
#define DivisoreFrequenza_COL ControlloLux->S0_S1_COL
#define SelezioneColore ControlloLux->S2_S3_COL
#define MisuraContinua_BW ControlloLux->Continous_BW
#define MisuraContinua_COL ControlloLux->Continous_COLOR
#define SelezioneScalaBW ControlloLux->scale_select_for_measure_BW
#define SelezioneScalaCOL ControlloLux->scale_select_for_measure_COLOR
#define StartTimer ControlloLux->Reg0_SetTimer
#define LetturaSingoloColore ControlloLux->start_single_color
#define LetturaMultiplaColore ControlloLux->start_all_color
#define LetturaBW ControlloLux->start_bw_only
#define LetturaCompleta ControlloLux->start_all
```

8.5.3 Esempio di utilizzo delle strutture dati

Di seguito viene presentato un esempio sull'utilizzo delle strutture dati definite prendendo ad esempio la routine TaraLux che permette, all'avvio di Nessie, di decidere le soglie di accetazione per le varie intensità di luminosità assolute all'interno dell'arena. L'ordine di acquisizione dei valori è il seguente:

- Materiale riflettente;
- Bianco;
- Grigio;
- Nero.

```
/*****************************************************************************
*       ROUTINE: Tara_Lux
* Imposta le variabili globali del sensore di luminosità in fase di test
*****************************************************************************/

void TaraLux (void)
{
AccensioneDisplay = 1;
ResetSevenSeg();
DatiVisualizzati = INITIAL_MESSAGE;
do
{
        Pause(100);                     //DEBOUNCING DELAY!
} while (StatoPulsante_1 == 0);

do
{
        Pause(100);                     //DEBOUNCING DELAY!
} while (StatoPulsante_1 != 0);

DatiVisualizzati = REFLECT_MESSAGE;
```

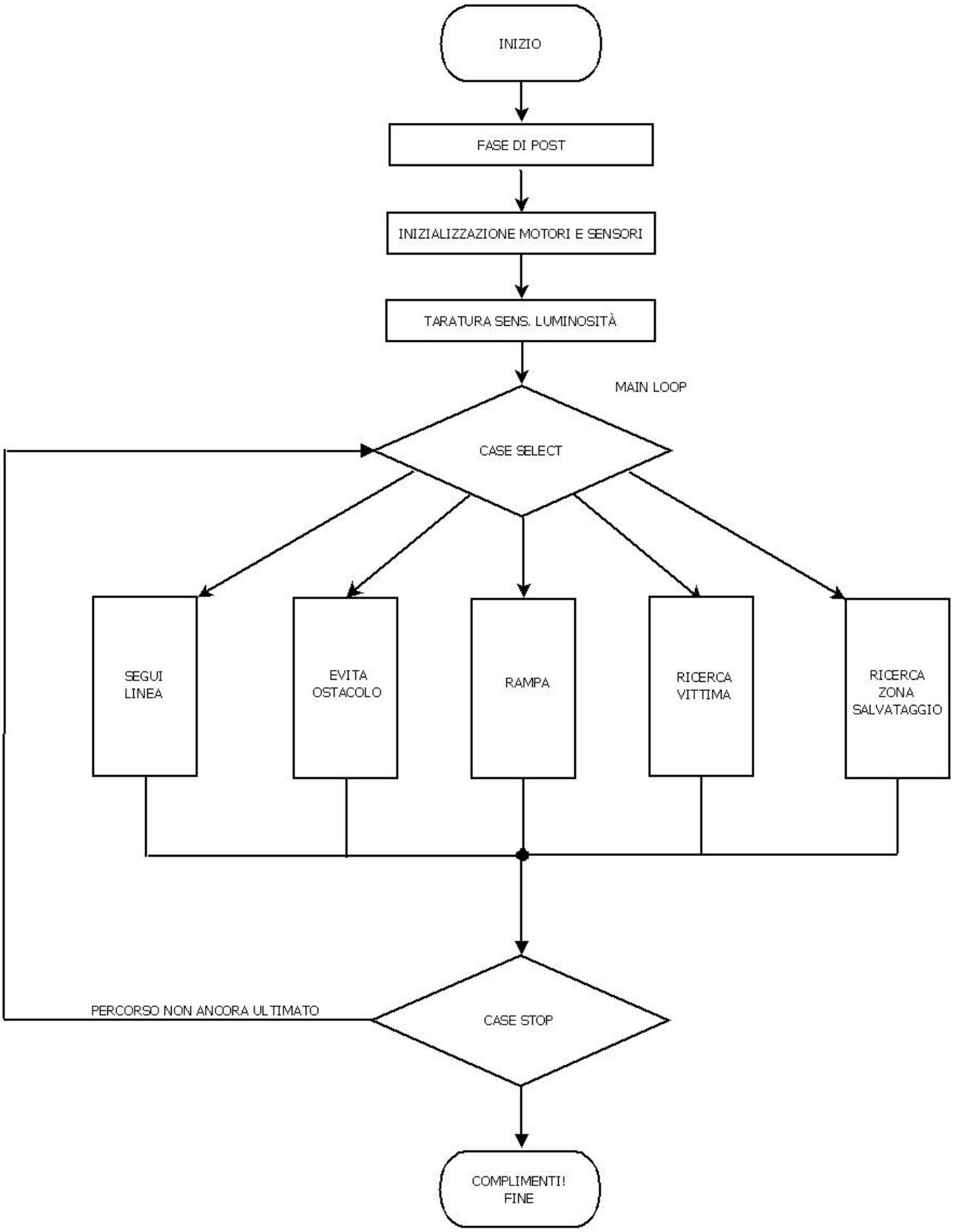
```
do
{
        //Misura luminosità
        LuxMisuraSpotCompleta();
        reflecting_victim_level = MediaTraSensori;
        Roll_Led();                     //Roll led: MISURA IN CORSO!
} while (StatoPulsante_1 == 0);

 printf("\r\nTimeOut = %d ; Reflecting Victim = %d\r\n", TimeOut, reflecting_victim_level);

do
{
        SevenSegDisplay(reflecting_victim_level);
        Pause(100);                     //DEBOUNCING DELAY!
} while (StatoPulsante_1 != 0);

ResetSevenSeg();
DatiVisualizzati = WHITE_BACKGROUND_MESSAGE;
do
{
        //Misura luminosità
        LuxMisuraSpotCompleta();
        white_background_level = MediaTraSensori;
        Roll_Led();                     //Roll led: MISURA IN CORSO!
} while (StatoPulsante_1 == 0);

printf("\r\nTimeOut = %d ; white_background = %d\r\n", TimeOut, white_background_level);

do
{
        SevenSegDisplay(white_background_level);
        Pause(100);                     //DEBOUNCING DELAY!
} while (StatoPulsante_1 != 0);

ResetSevenSeg();
DatiVisualizzati = GRAY_MESSAGE;
do
{
        //Misura luminosità
        LuxMisuraSpotCompleta();
        gray_victim_level = MediaTraSensori;
        Roll_Led();                     //Roll led: MISURA IN CORSO!
} while (StatoPulsante_1 == 0);

printf("\r\nTimeOut = %d ; gray_victim_level = %d\r\n", TimeOut, gray_victim_level);
```

```
do
{
        SevenSegDisplay(gray_victim_level);
        Pause(100);                     //DEBOUNCING DELAY!

} while (StatoPulsante_1 != 0);

ResetSevenSeg();
DatiVisualizzati = BLACK_MESSAGE;
do
{
        //Misura luminosità
        LuxMisuraSpotCompleta();
        black_trace_level = MediaTraSensori;
        Roll_Led();                     //Roll led: MISURA IN CORSO!
} while (StatoPulsante_1 == 0);
printf("\r\nTimeOut = %d; black_trace_level = %d\r\n", TimeOut, black_trace_level);
do
{
        SevenSegDisplay(black_trace_level);
        Pause(100);                     //DEBOUNCING DELAY!
} while (StatoPulsante_1 != 0);
}
```

Dal punto di vista della sequenza delle operazioni effettuate:

- STEP 1: attende prima premuta del pulsante 1 e quando ciò accade, attende che il tasto sia rilasciato prima di passare al passo successivo;
- STEP 2: Acquisisce livello superficie riflettente;
- STEP 3: Acquisisce livello superficie bianca;
- STEP 4: Acquisisce livello superficie grigia ;
- STEP 5: Acquisisce livello superficie nera.

8.6 Il diagramma a blocchi del sistema



8.6.1 Il listato del “main”

```
int main (void)
{
  print("\r\n -- Entering main() -- \r\n");
        /*************************************/

  //Spegne LED quadlux
  LedOff;

//Init periferiche GPIO su Nexys2
//Associare i driver alle porte fisiche e inizializzarle
        IO_status = XGpio_Initialize (&led8_out, led_dev_id);
        IO_status = XGpio_Initialize (&dipswitch8_in, dipswitch_dev_id);
        IO_status = XGpio_Initialize (&pulsanti_in, pulsanti_dev_id);
//Impostare la direzione dei singoli bit delle porte (0 = out, 1 = input)
        XGpio_SetDataDirection (&led8_out, CHANNEL, 0x0);
        XGpio_SetDataDirection (&dipswitch8_in, CHANNEL, 0xFFFFFFFF);
        XGpio_SetDataDirection (&pulsanti_in, CHANNEL, 0xFFFFFFFF);

//-----------------------> INIZIO TEST RISORSE BASE NEXYS2 <<--------------
//
{
        TestRisorseBase();
}
//-----------------------> FINE TEST RISORSE BASE <<----------------------

//Definizione delle VARIABILI del MAIN
        Xuint8  StatoNessie;
        Xuint16 Speed = 0;
        Xuint16 Sonar_Echo;
        Xuint16 Sonar;
        Xuint16 Direzione;
        Xuint32 Lux_Values;
/*-----------------------------------------------------------------------------
*                          FASE DI INIT
*Inserire qui tutte le inizializzazioni dell'hardware:                        */
//Inizializza le porte di IO standard presenti sulla Nexys2
         Xuint16 dati_da_visualizzare;
         dati_da_visualizzare = 0;
         DatiVisualizzati = 0;
         AccensioneDisplay = 1;
//Ritardo iniziale di accensione
Pause(100);
//Inizializza il controllo dei motori
init_motori();
avvia_motori();
set_speed_motore_destro(orario,0);
set_speed_motore_sinistro(orario, 0);
//Assegna i valori di default al controllo di luminosità
```

```
InitialSetLuxDefaultValue();

//Determina le soglie di riferimento dei livelli di riflessione
TaraLux();
//-------------------------------------------------------------
//
//          START: attende per Pulsante2 = 1
Start();
//
//Attiva SONAR A ULTRASUONI
US_SonarStartStop = 1;
//Azzera display 7 segmenti
ResetSevenSeg();
//
// PONI NESSIE NELLO STATO INIZIALE
StatoNessie = 0;
/*****************************************************************************/
/*                LOOP PRINCIPALE                                            */
do
{
        switch (StatoNessie)
        {
        case 0:
        // STATO 0 => SEGUI LINEA
                {
                //Legge misura sensore ultrasuoni
                Sonar = US_SonarMisDistanza;

                /*              LETTURA VALORE DI LUMINOSITA'
                                                */
                //Attiva lettura su tutti i sensori di luminosità
                //Consente di variare "al volo" i parametri di sensibilità
                //RunningSetLuxDefaultValue;
                //
                LuxMisuraSpotCompleta();
                //-------------------------> Imposta direzione
                Direzione = Volante ();
                DatiVisualizzati = hex_to_decimal(Direzione);
                //---------------------> Comanda i motori di conseguenza
                Pilota(Direzione);
                //
                // Solo per verifica!
                //PAUSA TEMPORANEA
                //      Pause(10000);
                //PrintLuxData();
                //
                //Verifica eventuale richiesta di "StandBy" su Pulsante3
```

```
if (StatoPulsante_3 == 1)
                {
                        stop_motori();
                        StandBy();
                        avvia_motori();
                }
                // -----------------> REGOLA VELOCITA <--------------------
                RegolaVelocita();
                // -----------------> REGOLA VELOCITA <--------------------
                //
                if (VerificaOstacoloVicino() == 1) StatoNessie = 1;
                if (RampaOn == 1) StatoNessie = 2;
                }
        break;
        case 1:
        // STATO 1 => EVITA OSTACOLO
                {
                EvitaOstacolo();
                StatoNessie = 0;
                }
        break;
        case 2:
        // STATO 2 => RAMPA
                {
                do
                        {
                        /*              LETTURA VALORE DI LUMINOSITA'
                                                */
                        LuxMisuraSpotCompleta();
                        //------------------------> Imposta direzione
                        Direzione = Volante ();
                        DatiVisualizzati = hex_to_decimal(Direzione);
                        //---------------------> Comanda i motori di conseguenza
                        Pilota(Direzione);
                        //Verifica eventuale richiesta di "StandBy" su Pulsante3
                        if (StatoPulsante_3 == 1)
                        {
                                stop_motori();
                                StandBy();
                                avvia_motori();
                        }
                        // -----------------> REGOLA VELOCITA <--------------------
                        RegolaVelocita();
                        // -----------------> REGOLA VELOCITA <--------------------
                        } while (RampaOn == 1);
                StatoNessie = 3;
                }
        break;
```

```
        case 3:
        // STATO 3 => RICERCA VITTIMA
                {
                CercaVittima();
                StatoNessie = 4;
                }
        break;
        case 4:
        // STATO 4 => RICERCA ZONA SALVATAGGIO
                {
                SalvaVittima();
                StatoNessie = 5;
                }
        break;
        default: FermaNessie();
        }
}while(StatoNessie < 5);
        FermaNessie();
        /************************************/
        print("--  URRAAAHHH!!!!!  --\r\n");
        /************** HALT: COMPLIMENTI !!!!! ******************/
  return 0;
}
```

CAP.9 ALIMENTAZIONE

Nessie è alimentato interamente da batterie ricaricabili durante la gara, ma è stato progettato in modo tale da poter essere alimentato anche attraverso un classico alimentatore da 9V (soprattutto nelle fasi di test).
Le batterie che alimentano il robot sono di due tipologie:

- le Sanyo NI-MH da 2700 mA/ora, di categoria AA, che servono per alimentare la Nexys 2 e tutte le periferiche ad esso associate. Se ne utilizzano 8 per avere una tensione di circa il doppio di 5V, ovvero la tensione necessaria per alimentare i regolatori di tensione 7805TV.

  8 batterie * 1,2V = 9,6V

- I due motori che muovono il robot e il motore della pinza, invece, viene alimentato tramite 6 batterie di categoria C. Esse sono da 3100 mA/ora, e alimentano i motori con 7,2V.

  6 batterie * 1,2V = 7,2V

Ogni alimentazione ha un suo interruttore in modo tale da non dover consumare le batterie dei motori in fase di testing dei sensori che non sfruttano il movimento del robot.

INDICE: