Università degli Studi di Padova

Facoltà di Ingegneria

Dipartimento di Tecnica e Gestione dei Sistemi Industriali

Corso di laurea triennale in Ingegneria

Meccanica e Meccatronica

Curriculum Meccatronico

Tesi di Laurea

Freescale Cup: Trasmissione dati Bluetooth tra PC e microcontrollore per un veicolo autonomo

Laureando:

Francesco Dal Santo

614105 – IMM

Relatore:

Ch.mo Prof.

Roberto Oboe

Anno Accademico: 2014 – 2015

Sommario

L'idea di aderire alla Freescale Cup nasce dalla volontà di riuscire a mettere in pratica i numerosi corsi approfonditi durante il Triennio di Ingegneria Meccatronica. L'opportunità di partecipare a questa competizione viene data principalmente agli studenti del terzo anno, in quanto più "formati" dal punto di vista delle conoscenze, utili ad affrontare questo tipo di progetto. Il fatto di lavorare in team, da per la prima volta una concreta possibilità agli studenti di confrontarsi tra loro attraverso uno scambio reciproco di idee ed esperienze, anche con gli avversari, offrendo così la possibilità di affrontare problemi difficilmente riscontrabili in aulee universitarie.
L'Università di Padova, con l'edizione del 2014, partecipa per la seconda volta a questa competizione Internazionale. Infatti dal team che ha gareggiato l'anno precedente, siamo riusciti a portar avanti il loro progetto e ad introdurre delle piccole, ma significative migliorie come l'implementazione via software di *interrupts*, un algoritmo più "robusto" per la lettura delle telecamere nonché l'introduzione di un dispositivo Bluetooth per la comunicazione tra il veicolo ed il PC. Questo ha richiesto una conoscenza approfondita di tutti i componenti utilizzati per il progetto ed i relativi manuali, ma soprattutto una certa dimestichezza con il linguaggio di programmazione "*C*" ed i sistemi *embedded.*
Il seguente elaborato non si prefigge l'obiettivo di entrare nel dettaglio riguardo la parte di controllo e di funzionamento del veicolo, ma di trattare approfonditamente il modulo Bluetooth utilizzato, l'interfaccia con un PC, i protocolli di comunicazione per interfacciare le periferiche utilizzate e la relativa programmazione software.
Ultimo, ma non meno importante, la possibilità di utilizzare questo lavoro come punto di partenza per implementare un algoritmo più efficiente per lo scambio dei dati e quindi l'opportunità di ottenere un veicolo con prestazioni più elevate in fase di *tuning.*

Indice

Capitolo 1

Introduzione

1.1 FREESCALE CUP

La Freescale Cup è una competizione mondiale che permette agli studenti di collaborare, competere e mettere in pratica le conoscenze riguardo i sistemi *embedded* e i controlli automatici.
Creata nel 2003, conta ora più di 20.000 studenti iscritti ogni anno e da ogni parte del mondo: India, Cina, Malesia, Giappone, Nord America, Sud America ed Europa (EMEA). Nel 2013, alla "Freescale Cup" Europea hanno partecipato 34 Università a partire dal Marocco fino alla Russia con 290 studenti iscritti da 11 stati diversi. Ogni squadra può essere composta da studenti laureandi (da un minimo di 2 ad un massimo di 4) oppure da 2 studenti laureandi ed un laureato e deve essere iscritta sotto il nome dell'università partecipante con l'ausilio di un coordinatore.

Questo offre l'opportunità alle squadre di studenti di costruire, programmare e gareggiare con veicoli in scala, con l'obiettivo di completare il percorso nel minor tempo possibile senza uscire. Il percorso è composto da pannelli con fondo bianco e larghi 50cm con al centro una linea guida nera di 2,5cm, snodato su una sequenza casuale di curve, dossi, incroci, tunnel, chicane e bande sonore. Per aumentare la difficoltà della competizione, ogni anno il tracciato viene modificato.

Figura 1.1 - Tracciati di prova della finale europea 2014

Il kit fornito dalla Freescale per la competizione consiste in:

- Un Telaio di un veicolo in scala 1/18
- Due motori
- Un servo motore per lo sterzo
- Una telecamera a sensori CMOS
- Una scheda di potenza
- Un Microcontrollore

Da regolamento ogni team può integraree aggiornare il kit con ulteriori sensori per migliorarne l'affidabilità e le prestazioni durante il funzionamento. Nell'elaborato verranno riportate le periferiche aggiunte per il progetto.

1.2 PROBLEMA CONSIDERATO

L'obiettivo della competizione "Freescale Cup" è quella di realizzare un veicolo, che attraverso un blocco di *sensing*, riesca a rilevare la posizione della linea nera posta al centro del tracciato bianco durante la sua corsa. Il microcontrollore a bordo della *smart car* deve essere quindi in grado di elaborare i dati provenienti dalla telecamera e conseguentemente impostare un opportuno angolo di sterzo al servo motore ed un riferimento di velocità ai due motori di trazione. La difficoltà sta nel fatto che il veicolo non deve solamente seguire la linea nera, ma anche percorrere il tracciato nel minor tempo possibile, che è l'obiettivo principale. Questa operazione viene svolta nella fase finale di *tuning*, ovvero di affinamento dei parametri utilizzati dal microcontrollore che ne determinano così lo "stile di guida".
L'idea quindi di utilizzare un dispositivo di comunicazione Wireless nasce dall'esperienza di un gruppo di studenti di Vicenza, che partecipando all'edizione 2013 della "Freescale Cup", hanno riscontrato una notevole difficoltà proprio in questa fase. Inizialmente questa operazione era svolta facendo girare il veicolo nel tracciato di prova ed una volta identificato il problema, lo si doveva fermare, collegare al PC per poterne aggiornare il programma con la nuova configurazione e così via, fino ad arrivare a quella ottimale con un notevole dispendio di tempo.
Il problema considerato è stato così risolto implementando un dispositivo Bluetooth sul microcontrollore freescale a bordo del veicolo, creando così una comunicazione wireless con il PC e questo permette di modificare *real-time* i parametri del programma mentre il veicolo gira e ottenere così un riscontro immediato delle modifiche apportate.

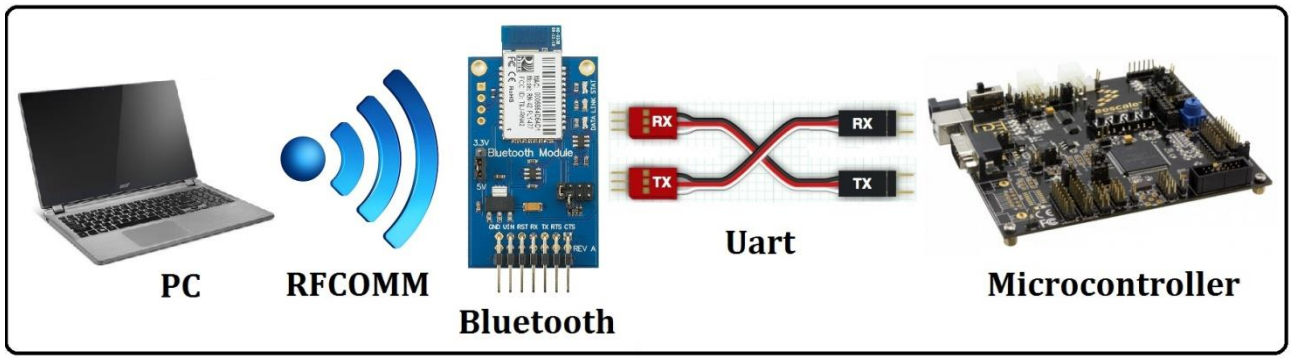

Figura 1.2 – Architettura del sistema di comunicazione tra PC e Microcontrollore

Inizialmente prima di sviluppare l'algoritmo, abbiamo riflettuto su quali fossero i dati della quale avessimo bisogno con il fine di ottimizzare il "comportamento" su strada del veicolo. Dopo un'attenta analisi e numerose prove, siamo riusciti a capire che sarebbe stato di fondamentale importanza ottenere in tempo reale la lettura dei sensori, acquisirne le variabili interne memorizzate nell'algoritmo del microcontrollore e spedirle ad una stazione *host* (PC) per poter farle interpretare dall'utente e quindi rielaborare. Da qui la necessità di sviluppare un software *client* (PC) per la ricezione dei dati provenienti dalla telecamera e dall'encoder ed una volta identificata la correzione da eseguire sul veicolo, poter modificare i parametri utilizzati dagli attuatori per migliorarne la dinamica. L'organizzazione del trasferimento dei dati avviene nel seguente modo: come si vede dalla *figura 1.2*, tramite apposito canale, l'utente ha la possibilità di scegliere da PC quali parametri leggere/modificare e successivamente tramite il protocollo RFCOMM inviarne la richiesta al dispositivo Bluetooth. Quest'ultimo simula una porta seriale RS-232 e ne trasferisce il buffer dati al microcontrollore che li riceve tramite la sua periferica seriale UART e genera così una richiesta di *interrupt* che permette di entrare nella routine di lettura/modifica dei parametri, precedentemente acquisiti dal blocco di *sensing*.

In base al comando scelto dell'utente, i dati in lettura, son subito rispediti al PC, mentre quelli in modifica vanno ad agire sulle variabili utilizzate in quel momento dal microcontrollore potendone così modificare, da tastiera, ad esempio la velocità di rettilineo, curva ed i PID del controllore. La possibilità di effettuare in tempo reale la telemetria del veicolo permette di capire istantaneamente se le variazioni apportate hanno l'effetto desiderato e come detto precedentemente, questo permette di risparmiare una notevole quantità di tempo, non che ottenere un *tuning* più adeguato.

1.3 STRUTTURA DELL'ELABORATO

L'obiettivo di questo elaborato esula dalla descrizione nel dettaglio di hardware, sensori e attuatori utilizzati per il funzionamento del veicolo, ma sarà focalizzato su come, partendo da zero, si possa ottenere un trasmissione dati un tra PC (munito di periferica Bluetooth) ed il microcontrollore della freescale (TRK MCB5604B) tramite l'integrazione di un dispositivo esterno Bluetooth. La descrizione dell'hardware freescale e delle relative periferiche connesse risulta essere necessaria al fine di comprendere nel dettaglio di come il dispositivo (Bluetooth) interagisca con quest'ultime e quali siano i vantaggi rispetto l'utilizzo di una più semplice connessione USB soprattutto nella fase finale di "tuning" del veicolo.
Con il presente documento si vuole fornire una solida base di partenza per gli studenti interessati a partecipare alla freescale cup, con la speranza che risulti utile e dia loro la possibilità di implementare un algoritmo più efficiente.
La comprensione di questo elaborato richiede le conoscenze di base della meccatronica, ricevute durante il triennio di studi, come l'elettronica, la teoria dei circuiti digitali, i controlli automatici, gli azionamenti elettrici, le misure per l'automazione, alcune nozioni sulla dinamica del veicolo, ma nello specifico una certa dimestichezza con i linguaggi di programmazione per sistemi *embedded*, soprattutto con il "C" e l'ambiente linux. Di seguito l'elaborato sarà suddiviso in capitoli per agevolare il lettore nella comprensione e avere una più ampia visione d'insieme degli argomenti trattati.

Capitolo 2: Microcontrollore

In questo capitolo verrà trattato il funzionamento del microcontrollore *TRK MCB5604B*, prodotto da Freescale Semiconductor™ e l'analisi dei relativi blocchi funzionali. Il primo di questi è l'*eMIOS* che, configurato opportunamente, permette di acquisire i dati provenienti dai sensori, di elaborarli secondo l'algoritmo di controllo e di inviarne i riferimenti agli attuatori per il funzionamento del veicolo. Successivamente si pone maggiore attenzione ai blocchi principalmente utilizzati per interfacciare il modulo Bluetooth come l'*INTC* che permette al microcontrollore di gestire le richieste di *interrupt* e di assegnarne una priorità, nel caso ce ne fossero più contemporaneamente. Infine viene esposto il blocco più importante per questo progetto, che è la *LINFlex*. Quest'ultimo programmato in modalità *UART* consente di inizializzare una comunicazione dati seriale con un buffer di 8 byte, di cui 4 in ricezione e 4 in trasmissione.
L'inizializzazione dei registri verrà ripresa nel capitolo riguardo la programmazione software, dove verrà riportato il registro stesso e il relativo codice in linguaggio *C*.

Capitolo 3: Sensori ed Attuatori

Il microcontrollore si interfaccia con l'ambiente esterno tramite i sensori che, sono una parte essenziale di tutti i sistemi di acquisizione e per questo progetto ne sono stati utilizzati due. Il primo è *la Line Scan Camera TSL1401CL*, prodotta dalla *TAOS* e presente nel Kit fornito dalla Freescale. Questa è l'unico *input* che permette al veicolo di monitorare la posizione della linea guida sul tracciato. Il secondo è un *encoder*, uno per ogni ruota e permette di monitorare istante per istante la velocità tramite appositi sensori.
La seconda parte si occupa degli attuatori interfacciati con il microcontrollore. Il primo è il *Motor Drive Board* che è la parte di potenza, staccata dalla parte di controllo e comanda i due motori in corrente continua del tipo Standard Motor: "rn 260 c" winding 18130 mediante due ponti ad H MC33931. Il secondo è il *Servo Motor Futaba S3010* che tramite apposito comando, imposta un determinato angolo allo sterzo del veicolo.

Capitolo 4: Bluetooth

Questo capitolo, assieme a quello esposto successivamente, saranno trattati in modo approfondito non che il fulcro di questo elaborato. Nella prima parte verranno esposte le generalità, in modo da introdurre il lettore nell'ottica di quello che sono i dispositivi Bluetooth descrivendone le caratteristiche principali e comprenderne il funzionamento. Nel paragrafo successivo si parlerà nel dettaglio del modulo utilizzato per il progetto che è l' RN-42 distribuito dalla *Parallax* e la relativa configurazione dei jumpers per preparalo alla comunicazione L'ultimo paragrafo descrive i due protocolli di comunicazione utilizzati in questo progetto. Il primo è l'*UART,* che realizza una comunicazione seriale via cavo tra il microcontrollore e il dispositivo Bluetooth e trasmette i byte di dati, un bit alla volta in modo sequenziale. Mentre l'interfaccia tra il Bluetooth e il PC è realizzato tramite il protocollo RFCOMM, che simula le impostazioni di una linea seriale via cavo di una porta del tipo RS-232.

Capitolo 5: Software

Nella prima metà di questo capitolo si parla del programma *Client.c* eseguito su Ubuntu, quindi nel lato PC, che tramite la propria Shell permette di visualizzare a video una serie di comandi divisi in due blocchi. Digitando da tastiera comandi compresi tra 0 e 40, permette di entrare nella routine per la sola stampa a video e questo risulta utile per monitorare come variano i parametri durante il funzionamento del veicolo. Per valori di comando superiori al 40 si entra nella routine per la sola modifica dei parametri che risulta essere fondamentale soprattutto nella fase di *tuning* del veicolo. Infatti una volta individuato il problema, permette all'utente di ottimizzare in *Real-Time* il funzionamento del veicolo.
Nella seconda metà invece verrà spiegato come i dati inviati da PC e ricevuti tramite il dispositivo Bluetooth, vengono elaborati dall'algoritmo di ricezione/modifica eseguito nel microcontrollore. Questa parte scritta in linguaggio *C* e compilata tramite *CodeWarrior* è suddivisa in quattro *Source:* IntcInterrupts.c per l'installazione di un interrupt via software, *Ivor_branch_table* che memorizza la posizione per eseguire la funzione associata all'ISR, l'*Uart.c* che permette di inizializzare una trasmissione dati tra l'UART e una periferica esterna e in fine il main.c che richiama tutte le funzioni implementate precedentemente.

Capitolo 2

Microcontrollore

2.1 MICRO CONTROLLER UNIT (MCU)

Il modello fornito per la competizione è il TRK MCB5604B prodotto da Freescale Semiconductor™. Rappresenta una nuova generazione di microcontrollori a 32 bit basati su Power Architecture® ed appartiene ad una famiglia più ampia di prodotti ideati per applicazioni *automotive*, mirati ad affrontare e gestire il numero sempre in crescita di applicazioni elettroniche a bordo del veicolo. Questo microcontrollore lavora a 64 MHz ed offre alte prestazioni di processing ottimizzate per applicazioni che prediligono il basso consumo di energia.

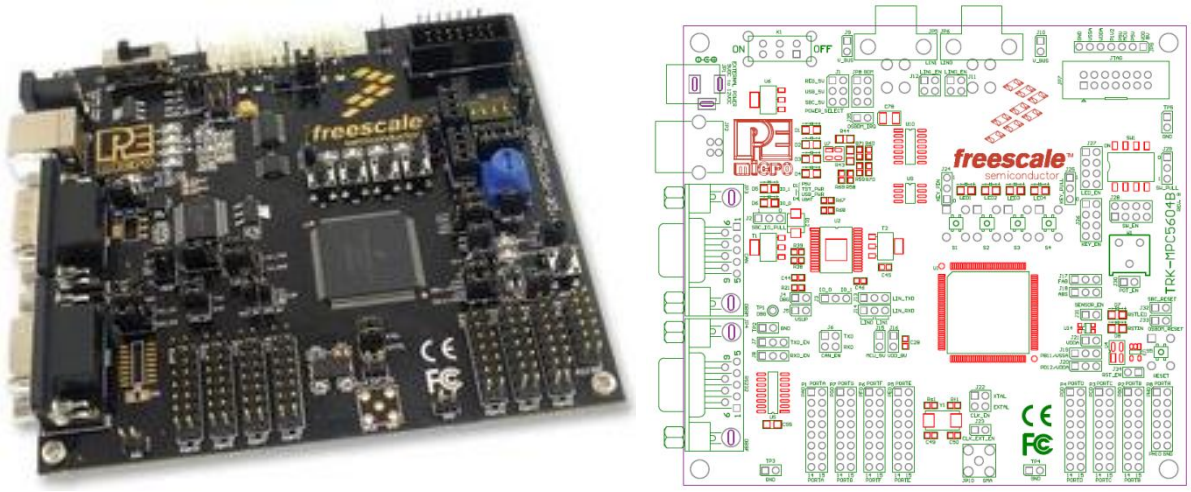

Figura 2.1 – Microcontrollore TRK MCB5604B

Nel relativo data sheet sono riportate le seguenti specifiche:

CORE:

- PowerPC e200z0 core running 48-64MHz
- VLE ISA instruction set for superior code density
- Vectored interrupt controller
- Memory Protection Unit with 8 regions, 32 byte granularity

MEMORY :
- 512 Kbyte embedded program Flash, 64 KByte data flash
- 64 Kbyte embedded data Flash (for EE Emulation)
- Up to 64MHz non-sequential access with 2WS
- ECC-enabled array with error detect/correct
- 48 Kbyte SRAM (single cycle access, ECC-enabled)

COMMUNICATIONS :
- 3x enhanced FlexCAN
- 64 Message Buffers each, full CAN 2.0 spec
- 4x LINFlex
- 3x DSPI, 8-16 bits wide & chip selects
- 1xI2C

ANALOG :
- 5V ADC 10-bit resolution

TIMED I/O :
- 16-bit eMIOS module

OTHER:
- CTU (Cross Triggering Unit) to sync ADC with PWM Channels
- I/O: 5V I/O, high flexibility with selecting GPIO functionality
- Packages: 100LQFP, 144LQFP, 208MAPBGA (Development only)
- Boot Assist Module for production and bench programming

I principali clock di sistema provengono da tre sorgenti:

- Fast External Oscillator 4-16 MHz (FXOSC)
- Fast Internal RC Oscillator 16 MHz (FIRC)
- Frequency Modulated Phase Locked Loop (FMPLL)

Il microcontrollore è dotato inoltre di ulteriori due oscillatori a bassa potenza:

- Slow Internal RC Oscillator 128 kHz (SIRC)
- Slow External Crystal Oscillator 32 kHz (SXOSC)

Schema di funzionamento

Nel layout dello schema si notano i principali blocchi che compongono il microcontrollore. In particolare si nota la CPU connessa alle memorie interne del µC tramite la "Crossbar Switch", i blocchi computazionali, i blocchia ausiliari per la generazione dei clock del sistema e la gestione degli interrupt (INTC). In basso si può notare la "Peripheral Bridge" che collega tutti i blocchi di I/O come l'ADC, l'eMIOS e la LINFlex.

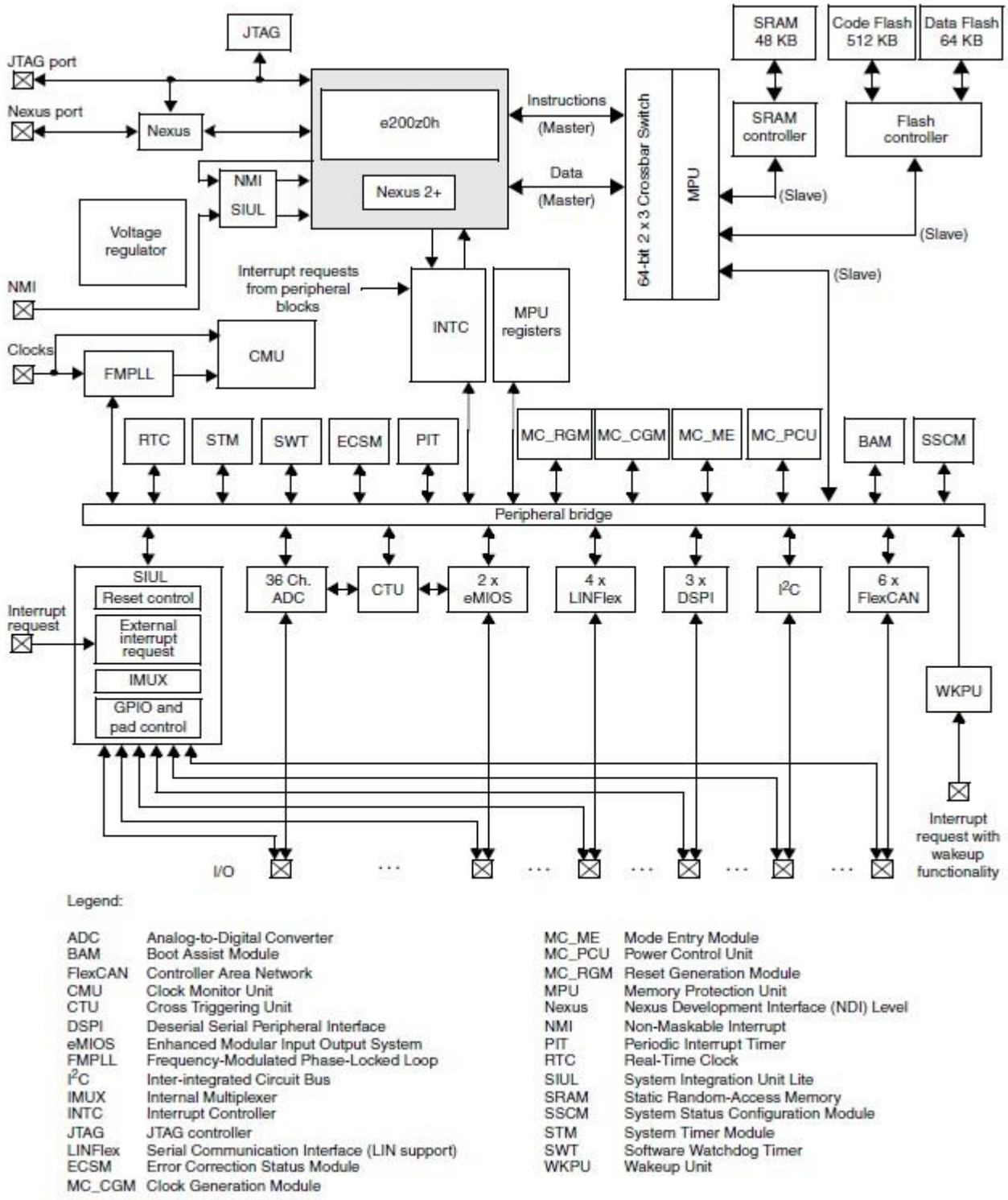

Figura 2.2 - Schema a blocchi ad alto livello della famiglia MPC5604B

2.2 Architettura dei moduli eMIOS

Ogni eMIOS offre una combianazione di PWM, immagazzinamento degli Output e funzioni di comparazioni degli ingressi. Ci sono diversi tipi di canali implementati e non tutti supportano tutte le funzioni dell'eMIOS, inoltre le funzionalità di ogni canale dipendono dal modulo eMIOS considerato. Ogni canale ha il suo contatore indipendente a 16-bit. Per permettere la sincronizzazione tra i diversi canali ci sono una serie di *counter busses* che possono essere usati come un riferimento comune per la temporizzazione. I *counter busses* possono essere utilizati con i contatori individuali dei canali per fornire caratteristiche avanzate. Il modulo eMIOS è temporizzato con il clock del sistema a 64 MHz.
A titolo di esempio viene riportata una delle tre mappature esistente tra i canali eMIOS_0 e i pins di output disposti nelle porte:

Channel	Pin function			Channel	Pin function		
	ALT1	ALT2	ALT3		ALT1	ALT2	ALT3
UC[0]	PA[0]			UC[16]	PE[0]		
UC[1]	PA[1]			UC[17]	PE[1]		
UC[2]	PA[2]			UC[18]	PE[2]		
UC[3]	PA[3], PB[11]			UC[19]	PE[3]		
UC[4]	PA[4], PB[12]			UC[20]	PE[4]		
UC[5]	PA[5], PB[13]			UC[21]	PE[5]		
UC[6]	PA[6], PB[14]			UC[22]	PE[6], PF[5]	PE[8]	
UC[7]	PA[7], PB[15]			UC[23]	PE[7], PF[6]	PE[9]	
UC[8]	PA[8]			UC[24]	PG[10]	PD[12]	
UC[9]	PA[9]			UC[25]	PG[11]	PD[13]	
UC[10]	PA[10], PF[0]			UC[26]	PG[12]	PD[14]	
UC[11]	PA[11], PF[1]			UC[27]	PG[13]	PD[15]	
UC[12]	PC[12], PF[2]						
UC[13]	PC[13], PF[3]						
UC[14]	PC[14], PF[4]						
UC[15]	PC[15]						

Figura 2.3 - eMIOS_0 channel to pin mapping

Caratteristiche dei moduli eMIOS:

- 2 blocchi eMIOS da 28 canali ciascuno
- 50 canali dispongono di funzionalità di tipo Output PWM Triggered (OPWMT)
- 6 canali con funzioni esclusivamente di tipo IC/OC
- Un Prescaler globale
- Registri a 16 bit
- 10 counter buses a 16 bit
- I contatori B,C,D,E possono essere comandati solo dai Unified Channel 0, 8, 16 e 24 rispettivamente
- Il contatore A può essere comandato solo dal Canale 23

Modalità operative:

I canali unificati possono essere configurati per operare nei seguenti modi:

- Single Action Input Capture (SAIC)

- Single Action Output Compare (SAOC)
- Input Pulse Width Measurement (IPWM)
- Input Period Measurement (IPM)
- Double Action Output Compare (DAOC)
- Modulus Counter (MC)
- Modulus Counter Buffered (MCB)
- Output Pulse Width and Frequency Modulation Buffered (OPWFMB)
- Output Pulse Width Modulation Buffered (OPWMB)
- Output Pulse Width Modulation with Trigger (OPWMT)
- Center Aligned Output Pulse Width Modulation Buffered (OPWMCB)

I Canali eMIOS possono essere configurati via software per operare in uno dei seguenti modi come illustrato in figura 2.4:

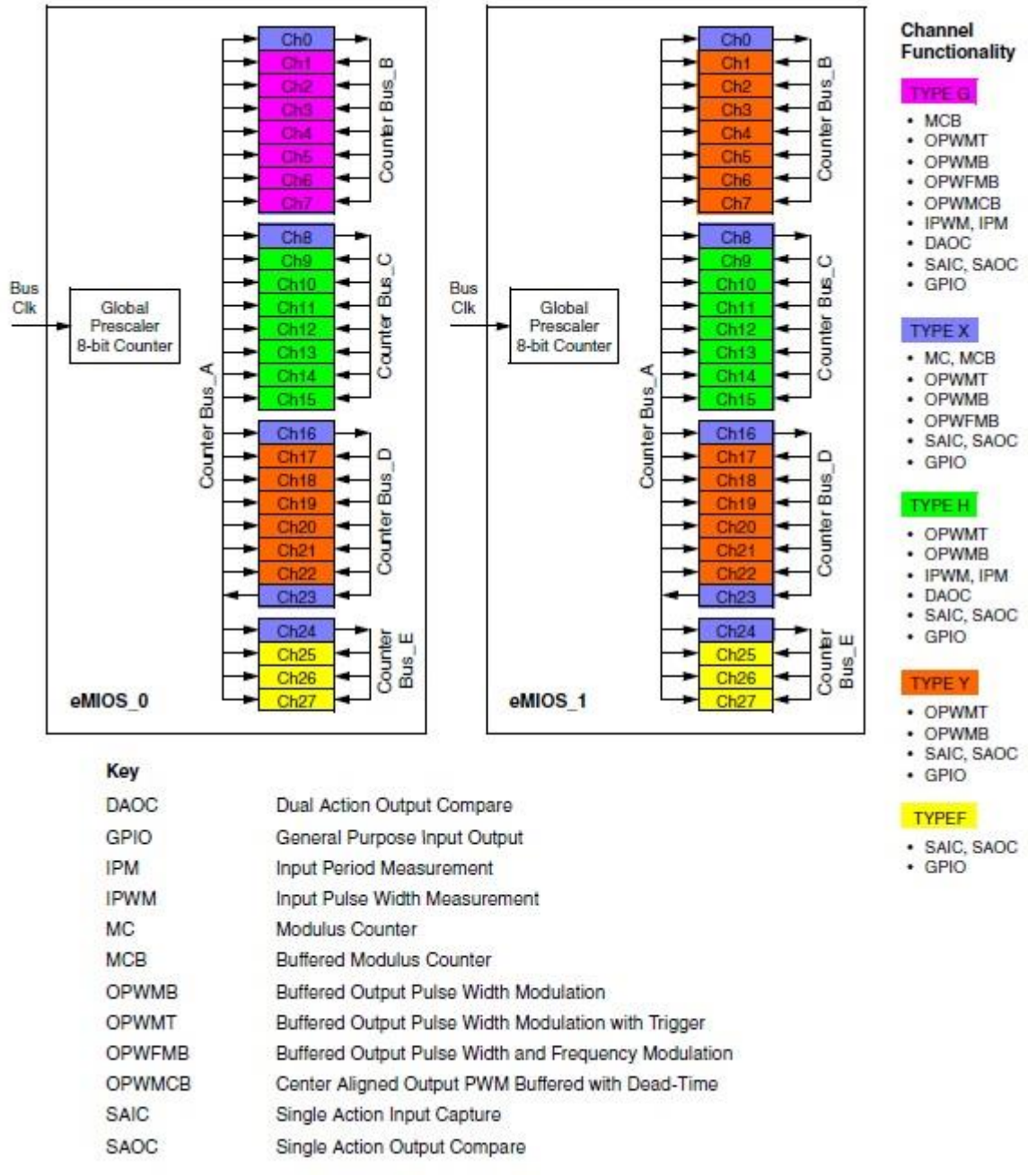

Figura 2.4 – Diverse modalità di funzionamento dei canali eMIOS

Selezione della modalita' di funzionamento dei canali

Le modalità di funzionamento dei canali possono essere impostate tramite i *mode selection bits* MODE[0:6] negli *eMIOS UC Control Register* come evidenziato in figura 2.5:

MODE [0:6]	Mode of operation
0000000	General purpose Input/Output mode (input)
0000001	General purpose Input/Output mode (output)
0000010	Single Action Input Capture
0000011	Single Action Output Compare
0000100	Input Pulse Width Measurement
0000101	Input Period Measurement
0000110	Double Action Output Compare (with FLAG set on B match)
0000111	Double Action Output Compare (with FLAG set on both match)
0001000 – 0001111	Reserved
001000b	Modulus Counter (Up counter with clear on match start)
001001b	Modulus Counter (Up counter with clear on match end)
00101bb	Modulus Counter (Up/Down counter)
0011000 – 0100101	Reserved
0100110	Output Pulse Width Modulation with Trigger
0100111 – 1001111	Reserved
101000b	Modulus Counter Buffered (Up counter)
101001b	Reserved
10101bb	Modulus Counter Buffered (Up/Down counter)
10110b0	Output Pulse Width and Frequency Modulation Buffered
10110b1	Reserved
10111b0	Center Aligned Output Pulse Width Modulation Buffered (with trail edge dead-time)
10111b1	Center Aligned Output Pulse Width Modulation Buffered (with lead edge dead-time)
11000b0	Output Pulse Width Modulation Buffered
1100001 – 1111111	Reserved

Figura 2.5 – Tabella delle modalità di funzionamento dei canali

2.3 Interrupt Controller (INTC)

2.3.1 Introduzione e schema di funzionamento

L'INTC fornisce una programmazione basata sulla priorità dell'ISR (Interrupt Service Request) ed è uno schema di programmazione adatto per sistemi con principale funzionamento in *real-time.*
L'INTC supporta 142 Interrupt request e sono mirati per funzionare con processori con *Power Architecture technology* e applicazioni *Automotive* dove l'ISR si annida su diversi livelli, ma anche con altri processori e applicazioni.
Per richieste di Interrupt con priorità elevate, il tempo trascorso tra la dichiarazione della richiesta da parte della periferica a quando il processore la soddisfa deve essere ridotto al minimo. Per questo l'INTC utilizza un solo vettore per l'ISR e inoltre fornisce 16 livelli di priorità in modo tale da non ritardare le richieste con priorità più elevata e sono programmabili tramite software.

Caratteristiche:

- Supporta 134 periferiche e 8 fonti di richiesta di interrupt configurabili via software
- Unico vettore a 9 bit per ogni sorgente di interrupt
- Ogni sorgente di interrupt può essere programmato per una delle 16 priorità
- Prelazione
  - Richieste priorizzate di interrupt al processore
  - ISR con priorità più alta precedono l'esecuzione di ISR con priorità inferiore
  - Uscita automatica da priorità ostacolate verso o da un LIFO
  - Possibilità di modificare la priorità dell'ISR o dell'attività. Modificare la priorità può essere utilizzata per implementare il *priority ceiling protocol* per l'accesso a risorse condivise.
- Bassa latenza - 3 colpi di clock dal ricevimento della richiesta di *interrupt* dalla periferica all'*interrupt* richiesto dal processo.

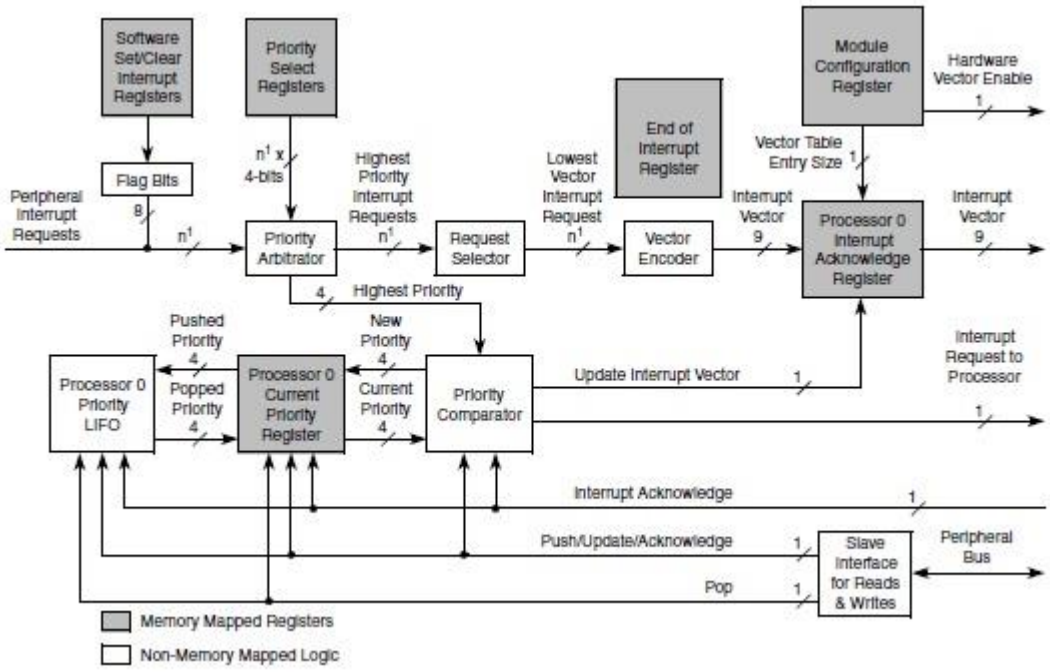

Figura 2.6 - Schema a blocchi dell'INTC

2.3.2 Interrupt Service Request (ISR)

Tutti i *controller real-time* in interazione con il loro ambiente possono operare interrompendo il loro programma in corso. L'esecuzione di alcune funzione dipendono da eventi esterni (per esempio la pressione di un tasto, la ricezione di un segnale, la rilevazione di una soglia di tensione ecc…). Gli ISR (Interrupt Service Request) sono predefiniti e associati a periferiche hardware, resets e richieste software.
Quando la CPU rileva le condizioni per abilitare un Interrupt, può essere eseguita una funzione dedicata al processo di ISR, che dipende dalla configurazione dell'interrupt (Interrupt *enabled* oppure non abiltato se è *maskable*), dal contenuto dell'IVR (Interrupt Vector Table) e dal livello di priorità dell'ISR.
La gestione degli Interrupt è complessa ed è svolta dal Controllore INTC (Interrupt Controller) che mira a pianificare l'ISR, per esempio:

- Notificare alla CPU che un ISR è trasmesso da una periferica o da software.
- Gestire le priorità tra i diversi ISR in arrivo.
- Trasmettere alla CPU gli indirizzi di programma per elaborare l'Interrupt.

2.3.3 Gestione di un Interrupt

Di seguito viene mostrato come è gestita una richiesta di *Interrupt* e la posizione del blocco INTC. Nel *Core* dell'MCU (Micro Controller Unit) e200z0h, i registri IVOR (Interrupt Vector Offset Register) formano una tabella ramificata (IVOR_Branch_Table) la quale gestisce le diverse eccezioni che avvengono durante le operazioni dell' MCU. L'IVOR4 è il registro utilizzato per la gestione degli *Interrupt*.

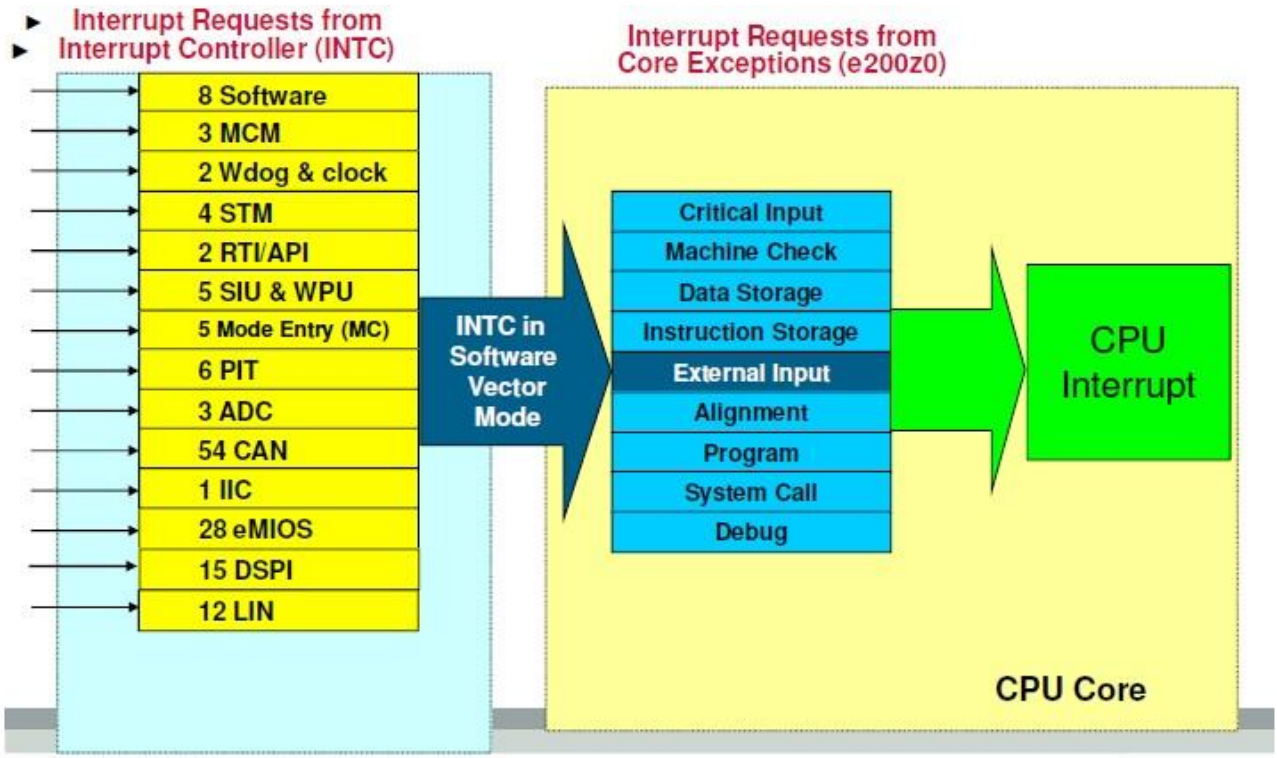

Figura 2.7 - Gestione della richiesta di Interrupt da parte del MCU

Il modulo INTC del MPC5604B gestisce l'ISR in base alle loro priorità programmabili e innesca eccezioni dell'IVOR4. Di seguito si mostra come un ISR viene gestito via Software.

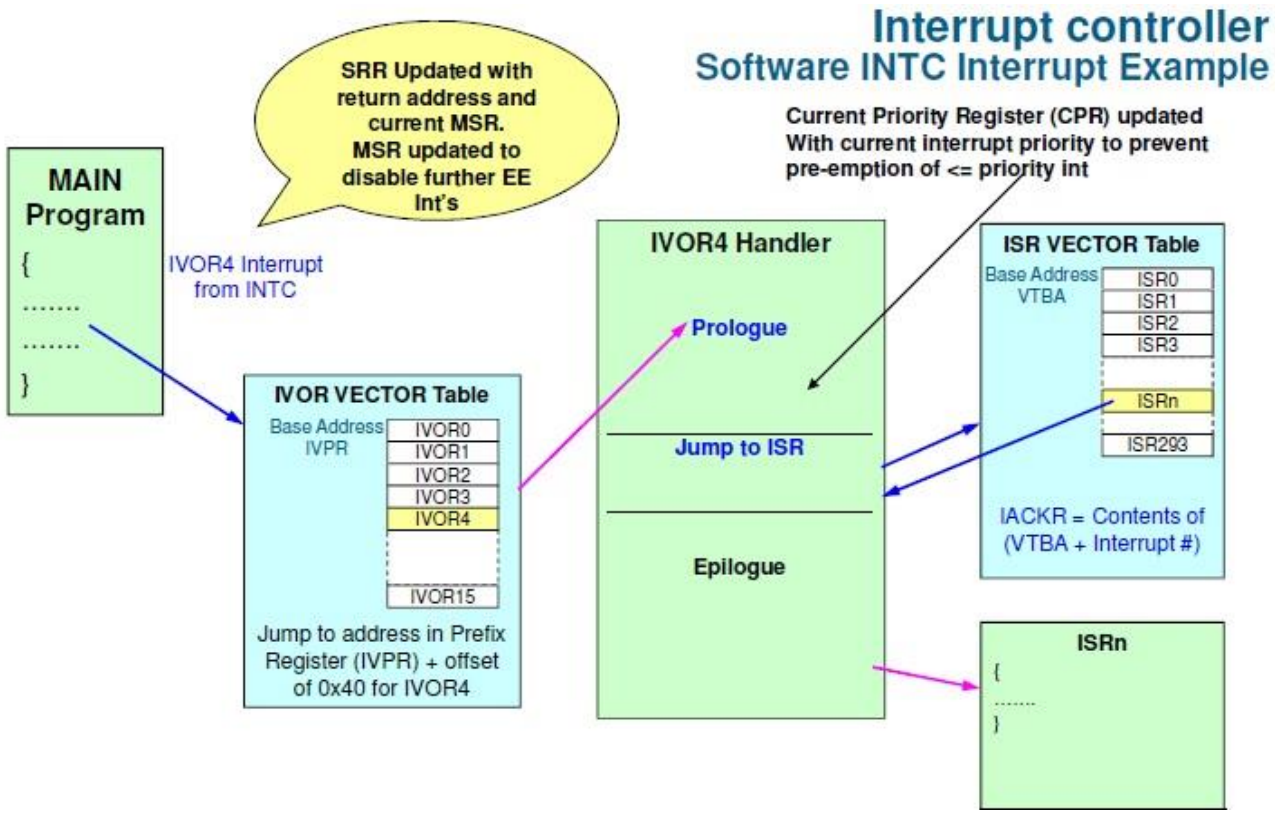

Figura 2.8 - ISR gestito in modalità Software

2.4 LIN Controller (LINFlex)

2.4.1 Caratteristiche principali in modalità LIN e UART

La periferica LINFlex (Local Interconnect Network Flexible) si interfaccia con la rete LIN e supporta il protocollo LIN nelle versioni 1.3, 2.0 , 2.1 e j2602 sia in modalità master/slave. La LINFlex prevede una modalità LIN che fornisce caratteristiche aggiuntive, comparate con lo standard UART, per diminuire l'utilizzo della LIN e quindi un miglioramento della robustezza del sistema minimizzando il carico verso la CPU.

Caratteristiche in modalità LIN

- Supporta le versioni del protocollo LIN 1.3, 2.0, 2.1 e J2602
- Modalità Master con gestione dei messaggi autonoma
- Classica e rafforzata verifica dei calcoli e controllo
- Buffer Single 8 byte per la trasmissione / ricezione
- Modalità *frame* estesa per In-Application Programming (IAP)
- Wake-up event sulla rilevazione di un bit dominante
- Vera macchina a stati con LIN *field*
- Rilevazione avanzata degli errori nella LIN
- Header, response and frame timeout
- Slave mode1
    - Gestione autonoma degli *header*
    - Gestione autonoma dei dati in ricezione/trasmissone.
- Risincronizzazione automatic della LIN, permette operazioni con oscillatori veloci a 16 MHz del tipo RC come risorsa di clock
- 16 filtri identificatori per la gestione autonoma dei messaggi in Slave mode.

Caratteristiche in modalità UART

- Comunicazione del tipo *Full duplex*
- Struttura dati a 8 o 9 bit, con bit di parità
- 4-byte di *buffer* in ricezione, 4-byte di *buffer* in trasmissione
- Contatore a 8-bit per la gestione del *timeout*

Caratteristiche comuni tra LIN e UART

- Generatore di baud rate frazionario
- 3 modalità di funzionamento per il risparmio energetico e di configurazione dei registri:
    - Initialization
    - Normal
    - Sleep
- 2 modalità di test:
    - Loop Back
    - Self Test
- *Interrupt* mascherabile

2.4.2 Descrizione generale e schema di funzionamento

Il crescente numero di periferiche di comunicazione *embedded* nei microcontrollori come per esempio CAN, LIN ed SPI, richiedono sempre più risorse della CPU per la gestione delle comunicazioni. Perfino un microcontrollore a 32-bit è sovraccaricato se le sue periferiche non forniscono prestazioni *high level* per gestire autonomamente le comunicazioni. Ad esempio un protocollo LIN con un baud rate di 20 Kbit/s è relativamente lento, ma questo genera un carico non trascurabile sulla CPU se la LIN è implementata su una una UART di tipo standard, come nella maggior parte dei casi.

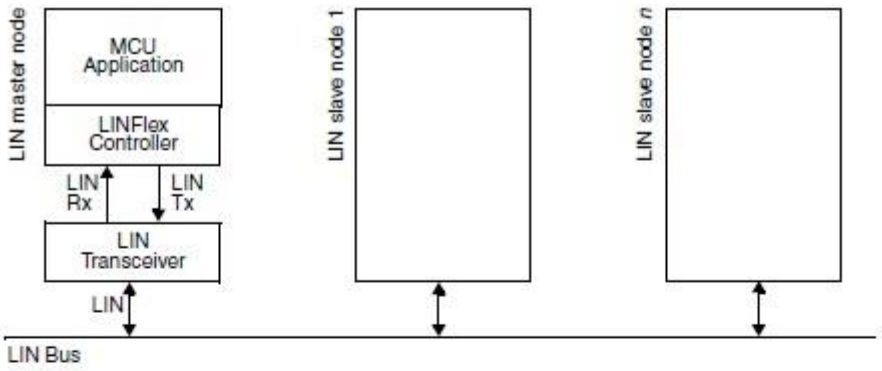

Figure 21-1. LIN topology network

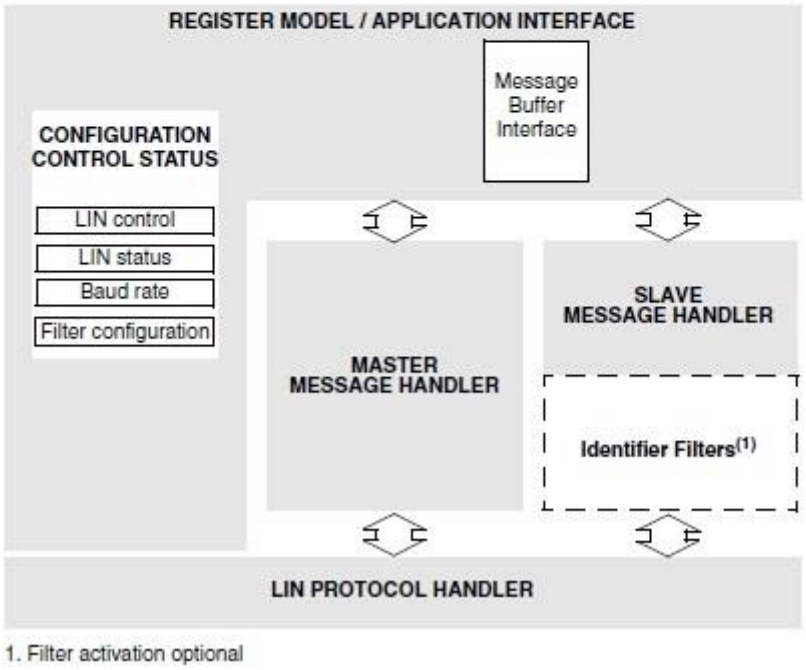

Figura 2.9 - Schema a blocchi della LINFlex

2.4.3 Modalità di funzionamento

La LINFlex ha tre principali modalità di funzionamento: Initialization, Normal e Sleep. Dopo un reset sull'hardaware, la LINFlex viene settata in modalità Sleep per ridurre in consumo di energia. Le istruzioni software permettono di impostare la LINFlex in modalità di Initialization o Sleep settando i bit relativi a INIT o SLEEP nel registro LINCR1.

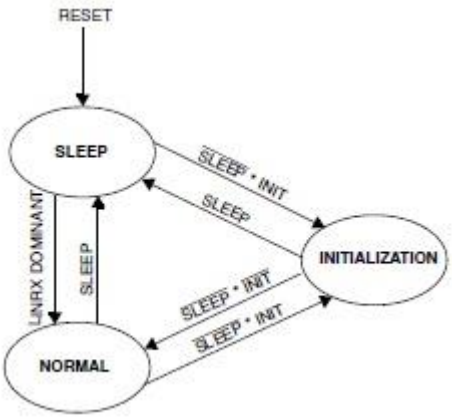

Figura 2.10 - Modalità di funzionamento della LINFlex

Modalità' "INITIALIZATION"

Il software può essere inizializzato mentre l'hardaware è nella modalità "Initialiazation". Per entrare in questa modalità il software imposta il bit INIT nel registro LINCR1, mentre per uscire lo pulisce. Durante questa modalità, tutti i messaggi da e verso la LIN bus vengo bloccati e l'uscita LINTX della LIN bus è nello stato "alto". Per inizializzare il controllore LINFlex, da software si possono scegliere le modalità LIN Master, LIN Slave or UART.

Modalità' "NORMAL"

Una volta che l'inizializzazione è completa, il software pulisce il bit INIT dal registro LINCR1 e lo pone in modalità normale.

Modalità "LOW POWER (Sleep)"

Per ridurre il consumo di energia, la LINFlex ha una modalità chiamata "Sleep mode". Per entrare in questa modalità, tramite software si imposta a il bit SLEEP nel registro LINCR1. Il clock della LINFlex viene bloccato e di conseguenza non aggiorna i bits di stato ma il software può accedere lo stesso ai registri della LINFlex.

2.4.4 Modalità UART

La LINFlex inizializzata in modalità UART si interfaccia con connessione via cavo con le periferiche esterne tramite i *pads* PC6 e PC7, rispettivamente LIN1_TX e LIN1_RX. Per la descrizione dettagliata si rimanda al paragrafo 4.3.1

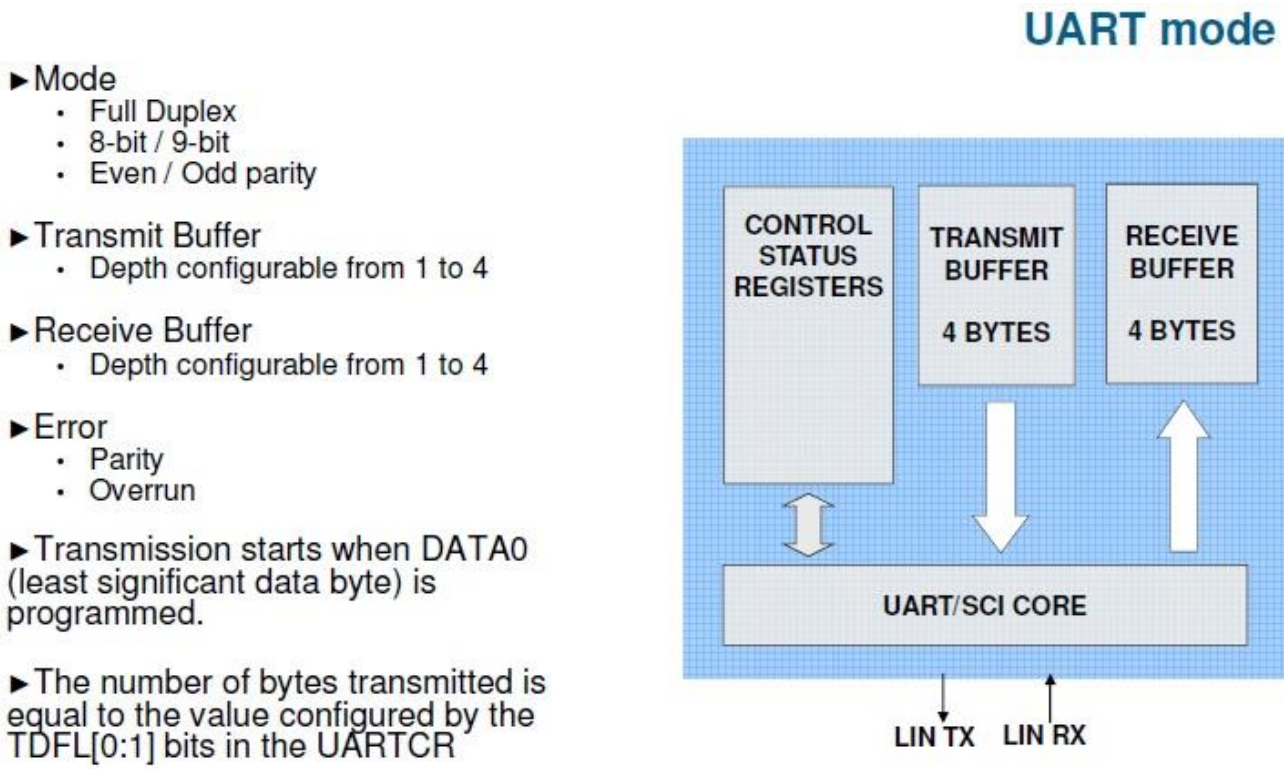

Figura 2.11 - Caratteristiche in modalità UART

Caratteristiche in modalità UART

- Comunicazione del tipo *Full duplex*
- Struttura dati a 8 o 9 bit, con bit di parità
- 4-byte di *buffer* in ricezione, 4-byte di *buffer* in trasmissione
- Contatore a 8-bit per la gestione del *timeout*

Struttura dati a 8-bit: l'ottavo bit può essere un dato o un bit di parità. "Even/Odd parity" viene abilitato dal bit di parità dal registro UARTCR. Se il bit "Even parity" viene selezionato, automaticamente il bit "odd parity" viene resettato.

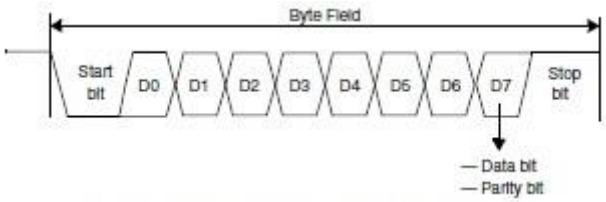

Figura 2.12 - Modalità UART con struttura dati a 8-bit

Struttura dati a 9-bit: Il nono bit può essere un dato o un bit di parità. "Even/Odd parity" viene abilitato dal bit di parità dal registro UARTCR. Se il bit "Even parity" viene selezionato, automaticamente il bit "odd parity" viene resettato.

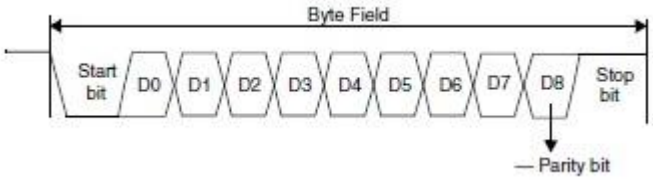

Figura 2.13 - Modalità UART con struttura dati a 9-bit

Buffer in modalità "UART"

Il *buffer data register* a 8-byte è diviso in due parti: BDRL[] per il buffer a 4 byte in trasmissione e BDRM[] per il buffer a 4 byte in ricezione:

Buffer data register	LIN mode		UART mode	
BDRL[0:31]	Transmit/Receive buffer	DATA0[0:7]	Transmit buffer	Tx0
		DATA1[0:7]		Tx1
		DATA2[0:7]		Tx2
		DATA3[0:7]		Tx3
BDRM[0:31]		DATA4[0:7]	Receive buffer	Rx0
		DATA5[0:7]		Rx1
		DATA6[0:7]		Rx2
		DATA7[0:7]		Rx3

Figura 2.14 - Tabella del buffer di comunicazione (message buffer)

UART transmitter

Per cominciare una trasmissione in modalità UART, si devono programmare a 1 i bits UART e TXEN (transmitter enable) nel registro UARTCR. La trasmissione comincia quando DATA0 (least significant byte) è programmato. Il numero di bits trasmessi sono uguali al valore configurato nel registro UARTCR[TDFL] (Transmitter Data Field Lenght).
Il buffer di trasmissione è a 4 bytes e quindi può essere stabilita una trasmissione a massimo 4 byte. Una volta che il numero programmato di bytes è stato trasmesso, il bit UARTSR[DTF] (Data Transmission Completed Flag) viene impostato. Se il bit UARTCR[TXEN] viene resettato durante una trasmissione, quella in corso viene completata, ma non possono essere svolte ulteriori trasmissioni.

UART receiver

L "UART receiver" è attivo non appena si esce dalla modalità "Initialization" e viene programmato a "1" il bit UARTCR[RXEN] (Receiver enable). Un buffer da 4-byte è dedicato per la ricezione dati. Il numero di bits ricevuti sono uguali al valore configurato nel registro UARTCR[RDFL] (Receiver Data Field Lenght). Una volta che il numero programmato di bytes è stato ricevuto, il bit UARTSR[DRF] (Data Reception Completed Flag) viene impostato. Se il bit UARTCR[RXEN] viene resettato durante una ricezione, quella in corso viene completata, ma non possono essere svolte ulteriori ricezioni.

Capitolo 3

Sensori ed Attuatori

3.1 SENSORI

Il microcontrollore si interfaccia con l'ambiente esterno tramite i sensori. Questi sono parte integrante di qualsiasi sistema di misurazione o di acquisizione e forniscono in uscita un segnale elettrico legato alla grandezza fisica in ingresso. Come vedremo di seguito, nel veicolo sono stati utilizzati una telecamera e due encoder.

3.1.1 Line scan camera

Per monitorare il tracciato e quindi la posizione del veicolo rispetto la linea nera di riferimento, si è utilizzato come sensore una *Line Scan Camera* che permette di interfacciare il microcontrollore con l'ambiente esterno essendo questa l'unica fonte di *input.*
Per il progetto è stata impiegata la TSL1401CL prodotta dalla TAOS® e fornitaci dalla *Freescale Semiconductor*™. E' composta da un *array* di fotodiodi disposti in una matrice 1x128, associati ad un circuito di amplificazione della carica e ad una funzione interna di *data-hold* che permette di attivare e disattivare simultaneamente l'integrazione per tutti i pixel. La matrice è costituita da 128 pixel, ciascuno dei quali ha una superficie fotosensibile di 3524.3 µm² e distanziati tra loro di 8 µm. Le operazioni sono semplificate dalla logica di controllo interna che richiede soltanto un clock e un serial input (SI).

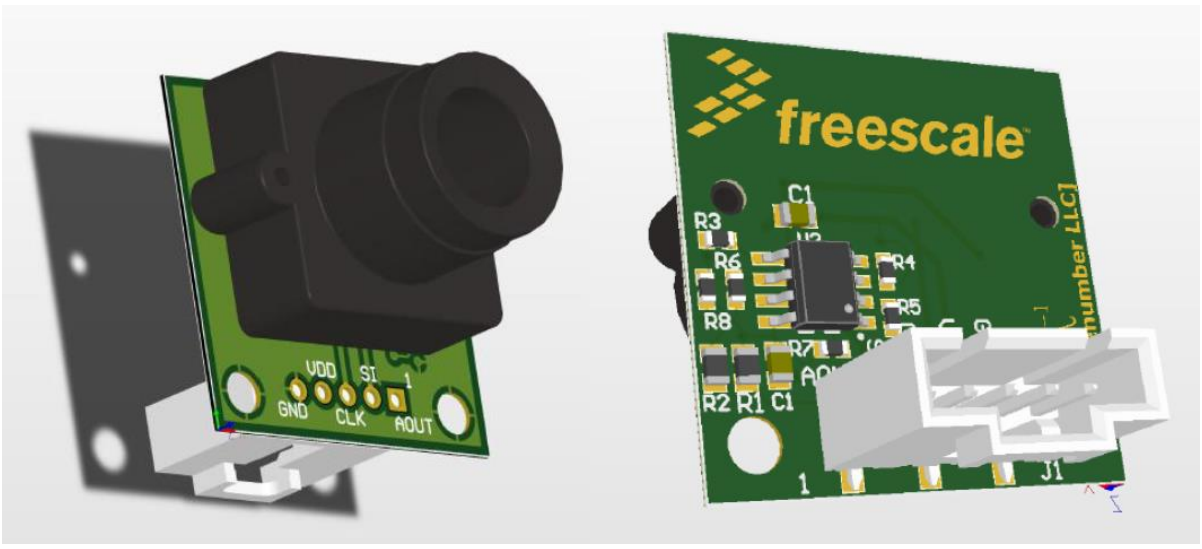

Figura 3.1 – *Sensore TSL1401CL*

Di seguito è riportato lo schema funzionale del sensore e la relativa piedinatura:

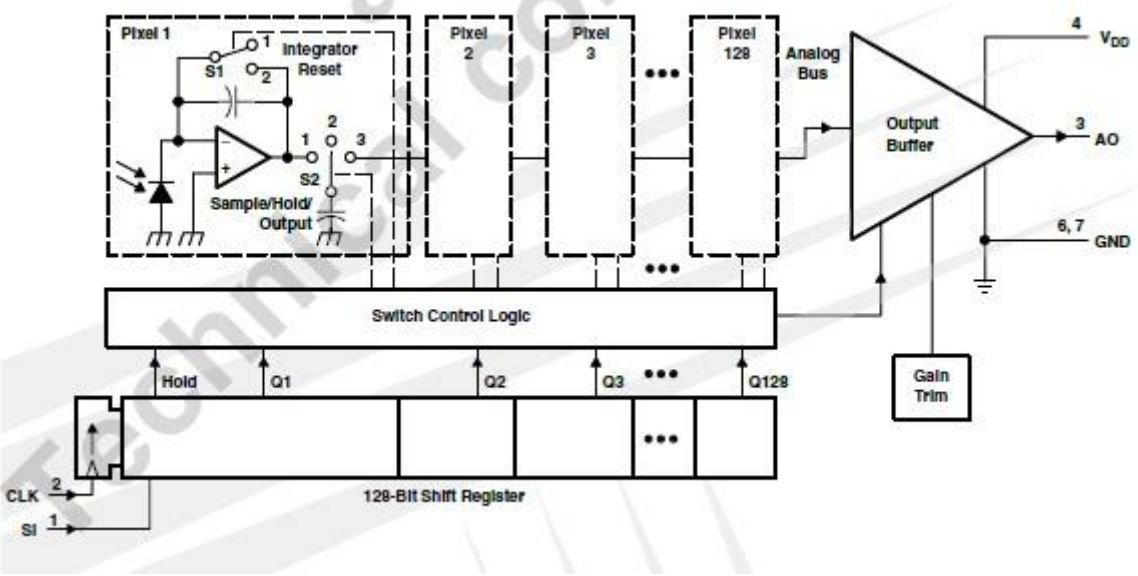

Figura 3.2 - Schema a blocchi del sensore

Tabella con la nomenclatura dei *pins della telecamera e relativa funzione:*

TERMINAL NAME	TERMINAL NO.	DESCRIPTION
AO	3	Analog output.
CLK	2	Clock. The clock controls charge transfer, pixel output, and reset.
GND	6, 7	Ground (substrate). All voltages are referenced to the substrate.
NC	5, 8	No internal connection.
SI	1	Serial input. SI defines the start of the data-out sequence.
V_{DD}	4	Supply voltage. Supply voltage for both analog and digital circuits.

Figura 3.3 - Funzionalità dei terminali

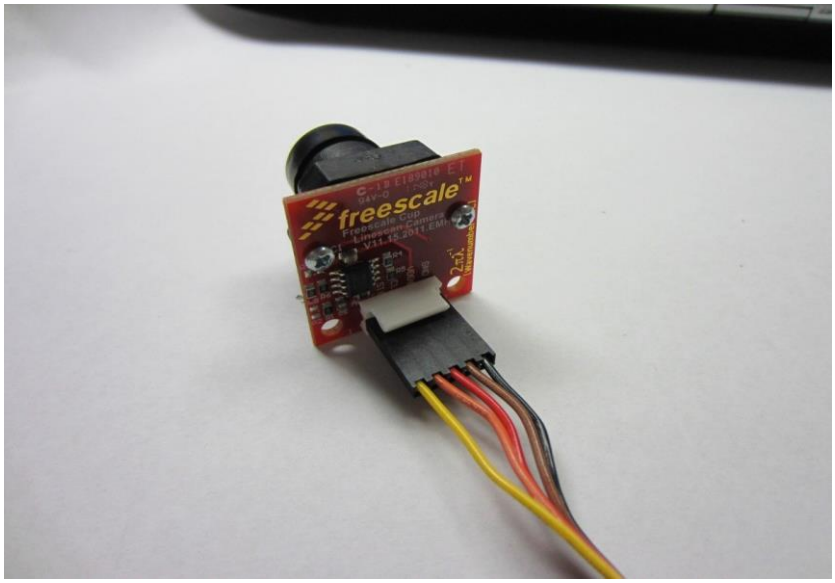

Figura 3.4 – Interfaccia a 5 pin per i segnali di I/O

La funzionalità principale di questo sensore è quella di convertire l'energia luminosa incidente sui fotodiodi in corrente elettrica e fatta passare attraverso un circuito di integrazione. Durante il periodo di integrazione, il condensatore di *sampling* mantiene la carica di ogni pixel proporzionale all'intensità luminosa e al tempo di integrazione. Successivamente l'uscita è mandata ad un convertitore analogico come si vede in figura 3.2.
Una della caratteristiche più importanti e utili della famiglia TAOS TSL14xx è data dalla possibilità di poter regolare il periodo di integrazione. Questo permette di modificare la quantità di carica accumulata dal condensatore durante la fase di campionamento e quindi evitare la possibilità di saturazione dell'uscita per diverse intensità di luce.
Il data sheet della telecamera fornisce anche i seguenti parametri:

- Interfaccia a 5 pin per i segnali di input e output (ground, power, SI, CLK, AO)
- Tempo di esposizione da 267μs a 68ms
- Stadio di amplificazione interno per il miglioramento del contrasto bianco/nero
- Risoluzione di 400 DPI
- Elevata linearità e uniformità
- Ampio Dynamic Range 4000:1 (72 dB)
- Tensione di uscita analogica riferita a GND
- Frequenza di lavoro fino a 8 MHz
- Alimentazione unica a 3 o 5 V
- Nessuna resistenza di carico richiesta per il funzionamento

3.1.2 Encoder

L' Encoder è un dispositivo elettromeccanico che effettua una conversione della posizione angolare del proprio asse in un segnale elettrico. Calettato opportunamente, è in grado di misurare movimenti rettilinei, velocità di rotazione, accelerazioni tramite appositi circuiti elettronici.
Per ottimizzare il funzionamento del veicolo e renderlo più affidabile è necessario conoscerne la velocità istantanea in ogni punto del tracciato. Per questo progetto sono stati utilizzati due encoder ottici, uno per ogni ruota posteriore. La scelta di calettarli sul retrotreno è dovuta principalmente alla semplicità di installazione e per le basse sollecitazioni meccaniche, anche se rispetto le ruote anteriori, sono soggette a slittamento e bloccaggio, rispettivamente in fase di accelerazione e frenata.
Il funzionamento degli encoder creati per questo utilizzo, si basa sul calcolo dell'intervallo di tempo trascorso tra un fronte di salita e l'altro grazie alla funzionalità IPM (Input Period Measurement) dei moduli eMIOS. Sono stati utilizzati:

- Un disco rigido di diametro 42 mm diviso in 20 spicchi regolari alternati in materiale plastico e rame:

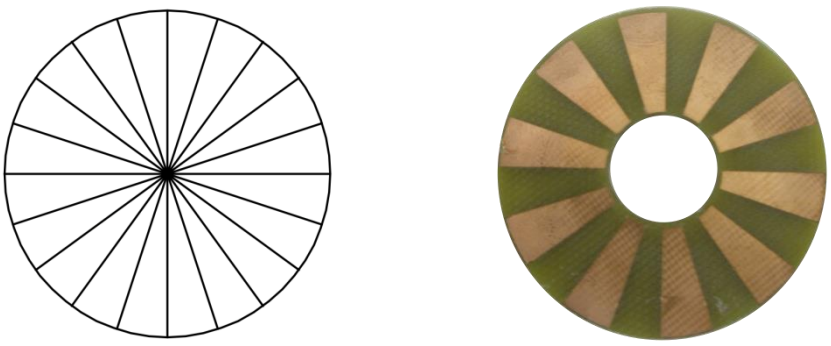

Figura 3.5 – Progetto e realizzazione del disco rigido

- Un sensore ottico riflettivo HOA1405:
  è un fotoaccoppiatore in cui vi è la combinazione, in uno stesso contenitore, di un LED e un fototransistor. Il fototransistor ha la proprietà di condurre corrente fra l'emettitore (E) e il collettore (C), proporzionale alla quantità di luce che incide sulla base. Dato che, sia l'emettitore sia il ricevitore dei raggi sono disposti sulla stessa superficie, è necessario che davanti ad entrambi sia presente una superficie riflettente, per fare in modo che il fototransistor possa ricevere i raggi che genera il LED. La superficie riflettente deve essere situata a pochi millimetri da quella su cui sono montati emettitore e ricevitore, per far si che i raggi riflessi abbiano sufficiente intensità.

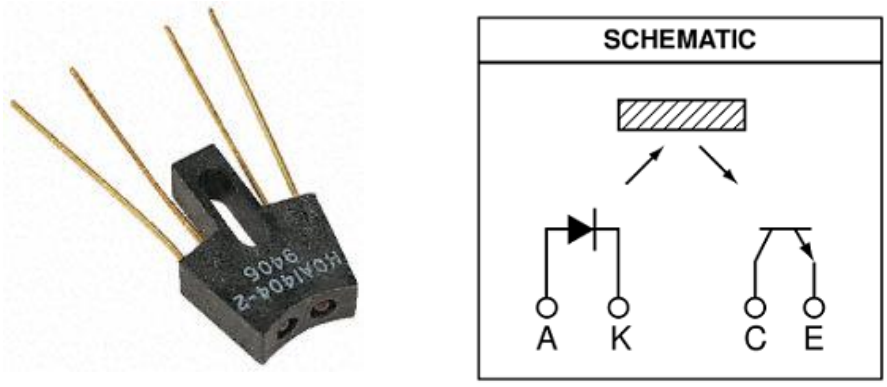

Figura 3.6 – Sensore HOA1405 e schematizzazione circuitale

- Un comparatore LM393:
  è costituito da due comparatori di tensione indipendenti progettati per operare con una singola alimentazione in un ampio intervallo di tensioni. Permette di lavorare con duplici alimentazioni con una differenza compresa tra i 2 V e i 36 V e Vcc sia di almeno 1,5 V maggiore rispetto alla tensione di ingresso di modo comune. L'assorbimento di corrente è indipendente dalla tensione di alimentazione.

Figura 3.7 – Comparatore LM393 e piedinatura integrato

Il sensore riflettivo viene fissato al telaio del veicolo e fatto puntare al disco rigido calettato alla ruota posteriore come in figura 3.8.

Figura 3.8 – Montaggio encoder

3.2 ATTUATORI

Gli attuatori sono dispositivi che convertono l'energia da una forma ad un'altra: principalmente un segnale di input elettrico in un movimento di una parte meccanica. Nel nostro caso sono il blocco motori e il servo motor.

3.2.1 MOTOR DRIVE BOARD

La scheda Motor Drive costituisce la parte di potenza del veicolo e inoltre monta l'hardware necessario per interfacciarsi con il servomotore e la telecamera. Nella scheda sono presenti due ponti ad H (o *chopper* a 4 quadranti) per l'alimentazione dei motori DC tramite tecnica PWM che permette di variare la tensione di armatura dei motori con a disposizione una sorgente di tensione continua fissa a 7.2 V. È anche presente un interruttore che controlla l'alimentazione proveniente dalla batteria.

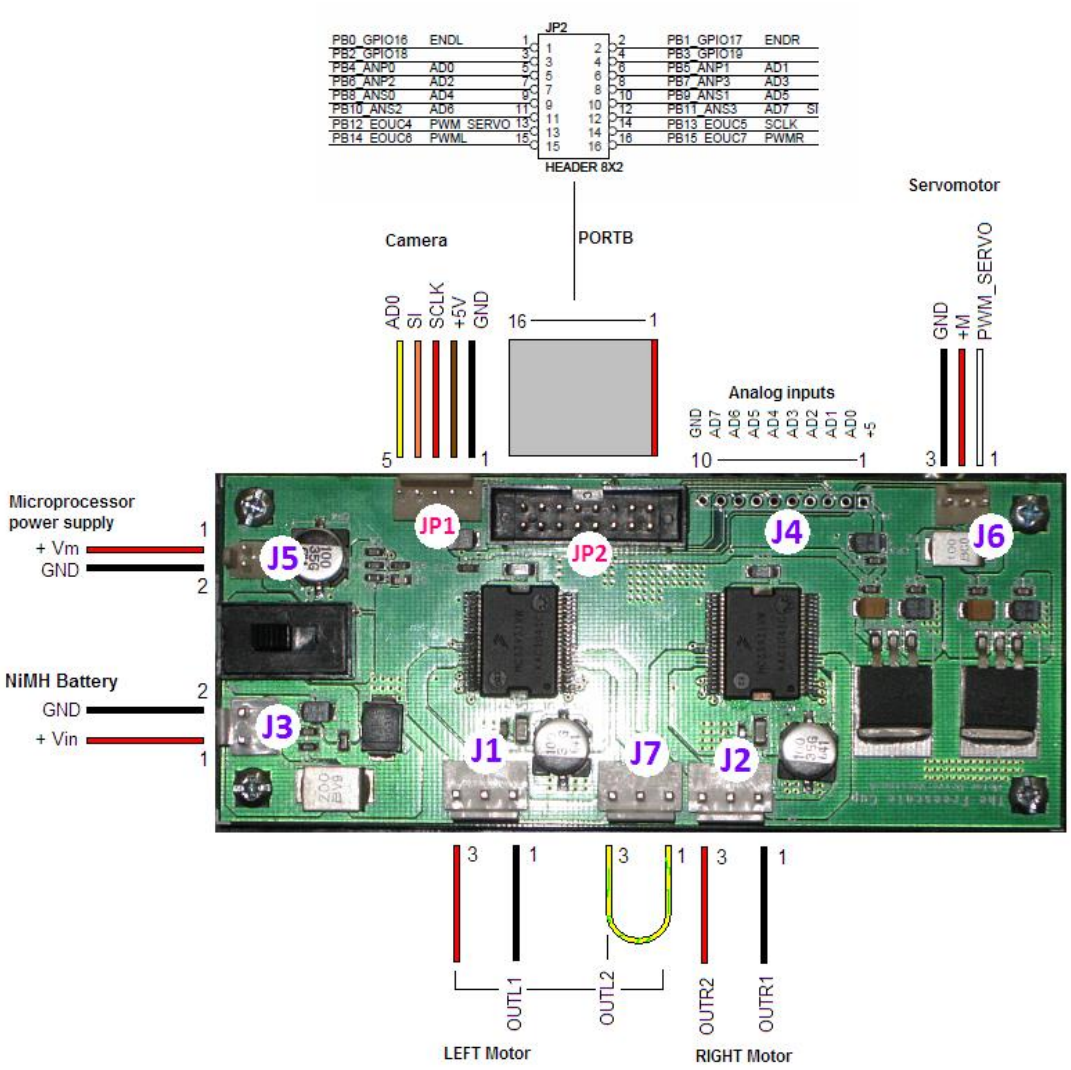

Figura 3.9 – Scheda Motor Drive con relativi segnali di I/O

I ponti ad H forniti nella scheda sono il modello MC33931: dei circuiti elettronici in grado di poter funzionare nei quattro quadranti del piano corrente-tensione sul carico. I 33931 dispongono anche di una limitazione della corrente in uscita e sono in grado di rilevare i cortocircuiti accidentali e le possibili condizioni di sovra-temperatura.
La connessione tra questa scheda ed il processore avviene tramite una piattina a 16 pin, che trasporta diversi segnali, tra cui quelli di comando per gli attuatori e quelli di lettura dei segnali provenienti dalla telecamera.

Figura 3.10 – Ponte ad H MC33931

3.2.1.1 Motori CC

Il motore elettrico rientra nella categoria degli attuatori ed è il componente che trasforma l'energia elettrica erogata dal convertitore statico in energia meccanica necessaria al moto delle parti meccaniche.
Per la trazione del veicolo, la Freescale ha fornito due motori in corrente continua del tipo Standard Motor: “rn 260 c” winding 18130 e successivamente collegati uno per ogni ruota posteriore in modo indipendente. La mancanza di un differenziale meccanico è stata sopperita implementandolo via software, necessario per un controllo adeguato del veicolo.

Figura 3.11 – Motore in CC utilizzato

I motori in corrente continua sono caratterizzati da un sistema meccanico di spazzole e collettore per trasferire corrente al rotore. Il principale vantaggio che si è riscontrato in questo tipo di motori consiste nelle rapide accelerazioni, mentre il principale difetto è dato dalla veloce e progressiva usura del sistema di alimentazione spazzole-collettore causando una forte diminuzione delle prestazioni.
Il catalogo dei motori riporta i seguenti dati:

Rated Voltage	V	4,5
No Load Speed	rpm	10000

No Load Current	m A	130
Max efficiency Current	m A	510
Max efficiency Speed	rpm	7950
Max efficiency Torque	gcm	18
Max output Current	m A	1070
Max output Speed	rpm	5000
Max output Torque	gcm	44
Stall current	m A	2000
Stall Torque	gcm	88

3.2.2 Servo comando

Per il posizionamento dello sterzo del veicolo vengo utilizzati particolari motori in continua o brushless, detti servomotori e montano al loro interno un sistema per l'acquisizione della posizione del rotore. Sono realizzati con l'obiettivo di produrre coppie elevate rispetto il proprio peso e volume. Al loro interno si trovano un motore in continua di piccole dimensioni ma di potenza discreta, una catena di riduzione del moto, un potenziometro dedicato al feedback di posizione e da un circuito per il controllo di posizione, inoltre nella maggior parte dei casi l'albero motore è vincolato a ruotare solo di un certo range di gradi, solitamente tra i 180 e i 270 gradi. La possibilità di posizionare il rotore in continuazione, rendono questi attuatori ideali nella robotica, nel modellismo e in molte altre applicazioni dov'è richiesta una buona precisione. Tramite opportuni collegamenti, possono essere interfacciati con un microcontrollore.

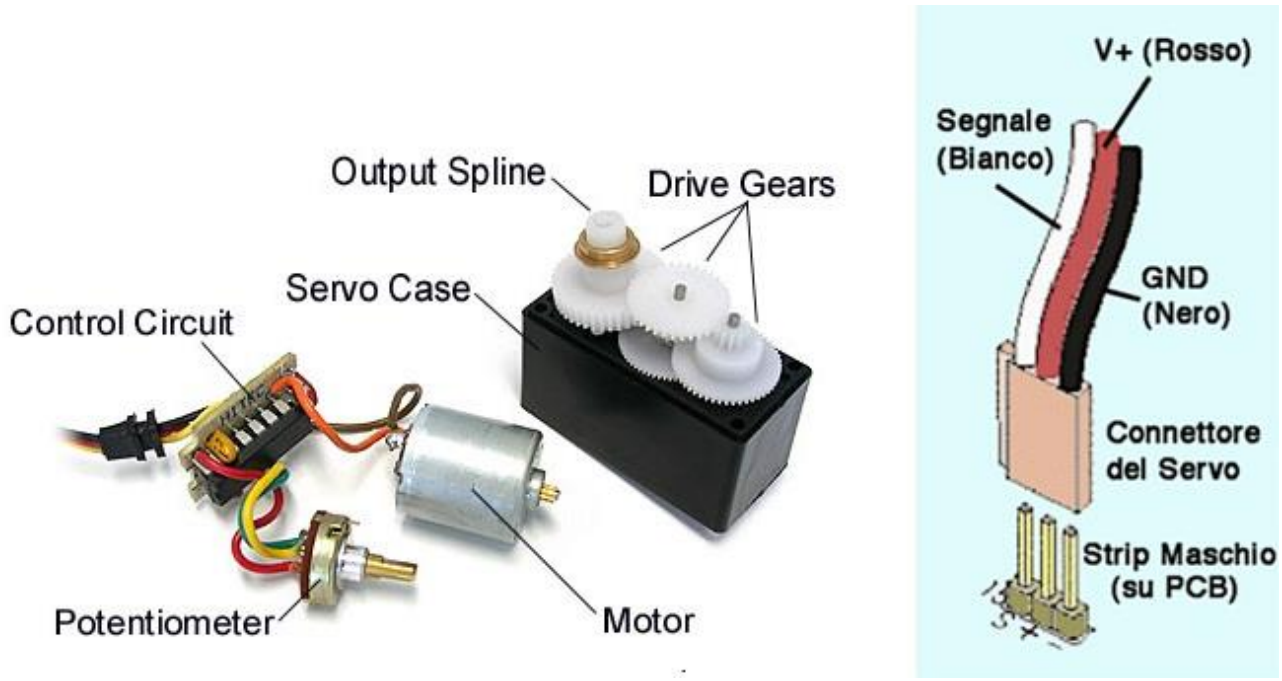

Figura 3.12 Struttura di un servomotore in continua e relativi collegamenti

I servomotori attuali sono di tipo proporzionale e possono quindi assumere tutte le configurazioni possibili all'interno del range di movimento. La potenza applicata al motore è proporzionale alla distanza che deve percorrere per posizionarsi correttamente. Così se l'albero necessita di spostarsi di un angolo molto ampio, esso si muoverà alla velocità massima, al contrario se necessita solo di una piccola regolazione il motore ruoterà a bassa velocità: questo tipo di controllo è detto proporzionale.
Generalmente con un impulso di durata pari a 1.5 ms il perno del servo si posiziona esattamente al centro del suo intervallo di rotazione. Da questo punto il perno può ruotare fino a -90 gradi

(senso antiorario) se l'impulso fornito ha una durata inferiore a 1.5 ms oppure fino +90 gradi (senso orario) se l'impulso fornito ha durata superiore a 1.5 ms.

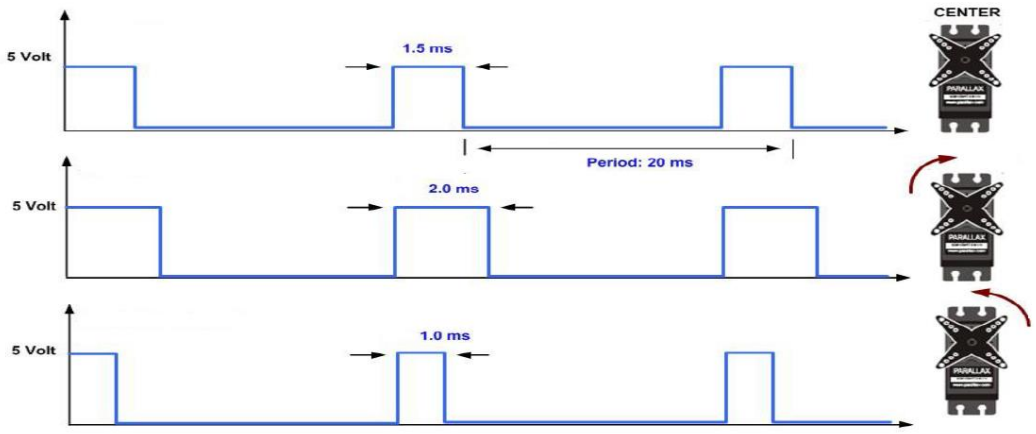

Figura 3.13 – Relazione tra durata dell'impulso e rotazione del servomotore

Nel data sheet del servomotore si trovano anche le seguenti specifiche:

Basic Information

Modulation:	Analog
Torque:	**4.8V:** 72.0 oz-in (5.18 kg-cm) **6.0V:** 90.0 oz-in (6.48 kg-cm)
Speed:	**4.8V:** 0.20 sec/60° **6.0V:** 0.16 sec/60°
Weight:	1.45 oz (41.0 g)
Dimensions:	Length: 1.57 in (39.9 mm) Width: 0.79 in (20.1 mm) Height: 1.50 in (38.1 mm)
Motor Type:	3-pole
Gear Type:	Plastic
Rotation/Support:	Single Bearing

Figura 3.14 – Caratteristiche generali del Servo Motor

Capitolo 4

Bluetooth
4.1 GENERALITA'

Il nome "Bluetooth" deriva dal re Danese del 10° secolo Harald Blatand o nella versione anglicizzata Harold Bluetooth. Quest'ultimo ha aiutato a riunire i belligeranti popoli di Norvegia, Svezia e Danimarca. Similmente la tecnologia Bluetooth è stata concepita come uno standard che permetta la connettività attraverso i prodotti industriali più diversi.

Il logo della tecnologia unisce infatti le rune nordiche ᚼ (Hagall) e ᛒ (Berkanan), analoghe alle moderne H e B, iniziali appunto di Harald Blåtand

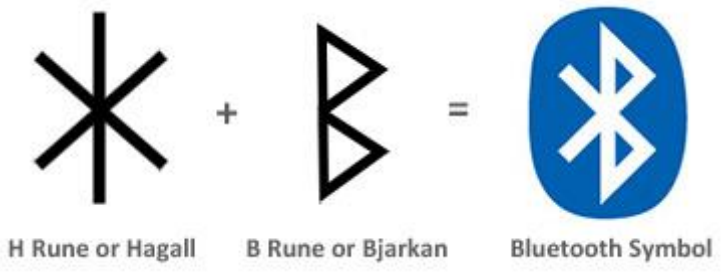

Figura 4.1 - Logo Bluetooth

Informazioni generali

Creato da Ericsson nel 1994, la tecnologia Bluetooth era stata originariamente concepita come un'alternativa wireless allo standard di trasmissione dati RS-232 via cavo. Quest'ultimo permette lo scambio di dati entro brevi distanze sfruttando le trasmissioni radio e operando in una banda libera compresa tra i 2.4 e i 2.485 GHz.

Interferenze

La tecnologia Bluetooth AFH (Adaptive Frequency Hopping) è stata progettata per ridurre le interferenze nello spettro dei 2.4 GHz. Quest'ultima funziona cercando i dispositivi in tale spettro, evitando le frequenze già utilizzate. Questo *Adaptative Hopping* divide la banda in 79 canali e continua a commutarla 1.600 volte/secondo ottenendo così un alto grado di immunità alle interferenze per un *data logging* più efficiente.

Range

Il Range è specifico per ogni applicazione e anche se un intervallo minimo è garantito dal dispositivo utilizzato, non ci sono limiti e ogni produttore può mettere a punto la loro realizzazione in base ai diversi utilizzi che essi permettono:
Il Range dipende dalla classe di trasmissione radio e dal tipo di impiego:

- Classe 3 - Range fino a 1 metro
- Classe 2 - Range fino a 10 metri (comunemente usato nei dispositivi mobili)
- Classe 1 - Range fino a 100 metri (comunemente usato nelle applicazioni industriali)

Power

La Classe 2 per la trasmissione radio è quella più utilizzata con un consumo di 2.5 mW. La tecnologia Bluetooth è appositamente progettata per avere un consumo molto basso di potenza e permette lo spegnimento del dispositivo quando è inattivo.
La tecnologia a basso consumo energetico, ottimizzata per i dispositivi che richiedono una lunga durata della batteria piuttosto che un'elevata velocità di scambio dati, può consumare tra 1/2 e 1/100 della potenza di un tipico dispositivo bluetooth.

Bluetooth vs wi-fi

Bluetooth e Wi-Fi hanno un impiego simile: stabilire connessioni di rete, collegamento con stampanti, trasferimento di files ecc… Il Wi-Fi è inteso come sostituto delle connessioni via cavo ad alta velocità per reti locali di lavoro (WLAN) ed è solitamente utilizzato in modalità di "access point" con una connessione asimmetrica "client - server", mentre il Bluetooth per equipaggiamenti portatili e le loro applicazioni (WPAN) con una connessione simmetrica. Quest'ultimo trova impiego in semplici applicazioni dove due dispositivi si connettono tra loro con una configurazione minima come premere un bottone, cuffie e controlli remoti. In fine l'access point per il BT permette connessioni ad-hoc in modo molto più semplicemente rispetto al Wi-Fi e con molte più funzionalità.

Conclusione

Data la totale inesperienza nell'utilizzo di dispositivi wireless, la principale caratteristica valutata è ricaduta sulla semplicità di interfaccia tra i dispositivi. Infatti, come vedremo in seguito nel dettaglio, il modulo Bluetooth simula un porta RS-232 per la trasmissione seriale di dati e questo risulta essere una notevole agevolazione per potersi allacciare con la seriale del microcontrollore (vedi paragrafo 4.3.1). In secondo piano, dopo un'attenta analisi, sono stati valutati l'alto grado di immunità e il basso costo. Questo ha fatto valutare la scelta su un dispositivo Bluetooth rispetto ad altri con funzionalità e caratteristiche equivalenti.

4.2 MODULO BLUETOOTH RN-42

4.2.1 Introduzione e descrizione generale

Il modulo Bluetooth RN-42 fornisce un affidabile metodo per creare un interfaccia di comunicazione seriale wireless tra due dispositivi come microcontrollori, PC, smarthpone o altri moduli. Questo permette di accoppiarsi con dispositivi che supportano il Bluetooth SPP (Serial Port Profile) per stabilire un interfaccia seriale.

Caratteristiche

- Modulo Bluetooth certificato 2.1/2.0/1.2/1.1
- Basso consumo
- Auto-discovery/pairing
- Auto-connect in modalità master
- Compatibile con microcontrollori 3.3 V e 5 V
- Indicatori LED per stato/connessione
- 9600 baud rate di default
- Jumper per selezione a 115K baud rate o Modificabile da software

Specifiche principali

-
- Alimentazione: 5.0 VDC or 3.3 VDC @ ~5 mA sleep; ~15 mA idle; ~20 mA transmit; ~50 mA max
- Interfaccia di comunicazione: 5 V / 3.3 V interfaccia di comunicazione seriale con RTS/CTS flow control, da 1200 bps a 921K bps
- Temperatura di funzionamento: -40 to +185 °F (-40 to +85 °C)
- Dimensioni: 2.33” x 1.16” x 0.45” in (59.18 x 29.46 x 11.43 mm)

Applicazioni

- Controllo remoto di robot via PC, smartphone o altri microcontrollori
- Sostituzione dei cavi (interfaccia seriale wireless
- Sistemi di misura e monitoraggio
- Controlli industriali e sensori

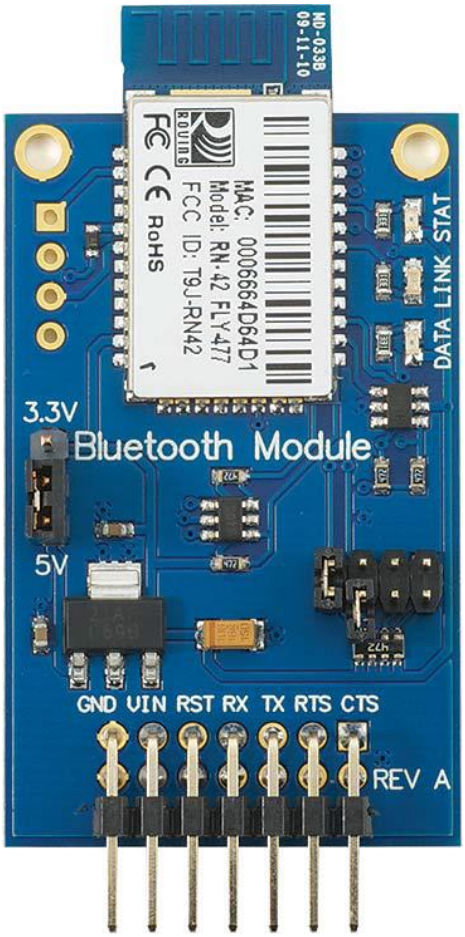

Figura 4.2 - Modulo Bluetooth Parallax RN-42

4.2.2 Schema elettrico e funzionale

Il dispositivo Bluetooth (U1) viene fornito di un modulo di supporto che lo rende "user friendly", dove sono presenti i sette pin di interfaccia con l'ambiente esterno (J1), un jumper per la scelta dell'alimentazione (J4) ed un altro per la scelta della modalità di funzionamento (J2).

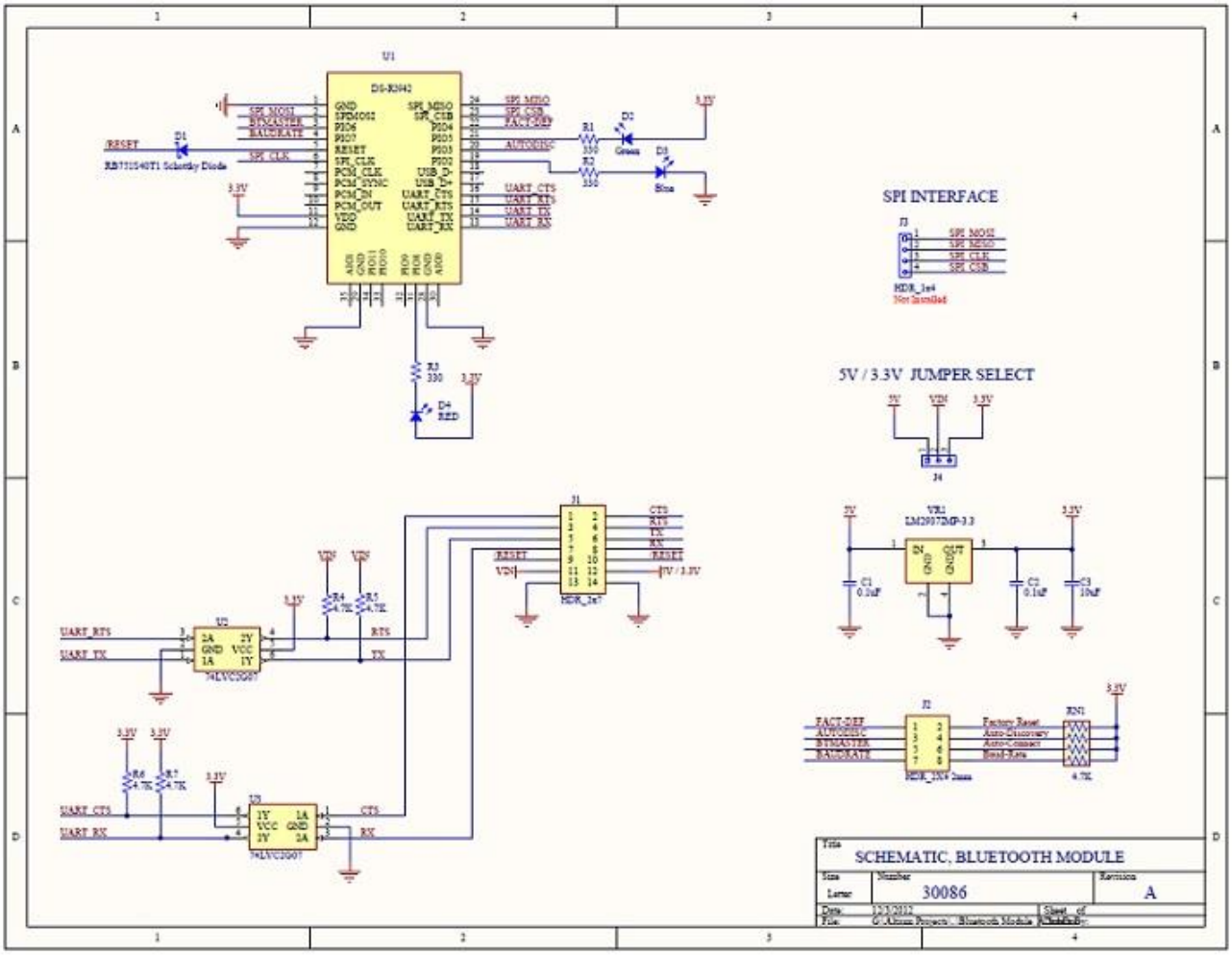

Figura 4.3 - Schema elettrico modulo RN-42

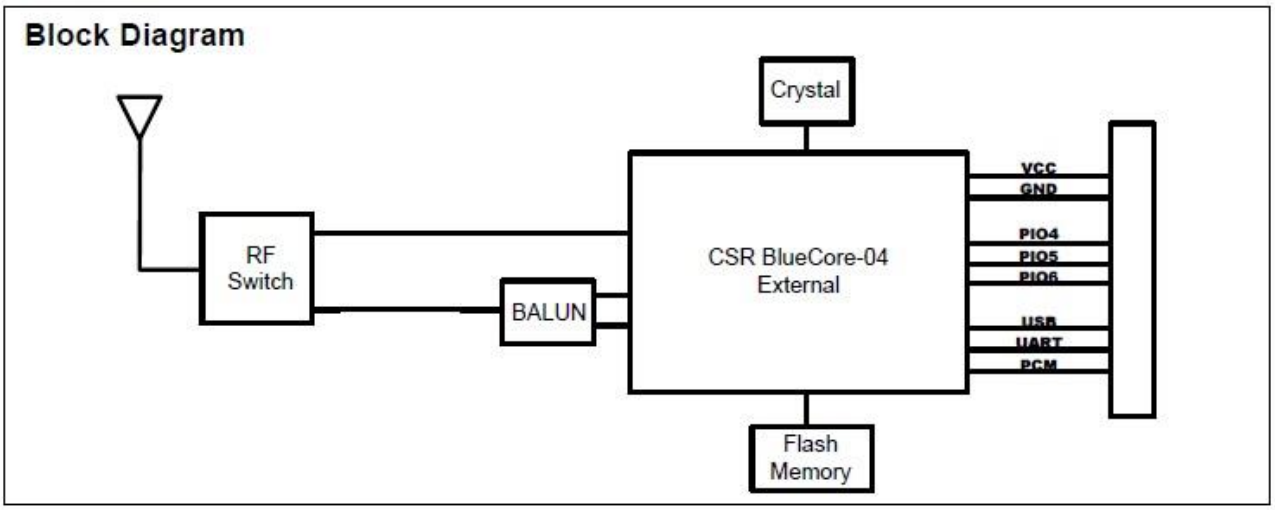

Figura 4.4 - Schema a blocchi modulo RN-42

4.2.3 Configurazione e layout dei pins

Si imposta la tensione di alimentazione del modulo a 3.3 V o 5 V in base a quella disponibile sul Microcontrollore, usando il jumper di selezione come indicato sopra e nel nostro caso sono i 5 V. Successivamente si connettono V_{in} e GND come da figura e i restanti RX, TX, RTS E CTS ai *pins* I/O relativi nel microcontrollore.

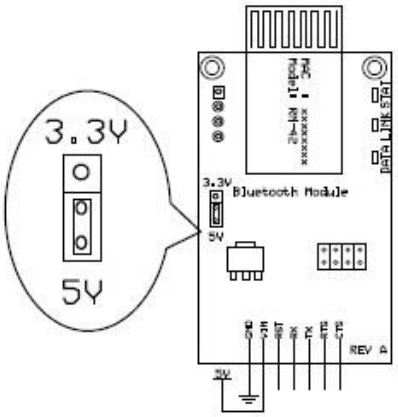

Figura 4.5 – Microcontrollore a 5V

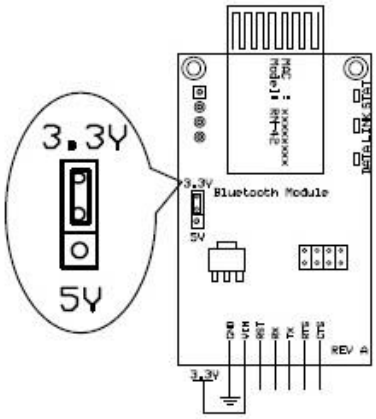

Figura 4.6 - Microcontrollore a 3.3V

Tabella con i valori nominali della tensione di alimentazione con le relative impostazioni dei *jumpers*:

Symbol	Quantity	Minimum	Typical	Maximum	Units
V_{IN}	Supply Voltage [1]	4.8	5V	5.2	V
V_{IN}	Supply Voltage [2]	3.2	3.3	3.4	V
I_{VIN}	Supply Current [1]	5	15	50	mA
—	Operating Range [3]	36	55	60	Feet

1. When supply voltage jumper is set to 5V
2. When supply voltage jumper is set to 3.3V
3. Approximate readings in office environment, throughput and range may vary depending on environmental conditions and RF interference

Figura 4.7 – Specifiche di alimentazione

Tabella con la nomenclatura dei *pins* e le relative funzionalità:

Pin	Name	Type	Function
1	GND	G	Digital Ground
2	VIN	P	5V or 3.3V power input (must be regulated)
3	RST	I	Resets the device when brought to GND (0V)
4	RX	I	Receive data input to RN-42 UART
5	TX	O	Transmit data output from RN-42 UART
6	RTS	O	Ready To Send output from RN-42 UART
7	CTS	I	Clear To Send input to RN-42 UART

Pin Type: P = Power, G = Ground, I = Input, O = Output

Figura 4.8 - Definizione dei pin e valori di funzionamento

4.2.4 Configurazione dei jumpers

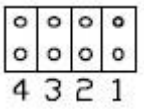

Figura 4.9 - Jumpers

Il modulo Bluetooth RN-42 ha Quattro possibili configurazioni come mostrato in figura. Due jumpers sono inclusi per ablitare le differenti caratteristiche, dove uno è già utilizzato mentre l'altro è solo appoggiato su un pin.

DEFAULT BAUD RATE (4)
OFF = 115K default (Può essere sovrascritto dalla configurazione di baud rate tramite software
ON = 9600 (Ignora la configurazione di baud rate da software. L'impostazione di default è ON)

AUTO MASTER (3)
ON = Il dispositivo è in configurazione di Master, funzione di "auto-connecting" ad un indirizzo remoto salvato. Prima si deve impostare l'indirizzo del dispositivo in modalità Slave tramite il comando SR.
L'impostazione di Default è OFF

AUTO DISCOVERY (2)
In modalità Slave, si imposta una classe particolare di dispositivi usati dal master per autoconnettersi. Se anche il jumper 3 è ON, il dispositivo fa una ricerca, si connette ed immagazzina l'indirizzo di un dispositivo remoto con jumper 2 impostato su ON. L'impostazione di Default è OFF

FACTORY RESET (1)
Si imposta questo jumper su ON, si alimenta il dispositivo e lo si stacca/riattacca per 3 volte per ritornare alle impostazionoi di fabbrica. L'impostazione di Default è OFF

Per le impostazione nel nostro progetto non sono stati utilizzati jumpers.

4.2.5 Accoppiamento e connessione

Il modulo Bluetooth comunica utilizzando una connessione seriale asincrona e supporta l'RTS/CTS flow control. Se non utilizzati o supportati, basta cortocircuitarli assieme.
L'RN-42 nella fase di *Pairing* con un altro dispositivo come PC o smartphone, la chiave di accesso è "1234" e viene visto con la sigla RN42-xxxx (dove xxxx sono le ultime 4 cifre dell'indirizzo MAC del dispositivo). Ad esempio in un PC con connessione Bluetooth, una volta assegnata la porta COM al modulo è possibile cominciare a spedire/ricevere dati in modo seriale.

4.3 INTERFACCIA µC/BLUETOOTH/PC

In questo paragrafo verrà descritto come il Bluetooth andrà ad interfacciare il PC (host) e il microcontrollore (controller) realizzando così una comunicazione Wireless tra le due periferiche. Successivamente saranno trattati separatamente i protocolli di comunicazione rispettivamente tra PC/BT e BT/uC in modo da comprendere nel dettaglio come i dispositivi comunicano tra loro.

4.3.1 Interfaccia µC/Bluetooth

L'UART (Universal Asynchronous Receiver/Transmitter) è un componente chiave nelle comunicazioni seriali di un microcontrollore. Prende i *byte* di dati e ne trasmette i singoli bit in modo sequenziale e una seconda UART in ricezione riassembla i bit in bytes completi.
La trasmissione asincrona consente ai dati di essere trasmessi senza che il mittente invii un segnale di clock al ricevitore. Mentre il mittente e il ricevitore devono essere impostati in anticipo e "bits speciali" sono aggiunti ad ogni parola e vengono utilizzati per sincronizzare i due dispositivi.
Quando una parola (*Data Word*) viene spedita all'UART per una trasmissione asincrona, viene aggiunto uno *Start bit* all'inizio di ogni parola che deve essere trasmessa. Il bit di start viene utilizzato per avvertire il ricevitore che sta per essere inviata una parola, e quindi forzare la sincronizzazione tra il clock del ricevitore e del trasmettitore. Successivamente i singoli bit della parola vengono inviati, a partire dal bit meno significativo (LSB).
Quando l'intera parola è stata inviata, il trasmettitore può aggiungere un bit di parità (*Parity bit*) che il trasmettitore genera e può essere utilizzato dal ricevitore per eseguire semplice controllo degli errori. Poi alla fine un bit di stop (*Stop bit*) viene inviato dal trasmettitore.
Quando tutti i bit della parola sono stati ricevuti, viene controllato il bit di parità (sia il ricevitore che il trasmettitore devono essere inizializzati per riceverlo/trasmetterlo) e alla fine il ricevitore verifica la ricezione del bit di stop. Se questa operazione di verifica va a buon fine, si può inviare il bit di start della nuova parola da trasmettere non appena il bit di stop della precedente è stato ricevuto.

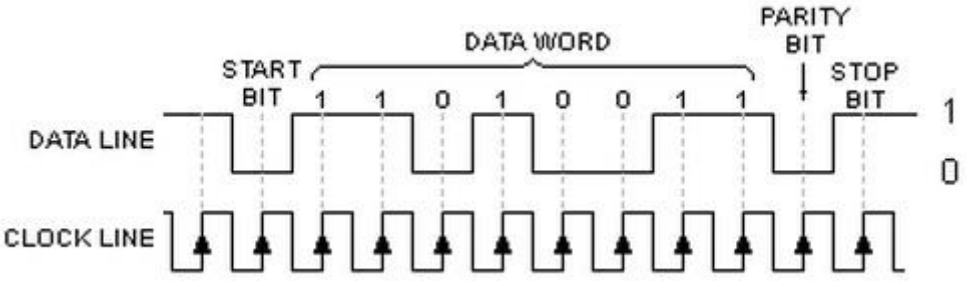

Figura 4.9 - Comunicazione Seriale Asincrona

Diagramma a blocchi descrittivo dell'interfaccia UART tra Bluetooth e Microcontrollore:

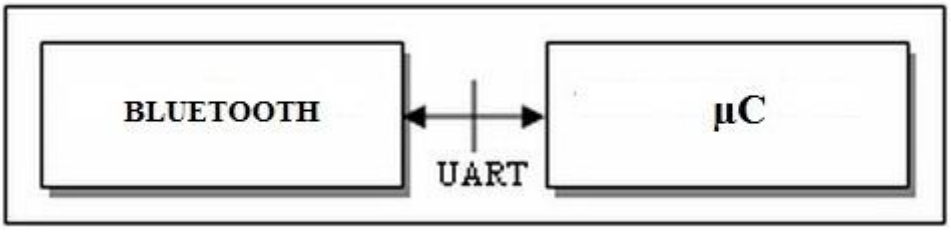

Figura 4.10 - Interfaccia µC/Bluetooth

4.3.2 - Interfaccia Bluetooth/PC

Il protocollo RFCOMM simula le impostazioni di una linea seriale via cavo di una porta del tipo RS-232 comunemente utilizzata per lo scambio di dati in modo seriale. L'RFCOMM si connette ai livelli più bassi dello stack del protocollo Bluetooth (Bluetooth Protocol Stack) attraverso il livello L2CAP.

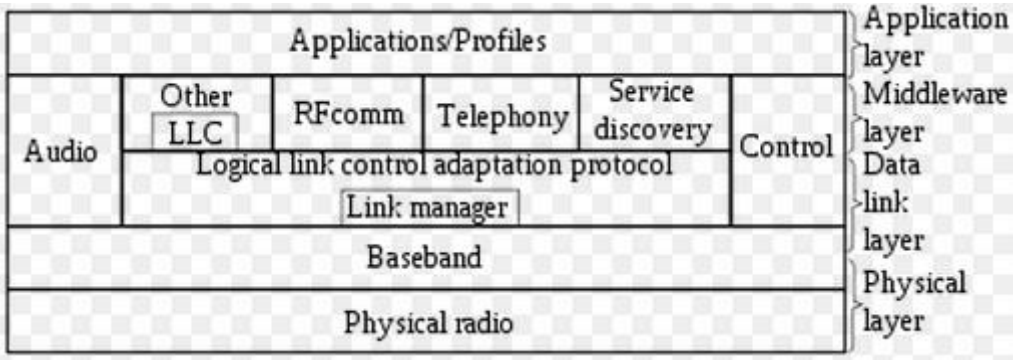

Figura 4.11- Bluetooth protocol stack

Il "*Bluetooth Protocol Stack*" è diviso in due parti: un "*controller stack*" contenente l'interfaccia radio e le temporizzazioni e un "*host stack*" che tratta i dati ad alto livello. Solitamente il "*controller stack*" è implementato su un dispositivo *low cost* contenente il Bluetooth e il microprocessore, mentre l'*host stack*" è generalmente implementato come parte di un sistema operativo, o di un pacchetto installato nel sistema operativo.

Controller Stack

Asynchronous Connection-Less (ACL):
Il tipico collegamento radio usato per trasmettere I "pacchetti" di dati usa un polling per arbitrare l'accesso. Può trasportare pacchetti di diversi tipi, i quali si possono distinguere in:

- Lunghezza (1,3 o 5 fasce temporali in base alla richiesta di carico)
- *Forward Error Correction* (Riduce la velocità dei dati in favore dell' affidabilità)
- Modulazione (L'"*Enhanced Data Rate Packets*" permette di triplicare la trasmissione dei pacchetti di dati usando una differente modulazione RF in base al carico)

Una connessione deve essere impostata in modo esplicito e accettata dai due dispositivi prima che i pacchetti possano essere trasferiti.

Host Stack

Logical link control and adaptation protocol (L2CAP):
L2CAP è usato all'interno del "Bluetooth protocol stack" e trasferisce i pacchetti all'interfaccia del "Host Controller (HCI)".
Le funzioni del protocollo L2CAP includono:

- Il *Multiplexing* dei dati tra i vari livelli
- La segmentazione e il riassemblaggio dei pacchetti
- Fornisce la gestione della trasmissione unidirezionale di dati ad un gruppo di altri dispositivi Bluetooth
- Qualità del servizio di gestione per protocolli di livello superiore

L2CAP viene usato per comunicare sul collegamento ACL del controller. La sua connessione viene stabilita dopo che il collegamento ACL è stato istituito.

Radio Frequency Communication (RFCOMM):

Il protocollo Bluetooth RFCOMM sono un insieme di protocolli di trasporto, costruiti al di sopra del protocollo L2CAP, fornendo un'emulazione di una porta seriale con standard RS-232.
Fornisce un semplice e affidabile flusso di dati per l'utente, simile al TCP e dovuto alla sua larga diffusione sono disponibili API (Application Programming Inteface) per la maggior parte dei sistemi operativi. Inoltre, le applicazioni che utilizzavano una porta seriale per comunicare possono essere rapidamente portati ad utilizzare una RFCOMM.

4.3.3 Utilizzo nel progetto

Ai fini dell'RFCOMM, una comunicazione completa prevede che due applicazioni siano in esecuzione su differenti dispositivi con un segmento di comunicazione tra loro. In questo caso il segmento di comunicazione è proprio il collegamento Bluetooth

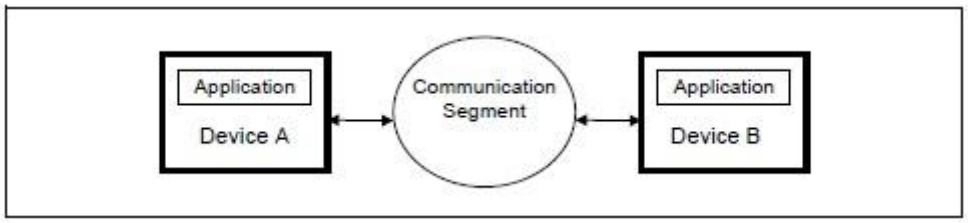

Figura 4.12 - RFCOMM communication segment

L'RFCOMM supporta anche la configurazione con un modulo Bluetooth che comunica wireless da un lato e un'interfaccia cablata dall'altra.

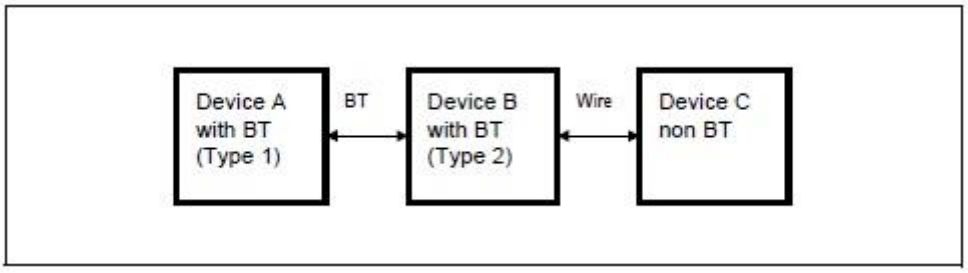

Figura 4.13 - RFCOMM usato con dispositivo COMM

Nel caso considerato i dispositivi A e B che comunicano con il protocollo Bluetooth RFCOMM sono rispettivamente il PC (con modulo BT integrato) e il dispositivo Bluetooth stesso. Quest'ultimo va ad interfacciarsi via cavo con il microcontrollore tramite l'UART che ne trasferisce i dati in modo seriale. Come vedremo nel dettaglio al punto 5.2.3, affinché i due protocolli possano comunicare in modo corretto, devono essere impostati allo stesso modo e in questo progetto sono stati utilizzati:

- Baud rate a 115K
- Trasmissione dati a 8 bits
- No bit di Parità
- 1 bit di Stop
- *Hardware flow control* abilitato
- Tx e Rx abilitati
- Buffer in ricezione/trasmissione di dimensione 1

Capitolo 5

Software

Il programma utilizzato per la trasmissione/ricezione e modifica dei dati si divide in due parti:

- **PC (Host):** costituito da un'interfaccia ad alto livello, dove dalla *shell* di Ubuntu mostrata a video, tramite appositi comandi, è possibile scegliere se visualizzare i valori provenienti da sensori o i riferimenti per gli attuatori oppure modificare i parametri digitandone di nuovi da tastiera.
- **Microcontrollore (Controller):** i comandi inviati da PC e ricevuti dal microcontrollore tramite Bluetooth, vengono elaborati dall'algoritmo in base alla canalizzazione decisa dall'utente e quindi subito rispediti alla stazione host se di sola lettura oppure la modifica delle variabili interne se di sola modifica.

5.1 UBUNTU

Motivazione della scelta di Ubuntu

La scelta di utilizzare il Sistema Operativo (S.O.) Ubuntu per comunicare con la periferica Bluetooth del PC, è dovuta principalmente al fatto che, a differenza del S.O. Windows, non è necessario avvalersi di altri *software* e quindi la conoscenza di altri linguaggi di programmazione oltre al *"C"*. Infatti tramite il suo *editor* di testo è possibile scrivere ed eseguire un programma per assegnare un *socket* alla periferica Bluetooth del PC e quindi creare una rete di comunicazione tra un programma *client* (lato *host* del PC) e un programma *server* (lato *controller* del microcontrollore) munito di dispositivo Bluetooth anch'esso.

5.1.1 Client.c

Il *main()* di questo programma è principalmente strutturato in un ciclo *while* infinito che permette all'utente la continua lettura e modifica in *real-time* delle variabili delle funzioni utilizzate dal *main()* caricato nel microcontrollore .
Con lo scopo di rendere più chiaro il funzionamento del programma, la funzione *main()* verrà divisa nelle seguenti parti:

- Dichiarazione delle variabili
- Assegnazione del *socket*
- Comandi per la sola ricezione
- Comandi per la sola modifica
- Generazione *Interrupt*

Dichiarazione delle variabili

In questa parte vengono dichiarate le variabili utilizzate nel programma. Fondamentale è la variabile *char dest[18]* che contiene l'*hostname* per poter "agganciarsi" al *server* e quindi al dispositivo Bluetooth connesso al microcontrollore.

```
struct sockaddr_rc addr = { 0 };
int status,num=4,i = 0;
char dest[18] = "00:06:66:61:EF:80"; //hostname del dispositivo BT
int offset = 0,ricezioni = 0,num_ricezioni = 0;
struct sigaction saio;
char
buffer[4],buffer_read,num_conv[2],prima_cifra[2],seconda_cifra[2],terza_cifra[2];
fpos_t pos;
char soglia[5];
int comando, virgola,lenght_soglia;
```

Assegnazione del socket

IL *Socket* è un metodo per comunicare tra un programma *client* e un programma *server* in una rete. E' definito come l'estremo di una comunicazione ed è creato ed usato con una serie di richieste di programmazione "*function calls*" , chiamate socket API (Application Programming Interface).

```
// allocate a socket
s = socket(AF_BLUETOOTH, SOCK_STREAM, BTPROTO_RFCOMM);
fcntl(s, F_SETOWN, getpid());
fcntl(s, F_SETFL, FASYNC);
```

Dal lato *client*: per eseguire una *connection request*, il *client* deve conoscere l'*hostname* nella quale è eseguito il *server* (lato microcontrollore) e il numero della porta nella quale è in ascolto. Anche il *client* ha bisogno di identificarsi per "agganciarsi" al numero della porta locale che userà durante la connessione. Solitamente è assegnata dal sistema.

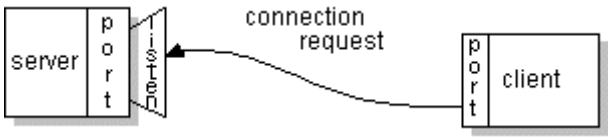

Figura 5.1 - Richiesta di connessione

Se tutto va a buon fine, una volta accettata la connessione, il *server* ottiene un nuovo socket per la comunicazione con la porta locale del *client.*

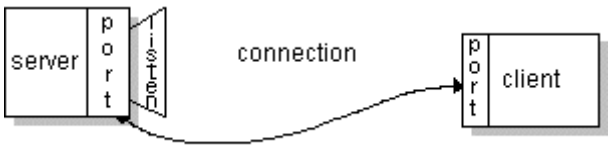

Figura 5.2 - Connessione con richiesta accettata

Comandi per la sola ricezione

Con comandi digitati da tastiera inferiori al 40, è possibile accedere alla routine di ricezione dei dati. Come già accennato precedentemente, la scelta di utilizzare un blocco per la sola ricezione nasce dall'esigenza di conoscere *real-time* come variano i parametri del software caricato nel microcontrollore durante il funzionamento del veicolo. Ad esempio, la principale funzione implementata riguarda la visualizzazione del valore di "centolinea" memorizzato nel relativo *array* e visto della telecamera bassa. Questo ci permetteva di capire cosa effettivamente vedeva il veicolo quando usciva di pista o in generale quando assumeva un "comportamento non programmato". Queste sono le selezioni visualizzabili dalla *shell* di Ubuntu:

```
printf("\n1. PER RICEVERE VELOCITA' MEDIA");
printf("\n2. PER RICEVERE VELOCITA' MEDIA RETTILINEO");
printf("\n3. PER RICEVERE VELOCITA' MEDIA CURVA");
printf("\n5. PER RICEVERE CENTRO LINEA");
printf("\n9. PER RICEVERE PERCENTUALE DIFFERENZIALE");
printf("\n12. PER RICEVERE SPEED RIGHT");
printf("\n13. PER RICEVERE SPEED LEFT");
printf("\n16. PER RICEVERE STEERING");
printf("\n17. PER RICEVERE ARRAY TELECAMERA BASSA");
```

Il ciclo *while* infinito si ferma all'istruzione *scanf()* in attesa della scrittura da tastiera del comando fra quelli disponibili elencati, e successivamente sarà possibile decidere il numero delle ricezioni da ottenere. Una volta riempito il buffer con il comando ed il numero di ricezioni, il primo ciclo *do-while*, tramite il comando *write()*, lo spedisce alla routine di ricezione in esecuzione nel veicolo. Questa ne è la parte di codice:

```
printf("\nCOMANDO: ");
scanf("%d",&comando);
printf("COMANDO RICEVUTO: %d\n",comando);
/*comandi di ricezione*/
if(comando > 0 && comando < 40 )
{
	/*buffer con comando corrispondente nel veicolo*/
	buffer[0] = (char)(comando);
	/*ricezione telecamere*/
	if(comando == 17 || comando == 18)
	{
		num_ricezioni = 128;
	}
	/*ricezione delle altre variabili*/
	else
	{
		printf("\nQUANTE RICEZIONI?");
		scanf("%d",&num_ricezioni);
	}
	buffer[1] = (char)num_ricezioni;/*buffer da spedire*/
	buffer[2] = '\0';
	buffer[3] = '\0';
	printf("\nINVIO COMANDO--->");
	do
	{
		num = num - write(s,buffer,4);/*num = 0  quando tutti i 4 byte*/
	}while(num!=0);                      /*sono stati spediti*/
	comando = comando ^ comando;
	num = 4;
	printf("Invio comando riuscito\n");
```

```
	do
	{
		if(wait_flag == FALSE) /*routine svolta con interrupt*/
		{                      /*(in fondo al programma)*/

			do{
				bytes_read = bytes_read - read(s,&buffer_read,1);
				/*num = 0 quando tutti i 6 byte sono stati letti*/
				read_buff[i] = buffer_read;
				/*salva il contenuto ricevuto nell'array*/
				i = i + 1;
			  }while(bytes_read!=0);
			  i = 0;
			  bytes_read = 6;
			  wait_flag = TRUE;
			  printf("RICEVUTO: [%s]\n",read_buff);
			  ricezioni = ricezioni + 1;
		}
	}while(num_ricezioni != ricezioni);
	ricezioni = 0;
}
```

Comandi per la sola modifica

Con comandi superiori al 40 è possibile entrare nella routine di modifica dei dati. Come precedentemente detto, la routine per la sola modifica dei parametri risulta essere fondamentale soprattutto nella fase di *tuning*. Durante il funzionamento nel tracciato è possibile affinarne il comportamento, ad esempio variandone la velocità di curva e rettilineo o più nel dettaglio le costanti K del PID dello sterzo per migliorarne il comportamento oscillatorio in uscita di curva. Per avere conferma che il parametro modificato appena inviato sia corretto, il secondo ciclo *do-while* (vedi allegato con il programma completo *Client.c*) ne permette la lettura dopo essere stato spedito dal *printserialsingned()*, che è una funzione implementata nell'algoritmo del microcontrollore per la trasmissione di caratteri (vedremo in successivamente). Queste sono le selezioni visualizzabili dalla *shell* di Ubuntu:

```
printf("\n40. PER MODIFICARE VELOCITA' RETTILINEO");
printf("\n41. PER MODIFICARE VELOCITA' CURVA");
printf("\n42. PER MODIFICARE KP_STEERING");
printf("\n43. PER MODIFICARE KD_STEERING");
printf("\n44. PER MODIFICARE KI_STEERING");
printf("\n45. PER MODIFICARE KP_SPEED");
printf("\n46. PER MODIFICARE KD_SPEED");
printf("\n47. PER MODIFICARE KI_SPEED\n");
```

Per poter digitare da tastiera sia numeri interi che con la virgola, si è deciso di suddividere in fasce la routine di modifica:

- 41-80 cifre del tipo 0.123
- 81-120 cifre del tipo 1.23
- 121-160 cifre del tipo 12.3
- 161-200 cifre del tipo 123.
- 201-240 cifre senza virgola

L'operazione di riconoscimento della posizione della virgola non è visualizzabile dalla *shell* e viene svolta autonomamente dal software. Questo facilita sia la scrittura de parte dell'utente che il riconoscimento della cifra digitata da parte del programma dal lato veicolo (vedi allegato).

Il concetto di funzionamento è lo stesso della parte per la sola ricezione con in più l'algoritmo per il riconoscimento della posizione della virgola e il relativo riempimento del buffer.

Generazione interrupt

Quando il *printserialsingned()* invia il valore della variabile scelta dall'utente, viene fatta una richiesta di *Input* al *socket* associato al Bluetooth e quindi viene generato un *interrupt*. Questa parte di codice pone *wait_flag = FALSE* e quindi permette di entrare nel ciclo *do-while* per lettura del *buffer* dati associato al *printserialsingned()* del lato veicolo. Questa è la parte di codice associata alla gestione della richiesta di *interrupt :*

```
void signal_handler_IO (int status)/*genera un interrupt quando viene fatta una*/
{                                  /*richiesta di I/O al socket*/
      wait_flag = FALSE;
      usleep(10000);
}
```

5.2 CODEWARRIOR

La programmazione avviene mediante il software *CodeWarrior* attualmente sviluppato da *Freescale Semiconductor™*. Questo software appartiene alla categoria degli IDE (Integrated Development Enviroment), contiene un compilatore per il linguaggio C e fornisce alcuni strumenti per il *debugging*.Per una maggiore chiarezza nella gestione delle funzioni generate si è deciso di dividere il programma in molteplici *source*, quali:

- IntcInterrupts.c
- Ivor_branch_table.c
- Uart.c
- Main.c

Nella programmazione, la possibilità di implementare un programma su diversi livelli permette di semplificare le operazioni di progettazione del software, di aumentare il livello di astrazione e di scomporre il problema in più parti.
Con questo criterio per utilizzare variabili dichiarate in un altro *file* sorgente come variabili globali, si è costretti a ridichiararle nel *source file* in cui si vogliono usare, oltre che con il proprio tipo, anche con la classe di memoria di tipo *extern* in modo da far capire al compilatore che si sta utilizzando la medesima variabile dichiarata in un altro *file*.

5.2.1 IntcInterrupts.c

Nella programmazione, un *Interrupt* è un segnale emesso via hardware o software che richiede immediata attenzione e segnala al processore una richiesta di interruzione del programma in esecuzione. Il processore risponde sospendendo le attuali attività, salvando il suo stato, ed esegue una funzione chiamata "Interrupt handler" o ISR (Interrupt Service Request). Al termine, il processore riprende la normale esecuzione del programma, a partire dal punto salvato in precedenza.
La funzione *void INTC_InstallINTCInterruptHandler(handlerFn,vectorNum,psrPriority)* è utilizzata per installare un *Interrupt handler* all'interno di un programma. Il *source* IntcInterrupts.c è completamente precompilato e contiene un'implementazione per gestire una generica richiesta di Interrupt per i microcontrollori della famiglia MPC55xx. Di seguito viene riportata la parte di codice che principalmente è servita per il progetto:

```
#include "MPC5604B.h"          /* MCU platform development header */
#include "IntcInterrupts.h"    /* Implement functions from this file */

void INTC_InstallINTCInterruptHandler(INTCInterruptFn handlerFn, unsigned short
vectorNum, unsigned char psrPriority)
{
    /* Set the function pointer in the ISR Handler table */
    INTCInterruptsHandlerTable[vectorNum] = handlerFn;
    /* Set the PSR Priority */
    INTC.PSR[vectorNum].B.PRI = psrPriority;
}
```

L'argomento *handlerFn* è il nome del programma che si vuole eseguire quando un ISR (Interrupt Service Request) viene richiesto da una periferica.

Il *vectorNum* è il numero dell' *Interrupt vector* associato all'ISR per una certa periferica ed è ricavabile dalla relativa tabella. Il microcontrollore utilizzato permette fino ad un massimo di 210 IRQ (Interrupt Request) della quale alcune sono riservate e quindi non usufruibili. Di seguito ne è riportata la parte di interesse per il progetto: .

IRQ #	Offset	Size (bytes)	Interrupt	Module
99	0x098C	4	LINFlex_RXI	LINFlex_1
100	0x0990	4	LINFlex_TXI	LINFlex_1
101	0x0994	4	LINFlex_ERR	LINFlex_1

Figura 5.3 - Interrupt Vector Table

Il *psrPriority* permette di settare il livello di priorità associato ad ogni ISR. Le richieste di Interrupt avvengono regolarmente se effettuate una alla volta, ma nel caso di richieste multiple non è possibile soddisfarle tutte contemporaneamente, ma è necessario assegnare una priorità in modo tale da decidere quale svolgere prima o interromperne una con priorità più bassa. Sono presenti 16 livelli configurabili per ogni *interrupt*: al livello 0 corrisponde la minima priorità, al livello 15 la massima. Inoltre se la priorità di un *interrupt* in esecuzione è minore di quella di uno appena chiamato, quello in esecuzione viene bloccato per far iniziare quello con priorità maggiore. Questo è impostabile dal campo PRI del registro INTC_PSR

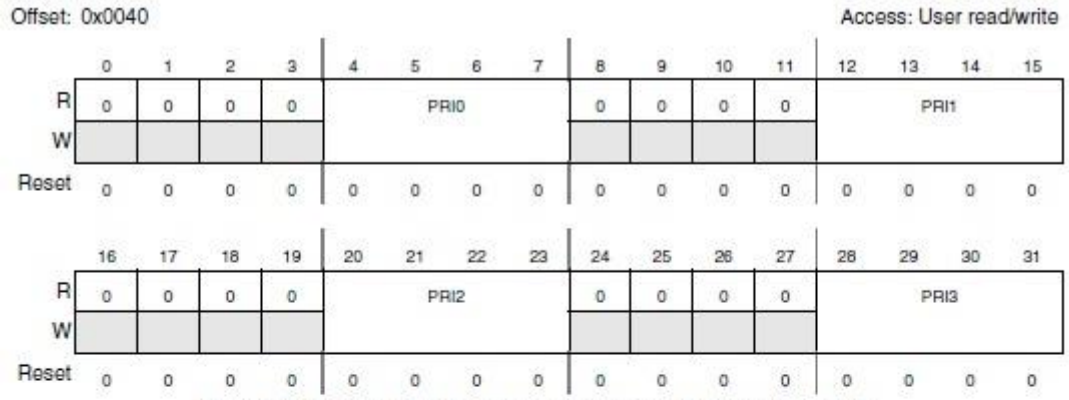

Figure 16-9. INTC Priority Select Register 0–3 (INTC_PSR[0:3])

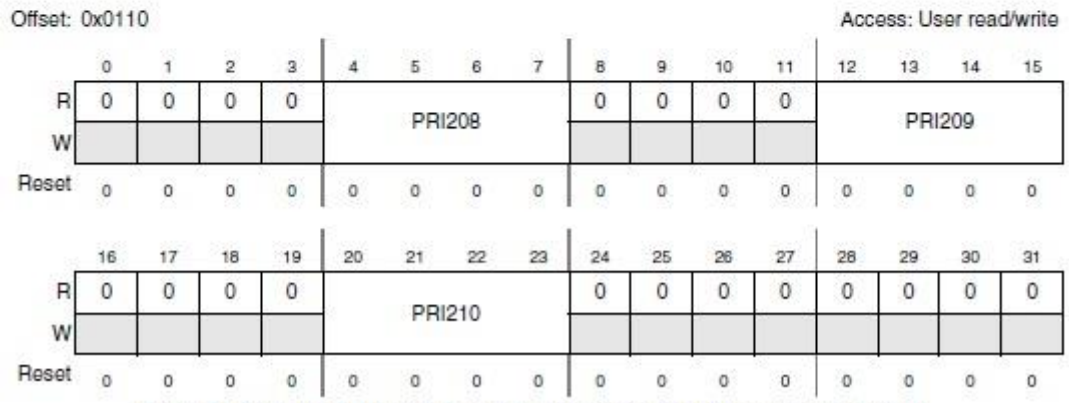

Figure 16-10. INTC Priority Select Register 208-210 (INTC_PSR[208:210])

Figura 5.4 - INTC Priority Select Register

Descrizione del *field* PRI del registro INTC_PSR, per assegnare il livello di priorità per ogni ISR:

Field	Description
PRI	Priority PRI is the priority of the currently executing ISR according to the field values defined in Table 16-5.

Figura 5.5 – PRI field

Tabella dei valori di priorità (PRI) impostabili nel registro INTC_PSR e relativo livello di priorità:

PRI	Meaning
1111	Priority 15—highest priority
1110	Priority 14
1101	Priority 13
1100	Priority 12
1011	Priority 11
1010	Priority 10
1001	Priority 9
1000	Priority 8
0111	Priority 7
0110	Priority 6
0101	Priority 5
0100	Priority 4
0011	Priority 3
0010	Priority 2
0001	Priority 1
0000	Priority 0—lowest priority

Figura 5.6 - Tabella dei valori di PRI (priorità)

5.2.2 IVOR_Branch_Table.c

La tabella IVOR (Interrupt Vector Offset Register) è un'area di memoria divisa in *interrupt vector* e ognuno di questi ha un indirizzo di memoria fissa ed è associato ad un determinato ISR (ad esempio il rilevamento del limite su un buffer di ingresso digitale o il time-out di un timer). All'indirizzo dell'*interrupt vector*, la memoria contiene l'indirizzo della funzione dedicata all'esecuzione dell'ISR, per questo il programmatore deve conoscerne esattamente la posizione. Quando un ISR viene attivato durante l'esecuzione del programma principale (*main*.c), l'indirizzo dell'istruzione successiva deve essere salvata in modo tale da poter tornare indietro dopo l'elaborazione dell'Interrupt. In pratica, prima di fermare l'esecuzione del *main.c* e lanciare il programma di interrupt, il contenuto del PC (Program Counter) viene salvato e sarà aggiornato alla fine del programma di *interrupt*.
Anche il source *ivor_branch_table.c* è completamente precompilato e contiene un implementazione che permette all'IVOR4 di chiamare il gestore dell'Interrupt. Di seguito è riportata una parte di codice:

```
/* IVOR4 will call this handler */
extern void INTC_INTCInterruptHandler(void);
```

5.2.3 Uart.c

Nel source *Uart.c* sono presenti le funzioni che permettono di realizzare una trasmissione dati cablata tra l'UART del microcontrollore ed una periferica esterna. Come già esposto nei capitoli precedenti, la periferica LINFlex inizializzata in modalità UART realizza un data logging seriale, prende i byte di dati e ne trasmette i singoli bit in modo sequenziale via cavo. Infatti il Bluetooth con i pins RX e TX si interfaccia fisicamente con il microcontrollore rispettivamente tramite i *pins* LIN1TX e LIN1RX (N.B.: Il pin TX del Bluetooth viene collegato al pad RX del micro e viceversa). Affinché i due dispositivi possano comunicare in modo corretto, devono essere impostati allo stesso modo :

- Baud rate a 115K
- Trasmissione dati a 8 bits
- No bit di Parità
- 1 bit di Stop
- *Hardware flow control* abilitato
- Tx e Rx abilitati
- Buffer in ricezione/trasmissione di dimensione 1

Di seguito, attraverso le relative funzioni, verranno elencati e descritti i registri che permettono di configurare l'UART con le caratteristiche appena elencate.

```
#include "MPC5604B.h"
#include "IntcInterrupts.h"
#include "uart.h"
#include "delay.h"

uint8_t read_uart = 0;
```

La funzione *void init_LinFLEX_1_UART (void)* tramite gli appositi registri, permette di inizializzare la LINFlex e di settarne le diverse impostazioni. Di seguito sarà trattata nel dettaglio la relativa programmazione:

```
void init_LinFLEX_1_UART (void)
{
	/* enter INIT mode */
	LINFLEX_1.LINCR1.R = 0x0085;	/* SLEEP=0, INIT=1 */
	/* wait for the INIT mode */
	while (0x1000 != (LINFLEX_1.LINSR.R & 0xF000)) {}
	/* configure pads */
	SIU.PCR[38].R = 0x0604;	/* Configure pad PC6 for AF1 func: LIN1TX */
	SIU.PCR[39].R = 0x0100;	/* Configure pad PC7 for LIN1RX */
	/* configure for UART mode */
	LINFLEX_1.UARTCR.R = 0x0001; /*set the UART bit first to be able to write*/
	                             /*the other bits*/
	LINFLEX_1.UARTCR.R = 0x0C33; /*8bit data,no parity,Tx and Rx enabled*/
	                             /*UART mode*/
	                             /* Transmit buffer size = 1 (TDFL = 0 */
	                             /* Receive buffer size = 1 (RDFL = 0) */
	/* configure baud rate 115200 */
	/* assuming 64 MHz peripheral set 1 clock */
	LINFLEX_1.LINFBRR.R = 12;
	LINFLEX_1.LINIBRR.R = 34;
	/*set the DRIE bit to be able to trigger an interrupt*/
	LINFLEX_1.LINIER.R = 0x0004;
	/* enter NORMAL mode */
	LINFLEX_1.LINCR1.R = 0x0080; /* INIT=0 */
}
```

Tramite il registro LINCR1 e settando il *field* INIT = 1, si fa entrare la LINFlex_1 in *init mode*. Una volta entrati in questa modalità e tramite appositi registri, è possibile impostare i parametri (baud rate, parità ecc…) per inizializzare una comunicazione TX/RX. Resettandolo, il micro entra in *Normal mode.*

Offset: 0x0000 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	CCD	CFD	LASE	AWUM	MBL				BF	SFTM	LBKM	MME	SBDT	RBLM	SLEEP	INIT
W																
Reset	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0

Figura 5.7 - LIN control register (LINCR1)

Descrizione del *field* INIT del registro LINCR1:

Field	Description
INIT	Initialization Request The software sets this bit to switch hardware into Initialization mode. If the SLEEP bit is reset, LINFlex enters Normal mode when clearing the INIT bit (see Table 21-6).

Figura 5.8 - LINCR1 field description

Per fare in modo che comunichino in modo corretto, il dispositivo Bluetooth e l'UART del microcontrollore devono avere le stesse impostazioni ed il registro UARTCR permette fare questo. Inizialmente si pone il bit UART = 1 per entrare in *UART mode* e successivamente è possibile impostare tutti gli altri parametri tramite gli appositi *fields*. Per l'utilizzo in questa applicazione l'UART è stata settata con 8 bit di dati, no bit di parità, Tx e Rx abilitati e Buffer in ricezione/trasmissione di dimensione 1(come descritto nel codice relativo alla funzione).

Offset: 0x0010 Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	0	TDFL		0	RDFL		0	0	0	0	RXEN	TXEN	OP	PCE	WL	UART
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 5.9 - UART mode control register (UARTCR)

Elenco e descrizione dei *field* utilizzati per impostare i parametri di comunicazione dell'UART:

Field	Description
TDFL	Transmitter Data Field length This field sets the number of bytes to be transmitted in UART mode. It can be programmed only when the UART bit is set. TDFL[0:1] = Transmit buffer size – 1. 00 Transmit buffer size = 1. 01 Transmit buffer size = 2. 10 Transmit buffer size = 3. 11 Transmit buffer size = 4.
RDFL	Receiver Data Field length This field sets the number of bytes to be received in UART mode. It can be programmed only when the UART bit is set. RDFL[0:1] = Receive buffer size – 1. 00 Receive buffer size = 1. 01 Receive buffer size = 2. 10 Receive buffer size = 3. 11 Receive buffer size = 4.
RXEN	Receiver Enable 0 Receiver disable. 1 Receiver enable. This bit can be programmed only when the UART bit is set.
TXEN	Transmitter Enable 0 Transmitter disable. 1 Transmitter enable. This bit can be programmed only when the UART bit is set. **Note:** Transmission starts when this bit is set and when writing DATA0 in the BDRL register.
OP	Odd Parity 0 Sent parity is even. 1 Sent parity is odd. This bit can be programmed in Initialization mode only when the UART bit is set.
PCE	Parity Control Enable 0 Parity transmit/check disable. 1 Parity transmit/check enable. This bit can be programmed in Initialization mode only when the UART bit is set.
WL	Word Length in UART mode 0 7-bit data + parity bit. 1 8-bit data (or 9-bit if PCE is set). This bit can be programmed in Initialization mode only when the UART bit is set.
UART	UART mode enable 0 LIN mode. 1 UART mode. This bit can be programmed in Initialization mode only.

Figura 5.10 - UARTCR field description

Il *baud rate* del ricevitore e del trasmettitore sono entrambi impostati allo stesso valore come programmato nei registri della mantissa (LINIBRR) e delle frazioni (LINFBRR) . Il registro con il valore di baud rate non deve essere modificato durante una transazione e il registro LINFBRR (contente i bits di frazione) deve essere programmato prima del registro LINIBRR. Il valore utilizzato per comunicare con il Bluetooth è di 115200 *baud rate.*

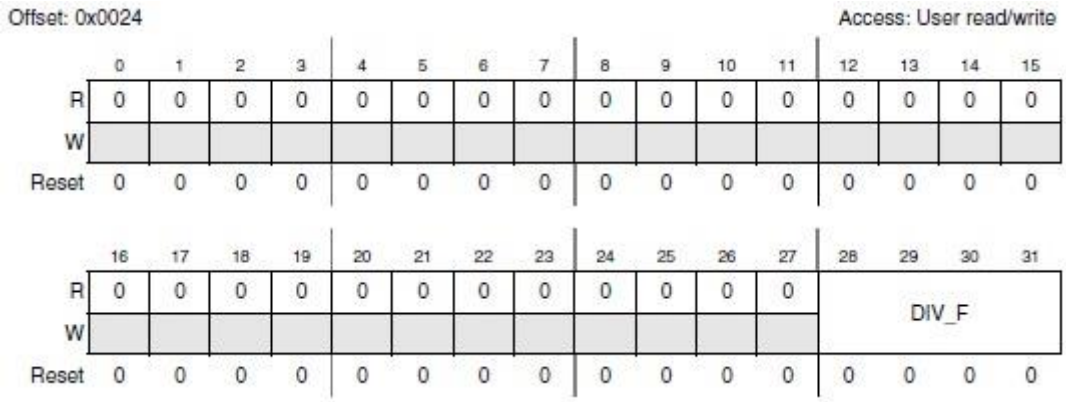

Figura 5.11 - LIN fractional baud rate register (LINFBRR)

Descrizione del *field* DIV_F del registro LINFBRR per l'impostazione del *bit* di frazione:

Field	Description
DIV_F	Fraction bits of LFDIV The 4 fraction bits define the value of the fraction of the LINFlex divider (LFDIV). Fraction (LFDIV) = Decimal value of DIV_F / 16. This field can be written in Initialization mode only.

Figura 5.12 - LINFBRR field description

Registro LINIBRR per l'impostazione della Mantissa:

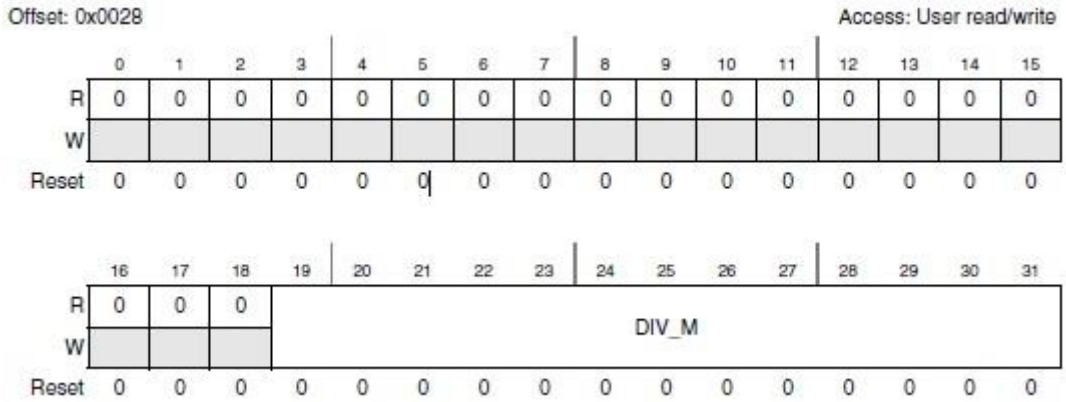

Figura 5.13 - LIN integer baud rate register (LINIBRR)

Descrizione del *field* DIV_M del registro LINIBRR per l'impostazione della Mantissa:

Field	Description
DIV_M	LFDIV mantissa This field defines the LINFlex divider (LFDIV) mantissa value (see Table 21-17). This field can be written in Initialization mode only.

Figura 5.14 - LINIBRR field description

Di seguito viene riportata la tabella con i valori dei registri LINIBRR e LINFBRR per settare la velocità di trasmissione dati (*baud rate*) ed il relativo errore calcolato tra il baud rate impostato e quello reale, per diversi valori predefiniti:

Baud rate	$f_{periph\_set\_1\_clk}$ = 64 MHz				$f_{periph\_set\_1\_clk}$ = 16 MHz			
	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate	Actual	Value programmed in the baud rate register		% Error = (Calculated – Desired) baud rate / Desired baud rate
		LINIBRR	LINFBRR			LINIBRR	LINFBRR	
2400	2399.97	1666	11	–0.001	2399.88	416	11	–0.005
9600	9599.52	416	11	–0.005	9598.08	104	3	–0.02
10417	10416.7	384	0	–0.003	10416.7	96	0	–0.003
19200	19201.9	208	5	0.01	19207.7	52	1	0.04
57600	57605.8	69	7	0.01	57554	17	6	–0.08
115200	115108	34	12	–0.08	115108	8	11	–0.08
230400	230216	17	6	–0.08	231884	4	5	0.644
460800	460432	8	11	–0.08	457143	2	3	–0.794
921600	927536	4	5	0.644	941176	1	1	2.124

Figura 5.15 - Valore programmato nei registri e relativo errore

Impostando il *field* DRIE = 1 nel registro LINIER è possibile generare un *interrupt* quando il flag di fine ricezione dati (DRF) viene abilitato da parte dei registri LINSR o UARTSR. Questo risulta essere di fondamentale importanza, perché come vedremo in seguito, permetterà di richiamare la funzione per la lettura dei dati in ricezione.

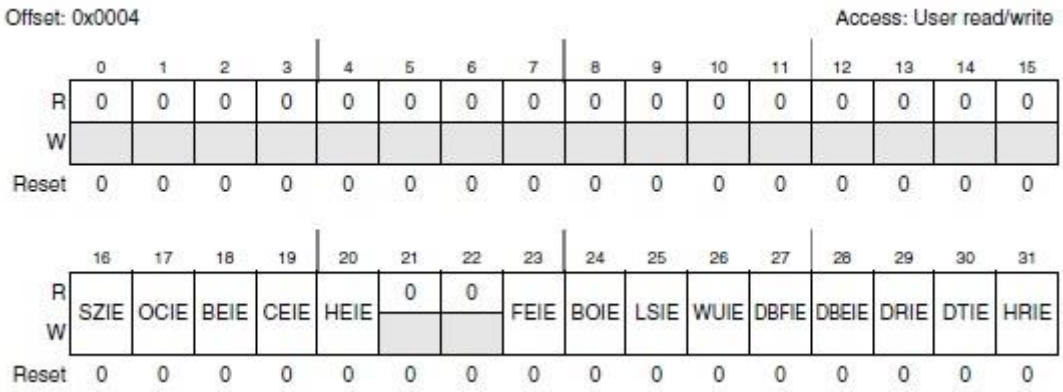

Figura 5.16 - LIN interrupt enable register (LINIER)

Descrizione del *field* DRIE del registro LINIER per abilitare l'*interrupt:*

Field	Description
DRIE	Data Reception Complete Interrupt Enable 0 No interrupt when data reception is completed. 1 Interrupt generated when data received flag (DRF) in LINSR or UARTSR is set.

Figura5.17 - LINIER field description

Quando viene generato un *interrupt* dovuto al DRF (Data Reception Completed Flag) viene chiamata la funzione *void Uart_Int void)* per gestire i dati in ricezione dal *buffer*. Questo metodo, come si vede dal codice, una volta invocato resta in attesa finché tutti i dati spediti da PC siano ricevuti, successivamente il flag viene resettato e prepara l'UART per la ricezioni di altri. Come si vedrà meglio nel paragrafo successivo, l'impostazione di *read_uart* = 1 permette di memorizzare i 4 byte del buffer in 4 variabili distinte, per poi utilizzarle nella routine di ricezione/modifica dei parametri.

```
void Uart_Int(void)
{
    /* wait for DRF */
    while (1 != LINFLEX_1.UARTSR.B.DRF) {}/*Wait for data reception completed flag*/
    /* wait for RMB */
    while (1 != LINFLEX_1.UARTSR.B.RMB) {}  /* Wait for Release Message Buffer */
    /* clear the DRF and RMB flags by writing 1 to them */
    LINFLEX_1.UARTSR.R = 0x0204;
    /* set Uart mode and wl by writing 1 to them */
    LINFLEX_1.UARTCR.R = 0x0003;
    /* abilita la routine di lettura del buffer dati nel main.c */
    read_uart = 1;
}
```

Il registro UARTSR (UART Status Register) abilita il flag DRF (Data Reception Completed Flag), impostato via *hardware*, e indica che la ricezione dei dati è completa. Successivamente viene abilitato il messaggio RMB (Relase Message Buffer) e indica che il buffer è pronto per essere letto via software.

Offset: 0x0014 — Access: User read/write

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
R	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
W																
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
R	SZF	OCF	PE3	PE2	PE1	PE0	RMB	FEF	BOF	RPS	WUF	0	0	DRF	DTF	NF
W	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c	w1c			w1c	w1c	w1c
Reset	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figura 5.18 - UART mode status register (UARTSR)

Descrizione dei *field* DRF e RMB del registro UARTSR:

Field	Description
DRF	Data Reception Completed Flag This bit is set by hardware and indicates the data reception is completed, that is, the number of bytes programmed in RDFL[0:1] in UARTCR have been received. This bit must be cleared by software. It is reset by hardware in Initialization mode. An interrupt is generated if DRIE bit in LINIER is set. **Note:** In UART mode, this flag is set in case of framing error, parity error or overrun.
RMB	Release Message Buffer 0 Buffer is free. 1 Buffer ready to be read by software. This bit must be cleared by software after reading data received in the buffer. This bit is cleared by hardware in Initialization mode.

Figura 5.19 - UARTSR field descriptions

La funzione *void printserialsingned(innum)* permette di inviare una parola di 5 cifre, anticipata dal segno +/- a seconda che il numero inviato sia positivo o negativo. Come anticipato nella paragrafo 5.1.1 riguardo al *client.c*, serve per visualizzare sulla *shell* di Ubuntu i valori dei parametri scelti o modificati dall'utente.

```
void printserialsingned(uint16_t innum)
{
   uint16_t j1,k1,l1,m1,in;
   uint8_t p1,p2,p3,p4,p5;

   if(innum < 0x8000)
   {
      in = innum;
      TransmitCharacter('+');
   }
   else
   {
      in = (uint16_t)(~innum);
      Delaylonglong();
      TransmitCharacter('-');
   }

   j1 = (in / 10);
   p1 = (uint8_t)(in - j1*10 +0x30);
   k1 = (j1 / 10);
   p2 = (uint8_t)(j1 - k1*10 +0x30);
   l1 = (k1 / 10);
   p3 = (uint8_t)(k1 - l1*10 +0x30);
   m1 = (l1 / 10);
   p4 = (uint8_t)(l1 - m1*10 +0x30);
   p5 = (uint8_t)m1 +0x30;

   TransmitCharacter(p5);
   TransmitCharacter(p4);
   TransmitCharacter(p3);
   TransmitCharacter(p2);
   TransmitCharacter(p1);
}
```

La funzione *void TransmitCharacter(ch)* si occupa della scrittura del carattere nel buffer di trasmissione. E' integrata all'interno della funzione *printserialsingned()* e viene richiamata sei volte, di cui cinque per l'invio dei caratteri alfanumerici e una per il segno.

```
void TransmitCharacter(uint8_t ch)
{
     LINFLEX_1.BDRL.B.DATA0 = ch; /* write character to transmit buffer */
     while (1 != LINFLEX_1.UARTSR.B.DTF) {}/* Wait data transmission completed flag */
     LINFLEX_1.UARTSR.R = 0x0002; /* clear the DTF flag and not the other flags */

}
```

5.2.4 Main.c

Tutti i programmi eseguibili in linguaggio C devono definire la funzione *main()*. Il codice sorgente di un programma può esser suddiviso in più *sources*, ma per creare un *file* eseguibile, solo uno di questi può contenere la funzione speciale *main()*.

```
#include "MPC5604B.h"
#include <string.h>
#include "IntcInterrupts.h"
#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include "pinout.h"
#include "uart.h"
#include "delay.h"
#include "define.h"

extern uint8_t read_uart;
char comando;
uint16_t UartV1;
uint16_t UartV2;
uint16_t UartV3;
char val0;
char val1;
char val2;
char val3;
uint8_t ReadyUart = 0;
double nuovo_valore = 0;
uint16_t k_bluetooth = 0;
uint8_t counter_bluetooth;
int16_t CameraBuffer[128];
extern double KpSterzo;
extern double KiSterzo;
extern double KdSterzo;
extern double KpSpeed;
extern double KdSpeed;
extern double KiSpeed;
int16_t SpeedRef = 3500;
int16_t SpeedStraight = 0;
int16_t SpeedTurn = 0;
double SpeedReadSx = 0;
double SpeedReadDx = 0;
float PerDifferenziale = 0.17;
extern int16_t DevArray[127];
uint8_t LineCenter;
extern uint16_t Steering;
```

Descrizione della funzione main()

La funzione principale di tutto il programma è il *void main()*, dove sono richiamate in un *loop* infinito tutte le funzioni implementate che ne determinano l'andamento autonomo del veicolo.
La funzione comincia con l'inizializzazione degli eMIOS che permettono al micro di comunicare con le periferiche ad esso collegate ed interfacciarsi con l'ambiente esterno. Successivamente vengono installati sia gli *interrupt* per la lettura ad intervalli regolari delle periferiche che quello per la ricezione dati da parte del Bluetooth. Nell'ultima parte, all'interno del ciclo infinito, continuano ad essere richiamate le funzioni riguardanti la telecamera, i pinout, lo sterzo ed i motori e si continua ad

aggiornare l'*array* dei centrilinea trovati, si imposta la velocità di rettilineo o di curva, si attiva il differenziale elettronico se si è in curva e si ferma il veicolo quando si è oltrepassato il segnale di stop. Di seguito non verrà riportato tutto il source *main.c*, ma solo una parte della funzione void main(void) inerente all'installazione degli *interrupt* e alla routine di lettura/modifica del buffer dati in ricezione. Quest'ultima verrà divisa in sezioni per semplificarne la comprensione:

- Installazione degli *interrupt*
- Prima sezione della routine
- Seconda sezione della routine

Installazione degli interrupt

Per inizializzare la periferica LINFlex_1 viene richiamata la funzione *init_LinFLEX_1_UART()*, mentre la funzione per l'installazione *dell'interrupt handle* è richiamata nel seguente modo: *INTC_InstallINTCInterruptHandler(Uart_Int,99,3)*, dove uart_int indica il programma da avviare quando viene fatta la richiesta di interrupt (ISR), 99 è il numero associato all'interrupt request (IRQ) da parte della LINFlex_RXI e 3 è la priorità assegnata alla richiesta di interrupt da parte della LINFlex.

Prima sezione della routine

Per entrare nella routine di ricezione del buffer dati proveniente da Bluetooth, deve essere soddisfatta la condizione del primo *if()*. Questo avviene quando è abilitato il *flag* di trasmissione completa (DTF) del registro UARTSR, permettendo così di eseguire la funzione *Uart_Int()* associata alla richiesta di interrupt. Come visto precedentemente, questa funzione pone read_uart = 1. Le istruzioni dell'*if()* permettono la lettura dei singoli byte DATA[4:7] del registro BDRM. Una volta memorizzati i dati proveniente dal buffer si può entrare nella sezione di lettura/modifica.

```
if(read_uart==1)
{
    /* get the data*/
    val0 = (char)LINFLEX_1.BDRM.B.DATA4; /*buffer dati in ricezione da UART*/
    val1 = (char)LINFLEX_1.BDRM.B.DATA5;
    val2 = (char)LINFLEX_1.BDRM.B.DATA6;
    val3 = (char)LINFLEX_1.BDRM.B.DATA7;

    comando = (uint8_t)val0; /*inizializzazione del buffer dati*/
    UartV1 = (uint8_t)val1;
    UartV2 = (uint8_t)val2;
    UartV3 = (uint8_t)val3;
    ReadyUart = 1;
    read_uart = 0;
}
```

Il buffer in *UART mode* è diviso in 4 byte per la trasmissione dati (*Transmit buffer*) e altri 4 byte per la ricezione (*Receive buffer*). Dal lato *Host* (PC), dopo aver riempito e inviato il buffer di 4 byte a disposizione, viene ricevuto dal microcontrollore e salvato nel registro BDRM, rispettivamente nei campi DATA[4:7]. I dati salvati vengono cancellati quando il micro è resettato o tolta l'alimentazione.

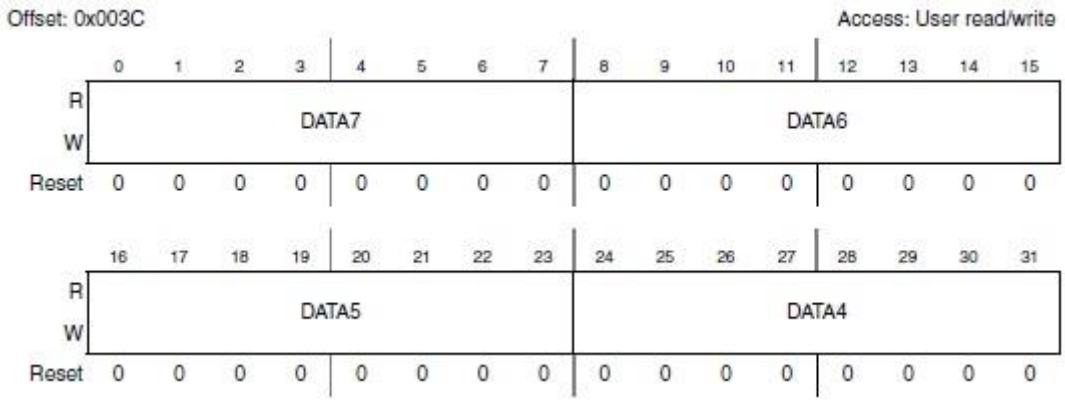

Figura 5.20 - Buffer data register MSB (BDRM)

Descrizione dei *fields* DATA[4:7] del registro BDRM per la memorizzazione del *buffer* dati.

Field	Description
DATA7	Data Byte 7 Data byte 7 of the data field.
DATA6	Data Byte 6 Data byte 6 of the data field.
DATA5	Data Byte 5 Data byte 5 of the data field.
DATA4	Data Byte 4 Data byte 4 of the data field.

Figura 5.21 - BDRM field descriptions

Seconda sezione della routine

Questa sezione si divide in due parti:

- **Ricezione**: comandi compresi tra 1 e 39
- **Modifica**: comandi da 40 in poi

Nella parte di ricezione non è possibile modificare i dati, ma soltanto di leggere i valori delle variabili che in quel momento sono presenti all'interno della funzione *void main(void)* e quindi che il veicolo sta utilizzando in quel momento. Una volta digitato da tastiera (lato PC) il comando corrispondente alla variabile della quale si vuole conoscere il valore ed il numero di ricezioni, si entra nel relativo *if()* e la funzione *printserialsingned()* tramite Bluetooth lo invia al PC che verrà visualizzato nella *shell* di Ubuntu.

```
/* dati in sola ricezione (no modifica) */
if(ReadyUart==1)
{
    if((comando > 0) && (comando < 40))
    {
        if((UartV1!=0)&&((k_bluetooth%5)==0))
        {
```

```
            if(comando == 1)    //lettura velocità media
            {
                printserialsingned((uint16_t)SpeedRef);
            }
            if(comando == 2)    //lettura velocità rettilineo
            {
                printserialsingned((uint16_t)SpeedStraight);
            }
            if(comando == 3)    //lettura velocità curva
            {
                printserialsingned((uint16_t)SpeedTurn);
            }
            if(comando == 5)    //lettura centrolinea
            {
                printserialsingned((uint16_t)LineCenter);
            }
            if(comando == 9)    //lettura percentuale differenziale
            {
                printserialsingned((uint16_t)PerDifferenziale);
            }
            if(comando == 12)   //lettura velocità ruota dx
            {
                  printserialsingned((uint16_t)SpeedReadDx);
            }
            if(comando == 13)   //lettura velocità ruota sx
            {
                  printserialsingned((uint16_t)SpeedReadSx);
            }
            if(comando == 16)   //lettura sterzo
            {
                  printserialsingned((uint16_t)Steering);
            }
            if(comando == 17)   //stampa array telecamera bassa
            {
            if(count == 0)
            {
                  for(z=0;z<128;z++)
                  {
                        CameraBuffer[z] = DevArray[z];
                  }
            }
            printserialsingned((uint16_t)CameraBuffer[count]);
            count++;
        }
        k_bluetooth = 0;
        UartV1 = UartV1 - 1;     // finchè UartV1 non è uguale a zero continua
                                 // continua a trasmettere
    }
    if(UartV1==0)
    ReadyUart = 0;

}
```

Nella parte di modifica è possibile cambiare i valori delle variabili precedentemente dichiarate nelle funzioni, durante il funzionamento del veicolo. Una volta inviato il comando corrispondente alla variabile da modificare è possibile digitare da tastiera il nuovo valore da attribuire. In fine tramite la funzione *printserialsingned()* viene inviato e visualizzato nella *shell* di Ubuntu il nuovo valore attribuito.
Di seguito è riportata solo la parte di scrittura dei valori x0.001 (diviso mille), le altre funzionano allo stesso modo.

```
if(comando >= 40 && comando < 80) //valori del tipo .122 (scrittura x0.001)
{
	nuovo_valore = (((UartV1*1.0)/10)+((UartV2*1.0)/100)+((UartV3*1.0)/1000));

	if (comando == 40)		//VELOCITA' RETTILINEO
		SpeedRef = (int16_t)nuovo_valore;
	if (comando == 42)		//KP_STEERING
		KpSterzo = nuovo_valore;
	if (comando == 43)		//KD_STEERING
		KdSterzo = nuovo_valore;
	if (comando == 44)		//KI_STEERING
		KiSterzo = nuovo_valore;
	if (comando == 45)		//KP_SPEED
		KpSpeed = nuovo_valore;
	if (comando == 46)		//KD_SPEED
		KdSpeed = nuovo_valore;
	if (comando == 47)		//KI_SPEED
		KiSpeed = nuovo_valore;
	ReadyUart=0;
	printserialsingned((UartV1*100)+(UartV2*10)+UartV3);
}
```

Conclusioni

Con questo elaborato si vuole offrire uno strumento di consultazione per gli studenti che si accingono a partecipare alla competizione internazionale "Freescale Cup", con l'intento di illustrare in modo semplice ed esaustivo le caratteristiche tecniche dei componenti hardware forniti con il kit freescale, ma sopratutto le potenzialità del modulo Bluetooth nella fase finale di *tuning* del veicolo. Infatti l'obiettivo di questa competizione è di percorrere il tracciato nel minor tempo possibile e quindi ottimizzare l'algoritmo per ottenere un veicolo performante risulta essere di fondamentale importanza.
Il documento esposto risulta essere più completo rispetto ai normali manuali tecnici e ha lo scopo di introdurre il lettore nell'ottica delle comunicazioni Bluetooth e della relativa programmazione software per ottenere un *data logging* efficiente, rimandando poi alla lettura di testi tecnici per ulteriori approfondimenti.
Gli argomenti trattati, si spera, possano servire come punto di partenza per gli studenti che si avvicinano a questa competizione, con la possibilità di trovare soluzioni più efficienti come ad esempio ottenere una telemetria più veloce e sicura, la ricezione e visualizzazione di più dati contemporaneamente e il salvataggio dei nuovi valori nella memoria del microcontrollore.
Infine nonostante i numerosi sforzi ed il costante impegno per sei mesi, questa è risultata essere una valida esperienza dove per la prima volta si è riusciti a mettere in campo le basi della meccatronica acquisite nel corso del triennio, nonché un modo più pratico di approcciarsi ai problemi affrontati e quindi utile per affacciarsi al mondo del lavoro.

Bibliografia e Sitografia

1. Freescale Semiconductor, MPC5604B/C Microcontroller Reference Manual, Rev. 8.1, May 2012
2. Taos, TSL1401CL 128x1 linear sensor array with hold, July 2011
3. Freescale Semiconductor, MC33931 Data Sheet, Rev. 3.0, June 2012
4. M. Zigliotto, Dispense del corso di Fondamenti di Macchine ed Azionamenti Elettrici, 2013
5. A. Pretto, Dispense del corso di Linguaggi di Programmazione per Sistemi Industriali, 2012
6. https://community.freescale.com
7. Futaba (http://www.futabarc.com)
8. Standard Motors (http://www.standardmotor.net/)
9. http://en.wikipedia.org/wiki/Freescale_Semiconductor
10. Parallax, RN-42 Bluetooth Module Guide, v1.0, 2013
11. Bluetooth (http://www.bluetooth.com/)
12. Bluetooth programming (http://people.csail.mit.edu/albert/bluez-intro/)
13. http://en.wikipedia.org/wiki/Bluetooth