



TOR VERGATA
UNIVERSITÀ DEGLI STUDI DI ROMA

**UNIVERSITÀ DEGLI STUDI DI ROMA
TOR VERGATA**

FACOLTÀ DI INGEGNERIA

CORSO DI LAUREA IN INGEGNERIA DELL'AUTOMAZIONE

A.A. 2020/2021

Tesi di Laurea

Sviluppo di algoritmi di controllo delle correnti nelle bobine poloidali di macchine per la fusione Tokamak,
con riguardo al design sistematico per la cooperazione tra sistemi embedded di attuazione, misurazione e controllo.

RELATORE

Daniele Carnevale

CANDIDATO

Emanuele Alfano

CORRELATORI

Marco Passeri

Dedico questa tesi ai miei cari nonni.

Grazie per aver sempre creduto in me.

Indice

Ringraziamenti	1
1 Introduzione	2
1.1 Fusione Termonucleare	2
1.2 Struttura di un Tokamak	2
1.3 Obiettivi della Tesi	2
1.4 Struttura della Tesi	2
2 Elementi Costitutivi	3
2.1 Trasformatore, un Modello di Tokamak	4
2.1.1 Richiami di elettronica	5
2.1.2 Modellazione Fisica	9
2.1.3 Semplificazione Primario - Secondario	10
2.1.4 Dal circuito alla dinamica	11
2.1.5 Funzione di Trasferimento	11
2.1.6 Misura del campo Elettrico del Plasma come indice per la Corrente	13
2.2 Trasduttore di Corrente	15
2.2.1 Sensore scelto	15
2.2.2 Criticità	16
2.2.3 Funzionamento Interno	16

2.2.4	Connessione elettrica	19
2.2.5	Misura	19
2.3	Driver di Corrente - IBT-2	21
2.3.1	Schema Elettrico	22
2.3.2	Connessione di Controllo	22
2.3.3	Benchmark del Driver	24
3	Architettura di Sistema	26
3.1	Architettura ad alto livello	26
3.1	EMP - Libreria di Comunicazione Seriale	28
3.1	Protocollo - COBS	29
3.1.1	Metodo di codifica	29
3.1.2	Caratteristiche chiave	30
3.1.3	Integrità dei pacchetti	32
3.1.4	Struttura del codice	33
3.1.5	Logica di Comunicazione	34
3.1.6	Code Flow	35
3.1.7	Test di Codifica/Decodifica su ogni Device	36
3.2	Online Sampling	37
3.2.1	Interconnessione μ Controllore \Leftrightarrow Companion	38
3.2.2	Storage su file delle informazioni	39
3.3	Post Elaborazione con Matlab	40
3.3.1	Conversioni Dati	40
3.3.2	Creazione dei grafici e Filtraggio	40
4	Modello teorico di Controllo	41

4.1	Controllo a Errore Nullo	41
4.2	Simulazione Qualitativa su Simulink	41
5	Sviluppo Controllo reale	42
5.1	Codifica del controllore	42
5.2	Tuning delle costanti	42
5.3	Esperimenti	42
6	Conclusioni e sviluppi futuri	43
7	Software, Toolchain, Strumenti	44
Appendice A - Codice Arduino		45
8.1	Set-up Registri	45
8.1.1	Tic Timer	45
8.1.2	Frequenza PWM	45
8.2	Generatore di Segnale	47
8.2.1	Segnali Base	47
8.2.2	Segnali Composti	48
8.3	Codici Controllore	49
Appendice B - Codice EMP		50
9.4	Pacchetti in EMP	50
9.4.1	Definizione di 2 Pacchetti	50
9.4.2	Pacchetti Multipli	51
Appendice C - Matlab Post Elaboration		52
Elenco delle figure		54

Ringraziamenti

Corpo dei ringraziamenti

Capitolo 1

Introduzione

1.1 Fusione Termonucleare

1.2 Struttura di un Tokamak

1.3 Obiettivi della Tesi

1.4 Struttura della Tesi

Capitolo 2

Elementi Costitutivi

In questo capitolo si vogliono descrivere e caratterizzare i 3 elementi salienti dell'esperimento:

1. *Trasformatore, un Modello di Tokamak*
2. *Trasduttore di Corrente*
3. *Driver di Corrente - IBT-2*

Verranno analizzate le loro caratteristiche chiavi per mettere in luce il perché della scelta, e si evidenzieranno eventuali problemi che affliggono in componenti, problemi di cui si è tenuto conto nello sviluppo del progetto per poterli annullare.

2.1 Trasformatore, un Modello di Tokamak

Come visto nell'introduzione, la tesi ha come obiettivo la prototipazione del sistema di controllo per le bobine poloidali presenti in impianti tokamak.

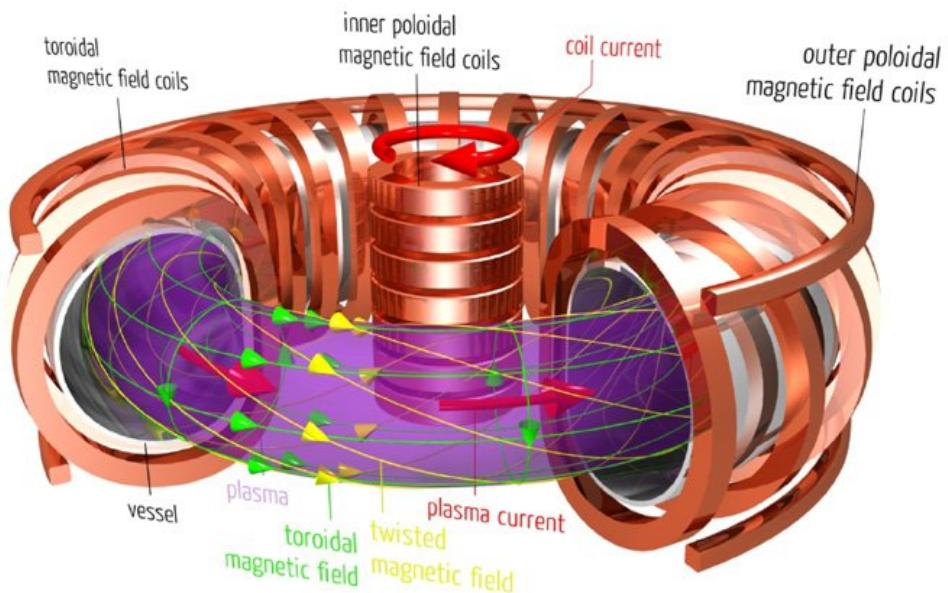


Figura 2.1: Interno Tokamak

Le bobine (poloidali e toroidali) servono a controllare il plasma presente nel *Vessel* dell'impianto e confinare il Plasma all'interno di un flusso compresso dentro la camera, le alte temperature e la forte compressione a cui è sottoposto il Plasma, permette di realizzare eventi di **Fusione Nucleare** tra gli atomi di idrogeno a base del plasma.

L'interazione tra le **Bobine e Plasma**, ha un modello matematico non dissimile da quello di **Trasformatore Elettrico** nella relazione *Primario-Secondario* (con i dovuti paragoni e le ovvie *Non-Linearità* presenti nel caso di un impianto Tokamak reale).

Grazie a questa similitudine è stato possibile replicare in sicurezza la fisica presente all'interno di un Tokamak, nell'ambiente controllato del laboratorio.

2.1.1 Richiami di elettronica

Prima di modellare ed analizzare l'esperimento della tesi, è necessario richiamare qualche proprietà/concetto di elettronica per poter comprendere i passaggi matematici e fisici:

- Prima legge di Kirchhoff (legge dei nodi)
- Seconda legge di Kirchhoff (legge delle maglie)
- Induttore Ideale
- Induttanza
- Trasformatore Ideale Monofase

Teorema 2.1.1 (Prima legge di Kirchhoff (legge dei nodi)).

La somma algebrica delle intensità di corrente nei rami facenti capo allo stesso nodo è nulla.

$$\sum I_k = 0 \quad (2.1.1)$$

■

Teorema 2.1.2 (Seconda legge di Kirchhoff (legge delle maglie)).

La somma algebrica delle f.e.m. agenti lungo i rami di una maglia è uguale alla somma algebrica dei prodotti delle intensità di corrente di ramo per le rispettive resistenze (del ramo).

$$\sum_{\forall k} V_k = \sum_{\forall k} f_{emk} \quad (2.1.2)$$

■

Oltre ai teoremi di Kirchhoff, che ci serviranno per ricavare le equazioni della dinamica, enunciamo ora le proprietà degli induttori, indispensabili per poter definire la loro relazione Corrente/Tensione.

Definizione 2.1.1 (Induttore Ideale).

Un Induttore Ideale si oppone solo alle variazioni di corrente, variando la tensione ai suoi capi di conseguenza, non presenta nessuna resistenza elettrica in caso di correnti costanti ai suoi capi.

Il suo valore è dato dal **coefficiente di autoinduzione**, tipicamente espresso con il simbolo L , la cui unità di è l'Henry [H].

Un Induttore accumula energia all'interno di un campo magnetico, e questa relazione è descritta dall'equazione:

$$\Phi_B = Li \quad (2.1.3)$$

$\Phi_B :=$ Flusso magnetico; $L :=$ Coefficiente di autoinduzione

Dalla legge di Faraday(ignorando momentaneamente la conservazione dell'energia, ovvero la legge di Lenz), applicata alla circuitazione del circuito costituito dalla induttanza stessa, si ha:

$$\frac{d\Phi_B}{dt} = V \quad (2.1.4)$$

Dove V è il potenziale indotto ai morsetti del circuito in questione. Perciò, derivando l'equazione 2.1.3 ad entrambi i membri rispetto al tempo, si ottiene:

$$\frac{d\Phi_B}{dt} = L \frac{di}{dt} + i \frac{dL}{dt} \quad (2.1.5)$$

In molti casi fisici, però, l'induttanza può essere considerata costante rispetto al tempo (o tempo-invariante), da cui:

$$\frac{d\Phi_B}{dt} = L \frac{di}{dt} \quad (2.1.6)$$

Combinando le equazioni precedenti si ha:

$$V(t) = L \frac{di(t)}{dt} \Leftrightarrow i(t) = \frac{1}{L} \int V(t) dt \quad (2.1.7)$$



Nel calcolo della 2.1.3, abbiamo omesso la legge di Lenz, poichè per parametrizzare l'induttore il segno

"- " complica inutilmente i calcoli essendo assegnato dalla polarizzazione del circuito esaminato.

Essa al contrario, prende un importanza notevole in presenza di fenomeni di **Induttanza**:

Definizione 2.1.2 (Induttanza).

L'induttanza è la proprietà dei circuiti elettrici tale per cui la corrente (intesa variabile nel tempo) che li attraversa induce una **Forza ElettroMotrice** (f.e.m.) Indotta che si oppone alla variazione di corrente, per la legge di Lenz. In questi scenari (vedi Trasformatore Elettrico ad esempio), 2 circuiti sono accoppiati magneticamente tra di loro attraverso 2 induttori, si ottiene quindi che, la variazione del flusso magnetico da parte del primo induttore, crea una **Forza ElettroMotrice** (f.e.m.) indotta contraria:

$$-\frac{d\Phi_B}{dt} = \mathcal{E} = V \quad (2.1.8)$$

Dove \mathcal{E} è la **Forza ElettroMotrice** (f.e.m.) indotta.



Per finire, unendo insieme i concetti di **Induttore Ideale** e **Induttanza** è possibile ottenere la descrizione matematica di un Trasformatore Monofase Ideale (*Trasformatore Monofase*¹):

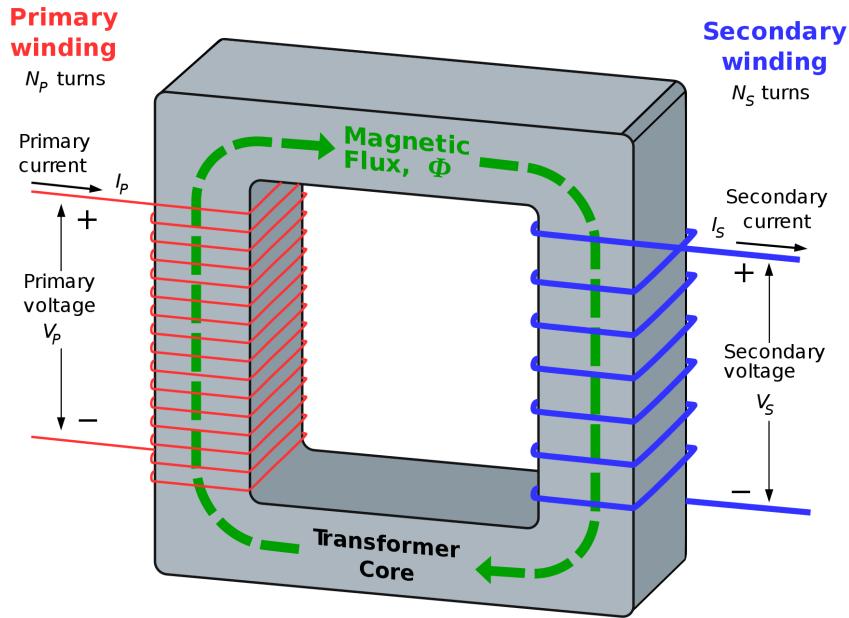


Figura 2.2: Trasformatore Ideale

Definizione 2.1.3 (Trasformatore Ideale Monofase).

Il Trasformatore è una macchina elettrica, basata sul fenomeno dell'induzione elettromagnetica, destinata a trasformare, tra il circuito primario (ingresso) e il circuito secondario (uscita) del trasformatore, i fattori tensione e corrente della potenza elettrica.

Trasferisce quindi energia elettrica da un circuito elettrico a un altro che ha una tensione diversa, accoppiandoli induttivamente, senza che siano a contatto tra loro gli avvolgimenti del trasformatore. Il trasformatore è una macchina reversibile.

Il Trasformatore è costruito affinché il circuito Primario e il Secondario condividano lo stesso campo magnetico, e quindi lo stesso flusso Φ_B .

Nel caso ideale, quindi, si ha che:

$$\Phi_B = \Phi_{B_P} - \Phi_{B_S} \quad (2.1.9)$$

¹Senso di misura del secondario opposto nell'articolo a quello qui presentato

Siccome tale flusso varia nel tempo, induce nei due avvolgimenti le f.e.m.([Induttanza](#)):

$$e_p = N_1 \frac{d\Phi_B}{dt}; e_s = -N_2 \frac{d\Phi_B}{dt} \quad (2.1.10)$$

I segni non sono entrambi concordi a causa del verso del flusso, che nel primario è concorde e nel secondario discorde (e qui torna la forza di Lenz).

Viste dai morsetti del trasformatore, abbiamo che le tensioni istantanee (dovute alle correnti presenti nei 2 rami) sono pari a:

$$\begin{cases} V_P = R_P I_P + L_{PP} \dot{I}_P - L_{SP} \dot{I}_S \\ V_S = R_S I_S - L_{PS} \dot{I}_P + L_{SS} \dot{I}_S \end{cases} \quad (2.1.11)$$

Dove R_P e R_S rappresentano le resistenze equivalenti viste dai morsetti (i carichi).

Con i coefficienti di mutua induttanza che si ricavano dalla Def di [Induttore Ideale](#):

$$\begin{aligned} L_{pp} &= \frac{N_P \Phi_{B_P}}{i_P} & L_{ps} &= \frac{N_S \Phi_{B_P}}{i_P} \\ L_{sp} &= \frac{N_P \Phi_{B_S}}{i_S} & L_{ps} &= \frac{N_S \Phi_{B_S}}{i_S} \end{aligned}$$



Arrivati a questo punto abbiamo gli strumenti necessari per poter modellare matematicamente l'esperimento.

2.1.2 Modellazione Fisica

Usando i concetti esposti, vediamo ora una buona approssimazione con modello lineare della dinamica tra **Bobina e Plasma**.

Come descritto nell'articolo di Romero et al., «[Real time current profile control at JET](#)», è possibile modellare la dinamica **Bobina - Plasma** come fosse il circuito di un [Trasformatore Ideale Monofase](#):

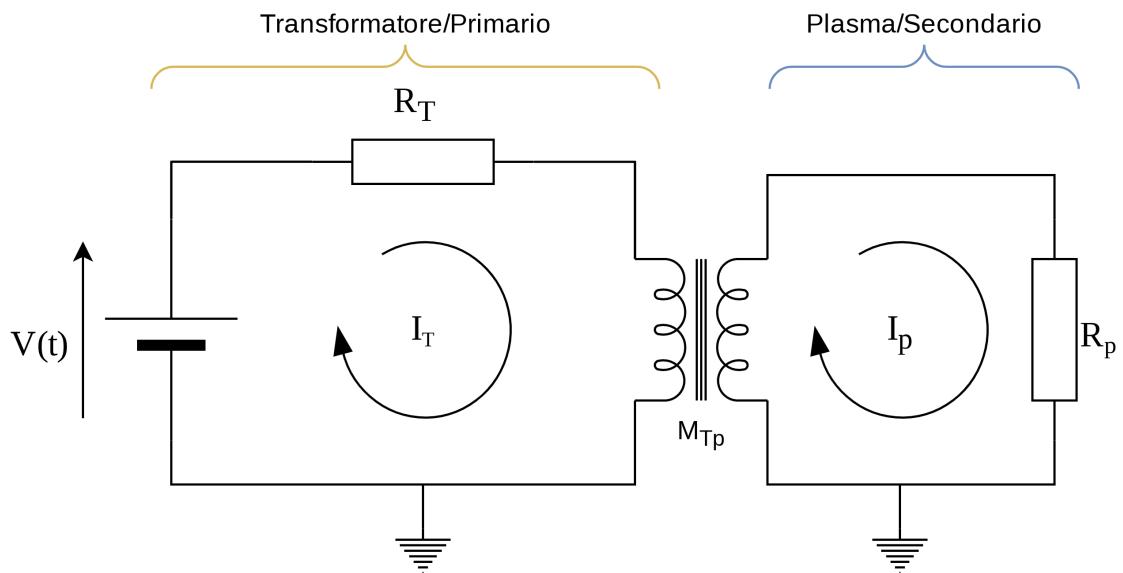


Figura 2.3: Circuito Equivalente Bobina/Plasma

- $V(t)$:= Tensione di controllo della Corrente I_T
- I_T := Corrente del Trasformatore
- R_T := Resistenza equivalente Trasformatore
- I_p := Corrente di Plasma
- R_p := Resistenza di Plasma
- M_{Tp} := Coefficiente di Induttanza Trasformatore → Plasma

Difformità dalla realtà Nella realtà R_p e M_{Tp} sono dei parametri che variano in funzione dello stato del plasma (Temperatura, Energia, Evoluzione dell'esperimento, ecc...), ma per ovvie ragioni di difficoltà nel riprodurre in laboratorio simili *Non-Linearità*, noi prenderemo per costanti questi parametri.

2.1.3 Semplificazione Primario - Secondario

Sempre dallo stesso articolo di Romero et al., «Real time current profile control at JET», si evince che è possibile modellare questa relazione tra i 2 circuiti prendendo in considerazione **solo** le forze di induzione dovute alla corrente che il Primario trasferisce sul Secondario, ciò permette di semplificare ulteriormente il circuito trascurando le correnti indotte dal Plasma dentro la Bobina del Primario.

Usando queste osservazioni si ottiene quindi il circuito equivalente:

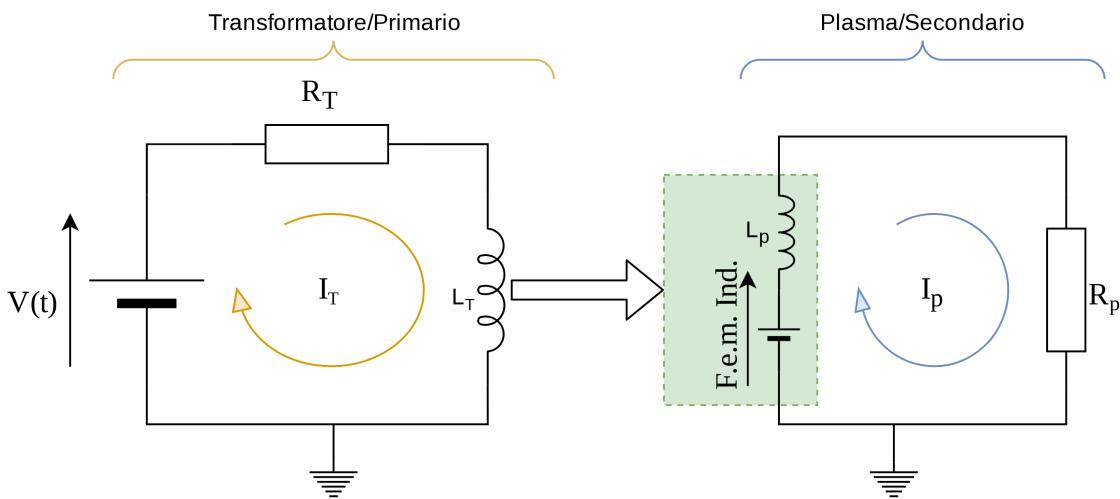


Figura 2.4: Circuito equivalente Bobina/Plasma semplificato

- $V(t)$:= Tensione di controllo della corrente I_T
- I_T := Corrente del Trasformatore
- R_T := Resistenza equivalente Trasformatore
- I_p := Corrente di Plasma
- R_p := Resistenza di Plasma
- M_{Tp} := Coefficiente di Induttanza Trasformatore → Plasma
 - L_T := Induttanza equivalente Trasformatore
 - L_p := Induttanza equivalente Plasma

Usando questa semplificazione dall'equazione 2.1.11 del trasformatore ignoriamo che la tensione sul Primario è influenzata dal Secondario; in questa condizione il circuito del Primario evolve come sistema indipendente rispetto al Secondario, e la sua evoluzione influenza il Secondario.

2.1.4 Dal circuito alla dinamica

Analizziamo ora la dinamica del circuito 2.1.3.

Dinamica di Plasma

Iniziando dal calcolare la dinamica nella corrente di Plasma usando la [Seconda legge di Kirchhoff \(legge delle maglie\)](#):

$$F_{em} = L_p \dot{I}_p + I_p R_p$$

Sappiamo inoltre che la F_{em} , grazie all'[Induttanza](#) del circuito è pari a:

$$F_{em} = -M_{Tp} \dot{I}_T$$

Da cui otteniamo:

$$-M_{Tp} \dot{I}_T = L_p \dot{I}_p + I_p R_p \quad (2.1.12)$$

Da notare la somiglianza con l'equazione del trasformatore 2.1.11.

Dalla eq 2.1.12 possiamo notare che rendere la corrente di Plasma costante, equivale a fissare una F_{em} di riferimento costante, ed essendo problematico misurare la corrente direttamente nel plasma reale, usiamo la **Tensione di Loop**, che nel caso del trasformatore equivale alla tensione di uscita del Trasformatore.

Dinamica del Trasformatore

Vediamo ora la dinamica della corrente del Trasformatore, in funzione della *Tensione di controllo*:

Sempre grazie alla [Seconda legge di Kirchhoff \(legge delle maglie\)](#) scriviamo:

$$V(t) = L_T \dot{I}_T + I_T R_T \quad (2.1.13)$$

2.1.5 Funzione di Trasferimento

Nella sezione 2.1.4 abbiamo ottenuto la dinamica istantanea dei 2 rami del circuito 2.1.3, ricaveremo ora le funzioni di trasferimento dei 2 rami e troveremo la dinamica del Trasformatore.

Funzione di Trasferimento Plasma

Partendo dall'equazione 2.1.12, supponendo condizioni iniziali nulle, abbiamo:

$$sI_p L_p + I_p R_p = -sI_T M_{Tp} \Rightarrow I_p(sL_p + R_p) = -sI_T M_{Tp} \Rightarrow I_p(s) = \frac{-sM_{Tp}}{(sL_p + R_p)} I_T(s)$$

La cui, funzione di trasferimento risulta essere:

$$\frac{I_p(s)}{I_T(s)} = \frac{-sM_{Tp}}{(sL_p + R_p)} \quad (2.1.14)$$

Funzione di Trasferimento Trasformatore

Similmente, partendo dall'equazione 2.1.13 si ricava:

$$sI_T L_T + I_T R_T = V(s) \Rightarrow I_T(sL_T + R_T) = V(s) \Rightarrow I_T(s) = \frac{1}{(sL_T + R_T)} V(s)$$

La cui, funzione di trasferimento risulta essere:

$$\frac{I_T(s)}{V(s)} = \frac{1}{(sL_T + R_T)} \quad (2.1.15)$$

Osservazione 2.1.1. La funzione di trasferimento che abbiamo ottenuto rispecchia il circuito dell'esperimento, noi però abbiamo un sistema che si controlla in corrente e che ha un uscita in corrente potremmo allora riassegnare l'ingresso come:

$$V(t) = I_{ref}(t) \cdot R_T \Rightarrow I_{ref}(t) = \frac{V(t)}{R_T}$$

Ottenendo così:

$$I_T(s) = \frac{R_T}{(sL_T + R_T)} I_{ref}(s) = \frac{1}{(s\frac{L_T}{R_T} + 1)} I_{ref}(t) \quad (2.1.16)$$

In questa forma possiamo osservare che, per $I_{ref}(t) = I_{ref}$ costante, superato il transitorio, si ottiene $I_T = I_{ref}$, da cui abbiamo che la corrente sul trasformatore viene attuata con un certo ritardo dovuto alla costante di tempo $\tau = \frac{L_T}{R_T}$ \square

Funzione di Trasferimento Complessiva

Connettendo in serie i 2 blocchi e tenendo conto dell'osservazione 2.1.1 si ottiene:

$$P(s) = \frac{I_p(s)}{I_{ref}(s)} = \frac{I_T(s)}{I_{ref}(s)} \cdot \frac{I_p(s)}{I_T(s)} = -\frac{sM_{Tp}}{(sL_p + R_p)(sL_T + R_T)} \quad (2.1.17)$$

Che vista come schema a blocchi diventa:

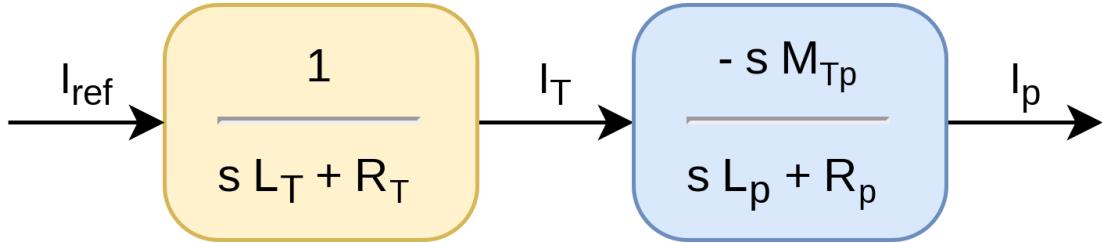


Figura 2.5: Schema a Blocchi Corrente di Plasma

Essendo in uscita il segno delle correnti inverso dall'ingresso, possiamo eliminare questo fastidioso problema di segno in molti modi, quello che si è usato da qui in avanti nella realizzazione della tesi e degli esperimenti, è stato banalmente invertire i poli del Secondario del trasformatore, ottenendo quindi come funzione di trasferimento:

$$P_{pos}(s) = -P(s) = \frac{sM_{Tp}}{(sL_p + R_p)(sL_T + R_T)} = \frac{-I_p(s)}{I_{ref}(s)} \quad (2.1.18)$$

2.1.6 Misura del campo Elettrico del Plasma come indice per la Corrente

Riprendendo l'equazione della corrente di Plasma 2.1.12, abbiamo che il campo elettrico indotto sul Plasma, ovvero la F_{em} , dipende direttamente dallo stato della corrente di Plasma I_p .

Da essa abbiamo che per I_p costanti, anche la F_{em} è costante.

Nel caso del trasformatore si potrebbe fisicamente misurare la corrente di plasma, simulata dalla corrente sul secondario, ma nel caso di un impianto reale, risulta evidente che è impossibile.

Proprio per questo, come descritto da Cianfarani, «**MAGNETIC DIAGNOSTICS FOR FUSION MACHINES**», a pagina 12, si usa come indice di misura Tensione di giro V_{loop} ($\Rightarrow F_{em}$ nel nostro esperimento).

Perciò il circuito reale usato nell'esperimento della tesi, comprensivo di misure è:

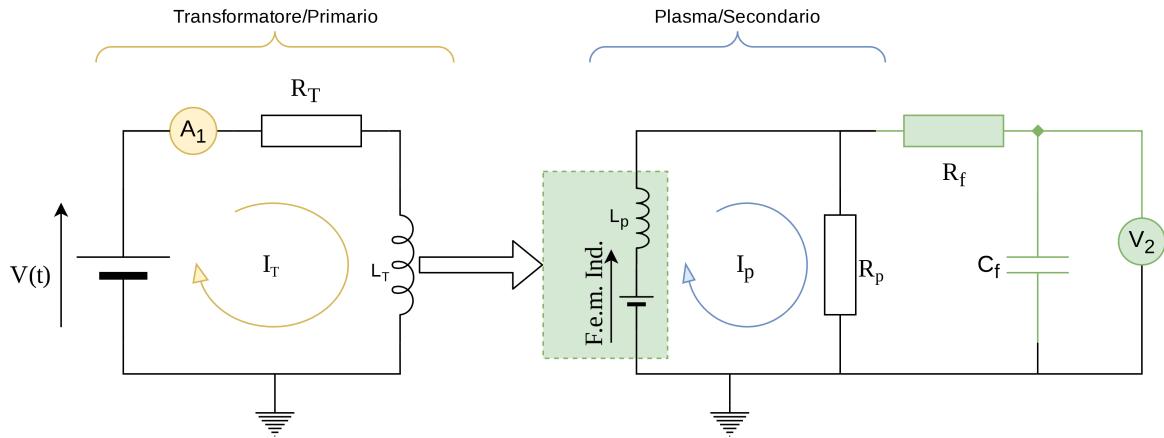


Figura 2.6: Circuito di misura

Abbiamo quindi, tra ingresso e uscite i parametri di:

- I_T misurata dal sensore di corrente A_1 **Trasduttore di Corrente**.
- F_{em} misura dal voltmetro V_2 (che coincide con un pin analogico del **μControllore**) e che diventa un indice della corrente I_p quando essa è costante (eq 2.1.12).
- $V(t)$ generata in base alla legge di controllo, e attuata mediante PWM (Redazione, **Nozioni di PWM**) dal driver di corrente **Driver di Corrente - IBT-2**.

2.2 Trasduttore di Corrente

Come detto nell'introduzione, l'obiettivo della tesi è di controllare la corrente sul secondario usando il campo elettrico del secondario, la misura della corrente scorrente sul primario in tale ottica non sarebbe una misura di interesse. Essendo però un lavoro di ricerca, si è preferito poter misurare la corrente effettivamente circolante nel Primario del trasformatore, così da poter meglio interpretare i dati di misura senza ambiguità.

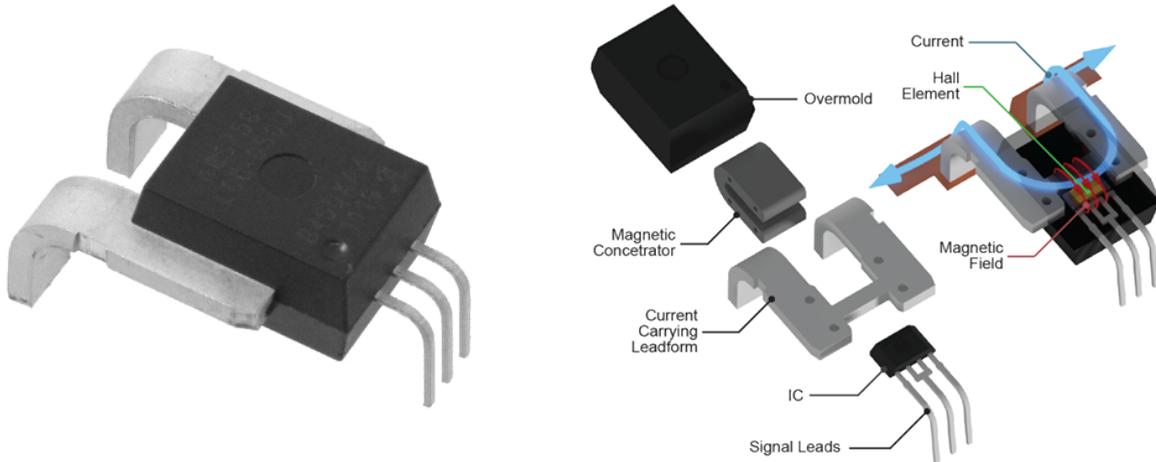


Figura 2.7: Sensore di Corrente

2.2.1 Sensore scelto

Per misurare la corrente del primario, si è posizionato in serie allo stesso il sensore di Corrente Allegro, *High-Precision Linear Current Sensor*.

La famiglia di sensori ha in generale le seguenti caratteristiche:

Bandwidth:	120 kHz
Output rise time :	4.1 μ s
Ultralow power loss:	100 μ Ω Resistenza Interna
Single supply operation	4.5 to 5.5 V
Extremely stable output offset voltage	

Inoltre, delle tante varianti presenti, si è scelto di usare la ACS770LCB-100B-PFF-T, le cui caratteristiche chiave sono:

Primary Sampled Current:	$\pm 100 \text{ A}$
Sensitivity Sens (Typ.)	20(mV/A)
Current Directionality	Bidirectional
T_{OP}	-40 to 150 ($^{\circ}\text{C}$)

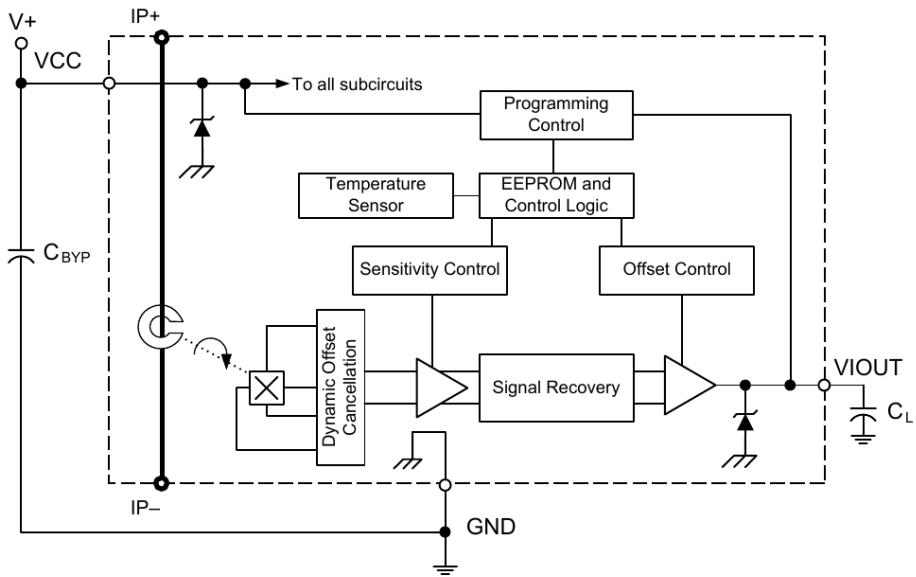
L'ampio margine di misura, e la robustezza alle variazioni di temperatura rendono rendono il dispositivo perfetto per misurare i nostri esperimenti.

L'ampio margine di misura permette di comprendere tutti i possibili valori di corrente ottenibili in laboratorio, rendendo il prototipo adatto a scopi futuri.

2.2.2 Criticità

Unico punto dolente è il suo principio di funzionamento: essendo il sensore basato su un l'effetto Hall, ovvero una misura diretta del campo magnetico indotto dalla corrente nel conduttore, è importante tenere distante il sensore dal Trasformatore Centrale che nei suoi momenti di massimo flusso di corrente, genera ovviamente un campo magnetico non indifferente.

2.2.3 Funzionamento Interno



Functional Block Diagram

Figura 2.8: Schema a Blocchi

Tra le caratteristiche chiave dell'Allegro, *High-Precision Linear Current Sensor*, troviamo il disaccoppiamento fisico tra la corrente da misurare e il circuito di misura.

Questa caratteristica chiave, garantisce la salvaguardia del circuito logico a valle, dai possibili eventi catastrofici a monte.

Esso è inoltre fornito di sensori di temperatura e sistemi di *Signal Recovery* che permettono all'hardware stesso di compensare parzialmente *Non-Linearità* termiche e nella misura dell'effetto Hall, ottenendo un output assimilabile a un segnale lineare:

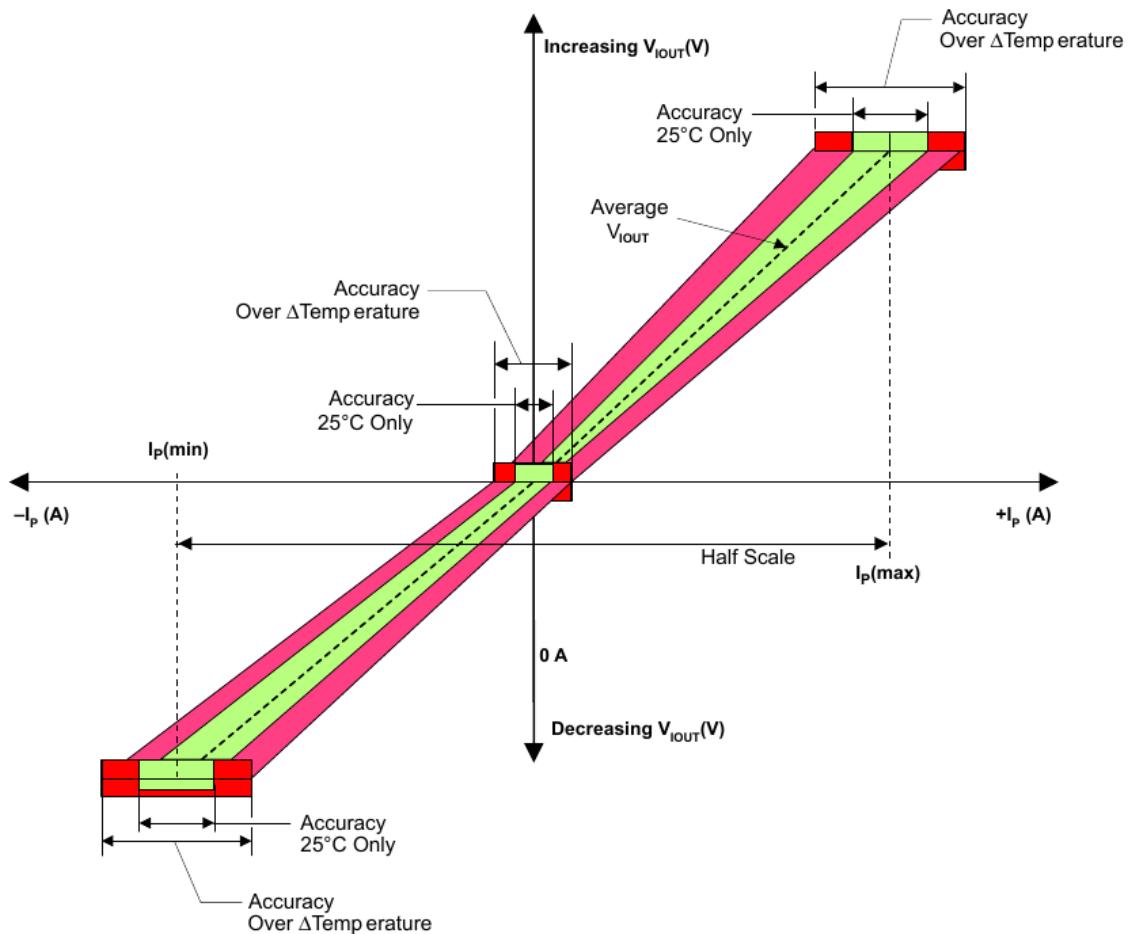


Figura 2.9: Sensibilità

Come si può vedere dal grafico, gli errori sono tanto più marcati quanto maggiore è la corrente da

misurare, ma leggendo dal datasheet abbiamo che questo errore, che dipende si fortemente dalle temperature di esercizio dell'esperimento, non è mai, neanche negli esperimenti più sfortunati, superiore al $\pm 2\%$.

Anzi, alle temperature $\approx 25^\circ$, si mantiene contenuto tra $\pm 0.5\%$.

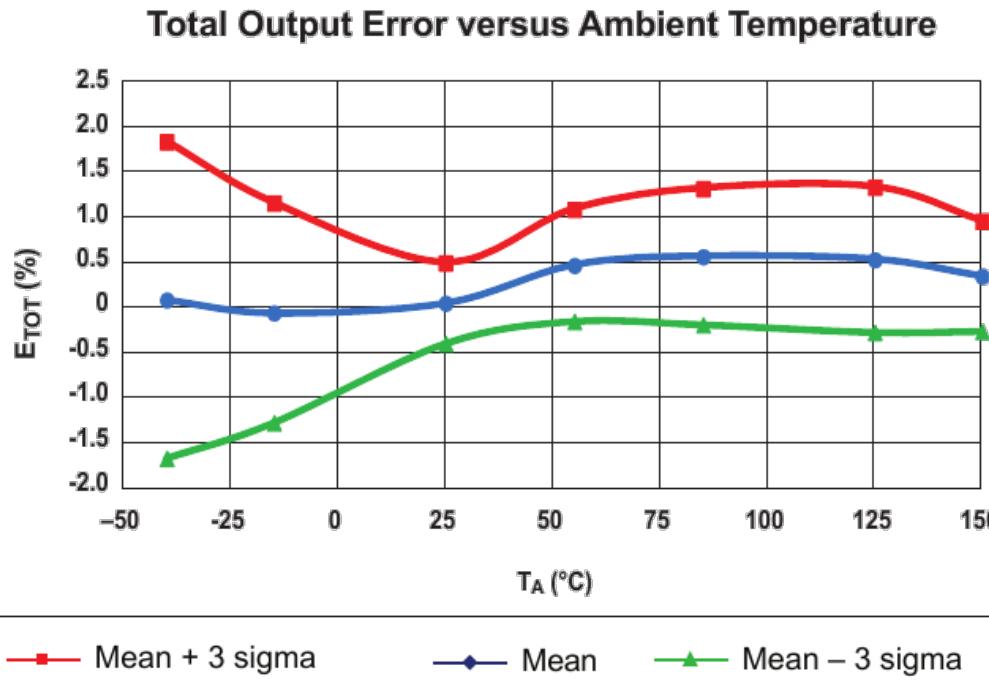


Figura 2.10: Temperatura/NonLinearità

2.2.4 Connessione elettrica

La connessione del sensore è particolarmente semplice, richiedendo esternamente solo un alimentazione stabilizzata e portando subito in uscita la misura.

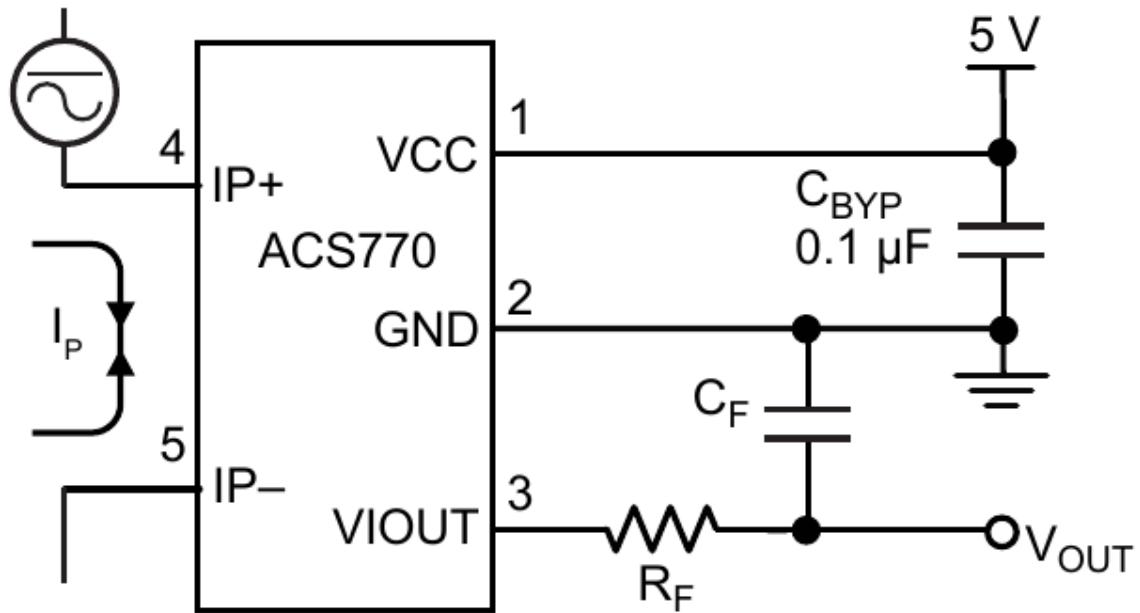


Figura 2.11: Collegamento dal Datasheet

Rispetto allo schema proposto dal datasheet, però, si è anche deciso di omettere il filtro passa-basso sulla **VIOUT**, questa scelta è stata presa per minimizzare il più possibile ritardi di misura della corrente istantanea, poiché le dinamiche del sistema sul secondario, come visto, sono di tipo derivativo, e quindi estremamente rapide.

2.2.5 Misura

Il transduttore, misura della corrente sotto forma di tensione, la quale varia in base alla **Sensibilità** del modello in uso. Avendo noi il ACS770LCB-100B-PFF-T, il datasheet riporta:

Primary Sampled Current:	$\pm 100 \text{ A}$
Sensitivity Sens (Typ.)	$20(\text{mV/A})$
Current Directionality	Bidirectional

Ciò implica che la corrente misurata, è calcolabile come:

$$I_{read} = \frac{V_{Read}[V]}{V_{sense}[V/A]}$$

Rimozione Offset Essendo però il device ad alimentazione singola (0–5V), ma la corrente misurabile Bi-direzionale, sorge la necessità di spostare gli 0A a una tensione superiore agli 0V.

Il datasheet riporta che $V_{offset} = \frac{V_{cc}}{2} \approx 2.5\text{V}$. Da cui deriva che la vera misura di corrente è:

$$I_{read} = \frac{V_{Read} - V_{offset}}{V_{sense}} \frac{V}{[V/A]} \quad (2.2.1)$$

Al fine di poter misurare l'offset effettivo, durante il set-up viene eseguito a esperimento fermo una misura dell'offset attuale, usando la [Macro per calcolo Offset](#).

Il risultato della computazione, oltre ad essere usato nel controllo è inviato al computer per la post elaborazione dei dati nei grafici.

Analisi Sensibilità Usando nell'esperimento un ADC a 10Bit con tensione di riferimento a 5V, abbiamo che la massima sensibilità del μC , ovvero il suo bit meno significativo è pari a:

$$V_{step} = \frac{V_{cc}}{2^{10} - 1} = 4,887\text{mV}$$

Il che equivale a una **Sensibilità di Corrente** del $\mu\text{Controllore}$ pari a:

$$I_{step} = \frac{V_{step}}{V_{sense}} = 244,379\text{mA}$$

Il ché rende la misura buona per osservare cosa stia accadendo, ma sicuramente non sufficientemente densa da poterla usare come parametro ingresso di controllo.

2.3 Driver di Corrente - IBT-2

Per l'attuazione del controllo di corrente nella bobina primaria del trasformatore, è stato usato il driver di corrente Handsontec, [BTS7960 High Current 43A H-Bridge Motor Driver](#).

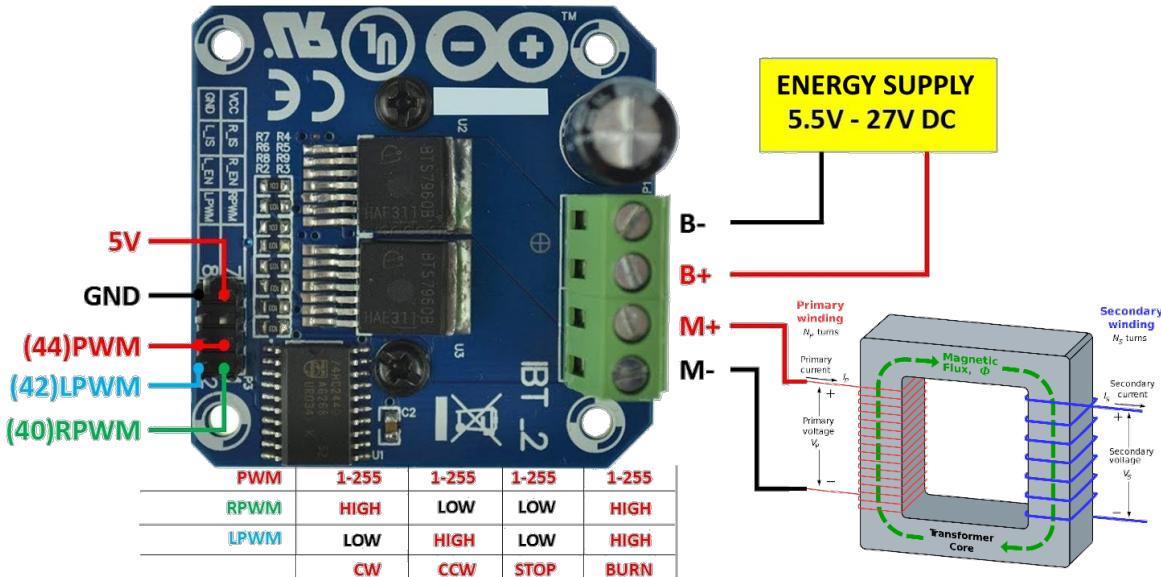


Figura 2.12: IBT-2 TopView.

Esso non è un comune Ponte-H integrato: per poter gestire potenze superiori è stato costruito usando 2 Half-Bridge (Infineon, [BTS7960B High Current PN Half Bridge NovalithIC T](#)) collegati assieme mediante una opportuna logica per ricreare un normale Ponte-H.

Questa scheda in particola ha prestazioni interessanti per gli scopi di questa tesi, i principali sono elencati di seguito:

Power Input Voltage:	6 – 27 V
Peak current:	43 A
Massima Frequenza di PWM:	25 kHz
Protezione Sovra Tensioni	
Disaccoppiamento Ingresso di Potenza/Logica di controllo	

Di particolare interesse per l'esperimento è proprio la corrente di picco gestibile: avendo le dinamiche del sistema tempi inferiori ai 5 secondi, poter reggere correnti di picco così elevate rende la scheda perfetta per i nostri scopi.

2.3.1 Schema Elettrico

Al suo interno il driver è composto da 2 Half-Bridge Infineon, *BTS7960B High Current PN Half Bridge NovalithIC T*, connesse secondo lo schema:

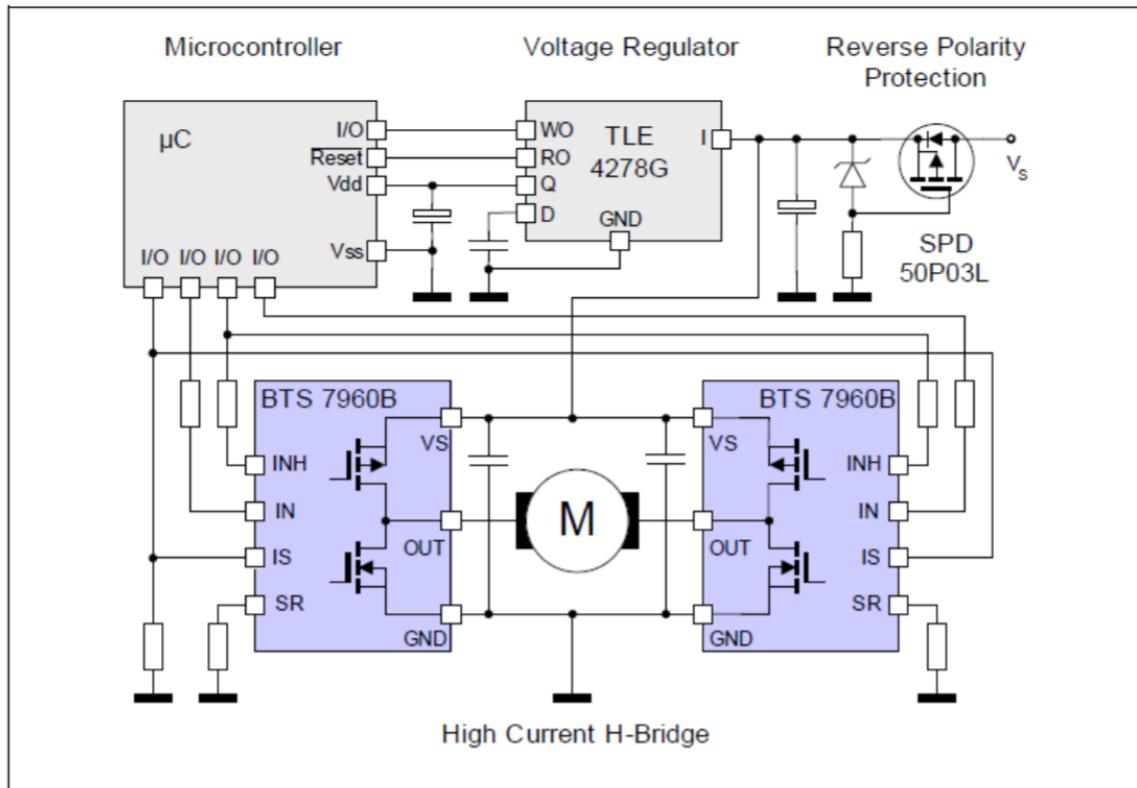


Figura 2.13: IBT-2 Schema Elettrico

Il μ C è protetto dal carico connesso all'interno del BTS7960b, lasciando al μ C solo il compito di controllare i segnali.

2.3.2 Connessione di Controllo

Il driver permette 2 modalità di funzionamento:

Doppio PWM Modalità operativa che richiede l'uso di 2 PWM

Ciascun PWM controlla uno dei 2 Half-Bridge, e per evitare di bruciare i driver devono essere controllati singolarmente, il vantaggio di questa configurazione è la possibilità di usare 2 frequenze di controllo diverse.

Singolo PWM Modalità operativa classica di un normale Ponte-H

In questa modalità, la porta nand presente sulla scheda attua la logica di controllo opportuna per governare i 2 Half-Brige come fossero un normale Ponte-H.

Per il nostro esperimento si è scelto di usare il collegamento **Singolo PWM** così da evitare spiacevoli sorprese e avere il PWM di controllo sempre sincronizzato.

2.3.3 Benchmark del Driver

Il driver sulla carta à buone prestazioni, ma non sono descritte le sue *Non-Linearità*, per farle risaltare si sono effettuati 2 esperimenti usando differenti input di controllo:

1. Onda Triangolare Periodica

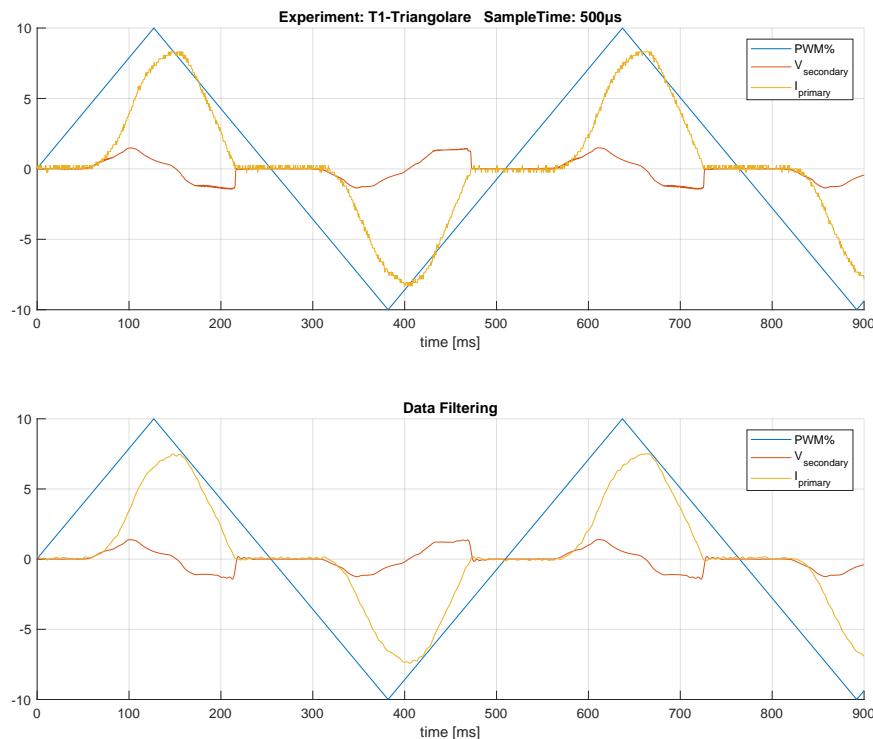


Figura 2.14: Onda Triangolare

Dead-Zone Inferiore L'onda triangolare si presta bene per far risaltare la problematica della Dead-Zone Inferiore, infatti in tutti gli intorni in cui il segnale passa per 0, è possibile vedere come la corrente non vari minimamente, è però possibile notare che i 2 lati non sono simmetrici tra di loro, questo è facilmente spiegabile dal fatto che il primo ha una condizione iniziale $\neq 0$ e di fatto stiamo ancora osservando l'esaurimento del transitorio, la soglia di Dead-Zone Inferiore è quindi calcolata vedendo il primo valore di PWM per cui il sistema risponde a destra degli 0.

2. Onda Trapezoidale Periodica

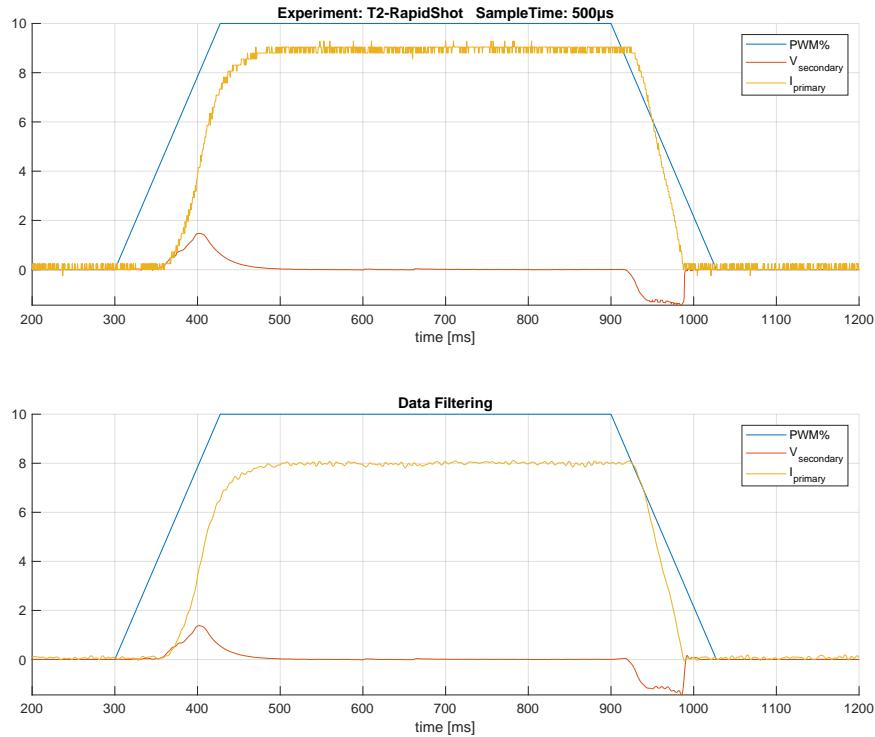


Figura 2.15: Onda Trapezoidale

Dead-Zone Superiore Con questo secondo segnale, si vuole mettere in evidenza il ritardo durante la discesa della rampa, pari a circa 20ms (*guarda 900ms*), questo ritardo è in realtà dovuto da una seconda Dead-Zone presente però ai Duty-Cycle alti del PWM.

Capitolo 3

Architettura di Sistema

3.1 Architettura ad alto livello

Il progetto finale ha come obiettivo la realizzazione di un architettura di controllo per le bobine poloidali presenti nei reattori tokamak.

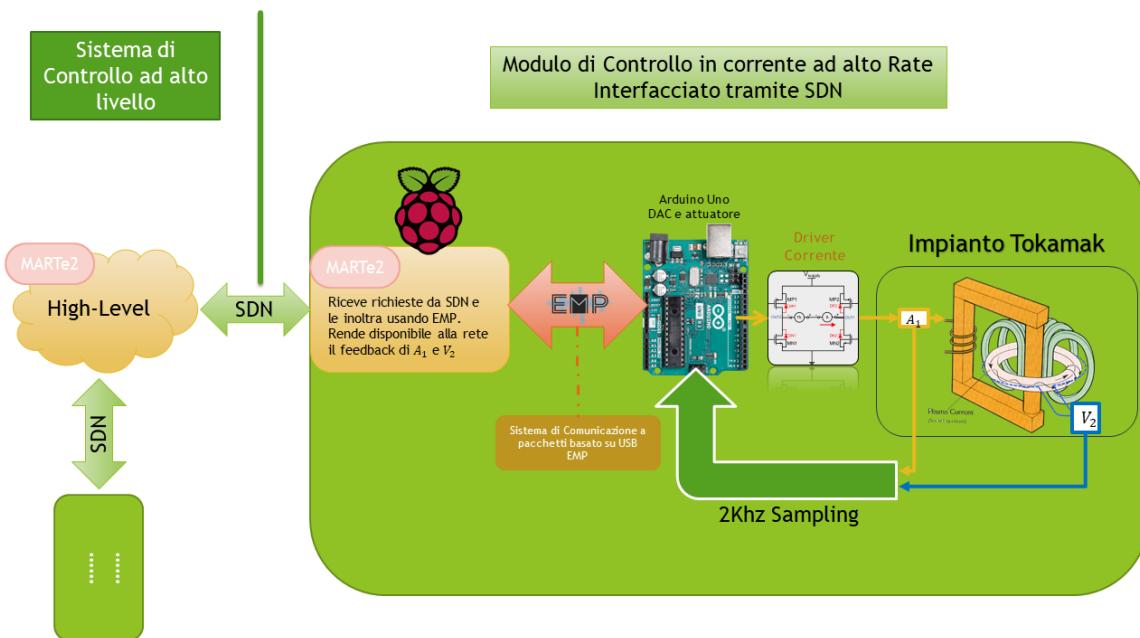


Figura 3.1: Architettura di controllo

Lo schema proposto realizza l'obiettivo è controllare una singola bobina, il progetto finale prevederà la ripetizione in serie del medesimo schema per il numero di bobine necessarie.

Dallo schema risulta evidente che tutti i componenti visti nel capitolo "???" si relazionano con lo stesso μ **Controllore**: l'**Arduino Uno**.

Per riportare i dati fuori e ricevere il riferimento da inseguire nella V_2 , è stata realizzata il ??, essa è stata scritta in C++ affinché possa essere Cross-Platform.

Il suo compito specifico, in questo progetto, è di mettere in comunicazione l'**Arduino Uno** con un nodo *MARTe2* installato su di una **Raspberry Pi**.

Quest'ultimo nodo ha il compito di mettere in rete il feedback dell'esperimento, e comunicare all'**Arduino Uno** eventuali cambio di riferimento. Questo ultimo tratto è realizzato mediante il protocollo **SDN**, che viaggia sopra Ethernet e dà garanzie Real-time.

Nella sua forma finale, il progetto prevede la riproduzione in serie di questo schema di controllo per arrivare a controllare tutte le bobine poloidali presenti in un tokamak.

EMP - Libreria di Comunicazione Seriale Embedded Message Pack



Embedded Message Pack(EMP) nasce con l'obiettivo di standardizzare un protocollo e creare una libreria C++ basata su classi Template, che permetta di automatizzare e standardizzare tutto il lavoro di programmazione necessario all'invio/ricezione di dei pacchetti dal formato Pre-Concordati tra 2 Device connessi Peer2Peer (Nessuna pretesa di network-ing). (Alfano, *Embedded Message Pack(EMP)*)

Il raggiungimento dei suoi obiettivi, si sposa con la possibilità di supportare altre features interessanti:

Multiple-Package Il protocollo di comunicazione che si è deciso di usare per EMP ha permesso di estendere il suo funzionamento e permettere il trasporto, attraverso lo stesso mezzo, di **pacchetti di tipologia e dimensione diversa** all'interno della stessa libreria, evitando al contempo di inviare per ogni pacchetto più byte di quelli strettamente necessario. ⇒ **Alta Efficienza**

Zero Tempo di negoziazione Sempre grazie al protocollo di comunicazione, EMP è adatto ad un uso ‘Streaming’, questo perché non è necessario alcuna fase di sincronizzazione iniziale o durante la trasmissione in caso di perdita di dati, in aggiunta a ciò, EMP è in grado di scartare pacchetti errati in maniera trasparente all’utilizzatore. Tutto questo grazie al protocollo che **Auto-delimita i singoli pacchetti**. ⇒ **Trasparenza Totale**

Responsabilità Le uniche responsabilità a carico degli utilizzatori sono il riempimento dei pacchetti e la definizione degli stessi tra i 2 estremi della comunicazione.

Consistent Overhead Byte Stuffing (COBS)

Il protocollo di comunicazione che permette l’invio di **pacchetti diversi e senza fasi di negoziazione** alla base della libreria è **COBS**(IEEE, *Consistent Overhead Byte Stuffing (COBS)*).

Si tratta di un algoritmo per la codifica di byte, progettato per essere al tempo stesso efficiente e non ambiguo, che permette la definizione di *data-pack frame* **Auto-delimiti**.

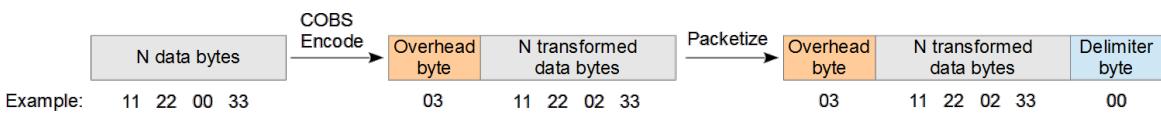


Figura 3.2: Esempio di COBS

3.1.1 Metodo di codifica

L’algoritmo di COBS trasforma una stringa arbitraria di byte, ciascuno dei quali ha un Range di valori da **[0:255]** in una nuova stringa di byte dove però ogni byte va da **[1:255]**. La dimensione della nuova stringa è sempre pari alla dimensione della precedente + 1.

L’obiettivo di questo metodo di codifica è di eliminare tutti i possibili byte **\0** dal pacchetto in maniera reversibile.

Questo processo rende il carattere **\0** (byte zero) ottimo candidato per essere usato come terminatore di stringa durante l’invio, rendendo un pacchetto COBS-Encoded mai ambiguo e sempre

Auto-delimitato.

L'algoritmo di codifica consiste nel:

1. Inserire un byte '\0' all'inizio del pacchetto
2. Individuare tutti gli altri byte '\0'
3. Inserire un byte '\0' alla fine del pacchetto
4. Sostituire tutti gli '\0' con la distanza dal successivo '\0' nella stringa, ignorando l'ultimo

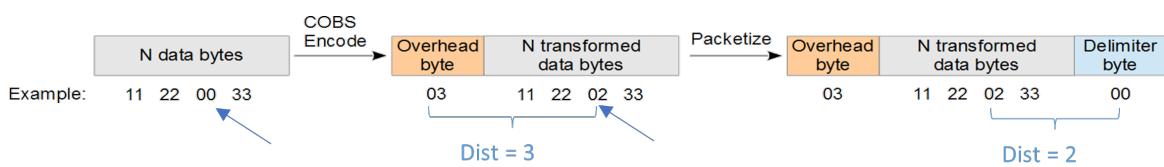


Figura 3.3: Esempio di COBS con distanza

La versione usata per questo progetto, che comunque non ha l'obiettivo di trasmettere quantità infinite di byte, ha il limite di non poter codificare blocchi di byte che hanno una distanza tra 2 '\0' superiore a 255, questo limite è potenzialmente rimovibile usando un algoritmo più sofisticato.

3.1.2 Caratteristiche chiave

Prima caratteristica vincente della libreria è il suo alto grado di adattabilità, essa infatti per poter funzionare richiede solo **2 informazioni critiche** (e altre di contorno per l'allocazione opportuna dei buffer), esse sono i tipi dei pacchetti in **Input** (*pIn*) e in **Output** (*pOut*) ovvero sia delle strutture in C, con i dati organizzati in base al messaggio da trasferire ([esempio di definizione Pacchetto in EMP](#)).

In secondo luogo, la codifica Consistent Overhead Byte Stuffing (COBS), che abbiamo appena visto permette di avere pacchetti **Auto-delimitati**, ciò permette quindi di scambiare pacchetti diversi tra loro (sia per tipologia che per lunghezza) lungo **lo stesso Stream** di dati.

L'unica condizione necessaria è che il destinatario sia capace di determinare la tipologia di contenuto

trasportato nel pacchetto semplicemente leggendolo.

Questo può essere facilmente risolto in più modi:

Lunghezza Univoca Ogni possibile pacchetto ha una lunghezza diversa da tutti gli altri, \Rightarrow la lunghezza implica il contenuto

Aggiunta di un campo Tipologia Aggiungendo all'inizio della trasmissione un *type byte*, ovviamente Pre-Concordato, diventa possibile per chiunque sapere come interpretare il contenuto del pacchetto.

Il metodo più universale è sicuramente l'aggiunta di un campo fisso per il tipo, di cui un esempio è visibile nell'appendice al listato 9.10. ([definizione Pacchetti Multipli](#))

In ogni caso, la codifica Consistent Overhead Byte Stuffing (COBS) aggiunge 2 byte extra al pacchetto che si vuole inviare, e tanto per la codifica quanto per la decodifica in ricezione, il **costo** è sempre pari a **O(n)**.

Vantaggi:

1. Pacchetti **Self-Delimited**
2. Canale **Multi-Packet ready**
3. Protocollo **Senza Negoziazioni**
4. All'utilizzatore è richiesto solo di Pre-Concordare il formato del pacchetto con l'altro lato dello stream
5. Prerequisiti implementativi minimale (mezzo di comunicazione a bytes di tipo *peer2peer* asincrono)

Svantaggi:

1. Aggiunge 2 byte fissi
2. Richiede O(n) elaborazione sia in codifica che decodifica

3.1.3 Integrità dei pacchetti

Per aumentare ulteriormente i campi d'uso e garantire un layer minimale di **integrità** sui pacchetti in transito, la libreria è stata progettata per includere in maniera trasparente anche un check di errore calcolato usando [*Checksum Calculation 8 Bit \(CRC8\)*](#), aggiunto in trasmissione e rimosso in ricezione.

L'aggiunta e il calcolo del CRC8 viene fatta sul pacchetto non ancora *COBS-Encodato*, ciò garantisce la possibilità di inviare il pacchetto a prescindere da quale sia il risultato del CRC8, e viene quindi verificato dopo aver de-*COBS-Encodato* il pacchetto in ricezione.

Questa feature deve essere attiva o disattivata da entrambi i lati della libreria (ne consegue che anche lei deve essere pre-concordata tra i 2 estremi).

La libreria, se attiva, è in grado di capire se il pacchetto ha subito degli errori (ovviamente nei limiti del CRC8) e in tal caso scarta il pacchetto ricevuto in maniera totalmente trasparente all'utilizzatore. L'aver usato COBS come sistema di codifica per la trasmissione, garantisce che la decodifica debba avvenire solo nei byte compresi tra 2 zeri, e se questa decodifica presenta un errore, toglie ogni ambiguità sul da farsi poiché il pacchetto viene scartato e si attende un successivo 0 mentre si memorizzano i byte ricevuti nel frattempo.

3.1.4 Struttura del codice

Lo sviluppo del codice è qui riassunto nel **Class Diagram** fatto in UML, del codice:

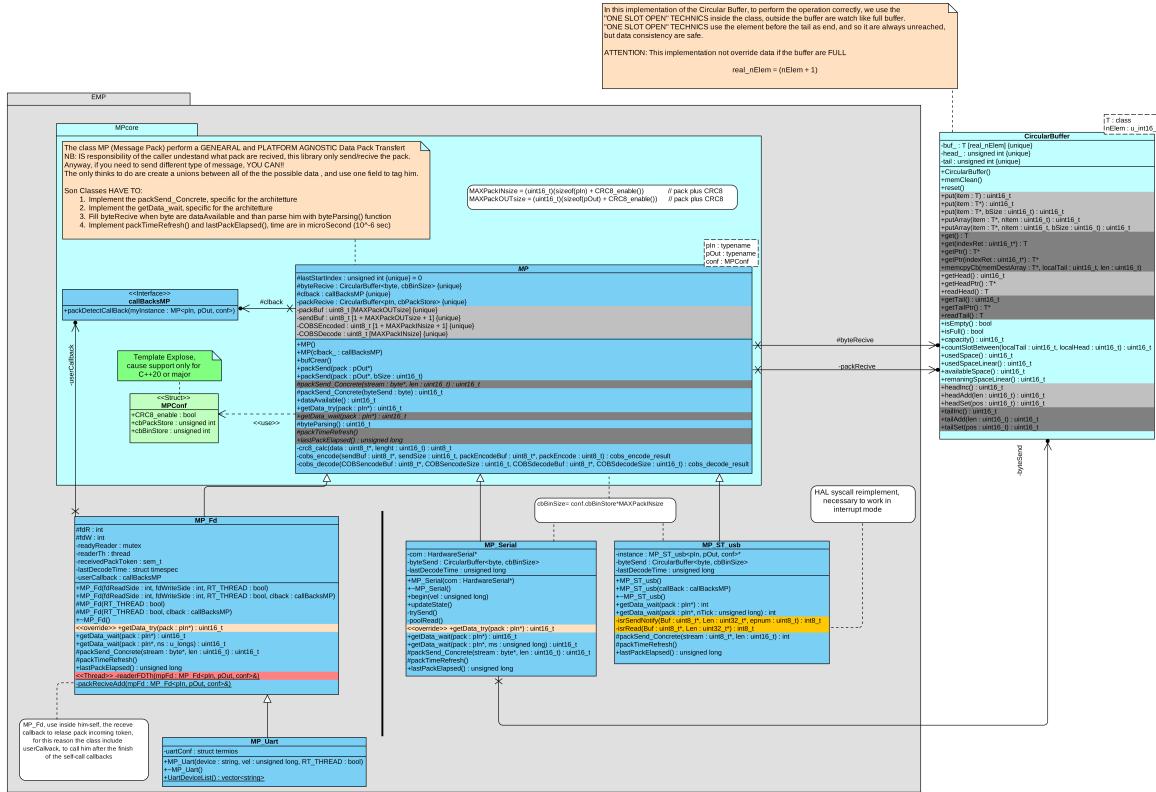


Figura 3.4: Class Diagram UML

Senza entrare eccessivamente nel dettaglio di Embedded Message Pack(EMP), essendo tutto reperibile nella versione più aggiornata all'interno del repository di *Embedded Message Pack(EMP)*, osserviamo che il codice è diviso in 2 macro blocchi (**MPCore**, **MP**) più una classe di supporto per il buffer circolare.

Questa organizzazione del codice è stata pensata per far compilare su tutte le piattaforme di interesse lo stesso *codice attivo* (codifica e decodifica dei pacchetti + accumulo), e demandare le particolarizzazioni dovute alle varie piattaforme o al mezzo di comunicazione usato nel caso specifico a delle classi figlie.

MPCore Il *Codice attivo* è presente all'interno del package **MPCore**. Tutto il codice contenuto in questo package è scritto in C++11 e per essere compilato necessita di un set minimale di librerie standard, sicuramente presenti in ogni piattaforma di sviluppo.

EMP Il package più generico comprende le classi figlie che concretizzano le operazioni di invio e ricezione, facendo poi elaborare i byte al codice che ereditano dalla classe **MP**.

Le classi sono scritte e pensate per funzionare su una piattaforma specifica, e su essa essere ottimizzate.

Se la piattaforma lo concede come nel caso di Linux, è quindi possibili fattorizzare ulteriormente delle funzionalità comuni e scrivere così, via via, sempre meno codice sicuri che quello in comune, se funziona su una classe, deve funzionare anche per l'altra.

Buffer Circolare La classe *CircularBuffer* è una classe di supporto che implementa un buffer circolare con logica "One Slot Open"(Johnston, [Creating a Circular Buffer in C and C++](#)) attraverso una classe template.

Il motivo di questa scelta, molto forte e vincolante, essendo principalmente lei la causa per cui tutte le altre sono anch'esse template, è dovuta alla possibilità di istanziare in fase di compilazione, tutta la memoria richiesta per il funzionamento del programma.

Una simile necessità nasce dal dover compilare la classe anche su schede embedded, le quali, notoriamente, non hanno a disposizione un heap in ram spazioso per la memoria dinamica, e beneficiano nelle prestazioni se in fase di compilazione gli indirizzi di memoria sono fissi.

3.1.5 Logica di Comunicazione

La libreria è pensata per automatizzare la trasmissione e la ricezione di 2 tipologie di pacchetti *Pre-concordati* tra 2 realizzazioni di **MP**, ovviamente non necessariamente sullo stesso dispositivo.

Come riportato in appendice (8.3), la definizione dei pacchetti è molto comoda, e usando un tipo e le `union` è anche possibile gestire la comunicazione di pacchetti diversi tra i 2 lati della comunicazione, basterà invertire l'ordine dei pacchetti tra le 2 classi.

Il riempimento corretto dei dati nel pacchetto, il calcolo della size utile e il riconoscimento in ricezione del pacchetto trasmesso, è tutto a carico dell'utilizzatore della libreria, essa garantisce la corretta ricetrasmissione e di svegliare un ascoltatore al ricevimento di un pacchetto, qualunque sia la sua lunghezza.

3.1.6 Code Flow

Vediamo ora come la libreria si frappone tra 2 device che vogliono comunicare mediante questo sequence diagram:

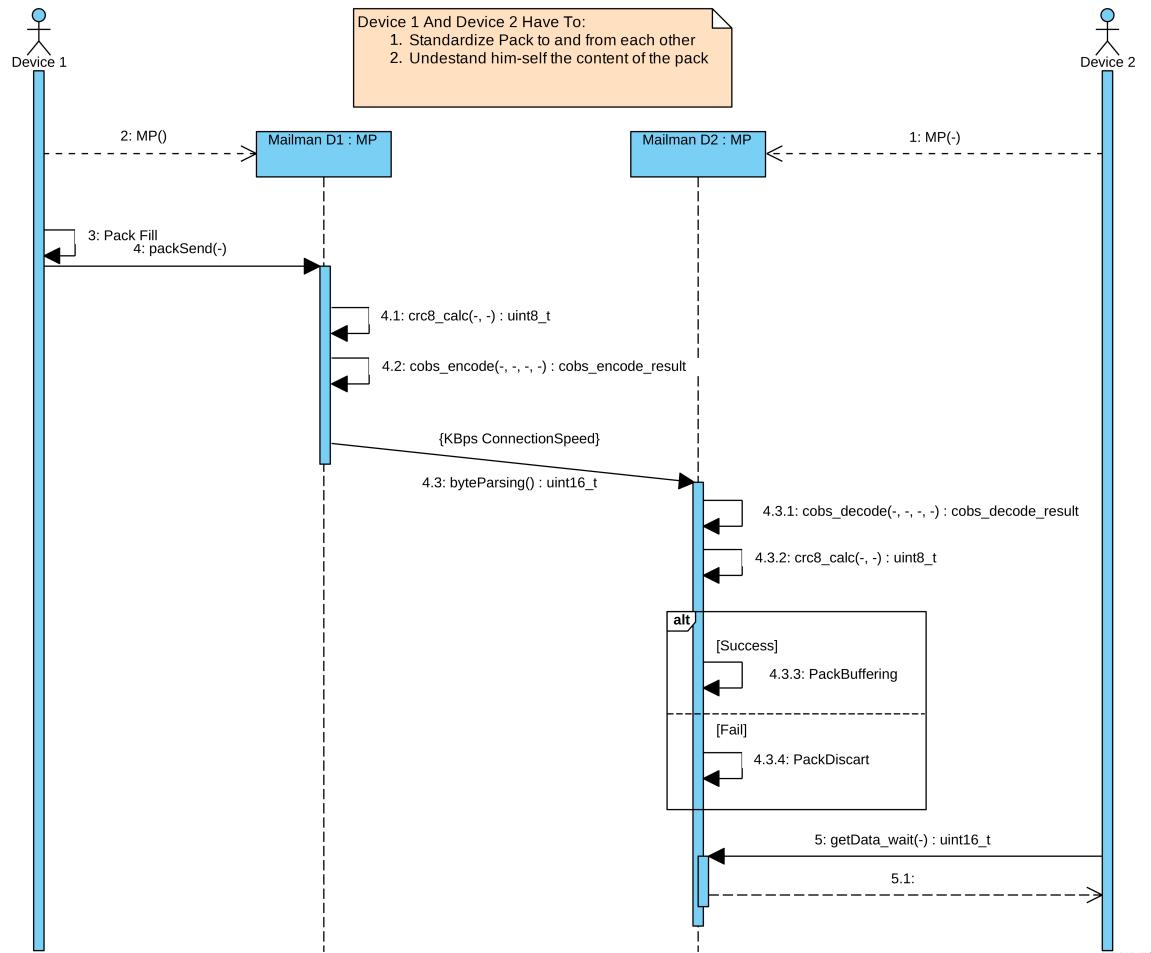


Figura 3.5: Sequence Diagram UML

In questo esempio generico i 2 device non sono definiti, ovviamente nessuno dei 2 può istanziare concretamente una classe **MP** (essendo una classe virtuale), ma come detto prima, tutti i *Codici attivi* sono racchiusi lì dentro.

Come si può osservare, eccetto il riempire i dati, inviarli e attendere che arrivi qualcosa, per gli attori il lavoro finisce subito, internamente alla libreria, invece, a parti invertite, abbiamo il calco-

lo del CRC8 ([Checksum Calculation 8 Bit \(CRC8\)](#)) e la codifica usando COBS (IEEE, [Consistent Overhead Byte Stuffing \(COBS\)](#)), e l'omologo dall'altro lato, dopo aver ricevuto i byte, procede alla de-codifica e check per l'integrità.

3.1.7 Test di Codifica/Decodifica su ogni Device

La serie di passi descritta nel [Code Flow](#) è stato testato su i vari dispositivi per cui la libreria è stata sviluppata usando il seguente schema:

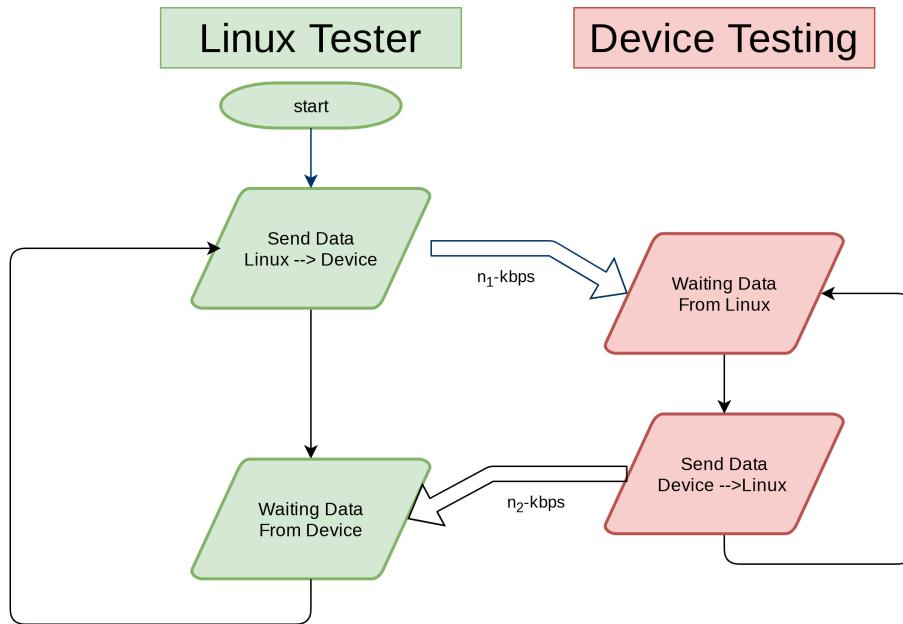


Figura 3.6: Testing Flow

I test vengono avviati da **Linux** e puntano a testare la perdita la correttezza della trasmissione usando Embedded Message Pack(EMP), e il funzionamento di Encoding e Decoding delle classi sulle entrambe le architetture.

Ad ora, sulle 3 Piattaforme di sviluppo (Linux, Arduino, STM32) i test sono stati un pieno successo.

3.2 Online Sampling

Come descritto in figura 3.1, il sistema controlla internamente la corrente, ma comunica con il mondo fuori l'attuale stato della bobina.

Per comunicare i Sample (e ricevere le Reference) è stata infatti sviluppata Embedded Message Pack(EMP)(Sezione 3.1).

Come visto nella sezione "Misura del campo Elettrico del Plasma come indice per la Corrente", il circuito di misura è quello riportato in figura 2.1.6.

E, come per il Trasduttore di Corrente, in realtà anche il voltmetro V_2 possiede un offset a $\frac{V_{cc}}{2}$, aggiunto per poter misurare tanto le correnti positive, quanto quelle negative.

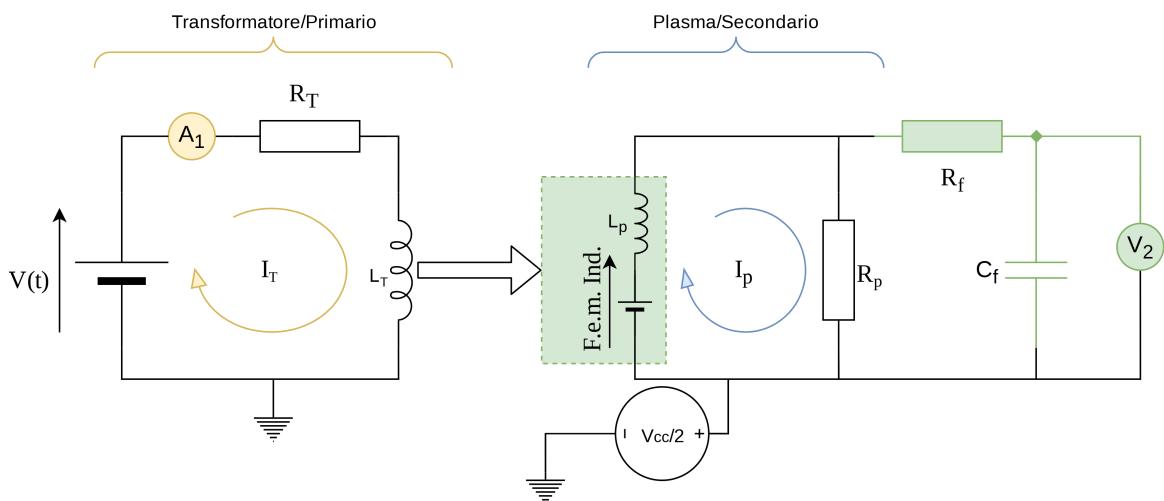


Figura 3.7: Circuito reale con Misure

Controllo e misure avvengono alla massima frequenza che l'Arduino è riuscito effettivamente a gestire, ovvero 0.5ms (2Khz), nei quali campiona lo stato del sistema, genera il nuovo valore del PWM per il controllo di $V(t)$, e invia i dati attraverso Embedded Message Pack(EMP).

3.2.1 Interconnessione μ Controllore \Leftrightarrow Companion

Per raggiungere gli obiettivi descritti nella sezione dell'[Architettura ad alto livello](#), i seguenti pacchetti multipli sono stati concordati tra le 2 parti:

```

1 // #####
2 // ##### Companion to Arduino #####
3 // #####
4 struct newRef {
5     int16_t newRef;
6 } __attribute__((packed));
7
8 struct setUpPackAsk {
9     int8_t padding;
10} __attribute__((packed));
11
12 enum LinuxSendType : uint8_t { newRefType, askType };
13
14 struct _packLinux2Ard {
15     LinuxSendType type;
16     union {
17         struct newRef ref;
18         struct setUpPackAsk ask;
19     };
20} __attribute__((packed));
21typedef struct _packLinux2Ard packLinux2Ard;

```

Listing 3.1: Pacchetti Companion \Rightarrow μ Controllore

```

1 // #####
2 // ##### Arduino to Companion #####
3 // #####
4 struct sample {
5     int16_t pwm;
6     int16_t V2_read;
7     int16_t Isense_read;
8     int16_t err;
9 } __attribute__((packed));
10
11 struct setUpPack {
12     int16_t V2_mean;           // Adc read
13     int16_t Isense_mean;      // Adc read
14     int16_t dt;                // Time in us (10^-6)
15 } __attribute__((packed));
16 enum ardSendType : uint8_t { sampleType, setUpPackType };
17
18 struct _packArd2Linux {
19     ardSendType type;
20     union {
21         struct sample read;
22         struct setUpPack setUp;
23     };
24} __attribute__((packed));
25typedef struct _packArd2Linux packArd2Linux;

```

Listing 3.2: Pacchetti μ Controllore \Rightarrow Companion

Essi permettono al Companion di conoscere tutto quello che sta succedendo nel μ C, con un piccolo ritardo dovuto alla trasmissione ed eventuali ritardi interni (nel caso di Linux dovuti allo scheduler).

Le informazioni riguardanti offset e parametri dell'esperimento (`struct setUpPack`), sono raccolti a macchina spenta, all'accensione della scheda nel [Loop di Controllo](#).

Successivamente la scheda evolve per tic di 0.5ms, dove nel tempo morto resta in ascolto di eventuali pacchetti di richiesta da parte del Companion all'interno del [Loop di Controllo](#).

3.2.2 Storage su file delle informazioni

Le informazioni ricevute dal Companion, vengono salvate all'interno di un file di testo contenente 2 tabelle, disposte una sotto l'altra.

La prima delle 2, tabula i dati del pacchetto `struct setUpPack`, il quale risulta utile per poter interpretare i dati successivamente.

$V_{2_{mean}}$	$I_{sense_{mean}}$	dt
510	510	500

Tabella 3.1: Salvataggio di "struct setUpPack"

La seconda invece è lo streaming dei dati `raw`, ottenuti dalla scheda: Essi sono salvati sotto forma

PWM	$V_{2_{read}}$	$I_{sense_{read}}$	e
0	509	510	1
0	509	511	1
...
0	509	510	59
44	510	510	36
48	518	510	28
53	514	510	32
59	513	510	33
...

Tabella 3.2: Salvataggio di "struct sample" $\forall dt$

di testo ascii normale, separando i campi con un '`tab`'.

3.3 Post Elaborazione con Matlab

I dati salvati su hardisk, come visto della sezione [Storage su file delle informazioni](#), vengono presi in ingresso da Matlab.

3.3.1 Conversioni Dati

3.3.2 Creazione dei grafici e Filtraggio

Capitolo 4

Modello teorico di Controllo

Se lo si desidera, utilizzare questo spazio per inserire un breve riassunto di ciò che verrà detto in questo capitolo. Inserire solo i punti salienti.

4.1 Controllo a Errore Nullo

4.2 Simulazione Qualitativa su Simulink

Capitolo 5

Sviluppo Controllo reale

Se lo si desidera, utilizzare questo spazio per inserire un breve riassunto di ciò che verrà detto in questo capitolo. Inserire solo i punti salienti.

5.1 Codifica del controllore

5.2 Tuning delle costanti

5.3 Esperimenti

Capitolo 6

Conclusioni e sviluppi futuri

Inserire qui le conclusioni trovate con la tesi, ed eventualmente eventuali idee per sviluppi futuri.

Capitolo 7

Software, Toolchain, Strumenti

Per la realizzazione di questa tesi sono stati usati:

Appendice A

Arduino Code

8.1 Set-up Registri

8.1.1 Tic Timer

```
1 void periodicTask(int time) { // time in micro secondi
2     // PWM pin Disable, motalita CTC(pt1)
3     TCCR2A = (0x0 << COM2A0) | (0x0 << COM2B0) | (0x2 << WGM20);
4     // CTC(pt2), Prescalere 256
5     TCCR2B = (0 << WGM22) | (0x6 << CS20);
6     // T_ckclock * Twant / Prescaler = valore Registro
7     OCR2A = (int)(16UL * time / 256);
8     TIMSK2 = (1 << OCIE2A); // attivo solo l'interrupt di OC2A
9 }
```

Listing 8.1: Tic Timer

Questa funzione imposta il TIMER2 in modalità Fast PWM, ovvero che si resetta quando arriva al conteggio finale, e calcola il valore da mettere nel registro affinchè il conteggio sia il più vicino possibile a tempo desiderato

8.1.2 Frequenza PWM

```
1 enum pwmFreq: char {
2     hz30, hz120, hz490, hz4k, hz30k
3 };
4
5 void setMotFreq(pwmFreq freq) {
6     // TCCR0B is for Timer 0
7     #define myTimer TCCR0B
8     switch (freq) {
9         // set timer 3 divisor to 1024 for PWM frequency of 30.64 Hz
10        case hz30:
11            myTimer = (myTimer & B11111000) | B00000101;
12            break;
13        case hz120:
14            // set timer 3 divisor to 256 for PWM frequency of 122.55 Hz
15            myTimer = (myTimer & B11111000) | B00000100;
16            break;
17        case hz490:
18            // set timer 3 divisor to 64 for PWM frequency of 490.20 Hz
```

```

19     myTimer = (myTimer & B11111000) | B00000011;
20     break;
21     case hz4k:
22     // set timer 3 divisor to 8 for PWM frequency of 3921.16 Hz
23     myTimer = (myTimer & B11111000) | B00000010;
24     break;
25     case hz30k:
26     // set timer 3 divisor to 1 for PWM frequency of 31372.55 Hz
27     myTimer = (myTimer & B11111000) | B00000001;
28     break;
29     default:
30     setMotFreq(hz4k);
31     break;
32   }
33 #undef myTimer
34 }
```

Listing 8.2: Frequenza PWM

Mediante questa funzione si modifica il valore del Prescaler per il TIMER 0, modificando la velocità di conteggio si ottiene un PWM con una periodo, e quindi frequenza, che varia.

Offset Calculation

```

1 #define offsetCalc(pin,n) \
2 ({ \
3   long read = 0; \
4   for (int i = 0; i < 1 << n; i++) { \
5     read += analogRead(pin); \
6     delay(1); \
7   } \
8   read = read >> 5; \
9   read; \
10 })
```

Listing 8.3: Macro per calcolo Offset

La macro misura con una distanza temporale di 1 ms 2^n -volte, la misura del Pin, e successivamente ne fa la media eseguendo un revers shift, questa tecnica di divisione semplicemente minimizza il tempo di calcolo, poiché ogni revers shift equivale a una divisione per 2, e si ha l'effetto di dividere per $\frac{read}{2^n}$ in $O(n)$ con un operazione per il processore molto semplice.

8.2 Generatore di Segnale

Per generare i segnali di controllo in Feed-Forward usati nel sistema, sono stati usati 2 diversi livelli di programmazione.

Un primo livello segnali di base, definiti su tutto \mathbb{R} , e usabili a piacere, e dei segnali composti e periodici da mandare durante l'esperimento. Tutti i segnali sono pensati per andare da -100% <-> 100%, è compito dell'attuazione eliminare le deadzone e traslare il controllo al valore più opportuno

8.2.1 Segnali Base

Rampa

```

1 int ramp(uint64_t t, int vStart, uint64_t tStart, int vEnd, uint64_t tEnd) {
2     // Saturazione
3     if (t < tStart)
4         return vStart;
5     else if (t > tEnd)
6         return vEnd;
7     // Retta
8     unsigned int dt = t - tStart;
9     return vStart + int((vEnd - vStart) / float(tEnd - tStart) * dt);
10 }
```

Listing 8.4: Rampa Saturata

La rampa è descritta come una retta nell'intervallo di interesse, saturata prima e dopo il tempo desiderato

$$RampaSat(t) = \begin{cases} v_{start} + \frac{v_{end}-v_{start}}{t_{end}-t_{start}} * (t - t_{start}) & \forall t \in [t_{start}, t_{end}] \\ v_{start} & t < t_{start} \\ v_{end} & t > t_{start} \end{cases}$$

8.2.2 Segnali Composti

Onda Triangolare

```

1 int triangleSignal(uint64_t t, int msQuartPeriod) {
2     static uint64_t startTic = 0;
3     int dTic = t - startTic;
4     int pwm = 0;
5     if (dTic < ticConvert(msQuartPeriod))
6         pwm = ramp(dTic, 0, 0, 100, ticConvert(msQuartPeriod));
7     else if (dTic < (ticConvert(msQuartPeriod) * 3))
8         pwm = ramp(dTic, 100, ticConvert(msQuartPeriod), -100, ticConvert(msQuartPeriod)
9             * 3);
10    else if (dTic < (ticConvert(msQuartPeriod) * 4))
11        pwm = ramp(dTic, -100, ticConvert(msQuartPeriod) * 3, 0, ticConvert(msQuartPeriod
12            ) * 4);
13    else {
14        pwm = 0;
15        startTic = t;
16    }
17    return pwm;
18}

```

Listing 8.5: Onda Triangolare Periodica

Onda Trapezoidale

```

1 int rapidShot(uint64_t t) {
2     static uint64_t startTic = 0;
3     int pwmRapidShot;
4     long dTic = t - startTic;
5     if (dTic > t4) {
6         startTic = t;
7         pwmRapidShot = 0;
8         dTic = t - startTic;
9     }
10    if (dTic <= t1) {
11        pwmRapidShot = ramp(dTic, 0, 0, 100, t1);
12    } else if (dTic <= t2) {
13        pwmRapidShot = 100;
14    } else if (dTic <= t3) {
15        // falling ramp
16        pwmRapidShot = ramp(dTic, 100, t2, 0, t3);
17    } else if (dTic <= t4) {
18        pwmRapidShot = 0;
19    }
20    return pwmRapidShot;
21}

```

Listing 8.6: Onda Trapezoidale Periodica

8.3 Codici Controllore

Setup dell'esperimento

```

1 void setup() {
2     mpSerial.begin(2000000);
3
4     memset(&pRead, 0, sizeof(pRead));
5     memset(&pWrite, 0, sizeof(pWrite));
6
7     // Motori
8     setMotFreq(hz30k);
9     mot = new DCdriver(enPwm, inA, inB);
10
11    // Mean find
12    pMean.V2_mean = offsetCalc(V2, 5);
13    pMean.Isense_mean = offsetCalc(Isense, 5);
14    pMean.dt = dtExperiment;
15
16    // Ctrl Reference
17    Ctrl.setNewRef(tic, 0);
18
19    // ##### Start Experiment #####
20    mpSerial.bufClear();
21    // Start Delay
22    memset(&pWrite, 0, sizeof(pWrite));
23    pWrite.type = sampleType;
24    delay(1000);
25    periodicTask(pMean.dt);
26    sei();
27 }
```

Listing 8.7: Loop di Controllo

Loop di Controllo

```

1 volatile u32 oldTic = tic;
2 int readData;
3 int pwm;
4 void loop() {
5     mpSerial.updateState();
6     do {
7         readData = mpSerial.getData_try(&pRead);
8         if (readData >= 0) {
9             serialExe(&pRead);
10        }
11    } while (tic == oldTic);
12
13    pWrite.read.V2_read = analogRead(V2);
14    pWrite.read.Isense_read = analogRead(Isense);
15    pwm = mot->drive_motor(Ctrl.ctrlStep(oldTic, pWrite.read.V2_read - pMean.V2_mean));
16    pWrite.read.pwm = pwm;
17    pWrite.read.err = Ctrl.lastErr;
18
19    oldTic++; // Suppose no over time
20    mpSerial.packSend(&pWrite, sizeof(pWrite.type) + sizeof(pWrite.read));
21 }
```

Listing 8.8: Loop di Controllo

Appendice B

EMP Code

9.4 Pacchetti in EMP

9.4.1 Definizione di 2 Pacchetti

Ecco di seguito un esempio di definizione asimmetrica per pacchetti da un Device (Linux), verso un altro (Arduino), e viceversa:

```
1 #include <stdint.h> // To be sure for the footPrint in any platform
2 struct _packLinux2Ard {
3     int16_t num;
4     char buf[20];
5 } __attribute__((packed));
6 typedef struct _packLinux2Ard packLinux2Ard;
7
8 struct _packArd2Linux {
9     int16_t num;
10    char buf[10];
11 } __attribute__((packed));
12 typedef struct _packArd2Linux packArd2Linux;
```

Listing 9.9: esempio di definizione Pacchetto in EMP

Come si può osservare, il sistema non necessita di alcun formato particolare per la definizione dei pacchetti, essi sono delle semplici strutture in C, con l'attributo `__attribute__((packed))` per eliminare Padding e ottimizzare la memoria. Come è possibile notare, le regole non vietano la definizione di 2 pacchetti diversi e asimmetrici.

9.4.2 Pacchetti Multipli

Per definire un pacchetto di pacchetti, il metodo migliore è creare un `enum` per numerare univocamente il tipo (nell'esempio `uint8_t` ma può essere reso maggiore all'occorrenza), e una `union` con tutti i tipi in fila.

Il pacchetto di interesse per Embedded Message Pack(EMP) è `dataSend`, ed è responsabilità dell'utente della libreria riempire correttamente i campi.

```

1 //(DataType1 ,DataType2 , DataType3  typedef  above;
2 enum packTypeSend : uint8_t{
3     DataType1code ,
4     DataType2code ,
5     DataType3code ,
6 };
7 typedef union{
8     DataType1 t1;
9     DataType2 t2;
10    DataType3 t3;
11 } __attribute__((packed)) dataSendUnion;
12
13 typedef struct{
14     packTypeSend type;
15     dataSendUnion pack;
16 } __attribute__((packed)) dataSend;
```

Listing 9.10: definizione Pacchetti Multipli

Per ottimizzare l'invio al minimo strettamente necessario si può quindi usare la variante nel metodo di `packSend(...)`, in cui viene specificata la dimensione:

```

1 //...
2 dataSend.type = DataType2code;
3 dataSend.pack.t2 = /*{data}*/ // fill correctly dataSend
4 //...
5 MP->packSend(&data , sizeof(packTypeSend) + sizeof(DataType2));
6 //...
```

Listing 9.11: definizione Pacchetti Multipli

Appendice C

Matlab Post Elaborazione

Elenco delle figure

2.1	Schema interno di un Tokamak	4
2.2	Trasformatore ideale con Campo magnetico	7
2.3	Circuito Equivalente Bobina/Plasma all'interno di un Tokamak	9
2.4	Circuito Equivalente Bobina/Plasma all'interno di un Tokamak trascurando l'induzione del plasma verso la bobina	10
2.5	Schema a Blocchi della funzione di trasferimento della corrente di Plasma	13
2.6	Circuito reale di misura dell'esperimento	14
2.7	Sensore di Corrente ACS770LCB-100B-PFF-T	15
2.8	ACS770LCB-100B-PFF-T Schema a Blocchi	16
2.9	ACS770LCB-100B-PFF-T Sensibilità rispetto Temperatura	17
2.10	ACS770LCB-100B-PFF-T <i>Non-Linearità</i>	18
2.11	ACS770LCB-100B-PFF-T Schema di collegamento dal Datasheet	19
2.12	Driver Motori IBT-2 TopView & PinOut	21
2.13	IBT-2 Schema Elettrico	22
2.14	Esempio di Onda Triangolare	24
2.15	Esempio di Onda Trapezoidale	25
3.1	Schema finale dell'archittettura di controllo	26
3.2	Esempio di COBS	29

3.3 Esempio di COBS con distanza	30
3.4 Class Diagram UML di EMP	33
3.5 Sequence Diagram UML di EMP	35
3.6 EMP Benchmark Testing Flow	36
3.7 Circuito equivalente del Plasma con l'offset delle Misure	37

Bibliografia

Alfano: Embedded Message Pack(EMP)**EMP**

Emanuele Alfano. *Embedded Message Pack(EMP)*. Repository. 24 Sep 2021. URL: <https://github.com/Alfystar/EMP>.

Allegro: High-Precision Linear Current Sensor**ACS770**

Allegro. *High-Precision Linear Current Sensor*. ACS770LCB-100B-PFF-T. Datasheet. 31 May 2019. URL: <https://www.allegromicro.com/~/media/Files/Datasheets/ACS770-Datasheet.pdf>.

Checksum Calculation 8 Bit (CRC8)**CRC8**

Checksum Calculation 8 Bit (CRC8). Rapp. tecn. Mouser Elettronics, 15 Apr 2019. URL: https://www.mouser.com/pdfDocs/SFM4100_CRC_Checksum_Calculation.pdf.

Cianfarani: MAGNETIC DIAGNOSTICS FOR FUSION MACHINES**MagneticDiagnostics**

Cesidio Cianfarani. «MAGNETIC DIAGNOSTICS FOR FUSION MACHINES». In: *MAGNETIC DIAGNOSTICS FOR FUSION MACHINES*. Associazione Euratom-ENEA sulla Fusione. 11 apr 2013. URL: <https://www.fsn-fusphy.enea.it/Intranet/index.php/seminari-interni/download-file?path=Seminari-Workshop-WIP-Corsi+Interni%2FCorsi+Interni%2FCorso+Rd0+2015%2FSLIDES+%26+AUDIO%2F13+Diagnostiche+Magnetiche+%28Cianfarani%29%2FMagnetic+Diagnostics+2.2+-+Cianfarani.pdf>.

Handsontec: BTS7960 High Current 43A H-Bridge Motor Driver**IBT-2**

Handsontec. *BTS7960 High Current 43A H-Bridge Motor Driver*. Datasheet. 18 Sep 2019. URL: <https://www.handsontec.com/dataspecs/module/BTS7960%20Motor%20Driver.pdf>.

IEEE: Consistent Overhead Byte Stuffing (COBS)**COBS**

IEEE. *Consistent Overhead Byte Stuffing (COBS)*. Rapp. tecn. IEEE, 29 Jun 2002. URL: https://www.ieee802.org/3/efm/public/jul02/copper/oksman_copper_1_0702.pdf.

Infineon: BTS7960B High Current PN Half Bridge NovalithIC T**BTS7960b**

Infineon. *BTS7960B High Current PN Half Bridge NovalithIC T*. BTS7960b. Datasheet. 12 Jul 2016. URL: <https://www.infineon.com/dgdl/bts7960b-pb-final.pdf?folderId=db3a3043156fd5730116144c5d101c30&fileId=db3a30431ed1d7b2011efe782ebd6b60>.

Johnston: Creating a Circular Buffer in C and C++ **CircularBuffer**

Phillip Johnston. *Creating a Circular Buffer in C and C++*. Online Page. 10 june 2021. URL: <https://embeddedartistry.com/blog/2017/05/17/creating-a-circular-buffer-in-c-and-c/>.

Redazione: Nozioni di PWM **modulazionePWM**

Redazione. *Nozioni di PWM*. Online Page. 20 october 2019. URL: <https://www.ingegnerando.it/nozioni-di-pwm/>.

Romero et al.: Real time current profile control at JET **TokamakCircuit**

Jesús Romero et al. «Real time current profile control at JET». In: *Fusion Engineering and Design* 43 (feb. 1998), pp. 37–58. DOI: [10.1016/S0920-3796\(98\)00261-0](https://doi.org/10.1016/S0920-3796(98)00261-0).

Trasformatore Monofase **Transformatore**

Trasformatore Monofase. URL: <https://www.docenti.unina.it/webdocenti-be/allegati/materiale-didattico/34030176>.