POLITECNICO DI TORINO

Facoltà di Ingegneria dell'Informazione
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea Magistrale

Realizzazione del modello ingegneristico di un controllo di assetto magnetico per satelliti modulari

Relatori:
Prof. Leonardo Reyneri
Prof. Claudio Passerone

Candidato:
Davide Masera

Aprile 2011

Sommario

Il crescente interesse in ambito universitario, negli ultimi anni, nel campo della progettazione di satelliti di ridotte dimensioni, ha portato alla definizione dello standard CubeSat, con l'obiettivo di fornire la metodologia di base per la realizzazione e lo sviluppo di picosatelliti.
In tutto il mondo, diversi atenei, tra i quali il Politecnico di Torino, hanno adottato o riadattato tale standard per lo sviluppo di un proprio satellite di piccole dimensioni.

Il primo progetto intrapreso dal Politecnico, denominato PiCPoT (Piccolo Cubo del Politecnico di Torino), prevedeva la realizzazione ed il lancio in orbita di un satellite di forma cubica di 13 cm di lato, i cui elementi costituenti fossero progettati ad-hoc, con lo scopo di acquisire e trasmettere dati, scattare fotografie e testare il funzionamento in ambiente spaziale di componenti COTS.
Il secondo progetto, AraMiS (Architettura Modulare per Satelliti), attualmente in via di sviluppo presso il Politecnico di Torino, costituisce la prosecuzione ed il miglioramento di PiCPoT.
Alla base di AraMiS, oltre agli obiettivi funzionali comuni a PiCPoT, è l'intento di realizzare un satellite modulare, di forma cubica (ma configurabile anche in altre forme), nel quale ogni scheda elettronica sia allo stesso tempo progettabile in modo indipendente dalle altre (consentendo una riduzione di costi e tempi di progettazione), esportabile in altri progetti (o utilizzabile in diverse missioni), specializzata nell'esecuzione dei propri compiti, compatibile con gli altri componenti, sostituibile e modificabile, senza che questo vada ad influire negativamente sul funzionamento complessivo del satellite.
I due tipi principali di moduli standard (detti appunto *tiles*) che costituiscono AraMiS sono:

- ***Power Management Tile***: ha la funzione di generare l'energia elettrica, necessaria ad alimentare i circuiti ed a caricare le batterie, convertendo l'energia solare grazie ai pannelli fotovoltaici presenti sulle facce esterne di AraMiS.
  Questi moduli costituiscono cinque delle sei facce del cubo di base (AraMiS è configurabile anche combinando otto cubi assieme in un'unica struttura, oppure sotto forma di prisma esagonale). Inoltre forniscono l'assetto al satellite, mediante un controllo di assetto attivo costituito da una ruota d'inerzia, un solenoide, un sensore di campo magnetico, un giroscopio e un motore brushless e montato sull'interno delle facce del cubo.
  Nel satellite sono presenti più Power Management tiles, per far sì che il satellite possa essere orientato nella direzione desiderata, allo scopo di direzionare il *Payload* (fotocamera, telecamera) o esporre alla luce solare le diverse facce.

- ***Telecommunication Tile***: ha la funzione di mettere in comunicazione il satellite con la stazione di Terra tramite due canali, alle frequenze di 437 MHz e 2.4 GHz, ed occupa una delle sei facce del cubo di base.
  Inoltre, tale modulo ha il compito di interpretare i comandi di assetto ricevuti da terra ed inviare ai singoli moduli di Power Management i comandi specifici per l'attuazione di tale assetto.

Oggetto di questa tesi è il controllo di assetto magnetico del satellite AraMiS, a partire dalla progettazione, fino alla realizzazione di un prototipo su circuito stampato ed al collaudo del medesimo.
A differenza di PiCPoT, dotato di controllo d'assetto passivo costituito da magneti permanenti, che non permetteva di orientare il satellite nello spazio ma lo vincolava alle linee di campo magnetico terrestre, AraMiS utilizza un controllo attivo che integra una parte inerziale, costituita da una ruota d'inerzia (e relativo motore brushless) e dal sensore giroscopico, e una parte magnetica, comprendente un solenoide utilizzato come attuatore ed un magnetometro. Grazie all'interazione di questi due sottosistemi ed all'effetto combinato di più Power Management Tiles, sarà possibile decidere l'orientamento del satellite inviando il comando dalla stazione di terra.

Requisito fondamentale per il sottosistema di controllo d'assetto (*Magnetic Attitude Subsystem*, codice di progetto 1B22) è la generazione del momento angolare richiesto dal controllo d'assetto centrale (1B2 *Attitude and Orbit Subsystem*), che risiede sulla Telecommunicatiom Tile, riceve i comandi di assetto dalla stazione di terra e li suddivide in molteplici comandi per i sottosistemi di assetto che risiedono sulle diverse Power Management Tiles.
Ciò avviene pilotando un solenoide circolare in rame a 180 spire dal diametro medio di circa 13 cm e sezione del filo di 0.25 mm con un driver bipolare full-bridge. Quando viene percorso da corrente (fino a oltre 0.6 A), esso origina un dipolo magnetico, direzionato lungo il proprio asse, che tende ad allinearsi al campo magnetico terrestre, fornendo una coppia alla tile. L'integrale di tale coppia, nel tempo di pilotaggio necessario, è il momento angolare richiesto. Dal momento che la coppia dipende dalle componenti vettoriali del campo magnetico e dall'intensità della corrente che scorre nel solenoide, occorre che questi valori possano essere misurati frequentemente.

Il progetto hardware del controllo di assetto magnetico si è basato sulla struttura di un preesistente sistema analogo, i cui elementi sono stati riutilizzati, ridimensionati o sostituiti, a seconda dei casi, in modo da soddisfare le attuali specifiche, durante le diverse fasi di progettazione.
Il sistema è costituito da una parte di attuazione, comprendente il solenoide, il suo driver e il circuito di condizionamento per la misura della corrente, e da una parte di misura del campo magnetico, che include un sensore magnetico biassiale e i circuiti di condizionamento per le sue uscite in tensione.
Il software di controllo del sistema, invece, è stato completamente reingegnerizzato, al fine di adattarsi alle nuove strutture e rispettare i vincoli del progetto, realizzandone le funzionalità.

In virtù del concetto di modularità che sta alla base del progetto AraMiS, è stato utilizzato il linguaggio di descrizione e modellizzazione visuale UML (Unified Modeling Language) per sviluppare il progetto in esame e tutti gli altri progetti costituenti i diversi sottoblocchi di AraMiS. Esso si basa su uno standard definito a metà degli anni '90, ma in continua evoluzione, e permette di documentare le specifiche del sistema mediante i *Diagrammi dei Casi d'Uso*, descrivere in forma visiva e completa gli elementi che ne fanno parte con i *Diagrammi delle Classi*, e illustrare il flusso di eventi tramite cui è possibile concretizzarne le specifiche, grazie ai *Diagrammi di Sequenza*.

In questo scritto, verrà effettuata una trattazione preliminare sui concetti fisici che stanno alla base della magnetizzazione del solenoide, del suo utilizzo come attuatore di coppia magnetica e della necessità di misurare frequentemente il campo magnetico terrestre in orbita.
Successivamente, dopo la stesura e la documentazione delle specifiche di progetto sotto forma di diagramma dei casi d'uso, si procederà con la descrizione dei componenti hardware e software che costituiscono il modulo di controllo d'assetto magnetico, utilizzando i diagrammi delle classi.

In seguito, con i diagrammi di sequenza, si descrive il comportamento del sistema durante la messa in atto dei diversi casi d'uso. Quindi, vengono mostrate le funzioni software, implementate a partire da tali diagrammi, eseguite dal microcontrollore presente nella Power Management Tile per realizzare il controllo di assetto magnetico.
Poi, una volta motivate le scelte dei componenti commerciali utilizzati per il progetto, verranno redatti gli schemi elettrici del modulo 1B22, a partire dai quali è stato realizzato il circuito stampato, e sarà fornito un resoconto sull'errore di misura introdotto e sui consumi di potenza del circuito.
Infine, saranno illustrate le procedure di collaudo effettuate sul circuito prototipo del modulo di controllo dell'assetto magnetico, dapprima sotto forma di test basilare del funzionamento dell'hardware e, in un secondo tempo, del software di controllo, interfacciando il modulo con il microcontrollore.

Il collaudo finale è stato effettuato nel laboratorio hardware del Dipartimento di Elettronica del Politecnico di Torino, collegando il circuito prototipo con una apposita scheda di adattamento, sulla quale è stato montato il microcontrollore, dotata di interfaccia JTAG.
Tramite la porta USB del PC ed un apposito debugger per microcontrollori, dotato di uscita JTAG, sono stati collaudati l'abilitazione e la disabilitazione dei circuiti di misura e attuazione, verificandone il funzionamento, e l'invio di comandi da parte del controllo d'assetto centrale al controllo di assetto magnetico, verificandone la corretta esecuzione.
Inoltre, sono state collaudate la misurazione e l'acquisizione dei dati di telemetria del progetto (campo magnetico biassiale e valore di corrente nel solenoide), è stata effettuata la misurazione delle correnti di alimentazione del circuito ed è stato eseguito il test della routine software di controllo degli errori nel sistema.
Il collaudo del circuito ha avuto esito positivo, fatta eccezione per il malfunzionamento di uno degli switch di abilitazione delle tensioni di alimentazione, dovuto ad un errato dimensionamento dei componenti nella progettazione del medesimo (già facente parte, in precedenza, del progetto globale della Power Management Tile), che, pertanto, andrà rivista in futuro.

Indice

Capitolo 1

1. Introduzione

Negli ultimi anni, il mercato dell'aeronautica spaziale, in particolar modo il settore satellitare, ha affrontato una continua crescita, dovuta principalmente al relativo abbassamento dei costi dei lanciatori, necessari al fine di portare i satelliti in orbita. Ciò, unito al contenuto costo di progettazione per satelliti di piccole dimensioni, ha fatto sì che le aziende operanti nel settore e le università di tutto il mondo vedessero crescere considerevolmente il proprio interesse di ricerca in tale campo e sviluppassero indipendentemente i propri satelliti.

Nel 2001 il professor Robert Twiggs, docente alla Stanford University, USA, in collaborazione con lo Space Systems Development Laboratory (SSDL) della Stanford University e la California Polytechnic State University, ha definito e realizzato lo standard per picosatelliti [1] CUBESAT [2], con l'obiettivo di sviluppare un insieme uniforme di specifiche e metodologie che facilitasse la progettazione ed il lancio di satelliti di piccole dimensioni. Cubesat identifica lo standard per picosatelliti di forma cubica con dimensione dei lati di 10 cm e massa massima di 1 Kg, aventi una struttura adattabile al lanciatore POD (Picosatellite Orbital Deployer) .

Altra peculiarità di Cubesat, oltre appunto alla definizione di uno standard, fu l'utilizzo, per la sua realizzazione, di componentistica commerciale a basso costo (COTS, Commercial Off-The-Shelf). Queste due caratteristiche agevolarono la riduzione, rispettivamente, dei costi e tempi di progettazione, e del costo dei componenti utilizzati per i picosatelliti. Inoltre, il peso ridotto (mentre i nanosatelliti arrivano a 10 Kg, i picosatelliti hanno un peso massimo di 1 Kg) permette di impiegare vettori di lancio meno costosi.

Tutto ciò pose le fondamenta per diversi progetti di picosatelliti, realizzati da molte università, nell'ultimo decennio, seguendo lo standard di Cubesat, tra le quali l'Università di Wurzburg in Germania, la Norwegian University of Science and Technology, e, in Italia, l'Università La Sapienza di Roma, l'Università di Trieste e, ovviamente, il Politecnico di Torino, prima con il satellite PiCPoT e attualmente con AraMiS.

1.1. PiCPoT

Il progetto PiCPoT [3][4] (Piccolo Cubo del Politecnico di Torino), intrapreso nel gennaio del 2004, coinvolgeva il Dipartimento di Ingegneria Elettronica ed il Dipartimento di Ingegneria Aerospaziale del Politecnico di Torino e consisteva nel progettare, realizzare e lanciare in orbita un nano satellite sperimentale con l'obiettivo di:

- Verificare il funzionamento e l'affidabilità nello spazio dei componenti COTS;
- Acquisire e trasmettere alla Stazione di Terra diverse immagini e misurazioni effettuate in orbita dai sensori di bordo;
- Scattare fotografie della superficie terrestre;
- Far interagire, attraverso un'attività interdisciplinare, i diversi dipartimenti, docenti, dottorandi e studenti del Politecnico.

PiCPoT è un satellite di forma cubica con lato di 13 cm e massa di 2.5 Kg. Le sei facce esterne sono realizzate in lega di alluminio di tipo 5000 AlMn.

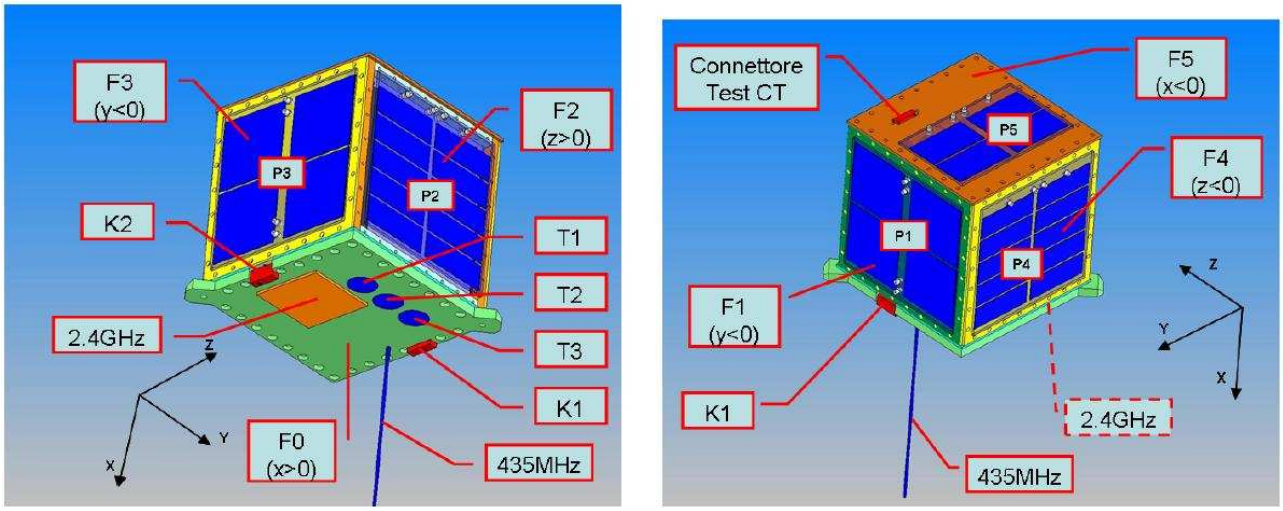


Figura 1.1 Vista esterna del satellite PiCPoT

Come si può vedere in figura 1.1, su cinque di esse sono situati i pannelli solari (P1, P2, P3, P4, P5), in grado di fornire l'alimentazione al satellite, inoltre su una di queste è presente un connettore di test (CT) per collaudare l'elettronica interna anche a satellite montato.

Sulla sesta faccia, invece, vi sono tre telecamere (T1, T2, T3), due antenne (alle frequenze di 437 MHz e 2.4 GHz) utilizzate per comunicare con la Stazione di Terra e due dispositivi denominati "Kill-Switch" (K1, K2), in grado di garantire l'assenza di alimentazione finché il satellite non sia completamente distaccato dal lanciatore.

L'interno del satellite ospita, oltre a due tipologie di batterie (NiCd e NiMH) per alimentare il satellite in assenza di sole, sei schede elettroniche:

- ***Power Supply*** ricarica le batterie convertendo la tensione in arrivo dalle celle fotovoltaiche e controlla dati elettrici e temperatura di queste ultime, delle batterie e dei caricabatterie;
- ***Power Switch*** converte la tensione delle batterie generando le diverse alimentazioni per tutte le altre schede e garantisce il risparmio energetico scollegando l'alimentazione dei componenti inutilizzati;
- ***RxTx*** riceve e trasmette segnali tra stazione di terra e satellite (quindi a *ProcA* e *ProcB*) tramite due canali half-duplex, uno con frequenza di 437 MHz e l'altro di 2.4 GHz, entrambi dedicati alla ricezione dei comandi da terra ed alla trasmissione dei dati da satellite a stazione di terra, così da ridondare la comunicazione e prevenire i guasti;
- ***ProcA*** e ***ProcB*** sono le schede contenenti i due computer di bordo; essi duplicano le stesse funzioni, ma processori e schede sono costruiti con tecnologie e componenti differenti, in modo da costituire una sorta di ridondanza, per evitare, in caso di guasti a taluni componenti, il malfunzionamento dell'una o dell'altra scheda; entrambe acquisiscono i dati di telemetria e comunicano con le schede di *Payload* e *RxTx*; i processori sono alimentati in modo che non operino mai contemporaneamente, per garantire il risparmio di energia;
- ***Payload*** si occupa di acquisire le immagini delle foto/telecamere, comprimerle in JPEG ed inviarle al modulo *ProcA* o *ProcB*, che ne gestisce la trasmissione a terra.

Il funzionamento basilare del sistema e le varie interazioni fra i moduli sopra descritti si possono vedere nello schema a blocchi dell'elettronica di bordo (figura 1.2).

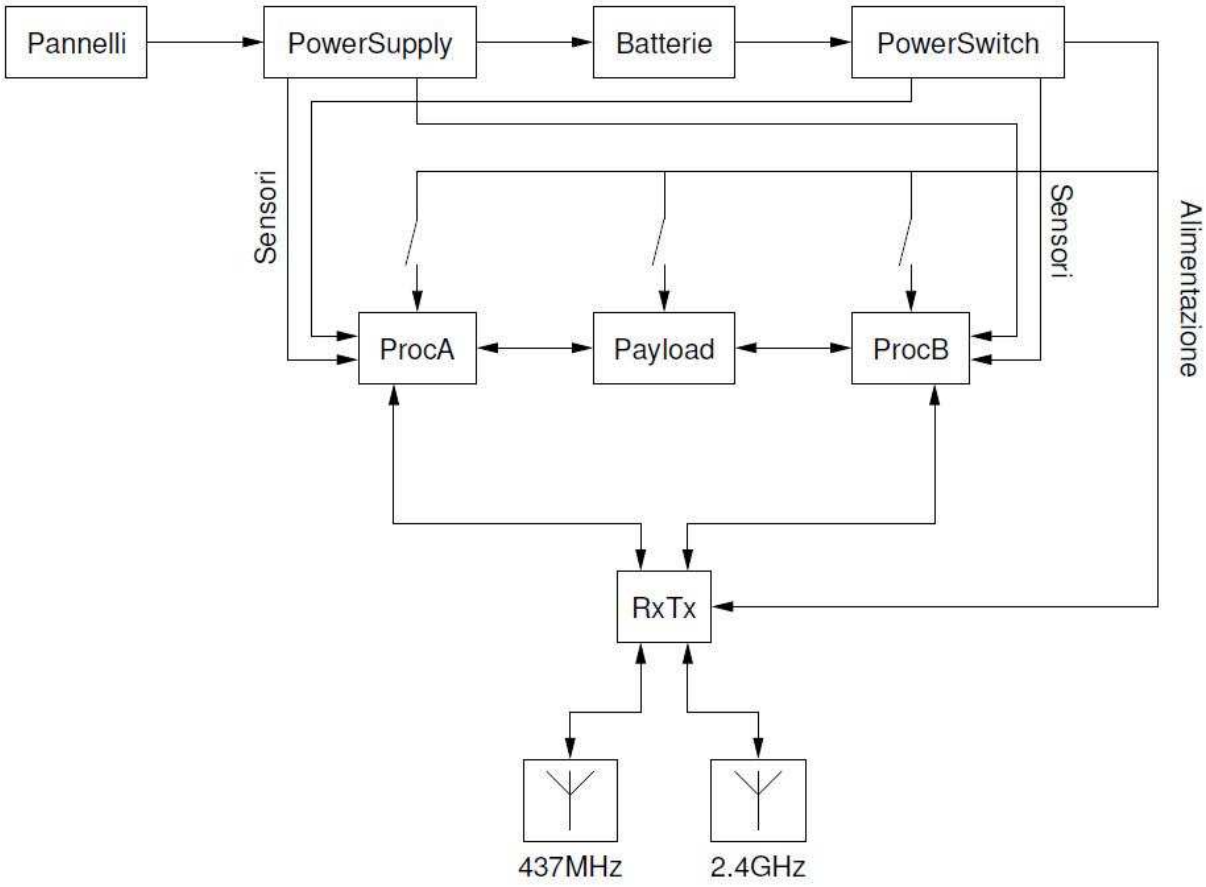


Figura 1.2 Schema a blocchi del satellite PiCPoT

A conclusione del progetto PiCPoT, il lancio del satellite avvenne il 26 luglio 2006 a Baikonur, Kazakistan, per mezzo del razzo vettore Dnepr-LV. Purtroppo, a causa di un problema idraulico al lanciatore, il lancio non è andato a buon fine.

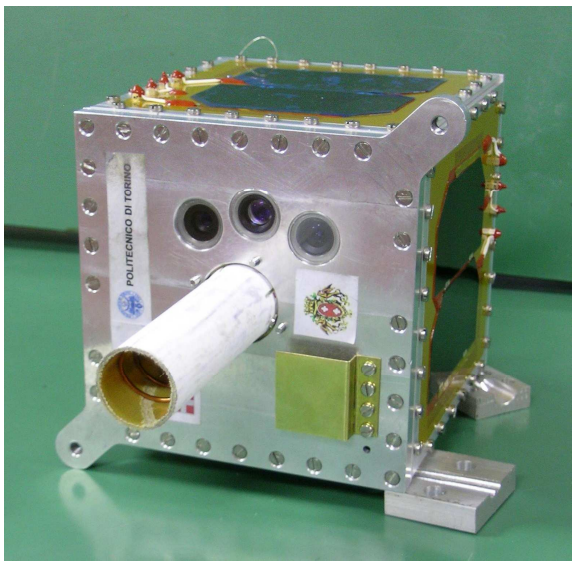


Figura 1.3 Satellite PiCPoT

1.2. AraMiS

AraMiS, acronimo di Architettura Modulare per Satelliti, è il nome del secondo progetto, iniziato nell'autunno del 2006 e portato avanti presso il Politecnico di Torino, che costituisce l'evoluzione di PiCPoT e sul quale attualmente stanno lavorando professori, dottorandi, ricercatori e studenti del Dipartimento di Ingegneria Elettronica in collaborazione con Neohm Componenti, Sky Technology e Spin Electronics.

Gli obiettivi di AraMiS rimangono gli stessi di PiCPoT, ed oltre ad essi si volle rivoluzionare l'approccio di base alla progettazione delle varie parti del satellite creando un sistema modulare. L'architettura del satellite è infatti suddivisa in diversi blocchi funzionali, ognuno dei quali svolge un compito ben preciso; tali moduli, appunto, sono stati ideati in modo da poter essere indipendenti tra loro (pur sempre, ovviamente, dipendendo dal modulo di livello superiore e facendo uso di codici comuni), e quindi separabili, ma soprattutto possono essere riutilizzati o meno, a seconda delle specifiche esigenze della missione.

Mentre PiCPoT, nelle sue varie parti, era progettato e costruito ad-hoc, AraMiS viene assemblato utilizzando i diversi sottosistemi, ognuno dei quali è situato su un circuito stampato separato dagli altri. Tale modularità [5] permette di:

- Ridurre i costi di realizzazione, che, grazie al riutilizzo dei vari moduli standard, gravano su più missioni;
- Ridurre i tempi di progettazione, grazie alla suddivisione del lavoro in vari gruppi che operano nello stesso tempo;
- Ottenere diverse configurazioni del satellite in base agli obiettivi della missione.

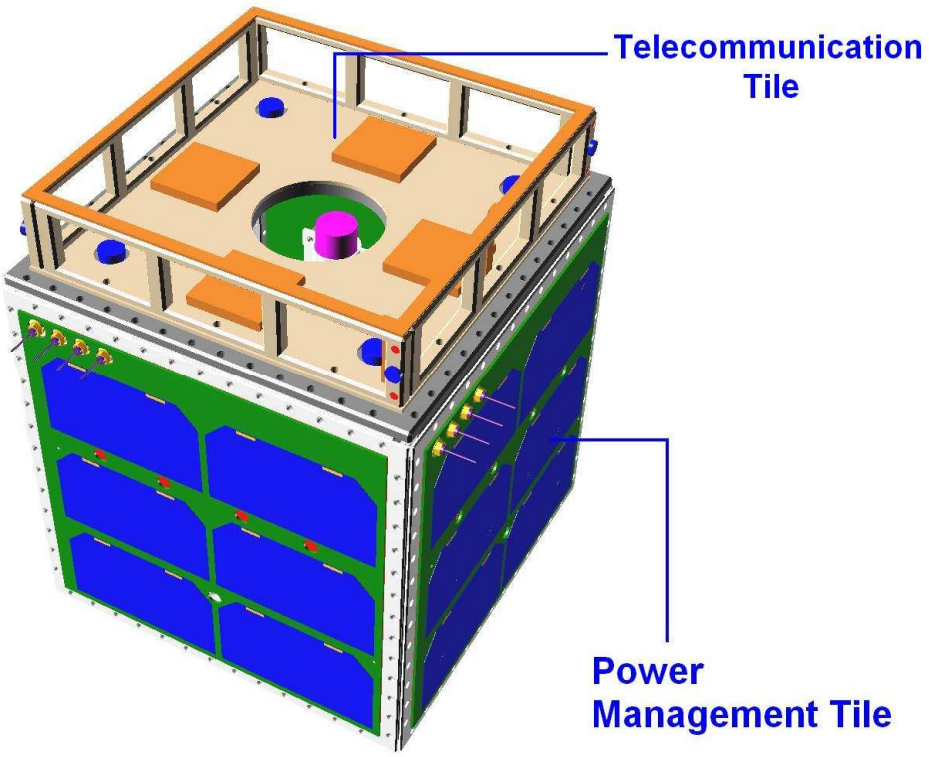


Figura 1.4 Satellite AraMiS, vista esterna

I diversi moduli di AraMiS [6], che vengono montati nelle facce esterne del cubo (figura 1.4), sono raggruppati in due categorie fondamentali, che ne accomunano caratteristiche concettuali, funzioni e codici comuni. Esse vengono definite *tile* (mattonelle), identificano i diversi tipi di moduli standard e sono:

- ***Power Management Tile*** gestisce le alimentazioni dei componenti a bordo, ovvero la ricarica delle batterie e la conversione della tensione, i controlli d'assetto; essa ingloba Power Supply, Power Switch, Solar Panel di PiCPoT con in più un controllo di assetto attivo;
- ***Telecommunication Tile*** riceve e trasmette dati e comandi con la stazione di terra, alle frequenze di 437 MHz e 2.4 GHz. La presenza di due canali serve a ridondare il sistema aggirando eventuali guasti su uno dei due canali. Comprende pertanto tutti i componenti necessari alla comunicazione, tra cui transceiver, microcontrollore, amplificatori di segnale e antenne.

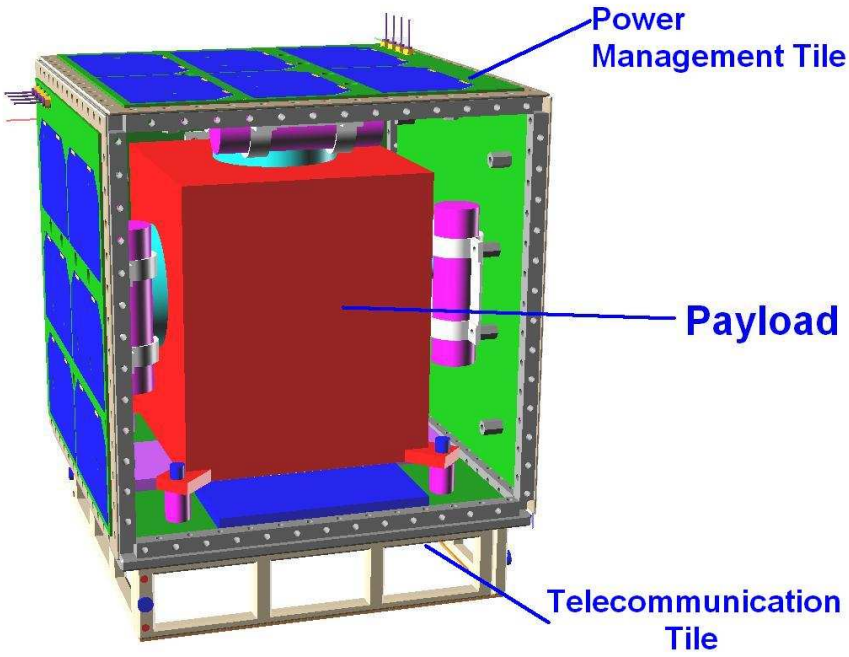


Figura 1.5 Satellite AraMiS, vista interna

All'interno del satellite (figura 1.5) sono invece presenti le batterie e i due moduli non standard:

- ***On-Board Processor*** gestisce il controllo e il funzionamento di tutto il sistema;
- ***Payload*** può avere funzioni analoghe al Payload di PiCPoT ma varia a seconda della missione.

Altro modulo essenziale per AraMiS è la Ground Station. In figura 1.6 si può vedere l'albero di prodotto del satellite, che descrive le varie tile.

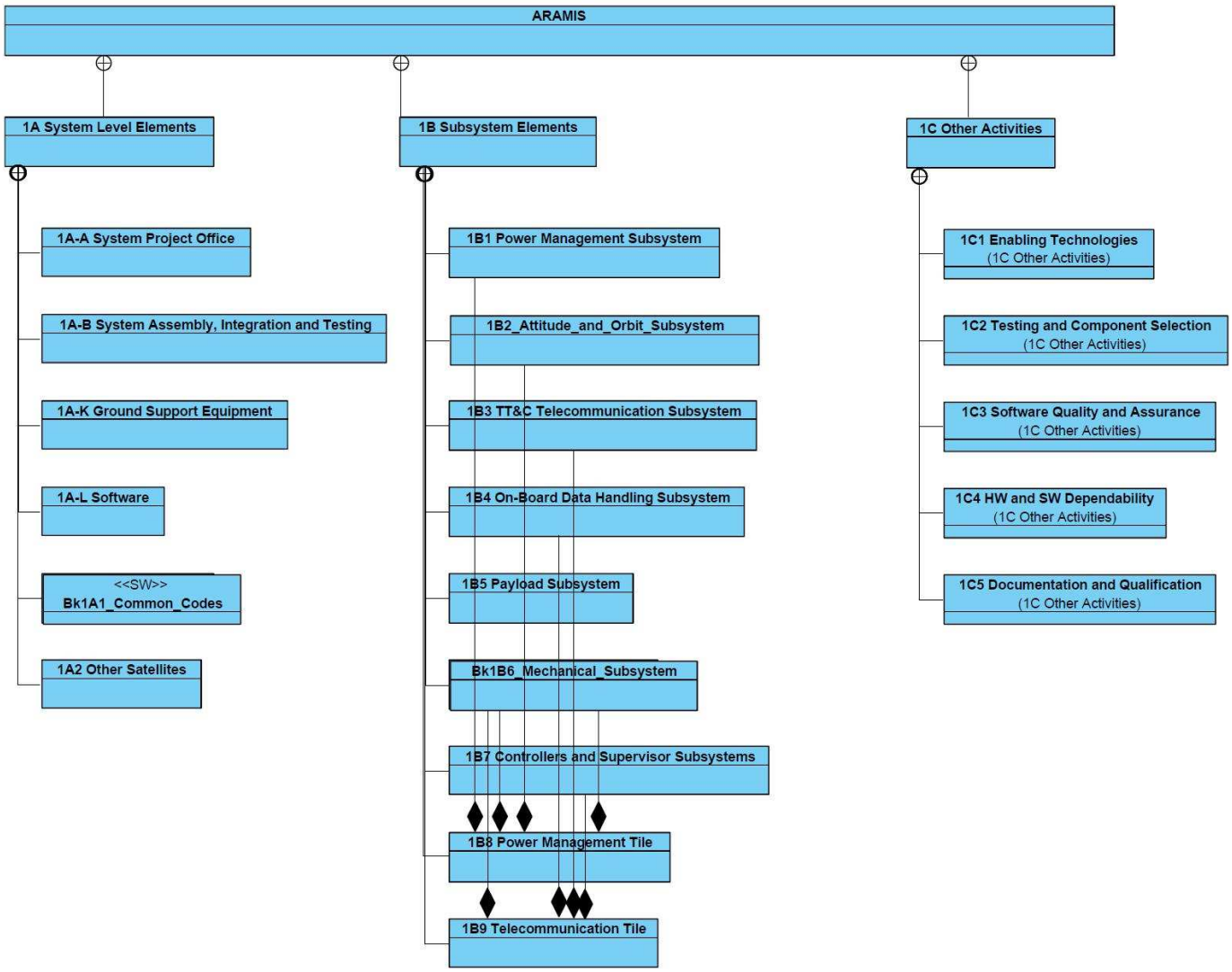


Figura 1.6 Schema di progetto del satellite AraMiS (realizzato mediante diagramma UML)

1.3. Controlli d'assetto

Per controllo d'assetto si intende l'insieme di tutti i componenti e la logica di controllo per garantire o variare in modo regolato la stabilità, la posizione e l'orientamento del satellite intorno ai propri assi di riferimento, presupposto che esso orbiti attorno alla Terra.

Nel progetto PiCPoT si scelse di dotare il satellite di due magneti fissi, posizionati sulla faccia F0 (vedere figura 1.1), che, allineandosi con il campo magnetico terrestre, facevano sì che l'asse x del satellite fosse orientato verso il centro della Terra, e così le due antenne e le fotocamere venivano rivolte verso la stazione di terra. Tale soluzione, però, forzando l'asse x a rimanere vincolato alle linee di campo magnetico, non consentiva di orientare PiCPoT diversamente attorno ai tre assi x, y, z.

Di conseguenza si decise, per AraMiS, di includere in ogni Power Management Tile, ovvero in cinque delle sei facce del satellite, un controllo di assetto attivo [4], composto da:

- **Controllo di Assetto Magnetico**, che utilizza un solenoide;
- **Controllo di Assetto Inerziale**, che utilizza una ruota d'inerzia.

Questo tipo di sistema consente di orientare il cubo in qualsiasi direzione attorno ai tre assi, così da poter decidere quali facce del satellite esporre alla radiazione solare e puntare le fotocamere a seconda delle esigenze.

Inoltre, mentre per PiCPoT, in simulazione, si notava un ondeggiamento al suo transitare sui poli magnetici terrestri, utilizzando un controllo d'assetto attivo è possibile ottenere una stabilità assai maggiore di quella che si avrebbe coi magneti permanenti.

Sia il modulo di controllo di assetto inerziale (denotato in figura 1.7 con il codice 1B21), sia il modulo di controllo di assetto magnetico (1B22) variano il momento angolare applicato alla relativa faccia, sulla quale è montata la Power Management Tile (1B8), e contribuiscono, di conseguenza, a regolare l'inclinazione del satellite, il primo grazie all'impiego di una ruota d'inerzia, un sensore giroscopico ed un motore brushless, il secondo mediante un solenoide ed un sensore di campo magnetico. I comandi di assetto vengono inviati dalla stazione di terra alla Telecommunication Tile ed interpretati dall'algoritmo di controllo di assetto, tramite un microcontrollore, così da essere opportunamente suddivisi in tanti comandi più specifici per i diversi controlli (inerziale o magnetico) e per le diverse tile, al fine di imprimere al satellite la rotazione desiderata.

I diversi moduli di controllo cooperano, in parole povere, secondo il concetto di modularità: ognuno esegue le proprie sotto-istruzioni, ignorando l'effetto complessivo di queste e di quelle degli altri moduli, nonché il comando primario che essi riescono ad eseguire grazie alla loro interazione.

La presenza di più moduli di Power Management aumenta nettamente le prestazioni del sistema di controllo d'assetto in termini di tempo e precisione, ma rende necessariamente più complessa la Telecommunication Tile, che dovrà gestire i vari moduli di controllo.

In figura 1.7 si può vedere la suddivisione in sotto-blocchi dell'intero sistema di controllo d'assetto, denominato 1B2 Attitude and Orbit Subsystem, facente parte della Power Management Tile (1B8).

Di seguito ci si occuperà della progettazione della parte magnetica del controllo di assetto di AraMiS.

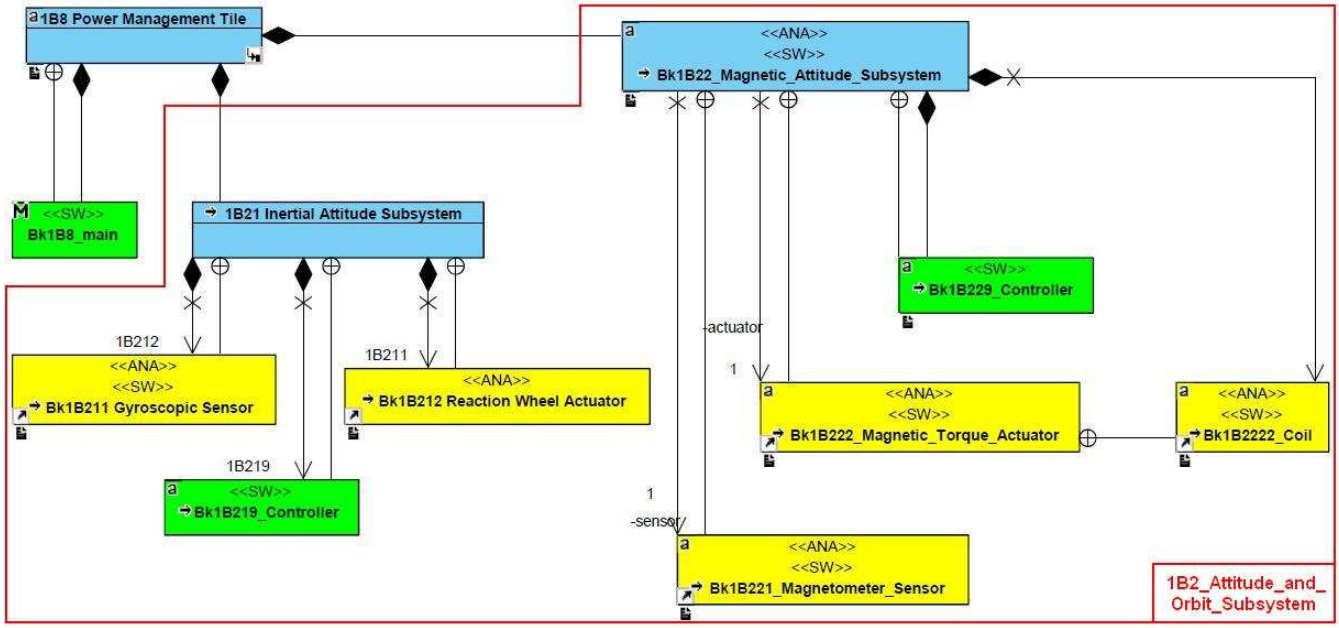


Figura 1.7 Schema del sistema di controllo d'assetto di AraMiS (realizzato mediante diagramma UML)

1.3.1. Controllo d'assetto magnetico – 1B22

Come già detto sopra, il sistema di controllo dell'assetto magnetico permette, insieme al sistema di controllo dell'assetto inerziale, di far compiere al satellite la rotazione desiderata, inviando i comandi direttamente da terra.

Tali sistemi sono costituiti da una parte di attuazione del comando ricevuto e da una parte di sensoristica, utilizzata per fornire dati di telemetria alla stazione di terra, per effettuare calcoli, verificare condizioni critiche o altri propositi. Nel caso specifico dell'assetto magnetico, il modulo 1B22 è costituito da:

- Blocco dell'Attuatore Magnetico (*Bk1B222_Magnetic_Torque_Actuator*), costituito da un solenoide, dal driver e dall'elettronica di condizionamento ed alimentazione;
- Blocco del Sensore di Campo Magnetico (*Bk1B221_Magnetometer_Sensor*), costituito da un sensore di campo magnetico e dall'elettronica di condizionamento ed alimentazione.

Il solenoide, percorso da corrente, consente di fornire alla tile il momento angolare richiesto dal comando, mentre il magnetometro misura i valori di campo magnetico terrestre in orbita, che vengono anche utilizzati nei calcoli necessari per eseguire il comando stesso.

Il blocco 1B22 fa uso di un microprocessore presente nella Power Management Tile per gestire i diversi sotto-comandi da impartire ai singoli componenti, eseguire i calcoli e calibrare i dati telemetrici.

Nel seguito di questo elaborato verranno esplicati in maniera più specifica tutti i diversi aspetti del progetto del controllo di assetto magnetico di AraMiS, il software per programmare il microcontrollore ed il collaudo del circuito finale.

1.4. Il linguaggio UML

Per la progettazione del satellite AraMiS, nonché per la descrizione del sistema, di tutti i suoi sottoblocchi e delle loro funzionalità si è scelto di utilizzare il linguaggio UML (Unified Modeling Language) [7][8].

Anche il modulo 1B22, argomento di questa tesi, come tutti gli altri blocchi di AraMiS, è stato ideato, definito e strutturato tramite l'UML. Pertanto, prima di proseguire nella trattazione, è opportuno fornire una breve descrizione di tale linguaggio, delle sue componenti fondamentali e della simbolistica utilizzata.

L'UML è un linguaggio visuale, nato nel 1995 per la progettazione del software ma ottimamente adattabile alla descrizione di sistemi misti hardware/software, e si basa sulla rappresentazione delle entità coinvolte nel funzionamento del sistema e di tutte le interazioni fra le stesse. Esso fa uso di diversi tipi di diagrammi ed offre molteplici vantaggi; i più importanti sono:

- Facilitare, grazie ad una rappresentazione grafico/concettuale degli elementi (componenti, sottosistemi, segnali, funzioni…) costituenti il sistema, la comprensione del progetto, a partire dai livelli più alti fino allo specifico, anche da parte di persone non esperte nel settore o estranee al progetto stesso;
- Migliorare, oltre che semplificare, la descrizione delle funzionalità del sistema e la definizione delle specifiche di progetto, fornendo una base comune, nell'approccio alla progettazione, tra i vari sottosistemi che compongono AraMiS;
- Rendere esportabili i vari blocchi, essendo questi indipendenti e suddivisi, in modo da poterli riutilizzare in altri progetti, concretizzando il concetto di modularità.

Il tool utilizzato per la creazione dei diagrammi e del progetto in UML è *Visual Paradigm for UML* [9], che permette anche di ottenere automaticamente i file (*source* .c++ e *header* .h) che andranno a costituire il software del microcontrollore, generando il codice direttamente a partire dai diagrammi delle classi e dal corpo delle funzioni, contenuto negli oggetti dei diagrammi stessi (si veda 1.4.2). Per realizzare e gestire il progetto del software, eseguirne il debug e programmare il microprocessore è stato invece utilizzato il tool *IAR*.

Il progetto che costituisce l'argomento di questa tesi è stato realizzato implementando tre diversi tipi di diagrammi UML: il diagramma dei casi d'uso, i diagrammi delle classi e i diagrammi di sequenza.

1.4.1. Diagramma dei casi d'uso

Il diagramma dei casi d'uso descrive le funzionalità e le specifiche del progetto, e da chi, o cosa, queste possano essere messe in atto. Esso rappresenta il punto di partenza nella modellizzazione del sistema e comprende le seguenti categorie di elementi:

- L'**Attore** è ogni qualsivoglia entità, che sia essa un utente umano, un altro sistema o l'ambiente esterno, che interagisce con il sistema in oggetto, richiedendo ad esso la realizzazione di uno o più casi d'uso; esso viene rappresentato con un omino stilizzato (vedere figura 1.8) e ve ne possono essere, eventualmente, più di uno in un diagramma;

Figura 1.8 Linguaggio UML – Attore

- I **Casi d'Uso** sono i compiti che l'attore richiede al sistema, quindi gli obiettivi del progetto in esame; vengono rappresentati graficamente tramite delle ellissi all'interno delle quali è scritta in breve la funzione realizzata dal sistema; possono essere direttamente chiamati dall'attore oppure correlarsi ad altri casi d'uso (vedere figura 1.9);

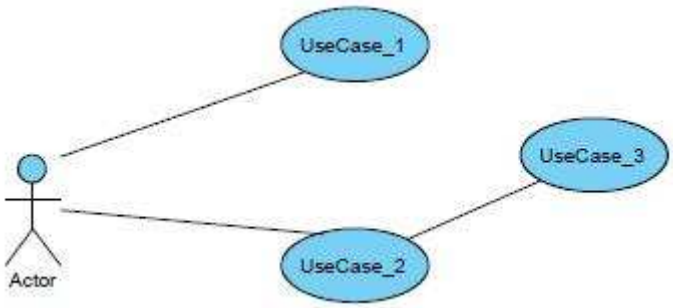


Figura 1.9 Linguaggio UML – Casi d'uso

- **Relazioni tra gli attori**: possono essere di più tipi, in questo progetto non vengono utilizzate, ma citiamo ad esempio la generalizzazione, in cui un attore A, dal quale parte una freccia che punta sull'attore B, può eseguire, oltre ai casi d'uso a cui è preposto, anche i casi d'uso a cui è preposto B; in questo caso si dice che A costituisce la generalizzazione di B;

- **Relazioni tra i casi d'uso**, citiamo ad esempio:
    a) *Generalizzazione*: analoga a quella descritta sopra per gli attori;
    b) *Inclusione*: indicata con una linea tratteggiata recante dicitura <<include>> (figura 1.10), indica che il caso d'uso base include, fra le azioni che può far compiere al sistema, anche le azioni del caso d'uso incluso;
    c) *Estensione*: graficamente simile all'inclusione ma con dicitura <<extend>>, sta a significare che il caso d'uso estensione (vedere figura 1.11) rappresenta una funzionalità opzionale del caso d'uso base;

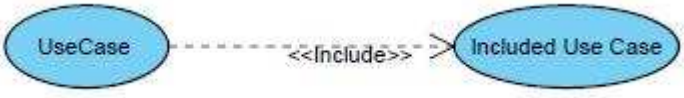


Figura 1.10 Linguaggio UML – Inclusione casi d'uso

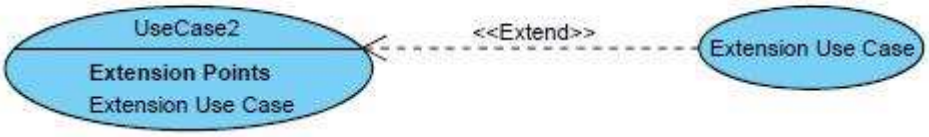


Figura 1.11 Linguaggio UML – Estensione casi d'uso

- **Associazione tra attore e caso d'uso**: è rappresentata con una linea retta (come si può vedere nel diagramma del cap.3) e indica quale attore ha la possibilità di mettere in atto determinati casi d'uso, ovvero, cambiando punto di vista, da quale attore viene richiesto ciascun caso d'uso; un attore è spesso associato a più casi d'uso e, viceversa, un caso d'uso può essere associato a più di un attore.

Oltre ai suddetti elementi, è opportuno puntualizzare che ad ogni elemento del diagramma, come anche ad ogni elemento di tutti gli altri diagrammi, è possibile associare un commento o una nota per dare maggiori spiegazioni. Inoltre, sempre al fine di specificare meglio tutti i vari aspetti della descrizione e delle funzionalità del sistema, è presente la **documentazione**, ossia un campo disponibile al progettista per scrivere ogni possibile ed utile descrizione, spiegazione o dato specifico. Tale documentazione, infatti, verrà riportata nelle tabelle dei capitoli 3, 4 e 5, che costituiscono integralmente, insieme ai diagrammi stessi ed al codice di programmazione, la documentazione del progetto, hardware e software, in tutte le sue parti.

1.4.2. Diagramma delle classi

Il diagramma delle classi è composto essenzialmente dagli **oggetti** e dalle varie associazioni fra di essi ed è volto a caratterizzare minuziosamente il sistema in esame, descrivendolo in tutte le sue componenti, hardware, meccaniche, software o miste hardware/software, ma anche di altro tipo, a seconda delle esigenze di progetto. Un **oggetto** è una qualsiasi entità, facente parte del sistema, che interagisce con l'esterno, con oggetti di altri o del proprio sistema. Per vedere in dettaglio questa interazione, si rimanda al "Diagramma di sequenza".

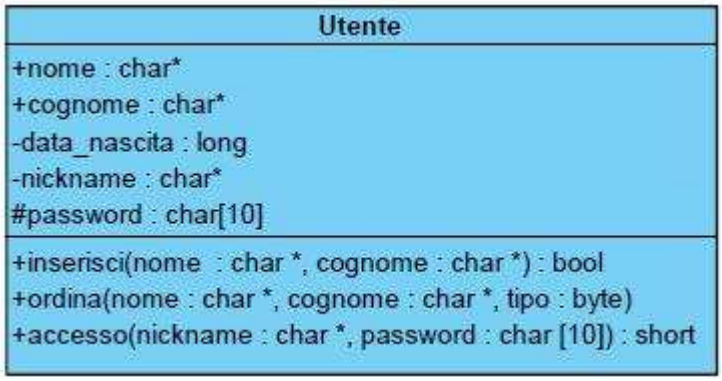


Figura 1.12 Linguaggio UML – esempio di classe

Nel nostro caso un oggetto può identificarsi con uno specifico componente hardware del sistema, ma anche con parti meccaniche del medesimo, o con sotto-blocchi logici indispensabili per il suo coordinamento e la sua funzionalità: in tal caso si parla di oggetti software, come ad esempio il *Controller* del progetto 1B22 (vedi cap.4). Per **classe** si intende, invece, l'astrazione, ovvero la generalizzazione, di un oggetto, che ne costituisce l'istanza specifica. Caratteristiche fondamentali di un oggetto (e della sua classe) sono i suoi **attributi** ed i suoi **metodi** (che nella classe vengono denominati **operazioni**).

- Un **attributo** è una proprietà dell'oggetto, sia essa logica, fisica o quant'altro: ad esempio, se l'oggetto in questione è un sensore, suoi attributi possono essere la sensibilità e il consumo di potenza; invece, se l'oggetto fosse un polinomio, gli attributi potrebbero essere i coefficienti delle diverse potenze; in termini di software, invece, gli attributi equivalgono a variabili, strutture e *define* del linguaggio C. Ogni attributo può essere caratterizzato da un tipo (ad es. *int* per i numeri interi), da un valore iniziale, dall'accessibilità (che definisce se il suo valore può essere modificato e, se sì, da quali oggetti), dalla visibilità, ed altre proprietà più specifiche. Caso usuale, nel progetto in esame, è l'utilizzo di un attributo per associare all'oggetto in questione un altro oggetto (o classe), che va proprio a rappresentare il tipo di tale attributo, in modo che il primo possa servirsi anche degli attributi del secondo, o i suoi metodi possano chiamare quelli dell'altro. Gli attributi compaiono nel primo riquadro sotto quello del nome dell'oggetto (vedi figura 1.12).

- Il **metodo** indica quali operazioni può compiere l'oggetto ed equivale alle funzioni C, per un oggetto software, oppure ai fili di segnale, per oggetti hardware. Esso può venire associato ad un valore di ritorno, di cui si indica il tipo (vedi esempio in figura 1.12), che esso restituisce al chiamante, ed ai parametri che esso riceve dal medesimo (indicati fra parentesi). I metodi compaiono nel riquadro in basso, sotto quello contenente gli attributi.

Un diagramma delle classi (o diagramma degli oggetti) riporta la raffigurazione dei diversi oggetti, nella loro gerarchia, connessi tramite i diversi tipi di associazioni, analoghe a quelle del diagramma dei casi d'uso, a cui si aggiunge la *composizione*, molto utile nei progetti di tipo modulare, nella quale un oggetto padre è costituito da due o più oggetti figli, ai quali è collegato con una freccia che si diparte da un rombo (vedi diagrammi del cap.4.)

1.4.3. Diagramma di sequenza

I diagrammi di sequenza descrivono, nel linguaggio UML, lo svolgimento vero e proprio delle azioni compiute dal sistema, ovvero i casi d'uso. Ognuno di questi ultimi, infatti, viene scomposto in un insieme di azioni, che necessariamente si susseguono nella messa in atto del caso d'uso stesso. Tale sequenza rappresenta il *corso d'azione base*, e sono gli oggetti, che compongono il sistema, ad implementare effettivamente le varie operazioni, mediante dei **messaggi** contenenti le chiamate ai loro diversi metodi. Ogni oggetto può chiamare i metodi di un altro oggetto, purché questi siano ad esso visibili, oppure i propri, ed il **messaggio** costituente tale chiamata deve anche contenere i parametri da passare al metodo in questione.

Rappresentazione più espansa di tale corso d'azione è, infatti, il diagramma di sequenza, in cui i messaggi vengono appunto elencati nell'esatta sequenza temporale in cui essi vengono inviati, tramite l'associazione di un numero d'ordine ad ognuno di essi, secondo la logica degli elenchi puntati: i messaggi subordinati ad un precedente messaggio principale costituiscono una sottonumerazione dell'elenco primario, dove ciò può ripetersi normalmente o a struttura annidata (lo si può vedere chiaramente nei diagrammi di sequenza del cap.5). Poiché nell'implementazione di uno specifico caso d'uso sono coinvolti un numero finito di oggetti, che, però, possono scambiarsi messaggi a vicenda, per dare maggior chiarezza alla rappresentazione, nel diagramma ogni oggetto ha una propria "linea di vita" (*lifeline*) dalla (alla) quale si dipartono (giungono) le chiamate ai metodi, dove l'asse verticale identifica il tempo d'azione, crescente dall'alto verso il basso.

Per tutto il tempo in cui un oggetto è impegnato nell'esecuzione di un proprio metodo o nell'attesa dell'avvenuta esecuzione da parte di un altro oggetto, la *lifeline* è coperta da una barra azzurra denominata *activation*. Altro elemento fondamentale nei diagrammi di sequenza che seguiranno è l'***Alt. Combined Fragment***, struttura a tabella ad una sola colonna e con più campi, che rappresentano le possibili serie di operazioni alternative eseguite dall'oggetto (o dagli oggetti) in questione, a seconda che l'una o l'altra (o le altre) condizione venga soddisfatta, o semplicemente nel caso in cui venga o meno soddisfatta la prima. Esso equivale esattamente al costrutto ***if*** del linguaggio C, con gli eventuali ***else*** ed ***else if***.

Risulta infatti evidente come i diagrammi di sequenza rappresentino l'implementazione delle specifiche di progetto descritte nel diagramma dei casi d'uso, e come essi costituiscano, allo stesso tempo, la rappresentazione (visivamente più comprensibile) delle funzioni C. Grazie all'UML, pertanto, viene facilitato il passaggio da elencazione concettuale delle azioni svolte dal sistema a codice di programmazione vero e proprio (sia esso C o altri linguaggi), il quale altro non è che la "traduzione" dei vari diagrammi di sequenza, ricavabile da un'immediata ed intuitiva analisi dei medesimi.

In figura 1.13 si può vedere un basilare esempio di diagramma di sequenza, mentre nei capitoli 3, 4 e 5 si potranno vedere in maniera dettagliata i diversi tipi di diagrammi UML, applicati al progetto del controllo di assetto magnetico, oggetto di questa tesi.

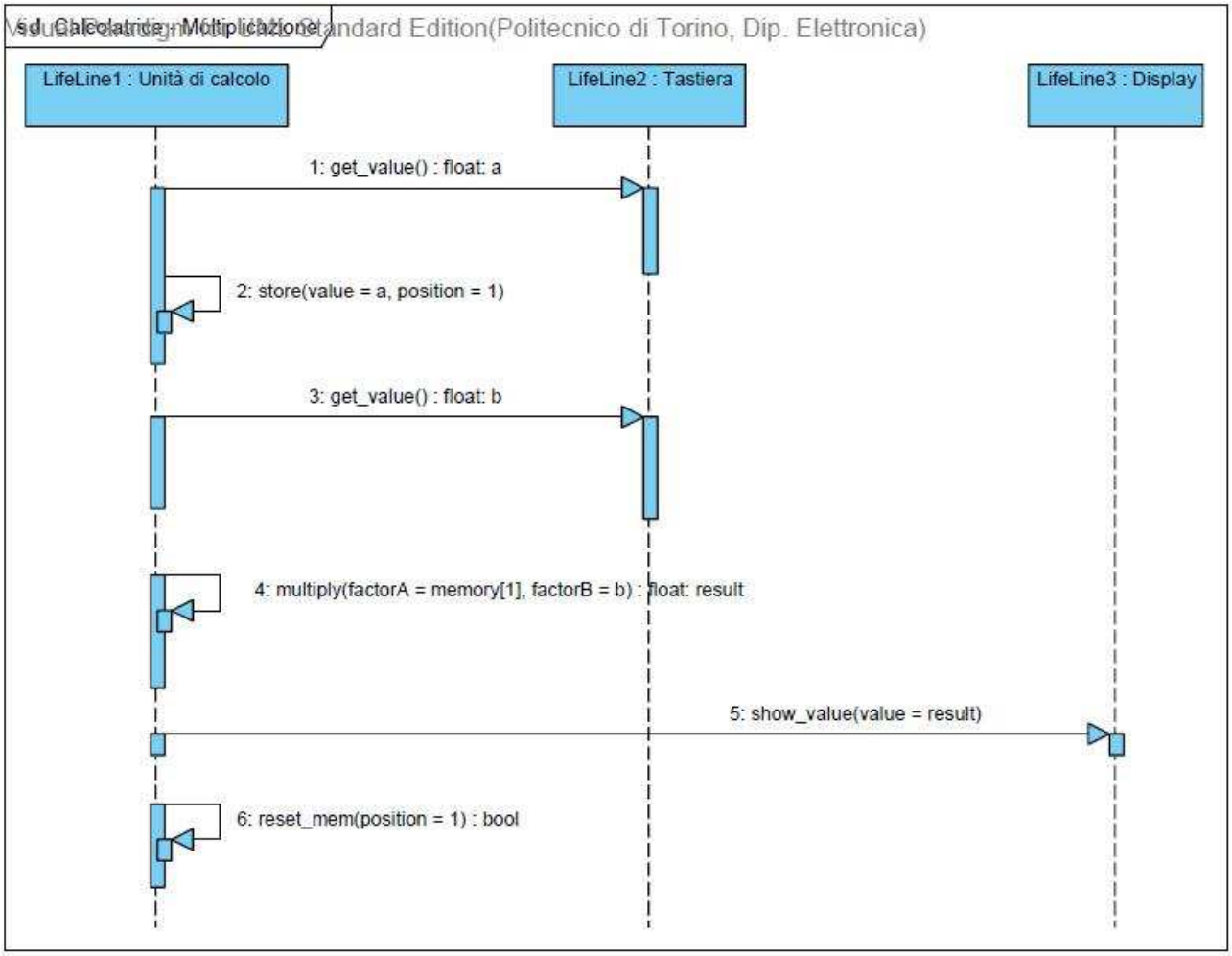


Figura 1.13 Linguaggio UML – esempio di diagramma di sequenza

Capitolo 2

2. Descrizione del problema e scelte progettuali

Prima di iniziare la trattazione specifica dell'argomento, è necessario focalizzare l'attenzione sugli aspetti fondamentali della descrizione di un problema empirico, alla base di ogni progetto scientifico. In questo capitolo ci si soffermerà anzitutto sulle condizioni ambientali in cui si troverà ad operare il satellite e, di conseguenza, il sistema in esame e, in seguito, verranno descritte le scelte progettuali volte a realizzare il controllo d'assetto magnetico ed i fondamenti fisici ed elettronici di queste.

2.1. Ambiente operativo

Al fine di comprendere meglio i vincoli di temperatura e campo magnetico considerati nel seguito di questo elaborato, ma anche altre problematiche riguardanti il satellite in generale, occorre descrivere l'ambiente operativo caratterizzante la missione di AraMiS. Le particolari condizioni dell'ambiente spaziale, infatti, determinano le specifiche, alle quali ogni modulo del progetto globale deve attenersi. Pertanto, il progettista deve prestare attenzione alle peculiarità ed ai punti deboli della tecnologia utilizzata per la costruzione dei componenti, e la scelta nonché il dimensionamento di questi ultimi devono essere necessariamente condizionati dalle caratteristiche estreme dell'ambiente spaziale.

Più precisamente, il satellite AraMiS viene progettato per orbitare ad una quota compresa tra i 600 Km e gli 800 Km, quindi va ad essere posizionato nella cosiddetta orbita LEO (Low Earth Orbit) [4][6], compresa fra l'atmosfera terrestre e le fasce di Van Allen. Molteplici missioni spaziali hanno luogo nell'orbita LEO, come ad esempio gli Space Shuttle o le stazioni spaziali. Tale scelta è dettata dalla necessità di rimanere tra i 200 Km e i 2000 Km: al di fuori di tale fascia, infatti, la sopravvivenza del satellite, a causa delle estreme condizioni esterne, verrebbe ulteriormente compromessa. Al di sotto dei 200 Km ci si troverebbe nell'atmosfera terrestre, dove il satellite verrebbe eccessivamente sollecitato dalle forze d'attrito di quest'ultima, peggiorando il proprio tempo di vita, mentre al di sopra dei 2000 Km si entrerebbe nelle fasce di Van Allen, dove una troppo elevata esposizione alle radiazioni comprometterebbe gravemente il funzionamento dei dispositivi costituenti il satellite.

Come è possibile vedere in figura 2.1, anche nella LEO sono presenti radiazioni solari e cosmiche, ma queste sono in misura ridotta rispetto alle fasce di Van Allen: ciò nonostante, nella progettazione di AraMiS si è scelto di utilizzare, ovunque fosse possibile, componenti elettronici testati per l'utilizzo in ambiente spaziale, ovvero in grado di sopportare tale esposizione radioattiva (i cosiddetti "Rad-Hard"). Tra le fonti di disturbo in ambiente spaziale [10] si notano principalmente:

- Protoni (p) ed elettroni (e) ad alta energia, che però non sono presenti nella LEO poiché vengono trattenuti nelle fasce di Van Allen;
- Protoni, particelle α, ioni (i) ed elettroni costituenti il vento solare;
- Ioni ad altissima energia costituenti i raggi cosmici.

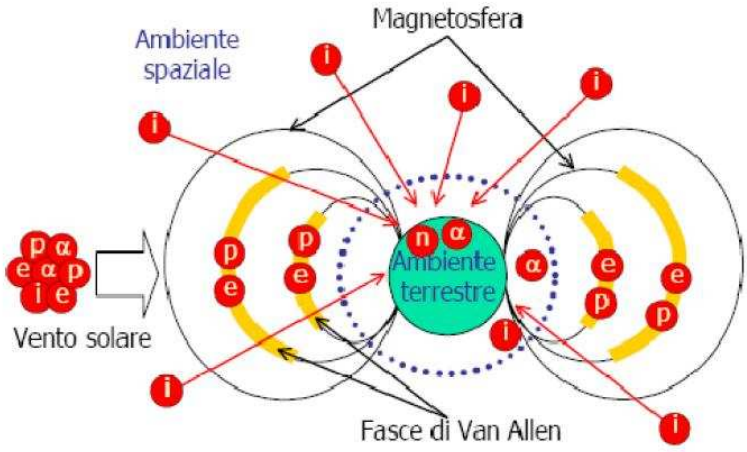


Figura 2.1 Radiazioni cosmiche e fonti di disturbo in ambiente spaziale

Tali particelle possono provocare diversi guasti ai dispositivi elettronici, tra cui:

- Cambiamenti dei parametri:
  - Total Ionizing Dose (TID)

- Cambiamenti della funzionalità:
  - Non distruttivi:
    - Single Event Upset (SEU)
    - Multiple Bit Upset (MBU)
    - Single Event Transient (SET)
    - Single Event Functional Interrupt (SEFI)

  - Distruttivi:
    - Single Event Latch-up (SEL)
    - Single Gate Rupture (SGR)

I principali vantaggi dell'orbita LEO sono il basso ritardo di propagazione (20-25 ms) dei segnali tra satelliti e rispettive stazioni di terra, confrontabile con quello di alcuni collegamenti terrestri, e, come abbiamo già visto, la presenza di ridotte fonti di disturbi. Le metodologie di analisi di tali disturbi e le tecniche per evitarne o limitarne i danni non costituiscono argomento di questa tesi, per cui non si prosegue ulteriormente nella descrizione specifica di questi effetti ma ci si limita a precisare che, nel progetto globale del satellite, sono previsti circuiti di protezione da latch-up per le alimentazioni dei componenti sensibili a tale fenomeno.

Altro aspetto preponderante in ambito spaziale è la **temperatura**. Si deve infatti tenere conto, nella progettazione di qualsiasi sistema destinato a orbitare nella LEO, della pressione e, quindi, della concentrazione dell'aria, estremamente basse (pressoché condizione di vuoto): ciò porta a temperature medie molto basse, ma, d'altro canto, fa sì che la dissipazione di calore per convezione (principale causa della dissipazione in ambiente terrestre) sia quasi assente. A questo si somma la maggiore esposizione all'irraggiamento solare, dovuta alla mancanza di nubi a quota così elevata: si avranno, pertanto, enormi variazioni di temperatura tra le facce del cubo esposte al sole e quelle in ombra. Da un lato, le facce esposte, pur surriscaldandosi, si raffredderanno non appena il satellite ruoterà su sé stesso (anche se non

raggiungeranno la temperatura della zona d'ombra, a causa della bassissima dissipazione del calore in orbita), dall'altro, le facce fredde vedranno aumentare la propria temperatura, una volta esposte. Considerando, inoltre, l'ottima conducibilità termica della struttura del satellite, costituita di materiale metallico, si può ipotizzare, con buona approssimazione, che all'interno del cubo (dove verranno montate le schede) ci si trovi in un campo di temperature comprese fra i -30°C e i 20°C, che costituiranno i limiti che andranno a vincolare le specifiche di progetto.

Infine, nello specifico del progetto in esame, bisogna sicuramente tenere conto del valore del **campo magnetico terrestre** in orbita, fondamentale per il funzionamento del controllo d'assetto così come impostato. Una volta che la corrente fluisce attraverso la bobina, infatti, questa si magnetizza, originando un proprio dipolo, che la forza ad allineare il proprio asse con le linee del campo magnetico (per il dettaglio si veda il paragrafo seguente); l'intensità e la direzione di quest'ultimo variano a seconda della posizione del solenoide nello spazio, ed è dunque di massima importanza conoscere il range di valori che il campo può assumere, per poter generare il momento angolare richiesto dal controllore d'assetto alla singola faccia di AraMiS.
I valori misurati di campo magnetico terrestre in superficie sono sempre effetto della sovrapposizione di più contributi [11], aventi ciascuno origine diversa:

- **Campo principale**, generato dal nucleo fluido della Terra;
- **Campo crostale**, generato dalle rocce magnetizzate della crosta terrestre;
- **Campo esterno**, generato da correnti elettriche aventi luogo nella ionosfera e nella magnetosfera;
- **Campo d'induzione elettromagnetica**, generato da correnti indotte nella crosta e nel mantello da parte del campo esterno, variabile nel tempo.

Tra questi, il campo principale origina il 99% del campo magnetico complessivo misurabile in superficie. Diversi studi dimostrano come tale campo sia assimilabile, al 95%, a quello che verrebbe generato da un dipolo magnetico centrato nel centro della Terra ma inclinato, rispetto all'asse di rotazione terrestre, di 11°30'.

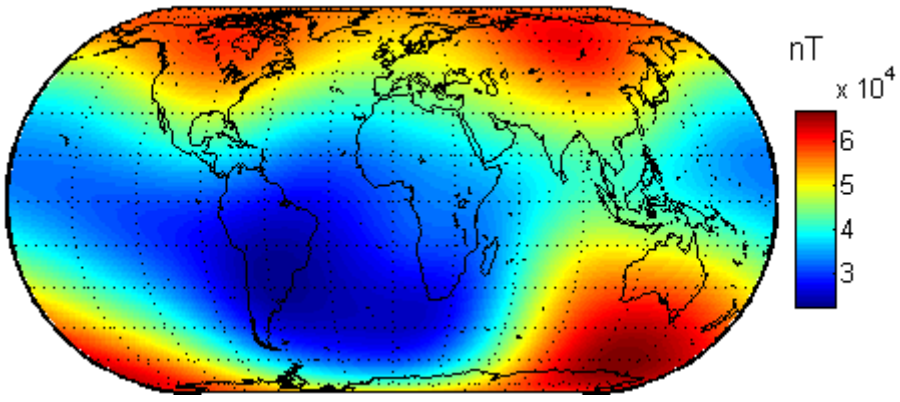


Figura 2.2 Distribuzione dell'intensità del campo magnetico terrestre nel 2009 [12]

Il campo magnetico terrestre si esprime, per convenzione internazionale, tramite il vettore di induzione magnetica $\vec{B}$. La sua unità di misura per il Sistema Internazionale è il Tesla (T), ma nelle applicazioni pratiche viene spesso utilizzato un sottomultiplo, il Gauss (G), uguale a 10^{-4} T. Il modulo del campo magnetico misurato sulla superficie terrestre varia da 0.2 G all'equatore a 0.7 G ai poli; in figura 2.2 si può vedere la distribuzione della sua intensità. In orbita, oltre alla quota, bisogna tenere conto dell'inclinazione dell'orbita stessa, che la porta ad assumere una forma polare, anziché perfettamente circolare, ma anche dell'effetto di altri campi magnetici presenti nello spazio, originati da altri corpi celesti o da innumerevoli fattori di disturbo. Ad esempio, per un'orbita ad 800 Km di quota con inclinazione di 89° (coordinate attendibili per la potenziale orbita di AraMiS), grazie al tool online di simulazione SPENVIS [13], si ricava un campo geomagnetico compreso tra 0.15 G e 0.45 G.
In conclusione, si è deciso di assumere come plausibili, per le componenti x e y del campo geomagnetico misurato in orbita LEO, valori compresi tra -0.625 G e 0.625 G, in base ai quali verranno scelti e dimensionati i componenti del magnetometro (vedere capitolo 6).

2.2. Solenoide, Dipolo, Campo magnetico, Coppia e Momento angolare

Consideriamo, in questa prima trattazione dei principi elettromagnetici che portano all'utilizzo di un solenoide per la realizzazione del progetto, la presenza di un solo sistema di controllo di assetto magnetico, e, quindi, di un solo solenoide. Questa scelta è dovuta in primo luogo al voler semplificare il più possibile le diverse grandezze in gioco nel sistema fisico in esame: le considerazioni e le equazioni sotto presentate, infatti, sarebbero valide qualora fosse attivo esclusivamente il blocco 1B22 di una sola tile; qualora vi sia la combinazione di più contributi, invece, non cessano di essere valide, ma occorre tenere conto di tutte le varie componenti di coppia e momento angolare attorno ai tre assi di rotazione del satellite. In secondo luogo, la coordinazione dei diversi sottosistemi di controllo dell'assetto non è argomento di questa tesi, ma sappiamo che ad ogni singolo sottosistema, nello specifico la parte di assetto magnetico, verrà richiesto un momento angolare, riferito al solo sottoblocco in questione, il quale sarà già stato calcolato, in altra sede, come componente, da applicarsi alla specifica faccia del satellite, di un effetto globale, da applicarsi all'intero satellite. Pertanto si effettueranno i calcoli relativi ad un solo asse di rotazione del satellite, tenendo presente che, nella struttura modulare del satellite, ogni tile sarà dotata di un analogo sistema di controllo.

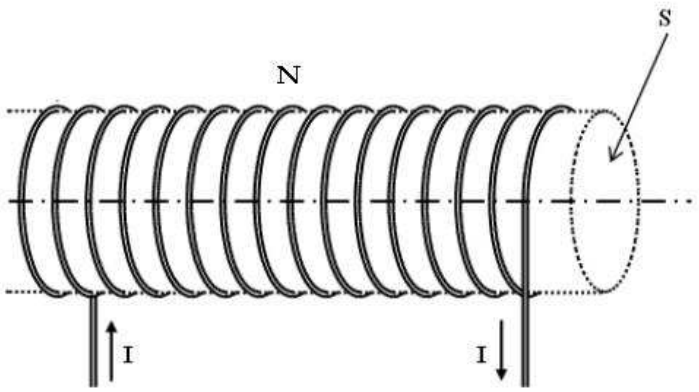


Figura 2.3 Solenoide percorso da corrente

Un generico solenoide (figura 2.3), se percorso da corrente, si magnetizza con un proprio **momento magnetico** (o *momento di dipolo magnetico* o, più brevemente, *dipolo*) $\vec{D}$, il cui modulo viene definito come segue:

$$\left|\vec{D}\right| = N \cdot S \cdot I$$

dove N rappresenta il numero di spire del solenoide, S la sezione della spira e I l'intensità della corrente che scorre nel solenoide [4][14]. Il vettore $\vec{D}$ giace sull'asse del solenoide e il verso è definito dalla regola della mano destra, come in figura 2.4.

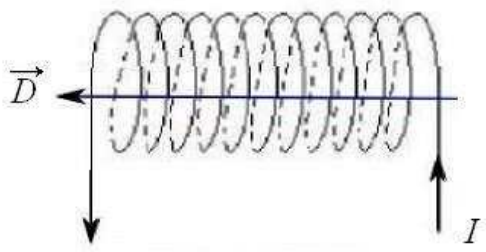


Figura 2.4 Momento di dipolo magnetico di un solenoide percorso da corrente

Se il solenoide viene immerso in un campo magnetico $\vec{B}$, il suo momento magnetico $\vec{D}$ tende ad allinearsi al campo originando una coppia $\vec{C}$. Questa importantissima proprietà consente, nella nostra applicazione, al satellite, o, più precisamente, alla faccia contenente il solenoide, qualora esso sia l'unico percorso da corrente, di allinearsi al campo magnetico terrestre. In ogni caso, l'i-esimo solenoide, fornisce alla propria tile un contributo di **coppia** (o *momento torcente magnetico*) dato dal prodotto vettoriale tra il proprio momento magnetico ed il vettore di campo magnetico terrestre:

$$\vec{C} = \vec{D} \wedge \vec{B}$$

La direzione di $\vec{C}$ sarà ortogonale sia al dipolo che al campo, mentre la sua intensità sarà così definita:

$$\left|\vec{C}\right| = \left|\vec{D}\right| \cdot \left|\vec{B}\right| \cdot \sin\alpha$$

Si può vedere, quindi, che la configurazione in cui tale coppia sarà massima, è quella in cui α=90°, cioè quando campo magnetico terrestre e asse del solenoide si trovano ad essere perpendicolari (figura 2.5);

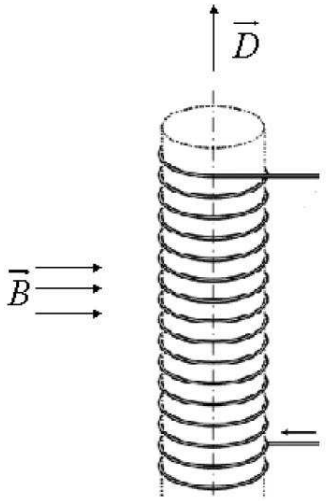


Figura 2.5 Solenoide in campo magnetico perpendicolare

quando invece si troveranno paralleli la coppia sarà nulla, poiché sinα=0 (figura 2.6).

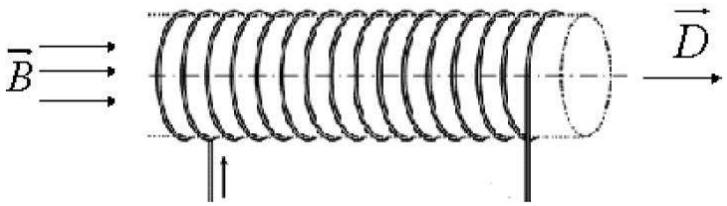


Figura 2.6 Solenoide in campo magnetico parallelo

Questo è lo stato finale di equilibrio a cui verrebbe portato il sistema solenoide – campo magnetico dalla coppia generatasi, se venisse lasciata scorrere la corrente nel solenoide per un tempo sufficientemente lungo. Invece, se l'alimentazione del solenoide venisse fermata prima di tale tempo, otterremmo l'inclinazione desiderata del medesimo rispetto al campo magnetico della Terra.

Sapendo l'intensità del campo magnetico terrestre (lo si può misurare), la corrente che scorre nel solenoide (anch'essa misurabile), la sua sezione e il numero di spire, e scelto un angolo di inclinazione da far raggiungere al satellite, possiamo conoscere la coppia che è necessario generare. Consideriamo la seguente relazione tra coppia ed accelerazione angolare:

$$\vec{C} = \underline{\underline{J}} \cdot \frac{d\vec{\omega}}{dt} \qquad \textbf{(1)}$$

dove $\underline{\underline{J}}$ rappresenta il momento d'inerzia del satellite, espresso nella forma di matrice tensoriale (poiché non si può conoscere a priori l'asse di rotazione), in Kg·m^2 e $\vec{\omega}$ la sua velocità angolare (attorno a tale asse) in rad/s, e teniamo sempre presente che da questo punto in avanti le equazioni presentate sono riferite al caso ideale in cui un solo sistema di controllo d'assetto magnetico agisca sul satellite.
Dalla (1) si ricava l'espressione della velocità angolare raggiunta dal satellite applicandogli una coppia $\vec{C}$ per un tempo T (come in figura 2.7):

$$\vec{\omega} = \int_0^T \vec{C} \cdot \underline{\underline{J}}^{-1} dt$$

Quindi il satellite assumerebbe un'inclinazione pari a θ, dove:

$$\vec{\theta} = \int_0^T \vec{\omega} \cdot dt$$

Come già precisato, le equazioni si riferiscono all'effetto di un singolo solenoide su una sola faccia del satellite, ma sono utili per rendere chiaro il principio di funzionamento del sistema: applicando una coppia $\vec{C}$ costante al satellite, la sua velocità angolare cresce linearmente nel tempo in cui tale coppia è applicata. Quando tale coppia non viene più applicata,

esso continua a ruotare a velocità angolare costante, mentre l'angolo θ avrà andamento parabolico finché viene applicata la coppia e rettilineo una volta cessata la medesima. Per ottenere una coppia costante, supponendo costante il campo magnetico, è sufficiente alimentare il solenoide con una corrente costante, la cui intensità si può ricavare conoscendo la tensione di alimentazione del solenoide e la resistenza del medesimo, calcolata conoscendone i dati fisici (paragrafo 2.2.2).

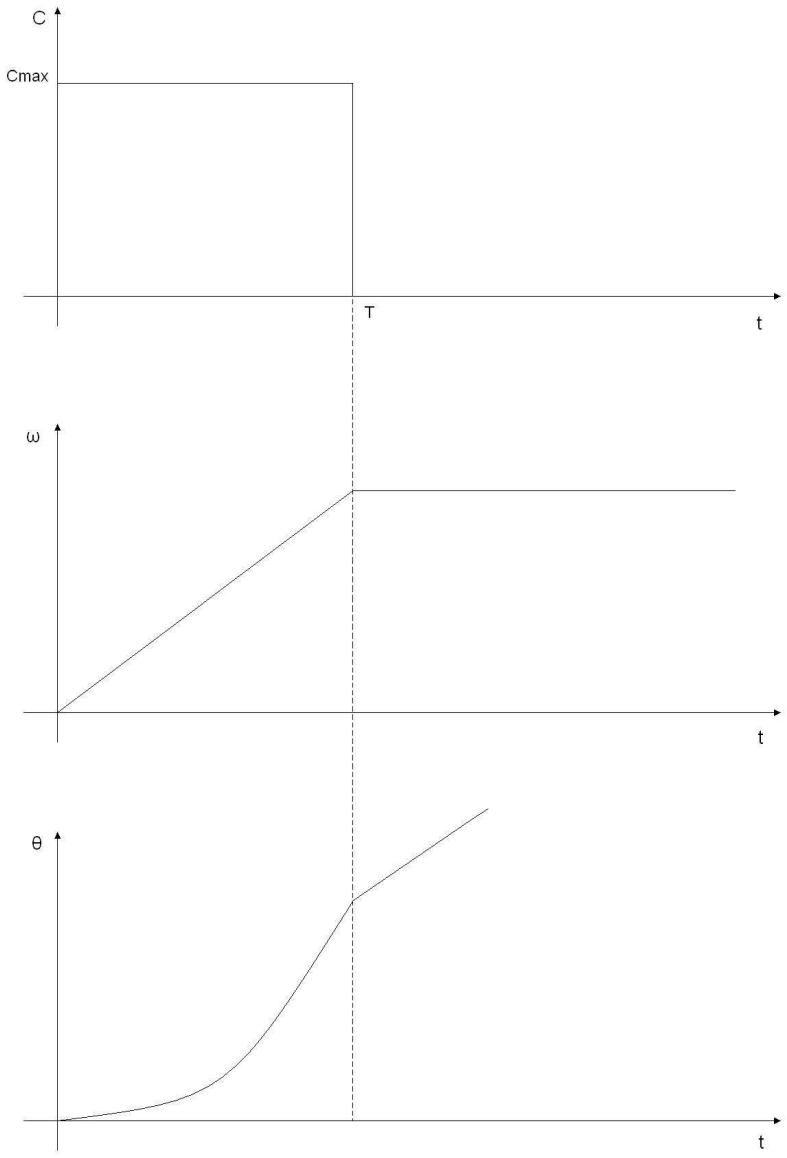


Figura 2.7 Applicazione di una coppia a gradino

Questo tipo di pilotaggio del solenoide con coppia a gradino, oltre a non tenere conto del contributo delle altre facce di AraMiS dotate di sistema di controllo di assetto, sarebbe possibile soltanto idealmente. Nella realtà è evidente come non siano perfettamente costanti grandezze quali la direzione e l'intensità del campo geomagnetico e la corrente nel solenoide, che varia al variare della tensione di alimentazione e/o al discostarsi della resistenza dal valore teorico. Pertanto, la scelta progettuale consiste nel far sì che ogni solenoide venga alimentato, se e quando richiesto dall'algoritmo di controllo di assetto, per generare un determinato momento angolare $\vec{L}$.

In un sistema di riferimento polare, sappiamo che il momento torcente, che nel nostro caso coincide con la coppia $\vec{C}$, è così definito:

$$\vec{C} \equiv \vec{r} \wedge \vec{F}$$

dove $\vec{F}$ è il vettore indicante la forza ed $\vec{r}$ il vettore distanza tra il fulcro e il punto di applicazione della forza. Il momento angolare di un corpo in rotazione rispetto al medesimo fulcro è invece:

$$\vec{L} \equiv \vec{r} \wedge \vec{p}$$

dove $\vec{p}$ è il vettore quantità di moto. Si può dimostrare (la dimostrazione è qui tralasciata per brevità) che:

$$\vec{C} = \frac{d\vec{L}}{dt} + \vec{v}_o \wedge \vec{p} \quad \textbf{(2)}$$

Dal momento che $\vec{v}_o$ è il vettore che indica la velocità di moto relativo del polo rispetto all'origine e noi consideriamo fermo tale polo, si deduce che il termine di prodotto vettoriale nella (2) è nullo, quindi si ricava:

$$\vec{C} = \frac{d\vec{L}}{dt} \Rightarrow \vec{L} = \int_0^T \vec{C} \cdot dt \quad \textbf{(3)}$$

È facile vedere che il momento angolare altro non è che il prodotto tra la velocità angolare ω ed il momento inerziale J, infatti si misura, secondo il Sistema Internazionale, in $Kg \cdot m^2/s$.

Siccome la coppia $\vec{C}$ non è costante e, allo stesso tempo, le sue variazioni non sono prevedibili nel tempo (ma possono essere misurate nei termini di corrente nel solenoide e campo magnetico), l'integrale della (3) non può essere svolto nella forma proposta, ovvero non possiamo conoscere il tempo T impiegato per fornire al solenoide un momento angolare $\vec{L}$ applicando una coppia $\vec{C}$. Immaginiamo invece di dividere l'intervallo temporale [0,T] in n infinitesimi intervalli di durata Δt, con $n \to \infty$; pertanto, considerando i soli moduli dei vettori poiché già sappiamo che il momento angolare sarà espresso in riferimento all'asse di rotazione del solenoide, l'integrale si può esprimere nella forma:

$$\left|\vec{L}\right| = \sum_{i=0}^{n} \left|\vec{C}\right| \cdot \Delta t = \sum_{i=0}^{n} N \cdot S \cdot I \cdot \left|\vec{B}\right| \cdot \Delta t \quad \textbf{(4)}$$

In tal modo, pur non conoscendo T, noto però il **passo di integrazione** Δt e noti tutti i fattori del prodotto all'interno della sommatoria, possiamo considerare trascorso tale tempo, ossia interrompere l'alimentazione del solenoide, non appena viene raggiunto il valore di momento angolare richiesto: questo è il procedimento sul quale si basa il controllo di assetto magnetico del satellite, illustrato dettagliatamente nella parte software del progetto. L'approccio più intuitivo, per eseguire questo integrale tramite sommatoria, è quello di inizializzare a 0 una variabile e sommare, ad ogni passo di integrazione, ovvero ogni Δt millisecondi, quando viene chiamata la funzione di integrazione (vedi cap. 4 e 5), il nuovo valore di

$L_i = N \cdot S \cdot \left|\vec{B}\right| \cdot I \cdot \Delta t$ (nuovo poiché modulo del campo magnetico e corrente nel solenoide possono variare e quindi vengono misurati e aggiornati ogni Δt millisecondi). Tale variabile conterrà in ogni istante il risultato dell'integrale, quindi, non appena essa diventa uguale o maggiore del valore di momento angolare richiesto, si cessa di alimentare il solenoide. Tuttavia, per ragioni pratiche di risparmio di memoria, si è preferito utilizzare un approccio decrementale, ovvero ad ogni passo di integrazione viene sottratta la quantità L_i direttamente dal valore memorizzato di momento angolare richiesto, e viene interrotta l'alimentazione del solenoide non appena questo diventa uguale o minore di zero.

2.2.1. Tipi di pilotaggio del solenoide

Come già si può intuire da quanto detto sopra, esistono più tipi alternativi di pilotaggio per il solenoide. Nel progetto in esame si è scelto, al fine di semplificare il più possibile il controllo del driver e gli algoritmi di calcolo, di utilizzare un pilotaggio a corrente costante (idealmente intesa come la massima fornibile al carico), la cui durata sia la minima indispensabile per ottenere il momento angolare richiesto. Pertanto, in questa sede, verranno accennati concisamente i diversi possibili metodi per alimentare il solenoide.

Prima di proseguire, è però necessario fare una precisazione: dal momento che una coppia $\vec{C}$ viene applicata ad un corpo rigido, essa tenderà ad imprimere a quest'ultimo una rotazione attorno all'asse *t*. La direzione di tale asse, come già anticipato nel sottocapitolo 2.2, non è determinabile a priori, bensì dipende sia dalla coppia applicata, nei suoi termini vettoriali di direzione e verso, sia dalla forma del corpo rigido stesso e dalla sua distribuzione delle masse, espresse nei termini della matrice $\underline{\underline{J}}$ (forma tensoriale del momento di inerzia di un corpo). A questo punto si può definire la direzione dell'asse *t* mediante il suo versore:

$$\hat{t} = \left(\vec{C} \cdot \underline{\underline{J}}^{-1}\right) \Big/ \left|\vec{C} \cdot \underline{\underline{J}}^{-1}\right|$$

Conoscendo l'asse di rotazione del satellite in virtù della coppia applicata, se ne può esprimere il momento inerziale nella forma scalare J_t, definita a partire dall'asse *t* come segue:

$$J_t = \hat{t} \cdot \underline{\underline{J}} \cdot \hat{t}^T$$

1) Immaginiamo di voler far compiere al satellite un angolo di rotazione ben definito, in condizioni ideali, senza tenere conto dell'azione di altri controlli di assetto ma solamente basandoci su quello in esame. Osservando la figura 2.8, vediamo come, dopo aver dato inizio alla rotazione applicando una coppia a gradino analoga a quella di figura 2.7 per un tempo T/2, sia necessario fornire una coppia di uguale durata ed intensità (C_{max}) ma opposta di segno, che svolga un'azione frenante nei confronti del satellite, così da far sì che esso si arresti dopo aver descritto esattamente l'angolo desiderato.
   Coi seguenti integrali si ottengono le espressioni della velocità angolare del satellite e della sua posizione angolare:

$$\omega_t = \int_0^T \frac{C}{J_t} dt = \int_0^{T/2} \frac{C_{\max}}{J_t} dt + \int_{T/2}^T \frac{-C_{\max}}{J_t} dt = \left(\frac{C_{\max} \cdot t}{J_t} \Big|_0^{T/2} ; \left(\frac{C_{\max} \cdot (T-t)}{J_t} \Big|_{T/2}^T \right. \right.$$

$$\theta_t = \int_0^T \omega_t \cdot dt = \left(\frac{C_{\max} \cdot t^2}{2J_t} \Big|_0^{T/2} ; \left(\frac{C_{\max} \cdot T}{J_t} \left(t - \frac{t^2}{2T} - \frac{T}{4} \right) \Big|_{T/2}^T \right. \right.$$

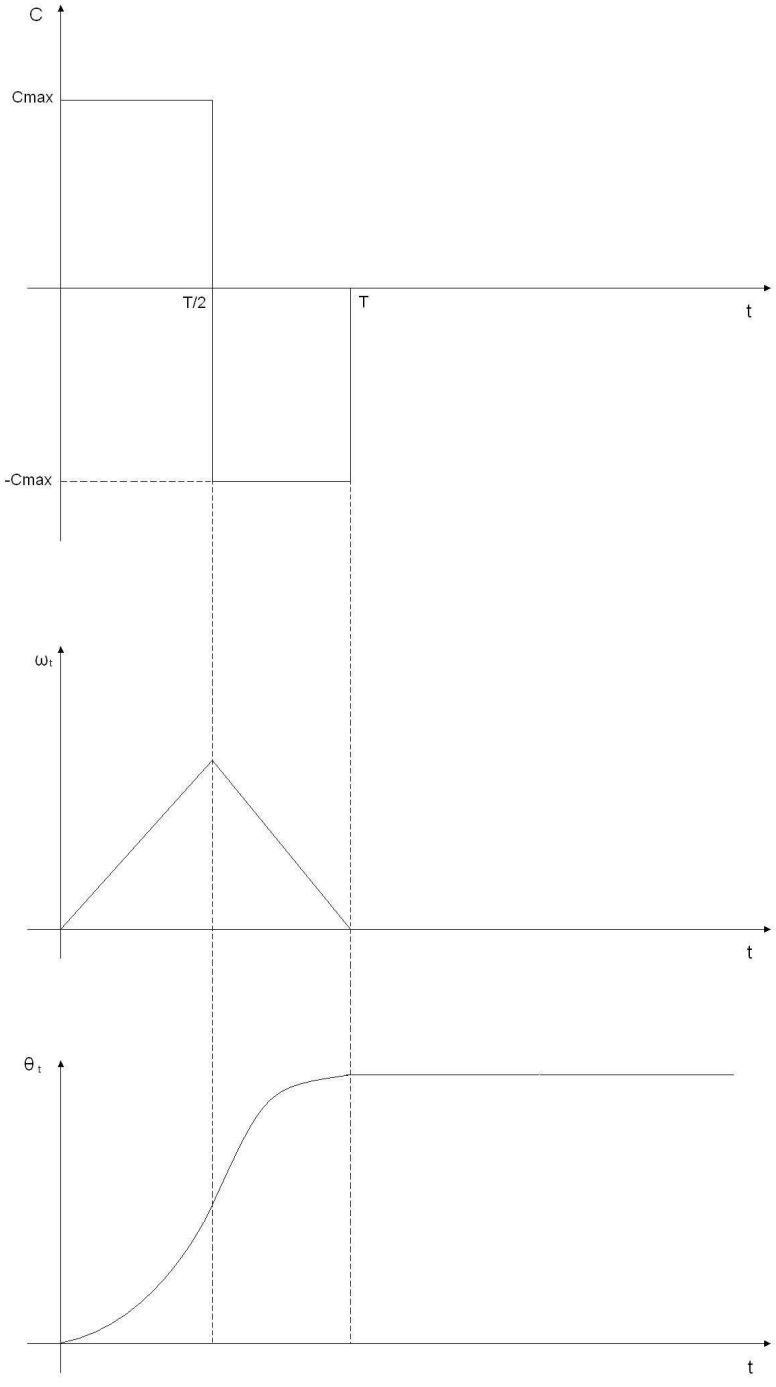


Figura 2.8 Pilotaggio del solenoide per far compiere al satellite una rotazione definita, nel caso ideale

Sostituendo t=T nell'espressione di θ_t, otteniamo l'angolo θ_f percorso dal satellite all'arrestarsi della rotazione, ovvero nell'istante T:

$$\theta_f = \frac{C_{\max} \cdot T^2}{4J_t}$$

Ora, sapendo quale coppia si intende applicare al satellite possiamo conoscere in quanto tempo esso percorrerà un angolo θ_f semplicemente invertendo la formula:

$$T = \sqrt{\frac{4\theta_f J_t}{C_{\max}}}$$

Oppure possiamo sapere qual è la coppia massima che occorre applicare se si vuole far ruotare il satellite di θ_f radianti in un tempo T:

$$C_{\max} = \frac{4\theta_f J_t}{T^2}$$

2) Se invece il satellite fosse già in rotazione con velocità angolare ω_0 e lo si volesse fermare, sarebbe necessario fornire una coppia negativa a gradino ($-C_{max}$), tale che la conseguente velocità angolare ω, che essa cercherebbe di fornire, sia uguale in modulo ma di verso opposto alla velocità di rotazione del satellite (figura 2.9).
In tal modo, la velocità risultante ω_R, inizialmente uguale a quella precedentemente posseduta dal satellite, dopo un certo tempo T, sarà nulla. Facendo uso della relazione tra velocità angolare e coppia applicata, si avrà, per $0 < t < T$:

$$\omega_t = \int_0^T \frac{C}{J_t} dt = \int_0^T \frac{-C_{\max}}{J_t} dt = \frac{-C_{\max}}{J_t} t$$

mentre, per t > T, ω assume il valore costante $\omega_{\max} = -\frac{C_{\max} \cdot T}{J_t}$

Tale valore finale, a cui tenderebbe la velocità se il satellite fosse inizialmente fermo, deve coincidere, in questo caso, con la sua velocità iniziale ω_0. Possiamo quindi ottenere il modulo della coppia negativa che il solenoide deve applicare per arrestare la rotazione del satellite nel tempo T:

$$C_{\max} = \frac{\omega_0 \cdot J_t}{T}$$

o, definita la coppia da applicare, determinare il tempo necessario per fermare la rotazione:

$$T = \frac{\omega_0 \cdot J_t}{C_{\max}}$$

La velocità risultante si può vedere sempre in figura 2.9 e sarà $\omega_R = \omega_0 + \omega_t$.

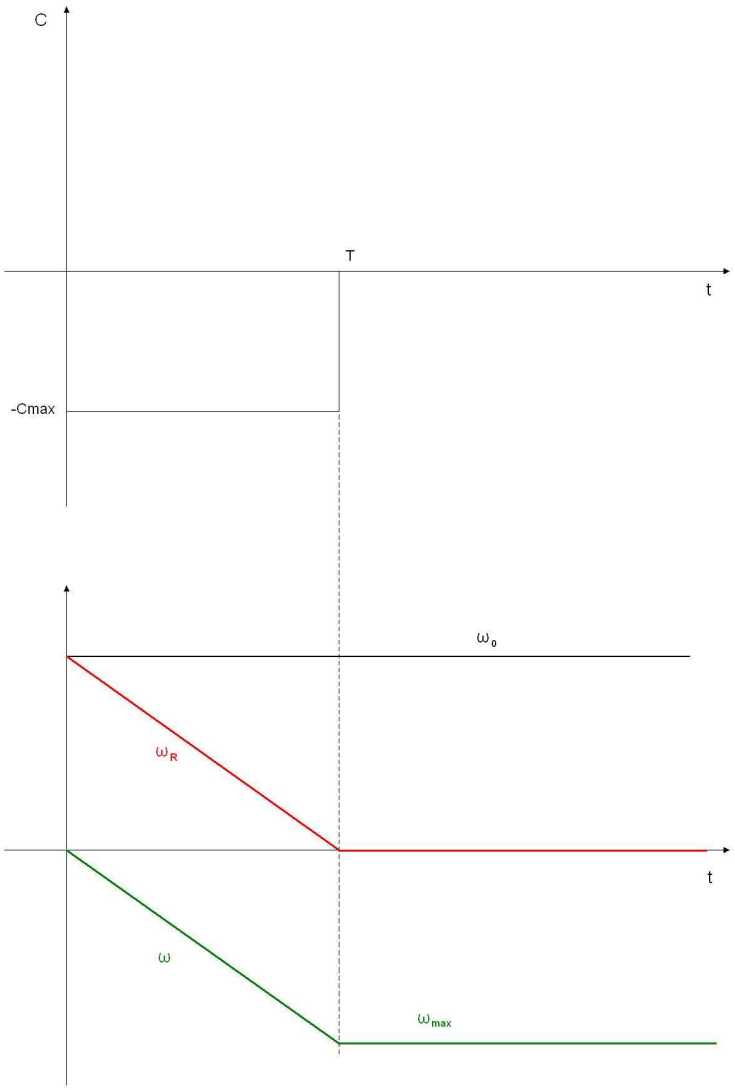


Figura 2.9 Applicazione di una coppia frenante al satellite, nel caso ideale

2.2.2. Dimensionamento del solenoide

Dal momento che il compito di gestire l'assetto e, quindi, l'eventuale rotazione del satellite e il relativo angolo, è assegnato al controllore centrale di assetto (l'attore *Central Attitude Controller* del par. 3.1), non è previsto un caso d'uso specifico, per il singolo sottoblocco, volto a frenare la rotazione di AraMiS, anche perché, essendo presente un sottoblocco identico su ogni faccia del cubo, nella maggior parte dei casi pratici non è assolutamente detto che la rotazione del satellite e l'arresto della medesima dipendano interamente dal solo sottoblocco considerato. Allo stesso tempo, tale azione di frenata è indispensabile, perché, qualora venisse a mancare, il satellite tenderebbe a protrarre la propria rotazione con velocità costante (par. 2.2); perciò si è deciso di contemplare, nell'unico caso d'uso che prevede il pilotaggio del solenoide (*Actuate Coil*, par. 3.2), l'applicazione di una coppia frenante, ovvero di segno negativo (par. 2.2.1), semplicemente attuata dal sistema nel caso in cui il momento angolare richiesto, in qualità di integrale della coppia, abbia anch'esso segno negativo. Questo ha una duplice funzione: oltre a frenare l'eventuale rotazione in atto, rende possibile anche la rotazione in senso opposto.

Come verrà spiegato nel paragrafo 2.3, l'applicazione di un momento angolare negativo consiste nel pilotare il solenoide con una corrente negativa, o, più precisamente, con una corrente positiva che, però, fluisce con verso opposto, percorrendo le spire del solenoide, supponiamo, in senso orario anziché antiorario. Per adesso consideriamo il solenoide un carico induttivo passivo (quale in effetti è) dotato di due connettori, tramite i quali viene connesso in un circuito elettrico. Tale circuito, denominato appunto **driver**, deve pilotare il carico in modo che la corrente vi possa scorrere in ambo i versi (dal connettore 1 al connettore 2 o viceversa).

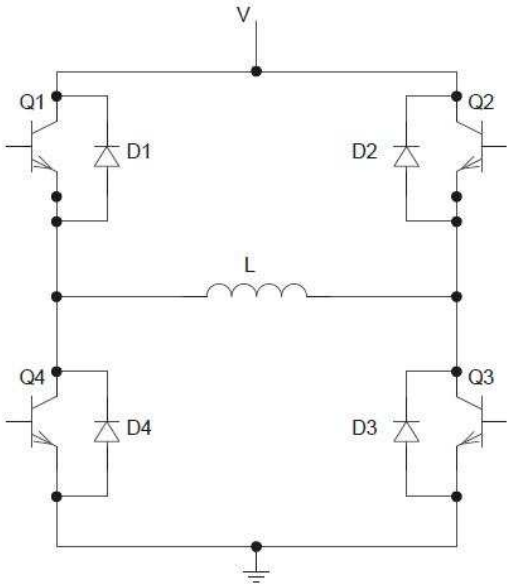


Figura 2.10 Schema elettrico circuito ponte ad H

Il driver dovrà essenzialmente avere la configurazione circuitale denominata **ponte ad H** (figura 2.10), oltre ad includere ovviamente altri circuiti per la logica di controllo, la protezione e quant'altro. Il circuito proposto in figura permette di pilotare il solenoide in modo bidirezionale, tramite la topologia a ponte, e di scaricare la corrente immagazzinata nel medesimo, grazie ai diodi di ricircolo. In figura 2.11, a sinistra, vediamo come, attivando i transistor Q1 e Q3, la corrente scorra da sinistra verso destra nel solenoide, mentre, quando vengono chiusi Q2 e Q4, essa scorre in senso opposto (a destra in figura 2.11).

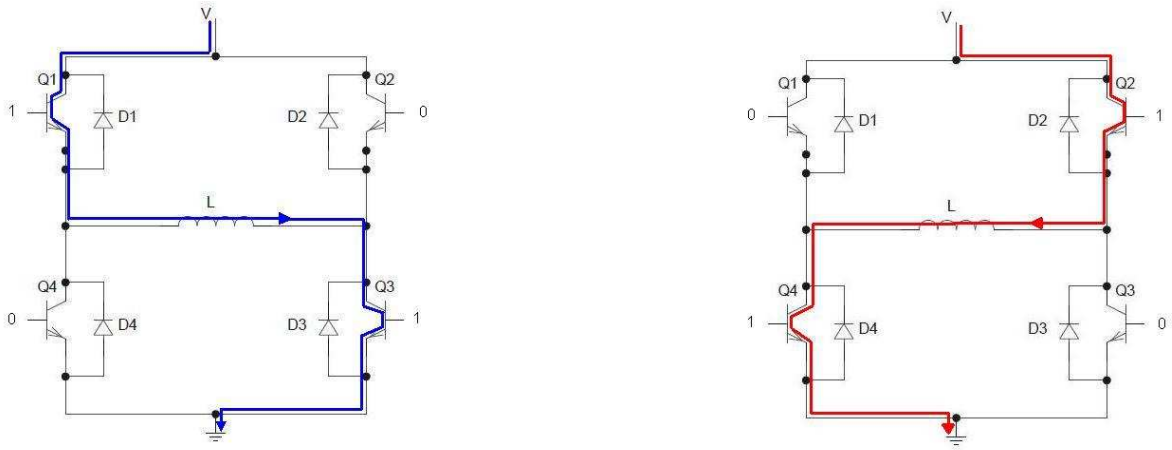


Figura 2.11 Percorsi della corrente nel ponte ad H

Nella figura 2.12 si può osservare il percorso della corrente di scarica attraverso i diodi D2 e D4, una volta che i transistor sono tutti interdetti, nel caso in cui il solenoide sia stato precedentemente pilotato tramite i transistor Q1 e Q3.

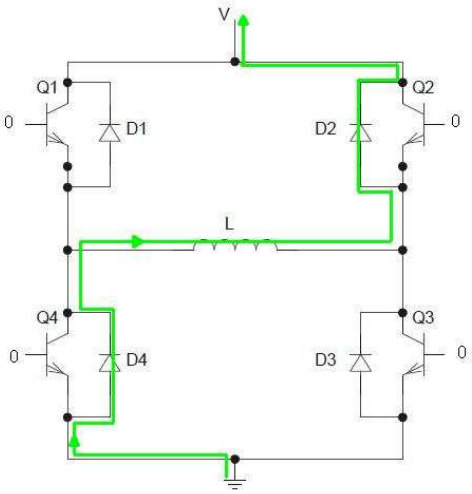


Figura 2.12 Percorso della corrente di scarica nel ponte ad H

Se, invece, il solenoide fosse stato caricato tramite Q2 e Q4, la sua scarica avverrebbe attraverso i diodi D1 e D3.

Ora, facendo riferimento alle situazioni appena descritte con tale modello semplificato del circuito di pilotaggio, possiamo ricavare l'andamento della corrente nel solenoide ed i suoi valori medi, a titolo di esempio, nel caso ideale di pilotaggio con tensione di alimentazione costante. Conducendo un'analisi più approfondita, ma con procedimenti analoghi, è possibile ricavare equazioni relative a pilotaggi di tipo più complesso, che d'altra parte esulano dal compito di questo progetto, per il quale l'aspetto prevalente di questa sede consiste nel determinare valori medi e massimi delle grandezze riguardanti il solenoide, affinché si possa correttamente effettuare il dimensionamento del medesimo e di tutta la componentistica del blocco di attuazione magnetica, per la quale si rimanda al sottocapitolo 6.2.

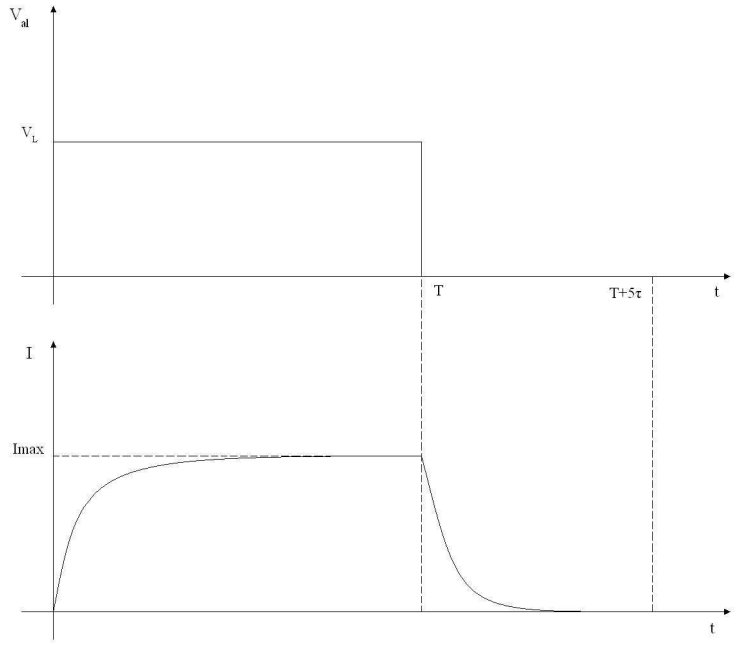


Figura 2.13 Carica del solenoide a tensione costante e scarica

L'andamento della corrente nel solenoide, confrontato con quello della tensione di alimentazione, è riportato in figura 2.13, dove si indica con T il tempo di pilotaggio (ovvero di carica), e si assume un successivo tempo di scarica pari a 5τ.
In figura 2.14 (pagina seguente) possiamo vedere il circuito equivalente del ponte ad H e del solenoide, per t < T, dove R_{ds_on} e R_{sol} rappresentano rispettivamente il contributo resistivo di ogni MOS in conduzione e quello del solenoide.

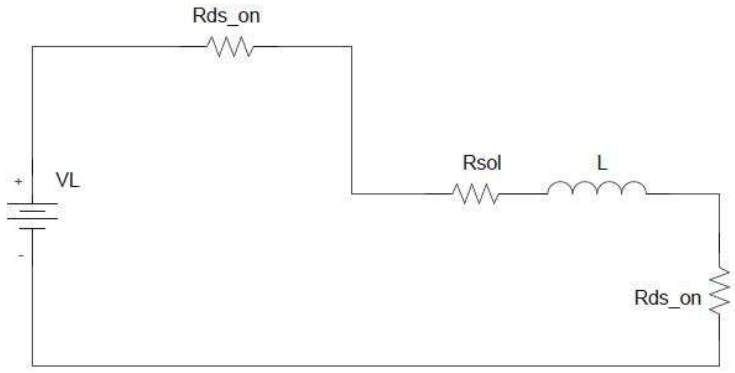


Figura 2.14 Circuito equivalente di ponte H pilotante un induttore

Indicando con R_{TOT_r} la resistenza equivalente serie, pari a $2R_{ds\_on} + R_{sol}$, si ricava il circuito equivalente di figura 2.15, con un induttore (il solenoide) in serie ad una resistenza.

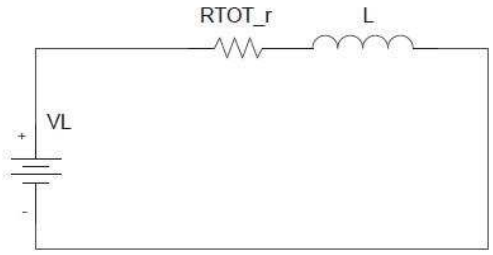


Figura 2.15 Circuito equivalente semplificato di ponte H con induttore

Pertanto l'espressione dell'andamento della corrente per t < T (graficamente rappresentato in fig. 2.13) sarà:

$$I(t \le T) = I_{\max} - I_{\max} \exp\left(\frac{-t}{\tau_r}\right)$$

dove $I_{\max} = V_L / R_{TOT\_r}$ è la massima corrente che scorre in tale circuito e $\tau_r = L / R_{TOT\_r}$ rappresenta la costante di tempo per la carica dell'induttore. Ora possiamo ricavare, integrando dall'istante 0 all'istante T e dividendo per il tempo T, la corrente media che fluisce nel solenoide nell'intervallo [0,T]:

$$\bar{I}_{rise} = \frac{1}{T}\int_0^T I(t)dt = I_{\max} - \frac{I_{\max}\left[-\tau_r \cdot \exp\left(-T/\tau_r\right) + \tau_r\right]}{T}$$

Il circuito equivalente per la scarica (t > T) sarà analogo a quello per la carica, ma la resistenza in serie al solenoide sarà $R_{TOT\_f} = 2R_D + R_{sol}$, dove RD è la resistenza di un diodo in conduzione. Procedendo in modo analogo si può ricavare l'espressione della corrente per t > T:

$$I(t > T) = I_{\max} \exp\left(\frac{-t}{\tau_f}\right)$$

dove $\tau_f = L/R_{TOT\_f}$ è la costante di tempo per la scarica. Quindi si avrà:

$$\bar{I}_{fall} = \frac{1}{T}\int_0^{5\tau_f} I(t > T)dt = \frac{\left[I_{\max} - I_{\max}\exp\left(-T/\tau_f\right)\right]\cdot\left[-\tau_f \exp(-5) + \tau_f\right]}{T}$$

Sapendo che τ_r e τ_f sono trascurabili poiché quasi nulli, le due correnti medie, che il solenoide deve essere in grado di sopportare, diventano:

$$\bar{I}_{rise} \cong I_{\max} = \frac{V_L}{R_{TOT\_r}}$$

$$\bar{I}_{fall} \cong 0$$

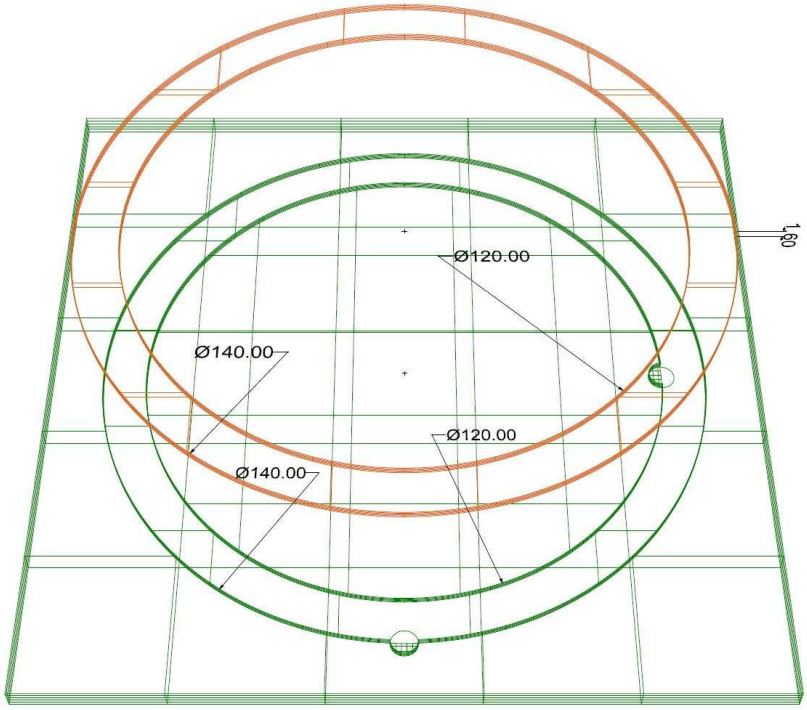


Figura 2.16 Schema del solenoide utilizzato per il controllo di assetto magnetico di AraMiS

Il solenoide che si è scelto di utilizzare, il cui prospetto è visibile in figura 2.16, è costituito da un filo di rame di diametro 0.25 mm avvolto in 180 spire circolari, disposte su diversi strati sovrapposti (per un'altezza totale di 1.6 mm) ed aventi diametro compreso tra 120 e 140 mm. Pertanto è possibile ricavare tutti i dati relativi al solenoide di cui abbiamo bisogno, ricordando la formula che permette di calcolare la resistenza di un filo conduttore:

$$R = \rho \cdot \frac{l}{A}$$

dove l rappresenta la lunghezza del filo, A la sua sezione e ρ la resistività del materiale di cui è costituito (nel nostro caso il rame, avente $\rho=1.7\cdot10^{-8}\ \Omega$m).
Quindi, considerando un diametro medio delle spire di 130 mm, la lunghezza della singola spira (in metri) sarà:

$$\zeta = \pi \cdot 0.13m \cong 0.4084m$$

da cui si ottiene la lunghezza dell'intero solenoide:

$$l = 180 \cdot \zeta \cong 73.51m$$

La sezione del filo è: $$A = \left(0.00025m/2\right)^2 \cdot \pi \cong 4.909 \cdot 10^{-8} m^2$$

Sostituendo i valori sopra calcolati nella formula della resistenza, si ottiene il risultato teorico $R^*_{sol} \cong 25.46\Omega$, che poco differisce dal valore reale della componente resistiva del solenoide, misurato in laboratorio: $R_{sol} = [27.92 \div 28.26]\Omega$ (la maggior parte delle misurazioni forniva come risultato esattamente 28 Ω; tale valore sarà pertanto considerato quello tipico).
La tensione di alimentazione del solenoide viene ricavata da quella del Power Distribution Bus (PDB) di AraMiS, che può variare da 12 V DC a 18V DC (valore nominale 14 V). Di logica conseguenza, i valori limite della corrente nel solenoide saranno:

$$I_{\max} = \frac{V_{\max}}{R_{\min}} = \frac{18V}{27.92\Omega} \cong 644.7mA$$
$$I_{\min} = \frac{V_{\min}}{R_{\max}} = \frac{12V}{28.26\Omega} \cong 424.6mA$$

Dal momento che, in orbita, molteplici fattori, come variazioni termiche della resistenza del solenoide o errori di qualsivoglia natura nell'alimentazione, potrebbero far sì che il valore reale della corrente si discosti da questi limiti teorici, è prevista, durante la procedura di supervisione per individuare eventuali errori ed agire di conseguenza (si veda il caso d'uso *Supervision* nel cap. 3 e il relativo diagramma di sequenza nel cap. 5), una certa tolleranza sul valore minimo e massimo di corrente durante il pilotaggio del solenoide.
Infine, sapendo che la sezione (media) delle spire del solenoide vale:

$$S = \pi \cdot \left(0.13m/2\right)^2 \cong 13.27 \cdot 10^{-3} m^2$$

e andando a sostituire nella (5), ricaviamo i valori del modulo del dipolo magnetico ottenibili alimentando il solenoide con la tensione PDB:

$$D_{\max} = N \cdot S \cdot I_{\max} \cong 1.54 Am^2$$
$$D_{\min} = N \cdot S \cdot I_{\min} \cong 1.014 Am^2$$

(5)

Quindi, in condizioni totalmente ideali di corrente di alimentazione costante e campo magnetico tipico di 0.266 G, la coppia massima che sarà possibile fornire alla faccia del satellite quando essa si trova posizionata parallelamente alle linee di campo magnetico terrestre, ovvero quando l'asse del solenoide montato su di essa è perpendicolare a queste (fig. 2.17) sarà:

$$C_{\max} = D_{\max} \cdot \left|\vec{B}\right|_{typ} \cong 1.54 Am^2 \cdot 2.66 \cdot 10^{-5} T \cong 41 \mu Nm$$

Va però nuovamente ricordato che l'intensità della coppia applicabile dipende, oltre che dalle dimensioni del solenoide, dall'inclinazione della tile rispetto al campo magnetico terrestre, dal campo stesso e dalla corrente. Essendo questi ultimi elementi variabili nel tempo, ha senso solamente parlare di valori massimi e minimi ideali, e, soprattutto, istantanei, di tale coppia, poiché, nell'applicazione pratica, non possiamo conoscerne il modulo durante un determinato intervallo di tempo, ma se ne può calcolare il valore attuale in base alle misurazioni effettuate. Tale valore viene utilizzato nel calcolo del modulo del momento angolare $\vec{L}$ tramite sommatoria (equazione (4)), nell'intervallo medesimo di tempo in cui la coppia viene applicata. Terminato tale intervallo, termina anche il calcolo del momento angolare, nonché il pilotaggio del solenoide.

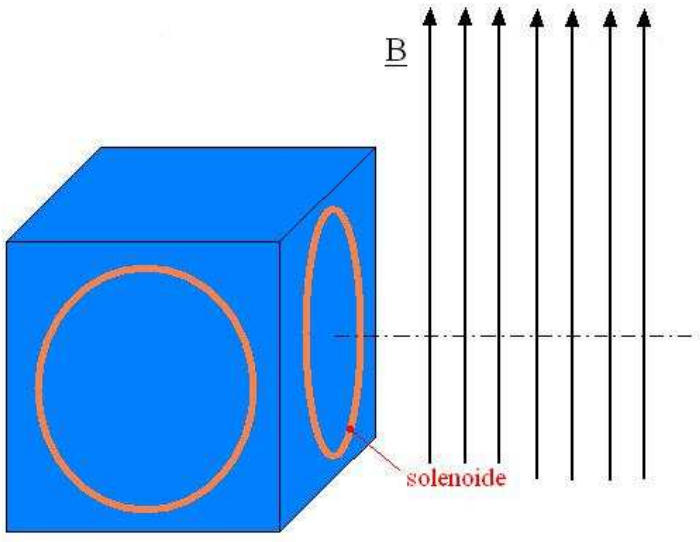


Figura 2.17 Configurazione di coppia massima applicabile alla singola tile

2.3. Sistema di riferimento del progetto

Un'ultima considerazione, prima di entrare nel dettaglio del progetto, va fatta sul sistema di riferimento scelto per le facce (*tiles*) del satellite AraMiS, che deve necessariamente essere univoco e comune a tutti i sottoblocchi che costituiscono il satellite, così da non lasciare adito ad errori od incomprensioni nella definizione delle grandezze misurate e immagazzinate nei dati di telemetria.

Ogni i-esima faccia di AraMiS definisce il proprio sistema di riferimento secondo i tre assi cartesiani, x, y e z, come mostrato in figura 2.18. In tal modo ciascuna faccia del satellite avrà un sistema di riferimento analogo a quello delle altre facce, ma ruotato a seconda dell'orientazione dell'i-esima faccia: l'asse z_i è sempre perpendicolare al piano contenente la faccia e positivo uscente da essa, mentre x_i e y_i, entrambi contenuti nel piano della faccia, sono perpendicolari a z_i e fra di loro e sono disposti secondo la terna destrorsa definita in figura.

Di conseguenza, per un satellite di forma cubica, saranno presenti sei riferimenti distinti $x_1,y_1,z_1;\ldots; x_6,y_6,z_6$ e il sottosistema di controllo di assetto di ogni i-esima tile si baserà esclusivamente sul sistema di riferimento ad essa associato, mentre sarà solamente il blocco del controllore d'assetto (1B2) a "conoscere" l'esistenza di più sistemi di riferimento e gestire le operazioni dei vari sottoblocchi tramite l'algoritmo di controllo d'assetto, al fine di portare avanti un'azione complessiva, rispetto al riferimento globale di AraMiS, ancora in fase di definizione.

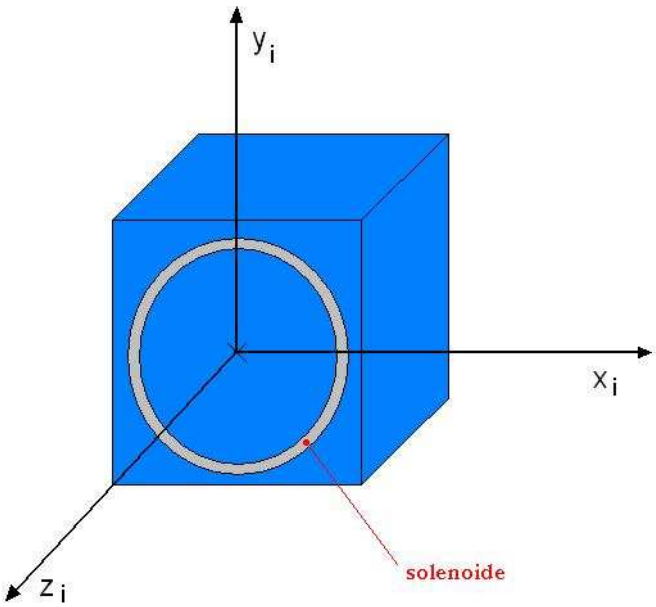


Figura 2.18 Sistema di riferimento per l'i-esima faccia del satellite AraMiS

Come già spiegato nel paragrafo 2.2.1, il dipolo generato dal solenoide ha la sola componente z, ovvero giace sull'asse z_i della tile, proprio perché il solenoide stesso giace sul piano x_i,y_i. Anche il sensore di campo magnetico giace sullo stesso piano, perciò si evince che sarebbe inutile misurare la componente z del campo, poiché, nel prodotto vettoriale da cui si origina la coppia fornita dal singolo solenoide, B_z darebbe un contributo nullo, andando a moltiplicare l'unica componente del dipolo, D_z, a cui essa è parallela. Le componenti B_x e B_y sono le uniche a fornire una coppia alla faccia, quindi le uniche che si debbano misurare.

Ora è chiaro che, per ogni faccia dotata di sistema di controllo di assetto, l'asse del solenoide montato su di essa, e, quindi, la direzione del vettore dipolo magnetico, coincidono con l'asse z_i. Il dipolo $\vec{D}$, a seconda che la corrente scorra in senso orario o antiorario nel solenoide (osservando la faccia dall'esterno del satellite), avrà verso positivo rispettivamente entrante nella faccia o uscente da essa, ovvero, rispettivamente, discorde o concorde al verso di z_i. Di conseguenza, allo scorrere di tale corrente, il dipolo tenderà ad allinearsi alle linee di campo magnetico (considerabili pressoché rette) e il verso della rotazione effettuata dal satellite nel compiere tale allineamento dipenderà dal verso di pilotaggio del solenoide. Secondo la definizione di **momento angolare positivo** richiesto alla tile (si vedano le specifiche di progetto nel cap. 3), si può affermare che:

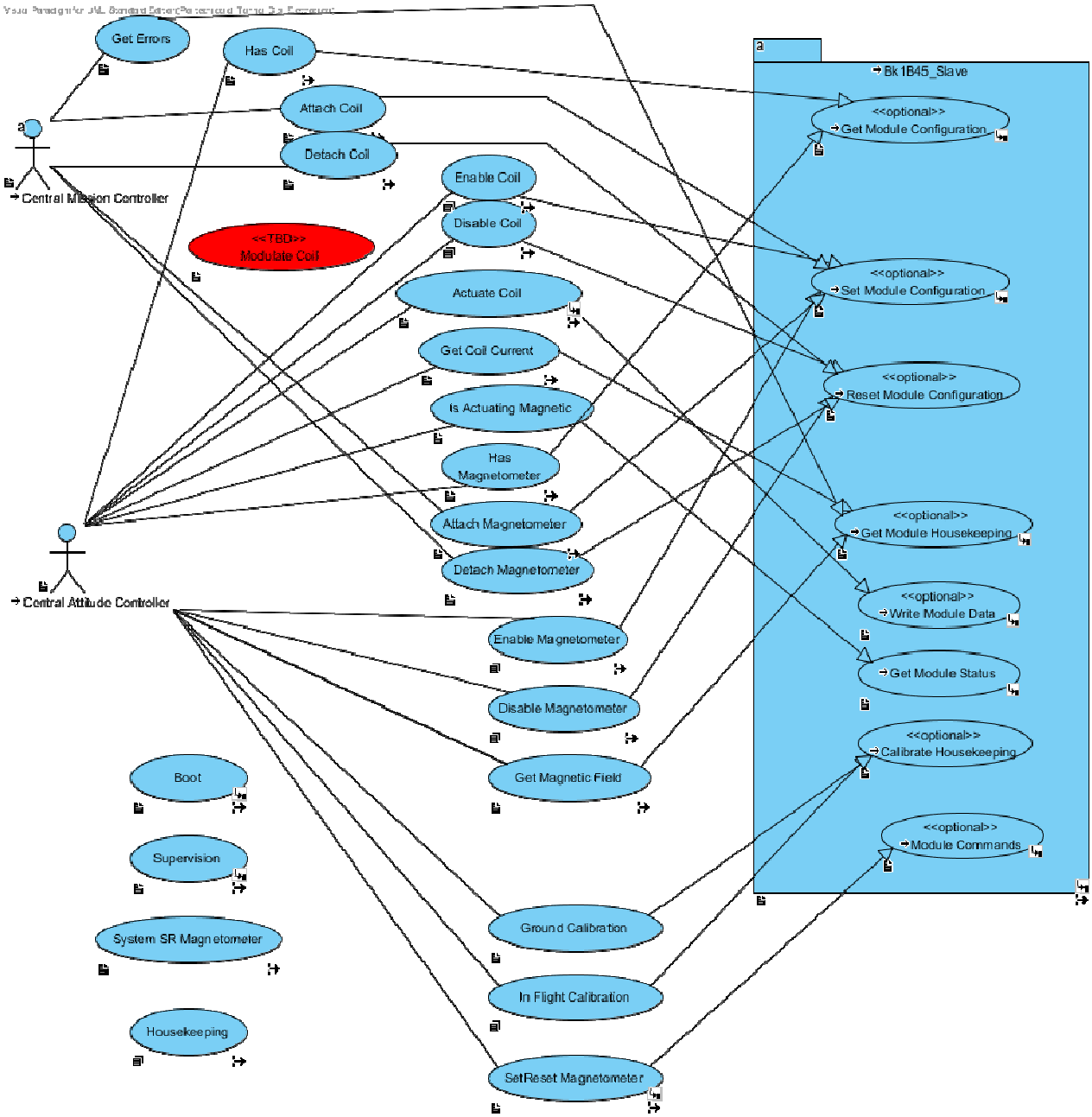
- un momento angolare $\vec{L}$ **positivo** tende ad avvicinare l'asse z della tile alle linee di campo magnetico; in questo caso il dipolo ha verso uscente dalla tile e perciò la corrente nel solenoide (visto dall'esterno della faccia) scorre in senso **antiorario**;
- $\vec{L}$ **negativo** tende ad allontanare l'asse z della tile dalle linee di campo magnetico; in questo caso il dipolo ha verso entrante nella tile e perciò la corrente nel solenoide (visto dall'esterno della faccia) scorre in senso **orario**.

Di conseguenza, il solenoide verrà montato sulla tile, ed i suoi due connettori collegati ai piedini di uscita del driver, in modo da rispettare tale convenzione.

Capitolo 3

3. Bk1B22 Magnetic Attitude Subsystem - Specifiche di progetto

This section details the use cases of the magnetic attitude subsystem of the ARAMIS architecture. These define the functional specifications of the project.



3.1. Descrizione degli attori

Name	*Documentation*
Central Mission Controller	The entity (likely a SW routine running on the OBC) in charge of satellite supervision, fault detection and management and emergency management.
Central Attitude Controller	The entity (likely a SW routine running on the OBC in charge of managing the 1B2 Attitude Control Subsystem in nominal operation. Fault and emergency handling are left to the Central Mission Controller.

3.2. Documentazione dei Casi d'Uso

Name	*Documentation*
::Get Magnetic Field	The System returns the last measured value of magnetic field along x and y axes both as t_sensor (signed 16-bits integers) in units of SENS_MAGNETIC_FIELD, respectively: x-axis (housekeeping**[**MAGNETIC_FIELD_X**]**) y-axis (housekeeping**[**MAGNETIC_FIELD_Y**]**) Magnetic field along z-axis is NOT available. The Central Attitude Controller will read magnetic field components as vector housekeeping using the Get Module Housekeeping use case of Bk1B45_Slave package.
::Actuate Coil	If the coil is enabled (see use case Enable Coil) and the magnetometer is enabled (see use case Enable Magnetometer), it generates a user-defined angular momentum by means of the magnetic coil actuator. The momentum is generated around an axis perpendicular to both the z axis and the external magnetic field. Positive momentum will push z axis towards the direction of the magnetic field. If previous command is still executing, the previous command is canceled and the new command is executed. If angular momentum is zero, the coil is stopped. Angular momentum is t_sensor (signed 16-bits integer) in units of 1 gcm2/s. Uses Write Module Data use case of Bk1B45_Slave package, with the CMD_ACTUATE_COIL command with a 2-byte message containing the angular momentum with MSB last. If either the coil is not enabled (see use case Disable Coil) or the magnetometer is not enabled (see use case Disable Magnetometer), no action is carried on.
::Enable Magnetometer	Enables magnetometer (giving power supply and periodically sampling its output). When enabled, the magnetometer power consumption should be as little as possible. To enable magnetometer, the Master shall set

	ENABLE_MAGNETOMETER bit of Configuration Word t_ConfigWord_1B2 using Set Module Configuration use case of Bk1B45_Slave package.
::Has Magnetometer	Returns the presence of the magnetic sensor in the system. This Use Case makes use of Get Module Configuration use case of Bk1B45_Slave package, looking at bit HAS_MAGNETOMETER of mechanical configuration word t_MechanicalConfiguration_1B8 of Bk1A_1B8_Codes package.
Bk1B45_Slave::Module Commands	The CPU actor sends a data-less command to the **System**. The CPU can use up to 8 different Module Commands, to issue as many commands to the system. It uses the Command Only use case, by issuing the CMD_COMMAND_0 through CMD_COMMAND_7, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which commands are issued, but the Designer-defined type of command. The Designer can use as many message types he wants. This use case is optional, therefore the Configurator shall #define an identifier whose name is contained in the tagged value **define**. After command has been received and checked, the 1B45 package calls the Command Interpreter.interpret operation and passes the command as an argument. The Designer shall then write his command interpretation routine inside the interpret operation.
::Disable Magnetometer	Disables magnetic sensor (removing power supply). When disabled, the magnetometer shall draw negligible power from the Power Bus. The **Master** shall reset ENABLE_MAGNETOMETER bit of Configuration Word t_ConfigWord_1B2 using Reset Module Configuration use case of Bk1B45_Slave package. By default it is disabled.
::Has Coil	Returns the presence of the magnetic coil in the system. This Use Case makes use of Get Module Configuration use case of 1B45 package, looking at bit HAS_MAGNETIC_COIL of mechanical configuration word t_MechanicalConfiguration_1B8 of Bk1A_1B8_Codes package.
::In Flight Calibration	Allows to change value of magnetometer offset on both axes (x,y), using two signed 16-bits integers (type t_sensor). Unit is SENS_MAGNETIC_FIELD Uses Calibrate Housekeeping use case of Bk1B45_Slave package, with: • magnetometer offset on x-axis in field offset of MAGNETIC_FIELD_X_SCALE element of scaling vector • magnetometer offset on y-axis in field offset of MAGNETIC_FIELD_Y_SCALE element of scaling vector
::Get Coil Current	The System returns the last measured value of current flowing into coil as t_sensor (signed 16-bits integer) in units of SENS_COIL_CURRENT:

	(housekeeping[COIL_CURRENT]) The Central Attitude Controller will read coil current as variable t_sensor housekeeping using the Get Module Housekeeping use case of Bk1B45_Slave package.
::Attach Magnetometer	Flags magnetic sensor as available to the user. Magnetometer is considered "attached" if it is physically present on the tile (flag HAS_MAGNETOMETER of t_MechanicalConfiguration_1B8) and it has not been detached (see use case Detach Magnetometer). If the magnetometer is attached and enabled (see use case Enable Magnetometer), any further action of magnetic sensor will be carried on. This Use Case makes use of Set Module Configuration use case of 1B45 package, with bit ATTACH_MAGNETOMETER of configuration word t_ConfigWord_1B2. The difference between Attach Magnetometer and Enable Magnetometer is that the former has effects only if the magnetometer is physically present on the tile, while the latter has effects only if the magnetometer is attached, therefore the former enables the latter. The Attach Magnetometer use case is intended only in rare occasions, to recover from an erroneous Detach Magnetometer operation.
::Disable Coil	Disables magnetic coil (magnetic actuator). Any further action of coil will be disregarded. If the coil is in process of generating torque while the Disable Coil use case is applied, the coil is turned off immediately. When disabled, the coil shall draw negligible power from the Power Bus. The **Master** shall reset ENABLE_MAGNETIC_COIL bit of Configuration Word t_ConfigWord_1B2 using Reset Module Configuration use case of Bk1B45_Slave package. By default it is disabled.
::Attach Coil	Flags coil as available to the user. Coil is considered "attached" if it is physically present on the tile (flag HAS_MAGNETIC_COIL of t_MechanicalConfiguration_1B8) and it has not been detached (see use case Detach Coil). If the coil is attached and enabled (see use case Enable Coil), any further action of magnetic actuator will be carried on. This Use Case makes use of Set Module Configuration use case of 1B45 package, with bit ATTACH_MAGNETIC_COIL of configuration word (t_ConfigWord_1B2) The difference between Attach Coil and Enable Coil is that the former has effects only if the coil is physically present on the tile, while the latter has effects only if the coil is attached, therefore the former enables the latter. The Attach Coil use case is intended only in rare occasions, to recover from an erroneous Detach Coil operation.
::Detach Coil	Flags magnetic coil as unavailable to the user. Coil is considered "detached" either if it is NOT physically present on the tile (flag HAS_MAGNETIC_COIL of t_MechanicalConfiguration_1B8)

	or it has not been attached (see use case Attach Coil). If the coil is either detached or disabled (see use case Disable Coil), no further action of magnetic actuator can be carried on. This Use Case makes use of Reset Module Configuration use case of 1B45 package, with bit (ATTACH_MAGNETIC_COIL) of configuration word (t_ConfigWord_1B2). The difference between Detach Coil and Disable Coil is that the former prevents the coil from being enabled, therefore it disables any further action on the coil, including Enable Coil. The Detach Coil use case is intended only in rare occasions, to permanently disable a faulty coil. Since the actor can know if the coil is attached (see use case Has Coil), the ARAMIS-level AOCS algorithms can be adapted consequently.
::Enable Coil	Enables magnetic coil (magnetic actuator). Any further action of coil will be carried on regularly. By default the coil is disabled. When enabled, the coil shall be active only when requested by the appropriate use cases (see Actuate Coil). When enabled but not active, power consumption should be as little as possible. To enable the magnetic coil actuator, the Master shall set ENABLE_MAGNETIC_COIL bit of Configuration Word t_ConfigWord_1B2 using Set Module Configuration use case of Bk1B45_Slave package.
::Detach Magnetometer	Flags magnetic sensor as unavailable to the user. Magnetometer is considered "detached" either if it is NOT physically present on the tile (flag HAS_MAGNETOMETER of t_MechanicalConfiguration_1B8) or it has not been attached (see use case Attach Magnetometer). If the magnetometer is either detached or disabled (see use case Disable Magnetometer), no further action of magnetic sensor can be carried on. This Use Case makes use of Reset Module Configuration use case of 1B45 package, with bit ATTACH_MAGNETOMETER of configuration word t_ConfigWord_1B2. The difference between Detach Magnetometer and Disable Magnetometer is that the former prevents the magnetometer from being enabled, therefore it disables any further action on the magnetometer, including Enable Magnetometer. The Detach Magnetometer use case is intended only in rare occasions, to permanently disable a faulty magnetometer. Since the actor can know if the magnetometer is attached (see use case Has Magnetometer), the ARAMIS-level AOCS algorithms can be adapted consequently.
::Is Actuating Magnetic	Checks whether the coil is being driven or not. Uses Get Module Status use case of Bk1B45_Slave package, which returns the status of the magnetic coil as ACTIVE_COIL of the status word t_StatusWord_1B2. The flag is true if the driver is feeding the coil and false if the coil is off.
::SetReset Magnetometer	Resets magnetometer by applying a Set/Reset sequence to magnetically reset the magnetometer, as indicated in data sheet, provided that magnetometer is enabled.

	The Master shall issue a CMD_SETRESET_MAGNETOMETER command using Module Commands use case of Bk1B45_Slave package.
::System SR Magnetometer	Periodically resets the magnetometer. This action shall be done automatically, if the magnetometer is enabled, every 10min to avoid the drift of the sensor and calculate the output offset term.
::Ground Calibration	Allows to correct measuring and drift errors which afflict sensitivity of Bi-axial Magnetic Field Sensor block and Magnetic Torque Actuator block and/or cause the presence of offset terms in both blocks' output readings. The correction is operated exclusively in software by using calibration parameters which are stored in scaling vector of Housekeeping class in Bk1B45_Slave package. Calibration parameters are obtained on ground by measuring known values (in particular coil current and magnetic field), scaled in standard units (see Get Coil Current and Get Magnetic Field use cases) and written in MAGNETIC_FIELD_X_SCALE, MAGNETIC_FIELD_Y_SCALE, COIL_CURRENT_SCALE positions of **scaling** vector. The offset value, contained in t_scaling structure type, is subtracted from ADC output, then result is multiplied by gain value of the same structure, to obtain the correct measure. Uses Calibrate Housekeeping use case of Bk1B45_Slave package.
::Housekeeping	Periodically calls housekeeping operation of Bk1B229_Controller class, which: • acquires magnetic field on X-axis, magnetic field on Y-axis, coil current values and writes them, respectively, in MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y, COIL_CURRENT positions of housekeeping vector of Housekeeping class of Bk1B45_Slave package. • calculates magnetic torque integration steps using integrate operation of Bk1B229_Controller class • manages SetReset Magnetometer and System SR Magnetometer use cases with a frequency of TIMER_FREQ of TimerA class of Common package. Parameter *index* returns on the same value every SENSOR_SAMPLE_TIME of t_Configuration_1B8 class of Bk1A_1B8_Codes package.
Bk1B45_Slave::Set Module Configuration	Sets to one some bits of the internal **configuration** word which contains some **Designer**-defined (TBD) configuration parameters (if any). The **Designer** shall define all configurable parameters (if any) in the type **t_Configuration**. This use case is different from the **Configure Module** use case, as the latter allows a compile-time configuration, while the former allows run-time configuration changes (if foreseen). The **Master** shall send to the Slave (using the **Write Data** use case with **CMD_SET_CONFIGURATION** command) as many bits as are in the internal **configuration** word. Each bit which is set to one in the **Master** message will set to one the corresponding bit in the internal **configuration** word. The other bits are untouched.

	Note that: • the use case **Reset Module Configuration** will reset to zero a few bits in the **configuration** word. It cannot set any bit to one. • the use case Set Module Configuration will set to one a few bits in the configuration word. It cannot reset any bit to zero. • the use case Write Module Configuration will write (either zero or one) all the bits of the configuration word. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**
Bk1B45_Slave::Reset Module Configuration	Resets to zero some bits of the internal **configuration** word which contains some **Designer**-defined (TBD) configuration parameters (if any). The **Designer** shall define all configurable parameters (if any) in the type **t_Configuration**. This use case is different from the **Configure Module** use case, as the latter allows a compile-time configuration, while the former allows run-time configuration changes (if foreseen). The **Master** shall send to the Slave (using the **Write Data** use case with **CMD_RESET_CONFIGURATION** command) as many bits as are in the internal **configuration** word. Each bit which is set to one in the **Master** message will reset to zero the corresponding bit in the internal **configuration** word. The other bits are untouched. Note that: • the use case **Reset Module Configuration** will reset to zero a few bits in the **configuration** word. It cannot set any bit to one. • the use case Set Module Configuration will set to one a few bits in the configuration word. It cannot reset any bit to zero. • the use case Write Module Configuration will write (either zero or one) all the bits of the configuration word. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
Bk1B45_Slave::Get Module Housekeeping	Returns last measured housekeeping data (see use case Module Housekeeping for details). It uses the Read Data use case by issuing the CMD_GET_HOUSEKEEPING command. This use case is optional, therefore the Configurator shall activate it, in the global header file **platform.h**, by **#define** an identifier whose name is contained in the tagged value **define** of this use case.
Bk1B45_Slave::Get Module Status	It returns to the **CPU** actor: • the 16-bit **status word** of the module. Status word content is application-dependent, although a few bits are predefined. Details are found and are to be defined in the **t_StatusWord** type. • the 8-bit word containing **last error**. Last error content is application-dependent, although a few error codes are predefined. Details are found

	and are to be defined in the **t_LastError** type. It also clears the **Error Indicator** signal. It uses the **SPI protocol - Read Data** use case by issuing the **CMD_GET_STATUS** command. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
Bk1B45_Slave::Write Module Data	The CPU actor sends up to 256B of Designer-defined data to the **System**. The CPU can use up to 8 different Write Module Data commands, to issue messages to as many different subsystems. It uses the Write Data use case, by issuing the CMD_WRITE_DATA_0 through CMD_WRITE_DATA_7 command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are written, but the Designer-defined type (therefore the destination subsystem). The Designer can use as many message types he wants. This use case is optional, therefore the Configurator shall #define an identifier whose name is contained in this use case's tagged value **define**. After command has been received completely and checked, the 1B45 package will allocate a buffer inside the Buffers class, then call the Command Interpreter.interpret operation and pass the command as an argument. The Designer shall then write his command interpretation routine inside the interpret operation. The command interpreter shall access the data message from class Buffers with the following sequence: 1. Call Buffers.get, which returns a pointer to the buffer containing the message and its length (number of bytes in data section, excluding the command itself and other ancillary data. 2. Use the message from the above vector 3. When the message is not used any more, release the buffer using Buffers.release
Bk1B45_Slave::Calibrate Housekeeping	Writes into the Slave calibration data for housekeeping sensors (if any). For each sensor, two values are to be supplied: an **unsigned t_sensor offset** and an **unsigned t_sensor gain_correction**. Both parameters are of **unsigned t_sensor** type. Values are written into FLASH memory. The **Module Housekeeping** use case shall then write into the **housekeeping** vector: **for (i=0; i<LENGTH_CORRECTION); i++)** **housekeeping[i] = (acquired_value[i] - offset[i]) * (gain_correction[i] / (max(t_sensor)/2));**

	where: **t_sensor housekeeping[i]** is the i-th value written in the housekeeping vector; **t_sensor acquired_value[i]** is the i-th value (unsigned integer) acquired from the i-th sensor **t_sensor offset[i]** is the i-th offset value **t_sensor gain_correction[i]** is the i-th gain_correction value **max(t_sensor)** is the max value which can be expressed by an (unsigned t_sensor) Only the first **LENGTH_CALIBRATION** sensors (from 0 to **LENGTH_CALIBRATION-1**) are calibrated. For all the other sensors, the acquired value is written into the **housekeeping** vector directly. Also if this use case is not implemented, the acquired value is written into the **housekeeping** vector directly. The default values for the offset and gain_correction parameters are set by the **Tester** (in the factory) after appropriate calibration. These parameters can be later updated by the Master**.** This use case is different from the **Set Module Configuration** and the **Configure Module** use cases: **Configure Module** allows a compile-time configuration, which cannot be changed during the life of the **System**; **Set Module Configuration** allows run-time configuration changes (if foreseen) but these do without affecting sensor calibration; **Calibrate Housekeeping** allows run-time (or post-fabrication) changes of sensor calibration parameters; This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
Bk1B45_Slave::Get Module Configuration	The **CPU** actor reads the module ID and the current configuration from the **System**. The configuration parameters shall be defined by the **Designer** in the in the type **t_Configuration**. Similarly the ID shall be defined by the **Designer** in the **StatusFlags** class. It uses the **SPI Protocol - Read Data** use case, by issuing the **CMD_GET_ID** command. This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
::Get Errors	Returns the complete list of errors which may afflict the Magnetic Attitude Subsystem and are detected by Supervision use case. Errors are listed by type as bits composing the error word hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS], respecting the following order: ON_CURR_EXCESS ->when magnetic actuator is operating, coil current exceeds maximum allowed value ON_CURR_LACK ->when magnetic actuator is operating, coil current is less than minimum functional value

	OFF_CURR_EXCESS ->when magnetic actuator is disabled, there is a significant coil current ERR_COIL_ENABLE ->error which stucks bit ENABLE_MAGNETIC_COIL at 'true' ERR_COIL_STATUS ->error which stucks bit ACTIVE_COIL at 'true' ERR_COIL_ATTACH ->error which stucks bit ATTACH_MAGNETIC_COIL at 'true' ERR_MAGN_ENABLE ->error which stucks bit ENABLE_MAGNETOMETER at 'true' ERR_MAGN_ATTACH ->error which stucks bit ATTACH_MAGNETOMETER at 'true' LOW_PDB_VOLT ->error in Power Distribution Bus Voltage: voltage is below minimum critical value This use case makes use of Get Module Housekeeping use case of Bk1B45_Slave package.
::Supervision	This use case is automatically executed at most every 1s. Detects errors in the Bk1B222_Magnetic_Torque_Actuator Subsystem by checking: 1. If coil is enabled and active, measured current (COIL_CURRENT field of housekeeping vector) shall be >Min_value_on and <Max_value_on. If it is >Max_value_on, then bit ON_CURR_EXCESS of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set and the coil is immediately disabled. If <Min_value_on, then bit ON_CURR_LACK of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. 2. If coil is either detached or disabled or not active (see below), current shall be <Max_value_off (should be 0 but there is a tolerance due to noise and/or disturbing field currents). If not, then bit OFF_CURR_EXCESS of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. If current is >Max_value_off, the coil is immediately detached. 3. If bit ATTACH_MAGNETIC_COIL of t_ConfigWord_1B2 configuration word is false, then both bit ENABLE_MAGNETIC_COIL of t_ConfigWord_1B2 and bit ACTIVE_COIL of t_StatusWord_1B2 shall be false too. If ENABLE_MAGNETIC_COIL is true, then bit ERR_COIL_ENABLE of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. If ACTIVE_COIL is true, then bit ERR_COIL_STATUS of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. 4. If bit ATTACH_MAGNETIC_COIL is true and bit ENABLE_MAGNETIC_COIL is false, then bit ACTIVE_COIL shall be false. If not, then bit ERR_COIL_STATUS of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. 5. If Magnetic Actuator is not present on the tile (bit HAS_MAGNETIC_COIL of class t_MechanicalConfiguration_1B8 is false), then bit ATTACH_MAGNETIC_COIL shall be false. If not, then bit ERR_COIL_ATTACH of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. Detects errors in the Bk1B221_Magnetometer_Sensor Subsystem by checking: 1. If bit ATTACH_MAGNETOMETER of t_ConfigWord_1B2

	configuration word is false, then bit ENABLE_MAGNETOMETER of t_ConfigWord_1B2 shall be false too. If not, then bit ERR_MAGN_ENABLE of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. 2. If magnetic sensor is not present on the tile (bit HAS_MAGNETOMETER of class t_MechanicalConfiguration_1B8 is false), then bit ATTACH_MAGNETOMETER of t_ConfigWord_1B2 shall be false. If not, then bit ERR_MAGN_ATTACH of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set. Detects lack of Power Distribution Bus voltage: if hk[PDB_VOLTAGE] is <CRITICAL_PDB, then bit LOW_PDB_VOLT of hk.housekeeping[MAGNETIC_ACTUATOR_ERRORS] is set and coil and magnetometer are immediately disabled.
::Modulate Coil	For MITOR.... to be updated If the coil is enabled (see use case Enable Coil) and the magnetometer is enabled (see use case Enable Magnetometer), it generates a user-defined angular momentum (kg m2 / s) by means of the magnetic coil actuator. The momentum is generated around an axis perpendicular to both the z axis and the external magnetic field. Positive momentum will push z axis towards the direction of the magnetic field. If previous command is still executing, the previous command is canceled and the new command is executed. If angular momentum is zero, the coil is stopped. Angular momentum is t_sensor (signed 16-bits integer) in units of 1 gcm2/s. Uses Write Module Data use case of Bk1B45_Slave package, with the CMD_ACTUATE_COIL command with a 2-byte message containing the angular momentum with MSB firs/last (TBD). If either the coil is not enabled (see use case Disable Coil) or the magnetometer is not enabled (see use case Disable Magnetometer), no action is carried on.
::Boot	At bootstrap, the System will initialize all the required elements. In particular: • disable magnetometer (like Disable Magnetometer) • disable coil (like Disable Coil) • initialize notSET and RESET signals to 'high' • initialize scaling[MAGNETIC_FIELD_X_SCALE], scaling[MAGNETIC_FIELD_Y_SCALE], scaling[COIL_CURRENT_SCALE] to ground calibration stored values • properly initialize flags of Bk1B229_Controller

Capitolo 4

4. Descrizione del sistema – Diagrammi delle classi

In questo capitolo verranno descritti, sotto forma di diagrammi degli oggetti (si veda il par. 1.4.2), tutti gli aspetti del sistema di controllo d'assetto magnetico, mettendo in risalto la suddivisione del medesimo in blocchi e sottoblocchi di natura hardware (oggetti di colore giallo), software (colore verde) o misti, ovvero contenenti elementi sia hardware che software. I principali elementi costituenti il sistema sono:

- Bk1B22_Magnetic_Attitude_Subsystem : modulo globale di controllo d'assetto magnetico, comprendente tutti gli elementi atti a garantire tale controllo; è il livello gerarchico superiore nel progetto 1B22 (sottocapitolo 4.1)

- Bk1B221_Magnetometer_Sensor : blocco del magnetometro (sottocapitolo 4.2)

- Bk1B222_Magnetic_Torque_Actuator : blocco del solenoide (sottocapitolo 4.3)

- Bk1B229_Controller : oggetto software costituente l'insieme di funzioni di controllo e gestione del sistema, variabili utilizzate nel codice e riferimenti a classi di altri progetti, necessarie al funzionamento del sistema stesso. (sottocapitolo 4.4)

Come spiegato nel capitolo introduttivo, ogni oggetto conterrà, in qualità di *attributi*, le specifiche delle scelte progettuali effettuate e le associazioni (fisiche o concettuali) ad altri oggetti, ed in qualità di *metodi*, le funzioni che verranno eseguite dal microcontrollore, se si tratta di un oggetto SW, oppure i segnali elettrici (analogici o digitali), se si tratta di un oggetto HW, oppure entrambe le cose, nel caso di oggetti misti.
Per una più esaustiva comprensione delle scelte progettuali in ambito hardware e delle relative specifiche e per visionare gli schemi elettrici del progetto, si rimanda al cap. 6, mentre il codice delle funzioni software sopra citate si può trovare nel cap. 5 (Progetto Software).
In ciascuno dei sottocapitoli seguenti, oltre a rappresentare il diagramma degli oggetti considerato, verranno descritti, in modo dettagliato, attributi e metodi degli oggetti maggiormente rilevanti, mediante la relativa documentazione UML. Per ogni nuovo diagramma verranno descritti tutti gli oggetti in esso contenuti (e gli elementi che fanno parte di tali oggetti), purché questi abbiano particolare importanza nell'operatività del sistema in esame (come ad esempio alcune classi del progetto 1A, contenente il codice comune a tutto il progetto AraMiS) o non siano gli stessi già contenuti in precedenti diagrammi, e, pertanto, già descritti.
Anche in questo caso si è utilizzato il report generato dal tool *Visual Paradigm*.

4.1. Bk1B22_Magnetic_Attitude_Subsystem

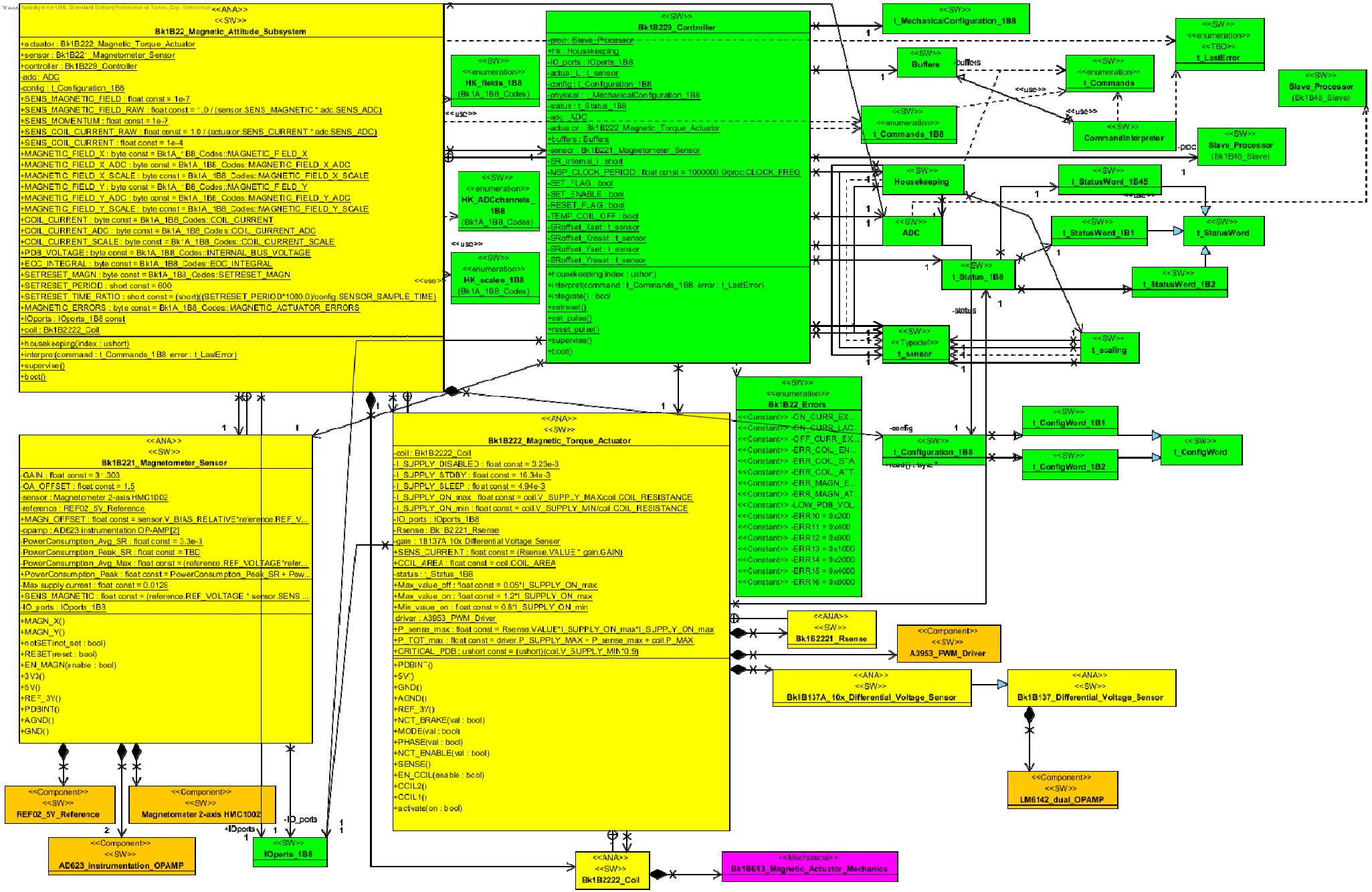
This class diagram represents the Bk1B22 Magnetic Attitude Subsystem, which is the global project of the magnetic attitude control system for the AraMiS satellite.
It contains:

- Class **Bk1B22_Magnetic_Attitude_Subsystem** indicates the external shell of the magnetic attitude subsystem project and contains: attributes which are constants, used internally or by other classes, and references to its subclasses or to other elements. (Attributes will be described subsequently); operations that refer to the ones of class **Bk1B229_Controller**, used for testing purpose.
- Hardware class **Bk1B221_Magnetometer_Sensor**, which represents magnetic sensor block, and its subclasses (described in par. 4.2).
- Hardware class **Bk1B222_Magnetic_Torque_Actuator**, which represents magnetic actuator block, and its subclasses (described in par. 4.3).
- Software class **Bk1B229_Controller**, which represents the magnetic attitude control management unit of the project 1B22, and its subclasses (described, as well as the Controller, in par. 4.4).

It uses:

- Class **HK_fields_1B8** of Bk1A_1B8_Codes package, which contains codes identifying the locations of Power Management Tile housekeeping vector values.
- Class **HK_ADCchannels_1B8** of Bk1A_1B8_Codes package, which contains ADC channels' numbers for housekeeping values acquiring of 1B8 Power Management Tile.

All classes in this diagram have been used to compile software elements (see next chapter).


Bk1B22_Magnetic_Attitude_Subsystem
Bk1B229_Controller
Bk1B221_Magnetometer_Sensor
Bk1B222_Magnetic_Torque_Actuator
t_MechanicalConfiguration_1B8
t_LastError
Buffers
t_Commands
Slave_Processor (Bk1B45_Slave)
t_Commands_1B8
CommandInterpreter
Housekeeping
t_StatusWord_1B45
ADC
t_StatusWord_1B1
t_StatusWord
t_Status_1B8
t_StatusWord_1B2
t_sensor
t_scaling
Bk1B22_Errors
t_Configuration_1B8
t_ConfigWord_1B1
t_ConfigWord_1B2
t_ConfigWord
Bk1B2221_Rsense
A3953_PWM_Driver
Bk1B137A_10x_Differential_Voltage_Sensor
Bk1B137_Differential_Voltage_Sensor
LM6142_dual_OPAMP
REF02_5V_Reference
Magnetometer Z-axis HMC1002
AD623_instrumentation_OPAMP
IOports_1B8
Bk1B2222_Coil
Bk1B613_Magnetic_Actuator_Mechanics
HK_fields_1B8 (Bk1A_1B8_Codes)
HK_ADCchannels_1B8 (Bk1A_1B8_Codes)
HK_scales_1B8 (Bk1A_1B8_Codes)

4.1.1. Class Bk1B22_Magnetic_Attitude_Subsystem

Magnetic Attitude Control Subsystem. Includes the Bk1B222_Magnetic_Torque_Actuator block, the Bk1B221_Magnetometer_Sensor block and the Bk1B229_Controller.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
actuator (public)	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B222_Magnetic_Torque_Actuator	
sensor (public)	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B221_Magnetometer_Sensor	
controller (public)	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B229_Controller	
adc (private) The processor ADC converter of 1B45 Common package.	Common.ADC	
config (private)	Bk1A_1B8_Codes$t_Configuration_1B8	
SENS_MAGNETIC_FIELD (public) Sensitivity of SW-calibrated MAGN_X and MAGN_Y signals from Bk1B221_Magnetometer_Sensor, in T/LSB.	float	1e-7
SENS_MAGNETIC_FIELD_RAW (public) Sensitivity of raw MAGN_X and MAGN_Y signals from Bk1B221_Magnetometer_Sensor, after ADC, in T/LSB.	float	1.0 / (sensor.SENS_MAGNETIC * adc.SENS_ADC)
SENS_MOMENTUM (public) Sensitivity of angular momentum, in (kg m2 /s)/LSB.	float	1e-7
SENS_COIL_CURRENT_RAW (public) Sensitivity of raw SENSE signals from Bk1B222_Magnetic_Torque_Actuator, after ADC, in A/LSB.	float	1.0 / (actuator.SENS_CURRENT * adc.SENS_ADC)
SENS_COIL_CURRENT (public) Sensitivity of SW-calibrated SENSE signals from Bk1B222_Magnetic_Torque_Actuator, in A/LSB.	float	1e-4
MAGNETIC_FIELD_X (public) Value of the index associated (for housekeeping and other purposes) to x-axis component of magnetic field. These	byte	Bk1A_1B8_Codes::MAGNETIC_FIELD_X

values are enumerated in HK_fields_1B8 class of Bk1A_1B8_Codes package.		
MAGNETIC_FIELD_X_ADC (public) The ADC channel number associated to x-axis component of magnetic field. These values are enumerated in HK_ADCchannels_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::MAGNETIC_FIELD_X_ADC
MAGNETIC_FIELD_X_SCALE (public) The number of Housekeeping.scaling vector position associated (for calibration purpose) to x-axis component of magnetic field. These values are enumerated in class HK_scales_1B8 of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::MAGNETIC_FIELD_X_SCALE
MAGNETIC_FIELD_Y (public) Value of the index associated (for housekeeping and other purposes) to y-axis component of magnetic field. These values are enumerated in HK_fields_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::MAGNETIC_FIELD_Y
MAGNETIC_FIELD_Y_ADC (public) The ADC channel number associated to y-axis component of magnetic field. These values are enumerated in HK_ADCchannels_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::MAGNETIC_FIELD_Y_ADC
MAGNETIC_FIELD_Y_SCALE (public) The number of Housekeeping.scaling vector position associated (for calibration purpose) to y-axis component of magnetic field. These values are enumerated in class HK_scales_1B8 of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::MAGNETIC_FIELD_Y_SCALE
COIL_CURRENT (public) Value of the index associated (for housekeeping and other purposes) to coil current reading. These values are enumerated in HK_fields_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::COIL_CURRENT
COIL_CURRENT_ADC (public) The ADC channel number associated to coil current acquiring. These values are enumerated in HK_ADCchannels_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::COIL_CURRENT_ADC
COIL_CURRENT_SCALE (public) The number of Housekeeping.scaling vector position associated (for calibration purpose) to coil current value. These values are enumerated in class HK_scales_1B8 of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::COIL_CURRENT_SCALE
PDB_VOLTAGE (public) Value of the index associated (for housekeeping and other purposes) to the Power Distribution Bus voltage. These values are enumerated in HK_fields_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::INTERNAL_BUS_VOLTAGE
EOC_INTEGRAL (public) Value of the index associated to Magnetic Angular Momentum calculation. It represents occurrence of each Magnetic Torque integral step. These values are enumerated in HK_fields_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::EOC_INTEGRAL
SETRESET_MAGN (public) Value of the index associated to automatic Magnetometer Sensor Set/Reset timing. These values are enumerated in	byte	Bk1A_1B8_Codes::SETRESET_MAGN

HK_fields_1B8 class of Bk1A_1B8_Codes package.		
SETRESET_PERIOD (public) The repetition rate of the System SR Magnetometer, in s.	short	600
SETRESET_TIME_RATIO (public) Represents the number of iterations of a given housekeeping index (which occurr every SENSOR_SAMPLE_TIME ms) necessary to reach the specified SETRESET_PERIOD.	short	(short)((SETRESET_PERIOD*1000.0)/config.SENSOR_SAMPLE_TIME)
MAGNETIC_ERRORS (public) Attribute that represents the value of the index associated to Magnetic Attitude Subsystem error word, contained in housekeeping vector of Housekeeping class of Bk1B45_Slave package. These values are enumerated in HK_fields_1B8 class of Bk1A_1B8_Codes package.	byte	Bk1A_1B8_Codes::MAGNETIC_ACTUATOR_ERRORS
IOports (public)	Bk1A_1B8_Codes$IOports_1B8	
coil (public)	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B22_Magnetic_Torque_Actuator$Bk1B2222_Coil	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
housekeeping (public) Calls controller.housekeeping operation.	index : ushort	
interpret (public) Calls controller.interpret operation.	command : Bk1A_1B8_Codes$t_Commands_1B8 error : t_LastError	
supervise (public) Calls controller.supervise operation.		
boot (public) Calls controller.boot operation.		

4.1.2. Class : HK_fields_1B8

Stereotypes :	<<SW>> <<enumeration>>
Documentation :	Codes for housekeeping of 1B8 Power management Tile. Each code identifies the location of the corresponding value in the 1B8 housekeeping vector

Attribute : MAGNETIC_FIELD_X

Stereotypes :	<<Constant>>
Documentation :	HK code for measured magnetic field along **x-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project).
Scope :	instance

Attribute : MAGNETIC_FIELD_Y

Stereotypes :	<<Constant>>
Documentation :	HK code for measured magnetic field along **y-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project). MAGNETIC_FIELD_Y **MUST** be equal to MAGNETIC_FIELD_X+1.
Scope :	instance

Attribute : COIL_CURRENT

Stereotypes :	<<Constant>>
Documentation :	HK code for measured current in magnetic asset coil. Associated with use case **Get Housekeeping** (see 1B8 project). COIL_CURRENT **MUST** be either less than MAGNETIC_FIELD_X-2 or greater than MAGNETIC_FIELD_Y+3, otherwise COIL_CURRENT is measured while it is disabled to allow measuring MAGNETIC_FIELD_X and MAGNETIC_FIELD_Y.
Scope :	instance

Attribute : EOC_INTEGRAL

Stereotypes :	<<Constant>>
Documentation :	This is not an index to the housekeeping field, but it is used to index the computation of the angular momentum of the magnetic coil (with reference to command CMD_ACTUATE_COIL)
Scope :	instance

Attribute : SETRESET_MAGN

Stereotypes :	<<Constant>>
Documentation :	This is not an index to the housekeeping field, but it is used to index the automatic set reset operation of the 1B22 Magnetic Attitude Subsystem.
Scope :	instance

Attribute : MAGNETIC_ACTUATOR_ERRORS

Stereotypes :	<<Constant>>

Scope :	instance

Attribute : INTERNAL_BUS_VOLTAGE

Stereotypes :	<<Constant>>
Scope :	instance

4.1.3. Class : HK_ADCchannels_1B8

Stereotypes :	<<SW>> <<enumeration>>
Documentation :	Codes for housekeeping of 1B8 Power management Tile. Each code identifies the location of the corresponding value in the 1B8 housekeeping vector

Attribute : MAGNETIC_FIELD_X_ADC

Stereotypes :	<<Constant>>
Documentation :	HK code for measured magnetic field along **x-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project). Associated with block 1B22 Magnetic Attitude Subsystem.
Type :	byte
Initial Value :	3
Scope :	instance

Attribute : MAGNETIC_FIELD_Y_ADC

Stereotypes :	<<Constant>>
Documentation :	HK code for measured magnetic field along y-**axis**. Associated with use case **Get Magnetic Field** (see 1B8 project). Associated with block 1B22 Magnetic Attitude Subsystem.
Type :	byte
Initial Value :	2
Scope :	instance

Attribute : COIL_CURRENT_ADC

Stereotypes :	<<Constant>>
Documentation :	HK code for measured current in magnetic attitude actuator (coil). Associated with use case **Get Housekeeping** (see 1B8 project). Associated with use case **Get Housekeeping** (see 1B8 project). Associated with block 1B22 Magnetic Attitude Subsystem.
Type :	byte
Initial Value :	0
Scope :	instance

4.1.4. Class : HK_scales_1B8

Stereotypes :	<<SW>> <<enumeration>>
Documentation :	Codes for housekeeping of 1B8 Power management Tile. Each code identifies the location of the corresponding value in the 1B8 housekeeping vector

Attribute : MAGNETIC_FIELD_X_SCALE

Stereotypes :	<<Constant>>
Documentation :	HK code for measured magnetic field along **x-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project).
Initial Value :	3
Scope :	instance

Attribute : MAGNETIC_FIELD_Y_SCALE

Stereotypes :	<<Constant>>
Documentation :	HK code for measured magnetic field along **y-axis**. Associated with use case **Get Magnetic Field** (see 1B8 project).
Initial Value :	4
Scope :	instance

Attribute : COIL_CURRENT_SCALE

Stereotypes :	<<Constant>>
Documentation :	HK code for measured current in magnetic asset coil. Associated with use case **Get Housekeeping** (see 1B8 project).
Initial Value :	6
Scope :	instance

4.2. Bk1B221_Magnetometer_Sensor

This class diagram represents the Bk1B221 Magnetometer Sensor block. In the magnetic attitude control system, it is the part of the circuitry that measures and returns value of magnetic field along x and y axes, which is used to calculate angular momentum or for housekeeping purpose.
It contains:

- Class **Bk1B221_Magnetometer_Sensor**, which identifies the shell of magnetic sensor block and includes some specification attributes, associations with its subclasses and hardware operations identifying the I/O electric signals of this block.
- Hardware component class **Magnetometer 2-axis HMC1002**, which is the magnetic field sensor and includes its specifications (by means of the attributes) and I/O signals (by means of the operations).
- Hardware component class **REF02_5V_Reference**, including some specifications, which gives supply voltage to the sensor HMC1002; its operations identify I/O pins.
- Hardware component class **AD623_instrumentation_OPAMP**, including some specifications, used for magnetic field signals conditioning.
- Software class **IOports_1B8** of Bk1A_1B8_Codes package, which contains constants identifying ports and bits numbers associated to 1B8 Power Management Tile digital signals as shown in figure (for now TBD).

All attributes and operations of the classes above (except IOports_1B8) are described subsequently in this paragraph.


<<ANALOG>>
<<SW>>
Bk1B221_Magnetometer_Sensor
-GAIN : float const = 31.303
-OA_OFFSET : float const = 1.5
-sensor : Magnetometer 2-axis HMC1002
-reference : REF02_5V_Reference
+MAGN_OFFSET : float const = sensor.V_BIAS_RELATIVE*reference.REF_VOLTAGE*GA...
-PDB_MAX : short const = 18
-P_REF_MAX : float const = 4.5e-3
-opamp : AD623 instrumentation OP-AMP[2]
-PowerConsumption_Avg_SR : float const = 3.3e-3
-PowerConsumption_Peak_SR : float cons: = TBD
-PowerConsumption_Avg_Max : float cons: = (reference.REF_VOLTAGE*reference.REF_V...
+PowerConsumption_Peak : float const = PowerConsumption_Peak_SR + PowerConsumpt...
-Max supply current : float const = 0.0128
+SENS_MAGNETIC : float const = (reference.REF_VOLTAGE * sensor.SENS * GAIN
-IO_ports : IOports_1B8
+MAGN_X()
+MAGN_Y()
+noiSet(not_sat : bool)
+RESET(reset : bool)
+EN_MAGN(enable : bool)
+3V3()
+5V()
+REF_3V()
+PDBINT()
+AGND()
+GND()
-IO_ports
1
<<SW>>
IOports_1B8
+Bk1B112_PWM_PORT : byte const* = &P1OUT
+Bk1B112_PWM_BIT : byte cons: = BIT1
+Bk1B112_DISABLE_PORT : byte cons:* = &P1OUT
+Bk1B112_DISABLE_BIT : byte const = BIT2
+Bk1B113_FAST_CHARGE_PORT : byte cons:* = &P2OUT
+Bk1B113_FAST_CHARGE_BIT : byte const = BIT0
+Bk1B113_NOT_BAT_CHARGED_PORT : byte const* = &P2OUT
+Bk1B113_NOT_BAT_CHARGED_BIT : byte const = BIT2
+Bk1B113_NOT_LATCHUP_ON_PORT : byte cons:* = &P2OUT
+Bk1B113_NOT_LATCHUP_ON_BIT : byte const = BIT4
+Bk1B113_NOT_SHUTDOWN_PORT : byte const* = &P2OUT
+Bk1B113_NOT_SHUTDOWN_BIT : byte cons: = BIT1
+Bk1B113_DISABLE_PORT : byte cons:* = &P2OUT
+Bk1B113_DISABLE_BIT : byte const = BIT3
+Bk1B114_ENABLE_EQUALIZER_PORT : byte const* = &P2OUT
+Bk1B114_ENABLE_EQUALIZER_BIT : byte const = BIT4
+Bk1B115_ENABLE_SHUNT_PORT : byte const* = &P2OU
+Bk1B115_ENABLE_SHUNT_BIT : byte const = BIT4
+Bk1B117_A_RX_PORT : byte const* = &P2OUT
+Bk1B117_A_RX_BIT : byte const = BIT4
+Bk1B117_A_TX_PORT : byte const* = &P2OUT
+Bk1B117_A_TX_BIT : byte cons: = BIT4
+Bk1B117_B_RX_PORT : byte const* = &P2OUT
+Bk1B117_B_RX_BIT : byte const = BIT4
+Bk1B117_B_TX_PORT : byte const* = &P2OUT
+Bk1B117_B_TX_BIT : byte cons: = BIT4
+Bk1B117_DEPLOYED_PORT : byte const* = &P2OUT
+Bk1B117_DEPLOYED_BIT : byte cons: = BIT4
+Bk1B117_B_ENABLE_PORT : byte const* = &P2OUT
+Bk1B117_B_ENABLE_BIT : byte const = BIT4
+Bk1B118_ENABLE_PORT : byte const* = &P2OUT
+Bk1B118_ENABLE_BIT : byte const = BIT4
+Bk1B121_A_ENABLE_A_PORT : byte const* = &P2OUT
+Bk1B121_A_ENABLE_A_BIT : byte const = BIT4
+Bk1B121_A_ENABLE_B_PORT : byte const* = &P2OUT
+Bk1B121_A_ENABLE_B_BIT : byte const = BIT4
+Bk1B121_B_ENABLE_A_PORT : byte const* = &P2OUT
+Bk1B121_B_ENABLE_A_BIT : byte const = BIT4
+Bk1B121_B_ENABLE_B_PORT : byte const* = &P2OUT
+Bk1B121_B_ENABLE_B_BIT : byte const = BIT4
+Bk1B411_A_RX_PORT : byte const* = &P2OUT
+Bk1B411_A_RX_BIT : byte const = BIT4
+Bk1B411_A_TX_PORT : byte const* = &P2OUT
+Bk1B411_A_TX_BIT : byte cons: = BIT4
+Bk1B411_B_RX_PORT : byte const* = &P2OUT
+Bk1B411_B_RX_BIT : byte const = BIT4
+Bk1B411_B_TX_PORT : byte const* = &P2OUT
+Bk1B411_B_TX_BIT : byte cons: = BIT4
+Bk1B211_ENABLE_PORT : byte const* = &P1OUT
+Bk1B211_ENABLE_BIT : byte const = BIT3
+Bk1B212_ENABLE_PORT : byte const* = &P1OUT
+Bk1B212_ENABLE_BIT : byte const = BIT3
+Bk1B212_PWM_PORT : byte const* = &P1OUT
+Bk1B212_PWM_BIT : byte cons: = BIT3
+Bk1B212_FWD_PORT : byte cons:* = &P1OUT
+Bk1B212_FWD_BIT : byte const = BIT3
+Bk1B212_NOT_BRAKE_PORT : byte const* = &P1OUT
+Bk1B212_NOT_BRAKE_BIT : byte const = BIT0
+Bk1B221_NOT_SET_PORT : byte const* = &P1OUT
+Bk1B221_NOT_SET_BIT : byte const = BIT1
+Bk1B221_RESET_PORT : byte const* = &P3OUT
+Bk1B221_RESET_BIT : byte const = BIT3
+Bk1B221_EN_MAGN_PORT : byte const* = &P3OUT
+Bk1B221_EN_MAGN_BIT : byte const = BIT1
+Bk1B222_EN_COIL_PORT : byte cons:* = &P3OUT
+Bk1B222_EN_COIL_BIT : byte const = BIT2
+Bk1B222_NOT_ENABLE_PORT : byte const* = &P2OUT
+Bk1B222_NOT_ENABLE_BIT : byte const = BIT6
+Bk1B222_MODE_PORT : byte const* = &P2OUT
+Bk1B222_MODE_BIT : byte const = BIT4
+Bk1B222_PHASE_PORT : byte const* = &P1OUT
+Bk1B222_PHASE_BIT : byte const = BIT0
+Bk1B222_NOT_BRAKE_PORT : byte const* = &P2OUT
+Bk1B222_NOT_BRAKE_BIT : byte const = BIT5
+Bk1B22_CALIBRATION_PORT : byte const* = &P3OUT
+Bk1B22_CALIBRATION_BIT : byte const = BIT0
-sensor
-reference
<<Component>>
<<SW>>
Magnetometer 2-axis HMC1002
+Manufacturer : String const = Honeywell
+Part number : String const = HMC1002
+SENS : float const = 32.0
+V_SUPPLY_MAX : float const = 12
+R_BRIDGE : float const = 850
+R_BRIDGE_MIN : float const = 600
+R_BRIDGE_MAX : float const = 1200
+V_BIAS_RELATIVE : float const = -0.001875
+V_OFFSET_RELATIVE : float const = 0.005625
+OUT + (A)()
+OUT - (A)()
+OUT + (B)()
+OUT - (B)()
+S/R+B()
+S/R-B()
+S/R+A()
+S/R-A()
+VbridgeA()
+VbridgeB()
+ACND()
<<Component>>
<<SW>>
REF02_5V_Reference
+Manufacturer : String const = Analog Devices
+Model : String const = REF02
+REF_VOLTAGE : float const = 5.0
+TOLERANCE : float const = 5e-3
+V_SUPPLY_MAX : float const = 40
+I_SUPPLY_MAX : float const = 1.4e-3
+I_OUT_MAX : float cons: = 21e-3
+INPUT()
+REF_5V()
+GND()
2
-opamp
<<Component>>
<<SW>>
AD623_Instrumentation_OPAMP
+V_SUPPLY_MIN : float const = 2.5
+V_SUPPLY_MAX : float const = 6
+I_SUPPLY_MAX : float const = 625e-6
+GBW : float const = 1e6
+COMPLIANCE_IN : float const = -
+COMPLIANCE_OUT : float const = -1
+ID : float const = -1
+LATCHUP_LEV : float const = -1

4.2.1. Class Bk1B221_Magnetometer_Sensor

This is the part of the circuitry that implements measuring of the actual magnetic field.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
GAIN (private) Gain of magnetic field components conditioning circuits.	float	31.303
OA_OFFSET (private) Offset of magnetic field components conditioning circuits, applied on opamp REF pins, in V.	float	1.5
sensor (private) Biaxial magnetic field sensor.	Bk1C24_Component_Selection$Transducers$Magnetometer 2-axis HMC1002	
reference (private) Voltage reference which supplies magnetic sensing bridge.	Bk1C24_Component_Selection$Voltage References$REF02_5V_Reference	
MAGN_OFFSET (public) Global offset of magnetometer sensor on both axes [V], in typical conditions. It depends on sensor typical offset and conditioning circuit gain and offset.	float	sensor.V_BIAS_RELATIVE*reference.REF_VOLTAGE*GAIN + OA_OFFSET
PDB_MAX (private) Maximum possible Power Distribution Bus voltage [V].	short	18
P_REF_MAX (private) Maximum dissipated power by each voltage divider on REF_3V, in W (see electric scheme in par. 6.1.6). Calculated value.	float	4.5e-3
opamp (private) Instrumentation Amplifier used for conditioning circuit of magnetometer.	Bk1C24_Component_Selection$OPAMPs$AD623 instrumentation OP-AMP[2]	
PowerConsumption_Avg_SR (private) Average power consumption of Set/Reset circuit, in W. Theoric value. Based on sensor's datasheet declared value of maximum S/R effective current from power supply (1 mA).	float	3.3e-3
PowerConsumption_Peak_SR (private) Set/Reset circuit peak power consumption, in W.	float	TBD
PowerConsumption_Avg_Max (private) Maximum average power consumption of Magnetometer	float	(reference.REF_VOLTAGE*r

Sensor block, in W. Calculated from maximum values declared in components' datasheet. Contributions to power dissipation are Magnetometer 2-axis HMC1002, REF02_5V_Reference, 2x AD623 instrumentation OP-AMP, 2x REF_3V voltage dividers and Set/Reset circuit.		eference.REF_VOLTAGE*2/sensor.R_BRIDGE_MIN) + (reference.I_SUPPLY_MAX*PDB_MAX) + 2*(opamp[1].I_SUPPLY_MAX*5) + (2*P_REF_MAX) + PowerConsumption_Avg_SR
PowerConsumption_Peak (public) Peak power consumption of Magnetometer Sensor block in W, including Set/Reset contribution -> PowerConsumption_Peak_SR.	float	PowerConsumption_Peak_SR + PowerConsumption_Avg_Max - PowerConsumption_Avg_SR
Max supply current (private) Maximum supply current of Magnetometer Sensor block (when enabled) in A. Tested with PDBINT voltage = 14 V.	float	0.0128
SENS_MAGNETIC (public) Sensitivity of conditioned magnetic sensor, in V/T.	float	(reference.REF_VOLTAGE * sensor.SENS * GAIN)
IO_ports (private) Attribute that represents the class IOports_1B8 of Bk1A_1B8_Codes containing I/O port numbers and relative bits' numbers.	Bk1A_1B8_Codes$IOports_1B8	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
MAGN_X (public) Output voltage for magnetic field along X-axis. Output voltage is 1.2V + SENS_MAGNETIC [V/T] * MagneticField(x) [T]		
MAGN_Y (public) Output voltage for magnetic field along Y-axis. Output voltage is 1.2V + SENS_MAGNETIC [V/T] * MagneticField(y) [T]		
notSET (public) Triggers a SET pulse on both magnetometer channels when transitioning from high to low. TTL input.	not_set : bool	
RESET (public) Triggers a RESET pulse on both magnetometer channels when transitioning from low to high. TTL input.	reset : bool	

EN_MAGN (public) Enables magnetometer, by providing supply to bridge and opamp. Active high. TTL input.	enable : bool	
3V3 (public) 3.3V supply voltage. Supplies Set/Reset circuit.		
5V (public) 5 V supply voltage. Supplies differential OpAmps.		
REF_3V (public) 3V voltage reference. It feeds a voltage divider whose output is connected to REF pin of each AD623.		
PDBINT (public) Power distribution bus voltage. It feeds REF02 voltage reference.		
AGND (public) Analog ground		
GND (public) Power and digital ground.		

4.2.2. Class Magnetometer 2-axis HMC1002

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
Manufacturer (public)	Bk1C24_Component_Selection$String	Honeywell
Part number (public)	Bk1C24_Component_Selection$String	HMC1002
SENS (public) Sensitivity of sensor, in V/V/T.	float	32.0
V_SUPPLY_MAX (public)	float	12
R_BRIDGE (public) Typical bridge resistance, in Ohm	float	850
R_BRIDGE_MIN (public) Minimum bridge resistance, in Ohm	float	600
R_BRIDGE_MAX (public) Maximum bridge resistance, in Ohm	float	1200
V_BIAS_RELATIVE (public) Average value of the bias differential voltage, relative to power supply voltage.	float	-0.001875
V_OFFSET_RELATIVE (public) Max absolute value of the offset differential voltage, relative to power supply voltage.	float	0.005625

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
OUT + (A) (public) OUT Differential magnetic signal x-axis		
OUT - (A) (public) OUT Differential magnetic signal x-axis		
OUT + (B) (public) OUT Differential magnetic signal y-axis		
OUT - (B) (public) OUT Differential magnetic signal y-axis		
S/R+B (public) IN to SR_B signal from Set/Reset circuit.		
S/R-B (public) OUT signal to GND		
S/R+A (public) IN to SR_A signal from Set/Reset circuit.		
S/R-A (public) OUT signal to GND		
VbridgeA (public) High stability IN supply voltage of 5V for magnetic sensor. Connected to REF_5V.		
VbridgeB (public) High stability IN supply voltage of 5V for magnetic sensor. Connected to REF_5V.		
AGND (public) IN/OUT analog GND signal		

4.2.3. Class REF02_5V_Reference

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
Manufacturer (public)	Bk1C24_Component_Selection$String	Analog Devices
Model (public)	Bk1C24_Component_Selection$String	REF02
REF_VOLTAGE (public)	float	5.0
TOLERANCE (public) Relative tolerance of reference voltage	float	5e-3
V_SUPPLY_MAX (public)	float	40

I_SUPPLY_MAX (public)	float	1.4e-3
I_OUT_MAX (public) Maximum output current	float	21e-3

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
INPUT (public) voltage input signal, connected to PDBPOS		
REF_5V (public) Reference voltage output, relative to pin GND. Supplies the measuring bridge.		
GND (public) Negative supply.		

4.2.4. Class AD623_instrumentation_OPAMP

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
V_SUPPLY_MIN (public)	float	2.5
V_SUPPLY_MAX (public)	float	6
I_SUPPLY_MAX (public)	float	625e-6
GBW (public)	float	1e6
COMPLIANCE_IN (public)	float	-1
COMPLIANCE_OUT (public)	float	-1
TID (public)	float	-1
LATCHUP_LET (public)	float	-1

4.3. Bk1B222_Magnetic_Torque_Actuator

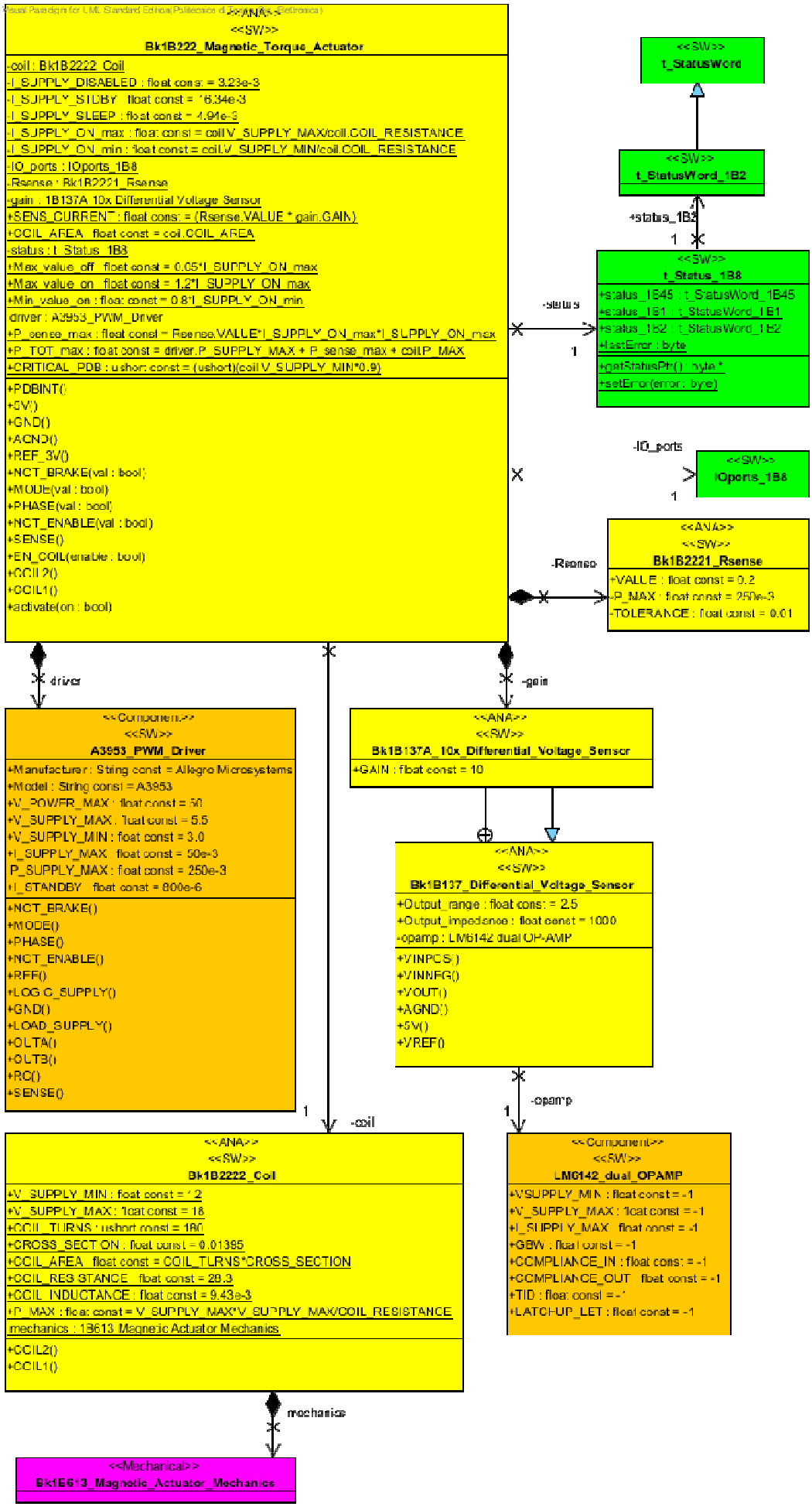
This class diagram represents the Bk1B222 Magnetic Torque Actuator block. In the magnetic attitude control system, it is the part of the circuitry that

- gives to the tile the angular momentum requested by the 1B2 Attitude and Orbit Subsystem, by driving a magnetic coil;
- measures and returns value of current flowing into the coil, which is used to calculate angular momentum or for housekeeping purpose.

It contains:

- Class **Bk1B222_Magnetic_Torque_Actuator**, which identifies the shell of magnetic actuator block and includes some specification attributes, associations with its subclasses and hardware operations identifying the I/O electric signals of this block. It also contains a software operation, *activate*, used to turn on/off the coil.
- Hardware component class **A3953_PWM_Driver**, which is the driver that provides supply to the coil and includes its specifications (by means of the attributes) and I/O signals (by means of the operations).
- Hardware class **Bk1B2221_Rsense**, including some specifications, identifies the resistor used to sense coil current.
- Hardware class **Bk1B2222_Coil**, including some specifications, identifies the magnetic coil; the operations identify its pins.
- Hardware class **Bk1B137_Differential_Voltage_Sensor**, including specifications and I/O signals, identifies the conditioning circuit used for coil current measuring and includes hardware component class **LM_6142_dual_OPAMP**.
- Hardware class **Bk1B137A_10x_Differential_Voltage_Sensor**, which specifies class Bk1B137_Differential_Voltage_Sensor.
- Mechanical components class **Bk1B613_Magnetic_Actuator_Mechanics**
- Software class **t_StatusWord_1B2**, which is a specification of sw class **t_StatusWord** (both described in par. 4.4).
- Software class **IOports_1B8** of Bk1A_1B8_Codes package, which contains constants identifying ports and bits numbers associated to 1B8 Power Management Tile digital signals as shown in figure (for now TBD).

All attributes and operations of the classes above (except IOports_1B8, Bk1B613_Magnetic_Actuator_Mechanics and t_StatusWord_1B2) are described subsequently in this paragraph.


<<ANA>>
<<SW>>
Bk1B222_Magnetic_Torque_Actuator
-coil : Bk1B2222_Coil
-I_SUPPLY_DISABLED : float cons: = 3.23e-3
-I_SUPPLY_STDBY : float const = 6.34e-3
-I_SUPPLY_SLEEP : float const = 4.91e-3
-I_SUPPLY_ON_max : float const = coil.V_SUPPLY_MAX/coil.COIL_RESISTANCE
-I_SUPPLY_ON_min : float const = coil.V_SUPPLY_MIN/coil.COIL_RESISTANCE
-IO_ports : IOports_1B8
-Rsense : Bk1B2221_Rsense
-gain : 1B137A 10x Differential Voltage Sensor
+SENS_CURRENT : float cons: = (Rsense.VALUE * gain.GAIN)
+COIL_AREA : float const = coil.COIL_AREA
-status : t_Status_1B8
+Max_value_off : float const = 0.05*I_SUPPLY_ON_max
+Max_value_on : float const = 1.2*I_SUPPLY_ON_max
+Min_value_on : float const = 0.8*I_SUPPLY_ON_min
driver : A3953_PWM_Driver
+P_sense_max : float const = Rsense.VALUE*I_SUPPLY_ON_max*I_SUPPLY_ON_max
+P_TOT_max : float const = driver.P_SUPPLY_MAX + P_sense_max + coilP_MAX
+CRITICAL_PDB : ushort const = (ushort)(coil.V_SUPPLY_MIN*0.9)
+PDBINT()
+3V()
+GND()
+AGND()
+REF_3V()
+NCT_BRAKE(val : bool)
+MODE(val : bool)
+PHASE(val : bool)
+NCT_ENABLE(val : bool)
+SENSE()
+EN_COIL(enable : bool)
+COIL2()
+COIL1()
+activate(on : bool)
<<SW>>
t_StatusWord
<<SW>>
t_StatusWord_1B2
+status_1B2
1
<<SW>>
t_Status_1B8
+status_1B45 : t_StatusWord_1B45
+status_1B1 : t_StatusWord_1B1
+status_1B2 : t_StatusWord_1B2
+lastError : byte
+getStatusPtr() : byte *
+setError(error : byte)
-status
1
-IO_ports
<<SW>>
IOports_1B8
1
<<ANA>>
<<SW>>
Bk1B2221_Rsense
+VALUE : float const = 0.2
+P_MAX : float const = 250e-3
+TOLERANCE : float const = 0.01
-Rsense
driver
-gain
<<Component>>
<<SW>>
A3953_PWM_Driver
+Manufacturer : String const = Allegro Microsystems
+Model : String const = A3953
+V_POWER_MAX : float const = 50
+V_SUPPLY_MAX : float const = 5.5
+V_SUPPLY_MIN : float const = 3.0
+I_SUPPLY_MAX : float const = 50e-3
P_SUPPLY_MAX : float const = 250e-3
+I_STANDBY : float const = 800e-6
+NCT_BRAKE()
+MODE()
+PHASE()
+NCT_ENABLE()
+REF()
+LOGIC_SUPPLY()
+GND()
+LOAD_SUPPLY()
+OLTA()
+OLTB()
+RC()
+SENSE()
<<ANA>>
<<SW>>
Bk1B137A_10x_Differential_Voltage_Sensor
+GAIN : float const = 10
<<ANA>>
<<SW>>
Bk1B137_Differential_Voltage_Sensor
+Output_range : float cons: = 2.5
+Output_impedance : float const = 1000
-opamp : LM6142 dual OP-AMP
+VINPOS()
+VINNEG()
+VOUT()
+AGND()
+5V()
+VREF()
1
-coil
1
-opamp
<<ANA>>
<<SW>>
Bk1B2222_Coil
+V_SUPPLY_MIN : float const = 2
+V_SUPPLY_MAX : float const = 18
+COIL_TURNS : ushort const = 180
+CROSS_SECTION : float const = 0.01395
+COIL_AREA : float const = COIL_TURNS*CROSS_SECTION
+COIL_RESISTANCE : float const = 28.3
+COIL_INDUCTANCE : float const = 9.43e-3
+P_MAX : float const = V_SUPPLY_MAX*V_SUPPLY_MAX/COIL_RESISTANCE
mechanics : 1B613 Magnetic Actuator Mechanics
+COIL2()
+COIL1()
<<Component>>
<<SW>>
LM6142_dual_OPAMP
+VSUPPLY_MIN : float const = -1
+V_SUPPLY_MAX : float const = -1
+I_SUPPLY_MAX : float const = -1
+GBW : float const = -1
+COMPLIANCE_IN : float cons: = -1
+COMPLIANCE_OUT : float const = -1
+TID : float const = -1
+LATCH-UP_LET : float const = -1
mechanics
<<Mechanical>>
Bk1B613_Magnetic_Actuator_Mechanics

4.3.1. Class Bk1B222_Magnetic_Torque_Actuator

This is the part of the circuitry that implements generation of a magnetic momentum to execute a command of ACTUATE_MAGNETIC.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
coil (private) Magnetic coil.	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B222_Magnetic_Torque_Actuator$Bk1B2222_Coil	
I_SUPPLY_DISABLED (private) Maximum supply current (A) when disabled. Tested. Maybe due to faulty switch on PDBINT voltage.	float	3.23e-3
I_SUPPLY_STDBY (private) Maximum supply current (A) when enabled (driver in **standby mode**) but not active, tested.	float	16.34e-3
I_SUPPLY_SLEEP (private) Maximum supply current (A) when enabled (driver in **sleep mode**) but not active, tested.	float	4.94e-3
I_SUPPLY_ON_max (private) Maximum supply current (A) when enabled and active, over all operating conditions.	float	coil.V_SUPPLY_MAX/coil.COIL_RESISTANCE
I_SUPPLY_ON_min (private) Minimum supply current (A) when enabled and active, over all operating conditions.	float	coil.V_SUPPLY_MIN/coil.COIL_RESISTANCE
IO_ports (private) Attribute that represents the class IOports_1B8 of Bk1A_1B8_Codes containing I/O port numbers and relative bits' numbers.	Bk1A_1B8_Codes$IOports_1B8	
Rsense (private) Resistor used to sense coil current.	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B222_Magnetic_Torque_Actuator$Bk1B2221_Rsense	
gain (private) Differential amplifier used for coil current conditioning.	Bk1B13_Sensors$Bk1B137_Differential_Voltage_Sensor$1B137A 10x Differential Voltage Sensor	

SENS_CURRENT (public) Sensitivity of SENSE output (conditioned coil current), in V/A	float	(Rsense.VALUE * gain.GAIN)
COIL_AREA (public) Equivalent area of magnetic coil Bk1B2222_Coil.	float	coil.COIL_AREA
status (private)	Bk1A_1B8_Codes$t_Status_1B8	
Max_value_off (public) Maximum allowed value of coil current when Magnetic Actuator block is disabled [A]. It corresponds to 5% of maximum theoric operating value.	float	0.05*I_SUPPLY_ON_max
Max_value_on (public) Maximum allowed value of coil current when Magnetic Actuator block is operating [A]. It corresponds to maximum theoric operating value + 20%.	float	1.2*I_SUPPLY_ON_max
Min_value_on (public) Minimum requested value of coil current when Magnetic Actuator block is operating [A]. It corresponds to minimum theoric operating value - 20%.	float	0.8*I_SUPPLY_ON_min
driver (Unspecified)	Bk1C24_Component_Selection$Drivers$A3953_PWM_Driver	
P_sense_max (public) Maximum power dissipated by coil current sensing resistor.	float	Rsense.VALUE*I_SUPPLY_ON_max*I_SUPPLY_ON_max
P_TOT_max (public) Maximum power dissipated by Bk1B222_Magnetic_Torque_Actuator block.	float	driver.P_SUPPLY_MAX + P_sense_max + coil.P_MAX
CRITICAL_PDB (public) Critical value of PDB voltage [V]. It is used for error Supervision use case.	ushort	(ushort)(coil.V_SUPPLY_MIN*0.9)

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
PDBINT (public) IN supply voltage for solenoid.		
5V (public) IN supply voltage for logic part of solenoid driver		
GND (public) IN/OUT ground voltage for digital and power signals		
AGND (public) IN/OUT analog groung voltage signal		
REF_3V (public) IN reference voltage signal necessary to coil driver to limit		

maximum load current		
NOT_BRAKE (public) IN signal to drive solenoid driver. Active low. If low, connects coil wires to ground (turns off both source drivers and turns on both sink drivers), discharging the energy previously cumulated into coil. Used to dynamically brake brush DC motors.	val : bool	
MODE (public) IN signal to drive solenoid driver. When ENABLE and BRAKE are high ('1'): - if MODE = '1' driver is in "Sleep Mode" (reduced power consumption) - if MODE = '0' driver is in Standby When ENABLE='0' and BRAKE='1': - if MODE = '1' coil is driven in fast current decay mode - if MODE = '0' in slow current decay mode (not used) When BRAKE='0': - if MODE = '1' coil is discharged in fast current decay mode - if MODE = '0' coil is discharged without current control	val : bool	
PHASE (public) IN signal to drive solenoid driver. It determines current-flow direction. If PHASE='1' coil is driven forward (current flows from COIL1 output to COIL2 output), if PHASE='0' coil is driven reverse (current flows from COIL2 to COIL1).	val : bool	
NOT_ENABLE (public) IN signal to drive solenoid driver. Active low. If low, enables coil driving by allowing current flow into it.	val : bool	
SENSE (public) OUT Current into solenoid signal from driver, conditioned.		
EN_COIL (public) When high, enables the whole Bk1B222_Magnetic_Torque_Actuator block by providing both power supplies. When low, both power supplies are removed internally from all analog and digital circuits, by means of PMOS devices.	enable : bool	
COIL2 (public) OUT signal to drive the solenoid		
COIL1 (public) OUT signal to drive the solenoid		
activate (public) Software operation which commands driver pins in the correct sequence. If true, coil is active and there is current (EN_COIL = 'true', NOT_ENABLE = 'false', NOT_BRAKE = 'true', MODE = 'true') and bit ACTIVE_COIL of t_StatusWord_1B2 class is set..	on : bool	

If false, coil is off (NOT_BRAKE = 'false', NOT_ENABLE = 'true', MODE = 'true', EN_COIL = 'false') and bit ACTIVE_COIL of t_StatusWord_1B2 class is reset.		

4.3.2. Class Bk1B2222_Coil

This is the Solenoid which generates a magnetic dipole when current flows through it.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
V_SUPPLY_MIN (public) Minimum supply voltage (V) for guaranteed operation over all operating conditions.	float	12
V_SUPPLY_MAX (public) Maximum supply voltage (V) for guaranteed operation over all operating conditions.	float	18
COIL_TURNS (public) Number of turns of coil	ushort	180
CROSS_SECTION (public) Average area of coil, in m2.	float	0.01395
COIL_AREA (public) Total equivalent area of coil, considering all turns, in m2.	float	COIL_TURNS *CROSS_SECTION
COIL_RESISTANCE (public)	float	28.3
COIL_INDUCTANCE (public) Inductance of whole coil at low frequency [H].	float	9.43e-3
P_MAX (public) Maximum coil dissipated power, in W.	float	V_SUPPLY_MAX*V_SUPPLY_MAX/COIL_RESISTANCE
mechanics (Unspecified)	Bk1B6_Mechanical_Subsystem$Bk1B61_PMT_Mechanical_Subsystem$1B613 Magnetic Actuator Mechanics	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
COIL2 (public) IN signal to drive the solenoid		

COIL1 (public) IN signal to drive the solenoid		

4.3.3. Class A3953_PWM_Driver

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
Manufacturer (public)	Bk1C24_Component_Selection$String	Allegro Microsystems
Model (public)	Bk1C24_Component_Selection$String	A3953
V_POWER_MAX (public) Maximum supply voltage for power lines, in V.	float	50
V_SUPPLY_MAX (public) Maximum supply voltage for logics, in V.	float	5.5
V_SUPPLY_MIN (public) Minimum supply voltage for logics, in V.	float	3.0
I_SUPPLY_MAX (public) Maximum supply current for logics, in A.	float	50e-3
P_SUPPLY_MAX (Unspecified) Max power consumption of logics, at 5V supply	float	250e-3
I_STANDBY (public) Supply current for logics, in sleep mode, in A.	float	800e-6

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
NOT_BRAKE (public) IN signal to drive solenoid driver. Active low. If low, connects coil wires to ground (turns off both source drivers and turns on both sink drivers), discharging the energy previously cumulated into coil. Used to dynamically brake brush DC motors.		
MODE (public) IN signal to drive solenoid driver. When ENABLE and BRAKE are high ('1'): - if MODE = '1' driver is in "Sleep Mode" (reduced power consumption) - if MODE = '0' driver is in Standby		

When ENABLE='0' and BRAKE='1': - if MODE = '1' coil is driven in fast current decay mode - if MODE = '0' in slow current decay mode (not used) When BRAKE='0': - if MODE = '1' coil is discharged in fast current decay mode - if MODE = '0' coil is discharged without current control		
PHASE (public) IN signal to drive solenoid driver. It determines current-flow direction. If PHASE='1' coil is driven forward (current flows from A output to B output), if PHASE='0' coil is driven reverse (current flows from B to A).		
NOT_ENABLE (public) IN signal to drive solenoid driver. Active low. If low, enables coil driving by allowing current flow into it.		
REF (public) IN reference voltage signal		
LOGIC_SUPPLY (public) IN supply voltage for logic part of the driver. Typically 5 V.		
GND (public) IN/OUT GND voltage signal		
LOAD_SUPPLY (public) IN supply voltage for load.		
OUTA (public) OUT signal to drive the solenoid		
OUTB (public) OUT signal to drive the solenoid		
RC (public) IN timing signal determining t_off of the driver.		
SENSE (public) OUT current level into solenoid signal from driver.		

4.3.4. Class Bk1B2221_Rsense

Kelvin-contact SMD 0805 Resistor. Used for measuring SENSE current.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
VALUE (public) Resistance of coil current sensing resistor, in ohms.	float	0.2
P_MAX (private) Sensing resistor maximum dissipable power, in W.	float	250e-3
TOLERANCE (private) Relative tolerance of sensing resistor value.	float	0.01

4.3.5. Class Bk1B137A_10x_Differential_Voltage_Sensor

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
GAIN (public)	float	10

4.3.6. Class Bk1B137_Differential_Voltage_Sensor

Differential voltage sensor.
Output voltage is 1.25V + (VINPOS - VINNEG) * Gain.
Output range is 0-2.5V

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
Output_range (public) Range of output voltage to the ADC. Must be positive voltage. Input and output voltages are referenced to analog ground **AGND**.	float	2.5
Output_impedance (public)	float	1000
opamp (private)	Bk1C24_Component_Selection$OPAMPs$LM6142 dual OP-AMP	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
VINPOS (public) Input voltage, positive side		
VINNEG (public) Input voltage, negative side.		
VOUT (public) Output voltage. VOUT = 10 (VINPOS-VINNEG) +VREF		
AGND (public) Analog ground.		
5V (public) Positive supply voltage; referenced to AGND		
VREF (public)		

Reference voltage for the output.		

4.3.7. Class LM6142_dual_OPAMP

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
VSUPPLY_MIN (public)	float	-1
V_SUPPLY_MAX (public)	float	-1
I_SUPPLY_MAX (public)	float	-1
GBW (public)	float	-1
COMPLIANCE_IN (public)	float	-1
COMPLIANCE_OUT (public)	float	-1
TID (public)	float	-1
LATCHUP_LET (public)	float	-1

4.4. Bk1B229_Controller

This class diagram represents the Bk1B229 Controller. This is a software block. In the magnetic attitude control system, it identifies the unit which manages tasks and realizes project specifications.
It contains the main sw class **Bk1B229_Controller**, which includes:

- operations (functions) that implement use case

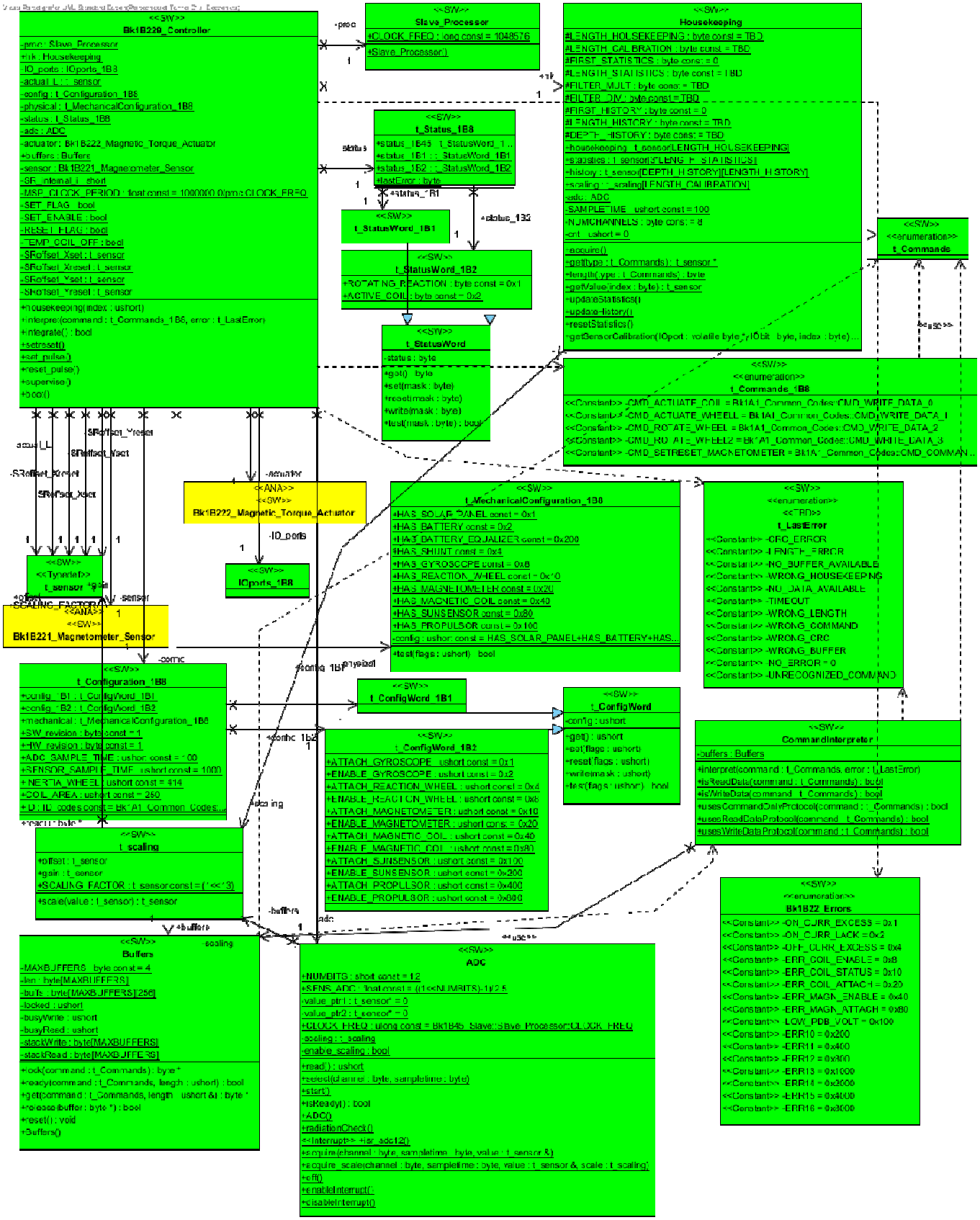
- attributes (variables and constants) which are used by these functions

- attributes which refer to other classes used by the Controller, like:
  - **ADC**: hw and sw ADC of 1B45 Common package, used to acquire, scale and get housekeeping values
  - **Buffers** of 1B45_Slave package, which represents the set of slave buffers, used to get magnetic coil actuate command, containing requested angular momentum
  - **Housekeeping** of 1B45_Slave package, which contains housekeeping and scaling data vectors and other information
  - **Slave Processor** of 1B45_Slave package, containing Power Management Tile microcontroller specifications
  - **t_Status_1B8**, **t_Configuration_1B8**, **t_MechanicalConfiguration_1B8**, which contain attributes that represent, respectively, status, configuration and mechanical configuration words of the whole Power Management Tile
  - **IOports_1B8**, **Bk1B221_Magnetometer_Sensor** and **Bk1B222_Magnetic_Torque_Actuator**

It also contains:

- Every class which the classes above use or depend on
- Classes that represent common types in the system, used by Controller too
- Class **Bk1B22_Errors**, which contains magnetic attitude control subsystem error word

All attributes and operations of the classes above (if used by Controller or its subclasses) are described subsequently in detail in this paragraph.


<<SW>>
Bk1B220_Controller
-pmc : Slave_Processor
+hk : Housekeeping
-IO_ports : IOports_1B3
-actual_L : : _sensor
-config : t_Configuration_1B8
-physical : t_MechanicalConfiguration_1B8
-status : t_Status_1B8
-ads : ADC
-actuator : Bk1B222_Magnetic_Torque_Actuator
+buffers : Buffers
-sensor : Bk1B221_Magnetometer_Sensor
-SR_internal_i : short
-MSP_CLOCK_PERIOD : float const = 1000000.0/pmc.CLOCK_FREQ
-SET_FLAG : bool
-SET_ENABLE : bool
-RESET_FLAG : bool
-TEMP_COIL_OFF : bool
-SRoffset_Xset : t_sensor
-SRoffset_Xreset : t_sensor
-SRoffset_Yset : t_sensor
-SRoffset_Yreset : t_sensor
+housekeeping(index : ushort)
+interpret(command : t_Commands_1B8, error : t_LastError)
+integrate() : bool
+setreset()
+set_pulse()
+reset_pulse()
+supervise()
+cool()
<<SW>>
Slave_Processor
+CLOCK_FREQ : long const = 1048576
+Slave_Processor()
<<SW>>
Housekeeping
#LENGTH_HOUSEKEEPING : byte const = TBD
#LENGTH_CALIBRATION : byte const = TBD
#FIRST_STATISTICS : byte const = 0
#LENGTH_STATISTICS : byte const = TBD
#FILTER_MULT : byte const = TBD
#FILTER_DIV : byte const = TBD
#FIRST_HISTORY : byte const = 0
#LENGTH_HISTORY : byte const = TBD
#DEPTH_HISTORY : byte const = TBD
+housekeeping : t_sensor[LENGTH_HOUSEKEEPING]
+statistics : t_sensor[LENGTH_STATISTICS]
+history : t_sensor[DEPTH_HISTORY][LENGTH_HISTORY]
+scaling : _scaling[LENGTH_CALIBRATION]
-adc : ADC
-SAMPLETIME : ushort const = 100
-NUMCHANNELS : byte const = 8
-cnt : ushort = 0
+acquire()
+getType(i : t_Commands) : t_sensor *
+length(type : t_Commands) : byte
+getValue(index : byte) : t_sensor
+updateStatistics()
+updateHistory()
+resetStatistics()
+getSensorCalibration(IOport : volatile byte*, IObit : byte, index : byte)...
<<SW>>
t_Status_1B8
+status_1B45 : t_StatusWord_1...
+status_1B1 : : StatusWord_1B1
+status_1B2 : : StatusWord_1B2
+lastError : byte
<<SW>>
t_StatusWord_1B1
<<SW>>
t_StatusWord_1B2
+ROTATING_REACTION : byte const = 0x1
+ACTIVE_COIL : byte const = 0x2
<<SW>>
t_StatusWord
-status : byte
+get() : byte
+set(mask : byte)
+reset(mask : byte)
+write(mask : byte)
+test(mask : byte) : bool
<<SW>>
<<enumeration>>
t_Commands
<<SW>>
<<enumeration>>
t_Commands_1B8
<<Constant>> -CMD_ACTUATE_COIL = Bk1A1_Common_Codes::CMD_WRITE_DATA_0
<<Constant>> -CMD_ACTUATE_WHEELL = Bk1A1_Common_Codes::CMD_WRITE_DATA_1
<<Constant>> -CMD_ROTATE_WHEEL = Bk1A1_Common_Codes::CMD_WRITE_DATA_2
<<Constant>> -CMD_ROTATE_WHEEL2 = Bk1A1_Common_Codes::CMD_WRITE_DATA_3
<<Constant>> -CMD_SETRESET_MAGNETOMETER = Bk1A1_Common_Codes::CMD_COMMAN...
<<ANA>>
<<SW>>
Bk1B222_Magnetic_Torque_Actuator
<<SW>>
IOports_1B8
<<SW>>
t_MechanicalConfiguration_1B8
+HAS_SOLAR_PANEL const = 0x1
+HAS_BATTERY const = 0x2
+HAS_BATTERY_EQUALIZER const = 0x200
+HAS_SHUNT const = 0x4
+HAS_GYROSCOPE const = 0x8
+HAS_REACTION_WHEEL const = 0x10
+HAS_MAGNETOMETER const = 0x20
+HAS_MAGNETIC_COIL const = 0x40
+HAS_SUNSENSOR const = 0x80
+HAS_PROPULSOR const = 0x100
-config : ushort const = HAS_SOLAR_PANEL+HAS_BATTERY+HAS...
+test(flags : ushort) : bool
<<SW>>
<<enumeration>>
<<TRD>>
t_LastError
<<Constant>> -CRC_ERROR
<<Constant>> -LENGTH_ERROR
<<Constant>> -NO_BUFFER_AVAILABLE
<<Constant>> -WRONG_HOUSEKEEPING
<<Constant>> -NO_DATA_AVAILABLE
<<Constant>> -TIMEOUT
<<Constant>> -WRONG_LENGTH
<<Constant>> -WRONG_COMMAND
<<Constant>> -WRONG_CRC
<<Constant>> -WRONG_BUFFER
<<Constant>> -NO_ERROR = 0
<<Constant>> -UNRECOGNIZED_COMMAND
<<SW>>
Tynwrite>>
t_sensor
<<ANA>>
<<SW>>
Bk1B221_Magnetometer_Sensor
<<SW>>
t_Configuration_1B8
+config_1B1 : t_ConfigWord_1B1
+config_1B2 : t_ConfigWord_1B2
+mechanical : t_MechanicalConfiguration_1B8
+SW_revision : byte const = 1
+HW_revision : byte const = 1
+ADC_SAMPLE_TIME : ushort const = 100
+SENSOR_SAMPLE_TIME : ushort const = 1000
+INERTIA_WHEEL : ushort const = 414
+COIL_AREA : ushort const = 250
+ID_codes const = Bk1A1_Common_Codes:...
<<SW>>
t_ConfigWord_1B1
<<SW>>
t_ConfigWord
-config : ushort
+get() : ushort
+set(flags : ushort)
+reset flags : ushort)
+write(mask : ushort)
+test(flags : ushort) : bool
<<SW>>
t_ConfigWord_1B2
+ATTACH_GYROSCOPE : ushort const = 0x1
+ENABLE_GYROSCOPE : ushort const = 0x2
+ATTACH_REACTION_WHEEL : ushort const = 0x4
+ENABLE_REACTION_WHEEL : ushort const = 0x8
+ATTACH_MAGNETOMETER : ushort const = 0x10
+ENABLE_MAGNETOMETER : ushort const = 0x20
+ATTACH_MAGNETIC_COIL : ushort const = 0x40
+ENABLE_MAGNETIC_COIL : ushort const = 0x80
+ATTACH_SUNSENSOR : ushort const = 0x100
+ENABLE_SUNSENSOR : ushort const = 0x200
+ATTACH_PROPULSOR : ushort const = 0x400
+ENABLE_PROPULSOR : ushort const = 0x800
<<SW>>
t_scaling
+offset : t_sensor
+gain : t_sensor
+SCALING_FACTOR : t_sensor const = (1<<13)
+scale(value : t_sensor) : t_sensor
<<SW>>
CommandInterpreter
-buffers : Buffers
+interpret(command : t_Commands, error : t_LastError)
+isWriteData(command : t_Commands) : bool
+isReadData(command : t_Commands) : bool
+usesCommandOnlyProtocol(command : t_Commands) : bool
+usesReadDataProtocol(command : t_Commands) : bool
+usesWriteDataProtocol(command : t_Commands) : bool
<<SW>>
Buffers
-MAXBUFFERS : byte const = 4
-len : byte[MAXBUFFERS]
-buffs : byte[MAXBUFFERS][256]
-locked : ushort
-busyWrite : ushort
-busyRead : ushort
-stackWrite : byte[MAXBUFFERS]
-stackRead : byte[MAXBUFFERS]
+lock(command : t_Commands) : byte *
+ready(command : t_Commands, length : ushort) : bool
+get(command : t_Commands, length : ushort &) : byte *
+release(buffer : byte *) : bool
+reset() : void
+Buffers()
<<SW>>
ADC
+NUMBITS : short const = 12
+SENS_ADC : float const = ((1<<NUMBITS)-1)/2.5
-value_ptr1 : t_sensor* = 0
-value_ptr2 : t_sensor* = 0
+CLOCK_FREQ : ulong const = Bk1B45_Slave::Slave_Processor::CLOCK_FREQ
-scaling : : _scaling
-enable_scaling : bool
+read() : ushort
+reload(channel : byte, sampletime : byte)
+start()
+isReady() : bool
+ADC()
+radiationCheck()
<<Interrupt>> +isr_adc12()
+acquire(channel : byte, sampletime : byte, value : t_sensor &)
+acquire_scale(channel : byte, sampletime : byte, value : t_sensor &, scale : t_scaling)
+off()
+enableInterrupt()
+disableInterrupt()
<<SW>>
<<enumeration>>
Bk1B22_Errors
<<Constant>> -ON_CURR_EXCESS = 0x1
<<Constant>> -ON_CURR_LACK = 0x2
<<Constant>> -OFF_CURR_EXCESS = 0x4
<<Constant>> -ERR_COIL_ENABLE = 0x8
<<Constant>> -ERR_COIL_STATUS = 0x10
<<Constant>> -ERR_COIL_ATTACH = 0x20
<<Constant>> -ERR_MAGN_ENABLE = 0x40
<<Constant>> -ERR_MAGN_ATTACH = 0x80
<<Constant>> -LOW_PDB_VOLT = 0x100
<<Constant>> -ERR10 = 0x200
<<Constant>> -ERR11 = 0x400
<<Constant>> -ERR12 = 0x800
<<Constant>> -ERR13 = 0x1000
<<Constant>> -ERR14 = 0x2000
<<Constant>> -ERR15 = 0x4000
<<Constant>> -ERR16 = 0x8000
+proc
status
+hk
+status_1B1
+status_1B2
+pmc
-actuator
-IO_ports
-sensor
-pmc
-config_1B1
-physical
-config_1B2
-buffers
-scaling
-adc
+use

4.4.1. Class Bk1B229_Controller

This software class is part of the Magnetic Attitude Subsystem and represents all variables, flags, references to other objects, functions that implement every required attitude controlling operation.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
proc (private)	Bk1B45_Slave.Slave_Processor	
hk (public) Housekeeping class of Bk1B45_Slave package.	Bk1B45_Slave.Housekeeping	
IO_ports (private)	Bk1A_1B8_Codes$IOports_1B8	
actual_L (private) Stored value of requested **angular momentum**. If > 0, whenever integrate operation is called (every 1s), it is decreased by calculated step of magnetic torque integral in time: |**Bxy**|***D*dt**, where Bxy and D are respectively actual value of magnetic field projection on x,y plane and actual value of coil dipole and dt = SENSOR_SAMPLE_TIME. When **actual_L** becomes <= 0, coil driving is stopped.	Bk1A1_Common_Codes$t_sensor	
config (private) t_Configuration_1B8 class of Bk1A_1B8_Codes package.	Bk1A_1B8_Codes$t_Configuration_1B8	
physical (private) t_MechanicalConfiguration_1B8 class of Bk1A_1B8_Codes package.	Bk1A_1B8_Codes$t_MechanicalConfiguration_1B8	
status (private) t_Status_1B8 class of Bk1A_1B8_Codes package.	Bk1A_1B8_Codes$t_Status_1B8	
adc (private) ADC class of 1B45 Common package.	Common.ADC	
actuator (private)	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B222_Magnetic_Torque_Actuator	
buffers (public) Buffers class of Bk1B45_Slave package.	Bk1B45_Slave.Buffers	
sensor (private)	Bk1B22_Magnetic_Attitude_Subsys	

	tem$Bk1B2 21_Magneto meter_Sens or	
SR_internal_i (private) Index used for automatic system Set/Reset timing.	short	
MSP_CLOCK_PERIOD (private) Clock period of the Slave Processor, in microseconds.	float	1000000.0/proc .CLOCK_FRE Q
SET_FLAG (private) Flag used for Set/Reset procedure.	bool	
SET_ENABLE (private) Flag used for Set/Reset procedure.	bool	
RESET_FLAG (private) Flag used for Set/Reset procedure.	bool	
TEMP_COIL_OFF (private) Flag used to verify whether coil has been temporarily disabled, due to magnetic field measuring, or not.	bool	
SRoffset_Xset (private) Stored value of x-axis magnetic field component after a set pulse, acquired during Set/Reset process. This is used to calculate offset term of x component to be applied during magnetic field acquiring operations to calibrate measures.	Bk1A1_Co mmon_Cod es$t_sensor	
SRoffset_Xreset (private) Stored value of x-axis magnetic field component after a reset pulse, acquired during Set/Reset process. This is used to calculate offset term of x component to be applied during magnetic field acquiring operations to calibrate measures.	Bk1A1_Co mmon_Cod es$t_sensor	
SRoffset_Yset (private) Stored value of y-axis magnetic field component after a set pulse, acquired during Set/Reset process. This is used to calculate offset term of y component to be applied during magnetic field acquiring operations to calibrate measures.	Bk1A1_Co mmon_Cod es$t_sensor	
SRoffset_Yreset (private) Stored value of y-axis magnetic field component after a reset pulse, acquired during Set/Reset process. This is used to calculate offset term of y component to be applied during magnetic field acquiring operations to calibrate measures.	Bk1A1_Co mmon_Cod es$t_sensor	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
housekeeping (public) Acquires, converts and calibrates housekeeping channel number **index**. It has to be automatically called with a frequency of TIMER_FREQ. It also periodically calls the integrate and supervise operations and manages the Set/Reset sequence for magnetometer sensor.	index : ushort	
interpret (public) Decodes commands received from the system (parameter **command**) identifying command type and executing correct	command : Bk1A_1B8_Co des$t_Comman	

operations for each type. It can interpret CMD_SET_CONFIGURATION, CMD_RESET_CONFIGURATION, CMD_ACTUATE_COIL, CMD_SETRESET_MAGNETOMETER commands.	ds_1B8 error : t_LastError	
integrate (public) If the system has to generate an angular momentum, it is periodically called every SENSOR_SAMPLE_TIME. Calculates the integral of measured B (module of magnetic field projection on x,y plane) multiplied by actual D (coil dipole) in time, decreasing the initial value (requested angular momentum). Where: N = number of coil windings; S = coil area; I = housekeeping[COIL_CURRENT]; D = N*S*I; Bx = housekeeping[MAGNETIC_FIELD_X]; By = housekeeping[MAGNETIC_FIELD_Y]; B = sqrt(Bx^2+By^2); --module of magnetic field projection on x,y plane actual_L = actual_L - B*D*SENSOR_SAMPLE_TIME; -- integral		bool
setreset (public) Sets SET_ENABLE to true to start either the SetReset Magnetometer or System SR Magnetometer use cases. The flag is reset automatically at the end of the set-reset operation.		
set_pulse (public) Gives a SET pulse to magnetic field sensor, respecting minimum times according to sensor's data-sheet.		
reset_pulse (public) Gives a RESET pulse to magnetic field sensor. Minimum times (according to sensor's data-sheet) are guaranteed by housekeeping operation timing.		
supervise (public) Realizes Supervision use case, finding errors eventually present in the system.		
boot (public) Realizes Boot use case, properly initializing elements in the system at bootstrap.		

4.4.2. Class Slave_Processor

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
CLOCK_FREQ (public)	long	1048576

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
Slave_Processor (public)		

4.4.3. Class : Housekeeping

Stereotypes :	<<SW>>

Attribute : adc

Type :	Common.ADC
Scope :	classifier

Attribute : housekeeping

Documentation :	The vector into which the **acquire()** operation shall periodically store the values of the last-acquired housekeeping values. The organization of the housekeeping vector is application-dependent, therefore it shall be defined and documented by the **Designer**. Also the body of the **acquire()** operation shall be defined by the **Designer**.
Visibility :	public
Type :	Bk1A1_Common_Codes$t_sensor[LENGTH_HOUSEKEEPING] [LENGTH_HOUSEKEEPING]
Scope :	classifier

Attribute : LENGTH_CALIBRATION

Documentation :	Number of components of the **housekeeping** vector which can be calibrated at run-time.
Visibility :	protected
Type :	byte
Initial Value :	TBD
Scope :	classifier

Attribute : LENGTH_HOUSEKEEPING

Documentation :	Length (in number of **t_sensor** values) of the **housekeeping** vector (see use case

	Housekeeping). Corresponds to the number of sensor or values acquired by the **acquire()** operation.
Visibility :	protected
Type :	byte
Initial Value :	TBD
Scope :	classifier

Attribute : scaling

Documentation :	The scaling to be added to the corresponding element in the **housekeeping** vector for run-time calibration. The organization of the **housekeeping** vector is application-dependent, therefore also the content of the **offset** vector, which shall be defined and documented by the **Designer** accordingly. Only the first **LENGTH_CALIBRATION** components can be calibrated. The offset shall be subtracted from the measurement from the ADC before being multiplied by **gain**.
Visibility :	public
Type :	Bk1A1_Common_Codes$t_scaling[LENGTH_CALIBRATION] [LENGTH_CALIBRATION]
Scope :	classifier

Operation : get

Documentation :	Returns a pointer to housekeeping data vector (10 bytes), Depending on the argument **type**, it returns pointer to: • last acquired housekeeping, if **type** == **CMD_GET_HOUSEKEEPING** • housekeeping history, if **type** == **CMD_GET_HISTORY**. History is organized as a t_sensor history[DEPTH_HISTORY][LENGTH_HISTORY] where the last sample is stored in history[0] • housekeeping statistics, if **type** == **CMD_GET_STATISTICS**. Statistics are organized as follows: all min values, all max values, all filtered values. If an unsupported command is given, it returns NULL
Parameter(s) :	type : Bk1A1_Common_Codes$t_Commands
Return type :	Bk1A1_Common_Codes$t_sensor
Type modifier :	*
Visibility :	public
Scope :	classifier

Operation : getSensorCalibration

Documentation :	Reads calibration data taken from one-wire memory connected to bit IObit of port IOport and returns calibration formatted as t_scaling (offset and gain).
Parameter(s) :	IOport : volatile byte* IObit : byte index : byte
Return type :	Bk1A1_Common_Codes$t_scaling
Visibility :	public

Scope :	instance

Operation : getValue

Documentation :	Returns the last-acquired housekeeping value identified by **index** (between 0 and LENGTH_HOUSEKEEPING-1). If index is outside the bounds, it returns 0;
Parameter(s) :	index : byte
Return type :	Bk1A1_Common_Codes$t_sensor
Visibility :	public
Scope :	classifier

4.4.4. Class t_sensor

Type to be used as much as possible to store and transfer telemetry data and actuator values.
It corresponds to a 16-bits signed integer.
Its value can be properly scaled by means of the t_scaling class.

4.4.5. Class t_Configuration_1B8

Configuration information for the 1B8 Power Management Tile.
It contains one configuration word for each of its major subsystems:

- config_1B1 for 1B1 Power Management Subsystem
- config_1B2 for 1B2 Attitude and Orbit Control System

It also contains other constant configuration data:

- SW_revision containing the SW revision of the Tile
- HW_revision containing the HW revision of the Tile

All above information can be retrieved at run-time by means of the 1B45 package.

The t_configuration_1B8 also contains other constant configuration data with additional geometrical, mechanical and electrical parameters on the tile, which can only be accessed at compile-time. Details on these are found in the attribute section.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
config_1B1 (public)	Bk1A_1B8_Codes$t_ConfigWord_1B1	
config_1B2 (public)	Bk1A_1B8_Codes$t_ConfigWord_1B2	
mechanical (public)	Bk1A_1B8_Codes$t_MechanicalConfiguration_1B8	
SW_revision (public) Attribute containing the SW revision of the 1B8 Tile.	byte	1
HW_revision (public)	byte	1

Attribute containing the HW revision of the 1B8 Tile.		
ADC_SAMPLE_TIME (public) Duration of sample phase of ADC in microseconds. Actual value might differ depending on ADC capabilities.	ushort	100
SENSOR_SAMPLE_TIME (public) Sample time of each housekeeping sensor, in milliseconds. Timer frequency must be higher, as only one sensor is acquired at every clock tick, so acquiring all sensors requires several clock ticks.	ushort	1000
INERTIA_WHEEL (public) Combined Moment of Inertia of the motor and the wheel in gcm2. motor inertia = 13.9 gcm2 wheel inertia = 400 gcm2	ushort	414
COIL_AREA (public) Number of turns times coil area in cm2.	ushort	250
ID (public)	Bk1A1_Common_Codes$ID_codes	Bk1A1_Common_Codes::Bk1B8_PMT_V0

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
read (public)		byte*

4.4.6. Class : t_ConfigWord_1B2

Stereotypes :	<<SW>>
Documentation :	Specific configuration word for ARAMIS 1B2 subsystem (Attitude and Orbit Subsystem). It inherits all attributes and operations from its parent t_ConfigWord.

Attribute : ATTACH_MAGNETIC_COIL

Documentation :	Set if the Tile has a magnetic coil actuator. This bit is set by the manufacturer if the coil has been mounted on the Tile. The Master can reset this bit (by means of the **Reset Module Configuration** use case of **1B45** package)) in case the coil is detected to fail. The Master may also try to set the bit to try to correct the failure, but the bit will actually set only if the coil is present on the Tile. If the Master attempts to set the bit but, if the coil is not present, this bit will remain reset.
Visibility :	public
Type :	ushort
Initial Value :	0x40
Scope :	classifier

Attribute : ATTACH_MAGNETOMETER

Documentation :	Set if the Tile has a magnetic sensor. This bit is set by the manufacturer if the sensor has been mounted on the Tile. The Master can reset this bit (by means of the **Reset Module Configuration** use case of **1B45** package)) in case the sensor is detected to fail. The Master may also try to set the bit to try to correct the failure, but the bit will actually set only if the sensor is present on the Tile. If the Master attempts to set the bit but the sensor is not present, this bit will remain reset.
Visibility :	public
Type :	ushort
Initial Value :	0x10
Scope :	classifier

Attribute : ENABLE_MAGNETIC_COIL

Documentation :	Enables (when set) the magnetic coil actuator on the 1B8 Power Management Tile. See use cases of **1B22** subsystem.
Visibility :	public
Type :	ushort
Initial Value :	0x80
Scope :	classifier

Attribute : ENABLE_MAGNETOMETER

Documentation :	Enables (when set) the magnetometer on the 1B8 Power Management Tile. See use cases of **1B22** subsystem.
Visibility :	public
Type :	ushort
Initial Value :	0x20
Scope :	classifier

4.4.7. Class t_ConfigWord

Generic configuration word for ARAMIS tiles. It is composed of a config word, plus all operations to:

- get the content of configuration word;
- set the content of individual bits of configuration word;
- reset the content of individual bits of configuration word;
- test the content of individual bits of configuration word;
- write the content of the whole configuration word;

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
config (private)	ushort	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
get (public) Returns the content of configuration word.		ushort
set (public) Sets all configuration bits corresponding to a 1 in argument **flags**.	flags : ushort	
reset (public) Resets all configuration bits corresponding to a 1 in argument **flags**.	flags : ushort	
write (public) Overwrites the whole configuration word with the argument **mask**.	mask : ushort	
test (public) Returns **true** if ALL bits corresponding to a 1 in argument **flags** are set; **false** otherwise.	flags : ushort	bool

4.4.8. Class : t_MechanicalConfiguration_1B8

Stereotypes :	<<SW>>

Attribute : config

Documentation :	Defines the actual eectro-mechanical configuration of the 1B8 Power Management Tile. This is a constant value, therefore it cannot be modified at run-time. It shall match, for each tile, the exact list of optional elements of the tile. The user will access in read mode the list of available items.
Type :	ushort
Initial Value :	HAS_SOLAR_PANEL+HAS_BATTERY+HAS_SHUNT+HAS_GYROSCOPE+HAS_REACTION_WHEEL+HAS_MAGNETOMETER+HAS_MAGNETIC_COIL+HAS_SUNSENSOR
Scope :	classifier

Attribute : HAS_MAGNETIC_COIL

Documentation :	Set if the Tile has a magnetic coil actuator. This bit is set by the manufacturer if the coil has been mounted on the Tile. The Master can reset this bit (by means of the **Reset Module Configuration** use case of **1B45** package)) in case the coil is detected to fail. The Master may also try to set the bit to try to correct the failure, but the bit will actually set only if the coil is present on the Tile. If the Master attempts to set the bit but, if the coil is not present, this bit will remain reset.
Visibility :	public
Initial Value :	0x40
Scope :	classifier

Attribute : HAS_MAGNETOMETER

Documenta	Set if the Tile has a magnetic sensor. This bit is set by the manufacturer if the sensor

tion :	has been mounted on the Tile. The Master can reset this bit (by means of the **Reset Module Configuration** use case of **1B45** package)) in case the sensor is detected to fail. The Master may also try to set the bit to try to correct the failure, but the bit will actually set only if the sensor is present on the Tile. If the Master attempts to set the bit but the sensor is not present, this bit will remain reset.
Visibility :	public
Initial Value :	0x20
Scope :	classifier

Operation : test

Documentation :	Returns **true** if ALL bits corresponding to a 1 in argument **flags** are set; **false** otherwise.
Parameter(s) :	flags : ushort
Return type :	bool
Visibility :	public
Scope :	instance

4.4.9. Class t_Status_1B8

Status information for the 1B8 Power Management Tile.
It contains one status word for each of its major subsystems:

- status_1B1 for 1B1 Power Management Subsystem
- status_1B2 for 1B2 Attitude and Orbit Control System
- status_1B45 for 1B45 Inter-tile communication package

It also contains the lastError attribute for additional details on the last reported error.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
status_1B45 (public)	Bk1A1_Common_Codes$t_StatusWord_1B45	
status_1B1 (public) Generic status word for the 1B1 Power Management Subsystem of the 1B8 Power Management Tile	Bk1A_1B8_Codes$t_StatusWord_1B1	
status_1B2 (public) Generic status word for the 1B2 Attitude Control Subsystem of the 1B8 Power Management Tile	Bk1A_1B8_Codes$t_StatusWord_1B2	
lastError (public) Attribute containing last error.	byte	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
getStatusPtr (public)		byte*
setError (public)	error : byte	

4.4.10. Class : t_StatusWord_1B2

Stereotypes :	<<SW>>
Documentation :	Specific status word for ARAMIS 1B2 subsystem (Attitude and Orbit Control Subsystem). It inherits all attributes and operations from its parent t_StatusWord.

Attribute : ACTIVE_COIL

Documentation :	Value for status bit indicating (when set) that the magnetic coil is rotating.
Visibility :	public
Type :	byte
Initial Value :	0x2
Scope :	classifier

4.4.11. Class t_StatusWord

Generic status word for ARAMIS tiles. It is composed of a status attribute, plus all operations to:

- get the content of status word;
- set the content of individual bits of status word;
- reset the content of individual bits of status word;
- test the content of individual bits of status word;
- write the content of the whole status word;

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
status (private) Generic status word for 1B45-compatible peripheral. Each bit indicates a different status condition, as enumerated by the other constant attributes of this class.	byte	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
get (public) Returns status word.		byte
set (public) Sets all bits of status word corresponding to bits set to 1 in argument **mask**.	mask : byte	
reset (public) Resets all bits of status word corresponding to bits set to 1 in argument **mask**.	mask : byte	
write (public) Overwrites the whole status word with the argument **mask**.	mask : byte	
test (public) Returns true if all bits of status word corresponding to bits set to 1 in argument **mask** are set.	mask : byte	bool

4.4.12. Class : ADC

Stereotypes :	<<SW>>

Attribute : CLOCK_FREQ

Visibility :	public
Type :	ulong
Initial Value :	Bk1B45_Slave::Slave_Processor::CLOCK_FREQ
Scope :	classifier

Attribute : NUMBITS

Documentation :	Number of bits of ADC output. Value is right-aligned.
Visibility :	public
Type :	short
Initial Value :	12
Scope :	classifier

Attribute : SENS_ADC

Documentation :	Sensitivity of ADC in units(LSB) / V
Visibility :	public
Type :	float
Initial Value :	((1<<NUMBITS)-1)/2.5
Scope :	classifier

Operation : acquire

Documentation :	Starts sample&hold and conversion of input channel **channel**. It enables interrupt for ADC12. At the end of conversion, it calls interrupt service routine **isr_adc12()** to transfer converted result into the location defined by the parameter **value**.
Parameter(s) :	channel : byte sampletime : byte value : Bk1A1_Common_Codes$t_sensor&
Visibility :	public
Scope :	classifier

Operation : acquire_scale

Documentation :	Starts sample&hold and conversion of input channel **channel**. It enables interrupt for ADC12. At the end of conversion, it calls interrupt service routine **isr_adc12()** to scale acquired data by scaling factor **scale** and to transfer converted result into the location defined by the parameter **value**.
Parameter(s) :	channel : byte sampletime : byte value : Bk1A1_Common_Codes$t_sensor& scale : Bk1A1_Common_Codes$t_scaling
Visibility :	public
Scope :	classifier

4.4.13. Class : Buffers

Stereotypes :	<<SW>>
Documentation :	This class implements a set of fixed-length buffers, to be used for **Write Module Data** and **Read Module Data** use cases, respectively. Up to **MAXBUFFERS** buffers of 256 bytes each can be allocated (provided that the processor has enough memory). Each buffer can be used for either **Write** or **Read** operation, although the class guarantees that at least one buffer is allocated for both **Write** and **Read** use case. In particular, for either **Write** (if **command** = CMD_WRITE_DATA_x) or **Read** (for **command** = CMD_READ_DATA_x): •throws away any buffer of that type still being written but not yet complete (namely, for which the ready() operation has not yet been called, if any) •finds the first available buffer of that type •locks it and declares it being written; the user shall then fill the buffer and call the ready() operation when finished. After that, the buffer is queued... •returns a pointer to it Returns **null** if no buffer of the chosen type is available or command is not supported.

Operation : get

Documentation :	Returns pointer to first buffer containing data: •written, if **command** == CMD_WRITE_DATA_x (with x = 0 through 7). •to be read, if **command** == CMD_READ_DATA_x (with x = 0 through 7). Returns NULL in case of no buffer containing data.

	The argument length also returns the amount of data actually written in that buffer
Parameter(s) :	command : Bk1A1_Common_Codes$t_Commands length : ushort&
Return type :	byte
Type modifier :	*
Visibility :	public
Scope :	instance

Operation : lock

Documentation :	Allocates and locks one of the MAXBUFFERS available buffers. There are two types of buffers, to be used for **Write Module Data** and **Read Module Data** use cases, respectively. In particular, for either **Write** (if **command** = CMD_WRITE_DATA_x) or **Read** (for **command** = CMD_READ_DATA_x): •throws away any buffer of that type still being written but not yet complete (namely, for which the ready() operation has not yet been called, if any) •finds the first available buffer of that type •locks it and declares it being written; the user shall then fill the buffer and call the ready() operation when finished. After that, the buffer is queued... •returns a pointer to it Returns **null** if no buffer of the chosen type is available or command is not supported.
Parameter(s) :	command : Bk1A1_Common_Codes$t_Commands
Return type :	byte
Type modifier :	*
Visibility :	public
Scope :	instance

Operation : ready

Documentation :	Declares that the buffer pointed to by **buffer** has been filled in with **length** bytes and is ready to be read Returns **false** if either no buffer or more than one buffer has been allocated. In the latter case declares that all of them are ready. Otherwise it return **true**.
Parameter(s) :	command : Bk1A1_Common_Codes$t_Commands length : ushort
Return type :	bool
Visibility :	public
Scope :	instance

Operation : release

Documentation :	Releases buffer identified by parameter **buffer**. No more reading can be done on that buffer, which is then available for further use (via the **lock()** operation) Returns false is no open buffer matches the parameter **buffer**. Otherwise it returns **true**.
Parameter(s) :	buffer : byte*
Return type :	bool
Visibility :	public
Scope :	instance

4.4.14. Class : t_Commands_1B8

Stereotypes :	<<SW>> <<enumeration>>

Attribute : CMD_ACTUATE_COIL

Stereotypes :	<<Constant>>
Initial Value :	Bk1A1_Common_Codes::CMD_WRITE_DATA_0
Scope :	instance

Attribute : CMD_SETRESET_MAGNETOMETER

Stereotypes :	<<Constant>>
Initial Value :	Bk1A1_Common_Codes::CMD_COMMAND_0
Scope :	instance

4.4.15. Class : t_Commands

Stereotypes :	<<SW>> <<enumeration>>
Documentation :	Lists all available command codes and the corresponding value. There is one command for each use case, Removing a command from this class removes the corresponding use case. Adding additional commands requires adding the appropriate code to properly interpret and execute the command.

Specific for Bk1B22 project

Attribute : CMD_WRITE_DATA_0

Stereotypes :	<<Constant>>
Documentation :	Used for **Write (Module) Data** use case; write **Designer**-defined data of **Designer**-type 0.
Initial Value :	0x40
Scope :	instance

Attribute : CMD_COMMAND_0

Stereotypes :	<<Constant>>
Documentation :	Issues a **Designer**-defined command of **Designer**-type 0.
Initial Value :	0xC0
Scope :	instance

Associated to Bk1B45_Slave Use Cases which are used by Bk1B22 Use Cases

Attribute : CMD_SET_CONFIGURATION

Stereotypes :	<<Constant>>
Initial Value :	0x10
Scope :	instance

Attribute : CMD_RESET_CONFIGURATION

Stereotypes :	<<Constant>>
Initial Value :	0x11
Scope :	instance

Attribute : CMD_GET_CONFIGURATION

Stereotypes :	<<Constant>>
Documentation :	Used to return module ID
Initial Value :	0x24
Scope :	instance

Attribute : CMD_GET_STATUS

Stereotypes :	<<Constant>>
Documentation :	Used for **Get (Module) Status** use case.
Initial Value :	0x20
Scope :	instance

Attribute : CMD_GET_HOUSEKEEPING

Stereotypes :	<<Constant>>
Documentation :	Used for **Get (Module) Housekeeping** use case.
Initial Value :	0x21
Scope :	instance

Attribute : CMD_CALIBRATE

Stereotypes :	<<Constant>> <<TBD>>
Documentation :	Used to send calibration data to the **Slave**.
Initial Value :	0x12
Scope :	instance

4.4.16. Class t_LastError

Defines codes for the last error which occurred in the slave.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
CRC_ERROR (private) It indicates that an error has been detected in the CRC of the last message or command coming from **CPU** actor.		
LENGTH_ERROR (private) It indicates that the lenght of the last message/command does not match the expected length. Either: • for **Command Only** messages, there were either more or less byte than the expected sequence. • for **Write Data** messages, the CS has terminated the transfer either too early or too late, when compared with the length field of the message. • for **Read Data** messages, it indicates that the **CPU** actor has terminated the communication either too early or too late to transfer the amount of dta declared by the **System**.		
NO_BUFFER_AVAILABLE (private) During **Write Module Data** use case, it indicates that there is no buffer available fro storing data.		
WRONG_HOUSEKEEPING (private)		

NO_DATA_AVAILABLE (private)		
TIMEOUT (private)		
WRONG_LENGTH (private)		
WRONG_COMMAND (private)		
WRONG_CRC (private)		
WRONG_BUFFER (private)		
NO_ERROR (private)		0
UNRECOGNIZED_COMMAND (private)		

4.4.17. Class t_scaling

A class to define all calibration factors for **housekeeping** (telemetry and commands).

The use of t_scaling for sensors shall be the following:
1) a sensor is acquired from the ADC (or any other source, when applicable)
2) the offset attribute is subtracted from the measurement;
3) the result of above step is multiplied by ((float)gain/SCALING_FACTOR);
4) the result is stored into the appropriate **housekeeping[]** vector.

The use of t_scaling for actuators shall be the following:
1) a value is taken from the appropriate **housekeeping[]** vector;
2) the offset attribute is subtracted from the value;
2) the result is multiplied by ((float))gain/SCALING_FACTOR);
4) the result is then sent to the DAC (or any other peripheral, when applicable)

Operation scale takes care of the above operations.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
offset (public)	Bk1A1_Common _Codes$t_sensor	
gain (public)	Bk1A1_Common _Codes$t_sensor	
SCALING_FACTOR (public)	Bk1A1_Common _Codes$t_sensor	(1<<13)

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
scale (public) Scales input value ADCvalue by the internal scaling factor and returns scaled value, according to the following formula: ((long)(value - offset) * scaling) / 2^13	value : Bk1A1_Common_Codes$t_sensor	Bk1A1_Common_Codes$t_sensor

4.4.18. Class Bk1B22_Errors

This enumeration class contains the masks used to identify each error type in 1B22_Magnetic_Attitude_Subsystem error word, which is in MAGNETIC_ACTUATOR_ERRORS position of housekeeping vector in 1B45.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
ON_CURR_EXCESS (private)		0x1
ON_CURR_LACK (private)		0x2
OFF_CURR_EXCESS (private)		0x4
ERR_COIL_ENABLE (private)		0x8
ERR_COIL_STATUS (private)		0x10
ERR_COIL_ATTACH (private)		0x20
ERR_MAGN_ENABLE (private)		0x40
ERR_MAGN_ATTACH (private)		0x80
LOW_PDB_VOLT (private)		0x100
ERR10 (private)		0x200
ERR11 (private)		0x400
ERR12 (private)		0x800
ERR13 (private)		0x1000
ERR14 (private)		0x2000
ERR15 (private)		0x4000
ERR16 (private)		0x8000

Capitolo 5

5. Progetto software del controllo di assetto magnetico

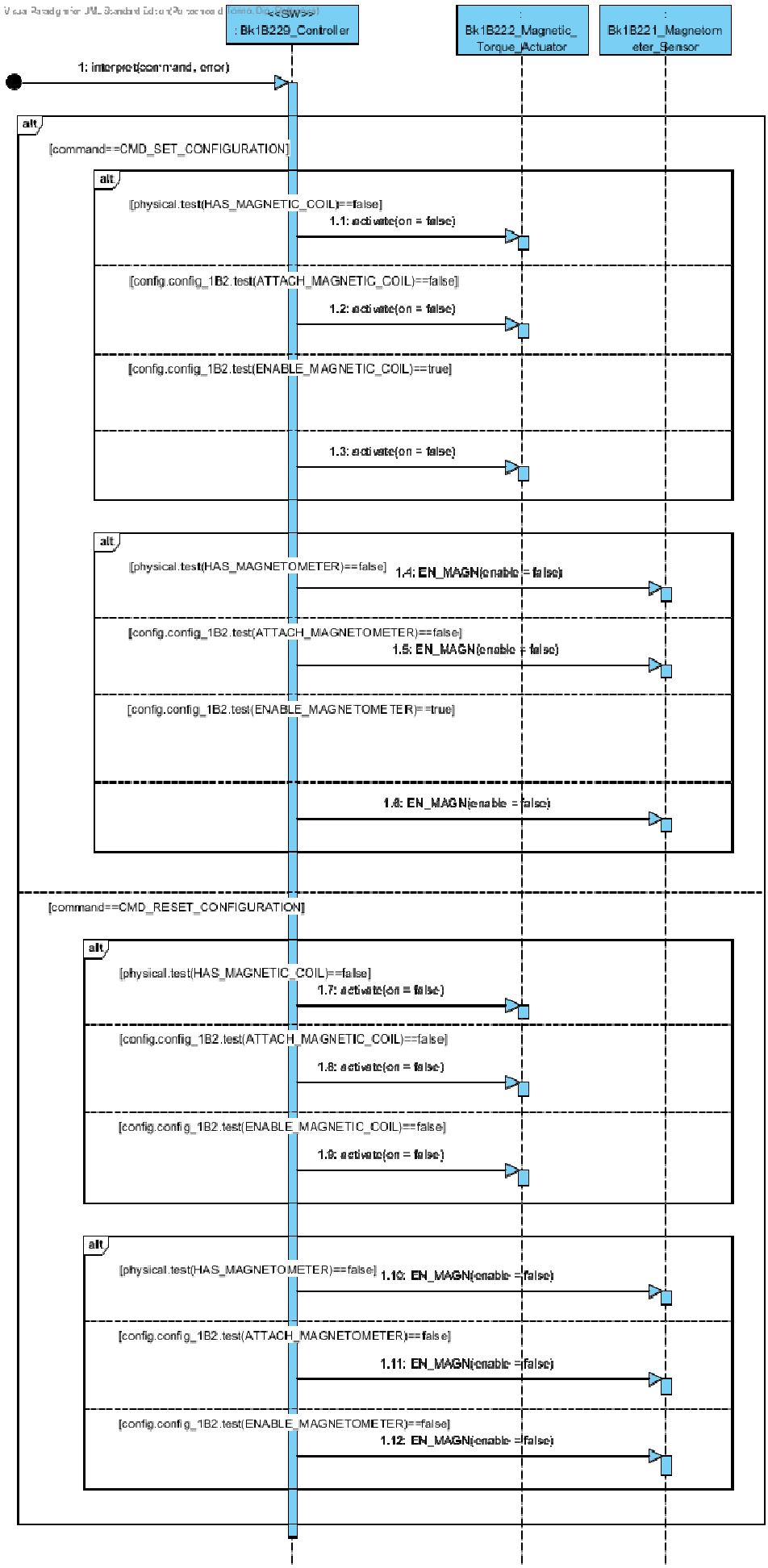
In questo capitolo verranno mostrati i diagrammi di sequenza UML, creati utilizzando il tool *Visual Paradigm*, relativi al progetto del controllo d'assetto magnetico del satellite AraMiS. Questi rappresentano, in forma visiva ed in modo intuitivo, la procedura attraverso cui vengono realizzati dal sistema i casi d'uso, descritti nel cap. 3, ovvero le specifiche e le funzionalità del progetto. Nei diagrammi sono chiaramente visibili tutte le interazioni tra gli elementi che costituiscono il sistema, cioè gli oggetti descritti nel cap. 4.
Inoltre, il tool consente di generare automaticamente, a partire dai diagrammi delle classi, tutte le strutture ed i file d'intestazione, ovvero i "gusci" esterni delle funzioni C++ che andranno a costituire il progetto software nel suo complesso. Il corpo centrale delle principali funzioni, invece, viene semplicemente trasposto in codice a partire dai diagrammi di sequenza, che ne costituiscono la versione grafica e, pertanto, meglio comprensibile.
Nei sottocapitoli 5.1 e 5.2 verranno presentati, rispettivamente, i diagrammi di sequenza ed il codice ad essi corrispondente, tenendo presente che una singola funzione (ad es. *interpret*), può essere descritta da più diagrammi di sequenza, qualora essa implementi più di un caso d'uso.

5.1. Diagrammi di sequenza

In this paragraph, the sequence diagrams of project 1B22 Magnetic Attitude Subsystem are shown. They represent the actual realizing procedure of system features.

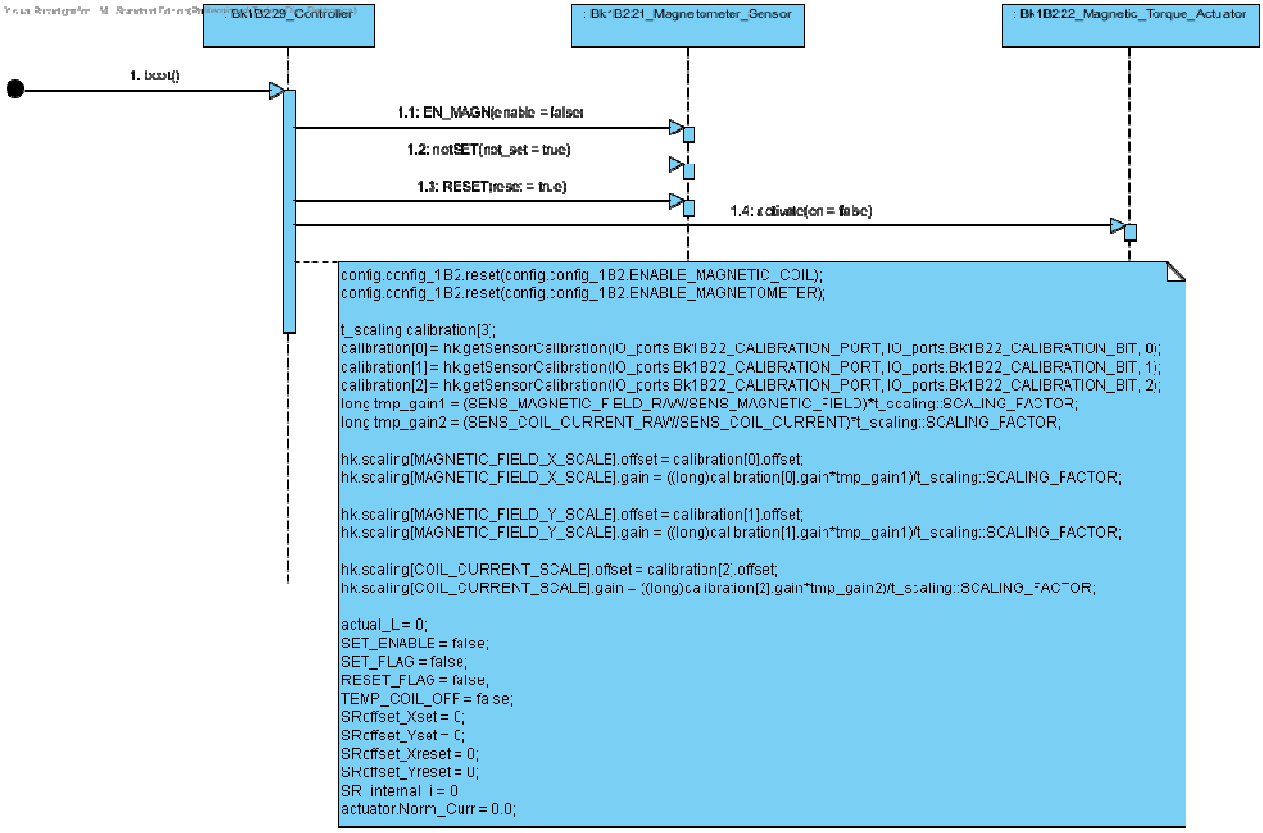
5.1.1. Enable/Disable Circuit

This diagram realizes Attach Coil, Attach Magnetometer, Detach Coil, Detach Magnetometer, Disable Coil, Disable Magnetometer, Enable Coil, Enable Magnetometer use cases.
If *interpret* operation receives a CMD_SET_CONFIGURATION or a CMD_RESET_CONFIGURATION command, it checks what bits have been set/reset and enables/disables the corresponding block (Magnetometer or Magnetic Actuator.
The relevant bits are: ENABLE_MAGNETOMETER, ENABLE_MAGNETIC_COIL, ATTACH_MAGNETOMETER, ATTACH_MAGNETIC_COIL of t_ConfigWord_1B2 and HAS_MAGNETOMETER, HAS_MAGNETIC_COIL of t_MechanicalConfiguration_1B8.
Enabling consists in doing nothing, because actual enable signal will be set only when necessary, to reduce power consumption.


: Bk1B229_Controller
: Bk1B222_Magnetic_Torque_Actuator
: Bk1B221_Magnetometer_Sensor
1: interpret(command, error)
alt
[command==CMD_SET_CONFIGURATION]
alt
[physical.test(HAS_MAGNETIC_COIL)==false]
1.1: activate(on = false)
[config.config_1B2.test(ATTACH_MAGNETIC_COIL)==false]
1.2: activate(on = false)
[config.config_1B2.test(ENABLE_MAGNETIC_COIL)==true]
1.3: activate(on = false)
alt
[physical.test(HAS_MAGNETOMETER)==false]
1.4: EN_MAGN(enable = false)
[config.config_1B2.test(ATTACH_MAGNETOMETER)==false]
1.5: EN_MAGN(enable = false)
[config.config_1B2.test(ENABLE_MAGNETOMETER)==true]
1.6: EN_MAGN(enable = false)
[command==CMD_RESET_CONFIGURATION]
alt
[physical.test(HAS_MAGNETIC_COIL)==false]
1.7: activate(on = false)
[config.config_1B2.test(ATTACH_MAGNETIC_COIL)==false]
1.8: activate(on = false)
[config.config_1B2.test(ENABLE_MAGNETIC_COIL)==false]
1.9: activate(on = false)
alt
[physical.test(HAS_MAGNETOMETER)==false]
1.10: EN_MAGN(enable = false)
[config.config_1B2.test(ATTACH_MAGNETOMETER)==false]
1.11: EN_MAGN(enable = false)
[config.config_1B2.test(ENABLE_MAGNETOMETER)==false]
1.12: EN_MAGN(enable = false)

5.1.2. Boot

This diagram shows in detail how the Boot use case is realized. It initializes all required elements at bootstrap.



5.1.3. Acquire-Integral-System_SetReset

This diagram realizes (totally or partially) the Get Magnetic Field**,** Get Coil Current**,** Actuate Coil**,** Housekeeping**,** Supervision**,** System SR Magnetometer**,** SetReset Magnetometer and In Flight Calibration use cases by using the *housekeeping(index)* operation.

According to the value of *index* it realizes different use cases:

- index = MAGNETIC_FIELD_X-2 : prepares magnetic field acquiring by turning coil off (if operating) and magnetometer block on; checks if a Set/Reset command is received: in this case, the next two housekeeping cycles the system will be in SR MODE, else in NORMAL MODE --> part of System SR Magnetometer and SetReset Magnetometer

- index = MAGNETIC_FIELD_X : acquires magnetic field X component (NORMAL MODE) or magnetic field offset X component after a Set pulse, first, and after a Reset pulse, next cycle (SR MODE) --> Get Magnetic Field**,** part of System SR Magnetometer and SetReset Magnetometer, part of In Flight Calibration

- index = MAGNETIC_FIELD_Y : acquires magnetic field Y component (NORMAL MODE) or magnetic field offset Y component after a Set pulse, first, and after a Reset pulse, next cycle (SR MODE) --> Get Magnetic Field, part of System SR Magnetometer and SetReset Magnetometer, part of In Flight Calibration

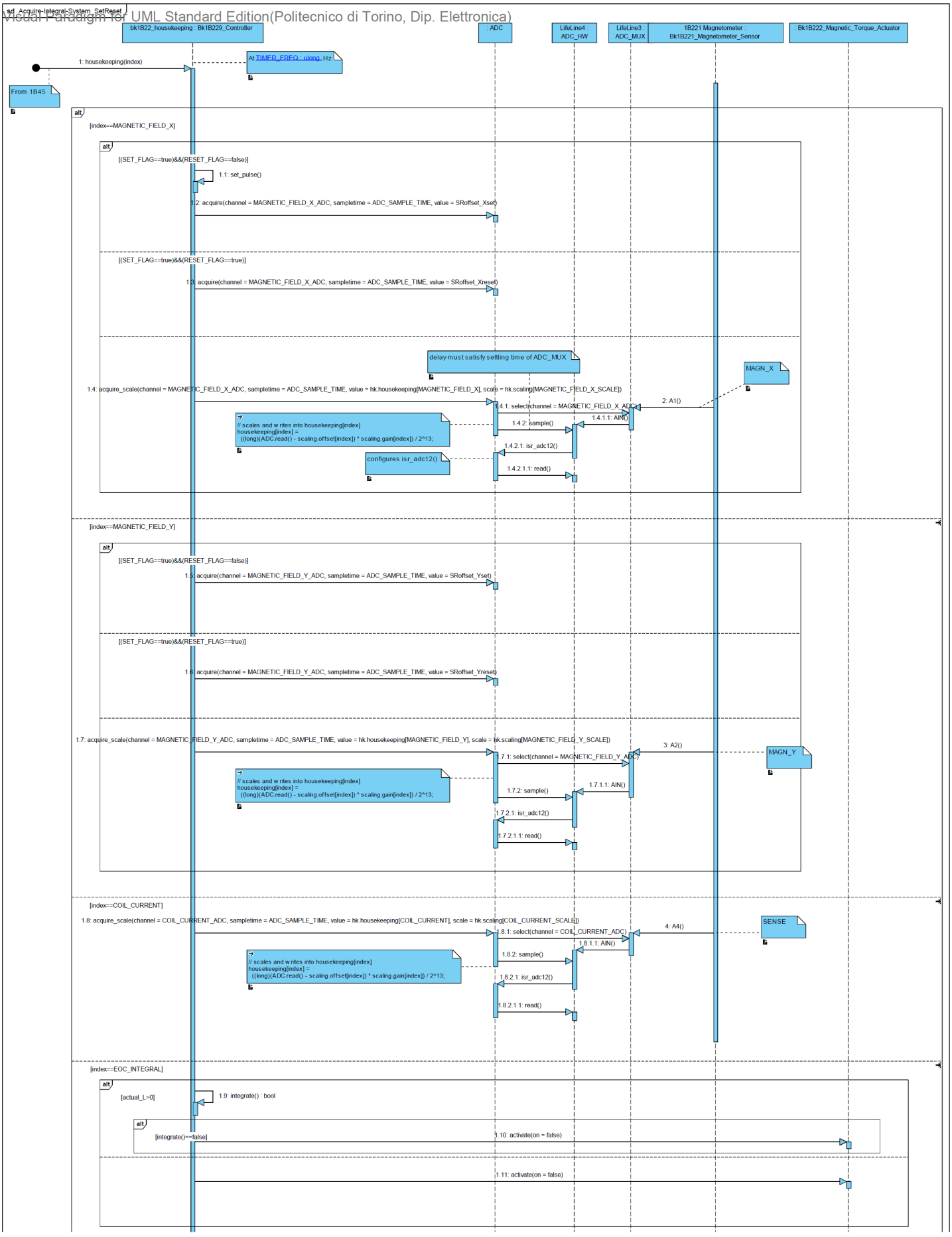
- index = MAGNETIC_FIELD_Y+1 : gives a Reset Pulse to magnetometer sensor (SR MODE) or saves offset components in hk.scaling vector (if SR is just completed) and turns magnetometer off (NORMAL MODE) --> part of System SR Magnetometer and SetReset Magnetometer, part of In Flight Calibration

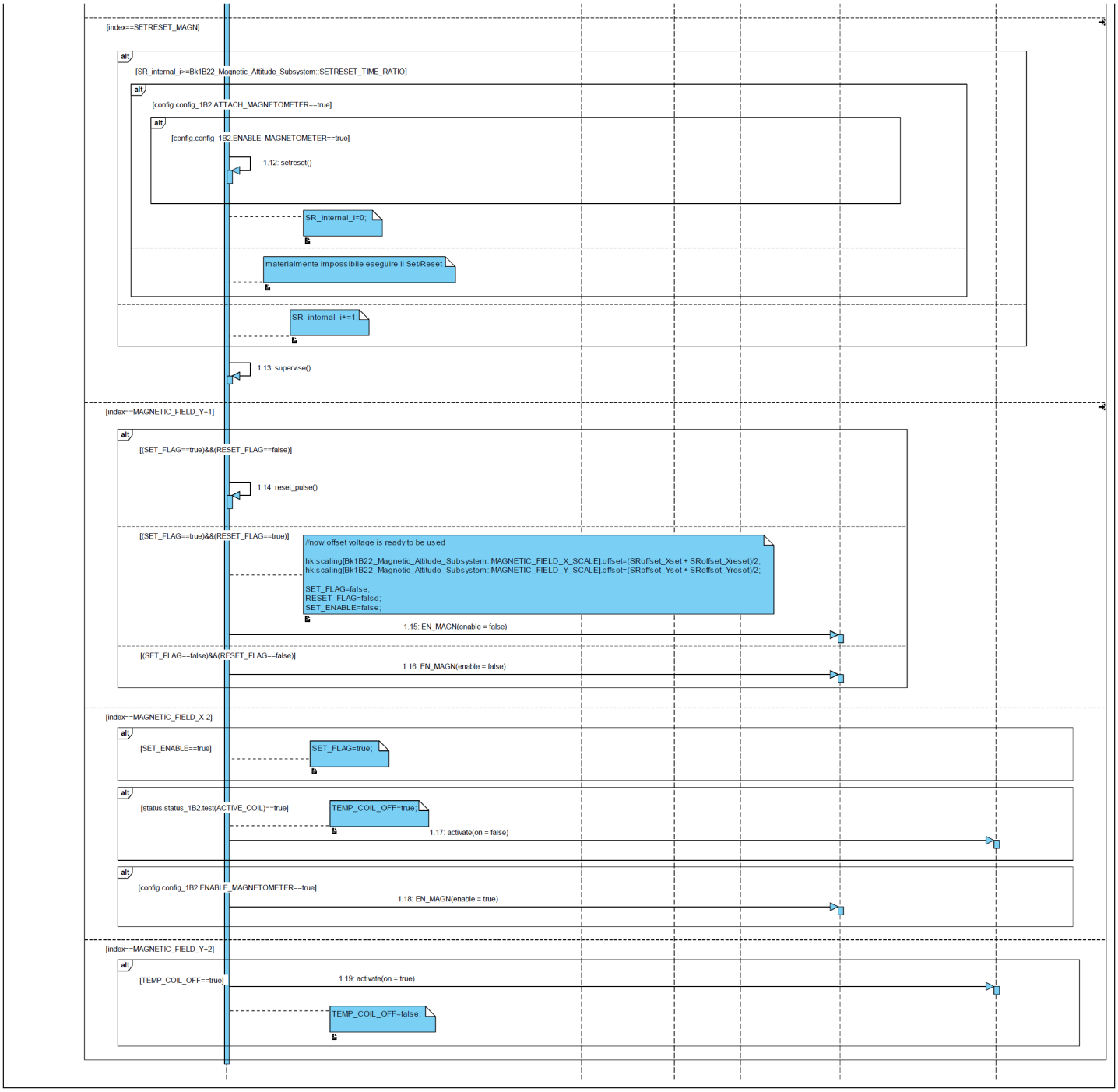
- index = MAGNETIC_FIELD_Y+2 : calls *supervise()* operation and turns coil on, if it had been temporarily disabled before --> part of Supervision

- index = COIL_CURRENT : acquires coil current value --> Get Coil Current

- index = EOC_INTEGRAL : calls *integrate()* operation to decrease, on every housekeeping cycle, angular momentum value, until it reaches 0. In this case coil driving is stopped. --> part of Actuate Coil

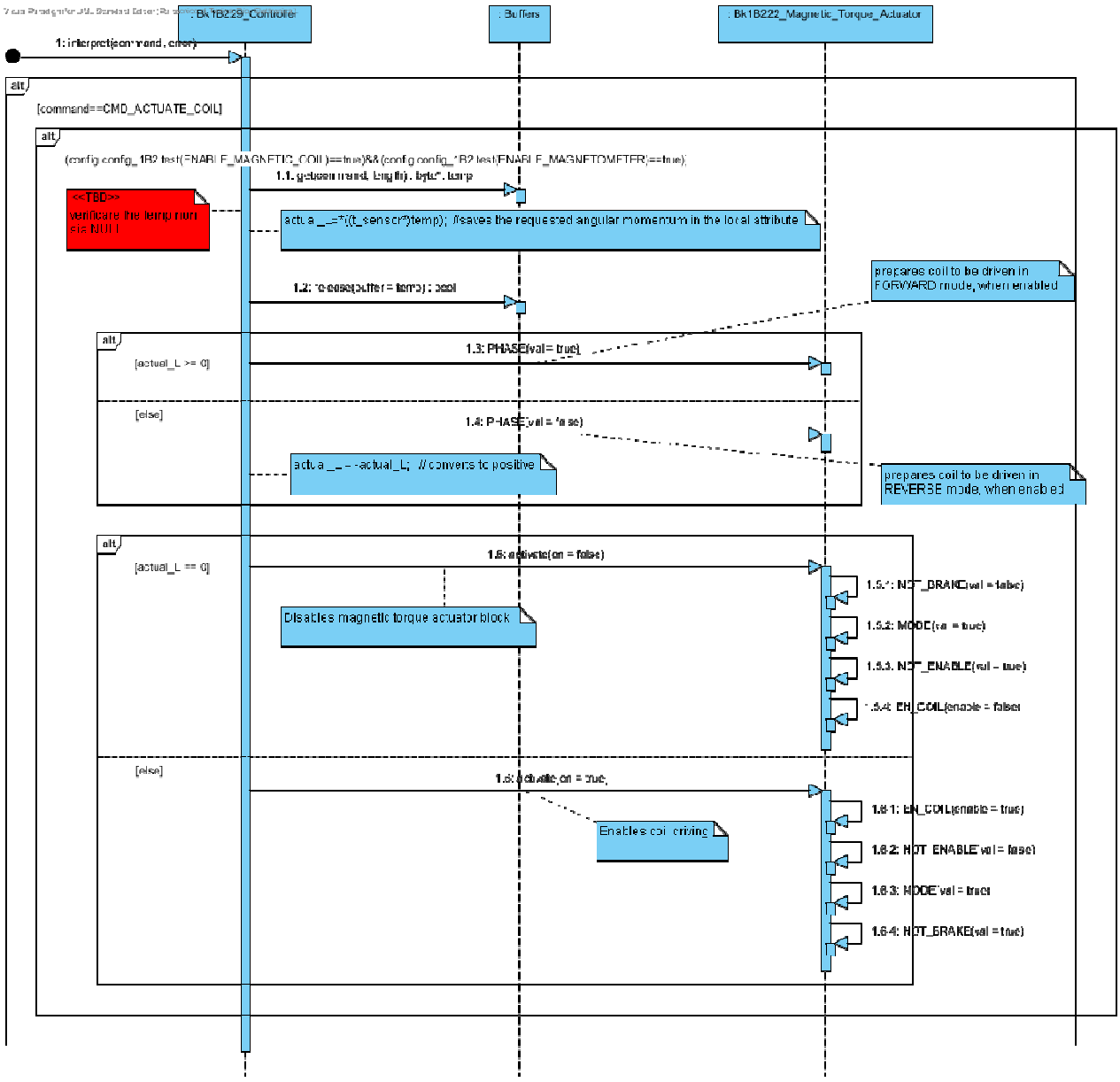
- index = SETRESET_MAGN : counts time for automatic Set/Reset procedure launching (every SETRESET_PERIOD seconds) --> part of System SR Magnetometer


sd Acquire-Integral-System_SetReset
Visual Paradigm for UML Standard Edition(Politecnico di Torino, Dip. Elettronica)
bk1B22_housekeeping : Bk1B229_Controller
: ADC
LifeLine4 : ADC_HW
LifeLine3 : ADC_MUX
1B221 Magnetometer : Bk1B221_Magnetometer_Sensor
Bk1B222_Magnetic_Torque_Actuator
1: housekeeping(index)
At TIMER_FREQ : ulong; Hz
From 1B45
alt
[index==MAGNETIC_FIELD_X]
alt
[(SET_FLAG==true)&&(RESET_FLAG==false)]
1.1: set_pulse()
1.2: acquire(channel = MAGNETIC_FIELD_X_ADC, sampletime = ADC_SAMPLE_TIME, value = SRoffset_Xset)
[(SET_FLAG==true)&&(RESET_FLAG==true)]
1.3: acquire(channel = MAGNETIC_FIELD_X_ADC, sampletime = ADC_SAMPLE_TIME, value = SRoffset_Xreset)
delay must satisfy settling time of ADC_MUX
MAGN_X
1.4: acquire_scale(channel = MAGNETIC_FIELD_X_ADC, sampletime = ADC_SAMPLE_TIME, value = hk.housekeeping[MAGNETIC_FIELD_X], scale = hk.scaling[MAGNETIC_FIELD_X_SCALE])
2: A1()
1.4.1: select(channel = MAGNETIC_FIELD_X_ADC)
1.4.1.1: AIN()
1.4.2: sample()
// scales and w rites into housekeeping[index]
housekeeping[index] =
((long)(ADC.read() - scaling.offset[index]) * scaling.gain[index]) / 2^13;
1.4.2.1: isr_adc12()
configures isr_adc12()
1.4.2.1.1: read()
[index==MAGNETIC_FIELD_Y]
alt
[(SET_FLAG==true)&&(RESET_FLAG==false)]
1.5: acquire(channel = MAGNETIC_FIELD_Y_ADC, sampletime = ADC_SAMPLE_TIME, value = SRoffset_Yset)
[(SET_FLAG==true)&&(RESET_FLAG==true)]
1.6: acquire(channel = MAGNETIC_FIELD_Y_ADC, sampletime = ADC_SAMPLE_TIME, value = SRoffset_Yreset)
1.7: acquire_scale(channel = MAGNETIC_FIELD_Y_ADC, sampletime = ADC_SAMPLE_TIME, value = hk.housekeeping[MAGNETIC_FIELD_Y], scale = hk.scaling[MAGNETIC_FIELD_Y_SCALE])
3: A2()
MAGN_Y
1.7.1: select(channel = MAGNETIC_FIELD_Y_ADC)
1.7.1.1: AIN()
1.7.2: sample()
// scales and w rites into housekeeping[index]
housekeeping[index] =
((long)(ADC.read() - scaling.offset[index]) * scaling.gain[index]) / 2^13;
1.7.2.1: isr_adc12()
1.7.2.1.1: read()
[index==COIL_CURRENT]
1.8: acquire_scale(channel = COIL_CURRENT_ADC, sampletime = ADC_SAMPLE_TIME, value = hk.housekeeping[COIL_CURRENT], scale = hk.scaling[COIL_CURRENT_SCALE])
4: A4()
SENSE
1.8.1: select(channel = COIL_CURRENT_ADC)
1.8.1.1: AIN()
1.8.2: sample()
// scales and w rites into housekeeping[index]
housekeeping[index] =
((long)(ADC.read() - scaling.offset[index]) * scaling.gain[index]) / 2^13;
1.8.2.1: isr_adc12()
1.8.2.1.1: read()
[index==EOC_INTEGRAL]
alt
[actual_L>0]
1.9: integrate() : bool
alt
[integrate()==false]
1.10: activate(on = false)
1.11: activate(on = false)


[index==SETRESET_MAGN]
alt
[SR_internal_i>=Bk1B22_Magnetic_Attitude_Subsystem::SETRESET_TIME_RATIO]
alt
[config.config_1B2.ATTACH_MAGNETOMETER==true]
alt
[config.config_1B2.ENABLE_MAGNETOMETER==true]
1.12: setreset()
SR_internal_i=0;
materialmente impossibile eseguire il Set/Reset
SR_internal_i+=1;
1.13: supervise()
[index==MAGNETIC_FIELD_Y+1]
alt
[(SET_FLAG==true)&&(RESET_FLAG==false)]
1.14: reset_pulse()
[(SET_FLAG==true)&&(RESET_FLAG==true)]
//now offset voltage is ready to be used
hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_SCALE].offset=(SRoffset_Xset + SRoffset_Xreset)/2;
hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_SCALE].offset=(SRoffset_Yset + SRoffset_Yreset)/2;
SET_FLAG=false;
RESET_FLAG=false;
SET_ENABLE=false;
1.15: EN_MAGN(enable = false)
[(SET_FLAG==false)&&(RESET_FLAG==false)]
1.16: EN_MAGN(enable = false)
[index==MAGNETIC_FIELD_X-2]
alt
[SET_ENABLE==true]
SET_FLAG=true;
alt
[status.status_1B2.test(ACTIVE_COIL)==true]
TEMP_COIL_OFF=true;
1.17: activate(on = false)
alt
[config.config_1B2.ENABLE_MAGNETOMETER==true]
1.18: EN_MAGN(enable = true)
[index==MAGNETIC_FIELD_Y+2]
alt
[TEMP_COIL_OFF==true]
1.19: activate(on = true)
TEMP_COIL_OFF=false;

5.1.4. ActuateCoil

Together with **Acquire-Integral-System_SetReset** sequence diagram, this diagram realizes the Actuate Coil use case.
If *interpret* operation receives a CMD_ACTUATE_COIL command, system reads required angular momentum value from buffer, properly initializes coil driver control signals and turns the driver on.
Angular momentum value is decreased every housekeeping cycle by operation integrate. Driver will be turned off when this value goes to 0 (See **Acquire-Integral-System_SetReset** diagram).

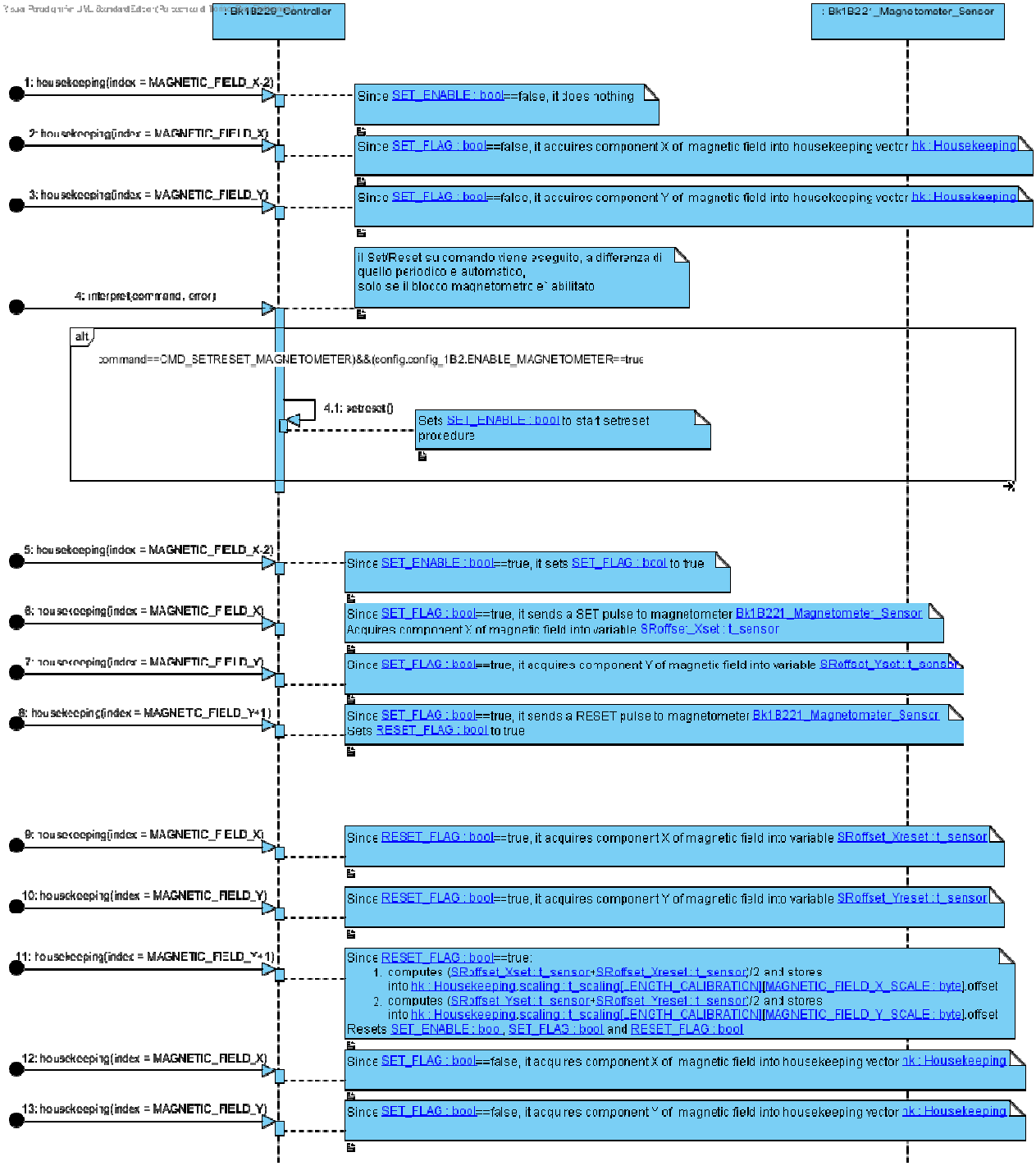


5.1.5. SetReset_Magnetometer

This diagram shows how Set/Reset procedure is realized, in case that the CMD_SETRESET_MAGNETOMETER command is interpreted after a complete magnetic field acquiring and before a new one.
The system performs Set/Reset steps by the following operations:

- *interpret(command,error)* when command is CMD_SETRESET_MAGNETOMETER (see diagram below)
- *housekeeping(index)*, when index is MAGNETIC_FIELD_X-2, MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y, MAGNETIC_FIELD_Y+1 (see **Acquire-Integral-System_SetReset** diagram too)

Set/Reset is also automatically launched by the system, about every 10 minutes, using *housekeeping*(SETRESET_MAGN) (see **Acquire-Integral-System_SetReset** diagram). After SET_ENABLE flag is set by operation *setreset()*, Set/Reset procedure is the same, irrespective of the activation type (on command or automatic).

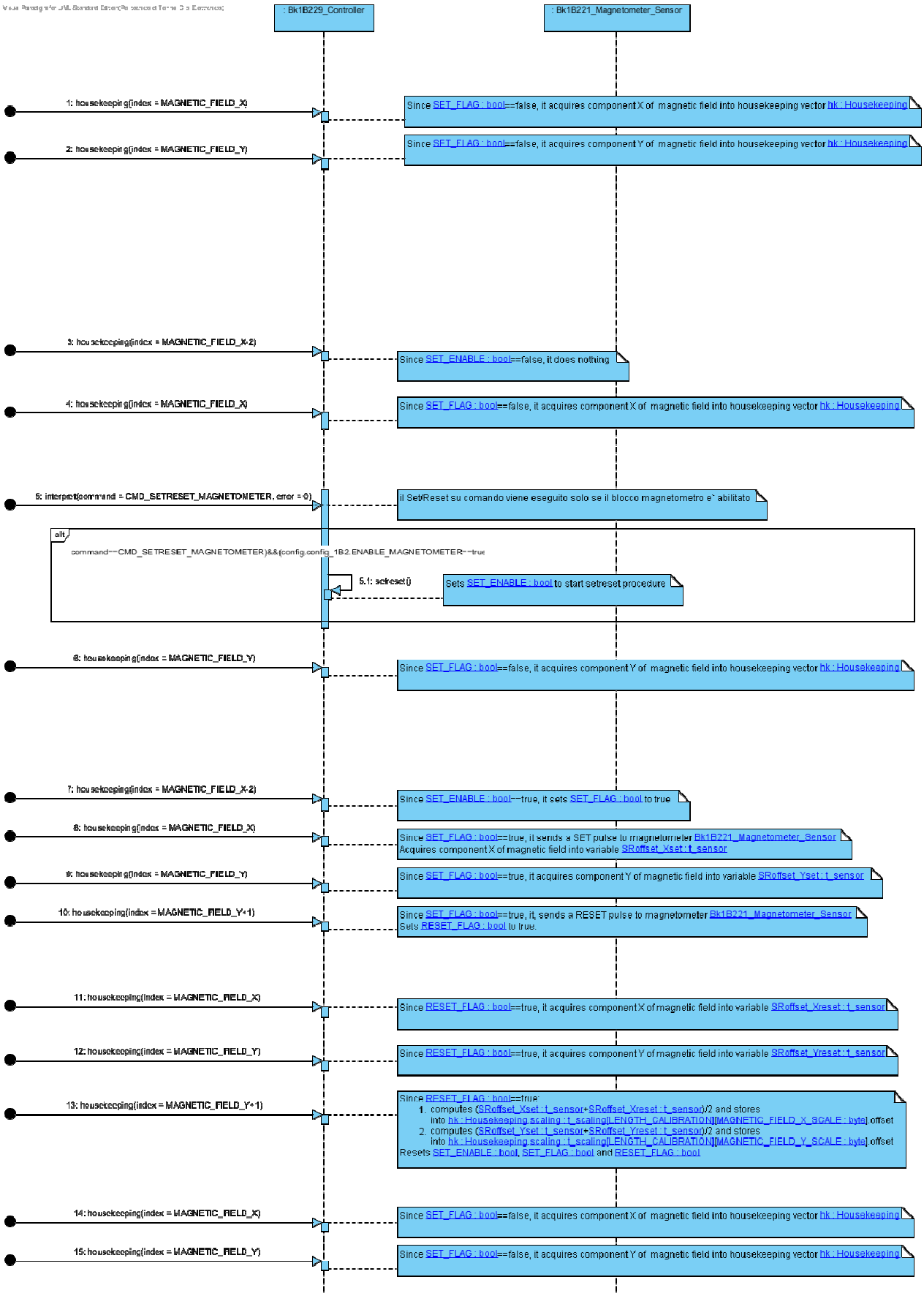

: Bk1B229_Controller
: Bk1B22'_Magnetometer_Sensor
1: housekeeping(index = MAGNETIC_FIELD_X-2)
Since SET_ENABLE : bool==false, it does nothing
2: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==false, it accuires component X of magnetic field into housekeeping vector hk : Housekeeping
3: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==false, it accuires component Y of magnetic field into housekeeping vector hk : Housekeeping
il SetReset su comando viene eseguito, a differenza di
quello periodico e automatico,
solo se il blocco magnetometro e' abilitato
4: interpret(command, error)
alt
command==CMD_SETRESET_MAGNETOMETER)&&(config.config_1B2.ENABLE_MAGNETOMETER==true
4.1: setreset()
Sets SET_ENABLE : bool to start setreset
procedure
5: housekeeping(index = MAGNETIC_FIELD_X-2)
Since SET_ENABLE : bool==true, it sets SET_FLAG : bool to true
6: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==true, it sends a SET pulse to magnetometer Bk1B221_Magnetometer_Sensor
Acquires component X of magnetic field into variable SRoffset_Xset : t_sensor
7: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==true, it acquires component Y of magnetic field into variable SRoffset_Yset : t_sensor
8: housekeeping(index = MAGNETIC_FIELD_Y+1)
Since SET_FLAG : bool==true, it sends a RESET pulse to magnetometer Bk1B221_Magnetometer_Sensor
Sets RESET_FLAG : bool to true
9: housekeeping(index = MAGNETIC_FIELD_X)
Since RESET_FLAG : bool==true, it acquires component X of magnetic field into variable SRoffset_Xreset : t_sensor
10: housekeeping(index = MAGNETIC_FIELD_Y)
Since RESET_FLAG : bool==true, it acquires component Y of magnetic field into variable SRoffset_Yreset : t_sensor
11: housekeeping(index = MAGNETIC_FIELD_Y+1)
Since RESET_FLAG : bool==true:
1. computes (SRoffset_Xset : t_sensor+SRoffset_Xreset : t_sensor)/2 and stores
into hk : Housekeeping.scaling : t_scaling[_ENGTH_CALIBRATION][MAGNETIC_FIELD_X_SCALE : byte].offset
2. computes (SRoffset_Yset : t_sensor+SRoffset_Yreset : t_sensor)/2 and stores
into hk : Housekeeping.scaling : t_scaling[_ENGTH_CALIBRATION][MAGNETIC_FIELD_Y_SCALE : byte].offset
Resets SET_ENABLE : boo , SET_FLAG : bool and RESET_FLAG : bool
12: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==false, it acquires component X of magnetic field into housekeeping vector hk : Housekeeping
13: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==false, it acquires component Y of magnetic field into housekeeping vector hk : Housekeeping

5.1.6. SetReset_Magnetometer - case 2

This diagram shows how Set/Reset procedure is realized, in case that the CMD_SETRESET_MAGNETOMETER command is interpreted during a magnetic field acquiring.
The system performs Set/Reset steps by the following operations:

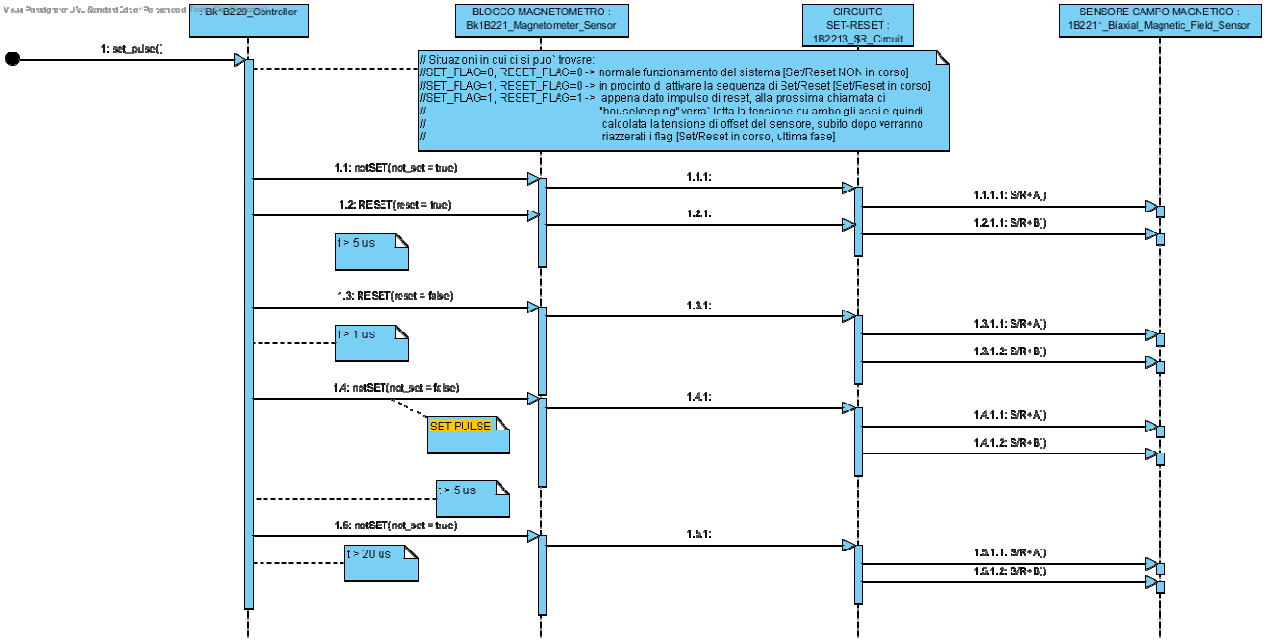
- *interpret(command,error)* when command is CMD_SETRESET_MAGNETOMETER (see diagram below)
- *housekeeping(index)*, when index is MAGNETIC_FIELD_X-2, MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y, MAGNETIC_FIELD_Y+1 (see **Acquire-Integral-System_SetReset** diagram too)

Set/Reset is also automatically launched by the system, about every 10 minutes, using *housekeeping*(SETRESET_MAGN) (see **Acquire-Integral-System_SetReset** diagram). After SET_ENABLE flag is set by operation *setreset()*, Set/Reset procedure is the same, irrespective of the activation type (on command or automatic).


: Bk1B229_Controller
: Bk1B221_Magnetometer_Sensor
1: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==false, it acquires component X of magnetic field into housekeeping vector hk : Housekeeping
2: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==false, it acquires component Y of magnetic field into housekeeping vector hk : Housekeeping
3: housekeeping(index = MAGNETIC_FIELD_X-2)
Since SET_ENABLE : bool==false, it does nothing
4: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==false, it acquires component X of magnetic field into housekeeping vector hk : Housekeeping
5: interpret(command = CMD_SETRESET_MAGNETOMETER, error = 0)
il SetReset su comando viene eseguito solo se il blocco magnetometro e` abilitato
alt
command==CMD_SETRESET_MAGNETOMETER)&&(config.config_1B2.ENABLE_MAGNETOMETER==true
5.1: setreset()
Sets SET_ENABLE : bool to start setreset procedure
6: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==false, it acquires component Y of magnetic field into housekeeping vector hk : Housekeeping
7: housekeeping(index = MAGNETIC_FIELD_X-2)
Since SET_ENABLE : bool==true, it sets SET_FLAG : bool to true
8: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==true, it sends a SET pulse to magnetometer Bk1B221_Magnetometer_Sensor
Acquires component X of magnetic field into variable SRoffset_Xset : t_sensor
9: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==true, it acquires component Y of magnetic field into variable SRoffset_Yset : t_sensor
10: housekeeping(index = MAGNETIC_FIELD_Y+1)
Since SET_FLAG : bool==true, it, sends a RESET pulse to magnetometer Bk1B221_Magnetometer_Sensor
Sets RESET_FLAG : bool to true.
11: housekeeping(index = MAGNETIC_FIELD_X)
Since RESET_FLAG : bool==true, it acquires component X of magnetic field into variable SRoffset_Xreset : t_sensor
12: housekeeping(index = MAGNETIC_FIELD_Y)
Since RESET_FLAG : bool==true, it acquires component Y of magnetic field into variable SRoffset_Yreset : t_sensor
13: housekeeping(index = MAGNETIC_FIELD_Y+1)
Since RESET_FLAG : bool==true
1. computes (SRoffset_Xset : t_sensor+SRoffset_Xreset : t_sensor)/2 and stores
into hk : Housekeeping.scaling : t_scaling[LENGTH_CALIBRATION][MAGNETIC_FIELD_X_SCALE : byte].offset
2. computes (SRoffset_Yset : t_sensor+SRoffset_Yreset : t_sensor)/2 and stores
into hk : Housekeeping.scaling : t_scaling[LENGTH_CALIBRATION][MAGNETIC_FIELD_Y_SCALE : byte].offset
Resets SET_ENABLE : bool, SET_FLAG : bool and RESET_FLAG : bool
14: housekeeping(index = MAGNETIC_FIELD_X)
Since SET_FLAG : bool==false, it acquires component X of magnetic field into housekeeping vector hk : Housekeeping
15: housekeeping(index = MAGNETIC_FIELD_Y)
Since SET_FLAG : bool==false, it acquires component Y of magnetic field into housekeeping vector hk : Housekeeping

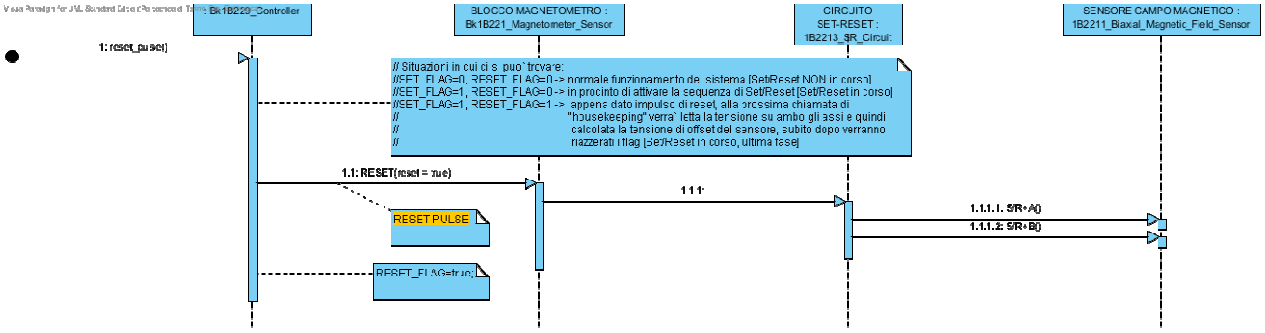
5.1.7. Bk1B22_Controller.set_pulse()

Diagram showing how the system gives a Set pulse to the Magnetic Field Sensor,in the Set/Reset procedure, respecting times that are indicated in datasheet.



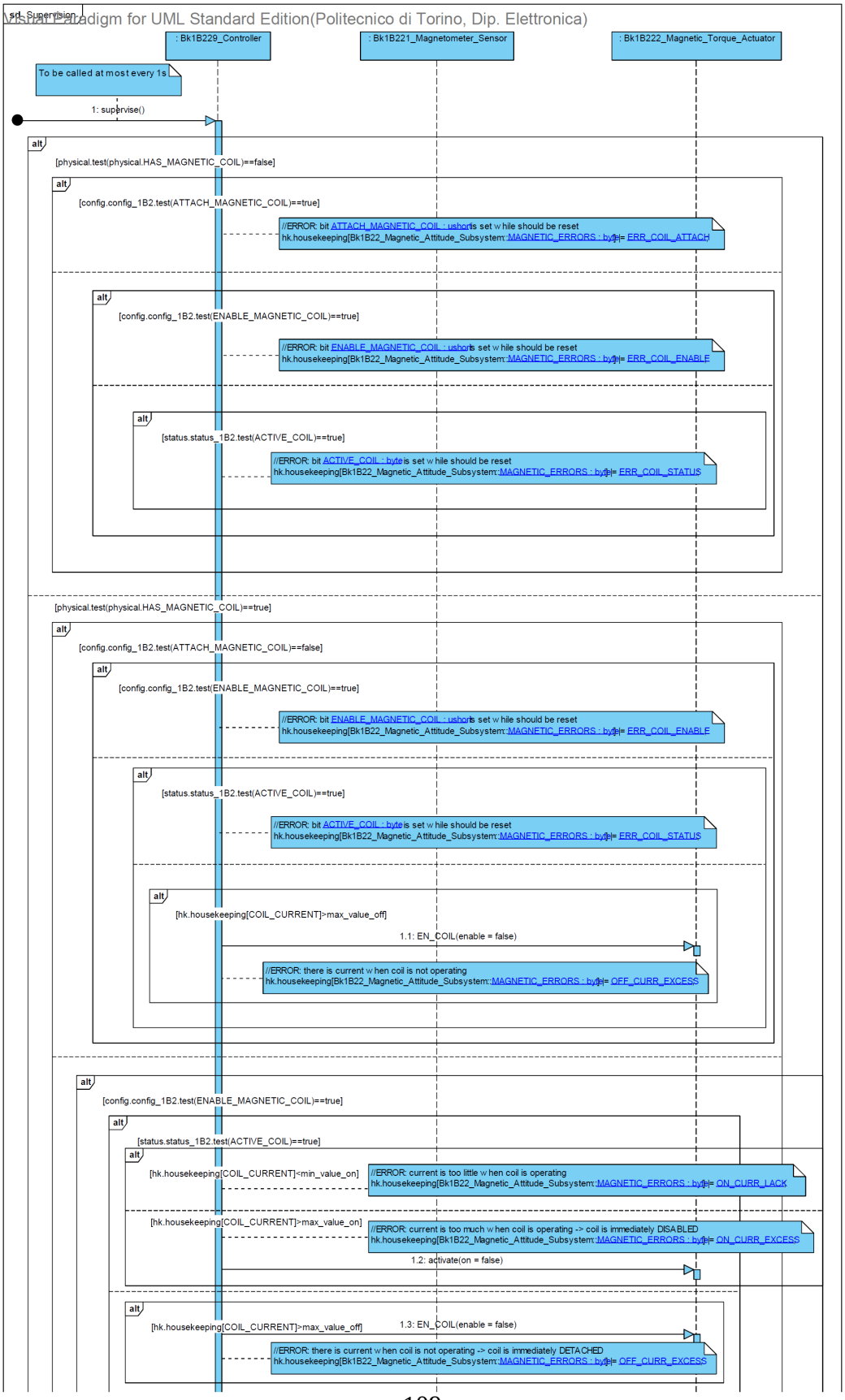
5.1.8. Bk1B22_Controller.reset_pulse()

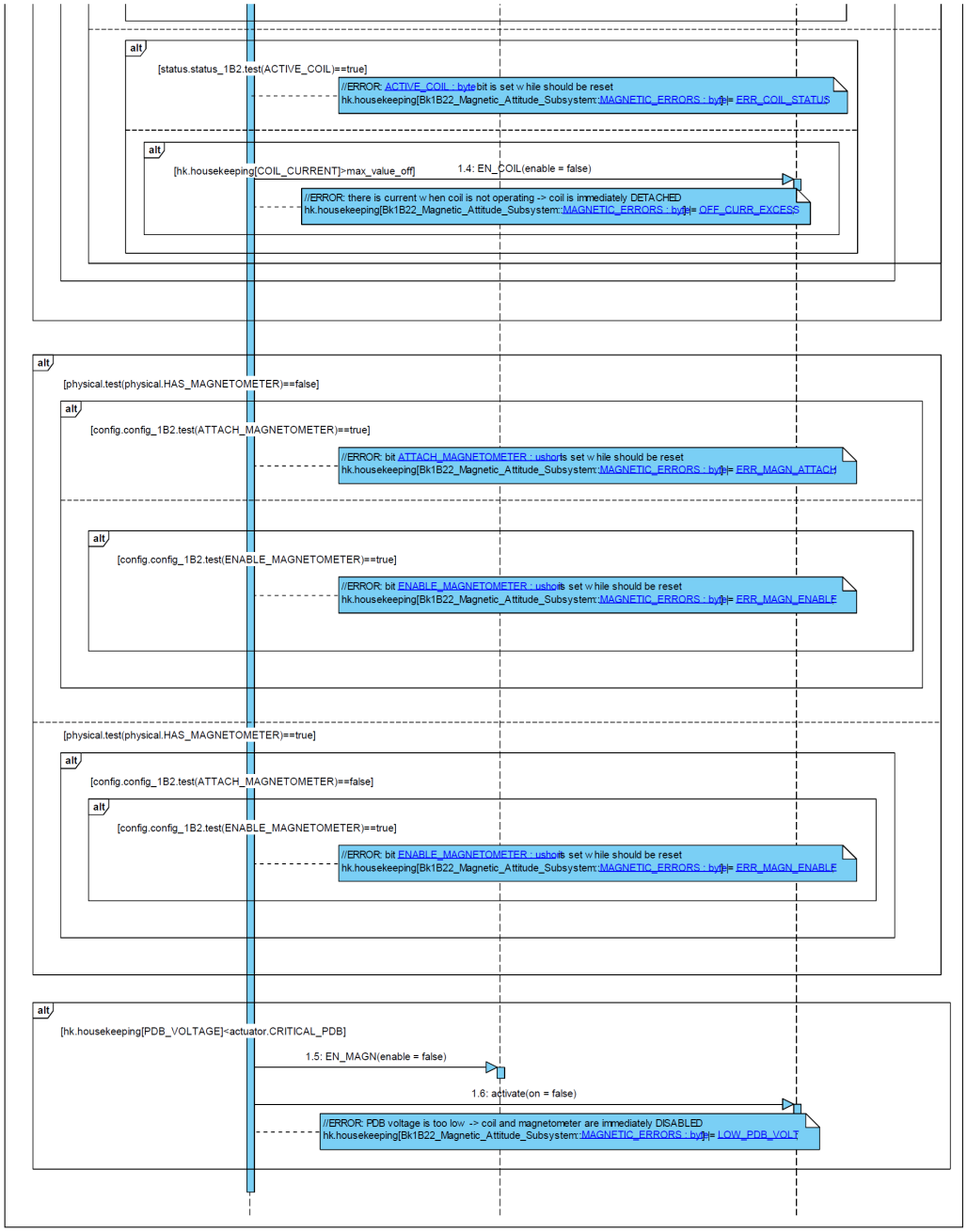
Diagram showing how the system gives a Reset pulse to the Magnetic Field Sensor, in the Set/Reset procedure.



5.1.9. Supervision

This diagram describes how the Supervision use case is realized. It checks whether there are errors in the system and has to be automatically executed at most every 1s, so the *supervise()* operation is called by *housekeeping(index)* operation, when index = SETRESET_MAGN.




alt
[status.status_1B2.test(ACTIVE_COIL)==true]
//ERROR: ACTIVE_COIL : byte bit is set while should be reset
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS : byte] |= ERR_COIL_STATUS
alt
[hk.housekeeping[COIL_CURRENT]>max_value_off]
1.4: EN_COIL(enable = false)
//ERROR: there is current when coil is not operating -> coil is immediately DETACHED
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS : byte] |= OFF_CURR_EXCESS
alt
[physical.test(physical.HAS_MAGNETOMETER)==false]
alt
[config.config_1B2.test(ATTACH_MAGNETOMETER)==true]
//ERROR: bit ATTACH_MAGNETOMETER : ushort is set while should be reset
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS : byte] |= ERR_MAGN_ATTACH
alt
[config.config_1B2.test(ENABLE_MAGNETOMETER)==true]
//ERROR: bit ENABLE_MAGNETOMETER : ushort is set while should be reset
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS : byte] |= ERR_MAGN_ENABLE
[physical.test(physical.HAS_MAGNETOMETER)==true]
alt
[config.config_1B2.test(ATTACH_MAGNETOMETER)==false]
alt
[config.config_1B2.test(ENABLE_MAGNETOMETER)==true]
//ERROR: bit ENABLE_MAGNETOMETER : ushort is set while should be reset
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS : byte] |= ERR_MAGN_ENABLE
alt
[hk.housekeeping[PDB_VOLTAGE]<actuator.CRITICAL_PDB]
1.5: EN_MAGN(enable = false)
1.6: activate(on = false)
//ERROR: PDB voltage is too low -> coil and magnetometer are immediately DISABLED
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS : byte] |= LOW_PDB_VOLT

5.2. Funzioni C++

This paragraph reports C++ functions of 1B22 Magnetic Attitude Subsystem software project, that are implemented from previous sequence diagrams.

Bk1B221_Magnetometer_Sensor.cpp

```
#include "Bk1B221_Magnetometer_Sensor.h"

#include "Magnetometer_2_axis_HMC1002.h"
#include "REF02_5V_Reference.h"
#include "AD623_instrumentation_OPAMP.h"
#include "IOports_1B8.h"

Magnetometer_2_axis_HMC1002 Bk1B221_Magnetometer_Sensor::sensor;
REF02_5V_Reference Bk1B221_Magnetometer_Sensor::reference;
AD623_instrumentation_OPAMP Bk1B221_Magnetometer_Sensor::opamp[2];
IOports_1B8 Bk1B221_Magnetometer_Sensor::IO_ports;

void Bk1B221_Magnetometer_Sensor::MAGN_X() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::MAGN_Y() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::_V3() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::_V() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::REF_3V() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::PDBINT() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::AGND() {
// throw "Not yet implemented";
}
void Bk1B221_Magnetometer_Sensor::GND() {
// throw "Not yet implemented";
}
```

Bk1B221_Magnetometer_Sensor.h

```
#pragma diag_suppress=Pa050
#ifndef __Bk1B221_Magnetometer_Sensor_h__
#define __Bk1B221_Magnetometer_Sensor_h__

#include "platform.h"

#include "Magnetometer_2_axis_HMC1002.h"
#include "REF02_5V_Reference.h"
#include "AD623_instrumentation_OPAMP.h"
#include "IOports_1B8.h"

class Magnetometer_2_axis_HMC1002;
class REF02_5V_Reference;
class AD623_instrumentation_OPAMP;
class IOports_1B8;
class Bk1B221_Magnetometer_Sensor;

class Bk1B221_Magnetometer_Sensor
{
    private: float  const GAIN = 31.303;
    private: float  const OA_OFFSET = 1.5;
    private: static Magnetometer_2_axis_HMC1002 sensor;
    private: static REF02_5V_Reference reference;
    public: float  const MAGN_OFFSET = sen-
sor.V_BIAS_RELATIVE*reference.REF_VOLTAGE*GAIN + OA_OFFSET;
    private: static short  const PDB_MAX = 18;
    private: float  const P_REF_MAX = 4.5e-3;
    private: static AD623_instrumentation_OPAMP opamp[2];
    private: float  const PowerConsumption_Avg_SR = 3.3e-3;
    private: float  const PowerConsumption_Peak_SR = TBD;
    private: float  const PowerConsumption_Avg_Max = (refer-
ence.REF_VOLTAGE*reference.REF_VOLTAGE*2/sensor.R_BRIDGE_MIN) +
(reference.I_SUPPLY_MAX*PDB_MAX) + 2*(opamp[1].I_SUPPLY_MAX*5) + (2*P_REF_MAX) +
PowerConsumption_Avg_SR;
    public: float  const PowerConsumption_Peak = PowerConsumption_Peak_SR + Power-
Consumption_Avg_Max - PowerConsumption_Avg_SR;
    private: float  const Max_supply_current = 0.0128;
    public: float  const SENS_MAGNETIC = (reference.REF_VOLTAGE * sensor.SENS *
GAIN);
    private: static IOports_1B8 IO_ports;

    public: void MAGN_X();
    public: void MAGN_Y();
    public: void notSET(bool not_set) {
            if (not_set)
              *IO_ports.Bk1B221_NOT_SET_PORT |= IO_ports.Bk1B221_NOT_SET_BIT;
            else
              *IO_ports.Bk1B221_NOT_SET_PORT &= ~IO_ports.Bk1B221_NOT_SET_BIT;
            return;
    }
    public: void RESET(bool reset) {
            if (reset)
              *IO_ports.Bk1B221_RESET_PORT |= IO_ports.Bk1B221_RESET_BIT;
            else
              *IO_ports.Bk1B221_RESET_PORT &= ~IO_ports.Bk1B221_RESET_BIT;
            return;
    }
    public: void EN_MAGN(bool enable) {
            if (enable)
              *IO_ports.Bk1B221_EN_MAGN_PORT |= IO_ports.Bk1B221_EN_MAGN_BIT;
            else
              *IO_ports.Bk1B221_EN_MAGN_PORT &= ~IO_ports.Bk1B221_EN_MAGN_BIT;
            return;
    }
    public: void _V3();
    public: void _V();
    public: void REF_3V();
    public: void PDBINT();
    public: void AGND();
    public: void GND();
};

#endif
```

Bk1B222_Magnetic_Torque_Actuator.cpp

```
#include "Bk1B222_Magnetic_Torque_Actuator.h"

#include "Bk1B2222_Coil.h"
#include "IOports_1B8.h"
#include "Bk1B2221_Rsense.h"
#include "Bk1B137A_10x_Differential_Voltage_Sensor.h"
#include "t_Status_1B8.h"
#include "A3953_PWM_Driver.h"

Bk1B2222_Coil Bk1B222_Magnetic_Torque_Actuator::coil;
IOports_1B8 Bk1B222_Magnetic_Torque_Actuator::IO_ports;
Bk1B2221_Rsense Bk1B222_Magnetic_Torque_Actuator::Rsense;
Bk1B137A_10x_Differential_Voltage_Sensor Bk1B222_Magnetic_Torque_Actuator::gain;
t_Status_1B8 Bk1B222_Magnetic_Torque_Actuator::status;
A3953_PWM_Driver Bk1B222_Magnetic_Torque_Actuator::driver;

void Bk1B222_Magnetic_Torque_Actuator::PDBINT() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::_V() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::GND() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::AGND() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::REF_3V() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::SENSE() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::COIL2() {
// throw "Not yet implemented";
}
void Bk1B222_Magnetic_Torque_Actuator::COIL1() {
// throw "Not yet implemented";
}
```

Bk1B222_Magnetic_Torque_Actuator.h

```
#pragma diag_suppress=Pa050
#ifndef __Bk1B222_Magnetic_Torque_Actuator_h__
#define __Bk1B222_Magnetic_Torque_Actuator_h__

#include "platform.h"

#include "Bk1B2222_Coil.h"
#include "IOports_1B8.h"
#include "Bk1B2221_Rsense.h"
#include "Bk1B137A_10x_Differential_Voltage_Sensor.h"
#include "t_Status_1B8.h"
#include "A3953_PWM_Driver.h"

class Bk1B2222_Coil;
class IOports_1B8;
class Bk1B2221_Rsense;
class Bk1B137A_10x_Differential_Voltage_Sensor;
class t_Status_1B8;
class A3953_PWM_Driver;
class Bk1B222_Magnetic_Torque_Actuator;

class Bk1B222_Magnetic_Torque_Actuator
{
   private: static Bk1B2222_Coil coil;
   private: float  const I_SUPPLY_DISABLED = 3.23e-3;
   private: float  const I_SUPPLY_STDBY = 16.34e-3;
   private: float  const I_SUPPLY_SLEEP = 4.94e-3;
   private: float  const I_SUPPLY_ON_max = coil.V_SUPPLY_MAX/coil.COIL_RESISTANCE;
   private: float  const I_SUPPLY_ON_min = coil.V_SUPPLY_MIN/coil.COIL_RESISTANCE;
   private: static IOports_1B8 IO_ports;
   private: static Bk1B2221_Rsense Rsense;
   private: static Bk1B137A_10x_Differential_Voltage_Sensor gain;
   public: float  const SENS_CURRENT = (Rsense.VALUE * gain.GAIN);
   public: float  const COIL_AREA = coil.COIL_AREA;
   public: static t_Status_1B8 status;
   public: float  const Max_value_off = 0.05*I_SUPPLY_ON_max;
   public: float  const Max_value_on = 1.2*I_SUPPLY_ON_max;
```

```
    public: float  const Min_value_on = 0.8*I_SUPPLY_ON_min;
    public: static A3953_PWM_Driver driver;
    public: float  const P_sense_max = Rsense.VALUE*I_SUPPLY_ON_max*I_SUPPLY_ON_max;
    public: float  const P_TOT_max = driver.P_SUPPLY_MAX + P_sense_max + coil.P_MAX;
    public: static ushort  const CRITICAL_PDB = (ushort)(coil.V_SUPPLY_MIN*0.9);

    public: void PDBINT();
    public: void _V();
    public: void GND();
    public: void AGND();
    public: void REF_3V();
    public: void NOT_BRAKE(bool val) {
           if (val)
              *IO_ports.Bk1B222_NOT_BRAKE_PORT |=  IO_ports.Bk1B222_NOT_BRAKE_BIT;
           else
              *IO_ports.Bk1B222_NOT_BRAKE_PORT &= ~IO_ports.Bk1B222_NOT_BRAKE_BIT;
           return;
    }
    public: void MODE(bool val) {
           if (val)
             *IO_ports.Bk1B222_MODE_PORT |= IO_ports.Bk1B222_MODE_BIT;
           else
             *IO_ports.Bk1B222_MODE_PORT &= ~IO_ports.Bk1B222_MODE_BIT;
           return;
    }
    public: void PHASE(bool val) {
           if (val)
             *IO_ports.Bk1B222_PHASE_PORT |= IO_ports.Bk1B222_PHASE_BIT;
           else
             *IO_ports.Bk1B222_PHASE_PORT &= ~IO_ports.Bk1B222_PHASE_BIT;
           return;
    }
    public: void NOT_ENABLE(bool val) {
           if (val)
             *IO_ports.Bk1B222_NOT_ENABLE_PORT |= IO_ports.Bk1B222_NOT_ENABLE_BIT;
           else
             *IO_ports.Bk1B222_NOT_ENABLE_PORT &= ~IO_ports.Bk1B222_NOT_ENABLE_BIT;
           return;
    }
    public: void SENSE();
    public: void EN_COIL(bool enable) {
           if (enable)
             *IO_ports.Bk1B222_EN_COIL_PORT |= IO_ports.Bk1B222_EN_COIL_BIT;
           else
             *IO_ports.Bk1B222_EN_COIL_PORT &= ~IO_ports.Bk1B222_EN_COIL_BIT;
           return;
    }
    public: void COIL2();
    public: void COIL1();
    public: void activate(bool on) {
           if (on) {
             status.status_1B2.set(status.status_1B2.ACTIVE_COIL);
             EN_COIL(true);
             NOT_ENABLE(false);
             NOT_BRAKE(true);
             MODE(true);
           } else {
             NOT_BRAKE(false);
             NOT_ENABLE(true);
             MODE(true);
             EN_COIL(false);
             status.status_1B2.reset(status.status_1B2.ACTIVE_COIL);
           }
           return;
    }
};

#endif
```

Bk1B229_Controller.cpp

```
#include "Bk1B229_Controller.h"

#include "IOports_1B8.h"
#include "t_sensor.h"
#include "t_Configuration_1B8.h"
#include "t_MechanicalConfiguration_1B8.h"
#include "t_Status_1B8.h"
#include "Bk1B222_Magnetic_Torque_Actuator.h"
#include "Bk1B221_Magnetometer_Sensor.h"
#include "Bk1A_1B8_Codes.h"
#include "t_LastError.h"
#include "Slave_Processor.h"
#include "Housekeeping.h"
#include "Buffers.h"
#include "ADC.h"

Bk1B45_Slave::Slave_Processor Bk1B229_Controller::proc;
Bk1B45_Slave::Housekeeping Bk1B229_Controller::hk;
IOports_1B8 Bk1B229_Controller::IO_ports;
t_sensor Bk1B229_Controller::actual_L;
t_Configuration_1B8 Bk1B229_Controller::config;
t_MechanicalConfiguration_1B8 Bk1B229_Controller::physical;
t_Status_1B8 Bk1B229_Controller::status;
Common::ADC Bk1B229_Controller::adc;
Bk1B222_Magnetic_Torque_Actuator Bk1B229_Controller::actuator;
Bk1B45_Slave::Buffers Bk1B229_Controller::buffers;
Bk1B221_Magnetometer_Sensor Bk1B229_Controller::sensor;
short Bk1B229_Controller::SR_internal_i;
bool Bk1B229_Controller::SET_FLAG;
bool Bk1B229_Controller::SET_ENABLE;
bool Bk1B229_Controller::RESET_FLAG;
bool Bk1B229_Controller::TEMP_COIL_OFF;
t_sensor Bk1B229_Controller::SRoffset_Xset;
t_sensor Bk1B229_Controller::SRoffset_Xreset;
t_sensor Bk1B229_Controller::SRoffset_Yset;
t_sensor Bk1B229_Controller::SRoffset_Yreset;

void Bk1B229_Controller::housekeeping(ushort index) {
          switch (index) {
          bool tmp;
          case Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X-2:
            if (status.status_1B2.test(status.status_1B2.ACTIVE_COIL)==true)  {
//disable coil 5+tau before measuring to get correct magnetic field
                  TEMP_COIL_OFF=true;
                  actuator.activate(false);
            }
            if (SET_ENABLE==true)
                  SET_FLAG=true;
            if(config.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER))
                  sensor.EN_MAGN(true);
            break;
          case Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X:
            if (SET_FLAG==true)  {
                  if (RESET_FLAG==false)  {
                        set_pulse();

adc.acquire(Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_ADC, con-
fig.ADC_SAMPLE_TIME, SRoffset_Xset);
                  }  else  {

adc.acquire(Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_ADC, con-
fig.ADC_SAMPLE_TIME, SRoffset_Xreset);
                  }
            }  else  {

adc.acquire_scale(Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_ADC, con-
fig.ADC_SAMPLE_TIME,
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X],
hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_SCALE]);
                  }
            break;
          case Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y:
            if (SET_FLAG==true)  {
                  if (RESET_FLAG==false)  {

adc.acquire(Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_ADC, con-
fig.ADC_SAMPLE_TIME, SRoffset_Yset);
                  }  else  {

adc.acquire(Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_ADC, con-
fig.ADC_SAMPLE_TIME, SRoffset_Yreset);
                  }
            }  else  {
```

```
adc.acquire_scale(Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_ADC, con-
fig.ADC_SAMPLE_TIME,
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y],
hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_SCALE]);
                }
              break;
            case (Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y+1):
              if (SET_FLAG==true)  {
                   if (RESET_FLAG==false)  {
                        reset_pulse();
                   }  else  {
//now offset voltage is ready to be used

hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_SCALE].offset=(SRof
fset_Xset + SRoffset_Xreset)/2;

hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_SCALE].offset=(SRof
fset_Yset + SRoffset_Yreset)/2;
                        SET_FLAG=false;
                        RESET_FLAG=false;
                        SET_ENABLE=false;
                        sensor.EN_MAGN(false);
                   }
              }  else  {
                   if (RESET_FLAG==false)  {
                        sensor.EN_MAGN(false);
                   }
                }
              break;
            case (Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y+2):
              if (TEMP_COIL_OFF==true)  {                        //reactivate coil if
it has been temporarily disabled
                   actuator.activate(true);
                   TEMP_COIL_OFF=false;
              }
              break;
            case Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT:
              adc.acquire_scale(Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT_ADC,
config.ADC_SAMPLE_TIME,
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT],
hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT_SCALE]);
              break;
            case Bk1B22_Magnetic_Attitude_Subsystem::EOC_INTEGRAL:
              if (actual_L>0) {
                tmp = integrate();
                if(!tmp)
                   actuator.activate(false);
              } else {
                actuator.activate(false);
              }
              break;
            case Bk1B22_Magnetic_Attitude_Subsystem::SETRESET_MAGN:
              if
(SR_internal_i>=Bk1B22_Magnetic_Attitude_Subsystem::SETRESET_TIME_RATIO) {
                if (config.config_1B2.test(config.config_1B2.ATTACH_MAGNETOMETER)) {
                  if (config.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER))
//else it's impossible to start Set-Reset sequence
                     setreset();
                }  SR_internal_i=0;
              } else {
                SR_internal_i+=1;
                }
              supervise();
              break;
            }
}

void Bk1B229_Controller::interpret(Bk1A_1B8_Codes::t_Commands_1B8 command,
t_LastError error) {
            switch (command) {
            case Bk1A1_Common_Codes::CMD_SET_CONFIGURATION:
              if (physical.test(physical.HAS_MAGNETIC_COIL)==false)  {
//coil
                   actuator.activate(false);
                   }
              else if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETIC_COIL)==false)  {
                              actuator.activate(false);
                              }
                    else if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETIC_COIL)==false)  {
                                        actuator.activate(false);
                                        }
              if (physical.test(physical.HAS_MAGNETOMETER)==false)  {
//magnetometer
                   sensor.EN_MAGN(false);
                   }
```

```
            else if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETOMETER)==false)  {
                                sensor.EN_MAGN(false);
                                }
                        else if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER)==false)  {
                                        sensor.EN_MAGN(false);
                                        }
              break;
            case Bk1A1_Common_Codes::CMD_RESET_CONFIGURATION:
              if (physical.test(physical.HAS_MAGNETIC_COIL)==false)  {
//coil
                   actuator.activate(false);
                  }
              else if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETIC_COIL)==false)  {
                                actuator.activate(false);
                                }
                        else if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETIC_COIL)==false)  {
                                        actuator.activate(false);
                                        }
              if (physical.test(physical.HAS_MAGNETOMETER)==false)  {
//magnetometer
                    sensor.EN_MAGN(false);
                   }
              else if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETOMETER)==false)  {
                                sensor.EN_MAGN(false);
                                }
                        else if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER)==false)  {
                                        sensor.EN_MAGN(false);
                                        }
              break;
            case Bk1A_1B8_Codes::CMD_SETRESET_MAGNETOMETER:
              if (config.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER))
                 setreset();
              break;
            case Bk1A_1B8_Codes::CMD_ACTUATE_COIL:
              if ((con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETIC_COIL)==true)&&(config.config_
1B2.test(config.config_1B2.ENABLE_MAGNETOMETER)==true))  {
                        byte* temp;
                        ushort length;
                        temp=buffers.get((Bk1A1_Common_Codes::t_Commands)command,
length);
                        if (temp!=NULL)  {
                                actual_L=*((t_sensor*)temp);
                                buffers.release(temp);
                                }
                        if (actual_L>=0)  {
                                actuator.PHASE(true);
                                }
                        else  {
                                actuator.PHASE(false);
                                actual_L=-actual_L;
                                }
                        if (actual_L==0)  {
                                actuator.activate(false);
                                }
                        else  {
                                actuator.activate(true);
                                }
              }
              break;
            }
}

bool Bk1B229_Controller::integrate() {
            t_sensor Bx, By, II;
            long B;

            Bx = hk.housekeeping[Bk1A_1B8_Codes::MAGNETIC_FIELD_X];     //magnetic
field components
            By = hk.housekeeping[Bk1A_1B8_Codes::MAGNETIC_FIELD_Y];

            if((Bx<5)&&(By<5)&&(Bx>(-5))&&(By>(-5)))
                return false;

            II = hk.housekeeping[Bk1A_1B8_Codes::COIL_CURRENT];    //coil current

            B = sqrt((float)((((long)Bx)*Bx) + (((long)By)*By)));     //magnetic
field module

            float const K = (actuator.COIL_AREA *
Bk1B22_Magnetic_Attitude_Subsystem::SENS_MAGNETIC_FIELD *
```

```
Bk1B22_Magnetic_Attitude_Subsystem::SENS_COIL_CURRENT * config.SENSOR_SAMPLE_TIME)
/ Bk1B22_Magnetic_Attitude_Subsystem::SENS_MOMENTUM / 1000;

//printf ("%ld %d %g \n",B, II, K);    //used for test session
//D = Coil Magnetic Dipole = n*S*I; n = number of coil windings ; S = coil area
//SENSOR_SAMPLE_TIME = end_of_count*1 ms, assuming that TIMER_FREQ of 1B45 is 1 kHz

// actual_L is the new value of integrated |B|*D, which is saved into 'actual_L'
attribute of 'Bk1B22_Controller' class

          actual_L -= B * II * K;

          return true;
}

void Bk1B229_Controller::supervise() {
          short Norm_Curr =
hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT];
          //reporting current critical values in units of SENS_COIL_CURRENT
          short const MAX_value_off_int = actua-
tor.Max_value_off/Bk1B22_Magnetic_Attitude_Subsystem::SENS_COIL_CURRENT;
          short const MIN_value_on_int = actua-
tor.Min_value_on/Bk1B22_Magnetic_Attitude_Subsystem::SENS_COIL_CURRENT;
          short const MAX_value_on_int = actua-
tor.Max_value_on/Bk1B22_Magnetic_Attitude_Subsystem::SENS_COIL_CURRENT;

          if (physical.test(physical.HAS_MAGNETIC_COIL)==false)  {
//magnetic actuator is not present
               if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETIC_COIL)==true)  {
//error
                    config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_COIL_ATTACH;
               }  else  {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_COIL_ATTACH;
                    if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETIC_COIL)==true)  {
//error
                         con-
fig.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_COIL_ENABLE;
                    }  else  {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_COIL_ENABLE;
                         if
(status.status_1B2.test(status.status_1B2.ACTIVE_COIL)==true)  {
//error

status.status_1B2.reset(status.status_1B2.ACTIVE_COIL);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_COIL_STATUS;
                         }  else  {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_COIL_STATUS;
                              }
                    }
               }
          }  else  {
               if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETIC_COIL)==false)  {
//magnetic actuator is present but not attached
                    if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETIC_COIL)==true)  {
//error
                         con-
fig.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_COIL_ENABLE;
                    }  else  {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_COIL_ENABLE;
                         if
```

```
(status.status_1B2.test(status.status_1B2.ACTIVE_COIL)==true)  {
//error

status.status_1B2.reset(status.status_1B2.ACTIVE_COIL);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_COIL_STATUS;
                        }  else  {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_COIL_STATUS;
                                if (Norm_Curr>MAX_value_off_int)  {
//current when detached
                                    actuator.EN_COIL(false);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
OFF_CURR_EXCESS;
                                }  else  {

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~OFF_CURR_EXCESS;
                                }
                            }
                        }
                }  else  {
                        if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETIC_COIL)==false)  {
//magnetic actuator is present and attached but not enabled
                            if
(status.status_1B2.test(status.status_1B2.ACTIVE_COIL)==true)  {
//error

status.status_1B2.reset(status.status_1B2.ACTIVE_COIL);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_COIL_STATUS;
                            }  else  {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_COIL_STATUS;
                                if (Norm_Curr>MAX_value_off_int)  {
//current when disabled
                                    actuator.EN_COIL(false);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
OFF_CURR_EXCESS;
                                    con-
fig.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
                                }  else  {

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~OFF_CURR_EXCESS;
                                }
                            }
                        }  else  {
//magnetic actuator is present, attached and enabled
                            if
(status.status_1B2.test(status.status_1B2.ACTIVE_COIL)==true)  {
//coil is operating
                                if (Norm_Curr>MAX_value_on_int)  {
//current excess
                                    actuator.activate(false);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ON_CURR_EXCESS;
                                    con-
fig.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
                                }  else  {

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ON_CURR_EXCESS;
                                    if (Norm_Curr<MIN_value_on_int)  {
//too little current

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ON_CURR_LACK;
                                    }  else  {

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ON_CURR_LACK;
                                    }
                                }
                            }  else  {
//coil is off
                if (Norm_Curr>MAX_value_off_int)  {
```

```
//current when coil is off
                                                actuator.EN_COIL(false);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
OFF_CURR_EXCESS;
                                                        con-
fig.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
                                                        con-
fig.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
                                                } else {

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~OFF_CURR_EXCESS;
                                                        }
                                                }
                                        }
                                }
                        }

                if (physical.test(physical.HAS_MAGNETOMETER)==false) {
//magnetic sensor is not present
                        if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETOMETER)==true) {
//error
                                config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_MAGN_ATTACH;
                        } else {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_MAGN_ATTACH;
                                if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER)==true) {
//error
                                        con-
fig.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_MAGN_ENABLE;
                                } else {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_MAGN_ENABLE;
                                        }
                                }
                        } else {
                                if (con-
fig.config_1B2.test(config.config_1B2.ATTACH_MAGNETOMETER)==false) {
//magnetic sensor is present but not attached
                                        if (con-
fig.config_1B2.test(config.config_1B2.ENABLE_MAGNETOMETER)==true) {
//error
                                                con-
fig.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] |=
ERR_MAGN_ENABLE;
                                        } else {
//correct

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~ERR_MAGN_ENABLE;
                                                }
                                        }
                                }

   if(hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::PDB_VOLTAGE]<actuator.CRI
TICAL_PDB) {            //too little bus voltage
                        actuator.activate(false);
                        sensor.EN_MAGN(false);
                        config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
                        config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);
                        hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS]
|= LOW_PDB_VOLT;
                } else {

hk.housekeeping[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_ERRORS] &=
~LOW_PDB_VOLT;
                        }
}

void Bk1B229_Controller::boot() {
                sensor.EN_MAGN(false);          //disabling all blocks
                actuator.activate(false);
```

```
          sensor.notSET(true);
          sensor.RESET(true);

          config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
//resetting bits in configuration word
          config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);

          t_scaling calibration[3];
          t_sensor const DIGITAL_OFFSET = sensor.MAGN_OFFSET * adc.SENS_ADC;
//converting global magnetometer block offset into digital format
          calibration[0] =
hk.getSensorCalibration(IO_ports.Bk1B22_CALIBRATION_PORT,
IO_ports.Bk1B22_CALIBRATION_BIT, 0);
          calibration[1] =
hk.getSensorCalibration(IO_ports.Bk1B22_CALIBRATION_PORT,
IO_ports.Bk1B22_CALIBRATION_BIT, 1);
          calibration[2] =
hk.getSensorCalibration(IO_ports.Bk1B22_CALIBRATION_PORT,
IO_ports.Bk1B22_CALIBRATION_BIT, 2);
          long tmp_gain1 =
(Bk1B22_Magnetic_Attitude_Subsystem::SENS_MAGNETIC_FIELD_RAW/Bk1B22_Magnetic_Attitu
de_Subsystem::SENS_MAGNETIC_FIELD)*t_scaling::SCALING_FACTOR;
          long tmp_gain2 =
(Bk1B22_Magnetic_Attitude_Subsystem::SENS_COIL_CURRENT_RAW/Bk1B22_Magnetic_Attitude
_Subsystem::SENS_COIL_CURRENT)*t_scaling::SCALING_FACTOR;

   //initializing calibration for magnetic field X

   hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_SCALE].offset =
calibration[0].offset + DIGITAL_OFFSET;

   hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_X_SCALE].gain =
((long)calibration[0].gain*tmp_gain1)/t_scaling::SCALING_FACTOR;

   //initializing calibration for magnetic field Y

   hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_SCALE].offset =
calibration[1].offset + DIGITAL_OFFSET;

   hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::MAGNETIC_FIELD_Y_SCALE].gain =
((long)calibration[1].gain*tmp_gain1)/t_scaling::SCALING_FACTOR;

   //initializing calibration for coil current
          hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT_SCALE].offset
= calibration[2].offset;
          hk.scaling[Bk1B22_Magnetic_Attitude_Subsystem::COIL_CURRENT_SCALE].gain =
((long)calibration[2].gain*tmp_gain2)/t_scaling::SCALING_FACTOR;

          actual_L = 0;       //resetting all flags, index and variables
          SET_ENABLE = false;
          SET_FLAG = false;
          RESET_FLAG = false;
          TEMP_COIL_OFF = false;
          SRoffset_Xset = 0;
          SRoffset_Yset = 0;
          SRoffset_Xreset = 0;
          SRoffset_Yreset = 0;
          SR_internal_i = 0;
}
```

Bk1B229_Controller.h

```
#pragma diag_suppress=Pa050
#include <math.h>
#include "Bk1B22_Magnetic_Attitude_Subsystem.h"
#include "Bk1B22_Errors.h"

#ifndef __Bk1B229_Controller_h__
#define __Bk1B229_Controller_h__

#include "platform.h"

#include "IOports_1B8.h"
#include "t_sensor.h"
#include "t_Configuration_1B8.h"
#include "t_MechanicalConfiguration_1B8.h"
#include "t_Status_1B8.h"
#include "Bk1B222_Magnetic_Torque_Actuator.h"
#include "Bk1B221_Magnetometer_Sensor.h"
#include "Bk1A_1B8_Codes.h"
#include "t_LastError.h"
#include "Slave_Processor.h"
#include "Housekeeping.h"
#include "Buffers.h"
#include "ADC.h"

class IOports_1B8;
// typedef t_sensor;
class t_Configuration_1B8;
class t_MechanicalConfiguration_1B8;
class t_Status_1B8;
class Bk1B222_Magnetic_Torque_Actuator;
class Bk1B221_Magnetometer_Sensor;
class Bk1A_1B8_Codes;
// enum t_LastError;
class Bk1B229_Controller;
namespace Bk1B45_Slave
{
   class Slave_Processor;
   class Housekeeping;
   class Buffers;
}
namespace Common
{
   class ADC;
}

class Bk1B229_Controller
{
   private: static Bk1B45_Slave::Slave_Processor proc;
   public: static Bk1B45_Slave::Housekeeping hk;
   private: static IOports_1B8 IO_ports;
   public: static t_sensor actual_L;
   private: static t_Configuration_1B8 config;
   private: static t_MechanicalConfiguration_1B8 physical;
   private: static t_Status_1B8 status;
   private: static Common::ADC adc;
   private: static Bk1B222_Magnetic_Torque_Actuator actuator;
   public: static Bk1B45_Slave::Buffers buffers;
   private: static Bk1B221_Magnetometer_Sensor sensor;
   private: static short SR_internal_i;
   private: float  const MSP_CLOCK_PERIOD = 1000000.0/proc.CLOCK_FREQ;
   private: static bool SET_FLAG;
   private: static bool SET_ENABLE;
   private: static bool RESET_FLAG;
   private: static bool TEMP_COIL_OFF;
   private: static t_sensor SRoffset_Xset;
   private: static t_sensor SRoffset_Xreset;
   private: static t_sensor SRoffset_Yset;
   private: static t_sensor SRoffset_Yreset;

   public: void housekeeping(ushort index);
   public: void interpret(Bk1A_1B8_Codes::t_Commands_1B8 command, t_LastError er-
ror);
   public: bool integrate();
   public: static void setreset() {
         SET_ENABLE = true;
   }
   public: static void set_pulse() {
         volatile int i;
         static const int t0=5, t1=1, t2=5, t3=20;
         //nx sono i numeri di ripetizioni dei vari cicli for
         static const int n0=(int)(t0/(10*MSP_CLOCK_PERIOD)); //siccome una singla
ripetizione di ciclo for occupa 10 colpi di clock del processore
         static const int n1=(int)(t1/(10*MSP_CLOCK_PERIOD));
         static const int n2=(int)(t2/(10*MSP_CLOCK_PERIOD));
```

```
            static const int n3=(int)(t3/(10*MSP_CLOCK_PERIOD));
            //inizializzazione a 1 logico di entrambi i segnali
            sensor.notSET(true);
            sensor.RESET(true);
            for(i=0;i<=n0;i++) ;          //ciclo for a vuoto per garantire t0
            sensor.RESET(false);
            for(i=0;i<=n1;i++) ;          //ciclo for a vuoto per garantire t1
            sensor.notSET(false);         //impulso di set
            for(i=0;i<=n2;i++) ;          //ciclo for a vuoto per garantire t2
            sensor.notSET(true);
            for(i=0;i<=n3;i++) ;          //ciclo for a vuoto per garantire t3, che
potrebbe anche essere garantito dai tempi dell'ADC
    }
    public: static void reset_pulse() {
            sensor.RESET(true);
            RESET_FLAG=true;
    }
    public: void supervise();
    public: void boot();
};

#endif
```

platform.h

```
#pragma diag_suppress=Pe382
#ifndef __MSP430FG439__
#define __MSP430FG439__
#endif
#define MSP430 defined (__MSP430F149__) || defined (__MSP430FG439__) || defined
(__MSP430F5436__)
#define CHIPCON false
#define def_SPI_MODE true
#define DEBUG

#ifndef __IAR_SYSTEMS_ICC__
#ifndef __intrinsic
#define __intrinsic
#endif
#ifndef __interrupt
#define __interrupt
#endif
#define __TID__ (0x2B<<8)
#include <in430.h>
#pragma language=extended
#define DEFC(name, address) static volatile unsigned char name;
#define DEFW(name, address) static volatile unsigned short name;
#define DEFXC  static volatile unsigned char
#define DEFXW  static volatile unsigned short
#endif  /* __IAR_SYSTEMS_ICC__  */

#include <intrinsics.h>
#include <msp430.h>
#include <math.h>
#include <stdlib.h>

//#define DEBUG
//#define DEBUG_MSG
//#define DEBUG_VAL

#ifdef DEBUG
#include <stdio.h>
#endif

#ifdef DEBUG_MSG
#define DEBUGMSG(x) x
#else
#define DEBUGMSG(x)
#endif

#ifdef DEBUG_VAL
#define DEBUGVAL(x) x
#else
#define DEBUGVAL(x)
#endif

#define pi 3.14159265358979

#define byte unsigned char
#define ushort unsigned short
#define ulong unsigned long
#define uint unsigned int
#define __abstract
```

```
#define volatile_byte volatile byte

#define def_CMD_WRITE_DATA
#define def_CMD_SET_CONFIGURATION

#define def_CMD_READ_DATA
#define def_CMD_GET_HOUSEKEEPING
//#define def_CMD_GET_HISTORY
#define def_CMD_GET_STATISTICS
#define def_CMD_GET_CONFIGURATION

#define def_CMD_STANDBY
#define def_CMD_WAKEUP
//#define def_CMD_RESET_STATISTICS
#define def_CMD_COMMAND
#define def_CMD_CALIBRATE_HOUSEKEEPING

#define ptrNULL           0       /* 0L if pointer same as long */

#define TBD 50
#define MAXMOD 4

//#define __DEBUG

//typedef short t_sensor;

namespace std{};
#define __DEBUG

namespace std{};
```

Capitolo 6

6. Progetto hardware del controllo di assetto magnetico

In questo capitolo verranno illustrati i componenti scelti per la realizzazione dell'hardware del progetto, rappresentati nei diagrammi degli oggetti del capitolo 4, con le relative specifiche. Ora non ci si soffermerà, pertanto, sulla pura elencazione delle medesime, ma verranno motivati, per ognuno dei due sottoblocchi principali:

- La scelta dei componenti;
- Il dimensionamento dei circuiti di condizionamento per le grandezze misurate;
- La necessità di una compensazione software dell'errore di misura (si veda cap.5) e l'entità dell'errore stesso.

In seguito verranno descritti i valori di consumo e mostrati gli schemi elettrici dei circuiti facenti parte del progetto. Esso è, nel suo complesso, rappresentabile tramite uno schema gerarchico (par. 6.4), all'interno del quale sono distinguibili due blocchi distinti, ideati seguendo il principio di modularità che sta alla base di AraMiS stesso. Ogni parte del sistema di controllo d'assetto viene resa esportabile, modificabile o sostituibile, per eventuali progetti futuri, senza andare a modificare l'intera struttura del sistema.
I blocchi costituenti lo schema globale del controllo di assetto magnetico sono:

- **Magnetometro**: include i circuiti di misurazione e condizionamento del campo magnetico (par. 6.1)
- **Solenoide**: include il driver del solenoide e il circuito di condizionamento per la misurazione della corrente nel solenoide (par. 6.2)
- **Memoria one-wire** per la conservazione dei dati di calibrazione

Insieme a questi, vanno considerati anche:

- **Microcontrollore**: viene descritto il microcontrollore utilizzato (par. 6.3)
- **Alimentazioni**: tensioni utilizzate in tutti gli schemi elettrici del progetto, sono:
  - **PDBINT**, tensione del Power Distribution Bus, ricavata dalla tensione delle batterie e variabile da 14 V a 18 V
  - **VCC_CPU**, di valore 3.3 V, ricavata dalla PDBINT (il modello di regolatore di tensione utilizzato non è ancora stato scelto in via definitiva)
  - **5V**, ricavata dalla PDBINT (il modello di regolatore di tensione utilizzato non è ancora stato scelto in via definitiva)
- **Riferimento REF_3V**: tensione di riferimento di valore 3 V, ricavata dall'alimentazione a 5 V (il modello del riferimento di tensione utilizzato non è ancora stato scelto in via definitiva)

Soltanto i primi due blocchi, ovvero il magnetometro e l'attuatore, costituendo le due parti principali del progetto, sono fisicamente presenti nel modulo 1B22, e, pertanto, come già detto, verranno descritti in maniera più dettagliata, mentre le alimentazioni, le tensioni di riferimento e il microcontrollore sono comuni a tutta la Power Management Tile. Quest'ultimo verrà illustrato separatamente, seppure in modo sintetico, poiché concettualmente necessario, nonché indispensabile al funzionamento del sistema.
I segnali provenienti dal microcontrollore compaiono nello schema gerarchico *1B22_Magnetic_Attitude_Subsystem* quali segnali di ingresso ed uscita, insieme alle tensioni di alimentazione PDBINT, 5V, VCC_CPU e alla tensione di riferimento REF_3V.

6.1. Magnetometro

Come abbiamo visto nei capitoli precedenti, per applicare un determinato momento angolare alla tile, occorre misurare, e condizionare opportunamente, i valori di campo magnetico nelle componenti x ed y e la corrente che fluisce nel solenoide. Di quest'ultima ci si occuperà nel prossimo sottocapitolo.
B_x e B_y, invece, contribuiscono, insieme alla sola componente di dipolo magnetico generato dal solenoide, D_z, a fornire, in ogni istante, la coppia:

$$\vec{C} = \vec{D} \wedge \vec{B} = D_z B_y \hat{x} + D_z B_x \hat{y}$$

Come già precedentemente enunciato, il sistema dovrà acquisire i valori di B_x e B_y, che, successivamente, dovranno essere condizionati in modo da adattarsi alla dinamica dell'ADC (0~2.5V), assumendo che essi siano compresi tra -0.625 G e 0.625 G.
Tali valori verranno infine utilizzati, ad ogni istante di integrazione $i \cdot \Delta t$, per calcolare il contributo $L_i = N \cdot S \cdot \left|\vec{B}\right| \cdot I \cdot \Delta t$.

6.1.1. Sensore di campo magnetico biassiale

Il sensore di campo magnetico dovrà, quindi, essere in grado di effettuare misure su due assi in un range maggiore o uguale a [-0.625 G ~ 0.625 G] e, possibilmente, avere un basso consumo di potenza ed una sensibilità elevata.
Si è scelto di utilizzare il sensore HMC1002 della Honeywell, le cui caratteristiche sono le seguenti:

- Due ponti di Wheatstone magnetoresistivi per la misura biassiale del campo magnetico
- Tensione di alimentazione tipica dei ponti: 5 V
- Campo misurabile, in Gauss: -2~2
- Offset del ponte compreso tra -60 mV e 30 mV con alimentazione a 8 V
- Sensibilità tipica 3.2 mV/V/G
- Impulsi di corrente di Set/Reset compresi tra 3 A e 5 A
- Risoluzione di 27 μG con alimentazione a 5 V
- Package 20-pin SOIC
- Uscite differenziali

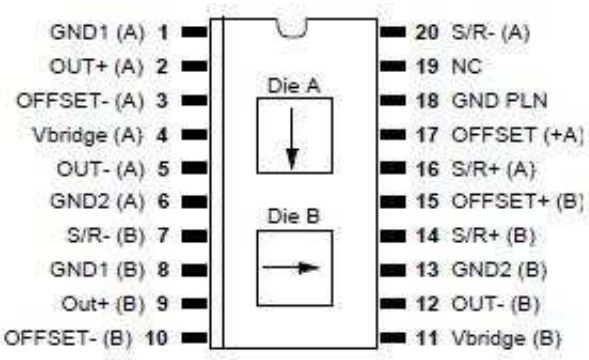


Figura 6.1 Piedinatura del sensore HMC1002

Come si può vedere dalla piedinatura del circuito in figura 6.1, le tensioni differenziali di uscita sono:

$$V_A = OUT_+(A) - OUT_-(A)$$
$$V_B = OUT_+(B) - OUT_-(B)$$

Dal momento che i due ponti di Wheatstone per la misura del campo magnetico sui due assi hanno identiche caratteristiche, d'ora in poi si tratterà esclusivamente V_A, tenendo presente che le formule esposte saranno valide anche per V_B.
Indicando con k_M la sensibilità magnetica di ogni ponte espressa in mV/G, con V_{off} la tensione di offset del ponte in mV e con B_x la componente di campo magnetico misurata dal ponte A ed espressa in G, si avrà che:

$$V_A = k_M \cdot B_x + V_{off}$$

6.1.2. Condizionamento

Il circuito di condizionamento dovrà allo stesso tempo adattare le uscite in tensione del sensore alla dinamica di ingresso del convertitore ADC e disaccoppiarle dal resto del circuito, di conseguenza verrà utilizzato un amplificatore operazionale opportunamente retroazionato, e, sapendo che tali uscite sono differenziali e che il sensore garantisce un'elevata sensibilità, verrà utilizzato un operazionale di tipo differenziale da strumentazione.
La configurazione scelta per il circuito di condizionamento è visibile nella figura sottostante.

Per riuscire ad acquisire il segnale tramite l'ADC del microcontrollore bisogna far sì che al massimo valore possibile V_{A_MAX} in uscita dal sensore corrisponda, in uscita dal condizionamento, il massimo valore leggibile dagli ingressi dell'ADC (2.5 V), mentre al valore minimo V_{A_MIN} corrispondano 0 V.
Tali valori massimo e minimo di V_A coincidono, rispettivamente, con le letture di campo magnetico massimo di 0.625 G e minimo di -0.625 G, ma occorre tenere presente che anche i parametri k_M e V_{off} del sensore possono variare in base al processo di costruzione, alle condizioni ambientali, alle tensioni e correnti di alimentazione, o altro…, pertanto si considereranno sempre i valori minimi, massimi e tipici, nel seguente modo:

$$V_{A\_MAX} = k_{M\_MAX} \cdot B_{x\_MAX} + V_{off\_MAX}$$
$$V_{A\_MIN} = k_{M\_MAX} \cdot B_{x\_MIN} + V_{off\_MIN}$$
$$V_{A\_TYP} = k_{M\_TYP} \cdot B_x + V_{off\_TYP}$$

(6)

Sul datasheet del sensore HMC1002 [15] troviamo i seguenti valori per la tensione di offset del ponte resistivo con una tensione di alimentazione di 8 V:

V_{off_MIN} [V_{al}=8V] = -60 mV
V_{off_TYP} [V_{al}=8V] = -15 mV
V_{off_MAX} [V_{al}=8V] = 30 mV

poiché la tensione di alimentazione scelta per il sensore è di 5 V, i valori relativi di offset si ricavano mediante proporzione e valgono:

V_{off_MIN} [V_{al}=5V] = -37.5 mV
V_{off_TYP} [V_{al}=5V] = -9.37 mV
V_{off_MAX} [V_{al}=5V] = 18.75 mV

Mentre per la sensibilità vengono riportati i valori relativi (dipendenti da V_{al}), che con un'alimentazione di 5 V diventano:

k_{M_MIN} = 12.5 mV/G
k_{M_TYP} = 16 mV/G
k_{M_MAX} = 20 mV/G

Sostituendo nella (6) possiamo ricavare la massima dinamica di V_A:

$$V_{A\_MAX} = 31.25mV$$
$$V_{A\_MIN} = -50mV$$

L'amplificatore scelto per condizionare ciascuna delle due uscite differenziali è l'AD623 della Analog Devices, di cui si elencano qui sotto le caratteristiche salienti:

- Single/Dual supply
- Facile da utilizzare
- Tensione di alimentazione tipica di 5 V
- Reazione interna
- Guadagno regolabile mediante resistenza esterna da 1 (c.a.) a 1000
- Basso consumo (massima corrente di alimentazione pari a 575 μA)
- Uscita rail to rail
- Elevata accuratezza
- Elevato CMRR (> 100dB)
- Basso costo
- Presenza di ingresso REF per traslare la tensione di uscita
- Package SO-8

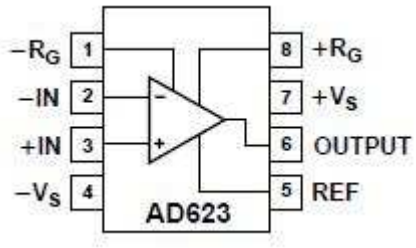


Figura 6.2 Piedinatura AD623

In uscita dall'operazionale si avrà quindi:

$$MAGN\_X = V_A \cdot A_D + V_{REF}$$

e, dal momento che il circuito di condizionamento è replicato anche per l'uscita del sensore corrispondente a B_y,

$$MAGN\_Y = V_B \cdot A_D + V_{REF}$$

dove V_{REF} è la tensione di offset applicata in ingresso all'AD623 e A_D il suo guadagno differenziale configurabile. Per ottenere i valori desiderati di guadagno ed offset imponiamo le seguenti relazioni:

$$V_A = V_{A\_MIN} \Rightarrow MAGN\_X = 0V$$
$$V_A = V_{A\_MAX} \Rightarrow MAGN\_X = 2.5V$$

Infine dal sistema

$$A_D \cdot (-50mV) + V_{REF} = 0V$$
$$A_D \cdot (-50mV) + V_{REF} = 2.5V$$

si ricava

$$A_D = 30.769$$
$$V_{REF} = 1.538V$$

Dai datasheet dell'AD623 [16] si ottiene la seguente formula per il calcolo del guadagno in funzione della resistenza esterna:

$$A_D = 1 + \frac{100K\Omega}{R_G}$$

Nel nostro caso la resistenza R_G coincide con R22 ed R23, inserite tra i pin 1 e 8 di ciascun AD623 (la cui piedinatura è visibile in figura 6.2); il valore di tale resistenze dovrà essere:

$$R22 = R23 = \frac{100K\Omega}{A_D - 1} = 3.359K\Omega$$

ma ovviamente si sceglie il valore più vicino della serie resistiva E12, che risulta essere 3.3 KΩ ± 1%. Il guadagno reale nominale risulterà, pertanto, essere:

$$A_D^* = 31.303$$

La tensione V_{REF}, invece, si ricava dal partitore formato dalle resistenze R3 ed R4 (e, in modo analogo, R5 ed R6), che poniamo uguali e del valore di 1 KΩ ± 1%, partendo dalla tensione di riferimento a 3 V (REF_3V), comune a tutta la Power Management Tile.
Il valore nominale ottenuto per la tensione di offset sarà:

$$V_{REF}{}^* = REF\_3V \cdot \frac{R3}{R3 + R4} = 1.5V$$

che si discosta esiguamente dal valore teorico desiderato.
Si possono così ottenere tre differenti dinamiche di variazione del segnale in uscita dal blocco del magnetometro, a seconda che il sensore abbia un comportamento tipico oppure nei casi estremi di parametri massimi e minimi:

$$typical \Rightarrow MAGN\_X_{TYP} = [0.894 \div 1.52]V$$

$$\max \Rightarrow MAGN\_X_{MAX} = [1.696 \div 2.478]V$$
$$\min \Rightarrow MAGN\_X_{MIN} = [-0.065 \div 0.717]V$$

In tutti e tre i casi si suppongono range di tensioni di uscita corrispondenti ad un range di variazione del campo magnetico di $[-0.625 \div 0.625]G$ (valori in realtà superiori ai massimi plausibili), inoltre si può vedere come la dinamica di ingresso dell'ADC sia stata rispettata, tranne che nell'ultimo caso: ciò è dovuto al fatto che $A_D^* > A_D$, ma non rappresenta un problema, in quanto è relativo ad un valore della componente Bx (-0.625 G) che è quasi impossibile raggiungere, e se anche si presentasse all'ingresso dell'ADC sarebbe riconosciuto come 0 V e non danneggerebbe il convertitore. Lo stesso dicasi per il valore massimo, riferito al secondo caso ed abbastanza vicino al limite superiore della dinamica dell'ADC: qualora esso risultasse (molto leggermente) superiore al valore di 2.5 V, a causa dell'innalzarsi di alcuni parametri al di là del valore massimo dichiarato (per via di fattori esterni), verrebbe letto come 2.5 V che, convertito in digitale, equivale ad un vettore di bit tutti settati ad "1".
Come ulteriore protezione per gli ingressi di quest'ultimo, si è deciso di introdurre altri due componenti: il diodo zener CZRU52C3 della Comchip (D1 e D2 nello schema elettrico), che ha la funzione di limitare la tensione in uscita dal circuito di condizionamento al valore di 3 V, e la resistenza R8 (analoga R7), del valore di 1 KΩ ± 1%, che limita la corrente che scorre nello zener ad un valore < 3 mA.
Concludendo, si può scrivere la relazione che lega il campo magnetico misurando con il valore di tensione in uscita dall'intero blocco *Bk1B221_Magnetometer_Sensor*, per valori tipici dei parametri del sensore:

$$MAGN\_X = V_{A\_TYP}(B_x) \cdot A_D^* + V_{REF}^* = 0.5008 \frac{V}{G} \cdot B_x + 1.207V$$

Come si poteva vedere nel diagramma degli oggetti *Bk1B22_Magnetic_Attitude_Subsystem* (cap. 4), è stato definito il seguente attributo, relativo al magnetometro:

$$SENS\_MAGNETIC = 5008 \frac{V}{T}$$

Ricordando che il Tesla [T] è un multiplo del Gauss, tale valore esprime l'effettiva sensibilità complessiva del blocco magnetometro, per parametri tipici del sensore. In uscita dall'ADC si otterrà quindi un numero binario su *n* bit in unità di *SENS_MAGNETIC_FIELD_RAW* [T/LSB], dove:

$$SENS\_MAGNETIC\_FIELD\_RAW = 1/(SENS\_MAGNETIC \cdot SENS\_ADC)$$
$$\text{con } SENS\_ADC\left[\frac{LSB}{V}\right] = \frac{2^n - 1}{2.5V}$$

considerando un ADC a 12 bit e sostituendo i valori otterremmo un valore digitale in uscita con una risoluzione di $1.219 \cdot 10^{-7}$ T, che chiaramente differisce dal valore di *SENS_MAGNETIC_FIELD* richiesto nelle specifiche per la rappresentazione del campo magnetico (si veda cap. 3 e 4): ecco perché nell'esecuzione del caso d'uso ***Boot*** è prevista la moltiplicazione del fattore di scalamento del guadagno per il seguente parametro (cap. 5):

$$\frac{SENS\_MAGNETIC\_FIELD\_RAW}{SENS\_MAGNETIC\_FIELD}$$

In tal modo il valore in uscita dall'ADC viene riscalato affinché possa essere salvato in memoria con l'unità di misura desiderata.

6.1.3. Circuito di Set/Reset

Secondo i dati forniti dal costruttore del sensore di campo magnetico, per ottimizzare la precisione della misura effettuata, è sufficiente collegare il dispositivo ad un circuito, denominato di ***Set/Reset***, tramite i piedini preposti. Tale circuito è in grado di fornire un picco di corrente positiva (impulso di SET), seguito da un picco di corrente negativa (impulso di RESET), entrante nel pin S/R+ e uscente dal pin S/R-, per ambo gli assi di misura A e B, così da riorientare i dipoli (il cui orientamento casuale, a lungo andare, degrada la sensibilità del sensore) degli elementi magneto-sensibili contenuti nelle resistenze dei ponti.
Questa corrente, che raggiunge valori superiori ai 3 A, consente quindi di massimizzare la sensibilità del sensore, minimizzare il rumore, eliminare l'effetto di memoria degli elementi magnetici e ridurre gli effetti dei forti campi di disturbo, migliorando la ripetibilità della misura.
Inoltre, la procedura di Set/Reset permette di conoscere con precisione la tensione di offset sull'uscita del sensore, che, riportata all'uscita finale del circuito di misura, fornisce la tensione di offset complessiva. Quest'ultima verrà salvata ed utilizzata durante la calibrazione, sottraendola al valore letto, così da eliminare diversi contributi di errore dovuti agli offset (vedi par. 6.1.4).

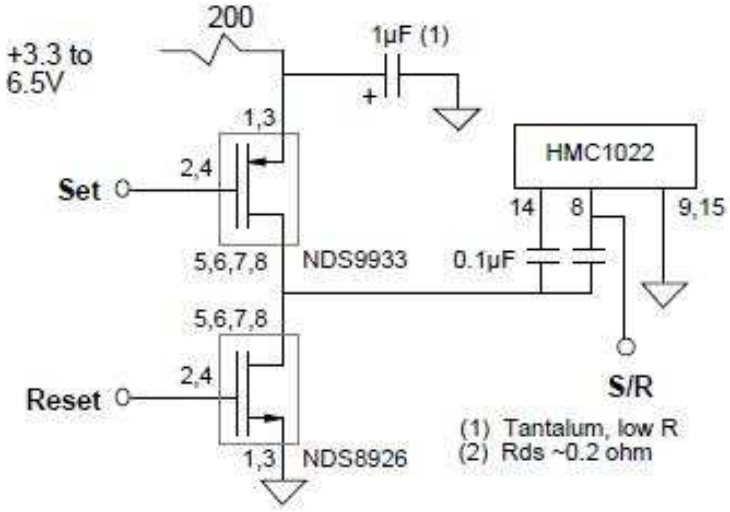


Figura 6.3 Circuito di Set/Reset per sensore HMC1002

Tra le varie possibili configurazioni del circuito di S/R proposte nel datasheet del sensore HMC1002, è stata scelta quella di pag. 11 (visibile anche in figura 6.3), adatta ad applicazioni Low Power e con i due segnali di controllo Set e Reset gestibili separatamente mediante microcontrollore, in modo da poter meglio adattare la lettura dell'offset all'impostazione progettuale di acquisizione dei dati (funzione *housekeeping(index)*, cap. 5).

Il circuito comparirà duplicato ed in parallelo a sé stesso nello schema elettrico, perché utilizzato sia per il ponte A che per il B.
In figura 6.4 sono riportate le forme d'onda dei segnali di comando per il circuito di S/R utilizzato e della relativa corrente di S/R.

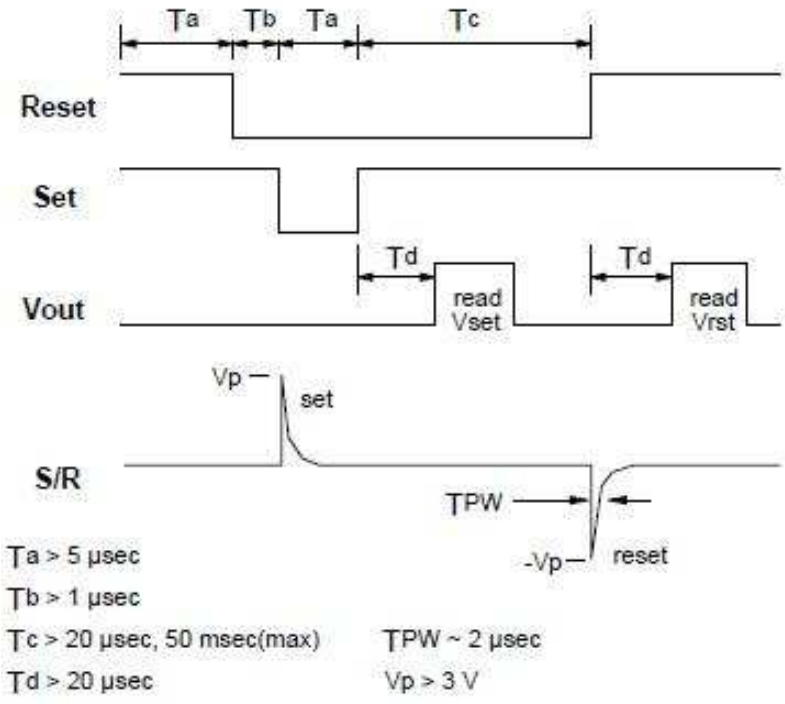


Figura 6.4 Forme d'onda segnali Set, Reset e corrente HMC1002

Il segnale Set comanda il MOSFET a canale p, pertanto nella piedinatura finale del progetto è stato rinominato *notSET*, anche se per ora faranno fede i nomi dei segnali riportati nella figura sopra, mentre il segnale Reset comanda il MOSFET a canale n.

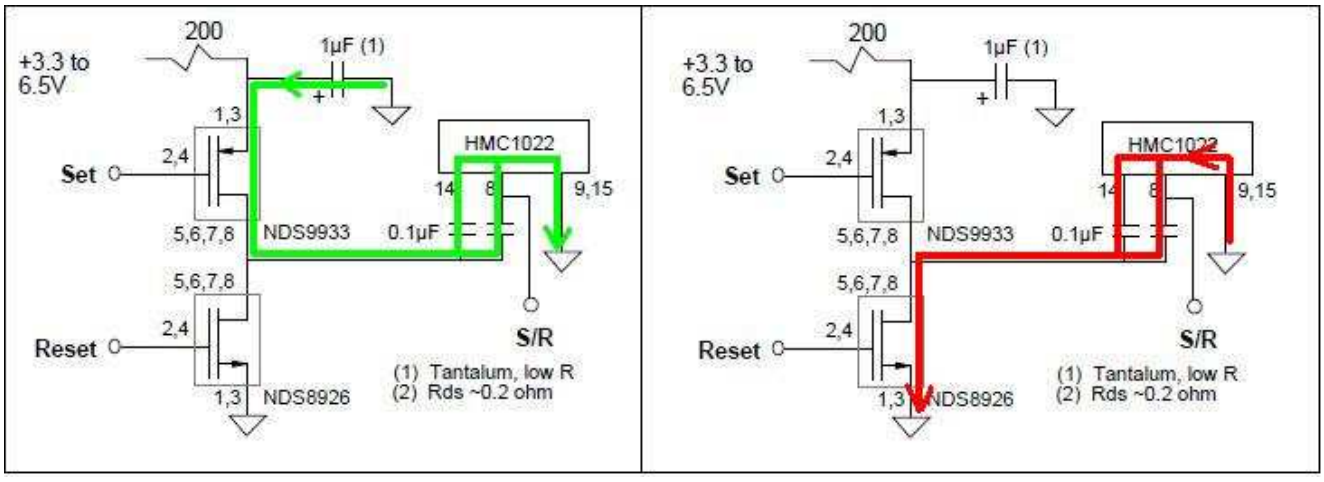


Figura 6.5 Percorsi delle correnti impulsive di Set e Reset del magnetometro

Quando ci si trova nella situazione Set=0, Reset=0 (a sinistra in figura 6.5), essendo in conduzione il p-MOS ed interdetto l'n-MOS, la corrente (positiva) entra nel pin S/R+ (A o B) ed esce dal pin S/R- (A o B), collegato a massa, originando l'impulso di Set. Nel caso Set=1, Reset=1, invece, conduce l'n-MOS mentre il p-MOS è interdetto, e la corrente scorre in senso opposto (a destra in figura) dando luogo all'impulso di Reset.
Indicando con $R_{S/R}$ il *Set/Reset Strap*, ovvero la resistenza vista dalla corrente impulsiva tra i piedini di S/R, e con R_{DS_p} la resistenza di ciascun p-MOS in conduzione e tenendo presente l'esistenza di due circuiti identici in parallelo, si possono ricavare gli estremi della corrente di Set:

$$I_{SET,MAX} = \frac{3.3V}{\frac{R_{S/R,MIN}}{2} + \frac{R_{DS\_p,MIN}}{2}}$$

$$I_{SET,MIN} = \frac{3.3V}{\frac{R_{S/R,MAX}}{2} + \frac{R_{DS\_p,MAX}}{2}}$$

Non essendo reperibili i MOS indicati nel circuito proposto, sono state operate le seguenti sostituzioni:

MOS a canale n: NDS8926 => IRF7311
MOS a canale p: NDS9933 => IRF7324

Sapendo che:

$$R_{DS(on)\_IRF7311} = [0.029 \div 0.046]\Omega$$

$$R_{DS(on)\_1RF7324} = 0.026\Omega$$

$$R_{S/R} = [1.5 \div 1.8]\Omega$$

si ricava:

$$I_{SET,MAX} \cong 4.3A$$

$$I_{SET,MIN} \cong 3.6A$$

Discorso analogo vale per la corrente di Reset:

$$I_{RESET,MAX} = \frac{3.3V}{\frac{R_{S/R,MIN}}{2} + \frac{R_{DS\_n,MIN}}{2}} \cong 4.3A$$

$$I_{RESET,MIN} = \frac{3.3V}{\frac{R_{S/R,MAX}}{2} + \frac{R_{DS\_n,MAX}}{2}} \cong 3.6A$$

dove R_{DS_n} indica la resistenza di ciascun n-MOS in conduzione.
La situazione Set=0, Reset=1 (entrambi i MOS in conduzione) è da evitarsi, oltre che impossibile se si seguono correttamente le forme d'onda consigliate, mentre con Set=1, Reset=0, invece, non vi è passaggio di corrente (entrambi i MOS interdetti).

Facendo sempre riferimento alle forme d'onda in fig. 6.4, si leggono i valori di tensione in uscita dal sensore (o meglio, da tutto il circuito), come indicato in figura, dopo l'impulso di Set e dopo l'impulso di Reset, chiamati rispettivamente Vset e Vrst. L'offset (OS) dovuto al sensore può semplicemente essere calcolato facendo la media di questi due valori:

$$OS = \frac{Vset + Vrst}{2}$$

Una volta applicata la sequenza di Set/Reset, il sensore si trova in condizioni ottime di sensibilità e precisione di misura, e l'offset assume un valore fisso e conoscibile, così da rendere quasi trascurabili gli errori residui nella misurazione del campo magnetico.
Il sensore può rimanere in questo stato anche per anni, o finché non sarà soggetto ad un campo disturbante (>4 G). Per ovviare a questa possibilità ed operare sempre nelle migliori condizioni possibili per quanto riguarda la correttezza della misura effettuata, è consigliabile applicare gli impulsi di S/R ogni 10 minuti circa, come mostrato nel flow-chart in figura 6.6, in modo da stabilizzare il sensore e aggiornare il termine di offset, da sottrarre poi all'uscita.

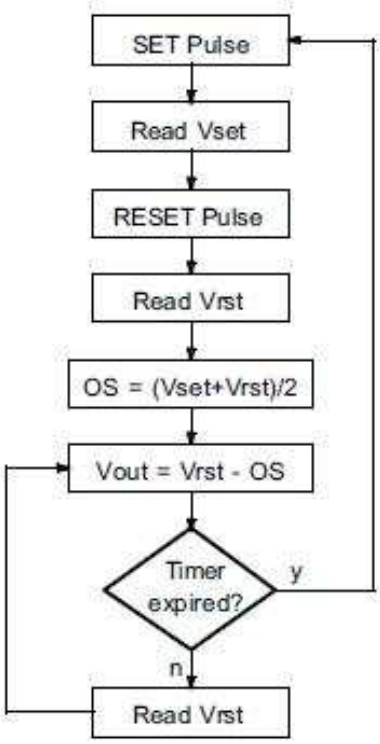


Figura 6.6 Diagramma di flusso per il calcolo dell'offset con la procedura di Set/Reset

6.1.4. Calcolo dell'errore di misura

Si vuole ora considerare l'errore introdotto nella misura del campo magnetico da parte degli elementi che costituiscono il blocco del magnetometro. Dal momento che i circuiti sono gli stessi, replicati per le due uscite MAGN_X e MAGN_Y, il massimo errore possibile sarà il medesimo per entrambe.
Analizzando separatamente i contributi di errore dei diversi componenti, si può notare che tale errore non è trascurabile, e dipende principalmente dal sensore di campo magnetico, dal circuito di condizionamento e dal convertitore analogico/digitale.
Per quanto riguarda il sensore HMC1002, riferendosi al datasheet, si possono distinguere i seguenti tipi di errore:

- Errore di **variazione della sensibilità**:

  rappresenta la variazione dell'uscita in funzione della variazione massima della sensibilità del sensore dal suo valore tipico.

$$\Delta V_A^{(1)} = \Delta k_{M,MAX} \cdot B_{MAX} = \left(k_{M\_MAX} - k_{M\_TYP}\right) \cdot B_{MAX}$$
$$\Rightarrow \Delta V_A^{(1)} = (20-16)\frac{mV}{G} \cdot 0.625G = 2.5mV$$

- Errore di **linearità** (Linearity error):

  tale errore è dovuto alla non idealità della caratteristica del sensore, che pertanto si discosta leggermente dalla retta rappresentata dall'equazione di VA. Per misure comprese nel range da -1 G a 1 G, tale errore è dello 0.5% della Full Scale (FS) al massimo. Introduce un errore in uscita pari a:

$$\Delta V_A^{(2)} = 0.5\% \cdot FS \cdot k_{M\_TYP} = 0.5 \cdot 10^{-2} \cdot 2G \cdot 16\frac{mV}{G} = 0.16mV$$

- Errore di **isteresi** (Hysteresis error):

  rappresenta l'errore introdotto dal sensore, per via dell'isteresi, quando il campo magnetico misurando assume valori che cambiano di segno; il suo valore massimo dichiarato è dello 0.1% FS, corrispondente al seguente errore sulla tensione di uscita:

$$\Delta V_A^{(3)} = 0.1\% \cdot FS \cdot k_{M\_TYP} = 0.032mV$$

- Errore di **ripetibilità** (Repeatability error):

  anche questo errore introduce un'incertezza massima pari allo 0.1% FS.

$$\Delta V_A^{(4)} = 0.1\% \cdot FS \cdot k_{M\_TYP} = 0.032mV$$

- Errore sulla **tensione di alimentazione**:

  esso si traduce in una variazione della sensibilità del sensore, dal momento che, secondo i dati rilasciati dal costruttore, essa dipende da V_{al}.
  Per l'alimentazione dei ponti, al fine di rispettare la precisione del sensore una volta calibrato (si veda più avanti), si è scelto di utilizzare, anziché la tensione di 5 V comune a tutta la Power Management Tile, proveniente da un regolatore del tipo L7805 e quindi poco precisa, l'uscita del riferimento di tensione REF02HSZ della Analog Devices, impiegato ad-hoc per questo progetto, in grado di assicurare ai ponti magnetoresistivi sufficiente corrente di alimentazione e di garantire un'accuratezza dello 0.5% sulla tensione di uscita a 5 V.
  Di conseguenza, V_{al} introdurrà un errore massimo dello 0.5% sulla sensibilità assoluta del sensore, che corrisponde, in uscita, a:

$$\Delta V_A^{(5)} = k_{M\_TYP} \cdot 0.5\% \cdot B_{MAX} = 16\frac{mV}{G} \cdot 0.005 \cdot 0.625G = 0.05mV$$

- **Deriva della sensibilità** con la temperatura (Sensitivity Tempco):

  questo errore è dovuto al discostarsi della sensibilità del sensore dal suo valore tipico in seguito a variazioni della temperatura, ed è quantificabile mediante il coefficiente detto Sensitivity Tempco, di valore negativo (poiché la sensibilità va degradandosi con la temperatura); dal datasheet si legge che il valore tipico di tale variazione è di -0.3% della variazione di temperatura rispetto alla temperatura di 25°C. Nel progetto in esame, la massima variazione di temperatura si avrà nel limite inferiore di -30°C (par. 2.1). La risultante incertezza sulla sensibilità vale:

$$\Delta k_{M,MAX}^{TEMP}[\%] = -0.3\frac{\%}{^\circ C} \cdot \Delta T_{MAX} = -0.3\frac{\%}{^\circ C} \cdot (-30-25)^\circ C = 16.5\%$$
$$\Rightarrow \Delta k_{M,MAX}^{TEMP}\left[\frac{mV}{G}\right] = 16.5\% \cdot k_{M\_TYP} = 2.64\frac{mV}{G}$$

  che contribuisce ad un errore in uscita dal sensore pari a:

$$\Delta V_A^{(6)} = \Delta k_{M,MAX}^{TEMP} \cdot B_{MAX} = 1.65mV$$

- **Deriva dell'offset** con la temperatura (Bridge Offset Tempco):

  tale errore indica la variazione dell'offset del sensore in funzione della variazione della temperatura, quantificata dal coefficiente Bridge Offset Tempco, che, utilizzando la procedura di Set/Reset, vale ±0.001%/°C.
  Si evince chiaramente come un contributo di errore di questa entità sia trascurabile rispetto agli altri, inoltre, occorre tenere presente che qualsiasi variazione dell'offset non ha effetti sul risultato finale della misura, in quanto esso verrà epurato da tale termine, ricavato grazie al Set/Reset, prima di essere salvato

(par. 6.1.3). La procedura di S/R verrà effettuata anche in orbita durante la missione del satellite, perciò l'offset, qualsivoglia esso sia, verrà calcolato e sottratto, anche a temperature ambiente diverse.

Come si può vedere, anche la possibile variazione dell'offset tra -37.5 mV e 18.75 mV (par. 6.1.2) non è stata considerata come errore, in quanto la procedura di Set/Reset ci permette di conoscere con precisione, periodicamente o su richiesta, l'entità di questo contributo, e di eliminarlo dalla misura risultante.
Passando ad analizzare l'errore dovuto al circuito di condizionamento, vediamo che esso si suddivide nei seguenti contributi:

- Errore intrinseco nel guadagno differenziale dell'AD623:
  secondo quanto dichiarato dal costruttore, per guadagni maggiori di 1 si ha:

$$\Delta A_D{}^{(1)}[\%] = 0.35\%$$

- Errore introdotto dalla resistenza usata per regolare il guadagno:
  dal momento che il guadagno differenziale dell'AD623 viene regolato dalla resistenza posta tra i suoi pin 1 e 8, questa introduce un errore pari alla sua tolleranza, ovvero dell'1%

$$\Delta A_D{}^{(2)}[\%] = 1\%$$

- Errore intrinseco di offset dell'AD623:
  questo contributo è dovuto alle due componenti di offset, V_{OSin} e V_{OSout}, rispettivamente in ingresso e in uscita, dell'amplificatore operazionale; alla sua uscita si vedrà pertanto la sovrapposizione del primo, moltiplicato per il guadagno, e del secondo:

$$V_{OS} = V_{OSin} \cdot A_D + V_{OSout} = 200\mu V \cdot 31.303 + 1mV = 7.26mV$$

  Non è stato considerato il contributo delle correnti di offset e di bias dell'operazionale, in quanto fornirebbero contributi di errore trascurabili. Sul datasheet dell'AD623 si vede infatti che, nel completo range di temperature [-40 ~+85]°C, $I_{b,max}$ = 27.5 nA (e comunque non contribuirebbe all'offset perché l'amplificatore è differenziale) e $I_{OSin,max}$ = 2.5 nA.

- Errore su V_{REF}:
  la tensione di traslazione dell'uscita viene generata mediante due resistenze uguali aventi ciascuna tolleranza dell'1% ed un riferimento di tensione, utilizzato per generare REF_3V. Come già anticipato, non è ancora stato scelto, a livello globale, il modello di riferimento di tensione da utilizzarsi a tal fine, pertanto si indicherà con ε_{REF_3V} la tolleranza, incognita, di questo componente. Si ricava:

$$\Delta V_{REF} = 3V \cdot \left(1\% + 1\% + \varepsilon_{REF\_3V}[\%]\right) = 60mV + \varepsilon_{REF\_3V}[\%] \cdot 3V$$

Tuttavia si può presupporre, considerando l'elevata accuratezza dei riferimenti di tensione in commercio, nel caso peggiore, una tolleranza non superiore allo 0.5%. Sostituendo tale valore si ottiene:

$$\Delta V_{REF,MAX} = 3V \cdot (1\% + 1\% + 0.5\%) = 75mV$$

A questo punto, occorre precisare che anche per quanto riguarda tutta l'elettronica a valle del sensore, per i contributi di offset vale lo stesso discorso fatto prima per il sensore stesso: essi sono stati riportati per rendere conto sull'entità (maggiore) di errore che si avrebbe se non si utilizzasse la procedura di Set/Reset, a causa dei contributi di offset, senza contare il degradamento di altri parametri nel tempo, che l'utilizzo di tale procedura consente di evitare.
In poche parole, se dal valore in uscita dall'ADC si sottrae la componente di offset conosciuta ed imposta in fase di progetto, ovvero quella che corrisponde a Bx=0, allo stesso tempo, grazie al S/R, nella misura dell'offset viene inclusa anche la componente di errore che affligge il medesimo ed anch'essa verrà rimossa. Pertanto, gli ultimi due contributi di errore, V_{OS} e ΔV_{REF}, verranno tralasciati poiché considerati bilanciati.
Infine, non va tralasciato, considerando gli errori di misura, quello introdotto dall'ADC, anche se esso non è prettamente incluso nell'elettronica del blocco in oggetto, ma è parte integrante del magnetometro, inteso come strumento di misurazione del campo magnetico, la cui uscita viene restituita in formato digitale ed opportunamente riscalata in unità di 1 mG. Facendo riferimento ai dati dell'ADC a 12 bit integrato sul microcontrollore scelto per AraMiS (MSP430F5437 della Texas Instruments), vediamo che tale errore è tipicamente di 2 LSB, quindi, tenendo presente che la tensione in ingresso al convertitore ha una dinamica di 2.5 V, ivi si avrà un errore:

$$\Delta V_{ADC} = 2 \cdot LSB[V] = 2 \cdot \frac{2.5V}{2^{12}} = 1.22mV$$

Ricapitolando, si considera la sovrapposizione degli effetti di:

- Errori dovuti al sensore, opportunamente riportati in uscita dal condizionamento moltiplicandoli per il guadagno
- Errore complessivo di guadagno del condizionamento
- Errore all'ingresso dell'ADC

E, osservando l'uscita del circuito di condizionamento, ossia l'ingresso dell'ADC, si riscontra il seguente errore:

$$\varepsilon_{MAGN\_X}[mV] = \Delta V_{A,MAX} \cdot A_D \cdot (1 + \Delta A_D) + \Delta V_{ADC}$$

dove

$$\Delta V_{A,MAX} = \Delta V_A{}^{(1)} + \Delta V_A{}^{(2)} + \Delta V_A{}^{(3)} + \Delta V_A{}^{(4)} + \Delta V_A{}^{(5)} + \Delta V_A{}^{(6)} =$$
$$= [2.5 + 0.16 + 0.032 + 0.032 + 0.05 + 1.65]mV = 4.424mV$$

e

$$\Delta A_D = \Delta A_D{}^{(1)} + \Delta A_D{}^{(2)} = \frac{1}{100} + \frac{0.35}{100} = 0.0135$$

da cui si ottiene:

$$\varepsilon_{MAGN\_X}[mV] = 4.424mV \cdot 31.303 \cdot 1.0135 + 1.22mV \cong 141.6mV$$

Indicando con $\Delta MAGN\_X_{TYP}$ e ΔB, rispettivamente, il range tipico di tensione in uscita dal blocco e il range di campo magnetico misurabile, e sapendo che

$$\Delta MAGN\_X_{TYP} = 1.52V - 0.894V = 626mV$$
$$\Delta B = 0.625G - (-0.625)G = 1.25G$$

possiamo riportare l'errore ε_{MAGN_X} in Gauss utilizzando la seguente proporzione:

$$\varepsilon_{MAGN\_X}[G] : \varepsilon_{MAGN\_X}[mV] = \Delta B : \Delta MAGN\_X_{TYP}$$

Invertendo opportunamente la formula e sostituendo si ottiene:

$$\varepsilon_{MAGN\_X}[G] = \frac{1.25G \cdot 141.6mV}{626mV} = 0.283G$$

Un errore di questa entità rappresenta oltre il 22% del range di misura scelto per il progetto, e pertanto non è assolutamente accettabile. È chiara, quindi, la necessità di un'ulteriore fase di compensazione.
Tale compensazione dell'errore viene eseguita durante le routine software del progetto, mediante l'utilizzo dei parametri della struttura *t_scaling*, ovvero delle correzioni di offset e guadagno. Questa struttura (o classe) è comune a tutto il progetto di AraMiS e il suo metodo principale consiste, appunto, nel sottrarre il valore contenuto nell'attributo *offset* dal valore in uscita dall'ADC, e nel moltiplicarlo, poi, per il fattore correttivo di guadagno, salvato in *gain*, consentendo, quindi, di calibrare gli errori di tutti i vari sottosistemi che effettuano misurazioni.
Nel caso specifico del modulo 1B22, ed in particolare del blocco *Bk1B221_Magnetometer_Sensor*, i parametri di offset per le uscite x e y, come già detto sopra, verranno ricavati tramite la procedura di S/R, mentre il parametro di gain verrà calibrato effettuando ripetute e periodiche misure di valori noti in ambiente ideale. Con tale calibrazione effettuata a terra (si veda il caso d'uso ***Ground Calibration*** nel cap. 3) è dunque possibile compensare:

- L'errore dovuto alla variazione della sensibilità del sensore di campo magnetico, incluso il contributo di errore dovuto al variare della tensione di alimentazione

- Entrambi gli errori di guadagno dell'operazionale di condizionamento

Inoltre, una volta che AraMiS verrà lanciato in orbita, grazie alla sensoristica di bordo sarà possibile anche la misurazione della temperatura.
Una volta conosciuta la temperatura (ipotizzando continuità termica tra termometro e sensore di campo magnetico), sarà possibile compensare anche l'errore di deriva della sensibilità con la temperatura. Occorre però una precisazione: la compensazione verrà effettuata, conoscendo il valore di temperatura operativa, in base al valore tipico di deriva della sensibilità del sensore, riportato sul datasheet, di -0.3%/°C. Come si può vedere, sempre dal datasheet, però, il Sensitivity Tempco può oscillare da un valore di -0.32%/°C ad uno di -0.28%/°C. Tale variazione, che prima non era stata considerata in quanto trascurabile rispetto all'entità

dell'errore senza compensazione, ora va tenuta in esame, poiché dà origine ad un contributo di errore non compensabile via software.
Dopo la compensazione SW, rimangono quindi esclusivamente i contributi di:

- Errore di non linearità del sensore
- Errore di ripetibilità della misura del sensore
- Errore di isteresi del sensore
- Errore dell'ADC
- Errore di **variazione del Sensitivity Tempco**:
  per tenere conto di questo errore con precisione, va considerato che esso introduce una variazione, in modulo, per la Sensitivity Tempco, di:

$$\begin{aligned}\varepsilon_{SENS\_TEMPCO} &= \max\left|SENS\_TEMPCO - SENS\_TEMPCO_{TYP}\right| \\ &= \left|-0.32\frac{\%}{°C} - 0.3\frac{\%}{°C}\right| = 0.02\frac{\%}{°C}\end{aligned}$$

  Questa incertezza massima è relativa al valore della sensibilità a temperature operative diverse da 25°C: in particolare, consideriamo la variazione della sensibilità tipica del sensore a -10°C, che assumiamo come temperatura tipica di funzionamento in orbita e fungerà da valore di riferimento per la compensazione:

$$\begin{aligned}k_{M\_TYP}[-10°C] &= k_{M\_TYP}[25°C]\cdot\left[1+0.3\frac{\%}{°C}(-10°C-25°C)\right] \\ &= 16\frac{mV}{G}\cdot[1-10.5\%] = 14.32\frac{mV}{G}\end{aligned}$$

  Una volta effettuata la compensazione, in base alla Sensitivity Tempco tipica di -0.3%/°C, lo scalamento del valore in uscita verrà effettuato in funzione del valore attuale della sensibilità, k_{M_TYP}[-10°C], la cui variazione rispetto al valore originario k_{M_TYP}[25°C], pertanto, non viene più considerata errore. L'unico possibile errore sarà la variazione casuale $\varepsilon_{SENS\_TEMPCO}$, che dà luogo ad una maggiore (o minore) variazione della sensibilità rispetto a quella per cui si è compensato. Riferendoci sempre a -10°C si ottiene il seguente valore di sensibilità:

$$\begin{aligned}k^{ERR}_{M\_TYP}[-10°C] &= k_{M\_TYP}[25°C]\cdot\left[1+\left(0.3\frac{\%}{°C}+\varepsilon_{SENS\_TEMPCO}\right)\cdot(-10°C\text{ - }25°C)\right] \\ &= 16\frac{mV}{G}\cdot\left[1-0.32\frac{\%}{°C}\cdot 35°C\right] = 14.208\frac{mV}{G}\end{aligned}$$

  Ora si può ricavare la discrepanza tra la sensibilità compensata e quella dovuta alla massima variazione del coefficiente rispetto al suo valore tipico:

$$\Delta k_{M\_TYP}[-10°\mathrm{C}] = k_{M\_TYP}^{ERR}[-10°\mathrm{C}] - k_{M\_TYP}[-10°\mathrm{C}] = -0.112\,\frac{mV}{G}$$

Tale differenza, in questo caso, ha segno (-), poiché la sensibilità risulta maggiormente ridotta quando la Sensitivity Tempco (di valore negativo) ha modulo maggiore rispetto al suo valore tipico, ma ora interessa il valore assoluto di variazione della sensibilità del sensore a parità di temperatura, in modo da tener conto del peggiore errore possibile in uscita dal sensore. Inoltre consideriamo che, prendendo in esame il limite opposto di tale coefficiente, -0.28%/°C, avremmo ottenuto esattamente la stessa differenza, in modulo, ma con segno (+), perché la riduzione della sensibilità, per ogni °C, sarebbe stata dello 0.02% minore. Pertanto ora si considera il valore positivo di Δk_{M_TYP}, così da massimizzare il contributo di errore, nella somma finale di valori positivi.
In tal caso si vedrà in uscita dal sensore un errore:

$$\Delta V_A^{(7)} = \left|\Delta k_{M\_TYP}[-10°\mathrm{C}]\right| \cdot B_{MAX} = 0.112\,\frac{mV}{G} \cdot 0.625G = 0.07mV$$

L'errore finale risultante sarà quindi:

$$\varepsilon_{compensato}[mV] = \left(\Delta V_{A,compensato}\right) \cdot A_D + \Delta V_{ADC}$$

dove

$$\Delta V_{A,compensato} = \Delta V_A^{(2)} + \Delta V_A^{(3)} + \Delta V_A^{(4)} + \Delta V_A^{(7)} = 2 \cdot 0.032mV + 0.16mV + 0.07mV = 0.294mV$$

$$\Rightarrow \varepsilon_{compensato}[mV] = 0.294mV \cdot 31.303 + 1.22mV = 10.42mV$$

Tale errore, rapportato alla dinamica di uscita di ampiezza 626 mV, equivale all'1.7% circa, decisamente migliore del risultato che si avrebbe senza compensare. Moltiplicando tale errore percentuale per la massima dinamica del campo magnetico in ingresso (1.25 G), infine, si ottiene l'errore del sistema compensato in Gauss:

$$\varepsilon_{compensato}[G] \cong 0.021G$$

Il campo di misura del magnetometro così progettato vale:

$$B = [\pm 0.625 \pm 0.021]G$$

6.1.5. Consumo

La potenza dissipata in totale dal blocco del magnetometro, P_{MAGN}, è frutto di quattro contributi distinti:

- P_{SENS}: potenza dissipata dal sensore HMC1002
- P_{REF_5}: potenza dissipata dal riferimento di tensione REF02HSZ che alimenta il sensore
- P_{COND}: potenza dissipata dai circuiti di condizionamento
- $P_{S/R}$: potenza dissipata dal circuito di Set/Reset

Il consumo del sensore equivale al doppio di quello di un singolo ponte resistivo; la tensione di alimentazione, V_{al}, è applicata alla resistenza equivalente del ponte:

$$R_{eq\_bridge} = (2R)//(2R) = R$$

Dove R è il valore di ciascuna delle quattro resistenze che costituiscono il ponte, compreso fra 600 Ω e 1200 Ω. Quindi:

$$P_{SENS} = 2 \cdot \frac{V_{al}^{\ 2}}{R}$$

La massima potenza dissipata dal sensore, con una tensione di alimentazione di 5 V, sarà:

$$P_{SENS,MAX} = 2 \cdot \frac{V_{al}^{\ 2}}{R_{MIN}} = 2 \cdot \frac{(5V)^2}{600\Omega} \cong 83.3mW$$

Il riferimento di tensione REF02HSZ della Analog Devices, alimentato dalla tensione PDB, secondo quanto dichiarato nel datasheet [17], assorbe una corrente massima di 1.4 mA, e, pertanto, provocherà una dissipazione massima di potenza pari a:

$$P_{REF\_5,MAX} = V_{PDB,MAX} \cdot 1.4mA = 25.2mW$$

Il condizionamento avrà un consumo di potenza pari al doppio di quello relativo ad ogni singolo circuito, che a sua volta è la somma della potenza dissipata dall'amplificatore operazionale, P_{AO} e di quella dissipata nel generare la tensione V_{REF}, P_{REF_3}:

$$P_{COND} = 2 \cdot (P_{AO} + P_{REF\_3})$$

dove:

$$P_{AO,MAX} = I_{CC,MAX} \cdot V_{AL} = 625\mu A \cdot 5V \cong 3.1mW$$

Come $I_{CC,MAX}$ è stato considerato il peggior caso possibile, ovvero il valor massimo della corrente di alimentazione su tutto il range di temperature [-40°C~+85°C] (si veda a tal proposito il datasheet dell'operazionale AD623).

P_{REF_3} è la potenza consumata dal partitore resistivo formato da due resistenze uguali del valore di 1 KΩ:

$$P_{REF\_3} = \frac{(REF\_3V)^2}{2 \cdot 1K\Omega} = 4.5mW$$

Sostituendo si ottiene:

$$P_{COND,MAX} = 2 \cdot (3.1mW + 4.5mW) = 15.2mW$$

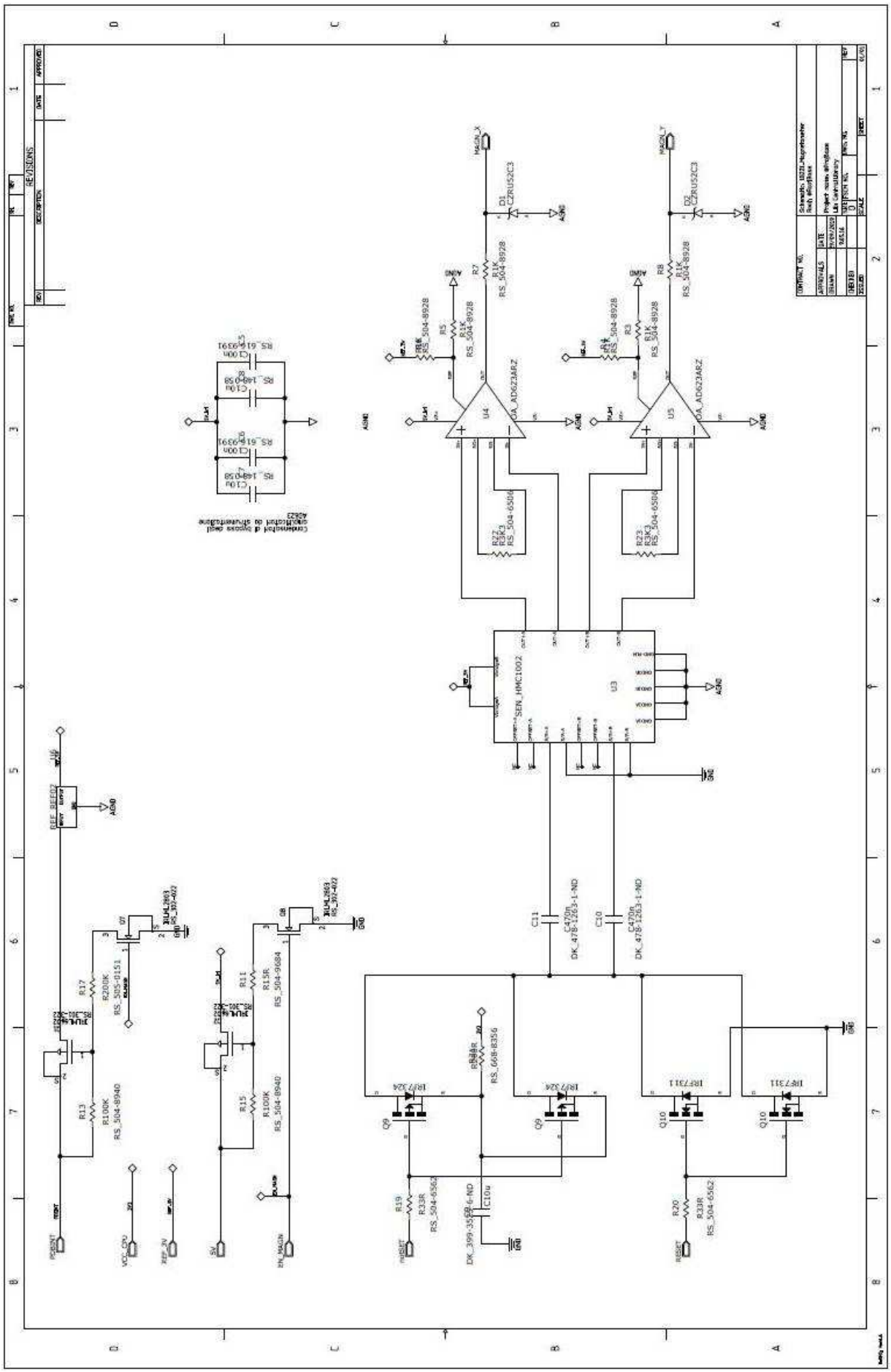
Infine, per quanto riguarda il circuito di Set/Reset, si potrebbe immaginare un consumo spaventoso, data l'entità delle correnti in gioco, ma si tratta di un valore di picco. Tali picchi impulsivi (~ 2 μs) di corrente, infatti, corrispondono circa ad un duty cycle dell'1% ed il costruttore garantisce una corrente media effettiva di Set/Reset inferiore ad 1 mA; si considera pertanto il valore massimo della potenza media dissipata dai MOS in conduzione (sapendo che il circuito viene alimentato a 3.3 V):

$$\overline{P}_{S/R,MAX} = 3.3V \cdot 1mA = 3.3mW$$

Ricapitolando, il consumo complessivo di potenza da parte del blocco *Bk1B221_ Magnetometer_Sensor*, vale:

$$P_{MAGN,MAX} = P_{SENS,MAX} + P_{REF\_5,MAX} + P_{COND,MAX} + \overline{P}_{S/R,MAX} =$$
$$= 83.3mW + 25.2mW + 15.2mW + 3.3mW = 127mW$$

6.1.6. Schema elettrico



6.2. Solenoide

La scelta ed il dimensionamento del solenoide, utilizzato come attuatore di coppia magnetica, sono già stati precedentemente trattati nel par. 2.2.2, dove si trovano indicati anche i valori massimi e minimi di corrente e tensione di alimentazione del medesimo, ai quali si farà riferimento in questa sede.
Gli obiettivi di questo sottocapitolo sono:

- Motivare la scelta del driver del solenoide
- Dimensionare opportunamente i componenti esterni al driver
- Misurare la corrente assorbita dal solenoide, attraverso un corretto dimensionamento del circuito di condizionamento
- Calcolare l'errore di misura di tale corrente
- Fornire un resoconto del consumo di potenza da parte del blocco di attuazione del controllo di assetto magnetico

Una volta misurata la corrente nel solenoide, si può finalmente ultimare il calcolo del momento angolare discretizzato, ricordandone la formula:

$$L_i = N \cdot S \cdot \left|\vec{B}\right| \cdot I \cdot \Delta t$$

Ora, conoscendo il valore di dipolo magnetico generato dal solenoide, non compaiono altre incognite nel calcolo del momento angolare, ed è quindi possibile, andando a leggere nei vettori di telemetria (cap. 3 e 4) i valori delle componenti di campo magnetico e della corrente I, attuare il pilotaggio del solenoide, decrementando ogni Δt il valore del momento angolare desiderato di una quantità L_i, finché tale valore non venga azzerato, ovvero, sia messo in atto il compito richiesto.
A titolo illustrativo, nella figura sottostante viene mostrato il solenoide, utilizzato per il controllo di assetto di AraMiS, del quale si è parlato in via teorica nel capitolo 2.

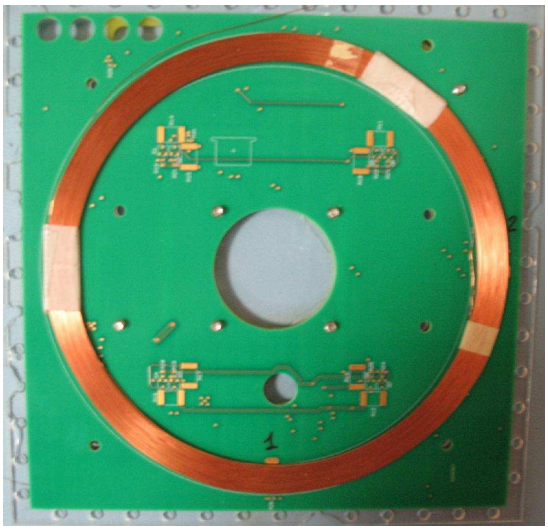


Figura 6.7 Solenoide utilizzato per il controllo di assetto magnetico di AraMiS (foto)

I due capi dell'avvolgimento di cui esso è costituito, verranno connessi ai due pin di uscita COIL1 e COIL2 del blocco di attuazione magnetica, *Bk1B222_Magnetic_Torque_Actuator*, il cui schema elettrico è visibile al fondo di questo sottocapitolo, in modo da rispettare la convenzione di segno della corrente (e, quindi, del dipolo magnetico e del momento angolare), enunciata nel sottocapitolo 2.3.

6.2.1. Driver

Dalle conclusioni tratte nel par. 2.2.2, si necessita di un driver in grado di:

- Alimentare in modo bidirezionale il solenoide (tramite circuito di tipo ponte ad H)
- Fornire al carico una corrente (bipolare) di almeno 645 mA
- Supportare tensioni di alimentazione per il carico fino a 18 V
- Avere diodi di ricircolo per la scarica dell'induttore
- Possibilmente, consentire di alimentare la logica con tensione diversa da quella del carico, per ridurre il consumo di potenza

Per far fronte a questi requisiti, è stato scelto il driver A3953 della Allegro MicroSystems [18], con le seguenti caratteristiche:

- Configurazione Full-Bridge
- Diodi di flyback integrati
- Corrente di uscita fino a ±1.3 A
- Tensione di alimentazione del carico fino a 50 V
- Regolazione interna della massima corrente fornita al carico (PWM)
- Tensione tipica di alimentazione della logica da 3 V a 5.5 V
- Diodi integrati per la protezione dai transitori
- Circuito interno di shutdown termico
- Protetto da corrente di crossover e UVLO
- Sleep-mode per ridurre il consumo di potenza quando non operativo

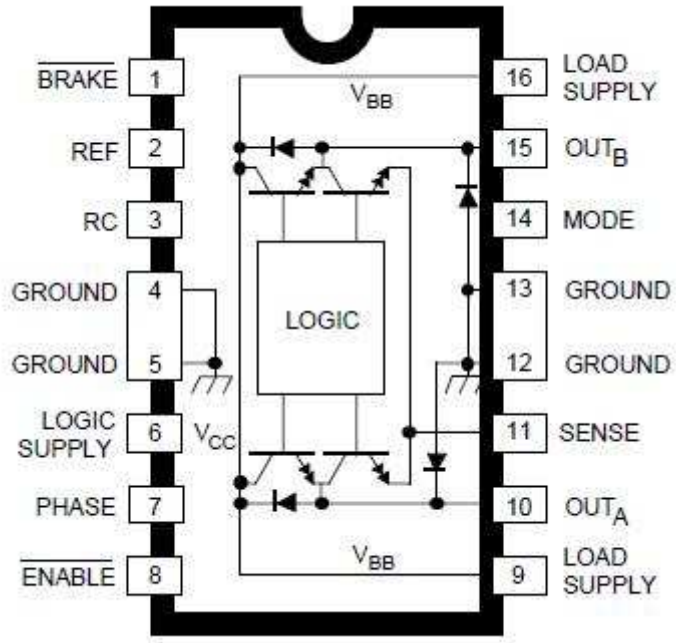


Figura 6.8 Piedinatura driver A3953

In figura 6.8 si può vedere la piedinatura del driver. I pin OUT_A e OUT_B verranno collegati ai due capi del solenoide, LOAD SUPPLY alla tensione PDB, LOGIC SUPPLY alla tensione di 5 V di sistema e tutti i pin GROUND alla massa digitale e di potenza, GND.

I pin notBRAKE, PHASE, notENABLE e MODE sono connessi ai segnali digitali in uscita dal microcontrollore adibiti al pilotaggio del solenoide. Il controllo del pilotaggio fa riferimento alla tabella di verità riportata qui sotto, presa dal datasheet dell'A3953. Le funzioni di tali segnali, in quanto attributi dell'oggetto *Bk1B222_Magnetic_Torque_Actuator*, sono state descritte anche nel cap. 4.
Per i pin REF, RC e SENSE si veda il paragrafo successivo.

BRAKE	ENABLE	PHASE	MODE	OUT_A	OUT_B	DESCRIPTION
H	H	X	H	Off	Off	Sleep Mode
H	H	X	L	Off	Off	Standby
H	L	H	H	H	L	Forward, Fast Current-Decay Mode
H	L	H	L	H	L	Forward, Slow Current-Decay Mode
H	L	L	H	L	H	Reverse, Fast Current-Decay Mode
H	L	L	L	L	H	Reverse, Slow Current-Decay Mode
L	X	X	H	L	L	Brake, Fast Current-Decay Mode
L	X	X	L	L	L	Brake, No Current Control

X = Irrelevant

Figura 6.9 Tabella di verità dei segnali di controllo del driver A3953

6.2.2. Componenti passivi

Il piedino di SENSE del driver A3953 viene utilizzato per la misurazione della corrente che scorre nel solenoide, e fornisce in uscita esattamente la stessa corrente, sempre positiva, ovvero uscente dal pin SENSE ed entrante a massa, sia che il solenoide venga pilotato "forward" (generazione di un dipolo magnetico positivo) che "reverse" (dipolo negativo).
Dal momento che anche in questo caso, come per la misurazione del campo magnetico, sarà l'ADC del microcontrollore a convertire il risultato della misura, si intende riportare tale valore di corrente in tensione. Per fare questo, viene connessa una resistenza tra il pin SENSE e massa, denominata appunto R_{SENSE}, ai capi della quale verrà preso il valore di tensione, proporzionale alla corrente in uscita, da condizionare poi opportunamente (par. 6.2.3).
Data l'entità della corrente di pilotaggio,

$$I_{\max} \cong 644.7mA$$
$$I_{\min} \cong 424.6mA$$

per poterne apprezzare al meglio le variazioni, nonché ridurre il consumo di potenza da parte della R_{SENSE}, si impone per quest'ultima un valore di 0.2 Ω. Il resistore che si è scelto di utilizzare a tal proposito, del medesimo valore, ha tolleranza 1% e classe di potenza ¼ W.
Al pin REF, invece, viene applicata una tensione di riferimento esterna, che chiameremo V_{REF_S}, utilizzata dal circuito PWM di controllo del driver per limitare la massima corrente fornibile al carico (per il funzionamento dettagliato di questo circuito si rimanda al datasheet dell'A3953), secondo la seguente relazione (letta nel medesimo datasheet):

$$I_{TRIP} \approx \frac{V_{REF\_S}}{R_{SENSE}} - I_{S0}$$

dove I_{TRIP} è appunto il limite massimo della corrente fruibile dal carico e I_{S0} è il contributo di offset della corrente di SENSE, dovuto alla corrente di base dei transistor nel driver, ed ha valore tipico, dichiarato dal costruttore, di 33 mA.
Per garantire un certo margine, si impone $I_{TRIP} = 1\ A > I_{max}$, e, sostituendo gli altri valori nella formula precedente ed invertendo, si ottiene:

$$V_{REF\_S} \approx \left(I_{TRIP} + I_{S0}\right) \cdot R_{SENSE} = 0.207V$$

V_{REF_S} si ricava dal partitore formato da R1 e R2 a partire dalla tensione di riferimento comune REF_3V (si veda lo schema elettrico, par. 6.2.6). Il valore di tale partitore dovrà essere:

$$\frac{R2}{R1+R2} = \frac{V_{REF\_S}}{REF\_3V} = \frac{0.207V}{3V} = 0.069$$

Utilizzando resistori della serie E12 e ponendo R2 = 1 KΩ ± 1%, si ricava R1 ≈ 13.5 KΩ. Ovviamente, poiché tale valore non esiste nella serie, si sceglie il valore più vicino:

$$R1 = 12K\Omega \pm 1\%$$

Sostituendo i valori scelti per le due resistenze si ricava il massimo valore reale della tensione in uscita dal partitore resistivo:

$$V_{REF\_S}^{*} = REF\_3V \cdot \frac{1K\Omega}{12K\Omega + 1K\Omega} = 0.23V$$

Invece, considerando il caso massimo, si avrà:

$$V_{REF\_S,MAX}^{*} = REF\_3V \cdot \frac{(1K\Omega + 1\%)}{(12K\Omega - 1\%) + (1K\Omega + 1\%)} = 0.235V$$

da cui si ricava il limite massimo della corrente, imposto dal controllo PWM, nel caso peggiore:

$$I_{TRIP,MAX}^{*} = \frac{0.235V}{(0.2\Omega - 1\%)} - I_{S0} = 1.154A$$

Come si può vedere, si è ancora al di sotto della massima corrente ammessa dal driver (1.3 A).
Qualora la corrente nel carico raggiunga il valore I_{TRIP}^{*}, il circuito di controllo spegne immediatamente i transistor, scaricando una minima parte dell'energia immagazzinata nel solenoide attraverso i diodi di flyback e facendo così diminuire leggermente tale corrente. Non appena trascorre un tempo t_{OFF} vengono riaccesi i transistor. La corrente viene così modulata come si può osservare in figura 6.10.

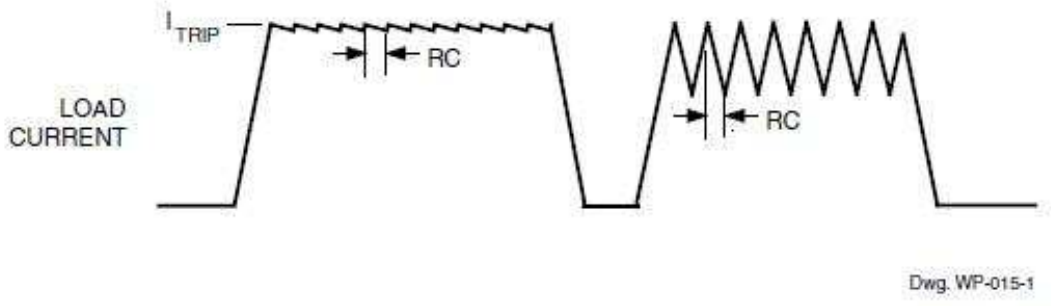


Figura 6.10 Corrente di carico driver A3953 con controllo PWM

Il tempo in cui i transistor rimangono spenti, t_{OFF}, può essere fissato connettendo una cella RC esterna all'apposito pin, denominato, appunto, RC, come si può vedere nello schema elettrico del blocco *1B222_Magnetic_Torque_Actuator* (par. 6.2.6). Nel datasheet si legge:

$$t_{OFF} \approx R_T \cdot C_T$$

Nello schema R_T e C_T coincidono rispettivamente con R9 e C2, collegati in parallelo tra il piedino RC del driver e massa. I valori di questi componenti sono quelli suggeriti dal datasheet: rispettivamente, 30 KΩ e 470 pF. In questo caso, $t_{OFF} \approx 14.1$ μs.

6.2.3. Condizionamento per la misura della corrente

Siccome il valore della corrente nella R_{SENSE} sarà sempre positivo, per condizionare il valore di tensione corrispondente, non è necessario fornire una componente di offset all'amplificatore di condizionamento, ma soltanto un opportuno guadagno.
Tuttavia, bisogna tenere conto di un problema, che nel blocco del sensore non sussisteva: mentre la resistenza R_{SENSE}, così come le tensioni di alimentazione del solenoide e della logica del driver, sono riferite a GND (la massa digitale e di potenza della Power Management Tile), gli amplificatori operazionali, così come i segnali analogici, sono riferiti ad AGND (la massa analogica).
I due diversi riferimenti avranno un solo punto di connessione all'interno del circuito globale e, mentre GND potrà subire diversi sbalzi di tensione e/o picchi di corrente, il riferimento AGND sarà mantenuto "pulito", in modo da garantire stabilità e precisione maggiori ai segnali analogici, come, ad esempio, le acquisizioni dei sensori.
Per ovviare a questa diversità di masse, è sufficiente utilizzare, per condizionare la tensione ai capi della R_{SENSE}, un amplificatore operazionale di tipo differenziale, come peraltro si era fatto per il magnetometro. Tenendo presente che l'assorbimento di corrente da parte dei pin di ingresso di un operazionale è pressoché nullo, si deduce che la corrente che scorre in questa resistenza, seppure di notevole intensità, non andrà a deteriorare il segnale in uscita dall'amplificatore, riferito ad AGND.
Data la semplicità del condizionamento richiesto per la misura della corrente, si è scelto di non dimensionarlo ad-hoc, ma di utilizzare uno dei circuiti di acquisizione differenziale di tensione già disponibili nel progetto del satellite AraMiS.
In particolare, nel sottoblocco *1B13_Sensors_and_Housekeeping* (facente parte del blocco 1B1, contenuto a sua volta in *1B8_Power_Management_Tile*), è stato scelto il componente *1B137A_Differential_Voltage_Sensor*, di cui si riporta qui sotto lo schema elettrico (figura 6.11).

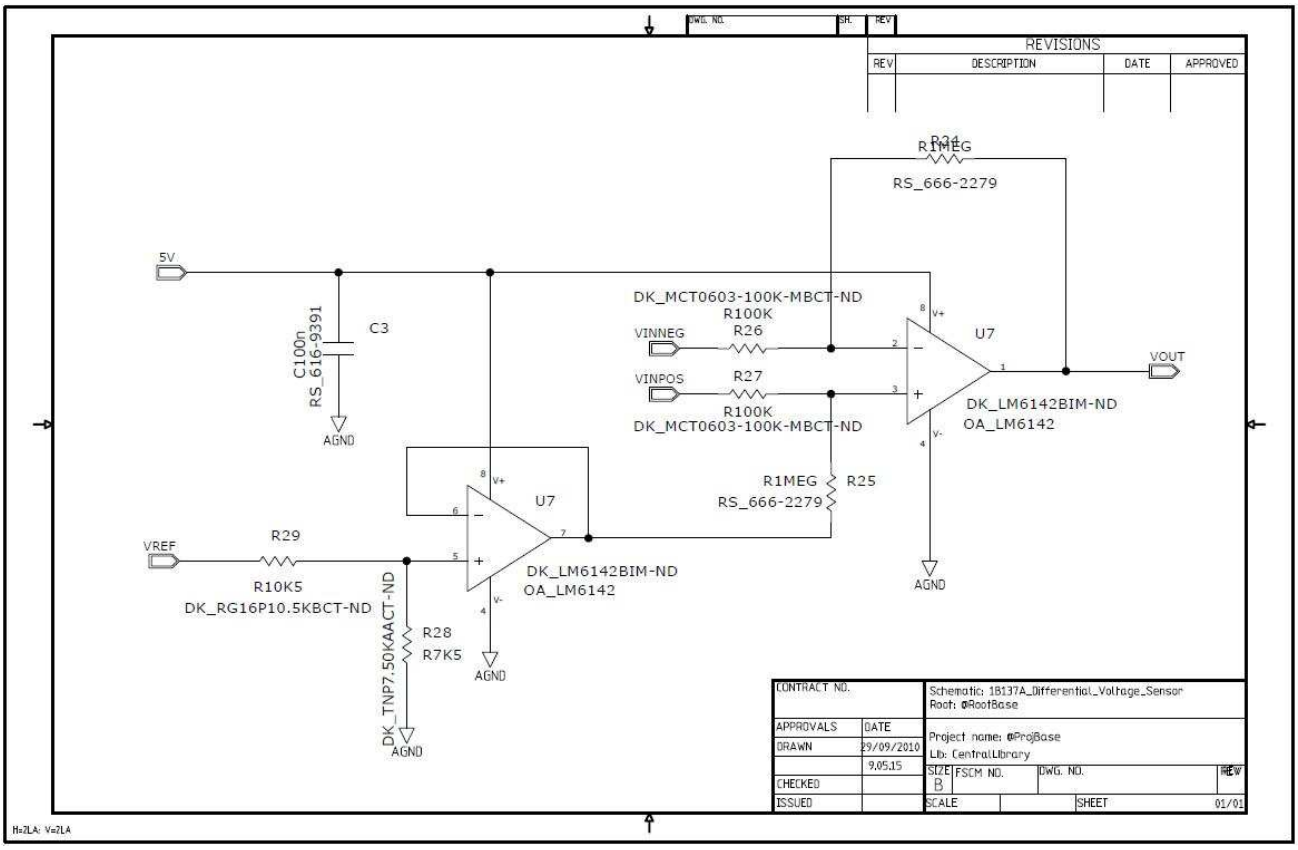


Figura 6.11 Schema elettrico circuito di condizionamento per la misura della corrente nel solenoide

Le caratteristiche di questo componente, già elencate in qualità di attributi dell'omonimo oggetto nel cap. 4, sono:

- Utilizzo di due op-amp LM6142
- Guadagno $A_D = 10$
- Tensione di uscita limitata a 2.5 V

Ora, sapendo che la tensione presente ai capi della R_{SENSE} vale

$$V_{R\_SENSE} = R_{SENSE} \cdot I = 0.2\Omega \cdot I$$

possiamo ricavare, moltiplicando per il guadagno, la tensione in uscita dal condizionamento (sullo schema elettrico anche il pin in uscita dal blocco si chiama SENSE come il pin di uscita del driver, ma, a differenza di quest'ultimo che è un'uscita in corrente, è in tensione, e quindi, viene specificata l'unità di misura per distinguere i due segnali):

$$SENSE[V] = 0.2\Omega \cdot 10 \cdot I = 2\Omega \cdot I$$

Pertanto, parlando di valori numerici, si avrà una tensione condizionata in uscita pari al doppio della corrente nel solenoide. Il suo valore massimo sarà:

$$SENSE_{MAX}[V] = 2\Omega \cdot I_{\max} = 2\Omega \cdot 644.7mA \cong 1.29V$$

Come si può osservare, utilizzando il Differential Voltage Sensor 1B137A, la dinamica dell'ADC è ampiamente rispettata, inoltre non sono necessari ulteriori circuiti di limitazione della tensione in uscita, in quanto tale componente è già limitato esattamente a 2.5 V.
In aggiunta, nel caso in cui la corrente dovesse eccedere il valore massimo teorico, essa verrebbe limitata dal circuito di controllo interno al driver al valore:

$$I_{TRIP}{}^{*} = \frac{V_{REF\_S}{}^{*}}{R_{SENSE}} - I_{S0} = \frac{0.23V}{0.2\Omega} - 33mA = 1.117A$$

La corrispondente tensione in uscita sarebbe:

$$I_{TRIP}{}^{*} \cdot 2\Omega = 2.234V$$

ed anche in questo caso si rispetterebbe la dinamica di ingresso dell'ADC. Per precisione, va ricordato che la corrente nel solenoide è ulteriormente controllata via software dal caso d'uso *Supervision* (cap. 3 e 5), che riscontra come errore una corrente > (I_{max} + 10%), valore comunque inferiore ad I_{TRIP}, disattivando immediatamente il pilotaggio del solenoide e spegnendo il driver. Tale controllo degli errori viene lanciato periodicamente, per cui, a meno che non succeda dopo un controllo e prima del successivo, è impossibile che la corrente raggiunga il valore I_{TRIP}.

6.2.4. Calcolo dell'errore di misura

Gli errori che interessano la misura, anche nel caso della corrente nel solenoide, vengono suddivisi a seconda di quale parte del circuito ne sia la causa, tenendo presente che tale corrente è considerata variabile, altrimenti non la si misurerebbe, pertanto si escludono dal conteggio degli errori le variazioni della tensione di alimentazione e della resistenza del solenoide rispetto ai loro valori ideali. Esse rappresentano l'origine del valore non costante della corrente che, proprio a causa della sua non prevedibilità, si intende misurare. Si considerano perciò i seguenti contributi:

- Errore dovuto alla **resistenza di sense**:
  tale errore fornisce un contributo massimo (rispetto al valore massimo della corrente nel solenoide):

$$\Delta V_{R\_SENSE} = 1\% \cdot R_{SENSE} \cdot I_{\max} \cong 0.01 \cdot 0.2\Omega \cdot 645mA = 1.29mV$$

- Errori di condizionamento, ovvero:

  - errore di **offset** del Differential Voltage Sensor:
    facendo riferimento allo schema elettrico del componente (figura 6.11), si vede che esso è costituito da due amplificatori operazionali LM6142 in cascata, entrambi alimentati in single supply con 5 V. Dal datasheet del LM6142 [19] si legge una tensione di offset massima in ingresso:

$$V_{OS\_I,MAX} = 3.3mV$$

    Nel caso peggiore si avrà offset massimo in ingresso ad entrambi gli operazionali, ma, siccome il primo è configurato come voltage follower, si vedranno 3.3 mV alla sua uscita, e quindi:

$$\Delta V_I^{(1)} = 2 \cdot V_{OS\_I,MAX} = 6.6mV$$

  - errore di **guadagno** del Differential Voltage Sensor:
    il guadagno differenziale globale del componente 1B137, connettendo a massa il pin VREF come nel caso specifico di questo progetto (poiché non si vuole traslare la tensione di uscita), dipende da R24, R25, R26 e R27. Siccome ognuna di queste resistenze ha tolleranza 0.1%, l'errore totale di guadagno sarà:

$$\Delta A_{D,MAX} = A_D \cdot 4 \cdot 0.1\% = 10 \cdot 0.004 = 0.04$$

  - errore dovuto alla **corrente di polarizzazione** (Bias) degli operazionali:
    sapendo che ogni LM6142 ha una massima corrente I_b = 526 nA e che

$$V_{I,BIAS} = (R_+ - R_-) \cdot I_b$$

si può notare che solo il primo amplificatore vede tale tensione in ingresso (e quindi riportata alla sua uscita poiché ha guadagno unitario), dal momento che il secondo vede due resistenze uguali ai pin di ingresso.

$$\Delta V_I^{(2)} = (10.5K\Omega // 7.5K\Omega - 0\Omega) \cdot 526nA = 2.3mV$$

- errore dovuto alle **correnti di offset**:
  sul datasheet si legge che la massima corrente di offset in ingresso a ciascun operazionale vale 80 nA. Tenendo conto dei contributi di entrambi, si ottiene:

$$\Delta V_I^{(3)} = \left(\frac{10.5K\Omega // 7.5K\Omega}{2} \right) \cdot 80nA \cdot 1 + (100K\Omega // 1M\Omega) \cdot 80nA = 7.4mV$$

- Errore dell'ADC:
  come nel caso del magnetometro, utilizzando un ADC a 12 bit, questo errore vale:

$$\Delta V_{ADC} = 2 \cdot LSB[V] = 2 \cdot \frac{2.5V}{2^{12}} = 1.22mV$$

L'errore massimo risultante all'ingresso dell'ADC, quindi, vale:

$$\varepsilon_{SENSE}[mV] = \left(\Delta V_{R\_SENSE} + \Delta V_I^{(1)} + \Delta V_I^{(2)} + \Delta V_I^{(3)}\right) \cdot \left(A_D + \Delta A_{D,MAX}\right) + \Delta V_{ADC}$$
$$= (1.29mV + 6.6mV + 2.3mV + 7.4mV) \cdot (10 + 0.04) + 1.22mV = 177.8mV$$

Considerando il campo di misura [0~645] mA e il range di tensione [0~1.29] V in uscita dal circuito di condizionamento, tale errore si può esprimere in mA:

$$\varepsilon_{SENSE}[mA] = \frac{80.4mV \cdot 645mA}{1290mV} = 88.9mA$$

Tale errore di misura rappresenta il 13.8% del range considerato, ma, attraverso la calibrazione software, effettuata a terra, di guadagno ed offset per il valore in uscita dall'ADC, l'errore residuo si limita al contributo del convertitore stesso, ΔV_{ADC}, ed eventualmente alle derive di alcuni valori (come ad esempio le resistenze) con la temperatura, una volta in orbita.
Tuttavia, tali variazioni introducono contributi di errore in uscita il cui ordine di grandezza è confrontabile con quello dell'errore dell'ADC (<<1%).

6.2.5. Consumo

Il consumo di potenza complessivo del blocco ***Bk1B222_Magnetic_Torque_Actuator*** equivale alla somma dei seguenti contributi:

- P_{SOL}: potenza dissipata dal solenoide
- P_{DRV}: potenza dissipata dal driver A3953
- P_{SENSE}: potenza dissipata sul resistore di sense
- P_{DVS}: potenza dissipata dal Differential Voltage Sensor

Il consumo di potenza del solenoide vale:

$$P_{SOL} = V_{PDB,MAX} \cdot I_{\max} = 18V \cdot 0.645A = 11.61W$$

dove V_{PDB} è la tensione di alimentazione del solenoide (Power Distribution Bus).

Per quanto riguarda il driver A3953, dal suo datasheet si legge che la massima corrente di alimentazione della logica si ha durante il pilotaggio del carico e vale 50 mA, quindi:

$$P_{DRV} = 5V \cdot 50mA = 250mW$$

La resistenza R_{SENSE} dissipa una potenza massima pari a:

$$P_{SENSE} = {I_{\max}}^2 \cdot R_{SENSE} = (0.645A)^2 \cdot 0.2\Omega = 83mW$$

Il consumo di potenza del DVS equivale al doppio di quello di ogni singolo amplificatore LM6142. Dal datasheet di quest'ultimo, si legge che la massima corrente di alimentazione di ogni singolo operazionale, con tensione di alimentazione 5 V, è di 880 μA. Si ottiene:

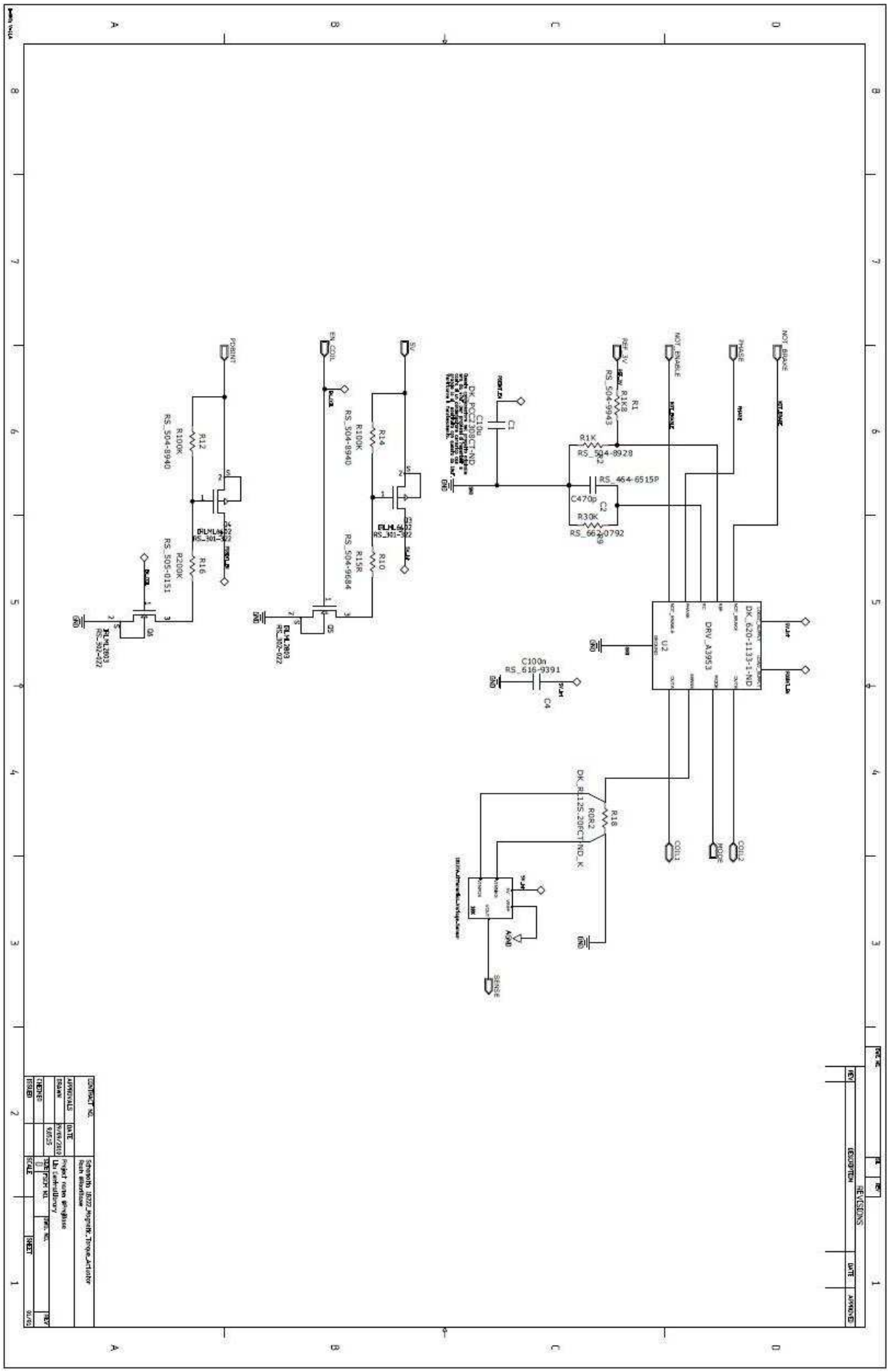
$$P_{DVS} = 2 \cdot 5V \cdot 880\mu A = 8.8mW$$

Infine, la potenza dissipata complessivamente dal blocco di attuazione magnetica risulta:

$$P_{TOT\_1B222} = P_{SOL} + P_{DRV} + P_{SENSE} + P_{DVS} =$$
$$= 11.61W + 0.25W + 0.083W + 0.0088W = 11.95W$$

Un consumo di potenza di questa portata potrebbe sembrare troppo elevato, ma si consideri che esso dipende principalmente dall'entità della corrente attraverso il solenoide. I restanti elementi del circuito, nel loro insieme, dissipano una potenza relativamente contenuta (342 mW).

6.2.6. Schema elettrico



6.3. Microcontrollore

Nel progetto del satellite AraMiS verranno utilizzati due microcontrollori per ogni Power Management Tile (1B8). Pertanto, tutti i moduli facenti parte della Power Management Tile condividono gli stessi microcontrollori, che dovranno quindi essere in grado di gestire tutti i canali in ingresso (digitali ed analogici) e in uscita (digitali). La presenza di due componenti, anziché uno, è dovuta al fatto che la somma di tutti i segnali richiesti dai diversi moduli contenuti in 1B8 eccede il numero massimo di segnali disponibili nel microcontrollore scelto per il progetto globale.

Si tenga presente, tuttavia, che ogni sottoblocco facente parte della tile andrà ad usufruire di un solo microprocessore. Quanti moduli siano collegati a quale microcontrollore esula dal ruolo di questa tesi ed è, peraltro, ancora in via di definizione, motivo per cui i numeri delle porte e dei bit, associati ai diversi segnali, non vengono definiti all'interno di 1B22 ma fanno riferimento alla classe *IOports_1B8* (cap. 4), che il progettista provvederà ad aggiornare coi valori definitivi.

Dal momento che il blocco 1B22, oggetto di questa tesi, utilizza parte delle uscite digitali del processore come segnali di controllo e tre ingressi analogici (ADC) per far acquisire al sistema i dati di campo magnetico (lungo le componenti x e y) e corrente nel solenoide, il microcontrollore dovrà avere, come minimo, le seguenti caratteristiche:

- 3 pin di uscita per controllare il magnetometro (segnali notSET, RESET, EN_MAGN)
- 5 pin di uscita per gestire l'attuatore magnetico (controlli NOT_ENABLE, NOT_BRAKE, MODE, PHASE del driver + il segnale EN_COIL)
- 3 canali per la conversione A/D
- 1 timer per la temporizzazione della chiamata di *housekeeping(index)*
- Livelli logici in uscita compatibili con quelli del driver A3953, degli switch sulle tensioni di alimentazione e del circuito di Set/Reset
- Basso consumo

Riassumendo, le caratteristiche minime richieste al microcontrollore, per la realizzazione del modulo 1B22, sono: 8 uscite digitali general purpose, 3 canali dell'ADC, 2 timer, livelli logici compatibili e basso consumo. Il soddisfacimento di quest'ultima caratteristica mediante una tensione di alimentazione a 3.3 V, garantisce automaticamente la compatibilità coi livelli logici del circuito di Set/Reset (anch'esso alimentato a 3.3 V) e del driver (si veda il datasheet dell'A3953); per quanto riguarda gli switch sulle tensioni di alimentazione (visibili negli schemi elettrici), che in questo progetto sono comandati dai segnali EN_MAGN ed EN_COIL, sono stati precedentemente progettati, globalmente, per poter dare o togliere l'alimentazione a qualsiasi sottoblocco della Power Management Tile. Essi sono dimensionati a seconda che si tratti della tensione a 5 V o quella del PDB, e sono sempre compatibili coi livelli logici in uscita dal controllore, che saranno $V_{OH} \approx 3.3$ V e $V_{OL} \approx 0$ V.

Tenendo conto dei suddetti requisiti e di quelli degli altri moduli componenti il progetto 1B8, il microcontrollore che si è scelto di utilizzare per l'intera tile è il modello MSP430F5437 della Texas Instruments [20], tra le cui caratteristiche principali si trovano:

- Tensione di alimentazione 2.2~3.6 V
- Architettura RISC a 16 bit
- ADC a 12 bit e 14 canali esterni + 2 interni
- 2 timer (Timer_A e Timer_B)
- 4 Universal Serial Communication Interfaces (USCI)

- Velocità di clock fino a 18 MHz
- Program Memory 256 KB Flash
- RAM 16 KB
- Low Power
- Oscillatore interno
- Connettività I²C, IrDA, LIN, SCI, SPI, UART/USART
- 67 I/O pins, package 80-LQFP
- Temperature supportate [-40 ~ 85]°C

Nella figura sottostante, si può osservare la piedinatura del componente MSP430F5437. Poiché l'assegnazione dei pin non è ancora stata definita e tale modello di microcontrollore non è momentaneamente disponibile, in questo capitolo non è stato riportato lo schema illustrante l'interfaccia tra i medesimi e i segnali in ingresso/uscita dal modulo 1B22. Questi ultimi sono visibili nello schema a blocchi globale del par. 6.4, mentre la loro corrispondenza con i segnali del microcontrollore, anche se non ancora in versione definitiva, è altresì indispensabile per la comprensione della routine di collaudo (cap. 7), pertanto verrà mostrata nel prossimo capitolo, insieme ad alcuni dettagli sull'utilizzo del microcontrollore.

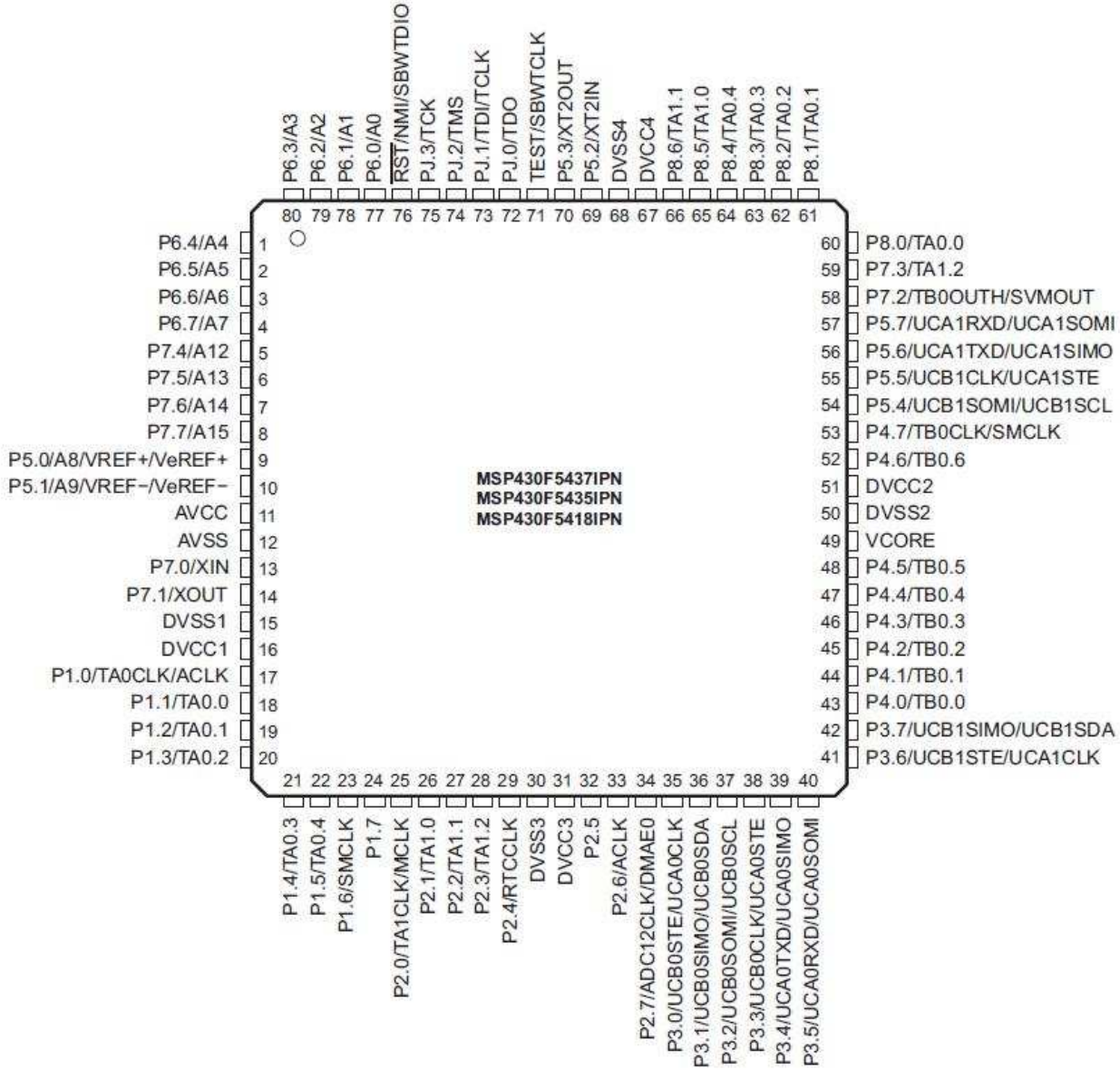
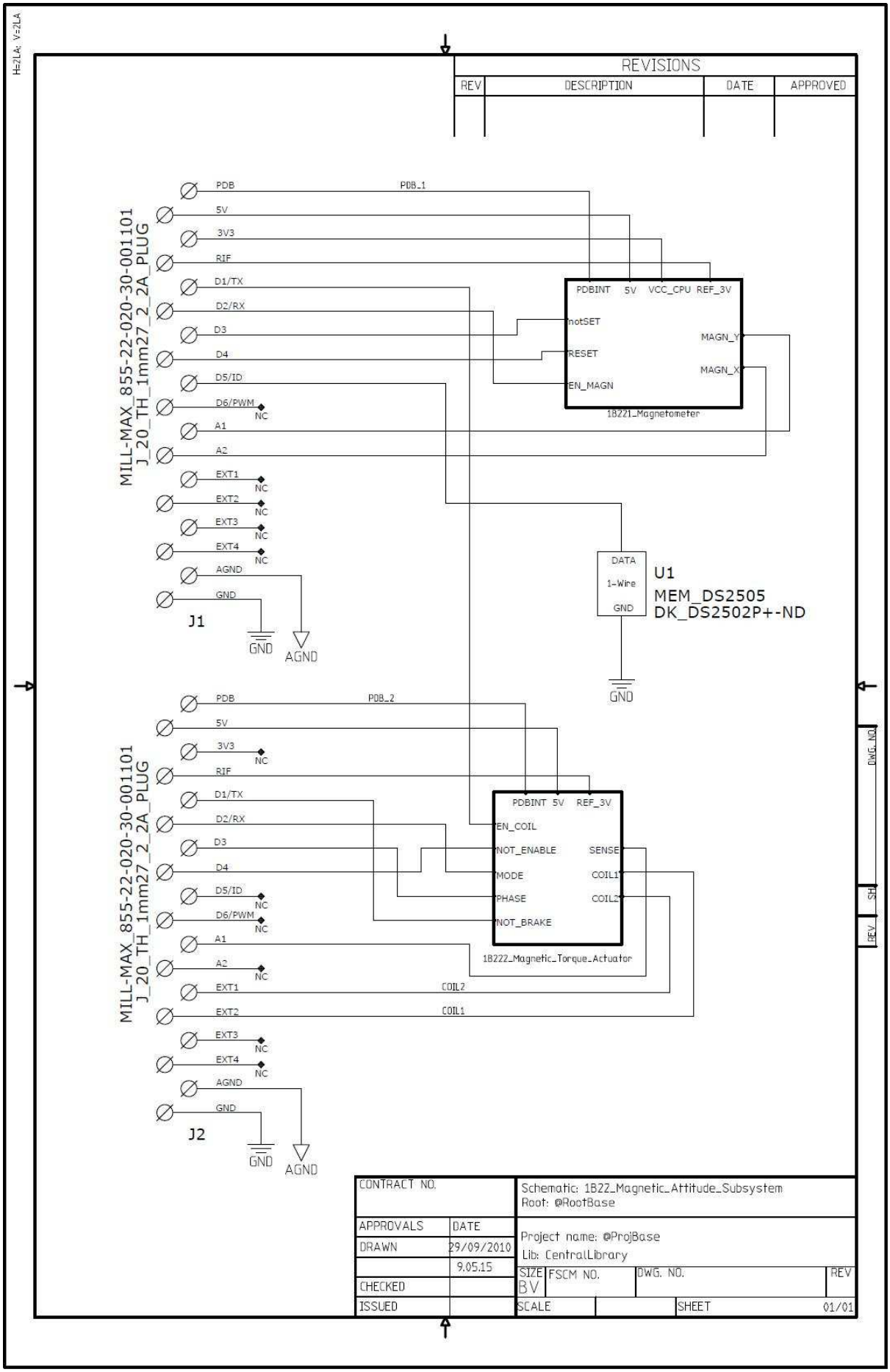


Figura 6.12 Piedinatura microcontrollore MSP430F5437

6.4. Schema globale



Capitolo 7

7. Testing - Collaudo e misure

In questo capitolo sono riportate le procedure di collaudo per l'intero sottosistema di controllo dell'assetto magnetico del satellite AraMiS, ovvero il progetto 1B22, oggetto di questa tesi.
Parallelamente alla descrizione di tali routine di test, sono riportati i risultati ottenuti eseguendo il collaudo sul circuito finale, visibile in figura 7.1. Esso rappresenta il prototipo del modulo 1B22, incluso nella Power Management Tile (si veda il cap. 1).
Il PCB del circuito stampato (appendice A), sul quale sono stati montati tutti i componenti hardware del progetto, è stato realizzato con il tool Expedition PCB della Mentor Graphics, a partire dagli schemi elettrici illustrati nel capitolo precedente.

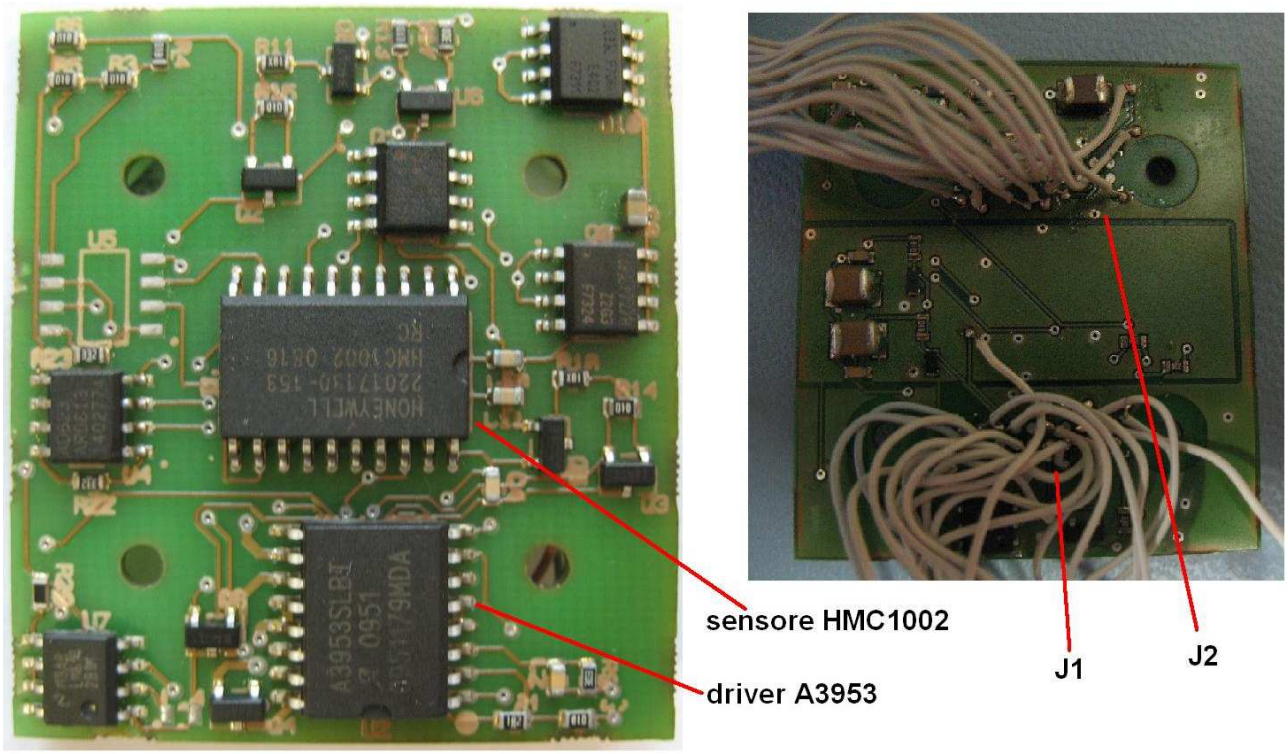


Figura 7.1 Foto del modulo 1B22: lato top (a sinistra) e lato bottom (a destra)

Dapprima è stato eseguito un collaudo basilare dell'hardware, fornendo le tensioni d'alimentazione al circuito e verificandone il funzionamento, ad esempio, mediante la misura delle tensioni sui piedini dei componenti.
In una seconda fase, è stata collaudata la sensoristica, misurando le tensioni in uscita dai circuiti di condizionamento per le componenti x e y del campo magnetico ed il valore di corrente nel solenoide.
Successivamente, si è proceduto al test delle funzioni che implementano il controllo d'assetto magnetico, ovvero la parte software del progetto, applicata, però, all'hardware realizzato.

Come è stato detto nel sottocapitolo 6.3, il microcontrollore scelto per il progetto della Power Management Tile di AraMiS è il modello MSP430F5437 della Texas Instruments. Tuttavia, questo componente non è attualmente disponibile in magazzino. Pertanto, per il collaudo del modulo 1B22 è stato utilizzato il modello MSP430FG439 [21], avente caratteristiche similari al primo, e le funzioni di configurazione e gestione di timer, ADC e piedini I/O sono state adattate a quest'ultimo. Il codice di queste funzioni non è stato riportato nel progetto software (cap. 5) in quanto esse sono comuni al progetto globale e quindi non fanno parte del modulo 1B22, ma sono state descritte mediante la loro documentazione nel cap. 4, se utilizzate dal sistema di controllo dell'assetto magnetico.
La comunicazione tra il microcontrollore ed il circuito di figura 7.1, da una parte, e il computer sul quale viene eseguito il programma di debug del software di progetto (incluse, chiaramente, le funzioni di test), dall'altra, avviene grazie al circuito adattatore *Surface Connector Adapter* (figura 7.2). Gli schemi elettrici dell'adattatore e dei suoi sottoblocchi sono visibili in appendice B.

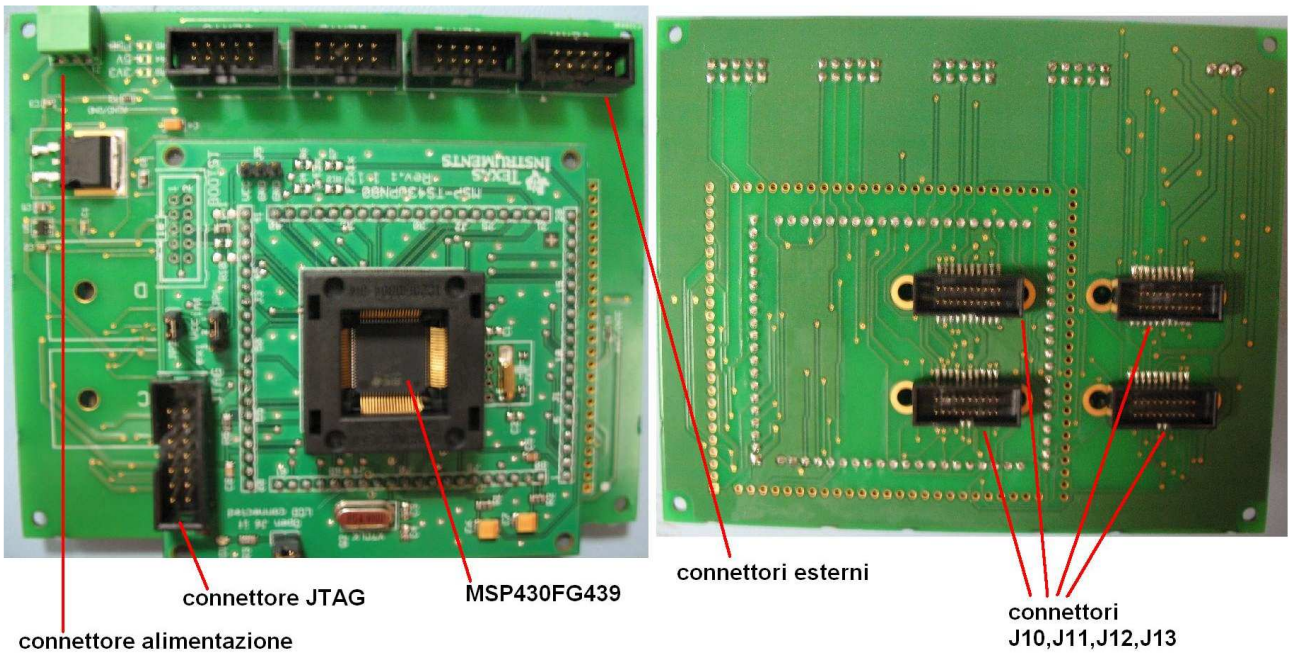


Figura 7.2 Foto dell'adattatore per MSP430: lato top (a sinistra) e lato bottom (a destra)

I connettori J1 e J2 del modulo 1B22 (si veda lo schema del par. 6.4) vengono collegati, rispettivamente, ai connettori J10 e J11 dell'adattatore (schema in B.1), sul quale è montato il microcontrollore (schema in B.2). Anche la tensione d'alimentazione PDB viene portata direttamente sull'adattatore (per i test è stata utilizzata una tensione di 14 V, eccetto dove diversamente specificato). A partire da quest'ultima vengono generate le tensioni 5 V, 3.3 V ed il riferimento di tensione a 3 V (schema in B.3). L'adattatore dispone, inoltre, di connettori esterni. Su uno di essi (J7 nello schema di appendice B.4) verrà collegato il solenoide.
Per caricare il software sul microcontrollore ed eseguire le funzioni di collaudo, esso viene collegato tramite interfaccia JTAG con il programmatore MSP-FET430UIF [22] della Texas Instruments, a sua volta connesso al pc tramite porta USB. Il tool utilizzato per il debug è IAR Embedded Workbench.

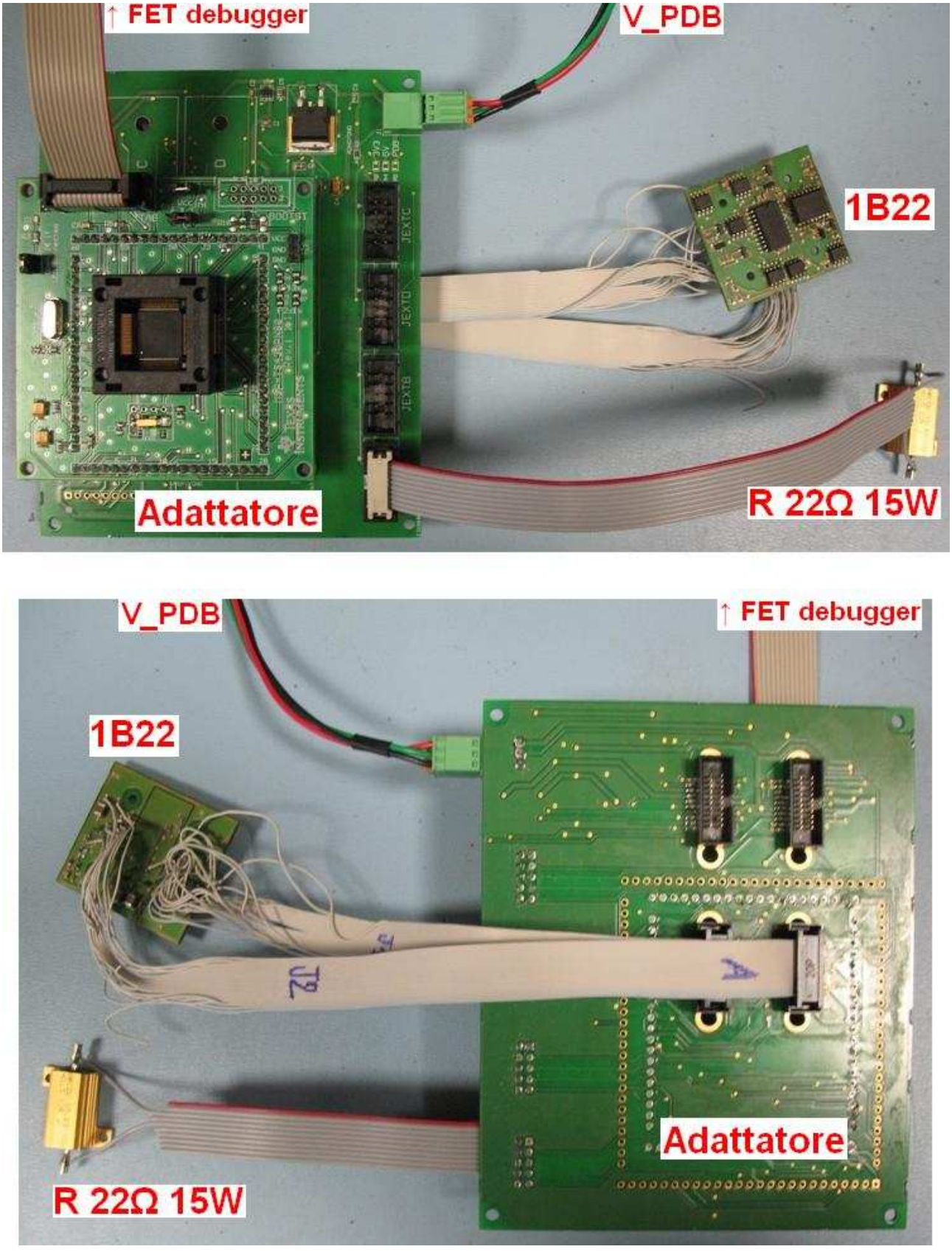


Figura 7.3 Foto del sistema complessivo di collaudo: vista da sopra (in alto) e vista da sotto (in basso)

Tutte le procedure appena presentate, per la cui descrizione dettagliata si rimanda al seguito di questo capitolo, consistono nel collaudare la corretta messa in atto di tutte le specifiche del sistema, illustrate in UML nel capitolo 3 sotto forma di casi d'uso, o, in altre parole, nell'implementare e testare questi ultimi.
Pertanto, analogamente ai capitoli 3, 4 e 5, si è deciso di utilizzare gli strumenti e le rappresentazioni visuali offerti dal linguaggio UML anche per descrivere le procedure svolte in fase di test, nonché le conseguenti risposte del circuito in esame.
A tal proposito, è stato redatto un diagramma dei casi d'uso specifico per il collaudo, visibile in figura sottostante, che descrive le procedure di test effettuate.
Facendo riferimento a quanto scritto sopra, i test di base dell'hardware verranno illustrati tramite le tabelle contenute nella documentazione dei relativi casi d'uso.
I test delle routine di controllo vere e proprie, invece, sono stati implementati con apposite funzioni C++, contenute nel diagramma delle classi **1B22-K GSE – Testing**, in qualità di metodi delle classi di testing, descritte dettagliatamente nel sottocapitolo 7.2, dove verrà riportato anche il codice di queste funzioni, contenente anche, sotto forma di commenti, la descrizione delle procedure di collaudo sw ed il resoconto delle risposte del sistema.

7.1. Documentazione dei Casi d'Uso

Name	*Documentation*
::Test Magnetometer	
::Test Coil Driver	
::Test supply currents	
::Test Housekeeping structure	Testing basic functionality of magnetic field and coil current acquiring. • Magnetometer block, even if enabled in configuration word t_ConfigWord_1B2, has to be supplied (supply voltage is enabled/disabled by signal EN_MAGN) just for the time necessary to acquire MAGNETIC_FIELD_X_ADC and MAGNETIC_FIELD_Y_ADC channels of ADC to reduce power consumption, so EN_MAGN signal, initially reset, must be set at MAGNETIC_FIELD_X-2 index and reset again at MAGNETIC_FIELD_Y+1, for every housekeeping cycle • Magnetic Coil, on the other hand, mustn't operate when system is acquiring magnetic field (if operating, it may heavily affect measure because of its disturbing field), so EN_COIL signal, if previously set, is reset at MAGNETIC_FIELD_X-2 index and set again at MAGNETIC_FIELD_Y+2, in order that coil current value can be acquired at COIL_CURRENT index Making use of test_housekeeping function of Bk1B22_Software_Tests class, test consists in forbidding supervise function (of Bk1B229_Controller class) to execute, forcing coil on and then start housekeeping cycle. Observing EN_COIL and EN_MAGN signals waveforms, has been verified that these signals respect the conditions above on two consecutive housekeeping cycles, as we can see in related pictures (7.6, 7.7, 7.8). In the second cycle Magnetic Actuator is not active; it has been disabled at EOC_INTEGRAL index of first cycle because angular momentum is zero.
::Test SetReset sequence	Testing the correctness of Set/Reset sequence in the Magnetic Attitude Sybsystem, using test_setreset function of Bk1B22_Software_Tests class. It tests both SetReset Magnetometer and System SR Magnetometer use cases. The S/R procedure is launched by calling the *interpret*(CMD_SETRESET_MAGNETOMETER) operation. To verify the occurrence of system Set/Reset automatic enabling, *housekeeping*(SETRESET_MAGN) test has been used.
::Test Coil Current Measuring	
::Test Coil Driver Enable	
::Test Magnetometer Enable	
::Test Magnetic Field Measuring	
::Test Interpret	Testing if interpret operation of Bk1B229_Controller class works with commands CMD_SET_CONFIGURATION, CMD_RESET_CONFIGURATION of t_Commands class. It makes use of test_interpret function of Bk1B22_Software_Tests class. Commands CMD_ACTUATE_COIL and

	CMD_SETRESET_MAGNETOMETER of t_Commands_1B8 class are specifically tested by Test Coil Actuating and Test SetReset sequence use cases, respectively.
::Test interpret(CMD_SET_CONFIGURATION)	See test_cmd_setconfig function of Bk1B22_Software_Tests class.
::Test interpret(CMD_RESET_CONFIGURATION)	See test_cmd_rstconfig function of Bk1B22_Software_Tests class.
::Test Coil Actuating	Tests Actuate Coil use case of Magnetic Attitude Subsystem in four cases: generation of a small angular momentum (1000 g*cm^2/s) using test_actuate_coil_1 function of Bk1B22_Software_Tests class; generation of a bigger momentum (30000 g*cm^2/s) using test_actuate_coil_2 function of Bk1B22_Software_Tests class; generation of a negative momentum using test_actuate_coil_3 function of Bk1B22_Software_Tests class; coil deactivation by a zero angular momentum request, using test_actuate_coil_4 function of Bk1B22_Software_Tests class. In all cases, main steps are the following: • enable Bk1B221_Magnetometer_Sensor and Bk1B222_Magnetic_Torque_Actuator blocks, so it is possibile to drive coil and get factors to calcultae magnetic torque integral (Bx, By, coil current) • write a 2-byte message into a buffer of Buffers class of Bk1B45_Slave package, containing the required angular momentum • start TimerA which automatically calls housekeeping operation of Bk1B229_Controller class • simulate that a CMD_ACTUATE_COIL command of t_Commands_1B8 class of Bk1A_1B8_Codes package has arrived passing this command to interpret operation of Bk1B229_Controller class: command will be decoded and buffer content will be read and saved into actual_L attribute of Bk1B229_Controller class • observe that, every housekeeping cycle, **actual_L** value is decreased by the correct amount, which is the actual value of magnetic torque (see related functions)
::Test Acquiring	Testing magnetic field and coil current acquiring. This use case is implemented by test_acquiring function of Bk1B22_Software_Tests class. The test process consists in initializing system by calling boot operation of Bk1B229_Controller class, then enabling Bk1B221_Magnetometer_Sensor and Bk1B222_Magnetic_Torque_Actuator blocks and activating coil. Finally TimerA is started to automatically call housekeeping operation of Bk1B229_Controller class and elements corresponding to MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y and COIL_CURRENT positions of Housekeeping.housekeeping vector are printed at the end of acquiring cycle. These values are produced by the following sequence of events: MAGN_X and MAGN_Y output voltages of Bk1B221_Magnetometer_Sensor circuit and SENSE output voltage of Bk1B222_Magnetic_Torque_Actuator are acquired, respectively, by MAGNETIC_FIELD_X_ADC, MAGNETIC_FIELD_Y_ADC and COIL_CURRENT_ADC ADC channels and converted into digital format. A/D conversion results are then scaled by using parameters of t_scaling type of Bk1A1_Common_Codes package. These parameters are stored in positions MAGNETIC_FIELD_X_SCALE,

	MAGNETIC_FIELD_Y_SCALE, COIL_CURRENT_SCALE of Housekeeping.scaling vector. Correctness of magnetic field components and coil current housekeeping values has been verified (see test_acquiring).
::Test Supervision	Testing the Supervision use case of Magnetic Attitude Subsystem, using test_supervise function of Bk1B22_Software_Tests class. It tests the correct execution of supervise operation of Bk1B229_Controller class, checking if errors ON_CURR_EXCESS, ON_CURR_LACK, OFF_CURR_EXCESS, ERR_COIL_ENABLE, ERR_COIL_STATUS, ERR_MAGN_ENABLE of Housekeeping.housekeeping[MAGNETIC_ERRORS] error word are correctly found. It has been verified that also LOW_PDB_VOLT error is found, because there is no PDB voltage acquiring in the tested system, so it detects a 0 V PDB voltage and reports it as wrong. ERR_COIL_ATTACH and ERR_MAGN_ATTACH testing was not possible because bits HAS_MAGNETIC_COIL and HAS_MAGNETOMETER of class t_MechanicalConfiguration_1B8 cannot be modified anyway by Bk1B22_Magnetic_Attitude_Subsystem. However, they are both set to 'true' by default.

Use Case Description : Test Magnetometer Enable

	Actor Input	*System Response*
1	Connect 1B22 to power supplies (14 V, 5 V, REF_3V, CPU_VCC)	
2	Measure voltages before power switches	
3		PDB is ok (14 V)
4		REF_3V is ok
5		5V ok
6		CPU_VCC is ok (3.3 V)
7	Connect EN_MAGN to GND	
8	Measure voltages on sensor pins	
9		Vbridge is 0 V (both A and B)
10	Measure voltages on REF02 input	
11		INPUT is 0 V
12	Measure voltages on AD623 pins	
13		Vs+ is 0 V
14		REF is ok (1.5 V cause REF_3V isn't switched)
15	Measure S/R circuit supply	
16		3V3 is ok (3.3 V, not switched)
17	Connect EN_MAGN to CPU_VCC	
18	Measure voltages on sensor pins	
19		Vbridge is 5 V (both A and B)
20	Measure voltages on REF02 input	
21		INPUT is 14 V

22	Measure voltages on AD623 pins	
23		Vs+ is 5 V
24		REF is 1.5 V
25	Measure voltage on MAGN_X output	
26	Measure voltage on MAGN_Y output	
27		MAGN_X varies
28		MAGN_Y varies

Use Case Description : Test Magnetometer

	Actor Input	*System Response*
1	Apply power supply	
2	Connect EN_MAGN to GND	
3		Outputs 0.989 V on MAGN_X output
4		Outputs 0.987 V on MAGN_Y output
5	Measure voltage on MAGN_X output	
6	Measure voltage on MAGN_Y output	
7	Connect EN_MAGN to CPU_VCC	
8	Rotate sensor	
9		Voltage on MAGN_X output varies
10		Voltage on MAGN_Y output varies
11	Measure voltage on MAGN_X output	
12	Measure voltage on MAGN_Y output	
13	Move a magnet close to sensor	
14		Voltage on MAGN_X output varies
15		Voltage on MAGN_Y output varies
16	Measure voltage on MAGN_X output	
17	Measure voltage on MAGN_Y output	
18	Remove power supply	

Use Case Description : Test Magnetic Field Measuring

	Actor Input	*System Response*
1	Interface PC to 1B22 module using MSP430 USB Debug Interface and MSP430 adapter (including microcontroller)	
2	Apply correct supply voltages to adapter (14 V)	
3	Set EN_MAGN pin to 'true'	
4		MAGN_X output depends on magnetic field

5	Find maximum MAGN_X value	
6		MAGN_X output is X_max
7	Measure voltage on MAGN_X output, X_max = 1.35 V	
8	Rotate sensor 180°	
9		MAGN_X output is X_min
10	Measure voltage on MAGN_X output, X_min = 0.87 V	
11	Repeat this procedure for channel MAGN_Y	
12		MAGN_Y output depends on magnetic field
13	Measure voltage on MAGN_Y output: when maximum is found, Y_max = 1.45 V; after a 180° sensor rotation, Y_min = 0.96 V	
14	Calculate magnetic field: B [V] = (X_max - X_min)/2 = 0.24 V but also B[V] = (Y_max - Y_min)/2 = 0.245 V; the two values are compatible. B[G] = B[V]/SENS_MAGNETIC[V/G] = 0.479 ÷ 0.489 G	

Use Case Description : Test Coil Driver Enable

	Actor Input	***System Response***
1	Connect 1B22 to power supplies (14 V, 5 V, REF_3V)	
2	Measure voltages before power switches	
3		PDB is ok (14 V)
4		REF_3V is ok
5		5V ok
6	Connect EN_COIL to GND	
7	Measure voltages on driver pins	
8		LOGIC SUPPLY is 0 V
9		REF is ok (0.23 V, not switched)
10		LOAD SUPPLY is wrong, there is still 10 V, decreasing if connected to ground (through the voltmeter)
11	Connect EN_COIL to CPU_VCC	
12	Measure voltages on driver pins	
13		LOGIC SUPPLY is 5 V
14		REF is ok
15		LOAD SUPPLY is 14 V

16	Remove power supplies	

Use Case Description : Test Coil Driver

	Actor Input	*System Response*
1	Interface PC to 1B22 module using MSP430 USB Debug Interface and MSP430 adapter (including microcontroller)	
2	Apply 14 V voltage to MSP adapter	
3	Set EN_COIL pin to 'false'	
4	Set NOT_ENABLE pin to 'true'	
5	Set NOT_BRAKE pin to 'true'	
6	Set MODE pin to 'true'	
7	Set PHASE pin to 'true'	
8	Connect a 22 ohm 15 W resistor to COIL1 and COIL2 pins to simulate Bk1B2222_Coil	
9		there is no current flowing through resistor
10	Measure current flowing through resistor	
11	Set EN_COIL pin to 'true'	
12	Set NOT_ENABLE pin to 'false'	
13		current flows through resistor
14	Measure current flowing through resistor: current value is between 0.6 A and 0.61 A (correct)	

Use Case Description : Test Coil Current Measuring

	Actor Input	*System Response*
1	Interface PC to 1B22 module using MSP430 USB Debug Interface and MSP430 adapter (including microcontroller)	
2	Apply 14 V voltage to MSP adapter	
3	Set EN_COIL pin to 'true'	
4	Set NOT_BRAKE pin to 'true'	
5	Set MODE pin to 'true'	
6	Set PHASE pin to 'true'	
7	Set NOT_ENABLE pin to 'false'	
8		SENSE output is proportional to coil current
9	Measure voltage on SENSE output: V_SENSE = 1.216 V (correct because	

	global current sensing sensitivity is 2 V/A)	

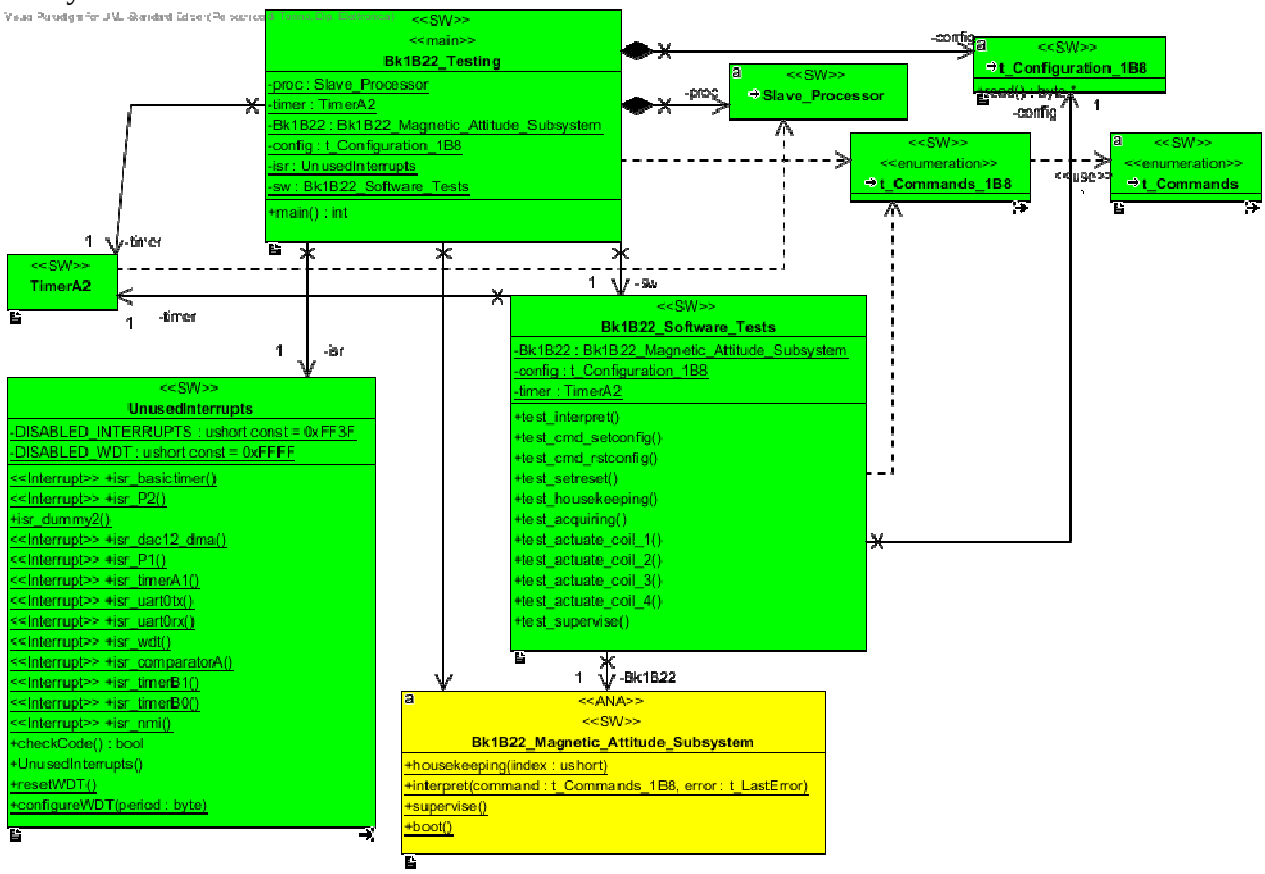
Use Case Description : Test supply currents

	Actor Input	*System Response*
1	Interface PC to MSP430 adapter (including microcontroller) using MSP430 USB Debug Interface	
2	Set power supply to 14 V	
3	Put a multimeter in series to the power supply (amperometer configuration)	
4	Apply power supply to MSP adapter (as PDB typical supply)	
5		Current is 5.16 mA
6	Measure current absorbed by MSP430 adapter only	
7	Connect 1B22 module to adapter	
8	Disable magnetometer block setting EN_MAGN pin to 'false'	
9	Disable magnetic actuator block setting EN_COIL pin to 'false'	
10		Current is 8.39 mA
11	Measure current absorbed by adapter plus 1B22 module with both blocks disabled	
12	Calculate value of current absorbed by 1B22 module with both blocks disabled	
13		1B22 quiescent absorbed current, due to parasite effects and faulty power switch controlled by EN_COIL signal, is (8.39 - 5.16)mA = 3.23 mA
14	Enable magnetometer block only, setting EN_MAGN pin to 'true'	
15		Current is 21.2 mA
16	Measure current absorbed by operating magnetometer plus adapter	
17	Calculate value of current absorbed by magnetometer only	
18		Result is (21.2 - 8.39) mA = 12.81 mA
19	Disable magnetometer block setting EN_MAGN pin to 'false'	
20	Set magnetic coil driver in standby mode -> NOT_ENABLE = ‘true’, NOT_BRAKE = ‘true’, MODE = ‘false’ (PHASE doesn’t care)	
21	Enable magnetic actuator block only,	

	setting EN_COIL pin to ‘true’	
22		Current is 21.5 mA
23	Measure current absorbed by coil driver in standby mode plus adapter	
24	Calculate current absorbed by coil driver in standby mode	
25		13.11 mA
26	Set magnetic coil driver in sleep mode by switching MODE to ‘true’	
27		Current is 10.1 mA
28	Measure current absorbed by coil driver in sleep mode plus adapter	
29	Calculate current absorbed by coil driver in sleep mode	
30		1.71 mA
31	Put the amperometer in series to resistor that simulates coil	
32	Set magnetic coil driver in forward driving mode with fast current decay -> PHASE = ‘true’, NOT_ENABLE = ‘false’	
33		Current is 0.59 A
34	Measure current absorbed by coil driver in driving mode (driver logic supply current, in this case, is inconsiderable)	

7.2. Bk1B22-K GSE - Testing

This class diagram defines testing structures and routines for Bk1B22_Magnetic_Attitude_ Subsystem.



7.2.1. Class Bk1B22_Testing

This is the main testing class for Bk1B22_Magnetic_Attitude_Subsystem. It contains main sw testing routine used on Slave_Processor.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
proc (private) Attribute that represents the Slave_Processor class of Bk1B45_Slave package.	Bk1B45_Slave.Slave_Processor	
timer (private)	Bk1B22K_GSE$TimerA2	
Bk1B22 (private)	Bk1B22_Magnetic_Attitude_Subsystem	
config (private)	Bk1A_1B8_Codes$t_Configuration_1B8	

isr (private)	UnusedInterrupts	
sw (private)	Bk1B22_Software_Tests	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
main (public) Main sw testing routine for Bk1B22_Magnetic_Attitude_Subsystem project. It makes use of operations included in class Bk1B22_Software_Tests.		int

7.2.2. Class Bk1B22_Software_Tests

This class contains all sw testing routines that are used by main operation of class Bk1B22_Testing.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
Bk1B22 (private)	Bk1B22_Magnetic_Attitude_Subsystem	
config (private)	Bk1A_1B8_Codes$t_Configuration_1B8	
timer (private)	Bk1B22K_GSE$TimerA2	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
test_interpret (public)		
test_cmd_setconfig (public)		
test_cmd_rstconfig (public)		
test_setreset (public)		
test_housekeeping (public)		
test_acquiring (public)		
test_actuate_coil_1 (public)		
test_actuate_coil_2 (public)		
test_actuate_coil_3 (public)		
test_actuate_coil_4 (public)		
test_supervise (public)		

Bk1B22_Testing.cpp

```
#include "Bk1B22_Testing.h"

#include "TimerA2.h"
#include "Bk1B22_Magnetic_Attitude_Subsystem.h"
#include "t_Configuration_1B8.h"
#include "UnusedInterrupts.h"
#include "Bk1B22_Software_Tests.h"
#include "Slave_Processor.h"

int main() {
    /**
     * Attribute that represents the Slave_Processor class of Bk1B45_Slave package.
     */
    static Bk1B45_Slave::Slave_Processor proc;
    static TimerA2 timer;
    static Bk1B22_Magnetic_Attitude_Subsystem Bk1B22;
    static t_Configuration_1B8 config;
    static UnusedInterrupts isr;
    static Bk1B22_Software_Tests sw;

    static Common::ADC adc;    //include ADC

    //configuring digital outputs on MSP430
    *(Bk1B22.IOports.Bk1B221_NOT_SET_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B221_NOT_SET_BIT;
    *(Bk1B22.IOports.Bk1B221_RESET_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B221_RESET_BIT;
    *(Bk1B22.IOports.Bk1B221_EN_MAGN_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B221_EN_MAGN_BIT;
    *(Bk1B22.IOports.Bk1B222_EN_COIL_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B222_EN_COIL_BIT;
    *(Bk1B22.IOports.Bk1B222_NOT_ENABLE_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B222_NOT_ENABLE_BIT;
    *(Bk1B22.IOports.Bk1B222_MODE_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B222_MODE_BIT;
    *(Bk1B22.IOports.Bk1B222_PHASE_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B222_PHASE_BIT;
    *(Bk1B22.IOports.Bk1B222_NOT_BRAKE_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B222_NOT_BRAKE_BIT;
    *(Bk1B22.IOports.Bk1B22_CALIBRATION_PORT + (&P1DIR-&P1OUT)) |=
Bk1B22.IOports.Bk1B22_CALIBRATION_BIT;

    //configuring ADC inputs on MSP430
    P6SEL |= 1<< Bk1B22.MAGNETIC_FIELD_X_ADC;
    P6SEL |= 1<< Bk1B22.MAGNETIC_FIELD_Y_ADC;
    P6SEL |= 1<< Bk1B22.COIL_CURRENT_ADC;

    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;
    __enable_interrupt();

        sw.test_interpret();
        sw.test_setreset();
        sw.test_housekeeping();
        sw.test_acquiring();
        sw.test_actuate_coil_1();
        sw.test_actuate_coil_2();
        sw.test_actuate_coil_3();
        sw.test_actuate_coil_4();
        sw.test_supervise();

    while(1);

}
```

Bk1B22_Testing.h

```
#pragma diag_suppress=Pa050
#include <intrinsics.h>

#ifndef __Bk1B22_Testing_h__
#define __Bk1B22_Testing_h__

#include "platform.h"

#include "TimerA2.h"
#include "Bk1B22_Magnetic_Attitude_Subsystem.h"
#include "t_Configuration_1B8.h"
#include "UnusedInterrupts.h"
#include "Bk1B22_Software_Tests.h"
#include "Slave_Processor.h"

class TimerA2;
class Bk1B22_Magnetic_Attitude_Subsystem;
class t_Configuration_1B8;
class UnusedInterrupts;
class Bk1B22_Software_Tests;
class Bk1B22_Testing;
namespace Bk1B45_Slave
{
   class Slave_Processor;
}

#endif
```

Bk1B22_Software_Tests.cpp

```
#include "Bk1B22_Software_Tests.h"

#include "Bk1B22_Magnetic_Attitude_Subsystem.h"
#include "t_Configuration_1B8.h"
#include "TimerA2.h"

Bk1B22_Magnetic_Attitude_Subsystem Bk1B22_Software_Tests::Bk1B22;
t_Configuration_1B8 Bk1B22_Software_Tests::config;
TimerA2 Bk1B22_Software_Tests::timer;

void Bk1B22_Software_Tests::test_interpret() {
          test_cmd_setconfig();
          test_cmd_rstconfig();
}

void Bk1B22_Software_Tests::test_cmd_setconfig() {
       config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);
   //magnetometer is detached
       config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
   //actuator is detached

//monitor EN_MAGN and EN_COIL supply enable signals using oscilloscope

Bk1B22.sensor.EN_MAGN(true);      //give power supply to magnetometer block
Bk1B22.actuator.EN_COIL(true); //give power supply to magnetic actuator block

//verify that signals are 'true'
//EN_MAGN is 'true'
//EN_COIL is 'true'

//launch SET_CONFIGURATION command

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A1_Common_Codes::CMD_SET_CONFIG
URATION, (t_LastError)0);

//verify that signals are 'false'
//EN_MAGN is 'false'
//EN_COIL is 'false'
//correct: if elements are not attached, system disables power supplies

config.config_1B2.set(config.config_1B2.ATTACH_MAGNETOMETER);   //magnetometer is
attached
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);   //actuator is at-
tached
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);   //magnetometer is
not enabled
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);   //actuator is
not enabled

Bk1B22.sensor.EN_MAGN(true);      //give power supply to magnetometer block
```

```
Bk1B22.actuator.EN_COIL(true);    //give power supply to magnetic actuator block
//EN_MAGN and EN_COIL are 'true'

//launch SET_CONFIGURATION command

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A1_Common_Codes::CMD_SET_CONFIG
URATION, (t_LastError)0);

//verify that signals are 'false'
//EN_MAGN is 'false'
//EN_COIL is 'false'
//correct: if elements are attached but not enabled, system disables power supplies

config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);   //magnetometer is
enabled
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);   //actuator is en-
abled

Bk1B22.sensor.EN_MAGN(true);      //give power supply to magnetometer block
Bk1B22.actuator.EN_COIL(true);    //give power supply to magnetic actuator block
//EN_MAGN and EN_COIL are 'true'

//launch SET_CONFIGURATION command

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A1_Common_Codes::CMD_SET_CONFIG
URATION, (t_LastError)0);

//verify that signals are 'true'
//EN_MAGN is 'true'
//EN_COIL is 'true'
//correct: if elements are attached and enabled, system does not disable power sup-
plies

}

void Bk1B22_Software_Tests::test_cmd_rstconfig() {
          config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);
//magnetometer is detached
          config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
//actuator is detached

//monitor EN_MAGN and EN_COIL supply enable signals using oscilloscope

Bk1B22.sensor.EN_MAGN(true);      //give power supply to magnetometer block
Bk1B22.actuator.EN_COIL(true);    //give power supply to magnetic actuator block

//verify that signals are 'true'
//EN_MAGN is 'true'
//EN_COIL is 'true'

//launch RESET_CONFIGURATION command

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A1_Common_Codes::CMD_RESET_CONF
IGURATION, (t_LastError)0);

//verify that signals are 'false'
//EN_MAGN is 'false'
//EN_COIL is 'false'
//correct: if elements are not attached, system disables power supplies

config.config_1B2.set(config.config_1B2.ATTACH_MAGNETOMETER);   //magnetometer is
attached
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);   //actuator is at-
tached
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);   //magnetometer is
not enabled
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);   //actuator is
not enabled

Bk1B22.sensor.EN_MAGN(true);      //give power supply to magnetometer block
Bk1B22.actuator.EN_COIL(true);    //give power supply to magnetic actuator block
//EN_MAGN and EN_COIL are 'true'

//launch RESET_CONFIGURATION command

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A1_Common_Codes::CMD_RESET_CONF
IGURATION, (t_LastError)0);

//verify that signals are 'false'
//EN_MAGN is 'false'
//EN_COIL is 'false'
//correct: if elements are attached but not enabled, system disables power supplies

}
```

```
void Bk1B22_Software_Tests::test_setreset() {
// !!! first of all, suppress supervise() operation calling in housekeep-
ing(SETRESET_MAGN) of Bk1B229_Controller.cpp, cause it might configure magnetometer
as disabled
//monitor notSET and RESET signals using oscilloscope
//monitor SET_ENABLE, SET_FLAG and RESET_FLAG flags using debug tool

//launch boot() operation, which resets SET_ENABLE, SET_FLAG and RESET_FLAG flags
and initializes notSET and RESET signals to 'true' (as suggested in HMC1002 sensor
datasheet)
Bk1B22.boot();

//notSET and RESET signals are 'true'
//SET_ENABLE, SET_FLAG and RESET_FLAG flags are 'false'

config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);   //enable magnetome-
ter block

timer.start();   //launch TimerA2 which periodically calls housekeeping(index)

//launch Set/Reset command

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A_1B8_Codes::CMD_SETRESET_MAGNE
TOMETER, NO_ERROR);

//SET_ENABLE, SET_FLAG and RESET_FLAG flags do respect the correct sequence
//after properly debugging, with oscilloscope, trigger on RESET signal falling edge
and observe waveforms (see pictures 7.4 and 7.5)
//correct: notSET and RESET signals, during the two housekeeping cycles which are
occupied by SET/RESET process, do respect minimum and maximum times indicated in
sensor's datasheet
}

void Bk1B22_Software_Tests::test_housekeeping() {
// !!! first of all, suppress supervise() operation calling in housekeep-
ing(SETRESET_MAGN) of Bk1B229_Controller.cpp, cause it might configure magnetometer
and coil as disabled
//monitor EN_MAGN and EN_COIL signals using oscilloscope
//monitor which position is index of housekeeping(index) function with debug tool,
using proper breakpoints

//both magnetometer and coil are considered attached and enabled
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//turn magnetometer supply off
Bk1B22.sensor.EN_MAGN(false);

Bk1B22.actuator.activate(true);   //force magnetic coil on

//EN_MAGN signal is 'false' and EN_COIL is 'true'

timer.start();      //launch TimerA2 which periodically calls housekeeping(index)

//see related pictures (7.6, 7.7, 7.8)
//after a few tens of ms, EN_MAGN signal switches to 'true' and simultaneously
EN_COIL to 'false': this is assumed as t=0
//correct -> at index MAGNETIC_FIELD_X-2, if enabled, magnetometer has to be sup-
plied, while coil has to be turned off

//at t=40 ms, EN_MAGN signal switches to 'false'
//correct -> at index MAGNETIC_FIELD_Y+1, magnetometer has to be switched off be-
cause magnetic field acquiring (both x and y) is finished

//at t=50 ms, EN_COIL signal switches to 'true'
//correct -> at index MAGNETIC_FIELD_Y+2, coil has to be turned on again

//at t~200 ms, EN_COIL signal switches to 'false'
//correct -> when index is EOC_INTEGRAL, coil is immediately deactivated because
there is no angular momentum to generate

//as we can see in picture 7.6, next cycle EN_MAGN signal repeats its routine cor-
rectly

}

void Bk1B22_Software_Tests::test_acquiring() {
    volatile int i;
//launch boot() operation, which initializes:
// - actual_L, SRoffset_Xset, SRoffset_Yset, SRoffset_Xreset, SRoffset_Yreset to 0
// - magnetometer and magnetic actuator as disabled
// - offset and gain parameters of t_scaling structure to proper values (see boot()
operation code) for MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y and COIL_CURRENT
Bk1B22.boot();
```

```
//enable magnetometer and magnetic torque actuator
                                    con-
fig.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);
                                    con-
fig.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//turn magnetic coil on to measure and get operating current value
Bk1B22.actuator.activate(true);

//start TimerA2, which automatically calls Bk1B229_Controller.housekeeping(index)
every 10 ms
timer.start();

//wait for 320000 processor clock cycles (10 clock ticks every 'for' cycle):
//this time guarantees that parameter index of controller.housekeeping operation is
> max {Bk1B22.COIL_CURRENT, Bk1B22.MAGNETIC_FIELD_X, Bk1B22.MAGNETIC_FIELD_Y}
for(i=0;i<32000;i++);

printf("MAGN_X = %d  MAGN_Y = %d COIL_CURRENT = %d\n",
Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_FIELD_X],
Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_FIELD_Y],
Bk1B22.controller.hk.housekeeping[Bk1B22.COIL_CURRENT]);
//opening Terminal IO, look at the values acquired and scaled by the system ADC:
//in a typical test session, watching the above 'printf' screen, MAGN_X was -39 mG,
MAGN_Y was -344 mG, COIL_CURRENT was 6013e-4 A (measure units are reported in pro-
ject specifications)
//ADC inputs for magnetic field on x-axis, magnetic field on y-axis and coil cur-
rent were respectively 1.187 V, 1.033 V and 1.203 V, measured with DVM9912
multimeter
//using proper breakpoints and 'watch' window of debug tool, ADC outputs before
scaling can be observed:
//channels MAGNETIC_FIELD_X_ADC, MAGNETIC_FIELD_Y_ADC and COIL_CURRENT_ADC conver-
sion results (before scaling) were respectively [LSB] 1944, 1693 and 1970
//coversion results are perfectly corresponding to input voltages -> ADC works cor-
rectly
//using proper breakpoints and 'watch' window of debug tool, [gain ; offset] pa-
rameters used by ADC.acquire_scale(...) operation can be viewed too:
//for channels MAGNETIC_FIELD_X_ADC, MAGNETIC_FIELD_Y_ADC and COIL_CURRENT_ADC,
they were respectively
//[9985 ; 1976], [9985 ; 1976], [25006 ; 0]
//MAGNETIC_FIELD_X -> non-scaled value = 1945 -> scaled value = (1944-
1976)*9985/2^13 = -39
//MAGNETIC_FIELD_X -> non-scaled value = 1694 -> scaled value = (1693-
1976)*9985/2^13 = -344
//COIL_CURRENT -> non-scaled value = 1970 -> scaled value = 1970*25006/2^13 = 6013
//scaling results properly express magnetic field and coil current values
}

void Bk1B22_Software_Tests::test_actuate_coil_1() {
// !!! first of all, suppress supervise() operation calling in housekeep-
ing(SETRESET_MAGN) of Bk1B229_Controller.cpp, cause it might configure magnetometer
and coil as disabled
                  byte *temp;

//launch boot() operation, which initializes:
// - actual_L, SRoffset_Xset, SRoffset_Yset, SRoffset_Xreset, SRoffset_Yreset to 0
// - magnetometer and magnetic actuator as disabled
// - offset and gain parameters of t_scaling structure to proper values (see boot()
operation code) for MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y and COIL_CURRENT
Bk1B22.boot();

//start TimerA2, which automatically calls Bk1B229_Controller.housekeeping(index)
every 10 ms
timer.start();

//write an angular momentum of 1000 g*cm^2/s into buffer
temp =
Bk1B22.controller.buffers.lock((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD_
ACTUATE_COIL);
t_sensor x = 1000;
* ((t_sensor *) temp) = x;

Bk1B22.controller.buffers.ready((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD
_ACTUATE_COIL, 2);
//buffer has been written and is ready to be read

//enable both MAGNETOMETER and MAGNETIC ACTUATOR blocks
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//launch CMD_ACTUATE_COIL command, which reads angular momentum into buffer and
saves it into 'actual_L' attribute and then turns the magnetic coil on

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A_1B8_Codes::CMD_ACTUATE_COIL,
NO_ERROR);

//with proper use of breakpoints in debug session, has been verified that coil is
```

```
active (except when acquiring magnetic field) until there is no more angular momen-
tum in 'actual_L'
//every housekeeping cycle, the controller.integrate() operation is called: on
every call it subtracts, from 'actual_L' value, the quantity B*II*K, where:
//II is coil current acquired value, B is module of magnetic field projection on xy
plane and K is the product of constants like equivalent coil area, integration time
step and sensitivities
//K = 25.11e-5, while, in a typical test session, B was 440 and II was 5990 (these
values are easily visible enabling the line containing 'printf' function in the in-
tegrate() operation)
//B*II*K = 661.8 -----> because of integer type, subtracted quantity was 661 (On
first step. On second step was slightly different because current and magnetic
field are not ideally constant)
//initial condition [t=0]--> actual_L = 1000 (command CMD_ACTUATE_COIL just inter-
preted)
//first housekeeping cycle [t=1s]--> actual_L = 339
//second housekeeping cycle [t=2s]--> actual_L = -317 at this moment actual_L<0, so
coil driving was stopped
//to generate a 1000 g*cm^2/s angular momentum, system drives magnetic actuator for
two housekeeping cycles (each one lasts 1s)
//this is correct, because typical values of coil current and magnetic field pro-
jection on tile plane (module) are, respectively, 0.599 A and 0.440 G
//in this case, generated torque module is 66.18e-6 Nm: this value perfectly
matches with single step decrement of 'actual_L'
//supposing constant torque, required time is (1000 g*cm^2/s) / (66.18e-6
Kg*m^2/s^2) = 1.51 s, whose upper integer value is 2s
}

void Bk1B22_Software_Tests::test_actuate_coil_2() {
// !!! first of all, suppress supervise() operation calling in housekeep-
ing(SETRESET_MAGN) of Bk1B229_Controller.cpp, cause it might configure magnetometer
and coil as disabled
//for further details, see similar function, test_actuate_coil_1()
byte *temp;

//launch boot() operation, which initializes:
// - actual_L, SRoffset_Xset, SRoffset_Yset, SRoffset_Xreset, SRoffset_Yreset to 0
// - magnetometer and magnetic actuator as disabled
// - offset and gain parameters of t_scaling structure to proper values (see boot()
operation code) for MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y and COIL_CURRENT
Bk1B22.boot();

//start TimerA2, which automatically calls Bk1B229_Controller.housekeeping(index)
every 10 ms
timer.start();

//write an angular momentum of 30000 g*cm^2/s into buffer
temp =
Bk1B22.controller.buffers.lock((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD_
ACTUATE_COIL);
t_sensor x = 30000;
* ((t_sensor *) temp) = x;

Bk1B22.controller.buffers.ready((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD
_ACTUATE_COIL, 2);
//buffer has been written and is ready to be read

//enable both MAGNETOMETER and MAGNETIC ACTUATOR blocks
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//launch CMD_ACTUATE_COIL command, which reads angular momentum into buffer and
saves it into 'actual_L' attribute and then turns the magnetic coil on

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A_1B8_Codes::CMD_ACTUATE_COIL,
NO_ERROR);

//with proper use of breakpoints in debug session, has been verified that coil is
active (except when acquiring magnetic field) until there is no more angular momen-
tum in 'actual_L'
//in a typical test session, to generate a 30000 g*cm^2/s angular momentum, system
drives magnetic actuator for 54 housekeeping cycles (each one lasts 1s)
//this is correct, because at the end of coil driving, last values of acquired coil
current and magnetic field projection on tile plane (module, calculated with ac-
quired Bx and By) were, respectively, 0.601 A and 0.376 G
//in this case, generated torque module is 56.74e-6 Nm
//supposing constant torque, required time is (30000 g*cm^2/s) / (56.74e-6
Kg*m^2/s^2) = 52.87 s, whose upper integer value is 53s
//there is a slight difference between the teoric value just calculated and the ac-
tual one (54s), but it's perfectly understandable: while the first one refers only
to the last acquirements, the second is the result of many different torque values
and, so, different quantities are subtracted from 'actual_L' on each housekeeping
cycle
//'actual_L' value when coil driving is stopped is -370
//medium decrease per housekeeping cycle is (30000 + 370) [g*cm^2/s] / 54 [s] =
56.24e-6 [Nm]
}
```

```
void Bk1B22_Software_Tests::test_actuate_coil_3() {
// !!! first of all, suppress supervise() operation calling in housekeep-
ing(SETRESET_MAGN) of Bk1B229_Controller.cpp, cause it might configure magnetometer
and coil as disabled
//for further details, see similar function, test_actuate_coil_1()
byte *temp;

//launch boot() operation, which initializes:
// - actual_L, SRoffset_Xset, SRoffset_Yset, SRoffset_Xreset, SRoffset_Yreset to 0
// - magnetometer and magnetic actuator as disabled
// - offset and gain parameters of t_scaling structure to proper values (see boot()
operation code) for MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y and COIL_CURRENT
Bk1B22.boot();

//start TimerA2, which automatically calls Bk1B229_Controller.housekeeping(index)
every 10 ms
timer.start();

//write an angular momentum of -1000 g*cm^2/s into buffer
temp =
Bk1B22.controller.buffers.lock((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD_
ACTUATE_COIL);
t_sensor x = -1000;
* ((t_sensor *) temp) = x;

Bk1B22.controller.buffers.ready((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD
_ACTUATE_COIL, 2);
//buffer has been written and is ready to be read

//enable both MAGNETOMETER and MAGNETIC ACTUATOR blocks
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//launch CMD_ACTUATE_COIL command, which reads angular momentum into buffer and
saves it into 'actual_L' attribute and then turns the magnetic coil on

Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A_1B8_Codes::CMD_ACTUATE_COIL,
NO_ERROR);

//signal on PHASE pin of coil driver has been correctly set to ‘false’
//with proper use of breakpoints in debug session, has been verified that ‘ac-
tual_L’ stored value has been changed of sign, from (-), to (+), when launching
function interpret(CMD_ACTUATE_COIL)
//then, coil driving proceeds normally like in case 1: coil is active (except when
acquiring magnetic field) until there is no more angular momentum in 'actual_L'
//to generate a -1000 g*cm^2/s angular momentum, system drives magnetic actuator
for two housekeeping cycles, but current flows reverse into magnetic coil

}

void Bk1B22_Software_Tests::test_actuate_coil_4() {
// !!! first of all, suppress supervise() operation calling in housekeep-
ing(SETRESET_MAGN) of Bk1B229_Controller.cpp, cause it might configure magnetometer
and coil as disabled
//for further details, see similar function, test_actuate_coil_1()
byte *temp;

//launch boot() operation, which initializes:
// - actual_L, SRoffset_Xset, SRoffset_Yset, SRoffset_Xreset, SRoffset_Yreset to 0
// - magnetometer and magnetic actuator as disabled
// - offset and gain parameters of t_scaling structure to proper values (see boot()
operation code) for MAGNETIC_FIELD_X, MAGNETIC_FIELD_Y and COIL_CURRENT
Bk1B22.boot();

//start TimerA2, which automatically calls Bk1B229_Controller.housekeeping(index)
every 10 ms
timer.start();

//write an angular momentum of 30000 g*cm^2/s into buffer
temp =
Bk1B22.controller.buffers.lock((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD_
ACTUATE_COIL);
t_sensor x = 30000;
* ((t_sensor *) temp) = x;

Bk1B22.controller.buffers.ready((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD
_ACTUATE_COIL, 2);
//buffer has been written and is ready to be read

//enable both MAGNETOMETER and MAGNETIC ACTUATOR blocks
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//launch CMD_ACTUATE_COIL command, which reads angular momentum into buffer and
saves it into 'actual_L' attribute and then turns the magnetic coil on
```

```
Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A_1B8_Codes::CMD_ACTUATE_COIL,
NO_ERROR);

//at this point, has been verified that coil is properly driven by measuring cur-
rent with multimeter DVM9912 -> coil current is 0.6 A

//write an angular momentum of 0 g*cm^2/s into buffer, which consists in aborting
previous actuation command and disabling coil driving
temp =
Bk1B22.controller.buffers.lock((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD_
ACTUATE_COIL);
x = 0;
* ((t_sensor *) temp) = x;

Bk1B22.controller.buffers.ready((Bk1A1_Common_Codes::t_Commands)Bk1A_1B8_Codes::CMD
_ACTUATE_COIL, 2);
//buffer has been written and is ready to be read

//now CMD_ACTUATE_COIL command can be launched again: after reading momentum into
buffer and saving it into 'actual_L' attribute, coil should be immediately turned
off
//put a breakpoint here, then enter into interpret(CMD_ACTUATE_COIL) and verify the
above constraints


Bk1B22.interpret((Bk1A_1B8_Codes::t_Commands_1B8)Bk1A_1B8_Codes::CMD_ACTUATE_COIL,
NO_ERROR);

//magnetic coil driving correctly stops; coil current is less than 0.01 A
}

void Bk1B22_Software_Tests::test_supervise() {
//before test begins, suppress supervise() operation automatic calling into con-
troller.housekeeping(SETRESET_MAGN) because it will be manually called

volatile int i;

//1st CASE:
//detach and disable both magnetometer and magnetic coil blocks
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);

//launch supervise operation, which shouldn't find any errors, except for the low
PDB voltage error: this is unavoidable, because there is no PDB voltage acquiring
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 256 = 2^8, that is ninth bit of error word, corresponding to
LOW_PDB_VOLT error set to 'true'
//all other bits are 'false' -> 1st case test successful

//2nd CASE:
//detach both magnetometer and magnetic coil blocks
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);

//force as enabled both blocks
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);

//launch supervise operation, which should find 3 errors:
//low PDB voltage error, corresponding to 9th bit
//ERR_MAGN_ENABLE (7th bit), because magnetometer is detached, so it should be dis-
abled too, but results as enabled
//ERR_COIL_ENABLE (4th bit), because magnetic coil is detached, so it should be
disabled too, but results as enabled
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 328 = 2^8 + 2^6 + 2^3 -> 9th, 7th, 4th bits of error word,
respectively corresponding to LOW_PDB_VOLT, ERR_MAGN_ENABLE, ERR_COIL_ENABLE er-
rors, are set to 'true'
                  //all other bits are 'false' -> 2nd case test successful

//3rd CASE:
//with both blocks disabled, force magnetic coil status as 'active', after changing
visibility of attribute 'status' of Bk1B222_Magnetic_Torque_Actuator class to 'pub-
lic'
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETOMETER);
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);
```

```
Bk1B22.actuator.status.status_1B2.set(Bk1B22.actuator.status.status_1B2.ACTIVE_COIL
);

//launch supervise operation, which should report the previous errors plus
ERR_COIL_STATUS error, because coil is detached, so it shouldn't result as active
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 344 = 2^8 + 2^6 + 2^4 + 2^3 -> 9th, 7th, 5th, 4th bits of
error word, respectively corresponding to LOW_PDB_VOLT, ERR_MAGN_ENABLE,
ERR_COIL_STATUS, ERR_COIL_ENABLE errors, are set to 'true'
//all other bits are 'false' -> 3rd case test successful

//before proceeding with coil current tests, error word is reset to better inter-
pret next cases' errors
Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS] = 0;

//launch controller.boot() operation, which initalizes gain and offset parameters
for magnetic field and coil current acquiring and scaling
Bk1B22.boot();

//from this point on, is necessary to launch TimerA2, which automatically calls
controller.housekeeping() operation to acquire coil current
timer.start();

//4th CASE: normal coil operating conditions
//set both magnetometer and magnetic actuator blocks as attached and enabled
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETOMETER);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETOMETER);

//start coil driving
Bk1B22.actuator.activate(true);

//wait for 320000 processor clock cycles (10 clock ticks every 'for' cycle):
//this time guarantees that parameter index of controller.housekeeping operation is
> max {Bk1B22.COIL_CURRENT, Bk1B22.MAGNETIC_FIELD_X, Bk1B22.MAGNETIC_FIELD_Y}
//so is assured that coil current value will be acquired in time for supervise()
intervention
for(i=0;i<32000;i++);

//with proper use of breakpoints and watch tool, in debug session has been observed
that:
//coil current has a regular value -> Bk1B22.actuator.Norm_Curr = 5.988e-1

//launch supervise operation, which shouldn't find any errors, except for the low
PDB voltage error
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 256 = 2^8, that is ninth bit of error word, corresponding to
LOW_PDB_VOLT error set to 'true'
//all other bits are 'false' -> 4th case test successful

//5th CASE: current flowing into coil when it is disabled
//set magnetometer as detached (and disabled, too) and magnetic coil as attached,
but disabled
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);

//magnetic coil must result as not active in status word

Bk1B22.actuator.status.status_1B2.reset(Bk1B22.actuator.status.status_1B2.ACTIVE_CO
IL);

//force current into coil by directly controlling driver pins
Bk1B22.actuator.EN_COIL(true);    //giving power supply to magnetic actuator block
Bk1B22.actuator.NOT_BRAKE(true);
Bk1B22.actuator.MODE(true);
Bk1B22.actuator.PHASE(true);
Bk1B22.actuator.NOT_ENABLE(false);   //enable coil driving

//restart TimerA2
timer.start();

//wait for 320000 processor clock cycles (10 clock ticks every 'for' cycle):
//so is assured that coil current value will be acquired in time for supervise()
intervention
for(i=0;i<32000;i++);

//with proper use of breakpoints and watch tool, in debug session has been observed
that:
```

```
//Bk1B22.actuator.Norm_Curr = 6.03e-1 -> it is a typical operating value, clearly
wrong

//launch supervise operation, which should find LOW_PDB_VOLT error plus
OFF_CURR_EXCESS error, because coil is disabled, so there shouldn't be current (ex-
actly, value should be <5% of maximum operating current)
//it should also detach magnetic coil, because of current presence when coil is
disabled
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 260 = 2^8 + 2^2 -> 9th and 3rd bit of error word, respec-
tively corresponding to LOW_PDB_VOLT and OFF_CURR_EXCESS errors, are set to 'true'
//all other bits are 'false', magnetic coil power supply is turned off and it is
detached -> 5th case test successful

//6th CASE: normal coil off conditions
//deactivate coil and reset error word
Bk1B22.actuator.activate(false);
Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS] = 0;

//detach and disable both magnetometer and magnetic coil blocks
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ENABLE_MAGNETOMETER);
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.reset(config.config_1B2.ATTACH_MAGNETOMETER);

//restart TimerA2
timer.start();

//wait for 320000 processor clock cycles (10 clock ticks every 'for' cycle):
//so is assured that coil current value will be acquired in time for supervise()
intervention
for(i=0;i<32000;i++);

//launch supervise operation, which shouldn't find any errors, except for the low
PDB voltage error
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 260 = 2^8 + 2^2 -> 9th and 3rd bit of error word, respec-
tively corresponding to LOW_PDB_VOLT and OFF_CURR_EXCESS errors, are set to 'true';
all other bits are 'false'
//there is an error!!! when magnetic coil is detached and disabled there is still
too much current:
//EN_COIL signal, which controls enables power supply of magnetic actuator block,
is 'false' -> correct
//voltage on COIL_CURRENT_ADC input channel of ADC is about 80 mV, even if load is
removed
//conversion result at the ADC output is 131 -> correct, it perfectly matches with
input voltage
//controller.hk.housekeeping[COIL_CURRENT] scaled value is 399, in units of e-4 A -
> correct: considering that coil current sensing circuit gain is 2, a current of
about 40 mA perfectly matches with measured ADC input voltage
//there is no voltage on 0.2 ohm sensing resistor (measured 1.7 mV on both resistor
ends), therefore there is no considerable current flowing through it
//so the 399e-4 A read value is not a real current flowing through coil (and sens-
ing resistor) and may be due to an excessive conditioning opamp offset
//in future, this virtual residual current will have to be considered and cali-
brated at ground

//7th CASE: too little current flowing into coil when it is on
//set magnetic actuator block as attached and enabled
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//activate coil and reset error word
Bk1B22.actuator.activate(true);
Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS] = 0;

//using a proper breakpoint, stop the execution at this moment and manually remove
22 ohm resistor (which simulates magnetic coil), so there is no more current

//restart TimerA2
timer.start();

//wait for 320000 processor clock cycles (10 clock ticks every 'for' cycle):
//so is assured that coil current value will be acquired in time for supervise()
intervention
for(i=0;i<32000;i++);
//after this time, using watch tool it can be verified that acquired coil current
value is 39 mA

//launch supervise operation, which should find LOW_PDB_VOLT error plus
ON_CURR_LACK error, because coil results as operating but current is too low
```

```
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 258 = 2^8 + 2^1 -> 9th and 2nd bit of error word, respec-
tively corresponding to LOW_PDB_VOLT and ON_CURR_LACK errors, are set to 'true'
//all other bits are 'false' -> 7th case test successful

//8th CASE: too high current flowing into coil when it is on
//set magnetic actuator block as attached and enabled
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//to get a bigger current value than before, 22 ohm 15W resistor which simulates
magnetic coil has been replaced by an 18 ohm 25W one and PDB supply voltage has
been turned up to 17.6 V

//activate coil and reset error word
Bk1B22.actuator.activate(true);
Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS] = 0;

//in this case, it is necessary to simulate a coil actuation situation, forcing a
value for angular momentum attribute 'actual_L', which is into Bk1B229_Controller,
after having changed its visibility from 'private' to 'public'
//then magnetometer is attached and enabled, to acquire not only coil current, but
also magnetic field components
//if not, magnetic coil will be configured as not active by control-
ler.housekeeping(EOC_INTEGRAL) before supervise() operation can begin
Bk1B22.controller.actual_L = 10000;
config.config_1B2.set(config.config_1B2.ATTACH_MAGNETIC_COIL);
config.config_1B2.set(config.config_1B2.ENABLE_MAGNETIC_COIL);

//restart TimerA2
timer.start();

//wait for 320000 processor clock cycles (10 clock ticks every 'for' cycle):
//so is assured that coil current value will be acquired in time for supervise()
intervention
for(i=0;i<32000;i++);
//after this time, using watch tool it can be verified that acquired coil current
value is 874 mA
//critical maximum value for coil current (see project specifications) is
I_SUPPLY_MAX + 20% = 763 mA

//launch supervise operation, which should find LOW_PDB_VOLT error plus
ON_CURR_EXCESS error, because coil results as operating but current is too high
//it should also disable magnetic coil, because of operating current excess
Bk1B22.supervise();

printf("%d \n",Bk1B22.controller.hk.housekeeping[Bk1B22.MAGNETIC_ERRORS]);

//Terminal IO displays 257 = 2^8 + 2^0 -> 9th and 1st bit of error word, respec-
tively corresponding to LOW_PDB_VOLT and ON_CURR_EXCESS errors, are set to 'true'
//all other bits are 'false' and magnetic coil is disabled -> 8th case test suc-
cessful

}
```

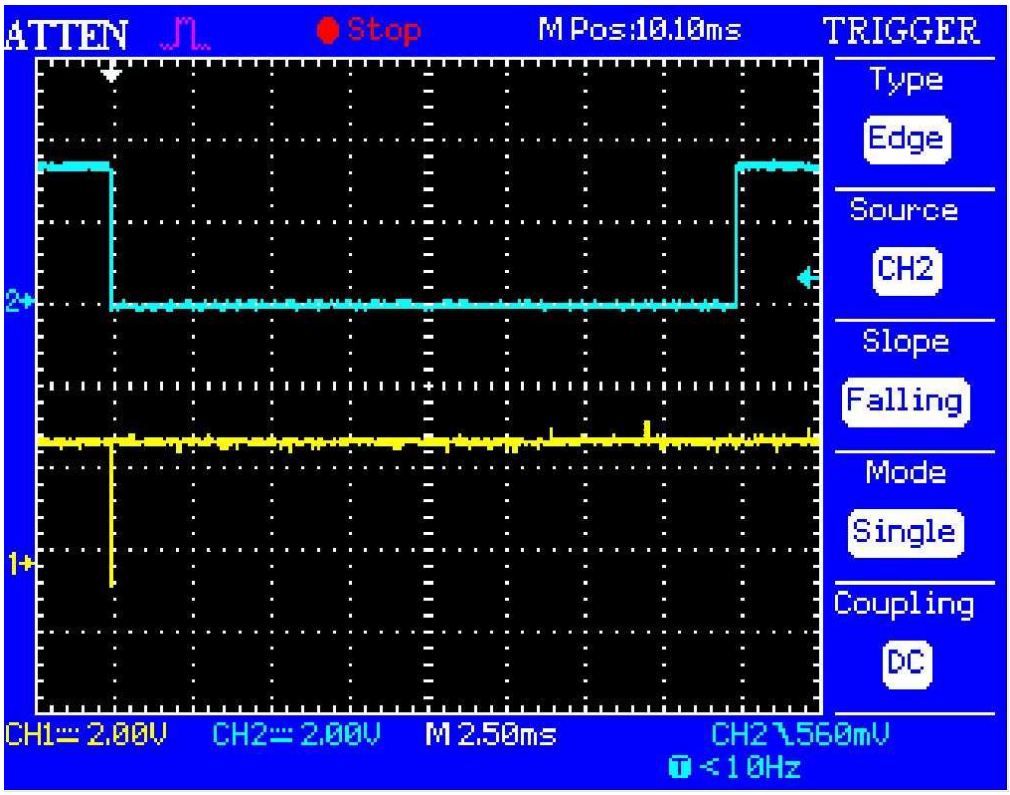


Figura 7.4 RESET (blue) and notSET (yellow) signals during *test_setreset()* routine, covering two *housekeeping()* cycles

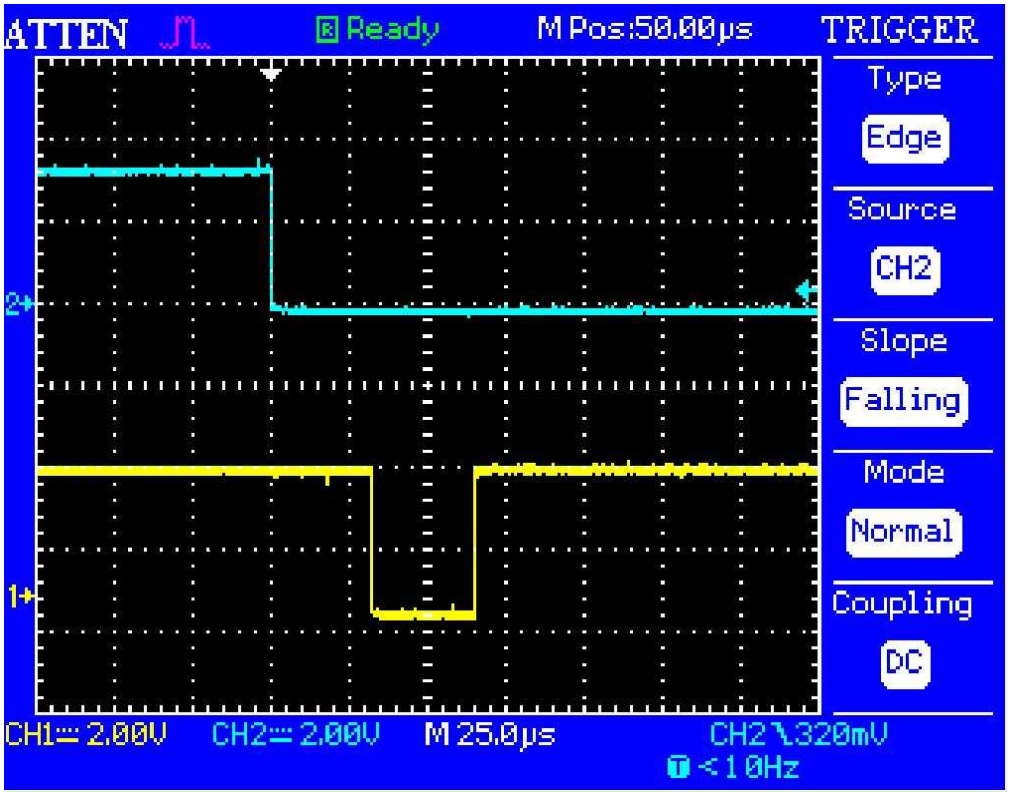


Figura 7.5 RESET (blue) and notSET (yellow) signals during *test_setreset()* routine: zoom on the first *housekeeping()* cycle

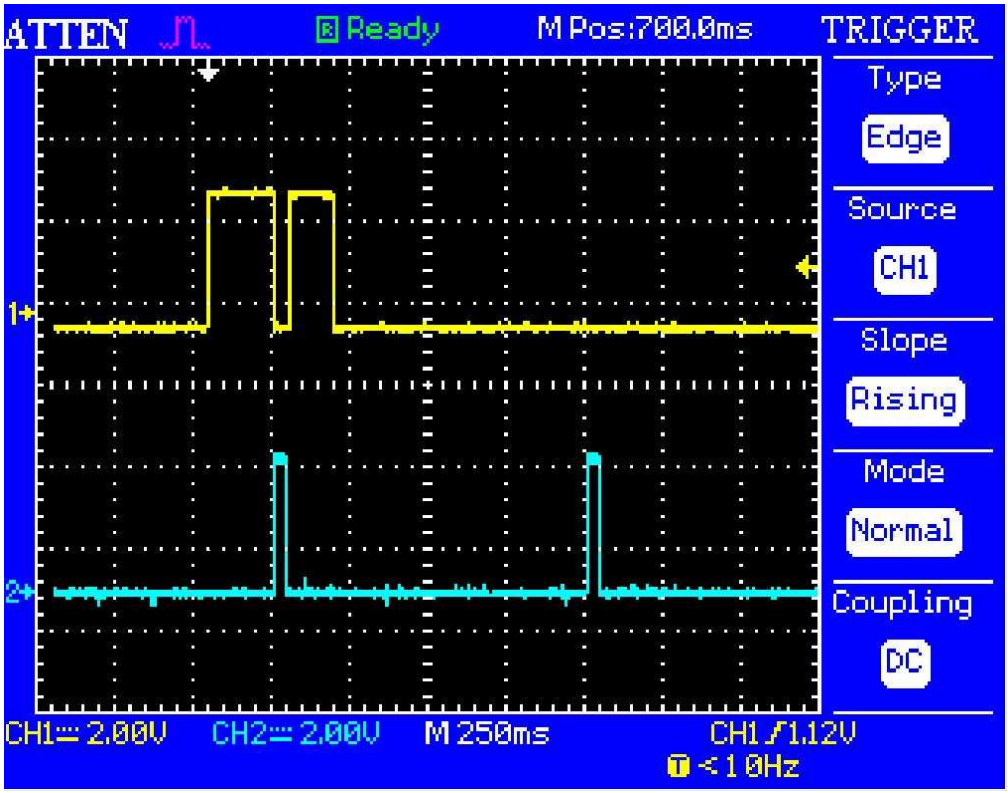


Figura 7.6 EN_COIL (yellow) and EN_MAGN (blue) signals during *test_housekeeping()* routine

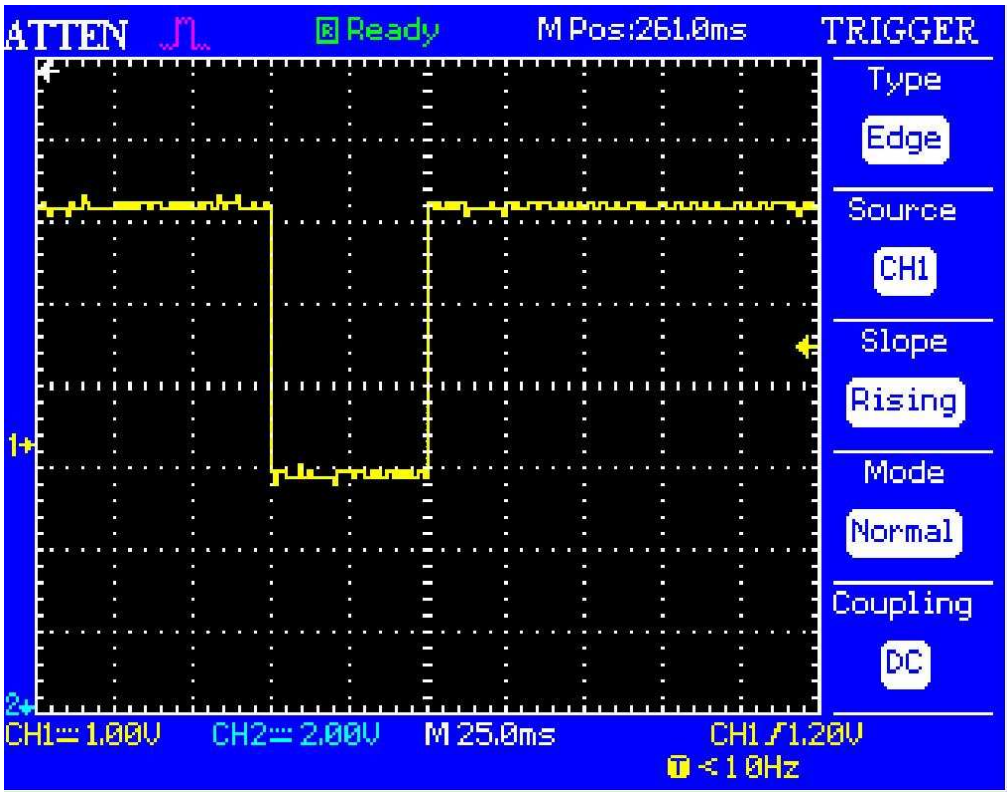


Figura 7.7 Zoom on EN_COIL signal during *test_housekeeping()* routine

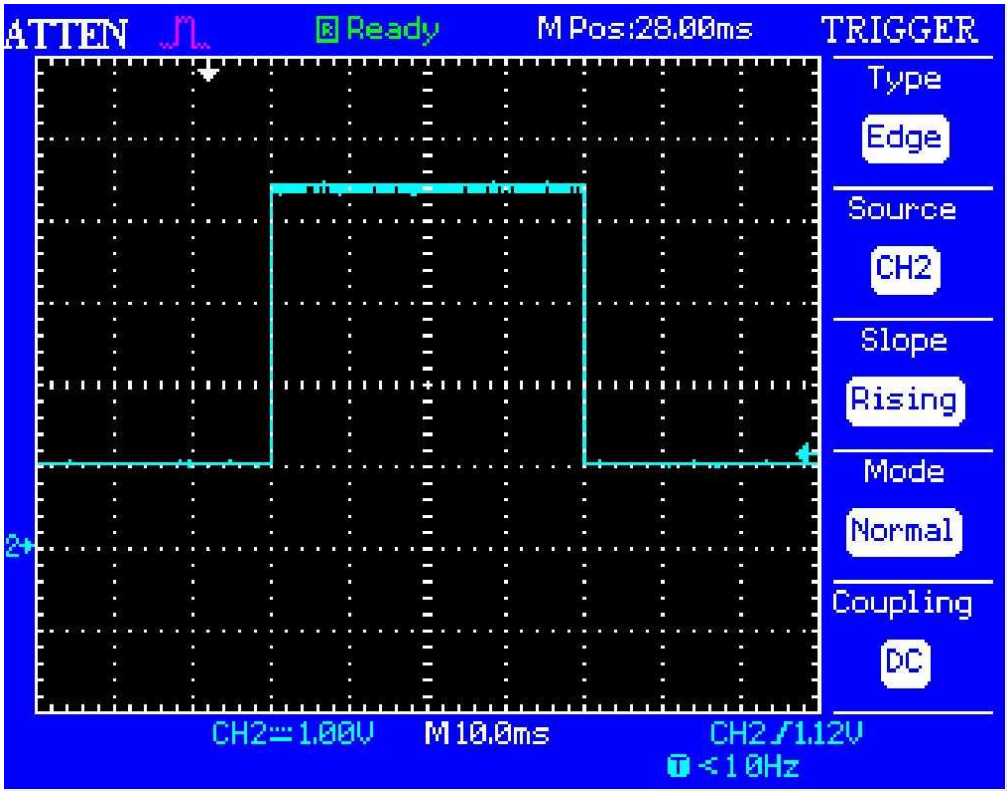


Figura 7.8 Zoom on EN_MAGN signal during *test_housekeeping()* routine

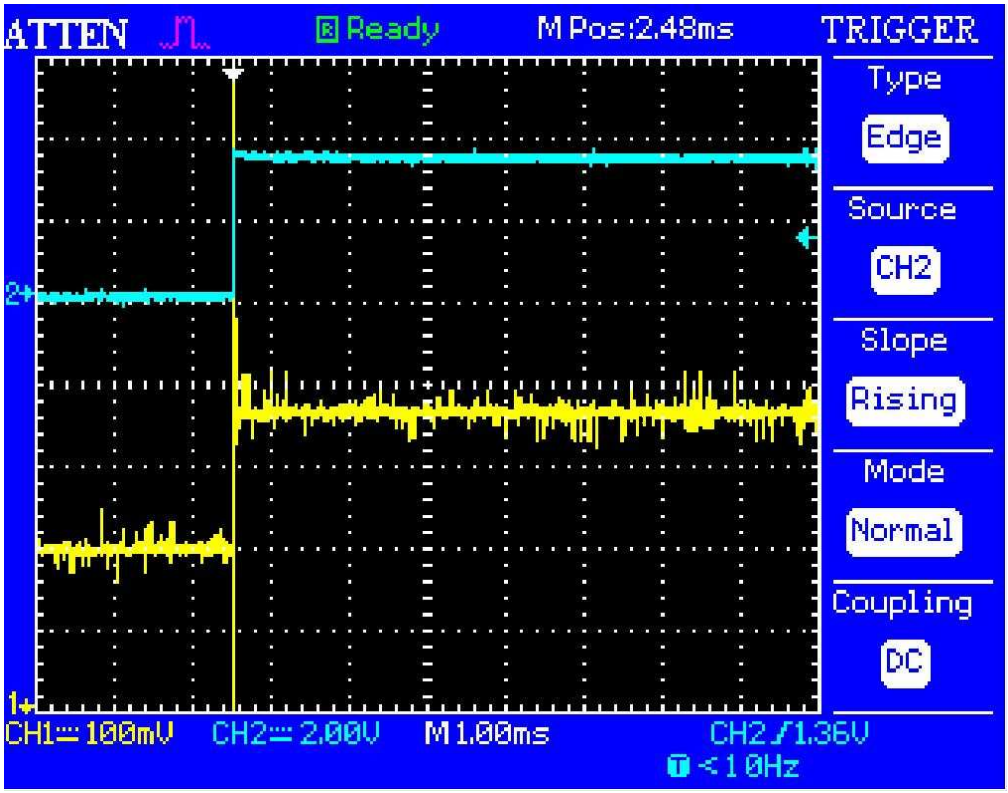


Figura 7.9 EN_MAGN (blue) and MAGN_X (yellow) signals during *test_acquiring()* routine

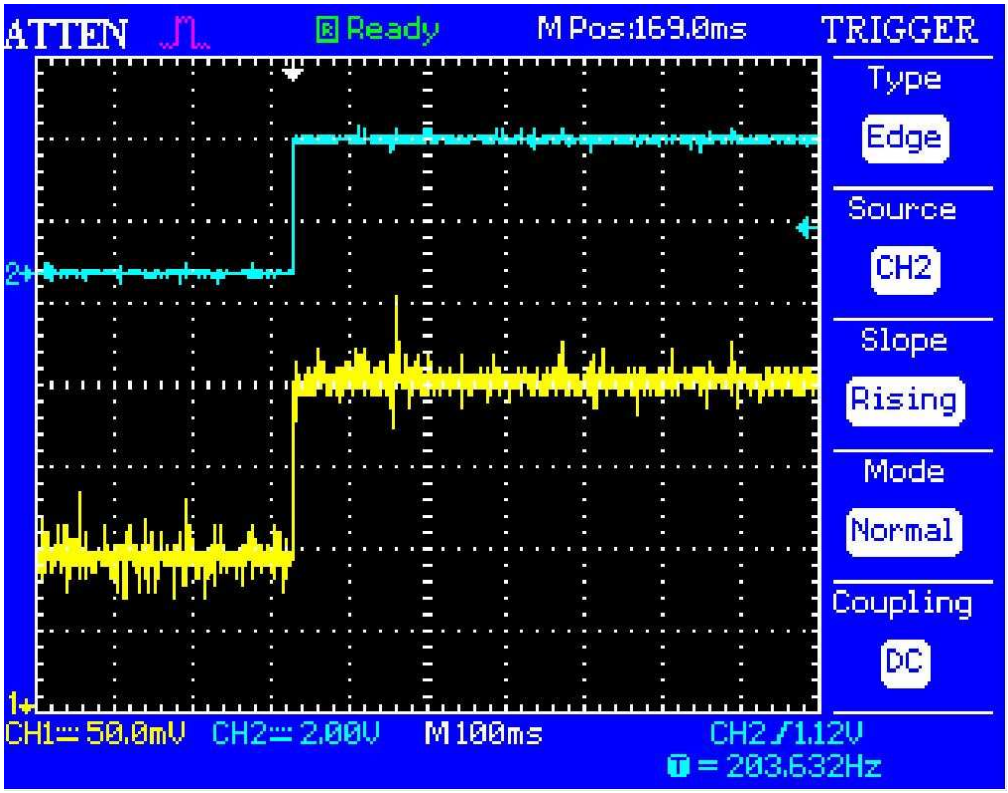


Figura 7.10 EN_MAGN (blue) and MAGN_Y (yellow) signals during *test_acquiring()* routine

Bk1B22_Software_Tests.h

```
#pragma diag_suppress=Pa050
#ifndef __Bk1B22_Software_Tests_h__
#define __Bk1B22_Software_Tests_h__

#include "platform.h"

#include "Bk1B22_Magnetic_Attitude_Subsystem.h"
#include "t_Configuration_1B8.h"
#include "TimerA2.h"

class Bk1B22_Magnetic_Attitude_Subsystem;
class t_Configuration_1B8;
class TimerA2;
class Bk1B22_Software_Tests;

class Bk1B22_Software_Tests
{
   private: static Bk1B22_Magnetic_Attitude_Subsystem Bk1B22;
   private: static t_Configuration_1B8 config;
   private: static TimerA2 timer;

   public: void test_interpret();
   public: void test_cmd_setconfig();
   public: void test_cmd_rstconfig();
   public: void test_setreset();
   public: void test_housekeeping();
   public: void test_acquiring();
   public: void test_actuate_coil_1();
   public: void test_actuate_coil_2();
   public: void test_actuate_coil_3();
   public: void test_actuate_coil_4();
   public: void test_supervise();
};

#endif
```

TimerA2.cpp

```
#include "TimerA2.h"

#include "Bk1B229_Controller.h"

ushort TimerA2::index;
Bk1B229_Controller TimerA2::controller;
```

TimerA2.h

```
#pragma diag_suppress=Pa050
#include "Slave_Processor.h"

#ifndef __TimerA2_h__
#define __TimerA2_h__

#include "platform.h"

#include "Bk1B229_Controller.h"

class Bk1B229_Controller;
class TimerA2;

class TimerA2
{
   private: static ulong  const CLOCK_FREQ =
Bk1B45_Slave::Slave_Processor::CLOCK_FREQ;
   private: static ulong  const TIMER_FREQ = 100;
   private: static ushort  const TIMER_TICKS = 100;
   private: static ushort index;
   public: static Bk1B229_Controller controller;

   public:
   #pragma vector = TIMERA0_VECTOR
   __interrupt static void isr_timerA0() {
          if (++index >= TIMER_TICKS) index = 0;

          // place user code; e.g. Bk1Bxx::acquire();

          controller.housekeeping(index);
   }
   public: TimerA2() {
          #if MSP430
            // Timer A control register
            // configures Timer A for up mode, clock on SMCLK divided by 8; inter-
rupt disabled; timer frequency = TIMERA_FREQ
            TACTL = TASSEL1 // source clock from SMCLK
                    | ID1 | ID0 // Input divide by 8
                    | MC0 // up mode (counts from 0 up to TACCR0 included)
                    | TACLR ; // reset
                   // Interrupt disabled
            TACCTL0 = 0; // Capture mode = no capture
                         // output mode = output bit value (normal port operation)
                         // other bits are immaterial
            TACCR0 = (unsigned int) ((CLOCK_FREQ / TIMER_FREQ / 8) - 1); // count
period such as to trigger at TIMER_FREQ Hz, supposing processor clock frequency at
CLOCK_FREQ Hz.
            reset();
          #endif

          #if CHIPCON

          #endif
   }
   public: static bool radiationCheck() {
          #if MSP430
            if ( ! (
            ((TACTL & ~(TACLR | TAIFG)) == (TASSEL1 | ID1 | ID0 | MC0)) &&
            ((TACCTL0 & ~(CCIE | CCI | OUT | COV | SCCI)) == 0) &&
            (TACCR0 == (unsigned int) ((CLOCK_FREQ / TIMER_FREQ / 8) - 1))
            ) ) {
            TimerA2();
            return false;
            }
          return true;
          #endif

          #if CHIPCON

          #endif
   }
```

```
    public: static void start() {
            #ifdef MSP430
              reset();
              TACCTL0 |= CCIE;
            #endif
    }
    public: static void disable() {
            #ifdef MSP430
              // disables interrupt
              TACCTL0 &= ~CCIE;
            #endif
    }
    public: static void reset() {
            #ifdef MSP430
              TACTL |= TACLR;
              index = 0;
            #endif
    }
    public: static ushort read() {
            return TAR;
    }
};

#endif
```

7.2.3. Class TimerA2

The hardware and driver routines for TimerA2.

Attributes

Name (visibility) Documentation	*Type*	*Initial Value*
CLOCK_FREQ (private) Clock frequency of processor, on SMCLK input; in Hz	ulong	Bk1B45_Slave::Slave_Processor::CLOCK_FREQ
TIMER_FREQ (private) Frequency (in Hz) at which TimerA2 calls interrupt service routine isr_timerA0. Actual frequency might be slightly different, as permitted by the chosen processor clock frequency. Frequency accuracy depends on accuracy of processor clock.	ulong	100
TIMER_TICKS (private) Number of TimerHW ticks between each call of housekeeping functions with the same index value.	ushort	100
index (private) Index for counting, from 0 to TIMER_TICKS-1 the calls to the isr_timerA0. It is reset to 0 by start. It is incremented by 1 at every call of isr_timerA0, then overflows to 0 after TIMER_TICKS-1.	ushort	
controller (Unspecified)	Bk1B22_Magnetic_Attitude_Subsystem$Bk1B229_Controller	

Operations

Name (visibility) Documentation	*Parameters*	*Return Type*
isr_timerA0 (public) Interrupt service routine for TimerA2, which is periodically invoked at frequency TIMER_FREQ Hz.		
TimerA2 (public) Configures TimerA2 for up mode, clock on SMCLK divided by 8; interrupt disabled; timer frequency = TIMER_FREQ; timer reset. For MSP430, it shall be less than = (CLOCK_FREQ / 2^16 / 8).		
radiationCheck (public) Checks if any of the TimerA2 configuration registers (which should be constant) has been corrupted by a SEU.		bool
start (public) Resets and starts TimerA2 counting. The first interrupt is generated at the end of counting, that is, after 1/TIMER_FREQ seconds.		

disable (public) Disables TimerA2 interrupts. No more interrupt is therefore generated.		
reset (public) Resets TimerA2 counting. The first interrupt is generated at the end of counting, that is, after 1/TIMER_FREQ seconds.		
read (public) Reads TimerA2 counting. Counting starts from 0, therefore an immediate call to read right after start will return either 0 or a very small number.		ushort

7.2.4. Class : UnusedInterrupts

Stereotypes :	<<SW>>
Documentation :	Disables all unused interrupts (if any) and traps any of them in case that, for any reason, they get enabled by mistake. The watchdog is also disabled. The constructor disables all interrupts. Each interrupt service routine disables the corresponding interrupt enable flag. If an interrupt becomes used, the Designer shall: • duplicate this class and name it appropriately • remove the corresponding interrupt service (or watchdog) routine from the duplicated class to the relevant class • remove its disabling from the constructor

7.3. Risultati

Analizzando i risultati del collaudo HW e SW del modulo 1B22, riportati nella documentazione e nel codice delle routine di test (par. 7.1), si può concludere che, nella quasi totalità dei casi, il sistema ha risposto correttamente a tali procedure, rispecchiando correttamente le proprie funzionalità.
Per quanto concerne la parte hardware del progetto, sarà necessario, in futuro, riprogettare lo switch (controllato dal segnale EN_COIL) che abilita/disabilita la tensione di alimentazione per il carico del driver A3953, poiché malfunzionante nello stato di off.
Durante il test del software, invece, è stato notato che la sensibilità scelta per rappresentare il momento angolare richiesto ai controlli d'assetto delle singole tiles, risulta essere eccessiva, così da rendere inadeguato un campo di 16 bit per tale rappresentazione. In futuro, andrà, pertanto, ridefinita quest'ultima o, in alternativa, rimpicciolito il passo temporale di integrazione della coppia magnetica, nella generazione del momento angolare, tramite una maggiore frequenza di chiamata della funzione integrate della classe Bk1B229_Controller.
Inoltre, durante il test della routine di controllo della presenza di errori nel sistema (test_supervise), viene rilevato, in quanto correttamente acquisito e convertito dall'ADC, un valore eccessivo di corrente nel solenoide (circa 40 mA), quando quest'ultimo è disabilitato, sebbene, effettivamente, tale corrente non sia presente. Infatti, misurando la tensione ai capi della resistenza di sense, si può vedere che non vi scorre alcuna corrente (se non di valore largamente trascurabile). Nonostante ciò, si misurano circa 80 mV all'uscita del circuito di condizionamento. Quindi, tale tensione è dovuta ad un eccessivo contributo di offset introdotto dal circuito di condizionamento Bk1B137_Differential_Voltage_Sensor, già presente fra i componenti di progetto di AraMiS, che utilizza due amplificatori operazionali LM6142. Una possibile soluzione futura potrebbe essere l'eventuale sostituzione dell'operazionale in questo componente, tenendo presente che, comunque, tale contributo di offset è calibrabile a terra.
Infine, restano ancora da definire, nell'ambito del progetto AraMiS, le procedure per ottenere i valori di calibrazione dei dati di telemetria acquisiti. Di conseguenza, nel progetto appena concluso, i parametri di offset e guadagno di calibratura, nella funzione getSensorCalibration della classe Housekeeping del package Bk1B45_Slave, sono stati impostati, rispettivamente, a 0 e a 1.

Capitolo 8

8. Conclusioni

Il lavoro svolto durante la progettazione del modulo di controllo d'assetto magnetico per il satellite AraMiS ha permesso di mettere a frutto le nozioni acquisite durante gli anni trascorsi al Politecnico di Torino.
Inoltre, ha consentito la maturazione di nuove esperienze pratiche, l'apprendimento di nuovi concetti in ambito elettronico e l'approfondimento di quelli preesistenti. È stato possibile utilizzare e conoscere strumentazioni hardware e applicativi software indubbiamente indispensabili nella progettazione, sviluppo e collaudo di circuiti elettronici.

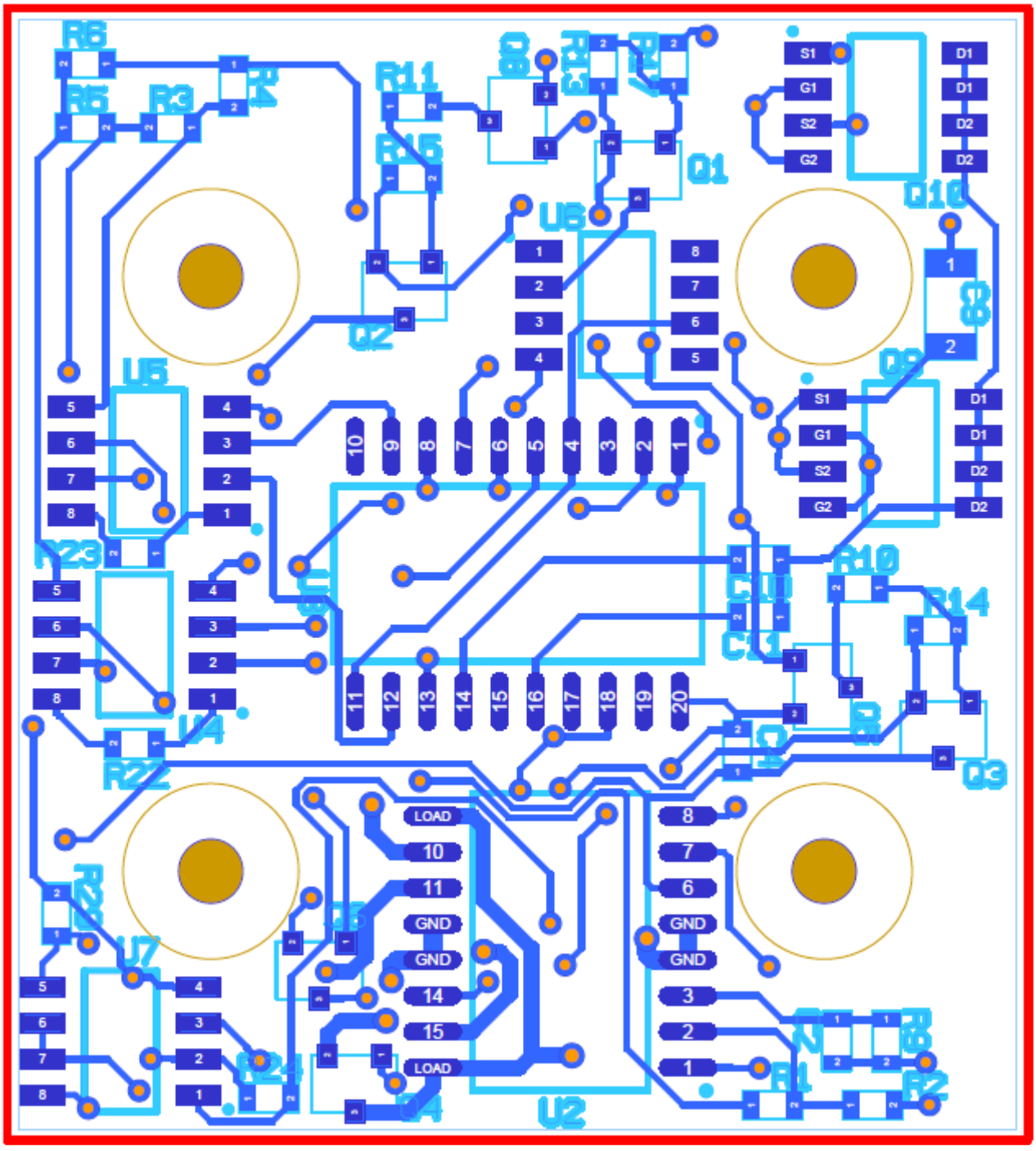
Dapprima è stata condotta un'analisi volta a comprendere le dinamiche del controllo di assetto di un satellite ed i vincoli ambientali a cui esso è sottoposto, per poi trattare in maniera più approfondita sugli aspetti inerenti il controllo di assetto magnetico, e la scelta del solenoide quale attuatore di coppia magnetica.
Dopodiché sono stati scelti e dimensionati i componenti hardware del sistema di controllo di assetto magnetico, riadattando, in parte, un progetto precedente [4] e, quindi, realizzati gli schemi elettrici. Dagli schemi è stato ricavato il layout PCB per il circuito stampato.
Una volta definita la struttura del progetto, quest'ultimo è stato totalmente descritto utilizzando il linguaggio UML, col quale sono stati documentati le specifiche ed i componenti. A partire dai diagrammi realizzati in UML, è stato implementato il software di controllo del sistema.
In seguito, sul circuito stampato, fabbricato (partendo dal PCB) da una ditta specializzata, sono stati montati i componenti elettronici, completando, così, il prototipo del modulo *1B22 Magnetic Attitude Subsystem*.

L'attività si è conclusa con il collaudo del modulo in laboratorio, sia dal punto di vista del funzionamento elettrico, sia per quanto riguarda il completo soddisfacimento delle specifiche operative del sistema. Il software del progetto è stato collaudato su un microcontrollore analogo a quello che verrà impiegato in via definitiva per la Power Management Tile di AraMiS.
L'esito dei test effettuati è stato complessivamente positivo. Tuttavia, sono stati riscontrati alcuni elementi che potrebbero essere migliorati in futuro.

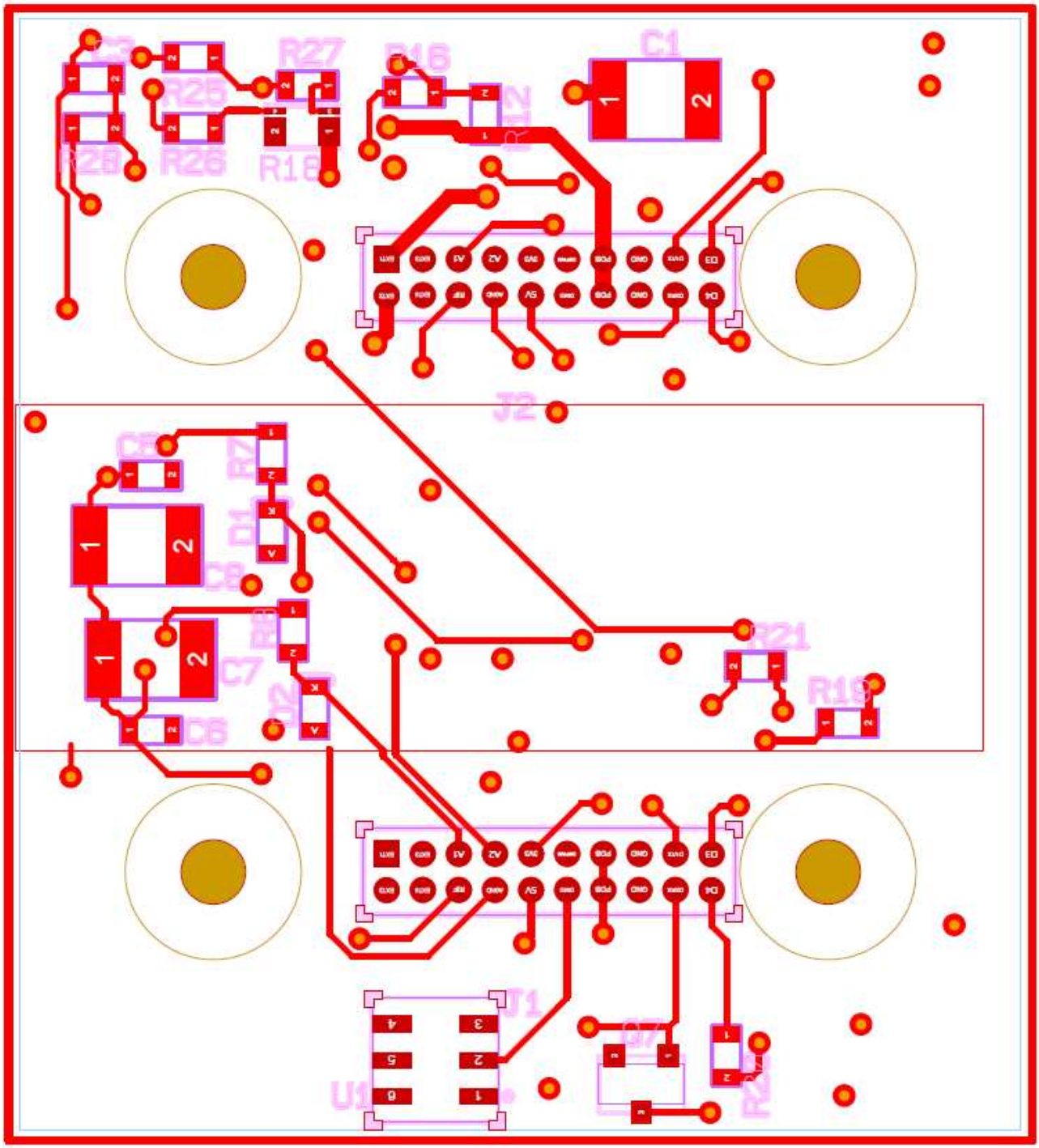
Appendici:

A. PCB di 1B22 – Magnetic Attitude Subsystem

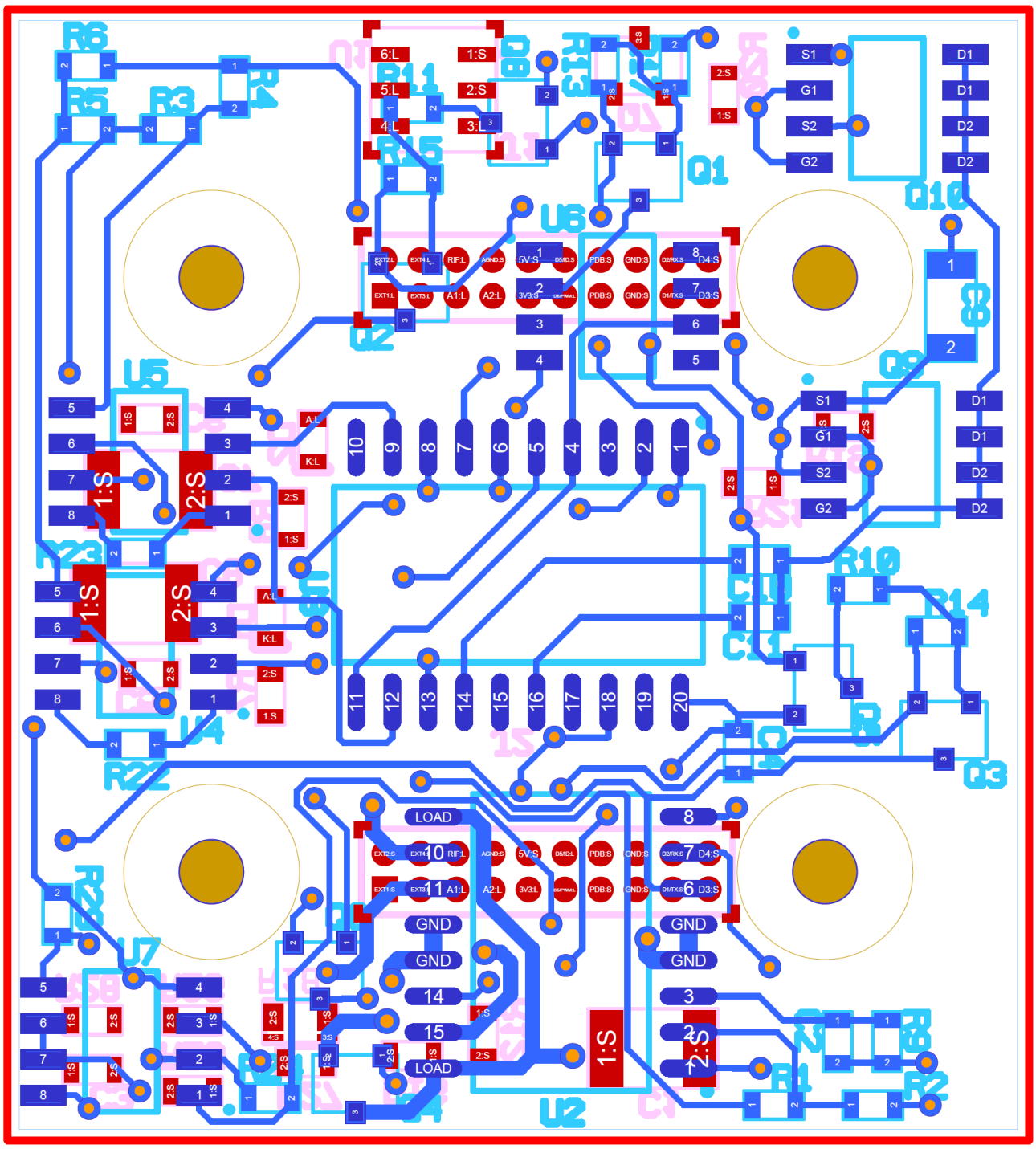
A.1. Top layer



A.2. Bottom layer

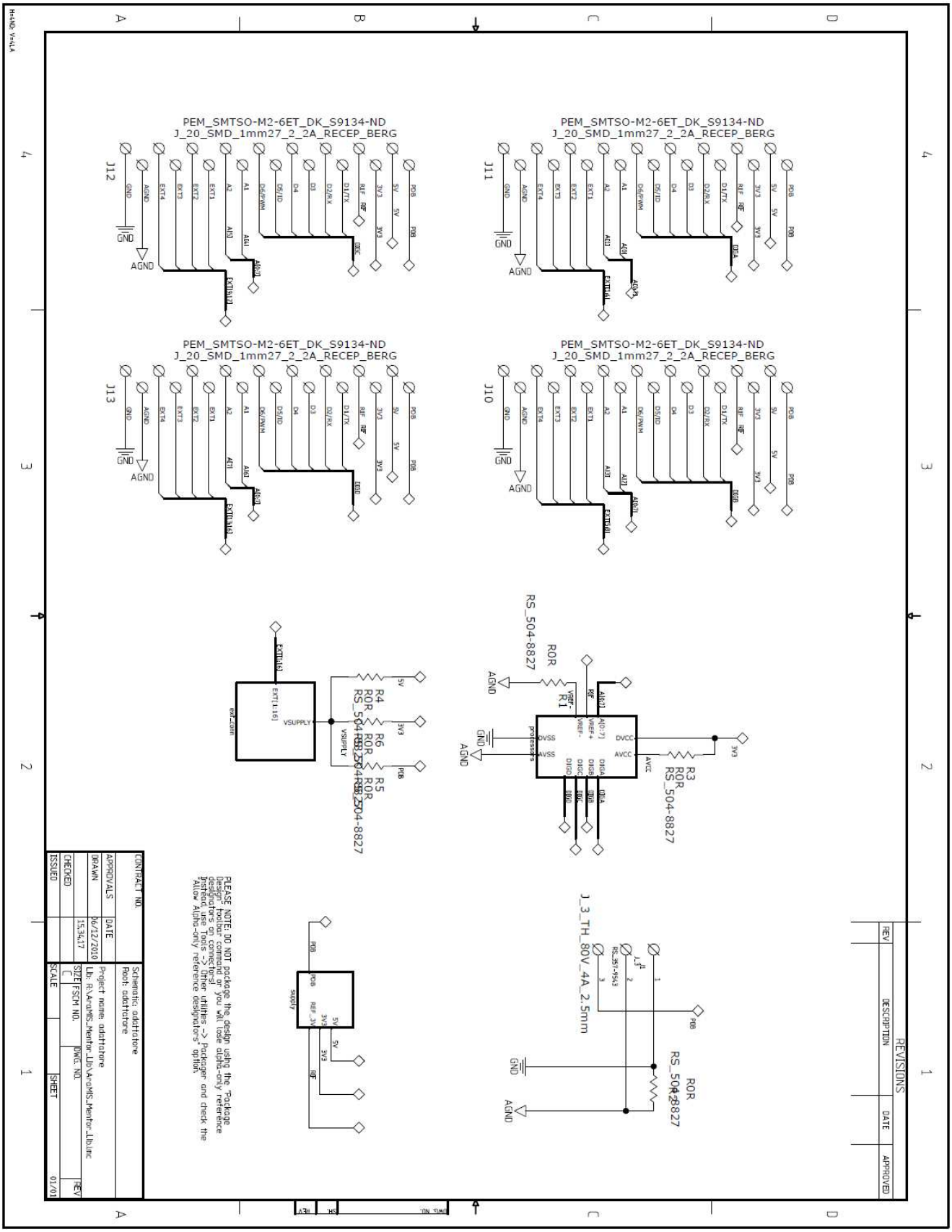


A.3. Sovrapposto

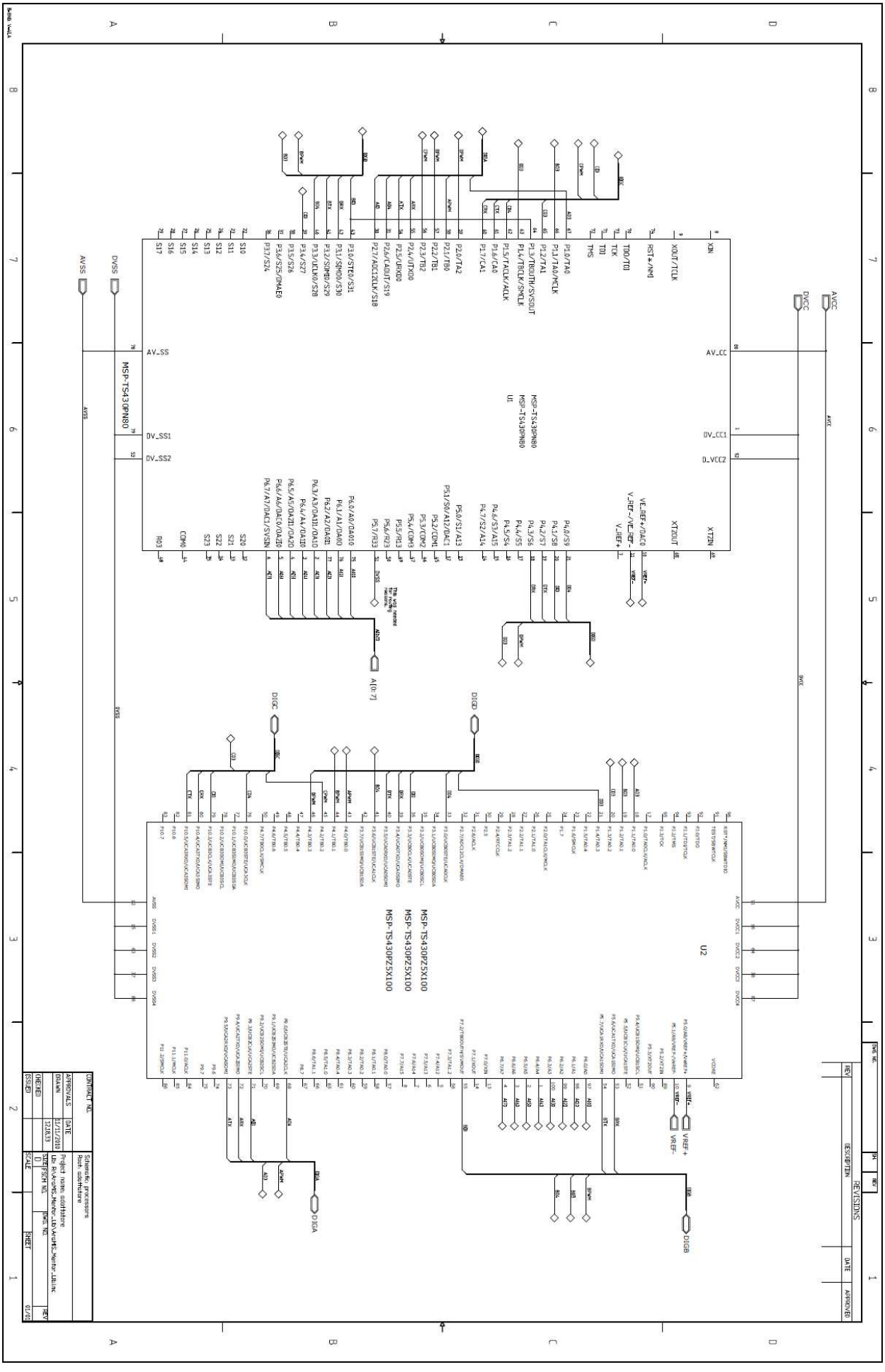


B. Schemi elettrici *Surface Connector Adapter (V1)*

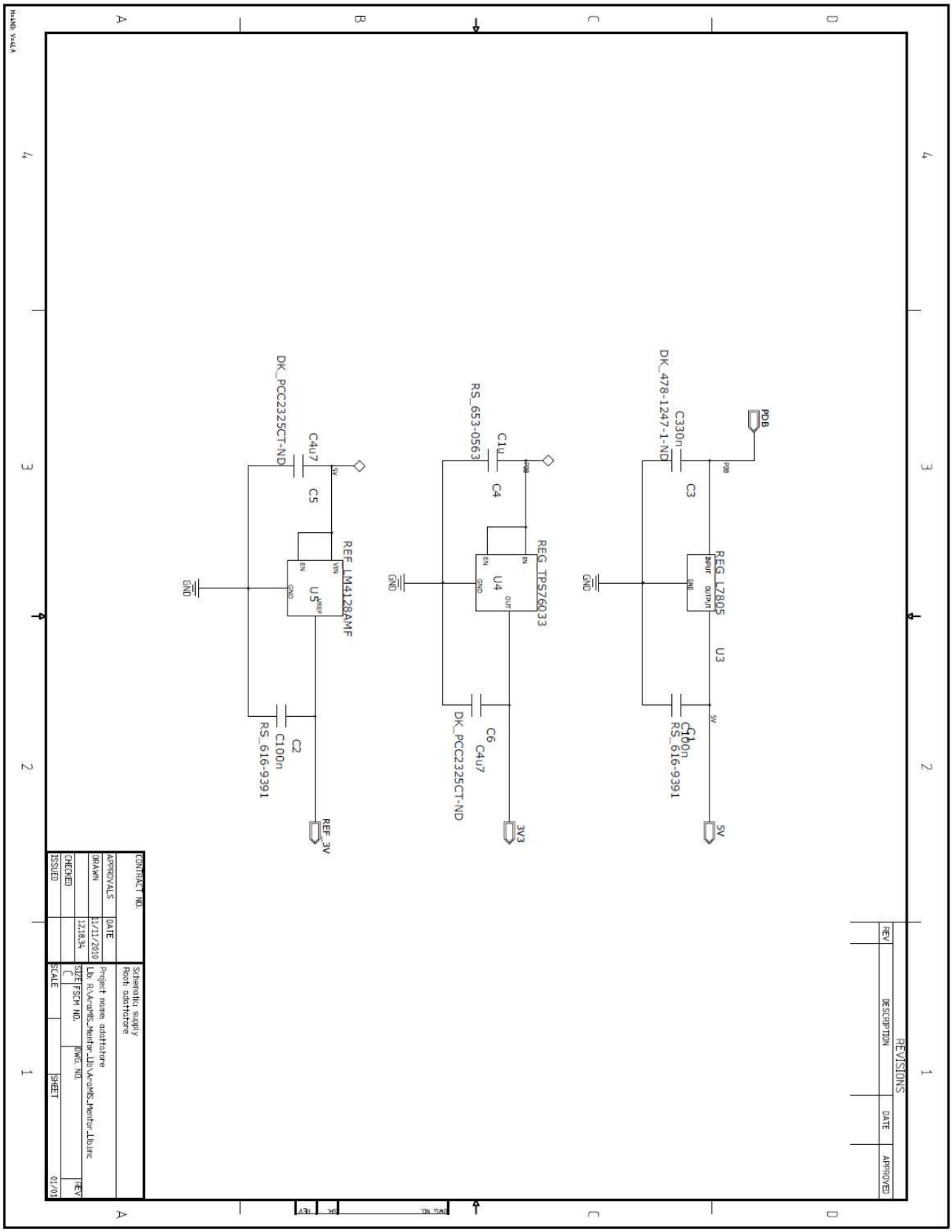
B.1. Schema complessivo



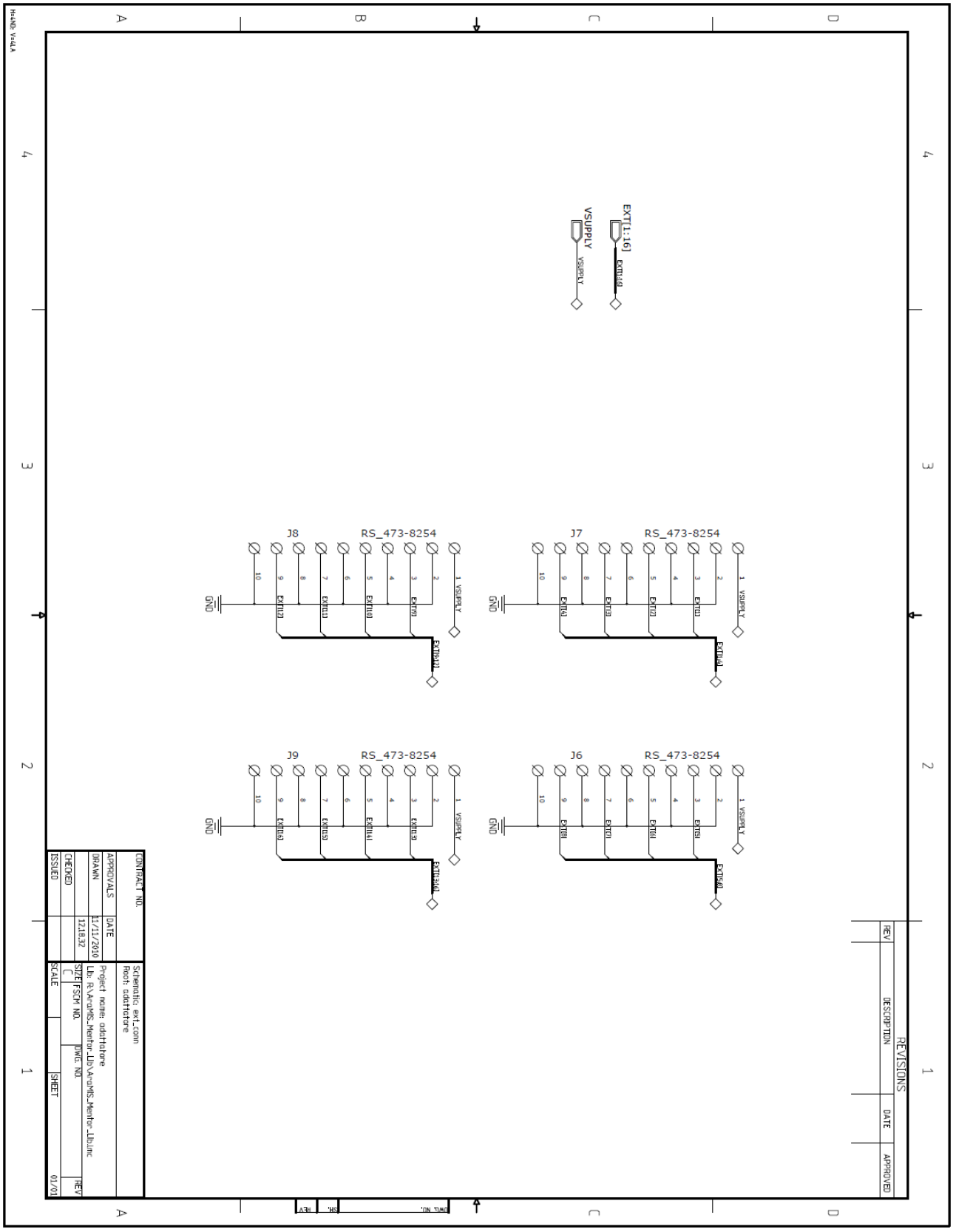
B.2. Schema elettrico blocco *processori*



B.3. Schema elettrico alimentazioni



B.4. Schema elettrico connettori esterni



Bibliografia

[1] Satellite Classification. http://centaur.sstl.co.uk/SSHP/sshp_classify.html

[2] Jake A. Schaffner. *The electronic system design, analysis, integration and constriction of the cal poly state university cp1 cubesat.* Technical report, Cal Poly State University, 2002.

[3] Maurizio Tranchero. *Progetto e realizzazione del satellite PiCPoT: Processore di bordo.* Master's thesis, Politecnico di Torino, 2005.

[4] Francesco Vico. *Satellite modulare: Sviluppo e collaudo di un sottosistema di controllo d'assetto per satellite modulare AraMiS.* Master's thesis, Politecnico di Torino, 2008.

[5] Stefano Speretta, Leonardo M. Reyneri, Claudio Sansoè, Maurizio Tranchero, Claudio Passerone, and Dante Del Corso. *"Modular architecture for satellites". In IAC-07-B4.7.09.* Politecnico di Torino, 2007.

[6] Danilo Roascio. *Sottosistema di comunicazione tollerante ai guasti per satellite modulare AraMiS.* Master's thesis, Politecnico di Torino, 2008.

[7] Kendall Scott. *"UML explained"*. Milano: Addison-Wesley Longman Italia Editoriale, 2001.

[8] OMG: Sito ufficiale per la definizione dello standard UML. www.omg.org

[9] Visual Paradigm: sito ufficiale. http://www.visual-paradigm.com/

[10] NASA. *Space radiation effects on electronic componets in low-earth orbit.* Preferred reliability Practices no. PD-ED1258, 1996.

[11] R.T. Merrill, M.W. McElhinny, PH.L. McFadden. *The Magnetic Field of the Earth - Paleomagnetism, the Core and the Deep Mantle*. San Diego, California: Academic Press, 1996.

[12] Mappa dell'intensità di campo geomagnetico nel 2009 http://www.ngdc.noaa.gov/geomag/WMM/WMM2010BetaSoft.shtml

[13] SPENVIS. http://www.spenvis.oma.be

[14] Enciclopedia in rete. *http://it.wikipedia.org/wiki/Pagina_principale*

[15] Datasheet sensore magnetico Honeywell HMC1002. http://www.ssec.honeywell.com/magnetic/datasheets/hmc1001-2_1021-2.pdf

[16] Datasheet amplificatore operazionale Analog Devices AD623. http://www.analog.com/static/imported-files/Data_Sheets/AD623.pdf

[17] Datasheet voltage reference Analog Devices REF02HSZ. http://www.analog.com/static/imported-files/Data_Sheets/REF02.pdf

[18] Datasheet driver bipolare Allegro MicroSystems A3953. http://www.allegromicro.com/en/Products/Part_Numbers/3953/3953.pdf

[19] Datasheet amplificatore operazionale National Semiconductor LM6142. http://www.national.com/ds/LM/LM6142.pdf

[20] Datasheet microcontrollore Texas Instruments MSP430F5437. http://focus.ti.com/lit/ds/symlink/msp430f5437.pdf

[21] Datasheet microcontrollore Texas Instruments MSP430FG439. http://focus.ti.com/lit/ds/symlink/msp430fg439.pdf

[22] MSP430 USB-Debug-Interface MSP-FET430UIF. http://focus.ti.com/docs/toolsw/folders/print/msp-fet430uif.html