POLITECNICO DI TORINO

III Facoltà di Ingegneria
Corso di Laurea in Ingegneria Elettronica

Tesi di Laurea

Sviluppo del software di controllo di un satellite universitario

Relatori:
Prof. Leonardo Reyneri
Prof. Dante Del Corso

Candidato:
Graziano Cordella

Novembre 2006

Sommario

Per la realizzazione del primo satellite universitario del Politecnico di Torino, PiC-PoT, è stato necessario valutare diversi fattori sulla base dei quali sviluppare parallelamente ogni sottosistema, in modo da raggiungere il risultato voluto. Il lavoro svolto in questa tesi ha riguardato la scheda ProcB. Date le specifiche progettuali si è passati all'analisi delle stesse per valutarne la fattibilità e identificarne nel dettaglio i vincoli progettuali, eseguendo un' integrazione HW e SW nella scheda che ha riguardato gli alimentatori a 3,3V per ProcB e Payload, a 12V per le telecamere e le rispettive protezioni da latch-up, un controllo della ruota d'inerzia, la gestione di carica delle batterie, la ricezione e la decodifica di telecomandi provenienti da terra con succesiva trasmissione delle telemetrie e delle foto scattate dalle telecamere di bordo. Questo progetto ha avuto inoltre un grande valore in termini di ricerca e di interazione tra persone che si sono occupate dei diversi sottosistemi del progetto, producendo risultati che costituiranno il punto di partenza per i successivi progetti satellitari.

Indice

Elenco delle figure

Ai miei genitori

Capitolo 1

PiCPoT

Nel gennaio del 2004 il Politecnico di Torino ha iniziato a progettare un nanosatellite universitario sulla scia di molte altre università europee. Questo progetto chiamato PiCPoT, acronimo di Piccolo Cubo del Politecnico di Torino, è un progetto interdipartimentale che ha coinvolto docenti, ricercatori, dottorandi e tesisti del Dipartimento di Elettronica, del Dipartimento di Ingegneria Aerospaziale e del Dipartimento di Energetica del Politecnico di Torino.
Il progetto del satellite si è sviluppato seguendo i requisiti imposti dalle specifiche Cubesat.
Lo scopo principale è quello di verificare il funzionamento dei componenti COTS (*Components Off The Shelf*) nello spazio, trasmettere le letture dei sensori di bordo alla stazione di terra, scattare fotografie a bassa risoluzione della superficie terrestre. Cubesat significa satellite di forma cubica ed è nel contempo una filosofia progettuale. E' uno standard per picosatelliti sviluppato nel 2001 dal professore Robert Twiggs, docente alla Stanford University, USA, in collaborazione con la Space Systems Development Laboratory (SSDL) della Stanford University e la California Polytechnic State University, USA, per permettere alle Università che intendono partecipare a questa sperimentazione di realizzare il proprio satellite e di mandarlo nello spazio a costi contenuti. I requisiti di progetto imposti sono i seguenti :

- forma cubica con lato di 13 cm;
- massa di circa 2.5 Kg;
- potenza media non superiore a 1.5 W;
- orbita LEO (Low Earth Orbit);
- almeno 90 giorni di vita.

Figura 1.1. PicPot

1.1 L'ambiente spaziale

PiCPoT è progettato per orbitare tra LEO(Low Earth Orbit) e MEO(Medium Earth Orbit) e, più precisamente, ad una quota compresa tra 600km ed 800km. Per questo motivo si devono tenere in considerazione tutte le caratteristiche dell'ambiente di lavoro in modo da garantirne il funzionamento in ogni condizione esterna. I problemi che possono verificarsi, quelli che mettono maggiormente a rischio le funzionalità del satellite sono:

- la dinamica della temperatura cui il satellite è sottoposto;
- la possibilità di single event latch-up dovuti alle radiazioni ionizzanti;
- la possibilità di single event up-set.

1.1.1 La temperatura

Ad altezze così elevate, la concentrazione d'aria molto bassa comporta una notevole difficoltà nel dissipare il calore prodotto all'interno dei circuiti, visto che viene meno

la dissipazione per convezione. A questo, si somma il problema che il satellite si trova esposto ad un maggiore irraggiamento solare dovuto all'assenza di nubi.
Viste le modeste dimensioni del satellite, le facce non rivolte al sole, pur non essendo colpite dalla luce, non riescono a raffreddarsi abbastanza velocemente prima di venir nuovamente esposte alla radiazione solare. Tuttavia la temperatura all'interno del cubo dovrebbe essere compresa tra i -10 °C ed i 70 °C.

1.1.2 Single event latch up

Il single event latchup è un errore dovuto ad un percorso anomalo di corrente, più nello specifico esso causa l'attivazione di strutture parassite all'interno di una struttura CMOS (si veda per maggior chiarezza la figura 1.2), questa struttura parassita una volta attivata, da vita ad una reazione positiva. Una volta verificatosi questo errore, se non viene immediatamente disattivata l'alimentazione, il dispositivo viene irrimediabilmente compromesso.

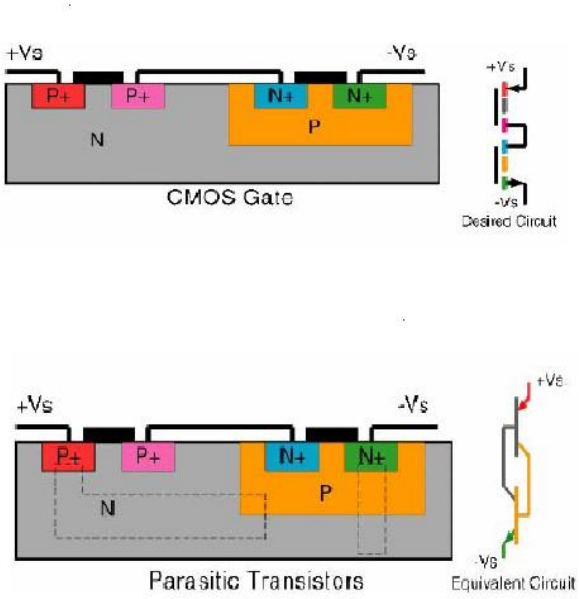


Figura 1.2. Struttura transistor parassiti

1.1.3 Single event up-set

Quando si verifica un single event upset, quello che accade è che ioni pesanti depositano una tale quantità di energia su un elemento bistabile, da causarne il cambiamento di stato logico. Tali effetti sono facilmente osservabili in memorie non protette. La figura 1.3, mostra un esempio di SEU su un dispositivo elettronico.

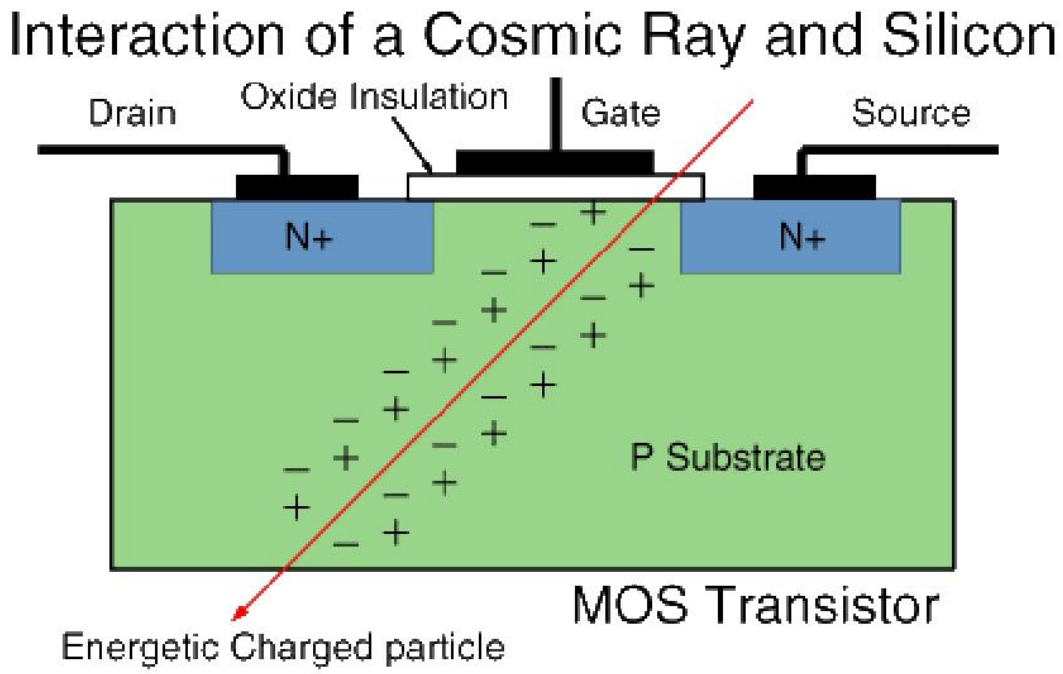


Figura 1.3. SEU su dispositivo elettronico

La cella di memoria è dimensionata di modo da avere due stati stabili rappresentanti lo zero logico e l' uno logico. Il passaggio di una particella carica in una cella di memoria, cosi fatta, può comportare la scrittura di un uno logico dove prima era presente uno zero logico.

1.2 I sottosistemi

1.2.1 Vista esterna

PiCPoT è un satellite di forma cubica dotato quindi di sei facce (F1, F2, F3, F4, F5, F6) quadrate di 13cm di lato in lega di alluminio tipo 5000 Al-Mn. Queste ospitano cinque pannelli solari due antenne da (437 Mhz e 2.44 Ghz), tre fotocamere (T1,

T2, T3), due Kill-switch (K1,K2) ed un connettore di test (CT) necessario per il controllo dell'elettronica di bordo dopo l'assemblaggio finale del satellite. La faccia (F0), quella che sarà rivolta verso la superficie terrestre, è la piastra di interfaccia al lanciatore ed ospita l'ottica delle fotocamere, le due antenne, e sugli spigoli esterni i due interruttori killswitch.
Le tre fotocamere con asse ottico e direzione di vista parallela a X positivo sono all'interno del satellite, con la lente a filo di F0. Ciascuna telecamera ha una diversa lunghezza focale e risoluzione di 512 linee per 582 colonne:

- **T1**: lunghezza focale, campo visivo totale sulla diagonale 3.6 mm +/- 68°C, corrispondente a 750 Km sulla terra e ad una risoluzione a terra di 800 m/pixel;
- **T2**: lunghezza focale 6 mm, campo visivo totale sulla diagonale +/- 42°C, corrispondente a 450 Km sulla terra e ad una risoluzione a terra di 500 m/pixel;
- **T3**: lunghezza focale 16 mm, campo visivo totale sulla diagonale +/- 17°C, corrispondente a 170 Km sulla terra e ad una risoluzione a terra di 180 m/pixel.

Essendo l'asse ottico parallelo a X, il centro del campo visivo delle telecamere non dipende dalla rotazione di PICPOT intorno al suo asse X (il probabile asse di spin del satellite). Nel satellite vengono inoltre utilizzati due tipi di celle solari, interconnesse e assemblate in cinque pannelli laminati e incorporanti un sensore di temperatura ciascuno:

- versione a 2V nominali (tensione complessiva, ai morsetti, a vuoto, in pieno illuminamento), detta tipo Chiesa: pannelli P1, P3 e P5; corrente massima 500mA.
- versione a 5V nominali (tensione complessiva, ai morsetti, a vuoto, in pieno illuminamento), detta tipo Maggiore: pannelli P2 e P4; corrente massima 200mA.

1.2.2 Vista interna

PICPOT é alimentato da 6 gruppi di batterie ricaricabili (batterie secondarie), disposte fra i pannelli solari e le schede elettroniche, come indicato in figura. Il connettore di test servirà anche per il caricamento delle batterie prima del lancio.
B1, B2, B4, B5: due celle Li-P in serie, per un totale di 7.2V nominali; capacità di 1500mAh; modello Kokam. Le batterie saranno affiancate lungo la faccia più estesa, ed inframmezzate dal sensore di temperatura, nonché circondate da uno strato di gomma di protezione; *B3, B6*: sei celle Ni-Cd in serie, per un totale di 7.2V nominali; capacità di 1000mAh; modello SANYO High Capacity. Le batterie saranno affiancate lungo la faccia piú piccola, ed inframmezzate dal sensore di temperatura, nonché circondate da uno strato di gomma di protezione.

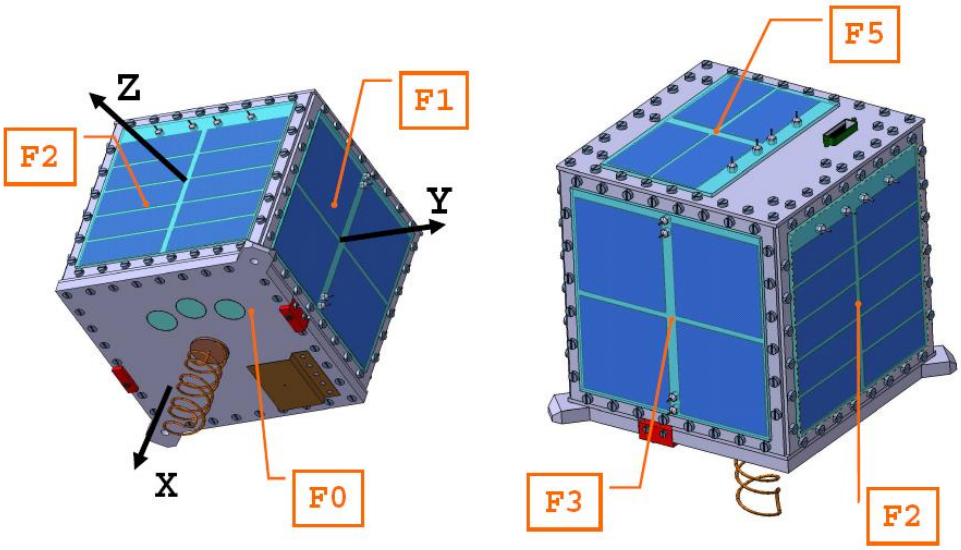


Figura 1.4. PiCPoT vista esterna

L'utilizzo dei due kill-switch è dovuto alla richiesta del lanciatore. Essi verranno connessi in parallelo, per assicurarsi contro la rottura di uno di essi. I killswitch intercetteranno la massa comune delle batterie. Questo dovrebbe togliere anche la tensione ausiliaria ai caricabatterie, bloccandoli ed annullando quindi tutte le tensione del satellite.
PICPOT non ha un controllo d'orbita. La sua orbita è determinata esclusivamente dalla sua posizione e velocità nel momento del distacco dal lanciatore. L'orbita viene tenuta sotto stretto controllo dal servizio americano NORAD, che periodicamente rileva la posizione del satellite tramite radar e rende disponibili le effemeridi per i successivi tre giorni attraverso il portale Internet. L'assetto del satellite è impostato da quattro magneti permanenti posizionati all'interno del satellite in corrispondenza degli spigoli paralleli all'asse X (quindi ortogonali a F0 e F5). Il loro Sud magnetico prossimo a F0 (direzione positiva di X) e il Nord magnetico prossimo a F5 (direzione negativa di X) dovrebbero favorire l'allineamento del satellite nella direzione del Nord magnetico terrestre.
L'assetto è inoltre controllato, attorno all'asse di spin, da una ruota d'inerzia il cui asse è parallelo all'asse X. La direzione positiva di rotazione della ruota è quella dall'asse Y positivo verso l'asse Z positivo, questo causa una rotazione inversa del satellite. La ruota d'inerzia viene mossa da un motore brushless MAXON tipo EC-flat32, da 9V, 6W nominali. Il motore ha tre avvolgimenti con connessione a stella e tre sensori di Hall. Il motore verrà controllato ad anello aperto, utilizzando i sensori

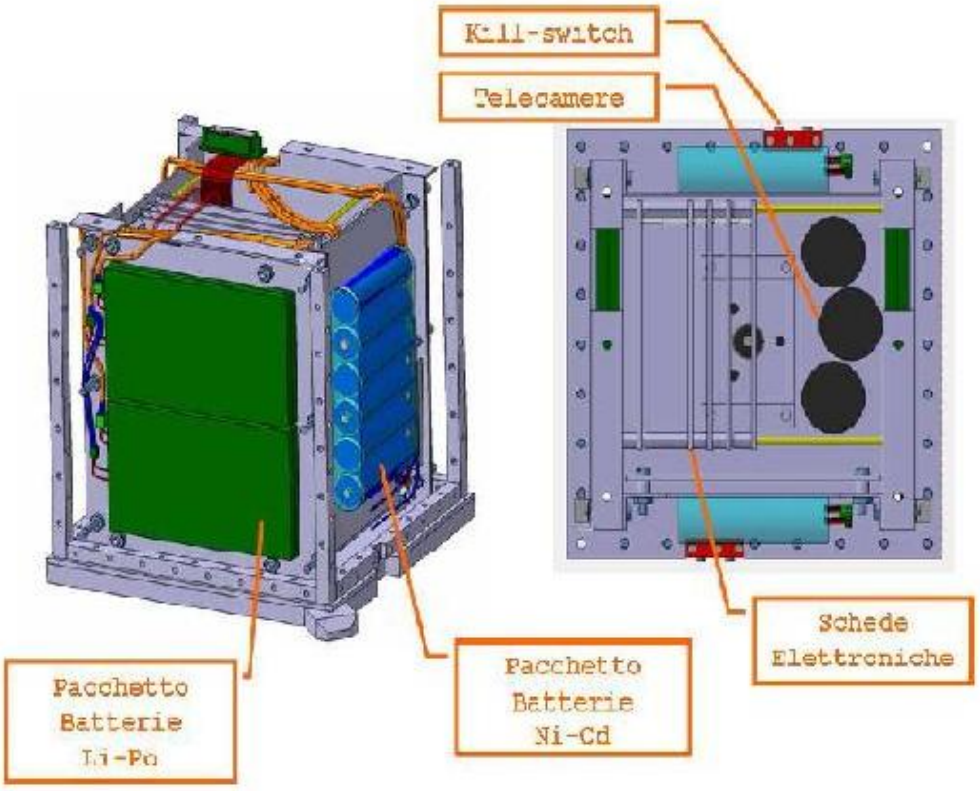


Figura 1.5. PiCPoT vista interna

di Hall, attraverso la scheda ProcB, ed attivato in base ai telecomandi ricevuti da terra.

I processori presenti all'interno di PICPOT sono tre:

- ProcA: per la gestione di bordo (housekeeping), associato al canale di comunicazione a 437MHz (TxRxA). E' stato utilizzato un Chipconn 1010 con al suo interno il modulatore/demodulatore a 437 MHz. Frequenza di clock 11 Mhz e tensione di alimentazione di 3.3V.

- ProcB: per la gestione di bordo (housekeeping), associato al canale di comunicazione a 2.4GHz (TxRxB). E' stato utilizzato un MSP430 a basso consumo della Texas Instrument con frequenza di clock 4 Mhz e tensione di alimentazione di 3.3V.

- Payload: per la gestione del payload (telecamere T1, T2, T3). E' stato utilizzato un Analog Devices BlackFINN, dotato d' interfacce dedicate per le telecamere, con frequenza di clock di 400Mhz e tensione di alimentazione 1.8V.

Il satellite è dotato di un certo numero di sensori per il monitoraggio dello stesso. I sensori vengono misurati e convertiti da ciascun processore ogni minuto e vengono trasmessi a terra come telemetria (TMH), automaticamente ogni minuto, inoltre vengono memorizzati per un tempo max di 120' (TMHE). Esiste un vettore di telemetria per ciascuna delle due frequenze di trasmissione.

PICPOT comunica a terra tramite due canali di comunicazione (per motivi di ridondanza), entrambi in banda amatoriale con Protocollo APRS, rispettivamente:

- ProcA: per la gestione di bordo (housekeeping), associato al canale di comunicazione a 437MHz (TxRxA). E' stato utilizzato un Chipconn 1010 con al suo interno il modulatore/demodulatore a 437 MHz. Frequenza di clock 11 Mhz e tensione di alimentazione di 3.3V.

- il *ProcA* a 437.541 KHz per TC1, bit-rate di 9,600 bit/s; modulazione di frequenza 2-FSK con deviazione di 6kHz;

- il *ProcB* a 2.440.000 KHz per TC2, bit-rate di 10,000 bit/s; modulazione di frequenza GFSK con deviazione di 125 kHz;

La trasmissione dei telecomandi avverrà per mezzo di un protocollo ad-hoc, con unique world di 40 bit, preceduta da preambolo e seguita da telecomando di 12+27 bit codificato con un codice a parità dispari; il protocollo non verrà pubblicizzato, per evitare comandi indesiderati da parte di esterni.

1.3 Le schede

Le schede elettroniche all'interno del satellite sono sei è precisamente sono: PowerSupply, PowerSwitch, ProcA, ProcB, Payload e TxRx. Esse sono direttamente o indirettamente tutte in comunicazione tra di loro ad eccezione di ProcA e ProcB che non hanno nessun segnale in comune. Adesso si passerà a illustrare le singole schede.

1.3.1 PowerSupply

La scheda PowerSupply ha il duplice compito di mantenere cariche le batterie di PICPOT e di monitorare lo stato elettrico e termico dei pannelli, dei caricabatterie e delle batterie stesse. La scheda PowerSupply ospita:

- cinque caricabatterie che ricevono potenza da altrettanti pannelli solari (esterni al satellite); i caricabatterie richiedono una tensione ausiliaria proveniente dalle batterie o, tramite due diodi, dai pannelli P2 e P4;

- sensori di corrente media e tensione media erogata dai cinque pannelli; questi sono alimentati dalla scheda PowerSwitch, quando richiesto da ProcA, ProcB;
- sensori di corrente di carica/scarica e tensione delle sei batterie; questi sono alimentati dalla scheda PowerSwitch, quando richiesto da ProcA, ProcB;
- i circuiti di condizionamento per cinque sensori NTC (esterni al satellite) che rilevano la temperatura dei pannelli solari; questi sono alimentati quando richiesto da ProcA, ProcB;
- i circuiti di condizionamento per sei sensori NTC (posizionati sulle batterie) che rilevano la temperatura delle batterie; questi sono alimentati quando richiesto da PowerSwitch, ProcA, ProcB;
- il circuito di condizionamento per un sensore NTC che rileva la temperatura in vicinanza dei caricabatterie; questi sono alimentati quando richiesto da ProcA, ProcB;
- due multiplexer (muxA, comandato da procA e muxB, comandato da procB) per la lettura sequenziale delle temperature, delle tensioni e correnti elencate sopra;
- due circuiti digitali (per ridondanza) e sei quadruple di interruttori PMOS per la selezione dell'unica batteria da caricare; questi sono sempre autoalimentati;
- due alimentatori non regolati per alimentare i cinque caricabatterie e i suddetti circuiti digitali, suddivisi in due gruppi, per ridondanza; e i circuiti di condizionamento (questi ultimi vengono alimentati dalla scheda PowerSwitch, su richiesta delle schede PowerSwitch, procA e procB.

Il satellite è dotato di un certo numero di sensori per il monitoraggio dello stesso. I sensori vengono misurati e convertiti da ciascun processore ogni minuto e vengono trasmessi a terra come telemetria (TMH), automaticamente ogni minuto, inoltre vengono memorizzati per un tempo max di 120' (TMHE). Esiste un vettore di telemetria per ciascuna delle due frequenze di trasmissione.

PICPOT comunica a terra tramite due canali di comunicazione (per motivi di ridondanza), entrambi in banda amatoriale con Protocollo APRS, rispettivamente:

- ProcA: per la gestione di bordo (housekeeping), associato al canale di comunicazione a 437MHz (TxRxA). E' stato utilizzato un Chipconn 1010 con al suo interno il modulatore/demodulatore a 437 MHz. Frequenza di clock 11 Mhz e tensione di alimentazione di 3.3V.
- il ProcA a 437.541 KHz per TC1, bit-rate di 9,600 bit/s; modulazione di frequenza 2-FSK con deviazione di 6kHz;

- il ProcB a 2.440.000 KHz per TC2, bit-rate di 10,000 bit/s; modulazione di frequenza GFSK con deviazione di 125 kHz;

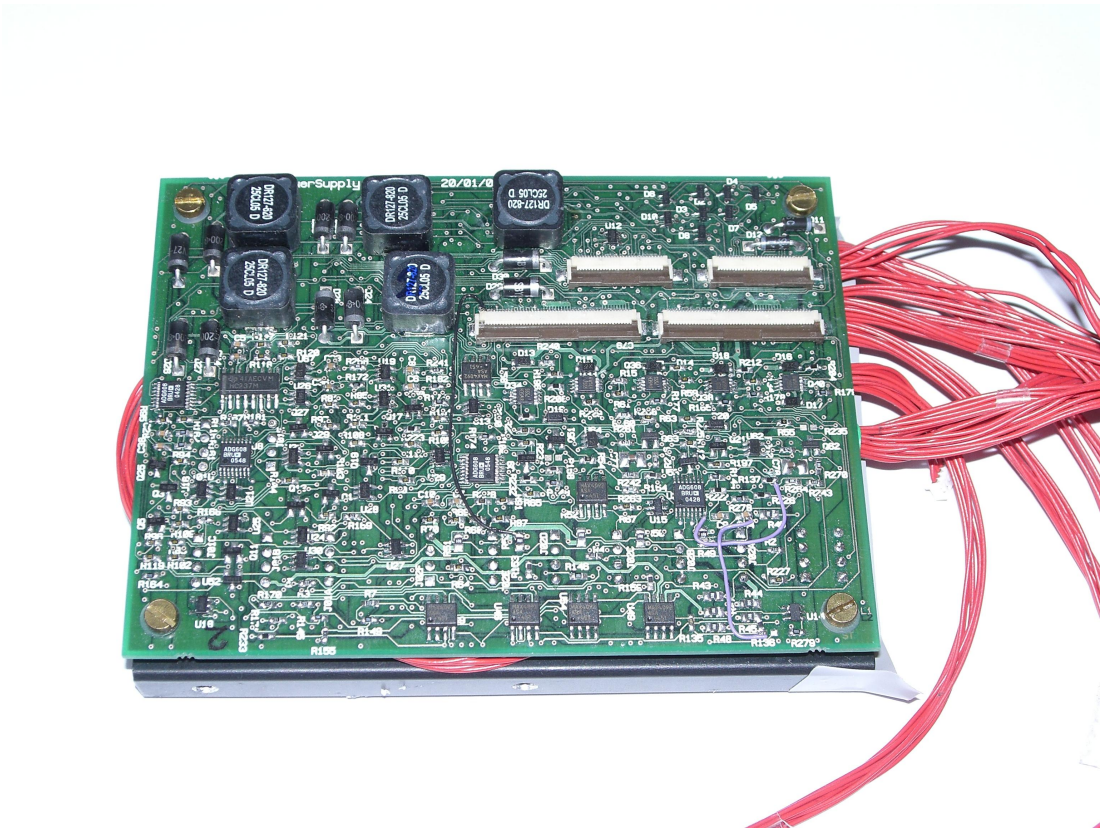


Figura 1.6. Scheda Power Supply

1.3.2 PowerSwitch

La scheda PowerSwitch ha molteplici compiti:

- definire quali batterie scaricare di volta in volta;
- generare le necessarie tensioni regolate per le altre schede, a partire dalle tensioni delle batterie,proteggendo, dove richiesto, da latchup;
- attivare, tramite l'alimentazione, le altre schede, su base temporale (ad esempio attivazione periodica dei processori e trasmettitori/ricevitori);
- mantenere una coppia di orologi di sistema (uno per ogni processore), sincronizzati fra loro, che periodicamente attivino i corrispondenti processori;

- conteggiare gli interventi di ciascun circuito di anti-latchup associato a ciascun alimentatore, inclusi quelli delle altre schede;
- condizionare e multiplare i sensori di tensione e corrente all'uscita degli alimentatori.

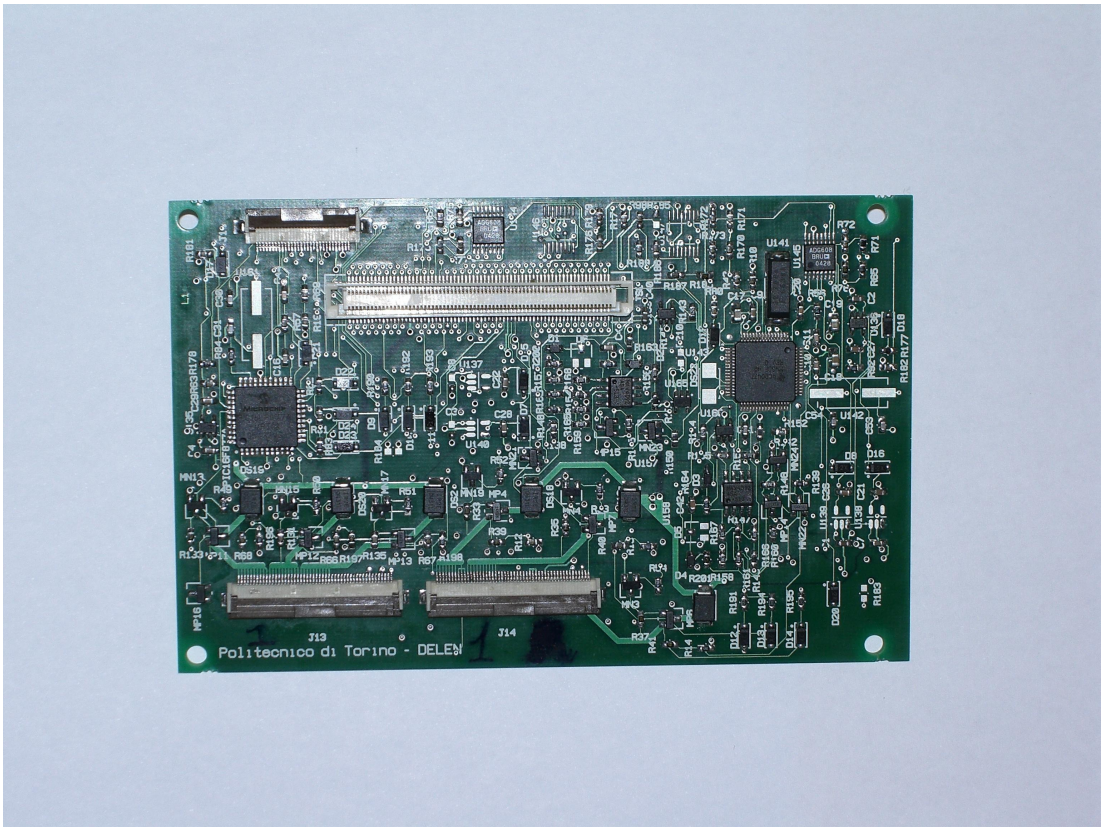


Figura 1.7. Scheda Power Sitch

1.3.3 ProcA, ProcB

Le due schede processore A e B sono funzionalmente molto simili. Si è scelto di duplicarle per motivi di ridondanza utilizzando quindi componenti di diversa natura e garantendo loro una indipendenza reciproca. Il procA usa Chipconn 1010 (core 8051 + modulatore/demodulatore a 435MHz) mentre il ProcB utilizza il microcontrollore della Texas MPS430F149 (solo processore, senza modulatore/demodulatore), questi svolgono le seguenti funzioni:

- ogni minuto vengono attivati automaticamente dalla scheda PowerSwitch;
- gestiscono tutti i sensori di bordo entro 4 secondi dall'accensione;

- assemblano il vettore di telemetria di housekeeping TMH contenente le letture dei sensori nel momento dell'accensione;
- comprimono l'informazione e la memorizzano nel vettore di telemetria estesa TMHE;
- attivano il ricevitore di telecomandi corrispondente;
- decodificano il telecomando, se questo è corretto lo interpretano, ed eseguono le funzioni ad esso associato, ovvero:
  - acquisizione dati,
  - gestione memoria,
  - impostazione dei timer,
  - attivazione fotocamere
  - avanzamento ruota di inerzia (solo Proc B)
  - comando di spegnimento inviando un segnale alla scheda Power Switch,
  - itemize

  altrimenti
  - inviano una telemetria (TMH) con la copia del telecomando errato
  - richiedono spegnimento a Power Switch

1.3.4 Payload

Compito della scheda Payload è quello di scattare fotografie, su richiesta da terra, comprimerle in formato JPEG, e inviarle, su richiesta, ad una delle schede procA e procB, che a loro volta le invieranno a terra.

La scheda Payload gestisce le tre telecamere, posizionate a fianco della scheda. Le telecamere verranno alimentate individualmente per il tempo strettamente necessario (max 3"), attraverso interruttori posizionati sulla scheda stessa. Il segnale d'uscita delle tre telecamere (formato PAL) viene multiplato per selezionare una sola telecamera alla volta, inviato ad un decoder digitale, che converte il segnale PAL in uno stream digitale a 8bit di intensità più 4+4 bit di crominanza. Lo stream digitale viene acquisito direttamente dal processore (un BlackFINN della AMD), tramite una sua porta dedicata, e trasferito in DMA alla memoria dati esterna al processore. Viene acquisito un solo fotogramma alla volta (funzione di fotocamera).

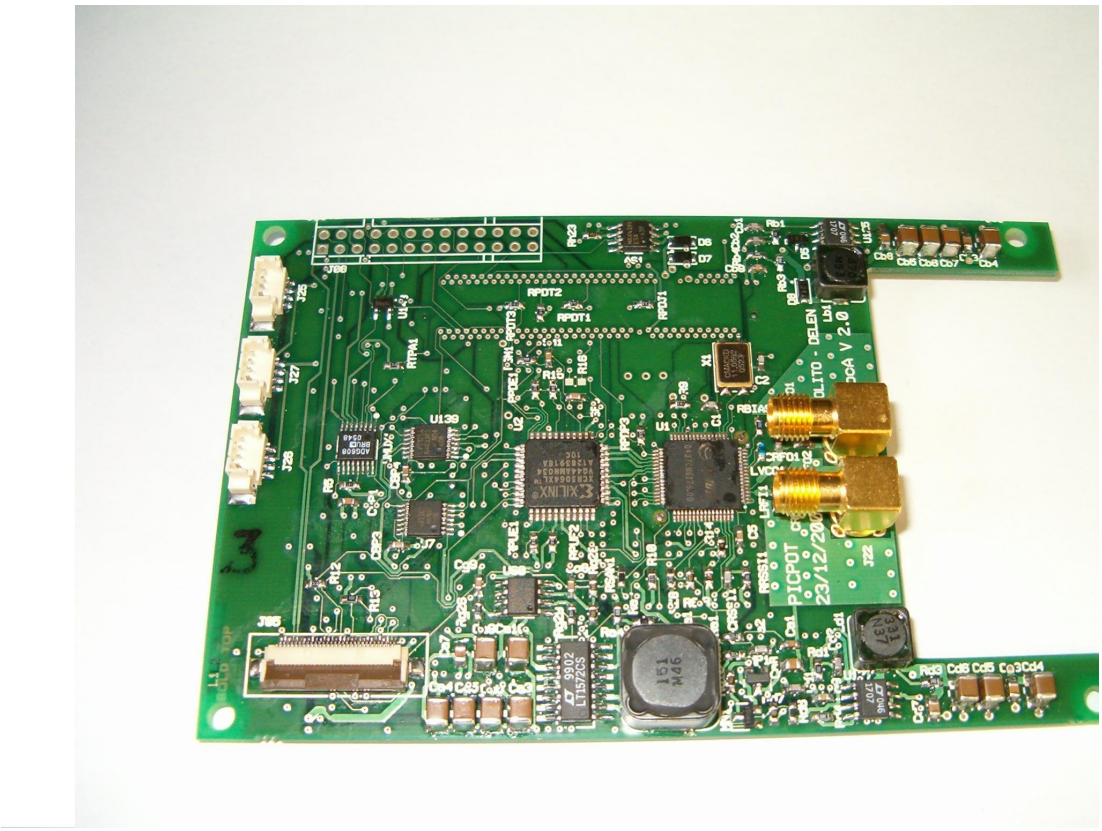


Figura 1.8. Proc A

L'immagine viene poi spezzata in 3x3 blocchi come si nota in figura 1.10 (numerati da 1, in alto a sinistra, 3, in basso a sinistra, fino a 9, in basso a destra) e poi ogni blocco viene compresso in formato JPEG via software e i 9 blocchi compressi memorizzati in una di 5 zone della memoria dati, come richiesto dal telecomando. Si reputa che il tempo di compressione totale sia inferiore a 30s, e che la dimensione dell'immagine compressa sia max 5 Kbyte.

Il funzionamento della scheda Payload sarà il seguente, nell'ordine:

1. Su richiesta di una delle schede procA o procB la scheda Payload viene alimentata;

2. il processore acquisisce, via SPI, il comando dalle schede procA e procB e lo interpreta.

3. Se è in grado di attuarlo, risponde, tramite la stessa SPI, con un segnale di conferma.

4. Se sta già eseguendo un comando richiesto dall'altro processore, risponde negativamente;

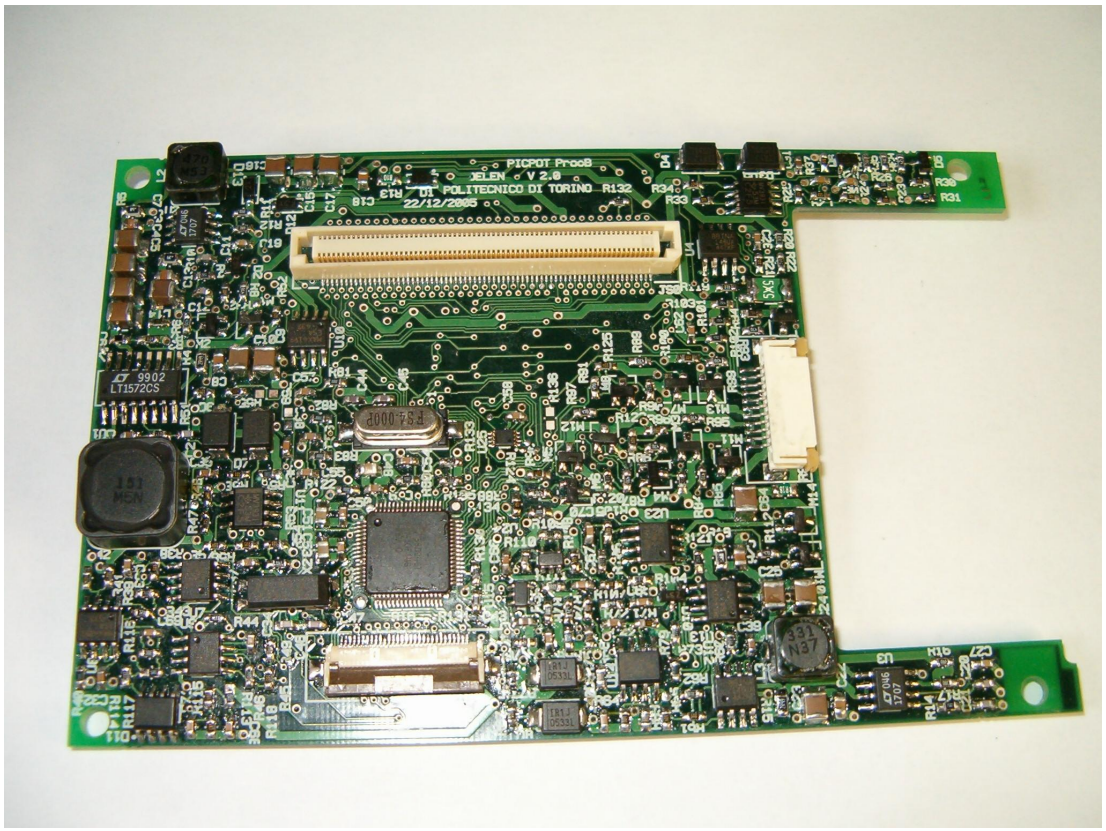


Figura 1.9. Proc B

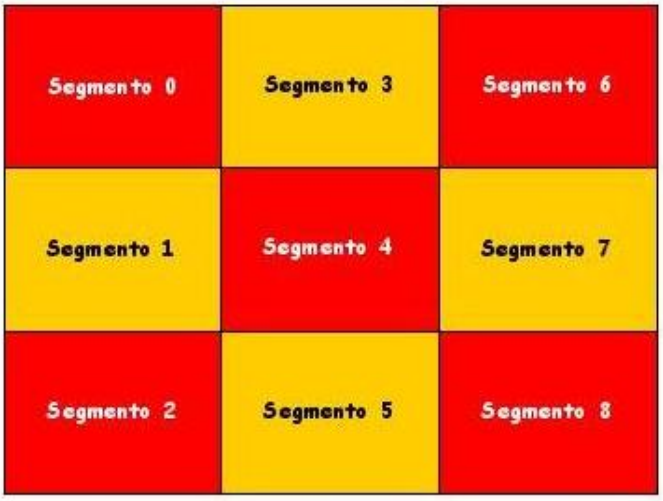


Figura 1.10. Divisione foto

successivamente, in alternativa:

1. alimenta una delle tre telecamere (T1 o T2 o T3); scatta una foto; spegne la telecamera;divide l'immagine in 9 blocchi, li comprime e li memorizza nella memoria;

2. invia l'immagine compressa al processore procA o procB (a seconda del richiedente), attraverso la corrispondente interfaccia (UART o SPI), terminata da un segnale di conferma.

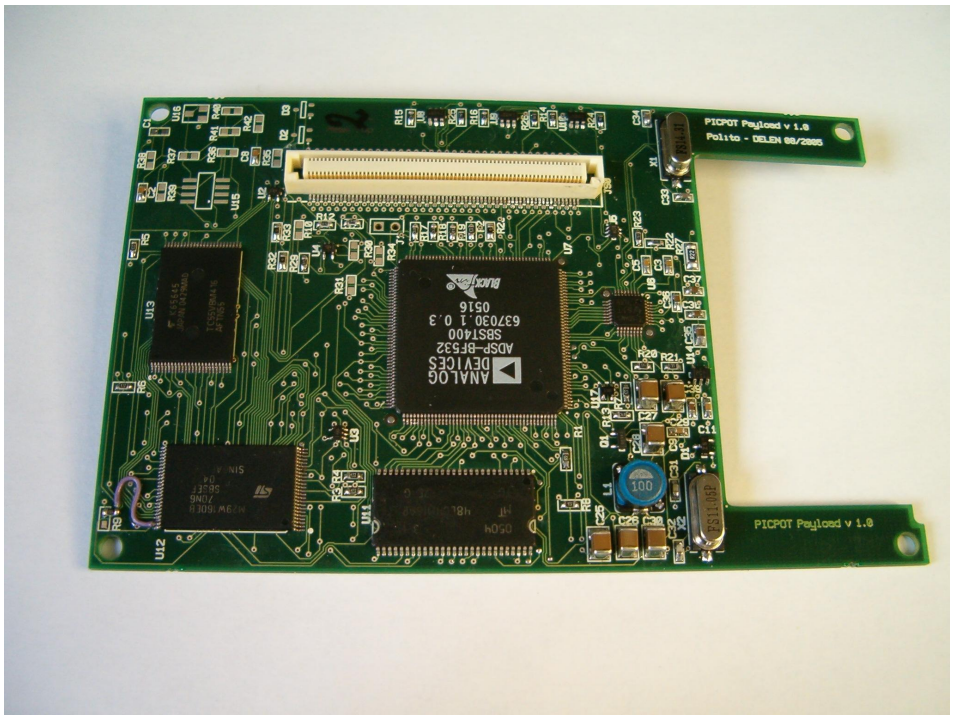


Figura 1.11. Scheda Payload

1.3.5 TX-RX

La scheda Tx-Rx ha il compito di far comunicare il satellite con la stazione di terra. Sono previsti due canali di comunicazione su due distinte bande amatoriali (437Mhz e 2.4Ghz). Ogni canale è half-duplex (trasmissione e ricezione mutuamente esclusive).

La scheda è composta da:

1. un amplificatore di potenza sulla banda 437MHz, che riceve un segnale a 0dBm già modulato in frequenza dalla scheda procA, e fornisce in antenna un segnale di 30dBm;

2. uno switch d'antenna 437MHz, per connettere l'antenna all'amplificatore di potenza o al demodulatore presente sulla scheda procA;
3. un modulatore a 2.4GHz (Chipconn 2400), che riceve la sequenza di bit dalla scheda procB;
4. un amplificatore di potenza sulla banda 2.4GHz, che fornisce in antenna un segnale da 23dBm;
5. un demodulatore a 2.4GHz (Chipconn 2400), che fornisce una sequenza di bit alla scheda procB;
6. un amplificatore a basso rumore (LNA) a 2.4GHz, per avere una sensibilità di -118dBm;
7. uno switch d'antenna 2.4GHz, per connettere l'antenna all'amplificatore di potenza all'amplificatore a basso rumore.

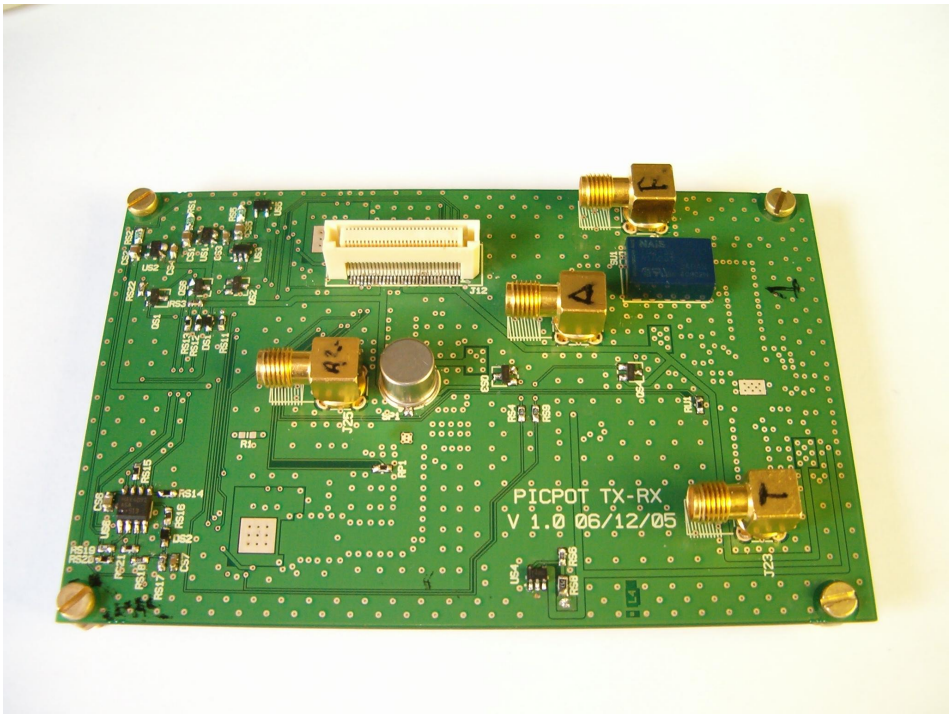


Figura 1.12. Scheda Tx-Rx

Capitolo 2

La scheda ProcB

2.1 L'Hardware della scheda

ULa scheda ProcB ha un ruolo centrale all'interno del sistema elettronico del satellite per quanto riguarda la gestione delle funzioni implementate sia sulla stessa scheda che nelle altre. Questo viene fatto attraverso quattro modi principali:

- acquisizione dei sensori analogici/digitali presenti su tutto il satellite;
- acquisizione delle immagini realizzate dalla scheda Payload;
- ricezione ed esecuzione dei telecomandi ricevuti da terra;
- invio di dati di telemetria e fotografie verso terra.

La scheda può essere schematizzata come riportato nella figura 2.1.

2.2 MSP430

Il cuore della scheda ProcB è l' MSP430 della Texas Instrument. La scelta è ricaduta in particolare su questo microcotrollore per il basso consumo che ha. In particolare è stato scelto l'MSP430F149 per l'elevato numero di I/O e la memoria disponibile. Le caratteristiche del microcontrollore sono:

- tensione di alimentazione da 1,8 a 3,6 V.
- consumo in standby 1,6uA
- 8 MHZ di frequenza massima
- 62 KB di memoria di cui 60 KB flash e 2KB RAM
- 48 pin di I/O riconfigurabili singolarmente

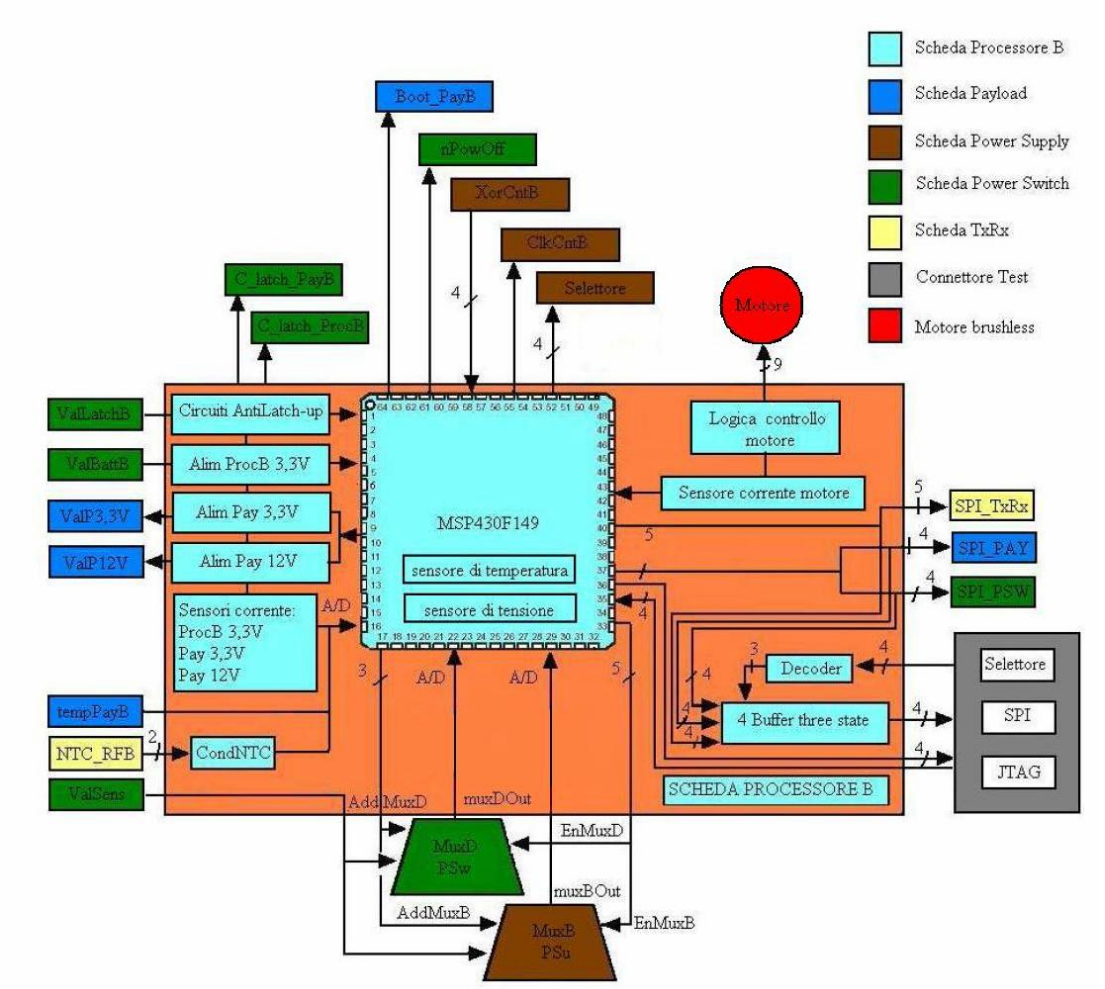


Figura 2.1. Schema a blocchi scheda ProcB

- watchdog a 16 bit
- 2 Timer, con 3 e 7 comparatori
- 2 USART utilizzabili ciascuna come SPI o UART
- 8 canali per il convertitore a 12 bit.

Altro vantaggio dell' MSP rispetto a altri microcontrollori è la possibilità della programmazione tramite JTAG ed in linguaggio C.
Per il core è stato utilizzato un quarzo da 4MHz, con le relative capacità di carico, sufficientemente veloce da eseguire il software richiesto e dai consumi inferiori rispetto all'utilizzo di un quarzo da 8 MHz. Vi era la possibilità di utilizzare il DCO interno, ma tale scelta non è stata fatta a causa della sua, non trascurabile,

variazione di frequenza al variare della temperatura. Una frequenza stabile è necessaria soprattutto nella comunicazione con il trasmettitore. Essa avviene in SPI e utilizzando un buffer con solo 32 bit. Il tentativo di riempire il buffer con più di 32 bit causerebbe la perdita dei primi bit inseriti, mentre un eccessivo ritardo nel riempire il buffer causerebbe lo svuotamento dello stesso e la successiva cessazione della trasmissione. Analoghi problemi si hanno in ricezione. E' stato inoltre utilizzato un secondo quarzo a 32768Hz, per l'utilizzo dei timer e per generare il PWM nel controllo motore, in quanto per i tempi richiesti, il divisore del quarzo ad alta frequenza non era sufficiente.
Per tale quarzo sono presenti all'interno dell' MSP due capacità di carico, con un valore di 12pF. Pur sufficienti per il quarzo selezionato, tuttavia sono state inserite nello schematico altre 2 capacità. Ciò allo scopo di poter montare altri quarzi richiedenti carichi maggiori, qualora il quarzo utilizzato non fosse più disponibile al momento dell'ordine, o risultasse non idoneo. Sono state anche inserite le capacità di bypass per le alimentazioni digitale ed analogica, oltre che per il riferimento interno. In aggiunta alla resistenza di pull-up per il reset indicata dal datasheet è stata inserita anche una capacità per evitare reset accidentali in caso di variazione della tensione di alimentazione.

2.3 Acquisizione A/D

L' MSP430F149 ha integrato al suo interno un convertitore A/D a 12 bit monotono. E' possibile selezionare come riferimenti interni 1.5V o 2.5V. Il valore convertito chiamato Nadc ha come fondo scala massimo 0FFFh corrispondente nel nostro caso a 1.5V.

Gli 8 canali esterni, più i 2 interni per il convertitore A/D, sono stati sufficienti, ed hanno permesso di non mettere un mux aggiuntivo. All'ingresso del mux interno, per i segnali provenienti dalle altre schede, ovvero i mux di PowerSupply e PowerSwitch, ed il sensore di temperatura di Payload e stato inserito un filtro RC per garantire un segnale più pulito.

Infatti, si possono avere dei disturbi dovuti ai regolatori switching (50KHz)e dalla componente a radiofrequenza (400MHz, 2,4GHz). La sua frequenza di taglio di tale filtro è:

ciò comporta ovviamente un aumento del tempo di sample dovuto sia al filtro esterno che all'MSP.

dove: RI = 2kΩe CI = 40 pF sono interne all' MSP.

Si ha per una conversione di 12 bit:

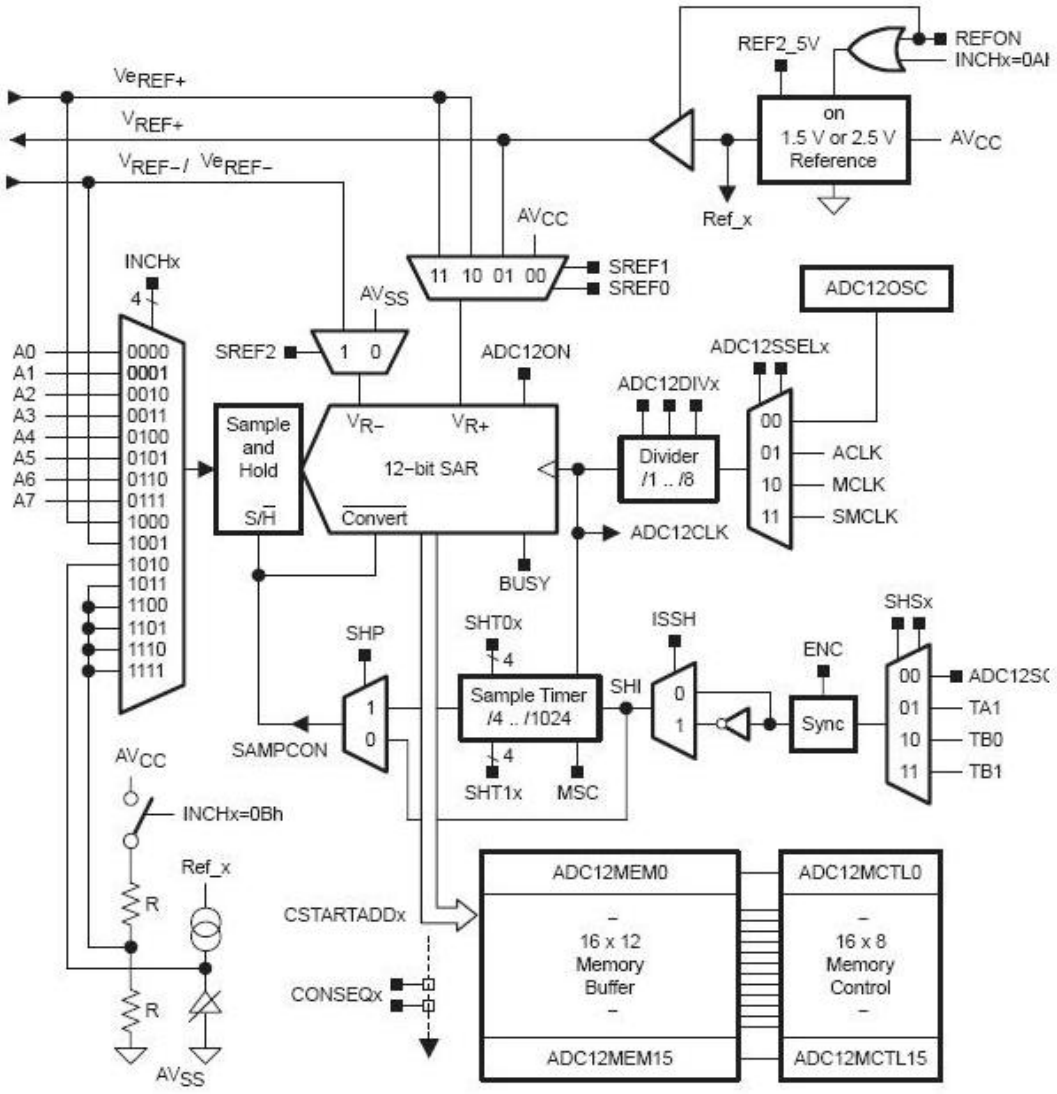


Figura 2.2. Convertitore A/D

$$f_t = \frac{1}{2\pi\tau} = \frac{1}{2 \times \pi R_{\mathrm{ADC}} \cdot C_{\mathrm{ADC}}} = \frac{1}{6{,}28 \times 100\,\Omega \times 100\,\mathrm{nF}} = \frac{0{,}16}{10\,\mu\mathrm{s}} = 16\,\mathrm{kHz}$$

2.4 I sensori

Per effettuare le misure dei sensori si è utilizzata una massa analogica, quindi l'A/D prende direttamente il riferimento nel sensore e le misure non sono influenzate dalla massa dall'alimentazione di potenza. Ciò può avvenire a causa dell'induttanza parassita delle piste. Infatti in caso di veloci variazioni di corrente si possono avere

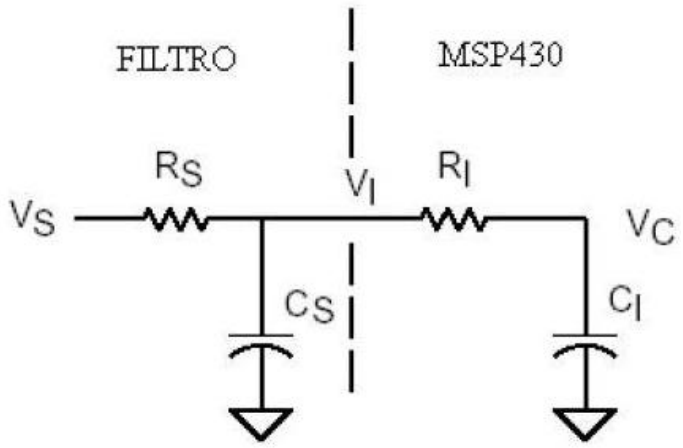


Figura 2.3. Passa basso A/D

$$t_{\text{sample}} >= 5\tau + R_I \cdot C_I \cdot ln(2^{13}) + 800\,\text{ns} = 5 \cdot R_S \cdot C_S + R_I \cdot C_I \cdot 9.011 + 800\,\text{ns} = 101.5\,\text{us}$$

variazione dei livelli di tensione (Ground Bounce), con conseguente errore nelle misure. I sensori presenti sono i seguenti

2.4.1 Sensore di corrente

Le misure di corrente servono per controllare i consumi delle varie parti del sistema. Permettono di:

- valutare lo stato di funzionamento del satellite;
- rilevare i consumi durante l'orbita.

La misura viene fatta attraverso una resistenza di *sense*. Si può procedere in due modalità: misura high-side o low-side. Le due modalità differiscono in base alla posizione della resistenza di sense. Nel caso di misura high-side la resistenza viene posta tra alimentazione e carico, nel caso di misura low-side, invece, è posta tra il carico e il riferimento di ground. Quest'ultimo presenta il vantaggio di avere un modo comune sulla resistenza circa nullo, ma ha lo svantaggio di interrompere il ritorno comune di massa. Nel primo caso si ha il vantaggio di non avere sporcato il riferimento del sistema, ma si ha una tensione di modo comune pari alla tensione di alimentazione.

Poiché il satellite può essere soggetto a problemi di compatibilità elettromagnetica, si è preferito utilizzare un sensore di tipo high-side. Nel sistema è necessario misurare non solo correnti unidirezionali, ma nel caso delle batterie anche correnti

bidirezionali (carica e scarica). In questo caso, però, il modo comune in ingresso all'operazionale diviene elevato (la tensione massima cui può arrivare un pacco batterie è di circa 8,8V) e ben superiore alla massima tensione di modo comune sopportabile dagli operazionali scelti. Per questo, visto che il percorso delle batterie non è critico dal punto di vista della compatibilità elettromagnetica, si è deciso di misurare la corrente in maniera low-side. Lo schema circuitale del sensore di corrente monodirezionale e del sensore bidirezionale sono riportati in figura.

La caduta sulla resistenza di sense, per limitare le perdite del circuito, deve essere molto piccola, ma questo dà un problema: la tensione misurata deve essere sufficientemente superiore alla tensione di offset dell'operazionale scelto, in modo che in ingresso al convertitore A/D si possa avere il segnale maggiore possibile rispetto alla tensione di offset. Ora la tensione di caduta deve essere intorno ai 40mV in modo da dare una perdita di potenza inferiore all'1

Questo ha portato alla ricerca di un amplificatore operazionale con una bassa tensione di offset. L'operazionale che è stato scelto è il MAX4091 prodotto dalla Maxim, che ha le seguenti caratteristiche:

- tensione di alimentazione compresa tra +2,7V e +6,0V;
- corrente di alimentazione di 165uA;
- tensione di offset inferiore a 30uV;
- correnti di offset inferiori a 7nA su tutto il range di temperatura;

In questi schemi vi è la presenza anche di alcuni condensatori che fungono da filtri nei confronti dei disturbi presenti a bordo, dati dai convertitori switching e dai sistemi di rice-trasmissione a radio frequenza. Tali filtri hanno una frequenza di taglio pari a

che permette di filtrare sia i disturbi a radiofrequenza (oltre i 400MHz), sia i disturbi indotti dai convertitori switching (intorno ai 50 kHz).

2.4.2 Sensore di tensione

Le misure di tensione sono fatte, insieme alle misure di corrente, per tenere conto della erogazione di potenza nelle varie parti del sistema. La misura di tensione viene realizzata tramite un partitore resistivo (si veda lo schema circuitale riportato nella figura).

Questo partitore riduce la dinamica delle tensioni presenti sul satellite (da 2V a 12V) per scalarla alla dinamica di ingresso del canale A/D del microcontrollore compresa tra 0V e 1,5V, ricavata tramite un riferimento interno al microcontrollore stesso.

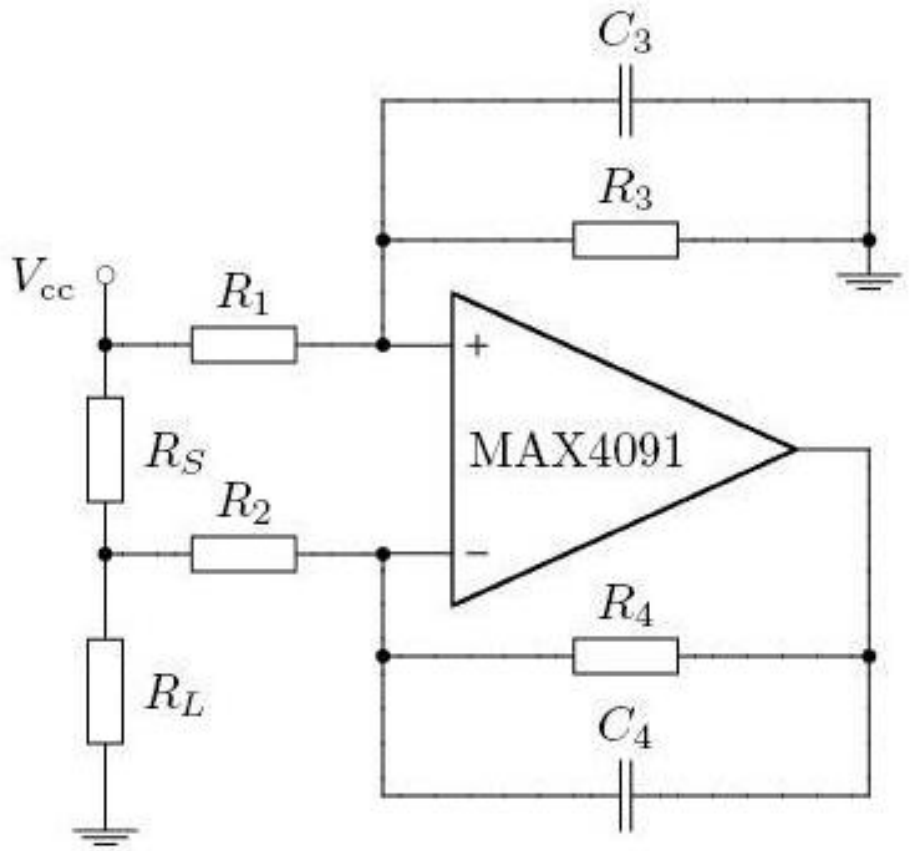


Figura 2.4. Schema circuitale per correnti di alimentazione

La sensibilità del partitore si ricava immediatamente, infatti si vede come più il valore della resistenza R2 è elevato maggiore è la sensibilità del circuito. Purtroppo tale rapporto è fissato dall'adattamento della tensione al range di ingresso dell'ADC (1,5V) e quindi non è possibile cambiarlo. Un aspetto da tenere in considerazione è il fatto che maggiore è il valore delle resistenze, minore è la potenza dissipata dal circuito di condizionamento (ed in questa applicazione è fondamentale il risparmio di energia, essendo infatti un sistema a batterie).

2.4.3 Misure di Temperatura

Le misure di temperatura sono utilizzate per conoscere le condizioni ambientali cui è sottoposto il satellite durante i periodi di tempo in cui è illuminato dal sole rispetto a quelli in cui è coperto e per conoscere il surriscaldamento relativo alle dissipazioni di energia per effetto Joule e per conoscere la temperatura cui si trovano le batterie. In base ai dati prelevati dai siti internet relativi ad altri progetti simili, si è potuto conoscere che il range di temperature per cui il pico-satellite deve essere progettato è pari a -40 °C : +150 °C, all'esterno della struttura, e 0 °C : 100 °C all'interno. Mentre

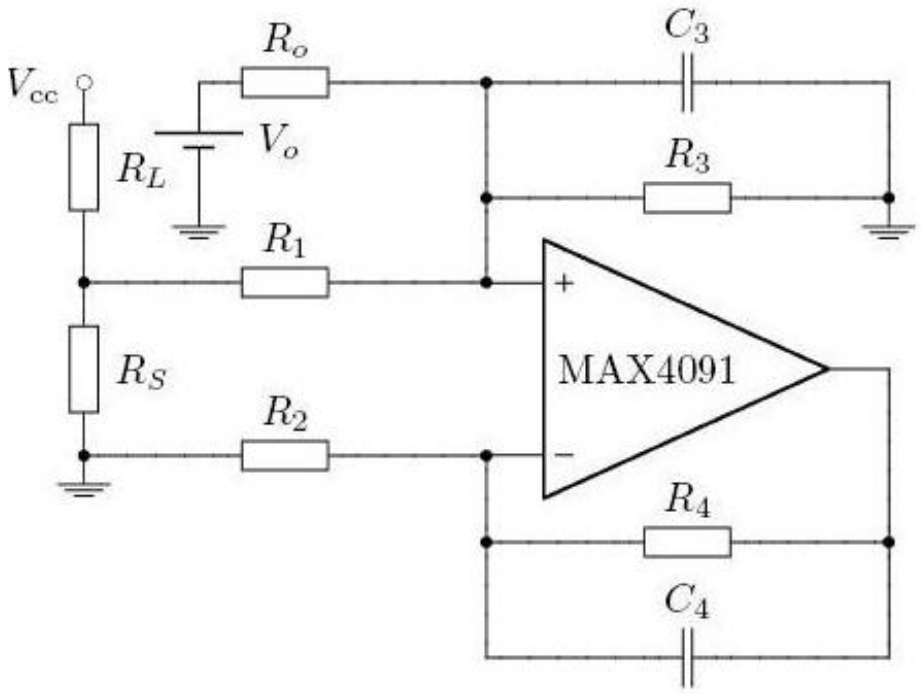


Figura 2.5. Schema circuitale per correnti entranti e uscenti nelle batterie

$$f_c = \frac{1}{R_4 \cdot C_4} = \frac{1}{150\,\mathrm{k\Omega} \times 100\,\mathrm{nF}} = 66\,\mathrm{Hz}$$

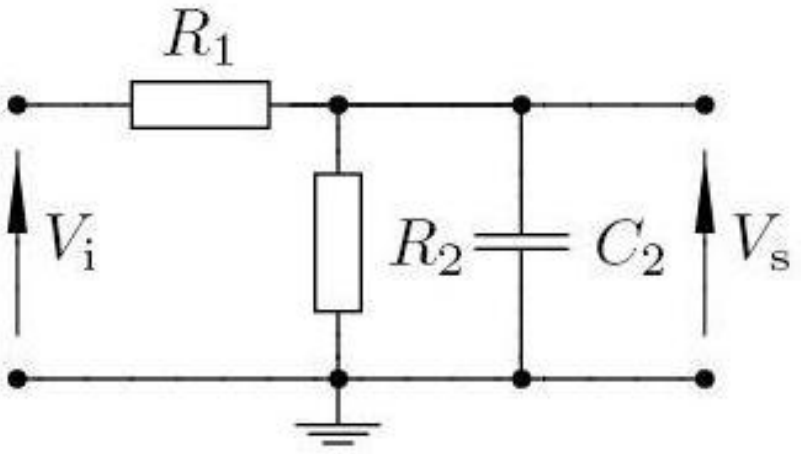


Figura 2.6. Sensore di tensione

$$\frac{\Delta V_o}{\Delta V_i} = \frac{R_2}{R_1 + R_2}$$

per l'ambiente interno è facile trovare sensori che siano in grado di sopportare tali escursioni, ben più difficile è trovare sensori adatti all'esterno. Infatti i sensori a filo metallico (lineari e con ampie regioni di funzionamento) hanno due svantaggi:

1. una bassa resistività del materiale utilizzato che causa una bassa resistenza del sensore e, quindi, un'elevata dissipazione di potenza;
2. un ingombro notevole, dato dall'avvolgimento del filo metallico, effettuato molte volte appunto per innalzare il valore resistivo del sensore stesso.

I sensori esterni, inoltre, hanno un grosso vincolo di natura fisica, infatti dovendo essere posizionati tra il pannello fotovoltaico e il suo supporto, devono avere uno spessore massimo di 0,5mm. A causa di questi inconvenienti si è stati costretti ad utilizzare dei termistori di tipo NTC. Questi hanno pregi ed inconvenienti scambiati rispetto ai sensori a filo, infatti sono poco lineari, poco precisi e con range limitato,ma offrono un'elevata resistenza, in concomitanza ad un occupazione modesta di spazio. Gli unici termistori capaci di soddisfare le specifiche sono stati gli NTC TR0603J252K della RTI Electronics, le cui caratteristiche sono:

- spessore pari a 0,4 mm;
- esistenza variabile da 100 a 42,5 kΩ
- range di funzionamento compreso tra -55 °C e +160 °C
- tolleranza costruttiva del sensore del 10 a 25 °C

La scelta di un valore di resistenza così basso è da imputare alla curva della caratteristica del sensore: infatti é l' NTC con valore resistivo maggiore che presenta una curva caratteristica con minore escursione. E' noto, come la caratteristica degli NTC sia altamente non lineare (si confronti a riguardo la figura) e si è scelto di effettuare una prima linearizzazione via hardware.

A tale scopo si è utilizzato un circuito di condizionamento a ponte di Wheatstone, ottenendo il circuito riportato nella figura.

L'introduzione del ponte ha inoltre permesso di ottenere un consumo di potenza ridotto, in quanto la resistenza inserita in serie all'NTC riduce notevolmente la corrente che lo attraversa. Per evitare in uscita delle tensioni negative, non gestibili né dall'operazionale, né dal convertitore AD, è importante progettare il ramo di riferimento in modo da fornire una tensione pari alla minima generata dal ramo di misura.

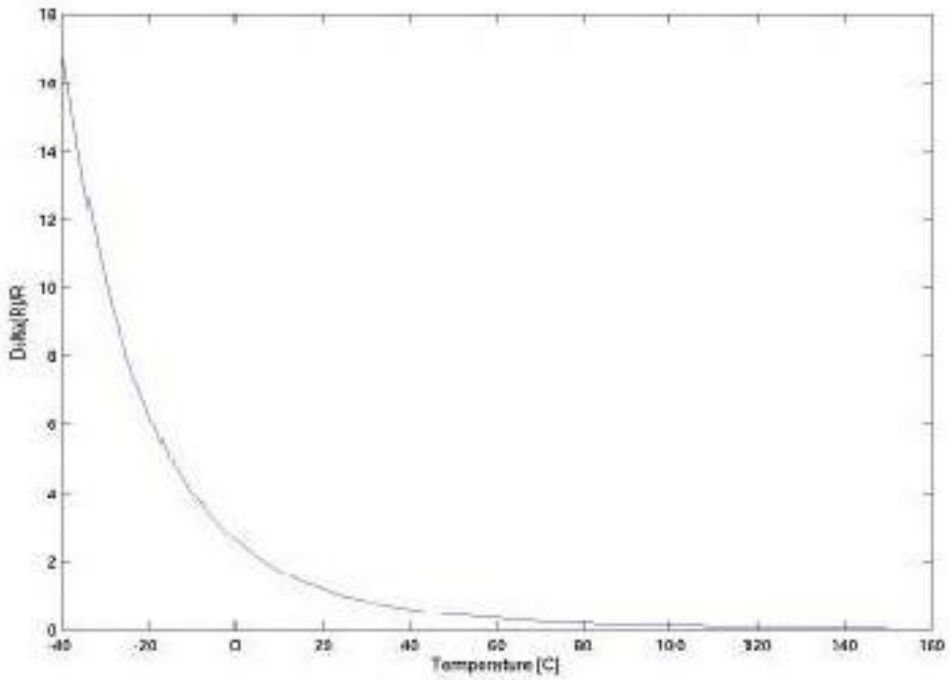


Figura 2.7. Sensore NTC

2.4.4 Misure di Campo Magnetico

PiCPoT è immerso nel campo magnetico terrestre ed il suo orientamento, rispetto alla superficie, segue l'andamento delle linee di flusso grazie alla presenza di quattro magneti permanenti disposti lungo la sua struttura. La misura del campo cui è sottoposto è utile per conoscere l'orientamento del satellite rispetto alle linee di campo. Per questo all'interno del satellite si prevede di inserire quattro sensori magnetoresistivi per permettere di rilevare il modulo dell'intensità del campo.

I sensori sono disposti su due schede differenti: la scheda Processore A e la scheda Processore B. Questi sono orientati in modo da coprire le tre direzioni (X,Y,Z) spaziali, implementando una ridondanza sull'asse X (asse magnetico di rotazione del satellite) in modo da permettere di conoscere l'intensità del campo nella direzione attorno cui il cubo ruota. I componenti utilizzati sono i dispositivi HMC1022 della Honeywell sensori per la misura del flusso in due direzioni ortogonali . I sensori magneto-resistivi sono composti da un ponte resistivo (uno per ogni asse di misura) integrato comprendenti resistori in permalloy (lega di ferro e nickel) che presentano un naturale asse magnetico (easy axis) e, in presenza di un campo esterno al dispositivo, variano la loro resistività.

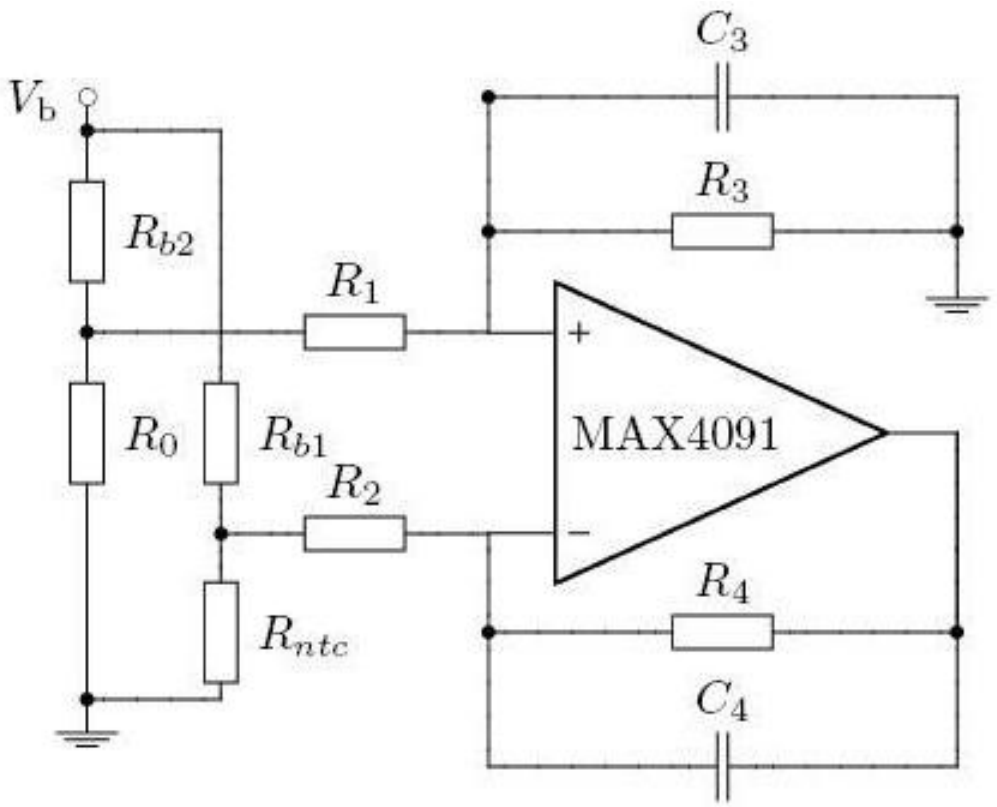


Figura 2.8. Schema circuitale per la misura della temperatura

Nome	*Valore*	*Tolleranz a*
NTC	2,5@25°C	10%
A_0R_0	47kΩ	1%
R_1	100Ω	1%
A_1R_1	47kΩ	1%

Figura 2.9. Valori resistivi per il condizionamento dei sensori di temperatura

Il circuito di condizionamento deve tenere conto del fatto che la differenza di potenziale ai capi del ponte varia in modo lineare con il campo magnetico applicato (1mV/V/G tipico), ma ha una dinamica di

$$\pm 20mV + Voff \tag{2.1}$$

in cui Voff è la tensione di offset data dalle tolleranze costruttive del sensore ed assume un valore massimo di

$$\pm 11mV \tag{2.2}$$

Questi hanno la caratteristica di fornire una tensione di uscita bipolare in cui il valore dell'offset è grande ed incognito. Per quest'ultimo problema è stato deciso di non utilizzare i pin appositi di compensazione dell'offset presenti sul sensore, in quanto necessitano di una corrente molto elevata (4,6mA/G) per eliminare l'offset, ma di misurare la tensione a campo nullo e poi rimuoverla via software dopo la conversione in numerico, da parte del micro-controllore.

Il circuito di condizionamento deve permettere di rilevare la tensione differenziale ai capi del ponte e riportarla in tensione positiva. Purtroppo non è, però possibile sbilanciare il ponte, come fatto per le misure di temperatura, in quanto il ponte magneto - resistivo è integrato; si é ricorso allora all'applicazione di una tensione di riferimento sull'operazionale come si vede nello schema circuitale riportato nella figura. In questo modo l'uscita dell'operazionale diventa

$$V_{out} = A \cdot V_d + \frac{1}{2} V_r$$

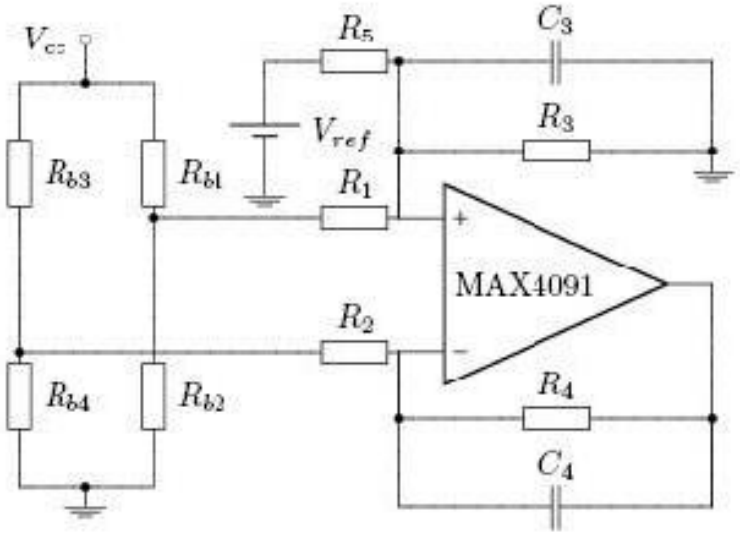


Figura 2.10. Schema del circuito di funzionamento del sensore per la misura del campo magnetico

Capitolo 3

I Protocolli

3.1 Il protocollo APRS

3.1.1 Introduzione

APRS è l'acronimo di Automatic Position Reporting System, che è stato introdotto da Bob Bruninga al TAPR/ARRL Digital Communication Conference del 1992. Fondamentalmente, l'APRS è un protocollo di comunicazione packet (a pacchetti) per la diffusione di dati attuali in tempo reale. Molte volte viene rappresentato come combinazione del packet radio con la rete satellitare Global Position System (GPS), in modo da permettere agli utenti la visualizzazione automatica delle posizioni delle stazioni radio e di altri oggetti sulla mappa di un PC. Sono supportate numerose altre caratteristiche che non sono direttamente correlate con il rapporto della posizione, come il rapporto di condizioni metereologiche, ricerca di segali e scambio di messaggi.

APRS si differenzia dal comune packet radio per molti aspetti:

- fornisce rappresentazioni di mappe ed altri dati, per la localizzazione di veicoli o persone e per i rapporti meteo in tempo reale;
- utilizza tutte le comunicazioni attraverso un protocollo uno-a-molti, in modo da aggiornare immediatamente tutti gli operatori collegati
- usa il digipeating generico, con gli alias dei codici di identificazione ben conosciuti, in questo modo non è necessario la conferma a priori della connessione alla rete.
- supporta il digipeating intelligente, con la sostituzione del codice di identificazione per ridurre l'affollamento del canale radio.
- utilizzando i frame **UI** del protocollo AX-25 (X25 modificato per il traffico digitale radio), supporta lo scambio di messaggi tra due operatori e la distribuzione di bollettini e di annunci, diffondendo informazioni in modo molto veloce.

- supporta il collegamento con le radio Kenwood TH-D7 e TM-D700 che dispongono, al loro interno, di TNC e firmware APRS.

Il packet radio convenzionale è realmente molto utile per trasportare traffico da una stazione all'altra (point-to-point) ed è stato sempre tradizionalmente difficile da applicare ad eventi in tempo reale dove le informazioni hanno una vita estremamente breve. L'APRS permette di utilizzare il packet radio per comunicazioni tattiche e visualizzazioni in real-time per emergenze e per applicazioni di pubblico servizio. L'APRS permette una connettività universale di tutte le stazioni, evitando la complessità, tempi di ritardo e le limitazioni di una rete a connessione. Permette lo scambio di dati da qualsivoglia numero di stazioni. Qualsiasi stazione che ha informazioni da comunicare le spedisce semplicemente, e tutte le stazioni le ricevono e le registrano.

3.1.2 Struttura del protocollo

L'implementazione dell'APRS può essere schematizzata a livelli: Il primo livello risulta essere il protocollo AX25, che verrà ampiamente spiegato successivamente. Successivamente è necessario implementare l'HDLC . In questo caso l'HDLC implementa tre funzioni da eseguire in sequenza. Vi è anche una quarta funzione: lo scrambling. Pur non facendo parte dell'HDLC, ma a una funzione dei modem essa deve essere eseguita.
L'ordine è:

- La codifica differenziale NRZI (Non Return to Zero Inverted)
- Scrambling
- Bit Stuffing
- Delimitazione del frame

La codifica differenziale NRZI, consiste nell'effettuare la differenza tra bit successivi. In pratica per trasmettere un 1 si lascia l'uscita invariata, mentre per trasmettere uno 0 si inverte l'uscita. La seguente figura può chiarirne meglio il funzionamento.

Successivamente è necessario fare lo scrambling. Questa operazione consiste nel far passare i bit attraverso il polinomio 1 + X12 + X 17, denominato Polinomio di Miller dal nome del suo inventore. Le operazioni da fare in trasmissione e in ricezione si possono schematizzare così:

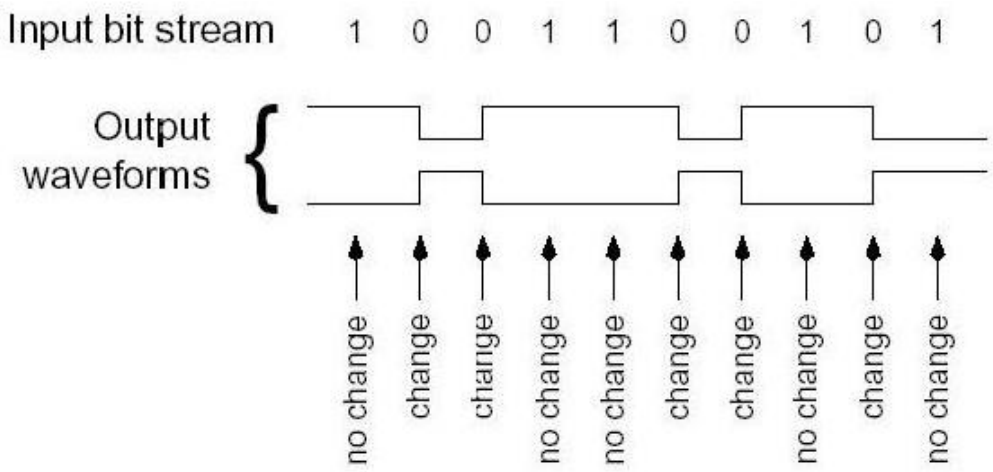


Figura 3.1. codifica NRZI

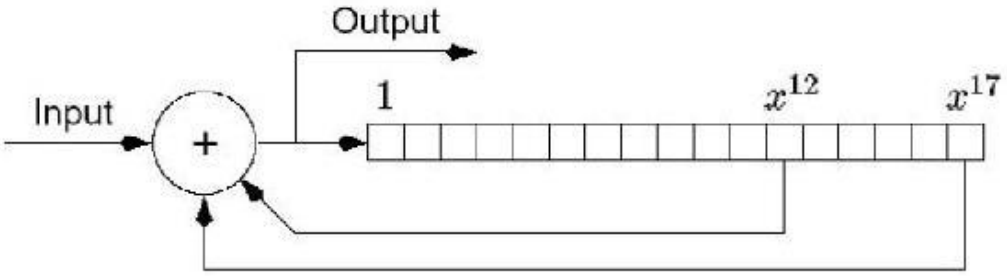


Figura 3.2. Polinomio in scrambling

3.2 Il protocollo AX.25

3.2.1 Introduzione

Questo protocollo nasce dalla necessità e dal desiderio espresso dalla comunità radio-amatoriale di definire un protocollo che possa garantire la compatibilità link-layer

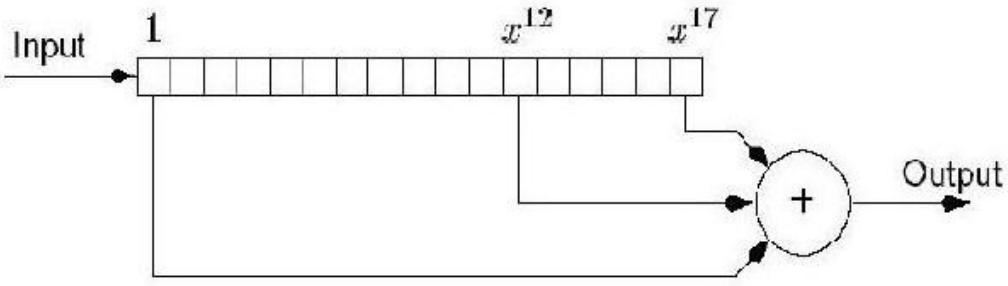


Figura 3.3. Polinomio in descrambling

fra stazioni. Questo protocollo è conforme alle raccomandazioni ISO 3309, 4335 (includendo DAD12) e 6256 high-level data link control (HDLC) e usa della terminologia rintracciabile in questi documenti. Questo protocollo è anche conforme alla ANSI X3.66, che descrive ADCCP, in modo bilanciato. Questo protocollo è scritto per poter lavorare egualmente bene in ambienti radioamatoriali sia half che full duplex. Questo protocollo permette che sia stabilito più di un collegamento al livello due (link layer) per apparecchiatura, sempre che questa lo permetta. La maggior parte dei link-layer protocols presume che un dispositivo primario (o master), generalmente chiamato Data Circuit Terminating Equipment (o DCE), sia connesso a uno o più dispositivi secondari(slave),generalmente chiamati Data Terminating Equipments(DTE).

Questo tipo di operazione non bilanciata non è auspicabile in un mezzo condiviso qual'è il tipico canale radioamatoriale. AX.25 presuppone invece che entrambe le terminazioni del collegamento siano della stessa classe eliminando perciò le due differenti classi di dispositivi.

3.2.2 Struttura del frame

Le trasmissioni Packet Radio a livello data link sono costituite da blocchi di dati più piccoli,chiamati *frames* (pacchetti). Ci sono tre diversi tipi di frame in AX.25:

1. Information frame (I frame);
2. Supervisory frame (S frame);
3. Unnumbered frame.

Ciascun frame è composto a sua volta da diversi blocchi più piccoli, chiamati fields(campi). La figura illustra i tre tipi di frame.

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

Figura 3.4. U and S frame construction

Flag Field
Questo campo è lungo un byte. Essendo che il flag delimita i frame,ne esso deve essere presente all'inizio e alla fine di ciascun frame. Due frame possono condividere un flag,che potrebbe quindi denotare la fine di quest'ultimo e l'inizio del successivo. Un flag consiste di uno zero seguito da una serie di sei uno seguiti da un' altro zero

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

Figura 3.5. Information Frame Construction

(01111110 o 7E in esadecimale).Questa sequenza è permessa solo all'inizio ed alla fine di ogni pacchetto valido.

Address Field

In questo campo abbiamo l' identificazione dell'indirizzo sorgente e quello di destinazione del frame per una lunghezza di 7 byte ciascuno. Nel nostro caso l'indirizzo sorgente è composto da 6 caratteri (PICPOT) ciascuno dei quali è shiftato a sinistra di un bit,seguiti da uno 0x61; mentre l'identificativo della stazione di destinazione è composto sempre da 6 caratteri (ALL+ 3 blank),ciascuno dei quali è shiftato a sinistra di un bit, seguiti da uno 0xE0.

Address Field of Frame	
Destination Address Subfield	Source Address Subfield
A1 A2 A3 A4 A5 A6 A7	A8 A9 A10 A11 A12 A13 A14

Figura 3.6. Address Field

Control Field

Il campo di controllo identifica i tipi di frame che sono stati mandati. I tre formati di campi di controllo usati in AX.25 sono:

- Information frame (I frame);
- Supervisory frame (S frame);
- Unnumbered frame (U frame).

La figura mostra il formato del campo di controllo associato a ciascuno di questi tre tipi di frame.

Control Field Type	Control-Field Bits							
	7	6	5	4	3	2	1	0
I Frame	N(R)			P	N(S)			0
S Frame	N(R)			P/F	S	S	0	1
U Frame	M	M	M	P/F	M	M	1	1

Figura 3.7. Control Field

In cui:

1. N(S) è il numero della sequenza inviata;
2. N(R) è il numero della sequenza ricevuta;
3. I bit Ssono i bit della funzione di sorveglianza;
4. I bit M sono i bit dell' unnumbered frame;
5. P/F Poll/Final bit.

La trama INFORMATIVA costituisce la trama del messaggio, quindi esegue il trasferimento delle informazioni tra le due stazioni. Le trame sono numerate in modo sequenziale progressivo da un contatore a 3 o 7 bit (se il campo si estende a 16) ed i bit b1, b2, b3 codificano il numero sequenziale delle trame che si sta trasmettendo (max. 7 TRAME).

La trama SUPERVISIONE esegue le funzioni di controllo del collegamento, ossia conferma la corretta ricezione delle trame d'informazione ed in caso di trame errate richiede la loro ritrasmissione. Non contiene campo di informazione. La trama di supervisione è numerata come segue:

- **NS**:assente;
- **S**: comando per le funzioni di supervisione (bit b2, b3); determina dei comandi o delle risposte secondo la seguente codifica;

- **RR** (Reiceve Read = pronto a ricevere) 00: il dispositivo è pronto per ricevere una trama numerata I;
- **REJ** (REJect = rifiuto) 01: il dispositivo fa la richiesta di ritrasmissione delle trame I numerate dal contatore NR (significa che le trame fino a NR -1 sono state ricevute correttamente);
- **RNR** (Reiceve Not Ready = ricezione non pronta) 10: il dispositivo segnala la propria indisponibilità temporanea a ricevere le trame I.

La trama SUPERVISIONE viene impiegata per stabilire il collegamento di linea, o meno (attivazione e disattivazione di una stazione). Può contenere il campo di informazione ma non contiene i contatori N(S) e N(R).

Nella zona di controllo della trama U è presente il comando di codifica M, rappresentato dai bit b2, b3 e b5, b6, b7 con la seguente codifica:

- **SARM** (Set Asynchronous Response Mode = predisposizione in modo di risposta asincrono) 1,1,0,0,0: richiesta alla stazione remota di porsi in ricezione in risposta asincrona. La stazione remota conferma inviando la trama
- **UA** (Unumbered Acknowledgment = accettazione non numerata)
- **DISC** (DISConnect = disconnessione) 0,0,1,1,0: la stazione remota conferma la disconnessione inviando la trama - UA;
- **SABM** (Set Asynchronous Balanced Mode = predisposizione in modo asincrono bilanciato) 1,1,1,0,0: la stazione remota è comandata a porsi in modo asincrono (può iniziare una trasmissione senza il permesso della stazione primaria) bilanciato. Essa conferma inviando la trama - UA.

Nel nostro caso APRS usa il formato UNNUMBERADED frame del campo di controllo dello standard AX.25 impostando i bit di controllo a entrambi a 1(0x03 in esadecimale).

PID Field

Protocol Identifier field: campo identificatore di protocollo. È usato solo nei pacchetti d'informazione, identifica che tipo di livello 3 è in uso, ammesso che ve ne sia uno. La sua decodifica ha i valori riportati in figura 3.8:

con "y" che indica tutte le combinazioni usate.

Nel nostro caso il valore è fisso a 0xF0.

Information Field

Il campo di informazioni (I) ha una lunghezza variabile da 1 a 256 byte. Il campo d'Informazione è valido solo per tre tipi di pacchetti: I frames, UI, frames e FRMR

HEX	MSB LSB	Translation
**	yy01yyyy	AX.25 layer 3 implemented.
**	yy10yyyy	AX.25 layer 3 implemented.
0x01	00000001	ISO 8208/CCITT X.25 PLP
0x06	00000110	Compressed TCP/IP packet. Van Jacobson (RFC 1144)
0x07	00000111	Uncompressed TCP/IP packet. Van Jacobson (RFC 1144)
0x08	00001000	Segmentation fragment
0xC3	11000011	TEXNET datagram protocol
0xC4	11000100	Link Quality Protocol
0xCA	11001010	Appletalk
0xCB	11001011	Appletalk ARP
0xCC	11001100	ARPA Internet Protocol
0xCD	11001101	ARPA Address resolution
0xCE	11001110	FlexNet
0xCF	11001111	NET/ROM
0xF0	11110000	No layer 3 protocol implemented.
0xFF	11111111	Escape character. Next octet contains more Level 3 protocol information.
Escape character. Next octet contains more Level 3 protocol information.	00001000	

Figura 3.8. PID Field

frames. Ovvero: Information frames; Unnumbered Information frames; FRaMe Rejected frames. Tutti i dati che vengono inseriti all'interno dell' I field vengono passati in modo trasparente dal mittente al destinatario lungo il collegamento, salvo per l'inserzione di un bit a 0 necessaria per evitare che un FLAG compaia all'interno del campo dati (bit stuffing).

Frame Check Sequence

Frame Check Sequence. Il campo di controllo del pacchetto (FCS o CRC), è un numero composto espresso in 16 bit e viene calcolato sia dal destinatario che dal mittente di ogni pacchetto.

Il mittente lo calcola e lo inserisce nel pacchetto da spedire, il destinatario lo ricalcola (in base ai dati ricevuti) e lo controlla con quello speditogli nel frame. Ha compiti di verifica dell'integrità del pacchetto dopo la trasmissione attraverso il mezzo, viene calcolato secondo l'algoritmo suggerito nella raccomandazione ISO 3309 (HDLC).

Il CRC - 16 ha le seguenti proprietà:

Si ricorda che è necessario trasmettere prima il byte meno significativo del FCS (in quanto contiene i coefficienti del termine più grande). Il FCS viene calcolato su tutti i bits di Address, Control, Protocol, Information and Padding fields, tranne che sui flag di inizio e fine frame e sullo stesso FCS.

Tipo di errore	*Percentuale di rilevarlo*
1 bit error	100%
2 bit error	100%
Odd error	100%
Burst error<16 bit	100%
Burst error su 17 bit	99,9969%
Altri burst error	99,9984%

Figura 3.9. Controllo di flusso del protocollo

3.2.3 Procedure di trasmissione

In questa sezione verranno illustrate le principali regole richieste dal protocollo per trasmettere i frame correttamente.

Bit stuffing

Per assicurare che la sequenza di bit che caratterizzano il Flag non appaia accidentalmente all'interno del frame,la stazione mandante monitora la sequenza di bit per un gruppo di cinque bit ad 1. Ogni volta che sono mandati cinque bit continui ad 1, la stazione che trasmette inserisce un bit a 0.

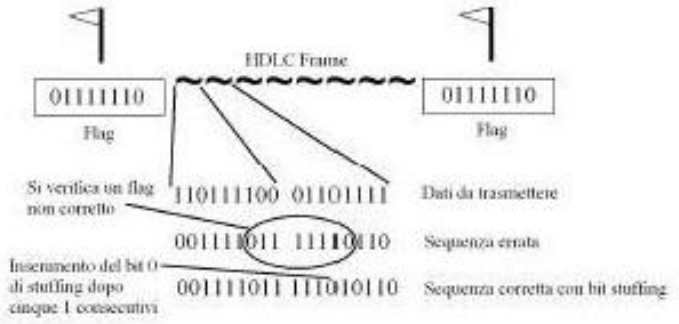


Figura 3.10. Bit stuffing

Durante la ricezione del frame, ogni volta che sono ricevuti cinque bit a 1, il bit 0, che segue i cinque bit ad 1,viene scaricato. Come si vede in figura la trasmissione viene fatta da sinistra verso destra ma di ogni byte che si considera bisogna trasmettere prima LSB. Questo comporta a dover esaminare ogni byte bit a bit e quindi a scomporlo. Essendo la trasmissione nel nostro caso implementata tramite l' SPI dell'MSP430 e non essendo possibile trasmettere singoli bit ma solo byte (o al limite 7 bit) è necessario ricomporre i singoli bit. Ciò è dovuto al fatto che, pur

utilizzando un interfaccia seriale quale SPI, per inviare i dati bisogna caricare un buffer.

Bit Order of Transmission

Il frame viene trasmesso da sinistra vero destra. Si ricorda che con l'eccezione dell' FCS (Frame Check Sequence), tutti gli altri campi di ogni pacchetto AX.25 vengono trasmessi partendo dal Least Significant Bit (LSB), bit meno significativo. In accordo con la pratica HDLC il campo FCS vede come primo bit trasmesso il Most Significant Bit, bit piu' significativo.

Invalid Frame

Un frame è considerato non valido dallo strato *link* se:

- è lungo meno di 136 bit(includendo i flag di apertura e chiusura);
- non è delimitato dai flag di apertura e chiusura;
- non ha un numero di byte allineati(byte incompleti, il numero di bits non è divisibile per 8 con resto 0).

Frame Abort

Se il frame deve essere abortito prematuramente, devono essere trasmessi almeno quindici bit consecutivi ad 1 senza bit stuffing.

Address Field Encoding

Il campo d'indirizzo di tutti i pacchetti viene codificato con i nominativi dei radioamatori, sia mittente che destinatario. Se viene usato un ripetitore a livello 2, il suo nominativo sarà riportato sempre nel campo d'indirizzamento. AX.25 segue la forma prevista dal protocollo HDLC per elongare il campo d'indirizzamento, per farvi entrare tutti i dati necessari. Per poter far ciò, HDLC prevede che il bit meno significativo di ogni byte se settato a 1 indichi che il byte successivo conterrà ancora un dato utile d'indirizzamento. Viceversa quando il bit sarà a zero, darà la fine del campo. Questo bit viene chiamato extender bit (bit d' estensione). Per creare spazio all' e.b., i nominativi vengono shiftati di un bit a sinistra.

3.2.4 Implementazione software

Calcolo del CRC

Nel frame HDLC, tutti i byte di dati sono sottomessi al calcolo dell' FSC, eccetto i byte di inizio e fine frame(flag). L'FCS calcolato è inserito appena prima del flag di chiusura del frame,mettendo prima il byte più basso e dopo quello più alto. Il ricevitore include i due byte FCS nel frame HDLC ricevuto per il calcolo. Il risultato finale perché sia positivo deve rendere il valore di F0B8 (hex) .

Lo standard HDLC ISO-3309 definisce quindi il CRC16 (utilizzato sia nel protocollo APRS, sia all'interno di altre comunicazioni quali Xmodem) e fornisce come polinomio caratteristico il seguente:

$$x^{16} + x^{12} + x^5 + 1$$

```
static unsigned short fcstab[256] = {

0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,
0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7eff,
0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
   };
```

Figura 3.11. Loop Table

dal quale deriva la seguente loop table:
A questo punto basta seguire questo semplice algoritmo per calcolare il CRC:

1. Inizializzare il CRC a FFFF;
2. EX-OR del nuovo byte in ingresso con il byte meno significativo del CRC per ottenere l'indice della lookup table;
3. Shiftare il CRC a destra di 8 bit;
4. EX-OR del CRC con il byte ottenuto in precedenza dalla lookup table;
5. Ripetere i passi dall'1 al 3 per tutti i byte;

Tale algoritmo è rappresentato dalla seguente implementazione:
La funzione che esegue l' algoritmo è:

$$CRC = Hi(CRCvalue) XOR Table[NewByte XOR Lo(CRCvalue)]$$

```
*************************************************************************
unsigned short APRSfcs(unsigned short fcs, char *byte, int len)
{
while (len--)
      fcs = (fcs >> 8) ^ fcstab[(fcs ^ *byte++) & 0xff];
return (fcs);
}

*************************************************************************
```

3.3 Il Protocollo Xmodem

Xmodem è un protocollo per il trasferimento file sviluppato da Ward Christensen nel 1977. Come la maggior parte dei protocolli di trasferimento file, anche l'Xmodem scompone i dati originali in una serie di pacchetti che sono trasmessi al ricevente, con informazioni supplementari, permettendo alla stazione ricevente di determinare se quel pacchetto sia stato ricevuto correttamente.

3.3.1 Struttura del protocollo

Xmodem è un protocollo relativamente semplice:

- un emettitore ed un ricevitore (collegamento punto-punto half duplex);
- scomposizione dei dati da trasmettere in blocchi di taglia fissa:128 byte;
- i blocchi contenenti i dati trasferiti con un numero di sequenza e una somma di controllo permettono di convalidare la trasmissione del blocco;
- può essere trasferito un solo file attraverso una connessione ed in caso di caduta della connessione, il trasferimento del file deve essere ripreso dall'inizio.

Il formato d'un blocco Xmodem è il seguente:

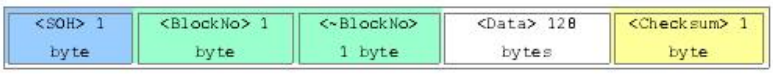


Figura 3.12. Blocco XModem

- SOH individua l'inizio del Frame

- BlockNo è il numero, (con dimensione un byte) del Frame . I numeri di Frame hanno gamma [1, 255]. Dopo 255, il ciclo del numero di Frame si ripete.

- BlockNo è il complementare del numero di frame.

- Data è il campo dati lungo 128 caratteri (byte). Se i dati sono più corti di 128 caratteri, il blocco di dati è completato con caratteri di riempimento (CtrlZ per default).

- Checksum è il controllo che consiste in una somma aritmetica senza riporto (modulo-256) di tutti i byte di dati.

- Handshake: il ricevitore comincia sempre la comunicazione nella fase di start up trasmettendo un carattere di handshake(in questo caso un carattere di NAK). Aspetta 10 secondi per la risposta. Se non riceve un blocco in 10 secondi ritrasmette un altro NAK. Il ricevitore tenta un massimo di 10 volte a stabilire l'handshake dopo di che interrompe la trasmissione.

- Response: il trasmettitore trasmette i dati chiesti. I dati sono divisi in blocchi lunghi 128 caratteri. Il Timeout affinché ogni carattere sia ricevuto è 1 secondo. Il trasmettitore attende dopo il primo blocco no ad 1 minuto per ricevere l'Acknowledgement del ricevitore. Ciò dà il tempo al ricevitore per la preparazione della memoria a dischi.

- Acknowledgement: il ricevitore calcola checksum e lo confronta col checksum ricevuto. Se i checksum sono gli stessi, trasmette l'acknowledgement positivo ACK. Se differiscono il ricevitore trasmette un NAK ed il trasmettitore ritrasmette il blocco corrente. Ciò continua fino alla trasmissione di tutti i dati.

Alla fine della trasmissione, l'emettitore termina la trasmissione inviando un EOT(0x04) che dovrà essere liberato da un ultimo ACK del ricevitore. E' possibile interrompere una trasmissione attraverso il codice CAN:0x18.

Un trasferimento tipico del protocollo Xmodem è simile a quanto illustrato:

Trasmettitore		*Ricevitore*	
	←	<NAK>	handshake
<SOH><1><254><Data128bytes><CHK>	→		response
	←	<ACK>	acknowledgment
<SOH><2><253><Data128bytes><CHK>	→		
	←	<ACK>	
<EOT>	→		end of transmission
	←	<ACK>	

Il protocollo non contiene alcuna informazione supplementare su ciò che è trasmessa, come il nome o le dimensioni. Quindi, la dimensione del le ricevuto è sempre un multiplo delle dimensioni del blocco. Di seguito vengono riportati i flow chart lato trasmettitore e lato ricevitore.

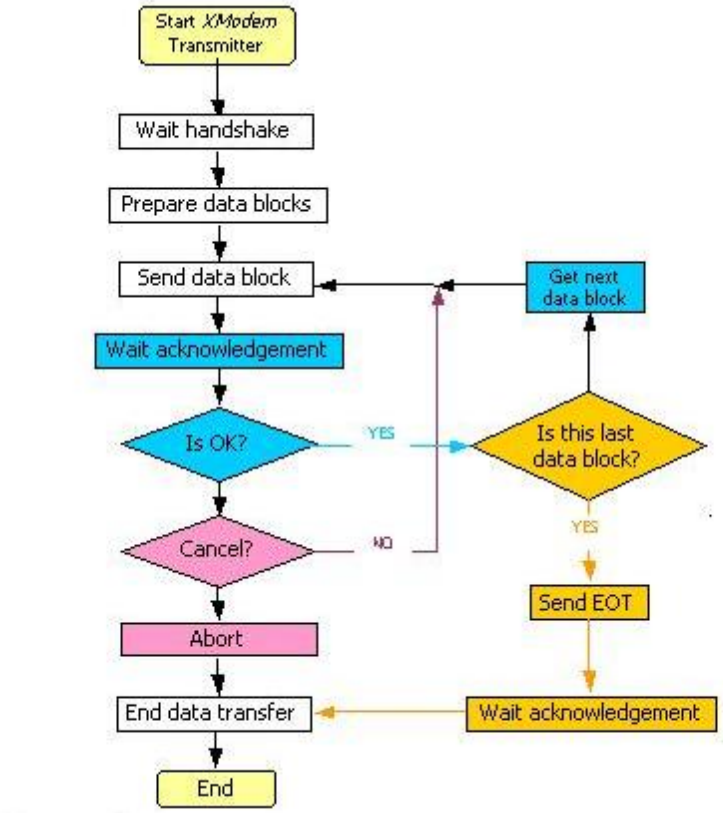


Figura 3.13. Sequenza di trasmissione

3.4 Il protocollo di comunicazione con Power Switch

La PowerSwitch fa uso della periferica SPI per comunicare con la scheda procB, che dovendo comunicare con altre schede, assume il ruolo di Master della comunicazione. Per questo motivo occorre allinearsi alle specifiche dettate dal 'Master' che prevedono la trasmissione di ogni singolo bit sul fronte di salita del clock. L'interfaccia SPI è gestita, all'interno dell'ambiente di sviluppo, tramite apposite funzioni. La configurazione della periferica viene fatta attraverso la funzione setupspi riconosciuta dal compilatore integrato nell'ambiente di sviluppo. Mentre la ricezione-trasmissione di un dato è ottenuta tramite la funzione spiread.

3.4.1 Specifiche di sistema

La scheda Power Switch deve:

- rendere disponibile al processore la data e l'ora corrente;

- riconoscere un comando di aggiornamento dell'orologio da parte del processore;

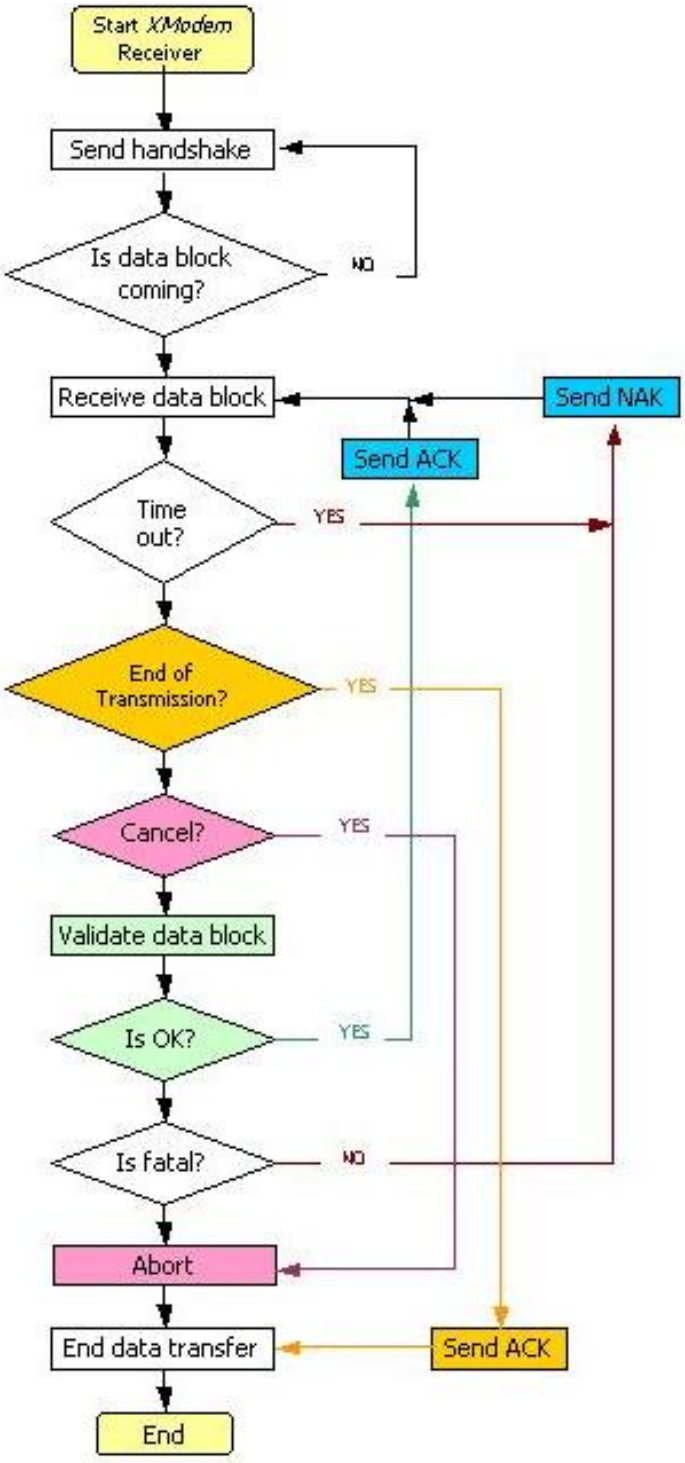


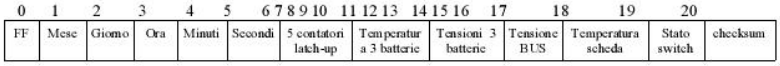
Figura 3.14. Sequenza di ricezione

- comunicare il numero di eventi di latch up conteggiati;
- temperature e tensioni delle batterie;
- temperatura interna della scheda;
- tensione del proprio BUS di potenza;
- eventuali guasti degl'interruttori di selezione delle batterie.

3.4.2 Protocollo di comunicazione

Abbiamo quindi scritto un semplice protocollo di comunicazione. Essendo i dati da trasmettere molto pochi, abbiamo contenuto le dimensioni del pacchetto adottando la seguente convenzione:

- Intestazione - 1 Byte
- Dati - 19 Byte
- Checksum - 1 Byte



Rispettando questa convenzione, il pacchetto trasmesso dalla PowerSwitch alla scheda processore è stato così definito: Dove ogni byte viene trasmesso e ricevuto a partire dall' MSB utilizzando l'SPI hardware del processore. Lo slot di 5 byte disponibile per i contatori di latch up è suddiviso in questo modo:

Indirizzo	Conteggio Latch-up
6	Pic
7	Payload
8	AlimTxRx
9	ProcB
10	sempre 0

Figura 3.15. Contatori Latch-up

Mentre gli slot relativi alle temperature si riferiscono, nell'ordine, alle batterie B1, B2 e B3 iniziando dall'indirizzo più basso con la temperatura della batteria (B1). Lo slot relativo alle tensioni è stato suddiviso in maniera analoga. Il byte relativo

Switch	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0
B1 OK	1	1	-	-	-	-	-	-
B1 KO	0	0	-	-	-	-	-	-
B2 OK				1	1			
B2 KO	-	-	-	0	0	-	-	-
B3 OK	-	-	-	-	-	-	1	1
B3 KO	-	-	-	-	-	-	0	0

Figura 3.16. Byte stato interrutori

allo stato degli interruttori, e quindi relativo ai guasti degli stessi, è costruito in questo modo:

Mentre il checksum è ottenuto sommando tutti i byte del pacchetto dall'indirizzo 0 (intestazione) all'indirizzo 19. Nel caso in cui il checksum ricevuto non è uguale a quello calcolato dalla Power-Switch, all'interno della routine, l'ora ricevuta viene ignorata.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
FF	Mese	Giorno	Ora	Minuti	Secondi	checksum		x	x	x	x	x	x	x	x	x	x	x	x	x

Figura 3.17. Pacchetto da PowerSwitch a ProcB

3.5 Il protocollo di comunicazione con Payload

La scheda Payload comunica con la scheda procB attraverso un'interfaccia SPI. I comandi inviati da ProcB sono formati da due byte con il MSB primo, con il clock attivo sul livello alto. La codifica dei comandi inviati è in parte simile a quella del telecomando TC:

- 3+(a-1)+(t-1)*5 (trigger(a,t,0)) scatto immediato foto a da telecamera t;
- 1818+(a-1) invio foto *a* completa;
- 1823+(b-1) resend(b), ritrasmissione blocco b dell'ultima foto trasmessa;
- 1895+(a-1)*9+(b-1) (sendBlock(a,b)): invio blocco b(1..9) della foto a (1..5);

La scheda Payload può quindi solo ricevere i comandi elencati sopra, in quanto gli altri non sono applicabili. Il comando send (o resend) viene prima convertito dalla scheda procB in un telecomando eseguibile da Payload e poi inviato alla stessa Payload. Le risposte dalla scheda Payload verso procB sono codificate come segue:

- 1 (ackPayload): conferma ricezione ed esecuzione comando; viene inviato immediatamente dopo la corretta ricezione del comando, se questo è eseguibile;

- 2 (nackPayload1): impossibilitato a eseguire comando perché impegnato con immediatamente d opo la corretta ricezione del comando, sequesto non è eseguibile;

- 3 (nackPayload2): impossibilitato a eseguire comando perché impegnato con immediatamente dopo la corretta ricezione del comando, sequesto non è eseguibile;

- 4 (endTrigger): viene inviato al termine della compressione della foto, per il comando trigger; questo causa la disattivazione dell'alimentazione;

- 5 (endPayload): viene inviato al termine del trasferimento immagine, per il comando resend; questo causa la disattivazione dell'alimentazione;

Oltre alle specifiche sopra descritte, per l'invio della foto, si utilizza il protocollo Xmodem descritto in precedenza. L'unica modifica rispetto al protocollo standard riguarda l'inserimento di un numero limitato di letture per quando riguarda gli ACK, in quanto nel protocollo standard si attende a tempo indefinito. Ciò per evitare che eventuali problemi nella scheda Payload, impedissero anche il regolare funzionamento del ProcB.

Capitolo 4

Controllo Motore

4.1 Specifiche

L'assetto del satellite viene controllato, attorno all'asse di spin, da una ruota d'inerzia il cui asse è parallelo all'asse X. La direzione positiva di rotazione della ruota è quella dell'asse Y positivo verso l'asse Z positivo, che causa una rotazione inversa del satellite. In base al telecomando ricevuto, la ruota d'inerzia del motore ruoterà intorno all'asse di spin di x giri(con x = 0, 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048, 4096, 8192, 16384, 32768, 65536) in verso antiorario o orario, per un totale di 31 codici di telecomando. Per svolgere tale lavoro si è usato un motore brushless in quanto dovendo lavorare nel vuoto, in cui c'è il problema del degasamento, questi tipi di motore non presentano elementi rotativi con spazzole. Inoltre visto che non c'erano richieste particolari riguardanti velocità o coppia, il controllo motore è stato fatto ad anello aperto.

4.2 I motori brushless

Sono motori di tipo sincrono, alimentati da un apposito circuito elettronico, in grado di variare la frequenza di alimentazione e quindi la velocità di rotazione. Sono i motori più usati in tutte quelle applicazioni che richiedano alte prestazioni dinamiche. Sono composti da uno statore con avvolgimenti opportunamente alimentati, in grado di generare un campo di orientazione e intensità arbitrarie. Sul rotore sono presenti magneti ad alto valore di campo coercitivo (Samario-cobalto o neodimio-ferro-boro). Rispetto ai motori sincroni tradizionali, i BLM hanno in più un sensore che legge la posizione rotorica. Un sistema elettronico usa questa informazione per alimentare le fasi statoriche in modo che il flusso generato sia sempre ortogonale all'asse magnetico rotorico. Questo implica che:

- il motore non perde il passo;

- la tensione di alimentazione alle fasi è controllata in modulo e fase.

Tenendo gli assi magnetici ortogonali (in "quadratura"), la coppia è sempre massima e dipende solo dalla corrente statorica (in modo proporzionale ad essa).

Si ha una certa somiglia al comportamento di un motore c.c., ma rispetto a quest'ultimo ha parecchi vantaggi:

- Non ha le spazzole;
- Meno attrito;
- Non c'è scintillio (ambienti esplosivi, EMI);
- Non c'è usura;
- Può operare nel vuoto;
- Non c'è manutenzione;
- Gli avvolgimenti sono a statore;
- Migliore smaltimento del calore;
- Minore inerzia rotorica (maggiori accelerazioni);
- Migliore bilanciamento e maggiori velocità;

Esistono due tipi di motori brushless (BLM), che si differenziano per come è distribuito il campo rotorico e, di conseguenza, per la forma della fem:

- Sinusoidale (AC brushless)
- Trapeziodale (DC brushless)

Entrambi hanno avvolgimenti a stella come mostrato in figura 4.1.

Se si vuole ottenere coppia costante, lo deve essere anche la corrente di fase, nel momento in cui la fem è al max (o min). La corrente resta attiva in due periodi di 120° e spenta per due da 60°

Come si nota in figura 4.2, non è necessario conoscere con precisione la posizione angolare del rotore, ma solo discriminare tra configurazioni distanti 60°. Per determinare la zona in cui si trova il rotore bastano quindi dei sensori molto economici, come i sensori di Hall. Le fasi opportune vengono accese come si nota in figura 4.3 mediane una modulazione PWM della corrente.

Le correnti di fase sono afflitte da ripple dovuto ai residui di PWM. Inoltre, la corrente non va al valore desiderato istantaneamente come mostrato in figura 4.4 : Ripple aggiuntivo di coppia.

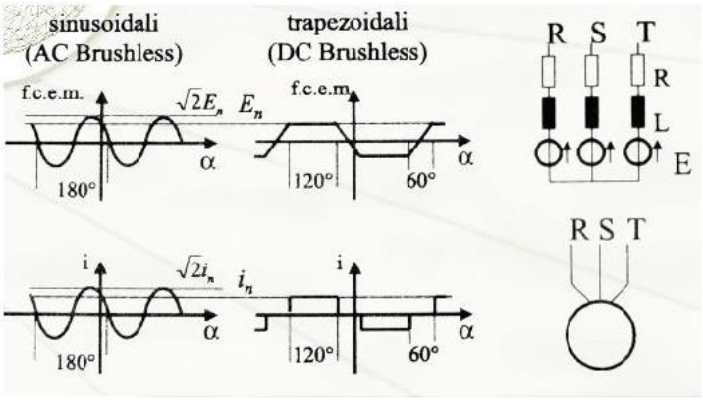


Figura 4.1. Forza elettromotrice nei motori Brushless

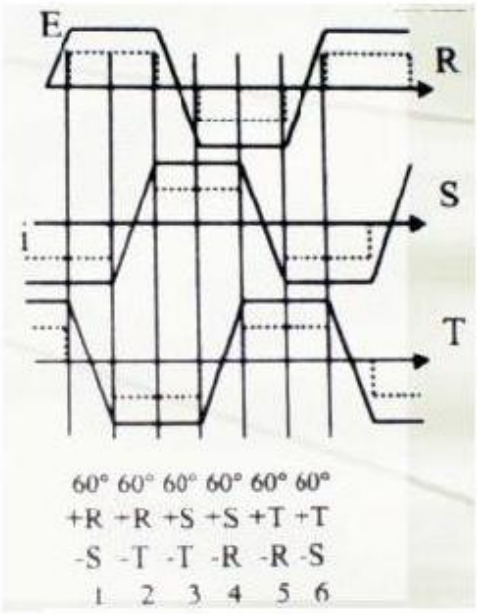


Figura 4.2. Corrente delle fasi del motore Brushless

4.3 Caratteristiche del motore

I motori EC maxon a commutazione elettronica sono motori brushless di elevata qualità con magneti in Neodimio. Contrariamente ai motori DC maxon, l'avvolgimento senza ferro è in posizione statorica ferma. Ruota invece, nel campo determinato dal campo trifase commutato elettronicamente, il magnete permanente. L'avvolgimento romboidale maxon è formato da tre settori parziali, a 120°. Questi settori sono

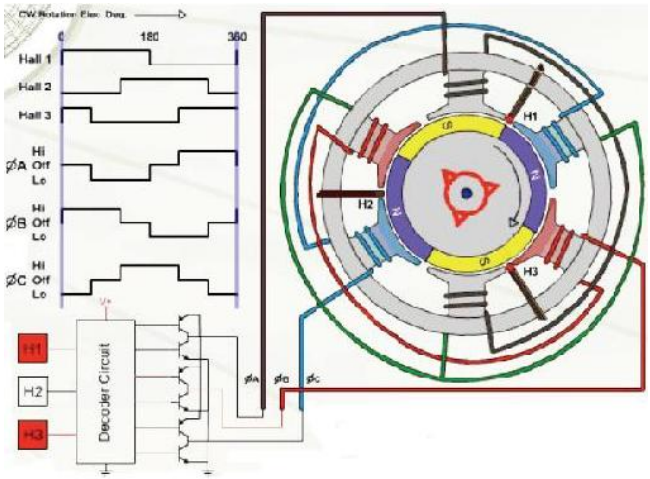


Figura 4.3. Fasi del motore Brushless

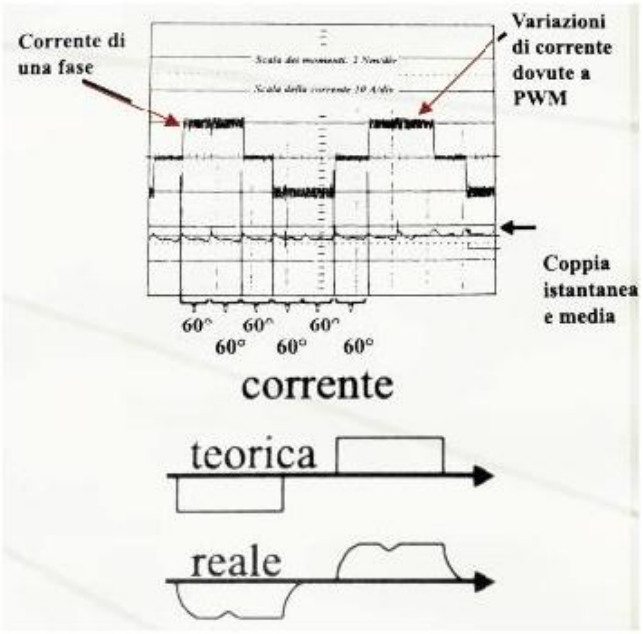


Figura 4.4. Corrente Motore

connessi a stella. Questo comporta che coppia e velocità di rotazione siano inversamente proporzionali con un fattore v3. La configurazione a stella comporta correnti del motore ridotte, tensioni più elevate e consente coppie elevate. Il riscontro della posizione del rotore avviene mediante tre sensori di Hall posti all'interno del motore. I tre sensori di Hall posizionati radialmente a 120° producono sei diverse combinazioni per giro. Essendo presenti 3 coppie di poli, i tre settori dell'avvolgimento sono collegati all'alimentazione da sei fasi di conduzione regolate in base alle informazioni dei sensori. L'andamento della corrente e della tensione sono a forma di gradino. Il

punto di commutazione di ognuno dei sensori a effetto Hall è spostato dal rispettivo picco di coppia di 30°. Nella seguente tabella vengono riportate le caratteristiche del motore.

Dati del motore	Valore	Unità
Potenza assegnata	6	W
Tensione nominale	9	Volt
Velocità a vuoto	8600	rpm
Coppia di stallo	20	mNm
Corrente a vuoto	110	mA
Corrente continua massima a 5000 rpm	1.03	A
Momento d'inerzia del rotore	13.9	gcm^2
Temperatura ambiente	-40/+100	°C
Peso del motore	32	g
Temperatura max. dell'avvolgimento	125	°C

Figura 4.5. Dati Motore

Il motore secondo le analisi termiche, lavora in un range di circa 0/+45°C, per cui entro i suoi limiti. La corrente del motore comporta un riscaldamento dell'avvolgimento a causa della resistenza dell'avvolgimento. Affinché il motore non si surriscaldi questo calore deve essere disperso nell'ambiente attraverso lo statore. L'avvolgimento autoportante è il punto termicamente critico. La temperatura massima del rotore non deve essere superata neppure per breve tempo. Essa è di 125°C. L'intervallo di funzionamento possibile per una temperatura ambiente di 25°C risulta quello rappresentato in figura:

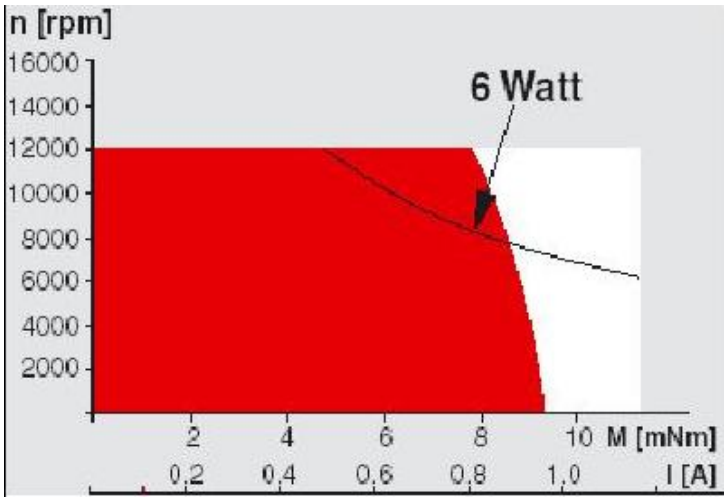


Figura 4.6. Intervallo di funzionamento a temperatura ambiente

Non essendo presente nello spazio la dissipazione per convezione la corrente massima fornibile al motore verrà limitata per evitarne il danneggiamento. Accorgimenti tecnici di montaggio, quali il fissaggio del motore su metallo fanno ridurre la resistenza termica, e si può diminuire sensibilmente la temperature. Si osserva anche che

la corrente a vuoto necessaria a vincere la sola inerzia del motore è di 110 mA. Non essendo disponibili al momento della progettazione dati sull'inerzia della ruota che verrà montata, si è stimata una corrente massima necessaria di 360 mA, comunque modificabile. Per come è costruito il motore, esso è in grado di ruotare ad una velocità attorno ai 50.000 rpm. Il limite imposto di 12.000 rpm è dettato dal fatto di garantire una durata di vita dei cuscinetti di almeno 20.000 ore in condizione di squilibrio residuo massimo ammesso del rotore.

4.4 Schema di controllo

A differenza di altri motori, il motore scelto necessita per muoversi, per come è stato costruito, di un apposito controllo.

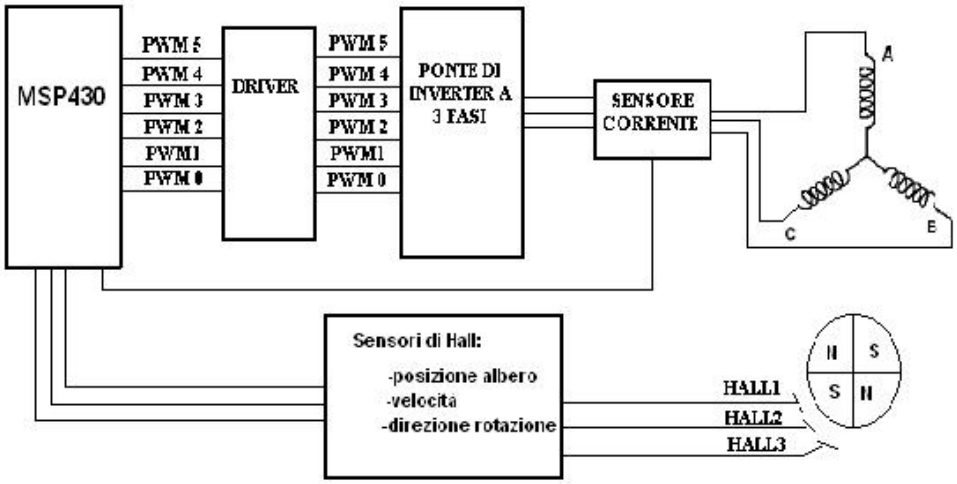


Figura 4.7. Schema a blocchi del controllo

Come mostrato in figura, l'MSP430 genera 6 uscite PWM, che sono collegate ad un blocco driver, che a loro volta sono collegati a sei MOSFET. Questi MOSFET sono collegati in una configurazione detta ponte di inverter a tre fasi, dove i tre avvolgimenti del motore prendono potenza. Nell'esecuzione continua, la tensione del MOSFET è massimo 7,5V, mentre la corrente è all'incirca di 1A. È importante quindi, garantire la dissipazione di calore sufficiente se si stanno usando le possibilità massime. I gate del MOSFET HI-side, inoltre richiedono una tensione più bassa (V-gate¡-16V) per funzionare, quindi questo livello di tensione deve essere fornito. I tre output del sensore ad effetto HALL sono collegati a tre pin di input dell'MSP abilitati a lavorare con l'interrupt. Se quindi avviene una variazione in ingresso a questi pin è generato un'interrupt.

La corrente al motore viene fornita da un resistore di basso valore (180 mO) collegato tra la tensione positiva e la terra. La tensione generata da questo resistore è amplificata da un INA146, il cui valore d'uscita è ancora comparato da un comparatore LMV331DBVR. L'uscita di quest'ultimo giunge in un pin dell' MSP anch'esso con interrupt abilitato.

Affinchè si abbia una rotazione dell'albero bisogna far girare il rotore. Si deve formare quindi un campo elettrico di rotazione. Il motore BLDC selezionato ha tre fasi dello statore, che devono essere eccitate due alla volta, per generare il necessario campo elettrico di rotazione. Questo è ragionevolmente facile da implementare, ma bisogna impedire che il magnete permanente del rotore rimanga bloccato con lo statore. A tale scopo l'eccitazione sullo statore deve essere ordinata in un modo specifico mentre si conosce la posizione esatta dei magneti del rotore. Le informazioni di posizione possono essere ottenute dai sensori di Hall che rilevano la posizione del magnete del rotore. Il motore con sensori è un trifase tipico, e in un ciclo elettrico vi sono sei regioni o settori distinti in cui due avvolgimenti specifici sono eccitati. I sensori di HALL, generano un codice a 3-bit con i valori che variano da 1 a 6. Ogni valore di codice rappresenta un settore su cui è posizionato il rotore. Ogni valore di codice, quindi, ci fornisce le informazioni sugli avvolgimenti che devono essere eccitati. Così può essere usata una semplice look-up table dal programma per determinare i due avvolgimenti specifici che devono essere eccitati. Si noti che la condizione '0 'e '7 'sono condizioni non valide per i sensori di hall. Il software dovrebbe quindi controllare l'eventuale presenza di questi valori ed eventualmente togliere tensione.

Un cambiamento dell'input su questi pin genera un'interruzione. Per ottenere una velocità variabile del motore BLDC, bisogna applicare una tensione variabile ai terminali degli avvolgimenti. Traducendo questo in termini digitali, la tensione variabile può essere ottenuta variando il duty-cycle del segnale PWM che va agli avvolgimenti del motore. Tramite i sensori di HALL è inoltre possibile stabilire il verso di rotazione e implementare un modesto controllo di velocità.

4.5 Controllo ad anello aperto

Nel controllo ad anello aperto è il PWM che controlla direttamente la velocità del motore che dipende dalla tensione media. Quando arriva un telecomando di rotazione del motore, si ha: l'inizializzazione del PWM, dell'ADC, e delle porte di I/O con interrupt, poi si da alimentazione alla logica del motore in modo che i sensori di hall possono essere letti.

Sulla base del loro valore, vengono attivate le due uscite opportune, in modo da fornire potenza al motore. A questo punto il motore si avvia, ed incomincia quindi a girare. Il valore del duty-cicle impostato al 50

Nella routine di interrupt, i sensori di hall sono letti ed in base al valore è possibile selezionare le nuove uscite per alimentare le fasi corrette. Questa procedura assicura

che sono eccitati gli avvolgimenti corretti nel giusto settore, ed il motore quindi continuerà a girare. Per frenare il motore si mettono gli avvolgimenti a massa in modo da facilitare l'operazione, una volta fermo le fasi vengono messe in alta impedenza (HI-Z).

4.6 Il blocco Driver

Esistono vari schemi per le applicazioni dell'azionamento del gate high-side. Questi includono trasformatori dell'azionamento del gate single-ended o double-ended, il driver bootstap ad alta tensione ICs, le tensioni di polarizzazione flottanti e l'azionamento opto-isolator. Le soluzioni di pilotaggio del gate con trasformatore risolvono il problema del pilotaggio high-side.

Il pilotaggio dell'avvolgimento che guida il gate MOSFET dell'alimentazione, può variare su tutto il potenziale e fornisce sia una tensione positiva che negativa dell'azionamento del gate. L'ingombro di tale soluzione non consente però l'utilizzo per questa applicazione. Il driver high-voltage half-bridge (di cui l'IR2301 è solo un esempio), come appare in figura fornisce una soluzione per pilotare high-side e non richiede all'utente di avere alcuna conoscenza dei trasformatori. Questi tipi di ICs utilizzano alta tensione e circuiti level-shiftig, insieme ad un condensatore di 'bootstrap', per fornire l'azionamento high-side del gate. Durante il tempo di accensione del FET Q2, il source di Q1 è a potenziale di terra. Ciò permette al condensatore C2 di caricarsi attraverso il diodo D alla tensione di polarizzazione Vcc. Quando Q2 è spento e Q1 è acceso, la tensione di source di Q1 comincia ad aumentare. Il condensatore C2 non può variare istantaneamente carica ai suoi capi per cui si porta ad una tensione Vs+Vcc e funge da sorgente di polarizzazione per la parte di pilotaggio high-side del driver e fornisce la corrente di carica per il gate di Q1. I circuiti level-shift del driver permettono allo stadio di pilotaggio high-side di spostarsi in su tramite la tensione di source di Q1.

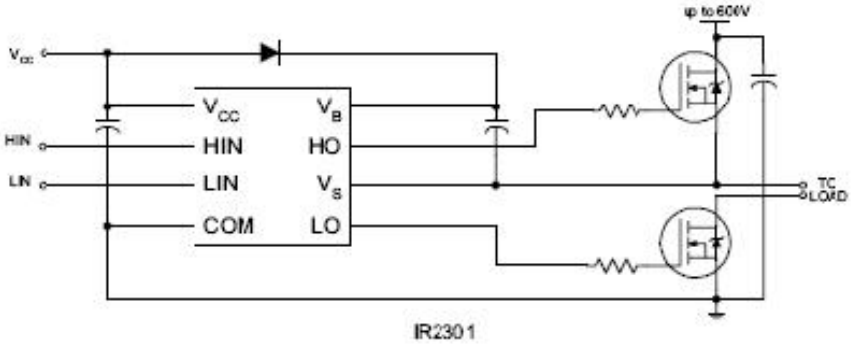


Figura 4.8. Driver

Uno degli svantaggi a molti di questi tipi di driver è il tempo di propagazione lungo tra il segnale in ingresso e il pilotaggio del high-side tra ON/OFF. Questo

risultato è dovuto ai circuiti di level-shift. Questi ritardi possono essere fra 500 nanosecondo e 1 microsec. Ciò può causare problemi per alcune applicazioni in alta-frequenza perché fa ritardare i periodi prendono troppo del periodo totale. Tale dispositivi però possono innescare durante il loro funzionamento un latch dinamico. Per tale motivo si è optato di non utilizzarlo. Si è utilizzata invece una soluzione semplice ma adeguata per le prestazioni richieste in questa applicazione.

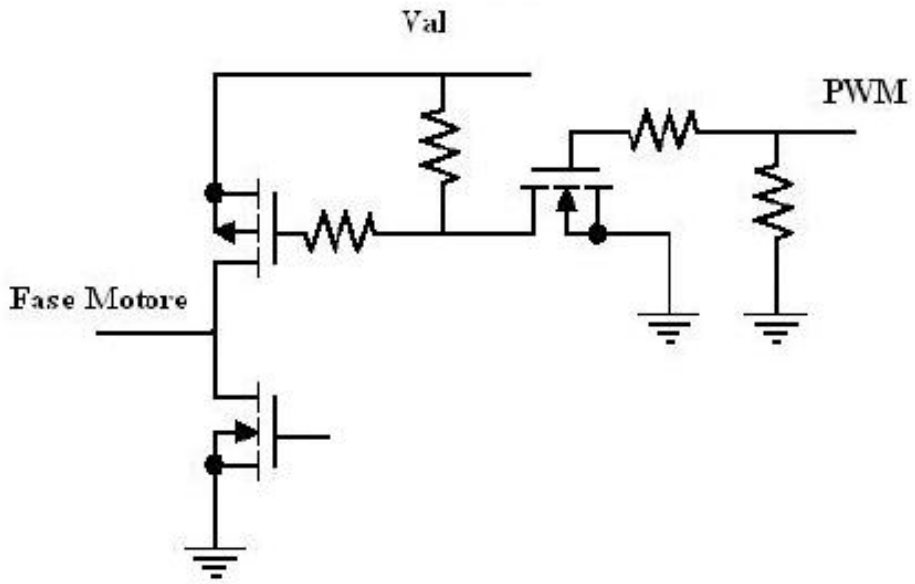


Figura 4.9. Pilotaggio in PWM

Si tratta di pilotare un p-MOS tramite un n-MOS. Supponiamo di fissare a 1L il segnale di PWM. Questo porterà in conduzione l'n-MOS portando a circa 0V il gate del p-MOS. Essendo il source del p-MOS a VCC, si avrà una VGS pari a VCC per cui il p-MOS condurrà anch'esso. Fissando invece uno 0L sul PWM si avrà una tensione sul gate del p-MOS pari a circa VCC per cui una VGS circa 0V e il p-MOS non conduce.

4.7 Sensori Hall

Come detto in precedenza tale motore è dotato si sensori di HALL. Il consumo di tali sensori non è specificato nel datasheet, ma da dai test effettuati risulta non trascurabile. Si ha infatti un assorbimento di 6,8mA con tensione di alimentazione di 5V. E' inoltre necessario inserire una resistenza di pull-up in uscita di ogni sensore come richiesto da datasheet. I valori dei sensori di HALL con le relative fasi da alimentare risulta il seguente.

Per fare una rotazione in senso antiorario è necessario abilitare, nello stesso settore, i rami HI-side e LO-side in modo opposto.

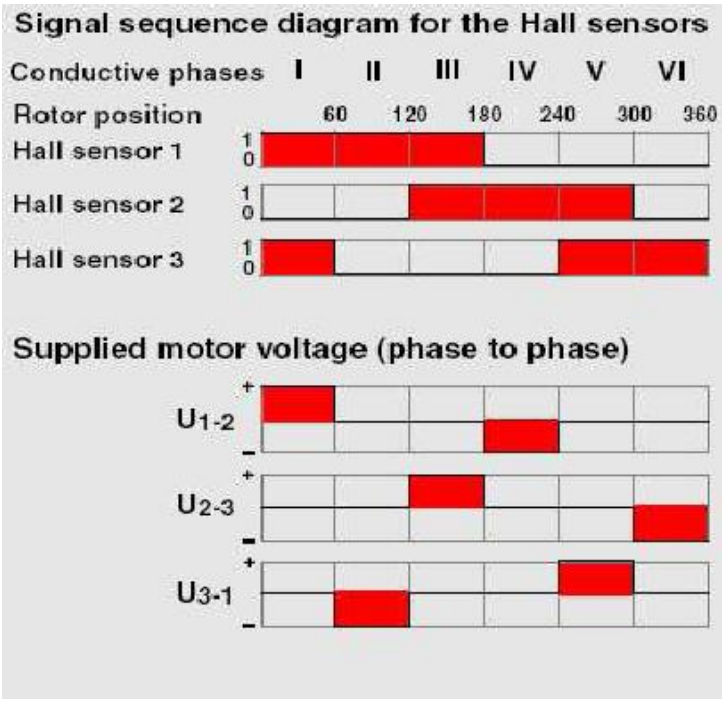


Figura 4.10. Fasi alimentate tramite sensori di hall

Capitolo 5

Il Software

Lo sviluppo della parte software costituisce il fulcro del lavoro svolto in questa tesi. In un primo momento si sono studiate le funzioni di libreria già sviluppate per la programmazione dei vari moduli del ProcB. Successivamente si è passati all'implementazione delle funzioni secondo le specifiche richieste. L'utilizzo dell'MSP430 ha consentito la possibilità di programmare in C anziché in assembler, velocizzando la fase di scrittura del software, in più, il programmatore JTAG usato con il programma IAR Embedded Workbench ha permesso di debuggare il software in maniera veloce ed approfondita, visto che era possibile visualizzare oltre al valore delle variabili, anche ogni singola cella di memoria ed ogni registro. Si passerà in questo capitolo alla descrizione in dettaglio delle operazioni eseguite e del codice che le implementa.

5.1 Diagramma degli stati

Il flusso di esecuzione del programma principale inizia ogni minuto, una volta avvenuta l'accensione della scheda da parte di PowerSwitch . Le operazioni da eseguire sono le seguenti:

- selezione del programma di boot;
- acquisizione delle misure;
- impostazioni delle batterie;
- ricezione dell'eventuale telecomando;
- esecuzione dell'eventuale telecomando;
- trasmissione a terra dei dati;
- richiesta di spegnimento.

La successione delle attività svolte dal microcontrollore è rappresentata nel seguente flowchart:

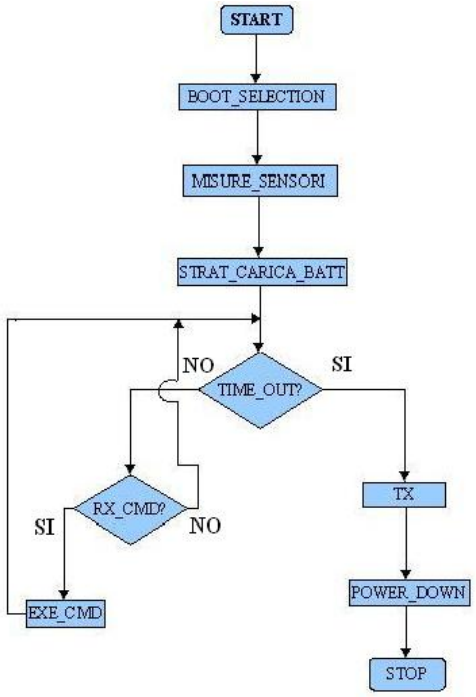


Figura 5.1. Flow chart

5.2 Boot selector

Ogni volta che il processore viene alimentato, esegue sempre lo stesso codice, il quale però è scaricato a rotazione da tre aree di memoria differenti allo scopo di evitare eventuali alterazioni della memoria programma dovuti alle radiazioni. L'area di memoria occupata dal codice è molto piccola per cui si riducono le possibilità che venga danneggiata. Successivamente si leggono tutte le altre variabili di boot. Tali variabili sono necessarie per conoscere la situazione in cui si trovava il processore prima dello spegnimento. L' algoritmo utilizzato per la lettura dalla memoria Flash delle variabili di boot è il seguente :

1. Inizializzazione del puntatore ad un indirizzo di memoria contenente lo stato del processore prima dell'ultimo spegnimento;
2. Lettura dell'ultimo comando di scatta foto inviato al Payload;
3. Lettura del timer riferito al tempo di carica delle batterie;
4. Lettura dell'ultima batteria sotto carica;
5. Viene spostato nuovamente il puntatore all'inizio dell'area di memoria destinata al boot;
6. Cancellazione dell'area di memoria occupata dalle variabili di boot.

```
*************************************************************************
 Lettura delle variabili di boot

*************************************************************************

void read_boot(void)
{
 char *Flash_ptr;

 Flash_ptr = INDIRIZZO_BOOT;  // Initialize Flash pointer

  last_foto[0]=*Flash_ptr;   // Lettura ultima foto scattata da Payload
  Flash_ptr++;
  last_foto[1]= *Flash_ptr;
  Flash_ptr++;
  timerH=*Flash_ptr;       // Lettura tempo di carica delle batterie
  Flash_ptr++;
  timerL=*Flash_ptr;
  Flash_ptr++;
  Old_batt=*Flash_ptr;      // Lettura ultima batteria sotto carica

 Flash_ptr = INDIRIZZO_BOOT;
 EraseFlashSegment(Flash_ptr);//Cancella il segmento per poter riscrivere
}

*************************************************************************
```

Per quanto riguarda la scrittura nella memoria Flash delle variabili di boot si è utilizzato il seguente algoritmo :

1. Inizializzazione del puntatore ad un area di memoria (vuota);
2. Disabilitazione di tutti gli interrupt;
3. Il controllore si pone in uno stato di attesa dal quale esce solo quando il valore del bit BUSY è uguale a 0.
4. Reset Lock bit per scrittura in Flash;
5. Set WRT bit per scrivere in Flash;
6. Scrittura delle variabili di boot nell'area di memoria allocata al punto 1;
7. Clear WRT bit;
8. Set Lock bit.

```
*********************************************************************
Scrittura delle variabili di boot

*********************************************************************

void write_boot(void)
{
  char *Flash_ptr;
  Flash_ptr = INDIRIZZO_BOOT;  // Initialize Flash pointer ;
  _DINT();                        //disabilita interrupt ;
  while(BUSY & FCTL3);   // attendi che il bit BUSY sia resettato;
  FCTL3 = FWKEY;                        // reset lock bit;
  FCTL1 = FWKEY + WRT;// enable write Set WRT  bit =1 for write operation

  *Flash_ptr=last_foto[0];
  Flash_ptr++;
  *Flash_ptr=last_foto[1];
  Flash_ptr++;
  *Flash_ptr=timerH;
  Flash_ptr++;
  *Flash_ptr=timerL;
  Flash_ptr++;
  *Flash_ptr=Old_batt;

  FCTL1 = FWKEY;                      // clear WRT  bit
  FCTL3 = FWKEY + LOCK;         // set lock bit
}

*********************************************************************
```

5.3 Misure dei sensori

Questa è la prima operazione che il ProcB fa entro 4 ” dall'accensione della scheda, quando cioè viene data alimentazione ai sensori di bordo del satellite. L'acquisizione A/D di tali misure avviene attraverso una lettura dei multiplexer presenti nelle schede Power Switch, Power Supply e ProcB stessa. L'acquisizione delle misure dei sensori avviene in diverse fasi. Innanzitutto viene selezionato l'indirizzo dei multiplexer, successivamente vengono acquisiti 64 campioni del valore dello stesso sensore e ne viene fatta una media in modo da ridurre i contributi del rumore all'interno del sistema. I valori acquisiti vengono quindi memorizzati in posizioni opportune nel vettore di telemetria. Il seguente flowchart rappresenta le operazioni svolte:

1. Inizializzazione del convertitore A/D a 12 bit.

Nella funzione di inizializzazione vengono settati i registri del convertitore A/D. In particolare vengono settati:

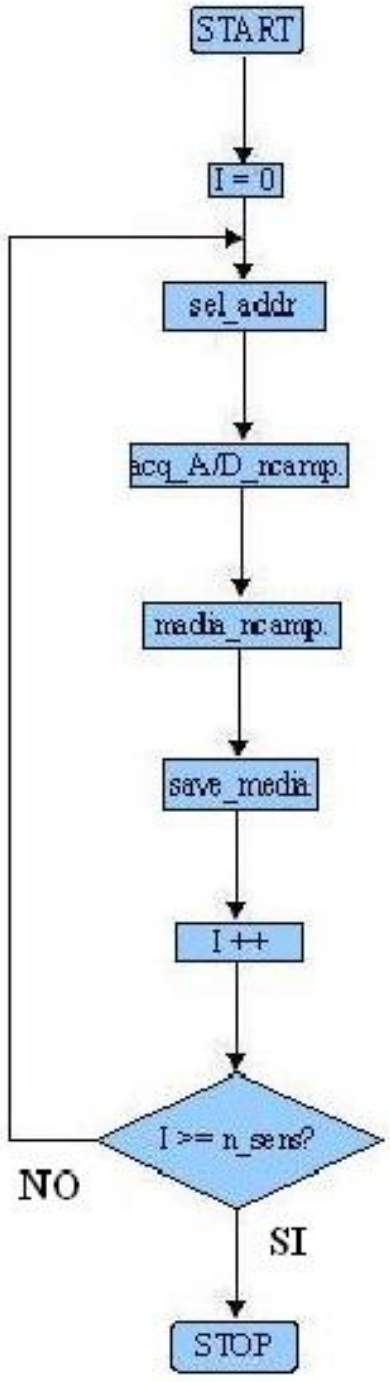


Figura 5.2. Flow chart

- il tempo di campionamento e mantenimento (**SHT03**) pari a

8Tck @ 4Mhz=2μs > 1,55μs(5.1)richiesti dal convertitore A/D per i registri che vanno da ADC12MEM0 a ADC12MEM7;

- il riferimento interno (**RFON**);
- il bit di accensione del convertitore A/D (**ADC12ON**);
- il bit Multiple Sample and Conversion (**MSC**) che fa si che il primo fronte di salita del segnale da campionare inneschi il timer di campionamento,e che il campionamento e le conversioni successive siano compiute automaticamente appena la conversione precedente è completata.
- i bit Sample and hold source select (**SHS0**) che seleziona come sorgente l'ADC12SC bit;
- i bit clock source select (**ADC12SSEL3**) che seleziona come sorgente il SMCLK;
- i bit Conversion sequence mode select (C**CONSEQ0**) che impongono una modalità d'acquisizione singolo canale, singola conversione.

2. Conversione e valor medio dei campioni acquisiti.

In questa funzione viene selezionato in primo luogo il riferimento a 1,5 V ed il canale interno prima della conversione. Successivamente viene abilitata l'acquisizione e la conversione. I valori convertiti vengono sommati e memorizzati nel registro di memoria **ADC12MEM0**.Viene fatta la media del valore ottenuto sul numero di campioni acquisiti. In seguito viene eseguito uno shift verso destra di 4 bit, ottenendo un valore di media su 6 bit in modo da rappresentare il valore in tensione rilevato dall'ADC misurato in centesimi di millivolt, evitando pertanto di utilizzare numeri in floating-point , più difficili da trattare. Infine il valore di media ottenuto, viene memorizzato nel vettore di telemetria nella j-esima posizione.

3. Assemblaggio del vettore di telemetria.

In questa funzione vengono selezionati il muxB ed il muxD per l'acquisizione dei valori di tensione, corrente e temperatura rispettivamente di PowerSupply e di PowerSwitch. Inoltre da PowerSwitch sono acquisite la data e l'ora corrente. Vengono inoltre selezionati da parte del ProcB i canali relativi all'acquisizione dei valori di temperatura e corrente delle schede Payload , TxRx e ProcB, di corrente del motore e tensione del ProcB. Quest'ultima avviene cambiando il riferimento interno, portandolo da 1,5V a 2,5V. E' stato quindi necessario aggiungere un ritardo SW di 20ms per stabilizzare il riferimento. Una volta acquisiti tutti i valori viene assemblato il vettore di telemetria. Il tempo di una singola acquisizione è pari a 367,2 *micro s*, mentre per l'acquisizione di tutti i 79 elementi del vettore di TMH occorrono 29ms.

4. Disattivazione convertitore A/D e del riferimento interno.

Questa funzione serve per disattivare il convertitore A/D e spegnere il riferimento interno perché consumano. Entrambi vengono disattivati dopo l'acquisizione di tutti i canali.

5.4 Strategia di carica batterie

Questa operazione viene fatta a cicli alterni dai due processori, sulle stesse batterie del satellite. All'accensione il processore non sa a priori la batteria che è selezionata. Per prima cosa quindi, tramite la letture di porte XOR, leggibili anche da ProcA, si individua la batteria che è attualmente sotto carica. Di questa batteria vengono letti i parametri seguenti:

- tempo di carica;
- temperatura;
- tensione.

Sulla base del tipo di batteria presa sotto esame, si vanno a confrontare questi parametri con i valori limite. Se i valori limite sono rispettati si lascia sotto carica altrimenti si seleziona una nuova batteria. Questa operazione prosegue per tutte le batterie. Se nessuna di queste soddisfa le condizioni, viene selezionata una resistenza di shunt per dissipare l'eccessiva carica che le batterie verrebbero ad avere.

5.5 Ricezione telecomando

La ricezione del telecomando avviene tramite il CC2400 che comunica col ProcB in SPI. Il settaggio dell'SPI avviene ad ogni avvio del ProcB insieme al settaggio di tutte le porte. Per la comunicazione con il tranceiver i registri dell'SPI sono settati nel modo seguente:

```
**********************************************************************

U0CTL  = CHAR + SYNC + MM + SWRST;// SW reset,8-bit transfer,SPI master
U0TCTL = CKPH + SSEL1 + STC;      // Data on Rising Edge, SMCLK, 3-wire

U0BR0  = 0x60;                          // SPICLK set baud(@ 11,363 kHz)
U0BR1  = 0x01;
U0MCTL = 0;                               // Dont need modulation control

ME1   |= USPIE0;                          // Module enable
U0CTL &= ~SWRST;                          // Remove RESET
```

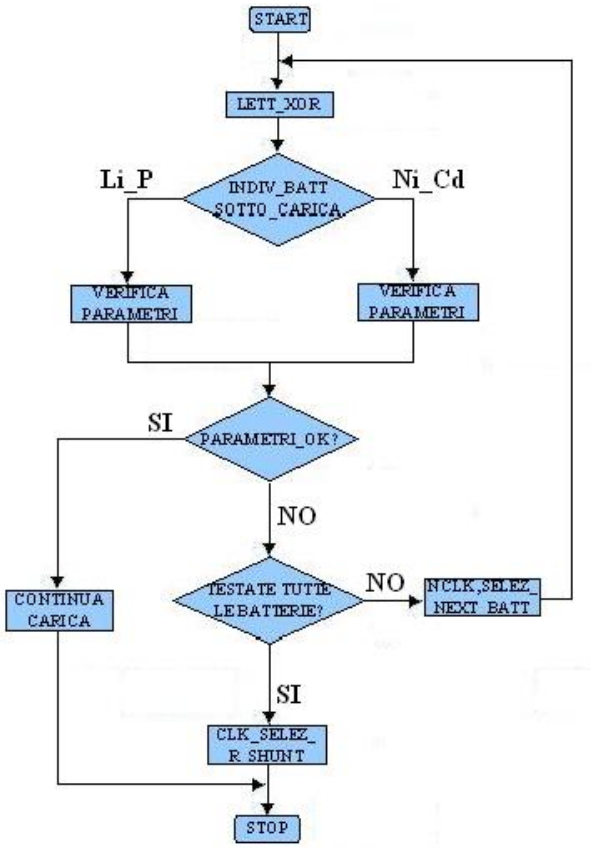


Figura 5.3. Strategia di carica batterie

****

Prima di poter utilizzare il tranceiver in modalità RX oppure TX è necessario che il quarzo sia attivo e stabile. E' necessario aspettare un tempo di start-up pari a circa 1,13 ms affinché l'oscillatore sia stabile. Per attivare l'oscillatore basta indirizzare il registro command strobe SXOSCON. In risposta alla chiamata di questo registro, il CC2400 invia un byte contenente alcune informazioni sullo stato del tranceiver. Il bit 7 denominato XOSC16MSTABLE se pari a 1 indica che l' oscillatore del CC2400 è funzionante e stabile.

Il passo successivo consiste nell'attivare il sintetizzatore di frequenza FS. E' necessario attendere un tempo di PLL lock time pari a circa 40us affinché il sintetizzatore di frequenza si agganci alla giusta frequenza. Quando il sintetizzatore FS è in aggancio è possibile programmare il CC2400 nella modalità RX oppure nella modalità TX.

A questo punto si resta in attesa di un telecomando fino a che non se ne riceve uno corretto o passa il timeout di 5s. Una volta ricevuto il telecomando se ne verifica la correttezza dello stesso. Se il telecomando è corretto viene interpretato ed eseguito, inviando successivamente a terra una telemetria di housekeeping (TMH) assieme

alla copia del telecomando ricevuto. Se il telecomando che arriva è errato si invierà comunque una copia del telecomando errato assieme al vettore di telemetria.

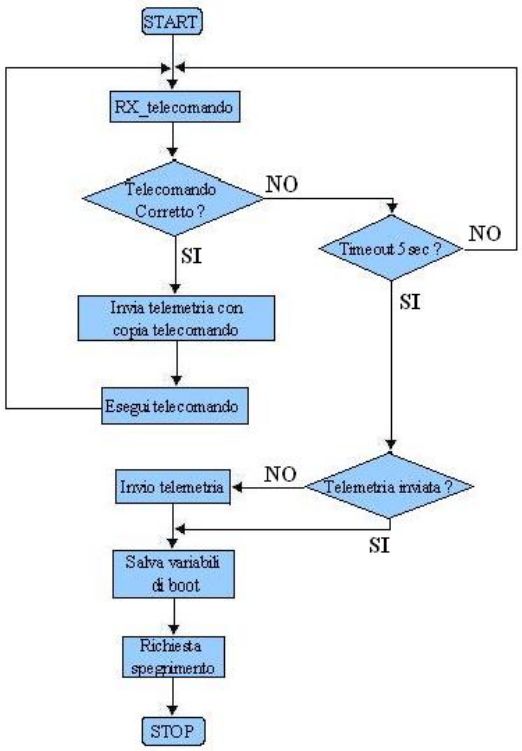


Figura 5.4. Flow chart

5.6 Esecuzione del telecomando

Una volta che il telecomando è stato acquisito ed elaborato, viene eseguito. I comandi possibili sono contenuti nella tabella seguente:

L'esecuzione dipende dal tipo di richiesta ricevuta.

5.6.1 Send TMH

Il vettore di telemetria di housekeeping, come già detto prima, viene assemblato ogni volta che il ProcB è attivato da PowerSwitch. Al suo interno contiene le misure dei sensori presenti su tutto il satellite. Viene inviato alla stazione di terra quando richiesto da telecomando,o, in mancanza di ricezione del telecomando entro 5 secondi dall'accensione.

Codice	*Comando*	*Descrizione*
0	*send* ***TMH***	Invio TMH
1	*send* ***TMHE***	Invio TMHE
2	*Reset* ***TMHE***	Azzeramento TMHE
3+(a-1)+(t-1)*5	*Trigger(**a,t**)*	Scatta foto ***a*** telecamera ***t***
1818+(a-1)	*Send (**a**)*	Invio foto ***a*** completa
1823+(b-1)	*Resend (**b**)*	Ritrasmissione blocco ***b*** dell'ultima foto trasmessa
1864+(x-1)	*Rotate(**x**)*	Rotazione della ruota d'inerzia di ±0, ±4, ±8, ...± 65536
1895+(a-1)*9+(b-1)	*Resend (**a,b**)*	Ritrasmissione blocco ***b*** della foto ***a***

5.6.2 Send TMHE

Richiede al processore di inviare verso terra la telemetria estesa, composta dai valori di telemetria max, minima, media e deviazione standard misurati negli ultimi 120 minuti. Ad ogni avvio del ProcB tale vettore viene aggiornato e memorizzato nella memoria flash in attesa che si riceva il comando di invio dalla stazione di terra. Come detto nel capitolo 3, il campo information field nel protocollo APRS può contenere al massimo 256 byte. Nel caso della telemetria estesa occorrono più di 256 byte, per cui si è pensato di inviare TMHE in due pacchetti.

Nel primo pacchetto si inviano i valori minimi e poi i valori massimi, nel secondo quelli di media e deviazione standard.

5.6.3 Reset TMHE

Questo comando, una volta ricevuto, comporta la cancellazione della telemetria estesa.

5.6.4 Scatta foto

```
**********************************************************************
U1BR0  = 0x20;   // SPICLK set baud: con quarzo a 4MHz (=125kHz)
U1BR1  = 0;      // Dont need baud rate control register 2 - clear it
U1MCTL = 0;      // Dont need modulation control

ME2   |= USPIE1; // Module enable
U1CTL &= ~SWRST; // Remove RESET
**********************************************************************
```

Il comando di scatta foto è codificato nel seguente modo:

$$3+(a\text{-}1)+(t\text{-}1)*5$$

Dove a indica il numero della foto (da 1 a 5) e t e la telecamera da utilizzare (da 1 a 3). Esso avrà quindi un valore compreso tra 3 e 17. Quando il telecomando è acquisito, il ProcB invia a terra una telemetria di conferma dell'avvenuta ricezione con la copia del telecomando, successivamente si incaricherà di comunicarlo al Payload.

Una volta che a Payload viene dato il comando di boot e viene alimentata, il ProcB invia il comando su due byte. Il primo byte che viene inviato è l'MSB. A questo punto ci si mette in attesa di una conferma da parte di Payload di un ACK (=1) per la corretta ricezione ed esecuzione del telecomando inviato, ed un ACK(=4) che viene inviato al termine della compressione della foto.

E' stato inoltre inserito un contatore per limitare l'attesa di risposta del Payload qualora avesse dei problemi.

```
**************************************************************************
if((comando>= 3) && (comando <= 17))
    {
    RX_to_TX_send_ACK(2);
    BOOTMODEPAY_B_1();    //seleziona il boot
    Run_Pay_33V_12V_ON();  //do tensione

    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(3200);

    send_Telecomando((unsigned char)(comando >> 8), (unsigned char)comando);

    contatore_timeout=0;
    do
      {
      SPI_ENABLE_PAY();
      FASTSPI_TX1_RX1(0x00,ACK_xx1);
      SPI_DISABLE_PAY();
      contatore_timeout++;
      }while((ACK_xx1!=1)&& (contatore_timeout < ((INT32)1100*110)));

    if (contatore_timeout < ((INT32)1100*110))
      {
      contatore_timeout=0;
      do
```

```
        {
        SPI_ENABLE_PAY();
        FASTSPI_TX1_RX1(0x00,ACK_xx1);
        SPI_DISABLE_PAY();
        contatore_timeout++;
        }while(ACK_xx1!=4);
      }
    //telecomando telemetria normale
    RX_to_TX_send_ACK(2); // lo invio se ricevo l'ack, altrimenti attendo
che mi spenga PowerSwitch

    Run_Pay_33V_12V_OFF();  //tolgo tensione

**********************************************************************
```

5.6.5 Invio foto completa

Il comando di invio foto completa è codificato nel modo seguente:

1818+(a-1)

Il codice che implementa il comando di invio foto è simile a quello visto prima:

- si seleziona il boot di Payload e la si alimenta,
- si aspetta che la tensione fornita dal ProcB a Payload arrivi ad un valore da permetterne il corretto funzionamento,
- viene rielaborato il codice del comando che sarà inviato al Payload,
- si invia il telecomando su due byte,
- si attende l'ACK della corretta ricezione del telecomando,
- si ricevono i 9 segmenti in cui è stata suddivisa la foto scattata dal Payload attraverso il protocollo XMODEM,
- si resta in attesa dell' ACK da parte di Payload del termine trasferimento immagine. Se questo non avviene in un tempo ragionevole, si invia a terra una telemetria di housekeeping con la copia del telecomando,
- si disabilita Payload.

```
**********************************************************************

if((1818 <=comando) && (comando<=1822))
```

```
{
RX_to_TX_send_ACK(2);
for(i=0; i<9; i++)
  {
  BOOTMODEPAY_B_1();    //seleziona il boot
  Run_Pay_33V_12V_ON();  //do tensione

  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(3200);

  comando1 = 1862 + (comando - 1818)*9 + i;

  send_Telecomando((unsigned char)(comando1>>8),(unsigned char)comando1);

  contatore_timeout = 0;
  do
    {
    SPI_ENABLE_PAY();
    FASTSPI_TX1_RX1(0x00,ACK_xx1);
    SPI_DISABLE_PAY();
    contatore_timeout++;
    }while((ACK_xx1!=1)&& (contatore_timeout < ((INT32)90*110)));

  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);

  if ((contatore_timeout < ((INT32)90*110)))
    {
    //ok stanno arrivando i dati
    contatore_timeout = 0;
    Xmodem_receive_and_send(comando - 1818 + 48);
    do
      {
      SPI_ENABLE_PAY();
      FASTSPI_TX1_RX1(0x00,ACK_xx1);
      SPI_DISABLE_PAY();
      contatore_timeout++;
```

```
          }while((ACK_xx1!=5)&& (contatore_timeout < ((INT32)90*110)));
        }
      else
        {
        //telecomando telemetria normale
        RX_to_TX_send_ACK(2);

        }

      Run_Pay_33V_12V_OFF();  //tolgo tensione

      }
}

*************************************************************************
```

5.6.6 Rotazione motore

In questa funzione viene alimentata la rotazione del motore con un segnale PWM. Leggendo i sensori di Hall a secondo della rotazione in senso orario o antiorario sappiamo la combinazione successiva dei sensori in quanto fornita dal datasheet ma non sappiamo l'attimo preciso in cui commuta, per questo usiamo la funzione di interrupt.

Tra una combinazione e la successiva cambia un solo bit . Tutti e tre i bit compiono, in un ciclo elettrico (6 commutazioni) sia una commutazione 0-1 che 1-0. Essendo l'interrupt sensibile solo a un fronte selezionabile (salita o discesa) ad ogni interrupt si seleziona il fronte successivo del corrispondente pin che deve cambiare. Successivamente si mettono le uscite a 0=OUTMOD0, quindi le fasi del motore in HI-Z e si selezionano le nuove uscite in PWM = OUTMOD7. Vediamo ora di descrivere le operazioni effettuate nella routine. Come prima cosa si disabilitano gli interrupt, vengono inizializzate le periferiche del PWM e del convertitore A/D. Successivamente viene data alimentazione alla logica del motore. Con questa operazione vengono attivati i sensori di Hall che accodano una richiesta di interrupt che viene resettata. A questo punto si esegue una prima lettura dei sensori di Hall, si alimentano le fasi corrette e il motore inizia la rotazione. Prima che il motore abbia ruotato di un settore, si abilitano gli interrupt e si va in low power mode. La rotazione del primo settore fa variare i sensori di Hall che scatenano un interrupt. Da questo punto in poi, il controllo della rotazione passa alla routine di interrupt. Ad ogni rotazione di un settore si entra nella routine di interrupt. Si disabilita l'interrupt e si verifica subito che non sia entrato a causa di un malfunzionamento. Contemporaneamente viene verificato se il numero di giri è terminato. Se si devono fare ancora rotazioni, si leggono i sensori di Hall, ed a seconda del senso di rotazione, si impostano i nuovi fronti di sensibilità degli interrupt, si alimentano le fasi e si abilitano gli interrupt. Una volta completati i numeri di giri prima di arrestare

si misura il consumo a regime. Poi si spegne il motore, il riferimento del convertitore A/D, si disabilitano gli interrupt e si va in active mode. Una volta terminato, l'esecuzione torna al main. Si riporta di seguito il codice relativo alla routine di interrupt per il solo verso antiorario:

```
**************************************************************************

#pragma vector=PORT1_VECTOR
__interrupt void INT_Port1(void){

//if((N_giri !=0) && ((P1IN&0x08)!=0x08) && (P1IN!=0x07))
 if((N_giri !=0) && (P1IN!=0x07))
   {
  _DINT();
  N_giri--;
  if(spin==1) // Verso antiorario
  {
  P1IFG =0x00;   // Disabilito flag interrupt porta1
   if ((P1IN |0xF8)==0xFD){  //60 gradi Hall3,Hall2,Hall1=>101
    P1IES=BIT1+BIT2 ; //il ciclo dopo HALL 3 ha transizione 1->0 è attiva P1.2 BIT2=1
    P1IFG =0x00;
    TBCCTL5 = OUTMOD_0; //due cicli prima aveva attivo 1->0 in P1.1 BIT1=1 immutato
    TBCCTL1 = OUTMOD_7; //il ciclo prima aveva attivo 0->1 in P1.0 BIT0=0 immutato
   }
  else if ((P1IN |0xF8)==0xF9){//120 gradi Hall3,Hall2,Hall1=>001
   P1IES=BIT2;           //il ciclo dopo HALL 2 ha transizione 0->1 è attiva P1.1 BIT1=0
   P1IFG =0x00;
   TBCCTL4 = OUTMOD_0;//due cicli prima aveva attivo 0->1 in P1.0 BIT0=0 immutato
   TBCCTL6 = OUTMOD_7;    //il ciclo prima aveva attivo 1->0 in P1.2 BIT2=1 immutato
   }
   else if ((P1IN |0xF8)==0xFB){  //180 gradi Hall3,Hall2,Hall1=>011
   P1IES=BIT0+BIT2; //il ciclo dopo HALL 1 ha transizione 1->0 è attiva P1.0 BIT0=1
   P1IFG =0x00;
   TBCCTL1 = OUTMOD_0; //due cicli prima aveva attivo 1->0 in P1.2 BIT2=1 immutato
   TBCCTL3 = OUTMOD_7; //il ciclo prima aveva attivo 0->1 in P1.1 BIT1=0 immutato
   }
   else if ((P1IN |0xF8)==0xFA){  //240 gradi Hall3,Hall2,Hall1=>010
   P1IES=BIT0;  //il ciclo dopo HALL 3 ha transizione 0->1 è attiva P1.2 BIT2=0
   P1IFG =0x00;
   TBCCTL6 = OUTMOD_0; //due cicli prima aveva attivo 0->1 in P1.1 BIT1=0 immutato
   TBCCTL2 = OUTMOD_7;  //il ciclo prima aveva attivo 1->0 in P1.0 BIT0=1 immutato
   }
   else if ((P1IN |0xF8)==0xFE){  //300 gradi Hall3,Hall2,Hall1=>110
   P1IES=BIT0+BIT1;  //il ciclo dopo HALL 2 ha transizione 1->0 è attiva P1.1 BIT1=1
   P1IFG =0x00;
   TBCCTL3 = OUTMOD_0; //due cicli prima aveva attivo 1->0 in P1.0 BIT0=1 immutato
   TBCCTL5 = OUTMOD_7;  //il ciclo prima aveva attivo 0->1 in P1.2 BIT2=0 immutato
   }
   else if ((P1IN |0xF8)==0xFC){ //360 gradi Hall3,Hall2,Hall1=>100
   P1IES=BIT1;   //il ciclo dopo HALL 1 ha transizione 0->1 è attiva P1.0 BIT0=0
   P1IFG =0x00;
```

```
   TBCCTL2 = OUTMOD_0; //due cicli prima aveva attivo 0->1 in P1.2 BIT2=0 immutato
   TBCCTL4 = OUTMOD_7;   //il ciclo prima aveva attivo 1->0 in P1.1 BIT1=1 immutato
      }
      }

*******************************************************************************************
```

5.7 Richiesta di spegnimento

Per effettuare tale operazione si usa un segnale attivo sulla transizione 1-0. Quando ProcB viene attivato, questo segnale si trova allo stato logico 0. Si esegue quindi un transizione 0-1 che viene ignorata da PSW. La transizione 1-0 da questo momento in poi disattiverà la scheda ProcB. Se PowerSwitch continua ad alimentare il ProcB, si ripete la transizione 0-1, 1 rhd0 fino a quando non avviene la disattivazione.

```
*************************************************************************
_nPowOffB;                        //dummy off procB

for (cicli_reset=0;cicli_reset<17;cicli_reset++)
{

  for (attesa_reset=0;attesa_reset<34;attesa_reset++)//aspetto 2.2 secondi
  {
        halWait(32000);
  }

  PowOnB;
  P4OUT |=  BM(nPowOffB   ) ;

  for (attesa_reset=0;attesa_reset<20;attesa_reset++)//aspetto 1,2 secondi
  {
        halWait(32000);
  }

  _nPowOffB;
}

*************************************************************************
```

Capitolo 6

Collaudo del software

La fase di collaudo può essere suddivisa in due fasi:

- una fase fatta dal collaudo del ProcB con le singole schede componenti il satellite;
- una fase di integrità delle schede fatta a satellite completamente montato.

Per prima cosa si è verificata la funzionalità dei vari moduli dell' MSP con le funzioni sviluppate precedentemente. Successivamente si è passati alla verifica del software vero e proprio per la gestione delle funzioni del satellite. In questa fase la possibilità di debuggare il programma col tool IAR tramite JTAG ha permesso di velocizzare molto il lavoro. In debug, infatti, si può eseguire passo passo il programma visualizzando le variabili e qualsiasi altro registro.

6.1 Collaudo con Power Switch

Il collaudo con questa scheda è stato fatto verificando il funzionamento di tutte le interfacce:

- attivazione dell'alimentazione ogni minuto,
- intervento dei circuiti anti latch-up,
- acquisizione A/D,
- comunicazione tramite SPI.

Le prove di comunicazione tra le due schede tramite SPI sono quelle che nella gestione PowerSwitch hanno richiesto più tempo. Sono state fatte sostanzialmente 2 correzioni software, una per scheda. La prima riguardava la scheda ProcB, in quanto i dati ricevuti non risultavano corretti perché non era corretto il tempo di attesa della prima lettura/scrittura, in quanto PowerSwitch non aveva ancora

generato il pacchetto da inviare. La seconda correzione ha riguardato PowerSwich, in quanto si è scoperto debaggando, che dopo un primo ciclo corretto, al minuto successivo PowerSwitch si bloccava in un loop. Ciò era dovuto alla chiamata di una funzione dentro un interrupt che non dava errori in compilazione. Tale funzione è stata spostata nel main e si è utilizzato un flag aggiuntivo.

6.2 Collaudo con PowerSupply

Con PowerSupply sono state verificate l'acquisizione A/D e la strategia di carica delle batterie. In particolare si è verificato la correttezza dei collegamenti del connettore e la sequenza di selezione dei vari indirizzi che in un primo tempo non risultava totalmente corretta. Si è anche tarato il tempo di selezione tra un canale e l'altro per avere un' acquisizione corretta, pari a

$$4{,}8\mu s.$$

In seguito si è verificata la correttezza dei valori campionati con quelli misurati.

6.3 Collaudo con Payload

Nel collaudo con Payload si è verificata come prima cosa l'acquisizione corretta del telecomando inviato dal ProcB. Il primo problema riscontrato è stato quello della mancata ricezione da parte del procB dei rispettivi ACK inviati da Payload. Per risolvere tale problema si è fatto uso dell'oscilloscopio con cui si sono osservati i segnali di comunicazione tra le due schede:

MISO: Master In Slave Out;

MOSI: Master Out Slave In;

CLK: Clock;

SPISEL: Segnale di selezione della porta SPI.

Osservando i quattro segnali all'oscilloscopio si è potuto notare che la risposta che Payload mandava a ProcB era sfasata di un bit. Il problema era dovuto quindi alla perdita del sincronismo durante la comunicazione. Si è pertanto adottata come soluzione al problema l'abilitazione e la disabilitazione del segnale $\mathrm{SPI}_S EL per ogni byte inviato e ricevuto in modo da evitare quindi le perdite di sincronismo.$

Successivamente si è testato l'invio e la ricezione delle foto scattate, tramite il protocollo XMODEM che è avvenuta senza riscontrare problemi.

6.4 Scheda PowerSwitch e TxRx

Per comunicare con la scheda TxRx, ProcB si serve della scheda PowerSwitch che fa da intermediaria tra le due. Tutte e tre le schede sono state testate a coppie risultando funzionanti. Ma una volta assemblate insieme il sistema risultava non funzionante. Il problema era dovuto all'errata mappatura di alcuni segnali nel connettore tra TxRx e PSW. Corretti tali segnali, si è potuto verificare il corretto funzionamento di ProcB nella programmazione TxRx e nell'invio dati, in TxRx il corretto funzionamento della parte RF e in PowerSwitch la corretta temporizzazione nel dare alimentazioni e nella gestione dei latch-up.

6.5 Collaudo a satellite montato

Le modifiche fatte al software nella fase di collaudo a satellite montato hanno riguardato più che altro le tempistiche ed i ritardi introdotti per garantire il corretto funzionamento del satellite. In particolare l'introduzione di ritardi hanno riguardato soprattutto le schede Payload e TxRx. Con la scheda Payload si sono riscontrati problemi che in seguito si sono scoperti dovuti all'hardware della scheda stessa. La parte software di ProcB è stata modificata invece aggiungendo un ritardo pari a 0,6 secondi dopo il comando di alimentazione per Payload in quanto la tensione che era fornita dal ProcB non cresceva istantaneamente al valore necessario per permetterne il corretto funzionamento. Stesso discorso vale per la scheda TxRx per la quale è stato aggiunto un ritardo pari a 0,64 secondi per attendere che la power supply di TxRx vada a regime.

Capitolo 7

Conclusioni

Durante questa tesi, durata 9 mesi, si è sviluppata e collaudata la parte software riguardante la gestione della scheda ProcB. Si è passati innanzitutto ad esaminare quali fossero le specifiche di progetto richieste che sono state di volta in volta concordate con i professori e i tesisti delle varie schede di PiCPoT. Si sono acquisite conoscenze nella programmazione embedded e nell'utilizzo dell'ambiente di sviluppo IAR Embedded Workbench. Tutte le interfacce di ProcB col resto del satellite sono state sviluppate e testate singolarmente ed a satellite montato, risultando perfettamente funzionanti. Si è avuta la possibilità di utilizzare strumenti di laboratorio come oscilloscopi e analizzatori di spettro andando a vedere le forme d'onda e da queste capire quali fossero le eventuali modifiche da effettuare. Aspetto di rilevante importanza nello sviluppo di questo lavoro di tesi è stato quello di aver lavorato all'interno di un team. Oltre ad aver ampliato le conoscenze specifico-tecniche riguardanti il progetto in sé, è stato possibile affrontare e superare gli aspetti riguardanti il confronto interpersonale tra i componenti del gruppo di lavoro per apprendere quelle che sono le metodologie basilari al fine di una efficace ed efficiente collaborazione produttiva.

Appendice A

Listato

Listato.c

```
#include "pictop.h"

int main( void )
{
 int i;
 int comando,comando1;
 INT32 contatore_timeout;
 unsigned char codice_blocco;
  unsigned char attesa_reset,cicli_reset;

do
  {

  comando_ricevuto = 0;

    WDTCTL = WDTPW + WDTHOLD;                              // Stop WDT

    setupXT2();
    set_porte();
    read_boot();

    PowOnB;

    TMH_giorni_ora();        // acquisizione dati da psw
    TMH();                   // acquisizione telemetrie
    carica_batteria();       // strategia di carica batterie

    // per attendere che power supply di TxRx arrivi a regime

    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
```

```
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);


    ReadFlash(contatore,(char *)&N_TMH,sizeof(int));

    // se leggo 0xff e' perche' la variabile e' stata cancellata!!!!!!
    N_TMH++;

    if ( N_TMH != 0xfe) // se ho gia' registrato 254 telemetrie mi fermo
      {
      if (N_TMH == 0)
        {
        // e' la prima telemetria
        // telemetria_max = telemetria
        for (i = 0; i < Num_of_elements; i++)
          {
          telemetria_max[i] = telemetria[i];    // inizializzazione vettori TMHE
          telemetria_min[i] = telemetria[i];
          telemetria_media[i] = (int)telemetria[i];
          telemetria_devstd[i] = (INT32)telemetria[i]*(INT32)telemetria[i];
          }
        }
  else
 {
// non e' la prima
ReadFlash(vet_max,(char *)telemetria_max,Num_of_elements);
ReadFlash(vet_min,(char *)telemetria_min,Num_of_elements);
ReadFlash(vet_media,(char *)telemetria_media,Num_of_elements*sizeof(int));
ReadFlash(vet_deviazione_std,(char *)telemetria_devstd,Num_of_elements*sizeof(INT32));

for(i = 0; i < Num_of_elements; i++)
{
if(telemetria[i]>telemetria_max[i])     telemetria_max[i]=telemetria[i];
if(telemetria[i]<telemetria_min[i])     telemetria_min[i]=telemetria[i];
telemetria_media[i]+=(int)telemetria[i];
telemetria_devstd[i]+=(INT32)telemetria[i]*(INT32)telemetria[i];
}
        }

      EraseFlashSegment(inizio_flash);
      EraseFlashSegment(inizio_flash+512);

      // scrittura in flash
WriteFlash(vet_max,(char *)telemetria_max,Num_of_elements);
WriteFlash(vet_min,(char *)telemetria_min,Num_of_elements);
WriteFlash(vet_media,(char *) telemetria_media,Num_of_elements*sizeof(int));
```

```
WriteFlash(vet_deviazione_std,(char *)telemetria_devstd,Num_of_elements*sizeof(INT32));

      // non faccio l-incremento perche' ho gia' incrementato all'inizio
      WriteFlash(contatore,(char *)&N_TMH,sizeof(int));
      }
      else
      {
        // carico comunque le telemetrie estese...
 ReadFlash(vet_max,(char *)telemetria_max,Num_of_elements);
 ReadFlash(vet_min,(char *)telemetria_min,Num_of_elements);
 ReadFlash(vet_media,(char *)telemetria_media,Num_of_elements*sizeof(int));
 ReadFlash(vet_deviazione_std,(char *)telemetria_devstd,Num_of_elements*sizeof(INT32));

      }

    // scrivo i dati di boot
    write_boot();

    N_giri=1;
    // azzero il vettore comando
    for(i=0; i <10; i++)
      {
      TC[i] = 0;
      }

    // attendo 5 secondi in ricezione
    if ( wait_for_rx())
      {
      for(i=0; i <10; i++)
        {
        FASTSPI_RX(TC[i]);
        }
      SPI_DISABLE_RX();
      comando_ricevuto = 1;
      }
    else
      {
      //no telecomando
      TC[0]= 0xE0;
      TC[1]= 0x67;
      TC[2]= 0x44;
      TC[3]= 0xC8;
      TC[4]= 0x7B;
      TC[5]= 0x40;
      TC[6]= 0x01;
      TC[7]= 0x42;
      TC[8]= 0xb1;
      TC[9]= 0x99;
      }

  comando = command_decode(TC);
```

```
if (comando == 1)
  {
  //calcolo della telemetria estesa
  for(i = 0; i < Num_of_elements; i++){
    telemetria_media_invio[i]=(char)(telemetria_media[i]/(N_TMH + 1));
    telemetria_devstd_invio[i]=(char)sqrt((telemetria_devstd[i]/(N_TMH + 1))
    - (telemetria_media[i]*telemetria_media[i]));
    }
  RX_to_TX_send_ACK(1);
  }
else if(comando==2) //resetTMHE
  {
  delete_TMH();
  RX_to_TX_send_ACK(2);
  }
else if (comando == 0)
  {
  //telecomando telemetria normale
  RX_to_TX_send_ACK(0);
  }
else if((1864 <=comando) && (comando<=1894))
  {
  RX_to_TX_send_ACK(2);
  comando1=comando-1863;
  if(comando1!=1)
    {
    TBCTL = TBSSEL_2 + MC_1;  // SMCLK, up mode ACLK=TBSSEL_1,attiva Timer B
    POWER_ON_MOTORE();     // Si alimentano il motore e la relativa logica
    if(comando1<=16)
      {
      spin=0;                                  // verso orario
      ON_OFF_motor(comando1);                  // accende e spegne il motore
      }
    else
      {
      comando1-=15;
      spin=1;                                 // verso antiorario
      ON_OFF_motor(comando1);                 // accende e spegne il motore
      }
    }
  else
  N_giri=0;
  }
else if((1818 <=comando) && (comando<=1822))
  {
  RX_to_TX_send_ACK(2);
  for(i=0; i<9; i++)
    {
    BOOTMODEPAY_B_1();     // seleziona il boot
    Run_Pay_33V_12V_ON();  // do tensione
```

```
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(3200);

comando1 = 1862 + (comando - 1818)*9 + i;

send_Telecomando((unsigned char)(comando1 >> 8), (unsigned char)comando1);

contatore_timeout = 0;
do
  {
  SPI_ENABLE_PAY();
  FASTSPI_TX1_RX1(0x00,ACK_xx1);
  SPI_DISABLE_PAY();
  contatore_timeout++;
  }while((ACK_xx1!=1)&& (contatore_timeout < ((INT32)90*110)));

halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);
halWait(32000);

if ((contatore_timeout < ((INT32)90*110)))
  {
  // ok stanno arrivando i dati
  contatore_timeout = 0;
  Xmodem_receive_and_send(comando - 1818 + 48);
  do
    {
    SPI_ENABLE_PAY();
    FASTSPI_TX1_RX1(0x00,ACK_xx1);
    SPI_DISABLE_PAY();
    contatore_timeout++;
    }while((ACK_xx1!=5)&& (contatore_timeout < ((INT32)90*110)));
  }
else
  {
  // telecomando telemetria normale
  RX_to_TX_send_ACK(2);
```

```
      }

    Run_Pay_33V_12V_OFF();  // spengo tensione

    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(32000);
    halWait(3200);
    }   // fine for i


  }
else if((comando>= 3) && (comando <= 17))
  {
  RX_to_TX_send_ACK(2);
  BOOTMODEPAY_B_1();     // seleziona il boot
  Run_Pay_33V_12V_ON();  // do tensione

  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(3200);

  send_Telecomando((unsigned char)(comando >> 8), (unsigned char)comando);

  contatore_timeout=0;
  do
    {
    SPI_ENABLE_PAY();
    FASTSPI_TX1_RX1(0x00,ACK_xx1);
    SPI_DISABLE_PAY();
    contatore_timeout++;
    }while((ACK_xx1!=1)&& (contatore_timeout < ((INT32)1100*110)));

  if (contatore_timeout < ((INT32)1100*110))
    {
    contatore_timeout=0;
    do
      {
```

```
      SPI_ENABLE_PAY();
      FASTSPI_TX1_RX1(0x00,ACK_xx1);
      SPI_DISABLE_PAY();
      contatore_timeout++;
      }while(ACK_xx1!=4);
    }
  // telecomando telemetria normale
  RX_to_TX_send_ACK(2);
  // lo invio se ricevo l'ack,altrimenti attendo che mi spenga powerswitch


  Run_Pay_33V_12V_OFF();  // do tensione

  last_foto[1]=last_foto[0];
  last_foto[0] = (comando & 7) - 3;
  }
else if (((comando>= 1823)&&(comando <= 1831))
||((comando>=1895) && (comando<=1939)))
  {
  RX_to_TX_send_ACK(2);
  // resend singolo blocco
  if (comando <= 1831)
  {
    comando1 = 1862 + (last_foto[0])*9 + (comando - 1823);
    codice_blocco = (char)(comando - 1818 + 48);
  }
  else
  {
    comando1 = comando - 33;
    codice_blocco = (char)(((comando-1895) % 9) + 0x30);
  }

  BOOTMODEPAY_B_1();     // seleziona il boot
  Run_Pay_33V_12V_ON();  // do tensione

  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(3200);

  send_Telecomando((unsigned char)(comando1 >> 8), (unsigned char)comando1);

  contatore_timeout = 0;
  do
    {
```

```
    SPI_ENABLE_PAY();
    FASTSPI_TX1_RX1(0x00,ACK_xx1);
    SPI_DISABLE_PAY();
    contatore_timeout++;
    }while((ACK_xx1!=1)&& (contatore_timeout < ((INT32)90*110)));

  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);

  if ((contatore_timeout < ((INT32)90*110)))
    {
    // ok stanno arrivando i dati
    contatore_timeout = 0;
    Xmodem_receive_and_send(codice_blocco);
    do
      {
      SPI_ENABLE_PAY();
      FASTSPI_TX1_RX1(0x00,ACK_xx1);
      SPI_DISABLE_PAY();
      contatore_timeout++;
      }while((ACK_xx1!=5)&& (contatore_timeout < ((INT32)90*110)));
    }
  else
    {
    // telecomando telemetria normale
    RX_to_TX_send_ACK(2);

    }

  Run_Pay_33V_12V_OFF();  // spengo tensione

  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(32000);
  halWait(3200);
  }
else
  {
  // invio la telemetria automaticamente + la copia del telecomando sbagliato
```

```
    RX_to_TX_send_ACK(2);
    }
}while (comando_ricevuto != 0);

// spengo il proc
_nPowOffB;

for (cicli_reset = 0; cicli_reset < 17; cicli_reset++)
{

    //aspetto 2.2 secondi
    for (attesa_reset = 0; attesa_reset < 34; attesa_reset++)
    {
        halWait(32000);
    }

    PowOnB;
    P4OUT |=  BM(nPowOffB   ) ;

    //aspetto 1,2 secondi
    for (attesa_reset = 0; attesa_reset < 20; attesa_reset++)
    {
        halWait(32000);
    }

    _nPowOffB;

}

    WDTCTL = 0;                             // Reset


}




// Wait for RX

int wait_for_rx(void) {
int status;

     _DINT();

    FASTSPI_SETREG_RX(CC2400_MAIN, 0x0000);
    FASTSPI_SETREG_RX(CC2400_MAIN, 0x8000);

        FASTSPI_SETREG_RX(CC2400_FSDIV,0x0987);
    // frequenza: 2439 MHz(in ricezione si riceve 1 mega in meno della tx)
        FASTSPI_SETREG_RX(CC2400_MANAND,0x7FFF); // override
        FASTSPI_SETREG_RX(CC2400_MDMCTRL,0x0020);
```

```
        FASTSPI_SETREG_RX(CC2400_MDMTST0,0x131E);
        FASTSPI_SETREG_RX(CC2400_MDMTST1, 0x001E);
        FASTSPI_SETREG_RX(CC2400_FSMTC,0x7AAF);

//      FASTSPI_SETREG_RX(CC2400_FREND,0x000F); // Livello di potenza d'uscita (default)
        FASTSPI_SETREG_RX(CC2400_IOCFG,0x7BE4);
        FASTSPI_SETREG_RX(CC2400_GRMDM,0x0DE8);
        FASTSPI_SETREG_RX(CC2400_GRDEC,0x0131);
//      FASTSPI_SETREG_RX(CC2400_INT,0x0020); // 32 byte threshold
        FASTSPI_SETREG_RX(CC2400_LMTST,0x2922);
//      FASTSPI_SETREG(CC2400_FSCTRL,0x0000); // lock threshold=64

        FASTSPI_STROBE_RX(CC2400_SXOSCON);         /*attiva oscillatore*/
        halRfWaitForCrystalOscillator_RX();
    /*polling, aspetta che l'oscillatorea sia stabile typ 1.13 mS.
    Si può ridurre a 15uS usando MANAND pg 63*/

        FASTSPI_STROBE_RX(CC2400_SFSON);
        RfWait_FS_LOCK_RX();

        FASTSPI_STROBE_RX(CC2400_SRXON);

        status = RfWait_SYNC_RECEIVED();
        // FASTSPI_RX_ADDR(CC2400_FIFOREG_RX);
        SPI_ENABLE_RX();
        FASTSPI_RX_ADDR(CC2400_FIFOREG_RX);

        U0RXBUF;
        halWait(5*500);
return status;
}


// send telecomando
void send_Telecomando(unsigned char TCC_0, unsigned char TCC_1)
{
    SPI_ENABLE_PAY();      //attivo SPI
    FASTSPI_TX1(TCC_0);
    SPI_DISABLE_PAY();
    SPI_ENABLE_PAY();
    FASTSPI_TX1(TCC_1);
    SPI_DISABLE_PAY();

  }


  /* invio dati a terra */

  void RX_to_TX_send_ACK(int dato_da_mandare)
{
int i;
```

```
      SPI_DISABLE_RX(); //ci dovrebbe essere già prima
      FASTSPI_SETREG_TX(CC2400_MAIN, 0x0000);
      FASTSPI_SETREG_TX(CC2400_MAIN, 0x8000);
      FASTSPI_SETREG_TX(CC2400_FSDIV,0x0988);  // frequenza: 2440MHz
      FASTSPI_SETREG_TX(CC2400_MANAND,0x7FFF); // override
#ifdef NOMOD
      FASTSPI_SETREG_TX(CC2400_MDMCTRL,0x0000);
      FASTSPI_SETREG_TX(CC2400_MDMTST0,0x134b);
      FASTSPI_SETREG_TX(CC2400_MDMTST1, 0x001E);
      FASTSPI_SETREG_TX(CC2400_FSMTC,0x7AAF);
      FASTSPI_SETREG_TX(CC2400_FREND,0x000F);
      // Livello di potenza d'uscita (default)
      FASTSPI_SETREG_TX(CC2400_IOCFG,0x07BF0); // GIO1 =1 GIO6=0
      FASTSPI_SETREG_TX(CC2400_GRMDM,0x1f0);
      FASTSPI_SETREG_TX(CC2400_GRDEC,0x0131);
      FASTSPI_SETREG_TX(CC2400_INT,0x0018);
      FASTSPI_SETREG_TX(CC2400_LMTST,0x2B22);
 #else
       FASTSPI_SETREG_TX(CC2400_MDMCTRL,0x0020);
      FASTSPI_SETREG_TX(CC2400_MDMTST0,0x131E);
      FASTSPI_SETREG_TX(CC2400_MDMTST1, 0x001E);
      FASTSPI_SETREG_TX(CC2400_FSMTC,0x7AAF);
      FASTSPI_SETREG_TX(CC2400_FREND,0x000B);
      // Livello di potenza d'uscita (default)
      FASTSPI_SETREG_TX(CC2400_IOCFG,0x07BF0); // GIO1 =1 GIO6=0
      FASTSPI_SETREG_TX(CC2400_GRMDM,0x0E79);
      FASTSPI_SETREG_TX(CC2400_GRDEC,0x0131);
    //  FASTSPI_SETREG_TX(CC2400_INT,0x0018);
      FASTSPI_SETREG_TX(CC2400_LMTST,0x2B22);
 #endif

      FASTSPI_STROBE_TX(CC2400_SXOSCON);         /*attiva oscillatore*/
      halRfWaitForCrystalOscillator_TX();
      /*polling, aspetta che l'oscillatorea sia strabile
      typ 1.13 mS. Si può ridurre a 15uS usando MANAND pg 63*/
      FASTSPI_STROBE_TX(CC2400_SFSON);
      RfWait_FS_LOCK_TX();
      switch (dato_da_mandare)
      {
        case 0:
          //telemetria di housekeeping
          fcs=APRS_frame_send(telemetria, Num_of_elements,tlc_copy);
        break;

        case 1:
          //telemetria estesa
          for (i = 0; i < Num_of_elements; i++)
          {
            telemetria_estesa1[i] = telemetria_min[i];
            telemetria_estesa1[i+Num_of_elements] = telemetria_max[i];
```

```
      telemetria_estesa2[i] = telemetria_media_invio[i];
      telemetria_estesa2[i+Num_of_elements] = telemetria_devstd_invio[i];
    }
    TC[10] = 0x40;
    fcs=APRS_frame_send(telemetria_estesa1, 2*Num_of_elements,&TC[5]);

    //attesa di 3 sec
    for (i = 0; i < 10; i++)
    {
      halWait(32000);
    }
    TC[10] = 0x41;
    fcs=APRS_frame_send(telemetria_estesa2, 2*Num_of_elements,&TC[5]);
  break;

  case 2:
    //telemetria di housekeeping + copia telecomando
    TC[10] = 0x39;
    fcs=APRS_frame_send(telemetria, Num_of_elements,&TC[5]);
    break;

  case '0':
  case '1':
  case '2':
  case '3':
  case '4':
  case '5':
  case '6':
  case '7':
  case '8':

    TC[10] = (char)dato_da_mandare;
    if (block[0] == 1)
    {
      fcs=APRS_frame_send(block, 132,&TC[5]);

    }
    else
    {
      fcs=APRS_frame_send(block, 1,&TC[5]);

    }

    break;

   default:
    // telemetria di housekeeping + copia telecomando
    TC[10] = 0x39;
    fcs=APRS_frame_send(telemetria, Num_of_elements,&TC[5]);

}
```

```
      FASTSPI_STROBE_TX(CC2400_SXOSCOFF);      /*attiva oscillatore*/
      FASTSPI_SETREG_TX(CC2400_IOCFG,0x7BE4); // GIO1=0 GIO6=0
      FASTSPI_SETREG_TX(CC2400_MAIN, 0x0000);
      FASTSPI_SETREG_TX(CC2400_MAIN, 0x8000); // forzo un reset

  }




/*Questa funzione riceve e trasmette direttamente in APRS SPI al TX la FOTO*/
/*10.000/8= 1250 8_bit-spi/s devo trasmettere ogni 800uS */

int Xmodem_receive_and_send(char n_blocco)
/*posso mettere un vettore al posto di addr*/
{
    char /*data,*/ crc;
    int n, i;//attemps

    n = 0;
    i = 0;

         do{
            //SPI_ENABLE_PAY();
             data=read_spi_byte();
            //SPI_DISABLE_PAY();
           }while(data==0);

       while (data != SPI_EOT) {
        if (data != SPI_SOH) {
            /* Unexpected data, bail out */
            return -1;
        }
        else
        {
        block[0] = data;

        }
        crc = 0;

        read_spi_blocking(&(block[1]), 131);
        for (i = 0; i < 128; i++) {
            crc += block[i + 3];
        }
        write_spi_byte(SPI_ACK);
        RX_to_TX_send_ACK(n_blocco);
        // si mette una lunghezza di 129
        n += 128;
```

```
        //Se si vuole trasmette anche il CRC di Xmodem per fare una verifica in piu

        halWait(32000);
        halWait(32000);

        data=read_spi_byte();
    }

    block[0] = SPI_EOT;
    RX_to_TX_send_ACK(n_blocco);

    return n;
}




//Questa funzione serve per inizializzare il convertitore A/D a 12 bit.
//Quindi attiva A/D riferimento interno, da spegnere dopo perchè consuma.
//disabilitare poi nel main quando finito ref,A/D . No ogni volta perchè ci vuole tempo.



void ADC12_ON(void)
// Riverificare tempo campionamento nel passaggio indirizzi
{
  ADC12CTL0 = SHT0_3 | REFON | ADC12ON| MSC;
  // Set sampling time 8 cycles @ 4MHz = 2uS > 1,55uS necessari.
  halWait(10000);
  // 10000*2uS=20mS  occorrono 17 mS per stabilizzare il riferimento
  ADC12CTL1 = SHS_0 | ADC12SSEL_3 | CONSEQ_0;
  // sample da ADC12SC, clock da SMCLK, single conversion,single channel
}




//Questa funzione acquisisce 64 campioni dello stesso valore
//del canale specificato, li accumula e fa il valor medio
//si acquisiscono anche i canali interni della temperatura e
//della tensione di alimentazione.

void ADC12_CONV(char INCH_X)
{
  int i, k, running;
  INT32 elem_telemetria = 0;

  ADC12MCTL0 = SREF_1 + INCH_X ;       // INCH_X indica il canale selezionato
```

```
                                        // interno prima della conversione.
  for (i=0; i< Num_di_campioni; i++) {
   ADC12CTL0 |= ENC + ADC12SC;              // Start acquisition
   ADC12CTL0 &= ~ADC12SC;                   // Sampling closed, start conversion
  for (k=0, running = 1; (k<100) & (running); k++)
      running = ADC12CTL1 & ADC12BUSY;

    elem_telemetria +=ADC12MEM0;
    ADC12CTL0 &= ~ENC;
  }
  telemetria[j] = (elem_telemetria >>4)/Num_di_campioni;
  // 6 di media + 4 di troncamento
  j++;
}



// Questa funzione commuta i mux e assembla il vettore di telemetria TMH
// tempo di una singola acquisizione = 367,2uS
// tempo per 78 canali = 28,6 mS


void TMH (void)
{
  unsigned char add_mux[16]={0,128,64,192,32,160,96,224,16,144,80,208,48,176,112,240};
  unsigned char add;

  ADC12_ON();

  // acquisizione POWER SUPPLY
  add=0;
  j=0;
  P3OUT|=BM(5);                // put 1 P3.5 abilito muxBout PSu
  P3OUT &=255-BM(6)-BM(7);
  P1OUT=add_mux[0];
  for (k = 0; k < 5; k++)     //indirizzi 0 4
  {
    halWait(attesa_PSU);              // T-wait=60us=(2uS*30)
    //Ttot=50,2uS   Ton-mux= 150 nS max + tRC=5T=5*100*100n=50uS
    ADC12_CONV(INCH_0);         // muxBout PSu
    add++;
    P1OUT=add_mux[add];
    // tempo di campionamento e conversione A/D=4,8uS  ;
    // t-SAMPLE=1,3uS+t-SYNC=0,25uS + t-CONV=13*1/CLK=3,25uS
  }

  add=8;
  P1OUT=add_mux[add];           // salta indirizzi 5 6 7
  for (k = 0; k < 6; k++)      // indirizzi 8 - 13
  {
```

```
  halWait(attesa_PSU);
  ADC12_CONV(INCH_0);
  add++;
  P1OUT=add_mux[add];
}

add=0;
P1OUT=add_mux[add];
P3OUT|=BM(7);                // salta indirizzo 14 15
for (k = 0; k <16; k++)      // indirizzi 16 - 31
{
  halWait(attesa_PSU);
  ADC12_CONV(INCH_0);
  add++;
  P1OUT=add_mux[add];
}

P3OUT&=~BM(7);
P3OUT|=BM(5);                // put 1 P3.5 abilito muxBout PSu
P3OUT|=BM(6);
add=0;
P1OUT=add_mux[add];
for (k = 0; k < 7; k++)      // indirizzi 32 38
{
  halWait(attesa_PSU);
  ADC12_CONV(INCH_0);
  add++;
  P1OUT=add_mux[add];
}

// acquisizione POWER SWITCH

P2OUT=8;                      // abilito muxDout PSw

for (k=0; k<8; k++)
{
  halWait(attesa_PSU);
  ADC12_CONV(INCH_1);           // muxDout PSw
  P2OUT++;
}
P2OUT=0;                     // diabilito muxDout PSw

// acquisizione ProcB

j=61;
ADC12_CONV(INCH_10);         //temperatura scheda ProcB
ADC12_CONV(INCH_5);          //temperatura scheda TxRx
ADC12_CONV(INCH_7);          //temperatura scheda Payload
ADC12_CONV(INCH_3);          // corrente Pay 3.3V
ADC12_CONV(INCH_4);          // corrente Pay 12V
ADC12_CONV(INCH_2);          // corrente ProcB
```

```
  j=68;
  ADC12_CONV(INCH_6);         // corrente Motore
  j=67;
  ADC12CTL0 = SHT0_1 + MSC + REF2_5V + REFON + ADC12ON;
   //cambio riferimento a 2,5 v per acquisire tensione ProcB
  halWait(attesa_PSU);
  // 10000*2uS=20mS  occorrono 17 mS per stabilizzare il
  //riferimento interno prima della conversione.
  ADC12_CONV(INCH_11);
  //tensione scheda ProcB
  ADC12_OFF();

}




//Questa funzione serve per disattivare convertitore A/D e
//riferimento interno da spegnere perchè consumano.
//si disattivano alla fine dell' acquisizione di tutti i canali.

void ADC12_OFF(void)
{
  ADC12CTL0 =0x0000; // Reference off, ADC12 off, ADC12 disable, Stop Conversion .
  ADC12IE = 0x0000;     // Disable interrupt
  _DINT();              // Disable interrupts
}




// rotate_motor

#pragma vector=PORT1_VECTOR
__interrupt void INT_Port1(void){


 if((N_giri !=0) && (P1IN!=0x07))
   {
     _DINT();
    N_giri--;
    if(spin==1)                                 // Verso antiorario
           {
            P1IFG =0x00;
            // Disabilito flag interrupt porta1
            if ((P1IN |0xF8)==0xFD){
                      //60 gradi Hall3,Hall2,Hall1=>101
               P1IES=BIT1+BIT2 ;
               //il ciclo dopo HALL 3 ha transizione 1->0 è attiva P1.2 BIT2=1
               P1IFG =0x00;
```

```
   TBCCTL5 = OUTMOD_0;
   TBCCTL1 = OUTMOD_7;
   //due cicli prima aveva attivo 1->0 in P1.1 BIT1=1 immutato


   //il ciclo prima aveva attivo 0->1 in P1.0 BIT0=0 immutato
   }
else if ((P1IN |0xF8)==0xF9){
     //120 gradi Hall3,Hall2,Hall1=>001
   P1IES=BIT2;
   //il ciclo dopo HALL 2 ha transizione 0->1 è attiva P1.1 BIT1=0
   P1IFG =0x00;
   TBCCTL4 = OUTMOD_0;
   //due cicli prima aveva attivo 0->1 in P1.0 BIT0=0 immutato
   TBCCTL6 = OUTMOD_7;
   //il ciclo prima aveva attivo 1->0 in P1.2 BIT2=1 immutato
   }
else if ((P1IN |0xF8)==0xFB){
      //180 gradi Hall3,Hall2,Hall1=>011
   P1IES=BIT0+BIT2;
   //il ciclo dopo HALL 1 ha transizione 1->0 è attiva P1.0 BIT0=1
   P1IFG =0x00;
   TBCCTL1 = OUTMOD_0;
   //due cicli prima aveva attivo 1->0 in P1.2 BIT2=1 immutato
   TBCCTL3 = OUTMOD_7;
   //il ciclo prima aveva attivo 0->1 in P1.1 BIT1=0 immuta
   }
else if ((P1IN |0xF8)==0xFA){
     //240 gradi Hall3,Hall2,Hall1=>010
   P1IES=BIT0;
   //il ciclo dopo HALL 3 ha transizione 0->1 è attiva P1.2 BIT2=0
   P1IFG =0x00;
   TBCCTL6 = OUTMOD_0;
   //due cicli prima aveva attivo 0->1 in P1.1 BIT1=0 immutato
   TBCCTL2 = OUTMOD_7;
   //il ciclo prima aveva attivo 1->0 in P1.0 BIT0=1 immutato
    }
else if ((P1IN |0xF8)==0xFE){
     //300 gradi Hall3,Hall2,Hall1=>110
   P1IES=BIT0+BIT1;
   //il ciclo dopo HALL 2 ha transizione 1->0 è attiva P1.1 BIT1=1
   P1IFG =0x00;
   TBCCTL3 = OUTMOD_0;
   //due cicli prima aveva attivo 1->0 in P1.0 BIT0=1 immutato
   TBCCTL5 = OUTMOD_7;
   //il ciclo prima aveva attivo 0->1 in P1.2 BIT2=0 immutato
   }
else if ((P1IN |0xF8)==0xFC){
     //360 gradi Hall3,Hall2,Hall1=>100
   P1IES=BIT1;
   //il ciclo dopo HALL 1 ha transizione 0->1 è attiva P1.0 BIT0=0
```

```
            P1IFG =0x00;
            TBCCTL2 = OUTMOD_0;
            //due cicli prima aveva attivo 0->1 in P1.2 BIT2=0 immutato
            TBCCTL4 = OUTMOD_7;
            //il ciclo prima aveva attivo 1->0 in P1.1 BIT1=1 immutato
            }
   }
 else if(spin==0)                                  // Verso orario
   {
         P1IFG =0x00;
         // Disabilito flag interrupt porta1
         if ((P1IN |0xF8)==0xFC){
                  //360 gradi Hall3,Hall2,Hall1=>100
            P1IES=BIT0;
            //il ciclo dopo HALL 2 ha transizione 0->1 è attiva P1.1 BIT1=0
            P1IFG =0x00;
            TBCCTL2 = OUTMOD_0;
            TBCCTL6 = OUTMOD_7;
            }

         else if ((P1IN |0xF8)==0xFE){
              //300 gradi Hall3,Hall2,Hall1=>110
            P1IES=BIT0+BIT2;
            //il ciclo dopo HALL 3 ha transizione 1->0 è attiva P1.2 BIT2=1
            P1IFG =0x00;
            TBCCTL3 = OUTMOD_0;
            TBCCTL1 = OUTMOD_7;
            }

         else if ((P1IN |0xF8)==0xFA){
              //240 gradi Hall3,Hall2,Hall1=>010
            P1IES=BIT2;
            //il ciclo dopo HALL 1 ha transizione 0->1 è attiva P1.0 BIT0=0
            P1IFG =0x00;
            TBCCTL6 = OUTMOD_0;
            TBCCTL4 = OUTMOD_7;
            }
         else if ((P1IN |0xF8)==0xFB){
              //180 gradi Hall3,Hall2,Hall1=>011
            P1IES=BIT1+BIT2;
             //il ciclo dopo HALL 2 ha transizione 1->0 è attiva P1.1 BIT1=1
            P1IFG =0x00;
            TBCCTL1 = OUTMOD_0;
            TBCCTL5 = OUTMOD_7;
            }
         else if ((P1IN |0xF8)==0xF9){
              //120 gradi Hall3,Hall2,Hall1=>001
            P1IES=BIT1;
            //il ciclo dopo HALL 3 ha transizione 0->1 è attiva P1.2 BIT2=0
            P1IFG =0x00;
            TBCCTL4 = OUTMOD_0;
```

```
               TBCCTL2 = OUTMOD_7;
               }
            else if ((P1IN |0xF8)==0xFD){
                 //60 gradi Hall3,Hall2,Hall1=>101
               P1IES=BIT0+BIT1;
               //il ciclo dopo HALL 1 ha transizione 1->0 è attiva P1.0 BIT0=1
               P1IFG =0x00;
               TBCCTL5 = OUTMOD_0;
               TBCCTL3 = OUTMOD_7;
               }
      }

    _EINT();
   }//end if giri
else
   {
      P1IFG =0x00;
      POWER_OFF_MOTORE();                    // Toglie alimentazione al motore
      _DINT();
        // Disabilito interrupt
      TBCTL =  MC_0;
       // Disabilitare il Timer B (consuma potenza)
      TBCCTL1 = OUTMOD_0;
      TBCCTL2 = OUTMOD_0;
      TBCCTL3 = OUTMOD_0;
      TBCCTL4 = OUTMOD_0 + OUT;
      TBCCTL5 = OUTMOD_0 + OUT;
      TBCCTL6 = OUTMOD_0 + OUT;

      N_giri=0;
      _BIC_SR_IRQ(LPM3_bits);               // Clear LPM3, Exit LPM3 on return
   }
}




// strategia di carica batterie

void carica_batteria(void)
{

char b;

batt_ok=FALSE;

  while (batt_ok==FALSE && batt_try < 6 )
     {
     b = (char)((P2IN&0xF0));
        switch(b)
```

```
            {
             case 0x10: {  // B1 -> B4
                           verify_batt(1, 4, 0x70, Li_P,  telemetria[27],
                           telemetria[16] );
                           batt_try++;
                         } break;

             case 0x30: {  // B2 -> B5
                           verify_batt(2, 5, 0x90 , Li_P,  telemetria[28],
                           telemetria[17] );
                           batt_try++;
                         } break;

             case 0x50: {  // B3  -> B6
                           verify_batt(3, 6, 0xB0 , Ni_Cd,  telemetria[29],
                           telemetria[18] );
                           batt_try++;
                         } break;

             case 0x70: {  //B4    -> B2
                           verify_batt(4, 2, 0x30 , Li_P,  telemetria[30],
                           telemetria[24] );
                           batt_try++;
                         } break;

             case 0x90: {  //B5  -> B3
                           verify_batt(5, 3, 0x50 , Li_P,  telemetria[31],
                           telemetria[25] );
                           batt_try++;
                         } break;

             case 0xB0: {  //B6   -> B1
                           verify_batt(6, 1, 0x70, Ni_Cd,  telemetria[32],
                           telemetria[26] );
                           batt_try++;
                         } break;


               default:  { //Error ProcB don't change battery
                           sel_next_batt(6, 6, 0xB0);
                           batt_ok=TRUE;
                           break;
                         }
              }

       }// end while


if (Old_batt == (P2IN&0xF0))
 {
  if (timerL == 255)
```

```
  {
   timerH++;
   timerL =0;
  }
  else
    timerL++;
 }
else
{
timerH=0;
timerL=0;
Old_batt = (P2IN&0xF0);
}


}// end carica_batteria







// calcolo_fcs tramite loop_table

unsigned short APRSfcs_Bit_stuffing_and_send(unsigned short fcs,
unsigned char *byte, int len)
{


int i;
for(i=0; i < len; i++)
{
  FASTSPI_TX(*byte);
  fcs = (fcs >> 8) ^ fcstab[(fcs ^ *byte++) & 0xff]; //calcola il CRC

}

return (fcs);
}







// Questa funzione invia un frame completo in APRS(AX.25) dandogli un
//vettore unico
// MAX LEN 256 BYTE ALTRIMENTI SI INVALIDA IL FRAME

unsigned short APRS_frame_send(unsigned char *byte,int len,unsigned char *tlc_copy)
```

```
{
unsigned char data_CRC1, data_CRC2;
unsigned short fcs= 0xffff;
unsigned short eot;


SPI_ENABLE_TX();
FASTSPI_TX_ADDR(CC2400_FIFOREG_TX);
fcs = APRSfcs_Bit_stuffing_and_send(fcs,PicPot_adress,8);
SPI_DISABLE_TX();
FASTSPI_STROBE_TX(CC2400_STXON);
SPI_ENABLE_TX();
FASTSPI_TX_ADDR(CC2400_FIFOREG_TX);
fcs = APRSfcs_Bit_stuffing_and_send(fcs,(PicPot_adress+8),8);
fcs = APRSfcs_Bit_stuffing_and_send(fcs, tlc_copy, 6);


fcs = APRSfcs_Bit_stuffing_and_send(fcs, byte, len);

fcs ^= 0xffff;
data_CRC1=(fcs&0xff);//fcs_lo
data_CRC2=(fcs >> 8)&0xff;//fcs_hi
fcs = APRSfcs_Bit_stuffing_and_send(fcs, &data_CRC1, 1);
fcs = APRSfcs_Bit_stuffing_and_send(fcs, &data_CRC2, 1);
SPI_DISABLE_TX();
do{
  FASTSPI_GETREG(CC2400_FSMSTATE,eot);
}while(eot!=15);
return (fcs);
}
```

Bibliografia

[1] Leonardo Reyneri, *Documento di specifiche dei sottosistemi elettronici*, Torino, 2006.

[2] Claudio Zamberland, *Progetto atmocube: analisi preliminare della geometria orbitale*, url: http://www.univ.trieste.it/ carrato/didatt/cubesat /docts/tesina orbita.pdf

[3] Isaac Nason, Michelle Credon, Nick Johansen, *CUBESAT, P-Pod Deployer Requirements* , San Louis, 2002.

[4] url: http://www.sec.noaa.gov/

[5] url: http://www.esa.int/esaCP/index.html.

[6] url: http://www.asi.it/index.htm.

[7] url: http://www.qsl.net/ve7ldh/thesis.html.

[8] Jim Lepkowski, *Motor Control Sensor Feedback Circuit*, Microchip Technology inc., 2003.

[9] Padmaraja Yedamale, *Brushless DC (BLDC) Motors Fundamentals*, Microchip Technology inc., 2003.

[10] Ward Brown, *DC Motor Control Made Easy/*, Microchip Technology inc., 2002.

[11] Jamie Dunn, *Determining MOSFET Driver Needs for Motor Drive Application*, Microchip Technology inc., 2003.

[12] url: http://www.ti.com.

[13] S. Franco, *di Circuiti Analogici con Amplificatori Operazionali*, Milano, Hoepli, 1992.

[14] W. S. Means, M. A. Bodie, *The book of SAX*, No Starch Press, 2002.

[15] B. W. Kernighan, D. M. Ritchie, *The C programming language*, New Delhi:, Prentice Hall, 2004.

[16] TAPR, *APRS protocol specification version 1.0.1*, 200.

[17] Vincenzo Pozzolo, *Appunti relativi al corso di elettronica*, Politeko, Torino