Sommario

Negli ultimi anni, nel campo della progettazione di satelliti di ridotte dimensioni, si sta andando nella realizzazione di uno standard CubeSat, con l'obiettivo di fornire la metodologia di base per la realizzazione e lo sviluppo di picosatelliti. In tutto il mondo, diversi atenei, tra i quali il Politecnico di Torino, hanno adottato o riadattato tale standard per lo sviluppo di un proprio satellite di piccole dimensioni.

Il primo progetto intrapreso dal Politecnico, denominato PiCPoT (Piccolo Cubo del Politecnico di Torino), prevedeva la realizzazione ed il lancio in orbita di un satellite di forma cubica di 13 cm di lato, i cui elementi costituenti fossero progettati ad-hoc, con lo scopo di acquisire e trasmettere dati, scattare fotografie e testare il funzionamento in ambiente spaziale di componenti COTS.

Il secondo progetto, AraMiS (Architettura Modulare per Satelliti), attualmente in via di sviluppo presso il Politecnico di Torino, costituisce la prosecuzione ed il miglioramento di PiCPoT.

L' obiettivo è quello di realizzare un satellite modulare, di forma cubica (ma configurabile anche in altre forme), nel quale ogni scheda elettronica sia allo stesso tempo progettabile in modo indipendente dalle altre (consentendo una riduzione di costi e tempi di progettazione), esportabile in altri progetti (o utilizzabile in diverse missioni), specializzata nell'esecuzione dei propri compiti, compatibile con gli altri componenti, sostituibile e modificabile, senza che questo vada ad influire negativamente sul funzionamento complessivo del satellite.

I due tipi principali di moduli standard (detti appunto tiles) che costituiscono AraMiS sono:

- ***Power Management Tile**: ha la funzione di generare l'energia elettrica necessaria ad alimentare i circuiti e a caricare le batterie, convertendo l'energia solare grazie ai pannelli fotovoltaici presenti sulle facce esterne di AraMiS. Questi moduli costituiscono cinque delle sei facce del cubo di base (AraMiS è configurabile anche combinando otto cubi assieme in un'unica struttura, oppure sotto forma di prisma esagonale). Inoltre forniscono l'assetto al satellite, mediante un controllo di assetto attivo costituito da una ruota d'inerzia, un solenoide, un sensore di campo magnetico, un giroscopio e un motore brushless e montato sull'interno delle facce del cubo.*
- ***Telecommunication Tile**: ha la funzione di mettere in comunicazione il satellite con la stazione di Terra tramite due canali, alle frequenze di 437 MHz e 2.4 GHz, ed occupa una delle sei facce del cubo di base. Inoltre, tale modulo ha il compito di interpretare i comandi di assetto ricevuti da terra ed inviare ai singoli moduli di Power Management i comandi specifici per l'attuazione di tale assetto.*

La tesi si inquadra in un contesto del genere abbracciando due specifici moduli che a loro volta, rappresentano due sottosistemi utili nell' acquisizione dell' immagine e relativa memorizzazione della stessa e cioè On Board Computer e Other Attitude Sensor. In realtà, l' obiettivo prefissato non può essere solo la realizzazione di un semplice sun sensor, ma in una visione del genere è quello di creare due blocchi che servano dapprima a calcolare l' assetto, ma nel contempo prelevare un' immagine (in verità più di una) e renderla utile per i vari scopi che una foto può garantire. Inoltre, anche se si è rimasti fedeli alla realizzazione del sensore di sole, non è detto che con le adeguate modifiche possa essere utilizzato come sensore di stelle (sicuramente più preciso da quello target della tesi).

Finita la premessa, brevemente nel concreto il lavoro ha seguito i seguenti passi: si è partiti dall' esaminare la realtà modulare di Aramis, si è capito dove e come si sarebbe inserito il blocco da realizzare tenendo presente che le ipotesi da cui partire sarebbero state un processore abbastanza veloce da gestire i dati, una memoria di dimensione tali da poter accogliere e immagazzinare i dati e cosa più importante, in quel momento il vero e proprio sensore d' immagine. Una volta ben chiaro il funzionamento del tutto, si è passati alla scelta del sensore con relative caratteristiche come formato colore, potenza, data rate e frequenza solo per citarne alcune e quindi allo stesso modo per il processore. Quest' ultimo non solo doveva rispondere alle esigenze dettate dall' image sensor, ma nello stesso tempo doveva rispettare il principio cardine di modularità del satellite e quindi cercare di essere maggiormente flessibile nel momento in cui si sarebbe dovuto interfacciare con altri sottosistemi "estranei" al mio. Definiti i vari componenti, si è passati infatti alla realizzazione hardware e software con i tool messi a disposizione dalla Visual Paradigm International e da Mentor Graphics.

Importante per il sottosistema per la determinazione dell'assetto (OtherAttitude Sensor) Subsystem, codice di progetto 1B23) è la rilevazione dell' immagine che avverrà tramite l' active imaging pixel array del sensore e trasmessa tramite PPI (Parallel Peripheral Interface) all' Imager Processor (On Board Computer codice di progetto 1B42), che a sua volta gestirà questi dati andandoli ad immagazzinare nella Sdram per poi utilizzarli per calcolare l' assetto del satellite attraverso un algoritmo contenuto nella Flash. Gli stessi dati potranno servire all' OBC anche per altre applicazioni non necessariamente connesse alla determinazione dell' angolo del sole come può essere per un sensore di stelle, ma in generale avvalendoci di ciò che viene richiesto secondo il preciso caso d' uso possiamo avere un' acquisizione immediata o "ritardata" (in quanto il sistema non si trova nella situazione ideale per scattare una foto) oppure ancora diverse immagini.

Visto che si sono citati i casi d' uso, anche in virtù del concetto di modularità che sta alla base del progetto AraMiS, è stato utilizzato il linguaggio di descrizione e modellizzazione visuale UML (Unified Modeling Language) per sviluppare il progetto in esame e tutti gli altri progetti costituenti i diversi sottoblocchi. Esso si basa su uno standard definito a metà degli anni '90, ma in continua evoluzione, e permette di documentare le specifiche del sistema mediante i Diagrammi dei Casi d'Uso, descrivere in

forma visiva e completa gli elementi che ne fanno parte con i Diagrammi delle Classi, e illustrare il flusso di eventi tramite cui è possibile concretizzarne le specifiche, grazie ai Diagrammi di Sequenza.

D' altra parte, il progetto hardware della determinazione dell' assetto del satellite si è basato sulla struttura di un preesistente sistema analogo, quello del Payload nel satellite PicPot, i cui elementi sono stati in piccola parte riutilizzati, ridimensionati e prevalentemente sostituiti, a seconda dei casi, in modo da soddisfare le attuali specifiche, durante le diverse fasi di progettazione.

Il sistema può essere scomposto in due blocchi; il blocco principale costituito dal processore con la memoria SDRAM che include tutta la parte di sensoristica per monitorare sia tensioni che correnti, l' altro invece è composto dal singolo sensore con current sensor. In linea di massima, il CMOS sensor acquisisce i frames con le sue tempistiche pre-programmate con il protocollo I2C e le invia all' image processor che si occupa di instradarle fino alla memoria; in un secondo momento, i dati possono essere elaborati o semplicemente trasferiti sulla Flash integrata per essere disponibili per l' utilizzo desiderato.

Saranno lasciate come oggetto per ulteriori necessari sviluppi, le procedure di collaudo sul circuito prototipale in quanto ci si è fermati fino alla realizzazione del PCB che accorpa i due blocchi, senza procedere alle adeguate misurazioni. D' altro canto, per quanto concerne il software, si è sviluppata una semplice routine per l' acquisizione ed elaborazione dell' immagine con il compilatore messo a disposizione dall' Analog Devices, CrossCore Embedded Studio.

Introduzione

Questa tesi è stata sviluppata basandosi su quelle che sarebbero state le condizioni di funzionamento nello spazio mirando a rappresentare quanto più possibile quella stessa realtà e cercando nello stesso tempo, di far fronte alle diverse problematiche che si sarebbero potute avere, Inoltre, è utile precisare che per quanto riguarda questo lavoro, non è che l' incipit di un qualcosa che verrà approfondito e sviluppato prima di essere efficiente per poi essere utilizzato in un sistema modulare satellitare. Perciò, il tutto si traduce in un emulazione dell' ambiente con componenti che riporteranno delle caratteristiche "quasi" realistiche, ma che nel complesso non lo sono. Un esempio potrebbe essere il sensore d' immagine utilizzato, è un Aptina della serie MT9V034, certamente non adatto a quelle situazioni, ma molto utile per sostituire un altro più complesso, più costoso e *radiation tolerant* come lo STAR250 della Cypress Semiconductor: a tal proposito, come sarà discusso nella sezione relativa alla scelta del componente, ci si avvarrà di certi criteri principali come consumi di potenza, dimensione, *rate* e periferiche disponibili che rappresentano alcune delle criticità nello spazio.

In secondo luogo, è utile soffermarci sulla visione modulare dell' insieme, rimanendo continuamente legati ad un certo grado di flessibilità sotto la quale non poter scendere per rendere lo stesso oggetto della tesi non solo utile ai fini della stessa, ma in un certo qual modo duttile anche all' utilizzo da parte di altri sottosistemi come potrebbe essere una comunicazione seriale con un altro processore o il pilotaggio di una ruota d' inerzia. Questo vincolo imprescindibile sarà ripreso più volte proprio per richiamare la causa di determinate scelte e in un' ottica più ampia, rispecchia la semplicità progettuale che mira non più alla sostituzione di un intero sistema, ma la possibilità di poter rimpiazzare e modificare solo quella specifica parte in tempi brevissimi dettati dalla efficiente modularità. Cardine principale di quanto detto, è l' utilizzo dell' UML e in particolare del software *Visual Paradigm*, che permette appunto di tradurre un progetto in maniera chiara e semplicistica in modo da poter continuamente monitorare lo sviluppo di un team work e permettere revisioni su dedicate sezioni tenendo una visione generale del tutto.

Allo stesso modo, per quanto concerne la parte software si è utilizzato un CAD della Mentor Graphics, per mezzo del quale è stato progettato il PCB e sfruttando la libreria in comune, si sono dapprima creati e poi messi a disposizione i vari componenti utili ai fini della realizzazione della scheda stessa e in generale per chi ne necessitasse in successivi lavori.

La tesi è dunque stata realizzata non in una direzione rivolta al completamento come obiettivo principale, ma per sua stessa natura deve essere vista come un piccolo tassello di un enorme mosaico che prevede tra l' altro migliorie e modifiche a riguardo. Volendo tradurre quanto detto, sono state fornite le

linee guida basilari e dimostrative di un progetto inserito in un quadro totalmente modulare che rimarca principi di semplicità e condivisione con strumenti indispensabili per l’ ambiente di lavoro.

Indice

Capitolo 1

Introduzione

1.1 Lo standard Cubesat

La tendenza dei componenti elettronici ad essere sempre più piccoli ha reso possibile la riduzione delle dimensioni dei satelliti e di conseguenza notevole riduzione dei costi sia per la produzione che per il lancio nello spazio. Ciò ha portato alla seguente classificazione in base al peso:

- Mini-satellite, (massa compresa tra 100 kg e 500 kg), ma nonostante la massa ridotta solitamente conserva la stessa tecnologia dei satelliti più grandi; ad esempio è alimentato con del carburante;
- Micro-satellite, (massa compresa tra 10 kg e 100 kg);
- Nano-satellite, (massa compresa tra 1 kg e 10 kg);
- Pico-satellite (massa compresa tra 100 g ed 1 kg);
- Femto-satellite (massa minore di 100 g);

In molti casi i satelliti più grandi, essendo molto costosi, sono progettati appositamente su commissione, tenendo conto delle loro dimensioni, del carico e del loro utilizzo; al contrario il minor costo dei piccoli satelliti ha permesso di realizzare delle architetture standardizzate, indipendenti dall'utilizzo finale. A tal proposito, CUBESAT é 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.

In questo ambito le università sono tra le protagoniste maggiormente attive, spinte soprattutto dalla consapevolezza che il forte valore didattico deve essere supportato da un progetto di natura altamente interdisciplinare, in cui é necessario avere conoscenze di tipo meccanico/aerospaziale ed anche elettronico/informatico.
Le caratteristiche principali dello standard CUBESAT sono:
. Forma cubica di dimensioni 10x10x10cm, con massa non superiore ad 1,33 kg;
. Compatibilità con il lanciatore *Poly-PicoSatellite Orbital Deployer (P-POD)*;

. Utilizzo di componenti elettronici COTS (Commercial Off The SHelf), ossia di componenti presenti nel mercato consumer ma più economici dei corrispondenti per applicazioni specifiche.
Dal 1999 ad oggi lo standard è stato progressivamente migliorato; al 2012 si stima ne siano stati lanciati con successo 75 satelliti. Inoltre nel 2004 era possibile lanciare un CUBESAT con prezzi relativamente bassi, tra i 50 ed i 60 mila euro.
L'idea di progettare un satellite è stata adottata da numerose Università italiane e straniere, fra le quali citiamo l'Universitat Wurzburg,(D), la Norwegian University of Science and Technology (NO), l'Aalborg University (DK) e l'Universita' della Sapienza(IT). Successivamente il Politecnico di Torino si `e impegnato a partecipare alla progettazione ideando PICPOT e collocandolo in un progetto più ampio, quello di una costellazione di satelliti universitari italiani, attualmente in via di definizione.

1.2 Il progetto PICPOT

Avendo ben chiaro il concetto di Cubesat, introduciamo il progetto PICPOT, PICosatellite del POlitecnico di Torino, ricordando che era nato nel Gennaio 2004 con l'obiettivo di costruire nell'arco di un anno uno picosatellite universitario, con una massa minore di 1 kg, a scopo educativo e di ricerca.

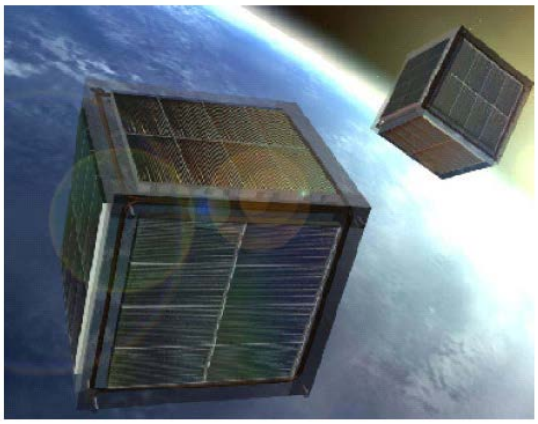


Figura 1.1 Prototipo di Cubesat ideato dalla CalPoly e dalla Stanford University

La figura 1.2 `e una fotografia della parte esterna del satellite, progettata e costruita dagli studenti del Dipartimento di Ingegneria Aeronautica e Spaziale del Politecnico di Torino.

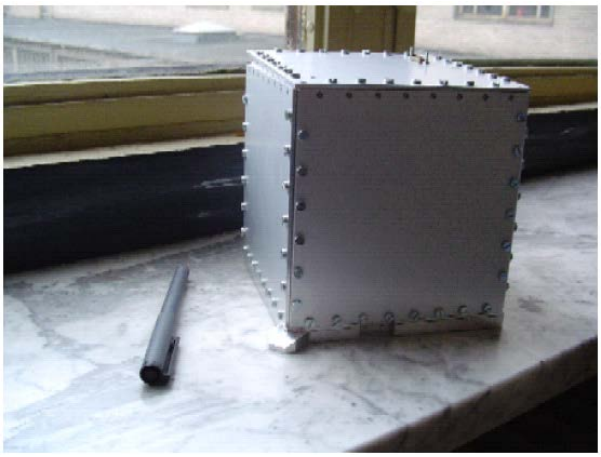


Figura 1.2 PICPOT-PICosatellite POlitecnico di Torino

Il progetto del satellite si era basato sui seguenti requisiti:

- forma cubica con lato di 13 cm
- massa inferiore ad 2 kg
- potenza non superiore a 1.5 W
- almeno 90 giorni di vita
- orbita LEO
- compatibile con il lanciatore POD

Inizialmente PICPOT aveva come obiettivi quelli di verificare il funzionamento di componenti COTS nello spazio, trasmettere dei dati alla stazione di Terra, scattare una o più fotografie e valutare il funzionamento del GPS in orbita LEO. In realtà, per motivi economici e di tempo, non è stato possibile dotare il satellite del GPS.

Nella figura 1.4 è rappresentata la struttura esterna definitiva di PICPOT: è un cubo dotato di sei facce quadrate ed ortogonali tra loro, di circa 13 cm di lato, ricoperto di pannelli solari (P1, P2, P3, P4, P5) e dotato esternamente di due antenne (2.4 GHz, 435 MHz), tre fotocamere(T1, T2, T3), due kill-switch (K1, K2) ed un connettore di test.

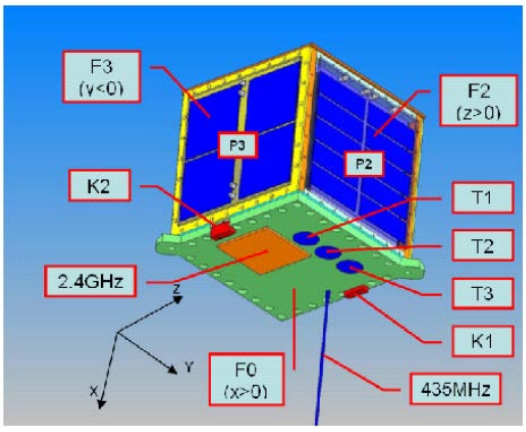


Figura 1.3 PICPOT-vista esterna

La parola chiave per capire l'idea alla base della struttura del satellite è stata ridondanza: cioè si era cercato infatti, dove possibile, di duplicare ogni sottosistema per evitare che il guasto di una singola parte compromettesse il funzionamento dell'intero satellite. Le celle solari, ad esempio, erano di due tipi differenti, come si vede nella figura 1.3, proprio per motivi di ridondanza. La comunicazione, inoltre, avveniva su due canali indipendenti e mutualmente esclusivi: quella associata alla frequenza di 435 MHz e gestita esclusivamente dal processore ProcA, mentre quella associata alla frequenza di 2.4 GHz era gestita esclusivamente dal ProcB. I due processori erano indipendenti e non comunicavano tra loro.
Le antenne erano un'antenna patch per la frequenza intorno ai 2.4 GHz progettata apposta per il satellite ed un'antenna a dipolo per la frequenza intorno ai 435 MHz, di tipo commerciale.
I kill-switch sono stati montati su richiesta del lanciatore per garantire l'isolamento elettrico del satellite al momento del lancio. La scheda Payload aveva il compito di gestire le tre telecamere e, in particolare, di scattare delle fotografie in seguito ad una richiesta da parte della stazione di Terra. L'uso delle fotocamere non era funzionale ad un rilevamento di tipo scientifico, ma aveva invece due obiettivi, uno di carattere didattico-educativo e l'altro di verifica del funzionamento di componenti COTS nello spazio, come il progetto dell'intero satellite richiede.
La parte elettronica del satellite era formata da una scheda PowerSwitch, da una scheda PowerSupply, da una scheda Tx/Rx e dai tre processori di bordo, ProcA, ProcB e Payload.
La scheda PowerSupply aveva il compito di mantenere cariche le batterie ricaricabili del satellite e di monitorare lo stato elettrico e termico dei pannelli, dei caricabatteria e delle batterie stesse.
La funzione principale della scheda PowerSwitch era quella di alimentare le altre schede generando le tensioni necessarie a partire dalle tensioni delle batterie. La scheda Tx/Rx aveva la funzione di trasmissione e ricezione di segnali tra il satellite PICPOT e la stazione di Terra ideata a tale proposito, alle frequenze di 435 MHz e 2.4 GHz nella banda dedicata alle comunicazioni satellitari amatoriali. I due processori, ProcA e ProcB, progettati ciascuno da un diverso studente, svolgevano funzioni simili alle due diverse frequenze, ma sono stati duplicati per motivi di ridondanza, come spiegato in precedenza. La figura 1.4 è una fotografia della parte interna di PICPOT che evidenzia la disposizione delle schede elettroniche e delle tre fotocamere.

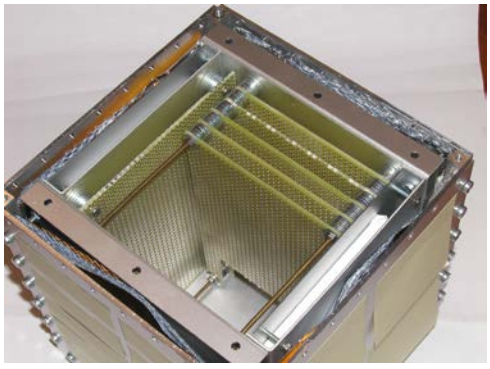


Figura 1.4 PICPOT-vista interna

1.3 Il progetto Aramis

AraMiS, acronimo di *ARchitettura Altamente Modulare* per *Infrastrutture Satellitari*, é l'evoluzione del progetto PicPoT ed è stato sviluppato a partire dal 2006. Le finalità di *AraMiS*, oltre a quelle di tipo didattico, sono molto ambiziose, infatti mira a diventare lo standard rivale a CUBESAT.
Le caratteristiche salienti di AraMiS in versione *cubo* sono:

- Forma cubica di dimensioni 16:5x16:5x16:5cm, con massa superiore a 5 kg.
- Potenza massima assorbita dai pannelli solari di circa 6 W.
- Orbita LEO (*Low Earth Orbit*).
- Tempo di vita stimato in 5 anni.
- Utilizzo di componenti COTS.
- Modularità a livello meccanico, elettronico e di testing.
- Progettazione unificata con linguaggio di modellizzazione UML.

Secondo la classificazione precedente si tratta di un nanosatellite che segue un'orbita LEO, ossia una distanza dalla Terra compresa tra i 600 e gli 800 km. Il principale concetto innovativo di AraMiS é la modularità. A differenza di CUBESAT si vuole creare uno standard in cui i sotto sistemi che lo compongono siano indipendenti tra loro, sia a livello fisico che in fase di sviluppo. In altre parole la struttura deve essere adattabile alle esigenze specifiche della missione.
Tale scelta porta ai seguenti vantaggi:

- I moduli standardizzati possono essere usati in più missioni riducendo il costo.
- E’ possibile progettare sottosistemi diversi in parallelo riducendo il tempo di progettazione.
- Per ogni missione é possibile riconfigurare il satellite in base agli obiettivi voluti.

Lo svantaggio maggiore risiede ovviamente nella difficoltà del progetto di creare uno standard funzionante ed efficiente. Il progetto AraMiS, al contrario, raccoglie tutti i precedenti sottosistemi dividendoli in sole due categorie dette Tiles, letteralmente mattonelle, figura 1.5. Queste mattonelle, che sono disposte sulle facce esterne del satellite avendo quindi anche una funzione strutturale, sono:

- Power Management e ACS Tile;
- On-Board Computer e Telecommunication Tile;

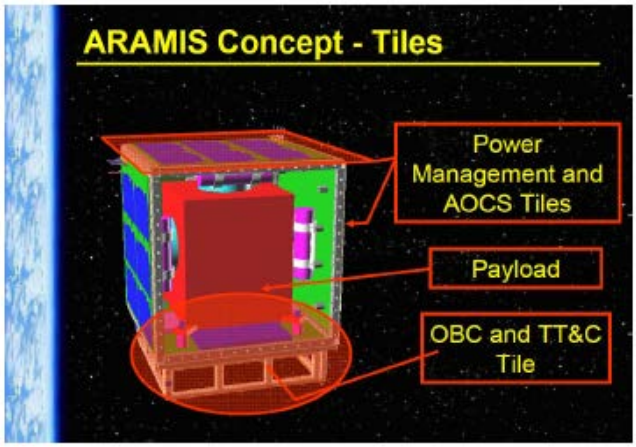


Figura 1.5 Visione di un satellite Aramis e delle relative tiles

Inoltre l'architettura prevede le seguenti ridondanze:

- Un pannello solare per ogni PM Tile.
- Una batteria per ogni PM Tile.
- Una ruota di reazione per ogni PM Tile.
- Una bobina magnetica per ogni PM Tile.
- Un sensore di posizione e di sole per ogni PM Tile.
- Un housekeeping per ogni PM Tile.

Prima di analizzare in maniera più approfondita il Power Management Tile è necessario dare alcune informazioni riguardo le condizioni ambientali operative in cui opera il satellite.

1.3.1 Sottosistemi del satellite

Aramis può raggiungere il livello desiderato di flessibilità attraverso la combinazione dei diversi sottosistemi che sono:

1. Meccanico;
2. Generazione e gestione della potenza;
3. ADCS;
4. Telecomunicazione;
5. Payload;

<u>Sottosistema meccanico</u>

Il sottosistema meccanico è la spina dorsale dell' intero satellite in quanto oltre a fornire la vera struttura fisica, lo tiene tutto insieme e lo protegge dalle condizioni dell' ambiente circostante. Il principale materiale usato per la costruzione è l' alluminio per il semplice fatto che pesa relativamente di meno rispetto agli altri metalli. La struttura scheletrica è composta dai diversi *tiles* quadrati metallici, mentre il *power management* e *telecommunication* sono montati su sottili pannelli avvitati alle stesse barre. Il

numero di queste mattonelle (o tiles), che sono connessi all' interfaccia esterna, dipende principalmente dalla dimensione del satellite e dalla potenza richiesta: ciò offre un grado di libertà ai progettisti in quanto basta aggiungere più moduli per variare dimensione e potenza. Diversamente, il payload è montato all' interno della struttura è può essere cambiato secondo i requisiti della missione. La figura sotto mostra:

Figura 1.5 struttura meccanica di Aramis

Sottosistema di generazione e gestione della potenza

Questo sottosistema, che è responsabile di generare, immagazzinare e fornire la potenza a tutti gli altri sottosistemi, è molto critico in quanto un guasto al suo interno potrebbe portare ogni altro sottosistema allo spegnimento. Perciò un parametro importante nella progettazione è stato sicuramente la tolleranza ai guasti e in questa ottica, diverse sono state le soluzioni adottate per evitare qualsiasi tipo di condizione fuori controllo.

Sottosistema di telecomunicazione di Aramis

Questa parte ricalca praticamente il concetto di modularità in un determinato tile, ma non per questo ciò significa che non possano esserne aggiunti degli altri per venire incontro a particolari criteri di speciali applicazioni. In generale, questo modulo è usato per ricevere comandi e pacchetti di controllo dalla stazione di terra e a sua volta, inviare telemetria e informazioni sullo stato del satellite. Siccome la banda necessaria per lo scambio di queste informazioni è di solito bassa, è stato progettato un collegamento RF per basse velocità e bassi consumi di potenza. Il modulo è stato progettato con componenti COTS che sono stati selezionati per raggiungere un buon livello di tolleranza ai guasti. Vi sono due diverse differenze "in gioco" tra satellite e *ground station*:

- Banda UHF 437 MHz;
- Banda SHF a 2.4 GHz

Entrambi i canali sono implementati su un protocollo half-duplex per ridurre la banda occupata, condividendo la stessa frequenza per downlink e uplink.

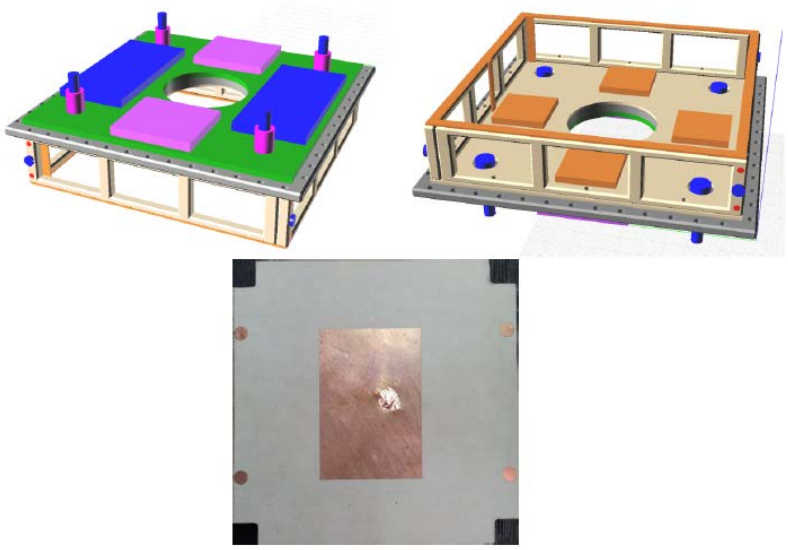


Figura 1.6 modulo di telecomunicazione di Aramis

Sottosistema OBC

Il Tile Computer (OBC) usato in AraMiS consiste di una serie di microcontrollori MSP430 e FPGA che sono responsabili principalmente nella gestione del sistema complessivo. Come si potrà vedere nei capitoli successivi, a queste CPUs è stato aggiunto un DSP(digital signal processor) utile nello sviluppo di questa tesi per accogliere ed elaborare i vari dati che arrivano dal sensore d' immagine. Non solo, l' image_processor, come è stato chiamato, è in grado di interfacciarsi attraverso quattro slots con gli altri processori o più in generale con gli altri sottosistemi, ed inoltre si può ritenere "pilotato" da due altrettante slots da un OBC centrale che gestisce segnali come reset, abilitazioni e modalità di boot.
In definitiva, alcune delle responsabilità che sono chiamati a risolvere sono:

1. Creazione e trasmissione attraverso la scheda del ricetrasmettitore di pacchetti Beacon;
2. Comandi di decodifica ed esecuzione;
3. Algoritmi di determinazione dell' attitudine e del controllo;
4. Salvataggio di dati di houseekeeping;
5. Controllo del sotto-sistema Payload;

Sottosistema del controllo e della determinazione dell' assetto

Questo sottosistema è responsabile di rilevare e modificare l' orientamento del satellite per tenere i sottosistemi nella posizione desiderata. Il controllo d' assetto può essere fatto in maniera attiva (attraverso degli attuatori su precisi comandi dell' OBC) o passiva (di solito è eseguito montando dei magneti permanenti nel satellite che si comportano come un compasso nel campo magnetico terrestre).

Il controllo passivo è estremamente semplice e non consuma potenza, mentre d' altra parte, il principale inconveniente è la mancanza del controllo dello spin a causa del variabile campo magnetico terrestre. In Aramis, il controllo dell' attitudine è eseguita automaticamente usando un bobina magnetica e ruote d' inerzia.

Per quanto riguarda, la parte relative alla determinazione dell' assetto verrà approfondita nel capitolo successivo essendo il contesto nel quale è stato realizzato il sensore di sole. E' bene solo ricordare che i sensori sono componenti COTS, selezionati in base alle dimensioni, peso e soprattutto consumo di potenza per raggiungere la migliore performance.

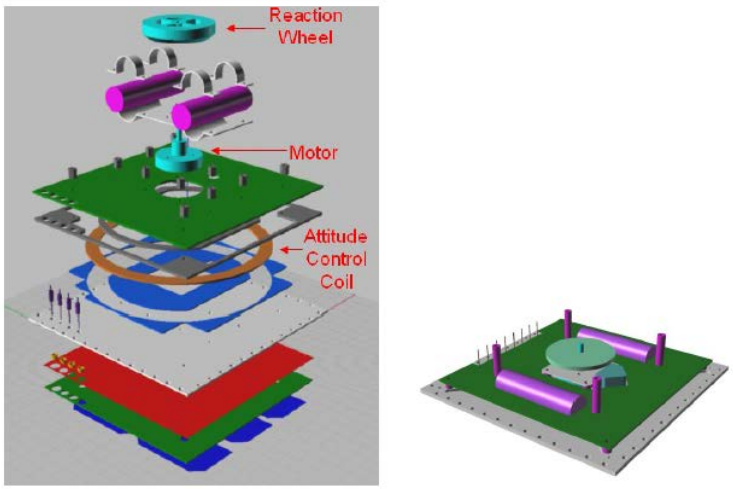


Figura 1.7 modulo ADCS nel satellite Aramis

Payload

Il payload è fortemente dipendente dalla missione e l' architettura è stata sviluppata anche per avere un grado di flessibilità su di esso: i principali requisiti dell' architettura imposti sul payload stesso sono la compatibilità che deve avere con la distribuzione della potenza e il bus dati.

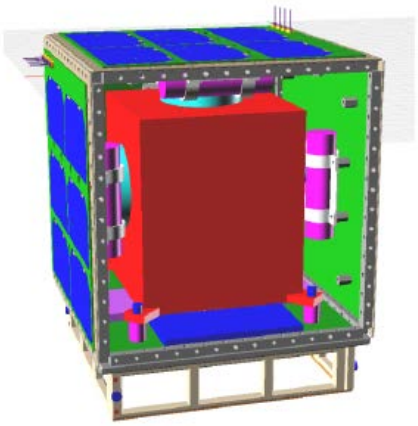


Figura 1.8 Payload presente in Aramis

1.4 Vincoli d’ ambiente spaziale

Il progetto AraMiS ed in particolare i dispositivi elettronici sono fortemente influenzati dalle condizioni ambientali in cui operano. Il satellite é progettato per lavorare ad un'orbita LEO, con una distanza dalla Terra compresa tra i 600 e gli 800 km; questa distanza corrisponde ad una zona compresa tra l'atmosfera terrestre e l'inizio delle fasce di Van Allen, posizione in cui iniziano ad esser presenti forti radiazioni, seppur più deboli rispetto alle fasce inoltrate.
Ad una distanza del genere, l'atmosfera è quasi inesistente, si è quindi in una condizione di vuoto dove il fenomeno della convezione è assente dato che un corpo caldo (ad esempio un componente elettronico) non riesce a dissipare energia a contatto con un fluido più freddo come potrebbe essere l’ aria. Nel vuoto gli scambi di calore avvengono solamente per conduzione od irraggiamento.
Inoltre bisogna prestare attenzione alla presenza di fluidi all'interno di tutti i componenti elettronici e meccanici, i quali potrebbero innescare fenomeni di surriscaldamento o addirittura esplosioni. Prendiamo in esempio le batterie agli ioni di Litio, esse hanno una pressione interna di 0:3 bar, trascurabile sulla Terra, ma nel vuoto tende a creare un'altissima forza per unità di superficie e a far esplodere la batteria. Per questo motivo nel progetto non possono essere usati condensatori di tipo elettrolitico.

1.4.1 Radiazione elettromagnetica

Uno degli aspetti più importanti tenuti in considerazione nel progetto di tutto il sistema ARAMIS è stato l’ambiente dove esso sarebbe dovuto funzionare: lo spazio.
Lo spazio libero presenta alcuni importanti aspetti legati al funzionamento dei circuiti integrati, quali i raggi cosmici. I Raggi Cosmici sono delle particelle prodotte dalle reazioni termonucleari che possono avvenire in alcuni fenomeni celesti, come l’esplosione di supernovae, i fenomeni di collassi stellari o direttamente all’interno delle stelle stesse. Essi sono particelle subatomiche e fotoni ad alta energia, che bombardano costantemente la Terra da ogni direzione. Le energie di queste particelle ricoprono un vasto intervallo fino ad arrivare oltre 1020eV. Tali raggi sono quindi, particelle provenienti dallo spazio, che, interagendo con gli atomi presenti nell'atmosfera, producono un alto numero di particelle che arrivano fino a terra.
Per sistemi orbitanti attorno alla terra, come i satelliti per le telecomunicazioni, la prima sorgente di radiazione sono gli elettroni ed i protoni intrappolati dalla magnetosfera terrestre e sono di interesse rilevante ad altitudini comprese fra 1000Km e 32000Km. La loro estensione è in realtà maggiore, ma il livello del flusso di particelle diminuisce rapidamente al di fuori di questo range di altitudine.

Le distribuzioni in altitudine per protoni ed elettroni sono significativamente diverse fra loro, ma entrambe sono accomunate dal fatto che sia per gli elettroni che per i protoni, le particelle più energetiche si trovano ad altitudini maggiori.

I dispositivi elettronici, impiegati nella fabbricazione di satelliti ed apparati spaziali di volo, devono normalmente rispondere ai requisiti normativi definiti ed emessi dalle agenzie spaziali NASA ed ESA. Per quanto non previsto in questi requisiti, ci si riferisce di regola alle specifiche MIL applicabili al settore avionico-spaziale. Questo perché i dispositivi elettronici al silicio sono particolarmente sensibili alle radiazioni cosmiche e possono cessare di funzionare se queste superano particolari livelli energetici, anche in dipendenza dalla tecnologia di fabbricazione utilizzata. Anche questi aspetti di resistenza alle radiazioni cosmiche sono regolamentati dalle normative sopra citate; queste prevedono per le differenti applicazioni l'uso di componenti definiti nel linguaggio tecnico "rad-hard" o "rad-tollerant".

I componenti rad-hard sono normalmente in grado di funzionare anche se esposti a radiazioni di livello fino a 100krad, ed in alcuni casi fino a 300krad; per quelli rad-tollerant i livelli si riducono rispettivamente a 10krad e 30krad. Dispositivi certificati resistenti alle radiazioni, per come sopra esposto, in accordo con la normativa hanno un costo fino a 1000 volte quello dell'equivalente dispositivo qualificato per uso industriale. La reperibilità sul mercato di materiali con queste caratteristiche richiede tempi di consegna che vanno tipicamente da 6 a 24 mesi.

Tutto questo è in fortissimo contrasto con lo sviluppo del mercato spaziale, che richiede già oggi la fabbricazione di apparati e satelliti in soli 6 mesi dall'emissione di un ordine (mercato spaziale commerciale) e a costi sempre più bassi.

Il progetto ARAMIS che si basa sull'utilizzo di componenti commerciali COTS (Components Off The Shelf) rappresenta una sfida alle considerazioni precedenti. Infatti, sebbene il satellite orbiterà in un'orbita bassa, circa 600 Km, sicuramente potrà risentire di queste radiazioni, per questo numerose scelte di progetto sono state fatte in funzione di questo.

Tra le principali conseguenze dell'interazione delle radiazioni cosmiche con i dispositivi a semiconduttore vi sono gli effetti di danneggiamento ad evento singolo (SEE). Questo tipo di danneggiamento riguarda l'azione di una singola particella che attraversa il substrato dei circuiti.

Tali effetti devono la loro importanza al notevole aumento della miniaturizzazione dei dispositivi elettronici.

I single event effect sono di diverso tipo e possono essere distinti, in effetti transitori ed effetti permanenti. In alcuni casi gli effetti permanenti possono essere così disastrosi da causare la rottura totale del dispositivo.

Per i diversi effetti ad evento singolo si può fare la seguente classificazione:

- Effetto transitorio: SEU ovvero single event burnout.
- Effetto permanente: SEL, single event latchup; SEB, single event burnout e SEGR, single event gate rupture ed altri effetti ad evento singolo.

1.4.1.1 Single event upset

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 1.9, mostra un esempio di SEU su un dispositivo elettronico.

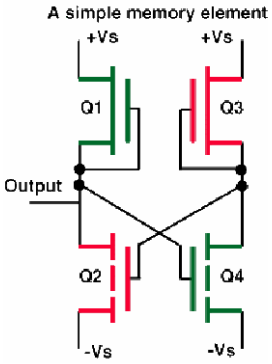


Figura 1.9: SEU su un dispositivo a un bit

Il circuito è disegnato in modo da avere due stati stabili rappresentati reciprocamente con “0” ed “1”. Per ogni stato due transistor vengono commutati on ed off. Un SEU avviene quando una particella energetica che attraversa il dispositivo fa cambiare stato al transistor.

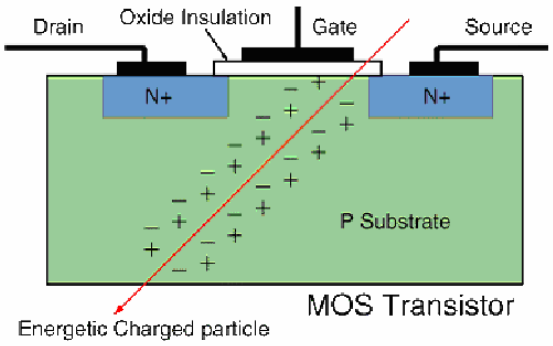


Figura 1.10: generazione di spike di corrente a causa di effetti SEU nei MOSFET

La figura 1.10 mostra, invece, come la particella energetica che penetra nel dispositivo genera un segnale elettrico spurio. La particella produce, infatti, cariche durante il suo percorso sotto forma di elettroni e lacune, che si addensano al source ed al drain e generano un impulso di corrente.

1.4.1.2 Single event latchup

Un single event latchup è un percorso anomalo di corrente, prodotto fra strutture n-p-n o p-n-p, che sparisce soltanto nel momento in cui viene rimossa l'alimentazione dal dispositivo. Tale problema affligge la maggior parte dei dispositivi microelettronici.

Si ha un latchup quando una particella carica che transita nel dispositivo, induce una corrente tale da aumentare la corrente operativa oltre le specifiche del dispositivo e notevolmente superiore a quella incontrata nelle normali operazioni.

Questo effetto è potenzialmente distruttivo perché la corrente aumenta oltre le specifiche per cui il dispositivo è realizzato, quindi se il dispositivo non viene protetto o limitato in corrente rischia di subire un danneggiamento irreparabile o comunque riduce notevolmente le sue prestazioni peggiorando notevolmente le sue specifiche di funzionamento.

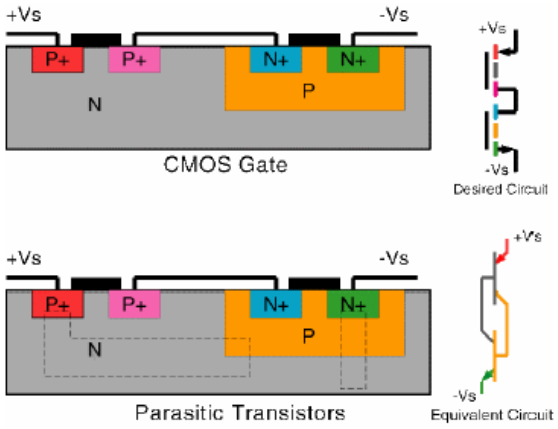


Figura 1.11: schema di un latchup CMOS

Un latchup nei circuiti digitali può avvenire anche quando un impulso spurio di corrente, come quelli prodotti dai raggi cosmici, attiva il transistor parassita presente nella tecnologia CMOS, che nelle normali condizioni operative è fuori uso. Il risultato che si ottiene è che il dispositivo permane nello stato di On. Si parla di transistor parassiti, perché il bulk dei CMOS contiene due transistor bipolari parassiti che formano una struttura a quattro layer.

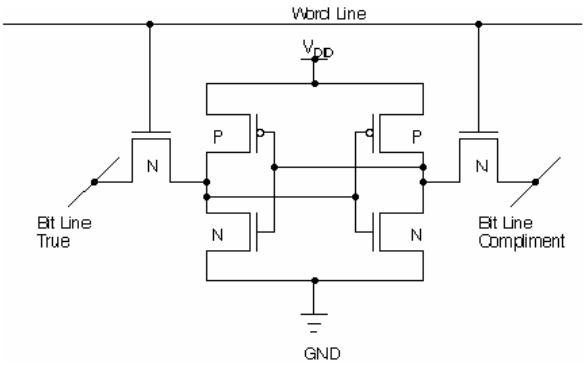


Figura 1.12: schema di una cella CMOS

Le sorgenti di latchup nei CMOS non sono contenute nelle normali operazioni di tali dispositivi ma, segnali transitori posti ai terminali di ingresso o di uscita li possono inavvertitamente triggerare portandoli nella condizione di On.

Quando i transistor parassiti sono attivi, pilotano correnti molto grandi che possono provocare danni anche permanenti. Una volta che il dispositivo è andato in latchup la struttura a quattro layer è portata in conduzione continua; da questa condizione si esce soltanto se viene rimossa l'alimentazione in un tempo dell'ordine di millisecondi.

Durante il latchup le correnti possono essere molto elevate dell'ordine delle centinaia di milliampere o maggiori e, circolando nel dispositivo, ne aumentano la temperatura. Tali aumenti di temperatura non solo possono danneggiare il circuito, ma possono far sì che il latchup si estenda ad altre regioni che prima non ne erano interessate.

Proprio per le sue potenzialità distruttive, il latchup si pone come un grave problema per i sistemi spaziali.

Quindi in generale, gli effetti descritti in questo paragrafo, causati dalla radiazioni, sono fenomeni di tipo istantaneo, esistono però anche dei fenomeni il cui effetto è legato alla quantità di radiazioni assorbite; é chiaro quindi che questi fenomeni dipendono dal tempo trascorso in orbita. Perciò si parlerà di Total Dose per indicare la quantità massima di radiazioni che può accumulare un dispositivo prima di avere dei malfunzionamenti (esempio può essere, la tensione di soglia di un CMOS che tende ad aumentare gradualmente con la quantità di radiazione assorbita e conseguente aumento dei tempi di propagazione con possibili errori).

1.4.2 Temperatura

Durante l'orbita, nello stesso istante, le diverse facce del satellite si trovano in due condizioni differenti: la faccia illuminata assorbe i raggi solari, mentre quella nella direzione opposta é in ombra. E' quindi presente un forte gradiente termico (cicli termici) tra le diverse facce del satellite. Inoltre a causa della quasi totale mancanza di atmosfera l'irraggiamento solare è nell'ordine di 1300 Wm^{-2}, molto maggiore che sulla Terra.

Si ricorda inoltre che nello spazio la temperatura dipende dal bilancio energetico della potenza assorbita dal Sole, di quella convertita in altre forme di energia e di quella generata dal surriscaldamento dei componenti. Secondo una stima teorica la temperatura T di lavoro del satellite è compresa in un intervallo di [-30, +40]°C quando la potenza massima Pj dissipata dai circuiti interni é di circa 200 W.

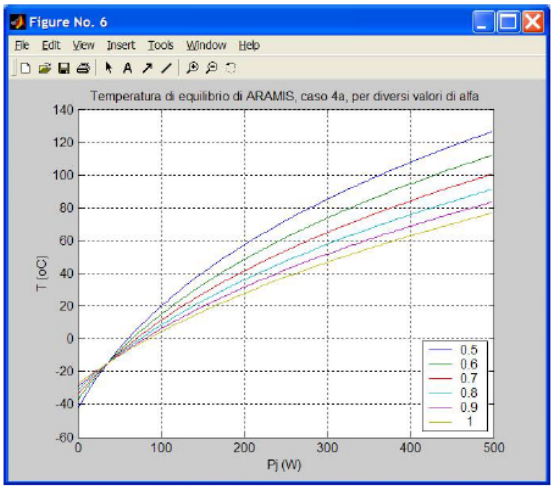


Figura 1.13: Andamento di T rispetto a Pj

Capitolo 2

Sun Sensor: determinazione d'assetto

2.1 Il modulo ADCS

L’orientamento nello spazio di un sistema di coordinate (solidale con il corpo) rispetto ad un sistema di riferimento assegnato è detto assetto di un corpo.

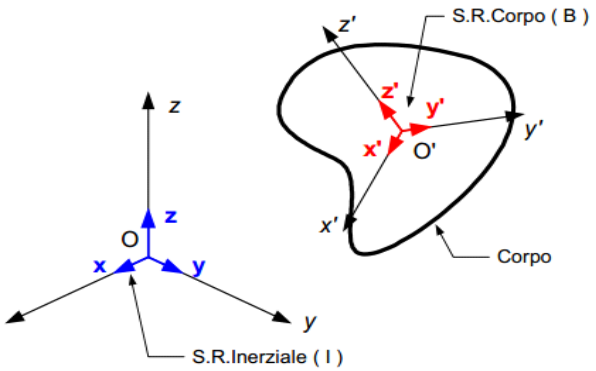


Fig. 2.1 Sistema di riferimento

La determinazione e il controllo dell’assetto di un satellite (Attitude Determination and Control System - ADCS) stabilizzano il veicolo nel corso della missione e provvedono al suo orientamento lungo la direzione desiderata nonostante la presenza di disturbi esterni che si manifestano inevitabilmente su di esso sotto forma di torsioni. Per fare ciò, è necessario che il veicolo determini il proprio assetto, utilizzando i sensori, e lo controlli, usando gli attuatori.

Spesso, oltre ad annullare i disturbi esterni, l’ADCS deve riorientare il veicolo (compiendo manovre di rotazione) per ripuntare il payload, i pannelli solari o le antenne. Questi puntamenti periodici possono richiedere degli attuatori più grandi. Al fine di ottenere un’orientazione corretta, si dovranno utilizzare i riferimenti esterni, come il Sole, le stelle o la direzione del campo magnetico locale, per determinare l’assetto assoluto del veicolo. Questi riferimenti esterni sono spesso misurati come distanze angolari, cioè come vettori, ed ognuno di tali vettori fornisce solo due dei tre parametri indipendenti necessari per specificare l’orientazione del satellite. Ciò porta all’utilizzo di più sensori di diverso tipo contemporaneamente. Inoltre, aldilà dei riferimenti esterni menzionati, anche i giroscopi possono fornire un’indicazione dell’assetto del satellite.

Il sistema di controllo dell'assetto e dell'orbita di un satellite mantiene l'orientamento del veicolo durante l'orbita e ne conserva la traiettoria orbitale, il che è essenziale per il successo e la sicurezza della missione.
L'ADCS è, quindi, costituita da tre parti: la prima è composta da una serie di sensori, che dovrebbero avere la capacità di percepire il comportamento corrente della piattaforma; la seconda parte consiste di un insieme di meccanismi mediante i quali può essere subito evidenziato e corretto un errore di assetto; la terza e ultima parte è formata da dei microcontrollori che rivedono tutti gli algoritmi progettati, che a loro volta devono essere in grado di calcolare errori tra l'orientamento attuale e quello “desiderato”.

2.2 Il sistema sensoristico e cenni di controllo

L’hardware della misura di assetto è usato per determinare l’assetto di un satellite rispetto ad un definito sistema di riferimento. Le grandezze di uscita della misura possono essere gli angoli di Eulero del satellite nel sistema di riferimento orbitale o le componenti del vettore sole nel sistema di riferimento assi corpo.
Dotare un satellite di sensori lo svincola dal dialogo continuo con la stazione di terra e fornisce una certa autonomia di comando. In taluni casi i sensori svolgono anche la funzione di sonde nel payload, ad esempio un magnetometro per campionare il campo magnetico terrestre.
Esistono due classi di base di sensori di assetto. La prima fa misure assolute; sono utilizzati negli algoritmi di determinazione dell'assetto statico. La seconda classe si occupa di misure relative, a cui appartengono strumenti giroscopici. L’hardware per la determinazione dell’assetto in generale include:

- Sensori di Sole :

I sensori solari sono dei rilevatori di luce visibile che misurano la differenza tra due angoli prodotti dalla luce attraverso delle fessure nella loro montatura. Per funzionare al meglio necessitano di angoli di vista molto aperti, e sono molto accurati e attendibili. Perciò sono molto diffusi.
In orbita bassa occorre considerare però il rapido alternarsi tra la semi-orbita illuminata e quella eclissata e prevedere altri sistemi per determinare la posizione in assenza del sole. Inoltre in orbite LEO (Low Earth Orbit) la terra occupa circa il 40% del cielo e talvolta può causare interferenze nei sensori a causa della luce riflessa.

- Sensori di Terra (o anche chiamati di orizzonte):

I sensori di orizzonte sono dei rilevatori infrarossi che percepiscono il contrasto tra il freddo dello spazio e il calore dell’atmosfera terrestre a circa 40 km dalla sua superficie. Questi sensori, che possiedono uno

stretto campo di vista, vengono montati a bordo dei satelliti dotati di spin per misurare la fase della Terra e gli angoli di corda che, insieme alla geometria dell'orbita, definiscono due angoli rispetto al vettore nadir. Questo tipo di sensori possono essere utilizzati anche a bordo di satelliti orientati costantemente verso il nadir: possono puntare parte del limbo (se posizionati in orbite basse) o l'intero disco terrestre (se in orbite GEO).
Precisioni tipiche per sistemi che utilizzano sensori di orizzonte sono comprese tra i 0.1 gradi e i 0.25 gradi. Solo qualche applicazione particolarmente precisa si avvicina ai 0.03 gradi. Vi sono due possibilità: o individuare due corde che tagliano la circonferenza della Terra, da questi valori si calcola il centro della Terra, oppure attraverso due diametri opposti si calcola la differenza dei segnali ai rami opposti e quando è 0 si ha che il satellite vede la Terra.

- Sensori di campo magnetico :

I magnetometri sono sensori semplici, attendibili e leggeri che misurano sia la direzione che l'intensità del campo magnetico terrestre. La loro misura, comparata con il campo magnetico terrestre noto, ci aiuta a stabilire l'assetto del satellite. La loro precisione, purtroppo, non è così elevata come quella dei sensori stellari o di orizzonte poiché il campo magnetico dipende da molti parametri e quindi (in realtà) esso non è noto in maniera così precisa. Per aumentare l'accuratezza spesso si combinano i dati provenienti dal magnetometro con quelli provenienti da eventuali sensori solari o da sensori di orizzonte. Quando un veicolo dotato di *magnetic torquer* passa attraverso inversioni del campo magnetico durante ogni orbita, viene utilizzato un magnetometro per misurare la polarità in uscita dall'attuatore magnetico. Inoltre, quando il magnetometro effettua una misura, i toquer dovranno naturalmente essere spenti per non influenzare la misura stessa.

- Sensori di stelle :

Sono molto precisi, confrontano l'immagine del cielo stellato con una mappa di riferimento, considerano una stella campione come zero e ricavano l'informazione sull'assetto. I dispositivi più sofisticati non si limitano a tracciare le stelle come spot luminosi, ma riescono a riconoscere il tipo di stella che stanno osservando fornendo in questo modo un'orientazione comparata a un riferimento inerziale.
Si hanno due varianti:

- Star tracker: sono dei puntatori che vengono usati per stabilizzare il satellite sui tre assi: puntano una o più stelle e, ricavando le informazioni necessarie, calcolano l'assetto del veicolo
- Star scanner: sono degli analizzatori; vengono usati a bordo di satelliti che ruotano su se stessi: le stelle passano nel campo di vista dello scanner attraverso una serie di fenditure e, dopo alcuni passaggi, è possibile risalire all'assetto del veicolo.

- Ricevitore GPS :

I Global Positioning System (GPS) sono dispositivi di navigazione estremamente precisi. Sono usati principalmente per identificare la posizione nello spazio del satellite, e in certi casi per la determinazione dell'assetto utilizzando i diversi segnali provenienti da antenne separate a bordo dello stesso satellite. Questi sensori vengono spesso montati anche a bordo di piccoli satelliti su orbite LEO a causa del loro basso costo e del peso limitato.

- Sensori giroscopici :

I giroscopi vengono impiegati come sensori inerziali che misurano la velocità lo spostamento angolare del satellite. Non sono in grado di fornire un riferimento assoluto e pertanto necessitano di essere accoppiati con altri sensori in grado di fornire riferimenti esterni, ottenendo così indicazioni d'assetto estremamente precise.

Il controllo di un satellite è l'insieme delle operazioni meccaniche che si compiono per correggerne il moto in modo che si mantenga la traiettoria e/o l'orientamento voluti.

- Razzi per il controllo d'assetto :

Il controllo d'assetto richiede spinte molto basse, dell'ordine dei mN allora non si possono utilizzare i motori chimici i quali danno spinte di almeno 1N , si utilizza la propulsione elettrica, in particolare un campo magnetico accelera un propellente ionizzato il quale viene espulso.

- Ruote d'inerzia e di reazione :

Sono attuatori che si basano su accelerazioni e decelerazioni di rotori, tutte scambiano momento della quantità di moto con il satellite infatti tutte le coppie applicate sulle ruote vengono applicate uguali e contrarie sul satellite. Per una velocità di rotazione detta di sincronismo si ha che la coppia è nulla e che la ruota è satura. Per desaturarla occorre frenarla ma staccandola dal satellite: ciò si ottiene tramite dei mazzetti nella ruota i quali forniscono una coppia esterna.

- Ruota di reazione: ha velocità angolare nominale nulla, agisce soltanto in presenza di disturbi;
- Ruota di inerzia: ha velocità angolare nominale NON nulla, viene posta in rotazione ad alta velocità intorno al suo asse al fine di garantire una stabilità giroscopica;
- Ruote giroscopiche: non vengono accelerate ma viene fatto ruotare l'asse di rotazione.

Si preferiscono le ruote d'inerzia in quanto le ruote di reazione presentano attrito statico necessario a metterle in rotazione inoltre nel momento di attivazione si deve evitare, mediante coppie esterne, che il satellite si metta in rotazione nel verso opposto.

- Attuatori magnetici :

Generano una coppia inducendo una corrente in una spira immersa nel campo magnetico terrestre. Le coppie che si possono realizzare vanno da 10^{-3} Nm a 10^{-6} Nm

2.3 La classificazione dei sensori solari

Si hanno le seguenti varianti:

- Sensori di presenza di Sole: utilizzano almeno due rilevatori posti angolati uno di fronte all'altro, l'informazione risultante è binaria. Forniscono in uscita la sola informazione circa la presenza o assenza del sole nel campo visivo dello strumento, ignorando la direzione dei raggi solari incidenti. Il principio di funzionamento del sensore di presenza è molto semplice e si basa sulla legge di rifrazione di un prisma ottico: detto n l'indice di rifrazione del medesimo e noto l'angolo d'incidenza del Sole, l'angolo di rifrazione θ' è dato da nsen(θ)=sen(θ'). Il prisma presenta delle fotocellule montate lateralmente, le quali forniscono una corrente diversa da zero soltanto per un certo intervallo dell'angolo di incidenza dei raggi luminosi sulla base del prisma; tale intervallo è definito dalle caratteristiche ottico-geometriche del prisma.

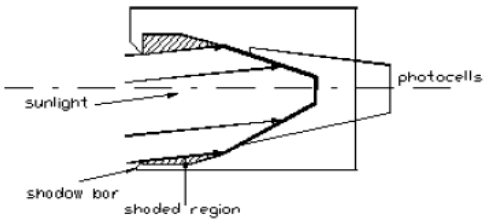


Fig.2.2 Sensore di presenza di sole.

- Sensori di posizione di Sole.

2.3.1 Analogici e digitali

I sensori di posizione di sole possono essere classificati in:

- Analogici, la cui uscita è una funzione continua rispetto all'angolo di vista del Sole, generalmente sono composti da stadi di condizionamento del segnale proveniente dal sensore attivo montato a bordo.
- Digitali, che forniscono un'uscita discreta ad una certa frequenza funzione dell'angolo di vista del Sole. L'elettronica di bordo è basata generalmente su un microprocessore o microcontrollore e comunicano con il bus del satellite attraverso interfacce di tipo digitale.

I sensori analogici si basano sull'uso di celle fotovoltaiche, o di fotoresistenze. Le prime, in particolare, sono in grado di fornire in uscita una corrente proporzionale all'energia radiante che incide sulla medesima, che a sua volta è proporzionale all'area della cella, al flusso di energia luminosa e all'angolo d'incidenza θ formato dalla normale alla superficie d'esposizione con la direzione dei raggi incidenti. La relazione tra corrente ed angolo può quindi scriversi come I= k cos(θ), da cui il nome di "cosine detectors".

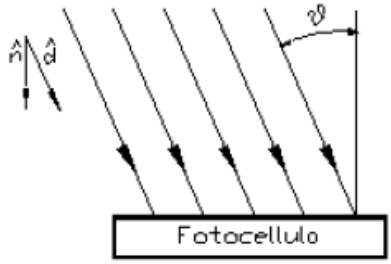


Fig.2.3 Angolo d'incidenza dei raggi solari.

I sensori digitali classici forniscono l'angolo d'incidenza dei raggi solari e sono costituiti per lo più da due parti fondamentali: la componente di comando e quella di misura (o "misuratore"). La componente di comando è un sensore che indica quando il Sole si trova nel piano della fenditura della componente di misura, a sua volta costituita da una fessura esterna e da una griglia interna di digitalizzazione. A seconda della direzione della radiazione luminosa, il misuratore fornisce un valore digitale relativo all'angolo d'incidenza dei raggi solari.

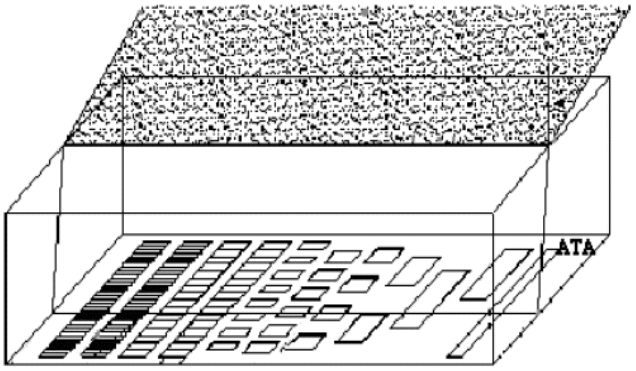


Fig.2.4 Sensore solare digitale con codice di Gray.

La griglia interna di digitalizzazione presenta una serie di fessure posizionate secondo la logica del codice che fornirà il valore digitale dell'angolo. Il codice più usato è quello Gray che, variando di un solo bit alla volta, riduce il rischio di errori nelle transizioni tra un valore e l'altro.

Il campo di vista di tali sensori è superiore ai 110°x110° con una precisione al più di 0.125°, quando si ricorre ai bit d'interpolazione. Spesso vengono montati più sensori al fine di estendere la "visuale" su tutta la sfera celeste. Questo tipo di sensori digitali devono, quindi, tale nome al fatto che la presenza o meno della luce solare in alcune ben definite zone sensibili determina la creazione di segnali digitali che identificano l'angolo del Sole.

2.4 Calcolo dell'assetto e riferimenti

2.4.1 Principali manovre per il controllo d'assetto di un satellite

Un satellite in orbita può generalmente compiere due tipi di operazioni per variare le sue condizioni operative.

La prima è una manovra detta di *shifting*, ovvero una traslazione che consente al corpo di spostarsi e variare la sua orbita, sia in termini di ampiezza (altezza dal suolo) sia in termini di inclinazione, ma anche di conferire al satellite una maggiore o minore velocità. Tale operazione si svolge usualmente con l'impiego di *thrusters* multipli a getto di gas o di ioni. Tale tipo di attuatore si basa sulla conservazione della quantità di moto. Infatti un'espulsione di massa (gas) con una certa velocità genera una variazione di velocità sul satellite, in funzione della massa, in direzione uguale e contraria.

$$\sum Q = m_g v_g + m_s v_s = 0$$

$$v_s = -v_g \frac{m_g}{m_s}$$

La seconda manovra che un satellite può compiere prende il nome di *slewing*. Si tratta di una rotazione dello stesso attorno al suo centro di massa. Le rotazioni sono necessarie per permettere al satellite di puntare una delle sue facce in una direzione desiderata. I motivi sono i più disparati, ad esempio dal puntamento dell'antenna per telecomunicazioni all'esposizione di un determinato pannello solare verso il sole o proteggere la strumentazione sensibile dai raggi cosmici. Solitamente tali manovre non sono rapide e i sistemi impiegati per tali rotazioni sono ruote di inerzia, ruote di reazione o attuatori di momento magnetici. Per manovre in cui è richiesta una rotazione più rapida vengono solitamente impiegati i *thruster*, che forniscono però una precisione minore sull'angolo di rotazione. La scelta del sistema da impiegare varia dunque in funzione delle velocità di rotazione necessarie a ottenere l'assetto desiderato.

I sistemi impiegati per lo slewing sono impiegati anche per smorzare attivamente eventuali momenti rotatori di disturbo al fine di mantenere l'assetto corretto.

La correzione di assetto può essere effettuata in due modi:

- in modo passivo si parla di compensazione di assetto e sono impiegati sistemi quali gradiente di gravità, stabilizzazione di spin e magnete permanente.
- In modo attivo, invece, si ha una variazione comandata dell'assetto e vengono impiegati gli stessi sistemi sopraindicati per la manovra di *slewing*.

2.4.2 Sistemi di riferimento

Lo studio del moto di un satellite in orbita attorno alla terra è piuttosto complesso, in quanto si tratta di un moto con sei gradi di libertà: tre coordinate per individuare il centro del velivolo nello spazio e altre tre per determinare l'assetto attorno al centro. Si rende quindi necessario definire dei sistemi di riferimento per definire la posizione del satellite ad ogni istante e poter quindi impartire comandi e ricevere dati posizionali univoci.

- *Body frame* (Assi corpo)

Iniziamo col definire un sistema di assi corpo (Body Frame). Con tale dicitura si intende una terna destrorsa di assi cartesiani centrata nel centro di massa del satellite, con l'asse X_B nella direzione del moto, l'asse Z_B rivolto verso terra e l'asse Y_B ortogonale al piano X_BZ_B tale che la terna sia destrorsa.
Si definiscono inoltre i moti di rollio (*roll*) come rotazione antiorario attorno all'asse X_B, beccheggio (pitch) come rotazione antioraria rispetto all'asse Y_B e imbardata (*yaw*) come rotazione antioraria rispetto all'asse Z_B.

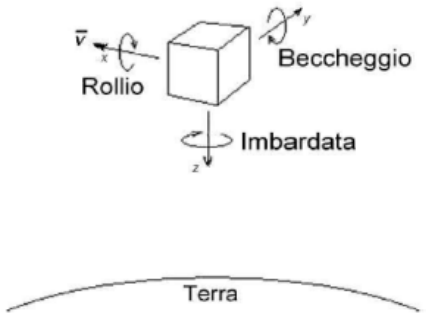


Fig.2.5 Body Frame.

I tre assi X_B, Y_B, Z_B assumono quindi rispettivamente la denominazione di asse di rollio, di beccheggio e di imbardata.

- *Orbit frame*

Con *Orbit frame* si intende un sistema di assi avente origine nel centro di massa del satellite. Gli assi sono solidali all'orbita percorsa dal satellite, con l'asse Z_O rivolto verso il centro dell'orbita, l'asse X_O tangente all'orbita e rivolto in direzione del moto e l'asse Y_O ortogonale ai due precedenti tale da rendere la terna destrorsa.

Lo scostamento tra *Orbit frame* e *Body frame* si misura tramite tre rotazioni attorno agli assi. Questi angoli di scostamento rappresentano l'assetto del satellite, più precisamente lo scostamento dalla condizione zero rappresentata dall' *Orbit frame*.

- *Earth-centered inertial* (ECI)

L'*ECI frame*, anche chiamato *Inertial Geocentric Reference Frame* è un sistema di riferimento inerziale centrato sulla terra, con asse Z_I diretto verso il polo Nord. Il riferimento è di tipo inerziale, significa che

è orientato rispetto alle stelle fisse. L'asse X_I è orientato verso Aries (Equinozio di primavera), e l'asse Y_I ortogonale a entrambi tale che la terna sia destrorsa.

Questo sistema di assi è solitamente impiegato per i dati di navigazione orbitale e per lo studio del moto di un corpo orbitante attorno alla terra.

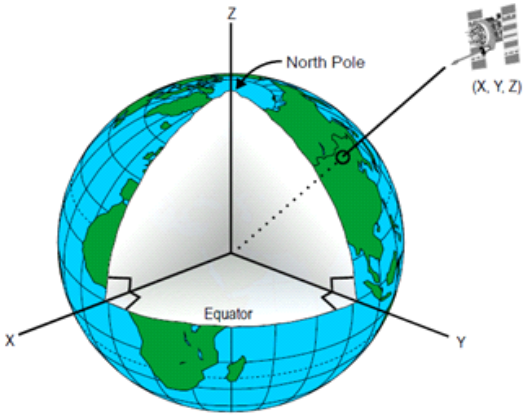


Fig.2.6 Earth-centered inertial.

- *Earth-centered earth fixed* (ECEF)

Questo riferimento terrestre è molto utilizzato per l'analisi e la rielaborazione di dati provenienti dai satelliti. Definisce la posizione di un generico punto nello spazio attraverso tre coordinate riferite al centro di massa dell'oggetto considerato.

L'ECEF ha origine nel centro della terra e gli assi sono fissi e solidali alla stessa. L'asse Z_E ha direzione del polo nord, l'asse X_E punta in direzione del meridiano di Greenwitch e giace sul piano equatoriale, così come l'asse Y_E che è ortogonale ai precedenti e orientato in modo da rendere la terna destrorsa.

Si introducono inoltre due angoli caratteristici quali λ, longitudine e φ latitudine, come mostrato nella figura seguente.

Il sistema ECEF si trova in rotazione rispetto al sistema ECI con velocità angolare Ω_E = 7.272*10-5 rad/s rispetto all'asse Z_E, che è coincidente con l'asse Z_I. Ω_E è la velocità di rotazione della terra rispetto a un sistema inerziale.

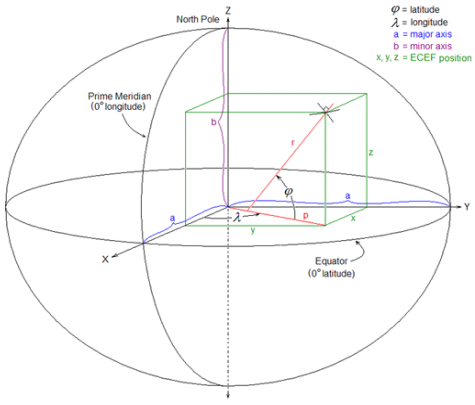


Fig.2.7 Earth-centered earth fixed.

La terra possiede anche una velocità di rivoluzione attorno al sole, Ω_R = 2*10-7 rad/s, che verrà però trascurata nel resto della trattazione.

2.4.3 Angoli di Eulero

Gli angoli di Eulero (Ψ, Θ, Φ) sono tre quantità indipendenti utili a definire la posizione di un generico sistema di riferimento rispetto ad un altro, non necessariamente inerziale. Tutti le terne di assi sono considerate cartesiane e destrorse.

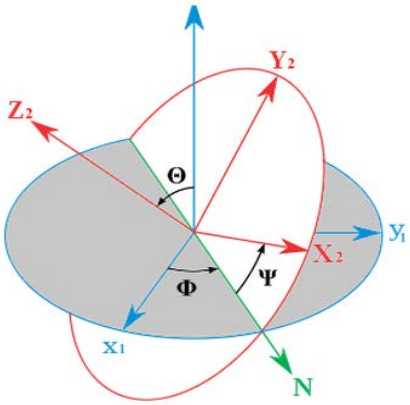


Fig.2.8 Angoli di Eulero.

Questi angoli definiscono una sequenza di rotazioni (Ψ, Θ, Φ) non invertibile, e che quindi non costituisce una base lineare. Tale trasformazione permette tuttavia di esprimere le componenti di un vettore nell'uno o nell'altro sistema di riferimento considerato.

I tre angoli sono determinati dati due sistemi Σ_1 e Σ_2 nel seguente modo:

- Θ, ($0 < \Theta < \pi$) è l'angolo compreso fra l'asse Z_1 e l'asse Z_2;
- Φ, ($0 < \Phi < 2\pi$) è l'angolo compreso tra l'asse X_1 e la linea dei nodi, che indicheremo con N. la linea dei nodi è l'intersezione tra i piani X_1Y_1 e X_2Y_2;
- Ψ, ($0 < \Psi < 2\pi$) è l'angolo tra la linea dei nodi N e l'asse X_2.

E' possibile sovrapporre il sistema Σ_2 con il sistema Σ_1 mediante tre rotazioni Ψ, Θ, Φ ben definite:

- Rotazione antioraria di angolo Ψ, eseguita attorno all'asse Z_2. Indichiamo con Σ'_2 il sistema così ottenuto
- Rotazione antioraria di angolo Θ, eseguita attorno alla linea dei nodi (che dopo la prima rotazione coincide con X_2): questa rotazione sovrappone Z_2 con Z_1. Indichiamo il nuovo sistema come Σ''_2
- Rotazione antioraria di angolo Φ intorno all'azze Z ($Z_2 = Z_1$). Tale rotazione porta a far coincidere gli assi X_2 con X_1.

Poiché gli assi X e Z coincidono, allora anche gli assi Y coincidono in quanto i due sistemi sono entrambi ortogonali destrorsi.

Alle tre rotazioni è possibile assegnare tre matrici di rotazione, che indicheremo con R_Ψ, R_Θ, R_Φ:

$$R_{\Psi} = \begin{bmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$R_{\Theta} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Theta & \sin\Theta \\ 0 & -\sin\Theta & \cos\Theta \end{bmatrix}$$

$$R_{\Phi} = \begin{bmatrix} \cos\Phi & \sin\Phi & 0 \\ -\sin\Phi & \cos\Phi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Si dimostra quindi che:

$$\Sigma_1 = R * \Sigma_2$$

Dove R è la matrice di rotazione globale data dal prodotto delle tre matrici di rotazione R_{Ψ}, R_{Θ}, R_{Φ} nell'ordine:

$$R = R_{\Psi} * R_{\Theta} * R_{\Phi}$$

E' ovviamente possibile anche il passaggio inverso tra i due sistemi di riferimento utilizzando la trasposta della matrice di rotazione globale R^T:

$$\Sigma_2 = R^T * \Sigma_1$$

Con:

$$R^T = (R_{\Psi} * R_{\Theta} * R_{\Phi})^T = R_{\Phi}{}^T * R_{\Theta}{}^T * R_{\Psi}{}^T$$

2.4.4 Quaternioni

In matematica i quaternioni sono entità introdotte da William Rowan Hamilton nel 1843, come estensione quadridimensionale dei numeri complessi.

I quaternioni costituiscono un corpo non commutativo, vale a dire che hanno tutte le proprietà usuali di un campo eccetto la commutatività del prodotto. Essi costituiscono un campo quadridimensionale complesso, scrivibile come:

$$\eta + e_1 i + e_2 j - e_3 k$$

Con η,e_1,e_2,e_3 numeri reali e i, j, k simboli letterali.

Somma e prodotto tra quaternioni sono definiti tenendo conto della relazione:

$$i^2 = j^2 = k^2 = ijk = -1$$

2.4.5 Parametri di Eulero

I parametri di Eulero sono una parametrizzazione dell'assetto riferita ai quaternioni. Si tratta di quattro parametri non singolari, i cui vincoli sono relativamente facili da stabilire.

Una rotazione α attorno a un asse

$$\bar{a} = a_1 \bar{l} + a_2 \bar{m} + a_3 \bar{n}$$

è ottenibile coi quaternioni come:

$$\bar{q} = \begin{bmatrix} \eta \\ e_1 \\ e_2 \\ e_3 \end{bmatrix} = \begin{bmatrix} \cos \alpha/2 \\ a_1 \cdot \sin \alpha/2 \\ a_2 \cdot \sin \alpha/2 \\ a_3 \cdot \sin \alpha/2 \end{bmatrix}$$

I quaternioni non sono tra loro indipendenti e vale la relazione:

$$\|q\| = \eta^2 + e_1^2 + e_2^2 + e_3^2 = 1$$

Presi *u*, *v*:

$$u = \begin{bmatrix} 0 \\ u \end{bmatrix} = \begin{bmatrix} 0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} \qquad v = \begin{bmatrix} 0 \\ v \end{bmatrix} = \begin{bmatrix} 0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix}$$

La rotazione da *u* a *v* è descritto da:

$$\begin{bmatrix} 0 \\ u_1 \\ u_2 \\ u_3 \end{bmatrix} = q_1 \otimes \begin{bmatrix} 0 \\ v_1 \\ v_2 \\ v_3 \end{bmatrix} \otimes \overline{q_1}$$

con q_1, q_1 seguenti quaternioni:

$$q_1 = \begin{bmatrix} \eta_1 \\ e_1 \end{bmatrix} = \begin{bmatrix} \eta_1 \\ e_{11} \\ e_{12} \\ e_{13} \end{bmatrix} \qquad \overline{q_1} = \begin{bmatrix} \eta_1 \\ -e_1 \end{bmatrix} = \begin{bmatrix} \eta_1 \\ -e_{11} \\ -e_{12} \\ -e_{13} \end{bmatrix}$$

E definendo il prodotto tra quaternioni $\otimes$ come:

$$q_1 \otimes q_2 = \begin{bmatrix} \eta_1 \\ e_1 \end{bmatrix} \otimes \begin{bmatrix} \eta_2 \\ e_2 \end{bmatrix} = \begin{bmatrix} \eta_1 e_2 + \eta_2 e_1 + S(e_1) e_2 \\ \eta_1 \eta_2 - e_1^T e_2 \end{bmatrix}$$

E alla quale si fa riferimento alla matrice S, definita come:

$$S(e) = \begin{bmatrix} 0 & -e_3 & e_2 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{bmatrix}$$

2.5 PSD

Un PSD (“Position Sensitive Device” o “Position Sensitive Detector”) è un sensore di posizione ottico (OPS) , in grado di misurare una posizione di un punto luminoso in una o due dimensioni su una superficie del sensore. I PSD possono essere divisi in due classi che funzionano secondo principi diversi: nella prima classe, i sensori hanno una superficie del sensore isotropo che ha una struttura che fornisce i dati di posizione continua. La seconda classe è composta da sensori distinti sulla superficie del sensore che forniscono dati locali separati.

Sensori non isotropi: se un semiconduttore laminare, un cosiddetto diodo PIN è esposto ad una piccola macchia di luce, questa esposizione provoca un cambiamento nella resistenza locale e pertanto il flusso

di elettroni nei quattro elettrodi. Dalle correnti I_a , I_b , I_c e I_d degli elettrodi , la posizione del punto luce è calcolata utilizzando le seguenti equazioni.

$$x = k_x \cdot \frac{I_b - I_d}{I_b + I_d} \quad y = k_y \cdot \frac{I_a - I_c}{I_a + I_c}$$

In cui k_x e k_y sono semplici fattori di scala, che consentono la trasformazione in coordinate.
Un vantaggio di questo processo è la misura continua della posizione del punto luminoso con misurazione fino ad oltre 100 kHz. La dipendenza della misura locale sulla forma e dimensioni del punto luminoso e la connessione non lineare è un inconveniente che può essere in parte compensato dalla particolare forma degli elettrodi.
Per quanto concerne i sensori PSD di tipo digitale possiamo avere due tipi di elaborazioni:

- ✓ Seriale: le applicazioni di sensori più comuni con una frequenza di campionamento inferiore a 1000 Hz sono dispositivi CCD o CMOS . Il sensore è suddiviso in singoli pixel il cui valore di esposizione possono essere letti in modo sequenziale . La posizione del punto luce può essere calcolata con i metodi di fotogrammetria direttamente dalla distribuzione della luminosità .
- ✓ Parallela: per le applicazioni più veloci, sono stati sviluppati sensori a matrice con elaborazione in parallelo, dove la densità della luce di ciascun pixel viene confrontata con un valore di soglia globale. I risultati del confronto diventano linee e colonne con logica OR. Successivamente, il punto più luminoso di un dato valore di soglia sarà il valore medio delle coordinate calcolati dello spot luminoso. In quest' ottica si colloca il progetto del nostro sensore di sole.

2.5.1 Image Sensor

Un sensore d'immagine è un dispositivo che converte un'immagine ottica in un segnale elettronico. Viene utilizzato soprattutto nelle fotocamere digitali, moduli per camere e altri dispositivi di imaging. I primi sensori simili erano i canali della videocamera; attualmente in uso, invece, ci sono dispositivi semiconduttori ad accoppiamento di carica (CCD) o dei sensori attivi pixel a complementare metallo-ossido-semiconduttore (CMOS) o metallo-ossido-semiconduttore tecnologie di tipo (NMOS , Live MOS) .
Entrando nel dettaglio, l' oggetto della tesi è la realizzazione di un sensore di sole che, con le opportune modifiche, può anche essere utilizzato come sensore di terra; perciò d' ora in poi, analisi e deduzioni saranno riferite al sensore solare, ma è immediato capire che la stessa cosa può similmente essere fatta per un horizon sensor: a tal proposito, nel capitolo riguardante il software, il nome del progetto verrà poi rinominato semplicemente SUN_HORIZON_SENSOR con un caso d' uso specifico, SET_INTENSITY, per mezzo del quale variando l' intensità lo si può adattare per filtrare anche spot luminosi con luminosità più bassa (esempio:Terra).

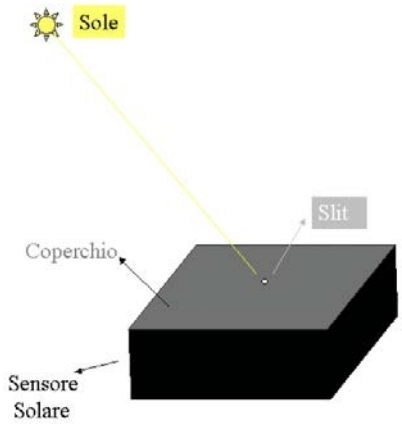


Fig.2.9 Schema del sensore.

Il sensore consta di uno schermo opaco (“coperchio” o “maschera”) sul quale sono praticati uno o più fori: la luce del sole, passando attraverso il/i foro/i, cade sul fondo dell’alloggio (che si trova ad una distanza focale F dal coperchio), dove è posta una matrice di pixel attivi (APS) sensibili alla luce incidente.

Consideriamo inizialmente il caso di un singolo foro circolare:

Il rivelatore APS fornirà in uscita, pixel per pixel, il livello di luminosità incidente. Un processore calcola la posizione del centroide di tale distribuzione rispetto ad un sistema di riferimento avente origine nel punto “CA”, centro dell’APS (sistema di riferimento CA XCA YCA ZCA):

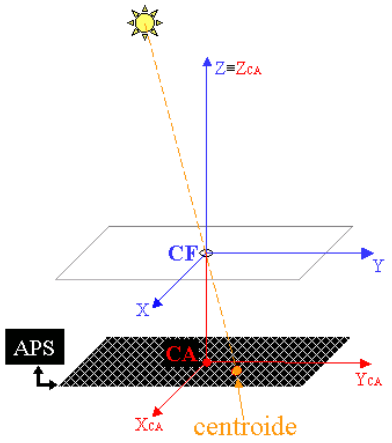


Fig.2.10 Baricentro della distribuzione di luce sull’APS.

Dal calcolo delle coordinate del centroide si può così ricavare una stima degli angoli di azimut (ϕ) e coelevazione (Θ) della line-of-sight del Sole nel sistema di riferimento CFXYZ. Tali angoli sono la vera e propria informazione data in uscita dal sensore:

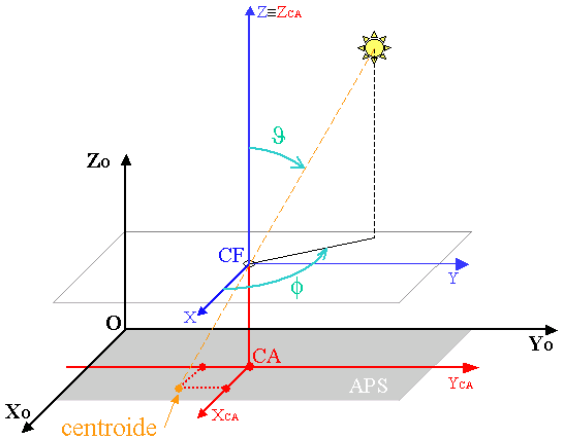


Fig. 2.11 Verso positivo degli angoli di azimut (ϕ) e coelevazione (Θ della line-of-sight del Sole e sistemi di riferimento Sensore (CFXYZ), APS (OXOYOZO) e Centro APS (CA XCA YCA ZCA)

La distanza “focale” F tra coperchio e APS viene invece scelta a partire dal campo di vista FOV che si desidera ottenere. Scelto difatti il campo di vista desiderato per il sensore, si passa a determinare la distanza F tale che, quando il Sole è al limite del FOV, il suo “spot” non cada fuori dell’APS. Con riferimento alla figura 2.12, si nota infatti che al crescere di F- a parità di tutte le altre grandezze geometriche- lo spot tenderebbe ad uscire dall’APS.

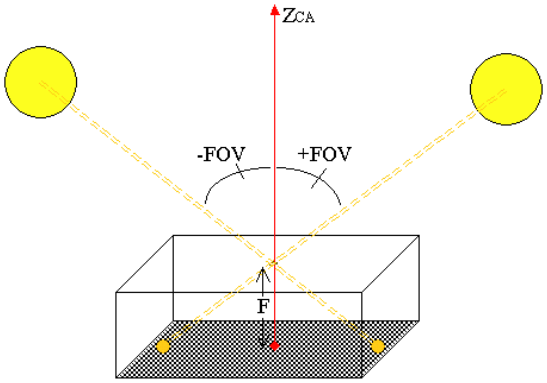


Fig 2.12 Campo di vista FOV del sensore e distanza focale F.

È necessario allora calcolare i fattori di vista di ogni pixel rispetto al sole. Innanzitutto, chiariamo cosa si intende per “cono di vista” di un pixel:

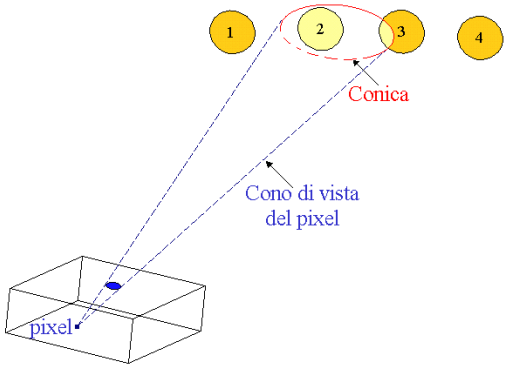


Fig.2.13 Cono di vista del pixel.

Il pixel è illuminato da qualunque sorgente di luce si trovi all'interno di un ipotetico cono avente vertice nel centro del pixel e per direttrice il bordo del foro del coperchio. Chiamerò, allora, questo cono "cono di vista" del pixel.

Supponiamo che l'unica fonte luminosa sia il Sole. Come si evince dalla figura

2.13, sono possibili tre casi:

- Il pixel è in ombra (il Sole non appartiene al suo cono di vista: Sole in posizione 1 o 4);
- Il pixel vede tutto il Sole (posizione 2);
- Il pixel è illuminato da una frazione del disco solare (posizione 3).

2.6 Breve progettazione sulla struttura meccanica

Questo è un breve accenno sulle misure fisiche da considerare per la realizzazione completa del sensore. Innanzitutto riferendoci al CMOS image sensor dell' Aptima (MT9V034), la cui scelta verrà discussa nel capitolo successivo, ed in particolare saranno importanti le seguenti caratteristiche, avremo:

- Active imager size: 4,51 mm (H) x 2.88 mm (V) con diagonale 5,35 mm;
- Pixel size: 6.0 x 6.0 µm.

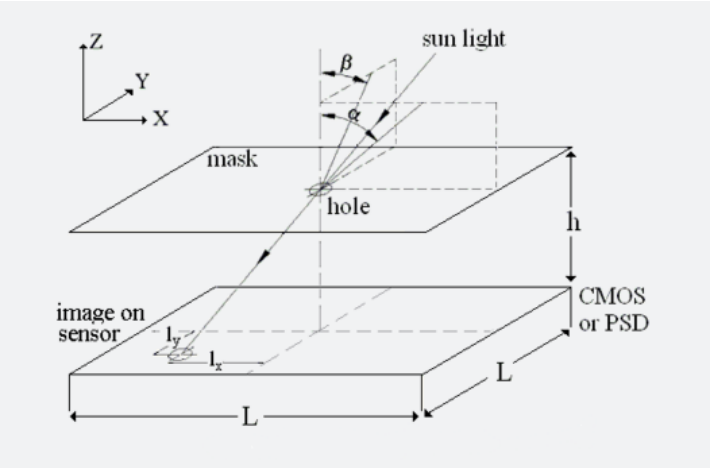


Fig. 2.14 Struttura fisica.

Il primo step è trovare la distanza (*h*) dal pinhole al sensore. Cioè in funzione delle dimensioni delle larghezza e lunghezza della Wide-VGA e del massimo angolo di incidenza.

Perciò avendo area effettiva d' immagine di dimensioni 4,51 mm x 2,88 mm, prendiamo quella più corta e 50° come massimo angolo d' incidenza ed abbiamo:

$$h = \frac{l_2}{\tan\theta} = \frac{2{,}88\ mm}{\tan 50°} = 2{,}42\,mm$$

Ora, usando la legge di Rayleigh, abbiamo::

$$d = 1{,}9\sqrt{h \cdot \lambda} = 80\,\mu m$$

Dove *d* è il diametro del pinhole e λ la lunghezza d'onda (≈ 750 nm). Con questo valore, il diametro del pinhole scelto è 100 μm, cosìcchè la distanza tra il pinhole e il sensore dovrebbe essere:

$$h = \frac{1}{\lambda} \cdot \left(\frac{d}{1{,}9}\right)^2 \approx 2{,}4 \text{ mm}$$

Nellla figura 2.15 sono riportate le misure e il progetto fisico in generale.

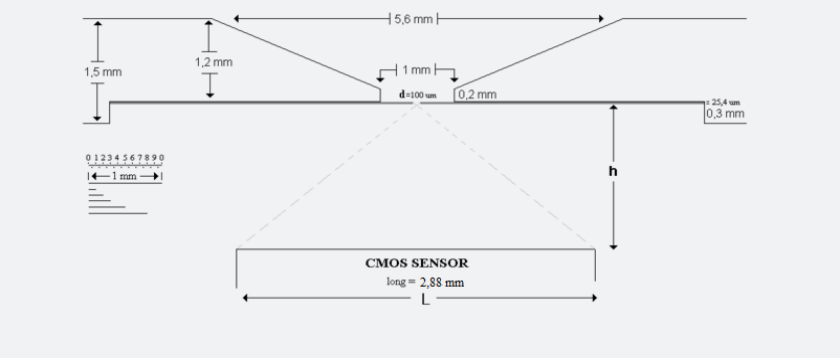


Fig. 2.15 Struttura opto-meccanica del sensore – pinhole.

Il pixel size è un dato da tenere presente nel momento in cui si vuole capire con quanta accuratezza si riesce a misurare l' angolo prodotto dallo spot luminoso; in realtà, ipotizzando che vengano illuminati una decina di pixels sull' array, è immediato capire che più la dimensione sarà piccola, maggiore sarà la risoluzione fornita Δθ.

2.6.1 Attenuazione della luce dal sole al sensore

CMOS Sensor MT9V034 ha 3 parametri::

- Sensitività: 4,8 V / lux·s
- Tensione di riferimento per ADC: 1.4 V

La costante solare è la quantità d' energia ricevuta al top dell' atmosfera terrestre sula superficie orientate perpendicolarmente ai raggi del Sole (alla principale distanza della Terra dal Sole). Generalmente la costante solare accettata è 1368 W/m^2

Così per calcolare il numero di lux sulla superficie terrestre usiamo la seguente equazione:

$$Ilum\ at\ earth = \frac{sun\ ilum}{sun\ energy\ earth} \cdot solar\ const$$

$$Ilum\ at\ earth = \frac{130000\ lux}{1000\ W/m^2} \cdot 1368\ W/m^2 = 177840\ lux$$

Ora è possibile usare questo valore per calcolare il tempo minimo per portare alla saturazione il sensore, cioè:

$$Time\ Sat = \frac{V_{ADC}}{Sensor\ Sensitivity \cdot Ilum\ \ earth}$$

$$Time\ Sat = \frac{1{,}4\ \mathrm{V}}{4{,}8\ V/lux \cdot s\ \times 177840\ lux} = \quad 1{,}64\ \ \mu s$$

Usando una frequenza master di clock di 26 MHz per il sensore, con massima risoluzione del frame il tempo di integrazione per ogni frame (immagine) è approssimativamente 16,66 ms. E' il tempo per avere il Massimo valore (saturazione), così la relazione tra I due tempi è il fattore di attenuazione della luce.

$$Attenuation\ Factor = \frac{16{,}66\ ms}{1{,}64\ \ \mu s} = \ 10158{,}54$$

La densità di potenza rispetto all'area della radiazione del sole che raggiunge il sensore dipende da molti fattori ambientali: è tipicamente di 1500 W/m^2. Si necessita di un filtro di attenuazione che porti la quantità dei *pixels* ad un livello ragionevole, altrimenti si rischierebbe la saturazione costantemente.
Il rosso (fotone) penetra più profondamente nel pixel del silicio e può introdurre problemi di *extra cross-talk* tra i *pixels*, quindi si ha una riduzione dell'accuratezza. Per limitare l'arrivo della luce al verde (o meglio ancora al blu) viene usato questo filtro. Quindi applicando il logaritmo per ricavare la Densità Ottica (OD) del filtro abbiamo:

$$OD = \log_{10}10158{,}54 = 4$$

Perciò, la OD richiesta è 4 e per ottenerla useremo un filtro a densità neutrale con OD = 3 e un altro filtro a densità neutrale con OD = 1.
Adesso includendo i due filtri ottici nell' assemblaggio fisico, abbiamo ora il nuovo progetto:

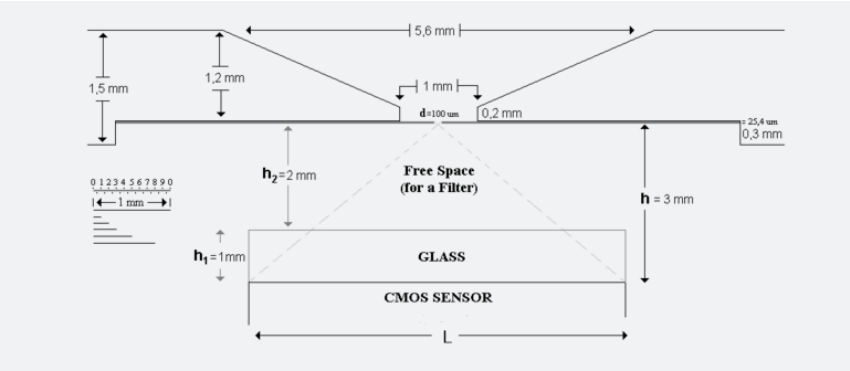


Fig. 2.16 Struttura opto-meccanica del sensore - lenti

Gli spessori dei filtri sono 1 mm per assorbimento (OD = 1), 3.0 mm per assorbimento (OD = 3) e 1,6 mm per la riflessione (OD = 3,0).
Ciò indica che da specifiche (con i parametri reali) il massimo angolo dovrebbe essere inferiore oppure possiamo ricambiare il progetto del modello fisico e riparare i filtri.

Capitolo 3

Progetto Generale

3.1 Specifiche

La scheda progettata si occupa di gestire il sensore presente al fine di ottenere le coordinate angolari (angolo α e β) che a loro volta permettono di determinare l' assetto del satellite. Il progetto è partito dalle seguenti specifiche di base:

- Acquisire un intero fotogramma dal sensore d' immagine presente sul satellite.
- Salvarlo in una memoria alla fine del processo di acquisizione.
- Estrarre le coordinate angolari attraverso un algoritmo ad-hoc del processore.
- Immagazzinare i dati a 16 bit nella Flash integrata nel processore e, quando richiesto, utilizzarli direttamente dallo stesso image processor (o attraverso un altro tile processor) per il controllo dell' assetto;

La scheda sarà totalmente comandata mediante diversi telecomandi codificati dall' OBC attraverso dei canali di comunicazione distinti (slots OBC_A, OBC_B in 1B48 (si veda il capitolo sulla progettazione software) e dalla scheda Power Management che fornirà le alimentazioni necessarie al funzionamento dei componenti della scheda.

Al fine di soddisfare tale specifiche si è partiti con un progetto di massima, ipotizzando un processore centrale affiancato da una memoria su cui memorizzare l'immagine e il sensore dell' Aptina. Il DSP avrà inoltre una coppia di segnali general purpose utili ad impostare i registri del trasduttore in modo tale che quest' ultimo sia poi in grado di funzionare garantendo la corretta trasmissione dei bits; a tal proposito, visto che appunto l' output è già un segnale digitale non si necessita di nessun video decoder.

Nella successiva figura 3.1, si può vedere in linea di massima, lo schema a blocchi che descrive lo schema di principio della scheda: da una parte abbiamo la CPU che, fungendo naturalmente da master nel nostro sistema, viene alimentata e si interfaccia con gli altri moduli; dall' altra il sensore vero e proprio, che da slave converte il fotone in corrente attraverso la matrice di pixels e li fornisce al processore. Naturalmente il tutto dovrà essere corredato da un set di sensori (tensione e corrente) per monitorare le grandezze elettriche costantemente e in ultimo, non meno importante, dei load switch in grado di isolare le diverse parti se dovesse accadere qualche malfunzionamento isolato. Quest' ultimi saranno trattati approfonditamente nei relativi paragrafi.

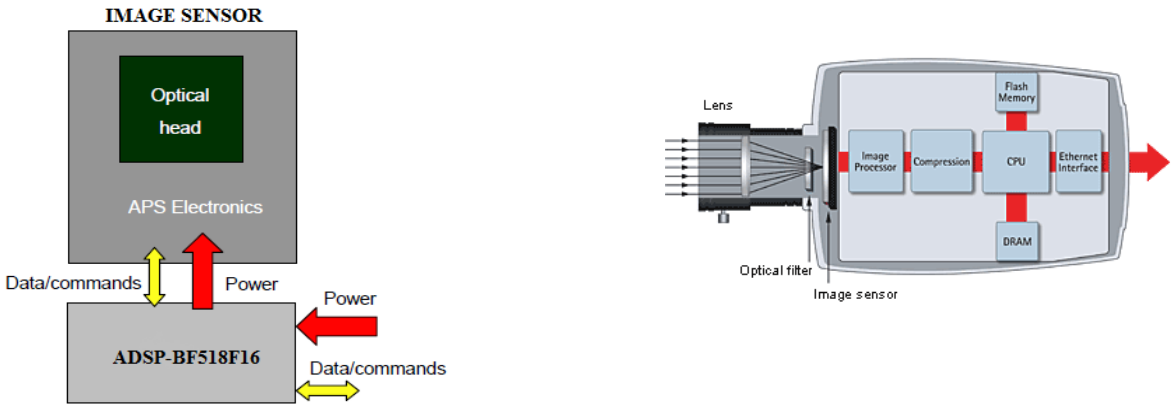


Figura: 3.1 Schema a blocchi specifiche scheda da due diversi punti di vista

3.2 Scelta dei componenti

3.2.1 Criteri di scelta dei componenti

Il progetto si è basato dapprima su alcune considerazioni sull'ambiente in cui lo stesso avrebbe dovuto funzionare, ossia logicamente nello spazio, su di un satellite di dimensioni ridotte quindi in un sistema alimentato da batterie caricate mediante pannelli solari; per questo, il consumo energetico della scheda è il fattore critico principale che sarà tenuto in considerazione per tutto l' arco della realizzazione. Legato ad esso vi è infatti un forte interesse a non far surriscaldare troppo i componenti durante il loro funzionamento, dato che l'unica modalità di dissipazione del calore a 600 Km di altitudine è l'irraggiamento e la conduzione dei piani di alimentazione, quindi in definitiva una bassa efficienza dissipativa.

Altro fattore importante per alcune scelte di progetto è stato il possibile utilizzo dei componenti di tecnologia CMOS, con particelle pesanti presenti nello spazio esterno alla nostra atmosfera, creando la possibilità di effetti di latchup nei circuiti CMOS stessi. Tale effetto può condizionare sia i dati memorizzati nelle memorie, sia le correnti erogate da regolatori di tensione, sia i valori di alcuni MOS integrati utilizzati come switch per alimentazioni come il load switch. Da questo punto di vista inoltre, in fase di test necessiterebbe di un accurato calcolo dei consumi con correnti massime nella scheda nelle varie fasi di funzionamento, al fine di poter permettere ai progettisti di realizzare dei circuiti che stacchino l'alimentazione in presenza di picchi di assorbimento di corrente, denominati circuiti anti-latchup, adeguati al normale assorbimento della scheda.

Un'ulteriore caratteristica di sistema che ha caratterizzato la scelta dei componenti riguarda anche la scheda Power Management, la quale fornisce l'alimentazione al sistema tramite le corrispettive slots.

Quando la scheda contenente il sensore e l' image processor (che a sua volta contiene la SDRAM) deve svolgere qualche operazione essa viene accesa interamente, ma non userà mai tutti i circuiti integrati presenti su di essa contemporaneamente, quindi è importante che essi presentino una modalità di Power Save (o Standby) con un bassissimo consumo di potenza, affinché l'integrale nel tempo dell'energia assorbita dal componente quando non è utilizzato sia molto basso in modo che possa rimanere accesa quasi un minuto ininterrottamente (da specifiche di sistema). Nella scheda quindi sono state studiate delle soluzioni affinché ogni componente non utilizzato venisse portato in Power Save Mode.
Infine anche il lato economico ha avuto un certo peso nelle scelte effettuate, favorendo la scelta di prodotti disponibili sotto forma di samples gratuiti in quantità adeguate, in modo da permettere l'utilizzo di soli samples per la prototipazione e le schede definitive.

3.2.2 Sensore d' immagine

Il sensore da utilizzare è stato scelto svolgendo dapprima delle considerazioni sulla sua tecnologia cercando di arrivare ad una decisione vagliando pro e contro. Esistono in commercio due grandi diverse famiglie:

- Sensori CMOS (Copper Metal Oxide Semiconductor)
- Sensori CCD (Charge Coupled Devices)

I sensori CCD utilizzano una tecnologia specificatamente sviluppata per le telecamere, mentre i sensori CMOS impiegano la tecnologia standard normalmente usata per i chip di memoria, come quelli installati nei PC.

<u>Tecnologia CCD</u>
I sensori CCD, utilizzati per le telecamere da oltre vent'anni, offrono molti vantaggi in termini di qualità, tra cui una maggiore sensibilità alla luce rispetto ai sensori CMOS. Questa maggiore sensibilità alla luce si traduce in immagini di migliore qualità anche in condizioni di scarsa illuminazione. Tuttavia, i sensori CCD presentano un costo più elevato poiché la loro integrazione nelle telecamere richiede operazioni complesse e laboriose. Inoltre, se la scena contiene un oggetto molto luminoso (ad esempio un lampo o la luce diretta del sole), il sensore CCD non è in grado di acquisire correttamente le immagini e su di essa sono spesso visibili strisce verticali sopra e sotto l'oggetto. Questo fenomeno viene chiamato distorsione a striscia verticale di luce.

<u>Tecnologia CMOS</u>

Grazie ai recenti sviluppi, i sensori CMOS sono ora in grado di offrire immagini di qualità equivalente a quella dei sensori CCD, ma sono comunque inadatti alle telecamere che devono generare immagini di altissima qualità. I sensori CMOS riducono significativamente il costo della telecamera e possono essere utilizzati per telecamere di piccole dimensioni. Questo tipo di sensori è però disponibile anche in formati più grandi, che forniscono una risoluzione in megapixel a numerose telecamere di rete. Uno dei limiti più significativi dei sensori CMOS deriva dalla loro minore sensibilità alla luce, che non rappresenta un problema in condizioni di normale illuminazione ma può diventarlo se la luce è scarsa. Le immagini generate da questi sensori in condizioni di scarsa illuminazione possono essere infatti molto scure o disturbate.

A titolo d' esempio, per meglio capire le due tecnologie, confronteremo le quattro operazioni fondamentali necessarie alla rivelazione delle immagini:

1-Generazione della carica

2-Separazione e raccolta della carica

3-Trasferimento della carica

4-Misura della carica

1)Generazione della carica:

La capacità del sensore di assorbire un fotone e di generare una coppia elettrone-lacuna per foto generazione è indicata con il termine: Efficienza Quantica (Q.E .). Un sensore ideale dovrebbe avere una Q.E. costante e pari al 100% per ogni lunghezza d'onda. Ciò significherebbe che ogni fotone incidente la superficie darebbe luogo alla generazione di una coppia libera elettrone-lacuna, ma com'è facilmente intuibile, ciò in realtà non avviene.

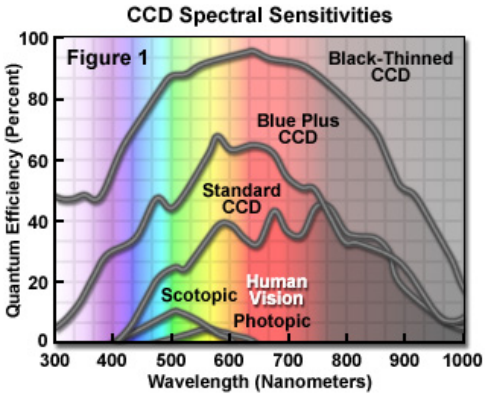


Figura: 3.2 Efficienza quantica in funzione della lunghezza d' onda

Le cause che degradano la Q.E. sono le perdite per assorbimento, riflessione e trasmissione. Gli array di CMOS manifestano una più alta perdita per assorbimento rispetto ai CCD visto che incorporano nella struttura fotosensibile i circuiti di uscita che sono otticamente inerti.

Ciò riduce il Fill Factor(F.F.), ovvero l'area utile fotosensibile. I CCD, data la struttura, hanno un meccanismo di lettura che non necessita di circuiteria di trasporto delle cariche e quindi un F.F. intrinsecamente pari al 100%.

Nei CMOS viene combattuta la riduzione del F.F. con l'utilizzo di microlenti che collimano il fascio luminoso nelle zone di massimo assorbimento. Ciò, però, comporta la dipendenza della Q.E. dall'angolo di incidenza del fotone e dalla sua lunghezza d'onda. Nei CMOS e CCD "back side illuminated" , l'illuminazione incide sul dispositivo in una zona attiva posta nel substrato comportando un aumento della superficie utile per la foto generazione. Il vantaggio di questa tecnica è stato immediatamente utile per i CCD visto che, unitamente all'utilizzo di opportuni strati antiriflesso, ha portato ad un aumento della Q.E. fino al 90% rispetto alla bassa efficienza dei CMOS che resta del 60% alla lunghezza d'onda di picco.

Entrambi i dispositivi manifestano una forte dipendenza, nell'operazione di generazione della carica, dalla temperatura. Infatti, con l'aumento della temperatura di esercizio aumentano le cariche generate per agitazione termica. Questo fenomeno che si manifesta anche in assenza di radiazione incidente, ovvero di segnale utile, è detto dark current.

2)Separazione e memorizzazione della carica:

La seconda operazione eseguita da un generico sensore di immagine è la separazione delle coppie generate e la ritenzione della carica. I fattori che governano questo processo sono i seguenti:

1. Numero e dimensione dei pixel;
2. Numero massimo di cariche accumulabili o Full Well (FW);
3. La variazione di sensibilità tra pixel o Fixed Pattern Noise (FPN);
4. La capacità di non disperdere le cariche catturate all'interno della struttura o Charge Collection Efficiency (CCE).

Il numero di pixel attualmente disponibile è paragonabile per i due dispositivi e varia dai 128 ai 4048, con dimensioni del pixel che arrivano a 5 μm x 5 μm. I CMOS hanno una FW più profonda dei CCD che riescono ad accumulare fino a 106 elettroni. La FPN è causata dalla non perfetta uguaglianza nelle caratteristiche geometriche e costruttive dei pixel che è causa, ad esempio, dei problemi noti come "Hot Pixel " e "Cold Pixel ".

Entrambe le tecnologie manifestano una FPN paragonabile e che tipicamente tecniche software o hardware, in fase di misura, compensano quasi del tutto previa opportuna calibrazione. L'ultimo dei quattro parametri, la CCE, risulta essere critico e sfavorevole per i CMOS.

Le cariche accumulate, a causa della diffusione termica, possono disperdersi interagendo con le cariche del vicino pixel e quindi provocando il cosiddetto "pixel crosstalk ". I CCD hanno eccellentemente risolto questo problema utilizzando substrati di wafer ad alta resistenza e tensioni di pilotaggio elevate (anche 12 V).

I CMOS, invece, utilizzano basse tensioni di alimentazione e una tecnologia di fabbricazione del tutto identica a quella utilizzata per la produzione di memorie, relativamente giovane rispetto alla trentennale tecnica produttiva dei sensori CCD, nata ed ottimizzata per questo tipo di dispositivo. La profondità della "pozza di carica" rende l'immagine acquisita dai CMOS più profonda, rendendo più difficile la saturazione.
Nei CCD che hanno una FW più bassa si richiede la presenza dell'anti-blooming system che, con una circuiteria adatta, evita la saturazione ma limita la QE poiché tale circuiteria utilizza spazio utile per la fotogenerazione con conseguente diminuzione del FF .

3)Trasferimento della carica:
Nei CCD la carica è trasferita, in maniera seriale, da un pixel al successivo, con un meccanismo logico simile a quello utilizzato dai registri a scorrimento elettronici, fino a che è consegnata all'amplificatore di lettura esterno dallo shift register. Il parametro che caratterizza l'efficienza di questo trasferimento è la Charge Transfer Efficiency che in alcuni CCD per uso tecnico-scientifico, raggiunge il 99.9999% per singolo pixel trasferito.
Il meccanismo di trasferimento fa sì che l'ultimo pixel rispetto al primo, risenta maggiormente delle perdite di carica trasferita che, comunque, data l'efficienza del metodo, risultano trascurabili in molte applicazioni.
Al contrario i CMOS indirizzano direttamente vero l'amplificatore di uscita il valore della carica dopo averla amplificata con uno stadio intermedio. La diversità di questi amplificatori (guadagni non esattamente uguali tra loro), è il motivo che avvantaggia i CCD nelle applicazioni ad alta sensibilità.

4)Misura della carica:
Quest'ultimo aspetto sfavorisce nettamente, allo stato attuale della tecnologia, i sensori CMOS. Entrambe le tipologie di sensori utilizzano un amplificatore connesso con un condensatore per consegnare il segnale in uscita. Il risultato di avere un basso rumore di lettura si ottiene non solo diminuendo le dimensioni del condensatore in uscita, ma anche migliorando l'elettronica che elabora il segnale.
In realtà molti dei CMOS in commercio lavorano con amplificatori di uscita non retroazionati e ciò comporta un aumento del rumore di lettura. Un indicatore comunemente utilizzato per quantizzare l'entità del rumore di misura della carica foto generata è il dynamic range che rappresenta, in valori naturali o in decibels, il rapporto tra il massimo numero di cariche accumulabili, cioè la dimensione della FW, e il rumore di lettura.

L’informazione sul rapporto segnale rumore che si evince dal dynamic range è un utile indicatore della bontà del dispositivo ed è importante anche in sede di progetto degli apparati di lettura visto che può aiutare a scegliere, ad esempio, il giusto livello di quantizzazione dell’uscita.

Un altro problema relativo alla fase di misura e particolarmente sentito dai CMOS è causato dal non perfetto reset del condensatore di uscita che, alla successiva lettura, inficia il nuovo valore da rilevare. Il problema viene oggi risolto con la tecnica del Correlated Double Sample (CDS), indifferentemente applicata sia ai CMOS che ai CCD.

Questa tecnica consiste nel campionare l’uscita subito dopo il reset per poi sottrarre il valore parassita dal segnale utile. Per i sensori CMOS-APS si utilizza anche la tecnica dell’Active Reset che fa uso di un’apposita circuiteria per il reset della capacità di uscita.

Riassumendo in una tabella il confronto delle due tecnologie abbiamo:

Caratteristiche	**CCD**	**CMOS**
Segnale d’ uscita dal pixel	Pacchetto di elettroni	In tensione
Segnale d’ uscita dal chip	In tensione (analogico)	Bits (digitale)
Segnale d’ uscita dalla telecamera	Bits (digitale)	Bits (digitale)
Fill factor	Alto	Mediocre
Disadattamento dell’ amplificatore	Non disponibile	Mediocre
Rumore del sistema	Basso	Mediocre
Complessità del sistema	Alto	Basso
Complessità del sensore	Basso	Alto
Componenti della telecamera	Sensore + chips per molteplici supporti + lente	Sensore + possibile lente da aggiungere al chip di supporto
Costi relativi per ricerca e sviluppo	Più basso	Più alto
Costi relativi del sistema	Dipende dall’ applicazione	Dipende dall’ applicazione
Responsività	Moderate	Leggermente migliore
Range dinamico	Alto	Mediocre
Uniformità	Alto	Da basso a mediocre
Uniform Shuttering	Veloce	Lento
Uniformità	Alto	Da basso a mediocre
Velocità	Da mediocre ad alto	Più alto
Windowing	Limitato	Ampio
Antiblooming	Da alto a nullo	Alto
Biasing and Clocking	Diverse tensioni, ma più alte	Singola bassa tensione

Consumi ridotti di potenza	Continui miglioramenti nella tecnologia CCDs	Vantaggiosa per i CMOS, ma margini ridotti
Dimensione della ridotta area attiva del sottosistema	La parte ottica, i vari chips e l' impacchettamento sono spesso i fattori dominanti	CCDs e CMOS confrontabili

Tabella 3.3: confronto delle due diverse tecnologie

Detto questo, la conclusione a cui si è arrivati è che ciascuno dei sensori è ben lontano dal soppiantare del tutto l'altro e anzi si stima che per parecchio tempo le due tecnologie saranno complementari e utilizzate all'occorrenza in base all'applicazione. Per questo motivo i sensori di tipo CCD rispetto ai sensori CMOS sono molto più sensibili, ossia forniscono alla videocamera la capacità di fornire immagini di buona qualità anche a basse condizioni di luce. Questo inoltre è anche associato ad un miglior rapporto segnale/rumore rispetto ai sensori CMOS, i quali a causa delle correnti di leakage presentano una maggiore interferenza di base, che diventa subito importante a bassi valori di illuminazione. Inoltre la risposta di un CCD è indipendente dall'intensità luminosa incidente; infatti, i sensori CCD sono dei rivelatori perfettamente lineari (la linearità è di solito migliore dello 0.01%). In pratica ciò significa che il numero di elettroni generati in un pixel è direttamente proporzionale alla quantità di luce incidente. Tutto questo comporta quindi una maggiore dinamica del sensore CCD.
I vantaggi tipici del sensore CMOS è il prezzo contenuto dato che sono prodotti nelle stesse fonderie da produttori di circuiti integrati, il cui numero prodotto fa quindi scendere notevolmente il costo; inoltre il sensore CMOS presenta un basso assorbimento di potenza rispetto a sensori CCD.

Questi due parametri sono due dei criteri fondamentali per le nostre scelte progettuali: il primo per filtrare i sensori secondo un limitato budget a nostra disposizione, il secondo per limitare al minimo i consumi in un ottica di più larga veduta che vede come target il risparmio di potenza in un ambiente come quello satellitare. Scendendo poi nel dettaglio e confrontando la scelta sulla base della realizzazione del sensore di sole, si può aggiungere che il nostro scopo non è registrare uno stream video di lunga durata, che quindi significherebbe un notevole consumo di energia ma consiste nell'acquisizione di un solo fotogramma (dalla durata relativamente breve di circa 20ms), e quindi sebbene il consumo di potenza istantanea di sensore d' immagine basato su tecnologia CCD sia importante, l'energia consumata è comunque molto piccola dato che il sistema viene acceso e poi spento solo limitatamente alle acquisizioni delle immagini.
Per tutta queste serie di motivi, si è selezionato un sensore d' immagine CMOS, il cui schema a blocchi è visibile nella tabella sottostante:

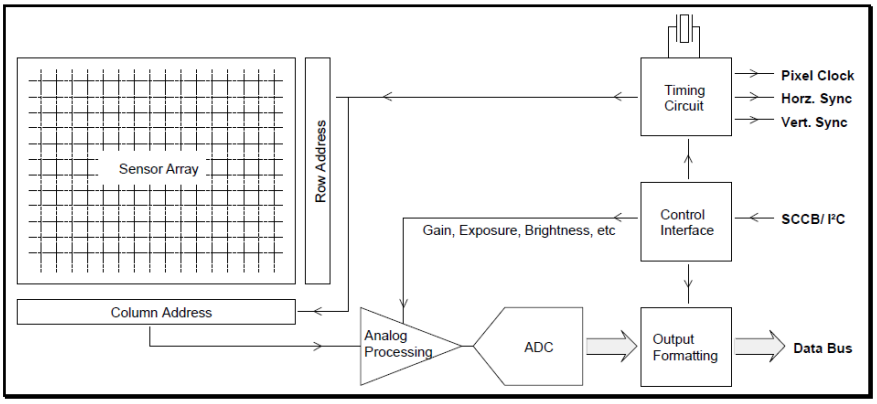


Fig. 3.4 Schema a blocchi di un tipico sensore CMOS

Per essere precisi, il progetto prevedeva l'utilizzo di un sensore d' immagine prodotto dalla ON Semiconductor, un NOIL1SM0300-QDC, ma non reperibile in tempi brevi e mediante canali diretti col Politecnico di Torino a causa di un lead time di 28 settimane; si è dunque preferito avvalersi di un MT9V034C12STM, prodotto dalla Aptina Imaging, con prestazioni inferiori ma perfettamente utile per il suo impiego in fase prototipale in quanto capace di soddisfare le stringenti esigente progettuali a costi contenuti e ridotti consumi.

3.2.3 Processore

La scelta del processore è stata fondamentale e sicuramente non ultima in ordine di importanza, ma seconda dopo quella del sensore. Prima di entrare in merito alla scelta del modello, si sono prese in rassegna quelle che sostanzialmente dovevano essere le peculiarità di base che esso doveva racchiudere secondo appunto le esigenze del sistema stesso. Innanzitutto, doveva soddisfare le tempistiche: come detto in precedenza, il sensore invia i dati al processore in modalità parallela ad una frequenza di circa 26 MHz o meglio 10 bit ogni 37ns di pixel clock. Per ottemperare a tale requisito, si ha perciò bisogno di un processore che abbia un core che lavori ad alte prestazioni, quindi si è stati indirizzati verso i digital signal processor (DSP), le cui frequenze di lavoro permettono di gestire ed elaborare notevoli quantità di bit.

Detto questo, però non ci si deve dimenticare che lavoriamo in un ambiente dove il consumo di potenza è un elemento per niente da sottovalutare perché se da una parte si ha bisogno di una discreta frequenza, dall' altra non possiamo eccedere nelle prestazioni dell' unità di calcolo che potrebbero andare fuori portata da parte del satellite. In commercio, ci si può più facilmente affidare a microcontrollori dotati di un' interfaccia in grado di gestire direttamente il sensore, ossia in grado di catturare il frame realizzando quell' operazione che viene chiamata "frame grabbing": sul mercato tutte le più importanti case costruttrici propongono almeno una soluzione in grado di comunicare direttamente con il sensore d' immagine. Solo per citarne qualcuna esistono i processori PXA270 e PXA320 della Marvell, la famiglia

Da Vinci della Texas Instruments con il TMS3200M642 e l' Atmel con la serie AT91SAM9 che, al contrario delle prime due, ha a disposizione una periferica integrata chiamata ISI (Imaging Sensor Interface), in grado di gestire un sensore fino a 4 megapixel con dati a 8 bit per sensori a colori e fino a 12 bit per quelli monocromatici ottenendo anche una preview dell' immagine grazie a meccanismi di ridimensionamento e una conversione automatica dai formati YCrCb e YUV verso l' RGB gestito dagli LCD.

In generale, l' elaborazione dell' immagine è una sfida per ogni processore: centinaia di Kb di dati devono essere processati in pochi ms: molti algoritmi spesso accedono non solo al pixel che si sta elaborando ma anche ai pixels vicini.
Come si nota dallo schema a blocchi in figura 3.1, il processore sarà il cuore di tutte le funzioni svolte dalla scheda e dovrà integrare varie funzionalità sia di tipo standard, come porte di comunicazione hardware standard (SPI, I2C, UART..) e almeno un controller per memorie non volatili , per mezzo del quale si avvierà il programma per il corretto funzionamento della scheda nonché per avviare l' algoritmo e salvataggio degli angoli, sia più specifiche per applicazioni video, come una porta per l'acquisizione di stream video nel formato del sensore. Inoltre come per ogni altro componente, il processore dovrà essere comunque contenuto in dimensioni e consumi e nello stesso tempo essere adeguato alla funzionalità di calcolo principale per cui sarebbe stato usato: il calcolo del baricentro dello spot luminoso.
A tutto questo per avere una maggiore accuratezza nel calcolo dell' angolo, si devono aggiungere però delle tecniche che appesantiscono il computo del processore come l' utilizzo di una ROI (region of interest) più piccola, e quindi la gestione di meno bits da una parte ma delle pre-scannerizzazioni per individuarla. Oppure l' impiego di molteplici pinhole che garantiscono una migliore risoluzione attraverso un calcolo del baricentro più complicato. Queste tecniche avanzate, appena citate, sono lasciate per successivi sviluppi del sun sensor.
Dato che nei controllori o processori general-purpose, l'operazione di moltiplicazione richiede un elevato costo computazionale, risulterebbe più adatto un processore progettato per l'elaborazione numerica, chiamato DSP (Digital Signal Processor), dove la sua unità di calcolo ALU (Arithmetic Logic Unit) contiene una unità funzionale chiamata MAC (Multiply Accumulator), predisposta ad eseguire moltiplicazioni e addizioni in contemporanea, abbassando notevolmente il costo stesso. A volte per venire incontro a questi processi sui dati e sull' enorme occupazione del bus che trasferirà i dati in memoria è necessario un data path aggiuntivo per l' acquisizione dell' immagine e per prevenire anche i coni di bottiglia. Volendo evitare costi addizionali aggiuntivi, dobbiamo essere capaci di leggere e memorizzare ciascun pixel dal sensore in tempo: infatti, se ci riconduciamo alle frequenze con le quali si opera normalmente, ci accorgiamo che il sensore produrrà 10 bits sull' uscita ogni 40 ns all' incirca , velocità che proibisce un acquisizione dati gestita da interrupts. Si necessita dunque di una PPI prima e di una DMA poi che saranno sincronizzate con il pixel clock del sensore (per esempio ad ogni impulso

di clock si avrà un trasferimento in memoria gestito dallo stesso controller). La maggior parte dei dispositivi di comunicazione hanno un interfaccia con bus lento: una tipica UART richiede un tempo d' accesso di 100ns o più, lo stesso tempo è impiegato per interfacciarsi ai controllers di rete (ethernet o CAN). Se l' interfaccia del DSP è bloccata dalla lettura/scrittura con un dispositivo di comunicazione del genere, non è possibile acquisire un' immagine dal sensore in tempo reale senza perdite di dati. Perciò se non si riesce ad avere un bus separato per l' acquisizione, bisogna avere una memoria FIFO necessaria per *bufferare* i dati dell' immagine o le comunicazioni devono essere limitate al gap inter-frame.

Sebbene questi processori siano estremamente veloci, la codifica non offrirà sempre i risultati (sulle prestazioni) voluti: i compilatori di oggi sono altamente efficienti e ottimizzati, ma l' elaborazione del codice che sfrutta a pieno le capacità del µP, non offrono ancora importanti performance specialmente sugli algoritmi di elaborazione dell' immagine. Perciò una libreria di funzioni che ottimizzano l' elaborazione dell' immagine di base è d' importanza a prescindere, per velocizzare lo sviluppo e ridurre i requisiti hardware. Simultaneamente lettura e memorizzazione in registri intermedi di multipli pixels riducono i requisiti di avere una larghezza di banda della memoria.

Partendo da questi presupposti, si è scelto la soluzione Blackfin prodotto da Analog Devices, che rappresenta un compromesso tra elevate performance e ridotti consumi senza trascurare la possibilità di un' interfaccia diretta con il sensore attraverso la PPI e una conseguente elaborazione del flusso dei dati video se necessario; inoltre, vari tipi di DMA permettono il trasferimento dei dati dalla porta alla memoria sia per un salvataggio temporaneo, nel qual caso si aspetti più di un fotogramma o permanente in attesa che venga elaborata attraverso gli algoritmi del DSP. Oltre a tutto ciò, si è cercato di rispettare il principio di modularità che vede alla base dell' organizzazione del satellite un insieme di sottosistemi gestiti da uno o più processori: nel dettaglio, si deve tenere presente che il processore può anche interfacciarsi con altri moduli come può essere per esempio un sensore di temperatura e quindi necessitare di un convertitore, oppure con un magnetometro e comandare una bobina che ne cambi l' assetto. Sarebbero molti i dispositivi presenti su ogni tile del nostro spacecraft e la cosa più importante è non rendere il nostro DSP utile solo ai fini del nostro sistema, ma disponibile ad accogliere e gestire dati su ogni periferica.

La famiglia Blackfin offre una serie di processori embedded a 16/32 bit, offre un software flessibile e scalabile per diverse applicazioni come multi-formato audio, video ed elaborazione dell' immagine. Essi sono stati progettati per venire incontro sia a esigenze di calcolo che vincoli di potenza delle nuove recenti applicazioni. Basati su un Architettura a Micro Segnale (MSA) unitamente sviluppati con Intel Corporation, i processori Blackfin combinano un instruction set a 32 bit RISC e un duale MAC (Multiply Accumulate) a 16 bit che elabora funzionalmente i segnali con attributi di facile uso (che si trovano nei microcontrollori general purpose).

Tra di questi, quello che si è utilizzato, è l' ADSP-BF518F16 che presenta una serie di caratteristiche utili al nostro progetto, ma anche ad un inquadramento più generalizzato nel satellite per via del fatto che possa comunicare con altri moduli che non saranno trattati in questa tesi, ma che è giusto menzionare in una visione d' insieme. Brevemente, focalizzandoci al nostro target, possiamo sfruttare le seguenti potenzialità del processore scelto:

- I2C (Inter Integrated Circuit) nel caso del nostro DSP equivale al TWI (two wire interface) che garantirà la programmazione dei registri del sensore per il suo corretto funzionamento.
- PPI (Parallel peripheral interface) che supporta formati di dati video ITU-R BT.656 a 8 bit offrendo un trasferimento half duplex di 8-10 bit; sarà utile per interfacciarsi direttamente con il sensore in modalità parallela ovviamente, acquisendo i bit dell' immagine catturata.
- DMA (Direct Memory Access) che attraverso il suo controller permette trasferimenti tra periferiche e memorie on chip/off chip o tra le stesse memorie; nel nostro caso gestirà il trasporto dei dati dalla PPI alla memoria SDRAM (tramite EBIU) ed ancora dalla SDRAM alla Flash.
- SPI (serial peripheral interface) che permetterà la comunicazione con dispositivi SPI attraverso i suoi 4 pins; si potranno dunque immagazzinare i dati nella memoria flash interna di 16 Mbit compatibile tramite questo protocollo (si ricordi che la lettera "F" nella sigla del nome ADSP-BF518F16, sta ad indicare il fatto che questa è incorporata nel processore) nella fattispecie il frame catturato dal sensore d' immagine;
- EBIU (external bus interface unit) che insieme alla DMA regola i trasferimenti di dati con le memorie esterne sincrone e asincrone.

Inoltre il Blackfin presenta due porte UART (Universal Asynchronous Receiver Transmitter) oltre ad altrettante seriali generiche veloci bidirezionali che quindi garantiranno la possibilità di avere due porte hardware distinte per avere due canali di comunicazione separati con altri processori sempre con l'obiettivo di avere ridondanza del sistema, nel caso uno dei due andasse perso, ciò non influenzerebbe la comunicazione con l'altro.

Infine il Blackfin essendo un Digital Signal Processor (DSP), è adatto come spiegato in precedenza anche a funzioni di compressione JPEG a cui è anche destinato mediante le sue potenti unità di calcolo che però verranno solo citate non essendo oggetto di questa tesi.

3.2.4 Memorie

Da quanto appena visto nel paragrafo precedente, la scelta della memoria è stato un parametro vincolante anche nella conseguente selezione del processore; tuttavia, per quanto concerne la selezione di un altro processore che non fosse stato l' ADSP-BF518F16 (con memoria integrata), si sarebbero

potute fare delle considerazioni più oculate sul tipo da interfacciare al bus EBIU che ricordo essere il mezzo sul quale transitano i dati verso la memoria esterna in un processore Blackfin.
Prendendo a titolo d' esempio, un semplice filtro 3 x 3, esso richiede nove accessi in memoria per ciascun pixel che si sta filtrando: per implementarlo in tempo reale (25 frames per secondo) con una risoluzione VGA, abbiamo bisogno di una larghezza di banda di 66 MBytes/s. La deduzione è presto fatta e porta a ragionare sul fatto che le memorie esterne sono limitate in velocità. A tal proposito, si è pensato ad una SDRAM capace di immagazzinare l' immagine provvisoriamente
Laddove non avessimo avuto altri vincoli da rispettare, se non dimensione tale da contenere l' immagine (e/o programma sorgente), l' immunità alle radiazioni elettromagnetiche alle quali il satellite è esposto, con la preferenza adottata verso il DSP Blackfin di Analog Devices, avremmo avuto un'ampia libertà di scelta sulle memorie, con due controller distinti, uno per memorie sincrone, quali SDRAM tipo PC100 e PC133, e uno per memorie asincrone, quali SRAM e FLASH.

3.2.4.1 SDRAM

L' utilizzo di una SDRAM o comunque di una memoria che sia abbastanza veloce, è dettata dalla frequenza elevata con la quale vengono letti i dati del sensore e resi disponibili dalla PPI tramite EBIU alla memoria esterna. In pratica, non è plausibile una normale acquisizione diretta con i tempi di scrittura tipici di una Flash e allo stesso modo, è impossibile immagazzinare i dati nella L1 Data SRAM essendo di dimensioni di 64Kbytes. Perciò si è pensato ad una logica di distribuzione che fosse un compromesso tra consumi e velocità
cercando di non sfruttare oltre ogni modo il PAB (bus che collega la SPI Flash). In questo modo, si è previsto l' utilizzo di una memoria SDRAM che seppur di limitate dimensioni (si sarebbe potuta scegliere di 16 Mbyte per un uso più generalizzato), ha un consumo di potenza importante relativamente all' acquisizione e all' elaborazione, e la flash, che verrà discussa dopo, con un consumo contenuto per anche un uso leggermente più generalizzato rivolto all' interfacciamento con gli dispositivi.

OBIETTIVO	MOTIVAZIONI E SCELTE:
DIMENSIONE **(16MByte – 128MByte supportabile)**	DIMENSIONI RIDOTTE PER CONSUMI (refresh su troppe celle non è adeguato), PER SPAZIO (ai fini della nostra tesi conterrà al massimo un paio di fotogrammi non compressi WideVGA 752H x 480V (nel nostro caso uno che occupa all' incirca 721Kbyte di spazio). Inoltre non deve essere troppo piccola in quanto non sarebbe suddivisibile in 4 banchi (ma solo in 2 per < 16Mbit)in quanto se ne vogliono più di due per una questione di risparmio di potenza. **SOLUZIONE COMPROMESSO: dimensione di 16 MByte (128Mb).**
CONSUMO	La scelta della SDRAM in sé per sé già prevede un consumo superiore rispetto a soluzioni asincrone come potrebbero essere SRAM e FLASH; come detto, molto dipende anche dall' aggiunta del clock, ma proprio in quest' ottica, sfruttando il fatto che il DSP si possa interfacciare con le mobile SDRAM (anche chiamate *low power*) potremmo sceglierne una di queste per il nostro scopo. Un problema che però risulterebbe è il fatto che esse hanno solo un package BGA con conseguenti difficoltà nella fase di saldatura. **SOLUZIONE COMPROMESSO: mobile SDRAM (sfruttando extended mode register che permette di ridurre la frequenza di self-refresh quando è in stato di idle (compensazione in temperatura) e il partial-array self refresh (cioè refresh solo di quei banchi che contengono i dati e sono in quel momento sfruttati).**
USO DI UNA MOBILE SDRAM PER SVILUPPI FUTURI	Siccome l' uso di una SDRAM fa aumentare non di poco i consumi nel sistema, bisognerebbe optare per una scelta sulle memorie low-power con le quali la serie ADSP-BF51x può interfacciarsi. In questo caso rappresenta un ruolo importante l' *extended mode register* che è un sottoinsieme del mode register , può essere abilitato dall' EBIU all' accensione. Il registro in questione è inizializzato con questi parametri:

<table>
<tr><td></td><td>

PARTIAL ARRAY SELF-REFRESH (bits A[2 – 0] con A [2] sempre a 0, bits A[1 : 0];
TEMPERATURE COMPENSATED SELF-REFRESH, bits A[4: 3], bits A[3] sempre a ‘1’, bit A[4];
DRIVE STRENGHT CONTROL, bits A[6 – 5] sempre a 0;
bits A[12 – 7] sempre a ‘0’ e bit A[13] sempre a 1.

In questa applicazione dove la SDRAM è usata raramente stando la maggior parte del tempo in idle, il dispositivo stesso offre delle modalità speciali che riducono il consumo di potenza:
- il self-refresh compensato in temperatura permette di ridurre la frequenza di self-refresh mentre è in stato di idle sotto una temperatura di 45°C in quanto essendo la corrente di leakage dipendente dalla temperatura in modalità direttamente proporzionale più è bassa, più si riduce la corrente. Di default, la temperatura è impostata al caso peggiore (85°C) in una SDRAM standard, mentre la si può configurare ad un valore minore sotto i 45°C;
- il parziale self-refresh dei banchi di memoria offre un buon approccio per limitare i consumi andando appunto a fare il nello stato di idle, il refresh dei soli banchi interessati nel processo.
SOLUZIONE IDEALE: uso di una low power SDRAM per ridurre i consumi, non tanto per la compensazione in temperatura che potrebbe risultare poco efficace nello stato di idle a causa della dissipazione poco facile sul satellite, ma per le caratteristiche elettriche peculiari che la contraddistinguono.
</td></tr>
<tr><td>FREQUENZA</td><td>Le frequenze in gioco potrebbero andare da 100 MHz (per una PC100 compliant) in su passando per 133 MHz (PC133 compliant). Il discorso è che più aumenta il clock di sistema, più aumentano i consumi. SOLUZIONE COMPROMESSO: si usi se non disponibile 100 MHz per quella SDRAM, una frequenza di 133/143 MHz.</td></tr>
<tr><td>POSSIBILI CONFIGURAZIONI SUPPORTATE</td><td>
<table>
<tr><th>System Size (M byte)</th><th>System Size (M bit)</th><th>SDRAM Configuration</th><th>Number of Chips</th><th>Good or Bad?</th></tr>
<tr><td>16</td><td>8M x 16</td><td>8M x 8</td><td>2</td><td>Moderate for power saving, but no for data organization!</td></tr>
<tr><td>16</td><td>8M x 16</td><td>8M x 16</td><td>1</td><td>Good solution 1!!</td></tr>
<tr><td>32</td><td>16M x 16</td><td>16M x 4</td><td>4</td><td>Bad for too many chips!</td></tr>
<tr><td>32</td><td>16M x 16</td><td>16M x 8</td><td>2</td><td>Moderate for power saving, but no for data organization!</td></tr>
<tr><td>32</td><td>16M x 16</td><td>16M x 16</td><td>1</td><td>Good solution 2!! For further improvements!</td></tr>
<tr><td>64</td><td>32M x 16</td><td>32M x 4</td><td>4</td><td>Bad for entire memory size (too large!!)</td></tr>
<tr><td>64</td><td>32M x 16</td><td>32M x 8</td><td>2</td><td>Bad for entire memory size (too large!!)</td></tr>
<tr><td>64</td><td>32M x 16</td><td>32M x 16</td><td>1</td><td>Bad for entire memory size (too large!!)</td></tr>
<tr><td>128</td><td>64M x 16</td><td>64M x 4</td><td>4</td><td>Bad for entire memory size (too large!!)</td></tr>
<tr><td>128</td><td>64M x 16</td><td>64M x 8</td><td>2</td><td>Bad for entire memory size (too large!!)</td></tr>
<tr><td>128</td><td>64M x 16</td><td>64M x 16</td><td>1</td><td>Bad for entire memory size (too large!!)</td></tr>
</table>
</td></tr>
<tr><td>CONFRONTO CON SRAM</td><td>La SRAM consuma meno ma è più lenta considerando i tempi d’ accesso di una generica RAM e confrontandoli con le sincrone DRAM che hanno CAS latency di 2,3 cicli di clock di media. Si tenga sempre presente che un semplice sensore d’ immagine lavora con frequenze di pixel clock di 27MHz, ma laddove ci riferissimo a uno un po’ più complicato rispetto a quello utilizzato da noi, avremmo sicuramente caratteristiche più vincolanti (maggiore fps, maggiore APS, maggiore risoluzione) da dover tenere in conto.</td></tr>
</table>

Tabella. 3.5 Low power SDRAM

Brevemente, le caratteristiche fondamentali nell’ uso di una memoria del genere sono:

- ✓ Uso della SDRAM il meno possibile (solo nell’ acquisizione dell’ immagine):
- ✓ Uso di tensioni basse (se si usa una low-power SDRAM si può beneficiare dei 1.8V 2.5V);
- ✓ Velocità di refresh più bassa possibile (tipicamente è di 64ms per uno scenario worst case (alta temperatura) in un ambiente a bassa temperatura;
- ✓ Privilegiare trasferimenti di dati tra banchi di memoria e non all’ interno di uno stesso;

Dopo questa attenta analisi, rivolta in particolar modo alle SDRAM si è deciso di utilizzarne una non a basso consumo, ma che rispettasse comunque le considerazioni fatte in precedenza; in pratica, modificando lo spazio di indirizzamento esterno del processore si è adottata una SDRAM a 8Mbyte PC100-compliant, quindi con un clock massimo di 100 MHz, da fornire attraverso il controller asincrono dell’ EBIU per una VDDINT = 1.4V. In questo maniera, si è ragionevolmente privilegiato il minimo consumo di potenza. Concludendo, la scelta è caduta sulla SDRAM MT48LC4M16A2 prodotta dalla Micron e attualmente ancora in produzione.

3.2.4.2 FLASH

Per quanto riguarda la prima limitazione, la funzione principale delle memorie che devono essere adottate è quella di contenere il codice sorgente del programma utilizzato dal DSP in maniera sicura rispetto ai problemi legati all'ambiente spaziale (spiegati approfonditamente nel primo capitolo) e allo stesso modo le immagini acquisite dal sensore. Si sarebbe quindi dovuta cercare una memoria adatta a svolgere questa funzione, quindi non volatile, riprogrammabile on board e a basso voltaggio, e compatibile con la tipologia di memorie supportate dal processore. Nel caso dell' *image processor*, questo problema è stato affrontato contemporaneamente alla scelta del processore stesso in quanto, come spiegato prima, la flash risulta già integrata al suo interno e ai fini della stessa tesi, l' utilizzo che ne viene fatto in fase prototipale, è quello di contenere i dati relativi agli angoli e conservare, in assenza di alimentazione, un tot di immagini. La flash in questione, è una S25FL116K (2 MByte) connessa alla porta SPI0 del Blackfin.
Nel qual caso volessimo approfondire la realizzazione della scheda, la memoria risulterebbe essere non più adatta a contenere sia il codice programma che le diverse immagini; perciò, visto la disponibilità data dal controller asincrono dell' EBIU, si potrebbe connettere una Flash di dimensioni giuste che possa accogliere il salvataggio di più dati. In quel caso, con un discorso più rigoroso e attento ad una realtà che va oltre il prototipo, avremmo sicuramente vagliato le due architetture a disposizione, a porte NAND e NOR, e scelto in base ai vincoli dell' ambiente spaziale, quella che sarebbe stata meno immune alle radiazioni cosmiche. Facendo solo un breve accenno, da una parte avremmo considerato la grande densità delle Flash a gate NAND, ma visto l' utilizzo di all' incirca un Mbyte, la scelta sarebbe caduta sicuramente verso le NOR che risultano essere più robuste alle radiazioni ionizzate presenti nello spazio libero.

In definitiva, la memoria FLASH deve quindi contenere il binario del software utilizzato dal DSP, (stimato al massimo di 200 Kbyte), e al massimo 5 immagini compresse JPEG di dimensioni massime di 100 Kbyte ciascuna. Inoltre si era previsto anche una possibile copia di backup del software da utilizzare nel caso di corruzione della copia principale.
Giunti a questo punto, dopo aver appurato che le dimensioni minime non possono essere inferiori ad 1Mbyte e ribadendo che la scelta della stessa è stata fortemente correlata a quella vista in precedenza per il processore, si è confermata una volta di più la scelta dell' ADSP-BF518F16.

3.3 Funzionamento componenti principali

3.3.1 DSP Blackfin BF-518F16

Il DSP Blackfin BF-518F16 di Analog Devices è un processore ad alte performances e bassi consumi, grazie alla possibilità di arrivare ad una frequenza di clock (precisamente core clock: cclk) di 400 MHz e varie modalità di funzionamento per una dinamica gestione della potenza assorbita. Infatti la possibilità di variare sia la tensione del core che la sua frequenza, permette di ottimizzare il consumo di potenza per ogni specifico task da eseguire.

Il processore è alimentato esternamente a 3,3V, ma tale tensione è utilizzata per alimentare tutte le periferiche, mentre l'alimentazione del core di 1,4V è ottenuta mediante un regolatore di tensione esterno che comanda un regolatore di tensione switching di tipo buck, montato a componenti discreti esternamente alla board. Il regolatore da una tensione d'ingresso da 6,5V a 36V, fornisce da 1,25V a 5V al core del processore. Lo schema utilizzato è il seguente:

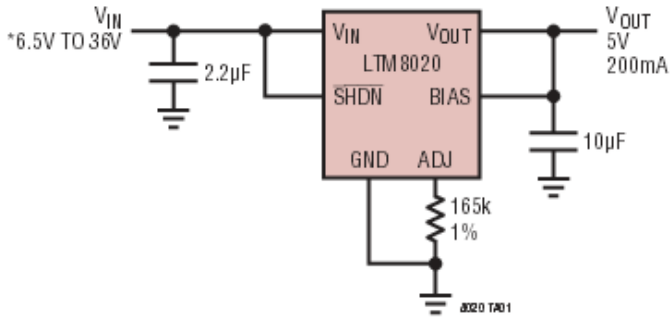


Figura 3.6: Regolatore di tensione esterno

Il transistor disponibile in forma di circuito integrato su 21 pin in package LGA, è prodotto dalla Linear Technology e presenta un pin di SHDN attivo basso che permette di spegnere il buck stesso attraverso l' apposito comando dall' OBC. Inoltre, sono importanti il valore di resistenza sul piedino ADJ in quanto permetterà di regolare Vout e il pin BIAS che sarà la tensione che andrà ad alimentare la circuiteria interna dell' integrato stesso. Nella fattispecie, sono anche state prese delle precauzioni attraverso un diodo schottky per evitare nel qual caso fosse spento, corrente inverse che potrebbero portare a danni irreversibili.

Ritornando ai processori Blackfin, essi supportano un architettura Harvard modificata in combinazione con una struttura a memoria gerarchica; il core per esempio, ha una struttura interna costituita da sei unità funzionali principali per l'esecuzione di algoritmi sia di processamento numerico che di controllo generico dei segnali provenienti dall'esterno; il DSP presenta:

- due unità MAC (Multiply Accmulate) da 16 bit
- due unità aritmetico/logiche (ALU) da 40 bit
- uno shifter di tipo Barrell a 40 bit
- 4 ALU video a 8 bits
- 40-bit shifter

- 116K bytes di memoria on chip ai vari livelli

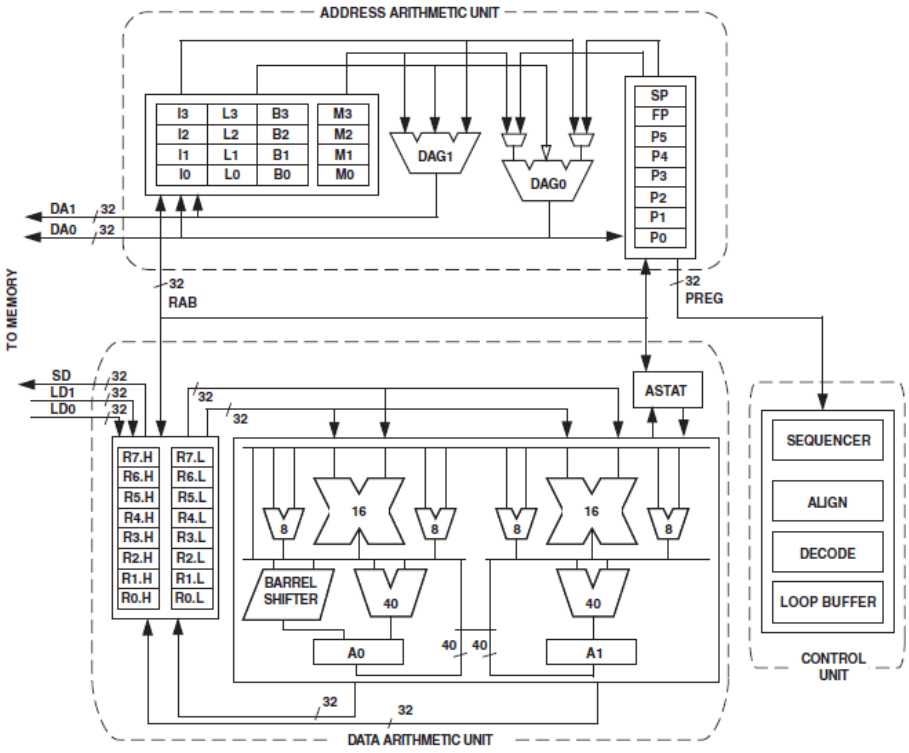


Figura 3.7: Architettura del core del processore

Le due unità MAC vengono utilizzate per operazioni di moltiplicazione o moltiplicazione e accumulo con addizione o sottrazione. Infatti i moltiplicatori operano su dati a virgola fissa a 16 bit e producono quindi risultati a 32 bit i quali possono essere sommati o sottratti dal registro accumulatore a 40 bit.
Quanto detto potrebbe essere sfruttato a pieno regime per calcoli di una certa complessità computazionale, nello specifico per sviluppi successivi, per una compressione dell'immagine in formato JPEG, rendendo la compressione molto più veloce rispetto ad un processore general purpose oppure ancora, se volessimo adottare degli algoritmi avanzati per l' individuazione di una zona più ristretta (ROI: region of interest), garantirebbero un supporto all' altezza in operazioni di tracking dello spot luminoso.

Le due unità ALU, sono ovviamente utilizzate dalle operazioni seguenti, presenti nel codice poi sviluppato, per compiere le varie funzione richieste dalle specifiche:

- somme e sottrazioni di registri in virgola fissa
- accumulo e sottrazione di risultati dei moltiplicatori
- funzioni logiche AND, OR, NOT, XOR
- funzioni: ABS, MAX, MIN

Il barrell shifter è ovviamente utilizzato per le operazioni di shift aritmetico e logico dei bit su registri da 16, 32 e 40 bit, rotazioni e vari test sui bit.

Infine, il set di quattro ALU vengono utilizzate per l' elaborazione dei segnali video con alta efficienza; ogni ALU video prevede come input da una a quattro coppie di registri a 8 bit e come output da uno a quattro registri da 8 bit; tra le funzioni implementate vi sono somma e sottrazioni, medie, packing o unpacking dei dati.

Essenzialmente acquisiremo i dati a 10 bits con relative tempistiche dal sensore, attraverso la porta parallela e secondo le regole del formato ITU-R BT 601, dopodiché, servendoci di uno dei canali messi a disposizione della DMA instraderemo sul determinato bus i 16 bits di dato che andremo successivamente e provvisoriamente ad immagazzinare nella SDRAM.

La DMA verrà settato in modalità di acquisizione di un singolo fotogramma, chiamato DMA STOP Mode, che alla fine dell' acquisizione, generando un interrupt fermerà lo stesso controller secondo l' adeguata interrupt service routine; infatti il processore prevede anche una modalità di acquisizione continua, utile per i video.

L' architettura offre 3 modi per operare: user mode, supervisor mode e emulation mode. La modalità utente ha un accesso ristretto a certe risorse del sistema così offrendo un ambiente sw protetto mentre per la supervisor mode si ha un completo accesso al sistema.

L'architettura della memoria interna del processore Blackfin è costituita da tre blocchi, i quali consentono un accesso ad elevata velocità dal core del processore; i tre blocchi sono i seguenti:

- L1 (instruction memory) che consiste di una SRAM a 48K bytes dei quali 16K bytes possono essere configurati come una cache set associative a 4 vie. Questa memoria è accessibile ad alta velocità dal processore.
- L1 (data memory) è il secondo blocco di memoria che consiste di una SRAM a 64K bytes dei quali 32K bytes possono essere configurati come cache. Questo blocco di memoria è accessibile dal processore ad alta velocità.
- Il terzo blocco di memoria è un 4K bytes di scratchpad SRAM, alla quale si accede alla stessa velocità di una memoria L1, ma questa memoria è solo accessibile come data SRAM e non può essere configurata come una memoria cache.

Nel nostro sistema il software si prevede che abbia dimensioni massime di circa 200 Kbyte, quindi i 48 Kbyte di memoria codice interna non sono sufficienti (senza utilizzarla come cache), quindi a seconda della modalità di boot predefinità, il codice verrà mantenuto su una memoria esterna tipo SDRAM o FLASH, come verrà spiegato in seguito. La seguente tabella riporta la mappa di memoria intera del processore e spiega in modo migliore la sua struttura, fornendo anche gli indirizzi utilizzati per accedere alle varie aree di memoria:

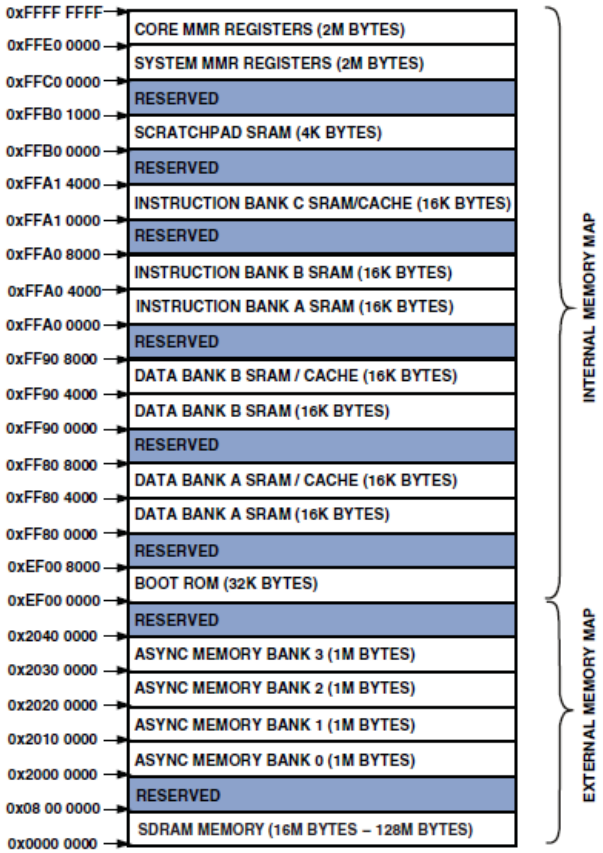


Figura 3.8: Mappa di memoria Blackfin BF-518F16

La memoria del processore Blackfin è vista come un singolo unificato spazio di indirizzi a 4G byte usando indirizzi a 32 bit. Tutte le risorse includono sia la memoria interna/esterna e i registri di controllo dì I/O che occupano separatamente sezioni di uno stesso spazio di indirizzi. Le porzioni di questo spazio di indirizzi comune è organizzato secondo una struttura gerarchica per offrire un buon bilanciamento tra costi e performance di alcune molto veloci e a bassa latenza memorie come cache oppure SRAM e offrire un interfaccia del sistema alla memoria più largo e più a basso costo.

La memoria esterna è gestita tramite l'External Bus Interface Unit (EBIU). Questa interfaccia a 16 bit offre una connessione glueless alla memoria flash interna e alla boot ROM. La memoria flash interna esce dalla produzione in maniera vergine (in erased state) eccetto per il blocco 0 (che è in uno stato sconosciuto e prima di programmare questo blocco bisognerebbe effettuare un operazione di cancellazione). Come si può facilmente vedere dalla figura 2.7, essa è disponibile partendo dall' indirizzo mappato come 0x20000000 disponibile solo sul DSP 518F16 e la sua propria architettura a banchi multipli permetterà operazioni duali: quindi mentre in un banco lo si programma o lo si cancella, dall' altra parte si può andare a leggerne altri.
Nel nostro caso, nel processore in questione dapprima era integrata una SST25WF040 SPI Flash memory, successivamente è stata rimpiazzata dalla S25FL116K da 16-Mbit (2-Mbyte) richiedendo un update dei drivers per quanto concerne il compilatore.

Le periferiche sono connesse al core medianti molti bus ad alta banda, come si può vedere dalla figura sotto dove sono evidenziati in rosso i moduli d' interesse allo scopo della nostra tesi:

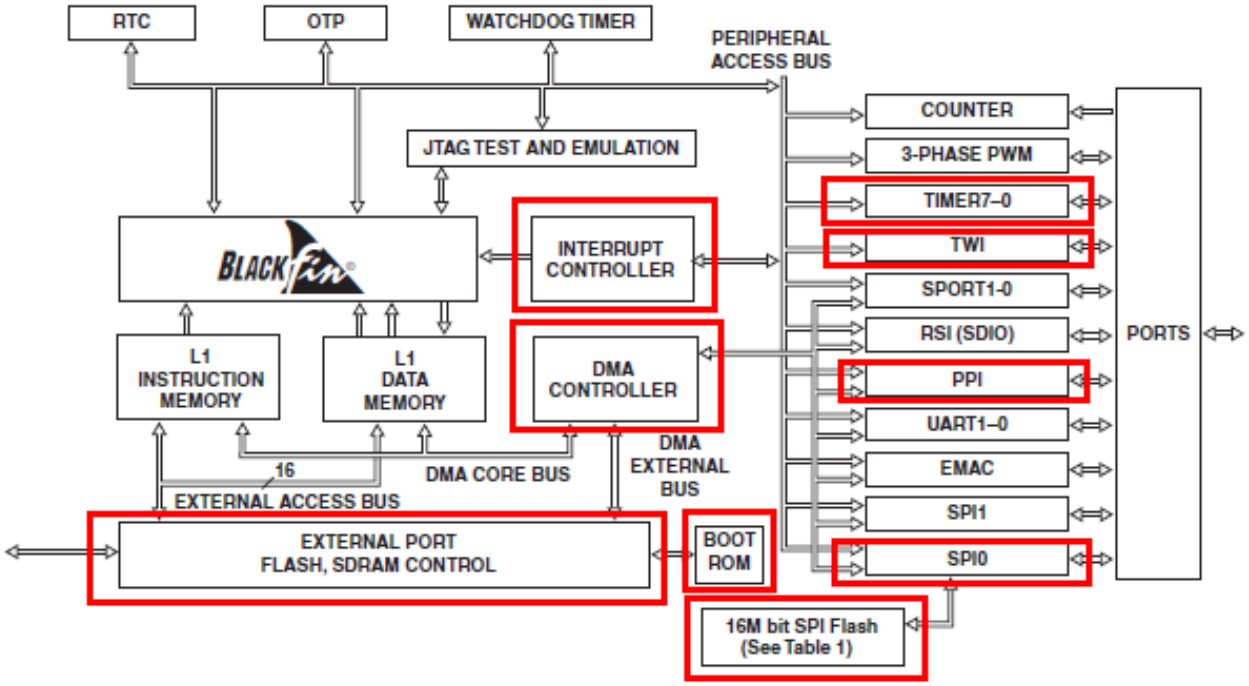


Figura 3.9: schema a blocchi processore

Oltre all'External Bus Interface Unit, il sistema di periferiche include diverse interfacce e unità funzionali:

- Parallel Port Interface (PPI)
- Two Wire Interface (TWI)
- 2 Serial Ports (SPORTs)
- 2 Serial Perpheral Interface (SPI)
- 2 Universal Asynchronous Receiver Transmitter(UART) con supporto IRDA;
- 8 General-purpose timers con supporto PWM
- Real time clock timer
- Watchdog timer
- 40 General purpose I/O

Nel nostro progetto, alcuni timers come anche le SPORTS e le UART non sono state utilizzate.
La porta denominata PPI è la porta parallela a 16 bit più un segnale di clock compatibile col formato video digitale ITU-R BT.656-4, che verrà collegata direttamente al sensore d' immagine per gestire l' acquisizione del frame secondo determinati segnali di sincronizzazione imposti in prima battuta dal clock del processore e in seconda, dalle impostazioni effettuate sul sensore stesso.

Infine i general-purpose I/O sono costituiti da 40 pins, i quali però sono multiplexati con alcuni segnali della porta PPI, dei timers e più in generale di tutte le funzioni messe a disposizione dal Blackfin. Come visto nel capitolo precedente e come è logico che sia nella modularità di Aramis, gran parte di questi sono stati utilizzati per interfacciarsi con gli altri sottosistemi e laddove questo non sia stato possibile, per le particolari funzioni presenti sullo stesso pin, si è stati costretti a dover compiere scelte obbligate (discusse ampiamente nel capitolo precedente). Essi comunque, possono essere configurati come sensibili al fronte o al livello logico, la polarità del fronte o del livello che li rende attivi e la loro direzione ossia, se viene utilizzato come pin d'ingresso (valore di default) o come pin d'uscita. Qui di seguito viene riportata la tabella riassuntiva di come sono stati impiegati questi general purpose pins per quanto concerne il controllo dell' acquisizione (bus IMAGER_CTRL):

DISPOSITIVO		**PIN**	**NOME**	**FUNZIONE**
I^2C		172	SDA	SDA
		173	SCL	SCL
SENSOR_CONTROL		155	PH5	STANDBY
		34	PG6	EXPOSURE
		25	PG13	HSYNC
		21	PG14	VSYNC
		20	PG15	FIELD
		26	PG12	PIXEL_CLOCK
		174	PF10	MASTER_CLK

Figura 3.10: descrizione connessioni general purpose I/O pin per acquisizione

Sono stati evidenziati solamente i segnali di controllo, mentre per quanto concerne i pin che accoglieranno in ingresso i pixels a 10 bit si rimanda la discussione al paragrafo riguardante la PPI.
Come è stato accennato all'inizio, la frequenza del core e delle periferiche è ottenuta tramite un quarzo esterno e poi agganciata ad un PLL frazionario, che quindi utilizza un divisore programmabile per ottenere una frequenza massima di 400 MHz. Inoltre i segnali di clock, ossia quelli uscenti dal VCO, destinati alle periferiche (chiamato SCLK) e al core (chiamato CCLK) sono distinti, e sui due sono presenti due divisori distinti, in modo da avere la possibilità di avere le periferiche a bassa frequenza ma il core ad una più elevata; questo comporta un notevole risparmio di potenza, dato che le periferiche sono alimentate a 3,3V. Lo schema è il seguente:

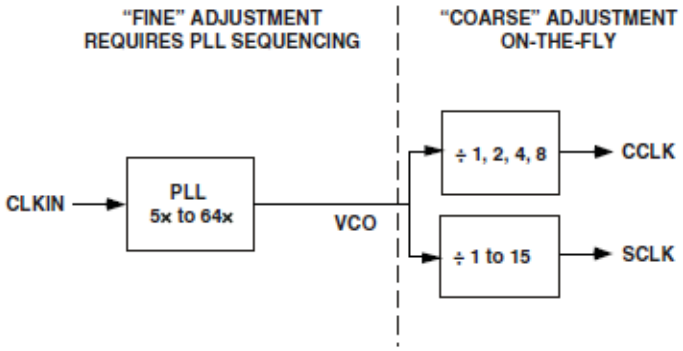


Figura 3.11: catena di gestione del clock

Il quarzo utilizzato per il processore BF-518F16 è da 26 MHz, prodotto dalla Epson in package SMD denominato TSX_3225. Il divisore del PLL è stato settato in modo tale da avere un VCO a 312 MHz. Così si è poi settato i due divisori a 2 e 3, così da avere la frequenza delle periferiche a 104 MHz (parecchio utilizzato) e sul core 156 MHz (poco utilizzato). La particolare impostazione è in funzione della poca elaborazione da parte del core e dal quarzo messo a disposizione per il progetto; ciò significa che se dovessimo fare delle modifiche per un elaborazione complessa, i parametri progettuali andrebbero modificati in relazione ai consumi. Questo permette di utilizzare la memoria SDRAM adottata in modalità PC100 e di non consumare eccessivamente con il core che ha frequenza superiore (si ricordi sempre che non può essere mai: CCLK < SCLK).

Interessante particolarità del processore Blackfin è la modalità di boot; esso prevede otto modalità di boot diverse, cinque dei quali caricano internamente il programma da una memoria interna/esterna nella memoria codice interna L1, dopo aver ricevuto il comando di reset esterno (nel nostro caso sono tutte disponibili grazie all' impiego del ADSP che è dotato di flash interna). Altri due modi sono accumunati, sempre dopo il reset, dall' esecuzione del programma da un SPI0 host o UART0 host . La modalità di boot viene impostata esternamente mediante tre pin, chiamati BMODE0 e BMODE1 e BMODE2, del processore che come visto saranno pilotati dall' OBC attraverso lo slot MODULO_OBC_B; un piccolo boot kernel presente all'interno della Boot ROM interna del processore all' indirizzo 0xEF000000, campiona dopo il reset il valore dei due pin e agisce secondo la seguente tabella:

BMODE[2:0]	SORGENTE DEL BOOT	DESCRIZIONE
000	No boot – idle	Il processore non si avvia piuttosto il boot kernel esegue un istruzione di IDLE.
001	Boot da una memoria flash esterna a 8-bit o 16 bit;	Il kernel si avvia dall' indirizzo 0x2000 0000 nel banco di memoria asincrona 0; il primo contiene ulteriori istruzioni circa la larghezza della memoria (8 o 16 bit);
010	**Boot da una memoria interna SPI**	**Il kernel si avvia dalla memoria flash on-chip indirizzata a 24-bit e connessa all' interfaccia SPI0.**
011	Boot da una memoria esterna SPI	Dopo un iniziale routine di rilevazione del dispositivo, il kernel si avvia da una flash SPI indirizzabile a 8-bit, o a 16 bit o a 24-bit o ancora a 32-bit. Oppure da una EEPROM che si connette al pin SPIO_SEL2.
100	Boot dall' host device da SPI0	In questa modalità slave, il kernel aspetta il flusso di boot che sarà poi applicato all' SPI0 da un dispositivo host esterno.
111	Boot dall' host device da UART0	In questa modalità slave, il kernel aspetta il flusso di boot che sarà poi applicato all' UART0 da un dispositivo host esterno. Prima di fornirlo però, l' host in attesa invia un carattere 0x040 al kernel che lo esamina per sincronizzare il bit rate.

110	Boot da una memoria SDRAM	Questa modalità fornisce un opzione di boot veloce “a caldo” e richiede che il relativo controller sia programmato da una routine preboot basata su impostazioni OTP. Il kernel inizia l avvio dalla locazione 0x0000 0010.
101	Boot da una memoria on-chip OTP	Questa è l’ unica modalità autonoma che si avvia dalla memoria seriale OTP presente sul chip. Di dafault, il flusso in attesa risiede sulla pagina attiva 0x40, ma può essere cambiata programmando il campo OTP_START_PAGE in OTP page PBS01H.

Tabella 3.12: Modalità di boot

Il boot kernel processa il boot stream blocco dopo blocco finchè l’ invio di un comando speciale non fa terminare la procedura e salta all’ indirizzo programmato di partenza del programma sorgente, che solitamente è 0xFFA00000, ossia quello della memoria codice L1. Per ogni modalità di boot, un header di 16 byte è dapprima letto dalla memoria esterna(esso specifica il numero di byte trasferiti e l’indirizzo della memoria destinazione),e poi non appena tutti byte sono stati caricati, l’esecuzione inizia dall’indirizzo di memoria caricato nel registro EVT1.
Nel nostro sistema, la modalità di boot sulla quale ci concentreremo è la terza (evidenziata in tabella), codificata con 010. Difatti il programma sarà contenuto nella memoria interna FLASH integrata, poi in maniera asincrona sfruttando i parametri temporali preservati; in BMODE = 010, il controller configura la flash per una comunicazione burst sincrona e da quest’ ultima si avvia.

Il processore Blackfin esegue il boot dopo aver ricevuto un segnale di reset da un pin esterno di durata minima di 11 cicli di clock proveniente dal modulo OBC attraverso la slot OBC_B; subito dopo, si potrebbero usare due segnali forniti dal processore per coadiuvare l’ interazione con il regolatore esterno:

- PG (attivo basso): permette al processore di attivarsi dopo che la tensione interna ha raggiunto un livello scelto (rilevazione del tempo di startup dopo l’ ibernazione); nella scheda, è stato posto provvisoriamente a 0V;
- EXT_WAKE (attivo basso -> processore in ibernazione) : per rimuovere la potenza dal core: alto al power-up, viene connesso all’ abilitazione del regolatore che fornisce l’ alimentazione di 1,4V.

Il consumo di corrente e di potenza del processore Blackfin risulta piuttosto complesso da calcolare; infatti, esso sarà composto dal consumo del core, sommato al consumo di tutti i componenti esterni attivi, ossia in commutazione:

$$P_{TOTAL} = P_{EXT} + (I_{DD} \text{ x } V_{DDINT})$$

l’assorbimento dei componenti esterni infatti, dipende dal numero di pin coinvolti(n), dalla frequenza di commutazione (dei pin coinvolti, quindi la metà del clock utilizzato nei protocolli sincroni quali quello

ITU-R BT.656-4 e SPI, in quanto transizioni successive ‘0’-‘1’ e ‘1’-‘0’ su linee dati o indirizzi avvengono ogni due cicli di clock), il cui valore non può superare quello del clock del processore per le periferiche (SCLK), dalla tensione di funzionamento, tipicamente 3,3V (sensore, SDRAM e processore) e dalla capacità d’ingresso (C), delle porte CMOS dei componenti integrati:

$$P_{EXT} = n \times C \times V^2_{DD} \times f$$

In seguito verranno presentati il funzionamento dei vari componenti presenti sulla scheda e la loro interfaccia con il processore; per ognuno quindi verrà calcolato il loro consumo interno e dell’interfaccia, in modo da poi riassumere i consumi totali del sistema nelle varie fasi di funzionamento nel sottocapitolo CONSUMI ENRGETICI, sapendo che l’assorbimento di corrente del core del processore funzionando a 100 MHz circa è di 60 mA alla tensione di 1,2V nominali.

3.3.2.Sensore d’ immagine

Come visto, in un paragrafo precedente, si è scelto come sensore il modello MT9V034C12STM prodotto dall’ Aptina Imaging, che soddisfa in pieno le nostre esigenze e garantisce attraverso le caratteristiche che di seguito verranno elencate, la copertura del FOV (capitolo 2), bassi consumi e una grande accuratezza in ottica complessiva del sistema.

Nello specifico, abbiamo:

- Sensor Image Color Type: monocromatico;
- Sensor Image Size: 752 x 480 pixels;
- Operating Supply Voltage: 3,3 V;
- Operating temperature range: -30 °C to +70 °C;
- Master clock (recommended) e maximum data rate: 27MPS;
- Power Consumption: < 160mw at maximum data rate (120 standby power at 3,3 V);
- Responsivity: 4,8 V/lux sec (550nm);
- Data Output Format: Single sensor mode: 10 bit parallel /stand-alone;
- Protocol: I^2C;

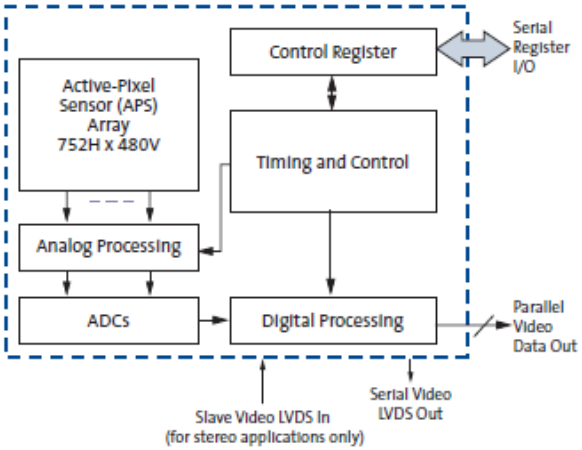


Figura 3.13: Schema a blocchi del sensore

L' area attiva, circondata da *false* colonne e righe trasparenti per migliorare l' uniformità dell' immagine, è composta da 782 colonne per 492 righe, delle quali le ultime 26 colonne e le prime 8 righe sono otticamente nere per ottimizzare il livello del colore nero, mentre sono attive le restanti 753 colonne e 481 righe.

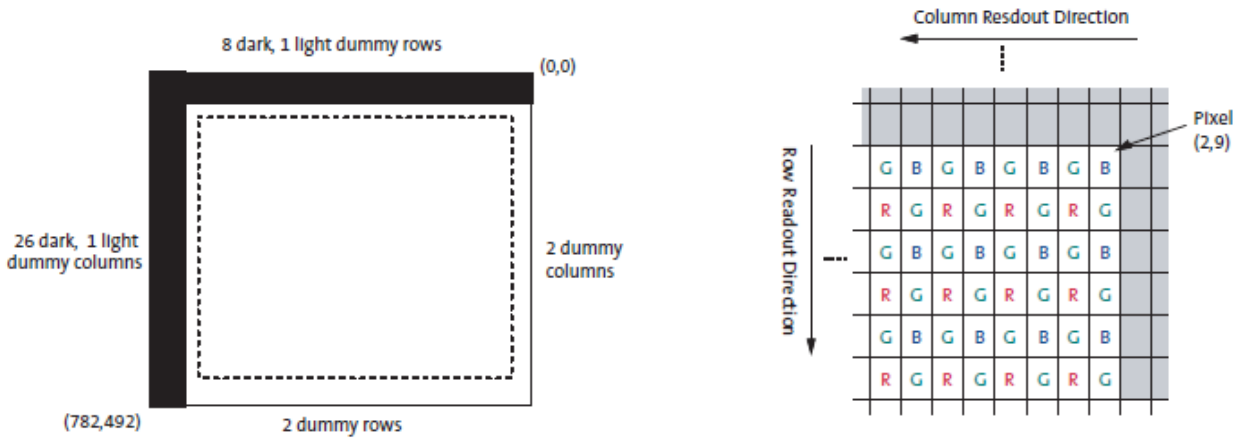


Figura 3.14: Descrizione del pixel array e un tipico pixel color pattern

Ritornando a quanto detto in precedenza, questo tipo di sensore permette di raggiungere i vantaggi della tecnologia CCD per quanto concerne rapporto segnale-rumore e problematiche legate alla bassa sensitività, sfruttando dimensioni, costi e integrazione di una tecnologia CMOS; inoltre, si consideri la possibilità di impostare attraverso il protocollo I^2C certi parametri come guadagno, tempo di esposizione e dimensione del frame per una corretta immagine secondo le tempistiche vincolanti del sistema. Di default, il sensore presenta una risoluzione VGA con una velocità di 60fps.

Il formato del dato d' uscita è di 10 bits per pixel ed è fornito da un convertitore analogico-digitale. Esso è sincronizzato con il pixel clock d' uscita e come si può facilmente vedere dal *timing diagram* sotto, quando il segnale di LINE_VALID è alto un pacchetto di dati di 10 bit è inviato sull' uscita ogni PIXCLK.

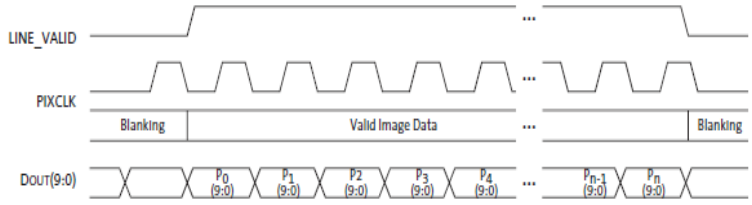


Figura 3.15: Temporizzazioni dei segnali di controllo con il flusso dati

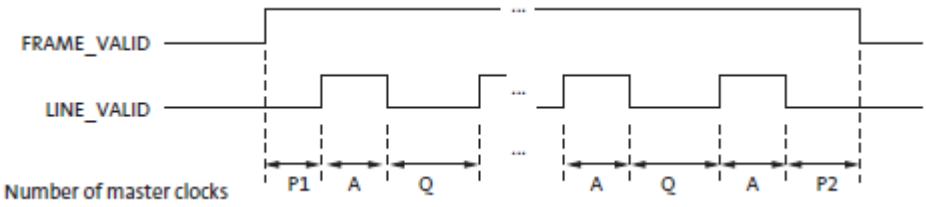
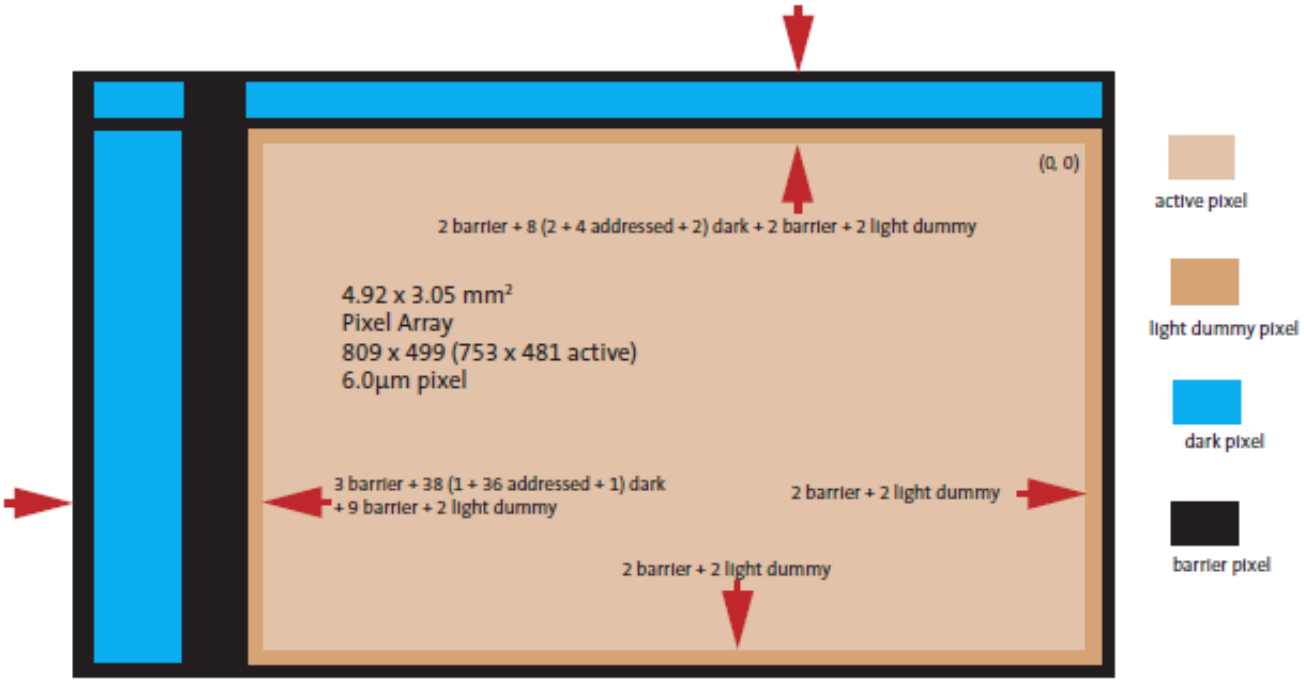


Figura 3.16: Temporizzazioni dei completa con i segnali di FRAME_VALID e LINE_VALID

Il PIXCLK è solitamente l' inversione del segnale SYSCLK, ovvero il master clock, la cui frequenza raccomandata è di 26,66 Mhz. In realtà, la struttura dell' array di pixels è una matrice 809 x 499 righe della quale alcune righe e colonne sono usate per la correzione del rumore e del livello di nero



.

Figura 3.17: Pixel array

Lo standard ITU – R 656 definisce i vari parametri per la connessione e la trasmissione unidirezionale di segnali video tra una singola sorgente e una singola destinazione. L'ADSP-BF518F16 supporta questo formato (conosciuto anche come CCIR-656) ed è in grado di ricevere un flusso digitale video con i segnali orizzontale (H), verticale (V) e field (F) inviati come parte dello stesso flusso dati in una serie di

bytes che formeranno poi una word di controllo: i segnali SAV (start of active video: transizione di H da 1 → 0) e EAV (end of active video transizione di H da 0 → 1) decretano l' inizio e la fine degli elementi di dato da leggere in ogni linea. Un intero campo video è compreso tra l' active video + blanking orizzontale (spazio tra EAV → SAV) e verticale (spazio dove V = 1). L' immagine in progressione non farà nessuna distinzione tra field 1 e field 2, mentre un interlacciata richiede che ogni campo venga gestito unicamente in quanto le righe alternate di ogni campo si combinano per creare l' immagine video reale.

Purtroppo, finita questa breve premessa, nella scheda la sorgente del segnale video in suddetto formato digitale sarà il sensore d' immagine, mentre il destinatario sarà il DSP Blackfin. A tal porposito, il modello MT9V034C12STM produce dei dati praticamente in formato digitale su 10 bit insieme ai segnali di sincronizzazione HSYNC e VSYNC, quindi non potremmo usare l' ITU-R BT 656-4 del quale il Blackfin è supportato, ma si userà una modalità d' ingresso *general purpose* con 2 *frame syncs* si faccia riferimento alle temporizzazioni viste sopra).

In conclusione, dal sensore d' immagine arrivano sulla porta PPI dell' ADSP-BF518F16, 451,2 KBytes ad una frequenza di circa 26 MHz in un tempo d' integrazione stimato con le impostazioni di default di 16,66ms (l' intervallo da quando FS2 va 0 -> 1 a 1->0). In tabella sono elencati i calcoli fatti per la fase di acquisizione:

SYSTEM	OBJECT	SIZE	NOTE
Image sensor	ACTIVE PIXELS:	752 H x 480 V = 360960 pixels;	Programmabile attraverso i registri : • R0x05 (horizontal blanking) e R0x06 (horizontal blanking) per context A; • R0xCD (horizontal blanking) e R0xCE (horizontal blanking) per context B; • Numero di pixels per linea (= 752) programmabile con R0x04; • Numero di righe: (= 480) programmabile con R0 x03;
Image sensor	OUTPUT	10 bits column – parallel (gray – level) 360960 x 10 = 3609600 bits = **451,2 KB** (intero frame)	Si ricordi che non fanno parte del readout i dummy e blanking pixels.
Image sensor	SYSCLOCK	≈ **26 MHz**	Il master clock viene generato dal timer TMR3 (PF10) del Blackfin ottenuto dalla SCLK/4
Image sensor	PIXEL CLOCK	Versione invertita del master clock ≈ **26 MHz** (≈ 37,5 ns: lasciato come raccomandabile)	Tempo di lettura di una riga attiva: 28,20 µs (=752 x 37,5µs); Tempo di lettura di una riga con horizontal blanking: 31,72 µs (=(752 + 94) x 37,5µs); Tempo di lettura di un frame valido: 15,23 ms (480 x 31,72 µs); Tempo di lettura di un intero frame **16,66 ms** (15,23 ms + 1,43ms);
Image sensor	EXPOSURE	Determina l' inizio del tempo d' integrazione	Tramite il pin PG6 nella modalità snapshot, si dà inizio all' acquisizione del frame per tutta la durata del tempo di integrazione; una volta terminata torna basso mentre si avvia il *readout*;
Image sensor	STANDBY	Abilita e disabilita il sensore	Tramite il pin PH5 del processore si può spegnere il sensore una volta terminata l' acquisizione del frame per salvaguardare i consumi.

Figura 3.18: Calcoli sull' interfaccia

3.3.2.1 Programmazione via I^2C

Prima di entrare nello specifico del protocollo, iniziamo con il dire che il sensore lavora in tre possibili diverse modalità:

- MASTER MODE: genera il readout timing;
- SLAVE MODE: il sensore viene controllato esternamente per quanto riguarda readout e tempo d' integrazione;
- SNAPSHOT MODE: accetta un sincronismo esterno che determini l' inizio del tempo d' integrazione e poi viene quindi generato il readout timing.

Il tempo d' integrazione è programmato attraverso l' interfaccia seriale TWI in *master* e *snapshot mode*;
In master mode, abbiamo due possibili metodi di funzionamento: simultaneo (il periodo di esposizione è il *readout* migliorandone notevolmente la velocità di risposta) e sequenziale (il periodo di esposizione è seguito dal *readout*) che sono opzionabili vicendevolmente attraverso l' I^2C .

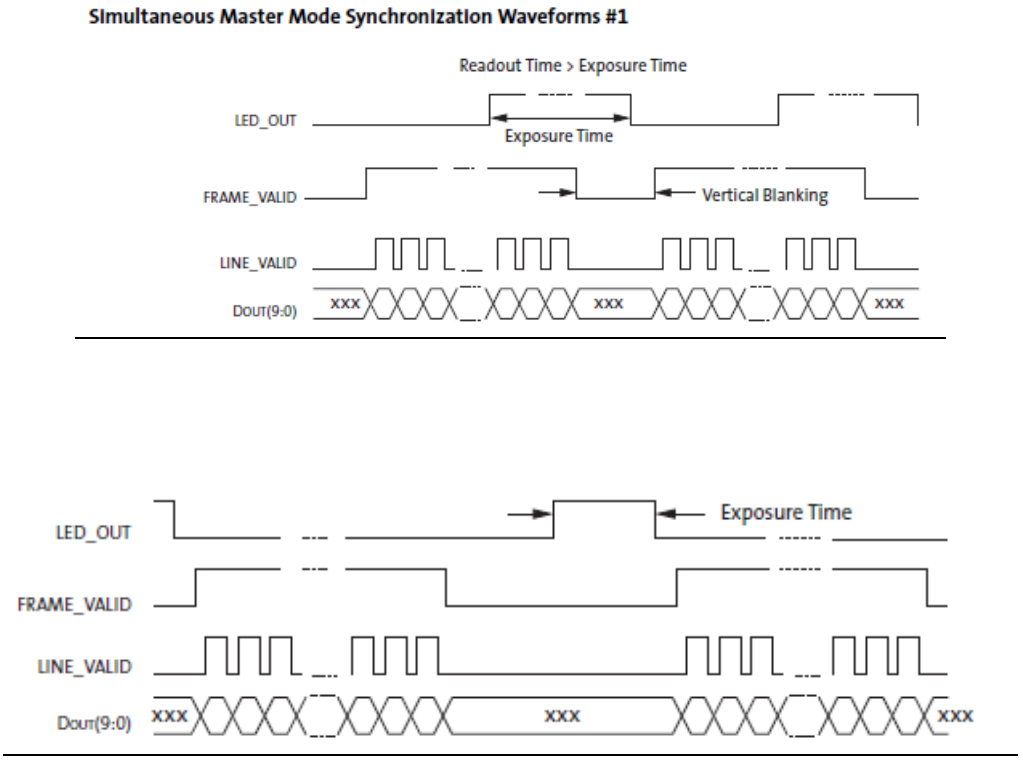


Figura 3.19: Due modalità di sincronizzazione master

In slave mode, attraverso tre pins EXPOSURE, STFRM_OUT (input) e STLN_OUT (input) si riescono a controllare l' esposizione e lettura dei pixels; in particolare, EXPOSURE e STFRM_OUT regolano l' inizio e fine dell' integrazione rispettivamente, anche se il segnale STFRM_OUT è usato pure per abilitare il processo di *readout*.
Il terzo segnale STLN_OUT sincronizza l' inizio di ogni lettura di riga, in altre parole tra due suoi impulsi fuoriescono i dati di una completa riga. Inoltre, per permettere al sensore di leggere le *vertical blanking rows*, è importante fornire degli aggiuntivi impulsi dello stesso segnale.

In snapshot mode, il sensore può catturare una singola o una sequenza di immagini. In questa modalità esso accetta un impulso in ingresso (EXPOSURE) che determina l' inizio del tempo di esposizione e al suo completamento, è immediatamente seguito dal readout con i relativi segnali in uscita di sincronismo per righe e frames.

I registri pre-programmabili attraverso il protocollo seriale e che contengono il valore del periodo d' integrazione sono R0x0B e R0xD2, mentre il frame rate può solo essere controllato cambiando il periodo del treno di impulsi di EXPOSURE forniti dal DSP.

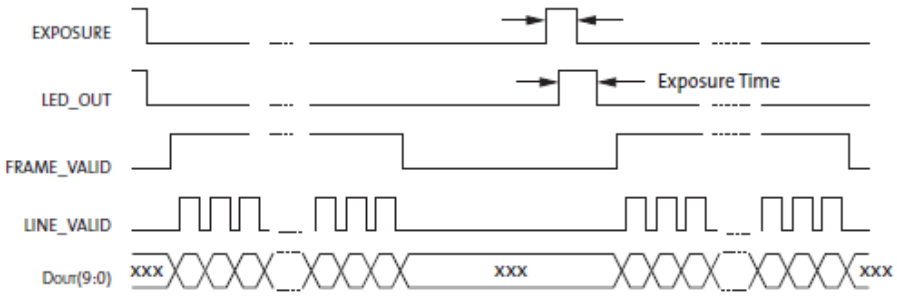


Figura 3.20: Modalità snapshot

I registri che devono venire settati per avere la configurazione di funzionamento descritta in precedenza è essenzialmente uno che è evidenziato nella tabella sotto in grigio:

FUNZIONALITA'	DESCRIZIONE	REGISTRO (bits)	DEFAULT (decimale)	MODIFICA
INIZIO COLONNA/RIGA	La prima colonna/riga da leggere	0x01 (9:0) 0x02 (8:0)	1 4	
ALTEZZA/LARGHEZZA FINESTRA	Il numero di righe/colonne da leggere nell' immagine	0x03 (9:0) 0x04 (8:0)	480 752	
BLANKING ORIZZONTALE/VERTICALE	Numero di colonne (righe)di blank in una riga (in un frame) . In modalità sequenziale è disabilitato, ma nel progetto sarà Vblanking = exposure + 6 righe.	0x05 (9:0) 0x06 (14 :0)	94 45	
SCAN mode	La modalità del readout: progressiva , interlacciata a due *fields* o singolo *field*	0x07 (2:0)	0	
MODALITA' DI FUNZIONAMENTO	**Obbligatoriamente da impostare in quanto non siamo in modalità slave mode (default), né in master ma in exposure mode. (cioè l' inizio del frame è campionato dall' impulso EXPOSURE)**	**0x07 (4:3)**	**0**	**3**
COARSE SHUTTER WIDTH1 /2	Utilizzato in HDR il numero di righe dopo le quali avviene il primo(secondo) ginocchio nella curva	0X08 (14:0) 0X09(14:0)	443 473	
MODALITA' DI LETTURA	In questo particolare registro si potrebbero impostare; Row Bin (lettura di due/quattro righe di pixels per riga d' uscita con la riduzione dell' immagine verticale di un fattore 2/4 senza influenzare data rate o pixel clock, ma solo il frame rate incrementato di 2/4), Column Bin (riduzione della dimensione orizzontale dell' immagine di un fattore 2/4 che si ripercuote su data rate e pixel clok ridotti della metà /di un quarto di master clock), Row/Column Flip (il readout per riga/colonna inizia dal basso verso l'alto/destra a sinistra (mirrored))	0X0D (5 :0)	0x00	
ADC COMPANDING MODE	Possibilità di avere il companding da 12 a 10 bit	0x1C	0x0302	
V1/V2/V3/V4 CONTROL	Tensioni di controllo per un high dynamic range (HDR) che si ottiene con il controllo dei livelli di saturazione del pixel durante il tempo di esposiizione	0X31, 0X32,0X33,0X34	0X0027,0X001A,0X005,0X0003 = 2.54V,2.23V,1.2V,0.8V	

ANALOG GAIN	A patto che il sensore sia in AGC mode, per migliorare la luminosità di un immagine (è consigliabile prima cambiare l' esposizione e poi portare un eventuale aumento di guadagno solo quando il valore dell' esposizione ha raggiunto il suo massimo valore). Il guadagno va da un x1 a x4 con uno step di 0,0625 Vi è la disponibilità anche di un' attenuazione dello 0,75X	0x35	16	
DIGITAL GAIN	Il guadagno logico divide l' immagine in 25 tiles con guadagno e dimensione per ciascuno regolabili dai rispettivi registri	0x80 – 0xA4		
SOFT RESET	Usato per resettare la logica digitale non includendo l' interfaccia del protocollo I2C	0X0C (0)		
TENSIONE DI RIFERIMENTO	Il valore della tensione di riferimento (da 1 a 2.1 V) ha un incremento di 0.1 V da 1V a 1.6V con 1LSB = 1mV.	0x2C (2:0)	4 (= 1.4V)	

Figura 3.21: Principali registri del sensore d' immagine

3.3.3 Memoria FLASH

La memoria SPI Flash integrata è la S25FL116K da 2Mbyte, con 3,3V forniti dall' alimentazione dedicata attraverso il pin VDDFLASH senza la necessità di una tensione maggiore per la cancellazione. Include anche il supporto per la protezione software dalla scrittura, per cancellazioni veloci e programmazione dei singoli byte. La Flash si connette al sistema host attraverso una comunicazione di tipo seriale sfruttando Serial In (SI) e Serial Output(SO) per trasferire i dati; riesce a raggiungere frequenze di clock di 108MHz con un singolo data path di 13.5 Mbytes/s ed è in grado di programmare pagine di 256 bytes ciascuna. Il processore è in grado di pilotarla in una delle seguenti modalità: MODE 0 (polarità del clock = 0 e fase del clock = 0) o MODE 3 (polarità del clock = 1 e fase del clock = 1) con la differenza sulla polarità del clock appunto che varia in standby per le due casistiche non trasferendo alcun dato.

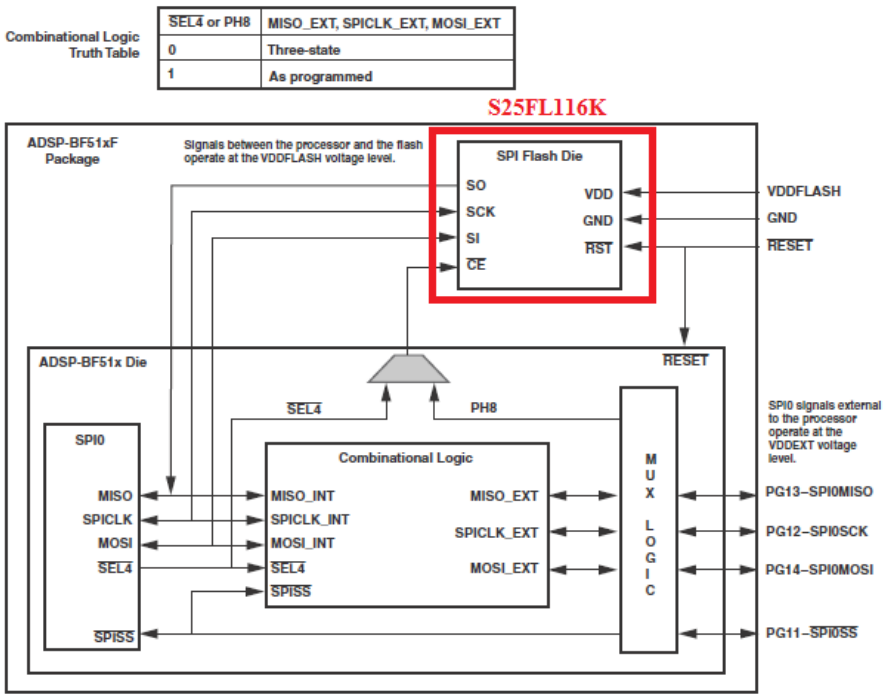


Figura 3.22: Collegamento della memoria Flash nel processore

La memoria è suddivisa in unità cancellabili chiamate settori di dimensione uniforme di 4kbytes.

Sector Size (kbyte)	Sector Count	Sector Range	Address Range (Byte Address)	Notes
4	512	SA0	000000h-000FFFh	Sector Starting Address
		:	:	—
		SA511	1FF000h-1FFFFFh	Sector Ending Address

Figura 3.23: Suddivisione della Flash

Ci sono inoltre quattro particolari registri a 256-bytes (Security Registers) che possono essere usati per salvare delle informazioni che devono essere permanentemente protette.

Il funzionamento della memoria FLASH è diverso da una memoria ad accesso casuale. Infatti la sua tecnologia permette la scrittura di soli ‘0’, quindi prima delle scrittura di un dato, la locazione di memoria deve essere cancellata, ossia i suoi bit devono essere tutti posti a ‘1’ e poi può essere scritta. Infatti per l’utilizzo della memoria sono previsti diversi tipi di comandi d’interfaccia, attivati presso la memoria mediante la trasmissione di codici in sequenze corrette in locazioni determinate, secondo le tabelle rappresentate in figura 3.21 – 3.22. Tramite tali sequenze di codici, vengono attivati i vari comandi della memoria e poi quindi possono essere letti, cancellati o scritti dei blocchi di memoria.

Command Name	BYTE 1 (Instruction)	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6
Read Status Register-1	05h	SR1[7:0] (2)(4)				
Read Status Register-2	35h	SR2[7:0] (2)(4)				
Read Status Register-3	33h	SR3[7:0] (2)				
Write Enable	06h					
Write Enable for Volatile Status Register	50h					
Write Disable	04h					
Write Status Registers	01h	SR1[7:0]	SR2[7:0]	SR3[7:0]		
Set Burst with Wrap	77h	xxh	xxh	xxh	SR3[7:0] (3)	
Page Program	02h	A23–A16	A15–A8	A7–A0	D7–D0	
Sector Erase (4 kB)	20h	A23–A16	A15–A8	A7–A0		
Block Erase (64 kB)	D8h	A23–A16	A15–A8	A7–A0		
Chip Erase	C7h/60h					
Erase / Program Suspend	75h					
Erase / Program Resume	7Ah					

Figura 3.24: Set di comandi (parte 1)

Command Name	BYTE 1 (Instruction)	BYTE 2	BYTE 3	BYTE 4	BYTE 5	BYTE 6
Read Data	03h	A23–A16	A15–A8	A7–A0	(D7–D0, ...)	
Fast Read	0Bh	A23–A16	A15–A8	A7–A0	dummy	(D7–D0, ...)
Fast Read Dual Output	3Bh	A23–A16	A15–A8	A7–A0	dummy	(D7–D0, ...) (1)
Fast Read Quad Output	6Bh	A23–A16	A15–A8	A7–A0	dummy	(D7–D0, ...) (3)
Fast Read Dual I/O	BBh	A23–A8 (2)	A7–A0, M7–M0 (2)	(D7–D0, ...) (1)		
Fast Read Quad I/O	EBh	A23–A0, M7–M0 (4)	(x,x,x,x, D7–D0, ...) (5)	(D7–D0, ...) (3)		
Continuous Read Mode Reset (6)	FFh	FFh				

Figura 3.25: Set di comandi (parte 2)

In tali condizioni, la scrittura di una parola da 16 bit richiede circa 17.5 µs (15µs per scrivere il primo byte e 2.5 per ogni byte aggiuntivo) sul canale MOSI della SPI0 del processore e per memorizzare un intero frame ci vorrebbero circa 1,8 secondi (a fronte dei soli 16,66 ms per la sua acquisizione), mentre l'operazione di scrittura o lettura seriale avviene ad una frequenza di clock SCLK di 108MHz. Per questi motivi la memoria FLASH risulta essere inutilizzabile per l'acquisizione dell'immagine dalla porta video.

I consumi di corrente e di potenza sono differenti nelle varie fasi di utilizzo, come riassume la seguente tabella:

Fase di utilizzo	Corrente	Potenza
Lettura	12 mA	10 mW
Scrittura/Cancellazione	20 mA	20 mW
Standby	15 µA	TBD
Power-down	2 µA	TBD

Figura 3.26: Tabella riassuntiva dei consumi

Inoltre bisogna calcolare quanto dissipato in potenza dall'interfaccia tra il processore e la memoria FLASH; per essere conservativi, essa prevede un tempo di accesso di 10 ns e quindi utilizzeremo la frequenza corrispondente di 108 MHz per . Infine sappiamo che in ingresso una capacità è di 8 pF su SCK, mentre il bus dati MOSI/MISO vede lo stesso valore di capacità nelle fasi di lettura /scrittura.

Ecco quindi il consumo delle due fasi:

- Lettura: $P = 108\ MHz * 8pF * (3,3V)^2 = 9,4\ mW \approx 10\ mW$
- Scrittura: $P = 2* (108\ MHz * 8pF * (3,3V)^2) = 18,\ 8\ mW \approx 20\ mW$

3.3.4 Memoria SDRAM

La memoria SDRAM MT48LC4M16A2TG, è una memoria sincrona PC100 e PC133 compliant; per questo prevede una frequenza di clock di 100MHz, quella appunto settata nel clock del processore per le periferiche (SCLK) . Alimentata a 3,3V come processore e Flash, questa memoria è di 8Mbyte, con parallelismo del data bus di 16 bit, divisa in 4 banchi da 2 Mbyte, ossia 1Mword. Per la selezione dei banchi di memoria, il processore utilizza due bit del bus indirizzi non utilizzati per l'indirizzamento vero nelle diverse possibili fasi di funzionamento, precisamente il bit 18 e il bit 19. Inoltre ogni banco è suddiviso in 4096 righe e 256 colonne; infatti, il processore per poter accedere ad una qualsiasi locazione di memoria, utilizzerà 12 bit (2^{12} = 4096) del bus indirizzi per mandare il valore della riga che vuole utilizzare e, nel ciclo successivo vengono utilizzati i primi 8 bit (2^{8} = 256) per mandare il valore della colonna puntata. In particolare però, l'undicesimo bit (chiamato A10 perché la numerazione inizia da A0) degli indirizzi non è comandato dall'undicesimo bit dell'address bus, come tutti gli altri segnali d'indirizzo ma è comandato da un pin distinto chiamato SA10 del controller del processore. Questo segnale infatti normalmente funge da indirizzo ma viene anche utilizzato dalla memoria durante il comando di PRECHARGE" per capire se tutti i banchi sono stati prevaricati (SA10 a livello logico alto) o solo quello indicato (SA10 a '0') tramite i due bit appositi.

Inoltre l'interfaccia prevede due segnali DQM, uno chiamato DQML che si riferisce ai primi 8 bit di dato mentre il secondo chiamato DQMH che si riferisce al byte più significativo del bus dati. Essi se campionati a livello logico alto durante una fase di lettura, il bus dati viene messo in alta impedenza, impedendo l'uscita del dato e causando 2 cicli di clock di ritardo, mentre se ciò avviene in fase di scrittura il dato presente sul bus viene ignorato e non memorizzato.

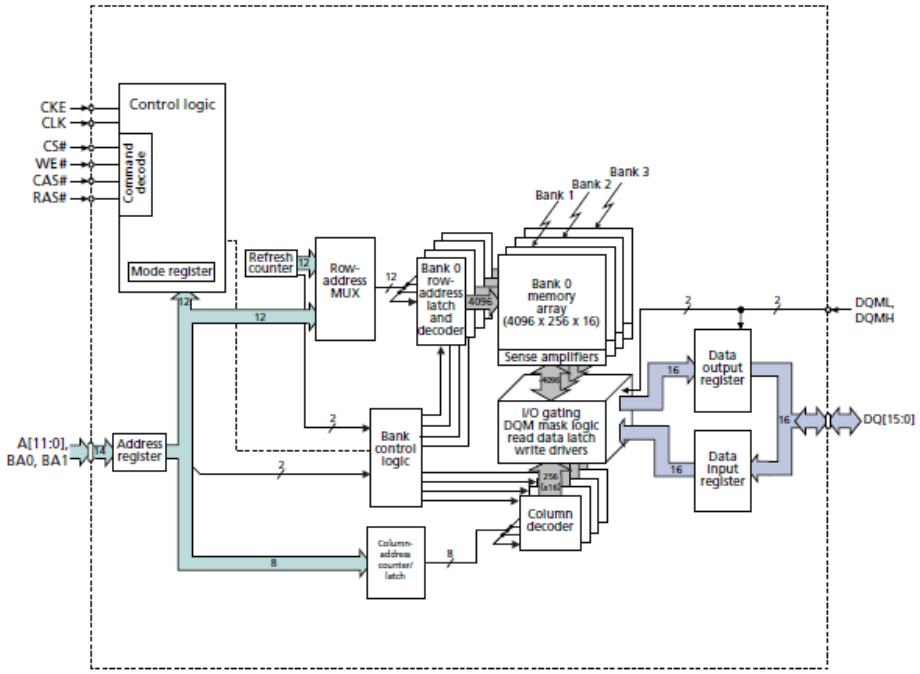


Figura 3.27: Diagramma a blocchi funzionale

Infine l'interfaccia presenta segnali di controllo di scrittura e di accesso alle righe (SRAS) e alle colonne (SCAS) attivi bassi, che quando attive indicano che l'indirizzo presente sul bus è riferito alla riga o alla colonna. Inoltre è presente un write enable (SWE) per definire la modalità di accesso, se in lettura o scrittura.

Inoltre è presente il segnale di clock per la sincronizzazione dei segnali e il chip select, il quale quando campionato alto rende la memoria insensibile a qualsiasi commutazione su qualsiasi suo pin.

Infine troviamo un segnale fondamentale denominato clock enable (SCKE), il quale non solo abilita o disabilita il funzionamento del segnale di sincronizzazione principale, ma in questo modo porta la memoria in modalità di power down.

L'interfaccia completa del controller sincrono del processore e la memoria SDRAM appena spiegata è la seguente:

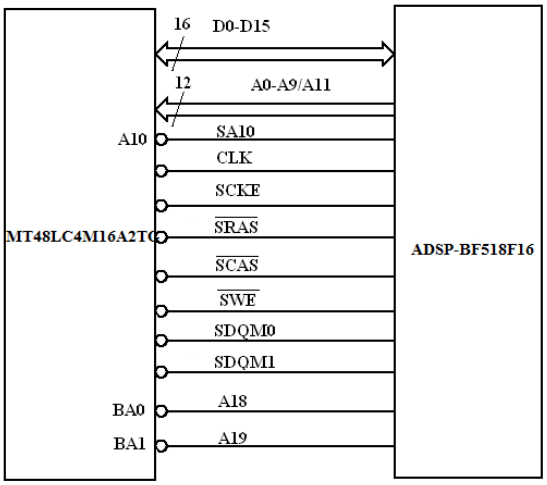


Figura 3.28: Interfaccia tra processore e memoria SDRAM

La memoria prevede all'accensione una fase di inizializzazione, dove vengono impostati i parametri principali, quali il CAS latency, che fornisce il ritardo in termini di colpi di clock dal comando, per esempio di lettura, e la disponibilità del dato sul bus e la modalità di accesso, singola o a burst; infatti la memoria prevede la possibilità di accedere a burst di 1,2,4,8 o l'intera colonna (256), in modo sequenziale o interfacciato, ossia che ad ogni comando di lettura o scrittura vengono letti o scritti 2,4,8 o 256 (nel nostro caso) locazioni di memoria, mandando solo l'indirizzo iniziale del burst. Il controller per memorie SDRAM del processore supporta solo accessi a burst pari a 1; in fase di inizializzazione del mode register verrà settato il burst length a 1 e il burst type come sequenziale, sebbene essendo a lungo 1 word questo settaggio è ininfluente. Inoltre la memoria adottata, prevede due cicli di clock come CAS latency. Le varie tempistiche sono state definite via software come verrà illustrato in seguito.

Particolarità fondamentale della memoria dinamica è la necessità di un continuo 'refresh' di ogni colonna ogni 15,625µs ossia di una riga ogni 64ms. Questo avviene automaticamente mediante un

contatore interno alla memoria che provvede al continuo refresh delle colonne di memoria. Questo causa un assorbimento di 3 mA con il controller sincrono del processore attivo, quindi con la linea di clock funzionante, altrimenti assorbe 1 mA se il sistema è acceso, ma la SDRAM e il controller sul Blackfin non sono utilizzati.

Quindi per il calcolo della corrente assorbita e della potenza dissipata dal componente, si devono distinguere le due modalità principali, quella operativa e quella di standby:

Fase di utilizzo	Corrente	Potenza
Lettura/Scrittura	115 mA	379.5 mW
Standby Active Mode	45 mA	148.5 mW
Standby	2 mA	6.6 mW

Figura 3.29: Tabella riassuntiva dei consumi per SDRAM

In realtà quella di standby può essere di due tipi, ossia quella a clock spento riportata nella tabella precedente e quella di stato di temporaneo inutilizzo tra accessi consecutivi, causati, per esempio, da elaborazione dei dati acquisiti da parte del processore. Noi supponiamo di trascurare tale fase e di considerare la memoria sempre operativa, in lettura o scrittura, nelle fasi in cui accediamo direttamente alla memoria; tale strategia comporta calcoli conservativi, infatti nei calcoli dei consumi di energia considereremo operativa la memoria per tutto il tempo in cui viene utilizzata, trascurando quindi i vari tempi morti sebbene di dimensioni ridotte ma sicuramente presenti. Comunque come si vedrà in seguito, potrebbe servire il consumo in questa fase perché non tutti gli accessi alla memoria SDRAM saranno noti al nostro programma, come verrà spiegato nella sezione software.

Come è stato accennato in precedenza, la memoria adottata è di 8Mbyte, in modo da poter avere 4 locazioni di memoria da 2 Mbyte ciascuno contigue, dove poter mettere temporaneamente 4 fotogrammi prima di un eventuale compressione JPEG; questo può essere utile nel caso si voglia scattare 4 foto una subito dopo l'altra, quindi nello stesso ciclo di funzionamento, magari per fotografare qualcosa di particolare. Si ricordi che ogni immagine occuperebbe circa 721,92 Kbytes e la metà dopo il write-back in seguito al troncamento.

Si parla di locazioni da 2Mbyte contigue perché il controller della memoria sincrona supporta un indirizzamento per memorie da 16Mbyte a 128 Mbyte, quindi prevede l'utilizzo di 13 linee di indirizzo per le righe e 2 per l'accesso ai banchi interni della memoria. Si poteva anche adottare una memoria da 16 Mbyte ma il consumo sarebbe notevolmente aumentato, in quanto le celle su cui passa la linea di clock o su cui eseguire il refresh sono il doppio. Con la nostra memoria si va ad utilizzare entrambi i segnali dei banchi di memoria ma non viene utilizzata una linea di indirizzo per le righe. Qui di seguito viene illustrato il particolare indirizzamento adottato per la memoria SDRAM:

Per capire meglio come è stato cambiato l' indirizzamento e si è commesso una memoria di dimensione più piccola all' ADSP-BF518F16, facciamo riferimento al mapping d' indirizzamento interno nel caso di una mappatura con la SDRAM più piccola e cioè a 16MBytes:

BANK SIZE (M byte) EBSZ bits	Col. Address Width (CAW) EBCAW bits	Page Size (K Byte)	Bank Address	Row Address	Column Address	Byte Address
16	8	0,512 KB	IA [23 : 22]	IA [21 : 9]	IA [8 : 1]	IA [0]
8	8	0,512 KB	IA [23 : 22]	IA [20 : 9]	IA [8 : 1]	IA [0]

Figura 3.30: Tabella dell' indirizzamento interno

L' indirizzamento delle righe IA [21] non viene utilizzato creando delle zone non contigue per mappare l' intera SDRAM, perciò abbiamo 4 blocco di 2 Mbytes e poi altrettanti non usati dal microprocessore.

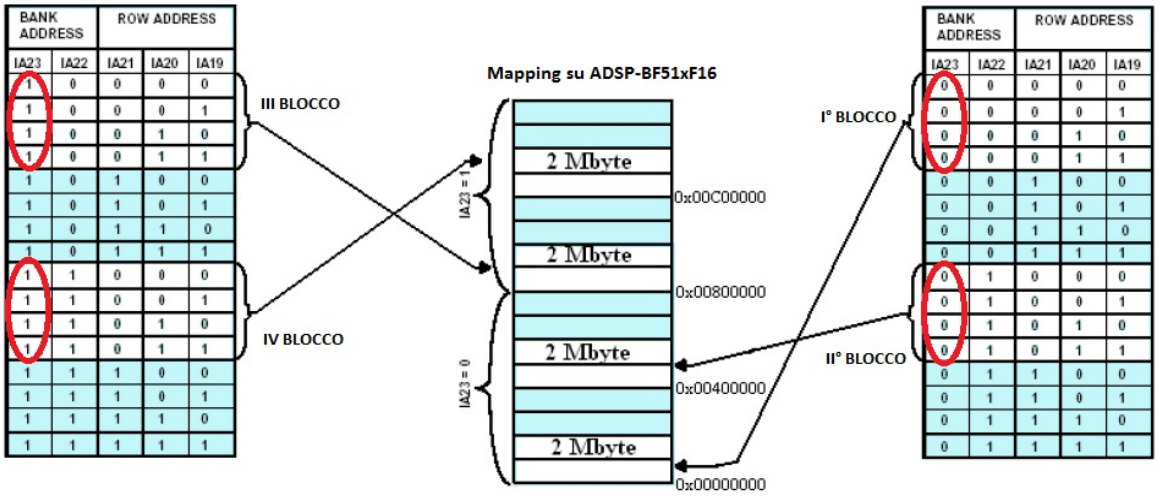


Figura 3.31: Mapping del Blackfin con Sdram

Concludendo, la potenza assorbita dal processore Blackfin nell'interfaccia con la memoria SDRAM risulta molto complicato da calcolare, dato il complesso funzionamento della memoria stessa. L'Analog Devices fornisce un datasheet per questo e da esso si sa che la potenza assorbita a questa frequenza è di circa 180 mW, in ottica molto conservativa.

3.4 Interfacce

3.4.1 PPI

Come descritto in precedenza il processore Blackfin è dotato di periferiche di diverso tipo e il sensore d' immagine necessita di alcune di esse, la più importante sicuramente nell' interfaccia sarà proprio la PPI.
La PPI è una porta half-duplex bidirezionale che può contenere fino a 16 bits di dato con un pin dedicato al clock (PPI_CLK: in ingresso generato esternamente) e tre pins per i frame syncs multiplexati. Il più alto throughput del sistema è raggiungibile con 8 bit di dato in quanto potrebbero essere impacchettati in una singola word di 16 bit: in questo caso, il primo campionamento è posizionato negli 8 bits meno significativi (LSBs). Naturalmente per quanto detto finora, i dati provenienti dal sensore sono 10 bits digitali quindi non è possibile nessun *package*. Le caratteristiche principali sono:

- Porta parallela bidirezionale e half duplex;
- Supporta fino a 16 bits di dato;
- Programmabile polarità del clock e del frame sync;
- Supporta ITU-R 656;
- Generazione dell' interrupt per overflow o underrun;

Inoltre i dispositivi periferici che si possono interfacciare alla porta PPI possono essere:

- Convertitori ADC e DAC;
- Sensori CMOS;

Il diagramma a blocchi della PPI è raffigurato sotto:

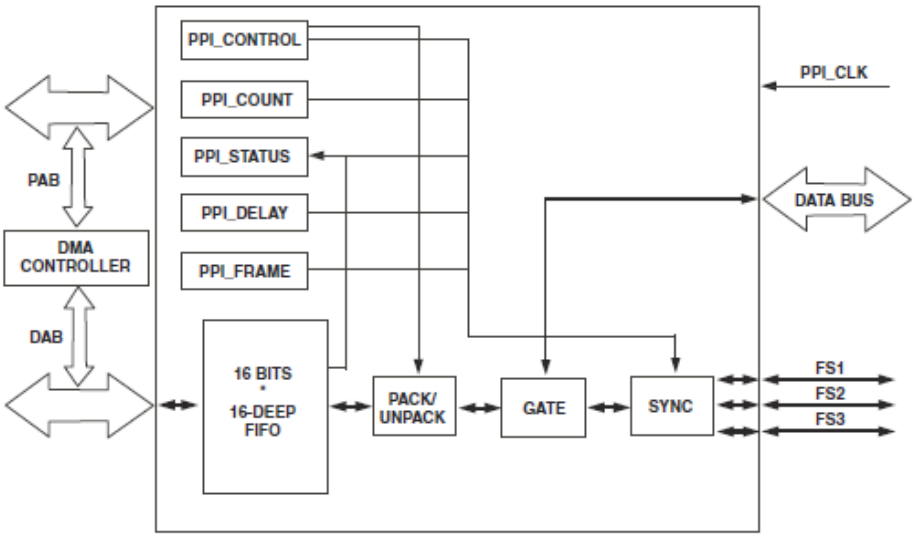


Figura 3.32: Diagramma a blocchi PPI

Il clock PPI_CLK che non può essere generato internamente, non è *free-running* in quanto sono presenti dei cicli addizionali di latenza prima che il dato venga ricevuto o trasmesso: almeno 2 cicli. Il clock inoltre non viene fornito al modulo, ma può servire per sincronizzare uno o più GP timers per lavorare insieme alla PPI.

Il processore supporta l' ITU-R656, ma il sensore no: per questo motivo tra le varie modalità di funzionamento è stata scelta quella evidenziata in tabella sotto:

GP PPI Mode	PPI_FS1 Direction	PPI_FS2 Direction	PPI_FS3 Direction	Data Direction
RX mode, 0 frame syncs, external trigger	Input	Not used	Not used	Input
RX mode, 0 frame syncs, internal trigger	Not used	Not used	Not used	Input
RX mode, 1 external frame sync	Input	Not used	Not used	Input
RX mode, 2 or 3 external frame syncs	Input	Input	Input (if used)	Input
RX mode, 2 or 3 internal frame syncs	Output	Output	Output (if used)	Input
TX mode, 0 frame syncs	Not used	Not used	Not used	Output
TX mode, 1 external frame sync	Input	Not used	Not used	Output
TX mode, 2 external frame syncs	Input	Input	Not used	Output
TX mode, 1 internal frame sync	Output	Not used	Not used	Output
TX mode, 2 or 3 internal frame syncs	Output	Output	Output (if used)	Output

Figura 3.33: Modalità general purpose di funzionamento PPI

1,2,o 3 External Frame Syncs : i frame syncs sono segnali level-sensitive e come si può vedere in figura sotto, la modalità prevede che siano usati tutti e tre i segnali (3-sync mode) o solo 2 (in quanto è possibile anche un sync 2 mode non abilitando PPI_FS3); la si usa per applicazioni video che usano determinate temporizzazioni (HSYNC = PPI_FS1, VSYNC = PPI_FS2 e FIELD= PPI_FS3) come specificato nella *ITU-R 601 reccomendation*.

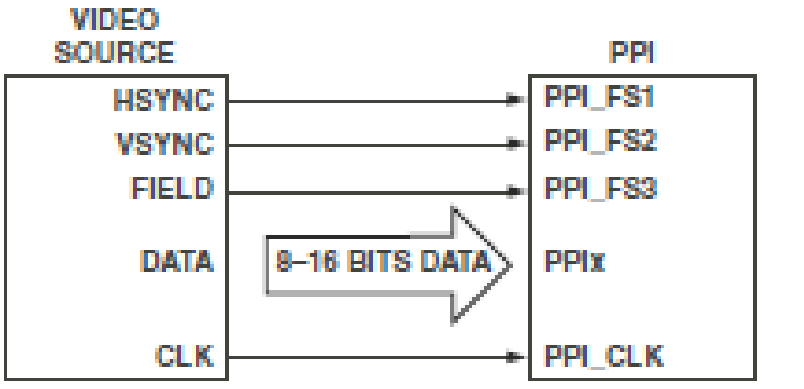


Figura 3.34: Connessione tipica sensore - PPI

Nel modello di programmazione, ci avvarremo anche della DMA attraverso una modalità di sincronismo con interrupt che sancisce appunto l' acquisizione dell' intero frame e la possibilità di disabilitare la PPI. In realtà, saranno utilizzati due timers per generare il master clock e il segnale exposure per avviare l' inizio del tempo di integrazione, ma non verranno presi in rassegna approfonditamente essendo il loro

lavoro limitato a generare un clock e un impulso per il sensore. Entrambi saranno poi disabilitati dalla stessa service interrupt routine della porta parallela del Blackfin.

3.4.2 DMA

La DMA (direct memory access) dispone di un controller in grado di gestire due tipi di trasferimento:

- Tra una memoria e una periferica (come nel caso la PPI);
- Tra una memoria e una memoria attraverso due canali distinti in lettura e scrittura;
- Tra una periferica off-chip e una memoria;

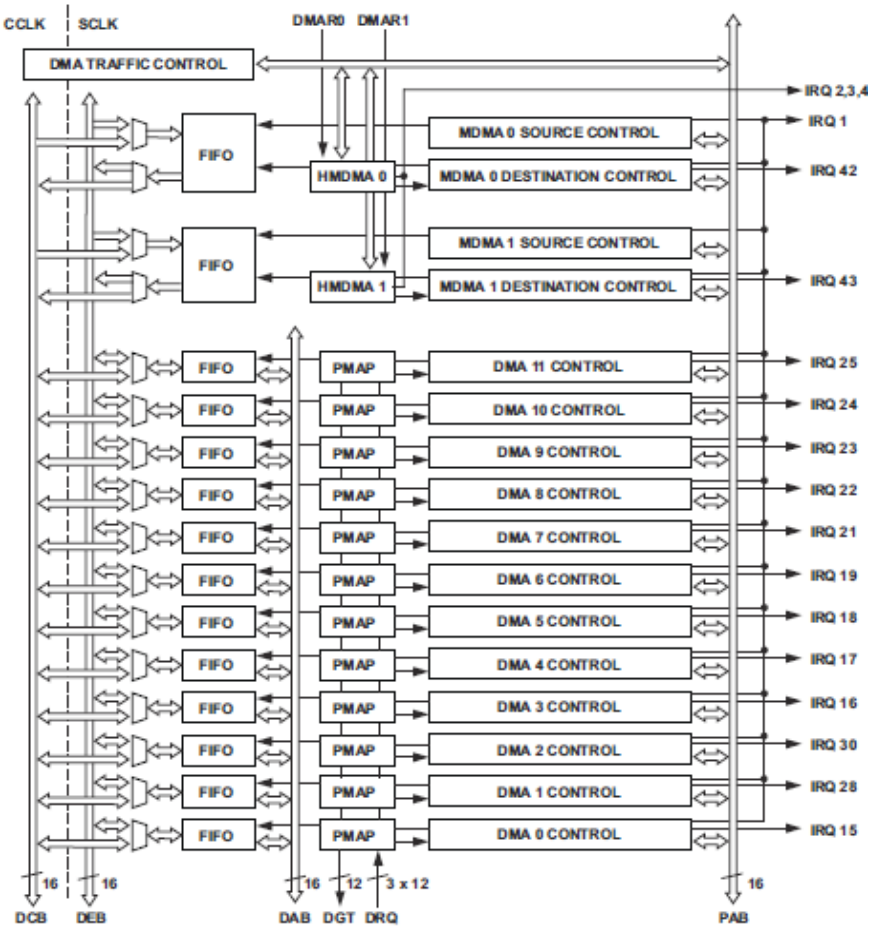


Figura 3.35: Struttura della DMA

Nel nostro caso, ci concentreremo unicamente sui trasferimenti che riguardano le periferiche ed in particolare sul cammino dei dati che passerà attraverso il DMA Access Bus (DAB) e per il DMA External Bus (DEB) che si connetterà direttamente all' EBIU. L' interfaccia esterna di questo modulo passa sempre attraverso il bus esterno senza comunicare direttamente. Il clock utilizzato sarà quello di sistema, cioè quello impostato nel PLL e determinerà il *rate* con il quale saranno instradati verso le memorie i dati del sensore dopo essere passati dalla PPI.

La peripheral DMA si caratterizza per i 12 canali che permettono trasferimenti in maniera indipendente dalle varie periferiche del processore e dei quali, l' utente ha un pieno controllo potendoli mappare secondo il grado di priorità:

DMA0>>DMA1>>…>>DMA11

Questi trasferimenti possono essere di due tipi:

- REGISTER-BASED: dove la DMA abilita il processore a programmare direttamente i registri di controllo per iniziare un trasferimento tramite quest' ultima; al compimento del tutto, i registri di controllo possono essere automaticamente ripristinati ai loro originali valori per ulteriori trasferimenti.
- DESCRIPTOR BASED: il trasferimento prevede che un insieme di parametri siano salvati all' interno della memoria per cominciare una determinata *work unit*; naturalmente, sono possibili delle concatenazioni in operazioni di questo tipo ed un canale può appunto essere programmato automaticamente per introdurre ed iniziare un altro trasferimento al termine della precedente sequenza.

Visto la semplicità del progetto, si è optato per la programmazione diretta dei registri supportati da un funzionamento a due dimensioni che sarà approfondito nel capitolo software. Una volta terminato il passaggio all' EBIU, il controller sarà disabilitato automaticamente in quanto in modalità STOP MODE, ossia l' operazione avviene una sola volta e trasferite le words desiderate e pre-impostate nei giusti registri, si riterrà completato.

3.4.3 EBIU

Una volta giunti all' EBIU (External Bus Interface Unit), i dati saranno gestiti dal controller sincrono per il salvataggio nella SDRAM. Questa unità assiste le richieste della memoria esterna da parte del core o dal canale DMA con priorità determinata dal controller EBIU e in generale, attraverso l' indirizzo della stessa richiesta si stabilisce se sarà gestita poi dall' adeguato controller. La DMA è in grado di gestire un' alta capacità di spostamento dei dati e quando usata insieme all' EBIU, quest' ultima funzionalità può essere usata per interfacciarsi a dispositivi ad alta velocità come le FIFO.
L' EBIU è campionata con il clock del sistema (SCLK) come del resto lo saranno tutte le memorie sincrone che ad essa si collegheranno. Nello spazio memoria del sistema, vi è una zona che è dedicata al supporto della SDRAM (da 0x0000 0000 fino a 0x2000 0000 che corrisponde ad un range di dimensioni da 16M byte a 128 Mbyte). Allo stesso modo ci sono 4 zone riservate alla memoria asincrona per essere programmate ciascuna in maniera indipendente, che non verranno prese in rassegna essendoci limitati solo all' utilizzo della SDRAM.

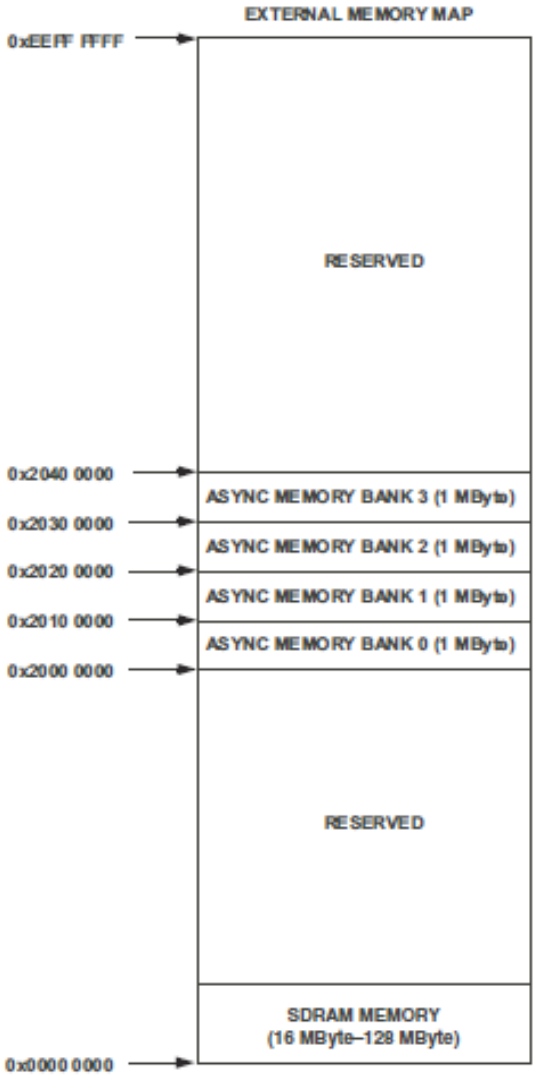


Figura 3.36: Mappatura dell' ADSP-BF518F16 per memorie esterne

L' EBIU funziona come uno slave attraverso i tre bus interni al processore:

- EAB: nel momento in cui bisogna servire una richiesta del bus esterno da parte del core;
- DEB: guidato dal controller DMA, nel momento in cui bisogna servire una richiesta del bus esterno da qualche suo canale (sarebbe il caso del nostro progetto);
- PAB: nel momento in cui bisogna servire una richiesta da parte del core per conto del MMR del sistema che non necessitano di un arbitraggio e sono indipendenti dagli altri bus;

Per capire meglio la sua struttura , nella figura sotto è evidenziato il controller che andremo a programmare, SDC, con i relativi pads utili al controllo dello stesso:

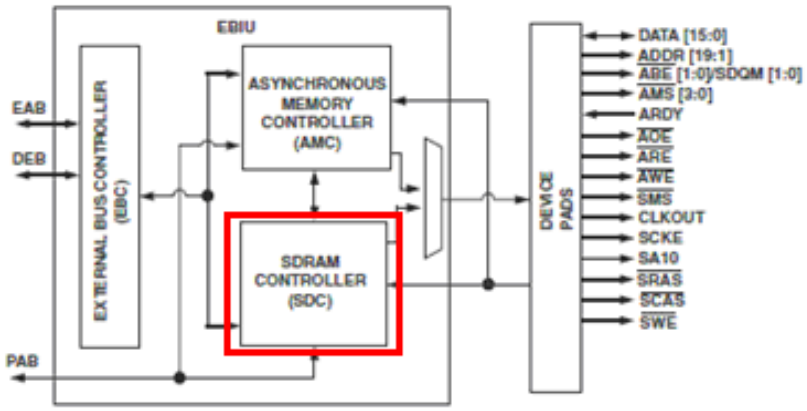


Figura 3.37: Diagramma a blocchi EBIU

Il controller sincrono abilita il processore per il trasferimento dati da/verso una memoria sincrona DRAM con una frequenza massima specificata nel datasheet, che essendo stata impostata una VDDINT a 1.4V sarà al massimo di 100MHz ; inoltre, dispone di una semplice interfaccia con un unico banco esterno di dimensioni che può variare da 64 Mbit a 512 Mbit e con configurazione x4, x8 e x16 fino ad un massimo totale di capacità di 128Mbytes.
Le sue caratteristiche principali sono:

- Data bus di 16 bit in I/O e relative alimentazioni di 1,8 2,5 o 3,3 Volts;
- Supporta fino a 128 Mbytes di SDRAM in un unico banco esterno;
- Tipi di 64, 128, 256 e 512M bit con I/O di x4, x8 e x16;
- Supporta dimensioni di pagina di 512 byte, 1K, 2K e 4K byte;
- Supporta operazioni multi-banco all' interno dell' SDRAM;
- SDC usa la regola *open-page* dove qualsiasi pagina è chiusa sole se avviene un nuovo accesso in un'altra pagina dello stesso banco;
- Sfrutta un programmabile refresh counter per coordinare le variazioni della frequenza di clock e la velocità di refresh richiesta dalla SDRAM;
- Supporta la modalità di auto-refresh per risparmiare potenza (utile quando il processore sarà nello stato di ibernazione);

La porta sarà utile per permettere ai dati di raggiungere la SDRAM, per regolare le sue tempistiche e, a tal proposito, questo sarà fatto attraverso un' approfondita analisi della memoria per interfacciarsi in maniera giusta al controller (ripreso in maniera dettagliata nella parte software). La relazione che intercorre tra la porta e il dispositivo esterno è molto stretta e necessita una configurazione adeguata anche alle varie modalità di funzionamento del processore: un esempio su tutti, può essere l' auto-refresh nel momento in cui in un' ottica di basso consumo, decidiamo di ibernare il processore : in

questa situazione importante sarà programmare i registri del PLL includendo la necessità di non perdere i dati salvati nella SDRAM.

3.4.4 SPI

La porta Serial Peripheral Interface (SPI) non sarà utilizzata come di solito avviene per una comunicazione con un dispositivo esterno al processore, ma adempirà alle sue funzioni di trasferimento dati in flash dei soli dati prodotti dall' elaborazione del core o di frame attraverso DMA. Per essere più chiari, sarà fatto solo un piccolo accenno a questa interfaccia in quanto, come ripetuto più volte e già trattato ampiamente nel paragrafo relativo alla Flash, è integrata nel processore e collegata al modulo SPI0.

La porta SPI offre un interfaccia I/O in grado di comunicare con una vasta gamma di dispositivi con periferiche compatibili SPI; è a 4 fili di cui due per i dati, un segnale di selezione del dispositivo ed il clock, full-duplex seriale sincrona che supporta ambienti multi-master (attraverso l' uso di open drain per evitare la contesa dei dati) oltre la modalità master/slave .

Come vedremo nella parte relativa ai moduli Aramis, i suoi stessi pins saranno utili per comunicare con gli altri sottosistemi presenti sul satellite stesso attraverso le slots del Modulo_A e Modulo C (moduli generici per indicare la disponibilità del sistema sensore di sole a colloquiare attraverso quei determinati canali)

Il diagramma a blocchi del modulo SPI è:

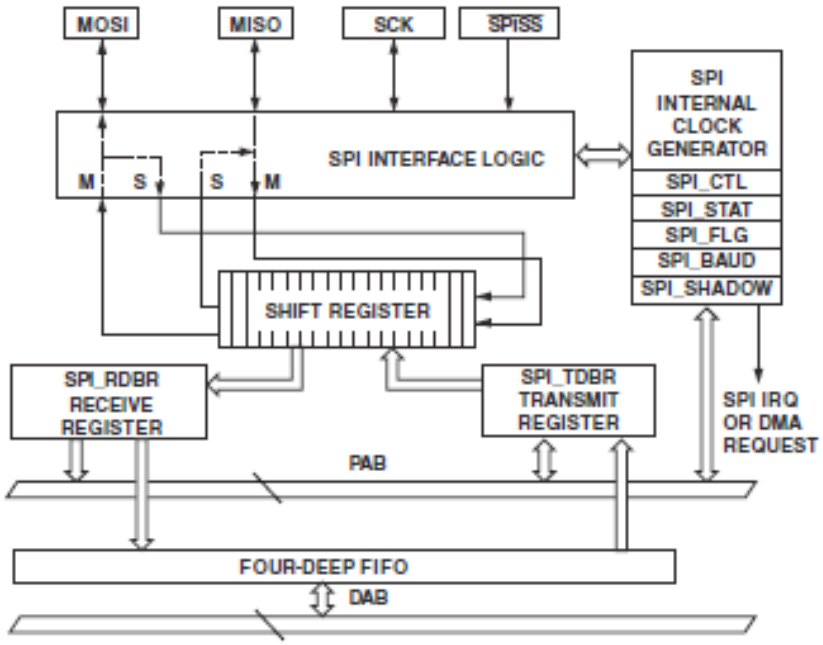


Figura 3.38: Diagramma a blocchi SPI

Sostanzialmente, le quattro linee principali in questo protocollo:

- Spi Clock Signal (SCK): clock seriale pilotato dal master che controlla il rapporto con i quali i dati sono trasferiti;
- Master-Out;Slave-In Signal (MOSI): bidirezionale (se il processore configurato come master, il pin trasmette fuori i dati; viceversa avviene il contrario)
- Master-In ;Slave-Out Signal (MISO): bidirezionale (se il processore configurato come master, il pin ricevei i dati; viceversa avviene il contrario);
- SPI Slave Select Input Signal (SPISS): segnale di selezione in ingresso (attivo basso) per gli slaves da parte del master (nel nostro caso il processore) ricalcando la funzione del chip select.

In generale , la connessione tipica che si potrebbe avere con un altro microcontrollore, come l' MSP430, sarebbe:

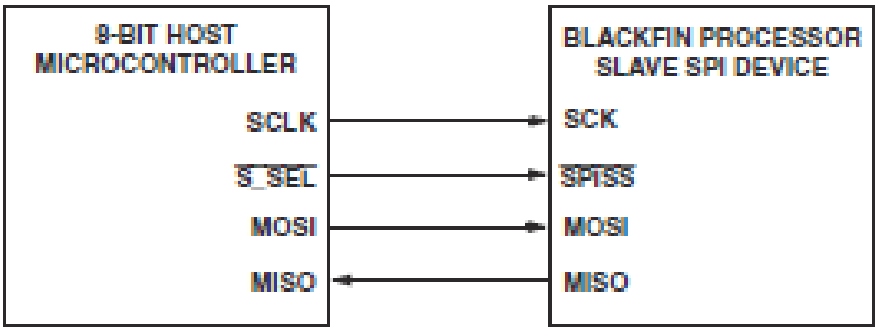


Figura 3.39: Esempio di interfaccia SPI tra due processori

3.5 Alimentazioni

Come spiegato in precedenza, i segnali di alimentazione per il funzionamento dell'intera scheda sono forniti dalla scheda Power Managment, presente all'interno di Aramis. Le tensioni richieste dai componenti adottati sulla scheda sono quattro:

- 3,3V necessari per la SDRAM , per il sensore d' immagine, per le periferiche del DSP (includendo la Flash integrata) e per tutti i restanti componenti integrati digitali utilizzati sulla scheda
- 5V necessari per il funzionamento degli operazionali (MAX4091) all' interno dei current sensors;
- 12V necessari per arrivare, attraverso un regolatore switching, a fornire l' alimentazione al core ;

Si deve notare che non sono state incluse la tensione necessaria per il funzionamento del core del DSP, 1,4V, la quale infatti è ottenuta sulla scheda stessa tramite un regolatore buck LTM8020 realizzato

mediante componenti discreti e comandata dal processore stesso o esternamente dall' OBC, quindi non richiede una tensione diretta esterna.

Allo stesso modo, essendo l' alimentazione a 2,5 V solo per la memoria ROM, sulla quale rimangono salvati particolari dati come quelli utili al *wake up* da ibernazione, quindi un ridotto assorbimento di corrente, di circa 2 mA, è facilmente ottenibile dalla linea a 3,3V. Allora, tale tensione è stata appunto ottenuta mediante un regolatore di tensione low voltage drop out (LDO) LT1761 montato direttamente sulla scheda.

Le tensioni quindi direttamente fornite dalla scheda Power Management sono quella da 12V, da 3,3V e 5V che, insieme ai segnali di massa, compongono l'interfaccia di alimentazione della scheda del Sun Horizon Sensor.

Inoltre saranno presenti segnali di massa di due tipi:

- digitale, per tutti i componenti digitali utilizzati all'interno del sistema
- analogica, per il circuito di condizionamento del sensore di temperatura, in quanto è quella utilizzata come riferimento dalle schede che ne andranno a leggere il valore.

Infine, come verrà spiegato in seguito, sono previste tre linee distinte di alimentazioni provenienti dal Module_OBC_A e una dal Module_OBC_B: esse sono alimentazioni gestite completamente dall' esterno del nostro PCB, in modo indipendente dal sistema oggetto di questa tesi e dal suo funzionamento. Per questo motivo non sono considerate delle componenti dell'interfaccia di alimentazione della scheda, ma ciascuna verrà inglobata nell'interfaccia. Inoltre, la 5V e 12 V si considerano giungano in maniera stabile e per questo non saranno controllate (in realtà avviene il controllo avviene più a valle,al Blackfin), mentre anche se non si discosta molto da quanto detto, la 3.3V per il fatto che vada ad alimentare i principali dispositivi su di essa necessita di un current sensor e voltage sensor.

In aggiunta, per ottemperare alle diverse casistiche che potrebbero avvenire durante il funzionamento, sono stati previsti due load switch che permettono di staccare le alimentazioni nel qual caso si vada incontro ad un corto o piuttosto a qualsiasi tipo di problematica che possa essere risolta attraverso un isolamento della scheda. Nello specifico, un interruttore è stato posto per l' alimentazione a 3,3 V che è fornita alla maggior parte dei dispositivi, l' altro per la 12 V per spegnere il regolatore buck del core. Essendo la 2,5 V ottenuta dalla stessa VAL = 3.3V, se agissimo sul primo switch, isoleremmo allo stesso modo anche la memoria ROM.

3.6 Sensori di corrente e tensione

Affinchè grandezze elettriche come tensione e corrente vadano correttamente ad alimentare il processore Blackfin, si ha l' estrema necessità di effettuare un controllo prima che avvengano malfunzionamenti o

addirittura guasti irreversibili. Il Blackfin, per come è stato progettato, purtroppo non ci aiuta a tal senso, cioè non possiamo monitorare i risultati ottenuti da tale misurazioni convertendo la grandezza analogica in digitale non avendo a disposizione un ADC. Volendo fare una breve chiosa, in fase di scelta, uno degli obiettivi indispensabili per il processore, è stato anche la disponibilità di un convertitore integrato , ma non si è riusciti a trovare una soluzione di compromesso in quanto da un lato il Blackfin che lo avesse non era poi a disposizione per l' acquisto o non aveva un controller per memorie esterne, dall' altro il DSP che ne era privo racchiudeva poi tutte le altre caratteristiche di estrema necessità per il progetto. Naturalmente, si è optato per la seconda scelta prevedendo momentaneamente un invio dei segnali analogici all' OBC e in un secondo momento l' inserimento di un ADC sulla scheda che si interfacci con la PPI per l' acquisizione diretta da parte dello stesso Blackfin.

I tipi di sensore di tensione utilizzati sono tutti e tre dello stesso tipo e prevedono:

- una tensione d' ingresso VIN minore di 5V per le tre tensioni 1,4V, 2,5V e 3,3V;
- una corrente d' ingresso massima di 400mA, ampiamente entro le correnti in gioco; si pensi che solo la massima per il processore , la IDD di 1,4V è di 108mA.

In questi casi non è necessario effettuare alcuna operazione sui segnali, poiché la tensione è maggiore della dinamica dell'A/D, quindi si è optato per un semplice partitore più un filtro anti aliasing. La scelta dei resistori è frutto di un compromesso tra necessità di basso consumo e periodo di campionamento.

In generale, l'idea su cui si basa una misura di corrente è amplificare la differenza di potenziale che si crea ai capi di un apposito resistore, detto di sense, attraverso il quale la corrente di interesse viene fatta fluire.

Il valore da utilizzare per la Rsense è anche qui il frutto di un compromesso: dovrà essere sufficientemente elevato da garantire un segnale con una discreta intensità, ma anche abbastanza bassa per non dissipare troppa potenza e portare una caduta di tensione elevata ai suoi capi.

Nella realizzazione del sensore di sole sono stati usati di due tipi per far fronte alle diverse alimentazioni: quello di tipo high-side in grado di sopportare una tensione di modo comune ai pin d'ingresso superiore alla tensione di alimentazione (il CMRR elevato è garantito dal componente INA138) e una corrente d' ingresso massima di 152mA. Ciò è possibile grazie alla struttura interna, visibile nella figura sotto:

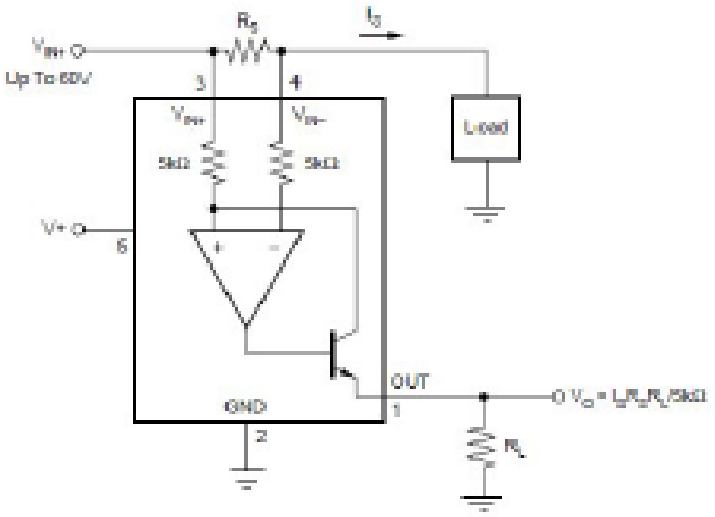


Figura 3.40: Current Sensor di tipo High Side

Ed ancora due di tipo low side, con l' inconveniente di avere a disposizione sulla scheda un' ulteriore tensione a 5V, ma estremamente necessaria per alimentare il MAX4091 e controllare dunque le alimentazioni a 3,3 V e 2,5V. La caratteristica essenziale di questo sensore è che la corrente d' ingresso può raggiungere massimo 0,4A e quindi monitorare correnti più alte come quelle della SDRAM che può raggiungere i 250 mA per la IDD di auto-refresh per esempio. Però, la tipologia low-side è però affetta da un grave difetto: interrompe di fatto, la massa del circuito, rendendolo non equipotenziale sull'intero PCB.

3.7 Consumi Energetici

I calcoli dei consumi di corrente, potenza ed energia sono di vitale importanza per un sistema alimentato a batterie, che deve funzionare in ambiente spaziale (quindi con una ridotta capacità di dissipazione del calore) e che deve essere alimentata da un alimentatore specifico. Infatti il consumo massimo e minimo di corrente servono al progettista degli alimentatori (in questo caso la scheda Power Management) per dimensionare i componenti degli stessi; la dissipazione di potenza serve per il calcolo dell'andamento del calore all'interno del satellite e il consumo energetico nelle diverse fasi di funzionamento serve a chi gestisce le batterie per sapere se l'energia immagazzinata in esse è sufficiente per più cicli di funzionamento.

Per tale scopo si sono suddivisi i calcoli in base ai diversi cicli di funzionamento, descritti poi meglio nella sezione dedicata al progetto software. Quindi questi dati saranno maggiormente comprensibili dopo aver chiarito come vengono organizzati i dati, soprattutto la memoria esterna, la cui interfaccia col Blackfin comporta un importante consumo di potenza, anche se però difficile da stimare con certezza in quanto molto diverso tra loro. Per questo per i calcoli sull'energia consumata, dove è presente anche il fattore tempo, si sono fatte delle stime del tempo di rispettivo utilizzo della memoria.

I calcoli teorici rimangono in realtà sono solo approssimativi se basati su quanto messo a disposizione dai datasheets; naturalmente per una maggiore precisione si rimanda a misure dettagliate effettuate in fase di test dopo la realizzazione fisica del PCB.

Capitolo 4

Realizzazione Hardware

La realizzazione del PCB della scheda è stata la fase che ha concluso il progetto, ma nello stesso tempo è stata anche la più lunga per diversi aspetti che saranno affrontati all' interno di questo stesso capitolo.

Gli strumenti messi a disposizione per la realizzazione del PCB sono raggruppati in un pacchetto software CAE (Computer Aided Design) di Mentor Graphics, disponibile presso il dipartimento di elettronica del Politecnico di Torino.

Per arrivare a realizzare uno stampato mediante tale ambiente di sviluppo, si devono seguire diversi step adottando diversi sottoprogrammi, ciascuno dei quali alquanto complessi e non sempre user friendly.
Infatti il processo si suddivide in tre grandi operazioni:

1. creazione di una libreria dei componenti utilizzati sulla scheda mediante Library Manager di Mentor Graphics
2. disegno dello schema elettrico, mediante il programma DxDesigner di Mentor Graphics
3. posizionamento dei componenti sullo stampato, routing delle piste e verifica dei constraints mediante Expedition PCB di Mentor Graphics.

4.1 Libreria

Il primo passo prevede la creazione della partizione con Partition Editor, all' interno della Central Library (precisamente Aramis_Mentor_Lib.lmc), libreria condivisa dal dipartimento e sincronizzata alla disponibilità di magazzino; tale libreria sarà composta da:

- la cella logica rappresentante il componente discreto o integrato, che fornisce informazioni sul tipo (alimentazione, massa, analogico, digitale, ingresso, uscita, bidirezionale), il nome e il numero dei pin del componente. Queste celle, chiamate SYMBOLS, sono impiegate per disegnare lo schema elettrico e le loro informazioni sono utilizzate per eseguire controlli sulla logica delle interconnessioni. Tali celle sono creabili e modificabili mediante Symbol Editor;
- la cella fisica, cioè le dimensioni del package, di ogni componente integrato o discreto. Tali celle, chiamate CELL, sono quelle poi impiegate nel posizionamento dei footprint

dei componenti sul PCB. Solitamente i componenti integrati utilizzano package standard e quindi tali celle si trovano in librerie standard già esistenti. Comunque mediante il Cell Editor, si può creare o modificare le dimensioni dei footprint;

- il componente completo, sia della parte logica che di quella fisica, chiamato PARTS. Questo passo è quello più importante nelle operazioni di libreria, in quanto esegue il collegamento tra la cella logica e la cella fisica, collegando i pin disegnati sulla cella per lo schema elettrico con quelli che poi andranno sul PCB e ordinandoli per tipo e numero. Il programma Part Editor permette di controllare il risultato di tale collegamento, perché se si sbaglia in questo step, il PCB non riporterà i collegamenti dello schema elettrico e tutto sarà sbagliato e molto difficile da vedere, se non con un'analisi del PCB, molto arduo in schede leggermente complesse. La parte è poi la cella a cui si fa riferimento nello schema elettrico, dove viene utilizzata la sua parte logica, e nel PCB dove viene impiegata la sua parte fisica.

In realtà, non tutte le parti sono state messe a disposizione perché condivise in quanto alcuni componenti non risultavano essere presenti e si è dovuto fare un lungo lavoro di progettazione partendo dai padstacks (a titolo d' esempio può essere citato il power pad del processore, molto particolare e che necessitava si essere connesso robustamente a massa) e finendo alla creazione della cella secondo il giusto package. Componenti come sensore d' immagine e processore sono stati creati da zero partendo dal datasheet e ed esaminando passo dopo passo sia la parte logica (numero di segnali disponibili sui pins) che quella fisica (package TSOP II e PLCC)

Alla parte viene associato un numero (part number), un nome (part name) e un'etichetta univoca (part label). Nella libreria della scheda si è inserito come part number per ogni componente il suo codice d'ordine del distributore da cui è stato acquistato, preceduto dalle sue iniziali, come DK per DigiKey, RS per RS Components, FA per Farnell e soprattutto AR per Arrow Europe Components (un distributore poco utilizzato rispetto agli altri, ma suggerito da Aptina e Analog Devices per l' acquisto dei principali componenti della scheda). Un part number tipico è AR_MTV03434C12STM. Se invece il componente è un stato ottenuto come sample, si è scritto la sigla del componente preceduta da SM. Invece come part name è stato messa la sigla del componente, se esso integrato, altrimenti il valore del componente discreto preceduto da R nel caso di una resistenza, C se condensatore e L se induttore. Infine come etichetta univoca è stata messa, nel caso di componenti integrati la loro sigla per intero, mentre nel caso di componenti discreti è stata utilizzata una stringa che raccoglie tutte le informazioni più rilevanti per ogni componente.

Purtroppo sia per quanto concerne la parte logica, sia per quella fisica non tutti i componenti erano a disposizione

Infatti a seconda della tipologia del componente le stringhe utilizzate per i principali componenti sono:

PART NUMBER	PART NAME	PART LABEL
AR_SP-BF518BSWZ4F16_	ADSP_BF_518F16	BLACKFIN_ADSP_176_LQFP
AR_MTV03434C12STM_	MTV03434C12STM_	IMAGE SENSOR_48CLCC
DK__LTM8020IV#PBF-ND	LTM8020_	Switching_Regulator__1V4_200mA
DK_557-1224-6- ND	MT48LC4M16A2P_TG	Micron_SDRAM_64Mb_54TSOP_2
SER3627DKR-ND	XTAL_TSX_3225	XTAL_26MHz

Figura 4.1. Tabella dei componenti del progetto

A titolo di esempio sono raffigurate sotto i due principali componenti progettati:

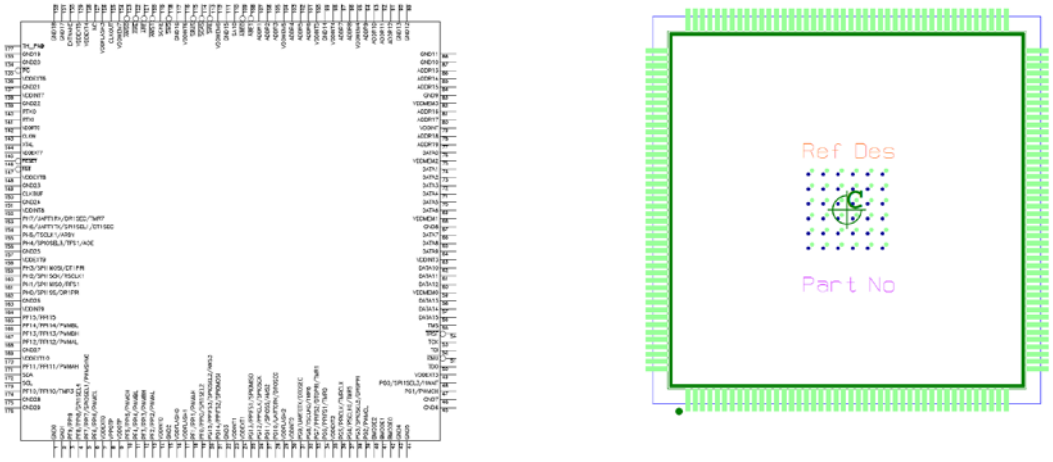


Figura 4.2. Componente ADSP-BF518F16

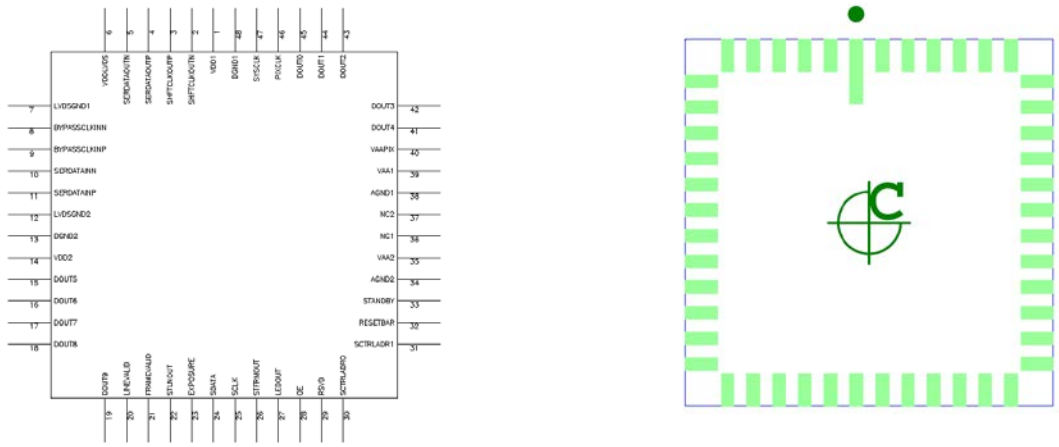


Figura 4.3. Componente MT9V034

Inoltre, per quanto riguarda gli altri componenti “di uso comune”, sono messi a disposizione dalla libreria centrale ad ogni studente ed in particolare, ad ogni parte ci sarà associata la lettera da utilizzare per il Reference Designator dei componenti; nella scheda sono state utilizzate:

- R per i resistori
- C per i condensatori
- D per il diodo (l’ unico presente è per il regolatore buck))
- Q per i transistor discreti (presente nel load switch))
- X per i quarzi
- U per i componenti integrati

Infine la libreria deve essere anche completa di tutti i fori utilizzati nella scheda, sia quelli strutturali che quelli funzionali come via e hole per connettori non superficiali. Tutte le informazioni sulle dimensioni e forme dei buchi impiegati sono creabili e modificabili mediante il PadStack Editor.
A questo punto si hanno tutti i componenti disponibili per la creazione dello schema elettrico e del PCB.

4.2 Schema elettrico

Per disegnare lo schema elettrico, si è utilizzato il software Dx Designer, che si deve riferire alla libreria della scheda. A questo punto si utilizzano le celle logiche disegnate prima nella libreria e si collegano tra di loro mediante il comando Wire oppure per collegamenti di segnali multipli si può utilizzare il comando Bus.

Complessivamente sono stati creati due reusable blocks che rispecchiano quanto detto più volte, il concetto di modularità o meglio nella fattispecie di riutilizzo. Per questo all’ interno del blocco del processore, sono stati utilizzati dei particolari connettori che permettono dei collegamenti tra i diversi moduli: MODULE_A, MODULE_B, MODULE_C, MODULE_D, MODULE_OBC_A, MODULE_OBC_B , MODULE_OBC_C e MODULO_JTAG. Questi simboli di “inter-page”, inoltre, indicano se il segnale è uscente o entrante alla net a cui sono collegati, per un controllo della logica delle interconnessione, per facilitare di trovare eventuali errori presenti sullo schema, nel caso per esempio di due net collegate a due simboli di output o di input, invece di essere uno di input e uno di output. Quindi i due Reusable Blocks creati sono:

- 1B4231_Image_Processor_BF518F16 (contiene il processore con la SDRAM, le capacità di bypass e tutta la parte hardware necessaria al controllo e all’ abilitazione);

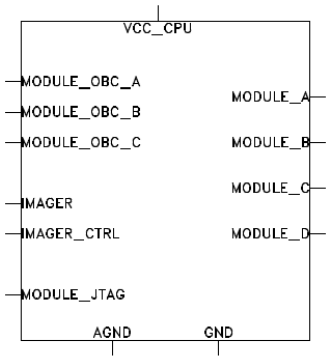


Figura 4.4. Blocco Image_Processor

- 1B511A_Mono_WVGA_CMOS_Imager (contiene principalmente l' image sensor dell' Aptina e le capacità di bypass);

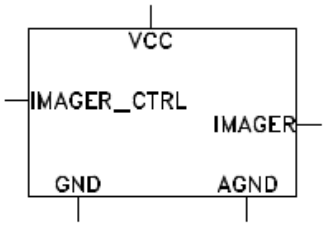


Figura 4.5. Blocco Imager

All'interno di una stessa pagina si sono anche utilizzati simboli di collegamenti chiamati 'intra-page', ossia che collegano due punti all'interno della stessa pagina e collegati a tali simboli, nominando allo stesso modo la net collegata ed esso nei due o più punti collegati insieme.

Lo schema elettrico della scheda è su un'unica pagina, ma è mostrato nell' appendice A dove sono raffigurati anche tutti gli altri reusable blocks come i current sensors, i voltage sensors, i load switches i regolatori. Essendo i segnali passanti all'interno del connettore, ad ogni pin dei due simboli vi sono simboli di connessione 'intra-page' con il nome della net corrispondente.

Nella stessa pagina sono contenuti tutti i condensatori di bypass per tutti i componenti integrati digitali presenti sulla scheda:

- Processore Blackfin BF-518F16: la linea di alimentazione da 3,3V prevede 22 pin d'ingresso sparsi su tutto il componente e quindi ci sono 22 condensatori di bypass, di diverso tipo per avere una maggiore copertura in frequenza. Sono stati impiegati 4 condensatori da 10 μF, 7 condensatori da 10 nF e i restanti 8 da 100 nF.

- Processore Blackfin BF-518F16: la linea di alimentazione per il core prevede 12 pin d'ingresso sparsi su tutto il componente e quindi ci sono 12 condensatori di bypass con valore che varia da 100nF, 10nF o 1nF. Il datasheet suggerisce che essi siano posti vicino all' exposed pad sul lato bottom, opposto a quello dove è saldato il processore.
- Processore Blackfin BF-518F16: la linea di alimentazione per la ROM prevede solo 2 pin ed in questo caso saranno usate due sole capacità da 100nF.
- Sensore d' immagine MT9V034: la linea di alimentazione per il sensore è fornita mediante 6 pin e quindi sono collegati 6 condensatori da 10 nF .
- Memoria SDRAM: possiede 7 pin per l'ingresso della tensione di alimentazione quindi sono stati inseriti 7 condensatori da 10 nF

Tali condensatori di bypass sono collegati alle relative tensioni tramite connettori intra page e a Gnd che sarà il nostro unico segnale globale.

Dopo aver completato lo schematico dei 2 blocchi si sono eseguiti i tre seguenti passi:

- controllo attraverso il comando Dx Designer Diagnostics, che effettua il test attraverso 17 passaggi per verificare che siano state correttamente configurate connettività, nome dei componenti, connessione dei bus, presenza di blocchi vuoti e così via..
- impacchettamento attraverso il packager che rileva tutte le informazioni relative ai simboli e alle reti del progetto dal file generato durante la compilazione, localizza le informazioni sulle corrispondenti celle fisiche nel PDB (parts database) e le assegna ai simboli riportati nello schematico. I Reference Designator e i numeri dei pins vengono assegnati al simbolo secondo quanto specificato nella corrispondente parte all'interno del PDB.
- Una volta che il passo precedente abbia dato esito positivo senza errori e warnings, possiamo procedere ad etichettare il reusable blocco come verificato. In questa condizione non può essere più modificato ed è pronto per essere utilizzato come fosse un singolo componente con i suoi propri pin che saranno gli stessi che abbiamo portato fuori attraverso i bus o connettori inter page. Il blocco per come è stato impacchettato, sicuramente non sarebbe funzionale ancora in quanto il simbolo necessita di essere editato. Quindi, nel qual caso si volessero apportare delle correzioni allo schematico, si ricordi che bisogna prima effettuare un *unverify*.

Finita la progettazione dei due blocchi, sono stati connessi attraverso i bus IMAGER_CTRL che raggruppa tutte le temporizzazioni del sensore come HSYNC, VSYNC, EXPOSURE, STANDBY, SCL e SDA, e IMAGER che è l' insieme dei 10 bits in uscita dal sensore e in ingresso alla PPI.

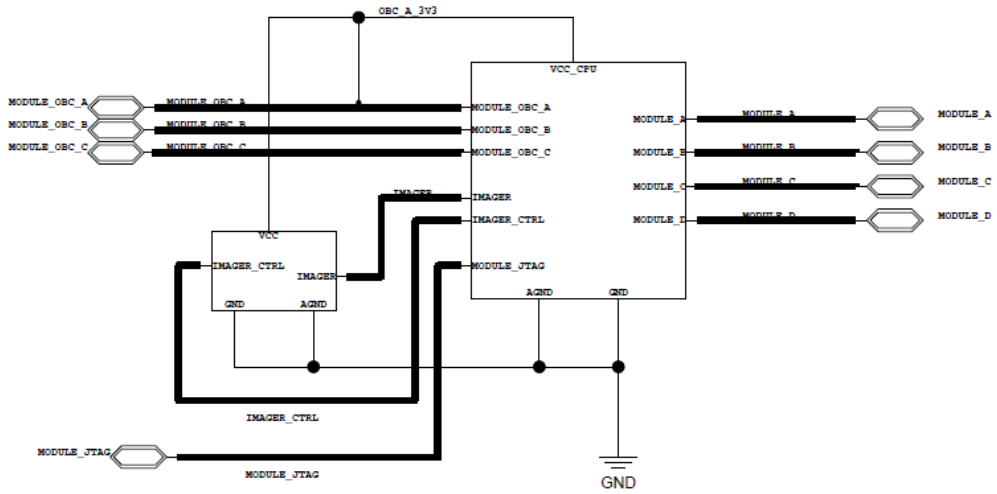


Figura 4.6. Schematico Sun_Horizon_Sensor

Allo stesso modo come è stato prima, si è effettuata prima la diagnostica e poi il packaging del progetto Sun_Horizon_Sensor, che appunto si avvale del riutilizzo dei due blocchi. Ora si è pronti per la realizzazione fisica della scheda.

4.3 PCB

La realizzazione del PCB è stata eseguita a livello unicamente dimostrativo mediante il software Expedition PCB. Dopo averlo collegato alla libreria e al suo schema elettrico, si può iniziare seguendo i seguenti passi:

1. disegnare il contorno della scheda
2. posizionare i componenti
3. eseguire il routine delle piste
4. verifica di tutte le interconnessioni e dei parametri di configurazione

4.3.1Vincoli strutturali

Il circuito stampato della scheda deve rispettare dei vincoli di dimensioni a causa dello spazio a disposizione dentro al satellite e di come sono state posizionate le schede al suo interno.

4.3.2 Posizionamento dei componenti

Lo stampato si sviluppa su 4 strati, due di segnale, lo strato 1 sul lato top della scheda è un piano di massa su cui sono anche piazzati tutti i componenti e parte dei condensatori di bypass, lo strato 2 è un piano negativo anch' esso di massa, lo strato 3 è un piano dove passa il segnale come pure il 4, ma con la differenza che sul lato bottom sono state piazzate le capacità di bypass della 1.4V, vicino all' exposed pad.

Inizialmente sono stati posizionati i principali componenti come processore, sensore e Sdram, poi successivamente resistenze e capacità che facevano riferimento ai segnali di controllo e poi tutte le capacità di bypass. Si può notare che non è stato posizionato nessun tipo di connettore perché ribadisco che il PCB è stato creato unicamente a livello dimostrativo per mettere insieme a grandi linee il lavoro prodotto. Ciò non toglie che si è effettuato un posizionamento accurato di ogni singolo componente rispettando ogni tipo di vincolo e problematica che si sarebbe potuto avere in fase di routing
Il processore, essendo quello che si interfaccia direttamente anche con gli altri moduli attraverso i suoi 177 pin, è risultato il più delicato da inserire: infatti, lo si è provato ad inserire prima sul lato bottom per avere anche più spazio per le tracce e poi, dopo i giusti accorgimenti, è stato messo sullo stesso layer degli altri componenti..Allo stesso modo, numerosi tentativi sono stati necessari per posizionare la memoria in modo da favorire un routing dei segnali ragionevole.

4.3.3 Routing

Lo step più importante è quello del routing. Infatti è fondamentale impostare correttamente tutte i parametri di route, al fine di non avere piste troppo vicine o troppo sottili o via di dimensioni sbagliate, non supportate poi dal costruttore del PCB.

La larghezza tipica utilizzata per le piste della scheda Payload è di 0,18 mm, ma nei casi limite la larghezza minima utilizzata è di 0,14mm. Quest'ultima dimensione è anche quella che è stata fissata per le distanze minime tra i via e i pad dei componenti, tra due pad, tra i pad e le piste, tra le piste e i via e tra due via chiamata *clearances* e modificabili in Constraints Editor).

Dapprima è stato avviato l'Auto Route, ossia ci ha pensato il software a tracciare ogni singola traccia possibile per connettere i vari pads e a disporre il via in maniera ottimale secondo i constraints editati. Successivamente, sono state tracciate manualmente le pochissime tracce rimaste e laddove si avesse necessità di una migliore connessione, sono state anche modificate.

==========

DESIGN STATUS

==========

Board Size Extents ..... 120 X 100 (mm)

Route Border Extents ..... 119 X 99,2 (mm)

Actual Board Area ..... 12.000 (mm)

Actual Route Area ..... 11.804,8 (mm)

Placement Areas: Name Available Required Required/Available

Entire Board 24.000 Sq. (mm) 1.512,71 Sq. (mm) 6.30 %

Pins 568

Pins per Route Area ..... 0,05 Pins/Sq. (mm)

Layers 4

Layer 1 is a signal layer

Trace Widths 0,14, 0,18, 0,25, 0,25

Layer 2 is a Negative Plane Layer with nets

GND

Trace Widths 0,14

Layer 3 is a signal layer

Trace Widths 0,14

Layer 4 is a signal layer

Trace Widths 0,14, 0,25

Nets 128

Connections 410

Open Connections 0

Differential Pairs 0

Percent Routed ..... 100.00 %

Netline Length ..... 0 (mm)

Netline Manhattan Length ..... 0 (mm)

Total Trace Length ..... 4.366,79 (mm)

Trace Widths Used (mm) 0,14, 0,18, 0,25, 0,25

Vias 279

Via Span Name Quantity

1-4 0,45mmVIA 13

026VIA 266

Teardrops..... 0

Pad Teardrops..... 0

Trace Teardrops..... 0

Custom Teardrops..... 0

Breakouts..... 0

Guide Pins 0

Parts Placed 109
Parts Mounted on Top 97
SMD 96
Through 1
Test Points 0
Mechanical 0
Parts Mounted on Bottom ... 12
SMD 12
Through 0
Test Points 0
Mechanical 0
Embedded Components 0
Capacitors 0
Resistors 0
Edge Connector Parts 0

Parts not Placed 0

Nested Cells 0

Jumpers 0

Through Holes 315
Holes per Board Area ..... 0,03 Holes/Sq. (mm)
Mounting Holes 0

Figura 4.7. Riepilogo misure del PCB

Per verificare se tutte le piste tracciate durante il routine sono corrette e rispettano tutte le distanze minime inserite nei parametri di configurazione, si esegue il controllo DRC (Design Rule Check). I controlli che esegue sono due:

- controlli di *Prossimità*: verifica che siano state rispettate tutte le clearances specificate dapprima nei parametri di routine
- controlli di *Connettività*: verifica che le tracce ed i piani connettano effettivamente i pin facenti parte della stessa rete definita nello schema elettrico e che non vi siano hanger, cioè tracce flottanti.

Infine gli ultimi tre step da eseguire per avere il PCB pronto per essere spedito al produttore sono:

- generazione del file per le macchine perforatrici che effettueranno i fori (Drill) sul PCB, chiamato NC Drill

- generazione del Silkscreen, mediante il Silkscreen Generator, ossia del lato superiore dello stampato ove vi saranno tutte le scritte e i Reference designator dei componenti. Infatti prima di fare ciò, si deve controllare che essi non siano sopra qualche via sullo stampato che li renderebbero illeggibili. Se avviene ciò, si passa in draw mode e si spostano manualmente
- generazione del file Gerber utilizzato dalle macchine della ditta produttrice di PCB per tracciare fisicamente il PCB, mediante il Gerber Output del Gerber Tool, il quale permette anche di eseguire una verifica finale del PCB.

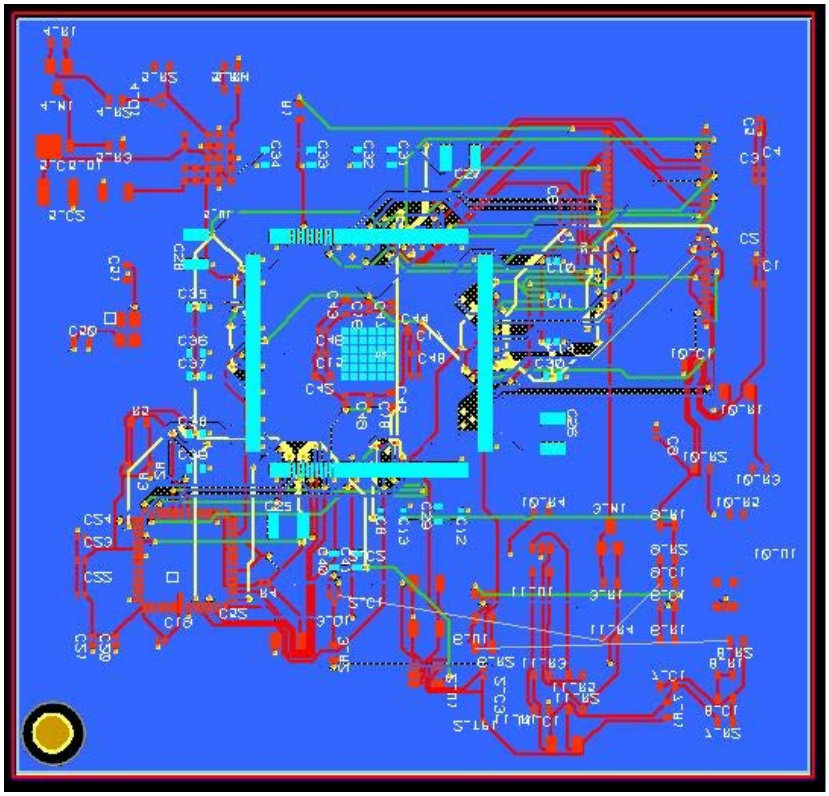


Figura 4.8. Sun Sensor su Expedition PCB

Capitolo 5

Progetto Software

5.1. Breve introduzione al linguaggio UML

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

Anche il modulo 1B23, argomento di questa tesi, come tutti gli altri blocchi di AraMiS, è stato ideato, definito e strutturato tramite l'UML. Pertanto, prima di proseguire nella trattazione, è opportuno fornire una breve descrizione di tale linguaggio, delle sue componenti fondamentali e della simbolistica utilizzata.

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

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

Il tool utilizzato per la creazione dei diagrammi e del progetto in UML è *Visual Paradigm for UML*, versione 11.2, che permette anche di ottenere automaticamente i file (*source* .c++ e *header* .h) che andranno a costituire il software del microcontrollore, generando il codice direttamente a partire dai diagrammi delle classi e dal corpo delle funzioni, contenuto negli oggetti dei diagrammi stessi . Per realizzare e gestire il progetto del software, eseguirne il debug e programmare il microprocessore è stato invece utilizzato il tool *CrossCore Embedded Studio* messo a disposizione dalla Analog Devices.

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

sequenza, per limitazioni di tempo non è stato realizzato, ma costituisce insieme agli altri due i principali diagrammi per rappresentare un sistema.

5.1.1. Diagramma dei casi d'uso

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

- L'**Attore** è ogni qualsivoglia entità, che sia essa un utente umano, un altro sistema o l'ambiente esterno, che interagisce con il sistema in oggetto, richiedendo ad esso la realizzazione di uno o più casi d'uso; esso viene rappresentato con un omino stilizzato (vedere figura 4.1) e ve ne possono essere, eventualmente, più di uno in un diagramma;

Fig. 5.1 Attore

- I **Casi d'Uso** sono i compiti che l'attore richiede al sistema, quindi gli obiettivi del progetto in esame; vengono rappresentati graficamente tramite delle ellissi all'interno delle quali è scritta in breve la funzione realizzata dal sistema; possono essere direttamente chiamati dall'attore oppure correlarsi ad altri casi d'uso (vedere figura 4.2);

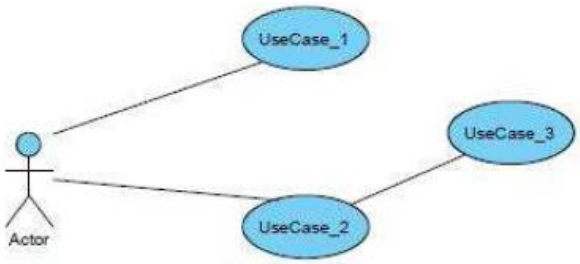


Fig. 5.2 Casi d' uso

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

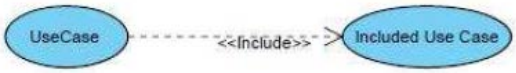


Fig. 5.3 Relazione *include*

- **Relazioni tra i casi d'uso**, citiamo ad esempio:

  a) *Generalizzazione*: analoga a quella descritta sopra per gli attori;

b) *Inclusione*: indicata con una linea tratteggiata recante dicitura <<include>> (figura 4.3), indica che il caso d'uso base include, fra le azioni che può far compiere al sistema, anche le azioni del caso d'uso incluso;

c) *Estensione*: graficamente simile all'inclusione ma con dicitura <<extend>>, sta a significare che il caso d'uso estensione (vedere figura 4.4) rappresenta una funzionalità opzionale del caso d'uso base;

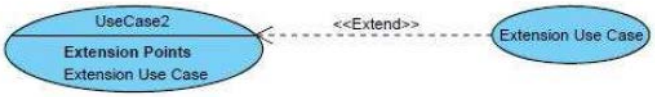


Fig. 5.4 Relazione *extend*

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

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

5.1.2. Diagramma delle classi

Il diagramma delle classi è composto essenzialmente dagli **oggetti** e dalle varie associazioni fra di essi ed è volto a caratterizzare minuziosamente il sistema in esame, descrivendolo in tutte le sue componenti, hardware, meccaniche, software o miste hardware/software, ma anche di altro tipo, a seconda delle esigenze di progetto.

Un **oggetto** è una qualsiasi entità, facente parte del sistema, che interagisce con l'esterno, con oggetti di altri o del proprio sistema.

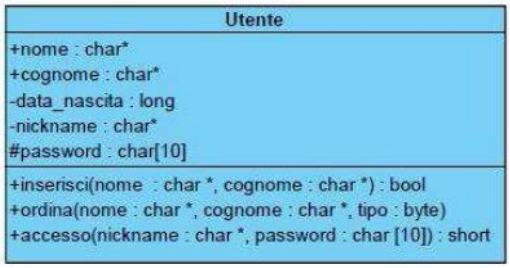


Fig. 5.5 Esempio di classe

Nel nostro caso un oggetto può identificarsi con uno specifico componente hardware del sistema, ma anche con parti meccaniche del medesimo, o con sotto-blocchi logici indispensabili per il suo coordinamento e la sua funzionalità: in tal caso si parla di oggetti software.

Per classe si intende, invece, l'astrazione, ovvero la generalizzazione, di un oggetto, che ne costituisce l'istanza specifica. Caratteristiche fondamentali di un oggetto (e della sua classe) sono i suoi attributi ed i suoi metodi (che nella classe vengono denominati operazioni):

- Un attributo è una proprietà dell'oggetto, sia essa logica, fisica o quant'altro: ad esempio, se l'oggetto in questione è un sensore, suoi attributi possono essere la sensibilità e il consumo di potenza; invece, se l'oggetto fosse un polinomio, gli attributi potrebbero essere i coefficienti delle diverse potenze; in termini di software, invece, gli attributi equivalgono a variabili, strutture e *define* del linguaggio C. Ogni attributo può essere caratterizzato da un tipo (ad es. *int* per i numeri interi), da un valore iniziale, dall'accessibilità (che definisce se il suo valore può essere modificato e, se sì, da quali oggetti), dalla visibilità, ed altre proprietà più specifiche. Caso usuale, nel progetto in esame, è l'utilizzo di un attributo per associare all'oggetto in questione un altro oggetto (o classe), che va proprio a rappresentare il tipo di tale attributo, in modo che il primo possa servirsi anche degli attributi del secondo, o i suoi metodi possano chiamare quelli dell'altro. Gli attributi compaiono nel primo riquadro sotto quello del nome dell'oggetto.
- Il metodo indica quali operazioni può compiere l'oggetto ed equivale alle funzioni C, per un oggetto software, oppure ai fili di segnale, per oggetti hardware. Esso può venire associato ad un valore di ritorno, di cui si indica il tipo, che esso restituisce al chiamante, ed ai parametri che esso riceve dal medesimo (indicati fra parentesi). I metodi compaiono nel riquadro in basso, sotto quello contenente gli attributi.

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

5.2 Aramis UML

La “standardizzazione” dell'architettura AraMiS prevede l'utilizzo del software UML-compatibile Visual Paradigm 11.2. Il progetto dell'architettura utilizza in maniera intensiva il linguaggio Unified Modeling Language UML ed il software UML-compatibile Visual Paradigm 11.2 come strumento.

Inoltre attraverso il tool UML-Visual Paradigm è possibile tradurre in codice per il processore i diagrammi creati e viceversa. Un altro vantaggio è la possibilità di creare un gruppo di lavoro, un team, con cui progettare e condividere tramite server il materiale prodotto.

Attraverso Visual Paradigm sono dunque utilizzate numerose strutture per descrivere in maniera approfondita il progetto AraMiS ed in particolare sono presenti:

- Diagrammi per le specifiche funzionali;
- Diagrammi per le specifiche prestazionali;
- Digrammi sequenziali per l'analisi dei sistemi e dei relativi sotto-sistemi;
- Diagramma per le specifiche architetturali con allegata documentazione HW/SW;

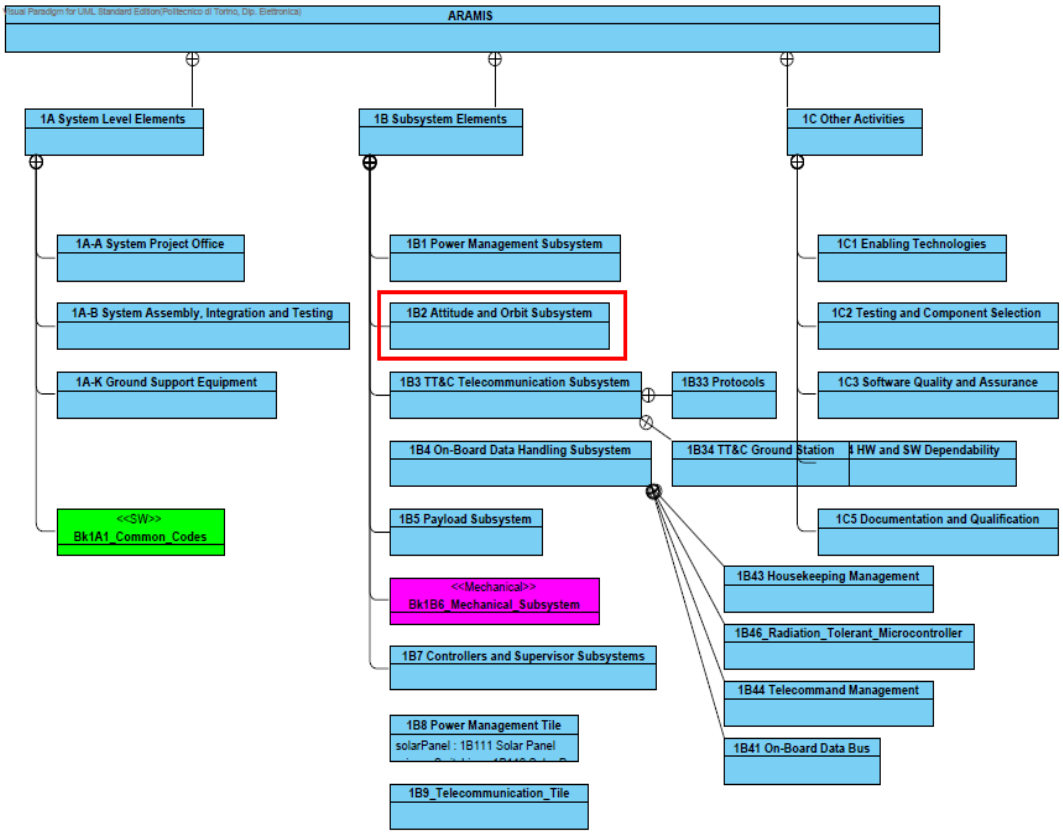


Fig. 5.6 Diagramma delle classi Aramis (Visual Paradigm)

Nell’ immagine sopra, in rosso, è evidenziato il sottosistema oggetto di questa tesi, *1B2_Attitude_and_Orbit_Subsystem.* Per il progetto AraMiS, si è pensato di dotare ciascun modulo di Power Management di un controllo d’assetto attivo che tramite un solenoide e una ruota d’inerzia permetta al satellite di assumere qualsiasi posizione attorno i tre assi permettendo quindi di puntare le

fotocamere in qualunque direzione e di scegliere quali facce del satellite devono essere esposte alla radiazione solare. Questo risultato è possibile assemblando il satellite con più moduli di Power Management i quali eseguendo i comandi che giungono dal modulo Telecommunication portano l'intero satellite a compiere la rotazione desiderata. Le prestazioni del sistema di controllo d'assetto aumenteranno aumentando il numero di moduli di Power Management, aumenterà però anche la complessità del modulo Telecommunication che deve fornire i comandi di assetto interpretando il comando d'assetto giunto dalla stazione di terra. Utilizzando un controllo d'assetto attivo si può ottenere una stabilità del satellite molto più accurata rispetto all'uso dei magneti permanenti che portavano PiCPoT ad avere un ondeggiamento quando esso transitava sui poli magnetici terrestri.
Nello specifico, andremo ad analizzare solo una parte del sottosistema e cioè quella denominata 1B23_Other_Attitude_Sensor che, come si può vedere sotto, conterrà i sensori di sole, di stelle, ricevitore GPS e differenziale RF, mentre saranno esclusi giroscopio e il sottosistema di assetto e controllo magnetico che saranno dei sottosistemi a parte.

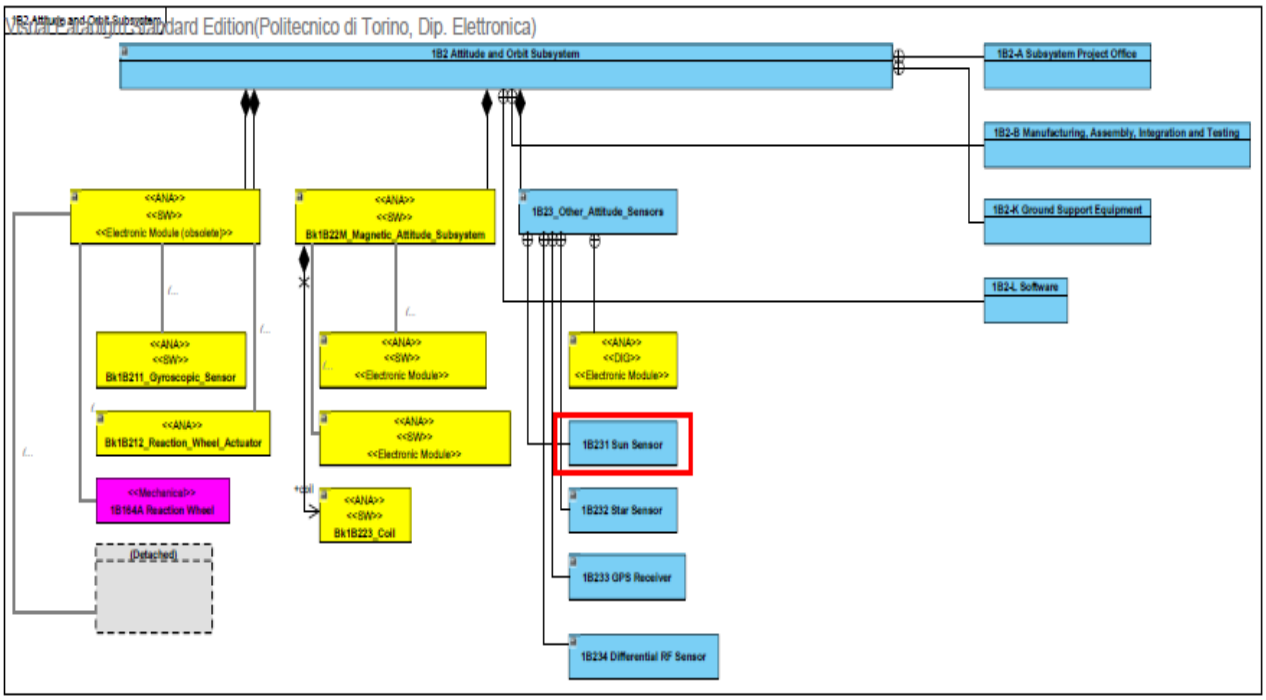


Fig. 5.7 Diagramma delle classi Sottosistema del controllo d' assetto (Visual Paradigm)

Infine, la classe da sviluppare completamente nel progetto, sarà la 1B231_Sun_Sensor che racchiude le caratteristiche meccaniche, hardware e software dell' intero sistema. Il sensore però, per come lo si è progettato, non può esimersi dall' avere un processore che elabori, instradi e regoli le temporizzazioni in quanto singolarmente rappresenta uno slave. In quest' ottica, la strategia seguita step-by-step è stata la seguente:

- creare la classe 1B4231_Imager_Processor_BF518F16 che a livello hardware può essere vista come uno Reusable Block;

- creare la classe 1B511A_Monochrome_WVGA_CMOS_Imager che a livello hardware può essere vista come uno Reusable Block;
- mettere le due classi insieme nel progetto 1B23_Other_Attitude_Sun_Sensors che include tra l' altro anche tutte le relative parti meccaniche e software;

A grandi linee è stato questo l' iter per la generazione poi del software tramite Visual Paradigm.

5.2.1 Interfaccia del processore

L'architettura di Aramis è modulare a livello elettrico , meccanico e software.

In questa architettura le principali funzionalità sono suddivise in un numero di moduli (sottosistemi) che sono integrati nello stesso PCB facendo del progetto, test ed integrazione un qualcosa di semplice. Sono specificati in tal senso:

- Interconnessioni tra i moduli;
- I dati scambiati dinamicamente;
- Potenza in architettura distribuita e autoconfigurazione (che è flessibile in quanto le grandezze elettriche, logiche e interfacce meccaniche tra i componenti sono specificate).

Con questo intento, ogni sottosistema è composto di:

- parte hardware;
- supporto SW corrispondente (irrobustito contro le radiazioni);
- di un appropriata interfaccia I/O.

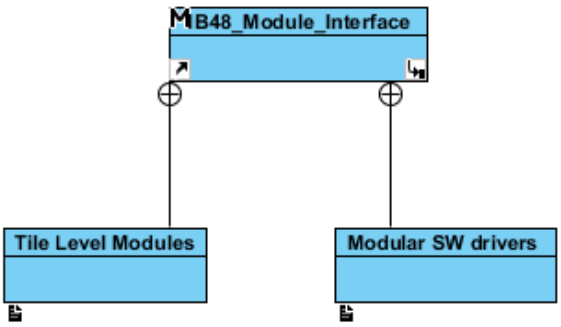


Fig. 5.8 Interfaccia Moduli – Visual Paradigm

L'idea di base che è dietro ad Aramis è il concetto dei *tiles* che è un blocco standard, usato per costruire un piccolo satellite secondo i requisiti specifici. Nell'architettura Aramis, le funzioni principali del bus sono suddivise in un numero adeguato di moduli identici rendendo il progetto più semplice. Le interconnessioni tra i moduli e gli scambi dinamici dei dati e della potenza in un'architettura distribuita garantisce che variazioni di determinate parti possano aversi sostituendo semplicemente i singoli componenti.

Questa strategia progettuale fornisce un alto grado di flessibilità nelle diverse configurazioni offrendo un grande numero di variazioni del sistema adattabili alle diverse missioni e dimensioni di satellite.
Ciascun tile include pannelli solari sul lato esterno, l'alimentazione di base e internamente l'elaborazione e l'instradamento dei dati.
C'è la possibilità di connettere fino a 16 connettori su un unico tile Aramis usando 2 MSP4305438A microcontrollori. Le connessioni, che sono elettricamente e meccanicamente modulari, partono dalla gestione di sistemi semplici attraverso un canale analogico ad arrivare a sistemi più grandi (possibili collegamenti fino a 28 canali analogici, 128 I/O digitali e CPU con seriali standard di comunicazione come UART, UART in modalità IrDA, SPI a 12C).
Ciascun segnale è dotato di un proprio segnale di abilitazione.

In questa visione d' insieme si inserisce anche il payload processor, ossia lo stesso processore visto finora, ma con funzioni che vanno oltre l' acquisizione. L' inserimento del Blackfin non è solo mirato ad essere un dispositivo master che pilota un sensore d' immagine, perché in un sistema che rispecchia continuamente la modularità come target non avrebbe avuto senso, ma deve potersi interfacciare con gli altri sottosistemi, comunicare con gli altri processori a bordo, inviare corrente e tensione per essere controllate ed essere abilitato o disabilitato attraverso determinati segnali.

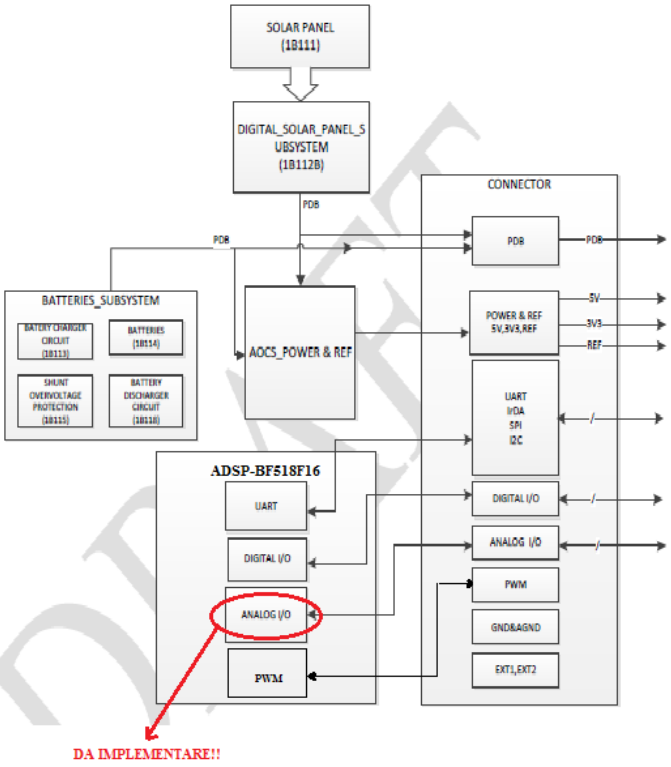


Fig.5.9 Singolo connettore per Blackfin interfacciato con altri sottosistemi

5.2.2 Pin Mapping

La scelta del sensore è stata allo stesso modo dettata dal principio di flessibilità, le periferiche del microprocessore dovevano essere tali da poter garantire la comunicazione (vedi SPI,TWI e UART), la possibilità di pilotare attuatori con determinati pattern PWM (vedi Timers e Modulo PWM a 3 fasi) e ricevere interrupt e abilitazioni su determinati pins. Purtroppo per problematiche legate alla disponibilità di alcuni processori della serie Blackfin, si è dovuto sacrificare l' aspetto legato alla conversione analogica in quanto si era evidenziata in un primo momento la disponibilità all' acquisto appunto di alcuni processori che rispettavano tutti i vincoli richiesti, ma successivamente non risultavano ancora in fase di produzione per futuri progetti. Si è optato dunque per la scelta di un ADSP che includesse al suo interno le periferiche di base, lasciando il canale analogico per ulteriori implementazioni.
Come è stato per l' MSP430, l' interfaccia con la quale si sarebbe dovuto collegare, doveva rispettare la tabella sotto:

Signal	PIN	Simbolo	I/O	Descrizione
Power Distribution Bus	19.20	PDB	O	Power Distribution Bus per alimentare tutti gli avionici Aramis con alimentazione da 12V-18V
+5V	16	5V	O	Alimentazione positiva a 5V ± 5
+3,3V	6	3V3	O	Alimentazione positiva a 3,3V ± 5
3V Reference	8	REF	O	Tensione di riferimento a 3V ± 1
Digital I/O	1-5, 7, 9-12	D0 – D9	I/O	Digital I/O: qualsiasi PIN di questi può essere usato per scopi digitali generali se non configurato già per altri scopi
Analog I	10, 12	A0 – A1	I	Analog Input
Uart/Irda Ricevitore	9,12	TX, RX	I/O	Una linea seriale standard Uart per TX/RX PINS dell'Uart dell'MSP430
SPI		SOMI, SIMO, CLK	I/O	SPI Signal e Clock
I2C		SCL, SDA	I/O	Serial Clock
Identification	4	ID	I/O	Da connettere con un filo alla memoria permette l'identificazione del modulo o recupero dati di calibrazione
Enable	2	EN		Ciascun modulo è abilitato dal segnale di attivazione
External Signals	13,15	EXT1 – EXT2	I/O	Connettori esterni GP per scopi di routine
Pulse Widtm Modulation	1,2	PWM, PWM2	O	Segnali PWM
Ground	17, 18	GND		Common Ground PIN
Analog Ground	14	AGND		Analog Ground

Tab. 5.10 Interfaccia segnali

In realtà, la tabella esprime fisicamente la distribuzione dei segnali su un connettore Mill-Max a 20 pin, che simbolicamente può essere visto così:

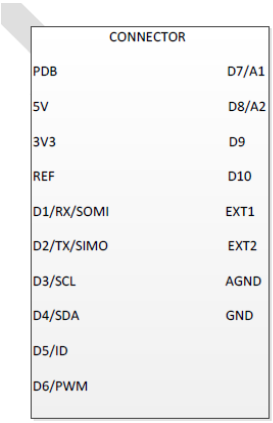


Fig. 5.11 Simbolo connettore

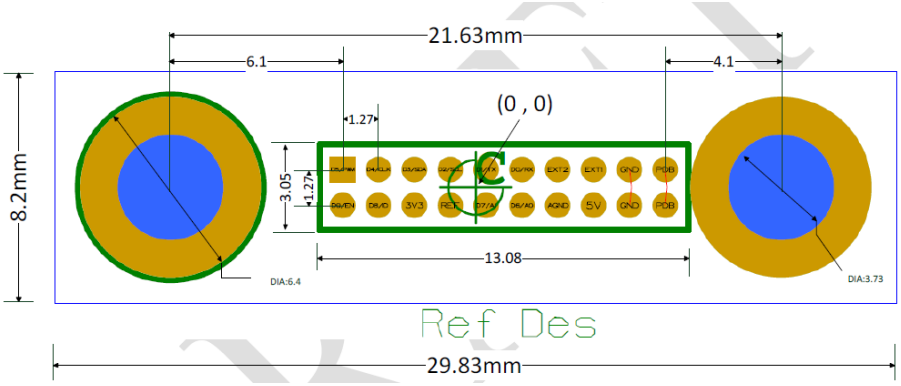


Fig. 5.12 Dimensioni del connettore

A livello del modulo, ciascun sottosistema è interfacciato con il tile processor usando una simmetrica sequenza di segnali. Ciascun modulo supporta funzioni multiple e diversi protocolli di comunicazione quando collegato al Tile Aramis.

PWM consiste soprattutto di segnali di timer.

Attraverso i canali analogici A0.A1 il microcontrollore può leggere le tensioni d'ingresso, corrente e temperatura dei sensori che saranno sensibilizzati dal microcontrollore.

5.2.2.1 Mappatura secondo i moduli standard

L'analisi dei segnali sui diversi pins è stata suddivisa dapprima in un lungo elenco delle funzioni multiplexate sui GP I/O e subito dopo in un continuo e contemporaneo confronto sui microprocessori della Texas Instruments pienamente integrati nel satellite rispetto all' inserimento del Blackfin. Le intenzioni erano quelle di raggruppare il più possibile tutti i segnali dell' acquisizione per rendere i moduli il più possibile indipendenti da tutto ciò. Purtroppo, malgrado il processore avesse a disposizione 176 pins, questo non è stato possibile e si è dovuto utilizzare lo stratagemma di mettere insieme tutti i

PPI0-9, le sincronizzazioni del frame e clocks nei moduli A e B. Dal fatto che sul pin 26 del processore abbiamo sia il PPICLK che le SPI0CLK, deduciamo che è impossibile avere contemporaneamente un' acquisizione e una comunicazione tramite protocollo SPI.

PIN CONNETTORE	**SEGNALI** PERIFERICHE DSP	**MODULO A**	**MODULO B**	**MODULO C**	**MODULO D**
11	D0/RX/SOMI	PG13/SPIOMISO PG10/UART0RX -> **25** +**28** (for *HSYNC*)	PF6 /PPI6/ TACLK1 -> **6** (D6)	PH1/SPI1MISO PH7/UART1RX/TMR7 → **161** + **153**	PG0 /SPI1_SEL3 → **48**
9	D1/TX/SIMO	PG14/SPIOMOSI PG9/UART0TX/TMR4 -> **21** +**31** (for *VSYNC*)	PF10 /PPI10/TMR3 → **174** (for *SYSCLOCK*)	PH3/SPI1MOSI + PH6/UART1TX → **159** + **154**	PH5/TSCLK1 -> **155** (for *STANDBY*)
7	D2/SCL/SOMI	*SCL* → **173**	*SCL* → **173**	*SCL* → **173**	*SCL* → **173**
5	D3/SDA/SIMO	*SDA* → **172**	*SDA* → **172**	*SDA* → **172**	*SDA* → **172**
3	D4/CLK/	PG12/ SPI0SCK /PPICLK /TMRCLK → **26** (for *PIXCLK*)	PG11/SPI0SS /AMS2/SPI1_SEL5 → **27**	PH2 /SPI1SCK → **160**	PH0/ SPI1SS → **162**
1	D5/ PWM	PG4/TMR5→ **37**	PG6 /TMR0 → **34** (for *EXPOSURE*)	PG7 /TMR1 → **33**	PG8 / TMR6 → **32**
12	D6/A0	PF0 /PPI0/TACLK6 -> **19** (D0)	PF2 /PPI2 -> **13** (D2)	PF4 /PPI4/ TACLK1 -> **11** (D4)	PG3/SPI0_SEL5 → **38**
10	D7/A1	PF1 /PPI1/TACLK7 -> **18** (D1)	PF3 /PPI3/ TACLK0 -> **12** (D3)	PG5 /TMRCLK -> **36**	PH4 /PPI7/ SPI0_SEL3 -> **156**
4	D8/ID/INT	PF8 /PPI8/SPI1_SEL4 → **4** (D8)	PF13 /PWM_BH -> **167**	PG2 /PWM_CH -> **39**	PF11 /PWM_AH -> **171**
2	D9/EN/PWM2/INT	PF9 /PPI9 /TMR2 -> **3**	PF14 /PWM_BH -> **166**	PG1 /PWM_CL -> **47**	PF12 /PWM_AL -> **168**

Tab. 5.13 Assegnazione pins ai moduli A,B.C.D

PG15/PPIFS3 -> **20** (for *FIELD* (a GND))

LEGENDA:

- SPI
- I2C
- UART
- **ADC (NON PRESENTE IL CONVERTITORE !!)**
- PWM
- PWM2 (GENERATORE A 3 FASI)
- INTERRUPT

Notazioni:

SPI = sul MODULO A (per clock e scambio dati) e sul MODULO C (per clock e scambio dati) (con possibilità di essere selezionato e selezionare (contemporaneamente) degli slaves su MODULO B/D).

Attenzione però : MODULO A E B NON POSSONO FUNZIONARE CONTEMPORANEAMENTE A IMAGER sul protocollo SPI. !! Sui moduli C e D SI', a patto che non si usino i pins evidenziati in giallo per acquisizione immagine dal sensore

I^2C = MODULO A,B,C,D;

UART= MODULO A; MODULO C;

ADC = non disponibile sul processore, ma possibilità di collegamento tramite PPI;

PWM A 3 FASI= MODULO A (per coppia complementare del canale AH/AL), MODULO B (per coppia complementare del canale BH/BL), MODULO C (per coppia complementare del canale CH/CL),)

PWM SINGOLO = USCITE GP DEI TIMERS PRESENTI SUL PIN 1 DEDICATO (TMRx) CON CLOCK IN INGRESSO DI OGNI TIMER TACLKx O PER TUTTI I TIMERS TMRCLK (MODULO A PIN 3 DEL CONNETTORE);

INTERRUPT = SUL PIN 2/4 DEDICATO DEL CONNETTORE ;

Tutto questo si traduce definitivamente in Visual Paradigm negli slots, ossia delle classi che permettono la comunicazione attraverso i tile processors. Come si può vedere dalla figura sotto, a seconda del tipo di CPU, possono riuscire a connettere o 8 moduli contemporaneamente o come per il Blackfin, solo 4 come da tabella 6.4. Il singolo slot rispecchia la simmetria dei segnali scelta e da essa non può discostarsi perché condivisa nell' intero satellite.

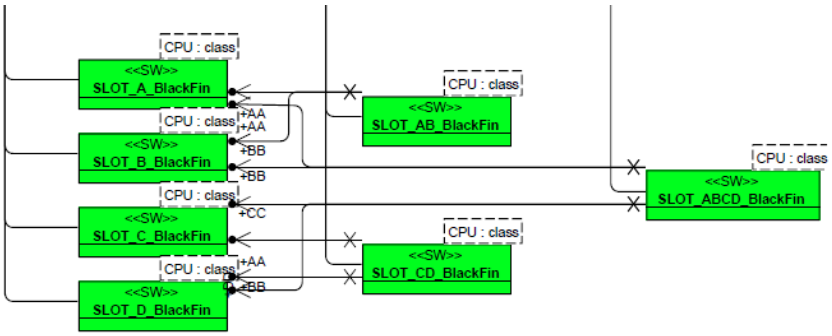


Fig. 5.14 Slots (1) – Visual Paradigm

In questo modo, se volessimo modificare, aggiungere e controllare ogni singolo modulo, basterebbe solo andare ad effettuare il cambiamento e verificare che il tutto si possa interfacciare secondo lo slot utilizzato. Ciò è molto utile senza andare a rimappare ogni singolo pin.

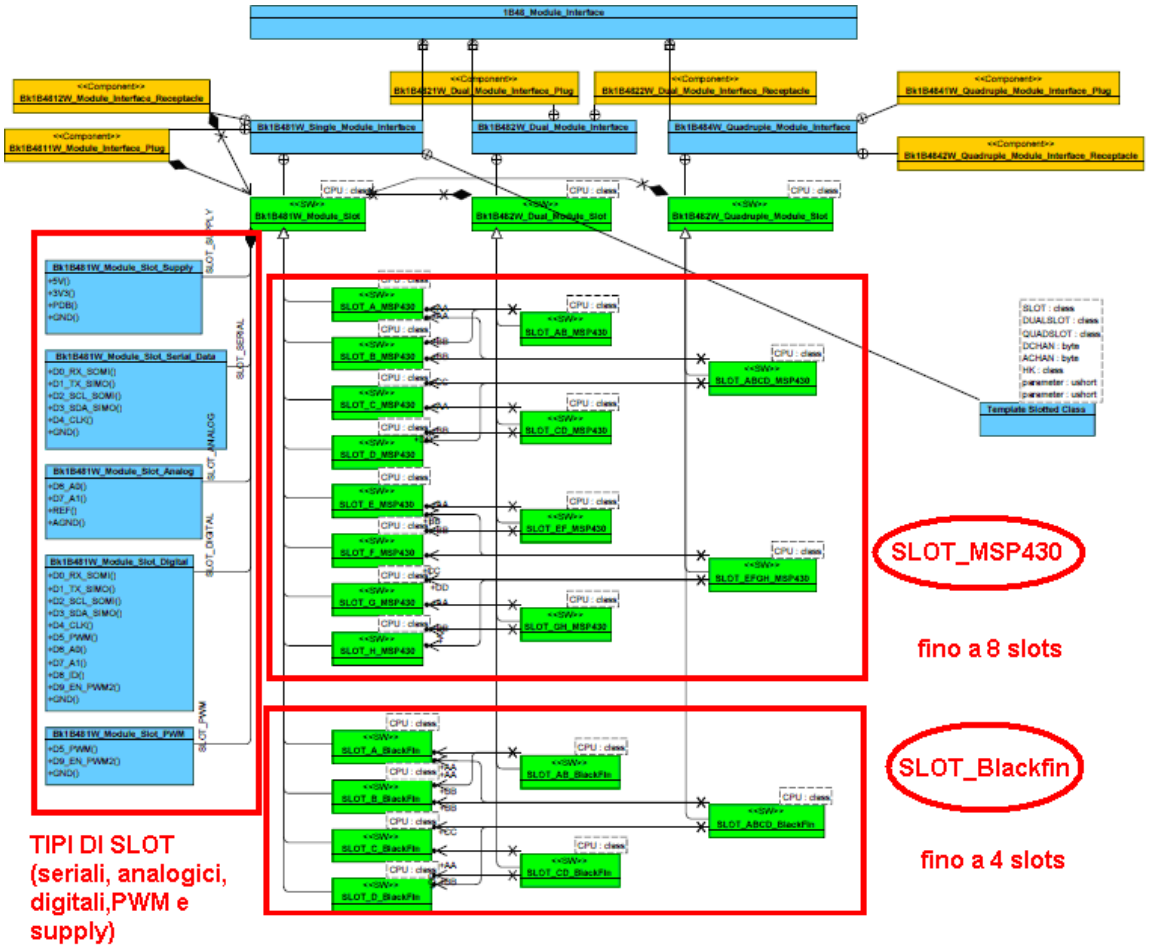


Fig. 5.15 Slots (2) – Visual Paradigm

5.2.2.2 Mappatura secondo i moduli OBC

Per quanto riguarda questo tipo di moduli, essi sono stati aggiunti in fase di progetto per far fronte alla necessità di vedere il processore non solo come master, ma anche dal lato di dispositivo pilotato da un altro processore che ne determina il funzionamento all' avvio (BMODE 0 -2) e lo disabiliti se necessario. Inoltre, l' OBC potrebbe stabilire un canale di comunicazione attraverso il protocollo SPI o I^2C con la stessa identica distribuzione del segnale vista in precedenza per i moduli standard. Non solo, visto che è stato affrontato più volte il discorso dell' assenza di un convertitore per questo processore, si è scelto di impiegare il canale analogico, in prima battuta, come mezzo per controllare tensioni e correnti di alimentazione. Questi stessi moduli, anche se non dichiaratamente scritto, forniranno l' alimentazione da 5V per il corretto funzionamento di alcuni current sensors e quella a 12V per i regolatori buck.

PIN CONNETTORE	SEGNALI PERIFERICHE DSP	MODULO OBC_A	MODULO OBC_B	MODULO OBC_C
11	D0/RX/SOMI	PH1/SPI1MISO PH7/UART1RX/TMR7 → 161 + 153	BMODE0 → 162	
9	D1/TX/SIMO	PH3/SPI1MOSI + PF6/UART1TX → 159 + 154	BMODE1 → 162	

7	D2/SCL/SOMI	*SCL* → **173**	*SCL* → **173**	
5	D3/SDA/SIMO	*SDA* → **172**	*SDA* → **172**	
3	D4/CLK/	PH2 /SPI1SCK → **160**	PH0/ SPI1SS → **162**	
1	D5/ PWM	PF15 /PPI15 → **165** (wake up from hibernate)	BMODE2 → **162**	
12	D6/A0	CS_VOUT (3.3V)	CS_VOUT (1.4V)	CS_VOUT (2.5V)
10	D7/A1	VS_VOUT (3.3V)	VS_VOUT (1.4V)	VS_VOUT (2.5V)
4	D8/ID/INT	NMI → **147**	RESET → **146**	
2	D9/EN/PWM2/INT	EN (3.3V,2.5V)	EN (1.4V)	

Tab. 5.16 Assegnazione pins ai moduli OBC A,B.C

MODULO_OBC_A e MODULO_OBC_B e MODULO_OBC_C sono moduli dove il processore ADSP-BF518F16 funzionerà da slave, mentre il master sarà lo OBC appunto.

PF15 sul pin del connettore 1 permette di risvegliare il processore dallo stato di ibernazione da un altro dispositivo che lo comanda LOW.

Dai pins 11-9-1 si può comandare la modalità di boot dopo il reset: BMODE0, BMODE1 e BMODE2.

Pin 4 è un NMI,cioè un non maskerable interrupt utile nella fase si reset insieme al watchdog.

Sul pin 2 dei due moduli 0BC_A e OBC_B gestiscono l' abilitazione o meno di tutte le alimentazioni.

CASISTICHE :

- **POSSIBILE CASO 1**: EN (1.4V) -> LOW, EN (3.3V,2.5V) = HIGH
  AZIONE: SI SPEGNE IL REGOLATORE CHE FORNISCE 1,4V (a 200mA) E IL PROCESSORE VA IN IBERNAZIONE (segnalato anche da EXT_WAKE) .
  STATO: IL CORE E' SPENTO (CCLK e SCLK sono diasattivati), MENTRE LE TENSIONI DI ALIMENTAZIONI VDDEXT E VDDMEM SONO ANCORA FORNITE ALL' ADSP. I CONTENUTI DI TUTTI I REGISTRI SONO PERSI TRANNE QUELLI DEL REGISTRO VR_CTL SCRITTE IN UNA MEMORIA NON VOLATILE. PER L' USO DELLA SDRAM, E' IMPORTANTE CHE SCKE SIA BASSO PER EVITARE COMPORTAMENTI ERRATI NELLA MEMORIA: SDRAM IN POWER-DOWN (CKE basso con NOP o comando di INIBIZIONE) CON DISATTIVAZIONE DEGLI INPUT E BUFFERS DI OUTPUTS (escludendo CKE). IL DEVICE SDRAM NON PUO' RIMANERE NELLO STATO DI

POWER-DOWN PER UN TEMPO PIU' LUNGO DEL TEMPO DI REFRESH SE NON AVVIENE NESSUNA AZIONE DI SELF-REFRESH (comando di autorefresh con CKE disabilitato) APPUNTO.

WAKE UP: ATTRAVERSO CERTE SORGENTI ABILITATE (con l' impostazione dei bits giusti nel SIC_IWRx). IL SEGNALE EXT_WAKE in OR con le sorgenti in questione provvederà ad abilitare il regolatore.

PF15 VA BASSO (evento esterno general-purpose)

RESET HARDWARE (attivazione dal reset PIN)

EN (3.3V, 2.5V)

- **POSSIBILE CASO 2**: EN (1.4V) -> LOW, EN (3.3V,2.5V) -> LOW

  AZIONE: SI SPEGNE IL PROCESSORE E LA MEMORIA .

  STATO: IL PROCESSORE E' SPENTO

  WAKE UP: EN (3.3V, 2.5V) -> HIGH, EN (1,4V) -> HIGH.

La classe IOMODULE_Blackfin_X (dove X può essere una lettera da A a M) implementa i software drivers per i segnali della tabella 1 che sono indicati come attributi della classe.
I digital I/O (D0-D9) sono associati con la classe I/O driver che è un generico driver SW di I/O Ciascun pin è instanziato dal settaggio del corrispondente parametro template per la funzione desiderata. In questa maniera, il driver può configurare i pins di input e output per una normale funzione digitale di I/O, o per interrupts sensibili al CK di salita/discesa ecc.
Ciascun driver configura ciascun modulo secondo i seguenti parametri template:

- PORT: indirizzo del registro corrispondente alla porta d'uscita associata al PIN desiderato;
- BIT: maschera corrispondente allo specifico BIT della porta d'uscita associata al PIN desiderato;
- INVERT: setta i livelli logici del segnale connesso con i PIN d'ingresso.

L'approccio UML ti permette di costruire sia un satellite che, come in questo caso, uno dei sottosistemi secondo i requisiti della specifica missione, con rischi molto limitati e poco sforzo e tempo di progettazione.
Se volessimo abilitare con il nostro Blackfin il sensore d' immagine sfruttando l' enable del pin PH5 presente sul bit digitale D1 dell' IOmodule_Blackfin_D, allora la sintassi del codice generato sarà :

```
#include " Monocrome_WVGA_CMOS_Imager.h"


template <class module> Monocrome_WVGA_CMOS_Imager <module>::enable(bool on) {

                module::D1.write(on);

}
```

5.3 Sun Sensor UML

Ora, vedremo singolarmente come è stato costruito ogni blocco partendo dai casi d' uso (cioè capire cosa deve fare e come si deve comportare per ogni determinata funzione che è chiamato a compiere) passando per la creazione del componente e terminando con la definizione completa della classe, che sarà costituita da parte hardware e software.
Naturalmente, non si scenderà nel dettaglio ad analizzare ogni classe, attributo o caso d' uso: verrà solo fornita una descrizione lineare delle singole parti lasciando ogni tipo di approfondimento alla documentazione in appendice C.

5.3.1 Imager Processor UML

Il processore è l' elemento fondamentale del sistema perché da una parte deve interfacciarsi con il CMOS sensor, dall' altra deve poterlo fare con i restanti sottosistemi. Da un punto di vista generalizzato, la sua funzione non è solo quella di acquisire l' immagine e immagazzinarla in memoria, ma deve anche poter dialogare con l' OBC secondo le richieste che gli vengono fatte e trasmettere i dati di houseekeping (correnti, tensioni..) attraverso il rispettivo modulo dopo aver convertito i segnali e dati in appositi vettori (chiamati *buffer)* in relazione al comando inviato. Deve inoltre poter essere programmato da un qualcuno che si impegna a configurarlo "a terra" e nello stesso tempo, di una routine software che supervisioni il tutto assicurandosi che non ci siano malfunzionamenti o guasti nel sistema. Chiari i vari usi, si può creare il componente Blackfin mettendo in evidenza le varie periferiche a disposizione dello stesso, e successivamente sarà la volta degli slots disponibili secondo la mappatura dei pin che sarà effettuata.

5.3.1.1 Casi d' uso

Essenzialmente sono stati previsti tre attori: configurator, payload controller e central mission controller; ciascuno interagisce a modo suo con il processore, a terra o in volo, attraverso una routine software mirata ad una gestione specifica del payload o più generica relativamente alla parte di controllo. I casi d' uso sono molteplici, ma principalmente riguardano l' invio di dati (fino a 256Byte) dopo la ricezione di un determinato comando che può essere l' immediata acquisizione o posticipata, il cambio della luminosità o del trigger (in generale di tutti i diversi parametri che potrebbero garantire maggiore elasticità al sistema: si ricordi quanto detto per ciò che concerne un possibile utilizzo dello stesso

processore come sensore di terra), la cancellazione di un certo numero di frame in flash e non sviluppata, la compressione JPEG del frame.

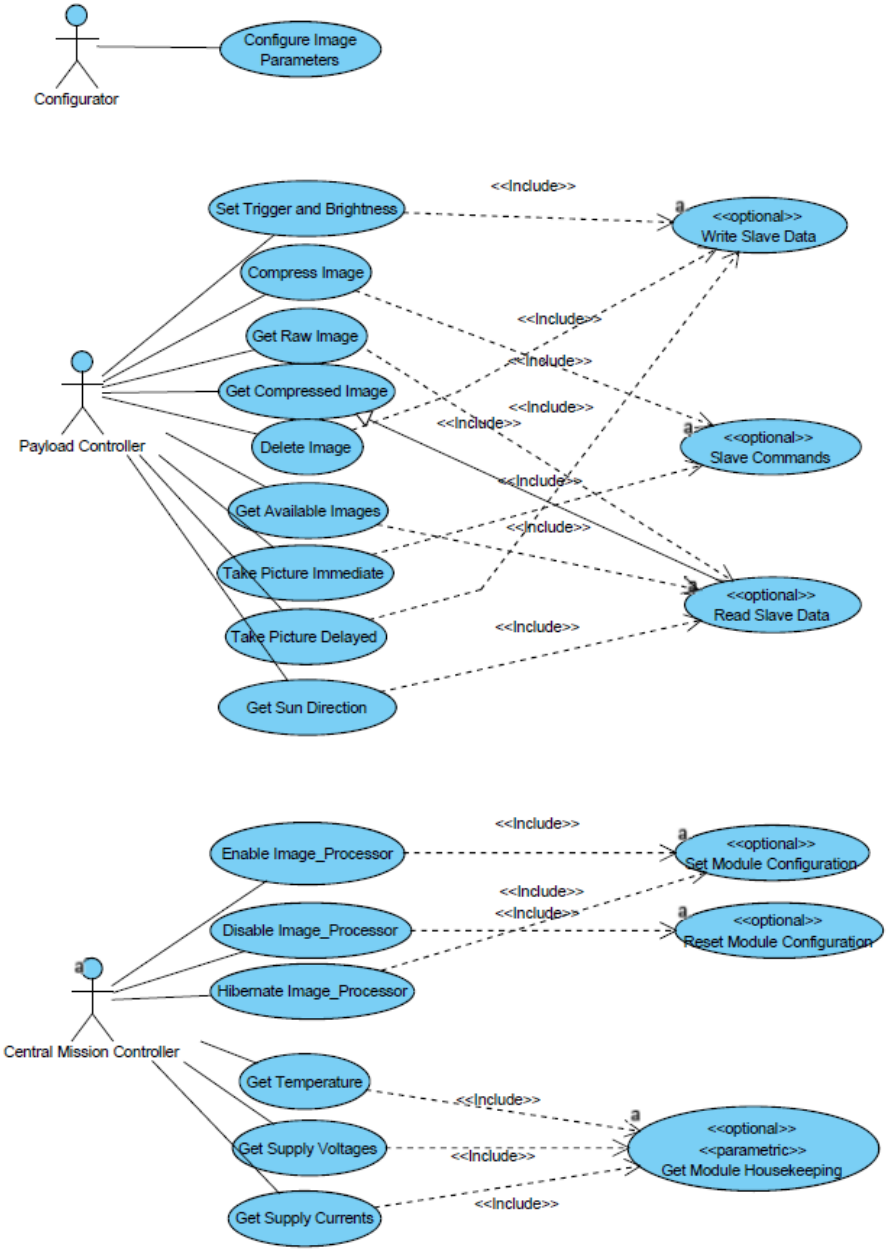


Fig. 5.17 Image Processor Use Cases (Visual Paradigm)

Tutti i casi d' uso appena accennati rientrano poi in altri casi d' uso che fanno capo ad un altro sottosistema che ne determina poi la comunicazione seriale, 1B45_Subsystem_Serial_Data_Bus. E' immediato differenziarli in base al fatto che essi si possano avvalere della ricezione di un singolo comando e l' invio o meno dei dati , cioè di 256 *buffers* a 8 bit: a titolo d' esempio potrebbero essere presi Get Raw Image (comando CMD_GET_RAW_IMAGE) che, una volta ricevuto dal processore, attraverso l' interpreter (metodo presente nella classe Image_Processor per interpretare il comando stesso), invia i dati dell' immagine compressa e Take Picture Immediate che è un semplice comando che

attiva l' acquisizione del frame senza ulteriori dati da trasmettere all' OBC. Brevemente, se si necessitasse del controllo di una determinata corrente o tensione, allora dopo il rispettivo comando CMD_GET_HOUSEKEEPING, attraverso il modulo: HK_REDUNDANCY [LENGTH_HOUSEKEEPING] si ritornerebbe l' ultimo valore misurato della grandezza elettrica richiesta.

5.3.1.2 Componente Blackfin

Questo step è stato molto delicato e laborioso occupandomi molto tempo sia per la traduzione di ogni singola caratteristica sia per la connessione delle diverse parti. Per poterlo creare abbiamo lavorato nel progetto 1C24_Component_Selection, una specie di libreria di componenti disponibile al team di lavoro. Essenzialmente, la CPU è stata divisa nella famiglia MSP430 (già realizzata in precedenza per il satellite) e ora Blackfin, nello specifico, con la mia serie. Una volta creato, il package, al suo interno sono state editate tutte le varie periferiche (SPI,PPI; UART, SPORT e TWI) che si interfacciano con i vari moduli e anche i TIMERS (concentrandoci su quelli utilizzati nel progetto). Per ogni singola parte, si sono creati poi gli attributi e i metodi, e i collegamenti tramite *"use"*: nel nostro caso, nell' utilizzo per esempio della DMA da parte della porta parallela, si è evidenziata questo tipo di relazione tra le due classi che condividono il canale.
Fuori dal package del Blackfin, sono visibili tre determinate classi che servono per irrobustire determinati registri opportunamente mascherati secondo le dimensioni degli stessi: TripleConfigULong, TripleConfigByte e TripleConfigWord (tutti appartenenti al progetto SW_hardening). Il loro funzionamento è abbastanza semplice e prevede una protezione per determinati registri che non sono read-only, ma che possono essere scritti: in pratica, nel momento in cui si vada a scrivere uno o più bit, questi sono triplicati dal software, in modo che il dato non venga definitivamente perso. Sicuramente verrà sfruttata più memoria, ma nello stesso tempo saremmo sicuri che il contenuto di una locazione di memoria in seguito ad un SEU per esempio, non porti ad un fallimento dell' operazione. I registri, protetti e non, sono visibili nei *tagged value* di ogni periferica come header file e saranno poi utilizzati nella generazione del codice secondo la corrispondenza che si è scritto.
Nella documentazione è dettagliatamente approfondito ogni singolo aspetto della stessa.

5.3.1.3 Classe Payload Processor

Una volta effettuate le precedente parti, si è messo il tutto insieme come in figura 4.7, da un lato sono visibili in giallo i vari componenti (di 1C24) che costituiscono la parte hardware della classe; dall' altro, di colore verde, il software con attributi e metodi, ma soprattutto i *tagged value*, valori particolari che esplicitano la stessa classe rendendola duttile per ulteriori usi. La classe di colore blu, rappresenta il

modulo elettronico vero e proprio ed è ciò che si interfaccerà con gli altri sottosistemi (argomento che verrà trattato nel paragrafo relativo ai moduli). Inoltre, per un corretto funzionamento, sono stati aggiunti con le rispettive molteplicità i current sensors, i voltage sensors, lo Switching Regulator, LDO e per le abilitazioni i load switches. Ciascuno ha un compito ben preciso nel sistema, controllando le tensioni e le correnti, fornendo le alimentazioni al processore e abilitandolo o meno.

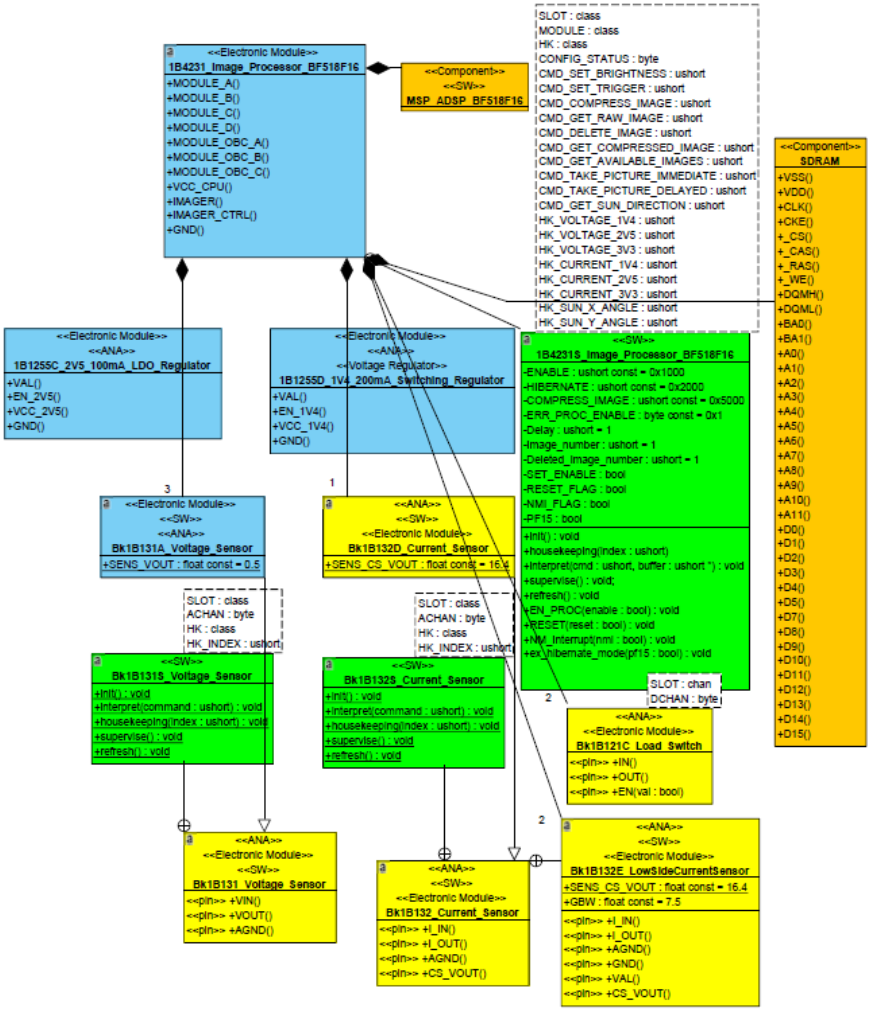


Fig. 5.18 Image Processor (Visual Paradigm)

E' inutile ribadire, che ogni singolo comportamento è espresso secondo quanto scritto nei casi d' uso ed esercita la propria funzione secondo il preciso comando inviato. A parte, ma ugualmente integrata, vi è la Sdram, progettata in 1C24 con le rispettive caratteristiche (produttore, modello, part number, temperatura minima e massima in funzionamento e nel momento della saldatura…) riportate sempre nei *tagged value*.

5.3.2 Image Sensor UML

Naturalmente, essendo il sensore Aptina solo un semplice trasduttore non avrà nessun caso d' uso, e si limiterà nel progetto ad assumere solo la funzione di "slave" acquisendo unicamente le immagini.

5.3.2.1 Componente CMOS Sensor

Come è stato per il processore, anche per il sensore d' immagine sono state riportate le caratteristiche peculiari che lo contraddistinguono mettendo in evidenza tra l' altro temperatura massima e minima, frequenza del master clock e tipo di data output e frame rate. Per quanto riguarda i consumi, sono stati riportati tensione massima e minima di alimentazione con la relativa corrente. Uno spazio particolare è stato dedicato agli attributi ossia le costanti che di default caratterizzano ogni singolo aspetto del sensore come guadagno, larghezza e altezza della array di pixels da leggere o ancora inizio di riga/colonna per un determinato readout.

Questa classe con stereotipo di tipo component è contenuta allo stesso modo fra i trasduttori nel progetto 1C24_Component_Selection.

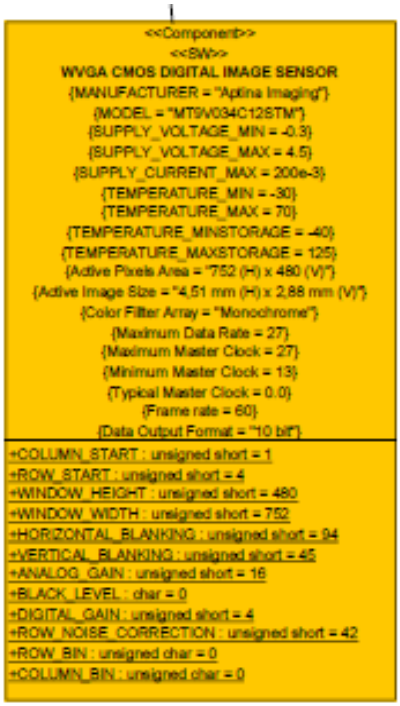


Fig. 5.19 CMOS Sensor (Visual Paradigm)

5.3.2.2 Classe CMOS_Imager

Questa classe è associata solo al componente visto in precedenza, ma insieme alla parte meccanica che riguarda lenti e filtri completerà il blocco del sun horizon sensor del quale l' imager è a sua volta composto. Il modulo elettronico rispecchia quanto visto nella parte hardware con il Reusable Block avendo tra i metodi i due bus IMAGER e IMAGER_CTRL che permettono una connessione con l' imager processor nella classe definitiva, e VCC_CPU e GND.

5.3.3. Sun Horizon Sensor UML

Questa sezione rappresenta il passo definitivo verso la completa realizzazione del sensore e grazie a Visual Paradigm, è stato molto semplice mettere insieme il tutto per garantire una grande chiarezza in ogni sua parte. Da un punto di vista funzionale, il software permette un alto grado di flessibilità e come anche nel mio caso, avendo a disposizione ogni elemento e chiari i concetti, si è trattato solo di unire la classe processore e image sensor. La cosa più importante è stata sicuramente rivedere i casi d' uso perché le varie funzioni del progetto finale sarebbero state più mirate rispetto a quelle viste in maniera più generalizzata per lo stesso processore.

5.3.3.1 Casi d' uso

Facendo un immediato confronto con i casi d' uso visti per il processore, potrebbe sembrare un *copy and paste* di quanto visto nel relativo paragrafo dell' *imager processor*. In realtà non è così, in quanto il sensore avrà gli stessi casi d' uso del processore, ma dovendo avere una funzione più specifica, gli saranno aggiunte tutte quelle casistiche che si potrebbero avere in fase di acquisizione e non.

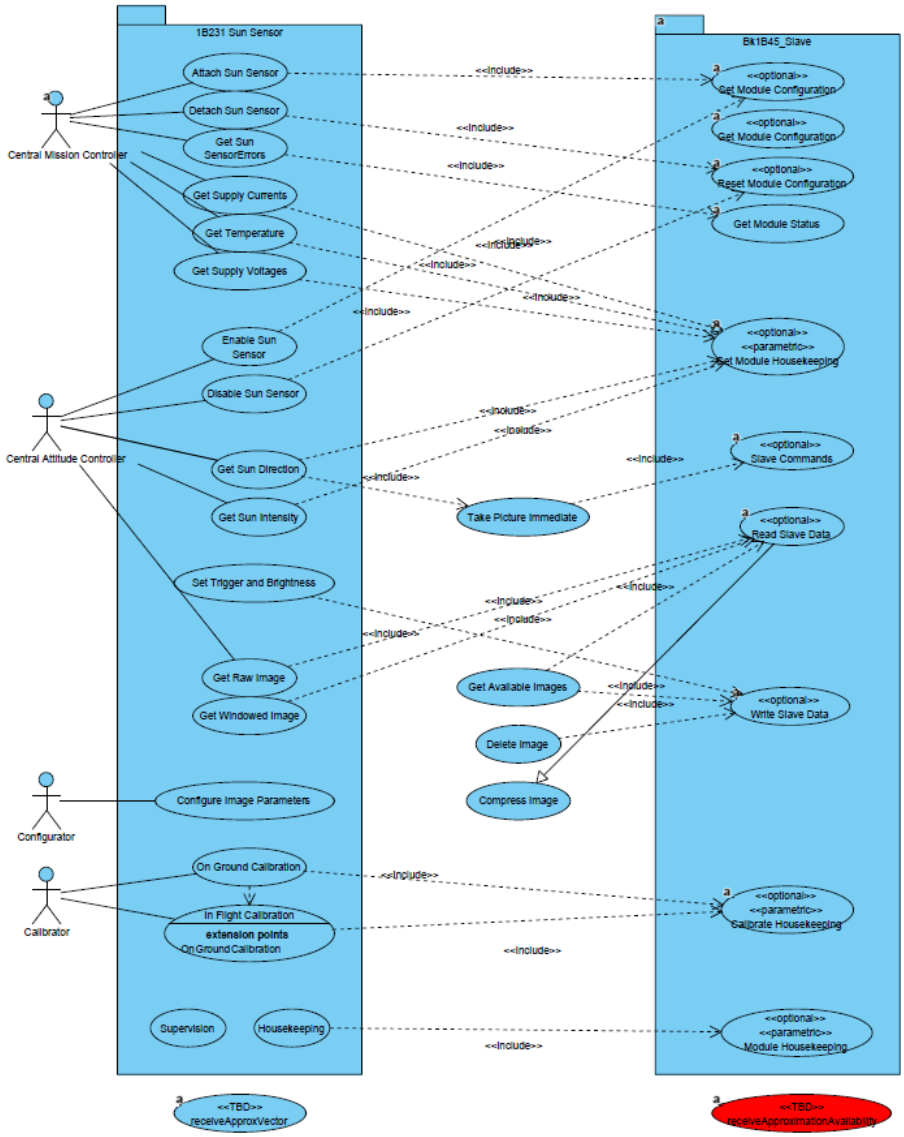


Fig. 5.20 Casi d' uso (Visual Paradigm)

Sostanzialmente può essere suddiviso in quattro parti: configurator e calibrator sono gli attori che da terra si preoccuperanno di configurare rispettivamente i vari registri e calibrare i parametri del tile; invece, central mission controller e central attitude controller saranno due routine software con l' obbligo di gestire la configurazione e l' elaborazione del sensore da due punti di vista diversi. Il primo controller in maniera più globale si occuperà di "sapere" attraverso il modulo di configurazione se lo stesso sensore è effettivamente connesso o scollegato, se vi sono degli errori e tutte le grandezze elettriche viste per il processore, con il modulo di housekeeping.

Per quanto riguarda il secondo controller, quello dell' assetto, non sarà altro che una routine software mirata ad assolvere i vari casi d' uso in volo al fine di acquisire la direzione del sole o un' immagine più piccola, o ancora abilitare/disabilitare il sensore. Come si può vedere, non tutti sono nel package Sun Sensor siccome molti utilizzi sono già stati inclusi e specificati nel processore: per esempio *Get Sun Direction* prevede lo scatto immediato di una foto (quindi include *Take Picture Immediate*) che, a sua volta, è incluso in Slave Commands (package 1B45).

5.3.3.2 Classe Bk1B231A_Sun_Horizon_Sensor

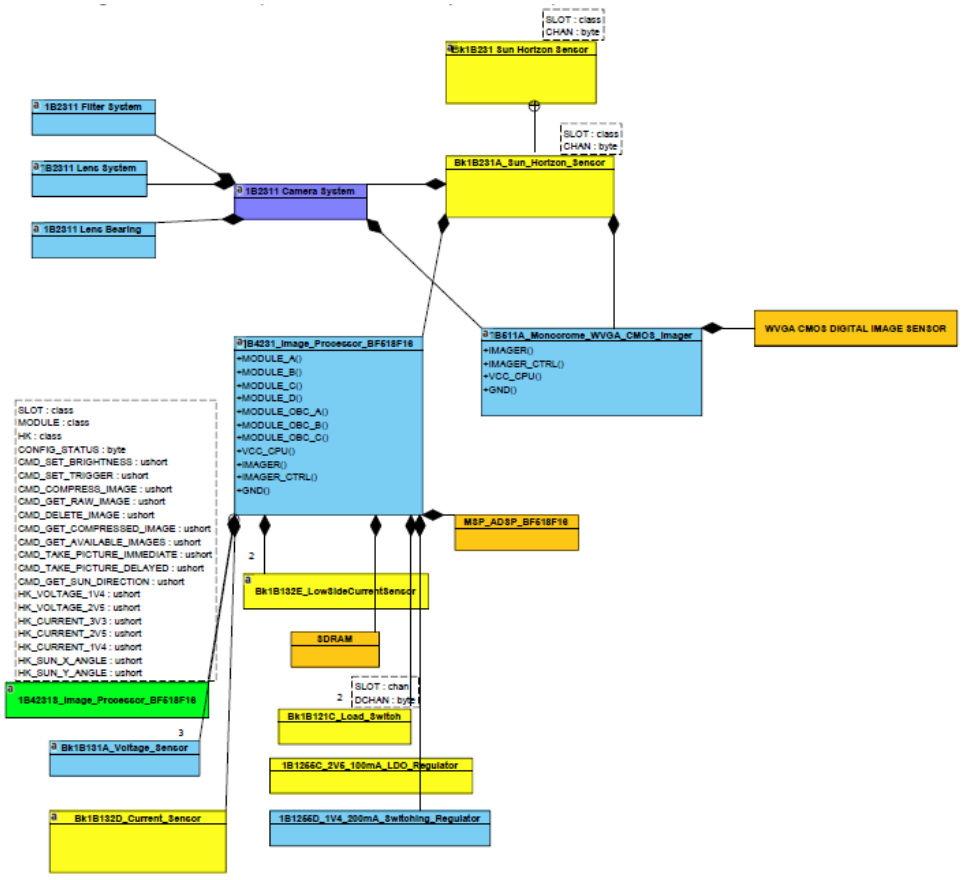


Fig. 5.21 1B231_Sun Sensor (Visual Paradigm)

Questa classe riunisce quanto visto finora includendo parte meccanica, hardware e software (quella naturalmente scritta per il processore). I moduli elettronici del processore e sensore d' immagine sono messi insieme con le relative classi, a formare il Sun Horizon Sensor.

5.4 Descrizione generale

L' operazione generica per la acquisizione di uno o più frame prevede che la scheda debba eseguire le seguenti azioni:

- ricevere un telecomando da parte dell' OBC;
- acquisire un fotogramma da parte del sensore d' immagine;
- salvarlo(i) in memoria SDRAM;
- effettuare un' operazione di troncamento sui pixels ricevuti;
- effettuare una eventuale compressione dell' immagine(i);
- eventuale calcolo del baricentro per la desiderata immagine;
- salvare l' immagine(i) in FLASH;
- salvataggio in FLASH dei vettori contenenti le coordinate angolari dello spot luminoso;

Il flusso di esecuzione del programma principale inizia una volta ricevuto il comando, verrà poi interpretato, in seguito il processore campionerà i boot mode pin ed eseguirà il boot.
L'unica modalità di boot finora testata è la modalità 101, ossia quella che parte leggendo il binario del programma dall'indirizzo iniziale della memoria FLASH e lo trasferisce sulla sua memoria interna veloce per poterlo eseguire.
Seguirà poi l'esecuzione del codice dall'indirizzo iniziale, specificato nei primi byte acquisiti dalla memoria FLASH, dove vengono svolte le diverse inizializzazioni:

- sulle frequenze di funzionamento del processore;
- sulle tempistiche del controller per memorie sincrone;
- sulle due porte di comunicazione UART e SPI;

Infine viene impostata la Interrupt Vector Table, in modo da rendere sensibile il sistema ai seguenti interrupt:

- interrupt generato dal DMA controller al termine dell'acquisizione di un fotogramma;

In realtà, anche se non previsto nell' oggetto della tesi, si dovrebbe garantire anche un funzionamento attraverso le porte UART e SPI mediante interrupt, in questo modo il sistema è più efficiente nelle tempistiche perché non deve rimanere in polling sui registri dei rispettivi flag.
Il processore, passata questa fase d'inizializzazione, si sveglia alla ricezione di un interrupt causato dall'arrivo di un telecomando su una delle due porte, come già visto nel capitolo relativo ai moduli. Il processore lo interpreta, trasmette un acknowledge all' OBC che ha trasmesso il telecomando e a seconda del valore della variabile di stato, il processore eseguirà i task diversi, come specificato nei casi d' uso.
Il seguente diagramma di flusso descrive in modo chiaro come avvengono le varie fasi dell' acquisizione di un' immagine; al termine dello stato è previsto che il processore vada nella modalità di profondo risparmio energetico, che viene detto HIBERNATE, dove il core del processore e la sua alimentazione sono fermati e sono solo alimentate le periferiche statiche. L' ADSP-BF518F16 può essere risvegliato attraverso l' invio di uno strobe (pilotato basso da parte dell' OBC) sul pin PF15 del sensore.

5.4.1 Ricezione del frame

Lo step prevede una serie di sotto-azioni atte a determinare il corretto svolgimento:

1. il processore pilota alto il segnale STANDBY del sensore d' immagine abilitando la sua logica digitale dopo la fase di reset, i clocks interni e i segnali di attivazione della potenza analogica; il sensore è allora pronto per inviare i propri i dati.
2. appena il sensore si troverà nel suo stato di normale operatività, avverrà la programmazione dei registri, mediante il protocollo I^2C implementato sui due rispettivi pin TWI SDA e SCL.
3. i dati con le relative temporizzazioni (attraverso i bus IMAGER e IMAGER_CTRL) sono inviati sulla porta parallela.
4. una volta che il controller DMA si accorgerà in base all' interrupt di fine ricezione, di aver acquisito tutto l' intero frame attraverso una ISR avvertirà il processore di provvedere allo spegnimento dell' imager (segnale STANDBY torna basso), di disabilitare il timer che fornisce il master clock e ritenere quindi concluso il lavoro da parte del sensore.

5.4.2. Salvataggio in memoria - DSP

Una volta che i dati sono giunti sulla porta parallela ad una frequenza intorno ai 26MHz, il passaggio del segnale di sincronismo verticale da alto a basso, sancisce la fine del readout e allacciandoci a quanto detto al punto 4, il sensore va in Power Save. Nello specifico, in tabella sotto, sono stati fatti i seguenti calcoli progettuali:

SYSTEM	OBJECT	SIZE	NOTE
Blackfin	PPI	bb bbbb bbbb → 0000 00bb bbbb bbbb (UNPACKED)	16 bit x 360960 pixels = 5775360 bits = **721,92 KB** (immagazzinati in un primo momento nelle 16 deep FIFO)
Blackfin	PPI CLOCK	Corrisponde al PIXEL_CLOCK	Valore minimo: (tSCLK x 2) – 1.5 = 18.5 ns. Valore nominale (nel progetto) : 40 ns; Ritardo del primo frame sync dall' attivazione del PPI_CLOK = (4 x tSCLK) x 4 = 16. Ritardo massimo dell' uscita TMR3 dalla frequenza del contatore (SCLK/4) = 6 ns Ritardo massimo nel sensore da TMR3 - > PIXCLK - > = 8 + 3 ns;
Blackfin	PPI -> memoria	DAB alla frequenza SCLK	SCLK regolata dal PLL con gli appositi registri
Blackfin	MASTER CLOCK	Generato dal TIMER3 in PWM_OUT mode	**TIMER_WIDTH3 = 4** ossia SCLK/4 = 104 MHz/4 -> ~ 26MHz; **TIMER_PERIOD3 > 403691** (pixel array completo contando anche dark righe e colonne: non incluse nel readout) verrà bloccato dalla ISR della PPI;

Fig. 5.22 Trasferimento dati da PPI a DMA

A tale scopo deve essere impostato il DMA controller:

- Acquisizione a 10 bit ma salvataggio ogni 2 byte messi insieme su una parola di 16 bit, parallelismo della memoria esterna; purtroppo non è possibile avere un ottimo throughput in quanto non è possibile effettuare l' impacchettamento di due pixel in una singola word (nessun data packaging);
- Stop Mode, ossia acquisizione di un singolo fotogramma e generazione di un interrupt al termine dell'operazione
- DMA a 2 dimensioni, ossia 2 contatori distinti, uno per le righe incrementato al termine di quello per le colonne, ideale per questo tipo di applicazioni
- Indirizzo iniziale della memoria destinazione del fotogramma

Infine dopo aver abilitato il DMA controller, anche la porta video PPI verrà settata con i seguenti parametri:

- Parallelismo dei dati a 10 bit;
- General purpose mode;
- Sincronizzazione con frame sync orizzontale e verticale;
- Ricezione di un unico frame (non interlacciamento);
- Numero di righe attive per frame: 480;
- Numero di pixels per riga 752:

Ed infine nell' EBIU, verranno impostate tutte le tempistiche della memoria SDRAM che verranno discusse approfonditamente nella sezione software, in modo che il controller sincrono possa lavorare in congiunzione con essa. Sarebbe stato possibile espandere il sistema, abilitando anche il controller asincrono (magari un' eventuale SRAM) e quindi inserendo nel comando una specie di selezione del tipo di dispositivo sul quale si sarebbe andata a salvare l' immagine, ma tutto è lasciato come sviluppo di questa tesi.

5.4.3 Elaborazione dell' immagine

Immagazzinata in memoria l' immagine, essa deve essere processata per garantire una certa consistenza rispetto alle locazioni dove essa è salvata. Si può ragionare in due diversi modi:

- se si necessita di un' immagine ad alta risoluzione, proprio come è stata acquisita, allora si potrebbe fare il fetch direttamente dalla Sdram (caso 1: GET_AVAILABLE_IMAGE);
- se si vuole un' immagine compressa per essere inviata all' OBC o si desidera calcolare unicamente il baricentro, magari cancellandola successivamente, allora in questo caso bisogna passare dalla fase di elaborazione (caso 2: GET_COMPRESSED_IMAGE).

Il caso 2 prevede una semplice routine (sviluppata) di troncamento del dato: il pixel a 16 bit di cui solo 10 consistenti viene alterato, eliminando i due bit meno significativi; in seguito viene impacchettato in modo da avere su una singola word, due pixel. Il write-back sulla Sdram, permette di avere una memoria più consistente.Questa semplice funzione verrà vista dopo ne paragrafo della descrizione codice.

5.4.4 Salvataggio in Flash

Nella Flash a 2Mbyte è possibile immagazzinare o il risultato del calcolo del baricentro o anche direttamente le immagini compresse o meno. Questo passaggio è utile nel momento in cui si decida di spegnere il processore in modo che i dati non vengano persi (perché ancora nella Sdram), ma siano utilizzati successivamente perché in una memoria non volatile.

5.4.5 Salvataggio e esecuzione del programma Utente

Lo stato di salvataggio di un nuovo programma esterno, come si può vedere dalla figura 3.2, è identificato da un telecomando specifico seguito poi dallo stream di byte che costituisce il nuovo programma da salvare. Il processore prenderà questi dati e li salverà in una locazione fissa e nota della memoria FLASH esterna.

Lo stato invece di esecuzione del programma salvato viene raggiunto da un differente telecomando; questo genera i seguenti step:

- allocazione dinamica di un'area di memoria interna delle massime dimensioni disponibili, assegnandola ad un puntatore a funzione
- passaggio dei dati del programma da eseguire dalla memoria FLASH esterna a tale locazione di memoria
- chiamata di tale funzione per l'esecuzione del programma Questa fase è da implementare, viene solo spiegata per completezza progettuale.

5.5 Flusso di sviluppo del software

La parte software della scheda Payload è stata sviluppata in linguaggio C mediante il software disponibile presso l'Analog Devices chiamato CrossCore Embedded Studio; questo ambiente permette di programmare il processore direttamente sulla scheda mediante la porta JTAG e l'emulatore collegato al PC.

Inoltre il programma permette la visualizzazione di tutti i registri del processore, di tutte le aree di memoria sia interna che esterna e diverse funzioni tipiche di debug, quali finestre di watch, inserimento di breakpoint e esecuzione step by step. Infine include la possibilità di visualizzare il file di gestione della memoria utilizzato dal linker, chiamato linker description file (.LDF) e il disassembly del codice.

Il flusso di sviluppo del software può essere diviso in tre fasi:

- Compiling e Assembling
- Linking
- Loading e Splitting

Le diverse fasi sono rappresentate nella figura 4.3.

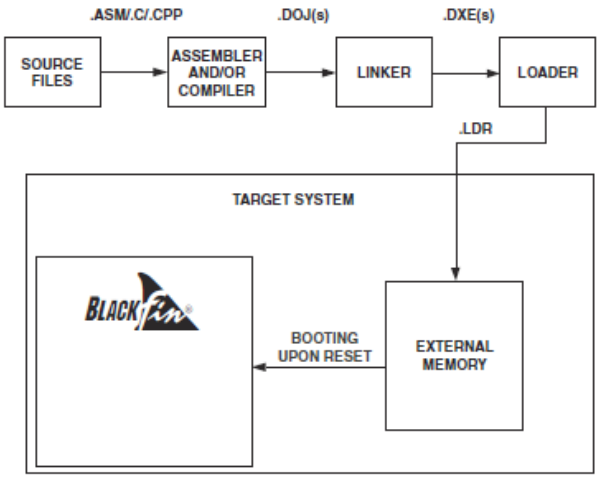


Figura 5.23: Flusso del programma in un sistema stand-alone BF-518F16

5.5.1 Compiling e Assembling

In questa fase, i file sorgente assembler (.ASM) e C (.C) (passando al formato assembler), portano alla creazione di file oggetto (.DOJ). Tale file è composto da parti chiamate input sections. Ogni input section contiene un particolare tipo di codice sorgente compilato o assemblato, ossia vengono raggruppate le constanti, il codice programma, le variabili. Alcune di tali input sections possono contenere informazioni per il debug del codice sorgente.

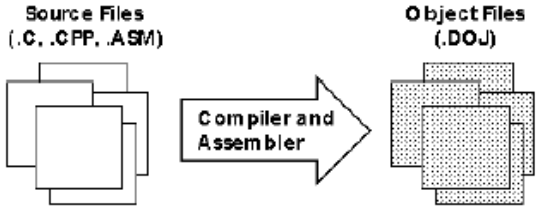


Figura 5.24 : Fase di Compiling

Nella scheda si è sviluppato, come detto in precedenza, tutto il codice sorgente in linguaggio C, tranne il codice di inizializzazione della memoria SRAM, il quale è stato scritto in assembler, data la sua semplicità e l'efficienza richiesta dalla sua posizione che verrà spiegata in seguito. Il compiler e l'assembler agiscono in modo a noi trasparente quindi non è possibile vedere come vengono organizzati i dati e il codice in ogni singolo programma.

5.5.2 Linking

Mediante quanto scritto nel Linker Description File (.LDF), un linker da linea di comando e i la configurazione impostata sul CrossCore, viene letto il file oggetto e prodotto il file eseguibile (.DXE). Il Linker mappa ogni input section (mediante una corrispondente output section nell'eseguibile) in un segmento di memoria, che consiste in un range contiguo di indirizzi di memoria sul processore utilizzato.

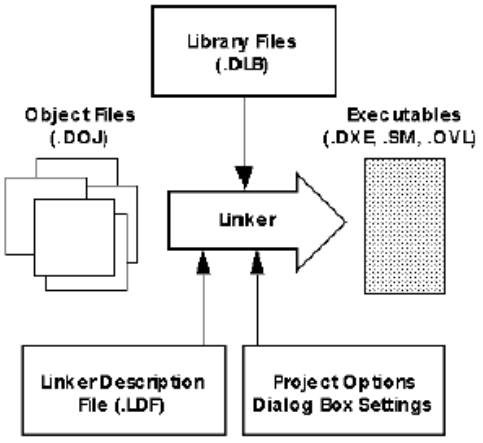


Figura 5.25: Fase di Linking

Ogni input section nel file (.LDF) richiede un nome unico e le principali sono:

- **program** che si riferisce alla sezione dove risiede il codice programma, solitamente mappato sulla memoria codice L1 ma nel nostro caso viene suddivisa tra questa e la memoria SDRAM esterna
- **data1** che si riferisce alla sezione dove risiede i dati globali del programma, come variabili
- **constdata** che si riferisce alla sezione dove risiedono tutte le costanti o le stringhe

5.5.3 Loading e Splitting

Il file eseguibile è processato dall'applicazione elfloader, la quale produce il file (.LDR) chiamato loader file. Questo file sarà quello che permette il boot dalla memoria FLASH, chamato boot-loadable file. Nella figura seguente viene visualizzato come dall'eseguibile si ottiene questo file mediante il loader, che attraverso l'emulatore ICE verrà portato sulla memoria interna del processore per le prove di debug. Nel caso della scheda Payload, l'eseguibile in formato loader era di 338 Kbyte, quindi doveva essere necessariamente messo in una memoria esterna, come la FLASH, per poi essere eseguito mediante un comando di reset. Il software Visual DSP++ mette a disposizione un pacchetto chiamato Flash Programmer che, dandogli il driver per la memoria FLASH utilizzata e il file loader da scriverci dentro, esso convertiva il file loader in binario e lo scriveva sulla memoria esterna utilizzando i registri JTAG.

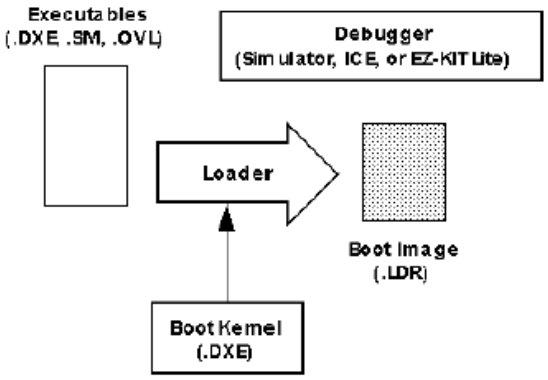


Figura 5.26: Fase di Loading

Il file di loader consiste in un insieme di blocchi preceduti da un'intestazione. Sono queste intestazioni quelle lette e processate dal kernel contenuto nella PROM durante il boot.
L'Analog Devices mette a disposizione un software chiamato Loader Viewer, il quale permette di evidenziare i diversi blocchi di dati del file loader e ne facilita la comprensione, dato che il file è in formato binario:

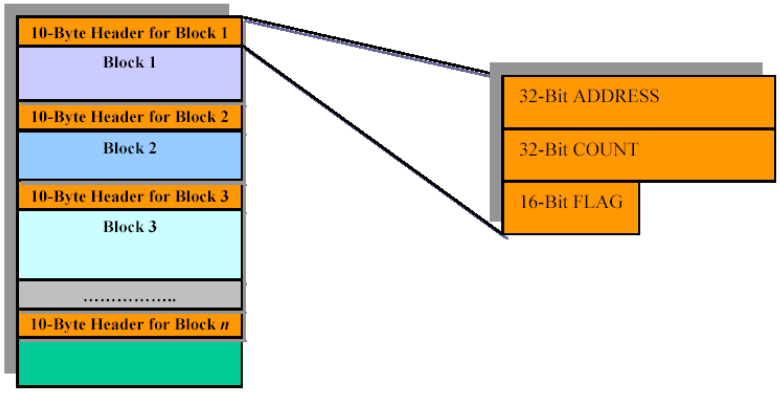


Figura 5.27: Struttura del file Loader(.ldr)

Ogni blocco ha un'intestazione composta da:

- **ADDRESS (4 bytes)** – l'indirizzo destinazione, in cui il blocco sarà scritto sulla memoria
- **COUNT (4 bytes)** – il numero di byte del blocco
- **FLAG (2 BYTES)** – specifica il tipo di blocco e i comandi di controllo

L'ultima sezione dell'header, è quella che definisce se il blocco è un blocco d'inizializzazione, ossia un blocco che dovrà essere eseguito prima del codice dell'applicazione, oppure se è un suo blocco di dati. Inoltre fornisce le informazioni di quale modalità di boot è stata settata, l'indirizzo d'inizio dal comando di reset, detto reset vector, se il blocco di dati è l'ultimo da scaricare, se è da ignorare o considerare e se il blocco è un blocco con dati tutti nulli, utilizzato per inizializzare dei buffer, poi utilizzati dal codice C.

5.6 Mappa di memoria

Il linker, associa a delle sezioni di memoria un range d'indirizzi, che sarà quello utilizzato per accedervi, oltre a definirne il tipo, cioè se RAM ossia leggibile e scrivibile o ROM, cioè solo leggibile. Inoltre associa le input section contenute nel file oggetto ad alcune di queste sezioni di memoria; tutto questo viene svolto mediante i parametri forniti mediante il Linker Description File (.LDF), il quale fornisce quindi tutta la mappatura della memoria di tutto il sistema che s'interfaccia con il processore BF518F16.
La prima parte del file contiene la mappatura della memoria, interna ed esterna. Le memorie interne hanno indirizzi fissi e le memorie esterne sincrona invece è stata divisa in due sezioni:

- la prima, di dimensioni di 256 Kbyte, contiene la parte di codice dell'applicazione che non sta nella memoria interna
- la seconda, ossia la restante porzione di memoria fino a 16Mbyte, è disponibile per l'allocazione dinamica e quindi viene destinata alla sezione chiamata heap

Il codice che mappa la memoria SDRAM è il seguente:

```
MEM_SDRAM0_HEAP{
      TYPE(RAM) WIDTH(8)
      START(0x00100000) END(0x00FFFFFF)
}
MEM_SDRAM0
      TYPE(RAM) WIDTH(8)
      START(0x00000004) END(0x000FFFFF)
}
```

Bisogna ricordare che qui il massimo indirizzamento della memoria specificato al linker è di 16 Mbyte, ma dalla sezione di progetto hardware sappiamo che solo 8 Mbyte possono essere utilizzati, raggruppati in 4 blocchi da 2 Mbyte ciascuno. Questa mappatura “logica” della memoria non è definibile all’interno del linker, ma deve essere gestita dall’applicazione, utilizzando gli indirizzi definiti, come verrà spiegato nel sottocapitolo successivo.

La seconda parte del file contiene l’associazione delle diverse input sections alle sezioni di memoria precedentemente specificate. Alla memoria interna L1 viene associato il codice programma e la configurazione della cache, nel nostro sistema non utilizzata, mediante il seguente codice:

```
program_ram
{
INPUT_SECTION_ALIGN(4)
INPUT_SECTIONS( $OBJECTS(L1_code) $LIBRARIES(L1_code))
INPUT_SECTIONS( $OBJECTS(cplb_code) $LIBRARIES(cplb_code))
INPUT_SECTIONS( $OBJECTS(cplb) $LIBRARIES(cplb))
INPUT_SECTIONS( $OBJECTS(program) $LIBRARIES(program))
} >MEM_L1_CODE
```

La sezione *L1_code* permette l’associazione del codice alla memoria L1 e la voce *program* lo associa, mentre *cplb* sta per Cache Protection Lookaside Buffer che sono tabelle che mantengono la configurazione della memoria utilizzata come cache, da noi non utilizzata, ma deve essere obbligatoriamente associata in qualche locazione di memoria. Come si vede dal codice, questa sezione chiamata a piacere *program_ram*, viene associata alla sezione *MEM_L1_CODE*, mappata precedentemente in questo modo:

```
MEM_L1_CODE {
      TYPE(RAM) WIDTH(8)
      START(0xFFA08000) END(0xFFA0FFFF)
}
```

Come già accennato in precedenza, l’indirizzo di start è quello d’inizio della SRAM codice interna da 32 Kbyte. Inoltre è utilizzata per il codice dell’applicazione anche la parte di SRAM interna utilizzabile anche come cache da 16 Kbyte, chiamata *MEM_L1_CODE_CACHE*, mappata dal seguente codice e associata alle stesse input sections:

```
MEM_L1_CODE_CACHE {
```

```
        TYPE(RAM) WIDTH(8)
        START(0xFFA10000) END(0xFFA13FFF)
    }
```

Alla parte di memoria esterna SDRAM destinata al codice, chiamata *MEM_SDRAM0*, vengono associate le input sections principali, descritte nel paragrafo del Linking, precedute dalla sections *sdram0*, la quale da la possibilità al sistema di utilizzare questa memoria nel caso non sia sufficiente la memoria interna:

```
sdram
{
INPUT_SECTION_ALIGN(4)
INPUT_SECTIONS($OBJECTS(sdram0) $LIBRARIES(sdram0))
INPUT_SECTIONS($OBJECTS(constdata) $LIBRARIES(constdata))
INPUT_SECTIONS($OBJECTS(data1) $LIBRARIES(data1))
INPUT_SECTIONS( $OBJECTS(program) $LIBRARIES(program))
} >MEM_SDRAM0
```

5.7 Processo di boot

La condizione iniziale affinché si possa procedere con il boot dell'applicazione è che il codice stesso risieda nella memoria FLASH esterna.
Questo obiettivo è stato raggiunto attraverso i seguenti step:

- conversione in binario del formato .LDR mediante un programma scritto in C ed eseguito sul PC dove risiede l'ambiente di sviluppo. La conversione consiste nel passaggio dal formato Intel Hex a binario.
- scrittura del file binario sulla memoria FLASH. Si è quindi realizzato un programma C pilota, il quale viene caricato ed eseguito sulla memoria interna SRAM mediante l'emulato ICE e l'ambiente CrossCore. Tale programma eseguiva due operazioni nella seguente sequenza:

Questo blocco è quello che setta tutti i parametri di configurazione della memoria SDRAM per averla disponibile prima di scaricare tutto il programma dalla FLASH. In questo modo, il processore dopo aver eseguito il blocco d'inizializzazione, inizia a acquisire i dati programma dalla FLASH, così 48 Kbyte della memoria interna verranno riempiti con le parti descritte precedentemente nella sezione di mappa della memoria, e i restanti saranno messi sulla SDRAM esterna.

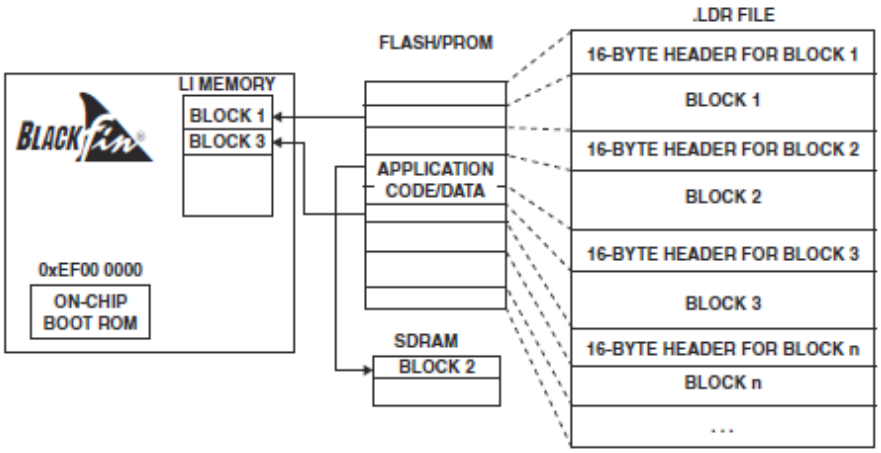


Figura 5.28: Processo di boot del BF-518F16

Ora il processore inizia ad eseguire i blocchi di codice dell'applicazione.
Con questa configurazione, la memoria FLASH non verrà più utilizzata per il codice dell'applicazione e quindi andrà in standby automatico, mentre la memoria SDRAM esterna dovrà rimanere sempre accesa.

5.8 Descrizione codice

In questa sezione verranno descritte parti delle principali funzioni implementate in C e le principali configurazioni del processore e delle sue periferiche per permettere un corretto utilizzo delle memorie e delle diverse interfacce.

5.8.1 Programmazione e Configurazione DSP e periferici

Il processore attraversa una prima fase dopo quella boot, in cui vengono settati tutti i registri per un suo corretto funzionamento.
Al fine di riferirsi a tutti i registri interni, l'ambiente di sviluppo CrossCore fornisce due file di libreria .h, dei quali uno contiene la dichiarazione dei nomi dei registri come *DEFINE* associati ai loro indirizzi interni. Nel file successivo sono dichiarate altre *DEFINE* che dichiarano un puntatore ai nomi dei registri, chiamati con gli stessi nomi preceduti dalla lettera *p*. Quindi nel codice dell'applicazione, si è dovuto semplicemente far riferimento a questi puntatori, utilizzando la loro sintassi, per la loro modifica.

PLL

Registri interessati:

I registri del PLL e del regolatore di tensione non si deve accedere direttamente, bensì si deve usare una funzione bfrom_SysControl, che risiede nella ROM, per cambiare/ leggere i valori in questione. Gli argomenti da passare sono:

- dActionFlags (word in R0)

```
#define SYSCTRL_READ 0x00000000
#define SYSCTRL_WRITE 0x00000001
#define SYSCTRL_SYSRESET 0x00000002
#define SYSCTRL_SOFTRESET 0x00000004
#define SYSCTRL_VRCTL 0x00000010
#define SYSCTRL_EXTVOLTAGE 0x00000020
#define SYSCTRL_OTPVOLTAGE 0x00000040
#define SYSCTRL_PLLCTL 0x00000100
#define SYSCTRL_PLLDIV 0x00000200
#define SYSCTRL_LOCKCNT 0x00000400
#define SYSCTRL_PLLSTAT 0x00000800
```

- pSysCtrlSettings (puntatore in R1 ad una struttura predefinita nell' header file bfrom.h)

```
typedef struct
{
u16 uwVrCtl;
u16 uwPllCtl;
u16 uwPllDiv;
u16 uwPllLockCnt;
u16 uwPllStat;
 } ADI_SYSCTRL_VALUES;
```

- valore zero in R2 (dovrebbe essere lasciato sempre a 0).

Il valore di ritorno è definito dalla bfrom_OtpRead().

C prototipo: **u32 bfrom_SysControl(u32 dActionFlags, ADI_SYSCTRL_VALUES *pSysCtrlSettings, void *reserved);**

- **PLL_DIV**: permette di impostare la frequenza del core (CCLK: bits 5:4) e quella del sistema (SCLK: ultimi 4 bits del registro) dividendo la frequenza del VCO. Essendo VCO = 312 MHz e volendo impostare CCLK (VCO/2) > SCLK = (VCO/3), allora il valore del registro sarà: 0x0013

- **PLL_CTL**: permette di impostare la frequenza del VCO che varia da un minimo di 72MHz sino a 400 MHz; siccome abbiamo connesso un quarzo da 26 MHz, allora impostiamo un x12 per rispettare il range permesso di frequenze portando f_{VCO} a 312 MHz. Perciò MSEL[5:0] = 12 cioè complessivamente PLL_CTL = **0x1800**.

- **VR_CTL=** configura una serie di parametri collegati allo stato di ibernazione come il clock che andrà alla SDRAM che dovrà essere basso anche durante il reset (se questo non avverà si perderanno i dati nella memoria volatile) e le sorgenti per risvegliare il processore come il pin PF15. In dfinitiva, **VR_CTL = 0xB4B0**.

LISTATO d' inizializzazione PLL

```
void Init_PLL(void)
{
ADI_SYSCTRL_VALUES write;
write.uwPllCtl = 0x1800;            //PLL Divide register – CCLK = VCO/2 -> 206 MHz - SCLK = VCO/3 -> 104 MHz
write.uwPllDiv = 0x0013;          // turn on PLL and set VCO requency at 12x 312
write.uwVrCtl = 0XB4B0;          //  setting  for hibernate mode
bfrom_SysControl (SYSCTRL_WRITE | SYSCTRL_PLLCTL |SYSCTRL_PLLDIV, &write, NULL);
*pSIC_IWR = 0x1;                    // Interrupt wake-up Register set first bit for PLL wake-up.
ssync();
idle();
}
```

GP PORTS (for STANDBY)

PFx, PGx e PHx sono sincronizzati al clock del sistema SCLK e bisogna tenere presente che se attivati come interrupts hanno bisogno di 4 SCLK dal momento in cui il segnale viene attivato al momento in cui viene interrotto il normale flusso di programmazione del core.

Registri interessati:

- **PORTxIO_DIR**: permette di impostare la direzione del GPIO associata a quel determinato pin. Nel progetto, come output quindi impostiamo il bit5 a 1: **PORTHIO_DIR = 0x0020**;

- **PORTx_FER**: abilita o meno la funzione della periferica associata a quel determinato pin. Viene lasciata al reset -> 0 (in particolare sul bit 5) per GPIO mode.

- **PORTxIO_SET/PORTxIO_CLEAR/PORTxIO_TOGGLE** = permette di modificare lo stato di una porta (di pochi pin o di un singolo pin) attraverso un W1S (write-1-to set),W1C (write-1-to-clear) e Write 1-to toggle.

- PORTxIO_POLAR/ PORTxIO_EDGE / PORTxIO_BOTH= dopo aver impostato a '1' per abilitazione come input PORTxIO_INEN, la richiesta di interrupt si può avere secondo lo stato del pin (alto o basso sul livello), sul fronte di transizione (bassa -> alta o alta -> bassa) o su entrambi i fronti (bassa -> alta e alta -> bassa). Non rilevante nel mio caso in quanto non utilizzo degli interrupts sui GPIO.

LISTATO d' inizializzazione GPIO

```
void Init_ProgrammableFlags(void)
*pPORTHIO_DIR |= PH5 ;              // enable as output
*pPORTH_FER &= PH5;                 // set PH5 high
}

*pPORTHIO_SET &= PH5;               // set PH5 high in init_PPI
*pPORTHIO_CLEAR &= PH5;             // set PH5 basso in ISR of PPI
```

PPI_

Registri interessati:

- **PPI_FRAME**: numero di righe per frame (un ciclo completo di PPI_FS2): **LINES_PER_FRAME** = **480**

- **PPI_COUNT**: numero di pixels ricevuti per ciascuna riga - 1 (un ciclo completo di PPI_FS1): **PPI_COUNT** = **751**

- PPI_DELAY = ritardo in cicli di clock PPI_CLK dopo l' attivazione di PPI_FS1 prima di iniziare a leggere i dati -> non rilevante lasciato di default;

- PPI_STATUS = informazioni sullo stato operativo della PPI (in particolare sulla FIFO profonda 16 bit e larga 16) attraverso 3 sticky bit nel mio caso mascherabili tramite SIC_IMASK:
  UNDRN (FIFO UNDERRUN) = FIFO senza dati genera un interrupt se impostato a '1';
  OVR (FIFO OVERRUN) = FIFO con dati da gestire superiore alla sua capienza e non può accettarne altri: un interrupt se impostato a '1';
  LT_ERR_UNDR (FIFO UNDERRUN) = flag che dice l'avvenuta attivazione di un PPI_FS1 prima che PPI_COUNT sia raggiunto;

- **PPI_CONTROL** = configura la PPI per la voluta modalità operativa:
  POLS = (polarità del segnali di controllo) PPI_FS1 (HSYNC = PIXELS_PER_LINE) e PPI_FS2 (VSYNC = LINES_PER_FRAME) sono attivi sul fronte di salita se = '0' 0x 0000;
  POLC = (polarità del clock) PPI campiona i dati sul fronte di salita mentre li pilota sul fronte di discesa di PPI_CLK = '0' = 0x 0000;
  DLEN [2:0] (lunghezza dati) = nel nostro caso il sensore d' immagine ci fornisce 10 bits = ->**0x0800** ;
  SKIP_EN (skip enable) = permette di bypassare o gli elementi pari o quelli dispari in un flusso di ingresso di dati -> non rilevante lasciato di default (in quanto non interlacciato nel monocromatico);
  PACK_EN (packing mode enable) = impacchettamento di due campioni in un unica word, utile per aumentare il throughput nel caso ricevessimo 8 bit; -> non rilevante lasciato di default (in quanto riceviamo 10 bits e si ha solo un riempimento di zeri tra i bit più significativi da 10 a 15);
  FLD_SEL (Active Field Select) = 0x 0000 per trigger esterni;
  PORT_CFG [1:0] (Port Configuration) = **0x 0020** per 2 o 3 trigger esterni;
  XFR_TYPE [1:0] (Transfer type) = **0x000C** per selezionare la modalità non ITU-R 656;
  PORT_DIR (Direction) = PPI in modalità ricezione se impostato a '0' -> 0x 0000;
  PORT_EN (Enable) = abilitazione della PPI se impostato a '1' -> **0x0001**;

LISTATO d' inizializzazione PPI

```
void Init_PPI(void){
*pPPI0_FRAME = LINES_PER_FRAME ;
*pPPI0_COUNT = PPICOUNT;
*pPPI0_CONTROL = POL_S |POL_C | DATALEN | CFG_GP_INPUT_2SYNCS | GP_INPUT_MODE;
// Upon enabled DMA controller
*pPPI0_CONTROL |= PORT_EN;
ssync();
}
```

TIMER : TMR3 per generare il master clock e TMR0 per generare il segnale EXPOSURE

Registri interessati:

- **TIMER_ENABLE**: abilitazione di ogni registro scrivendo un ‘1’ nel rispettivo bit di controllo. Nel nostro caso, avendo collegato il timer 3 e il timer 5, avremmo **TMR3 = 0x08 e TMR0 =.0x01**

- **TIMER_DISABLE**: disabilitazione di ogni registro scrivendo un ‘1’ nel rispettivo bit di controllo, una lettura di questo registro ritorna un valore identico ad una lettura del TIMER_ENABLE: se il bit in questione, da controllare, è un 1, allora ciò significa che il timer è ancora abilitato. Si ricordi che scrivendo un ‘1’, in PWM_OUT mode , non blocca immediatamente il timer, ma esso continua ad operare finché non si ferma all’ attuale periodo (se PERIOD_CNT = 1) o impulso (se PERIOD_CNT = 0). Solo l’ azione contemporanea di un ‘1’ su TIMER_DISABLE e di un altrettanto ‘1’ su TRUN (in TIMER_STATUS) potrebbe stopparlo immediatamente. Nel nostro caso, avendo collegato il timer 3, avremmo **TMR3 = 0x08** e **TRUN3 = 0x0080.** Per quanto riguarda il timer 0, avendolo programmato per un solo impulso, non necessita di nessuna disabilitazione.

- **TIMER_CONFIG** : specifica la modalità di funzionamento e può essere fatta solo quando il timer non è operativo (controllo di TRUN se fosse stato prima attivato per assicurarsi che sia disabilitato);
  CLKSEL = seleziona il clock del timer che, per il progetto deve essere generato dal clock di sistema. Quindi nello specifico il clock del contatore è SCLK e quindi verrà lasciato di default;

TMODE [1:0] = siccome la modalità desiderata è quella di generare un segnale PWM_OUT, imposteremo il campo con **b#01**.

TIN_SEL = selezione del clock (TACLK o TMRCLK) per la modalità PWM_OUT; siccome abbiamo lasciato CLK_SEL a '0', di conseguenza verrà lasciato anche questo bit di default a '0';

PERIOD_CNT= questo bit, è molto importante in quanto rappresenterà la differenza nelle due configurazioni per i due timers. Per il timer 3, volendo avere più impulsi e non uno solo, bisogna settare a -> **1** questo bit (conta fino alla fine del periodo e non della larghezza) in modo che si possa generare una forma d' onda ripetuta e possibilmente modulata. Per il timer 0, sarà configurato a -> **0**, volendo un singolo impulso (il contatore non arriva fino alla fine del periodo, ma fino alla fine della larghezza).

- **TIMER_COUNTER** = a 32 bits, questo registro di sola lettura (in qualsiasi momento) conserva il suo stato quando viene disabilitato, mentre all' abilitazione viene re-inizializzato dall' hardware basandosi su configurazione e modalità settata. Per noi sarà campionato dall' SCLK come impostato nel registro di configurazione.
  Di dafault, il registro blocca il suo conteggio durante un emulazione per rimanere sincrono con il software e porta il suo rispettivo pin di output a fornire una forma d' onda d' uscita allungata.
  Avendo impostato CLK_SEL = 1, il counter inizierà a contare dal valore 0x1.

- **TIMER_PERIOD/TIMER_WIDTH** = in PWM_OUT mode, insieme a TIMER_WIDTH, può essere caricato *on-the-fly* dal rispettivo buffer, in quanto i loro valori cambiano simultaneamente (laddove questi valori sono stati scritti siano stati scritti in precedenza, pena l' uso di quelli usati prima). Come TIMER_WIDTH, le scritture sono atomiche a 32 bits.
  TIMER_WIDTH = contiene il numero della larghezza dell' impulso, che nel nostro caso sarà 4 per il timer3 volendo avere un SCLK/4 ; mentre per il timer 0 bisogna valutare il tempo d' integrazione del sensore: in particolare, il valore scritto nel registro R0x0B, che di default sarà 480 (in numero di righe) per il tempo di righe, ossia il Coarse Shutter Width sostanzialmente.
  TIMER_PERIOD = contiene il numero di periodi che si vogliono avere per il segnale PWM (pattern) da generare. Quando questo valore sarà uguagliato da quello contenuto in TIMER_COUNTER, quest' ultimo ripartirà da 1. Nel nostro progetto, questo valore è importante al fine di trovarne un minimo che sarebbe la matrice di pixels completa del sensore MT9V034 (circa 403691 contando anche le righe/colonne scure che non fanno

parte del readout, ma che verranno prese in considerazione dal SYSCLK per monitorare il livello di nero). A questo valore si devono aggiungere i ritardi (si veda tabella note del PPI_CLK) che sono almeno altri 18. Quindi si potrebbe pensare di impostare TIMER_PERIOD = 403710, ma per un lungo tempo d' integrazione conviene invece stimare un valore superiore come minimo e cioè:

F (Total Frame Time) = V (Vertical Blanking) + N° rows x (row time) = 444154 master clock/pixel clock.

Quindi questo valore sarà inserito come riferimento in TIMER_WIDTH per il Timer0 e come minimo in TIMER_PERIOD per il Timer3 -> 444200 = 0x6C728

LISTATO d' inizializzazione TIMER3

```
void Init_Timer3(void){
*pTIMER3_CONFIG = 0x0009          // configures as PWM mode
*pTIMER3_WIDTH = PULSE_WIDTH3;
*pTIMER3_PERIOD = PULSE_PERIOD3;
*pTIMER_ENABLE |= TMR3;
}

void Init_Timer0(void){
*pTIMER0_CONFIG = 0x0005          // configures as PWM mode
*pTIMER0_WIDTH = PULSE_WIDTH0;
*pTIMER_ENABLE |= TMR0;
}
```

In PPI interrupt service routine:

```
*pTIMER_DISABLE |= TMR3;            // for power save on master clock
*pTIMER_STATUS |= TRUN3;
```

TWI (Two Wire Interface)

Registri interessati:

- **TWI_CONTROL**: usato per abilitare il modulo, stabilendo una relazione tra il clock del sistema (SCLK) e il controller TWI stesso. Il riferimento temporale interno deriva da SCLK con un valore di PRESCALE interno da impostare che si ottiene dalla formula :

  PRESCALE: f_{SCLK}/ 10MHz

  PRESCALE[6 : 0]= 10 essendo il clock impostato a 104 MHz per il progetto,

  TWI_ENA = abilitazione del controller TWI che è raccomandabile da fare nello stesso momento in cui impostiamo il valore di PRESCALE per permettere di rilevare dalla logica la presenza di un bus impegnato o meno. Quindi TWI_ENA = 0x80.

  Perciò complessivamente avremmo: **TWI_CONTROL = 0x008A**

  NB: in questo registro c' è un bit SCCB che abilita un' interfaccia compatibile con l' I2C, l' SCCB (serial camera control bus),utile per programmare i registri di certi sensori d' immagine, specialmente prodotti dalla OmniVision con la possibilità di gestire più slaves con un terzo filo d' abilitazione.

- **TWI_CLKDIV**: serve per creare una durata a livello alto e basso del SCL, siccome il segnale di clock è in uscita in master mode con una frequenza che può variare da meno di 20 KHz a 400KHz (fast mode).

  Perciò volendo avere uno SCL di 400 KHz, avremmo un periodo di 2500 ns (1/400 KHz) in riferimento all' accuratezza di riferimento di 10 MHz (periodo = 100ns) e cioè:

  CLKDIV = TWI SCL period / period of reference= 2500ns / 100 ns = 25.

  Ora:

  CLKHI[0 : 7] = il numero di periodi di riferimento di attesa prima che inizi un periodo nel quale lo stesso clock è basso assumendo un singolo master; -> facciamo che da esempio sia 8 (30% di SCL), cioè CLKHI = 8;

  CLKLOW[0 : 7] = il numero di periodi di riferimento tenuti a livello -> di conseguenza a quanto detto in precedenza, deve essere 17, cioè CLKLOW = 7;

  Perciò complessivamente avremmo: **TWI_CLKDIV = 0x0811;**

  NOTA: <u>impostando il fast mode, andiamo alla frequenza massima consentita per il sensore Aptina MTV034, volendo l' avremmo pure potuta ridurre.</u>

- **TWI_MASTER_ADDR** = a 7 bit, contiene il registro dello slave abilitato senza mettere in conto l' eventuale ottavo bit che ne specifica se in lettura o in scrittura (questo perché verrà valutato dallo stato del bit MDIR nel registro di controllo)

MADDR [6:0]: avendo posto i due pins S_CTRL_ADDR0 e S_CTRL_ADDR1 a VAL, gli unici valori che potrebbero essere usati per indirizzare il sensore potrebbero essere 0xB8 (in lettura) e 0xB9 (in scrittura).Per il controller TWI, non considerando il bit 8 avremmo: **MADDR = 0XB8.**

- **TWI_FIFO_CTL** = influenza solo la FIFO (non è legato in nessun modo alla modalità selezionata) e serve a determinare con quale proporzione verranno generati gli interrupts cioè dopo uno o dopo due bytes trasmessi:

  RCVINTLEN (receive buffer interrupt lenght) : un interrupt è generato quando il campo RCVSTAT (in TWI_FIFO_STAT) indica che i due bytes nella FIFO sono pieni (11). Perciò avremmo **RCVINTLEN -> 1** visto i registri a 16 bits del sensore.

  XMTINTLEN (transmit buffer interrupt lenght) : un interrupt è generato quando il campo XMTSTAT (in TWI_FIFO_STAT) indica che i due bytes nella FIFO sono vuoti (11). Perciò avremmo **XMTINTLEN -> 1** visto i registri a 16 bits del sensore.

  RCVFLUSH (receive buffer flush): svuota i contenuti del buffer di ricezione e carica il bit di stato RCVSTAT per indicare che il buffer è vuoto (questo stato è mantenuto finchè il bit non è impostato a '0'): infatti durante un' attività di ricezione il buffer in questione risponde in questo stato alla logica di rx come se fosse pieno. Senza complicare troppo la gestione con gli interrupt, per sola semplicità, lo lasciamo di default **RCVFLUSH** a **0** in modo che che si abbia la normale operazione di ricezione nel buffer e il solo caricamento del suo stato.

  XMTFLUSH (transmit buffer flush): svuota i contenuti del buffer di trasmissione e carica il bit di stato XMTSTAT per indicare che il buffer è vuoto (questo stato è mantenuto finchè il bit non è impostato a '0'): infatti durante un' attività di trasmissione il buffer in questione risponde in questo stato alla logica di tx come se fosse pieno. Senza complicare troppo la gestione con gli interrupt, per sola semplicità, lo lasciamo di default **XMTFLUSH** a **0** in modo che che si abbia la normale operazione di trasmissione nel buffer e il solo caricamento del suo stato.

  Perciò complessivamente avremmo: **TWI_FIFO_CTL = 0x000C**;

- **TWI_INT_MASK** = abilita o meno le sorgenti di interrupt per attivarli come output: infatti ogni bit in questo registro corrisponde ad un determinato interrupt nel registro TWI_INT_STAT. Brevemente si potrebbero originare da:

  RCVSERVM = la FIFO di ricezione ha uno o due locazioni di 8-bit disponibili per essere lette.

XTMSERVM = la FIFO di ricezione ha uno o due locazioni di 8-bit disponibili per essere scritte.

MERRM = è accaduto un errore del master; controllare le condizioni che hanno portato a questo su TWI_MASTER_STAT;

MCOMPM = il trasferimento iniziato del master è stato completato e in assenza di una condizione ripetuta di start, il bus è stato rilasciato.

Quindi impostiamo per semplicità **TWI_INT_MASK: 0x00F0** attivando la gestione di tutti gli interrupts, ma scriveremo le ISRs per servirne solo i principali ai fini del progetto.

SOVFM = il bit di trasferimento completato è stato impostato nello stesso momento in cui il successivo trasferimento ha riconosciuto la propria fase di indirizzamento; quindi, il trasferimento sarà continuato, ma potrebbe essere difficoltoso distinguere i dati di un trasferimento dall' altro.

… altri interrupts relativi allo slave (non rilevanti ai fini del nostro progetto)

Impostiamo ora la direzione del controller e per questo possiamo avere due possibili configurazioni;

SCRITTURA DEI REGISTRI DEL SENSORE

- **TWI_MASTER_CTL** = gestisce la logica associata alla modalità master, includendo le seguenti informazioni:

  DCNT [7:0] = indica il numero di bytes di dato da trasferire dove ad ogni bytes trasferito, questo campo è decrementato per poi generare una condizione di stop a '0'. A titolo di esempio, poniamo il sensore nella modalità di funzionamento Snapshot mode, perciò inviamo 3 byte del rispettivo registro 0x07. **DCNT = 0x00C0**

  MDIR = impostarlo a '0' per definire che si sta facendo una trasmissione; **-> 0**.

  FAST = impostarlo a '1' per impostare un trasferimento in fast mode con le relative specifiche in uso -> 1, quindi **FAST = 0x0008**

  MEN = questo bit si imposta a '0' al completamento di un trasferimento (quando DCNT va a 0) anche con errori; una volta a 0 viene disabilitata la modalità con tutta la logica ad esso associata (che viene appunto resettata). Non vengono più pilotate le due linee di clock e dato. Per quanto riguarda il progetto, verrà impostato a '1' per abilitare la funzionalità di master generando una condizione di start se il bus è idle. **MEN = 0x0001.**

- **TWI_XMT_DATA16** = contiene il valore a 16 bit scritto nella FIFO di trasmissione; viene usato questo registro rispetto a quello a 8 bits per garantire un doppio trasferimento

di due bytes con un singolo accesso nella scrittura della FIFO (in little endian) diminuendo anche la proporzione degli interrupts. All' accesso (precisamente ad ogni accesso si ha il bit di stato di trasmissione (XMTSTAT) caricato) se la FIFO non fosse vuota, il dato da trasferire non sarebbe scritto.

XMTDATA16 [15:0] = si imposti il valore da scrivere nel registro del sensore:

LETTURA DEI REGISTRI DEL SENSORE

Unica variante è modificare obbligatoriamente MDIR oltre ad impostare un nuovo valore per DCNT per leggere diversi registri:

MDIR = impostarlo a '1' per definire che si sta facendo una ricezione di dati; **-> 1**.

Alla fine verrà impostato per la scrittura TWI_MASTER_CTL = …
Alla fine verrà impostato per la lettura TWI_MASTER_CTL = …

- **TWI_RCV_DATA16** = contiene il valore a 16 bit letto dalla FIFO di ricezione; viene usato questo registro rispetto a quello a 8 bits per garantire uno svuotamento della FIFO con un singolo accesso diminuendo anche la proporzione degli interrupts. All' accesso appunto (precisamente ad ogni accesso si ha il bit di stato di trasmissione (RCVSTAT) caricato per indicare che la FIFO è vuota) se la FIFO non è piena, il dato da leggere non sarebbe noto.

  RCVDATA16 [15:0] = qui è presente il dato proveniente dal registro del sensore:

```
void Init_TWI(void){
 flag_byte = 1;
*pTWI_CONTROL = TWI_EN | PRESCALE ;      //enables TWI and simultaneously sets PRESCALE
*pTWI_CLKDIV = CLKHI | CLKLOW;
*pTWI_MASTER_ADDR = 0xB8;                  // address of CMOS sensor for writing and reading
*pTWI_FIFO_CTRL = 0x000C;
*pTWI_INT_MASK = 0x00B0;                   // enables main interrupt for master mode

// If in transmit mode
*pTWI_MASTER_CTL = DCNT |FAST | MEN;
If ((*pTWI_INT_STAT & XMTSERV) == flag_byte) {
     *pTWI_XMT_DATA8 = REGADDR;
```

```
        ssync();
}
else
{     *pTWI_XMT_DATA16 = SNAPMODE;
      ssync();
}
}
```

DMA

Registri interessati:

- **DMAx_PERIPHERAL_MAP** : mappatura della periferica al canale (bit 12 – 15) e se il trasferimento interessa o meno una periferica (bit 6) : -> **0x0000** (poteva anche essere lasciato di default visto che il canale 0 è mappato alla periferica PPI RX/TX;

- **DMAx_X_COUNT**: in modalità 2-D, è il contatore del loop interno che conteggia appunto il numero di righe (cioè pixels da trasferire per ogni riga) : **PIXELS_PER_LINE** = 752 -> **0x02F0**;
- DMAx_CURR_X_COUNT**:** conserva il numero di trasferimenti che rimangono in una riga (del loop interno); al primo trasferimento è caricato con il valore presente in DMAx_X_COUNT e poi viene decrementato ogni volta fino alla penultima inclusa (all' ultima invece assume lo stesso valore iniziale mentre DMAx_CURR_Y_COUNT è decrementato). Il registro DMAx_CURR_X_COUNT sarà 0 solo nel momento in cui avremmo completato l' intero trasferimento (in modalità 2-D), mentre tra due righe, come detto, assumerà il valore di DMAx_X_COUNT.
- **DMAx_Y_COUNT**: solo in modalità 2-D, è il contatore del loop esterno che conteggia appunto il numero di colonne (cioè righe per frame) : **LINES_PER_FRAME** = 480 -> **0x01E0**;
- DMAx_CURR_Y_COUNT: conserva il numero di tutte o le parziali righe (del loop esterno) che rimangono nell' attuale work unit; al primo trasferimento è caricato con il valore presente in DMAx_Y_COUNT e poi viene decrementato ogni volta che il registro DMAx_CURR_X_COUNT passa da 1 -> DMAx_X_COUNT (cambio riga) o 1 -> 0 (ultima riga).

- **DMAx_X_MODIFY:** in complemento a 2 con segno, è l' incremento in byte (senza considerare la dimensione del trasferimento) applicato dopo il trasferimento di ogni elemento nel loop interno fino all' ultimo (non incluso) elemento in ogni loop interno

(cioè incremento del DMAx_Y_MODIFY) -> **2** (è allineato con WDSIZE[1:0] per non portare ad un *DMA error*).

- **DMAx_Y_MODIFY:** in complemento a 2 con segno, è l' incremento in byte ad ogni decremento DMA_x_CURR_Y_COUNT tranne per l' ultimo elemento dell' array 2-D dove cioè DMA_x_CURR_Y_COUNT scade -> **2**.

- **DMAx_START_ADDR:** registro a 32bit, rappresenta l' indirizzo di partenza per il salvataggio dell' immagine; nel nostro caso è -> (unsigned int *) **frame_pointer**
- **DMAx_CURR_ADDR** = registro a 32bit, contiene l' indirizzo dell' attuale trasferimento di memoria per una data sessione DMA; al primo trasferimento il valore è DestAddr, poi successivamente viene incrementato con il passo specificato a ogni trasferimento.

- Alla fine avremmo: DMAx_CURR_Y_COUNT = 1 e DMAx_CURR_X_COUNT = 0 e DMAx_CURR_ADDR = indirizzo ultimo elemento + DMAx_X_MODIFY.

- **DMAx_CONFIG** = configura la DMA per la voluta modalità operativa (solo quando quest' ultima è disattivata):

  FLOW[2:0] = (next operation) STOP MODE, cioè si desidera avviare l' operazione solo una volta cioè solo per tutto il trasferimento dell' intero frame (si blocca quando l' immagine è completamente trasferita in memoria dopo la segnalazione con un interrupt se selezionato). Il bit di stato DMA_RUN (nel registro DMAx_IRQ_STATUS) cambia da 1 a 0, mentre DMAEN non cambia (il canale va in pausa e teoricamente si ha bisogno del software per impostarlo a '0' e disattivare definitivamente il canale; nella successiva work unit, una riscrittura del registro DMAx_CONFIG con nuovi settaggi permetterà di ripristinare i trasferimenti su quel canale**). STOPMODE = 0x0000**.

  Se avessimo usato AUTOBUFFER MODE, la DMA avrebbe operato in maniera circolare e al trasferimento di tutte le words, i puntatori sarebbero stati ripristinati ai valori iniziali , cioè DMA_x_START_ADDRESS, DMAx_X_COUNT e DMAx_Y_COUNT.

  NDSIZE[3:0] = specifica il numero di elementi descrittori in memoria da caricare; lavorando però in stopmode, abbiamo: **RESTART = 0X0000;**

  DI_EN (Data Interrupt Enable) abilitazione dell' interrupt quando l' operazione tramite DMA è terminata **DI_EN = 0x0080** ;

  DI_SEL (Data Interrupt Timing Select) -> **0x0000** solo in modalità 2-D , è stato preferito non abilitare l' interrupt ad ogni completamento di riga (loop interno) in quanto ne avremmo potuto avere fino a 752;

SYNC (Work Unit Transitions): rappresenta come avviene la transizione tra una work unit e la successiva e cioè in modo continuo (a bassa latenza, ma con vincoli su formato dei dati e spazio d' indirizzamento) o sincronizzato (tramite interrupt a più alta latenza, ma con maggiore flessibilità); nel nostro caso, abbiamo una sola work unit e quindi non rilevante lasciato di default;

DMA2D (DMA Mode) (modalità della DMA) = nel nostro caso è a due dimensioni quindi **DMA2D = 0x0010** ;

WDSIZE[1:0] (Transfer Word Size) = rappresenta la dimensione del trasferimento dei singoli elementi dove ogni richiesta/concessione vuol dire un singolo accesso in memoria; è in stretta relazione con DMAx_X_MODIFY nel senso che i passi (gli incrementi) sono dipendenti dalla dimensione del dato. Nel nostro caso, sfruttiamo a massimo la capacità del bus che collega le periferiche e trasportiamo un item da 16bit, cioè **WDSIZE_16 = 0x0004**;

WNR (DMA Direction) = **0x0002** in quanto è un operazione di scrittura in memoria

DMAEN (DMA Channel Enable) = **0x0001** abilitazione di quel determinato canale DMA se impostato a '1'

```
void Init_DMA(unsigned char * frame_pointer){
*pDMA0_X_COUNT= PIXELS_PER_LINE;    // pixels per line da trasferire
*pDMA0_Y_COUNT= LINES_PER_FRAME ;  // linee per frame da trasferire
*pDMA0_X_MODIFY= 0x2 ;  // siccome a 16 bit
*pDMA0_Y_MODIFY= 0x2 ;
(*(unsigned int **) pDMA0_START_ADDR) = (unsigned int*) frame pointer;   // indirizzo iniziale di memoria
*pDMA0_PERIPHERAL_MAP = 0x0000;
*PDMA0_CONFIG = STOPMODE | RESTART | DI_EN | DMA2D | WDSIZE_16 | WNR;
// Subito dopo aver configurato la PPI
*PDMA0_CONFIG = |= DMAEN; // DMA enable
ssync();
}
```

EBIU

Registri interessati

- **EBIU_AMGCTL** (Asynchronous memory global control register) :

AMBEN[2:0]= (abilitazione banchi asincroni) se si fa uso di tutti i banchi di memorie asincrone FLASH/SRAM per il nostro progetto bisogna porre il campo (dal bit 3 – 1) a 1xx e quindi **AMBEN (soluzione flessibile)** : **0x000E** (sennò non rilevante lasciato di default AMBEN= 0x0000**)**.

Se si accede ad un banco asincrono,l' EBIU risponde attivando il segnale di errore sul bus richiesto ritornando la richiesta al master. Viene generata un' eccezione hardware al core se è il richiedente, mentre, se è gestita dalla DMA, l' errore è mostrato nel rispettivo status register.

AMCKEN = abilitazione del CLKOUT per dispositivi esterni che necessitano di un clock; nel nostro caso è posto a '0' in quanto non rilevante e lasciato di default.

- **EBIU_AMBCTL0** (Asynchronous memory bank control 0 register): a 32 bit, regola le tempistiche dei banchi 0 – 1 in base alla memoria sulla quale si interfacciano; non ci soffermiamo sulla configurazione in quanto non collegati ai fini del nostro progetto. Quindi il registro non è rilevante e viene lasciato al valore di default;
- **EBIU_AMBCTL1** (Asynchronous memory bank control 1 register): a 32 bit, regola le tempistiche dei banchi 2 – 3 in base alla memoria sulla quale si interfacciano; non ci soffermiamo sulla configurazione in quanto non collegati ai fini del nostro progetto. Quindi il registro non è rilevante e viene lasciato al valore di default;

- **EBIU_SDRRC:** a 16 bit, provvede a regolare il meccanismo di auto-refresh con un contatore di refresh il cui periodo è programmabile con RDIV e che appunto permette di campionare un comando di auto.-refresh verso la SDRAM.

  RDIV [11:0] = il campo rappresenta in pratica il ritardo desiderato in cicli di clock tra scadenze consecutive del contatore di refresh; di default è 0x081A, nel nostro caso sarà:

  RDIV:

  Calcolato dalla formula:

  RDIV = $((f_{SCLK} \times t_{REF}) / NRA) - (t_{RAS} + t_{RP}) = (f_{SCLK} \times t_{REFI}) - (t_{RAS} + t_{RP})$

  Con f_{SCLK} = frequenza di clock della SDRAM (nel nostro caso 100 MHz con CLKOUT);

  t_{REF} = periodo di refresh della riga (mentre t_{REFI} = intervallo di refresh di una riga) che per la nostra SDRAM è 64ms;

  NRA = numero di indirizzi di riga (ciclo di refresh dell' intera SDRAM): 4096;

  t_{RAS} (valore scritto nel campo TRAS del regitro EBIU_SDGCTL) = attivo al tempo di precarica in numero di cicli di clock; con una tempistica rilassata, possiamo stimarlo in circa 6 CK (si veda tabella sotto)

t_{RP} (valore scritto nel campo TRP del regitro EBIU_SDGCTL) = RAS al tempo di precarica in numero di cicli di clock; con una tempistica rilassata, possiamo stimarlo in circa 5 CK (si veda tabella sotto)

si tenga presente che il valore da sottrarre nell' equazione $t_{RC} = t_{Ras} + t_{RP}$,rappresenta il ritardo richiesto tra i comandi di attivazione banco sullo stesso banco interno.

SPEED GRADE	**CLOCK FREQUENCY (MHz)**	**TARGET t_{RCD} – t_{RP} - CL**	**t_{RCD}(ns)**	**t_{RP} (ns)**	**CL(ns)**	**t_{RAS} (ns)**
-75	133 (7.5ns)	3 - 3 - 3	20	20	20	44 (min)
-7E	133 (7.5ns)	2 - 2 - 2	15	15	15	37 (min)

Tab. 5.17 Ttempistiche principali della SDRAM

Si notino che queste stime sono state fatte con una frequenza di 133 MHz, ma sapendo che la SDRAM è anche PC100 compliant bisogna ragionare con tempi leggermente diversi tenendo conto del periodo che diventa da 7.5 ns a 10 ns.

Quindi facendo i conti : RDIV = ((f_{SCLK} x t_{REF}) /NRA) – (t_{RAS} + t_{RP}) = (($100 \times 10^6 \times 64 \times 10^{-3} / 4096$)) – (6 + 5) = 1551 = **0x0000060F** arrotondato per difetto!

- **EBIU_SDBCTL**: configurato prima del powerup, contiene i parametri specifici e programmabili dei banchi esterni che possono essere configurati anche secondo una SDRAM di dimensioni diverse;

  EBE = (abilitazione del banco esterno):se la SDRAM è disabilitata (bit a '0'), ogni accesso al suo spazio degli indirizzi genera un errore interno del bus e l' accesso non avviene esternamente; **EBE** = **0x0001**;

  EBSZ [1:0] = (dimensione del banco esterno):tiene conto della configurazione della SDRAM secondo la size supportata (da 16 Mbyte a 128 Mbyte) dal processore. Approfondiremo questo discorso dopo per l' uso di una memoria di dimensioni inferiori: nel nostro caso 8 Mbyte. **EBSZ** = **b#00 (16 MByte con modifica "interna" a 8Mbytes spiegata dopo)** ;

  EBCAW [1:0] = rappresenta la page size della SDRAM interna che può essere di 512 byte (00: 8 bits), 1K byte (01: 9 bits), 2 Kbyte (10: 10 bits) e 4k bytes (11: 11 bits) secondo la formula nel nostro caso a a 16 bits sarà: *page size* = $2^{(CAW + 1)}$ con (CAW +1) = *column address width* +1 essenso 1 address bits = 2 bytes. **EBE** = **b#00**;
- **EBIU_SDGCTL**: deve essere seguito da una istruzione SSYNC, contiene i parametri specifici e programmabili che riguardano configurazione e timing;

SCTLE = (abilitazione controller): quando è impostato a ‘0’, tutti i pins di controllo (SDQM[3:0], SCAS, SRAS, SWE, SCKE e CLKOUT) sono nello stato inattivo e il clock non è operativo; deve essere impostato a ‘1’ per le operazioni SDC (come lo è di default al reset);

CL (CAS latency) [1:0] = impostazione della CAS latency ossia il ritardo in cicli di clock da quando il controller accede in lettura ad una determinata colonna della memoria a quando il dato è disponibile sul pin di output : 2 CK per 20 ns a 133 MHz (**b#10**) ;

PASR (partial array self refresh)[1:0] = determina quanti banchi della SDRAM subiscono il self-refresh che naturalmente è un criterio da tenere ben presente nel consumo di potenza. Se avessimo avuto una SDRAM low power, limitandoci alla nostra applicazione e siccome l’ acquisizione di un fotogramma può tranquillamente starci in un solo banco (I° blocco nella figura sopra), ci saremmo al refresh di questo. Quindi, nel nostro caso avendo una SDRAM standard avremo **b#00**; si ricordi che la selezione dei banchi avviene tramite A[19 :18].

TRAS (bank activate command delay) [3:0] = impostazione del valore t_{RAS} da 0 – 15 CK che rappresenta il minimo ritardo necessario per accedere ad una determinata riga, tempo che intercorre tra la richiesta del dato e il comando di precarica. Deve essere $t_{RAS} = t_{RCD} + t_{WR}$ che nel nostro caso dal datasheet deve essere superiore rispettivamente al minimo 20ns + (1CLK + 10ns) = (**5 CK = b# 0101**)(44ns per essere precisi).

TRP (bank precharge delay) [2:0] = impostazione del valore t_{Rp} da 1 – 7 CK che rappresenta il numero di colpi di clock necessari per terminare l’ accesso ad una particolare riga e aprire la riga successiva.Per la nostra SDRAM è minimo 20ns, cioè (**3 CK = b#011**) (stima rilassata);

TRCD (RAS to CAS delay) [2:0] = impostazione del valore t_{RCD} da 1 – 7 CK che rappresenta il ritardo richiesto tra un comando di attivazione riga e un CAS cioè il tempo che passa per il controller tra l’ attivazione di un indirizzo riga e uno colonna durante comandi consecutivi di lettura/scrittura. Per la nostra SDRAM è minimo 20ns, cioè (**3 CK = b#011**) (stima rilassata come prima per t_{RP});

TWR (write to precharge delay) [2:0] = impostazione del valore t_{RCD} da 1 – 3 CK che rappresenta il tempo necessario al controller per far memorizzare il dato prima di un comando di precarica. Per la nostra SDRAM è minimo 1CLK + 10ns (**2 CK = b#10**) questo perché la precarica inizia dopo il primo ritardo di clock e dopo l’ ultimo *write* eseguito.

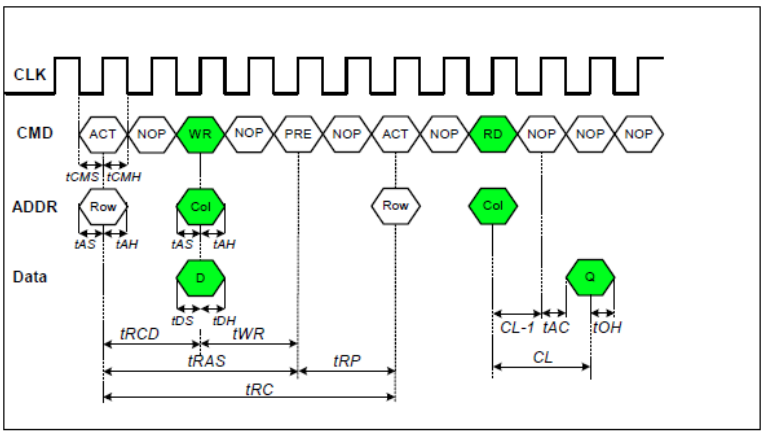


Fig. 5.29 Impulso controllato della SDRAM di scrittura e lettura

POWER-UP START DELAY (PUPSD) = permette l' inserimento o meno di un ritardo aggiuntivo prima del primo comando di precarica di 15 CK utile per i sistemi multiprocessore che si interfacciano con una SDRAM che condividono. Quindi il bit non è rilevante e viene lasciato al valore di default cioè a '0';

PSM (power-up sequence mode) = se impostato a '1', la sequenza del controller 1 prevede gli steps di precarica di tutti i banchi (1), impostazione del mode register (2) e 8 cicli di auto-refresh(3); se a '0' farà le stesse azioni invertendo 2 con 3 ossia 1-3-2. Per la nostra memoria verrà lasciato di default a '**0**'.

PSSE (power-up sequence start enable) = funziona insieme al PSM per specificare il modo e campionare una sequenza di accensione, ma è importante che questo bit sia impostato prima di abilitare la sequenza di power-up (insieme all' evento che prevede un accesso in scrittura/lettura prima che sia stato abilitato lo spazio di indirizzamento per avere garanzia di un bus esterno concesso all' SDC: ciò occuperà diversi cicli per completarsi). Per la nostra memoria verrà impostato a '**1**'.

SRFS (self – refresh settings) = funziona insieme a SCTLE per controllare il self-refresh mode: se questo bit è abilitato, viene allora campionato e perciò ogni volta che si completa un trasferimento, il controller esegue una sequenza di comandi per mettere la SDRAM in modalità di refresh. Per la nostra memoria verrà impostato a '**1**'.

Modalità di ingresso in self refresh: una volta che l' SDC entra in uno stato di idle invia un comando di precarica e successivamente uno di self-refresh che, se si ha un accesso pendente in memoria, verrà automaticamente ritardato fino a quel completamento (l' ingresso in questa modalità può essere monitorato attraverso il bit SDSRA del registro di stato. Si ricordi solo che prima di disabilitare il pin di CLKOUT con il bit SCTLE, ci si assicuri di mettere il controller in self-refresh mode attraverso il bit SRFS per un corretto funzionamento dell' SDC stesso.

Modalità di uscita dal self refresh: si esce solo quando l' SDC riceve le richieste dalla DMA o dal core; nello specifico, se il bit SRFS rimane a '1' prima che della DMA, il controller esce temporaneamente dalla modalità per una singola richiesta e ci ritorna fino al campionamento di un' altra; se il bit SRFS invece, è impostato a '0' prima della richiesta dal core/DMA, il controller esce dall modalità di self-refresh per entrare in quella di auto-refresh. Si ricordi solo che prima di uscir, assicurarsi di abilitare il pin di CLKOUT attraverso il bit SCTLE affinchè tutto funzioni correttamente.

EBUFE (external buffering enabled) = serve per gestire il timing tra sistemi che sfruttano diverse SDRAM connesse in parallelo in particolare, per *buffering* i dati tra il processore e le diverse memorie attraverso un registro e un driver. Nel nostro caso, essendo un semplice sistema non se ne ha bisogno e quindi non essendo rilevante e viene lasciato di default a '0'.

FBBRW (fast back-to-back read to write) = abilita una lettura della SDRAM seguita da una scrittura su cicli consecutivi in sistemi nei quali è difficile gestire il timing di spegnimento della memoria essendo troppo lungo e lasciandolo al processore. Ora, non essendo rilevante viene lasciato di default a '0' (cioè viene inserito un CK tra accessi in lettura seguiti da scritture).

Se invece avessimo avuto una Mobile Sdram, avremmo dovuto impostare anche i seguenti registri:

EMREN (extended mode register enabled) = abilita la programmazione del registro *extended mode* durante lo startup e questo controlla il consumo di potenza in certe SDRAMs low power; se è impostato a '1', allora i bits TCSR e PASR[1:0] controllano il valore scritto nel registro appunto *extended mode*.

TCSR (temperature compensated self-refresh) = segnala alla SDRAM, nel caso peggiore, il range di temperatura per il sistema in modo da valutare quanto spesso i banchi necessitano di avere il refresh durante il self-refresh. Tutti i bits in questo registro devono sempre essere scritti con degli '0'.

- **EBIU_SDBSTAT**: raccoglie le informazioni sullo stato del controller sincrono;

  SDRS (SDC powerup delay) = se è a '0', ciò significa che l' SDC è già acceso, viceversa a '1' vuol dire che la sequenza di powerup deve ancora essere eseguita. E' utile per il *polling* per interrogare l' EBIU circa lo stato del controller sincrono e per sapere se avviare o meno la sequenza di initSDRAM (si veda il listato C sotto)

```
void InitSDRAM(void){
if (*pEBIU_SDSTAT & SDRS) {     // SDC già acceso
*pEBIU_SDBCTL = 0x00000001;
ssync();
 *pEBIU_SDRRC = 0x0000060F; // impostazione del refresh della SDRAM
ssync();
*pEBIU_SDGCTL = 0xE1999941; // configurazione in base al timing della SDRAM
ssync();
}
}
```

In Visual Paradigm, queste inizializzazioni sono state inserite come metodi delle specifiche classi nella sezione *Operation Code Details* al fine di generare il codice. Dapprima però sono state prima compilate tramite CrossCore per controllare la presenza di possibili errori nel software. Tutte le costanti saranno poi specificate nella sezione “attributi”.

5.8.2 Gestione dell’ interrupt

Un interrupt è un evento asincrono che cambia il normale flusso d’istruzioni all’interno del processore, mentre una exception è un evento generato dal software quindi sincrono con il flusso di programma. Entrambi sono gestiti da un sistema di eventi a priorità, dove ognuno di questi può essere associato ad una routine diversa che viene attivata con una priorità fissa.
Il processore impiega un meccanismo di controllo per gli eventi a due livelli. Il System Interrupt Controller (SIC) agisce insieme al Core Event Controller (CEC) per controllare e gestire le priorità tra i diversi interrupt. Il SIC provvede alla mappatura tra le diverse possibili sorgenti periferiche di interrupt e gli input a priorità fissa general-purpose del core. Questa mappatura è programmabile mediante un registro chiamato SIC_IMASK, il quale può mascherare una sorgente d’interrupt al SIC, rendendolo insensibile a l’evento generato dalla periferica associata.

Il CEC supporta nove interrupts general-purpose (IVG7 - IVG15), che diventano sette (IVG7 – IVG13), dato che gli ultimi due sono riservati per le interrupt software handlers.

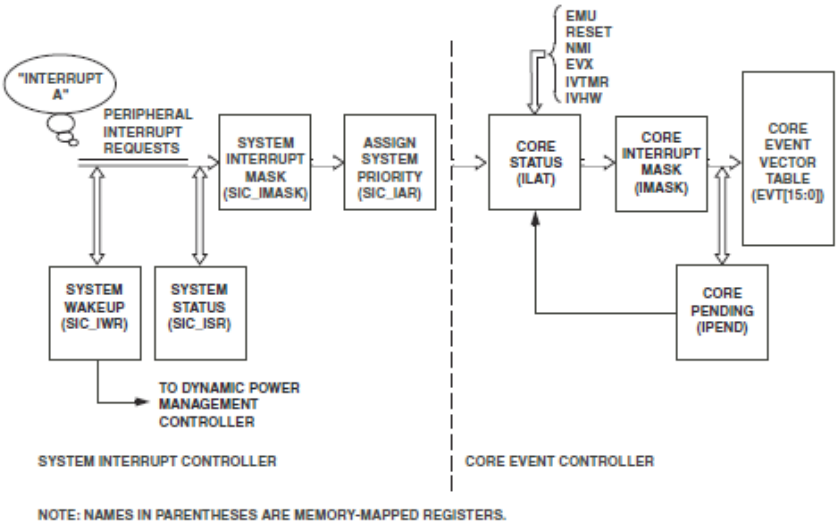


Figura 5.30: Schema a blocchi dell'Interrupt Processing

Ognuno di questi è associato ad un numero identificativo, utilizzato nel registro chiamato SIC_IAR (Interrupt Assignment Register) in cui si associa la periferica all'interrupt general-purpose nel core. Più di una periferica è mappata sullo stesso interrupt del core, e le sorgenti sono messi on OR, senza priorità alcuna.

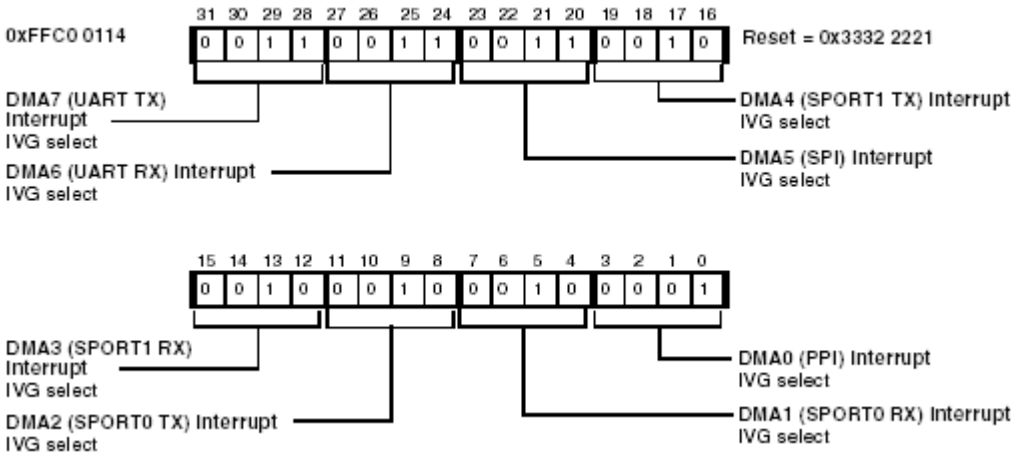


Figura 5.31: System Interrupt Assignment Register 1

Nel programma per il sistema l' interrupt utilizzato è stato essenzialmente uno solo per necessità dettate dal generare un codice:

- DMA 0 generato dal DMA controller in acquisizione della porta PPI in modalità *DMA_STOP*, al termine di un fotogramma dalle righe indicate nel registro del controller. Questo interrupt è mappato all'interrupt general-purpose IVG8, il cui core interrupt ID è 1.

Il codice che implementa tali eventi è il seguente, che va a scrivere sul registro il valore 0x00000001:

```
*pSIC_IAR1 = (*pSIC_IAR1 & 0x00fffff0) | 0x00000001;
register_handler(ik_ivg8, DMA0_PPI_ISR);
```

La funzione register_handler associata le diverse Interrupt Service Routine (ISR) all'interrupt associato alla periferica.

Come spiegato prima si è poi mascherato tutti gli altri interrupt, lasciando il SIC sensibile solo a questo ponendo a '1' i bit associati nel registro SIC_IMASK.

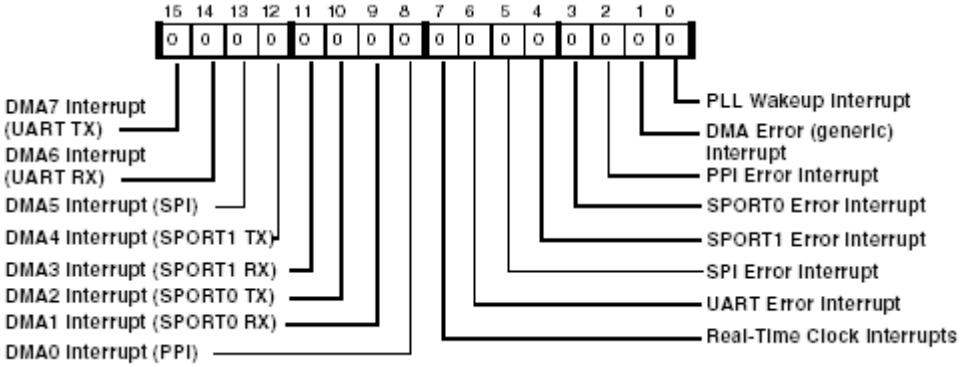


Figura 5.32: System Interrupt Mask Register (parte bassa)

Il codice quindi che maschera tutti gli interrupt possibili tranne questi tre è il seguente:

```
*pSIC_IMASK=0x00000100;
```

Alla ricezione dell'evento, il processore esegue la funzione interrupt handler associata all'interrupt con maggiore priorità e setta il bit associato in un registro chiamato IPEND, il quale mantiene traccia di tutti gli interrupt del sistema al momento caricati nel processore e diventa zero al termine dell'esecuzione della routine.

Dal lato della periferica, ciascuna possiede un registro degli interrupt generati. All'esecuzione della funzione interrupt handler, tale registro va ripristinato al valore che aveva prima della generazione dell'interrupt, ossia va azzerato il bit che identifica l'interrupt pending, affinché al termine dell'esecuzione della funzione la periferica possa generare un altro interrupt associato allo stesso evento, altrimenti la porta ritiene di averlo già generato e tutto il sistema rimane fermo.

In questo stesso modo verrà implementato il driver per la comunicazione mediante la porta SPI, al momento non ancora implementato.

Conclusioni

Questa tesi rappresenta solo l’ incipit per una corretta realizzazione di un sensore di sole in quanto sostanzialmente ha portato al primo passo per un possibile e completo inserimento di un Blackfin nel progetto Aramis. Gli ulteriori sviluppi riguarderebbero innanzitutto la definizione delle periferiche non interessate all’ acquisizione partendo dallo sviluppo di nuovi possibili casi d’ uso e terminando con una mappatura ancora più efficiente dei pins. Dal lato sensore, sarebbe sicuramente più utile lavorare con un image sensor più adatto ad un ambiente spaziale perché oltre a specifiche fisiche che lo renderebbero più robusto alle radiazioni, integrerebbe caratteristiche innovative che si rapporterebbero in maniera più fluida con l’ elaborazione a bordo. Allo stesso modo, dovrebbe essere vagliata la disponibilità degli ultimi processori della serie Blackfin non cercando più un eventuale compromesso legato a tempistiche e costi, ma una CPU capace di processare, con un efficace risparmio di potenza, una più elevata mole di dati con throughput più alto. Globalmente, includendo il lato fisico-meccanico, sicuramente la direzione è segnata dall’ abbandono del singolo pinhole a favore di sensori multi-apertura, che garantirebbero maggiore precisione su FOV più grandi. Infine, è utile ricordare in generale, che la tesi rimane a disposizione per validi approfondimenti essendo parte di un progetto che sfrutta la modularità come principio di base.

Appendice A

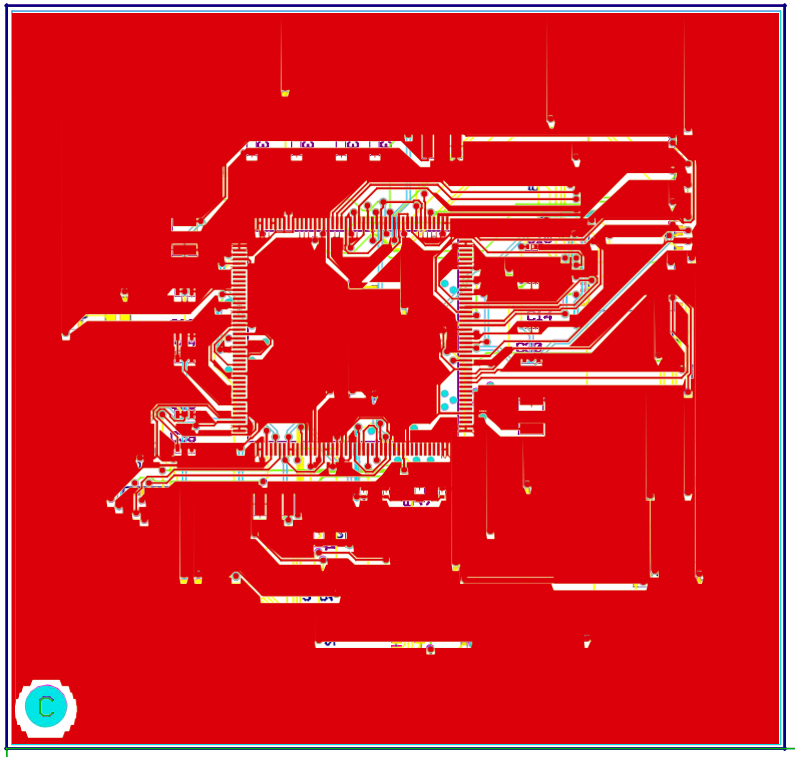
I Versione:

LATO TOP: processore
LATO BOTTOM: sensore, Sdram

II Versione:

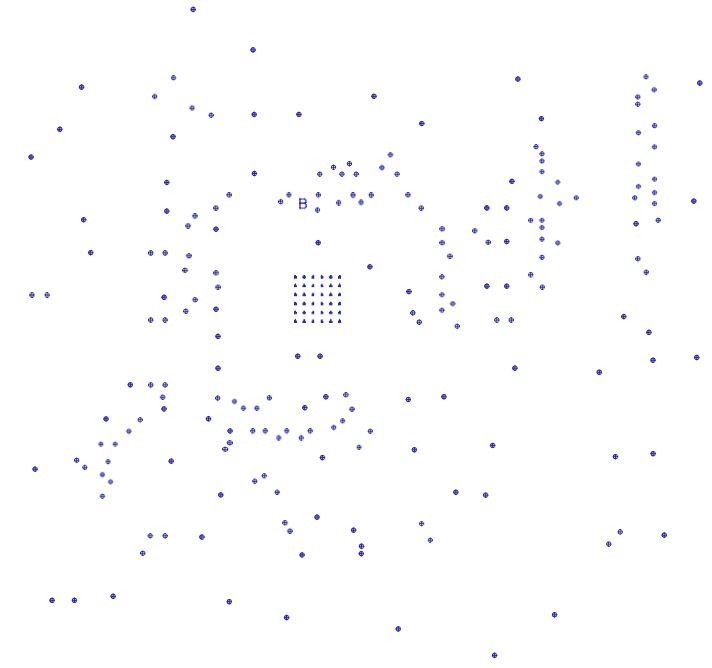
LATO TOP: processore, sensore, Sdram
LATO BOTTOM: //

I REVISIONE

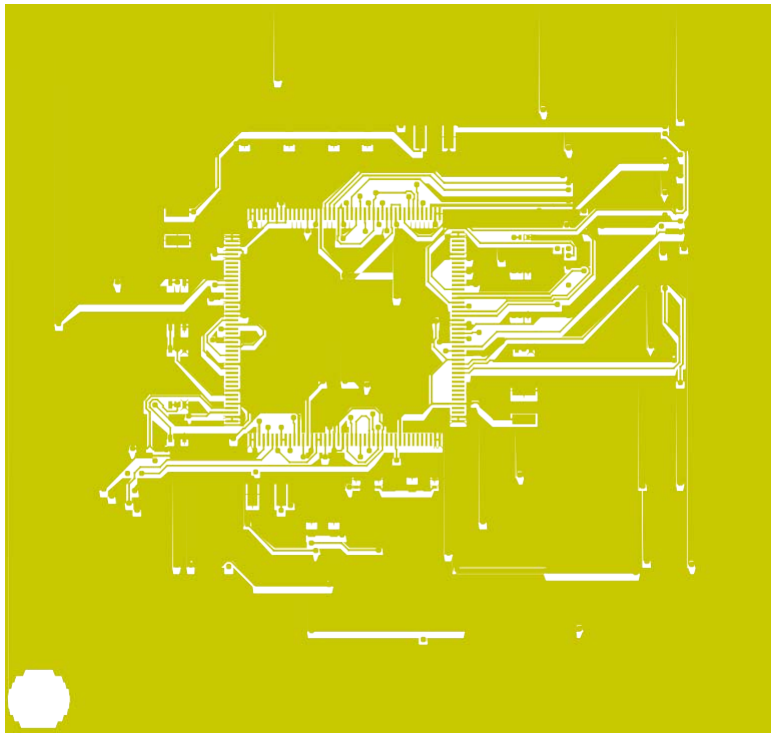


Through Holes						
Symbol	Diameter	Tolerance	Plated	Punched	HoleName	Quantity
A	0.0098		Yes	No	Rnd 0.25	36
B	0.0177		Yes	No	Rnd 0.45	1
⊕	0.0190	+0.0000 / -0.0030	Yes	No	Rnd 19 +Tol 0 -Tol -3	204
C	0.1496		No	No	Rnd 3.8 Non-Plated	1

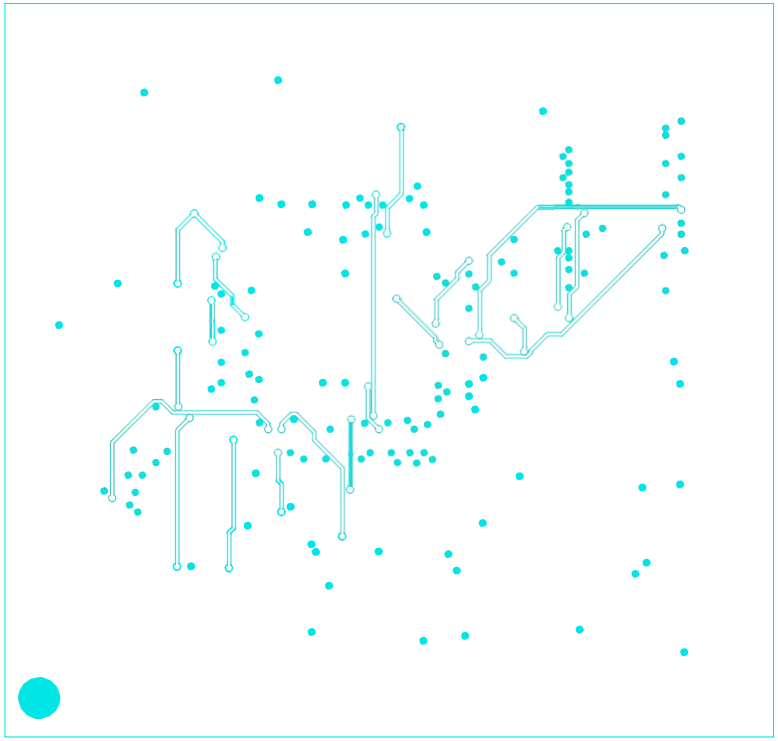
DrillDrawingThrough.gdo_1 EtchLayer1Top.gdo_1 EtchLayer2.gdo EtchLayer2Neg.gdo EtchLayer4Bottom.gdo
GeneratedSilkscreenBottom.gdo GeneratedSilkscreenTop.gdo SoldermaskBottom.gdo SoldermaskTop.gdo SolderPasteBottom.gdo
SolderPasteTop gdo EtchLayer1Top gdo DrillDrawingThrough gdo EtchLayer3 gdo



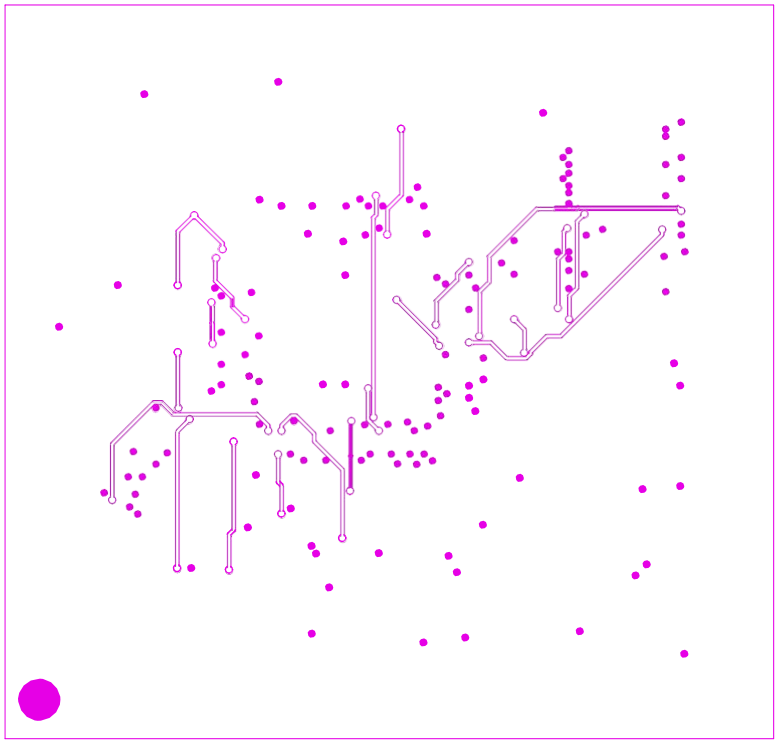
Through Holes						
Symbol	Diameter	Tolerance	Plated	Punched	HoleName	Quantity
A	0.0098		Yes	No	Rnd 0.25	36
B	0.0177		Yes	No	Rnd 0.45	1
⊕	0.0190	+0.0000 / -0.0030	Yes	No	Rnd 19 +Tol 0 -Tol -3	204
C	0.1496		No	No	Rnd 3.8 Non-Plated	1



EtchLayer1Top.gdo_1

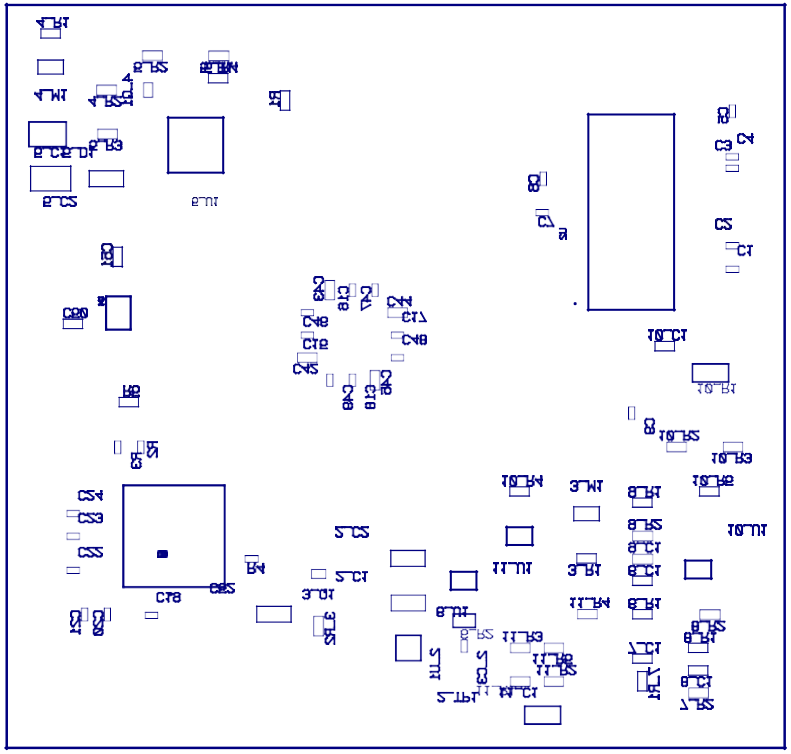


EtchLayer2.gdo

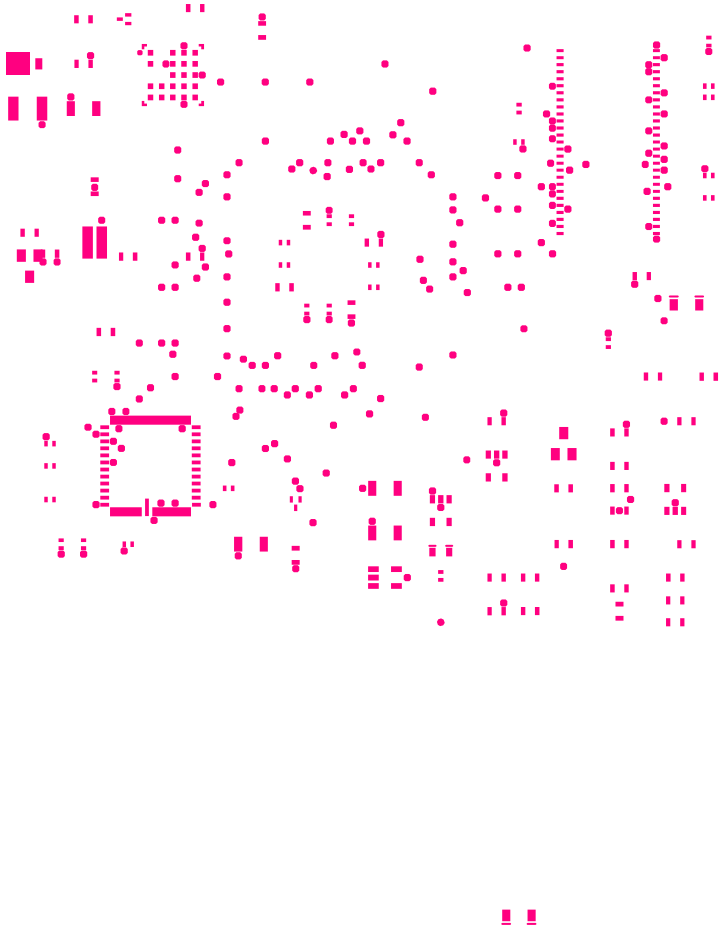


EtchLayer2Neg.gdo

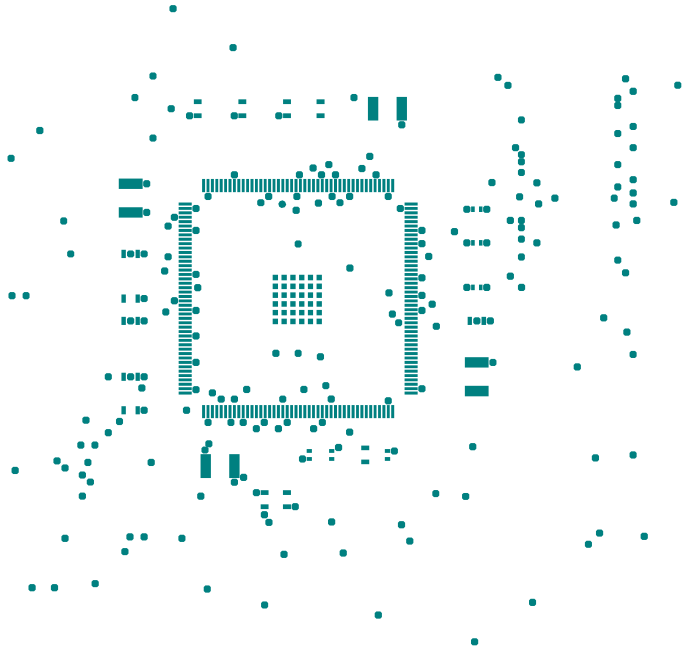
EtchLayer4Bottom.gdo



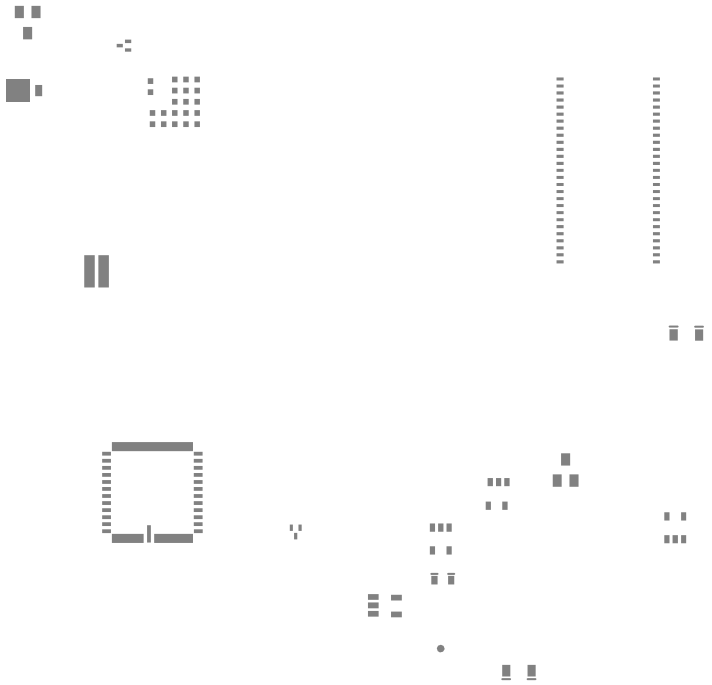
GeneratedSilkscreenBottom.gdo



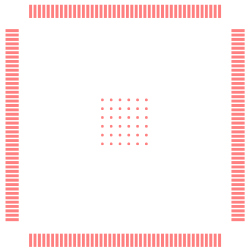
SoldermaskBottom.gdo



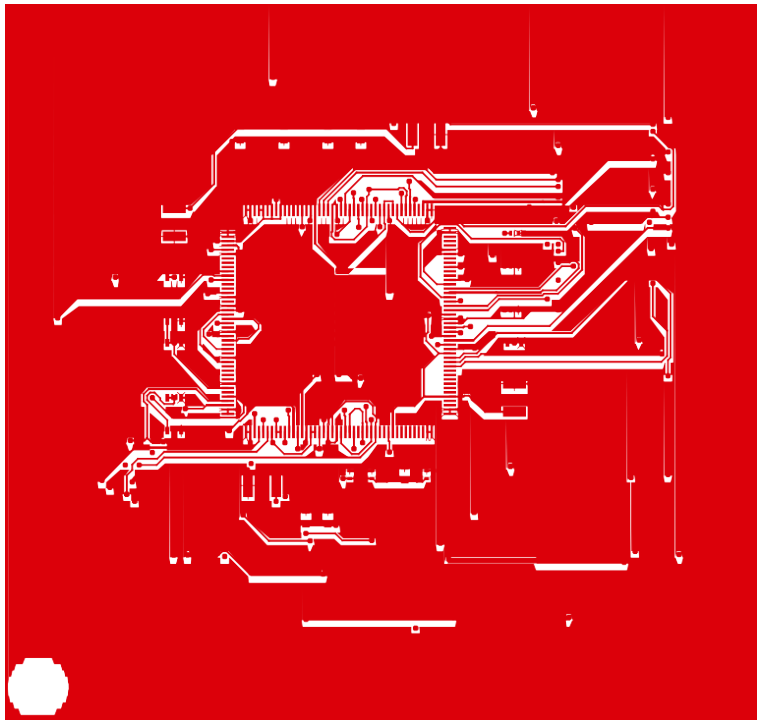
SoldermaskTop.gdo



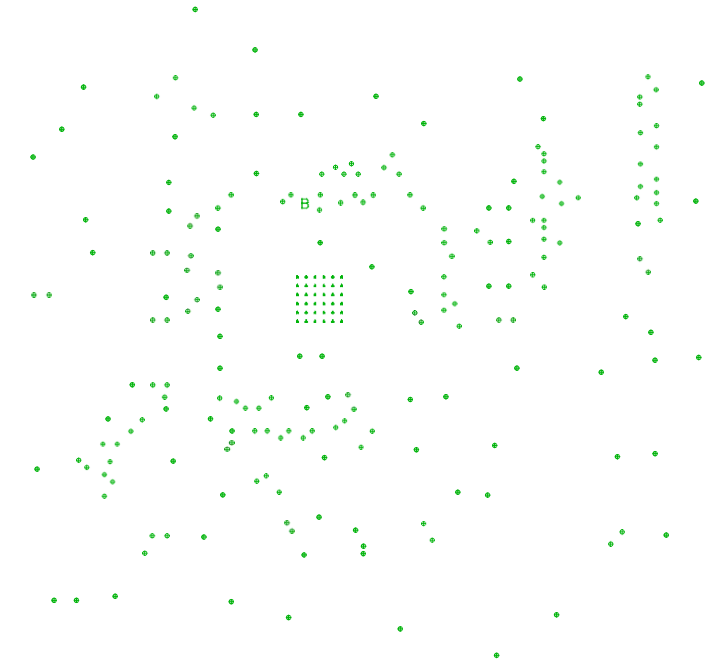
SolderPasteBottom.gdo



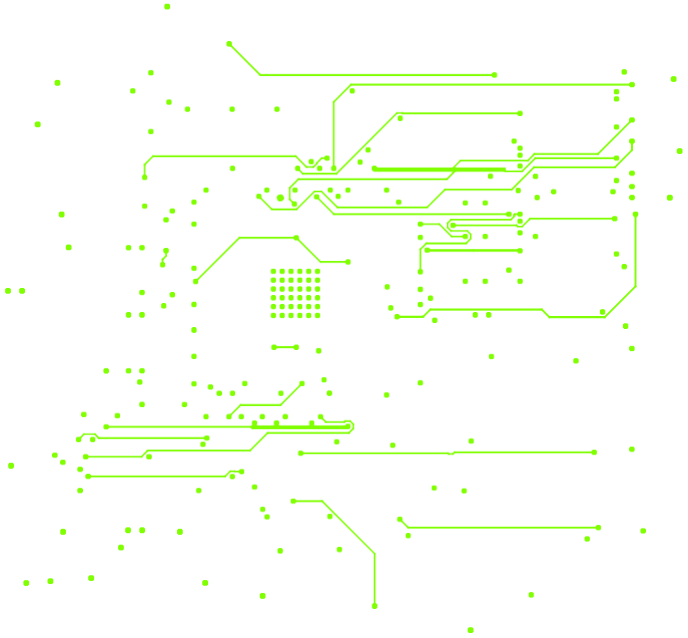
SolderPasteTop.gdo



EtchLayer1Top.gdo



Through Holes						
Symbol	Diameter	Tolerance	Plated	Punched	HoleName	Quantity
A	0.0098		Yes	No	Rnd 0.25	36
B	0.0177		Yes	No	Rnd 0.45	1
⊕	0.0190	+0.0000 / -0.0030	Yes	No	Rnd 19 +Tol 0 -Tol -3	204
C	0.1496		No	No	Rnd 3.8 Non-Plated	1



EtchLayer3.gdo

Appendice B

Schematici Elettrici – Mentor Graphics

Reusable Blocks:

1B231A_Sun_Horizon_Sensor

1B511A_Mono_WVGA_CMOS_Imager

1B4231_Image:Processor_BF518F16

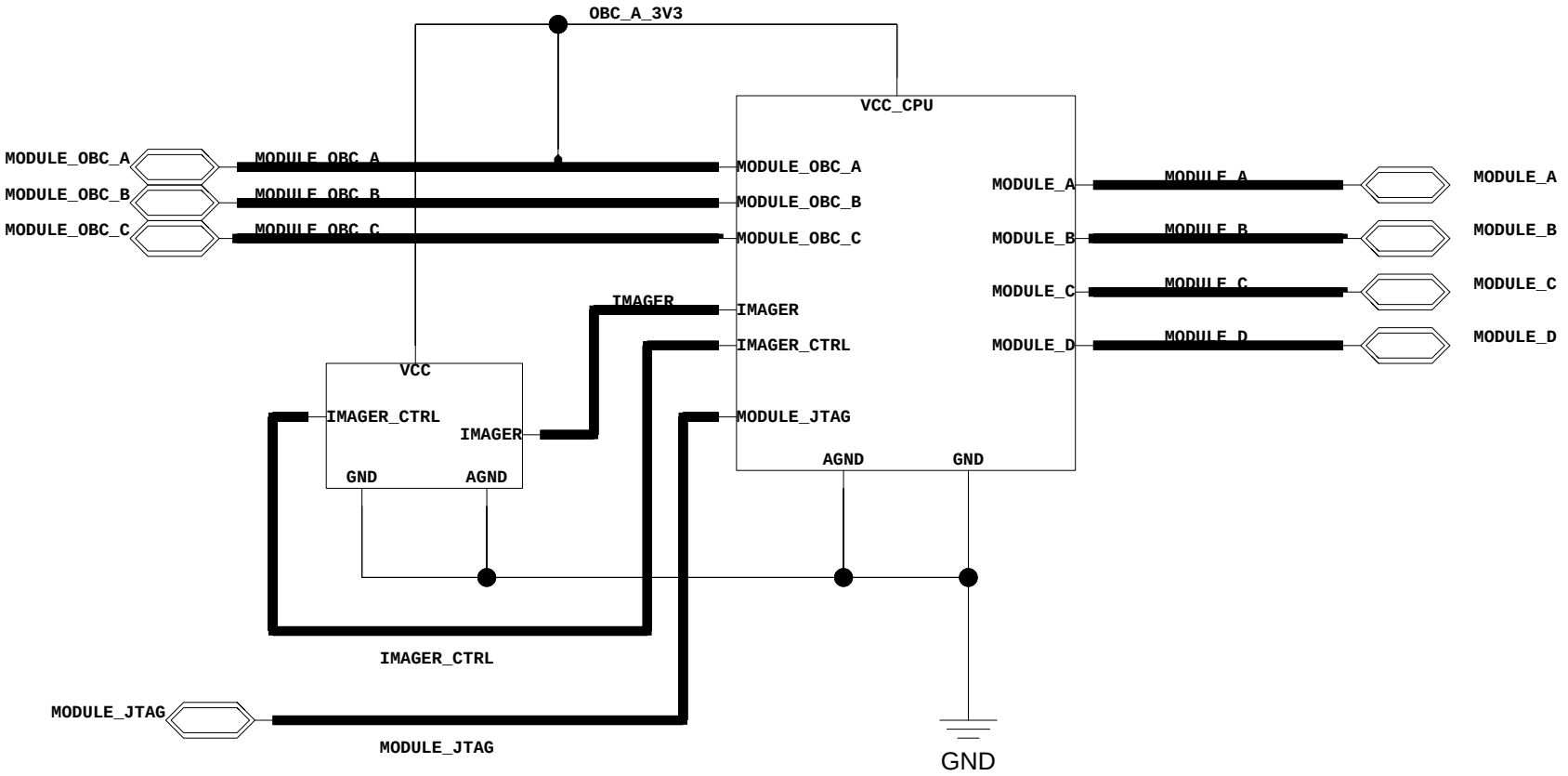
1B1255D_1V4_200mA_Switching_Regulator

1B132D_Current_Sensor

1B132E_Current_Sensor

1B131A_Voltage_Sensor

1B121C_Load_Switch



Mentor Graphics®

TITLE

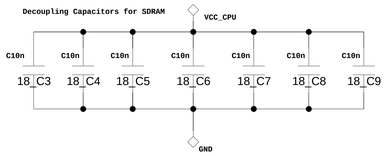
1B231A_SUN_HORIZON_SENSOR

SIZE	DWG NO	REV
A4		

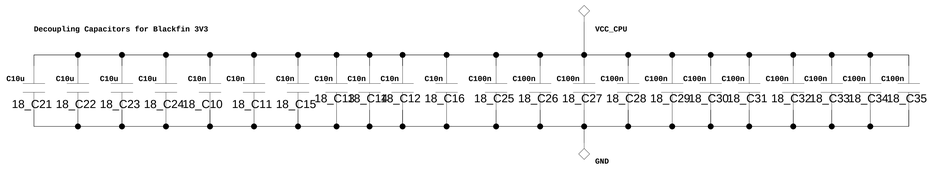
DRAWN BY	SCALE	SHEET	of	
		1	13	26/11/2014:11:51

ISSUE	REVISION	CLASS

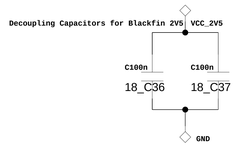
Decoupling Capacitors for SDRAM



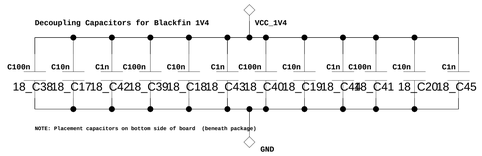
Decoupling Capacitors for Blackfin 3V3



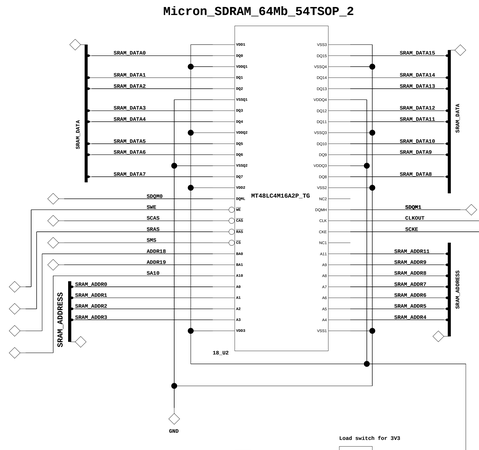
Decoupling Capacitors for Blackfin 2V5



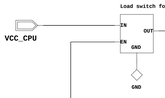
Decoupling Capacitors for Blackfin 1V4

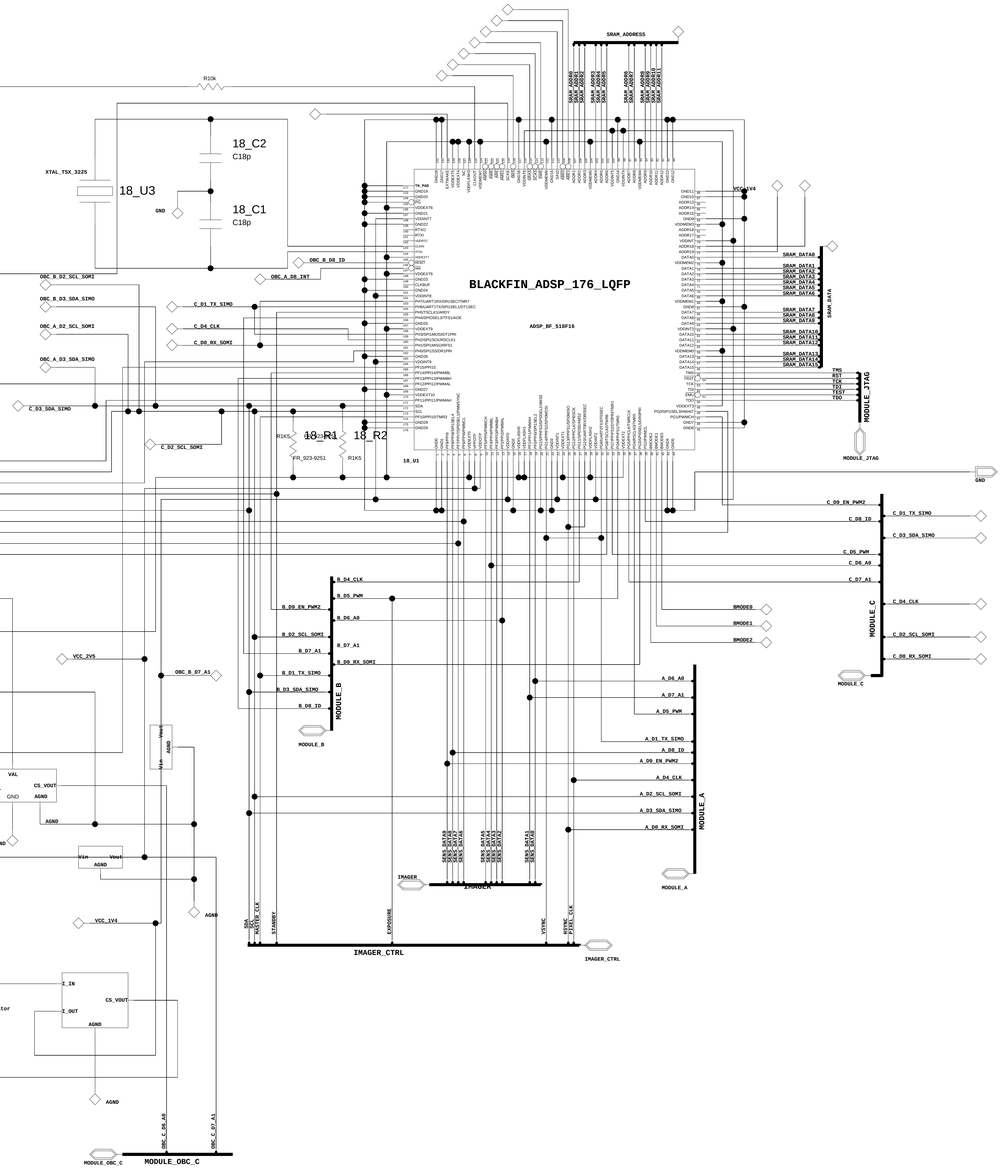


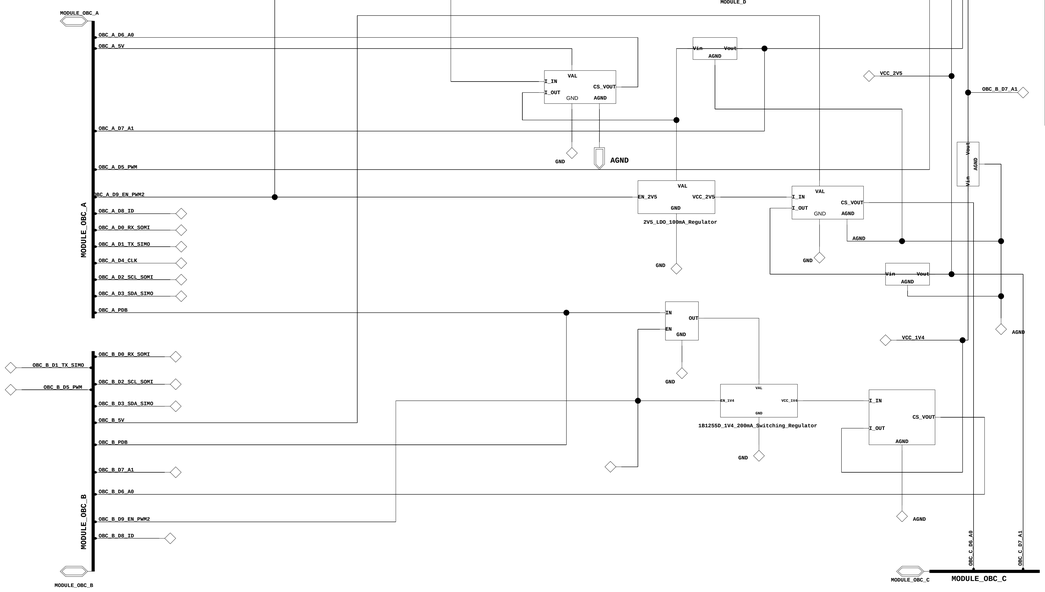
Micron_SDRAM_64Mb_54TSOP_2



Load switch for 3V3



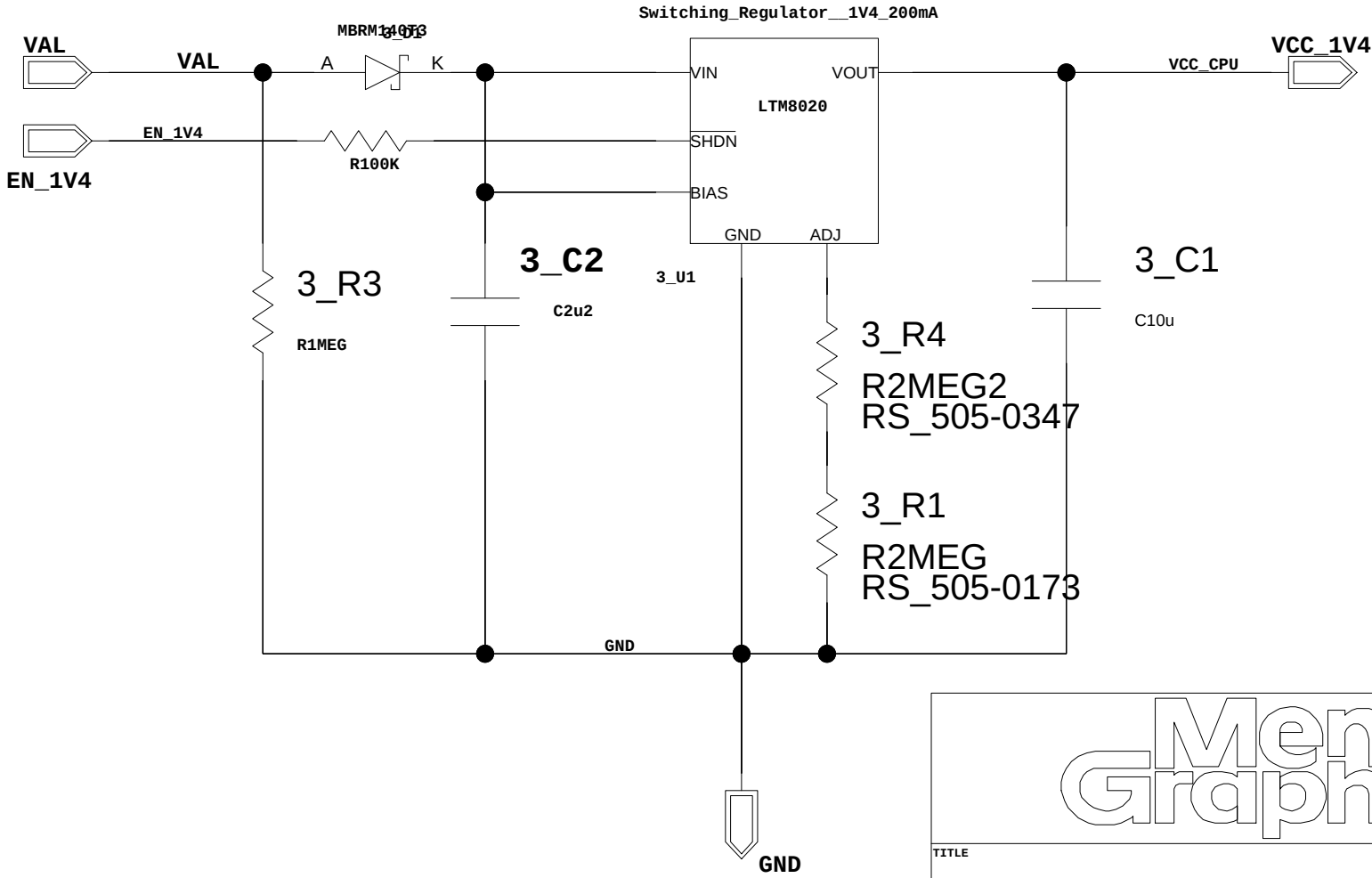




ISSUE			
DATE			
DRAWN	CHECKED	CERT'D	

TITLE	CIRCUIT DIAGRAM,
DRG No.	SHT OF
SECURITY CLASSIFICATION	

1B1255D_1V4_200mA_Switching_Regulator



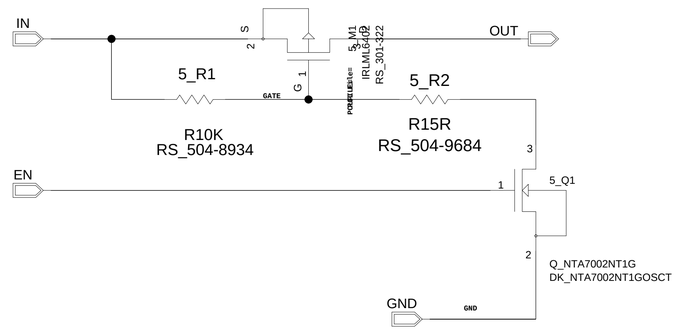
Mentor Graphics®

TITLE			
SIZE A4	DWG NO		REV
DRAWN BY	SCALE	SHEET 1 of 11	26/11/2014:11:50

DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED



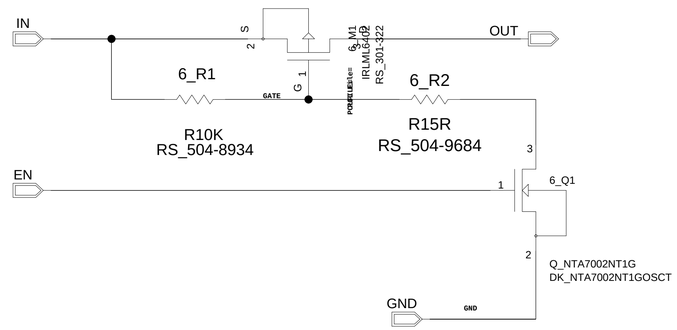
CONTRACT NO.		Schematic: 1B121C_Load_Switch_V1 Root: Bk1B11XAM_Partial_Power_Management	
APPROVALS	DATE		
DRAWN	26/11/2014:11:50	Project name: Bk1B11XAM_Partial_Power_Management Lib: R:\AraMiS_Mentor_Lib\AraMiS_Mentor_Lib.lmc	
	26/11/2014:11:50	SIZE D	FSCM NO. / DWG. NO. / REV
CHECKED			
ISSUED		SCALE	SHEET 01/01

H=8ND; V=4LA

DWG. NO.	SH.	REV

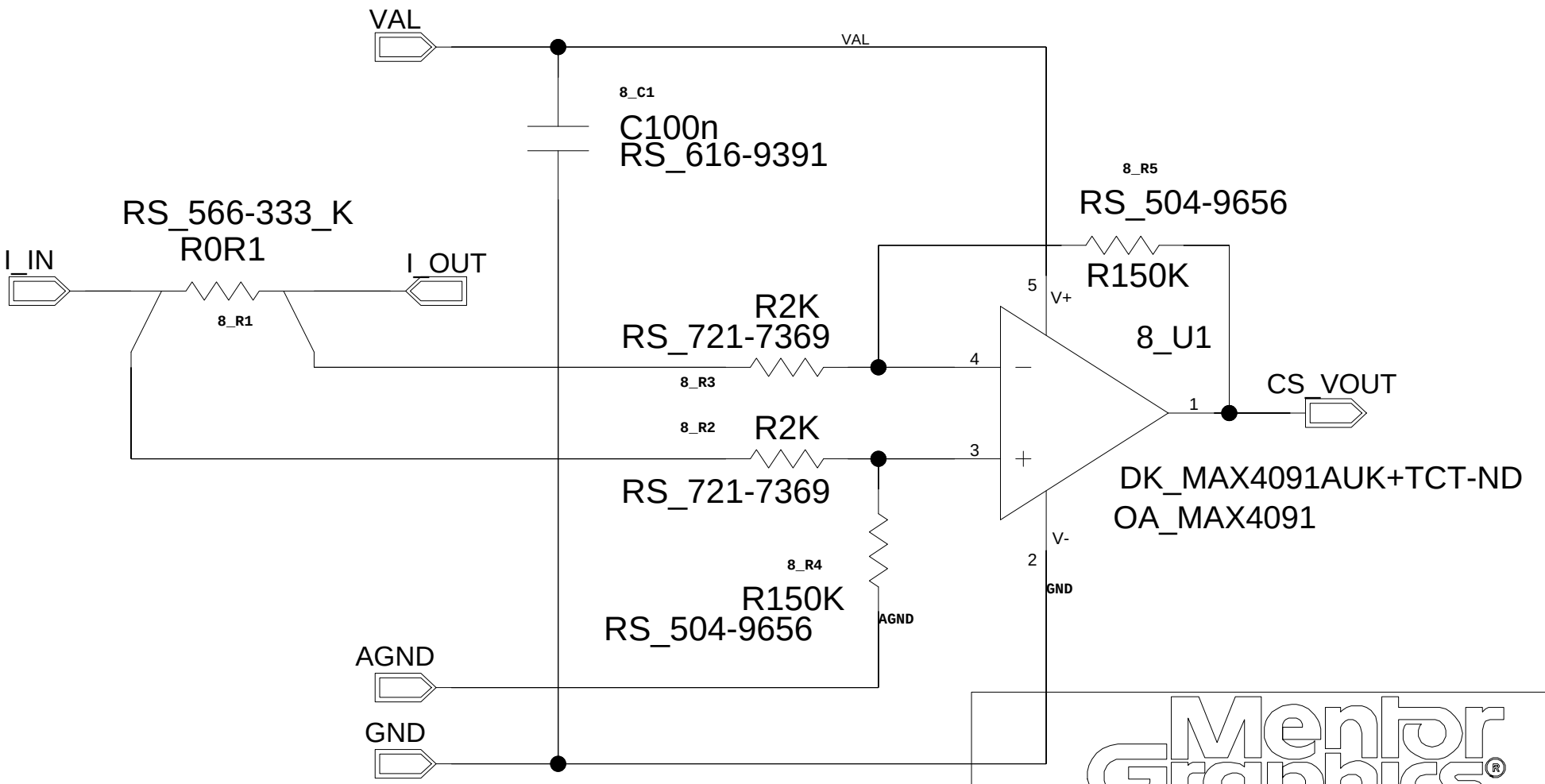
REVISIONS

REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		Schematic: 1B121C_Load_Switch_V1 Root: Bk1B11XAM_Partial_Power_Management	
APPROVALS	DATE		
DRAWN	26/11/2014:11:50	Project name: Bk1B11XAM_Partial_Power_Management Lib: R:\AraMiS_Mentor_Lib\AraMiS_Mentor_Lib.lmc	
	26/11/2014:11:50	SIZE D	FSCM NO. / DWG. NO. / REV
CHECKED			
ISSUED		SCALE	SHEET 01/01

H=8ND; V=4LA



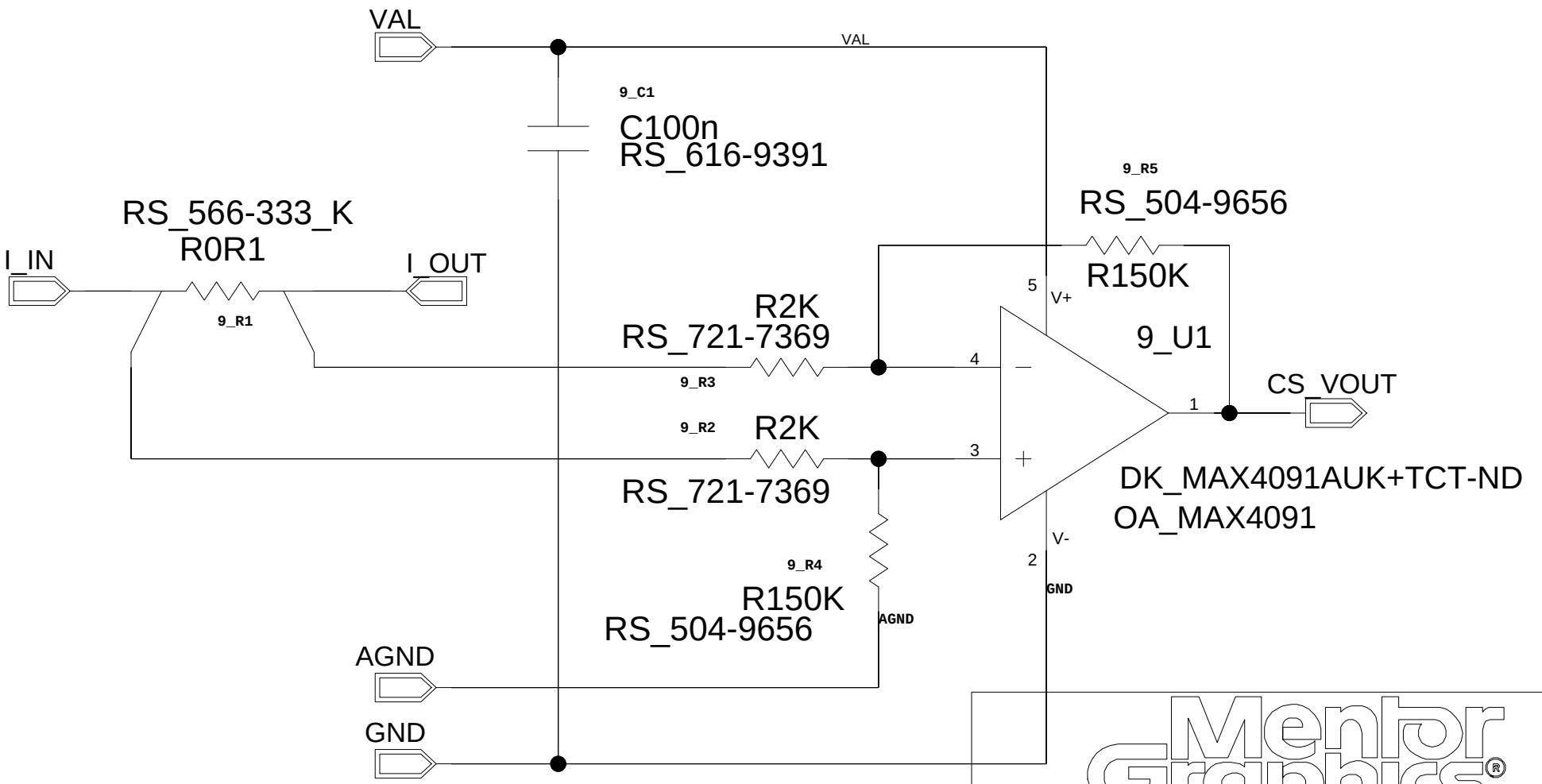
Mentor Graphics®

TITLE

1B132E_Current_Sensor_V1

SIZE	DWG NO	REV
A4		

DRAWN BY	SCALE	SHEET of	
		1 1	26/11/2014:11:50



Mentor Graphics®

TITLE

1B132E_Current_Sensor_V1

SIZE	DWG NO	REV
A4		

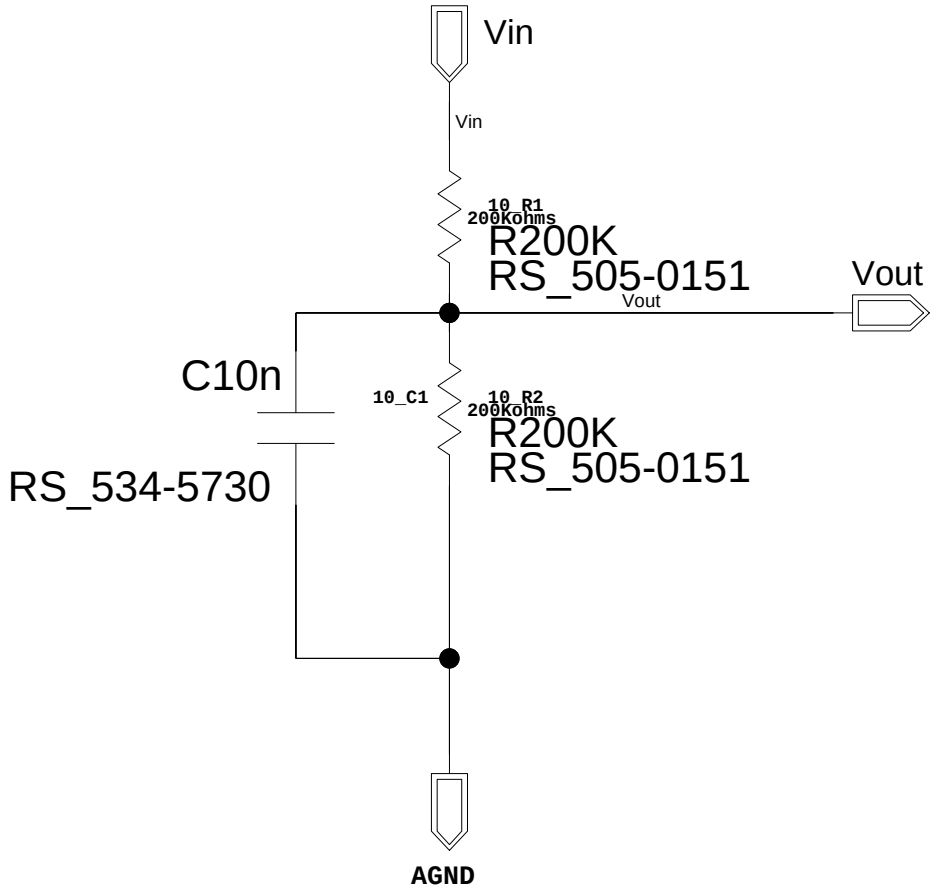
DRAWN BY	SCALE	SHEET of	
		1 1	26/11/2014:11:50

REV

SH.

DWG. NO.

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED



MAXIMUM INPUT VOLTAGE 5V

CONTRACT NO.		Schematic: 1B131A_Voltage_Sensor_V1 Root: Bk1B31B1M_Transceiver_2_4_GHz		
APPROVALS	DATE	Project name: Bk1B131B_2_4GHz_OBRF_Board Lib: R:\AraMiS_Mentor_Lib\AraMiS_Mentor_Lib.lmc		
DRAWN	26/11/2014:11:50			
	26/11/2014:11:50	SIZE	FSCM NO.	DWG. NO. / REV
CHECKED		AV		
ISSUED		SCALE		SHEET 01/01

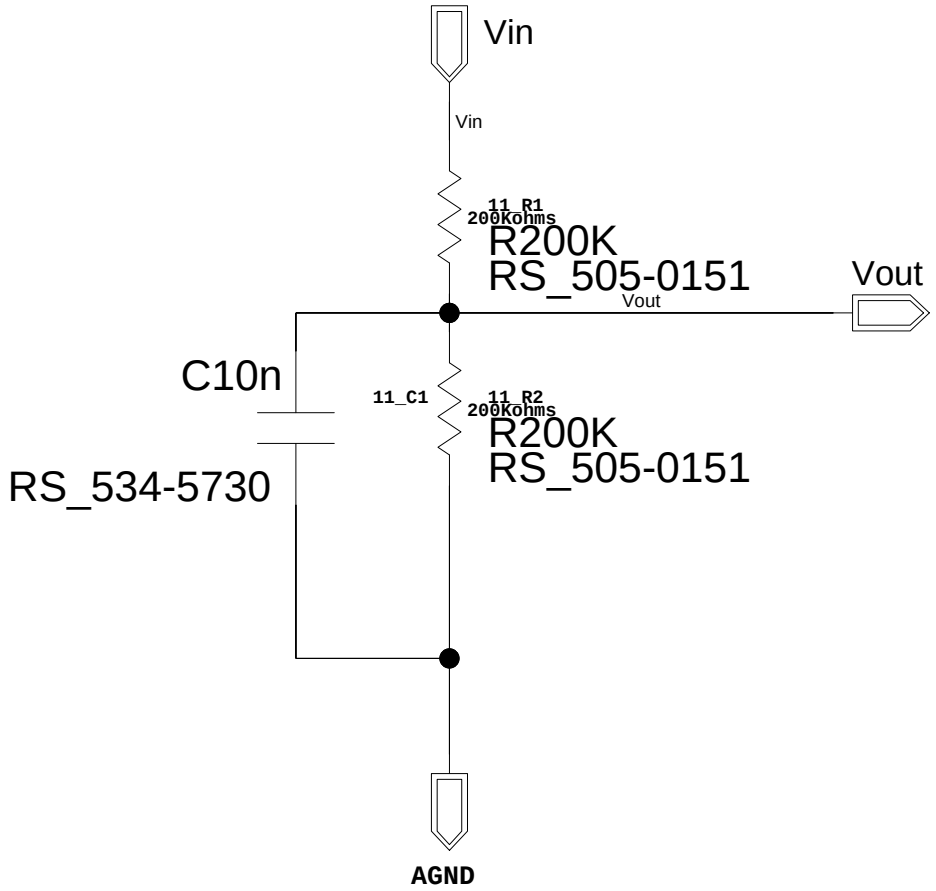
H=2LA; V=2LA

REV

SH.

DWG. NO.

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED



MAXIMUM INPUT VOLTAGE 5V

CONTRACT NO.		Schematic: 1B131A_Voltage_Sensor_V1 Root: Bk1B31B1M_Transceiver_2_4_GHz		
APPROVALS	DATE	Project name: Bk1B131B_2_4GHz_OBRF_Board Lib: R:\AraMiS_Mentor_Lib\AraMiS_Mentor_Lib.lmc		
DRAWN	26/11/2014:11:50			
	26/11/2014:11:50	SIZE	FSCM NO.	DWG. NO. / REV
CHECKED		AV		
ISSUED		SCALE		SHEET 01/01

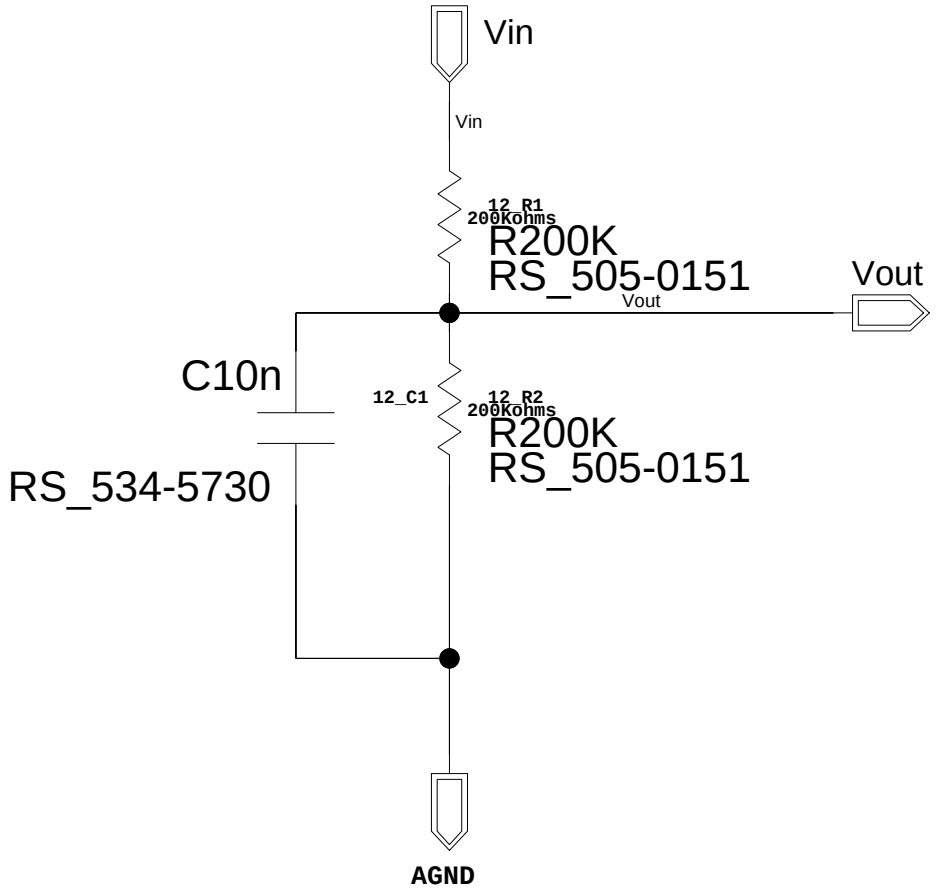
H=2LA; V=2LA

REV

SH.

DWG. NO.

REVISIONS			
REV	DESCRIPTION	DATE	APPROVED



MAXIMUM INPUT VOLTAGE 5V

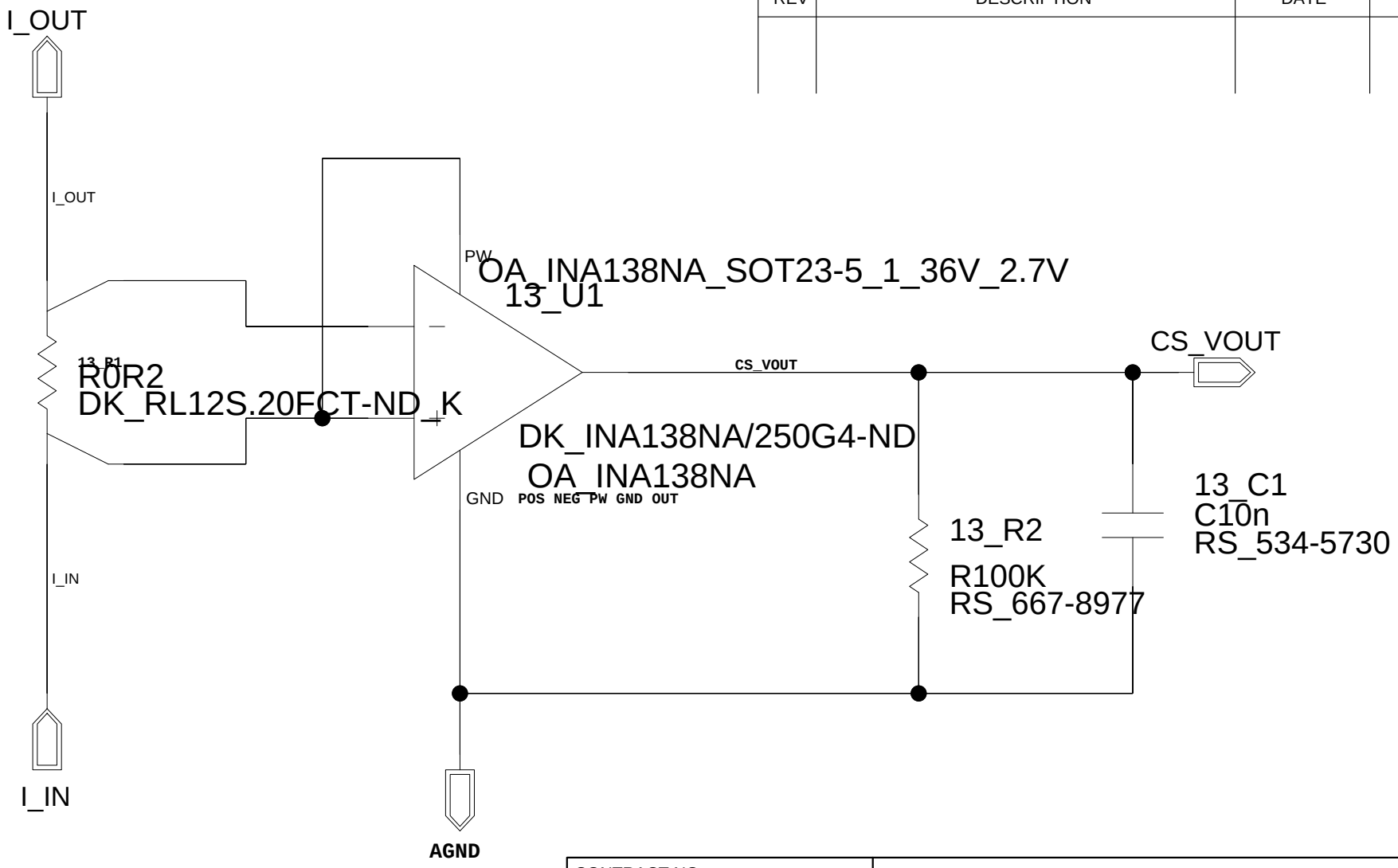
CONTRACT NO.		Schematic: 1B131A_Voltage_Sensor_V1 Root: Bk1B31B1M_Transceiver_2_4_GHz		
APPROVALS	DATE	Project name: Bk1B131B_2_4GHz_OBRF_Board Lib: R:\AraMiS_Mentor_Lib\AraMiS_Mentor_Lib.lmc		
DRAWN	26/11/2014:11:50			
	26/11/2014:11:50	SIZE	FSCM NO.	DWG. NO. / REV
CHECKED		AV		
ISSUED		SCALE		SHEET 01/01

H=2LA; V=2LA

DWG. NO.	SH.	REV

REVISIONS

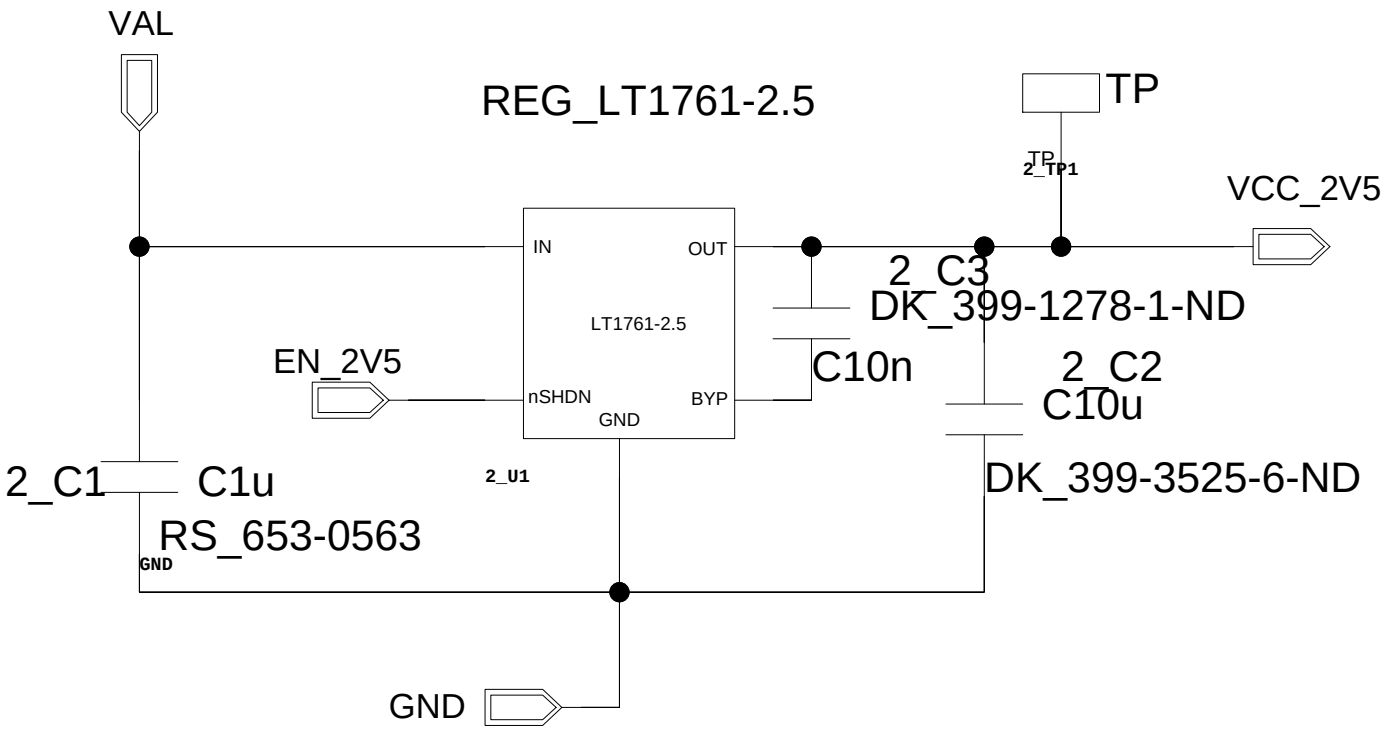
REV	DESCRIPTION	DATE	APPROVED



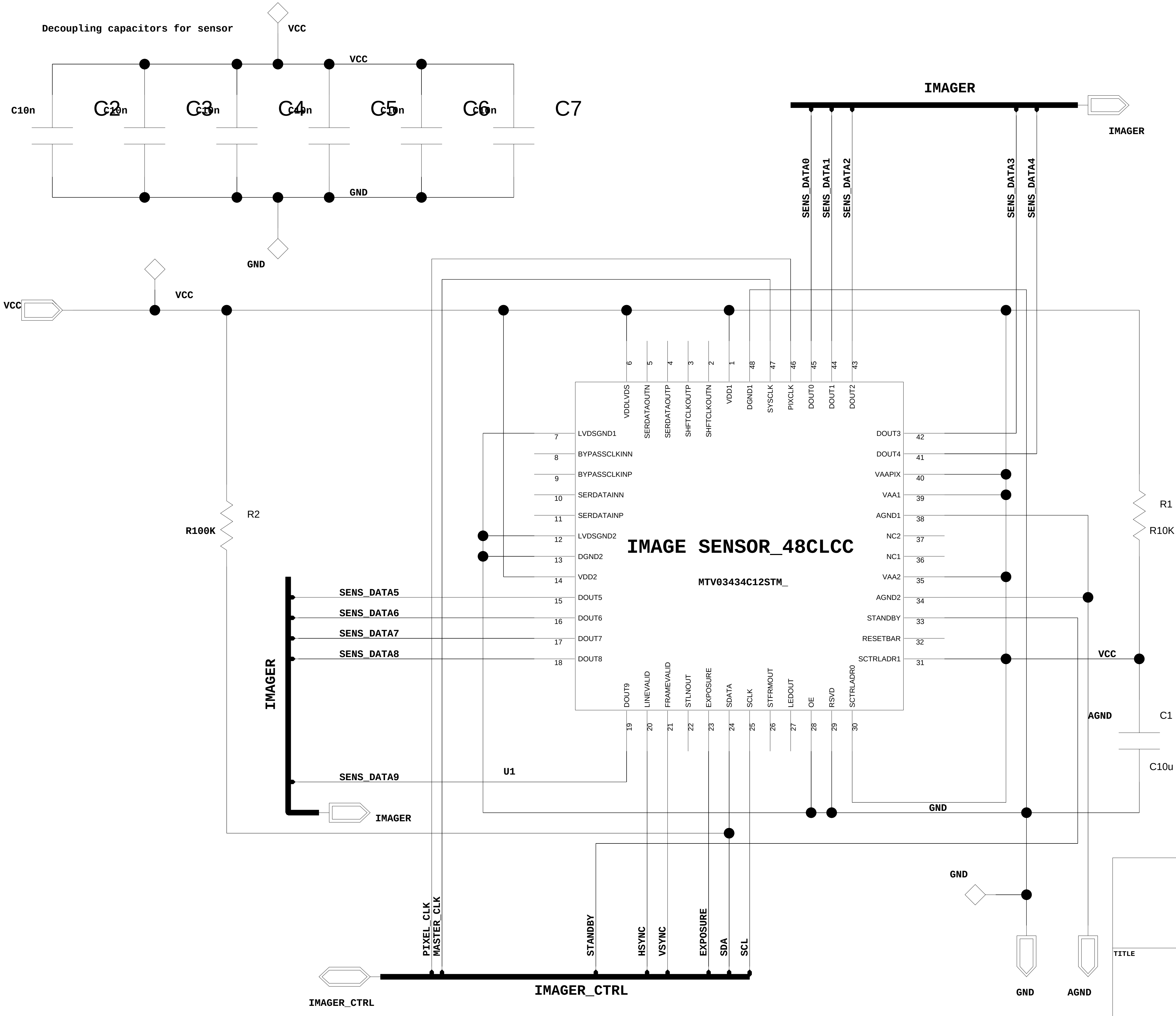
CONTRACT NO.		Schematic: 1B132A_Current_Sensor_V1 Root: 1B132A_Current_Sensor_V1	
APPROVALS	DATE	Project name: 1B132D_Current_Sensor_V1 Lib: R:\old_AraMiS_Mentor_Lib_2005\AraMiS_Mentor_Lib.lmc	
DRAWN	26/11/2014:11:50		
	26/11/2014:11:50	SIZE A / FSCM NO. / DWG. NO.	REV
CHECKED			
ISSUED		SCALE /	SHEET 01/01

H=2LA; V=2LA

1B1255C_2V5_100mA_LDO_Regulator_V1



TITLE			
SIZE A4	DWG NO		REV
DRAWN BY	SCALE	SHEET 1 of 11	26/11/2014:11:50



1B511A_Mono_WVGA_CMOS_Imager

TITLE

SIZE A3

DWG NO

REV 0

DRAWN BY <YOUR NAME HERE>

SCALE

SHEET 1 of 2

26/11/2014:09:20

Appendice C

Codice Sorgente

In questa sezione è riportato tutto il codice scritto in linguaggio C e assembler per l'applicazione richiesta dalle specifiche del sistema.

Il codice che segue è strutturato nelle seguenti sezioni:

1. Codice del Linker Description File dell'applicazione
2. Codice d'inizializzazione memoria SDRAM in assembler
3. Codice header file
4. Codice main.cpp
5. Driver Flash SPI per BOOT

```
/*
```

```
** ADSP-BF518 linker description file generated on Dec 02, 2014 at 15:02:07.
*/
/*
** Copyright (C) 2000-2014 Analog Devices Inc., All Rights Reserved.
**
** This file is generated automatically based upon the options selected
** in the System Configuration utility. Changes to the LDF configuration
** should be made by modifying the appropriate options rather than editing
** this file. To access the System Configuration utility, double-click the
** system.svc file from a navigation view.
**
** Custom additions can be inserted within the user-modifiable sections,
** these are bounded by comments that start with "$VDSG". Only changes
** placed within these sections are preserved when this file is re-generated.
**
** Product      : CrossCore Embedded Studio 1.1.0.0
** Tool Version : 6.0.3.2
*/

ARCHITECTURE(ADSP-BF518)

/*
** Define a linked objects list.
*/
$OBJECTS =
    "app_startup.doj"

/*$VDSG<insert-user-objects-at-beginning>                    */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-user-objects-at-beginning>                    */

   ,$COMMAND_LINE_OBJECTS
   ,"app_cplbtab.doj"
   ,crtn512.doj

/*$VDSG<insert-user-objects-at-end>                          */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-user-objects-at-end>                          */

   ;

/*
** Define a linked library list.
*/
$LIBRARIES =

/*$VDSG<insert-user-libraries-at-beginning>                  */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-user-libraries-at-beginning>                  */

    libcc.dlb
   ,libio.dlb
   ,libc.dlb
   ,librt_fileio.dlb
   ,libcpp.dlb
   ,libdsp.dlb
   ,libetsi.dlb
   ,libssl.dlb
   ,libdrv.dlb
```

```
   ,libdyn.dlb
   ,libsmall.dlb
   ,libevent.dlb
   ,libprofile.dlb
   ,libosal_noos.dlb

/*$VDSG<insert-user-libraries-at-end>                    */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-user-libraries-at-end>                    */

   ;

/*
** List of all objects and libraries.
*/
$OBJS_LIBS =

/*$VDSG<insert-objects-libraries-start>                  */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-objects-libraries-start>                  */

    $OBJECTS, $LIBRARIES

/*$VDSG<insert-objects-libraries-end>                    */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-objects-libraries-end>                    */

   ;

/*
** List of objects and libraries which prefer internal memory as
** specified by prefersMem attribute.
*/
$OBJS_LIBS_INTERNAL =

/*$VDSG<insert-libraries-internal>                       */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-libraries-internal>                       */

    $OBJS_LIBS{prefersMem("internal")}

/*$VDSG<insert-libraries-internal-end>                   */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-libraries-internal-end>                   */

   ;

/*
** List of objects and libraries which don't have a preference for
** external memory as specified by prefersMem attribute.
*/
$OBJS_LIBS_NOT_EXTERNAL =

/*$VDSG<insert-libraries-not-external>                   */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-libraries-not-external>                   */

    $OBJS_LIBS{!prefersMem("external")}
```

```
/*$VDSG<insert-libraries-not-external-end>                    */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-libraries-not-external-end>                    */

  ;

/*
** Determine which start label to use and its location
*/

/*$VDSG<insert-user-macros>                                   */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-user-macros>                                   */

#if !defined(START_ADDR)
#define START_ADDR  0xFFA00000 /* reset address */
#endif
#if !defined(START_SYM)
  #define START_SYM start
#endif


/*$VDSG<customize-async-macros>                               */
/* This code is preserved if the LDF is re-generated.          */

#define ASYNC0_MEMTYPE ROM
#define ASYNC1_MEMTYPE ROM
#define ASYNC2_MEMTYPE ROM
#define ASYNC3_MEMTYPE ROM

/*$VDSG<customize-async-macros>                               */


MEMORY
{
/*
** ADSP-BF518 MEMORY MAP.
**
** The known memory spaces are as follows:
**
** 0xFFE00000 - 0xFFFFFFFF  Core MMR registers (2MB)
** 0xFFC00000 - 0xFFDFFFFF  System MMR registers (2MB)
** 0xFFB01000 - 0xFFBFFFFF  Reserved
** 0xFFB00000 - 0xFFB00FFF  Scratch SRAM (4K)
** 0xFFA14000 - 0xFFAFFFFF  Reserved
** 0xFFA10000 - 0XFFA13FFF  Code SRAM/CACHE (16K)
** 0xFFA0C000 - 0xFFA0FFFF  Reserved
** 0xFFA08000 - 0xFFA0BFFF  Instruction Bank B SRAM (16K) in 0.0, Reserved on 0.1 and up
** 0xFFA04000 - 0xFFA07FFF  Reserved in 0.0, Instruction Bank B SRAM (16K) in 0.1 and up
** 0xFFA00000 - 0xFFA03FFF  Instruction Bank A SRAM (16K)
** 0xFF908000 - 0xFF9FFFFF  Reserved
** 0xFF904000 - 0xFF907FFF  Data Bank B SRAM/CACHE (16K)
** 0xFF900000 - 0XFF903FFF  Data Bank B SRAM (16K)
** 0xFF808000 - 0xFF8FFFFF  Reserved
** 0xFF804000 - 0xFF807FFF  Data Bank A SRAM/CACHE (16K)
** 0xFF800000 - 0XFF803FFF  Data Bank A SRAM (16K)
** 0xEF008000 - 0xFF7FFFFF  Reserved
** 0xEF000000 - 0xEF007FFF  Boot ROM (32K)
** 0x20400000 - 0xEEFFFFFF  Reserved
```

```
** 0x20300000 - 0x203FFFFF  ASYNC MEMORY BANK 3 (1MB)
** 0x20200000 - 0x202FFFFF  ASYNC MEMORY BANK 2 (1MB)
** 0x20100000 - 0x201FFFFF  ASYNC MEMORY BANK 1 (1MB)
** 0x20000000 - 0x200FFFFF  ASYNC MEMORY BANK 0 (1MB)
** 0x00000000 - 0x07FFFFFF  SDRAM MEMORY (16MB - 128MB)
**
** Notes:
** 0xFF807FEF-0xFF807FFF
**   Required by boot-loader. Used as heap or cache below which is ok. Cannot
**   contain initialized data or code.
*/

  MEM_L1_SCRATCH         { TYPE(RAM) START(0xFFB00000) END(0xFFB00FFF) WIDTH(8) }
  MEM_L1_CODE_CACHE      { TYPE(RAM) START(0xFFA10000) END(0xFFA13FFF) WIDTH(8) }
  MEM_L1_CODE            { TYPE(RAM) START(0xFFA00000) END(0xFFA07FFF) WIDTH(8) }
  MEM_L1_DATA_B          { TYPE(RAM) START(0xFF900000) END(0xFF907FFF) WIDTH(8) }
  MEM_L1_DATA_A          { TYPE(RAM) START(0xFF800000) END(0xFF807FFF) WIDTH(8) }
  MEM_ASYNC3             { TYPE(ASYNC3_MEMTYPE) START(0x20300000) END(0x203FFFFF) WIDTH(8) }
  MEM_ASYNC2             { TYPE(ASYNC2_MEMTYPE) START(0x20200000) END(0x202FFFFF) WIDTH(8) }
  MEM_ASYNC1             { TYPE(ASYNC1_MEMTYPE) START(0x20100000) END(0x201FFFFF) WIDTH(8) }
  MEM_ASYNC0             { TYPE(ASYNC0_MEMTYPE) START(0x20000000) END(0x200FFFFF) WIDTH(8) }

  /*$VDSG<insert-new-memory-segments>                    */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-new-memory-segments>                    */

} /* MEMORY */

PROCESSOR p0
{
  OUTPUT($COMMAND_LINE_OUTPUT_FILE)
  RESOLVE(START_SYM, START_ADDR)
  KEEP(START_SYM, _main)

  /*$VDSG<insert-user-ldf-commands>                      */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-user-ldf-commands>                      */

  SECTIONS
  {
    /* Workaround for hardware errata 05-00-0189 and 05-00-0310 -
    ** "Speculative (and fetches made at boundary of reserved memory
    ** space) for instruction or data fetches may cause false
    ** protection exceptions" and "False hardware errors caused by
    ** fetches at the boundary of reserved memory ".
    **
    ** Done by avoiding use of 76 bytes from at the end of blocks
    ** that are adjacent to reserved memory. Workaround is enabled
    ** for appropriate silicon revisions (-si-revision switch).
    */
    RESERVE(___wab0=MEMORY_END(MEM_L1_SCRATCH) - 75, ___l0 = 76)
    RESERVE(___wab2=MEMORY_END(MEM_L1_CODE) - 75, ___l2 = 76)
    RESERVE(___wab3=MEMORY_END(MEM_L1_DATA_B) - 75, ___l3 = 76)
    RESERVE(___wab4=MEMORY_END(MEM_L1_DATA_A) - 75, ___l4 = 76)
    RESERVE(___wab5=MEMORY_END(MEM_ASYNC3) - 75, ___l5 = 76)

    /*$VDSG<insert-new-sections-at-the-start>              */
    /* Text inserted between these $VDSG comments will be preserved */
    /*$VDSG<insert-new-sections-at-the-start>              */
```

```
#define DEF_SECTION_QUAL /* None, rely on load-time initialization. */
scratchpad NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-scratchpad>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-scratchpad>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_scratchpad))

  /*$VDSG<insert-input-sections-at-the-end-of-scratchpad>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-scratchpad>  */

} > MEM_L1_SCRATCH

scratchpad_stack_heap NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-scratchpad_stack_heap>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-scratchpad_stack_heap>  */

} > MEM_L1_SCRATCH


/*$VDSG<insert-new-sections-after-SCRATCH>              */
/* Text inserted between these $VDSG comments will be preserved */
/*$VDSG<insert-new-sections-after-SCRATCH>              */

L1_code DEF_SECTION_QUAL
{
  INPUT_SECTION_ALIGN(4)
  INPUT_SECTIONS($OBJS_LIBS(L1_code cplb_code cplb noncache_code))

  /*$VDSG<insert-input-sections-at-the-start-of-l1_code>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-l1_code>  */

  INPUT_SECTIONS($OBJS_LIBS_INTERNAL(program))
  INPUT_SECTIONS($OBJS_LIBS_NOT_EXTERNAL(program))
  INPUT_SECTIONS($OBJS_LIBS(program))

  /*$VDSG<insert-input-sections-at-the-end-of-l1_code>   */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-l1_code>   */

} > MEM_L1_CODE

L1_code_cache NO_INIT
{
  INPUT_SECTION_ALIGN(4)
  ___l1_code_cache = 1; /* Instruction cache is enabled. */
} > MEM_L1_CODE_CACHE

L1_data_a_prio0 DEF_SECTION_QUAL
```

```
{
  INPUT_SECTION_ALIGN(4)
  ___l1_data_cache_a = 0; /* DATA A cache is not enabled. */
  RESERVE(heaps_and_stack_in_L1_data_a, heaps_and_stack_in_L1_data_a_length = 2048, 4)
  EXECUTABLE_NAME(__executable_name)
  INPUT_SECTIONS($OBJS_LIBS(L1_data_a))

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_prio0>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_data))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_prio0>  */

} > MEM_L1_DATA_A

L1_data_a_bsz_prio0 ZERO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_bsz_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_bsz_prio0>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_bsz_a L1_bsz))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_bsz_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_bsz_prio0>  */

} > MEM_L1_DATA_A

L1_data_a_no_init_prio0 NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_no_init_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_no_init_prio0>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_noinit_data_a L1_noinit_data))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_no_init_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_no_init_prio0>  */

} > MEM_L1_DATA_A

L1_data_a_prio3 DEF_SECTION_QUAL
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_prio3>  */
```

```
  INPUT_SECTIONS($OBJS_LIBS(data1 voldata cplb_data constdata))
  INPUT_SECTIONS($OBJS_LIBS(vtbl .edt .cht .rtti))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_prio3>  */

} > MEM_L1_DATA_A

L1_data_a_bsz_prio3 ZERO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_bsz_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_bsz_prio3>  */

  INPUT_SECTIONS($OBJS_LIBS(bsz))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_bsz_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_bsz_prio3>  */

} > MEM_L1_DATA_A

L1_data_a_no_init_prio3 NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_no_init_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_a_no_init_prio3>  */

  INPUT_SECTIONS($OBJS_LIBS(noinit_data))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_no_init_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_a_no_init_prio3>  */

} > MEM_L1_DATA_A

stack_and_heap_L1_data_a NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-stack_and_heap_L1_data_a>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-stack_and_heap_L1_data_a>  */

  RESERVE_EXPAND(heaps_and_stack_in_L1_data_a, heaps_and_stack_in_L1_data_a_length, 0, 4)
  ldf_stack_space = heaps_and_stack_in_L1_data_a;
  ldf_stack_end = (ldf_stack_space + (heaps_and_stack_in_L1_data_a_length - 4)) & 0xfffffffc;
} > MEM_L1_DATA_A

L1_data_b_prio0 DEF_SECTION_QUAL
{
  INPUT_SECTION_ALIGN(4)
  ___l1_data_cache_b = 0; /* DATA B cache is not enabled. */
  RESERVE(heaps_and_stack_in_L1_data_b, heaps_and_stack_in_L1_data_b_length = 2048, 4)
```

```
  INPUT_SECTIONS($OBJS_LIBS(L1_data_b))

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_prio0>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_data))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_prio0>  */

} > MEM_L1_DATA_B

L1_data_b_prio0_tables DEF_SECTION_QUAL
{
  INPUT_SECTION_ALIGN(4)
  FORCE_CONTIGUITY

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_prio0_tables>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_prio0_tables>  */

  INPUT_SECTIONS($OBJS_LIBS(ctor))  /* global C++ constructors list */
  INPUT_SECTIONS($OBJS_LIBS(ctorl)) /* NULL terminator for ctor     */
  INPUT_SECTIONS($OBJS_LIBS(.gdt))  /* C++ exceptions data          */
  INPUT_SECTIONS($OBJS_LIBS(.gdtl)) /* NULL terminator for .gdt     */

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_prio0_tables>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_prio0_tables>  */

} > MEM_L1_DATA_B

L1_data_b_bsz_prio0 ZERO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_bsz_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_bsz_prio0>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_bsz_b L1_bsz))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_bsz_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_bsz_prio0>  */

} > MEM_L1_DATA_B

L1_data_b_no_init_prio0 NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_no_init_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_no_init_prio0>  */

  INPUT_SECTIONS($OBJS_LIBS(L1_noinit_data_b L1_noinit_data))
```

```
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_no_init_prio0>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_no_init_prio0>  */

} > MEM_L1_DATA_B

L1_data_b_prio3 DEF_SECTION_QUAL
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_prio3>  */

  INPUT_SECTIONS($OBJS_LIBS(data1 voldata cplb_data constdata))
  INPUT_SECTIONS($OBJS_LIBS(vtbl .edt .cht .rtti))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_prio3>  */

} > MEM_L1_DATA_B

L1_data_b_bsz_prio3 ZERO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_bsz_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_bsz_prio3>  */

  INPUT_SECTIONS($OBJS_LIBS(bsz))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_bsz_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_bsz_prio3>  */

} > MEM_L1_DATA_B

L1_data_b_no_init_prio3 NO_INIT
{
  INPUT_SECTION_ALIGN(4)

  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_no_init_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-start-of-L1_data_b_no_init_prio3>  */

  INPUT_SECTIONS($OBJS_LIBS(noinit_data))

  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_no_init_prio3>  */
  /* Text inserted between these $VDSG comments will be preserved */
  /*$VDSG<insert-input-sections-at-the-end-of-L1_data_b_no_init_prio3>  */

} > MEM_L1_DATA_B

stack_and_heap_L1_data_b NO_INIT
{
  INPUT_SECTION_ALIGN(4)
```

```
        /*$VDSG<insert-input-sections-at-the-start-of-stack_and_heap_L1_data_b>  */
        /* Text inserted between these $VDSG comments will be preserved */
        /*$VDSG<insert-input-sections-at-the-start-of-stack_and_heap_L1_data_b>  */

        RESERVE_EXPAND(heaps_and_stack_in_L1_data_b, heaps_and_stack_in_L1_data_b_length, 0, 4)
        ldf_heap_space = heaps_and_stack_in_L1_data_b;
        ldf_heap_end = (ldf_heap_space + (heaps_and_stack_in_L1_data_b_length - 4)) & 0xfffffffc;
        ldf_heap_length = ldf_heap_end - ldf_heap_space;
      } > MEM_L1_DATA_B


      /*$VDSG<insert-new-sections-at-the-end>                */
      /* Text inserted between these $VDSG comments will be preserved */
      /*$VDSG<insert-new-sections-at-the-end>                */

    } /* SECTIONS */
} /* p0 */



/*****************************************************************************/
/*  This file contains 3 sections:                                           */
/*          1) A Pre-Init Section - this section saves off all the           */
/*                  registers of the DSP.                                    */
/*          2) A Init Code Section - this section is the customer            */
/*                  initialization code which can be modified by the customer.*/
/*                  As an example, an SDRAM initialization code is supplied. */
/*          3) A Post-Init Section - this section restores all the           */
/*                  register from the stack.                                 */
/*          Customers should not modify the Pre-Init and Post-Init Sections.*/
/*          The Init Code Section can be modified for application use.       */
/*****************************************************************************/

#include <defBF532.h>

.section program;

/********Pre-Init Section*************************************************/
            [--SP] = ASTAT;                     //Save Regs onto stack
            [--SP] = RETS;
            [--SP] = (r7:0);
            [--SP] = (p5:0);
            [--SP] = I0;
            [--SP] = I1;
            [--SP] = I2;
            [--SP] = I3;
            [--SP] = B0;
            [--SP] = B1;
            [--SP] = B2;
            [--SP] = B3;
            [--SP] = M0;
            [--SP] = M1;
            [--SP] = M2;
            [--SP] = M3;
            [--SP] = L0;
            [--SP] = L1;
            [--SP] = L2;
```

```
        [--SP] = L3;
/*****************************************************************************/
/*******Init Code Section*****************************************************/
/*******SDRAM Setup************/
Setup_SDRAM:
        P0.L = EBIU_SDRRC & 0xFFFF;
        P0.H = (EBIU_SDRRC >> 16) & 0xFFFF;          //SDRAM Refresh Rate Control Register
        R0 = 0x060E(Z);
        W[P0] = R0;
        SSYNC;

        P0.L = EBIU_SDBCTL & 0xFFFF;
        P0.H = (EBIU_SDBCTL >> 16) & 0xFFFF;        //SDRAM Memory Bank Control Register
        R0 = 0x0001(Z);
        [P0] = R0;
        SSYNC;

        P0.L = EBIU_SDGCTL & 0xFFFF;
        P0.H = (EBIU_SDGCTL >> 16) & 0xFFFF;        //SDRAM Memory Global Control Register
        R0.L = 0x1109;
        R0.H = 0x0091;
        [P0] = R0;
        SSYNC;
/*******************************/
/*****************************************************************************/
/*******Post-Init Section*****************************************************/
        L3 = [SP++];
        L2 = [SP++];
        L1 = [SP++];
        L0 = [SP++];
        M3 = [SP++];
        M2 = [SP++];
        M1 = [SP++];
        M0 = [SP++];
        B3 = [SP++];
        B2 = [SP++];
        B1 = [SP++];
        B0 = [SP++];
        I3 = [SP++];
        I2 = [SP++];
        I1 = [SP++];
        I0 = [SP++];
        (p5:0) = [SP++];                    //Restore Regs from Stack
        (r7:0) = [SP++];
        RETS = [SP++];
        ASTAT = [SP++];
/*****************************************************************************/
END:    RTS;
/*****************************************************************************/

/*****************************************************************************
 * get_frame.h
 *****************************************************************************/

#ifndef __GET_FRAME_H__
#define __GET_FRAME_H__

/* Add your custom header content here */
```

```
/*****************************************************************************
 * get_raw_image.h
 *****************************************************************************/

#ifndef __GET_RAW_IMAGE_H__
#define __GET_RAW_IMAGE_H__

/* Add your custom header content here */


#define CFG_GP_INPUT_2SYNCS 0x0020
#define GP_INPUT_MODE 0x000C
#define POL_S  0x0000
#define POL_C  0x0000
#define DATALEN  0x0800
#define PORT_EN  0x0001
#define PPICOUNT  0x02EF
#define TMR0 0x01
#define TMR3 0x08
#define PULSE_WIDTH0 0x6C6FA
#define PULSE_WIDTH3  4
#define PULSE_PERIOD3  0x6C728
#define PRESCALE_S 0x000A
#define DCNT_S  0x00C0
#define FAST  0x0008
#define MEN  0x0001
#define XMTSERV  0x0040
#define REGADDR  0x07
#define SNAPMODE  0x0398
#define PIXELS_PER_LINE 752
#define LINES_PER_FRAME  480
#define STOPMODE  0x0000
#define RESTART  0x0000
#define DI_EN  0x0080
#define DMA2D  0x0010
#define WDSIZE_16  0x0004
#define WNR  0x0002
#define DMAEN  0x0001
#endif /* __GET_RAW_IMAGE_H__ */
#endif /* __GET_FRAME_H__ */

/*****************************************************************************
 * get_frame.cpp
 *****************************************************************************/

#include "adi_initialize.h"
#include "get_frame.h"
#include "adi_initialize.h"
#include <cdefBF51x_base.h>
#include <ccblkfn.h>
#include <stdio.h>
#include <sys\exception.h>
#include <blackfin.h>
#include <bfrom.h>


/**
```

```
 * Place program arguments in the following string.
 */
//extern char __argv_string[] = "prog_name -in x.gif -out y.jpeg";


   void Init_PLL(void);
        void Init_ProgrammableFlags(void);
        void VideoSensorFrameCapture ( unsigned short *usPTR );
        void InitSDRAM(void);
        void Init_TWI(void);
        void Write_TWI8(REGADDR);
        void Write_TWI16(SNAPMODE);
        void Truncating(unsigned short VideoInputFrame);
        /*************************************************************/
        /****************** variable definition ***********************/
        /*************************************************************/
        section ("sdram_data")
        volatile unsigned short VideoInputFrame[LINES_PER_FRAME][PIXELS_PER_LINE/2];
        volatile unsigned short *usDestAddr;
   extern int flag_byte;


void main()
{
        /**
         * Initialize managed drivers and/or services that have been added to
         * the project.
         * @return zero on success
         */


        adi_initComponents();
        usDestAddr = &VideoInputFrame[0][0];     // init the pointer with the start  addr of the array

                // initialise system and core clock


                Init_PLL();

                // initialise SDRAM

                InitSDRAM();

                // initialise Sensor
                                                        // Calls an I2C function to program the sensor

                Init_TWI();

                if ((*pTWI_INT_STAT & XMTSERV) == flag_byte){
                   Write_TWI8(REGADDR);
                   flag_byte = 0;
                }
                else {
                    Write_TWI16(SNAPMODE);
                   }

                void Init_ProgrammableFlags();                  // Configures pin to enabling CMOS sensor

                Init_ProgrammableFlags();
```

```
                while(1){
                VideoSensorFrameCapture ( (unsigned short*) usDestAddr );              // function call to config
                void Truncating(unsigned short **VideoInputFrame);
                }


}
   void Init_PLL(void){
        ADI_SYSCTRL_VALUES write;
        write.uwPllCtl = 0x1800;             //PLL Divide register – CCLK = VCO/2 -> 206 MHz - SCLK = VCO/3 -> 104
MHz
        write.uwPllDiv = 0x0013;          // turn on PLL and set VCO requency at 12x 312
        write.uwVrCtl = 0XB4B0;          //  setting  for hibernate mode
        bfrom_SysControl (SYSCTRL_WRITE | SYSCTRL_PLLCTL |SYSCTRL_PLLDIV, &write, NULL);
        *pSIC_IWR = 0x1;                    // Interrupt wake-up Register set first bit for PLL wake-up.

        }


        void Init_ProgrammableFlags(void) {
        *pPORTHIO_DIR |= PH5 ;               // enable as output
        *pPORTH_FER &= PH5;                   // set PH5 high
        }


        void Init_TWI(void){
        int flag_byte = 1;
        *pTWI_CONTROL = TWI_ENA | PRESCALE_S ;      //enables TWI and simultaneously sets PRESCALE
        *pTWI_CLKDIV = 0x0811 ;
        *pTWI_MASTER_ADDR = 0xB8;                  // address of CMOS sensor for writing and reading
        *pTWI_FIFO_CTL = 0x000C;
        *pTWI_INT_MASK = 0x00B0;                   // enables main interrupt for master mode

        }

        void Write_TWI8(REGADDR){
        // Since in transmit mode
        *pTWI_MASTER_CTL = DCNT_S |FAST | MEN;
        *pTWI_XMT_DATA8 = REGADDR;
        }

        void Write_TWI8(SNAPMODE){
                // Since in transmit mode
                *pTWI_MASTER_CTL = DCNT_S |FAST | MEN;
                *pTWI_XMT_DATA16 = SNAPMODE;

        }




        void VideoSensorFrameCapture ( unsigned short *usPTR ){

        // configure DMA for PPI
        *pDMA0_X_COUNT= PIXELS_PER_LINE;                        // pixels per line da trasferire
        *pDMA0_Y_COUNT= LINES_PER_FRAME ;                       // linee per frame da trasferire
        *pDMA0_X_MODIFY= 0x2 ;  // siccome a 16 bit
        *pDMA0_Y_MODIFY= 0x2 ;
```

```
*pDMA0_PERIPHERAL_MAP = 0x0000;

 *pDMA0_START_ADDR = usPTR;                          // Destination address of the image
// Autobuffer mode| Restart FIFO | 2-D DMA | Bus width 16 bit | write to memory
 *pDMA0_CONFIG = STOPMODE | RESTART | DI_EN | DMA2D | WDSIZE_16 | WNR;

*pPORTHIO_SET &= PH5;                                // set PH5 -> high and sensor exits from STANDBY mode
// PPI0 setup
 *pPPI_FRAME = LINES_PER_FRAME;                      //The PPI is set to receive X lines per frame
 *pPPI_COUNT = PPICOUNT;                             //The PPI is set to stop receiving after X
 // number of samples for each line
// Polarity | 16-bit bus | no packing | two/three frame syncs | Input mode
*pPPI_CONTROL = POL_S |POL_C | DATALEN | CFG_GP_INPUT_2SYNCS | GP_INPUT_MODE;

// Upon to having configured PPI
*pDMA0_CONFIG  |= DMAEN;                             // DMA enable

}


// Configures master clock for image sensor
*pTIMER3_CONFIG = 0x0009;          // configures as PWM mode
*pTIMER3_WIDTH = PULSE_WIDTH3;
*pTIMER3_PERIOD = PULSE_PERIOD3;
*pTIMER_ENABLE |= TMR3;

// Configures EXPOSURE signal to enable integration time
*pTIMER0_CONFIG = 0x0005;          // configures as PWM mode
*pTIMER0_WIDTH = PULSE_WIDTH0;
*pTIMER_ENABLE |= TMR0;

// Upon enabled DMA controller
 *pPPI_CONTROL |= PORT_EN;                           // | Start PPI

}


void InitSDRAM(void){
if (*pEBIU_SDSTAT & SDRS) {     // SDC già acceso
*pEBIU_SDBCTL = 0x00000001;

*pEBIU_SDRRC = 0x0000060F; // impostazione del refresh della SDRAM

*pEBIU_SDGCTL = 0xE1999941; // configurazione in base al timing della SDRAM

}


void Truncating(unsigned short **VideoInputFrame){



for( int k=0;k<480;k++){
        for( int j=0;j<376;j++){
        int a=(VideoInputFrame[k][j] >> 2)& 0xFF;
        int b=(VideoInputFrame[k][j+1] << 6) & 0xFF00;
        VideoInputFrame[k][j] = a | b;
}
}
```

```
#include <stdlib.h>
#include <string.h>

#include "../common/spi.h"
#include "../common/flash.h"

/* Instruction Set */

#define READ_DATA                       0x03
#define FAST_READ                       0x0b
#define SECTOR_ERASE                    0xd8
#define BULK_ERASE                      0xc7
#define PAGE_PROGRAM                    0x02
#define READ_STATUS_REGISTER    0x05
#define WRITE_STATUS_REGISTER   0x01
#define WRITE_ENABLE                    0x06
#define WRITE_DISABLE                   0x04
#define JEDEC_ID                        0x9f
#define READ_SIGNATURE                  0xab

/* STATUS bits */
#define STATUS_BUSY                     0x01
#define STATUS_WEL                      0x02


#define NUM_SECTORS 512                 // Changed sectors from 32 to 512 for Spansion -DJH
#define SECTOR_SIZE 0x1000              // Changes sector size from 0x10000 to 0x1000 - DJH


static int s25fl116k_write_enable(const struct flash_info *fi);
static int s25fl116k_write_status(const struct flash_info *fi, uint8_t status, uint8_t dummy);

static int s25fl116k_open(struct flash_info *fi)
{
        fi->size = 0x200000;
        fi->number_of_regions = 1;
        fi->erase_block_regions = malloc(fi->number_of_regions * sizeof (struct erase_block_region));

        if (fi->erase_block_regions == NULL)
                return -1;

        fi->erase_block_regions[0].block_size = SECTOR_SIZE;
        fi->erase_block_regions[0].number_of_blocks = NUM_SECTORS;

        /* Clear all block protection bits */
        s25fl116k_write_enable(fi);
        s25fl116k_write_status(fi, 0, 0);

        return 0;
}

static int s25fl116k_close(struct flash_info *fi)
{
        free(fi->erase_block_regions);
        return 0;
}

static int s25fl116k_write_enable(const struct flash_info *fi)
```

```
{
	return generic_write_enable(fi, WRITE_ENABLE);
}

static int s25fl116k_write_disable(const struct flash_info *fi)
{
	return generic_write_disable(fi, WRITE_DISABLE);
}

static int s25fl116k_read_mid_did(struct flash_info *fi, uint8_t *mid, uint8_t *did)
{
	/* s25fl116k uses JEDEC_ID.  */
	return -1;
}

static int s25fl116k_read_uid(const struct flash_info *fi, uint64_t *uid)
{
	return -1;
}

static int s25fl116k_read_jedec_id(const struct flash_info *fi, uint8_t *mid,
		uint8_t *memory_type_id, uint8_t *capacity_id)
{
	return generic_read_jedec_id(fi, JEDEC_ID, mid, memory_type_id, capacity_id);
}

static int s25fl116k_read_status(const struct flash_info *fi, uint8_t *status)
{
	uint8_t tbuf[1];
	uint8_t rbuf[1];
	int count;

	select_flash();

	count = 0;
	assign_instruction(fi, tbuf, READ_STATUS_REGISTER, &count);
	spi_send(tbuf, count);

	spi_recv(rbuf, 1);
	*status = rbuf[0];

	unselect_flash();

	return 0;
}

static int s25fl116k_not_busy(const uint8_t status)
{
	return (status & STATUS_BUSY) ? 0 : 1;
}

static int s25fl116k_wait_ready(const struct flash_info *fi)
{
	uint8_t tbuf[1];
	int count;

	select_flash();

	count = 0;
```

```
	assign_instruction(fi, tbuf, READ_STATUS_REGISTER, &count);
	spi_send(tbuf, count);

	spi_recv_until(s25fl116k_not_busy);

	unselect_flash();

	return 0;
}

static int s25fl116k_write_status(const struct flash_info *fi, uint8_t status, uint8_t dummy)
{
	uint8_t tbuf[2];
	int count;
	uint8_t old_status;

	s25fl116k_write_enable(fi);

	s25fl116k_read_status(fi, &old_status);
	if (!(old_status & STATUS_WEL))
		return -1;

	select_flash();

	count = 0;
	assign_instruction(fi, tbuf, WRITE_STATUS_REGISTER, &count);
	tbuf[count++] = status;
	spi_send(tbuf, count);

	unselect_flash();

	s25fl116k_wait_ready(fi);

	return 0;
}

static int s25fl116k_read(const struct flash_info *fi, uint32_t addr, uint8_t *buf, int size)
{
	uint8_t read_insn;
	uint8_t tbuf[5];
	int count;

	/*  TODO frequency will limit instruction chosen.  */
	read_insn = READ_DATA;

	select_flash();

	count = 0;
	assign_instruction(fi, tbuf, read_insn, &count);
	assign_address(fi, tbuf, addr, &count);

	if (read_insn == FAST_READ)
		count++;

	spi_send(tbuf, count);

	spi_recv(buf, size);

	unselect_flash();
```

```
	return 0;
}

/* TODO make erase functions generic */

/* Internal function used by s25fl116k_erase and s25fl116k_erase_chip */

static int s25fl116k_erase_internal(const struct flash_info *fi,
		uint32_t addr, erase_type_t erase_type)
{
	uint8_t tbuf[4];
	int count;
	uint8_t status;
	uint8_t insn;
	uint32_t addr_mask;

	switch (erase_type)
	{
	case ERASE_4KB:
		insn = SECTOR_ERASE;
		addr_mask = 0xfffff000;
		break;

	case ERASE_CHIP:
		insn = BULK_ERASE;
		addr_mask = 0;
		break;

	default:
		/* bad erase type */
		return -1;
	}

	s25fl116k_write_enable(fi);

	s25fl116k_read_status(fi, &status);
	if (!(status & STATUS_WEL))
		return -1;

	select_flash();

	count = 0;
	assign_instruction(fi, tbuf, insn, &count);
	if (erase_type != ERASE_CHIP)
		assign_address(fi, tbuf, addr & addr_mask, &count);
	spi_send(tbuf, count);

	unselect_flash();

	s25fl116k_wait_ready(fi);

	return 0;
}

static int s25fl116k_erase(const struct flash_info *fi, uint32_t addr, int size)
{
#define ERASE_TYPE ERASE_4KB
#define ERASE_SIZE (4 * 1024)
```

```
	size += addr % ERASE_SIZE;
	addr -= addr % ERASE_SIZE;

	while (size > 0)
	{
		s25fl116k_erase_internal(fi, addr, ERASE_TYPE);
		addr += ERASE_SIZE;
		size -= ERASE_SIZE;
	}

	return 0;
}

static int s25fl116k_erase_chip(const struct flash_info *fi)
{
	return s25fl116k_erase_internal(fi, 0, ERASE_CHIP);
}

#define PAGE_SIZE 256

static int s25fl116k_page_program (struct flash_info *fi,
		uint32_t addr, const uint8_t *buf, int size)
{
	uint8_t tbuf[4 + size];
	int count;
	uint8_t old_status;

	if ((addr % PAGE_SIZE) + size > PAGE_SIZE)
		return -1;

	s25fl116k_write_enable (fi);

	s25fl116k_read_status (fi, &old_status);
	if (!(old_status & STATUS_WEL))
		return -1;

	select_flash ();

	count = 0;
	assign_instruction (fi, tbuf, PAGE_PROGRAM, &count);
	assign_address (fi, tbuf, addr, &count);
	memcpy (tbuf + count, buf, size);
	spi_send (tbuf, count + size);

	unselect_flash ();

	s25fl116k_wait_ready (fi);

	return 0;
}

/* Program arbitary size */
static int s25fl116k_program (struct flash_info *fi,
		uint32_t addr, const uint8_t *buf, int size)
{
	int program_size;

	while (size > 0)
```

```
	{
		if (addr % PAGE_SIZE != 0)
			program_size = ((PAGE_SIZE - addr % PAGE_SIZE) < size
					? (PAGE_SIZE - addr % PAGE_SIZE) : size);
		else if (size >= PAGE_SIZE)
			program_size = PAGE_SIZE;
		else
			program_size = size;

		s25fl116k_page_program (fi, addr, buf, program_size);

		addr += program_size;
		buf += program_size;
		size -= program_size;
	}

	return 0;
}

static int s25fl116k_reset(const struct flash_info *fi)
{
	/* TODO implement it */
	return -1;
}

struct flash_info s25fl116k_info =
{
	"s25fl116k",	/* name */
	"Spansion",	/* name */
	0x01,/* manufacturer ID m25 = 0x20 */                                                    //
Spansion manID = 0x01 - DJH
	/* device ID.  s25fl116k's DID is 16-bit and is "memory type ID, capacity Id".
	 * so we just check them instead of device ID.  */
	0xff,	/*0xff*/
	0x40,		/* memory type ID m25 = 0x20 */                                         //Spansion
memID = 0x40 - DJH
	0x15,		/* capacity ID m25 = 0x15 */
	//Spansion capID = 0x15 - DJH
	STANDARD,	/* supported modes */

	/* The following three fields are better be
	   initialized in flash_open functions.  */
	0,			/* size in Byte */
	0,			/* number_of_regions */
	NULL,		/* erase_block_regions */

	25,		/* max_freq */
	0,		/* max_quad_freq */
	20,		/* max_read_data_freq */

	0,		/* cpha */
	0,		/* cpol */
	0,		/* lsb_first */
	0,		/* start_on_mosi */

	0,		/* frequency */
	STANDARD,	/* current_mode */

	s25fl116k_open,
```

```
        s25fl116k_close,

        s25fl116k_read_mid_did,
        s25fl116k_read_uid,
        s25fl116k_read_jedec_id,
        s25fl116k_read_status,
        NULL, /* read_status2 */
        s25fl116k_write_status,
        NULL, /* flash_enable_quad_mode */
        NULL, /* flash_disable_quad_mode */
        s25fl116k_read,
        s25fl116k_write_enable,
        s25fl116k_write_disable,
        s25fl116k_erase,
        s25fl116k_erase_chip,
        s25fl116k_program,
        s25fl116k_reset,
};
```

Appendice D

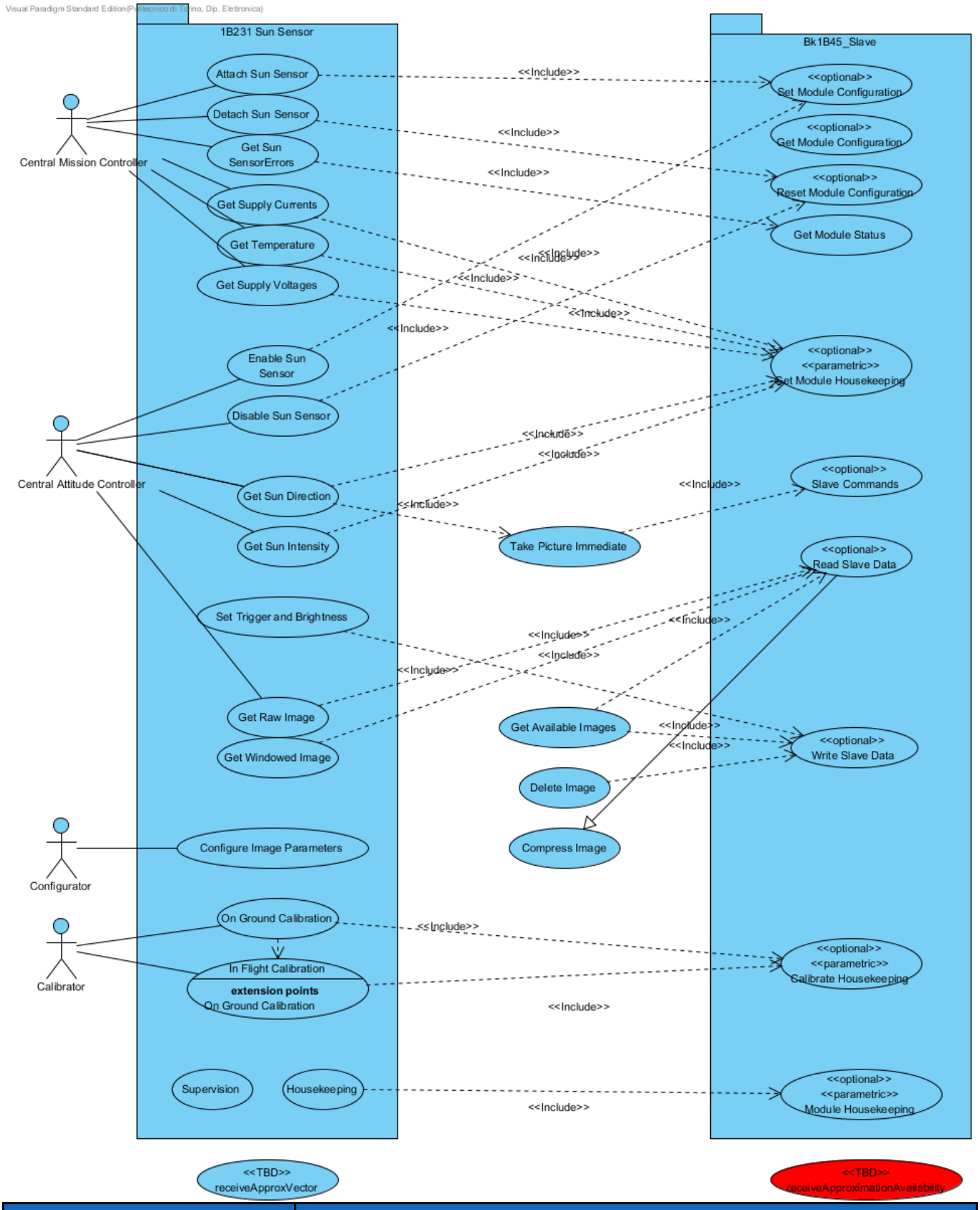
Documentazione Sun Sensor - Visual Paradigm

Table of Contents

Table of Figures

Use Case Diagram

1. 1B231 Sun Sensor



Name	Value

Name	1B231 Sun Sensor
Create Date Time	28-gen-2010 13.24.24
Last Modified	3-dic-2014 10.20.14

1.1. Summary

Name	Documentation
Central Attitude Controller	The entity (likely a SW routine running on the OBC in charge of managing the 1B2 Attitude Control Subsystem in nominal operation. Fault and emergency handling are left to the Central Mission Controller.
Configurator	The person in charge of configuring HW/SW parameters according to spacecraft architecture and mission requirements.
Calibrator	The person (together with the appropriate equipment) in charge of calibrating both the tiles and the whole satellite.
Central Mission Controller	The entity (likely a SW routine running on the OBC) in charge of satellite supervision, fault detection and management and emergency management.
Get Windowed Image	Get raw image with reduced size in order to high resolution for computation of barycenter. This operation must do upon a possible tracking algorithm.
Slave Commands	OBSOLETE: The Master actor sends a data-less command to the **System**. The Master can use up to 8 different Slave Commands, to issue as many commands to the system. It uses the Command Only use case, by issuing the (model element not found) through (model element not found), where **x** (0..7) identifies the message type; **x** does not identify the sequence in which commands are issued, but the (model element not found)-defined type of command. The (model element not found) can use as many message types he wants. This use case is optional, therefore the (model element not found) shall #define an identifier whose name is contained in the tagged value **define**. After command has been received and checked, the 1B45_Subsystem_Serial_Data_Bus calls the CommandInterpreter.interpret(command: t_Commands, error: t_LastError): void operation and passes the command as an argument. The (model element not found) shall then write his command interpretation routine inside the interpret(command: t_Commands, error: t_LastError): void operation.
Write Slave Data	OBSOLETE: The Master actor sends up to 256B of Designer-defined data to the **System**. The Master can use up to 8 different Write Slave Data commands, to issue messages to as many different subsystems. It uses the Write Data use case, by issuing the (model element not found) through (model element not found) command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are written, but the Designer-defined type (therefore the destination subsystem). The Designer can use as many message types he wants.

	This use case is optional, therefore the Designer shall **#define** an identifier whose name is contained in thIs use case's tagged value **define**. After command has been received completely and checked, the 1B45_Subsystem_Serial_Data_Buswill allocate a buffer inside the Buffers class, then call the CommandInterpreter:interpret(command: t_Commands, error: t_LastError): void operation and pass the command as an argument. The Designer shall then write his command interpretation routine inside the interpret(command: t_Commands, error: t_LastError): void operation. The command interpreter shall access the data message from class Buffers with the following sequence: 1. Call Buffers.(model element not found), which returns a pointer to the buffer containing the message and its length (number of bytes in data section, excluding the command itself and other ancillary data. 2. Use the message from the above vector 3. When the message is not used anymore, release the buffer using Buffers.release(buffer: byte*): bool
Compress Image	Once stored image in one dedicated memory, it's compressed after to have issued command CMD_COMPRESS_IMAGE that uses Slave Commands use case TBD
Configure Image Parameters	Configures at compilation/integration time parameters like: 1. Image size of imager (first row (ROW_START: unsigned short)/column (COL_START: unsigned short) to be read out,WINDOWS_HEIGHT: unsigned short/ WINDOWS_WIDTH: unsigned short/, row/column flip) 2. Operating Mode (Slave/Master/Snapshot mode)by CHIP_CONTROL: unsigned short 3. Simultaneous or sequential mode 4. Integration time setting COARSE_SHUTTER_WIDTH: unsigned short in SNAPSHOT MODE 5. Enable High Dynamic Range (set in R0x0F[0]) and sets V1 /V2/V3/V4 levels) by means of V1_STEP: unsigned char, V2_STEP: unsigned char, V3_STEP: unsigned char and V4_STEP: unsigned char 6. READ MODE (normal or binning of rows/columns)by READ_MODE: unsigned short -> reduced image size snd altered frame rate (just for row bin up to increased by 4)such as pixel clock (just for column bin up to reduced by one-fourth) 7. Black calibration by means of BLACK_CALIBRATIOM: unsigned short , ANALOG_GAIN: unsigned short and VREF: unsigned short 8. Analog Gain ANALOG_GAIN: unsigned short and Digital Gain 9. Noise Correction NOISE_CORRECTION: unsigned short 10. Enable compading 12 -> 10 11. Enable AGC/AEG

	For each register there are two different options to be set: Context A e Context B
Delete Image	This use case Delete Image allows to discard one image by means of CMD_DELETE_IMAGE followed by the following data: 1. buffer[0]: number of image to be deleted according to specific section of FLASH memory which has been used
Get Available Images	Once stored everyone available images, OBC can issue CMD_GET_AVAILABLE_IMAGES command to request to image processor different images for additional purposes. As a result of this, image processor can send towards OBC the requirement bytes to be read Read Slave Data. This is managed by Get Available Images use case.
Get Raw Image	Get raw image on behalf of OBC for specific purposes. Uses Read Slave Data use case by issuing command CMD_GET_RAW_IMAGE followed by the following
Get Supply Currents	
Get Supply Voltages	Periodically measures and stores into housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector the following voltages: 1. 1.4V of ... into vector element HK_VOLTAGE_1V4 2. 2.5V
Get Temperature	Returns the last acquired value of temperature.....
Set Trigger and Brightness	Configures integration time for all subsequent picture acquisitions. Uses Write Slave Data use case by issuing command CMD_SET_TRIGGER followed by the following data: 1. buffer[0]: Integration time in number of rows And/or configures voltages for all subsequent picture acquisitions. Uses Write Slave Data use case by issuing command CMD_SET_BRIGHTNESS followed by the following data: 1. buffer[0]: light intensity in value of voltage for V1 2. buffer[1]: light intensity in value of voltage for V2 3. buffer[2]: light intensity in value of voltage for V3 4. buffer[3]: light intensity in value of voltage for V4 (seves as an antiblooming for photodetector) 5. buffer[4]: ADC gain (by default is X1) 6. buffer[5]: ADC voltage references (by default is 1.4V) Light intensities work approximately as a reciprocal of the partial exposure time.
Take Picture Immediate	By means of CMD_TAKE_PICTURE_IMMEDIATE command, it allows image processor to take one picture without any delays since probably spacecraft is in desirable situation to do

	it. This command enables any kind of routine to acquire one frame and store it in SDRAM temporarily.
Read Slave Data	OBSOLETE: The Master actor reads up to 256B of Designer-defined data from the **System**. The Master can use up to 8 different Read Slave Data commands, to read messages from as many different subsystems. It uses the Read Data use case, by issuing a (model element not found) through (model element not found) command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are written, but the Designer-defined type (therefore the source subsystem). The Designer can use as many message types he wants. This use case is optional, therefore the Designer shall **#define** an identifier whose name is contained in the tagged value **define**. The message to be read must already be present into a buffer inside the Buffers class before the Master actor starts this use case, otherwise this use case returns unpredictable results and sets (model element not found). this can be accomplished by: 1. Call Buffers.lock(command: t_Commands): byte*, which returns a pointer to an empty buffer available for the message (buffer storage is already allocated). 2. Fill up the buffer with the message data he wants 3. As soon as the message is complete, declare that the buffer is ready and specify its length by using Buffers.ready(command: t_Commands, length: ushort): bool operation. All the above operations shall be done before the Master issues the Read Slave Data command. Once the command is received and checked, the 1B45_Subsystem_Serial_Data_Buswill immediately send the message back to the Master, then it calls the CommandInterpreter:interpret(command: t_Commands, error: t_LastError): void operation and pass the command as an argument. The Designer shall then write his command interpretation routine inside the interpret(command: t_Commands, error: t_LastError): void operation. The Designer shall decide what to do once the message has been read. Either: • if the <<optional>> (model element not found) use case is NOT implemented, the buffer is automatically released by the 1B45_Subsystem_Serial_Data_Bus, before the call to the interpret(command: t_Commands, error: t_LastError): void..Since then, the Designer can prepare data for the next Read Slave Data command, whenever he likes. • if the <<optional>> (model element not found) use case is implemented: the 1B45_Subsystem_Serial_Data_Busleaves data in the buffer and keeps it available for further reading, in case the Master does not receive it properly. In this case, the Master, as soon as it properly receives data, shall use the (model element not found) use case, with which the 1B45_Subsystem_Serial_Data_Busautomatically releases the buffer , before calling the interpret(command: t_Commands, error: t_LastError): void operation..Since then, the Designer can prepare data for the next Read Slave Data command, whenever he likes. To decide for either the former or the latter case, the Designer shall implement (respectively, not implement) the (model element not found) use case.
Module Housekeeping	An autonomous function which periodically samples (with period SAMPLETIME), calibrates and stores a number of Designer-defined housekeeping data. Values are stored as ushort words, with Designer-defined resolution (at most 16 bits), in

	the housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector. The Designer shall define a list of parameters to be acquired containing at least, for each parameter: • type of parameter and location of sensor • resolution (number of bits and signed/unsigned data type) • scale factor (namely, number of physical units per each unit of stored parameter) • offset (namely, real value corresponding to a stored 0 value) • sample rate (samples/s) and optional analog/digital filtering • index of data into the internal housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector (starting from 0). The length of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector shall be at most 128 short. This function also optionally computes statistics (min, max and average values) of a subset of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector; namely, LENGTH_STATISTICS elements starting from index FIRST_STATISTICS. The average is approximated by low-pass filtering, such that, at every new housekeeping cycle, the new filtered value will be: **y(t) = (short) (((long) y(t-1) * (FILTER_DEN - FILTER_NUM)) / FILTER_DEN) + (short) (((long) in * FILTER_NUM) / FILTER_DEN)** where **y(t)** is the filtered value, while **in** is the last sampled value; both of type short, while FILTER_NUM and FILTER_DEN define filter's time constant (FILTER_NUM must be smaller than FILTER_DEN, under Designer's responsibility). The time constant of the filter will therefore be: **τ = SAMPLETIME * (FILTER_DEN / FILTER_NUM)** where SAMPLETIME is the the period of housekeeping cycle (see Module Housekeeping), while the bandwidth is: **BW = 1 / (2 * π * SAMPLETIME * (FILTER_DEN / FILTER_NUM))** Statistics are stored into vector statistics: HK_REDUNDANCY[3][LENGTH_STATISTICS]. This function also computes history of a subset of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector; namely, the last DEPTH_HISTORY samples of LENGTH_HISTORY elements starting from index FIRST_HISTORY. The history is stored into vector history: HK_REDUNDANCY[DEPTH_HISTORY][LENGTH_HISTORY]. The Calibrate Housekeeping use case allows to calibrate all (or a number of) sensors by applying appropriate offset and gain corrections.. The Get Module Housekeeping use case returns the last acquired housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector. The Get Module History use case returns history of the last DEPTH_HISTORY samples of the housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector (or a part of it, namely LENGTH_HISTORY elements starting from FIRST_HISTORY), for at most 256B of data. The Designer shall define the list of parameters for which the history has to be kept and the number of samples which have to be saved.

	The Get Module Statistics use case returns min, max and a low-pass filtering of the housekeeping vector (or a TBD part of it). Low-pass filtering time constant is also TBD. Statistics shall be computed starting from the last Reset Statistics. The Designer shall define the list of parameters for which the statistics have to be kept. The Reset Module Statistics use case resets the statistics for the housekeeping vector. This use case is optional, therefore the Configurator shall #define an identifier whose name is contained in the tagged value define.

Get Module Status	`It returns to the OBC the status information (statusRegister: CS_REDUNDANCY[LENGTH_STATUS]) of one of many spacecraft Tiles.`
	
	`This use case first uses the Read Data use case by issuing the CMD_GET_STATUS command with the address of the desired Tile, which returns its statusRegister: CS_REDUNDANCY[LENGTH_STATUS] to the Master.`
	`Using this use case also also clears the Error Indicator signal (if present).`

Get Sun Intensity	`Get Sun Intensity for later to filter possible bright sources as stars.`
receiveApproxVector	Actor sends x, y, and z coordinates of the unitary vector in J2000 coordinates, derived from the measurements of the sun sensor and/or magnetic field of the earth.
receiveApproximationAvailability	Receives the availability or not, of the initial approximation of attitude vector. If not available, the sensor performs a Lost in Space measurement.
Calibrate Housekeeping	Writes into the Slave calibration data for housekeeping sensors (if any). For each sensor, two values are to be supplied: an **unsigned t_sensor offset** and an **unsigned t_sensor gain_correction**. Both parameters are of **unsigned t_sensor** type. Values are written into FLASH memory. The **Module Housekeeping** use case shall then write into the **housekeeping** vector: **for (i=0; i<LENGTH_CORRECTION); i++)** **housekeeping[i] = (acquired_value[i] - offset[i]) * (gain_correction[i] / (max(t_sensor)/2));** where: **t_sensor housekeeping[i]** is the i-th value written in the housekeeping vector; **t_sensor acquired_value[i]** is the i-th value (unsigned integer) acquired from the i-th sensor **t_sensor offset[i]** is the i-th offset value **t_sensor gain_correction[i]** is the i-th gain_correction value **max(t_sensor)** is the max value which can be expressed by an (unsigned t_sensor)

	Only the first **LENGTH_CALIBRATION** sensors (from 0 to **LENGTH_CALIBRATION-1**) are calibrated. For all the other sensors, the acquired value is written into the **housekeeping** vector directly. Also if this use case is not implemented, the acquired value is written into the **housekeeping** vector directly. The default values for the offset and gain_correction parameters are set by the **Tester** (in the factory) after appropriate calibration. These parameters can be later updated by the Master**.** This use case is different from the **Set Module Configuration** and the **Configure Module** use cases: **Configure Module** allows a compile-time configuration, which cannot be changed during the life of the **System**; **Set Module Configuration** allows run-time configuration changes (if foreseen) but these do without affecting sensor calibration; **Calibrate Housekeeping** allows run-time (or post-fabrication) changes of sensor calibration parameters; This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
Get Module Housekeeping	Returns last measured housekeeping data (see use case Module Housekeeping for details). The Master shall uses the Read Data use case by issuing the CMD_GET_HOUSEKEEPING command. The Slave shall assemble ALL last saved housekeeping data into the response message and return them to the Master. No consistency is guaranteed between sampling time of different housekeeping data; it may therefore happen that some values have been just sampled, while others may be several seconds old, depending on sampling rate of Module Housekeeping. This use case is optional; refer to the Optional Use Cases section for further details. This use case is also parametric; its parameters are the following: • LENGTH_HOUSEKEEPING, namely the number of elements (different sensors, measurements, other data) that will be stored into the housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector.
Reset Module Configuration	Resets to 0 some bits of the internal configuration word (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) of one of many spacecraft Tiles. The OBC shall send to the Tile as many bits as are in its configuration word configRegister: CS_REDUNDANCY[LENGTH_CONFIG]. Each bit which is 1 in the message will reset to zero the corresponding bit in the Tile configuration word. The other bits are left unchanged. This use case sends configuration bits by means of the Write Data use case, by issuing the CMD_RESET_CONFIGURATION command with the address of the desired Tile. By no means, configRegister[0] can be written, as this contains the Designer-defined HW/SW version. This use case differs from a set of related use cases; in particular: • the use case Configure Module allows a compile-time configuration. • the use case Reset Module Configuration resets to 0 a few bits in the

	configuration word. It cannot set any bit to 1. • the use case Set Module Configuration sets to 1 a few bits in the configuration word. It cannot reset any bit to 0. • the use case Write Module Configuration writes (either 0 or 1) all the bits of the configuration word. This use case is optional, therefore the Configurator shall #define inside file platform.h an identifier whose name is contained in the tagged value define.
Set Module Configuration	Sets to 1 some bits of the internal configuration word (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) of one of many spacecraft Tiles. The OBC shall send to the Tile as many bits as are in its configuration word configRegister: CS_REDUNDANCY[LENGTH_CONFIG]. Each bit which is 1 in the message will set to 1 the corresponding bit in the Tile configuration word. The other bits are left unchanged. This use case sends configuration bits by means of the Write Data use case, by issuing the CMD_SET_CONFIGURATION command with the address of the desired Tile. By no means, configRegister[0] can be written, as this contains the Designer-defined HW/SW version. This use case differs from a set of related use cases; in particular: • the use case Configure Module allows a compile-time configuration. • the use case Reset Module Configuration resets to 0 a few bits in the configuration word. It cannot set any bit to 1. • the use case Set Module Configuration sets to 1 a few bits in the configuration word. It cannot reset any bit to 0. • the use case Write Module Configuration writes (either 0 or 1) all the bits of the configuration word. This use case is optional, therefore the Configurator shall #define inside file platform.h an identifier whose name is contained in the tagged value define.
Get Module Configuration	It returns to the OBC the current configuration word (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) of one of many spacecraft Tiles. The configuration word was last written by the OBC, or the default for the Tile if not yet modified by the OBC. This use case first uses the Read Data use case by issuing the CMD_GET_CONFIGURATION command with the address of the desired Tile, which then returns its configRegister:

	CS_REDUNDANCY[LENGTH_CONFIG] to the Master. This use case is optional, therefore the Configurator shall #define inside file platform.h an identifier whose name is contained in the tagged value define.
Housekeeping	
Supervision	
Get Sun SensorErrors	Returns the complete list of errors which may afflict the sensors of Subsystem and are detected by Supervision Subsystem use case. Errors are listed by type as bits composing the error word (model element not found).housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING][(model element not found)], respecting the following order: ON_CURR_EXCESS->when image processor is operating, processor current exceeds maximum allowed value ON_CURR_LACK->when image processor is operating, processor current is less than minimum functional value OFF_CURR_EXCESS ->when image processor is disabled, there is a significant current which supply processor ERR_SENS_ENABLE ->error which stucks bit (model element not found) at 'true' ERR_SENS_STATUS->error which stucks bit (model element not found) at 'true' ERR_SENS_ATTACH->error which stucks bit (model element not found) at 'true' ERR_IM_PROC_ENABLE->error which stucks bit (model element not found) at 'true' ERR_IM_PROC_ATTACH->error which stucks bit (model element not found) at 'true' LOW_PDB_VOLT->error in Power Distribution Bus Voltage: voltage is below minimum critical value This use case makes use of Get Module Housekeeping use case of Bk1B45_Slave package.
Detach Sun Sensor	Flags sun sensor as unavailable to the user. Sensor is considered "detached" either if it is NOT physically present on the tile (flag (model element not found) of (model element not found)) or it has not been attached (see use case Attach Subsytem). If the sensor is either detached or disabled (see use case Disable Subsystem), no further action can be carried on. This Use Case makes use of Reset Module Configuration use case of 1B45 package, with bit ((model element not found)) of configuration word ((model element not found)). The difference between Detach Subsystem and Disable Subsystem is that the former prevents the sensore from being enabled, therefore it disables any further action on the sensor, including Enable Subsystem.

	The Detach Subsystem use case is intended only in rare occasions, to permanently disable a faulty sensor. Since the actor can know if the sensor is attached (see use case Has Subsystem), the ARAMIS-level AOCS algorithms can be adapted consequently.
Attach Sun Sensor	Sun sensor as available to the user. Sensor is considered "attached" if it is physically present on the tile (flag (model element not found) of (model element not found)) and it has not been detached (see use case Detach Subsystem). If the sensor is attached and enabled (see use case Enable Subsystem), any further action of image processor will be carried on. This Use Case makes use of Set Module Configuration use case of 1B45 package, with bit (model element not found) of configuration word ((model element not found)) The difference between Attach Subsytem and Enable Subsystem is that the former has effects only if the sensor is physically present on the tile, while the latter has effects only if the sensor is attached, therefore the former enables the latter. The Attach Subsytem use case is intended only in rare occasions, to recover from an erroneous Detach Subsystem operation.
Enable Sun Sensor	Enables sun sensor(STANDBY signal as low). Any further action of sensor will be carried on regularly. By default the sensor is disabled. When enabled, the sensor shall be active only when requested by the appropriate use cases (see (model element not found)). When enabled but not active, power consumption should be as little as possible. The Master shall set (model element not found) bit of Configuration Word (model element not found) using Set Module Configuration use case of Bk1B45_Slave package.
Disable Sun Sensor	Disables sun sensor (STANDBY signal as HIGH) . Such condition allows to sensor of going into standby mode and then it completes the current frame before disabling the digital logic, internal clocks and analog power enable signal. Any further action of sensor will be disregarded. If the sensor is in process of getting raw image while the Disable Subsystem use case is applied, the sensor is turned off upon it acquires frame.. When disabled, the sensor shall draw negligible power from the Power Bus. The **Master** shall reset (model element not found) bit of Configuration Word (model element not found) using Reset Module Configuration use case of Bk1B45_Slave package. The default is disabled.
In Flight Calibration	Allows to change value of coil area, offset and gain of magnetometer, using three signed 16-bits integers. Units are: TBD, TBD, TBD. Uses Calibrate Housekeeping use case of Bk1B45_Slave package, with: • magnetometer gain on x-axis in field **magnetometer_gain_X**, • magnetometer gain on y-axis in field **magnetometer_gain_Y**,

	• magnetometer offset on x-axis in field **magnetometer_offset_X**, • magnetometer offset on y-axis in field **magnetometer_offset_Y**, • coil area in field **coil_area**.
On Ground Calibration	
Get Sun Direction	Returns the last measured direction of sun; two numbers, 16-bits each, giving sun direction along X and Y axis. It can be read using **getHousekeeping** method of **1B45** package. • Value **housekeeping[SUN_SENSOR_X]** returns the angle between sun direction and the y-z plane, in units of in 180/2048 degrees (= pi/2048 radians); positive value indicates that sun is towards positive x-axis. In case sun is not visible, it returns -32768. • Value **housekeeping[SUN_SENSOR_Y]** returns the angle between sun direction and the x-z plane, in units of in 180/2048 degrees (= pi/2048 radians); positive value indicates that sun is towards positive y-axis. In case sun is not visible, it returns -32768. The (model element not found) will read components of captured image as variable **t_sensor housekeeping** using the Get Module Housekeeping use case of Bk1B45_Slave package.
Bk1B45_Slave	This package contains all specifications of the **Peripheral** (slave) side of the 1B45_Subsystem_Serial_Data_Bus. In these diagrams, the **System** is the Peripheral side. It comprises two Use Case diagrams: • **Housekeeping Use Case Diagram**, which incorporates all use cases related to reading from the slave its housekeeping data and the related history and statistics, system status; to issue commands to the system like wake-up, standby and reset, and to read/write application-specific (**Designer**-defined) data. • **Supply, Enable, Configuration Use Case Diagram**, which incorporates use cases related to power supply, static configuration and testing of the system. The document only describes commonly-used functions of a **Peripheral** (slave), and the Designer may add as many functions as he requires. Yet any added function should comply as much as possible to the basic protocol described herein. Furthermore, not all functions described in these use case diagrams need be implemented. Most use cases are optional. If used, they shall be implemented as specified. If not used, they can be disabled by removing appropriate attributes from the relevant classes, as indicated in the **Configure Module** use case.
1B231 Sun Sensor	

1.1.1.1. Documentation

The use cases of the magnetic attitude subsystem of the ARAMIS architecture.

1.2. Details

1.2.1. Central Attitude Controller

Name	Value
Documentation	`The entity (likely a SW routine running on the OBC in charge of managing the 1B2 Attitude Control Subsystem in nominal operation. Fault and emergency handling are left to the Central Mission Controller.`
Visibility	public

Abstract	false
Leaf	false
Root	false
Business Model	false
Project Management	**Name** / **Value** Author / L.M. Reyneri Create Date Time / 1-giu-2010 14.12.05 Last Modified / 30-gen-2014 15.35.02

1.2.1.1. Relationships

Unnamed Association	
To	**Name** / **Value** End Model Element / Enable Sun Sensor Provide Property Getter Method / false Provide Property Setter Method / false Multiplicity / Unspecified Visibility / Unspecified Aggregation Kind / None Navigable / Navigable Derived / false Derived Union / false Read Only / false Static / false Leaf / false Project Management / **Name** / **Value** Author / dmasera Create Date Time / 15-mar-2010 17.13.08 Last Modified / 1-giu-2010 17.41.35
Abstract	false
Leaf	false
Visibility	Unspecified
Derived	false
Project Management	**Name** / **Value** Author / dmasera Create Date Time / 15-mar-2010 17.13.08

	Last Modified	25-nov-2014 9.40.16

Unnamed Association

To

Name	Value
End Model Element	Disable Sun Sensor
Provide Property Getter Method	false
Provide Property Setter Method	false
Multiplicity	Unspecified
Visibility	Unspecified
Aggregation Kind	None
Navigable	Navigable
Derived	false
Derived Union	false
Read Only	false
Static	false
Leaf	false
Project Management	

Name	Value
Author	dmasera
Create Date Time	15-mar-2010 17.13.08
Last Modified	1-giu-2010 17.41.35

Abstract	false
Leaf	false
Visibility	Unspecified
Derived	false
Project Management	

Name	Value
Author	dmasera
Create Date Time	15-mar-2010 17.13.08
Last Modified	25-nov-2014 9.40.16

Unnamed Association

To

Name	Value
End Model Element	Get Sun Direction
Provide Property Getter Method	false
Provide Property Setter Method	false

	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author / Administrator Create Date Time / 19-mar-2010 17.32.54 Last Modified / 1-giu-2010 17.41.35
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 19-mar-2010 17.32.54 Last Modified / 25-nov-2014 9.40.16	

Unnamed Association		
To	**Name**	**Value**
	End Model Element	Get Sun Direction
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false

	Project Management	**Name**	**Value**
		Author	dmasera
		Create Date Time	15-mar-2010 17.13.05
		Last Modified	1-giu-2010 17.41.35
Abstract	false		
Leaf	false		
Visibility	Unspecified		
Derived	false		
Project Management	**Name**	**Value**	
	Author	dmasera	
	Create Date Time	15-mar-2010 17.13.05	
	Last Modified	25-nov-2014 9.40.16	

Unnamed Association			
To	**Name**	**Value**	
	End Model Element	Get Sun Intensity	
	Provide Property Getter Method	false	
	Provide Property Setter Method	false	
	Multiplicity	Unspecified	
	Visibility	private	
	Aggregation Kind	None	
	Navigable	Navigable	
	Derived	false	
	Derived Union	false	
	Read Only	false	
	Static	false	
	Leaf	false	
	Type	Get Sun Intensity	
	Project Management	**Name**	**Value**
		Author	gabriele.defranciscis
		Create Date Time	17-giu-2014 16.32.49
Abstract	false		
Leaf	false		
Visibility	Unspecified		
Derived	false		

Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	17-giu-2014 16.32.49
	Last Modified	25-nov-2014 9.40.16

Unnamed Association		
To	Name	Value
	End Model Element	Get Raw Image
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Type	Get Raw Image
	Project Management	Name: Author; Value: gabriele.defranciscis Name: Create Date Time; Value: 13-nov-2014 15.13.41
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	13-nov-2014 15.13.41
	Last Modified	25-nov-2014 9.40.16

1.2.2. Configurator

Name	Value

Documentation	The person in charge of configuring HW/SW parameters according to spacecraft architecture and mission requirements.
ID	AC01
Visibility	public
Abstract	false
Leaf	false
Root	false
Business Model	false
Project Management	**Name** / **Value** Author: leonardo.reyneri Create Date Time: 23-mag-2012 10.36.44 Last Modified: 29-giu-2012 15.33.13

1.2.2.1. Relationships

Unnamed Association

To

Name	Value
End Model Element	Configure Image Parameters
Provide Property Getter Method	false
Provide Property Setter Method	false
Multiplicity	Unspecified
Visibility	private
Aggregation Kind	None
Navigable	Navigable
Derived	false
Derived Union	false
Read Only	false
Static	false
Leaf	false
Type	Configure Image Parameters
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 12.18.01

Abstract	false
Leaf	false
Visibility	Unspecified

Derived	false
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 12.18.01 Last Modified: 25-nov-2014 9.40.16

1.2.3.  Calibrator

Name	Value
Documentation	`The person (together with the appropriate equipment) in charge of calibrating both the tiles and the whole satellite.`
Visibility	public
Abstract	false
Leaf	false
Root	false
Business Model	false
Project Management	**Name** / **Value** Author: L.M. Reyneri Create Date Time: 8-giu-2010 7.23.14 Last Modified: 24-mag-2012 17.18.27

1.2.3.1. Relationships

Unnamed Association	
To	**Name** / **Value**
	End Model Element: In Flight Calibration
	Provide Property Getter Method: false
	Provide Property Setter Method: false
	Multiplicity: Unspecified
	Visibility: private
	Aggregation Kind: None
	Navigable: Navigable
	Derived: false
	Derived Union: false
	Read Only: false
	Static: false

	Leaf	false
	Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 9-giu-2011 17.47.05
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 25-nov-2014 9.40.16	

Unnamed Association		
To	**Name**	**Value**
	End Model Element	On Ground Calibration
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author: Administrator Create Date Time: 19-mar-2010 17.32.02 Last Modified: 9-giu-2011 17.39.34
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	

Project Management	**Name**	**Value**
	Author	Administrator
	Create Date Time	19-mar-2010 17.32.02
	Last Modified	25-nov-2014 9.40.16

1.2.4.  Central Mission Controller

Name	**Value**
Documentation	`The entity (likely a SW routine running on the OBC) in charge of satellite supervision, fault detection and management and emergency management.`
ID	AC45
Visibility	public
Abstract	false
Leaf	false
Root	false
Business Model	false

Project Management	**Name**	**Value**
	Author	L.M. Reyneri
	Create Date Time	1-giu-2010 14.11.26
	Last Modified	21-nov-2014 18.25.31

1.2.4.1. Relationships

Unnamed Association		
To	**Name**	**Value**
	End Model Element	Get Sun SensorErrors
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false

	Leaf	false
	Project Management	**Name** / **Value** Author: Administrator Create Date Time: 1-giu-2010 14.29.44 Last Modified: 14-giu-2010 18.41.42
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: Administrator Create Date Time: 1-giu-2010 14.29.44 Last Modified: 25-nov-2014 9.40.16	

Unnamed Association		
To	**Name** / **Value** End Model Element: Detach Sun Sensor Provide Property Getter Method: false Provide Property Setter Method: false Multiplicity: Unspecified Visibility: Unspecified Aggregation Kind: None Navigable: Navigable Derived: false Derived Union: false Read Only: false Static: false Leaf: false	
	Project Management	**Name** / **Value** Author: Administrator Create Date Time: 18-mag-2010 15.41.35 Last Modified: 1-giu-2010 17.41.35
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	

Project Management	Name	Value
	Author	Administrator
	Create Date Time	18-mag-2010 15.41.35
	Last Modified	25-nov-2014 9.40.16

Unnamed Association

To	Name	Value
	End Model Element	Attach Sun Sensor
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	Name / Value
		Author: Administrator
		Create Date Time: 18-mag-2010 15.41.23
		Last Modified: 1-giu-2010 17.41.35

Abstract	false
Leaf	false
Visibility	Unspecified
Derived	false

Project Management	Name	Value
	Author	Administrator
	Create Date Time	18-mag-2010 15.41.23
	Last Modified	25-nov-2014 9.40.16

Unnamed Association

To	Name	Value
	End Model Element	Get Supply Currents

	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Type	Get Supply Currents
	Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.00.33
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.00.33 Last Modified / 11-nov-2014 11.51.00	

Unnamed Association		
To	**Name**	**Value**
	End Model Element	Get Supply Voltages
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false

	Read Only	false
	Static	false
	Leaf	false
	Type	Get Supply Voltages
	Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.19
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.19 Last Modified: 11-nov-2014 11.51.00	

Unnamed Association		
To	**Name**	**Value**
	End Model Element	Get Temperature
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Type	Get Temperature
	Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.07
Abstract	false	

Leaf	false
Visibility	Unspecified
Derived	false
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.00.07 Last Modified / 11-nov-2014 11.51.00

1.2.5. Get Windowed Image

Name	**Value**
Documentation	`Get raw image with reduced size in order to high resolution for computation of barycenter. This operation must do upon a possible tracking algorithm.`
ID	84
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 12.26.23 Last Modified / 25-nov-2014 9.40.16

1.2.5.1. Relationships

Unnamed Include	
To	Read Slave Data
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 12.34.37 Last Modified / 25-nov-2014 9.40.16

1.2.6. Slave Commands

Name	Value
Documentation	OBSOLETE: The Master actor sends a data-less command to the **System**. The Master can use up to 8 different Slave Commands, to issue as many commands to the system. It uses the Command Only use case, by issuing the (model element not found) through (model element not found), where **x** (0..7) identifies the message type; **x** does not identify the sequence in which commands are issued, but the (model element not found)-defined type of command. The (model element not found) can use as many message types he wants. This use case is optional, therefore the (model element not found) shall #define an identifier whose name is contained in the tagged value **define**. After command has been received and checked, the 1B45_Subsystem_Serial_Data_Bus calls the CommandInterpreter.interpret(command: t_Commands, error: t_LastError): void operation and passes the command as an argument. The (model element not found) shall then write his command interpretation routine inside the interpret(command: t_Commands, error: t_LastError): void operation.
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.27 Last Modified: 14-nov-2014 16.26.22

1.2.6.1. Relationships

Unnamed Include	
From	Take Picture Immediate
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 12.15.09 Last Modified: 25-nov-2014 9.40.16

1.2.6.2. Details

Name	Value
define	def_CMD_COMMAND

1.2.6.3. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_COMMAND
Tag Definition	define
Project Management	Name: Author — Value: lilian Name: Create Date Time — Value: 5-feb-2010 9.40.27 Name: Last Modified — Value: 3-giu-2010 14.26.27

1.2.6.4. Sub Diagrams

Name	Documentation
Module Commands	

1.2.7. Write Slave Data

Name	Value
Documentation	OBSOLETE: The Master actor sends up to 256B of Designer-defined data to the **System**. The Master can use up to 8 different Write Slave Data commands, to issue messages to as many different subsystems. It uses the Write Data use case, by issuing the (model element not found) through (model element not found) command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are written, but the Designer-defined type (therefore the destination subsystem). The Designer can use as many message types he wants. This use case is optional, therefore the Designer shall **#define** an identifier whose name is contained in thIs use case's tagged value **define**. After command has been received completely and checked, the 1B45_Subsystem_Serial_Data_Buswill allocate a buffer inside the Buffers class, then call the CommandInterpreter:interpret(command: t_Commands, error: t_LastError): void operation and pass the command as an argument. The Designer shall then write his command interpretation routine inside the interpret(command: t_Commands, error: t_LastError): void operation. The command interpreter shall access the data message from class Buffers

	with the following sequence: 1. Call Buffers.(model element not found), which returns a pointer to the buffer containing the message and its length (number of bytes in data section, excluding the command itself and other ancillary data. 2. Use the message from the above vector 3. When the message is not used anymore, release the buffer using Buffers.release(buffer: byte*): bool
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / lilian Create Date Time / 5-feb-2010 9.40.26 Last Modified / 14-nov-2014 16.26.22

1.2.7.1. Relationships

Unnamed Include	
From	Delete Image
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 12.10.15 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
From	Get Available Images
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 12.11.07 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
From	Set Trigger and Brightness
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 13.19.34 Last Modified: 25-nov-2014 9.40.16

1.2.7.2. Details

Name	Value
define	def_CMD_WRITE_DATA

1.2.7.3. Use Case Descriptions

Main			
Author	Administrator		
Date	Oct 28, 2008 5:14:28 PM		
Brief Description			
Preconditions			
Post-conditions			
Flow of Events		**Actor Input**	**System Response**

1.2.7.4. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_WRITE_DATA
Tag Definition	define
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.26 Last Modified: 3-giu-2010 14.26.27

1.2.7.5. Sub Diagrams

Name	Documentation
Write Module Data	

1.2.8. Compress Image

Name	Value

Documentation	`Once stored image in one dedicated memory, it's compressed after to have issued command CMD_COMPRESS_IMAGE that uses` Slave Commands `use case`
	
	`TBD`

ID	1112
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 15.29.25 Last Modified / 18-nov-2014 14.34.54

1.2.8.1. Relationships

Unnamed Generalization	
To	Read Slave Data
Substitutable	false
Visibility	Unspecified
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 12.08.59 Last Modified / 25-nov-2014 9.40.16

1.2.9. Configure Image Parameters

Name	Value
Documentation	Configures at compilation/integration time parameters like: 1. Image size of imager (first row (ROW_START: unsigned short)/column (COL_START: unsigned short) to be read out,WINDOWS_HEIGHT: unsigned short/ WINDOWS_WIDTH: unsigned short/, row/column flip) 2. Operating Mode (Slave/Master/Snapshot mode)by CHIP_CONTROL: unsigned short 3. Simultaneous or sequential mode 4. Integration time setting COARSE_SHUTTER_WIDTH: unsigned short in SNAPSHOT MODE 5. Enable High Dynamic Range (set in R0x0F[0]) and sets V1 /V2/V3/V4 levels) by means of V1_STEP: unsigned char, V2_STEP: unsigned char, V3_STEP: unsigned char and V4_STEP: unsigned char 6. READ MODE (normal or binning of rows/columns)by READ_MODE: unsigned short -> reduced image size snd altered frame rate (just for row bin up to increased by 4)such as pixel clock (just for column bin up to reduced by one-fourth) 7. Black calibration by means of BLACK_CALIBRATIOM: unsigned short , ANALOG_GAIN: unsigned short and VREF: unsigned short 8. Analog Gain ANALOG_GAIN: unsigned short and Digital Gain 9. Noise Correction NOISE_CORRECTION: unsigned short 10. Enable compading 12 -> 10 11. Enable AGC/AEG For each register there are two different options to be set: Context A e Context B
ID	1111
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value**

	Author	gabriele.defranciscis
	Create Date Time	10-nov-2014 15.29.02
	Last Modified	18-nov-2014 14.34.54

1.2.9.1. Relationships

Unnamed Association		
From	**Name**	**Value**
	End Model Element	Configurator
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Type	Configurator
	Project Management	**Name** / **Value**: Author: gabriele.defranciscis; Create Date Time: 14-nov-2014 12.18.01
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name**	**Value**
	Author	gabriele.defranciscis
	Create Date Time	14-nov-2014 12.18.01
	Last Modified	25-nov-2014 9.40.16

1.2.10. Delete Image

Name	Value

Documentation	This use case Delete Image allows to discard one image by means of CMD_DELETE_IMAGE followed by the following data: 1. buffer[0]: number of image to be deleted according to specific section of FLASH memory which has been used
ID	1115
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 15.30.34 Last Modified: 18-nov-2014 14.34.54

1.2.10.1. Relationships

Unnamed Include	
To	Write Slave Data
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 12.10.15 Last Modified: 25-nov-2014 9.40.16

1.2.11. Get Available Images

Name	**Value**
Documentation	Once stored everyone available images, OBC can issue CMD_GET_AVAILABLE_IMAGES command to request to image processor different images for additional purposes. As a result of this, image processor can send towards OBC the requirement bytes to be read Read Slave Data. This is managed by Get Available Images use case.

ID	1116
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 15.30.46 Last Modified: 18-nov-2014 14.34.54

1.2.11.1. Relationships

Unnamed Include	
To	Write Slave Data
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 12.11.07 Last Modified: 25-nov-2014 9.40.16

Unnamed Include	
To	Read Slave Data
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 15.45.00 Last Modified: 11-nov-2014 11.51.00

1.2.12. Get Raw Image

Name	**Value**
Documentation	`Get raw image on behalf of OBC for specific purposes.`

	Uses Read Slave Data use case by issuing command CMD_GET_RAW_IMAGE followed by the following
ID	1113
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 15.29.54 Last Modified / 18-nov-2014 14.34.54

1.2.12.1. Relationships

Unnamed Include	
To	Read Slave Data
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 15.44.39 Last Modified / 11-nov-2014 11.51.00

Unnamed Association	
From	**Name** / **Value** End Model Element / Central Attitude Controller Provide Property Getter Method / false Provide Property Setter Method / false Multiplicity / Unspecified Visibility / private Aggregation Kind / None Navigable / Navigable Derived / false Derived Union / false

	Read Only	false
	Static	false
	Leaf	false
	Type	Central Attitude Controller
	Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 13-nov-2014 15.13.41
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 13-nov-2014 15.13.41 Last Modified / 25-nov-2014 9.40.16	

1.2.13. Get Supply Currents

Name	Value
ID	1122
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.00.33 Last Modified / 11-nov-2014 11.51.00

1.2.13.1. Relationships

Unnamed Include	
To	Get Module Housekeeping
Visibility	Unspecified

Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.01.36 Last Modified: 11-nov-2014 11.51.00
Unnamed Association	
From	**Name** / **Value** End Model Element: Central Mission Controller Provide Property Getter Method: false Provide Property Setter Method: false Multiplicity: Unspecified Visibility: Unspecified Aggregation Kind: None Navigable: Navigable Derived: false Derived Union: false Read Only: false Static: false Leaf: false Type: Central Mission Controller Project Management: **Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.33
Abstract	false
Leaf	false
Visibility	Unspecified
Derived	false
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.33 Last Modified: 11-nov-2014 11.51.00

1.2.14. Get Supply Voltages

Name	Value

Documentation	Periodically measures and stores into housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector the following voltages: 1. 1.4V of ... into vector element HK_VOLTAGE_1V4 2. 2.5V
ID	1121
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified

Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	10-nov-2014 16.00.19
	Last Modified	11-nov-2014 11.51.00

1.2.14.1. Relationships

Unnamed Include	
To	Get Module Housekeeping
Visibility	Unspecified
Stereotypes	Include

Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	10-nov-2014 16.01.34
	Last Modified	11-nov-2014 11.51.00

Unnamed Association		
From	Name	Value
	End Model Element	Central Mission Controller
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable

	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Type	Central Mission Controller
	Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.19
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.19 Last Modified: 11-nov-2014 11.51.00	

1.2.15. Get Temperature

Name	Value
Documentation	`Returns the last acquired value of temperature.....`
ID	1120
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.00.07 Last Modified: 21-nov-2014 20.06.47

1.2.15.1. Relationships

<table>
<tr><th colspan="2">Unnamed Include</th></tr>
<tr><td>To</td><td>Get Module Housekeeping</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Stereotypes</td><td>Include</td></tr>
<tr><td>Project Management</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>gabriele.defranciscis</td></tr>
<tr><td>Create Date Time</td><td>10-nov-2014 16.01.25</td></tr>
<tr><td>Last Modified</td><td>11-nov-2014 11.51.00</td></tr>
</table>
</td></tr>
<tr><th colspan="2">Unnamed Association</th></tr>
<tr><td>From</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>End Model Element</td><td>Central Mission Controller</td></tr>
<tr><td>Provide Property Getter Method</td><td>false</td></tr>
<tr><td>Provide Property Setter Method</td><td>false</td></tr>
<tr><td>Multiplicity</td><td>Unspecified</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Aggregation Kind</td><td>None</td></tr>
<tr><td>Navigable</td><td>Navigable</td></tr>
<tr><td>Derived</td><td>false</td></tr>
<tr><td>Derived Union</td><td>false</td></tr>
<tr><td>Read Only</td><td>false</td></tr>
<tr><td>Static</td><td>false</td></tr>
<tr><td>Leaf</td><td>false</td></tr>
<tr><td>Type</td><td>Central Mission Controller</td></tr>
<tr><td>Project Management</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>gabriele.defranciscis</td></tr>
<tr><td>Create Date Time</td><td>10-nov-2014 16.00.07</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>Abstract</td><td>false</td></tr>
<tr><td>Leaf</td><td>false</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Derived</td><td>false</td></tr>
<tr><td>Project Management</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>gabriele.defranciscis</td></tr>
<tr><td>Create Date Time</td><td>10-nov-2014 16.00.07</td></tr>
<tr><td>Last Modified</td><td>11-nov-2014 11.51.00</td></tr>
</table>
</td></tr>
</table>

1.2.16. Set Trigger and Brightness

Name	Value
Documentation	Configures integration time for all subsequent picture acquisitions. Uses Write Slave Data use case by issuing command CMD_SET_TRIGGER followed by the following data: 1. buffer[0]: Integration time in number of rows And/or configures voltages for all subsequent picture acquisitions. Uses Write Slave Data use case by issuing command CMD_SET_BRIGHTNESS followed by the following data: 1. buffer[0]: light intensity in value of voltage for V1 2. buffer[1]: light intensity in value of voltage for V2 3. buffer[2]: light intensity in value of voltage for V3 4. buffer[3]: light intensity in value of voltage for V4 (seves as an antiblooming for photodetector) 5. buffer[4]: ADC gain (by default is X1) 6. buffer[5]: ADC voltage references (by default is 1.4V) Light intensities work approximately as a reciprocal of the partial exposure time.
ID	1119
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 15.32.45 Last Modified: 18-nov-2014 14.34.54

1.2.16.1. Relationships

Unnamed Include	
To	Write Slave Data
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 14-nov-2014 13.19.34 Last Modified: 25-nov-2014 9.40.16

1.2.17. Take Picture Immediate

Name	Value

Documentation	`By means of CMD_TAKE_PICTURE_IMMEDIATE command, it allows image processor to take one picture without any delays since probably spacecraft is in desirable situation to do it.`
	
	`This command enables any kind of routine to acquire one frame and store it in SDRAM temporarily.`

ID	1117
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 15.31.13 Last Modified: 18-nov-2014 14.34.54

1.2.17.1. Relationships

Unnamed Include	
To	Slave Commands
Visibility	Unspecified
Stereotypes	Include

Project Management	**Name**	**Value**
	Author	gabriele.defranciscis
	Create Date Time	14-nov-2014 12.15.09
	Last Modified	25-nov-2014 9.40.16
Unnamed Include		
From	Get Sun Direction	
Visibility	Unspecified	
Stereotypes	Include	
Project Management	**Name**	**Value**
	Author	gabriele.defranciscis
	Create Date Time	10-nov-2014 16.14.52
	Last Modified	25-nov-2014 9.40.16

1.2.18. Read Slave Data

Name	Value
Documentation	OBSOLETE: The Master actor reads up to 256B of Designer-defined data from the **System**. The Master can use up to 8 different Read Slave Data commands, to read messages from as many different subsystems. It uses the Read Data use case, by issuing a (model element not found) through (model element not found) command, where **x** (0..7) identifies the message type; **x** does not identify the sequence in which messages are written, but the Designer-defined type (therefore the source subsystem). The Designer can use as many message types he wants. This use case is optional, therefore the Designer shall **#define** an identifier whose name is contained in the tagged value **define**. The message to be read must already be present into a buffer inside the Buffers class before the Master actor starts this use case, otherwise this use case returns unpredictable results and sets (model element not found). this can be accomplished by: 1. Call Buffers.lock(command: t_Commands): byte*, which returns a pointer to an empty buffer available for the message (buffer storage is already allocated). 2. Fill up the buffer with the message data he wants 3. As soon as the message is complete, declare that the buffer is ready and specify its length by using Buffers.ready(command: t_Commands, length: ushort): bool operation. All the above operations shall be done before the Master issues the Read Slave Data command. Once the command is received and checked, the 1B45_Subsystem_Serial_Data_Buswill immediately send the message back to the Master, then it calls the CommandInterpreter:interpret(command: t_Commands, error: t_LastError): void operation and pass the command as

	an argument. The Designer shall then write his command interpretation routine inside the interpret(command: t_Commands, error: t_LastError): void operation. The Designer shall decide what to do once the message has been read. Either: • if the <<optional>> (model element not found) use case is NOT implemented, the buffer is automatically released by the 1B45_Subsystem_Serial_Data_Bus, before the call to the interpret(command: t_Commands, error: t_LastError): void..Since then, the Designer can prepare data for the next Read Slave Data command, whenever he likes. • if the <<optional>> (model element not found) use case is implemented: the 1B45_Subsystem_Serial_Data_Busleaves data in the buffer and keeps it available for further reading, in case the Master does not receive it properly. In this case, the Master, as soon as it properly receives data, shall use the (model element not found) use case, with which the 1B45_Subsystem_Serial_Data_Busautomatically releases the buffer , before calling the interpret(command: t_Commands, error: t_LastError): void operation..Since then, the Designer can prepare data for the next Read Slave Data command, whenever he likes. To decide for either the former or the latter case, the Designer shall implement (respectively, not implement) the (model element not found) use case.
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase
Business Model	false
Status	Identify
Rank	Unspecified

Project Management

Name	Value
Author	lilian
Create Date Time	5-feb-2010 9.40.26
Last Modified	14-nov-2014 16.26.22

1.2.18.1. Relationships

Unnamed Generalization

From	Compress Image
Substitutable	false
Visibility	Unspecified

Project Management

Name	Value
Author	gabriele.defranciscis
Create Date Time	14-nov-2014 12.08.59
Last Modified	25-nov-2014 9.40.16

Unnamed Include	
From	Get Windowed Image
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 12.34.37 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
From	Get Available Images
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 15.45.00 Last Modified / 11-nov-2014 11.51.00

Unnamed Include	
From	Get Raw Image
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 15.44.39 Last Modified / 11-nov-2014 11.51.00

1.2.18.2. Details

Name	Value
define	def_CMD_READ_DATA

1.2.18.3. Use Case Descriptions

Main	
Author	Administrator
Date	Mar 23, 2009 10:53:39 PM

Brief Description			
Preconditions			
Post-conditions			
Flow of Events		**Actor Input**	**System Response**

1.2.18.4. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_READ_DATA
Tag Definition	define
Project Management	

Name	**Value**
Author	lilian
Create Date Time	5-feb-2010 9.40.26
Last Modified	3-giu-2010 14.26.27

1.2.18.5. Sub Diagrams

Name	**Documentation**
Read Module Data	

1.2.19. Module Housekeeping

Name	**Value**
Documentation	An autonomous function which periodically samples (with period SAMPLETIME), calibrates and stores a number of Designer-defined housekeeping data. Values are stored as ushort words, with Designer-defined resolution (at most 16 bits), in the housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector. The Designer shall define a list of parameters to be acquired containing at least, for each parameter: • type of parameter and location of sensor • resolution (number of bits and signed/unsigned data type) • scale factor (namely, number of physical units per each unit of stored parameter) • offset (namely, real value corresponding to a stored 0 value) • sample rate (samples/s) and optional analog/digital filtering • index of data into the internal housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector (starting

	from 0). The length of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector shall be at most 128 short. This function also optionally computes statistics (min, max and average values) of a subset of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector; namely, LENGTH_STATISTICS elements starting from index FIRST_STATISTICS. The average is approximated by low-pass filtering, such that, at every new housekeeping cycle, the new filtered value will be: **y(t) = (short) (((long) y(t-1) * (FILTER_DEN - FILTER_NUM)) / FILTER_DEN) + (short) (((long) in * FILTER_NUM) / FILTER_DEN)** where **y(t)** is the filtered value, while **in** is the last sampled value; both of type short, while FILTER_NUM and FILTER_DEN define filter's time constant (FILTER_NUM must be smaller than FILTER_DEN, under Designer's responsibility). The time constant of the filter will therefore be: **τ = SAMPLETIME * (FILTER_DEN / FILTER_NUM)** where SAMPLETIME is the the period of housekeeping cycle (see Module Housekeeping), while the bandwidth is: **BW = 1 / (2 * π * SAMPLETIME * (FILTER_DEN / FILTER_NUM))** Statistics are stored into vector statistics: HK_REDUNDANCY[3][LENGTH_STATISTICS]. This function also computes history of a subset of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector; namely, the last DEPTH_HISTORY samples of LENGTH_HISTORY elements starting from index FIRST_HISTORY. The history is stored into vector history: HK_REDUNDANCY[DEPTH_HISTORY][LENGTH_HISTORY]. The Calibrate Housekeeping use case allows to calibrate all (or a number of) sensors by applying appropriate offset and gain corrections.. The Get Module Housekeeping use case returns the last acquired housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector. The Get Module History use case returns history of the last DEPTH_HISTORY samples of the housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector (or a part of it, namely LENGTH_HISTORY elements starting from FIRST_HISTORY), for at most 256B of data. The Designer shall define the list of parameters for which the history has to be kept and the number of samples which have to be saved. The Get Module Statistics use case returns min, max and a low-pass filtering of the housekeeping vector (or a TBD part of it). Low-pass filtering time constant is also TBD. Statistics shall be computed starting from the last Reset Statistics. The Designer shall define the list of parameters for which the statistics have to be kept.

	The Reset Module Statistics use case resets the statistics for the housekeeping vector. This use case is optional, therefore the Configurator shall #define an identifier whose name is contained in the tagged value define.
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase, parametric
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / lilian Create Date Time / 5-feb-2010 9.40.26 Last Modified / 14-nov-2014 16.26.21

1.2.19.1. Relationships

Unnamed Include	
From	Housekeeping
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.36.03 Last Modified / 25-nov-2014 9.40.16

1.2.19.2. Details

Name	**Value**
define	def_CMD_GET_HOUSEKEEPING

1.2.19.3. Use Case Descriptions

Main	
Super Use Case	
Author	Administrator
Date	Jan 22, 2009 1:15:03 PM

Brief Description			
Preconditions			
Post-conditions			
Flow of Events		**Actor Input**	**System Response**

1.2.19.4. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_GET_HOUSEKEEPING
Tag Definition	define
Project Management	**Name** / **Value** Author / lilian Create Date Time / 5-feb-2010 9.40.26 Last Modified / 14-mag-2010 7.57.30

parameters	
Type	Multi-line Text
Multiplicity	Unspecified
Tag Definition	parameters
Tag Definition Stereotype	parametric
Project Management	**Name** / **Value** Author / lreyneri Create Date Time / 29-gen-2013 18.09.39 Last Modified / 17-apr-2014 9.46.55

1.2.19.5. Sub Diagrams

Name	**Documentation**
Module Housekeeping	

1.2.20. Get Module Status

Name	**Value**
Documentation	`It returns to the OBC the status information (statusRegister: CS_REDUNDANCY[LENGTH_STATUS]) of one of many spacecraft Tiles.`

	This use case first uses the Read Data use case by issuing the CMD_GET_STATUS command with the address of the desired Tile, which returns its statusRegister: CS_REDUNDANCY[LENGTH_STATUS] to the Master. Using this use case also also clears the Error Indicator signal (if present).
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified

Project Management	Name	Value
	Author	lilian
	Create Date Time	5-feb-2010 9.40.26
	Last Modified	14-nov-2014 16.26.22

1.2.20.1. Relationships

Unnamed Include	
From	Get Sun SensorErrors
Visibility	Unspecified
Stereotypes	Include

Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	17-giu-2014 16.35.13
	Last Modified	25-nov-2014 9.40.16

1.2.20.2. Use Case Descriptions

Main	
Author	Administrator
Date	Dec 15, 2008 5:27:37 PM
Brief Description	
Preconditions	
Post-conditions	

Flow of Events		Actor Input	System Response

1.2.20.3. Sub Diagrams

Name	Documentation
Get Module Status	

1.2.21. Get Sun Intensity

Name	Value
Documentation	`Get Sun Intensity for later to filter possible bright sources as stars.`
ID	82
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.32.49 Last Modified / 25-nov-2014 9.40.16

1.2.21.1. Relationships

Unnamed Include	
To	Get Module Housekeeping
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 13.20.00 Last Modified / 25-nov-2014 9.40.16
Unnamed Association	
From	**Name** / **Value**

	End Model Element	Central Attitude Controller
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Type	Central Attitude Controller
	Project Management	**Name** / **Value**: Author: gabriele.defranciscis; Create Date Time: 17-giu-2014 16.32.49
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value**: Author: gabriele.defranciscis; Create Date Time: 17-giu-2014 16.32.49; Last Modified: 25-nov-2014 9.40.16	

1.2.22. receiveApproxVector

Name	Value
Documentation	Actor sends x, y, and z coordinates of the unitary vector in J2000 coordinates, derived from the measurements of the sun sensor and/or magnetic field of the earth.
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase, TBD
Business Model	false
Status	Identify

Rank	Unspecified	
Project Management	**Name**	**Value**
	Last Modified	25-nov-2014 9.40.16

1.2.23. receiveApproximationAvailability

Name	**Value**	
Documentation	Receives the availability or not, of the initial approximation of attitude vector. If not available, the sensor performs a Lost in Space measurement.	
Abstract	false	
Leaf	false	
Root	false	
Stereotypes	TBD, UseCase	
Business Model	false	
Status	Identify	
Rank	Unspecified	
Project Management	**Name**	**Value**
	Last Modified	25-nov-2014 9.40.16

1.2.24. Calibrate Housekeeping

Name	**Value**
Documentation	Writes into the Slave calibration data for housekeeping sensors (if any). For each sensor, two values are to be supplied: an **unsigned t_sensor offset** and an **unsigned t_sensor gain_correction**. Both parameters are of **unsigned t_sensor** type. Values are written into FLASH memory. The **Module Housekeeping** use case shall then write into the **housekeeping** vector: **for (i=0; i<LENGTH_CORRECTION); i++)** **housekeeping[i] = (acquired_value[i] - offset[i]) * (gain_correction[i] / (max(t_sensor)/2));** where: **t_sensor housekeeping[i]** is the i-th value written in the housekeeping vector; **t_sensor acquired_value[i]** is the i-th value (unsigned integer) acquired from the i-th sensor

	t_sensor offset[i] is the i-th offset value **t_sensor gain_correction[i]** is the i-th gain_correction value **max(t_sensor)** is the max value which can be expressed by an (unsigned t_sensor) Only the first **LENGTH_CALIBRATION** sensors (from 0 to **LENGTH_CALIBRATION-1**) are calibrated. For all the other sensors, the acquired value is written into the **housekeeping** vector directly. Also if this use case is not implemented, the acquired value is written into the **housekeeping** vector directly. The default values for the offset and gain_correction parameters are set by the **Tester** (in the factory) after appropriate calibration. These parameters can be later updated by the Master**.** This use case is different from the **Set Module Configuration** and the **Configure Module** use cases: **Configure Module** allows a compile-time configuration, which cannot be changed during the life of the **System**; **Set Module Configuration** allows run-time configuration changes (if foreseen) but these do without affecting sensor calibration; **Calibrate Housekeeping** allows run-time (or post-fabrication) changes of sensor calibration parameters; This use case is optional, therefore the **Configurator** shall #define an identifier whose name is contained in the tagged value **define**.
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase, parametric
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.27 Last Modified: 13-nov-2014 18.13.06

1.2.24.1. Relationships

Unnamed Include	
From	On Ground Calibration
Visibility	Unspecified
Stereotypes	Include

Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	17-giu-2014 16.35.34
	Last Modified	25-nov-2014 9.40.16

Unnamed Include		
From	In Flight Calibration	
Visibility	Unspecified	
Stereotypes	Include	
Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	17-giu-2014 16.35.37
	Last Modified	25-nov-2014 9.40.16

1.2.24.2. Details

Name	Value
define	def_CMD_CALIBRATE_HOUSEKEEPING

1.2.24.3. Tagged Values

define		
Type	Text	
Multiplicity	Unspecified	
Value	def_CMD_CALIBRATE_HOUSEKEEPING	
Tag Definition	define	
Project Management	Name	Value
	Author	lilian
	Create Date Time	5-feb-2010 9.40.27
	Last Modified	14-mag-2010 7.57.30

parameters		
Type	Multi-line Text	
Multiplicity	Unspecified	
Tag Definition	parameters	
Tag Definition Stereotype	parametric	
Project Management	Name	Value

	Author	lreyneri
	Create Date Time	29-gen-2013 18.09.39
	Last Modified	30-gen-2013 11.43.01

1.2.25. Get Module Housekeeping

Name	Value
Documentation	Returns last measured housekeeping data (see use case Module Housekeeping for details). The Master shall uses the Read Data use case by issuing the CMD_GET_HOUSEKEEPING command. The Slave shall assemble ALL last saved housekeeping data into the response message and return them to the Master. No consistency is guaranteed between sampling time of different housekeeping data; it may therefore happen that some values have been just sampled, while others may be several seconds old, depending on sampling rate of Module Housekeeping. This use case is optional; refer to the Optional Use Cases section for further details. This use case is also parametric; its parameters are the following: • LENGTH_HOUSEKEEPING, namely the number of elements (different sensors, measurements, other data) that will be stored into the housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] vector.
Transit To	def_CMD_GET_HOUSEKEEPING
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase, parametric
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.26 Last Modified: 14-nov-2014 16.26.22

1.2.25.1. Relationships

Unnamed Include	
From	Get Sun Intensity
Visibility	Unspecified

Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 14-nov-2014 13.20.00 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
From	Get Sun Direction
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 13-nov-2014 15.35.56 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
From	Get Supply Currents
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.01.36 Last Modified / 11-nov-2014 11.51.00

Unnamed Include	
From	Get Supply Voltages
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.01.34 Last Modified / 11-nov-2014 11.51.00

Unnamed Include	
From	Get Temperature
Visibility	Unspecified

Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 10-nov-2014 16.01.25 Last Modified: 11-nov-2014 11.51.00

1.2.25.2. Details

Name	Value
define	def_CMD_GET_HOUSEKEEPING

1.2.25.3. Use Case Descriptions

Main		
Author	Administrator	
Date	Oct 23, 2008 4:26:46 PM	
Brief Description		
Preconditions		
Post-conditions		
Flow of Events	**Actor Input**	**System Response**

1.2.25.4. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_GET_HOUSEKEEPING
Tag Definition	define
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.26 Last Modified: 5-dic-2010 13.30.53

parameters	
Type	Multi-line Text
Multiplicity	Unspecified
Tag Definition	parameters

Tag Definition Stereotype	parametric
Project Management	**Name** / **Value** Author: leonardo.reyneri Create Date Time: 31-mar-2014 18.26.24 Last Modified: 17-apr-2014 9.46.55

1.2.25.5. Sub Diagrams

Name	Documentation
Get Module Housekeeping	

1.2.26. Reset Module Configuration

Name	Value
Documentation	Resets to 0 some bits of the internal configuration word (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) of one of many spacecraft Tiles. The OBC shall send to the Tile as many bits as are in its configuration word configRegister: CS_REDUNDANCY[LENGTH_CONFIG]. Each bit which is 1 in the message will reset to zero the corresponding bit in the Tile configuration word. The other bits are left unchanged. This use case sends configuration bits by means of the Write Data use case, by issuing the CMD_RESET_CONFIGURATION command with the address of the desired Tile. By no means, configRegister[0] can be written, as this contains the Designer-defined HW/SW version. This use case differs from a set of related use cases; in particular: • the use case Configure Module allows a compile-time configuration. • the use case Reset Module Configuration resets to 0 a few bits in the configuration word. It cannot set any bit to 1. • the use case Set Module Configuration sets to 1 a few bits in the configuration word. It cannot reset any bit to 0. • the use case Write Module Configuration writes (either 0 or 1) all the bits of the configuration word. This use case is optional, therefore the Configurator shall #define inside file platform.h an identifier whose name is contained in the tagged value define.
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase
Business Model	false

Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / lilian Create Date Time / 5-feb-2010 9.40.27 Last Modified / 6-mag-2013 15.53.22

1.2.26.1. Relationships

Unnamed Include	
From	Detach Sun Sensor
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.34.28 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
From	Disable Sun Sensor
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.34.32 Last Modified / 25-nov-2014 9.40.16

1.2.26.2. Details

Name	**Value**
define	def_CMD_SET_CONFIGURATION

1.2.26.3. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_SET_CONFIGURATION

Tag Definition	define
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.27 Last Modified: 14-mag-2010 7.57.30

1.2.27. Set Module Configuration

Name	Value
Documentation	Sets to 1 some bits of the internal configuration word (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) of one of many spacecraft Tiles. The OBC shall send to the Tile as many bits as are in its configuration word configRegister: CS_REDUNDANCY[LENGTH_CONFIG]. Each bit which is 1 in the message will set to 1 the corresponding bit in the Tile configuration word. The other bits are left unchanged. This use case sends configuration bits by means of the Write Data use case, by issuing the CMD_SET_CONFIGURATION command with the address of the desired Tile. By no means, configRegister[0] can be written, as this contains the Designer-defined HW/SW version. This use case differs from a set of related use cases; in particular: • the use case Configure Module allows a compile-time configuration. • the use case Reset Module Configuration resets to 0 a few bits in the configuration word. It cannot set any bit to 1. • the use case Set Module Configuration sets to 1 a few bits in the configuration word. It cannot reset any bit to 0. • the use case Write Module Configuration writes (either 0 or 1) all the bits of the configuration word. This use case is optional, therefore the Configurator shall #define inside file platform.h an identifier whose name is contained in the tagged value define.
Abstract	false

Leaf	false
Root	false
Stereotypes	optional, UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.26 Last Modified: 14-nov-2014 10.04.26

1.2.27.1. Relationships

Unnamed Include	
From	Attach Sun Sensor
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 17-giu-2014 16.34.15 Last Modified: 25-nov-2014 9.40.16

Unnamed Include	
From	Enable Sun Sensor
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 17-giu-2014 16.34.34 Last Modified: 25-nov-2014 9.40.16

1.2.27.2. Details

Name	**Value**
define	def_CMD_SET_CONFIGURATION

1.2.27.3. Tagged Values

define	
Type	Text
Multiplicity	Unspecified
Value	def_CMD_SET_CONFIGURATION
Tag Definition	define
Project Management	

Name	Value
Author	lilian
Create Date Time	5-feb-2010 9.40.26
Last Modified	14-mag-2012 11.28.05

1.2.27.4. Sub Diagrams

Name	Documentation
Set Module Configuration	

1.2.28. Get Module Configuration

Name	Value
Documentation	It returns to the OBC the current configuration word (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) of one of many spacecraft Tiles. The configuration word was last written by the OBC, or the default for the Tile if not yet modified by the OBC. This use case first uses the Read Data use case by issuing the CMD_GET_CONFIGURATION command with the address of the desired Tile, which then returns its configRegister: CS_REDUNDANCY[LENGTH_CONFIG] to the Master. This use case is optional, therefore the Configurator shall #define inside file platform.h an identifier whose name is contained in the tagged value define.
Abstract	false
Leaf	false
Root	false
Stereotypes	optional, UseCase
Business Model	false
Status	Identify
Rank	Unspecified

Project Management	**Name**	**Value**
	Author	lilian
	Create Date Time	5-feb-2010 9.40.27
	Last Modified	14-nov-2014 10.04.26

1.2.28.1. Details

Name	Value
define	def_CMD_GET_CONFIGURATION

1.2.28.2. Use Case Descriptions

Main			
Author	Administrator		
Date	Mar 23, 2009 10:57:56 PM		
Brief Description			
Preconditions			
Post-conditions			
Flow of Events		**Actor Input**	**System Response**

1.2.28.3. Tagged Values

define		
Type	Text	
Multiplicity	Unspecified	
Value	def_CMD_GET_CONFIGURATION	
Tag Definition	define	
Project Management	**Name**	**Value**
	Author	lilian
	Create Date Time	5-feb-2010 9.40.27
	Last Modified	29-mag-2012 17.59.51

1.2.28.4. Sub Diagrams

Name	Documentation
 Get Module Configuration	

1.2.29. Housekeeping

Name	Value
ID	57
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 9-giu-2010 8.08.01 Last Modified / 25-nov-2014 9.40.16

1.2.29.1. Relationships

Unnamed Include	
To	Module Housekeeping
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.36.03 Last Modified / 25-nov-2014 9.40.16

1.2.30. Supervision

Name	Value
ID	68
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified

Project Management	Name	Value
	Author	Administrator
	Create Date Time	1-giu-2010 14.29.50
	Last Modified	25-nov-2014 9.40.16

1.2.31. Get Sun SensorErrors

Name	Value
Documentation	Returns the complete list of errors which may afflict the sensors of Subsystem and are detected by Supervision Subsystem use case. Errors are listed by type as bits composing the error word (model element not found).housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING][(model element not found)], respecting the following order: ON_CURR_EXCESS->when image processor is operating, processor current exceeds maximum allowed value ON_CURR_LACK->when image processor is operating, processor current is less than minimum functional value OFF_CURR_EXCESS ->when image processor is disabled, there is a significant current which supply processor ERR_SENS_ENABLE ->error which stucks bit (model element not found) at 'true' ERR_SENS_STATUS->error which stucks bit (model element not found) at 'true' ERR_SENS_ATTACH->error which stucks bit (model element not found) at 'true' ERR_IM_PROC_ENABLE->error which stucks bit (model element not found) at 'true' ERR_IM_PROC_ATTACH->error which stucks bit (model element not found) at 'true' LOW_PDB_VOLT->error in Power Distribution Bus Voltage: voltage is below minimum critical value This use case makes use of Get Module Housekeeping use case of Bk1B45_Slave package.
ID	65
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified

Project Management	Name	Value
	Author	Administrator
	Create Date Time	1-giu-2010 14.29.43
	Last Modified	25-nov-2014 9.40.16

1.2.31.1. Relationships

Unnamed Include		
To	Get Module Status	
Visibility	Unspecified	
Stereotypes	Include	
Project Management	Name	Value
	Author	gabriele.defranciscis
	Create Date Time	17-giu-2014 16.35.13
	Last Modified	25-nov-2014 9.40.16

Unnamed Association		
From	Name	Value
	End Model Element	Central Mission Controller
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	Unspecified
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	Name / Value
		Author: Administrator
		Create Date Time: 1-giu-2010 14.29.44
		Last Modified: 14-giu-2010 18.41.42
Abstract	false	
Leaf	false	
Visibility	Unspecified	

Derived	false
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 1-giu-2010 14.29.44 Last Modified / 25-nov-2014 9.40.16

1.2.32. Detach Sun Sensor

Name	Value
Documentation	Flags sun sensor as unavailable to the user. Sensor is considered "detached" either if it is NOT physically present on the tile (flag (model element not found) of (model element not found)) or it has not been attached (see use case Attach Subsytem). If the sensor is either detached or disabled (see use case Disable Subsystem), no further action can be carried on. This Use Case makes use of Reset Module Configuration use case of 1B45 package, with bit ((model element not found)) of configuration word ((model element not found)). The difference between Detach Subsystem and Disable Subsystem is that the former prevents the sensore from being enabled, therefore it disables any further action on the sensor, including Enable Subsystem. The Detach Subsystem use case is intended only in rare occasions, to permanently disable a faulty sensor. Since the actor can know if the sensor is attached (see use case Has Subsystem), the ARAMIS-level AOCS algorithms can be adapted consequently.
ID	59
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 18-mag-2010 15.39.44 Last Modified / 25-nov-2014 9.40.16

1.2.32.1. Relationships

<table>
<tr><th colspan="2">Unnamed Include</th></tr>
<tr><td>To</td><td>Reset Module Configuration</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Stereotypes</td><td>Include</td></tr>
<tr><td>Project Management</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>gabriele.defranciscis</td></tr>
<tr><td>Create Date Time</td><td>17-giu-2014 16.34.28</td></tr>
<tr><td>Last Modified</td><td>25-nov-2014 9.40.16</td></tr>
</table>
</td></tr>
<tr><th colspan="2">Unnamed Association</th></tr>
<tr><td>From</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>End Model Element</td><td>Central Mission Controller</td></tr>
<tr><td>Provide Property Getter Method</td><td>false</td></tr>
<tr><td>Provide Property Setter Method</td><td>false</td></tr>
<tr><td>Multiplicity</td><td>Unspecified</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Aggregation Kind</td><td>None</td></tr>
<tr><td>Navigable</td><td>Navigable</td></tr>
<tr><td>Derived</td><td>false</td></tr>
<tr><td>Derived Union</td><td>false</td></tr>
<tr><td>Read Only</td><td>false</td></tr>
<tr><td>Static</td><td>false</td></tr>
<tr><td>Leaf</td><td>false</td></tr>
<tr><td>Project Management</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>Administrator</td></tr>
<tr><td>Create Date Time</td><td>18-mag-2010 15.41.35</td></tr>
<tr><td>Last Modified</td><td>1-giu-2010 17.41.35</td></tr>
</table>
</td></tr>
</table>
</td></tr>
<tr><td>Abstract</td><td>false</td></tr>
<tr><td>Leaf</td><td>false</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Derived</td><td>false</td></tr>
<tr><td>Project Management</td><td>
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>Administrator</td></tr>
<tr><td>Create Date Time</td><td>18-mag-2010 15.41.35</td></tr>
<tr><td>Last Modified</td><td>25-nov-2014 9.40.16</td></tr>
</table>
</td></tr>
</table>

1.2.33. Attach Sun Sensor

Name	Value
Documentation	Sun sensor as available to the user. Sensor is considered "attached" if it is physically present on the tile (flag (model element not found) of (model element not found)) and it has not been detached (see use case Detach Subsystem). If the sensor is attached and enabled (see use case Enable Subsystem), any further action of image processor will be carried on. This Use Case makes use of Set Module Configuration use case of 1B45 package, with bit (model element not found) of configuration word ((model element not found)) The difference between Attach Subsytem and Enable Subsystem is that the former has effects only if the sensor is physically present on the tile, while the latter has effects only if the sensor is attached, therefore the former enables the latter. The Attach Subsytem use case is intended only in rare occasions, to recover from an erroneous Detach Subsystem operation.
ID	58
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Discuss
Rank	Unspecified
Project Management	**Name** / **Value** Author: Administrator Create Date Time: 18-mag-2010 15.39.10 Last Modified: 25-nov-2014 9.40.16

1.2.33.1. Relationships

Unnamed Include	
To	Set Module Configuration
Visibility	Unspecified
Stereotypes	Include

<table>
<tr><td>Project Management</td><td><table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>gabriele.defranciscis</td></tr>
<tr><td>Create Date Time</td><td>17-giu-2014 16.34.15</td></tr>
<tr><td>Last Modified</td><td>25-nov-2014 9.40.16</td></tr>
</table></td></tr>
<tr><th colspan="2">Unnamed Association</th></tr>
<tr><td>From</td><td><table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>End Model Element</td><td>  Central Mission Controller</td></tr>
<tr><td>Provide Property Getter Method</td><td>false</td></tr>
<tr><td>Provide Property Setter Method</td><td>false</td></tr>
<tr><td>Multiplicity</td><td>Unspecified</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Aggregation Kind</td><td>None</td></tr>
<tr><td>Navigable</td><td>Navigable</td></tr>
<tr><td>Derived</td><td>false</td></tr>
<tr><td>Derived Union</td><td>false</td></tr>
<tr><td>Read Only</td><td>false</td></tr>
<tr><td>Static</td><td>false</td></tr>
<tr><td>Leaf</td><td>false</td></tr>
<tr><td>Project Management</td><td><table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>Administrator</td></tr>
<tr><td>Create Date Time</td><td>18-mag-2010 15.41.23</td></tr>
<tr><td>Last Modified</td><td>1-giu-2010 17.41.35</td></tr>
</table></td></tr>
</table></td></tr>
<tr><td>Abstract</td><td>false</td></tr>
<tr><td>Leaf</td><td>false</td></tr>
<tr><td>Visibility</td><td>Unspecified</td></tr>
<tr><td>Derived</td><td>false</td></tr>
<tr><td>Project Management</td><td><table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>Author</td><td>Administrator</td></tr>
<tr><td>Create Date Time</td><td>18-mag-2010 15.41.23</td></tr>
<tr><td>Last Modified</td><td>25-nov-2014 9.40.16</td></tr>
</table></td></tr>
</table>

1.2.34. Enable Sun Sensor

Name	Value
Documentation	Enables sun sensor(STANDBY signal as low). Any further action of sensor will be carried on regularly. By default the sensor is disabled.

	When enabled, the sensor shall be active only when requested by the appropriate use cases (see (model element not found)). When enabled but not active, power consumption should be as little as possible. The Master shall set (model element not found) bit of Configuration Word (model element not found) using Set Module Configuration use case of Bk1B45_Slave package.
ID	61
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 25-nov-2014 9.40.16

1.2.34.1. Relationships

Unnamed Include	
To	Set Module Configuration
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 17-giu-2014 16.34.34 Last Modified: 25-nov-2014 9.40.16
Unnamed Association	
From	**Name** / **Value** End Model Element: Central Attitude Controller Provide Property Getter Method: false Provide Property Setter Method: false Multiplicity: Unspecified Visibility: Unspecified Aggregation Kind: None

	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 1-giu-2010 17.41.35
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 25-nov-2014 9.40.16	

1.2.34.2. Comments

Better using Set Module Configuration	
Documentation	Better using **Set Module Configuration** use case. Define one bit of configuration for enabling disabling the wheel.
Author	L.M. Reyneri
Date Time	Sep 25, 2009 5:21:47 PM
Project Management	**Name** / **Value**

1.2.35. Disable Sun Sensor

Name	**Value**
Documentation	Disables sun sensor (STANDBY signal as HIGH) . Such condition allows to sensor of going into standby mode and then it completes the current frame before disabling the digital logic, internal clocks and analog power enable signal. Any further action of sensor will be disregarded. If the sensor is in process of getting raw image while the Disable Subsystem use case is applied, the sensor is turned off upon it acquires frame.. When disabled, the sensor shall draw negligible power from the Power Bus. The **Master** shall reset (model element not found) bit of Configuration Word

	(model element not found) using Reset Module Configuration use case of Bk1B45_Slave package. The default is disabled.
ID	60
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author: Administrator Create Date Time: 19-mar-2010 16.40.11 Last Modified: 25-nov-2014 9.40.16

1.2.35.1. Relationships

Unnamed Include	
To	Reset Module Configuration
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author: gabriele.defranciscis Create Date Time: 17-giu-2014 16.34.32 Last Modified: 25-nov-2014 9.40.16

Unnamed Association	
From	**Name** / **Value** End Model Element: Central Attitude Controller Provide Property Getter Method: false Provide Property Setter Method: false Multiplicity: Unspecified Visibility: Unspecified Aggregation Kind: None Navigable: Navigable Derived: false

	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 1-giu-2010 17.41.35
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.08 Last Modified: 25-nov-2014 9.40.16	

1.2.35.2. Comments

Better using Set Module Configuration	
Documentation	Better using **Set Module Configuration** use case. Define one bit of configuration for enabling disabling the wheel.
Author	L.M. Reyneri
Date Time	Sep 25, 2009 5:22:48 PM
Project Management	**Name** / **Value**

1.2.36. In Flight Calibration

Name	**Value**
Documentation	Allows to change value of coil area, offset and gain of magnetometer, using three signed 16-bits integers. Units are: TBD, TBD, TBD. Uses Calibrate Housekeeping use case of Bk1B45_Slave package, with: • magnetometer gain on x-axis in field **magnetometer_gain_X**, • magnetometer gain on y-axis in field **magnetometer_gain_Y**, • magnetometer offset on x-axis in field **magnetometer_offset_X**, • magnetometer offset on y-axis in field **magnetometer_offset_Y**, • coil area in field **coil_area**.
ID	67
Abstract	false

Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 15-mar-2010 16.28.35 Last Modified / 25-nov-2014 9.40.16

1.2.36.1. Extension Points

On Ground Calibration	
Project Management	**Name** / **Value** Author / lreyneri Create Date Time / 9-giu-2011 17.37.45 Last Modified / 9-giu-2011 17.47.05

1.2.36.2. Relationships

Unnamed Extend	
To	On Ground Calibration
Visibility	Unspecified
Extension Point	On Ground Calibration
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 9-giu-2011 17.37.45 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
To	Calibrate Housekeeping
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.35.37

	Last Modified	25-nov-2014 9.40.16

Unnamed Association		
From	**Name**	**Value**
	End Model Element	Calibrator
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value**: Author: dmasera; Create Date Time: 15-mar-2010 17.13.08; Last Modified: 9-giu-2011 17.47.05
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name**	**Value**
	Author	dmasera
	Create Date Time	15-mar-2010 17.13.08
	Last Modified	25-nov-2014 9.40.16

1.2.36.3. Comments

do we need it?	
Documentation	and how to implement it?
Author	aa.vv.
Date Time	Jun 17, 2008 11:15:20 AM
Project Management	**Name** / **Value**

1.2.37. On Ground Calibration

Name	Value
ID	66
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 19-mar-2010 17.32.02 Last Modified / 25-nov-2014 9.40.16

1.2.37.1. Relationships

Unnamed Include	
To	Calibrate Housekeeping
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 17-giu-2014 16.35.34 Last Modified / 25-nov-2014 9.40.16

Unnamed Association	
From	**Name** / **Value** End Model Element / Calibrator Provide Property Getter Method / false Provide Property Setter Method / false Multiplicity / Unspecified Visibility / private Aggregation Kind / None Navigable / Navigable Derived / false

	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author: Administrator Create Date Time: 19-mar-2010 17.32.02 Last Modified: 9-giu-2011 17.39.34
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: Administrator Create Date Time: 19-mar-2010 17.32.02 Last Modified: 25-nov-2014 9.40.16	

Unnamed Extend	
From	In Flight Calibration
Visibility	Unspecified
Extension Point	On Ground Calibration
Project Management	**Name** / **Value** Author: Administrator Create Date Time: 9-giu-2011 17.37.45 Last Modified: 25-nov-2014 9.40.16

1.2.38. Get Sun Direction

Name	**Value**
Documentation	Returns the last measured direction of sun; two numbers, 16-bits each, giving sun direction along X and Y axis. It can be read using **getHousekeeping** method of **1B45** package. • Value **housekeeping[SUN_SENSOR_X]** returns the angle between sun direction and the y-z plane, in units of in 180/2048 degrees (= pi/2048 radians); positive value indicates that sun is towards positive x-axis. In case sun is not visible, it returns -32768. • Value **housekeeping[SUN_SENSOR_Y]** returns the angle between sun direction and the x-z plane, in units of in 180/2048 degrees (= pi/2048 radians); positive value indicates that sun is towards positive y-axis. In case sun is not visible, it returns -32768.

	The (model element not found) will read components of captured image as variable **t_sensor housekeeping** using the Get Module Housekeeping use case of Bk1B45_Slave package.
ID	62
Abstract	false
Leaf	false
Root	false
Stereotypes	UseCase
Business Model	false
Status	Identify
Rank	Unspecified
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 19-mar-2010 16.40.11 Last Modified / 25-nov-2014 9.40.16

1.2.38.1. Relationships

Unnamed Include	
To	Get Module Housekeeping
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 13-nov-2014 15.35.56 Last Modified / 25-nov-2014 9.40.16

Unnamed Include	
To	Take Picture Immediate
Visibility	Unspecified
Stereotypes	Include
Project Management	**Name** / **Value** Author / gabriele.defranciscis Create Date Time / 10-nov-2014 16.14.52 Last Modified / 25-nov-2014 9.40.16
Unnamed Association	
From	**Name** / **Value**

	End Model Element	Central Attitude Controller
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false
	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author / Administrator Create Date Time / 19-mar-2010 17.32.54 Last Modified / 1-giu-2010 17.41.35
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author / Administrator Create Date Time / 19-mar-2010 17.32.54 Last Modified / 25-nov-2014 9.40.16	

Unnamed Association		
From	**Name**	**Value**
	End Model Element	Central Attitude Controller
	Provide Property Getter Method	false
	Provide Property Setter Method	false
	Multiplicity	Unspecified
	Visibility	private
	Aggregation Kind	None
	Navigable	Navigable
	Derived	false

	Derived Union	false
	Read Only	false
	Static	false
	Leaf	false
	Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.05 Last Modified: 1-giu-2010 17.41.35
Abstract	false	
Leaf	false	
Visibility	Unspecified	
Derived	false	
Project Management	**Name** / **Value** Author: dmasera Create Date Time: 15-mar-2010 17.13.05 Last Modified: 25-nov-2014 9.40.16	

1.2.38.2. Use Case Descriptions

Main		
Super Use Case		
Author	Administrator	
Date	May 21, 2008 3:53:03 PM	
Brief Description		
Preconditions		
Post-conditions		
Flow of Events	**Actor Input**	**System Response**

1.2.39. Bk1B45_Slave

Name	Value
Documentation	This package contains all specifications of the **Peripheral** (slave) side of the 1B45_Subsystem_Serial_Data_Bus. In these diagrams, the **System** is the Peripheral side. It comprises two Use Case diagrams: • **Housekeeping Use Case Diagram**, which incorporates all use cases related to reading from the slave its housekeeping data and the related history and statistics, system status; to issue commands to the system like wake-up, standby and reset, and to read/write application-specific (**Designer**-defined) data.

	• **Supply, Enable, Configuration Use Case Diagram**, which incorporates use cases related to power supply, static configuration and testing of the system. The document only describes commonly-used functions of a **Peripheral** (slave), and the Designer may add as many functions as he requires. Yet any added function should comply as much as possible to the basic protocol described herein. Furthermore, not all functions described in these use case diagrams need be implemented. Most use cases are optional. If used, they shall be implemented as specified. If not used, they can be disabled by removing appropriate attributes from the relevant classes, as indicated in the **Configure Module** use case.
Abstract	false
Leaf	false
Root	false
Visibility	public
Project Management	**Name** / **Value** Author: lilian Create Date Time: 5-feb-2010 9.40.25 Last Modified: 17-apr-2014 9.46.55

1.2.39.1. Children

Name	Documentation
Supervisor	It has the function of system supervisor, including **Designer**-defined tasks such as boot, low-power and standby and wakeup modes.
UART0_slave_P1_0	This class contains all drivers for a UART configured to be the master of a 4-wires SPI bus, with a non-standard CS signal active-high. CS shall be replicated twice, namely sent directly to CS but inverted to CS_neg pins. CS high (therefore CS_neg low) enables the slave; the opposite disable it. When disabled, MISO pin is set to 0, while other pins are inputs. When enabled; MISO level is unpredictable; other pins are inputs. MOSI is sampled on falling edge of SCK, while MISO is updated on rising edge of SCK.
UnusedInterrupts_slave	Disables all unused interrupts (if any) and traps any of them in case that, for any reason, they get enabled by mistake. The watchdog is also disabled. The constructor disables all interrupts. Each interrupt service routine disables the corresponding interrupt enable flag. If an interrupt becomes used, the Designer shall: • duplicate this class and name it appropriately • remove the corresponding interrupt service (or watchdog) routine from the duplicated class to the relevant class • remove its disabling from the constructor
MessageHandler	
CommandInterpreter	
Module Boot	Boots the system and puts it into a safe state. Details of the system after boot are TBD by the **Designer**.

	All classes shall be initialized via their constructors. It counts the number of boots and monitors the status of the **LUP** actor, to count up the number of latchups.
InterruptHandler	This class handles all the interrupts associated with normal I/O ports. UART and timer interrupts are handled separately by the corresponding objects.
Module Testing	Still TBD
Configure Module	Changes some compile-time configuration parameter. Among others: • define which <<optional>> use cases are implemented and which ones are not. An <<optional>> use case is implemented as soon as the user #define an identifier whose name is contained in the associated tagged-value **define.** • length of housekeeping vector and list of housekeeping data (see **Housekeeping** class) • timeout after which the System automatically wakes up from Standby. If 0, the System will not wake up automatically, • length and list of housekeeping history (see **Housekeeping** class) • length and list of **housekeeping** statistics (see **Housekeeping** class) • other **Designer**-defined configurations • TBD ??? The **Designer** shall define all configurable parameters.\ configRegister[0] shall be written with the Designer-defined HW/SW version: • HW configuration is the Least Significant Byte of configRegister[0] • SW configuration is the Most Significant Byte of configRegister[0] This use case is different from the **Set Module Configuration** use case, as the latter allows run-time configuration changes (if foreseen by the **Designer**), while this use case allows compile-time configuration.
Load Module Program	Load configured and compiled program into the **System** via the **DEBUG I/F**.
Housekeeping	`Class Housekeeping is aimed at storing global housekeeping, status and configuration data for AraMiS-compatible AraModules. In particular:` • `mission-dependent housekeeping data, inside vector housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING]. At most LENGTH_HOUSEKEEPING data of type ushort can be stored into the vector.` • `optionally, statistics (min, max and average value) for a selected number of housekeeping data, which are stored into vector statistics: HK_REDUNDANCY[3][LENGTH_STATISTICS]. In particular LENGTH_STATISTICS ushort, from element number FIRST_STATISTICS to element number FIRST_STATISTICS+LENGTH_STATISTICS-1 of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING].`

	• optionally, the "history" for a selected number of housekeeping data, which are stored into vector history: HK_REDUNDANCY[DEPTH_HISTORY][LENGTH_HISTORY]. History is defined as the collection of the last DEPTH_HISTORY samples of housekeeping (from element number FIRST_HISTORY to element number FIRST_HISTORY+LENGTH_HISTORY-1 of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING]). The last acquire samples are still available in vector housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING], while the previous DEPTH_HISTORY samples are stored into history: HK_REDUNDANCY[DEPTH_HISTORY][LENGTH_HISTORY], where history[0] is the most recent sample prior to the actual one. • optionally, calibration data to calibrate a selected number of housekeeping data, which are stored into vector history: HK_REDUNDANCY[DEPTH_HISTORY][LENGTH_HISTORY] (from element number FIRST_HISTORY to element number FIRST_HISTORY+LENGTH_HISTORY-1 of housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING]). The last acquire samples are still available in vector housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING] Class Housekeeping also offers some protection against radiation-induced data corruption by means of Designer-dependent multiple-modular redundancy. The template parameter HK_REDUNDANCY is used to set the type of redundancy. It can either be: • TripleData for triple-modular redundancy; • SingleData.for no reduncancy. Class Housekeeping also offers support for status and configuration data for AraMiS-compatible AraModules. In particular: • configuration registers (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]), usually one word of data per each AraModule. Each work of data is associated with one and only one AraModule. Template parameter LENGTH_CONFIG defines the max number of configuration words which are available. The meaning of each configuration word is completely Designer-dependent and shall be defined in the corresponding AraModule documentation. In many cases each individual bit of the configuration word may have a different meaning. • status registers (statusRegister: CS_REDUNDANCY[LENGTH_STATUS]), usually one word of data per each AraModule. Each work of data

	is associated with one and only one AraModule. Template parameter LENGTH_STATUS defines the max number of words which are available. The meaning of each status word is completely Designer-dependent and shall be defined in the corresponding AraModule documentation. In many cases each individual bit of the status word may have a different meaning. Class Housekeeping also offers some protection against radiation-induced status/configuration corruption by means of Designer-dependent multiple-modular redundancy. The template parameter HK_REDUNDANCY is used to set the type of redundancy. It can either be: • TripleData for triple-modular redundancy; • SingleData.for no reduncancy.
 Buffers	This class implements a set of fixed-length buffers, to be used for both Write Slave Data and Read Slave Data use cases. Up to MAXBUFFERS buffers of 256 bytes each can be allocated (provided that the processor has enough memory). Each individual buffer can be dynamically associated with either Write Slave Data or Read Slave Data, although the class guarantees that at least one buffer is always available for Write Slave Data and at least one for Read Slave Data use case. In particular, for either Write Slave Data (if command = CMD_WRITE_DATA_x) or Read Slave Data (for command = CMD_READ_DATA_x), the operations: • lock(command: t_Commands): byte* allocates a buffer for either Write Slave Data or Read Slave Data (separately) and returns a pointer to it. The caller can then start filling the buffer, up to 256 byte of data. When the buffer is to be filled with short or long or float data, least significant byte (???) is stored at a lower index than most significant byte. Furthermore, 16-bits data must be stored starting at an even index, while 32-bits data shall be stored at indices multiple of 4. • ready(command: t_Commands, length: ushort): bool must be called when the caller has finished filling the vector, to flag that the buffer is complete and ready for use and to store its actual length. Only one buffer can be under filling at any given time, yet Write Slave Data or Read Slave Data buffers can be filled and declared ready independently of each other. • (model element not found) returns a pointer to the oldest buffer which has been declared ready. Write Slave Data or Read Slave Data buffers are treated separately of each other. • release(buffer: byte*): bool states that the content of a buffer has been used, therefore the buffer is released for subsequent use. • reset(): void clears all init(): void and put the Buffers class in its initial situation. • throws away any buffer of that type still being written but not yet complete (namely, for which the ready() operation has not yet been called, if any) • finds the first available buffer of that type • locks it and declares it being written; the user shall then fill the buffer and call the ready() operation when finished. After that, the buffer is

	queued... • returns a pointer to it Returns null if no buffer of the chosen type is available or command is not supported.
Configuration and Status Management	This is a group of use cases aimed at transferring either: • status information (statusRegister: CS_REDUNDANCY[LENGTH_STATUS]) from one of many Tiles to the only Master, or • configuration information (configRegister: CS_REDUNDANCY[LENGTH_CONFIG]) to/from the only Master from/to one of many Tiles As a rule of thumb, the status contains a 16-bits (model element not found) for each AraModule present on the Tile, plus one word (status[0]) for the status of Inter Tile Communication, plus any other mission-dependent status information which might be defined by the Designer. The word status[0] also contains lastError: t_LastError (shared by all AraModule) which stores additional details on the last error reported by each AraModule. As a rule of thumb, the (optional) configuration contains a 16-bits (model element not found) for each AraModule present on the Tile, plus any other mission-dependent status information which might be defined by the Designer. Refer to the documentation of each AraModule for the detailed meaning of each status and configuration bit and the codes for the last error reported (see also the definition of t_LastError). For other types of non-standard AraModules, the Designer shall specify both the meaning of each status and configuration bit and (when used) the lastError: t_LastError codes.
Bk1B45_Slave_Root	
t_Housekeeping	
Housekeeping Management	
User Defined Messages and Commands	
Supervision and Emergency Recovery	
Basic Protocol	This section describes the half-duplex Basic Protocol for communication between one and only one Master (the actor which initiates communication; often either the OBC or a Tile Processor) and one or more Slaves (often either a Tile or an AraModule).

	The Basic Protocol supports the following actions (see the corresponding descriptions): • Write Data - when a Master wants to transfer up to 256B of data to a Slave; • Read Data - when a Master wants to read up to 256B of data from a Slave; • Command Only - when a Master wants to deliver a data-less command to a Slave. • Broadcast Write Data - when a Master wants to transfer up to 256B of data to all Slaves; • Broadcast Command Only - when a Master wants to deliver a data-less command to all Slaves; • WriteRead Data - when a Master wants to transfer up to 256B of data bidirectionally to/from a Slave. Most data transfers contain, from the Master to the Slave: • an appropriate START Indicator; the nature of the START Indicator depends on the actual protocol chosen; • an 8-bit Master address; • an 8-bit Slave address to address a specific Slave; • a 16-bits command; • an 8-bit data length field (only for Write Data, WriteRead Data and Broadcast Write Data); • data (1B to 256B; only for Write Data, WriteRead Data and Broadcast Write Data); • a 16-bit CRC check. CRC algorithm is a CRC-16 of all bytes (including command/ID and length fields); • an appropriate STOP Indicator; the nature of the STOP Indicator depends on the actual protocol chosen; and, from the Slave to the Master: • an 8-bit Slave ID to identify the Slave type; • an 8-bit data length field (only for Read Data and WriteRead Data); • data (1B to 256B; only for Read Data and WriteRead Data); • a 16-bit CRC check. CRC algorithm is a CRC-16 of all bytes (including command/ID and length fields). Further details are available in the Basic protocol - Nominal Operation diagram. If an error occurs (either wrong CRC or wrong length or no memory available, etc.) the Slave internally sets an ErrorFlag: bool and does not send any answer. By calling the Get Module Status use case, a Slave can read details on the last error and clear the ErrorFlag: bool.
Buffers Operation	
Module On Off	The **ENABLE** actor can either: • activate this signal to turn the **system** ON and enable all its functions; • deactivate this signal to turn the **system** OFF and disable all its

	functions, including the capability to receive any signal from the **CPU** and any other actor. Disabling the **system** can be dangerous, therefore this function shall be used carefully. Enabling and disabling the system is totally equivalent to putting and removing **Power Supply**.
Module Power Supply	**Power Supply** actor supplies power to the whole system. Whenever power supply is applied, the processor is rebooted by calling the **Module Boot** use case. The external circuit shall be such as to let ALL INPUTS go low when the **System** is unpowered, even during a latchup condition, to avoid stressing the input protection diode. For instance, a pull-up plus diode mechanism can be used. Alternatively, the driver shall take care of resetting all inputs to zero before unpowering the **System**. A potential problem to be carefully considered is the following: all inputs shall be set to zero when the **System** is unpowered; but, when the **System** is receiving a message from a **CPU** which is controlled by a different latchup protector, it might happen that, while transmitting, the **CPU** keeps SPI signals high during a latchup condition of the System. This forces to have both the **CPU** and the **System** under the same latchup protection device.
1B45_Subsystem_Serial_Data_Bus	
Design Module	The operation to define, design, simulate, prototype and test (both HW and SW) the specific module, starting from 1B45_Subsystem_Serial_Data_Bus specifications.
Module Standby Mode HW	Rising edge of **ENABLE** forces the System in Stand By (see **Module Standby** use case). After a timeout (see **Configure Module** use case), the system can automatically exit the **Standby** mode and enter into Wake up mode.
Ground Calibration	
defines	
N/A	difference between WRONG_LENGTH and LENGTH_ERROR WRONG_CRC and CRC_ERROR
t_StatusWord_1B45	`Specific status word for 1B45 package (inter-tile communication package).` `It inherits all attributes and operations from its parent (model element not found).`

1.2.39.2. Sub Diagrams

Name	Documentation
Buffers Operation	
Slave Class Diagram	The class diagram of the **Peripheral**

1.2.40. 1B231 Sun Sensor

Name	Value
Abstract	false
Leaf	false
Root	false
Visibility	public
Project Management	**Name** / **Value** Author: Administrator Create Date Time: 9-giu-2011 17.34.16 Last Modified: 25-nov-2014 9.40.16

1.2.40.1. Children

Name	Documentation
Housekeeping	
Attach Sun Sensor	Sun sensor as available to the user. Sensor is considered "attached" if it is physically present on the tile (flag (model element not found) of (model element not found)) and it has not been detached (see use case Detach Subsystem). If the sensor is attached and enabled (see use case Enable Subsystem), any further action of image processor will be carried on. This Use Case makes use of Set Module Configuration use case of 1B45 package, with bit (model element not found) of configuration word ((model element not found)) The difference between Attach Subsytem and Enable Subsystem is that the former has effects only if the sensor is physically present on the tile, while the latter has effects only if the sensor is attached, therefore the former enables the latter. The Attach Subsytem use case is intended only in rare occasions, to recover from an erroneous Detach Subsystem operation.
Detach Sun Sensor	Flags sun sensor as unavailable to the user. Sensor is considered "detached" either if it is NOT physically present on the tile (flag (model element not found) of (model element not found)) or it has not been attached (see use case Attach Subsytem). If the sensor is either detached or disabled (see use case Disable Subsystem), no further action can be carried on. This Use Case makes use of Reset Module Configuration use case of 1B45 package, with bit ((model element not found)) of

	configuration word ((model element not found)). The difference between Detach Subsystem and Disable Subsystem is that the former prevents the sensore from being enabled, therefore it disables any further action on the sensor, including Enable Subsystem. The Detach Subsystem use case is intended only in rare occasions, to permanently disable a faulty sensor. Since the actor can know if the sensor is attached (see use case Has Subsystem), the ARAMIS-level AOCS algorithms can be adapted consequently.
Disable Sun Sensor	Disables sun sensor (STANDBY signal as HIGH) . Such condition allows to sensor of going into standby mode and then it completes the current frame before disabling the digital logic, internal clocks and analog power enable signal. Any further action of sensor will be disregarded. If the sensor is in process of getting raw image while the Disable Subsystem use case is applied, the sensor is turned off upon it acquires frame.. When disabled, the sensor shall draw negligible power from the Power Bus. The **Master** shall reset (model element not found) bit of Configuration Word (model element not found) using Reset Module Configuration use case of Bk1B45_Slave package. The default is disabled.
Enable Sun Sensor	Enables sun sensor(STANDBY signal as low). Any further action of sensor will be carried on regularly. By default the sensor is disabled. When enabled, the sensor shall be active only when requested by the appropriate use cases (see (model element not found)). When enabled but not active, power consumption should be as little as possible. The Master shall set (model element not found) bit of Configuration Word (model element not found) using Set Module Configuration use case of Bk1B45_Slave package.
Get Sun SensorErrors	Returns the complete list of errors which may afflict the sensors of Subsystem and are detected by Supervision Subsystem use case. Errors are listed by type as bits composing the error word (model element not found).housekeeping: HK_REDUNDANCY[LENGTH_HOUSEKEEPING][(model element not found)], respecting the following order: ON_CURR_EXCESS->when image processor is operating, processor current exceeds maximum allowed value ON_CURR_LACK->when image processor is operating, processor current is less than minimum functional value OFF_CURR_EXCESS ->when image processor is disabled, there is a significant current which supply processor ERR_SENS_ENABLE ->error which stucks bit (model element not found) at 'true' ERR_SENS_STATUS->error which stucks bit (model element not found) at 'true'

	ERR_SENS_ATTACH->error which stucks bit (model element not found) at 'true' ERR_IM_PROC_ENABLE->error which stucks bit (model element not found) at 'true' ERR_IM_PROC_ATTACH->error which stucks bit (model element not found) at 'true' LOW_PDB_VOLT->error in Power Distribution Bus Voltage: voltage is below minimum critical value This use case makes use of Get Module Housekeeping use case of Bk1B45_Slave package.
On Ground Calibration	
In Flight Calibration	Allows to change value of coil area, offset and gain of magnetometer, using three signed 16-bits integers. Units are: TBD, TBD, TBD. Uses Calibrate Housekeeping use case of Bk1B45_Slave package, with: • magnetometer gain on x-axis in field **magnetometer_gain_X**, • magnetometer gain on y-axis in field **magnetometer_gain_Y**, • magnetometer offset on x-axis in field **magnetometer_offset_X**, • magnetometer offset on y-axis in field **magnetometer_offset_Y**, • coil area in field **coil_area**.
Supervision	
receiveApproximationAvailability	Receives the availability or not, of the initial approximation of attitude vector. If not available, the sensor performs a Lost in Space measurement.
receiveApproxVector	Actor sends x, y, and z coordinates of the unitary vector in J2000 coordinates, derived from the measurements of the sun sensor and/or magnetic field of the earth.
Get Sun Intensity	`Get Sun Intensity for later to filter possible bright sources as stars.`
Get Sun Direction	Returns the last measured direction of sun; two numbers, 16-bits each, giving sun direction along X and Y axis. It can be read using **getHousekeeping** method of **1B45** package. • Value **housekeeping[SUN_SENSOR_X]** returns the angle between sun direction and the y-z plane, in units of in 180/2048 degrees (= pi/2048 radians); positive value indicates that sun is towards positive x-axis. In case sun is not visible, it returns -32768. • Value **housekeeping[SUN_SENSOR_Y]** returns the angle between sun direction and the x-z plane, in units of in 180/2048 degrees (= pi/2048 radians); positive value indicates that sun is towards positive y-axis. In case sun is not visible, it returns -32768. The (model element not found) will read components of captured image as variable **t_sensor housekeeping** using the Get Module Housekeeping use case of Bk1B45_Slave package.
Get Windowed Image	`Get raw image with reduced size in order to high resolution for computation of barycenter. This operation must do upon a possible tracking algorithm.`

Bibliografia

[1] **EE-257**: A Boot Compression/Decompression Algorithm for Blackfin Processors
http://www.analog.com/UploadedFiles/Application_Notes/48548288698368EE257v01.pdf

[2] **EE-367:** Flash Programmer Drivers for ADSP-BF51xF16 Blackfin® Processors
http://www.analog.com/static/imported-files/application_notes/EE367.pdf

[3] **EE-237**: Guide to Blackfin Processor LDF Files
http://www.analog.com/UploadedFiles/Application_Notes/344674193228546579 5EE237v01.pdf

[4] **EE-356:** Emulator and Evaluation Hardware Troubleshooting Guide for CCES Users
http://www.analog.com/static/imported-files/application_notes/EE356.pdf

[5] **EE-210:** SDRAM Selection and Configuration Guidelines for ADI Processors
http://www.analog.com/UploadedFiles/Application_Notes/53047287221168EE210v02.pdf

[6] **EE-352:** Soldering Considerations for Exposed-Pad Packages
http://www.analog.com/static/imported-files/application_notes/EE352.pdf

[7] **ADSP-BF512/BF514/BF514F16/BF516/BF518/BF518F16 Blackfin Embedded Processor**
http://www.analog.com/static/imported-files/data_sheets/ADSP-BF512_514_ 514F16_516_518_518F16 .pdf

[8] **ADSP-BF518F16 Blackfin Processor Hardware Reference**
http://www.analog.com/processors/epManualsDisplay/0,2795,,00.html?SectionWeblawId=207&ContentID=69184&Language=English

[9] ADSP BF518F16 Manual
http://www.analog.com/en/processors-dsp/blackfin/adsp-bf518f16/products/product.html

[10] CrossCore Embedded Studio 1.1 Loader and Utilities Manual
http://www.analog.com/static/imported-files/software_manuals/cces_1-1-0_ldr_util_man_rev_1-3.pdf

[11] CrossCore Embedded Studio 1.1 Linker and Utilities Manual
http://www.analog.com/static/imported-files/software_manuals/cces_1-1-0_link_util_man_rev_1-3.pdf

[12] CrossCore Embedded Studio 1.1 Compiler and Library Manual
http://www.analog.com/static/imported-files/software_manuals/cces_1-1-0_bf_cpp_lib_man_rev_1-3.pdf

[13] **EE-271:** Using Cache Memory on Blackfin® Processors
http://www.analog.com/static/imported-files/application_notes/EE-271_Rev2.pdf

[14] Cross Core Embedded Studio site
http://www.analog.com/en/evaluation/adswt-cces/eb.html

[15] MT48LC4M16A2 SDRAM Memory Data Sheet
http://fp.cse.wustl.edu/cse362/Datasheets/sdram.pdf

[16] Aptina Sensor MT9V034C12STC site
https://www.aptina.com/products/image_sensors/mt9v034c12stc/

[17] ITU-R BT.656-4 Reccomendation
http://www-inst.eecs.berkeley.edu/~cs150/Documents/ITU656.doc

[18] ITU-R BT.601-4 Reccomendation
http://www-inst.eecs.berkeley.edu/~cs150/Documents/ITU601.PDF

[19] Brian W.Kernighan, Dennis M.Ritchie. *Linguaggio C*. Gruppo Editoriale Jackson, Milano, 1989