Università degli Studi di Trieste

FACOLTÀ DI INGEGNERIA
Corso di Laurea Magistrale in Ingegneria delle Telecomunicazioni

Tesi di Laurea

Sviluppo di un sistema di trasmissione dati per il satellite AtmoCube e realizzazione *software* tramite il Sistema Operativo PICOS18

Candidato:
Simone Carneglia

Relatore:
Chiar.mo Prof. Fulvio Babich

Correlatore:
Dr. Livio Tenze

Anno Accademico 2008–2009

A Giorgio ed Isabella,
i miei genitori...

Indice

1 Il Sistema AtmoCube **1**

1.1 Sottosistemi di AtmoCube 2

1.1.1 Lo spettro–dosimetro 2

1.1.2 Il Magnetometro 5

1.1.3 Il modulo Global Positioning System 5

1.1.4 Il modulo On Board Radio (OBR) 6

1.1.5 Il modulo di controllo On Board Data Handling 7

2 Aspetti della missione **12**

2.1 L'orbita del satellite e i tempi di accesso 12

2.2 Il sistema di Comunicazione di AtmoCube 13

2.2.1 Il livello fisico 13

2.2.2 Il livello di collegamento 13

2.3 Le specifiche di progetto 14

2.3.1 Requisiti elettrici e operativi 14

2.4 Effetti delle radiazioni ionizzanti sui dispositivi elettronici 15

2.4.1 Effetti *soft* ad evento singolo *Single Event Effect* (SEE) 17

2.4.2 Effetti SEU su una memoria SRAM 18

2.5 Le misure scientifiche di AtmoCube 19

2.5.1 Funzionalità del modulo *Global Positioning System* (GPS) SGR-05u 19

2.5.2 Acquisizioni del magnetometro e dati allegati 23

2.5.3 La misura dello spettro–dosimetro 24

2.6 Le informazioni di *Housekeeping* 25

2.6.1 Gli *HouseKeeping*–scienza 25

2.6.2 Gli *HouseKeeping*–sistema di tipo 1 27

2.6.3 Gli *HouseKeeping*–sistema di tipo 2 27

2.7 Operazioni che il satellite dovrà eseguire durante la sua missione 28

3 Il sistema operativo **31**
3.1 Il nucleo *multitasking* in tempo reale 31
3.2 Le norme OSEK/VDX[1] 32
3.2.1 Architettura del sistema operativo OSEK 32
3.2.2 La gestione dei *task* 34
3.2.3 Il modello a stati dei *task* 34
3.2.4 Le priorità dei *task* 37
3.2.5 La politica di schedulazione 38
3.2.6 I processi di interrupt ISR 41
3.2.7 Il meccanismo degli eventi 41
3.2.8 La gestione delle risorse 42
3.2.9 Funzionamento dell'accesso di una risorsa occupata 43
3.2.10 Restrizioni nell'uso delle risorse 43
3.2.11 Problemi tipici del meccanismo di sincronizzazione 43
3.2.12 Il protocollo di massima priorità OSEK 45
3.2.13 Gli Allarmi 45
3.2.14 I contatori 46
3.2.15 La gestione degli allarmi 46
3.3 Scelta del Sistema Operativo per AtmoCube 47
3.3.1 Salvo RTOS 47
3.3.2 Il sistema operativo μC/OS–II 47
3.3.3 Il Sistema Operativo PICOS18 48
3.3.4 La scelta di PICOS18 per AtmoCube 49
3.3.5 Le API di PICOS18 usate per il Sistema Operativo di AtmoCube . 50

4 Problematiche riscontrate durante lo sviluppo del Sistema Operativo e soluzioni proposte **55**
4.1 Reperimento delle informazioni di *HouseKeeping* 55
4.2 La sincronizzazione tra le acquisizioni del GPS e il campionamento del magnetometro 57
4.3 Il problema del *Single Event Upset* (SEU) 59
4.3.1 Il codice esteso di Hamming utilizzato per Atmocube 61
4.3.2 Salvataggio e lettura dei dati nella SRAM 62
4.3.3 Calcolo della probabilità di errore utilizzando il codice esteso di Hamming 64
4.3.4 Considerazioni sulla codifica utilizzata 66
4.4 Le risorse energetiche 67

5 Implementazione del Sistema Operativo di AtmoCube **68**
5.1 I pacchetti di Telemetria 68

5.1.1 Il Pacchetto di Telemetria 1 . . . 69
5.1.2 I Pacchetti di Telemetria 2 . . . 69
5.1.3 Il Pacchetto di Telemetria 3 . . . 70
5.2 La trasmissione *stop and wait* . . . 71
5.2.1 Lo *stop and wait* per la Telemetria 1 . . . 72
5.2.2 Lo *stop and wait* per la Telemetria 2 . . . 72
5.2.3 Lo *stop and wait* per la Telemetria 3 . . . 72
5.3 Organizzazione della memoria SRAM . . . 74
5.3.1 Gestione della memoria dedicata al salvataggio dei Pacchetti di Telemetria . . . 74
5.3.2 Salvataggio dei Pacchetti di Telemetria . . . 77
5.4 Le fasi del Sistema Operativo di AtmoCube . . . 80
5.4.1 La *Start Phase* . . . 80
5.4.2 La Earlier Operation Phase . . . 83
5.4.3 La *Standard Operating Phase* . . . 84
5.4.4 La *Unidirectional Operating Phase* . . . 87
5.4.5 La *Safe Mode Phase* . . . 88
5.5 I messaggi della *Ground Station* (GS) . . . 88
5.5.1 Il messaggio 'Passa allo stato *Early Operating Phase* (EOP) *Beacon Frame State*' . . . 89
5.5.2 Il messaggio 'Passa allo stato EOP *Extended Beacon Frame State*' 90
5.5.3 Il messaggio 'Programma il ricevitore GPS' . . . 90
5.5.4 Il messaggio 'Comando Dummy' . . . 91
5.5.5 Il messaggio 'Passa alla fase *Safe mode Phase* (SMP)' . . . 91
5.5.6 Il messaggio 'Passa allo stato *Standard Operating Phase* (SOP) *Measurement State*' . . . 92
5.5.7 Il messaggio 'Passa allo stato SOP *Transmit Telemetry State*' . . . 93
5.5.8 Il messaggio 'Aggiorna i parametri di Telemetria' . . . 93
5.5.9 Il messaggio 'Acknowledgement Telemetria' . . . 94
5.5.10 Il messaggio 'Ritorna dalla SMP' . . . 94

6 Realizzazione del Sistema Operativo di AtmoCube 97
6.1 Struttura generale di un task . . . 97
6.2 Le *Interrupt Service Routines* di AtmoCube . . . 99
6.2.1 ISR per il segnale PPS del modulo GPS . . . 99
6.2.2 ISR per un segnale in arrivo dalla OBR . . . 99
6.2.3 ISR per la gestione della carica della batteria . . . 99
6.3 Il task di sistema . . . 100
6.4 Il task per il controllo della tensione di batteria . . . 102
6.5 Il task idle . . . 104
6.6 Il task per le misurazioni . . . 108

6.6.1 Lo stato EOP *Beacon Frame State* per il task measure 110
6.6.2 Lo stato EOP *Extended Beacon Frame State* per il task measure . 111
6.6.3 Lo stato SOP *Measure State* per il task measure 112
6.6.4 Lo stato SOP *Communication State* per il task measure 114
6.6.5 Gli stati SOP *Transmit Telemetry State* e SOP *Unresponse State* per il task measure 114
6.6.6 La fase *Unidirectional Operating Phase* (UOP) per il task measure 114
6.6.7 Lo stato *Safe Mode Phase State* per il task measure 114
6.7 Il task per la gestione della radio 116
6.7.1 La fase EOP per il task radio 116
6.7.2 Lo stato SOP *Communication State* per il task radio 118
6.7.3 Lo stato SOP *Unresponse State* per il task radio 118
6.7.4 Lo stato SOP *Transmit Telemetry State* per il task radio 122
6.7.5 Lo stato UOP *Communication State* per il task radio 125
6.7.6 Lo stato *Safe Mode Phase State* per il task radio 125
6.8 Analisi delle priorità di task e risorse 125
6.9 L'ambiente di sviluppo del codice 128

7 Risultati ottenuti **129**
7.1 Calcolo del *throughput* con il sistema di trasmissione *Stop and Wait* . . . 129
7.1.1 Calcolo del tempo medio necessario per la trasmissione dei pacchetti di telemetria 130
7.1.2 Calcolo del *throughput* per i pacchetti di telemetria 1 131
7.1.3 Calcolo del *throughput* per i pacchetti di telemetria 2 132
7.1.4 Calcolo del *throughput* per i pacchetti di telemetria 3 133
7.1.5 Calcolo del throughput medio totale 133
7.2 L'intervallo di tempo tra il dato GPS e il dato del magnetometro 133

8 Conclusioni e sviluppi futuri **135**
8.1 Conclusioni 135
8.2 Sviluppi futuri 136

9 Ringraziamenti **138**

Elenco delle figure

1.1 Schema elettrico del satellite AtmoCube 3
1.2 Schema a Blocchi del modulo *Silicon Drift Detector* (SDD) 4
1.3 Schema a blocchi della OBDH descritta in [2] 8

2.1 Modulo di espulsione P–POD 15
2.2 Effetto singolo su una giunzione p–n 17
2.3 Single Event Upset su una cella di memoria SRAM 18
2.4 Contenuto del pacchetto PVT dello SGR 22
2.5 Contenuto del pacchetto 0x70 (Orbital Elements) dello SGR 22
2.6 Contenuto del pacchetto 0x72 (Comms Diagnostic) dello SGR 23
2.7 Contenuto del pacchetto 0x75 (Analogue to Digital Response) dello SGR 24
2.8 Tempi di acquisizione dello spettro–dosimetro 24

3.1 Livelli di elaborazione del sistema operativo OSEK 33
3.2 Modello a stati dei task estesi 37
3.3 Modello a stati dei task base 38
3.4 Schedulatore: Ordine di esecuzione dei task 39
3.5 Schedulazione non pre-emptive 39
3.6 Schedulazione full pre-emptive 40
3.7 Sincronizzazione dei *task* in modalità full pre-emptive 42
3.8 Inversione di priorità durante l'occupazione di un semaforo 44
3.9 Situazione di deadlock usando i semafori 44
3.10 Assegnamento di una risorsa con il protocolo di massima priorità tra due *task* usando la politica *pre-emptive* 46
3.11 Servizi offerti dal nucleo PICOS18 48

4.1 Schema a blocchi della *On Board Data Handling* (OBDH) con le modifiche apportate 56

4.2 Algoritmo di salvataggio di un byte dati con la codifica di Hamming estesa (Hamming(8,4)) 63
4.3 Decodifica di un byte di informazione da due byte di dati salvati in assenza di errori. 63
4.4 Decodifica di un byte di informazione da due byte salvati in presenza di un errore. 64
4.5 Decodifica errata di un byte di informazione da due byte salvati, in presenza di un numero dipari di bit errati. 65
4.6 Rivelazione di un errore pari e cancellazione del byte di informazione . . 66

5.1 Tempistiche necessarie alle trasmissione dei Pacchetti di Telemetria . . . 73
5.2 Organizzazione della memoria SRAM 75
5.3 Realizzazione della memoria circolare di Telemetria 76
5.4 Casi particolari che possono verificarsi quando NTR<NTW 77
5.5 Caso particolare di riscrittura della memoria quando NTW<NTR 78
5.6 Schema raffigurante le Telemetrie salvate in 10 minuti (figura 1 di 4) . . . 79
5.7 Schema raffigurante le Telemetrie salvate in 10 minuti (figura 2 di 4) . . . 79
5.8 Schema raffigurante le Telemetrie salvate in 10 minuti (figura 3 di 4) . . . 80
5.9 Schema raffigurante le Telemetrie salvate in 10 minuti (figura 4 di 4) . . . 81
5.10 Stati del Sistema Operativo di AtmoCube 82

6.1 Diagramma di flusso tipico di un task. 98
6.2 Soglie di tensione impostabili tramite i bit HLVDL3:HLVDL0 101
6.3 Schema a blocchi del modulo HLVD 101
6.4 Diagramma di flusso del task_system 103
6.5 Esempio di funzionamento del sistema di sospensione delle operazioni del satellite in caso di bassa carica della batteria 105
6.6 Diagramma di flusso del task_power 106
6.7 Diagramma di flusso del task idle. 107
6.8 Sospensione delle operazioni dei task nel caso in cui avvenga una caduta di tensione. 107
6.9 Diagramma di flusso generale del task measure e dello stato SMP *Safe Mode State* 109
6.10 Operazioni eseguite dal task measure nello stato EOP *Beacon Frame State*. 111
6.11 Diagramma di flusso delle operazioni eseguite dal task measure nello stato EOP *Extended Beacon Frame State* 113
6.12 Diagramma di flusso delle operazioni eseguite dal task measure nello stato SOP *Measure State* 115
6.13 Diagramma di flusso delle operazioni eseguite dal task measure nello stato SOP *Communication State* 116

6.14 Diagramma di flusso del task radio e operazioni eseguite nello stato *Safe Mode Phase State*. 117
6.15 Diagramma di flusso delle operazioni eseguite dal task radio nello stato EOP *Beacon Frame State* 119
6.16 Diagramma di flusso delle operazioni eseguite dal task radio nello stato EOP *Extended Beacon Frame State* 120
6.17 Diagramma di flusso delle operazioni eseguite dal task radio nello stato SOP *Communication State* 121
6.18 Diagramma di flusso delle operazioni eseguite dal task radio nello stato SOP *Unresponse State* 122
6.19 Diagramma di flusso delle operazioni eseguite dal task radio nello stato SOP *Telemetry Transmit State* 124
6.20 Diagramma di flusso delle operazioni eseguite dal task radio nello stato UOP *Communication State* 126

Elenco delle tabelle

2.1 Tempi di accesso e di Gap 13
2.2 Struttura di un pacchetto dello SGR 21

3.1 Livelli dei processi del sistema operativo OSEK 34
3.2 Descrizione degli stati dei task estesi. 35
3.3 Descrizione delle transizioni tra stati 36
3.4 Confronto tra i sistemi operativi per PIC 49
3.5 GetTaskEvent 50
3.6 SetEvent 51
3.7 ClearEvent 51
3.8 WaitEvent 52
3.9 SetRelAlarm 52
3.10 CancelAlarm 53
3.11 GetResource 53
3.12 ReleaseResource 54

4.1 Confronto tra le probabilità di errore utilizzando la codifica *Forwared Error Correction* (FEC) estesa di Hamming 66

5.1 Valori predefiniti dei parametri di Telemetria 78
5.2 Struttura del messaggio 'Passa allo stato EOP *Beacon Frame State*' 89
5.3 Struttura del messaggio 'Passa allo stato EOP *Extendd Beacon Frame State*' 90
5.4 Struttura del messaggio 'Programma il ricevitore GPS' 91
5.5 Struttura del messaggio 'Comando Dummy' 91
5.6 Struttura del messaggio 'Passa alla fase SMP' 92
5.7 Struttura del messaggio 'Passa allo stato SOP *Measurement State*' 92
5.8 Struttura del messaggio 'Passa allo stato SOP *Transmit Telemetry State*' . 93
5.9 Struttura del messaggio 'Aggiorna i parametri di Telemetria' 94
5.10 Struttura del messaggio 'Acknowledgement telemetria' 94

5.11 Struttura del messaggio ‘Ritorna dalla SMP’ 95
5.12 Messaggi che possono essere inviati dalla GS in base agli stati del Sistema Operativo 96

6.1 Livelli di priorità dei task e delle risorse del Sistema Operativo di AtmoCube 127

Acronimi

AD	*Analog Digital*
ADC	*Analog to Digital Converter*
API	*Application Program Interface*
ARQ	*Automatic Repeat reQuest*
BER	*Bit Error Rate*
BFSK	*Binary Frequency Shift Keying*
CDS	*Cubesat Design Specifications*
CM	codice a maggioranza
CRC	*Cyclic Redundancy Code*
CSA	*Charge Sensitive Amplifier*
CTS	*Conventional Terrain System*
DEEI	Dipartimento di Elettronica, Elettrotecnica ed Informatica
EADD	*End Address*
EIRP	*Equivalent Isotropical Radiation Power*
EMI	*External Memory Interface*
EOP	*Early Operating Phase*
ESA	*European Space Agency*

FCC	*Federal Communication Commission*
FEC	*Forwared Error Correction*
FPGA	*Field Programmable Gate Array*
FTR	*First To Read*
FTW	*First To Write*
GPL	*Generic Public License*
GPS	*Global Positioning System*
GS	*Ground Station*
HK	*HouseKeeping*
HLVD	*High/low Voltage Detector*
ISO/OSI	*International Standard Organization/Open System Interconnection*
ISR	*Interrupt Service Routine*
LEO	*Low Earth Orbit*
LET	*Linear Energy Transfer*
MBU	*Multiple Bit Upset*
MISO	*Master Input Serial Output*
MOSI	*Master Output Slave Imput*
NMEA	*National Marine Electronic Association*
NTR	*Next To Read*
NTW	*Next To Write*
OBDH	*On Board Data Handling*
OBR	*On Board Radio*
P–POD	*Poly Pycosat Orbit Deployer*
PPS	*Pulse Per Second*
PSU	*Power Supply Unit*

PVT Posizione Velocità Tempo

RAM *Random Access Memory*

RTT *Round Trip Time*

SAA *South Atlantic Anomaly*

SADD *Start Address*

SDD *Silicon Drift Detector*

SEB *Single Event Burnout*

SEE *Single Event Effect*

SEFI *Single Event Functional Interrupt*

SEGR *Single Event Gate Rupture*

SEL *Single Event Latchup*

SEU *Single Event Upset*

SMP *Safe mode Phase*

SOP *Standard Operating Phase*

SP *Start Phase*

SPI *Serial Peripheral Interface*

UART *Universal Asynchronous Receiver-Transmitter*

UOP *Unidirectional Operating Phase*

UTC *Coordinate Universal Time*

VEGA Vettore Europeo di Generazione Avanzata

WGS–84 *World Geodetic System* 1984

Introduzione

L'obiettivo che ci si è posti durante l'attività descritta in questo lavoro di Tesi, svolta in collaborazione con Enteos s.r.l. e MyWave di Padriciano (TS), è stato quello di realizzare un sistema di acquisizione, salvataggio e trasmissione dei dati ottenuti dal satellite AtmoCube tramite l'utilizzo di un nucleo *multitasking real–time* per un microcontrollore della famiglia PIC 18.

La missione di Atmocube è quella di studiare l'ambiente spaziale vicino alla Terra ad altitudini minime di 350 km. AtmoCube è un nano satellite di forma cubica di 10 cm di lato con una massa totale massima di 1 kg che verrà messo in orbita dal 'lanciatore' Vettore Europeo di Generazione Avanzata (VEGA) in modo da percorrere un'orbita ellittica e retrograda compresa tra 350 e 1450 km con un'inclinazione di 71^o e dovrà essere capace di instaurare delle comunicazioni bidirezionali con un'unica stazione di Terra situata a Basovizza (TS) per poter trasferire i dati acquisiti. Il sistema è un progetto innovativo supportato dal dipartimento di Fisica e dal Dipartimento di Elettronica, Elettrotecnica ed Informatica (DEEI) dell'Università degli Studi di Trieste, che si pone l'obiettivo scientifico di effettuare delle misurazioni sulla magnetosfera e sul vento solare e l'obiettivo tecnologico ed ingegneristico di realizzare un sistema satellitare costituito da apparati e tecnologie a basso costo, nate per applicazioni commerciali di uso prettamente terrestre, con l'utilizzo di risorse energetiche limitate.

In questo elaborato viene presentata la realizzazione di un Sistema Operativo capace di effettuare tutte le operazioni che il satellite dovrà eseguire durante la sua missione. AtmoCube è il primo tra i satelliti di tipo Cubesat nato con lo scopo di rilevare particolari grandezze fisiche, è dotato di un microcontrollore di tipo commerciale per la gestione dell'intero apparato e richiede la progettazione e lo sviluppo di un sistema originale per il controllo di tutte le sue funzioni e periferiche.

In particolare gli obiettivi che questo elaborato si prefigge di raggiungere sono:

- la progettazione di un Sistema Operativo capace di far funzionare il satellite secondo le specifiche richieste dai satelliti di tipo CubeSat e che sia capace di porre At-

moCube in varie modalità di funzionamento in base ad eventi esterni ed ai comandi ricevuti dalla stazione di Terra;

- la creazione di un sistema capace di coordinare gli strumenti scientifici a bordo del satellite cercando di ottenere dei dati che abbiano valenza scientifica e che non siano affetti da ritardi o da *offset*;
- l'implementazione di una memoria di tipo circolare che permetta di salvare continuamente i dati acquisiti, sovrascrivendo quelli meno recenti, proteggendoli da eventuali errori dovuti ad effetti ionizzanti comuni ai dispositivi elettronici che si trovano nello spazio;
- la realizzazione di un sistema di comunicazione bidirezionale che cerchi di migliorarne l'affidabilità attraverso un metodo di ripetizione della trasmissione, se quest'ultima non è andata a buon fine, e che permetta al satellite di trasferire i dati scientifici assieme alle informazioni sullo stato e sul funzionamento dell'intero sistema alla stazione di Terra. Quest'ultima deve essere in grado di inviare dei comandi per modificare il funzionamento del satellite ed in particolare, di riprogrammare il modulo GPS e di riconfigurare le tempistiche di acquisizione delle misure scientifiche;
- la gestione efficiente delle risorse, cercando di utilizzare la poca energia disponibile per le misurazioni solo quando ciò è strettamente necessario;
- la realizzazione di un sistema capace di monitorare lo stato della batteria e di interrompere le proprie operazioni nel caso in cui la carica si stia esaurendo, permettendo così di evitare lo spegnimento dei sistemi e di ridurre il periodo di ricarica.

Nel capitolo 1 dell'elaborato vengono descritti i principali sistemi che compongono AtmoCube: lo spettro-dosimetro, il magnetometro, il modulo GPS, il modulo radio ed il modulo di controllo *On Board Data Handling*. Nel capitolo 2 vengono invece affrontati tutti gli aspetti riguardanti la missione di AtmoCube come l'orbita del satellite, gli effetti ionizzanti e le modalità di acquisizione richieste dalla missione. Nel capitolo 3 viene spiegato brevemente il funzionamento del nucleo del Sistema Operativo *multitasking* in tempo reale utilizzato per AtmoCube e viene fatta una panoramica sui Sistemi Operativi presenti in letteratura descrivendo le motivazioni che hanno portato alla scelta di PICOS18 [3]. I problemi riscontrati durante la fase di progettazione e come questi siano stati risolti sono oggetto del capitolo 4; vengono inoltre elencate le motivazioni che hanno portato alla scelta di usare il codice esteso di Hamming per salvare i dati in memoria ed i metodi adottati per ridurre il consumo energetico del satellite. Nel capitolo 5 vengono illustrate l'architettura del Sistema Operativo e la realizzazione del codice delle diverse fasi operative per far sì che il satellite operi in diverse modalità di funzionamento. Viene inoltre descritta la struttura dei dati da inviare alla stazione di Terra sotto forma di

pacchetti di Telemetria, la modalità di trasmissione di tipo *Stop and Wait* adottata ed i possibili messaggi di comando che la stazione di Terra può inviare al satellite. Nel capitolo 6 vengono dettagliati i *task* del Sistema Operativo ed il loro funzionamento per ogni fase operativa del satellite. Nel capitolo 7 vengono riportati i calcoli utili a stimare il *throughput* ottenibile con il metodo *Stop and Wait* implementato e viene descritta la procedura per la misura del tempo di ritardo tra il dato di posizione proveniente dal GPS ed il campionamento del campo magnetico eseguito dal magnetometro. Infine, nel capitolo 8 vengono presentate le conclusioni del lavoro di Tesi ed i possibili sviluppi futuri.

Capitolo 1

Il Sistema AtmoCube

AtmoCube è composto da sette parti principali:

- lo spettro–dosimetro;
- il magnetometro;
- il GPS;
- il ricetrasmettitore *On Board Radio* (OBR);
- la *Power Supply Unit* (PSU);
- l'OBDH;
- L'Attitude Determination Control Board.

I primi due elementi elencati costituiscono la parte sperimentale del satellite: sono gli strumenti necessari ad effettuare le misurazioni delle particelle ionizzanti e del campo magnetico. Il modulo GPS fornisce informazioni di posizione, velocità e tempo, che servono come riferimento ai dati scientifici misurati.

Il ricetrasmettitore ha il compito di instaurare la comunicazione tra il satellite e la GS di Basovizza (TS), permettendo di trasmettere a Terra i dati raccolti durante le misurazioni e le informazioni sullo stato dell'intero sistema. Il trasferimento dei dati è possibile solamente in una condizione di visibilità tra il satellite e la GS e quindi in una finestra temporale limitata.

La PSU alimenta il satellite fornendo le tensioni e le correnti necessarie al funzionamento dei sottosistemi del satellite. Per questioni di risparmio energetico, essa dispone di alcune linee di abilitazione, che consentono l'esclusione di alcuni dispositivi.

Il modulo OBDH gestisce e controlla tutte le parti che compongono AtmoCube attraverso il microcontrollore sul quale è installato il Sistema Operativo Real-Time PICOS18.

Durante lo sviluppo della tesi, è stata presa in esame la possibilità di installare un nuovo sistema che effettui la cosiddetta *Attitude Determination Control*[1] e la raccolta dei dati di *HouseKeeping* (HK)[2] da trasmettere al microcontrollore dell'OBDH. Alcuni aspetti, che verranno descritti in questo elaborato, fanno riferimento proprio a questa possibilità.

Oltre a questi sette componenti, il satellite dispone: di alcune celle solari, di una serie di fotodiodi, di sensori di temperatura e di un dispositivo di sgancio dell'antenna. Le celle solari permettono di immagazzinare energia nell'accumulatore principale, costituito da una batteria agli ioni di litio; i fotodiodi sono posizionati sui lati del cubo e consentono di determinare l'assetto del satellite; i sensori di temperatura, invece, fanno parte delle misure di HK. Il dispositivo di sgancio dell'antenna è costituito da uno strumento elettromeccanico che consente di espandere l'antenna dopo l'espulsione del satellite nello spazio.

Nella Figura 1.1 è riportato uno schema elettrico del satellite AtmoCube.

1.1 Sottosistemi di AtmoCube

In questa sezione verranno brevemene descritti, per una migliore comprensione del testo, i sottosistemi che compongono il satellite. Nei capitoli successivi verrà poi descritto in dettaglio la modalità di gestione delle periferiche da parte del Sistema Operativo PICO18.

1.1.1 Lo spettro–dosimetro

Lo spettro–dosimetro misura il flusso e l'energia delle particelle che colpiscono il satellite in orbita. Il nome 'spettro–dosimetro' deriva dal fatto che lo strumento è in grado di ottenere uno spettro di energia delle particelle rilevate. La misura avviene in un certo intervallo temporale all'interno del quale viene misurata l'energia della particelle incidenti e, alla fine dell'intervallo di misura, l'insieme dei dati misurati fornisce uno spettro di energia. Le particelle con energia superiore a 70KeV non vengono processate nella modalità appena descritta, ma vengono solamente contate (il conteggio sfrutta solamente il *Charge Sensitive Amplifier* (CSA), il trigger e l'unità di controllo digitale, Figura 1.2).

Lo spettro–dosimetro svolge il suo compito grazie alla presenza di un rivelatore chiamato *Silicon Drift Chamber* (camera a deriva in silicio), che, essendo costruito in materiale semiconduttore, si presta a rilevare i fotoni nello spettro dei raggi X, dato il basso valore dell'energia di ionizzazione necessaria a creare una coppia elettrone–lacuna.

[1] È un sistema di determinazione e controllo dell'assetto che permette limitare la rotazione del satellite e di orientarlo nel modo corretto.

[2] I dati diHK sono una raccolta di dati provenienti dai sottosistemi di AtmoCube e da alcune conversioni ADC.

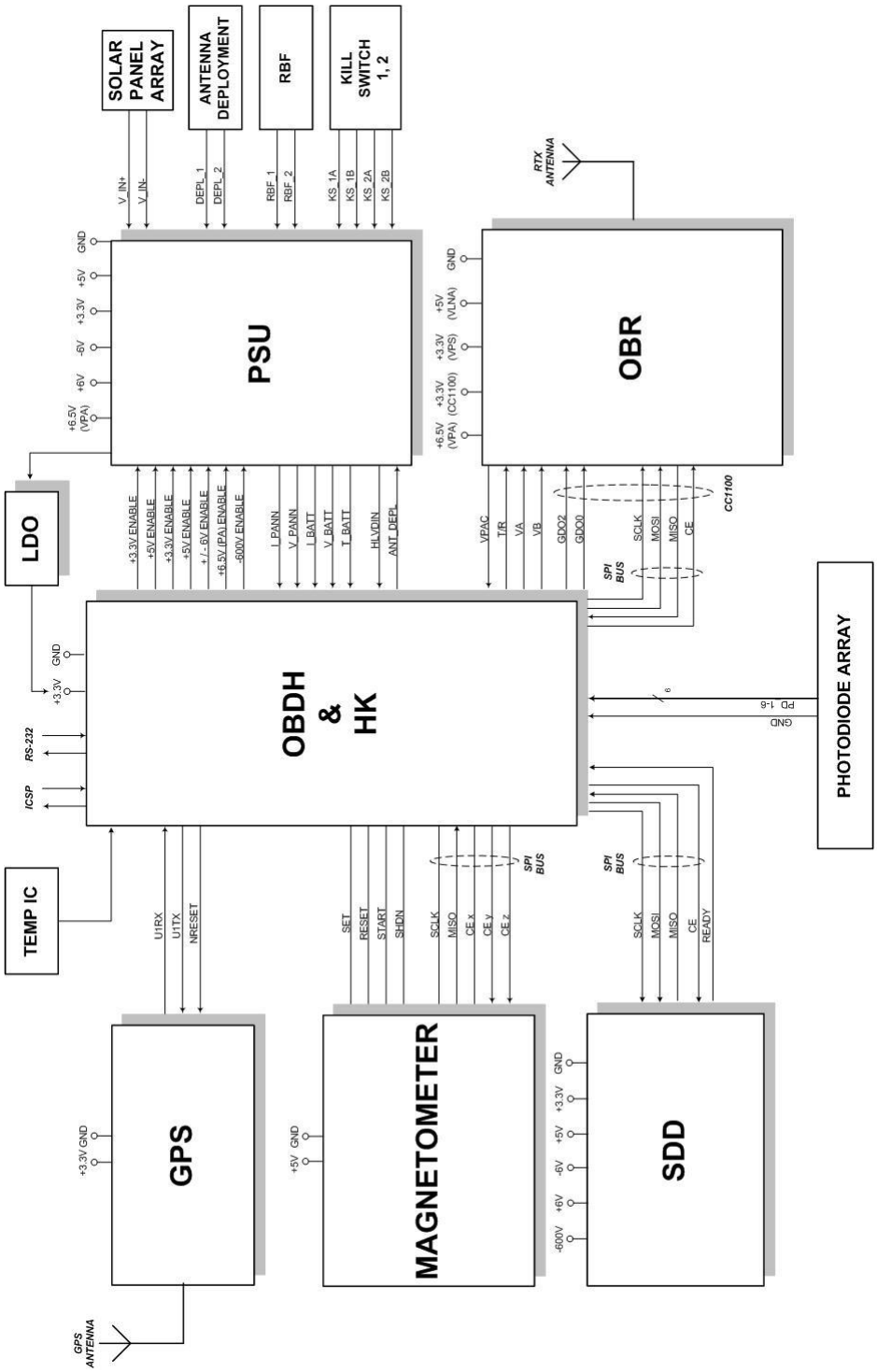


Figura 1.1: Schema elettrico del satellite AtmoCube

L'energia, così ricavata, viene poi convertita in tensione dall'amplificatore di carica, il *Charge Sensitive Amplifier*. Il segnale in uscita passa poi attraverso un filtro formatore (lo *shaper*), che ne modifica la forma per ridurre i disturbi e migliorare il rapporto segnale–rumore. Questo filtro è costruito in modo da ridurre il rumore elettronico che aumenta all'aumentare della temperatura e, poiché il satellite AtmoCube sarà sottoposto a temperature variabili, è stato necessario fornire le informazioni di temperatura allo spettro–dosimetro. All'uscita del filtro *shaper* si ottiene un impulso che mantiene le caratteristiche di proporzionalità diretta con la carica fornita dalla camera a deriva. Il *Peak Detector* mantiene il valore dell'impulso e lo invia al convertitore analogico digitale. Quest'ultimo, assieme a parte dell'unità di controllo, funge da analizzatore multicanale, in modo da contare le particelle che hanno energia compresa in un certo intervallo. Il fondo scala è tarato per coprire energie fino a 70KeV, in tal maniera, con una risoluzione dell'AD converter di 8bit, ogni canale ha un'ampiezza pari a 274eV. L'informazione ottenibile, quindi, da questo strumento è uno spettro di energia discreto dei fotoni che si scontrano con la camera a deriva. I dati ottenuti vengono immagazzinati nella memoria presente nell'unità di controllo ed aggiornati all'arrivo di ogni particella.

Dallo schema a blocchi di Figura 1.2 si nota un ulteriore circuito di *Trigger* che ha come compito quello di segnalare l'inizio di una nuova misura in corrispondenza di ogni nuovo fotone.

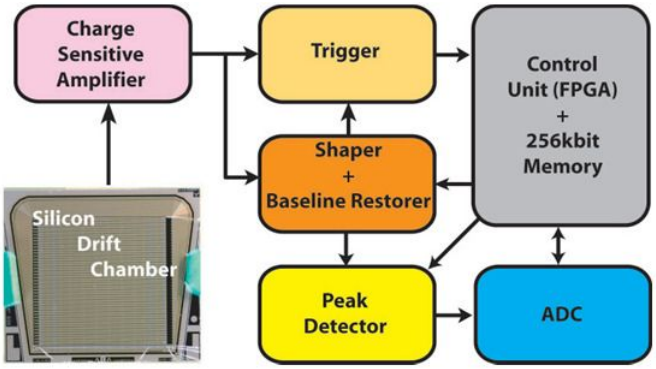


Figura 1.2: Schema a Blocchi del modulo SDD

L'interfacciamento con il modulo OBDH avviene tramite la porta di comunicazione *Serial Peripheral Interface* (SPI) ed i comandi che il microcontrollore dovrà fornire sono:

- il comando di *Reset*, che provoca la cancellazione della *Field Programmable Gate Array* (FPGA) dello SDD;
- il comando di *Start*, per incominciare una nuova acquisizione;
- il comando di *Stop*, per fermare l'acquisizione in corso;

- il comando di *Read*, per trasferire i dati acquisiti dallo SDD al microcontrollore;
- i comandi di impostazione dello *shaper*.

1.1.2 Il Magnetometro

Il circuito integrato della 'Honeywell' HMC2003, adibito all'acquisizione delle tre componenti del vettore campo magnetico terrestre (B_x, B_y, B_z), contiene tre sensori costituiti da ponti di Wheatston, realizzati con elementi magnetoresistivi di Permalloy (NiFe). Il segnale uscente viene amplificato da amplificatori a basso rumore ed opportunamente filtrati. Poiché la misura delle tre componenti deve essere contemporanea, vi è stata la necessità di utilizzare dei circuiti *sample hold* per mantenere costante il segnale per gli *Analog to Digital Converter* (ADC). Quest'ultimi sono poi interfacciati tramite SPI al microcontrollore per il salvataggio delle misure.

Per effettuare le misure in modo corretto, è necessario regolare i valori di offset che influenzano la misura, agendo sulle alimentazioni dei ponti di Wheatston. Atmocube è infatti dotato di magneti per il controllo dell'assetto che possono influenzare notevolmente le misure. Inoltre, dato che i sensori di Permalloy presentano degli effetti di isteresi, prima di ogni utilizzo bisogna orientare i domini di Weiss[3] per limitare la memoria degli eventi passati e massimizzare la correlazione tra il campo magnetico ed i segnali misurati.

1.1.3 Il modulo Global Positioning System

Il modulo GPS installato è lo SGR-05u, fornito dalla Surrey Satellite Tecnology, appositamente progettato per applicazioni spaziali con caratteristiche sia di dimensioni ridotte che di bassi consumi; esso riceve e decodifica i segnali da almeno 4 satelliti GPS e fornisce i dati di posizionamento utilizzando il protocollo *National Marine Electronic Association* (NMEA)[4]. I pacchetti dati forniti sono di diversi tipi, ma il più importante tra questi è quello che fornisce le informazioni di Posizione Velocità Tempo (PVT) da

[3]I Domini di Weiss sono delle aree della struttura cristallina di un materiale ferromagnetico, che hanno un'orientazione magnetica. Nel momento in cui lo stesso materiale viene sottoposto ad un campo magnetico, i domini di Weiss vengono orientati secondo un'unica direzione. Una volta portato il materiale ferromagnetico nello stato di saturazione si può affermare che esso ha raggiunto una polarizzazione magnetica totale, con la magnetizzazione di tutti i domini di Weiss allineata lungo un'unica direzione. Nello stato smagnetizzato invece la direzione della magnetizzazione all'interno dei domini di Weiss risulta diretta mediamente in modo casuale. Questo è il motivo per il quale, dal punto di vista macroscopico, il corpo non appare magnetizzato.

[4]NMEA 0183 (o NMEA per breve) è protocollo che specifica la comunicazione dati tra dispositivi elettronici marini come echo sounder, sonar, Anemometro (velocità e direzione vento), gyrocompass, pilota automatico, GPS e molti altri tipi di strumenti. E 'stato definito da US-based National Marine Electronics Association. Il NMEA 0183 standard utilizza un semplice ASCII, protocollo di comunicazione seriale che definisce come i dati vengono trasmessi in una frase da un oratore ad uno ascoltatore alla volta.

correlare con le misure le scientifiche fatte dal satellite. Un altro motivo per il quale il dato PVT è rilevante per la missione di Atmocube è che fornisce informazioni sulla posizione istantanea del satellite, permettendo al microcontrollore di conoscere la posizione del modulo satellitare per dare inizio alla trasmissione dati con la GS di Basovizza (TS).

I segnali messi a disposizione dall'interfaccia del GPS, di cui ci occuperemo in questo elaborato sono:

- *Universal Asynchronous Receiver-Transmitter* (UART): permette un collegamento punto punto con il microcontrollore interno del GPS e costituisce il metodo principale di controllo e comunicazione;
- *Pulse Per Second* (PPS): è un'uscita di tipo TTL che può essere impiegata per sincronizzare segnali di clock esterno al ricevitore GPS;
- NRESET: è un comando a logica TTL impiegato per mandare un segnale di reset al microcontrollore del GPS.

1.1.4 Il modulo On Board Radio (OBR)

Il ricetrasmettitore di bordo ha il compito di stabilire una comunicazione *half–duplex* con la GS. Il modulo OBR è costituito da un circuito stampato al cui interno trovano posto i seguenti sottosistemi:

- il Ricevitore ed il trasmettitore a bassa potenza, composto dal circuito integrato CC1101;
- il commutatore Rx/Tx in bassa potenza;
- il circuito del Front End del ricevitore, comprensivo di amplificatore a basso rumore e filtri passa banda per l'attenuazione dei prodotti di intermodulazione di secondo ordine;
- l'amplificatore di potenza del trasmettitore;
- il commutatore di antenna;
- il circuito per la misura della potenza emessa.

Il ricetrasmettitore a bassa potenza è un circuito integrato dotato di un modem e di un ricetrasmettitore radio UHF. Tale dispositivo si interfaccia all'OBDH attraverso la porta SPI, permettendo così lo scambio di dati con il microcontrollore. In fase di trasmissione, il microcontrollore trasferisce i byte da trasmettere alla GS nella memoria FIFO del modulo radio CC1101; quest'ultimo preleva i dati ricevuti e li inserisce all'interno del

payload del pacchetto da trasmettere. L'invio dei dati inizia con la generazione di un segnale di preambolo da parte del ricetrasmettitore seguito dalla trasmissione del pacchetto. Il segnale uscente viene quindi portato ad una potenza di 10 dBm dall'amplificatore ad alta potenza e trasmesso attraverso l'antenna, un bipolo a $\lambda/2$.

In fase di ricezione, il commutatore d'antenna viene collegato all'amplificatore a basso rumore, il quale amplifica il segnale ricevuto e lo trasferisce al modulo CC1101. Sarà compito di quest'ultimo demodulare il segnale ricevuto, avvertire il microcontrollore che è stato ricevuto un pacchetto e trasferire il relativo payload al microcontrollore, sempre tramite la porta SPI.

1.1.5 Il modulo di controllo On Board Data Handling

Uno degli scopi di questa tesi è quella di descrivere come si è giunti allo sviluppo del software per il Sistema Operativo di Atmocube. Tale *software* è stato creato appositamente per essere eseguito dal microcontrollore PIC18F8722, installato nella OBDH della quale si rende quindi necessaria una descrizione dettagliata. Lo studio e la progettazione del modulo OBDH sono descritte in [2]. Per poter comunicare con la Attitude Determination Control Board e utilizzare la linea PPS del GPS, sono state apportate alcune modifiche all'elaborato di [2] (come è riportato nellaFigura 1.3), che verranno descritte nei capitoli successivi.

Il microcontrollore PIC18LF8722

Il microcontrollore, installato sulla OBDH, è il PIC18LF8722 (costruito dalla Microchip[5]) e fa parte della famiglia PIC18F, che identifica la categoria a più alte prestazioni dei microcontrollori a 8 bit. I PIC della famiglia 18F sono caratterizzati da:

- un'architettura che usa parole di programma a 16 bit di tipo RISC con uno stack a 32 livelli di profondità;
- un moltiplicatore *hardware* 8x8;
- *interrupt* multipli interni ed esterni;
- velocità di esecuzione di 16 MIPS;
- una piattaforma di programmazione in ambiente C.

[5]Microchip Technology è una società produttrice di semiconduttori soprattutto microcontrollori, ma anche memorie di tipo EEPROM, amplificatori operazionali, dispositivi per comunicazione wireless e radio frequenza, e dispositivi per la gestione dell'alimentazione di potenza. L'azienda è nota soprattutto per essere la produttrice dei microcontrollori PIC, famiglia di microcontrollori derivati dal PIC1650 originariamente sviluppato dalla General Instrument da cui la Microchip Technology fu creata

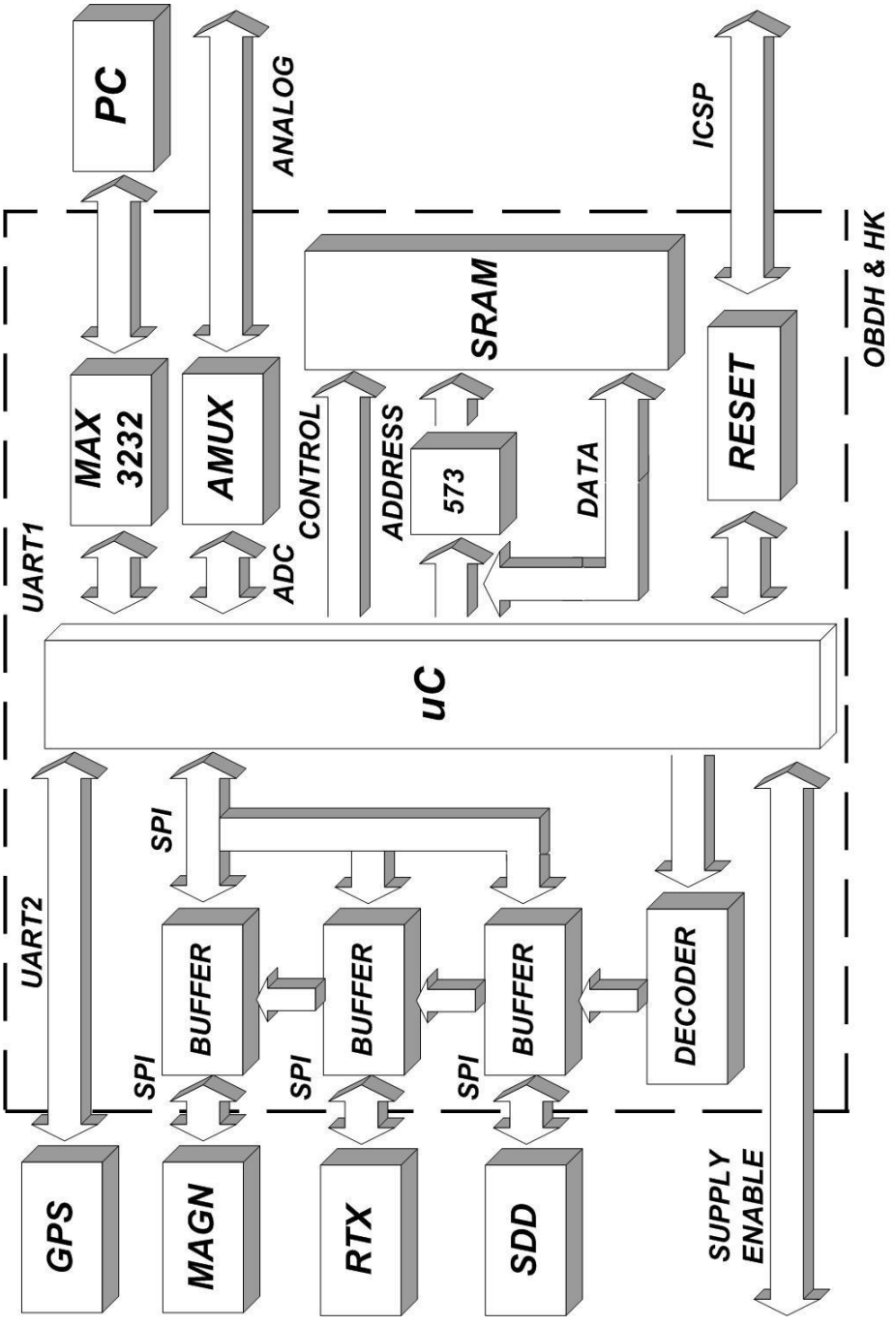


Figura 1.3: Schema a blocchi della OBDH descritta in [2]

Le caratteristiche che hanno portato alla scelta del PIC18LF8722 come microcontrollore della OBDH sono:

- **Numero adeguato di porte**: il microcontrollore in questione dispone di 80 pin e 70 di questi sono utilizzati come *Input* ed *output*.
- **La possibilità di controllare una memoria SRAM esterna**: la famiglia PIC18LF8722 implementa la *External Memory Interface* (EMI). Essa è nata con lo scopo di espandere la memoria programma nel caso in cui la memoria flash sia insufficiente ad ospitare l'intero codice macchina. Tramite l'uso della EMI, il *program counter* del microcontrollore è in grado di indirizzare fino a 2Mbyte di memoria. Le modalità d'uso della EMI sono:
  - gestione del microcontrollore interamente dalla memoria esterna;
  - impiego di una combinazione di memoria interna ed esterna sempre con limite massimo di 2Mbyte;
  - uso di una memoria *flash* per applicazioni di riprogrammazione del codice o tabelle dati molto ampie;
  - uso di dispositivi *Random Access Memory* (RAM) per l'immagazzinamento di grandi quantità di dati o variabili.

  La modalità d'uso della memoria scelta per il microcontrollore dell'OBDH permette l'indirizzamento del *program counter* fino a 2MB. Tale modalità prende il nome di *extended microcontrolled mode (EMC)* e permette l'accesso sia alla memoria interna di 128KB che alla memoria esterna. La somma delle due memorie utilizzabili dal *program counter* deve però mantenersi al di sotto dei 2MB.
- **Periferica di comunicazione SPI**: per le comunicazioni seriali sincrone dello SDD, il magnetometro, OBR e il modulo per l'*Attitude Determination Control*.
- **Periferica di comunicazione seriale UART**: usata per la comunicazione con il GPS;
- **convertitore analogico–digitale**: alcune porte, configurabili via *software*, del microcontrollore sono adibite alla conversione ADC a 10 bit.
- **Consumi ridotti**: il PIC18LF8722 dispone della tecnologia nanoWatt grazie la quale si è in grado di ridurre drasticamente il consumo di energia agendo in diversi modi:
  - **Run Modes alternati**: fornendo al controller la sorgente di *clock* dal Timer1 o dal blocco dell'oscillatore interno, il consumo di potenza durante l'esecuzione del codice può essere drasticamente ridotto;

  - **Idle Modes multipli**: un altro modo per ridurre i consumi consiste nel porre il microcontrollore nella modalità *Idle*, cioè mantenere attive tutte le periferiche, ma disabilitare la CPU;
  - **On–the–fly Mode Switching**: le varie modalità di power management possono essere richiamate dal codice, permettendo all'utente di incorporare applicazioni di gestione della potenza nel progetto del *software*;
  - **basso consumo dei moduli base**: i consumi minimi, per quanto riguarda il Timer1 (che è la sorgente di clock a bassa frequenza) ed il *Watchdog Timer*, sono per loro natura ridotti;

- **memoria programma**: la memoria RAM interna, adibita allo stoccaggio delle variabili programma e dei vari registri di configurazione e funzionamento, è di 4KB; la EEPROM, per il salvataggio dei dati non volatili è di 1KB e la memoria *flash* adibita al salvataggio del codice è di 128KB;

- **tensione di funzionamento**: il PIC18F8722 utilizzato, per la OBDH, funziona tra 2,0 e 5,5 V. La tensione bassa di funzionamento permette così di ridurre ulteriormente l'assorbimento di corrente del dispositivo.

I dati raccolti da AtmoCube possono essere immagazzinati nella porzione di memoria RAM esterna al microcontrollore, a partire dall'indirizzo di memoria 01FFFFh fino all'indirizzo 1FFFFFh: si hanno quindi a disposizione circa 1966KB.

Bus SPI per la comunicazione delle peiferiche esterne

La comunicazione SPI è di tipo bidirezionale tra due dispositivi: uno *master* e uno *slave*. Lo standard SPI prevede un *bus* di comunicazione a 4 linee:

- SCLK: per trasferire il comando di sincronizzazione (*Clock*) dal dispositivo *master* a quello *slave*;

- *Master Input Serial Output* (MISO): per il trasferimento dei dati dal dispositivo *master* a quello *slave*;

- *Master Output Slave Imput* (MOSI): per il trasferimento dati dallo *slave* al *master*;

- CE: linea di abilitazione per instaurare la comunicazione.

Nel satellite AtmoCube, sono presenti alcuni moduli che impiegano lo standard seriale SPI per la comunicazione. Questi dispositivi devono essere tutti collegati al microcontrollore, ma, dato che quest'ultimo è dotato di una sola porta SPI, è stato necessario usare delle linee di abilitazione per permettere la condivisione dello stesso *Bus* SPI. Su ogni *bus*

delle periferiche è stato posto un *buffer* che abilita o meno il collegamento con il microcontrollore. Le linee CE, del rispettivo *bus* di comunicazione delle periferiche abilitano i *buffer*, permettendo la comunicazione diretta della periferica con il microcontrollore. Le linee CE sono comandate da un *decoder* che, a sua volta, è comandato dal microcontrollore. Per selezionare una periferica, il microcontrollore deve quindi scegliere la linea CE opportuna agendo sugli ingressi del *decoder*.

Comunicazione seriale asincrona del microcontrollore con il GPS

Il microcontrollore utilizza la porta USART e comunica, attraverso uno traslatore di livello, con il GPS che usa il protocollo RS–232.

Misura dei segnali analogici per mezzo del convertitore *Analog Digital* (AD)

Il microcontrollore dell'OBDH viene utilizzato anche per la misurazione di alcune tensioni provenienti dai vari sottosistemi di AtmoCube. L'OBDH è dotata di alcune porte che possono essere usate come convertitore ADC ed è dotata di un *multiplexer* analogico che collega tutti i segnali analogici ad un'unico ADC del microcontrollore che seleziona la linea sulla quale effettuare la conversione operando sugli ingressi del *multiplexer*.

Abilitazione dei blocchi funzionali della PSU

Alcune porte del microcontrollore vengono dedicate alle abilitazioni delle linee di alimentazione della PSU. I sottosistemi di alimentazione del satellite, infatti, sono per lo più indipendenti: ogni linea di alimentazione è attiva solamente quando un MOS, presente sulla PSU, viene posto in stato di conduzione. Sono accessibili in totale otto alimentazioni organizzati con il seguente criterio:

- 3 abilitazioni per i moduli a $+3,3V$;
- 2 abilitazioni per i moduli a $+5V$;
- 1 abilitazione per la tensione di alimentazione dell'SDD di $+6V$;
- 1 abilitazione per l'alimentazione del ricetrasmettitore di $+6,5V$;
- 1 abilitazioni per l'alimentazione a $-600V$.

Capitolo 2

Aspetti della missione

2.1 L'orbita del satellite e i tempi di accesso

L'orbita di AtmoCube sarà di forma ellittica inclinata di 71° rispetto al piano equatoriale della Terra, con un perigeo di 350 Km ed un apogeo di 1450 Km. Inoltre, il satellite compirà un orbita di tipo retrogrado: ciò significa che dopo il completamento di un'orbita (in circa 6888 s), il satellite non si troverà più sopra il punto di partenza, ma un po' più a Est. Si evince quindi che il periodo di tempo all'interno del quale il satellite può trasmettere verso la GS di Basovizza (TS) è tempo–variante. È stato analizzato in [4] che, a causa dell'attrito con l'atmosfera terrestre, il satellite, durante il moto orbitale, subirà un rallentamento di velocità di 20 Km/h al mese. Secondo i calcoli effettuati dal dipartimento di Fisica dell'Università degli Studi di Trieste, il satellite, a regime, si porterà su un orbita circolare a 350Km dalla Terra.

AtmoCube è stato progettato per comunicare con un'unica stazione ricevente di Terra (la GS) posizionata a Basovizza (TS). La regione di spazio, denominata 'finestra d'accesso', all'interno della quale il satellite può trasmettere i dati a Terra è delimitata da un'elevazione di almeno 30° e da una distanza dalla GS minore o uguale a 4000 Km. In base alla configurazione dell'orbita, alla velocità del satellite ed ad altri parametri il Dipartimento di Fisica dell'Università di Trieste ha stimato la variazione dell'intervallo di tempo durante il quale il satellite si troverà all'interno della finestra d'accesso. Nella Tabella 2.1 sono riportati i tempi di accesso[1] ed i tempi di Gap[2] ottenuti dalle simulazioni.

[1] Il 'tempo di accesso' è periodo di tempo in cui il satellite si trova in condizione di visibilità elettromagnetica con la GS.

[2] Il 'Gap' è il periodo di tempo in cui il satellite non si trova nella finestra trasmissiva

Tabella 2.1: Tempi di accesso e di Gap

	Minimo	Medio	Massimo
Gap [s]	4938.842	9674.479	47371.808
Tempo di accesso [s]	19.763	929.206	1413.582

2.2 Il sistema di Comunicazione di AtmoCube

Come tutti i sistemi di comunicazione, anche quello di Atmocube con la GS di Basovizza (TS), può essere descritto tramite il modello *International Standard Organization/Open System Interconnection* (ISO/OSI): nelle due sezioni seguenti verranno descritti i primi due livelli della pila.

2.2.1 Il livello fisico

Seguendo il modello ISO/OSI, il livello fisico descrive il mezzo trasmissivo, la modulazione e le potenze utilizzate.

Il mezzo trasmissivo è quello radio e sia Atmocube che la GS sono dotati dello stesso circuito integrato ricetrasmittente a bassa potenza (il CC1101) che effettua la modulazione *Binary Frequency Shift Keying* (BFSK) alla frequenza portante di 437,5Mhz. L'antenna di bordo del satellite è composta da un dipolo in $\lambda/2$ e l'*Equivalent Isotropical Radiation Power* (EIRP) di 7,58 dBW circa. La GS fa uso di un'antenna YAGI a 17 elementi a doppia polarizzazione e ha EIRP di 28,4 dBW circa.

La comunicazione è di tipo bidirezionale *half–duplex* con velocità trasmissiva di 10kbit/s e ritardo di propagazione, nel caso in cui la comunicazione avvenga alla massima distanza (4000Km), di 13 ms.

Secondo i calcoli di *data–link budget*[5], la *Bit Error Rate* (BER) stimata in *downlink* nel caso peggiore è di $1,05 * 10^{-6}$, mentre quella in *uplink*, nel caso peggiore, è di $2,08 * 10^{-10}$.

2.2.2 Il livello di collegamento

Il livello di collegamento si occupa di trasferire senza errori la sequenza di bit sulla linea di trasmissione. Il livello di collegamento è implementato dal modulo CC1101, che è in grado di generare *frame* trasmissivi a dimensione fissa così formati:

- preambolo di 4byte costituito da una successione di simboli “1” e “0”;
- parola di sincronizzazione di 4byte;
- *payload* di dimensione massima di 255 byte;

- 2 byte finali per il controllo del *Cyclic Redundancy Code* (CRC) sul *payload*.

È stato deciso che il *payload* dei *frame* avrà dimensione massima per quelli trasmessi da AtmoCube e 64 byte per quelli trasmessi dalla GS.

Conoscendo la velocità di trasmissione e le dimensioni dei *frame* trasmissivi, si è in grado di ricavare il tempo necessario per la loro trasmissione:

Dimensione del *frame* in *down–link* $F_d = 2120\ bit$

Dimensione del *frame* in *up–link* $F_u = 592\ bit$

Velocità di trasmissione $f_b = 10Kbit/s$

Tempo di trasmissione *frame* in *down–link*: $T_{F_d} = \frac{F_d}{f_b} = \frac{2120}{10000} = 212\ ms.$

Tempo di trasmissione *frame* in *up–link*: $T_{F_u} = \frac{F_u}{f_b} = \frac{592}{10000} = 59\ ms.$

2.3 Le specifiche di progetto

L'*European Space Agency* (ESA) ha stabilito delle specifiche [6], a cui attenersi affinché i CubeSat vengano accettati per essere messi in orbita; tali regole nascono dall'esigenza di assicurare il corretto svolgimento della missione del lanciatore (il VEGA) e degli altri Cubesat.

Un argomento descritto nelle specifiche riguarda le caratteristiche meccaniche del satellite, ovvero le caratteristiche di massa, dimensioni e materiali utilizzati. I nano–satelliti devono essere capaci di entrare all'interno del modulo di espulsione denominato *Poly Pycosat Orbit Deployer* (P–POD) (vedi Figura 2.1), dove verranno posizionati nella fase di lancio nello spazio. Quando verrà raggiunta la zona di espulsione, il P–POD verrà aperto e, grazie a delle molle di espulsione, i Cubesat contenuti usciranno dal modulo. Nella fase di distaccamento dal P–POD verranno rilasciati dei pulsanti, posizionati su una faccia dei satelliti, che chiuderanno i circuiti elettrici, permettendo alle schede elettroniche dei satelliti di essere alimentate. Il Sistema Operativo di AtmoCube è stato concepito seguendo le specifiche *Cubesat Design Specifications* (CDS) che riguardano i requisiti elettrici ed operativi.

2.3.1 Requisiti elettrici e operativi

Nello sviluppo del Sistema Operativo di Atmocube si è tenuto conto delle seguenti norme:

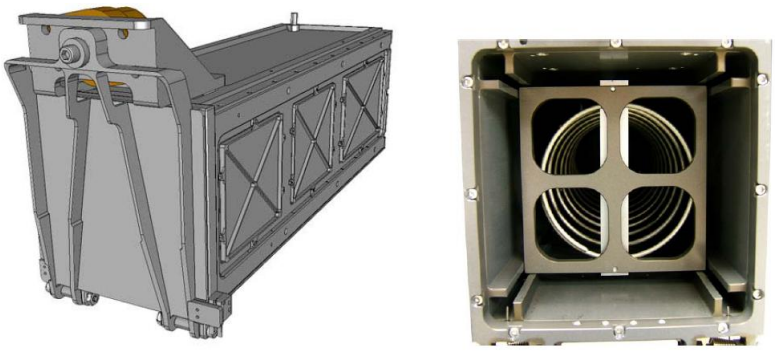


Figura 2.1: Modulo di espulsione P–POD

- nessun dispositivo elettronico deve essere attivo durante il lancio per prevenire eventuali interferenze a radio frequenza con il veicolo di lancio ed il suo carico primario[3]. I satelliti CubeSat dovranno essere completamente disattivati, durante il lancio, oppure con la batteria completamente scarica.
- Ogni CubeSat dovrà avere almeno un interruttore di accensione, sul lato orientato verso il binario di espulsione, che disabilita completamente l'alimentazione quando premuto.
- Tutti i sistemi devono essere spenti, compresi gli orologi *real–time* di sistema.
- I Cubesat dotati di batterie devono avere la capacità di ricevere un comando di blocco delle trasmissioni; questo per essere conformi alle regole della *Federal Communication Commission* (FCC).
- Tutte le parti estendibili, come ad esempio le antenne o i pannelli solari, devono essere estratti almeno 30 minuti dopo l'attivazione dell'interruttore di lancio, avvenuta all'espulsione dal P–POD.
- I trasmettitori a radio frequenza con almeno 1mW di potenza devono attendere almeno 30 minuti dopo l'espulsione dal P–POD prima di poter trasmettere.

2.4 Effetti delle radiazioni ionizzanti sui dispositivi elettronici

Nello spazio sono presenti numerose radiazioni, denominate 'raggi cosmici', costituite da particelle emesse da numerose sorgenti ed in particolare dovute all'attività solare.

[3]Il carico primario del lanciatore VEGA è costituito dalle apparecchiature principali che devono essere mandate nello spazio.

L'effetto delle radiazioni può causare sui dispositivi elettronici satellitari funzionamenti inattesi ed un loro progressivo degrado. I raggi cosmici sono formati da particelle subatomiche e da fotoni ad alta energia, che bombardano costantemente la Terra da ogni direzione. L'interazione di questi raggi con gli atomi presenti nell'atmosfera produce ulteriori radiazioni che possono arrivare fino alla superficie terrestre.

Il processo di fusione nucleare che avviene all'interno del Sole produce elettroni e protoni in grande quantità che, assieme a molecole di elio, vengono espulse in tutte le direzioni. Questo flusso di particelle prende il nome di vento solare la cui intensità dipende dall'attività della superficie ed in particolare dall'attività delle macchie solari. Queste radiazioni vanno ad aggiungersi inoltre alle particelle provenienti da altre stelle e dalle supernove della nostra galassia. Nell'avvicinarsi alla Terra, tali particelle vengono influenzate dal campo magnetico terrestre e si concentrano in fasce, note come 'fasce di Van Allen'. Dal momento che le particelle si concentrano in determinate regioni, l'intensità e la composizione della radiazione varia di molto, lungo la traiettoria seguita da un veicolo spaziale.

Dalle missioni spaziali effettuate finora dalla NASA[7] si è osservato che le più alte concentrazioni di elettroni si trovano in entrambi gli emisferi tra i 45° e gli 85° di latitudine e che, nelle zone polari, le fasce di Van Allen sono presenti anche ad altitudini ridotte. Quindi i satelliti che orbitano in tale zona devono resistere ad una notevole dose di radiazioni, contrariamente a quanto avviene per i dispositivi satellitari in orbita *Low Earth Orbit* (LEO) inclinata meno di 30°. Nella zona Sud atlantica ed in particolare vicino la costa argentina e brasiliana, a causa dell'asimmetria del campo magnetico terrestre, si trova la cosiddetta *South Atlantic Anomaly* (SAA), un zona caratterizzata da una elevata concentrazione di elettroni.

I veicoli spaziali non sono solo soggetti ad esposizioni di particelle presenti nelle fasce di Van Allen, ma anche ad ioni pesanti come ad esempio 'raggi cosmici', raggi X e bremsstrahlung[4] generati dalle particelle che, attraversando il metallo che ricopre il veicolo, perdono la loro energia. Gli effetti che una particella è in grado di produrre sui dispositivi elettronici possono essere di vario tipo; i più importanti sono i SEE che possono distinti transitori (*soft*) e permanenti (*hard*).

I dispositivi elettronici, utilizzati per esperimenti spaziali sia per usi civili che militari, devono preservare le loro funzionalità nello spazio per tutta la durata della vita della missione.

[4]La radiazione di frenamento o bremsstrahlung è la radiazione emessa da particelle cariche quando subiscono un'accelerazione (cioè una variazione di velocità, in modulo o in direzione). Ciò avviene tipicamente quando le particelle vengono scagliate contro un bersaglio metallico.

2.4.1 Effetti *soft* ad evento singolo SEE

I *Single Event Effect* si verificano quando una particella attraversa il substrato di un circuito. Essi devono la loro importanza al notevole aumento della miniaturizzazione dei dispositivi elettronici. I SEE sono di diverso tipo e possono essere distinti, in effetti transitori ed effetti permanenti. In alcuni casi gli effetti permanenti possono essere talmente disastrosi da causare la rottura totale del dispositivo. Le conseguenze che una particella è in grado di produrre dipendono dalla sua *Linear Energy Transfer* (LET), ossia dalla sua capacità di rilasciare energia, e quindi creare ionizzazione lungo il suo cammino.

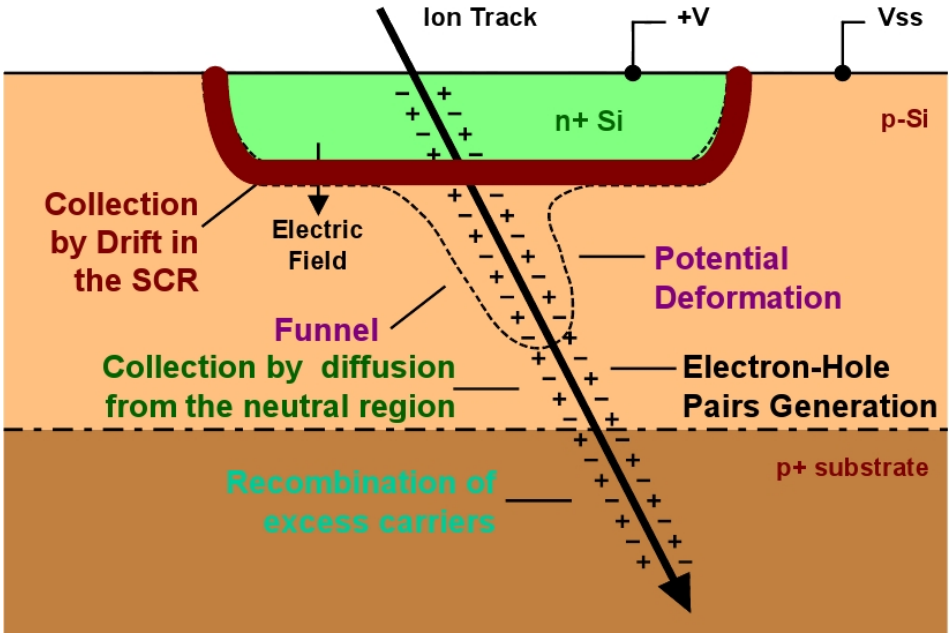


Figura 2.2: Effetto singolo su una giunzione p–n

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

- i SEU sono errori singoli non permanenti e non distruttivi, che appaiono come la commutazione errata di un bit. Essi consistono in una modifica dello stato logico di una cella elementare di memoria, indotta dalle cariche generate lungo il percorso dalla particella incidente. Questi tipi di problemi riguardano i circuiti digitali e possono essere eliminati mediante ridondanza, con algoritmi di rivelazione e correzione dell'errore;
- i *Multiple Bit Upset* (MBU) sono analoghi ai SEU, ma avvengono su più celle contemporaneamente;

- i *Single Event Functional Interrupt* (SEFI) si verificano, ad esempio, quando viene alterato un registro di configurazione che comporta la perdita di funzionalità dell'intero dispositivo.

2.4.2 Effetti SEU su una memoria SRAM

Il satellite AtmoCube utilizza una memoria SRAM esterna per l'immagazzinamento temporaneo dei dati, accumulati dalle misure scientifiche, nell'attesa di trasmetterli alla GS. In questo periodo di tempo, i bit possono subire sia SEU che MBU. Per spiegare semplicemente cosa può accadere durante un SEU in una memoria SRAM, si può pensare ad una cella di memoria elementare, costituita da un (latch) flip–flop di 6 MOS. Se uno dei transistor viene colpito da una radiazione ionizzante, le cariche prodotte nel substrato possono generare una breve corrente capace di far cambiare di stato il circuito. In questo modo può capitare che la cella di memoria passi da uno stato logico all'altro.

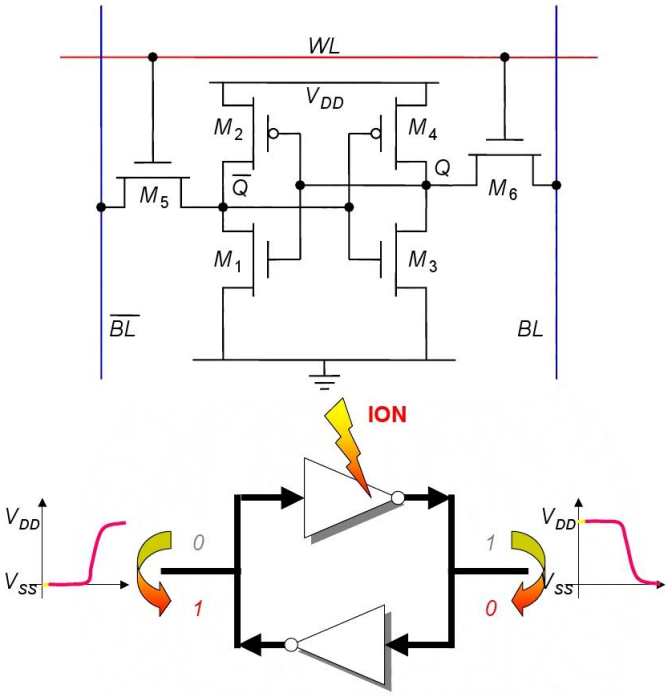


Figura 2.3: Single Event Upset su una cella di memoria SRAM

La probabilità che un evento capiti durante la missione spaziale è un dato molto importante da conoscere per poter operarare in modo da limitare i danni. La 'SEU Error Probability' è la probabilità che un bit in memoria cambi di stato; tale valore viene normalmente calcolato tramite delle simulazioni. Al simulatore vengono forniti i seguenti dati:

1. la distribuzione di particelle che ci si aspetta in orbita in funzione della LET;
2. la misura della sezione trasversale del dispositivo elettronico e la risposta dello stesso in termini di *upset* e *latch up* in funzione della LET;
3. il tasso di errore stimato, ottenuto combinando i parametri (1) e (2) con gli effetti prodotti da un flusso omnidirezionale di particelle incidenti sul dispositivo.

Dai calcoli forniti dalla NASA[7], per quanto riguarda l'orbita e la struttura di AtmoCube, risulta che la SEU Error Probability, per un dispositivo elettronico di tipo commerciale, che si trova su un orbita LEO inclinata tra i 20° e i 80°, è dell'ordine di 10^{-5} errori al giorno.

2.5 Le misure scientifiche di AtmoCube

In questa sezione verranno descritte le modalità di acquisizione delle misure scientifiche di AtmoCube. In particolare verranno descritti i pacchetti dati del GPS necessari alla missione di AtmoCube e le procedure necessarie ad effettuare le misure del magnetometro e dello spettro–dosimetro.

2.5.1 Funzionalità del modulo GPS SGR-05u

Verrà ora descritto brevemente il funzionamento del modulo GPS della 'Surrey'[8]. Come già accennato, il modulo è dotato di un interfaccia RS-232 tramite la quale è possibile comunicare con il suo *software* applicativo.

Acquisizione

All'accensione il modulo SGR–05u acquisisce i segnali provenienti dai satelliti GPS per ricavare una stima grossolana della loro distanza e della deviazione Doppler, poiché esso non ha alcun riferimento circa la sua posizione, velocità e tempo. Per ricavare tali valori, il modulo SGR-05u, si pone in una condizione denominata '*cold search*' (ricerca a freddo). In questa modalità il dispositivo inizia la ricerca su tutte le frequenze ammissibili e ad analizzare i ritardi di propagazione dei segnali provenienti dai satelliti GPS. Sebbene tutti i satelliti GPS trasmettano tutti alla stessa frequenza, questi segnali subiscono una traslazione in frequenza a causa dell'effetto Doppler, dovuto al moto relativo tra il satellite GPS e lo SGR–05u. Se il ricevitore GPS si trovasse sulla superficie terrestre, la traslazione di frequenza sarebbe di ± 4 Khz: questo sarebbe dovuto principalmente al moto del satellite GPS. Se il modulo si trovasse in orbita LEO, invece, la velocità relativa aumenterebbe, incrementando così anche la deviazione Doppler fino a ± 45 Khz.

Posizionamento

L'operazione fondamentale di un ricevitore GPS, come lo SGR-05u, è quella di determinare la sua posizione attuale. L'operazione di posizionamento può avvenire solo quando il modulo trova almeno quattro satelliti GPS ed ha ricevuto le loro effemeridi[5] aggiornate. Una volta che il ricevitore si è aggiornato ad un numero sufficiente di satelliti GPS può cominciare a calcolare, ogni secondo, la posizione, la velocità ed il tempo. Questo calcolo ha inizio quando viene inviato dallo SGR, per la prima volta, un segnale sulla linea PPS. Ogni volta che viene inviato un segnale sulla linea PPS significa che è disponibile un nuovo dato PVT indipendente dai precedenti. Quando sono stati acquisiti sufficienti satelliti e la SGR ha stimato la sua posizione, velocità e tempo, la modalità dello SGR passa da *cold search* a *Highest Elevation*, nella quale i satelliti vengono selezionati in base alla loro altezza sopra lo SGR.

Il sistema di coordinate

Secondo le impostazioni predefinite, lo SGR comunica la posizione nel sistema di coordinate cartesiane in formato *World Geodetic System* 1984 (WGS–84)[6], ma è comunque possibile impostarlo in modo che fornisca valori in latitudine, longitudine e altitudine per comunicare la posizione ed i punti cardinali per indicare la direzione e verso della velocità.

Il tempo

Il tempo è un dato del modulo SGR che viene ricavato, con un tasso di incertezza di circa 1 microsecondo, dagli orologi interni dei satelliti GPS i quali sono sincronizzati con il riferimento *Coordinate Universal Time* (UTC) con un errore massimo di 30 nanosecondi.

Modalità di produzione dei pacchetti dati

Lo SGR può essere programmato per fornire in uscita i pacchetti dati a cadenza regolare. L'intervallo di tempo tra un pacchetto ed un altro può andare da 1 secondo a 30 minuti. Vi sono diversi tipi di pacchetti dati, ognuni dei quali ha una diversa frequenza di uscita dallo SGR che può essere modificata. In alternativa si può usare una delle tre modalità di funzionamento: *Verbose*, *Quiet* e *Silent*. Nella modalità *Verbose*, per esempio,

[5]Le effemeridi sono informazioni oribitali trasmesse dai satelliti GPS, che forniscono i dati necessari per il calcolo della posizione, velocità e tempo dei ricevitori GPS.

[6]WGS–84 è un particolare sistema terrestre convenzionale *Conventional Terrain System* (CTS), ovvero un sistema di riferimento cartesiano usato per descrivere la terra, che prende in considerazione il centro della Terra come centro del sistema cartesiano, l'asse z passante per il polo Nord, l'asse X passante per il meridiano di Greenwich e l'asse Y scelta in modo da avere una terna destrorsa, e quindi verso Est.

vengono passati in uscita diversi tipi di pacchetto con una cadenza di 1Hz; in modalità *Quiet*, invece, il pacchetto contenente il dato *PVT* viene fornito con una frequenza di 0,1Hz e gli altri pacchetti vengono forniti con frequenza ancora minore. Nella modalità *Silent*, non viene fornito spontaneamente alcun pacchetto finché non ne viene fatta esplicita richiesta.

I pacchetti dati

I pacchetti dati forniti dallo SGR sono stringhe di byte che provengono dall'interfaccia RS-232. Tutti i pacchetti hanno il formato descritto in Tabella 2.2 dove:

Tabella 2.2: Struttura di un pacchetto dello SGR

STX (0x02)	SYNC'U' (0x55)	ID (1 byte)	Lenght N (1 byte)	Data bytes D0–D(N–1)	Checksum (1 byte)	ETX (0x03)

- **STX (0x02)** è un byte, formato dalla parola esadecimale 0x02 che indica l'inizio di un nuovo pacchetto;
- **SYNC'U' ((0x55)** è un byte di sincronizzazione, formato dalla parola esadecimale 0x55;
- **ID (1 byte)** indica il tipo di pacchetto;
- **Lenght N 1 byte)** indica la lunghezza N di byte di cui il pacchetto è formato;
- **Data bytes D0–D(N–1)** è il pacchetto, formato da N byte;
- **Checksum (1 byte)** è un byte di controllo calcolato effettuando l'operazione di 'OR esclusivo' su tutti gli N byte del pacchetto;
- **ETX (0x03)** è un byte del valore esadecimale 0x03 che indica la fine del pacchetto.

Lo SGR fornisce una moltitudine di pacchetti, all'interno dei quali trovano posto numerose informazioni. Verranno ora descritti i pacchetti che saranno usati all'interno del progetto AtmoCube

Il pacchetto *Position/Velocity/Time (PVT) Solution ID 0x10*

Questo pacchetto, di lunghezza 45 byte, fornisce l'informazione di posizione, velocità e tempo. La posizione e la velocità possono essere in formato WGS–84 oppure in latitudine, longitudine e altitudine in base alle necessità dell'utilizzatore dello SGR. Oltre alle informazioni appena elencate, nel pacchetto vengono forniti:

- i dati di *clock bias*[7];
- il numero di satelliti acquisiti dallo SGR;
- un *flag* che informa se i dati di PVT sono il risultato di una nuova acquisizione, oppure una stima.

Il contenuto del pacchetto è ilustrato nella Figura 2.4.

Data Bytes	Content	Type	Units
0-3	Position x / latitude	FLOAT	m or Degrees
4-7	Position y / longitude	FLOAT	m or Degrees
8-11	Position z / Altitude	FLOAT	m
12-15	Velocity x / Velocity East	FLOAT	m/s
16-19	Velocity y / Velocity North	FLOAT	m/s
20-23	Velocity z / Velocity Up	FLOAT	m/s
24-25	GPS Time - Week Number	SHORT	Weeks since 6/1/1980
26-33	GPS Time of last position fix Seconds	DOUBLE	Seconds in week
34-37	Clock Bias (* speed of light *c*)	FLOAT	m
38-41	Clock Bias Rate (*speed of light *c*)	FLOAT	m/s
42	Number of Satellites used in position fix	BYTE	-
43	GDOP	BYTE	GDOP*10
44 =0 =1 =2	Position Fix Validity No Navigation 2D Fix 3D Fix	BYTE	-

Figura 2.4: Contenuto del pacchetto PVT dello SGR

Il pacchetto *Orbital Elements ID 0x70*

Questo pacchetto contiene i dati relativi all'orbita che sta seguendo lo SGR. Il contenuto di tale pacchetto è illustrati nella Figura 2.5.

Data Bytes	Content	Type	Units
0-7	Semi Major Axis (osc) / Mean Motion	DOUBLE	metres / radians/sec
8-15	Eccentricity	DOUBLE	-
16-23	Inclination	DOUBLE	radians
24-31	Right Ascension of the Ascending Node	DOUBLE	radians
32-39	Argument of Perigee	DOUBLE	radians
40-47	Mean Anomaly	DOUBLE	radians
48-49	Time of Validity (Year)	SHORT	Years (e.g. 1999, 2000)
50-57	Time of Validity (Day Number)	DOUBLE	(e.g. 365.98543)
58-65	GPS seconds at last TIC	DOUBLE	seconds
66 =0 =1 =2 =3	Source / Status of elements Osculating Elements invalid Valid osculating elements (epoch) Invalid mean elements Valid mean elements (epoch)	BYTE	

Figura 2.5: Contenuto del pacchetto 0x70 (Orbital Elements) dello SGR

[7] I clock bias sono degli errori disallineamento delle basi di tempo del ricevitore GPS

Il pacchetto *Comms Diagnostics 0x72*

Questo pacchetto contiene alcuni dati di telemetria inerenti l'interfaccia di comunicazione seriale. Il contenuto è illustrato nella Figura 2.6.

Data Bytes	Content	Type	Units
0-7	GPS Seconds	DOUBLE	s
	SBPP Telemetry Information		
8-9	Valid command counter (counts valid commands sent by user)	SHORT	-
10-11	Invalid command counter	SHORT	-
12	Last valid input packet ID	BYTE	-
13	Last invalid input packet ID with error	BYTE	-
14 =0 =1 =2 =3 =4	Error code of last invalid packet No error has been logged Checksum error Invalid length Invalid contents Other error	BYTE	-
15-16	Unattributed byte received count (i.e. no sync received)	SHORT	-
17	No of packet types that are programmed with output interval.	BYTE	-
18	Input buffer high water mark (per second)	BYTE	%
19	Output buffer high-water mark (per second) i.e. 80% means programmed output packets filled buffer up to 80% at one point	BYTE	%
20 =5 =6 =7	Baud Rate 9600 19200 (default) 38400	BYTE	-
21 =0 =1	Byte Stuffing Status Byte stuffing on output disabled Byte stuffing as normal	BYTE	-
	CAN-Bus Telemetry Information		
22 bits 0-2 =3 =4 =5 =6 =7	CAN peripheral status byte Last Error Code (LEC) Tx OK – CAN message transmitted OK Rx OK – CAN message received OK Wake up Status Warning Status – If high, 96 errors detected Bus Off Status – If high, 256 errors & bus off	BYTE	If byte is 7, then it has it not been updated by periph. since last read.
23	CAN Latched Last Error Code	BYTE	-
24-25	CAN BUS Off Count	SHORT	-
26-27	CAN Warning Count	SHORT	-
28-29	CAN Frames Received	SHORT	-
30-31	CAN Frames Transmitted	SHORT	-
32-33	CAN Telemetry requests sent	SHORT	-
34-35	CAN Telemetry Acks Sent	SHORT	-
36-37	CAN Telemetry Nacks Sent	SHORT	-
38-39	CAN Retries	SHORT	-
40-41	CAN Buffer margin	SHORT	-

Figura 2.6: Contenuto del pacchetto 0x72 (Comms Diagnostic) dello SGR

Il pacchetto *Analogue to Digital Response 0x75*

Questo pacchetto contiene alcuni dati riferiti a delle misure ADC effettuate all'interno dello SGR. Il contenuto del pacchetto è illustrato in Figura 2.7.

2.5.2 Acquisizioni del magnetometro e dati allegati

L'acquisizione dei dati del magnetometro, come già accennato, avviene tramite il campionamento del campo magnetico nel punto in cui si trova AtmoCube. È stato deciso che i campioni del campo magnetico debbano venire acquisiti ad intervalli regolari e che il numero di secondi tra un campione ed il successivo sia un valore che deve poter essere

Data Bytes	Content	Type	Units
0-3	Channel 0: Secondary Voltage	FLOAT	V
4-7	Channel 1: Analogue Current	FLOAT	mA
8-11	Channel 2: AGC level on RF 1	FLOAT	-
12-15	Channel 3: Total LNA Current	FLOAT	mA
16-19	Channel 4: Digital Current	FLOAT	mA
20-23	Channel 5: SGR Temperature	FLOAT	° C
24-27	Channel 6: Power Supply Temperature	FLOAT	° C
28-31	Channel 7: Not Connected	FLOAT	-
32-35	Status bits See TTC Node Data Interface	FLOAT	(Convert to Hexadecimal)
36-43	GPS Time at last TIC	DOUBLE	Seconds

Figura 2.7: Contenuto del pacchetto 0x75 (Analogue to Digital Response) dello SGR

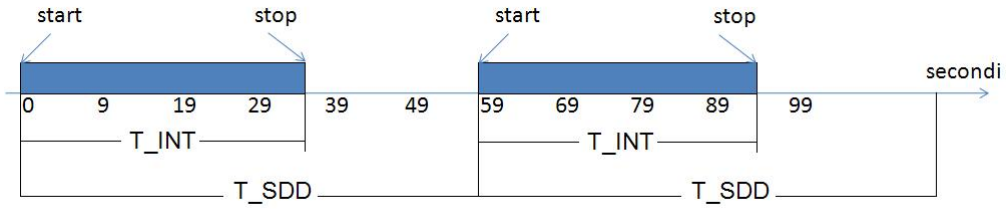


Figura 2.8: Tempi di acquisizione dello spettro–dosimetro

aggiornato da Terra. Per permettere l'analisi a Terra si rende necessario corredare i dati acquisiti con la posizione ottenuta dal GPS: bisogna quindi effettuare il campionamento del magnetometro ed allegare, ai dati ricavati, il pacchetto più recente PVT, dello SGR. Il moto che avrà AtmoCube non sarà solo di rivoluzione attorno alla Terra, ma sarà anche di rotazione su se stesso. Tale comportamento porterà ad una rotazione del sistema di riferimento di AtmoCube e quindi dei tre assi del magnetometro, rispetto al campo magnetico terrestre. Ad ogni acquisizione bisognerà quindi allegare le informazioni sull'assetto del satellite; tali informazioni derivano dalle misure dei 6 fotodiodi, uno su ogni faccia del satellite e dal valore di corrente e tensione delle celle solari.

2.5.3 La misura dello spettro–dosimetro

Lo spettro–dosimetro misura le particelle ionizzate che colpiscono la camera a deriva: dal momento in cui si invia il comando di *start*, lo spettro–dosimetro conta e misura le particelle che colpiscono la camera a deriva e ne salva i risultati sulla memoria della FPGA. L'acquisizione termina quando viene dato il comando di *stop*. Le tempistiche di acquisizione dello spettro–dosimetro sono descritte da due valori: l'intervallo tra una misura e l'altra (T_SDD) ed il periodo di integrazione di una misura (T_INT). La Figura 2.8 spiega l'evoluzione temporale delle misure dello spettro–dosimetro.

I valori di T_SDD e T_INT devono poter essere aggiornati durante la missione di AtmoCube.

2.6 Le informazioni di *Housekeeping*

Le informazioni che AtmoCube dovrà inviare alla GS non sono composte soltanto da misurazione scientifiche, ma anche da dati provenienti dai vari sottosistemi che compongono il satellite. Tali informazioni prendono il nome di '*HouseKeeping*' e servono a far comprendere a Terra come sta operando Atmocube. In questa sezione verranno elencati e spiegati tutti i dati che devono essere acquisiti dai sottosistemi ed inviati alla GS.

Durante la missione di AtmoCube, gli *HouseKeeping* verranno ricavati dai sottosistemi e salvati nella memoria assieme ai dati scientifici. Quanto il satellite si troverà nella finestra di accesso essi dovranno essere inviati alla GS.

Gli *HouseKeeping* suddividono in tre gruppi fondamentali:

- gli ***HouseKeeping*–scienza**, caratterizzati da dati che devono essere allegati alle misure scientifiche;
- gli ***HouseKeeping*–sistema di tipo 1**, costituiti da informazioni riguardanti alcuni valori ricavati da misurazioni dei sottosistemi di AtmoCube;
- gli ***HouseKeeping*–sistema di tipo 2**, costituiti da informazioni del modulo GPS GSR-05u.

2.6.1 Gli *HouseKeeping*–scienza

I dati contenuti in questa categoria di *HouseKeeping* forniranno informazioni utili a conoscere l'orientamento del satellite nel momento in cui stava eseguendo le misure con il magnetometro (vedi cap. 2.5.2). Per tale motivo essi dovranno essere salvati in memoria assieme ai dati ottenuti dal campionamento del magnetometro.

I dati contenuti negli *HouseKeeping*–scienza provengono per la maggior parte da conversioni ADC e sono:

- temperatura della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse x;
- temperatura della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse x;
- corrente della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse x;
- corrente della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse x;

- tensione della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse x;
- tensione della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse x;
- temperatura della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse y;
- temperatura della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse y;
- corrente della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse y;
- corrente della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse y;
- tensione della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse y;
- tensione della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse y;
- temperatura della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse z;
- temperatura della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse z;
- corrente della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse z;
- corrente della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse z;
- tensione della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse z;
- tensione della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse z;
- tensioni dei fotodiodi;
- corrente della batteria;
- tensione della batteria;
- temperatura della camera a deriva.

L'unico dato degli *HouseKeeping* non ricavato da una conversione ADC consiste in un contatore che informerà sul numero di acquisizioni che ha effettuato il magnetometro.

2.6.2 Gli *HouseKeeping*–sistema di tipo 1

Questi *HouseKeeping* forniranno informazioni sul funzionamento dei vari sottosistemi che compongono AtmoCube. Questi dati, a differenza degli *HouseKeeping*–scienza, non avranno necessità di essere allegati alle misure scientifiche, ma dovranno essere salvati in memoria almeno ogni minuto. I dati che costituiranno gli *HouseKeeping*–sistema di tipo 1 proverranno da conversioni ADC e da valori che verranno ricavati dal microcontrollore tramite calcoli e l'accesso a registri di memoria interni.

Gli *HouseKeeping*–sistema di tipo 1 saranno così composti:

- un contatore delle misure effettuate (lo stesso degli *HouseKeeping*–scienza);
- lo stato delle abilitazioni delle alimentazioni indipendenti della PSU;
- due valori di temperatura della radio;
- la potenza trasmessa dalla OBR nell'ultima trasmissione;
- la tensione della batteria al momento dell'ultima carica avvenuta;
- la percentuale di carica della batteria;
- una misura della tensione di 600V;
- una misura della tensione del *front–end*;
- una misura della tensione del sistema di controllo dell'assetto;
- una misura della corrente del sistema di controllo dell'assetto;
- la corrente massima assorbita dal sistema del controllo dell'assetto;
- il guadagno del controllo dell'assetto.

2.6.3 Gli *HouseKeeping*–sistema di tipo 2

I dati contenuti in questa tipologia di *HouseKeeping* saranno costituiti dai pacchetti *Comms Diagnostics 0x72* e *Analogue to Digital Response 0x75* (vadi cap 2.5.1). Gli *HouseKeeping*–sistema di tipo 2 forniranno informazioni sullo stato di funzionamento del modulo GSR–05u e dovranno essere salvati con una frequenza di dieci minuti.

2.7 Operazioni che il satellite dovrà eseguire durante la sua missione

Le operazioni che AtmoCube dovrà eseguire durante la sua missione sono di diverso tipo. In questa sezione verranno illustrate brevemente quelle principali (si veda nel capitolo 5.4 per maggior dettaglio).

Quando il satellite verrà espulso dal modulo di lancio P–POD, i pulsanti, posizionati sul lato esterno del cubo, verranno rilasciati e chiuderanno il circuito che alimenta tutti i sottosistemi, dando così inizio alla missione di AtmoCube, che si strutturerà in varie fasi, ognuna caratterizzata da diverse operazioni da eseguire.

Nella fase iniziale della missione, come richiesto dalle specifiche CDS, AtmoCube dovrà aspettare 30 minuti prima di estrarre l'antenna. Passato questo periodo di tempo, il satellite dovrà:

- estrare l'antenna;
- abilitare la linea che comanda l'alimentazione al modulo SGR–05u;
- passare automaticamente in una modalità di diagnostica di base.

Il satellite, nella modalità diagnostica, dovrà comunicare, ogni minuto, le informazioni sullo stato dei suoi sottosistemi ed aspettare di ricevere dei messaggi di comando dalla GS. Questa modalità avrà due scopi: informare la GS sullo stato dei sottosistemi di AtmoCube e stabilire un primo contatto con la GS. Dovranno esistere due tipi di diagnostica: una di diagnostica di base e una di diagnostica estesa.

Nella modalità di diagnostica di base, il satellite dovrà inviare, ogni minuto, dei messaggi, denominati '*Beacon Frame*', contenenti tutti gli *HouseKeeping* e, dopo averli inviati, dovrà porsi in attesa di un messaggio proveniente dalla GS. I comandi che la GS potrà comunicare ad AtmoCube sono:

- il comando di trasferire una data stringa di byte al modulo GSR per effettuare una sua riprogrammazione da remoto e rimanere poi nella stessa modalità;
- il comando di passare alla modalità di misurazione;
- il comando di passare alla modalità di diagnostica estesa;
- il comando di passare alla modalità 'sicura';
- il comando di rispondere ad una richiesta dummy.

Nella modalità di diagnostica estesa, in aggiunta alle operazioni eseguite nella modalità 'di base', effettuerà delle misure di prova e, ogni minuto, verranno inviati sia gli *HouseKeeping* che i dati ricavati dalle misure. Questa modalità fornirà quindi alla GS, oltre ai

dati di *HouseKeeping*, anche le informazioni sulla capacità delle apparecchiature scientifiche di operare nel modo corretto. Dopo aver inviato tali informazioni, AtmoCube si dovrà porre in attesa di un messaggio trasmesso dalla GS che potrà fornire i seguenti comandi:

- il comando di trasferire una data stringa di byte al modulo GSR per effettuare una sua riprogrammazione da remoto e rimanere poi nella stessa modalità;
- il comando di passare alla modalità di misurazione;
- il comando di passare alla modalità di diagnostica di base;
- il comando di passare alla modalità 'sicura';
- il comando di rispondere ad una richiesta dummy.

Sia nella modalità diagnostica di base, che in quella estesa, il satellite, dovrà contare il numero di messaggi inviati consecutivamente che non avranno ricevuto alcun messaggio di risposta dalla GS. Se tale conteggio raggiungerà un certo valore massimo di 65000, il satellite considererà impossibile il collegamento tra AtmoCube e la GS e passerà quindi in modalità di trasmissione unidirezionale verso la GS.

Assieme alle operazioni di misura e salvataggio, il satellite dovrà monitorare continuamente la sua posizione e calcolare la sua distanza dalla GS. AtmoCube comincerà ad inviare dei *Beacon Frame* ogni 5 secondi per comunicare alla GS che è appena entrato in una nuova finestra di accesso. Nell'intervallo di tempo tra un *Beacon Frame* e l'altro, il satellite si porrà in attesa di un messaggio dalla GS al cui interno sarà presente uno dei seguenti comandi:

- il comando di trasmettere i dati salvati in memoria: AtmoCube trasmetterà i dati salvati, non ancora trasmessi a partire da quelli meno recenti, assicurandosi che i dati appena trasmessi siano ricevuti correttamente e, in caso contrario, dovrà ritrasmetterli. La trasmissione dovrà continuare per tutto il tempo di accesso e, una volta che il satellite sarà uscito dalla finestra di accesso, dovrà tornare ad eseguire nuove acquisizioni;
- il comando di trasferire una data stringa di byte al modulo GSR per effettuare una sua riprogrammazione da remoto e rimanere poi nella stessa modalità;
- il comando di rispondere ad una richiesta 'dummy': il comando sarà costituito da una stringa di bit che AtmoCube dovrà ritrasmettere alla GS. Tale operazione è utile per sapere, da Terra, se il satellite sia in grado di comprendere i comandi fornitigli. Dopo aver trasmesso il comando 'dummy', AtmoCube, se sarà ancora all'interno della finestra di accesso, tornerà a trasmettere i *Beacon Frame* ogni 5 secondi, in attesa di un nuovo comando dalla GS;

- il comando di passare alla modalità ‘sicura’;
- il comando di passare alla modalità diagnostica;
- il comando di aggiornare alcuni parametri sulle temporizzazioni delle misure scientifiche:la GS dovrà poter inviare i nuovi parametri ed AtmoCube li modificherà per operare secondo le nuove richieste. Al termine dell’operazione di aggiornamento, il satellite, se possibile riinvierà i *Beacon Frame* ogni 5 secondi, in attesa di un nuovo comando dalla GS.

Nel caso in cui, per un qualsiasi motivo, AtmoCube non fosse in grado di ricevere i comandi dalla GS, dovrà passare ad una modalità di trasmissione unidirezionale, la quale prevede che il satellite acquisisca le misure scientifiche, le salvi in memoria (come nella modalità di misurazione) e, nella finestra di accesso, trasmetta i dati salvati senza accertarsi che siano ricevuti correttamente dalla GS.

La modalità ‘sicura’ nasce dall’esigenza di rispettare le specifiche CDS, in quanto, in questa modalità, il satellite smette di fare acquisizioni e di trasmettere qualsiasi segnale finché, dalla GS, non arriveri il comando di sblocco che permetta di tornare alla modalità in cui si trovava precedentemente.

Capitolo 3

Il sistema operativo

Il sistema operativo è un insieme di *subroutine* e strutture dati responsabili del controllo e della gestione dei componenti hardware che costituiscono un computer e dei programmi che su di esso vengono eseguiti.

3.1 Il nucleo di un sistema operativo *multitasking* in tempo reale

La principale interfaccia tra l'*hardware* della macchina di base e il Sistema Operativo è fornita dal Nucleo che ne costituisce lo strato più interno. Si tratta di un *software* avente il compito di fornire ai processi in esecuzione sull'elaboratore un accesso sicuro e controllato all'*hardware*. Si può parlare di sistema *multitasking* quando si ha la capacità di gestire l'esecuzione parallela di più processi (*task*) garantendo e controllando l'uso delle risorse e dei servizi messi a disposizione. Si parla di sistema *multitasking pre–emptive*, se è prevista la possibilità che l'esecuzione di un *task* possa essere interrotta in qualsiasi momento, ad esempio per far eseguire un altro processo o una *routine di interrupt*. Se, invece, i *task* devono cedere volontariamente il controllo del processore una volta finita l'operazione in corso, allora si ha un sistema di tipo *multitasking cooperative*.

Il nucleo *multitasking* potrebbe semplicemente gestire il parallelismo suddividendo il tempo in parti uguali per ogni *task* presente in memoria. Una soluzione del genere sarebbe poco efficiente poiché raramente i *task* possiedono le stesse necessità. Infatti possono esserci *task* sempre attivi che non hanno bisogno di una temporizzazione rigorosa e *task* attivi in modo sporadico, che richiedono tutta la potenza di calcolo del sistema nel momento in cui vengono riattivati o che hanno bisogno di essere eseguiti in istanti precisi. In base alle loro caratteristiche, ad ogni *task* è associato un valore di *priorità*. I *task* dovranno poter rispondere ad un evento in un tempo più corto possibile in accordo con le necessità degli altri *task* in esecuzione.

3.2 Le norme OSEK/VDX[1]

OSEK-VDX [1] è un progetto dell'industria automobilistica, comune ai più grandi costruttori e fornitori d'equipaggiamenti automobilistici tedeschi e francesi che ha l'obiettivo di definire uno standard per la creazione di un sistema operativo che effettui il monitoraggio delle architetture *embedded* distribuite per gli autoveicoli.

3.2.1 Architettura del sistema operativo OSEK

Le specifiche del sistema operativo OSEK servono come guida per lo sviluppo dei programmi applicativi in ambiente *embedded* affinché essi vengano eseguiti in modo parallelo ed indipendente. Il sistema operativo OSEK consente un'esecuzione controllata, in tempo reale di diversi processi e fornisce un insieme di interfacce utente, che vengono utilizzate in modo che i processi sembrino eseguiti in parallelo.

Ci sono due tipi di soggetti:

- routine di servizio di interrupt gestite dal sistema operativo;
- *task* (di base o estesi).

Le risorse *hardware* di una unità di controllo possono essere gestite dai servizi del sistema operativo *Application Program Interface* (API) che vengono chiamati o da un'unica interfaccia tramite il programma applicativo o internamente dal sistema operativo.

Per l'utilizzo di tali risorse, OSEK definisce tre livelli di istruzioni del codice a cui corrisponde un diverso trattamento da parte dello schedulatore:

- livello dei processi di *interrupt*;
- livello logico per scheduler;
- livello dei *task*.

Al livello di *task*, i vari processi competono l'un l'altro per far eseguire le proprie istruzioni al processore e possono essere schedulati in modo *non pre-emptive*, totalmente *pre-emptive* o misto, in base alla loro priorità assegnata dall'utente. Dalla Figura 3.1 si può notare che, per ogni istruzione, il contesto di esecuzione viene occupato per tutto il tempo necessario.

OSEK definisce poi le seguenti regole di priorità:

[1] Il termine OSEK sta ad indicare « Offene Systeme und deren Schnittstellen für die Elektronik im Kraftfahrzeug » (Sistema aperto e le interfacce corrispondenti per l'elettronica automobilistica »). Il termine VDX significa Vehicle Distributed eXecutive

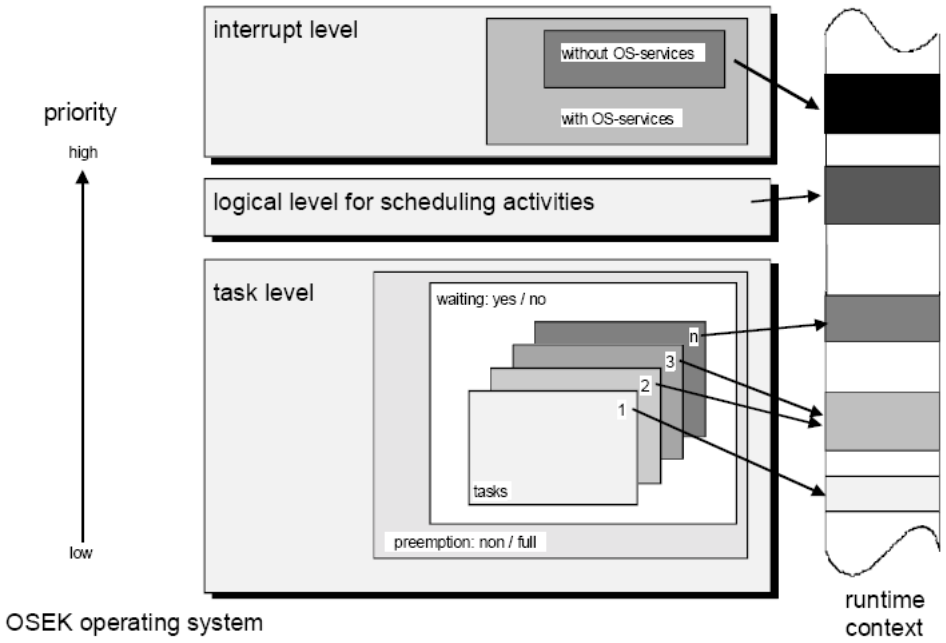


Figura 3.1: Livelli di elaborazione del sistema operativo OSEK

- i processi di *interrupt* hanno la precedenza sui *task*;
- il livello dei processi di *interrupt* possono avere uno o più livelli di priorità assegnati staticamente;
- l'assegnazione di un livello di priorità di *interrupt* ad una ISR dipende dall'implementazione e dall'architettura dell'*hardware*;
- quanto più il numero che fa riferimento alla priorità del *task* e delle risorse è elevato, tanto più tale elemento ha priorità maggiore;
- la priorità dei *task* è assegnata staticamente dall'utente.

I livelli di priorità dei *task*, dello schedulatore e delle *routine* di *interrupt* sono rappresentati come una successione di valori consecutivi. Nella Tabella 3.1 si possono vedere i livelli di priorità per i vari tipi di processi.

La priorità tra i livelli viene definita dalla seguente regola:

$$0 \leq i < j < k \leq m \tag{3.1}$$

Si evince quindi che i processi a priorità più alta sono le *routine* di *interrupt* (*Interrupt Service Routine* (ISR)), seguite poi dal processo *scheduler* ed infine dai vari *task*.

Tabella 3.1: Livelli dei processi del sistema operativo OSEK

Livello del processo	Tipo di processo
k...m	Interrupt
j	Scheduler
0...i	*task*

3.2.2 La gestione dei *task*

Il concetto di *task*

Il codice, a causa della sua complessità, può essere convenientemente suddiviso in parti differenti: tali parti sono denominate *task* al cui interno risiedono le istruzioni per l'esecuzione dell'applicazione; il sistema operativo fornisce invece l'esecuzione asincrona e concorrente dei *task*.

Le norme OSEK prevedono due tipi di *task*:

- *task* base;
- *task* estesi.

task base

I *task* base sono caratterizzati dal metodo con il quale essi rilasciano l'uso del processore agli altri *task*. Difatti essi rilasciano il processore solo se:

- terminano;
- il sistema operativo OSEK commuta ad un *task* con un livello più alto;
- avviene un interrupt che causa la commutazione ad una *routine* di *interrupt* (ISR).

task estesi

I *task* estesi si distinguono dai *task* base in quanto usano la chiamata del sistema operativo *WaitEvent*: questa pone il *task* in uno stato di *Waiting* che permette al processore di rilasciare il *task* corrente ed eseguirne un altro a priorità minore, ma senza che quello in stato di *Waiting* termini.

3.2.3 Il modello a stati dei *task*

Lo schedulatore ha il compito di gestire le risorse del sistema tra *task* concorrenti, ponendoli temporaneamente in attesa, per consentire l'accesso ordinato alle risorse. Una

risorsa fondamentale a cui tutti i *task* vogliono accedere è il processore: per questo motivo è stato creato il modello a macchina a stati, che permette di stabilire quale *task* debba essere eseguito, quali sono momentaneamente in attesa, in esecuzione o sospesi. Il sistema operativo OSEK è responsabile del salvataggio e del ripristino del contesto di ogni *task* ogni volta che avviene una transizione di stato. Inoltre, come esposto precedentemente, i *task* possono essere di tipo base od estesi e quindi sono previsti due differenti modelli di macchina a stati.

Modello per i *task* estesi

Nella Tabella 3.2 sono descritti gli stati dei *task* estesi; le transizioni da uno stato all'altro sono invece descritte nella Tabella 3.3.
Nella Figura 3.2 viene rappresentato il modello a stati dei *task* estesi.

Tabella 3.2: Descrizione degli stati dei task estesi.

running	in questo stato il *task* viene eseguito dalla CPU. Solo un *task* alla volta può trovarsi in questo stato nello stesso istante, mentre tutti gli altri stati possono essere associati a più *task* contemporaneamente;
ready	nello stato di *ready* il *task* è pronto per passare allo stato di *running* ed è in attesa che gli sia assegnato l'uso della CPU. Lo schedulatore ha il compito di decidere quale dei *task* in stato di *ready* abbia il diritto ad essere il prossimo a passare nello stato di *running*;
waiting	il *task* in questo stato non può essere eseguito poiché sta aspettando (*waiting*) almeno un evento;
suspended	in questo stato il *task* non è attivo e viene ignorato dallo schedulatore, però può essere attivato dagli altri *task*.

Modello per i *task* base

Il modello degli stati per i *task* base è essenzialmente uguale a quello per i *task* estesi, tranne per l'assenza dello stato di *waiting*.
Il modello degli stati per i *task* base è raffigurato nella Figura 3.3

Confronto tra i tipi di *task*

Si è visto in precedenza che i *task* base non possiedono lo stato di *waiting* e quindi sono dotati soltanto di elementi di sincronizzazione che devono essere implementati al-

Tabella 3.3: Descrizione delle transizioni tra stati

Transizione	Stato di partenza	Stato di Arrivo	Descrizione
activate	*suspended*	*ready*	Un servizio di sistema imposta un nuovo *task* sullo stato *ready*. Il sistema operativo OSEK assicura che il nuovo *task* parta dalla prima riga del codice.
start	*ready*	*running*	Uno dei *task* in stato di **ready** viene eseguito passando così in stato di *running*.
preempt	*running*	*ready*	Lo schedulatore decide di far passare un altro *task* in stato di *running* e quello attualmente in stato di *running* viene posto in stato di *ready*.
terminate	*running*	*suspended*	Il *task* attualmente in *running* si pone in stato *suspended* attraverso una chiamata di servizio *TerminateTask*.

l'interno del loro codice. Un vantaggio dei *task* base consiste nel fatto che essi hanno un contesto ridotto e quindi hanno un bisogno minore di risorse RAM rispetto ai *task* estesi.

Il vantaggio dei *task* estesi consiste nel fatto che essi non necessitano, nell'implementazione, di chiamate a funzioni di sincronizzazione. Nel caso in cui un *task*, per continuare la sua esecuzione abbia bisogno di un certo evento, che però non si è ancora verificato, il *task* passerà allo stato di *waiting* per lasciare il controllo della CPU agli altri processi. Al verificarsi dell'evento, il *task* passerà in stato di *ready*.

Attivazione di un *task*

L'attivazione di un *task* viene eseguita attraverso la chiamata di sistema *ActivateTask* oppure da *ChainTask* e lo stato in cui si trova il *task* appena attivato è quello di *ready*. Il sistema operativo OSEK non supporta, per quanto riguarda l'attivazione dei *task*, il passaggio di parametri tipico del linguaggio C. Le informazioni tra i *task* devono essere quindi veicolate attraverso l'uso di variabili globali.

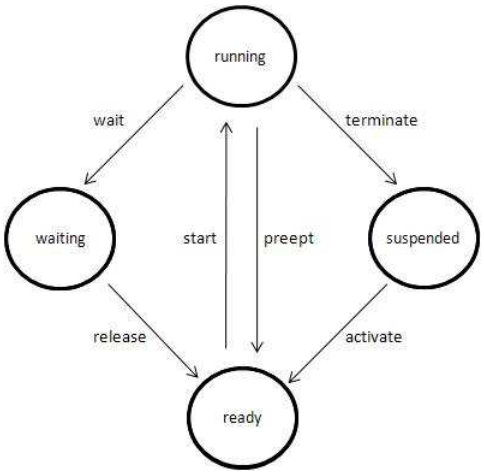


Figura 3.2: Modello a stati dei task estesi

3.2.4 Le priorità dei *task*

Lo schedulatore definisce delle liste di priorità al cui interno sono presenti i *task* in stato di *ready*. Il prossimo *task* a passare in stato di *running* è il primo che deve uscire dalla lista non vuota a priorità maggiore. Il *task* che è stato nella lista per più tempo è il prossimo ad uscire, ovvero la lista assume una logica di tipo LIFO. In una lista possono entrare *task* che hanno subito una *preemption*, una *Release* o una *activate*. La Figura 3.4 mostra un esempio di funzionamento dello schedulatore per la gestione della priorità tra i *task*. Si vede che ci sono numerosi *task* in stato di **ready**, per esempio 3 *task* sulla lista con livello di priorità '3', 1 *task* sul livello '2', 1 *task* sul livello '1' e '2' *task* sul livello '0'. La lista con livello '15' ha la priorità maggiore, mentre quella con livello '0' ha la priorità minore. Per una questione di efficienza, non è supportato l'uso di una gestione dinamica delle priorità, quindi, ad ogni *task* è staticamente assegnata una priorità che non può cambiare durante l'esecuzione. I *task* che hanno atteso per un tempo maggiore sono quelli raffigurati più in basso nelle code. Il processore ha appena terminato l'esecuzione di un *task* e lo schedulatore seleziona il prossimo *task* che deve essere eseguito scegliendo l'ultimo tra i *task* nella coda a priorità maggiore. Lo schedulatore farà eseguire i *task* della coda a priorità inferiore non appena si svuoteranno tutte le code di priorità superiore. Per determinare quindi quale processo sarà eseguito, viene applicato il seguente algoritmo:

- lo schedulatore controlla se ci sono *task* negli stati *ready* o *running*;
- dalla lista di *task* in stato di *ready* o *running* determina il gruppo di *task* a priorità maggiore;
- tra i *task* presenti nel gruppo, fa eseguire quello che è in stato di *ready* da maggior tempo.

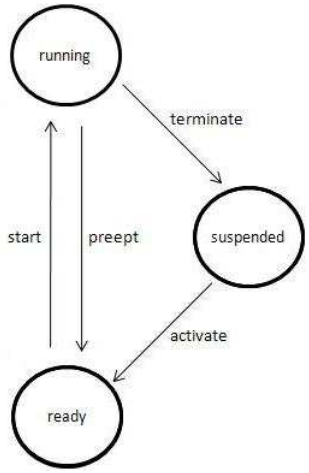


Figura 3.3: Modello a stati dei task base

3.2.5 La politica di schedulazione

La schedulazione di tipo non pre-emptive

La politica di schedulazione prende il nome di *non pre-emptive* se la commutazione tra i *task* avviene esclusivamente ed esplicitamente tramite una chiamata di sistema. La schedulazione *non pre-emptive* impone particolari vincoli per quanto riguarda le tempistiche dei *task*. Per esempio, se un *task* a priorità più bassa è in stato di **running** ed un altro *task* a priorità più alta è in stato di *ready*, si ha che il *task* a priorità minore ritarda l'esecuzione di un *task* a priorità maggiore fino a quando non ha terminato l'esecuzione.

Nella Figura 3.5 il *task* a priorità più bassa T2 ritarda l'escuzione del *task* T1 che ha la priorità maggiore, finché non viene eseguito un comando che obbliga la rischedulazione, in questo caso la chiusura del *task* T2.

I comandi che provocano la rischedulazione **non pre-emptive** sono le seguenti chiamate di sistema:

- il termine di un *task*;
- il termine di un *task* con richiesta esplicita di attivare un nuovo *task*;
- la richiesta esplicita ad effettuare una schedulazione;
- una transizione nello stato di *waiting*.

La schedulazione di tipo full pre-emptive

La schedulazione *full pre-emptive* si ha quando un *task*, attualmente in stato di *running*, può essere rischedulato, alla fine di una sua qualsiasi istruzione, a discrezione dello

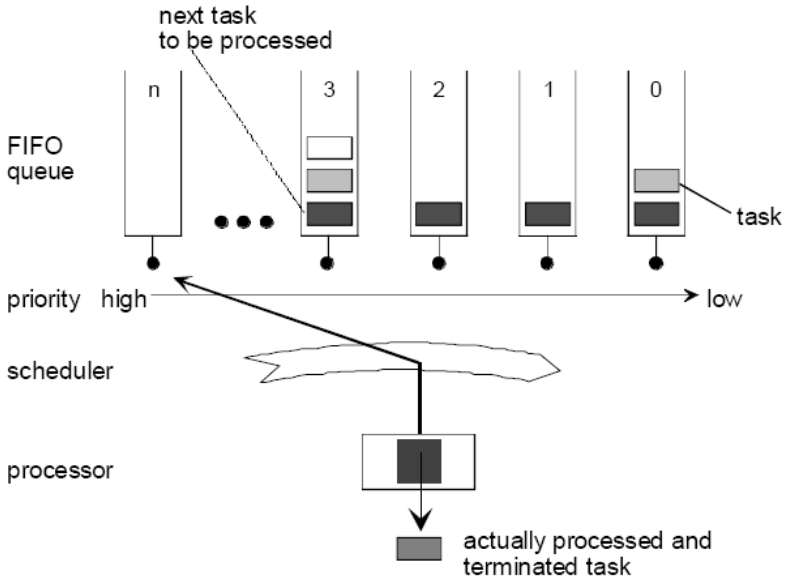


Figura 3.4: Schedulatore: Ordine di esecuzione dei task

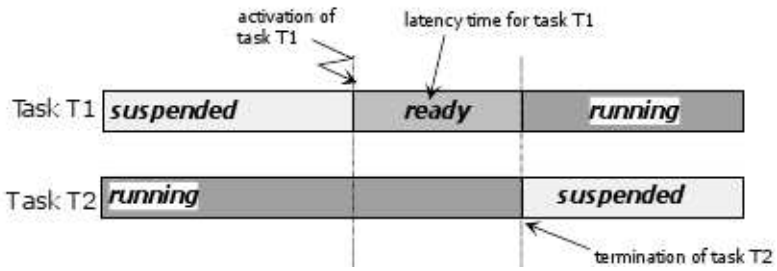


Figura 3.5: Schedulazione non pre-emptive

schedulatore. Esso, infatti, può passare dallo stato di *running* allo stato di *ready* non appena un altro *task*, a priorità maggiore, assume lo stato di *ready* che verrà subito commutato allo stato di *running*. Il contesto del *task* che subisce la *pre-emption* viene salvato e l'esecuzione del codice riprenderà dal punto in cui era stato interrotto non appena il processo tornerà in stato di *running*.

Nella schedulazione *full pre-emptive* il tempo di latenza di un *task* a priorità maggiore è indipendente dall'esecuzione dei *task* a priorità minore.

La Figura 3.6 mostra un *task* T2 con la priorità minore che subisce la *pre-emption* da parte del *task* T1 che ha la priorità maggiore. Si può notare che il *task* T2 non produce alcun ritardo per quanto riguarda l'esecuzione del *task* T1.

Nel caso di schedulazione *full pre-emptive* l'utente deve costantemente tenere a mente che è sempre possibile che avvenga la *pre-emption* in un *task* in stato di *running*. Se è

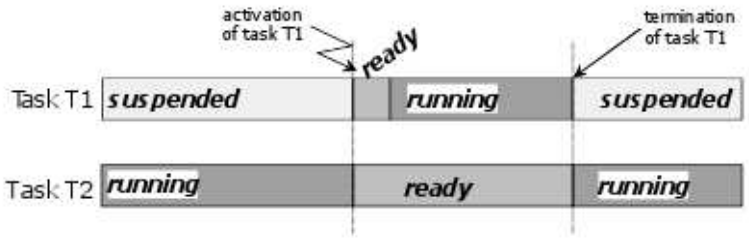


Figura 3.6: Schedulazione full pre-emptive

necessario che un *task* non subisca *pre-emption* in una certa porzione di codice, si possono usare i servizi di sistema per la gestione delle risorse (vedi capitolo 3.2.8).

La rischedulazione di tipo *full pre-emptive avviene* nei seguenti casi:

- termine di un *task*;
- chiusura di un *task* con esplicita richiesta di attivazione di un nuovo *task*;
- esplicito uso del comando *WaitEvent* che comporta il passaggio del *task* dallo stato di running allo stato di waiting;
- se un *task* in stato di (*WaitEvent*) passa in stato di *ready*, poiché è stato generato un evento;
- rilascio di una risorsa;
- ritorno da un processo di interrupt ad un processo di *task*.

Durante l'esecuzione di un processo di interrupt (ISR) la politica di schedulazione viene sospesa fino alla terminazione della ISR.

La schedulazione pre-emptive mista

Se la schedulazione *full pre-emptive* e quella *non pre-emptive* sono presenti in uno stesso sistema operativo, allora si parla di schedulazione *pre-emptive mista*. In questo caso la politica di schedulazione dipende dalle impostazioni di ogni *task*. Se un *task* in stato di *running* è impostato come *non pre-emptive*, allora gli verrà applicata la politica di schedulazione *non pre-emptive*, viceversa per i *task* in *running* impostati come *pre-emptive*.

Chiusura di un *task*

Nel sistema operativo OSEK un *task* può solamente terminare se stesso, cioè non è previsto che un *task* ne faccia terminare un altro; tuttavia è possibile utilizzare il comando *ChainTask* per fare in modo che il *task* che lo invoca venga chiuso e che un determinato *task* si attivi al suo posto. La possibilità di usare questo comando sullo stesso *task* provoca la sua chiusura e la successiva riattivazione; questa operazione ha lo scopo di porre il *task* all'inizio della lista di priorità.

3.2.6 I processi di interrupt ISR

I processi di *interrupt* (ISR) sono porzioni di codice che vengono eseguite ogni qualvolta avviene un certo interrupt nel sistema. Durante l'esecuzione di una ISR la rischedulazione viene sospesa e riprende non appena termina la ISR invocata. Il numero massimo di priorità tra gli interrupt dipende dall'implementazione e la schedulazione degli *interrupt* dipende dall'*hardware* e non è specificata nelle norme OSEK.

3.2.7 Il meccanismo degli eventi

Il meccanismo degli eventi è un metodo di sincronizzazione tra i *task*, esso è implementato soltanto per i *task* estesi e determina la transizione dei *task* da e verso lo stato di *waiting*.

Gli eventi sono segnali gestiti dal sistema operativo, che possono essere assegnati staticamente ad uno specifico *task* esteso detto proprietario del *task*. Ogni *task* possiede un numero finito di eventi ed un singolo evento è univocamente determinato dal suo nome e dal suo possessore. Un evento può essere generato o disabilitato e, quando un *task* viene attivato, il sistema operativo disabilita tutti gli eventi di cui il *task* è il proprietario. I *task* e le ISR possono abilitare tutti gli eventi degli altri *task* estesi che non sono in stato *suspended*, ma soltanto i *task* che ne sono proprietari possono disabilitarli.

Gli eventi sono il criterio per il quale avvengono le transizioni dei *task* estesi dallo stato di *running* a quello di *waiting* e da quello di *waiting* a quello di *ready*. Il sistema operativo è dotato di alcune API per abilitare, disabilitare, mettere in attesa (*waiting*) ed effettuare interrogazioni sugli eventi. Un *task* esteso nello stato di *waiting* può passare allo stato di *ready* se almeno uno degli eventi per il quale si è messo in attesa viene abilitato. Se un *task* in stato di *running* cerca di porsi in *waiting* per un evento che è già stato abilitato, esso permane in stato di *running*.

La Figura 3.7 mostra il processo di sincronizzazione tra *task* con l'uso degli eventi con la politica di schedulazione *full pre-emptive*. Il *task* T1, che è in stato di *waiting* per un evento, ha priorità maggiore del *task* T2 che è in stato di *running*. T2, ad un certo punto, genera l'evento T1 e lo schedulatore fa passare lo stato del *task* T1 da *waiting* a *ready*. Data la priorità maggiore del *task* T1, il *task* T2 subisce la pre-emption passando

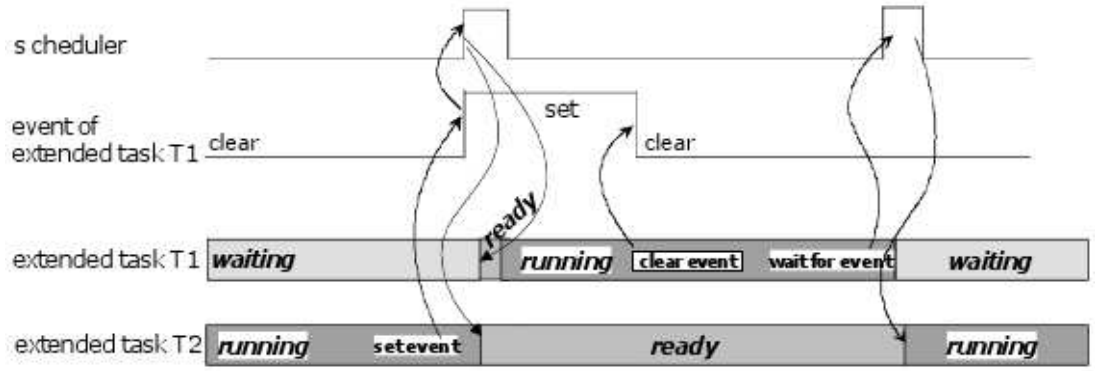


Figura 3.7: Sincronizzazione dei *task* in modalità full pre-emptive

allo stato di *ready* e il *task* T1 passa allo stato di *running* per disabilitare l'evento ed eseguire il proprio codice fino a quando il *task* T1 si pone nuovamente in stato di *waiting* in attesa di un evento. T2 può tornare allo stato di *running* e continuare ad eseguire il proprio codice dal punto in cui era stato interrotto.

3.2.8 La gestione delle risorse

La gestione delle risorse è usata per coordinare accessi concorrenti da parte di più *task* con diversa priorità ad una stessa risorsa condivisa del sistema, ad esempio l'accesso alla memoria o alle risorse *hardware*. L'uso della gestione delle risorse può essere usato anche per gestire l'accesso concorrente ad una stessa risorsa tra *task* e ISR. Il sistema operativo OSEK, gestice le risorse attraverso l'uso dei semafori binari: ad ogni risorsa viene assegnata una variabile che può valere '0' se la risorsa è libera, oppure '1' se la risorsa è occupata. La gestione delle risorse assicura che:

- due *task* o le ISR non possano avere accesso ad una stessa risorsa nello stesso istante;
- che non avvenga l'inversione di priorità (vedi cap. 3.2.11);
- non possano verificarsi eventi '*deadlock*' (vedi cap. 3.2.11);
- l'accesso alle risorse non avvenga in stato di *waiting*.

Le funzionalità della gestione delle risorse devono essere implementate quando:

- si usa la politica *full pre-emptive*;
- si usa la politica non *pre-emptive* e le risorse devono rimanere occupate anche dopo il comando di *rescheduling*;

- si usa la politica non *pre-emptive* e l'utente vuole far eseguire il codice applicativo con un'altra politica di scheduling;
- si ha la gestione condivisa di una stessa risorsa tra più *task* o tra *task* ed ISR.

3.2.9 Funzionamento dell'accesso di una risorsa occupata

Il sistema operativo OSEK segue il *Protocollo di massima priorità OSEK* che assicura che non si verifichi mai il caso in cui un *task* o una ISR abbia accesso ad una risorsa già occupata. Se il concetto di gestione delle risorse è usato per il coordinamento tra *task*, il sistema operativo OSEK assicura che, anche se si verifica l'esecuzione di una ISR, essa può essere processata solo se fa uso di risorse che non sono già occupate da altri processi. Inoltre, il sistema operativo OSEK proibisce l'uso ricorsivo delle risorse.

3.2.10 Restrizioni nell'uso delle risorse

È necessario seguire alcuni accorgimenti nell'impiego delle risorse condivise:

- durante l'occupazione di una risorsa non si possono usare le seguenti API: *TerminateTask*, *ChainTask* e *WaitEvent*;
- le ISR non devono terminare senza aver rilasciato la risorsa che stavano occupando;
- nel caso di multipla occupazione di risorse in un unico *task*, l'utente deve rilasciare le risorse seguendo il principio LIFO[2].

3.2.11 Problemi tipici del meccanismo di sincronizzazione

L'inversione di priorità

Un tipico problema del meccanismo di sincronizzazione è l'inversione della priorità, che avviene quando un *task* con priorità minore, facendo uso di semafori, ritardi l'esecuzione di un altro a priorità maggiore. Per evitare questo fatto bisogna operare come descritto nel *Protocollo di massima priorità OSEK*. Un esempio di inversione di probabilità, con una tipica sequenza di accesso da parte di due *task* a un semaforo in un sistema *full pre-emptive*, è illustrato nella Figura 3.8. Il *task* a priorità più alta è T1, seguono poi T2, T3 e infine quello con priorità minore è T4. Quest'ultimo occupa il semaforo S1 e subito dopo subisce una *pre–emption* da parte di T1, il quale richiede anch'esso l'accesso al semaforo S1. Il *task* T1 quindi, essendo la risorsa occupata, entra nello stato di waiting. Nel frattempo i *task* T2 e T3 sono passati in stato di ready ed essendo a priorità maggiore

[2]LIFO: Last In First Out, La risorsa occupata per ultima deve essere rilasciata per prima, le altre di conseguenza

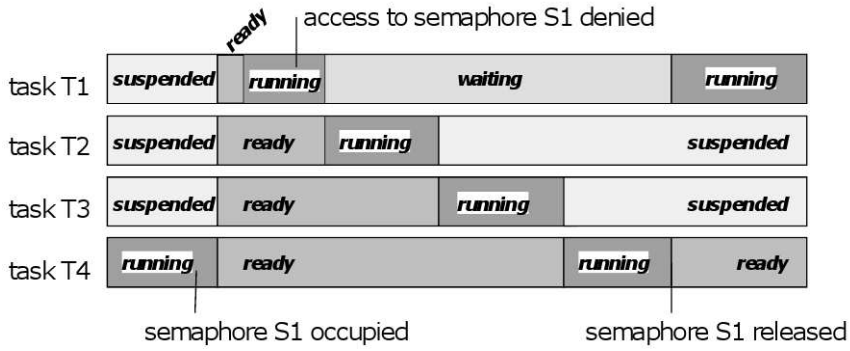


Figura 3.8: Inversione di priorità durante l'occupazione di un semaforo

di T4 passano in stato di running ritardando l'esecuzione di T4. Non appena T3 passa in stato di waiting T4 può passare in running e rilasciare il semaforo S1. Il *task* T1 può tornare in stato di running occupando il semaforo S1. Si nota da questo esempio che il *task* a priorità minore T4, ma anche i *task* T2 e T3 che non usano la risorsa S1, ritardano l'esecuzione di un *task* con priorità maggiore finché S1 non viene rilasciato.

I Deadlock

Un altro tipico problema del meccanismo di sincronizzazione a causa dell'errato uso dei semafori è il cosiddetto *deadlock*. Un esempio di questa condizione è illustrata nella Figura 3.9.

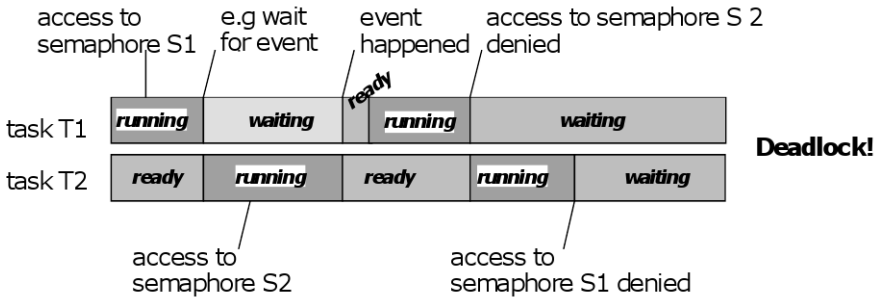


Figura 3.9: Situazione di deadlock usando i semafori

Lo scenario mostrato nella Figura 3.9 è un esempio di *deadlock*. Il *task* T1, che ha la priorità maggiore, occupa il semaforo S1 ed in seguito si pone in *waiting* in attesa di un certo evento. A questo punto il *task* T2 passa in *running*, occupa il semaforo S2 e subisce la *pre-emption* da parte di T1 poiché è stato abilitato l'evento che stava aspettando. T1, in stato di *running*, cerca di accedere al semaforo S2 e visto che è occupata si pone in

stato di *waiting* in attesa che si liberi. T2 cerca di accedere a S1 ma, essendo la risorsa occupata si pone in stato di *waiting* in attesa che si liberi. La situazione così creata non ha modo di risolversi in quanto T1 aspetta che T2 rilasci il semaforo S2 e T2 aspetta che T1 rilasci il semaforo S1. Questa è una tipica situazione di *deadlock*.

3.2.12 Il protocollo di massima priorità OSEK

Per evitare i problemi descritti di inversione della priorità e di *deadlock* il sistema operativo OSEK prevede che si adoperino le seguenti regole nella realizzazione dei *task*:

- durante la fase di progettazione del sistema bisogna assegnare staticamente alla risorsa un livello di priorità. Tale livello deve essere maggiore della priorità dei *task* che utilizzeranno quella risorsa e minore dei *task* che non la utilizzeranno.
- Se un *task* con una certa priorità accede ad una risorsa con livello di priorità maggiore, allora la priorità del *task* aumenta automaticamente al livello di priorità della risorsa che sta utilizzando.
- Se il *task* rilascia la risorsa, la priorità del *task* torna automaticamente al livello che aveva prima di occuparla.

I *task* che potrebbero occupare la stessa risorsa di un altro *task* in *running* non hanno la possibilità di passare in *running* poiché si trovano nella situazione di essere con una priorità inferiore e quindi non possono eseguire la *pre-emption* sul *task* attualmente in *running*. Se la risorsa occupata da un *task* viene liberata, un altro *task* potrebbe richiederla e passare in stato di running.

La Figura 3.10 mostra un esempio del meccanismo del protocollo di massima priorità. Il *task* T0 ha la massima priorità e il *task* T4 ha quella minore e sia T1 che T4 vogliono accedere alla stessa risorsa. Si nota che non vi è alcuna situazione di inversione di priorità, infatti il *task* T1 aspetta per un breve periodo che la risorsa venga rilasciata e non appena questo avviene può passare in stato di running ed occuparla a sua volta senza aspettare che anche i *task* T2 e T3 vadano in stato di waiting. Si vede quindi, a differenza dello scenario descritto in Figura 3.8, che il tempo di attesa si è molto ridotto.

3.2.13 Gli Allarmi

Il sistema operativo OSEK è dotato di un servizio per la generazione di eventi ricorrenti che possono essere generati da allarmi prodotti dallo scadere di un timer. Mediante l'uso di API opportune si possono generare degli allarmi, che si attivano ciclicamente dopo un tempo stabilito, che provocano degli eventi necessari ai *task* per la sincronizzazione.

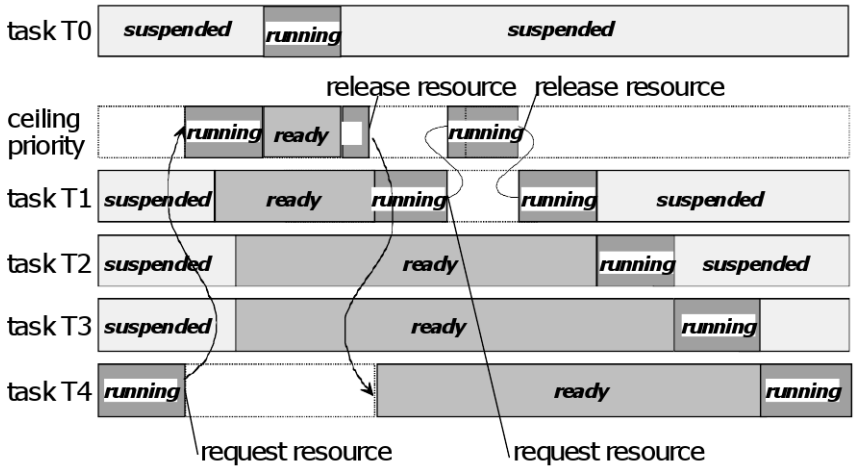


Figura 3.10: Assegnamento di una risorsa con il protocolo di massima priorità tra due *task* usando la politica *pre-emptive*

3.2.14 I contatori

I *tick* di sistema sono il modo mediante il quale il sistema operativo scandisce il tempo e vengono contati da delle variabili denominate contatori. Quando un contatore raggiunge un valore impostato via *software*, si ha lo scadere del timer associato e quindi la generazione dell'allarme.

3.2.15 La gestione degli allarmi

Il sistema operativo OSEK, all'avvio, imposta il valore zero al contatore di sistema e viene incrementato ad ogni *tick*; quando il contatore raggiunge un certo valore, l'allarme scade. Il valore del contatore che determina la scadenza di un allarme può essere determinato in modo relativo o in modo assoluto. Il modo relativo imposta l'attuale valore del contatore di sistema come punto di inizio del conteggio dell'allarme, il quale dovrà scadere dopo un certo numero di *tick*. Il modo assoluto consiste nel far scadere l'allarme quando il contatore di sistema raggiunge un certo valore, indipendentemente dal valore che assumeva lo stesso al momento della chiamata di sistema. Il sistema operativo OSEK prevede dei servizi per attivare *task* o generare degli eventi quando un allarme scade. Gli allarmi possono essere impostati in modo che avvengano singolarmente o ripetutamente. Il sistema operativo OSEK è provvisto inoltre di API per la cancellazione degli allarmi precedentemente abilitati. Ad ogni allarme da dichiarare bisogna assegnare staticamente l'evento che deve generare ed il *task* proprietario dell'evento.

3.3 Scelta del Sistema Operativo per AtmoCube

I microcontrollori prodotti precedentemente a quelli della famiglia PIC18 non avevano la possibilità di sviluppare un nucleo *Multi-Tasking* poiché possedevano un'unica pila di chiamate di funzione. Dato che in un sistema Multi-Tasking, ogni *task* ha una propria pila di chiamate di funzione, non era possibile far coabitare insieme numerosi *task*.

I microcontrollori della famiglia PIC18, invece, permettono la manipolazione della pila *hardware* delle chiamate di funzione; ciò ha permesso lo sviluppo dei sistemi operativi *multitasking* per i microcontrollori.

In questa sezione verranno analizzati i Sistemi Operativi 'Salvo RTOS', μC/OS–II e PICOS18 che rappresentano una panoramica sullo stato dell'arte dei Sistemi Operativi per i microcontrollori PIC18. Verranno in seguito elencate le motivazioni che hanno portato alla scelta di PICOS18 per implementare le funzionalità di AtmoCube.

3.3.1 Salvo RTOS

Salvo [9] è un sistema operativo *real-time* multitasking *cooperative* sviluppato dalla società americana 'Hi–Tech'. Ad ogni *task* viene assegnato un livello di priorità, il cui valore può andare da un minimo di '0' ad un massimo di '15'. Nel caso in cui due *task* abbiano la stessa priorità viene utilizzata la strategia *round-robin*. La gestione degli eventi prevede l'uso di semafori, messaggi e code di messaggi per le comunicazioni tra processi e per la gestione delle risorse. *Salvo* è scritto in ANSI C, limitando il più possibile l'uso dell'*assembly*, permettendo così di essere facilmente portabile su una vasta gamma di processori. I *task* vengono dichiarati staticamente assieme al nucleo durante la fase di compilazione e non possono cambiare le loro caratteristiche durante l'esecuzione. Nell'implementazione dei *task*, bisogna sempre tenere conto della caratteristica cooperativa del sistema *multitasking* e quindi fare in modo che i *task* cedano il controllo della CPU allo schedulatore. Il numero di *task* gestibili da *Salvo* dipende esclusivamente dalla memoria disponibile.

Salvo è un sistema operativo a pagamento, ma 'Hi–Tech' ha reso disponibile una versione *freeware* denominata *Salvo Lite*, che però è caratterizzata da alcune limitazioni nel numero di *task* possibili e di eventi gestibili.

3.3.2 Il sistema operativo μC/OS–II

μC/OS–II [10] è un sistema operativo *real-time* multitasking *pre–emptive* sviluppato dalla società americana 'Micrium Inc.'

I *task* vengono dichiarati staticamente assieme al nucleo durante la fase di scrittura del codice e non possono cambiare durante la sua esecuzione. Il numero massimo di *task* implementabili è 63 e il sistema *multitasking* si basa sul livello di priorità assegnato ad ognuno di essi. Non è previsto il caso in cui due *task* abbiano lo stesso valore di priorità

e quindi non è implementata la gestione *round-robin*. La gestione degli eventi prevede l'uso di semafori, messaggi e code di messaggi.

μC/OS–II può essere eseguito su una vasta gamma di architetture, tra le quali i PIC18 della 'Microchip', i microcontrollori Freescale della 'Motorola' e i microprocessori ARM7.

La licenza di questo sistema operativo è proprietaria, ma è possibile acquistarne i sorgenti, scritti in ANSI C, per scopi didattici.

3.3.3 Il Sistema Operativo PICOS18

PICOS18 [3] è un Sistema Operativo *Real-Time Multi-Tasking Pre-emptive* creato dalla società PRAGMATEC appositamente per i microcontrollori della 'Microchip' PIC18 secondo le norme OSEK/VDX. Il nucleo, scritto in ANSI C, è *open-source* ed è distribuito gratuitamente sotto licenza *Generic Public License* (GPL). Il nucleo del Sistema Operativo PICOS18 è di tipo modulare, nel senso che l'accesso specifico alle risorse (driver, file system manager, etc) può essere realizzato attraverso dei *task* indipendenti dal nucleo.

Il nucleo di PICOS18 ha le seguenti caratteristiche:

Figura 3.11: Servizi offerti dal nucleo PICOS18

- il '**cuore del nucleo**' (*Init+Scheduler+Task Manager*): ha la responsabilità di gestire i tesk dell'applicazione e quindi di determinare il prossimo *task* da fare eseguire in base al suo stato ed alla sua priorità.
- Il **gestore degli allarmi e dei contatori** (Alarm Manager): ha il compito di aggiornare i contatori degli allarmi e dei vari *task* associati ogni volta che avviene un interruzione del *TIMER0*.
- Le **Hook Routines**: sono dei servizi messi a disposizione dell'utilizzatore per agevolare l'operazione di sviluppo del codice, in quanto forniscono informazioni utili allo sviluppatore per effettuare l'operazione di *debugging*.

- Il **gestore dei processi** (*Process Manager*): è un servizio del nucleo che ha il compito di offrire all'applicazione le funzioni necessarie alla gestione degli stati (cambiamento dello stato di un *task*, concatenazione dei *task*, attivazione di un *task*...).
- Il **Gestore degli Eventi** (*Event Manager*): è un servizio che consente all'applicazione di usufruire delle funzionalità necessarie alla gestione degli eventi di un *task*.
- Il **Gestore delle Interruzioni** (*INT Manager*) offre all'applicazione le funzioni necessarie all'attivazione ed alla disattivazione delle interruzioni di sistema.

3.3.4 La scelta di PICOS18 per AtmoCube

Nelle sezioni precedenti è stata fornita una breve panoramica dei principali sistemi operativi per la famiglia PIC18F. Nella Tabella 3.4 verranno elencate le caratteristiche principali caratteristiche dei sistemi operativi citati; in questo modo risulta più semplice fare un confronto tra i tre.

Tabella 3.4: Confronto tra i sistemi operativi per PIC

	Salvo RTOS	**PICOS18**	**μOS–II**
Real Time	Si	Si	Si
Scheduling	Cooperativo con 15 livelli di priorità	Pre–emptive con 15 livelli di priorità	Pre–emptive con 63 livelli di priorità
Numero massimo di *task*	Dipende dalla memoria disponibile sul μC	16	63
Gestione degli Eventi	Semafori, messaggi, code di messaggi	Semafori	Semafori, messaggi, code di messaggi
Occupazione di memoria ROM del nucleo	7Kbytes	5Kbytes	9Kbytes
Architetture supportate	Vasta gamma (PIC,Pentium,etc.)	PIC 18,24,30,33	Vasta gamma (PIC, Freescale, ATR, etc.)
Licenza	A pagamento	GPL	A pagamento

Nella scelta del sistema operativo di AtmoCube, sono state considerate le seguenti specifiche:

- semplicità di scrittura del codice;
- per l'applicazione di AtmoCube non sono necessari più di 10 *task*;
- necessità di utilizzare il minor numero di risorse possibili;
- uso di sorgenti *freeware*.

Date le caratteristiche dei vari sistemi operativi e date le specifiche per l'implementazione di AtmoCube si è scelto di usare il nucleo del Sistema Operativo PICOS18. Questo perché:

- essendo un sistema *multitasking pre–emptive*, non serve, durante la scrittura del codice, preoccuparsi che i *task* cedano il controllo della CPU;
- la licenza GPL fornisce i sorgenti *freeware* e quindi senza alcun costo;
- l'uso delle risorse da parte del nucleo è minimo.

3.3.5 Le API di PICOS18 usate per il Sistema Operativo di AtmoCube

In questa sezione verranno descritte le API [3] che sono state usate per lo sviluppo del Sistema Operativo di AtmoCube.

Tabella 3.5: GetTaskEvent

Prototipo	*StatusType GetTaskState(TaskType TaskID,TaskStateRefType State)*
Descrizione	La funzione ritorna lo stato del *task* richiesto (*TaskType TaskID*).
Parametri In Out	 TaskID: Il numero identificativo del *task* richiesto State: lo stato del *task* richiesto
Codice di ritorno	E_OK se il numero identificativo del *task* richiesto esiste E_OS_IS altrimenti
Commenti	*GetTaskState* ritorna lo stato del *task* in questione al momento in cui la funzione è chiamata.

Tabella 3.6: SetEvent

Prototipo	*StatusType SetEvent(TaskType TaskID,EventMaskType Mask)*
Descrizione	La funzione crea un evento dal nome definito da *EventMaskType Mask* per il *task* il cui identificativo è *TaskType TaskID*
Parametri In Out	 TaskID: Il numero identificativo del *task* a cui associare l'evento Mask: l'identificativo dell'evento
Codice di ritorno	E_OS_IS se il *task* è in stato *suspended* E_OK se il *task* non è in stato di *waiting* per l'evento non ritorna nulla se il *task* è in stato di *waiting* per l'evento
Commenti	Questa funzione è usata per creare un evento verso un altro *task* o ISR. Se il *task* a cui è diretto l'evento è in stato di *waiting* per tale evento, si ha una rischedulazione e se il *task* ha la priorità maggiore, allora passa in stato di *running*. L'identificativo dell'evento creato deve essere un numero che è potenza di 2 e possono esistere al massimo 8 eventi per ogni *task*.

Tabella 3.7: ClearEvent

Prototipo	*StatusType ClearEvent(EventMaskType Mask)*
Descrizione	La funzione disattiva l'evento dal nome definito da *EventMaskType Mask*
Parametri In	 TaskID: l'identificativo dell'evento
Codice di ritorno	E_OK in qualsiasi caso
Commenti	Questa funzione è usata per rimuovere un evento ricevuto da un *task*. Il *task* a cui è riferito l'evento è responsabile della sua cancellazione una volta ricevuto. Se l'evento non viene rimosso, il *task* andrà in un ciclo infinito, è quindi indispensabile che il *task*, una volta ricevuto l'evento atteso, lo cancelli.

Tabella 3.8: WaitEvent

Prototipo	*StatusType WaitEvent(EventMaskType Mask)*
Descrizione	La funzione provoca il cambiamento di stato del *task* che l'ha invocata, da *running* a *waiting*. *EventMaskType Mask* indica la somma 'OR' degli identificativi degli eventi attesi dal *task*.
Parametri In	 Mask: È il numero composto dal'operazione 'OR' fatta da tutti gli identificativi degli eventi attesi dal *task*.
Codice di ritorno	E_OK se nessuno degli eventi attesi è stato già creato. E_OS_ID se l'identificativo non corrisponde a nessun evento associato al *task* non ritorna nulla se uno degli eventi attesi è già stato creato.
Commenti	Se uno degli eventi attesi è già stato creato prima che la funzione sia stata chiamata, si ha una rischedulazione e se il *task* ha la priorità maggiore continua la sua esecuzione. Se, invece, al momento della chiamata della funzione, nessuno degli eventi attesi è stato creato, il *task* passa in stato di *waiting* e l'esecuzione passa ad un altro *task*.

Tabella 3.9: SetRelAlarm

Prototipo	*StatusType SetRelAlarm(AlarmType AlarmID,TickType increment,TickType cycle)*
Descrizione	La funzione programma un allarme che si attiva ad uno specifico intervallo di tempo.
Parametri In In In	 AlarmID: Il numero identificativo dell'allarme da programmare. Increment: è il tempo che deve trascorrere prima della prima attivazione dell'allarme Cycle: è il tempo che deve trascorrere tra un'attivazione e l'altra dell'allarme.
Codice di ritorno	E_OS_STATE se l'allarme è già programmato. E_OS_ID se l'identificativo dell'allarme non corrisponde a nessun allarme. E_OK altrimenti.
Commenti	Questa funzione rende possibile il funzionamento ciclico dei *task*, in quanto ogni volta che un allarme si attiva, questi creano un evento per uno specifico *task*. Il *task* destinatario di tale evento passa quindi in stato di *running* con cadenza regolare.

Tabella 3.10: CancelAlarm

Prototipo	*StatusType CancelAlarm(AlarmType AlarmID)*
Descrizione	La funzione disattiva l'allarme precedentemente creato.
Parametri In	 AlarmID: Il numero identificativo dell'allarme da programmare.
Codice di ritorno	E_OS_NOFUNC se l'allarme è già stato disattivato. E_OS_ID se l'identificativo dell'allarme non corrisponde a nessun allarme. E_OK altrimenti.
Commenti	Questa funzione ferma l'esecuzione continua dell'allarme, blocca quindi la creazione ciclica degli eventi ogni volta che l'allarme si attivava.

Tabella 3.11: GetResource

Prototipo	*StatusType GetResource(ResourceType RedID)*
Descrizione	La funzione è usata per riservare una risorsa come descritto dal *Protocollo di massima priorità OSEK/VDX*
Parametri In	 RedID: Il numero identificativo della risorsa da riservare
Codice di ritorno	E_OS_ACCESS se la risorsa è già stata riservata. E_OS_ID se l'identificativo della risorsa non corrisponde a nessuna risorsa. E_OK se la risorsa è stata assegnata con successo.
Commenti	Questa funzione permette di bloccare l'uso della risorsa ad altri *task* o ISR

Tabella 3.12: ReleaseResource

Prototipo	*StatusType ReleaseResource(ResourceType RedID)*
Descrizione	La funzione è usata per rilasciare una risorsa riservata come descritto dal *Protocollo di massima priorità OSEK/VDX*
Parametri In	 RedID: Il numero identificativo della risorsa da rilasciare
Codice di ritorno	E_OS_ACCESS se la risorsa è già stata rilasciata. E_OS_ID se l'identificativo della risorsa non corrisponde a nessuna risorsa. E_OK se la risorsa è stata rilasciata con successo.
Commenti	Questa funzione permette di sbloccare l'uso della risorsa ad altri *task* o ISR

Capitolo 4

Problematiche riscontrate durante lo sviluppo del Sistema Operativo e soluzioni proposte

Durante lo sviluppo del Sistema Operativo per AtmoCube, nel rispettare le specifiche di funzionamento del satellite, sono insorti alcuni problemi implementativi. In questo capitolo verranno analizzate tutte le problematiche riscontrate ed illustrate le soluzioni adottate.

4.1 Reperimento delle informazioni di *HouseKeeping*

Come si è visto nel capitolo 2.6, i dati che costituiscono gli *HouseKeeping* provengono da un consistente numero di conversioni ADC. Dall'analisi dello schema elettrico del modulo OBDH si nota che l'*hardware* non è adatto ad eseguire tutte le conversioni ADC necessarie poiché non ci sono abbastanza linee disponibili. È stato deciso, quindi, che il microcontrollore della scheda adibita all'*Attitude Determination Control* esegua le conversioni ADC per conto del PIC 18LF8722 dell'OBDH. Per consentire il trasferimento delle informazioni tra i due microcontrollori, si è reso necessario modificare lo schema elettrico dell'OBDH raffigurato in Figura 1.3. Essendo il *bus* SPI dell'OBDH condiviso tra più periferiche, si è deciso di sfruttare tale architettura di condivisione per aggiungere una nuova porta per la comunicazione con il microcontrollore della scheda adibita all'*Attitude Determination Control*.

La modifica dello schema elettrico della OBDH consiste nell'aggiunta di tre *buffer* (vedi Figura 4.1) che vengono abilitati da una linea in uscita del *decoder* che, in [2], non era stata utilizzata; tale linea viene poi posta in uscita dell'OBDH. Due dei tre *buffer* aggiunti vengono posti come uscita dalla OBDH per fornire al microcontrollore per

l'*Attitude Determination Control* le linee MOSI e SCLK. Il terzo *buffer* viene posto, invece, come ingresso della OBDH per la linea MISO proveniente dal microcontrollore adibito all'*Attitude Determination Control*. Nella configurazione appena descritta per la nuova comunicazione SPI, il PIC18LF8722 è il dispositivo *master*, mentre il microcontrollore della scheda per l'*Attitude Determination Control* è il dispositivo *slave*.

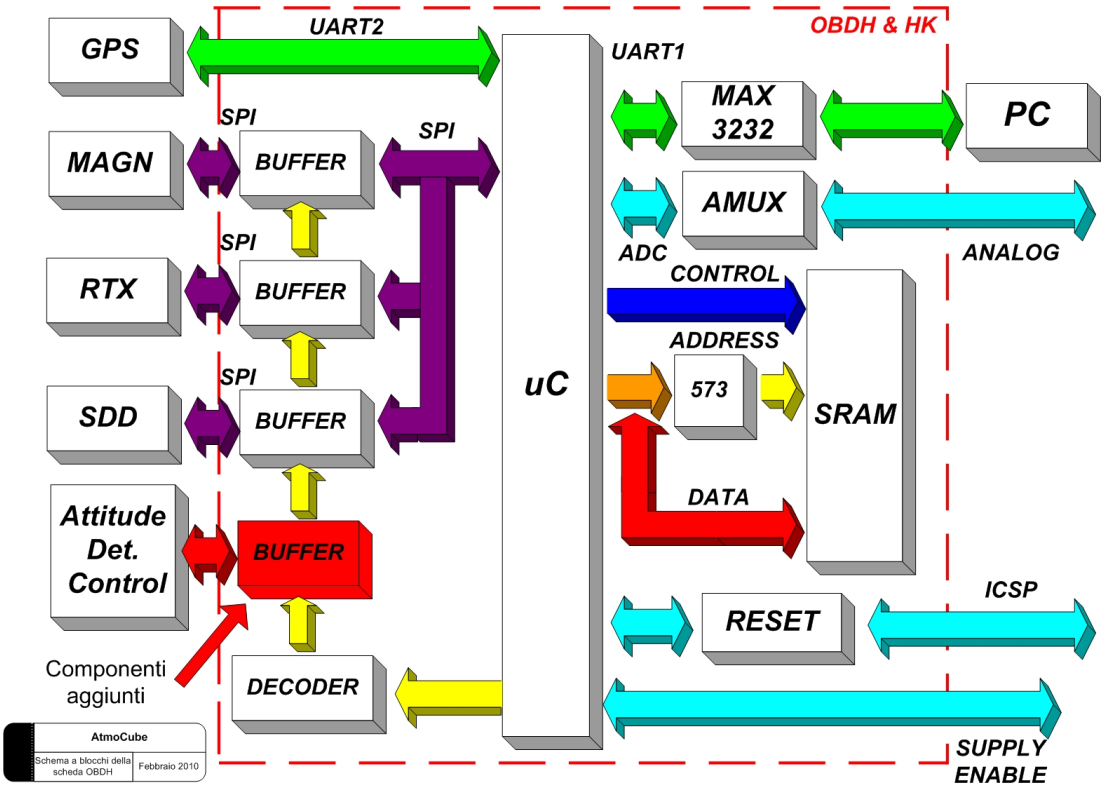


Figura 4.1: Schema a blocchi della OBDH con le modifiche apportate

Le informazioni che il PIC 18LF8722 richiederà al microcontrollore adibito all'*Attitude determination Control* si dividono in 2 categorie: una per la tipologia di *HouseKeeping*–scienza ed una per la tipologia di *HouseKeeping*–sistema 1.

Le informazioni per gli *HouseKeeping*–scienza sono:

- temperatura della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse x;
- temperatura della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse x;
- corrente della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse x;
- corrente della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse x;

- tensione della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse x;
- tensione della cella fotovoltaica sulla secponda faccia del cubo ortogonale all'asse x;
- temperatura della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse y;
- temperatura della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse y;
- corrente della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse y;
- corrente della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse y;
- tensione della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse y;
- tensione della cella fotovoltaica sulla secponda faccia del cubo ortogonale all'asse y;
- temperatura della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse z;
- temperatura della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse z;
- corrente della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse z;
- corrente della cella fotovoltaica sulla seconda faccia del cubo ortogonale all'asse z;
- tensione della cella fotovoltaica sulla prima faccia del cubo ortogonale all'asse z;
- tensione della cella fotovoltaica sulla secponda faccia del cubo ortogonale all'asse z;
- tensioni dei fotodiodi.

Le informazioni per gli *HouseKeeping*–sistema di tipo 1 sono:

- una misura della tensione di 600V;
- una misura della tensione del *frontend*;
- una misura della tensione del sistema di controllo dell'assetto;
- una misura della corrente del sistema di controllo dell'assetto;
- la corrente massima assorbita dal sistema del controllo dell'assetto;
- il guadagno del controllo dell'assetto.

4.2 La sincronizzazione tra le acquisizioni del GPS e il campionamento del magnetometro

Le misure ottenute dal magnetometro, come già accennato nel capitolo 2.5.2, necessitano di essere corredate dall'informazione di posizione del satellite. Dato che il satellite si muove con una velocità media di 8 Km/s, è necessario che il tempo che intercorre tra

l'acquisizione dei dati di PVT del GPS ed il campionamento del magnetometro sia il minore possibile. Se, ad esempio, il campionamento viene fatto 30 ms dopo l'acquisizione del dato PVT, si ha che la misura del campo magnetico viene fatta a 240 metri di distanza da dove è indicato nell'informazione di posizione del pacchetto PVT.

Per assicurare una sincronizzazione perfetta tra l'informazione di posizione contenuta nel pacchetto PVT dello SGR e la misura del campo magnetico, il microcontrollore dovrebbe far eseguire il campionamento del magnetometro nello stesso istante in cui lo SGR acquisisce il dato PVT. Nel capitolo 2.5.1 è stato spiegato che lo SGR fornisce un impulso sulla linea PPS ogni volta che viene acquisito un nuovo dato PVT con un errore di circa $1\mu s$. È stato deciso, quindi, di utilizzare la linea PPS per la temporizzazione del campionamento del magnetometro. Il microcontrollore fornisce il comando di eseguire la misura al magnetometro non appena arriva un impulso sulla linea PPS. Il tempo che intercorre tra la ricezione dell'impulso PPS e l'effettivo campionamento del magnetometro dipende da:

- il tempo necessario affinché il microcontrollore comici la trasmissione del comando di campionamento;
- il tempo per la trasmissione del comando;
- il tempo tra la ricezione del comando da parte del magnetometro e l'effettivo campionamento.

Questi tempi di ritardo sono dovuti all'*hardware* di AtmoCube e non possono essere modificati.

Da quanto detto, risulta impossibile ottenere una sincronizzazione perfetta. Si è cercato, quindi, un valore di tempo massimo tra l'acquisizione della posizione ed il campionamento effettivo del magnetometro. Dalle specifiche [8] dello SGR risulta che il dato di posizione nel PVT, per un uso spaziale del ricevitore GPS, ha un'incertezza di circa ± 10 metri. Si vede quindi che non ha senso cercare una sincronizzazione perfetta tra il dato di posizione e quello del magnetometro, in quanto la misura di posizione può essere già affetta, di per sé, da un errore. Non è, quindi, necessario campionare il campo magnetico nello stesso istante in cui è stato acquisito il dato di posizione, ma basta farlo all'interno dell'intervallo di incertezza descritto nelle specifiche dello SGR. Per la progettazione del Sistema Operativo di Atmocube, è stato deciso che il tempo massimo che intercorre tra l'acquisizione del PVT ed il campionamento debba essere di 1 ms poiché esso corrisponde ad un'incertezza massima di 8 metri.

Per utilizzare la linea PPS come temporizzazione per effettuare il campionamento del magnetometro, si è reso necessario modificare la scheda OBDH in modo da collegare la linea PPS ad un ingresso di *interrupt* del microcontrollore. In questo modo il PIC può ricevere l'impulso PPS in maniera asincrona ed inviare al magnetometro il comando di avvio del campionamento.

Per utilizzare la linea di *interrupt* Int0 del microcontrollore, che, dallo schema elettrico della OBDH illustrato in [2], viene usato come *output* per fornire il comando di *start* allo spettro–dosimetro, sono stati adottati i seguenti cambiamenti:

1. è stato aggiunto l'integrato CD4049, costituito da 6 porte NOT, alla scheda OBDH;
2. è stata spostata la linea di comando per l'estrazione dell'antenna dalla porta RC2 del microcontrollore all'uscita di una porta logica NOT dell'integrato appena aggiunto;
3. l'ingresso della porta NOT è stata collegata con l'uscita $\overline{y5}$ del decoder 74HC238, che era inutilizzata;
4. la linea per il comando di *start* per lo spettro–dosimetro è stata spostata da Int0 a RC2;
5. è stata collegata la linea PPS dello SGR alla porta Int0.

In questo modo, ponendo le linee di ingresso del decoder 74HC238 A0, A1, A2 rispettivamente in stato *High*, *Low* e *High*, il decoder fornisce un'uscita *Low* alla porta $\overline{y5}$ che viene invertita dalla porta NOT in modo che il comando di estrazione dell'antenna sia dato da un impulso di tipo *High*. Il comando di *start* allo spettro–dosimetro deve ora essere fornito dalla porta RC2 e la linea PPS invia un *interrupt* al microcontrollore ogni volta che il GPS ha acquisito un nuovo dato PVT.

4.3 Il problema del SEU

Nel capitolo 2.4.1 è stato spiegato l'effetto di un urto di una particella ionizzata su una memoria SRAM.

I dati scientifici ricoprono un ruolo fondamentale per la missione di AtmoCube ed è quindi importante cercare di mantenerli integri dal momento della loro acquisizione fino alla loro ricezione dalla GS. In questo intervallo di tempo i dati delle misure scientifiche possono essere affetti da qualche SEU, che può causare la modifica di alcuni bit. Durante la missione di AtmoCube, esistono due momenti all'interno dei quali i dati possono essere colpiti da SEU:

1. durante l'acquisizione dello spettro–dosimetro, poiché vengono provvisoriamente salvati nella memoria del FPGA;
2. nel periodo di tempo che intercorre tra la loro acquisizione e la loro trasmissione alla GS, poiché vengono salvati nella memoria SRAM della OBDH.

Come spiegato nel capitolo 2.4.1, il tasso giornaliero di errori dovuti ad eventi SEU per i dati contenuti nelle zone di memoria di AtmoCube è $SEU_{RATE_d} \approx 10^{-5}\ \frac{Bit\ errati}{1\ giorno}$. In base alle specifiche di acquisizione dello spettro–dosimetro, il tempo in cui i dati vengono salvati provvisoriamente nella memoria del FPGA è, al massimo, di un minuto. Quindi la SEU Error Probability per tale intervallo di tempo diviene $Pe_{SEU_m} = \frac{SEU_{RATE_d}}{60*24} = 7 * 10^{-9}\ \frac{Bit\ errati}{minuto}$. Per ridurre ancora la probabilità di errore, lo spettro–dosimetro salva i dati nella memoria del FPGA in tre copie uguali. Lo spettro–dosimetro trasmette le tre copie dei dati appena acquisiti, non appena riceve il comando di *read* inviato dal microcontrollore. Il microcontrollore, a questo punto, deve:

- salvare momentaneamente le tre copie dei dati ricevuti dallo spettro–dosimetro;
- effettuare, per ogni bit, un controllo 'a maggioranza' tra le tre copie;
- salvare i bit ottenuti dal controllo nella memoria SRAM, per essere poi trasmessi alla GS.

Si calcola ora la probabilità che il controllo a maggioranza sbagli, ovvero quando due o tutte e tre le copie siano errate. Supponendo che i SEU siano eventi indipendenti si ha che la probabilità di avere un bit errato con il (codice a maggioranza (CM)) è:

$$Pe_{SEU_m\ CM} = \binom{3}{2} * Pe^2_{SEU_m} * (1 - Pe_{SEU_m}) + Pe^3_{SEU_m} = 1,47 * 10^{-16} \quad (4.1)$$

La probabilità che un bit salvato in memoria SRAM della OBDH, in attesa di essere trasmesso alla GS subisca un SEU, dipende dalla lunghezza del periodo di tempo che intercorre tra il suo salvataggio e la sua trasmissione. Bisogna quindi ricavare tale periodo di tempo. Come già detto nei capitoli precedenti, AtmoCube implementa un sistema di tipo *store and forward*, ovvero durante il Gap acquisisce e salva i dati nella memoria e, durante la finestra di accesso, li trasferisce alla GS. Supponendo che la trasmissione dei dati verso la GS avvenga senza alcun *overhead* alla velocità di 10Kbit/s e che la durata media della finestra di accesso sia di 929 s, i dati trasmissibili risultano essere pari a 9290 Kbit. Come si vedrà nei capitoli successivi, la porzione di memoria destinata al salvataggio di tali dati è di circa $M_{size} = 7674$ Kbit. Quindi, essendo i dati salvabili in quantità minore rispetto a quelli trasmissibili, si può dire che, nel caso in cui la memoria venisse riempita completamente in un Gap, essa verrebbe svuotata nella finestra di accesso successiva. Pertanto, il periodo massimo in cui un singolo bit di informazione può trovarsi nella SRAM è pari alla durata di un Gap.

Per il calcolo della probabilità di errore, verrà ora preso come tempo di riferimento il valore del Gap medio: $\mathrm{Gap}_{av} = 9674$ s. Dal tasso di errore SEU di $\mathrm{SEU}_{RATE_d} = 1{*}10^{-5}$ bit errati al giorno risulta che la probabilità di errore di un bit in un Gap medio è di

$$Pe_{SEU_{Gap}} = \frac{\mathrm{SEU}_{RATE_d} * Gap_{av}}{24 * 3600} = \frac{1 * 10^{-5} * 9674}{24 * 3600} \approx 1 * 10^{-6} \quad (4.2)$$

La probabilità di errore appena ricavata permette di trovare la probabilità che un byte ($Pe_{byte_{Gap}}$) salvato in memoria venga alterato. Si ha quindi:

$$Pe_{byte_{Gap}} = 1 - (1 - Pe_{SEU_{Gap}})^8 = 1 - (1 - 1 * 10^{-6})^8 \approx 7 * 10^{-6}. \quad (4.3)$$

È interessante calcolare la probabilità che avvenga almeno un SEU tra i 7674 Kbit della memoria ($Pe^{mem}_{SEU_{Gap}}$). Per calcolare tale valore, si ricava la probabilità che non avvenga alcun errore ($Pc^{mem}_{SEU_{Gap}}$) e si calcola la probabilità complementare.

$$Pc^{mem}_{SEU_{Gap}} = (1 - Pe_{SEU_{Gap}})^{M_{size}} = (1 - (1 * 10^{-6}))^{7858176} \approx 3,86 * 10^{-4} \quad (4.4)$$

$$Pe^{mem}_{SEU_{Gap}} = 1 - Pc^{mem}_{SEU_{Gap}} \approx 0,9996 \quad (4.5)$$

Dai calcoli appena descritti, si può notare che, quasi sicuramente, nell'intervallo di tempo di un Gap, i dati in memoria subiscono almeno un SEU.

Per diminuire tale probabilità, si è pensato di salvare i dati nella SRAM utilizzando un codice a correzione di errore e di decodificarli con una tecnica di tipo FEC prima di trasmetterli.

4.3.1 Il codice esteso di Hamming utilizzato per Atmocube

Il codice esteso di Hamming [11] è stato scelto per codificare i dati da salvare nella memoria di Atmocube per i seguenti motivi: permette di correggere 1 bit errato su 4, è facile da implementare in un'architettura di memoria ad 8 bit suddividendo 1 byte dati in 2 mezzi byte e le operazioni di codifica e decodifica richiedono semplici calcoli che non rallentano le altre operazioni del microcontrollore. Il codice esteso di Hamming deriva dal codice Hamming(7,4) che appartiene alla classe dei codici lineari a blocco e permette di correggere un errore ogni 7 bit. Al codice Hamming(7,4) viene aggiunto un bit di controllo della parità ottenendo il codice esteso di Hamming (Hamming(8,4)), che permette di rilevare anche tutti gli errori pari.

La matrice generatrice $\mathcal{G}$ e la matrice $\mathcal{H}$ di controllo di parità del codice utilizzato per AtmoCube sono:

$$\mathcal{G} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{pmatrix}$$

$$\mathcal{H} = \begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

4.3.2 Salvataggio e lettura dei dati nella SRAM

I dati salvati tramite codifica Hamming(8,4), sono costituiti dalle acquisizioni del magnetometro, dello spettro–dosimetro e dalle informazioni di *Housekeeping*. La memoria SRAM della OBDH viene divisa in varie sezioni e la più grande tra queste viene dedicata al salvataggio dei dati sopra menzionati.

In questa sezione viene descritto il processo di salvataggio e di lettura di tali dati utilizzando il codice esteso di Hamming.

Ad ogni byte da salvare viene applicato il seguente algoritmo (vedi anche la Figura 4.2):

1 – ogni byte viene diviso in due mezzi byte;

2 – ogni mezzo byte è codificato con il codice Hamming(8,4), ottenendo così 2 byte di cui il primo costituitito dai primi 4 bit di informazione e dai relativi 4 bit di controllo e il secondo costituito dai secondi 4 bit di informazione e dai relativi 4 bit di controllo.

3 – le i 2 byte vengono salvati nelle locazioni di memoria contigue ad essi riservati.

Per la lettura di un singolo byte di informazione, vengono letti i 2 byte che contengono i primi e i secondi 4 bit di informazione, entrambi con i rispettivi 4 bit di controllo. In base al numero di errori avvenuti, l'operazioni di decodifica del byte salvato potrà avere esiti diversi. Possono capitare 4 casi differenti:

1. non ci sono stati errori;
2. un singolo bit è errato in almeno uno dei due byte salvati;
3. almeno 1 byte salvato possiede un numero dispari (maggiore di uno) di bit errati;
4. almeno 1 byte salvato possiede un numero pari di bit errati.

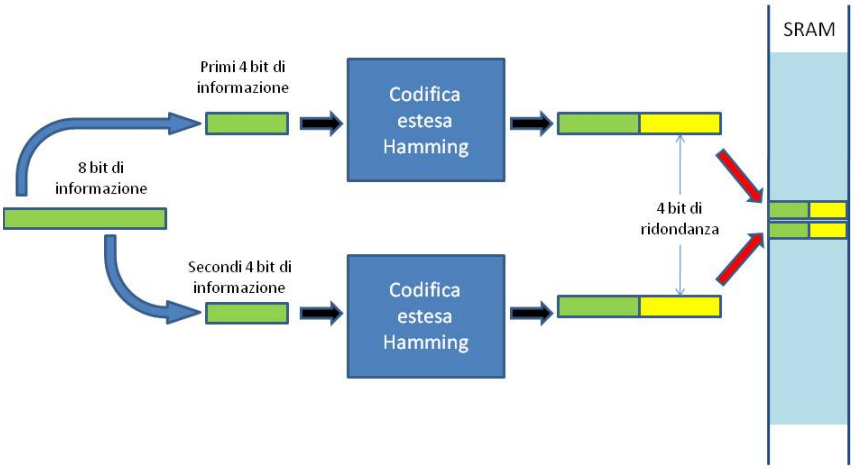


Figura 4.2: Algoritmo di salvataggio di un byte dati con la codifica di Hamming estesa (Hamming(8,4))

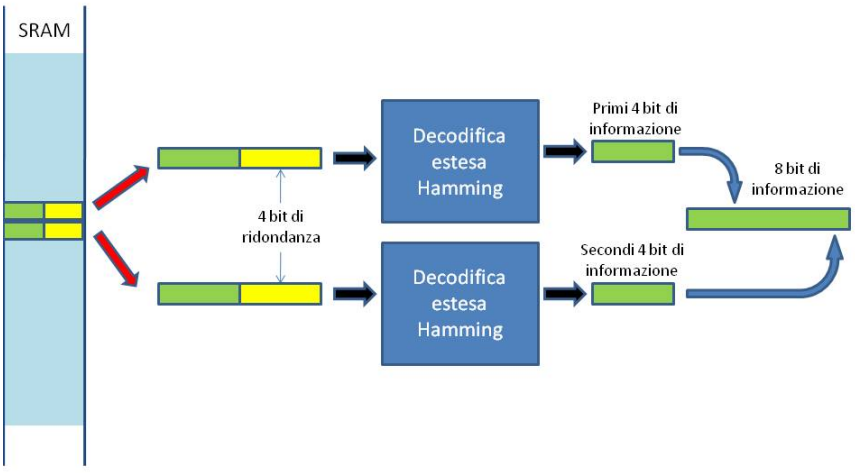


Figura 4.3: Decodifica di un byte di informazione da due byte di dati salvati in assenza di errori.

Nel caso (1) (vedi Figura 4.3), non essendoci stati errori, la decodifica con l'Hamming esteso avviene togliendo i bit di controllo da quelli di informazione e ricomponendo, così, il byte di informazione originario.

Nel caso (2)(vedi Figura 4.4) la decodifica estesa di Hamming rivela l'errore avvenuto e lo corregge. Il byte di informazione viene quindi ricomposto in maniera corretta.

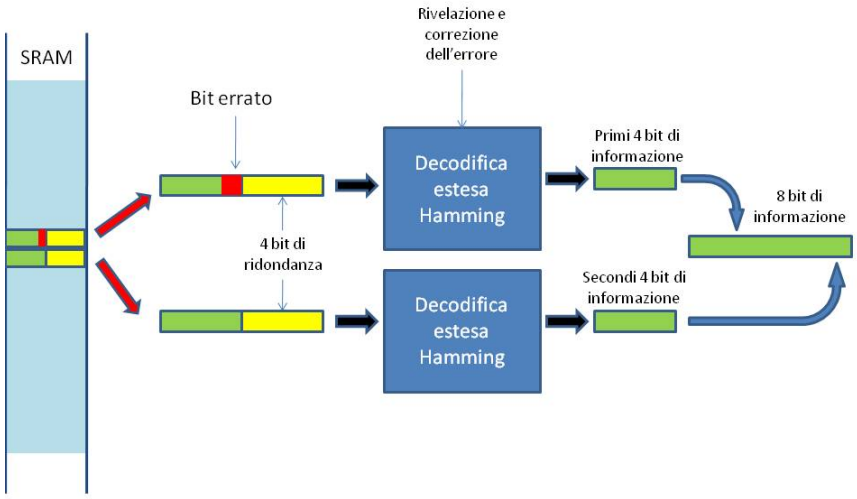


Figura 4.4: Decodifica di un byte di informazione da due byte salvati in presenza di un errore.

Nel caso (3)(vedi Figura 4.5) la decodifica estesa di Hamming deduce erroneamente, che è avvenuto un solo errore e cerca di correggerlo. Il byte di informazione viene quindi ricomposto non correttamente. Nel caso (4)(vedi Figura 4.6), la decodifica estesa di Hamming si accorge che sono avvenuti degli errori pari ed il byte di informazione viene sostituito con un byte pari a '0'.

4.3.3 Calcolo della probabilità di errore utilizzando il codice esteso di Hamming

Dal capitolo precedente, si nota come la decodifica del codice esteso di Hamming fornisca in uscita un byte errato quando si è in presenza di un numero dispari (maggiore di uno) di errori. Per conoscere la probabilità che il codice sbagli, si cerca quindi la probabilità che avvenga un numero dispari di errori (maggiore di 1).

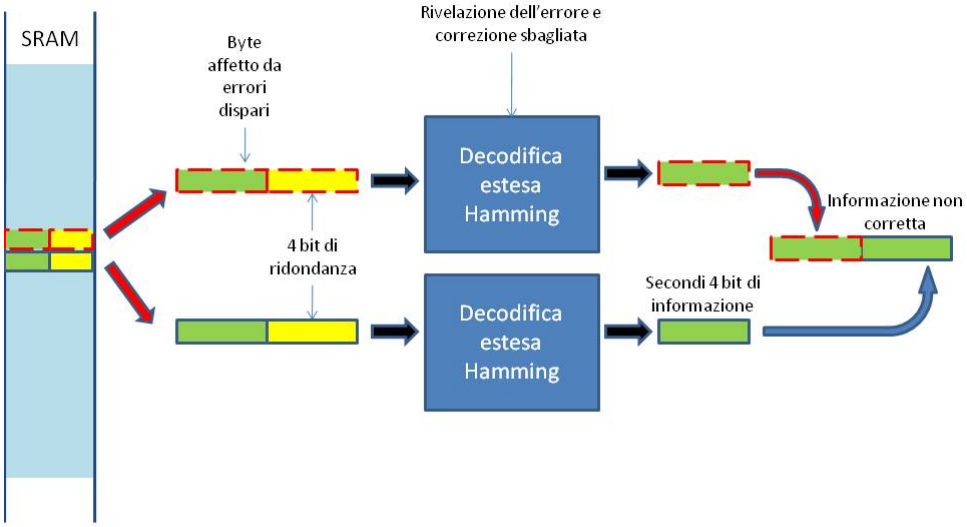


Figura 4.5: Decodifica errata di un byte di informazione da due byte salvati, in presenza di un numero dipari di bit errati.

$$\begin{aligned} Pe_{byte_{H84}} &= \binom{8}{3} * Pe_{SEU_{Gap}}^{3} * (1 - Pe_{SEU_{Gap}})^{5} + \\ &+ \binom{8}{5} * Pe_{SEU_{Gap}}^{5} * (1 - Pe_{SEU_{Gap}})^{3} + \\ &+ \binom{8}{7} * Pe_{SEU_{Gap}}^{7} * (1 - Pe_{SEU_{Gap}}) \approx 6 * 10^{-17} \end{aligned} \tag{4.6}$$

La probabilità che almeno un byte in memoria venga alterato nel corso di un Gap con l'utilizzo della codifica estesa di Hamming si ottiene calcolando la probabilità che non avvenga alcun errore e si ricava la sua probabilità complementare. In questo calcolo, a differenza dell'equazione 4.4, il numero di byte viene raddoppiato poiché si usa una codifica a tasso 1/2.

$$Pc_{byte_{H84}}^{mem} = (1 - Pe_{byte_{H84}})^{2 * M_{size_{byte}}} = (1 - (6 * 10^{-17}))^{\frac{2 * 7858176}{8}} \approx 1 \tag{4.7}$$

Dal calcolo effettuato in 4.7, si ha quasi la certezza che non avvengano errori e quindi la probabilità che almeno un byte venga alterato in un Gap risulta $Pe_{byte_{H84}}^{mem} \approx 0$.

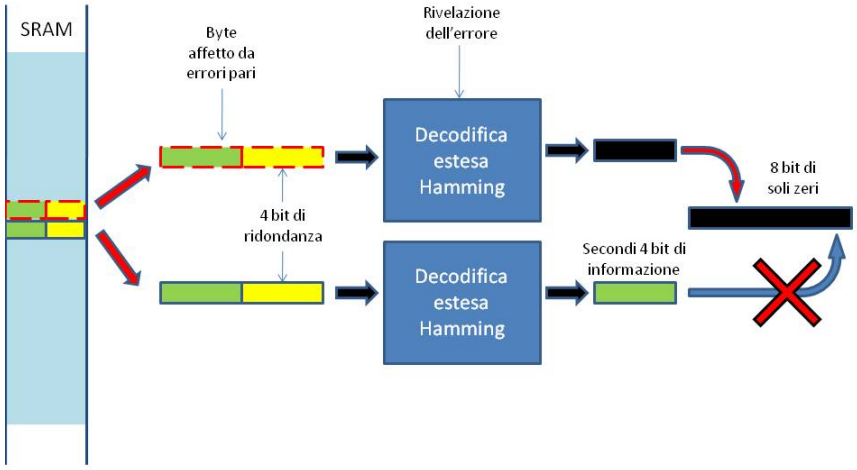


Figura 4.6: Rivelazione di un errore pari e cancellazione del byte di informazione

4.3.4 Considerazioni sulla codifica utilizzata

I calcoli effettuati in 4.3 ed in 4.3.3 sono stati fatti con le ipotesi che gli eventi SEU, e quindi gli errori sui singoli bit, siano indipendenti ed equidistribuiti.

La probabilità che un bit subisca un SEU durante un Gap è abbastanza piccola da poter dire che non ci sarebbe bisogno di applicare alcun codice, ma, analizzando l'intera memoria dati, la probabilità che almeno un bit in memoria subisca un SEU (4.5) non è affatto remota e quindi, per abbassare tale probabilità, è stato applicato il codice a correzione d'errore. Per un confronto tra i risultati ottenuti, nella Tabella 4.1 sono riportati i valori di probabilità con e senza codifica.

	Senza codice	Con codice esteso di Hamming
Probabilità che il singolo byte venga alterato in un Gap	$Pe_{byte_{Gap}} \approx 7 * 10^{-6}$	$Pe_{byte_{H84}} \approx 6 * 10^{-17}$
Probabilità che almeno un byte venga alterato nella SRAM in un Gap	$Pe^{mem}_{SEU_{Gap}} \approx 0,9996$	$Pe^{mem}_{byte_{H84}} \approx 0$

Tabella 4.1: Confronto tra le probabilità di errore utilizzando la codifica FEC estesa di Hamming

4.4 Le risorse energetiche

AtmoCube è dotato di alcune batterie che vengono alimentate dalle celle solari presenti sulle sue facce (vedi capitolo 1). Durante lo svolgimento delle operazioni del satellite può capitare che la carica della batteria scenda al di sotto del limite minimo di tensione di funzionamento del sistema. Per scongiurare questo evento, AtmoCube è stato progettato per limitare i consumi alimentando le periferiche solo nei periodi di tempo strettamente necessari al loro uso. Il sistema Operativo di AtmoCube, inoltre, quando si accorge che la tensione di carica della batteria scende sotto un certo livello, pone il microcontrollore in una condizione di basso consumo e, contemporaneamente, sospende le operazioni che stava eseguendo, disabilitando quindi le alimentazioni a tutte le periferiche. Operando in questo modo, si permette alla batteria di ricaricarsi più velocemente cercando così di scongiurare la sua scarica completa. Quando la tensione di batteria supera una certa soglia, che indica la carica avvenuta, il microcontrollore può tornare ad eseguire le normali operazioni.

Capitolo 5

Implementazione del Sistema Operativo di AtmoCube

Nei precedenti capitoli sono state esposte le operazioni che il satellite dovrà eseguire durante la sua missione ed i problemi riscontrati durante la progettazione del Sistema Operativo. In questo capitolo verranno descritte le soluzioni adottate per soddisfare le specifiche richieste dal progetto. In particolare verrà descritto:

- come vengono strutturati i pacchetti dati da trasmettere alla GS;
- come viene implementato il metodo di trasmissione *stop and wait*;
- come viene organizzata la memoria SRAM;
- le fasi operative del Sistema Operativo.

5.1 I pacchetti di Telemetria

L'insieme di informazioni che devono essere inviate alla GS, può essere diviso in gruppi di dati che possono essere visti come blocchi di informazione indipendenti dagli altri. Di questi gruppi ne esistono, poi, tre tipologie, che prendono il nome di 'Telemetria 1', 'Telemetria 2' e 'Telemetria 3' le quali contengono dati notevolmente correlati. Dato che la comunicazione con la GS avviene trasmettendo dei *frame* di lunghezza fissa, per evitare frammentazioni di dati, nasce l'esigenza che ogni tipologia di dato venga trasmessa in un unico frame.

5.1.1 Il Pacchetto di Telemetria 1

La prima tipologia di informazioni è formata da una acquisizione del magnetometro, dal dato PVT e dagli *HouseKeeping*–scienza. Questo perché, come visto nel capitolo 2.5.2, i dati del magnetometro, per avere interesse scientifico, devono essere corredati dai dati di posizione, velocità, tempo e dalle informazioni sull'orientamento del satellite. Le informazioni che quindi compongono l'insieme di dati di Telemetria 1 sono:

- misure del magnetometro;
- pacchetto PVT;
- *HouseKeeping*–scienza.

I dati di Telemetria 1, per essere trasferiti alla GS, devono essere inseriti all'interno di un *frame* che ha un *payload* di 255 byte. Affinchè la GS sia in grado di riconoscere quali tipi di Telemetrie sono contenuti nel *payload* del *frame* ricevuto, si è reso necessario creare una mappatura che definisca la struttura dei byte di Telemetria 1. Tale mappatura prende il nome di 'Pacchetto di Telemetria 1' ed è così formato:

- 1 byte per l'intestazione, necessario per informare la GS si trova in presenza di un Pacchetto di Telemetria 1 (*Inizio Telemetria 1*);
- 49 byte per il pacchetto PVT del GPS;
- 2 byte per la misura dell'asse x del campo magnetico;
- 2 byte per la misura dell'asse y del campo magnetico;
- 2 byte per la misura dell'asse z del campo magnetico.
- 70 byte per la *HouseKeeping*–scienza.

Il Pacchetto di Telemetria 1 ha quindi la dimensione di 126byte. Per migliorare l'efficienza della trasmissione, è stato deciso di includere in un unico *frame* 2 Pacchetti di Telemetria 1. In questo modo all'interno di un *frame* vengono inviati 252 byte di informazione, riempiendo così il *payload* al 99% della sua capacità. Nel caso in cui non ci fossero in memoria 2 pacchetti di Telemetria 1 consecutivi, verrebbe inviato un *frame* con soltanto un Pacchetto di Telemetria 1.

5.1.2 I Pacchetti di Telemetria 2

La seconda tipologia prende il nome di Telemetria 2 ed è formata dai dati acquisiti dallo spettro–dosimetro e dagli *HouseKeeping*–sistema di tipo 1, i quali vengono campionati alla stessa frequenza ed in modo sincrono. Questi due tipi di dato vengono raggruppati, per semplicità implementativa, nella stessa tipologia. La Telemetria 2 è costuita dai seguenti dati:

- 516 byte che corrispondono ad un'acquisizione dello spettro–dosimetro;
- 26 byte per gli *HouseKeeping*–sistema di tipo 2.

Dato che la trasmissione alla GS di tali dati non può avvenire attraverso l'invio di un unico *frame*, la Telemetria 2 viene frammentata in 3 pacchetti denominati 'Pacchetto di Telemetria 2_1', 'Pacchetto di Telemetria 2_2', 'Pacchetto di Telemetria 2_3'. La struttura di tali Pacchetti è:

Pacchetto di Telemetria 2_1

- 1 byte per l'intestazione, necessario per informare la GS che si trova in presenza di un Pacchetto di Telemetria 2_1 (*Inizio Telemetria 2_1*);
- i primi 181 byte dell'acquisizione dello spettro–dosimetro.

Pacchetto di Telemetria 2_2

- 1 byte per l'intestazione, necessario per informare la GS che si trova in presenza di un Pacchetto di Telemetria 2_2 (*Inizio Telemetria 2_2*);
- i successivi 181 byte dell'acquisizione dello spettro–dosimetro.

Pacchetto di Telemetria 2_3

- 1 byte per l'intestazione, necessario per informare la GS che si trova in presenza di un Pacchetto di Telemetria 2_3 (*Inizio Telemetria 2_3*);
- gli ultimi 154 byte dell'acquisizione dello spettro–dosimetro;
- 26 byte per gli *HouseKeeping*–sistema di tipo 2.

La trasmissione della Telemetria 2 avviene quindi inviando 3 *frame* consecutivi che contengono ciascuno un Pacchetto di Telemetria 2. Tramite questo metodo di trasmissione, il *payload* viene riempito al 71% della sua capacità.

5.1.3 Il Pacchetto di Telemetria 3

Il terzo gruppo di informazioni, denominato Telemetria 3, è formato dal pacchetto 'Orbital Elements' del GPS e dagli *HouseKeeping*–sistema di tipo 2. Per gli stessi motivi che accomunano le informazioni di Telemetria 2, anche in questo caso i dati nella Telemetria 3 vengono posti nello stesso gruppo per semplicità implementativa. Il pacchetto di Telemetria 3 che verrà inserito nel *payload* del *frame* è così formato:

- 1 byte per l'intestazione, necessario per informare la GS che si trova in presenza di un Pacchetto di Telemetria 3 (*Inizio Telemetria 3*);
- 71 byte per il pacchetto GPS 'Orbital Elements';
- 90 byte per gli *HouseKeeping*–sistema di tipo 3.

La trasmissione di una Telemetria 3 avviene inviando un singolo *frame*, il cui *payload* viene riempito al 63% dal Pacchetto di Telemetria 3.

5.2 La trasmissione *stop and wait*

Nella fase di trasmissione delle Telemetrie Atmocube deve assicurarsi che ogni pacchetto trasmesso alla GS venga ricevuto correttamente. Dato che il canale di comunicazione è bidirezionale ed il tempo a disposizione lo permette, è stato deciso di adottare per AtmoCube un metodo di trasmissione *Automatic Repeat reQuest* (ARQ) di tipo *stop and wait*; ovverosia, ogni volta che AtmoCube trasmette un pacchetto di telemetria, aspetta, per un certo intervallo di tempo (*timeout*) che arrivi un messaggio che confermi la corretta ricezione (*acknowledgement*) da parte della GS. Dopo aver inviato un pacchetto, i casi possibili sono tre:

- il pacchetto di telemetria inviato da AtmoCube arriva correttamente a destinazione e quindi la GS invia il messaggio di conferma. AtmoCube riceve il messaggio di conferma prima dello scadere del tempo massimo ed invia un nuovo pacchetto;
- il pacchetto di telemetria inviato da AtmoCube non arriva correttamente a destinazione e quindi la GS non invia il messaggio di conferma. AtmoCube aspetta per tutto il tempo di *timeout* e poi ritrasmette lo stesso pacchetto;
- il pacchetto di telemetria inviato da AtmoCube arriva correttamente a destinazione e la GS invia il messaggio di conferma che però non viene ricevuto da AtmoCube. Quest'ultimo aspetta fino allo scadere del tempo di *timeout* e poi ritrasmette lo stesso pacchetto.

Il tempo necessario per trasmettere un pacchetto di telemetria comprende:

- il tempo per la trasmissione del *frame* di *down–link* (T_{F_d});
- il tempo necessario alla propagazione del segnale radio (T_p);
- il tempo per la trasmissione del *frame* di *up–link* (T_{F_u}).

Per i calcoli dei tempi di trasmissione dei pacchetti, si assume che:

- i tempi di elaborazione delle informazioni siano trascurabili;
- il tempo di propagazione rimanga fissato al valore che assume quando AtmoCube e la GS sono alla distanza massima all'interno della finestra trasmissiva: $T_p = D_{Max}/c = 4000/300000 = 13\ ms$;
- il *Round Trip Time* (RTT) venga fissato a 2 volte il tempo di propagazione ($RTT = 2 * T_p = 26\ ms$);
- il tempo di *timeout* sia fissato a 4 volte RTT ($T_{timeout} = 4 * RTT = 104\ ms$).

Come descritto dal capitolo 5.1, l'invio dei pacchetti avviene tramite la trasmissione dei *frame*. Il metodo di trasmissione *stop and wait* implementato per AtmoCube è stato progettato per confermare la corretta ricezione dei pacchetti di telemetria trasmessi e non dei singoli *frame*. Il messaggio di *ackowledgement*, quindi, conferma l'avvenuta ricezione dell'ultimo pacchetto di telemetria trasmesso. Si rende necessario specificare il metodo *stop and wait* implementato per ogni pacchetto di telemetria (vedi Figura 5.1) poiché la modalità di trasmissione dei *frame* è caratteristica per ogni tipo di telemetria.

5.2.1 Lo *stop and wait* per la Telemetria 1

I Pacchetti di Telemetria 1, come detto nel capitolo 5.1.1, vengono inviati a coppie in un singolo *frame*. Quando AtmoCube invia un *frame* contenente due Pacchetti di Telemetria 1, aspetta un solo messaggio dalla GS per la conferma di entrambi. Il tempo totale necessario per la trasmissione è quindi:

$$T_{T1} = T_{F_d} + 2 * \mathrm{RTT} + T_{F_u} = 212 + 26 + 59 = 297\ ms \tag{5.1}$$

5.2.2 Lo *stop and wait* per la Telemetria 2

La Telemetria 2, come detto nel capitolo 5.1.2, per essere inviata, viene deframmentata in tre pacchetti ed ognuno di questi viene inviato attraverso un *frame* diverso. AtmoCube, per inviare tale Telemetria, trasmette consecutivamente i tre pacchetti e aspetta un solo messaggio di conferma. Il tempo totale necessario per la trasmissione è quindi:

$$T_{T2} = 3 * T_{F_d} + 2 * \mathrm{RTT} + T_{F_u} = 212 + 26 + 59 = 721\ ms \tag{5.2}$$

5.2.3 Lo *stop and wait* per la Telemetria 3

Il pacchetto di Telemetria 3, come detto nel capitolo 5.1.3, viene inviato attraverso la trasmissione di un *frame*. Quando AtmoCube invia un *frame* contenente il Pacchetto

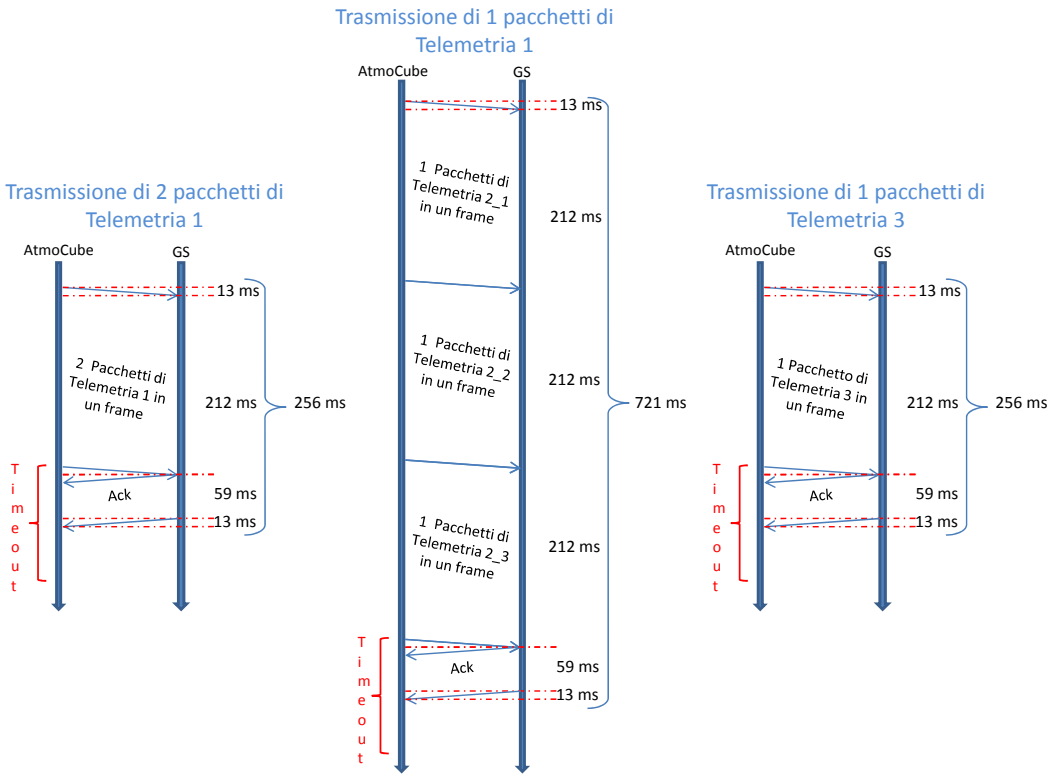


Figura 5.1: Tempistiche necessarie alle trasmissione dei Pacchetti di Telemetria

di Telemetria 3, aspetta il messaggio dalla GS per la conferma del pacchetto. Il tempo totale necessario per la trasmissione è quindi:

$$T_{T3} = T_{F_d} + 2 * \mathrm{RTT} + T_{F_u} = 212 + 26 + 59 = 297\ ms \tag{5.3}$$

5.3 Organizzazione della memoria SRAM

La memoria SRAM di 1920 KB presente sulla OBDH non deve solamente mantenere immagazzinate le Telemetrie, ma viene usata anche per altri scopi, come ad esempio per il salvataggio degli ultimi *HouseKeeping* e la memorizzazione temporanea dei dati ottenuti dallo spettro–dosimetro e dei *frame* da trasmettere. La memoria viene così divisa:

- 1548 byte per il salvataggio temporaneo dei dati, in triplice copia, provenienti dallo spettro–dosimetro in modo che siano controllati a maggioranza dal microcontrollore;
- 186 byte per il salvataggio dei dati degli *HouseKeeping* più recenti;
- 15 byte lasciati liberi per eventuali modifiche future;
- 1964330 byte per il salvataggio dei Pacchetti di Telemetria.

Nella fase di trasmissione dei *frame* vengono generati i successivi 2 *payload* da inviare, salvandoli temporaneamente nella SRAM. Per non allocare staticamente uno spazio di memoria solo per questo scopo, si utilizza lo spazio dedicato al salvataggio temporaneo delle acquisizioni dello spettro–dosimetro.

È possibile utilizzare lo stesso spazio di memoria perché:

- la trasmissione dei pacchetti e le acquisizioni dello spettro–dosimetro non vengono effettuate allo stesso istante;
- la porzione di memoria temporanea dei dati per lo spettro–dosimetro è abbastanza capiente per contenere 2 Pacchetti di Telemetria 2, corrispondenti a 6*255=1630 byte.

La figura 5.2 mostra come è stata suddivisa la memoria SRAM.

5.3.1 Gestione della memoria dedicata al salvataggio dei Pacchetti di Telemetria

Le Telemetrie vengono salvate direttamente sotto forma di Pacchetti di Telemetria. In questo modo è possibile, per il microcontrollore, riconoscere quali Telemetrie sono in memoria e, come detto nel capitolo 4.3.2, esse vengono salvate con una codifica estesa

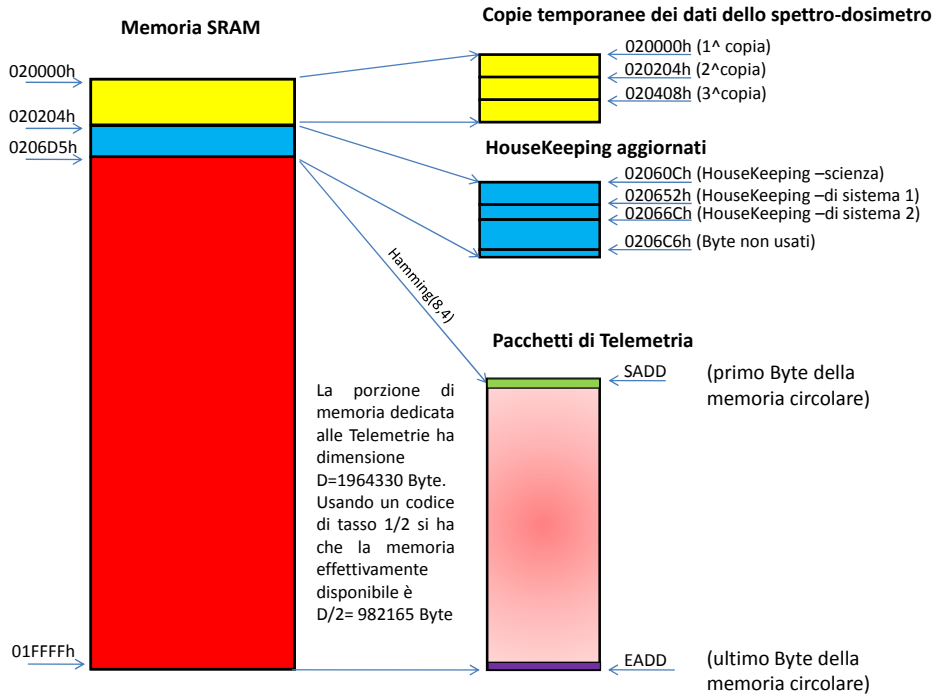


Figura 5.2: Organizzazione della memoria SRAM

di Hamming che ha un tasso RC=1/2. Questo significa che, per ogni byte di Telemetria, sono necessari 2 byte di memoria e quindi lo spazio di memoria effettivo per il salvataggio delle Telemetrie si dimezza, arrivando a 982165 byte. Per un uso completo ed efficiente della memoria di Telemetria è stata realizzata una memoria di tipo circolare che, una volta riempita, permette di salvare i nuovi dati, sovrascrivendo quelli meno recenti. Per creare la memoria circolare, l'indirizzo del primo byte della memoria di Telemetria viene denominato '*Start Address* (SADD)' e quello dell'ultimo byte '*End Address* (EADD)'. Vengono definiti due indirizzi di memoria che prendono il nome di '*First To Write* (FTW)' e '*First To Read* (FTR)' di cui il primo di questi rappresenta l'indirizzo del primo byte da riscrivere per salvare un nuovo dato ed il secondo rappresenta l'indirizzo di inizio del prossimo Pacchetto di Telemetria da trasmettere. Vi è poi la necessità di utilizzare altri due indirizzi: '*Next To Write* (NTW)' per indicare il prossimo byte da riscrivere e '*Next To Read* (NTR)' per indicare il prossimo byte da trasmettere. Il ruolo dello FTR consiste nell'indicare il primo byte del Pacchetto di Telemetria e quindi deve indicare solo byte di tipo '*Inizio Telemetria 1*', '*Inizio Telemetria 2_1*', '*Inizio Telemetria 2_2*', '*Inizio Telemetria 2_3*', '*Inizio Telemetria 3*'.

Per comprendere la relazione tra FTR e NTR, si fa ora l'esempio di una trasmissione di un Pacchetto di Telemetria. All'inizio della trasmissione dati, NTR ha lo stesso valore di FTR; ad ogni byte trasmesso NTR viene incrementato fino ad arrivare all'ultimo by-

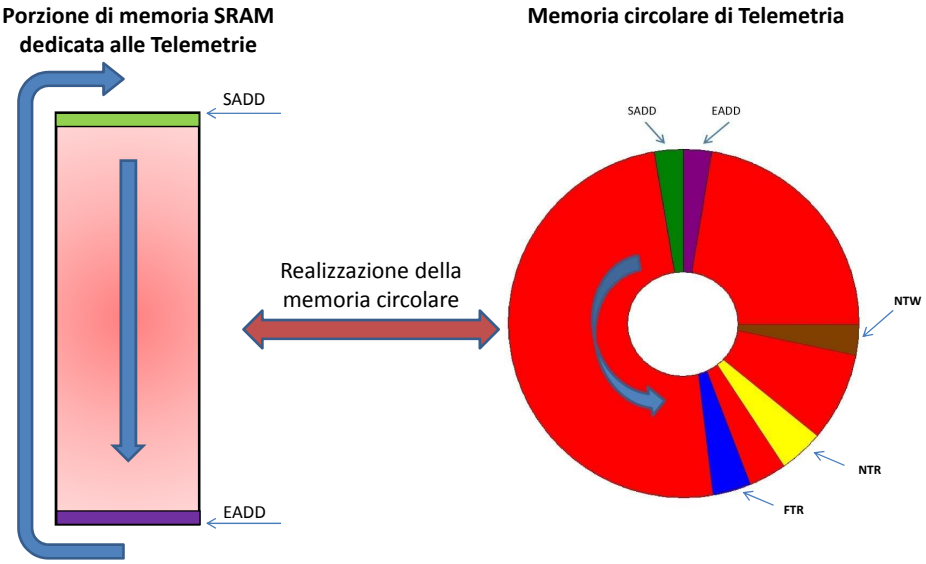


Figura 5.3: Realizzazione della memoria circolare di Telemetria

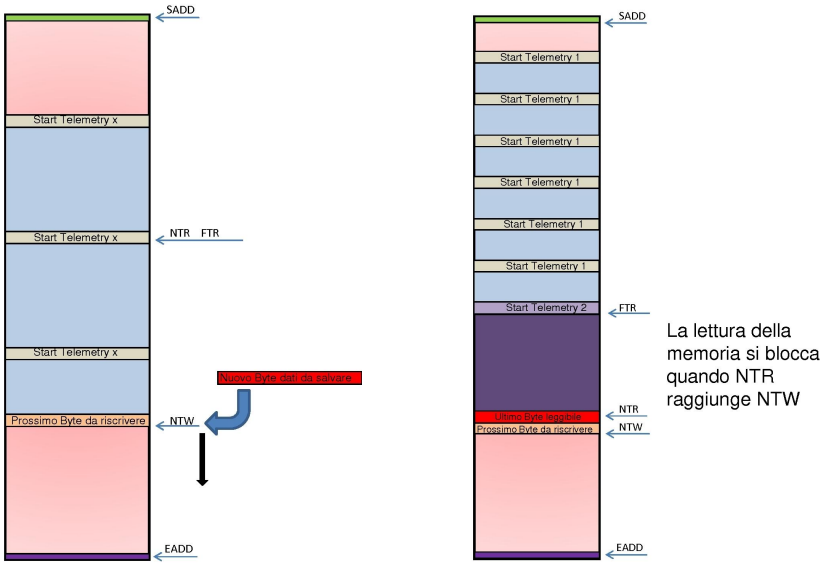
te del Pacchetto trasmesso. Se AtmoCube riceve la conferma di avvenuta ricezione da parte della GS, allora NTR e FTR assumono il valore dell'indirizzo del byte di inizio del Pacchetto di Telemetria successivo.

Per realizzare la memoria circolare, si implementa un algoritmo per FTW, NTW, FTR e NTR che segue la seguente regola: *se uno tra FTW, NTW, FTR o NTR è uguale a EADD, allora il suo prossimo valore sarà uguale a EADD*. La figura 5.3 mostra una rappresentazione della memoria circolare realizzata. Ci sono due casi possibili:

1. NTR$\leq$NTW;
2. NTR>NTW.

Nel primo caso, come raffigurato nella Figura 5.4a, se AtmoCube sta salvando dei dati, NTW può essere incrementato fino al valore massimo di EADD. Se, invece, si stanno trasmettendo dati (vedi Figura 5.4b), allora NTR può essere incrementato fino a che NTR=NTW poiché, in questo caso, i byte successivi a quelli indicati da NTW sono già stati trasmessi oppure possono essere riscritti.

Il secondo caso si verifica quando NTW supera EADD ed è quindi posto nuovamente uguale a SADD per realizzare la memoria circolare. La fase di lettura dei dati non presenta sostanziali problemi, in quanto NTR può essere incrementato senza tener conto di dove si trovi NTW. Nella fase di scrittura, invece, può capitare che ad un certo punto



(a) Lettura dell'ultimo byte in memoria, nel caso NTR<NTW.

(b) Scrittura di un byte nel caso NTR<NTW.

Figura 5.4: Casi particolari che possono verificarsi quando NTR<NTW

NTW oltrepassi FTR. In questo caso (Figura 5.5) si verifica la riscrittura della memoria e sia FTR, che NTR vengono cambiati ed assumono l'indirizzo del prossimo byte di inizio di Telemetria.

5.3.2 Salvataggio dei Pacchetti di Telemetria

Come detto nei capitoli 2.5.2 e 2.5.3, le Telemetrie vengono acquisite ciclicamente dopo un certo numero di impulsi PPS. Per sapere ogni quanti secondi acquisire una nuova Telemetria, vengono definiti quattro parametri:

- **PPS_TEL1**, che indica ogni quanti secondi salvare un nuovo Pacchetto di Telemetria 1;
- **PPS_TEL2**, che indica ogni quanti secondi salvare un nuovo Pacchetto di Telemetria 2;
- **PPS_SDD_INT**, che indica la durata di un'integrazione dello spettro–dosimetro;
- **PPS_TEL3**, che indica ogni quanti secondi salvare un nuovo Pacchetto di Telemetria 3.

I quattro parametri sopra elencati devono poter essere aggiornati dalla GS e nella Tabella 5.1 sono riportati i valori predefiniti.

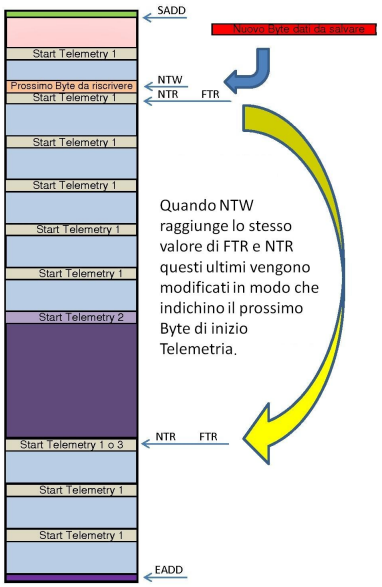


Figura 5.5: Caso particolare di riscrittura della memoria quando NTW<NTR .

Tabella 5.1: Valori predefiniti dei parametri di Telemetria

Parametro	Valore predefinito [s]
PPS_TEL1	10
PPS_TEL2	60
PPS_SDD_INT	60
PPS_TEL3	600

le Figure 5.6, 5.7, 5.8 e 5.9 mostrano un esempio dei pacchetti acquisiti in 10 minuti se vengono utilizzati i valori predefiniti dei parametri di acquisizione delle telemetrie.

Time (s)	Offset	Byte	Tipo	Byte (block)
1	0	1	Start Telemetry 1	126
	1	49	GPS 0x10 PVT	
	50	6	Magnetometro	
	56	70	HK - Scienza	
10	126	1	Start Telemetry 1	126
	127	49	GPS 0x10 PVT	
	176	6	Magnetometro	
	182	70	HK - Scienza	
20	252	1	Start Telemetry 1	126
	253	49	GPS 0x10 PVT	
	302	6	Magnetometro	
	308	70	HK - Scienza	
30	378	1	Start Telemetry 1	126
	379	49	GPS 0x10 PVT	
	428	6	Magnetometro	
	434	70	HK - Scienza	
40	504	1	Start Telemetry 1	126
	505	49	GPS 0x10 PVT	
	554	6	Magnetometro	
	560	70	HK - Scienza	
50	630	1	Start Telemetry 1	126
	631	49	GPS 0x10 PVT	
	680	6	Magnetometro	
	686	70	HK - Scienza	
	756	1	Start Telemetry2_1	182
	757	181	SDD_1/3	
	938	1	Start Telemetry2_2	182
	939	181	SDD_2/3	
	1120	1	Start Telemetry2_3	181
	1121	154	SDD_3/3	
	1275	26	HK – system 1	

Time (s)	Offset	Byte	Tipo	Byte (block)
60	1301	1	Start Telemetry 1	126
	1302	49	GPS 0x10 PVT	
	1351	6	Magnetometro	
	1357	70	HK - Scienza	
70	1427	1	Start Telemetry 1	126
	1428	49	GPS 0x10 PVT	
	1477	6	Magnetometro	
	1483	70	HK - Scienza	
80	1553	1	Start Telemetry 1	126
	1554	49	GPS 0x10 PVT	
	1603	6	Magnetometro	
	1609	70	HK - Scienza	
90	1679	1	Start Telemetry 1	126
	1680	49	GPS 0x10 PVT	
	1729	6	Magnetometro	
	1735	70	HK - Scienza	
100	1805	1	Start Telemetry 1	126
	1806	49	GPS 0x10 PVT	
	1855	6	Magnetometro	
	1861	70	HK - Scienza	
110	1931	1	Start Telemetry 1	126
	1932	49	GPS 0x10 PVT	
	1981	6	Magnetometro	
	1987	70	HK - Scienza	
	2057	1	Start Telemetry2_1	182
	2058	181	SDD_1/3	
	2239	1	Start Telemetry2_2	182
	2240	181	SDD_2/3	
	2421	1	Start Telemetry2_3	181
	2422	154	SDD_3/3	
	2576	26	HK – system 1	

Time (s)	Offset	Byte	Tipo	Byte (block)
120	2602	1	Start Telemetry 1	126
	2603	49	GPS 0x10 PVT	
	2652	6	Magnetometro	
	2658	70	HK - Scienza	
130	2728	1	Start Telemetry 1	126
	2729	49	GPS 0x10 PVT	
	2778	6	Magnetometro	
	2784	70	HK - Scienza	
140	2854	1	Start Telemetry 1	126
	2855	49	GPS 0x10 PVT	
	2904	6	Magnetometro	
	2910	70	HK - Scienza	
150	2980	1	Start Telemetry 1	126
	2981	49	GPS 0x10 PVT	
	3030	6	Magnetometro	
	3036	70	HK - Scienza	
160	3106	1	Start Telemetry 1	126
	3107	49	GPS 0x10 PVT	
	3156	6	Magnetometro	
	3162	70	HK - Scienza	
170	3232	1	Start Telemetry 1	126
	3233	49	GPS 0x10 PVT	
	3282	6	Magnetometro	
	3288	70	HK - Scienza	
	3358	1	Start Telemetry2_1	182
	3359	181	SDD_1/3	
	3540	1	Start Telemetry2_2	182
	3541	181	SDD_2/3	
	3722	1	Start Telemetry2_3	181
	3723	154	SDD_3/3	
	3877	26	HK – system 1	

Figura 5.6: Schema raffigurante le Telemetrie salvate in 10 minuti (figura 1 di 4)

Time (s)	Offset	Byte	Tipo	Byte (block)
180	3903	1	Start Telemetry 1	126
	3904	49	GPS 0x10 PVT	
	3953	6	Magnetometro	
	3959	70	HK - Scienza	
190	4029	1	Start Telemetry 1	126
	4030	49	GPS 0x10 PVT	
	4079	6	Magnetometro	
	4085	70	HK - Scienza	
200	4155	1	Start Telemetry 1	126
	4156	49	GPS 0x10 PVT	
	4205	6	Magnetometro	
	4211	70	HK - Scienza	
210	4281	1	Start Telemetry 1	126
	4282	49	GPS 0x10 PVT	
	4331	6	Magnetometro	
	4337	70	HK - Scienza	
220	4407	1	Start Telemetry 1	126
	4408	49	GPS 0x10 PVT	
	4457	6	Magnetometro	
	4463	70	HK - Scienza	
230	4533	1	Start Telemetry 1	126
	4534	49	GPS 0x10 PVT	
	4583	6	Magnetometro	
	4589	70	HK - Scienza	
	4659	1	Start Telemetry2_1	182
	4660	181	SDD_1/3	
	4841	1	Start Telemetry2_2	182
	4842	181	SDD_2/3	
	5023	1	Start Telemetry2_3	181
	5024	154	SDD_3/3	
	5178	26	HK – system 1	

Time (s)	Offset	Byte	Tipo	Byte (block)
240	5204	1	Start Telemetry 1	126
	5205	49	GPS 0x10 PVT	
	5254	6	Magnetometro	
	5260	70	HK - Scienza	
250	5330	1	Start Telemetry 1	126
	5331	49	GPS 0x10 PVT	
	5380	6	Magnetometro	
	5386	70	HK - Scienza	
260	5456	1	Start Telemetry 1	126
	5457	49	GPS 0x10 PVT	
	5506	6	Magnetometro	
	5512	70	HK - Scienza	
270	5582	1	Start Telemetry 1	126
	5583	49	GPS 0x10 PVT	
	5632	6	Magnetometro	
	5638	70	HK - Scienza	
280	5708	1	Start Telemetry 1	126
	5709	49	GPS 0x10 PVT	
	5758	6	Magnetometro	
	5764	70	HK - Scienza	
290	5834	1	Start Telemetry 1	126
	5835	49	GPS 0x10 PVT	
	5884	6	Magnetometro	
	5890	70	HK - Scienza	
	5960	1	Start Telemetry2_1	182
	5961	181	SDD_1/3	
	6142	1	Start Telemetry2_2	182
	6143	181	SDD_2/3	
	6324	1	Start Telemetry2_3	181
	6325	154	SDD_3/3	
	6479	26	HK – system 1	

Time (s)	Offset	Byte	Tipo	Byte (block)
300	6505	1	Start Telemetry 1	126
	6506	49	GPS 0x10 PVT	
	6555	6	Magnetometro	
	6561	70	HK - Scienza	
310	6631	1	Start Telemetry 1	126
	6632	49	GPS 0x10 PVT	
	6681	6	Magnetometro	
	6687	70	HK - Scienza	
320	6757	1	Start Telemetry 1	126
	6758	49	GPS 0x10 PVT	
	6807	6	Magnetometro	
	6813	70	HK - Scienza	
330	6883	1	Start Telemetry 1	126
	6884	49	GPS 0x10 PVT	
	6933	6	Magnetometro	
	6939	70	HK - Scienza	
340	7009	1	Start Telemetry 1	126
	7010	49	GPS 0x10 PVT	
	7059	6	Magnetometro	
	7065	70	HK - Scienza	
350	7135	1	Start Telemetry 1	126
	7136	49	GPS 0x10 PVT	
	7185	6	Magnetometro	
	7191	70	HK - Scienza	
	7261	1	Start Telemetry2_1	182
	7262	181	SDD_1/3	
	7443	1	Start Telemetry2_2	182
	7444	181	SDD_2/3	
	7625	1	Start Telemetry2_3	181
	7626	154	SDD_3/3	
	7780	26	HK – system 1	

Figura 5.7: Schema raffigurante le Telemetrie salvate in 10 minuti (figura 2 di 4)

Time (s)	Offset	Byte	Tipo	Byte (block)
360	7806	1	Start Telemetry 1	126
	7807	49	GPS 0x10 PVT	
	7856	6	Magnetometro	
	7862	70	HK - Scienza	
370	7932	1	Start Telemetry 1	126
	7933	49	GPS 0x10 PVT	
	7982	6	Magnetometro	
	7988	70	HK - Scienza	
380	8058	1	Start Telemetry 1	126
	8059	49	GPS 0x10 PVT	
	8108	6	Magnetometro	
	8114	70	HK - Scienza	
390	8184	1	Start Telemetry 1	126
	8185	49	GPS 0x10 PVT	
	8234	6	Magnetometro	
	8240	70	HK - Scienza	
400	8310	1	Start Telemetry 1	126
	8311	49	GPS 0x10 PVT	
	8360	6	Magnetometro	
	8366	70	HK - Scienza	
410	8436	1	Start Telemetry 1	126
	8437	49	GPS 0x10 PVT	
	8486	6	Magnetometro	
	8492	70	HK - Scienza	
	8562	1	Start Telemetry2_1	182
	8563	181	SDD_1/3	
	8744	1	Start Telemetry2_2	182
	8745	181	SDD_2/3	
	8926	1	Start Telemetry2_3	181
	8927	154	SDD_3/3	
	9081	26	HK – system 1	

Time (s)	Offset	Byte	Tipo	Byte (block)
420	9107	1	Start Telemetry 1	126
	9108	49	GPS 0x10 PVT	
	9157	6	Magnetometro	
	9163	70	HK - Scienza	
430	9233	1	Start Telemetry 1	126
	9234	49	GPS 0x10 PVT	
	9283	6	Magnetometro	
	9289	70	HK - Scienza	
440	9359	1	Start Telemetry 1	126
	9360	49	GPS 0x10 PVT	
	9409	6	Magnetometro	
	9415	70	HK - Scienza	
450	9485	1	Start Telemetry 1	126
	9486	49	GPS 0x10 PVT	
	9535	6	Magnetometro	
	9541	70	HK - Scienza	
460	9611	1	Start Telemetry 1	126
	9612	49	GPS 0x10 PVT	
	9661	6	Magnetometro	
	9667	70	HK - Scienza	
470	9737	1	Start Telemetry 1	126
	9738	49	GPS 0x10 PVT	
	9787	6	Magnetometro	
	9793	70	HK - Scienza	
	9863	1	Start Telemetry2_1	182
	9864	181	SDD_1/3	
	10045	1	Start Telemetry2_2	182
	10046	181	SDD_2/3	
	10227	1	Start Telemetry2_3	181
	10228	154	SDD_3/3	
	10382	26	HK – system 1	

Figura 5.8: Schema raffigurante le Telemetrie salvate in 10 minuti (figura 3 di 4)

5.4 Le fasi del Sistema Operativo di AtmoCube

La missione di AtmoCube, come detto nel capitolo 2.7, può essere suddivisa in varie fasi operative, ognuna di queste caratterizzata da diverse operazioni e funzionalità che AtmoCube deve eseguire. Il Sistema Operativo di AtmoCube è strutturato in stati che rispecchiano le fasi operative del satellite che sono cinque:

- *Start Phase* (SP);
- *Earlier Operating Phase* (EOP);
- *Standard Operating Phase* (SOP);
- *Unidirectional Operating Phase* (UOP);
- *Safe Mode Phase* (SMP).

Nella Figura 5.10 sono raffigurati gli stati del Sistema Operativo di AtmoCube.

5.4.1 La *Start Phase*

Il Sistema Operativo di AtmoCube può trovarsi in questa fase quando:

- avviene l'espulsione dal P–POD;

Time (s)	Offset	Byte	Tipo	Byte (block)
480	10408	1	Start Telemetry 1	126
	10409	49	GPS 0x10 PVT	
	10458	6	Magnetometro	
	10464	70	HK - Scienza	
490	10534	1	Start Telemetry 1	126
	10535	49	GPS 0x10 PVT	
	10584	6	Magnetometro	
	10590	70	HK - Scienza	
500	10660	1	Start Telemetry 1	126
	10661	49	GPS 0x10 PVT	
	10710	6	Magnetometro	
	10716	70	HK - Scienza	
510	10786	1	Start Telemetry 1	126
	10787	49	GPS 0x10 PVT	
	10836	6	Magnetometro	
	10842	70	HK - Scienza	
520	10912	1	Start Telemetry 1	126
	10913	49	GPS 0x10 PVT	
	10962	6	Magnetometro	
	10968	70	HK - Scienza	
530	11038	1	Start Telemetry 1	126
	11039	49	GPS 0x10 PVT	
	11088	6	Magnetometro	
	11094	70	HK - Scienza	
	11164	1	Start Telemetry2_1	182
	11165	181	SDD_1/3	
	11346	1	Start Telemetry2_2	182
	11347	181	SDD_2/3	
	11528	1	Start Telemetry2_3	181
	11529	154	SDD_3/3	
	11683	26	HK – system 1	

Time (s)	Offset	Byte	Tipo	Byte (block)
540	11709	1	Start Telemetry 1	126
	11710	49	GPS 0x10 PVT	
	11759	6	Magnetometro	
	11765	70	HK - Scienza	
550	11835	1	Start Telemetry 1	126
	11836	49	GPS 0x10 PVT	
	11885	6	Magnetometro	
	11891	70	HK - Scienza	
560	11961	1	Start Telemetry 1	126
	11962	49	GPS 0x10 PVT	
	12011	6	Magnetometro	
	12017	70	HK - Scienza	
570	12087	1	Start Telemetry 1	126
	12088	49	GPS 0x10 PVT	
	12137	6	Magnetometro	
	12143	70	HK - Scienza	
580	12213	1	Start Telemetry 1	126
	12214	49	GPS 0x10 PVT	
	12263	6	Magnetometro	
	12269	70	HK - Scienza	
590	12339	1	Start Telemetry 1	126
	12340	49	GPS 0x10 PVT	
	12389	6	Magnetometro	
	12395	70	HK - Scienza	
	12465	1	Start Telemetry2_1	182
	12466	181	SDD_1/3	
	12647	1	Start Telemetry2_2	182
	12648	181	SDD_2/3	
	12829	1	Start Telemetry2_3	181
	12830	154	SDD_3/3	
	12984	26	HK – system 1	
	13010	1	Start Telemetry 3	162
	13011	71	GPS 0x70 Orbit	
	13082	90	HK – system 2	

Figura 5.9: Schema raffigurante le Telemetrie salvate in 10 minuti (figura 4 di 4)

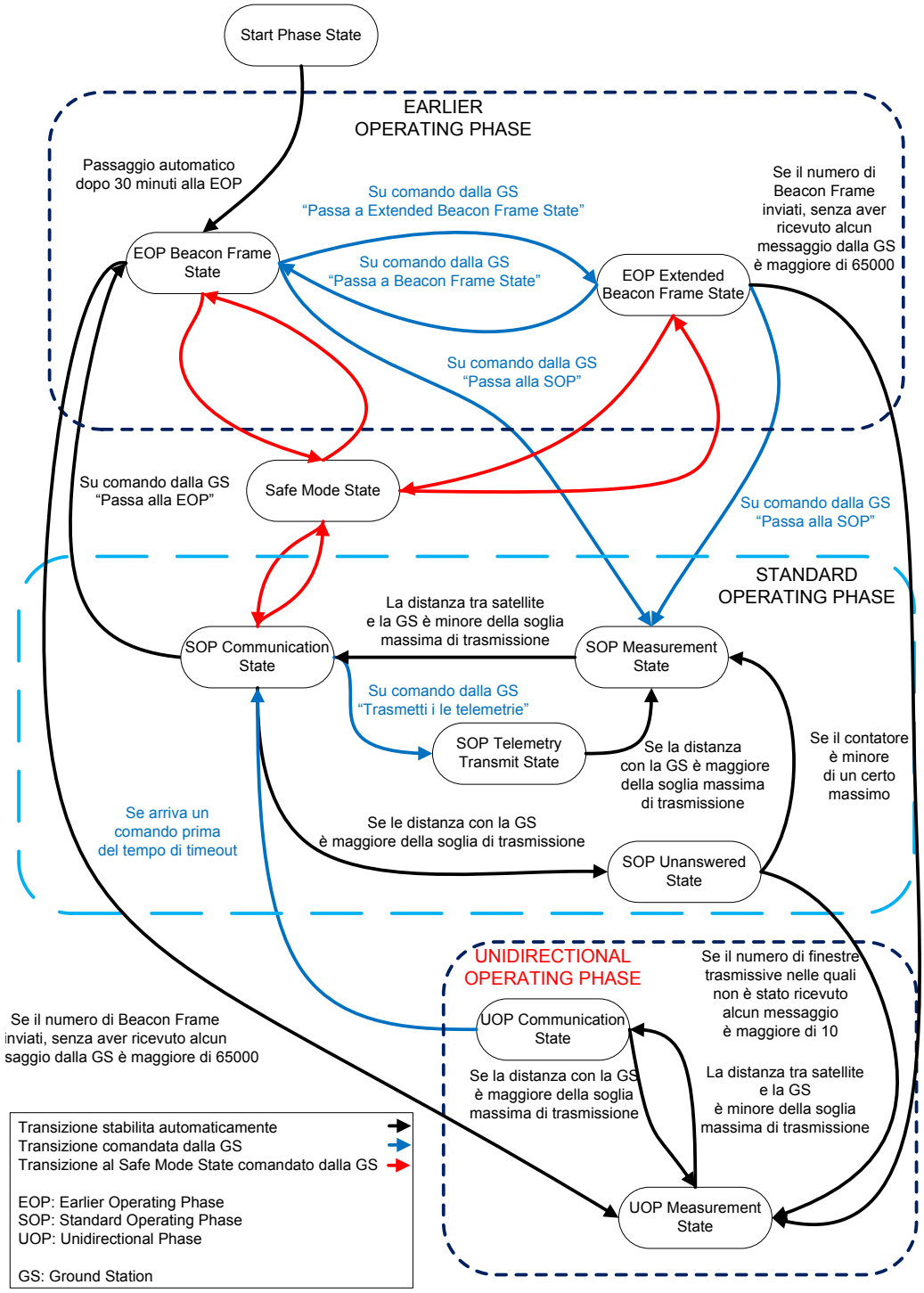


Figura 5.10: Stati del Sistema Operativo di AtmoCube

- si verifica un reset *software*;
- l'alimentazione al microcontrollore viene ristabilita a seguito di una completa scarica della batteria.

Le operazioni che vengono eseguite in questa fase sono necessarie per la configurazione delle impostazioni prestabilite di sistema (abilitazione degli *interrupt*, configurazione delle porte del microcontrollore, abilitazione delle linee di alimentazione, etc.). Inoltre viene impostato un timer per 30 minuti e, al suo scadere, viene dato il comando di espulsione dell'antenna e di abilitazione della linea di alimentazione destinata al modulo GPS. Terminate tali operazioni, il Sistema Operativo passa automaticamente alla EOP nello stato operativo 'EOP Beacon Frame State'.

5.4.2 La Earlier Operation Phase

Questa fase operativa è la fase di diagnostica del satellite descritta nel capitolo 2.7. Nella EOP esistono due stati operativi:

- EOP *Beacon Frame State* che corrisponde alla modalità diagnostica di base, nella quale, ogni minuto, vengono inviati i *Beacon Frame*;
- EOP *Extended Beacon Frame State* che corrisponde alla modalità diagnostica estesa, nella quale, ogni minuto, vengono inviati i *Beacon Frame* estesi.

Lo stato EOP *Beacon Frame State*

In questo stato, ogni minuto, il satellite acquisisce nuovi *HouseKeeping* salvandoli nella SRAM sempre nella stessa locazione di memoria (vedi capitolo 5.3); in seguito li trasferisce all'interno di un *payload* ed invia il relativo *frame*. Ad ogni invio, il Sistema Operativo incrementa una variable denominata BEACON_COUNTER che conta il numero di *Beacon Frame* inviati. Quando BEACON_COUNTER raggiunge il numero massimo di MAX_BEACON_COUNTER=65000, significa che AtmoCube deve passare alla modalità di trasmissione unidirezionale che corrisponde al passaggio del Sistema Operativo alla fase UOP. Ogni volta che AtmoCube invia un *Beacon Frame*, il Sistema Operativo aspetta per un tempo pari a $2 * T_{timeout}$ (vedi capitolo 5.2) che arrivi un messaggio dalla GS; quando questo avviene, la variabile BEACON_COUNTER viene azzerata. I messaggi che possono essere inviati in questo stato sono indicati nella Tabella 5.12. Il *payload* del *frame* che contiene un *Beacon Frame* è così formato:

- 70 byte per gli *HouseKeeping*–scienza;
- 26 byte per gli *HouseKeeping*–sistema di tipo 1;
- 90 byte per gli *HouseKeeping*–sistema di tipo 2;

- 69 byte posti a '0'.

Lo stato EOP *Extended Beacon Frame State*

Nello stato EOP *Extended Beacon Frame*, il satellite effettua, una volta ogni ora, delle misurazioni scientifiche della durata di un minuto e salva i dati ottenuti nella memoria di Telemetria. In base ai parametri prestabiliti di acquisizione delle Telemetrie, in un minuto vengono salvati in memoria 6 Pacchetti di Telemetria 1 ed un Pacchetto di Telemetria 2. Ogni minuto, AtmoCube invia i *Beacon Frame* estesi ed aspetta per un tempo pari a 2 volte il tempo $T_{timeout}$ (vedi capitolo 5.2) che arrivi un messaggio dalla GS. Quando il Sistema Operativo entra in questo stato, la variabile BEACON_COUNTER viene azzerata e, ad ogni *Beacon Frame* esteso inviato, viene incrementata. Quando BEACON_COUNTER raggiunge il numero massimo di MAX_BEACON_COUNTER=65000, significa che AtmoCube deve passare alla modalità di trasmissione unidirezionale che corrisponde al passaggio del Sistema Operativo alla fase UOP. I messaggi che AtmoCube può ricevere in questo stato sono indicati nella Tabella 5.12 e, se AtmoCube riceve un messaggio dalla GS, la variabile BEACON_COUNTER viene azzerata.

Per l'invio di un *Beacon Frame* Esteso è necessario trasmettere 6 *frame*. In base ai parametri prestabiliti di acquisizione delle Telemetrie, il *Beacon Frame* Esteso è così formato:

- 70 byte per gli *HouseKeeping*–scienza;
- 26 byte per gli *HouseKeeping*–sistema di tipo 1;
- 90 byte per gli *HouseKeeping*–sistema di tipo 2;
- 756 byte per 6 Pacchetti di Telemetria 1;
- 545 byte per 1 Pacchetto di Telemetria 2;
- 43 byte posti a '0'.

5.4.3 La *Standard Operating Phase*

Le operazioni che il satellite deve eseguire in questa fase sono:

- monitoraggio continuo della sua posizione, ovvero ogni volta che il GPS invia il comando PPS (ogni secondo) il microcontrollore richiede il dato PVT, dal quale ricava la posizione e ne calcola la distanza rispetto la GS. Se il satellite si trova ad una distanza minore di 4000 Km dalla GS lo stato del Sistema Operativo passa allo stato SOP *Communication State*;
- acquisizione e salvataggio dei dati sperimentali;

- operare secondo i comandi inviati dalla GS;
- ricezione e trasmissione di dati da e verso la GS;

Per realizzare le operazioni sopra elencate si è deciso di suddividere questa fase operativa nei seguenti stati:

- il SOP *Measure State*;
- il SOP *Communication State*;
- il SOP *Transmit Telemetry State*;
- il SOP *Unanswered State*.

Il SOP *Measure State*

In questo stato vengono effettuate diverse operazioni, le principali consistono nell'effettuare le misure sperimentali per poi salvare i dati per formare le varie Telemetrie.

Per scandire il tempo di acquisizione dei dati di telemetria, viene usata la linea PPS del modulo GPS che fornisce un impulso al secondo. Ad ogni attivazione del segnale PPS, il sistema incrementa una varabile denominata PPS_COUNTER che conta il numero di impulsi PPS. In base al valore che assume la variabile PPS_COUNTER il sistema decide quali Telemetrie salvare.

Le misure sperimentali contenute nella Telemetria 1 consistono nei campionamenti del magnetometro. Questi dati devono essere temporalmente correlati con l'ultima acquisizione PVT del GPS. Non appena avviene un impulso PPS, la variabile PPS_COUNTER viene incrementata e, se il suo nuovo valore ottenuto è multiplo di PPS_TEL1, vengono effettuati i campionamenti del campo magnetico e vengono aggiornati gli *Housekeeping* – scienza.

A questo punto, a partire dalla posizione di memoria indicata da FTW, viene salvato:

- il pacchetto PVT 0x10 del GPS;
- i 6 byte ottenuti dal campionamento del magnetometro;
- gli *Housekeeping* – scienza.

Infine, NTW e FTW vengono aumentati in modo da indicare la posizione del primo byte dopo l'ultimo byte della Telemetria 1 appena salvata.

Successivamente alle operazioni di salvataggio della Telemetria 1, il Sistema Operativo valuta il tempo passato dall'ultimo comando di 'Start' dato allo spettro–dosimetro. Per conoscere tale intervallo di tempo deve eseguire il calcolo:

$$PPS\_COUNTER \pmod{PPS\_TEL2}.$$

Se il risultato di tale operazione è uguale a PPS_SDD_INT viene dato il comando di 'Stop' allo spettro–dosimetro, che comporta l'arresto della misura.

Indipendentemente dal risultato del calcolo appena descritto, se PPS_COUNTER è multipla di PPS_TEL2 viene dato il comando di 'Read' allo spettro–dosimetro per leggere i dati dell'acquisizione e salvare un nuovo Pacchetto di Telemetria 2. Questi dati vengono salvati nella zona di memoria temporanea destinata ai dati ricevuti dallo spettro–dosimetro e, dopo averli ricevuti, vengono dati i comandi di 'Reset' e successivamente di 'Start' allo spettro–dosimetro, per cominciare una nuova misurazione. I byte ricevuti dallo spettrodosimetro si presentano in triplice copia ed è compito del sistema confrontare le tre copie di ogni byte per decidere a maggioranza quale salvare per correggere un eventuale errore dovuto al SEU. Il salvataggio avviene nella zona di memoria indicata da NTW e, dopo aver salvato i dati dello spettrodosimetro, viene salvata una nuova *Housekeeping*–sistema di tipo 1; infine FTW e NTW vengono modificati in modo che indichino il primo byte dopo l'ultimo Pacchetto di telemetria 2 salvato.

Il Pacchetto di Telemetria 3 viene salvato ogni volta che PPS_COUNTER è uguale a PPS_TEL3 e, quando questo avviene, viene richiesto il dato GPS '*Orbital Elements*'. Tale pacchetto viene salvato a partire dall'indirizzo di memoria indicato da NTW ed in seguito viene aggionata e salvata la *Housekeeping*–sistema di tipo 2. I valori di NTW e FTW vengono quindi modificati in modo che indichino il primo byte dopo l'ultimo appena salvato e PPS_COUNTER viene azzerata.

Quindi, riassumendo quanto appena detto, ogni secondo la variabile PPS_COUNTER viene incrementata e in base al nuovo valore ottenuto, vengono eseguite le seguenti azioni:

- se $\mathrm{PPS\_COUNTER} \pmod{\mathrm{PPS\_TEL1}} = 0$ viene salvato un nuovo Pacchetto di Telemetria 1;
- se $\mathrm{PPS\_COUNTER} \pmod{\mathrm{PPS\_TEL2}} = \mathrm{PPS\_SDD\_INT}$ viene dato il comando di 'stop' allo spettro–dosimetro;
- se $\mathrm{PPS\_COUNTER} \pmod{\mathrm{PPS\_TEL2}} = 0$ viene salvato un nuovo Pacchetto di Telemetria 2;
- se $\mathrm{PPS\_COUNTER} \pmod{\mathrm{PPS\_TEL3}} = 0$ viene salvato un nuovo Pacchetto di Telemetria 3.

Infine, per il monitoraggio continuo della posizione del satellite, ogni secondo viene letto il pacchetto PVT del GPS. Se la distanza tra AtmoCube e la GS è minore di 4000 Km, il Sistema Operativo passa automaticamente allo stato SOP *Communication State*.

Il SOP *Communication State*

Le operazioni svolte in questo stato sono necessarie per stabilire una comunicazione bidirezionale con la GS. Una volta che il sistema entra in questo stato, il microcontrollore esegue, ogni 5 secondi le seguenti operazioni:

- invia un *Beacon Frame*;
- aspetta un messaggio di risposta dalla GS per un tempo pari a 2 volte $T_{timeout}$;
- se riceve un messaggio dalla GS, esegue il comando contenuto nel pacchetto come descritto nel capitolo 5.4.4.

Inoltre, il Sistema Operativo calcola, ogni secondo, la distanza con la GS attraverso i dati forniti dal pacchetto PVT del GPS. Se tale distanza è maggiore di 4000 Km, si ha il passaggio di stato operativo al SOP *unanswered state*.

Il SOP *Telemetry Transmit State*

In questo stato avviene la trasmissione *stop and wait* dei Pacchetti di Telemetria come descritto nel capitolo 5.2 e viene aggiornato, ogni secondo, il valore della distanza tra AtmoCube e la GS. Ogni volta che viene inviato un pacchetto di Telemetria, il Sistema Operativo imposta un *timer* di durata $T_{timeout}$. Se non arriva il messaggio di 'Acknowledgement' in tale intervallo di tempo e se la distanza tra AtmoCube e la GS è minore di 4000 Km, il satellite ritrasmette l'ultimo pacchetto di Telemetria, altrimenti, se la distanza è maggiore di 4000 Km, lo stato passa a SOP *Measurement State*.

Il SOP *Unanswered State*

Questo stato operativo ha il compito di valutare se è il caso, per AtmoCube, di passare alla UOP o meno. Se nell'ultima finestra trasmissiva non è stato ricevuto alcun messaggio dalla GS, allora viene incrementata la variabile UNRESPONSE. Se, invece, nell'ultima finestra trasmissiva AtmoCube ha ricevuto un messaggio da Terra, la variabile UNRESPONSE viene azzerata. Infine, se la variabile UNRESPONSE è maggiore di MAX_UNRESPONSE, allora il sistema passa alla fase UOP nello stato UOP *Measurement State*, altrimenti si ha il passaggio allo stato SOP *Measurement State*.

5.4.4 La *Unidirectional Operating Phase*

Nel caso in cui non sia possibile una comunicazione bidirezionale tra AtmoCube e la GS, a causa di un'avaria del sistema di ricezione di bordo del satellite o di trasmissione della GS, questa fase permette il trasferimento delle telemetrie da AtmoCube alla GS. Gli stati operativi in questa fase sono due:

- *UOP Measurement State*;
- *UOP Communication State*.

L'*UOP Measurement State*

Le misure scientifiche ed il salvataggio dei Pacchetti di Telemetria in questo stato, avvengono nella stessa maniera in cui avvengono nello stato *SOP Measurement State*. Anche l'aggiornamento della distanza avviene nello stesso modo, ma, quando AtmoCube è a meno di 4000 Km dalla GS, lo stato passa a UOP *measurement* per la trasmissione unidirezionale dei Pacchetti di Telemetria.

L'*UOP Communication State*

Quando il satellite entra nella finestra trasmissiva, inizia la trasmissione non confermata dei pacchetti di telemetria. Ogni volta che viene trasmesso un pacchetto, il modulo radio di bordo si pone in stato di ricezione nell'attesa di un eventuale messaggio dalla GS. Se questo non arriva entro un intervallo di tempo pari a $2 * T_{timeout}$, si ha la trasmissione del prossimo pacchetto di Telemetria; se invece arriva un messaggio di qualsiasi tipo, si ha il passaggio alla fase SOP nello stato SOP *Communication State*. La trasmissione dei pacchetti di Telemetria avviene fino a che la distanza tra AtmoCube e la GS rimane minore di 4000Km; quando non lo è più, il sistema passa allo stato *UOP Measurement State*.

5.4.5 La *Safe Mode Phase*

In questa fase AtmoCube controlla, ogni secondo, la sua posizione e valuta la distanza con la GS: nel caso in cui essa sia minore di 4000 Km, pone la radio in stato di ricezione, in attesa di un messaggio proveniente dalla GS. Tutte le altre operazioni, che di solito vengono svolte dal satellite, vengono sospese e quindi non vengono né salvati né trasmessi pacchetti di Telemetria. Se, la radio è posta in ricezione ed il satellite riceve il comando di sblocco dalla *Safe Mode Phase*, il Sistema Operativo passa allo stato in cui si trovava prima di entrare nella *Safe Mode Phase*.

5.5 I messaggi della GS

La GS può inviare dei *frame* ad AtmoCube, all'interno dei quali sono presenti dei messaggi di controllo per il satellite. Come detto nel capitolo 2.2.2, i *frame* di *uplink* hanno un *payload* della lunghezza di 64 byte. I messaggi che possono essere inviati sono:

- 'Passa allo stato EOP *Beacon Frame State*';
- 'Passa allo stato EOP *Extended Beacon Frame State*';
- 'Programma il ricevitore GPS';
- 'Comando Dummy';
- 'Passa alla fase SMP';
- 'Passa allo stato SOP *Measurement State*';
- 'Passa allo stato SOP *Transmit Telemetry State*';
- 'Aggiorna i parametri di Telemetria';
- 'Acknowledgement Telemetria';
- 'Ritorna dalla SMP'.

Se, durante la fase UOP, arriva un qualsiasiasi messaggio, si ha il passaggio del Sistema Operativo alla fase SOP.

5.5.1 Il messaggio 'Passa allo stato EOP *Beacon Frame State*'

Il messaggio 'Passa allo stato EOP *Beacon Frame State*' provoca il passaggio del Sistema Operativo allo stato EOP *Beacon Frame State*. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- EOP *Extended Beacon Frame State*;
- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.2 descrive come è formato il messaggio.

Tabella 5.2: Struttura del messaggio 'Passa allo stato EOP *Beacon Frame State*'

Numero di byte	Valore	Descrizione
1	0b01010101	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

5.5.2 Il messaggio 'Passa allo stato EOP *Extended Beacon Frame State*'

Il messaggio 'Passa allo stato EOP *Extended Beacon Frame State*' provoca il passaggio del Sistema Operativo allo stato EOP *Extended Beacon Frame State*. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- EOP *Beacon Frame State*;
- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.3 descrive come è formato il messaggio.

Tabella 5.3: Struttura del messaggio 'Passa allo stato EOP *Extendd Beacon Frame State*'

Numero di byte	Valore	Descrizione
1	0b01010010	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

5.5.3 Il messaggio 'Programma il ricevitore GPS'

Il messaggio 'Programma il ricevitore GPS' fa in modo che il Sistema Operativo trasmetta allo SGR la sequenza di N (< 62) byte contenuta nel messaggio. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- EOP *Beacon Frame State*;
- EOP *Extended Beacon Frame State*;
- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.4 descrive come è formato il messaggio.

Tabella 5.4: Struttura del messaggio 'Programma il ricevitore GPS'

Numero di byte	Valore	Descrizione
1	0b01010010	byte identificativo del messaggio
1	N	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
N	Dati	byte da trasferire al modulo SRG
62-N	0b00000000	byte non utilizzati

5.5.4 Il messaggio 'Comando Dummy'

Il messaggio 'Comando Dummy' serve per sapere se il satellite è in grado di ricevere i comandi inviati dalla GS; al suo interno è presente un byte che deve essere ritrasmesso da AtmoCube all'interno di un *frame*. Se il *frame* ricevuto dalla GS contiene il byte inviato, significa che AtmoCube è in grado di comprendere i comandi inviatigli. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- EOP *Beacon Frame State*;
- EOP *Extended Beacon Frame State*;
- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.5 descrive come è formato il messaggio.

Tabella 5.5: Struttura del messaggio 'Comando Dummy'

Numero di byte	Valore	Descrizione
1	0b01010110	byte identificativo del messaggio
1	1	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
1	Dato	byte da ritrasmerire alla GS
61	0b00000000	byte non utilizzati

5.5.5 Il messaggio 'Passa alla fase SMP'

Il messaggio 'Passa alla fase SMP' provoca il passaggio del Sistema Operativo alla fase SMP. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- EOP *Beacon Frame State*;

- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.6 descrive come è formato il messaggio.

Tabella 5.6: Struttura del messaggio 'Passa alla fase SMP'

Numero di byte	Valore	Descrizione
1	0b01010100	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

5.5.6 Il messaggio 'Passa allo stato SOP *Measurement State*'

Il messaggio 'Passa allo stato SOP *Measurement State*' provoca il passaggio del Sistema Operativo allo stato SOP *Measurement State*. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- EOP *Beacon Frame State*;
- EOP *Extended Beacon Frame State*;
- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.7 descrive come è formato il messaggio.

Tabella 5.7: Struttura del messaggio 'Passa allo stato SOP *Measurement State*'

Numero di byte	Valore	Descrizione
1	0b01010011	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

5.5.7 Il messaggio 'Passa allo stato SOP *Transmit Telemetry State*'

Il messaggio 'Passa allo stato SOP *Transmit Telemetry State*' provoca il passaggio del Sistema Operativo allo stato SOP *Transmit Telemetry State*. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.8 descrive come è formato il messaggio.

Tabella 5.8: Struttura del messaggio 'Passa allo stato SOP *Transmit Telemetry State*'

Numero di byte	Valore	Descrizione
1	0b01011000	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

5.5.8 Il messaggio 'Aggiorna i parametri di Telemetria'

Il messaggio 'Aggiorna i parametri di Telemetria' serve per aggiornare i parametri di temporizzazione delle Telemetrie. Affinché il Sistema Operativo accetti i nuovi parametri ricevuti, essi devono rispettare alcune regole create per ottimizzare l'uso della memoria e dei *frame* trasmessi per inviare i Pacchetti di telemetria. Le regole sono:

- $\mathrm{PPS\_TEL1} \leq \mathrm{PPS\_TEL2} \leq \mathrm{PPS\_TEL3}$;
- $\mathrm{PPS\_SDD\_INT} \leq \mathrm{PPS\_TEL2}$;
- $\mathrm{PPS\_TEL3}/\mathrm{PPS\_TEL1} = 2$.

Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.9 descrive come è formato il messaggio.

Tabella 5.9: Struttura del messaggio 'Aggiorna i parametri di Telemetria'

Numero di byte	Valore	Descrizione
1	0b01010111	byte identificativo del messaggio
5	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
1	PPS_TEL1	nuovo valore per la temporizzazione della Telemetria 1
1	PPS_SDD_INT	nuovo valore per la temporizzazione del tempo di integrazione dello spettro–dosimetro
1	PPS_TEL2	nuovo valore per la temporizzazione della Telemetria 2
2	PPS_TEL3	nuovo valore per la temporizzazione della Telemetria 3
57	0b00000000	byte non utilizzati

5.5.9 Il messaggio 'Acknowledgement Telemetria'

Il messaggio 'Acknowledgement Telemetria' serve per comunicare ad AtmoCube che la GS ha ricevuto correttamente l'ultimo pacchetto di telemetria inviato. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- SOP *Communication State*;
- UOP *Communication State*.

La Tabella 5.10 descrive come è formato il messaggio.

Tabella 5.10: Struttura del messaggio 'Acknowledgement telemetria'

Numero di byte	Valore	Descrizione
1	0b01011001	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

5.5.10 Il messaggio 'Ritorna dalla SMP'

Il messaggio 'Ritorna dalla SMP' serve per comunicare ad AtmoCube di ritornare allo stato in cui si trovava prima di ricevere il comando di passare alla fase SMP. Questo messaggio può essere ricevuto dal satellite quando si trova nei seguenti stati:

- SMP *Communication State*;
- UOP *Communication State*.

La Tabella 5.11 descrive come è formato il messaggio.

Tabella 5.11: Struttura del messaggio 'Ritorna dalla SMP'

Numero di byte	Valore	Descrizione
1	0b01011010	byte identificativo del messaggio
1	0b00000000	byte che specifica la lunghezza del messaggio a partire dal 3^o byte
62	0b00000000	byte non utilizzati

Tabella 5.12: Messaggi che possono essere inviati dalla GS in base agli stati del Sistema Operativo

Messaggio	Earlier Operation Phase		Standard Operation Phase		Unilateral Operation Phase	Safe Mode State
	Beacon Frame State	Extended Beacon Frame State	Communication State	Transmit Telemetry State	Communication State	Safe Mode State
'Passa a EOP Beacon Frame State'		X	X		X	
'Passa a EOP Extended Beacon Frame State'	X				X	
'Programma il GPS'	X	X	X		X	
'Comando Dummy'	X	X	X		X	
'Passa a SMP'	X	X	X	X	X	
'Passa a SOP Measurement State'	X	X			X	
'Passa a SOP Transmitt Telemetry State'			X		X	
'Aggiorna i parametri di telemetria'			X		X	
'Acknowledgement telemetria'				X	X	
'Return from SMP'					X	X

Capitolo 6

Realizzazione del Sistema Operativo di AtmoCube

Le funzionalità del Sistema Operativo di Atmocube sono implementate per mezzo del PICOS18 che, come detto nel capitolo 3, è un Sistema Operativo *multitasking*. In questo capitolo vengono descritti i task, le ISR e le funzioni create per il Sistema Operativo di AtmoCube.

6.1 Struttura generale di un task

Per il Sistema Operativo di AtmoCube sono stati realizzati alcuni task, ciascuno con compiti differenti da eseguire. In generale, anche se i task svolgono operazioni molto diverse tra loro, sono caratterizzati da una struttura similare. Nella Figura 6.1 è rappresentato un esempio di diagramma di flusso la cui struttura può essere riconosciuta in qualsiasi task. Nel Sistema Operativo esistono delle variabili globali che, in base al valore che assumono, fanno sì che i task si pongano in un certo stato. Sempre nella Figura 6.1, si può notare una fase di inizializzazione del task, all'interno della quale possono trovare posto operazioni di configurazione di alcune variabili o registri del Sistema Operativo che devono essere eseguite una volta soltanto. Dopo la fase di inizializzazione, il task entra in un ciclo infinito di operazioni ed in questo modo si garantisce che il programma venga eseguito di continuo, ripetutamente. Seguendo il flusso delle operazioni, si arriva ad una struttura di controllo di tipo *switch*, la quale, in base al valore della variabile di stato (*state* nella figura), fa in modo che vengano eseguite le istruzioni di uno stato piuttosto che un altro. Se, ad esempio, la variabile *state* è uguale ad 'a', vengono eseguite le operazioni di inizializzazione di quello stato ed in seguito si entra in un ciclo all'interno del quale vengono eseguite le instruzioni dello stato 'a' finché la variabile *state* rimane uguale ad 'a'. Se, ad un certo punto, *state* cambia valore, ad esempio diventa uguale a 'b',

si ha l'uscita dal ciclo e l'esecuzione delle operazioni per terminare lo stato 'a'. Quando il flusso delle operazioni torna di nuovo allo *switch*, vengono eseguite le istruzioni per lo stato 'b'.

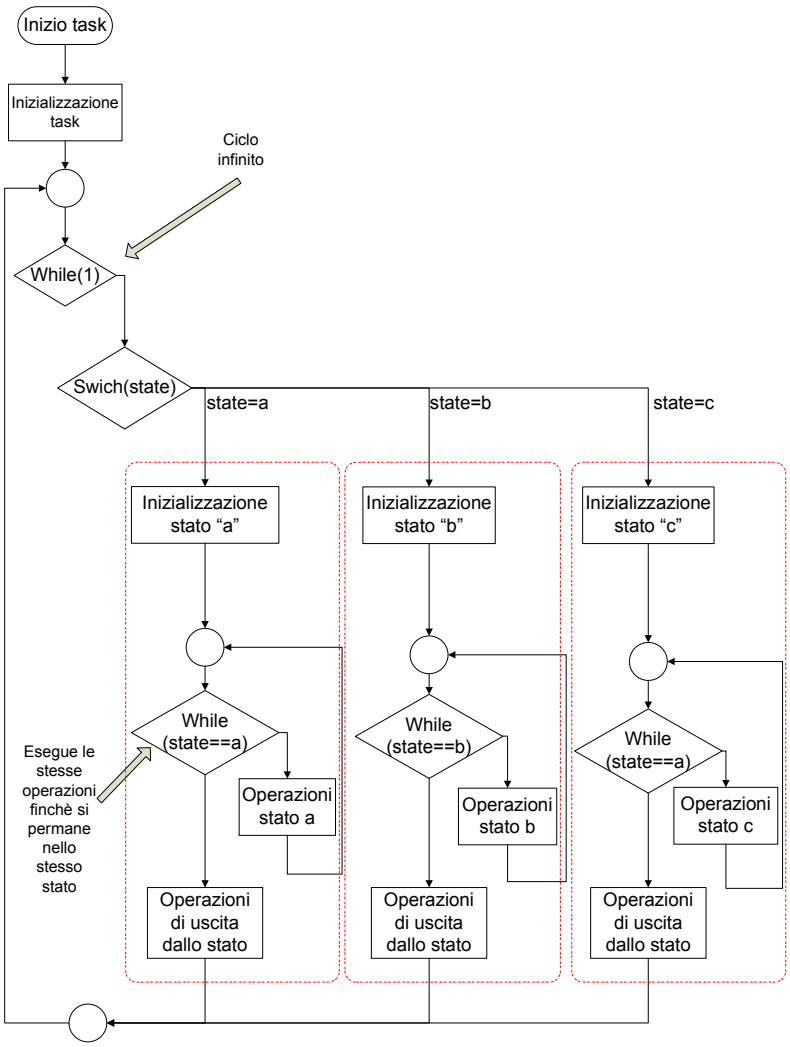


Figura 6.1: Diagramma di flusso tipico di un task.

Per i task realizzati per AtmoCube esistono due variabili che svolgono il ruolo di variabili di stato:

Phase per stabilire le fasi operative (inizialmente posto uguale a SP);

State per indicare lo stato operativo (inizialmente posto uguale a 1).

6.2 Le *Interrupt Service Routines* di AtmoCube

Il Sistema Operativo di AtmoCube utilizza degli ingressi di *interrupt* per interfacciarsi e sincronizzarsi con le periferiche esterne al microcontrollore. Ogni volta che una periferica fornisce un segnale di interrupt al microcontrollore, quest'ultimo esegue delle porzioni di codice che prendono il nome di ISR (vedi capitolo 3.2.6) ed in questo capitolo vengono descritte le ISR per AtmoCube.

6.2.1 ISR per il segnale PPS del modulo GPS

Il segnale PPS fornito dal modulo GSR, come detto nel capitolo 4.2, informa il microcontrollore che è stato appena acquisito un nuovo dato PVT. Non appena il microcontrollore riceve questo segnale dall'ingresso INT0, viene eseguita la ISR denominata 'PPS_ISR' che esegue le seguenti istruzioni:

1. se Phase è diversa da EOP genera l'evento 'MEASURE_TASK_PPS_EVENT' per informare il task measure (capitolo 6.6) che è stato acquisito un nuovo dato PVT;
2. viene cancellato il bit *flag* INT0IF del registro INTCON del microcontrollore per disabilitare l'*interrupt*.

6.2.2 ISR per un segnale in arrivo dalla OBR

Quando il modulo OBR riceve un nuovo *frame*, lo comunica al microcontrollore tramite la linea GDO2 collegata all'ingresso INT1 del microcontrollore. Nell'istante successivo il microcontrollore esegue la ISR 'INCOMING_ISR' che esegue le seguenti istruzioni:

1. pone la variabile globale INCOMING del sistema uguale a 1, per segnalare al task radio (capitolo 6.7) che il modulo OBR sta ricevendo un nuovo frame;
2. viene cancellato il bit *flag* INT1IF del registro INTCON3 per disabilitare l'*interrupt*.

6.2.3 ISR per la gestione della carica della batteria

Il microcontrollore PIC 18LF8722 è dotato di un modulo di nome *High/low Voltage Detector* (HLVD)[12] (vedi schema a blocchi in Figura 6.3) che permette al microcontrollore di controllare la tensione di alimentazione. Tramite tale modulo è possibile specificare via *software* una soglia di tensione e fare in modo che, quando la tensione supera tale livello, venga generato un segnale di *interrupt*. Si può inoltre impostare il modulo in modo che generi il segnale di *interrupt* quando la tensione scende sotto di una certa soglia. Il modulo HLVD viene gestito attraverso il registro di sistema HLVDCON così formato:

bit 7 VDIRMAG: *Voltage Direction Magnitude Select bit*:

- 1 = viene generato un *interrupt* quando la tensione supera la soglia impostata dai bit (HLVDL3:HLVDL0);
- 0 = viene generato un *interrupt* quando la tensione scende sotto la soglia impostata dai bit (HLVDL3:HLVDL0);

bit 6 inutilizzato;

bit 5 IRVST: *Internal Reference Voltage Stable Flag bit*:

- 1 = indica che il modulo genera un *interrupt* ad una determinata tensione;
- 0 = indica che il modulo HLVD non genera alcun *interrupt*;

bit 4 HLVDEN *High/Low Voltage Detect Enable bit*:

- 1 = HLVD è abilitato;
- 0 = HLVD è disabilitato;

bit 3–0 HLVDL3:HLVDL0 *Voltage Detection Limit bits*:

- 1111 = la soglia è impostata analogicamente dall'ingresso HLVDIN del microcontrollore;
- 1110 = massima tensione di soglia selezionabile (vedi Figura 6.2);
- ...
- 0000 = minima tensione di soglia selezionabile (vedi Figura 6.2);

Se, ad esempio, viene impostata una soglia di tensione a 2,60 V e il bit 7 viene impostato a '0', quando la tensione di alimentazione scende al di sotto di tale tensione, viene generato l'*interrupt* HLVDIF. Quando il microcontrollore riceve tale segnale di *interrupt*, chiama la ISR 'HLVDIN_ISR' che esegue le seguenti istruzioni:

- genera l'evento 'POWER_EVENT' per il task_power (capitolo 6.4);
- viene cancellato il bit *flag* HLVDIF del registro PIR2.

6.3 Il task di sistema

Il task di sistema (task_system) ha il compito di inizializzare il Sistema Operativo ed eseguire le operazioni della 'Start Phase'. Esso ha un livello di priorità 7 e quando si avvia il sistema operativo il suo stato è *READY*. Il task inizia entrando, come prima operazione, in un ciclo *while* infinito. Dato che il sistema si avvia con le variabili di stato impostate

Standard Operating Conditions (unless otherwise stated) Operating temperature -40°C ≤ TA ≤ +85°C for industrial								
Param No.	**Symbol**	**Characteristic**		**Min**	**Typ**	**Max**	**Units**	**Conditions**
D420		HLVD Voltage on VDD Transition High-to-Low	HLVDL = 0000	2.06	2.17	2.28	V	
			HLVDL = 0001	2.12	2.23	2.34	V	
			HLVDL = 0010	2.24	2.36	2.48	V	
			HLVDL = 0011	2.32	2.44	2.56	V	
			HLVDL = 0100	2.47	2.60	2.73	V	
			HLVDL = 0101	2.65	2.79	2.93	V	
			HLVDL = 0110	2.74	2.89	3.04	V	
			HLVDL = 0111	2.96	3.12	3.28	V	
			HLVDL = 1000	3.22	3.39	3.56	V	
			HLVDL = 1001	3.37	3.55	3.73	V	
			HLVDL = 1010	3.52	3.71	3.90	V	
			HLVDL = 1011	3.70	3.90	4.10	V	
			HLVDL = 1100	3.90	4.11	4.32	V	
			HLVDL = 1101	4.11	4.33	4.55	V	
			HLVDL = 1110	4.36	4.59	4.82	V	

Figura 6.2: Soglie di tensione impostabili tramite i bit HLVDL3:HLVDL0

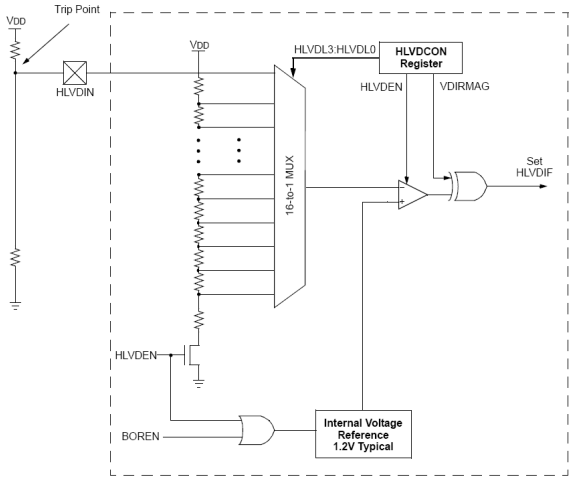


Figura 6.3: Schema a blocchi del modulo HLVD

per la fase SP, tramite la struttura di controllo *switch*, viene eseguito il codice per tale fase. Le prime operazioni consistono nell'abilitare la linea di alimentazione adibita al modulo GSR e nella creazione di un allarme che generi l'evento ALARM_EVENT ogni 30 minuti.

Nel caso in cui il sistema si riavvii, per evitare di fornire nuovamente il comando di espulsione dell'antenna, con conseguente consumo di energia, viene eseguito un algoritmo che determina se l'antenna è già stata espulsa o meno. Viene, quindi, effettuata una misura ADC della linea di controllo dell'espulsione dell'antenna[1] e ne viene salvato il risultato ottenuto sulla variabile 'ant'. Se il valore di quest'ultima variabile è uguale a 0, allora viene eseguita la procedura di espulsione dell'antenna che consiste in un'attesa dell'evento ALARM_EVENT (30 minuti) e nell'entrata in un ciclo *do–while* che viene eseguito fintantoché alt è uguale a 0. All'interno di tale ciclo vengono eseguite le seguenti operazioni:

- generazione di un impulso sulla linea di espulsione dell'antenna;
- nuova misura ADC sulla linea di controllo di espulsione dell'antenna e conseguente aggiornamento della variabile ant.

Infine, le ultime operazioni eseguite in questa fase consistono nella cancellazione dell'allarme generato e nella modifica delle variabili di stato nel modo seguente:

- Phase = EOP;
- State = 1.

Per le fasi EOP, SOP, SMP e UOP, il task esegue le stesse operazioni: ogni secondo verifica lo stato di ogni task. Se un task è in stato 'SUSPENDED', il task_system provvede ad attivarlo.

6.4 Il task per il controllo della tensione di batteria

Per evitare che la carica di batteria scenda al di sotto del limite minimo di funzionamento del microcontrollore, è stato implementato il task power che, in caso di necessità, pone il microcontrollore ed alcune periferiche in una condizione di basso consumo. All'avvio del Sistema Operativo, il task power è in stato di 'READY' ed, avendo il livello di priorità pari a 15, è il task a priorità maggiore. Come si vede dal diagramma di flusso in Figura 6.6, le prime operazioni che vengono eseguite dal task sono quelle di abilitazione del modulo HLVD ponendo a livello alto i bit IRVST e HLVDEN del registro del sistema HLVDCON. In seguito, il task entra in un ciclo *while* infinito, all'interno del quale

[1] La linea di controllo di espulsione dell'antenna ha una tensione pari a 0 V se l'antenna non è stata espulsa ed ha, invece, una tensione di 3 V se l'antenna è espulsa.

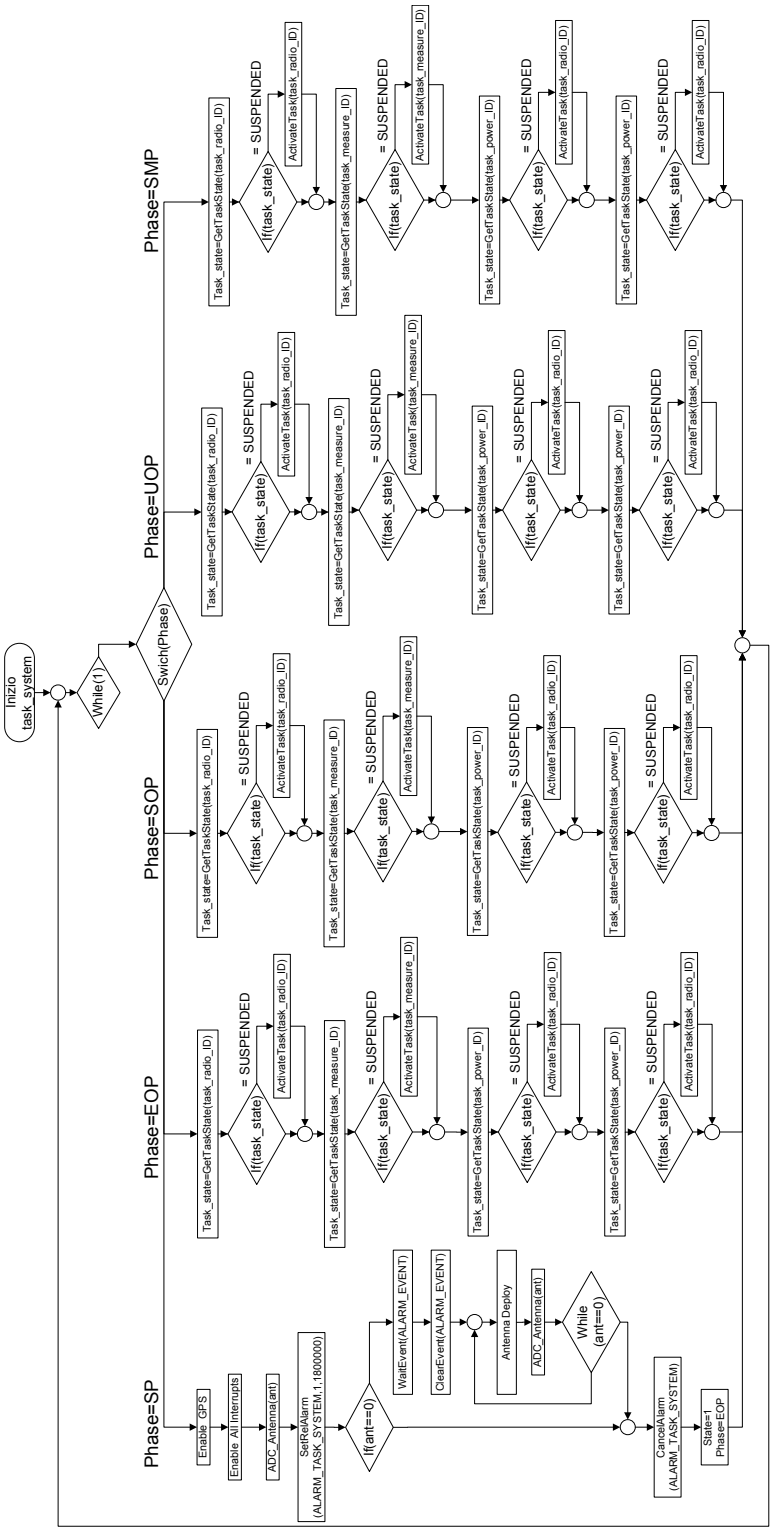


Figura 6.4: Diagramma di flusso del task_system

imposta una soglia di tensione pari a 2,12 V (vedi capitolo 6.2.3), imposta il bit VDIRMAG a livello basso e pone l'oscillatore primario del microcontrollore come sorgente di clock in modalità PRI_RUN[2]. Si pone poi in stato di 'WAITING' in attesa dell'evento 'POWER_EVENT', che avviene, date le impostazioni, quando la tensione di alimentazione scende al disotto dei 2,12 V. Quando questo succede, il task power torna in stato di 'READY' ed esegue le seguenti istruzioni:

- imposta la variabile globale POWER_PAUSED uguale a '1' per comunicare al task measure che le operazioni sono state sospese;
- imposta la variabile globale IDLE uguale a '1';
- attiva il task idle;
- disabilita tutte le alimentazioni esterne, tranne quella per il modulo GPS;
- imposta una soglia di tensione sul modulo HLVD pari a 2,96V;
- imposta il bit VDIRMAG a livello alto in modo che sia generato l'*interrupt* quando la tensione di alimentazione supera la soglia;
- imposta la modalità SEC_RUN[2], nella quale sia la CPU sia le periferiche vengono sincronizzate dal clock proveniente dal Timer1;
- si pone in stato di 'WAITING' in attesa dell'evento 'POWER_EVENT'.

Quando la tensione di alimentazione del microcontrollore supera la tensione di soglia impostata, il task power ritorna in stato di 'READY' e vengono eseguite nuovamente le prime istruzioni del ciclo *while*. La Figura 6.5 mostra un esempio di come vengono sospese le operazioni del satellite in caso di un abbassamento della tensione di alimentazione sotto 2,12 V e di come vengono ristabilite una volta che la tensione torna ad un valore di 2,96 V.

6.5 Il task idle

Quando la tensione di alimentazione scende al di sotto della soglia minima impostata, il task power attiva il task idle, che ha il compito di bloccare le operazioni ai task con priorità minore. Infatti, dato che il task idle ha priorità 14, se è in stato di 'RUNNING' gli altri task a priorità minore possono solo essere in stato di 'READY', 'WAITING' o 'SUSPENDED'.

Dalla Figura 6.7 si può vedere il diagramma di flusso del task idle.

La prima operazione che esegue il task consiste nell'entrare in un ciclo *while* che ripete l'operazione di *reset* del *watchdog timer* del microcontrollore finché la variabile

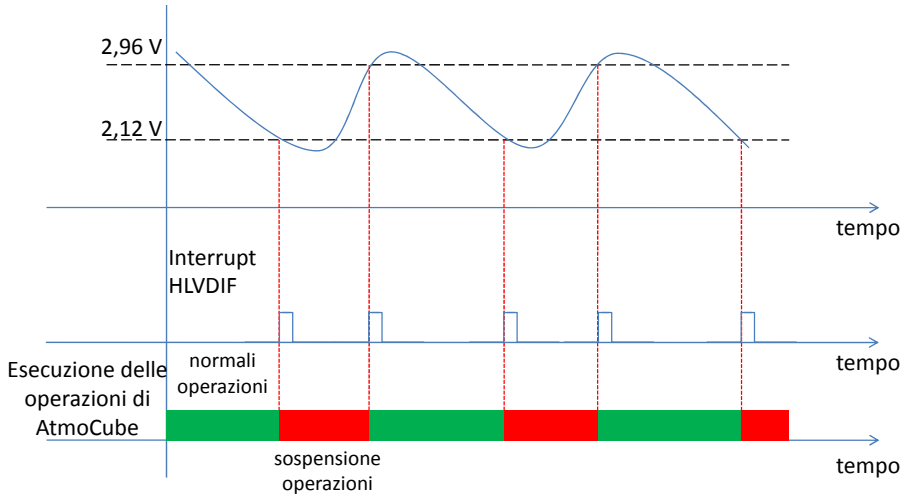


Figura 6.5: Esempio di funzionamento del sistema di sospensione delle operazioni del satellite in caso di bassa carica della batteria

IDLE è uguale a 1. Quando il task power pone la variabile IDLE uguale a zero, il task idle esce dal ciclo *while* ed esegue la la funzione *TerminateTask()* che lo pone in stato 'SUSPENDED'.

Nella Figura 6.8 sono mostrate le operazioni che vengono effettuate nel caso in cui la tensione di alimentazione scende sotto la soglia minima:

1. viene generato l'*interrupt*;
2. la routine 'HLVDIN_ISR' crea l'evento 'POWER_EVENT';
3. il task inizialmente in stato di 'RUNNING' passa in stato di 'READY';
4. il task power passa in stato di 'RUNNING' e:
   - pone la variabile IDLE uguale a 1;
   - attiva il task idle;
   - imposta la soglia di tensione alta in modo tale da generare l'*interrupt* quando la batteria si è caricata;
   - si pone in stato di 'WAITING' in attesa di un nuovo 'POWER_EVENT';
5. il task idle esegue un ciclo di istruzione *while* fintantoché IDLE rimane uguale a 1;
6. la tensione di alimentazione supera la soglia massima;
7. viene generato l'*interrupt*;

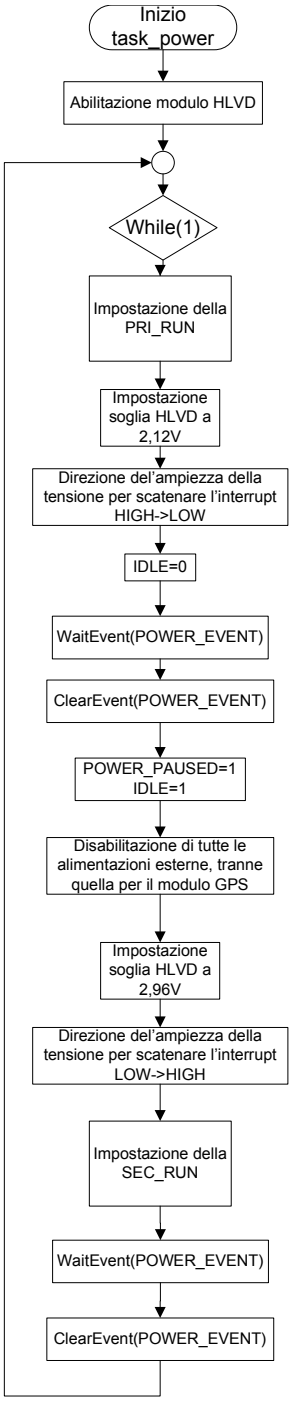


Figura 6.6: Diagramma di flusso del task_power

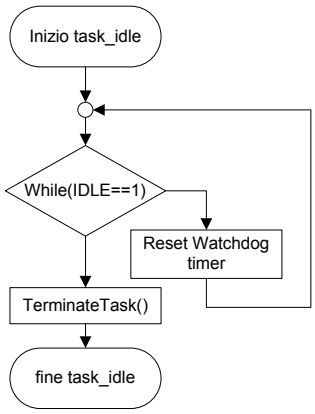


Figura 6.7: Diagramma di flusso del task idle.

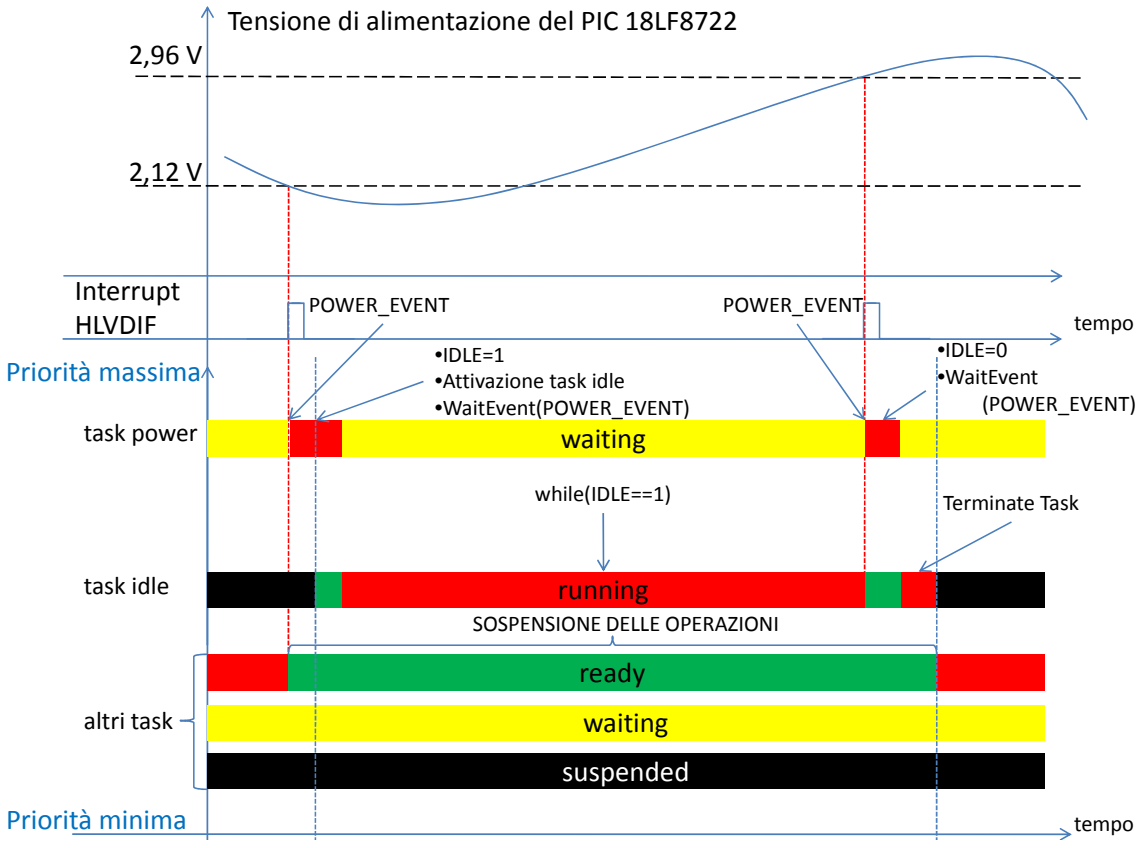


Figura 6.8: Sospensione delle operazioni dei task nel caso in cui avvenga una caduta di tensione.

8. la routine ‘HLVDIN_ISR’ crea l’evento ‘POWER_EVENT’;
9. il task power passa in stato di ‘RUNNING’ e:
   - pone la variabile IDLE uguale a 0;
   - imposta la soglia bassa di tensione in modo tale da generare l’*interrupt* quando la batteria si sta scaricando;
10. il task idle esce dal ciclo *while* e si sospende;
11. il task che era inizialmente in stato di ‘RUNNING’ e che poi è stato posto in ‘READY’, ritorna in stato di ‘RUNNING’;
12. gli altri task non subiscono alcun cambiamento in seguito alle operazioni appena descritte.

6.6 Il task per le misurazioni

Il task che prende il nome di ‘task measure’ si occupa di eseguire le operazioni che riguardano la raccolta, l’elaborazione ed il salvataggio delle informazioni necessarie ad AtmoCube. Queste operazioni consistono in:

- raccolta e salvataggio degli *HouseKeeping*;
- raccolta e salvataggio dei pacchetti di telemetria;
- calcolo della distanza tra AtmoCube e la GS;

Il task è suddiviso in 9 blocchi, uno per ogni stato[2], all’interno dei quali sono implementate le operazioni che il task deve eseguire.

Nei diagrammi di flusso riportati in Figura 6.9 è rappresentato come viene realizzata la divisione del task nei vari blocchi di codice. Si nota quindi che il task measure è così diviso:

- EOP:
  - EOP *Beacon Frame State*;
  - EOP *Extended Beacon Frame State*;
- SOP:
  - SOP *Measure State*;

[2]Si nota che gli stati del task measure sono 9 e non 10 poichè durante lo stato per la SP il task measure è sempre sospeso.

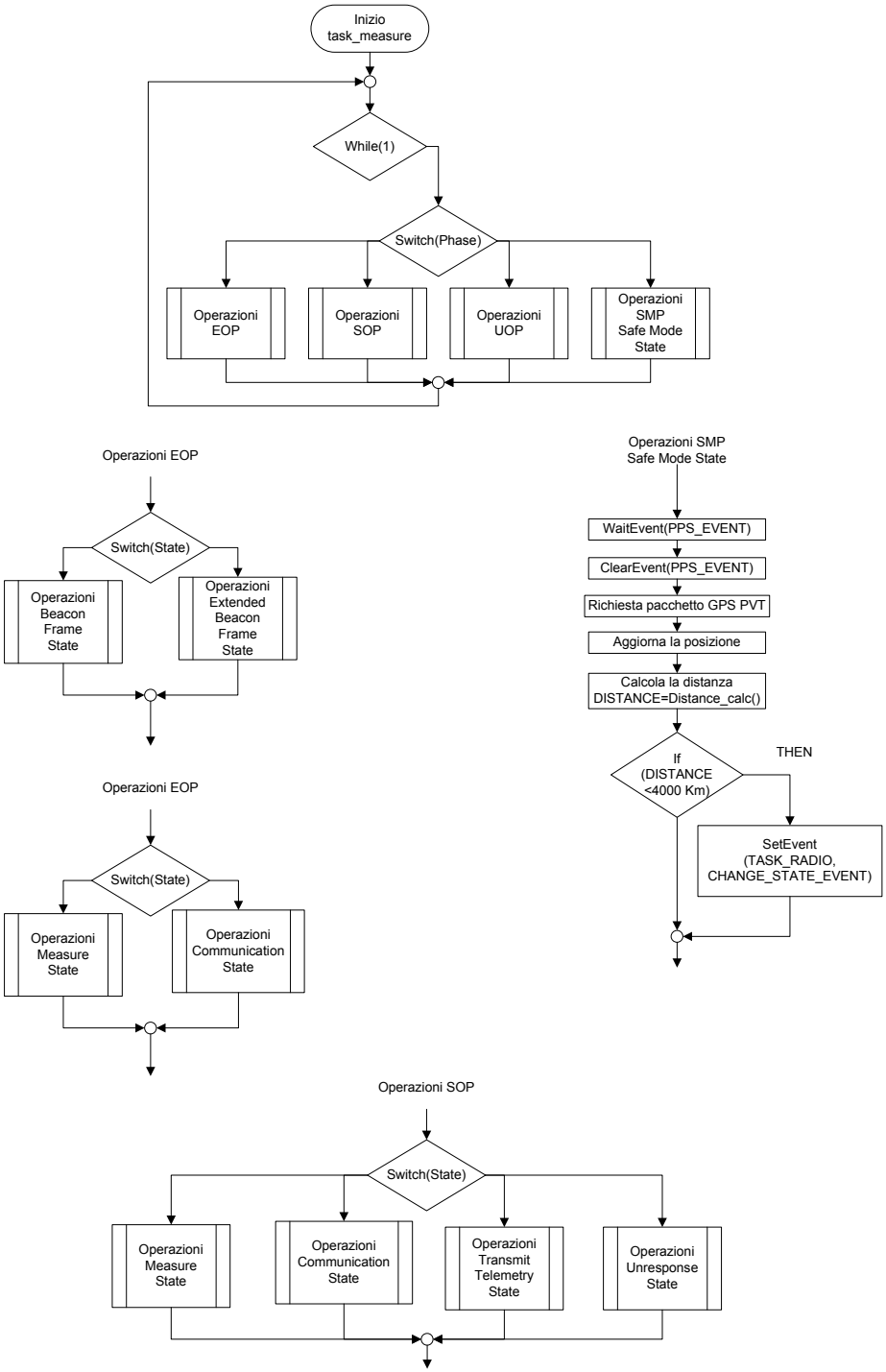


Figura 6.9: Diagramma di flusso generale del task measure e dello stato SMP *Safe Mode State*

  - SOP *Communication State*;
  - SOP *Transmit Telemetry State*;
  - SOP *Unresponse State*;

- UOP:
  - UOP *Measure State*;
  - UOP *Communication State*;

- SMP:
  - SMP *Safe Mode State*;

Le operazioni che il task measure svolge negli stati sopraelencati sono descritte con maggior dettaglio nei capitoli seguenti.

6.6.1 Lo stato EOP *Beacon Frame State* per il task measure

Il task measure, in questo stato, ha il solo compito di aggiornare, ogni minuto, gli *HouseKeeping*. Le operazioni eseguite in questo stato sono (vedi Figura 6.10):

- attivazione dell'allarme 'ALARM_TASK_MEASURE' in modo che venga generato, ogni minuto, l'evento 'MEASURE_TASK_EVENT';
- la variabile MEASURE_COUNTER viene azzerata;
- entra in un ciclo *while* le cui operazioni, che vengono eseguite fintantoché lo stato non cambia, sono:
  - attesa dell'evento 'MEASURE_TASK_EVENT';
  - disabilitazione dell'evento;
  - incremento della variabile MEASURE_COUNTER;
  - aggiornamento degli *HouseKeeping–scienza*;
  - aggiornamento degli *HouseKeeping–sistema di tipo 1*;
  - aggiornamento degli *HouseKeeping–sistema di tipo 2*;
- successivamente al ciclo *while*, viene disattivato l'allarme 'ALARM_TASK_MEASURE'.

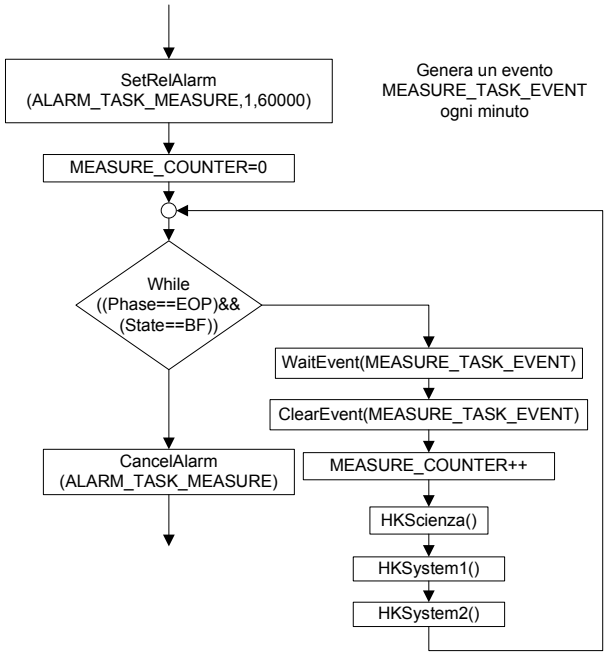


Figura 6.10: Operazioni eseguite dal task measure nello stato EOP *Beacon Frame State*.

6.6.2 Lo stato EOP *Extended Beacon Frame State* per il task measure

In questo stato, il task effettua delle misurazioni di prova della durata di un minuto e ne salva i relativi pacchetti di telemetria 1 e 2. Dato che in fase EOP la temporizzazione utilizzata non proviene dal segnale PPS, si fa uso di un allarme che genera ogni secondo l'evento 'MEASURE_TASK_EVENT'. Per il conteggio degli eventi avvenuti viene utilizzata la variabile EVEN_COUNTER ed in base al suo valore viene deciso quali operazioni eseguire. Nel primo minuto di ogni ora, corrispondente al caso in cui EVEN_COUNTER è minore o uguale a 60 si devono effettuare le misure scientifiche e le operazioni che vengono eseguite sono:

- se EVENT_COUNTER è uguale a 0 viene dato il comando di *Reset* e successivamente di *Start* allo spettro–dosimetro per fare iniziare una nuova misurazione;
- se EVENT_COUNTER è diversa da 0 e multipla di 10 viene creato un nuovo pacchetto di telemetria 1 e vengono eseguite le seguenti operazioni:
  - viene acquisito un nuovo dato dal magnetometro;
  - viene richiesto un nuovo pacchetto PVT dal GPS;
  - viene salvato un nuovo pacchetto di telemetria 1.

- Se EVENT_COUNTER è uguale a 60 viene creato un nuovo pacchetto di telemetria 2 e vengono eseguite le seguenti operazioni:
  - viene dato il comando di Stop allo spettro–dosimetro;
  - viene dato il comando di Read allo spettro–dosimetro salvando temporaneamente le 3 copie dei dati ricevuti;
  - viene effettuato un controllo a maggioranza delle tre copie di dati;
  - viene salvato un nuovo pacchetto di telemetria 2.

In seguito, se EVEN_COUNTER è multiplo di 60, vengono salvati i nuovi *HouseKeeping*. Quando EVEN_COUNTER è uguale a 3600 (corrispondente ad 1 ora) viene azzerata, invece, se è minore di tale valore viene incrementata. La Figura 6.11 mostra il diagramma di flusso delle operazioni eseguite dal task measure in questo stato.

Visto che i dati acquisiti in questa fase hanno il solo scopo di informare la GS se il satellite è in grado o meno di eseguire le misure, i nuovi pacchetti di telemetria vengono salvati ogni ora sovrascrivendo quelli dell'ora precedente.

6.6.3 Lo stato SOP *Measure State* per il task measure

In questo stato vengono effettuate le misure scientifiche e salvati i pacchetti di telemetria come descritto del capitolo 5.1. Come si vede dalla Figura 6.12, come prima operazione, si attende l'evento PPS_EVENT che indica che il GPS ha appena acquisito un nuovo dato PVT ed in seguito si incrementa la variabile PPS_COUNTER che conta il numero di acquisizioni del GPS. Si eseguono poi le seguenti operazioni:

- si valuta il valore di PPS_COUNTER e se esso è multiplo di PPS_TEL1 si esegue un nuova misura dal magnetometro;
- si acquisisce un nuovo pacchetto PVT dal GPS;
- se PPS_COUNTER $(\mathrm{mod}\ PPS\_TEL2)$ è uguale a PPS_SDD_INT viene dato il comando di Stop allo spettro–dosimetro;
- se PPS_COUNTER è multiplo di PPS_TEL2 vengono letti e salvati temporaneamente le tre copie dei dati dallo spettrodosimetro;
- se PPS_COUNTER è multiplo di PPS_TEL1 viene salvato un nuovo *HouseKeeping*–scienza e salvato un nuovo pacchetto di telemetria1;
- se PPS_COUNTER è multiplo di PPS_TEL2 viene salvato un nuovo *HouseKeeping*–sistema di tipo 1 e un nuovo pacchetto di telemetria 2;

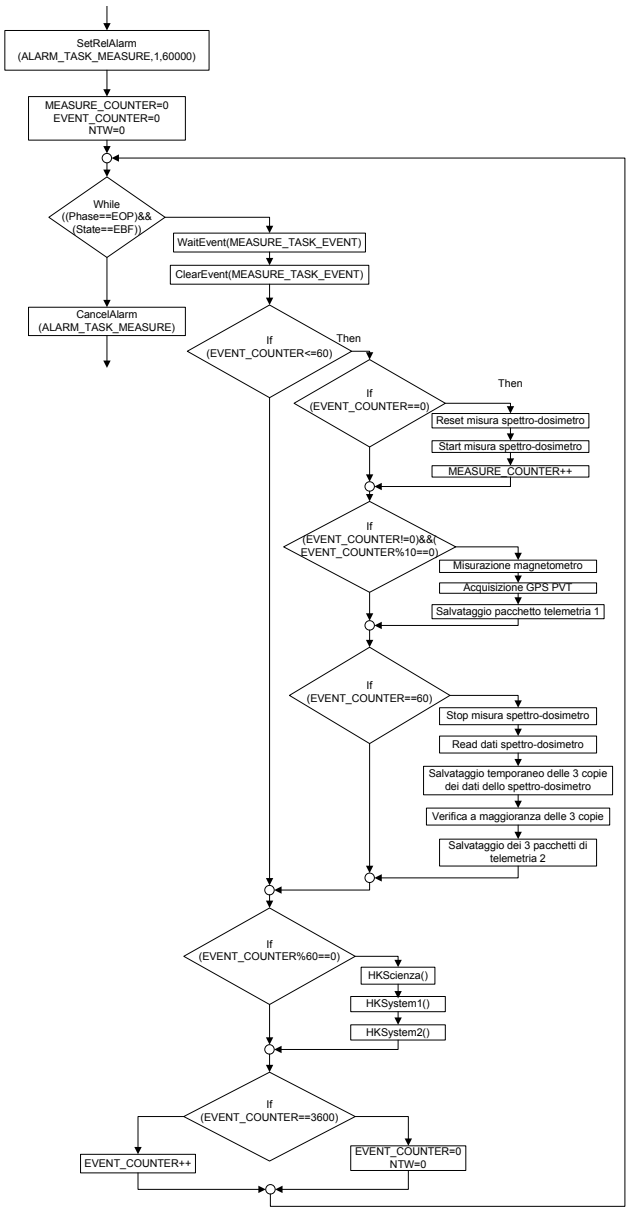


Figura 6.11: Diagramma di flusso delle operazioni eseguite dal task measure nello stato EOP *Extended Beacon Frame State*

- se PPS_COUNTER è multiplo di PPS_TEL3 viene salvato un nuovo *HouseKeeping*–sistema di tipo 2 e un nuovo pacchetto di telemetria 3.

Viene poi valutato il valore della variabile POWER_PAUSED, la quale diventa pari a 1 se le operazioni sono state sospese a causa della caduta di tensione dell'alimentazione (capitolo 6.4). Se durante le misurazioni non è avvenuta alcuna sospensione delle operazioni le misure appena effettuate sono da ritenersi valide e quindi i pacchetti salvati possono essere confermati in memoria; aggiornando il valore di FTW al nuovo valore assunto da NTW.

Come ultima operazione viene calcolata la posizione tra satellite e GS e se questa risulta minore di 4000 Km lo stato passa a SOP *Communication State*.

6.6.4 Lo stato SOP *Communication State* per il task measure

In questo stato il task measure aspetta l'acquisizione, da parte del GPS, di un nuovo dato PVT per ricalcolare la distanza tra satellite e GS ed in seguito aggiorna gli *HouseKeeping* per l'invio dei *BeaconFrame*. Il diagramma a stati delle operazioni è raffigurato nella Figura 6.13

6.6.5 Gli stati SOP *Transmit Telemetry State* e SOP *Unresponse State* per il task measure

In questi stati non vengono effettuate operazioni di salvataggio di pacchetti di telemetria o aggiornamento degli *Housekeeping* e quindi nello stato SOP *Transmit Telemetry State* il task measure attende una nuova acquisizione di un dato PVT e ricalcola la distanza tra la GS ed AtmoCube e, nello stato SOP *Unresponse State*, si attende un nuovo evento 'PPS_EVENT' senza nemmeno richiedere il dato PVT al modulo GPS.

6.6.6 La fase UOP per il task measure

Le operazioni eseguite nei due stati di questa fase sono le stesse eseguite nella fase SOP, in particolare le operazioni eseguite nello stato UOP *Measure State* sono le stesse dello stato SOP *Measure State* e quelle eseguite nello stato UOP *Communication State* sono le stesse eseguite nello stato SOP *Communication State*.

6.6.7 Lo stato *Safe Mode Phase State* per il task measure

In questo stato il task measure aspetta una nuova acquisizione PVT dello SGR ed acquisisce il pacchetto PVT. Calcola poi la distanza tra la GS ed AtmoCube e se questa risulta minore di 4000 Km genera l'evento 'CHANGE_STATE_EVENT' per il task

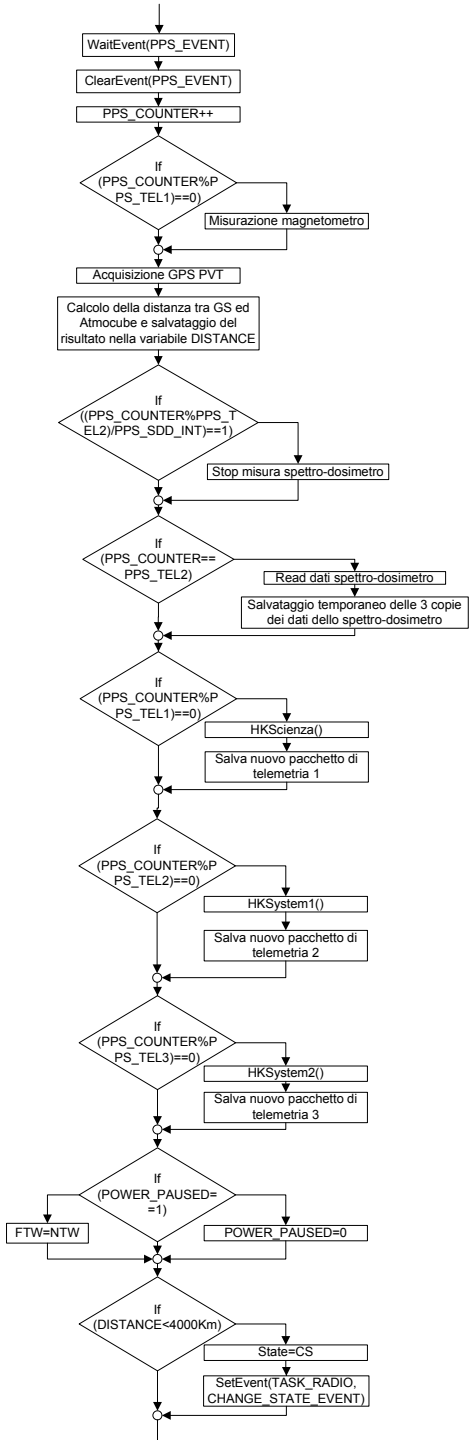


Figura 6.12: Diagramma di flusso delle operazioni eseguite dal task measure nello stato SOP *Measure State*

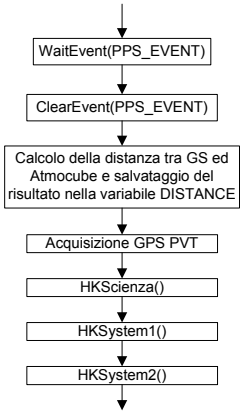


Figura 6.13: Diagramma di flusso delle operazioni eseguite dal task measure nello stato SOP *Communication State*

radio. Le operazioni eseguite in questo stato dal task measure sono raffigurate nella Figura 6.9.

6.7 Il task per la gestione della radio

Il task radio si occupa della trasmissione e ricezione dei frame, della lettura della SRAM di telemetria, della trasmissione *Stop and Wait* dei pacchetti di telemetria; esso ha un valore di priorità pari a 10 e, come per il task measure, è formato da 9 blocchi, uno per ogni stato implementato (vedi Figura 6.14).

6.7.1 La fase EOP per il task radio

Nello stato EOP *Beacon Frame State* devono essere trasmessi, ogni minuto, i *Beacon Frame*. Come prime operazioni, viene attivato l'allarme 'ALARM_TASK_RADIO' in modo tale da generare l'evento 'RADIO_TASK_EVENT' ogni minuto e viene azzerata la variabile BEACON_COUNTER che conta il numero di *Beacon Frame* inviati in questo stato. Viene poi eseguito un ciclo *while* fintantoché si resta in questo stato e le operazioni eseguite al suo interno sono (vedi Figura 6.15):

- attesa dell'evento 'RADIO_TASK_EVENT';
- incremento della variabile BEACON_COUNTER;
- creazione di un nuovo payload contenente un *Beacon Frame* salvando i Byte relativi nella porzione di SRAM adibita al salvataggio temporaneo del primo buffer di trasmissione (vedi capitolo 5.3).

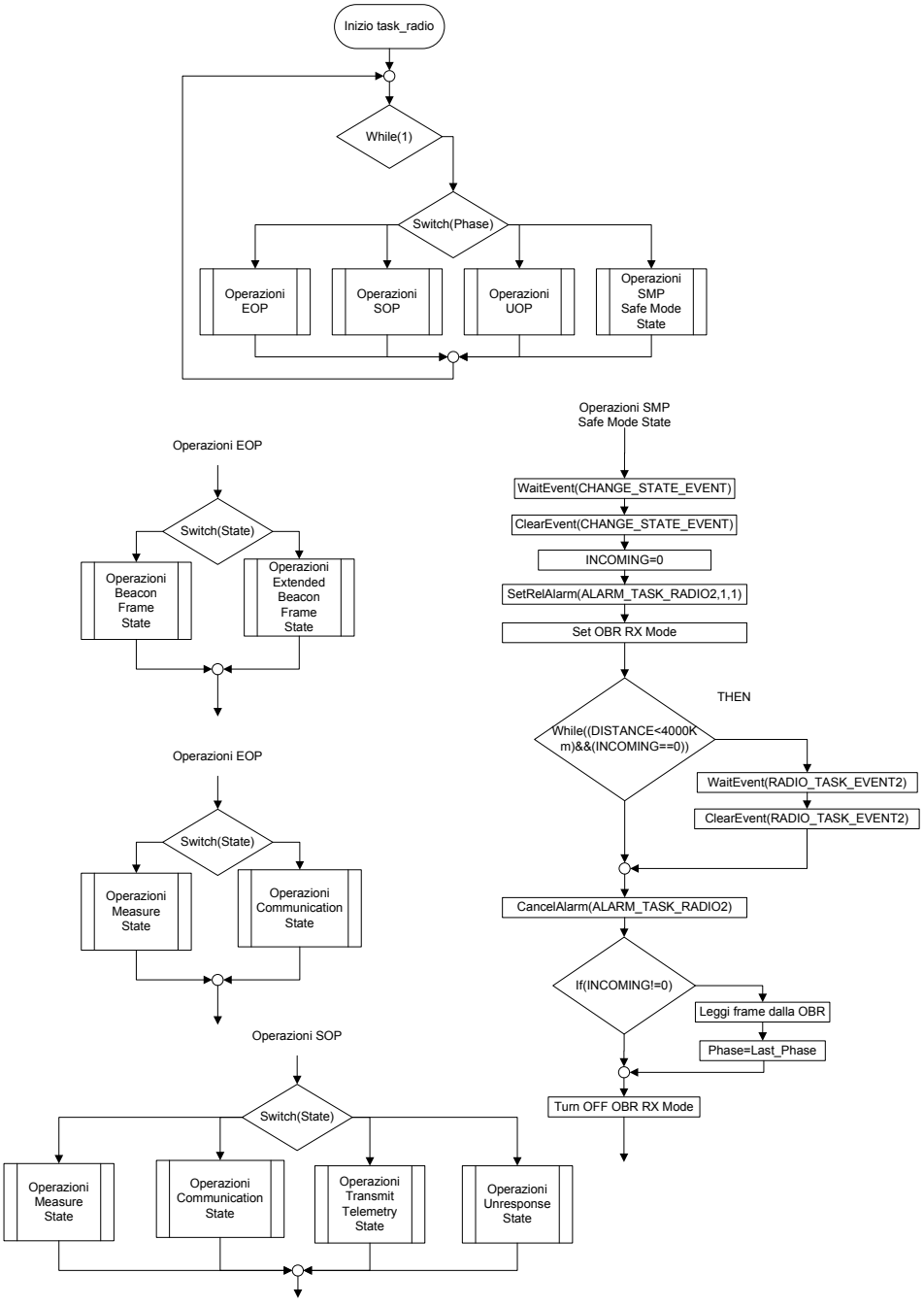


Figura 6.14: Diagramma di flusso del task radio e operazioni eseguite nello stato *Safe Mode Phase State*.

- trasmissione di un frame con il nuovo payload;
- attesa che arrivi un nuovo frame dalla GS per un tempo pari a $2 * T_{Timeout}$;
- se è arrivato un frame valido, vengono eseguiti i comandi contenuti all'interno.
- se la variabile BEACON_COUNTER supera il valore massimo di *Beacon Frame* inviati senza aver ricevuto alcun *frame* dalla GS, si passa alla fase UOP.

Per quanto riguarda lo stato *Extended Beacon Frame State*, le uniche differenze con lo stato precedente sono:

- la creazione ed il salvataggio temporaneao nella SRAM di 6 payload contententi un *Beacon Frame* esteso;
- la trasmissione di 6 *frame* contenenti ciascuno un payload;

Il diagramma di flusso delle operazioni eseguite dal task radio nello stato EOP *Extended Beacon Frame State* è raffigurato nella Figura 6.16.

6.7.2 Lo stato SOP *Communication State* per il task radio

Le operazioni eseguite in questo stato, raffigurate nella Figura 6.17, consistono nell'invio di un *Beacon Frame* ogni 5 secondi e sono sostanzialmente le stesse fatte per lo stato EOP *Beacon Frame State*. In questo stato è possibile che venga ricevuto:

- il comando di aggiornamento dei parametri di telemetria;
- il comando a passare allo stato SOP *Transmit Telemetry State*.

Ogni volta che arriva un nuovo messaggio dalla GS, viene azzerata la variabile UNRESPONSE_COUNTER che conta il numero di finestre di accesso all'interno delle quali non è stato ricevuto alcun messaggio dalla GS. Se, invece, dopo aver trasmesso un *Beacon Frame*, non arriva alcun messaggio, viene valutata la distanza tra il satellite e la GS e, se risulta maggiore di 4000 Km, lo stato passa a SOP *Unresponse Counter*.

6.7.3 Lo stato SOP *Unresponse State* per il task radio

Quando il task measure si trova in questo stato, viene valutato il valore della variabile UNRESPONSE_COUNTER e se è minore di 10, lo stato passa a SOP *Measure State*, altrimenti, se il numero di finestre trasmissive è maggiore di 10, si ha il passaggio alla fase UOP nello stato UOP *Measure State*. Le operazioni eseguite in questo stato sono raffigurate nella Figura 6.18.

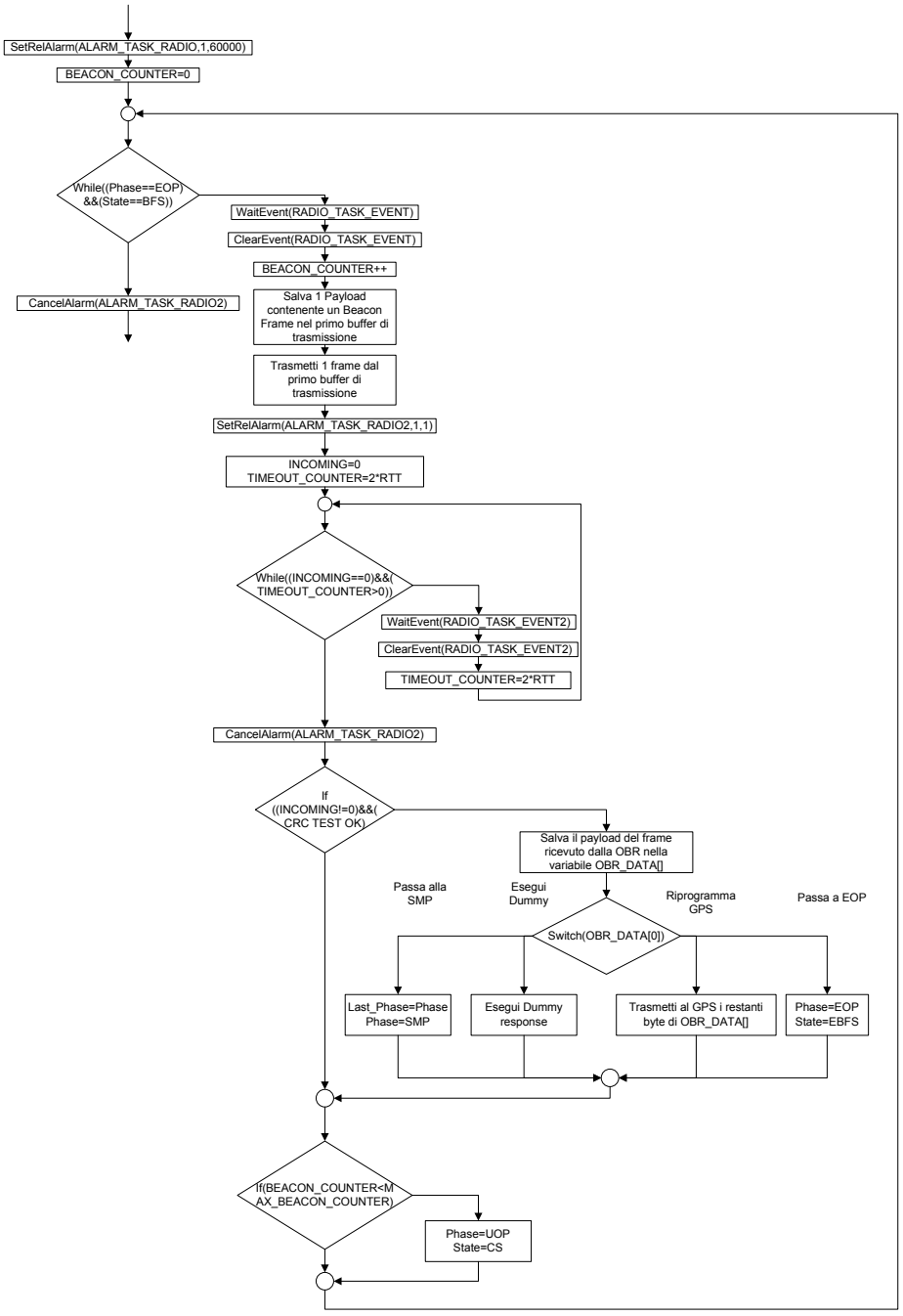


Figura 6.15: Diagramma di flusso delle operazioni eseguite dal task radio nello stato EOP *Beacon Frame State*

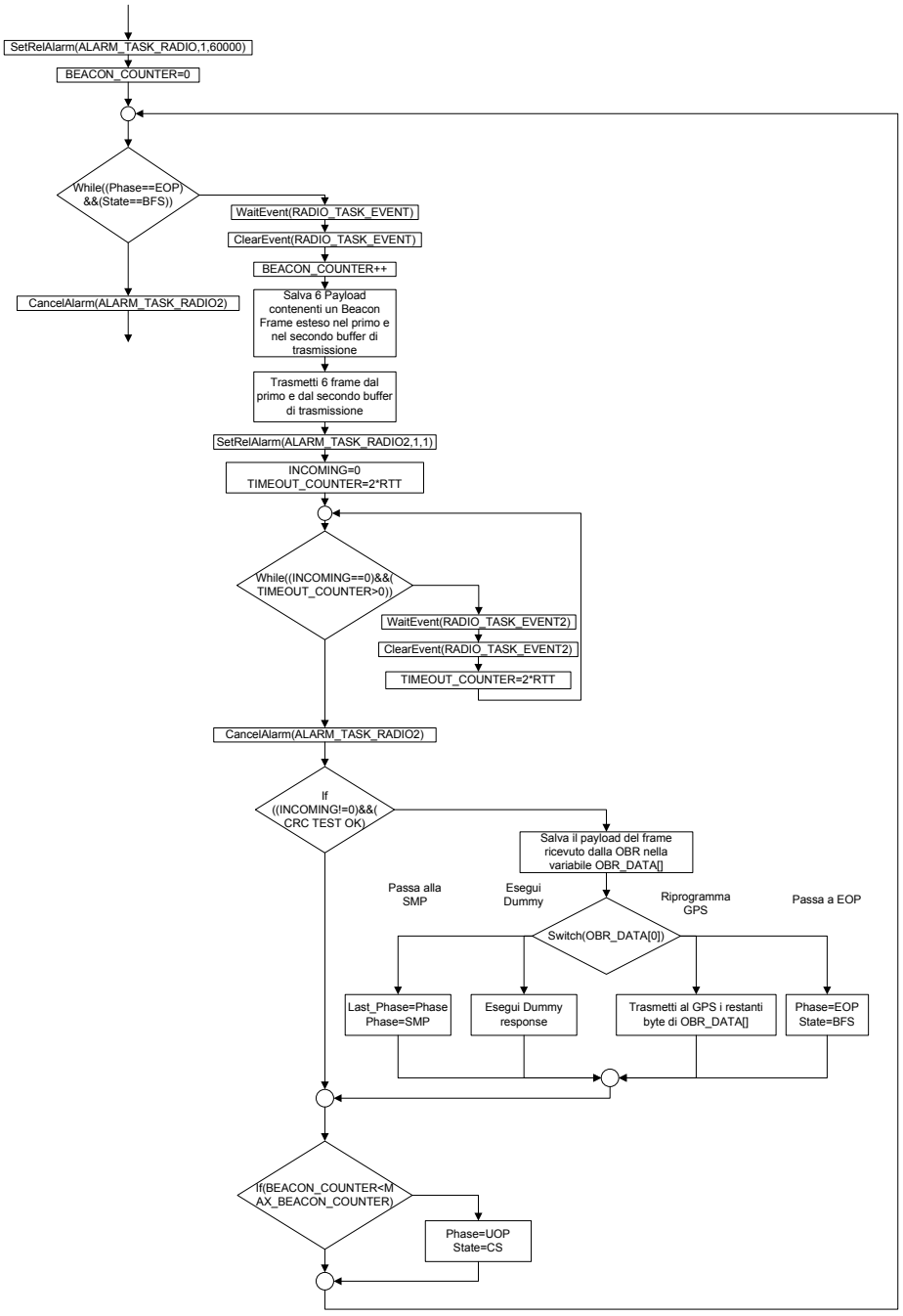


Figura 6.16: Diagramma di flusso delle operazioni eseguite dal task radio nello stato EOP *Extended Beacon Frame State*

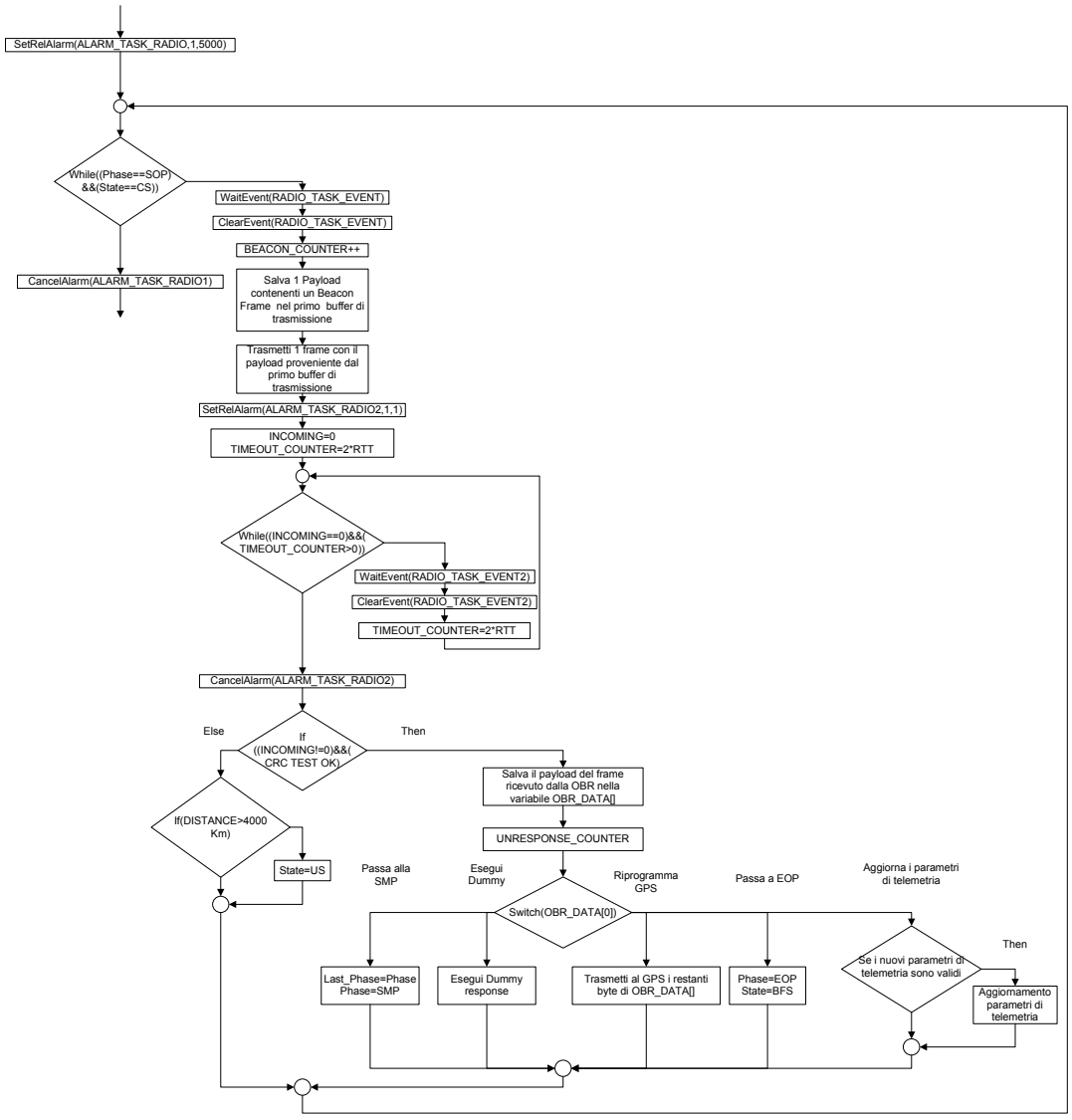


Figura 6.17: Diagramma di flusso delle operazioni eseguite dal task radio nello stato SOP *Communication State*

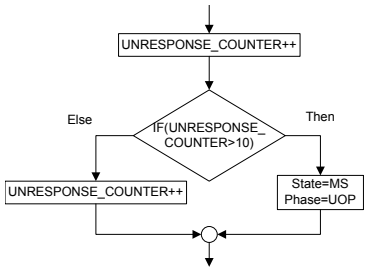


Figura 6.18: Diagramma di flusso delle operazioni eseguite dal task radio nello stato SOP *Unresponse State*

6.7.4 Lo stato SOP *Transmit Telemetry State* per il task radio

In questo stato, il task radio esegue la trasmissione *Stop and Wait* dei pacchetti di telemetria e dal diagramma di flusso in Figura 6.19 si possono vedere le operazioni eseguite che sono:

- si valuta, inanzitutto, se la memoria non è vuota, se FTR è diverso da FTW significa che ci sono pacchetti da trasmettere;
- se FTW è diverso da FTR, viene preparato il primo payload da trasmettere nel modo seguente:
  - la variabile NTR1 assume il valore di FTR che indica il primo byte da trasmettere;
  - si salva il primo payload da trasmettere nella prima porzione di memoria temporanea dei payload incrementando la variabile NTR1 in modo che indichi il primo byte da trasmettere nel prossimo payload;
  - si trasmettono i frame contenenti il payload preparato;
  - la variabile LAST_TRANSMITTED, che indica la posizione del payload appena trasmesso, viene posta pari ad 1;
  - le variabili NTR e NTR2 assumono il valore di NTR1;
  - viene preparato il secondo payload da trasmettere a partire dal byte indicato da NTR2, salvandolo temporaneamente nella seconda porzione di memoria per i payload;
  - NTR2 viene modificato.

Seguendo il diagramma di flusso delle operazioni, si entra in ciclo *while* che ripete le operazioni fino a quando la distanza tra AtmoCube e la GS è minore di 4000 Km. La prima sequenza di operazioni eseguite dentro al ciclo consiste nell'attesa, per un tempo

pari a $T_{Timeout}$ di un messaggio di '*Acknwoledgement*' dalla GS che conferma l'avvenuta ricezione dell'ultimo pacchetto di telemetria inviato. Possono ora capitare 3 casi:

1. se scade il tempo di attesa e non è stato ricevuto alcun messaggio, si ritrasmette l'ultimo payload e si valuta la variabile LAST_TRASMITTED per conoscere quale payload è stato trasmesso. Se tale variabile è uguale ad 1 si trasmette il payload nella prima posizione di memoria temporanea mentre, se è uguale a 2, si trasmette il payload nella seconda posizione;

2. se arriva un messaggio dalla GS, che però non corrisponde ad un messaggio di tipo '*Acknwoledgement*', viene ritrasmesso l'ultimo payload come descritto al punto 1;

3. se arriva un messaggio di tipo '*Acknwoledgement*', significa che è stato ricevuto l'ultimo pacchetto di telemetria inviato e viene confermata l'avvenuta trasmissione ponendo FTR uguale a NTR. Si procede poi all'invio del successivo payload, precedentemente preparato, valutando il valore della variabile LAST_TRANSMITTED. Se tale variabile è uguale ad 1 allora bisogna:

   - trasmettere il payload presente nella seconda porzione di memoria temporanea;
   - LAST_TRASMITTED viene posta uguale a 2;
   - NTR e NTR1 assumono il valore di NTR2;
   - si prepara il prossimo payload da trasmettere a partire dal byte indicato da NTR1 e lo si salva nella prima porzione di memoria temporanea.

   Se, invece, LAST_TRASMITTED è uguale a 2 allora bisogna eseguire le seguenti operazioni:

   - trasmettere il payload presente nella prima porzione di memoria temporanea;
   - LAST_TRASMITTED viene posta uguale a 1;
   - NTR e NTR2 assumono il valore di NTR1;
   - si prepara il prossimo payload da trasmettere a partire dal byte indicato da NTR2 e lo si salva nella seconda porzione di memoria temporanea.

La trasmissione e la preparazione dei payload non avviene sempre allo stesso modo, ma in maniera differente in base a quale pacchetto di telemetria si deve trasmettere come descritto nei capitoli 5.1 e 5.2.

Il task radio, nello stato SOP *Measure State* e nello stato UOP *Measure State* non deve eseguire alcuna operazione e quindi si pone in stato di 'WAITING', in attesa dell'evento CHANGE_STATE_EVENT.

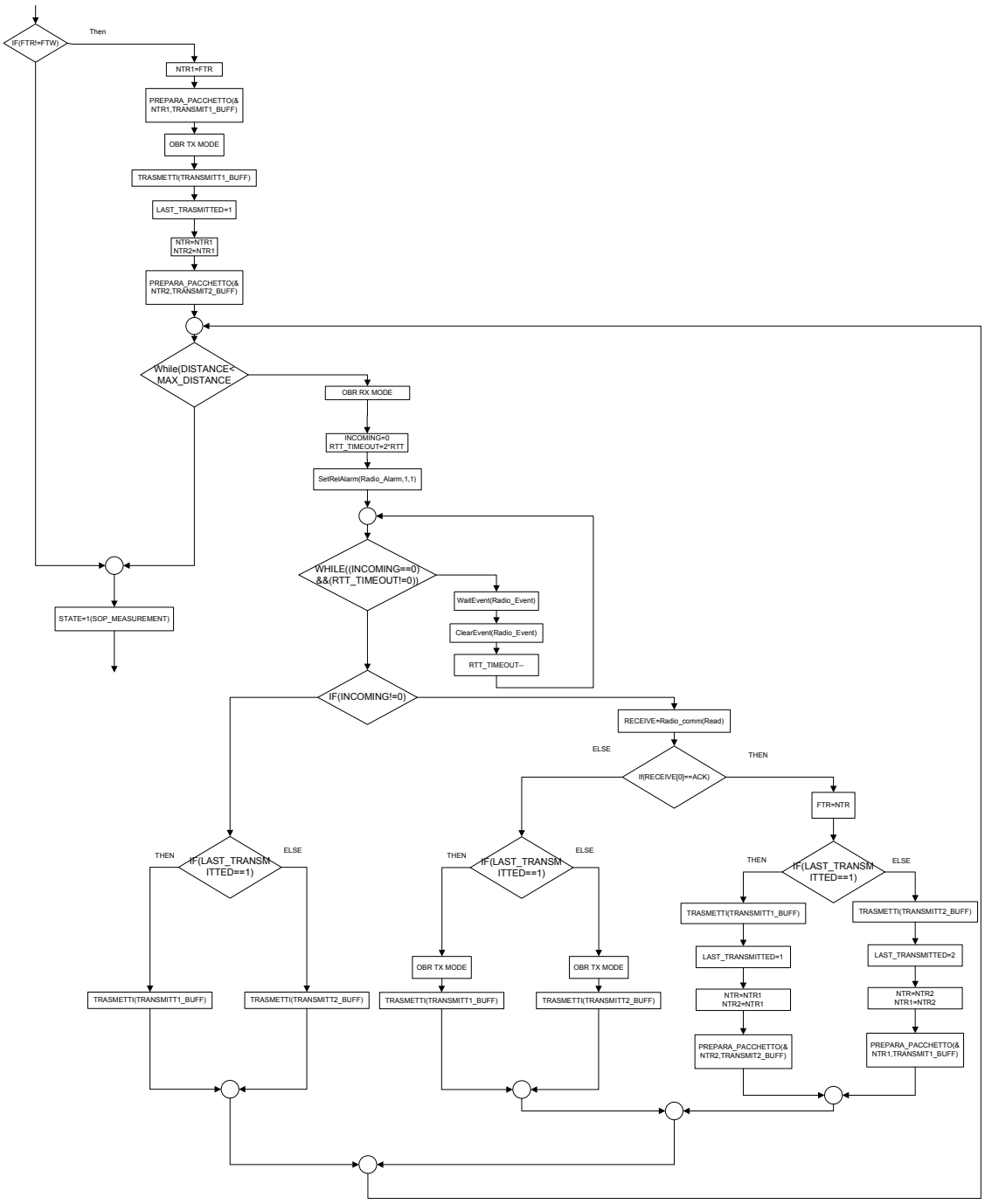


Figura 6.19: Diagramma di flusso delle operazioni eseguite dal task radio nello stato SOP *Telemetry Transmit State*

6.7.5 Lo stato UOP *Communication State* per il task radio

In questo stato il task radio trasmette i pacchetti di telemetria senza verificare se la GS li abbia ricevuti correttamente. Le operazioni in questo stato si ripetono fintantoché la distanza tra la GS ed AtmoCube rimane minore di 4000 Km e consistono nella preparazione dei payload e nel loro invio, come avviene per lo stato SOP *Telemetry Transmit State*. Ad ogni invio di un payload, viene preparato il successivo payload da trasmettere e si aspetta un messaggio dalla GS per un tempo pari a $T_{Timeout}$. Se durante tale periodo di tempo non arriva alcun messaggio, si procede con l'invio dell'ultimo payload preparato e con la preparazione di quello successivo. Se invece arriva un messaggio dalla GS, lo stato diventa EOP *Communication State*. Nella Figura 6.20 è raffigurato il diagramma di flusso delle operazioni eseguite nello stato UOP *Communication State*.

6.7.6 Lo stato *Safe Mode Phase State* per il task radio

Le operazioni eseguite dal task radio in questo stato sono raffigurate nella Figura 6.14 e consistono:

- nell'attesa dell'evento CHANGE_STATE_EVENT che deve essere generato dal task measure quando è stata calcolata la nuova distanza tra AtmoCube e la GS;
- nel porre la variabile INCOMING uguale a zero;
- nell'attesa, mentre il satellite si trova nella finestra trasmissiva, del messaggio proveniente dalla GS che provochi lo sblocco dalla fase SMP;
- se arriva il comando atteso, ponendo la variabile Phase uguale a last_phase, la fase torna quella nell'istante in cui il satellite aveva ricevuto il comando di passare alla fase SMP.

6.8 Analisi delle priorità di task e risorse

I task realizzati per il Sistema Operativo di Atmocube svolgono le azioni necessarie per coordinare le operazioni del satellite. Ogni task si occupa di svolgere un certo insieme di operazioni, che ha differenti esigenze temporali rispetto agli altri e, per questo motivo, gli viene assegnato un livello di priorità diverso dagli altri. Si è reso necessario implementare un sistema di gestione delle risorse condivise, all'interno del Sistema Operativo, in modo che le risorse siano utilizzate da un task alla volta. La gestione delle risorse implementata segue le direttive indicate dal *protocollo di massima priorità OSEK* (capitolo 3.2.12). Per tale motivo il task a massima priorità nel sistema è il task power, il quale, dato il suo livello di priorità (15) è in grado di bloccare qualsiasi altra operazione eseguita dagli altri task in caso di una caduta di tensione ed il task idle ha un livello 14

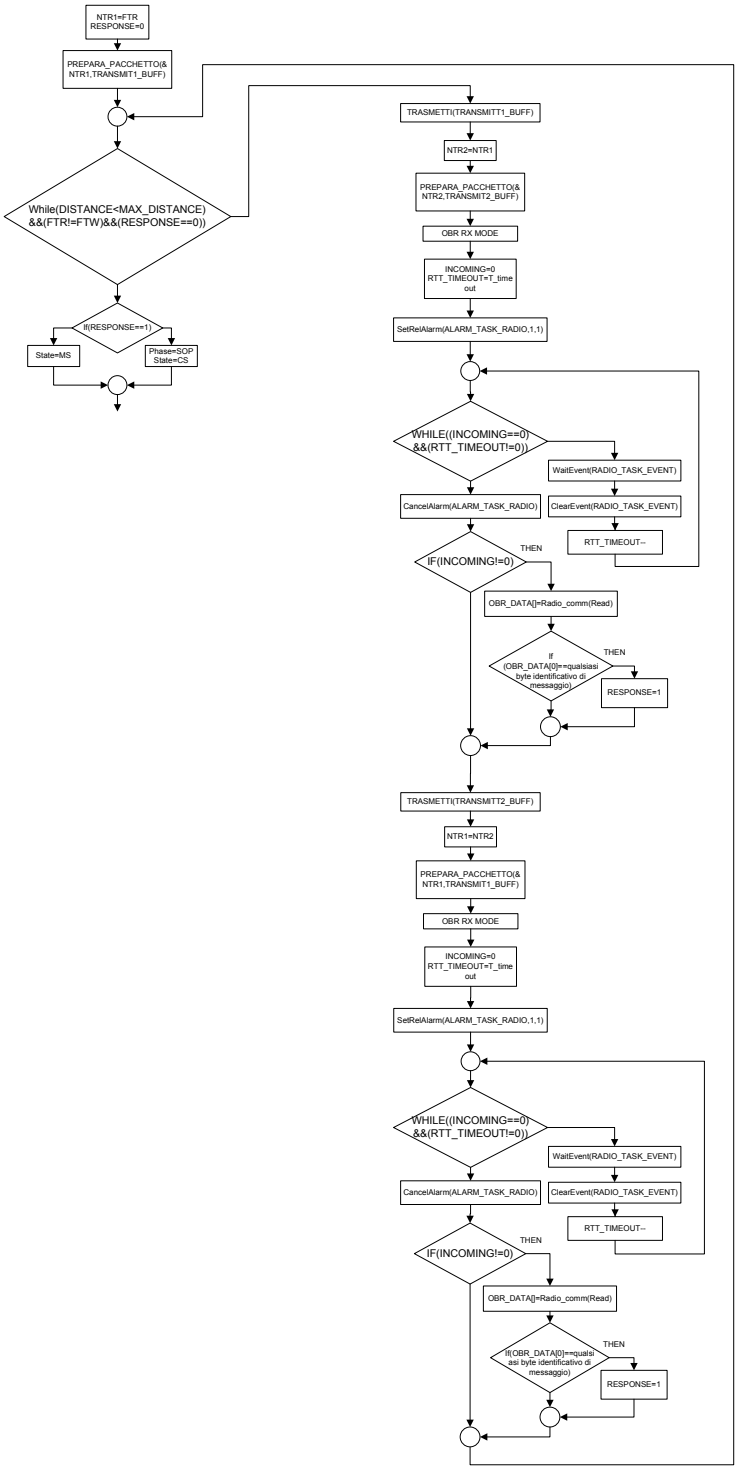


Figura 6.20: Diagramma di flusso delle operazioni eseguite dal task radio nello stato UOP *Communication State*

di priorità, per evitare che gli altri task passino in stato di 'RUNNING' durante il periodo di carica della batteria. Le risorse condivise utilizzate dal sistema operativo sono: la porta SPI, la memoria SRAM e la porta USART, le quali hanno un livello di priorità rispettivamente di 13, 12 e 11.

I motivi per i quali il task radio ha un livello di priorità 10, maggiore di quello del task measure (priorità 9), sono:

- AtmoCube comunica con la GS in intervalli di tempo molto ristretti e quindi le temporizzazioni per le comunicazioni tra le due parti non devono subire ritardi;
- durante le trasmissioni, il task measure non ha richieste di tempo stringenti per eseguire le sue operazioni;
- durante le misurazioni scientifiche, il task radio è per la maggior parte di tempo in stato di 'WAITING', permettendo al task measure di operare senza ritardi.

Quando gli altri task non stanno eseguendo nessuna operazione, il task system deve controllare se essi stanno operando nel modo corretto e per questo motivo ha un livello di priorità 7, che è inferiore ai task precedentemente menzionati. Infine, il task watchdog ha la priorità 1, poiché si tratta di un task che rimane attivo in sottofondo al sistema e viene comunque preso in considerazione dal Sistema Operativo; il tick di sistema provvede ad aggiornare periodicamente il watchdog timer, evitando di porre il microcontrollore in uno stato di reset. La condizione di reset si verifica soltanto se il software si blocca da qualche parte del codice e non permette al tick di intervenire su questo contatore.

La Tabella 6.1 riporta i livello di priorità delle entità software (task e risorse) del Sistema Operativo di AtmoCube.

Tabella 6.1: Livelli di priorità dei task e delle risorse del Sistema Operativo di AtmoCube

Entità software	Livello di priorità	Stato iniziale all'avvio del Sistema Operativo
task power	15	READY
task idle	14	SUSPENDED
porta di comunicazione SPI	13	–
memoria SRAM	12	–
porta di comunicazione USART	11	–
task radio	10	SUSPENDED
task measure	9	SUSPENDED
task system	7	READY
task watchdog	1	READY

6.9 L'ambiente di sviluppo del codice

Per realizzare il codice Sistema Operativo di AtmoCube, è stato utilizzato l'ambiente di sviluppo MPLAB–IDE[13] di proprietà della Microchip, che è in grado di interfacciarsi con il programmatore *in–circuit* MPLAB ICD2 per dispositivi di tipo flash tramite il quale è possibile interfacciarsi direttamente al microcontrollore PIC 18LF8722 della OBDH. In questo modo è possibile esaminare in tempo reale il codice realizzato, facendolo eseguire direttamente sul microcontrollore ed analizzare passo–passo, le istruzioni eseguite e le variabili di sistema tramite l'utilizzo di *breakpoint*.

Il codice è stato scritto in linguaggio C e, per poter essere eseguito sul microcontrollore i file creati devono essere compilati e poi *linked* per ottenere un file finale di tipo HEX che programma il microcontrollore. La catena di compilazione Microchip è composta da tre elementi:

- l'assemblatore MPASM che permette di trasformare un file ASM in un file O;
- il compilatore MCC18 che permette di trasformare un file C in un file O;
- il linker MPLINK che permette di racchiudere tutti i file O in un unico file HEX.

MPLAB è inoltre dotato di alcuni strumenti utili all'operazione di controllo del codice come ad esempio lo *stopwatch*, che permette di misurare il tempo di esecuzione tra un'istruzione e l'altra e il *watch*, che permette di osservare il valore assunto dalle variabili di sistema durante l'esecuzione del codice.

Capitolo 7

Risultati ottenuti

Durante il lavoro di Tesi è stato realizzato il codice per il Sistema Operativo di AtmoCube ed in questo capitolo vengono illustrati i risultati ottenuti dalle scelte implementative e dal codice scritto.

7.1 Calcolo del *throughput* con il sistema di trasmissione *Stop and Wait*

In questa sezione viene calcolato il *throughput* che si ottiene attraverso la trasmissione dei pacchetti di telemetria tramite la tecnica ARQ di tipo *Stop and Wait* descritta nel capitolo 5.2. La trasmissione dei pacchetti di telemetria può essere vista come la trasmissione di tre flussi informativi di throughput diverso, uno per ogni tipo di pacchetto di telemetria.

I parametri PPS_TEL1, PPS_TEL2 e PPS_TEL3 determinano le frequenza di salvataggio dei pacchetti e quindi determinano la percentuale di flussi informativi trasmessi. Infatti, in base ai valori predefiniti dei parametri, si ha che:

- i pacchetti di telemetria 1 vengono acquisiti 1 volta ogni 10 secondi;
- i pacchetti di telemetria 2 vengono acquisiti 1 volta ogni 60 secondi;
- i pacchetti di telemetria 3 vengono acquisiti 1 volta ogni 10 minuti.

In base a tali intervalli di tempo, in 10 minuti di misurazioni si ha che i pacchetti salvati sono:

- 60 di telemetria 1;
- 10 di telemetria 2;

- 1 di telemetria 3.

Quindi le percentuali dei pacchetti trasmessi sono:

- per l'85% di telemetria 1;
- per il 14% di telemetria 2;
- per l'1% di telemetria 3.

Nei capitoli seguenti viene calcolato il throughput per ogni singolo flusso informativo e viene poi calcolato il thrughput medio ottenuto tramite una media ponderata dei singoli throughput.

I calcoli sono stati eseguiti in base ai dati di probabilità di errore dei bit trasmessi riportati nel documento di link–budget[5]. Si assume quindi che:

- la BER in *down–link* è pari a 10^{-6};
- la BER in *up–link* è pari a 10^{-10}.

7.1.1 Calcolo del tempo medio necessario per la trasmissione dei pacchetti di telemetria

La trasmissione di un pacchetto di telemetria avviene attraverso la tecnica ARQ *Stop and Wait*, ovvero AtmoCube trasmette un pacchetto di telemetria ed aspetta che arrivi il messaggio di '*ackowledgement*' dalla GS. Se tale messaggio arriva entro il tempo $T_{timeout}$, il tempo totale per la trasmissione di un pacchetto è T_T(capitolo 5.2). Se, invece, la trasmissione non è andata a buon fine, il tempo necessario totale per la ritrasmissione di un pacchetto T_{err} è dato dalla somma del tempo di trasmissione del pacchetto e del tempo $T_{timeout}$. Quindi, supponendo che per la trasmissione di un pacchetto siano state necessarie i ritrasmissioni, il tempo totale per la trasmissione di tale pacchetto T_i può essere calcolato come:

$$T_i = T_T + (i-1)T_{err} \text{ per } i \geq 1$$

I calcoli finora effettuati sono stati fatti con i valori di BER e ritardo di propagazione peggiori che si presentano all'inizio ed alla fine della finestra trasmissiva. Difatti, all'inizio della finestra trasmissiva, il satellite si trova alla distanza massima ed all'elevazione minima dalla GS. Durante la finestra trasmissiva, il satellite si avvicina alla GS aumentando così il grado di elevazione e migliorando quindi le condizioni per la trasmissione, le quali vanno poi a peggiorare quando il satellite si allontana dalla GS, fino ad arrivare di nuovo a quelle iniziali. Dato che la probabilità di corretta ricezione di un frame in

down–link nel caso peggiore ha un valore tale da dire che quasi sicuramente ogni frame viene ricevuto correttamente, il modello matematico utilizzato per il calcolo del tempo di trasmissione medio viene semplificato mantenendo costante il tempo di ritrasmissione per ogni i–esima ritrasmissione ottenendo così che $T_i = i \cdot T_T$ e considerando illimitato il numero di ritrasmissioni per ogni frame. Si ha quindi il calcolo della media di una variabile aleatoria geometrica.

$$T_{med} = \sum_{i=1}^{\infty} T_i \cdot \mathrm{PS}_i \tag{7.1}$$

dove ${\mathrm{P_S}}^i = \mathrm{P_{CF}} \cdot (1 - \mathrm{P_{CF}})^{i-1}$ è la probabilità che la trasmissione i–esima vada a buon fine.

Quindi, l'equazione 7.1, svolgendo i calcoli, diventa:

$$\begin{aligned} T_{med} &= \sum_{i=1}^{\infty} T_i \cdot {\mathrm{P_S}}^i \\ &= \sum_{i=1}^{\infty} T_T \cdot i \cdot (1 - \mathrm{P_{CF}})^{(i-1)} \cdot \mathrm{P_{CF}} \\ &= T_T \cdot \sum_{i=1}^{\infty} i \cdot (1 - \mathrm{P_{CF}})^{(i-1)} \cdot \mathrm{P_{CF}} \\ &= T_T \cdot \frac{1}{\mathrm{P_{CF}}} \end{aligned} \tag{7.2}$$

7.1.2 Calcolo del *throughput* per i pacchetti di telemetria 1

I pacchetti di telemetria 1, come detto nel capitolo 5.1.1, vengono trasmessi 2 alla volta all'interno di un singolo *frame* e AtmoCube attende l'arrivo di un solo messaggio di '*acknowledgement*' per la conferma di entrambi. Bisogna quindi precisare che il numero dei *frame* destinati all'invio di pacchetti di telemetria 1 si dimezza e quindi le percentuali di frame inviati in base ai pacchetti di telemetria inviati, in realtà sono:

- il 73% per i pacchetti di telemetria 1;
- il 24% per i pacchetti di telemetria 2;
- il 3% per i pacchetti di telemetria 3.

Il tempo necessario per la trasmissione di un pacchetto di telemetria 1 (vedi capitolo 5.2.1 è $T_{T_1} = 297$ ms nel caso in cui la trasmissione vada a buon fine.

In base a quanto riportato nel capitolo 2.2.2 si ha che la lunghezza del frame in *down–link* è pari a 2120 bit e quella del frame in *up–link* è pari a 529 bit.

Si calcola ora la probabilità di corretta ricezione di un *frame*, come la probabilità che tutti i bit del singolo *frame* vengano ricevuti correttamente:

$$\mathrm{PCF} = (1 - \mathrm{BER})^{nb} \tag{7.3}$$

dove nb è la lunghezza del frame. Si ha quindi che:

$$\mathrm{P_{CF}}_{dl} = (1 - \mathrm{BER}_{dl})^{nb_{dl}} = (1 - 10^{-6})^{2120} \approx 0,99788 \tag{7.4}$$

$$\mathrm{P_{CF}}_{ul} = (1 - \mathrm{BER}_{ul})^{nb_{ul}} = (1 - 10^{-10})^{529} \approx 0,99999 \tag{7.5}$$

Dai risultati ottenuti nell'equazione 7.5 si possono approssimare i calcoli considerando che il *frame* in *up–link* venga ricevuto sempre in modo corretto e pertanto, si può porre $\mathrm{P_{CF}} = \mathrm{P_{CF}}_{dl}$.

Dall'equazione 7.2 si ricava il tempo medio necessario alla trasmissione di 2 pacchetti di telemetria 1:

$$\begin{aligned} T_{med_1} &= T_{T_1} \cdot \frac{1}{\mathrm{P_{CF}}} = \\ &= \frac{297}{0,99788} \approx 298ms \end{aligned} \tag{7.6}$$

Per il calcolo del *throughput* medio di telemetria 1 si effettua la divisione tra la quantità di dati di informazione inviati ed il tempo medio necessario e si ha quindi:

$$R_{thr_1} = \frac{2 \cdot L_{Tel1}}{T_{med_1}} = \frac{2 \cdot 126 \cdot 8}{298} \approx 6,77Kbit/s \tag{7.7}$$

7.1.3 Calcolo del *throughput* per i pacchetti di telemetria 2

I pacchetti di telemetria 2, come detto nel capitolo 5.1.2, vengono inviati trasmettendo 3 *frame* consecutivi e quindi la probabilità che tali frame siano ricevuti correttamente è data da $\mathrm{P_{CF}}^3$.

Il tempo necessario per la trasmissione di un pacchetto di telemetria 2 è $T_{T_2} = 721$ ms nel caso in cui la trasmissione vada a buon fine (vedi capitolo 5.2.2).

Dall'equazione 7.2 si ricava il tempo medio necessario alla trasmissione di 1 pacchetto di telemetria 2:

$$T_{med_2} = T_{T_2} \cdot \frac{1}{\mathrm{P_{CF}}^3} = \\ = \frac{721}{0,99788^3} \approx 726ms \tag{7.8}$$

Per il calcolo del *throughput* medio di telemetria 2 si effettua la divisione tra la quantità di dati di informazione inviati e il tempo medio necessario e si ha quindi:

$$R_{thr_2} = \frac{L_{Tel2}}{T_{med_2}} = \frac{545 \cdot 8}{726} \approx 6,00Kbit/s \tag{7.9}$$

7.1.4 Calcolo del *throughput* per i pacchetti di telemetria 3

I pacchetti di telemetria 3, come avviene per quelli di telemetria 1, vengono trasmessi attraverso un singolo *frame* e quindi i tempi di ritrasmissione dei pacchetti di telemetria 3 hanno lo stesso valore di quelli per i pacchetti di telemetria 1 e la stessa probabilità PCF. Quindi, anche il tempo medio di trasmissione di un pacchetto di telemetria 3 assume lo stesso valore di quello per la trasmissione di 2 pacchetti di telemetria 1: $T_{med_3} = T_{med_1} = 298$ ms.

Per il calcolo del *throughput* medio di telemetria 3 si effettua la divisione tra la quantità di dati di informazione inviati e il tempo medio necessario e si ha quindi:

$$R_{thr_3} = \frac{L_{Tel3}}{T_{med_3}} = \frac{162 \cdot 8}{298} \approx 4,35Kbit/s \tag{7.10}$$

7.1.5 Calcolo del throughput medio totale

Si esegue ora il calcolo del throughput medio totale eseguendo la media ponderata dei throughput medi per ogni pacchetto di telemetria in base alle percentuali di trasmissione dei diversi tipi di pacchetto. Si ha quindi che il throughput medio totale è dato da:

$$R_{thr_T} = 0,73 \cdot R_{thr_1} + 0,24 \cdot R_{thr_2} + 0,03 \cdot R_{thr_3} \approx 6,51Kbit/s \tag{7.11}$$

7.2 L'intervallo di tempo tra il dato GPS e il dato del magnetometro

Il tempo che intercorre tra l'acquisizione del dato PVT del GPS e il campionamento del magnetometro, come detto nel capitolo 2.5.2, è di grande interesse per l'analisi dei dati scientifici poiché determina l'accuratezza delle misurazioni fatte dal magnetometro.

Nel capitolo 4.2 è indicato che il tempo massimo tra le due misurazioni non deve superare 1 ms.

Il software di sviluppo MPLAB–IDE offre uno strumento, chiamato '*StopWatch*', che permette di misurare il tempo tra due istruzioni di codice. Tramite tale strumento, è stato possibile misurare quanto tempo passa tra l'arrivo dell'*interrupt*, provocato dalla linea PPS del modulo SGR, e l'istruzione del codice che ordina al magnetometro di effettuare la conversione ADC.

Per effettuare la misurazione è stato posizionato un *BreakPoint* nel punto in cui ha inizio la porzione di codice relativa all'ISR ed un altro nel punto in cui la misura del magnetometro è stata eseguita. Quando arriva l'interrupt INT0, causato dalla linea PPS, l'esecuzione del programma si blocca nella linea indicata dal primo *BreakPoint* ed in quel momento è stato dato il comando di azzeramento al cronometro del *StopWatch*. In seguito viene fatta continuare l'esecuzione del codice fino al punto indicato dal secondo *BreakPoint* ed il valore mostrato dal cronometro dello *StopWatch* indica quindi il tempo trascorso tra i due punti indicati dai *BreakPoint*. A tale valore si deve sommare il tempo di indeterminazione caratteristico di PICOS18[3] che corrisponde a 50μs e, quindi, il tempo tra l'arrivo del segnale di interrupt, che indica l'avvenuta ricezione di un nuovo dato PVT e la misura del magnetometro, è pari a circa 391μs.

Capitolo 8

Conclusioni e sviluppi futuri

In questo capitolo vengono indicati i risultati ottenuti nel lavoro di Tesi e le prossime tappe per lo sviluppo del Sistema Operativo di AtmoCube.

8.1 Conclusioni

In questa Tesi è stato illustrato il progetto di un sistema completo per la gestione del satellite AtmoCube capace di gestire e coordinare gli strumenti scientifici, di salvare i dati ottenuti e di trasmetterli alla GS utilizzando il nucleo del Sistema Operativo PICOS18 per microcontrollori della famiglia PIC 18.

Sono stati progettati e realizzati, tramite l'ambiente di sviluppo MPLAB–IDE, tutti i task necessari per il corretto funzionamento del Sistema Operativo. In particolare è stato progettato un metodo per sincronizzare la misura della posizione del satellite, effettuata dal modulo GPS con la misura del campo magnetico, permettendo così di limitare l'indeterminazione sulla posizione spazio–temporale del satellite (capitolo 4.2).

È stata poi implementata una gestione della memoria SRAM di tipo circolare che permette di salvare le informazioni, sotto forma di pacchetti di Telemetria, sovrascrivendo quelli meno recenti. Tali pacchetti vengono salvati codificati con il codice di correzione dell'errore esteso di Hamming (vedi capitolo 4.3) che permette di ridurre notevolmente la probabilità che un evento SEU possa modificare qualche bit salvato. Il sistema è inoltre capace di modificare le frequenze di acquisizione e salvataggio dei pacchetti secondo le indicazioni fornitegli dalla stazione di Terra, permettendo così una gestione dinamica delle misurazioni scientifiche.

L'invio dei pacchetti avviene attraverso una trasmissione di tipo ARQ *Stop and Wait* (capitolo 5.2), capace di modificare le sue caratteristiche in base al tipo di pacchetto di telemetria inviato per migliorare l'efficienza della trasmissione. I calcoli del *throughput*

descritti nel capitolo 7.1) permettono di affermare che, in una finestra di accesso di durata media, il satellite è in grado di trasferire alla GS tutti i dati contenuti nella memoria.

Il sistema realizzato è in grado di ricevere ed eseguire i comandi forniti dalla stazione di Terra (GS), permettendo così di operare in diverse modalità (ad esempio iniziale, diagnostica, normale o d'emergenza). Inoltre, nel caso in cui la comunicazione dalla GS verso il satellite, fosse, per qualche motivo impossibile, il sistema sarebbe in grado di operare autonomanente secondo gli obiettivi imposti dalla missione.

La scelta di utilizzare un nucleo di un Sistema Operativo *multitasking real–time*, come PICOS18, ha permesso di sviluppare applicazioni e task, su un *hardware* costituito da tecnologia per uso commerciale per applicazioni terrestri e con scarse risorse energetiche, creato appositamente per gestire gli strumenti scientifici e gli altri apparati del satellite. Il Sistema Operativo descritto in questa Tesi è infatti, capace di sfruttare le limitate risorse di calcolo fornite dal microcontrollore della scheda OBDH facendo uso delle risorse energetiche solo quando è strettamente necessario ed è capace di sospendere le operazioni nel caso in cui la carica della batteria risulti critica. È stato spiegato inoltre come il Sistema Operativo realizzato sia in grado di coordinare le misurazioni scientifiche in modo che vengano eseguite nel modo corretto e che i dati ricavati siano il più possibile esenti da errori dovuti ad errate temporizzazioni delle misurazioni e ad *offset*.

Si può infine affermare che il lavoro di Tesi abbia soddisfatto gli obiettivi prefissati, sia relativamente agli scopi scientifici sia a quelli tecnologici

8.2 Sviluppi futuri

Il codice per il Sistema Operativo è stato realizzato attraverso il sistema di sviluppo MPLAB–IDE e simulato attraverso lo strumento MPLAB–SIM che simula l'esecuzione del codice come avverrebbe nel microcontrollore. Per completare lo sviluppo del Sistema Operativo, si dovrà effettuare il processo di validazione del codice scritto per verificare la corretta gestione delle perifiche da parte del microcontrollore, in particolare, quando l'*hardware* di tali periferiche sarà completo, bisognerà verificare la corretta gestione delle linee di comando e di comunicazione con il GPS, lo spettro–dosimetro ed il microcontrollore adibito all'operazione di *Attitude Determination Control*. In seguito, quando verrà completato il progetto della GS si dovrà verificare il corretto funzionamento del satellite per quanto riguarda la gestione e l'esecuzione dei comandi forniti da remoto.

Bibliografia

[1] OSEK Group. *OSEK/VDX Operating System Specification 2.1r1*, 2000.

[2] Stefano Punis. Progetto hardware e software del sistema di controllo del satellite atmocube. Master's thesis, Università degli Studi di Trieste, 2007–2008.

[3] Pragmatech. *PICOS18 Real Time Kernel for PIC18*, 2006.

[4] AtmoCube Development Team. Atmocube proposals.

[5] Alessandro Cuttin. Progetto di sistema del satellite atmocube architettura di comunicazione e architettura di sistema. Master's thesis, Università degli Studi di Trieste, 2005–2006.

[6] ESA. Cubesat design specification. Technical report, ESA, 2009.

[7] *Space Radiation Effects on Electronic Components in Low-Earth Orbit, lesson Number 0824* . http://www.nasa.gov/offices/oce/llis/0824.html.

[8] Sgr-05u gps receiver. User Interface Manual and Interface Control Document.

[9] Hi-Tech. *Salvo User Guide*. http://www.pumpkininc.com/content/doc/manual/.

[10] J. Labrosse. μc/os–ii *The Real Time Kernel*, 2002.

[11] Giuseppe O. Longo. *Teoria dell'Informazione*. Boringhieri, 1980.

[12] MICROCHIP. Pic18f8722 family data sheet. Technical report, Microchip Tecnology Inc., 2004.

[13] MICROCHIP. Mplab ide user's guide. Technical report, Microchip Tecnology Inc., 2006. http://ww1.microchip.com/downloads/en/DeviceDoc/51519B.pdf.

Capitolo 9

Ringraziamenti

Desidero inanzitutto ringraziare il Dr. Livio Tenze per i preziosi consigli che mi ha dato durante lo sviluppo del codice, per i pranzi in Area di Ricerca e sopratutto per le correzioni di questo testo. Ringrazio Mywave per avermi permesso di svolgere il tirocinio e la Tesi e ringrazio l'Ing. Mario Fragiacomo per la sua disponibilità e gli utilissimi consigli. Ringrazio il Professor Fulvio Babich per il tempo dedicatomi e per i preziosi consigli per migliorare questo lavoro di Tesi.

Questo testo rappresenta per me la conclusione di un capitolo molto importante della mia vita e sono molte le persone che in questo periodo di studi universitari hanno contribuito, sia per quanto riguarda un aiuto nello studio, sia per un sostegno morale, a farmi raggiungere questo risultato. Ringrazio quindi gli zii, che hanno sempre fatto il tifo per me. Ringrazio mia sorella Marina per il tempo passato assieme, per le numerose chiacchierate e per il piccolo Thomas che riesce sempre a darmi una gran felicità ogni volta che sto assieme a lui. Ringrazio nonna Marina per tutti i pranzi che mi ha fatto e perché, ogni anno, nel mese di giugno, mi chiedeva se sarei stato promosso al prossimo anno universitario (anche se in Università il concetto di promozione all'anno successivo non è molto corretto), dimostrandomi sempre il suo affetto e l'interesse in quello che stavo facendo. Ai miei genitori, Giorgio ed Isabella, ai quali ho dedicato questa Tesi, va il mio più caloroso abbraccio per ringraziarli dell'affetto, dei sacrifici e della pazienza che mi hanno dato e colgo l'occasione per dire loro che forse è arrivato il momento di ascoltare e discutere con calma e moderazione le mie idee ed accettare le mie decisioni che saranno prese sempre tenendo conto di tutti i loro insegnamenti. Ringrazio Sara per tutti i bei momenti passati assieme, per l'infinita pazienza dimostrata durante le mie fasi 'orso Bubu' e per la sua capacità di farmi sorridere in ogni momento. Ringrazio i miei nonni acquisiti Romana e Giorgio per avermi trattato fin dal primo giorno come un loro nipote e per tutti i pranzi e cene offerte. Ringrazio Patrizia per avermi ospitato in casa sua ed aver sopportato le bacinelle spostate, le prese collegate alla spina e per il fatto che ogni tanto facevo finta di ascoltarla.

Ringrazio Marco, il mio amico d'infanzia, per avermi sempre aiutato nello studio quando avevo bisogno e per i bei momenti passati assieme fuori e dentro l'Università. Ringrazio Alessia Violin per i suoi preziosi consigli, per tutti gli appunti imprestati e per essere sempre stata una gentile amica; spero un giorno di poterle ricambiare tutti i favori che mi ha fatto.

Saluto poi gli amici a cui mi sento più legato, che in questi anni hanno condiviso con me numerosi momenti felici di svago e spensieratezza: Enrico, Sara, Sante, Eleonora, Anna, Antonio, Giovanni, Matteo, Marco A., Peter, Alessandro (Sa).

Un grazie particolare ad Alessandro per...

- essere stato il miglior compagno di banco che abbia mai avuto;
- essere sempre stato disponibile ad ogni ora del giorno a spiegarmi e rispiegarmi i concetti che non mi entravano in testa;
- avermi consigliato sempre su come studiare e approfondire gli argomenti degli esami;
- i consigli sulla Tesi;
- tutti gli appunti che mi ha imprestato.

Spero un giorno di poter ricambiare, almeno in parte, tutto quello che mi hai dato.