



Sapienza University of Rome
Master's degree in Space and Astronautical Engineering

Spacecraft Design 2020/21

Prof. Fabio Santoni

Prof. Assistant Paolo Marzioli

FINAL REPORT ON THE OBDH SUBSYSTEM FOR
THE CUBESAT HYONOSAT

Team members

Srikanth Cheruku

Alessandro Di Muzio

Jacopo Ginobbi

Ashish Kallikkattil Kuruvila

Anantha Krishnan Orunnukaran Mani

SUMMARY

1. Introduction.....	5
1.1 Mission Requirement Analysis	
1.2 Modeling of the OBC and Software	
1.3 Literature Study	
1.3.1 OBC and Interface Board	
1.3.2 Software	
1.3.3 Subsystem Components and Power Systems	
2. Market analysis.....	9
2.1 OBDH subsystem	
2.2 Trade-off	
2.2.1 On board computer	
2.2.2 Bus	
2.2.3 Mass memory	
2.2.4 Flight software	
3. Requirements.....	15
3.1 Design requirements	
3.2 Interface requirements	
3.3 Functional requirements	
3.4 Operational requirements	
3.5 Mission requirements	
3.6 Environmental requirements	
4. Hardware.....	19
4.1 On board computer	
4.2 FLASH Memory	
4.3 OBDH Motherboard	
4.4 Interface board	
5. Architecture.....	27
5.1 Architecture models	
5.2 Actual architecture	
6. Operation.....	31
6.1 Definition of the CubeSat operational modes	
6.2 Transition between the different modes	
6.3 Launch mode	
6.4 Safe mode	
6.5 FDIR	
6.6 Detumbling mode	
6.7 Nominal mode	
6.8 Primary payload strategy	

7. Software.....	45
7.1 Operating System	
7.1.1 FreeRTOS Initialization	
7.1.2 Task Implementation	
7.1.3 Choosing Memory Scheme	
7.1.4 Configuring FreeRTOS	
7.2 Flight Software	
8. Protocols.....	51
8.1 The OSI model	
8.2 The actual interfaces	
8.2.1 CSP	
8.2.2 CAN	
8.2.3 I2C	
8.2.4 KISS	
8.2.5 RS-422/RS-485	
8.2.6 UART (TTL)	
9. Memory budget.....	57
9.1 Memories	
9.2 Data budget	
10. Processing power estimation.....	61
10.1 Model	
10.2 Central OBC	
10.3 AODCS Dedicated OBC	
10.4 Insights	
11. References.....	67

Chapter 1

Introduction

The On-Board Data Handling system is the brain of the CubeSat, and it plays a vital role in the HyonoSat mission. The HyonoSat Mission is a 6U CubeSat mission which will obtain precise hyperspectral information regarding both terrestrial and marine resources, and atmospheric pollution. The CubeSat focuses its attention on Europe.

The On-Board Computer of the satellite acts as the primary source of all the commands and monitors the status of the various subsystems. The satellite follows an amalgamation of centralized and distributed architecture. The collected data is used for maintaining an information log which is used as input to algorithms that aid in pointing and detumbling the satellite. This data is also downlinked by the telemetry to the ground station. OBDH consist of several parts, i.e., one main microcontroller NanoMind A3200 32-bit High-performance AVR32 MCU, an Interface board NanoDock DMC-3, and several housekeeping sensors.

The works on the On-Board Data Handling for the HyonoSat Mission have been classified into 3 Main Subcategories which include the mission requirement analysis, the design and the modelling of the OBC and the software.

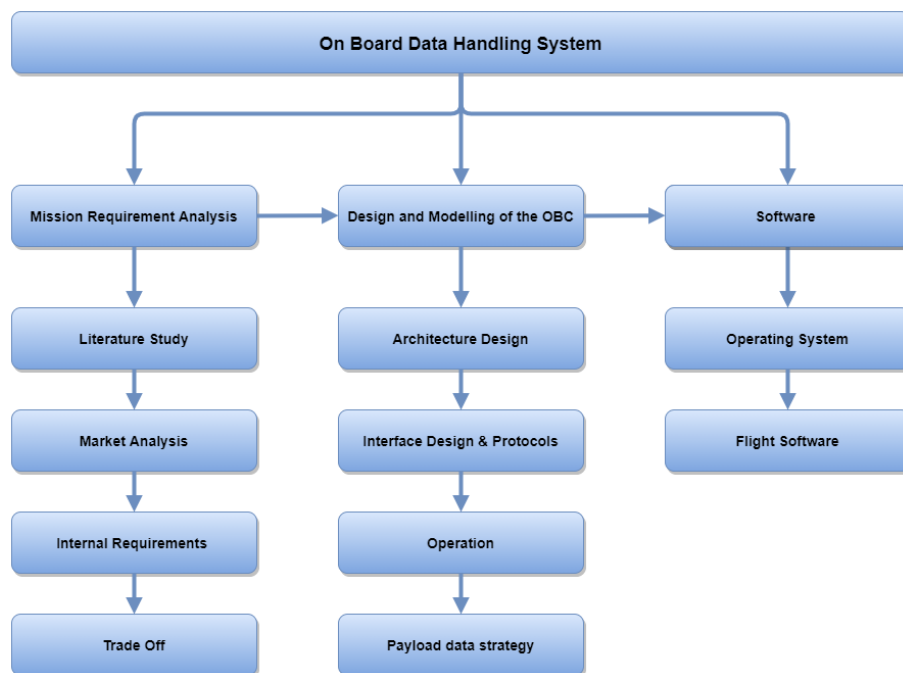


Fig 1.1 – Workflow chart

1.1 Mission Requirement Analysis

Once the CubeSat mission has been finalized the next objective is, to begin with, the Subsystem work, In the case of the OBDH subsystem the initial objective is to conduct a Mission requirement analysis.

The primary objective is a Literature Study, by taking advantage of the published journals and books a deep study about the On-Board Data Handling systems are conducted. A proper definition and functions of OBDH have been sorted out. This enabled the whole project to uphold the standard and the mission requirement throughout.

The Market Analysis is the task soon after the literature study, a set of components which are required to fulfill the mission objective and for the housekeeping of the CubeSat have been listed out and a thorough comparative study on the capabilities and functionality of the component has been conducted. The Market analysis made it possible to make the internal standard requirements which are to be followed throughout.

The Internal Requirements are the basic standards that should be strictly followed in the whole mission process and while considering the design and modeling process. Finally, from the market analysis considering the requirements, the component trade-off has been done.

1.2 Modeling of the OBC and Software

The OBC was designed to provide a platform for the control and data handling programs to interface with the subsystem components. The Primary objective in OBC design and modeling is Architecture Design, which has been divided into two, Hardware architecture and Software architecture. The hardware architecture deals with the Microcontroller architecture considering the interface requirements and, in our case, an interface board has been provided for the smooth running of the On-Board Computer. The Software architecture defined the platform of the On-Board Computer NanoMind A3200 and the program which is used to control and conduct the data handling from each subsystem component. We have decided to use a Real-Time Operating system i.e., FreeRTOS as the Operating system for the Onboard Computer and the brain which controls the whole satellite has been chosen as Core Flight Systems (cFS) by NASA.

Once the Architecture of the CubeSat is designed the next goal is to enable the communication between the subsystems and the OBC, which is made possible by choosing suitable Interface network protocols. This is the most important task in the Design and Modelling phase of the CubeSat.

Once the Interface modeling is done, then the Data/Memory Budget calculation for every single interface with the subsystem. This deep process is held before choosing the Main Memory drive and plays a vital role in deciding the storage drive for the whole Mission.

1.3 Literature Study

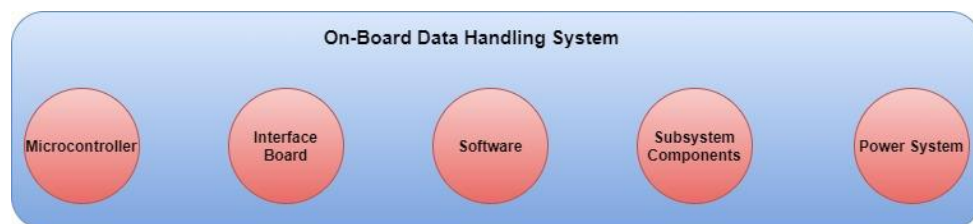


Fig 1.2 – General OBDH subsystem

1.3.1 OBC and Interface Board

An On-Board Computer usually contains a central processor, I/O ports, memory for program and data storage, an internal clock, analog-to-digital converter (ADC), serial communication facilities, and circuits.

- Functions of the On-Board Computer subsystem
- Determination of mode of operation and flow of control.
- Storing the payload data.
- Acquisition of the image and execution of the compression algorithm.
- Acquiring housekeeping data from other subsystems.
- Sending the data to the Telemetry for downlinking.
- Processing of the Telecommand Data.
- Executing attitude determination and control algorithms.

1.3.2 Software

The Software section of the OBC is classified into two, first one is the Platform which the OBC i.e., the Operating System and the other is the program which controls and handles the data i.e., the Flight Software.

It is important to know that the CubeSat is a “*time critical system*” thus a Real time Operating system will be in operational over a normal Operating System. An RTOS is necessary when there are several processes and devices, and the processes’ timing is more important than average performance. If you need multiple processes to run on a schedule, you need an RTOS. An RTOS can guarantee the timing requirements for processes under its control, has low latency, is predictable in that it can determine a task’s completion time with certainty, and enables the coexistence of both time-critical and non-time critical tasks. An RTOS can effectively handle interrupts based on priority to control scheduling. Unlike a general-purpose OS, an RTOS is expected to meet computational deadlines, regardless of how bad the scenario can get for the RTOS. With help of Software Market Analysis and comparative study we choose FreeRTOS as the Operating System for the OBC. The major benefits of choosing an FreeRTOS are Multitasking, Efficiency, Service, structure of the OS, Portability, and security.

Then the next objective is to decide the program which should accurately control and well as securely store the handling data. The SW market analysis also played a vital role in deciding the flight SW, and after going through the capabilities, the performance measures, and the telemetry data representations we chose the Core Flight System (cFS) developed by NASA. The core Flight System (cFS) is a platform and project independent reusable software framework and set of reusable software applications. There are three key aspects to the cFS architecture: a dynamic run-time environment, layered software, and a component-based design.

1.3.3 Subsystem Components and Power Systems

The main objective of the work was to decide among the whole subsystem components for which require a control action, or a data handling required. Thus, the components have been classified as housekeeping components and the telemetry available components. The housekeeping components will play the control actions of the CubeSat, mainly the AODCS

components, and the payload, TTC, and the EPS Motherboard will be the telemetry available components and the telemetry data are stored and send to ground stations. The data accumulated from the scientific payloads can be either transmitted in real time or stored, which will be transmitted to ground during the satellite visibility. The Power system components include the Battery and the Solar panels together, which act as the powerhouse for the whole CubeSat and to the OBC.

Chapter 2

Market analysis

2.1 OBDH subsystem

The on-board data handling (OBDH) subsystem of a spacecraft is the subsystem which carries and stores data between the various electronics units and the ground segment, via the TT&C subsystem.





The OBDH subsystem is divided in two different components hardware and software. The hardware component comprises all the physical electronic components that are located on a PCB. It is the lowest level in the OBC model and consist of a microcontroller and supporting hardware with a memory module to store programs and data. The software component commands the processor and controls its operability and functionality.

2.2 Trade-off

2.2.1 On board computer

The OBC is the computer of the satellite's avionic sub-system, tasked with controlling the spacecraft and maintaining its safety. Though there is scope to design in house development of there are plenty of open-source projects choices to select On Board Software and almost all the COTS OBC's provide their own On-Board Software. Based on market research there are plenty of OBCs available among which few promising boards are shortlisted at the trade-off section with their key specifications.

On board computers shortlisted for comparison are all compatible with minimum mission requirements and fortunately provide more features than required at this stage but can add value for the long term prospective. Since every shortlisted computer satisfies mission requirements comparison is done taking primary consideration of mass, power consumption, dimensions and development support.

Product Name	NanoMind A3200 + NanoDock DMC-3	Sirius OBC LEON3FT	NANOSATPRO	ABACUS 2017
Supplier	GomSpace (Denmark) 	AAC Clyde Space (Sweden) 	STM (Turkey) 	GAUSS (Italy) 
CPU	AVR32	32-bit LEON3FT processor	8051 Softcore / Leon3	MCU MSP430 + FPGA Spartan-3E

CPU Speed	8MHz – 64 MHz	50 MHz	48 MHz	up to 25 MHz (MSP430) and 25/100 MHz (FPGA)
RAM	32kB FRAM +32 MB SDRAM	64 MB + 16 kB	32 MB + 8MB	2 MB SRAM (FPGA)
Storage	512Kb (CPU) + 128 MB	2 GB	128 MB	2 x 16 MB
Interfaces	2x I2C, UART, CAN, 7x GPIO, USB	6x RS422/RS485 UART, 2x SpaceWire, 16x GPIO, 2x CAN	SPI, I2C, 2x CAN, 3x RS485 UART, 8x GPIO	34x GPIO, 8x GPI, 4x COM, 2x I2C, SPI
Power	0.9 ÷ 1.1 W	1.3 W	1.5 W	N/A
Operating Temperature	-30 ÷ 85 C	-30 ÷ 60 C	-40 ÷ 85 C	-40 ÷ 85 C
Size	65 x 40 x 7.1 mm (NanoMind) 91.9 x 88.7 x 8.6 mm (NanoDock)	95.89 x 90.17 x 17.20 mm	95.90 x 90.18 x 18.21 mm	90.14 x 95.86 x 23.24 mm
Mass	24 g (NanoMind) 51 g (NanoDock)	130 g	100 g	62 g

Table 2.1 – Comparison of OBCs

The GOMspace NanoMind A3200 is based on an Atmel AT32UC3C MCU. This is a high performance 32-bit RISC architecture with advanced power saving features to both facilitate tasks with a high computational demand and tasks where the MCU is idle most of the time. For applications as ADCS the MCU has floating point support that is based on IEEE 754 floating point standard.

Next, we have AAC CLYDE SPACE in the shortlist with 50 MHz LEON3FT fault-tolerant soft processor, its RTEMS real-time operating system (RTOS) is compliant to IEEE 1754 SPARCv8. Utilizing SpaceWire on-board the main data bus, the on-board computer is designed to emphasis high performance, resilience and reliability. Sirius Command and Data Handling system has a standard single string system that consists of an on-board computer (Sirius OBC) and a combined mass memory with CCSDS stack (Sirius TCM). The OBC runs mission specific software and manage the spacecraft system.

Next, we have Nano-Satellite Processor Unit (NANOSATPRO) is a high-performance on-board computer (OBC) that is compatible with nano-satellite platforms for advanced space

missions. It is a control unit that has high fault tolerance and processing power specifications for difficult space conditions. As a target, NANOSATPRO is designed to be operational in a low earth orbit (LEO) for at least two years. NANOSATPRO has an operating system running on an FPGA (soft processor-based) and supports the most commonly used interfaces (UART, RS485, CAN, SPI, I2C, etc.) in nanosatellite platforms. With its modular design, it easily adapts to the connection constraints of the target platform.

Finally, we have ABACUS 2017 It is designed to be flexible and scalable in terms of processing power, with the goal of maintaining a very low power consumption. The presence of a MSP430 (EP series) microcontroller and a Spartan-3E FPGA, organized in two independent but cooperative cores, provides the system with hardware redundancy and common mode fault tolerance. It is also embedded with 3 axis magnetometer, accelerometer, gyroscope and sensors: 3x temperature sensors, 1x drawn current monitor. ABACUS is compatible with FreeRTOS Real-time Operating System, as the MCU used in ABACUS has been already successfully employed in FreeRTOS ports.

Clearly, we can say that in this shortlist it's a tight call between GomSpace Nanomind A3200 and ABACUS 2017 but with small edge GomSpace Nanomind A3200 has smallest form factor and with weighs just 24g and wins if we consider mass and dimensions as primary factor. Also, it has less power consumption. The Nanomind A3200 in conjunction with the NanoDock DMC-3 is also compatible with FreeRTOS for simple and lightweight multitasking operation.

2.2.2 Bus

Conventional space borne controls are based on Centralized Control systems where a single central control unit is responsible to control the application tasks and performing data management, resulting often in relatively high-performance requirements for the CPU and high OBSW complexity. MIL-STD-1553 is used virtually in all ESA and European OBDH systems, and the considerable amount of experience and know-how built by European industries is enormous. Other low-medium speed data buses used are UART based on RS-422 physical layer and CAN. SpaceWire is a standard for high-speed links (up to 200 Mbps) and networks for use onboard spacecraft, developed by the European Space Agencies in collaboration with European Universities and Industries.

2.2.3 Mass Memory

In many missions it is necessary to temporarily store payload or spacecraft telemetry data on-board before transmission to ground. A typical example is the Earth Observation missions where it is often only possible to send data to ground for 10 minutes once every 1.5 hours. It is therefore essential to have a very robust and compact storage capability on-board. The mass-memories are being developed using various solid-state storage technologies such as dynamic RAMs and Flash memories.

Product Name	Memory (97D2H8G64)	Memory (97D2H2G16)	Memory (69F64G16)	Memory (28LV010)
Supplier	DCC (USA)	DCC (USA)	DCC (USA)	DCC (USA)

Type	DDR2 SDRAM	DDR2 SDRAM	FLASH	EEPROM
Non-Volatile	NO	NO	YES	YES
Op. Voltage	1.8 V	1.8 V	1.8 V	3.3 V
Density	8 Gb	2 Gb	64 Gb	1 Mb
Configuration	128M x 64	128M x 16	x16 NAND	128k x 8
Speed	400MHz	400MHz	50 MT/sec	200, 250 ns
Radiation	RAD Tolerant, RAD-PAK	RAD Tolerant, RAD-PAK	RAD Tolerant, RAD-PAK	RAD Tolerant, RAD-PAK

Table 2.2 – Comparison of mass memories

2.2.4 Flight software

Flight Software (FSW) is software that runs on a processor embedded in a spacecraft's avionics. It is responsible for managing on-board activities, data processing and spacecraft health and safety. Selection of on-board software is totally based on hardware compatibility choice of developer. We have plenty of software available in market both commercial and open source. Since developing and maintaining mission critical embedded software is difficult task and needs team of expert developer it is always preferred to choose the development framework, which is trusted, well documented and main tend provided committed support from development team. There are plenty of open-source CubeSat project mostly based on Free RTOS, among which NASA cFS is well documented and has great support from NASA.

Name	KubeOS	Core flight software (cFS)	PWSat2
OS Architecture	Linux	VxWorks, RTEMS, Linux, RTOS	RTOS
Compatible OBC	Pumpkin MBM2, ISISpace iOBC	Platform independent	ELS Cube Computer, PWSat2OBC 32bit ARM Cortex-M3
Link	https://www.kubos.com/	https://cfs.gsfc.nasa.gov/	https://github.com/PW-Sat2/PWSat2OBC
Developer	KubeOS(USA)	NASA Open Software (USA)	PW.Sat Team, Warsaw University (Poland)

Table 2.3 – Comparison of flight software

- KubeOS is an open-source, integrated platform designed to increase development speed, lower risk, and allow teams to focus on their payload. KubOS packages Linux, subsystem APIs, and core services into a single platform. But we do not stop there,

we provide intuitive developer tools, an SDK, documentation, and Service Level Agreement Support.

- The core Flight System (cFS) is a platform and project independent reusable software framework and set of reusable software applications. There are three key aspects to the cFS architecture: a dynamic run-time environment, layered software, and a component-based design. It is the combination of these key aspects that makes it suitable for reuse on any number of NASA flight projects and/or embedded software systems at a significant cost savings.
- PW-Sat2 is a student satellite project being built at the Warsaw University of Technology by members of the Student Astronautic Circle.

The Core Flight System (cFS) has been chosen as the mission's main software, which is a platform and project independent reusable software framework and set of reusable software applications. NASA cFS is well documented and has great support from NASA.

Chapter 3

Requirements

A set of requirements is used as inputs into the design stages of OBDH development. Requirements are also an important input into the verification process since tests should trace back to specific requirements. They also show what elements and functions are necessary for the OBDH project. To support the platform and mission operations, the OBDH system shall provide the following requirements.

The requirements are generally classified into 6 classes:

- DES - Design
- IF - Interface
- FKT - Functional
- OPS - Operational
- MIS - Mission
- ENV - Environmental

3.1 Design requirements

ID	REQUIREMENTS
1	The on-board computers shall fit the maximum dimensions of 100 x 100 x 20 mm each (included the motherboard).
2	The on-board computers shall have a mass under 250 g.
3.1	The single on-board computer shall consume less than 0.4 W for nominal operation of the mission.
3.2	The power requirements of the single on-board computer shall be always less than 1 W.
4	The on-board bus shall operate at 3.3-5 V.
5	The on-board storage capability shall be able to store all packets generated on-board for a duration at least equal to the longest non-coverage period.
6	On-board storage shall be either circular or linear.
7	On-board time shall be common to all spacecraft modes, including safe mode.

3.2 Interface requirements

ID	REQUIREMENTS
1	The on-board computer shall interface with selected sensors and actuators through the CAN bus.

2	The software shall distribute telecommands to all the satellite subsystems.
3	The software shall communicate with the chosen ground stations through the TT&C subsystems using telecommand and telemetry source packets specifically designed for the purpose.
4	The flight software shall read housekeeping telemetry from other subsystems according to the needs of those systems.

3.3 Functional requirements

ID	REQUIREMENTS
1	The OBC shall use UTC as its time reference frame.
2	The OBDH system shall have a watchdog timer to perform an onboard computer restart.
3	The software shall run error correction codes in case of SEUs.
4	Whenever a data bus error is detected, an event report shall be generated.
5	The event reports that are generated in the case of a failure shall indicate the type of failure, its location and any additional information needed for failure diagnosis.
6	The software shall receive and decode the telecommand messages from the ground.
7	On-board storage shall be able to store payload data until the spacecraft can relay it to the ground station.
8	The capability shall be provided to save the operational context in non-volatile memory so that it can be restored if a processor is reset or temporarily switched off.
9.1	The failure detection, isolation and recovery (FDIR) functions shall be implemented hierarchically to detect, isolate and recover failures at the lowest possible implementation level.
9.2	The FDIR functions shall make use of the redundancy onboard if said redundancy of the component is indeed implemented.
9.3	The FDIR activities performed on-board shall be reported unambiguously to the ground segment.
10	The capability shall be provided for the ground segment to load any telecommand into the on-board operations schedule.

3.4 Operational requirements

ID	REQUIREMENTS
1	The on-board computer shall have the capability to execute more than one task at the same time.
2	During safe mode, bandwidth allocation priority shall be given to essential commands and telemetry.
3	The space communication system shall ensure that data is not lost due to congestion.
4	The space communication system shall enable telecommand acknowledgements to be returned to the telecommand source.
5	All telemetry data generated onboard the spacecraft shall be time-stamped such that the temporal ordering of the acquired telemetry can be determined on the ground.

3.5 Mission requirements

ID	REQUIREMENTS
1	The software shall give priority to the payload data, when forced to do so.
2	The on-board computer shall be capable of sending the satellite health data to the ground station.
3	The spacecraft shall enter a safe state if any hazard exists that affects spacecraft or payload health or mission

3.6 Environmental requirements

ID	REQUIREMENTS
1	The OBDH hardware shall work nominally within the range from -30 to +80 C.
2	The OBDH hardware shall be tolerant/harden to at least 10^2 rad.

Chapter 4

Hardware

4.1 On board computer

The A3200 on-board computer is designed as an efficient system for space applications with limited resources, such as CubeSat or nano-satellite missions. The NanoMind A3200 contains also a 3-Axis magnetometer and coil-drivers that can be used to implement attitude control based on magnetic sensing and actuation and a 3-Axis gyroscope used for attitude control. Its main interface to other subsystems is CAN and I2C. For storage, the board carries a 128 MB NOR serial flash. The mass of the NanoMind A3200 is 24 g. The operational temperature is: -30 °C to +85 °C. NanoMind A3200 presents a RTC clock and its CPU clock rate averages around 32 MHz, with the maximum being 64 MHz. The size of NanoMind A3200 is 65 x 40 x 7.1 mm.



Fig 4.1 - Nanomind A3200 OBC

A3200 has two I2C buses supporting bidirectional data transfer between masters and slaves, multi-master bus, arbitration between simultaneously transmitting masters without corruption of serial data on the bus. One of the main interfaces of the A3200 to communicate with another subsystem hardware is a CAN bus interface. The Controller Area Network (CAN) is a serial communications protocol that supports distributed real-time control with a high level of security. The maximum bus speed is 1 Mbits/s. To sample external analog values the board supplies 8 ADC channels in one of the main connectors. These 8 pins can also be configured to be GPIO instead of ADC inputs. The Picoblade connector make it possible to establish the specific tasks.

- P1- USART
- P2- JTAG
- P6 - PWM output
- P7- I2C and VBAT

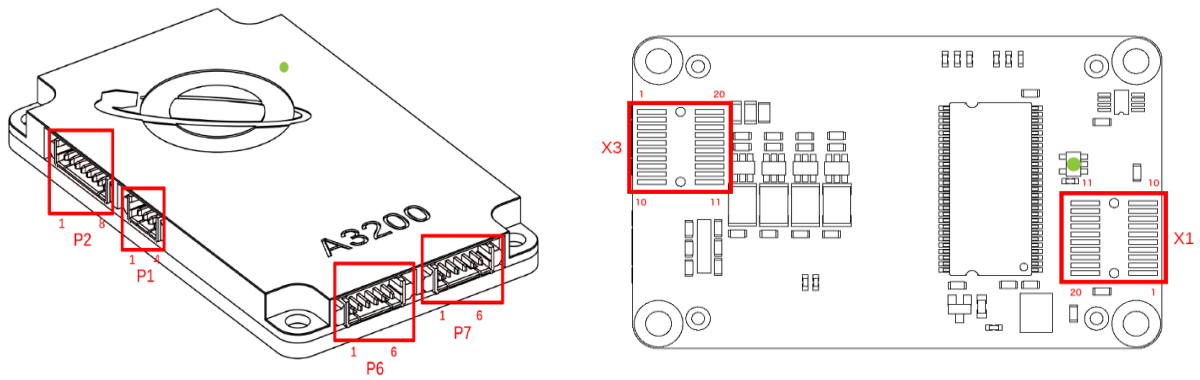


Fig 4.2 & 4.3 - Nanomind A3200 top and bottom view

The main connectors are X1 and X3 and are built into the PCB as two 20-position hard-gold plated FSI one-piece connectors. The motherboard connector is a SAMTEC-FSI-110-D.

X1	<ul style="list-style-type: none"> ○ 2x I2C ○ 4x GND ○ SCL ○ SDA ○ 2x CAN ○ 4x UART0
X3	<ul style="list-style-type: none"> ○ 2x I2C ○ 6x SPI

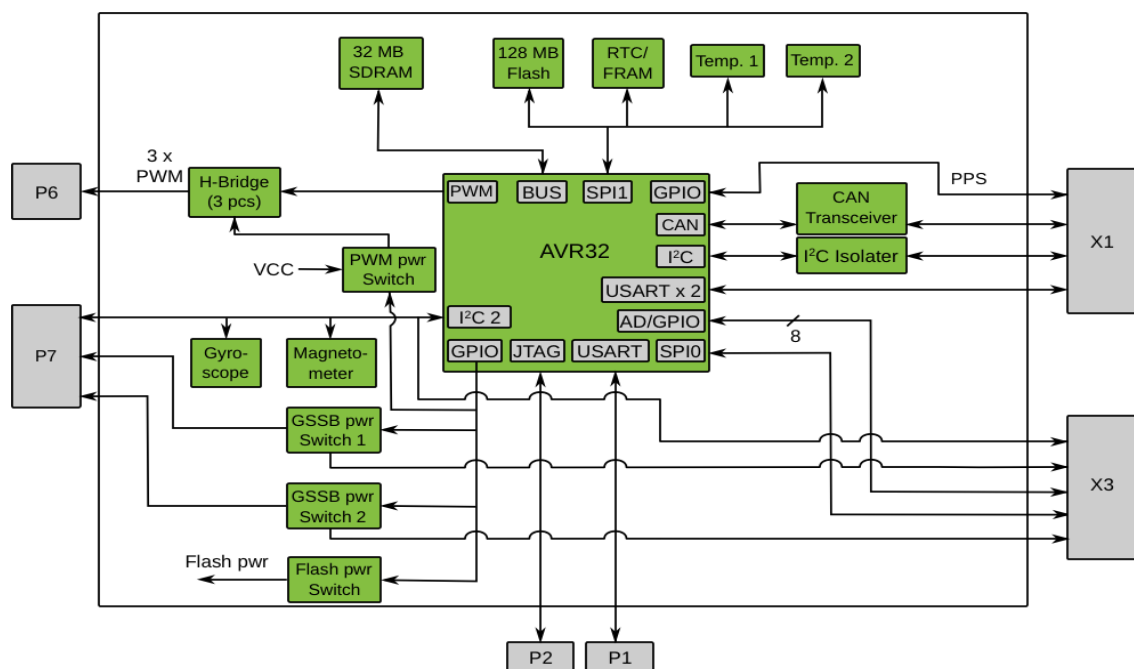


Fig 4.4 - Block diagram of the A3200 board

4.2 FLASH Memory

The memory 69F64G16 is used to store data and programs that are used by the computer. Memory can be classified in two main categories: volatile and non-volatile. Volatile memory is a type of memory that requires constant power in order to maintain the information stored within it. Non-volatile memory will retain its data even when no power is supplied to it. The selection of a non-volatile memory for a nanosatellite mission depends on several criteria: size, power consumption, price and durability. In space, access to the satellite may not be available at all times of day. Data that the satellite collects and produces must be stored in a memory location so that the satellite can access it at a later time, possibly to transmit it to a ground station. Flash memory is an electronic non-volatile memory storage medium that can be electrically erased and reprogrammed. The flash memory is the non-volatile portion of the memory and it has the advantage of presenting fast access time and has a high tolerance to radiation effects. This NAND flash uses Single-Level Cell (SLC) NAND technology. Storing 1 bit of data per memory cell, SLC NAND offers fast read and write capabilities and boot times, excellent endurance and reliability. Because of these good characteristics, between the two main types of flash memory it has been chosen a NAND flash instead of a NOR flash. They use the same cell design but differ at a circuit level. The temperature range is: -55°C to 125°C.



Fig 4.5: NAND flash module

The NAND memories are accessed like block devices, such as hard disks. Each block consists of a number of pages. A more precise description of this kind of structural organization is reported in the figure below.

From the figure it is possible to recognize:

- A Logical Unit (LUN) that is the smallest unit that can independently execute commands or report status.
- Two planes where concurrent operations can take place although with some restrictions.
- 2048 x 2 blocks which are the smallest unit that can be erased.
- 128 x 2048 x 2 pages which are the smallest unit that can be programmed.

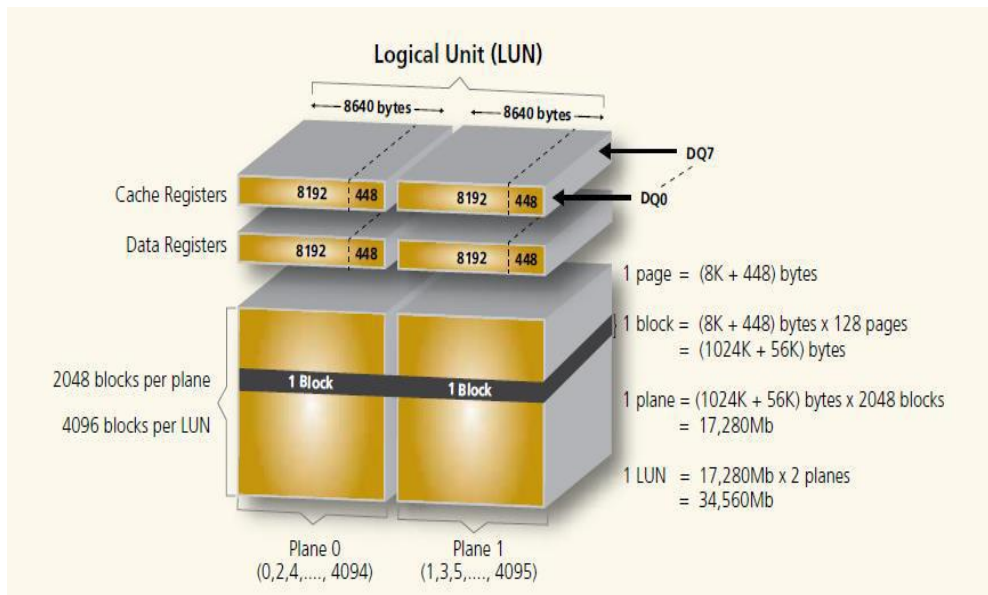


Fig 4.6 - NAND flash array organization

4.3 OBDH Motherboard

The OBDH motherboard is the NanoDock DMC-3. A figure of the said component is reported down below. The motherboard follows the standard PC/104, meaning that offers a series of advantages in building a computer. It is stackable, and different PC/104 can be mounted on top of each other to form finally a PC/104 stack. Its connectors readily allow this modularity of the computer. The PC/104 is also inherently rugged: it can withstand high vibration scenarios, mostly due to its small footprint, and allows operation in a range from -40 to +85°C.

The NanoDock DMC-3 is produced by GomSpace, and works in conjunction with their other products, primarily the NanoMind (actively supported by it) and the NanoCom AX100, the UHF transceiver. The motherboard allows 4 components to be mounted on top or at its bottom. It was decided to mount one NanoMind A3200 and 2 transceivers. The mass of the dock itself is about 51 grams and its dimensions are 91.9 x 88.7 x 8.6 mm.

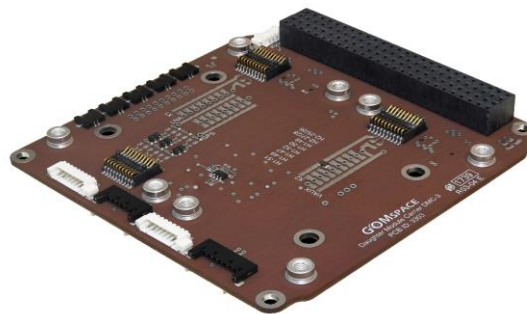


Fig 4.7 – NanoDock DMC-3

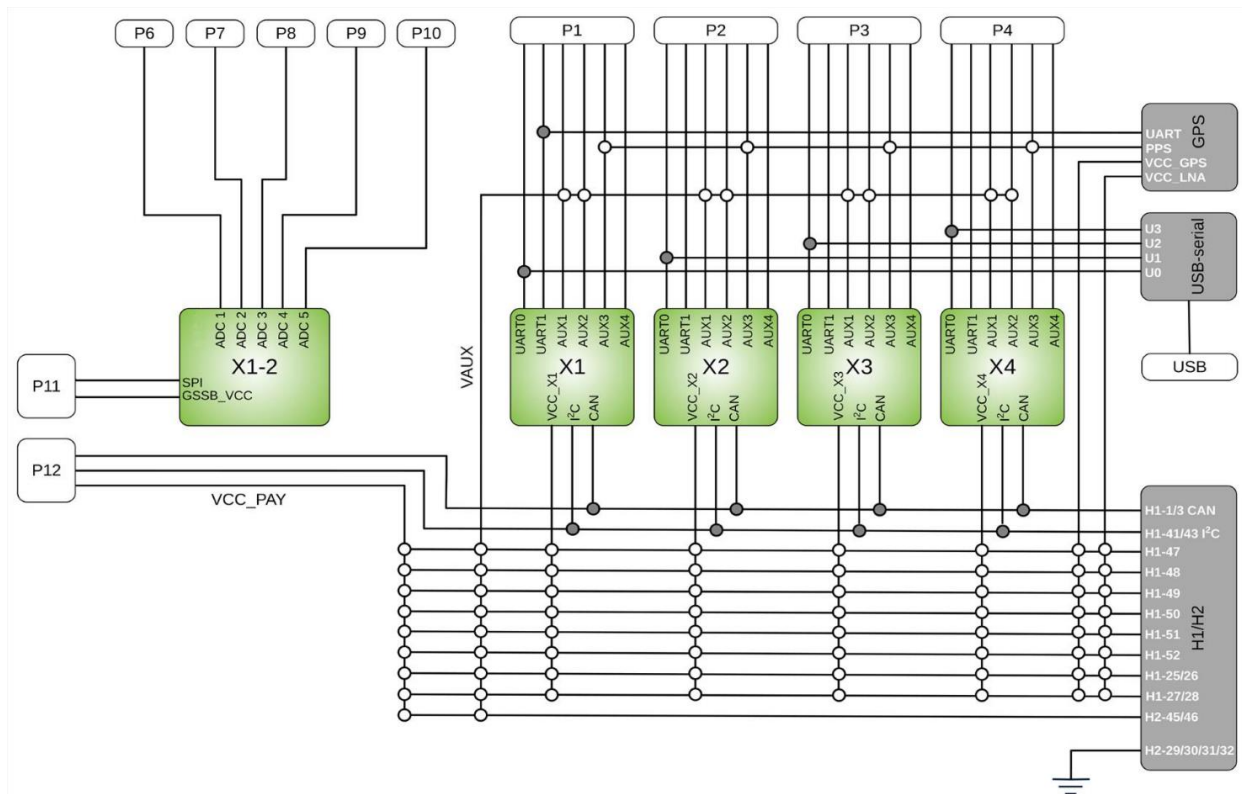


Fig 4.8 - NanoDock DMC-3 block diagram

The block diagram above illustrates all the connections the DMC-3 offers. The board is designed to be very flexible allowing any daughterboard to be supplied from any of the power supply pins used in GomSpace CubeSat products. The white dots show configurable connections. Gray dots show permanent connections.

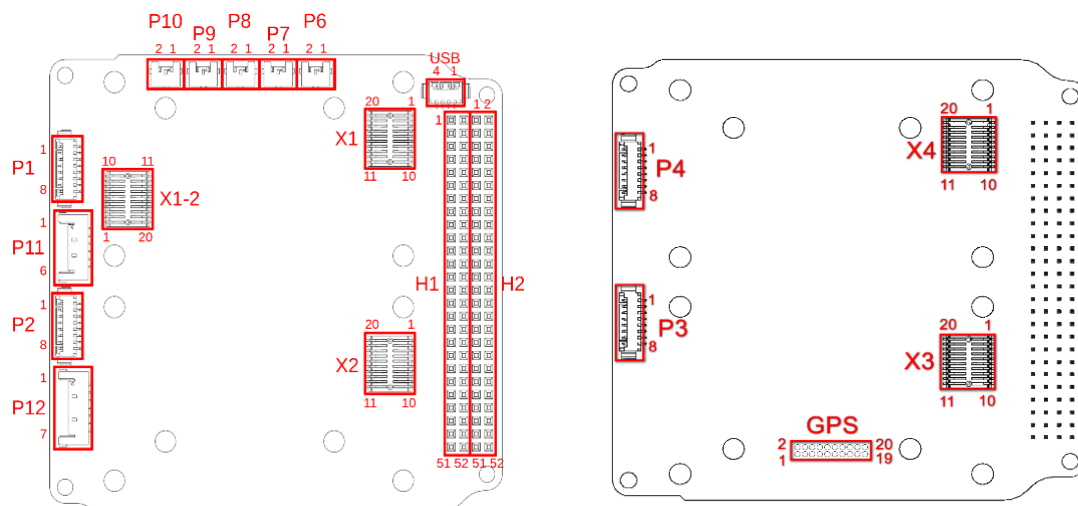


Fig 4.9 & 4.10 - NanoDock DMC-3 top and bottom view

Stack Connector H1/H2	H1 & H2 <ul style="list-style-type: none"> ○ 2x CAN ○ SDA & SCL ○ GND ○ 5 & 3.3 V
X1-FSI	<ul style="list-style-type: none"> ○ 4x GND ○ SCL ○ SDA ○ 2x CAN ○ 4x UART0 ○ 2x I2C
X2-FSI	<ul style="list-style-type: none"> ○ SCL ○ SDA ○ CAN high ○ CAN low ○ 4x UART
P1 & P2 – Breakout Connector	<ul style="list-style-type: none"> ○ 4x UART
P6-P10	<ul style="list-style-type: none"> ○ 4x UART ○ Analog In/GPIO
P12-I2C and CAN	<ul style="list-style-type: none"> ○ SCL ○ SDA ○ 2x CAN

X3-FSI	<ul style="list-style-type: none"> ○ SCL ○ SDA ○ 2x CAN ○ 4x UART
X4-FSI	<ul style="list-style-type: none"> ○ SCL ○ SDA ○ 2x CAN ○ 4x UART
P3 – Breakout Connector	<ul style="list-style-type: none"> ○ 4x UART
P4 – Breakout Connector	<ul style="list-style-type: none"> ○ 4x UART

4.4 Interface board

The interface board is another circuit integrated board that allows for flexibility in designing a computer architecture. It is built by Eurotech and its model name is COM-1274. It follows the before mentioned PC/104 standard, which result in easy structural interfacing between the motherboards. The selected interface board gaps the bridge between the CAN bus and the RS-422/485 connections, which are used to connect to the primary payload and to the S-

band transmitter. It also supports the FLASH memory as it directly interfaces it and grants the two OBC (both central and AODCS dedicated units) access to the memory through itself.



Figure 4.11 - Interface board

The J1 and J2 connectors are designed to allow the connection of the COM-1274 according to the PC/104 specifications., while the J7-J14 support Serial Ports such as the eight-software selectable RS232/RS422/RS485. Lastly, up to two CAN interfaces are available on the J15 connector.

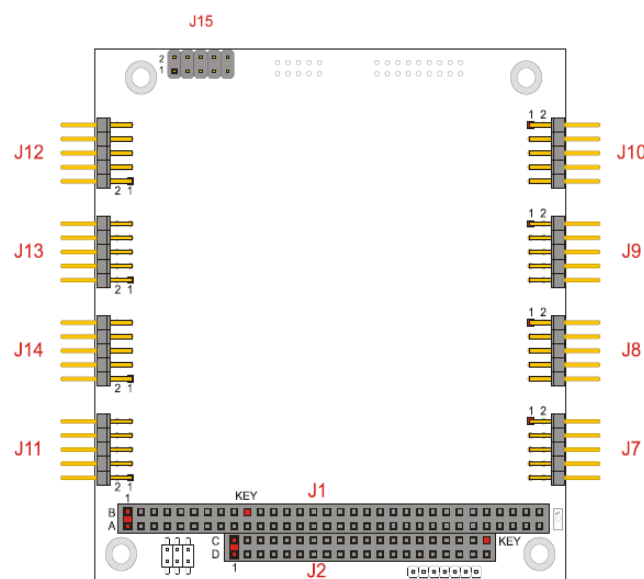


Figure 4.12 – Interface board connectors layout

Chapter 5

Architecture

Architecture is a framework for developing a computer system. The architecture shows the system's parts and how they interact through a block diagram. Data architecture addresses the physical structure of the data network or bus, as well as the protocol or logical interaction across the bus. The OBC architecture must be flexible and general-purpose to increase its reusability. Three different data architectures are used to connect the board computer with external components or subsystems.

5.1 Architecture models

A Centralized Architecture has Point-to-Point interfaces between OBC and other satellite subsystems. It is also referred as Star Architecture. Its advantages are that failure of a component or line does not cause system loss and that individual interfaces are possible for secure data. Its disadvantages comprise a high wiring effort is necessary which may also cause electromagnetic compatibility.

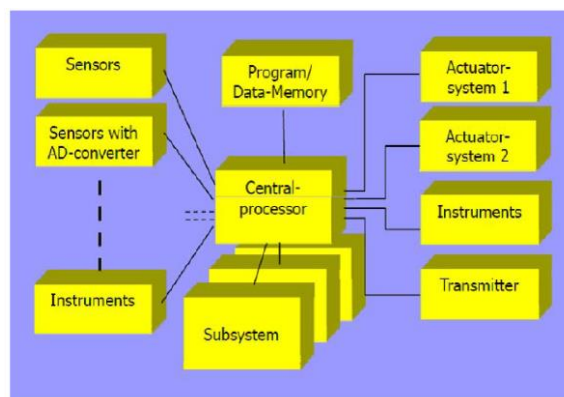


Fig 5.1 - Centralised architecture.

Often called ring architecture, it establishes a way to arbitrate information flow control as the data are passed in a circular pattern. Its advantages are that a low cabling effort is needed and can be distributed throughout the satellite structure. Components can also be tested independently and adding new nodes will have limited impact on OBC. Its disadvantages consist of less reliability since each node is in-line and thus required to achieve transmission to the next node. Address decoder in each node is required.

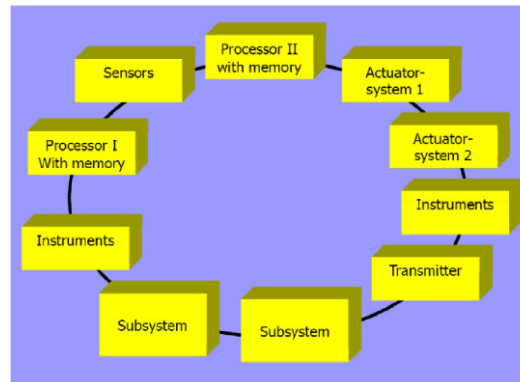


Fig 5.2 - Ring architecture.

The bus architecture utilizes a common data bus with all subsystems sharing the bus. It uses standard protocols and communication schemes for all nodes. Its advantages are the simple realization of the system, deterministic data transmissions which reduce test and troubleshooting time while increase reliability. Its disadvantages are that all components must be developed with a specific interface (both physically as well as electrically). Also, a failure of a component or line may cause the loss of the system.

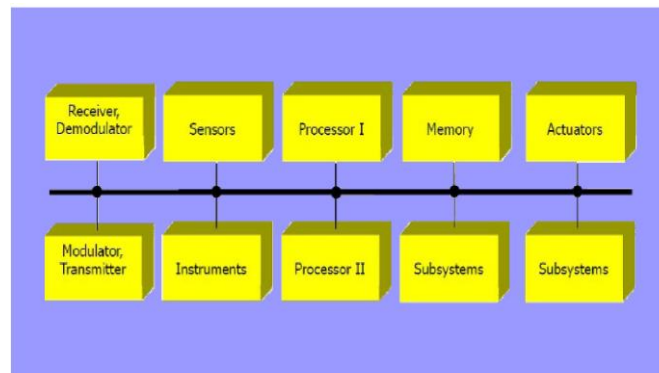


Fig 5.3 - Bus architecture

5.2 Actual architecture

In the case of this mission there is a central on-board computer like in the centralised architecture. However, it is not the only processor on board the CubeSat. There are two other dedicated on-board computers: one that supports the operations of AODCS subsystem and another that is proper of the hyperspectral camera.

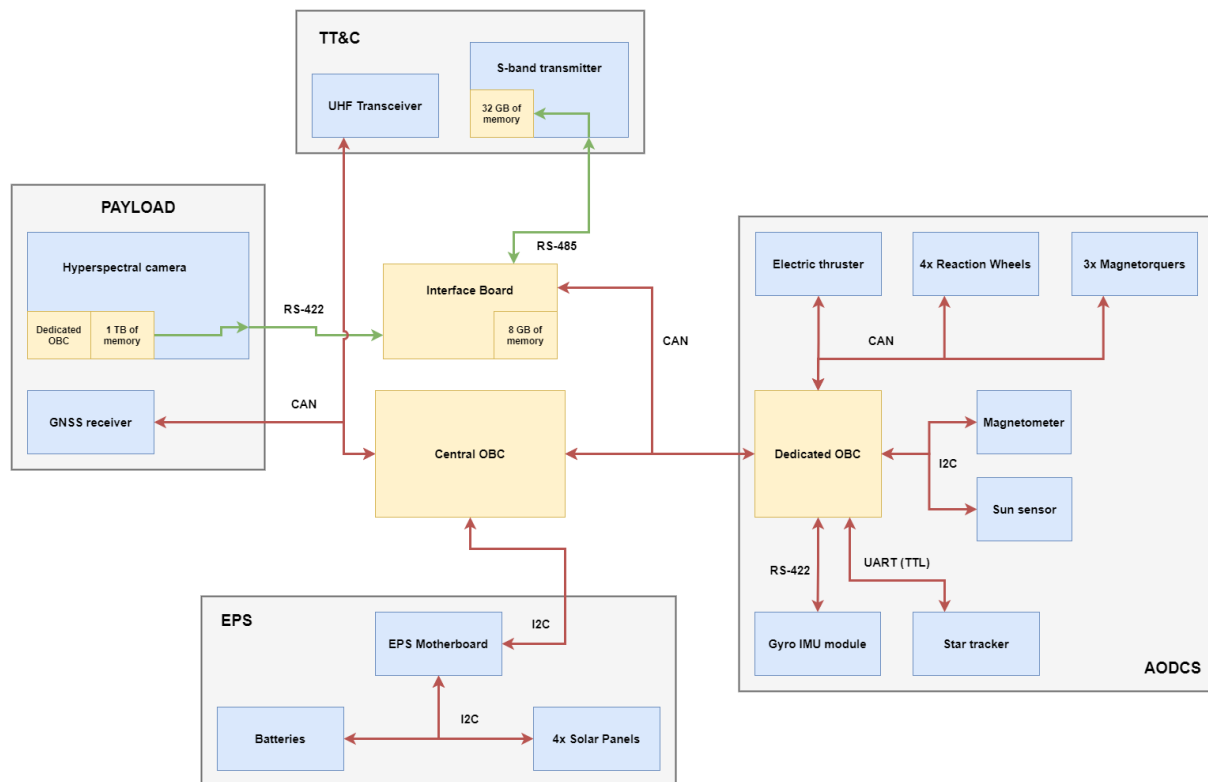


Fig 4.3 - Spacecraft scheme

In the figure above it is reported a simplified scheme of the CubeSat. The intention is to highlight the most significant characteristics of how the data flow and how they are exchanged between and through the different subsystems and the various components on board the CubeSat. The actions of the AODCS subsystem are managed by a dedicated OBC. In fact, considering the peculiar characteristics of this mission (observation of the Earth is one of the most important objective) the attitude control of the CubeSat represents a critical aspect that in order to be efficiently performed needs real time complex algorithms to be continuously executed during CubeSat orbital motion. The transmission to ground of payload data from the hyperspectral camera is performed through the S-band transmitter while the transmission of other kinds of data from the rest of the CubeSat subsystems is performed through the UHF transceiver. Of course, these kinds of operations are monitored by the central OBC. The data from the GNSS receiver are managed by the central OBC that can decide to send them to the UHF transceiver (and so subsequently to ground) or make them accessible to the AODCS subsystem. The central OBC must organize and schedule the various operations on board the CubeSat and the commands received from ground. Not in all scenarios and not all kinds of data are directly and immediately transmitted to the components of TT & C subsystem (UHF transceiver and S-band transmitter). Sometimes there can be the necessity to store them in some memory before their transmission. For this reason, besides the flash memory, it is useful the exploitation of the memory of the hyperspectral camera and the one on board the S-band transmitter.

Chapter 6

Operation

6.1 Definition of the CubeSat operational modes

The definition of spacecraft (including CubeSat) modes is a topic covered in the spacecraft operability concept. The modes have already to be considered during spacecraft system conceptualization, must be refined subsequently over the development phases and must be treated during OBDH requirements definition, OBDH design and testing. However, the spacecraft modes and the system autonomy become fully visible not before the start of spacecraft operations tests. The CubeSat is designed to undergo different operational modes. The autonomous switching is preferred over manual switching since it does not require telecommand and hence the modes can be switched even when the CubeSat is not in line of sight. However, this is not always possible. The CubeSat operational modes that have been identified are:

- launch mode
- safe mode
- nominal mode
- detumbling mode

6.2 Transition between the different modes

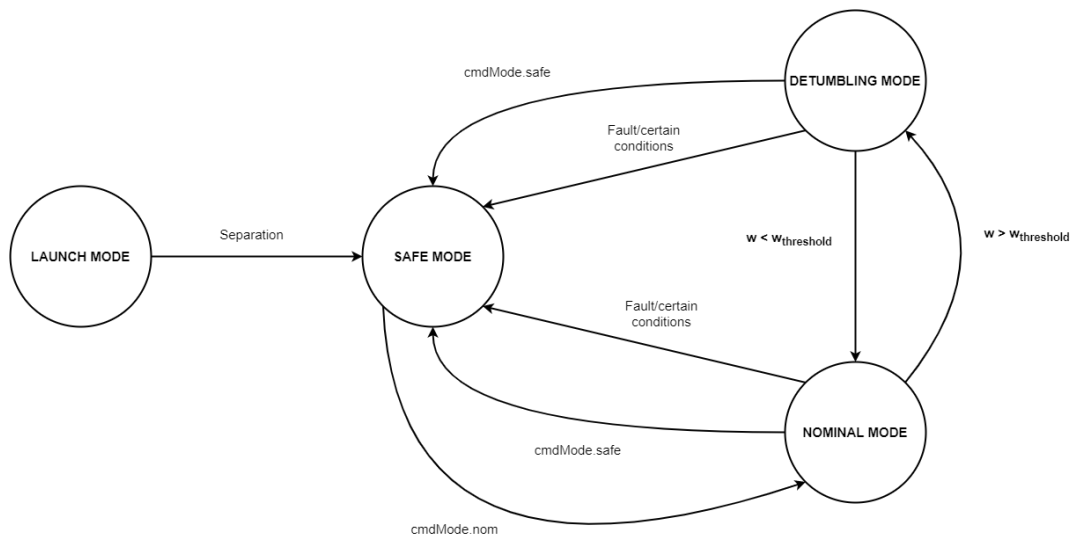


Fig. 6.1 – Status transition

6.3 Launch mode

- is only useful in the launch phase until the orbital deployment of the CubeSat
- the kill switches are pressed down while the CubeSat is placed in its orbital deployer
- while the buttons are pressed the CubeSat cannot power up
- as soon as the CubeSat is released, it can power up

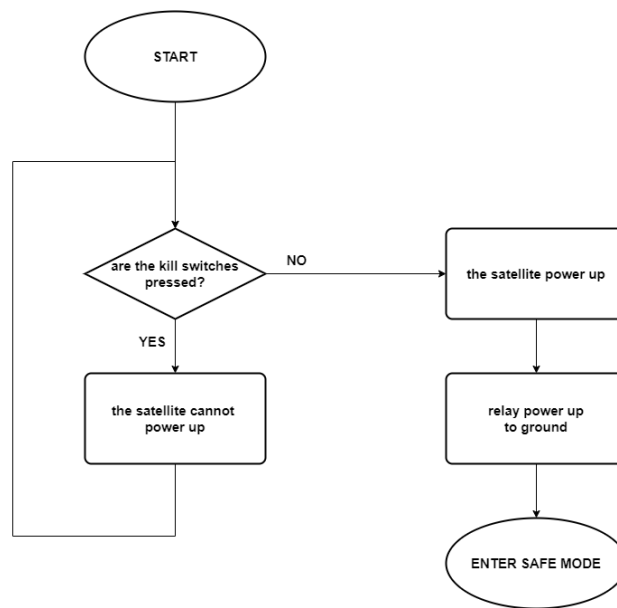


Fig 6.2 - Launch mode flowchart

6.4 Safe mode

- is entered when a fault in components is detected
- is entered when dangerous states (previously defined) are met
- thermal is monitored
- power is switched off for non-essential components
- attitude control is active
- radio link is active (telemetry only on request, telecommand always available)
- science payload is inactive
- housekeeping is maintained
- provides access to utility and diagnostics programs for troubleshooting the fault
- error codes implemented for use at the ground station
- is intended for maintenance, not functionality

Safe mode is the operating mode of the CubeSat during which all non-essential systems are shut down and only essential functions such as thermal management, radio reception and attitude control are active. Safe mode can be entered automatically upon the detection of a predefined operating condition or event that may indicate loss of control or damage to the CubeSat. Usually, the trigger event is a system failure or detection of operating conditions considered dangerously out of the normal range. For example, cosmic rays penetrating spacecraft electrical systems can create false signals or commands and thus cause a trigger event. The central processor electronics are especially prone to such events. Another example could be that hardware failures can cause the lack of received command within a given time window. The process of entering safe mode, sometimes referred to as safing, involves several immediate physical actions taken to prevent damage or complete loss. Power is removed from non-essential subsystems. Regaining attitude control, if lost, is the highest priority because it is necessary to maintain thermal balance and proper illumination of the solar panels. On board software (OBSW) is operational in safe mode controlling the CubeSat in a way to assure attitude stability and sufficient power generation by solar array

pointing. Unnecessary equipment such as payload instruments and other equipment from the payload data handling chain is not used due to interrupted mission product generation and will be shut off or down to a low resource consuming state.

Safe mode can be induced from ground via the following mechanisms:

- By execution of a dedicated High Priority Command for safe mode
- By execution of a dedicated safe mode TC function representing a critical command

Safe mode can be induced on board at least by the following mechanisms:

- Failure detected by AODCS
- Failures detected by essential system monitors
- System undervoltage detection

The severity of a safing event is realized as an operational impact, requiring time to diagnose and recover the CubeSat before resuming the nominal mission sequences. To capture these details, a standardized timeline structure is presented.

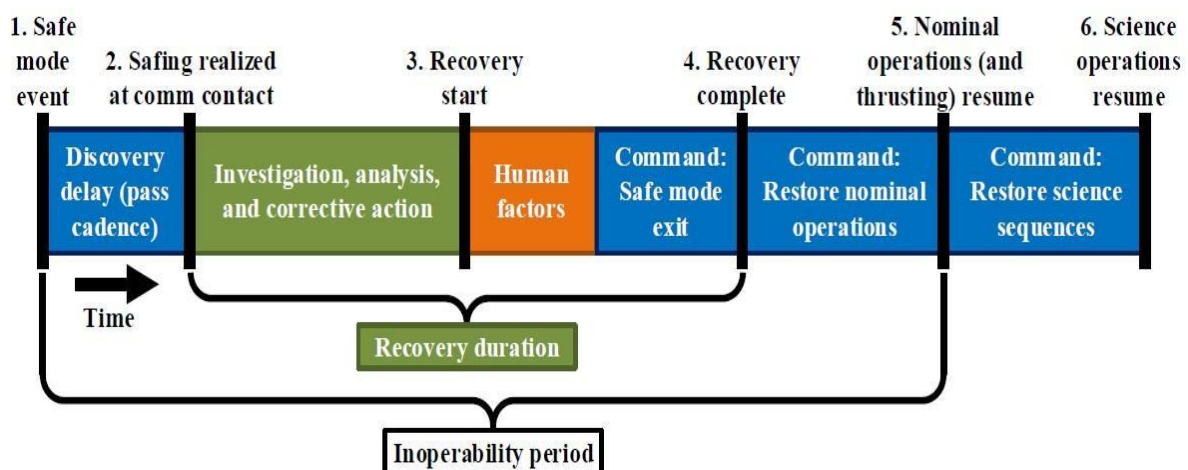


Fig. 6.3 – Safe mode timeline

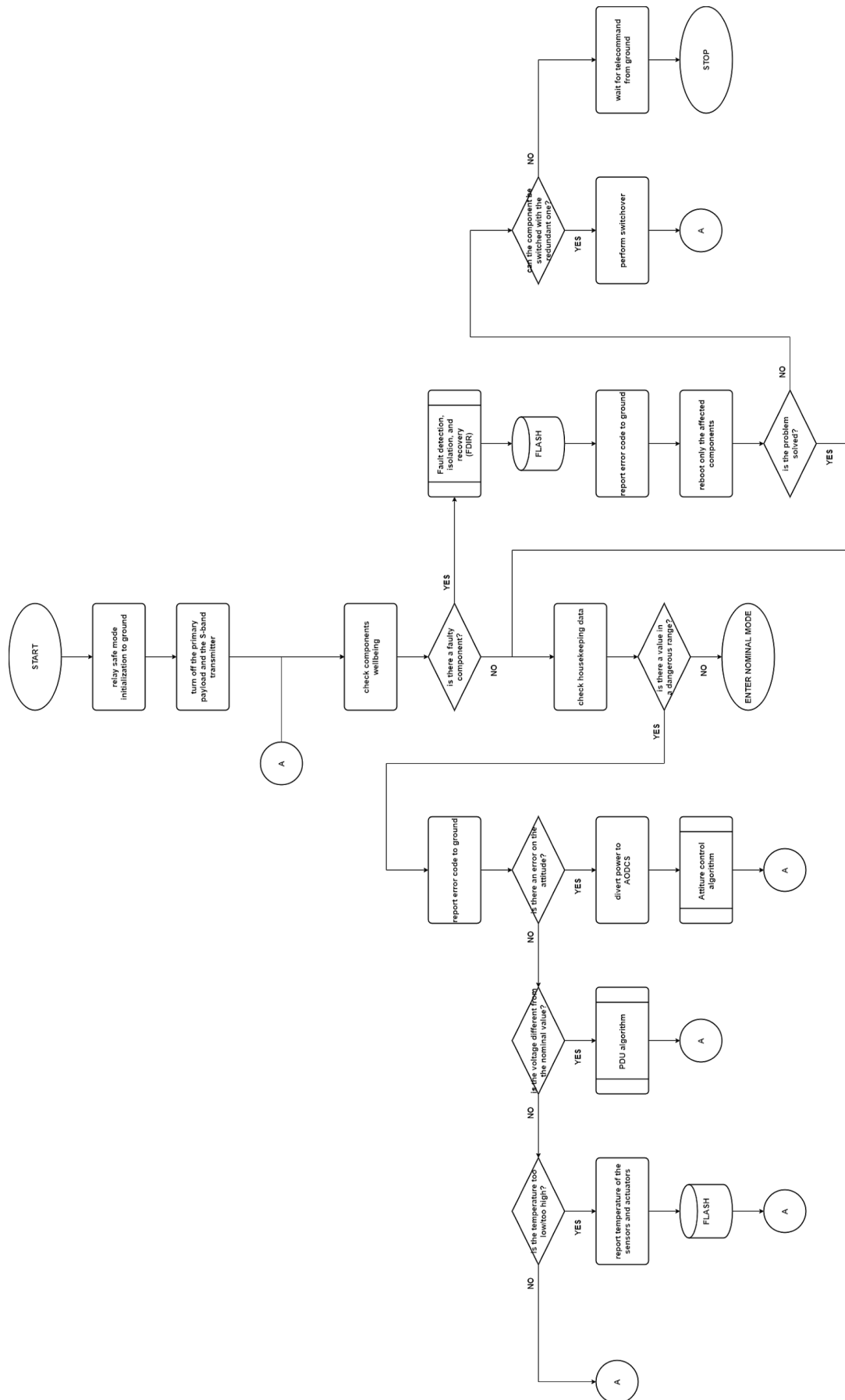


Fig 6.4 - Safe mode flowchart

6.5 FDIR

FDIR is a key functionality of the OBSW. Obviously not all failures are subject to onboard identification and not all failures are subject to onboard recovery. The FDIR concept to be worked out for the spacecraft during the engineering phase follows some basic requirements and principles, implements a certain failure hierarchy- specifying furthermore on which level the failure is to be fixed- and finally it implements a consistent approach for the functionality transferring the spacecraft to Safe Mode and how to recover from there. Since a FDIR concept usually follows a hierarchical approach, in the following it will be indicating a FDIR and safeguarding hierarchy example. In this structure there are indicated the levels of failures which handled by unit internal, subsystem software, satellite system software, onboard computer hardware reconfiguration unit and ground.

FDIR concept usually follows a hierarchical approach:

Level 4 Handled by Ground	Major overall system failures - Communication failures - Deployment failures - etc.		
Level 3 Handled by OBC HW Reconfiguration Unit	Hardware induced alarm - Multiple EDAC alarms - S/C power failures - etc.		
Level 2 Handled by S/C System SW	System Malfunction - Attitude computation inconsistencies - S/C power failures - etc.		
Level 1 Handled by Subsystem SW	Subsystem Malfunction - subsystem equipment failure - subsystem intercommunication failure - etc.		
Level 0 Unit internal Handling	Unit internal Malfunction - internally recoverable - EDAC error or similar - etc.	Unit internal Malfunction - requiring instant reaction - short current protection - etc.	Data bus Malfunction - recoverable failure - MIL-bus retries - etc.

Fig 6.5 – FDIR hierarchy example

The lowest level comprises the handling of failures entirely on unit level, either because it is feasible – such as EDAC error handling – or because the equipment by default provides this feature, or because a certain FDIR function on lowest level is extremely time critical – such as reaction to short currents or overvoltage. This level also comprises data bus failures invoked by electromagnetic effects and the like.

The next higher levels 1 and 2 cover failures being handled on OBSW level, either on subsystem control level or requiring upper system level. Examples are also indicated in the figure. On this level of above equipment there are monitors available for limit check of unit parameters but also subsystem level abstract verifications such as for example a plausibility check of GPS provided position against internal solution from orbit propagator functions.

Level 3 then comprises failures which need hardware reconfigurations via the OBC's reconfiguration unit. These include the monitoring and reaction to HW alarms and the like.

And finally, level 4 comprises the failures that cannot be handled on board the S/C itself without ground intervention at all.

6.6 Detumbling mode

- attitude control is active (emphasis is on the angular velocity, which has to be reduced within acceptable limits)
- housekeeping is active
- power is diverted to attitude components (mostly magnetometer and magnetorquer)
- thermal is monitored
- payload camera is kept off from working until detumbling is completed
- S-band transmitter is powered off
- radio link is not guaranteed

The detumbling mode is the mode in which the main task is to detumble the satellite. In all modes, the OBC is collecting data regarding the state of the satellite. One of the parameters collected as part of this housekeeping is the angular velocity. If the angular velocity is found to be above the threshold, then the CubeSat is considered to be tumbling. It is important that the CubeSat should not continue to tumble because a tumbling satellite generates much lesser power and takes a long time to perform more productive operations like downlinking and image taking. This mode has higher priority than all other non-emergency modes and is immediately entered into if the angular velocity goes above ω_{entry} . In the detumbling mode the satellite is stabilized by creating a restoring torque. The uncontrolled rotations are damped by the actuators which receive information by the B-dot control law. The B-dot control processes the magnetometers readings. The satellite continues to be in the detumbling mode until the angular velocity becomes less than ω_{exit} . In general $\omega_{\text{exit}} < \omega_{\text{entry}}$ allowing for a more stringent leaving criterion. This is done to avoid oscillation between tumbling and detumbling conditions. The angular velocity values are received continuously from IMU which are compared with the previous angular velocity after each B-dot control law execution. Iterations of the B-dot control law occur hence depends on the power available to the system. Therefore, even if power is relatively low, it will still go into the detumbling mode and perform the B-dot at low frequency. This will be the case until the power is so low that the AODCS subsystem components need to be switched off. Since the amount of power generated while the CubeSat is tumbling is diminished, it could be useful to try to decrease the amount of power consumed in this mode.

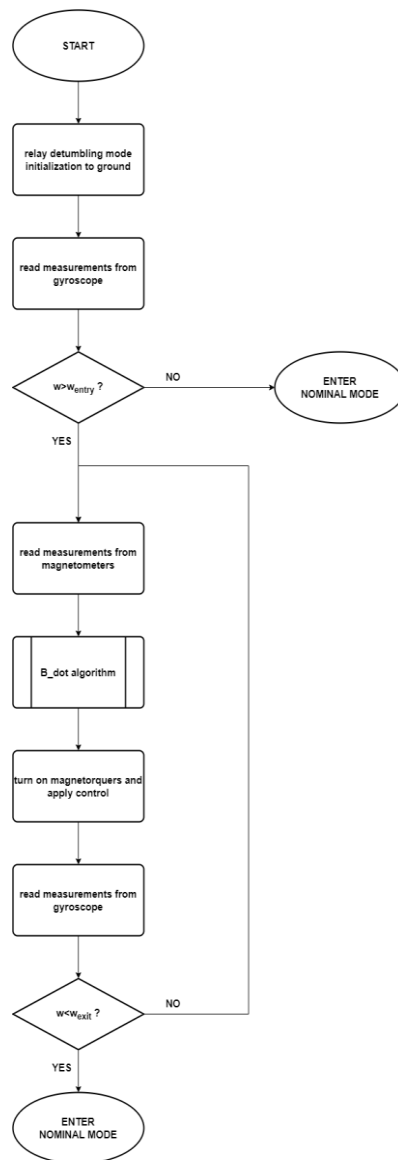


Fig 6.6 - Detumbling mode flowchart

6.7 Nominal mode

- power is switched on for the requiring components
- radio link is active (telemetry and telecommand)
- payload and S-band transmitter are turned on in certain time intervals
- attitude control is active
- housekeeping is active
- thermal is monitored

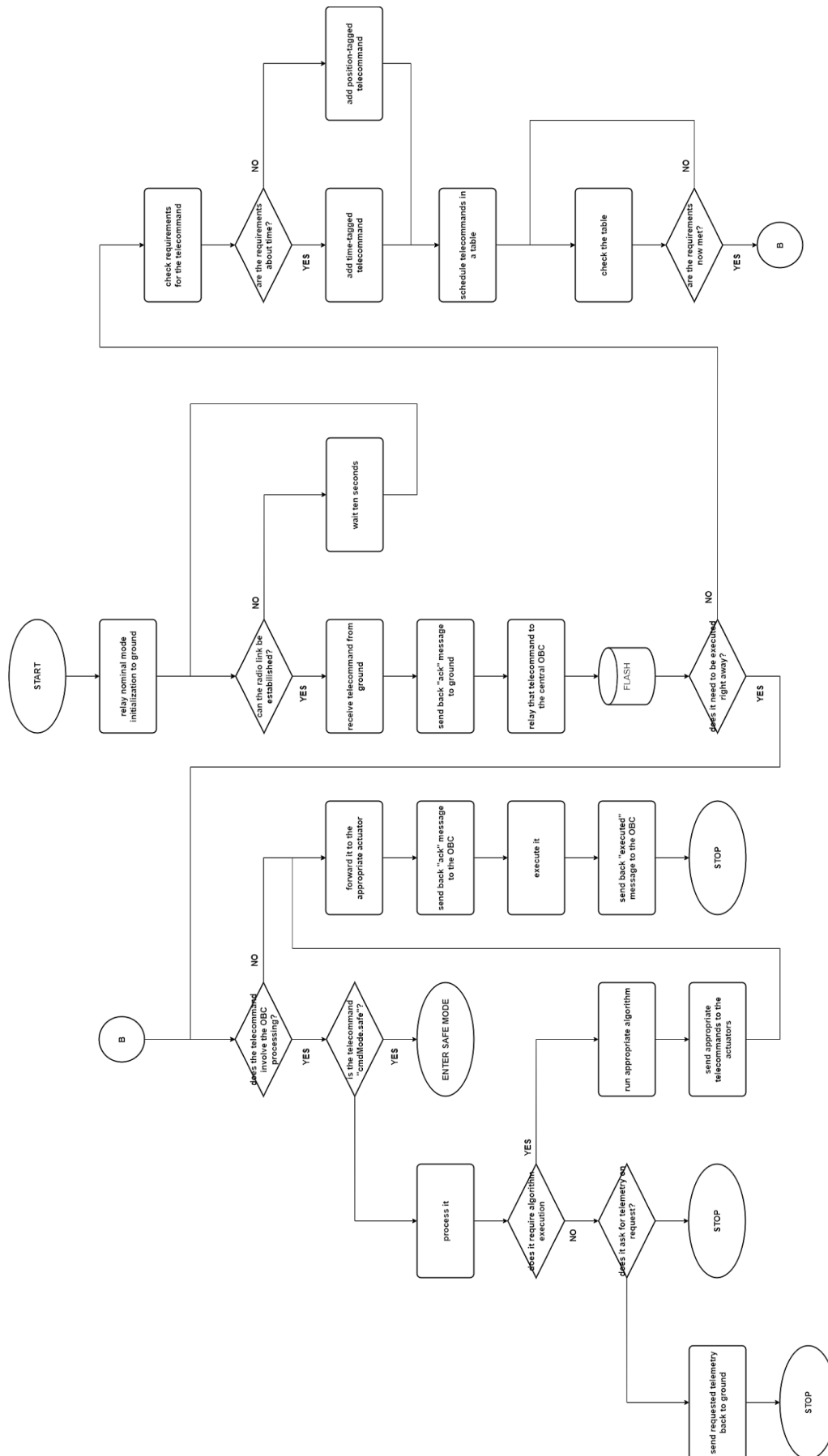


Fig 6.7 - Nominal mode (telecommand) flowchart

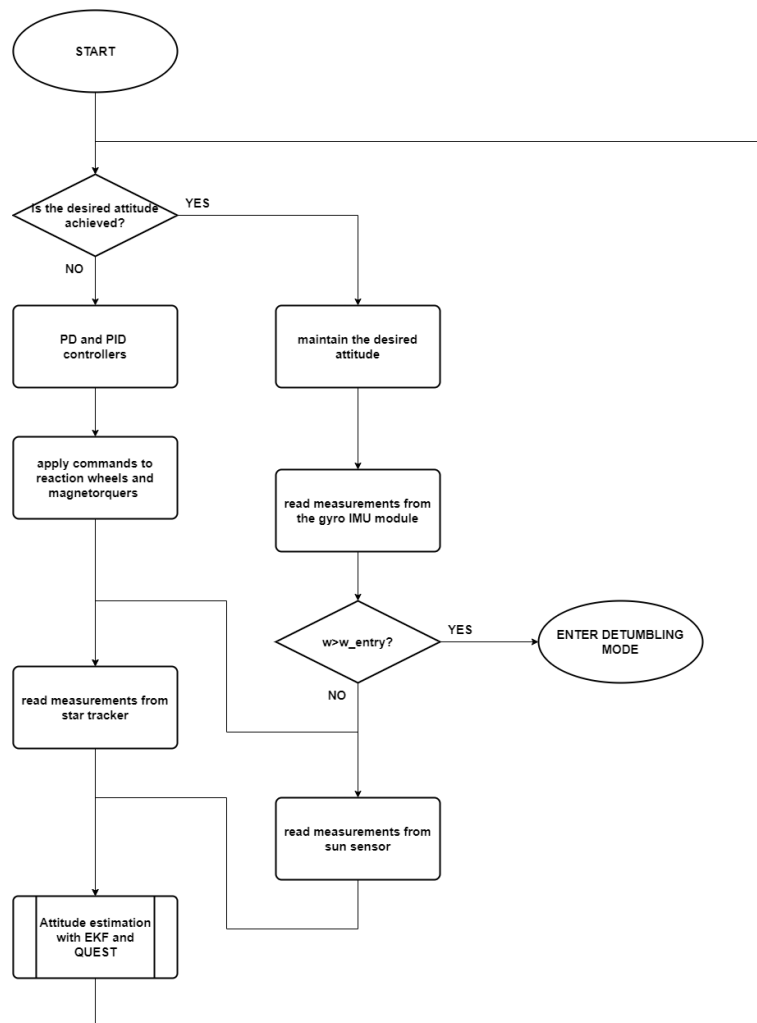


Fig 6.9 - Three-axes stabilization flowchart

6.8 Primary payload strategy

The operation scheme for the payload data generated by the hyperspectral camera is shown in the following figures. Usually, the camera turns on when in coverage of the continent of Europe, and this event happens quite regularly for a LEO orbit. Since the revisit time of the mission orbit is 7 days, the strategy for the acquisition of data and its subsequent downlink is to be analysed within those 7 days and then extended to the full course of the mission through simple repetitions.

The OBC runs a flight software that can keep track of where the satellite is positioned in time. Through orbit propagation, it knows the instances of time when the pass over Europe begins and when it ends and can therefore command the hyperspectral camera to activate and start and stop acquiring. Usually, several observations are taken each single day. At the end of each acquisition, the OBC on board the camera will process the data and store it on the internal SATA drive. It does not proceed to compress the data until the last observation for the day is completed: then after the compression of the whole day, the compressed information is routed through the interface board to the memory on board the S-band transmitter, which will eventually take care of downlink that same data. The compressed

data is also stored in the same SATA drive, at least to work as a backup copy, in case something was to go wrong in the transfer or downlink process. This strategy of acquiring and storing the compressed data encompasses just the first 5 days of the 7-day period. Indeed, the short revisit time makes sure that the CubeSat is not in coverage for long times of the European continent, so to optimize the acquisition process it was thought best to use this idle time to downlink the data acquired in the 5 days prior. Downlink will happen as soon as the last observation of the 7-day period has been transferred to the transmitter memory. The chosen ground stations for the payload data are Rome, Italy and Svalbard, Norway and grant short, but frequent visibility windows in which the large data can be downlinked.

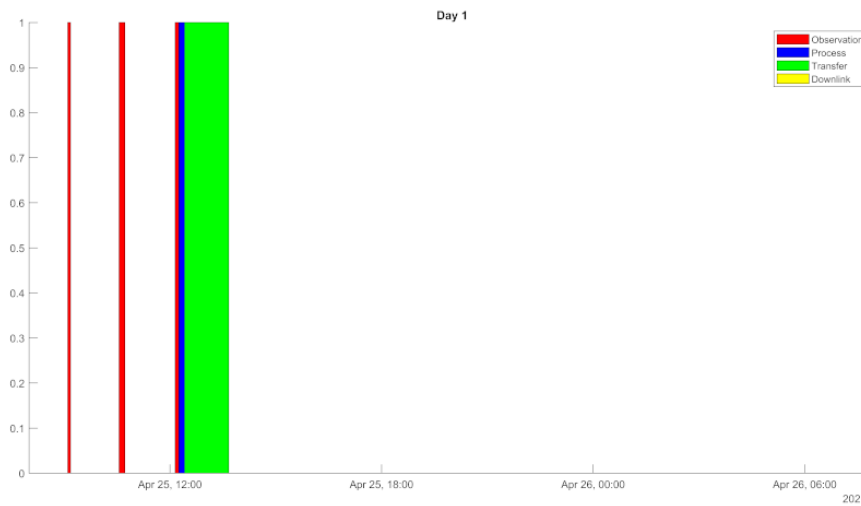


Fig. 6.10 – Day 1 of the primary payload strategy

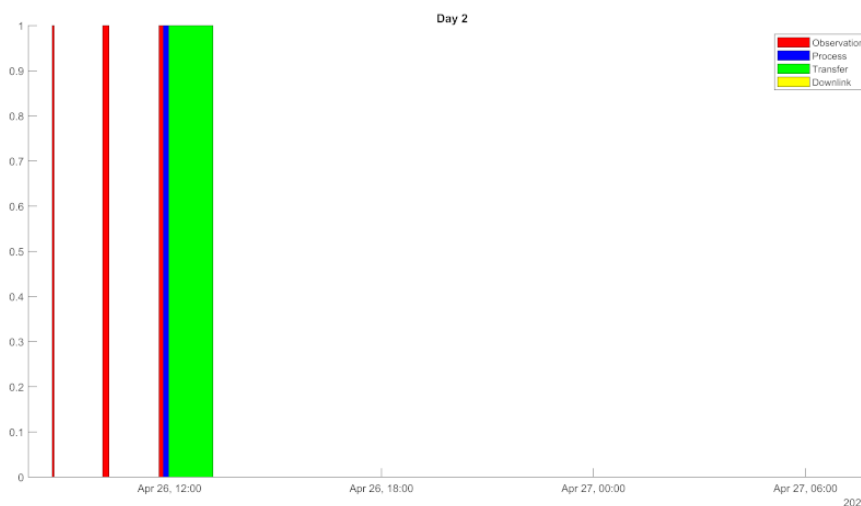


Fig. 6.11 – Day 2 of the primary payload strategy

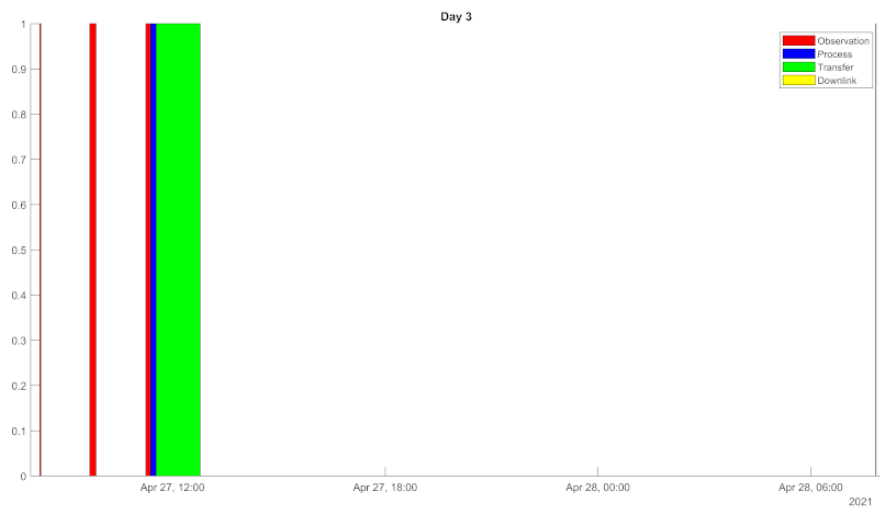


Fig. 6.12 – Day 3 of the primary payload strategy

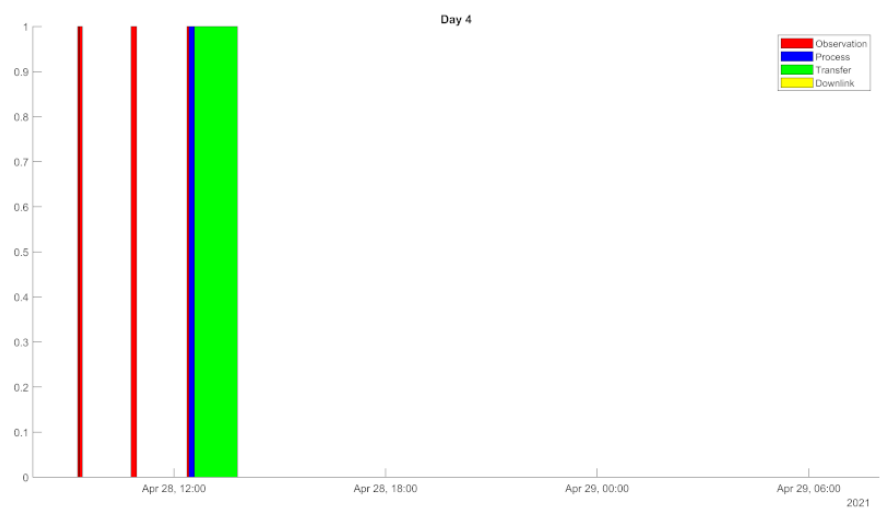


Fig. 6.13 – Day 4 of the primary payload strategy

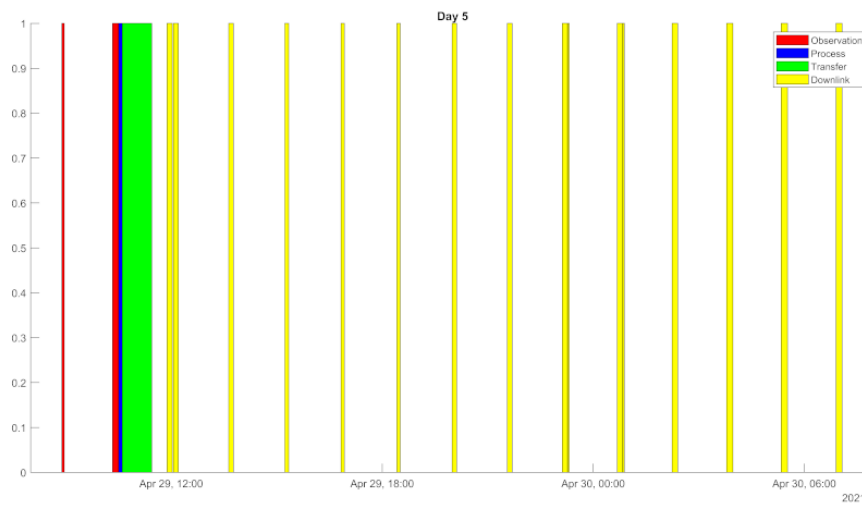


Fig. 6.14 – Day 5 of the primary payload strategy

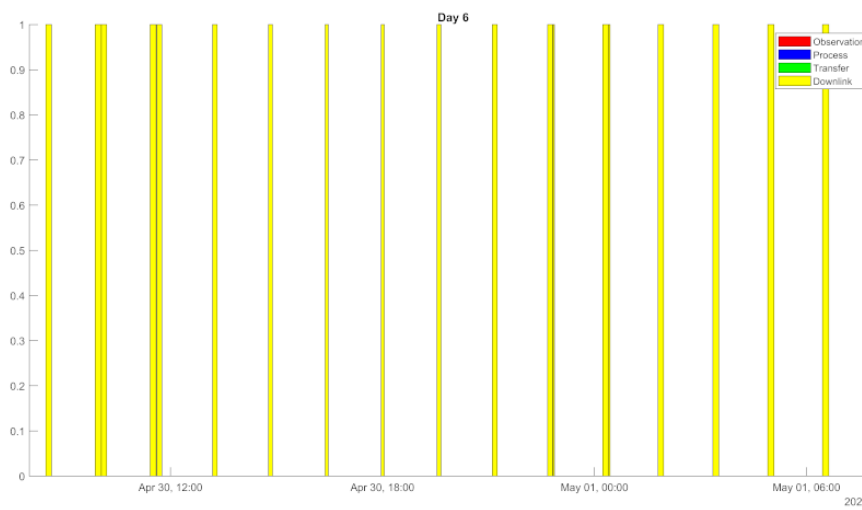


Fig. 6.15 – Day 6 of the primary payload strategy

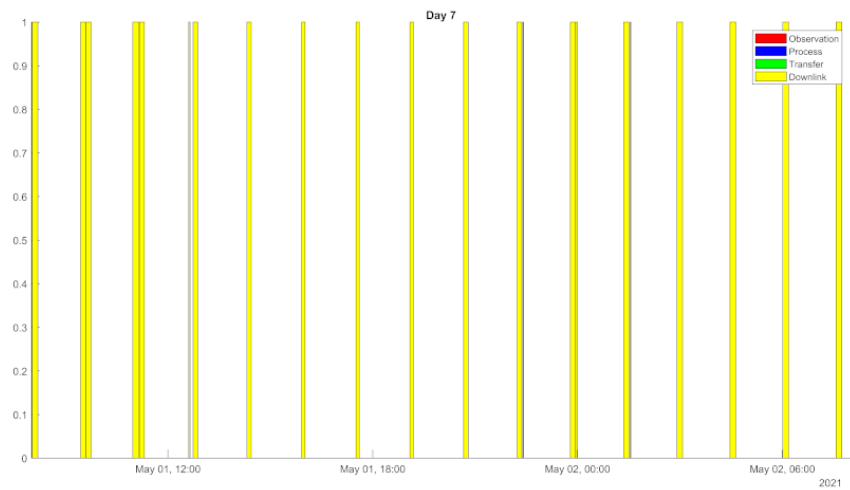


Fig. 6.16 – Day 7 of the primary payload strategy

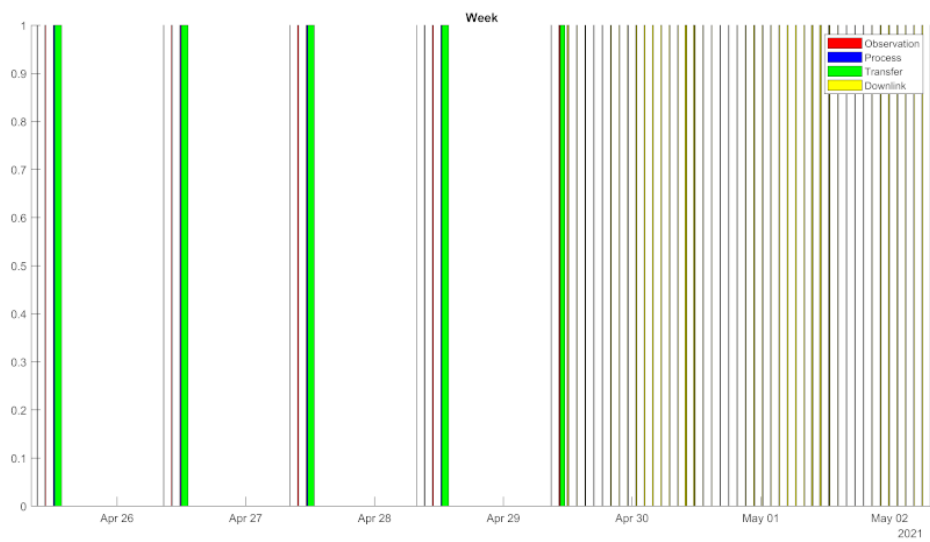


Fig. 6.17 – 7-day period of the primary payload strategy

Chapter 7

Software

7.1 Operating System

In previous chapter we have decided that the OBC should sport a Real-time operating system (RTOS). FreeRTOS was chosen as this RTOS. In this chapter the setup and configuration of the OBC RTOS will be further elaborated upon, starting with choosing the FreeRTOS version, before discussing the configuration. FreeRTOS source code is available from Sourceforge (<https://sourceforge.net>). The latest release candidate is FreeRTOS V10.4.3 released on Dec 2020.

7.1.1 FreeRTOS Initialization

There are two types of RTOS scheduler which are the non-preemptive and preemptive type. In the non- preemptive, once a task is started, the task must be executed until completed or until it is blocked or waiting for resources. While in the preemptive scheduler, a task can be interrupted by other tasks which more important (or have high priority) and then can be resumed. In general, if possible, a critical task should be allowed to interrupt other tasks which are less critical in order to meet the deadline. We use CoCoX CoOS preemptive scheduler in this research, in the main program, first we initialize the RTOS, and then we create and activate these three tasks. The priority of each task is shown in Table, the communication task has high priority, because the instruction coming from the ground station should be handle first.

Priority assignment	Priority	Task Delay
0	High	Communication
1	Medium	Housekeeping
2	Low	Condition-Check

7.1.2 Task Implementation

The Condition-Check task is a task with the lowest priority, which has a function to read current, voltage and temperature sensors on OBC/OBDH. The current values are obtained through EPS, the voltage values through a voltage sensor that reads by the microcontroller through ADC while the temperature from the temperature sensor through I2C communication.

The Housekeeping task arranges OBC/OBDH sensor data package, access to EPS and TTC emulators, as well as preparation of three housekeeping data packets into a single packet of data. The Housekeeping task also set the delay time according to the needs of

data storage tasks. In this case, the time delay is implemented only for 30 seconds to replace the 15 minutes of the satellite in space.

The Communication task has to serve communications access with the ground station. This task has the highest priority based on the needs of communication between satellites with GS which should be responded quickly.

7.1.3 Choosing Memory Scheme

A real time operating system kernel must allocate RAM dynamically each time a task, queue, or semaphore is created. The use of the standard malloc() and free() library functions can be accompanied with some undesirable side effects for one or more of the following reasons:

- Their implementation can be relatively large, taking up valuable code space.
- They are rarely thread safe.
- They are not deterministic; the amount of time taken to execute the functions will differ from call to call.
- They can suffer from memory fragmentation.
- They can complicate the linker configuration.

To be able to best satisfy the different requirements of different applications, FreeRTOS allocates memory by calling pvPortMalloc() and frees it by calling vPortFree(), where the implementations can be decided by the programmer.

7.1.4 Configuring FreeRTOS

FreeRTOS is customized using a configuration file called 'FreeRTOSConfig.h'. Every FreeRTOS application must have a FreeRTOSConfig.h header file in its pre-processor include path. For the OBC application the configuration file is included into the config folder along with all the other configuration files. There are many settings one can configure for FreeRTOS, the most important will be shortly discussed below. To be able to best satisfy the different requirements of different applications, FreeRTOS allocates memory by calling pvPortMalloc() and frees it by calling vPortFree(), where the implementations can be decided by the programmer.

Scheduling: Preemptive scheduling is being used as this is the only mode that supports tickless sleep for FreeRTOS.

Heap Size: FreeRTOS heap is configured to be $384\text{Kb} * 50\% = 198\text{KB}$. This way, whenever an application developer violates the memory constraint, the malloc fail will be caught in the respective FreeRTOS hook and notify the programmer of the violation. If this happens due to some last-minute fix, the heap size can safely be increased. However, if it happens in normal design situation, the developer should further investigate the memory usage of the application and try to optimize the memory usage before proceeding.

Memory Violation and Debugging: FreeRTOS has support for stack overflow checking. This can be done in two ways. The first method (method one) is the fastest, but only checks that the processor stack pointer is within legal range when a task is switched out of the running state. To catch past overflows method two can be used, which includes the same check as method one, but also checks the last 16 bytes of the valid stack range, to see if they have been overwritten. This last method is very similar to that of using stack canaries. If a stack

overflow is detected, the stack overflow hook is fired. Malloc fail hook is also enabled. As well as the use of trace facility (needed for hooks used by Perception Trace Snapshot library).

7.2 Flight Software

The Core Flight System (cFS) has been chosen as the mission's main software, which is a platform and project independent reusable software framework and set of reusable software applications. There are three key aspects to the cFS architecture: a dynamic run-time environment, layered software, and a component-based design. The cFS architecture has been proven to:

- Reduce time to deploy high quality flight software.
- Reduce project schedule and cost uncertainty.
- Facilitate formalized software reuse.
- Enable collaboration across organizations.
- Simplify flight software sustaining engineering.
- Provide a platform for advanced concepts and prototyping.

The Flight Software Architecture consisting of an OS Abstraction Layer (OSAL), Platform Support Package (PSP), cFE Core, cFS Libraries, and cFS Applications.

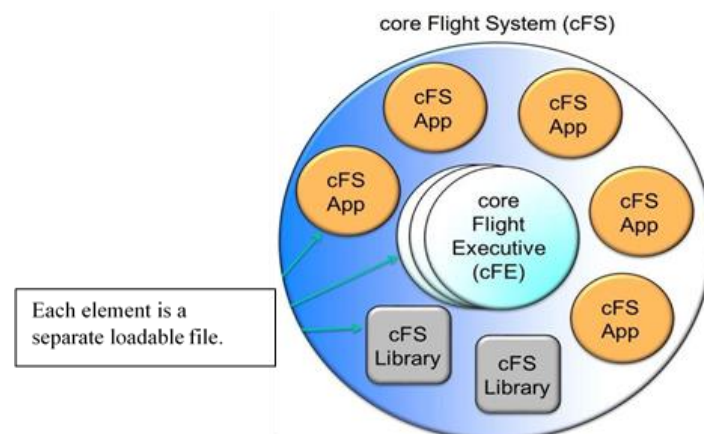


Fig. 7.1 – Flight software packages

The Core Flight Executive (cFE) services include:



Fig. 7.2 – Flight software core executive

The Executive Services (ES) provides ability to start, restart and delete cFS applications.

- On startup
- During runtime
- Manages a Critical Data Store which can be used to preserve data (except in the case of a power-on reset) Provides ability to load shared libraries.
- Provides support for device drivers.
- Log information related to resets and exceptions.
- Manages a system log for capturing information and errors.
- Provides Performance Analysis support.

The Event Services (EVS) provide an interface for registering an application's event filter masks, types, and type enable status. Also, provides an interface for un-registering an application from using event services.

The Software Bus (SB) provides a portable inter-application message service which Routes messages to all applications that have subscribed to the message.

The Table Services (TBL) manages all cFS table images.

The Time Services (TIME) provides a user interface for correlation of spacecraft time to the ground reference time (epoch) • Provides calculation of spacecraft time, derived from mission elapsed time (MET), a spacecraft time correlation factor (STCF), and optionally, leap seconds.

The Core Flight Software Applications

Applications	Functions
CFDP	Transfers/receives file data to/from the ground
Checksum	Performs data integrity checking of memory, tables and files
Command Ingest Lab	Accepts CCSDS telecommands packets over a UDP/IP port
Data Storage	Records housekeeping, engineering and science data onboard for downlink
File Manager	Interfaces to the ground for managing files
Housekeeping	Collects and re-packages telemetry from other applications
Health and Safety	Ensures that critical tasks check in, services watchdog, detects CPU hogging and calculates CPU utilization
Limit Checker	Provides the capability to monitor values and take action when exceed threshold
Memory Dwell	Allows ground to telemeter the contents of memory location. Useful for debugging
Memory Manager	Provides the ability to load and dump memory
Software Bus Network	Passes software bus messages over "plug-in" network protocols

Scheduler	Schedules onboard activities (e.g. HK requests)
Scheduler Lab	Simple activity scheduler with a one second resolution
Stored Command	Onboard Commands Sequencer (absolute and relative)
Telemetry Output Lab	Sends CCSDS telemetry packets over a UDP/IP port

Chapter 8

Protocols

A protocol is a set of rules that are agreed upon to establish the communication between two or more entities. Indeed, communication is realized through cooperation of more entities with the objective of transferring information. More than one protocol is needed to achieve this transfer, as each layer of the architecture requires a specific protocol to grant its services to the entities, and therefore its users.

8.1 The OSI model

Some theoretical models were developed in recent years to explain how network communication works. The OSI (Open Systems Interconnection) model is used to define the communication between systems that adopt a certain set of rules (OSI rules), which are enough to allow for cooperation between said systems and therefore the interconnection and subsequent transmission of information.

The definition of the layers that comprise the OSI architecture is based on the principle of hierarchy. There are seven functional layers:

- The application layer interacts directly with the software applications that require communication services. Application-layer functions typically include identifying communication partners, determining resource availability, and synchronizing communication.
- The presentation layer establishes context between application-layer entities, in which the application-layer entities may use different syntax and semantics if the presentation service provides a mapping between them.
- The session layer controls the dialogues (connections) between computers. It establishes, manages and terminates the connections between the local and remote application.
- The transport layer grants to the session-layer entities a virtual resource for the transparent transfer of data, while also making sure of keeping an appropriate quality of service (QoS), that oscillates and degrades due to the network utilization.
- The network layer interfaces the different sub-networks in such a way that is transparent to the transport layer. The only element available to the upper layer is the quality of service offered by the network layer.
- The data link layer reveals and recovers transmission errors that can happen throughout the physical transfer of data. It also defines the protocol for flow control between nodes of a certain network.
- The physical layer has the main task of performing the physical transmission of data, which has the form of a binary string, between different systems of the network. The transmission happens through a physical transmission medium. Layer specifications define characteristics such as voltage levels, the timing of voltage changes, physical data rates, maximum transmission distances and such.

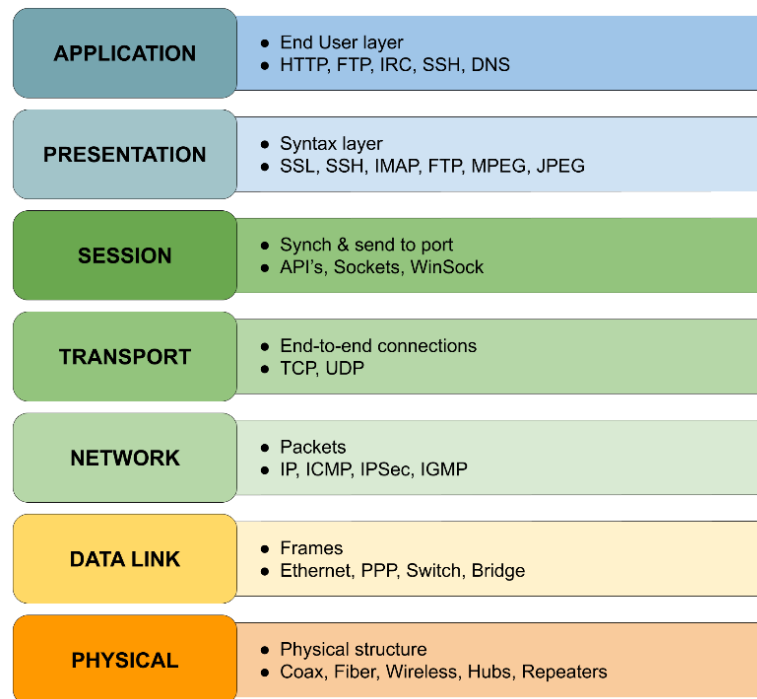


Fig 8.1 - Architecture of the OSI model

Before the real transmission of data through the physical medium can occur, the different layers of the same system must cooperate, both on the transmitting and receiving side.

Therefore, we must define the Service Data Unit (SDU), the payload of each layer, which is obtained by the upper layer entity, and the Protocol Control Information (PCI), a string of binary code that encapsulates the corresponding protocol. The process is shown in the figure below Fig. 8.2, where the SDUs are shown in green, while the PCIs are red. In general, an (N)-SDU is equal to (N+1)-PCI + (N+1)-SDU. This means that the sending process can be seen as an enveloping of the data, while the receiving process is more of an extraction, since each entity removes its PCI bits, instead of adding more.

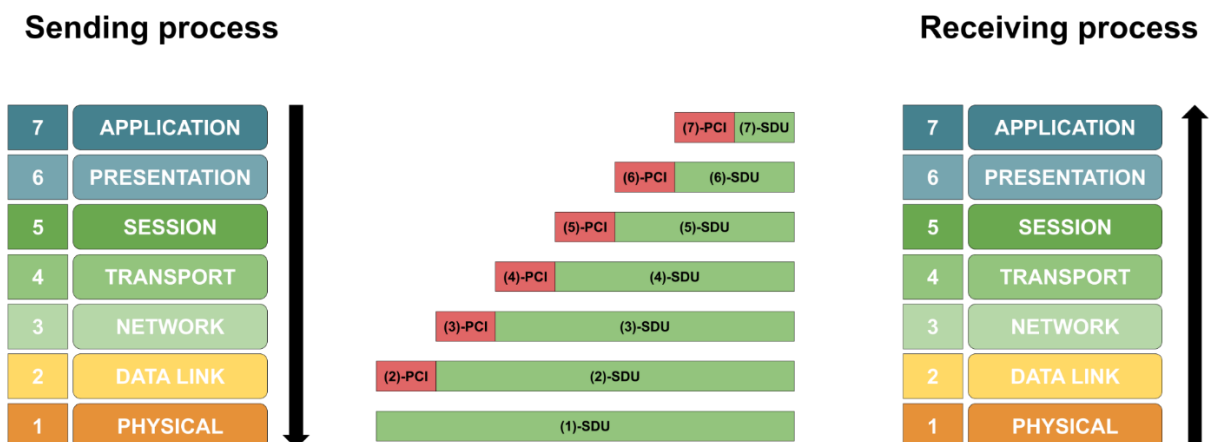


Fig 8.2 - Sending and receiving processes in the OSI model

8.2 The actual interfaces

From the spacecraft scheme we can observe that interfacing is required to link together the components of the CubeSat, since they offer different interfaces and therefore only apply those corresponding protocols. We also do not analyze here the behavior of the upper layers of the systems, while we focus heavily on the lower layers, comprising the transport, network, data link and physical layers.

As it can be seen in Fig 8.3, we have an architecture corresponding to the previously examined OSI model. The two upper layers, nominally the transport and network layers are governed by a single protocol called the CubeSat Space Protocol. It was developed in recent years by the Danish company GomSpace and can be readily used on their products. The CSP also interfaces with a series of other protocols that are commonly found in space architectures. For this reason and since a certain number of components for this project are built by GomSpace and the protocol itself has a flight heritage, it was decided to utilize the CubeSat Space Protocol as the transport and network layer.

The data link layer protocol is determined by the presence of proper connections in the components of the satellite. Most of them, however, can communicate through a CAN or an I2C bus. In these cases, the CAN and the I2C protocols also regulate the physical layer. The KISS protocol was chosen as a data link protocol as it is capable of interfacing with both the CSP and the physical layer standards that are utilized in our satellite project, namely the RS-422, the RS-485 and the UART. Lastly, the AX.25 protocol is only utilized by the UHF transceiver as it is used extensively on amateur packet radio networks.

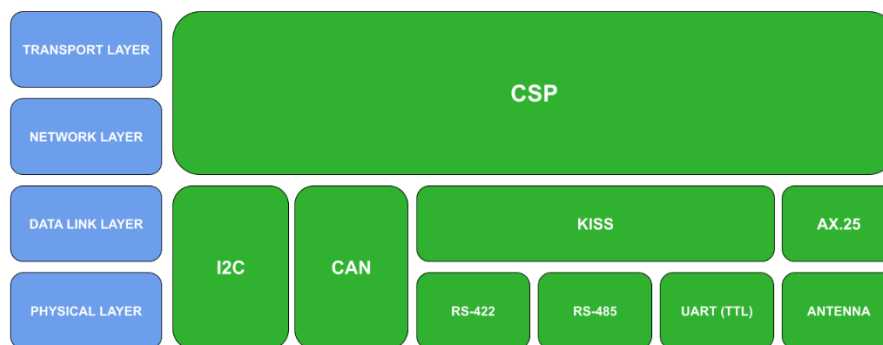


Fig 8.3 - Scheme of the protocols used for the CubeSat architecture

8.2.1 CSP

The CubeSat Space Protocol (CSP) is a network and transport layer delivery protocol expressly designed for CubeSats. Its header size is 4 bytes containing both transport and network information. Its layering corresponds to the same layers as the TCP/IP model. The implementation is compliant with CCSDS standard and supports a connection-oriented transport protocol, a network protocol, and several network interfaces. The physical layer includes several other technologies such as CAN, I2C, RS-232 using the KISS protocol and CCSDS Space Link Protocol. Its implementation is written in C language and can be used on the following operating systems: FreeRTOS, Linux, MacOS and Windows. The CAN Fragmentation Protocol (CFP) can be used together with the CAN protocol to send data that

is larger than the allowable limit. This can be achieved through the fragmentation of the data in different CAN frames.

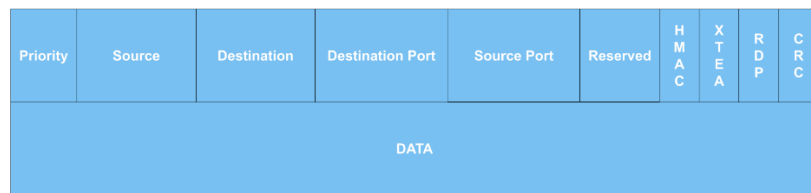


Fig 8.4 - Typical CSP frame

The header of the CSP is composed of

- Priority of the frame
- Source address (0 to 31 available addresses)
- Destination address (0 to 31 available addresses)
- Destination port
- Source port
- CRC (cyclic redundancy check) is used for error detection and can be further used for error correction

$$(CSP)\text{-}PCI = 32 \text{ bits}$$

$$(CSP)\text{-}SDU = (APP)\text{-}SDU + (CSP)\text{-}PCI$$

8.2.2 CAN

The Controller Area Network (CAN) is a communication protocol that encompasses both the data link and physical layer. It is a carrier-sense multiple-access protocol with collision detection and arbitration on message priority (CSMA/CD+AMP). The CAN bus was designed as a multi-master, message broadcast system that specifies a maximum bitrate of 1Mbps. CSMA means that each node on a bus must wait for a prescribed period of inactivity before attempting to send a message. The CAN protocol only requires two wires to transmit data.

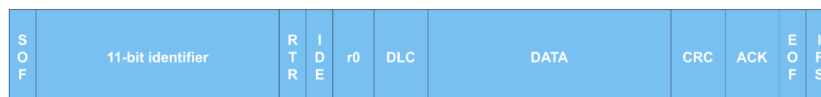


Fig 8.5 - Typical CAN frame

- SOF (start of frame)
- 11-bit identifier establishes the priority of the message (the lower the value, the higher the priority)
- RTR (remote transmission request) identifies the frame as a request if bit is set to 1
- DLC (data length code) is the length of the data field in bytes
- DATA (up to 64 bits of application data may be transmitted in a CAN frame)

- CRC (cyclic redundancy check)
- ACK (acknowledgement bit) is set to 1 if previous message was error-free
- EOF (end of frame)

$$(\text{CAN})\text{-PCI} = 51 \text{ bits}$$

$$(\text{CAN})\text{-SDU} = (\text{CSP})\text{-SDU} + (\text{CAN})\text{-PCI}$$

8.2.3 I2C

The I2C (Inter-Integrated Circuit) is a synchronous, multi-master, multi-slave, serial communication bus. A single-master, multi-slave configuration is decided to be used in the CubeSat scheme. I2C uses only two wires to transmit the data between devices. I2C is a serial communication protocol, so data is transferred bit by bit along a single wire of those two, but it is also synchronous, so the output of bits is synchronized to the sampling of bits by a clock signal shared between the master and the slave.



Fig 8.6 - Typical I2C frame

- Start
- Address (identification of the slave)
- Read/Write bit that specifies if the master is sending data or requesting it
- ACK/NACK bit sent by the receiver every 8 bits of data
- DATA is split along multiple frames each of 8 bits maximum, but it does not have an upper limit to number of bits that can be sent
- Stop

$$(\text{I2C})\text{-PCI} = 11 \text{ bits} + \text{ceil}((\text{CSP})\text{-SDU} / 8)$$

$$(\text{I2C})\text{-SDU} = (\text{CSP})\text{-SDU} + (\text{I2C})\text{-PCI}$$

8.2.4 KISS

The KISS (Keep It Simple, Stupid) is a data link layer protocol for serial end-to-end communication between devices. It can be used alongside different standards for physical layers, such as RS-422 and UART, but does not offer support for flow control or error handling.



Fig 8.7 - Typical KISS frame

- FEND (Frame End) are used for delimitation of the frame
- Command code identifies the class of message. A bit 0 identifies the message as a data frame
- DATA does not have any upper limit for the transmittable data
- FEND

$$(KISS)\text{-PCI} = 48 \text{ bit}$$

$$(KISS)\text{-SDU} = (\text{CSP})\text{-SDU} + (KISS)\text{-PCI}$$

8.2.5 RS-422/RS-485

The RS-422 is a physical standard that specifies data rates up to 10 Mbps and line lengths up to 1220 meters. Similarly, the RS-485 is another standard that specifies the same qualities as the one before. These two standards both use supports multidrop communications links, using the same differential signaling over twisted pair. This also means that they are both much more resilient to common mode interference due to the differential mode of transmission.

8.2.6 UART (TTL)

The UART stands for Universal Asynchronous Receiver-Transmitter. It is not a protocol, nor a standard, it is instead a physical circuit in a microcontroller. There must be then two UARTs, one at each end of the transmission. Its purpose is the very same: to transmit and receive serial data. UART only uses two wires to transmit data between devices. UARTs transmit data asynchronously, which means there is no clock signal to synchronize the output of bits from the transmitting UART to the sampling of bits by the receiving UART.

The Transistor–transistor logic (TTL) is a logic family built from bipolar junction transistors and can be used in different integrated circuits, such as the UART.

Chapter 9

Memory budget

9.1 Memories

The Hyonosat architecture is designed to have five memories, arranged in the following manner:

- 1 TB SATA hard drive

It is the memory already located inside the primary payload sensor, the HyperScout-2. It can store a week worth of data, given the correct configuration. The data acquired from the observation is saved in the hard drive until the end of the last observation, when they are compressed, and a backup copy is saved in the same place. The day before observation data can be removed to free up storage space for the following days. The evolution of the storage throughout a week of normal operations can be seen in fig 9.1. In blue it is the predicted evolution, while the green considers a margin of 20%. It can also be seen that the upper boundary is 750 GB, giving us ample storage for backup and at the same time protection from bad sectors.



Fig. 9.1 - SATA hard drive storage during a week

- 32 GB SD card

An SD card can be inserted into the S-band transmitter to store more data. It is certainly useful to store the processed and then compressed data from the observation to downlink it each time the window of visibility opens. At the end of the week, the saved data is removed from the SD card. The SD card cannot be larger

than 32 GB, since that is the largest density allowable that is reported on the transmitter datasheet. The actual evolution of the storage is reported in blue in fig 9.1 and as before a margin of 20% was considered (the green line).

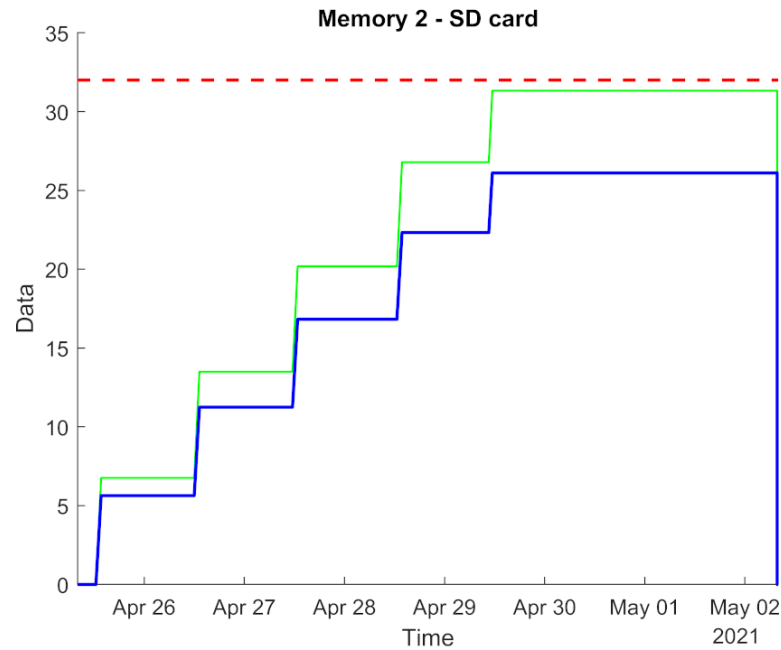


Fig. 9.2 - SD card storage during a week

- **8 GB Flash memory**
A flash memory card was added to the architecture to ensure long redundancy for storage. It can be accessed from the interface board, which can be in turn reached from each on board computer, making sure there is always a backup copy of the data generated by the sensors (telemetry) on board the CubeSat to be downlinked again, if there is need to. Since it is a non-volatile memory, it stores a backup of the running software for eventual reboots. It is also the storage for additional updates for the software. The unit works in a circular fashion, meaning that the oldest data is automatically overwritten when the storage is full.
- 128 MB internal storage of the central OBC
- 128 MB internal storage of the AODC dedicated OBC

Memory	Size	Physical location	stores
SATA drive	1 TB	Hyperspectral camera	payload data (backup)
SD card	32 GB	S-band transmitter	payload data
FLASH	8 GB	accessible from the interface board	telemetry data (backup) + software updates
Internal	128 MB	Central OBC	telemetry and telecommand data

Internal	128 MB	AODC dedicated OBC	attitude data
Total memory	1040.3 GB	distributed	

Table 9.3 – Summary of the available storage on board

The table above is a short summary of all the mass memories modules that are considered in the CubeSat architecture.

9.2 Data budget

SYSTEM	SENSOR	NUMBER	FREQUENCY (Hz)	BITS	DATA RATE (bps)
PAYLOAD	GSS receiver	1	5	64	320
AODC	Magnetometer	1	5	72	360
	Star tracker	1	5	256	1280
	Sun sensor	1	5	64	320
	Gyro IMU module	1	5	128	640
	Magnetorquer	3	5	8	120
	Reaction wheel	1	5	8	40
	Electric Thruster	2	5	8	8
THERMAL	Temperature measurements	30	0,1	16	48
EPS	Voltage measurements	10	0,	16	16
	Current measurements	10	0,1	16	16
	Battery state	15	0,1	16	24
OTHER	Status bits	30	0,1	1	
OBDH	Central OBC	1		194	
	AODC dedicated OBC	1		80	
TOTAL				1976	3267

Table 9.4 – Data budget estimation

We distinguish between two classes of data produced on board the spacecraft:

- **Telemetry**
taken once every 10 seconds, it is collected from all relevant sensors and actuators here above and transferred to the central OBC, to either forward it to the transceiver, if the CubeSat is in a window of visibility, or to store it in the FLASH memory otherwise. It amounts to 1976 bits every time, meaning that more than one year of telemetry data can be appropriately stored at a time, even considering not all of the FLASH memory is granted to the telemetry alone.

- Attitude
taken once every 0.2 seconds, according to the needs of the AODC algorithm for stabilization of the CubeSat. Data is collected from sensors and relayed to the dedicated OBC that runs the algorithm and forwards the proper telecommands to the actuators. It amounts to 648 bits every time.

Class of data	Frequency
Telemetry	0.1 Hz
Attitude	5 Hz

Table 9.5 – Summary of the data generated by the sensors

Chapter 10

Processing power estimation

10.1 Model

The calculus power is a necessary criterion to size the process of the OBC.

The speed of a given CPU depends on many factors, such as the type of instructions being executed, the execution order and the presence of branch instructions. CPU instruction rates are different from clock frequencies, usually reported in Hz, as each instruction may require several clock cycles to complete, or the processor may be capable of executing multiple independent instructions simultaneously. MIPS stands for Millions of Instructions Per Second and is used as a measure to estimate the speed of a CPU that is executing certain instructions.

$$MIPS = \frac{\text{clock rate}}{CPI}$$

CPI (clock cycles per instruction) is an aspect of a CPU's performance: the average number of clock cycles per instruction for a program or program fragment. This value depends on the instruction set and the total number of instructions required to be executed: each instruction will necessitate a different number of clock cycles to be finalized. In this report we will consider the frequency of the instructions instead.

$$\begin{aligned} CPI &= \frac{\sum_i n. \text{instructions}(i) \times \text{clock cycle count}(i)}{\text{total } n. \text{instructions}} = \\ &= \sum_i \text{instruction frequency}(i) \times \text{clock cycle count}(i) \end{aligned}$$

The processing time (or execution time) is then measured by calculating the number of instructions per second first. This calculation will depend on the instruction set and the speed of the clock.

$$\text{execution time} = \frac{\text{total } n. \text{instructions} \times CPI}{\text{clock rate}}$$

10.2 Central OBC

The central OBC performs a series of tasks like housekeeping, and the managing of telemetry and telecommand. It also plays a major role into saving the housekeeping data in

the appropriate memories. The instruction set for the central unit is specified in Table 10.1 and considers the frequencies of these tasks as well as the number of clock cycles required to run one instruction of that kind. The managing of the data requires a higher clock cycle count than other applications, while the most demanding arithmetic instructions involve matrices.

The NanoMind A3200 CPU runs at 32 MHz typically, while at most 64 MHz is achievable. In this analysis we considered the clock rate to be around 48 MHz, with small variations added to simulate a realistic scenario of application. It is to be noted that even at the nominal speed the CPU can manage the tasks requested from it.

Instruction type	Instruction frequency	Clock cycle count
Load/store data	0.4	6
Arithmetic instructions	0.5	4
All others	0.1	3

Table 10.1 – Instruction set for the central OBC

The CPI is estimated using the previous formula and returns a value of 4.7 cycles/instructions. The total number of instructions for the central OBC is set to $1e+06$.

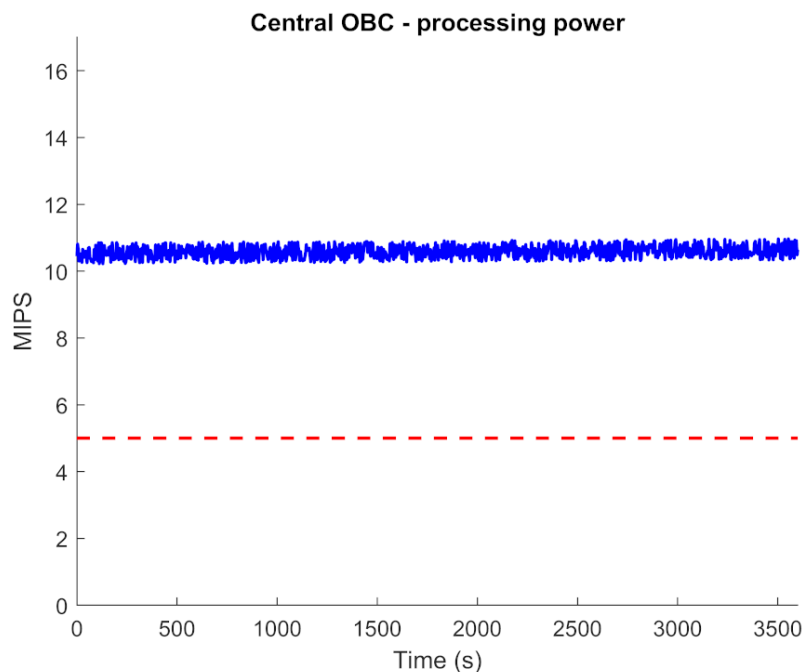


Fig. 10.2 – Estimated calculus power for the central OBC

The results from the model implementation are shown in Fig. 10.2, where the red line represents the value of MIPS required by the CPU to run ensuring the execution time is no more than 0.2 seconds, therefore giving ample time between subsequent instances of the

telemetry exchange data. The blue line represents instead the MIPS offered by the OBC when the clock rate is set to 48 MHz and has a value of around 10.2 MIPS. The processing power offered by the NanoMind A3200 is consequently enough to handle the request from the system, as it never dips under the red line, set at 5 MIPS.

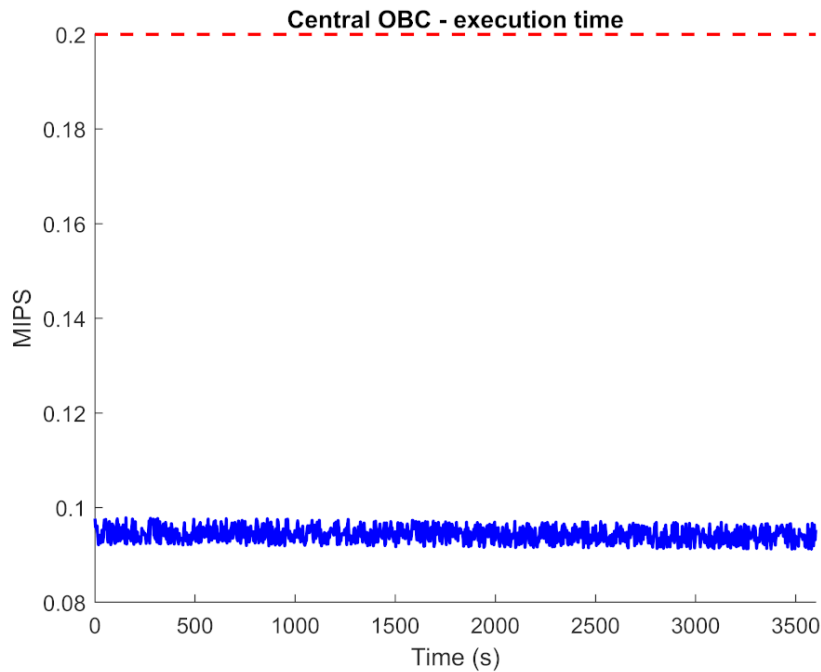


Fig 10.3 – Estimated execution time for the central OBC

The corresponding execution times are reported in Fig. 10.3, where both the upper limit established before and the value offered by the OBC are shown. In blue, we have variations of time depending on the workload of the central unit processor that were estimated as spikes around a constant value of 9.79×10^{-2} seconds. The processing time is smaller than the limit, meaning that the A3200 can run its tasks in time.

10.3 AODCS Dedicated OBC

The instruction set of the dedicated OBC is a little different from the previous one, since this computer focuses mainly on the tasks that regard the AODC system, meaning that it needs to run the attitude algorithms more times a second to perform correctly. The frequencies for the instructions were therefore changed to reflect this variation. The clock cycle count remains the same.

Instruction type	Instruction frequency	Clock cycle count
Load/store data	0.3	6
Arithmetic instructions	0.6	4
All others	0.1	3

Table 10.4 – Instruction set for the central OBC

The CPI returns a value of 4.5 cycles/instruction, while the total number of instructions is now set at $1e+05$.

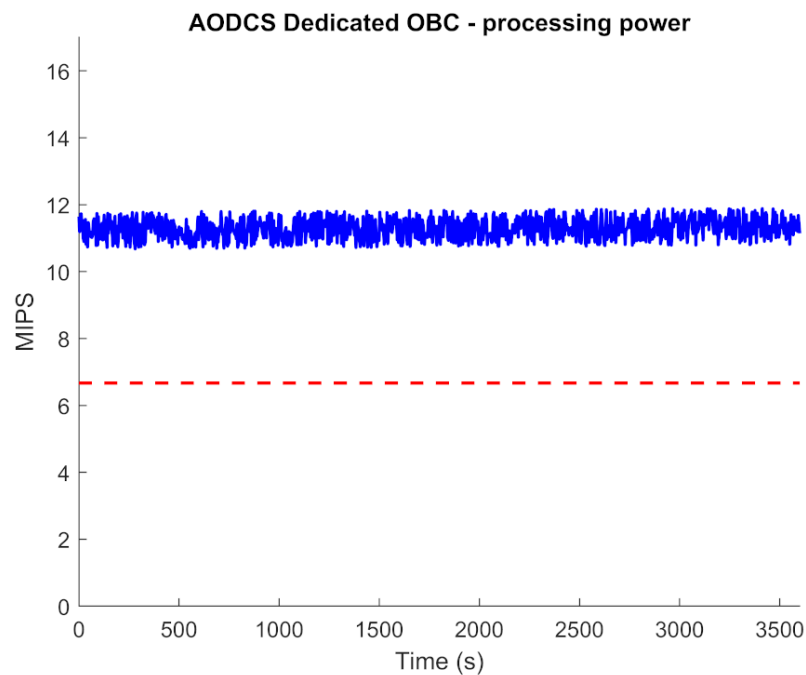


Fig. 10.5 – Estimated calculus power for the dedicated OBC

The estimated calculus power is shown in the above Fig. 10.5. We can see clearly that even this time the A3200 does not have a problem running the tasks that are requested. The requested calculus power is 6.67 MIPS, while the value from the actual OBC is around 10.67 MIPS.

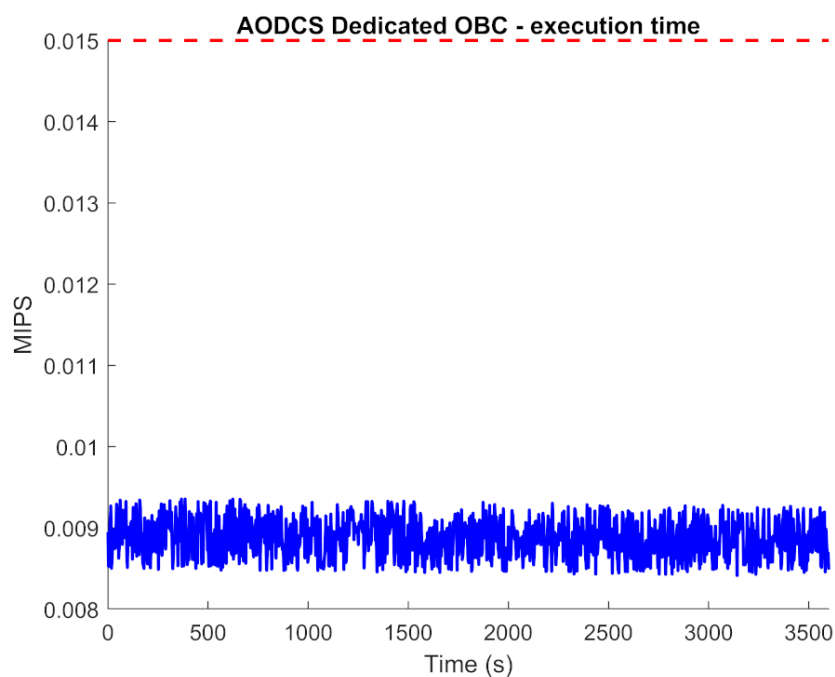


Fig 10.6 – Estimated execution time for the dedicated OBC

The execution time for the dedicated OBC is shown in Fig. 10.6 and exhibit an actual processing time that is less than the one required to perform correctly, set at 0.015 seconds to not interfere too much with the sampling rates of the AODCS components.

10.4 Insights

The analysis was also expanded on more general terms, without constructing an instruction set, and therefore the results do not depend on which OBC performs those tasks. In the report we consider the CPI, instead of an instruction set, and its effects on the MIPS.

The NanoMind A3200 usually runs at 32 MHz, but at peaks it can also achieve 64 MHz. It is generally not a good idea to overclock the CPU, as this will shorten greatly the life of the processor, and thus of the entire component. 4 different clock rates were therefore considered (32, 48, 56 and 64 MHz) and the trend of the resulting MIPS was plotted against increasing values of the CPI (ranging from 2.5 to 6.5 cycles/instruction) in Fig. 10.7. In the previous graphs the resulting CPI was in the range of 4.5, and we can clearly see that even small changes of CPI around that value causes almost no difference in the resulting MIPS.

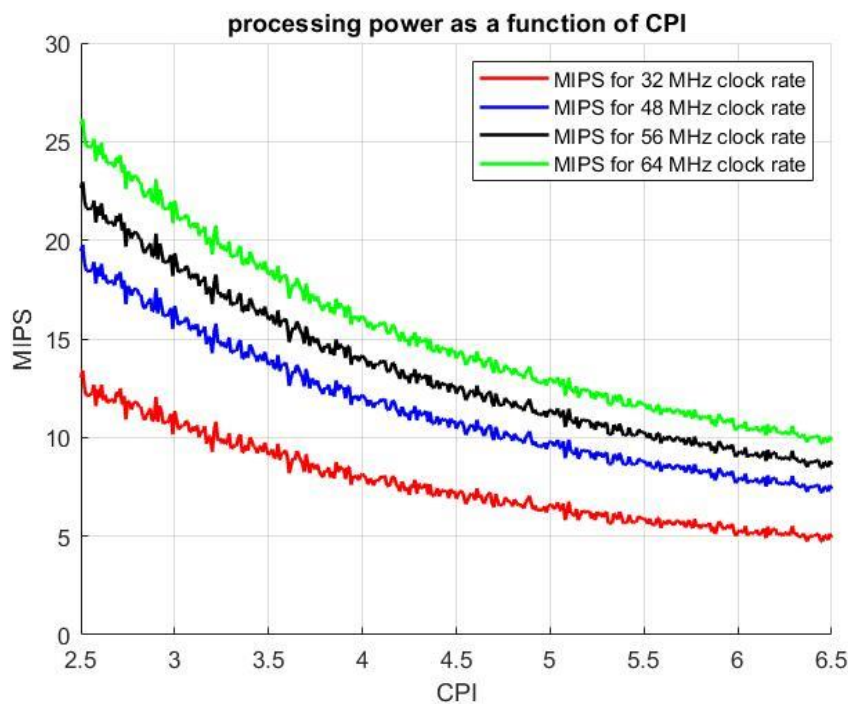


Fig 10.7 – Processing power vs CPI

Chapter 11

References

- [1] Website https://www3.ntu.edu.sg/crest/XSAT_OBDH.htm
- [2] “State-of-the-Art: Small Spacecraft Technology”, NASA/TP-2020–5008734
- [3] GomSpace, “NanoMind A3200 datasheet”
- [4] GomSpace, “NanoDock DMC-3 datasheet”
- [5] Eurotech, “COM-1274 datasheet”
- [6] DDC, “69F64G16 datasheet”
- [7] Razzaghi E., “Design and Qualification of On-Board Computer for Aalto-1 CubeSat”
- [8] Lumbwe L.T., “Development of an onboard computer (OBC) for a CubeSat”
- [9] Eickhoff J., “Onboard Computers, Onboard Software and Satellite Operations: An Introduction”, Springer
- [10] Lamichhane K., Kiran M., Kannan T., Sahay D., Ranjith H.G., Hegde S.R. and Sandya S., “Operational Flow of Twin Satellite Mission”, 2015 IEEE Metrology for Aerospace
- [11] Bhat M.H., Bhat P.A., Gulur D.R. and Nanda R., “Operation of Onboard Data Handling System in Different Switching Modes for RVSAT-1”
- [12] ECSS, “Spacecraft on-board control procedures”, ECSS-E-ST-70-01C (16 April 2010)
- [13] GomSpace, “NanoDock P60 datasheet”
- [14] Website <https://cfs.gsfc.nasa.gov/Features.html>
- [15] Website <https://www.freertos.org/>
- [16] Arseno D., Edwar E., Harfian A. R. and Salsabila J. N., "Characterization of On-Board Data Handling (OBDH) Subsystem," 2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications (TSSA), 2019, pp. 223-227, doi: 10.1109/TSSA48701.2019.8985466
- [17] Prasad A., Jain Y., Joshi N., Gupta N., Singhanian V. and Sreedharan Y., "Interfacing Architecture between Telemetry and On-Board Computer for a Nanosatellite," 2020 IEEE Aerospace Conference, 2020, pp. 1-6, doi: 10.1109/AERO47225.2020.9172773.
- [18] Roveri A., “Retematica Vol. 1”, Sapienza University of Rome, INFOCOM department
- [19] GomSpace, “CubeSat Space Protocol (CSP)”
- [20] Corrigan S., “Introduction to the Controller Area Network (CAN)”, Texas Instruments SLOA101B
- [21] ECSS, “CANbus extension protocol”, ECSS-E-ST-50-15C (1 May 2015)
- [22] Website <https://en.wikipedia.org/wiki/I%C2%B2C>

- [23] Website <http://www.ax25.net/kiss.aspx>
- [24] Marais H., "RS-485/RS-422 Circuit Implementation Guide", Analog Devices, AN-960
- [25] Cypress, "Universal Asynchronous Receiver Transmitter (UART)", Document Number: 001-65468
- [26] Pincioli R., Yang L., Alter J. and Smirni E., "The Life and Death of SSDs and HDDs: Similarities, Differences, and Prediction Models"
- [27] Website <https://www.ednasia.com/on-board-mass-memory-requirements-for-the-new-space-age/>
- [28] P. Gaudenzi P., Marzano F. and Morelli G., "CDF Study Report FLORAD", Sapienza University of Rome, Master in satellite systems and services
- [29] Website [https://en.wikipedia.org/wiki/Instructions_per_second#Millions_of_instructions_per_second_\(MIPS\)](https://en.wikipedia.org/wiki/Instructions_per_second#Millions_of_instructions_per_second_(MIPS))
- [30] Website https://en.wikipedia.org/wiki/Cycles_per_instruction
- [31] Hanafi A., "Etude et Conception du Segment Spatial du Nanosatellite Universitaire MASAT1 avec Ordinateur de Bord Secondaire Reconfigurable a Base de FPGA", Faculty of Sciences, University of Fes
- [32] Website <https://www.d.umn.edu/~gshute/arch/performance-equation.xhtml>
- [33] De Melo A.C., "A Benchmark for Assessing Nanosatellite On-Board Computer Power-Efficiency and Performance", Faculty of Technology, University of Brasilia