UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

TREBALL DE FI DE GRAU

TÍTOL DEL TFG: On-Board Computer software and FlatSat testing for the 3Cat-4 CubeSat mission

TITULACIÓ: Grau en Enginyeria de Sistemes Aeroespacials

AUTOR: Albert Morea Font

DIRECTORS: Hyuk Park, PhD.
Lara Fernández

SUPERVISOR: Joan A. Ruiz-de-Azua, PhD.

DATA: September 2, 2021

Títol: Programari per l'ordenador de a bord i comprovacions en FlatSat per la missió Cubesat 3Cat-4

Autor: Albert Morea Font

Directors: Hyuk Park, PhD.
Lara Fernández

Supervisor: Joan A. Ruiz-de-Azua, PhD.

Data: 2 de setembre de 2021

Resum

Els avenços tecnològics dels últims temps han permès la miniaturització de components així com l'aparició de l'estàndard CubeSat ha resultat en la creació d'instrumentació sense necessitat d'adaptació a la plataforma. Una nova indústria, anomenada NewSpace, ha nascut d'aquestes dues premisses, on les barreres d'entrada tant en l'àmbit econòmic com tècnic són més baixes que en la indústria espacial tradicional, coneguda com a OldSpace.

En el marc del Newspace, el centre de recerca de la UPC-BarcelonaTech, Nanosatellite and Payload Laboratory, conegut com a NanoSat Lab, desenvolupa missions CubeSat especialitzant-se en experiments per a observació de la terra, tals com reflectometria i radiometria.

Una d'aquestes missions és l'anomenada 3Cat-4, un nanosatèl·lit que segueix l'estàndard CubeSat amb una estructura d'una unitat. 3Cat-4 es desenvolupa sota el paraigües del projecte "Fly your satellite! II"de l'Agència Espacial Europea, qui proveïx de suport tècnic durant tot el desenvolupament així com patrocina el llançament.

Aquesta tesi contribueix en el desenvolupament de la missió 3Cat-4, específicament en el desenvolupament de programari. Per una banda, es contribueix en el programari de vol, encastat en l'ordinador de a bord del nanosatèl·lit. Les aportacions més rellevants són la integració dels diferents subsistemes a l'ordenador de a bord, la creació d'una capa per a gestionar els possibles errors i la implementació d'una lògica d'estats que dota al sistema de més autonomia per autogestionar-se. Per altra banda, es contribueix implementant un sistema de visualització de dades del que permeten el control durant les fases de testatge i desenvolupament i durant la fase d'operacions, ja en òrbita.

Finalment, les implementacions de programari realitzades es verifiquen realitzant tests en configuració FlatSat, on en aquesta tesi s'en destaquen els més rellevants. Aquest testatge permet validar el sistema tant en l'àmbit de programari com en l'àmbit de la maquinària.

Les conclusions d'aquesta tesi son la importància de produir un programari robust que permeti controlar el estat del sistema en tot moment i de forma autònoma, a causa de la forta limitació que suposa no poder accedir al nanosatèl·lit en qualsevol moment. També es destaca la importancia de la realització del testeig adient per cobrir tots els possibles casos i caracteritzar el comportament del sistema.

Title : On-Board Computer software and FlatSat testing for the 3Cat-4 CubeSat mission
Author: Albert Morea Font

Advisors: Hyuk Park, PhD.
Lara Fernández

Supervisor: Joan A. Ruiz-de-Azua, PhD.

Date: September 2, 2021

Overview

Last times technological breakthroughs allowed miniaturization of components and combined with the introduction of the CubeSat standard resulted in the creations of components without the need of platform customization. A new industry, known as NewSpace, emerged from these two premises, where the entry barriers both from an economical and technical point of view are lower than the traditional space industry, known as OldSpace.

In the frame of NewSpace, the research center from UPC-BarcelonaTech, Nanosatellite and Payload Laboratory, known as NanoSat Lab, develops CubeSats missions specializing in Earth Observation payloads, such as reflectometry and radiometry.

One of these missions is the one known as 3Cat-4, a nanosatellite that follows the standard of 1 unit. 3Cat-4 develops under the scope of the project "Fly Your Satellite! II" from the European Space Agency. The agency provides support during the development of the mission as well as sponsors the launch.

This thesis contributes in the development of the 3Cat-4 mission, specifically in the development of software. On one side, it contributes to flight software, which runs embedded in the on-board computer of the nanosatellite. The most relevant contributions are the integration of the different subsystems onto the on-board computer, the creation of an error management layer and the implementation of a state logic that provides the system more autonomy for self-management. On the other side, it contributes implementing a satellite data visualization system that allow to control the satellite during the testing and developing phase as well as the operations phase, in orbit.

Finally, the software implementations done are verified performing test in FlatSat configuration. The most relevant ones are included in this thesis. This testing allows to validate the system both from a software and hardware point of view.

The conclusions of the thesis are the importance of developing robust software that allows the control of the system's state in every moment and autonomously from ground, due to the important constraint of not being able to access the spacecraft anytime. Also, it is important to perform the required testing in order to cover all the possible cases and to characterize the system's behavior.

Aim higher

CONTENTS

LIST OF FIGURES

GLOSSARY

ADCS Attitude Determination and Control Subsystem. xiii, xiv, 9, 11, 13, 18, 19, 28–30, 33, 39, 41, 50–52, 57, 58, 71–75

AIS Automatic Identification System. 3, 4, 12

ATC Ambient Test Campaign. 58

COMMS Communications Subsystem. 9, 11, 13, 41, 50, 57

COTS Commercial off-the-shelf. 2, 3, 10–13, 24, 33

DC-to-DC Direct Current to Direct Current. 30, 31

EGSE Electrical Ground Support Equipment. 43, 50

EO Earth Observation. 1–3, 12

EPS Electric Power Subsystem. xiii, xiv, 9–11, 13, 18, 19, 24–26, 30, 41, 42, 44–48, 50–52, 57

ESA European Space Agency. 2, 3, 18, 58

ESEC European Space Security and Education Centre. 58

ETC Environmental Test Campaign. 58

FFT Full Functional Test. 58

FMPL-1 Flexible Microwave Payload 1. 3, 9, 11–13, 34, 41

FSS Federated Satellite System. 3

GNSS Global Navigation Satellite System. 4

GNSS-R Global Navigation Satellite System - Reflectometry. xiii, 3, 4, 12

GPIO General Purpose Input/Output. 31–34, 57

I2C Inter-Integrated Circuit (Serial Communication Bus). 6, 7, 11, 13, 24, 26, 29, 30, 42

ICGC Institut Cartografic i Geologic de Catalunya. 3

IEEC Institut d'Estudis Espacials de Catalunya. 3

IGRF International Geomagnetic Reference Field. 28, 50

ISO International Organization for Standardization. 43

LEO Low Earth Orbit. 43

LEOP Launch and Early Orbit Phase. 15, 16

LNA Low Noise Amplifier. 11

MGSE Mechanical Ground Support Equipment. xiii, 43–45

MT Mission Test. 58

NADS Nadir Antenna Deployment Subsystem. xiii, xiv, 9, 11–13, 15, 30–34, 41, 52–55, 57

NASA National Aeronautics and Space Administration. 2, 18

OBC On-Board Computer. 9–11, 13, 16, 21, 23, 24, 29–31, 33, 34, 41–43, 45, 47, 50, 52

OBDH On-Board Data Handling. 11, 18, 19

OISL Optical Inter-Satellite Link. 3

PA Power Amplifier. 11

PCB Printed Circuit Board. 31, 52

PID Proportional-Integral-Derivative. 11

SDR Software Defined Radio. xiii, 3, 12, 38

TLE Two Line Element Set. 28–30, 57

TTC Telemetry, Tracking & Command. xiii, 18, 19, 26–28, 30

TVAC Thermal and Vacuum Chamber. xiii, 6, 43–45, 55

UART Universal Asynchronous Receiver-Transmitter (Serial Communication Bus). 6, 7, 11, 13, 34, 42

UHF Ultra High Frequency. 11, 12, 26, 38, 49

VHF Very High Frequency. 4, 12, 26, 38

ZADS Zenital Antenna Deployment Subsystem. 9, 12, 13, 15, 30, 33, 34, 41

INTRODUCTION

Nanosatellites and the CubeSat Standard

Since the early launch of Sputnik in 1957, the satellite industry has experienced considerable changes and improvements. First, the predominant trend was to build and launch bigger satellites year after year. This increase in size resulted in an increase of the power and mass budgets, allowing to perform several experiments and functions in orbit. One of the most memorable ones is the Hubble Space Telescope, launched in 1990 with a total weight of 1110 kg. Nonetheless, one of the turning points in this evolution is the production of miniaturized electronic components and systems thanks to technological breakthroughs. Moreover, another key aspect regarding satellites size is the relation between payload mass and launch cost, which is direct and proportional. As a consequence, in the last few years a huge step has been made on the development of smaller satellites (Figure 1).

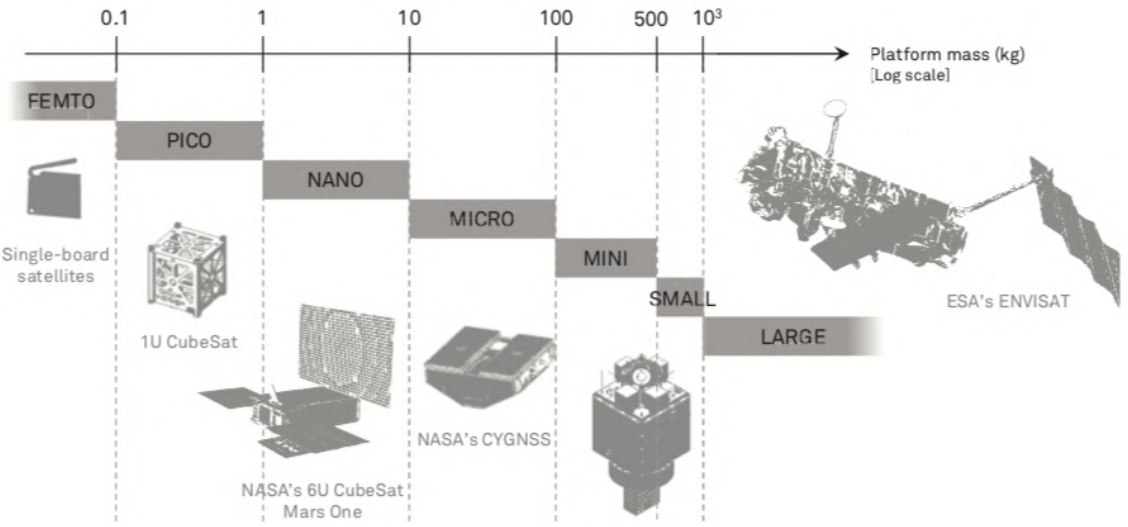


Figure 1: Satellite classification in function of mass [1].

The satellites with a weight comprised between 1 to 10 kg, categorized as nanosatellites or nanosats, have been a successful model of compromise solution for many use cases, like Earth Observation (EO). Specifically, the CubeSat standard has been the most relevant type of nanosat.

The CubeSat specifications [4] were defined in 1999 by two academia professors: Jordi Puig-Suari, from California Polytechnic State University and Bob Twiggs, from Standford University. The purpose of these specifications was to enable students to participate in the design, build, test and operations of a space mission within a short time range. The specifications were the result of a simplification and improvement of prior nanosatellites designs from several initiatives. The CubeSat became a de-facto standard when their specifications were adopted by other space missions after the launch of the first CubeSat mission in June 2003.

The CubeSat specifications are a set of key points that define high-level design goals.

The most relevant aspects are the standardization of size, weight and envelope of the spacecraft. The CubeSat envelope is organized by units, being a unit (named 1U CubeSat) a cube with a 10 cm edge. Linked to these dimension constraints, there is a second constraint of total weight up to 1.33 kg per unit. From this point, several combinations could be made in function of mission requirements, ranging from 0.5U up to 27U. Nonetheless, the most common envelopes are from 1 to 12 units, depicted in Figure 2.

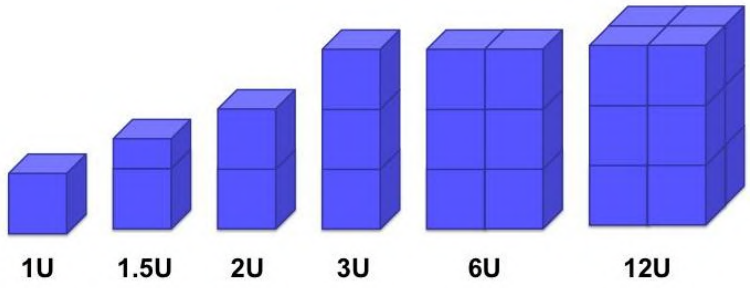


Figure 2: Most common CubeSat envelopes [2]

The standardization of size, weight and envelope of CubeSats has many benefits. One of them is the facilitation of the deployment system, allowing it to be generalist for any kind of mission that meet the specifications. Furthermore, the standard allows space companies providers to mass-produce components and offer them as a commercial off-the-shelf (COTS) component. This relates directly to time and cost of the mission, which are reduced as a consequence of the absence of highly customized interfaces and platforms in order to test, transport, launch and deploy the satellites.

The success of the CubeSat standard can be justified with its adoption by several space institutions. Until 2013, only academia was experimenting and using CubeSats platforms as experiment platform and technology demonstrators, but in the last few years several private initiatives adopted the standard. One of the best examples is Planet Labs, who operates an EO CubeSats constellation formed by more than 200 satellites. Planet Labs 2020 annual revenue was $113 million and is expected to keep growing year by year [5]. Also, governmental institutions such as National Aeronautics and Space Administration (NASA) [6] and European Space Agency (ESA)[7] started to launch CubeSat missions with several purposes this last decade. One of this missions is from the NanoSat Lab, named FSSCat [8], a two 6U satellite constellation that provides support in EO capabilities to the ESA's Sentinel mission.

UPC - NanoSat Lab Missions

This final degree thesis is being pursued in the Nano-Satellite and Payload Laboratory of the Technical University of Catalonia, known as UPC-NanoSat Lab [9] . This cross-departmental initiative aims at designing and developing nanosatellites missions and payloads. The lab is focused on the research of innovative small spacecraft systems and specifically in EO subsystems and payloads.

The 3Cat (pronounced cube-cat) family are CubeSat missions developed in the NanoSat

Lab. All the missions follow the same structured nomenclature system of 3Cat-X, being X the mission number.

1. **3Cat-1[10]:** This 1U CubeSat was the first ever entirely designed in Catalonia. Its main purpose was to be a technology demonstrator using mainly components off the shelf and up to seven different payloads. Although it was finished by 2014, the launch was in November 29th, 2018 due to several issues during the launch campaign.

2. **3Cat-3[11]:** This 6U CubeSat is a study conducted under the demand of the Institut Cartografic i Geologic de Catalunya (ICGC). Its main purpose s to analyze the feasibility of a small-satellite philosophy. For now, the project is in stand-by.

3. **3Cat-4[12]:** This 1U CubeSat is the fourth mission from the NanoSat Lab. Awarded by ESA Fly Your Satellite! program, mounts three different payloads and an innovative deployable antenna system. This thesis is based on the 3Cat-4 mission so on the following pages the mission and its objectives will be explained in-depth.

4. **3Cat-5A/B[8]:** The fifth NanoSat Lab mission and also known as FSSCat, consist in an innovative concept of two federated 6U CubeSat. This program is the winner of the Copernicus Masters call to complement the Sentinel constellation for EO. Each satellite carries a payload, being one of them a GNSS-R and a L-band radiometer and the other a multi-spectral orbital payload. The main objectives are to measure soil moisture, ice extent, ice thickness and to detect melting ponds over ice. It is also a technology demonstrator for an Optical Inter-Satellite Link (OISL) and a Federated Satellite System (FSS).

3Cat-4 Mission

3Cat-4 is a 1 unit (1U) educational CubeSat mission. It is under the development of the UPC-NanoSat Lab in collaboration with Institut d'Estudis Espacials de Catalunya IEEC and member of the ESA's program "Fly your satellite! II".

This thesis is part of the development of 3Cat-4 mission. As a consequence, the whole mission is explained in detail in contrast to the others. All the aspects covered in this work are strictly related to the development and testing of this specific mission.

Scientific objectives

The main objective is to demonstrate the capabilities of using nanosatellites for challenging EO applications. The satellite is equipped with the in-house developed payload named as Flexible Microwave Payload 1 (FMPL-1). FMPL-1 combines three different instruments in a single board: an Automatic Identification System (AIS) receiver; a L-band radiometer; a Global Navigation System - Reflectometer (GNSS-R). All of them are executed in the same Software Defined Radio (SDR) COTS component, powered by a Linux computer.

The three different experiments are explained hereunder:

- **Automatic Identification System (AIS) receiver**

  The Automatic Identification System (AIS) operates in the maritime Very High Frequency (VHF) band (between 30-300 MHz) and enables the wireless exchange of navigation status between vessels. The broadcast messages include the vessel's name, course, speed and current navigation status. Having this receiver as a payload for the mission allows to receive AIS messages from vessels that are far from land, where there is a fixed structure of AIS receivers.

- **L-band radiometer**

  An L-band radiometer is an antenna that receives Earth power radiation at L band [3] (1.5 - 2.7 GHz). This measurements can be processed to obtain several environmental parameters such as soil moisture, sea surface salinity, snow density and vegetation optical depth.

- **GNSS-Reflectometer (GNSS-R)**

  The GNSS-R technique consists of measuring an original GNSS signal in its way to Earth and also its reflection on Earth surface [3]. Comparing the variations between both signals, which theoretically are the same, it is possible to infer properties of the reflection surface. Using this technique it is possible to obtain environmental parameters relevant for research in altimetry, oceanographic wave height and wind speed, cryosphere monitoring and soil moisture monitoring.

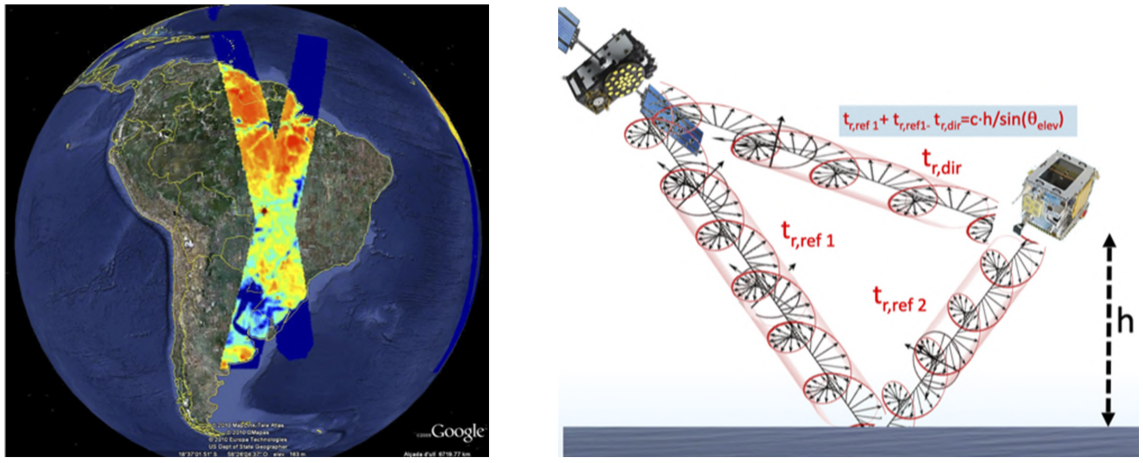


Figure 3: Sample of a radiometry measure (left) and operational principle of GNSS-R (right)

Some of the places targeted for data recollection by this mission are the dry lands of Nile river and the forest zones of Congo river, both in Africa. Furthermore, the ocean winds are also targeted as this payload allows the monitoring of calm and agitated ocean regions, such as the coast of South Africa. Moreover, it is planned also to obtain data over the Himalayas region, as a contribution on cryosphere studies.

Educational objectives

The 3Cat-4 mission is conducted entirely at UPC-NanoSat Lab. The whole team is composed by students from different engineering backgrounds, such as telecommunications, electronics and aerospace engineers, ranging from different educational levels from degrees to doctoral students. The main educational objective is to provide a significant experience and knowledge to promote qualified future professionals in these fields.

Document's structure

This document is structured in three chapters. In these, the first one covers the satellite architecture, then chapter two covers software development and finally chapter three covers relevant tests performed. These three chapters represents the body of the work.

Motivations

One of the most important aspect of a space mission is the software running behind the system. This software is the one responsible of running all the processes and tasks, having the entire mission dependence and even human lives in its bits. For this reason, how this software is designed and implemented is a crucial point.

During the last twenty years of XX century, several space missions have failed due to tiny and apparently insignificant software bugs, such as a variable overflow. The implementation of real time operating systems and highly revised code conventions decreased the number of fatal flaws in space missions for these last years, implementing code with lots of flight heritage and robustness.

Nonetheless, there is always room to improve. For instance, a few days before writing this lines, in July 29th, the Nauka module safely reached the ISS and proceed to the docking procedure. About three hours later, the thrusters of the Russian module were powered on, unexpectedly. This caused a loss of attitude control of the complete International Space Station spacecraft for about 45 minutes. Roscosmos concluded that the fault was a software glitch that almost put in serious danger the twenty years old space laboratory.

This project aims at designing and implementing a reliable and autonomous flight software solution for the 3Cat-4 mission. In this mission, although most of the subsystems software was already implemented, it lacked a software implementation that safely manages all these subsystems and: (1) ensures a proper communication between them and the on board computer; (2) correctly processes and crafts its data in packets ready to be send to ground; (3) safely manages the errors and potential contingency scenarios and (4) have a certain autonomy to perform in nominal condition without the constant monitoring from ground.

Incidents like the before mentioned evidence that there is still much way left to work on reliable flight software. As an aerospace systems engineer who is working in a thesis related to flight software development, this encourages me to keep working.

Objectives

In order to achieve the last section premises, the main specific objectives of the project are stated below:

- Contribution to the flight software development of the 3Cat-4 mission.
  - Implement an error management layer to the system.
  - Implement new specific subsystem functionalities for a better management of the system.
  - Implement or integrate a visual solution to improve the operation of the mission.
- Conduction of tests using the FlatSat configuration, in parallel with the software development.
  - Test the error management layer implemented.
  - Test all the previous and new functions of the system.
  - Test the new visual solution for operations.

Nonetheless, as an engineering student and a near-future professional, I would like to express my personal objectives and skills that I expect to acquire while working on this thesis.

First, there are skills derived from working in the different aspects of the project. The most important ones are stated hereunder:

- Learn how to code in C/C++ languages.
- Learn how embedded systems and real time operative systems work, main features and main constraints.
- Learn serial communications protocols such as I2C and Universal Asynchronous Receiver-Transmitter (UART), used in the avionics of the satellite.
- Learn how to work with flight hardware: precautions, handling and storage.
- Learn how to operate the testing facilities, such as the Thermal and Vacuum Chamber (TVAC) and the shaker.
- Learn how to work in a cleanroom environment facility.

Furthermore, there is a set of soft skills that I would like to train during the development of this thesis, which are the following:

- Work in a real, long-term and ambitious project with a multidisciplinary team of engineers.
- Work in a space-industry project subject to several requirements and restrictions from all the parts involved.

- Learn how the schedule is affected by several parameters and events, internals and externals.
- Learn from other engineering disciplines such as telecommunications, electronics and mechanics as a consequence of collaborating with team members tangentially.
- Develop the ability to work in teams, relying on each member into pursuing a shared achievement.

Methodology

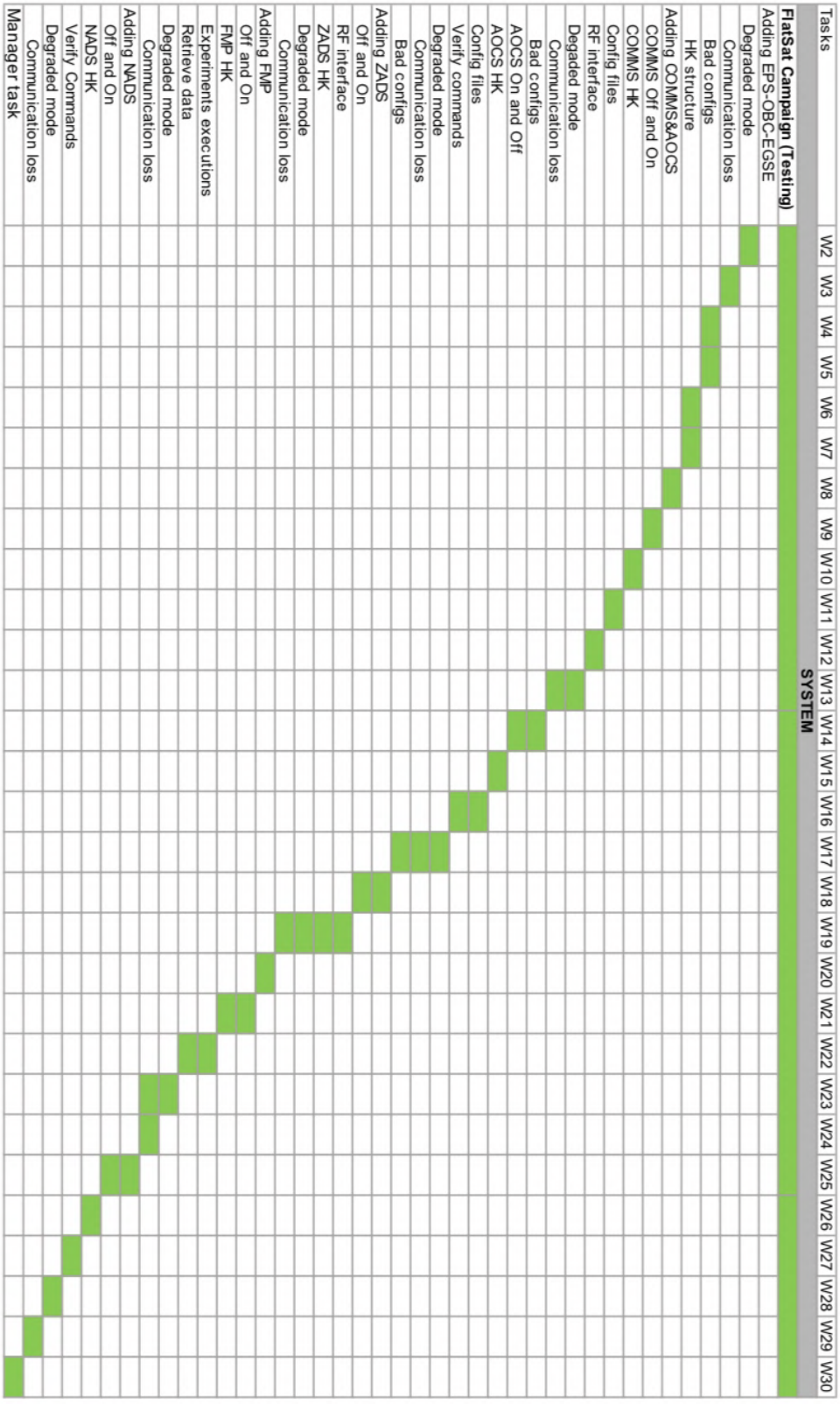
The methodology followed in the development of this thesis is formed by a first period of documentation and comprehension of the already implemented code for the mission as well as the mission as a whole. Once the work to be done was clear and I was more familiar with most of the concepts, a sequential set of steps were iterated over every software task developed and linked to a subsystem of the spacecraft.

The steps followed can be checked below:

1. Definition of the expected inputs and outputs of the subsystem.
2. Integration of the subsystem to the controlled testing configuration.
3. Implement communications interface between the subsystem and the on-board computer using I2C or UART protocols.
4. Retrieve status data from each subsystem that compose the satellite.
5. Code the specific subsystem to on-board computer based functions.
6. Add a layer of error managing to ensure non-blocking scenarios between tasks.
7. Test and validate the new implemented functions and the global functioning of the system.
8. Display in the ground segment infrastructure the status of the satellite.

Gantt diagram

This thesis has been developed during 29 weeks, from the 2nd week of 2021 until the 30rd week. In the following Gantt diagram, the evolution of the work done can be observed:



CHAPTER 1. 3CAT-4 SATELLITE ARCHITECTURE

1.1 Satellite stack

This section aims at explaining which components and modules form the stack of the spacecraft, with an in-depth explanation about which are its role in the system.

The spacecraft is formed by seven subsystems/modules. In Figure 1.1, an exploded view of the 3Cat-4 satellite can be observed. Each part is labelled with a number and identified below.

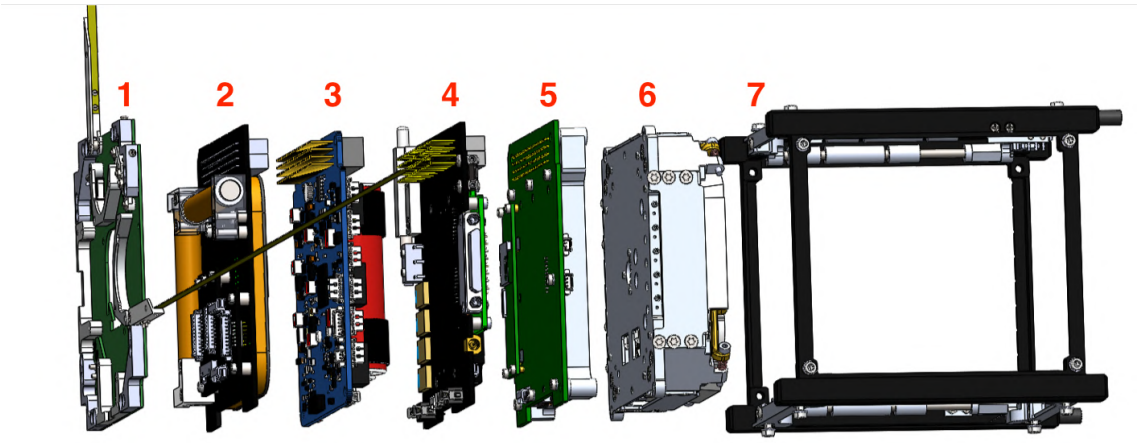


Figure 1.1: Rendering of the exploded view of 3Cat-4 satellite.

Starting from the left, there is the top face. The components are stacked as seen in the 3D model, detailing hereunder each number to the component:

1. Zenith Antenna Deployment System (ZADS).
2. Communications and Attitude Determination and Control System board (COMMS&ADCS).
3. Electrical and Power Subsystem (EPS).
4. On-Board Computer (OBC).
5. Flexible Microwave Payload 1 (FMPL-1).
6. Nadir Antenna Deployment System (NADS).
7. 1U CubeSat structure.

Figure 1.2 shows how the final stack of the spacecraft looks like, without including the solar panels.

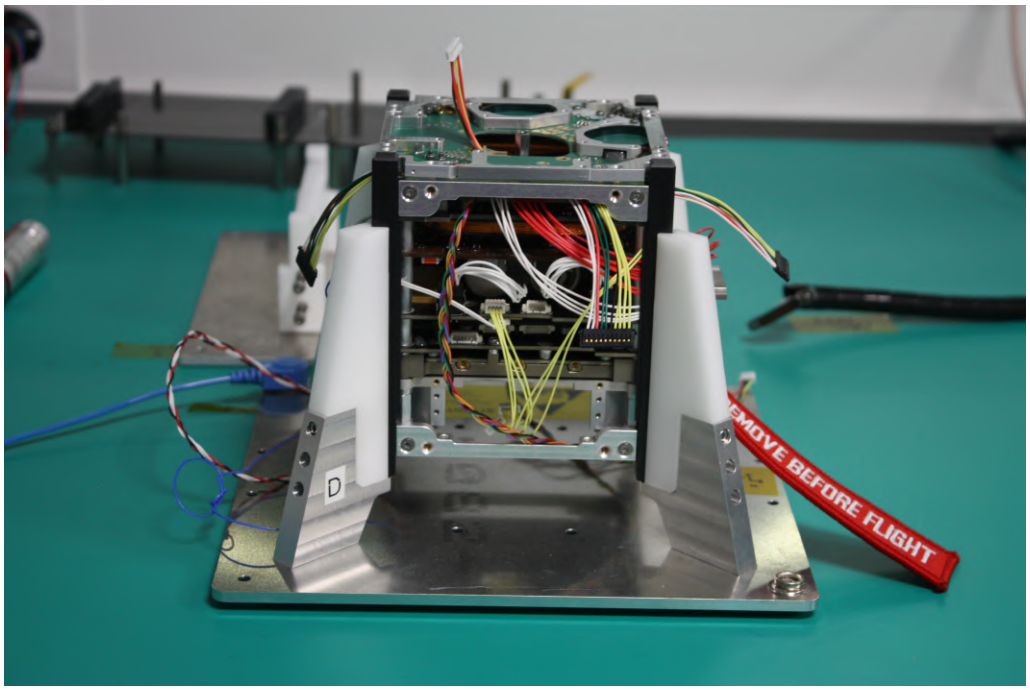


Figure 1.2: Picture of the 3Cat-4 satellite integration process [3].

1.2 Subsystems

This section covers an in-depth vision of each subsystems modules. Also, how they are stacked and communicate to each other.

1.2.1 Electrical and Power Subsystem (EPS

The EPS is the one in charge of supplying power to the satellite. It has two batteries that store the energy for running the spacecraft while in eclipse. The batteries are charged when the satellite receives sunlight, through ten solar panels mounted in the CubeSat. Its switchable outputs are controlled by the OBC, which toggle each output according to the state of the spacecraft.

Also, the EPS of this mission is a purchased COTS component from GomSpace. Specifically, it is the model P31u, which is interfaced with ten photo-voltaic cells, two per each of the five faces available. This module provides two voltage channels: 3.3 V and 5 V. These channels are available both from a permanent connection (cannot be switched off) and three switchable outputs for each line. The battery technology is lithium-ion and a heater is installed, which ensures a tolerable temperature range in all the scenarios that prevents cold damage in them.

1.2.2 On-Board Computer (OBC)

The processing capacity is delivered by the OBC, which is connected to all the subsystems and is in charge of the EPS control (toggling each subsystem power as required) as well

as controlling the operational state of the satellite. The On Board Data Handling (OBDH) subsystem is integrated in the OBC and together the whole satellite is managed.

The OBC, like the EPS, is a purchased COTS component from GomSpace. Specifically, it is the model NanoMind A3200. This OBC is equipped with an AVR32 microcontroller as well as with two I2C and other two UART bus lines for communications. Nevertheless, all the software executed in the OBC is implemented in the NanoSat Lab, remarking the one from this dissertation.

The OBC is mounted on a in-house designed and manufactured motherboard that provides the required connections and housings in order to access all the OBC pins.

1.2.3 Communications Subsystem (COMMS)

The COMMS subsystem is designed and manufactured by NanoSat Lab's students. It provides direct downlink and uplink communications between the CubeSat and the ground station. It is thus used to send telecommands to the satellite and receive telemetry and data. The radio frequency link is in the amateur band of 437 MHz, which is labelled as Ultra High Frequency (UHF). Reception and transmission chains are composed of a Low Noise Amplifier (LNA) module and a Power Amplifier (PA) respectively. This results in a reception gain of 15 dB and a transmitted power of 30 dBm.

This subsystem is integrated in the same board as the ADCS, resulting in the COMMS&ADCS module.

1.2.4 Attitude Determination and Control Subsystem (ADCS)

The ADCS is an in-house designed and built module. The ADCS is in charge of two functions: first, when the spacecraft is launched, it performs a B-dot detumbling algorithm that stabilizes and slows rotation speed of the satellite below 2 ^{o}/s . Then, when the satellite is detumbled and the NADS successfully deployed, it executes a nadir-pointing algorithm that ensures that the NADS antenna is pointing to the Earth. In order to perform these two functions, three different sensors are mounted: magnetometers, sun sensors and gyroscopes. From these sources, the algorithm obtains the state of the satellite, is processed by a proportional-integral-derivative (PID) controller and then performs the corrections using the actuators, consisting in three independent magnetorquers that acts in each body axis. The magnetorquers create a magnetic field that interferes with the Earth magnetic field, resulting in a net torque that rotates the spacecraft.

Nonetheless, at the end of the NADS there is a Teflon piece that actuates as a gravity boom, creating a non-negligible gravity gradient towards the Earth and passively pointing the satellite. The gravity boom, which will be explained later, can be observed in Figure 1.3, at the end of the deployed antenna, the white piece.

1.2.5 Flexible Microwave Payload 1 (FMPL)

The FMPL-1 is an in-house designed and built module. It combines three experiments in a single board thanks to a SDR module with lots of flexibility for adapting the algorithm

that processes the incoming signals. These experiments are an AIS receiver, a L-band radiometer and a GNSS-R.

The FMPL-1 is the component that gives meaning and value to the whole mission. The experiments that this component executes provide meaningful data for EO and its implementation using a SDR shows a novel approach for embedding multiple passive microwave payloads in a single platform. A more in-depth vision of this payload can be found in its corresponding journal publication [13].

1.2.6 Deployments Subsystems

This mission has two deployment systems: Zenith Antenna Deployment System (ZADS) and Nadir Antenna Deployment System (NADS).

The ZADS is a COTS component from ISISPACE and is equipped with two deployable UHF and VHF monopole antennas. Allocated in the top face, the VHF antenna is used for one of the payloads, specifically the AIS receiver. Then, the UHF antenna is used for ground station communication.

Then, the NADS is an in-house built subsystem. Placed in the opposite face, this subsystem includes an L-band helix antenna, deployable to 50.6 cm equally divided by 11 coil turns. This antenna ensures high directivity with a gain of up to 13.5 dB operating at 1575.42 MHz. This high directivity allows the FMPL-1 to perform payload executions and obtain a high resolution output. In Figure 1.3 a full deployment of the NADS can be observed.

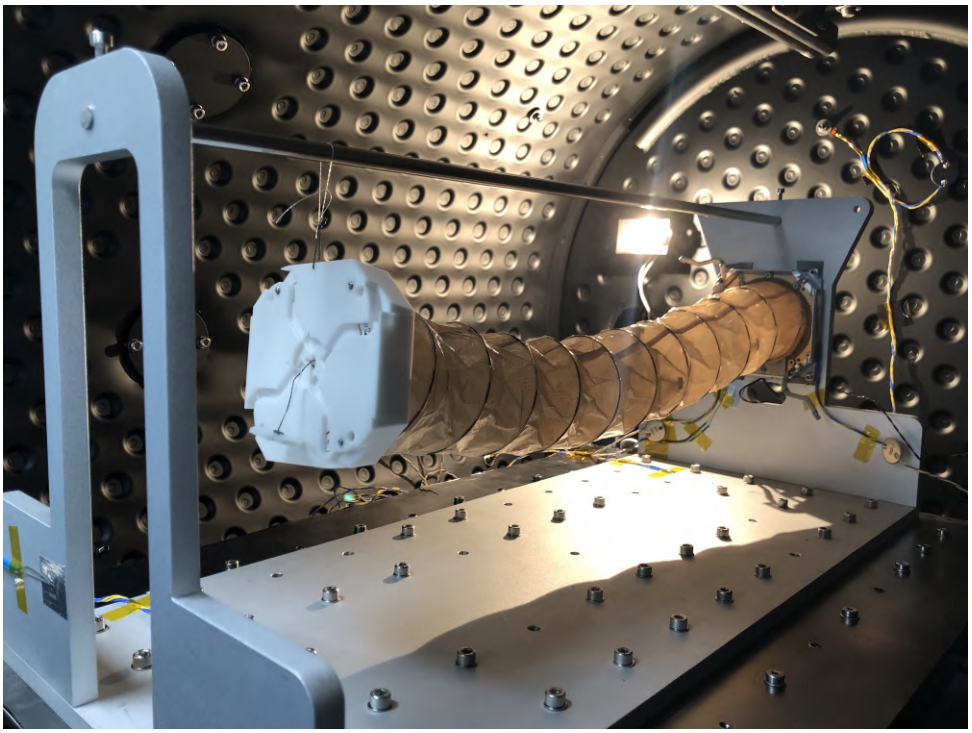


Figure 1.3: Picture of a test deployment of the NADS antenna (NanoSat Lab).

Due to its complexity, the deployment system is a relevant mechanical challenge. Further information can be found on this development on Marco Sobrino publication [14].

1.2.7 Communication buses

In order to enable communications between the subsystems and the OBC, a PC/104 standard connection bus is implemented in EPS, OBC, COMMS&ADCS andFMPL-1. The PC/104 standard counts with two lines of 52 connections each, summing up a total of 104 available connections. The two deployments components, NADS and ZADS, are connected through two picoblades connectors to the OBC.

The OBC has two I2C ports available. In one of them, Port 0, the EPS and the ADCS subsystems are connected. In the other one, Port 2, NADS and ZADS are connected. Both four are configured as slaves, obtaining the clock signal from the OBC, which is configured as the master.

Then, there are two UART buses available, one for the communications subsystem and the other one for the flexible microwave payload subsystem.

The reason why a subsystem is connected to one protocol or the other is due to prioritizing critical communication between OBC and communications board. Allocating to this communication link a UART, which is more reliable than I2C protocol, ensures a less risk of critical failure between both subsystems.

Also, manufacturer constraints in COTS components influence the final communication buses configuration. On the one hand, EPS and ZADS, as a COTS component, only have I2C hardware mounted, thus these will be connected to an I2C bus. On the other hand, both COMMS and FMPL-1 which are in-house manufactured, have UART hardware installed, preferred by team members in charge of this part. One of the reasons why COTS use I2C hardware is because it is significantly cheaper than the UART one.

A diagram expliciting the available communication buses and the subsystems connected to them can be checked in Figure 1.4.

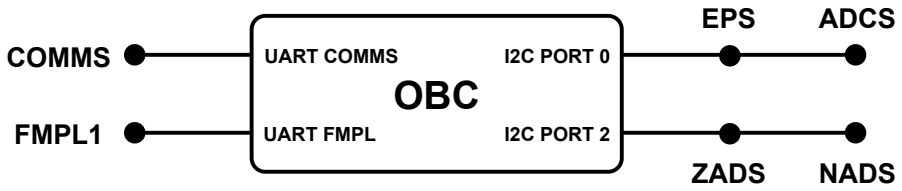


Figure 1.4: Diagram of the communication buses available in the satellite and its connections.

1.3 Satellite modes

1.3.1 Global overview

The 3Cat-4 states or modes follows the principle of a state machine. A state machine is a mathematical model of a system composed by states, transitions and actions. Then, the device where the state machine is implemented can be in exactly one of the states at any given time. Transitions between states occur when defined conditions are met. Applied to our spacecraft, the satellite is determined by modes of operations. Each mode represents defined conditions that force the system to have a deterministic behavior. Using

this methodology, the spacecraft behaviour remains stable over time. For example, the deployment of the antennas cannot be performed if the satellite mode is not the payload deploy mode. This means that, even sending the deploy telecommands from ground, the satellite would not process them.

In the state diagram of the following Figure 1.5, the whole state machine can be observed as well as the transition conditions imposed. This subsection will explain each mode briefly and then specific parts of the diagram will be more in-depth analyzed.

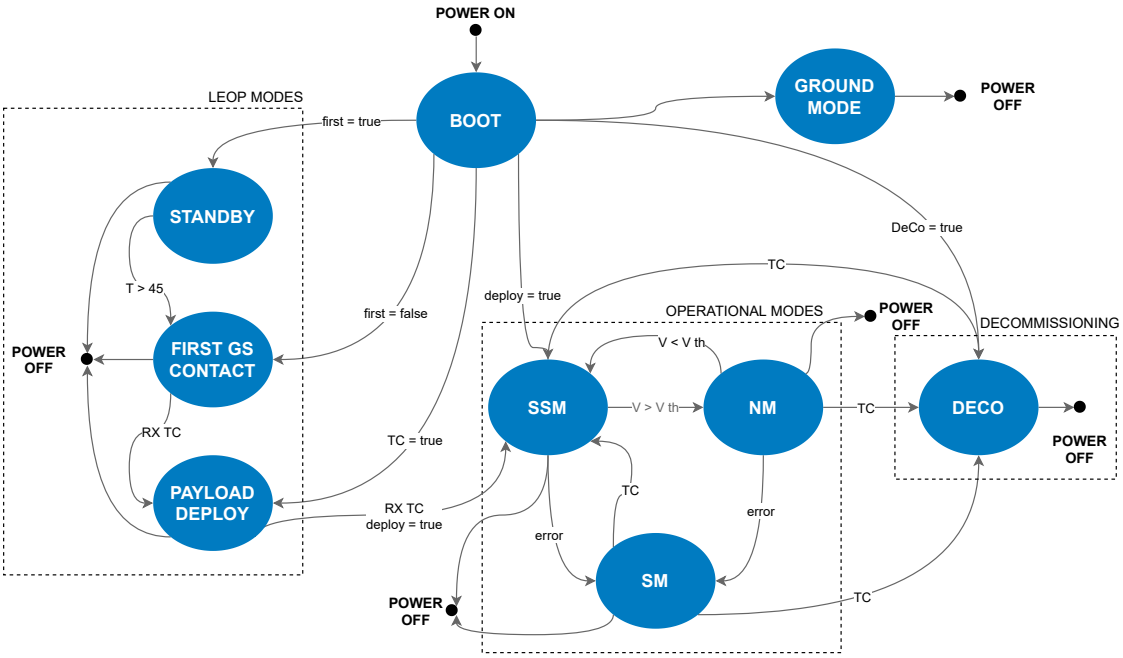


Figure 1.5: State diagram implemented to control the behavior of the satellite.

The different modes are listed as follows:

- **Boot Mode:** Initial mode after powering up the spacecraft. This mode function evaluates the conditions saved in the persistent storage memory to decide in which state should be the satellite. These conditions are first time boot (first), deployed antennas (deploy) and decommission (DeCo). Depending of its state is True or False, the state machine will transition to a state or another. Moreover, it initializes the memory library, the communication buses, etc.

- **StandBy Mode:** Mode after the orbit injection when the satellite has to wait 45 minutes without transmitting in order to prevent interference with other satellites. This behaviour is fixed by the launch regulations.

- **Ground Mode:** Mode designed for ground testing but also accessible in orbit. No limitation in terms of function execution, designed for contingency scenarios. It gives to the operator a total control of the spacecraft

- **First Ground Station Contact Mode:** Mode reached after expired wait time of 45 minutes in StandBy mode. Looks for first communication to Ground Station, which is detected when a command reception occurs.

- **Payload Deploy Mode:** Mode reached after receiving the first command. In this mode, the NADS is deployed sequentially command-by-command.
- **Sun Safe Mode (SSM):** Mode entered automatically when battery voltage drops below the setted voltage threshold. In this mode, several high-consuming energy functions cannot be performed, such as payload execution. It is used to protect the satellite and extend it operating life.
- **Nominal Mode (NM):** Mode entered after the Launch and Early Orbit Phase (LEOP) modes and when battery voltage is over the nominal threshold value. It is the desired mode to be during the operations of the satellite because payload executions can be performed.
- **Survival Mode (SM):** Contingency mode, entered when an error is raised in any of the subsystem and waits for the operator to solve it or override it uploading a reference error mask.
- **Decommissioning Mode (DeCo):** Mode entered when the operations of the satellite are concluded. Nonetheless, the operations period will be evaluated when in orbit in order to extent it at its maximum.

1.3.2 Launch and Early Orbit Phase (LEOP) modes

The LEOP phase is the most critical phase of the mission as it comprises the launch of the satellite, the first ground station contact, antennas deployments and verification that the correct satellite status is reached. In terms of the state diagram, the LEOP modes can be observed in Figure 1.6.

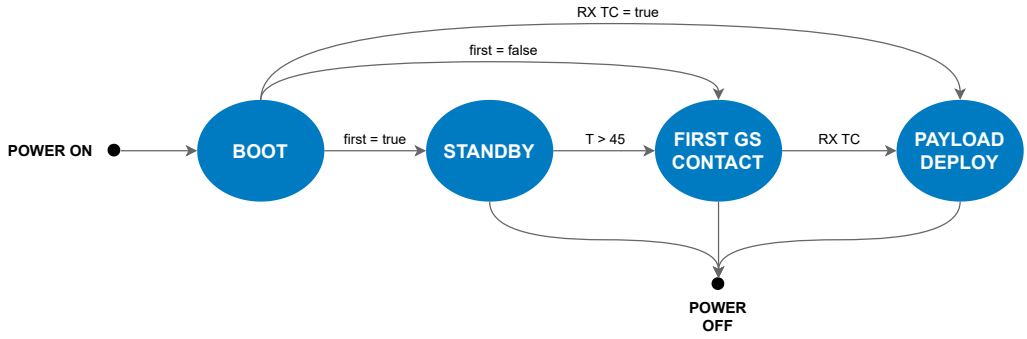


Figure 1.6: LEOP modes.

First, the satellite enters in boot and evaluates the state. When launching, it transits to standby when awaits for the 45 minutes regulation of no transmitting. Then, transits to first ground station contact and deploys the ZADS automatically, waiting for a telecommand reception. Finally, when the first ground station communication is received the spacecraft transits to payload deploy where the operator is able to deploy the NADS command-by-command.

1.3.3 Operational modes

After the LEOP modes, the satellite enters its operation phase, in which it follows the nominal sequence represented in the state diagram of Figure 1.7 hereunder:

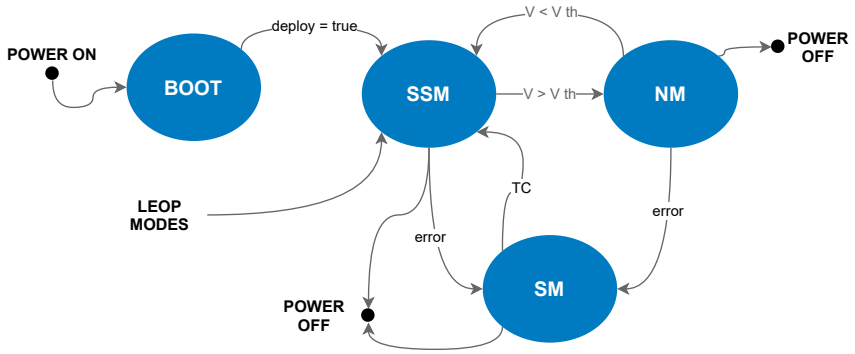


Figure 1.7: Nominal sequence.

When booting, the satellite first checks the deploy bool. If True, this means that all the LEOP modes have been concluded successfully and jumps straight into Sun Safe Mode (SSM). If the deploy bool is False, it must first finish the LEOP sequence.

From both origins, the first state of the nominal sequence is Sun Safe Mode. In Sun Safe, the system runs as usual but without performing any experiment, just ground communication and attitude control. If there is no error and the battery voltage is higher than the nominal threshold voltage, the system jumps to Nominal Mode (NM). There the spacecraft is able to schedule an experiment execution.

If an error is raised in both Sun Safe or Nominal Mode, the system jumps directly to Survival Mode (SM). In Survival Mode, the satellite behaves very similar to Sun Safe Mode (No payload execution) but informs the operator that an error has been raised, indicating from which subsystem and which error has raised. Then, the operator can solve this error by sending telecommands (for example, executing a controlled reboot of the OBC) or can choose to override this error uploading the error's subsystem reference error mask with the specific error bit position set to zero.

The implementation of this reference error mask gives the operator full control regarding error management on board the satellite.

1.3.4 Decommissioning mode

The decommission mode (DECO) is the state reached when the mission has been accomplished and the satellite is no longer required. Decommissioning can be reached through telecommand from any state of the satellite.

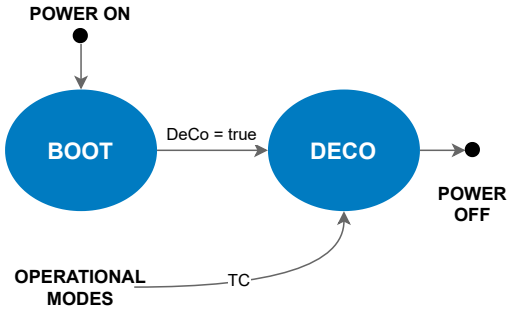


Figure 1.8: Decommissioning mode.

The main intention of the 3Cat-4 team is to nominally operate the satellite until any of the subsystem components fails, extending its operative life. So, decommissioning time will be evaluated when the satellite reaches orbit and depending on its overall functioning.

1.3.5 Ground mode

The ground mode is a state designed for ground testing purposes, with the special condition that it does not have any function execution limitation. This means that all the commands can be executed, without considering the satellite state. Although, it can be entered in orbit or a contingency plan/mode.

The ground mode is the wild card of the state machine. Can be accessed from any state by telecommand and allows to execute any telecommand without conditions. By design, there is no plan to use ground mode in orbit, but the option exists if there is a contingency that needs its powers.

1.4 Software architecture

This section covers the software architecture of the whole satellite. First, an explanation about Real Time Operating Systems (RTOS) and why are important in space applications. Then, a more in-depth explanation of FreeRTOS, the option implemented in this development. Hereafter, the architecture implemented between the subsystems will be explained.

1.4.1 Real Time Operating Systems and spacecrafts

A real time operating systems (RTOS)[15][16] is a timebounded operating system which has well-defined, fixed time constraints. This means that all the processes executed have a maximum time to finish or the process is killed and considered as a failure. This approach enables to work with an operative system that does not get blocked by any specific failed process, allowing the computer as a whole to keep operating and thus having a deterministic behavior.

Inside RTOS, there are two different architectures regarding tasks execution: (1) event-driven RTOS switch between tasks depending on their priorities, creating a hierarchy of tasks relevance; (2) time-sharing systems allocate execution times to each task based

on clock interrupts. The main time unit in RTOS are ticks, which value is customized but usually in the order of magnitude of milliseconds.

One of the most important applications of RTOS is as operating system of spacecrafts. The main reason is thanks to its deterministic condition, which enables flight software engineers to design deterministic processes with predictable outputs, managing both nominal and contingency scenarios. Space-grade software has to be completely predictable and perform within specific time bounds.

Both NASA and ESA missions have run with RTOS since its beginnings. Although in the first missions, the operative systems was specifically build according to mission requirements and constraints, two major RTOS names became popular in space missions in both sides of the Atlantic. On the one hand, NASA used to implement a proprietary RTOS called VxWorks, which became famous for being the first relevant commercial licensed software for space operating systems. On the other hand, ESA, although starting also with VxWorks, quickly switched to RTEMS. RTEMS standed for Real Time Executive for Missile Systems, as it was first created for managing flight control of missiles. Nonetheless, ESA invested heavily in developing RTEMS software thanks to its open source philosophy, which allows access to the source code (unlike VxWorks). Then, the software was renamed as Real Time Executive for Multiprocessor Systems.

Nowadays, there is a wide variety of RTOS for space applications. Despite the two more influent and used ones are still VxWorks and RTEMS, there are other options such as Linux or FreeRTOS. On the one hand, Linux is the widely known open source operating system, which can have its kernel modified to lighten its weight and boost real time behavior. It is also increasing in popularity among those missions with power, weight and space constraints not limited that can allow to use a powerful microprocessor. On the other hand, the FreeRTOS kernel is a lightweight and open source operating system which needs a small amount of resources to successfully work. Nonetheless, it lacks some of the most powerful and advanced functions found on larger operative systems. It is popular for CubeSat missions, where the budget in terms of power and memory available is limited, where FreeRTOS fits well.

1.4.2 FreeRTOS

In the 3Cat-4 mission, the chosen real time operating system is FreeRTOS, due to its flight heritage on this kind of CubeSat missions. Although having a total weight of between 4KB to 9KB, the main functionalities of RTOS are available: unlimited tasks, memory management, queues, semaphores and mutex. The main code language used is C.

1.4.2.1 Tasks

FreeRTOS is organized in tasks. A task is a block of code that is executed independently of the others in the core of the processor. Then, the scheduler is the one that manages the start/stop of task's executions based on fixed task priorities and the ticks assigned to each task. In the 3Cat-4 mission, there is the manager task, which is the main one that controls and manages the other tasks and satellite modes transitions. Then, there are six tasks that depend directly from the manager. These are EPS task, Tracking, Telemetry and Command (TTC) task, ADCS task, OBDH task, Deploys task and Payload task.

Then the Scheduler of the operating system assigns a decimal value to each task. The higher this value, the more priority has the task to be executed. This means that if when executing a task and a notification from another task with higher priority is received, the system stops the first task to execute to process and execute the required functions in the higher priority task. For our mission, this is defined in the system.h file and the tasks are listed in a priority descending order: (1) Manager task, (2) EPS task, (3) TTC task, (4) ADCS task, (5) OBDH task, (6) Payload task and (7) Deploys task.

This priority order is based in setting the Manager task as the one with highest priority, which is reasonable as is the one that manages the other tasks and thus the most critical one. Then the EPS task, which supplies power to the satellite and is also a critic subsystem. Following, the TTC task as is the one that processes the telecommands received from ground station. Then, the remaining tasks that are relatively low critical in terms of instant execution which are ADCS task, both OBDH and Payload task and finally the Deploys task, which is only relevant in a short period in the satellite operating lifetime.

CHAPTER 2. SOFTWARE DEVELOPMENT

Previous chapters present the context and the motivations on how real time operating systems works. This chapter contains the core work of this thesis, which is the software development implemented, divided in two groups: space segment and ground segment. It is important to differentiate between the software development involved in any space mission, as both space and ground segment software have particularities. These two groups are explained and described in the following sections of this chapter.

2.1 Space segment

Space segment software is the one that runs entirely on the spacecraft. This corresponds to the main software executed in the OBC, but also the ones executed in each subsystem. Space segment software must be reliable, specially in this mission due to the inability of performing patching in orbit as a consequence of the software architecture. To achieve this reliability level, some aspects must be satisfied: (1) its behavior should be deterministic (the reason why RTOS are used); and (2) most of the actions must be executed autonomously, due to the inability to directly control and maneuver the spacecraft in real-time during the mission. In our mission, for example, the communication between space segment and ground segment is only available during the time visibility window, which is a small percentage of time (around 10 minutes each pass) over the total duration of the mission.

Below, the nominal task structure is explained and then the different tasks running on the OBC are explained, remarking its main and important functionalities. Also, flowcharts are provided in order to enhance the comprehension of each task functions.

2.1.1 Task structure

Each subsystem task share a common execution structure because all the tasks have common functions, such as initialization and process. Therefore, a standard and common procedure is determined for each task. This structure has three steps: (1) the task loads the init function, which is in charge of initializing several parameters used in the task; (2) the flow checks if the configuration is uploaded in the subsystem correctly; (3) the process function is executed, which runs in an infinite loop and is in charge of managing the subsystem software itself. Most of the general and specific functions are inside process global function. This is characteristic of RTOS and specifically FreeRTOS, where the computer iterates over the process function of each running task, listening to queues and notifications between them and performing the required functions.

The fixed structure can be observed in the flowchart of Figure 2.1, hereunder.

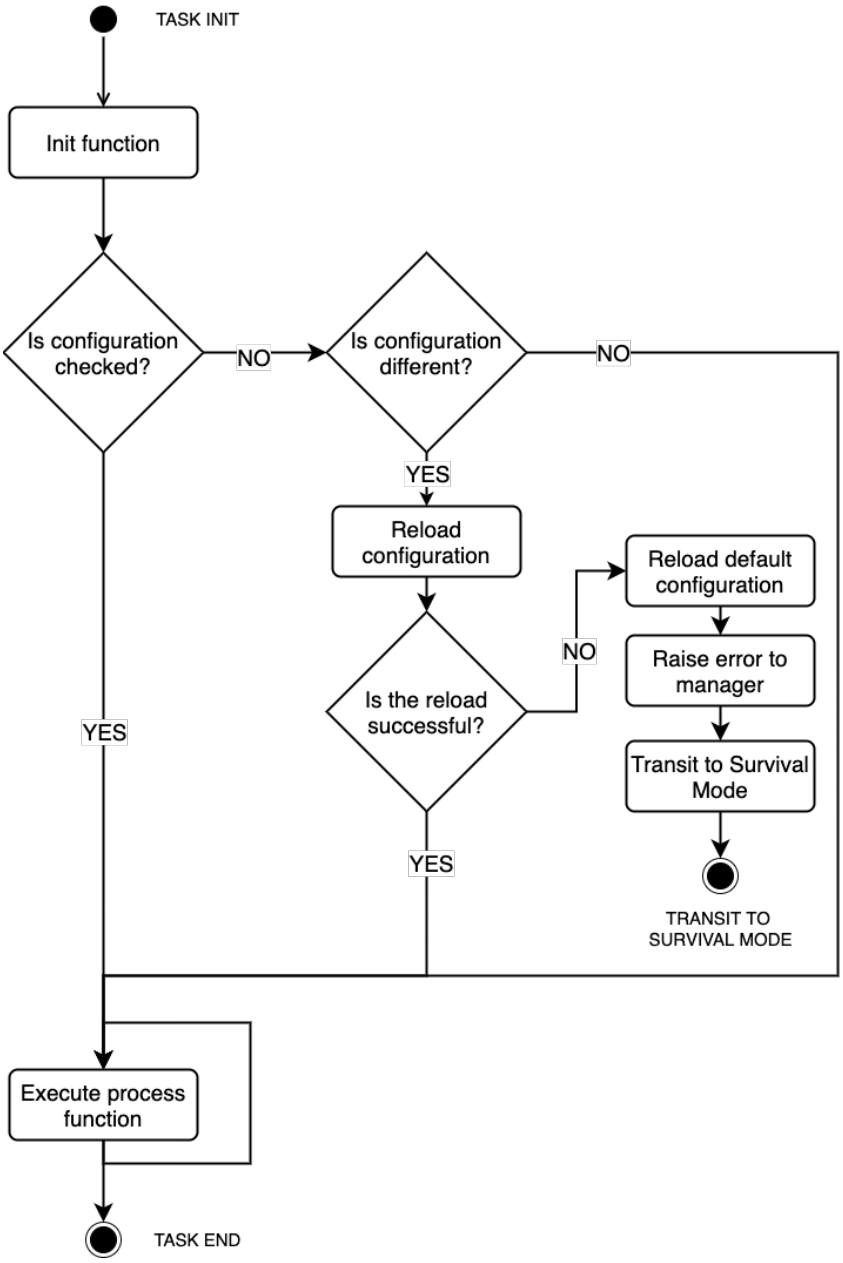


Figure 2.1: Tasks fixed structure flowchart.

As seen, once the init function and the first check of configuration of the subsystem is done, the software loops infinitely in the process function.

Moreover, each subsystem and, in consequence, each task has its own configuration file with general and specific parameters of the subsystem. Also, each subsystem has a housekeeping structure and an error structure.

These key points are analyzed and explained for each subsystem in the annex I, due to its extension: (1) Configuration parameters; (2) Housekeeping parameters, (3) Error structure, (4) Functions.

Instead, in the following section I will write about the most relevant contributions to each subsystem task as well as a high-level logic flowchart of the process function of each. This contribution is about the design of logics and algorithms, implement them and finally

test that the behavior is as expected. All the software developed is being executed in the OBC, this project does not cover the work done implementing the software inside each subsystem.

2.1.2 Error management

All the tasks follow the same error management strategy. This management is an important concept to take into account when designing flight software. It increases the autonomy of the system, managing unexpected errors and malfunctions and raising the corresponding errors to warn the satellite operator, according to Figure 2.2.

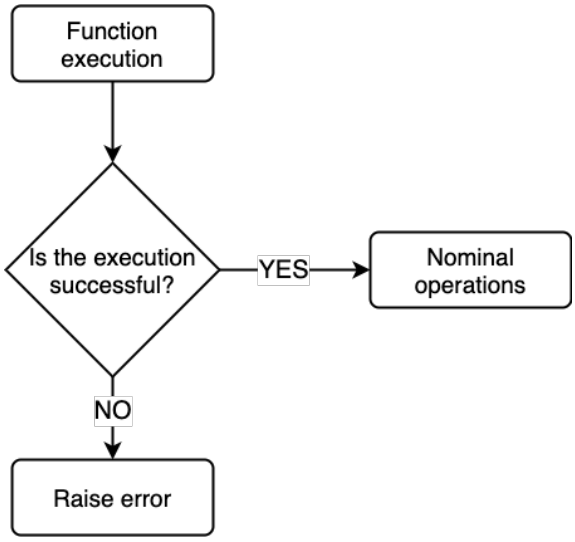


Figure 2.2: Error logic flowchart.

The 3Cat-4 error management consists in the implementation of the specific function that detects this error and processes it. Nominally, the error warning is sent to the manager and this changes the mode of the satellite to Survival Mode (Section 1.3.1). Nonetheless, the function first reads the state of the spacecraft and the reference error mask, and depending on them it finally sends the error to the manager or not. For example, if the spacecraft mode is already in Survival, it makes no sense to send the error to the manager, so the error is queued.

Each task, appart of the error notification capability, has an error mask to represent the type of error. The reference error mask is an 8-bit bitwise mask where each bit represents each subsystem errors (up to 8). Normally, this mask is set to the decimal value of 255, which equals to eight one's (1111 1111). With this value, the error mask has no effect over the overall functioning of the system. Nonetheless, if the operator changes one of this bits to 0, the corresponding error, if raised, would be overridden. This gives the operator the power to manage the errors on board the spacecraft and the tool to override them if a dead-end situation happens in the software logic. The logic of the raise error function can be checked in Figure 2.3.

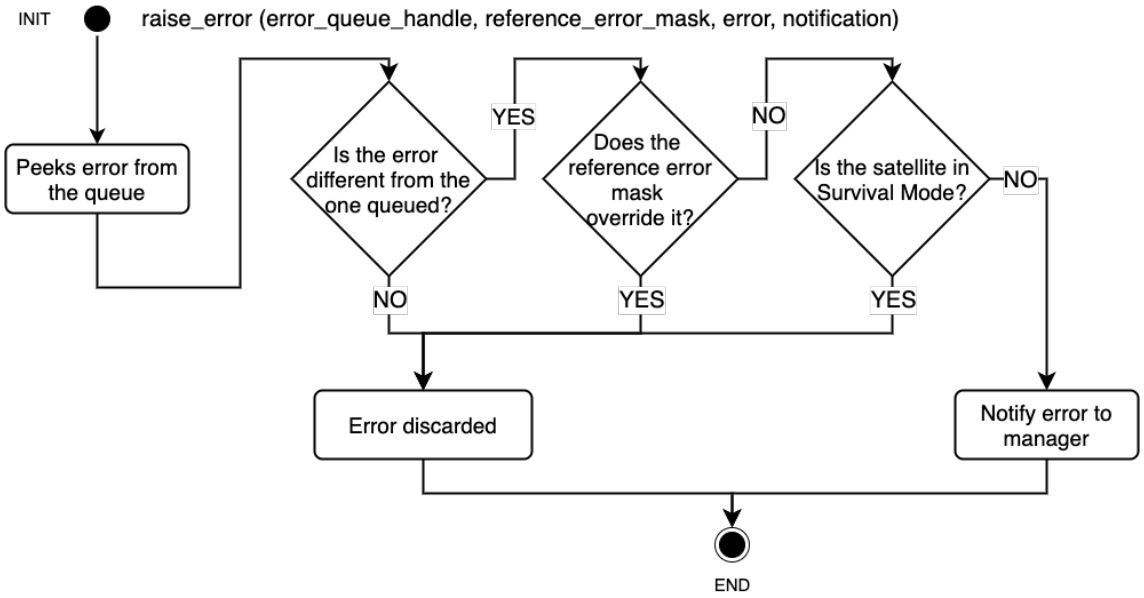


Figure 2.3: Raise error logic function flowchart.

2.1.3 EPS task

The main functionality of the EPS task is to manage the EPS subsystem [Section 1.2.1]. The EPS is a COTS component, manufactured by GomSpace, and has its own communications interface created. For this reason, when developing the EPS task we had to follow the GomSpace datasheet for controlling the EPS from the OBC. This is done by transmitting and receiving a series of fixed I2C packets, in which the content field is the command that we want to perform in the EPS and is specified by EPS datasheet.

The EPS subsystem includes several layers of safety and self-management, being the most relevant the watchdog timers and the battery modes.

On the one hand, the watchdog timers are a resource used in RTOS that consist in a timer that counts backwards to zero from a given positive natural number (normally in seconds) and when the zero is reached, an action is triggered, normally a reboot of the subsystem or the task. In this case, the EPS counts with two watchdog timers, being the first one regarding I2C communications. When a I2C packet is received, the watchdog timer is reset back to its initial value. In our mission, this value is set to 120 seconds. Also, the second one is the ground watchdog timer. This timer is designed as to be executed when a signal from ground station is received, as a health check system of the communications between the spacecraft and ground station.

On the other hand, the EPS has three battery modes in function of its current voltage. This modes are critical, safe and normal and the threshold values can be configured through the configuration file. The OBC adopts this system and only considers two different modes, safe and normal and are the ones that are directly related to the satellite state modes of Sun Safe mode (EPS battery in safe mode) and Nominal mode (EPS battery in normal mode).

2.1.3.1 EPS process function

The process of the EPS task consists of two separates steps. First, it checks if there is any new notification queued and which kind are they. It accepts two types of notifications which are the one that requests to load a new configuration and the one that requests to change the value of one of the point of loads.

Second, after this notifications are processed, it enters to the function to process the housekeeping that consists on first requesting the new housekeeping and then parsing the values in order to update the manager task about key parameters such as the battery mode and the battery voltage. Also, it checks during three iterations if the batteries temperature is lower than the low hysteresis temperature value. If it's below, automatically an error is raised to the manager and the satellite transits to survival mode.

This logic can be better observed in the flowchart hereunder, in Figure 2.4.

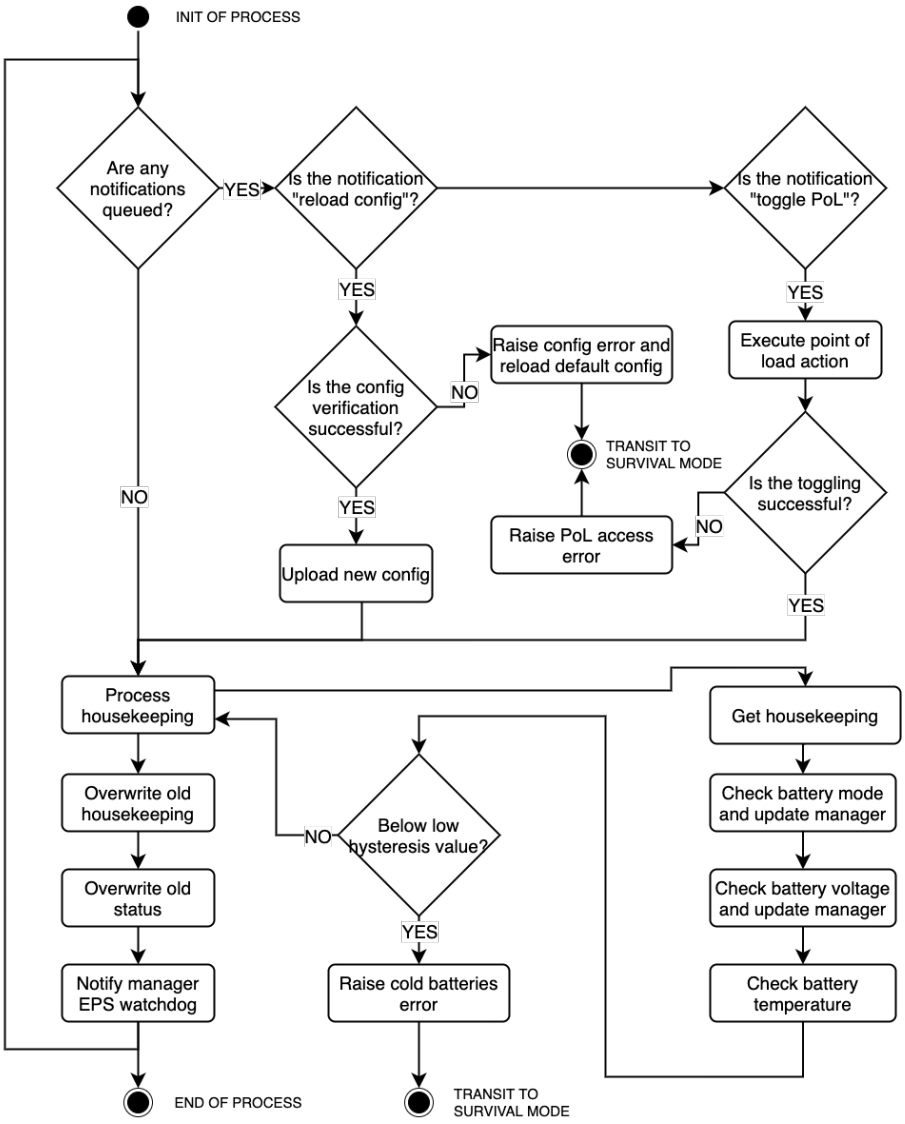


Figure 2.4: EPS process function flowchart.

2.1.3.2 Contributions to the EPS software

The contributions from this thesis to the EPS task software are the implementation of the error management layer and up to four new functions have been added to the EPS subsystem task:

- **eps_reset_wdt():** This function aims to reset the ground watchdog timer of the EPS back to its original value of 48 hours (172800 seconds). Its logic is to send an I2C packet with the reset ground watchdog timer command in its content field. This function is executed every time that there is ground contact with a base station, indicating that everything is fine and that there is no need to reboot the EPS subsystem.
- **eps_set_heater():** This function aims to set the EPS heater to manual or automatic mode. Although it is set to automatic by default and nominal operations don't require a change, the software team considered that it would be important to be able to manage this parameter in case of a contingency related. Its logic is to send an I2C packet with the set heater plus a one or zero, depending on the requested mode.
- **eps_hard_reset():** This function aims to execute a hard reset of the EPS subsystem. Its logic is to send an I2C packet with the hard reset command in its content field. This function is available to be used in a contingency case if a hard reset of the EPS is needed instantly and it is no possible to wait until the ground watchdog timer reboots itself.
- **eps_counter_reset():** This function aims to reset the EPS boot counter parameter. Its logic is to send an I2C packet with the reset boot count command in its content field. This function is no planned to be executed nominally but can be useful in a scenario in which the EPS behavior needs to be monitored.

Also, EPS subsystem tests have been performed at system level, ensuring that the overall functioning is correct and the error management performs accordingly.

2.1.4 TTC task

The main objective of the TTC task is to manage all the communications between the spacecraft and the ground station. This task receives the data that has to be sent to ground, normally the instantaneous and historic telemetries and also the experiments results, packs it and sends through the UHF/VHF antennas.

2.1.4.1 TTC process function

The process function of the TTC task first checks if the transmission of packets is enabled or disabled, in function of the spacecraft mode (remember than in first boot time the satellite cannot transmit). If the mode allows to transmit, checks for notifications of reload configuration, new scientific data and new telemetry. If one of these notifications is found, it follows to perform the required action.

Then, it checks the internal watchdog timer of the task with the manager task. If the watchdog timer is still alive, it proceeds to update the command file tracker, parse the new telemetry and forward it to manager as well as doing the same with the task state.

This logic can be better observed in the flowchart hereunder, in Figure 2.5.

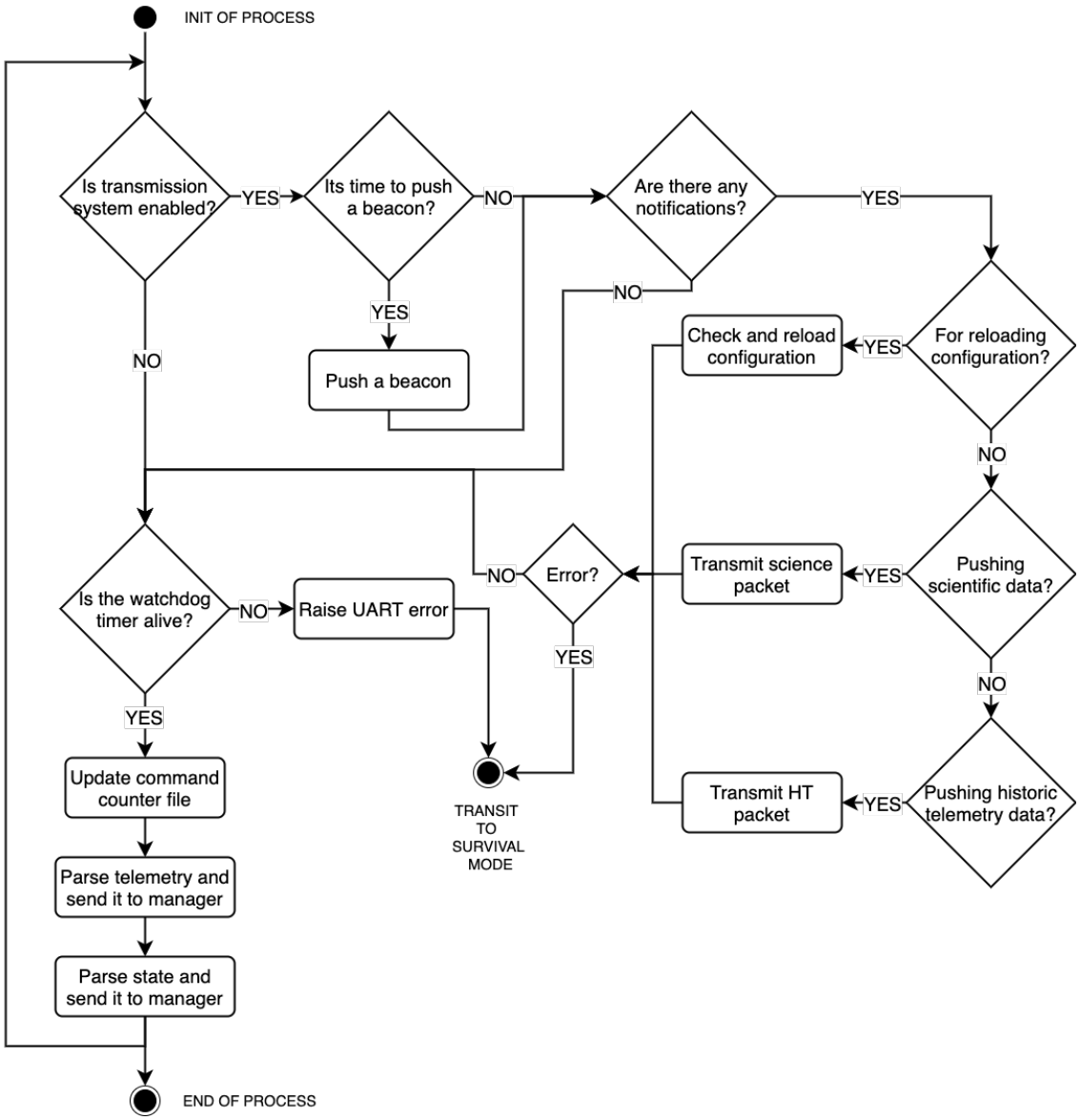


Figure 2.5: TTC process function flowchart.

2.1.4.2 Contributions to the TTC software

The contributions from this thesis to the TTC task software are the implementation of the error management layer and one new function has been added to the TTC subsystem task:

- **reset_cmd_file():** This function aims to reset the file that stores the command counter of the TTC task. Its logic is to open the TTC reference file and overwrite any existing content with the same structure but set to zero. This function is executed every time

that there is a problem derived of the command sequence not following a natural order due to a lost command.

Also, TTC subsystem tests have been performed at system level, ensuring that the overall functioning is correct and the error management performs accordingly.

2.1.5 ADCS task

This subsection covers the key aspects of the ADCS task. The main objective of this task is to properly set the uploaded configuration values and retrieve correctly the housekeeping parameters to and from the ADCS board. Also, there is a mission test mode implemented with the intention to perform a short high-sampling data set that allows the operator to perform an ellipsoid fitting test on ground as well as a complete ADCS health check in orbit.

2.1.5.1 ADCS Two Line Element set (TLE) file

Along with the configuration file, the ADCS subsystem requires another input which is the Two Line Element set (TLE) [17]. The TLE is provided by the US Army and is a two line element set that describes the orbit of the body around the Earth and its position for a given time. This TLE is fed to an orbit propagation algorithm inside the ADCS board that obtains high precision coordinates of the spacecraft for the next few days. This allows the system to obtain, for example, the International Geomagnetic Reference Field (IGRF) values for each Earth zone and calibrate the magnetometers accordingly.

The TLE is uploaded as a text file, not as a JSON file like the normal configuration and has its own specific commands for uploading it. A sample of a previous NanoSat Lab's mission TLE is stated below:

```
1 46292U 20061W   21124.16166338  .00000590  00000-0  39655-4 0  9992
2 46292  97.4889 198.4490 0003399 173.0253 187.1024 15.10457944 36642
```

2.1.5.2 ADCS process function

The ADCS process function is more structured than previously seen process. First, it checks for the mode of the board. If nominal, retrieves its housekeeping and sends it to the manager. If in mission test mode, perform the specific functions of the mission test process, which are a shorter period for recollecting more data and the creation of the test data packets and forwarding them directly to TTC task.

Then, it checks the notifications queue and processes three different notifications, if any. First, checks for a mission test request, then for a configuration reload and finally for a TLE reload.

Finally, it notifies the consequent internal watchdog timer to the manager task in order to keep the task running.

This logic can be better observed in the flowchart hereunder, in Figure 2.6.

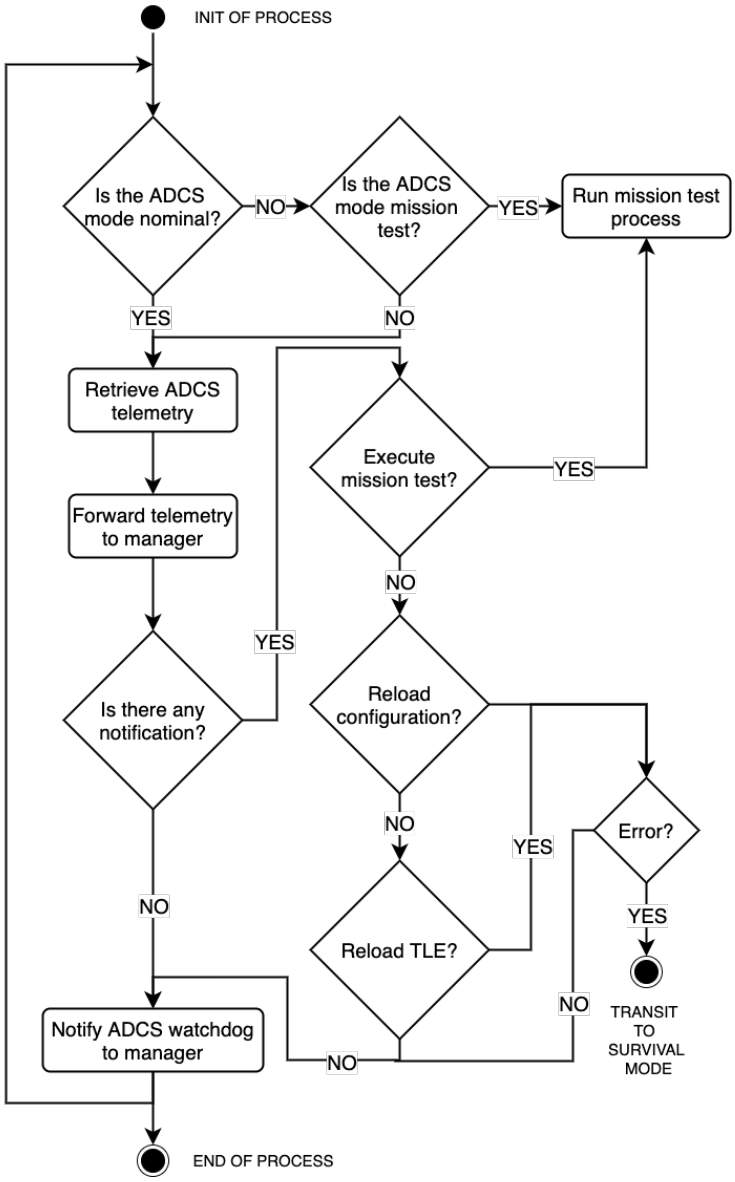


Figure 2.6: ADCS process function flowchart.

2.1.5.3 Contributions to the ADCS software

The contributions from this thesis to the ADCS task software are the implementation of the error management layer and most of the used functions. ADCS task is the first one where I had to port and rewrite most of the code from the Raspberry Pi that was performing the OBC role when used for the subsystem development. As a consequence, I spend more hours in development than the two previous tasks combined. All the functions are implemented by transmitting or receiving I2C packets from the ADCS subsystem board.

- **adcs_set_conf():** After having the configuration file values, this function sets to the ADCS board the provided configuration parameters.
- **adcs_set_tle():** This function sets the TLE to the ADCS board. In the board, the TLE is feed directly to the orbit propagator to obtain the required values of position and others.

- **adcs_get_hk():** This function retrieves the housekeeping from the subsystem, parses it into an ADCS housekeeping structure and sends it to the dedicated queue. Outline that this function encompass specific parameters get functions.
- **init_process_test():** This is the specific init function of the mission test process. As a difference with the normal init function, this sets a higher tick frequency in order to increase the sampling resolution and obtaining more data in the same amount of time.
- **process_test():** This is the specific process function of the mission test process. It performs the same actions as the usual process but adds in the telemetry the EPS data about total and ADCS point of load current consumption as well as structuring and sending the test data packages directly to TTC task.

Furthermore, this thesis work comprises the design of the logic for uploading a TLE in a TXT file and set it to ADCS, previously parsing it into a structure and ensuring that the values were sounding and there was no error in the string.

Moreover, with the other software team members we agreed to implement an ADCS mission test mode. This test consists in storing data at a higher frequency than usual in order to have more resolution of the readings. A test of this mode can be found in chapter 5, where some interesting FlatSat tests are covered.

Also, ADCS subsystem tests have been performed at system level, ensuring that the overall functioning is correct and the error management performs accordingly.

2.1.6 Deploys task

This subsection covers the key aspects of the deploys task. The main objective of this task is to manage the deployment of the two subsystems involved: NADS and ZADS. Both subsystems have deployable antennas and the functions to get its telemetry or send the command to arm, disarm and deploy them are explained.

Regarding communications, both NADS and ZADS are slaves in the same I2C port, so the communication protocol between OBC and them is I2C. Thus, the functions of this task create I2C packets with the correct format and use the content field to send the commands required.

2.1.6.1 Deploys burn safety addition

When the deploys task was almost practically closed, we encountered a problem with a new implementation in the satellite. Due to a EPS constraint, the NADS burning had to be done from the permanent 3V3 port instead than from the ZADS/NADS point of load. The main problem was that the permanent 3V3 line wasn't switchable so we rewired the NADS board in order to use the switchable output as a reference for the built-in Direct Current to Direct CurrentDC-to-DC converter to follow. This way, only when the ZADS/NADS point of load was up, the 3V3 permanent line would be available to burn.

Nonetheless, after some testing we noticed that when rebooting the EPS, for a short period of time of about one and a half second, the switchable output had an undetermined value

that the NADS DC-to-DC converter understood as 1. Also, the microprocessor of the NADS board was booting and not cutting the 3V3 line. In consequence, there as an actual burning of all the connected wires at the same time. This behavior with all the resistors and khantal wires mounted would mean a curren peak of almost 5 A, enough to burn several components.

In order to solve it, I teamed up with the hardware teammates and proceed to manufacture a small PCB that obtains an available GPIO from the OBC. From the OBC we have the complete control of this GPIO, without undetermined time periods. Then, we proceed to rewire it to make this GPIO the reference when opening the 3V3 permanent line for burning purposes in NADS.

In the figure it can be observed how the functions to set high and low this GPIO are executed inside the burning sequence.

2.1.6.2 Deploys process function

The deploys process function has a similar structure like other task's process. First, it checks for new notifications. If this notification is for reloading the configuration, it is executed. If not, first checks if the spacecraft is in nominal mode. If yes, proceeds to check if the notification is to arm, disarm, deploy or reboot the NADS. This is because the NADS deployment is only possible if the batteries of the satellite are fully loaded, due to the high battery drain that the burning sequence represents. The required action is performed and after receiving the acknowledgment, goes directly to execute the retrieve housekeeping function. This function is formed by two different functions, one for each subsystem. If the housekeeping retrieval is correct, the function sends the internal watchdog timer notification to manager and starts again.

This logic can be better observed in the flowchart hereunder, in Figure 2.7.

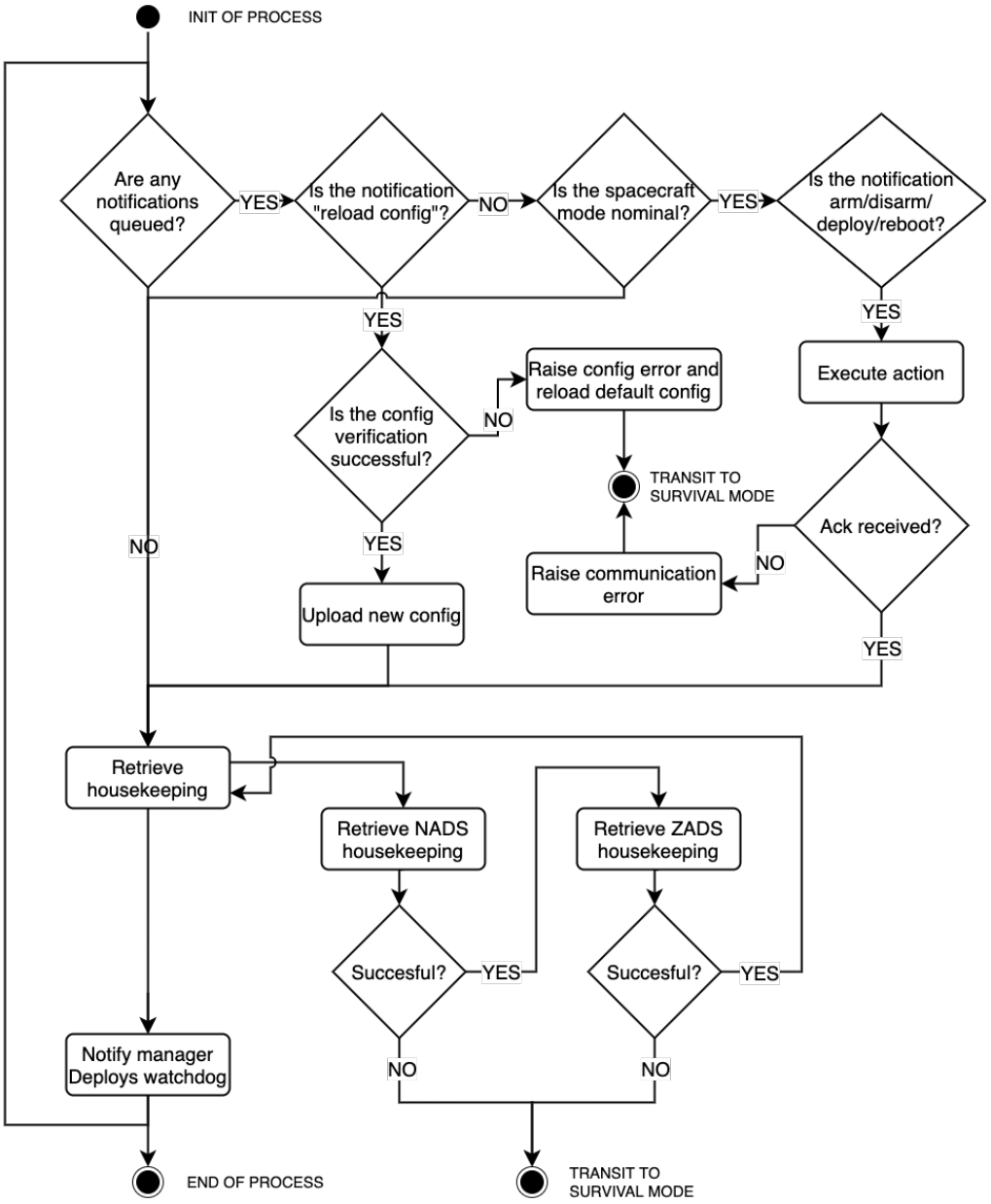


Figure 2.7: Deploys process function flowchart.

In this task it is worth mentioning also the burning sequence of the NADS antenna. In this sequence, three commands take part, which are arm, deploy and disarm. Although being a relatively easy logic, it is worth to notice the moments in which the GPIO pin is set high and low, as explained in the previous subsection of this section.

This logic can be better observed in the flowchart hereunder, in Figure 2.8.

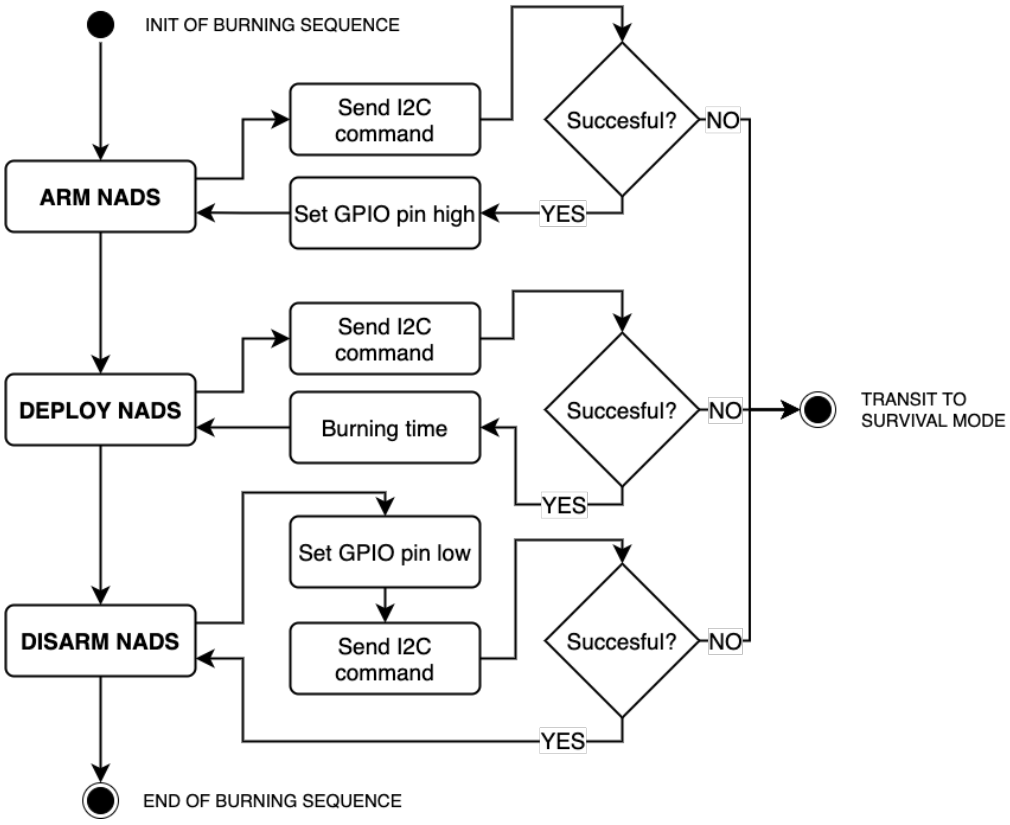


Figure 2.8: Deploys burning sequence flowchart.

2.1.6.3 Contributions to the Deploys software

The contributions from this thesis to the Deploys task software are the implementation of the error management layer and the NADS functions. Similarly to ADCS task, for the NADS I had to port and rewrite most of the code from the Raspberry Pi that was performing the OBC role when used for the subsystem development. Although, the ZADS software was already done by the manufacturer of the COTS component, ISISpace. My contribution in ZADS was to add the error management layer.

- **retrieve_NADS_hk():** This function retrieves and parses the NADS housekeeping from the NADS deployment board.
- **deploy_NADS():** This function sends the deploy command to the NADS board with a structure as content. In this structure, the type (resistor or khantal wire) is indiciated, as long as the id (1 to 5) and the burning duration in seconds (normally 5 seconds).
- **arm_NADS():** This function sends the arm command to the NADS board. This command allows the burning sequence inside the subsystem board and it is completely necessary to send it before sending the deploy command. If the NADS board return an ack, then the OBC GPIO that allows the power from the 3V3 permanent voltage is set to 1.
- **disarm_NADS():** This function sends the disarm command to the NADS board. This command disallows the burning sequence inside the subsystem board. Before

sending the disarm command, the GPIO pin is set to 0 to minimize the timespan of the GPIO being at 1 and thus able to drain the batteries if there's a short-circuit or malfunction in the circuit.

- **reboot_NADS():** This function sends the reboot command to the NADS board.

Also, Deploys subsystem tests have been performed at system level. Apart from testing all the implemented commands, a full burning sequence and deployment operations of both NADS and ZADS antennas has been tested. This tests are further explained in chapter 5 FlatSat testing.

2.1.7 Payload task

This subsection covers the key aspects of the Payload task. The main objective of this task is to manage the execution of the three different experiments that can perform the payload module.

Regarding communications, the FMPL-1 module has an UART connection with the OBC.

2.1.7.1 Payload process function

The Payload process function has a similar structure like other task's process. First, it checks for new notifications. If this notification is for reloading the configuration, it is executed. If not, first checks if the spacecraft is in nominal mode. If yes, proceeds to check if the notification is to stop or execute any of the available experiments. This is because the experiment execution is only possible if the batteries of the satellite are within a health voltage range, due to the high battery drain that the payload execution represents. If there is any payload event, it is executed and if not the algorithm executes the function to find closer experiments from the experiments file. Then, housekeeping is retrieved and, if correct, the housekeeping and status queue are overwritten with the new data. Finally, the internal watchdog timer is notified to the manager task.

This logic can be better observed in the flowchart hereunder, in Figure 2.9.

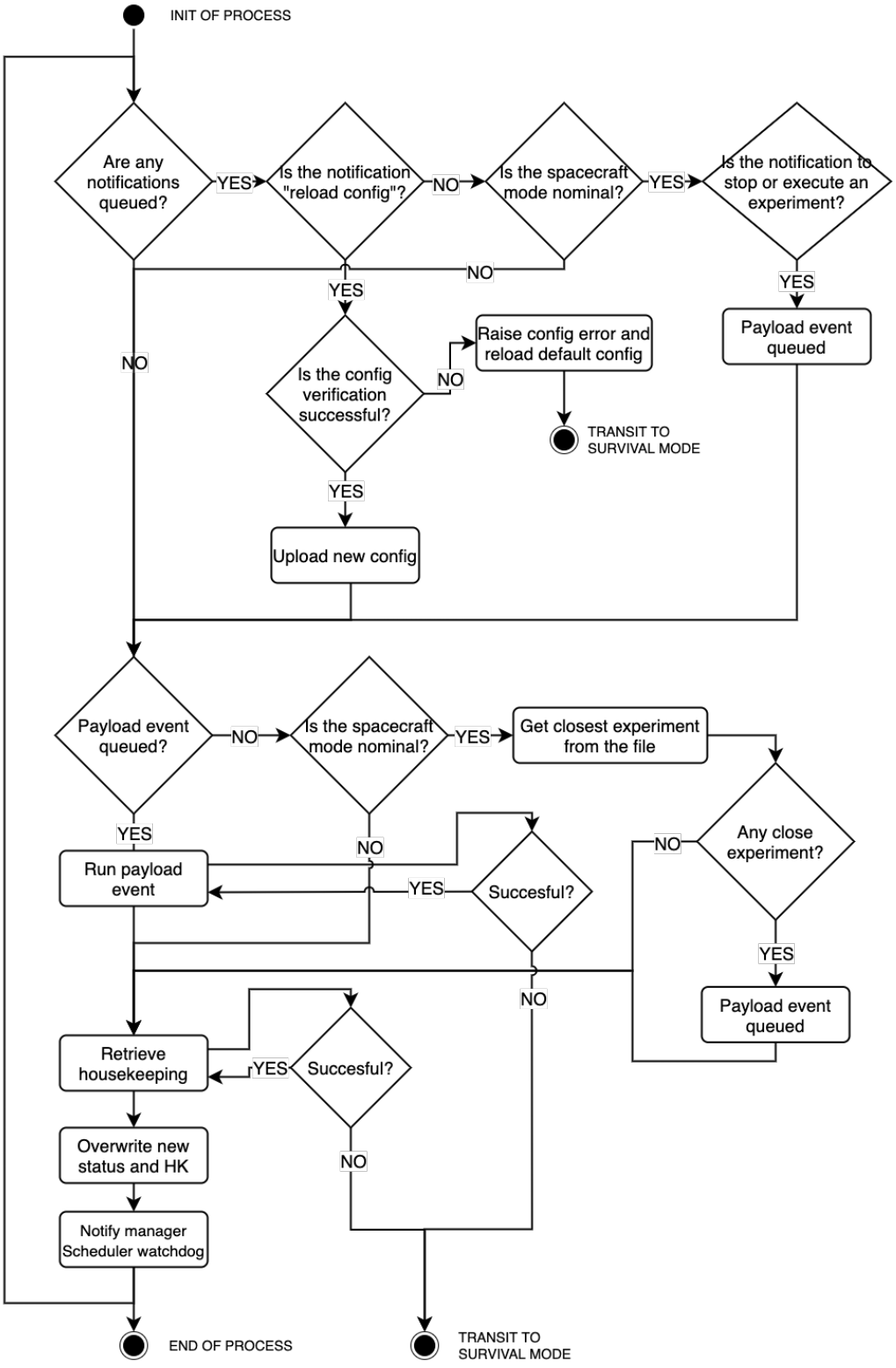


Figure 2.9: Payload process function flowchart.

2.1.7.2 Contributions to the Payload software

The contributions from this thesis to the Payload task software are the implementation of the error management layer and implement small modifications in the already coded func-

tions and logic. Although I haven't created new functions like in the previous tasks, I have spent many hours in this task understanding how its internal logic works. Nonetheless, I did implement the parsing of the experiment files, where the experiments are uploaded providing the experiment ID, start time and end time.

2.1.8 Manager task

This subsection covers the main role of the manager task. Its main objective is to manage the rest of the tasks and to transit the satellite between the modes of the state diagram (Section 1.3) .

In the manager's init function, all the tasks are initialized and the queues used in the system are created. Is the function that generates all the tools used to properly manage the spacecraft.

2.1.8.1 Manager process function

The manager process functions follows a similar structure regarding previously seen process functions. First, it checks for notifications, but only for the reload configuration one. If it is the case, the new configuration upload is performed.

Then, it executes three different functions: First, it processes all the notifications coming from the different tasks to update its watchdog timer. When received, the manager identifies from which subsystem the notification comes from and updates the corresponding value to prevent the rebooting. It is the health check of the system.

Then, it analyzes the error queue looking for a new notification. If there is any, it is processed and the spacecraft state is transited to Survival Mode. In nominal operations, this functions should not raise unless a exception is raised in any of the subsystems.

Finally, the manager task processes if there is any notification that request a transit of state. If it is the case, the manager checks which is its actual state and then evaluates if the transition requested is valid according to the logic implemented in the state diagram (Section 1.5).

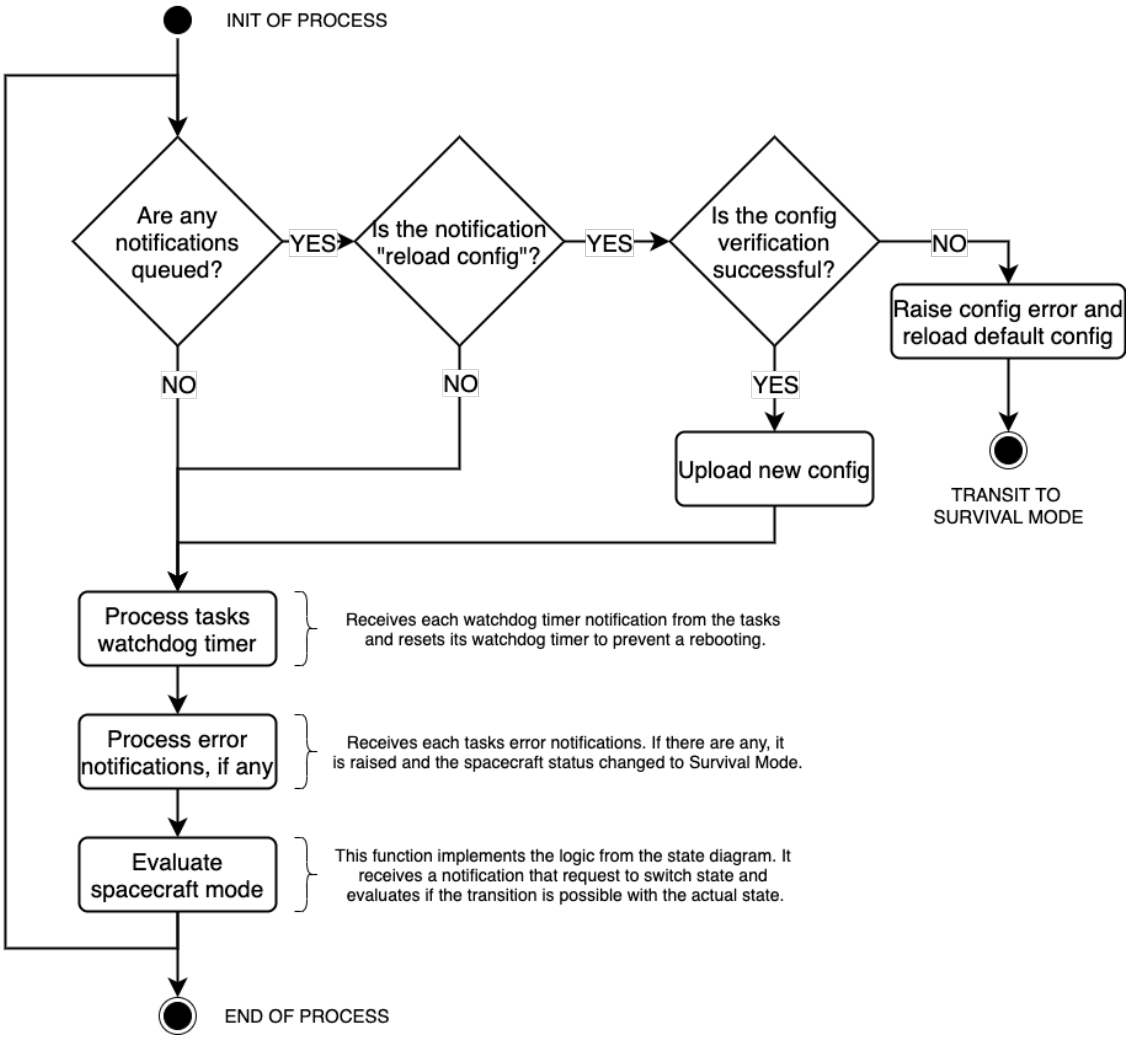


Figure 2.10: Manager process function flowchart.

As a conclusion, the manager process function is in charge of controlling and processing the notifications sent by the rest of the tasks.

2.1.8.2 Contributions to the Manager software

The contributions of this thesis to the Manager software are the modification of these two functions:

- **critical_event_handle():** This function is the one in charge of processing the error notifications from the tasks. When received, identifies the origin task and requests a transit to Survival Mode.
- **evaluate_mode():** This function is the one in charge of processing the state transit requests, where the state diagram logic is implemented.

2.2 Ground Segment

Ground segment software is considered all the software not running in space, onboard the spacecraft. Mainly, it is the software that manages the ground station but also the one that prepares and process the data that is sent to and received from the satellite.

In this thesis, the main contribution regarding the ground segment software is the implementation of dashboards for each subsystem in order to display all the data incoming from the satellite. This work will be explained in the following sections.

The NanoSat Lab operates a ground station located at Parc Astronòmic del Montsec (PAM), which has installed antennas to operate at UHF, VHF and S-band (2 to 4 GHz) frequencies.

Nonetheless, in order to contribute in the development and testing of the different missions, including 3Cat-4, an emulated ground station is installed in the NanoSat Lab's space in Barcelona. This station, normally referred as emulated ground station, is based on a computer that manages a SDR board (Figure 2.11), which is used as the radio module needed to communicate.

Figure 2.11: Picture of the Pluto SDR board used for the emulated ground station.

The emulated ground station is coded in C++, implements a (include type of RF encoding, etc.). For this thesis, I almost haven't contributed in the Ground Station software.

2.3 Operational Display

Once the telemetry data is received and processed in the ground station, this data must be displayed in a proper way in order to enable and facilitate the operation of the satellite when in orbit. For this reason, NanoSat Lab's team agreed to implement an open-source software called Grafana. Grafana is a complete software that manages analytics and interactive visualization of data stored in a time series database. It is widely used for the operations of amateur CubeSat missions like 3cat-4.

My contribution to the ground segment software is the creation and design of the Operational Display for the 3cat-4 mission. This has been separated in two parts. First, send

all the selected telemetry data from each subsystem to a local InfluxDB database using a socket connection. This is implemented in the ground link software. Second, create and design a dashboard that properly displays the required and relevant data of each subsystem in order to facilitate the operations of the satellite.

For example, in Figure 2.12 below it can be seen the first version of the ADCS operations dashboard. The most relevant parameters of the ADCS housekeeping data are represented in a structured and clear way.

Figure 2.12: First version of the ADCS dashboard using Grafana.

Although the dashboards for all the versions are started, future work needs to be done in order to improve them.

In the following chapter, some more screenshots of other versions of the display can be seen while evaulating some of the FlatSat tests.

CHAPTER 3. FLATSAT TESTING

FlatSat tests are a kind of tests performed in the moment before the integration and after subsystem test. When all the subsystems are developed and tested, they must be connected to the OBC to test the spacecraft as a complete system. Nonetheless, as things can go wrong during testing and unexpected modifications must be performed in the components, the team cannot integrate the satellite to perform these system tests. This is when FlatSat testing is performed.

FlatSat means that all the components and modules of the satellite are connected but not stacked (or integrated), normally using a FlatSat testbed or using jumper cables to interconnect the systems. FlatSat configuration enables to quick modification of connections while testing and easiness for reworking components.

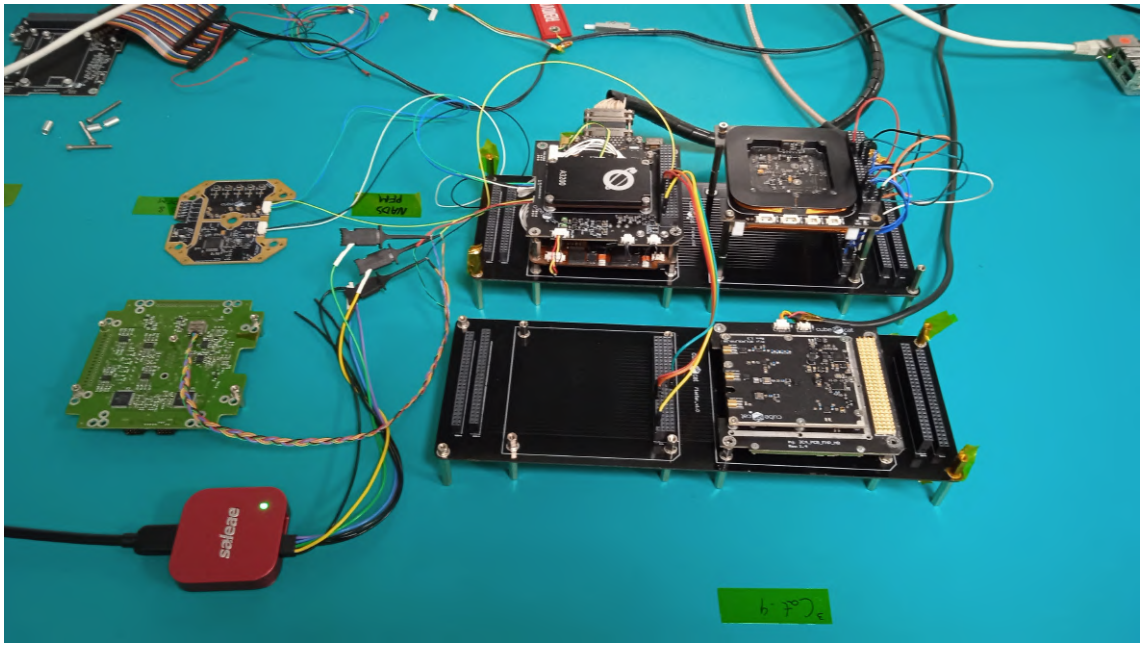


Figure 3.1: FlatSat configuration with two testbeds.

It is important to analyze the previous Figure 3.1. In the testbeds, we can find up to four modules: on the top left bed, there are the OBC stacked over the EPS; on the top right, there is the COMMS&ADCS board floating and connected using jumper wires. On the bottom bed, there is the FMPL-1 board.

Then, on the left part of the picture, we have the two deployment boards: first, at the top we can find the NADS deployment board and at the bottom the ZADS Engineering Model board.

Furthermore, we can spot the logic analyzer named *Saleae* in red, at the bottom-left of the image. It is used to debug the communications problems in the power lines and communi-

cation buses.

3.1 Methodology

In FlatSat testing is when all the work previously explained in chapter 4 is tested and verified. The methodology used to test each subsystem is explained hereunder:

1. Physical connection of the subsystem, mainly power lines and communication buses.
2. Ensure that the subsystem is correctly powered, checking EPS consumption.
3. Ensure proper communications between OBC and the subsystem, through UART or I2C buses.
4. Perform specific functionality tests.

Although it may seem easy, one of the key aspects in where I have spent lots of time resources is in communications between the OBC and the subsystem. Specifically, the I2C bus is the one that gives most of the problems, mainly because it is used by more than one subsystem, unlike UART.

I have spent several hours (Figure 3.2) understanding how I2C protocol works and debugging the communication buses. As a conclusion, most of the problems are caused by a wrong pull-up resistors configuration.

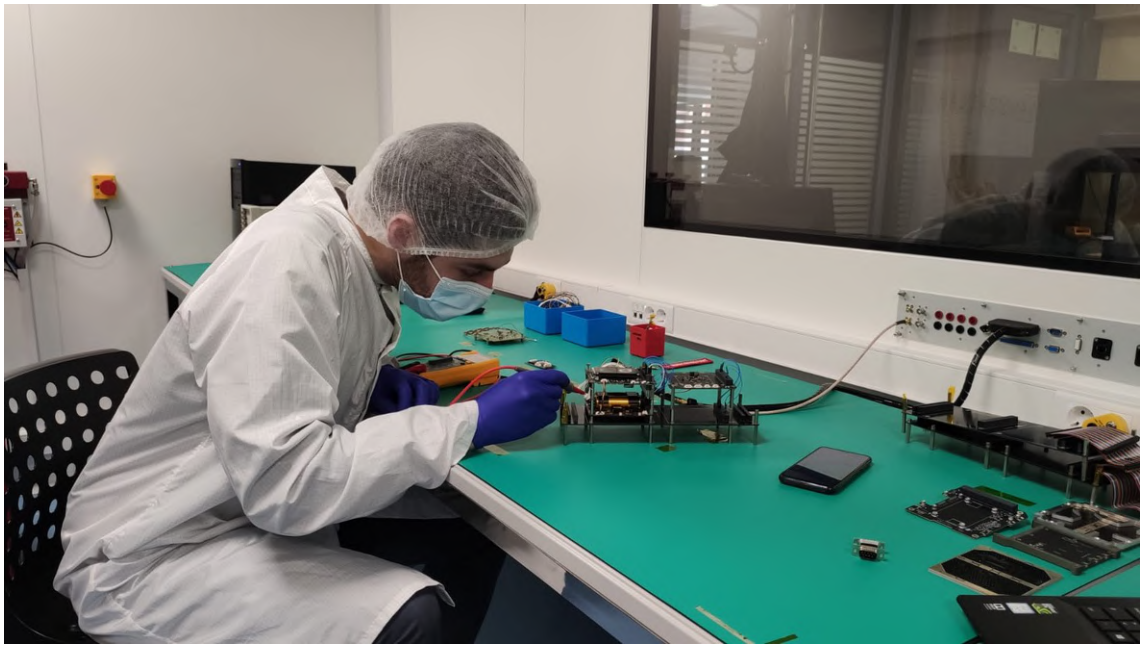


Figure 3.2: Process of debugging the FlatSat setup using a multimeter inside NanoSat Lab's cleanroom.

3.1.1 Facilities and equipment used

In order to properly perform some of the FlatSat tests, special equipment and facilities of the NanoSat Lab are used. They are explained briefly hereunder:

- **Cleanroom:** A cleanroom is an isolated room that has HEPA filtration to remove particles from the air. This type of facility are used for manufacturing where high levels of cleanliness and sterility are required. One of these applications is the manufacture and assembly of satellites. Dust can seriously interfere in the electronics and payloads. As a consequence, most of them are manipulated inside cleanrooms. NanoSat Lab's cleanroom is an International Organization for Standarization (ISO) 8 cleanroom, sufficient for the production of satellite platforms in most cases.

- **Thermal Vacuum Chamber (TVAC):** This facility is used to simulate space conditions. Two major space conditions are simulated: first, a compressor suck out the air to create vacuum. Then, a charge of liquid nitrogen cool down the chamber and with the help of infrared lamps, a temperature cycle is created. This cycle ranges between $-35^{\circ}\,C$ and $50^{\circ}\,C$, which are similar values that the satellite will experiment in a Low Earth Orbit (LEO). The TVAC is placed inside the cleanroom. In Figure 3.3 we can observe a picture of the TVAC opened, with the Mechanical Ground Support EquipmentMGSE inside and the structure that surrounds it, which are the infrared lamps:

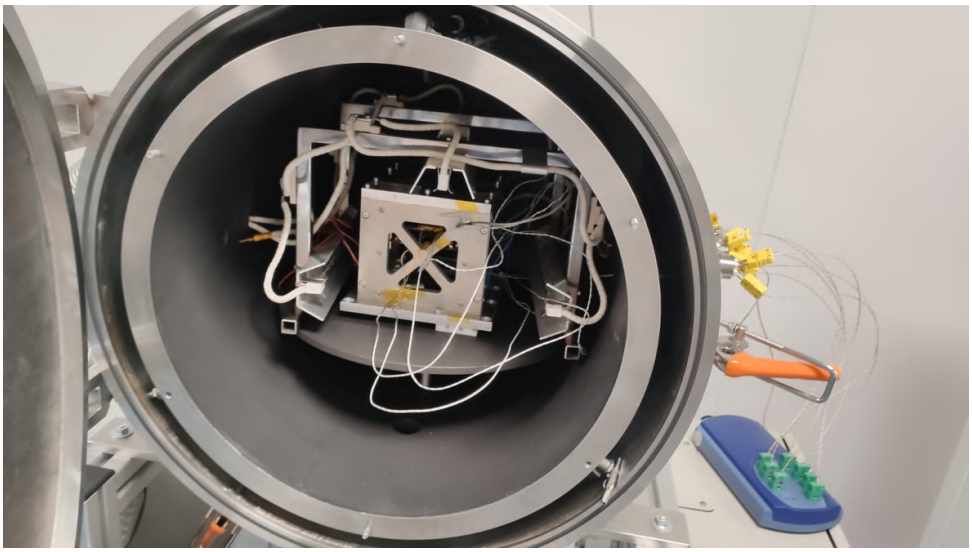


Figure 3.3: Picture of the TVAC opened with the 3Cat-4 MGSE inside.

- **Electric Ground Support Equipment (EGSE):** This equipment is the one used to work with the satellite while its developing. Its inputs are up to three different voltages from an electric supply and a JTAG picoblade to flash the satellite. As an output, it has an umbilical cable that goes through the cleanroom feed-through and to the satellite. It is used to provide external power, charge the batteries, flash the OBC and bypass the satellite kill switches. This equipment is designed by NanoSat Lab students. A picture of the EGSE can be checked at Figure 3.4 .

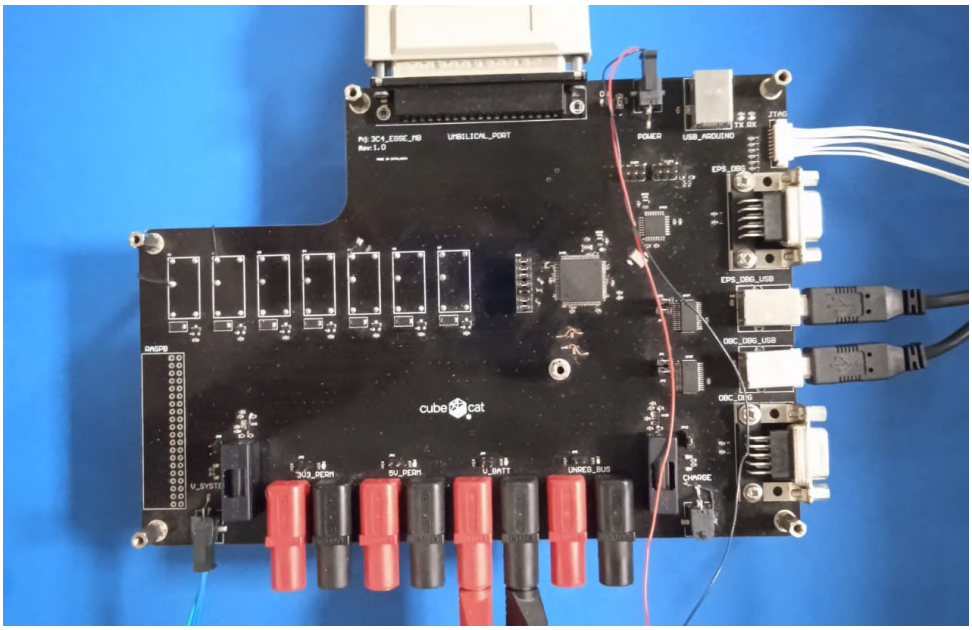


Figure 3.4: Picture of the EGSE, with the umbilical connection at the top, power supply connections at the bottom and flash and debug connections at the right.

- **Mechanic Ground Support Equipment (MGSE):** This equipment is an aluminium 7075 structure. Is the one used to place the satellite inside and acts as an interface for the shaker table. Furthermore, it provides thermal stability due to the mass increase and protection against strong infrared rays when performing TVAC tests. The MGSE is placed inside the cleanroom. This equipment is designed by NanoSat Lab students. A picture of the MGSE can be checked at Figure 3.5 .

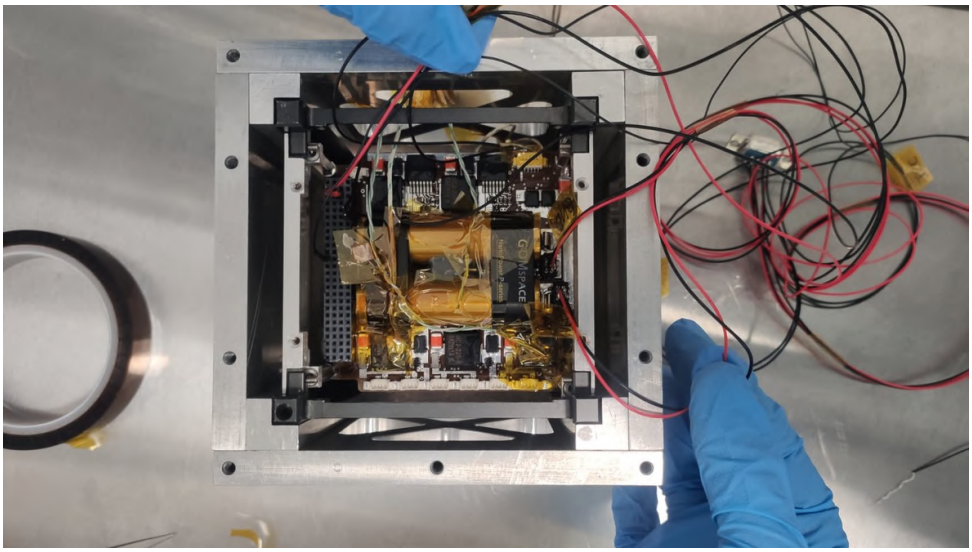


Figure 3.5: Picture of the 3Cat-4 MGSE with the EPS component inside, getting prepared for a TVAC test.

Although my main work as a software engineer of the mission is to ensure the points stated above, I also contributed providing support in more complex tests performed in the subsystems. These tests needed also the assistance of a hardware engineer in order

to prepare the assembly of the TVAC or the refurbishment of specific parts. Find stated hereunder this specific FlatSat tests explained in detail.

3.2 EPS heater test

This section covers the test performed in order to verify the EPS heater automatic system. This is a system test, so the EPS is behaving as slave in respect to the OBC, which is master.

3.2.1 Test objectives

The objective of this test is to ensure that the subsystem built-in heater is turned on automatically when the temperature reading from the sensor placed in the batteries drops below the low hysteresis temperature. Furthermore, when the same temperature reading reaches the high hysteresis temperature, the heater should turn off. The hysteresis temperature values should be set to the EPS by uploading a configuration file to the OBC, which has to upload this configuration to the EPS.

The pass/fail criteria of the test is if the system is able to behavior as expected or not.

3.2.2 Test setup

In order to simulate the temperature cycles that the satellite will experience in orbit, the TVAC of the NanoSat Lab must be used. Thus, the EPS subsystem is mounted inside the MGSE (Figure 3.5) and then mounted inside the TVAC (Figure 3.3).

The OBC will not be mounted inside the TVAC because the umbilical cable between the OBC and the Electric Ground Support Equipment is needed. The umbilical cable cannot be entered the TVAC due to the high number of pins needed, as the number of pins that can be passed through the TVAC feedthrough is strictly limited. Also, it is not feasible to try to communicate by an RF link because the TVAC acts as a closed echo chamber.

In consequence, a special cable that connects OBC to EPS is manufactured in order to perform this test. This cable is split in two parts, outside and inside the TVAC. The external one does not have any specific requirement and the internal part must be made of Teflon-recovered wires, in order to withstand the wide temperature range inside the chamber.

Once everything is mounted and secured inside the TVAC, a communications test is performed. This means that from the coding computer outside the cleanroom we must be able to properly communicate with OBC and EPS, ensuring that all the connections are correct and working.

Then, the next step is to close TVAC's hatch and turn on the vacuum mechanism. It is necessary to reach down to 10^{-5} bars of pressure. When reached, the TVAC's shroud can be loaded with the needed liquid nitrogen.

After 2 to 3 hours, the inner temperature of EPS subsystem will approach the target low hysteresis temperature, when the test will be performed.

The setup diagram is stated in Figure 3.6 below:

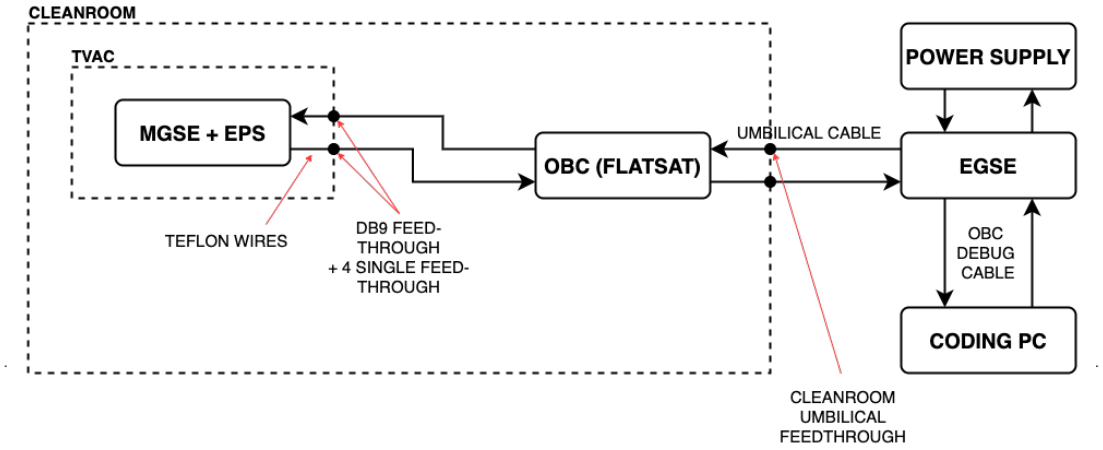


Figure 3.6: Diagram that represents the setup for the EPS Heater test.

3.2.3 Test results

After reaching the low hysteresis temperature, the heater turned on at the configured low hysteresis temperature of $2^{\circ}\ C$ and remained powered until the same sensor reading reached $5^{\circ}\ C$, as this is the configured high hysteresis temperature. The cycling between this range lasted for some hours, while liquid nitrogen was slowly evaporating.

We were able to obtain several valuable data from the telemetry of the EPS subsystem. This telemetry was displayed using a first version of the operational dashboard for the EPS, where three parameters were the most relevant to ensure the correct behavior of the EPS heaters. Please mind that the following plots are directly obtained from the real time display, thus the x-axis does not start at 0.

The first and most important one, temperature readings of the temperature sensor placed in the EPS batteries, in Figure 3.7 below.

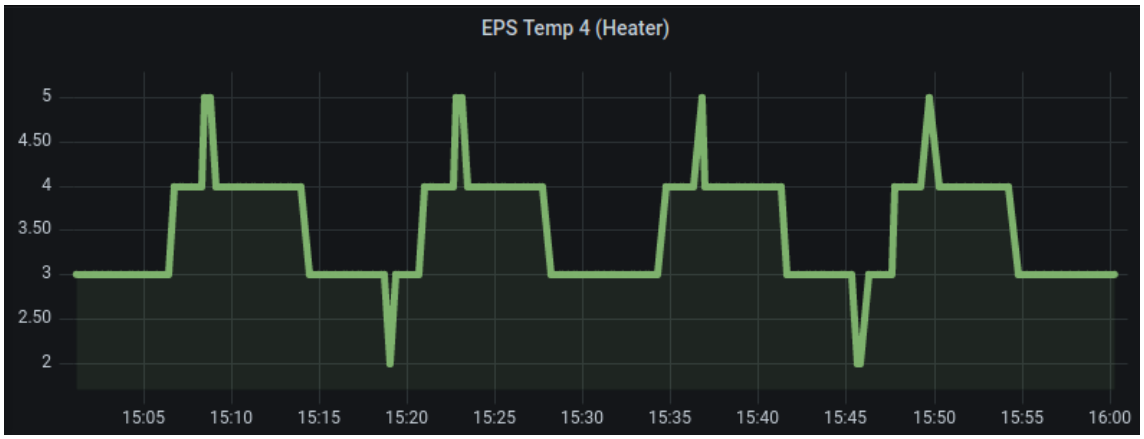


Figure 3.7: Snapshot from the EPS dashboard plotting the EPS temperature sensor values over time.

As observed, the rise time is much quicker than the fall time. Rise lasts up to 4 minutes

while fall lasts up to 10 minutes, approximately both. Nonetheless, the fall time tends to last longer each cycle due to the loss of thermal inertia of the liquid nitrogen.

Also, we can verify that the voltage range of the batteries is nominal (Figure 3.8) and that the current consumption of the heater peaks to almost 400 mA (Figure 3.9).

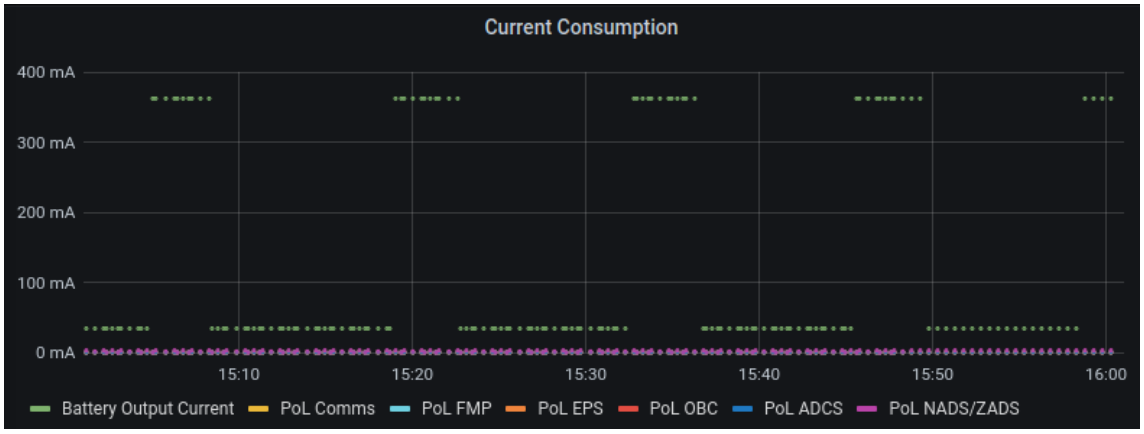


Figure 3.8: Snapshot from the EPS dashboard plotting the EPS battery current consumption sensor values over time.

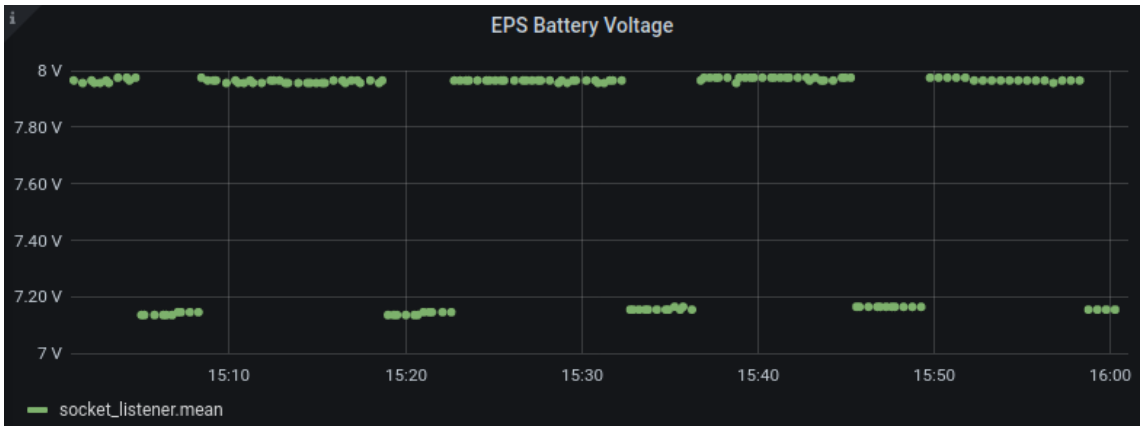


Figure 3.9: Snapshot from the EPS dashboard plotting the EPS battery voltage values over time.

Moreover, during the experiment the hysteresis values were changed several times to ensure that the OBC was able to properly parse the new values and set to the EPS subsystem software during the experiment timespan.

3.2.4 Test conclusions

The EPS heater works nominally. The behavior is as expected, maintaining the temperature of the batteries inside the hysteresis range to prevent damage due to low temperatures. Also, the consumption is nominal and there is no risk to drain the batteries. Furthermore, it is possible to correctly modify the hysteresis temperature values through the configuration uploaded to the OBC.

Concluding, the EPS heater will contribute on the thermal stability of the 3Cat-4 spacecraft. A more detailed study of the EPS subsystem and specifically the heater test is held by my colleague Albert Rodríguez in his final degree thesis [18].

3.3 RF chain test

This section covers the test performed in order to verify the RF communication chain between the satellite and the emulated ground station.

3.3.1 Test objectives

The objective of this test is to verify the capacity of the system to receive telecommands and transmit beacons to the emulated ground station.

The pass/fail criteria of the test is if the system is able to properly communicate with the emulated ground station.

3.3.2 Test setup

The setup of this test requires an extra equipment of the NanoSat Lab: The emulated ground station. This is a computer with a Software Defined Radio module, model Adalm-Pluto, that replicates the RF chain of the Montsec Ground Station. This way, we can mock-up the functional behavior of the ground station from the lab using an easy setup.

For simulating the satellite part, a coaxial cable brings the antenna outside the cleanroom for more ease-of-use (Figure 3.10). Also, a 30 dB attenuator is added to the coaxial cable in order to reduce the power of both transmitted and received signals as, due to short distance between transmitter and receiver, it would damage components of the RF chain. Also, the multipath caused due to transmitting inside a room would also heavily saturate the channel.

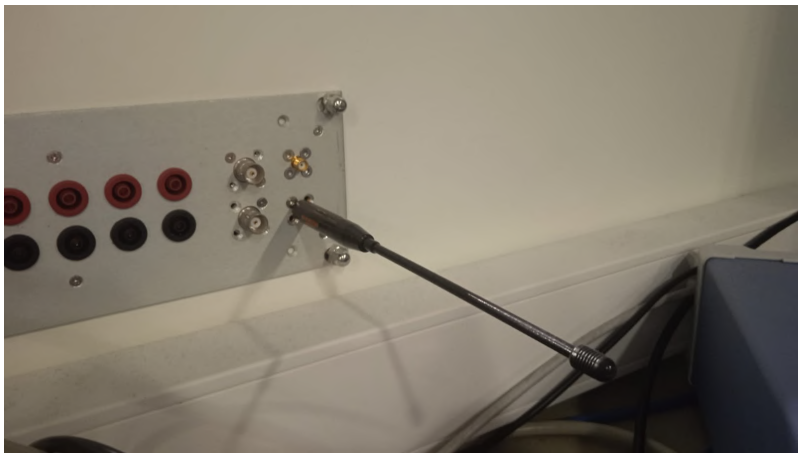


Figure 3.10: Antenna connected to the cleanroom's coaxial feedthrough that represents the satellite's antenna.

At a software level, we must execute some scripts to start the emulated ground station. First, we must connect using ssh to the ground station computer. There, we must execute the following two command:

```
./nanosat_uhf.py -d 437.35M -u 437.35M
./cubecat4_block_test.py
```

This executes two Python scripts that simulate the ground segment chain using GNUradio software and the Pluto board. The specified frequency values are the ones for downlink (-d option) and uplink (-u option).

Also, in order to verify all the steps, a UHF visualizer is initiated in order to observe if there is a communication between the two blocks. The software is executed with the command:

```
./UHF_visualizer_3cat4.py
```

The setup diagram is stated in Figure 3.11 below:

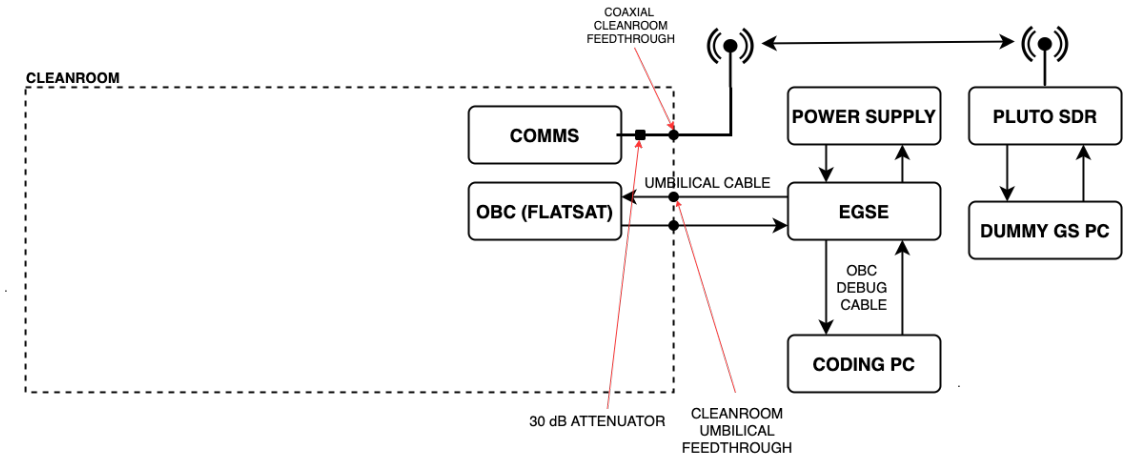


Figure 3.11: Diagram that represents the setup for the RF test.

3.3.3 Test results

The system behaves correctly. It is able to send telecommands from ground station and receive them at communications board. Also, the beacon and thus, telemetry is received and processed accordingly, displaying it to the operational display.

Nonetheless, sometimes a telecommand is lost.

3.3.4 Test conclusions

As a conclusion, the RF chain is validated and works as expected. Nonetheless, we must take in consideration the loss of some of the packets sent from emulated ground station. A probable explanation is that the attenuator is maybe too strong. Also, multipath signals probably interfere and in some occasion may contribute to this packet loss.

When the satellite will be integrated and when validating again the RF communications,

further checks must be performed in order to characterize this packet loss.

3.4 ADCS high-sampling mode and functional chain test

This section covers the test performed in order to verify the mission test mode of the ADCS subsystem. Also, the correct functioning of the ADCS logic is tested, as a movement is induced in the FlatSat to observe the reaction of the detumbling system.

3.4.1 Test objectives

This test has two objectives: First, the validation of the new implemented mission test mode, which performs a high-speed sampling of the ADCS parameters and sends them directly to the ground station.

Then, the validation of the ADCS chain that takes as an input the magnetometers reads, compares them to the IGRF model and powers the magnetorquers in order to perform the needed torque. Nonetheless, in this test the B-Dot detumbling algorithm will not be tested as the setup does not allow it.

3.4.2 Test setup

The setup of this test consist in having the FlatSat modules connected as usual. At least, it is required to have OBC, EPS and COMMS&ADCS board. The three components are in the same FlatSat testbed.

Then, the OBC must be connected to the EGSE with the umbilical cable and the EGSE correctly connected to the coding computer. The testbed must have freedom of movement, in order to perform the required moves for the test.

A diagram of the setup can be find in Figure 3.12 below.

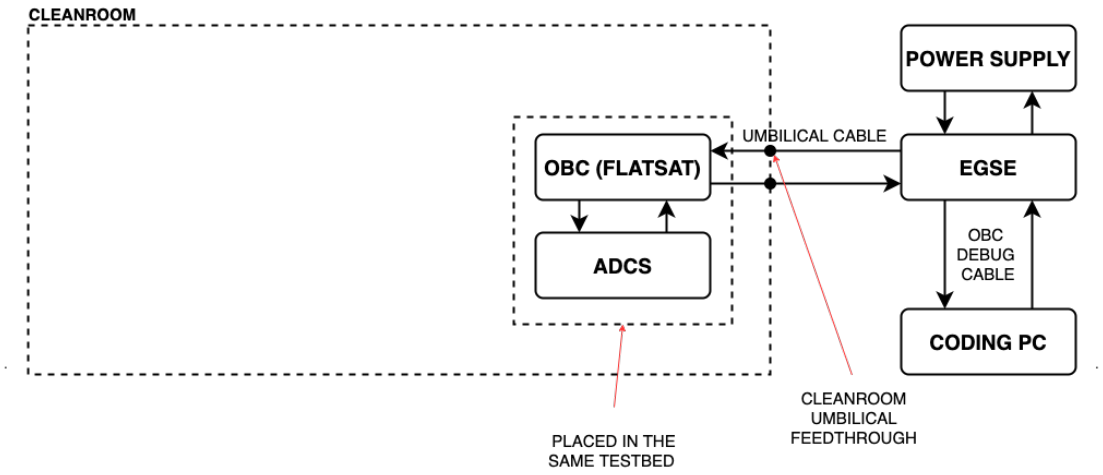


Figure 3.12: Diagram that represents the setup for the ADCS test.

For the execution of this test, I entered the cleanroom and picked up the FlatSat testbed. Then, I started moving it slowly along the three axis. Finally, I performed quicker moves in order to spot the difference.

3.4.3 Test results

The test is evaluated observing the behavior of the different parameters plotted in the ADCS dashboard of the operational display. In this test, there are three parameters to control:

1. **Gyroscopes:** This parameter represents the acceleration measured by the gyroscopes of the ADCS board. The units are $^{\circ}/s$.
2. **Control values:** This parameter represents the percentage of saturation sent to the actuators of the system, in this case the magnetorquers. The unit is a percentage, being 0 % no actuation and 100% maximum actuation.
3. **EPS Consumption:** This parameter represents the current consumption of the satellite's batteries.

In Figure 3.13, it can be appreciated how these three parameters value during the test timespan, which is between 16:49:40 and 16:51:48, around the two minutes:

Figure 3.13: Snapshot from the ADCS dashboard plotting the gyroscopes acceleration, control values and EPS current consumption values over time.

Analyzing the results, we can identify two different parts: first, between the beginning of the test until 16:50:30, the movements of the FlatSat testbed are slow, derived from the readings of the gyroscopes. In consequence, the values of the control values are low most of the time, as it is the EPS current consumption.

Then the second part, starting from 16:50:30 until the end of the test, more agressive movements have been exerted on the testbed, resulting in higher accelerations and thus, higher control values in all the three axis. As a consequence, the EPS current consumption peaked four times up to 250 mA of total consumption. This current peak is caused by the magnetorquers, which require more power to create a stronger magnetic field that detumbles the satellite.

3.4.4 Test conclusions

The test results are coherent with the expectations, correctly actuating the ADCS magnetorquers in function of the acceleration read by the gyroscopes.

When the satellite is integrated, another test will be performed to ensure that the ADCS B-Dot algorithm is capable of correctly detumbling and control the attitude of the spacecraft.

Also, the test mode has worked as expected, increasing the sampling rate to obtain a more accurate read of the ADCS sensors.

3.5 NADS deployment test

This section covers the test performed in order to verify the NADS deployment as part of the FlatSat.

3.5.1 Test objectives

The main objectives are to successfully deploy the NADS using the nominal command sequence. Although a NADS deployment has been tested before, it was at subsystem level, which means that it was only testing the subsystem software. In this test, the NADS is connected to the OBC as part of the system. Furthermore, there are no tests executed that included the NADS burning PCB, required for the correct implementation of the NADS subsystem into the FlatSat.

3.5.2 Test setup

The setup of this test consisted in having the NADS refurbished ready to deploy and mounted on the auxiliary structure that simulates the zero gravity condition by hanging the NADS against gravity to force a straight deployment. Then, the NADS is connected to the OBC and this last one connected to the computer used for development. All the equipment is placed in the cleanroom.

The NADS antenna is stowed thanks to five dyneema wires that subjects different parts of the mechanism. To deploy the antenna, a resistor or a khantal cable (there is a re-

dundancy) that is placed touching the dyneema wire is short-circuited and, due its low resistance, warms up to more than $150^{\circ}\ C$. This cuts the dyneema wire and releases the compressed part of the antenna.

The burning of the resistor/khantal wire is performed individually, in order to totally control the operation when in orbit. Having the full deployment divided in five steps allows the operator to control more specifically the situation and solving of a possible problem, such as battery drain.

The setup diagram is stated in Figure 3.14 below:

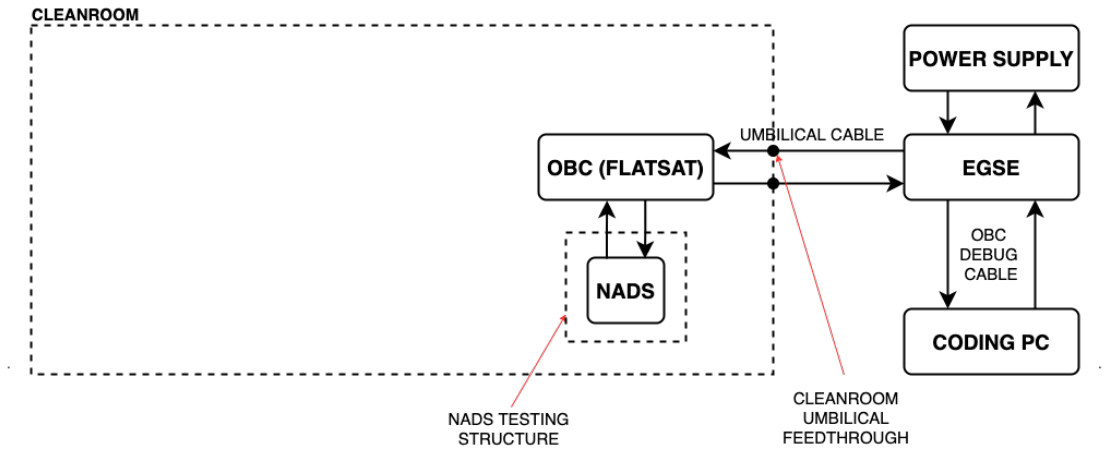


Figure 3.14: Diagram that represents the setup for the NADS deployment test.

3.5.3 Test results

The result of the test is the correct deployment of the NADS antenna. As it can be seen in the following figures, the sequence has been followed and performed as expected.

The initial state of the NADS stowed is the one pictured in Figure 3.15.

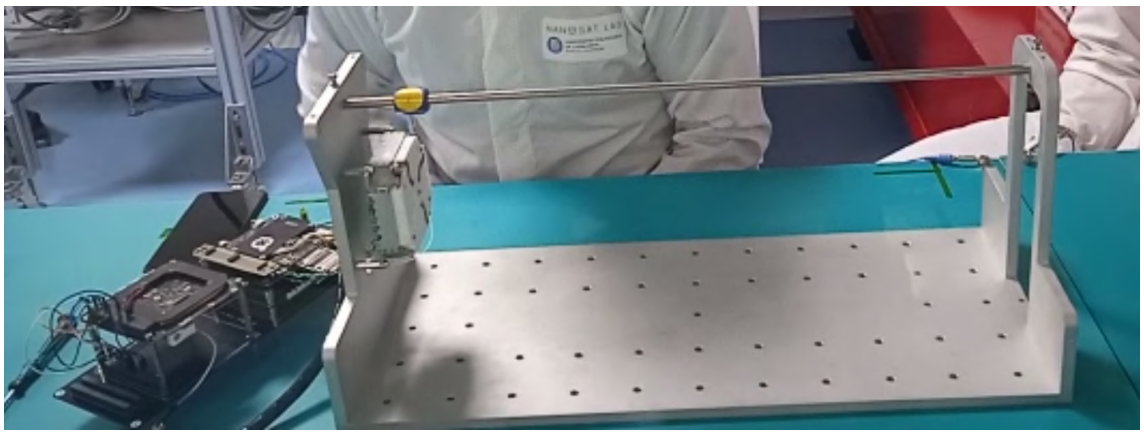


Figure 3.15: Picture of the initial state of the NADS deployment test.

Then, the first two dyneema wires are cutted, sending the command to burn the first khantal wire and then the second. The consequence is the release of the fingers that holds the

gravity boom (white piece), as it can be seen in Figure 3.16. Notice that in the NADS, the arms are now released. Also notice that at the right part of the picture, there is a small round white piece. This is the piece that the two dyneema wires hold on tight to the satellite and when these are cutted, the piece is ejected.

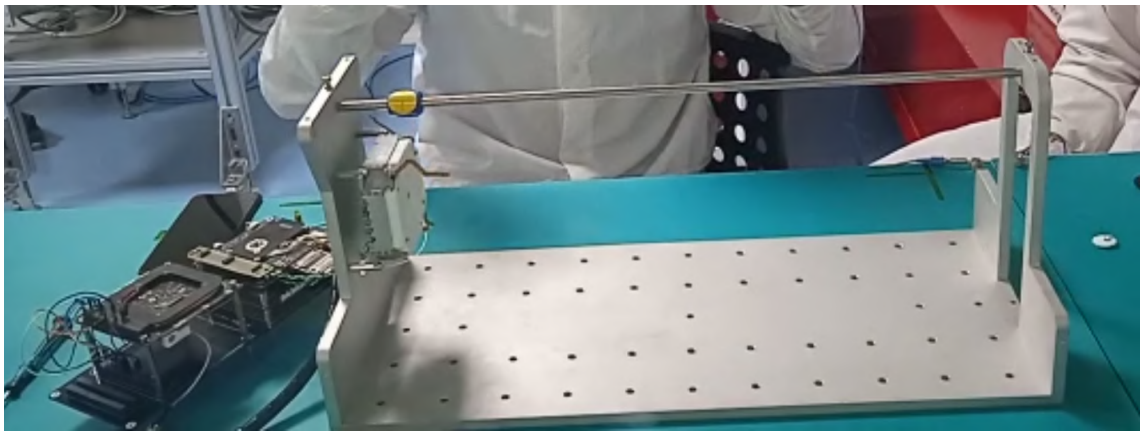


Figure 3.16: Picture of the fingers deploy of the NADS deployment test.

Hereafter, the dyneema wires three and four are burned, using its respective telecommands. When the last one is burned, the antenna deploys to half of its total length, around 25 cm. In Figure 3.17 this deployment can be observed.

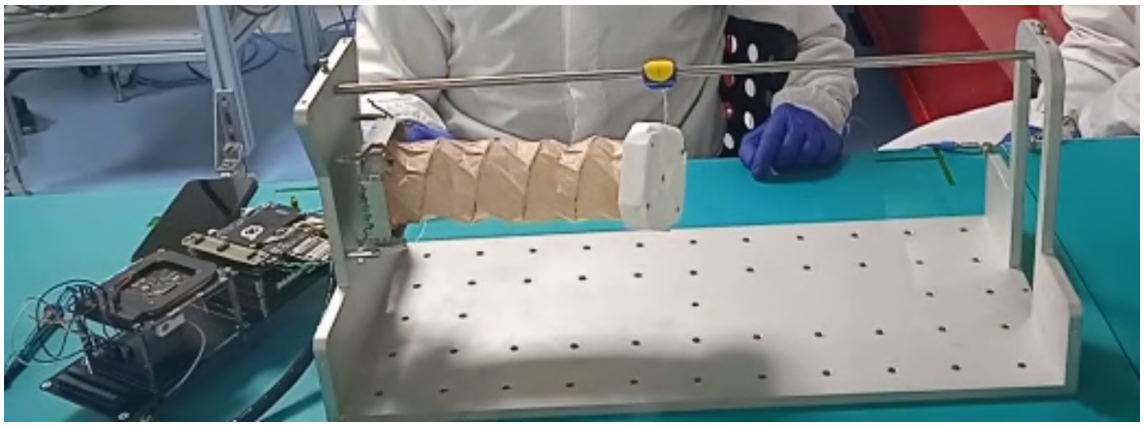


Figure 3.17: Picture of the half deployment of the NADS deployment test.

Finally, there is one last dyneema wire holding the last section of the antenna, which is linked to resistor five. When the command is sent to burn this resistor or khantal, the antenna fully deploys, as seen in Figure 3.18.

Figure 3.18: Picture of the full deployment of the NADS deployment test.

3.5.4 Test conclusions

As a conclusion, the test has been a success. All the steps were nominal and the behavior of the system has been the expected one.

Now, another test should be executed but not in ambient conditions as this one. The NADS should be deployed in the TVAC to verify that the deployment system also works in other thermal conditions.

CONCLUSIONS

Conclusions and objectives evaluations

As a conclusion, this thesis has greatly contributed to the development of the 3Cat-4 mission. The software implementations resulting of this work deliver a more robust system and an increased autonomy. Thanks to the error management layer added, the odds of a critical failure that could ruin the mission are reduced. Also, the operations efficiency has increased with the addition of the dashboard to control each subsystem from ground. Moreover, the implementation of the state diagrams logic provides the plus of autonomy required for the project. Finally, the FlatSat tests performed served to verify that all the implementations done were correct and to ensure the proper functioning of the system.

The objectives of this work were set formerly in Objectives section in the Introduction part . Now, the defined objectives are evaluated:

- **Contribution to the flight software development of the 3Cat-4 mission.**
  - **Implement an error management layer to the system.**

    Achieved. The error management layer has been successfully implemented thanks to the implementation of a general function to raise the errors of the subsystems to the manager. Also, the implementation of the reference error mask in each subsystem allows the operator to fully control the error flow.
  - **Implement new specific subsystem functionalities for a better management of the system.**

    Achieved. Several new functions have been added to the subsystems in order to improve its functioning. Some examples are the watchdog timer management functions of the EPS, the reset command files in COMMS, the test mode and TLE configuration in ADCS and the GPIO safety feature in the NADS deployment.
  - **Implement a visual solution to improve the operation of the mission.**

    Achieved. The implementation of the Grafana software to create dashboards for each subsystems has been successfully integrated on the ground software of the mission. Nonetheless, the created dashboards need iterations to improve the experience of the satellite operations.
- **Conduction of tests using the FlatSat configuration, in parallel with the software development.**
  - **Test the error management layer implemented.**

    Achieved. The error management layer has been tested in each subsystem, forcing all the errors to verify that the behavior is the expected one.
  - **Test all the previous and new functions of the system.**

    Achieved. All the functions of each subsystem has been tested as part of the FlatSat testing, which requires to validate each function.

- **Test the new implemented visual solution for operations.**

  Achieved. The implementation of the dashboards for the operations has been tested during the FlatSat testing and has resulted in a very useful tool for some tests such as the ADCS test mode and detumbling algorithm (Section 3.4).

At a personal level, this thesis has awarded me invaluable hands-on experience in a real space project. From software to hardware, the lessons learned are many and in several levels. Also, the possibility to work with a multidisciplinary team of engineers provided priceless teamwork skills, which I highly cherish. Finally, although this thesis has come to an end, I will continue my collaboration with the 3cat-4 mission and the future endeavours of the NanoSat Lab.

Future work

Although much work has been done to the 3Cat-4 mission, there are still a few steps left until its culmination.

Once the satellite integration is finished, the test campaign starts with the Ambient Test Campaign (ATC), which is about performing two different tests: (1) Full Functional Test (FFT), which requires a test of all the functionalities of the satellite but where the umbilical tether is allowed; (2) Mission Test (MT), which is a test similar to the FFT but the satellite has to be completely managed as if it was in orbit. This means that the team is not able even to see the satellite, only monitor it through the telemetry received.

Then, the satellite is required to pass an Environmental Test Campaign (ETC), which will be held at ESA's European Space Security and Education Centre (ESEC) facilities in Belgium, where special facilities and support from technical staff will be received.

In parallel, the operations manual would be required to prepare for the operations of the satellite once launched.

Finally, the predicted launch of the satellite is projected towards the middle of the next year, 2022. Nonetheless, there is no yet a confirmed ride by ESA, the agency in charge of the launch.

BIBLIOGRAPHY

[1] Carles Araguz López. *In pursuit of autonomous distributed satellite systems*, 2019. Doctoral Thesis, available at http://hdl.handle.net/2117/175253.

[2] National Aeronautics and Space Administration. *What are SmallSats and CubeSats?*, 2015. www.nasa.gov/content/what-are-smallsats-and-cubesats.

[3] Guillem Anglada. *Photographies from the integration of 3Cat-4 mission*. Only availabe under request to NanoSat Lab.

[4] Alicia Johnstone. *Cubesat Design Specifications Rev. 14*, 2020. https://www.cubesat.org/cds-announcement.

[5] Planet Labs. *Investor Presentation*, July, 2021. https://www.planet.com/investors/presentations/2021/investor-presentation-20210707.pdf.

[6] National Aeronautics and Space Administration. *NASA Cubesat Launch Initiative*. https://www.nasa.gov/content/about-cubesat-launch-initiative.

[7] European Space Agency. *ESA CubeSats*. https://www.esa.int/Enabling_Support/Preparing_for_the_Future/Discovery_and_Preparation/CubeSats.

[8] A. Camps, A. Golkar, A. Gutierrez, J.A. Ruiz de Azua, J.F. Munoz-Martin, L. Fernandez, C. Diez, A. Aguilella, S. Briatore, R. Akhtyamov, and N. Garzaniti. FSSCAT, the 2017 Copernicus Masters' "ESA Sentinel Small Satellite Challenge" Winner: A Federated Polar and Soil Moisture Tandem Mission Based on 6U Cubesats. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 8285–8287, 2018.

[9] UPC-NanoSat Lab. *UPC-NanoSat Lab webpage*. nanosatlab.upc.edu.

[10] Roger Jove-Casurellas, Carles Araguz, Pol Via, Arnau Solanellas, Adrià Amézaga, David Vidal, Joan Francesc Muñoz, Marc Marí, Roger Olivé, Alberto Saez, et al. 3cat-1 project: A multi-payload cubesat for scientific experiments and technology demonstrators. *European Journal of Remote Sensing*, 50(1):125–136, 2017.

[11] Jordi Castellví, Adriano Camps, Jordi Corbera, and Ramon Alamús. 3Cat-3/MOTS nanosatellite mission for optical multispectral and GNSS-R earth observation: Concept and analysis. *Sensors*, 18(1):140, 2018.

[12] J.A. Ruiz-de Azua, J.F. Muñoz, L. Fernández, M. Badia, D. Llavería, C. Diez, A. Aguilella, A. Pérez, O. Milian, M. Sobrino, A. Navarro, H. Lleó, M. Sureda, M. Soria, A. Calveras, and A. Camps. 3Cat-4 Mission: A 1-Unit CubeSat for Earth Observation with a L-band Radiometer and a GNSS-Reflectometer Using Software Defined Radio. In *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*, pages 8867–8870, 2019.

[13] J. F. Munoz-Martin, N. Miguelez, R. Castella, L. Fernandez, A. Solanellas, P. Via, and A. Camps. 3Cat-4: Combined GNSS-R, L-Band Radiometer with RFI Mitigation, and AIS Receiver for a I-Unit Cubesat Based on Software Defined Radio. In *IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, pages 1063–1066, 2018.

[14] Miquel Sureda, Marco Sobrino, Oriol Millan, Andrea Aguilella, Arnau Solanellas, Marc Badia, Joan Francesc Munoz-Martin, Lara Fernandez, Joan A. Ruiz-De-Azua, and Adriano Camps. Design and Testing of a Helix Antenna Deployment System for a 1U CubeSat. *IEEE Access*, 9:66103–66114, 2021.

[15] Ars Technica. *What operating systems keep things running in space?*, February, 2020. https://arstechnica.com/features/2020/10/the-space-operating-systems-booting-up-where-no-one-has-gone-before/.

[16] NASA. *NASA's manual of flight software for small sats*. https://www.nasa.gov/smallsat-institute/sst-soa-2020/flight-software.

[17] AI Solutions. *Norad Two-Line Orbital Element Set File*. https://ai-solutions.com/_help_Files/two-line_element_set_file.htm.

[18] Albert Rodríguez Casellas. *Thermal analysis and testing for CubeSat based missions*. *UPC-BarcelonaTech*, 2021.

APPENDICES

APPENDIX A. SOFTWARE RELATED CONTENT

In this appendix some context will be given regarding the chapter 1 of the memory, Software Development.

A.1 EPS task

A.1.1 EPS configuration parameters

The EPS has up to 16 configurable parameters, explained below. Find first the full name and in parenthesis the variable name as used in code.

1. **Sunsafe to nominal threshold (ss_to_nominal_th):** Threshold in miliVolts that delimitates the Sunsafe and Nominal modes. If the battery voltage is higher, the mode will be Nominal. Normally this value is set to 8000 mV.

2. **Nominal to sunsafe threshold (nominal_to_ss_th):** Threshold in miliVolts that delimitates the Sunsafe and Nominal modes. If the battery voltage is lower, the mode will be Sunsafe. Normally this value is set to 7300 mV.

3. **Power-Point Tracking (PPT) mode (ppt_mode):** Mode of the power-point tracking, which can be 1 (Maximum Power-Point Tracking) and 2 (SW Fixed Power Point Tracking). In our mission, it is set to 1, which is automatic.

4. **Battery's heater mode (battheater_mode):** Mode for activating the battery's heater. If manual operations are intended, it is set to 0, if automatic operations are planned, it is set to 1. In our mission, this value is set to 1 as its activation is managed by software.

5. **Low hysteresis temperature (low_hyst_temperature):** Lower bound of the temperature range that the EPS is set to oscillate when the ambient temperature drops below zero degrees. The units are celsius degrees and the usual value is set to 2.

6. **High hysteresis temperature (high_hyst_temperature):** Higher bound of the temperature range that the EPS is set to oscillate when the ambient temperature drops below zero degrees. The units are celsius degrees and the usual value is set to 5.

7. **EPS normal point of load values (eps_normal_pol_values):** 8bit bitmask that indicates which points of load must be up when the subsystem is operating in normal regime. There are no units, it is a decimal value that ranges from 0 to 255 that when converted to binary each position correspond to a point of load of the EPS. In our case, it is set to decimal value of 61.

8. **EPS safe point of load values (eps_safe_pol_values):** 8-bit bitmask that indicates which points of load must be up when the subsystem is operating in safe regime. There are no units, it is a decimal value that ranges from 0 to 255 that when converted to binary each position correspond to a point of load of the EPS. In our case, it is set to decimal value of 13.

9. **On delay (on_delay):** Delay in seconds in between the EPS power on moment and the switchable outputs power on. In our case, it is set to 0 seconds.

10. **Off delay (off_delay):** Delay in seconds in between the EPS power of command and the switchable outputs power off. In our case, it is set to 0 seconds.

11. **Voltage boost (v_boost):** Fixed PPT point for boost converters, in mV. In our case, it is set to 3700 mV.

12. **Battery maximum voltage (batt_maxvoltage):** Upper bound of battery voltage for a normal operation. Units in miliVolts, usually set at 8300 mV.

13. **Battery normal voltage (batt_normalvoltage):** Minimum value of battery voltage for operating in normal regime. Below this threshold value, regime changes to safe. Units in miliVolts, usually set at 7800 mV.

14. **Battery safe voltage (batt_safevoltage):** Minimum value of battery voltage for operating at safe regime. Below this threshold value, regime changes to critical. Units in miliVolts, usually set to 7000 mV.

15. **Battery critical voltage (batt_criticalvoltage):** Minimum value of battery voltage for operating at critical regime. Below this threshold value, regime changes to decommission. Units in miliVolts, usually set to 6500 mV.

16. **EPS reference error mask (eps_reference_error_mask):** 8-bit bitmask that indicates which errors must be ignored and which not. Each position is linked to a specific error and if the mask value is 0, the error is bypassed and not sent to the manager task. In our case, it is set to decimal value of 255, which means that there are no bypassed errors.

A.1.2 EPS housekeeping parameters

The EPS instantaneous telemetry is formed by the parameters explicited below. The total size of the telemetry is of 87 bytes. Find first the full name and in parenthesis the variable name as used in code.

- **Voltage boost (vboost[3]):** Vector of three positions that encapsulates the values of voltage of the boost converters PV1, PV2 and PV3.
- **Battery voltage (vbat):** Current voltage of the battery, in mV.
- **Current in (currin[3]):** Input current, in mA.
- **Current boost (currboost):** Current from boost converters, in mA.
- **Battery current (currbatt):** Current out of battery, in mA.
- **Current out (currout[6])** Current out from the six switchable outputs (points of load) in mA.
- **Output (output):** Bitmask status of the output. Each bit is linked to a point of load, if the bit is 1 it means that is up, otherwise is down.

- **Latch count (latch_count):** Number of latch-up protections, linked to each point of load.
- **I2C watchdog time left (wdt_i2c_time_left):** I2C watchdog time in seconds that is left until zero is reached and the watchdog is triggered and thus, the EPS is rebooted.
- **Ground watchdog time left (wdt_gnd_time_left):** Ground watchdog time in seconds that is left until zero is reached and the watchdog is triggered and thus, the EPS is rebooted.
- **CSP watchdog pings left (wdt_csp_pings_left[2]):** CSP watchdog pings left. Not used in our case because we do not implement CSP communication protocol.
- **I2C watchdog reboot count (wdt_i2c_reboot_count):** Counts the number of times that the I2C watchdog has been executed.
- **Ground watchdog reboot count (wdt_gnd_reboot_count):** Counts the number of times that the ground watchdog has been executed.
- **Watchdog CSP reboot count (wdt_csp_reboot_count):** Counts the number of times that the I2C watchdog has been executed.
- **Boot count (boot_count):** Counts the global boot count of the EPS.
- **Temperature sensor values (temperatures[6]):** Vector of six temperature values in Celsius degrees (formatted in a uint16 variable). Each position of the vector is linked to a temperature sensor in a specific position inside the EPS component.
- **Battery mode (battery_mode):** Value that indicates in which regime is the EPS working. Can vary from 0 (initial), 1 (undervoltage), 2 (safemode), 3 (nominal) and 4 (full).
- **Mode of Power-Point Tracking tracker (ppt_tracker):** Mode of the power-point tracking, which can be 1 (Maximum Power-Point Tracking) and 2 (SW Fixed Power Point Tracking).
- **Configuration checksum (config_checksum):** The checksum obtained from the configuration uploaded in the EPS. Used for checking with new configurations checksums.

A.1.3 EPS Error structure

The EPS error structure is formed by 8 different errors that can be raised in specific points of the EPS algorithm. These errors are stated below:

1. **I2C error:** Raised when it is not possible to contact the EPS through I2C protocol.
2. **Read memory error:** Raised when it is not possible to read the memory region where the configuration file is stored.
3. **Configuration format error:** Raised when the configuration format of the parsed configuration file is different than a JSON object.

4. **Configuration incoherent error:** Raised when, once the new uploaded configuration is parsed, one or more of the values are not coherent with the rules imposed (for example, no voltage can be negative).
5. **Configuration upload error:** Raised when it is not possible to set the new configuration to EPS.
6. **Point of Load access error:** Raised when it is not possible to toggle the requested point of load.
7. **Watchdog reset error:** Raised when it is not possible to reset the ground watchdog reset of the EPS.
8. **Cold batteries error:** Raised when the temperature sensor of the batteries drops below the low hysteresis temperature value. This means that the heater hasn't turned on due to an anomaly.

Now, the functions implemented in the OBC in order to control the EPS are grouped by target actions:

A.1.4 Functions related to EPS configuration

These functions control the EPS configuration values, ensures that the version uploaded to the subsystem is coherent and sends a hardcoded version in case that an error is raised during the process.

- **eps_get_conf():** Sends a command to receive the configuration uploaded to the EPS.
- **is_eps_conf_different():** Receives as an input the configuration uploaded to the EPS and the one uploaded to the OBC and compares them.
- **eps_set_conf():** Sends a command to set the configuration from the OBC to EPS.
- **verify_config():** Parses the configuration uploaded to the OBC to ensure that there is no incoherent values. Compares the uploaded values to ones fixed by the team.
- **reload_default_config():** In case that an uploaded configuration is not valid, a default hardcoded configuration is sent.
- **eps_set_heater():** Sends a command to modify the EPS heater modes between manual or automatic.

A.1.5 Functions related to EPS housekeeping

These functions are related to manage the EPS housekeeping data, specifically the retrieve of this telemetry from the subsystem itself into the OBC and then the parsing and processing of it.

- **eps_get_hk():** Sends a command requesting the latest set of housekeeping values.
- **process_housekeeping():** Includes the previous function *eps_get_hk()* and, after receiving the housekeeping, analyzes it and modifies the current status of the satellite accordingly. For example, the battery mode indicates in which power state is the satellite. Also, the temperature is analyzed and if it drops under a threshold, the heaters are turned on.

A.1.6 Functions related to EPS self-management and watchdog timers

These functions actuate over the subsystem mechanism of self-management. As the EPS is one of the key subsystems of the spacecraft, it has an extra layer of safety. The internal EPS software is structured with two watchdog timers. Watchdog timers are reverse timers that when arrives to zero an action is taken, normally a reboot of the system. This is the case for the i2c watchdog, which is set at 120 seconds and reset back to this value every time the subsystem receives an i2c communication. Also, the ground watchdog, which is set to 48 hours (172800 seconds), can be reset by a command sent by one of these functions.

- **eps_reset_wdt():** Sends a command to reset the ground watchdog timer to its initial value of 72 hours. This command is sent at each ground contact in order to prevent EPS reboot if it's not necessary.
- **eps_hard_reset():** Sends a command that completely reboots the subsystem, without the need of reaching any of the watchdog timers.
- **eps_reset_counter():** Sends a command to reset to 0 the boot count in order to monitorize a specific behaviour from ground if needed.

A.1.7 Functions related to EPS control of other subsystems

These functions are the ones used by the OBC to order the EPS to control the points of load where the other subsystems are connected.

- **shutdown_subsystem():** Sends a notification to toggle to 0 the point of load of the subsystem channel given as an input to the function.
- **poweron_subsystem():** Sends a notification to toggle to 1 the point of load of the subsystem channel given as an input to the function.
- **reboot_subsystem():** Sends a notification to toggle to 0 and then 1 the point of load of the subsystem channel given as an input to the function.
- **eps_change_pol():** Receives the notification from the three previous functions and acts accordingly sending the command to the EPS subsystem through I2C.

A.2 TTC task

A.2.1 TTC configuration parameters

There are three configurable parameters of the TTC task:

- **Beacon peek period (beacon_peek_period):** Period of time between beacon transmission. In milliseconds, our default value is 30000 ms. This means that every 30 seconds a telemetry beacon is transmitted.
- **Transceiver frequency (transceiver_frequency):** This is the frequency at which the onboard transceiver operates. The used unit is Hertz (Hz). In our case, the frequency is inside the amateur radio bands with the specific value of 437.35e6 Hz.
- **TTC Reference error mask (ttc_error_reference_mask):** 8-bit bitmask that indicates which errors must be ignored and which not. Each position is linked to a specific error and if the mask value is 0, the error is bypassed and not sent to the manager task. In our case, it is set to decimal value of 255, which means that there are no bypassed errors.

A.2.2 TTC housekeeping parameters

There are up to 23 housekeeping parameters from the TTC task, listed below:

- **Boot count (boot_count):** Counts the number of boots of the subsystem.
- **Actual RSSI (actual_rssi):** Measurement of the power from the received radio signal, known as Received Signal Strength Indicator, in dBm.
- **Last RSSI (last_rssi):** Measurement of the previous signal RSSI, in dBm.
- **Last LQI (lsat_lqi):** Measurement of the signal LQI.
- **Transmitted power (transmitted_power):** Measurement of the transmitted power, in dBW.
- **Physical transmission packets (phy_tx_packets):** Counts the amount of physical layer transmitted packets.
- **Physical received packets (phy_rx_packets):** Counts the amount of physical layer received packets.
- **Link layer transmitted packets (ll_tx_packets):** Counts the amount of link layer transmitted packets.
- **Link layer received packets(ll_rx_packets):** Counts the amount of link layer received packets.
- **Physical transmission error packets (phy_tx_err):** Counts the amount of physical layer transmitted error packets.

- **Physical received error packets (phy_rx_err):** Counts the amount of physical layer received error packets.
- **External temperature (ext_temp:** Measurement of the external temperature from the COMMS&ADCS board.
- **Internal temperature (int_temp:** Measurement of the internal temperature from the COMMS&ADCS board.
- **Operating frequency (freq):** Frequency of operation of the communications subsystem. Should match the frequency inputted in the configuration file.
- **Application layer received packets (app_layer_received):** Counts the amount of application layer received packets.
- **Application layer transmitted packets (app_layer_sent):** Counts the amount of application layer transmitted packets.
- **Command received (cmd_received:)** Counts the amount of commands received. Useful for controlling if any command has been sent but not delivered.
- **Last command id (last_command_id:** ID of the last command sent from the ground station to the satellite.
- **Bad authenticated commands (bad_auth_commands):** Counts the amount of bad authenticated commands received. A bad authenticated command happens when the command sequence is not followed.
- **Configuration checksum (config_checksum):** The checksum obtained from the configuration uploaded in the TTC task. Used for checking with new configurations check-sums.

A.2.3 TTC error structure

In the TTC task, there are up to 7 error defined, stated below:

1. **UART error:** Raised when it is not possible to contact the communication board through UART protocol.
2. **Read memory error:** Raised when it is not possible to read the memory region where the configuration file is stored.
3. **Write memory error:** Raised when it is not possible to write the memory region where the configuration file is stored.
4. **Configuration format error:** Raised when the configuration format of the parsed configuration file is different than a JSON object.
5. **Configuration incoherent error:** Raised when, once the new uploaded configuration is parsed, one or more of the values are not coherent with the rules imposed (for example, transceiver frequency not in the 420 to 450 MHz range).

6. **Configuration upload error:** Raised when it is not possible to set the new configuration to communications board.
7. **Transmission packet error:** Raised when it is not possible to transmit a packet from the satellite due to a flow error.

Now, the functions implemented in the OBC in order to control the communications are grouped by target actions:

A.2.4 Functions related to TTC configuration

- **verify_config():** Parses the configuration uploaded to the OBC to ensure that there is no incoherent values. Compares the uploaded values to ones fixed by the team.
- **reload_default_config():** In case that an uploaded configuration is not valid, a default hardcoded configuration is sent.

A.2.5 Functions related to TTC packet receiving and transmitting and housekeeping

- **handle_packet_reception():** This function processes the received packet and obtains its content, the telecommand from ground.
- **handle_beacon_transmission():** This function prepares and sends the historic telemetry beacon if there are any historic telemetry to be sent.
- **transmit_app_packet():** This function prepares and sends the application layer packet. It is used by most of the others functions in the task.
- **evaluate_beacon_mode_it_ht():** This function evaluates the state of the spacecraft and prevents of transmitting any beacon if the spacecraft mode is boot or ejection standby, as required.
- **parse_comms_telemetry():** This function receives and copies the ttc telemetry to its structured queue in order to be processed.

A.2.6 Functions related to TTC self-management and watchdog timers

- **comms_wdt_get():** This function obtains the comms watchdog token value.
- **comms_wdt_set():** This function inputs a token value and sets it as the comms watchdog token value.
- **parse_comms_state():** This function inputs the state of the ttc task and raises an error to the manager if needed.
- **reset_cmd_file():** This function resets the command counter value to 0, in case that any command is lost and to align again both transmitted and received command counts.

- **set_ttc_state():** This functions modifies the state of the ttc task, used to enable or disable the transmission procedure under specific commands from ground.

A.3 ADCS task

A.3.1 ADCS configuration parameters

ADCS counts with up to 17 configuration parameters. All of them except the reference error mask serve as a calibration of the algorithms running inside the ADCS subsystem software in order to perform a correct attitude and determination control. These parameters are all listed below:

- **Bdot proportional gain (bdot_kp:)** Proportional gain of the bdot algorithm, used for detumbling purposes.
- **Nadir proportional gain (nadir_kp_w[3]):** Proportional gain of the nadir part of the Bdot algorithm.
- **Nadir derivative gain (nadir_kd_w[3]):** Derivative gain of the nadir part of the Bdot algorithm.
- **Nadir proportional gain angle (nadir_kp_angle[2]):** Proportional gain angle of the nadir part of the Bdot algorithm.
- **Nadir derivative gain angle (nadir_kd_angle[2]):** Derivative gain angle of the nadir part of the Bdot algorithm.
- **Gyroscope variation (gyro_var):** Gyroscope variation.
- **Magnetic variation (gyro_var):** Magnetic variation.
- **Sun sensor variation (gyro_var):** Sun sensor variation.
- **Magnetometer 1 offset (mag1_offset[9]):** Magnetometer 1 offset.
- **Magnetometer 1 matrix (mag1_matrix[9]):** Magnetometer 1 calibration matrix.
- **Magnetometer 2 offset (mag1_offset[9]):** Magnetometer 2 offset.
- **Magnetometer 2 matrix (mag1_matrix[9]):** Magnetometer 2 calibration matrix.
- **Gyroscope offset (mag1_offset[9]):** Gyroscope offset.
- **Gyroscope matrix (mag1_matrix[9]):** Gyroscope calibration matrix.
- **Sun sensor offset (mag1_offset[9]):** Sun sensor offset.
- **Sun sensor matrix (mag1_matrix[9]):** Sun sensor calibration matrix.
- **ADCS reference error mask (adcs_reference_error_mask):** 8-bit bitmask that indicates which errors must be ignored and which not. Each position is linked to a specific error and if the mask value is 0, the error is bypassed and not sent to the manager task. In our case, it is set to decimal value of 255, which means that there are no bypassed errors.

A.3.2 ADCS TLE file

Along with the configuration file, the ADCS subsystem requires another input which is the TLE. The TLE is provided by the US Army and is a two line element set that describes the orbit of the body around the Earth and its position for a given time. This TLE is feeded to an orbit propagation algorithm inside the ADCS board that obtains high precision coordinates of the spacecraft for the next few days. This allows the system to obtain, for example, the IGRF values for each Earth zone and calibrate the magnetometers accordingly.

The TLE is uploaded as a TXT file, not as a JSON file like the normal configuration and has its own specific commands for uploading it. A sample of a previous NanoSat Lab's mission TLE is stated below:

```
1 46292U 20061W   21124.16166338  .00000590  00000-0  39655-4 0  9992
2 46292  97.4889 198.4490 0003399 173.0253 187.1024 15.10457944 36642
```

A.3.3 ADCS housekeeping parameters

Now the ADCS housekeeping parameters are listed and explained:

- **ADCS mode (mode):** Indicates the mode of the ADCS board. There are three possibilities: Detumbling mode (1), Nominal mode (2) and Mission Test mode (3).
- **Boot count (boot_count):** Counts the amount of board boots.
- **IGRF ECI coordinates value (igrf_eci[3]):** Provides the IGRF values for a given Earth-centered inertial coordinates. These values are obtained from the IGRF model uploaded to the ADCS subsystem.
- **LLH coordinates (pos_llh[3]):** Longitude, latitude and height coordinates of the satellite following the orbit propagator.
- **Sun ECI coordinates (sun_pos_eci[3]):** Position of the Sun in Earth-centered inertial coordinates.
- **Magnetometer 1 calibrated (mag_cal1[3]):** Calibrated values from the magnetometer 1 for each axis.
- **Magnetometer 2 calibrated (mag_cal2[3]):** Calibrated values from the magnetometer 2 for each axis.
- **Gyroscope calibrated (gyr[3]):** Calibrated values from the gyroscope for each axis.
- **Sun vector calculation (sun_vec_cal[3]):** Vector that represents the sun position in the satellite reference. Obtained from the sun sensor values.
- **Quaternion estimated values (quat_est[4]):** Estimation of the quaternion values, in a four-position vector.
- **Control values (control[3]):** Control values from each axis. Ranges from 0 to 1 and represent the percentage of activation of the magnetorquers.

- **External temperature (ext_temp):** External temperature value from the ADCS board sensor.
- **Internal temperature (int_temp):** Internal temperature value from the ADCS board sensor.
- **Solar panel temperature sensors (pd_temp[5]):** Readings of temperature from each solar panels temperature sensors.
- **Configuration checksum (config_chksm):** The checksum obtained from the configuration uploaded in the ADCS. Used for checking with new configurations checksums.

A.3.4 ADCS error structure

In the ADCS task, there are up to 8 error defined, stated below:

1. **I2C error:** Raised when it is not possible to contact the ADCS board through I2C protocol.
2. **Read memory error:** Raised when it is not possible to read the memory region where the configuration file is stored.
3. **Configuration format error:** Raised when the configuration format of the parsed configuration file is different than a JSON object.
4. **Configuration incoherent error:** Raised when, once the new uploaded configuration is parsed, one or more of the values are not coherent with the rules imposed (for example, transceiver frequency not in the 420 to 450 MHz range).
5. **Configuration upload error:** Raised when it is not possible to set the new configuration to communications board.
6. **Transmission packet error:** Raised when it is not possible to transmit a packet from the satellite due to a flow error.
7. **Set configuration error:** Raised when it is not possible to set a new configuration to the ADCS board.
8. **TLE format error:** Raised when the TLE format of the parsed TLE file has more than 144 characters.
9. **TLE incoherent error:** Raised when, once the new uploaded TLE is parsed, one or more of the values are not coherent with the rules imposed (for example, negative ascend node or inclination).

Now, the functions implemented in the OBC in order to control the ADCS are grouped by target actions:

A.3.5 Functions related to ADCS configuration

- **verify_config():** Parses the configuration uploaded to the OBC to ensure that there is no incoherent values. Compares the uploaded values to ones fixed by the team.
- **reload_default_config():** In case that an uploaded configuration is not valid, a default hardcoded configuration is sent.
- **adcs_set_conf():** After having the configuration file values, this function sets to the ADCS board the provided configuration parameters.
- **adcs_set_tle():** This function sets the TLE to the ADCS board. In the board, the TLE is feed directly to the orbit propagator to obtain the required values.

A.3.6 Functions related to ADCS housekeeping

- **adcs_get_hk():** This functions retrieves the housekeeping from the subsystem, parses it into an ADCS housekeeping structure and sends it to the dedicated queue. Outline that this function encompass specific parameters get functions.

A.3.7 Functions related to ADCS mission test mode

- **init_process_test():** This is the specific init function of the mission test process. As a difference with the normal init function, this sets a higher tick frequency in order to increase the sampling resolution and obtaining more data in the same amount of time.
- **process_test():** This is the specific process function of the mission test process. It performs the same actions as the usual process but adds in the telemetry the EPS data about total and ADCS point of load current consumption as well as structuring and sending the test data packages directly to TTC task.

A.4 OBDH task

A.4.1 OBDH configuration parameters

OBDH task has up to three different configuration parameters, stated below:

- **Beacon refill period (beacon_refill_period):** This parameter defines the period of beacon refilling, in milliseconds.
- **HT generation period (ht_generation_period):** This parameter defines the period of historic telemetry generation, in milliseconds.
- **OBDH reference error mask (obdh_reference_error_mask):** 8-bit bitmask that indicates which errors must be ignored and which not. Each position is linked to a specific error and if the mask value is 0, the error is bypassed and not sent to the

manager task. In our case, it is set to decimal value of 255, which means that there are no bypassed errors.

A.4.2 OBDH housekeeping parameters

The OBDH housekeeping is inside the global OBC housekeeping, as this subsystem runs entirely on the OBC. Its parameters are stated below:

- **FreeRTOS stack usage (freertos_stack_usage[8]):** FreeRTOS stack usage of each subtask initiated by the manager task.
- **RAM usage (ram_usage):** Current RAM usage of the system.
- **Flash memory usage (flash_usage):** Current Flash memory usage of the system.
- **Configuration checksum (config_chksm):** The checksum obtained from the configuration uploaded in the ADCS. Used for checking with new configurations checksums.

A.4.3 OBDH error structure

In the OBDH task, there are up to 8 error defined, stated below:

1. **I2C error:** Raised when it is not possible to contact the ADCS board through I2C protocol.
2. **Read memory error:** Raised when it is not possible to read the memory region where the configuration file is stored.
3. **Write memory error:** Raised when it is not possible to write the memory region where the configuration file is stored.
4. **Configuration format error:** Raised when the configuration format of the parsed configuration file is different than a JSON object.
5. **Configuration incoherent error:** Raised when, once the new uploaded configuration is parsed, one or more of the values are not coherent with the rules imposed (for example, beacon refill period under 5 seconds).
6. **Configuration upload error:** Raised when it is not possible to set the new configuration to communications board.
7. **Folder tree error:** Raised when there is an incorrect folder tree provided.
8. **Mount SD error:** Raised when it is impossible to mount the SD card.