



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Tècnica Superior d'Enginyeria
de Telecomunicació de Barcelona



Design and Implementation of a Communications System for a PocketQube based on LoRa

Degree Thesis
submitted to the Faculty of the
Escola Tècnica d'Enginyeria de Telecomunicació de Barcelona
Universitat Politècnica de Catalunya

by

Daniel Herencia Ruiz

In partial fulfillment
of the requirements for the degree in
Telecommunications Technologies and Services Engineering

Advisor: Dr. Adriano Camps Carmona
Barcelona, June 21, 2022

Abstract

We have recently been able to witness how large companies have set themselves the goal of launching fleets of hundreds or even thousands of satellites to offer different services. This new generation of satellites is marking a clear trend to reduce their size and cost.

However, this also leads to increasingly demanding challenges and constraints that require innovative solutions in the design of small-size, high-capacity communications systems. The present work offers the complete process from the design, to the implementation and test of a communications system for PocketQube picosatellites. The proposed solution benefits from LoRa technology, and includes both the hardware part of the communications subsystem and software for its configuration and operation according to the defined protocol.

Finally, this project has been designed and integrated, and its compatibility and co-operation with the rest of the subsystems has been tested, for the ^PoCAT picosatellites developed at the UPC NanoSat Lab.

Resumen

Recientemente hemos podido presenciar como grandes compañías se han marcado el objetivo de lanzar flotas de cientos o incluso miles de satélites para ofrecer diferentes servicios. Esta nueva generación de satelites está marcando una clara tendencia a reducir el tamaño y coste de éstos.

Sin embargo, esto también conlleva retos y restricciones cada vez más exigentes que necesitan soluciones innovadoras en el diseño de sistemas de comunicaciones de tamaño reducido y alta capacidad. El presente trabajo ofrece el proceso completo desde el diseño, pasando por la implemetación y test, de un sistema de comunicaciones para picosatellites PocketQube. La solución propuesta se beneficia de la tecnología LoRa, y comprende tanto la parte de hardware del subsistema de comunicaciones, como software para su configuración y funcionamiento según el protocolo definido.

Finalmente, este proyecto ha sido diseñado e integrado, y se ha probado su compatibilidad y cooperación con el resto de subsistemas, para los picosatellites ^{Po}CAT desarrollados por el UPC NanoSat Lab.

Resum

Recentment, hem pogut presenciar com grans companyies s'han marcat l'objectiu de llançar flotes de centenars o fins i tot milers de satèl·lits per oferir diferents serveis. Aquesta nova generació de satèl·lits està marcant una clara tendència a reduir-ne la mida i el cost.

No obstant, això també comporta reptes i restriccions cada vegada més exigents que necessiten solucions innovadores en el disseny de sistemes de comunicacions de mida reduïda i alta capacitat. Aquest treball ofereix el procés complet des del disseny, passant per la implementació i test, d'un sistema de comunicacions per a picosatèl·lits PocketQube. La solució proposada es beneficia de la tecnologia LoRa, i comprèn tant la part de hardware del subsistema de comunicacions, com el software per a la seva configuració i funcionament segons el protocol definit.

Finalment, aquest projecte ha estat dissenyat i integrat, i s'ha provat la seva compatibilitat i cooperació amb la resta de subsistemes, pels picosatèl·lits ^{Po}CAT desenvolupats per l'UPC NanoSat Lab.

Acknowledgements

Through these lines I wish to express my most sincere gratitude to all those who in one way or another have been related and have collaborated in the development of this work.

First of all, I would like to express my sincere gratitude to my tutor of the Degree Thesis, Dr. Adriano Camps, for giving me the possibility to become a member of the Nanosat Lab, and for the orientation and guidance that he have offered me throughout this project.

I would also like to thank the help of the students that have worked in the other subsystems of the satellites, specially to Gerard Bou and Ștefan Podaru, who together with me have taken the reins of this project during the last semester and that without their support this project would not have been possible.

I would also like to remind the help of Robert Molina, who started with me this project in the PAE subject, and of Mireia Nogu e, whose help has been very useful these last months.

I am also deeply thankful for the support offered by Dr. Llu s Pradell and Dr. Juan Manuel Rius, who helped me find some mistakes in the original design.

My final thanks, and not least, to my family, in particular to my parents, Federico and Pilar, for their unconditional support, and to my brothers Natalia, David and Laura for their understanding and patience.

Many thanks also to all those I have not mentioned, and who have somehow helped me to finish this work.

Revision history and approval record

Revision	Date	Purpose
0	05/03/2022	Document creation
1	11/06/2022	First version finished
2	12/06/2022	First revision
3	17/06/2022	Document correction and updates
4	20/06/2022	Document final revision

DOCUMENT DISTRIBUTION LIST

Name	e-mail
Daniel Herencia Ruiz	daniel.herencia@estudiantat.upc.edu
Adriano Jose Camps Carmona	adriano.jose.camps@upc.edu

Written by:		Reviewed and approved by:	
Date	20/06/2022	Date	20/06/2022
Name	Daniel Herencia Ruiz	Name	Adriano Camps Carmona
Position	Project Author	Position	Project Supervisor

Contents

ABSTRACT	i
ACKNOWLEDGEMENTS	iv
REVISION HISTORY AND APPROVAL RECORD	v
ABBREVIATIONS	ix
LIST OF FIGURES	xiii
LIST OF TABLES	xiv
1 INTRODUCTION	1
1.1 History of Small Satellites	1
1.2 PocketQube	3
1.2.1 The PocketQube Standard	3
1.3 Objectives and Scope	4
1.4 Precedents	4
1.5 Planning	5
2 PoCAT Project	6
2.1 Subsystems	6
2.2 Mission	6
2.2.1 Launching	7
2.2.2 Communications	7
2.3 State of the Art of Communications in Satellites	7
2.4 Technology Selection	8
2.5 Communications Subsystem Requirements and Specifications	9
3 SYSTEM DESIGN	11
3.1 Antenna Design	11
3.1.1 Antenna selection	11
3.1.2 Monopole design	12
3.1.3 Antenna design in lateral PCB	13
3.1.4 Antenna tuning	16

3.1.5	Radiation pattern	16
3.1.6	Antenna deployment	17
3.2	Communication Module	18
3.2.1	Component selection and schematic	18
3.2.2	COMMS RF PCB design	20
3.2.3	OBC-COMMS PCB	21
4	PROTOCOL AND SOFTWARE	24
4.1	TT&C	24
4.1.1	Beacon	24
4.1.2	Telecommands	24
4.1.3	Telemetry	26
4.1.4	Configuration	26
4.1.5	Remote calibration	26
4.1.6	Data packet	27
4.2	SOFTWARE	27
4.2.1	Multithread	28
4.2.2	SX1262 Library	28
4.2.3	COMMS State Machine	28
4.2.4	Transmission and Reception	29
5	LINK AND DATA BUDGET	31
5.1	Link Budget	31
5.2	Data Budget	33
6	CONCLUSIONS AND FUTURE WORK	34
6.1	Conclusions	34
6.2	Future Developments	34
	BIBLIOGRAPHY	36
	APPENDICES	40
A	Theoretical Background	40
A.1	LoRa	40
A.2	RF lines and waveguides	42

B	Hardware schematics and details	43
B.1	OBC-COMMS schematic	43
B.2	JLPCB design rules	46
B.3	OBC-COMMS design criteria	46
B.4	List of Components	50
C	Budget and Planning	53
C.1	Budget	53
C.2	GANTT Diagram	56
D	Link Budget Details	57
D.1	Montsec Ground Segment	57
D.2	PocketQube	57
D.3	Channel Impairments	59
E	SX1262 Internal State Machine	61
F	SX1262 library	63
G	Protocol Tests and Code	65
G.1	Tests	66
G.1.1	STM32 configuration	66
G.1.2	Transmission and reception	67
G.2	Code	69
G.2.1	Comms.c	70
G.2.2	Comms.h	87
G.2.3	GroundStation.ino	90

Abbreviations

ADC	—	Analog-to-Digital Converter
ADCS	—	Attitude Determination and Control System
API	—	Application Programming Interface
BOM	—	Bill Of Materials
BPSK	—	Binary Phase-Shift Keying
CAD	—	Channel Activity Detection
COMMS	—	Communications (subsystem)
COTS	—	Commercial Off-The-Shelf
CPU	—	Central Processing Unit
CR	—	Coding Rate
CSS	—	Chirp Spread Spectrum
DIO	—	Digital Input/Output
DSSS	—	Direct Sequence Spread Spectrum
DTN	—	Delay Tolerant Network
EIRP	—	Equivalent, Isotropically Radiated Power
EO	—	Earth Observation
EOL	—	End Of Life
EOS	—	Earth Observing System
EPS	—	Electrical Power System
ESA	—	European Space Agency
ETSETB	—	Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona
FIFO	—	First Input, First Output
FS	—	Frequency Synthesis
FSK	—	Frequency-Shift Keying
FSPL	—	Free-Space Path Losses
GANTT	—	Generalized Activity Normalization Time Table
GEO	—	Geostationary Orbit
GND	—	Ground (reference voltage level in electrical circuits)
GPIO	—	General Purpose Input/Output
GS	—	Ground Station
ID	—	Identification
IEEE	—	Institute of Electrical and Electronics Engineers
IoT	—	Internet of Things
IRQ	—	Interrupt Request
ISM	—	Industrial, Scientific and Medical (frequency band)
LEO	—	Low Earth Orbit
LNA	—	Low Noise Amplifier
LO	—	Local Oscillator
LoRa	—	Long Range
LoRaWAN	—	Long Range Wide Area Network
LPWAN	—	Low Power Wide Area Network
LTE	—	Long-Term Evolution (frequency band)
MCU	—	Microcontroller Unit
MEO	—	Medium Earth Orbit

MISO	—	Master In Slave Out
MOSI	—	Master Out Slave In
MPN	—	Manufacturer Part Number
MSc	—	Masters of Sciences
NB-IoT	—	Narrowband IoT
NSS	—	active-low (Not) Slave-Select
OBC	—	On-Board Computer
OSI	—	Open Systems Interconnection
PA	—	Power Amplifier
PAE	—	Projecte Avançat d'Enginyeria (Advanced Engineering Project)
PCB	—	Printed Circuit Board
PEC	—	Perfect Electric Conductor
PIN	—	Personal Identification Number
PL	—	Payload
PLL	—	Phase-Locked Loop
QPSK	—	Quadrature Phase Shift Keying
RF	—	Radio Frequency
RFI	—	Radio Frequency Interference
RFO	—	Radio Frequency Output
RPE	—	Radiation Pattern Envelope
RTC	—	Real-Time Clock
RTOS	—	Real-Time Operating System
RX	—	Receive
SA	—	Spectrum Analyser
SCK	—	Serial Clock
SF	—	Spreading Factor
SNR	—	Signal-to-Noise Ratio
SPI	—	Serial Peripheral Interface
TEM	—	Transverse Electromagnetic
THT	—	Through-Hole Technology
TLE	—	Two Line Elements
TT&C	—	Tracking Telemetry And Command
TVAC	—	Thermal and Vacuum Chamber
TX	—	Transmit
UART	—	Universal Asynchronous Receiver/Transmitter
VGA	—	Video Graphics Array
VNA	—	Vector Network Analyzer
WRT	—	With Respect To

List of Figures

Figure 1.1	Nanosatellite launches by type and forecast	2
Figure 1.2	Dimensions of a 1P PocketQube	4
Figure 2.1	Prototype of the ³ CAT-8 deployer	7
Figure 3.1	Equivalent dipole of a monopole	12
Figure 3.2	PocketQube with the antenna coming out	12
Figure 3.3	Inner face of the lateral PCB	13
Figure 3.4	Inner layer of the lateral PCB	13
Figure 3.5	Outside face of the lateral PCB	13
Figure 3.6	Return losses of the antenna circuit in lateral PCB	14
Figure 3.7	Inner face of the final design lateral PCB	15
Figure 3.8	Photography of the lateral PCB (final design)	15
Figure 3.9	3D model of the lateral PCB with the resistor (final design)	15
Figure 3.10	Outside face of the final design of the lateral PCB	15
Figure 3.11	Measurement of the antenna return losses with the VNA	16
Figure 3.12	UPC anechoic chamber	16
Figure 3.13	Normalized radiation pattern	16
Figure 3.14	XY cut of the normalized RPE	17
Figure 3.15	YZ cut of the normalized RPE	17
Figure 3.16	ZX cut of the normalized RPE	17
Figure 3.17	Gain radiation pattern in dB	17
Figure 3.18	Schematic of the deployment system	17
Figure 3.19	TVAC of the UPC NanoSat Lab	18
Figure 3.20	PocketQube before the deployment	18
Figure 3.21	PocketQube after the deployment	18
Figure 3.22	COMMS subsystem schematic design	19
Figure 3.23	Oscillator and power circuits to feed the transceiver	19
Figure 3.24	Schematic of the connections between the transceiver and the connector for the antenna	20
Figure 3.25	Table of the 1.6 mm 4-layer PCB materials and thickness	21
Figure 3.26	Coplanar waveguide dimensions	21
Figure 3.27	OBC-COMMS top layer surface 2D model	22

Figure 3.28	OBC-COMMS bottom layer surface 2D model	22
Figure 3.29	OBC-COMMS top layer 3D model tilted view	22
Figure 3.30	OBC-COMMS bottom layer 3D model tilted view	22
Figure 3.31	OBC-COMMS top layer 3D model	23
Figure 3.32	OBC-COMMS bottom layer 3D model	23
Figure 3.33	S_{12} parameter measured with the VNA	23
Figure 3.34	Pulse RX, measured with the Spectrum Analyser	23
Figure 4.1	Telecommand packet structure	24
Figure 4.2	Telemetry packet structure	26
Figure 4.3	Configuration packet structure	26
Figure 4.4	Calibration packet structure	27
Figure 4.5	Data packet structure	27
Figure 4.6	COMMS state machine	29
Figure 4.7	Telecommand reception and execution protocol flowchart	30
Figure 5.1	Received power and SNR as a function of the elevation	32
Figure A.1	LoRa CSS	40
Figure A.2	LoRa symbols	41
Figure A.3	Types of RF lines and waveguides	42
Figure B.1	STM32 pin connections	43
Figure B.2	STM32 power supply connections	43
Figure B.3	STM32 reference clocks connections	43
Figure B.4	Power and deployment switches connections	44
Figure B.5	COMMS schematic connections	44
Figure B.6	SX1262 Reference Design Layout – Top Layer	46
Figure B.7	SX1262 Reference Design Layout – Layer 2	46
Figure B.8	SX1262 Reference Design Layout – Layer 3	47
Figure B.9	SX1262 Reference Design Layout – Bottom Layer	47
Figure B.10	SX1262 Reference Design RF transceiver circuit	47
Figure B.11	OBC-COMMS PCB RF part	47
Figure B.12	Probe operation on the OBC-COMMS PCB	48
Figure B.13	Semtech Reference Design crystal oscillator and transceiver	49
Figure B.14	OBC-COMMS PCB crystal oscillator	49

Figure B.15 OBC-COMMS PCB transceiver	49
Figure B.16 3D model of the OBC-COMMS PCB with the coaxial cable	50
Figure D.1 Montsec Ground Segment	57
Figure D.2 Montsec UHF antenna	57
Figure D.3 Radiation pattern cut in the maximum direction	58
Figure D.4 RF circuit seen through an electronic microscope	58
Figure D.5 Antenna matching measurement set up	58
Figure D.6 Measurement of RF losses	59
Figure D.7 Rain attenuation as a function of frequency	59
Figure D.8 Gaseous absorption attenuation as a function of frequency	59
Figure D.9 Frequency of ionospheric disturbances in Earth	60
Figure D.10 Duration and time between fadings below 4 dB for the S_4 cases	60
Figure E.1 Transceiver circuit modes	61
Figure G.1 Nucleo-L476RG board	65
Figure G.2 SX126xMB2xAS board	65
Figure G.3 Heltec HTCC-AB01 Dev-Board	65
Figure G.4 STM32 pin type declaration	66
Figure G.5 Protocol timeout timer functioning	67
Figure G.6 Heltec module serial monitor	68
Figure G.7 Flash memory of the STM32	69

List of Tables

Table 1.1	Classification of small satellites by weight	2
Table 2.1	Comparison between the main LPWAN technologies	8
Table 2.2	^{Po} CAT COMMS mission requirements	9
Table 2.3	^{Po} CAT COMMS system requirements	9
Table 2.4	^{Po} CAT COMMS specifications	10
Table 3.1	Comparison between Semtech LoRa transceivers	18
Table 3.2	Connection lines between the transceiver and the OBC	19
Table 3.3	Final grounded coplanar waveguide specifications	21
Table 3.4	OBC-COMMS layer distribution	21
Table 4.1	Telecommands	25
Table 4.2	Telemetry	26
Table 4.3	Calibration	27
Table 5.1	Link Budget parameters	31
Table 5.2	Minimum elevation angle and sensitivity	32
Table 5.3	LoRa parameters depending on the SF	33
Table 5.4	Average contact time and amount of data transmitted per day	33
Table B.1	JLCPCB's Capabilities summary	46

1. INTRODUCTION

*“Because it’s there... Everest is the highest mountain in the world, and no man has reached its summit. Its existence is a challenge. The answer is instinctive, a part, I suppose, of man’s desire to conquer the universe.”*¹

Since the beginning of the time, the humankind has always looked beyond. Where there was a challenge, someone saw an opportunity. In people’s hearts there has always been a desire and aspiration to go higher, further, to improve. First the Everest was reached, then the space, after that, the Moon... *“It is not the mountain we conquer, but ourselves”* also said George Mallory. It is not the Earth we conquer, but ourselves. It is not the Space we conquer, but ourselves. The further the humanity gets, the more it realizes the complexity and opportunities that the Universe offers. We are not conquering it, but following the inclination of our hearts to know it and treat it properly, to perfect it while improving our lives.

This aim is what has motivated generations of scientists and engineers to reach the space: to know better our planet in order to know us better, to go further to create things that reduce the distance between us and limitations. These are the reasons that have led us to launch hundreds of satellites during the last decade. These are the reasons that have led us to the accelerated growth in the technological field that we are experiencing. And these are the reasons that have motivated me in this work.

1.1 History of Small Satellites

When Isaac Newton discovered gravitation and wrote down the Law of Gravity, the possibility of artificial satellites orbiting the Earth was introduced in the scientific world. In 1867, Everett Edward Hale in his tale *The Brick Moon* speculated on the idea of using artificial satellites for navigation, remote sensing and communications [1]. And in October 1945, Arthur C. Clarke published an article explaining how to provide worldwide communications coverage placing three space stations in a geosynchronous orbit (GEO) [1].

However, it was not until 1957, with the launching of the Sputnik 1, the first artificial satellite, that the space era started and the world was convinced that the idea of artificial satellites was feasible from an engineering point of view. Sputnik was a metal sphere with a diameter of 58 cm and an approximate weight of 84 kg. From a LEO orbit and with four antennas, it transmitted at 20.005 and 40.002 MHz. However, because of the absence of solar panels, the battery ended after 3 weeks [2].

After that, several experiments were done in the subsequent years. Some of the most significant are the following: in 1958, SCORE satellite boardcasted a message from President Eisenhower and Explorer 1 detected the Van Allen radiation belts. Also in 1958, Vanguard 1 was the first one that used solar cells. The reflecting satellite ECHO was

¹Answer given by George Mallory, who took part in the second expedition to the Mount Everest, without supplemental oxygen, and reached a record altitude of 8,225 m, to the question: *“Why climb Everest?”*

launched in 1960, the powered relay satellites TELSTAR and RELAY in 1962, and in 1963, SYCOM was the first geostationary satellite. Finally, in 1965, INTELSAT I (also called Early Bird) was the first commercial geostationary satellite [3][4].

During the 1980s, microsatellites appeared, with a design focus on reducing costs and using the available technologies and commercial off-the-shelf (COTS) components. This contrasted with the earlier philosophy where each satellite had its own design, and in this decade started the standardization process [4].

Nowadays, there are several types and standards of small satellites. The most frequent classification is based on the weight, as it can be seen in table 1.1. In the year 1999, Bob Twiggs and Jordi Puig-Suari proposed the Cubesat standard, which made the access to the space more generalized. The first Cubesat was launched in June, 2003. In figure 1.1 the evolution on the number of launches of nanosatellites is presented.

Table 1.1: Classification of small satellites by weight [4].

Satellite	Weight
Femtosatellite	10 - 100 g
Picosatellite	0.1 - 1 kg
Nanosatellite	1 - 10 kg
Microsatellite	10 - 100 kg
Mini-satellites or Small/Medium satellites	100 - 1000 kg

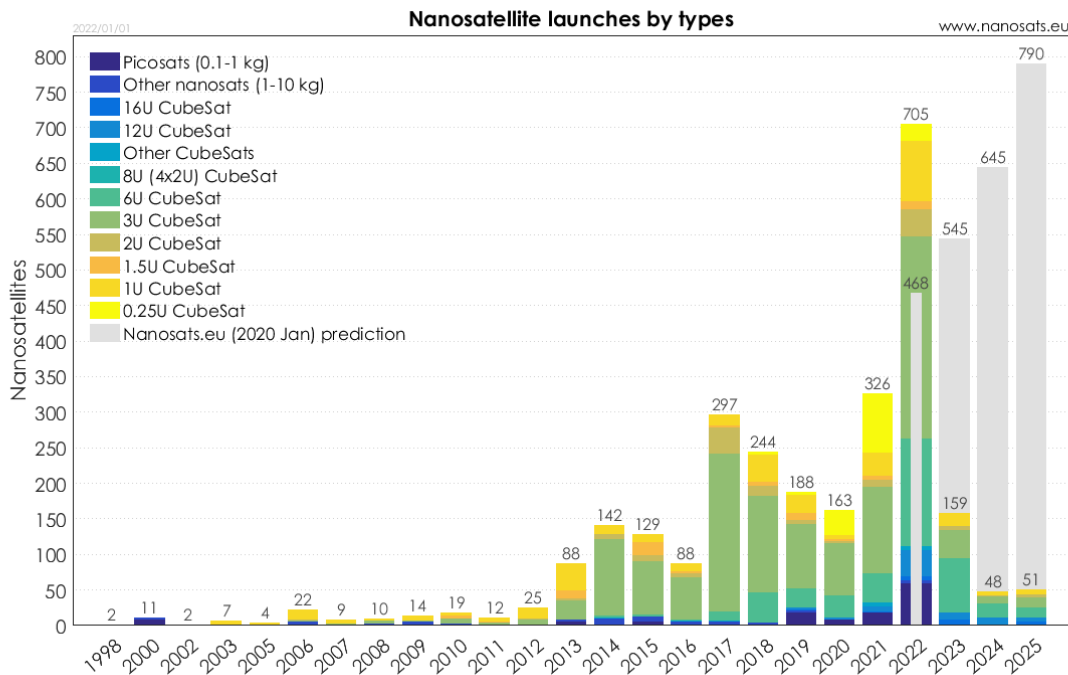


Figure 1.1: Nanosatellite launches by type and forecast. Image from [5]

These days, the applications of small satellites are divided in two main fields: the Earth Observation (EO), and the field of satellite communications. The reduced costs make possible the launching of massive constellations of nanosatellites, which highly reduces

the revisit times and makes possible the idea of providing worldwide Internet coverage with LEO satellites and IoT services [4].

1.2 PocketQube

Professor Robert J. Twiggs of Astronautics and Space Science at Morehead State University and professor Jordi Puig-Suari (and aerospace technology developer) at California Polytechnic State University proposed in 1999 the Cubesat concept, which was the first attempt to develop a nanosatellite standard [6]. A 1U Cubesat consists in a cube-shaped satellite of 10 x 10 x 11.35 cm side length, originally with a maximum weight of 1 kg [7].

Ten years later, Robert J. Twiggs was the first person that proposed the PocketQube concept, after a collaboration between the university where he works and Kentucky Space, in 2009. The name of PocketQube comes from, in words of professor Twiggs, “*a satellite that fits in your pocket*” [8]. The results of this was a series of specifications to help universities developing small space applications. However, it was not until November of 2013 that the first 4 PocketQubes were launched [9].

Therefore, PocketQubes are small size picosatellites that can be implemented with COTS components. These characteristics make them more affordable for small companies, universities and amateur engineers to design and launch them. For this reason, some companies that commercialize PocketQube, for example FOSSA Systems [10] or Alba Orbital [11], have the motivation of “Democratizing access to space”.

1.2.1 The PocketQube Standard

Although there is no complete standard for PocketQube, on June of 2018 a version of the PocketQube Standard was developed by three organizations: Delft University of Technology, Alba Orbital and Gauss Srl. This document contains some general requirements, and other related to the mechanical aspects, the mass and the materials. Future versions should contain requirements on electrical, operational and testing aspects, but nowadays they are not standardized, and may be adjusted to the ones specified by the Launch Service Provider.

The main requirements stated in the standard are the following [12]:

- The external dimensions of a one-unit PocketQube (1P) are 50x50x50 mm.
- The sliding backplate dimensions (1P) are 58x64x1.6 mm.
- The envelope around the PocketQube shall be no more than 7 mm for components, and a maximum of 10 mm for appendages and deployables, if the Launch Service Provider can adapt to this requirement.
- All PocketQubes shall use at least two kill switches to keep the satellite offline while in deployer.
- Each PocketQube unit (1P) shall not exceed 250 g mass.

In figure 1.2 the dimensions described before can be seen.

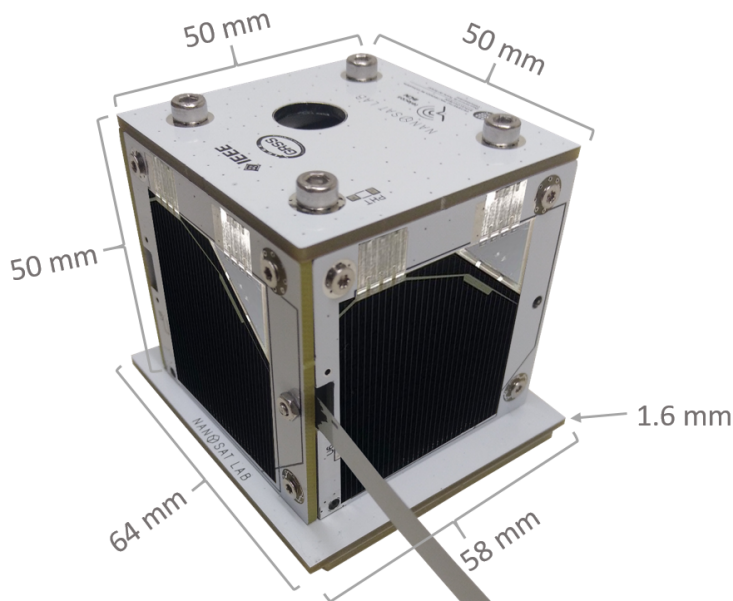


Figure 1.2: Dimensions of a 1P PocketQube²

1.3 Objectives and Scope

This project is an integral part of a more complex one, which is the creation of three educational 1P PocketQubes for the IEEE Geoscience and Remote Sensing Society. These 3 PocketQubes will also be launched into space on the UPC NanoSat Lab's ^{Po}CAT mission. As it is necessary to explain before some aspects of the whole mission and the state of the art, the requirements and specifications are presented in section 2.5 of Chapter 2. However, in broad terms, it can be said that the main objectives of this degree thesis is the design, assembly and testing of a complete communication system for a PocketQube, from the transceiver to the antenna, and with all the necessary connections to be controlled by a microprocessor.

With this objective, the scope of this project will cover the whole engineering process, beginning with the communications technology selection, which will lead to the hardware design besides the selection of its components. After that, all the circuits will be assembled and tested. Finally the communications protocol will be designed and programmed, with its corresponding tests. At the end, some improvements and conclusions will be presented.

1.4 Precedents

This work started in September 2021, when I enrolled the PAE ³CAT-NXT subject, which was also the continuation from another PAE from the previous semester. From

²In all the figures where the source is not specified, it is because they have been elaborated by the author of this work

the work done before I took care of the communications part, it only remains some of the requirements and the idea of using LoRa (which was proposed by PAE professor). My work during the four months that the PAE subject lasted, was basically an initial approach to the world of nanosatellites and space communications. Some initial designs of the antenna and COMMS PCB were done, but a more deep research during the period that I have been doing the final degree thesis, probed that the designs done in PAE were wrong. However, they served as a first iteration of the design and made me learn from the mistakes made. Finally, the basic points of the protocol and some coding were also done in this subject, but they were not tested.

In a summary, PAE subject was the beginning of this project and a good time to consolidate the initial ideas, and learn from beginner mistakes. In this work, the communications PCB design has been inspired in some Semtech evaluation boards, and the SX1262 library programmed by Semtech has also been used in the software part.

1.5 Planning

In the GANTT diagram in the appendix C.2, the organization of the tasks and workpackage, and its fulfillment, is presented. The first workpackage corresponds to the precedent work, done in PAE subject. The rest of the tasks have been classified in hardware and software workpackage. The only significant deviation from the initial workplan was that the arrival of some purchased components was delayed. Therefore, the initial idea of finishing all the hardware tests before starting the programming code was discarded and some software was advanced.

2. P^oCAT Project

The P^oCAT project consists of the design of an open-source PocketQube kit so that any organization or individual interested in this topic could use it as a model to start in the field. With this aim, three different types of PocketQubes are being created, sharing the same design in all the subsystems except from the payload, which will give a different functionality to every picosatellite.

2.1 Subsystems

The interaction of different subsystems is what makes possible the operations of a PocketQube, and of any satellite in general. These are basically 5 [13][14]:

- *OBC*: it is the brain of the satellite. The OBC is responsible for controlling and giving orders to the other subsystems, and handling its interactions. It is also the one that processes the data obtained by the payload and executes the orders received from the ground station.
- *COMMS*: it is the subsystem in charge of communicating the satellite with the Earth Station, transmitting the requested information, and receiving and processing the orders. It also transmits periodically a beacon with the satellite identifier and some state information.
- *ADCS*: it controls the PocketQube attitude in the space, and points it properly. To achieve that, it has to stabilize the rotation, accurately acquire the coordinates of each axis of the satellite, and point it in the desired direction, generating magnetic fields with active coils.
- *EPS*: it is responsible for managing the electrical power generated by the solar panels, storing it into the battery and controlling and distributing the regulated power supply throughout every subsystem of the satellite.
- *PAYLOAD*: it is the subsystem that defines the specific purpose of the satellite and develops its function, that can be a communication satellite, navigation, meteorological, Earth observation or other scientific sensors.

2.2 Mission

For the P^oCAT project, 3 different payloads have been created for the 3 different PocketQubes (P^oCAT-1, P^oCAT-2 and P^oCAT-3). The first one consists of a VGA camera, to take photos of a specific area to monitor deforestation. The second payload is a RFI monitoring receiver with an omnidirectional antenna from 1GHz to 2GHz. Lastly, the third payload aims to detect RFI in the 5G band of ~ 25 GHz, which is also near to the water vapor band of 24.8 GHz.

2.2.1 Launching

The 3 PocketQubes of ^{Po}CAT will be deployed in a LEO orbit with a deploying mechanism inside the ³CAT-8, which is a 6-units cubesat. In other words, they will be inside the cubesat during the launching process and ejected once the ³CAT-8 is in the corresponding orbit. In figure 2.1 the deployer is shown.



Figure 2.1: Prototype of the ³CAT-8 deployer, designed by MSc. Student Lluís Contreras [15]

2.2.2 Communications

After the deployment in orbit, the communications antenna will be deployed, and then the PocketQubes will start a stabilization process. After this, the satellite will always be in reception mode except when transmitting the beacon once every minute or when the Ground Station requests sending data.

Once the general aspects of the satellite have been explained, the following sections and chapters will focus on the Communications Subsystem.

2.3 State of the Art of Communications in Satellites

Nowadays, some important companies are starting to deploy Wide Area Networks with fleets of small satellites, mainly in LEO orbits. In recent times, these applications have been deployed mainly with Cubesats, but some companies are starting to migrate to PocketQubes.

However, as high gain antennas cannot be used in small satellites due to its dimensions, protocols resilient to interference, delays and attenuations are needed to establish

reliable links between the Earth Station and the satellite. Many communication protocols exist, but most of them are designed for a certain applications. The Low Power Wide Area Networks (LPWAN) technology allows to establish connections between two devices separated by hundreds of kilometers, converting it into an interesting protocol for satellite purposes. Some of the most used LPWAN technologies are SigFox, Narrowband IoT (NB-IoT) or Long Range (LoRa). The main characteristics of these protocols are the following:

- *SigFox*: it is a ultra-narrowband technology. It uses Binary Phase-Shift Keying (BPSK) modulations, taking a very narrow part of spectrum and changing the phase of the carrier radio wave to encode the data. Thanks to this, the receiver only has to listen a small spectral band, considerably reducing the noise. The network is constituted with simple endpoint terminals and sophisticated base stations. Therefore, the main constraint is the sensitivity of the endpoint receivers [16].
- *NB-IoT*: it is a standards-based (set up by 3GPP release 13 [17]) LPWAN technology that enables a wide range of IoT devices. It stands out for the low power consumption of the devices, the system capacity, and the spectrum efficiency [18]. This technology provides high data rates, but it can be oversized (and expensive) for normal situations. They require complex coding and synchronous signaling protocol techniques [17].
- *LoRa*: this technology works on unlicensed spectrum supporting an asynchronous bidirectional link layer protocol called LoRaWAN. Its physical layer uses the Chirp Spread Spectrum (CSS) modulation, which is very resilient to jamming, interference and fading, at the expense of losing quality of service (mainly bit rate) in comparison with other protocols, such as NB-IoT, that operate over the licensed spectrum [17]. The specification that governs how the network is managed is relatively open and provides more flexibility to the users [16].

2.4 Technology Selection

In the case of the ^{Po}CAT PocketQubes, a long range communication link is needed. In order to choose which technology to use, it is necessary to understand the capabilities of all of the previously stated alternatives. In table 2.1, a comparison between technologies is done.

Table 2.1: Comparison between the main LPWAN technologies

	LoRa	NB-IoT	Sigfox
Bandwidth (kHz)	125, 250, 500	200	0.1
Modulation	CSS	QPSK	BPSK
Frequency Band (MHz)	ISM	LTE	ISM
Maximum Tx power (dBm)	22	23	22
Rx power sensitivity (dBm)	-125	-125	-126
Maximum data rate (kbps)	27	250	0.6
Delay Tolerant Network (DTN)	Yes	No	Yes

Despite the fact that NB-IoT has better features in terms of bit rate and transmitted power, it is not delay tolerant. This characteristic is critical in satellite communications because, due to the distance between the satellite and the Earth station, the connection has to be tolerant to delays. Also, some LoRa bandwidths are resilient to Doppler, while Doppler frequency compensation is notably more complex in NB-IoT and Sigfox. Therefore, as LoRa has also better bit rate than Sigfox, the COMMS subsystem is based on LoRa technology.

Two other features not mentioned in table 2.1 make also LoRa a good choice. First, its resilience to interference, which is an important aspect to consider because P^oCAT mission will have 3 PocketQubes in the same orbit, and, very likely, they will be close between them. Second, its flexibility, that allows you to configure some parameters. For more information on how Lora works, see appendix A.1.

2.5 Communications Subsystem Requirements and Specifications

From the requirements of the P^oCAT, the mission requirements related with COMMS are presented in table 2.2.

Table 2.2: P^oCAT COMMS mission requirements

ID	REQUIREMENT
M-COMMS-0010	Send the satellite's gathered information and the satellite state to the ground station
M-COMMS-0020	Receive orders from the ground station

And the system requirements are gathered in table 2.3.

Table 2.3: P^oCAT COMMS system requirements

ID	REQUIREMENT
S-COMMS-0030	Establish a long-range communication with a low transmitted power
S-COMMS-0040	The antenna must be folded inside the PocketQube and deployed once in orbit
S-COMMS-0050	The link budget must be able to accurately establish when the information exchange can start
S-COMMS-0060	The packet transmission frequency must be the maximum possible once the contact between the satellite and the ground station has been established
S-COMMS-0070	The communication protocol has to keep the possible number of transmission errors at minimum

Finally, derived from the previous requirements, the specifications summarized in table 2.4 are obtained.

Table 2.4: P^oCAT COMMS specifications

ID	SPECIFICATION	VALIDATION
M-COMMS-0010	Each satellite will transmit the information gathered by its payload, and the telemetry	The transceiver receives correctly the information from memory
M-COMMS-0020	The satellite shall be able to know the moment when sending is possible and process the orders received	Notifications are received by the OBC after receiving an order from the GS
S-COMMS-0030	For long range low power transmissions, LoRa is the one with better specifications	No packets lost during the transmission
	In Europe the frequency band used is 863-870 MHz	Transceiver is correctly configured
S-COMMS-0040	The antenna will be folded in such a way that it does not exceed the allowed limits and will not be deployed until it is in orbit	Measure the dimensions of the antenna folded
	The deployment system shall be controlled by the OBC	Deployment test in ambient conditions and TVAC
S-COMMS-0050	The communication will only be possible when the received power exceeds the transceiver sensitivity	Communications test
S-COMMS-0060	Only some passes can be used to transmit information, but our system must transmit information whenever possible	Communications test varying the elevation angle
S-COMMS-0070	The protocol shall include redundancy to detect and correct errors	LoRa redundancy configured and tested

3. SYSTEM DESIGN

All wireless communication systems are mainly composed of two elements: the antenna, which is the part that radiates the signal, and the communications module, which consists of the transceiver and the necessary circuits to connect it with the antenna. This chapter provides the designs of both elements.

3.1 Antenna Design

The antenna is the part of the communications system that radiates the signals towards the ground station and receives the signals coming from it. There are different kinds of antennas, but not all of them fit the needs and constraints of a PocketQube mission.

Derived from the specifications in section 2.5, the antenna needs to have the following features:

1. Tuned at 868 MHz ($\lambda = 0.346 \text{ m}$), to use Europe LoRa ISM frequency.
2. Foldable within the PocketQube standard maximum dimensions.
3. Wide beam width (large θ_{3dB}) to avoid having pointing mismatch with the GS.

3.1.1 Antenna selection

The main types of antennas can be sorted as in the following classification:

- *Wire antenna (monopole, dipole,...)*: these are antennas that use a conductor as radiating element with a negligible section as compared to the length. Wire antennas, despite not having a high gain, are largely used due to its wide radiation pattern and its flexibility to be folded and deployed, which is an important requirement in satellite missions.
- *Aperture antenna (slot, horn,...)*: directive antennas that concentrate the radiated field in one direction. However, for 868 MHz, the antenna size is too large, and considering the power budget limitations, it is not feasible to occupy one whole lateral face (or more) with the antenna.
- *Microstrip antenna*: these are printed antennas that can be easily designed in a PCB. However, as for aperture antennas, patch dimensions do not fit in the PocketQube (80 x 103 mm approximately).
- *Reflector antenna*: this kind of antennas achieve high gain and directivity thanks to the use of a parabolic reflector. Despite the good results that would provide to the sensitivity and link budget, it is not feasible to design a deployable reflector for the used frequencies, and it will also generate more demanding requirements in terms of ADCS (pointing to the ground station).

- *Arrays*: they consist of grouping several identical antennas, often to increase its directivity or coverage. However, regarding the size requirements and the frequency used, it would complicate the design considerably using an array.

Having analyzed the different alternatives of antennas, only wire antennas accomplish the size requirements while keeping an easy method to fold them. Between a dipole and a monopole, the last one is the antenna used in this subsystem, mainly for these 3 reasons:

1. As the connections between OBC-COMMS board and the lateral board are done with a coaxial cable, using a dipole will require a balun circuit to convert from unbalanced signal (coaxial) to balanced signal (dipole).
2. As the outside part of the lateral board is occupied by the solar cells, the antenna has to be connected in the inner part and then go outside through a hole. Using a dipole complicates this design, because two holes are needed, and if both elements of the dipole have to be kept near the coaxial end to avoid extra microstrip lines, a significant part of the antenna will be inside the PocketQube, generating possible interferences and modifying some features of the radiation pattern.
3. The monopole has only one element, which is easier to fold than the two elements of the dipole.

3.1.2 Monopole design

Despite all the advantages that provides the use of a monopole, its performance in picosatellites differ a bit from the theoretical results.

An ideal monopole has an infinite ground plane that, following the image theory for currents, will radiate as if the monopole had another symmetric element on the other side of the ground plane (it is normally considered as an equivalent dipole, see figure 3.1). However, in real life, the ground is neither a perfect conductor which cancels the radiation in the horizontal direction, nor it has an infinite ground plane, which generates some back lobes. These are the reasons why the radiation pattern is modified regarding the ideal one.

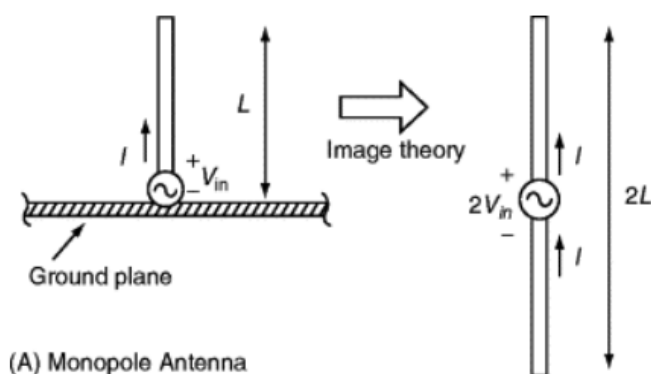


Figure 3.1: Equivalent dipole of a monopole. Image from [19]

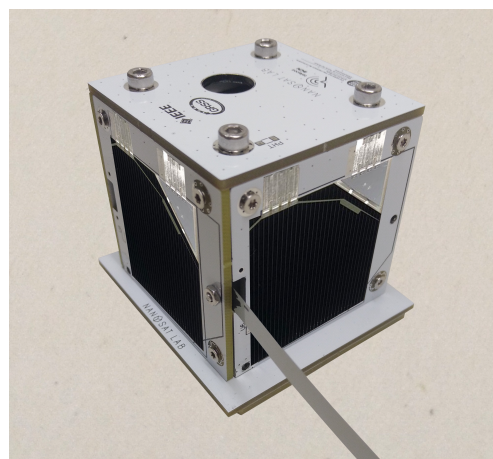


Figure 3.2: PocketQube with the antenna coming out

A monopole antenna with a small ground plane, or a ground plane with not too low impedance, may result in currents flowing in the common mode of the coaxial cable that feeds the antenna [20]. These currents on the shield of the coaxial cable can flow in any direction (and therefore not only in the same or opposite direction as the center conductor), making it a radiating element [21]. When talking about the image generated by the ground plane, that really means in phase currents, which are perpendicular to the ground plane. However, if the ground plan is not a Perfect Electric Conductor (PEC), the electromagnetic boundary conditions are not completely satisfied, and then, parallel currents are not totally compensated [22]. These two factors alter the properties of the monopole, which differs from the ideal equivalent dipole, especially in the radiation pattern.

With an "infinite" ground plane, currents decrease with distance from the feedpoint, making them negligible at far distances. Usually, it is intended to achieve a ground plane of at least a quarter of the wavelength in radius. Regarding the radiating element, every metallic object can act as an antenna. In this case, a metallic tape measure was used, because of its flexibility to be fold, the small thickness and the easiness to cut it to the desired length. The length of an ideal monopole is a quarter of the wavelength ($\frac{\lambda}{4} = \frac{c}{4f} = \frac{3 \cdot 10^8}{4 \cdot 868 \cdot 10^6} = 8.64 \text{ cm}$), but as is explained in section 3.1.4, the real length has to be adjusted after assembling it.

3.1.3 Antenna design in lateral PCB

As we can neither obtain an infinite ground plane nor one of a quarter wavelength radius (8.64 cm), we have to acquiesce to a ground plane "as big as possible". Therefore, as the perpendicular lateral PCB is being used as ground plane, the plane size is 5x5 cm. It actually is only half ground plane, because the antenna is not located in the center but in one side (this location was necessary in order to include a solar cell in the PCB, that for the antenna acts as a ground plane, see figure 3.2).

Apart from the antenna and the solar cell, the lateral PCB also includes in the inner part a temperature sensor, the circuit to gather the power from the solar cell and deliver it to the picoblade connector that connects with the EPS board, a photodiode, and 4 inner layers of coils that act as a magnetorquers for the ADCS subsystem. The connections between the communications PCB and the lateral PCB where the antenna is placed are done by means of a microcoaxial cable and U.FL connectors in both PCBs.

3.1.3.1 First design

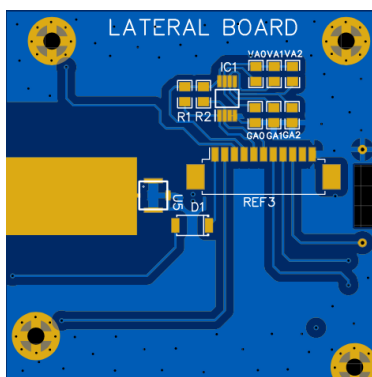


Figure 3.3: Inner face of the lateral PCB

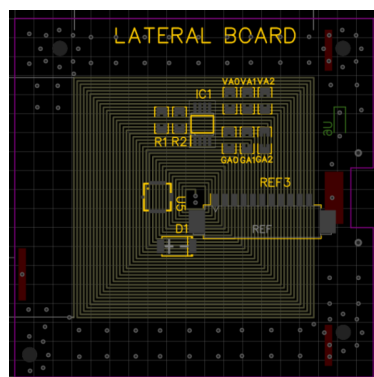


Figure 3.4: Inner layer of the lateral PCB

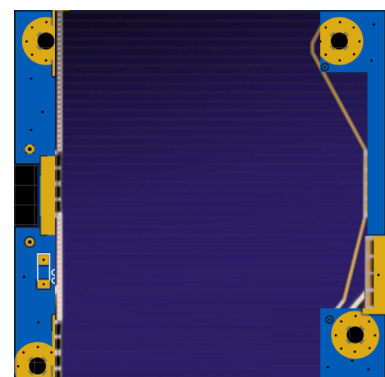


Figure 3.5: Outside face of the lateral PCB

The initial design that was done to test the antenna is presented in figures 3.3, 3.4 and 3.5. The gold area on the left of the figure 3.3 is where the antenna is welded and the component on the right (U5) is the U.FL connector. The other components are from the EPS and ADCS subsystems.

Once this design was manufactured, and the antenna and the connector were welded in the PCB, the return losses of the antenna were measured in order to determine if the antenna was radiating and at which frequency. Almost all the test showed the same results: one peak at 1.7 GHz and almost no radiation at the desired frequency, as can be seen in figure 3.6.

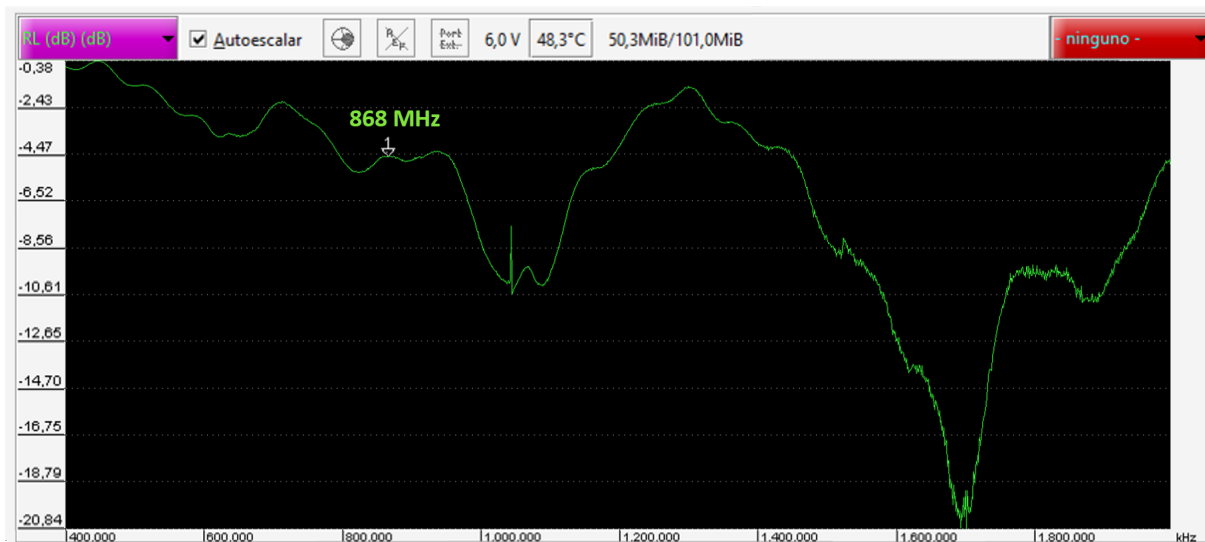


Figure 3.6: Return losses of the antenna circuit in lateral PCB

Another remarkable problem observed was that after varying the length of the antenna, the results did not change too much. This was a clear sign that the antenna was being short-circuited somewhere. After several tests with different configurations and a deep research in microwave circuits and antenna theory, the following conclusions were obtained:

1. Ground plane: as the thickness of the PCB is 1.6 mm, the inner and the outside faces of the PCB are too close in terms of wavelength. Therefore, the ground plane of the outside face that is under the antenna is short-circuiting it (it is like welding the antenna directly to GND). Also, to be on the safe side, the ground parallel to the antenna in the inner face has to be as far as possible. One last remark is that the GND has to be as uniform as possible and uniformly connected between PCBs.
2. Coils in inner layers: they will act as a ground plane. Therefore, as in the previous case, it can not be a coil under the area of the antenna. It will also prevent from generating fields in the antenna direction.
3. Solar cells: as the solar cells have a ground plane under them, the same considerations have to be taken into account as in the case of the coils and the GND.
4. Area to weld the antenna: reducing the area where the antenna is welded can assure that it will not act as another radiating element that could interfere with the antenna.

5. 1.7 GHz peak: the most reasonable explanation found to the 1.7 GHz peak is that, as the ground plane of the PCB is not ideal, then it can act to a lesser extent as a radiating element, as the other arm of a dipole. $\frac{\lambda}{4} = \frac{c}{4f} = \frac{3 \cdot 10^8}{4 \cdot 1.7 \cdot 10^9} = 4.4 \text{ cm}$, which is more or less the size of the PCB.

3.1.3.2 Final design

With the results obtained in the tests with the first model, the lateral PCB was redesigned and improved, correcting the errors mentioned in the previous case. In the new design, the area to weld the antenna was moved towards the end of the board (to avoid having the coils under) and it was also reduced. To add resistance to the antenna, a hole was done in order to screw it. Finally, the ground that was under the antenna was removed and the solar cell was cut (only in the PCB with the antenna, in the others the size of the solar cell was the same as before). The design of the coils and the components from EPS and ADCS was not modified. The final design can be seen in figures 3.7, 3.8, 3.9 and 3.10.

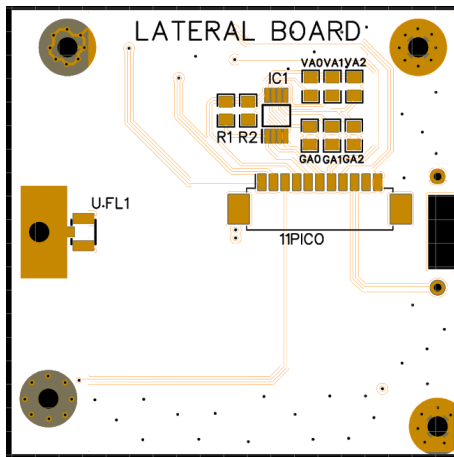


Figure 3.7: Inner face of the final design lateral PCB

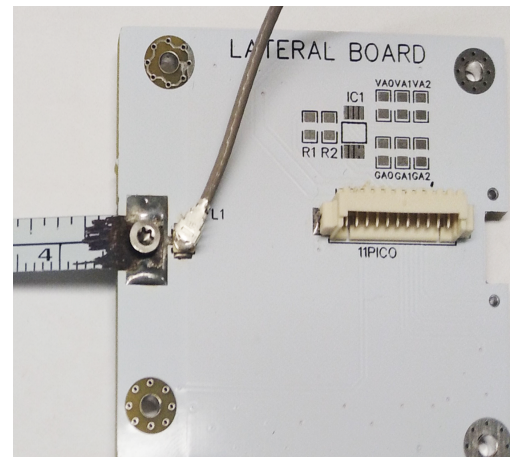


Figure 3.8: Photography of the lateral PCB (final design)

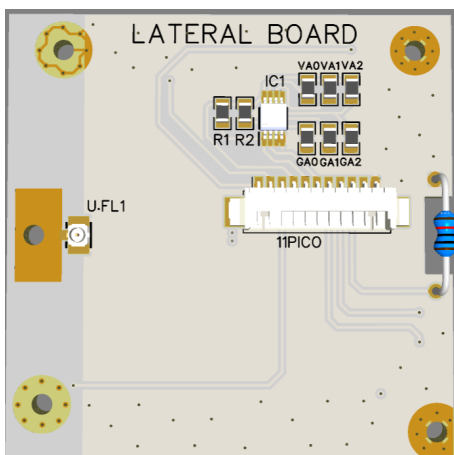


Figure 3.9: 3D model of the lateral PCB with the resistor (final design)

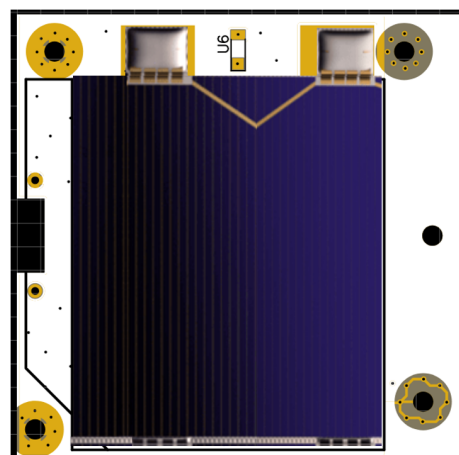


Figure 3.10: Outside face of the final design of the lateral PCB

3.1.4 Antenna tuning

Theoretically, a quarter wavelength monopole design to radiate at 868 MHz should have a length of 8.64 cm. However, in the practice, depending on the characteristics of the antenna and where it is placed, this length can differ a bit. In order to get an antenna resonant at the desired frequency, it was connected to a VNA and then cut until the peak was placed at 868 MHz. The obtained length was 9.5 cm, also counting the part of the antenna that is inside the PocketQube (the outside part is about 8.6 cm long, which is the one above the ground plane, and therefore, the one that counts.). As can be seen in figure 3.11, at 868 MHz, the value of the return losses (S_{11}) is -26 dB, which is a very good value (-20 dB means that the 99% of the power is radiated by the antenna).

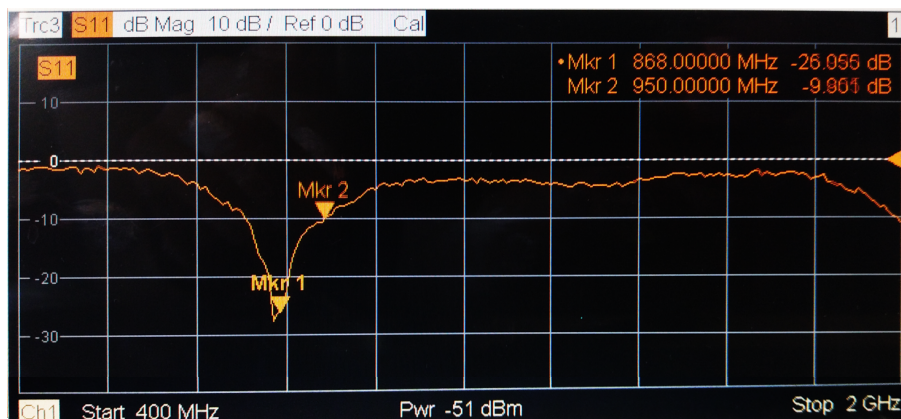


Figure 3.11: Measurement of the antenna return losses with the VNA

3.1.5 Radiation pattern

Another important test to determine the gain (or loss) of the antenna and the direction of radiation is the measurement of the radiation pattern. This was done in the anechoic chamber of the department of Signal Theory and Communications. Figure 3.12 is a picture of the anechoic chamber, and figures 3.13, 3.14, 3.15 and 3.16 show the results of the radiation pattern obtained.

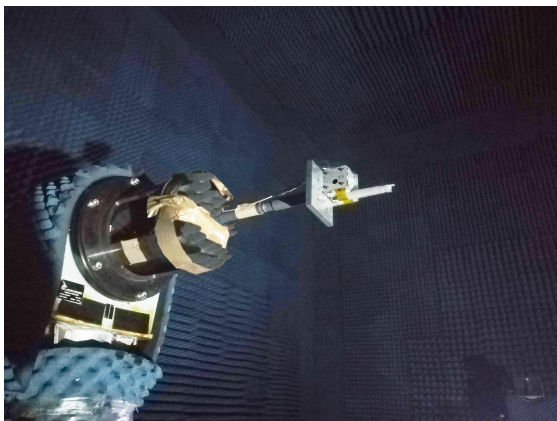


Figure 3.12: UPC anechoic chamber

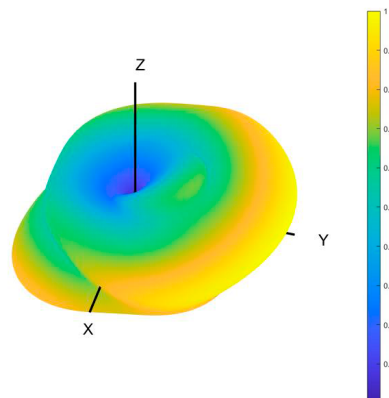


Figure 3.13: Normalized radiation pattern

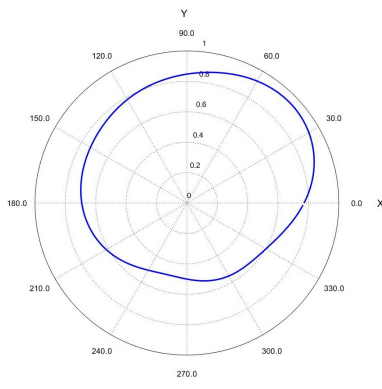


Figure 3.14: XY cut of the normalized RPE

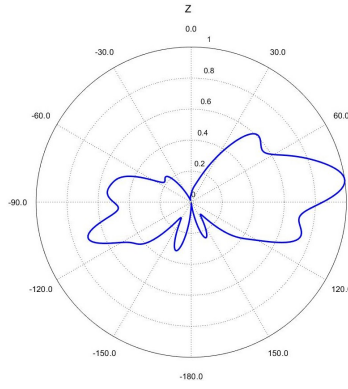


Figure 3.15: YZ cut of the normalized RPE

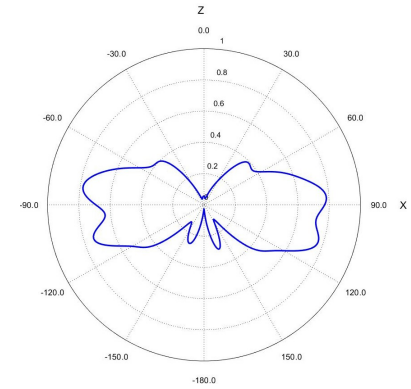


Figure 3.16: ZX cut of the normalized RPE

The maximum gain obtained is 1.15 dB, and the directivity is 3.115 dB. The θ_{-3dB} is approximately 60° . Finally, figure 3.17 presents the gain radiation pattern.

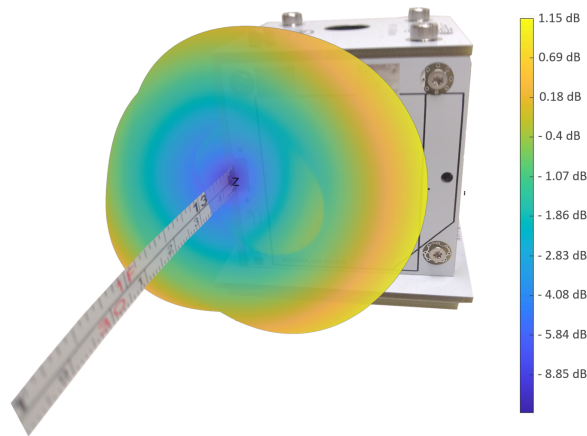


Figure 3.17: Gain radiation pattern in dB

3.1.6 Antenna deployment

As stated in the requirements, the antenna must be folded until the satellite is injected in its orbit. For this reason, a deployment mechanism has been designed. It consists of tying the antenna with dyneema wire to a THT resistor (on the lateral PCB, see figure 3.9), whose feeding is controlled by a switch in the OBC-COMMS PCB, and connected with a picoblade (see schematic in figure 3.18). Due to its low resistance, the resistor heats up quickly when it conducts current, and this burns the wire and deploys the antenna.

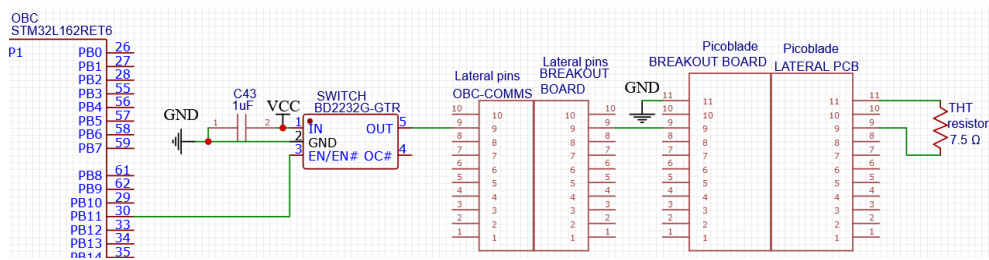


Figure 3.18: Schematic of the deployment system

Several values of resistors were tested, and the one which provided better results considering the power constraints ($I_{max} = 0.57A$, $V = 3.3V$) was the 7.5Ω resistor, that burnt the dyneema in 5 seconds. The final test was done in the TVAC (figure 3.19) at $3.1 \cdot 10^{-5} \text{ mbar}$ (almost orbit conditions), see figures 3.20 and 3.21.

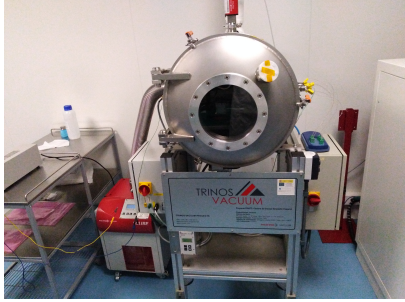


Figure 3.19: TVAC of the UPC NanoSat Lab

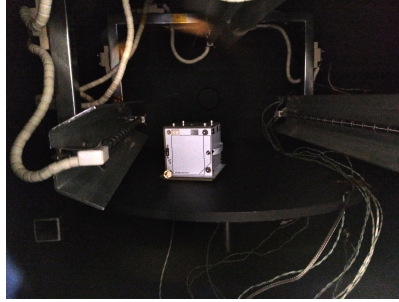


Figure 3.20: PocketQube before the deployment

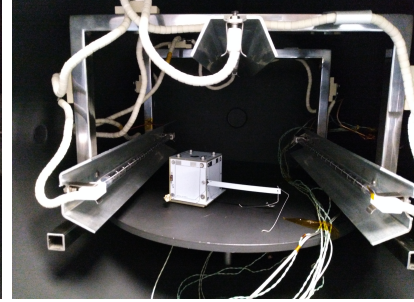


Figure 3.21: PocketQube after the deployment

3.2 Communication Module

The communication module is the part of the COMMS subsystem which implements the chosen technology (LoRa in this case) with the corresponding parameters. It also generates the signal to be transmitted, delivers it to the antenna and processes the signals received. Finally, it can also perform other functions such as amplifying the transmitted and received signals, using a power amplifier (PA) in the first case or a low noise amplifier (LNA) and a filter for the received signals.

3.2.1 Component selection and schematic

As LoRa technology is property of Semtech, they are the only providers that manufacture LoRa transceivers. Looking at the main features of the principal LoRa transceivers, summarized in table 3.1, the one that best fits our needs is the SX1262 [23], because it works at LoRa Europe frequency (868 MHz), it has the higher transmitted power, and it has good sizes and current consumption.

Table 3.1: Comparison between Semtech LoRa transceivers

Name	Freq. range	Out. power (P_{max})	Consumption (I_{max})	Size
SX1261	150-960 MHz	15 dBm	25.5 mA	4 x 4 mm
SX1262	150-960 MHz	22 dBm	118 mA	4 x 4 mm
SX1268	410-810 MHz	22 dBm	107 mA	4 x 4 mm
SX1272	860-1020 MHz	20 dBm	125 mA	6 x 6 mm
SX1276	137-1020 MHz	20 dBm	120 mA	6 x 6 mm
SX1278	137-525 MHz	20 dBm	120 mA	6 x 6 mm

Once the transceiver is selected, the circuit has to be designed according to the manufacturer specifications in the datasheet [23], and the instructions provided in the reference design [24].

Following the recommended design with small variations (just adding a couple of optional capacitors in the crystal oscillator and changing the RF switch, which was not in stock, for another available), the final schematic design is the one shown in figure 3.22.

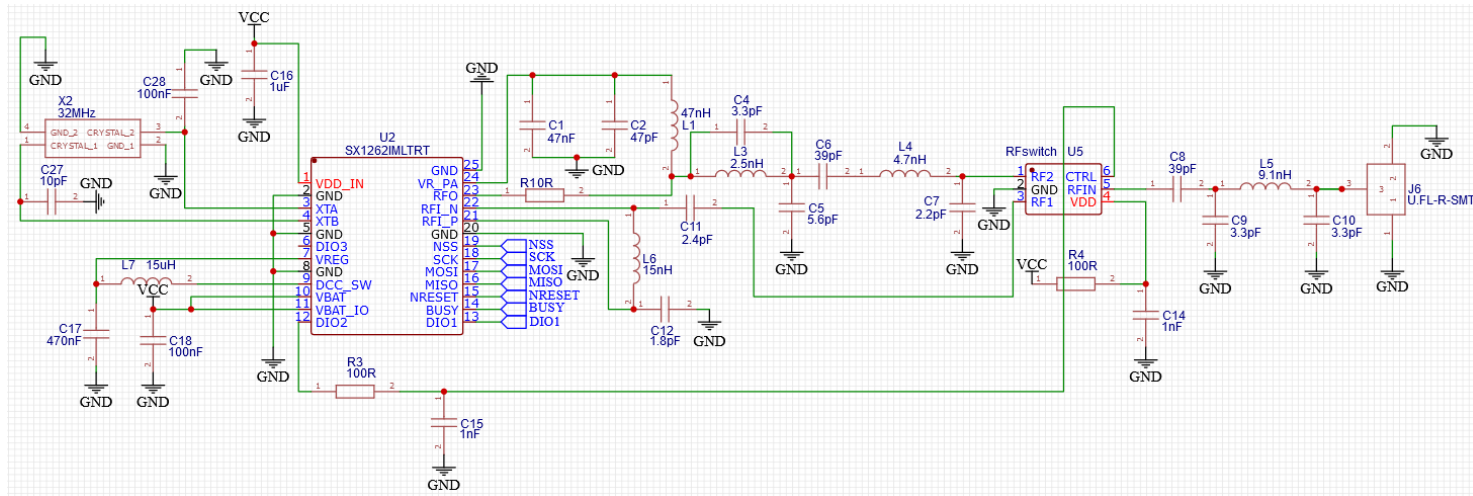


Figure 3.22: COMMS subsystem schematic design

In figure 3.23, the circuits needed for the transceiver to operate can be seen: the Crystal oscillator, which generates a signal with a constant frequency (32 MHz in this case), that is used to provide stable clock signals and to stabilize frequencies for the transmission and reception, and the power circuit that feeds the transceiver. The connections between the OBC (STM32) and the transceiver are done with SPI, and the DIO line is used to transmit interruption to the OBC. The 7 connection lines are summarized in table 3.2.

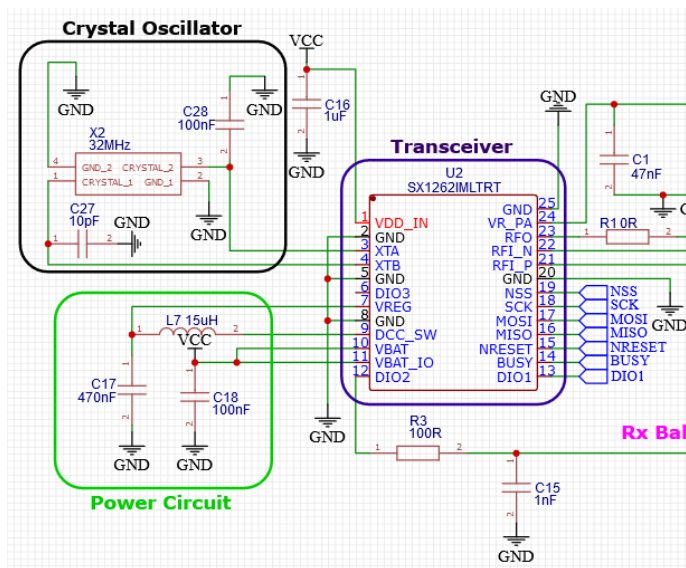


Figure 3.23: Oscillator and power circuits to feed the transceiver

Table 3.2: Connection lines between the transceiver and the OBC

Pin name	Description	Type
DIO1	IRQ line	Output
BUSY	Busy indicator	Output
NRESET	Active low reset signal	Input
MISO	SPI slave output	Output
MOSI	SPI slave input	Input
SCK	SPI clock	Input
NSS	SPI slave select	Input

Finally, figure 3.24 indicates the connections between the transceiver and the U.FL connector, which through a microcoaxial cable, goes directly to the lateral PCB and the

antenna. As the same antenna is used to transmit and receive signals, an RF switch is necessary to alternate both signals.

The upper part of the connection between the transceiver and the switch transmits the signal generated in the SX1262 (RFO line), then it is amplified (VR_PA line) and the harmonics of second and higher order are filtered. The bottom lines are the ones that take the signal received from the switch to the transceiver. As the transceiver needs a balanced signal and the switch provides an unbalanced one, a balun is added in between to do the unbalanced-balanced conversion. One last important aspect is that all the circuits that connect the main components (transceiver-switch-connector) are matched to $50\ \Omega$.

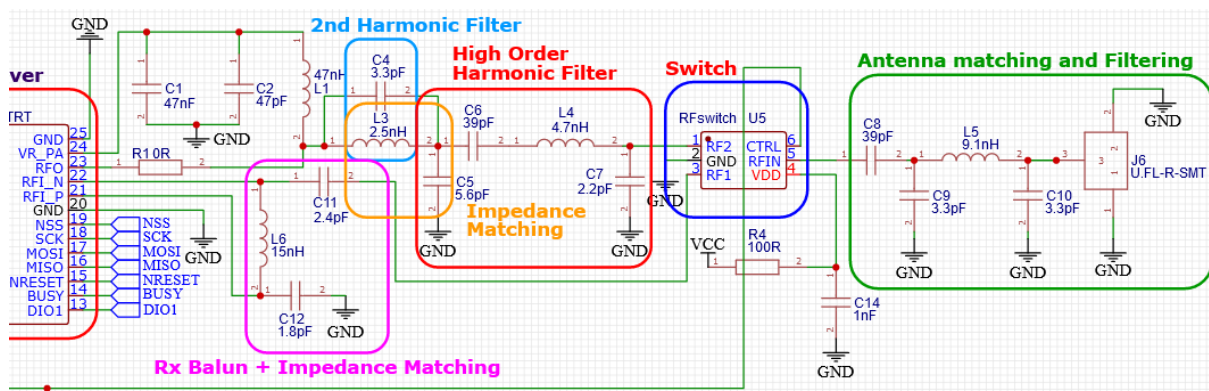


Figure 3.24: Schematic of the connections between the transceiver and the connector for the antenna

3.2.2 COMMS RF PCB design

Once the components have been selected and the circuit of the communications module designed, the next step is to translate it into a Printed Circuit Board (PCB). It has been designed with EasyEDA Designer software³ and manufactured by JLCPCB⁴.

It is also important to take into account that most of the lines shown in figure 3.24 are RF lines. Therefore, the design of this part of the circuit has to ensure the $50\ \Omega$ characteristic impedance in the printed lines to avoid power losses, and then, the line width, the dielectric, the separation between components,... matters.

Before starting the PCB, it is important to check the design rules that the manufacturer demands. In the case of JLCPCB, the most important ones are summarized in appendix B.2. After that, the type of line/waveguide to guide the RF signal have to be chosen. The main kinds are summarized in the appendix A.2.

Since the grounded coplanar waveguide arrangement provides better results, and this is also the kind of lines used in Semtech SX1262 evaluation board [25], the communications module RF circuit has been designed with them. As a ground plane is needed under the top layer to ensure the grounded effect, a minimum of 4 layers are required in the PCB. The specifications for a 4-layer PCB are shown in figure 3.25.

³<https://easyeda.com/editor>

⁴<https://jlcpcb.com/>

Layer	Material Type	Thickness	
Top Layer1	Copper	0.035 mm	
Prepreg	7628*1	0.2 mm	
Inner Layer2	Copper	0.0175 mm	1.1mm (with copper core)
Core	Core	1.065 mm	
Inner Layer3	Copper	0.0175 mm	
Prepreg	7628*1	0.2 mm	
Bottom Layer4	Copper	0.035 mm	

Figure 3.25: Table of the 1.6 mm 4-layer PCB materials and thickness. Image from [26]

Now, with all the specifications from the manufacturer, the characteristics of the pads have to be determined. As in the case of coplanar waveguides elliptical integrals have to be solved, the Software from Saturn PCB [27] has been used to compute the specifications of the pads. Table 3.3 and figure 3.26 show the final results obtained after some iterations.

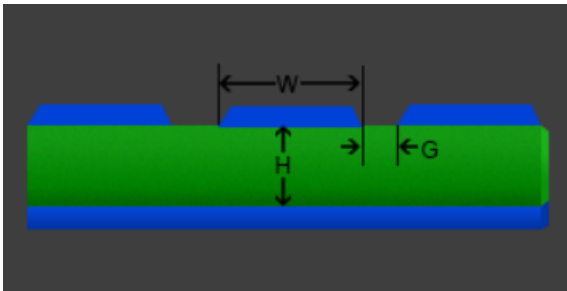


Figure 3.26: Coplanar waveguide dimensions

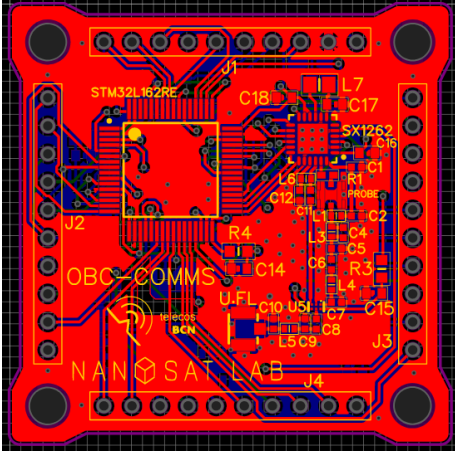
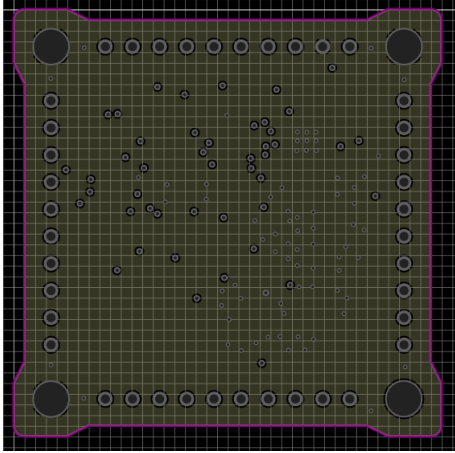
Table 3.3: Final grounded coplanar waveguide specifications

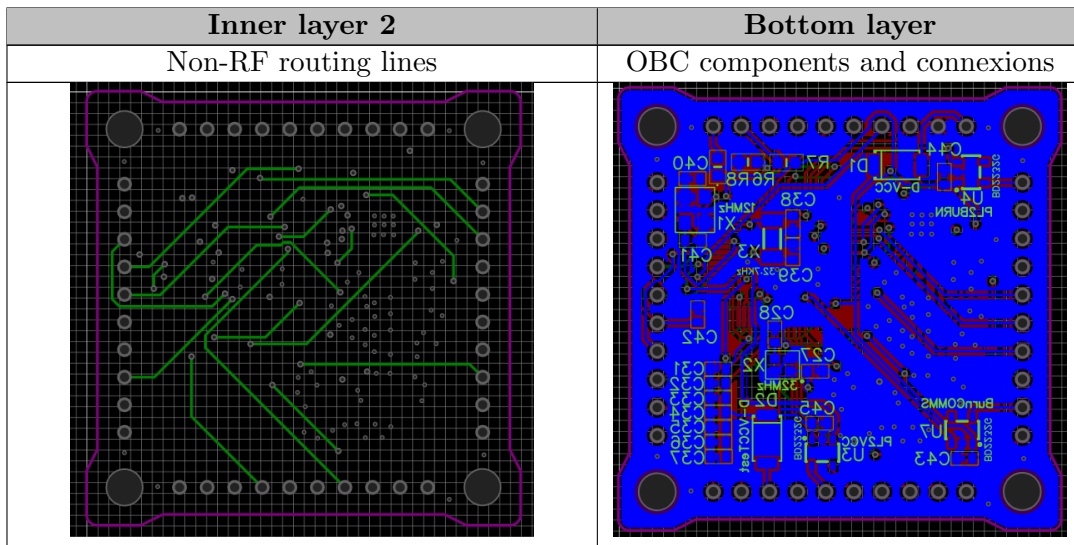
Magnitude	Value
Substrate ϵ_r	4.6 (FR4-STD)
Base Copper Weight	18 μm
Plating Thickness	35 μm
Conductor Width (W)	0.35 mm
Conductor Height (H)	0.2 mm
Conductor Gap (G)	0.2 mm
W/H	1.75
Z_o	49.84

3.2.3 OBC-COMMS PCB

As the space in PocketQubes is limited, the OBC and COMMS subsystem share the same PCB in the $\text{P}^{\text{o}}\text{Cat}$ mission. The schematic of the whole PCB is presented in the appendix B.1. In table 3.4 the different layers and the component distribution can be seen.

Table 3.4: OBC-COMMS layer distribution

Top layer	Inner layer 1
STM32, COMMS components and RF lines	Reference ground for all RF circuit
	



The only important difference with the schematic design is that a probe has been added at the output of the transceiver, in order to compare the power measurements at this point with those at the antenna connector, and therefore characterize better the behaviour of the circuit, the losses and the matching.

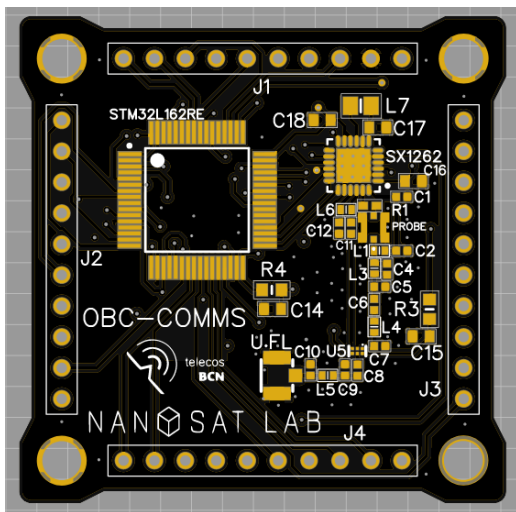


Figure 3.27: OBC-COMMS top layer surface 2D model

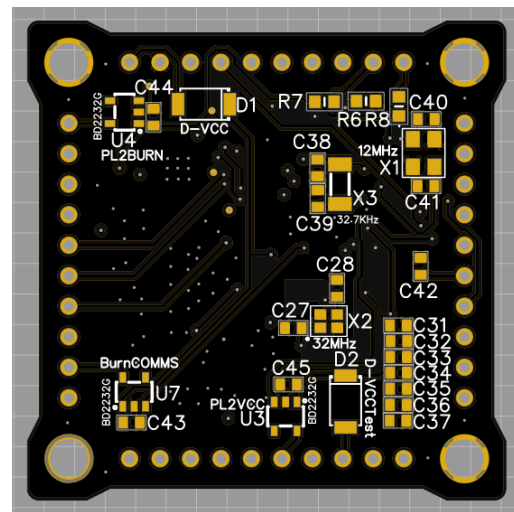


Figure 3.28: OBC-COMMS bottom layer surface 2D model

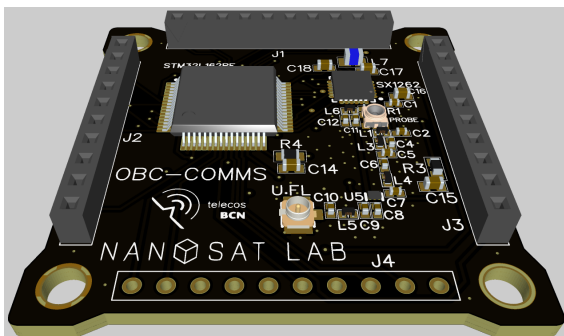


Figure 3.29: OBC-COMMS top layer 3D model tilted view

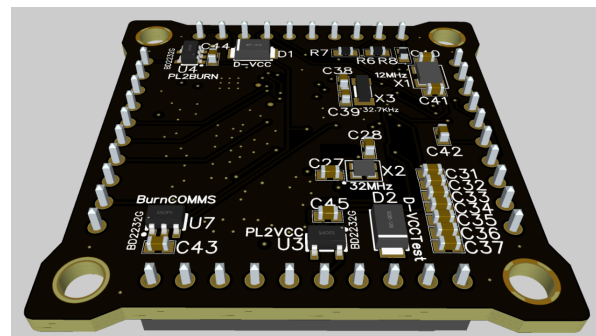


Figure 3.30: OBC-COMMS bottom layer 3D model tilted view

Finally, figures 3.27, 3.28, 3.29, 3.30, 3.31 and 3.32, show the 2D and 3D model design of the PCB. For more information on the PCB design, see the appendix B.3.

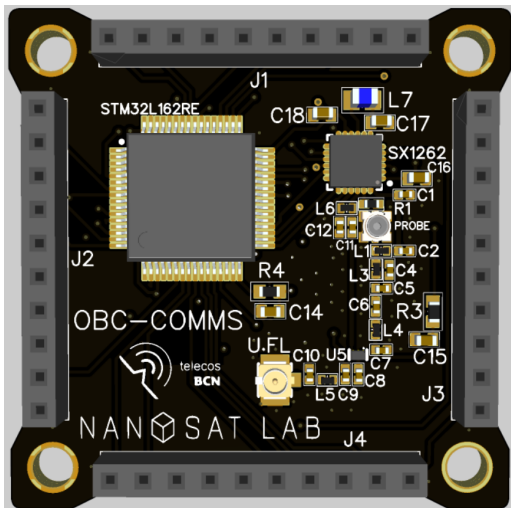


Figure 3.31: OBC-COMMS top layer 3D model

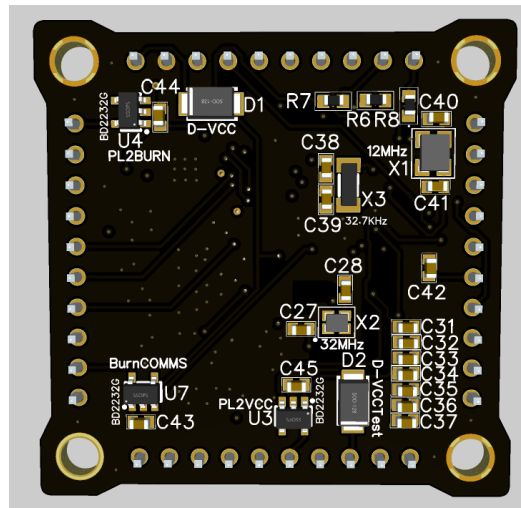


Figure 3.32: OBC-COMMS bottom layer 3D model

3.2.3.1 OBC-COMMS PCB tests

After assembling and welding all the components in the COMMS circuit, all the connections were electrically checked. Then, the losses in the Tx circuit (between the probe and the U.FL connector) were measured in two different ways. The first one consists of measuring the S_{12} parameter between these two points (see results in figure 3.33). The second test consisted of transmitting a pulse generated with the signal generator, and measuring with the spectrum analyzer the signal received at the other end (see results in figure 3.34). In this test, a pulse of -10 dBm was transmitted, and the cables to do the measurements introduced 2.5 dB of extra losses. With both tests we can conclude that the whole RF circuit introduces a bit less than 5 dB of losses.



Figure 3.33: S_{12} parameter with VNA

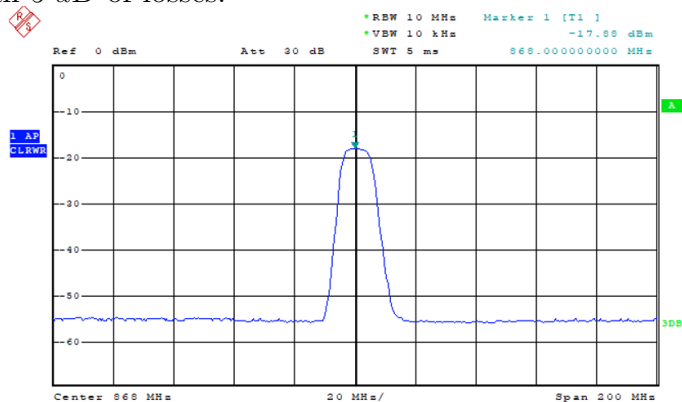


Figure 3.34: Pulse RX, measured with SA

These results are worse than expected. The causes can be mainly two. The first one is the fact of not using the RF switch proposed by the Semtech (because it is no longer manufactured). The second is that the circuit proposed by manufacturer is probably optimized for 915 MHz (LoRa frequency in North America) instead of 868 MHz. But to verify this, a more in-depth analysis would have to be done.

4. PROTOCOL AND SOFTWARE

The protocol is the sequence that rules the functioning of a communications system. A good protocol has to consider all the possible situations and define the actuation to follow in every case. The protocol designed for the P^oCAT PocketQubes forces the transceiver staying in a low power reception mode until a packet from the GS is received.

4.1 TT&C

This section explains the different packets transmitted by the satellite and the GS. The ID of the satellite and the mission has not been included in the packet size.

4.1.1 Beacon

When no packet is received (usually when there is no contact with the ground station), the transceiver transmits a beacon once every minute. A beacon is a packet that contains the most critical information, such as the satellite identification, the status of the battery... This data can be decoded by any ground station capable of receiving LoRa signals, and it also makes easier to track the satellite.

This has been programmed with a timer that throws an interruption every minute. Then, after checking that no transmission nor reception process has been initiated, it transmits two equal beacon packets one after the other (just in case one gets lost).

4.1.2 Telecommands

Telecommands are the orders and packets sent by the ground station to the satellite. The structure of every telecommand packet is presented in figure 4.1, and it contains, first, a pin used as a password only known by the ground station, to avoid the satellite executing commands sent by other sources (if the pin is not correct, the telecommand is rejected); then, it comes the ID of the telecommand; and finally, the remaining bytes contain, if needed, the information of the command.

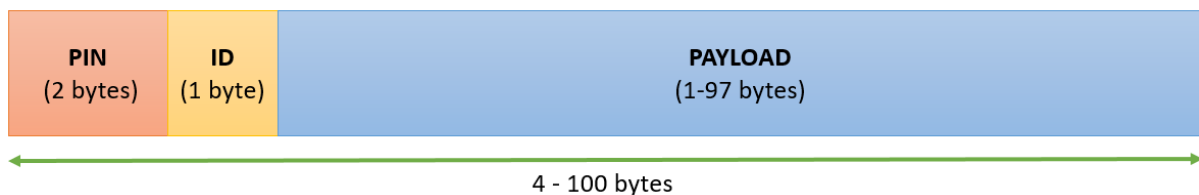


Figure 4.1: Telecommand packet structure

In table 4.1 are listed all the telecommands defined for the PocketQubes, along with their functionality and size.

Table 4.1: Telecommands

Telecommand ID	Function	Payload	Size
RESET2 (01)	Resets the whole PQ	-	-
NOMINAL (02)	Modifies the threshold used to change between each battery state	New threshold (percentage of the battery)	1 byte
LOW (03)			1 byte
CRITICAL (04)			1 byte
EXIT_LOW_POWER (05)	Leave the low power state	-	-
EXIT_CONTINGENCY (06)	Leave the contingency state	-	-
EXIT_SUNSAFE (07)	Leave the sunsafe state	-	-
SET_TIME (08)	Synchronises the time of the satellite and the GS	Unix time format	4 bytes
SET_CONSTANT_KP (10)	Updates the constant Kp	New value	1 byte
TLE (11)	Updates the TLE of the orbit	TLE size is 138 bytes, two packet are needed	69 bytes
SET_GYRO_RES (12)	Modifies the resolution of the gyroscopes	There are 4 different states for the resolution	1 byte
SEND_DATA (20)	Orders the satellite to transmit the payload data	Send data from payload. In payload 1, parameter for requesting big or small photos	1 byte
SEND_TELEMETRY (21)	Indicates the satellite to send the telemetry data acquired	-	-
STOP_SENDING_DATA (22)	Orders the satellite to stop sending data	-	-
ACK_DATA (23)	ACK packet to acknowledge all the packets received from last window	Array with the numbers of the packets to retransmit (max 20)	20 bytes
SET_SF_CR (24)	Modifies the SF and CR LoRa parameters	SF [7,12] 6 cases (1 byte) CR [4/5, 4/6, 4/7, 1/2] 4 different cases (1 byte)	2 bytes
SEND_CALIBRATION (25)	Sends all the parameters needed for calibration	Calibration parameters	96 bytes
CHANGE_TIMEOUT (26)	Modifies the timeout within transmitted packet	Timeout in milliseconds	2 bytes
TAKE_PHOTO (30)	Informs when to take the next photo, and sets the resolution and compression	Unix time format (4 bytes) 2 cases of resolution (1 byte) photo compression value (1 byte)	6 bytes
TAKE_RF (40)	Informs the satellite on when to start and finish taking RF measurements, f_{min} , f_{max} , resolution and the integration time	2 Unix time format (8 bytes) $f \in [1, 2] GHz$ (1 byte) 20 MHz precision (1 byte) integration time in 1 byte (For payload 3 values change)	12 bytes
SEND_CONFIG (50)	Orders the satellite to send its configuration	-	-

4.1.3 Telemetry

The telemetry packet contains the present status of the satellite and its sensors. Figure 4.2 presents its structure and table 4.2 summarizes the information contained.

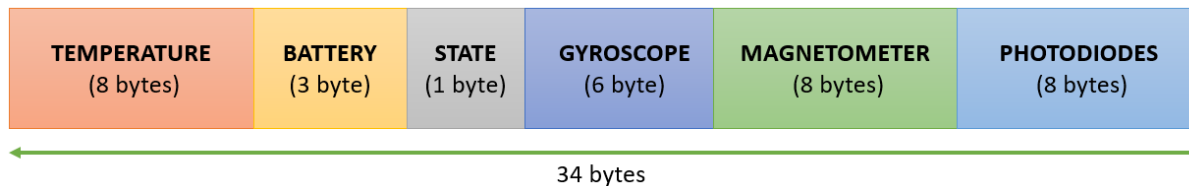


Figure 4.2: Telemetry packet structure

Table 4.2: Telemetry

	Information and sensor	Number of sensors	Total size
Temperature	Temperature sensors measurements of the 6 lateral PCBs, the battery and the microprocessor	8	8 bytes
Voltage	Battery voltage sensor	1	1 byte
Current	Battery current sensor	1	1 byte
Battery	Percentage of the battery capacity	1	1 byte
State	Current state (init, idle, contingency, sun safe or survival)	1	1 byte
Gyroscope	Last measurements of the gyroscopes	3	6 bytes
Magnetometer	Last measurements of the magnetometers	3	8 bytes
Photodiodes	Last measurements of the photodiodes	6	8 bytes

4.1.4 Configuration

This packet contains the current value of the all the configurable parameters (the ones that can be changed via telecommand). Figure 4.3 presents its different fields.

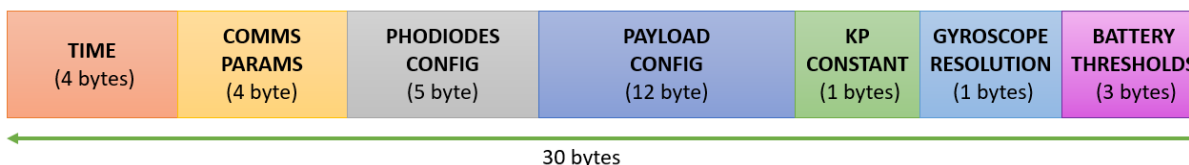


Figure 4.3: Configuration packet structure

4.1.5 Remote calibration

In order to calibrate the different sensors of the satellite, the ground station can send the remote calibration packet. The structure of the packets is the one explained in figure 4.4 and detailed in table 4.3.

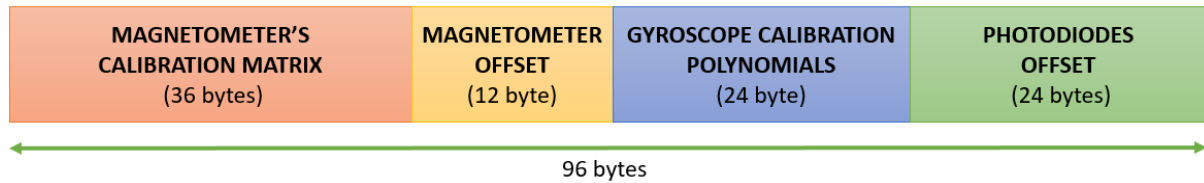


Figure 4.4: Calibration packet structure

Table 4.3: Calibration

	Information (stored in floats)	Size
Calibration matrix (3x3)	Rotates the measured values to match the axis. Crosstalk is used to compute axis overlapping	36 bytes
Magnetometer offset	Offset of the 3 magnetometer measurements	12 bytes
Gyroscope polynomials	Coefficients used to calibrate the offsets of the 6 gyroscopes	24 bytes
Photodiodes offset	Photodiodes measurements offset (x6)	24 bytes

4.1.6 Data packet

When the telecommand *SEND_DATA* is received, the satellite starts sending data packets with all the information gathered by the payload until the twentieth packet is sent. Then, it waits for the acknowledgment of the last 20 packets, which is sent by the ground station with a packet of maximum 20 bytes (*ACK_DATA* telecommand). This packet contains the numbers of the packets that need a retransmission, and after retransmitting them, it continues sending new data packets until the 20th packet of this transmission window is sent (included the retransmitted ones). The structure of a data packet, in this case containing the identification headers and the end flag, is shown in figure 4.5.

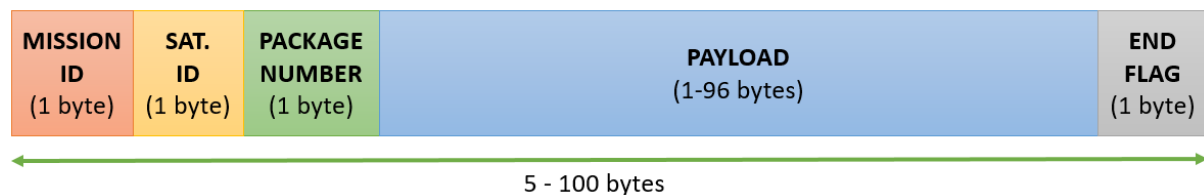


Figure 4.5: Data packet structure

4.2 SOFTWARE

The largest part of the communications subsystem functions are carried out by the SX1262 transceiver. However, this chip does not have a microcontroller integrated, and cannot be programmed directly, only configured and controlled by means of commands. Therefore, all the COMMS programming code must be integrated in the OBC, which is an STM32 microcontroller. All the code has been programmed in C language and compiled with STM32CubeIDE.

4.2.1 Multithread

As the code from OBC, ADCS, the PAYLOAD and COMMS has to be integrated in the same chip, a multithread system based on free RTOS has been implemented, which facilitates multitasking and integration in designs with limited resources and time. Then, at the beginning of the mission, when the initialization tasks have finished and the antenna has been deployed, the OBC initializes the COMMS thread. This thread works independently from the others and communicates with the OBC by means of notifications. If any error occurs or the satellites enters in a critical state due to the battery level, the COMMS thread is killed and reinitialized when the OBC feels it appropriate.

4.2.2 SX1262 Library

The best way to manage the SX1262 transceiver is through the functions provided by Semtech in the transceiver library. The internal functioning of the transceiver and an explanation of the library can be seen in the appendices E and F respectively.

4.2.3 COMMS State Machine

In order to organize the COMMS code structure, a State Machine has been designed and programmed with the following states (see also flowchart in figure 4.6):

- *TX*: this state prepares the corresponding packet (data, error, telemetry, configuration, beacon, confirmation...) depending on the activated flags, transmits the packet and goes to *LOWPOWER*.
- *TX_TIMEOUT*: if a transmission is not completed for any error, the tx timeout IRQ will lead the code to this state, to prevent the transceiver getting stuck in an erroneous transmission.
- *RX*: the reception state checks all the conditions of the protocol followed when a telecommand is received, explained in section 4.2.4, activates the corresponding flags and goes to *LOWPOWER*.
- *RX_ERROR*: when an error in the reception process occurs, an interruption leads to this state, to restart the reception process.
- *RX_TIMEOUT*: as in the case of the transmitting process, when trying to receive a packet, a timer is activated to avoid getting stuck in a bug or other receiving errors. It is important to notice the difference between this timeout, which starts when channel activity has been detected and the transceiver begins the reception, and the protocol timeout, which refers to the time that the transceiver is listening to detect and start receiving new packets.
- *LOWPOWER*: to minimize the power consumption, if no transmission is needed, the transceiver goes to this state and switches between it and *RX*.

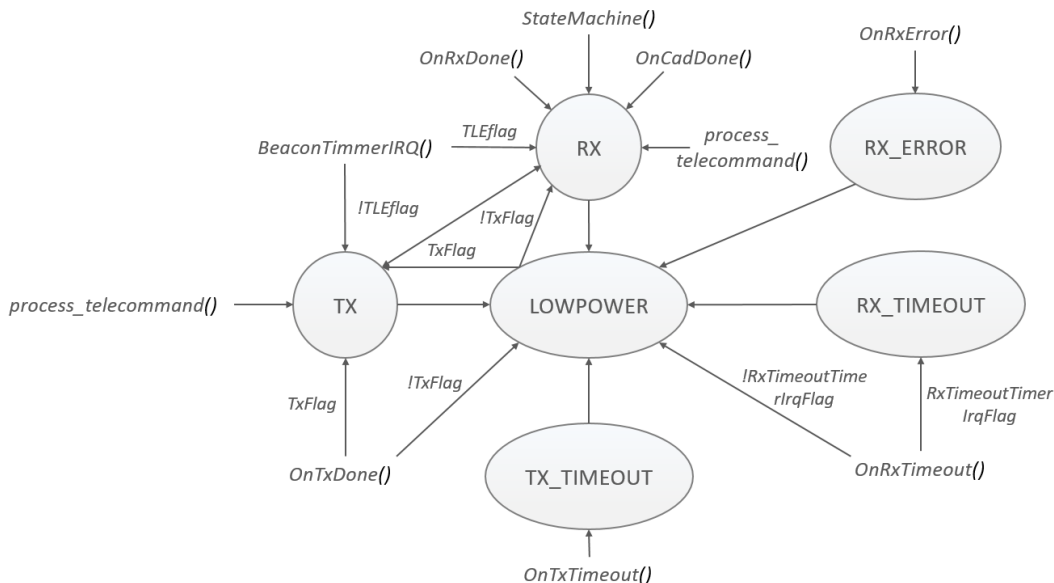


Figure 4.6: COMMS state machine

4.2.4 Transmission and Reception

The satellite does not know when it has visibility with the ground station until it receives a telecommand. Then, it starts the following transmission and reception protocol:

1. After receiving one packet, the satellite checks if it has its mission and satellite ID, and then checks if the PIN and the telecommand ID are correct. Otherwise, the packet is discarded, and an error packet is sent.
2. As the TLE telecommand is the only one that needs two packets to be transmitted, if one is received, the satellite waits in RX state for the second one, and then processes them. If another telecommand is received instead of the second packet, then it ignores the previous TLE telecommand.
3. As all the packets are transmitted in pairs (except from the data and TLE packets), which means that the same packet is transmitted twice with a small separation time (100 ms) in case one gets lost, the satellite waits a safety time (500 ms for SF 7) to avoid collisions after one telecommand is received.
4. In the case that both are received or only one is received, then there are two options. Otherwise, if two different telecommands are received during this safety time, both are discarded, and an error message is sent.
 - (a) If the telecommand is a sending order or the acknowledgment from the previous packets sent, the transceiver goes to TX and sends the required information.
 - (b) Otherwise, it retransmits the telecommand received to the ground station, to confirm if the order has been received correctly.
5. If the GS confirms the order, it is then executed. If the timeout is triggered 3 times, which means that the satellite has asked for confirmation 3 consecutive times without answer, the telecommand is discarded and an error message is sent.

The flowchart of this protocol is schematized in figure 4.7. All the communications code programmed is accessible in the github of the author⁵, and some files are included in the appendix G.2. Also, all the code has been tested with a STM32L476 nucleo board and a SX1262 transceiver, and a Heltec LoRa module acting as GS (see appendix G.1).

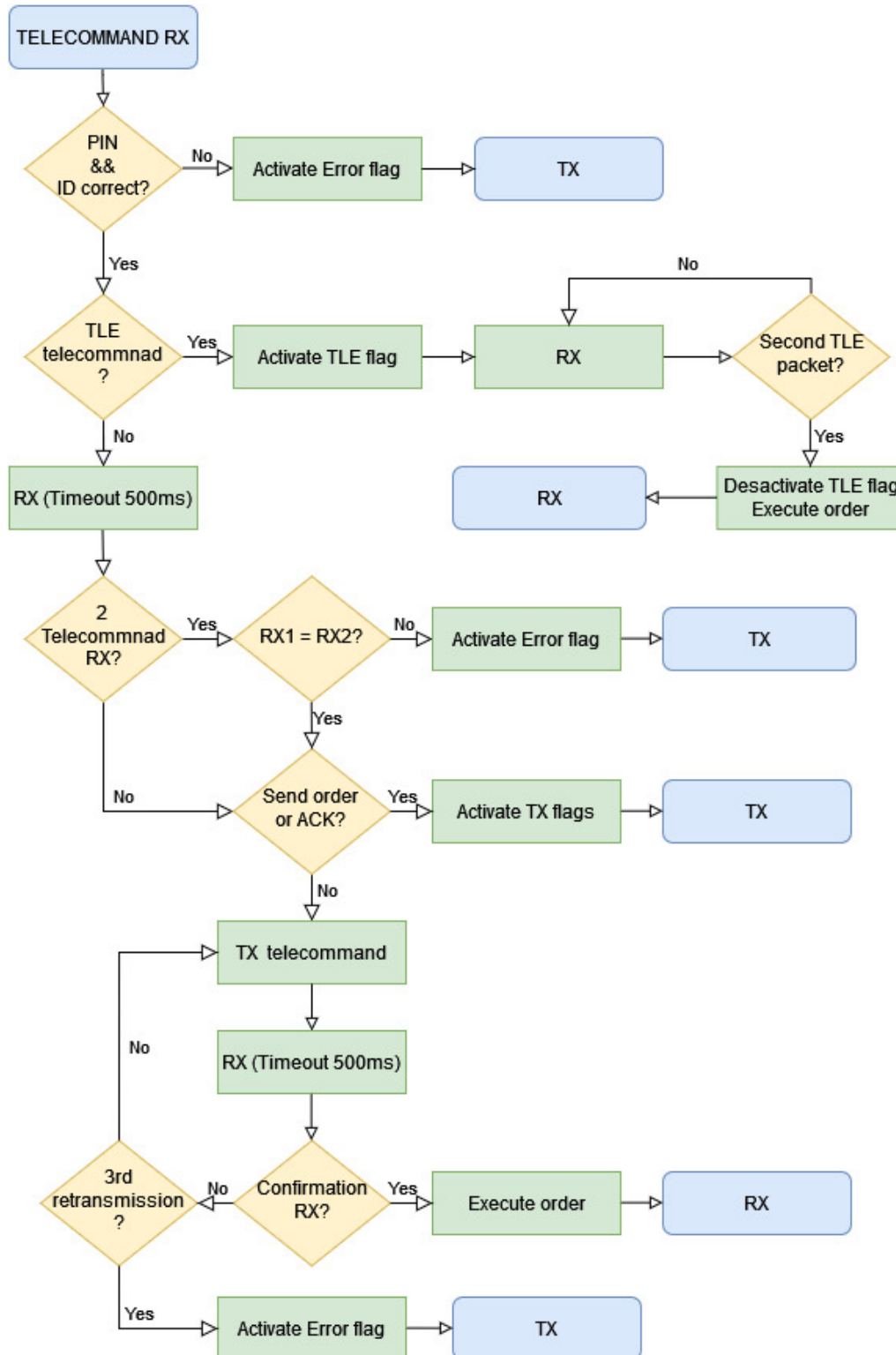


Figure 4.7: Telecommand reception and execution protocol flowchart

⁵https://github.com/daniel-herencia/Nucleo-L476RG-COMMS_TEST

5. LINK AND DATA BUDGET

5.1 Link Budget

To compute the link budget, we are going to use the information presented in table 5.1 (for more details, see appendix D).

Table 5.1: Link Budget parameters

Montsec GS	PocketQube	Channel
$El_{min} = 5^\circ$	$P_{tx-max} = 22 \text{ dBm}$	$h = 550 \text{ km}$ circular orbit
$\theta_p < 10^\circ$	Control error $\pm 10^\circ$	(unknown, worst case)
Circular polarization	Linear polarization	$L_{pol} = 3 \text{ dB}$
$T_a = 750 \text{ K}$ (rural noise)	$T_a = 290 \text{ K}$ (space)	$f = 868 \text{ MHz}$
$G_{max} = 12.8 \text{ dB}$ (Yagi)	$G_{max} = 1.15 \text{ dB}$ (Monopole)	$\lambda = 0.3456 \text{ m}$
$G(< 10^\circ) = 11.8 \text{ dB}$	RF losses = 5 dB	Ionosphere scintillation
$\theta_{3dB} = 36^\circ$	$\theta_{3dB} = 60^\circ$	1 dB (peaks up to 20 dB , typically last 0.4 s)
$T_{RX} = 450 \text{ K}$	$f_{offset-max} = \pm 25\% \cdot BW$	$v_{sat} = \sqrt{\frac{\mu}{h+R_E}} = 7.59 \frac{\text{km}}{\text{s}}$
$T_{sky} = 20 \text{ K}$	Sensitivity = $[-117, -148] \text{ dBm}$	$L_{FS} = 20 \log\left(\frac{4\pi R}{\lambda}\right) [\text{dB}]$
$42^\circ \text{ N}, 0.73^\circ \text{ E}$	$L_{point} \simeq 1 \text{ dB}$ (10^0)	

However, before starting, it is necessary to choose some LoRa parameters which will determine the sensitivity. The first one is the Bandwidth. With the selected transceiver, there are 4 possible values: 10.4, 125, 250 and 500 KHz. Largest BW have worst sensitivity. Therefore, the smallest one have to be chosen, but considering that the maximum frequency drift tolerated by the transceiver should be higher than the doppler.

$$\theta_{max} = \arcsin\left(\frac{R_E \cdot \cos(El_{min})}{R_E + h}\right) = 66.49^\circ \quad f_d = \frac{v_{sat}}{c} \cdot f \cdot \sin(\theta_{max}) = 20.14 \text{ kHz} \quad (5.1)$$

Therefore, the minimum bandwidth is $BW_{min} = f_d/0.25 = 80.55 \text{ kHz}$. Then, the optimum value is 125 kHz, which provides a sensitivity of -124 dBm for $SF = 7$, and -137 dBm for $SF = 12$.

As the transceiver of the Ground Station has not been decided yet, it will be considered that the same as in the satellite is used. Usually the power limitations are more critical in the satellite, therefore, the downlink will be computed, as the critical link. The power radiated by the satellite is the following:

$$EIRP_{sat} = G_{max} - L_{point} + P_{tx} - L_{RF} = 17.15 \text{ dBm} \quad (5.2)$$

And the effect of the channel:

$$L_{channel} = L_{FS} + L_{scint} + L_{pol} = 158.08 + 1 + 3 = 162.08 \text{ dB} \quad (5.3)$$

The gain and noise at the Ground station are computed as follows:

$$\begin{aligned} G_{GS} &= G_{RX} - L_{point} = 11.8 \text{ dB} \\ P_{noise_{DL}} &= 10 \log(k(T_a + T_{RX})BW) + 30 = -116.84 \text{ dBm} \end{aligned} \quad (5.4)$$

Then, the total power received is the following:

$$\begin{aligned} P_{RX} &= EIRP_{sat} - L_{channel} + G_{GS} = -133.13 \text{ dBm} \\ SNR_{downlink} &= P_{RX} - P_{noise} = -16.29 \text{ dB} \end{aligned} \quad (5.5)$$

Figure 5.1 shows the received power and the SNR as a function of the elevation, with the sensitivity for the different SF.

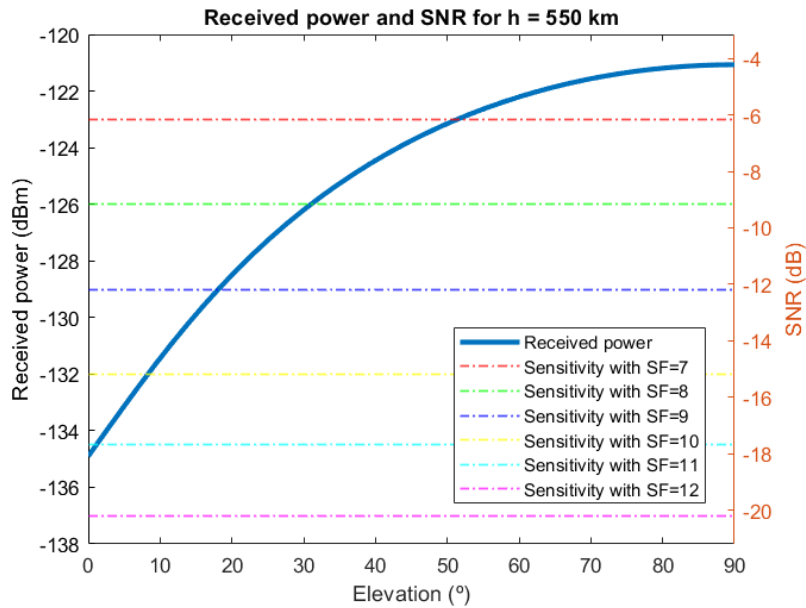


Figure 5.1: Received power and SNR as a function of the elevation

Therefore, the minimum elevation angle that makes possible to establish a link, for each spreading factor, is presented in table 5.2.

Table 5.2: Minimum elevation angle and sensitivity

	SF 7	SF 8	SF 9	SF 10	SF 11	SF 12
El_{min}	51°	31°	18.5°	9°	1°	0°
Sensitivity (dBm)	-123	-126	-129	-132	-134.5	-137

Interference between PocketQube has not been considered because as a LoRa receiver can demodulate simultaneously signals with different SF without collision, an option could be transmitting with different spreading factors in each satellite. Another option is to use different frequencies (separating them 200 kHz should be more than enough to avoid interference).

5.2 Data Budget

In order to compute the bit rate and determine the best SF, it has to be considered that the higher the SF is, the longer the contact time lasts. However, as presented in table 5.3, packets with higher values of SF need more separation in time and have a smaller size. In LoRa, the time on air is the amount of time between the signal is send by the sender and it is fully received by the receiver.

Table 5.3: LoRa parameters depending on the SF

	SF 7	SF 8	SF 9	SF 10	SF 11	SF 12
Max. payload size (bytes)	222	222	115	51	51	51
Data packet size used (bytes)	100	100	100	50	50	50
Air time for data packet (ms)	189.7	338.4	615.4	698.4	1478.7	2793.5
Bitrate (bps)	4217	2364	1300	573	270	143

In the practice, it has been tested that the real time between packets (the necessary time to receive and process them correctly) is higher than the theoretical shown in table 5.3. Nevertheless, it can be used to determine the optimum SF. Considering the minimum elevation angle, the average contact time, and the bitrate provided by each SF, the amount of data transmitted per day is shown in table 5.4. To compute the average contact time, STARLINK-1506 orbit has been used (because its height is approximately 550 km) [28][29].

Table 5.4: Average contact time and amount of data transmitted per day

	SF 7	SF 8	SF 9	SF 10	SF 11	SF 12
Average contact time per day (s)	360	720	1200	1560	1560	1560
Data transmitted per day (kBytes)	189.77	212.76	195	111.74	52.65	27.89

With these results, the optimum spreading factor is 8. However, as mentioned before, this is only an illustrative result, and a more detailed data budget should be done with the real values of the orbit and empirical values of the bitrate and airtime for each SF.

Finally, LoRa also allows to change the redundancy parameter (CR). By default it is set to 4/5 to maximize the bitrate, but the values of 4/6, 4/7 and 4/8 are also available and provide more robustness to interference at the expenses of longer transmission time. All the previous values of the data budget has been computed considering a CR of 4/5.

6. CONCLUSIONS AND FUTURE WORK

6.1 Conclusions

The purpose of this final degree thesis was the design, assembly, and testing of a complete communication system for a PocketQube. After establishing the requirements and specifications, LoRa technology was selected among the state-of-the-art of low power and long range communications technologies. Based on this decision, the subsystem was designed to work at 868 MHz, within Europe's ISM band.

First, a monopole antenna has been designed and integrated in a lateral face of the PocketQube, with a deployment system, to fulfill the specifications. The deployment system is controlled by the OBC by means of a switch, and the monopole has enough flexibility to be folded without exceeding the limits specified in the PocketQube standard. In order to ensure the correct operation of the antenna, it has been matched to the desired frequency, and the radiation pattern has been measured, providing a maximum gain of 1.15 dB. Also, the deployment system has been tested in the vacuum with a TVAC, obtaining a deployment in 5 seconds with a 1.65 W power supply. After that, the circuit to connect the transceiver, switch between the transmitted and received signals, filter them, and match the impedance, was designed and converted into an RF PCB.

Once all the hardware tested, the protocol to rule the communications subsystem was defined and programmed. This protocol makes it possible the exchange of information, data and commands between the satellite and the ground station. The different packets and telecommands have been defined, and the satellite knows when it has contact thanks to the telecommands sent by the ground station.

Finally, a study of the link and data budget has been done in order to accurately establish when the communication is possible, define the optimum LoRa parameters to maximize the bit rate in transmission, and assure the minimum packet loss. Besides that, redundancy has been added to the packets, and also a system of acknowledgment to retransmit the lost ones. In the end, the protocol has been tested, in evaluation boards, with satisfactory results.

Summarizing, all the initially established requirements and specifications have been satisfactorily met, and despite being room for improvement, the system is ready to work in a PocketQube mission.

6.2 Future Developments

The next step in this project should be testing all the protocol inserting the code in the OBC-COMMS PCB. This could not be done before because the progress done by the students in charge of the OBC subsystem has not been as fast as in the COMMS subsystem, and some previous steps are necessary before this test.

One improvement, which is not critical, as seen in the link budget, but that could improve the bit rate, is reducing the losses in the RF part of the COMMS PCB. This value was a little bit larger than expected, so a more detailed analysis of the circuit and an optimization of the matching networks and components could reduce the losses.

Finally, an interesting study that could be done is checking if it is worth using more robust encodings, such as Reed-Solomon or convolutional codes, at the expense of losing bitrate and memory in the OBC, but avoiding packet loss (and sending ACKs) due to scintillation effects of the ionosphere.

Bibliography

- [1] Joseph Pelton. “The start of commercial satellite communications [History of communications]”. In: *Communications Magazine, IEEE* 48 (Apr. 2010), pp. 24–31. DOI: 10.1109/MCOM.2010.5434368.
- [2] Wikipedia. *Sputnik 1*. URL: https://en.wikipedia.org/wiki/Sputnik_1.
- [3] “Introduction”. In: *Satellite Communications Systems*. John Wiley & Sons, Ltd, 2020. Chap. 1, pp. 1–27. ISBN: 9781119673811. DOI: <https://doi.org/10.1002/9781119673811.ch1>. eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119673811.ch1>. URL: <https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119673811.ch1>.
- [4] Adriano Camps. “Nanosatellites and Applications to Commercial and Scientific Missions”. In: Nov. 2019. ISBN: 978-1-78985-995-9. DOI: 10.5772/intechopen.90039.
- [5] Nanosats database. *Nanosatellite launches by type*. URL: https://www.nanosats.eu/img/fig/Nanosats_years_types_2022-01-01.png.
- [6] eoPortal. *CubeSat concept*. URL: <https://earth.esa.int/web/eoportal/satellite-missions/c-missions/cubesat-concept>.
- [7] Mr Heidt et al. “CubeSat: A new generation of picosatellite for education and industry low-cost space experimentation”. In: (Jan. 2000).
- [8] GAUSS SRL. *PocketQubes*. URL: <https://www.gaussteam.com/services/satellite-design/pocketqubes/>.
- [9] Wikipedia. *PocketQube*. URL: <https://en.wikipedia.org/wiki/PocketQube>.
- [10] FOSSA. *A new access to Space*. URL: <https://fossa.systems/our-history/>.
- [11] Albaorbital. *Democratizing access to space*. URL: <http://www.albaorbital.com/>.
- [12] Alba Orbital TU Delft and Gauss Srl. “The PocketQube Standard”. In: (June 2018). URL: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&cad=rja&uact=8&ved=2ahUKEwjX8eOv-NT2AhVDqxoKHQmvBa4QFnoECAMQAQ&url=https%3A%2F%2Fstatic1.squarespace.com%2Fstatic%2F53d7dcdce4b07a1cdbbc08a4%2Ft%2F5b34c395352f5303fcec6f45%2F1530184648111%2FPocketQube%2BStandard%2Bissue%2B1%2B-%2BPublished.pdf&usg=AOvVaw1FasGZiQZgou3Y85gAYgSu>.

- [13] Sumit Karki and Adriano José Camps Carmona. “CubeCat-1: Communications System of a Nano-satellite”. In: July 2013, pp. 15–28. URL: <https://upcommons.upc.edu/bitstream/handle/2099.1/19232/ThesisSumitKarki.pdf?sequence=4&isAllowed=y>.
- [14] TELES department ETSETB. “Unit 3: The Communication Payload”. In: pp. 2–22.
- [15] Luis J. Contreras. “Mechanical design and analysis of nano- and pico-satellites and deployers”. In: (May 2022).
- [16] Mark Bloechl. *SigFox Vs. LoRa: A Comparison Between Technologies and Business Models*. URL: <https://www.link-labs.com/blog/sigfox-vs-lora>.
- [17] Juan Fraire, Sandra Céspedes, and Nicola Accettura. “Direct-To-Satellite IoT - A Survey of the State of the Art and Future Research Perspectives: Backhauling the IoT Through LEO Satellites”. In: Sept. 2019, pp. 241–258. ISBN: 978-3-030-31830-7. DOI: 10.1007/978-3-030-31831-4_17.
- [18] GSMA. *Narrowband - Internet of Things (NB-IoT)*. URL: <https://www.gsma.com/iot/narrow-band-internet-of-things-nb-iot/>.
- [19] Nirod K. Das. “6 - Antennas and Radiation”. In: *The Electrical Engineering Handbook*. Ed. by WAI-KAI CHEN. Burlington: Academic Press, 2005, pp. 553–569. ISBN: 978-0-12-170960-0. DOI: <https://doi.org/10.1016/B978-012170960-0/50042-6>. URL: <https://www.sciencedirect.com/science/article/pii/B9780121709600500426>.
- [20] Andreea Constantin, Razvan Tamas, and Alexandru Brumaru. “Radiation from Common Mode Currents on Coaxial Lines Feeding Small Monopole Antennas”. In: Feb. 2021, pp. 1–5. DOI: 10.1109/ConfTELE50222.2021.9435449.
- [21] CTR Engineering. *Common mode current*. URL: <http://www.jdunman.com/www/AmateurRadio/Electronics/Common%20mode%20current.htm>.
- [22] Constantine A. Balanis. *Antenna Theory: Analysis and Design*. USA: Wiley-Interscience, 2005. ISBN: 0471714623.
- [23] Semtech. *SX1261/2 Long Range, Low Power, sub-GHz RF Transceiver*. Dec. 2021. URL: <https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000Un7F/yT.fKdAr9ZAo3cJLc4F2cBdUsMftpT2vsOICP7NmvMo>.
- [24] Semtech. *Application Note: Reference Design Explanation*. May 2018. URL: <https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000HSSf/GT2IXjK2nH8bw6JdEXfFBd.HmFATeLopL402mZwpSho>.
- [25] Semtech. *SX1262MB2xAS mbed shield*. URL: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000UhsP/0RkqtXdTfF7wC0agls1NHH0.RgupS_ElUJ3glIj5XHk.
- [26] JLCPCB. *Multilayer high precision PCB’s with impedance control*. URL: <https://cart.jlpcb.com/impedance>.
- [27] Saturn PCB Design Inc. *Saturn PCB Design Toolkit Version 8.10*. URL: <https://saturnpcb.com/saturn-pcb-toolkit/>.

- [28] Dr. T.S. Kelso. *CelesTrak*. URL: <https://www.celestrak.com/NORAD/elements/gp.php?INTDES=2020-038>.
- [29] N2YO. *N2YO 10-DAY PREDICTIONS*. URL: <https://www.n2yo.com/passes/?s=45747>.
- [30] Semtech. *LoRa and LoRaWAN Documentation*. URL: <https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/>.
- [31] 2CI Group. *Conceptos de actualidad: LoRa y LoRaWan*. URL: <https://www.2cigroup.com/es/conceptos-de-actualidad-lora-y-lorawan/>.
- [32] Upverter Blog admin. *When to Use Coplanar Waveguide Routing for HF Boards*. URL: <https://blog.upverter.com/2019/10/15/when-to-use-coplanar-waveguide-routing-for-hf-boards/>.
- [33] JLCPCB. *Capabilities: Know JLCPCB's Capabilities and Get your PCBs Built Fast*. URL: <https://jlcpcb.com/capabilities/Capabilities>.
- [34] Semtech. *Application Note: Recommendations for Best Performance*. Jan. 2018. URL: https://semtech.my.salesforce.com/sfc/p/#E0000000JelG/a/2R000000HSRc/sHUFJGBBHvYnK9cdfu.jexM2OpMe.LWQG6PpY_wg9YE.
- [35] Park Hyuk. *FSSCat assemble is progressing*. URL: <https://prs.upc.edu/2018/11/21/fsscat-assemble-is-progressing/>.
- [36] *Encyclopedia of physical science and technology*. eng. 3rd ed. San Diego: Academic Press, 2002. ISBN: 0122274105.
- [37] TELES department ETSETB. "Unit 4: Satellite Communication Channel". In: pp. 10–11.
- [38] Akos Farago, Peter Kantor, and Janos Bitó. "Rain Effects on 5G millimeter Wave ad-hoc Mesh Networks Investigated with Different Rain Models". In: *Periodica Polytechnica Electrical Engineering and Computer Science* 60 (Jan. 2016), pp. 44–50. DOI: 10.3311/PPEe.8644.
- [39] International Telecommunication Union. "Recommendation ITU-R P676-10. Attenuation by atmospheric gases". In: (Sept. 2013).
- [40] Lara Fernandez et al. "Assessing LoRa for Satellite-to-Earth Communications Considering the Impact of Ionospheric Scintillation". In: *IEEE Access* 8 (Jan. 2020), pp. 165570–165582. DOI: 10.1109/ACCESS.2020.3022433.
- [41] Paul M. Kinter, Todd E. Hmphreys, and Joanna C. Hinks. "CGnss and ionospheric scintillation". In: (July 2009). URL: www.insidegnss.com/auto/julyaug09-kintner.pdf.
- [42] Maties Pons Subirana and Adriano José Camps Carmona. "High Speed S-band Communications System for Nanosatellites". In: June 2016, pp. 17–21. URL: <https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=&ved=2ahUKEwj958e8hbX4AhUTKhoKHem5DkwQFnoECAYQAQ&url=https%3A%2F%2Fupcommons.upc.edu%2Fbitstream%2Fhandle%2F2117%2F90146%2FMasterThesisMatiesPonsSubirana.pdf&usq=AOvVawlgqYfvO-g1SylwPT60y9Vhy>.

- [43] botland. *STM32 NUCLEO-L476RG - with STM32L476RGT6 ARM Cortex M4 MCU*. URL: <https://botland.store/stm32-nucleo/18800-stm32-nucleo-l476rg-with-stm32l476rgt6-arm-cortex-m4-mcu-5904422364977.html>.
- [44] Semtech - arm MBED. *SX126xMB2xAS*. URL: <https://os.mbed.com/components/SX126xMB2xAS/>.
- [45] Heltec Automation. *CubeCell - Dev-Board*. URL: <https://heltec.org/project/htcc-ab01/>.

A. Theoretical Background

A.1 LoRa

LoRa is an RF modulation technology for low-power, wide area networks (LPWANs) [30]. LoRa is purely implemented in the physical layer (PHY in OSI).

Traditional DSSS systems transform the carrier phase of the transmitter signal according to a code sequence, at a much higher rate, which is called spreading code or chip sequence. This spreads the signal bandwidth. When the signal is received, it is multiplied by the same spreading code of the transmitter, to recover the original information. This systems allow the receiver to recover the information even if the SNR is negative. But DSSS systems require highly-accurate reference clocks [30].

LoRa uses CSS-based modulation. Chirp Spread Spectrum is an alternative to DSSS systems which not requires a high precision clock and that can operate at low power still maintaining the robustness. In CSS, the spreading of the spectrum is achieved with chirp signals that continuously vary in frequency. Chirp Spread Spectrum is shown in figure A.1.

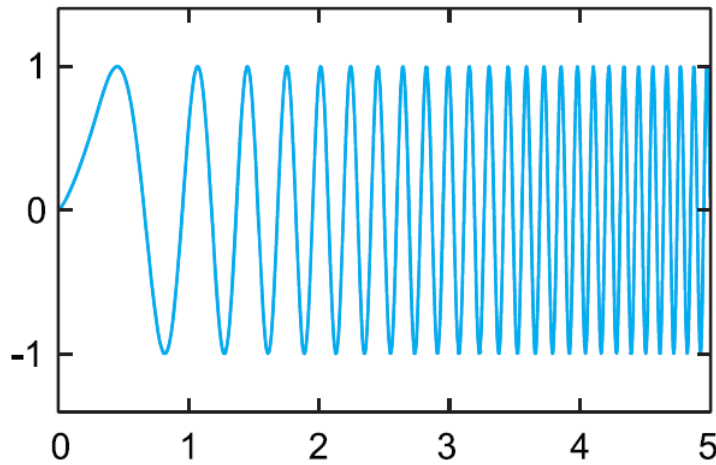


Figure A.1: LoRa CSS. Image from [30]

The symbol transmitted, or chirp, is a simple ramp from the minimum frequency up to the maximum frequency (given that $f_{max} - f_{min} = BW$) or vice versa. In the image A.2, the first 8 are basic chirps, the first 6 are the so-called “up-chirps”, used as a preamble to facilitate the receiver the detection of the signal, and the last two are “down-chirps”, used for synchronization purpose.

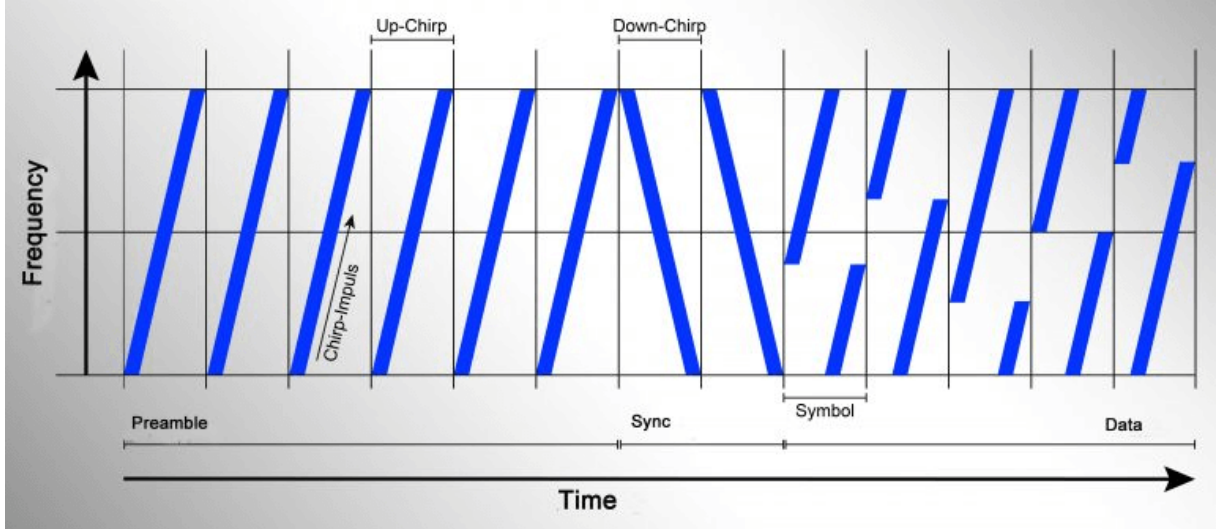


Figure A.2: LoRa symbols. Image from [31]

LoRa uses orthogonal spreading factors (SF) to optimize the communication in terms of power levels and data rates, depending on the channel and the device conditions. High values of spreading factor increase the processing gain and the RX sensitivity, at the cost of reducing the data rate. The orthogonality of the spreading factor allows transmitting signals in the same frequency channel at the same time, without interfering with each other (they will appear to be noise to each other). And what is more, two packets that arrive at the same time with different SF, both can be demodulated by the modem. If they have the same SF, to avoid collision, one should be at least 6 dB stronger than the other [30].

The transmitted signal is a constant envelope signal. Equivalently, one chip is sent per second per Hz of bandwidth. To generate symbols/chirps, the modem modulates the phase of an oscillator, and the number of times per second that the modem adjusts the phase is known as the chip rate. The number of chips contained in each symbol is 2^{SF} , the lower the SF, the faster the frequency sweep. The number of bits that can be encoded by each symbol/chirp is SF.

LoRa modulation also includes an error correction scheme, that can be adjusted with the CR factor. The minimum is 4/5, which adds a fifth bit of parity every four bits sent. This improves the robustness of the modulation.

The following expressions provided by Semtech in the transceiver datasheet [23] can be used to compute the bit rate:

$$SF \cdot CR = \frac{\text{bits}}{\text{symbol}} \cdot \frac{\text{Information bits}}{\text{Total bits}} = \text{Info. bits per symbol} [\text{bits/symbol}] \quad (\text{A.1})$$

$$\text{Symbol period} : T_s = \frac{\text{Number of chips per symbol}}{BW} = \frac{2^{SF}}{BW} [s] \quad (\text{A.2})$$

$$\text{Total bit rate} = \frac{\text{Bits per symbol}}{\text{Symbol period}} = \frac{SF}{\frac{2^{SF}}{BW}} [bps] \quad (\text{A.3})$$

$$\text{Information bit rate} = \frac{\text{Information bits per symbol}}{\text{Symbol period}} = \frac{SF \cdot CR}{\frac{2^{SF}}{BW}} [\text{bps}] \quad (\text{A.4})$$

A.2 RF lines and waveguides

There are different alternatives to guide the signals in RF circuits, which are shown in figure A.3 and whose main characteristics are the following:

- *Microstrip lines*: type of transmission line where the conductor that guides the signal is separated from the ground plane by a dielectric (substrate). They are easy to model, at the expense of lower power handling capacity, and higher losses when increasing the frequency.
- *Striplines*: in this case, the conductor is covered by a ground plane above and below. It works as a transverse electromagnetic (TEM) transmission line medium. The drawback is that 3 different layers are used to design this kind of lines, and that components cannot be welded.
- *Coplanar waveguide*: arrangement where the signal trace is routed in parallel to 2 ground planes. This layout provides a good shielding against interference and reduces radiation losses. Resistive heating losses are also reduced [32].
- *Grounded coplanar waveguide*: variety of coplanar waveguide that uses another ground plane underneath, connected with vias. This works as a third return conductor which provides even better losses than the previous geometry.

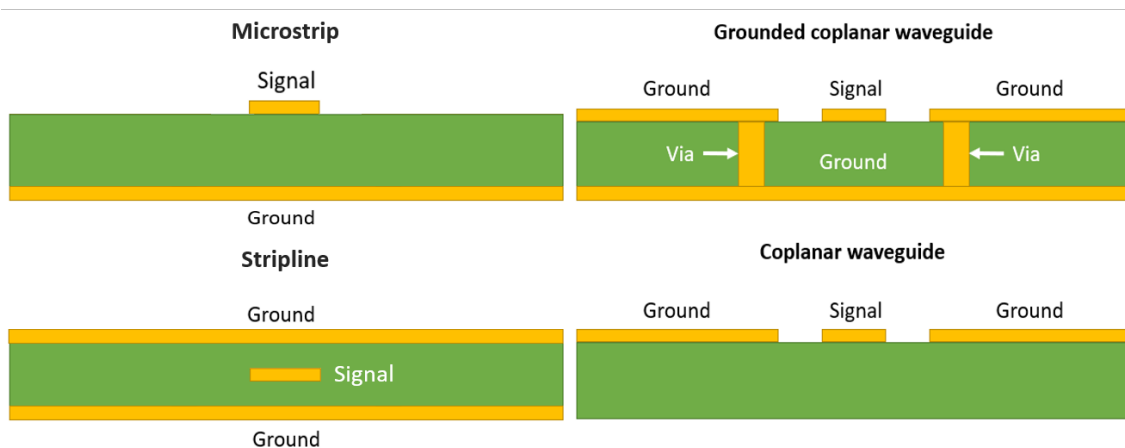


Figure A.3: Types of RF lines and waveguides

B. Hardware schematics and details

B.1 OBC-COMMS schematic

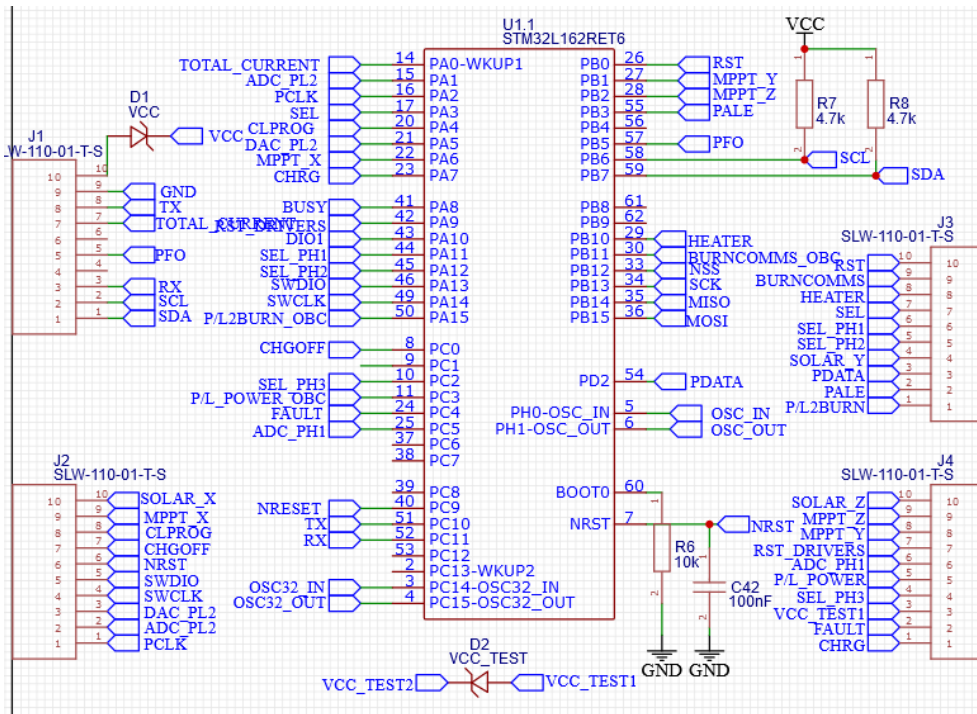


Figure B.1: STM32 pin connections

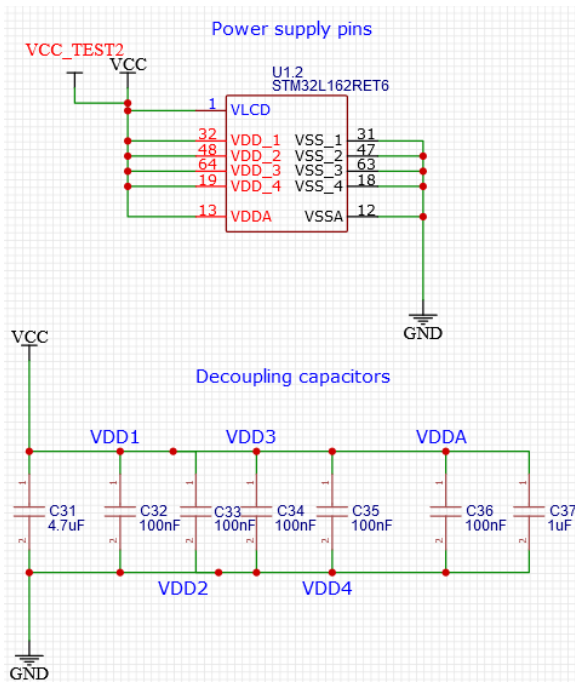


Figure B.2: STM32 power supply connections

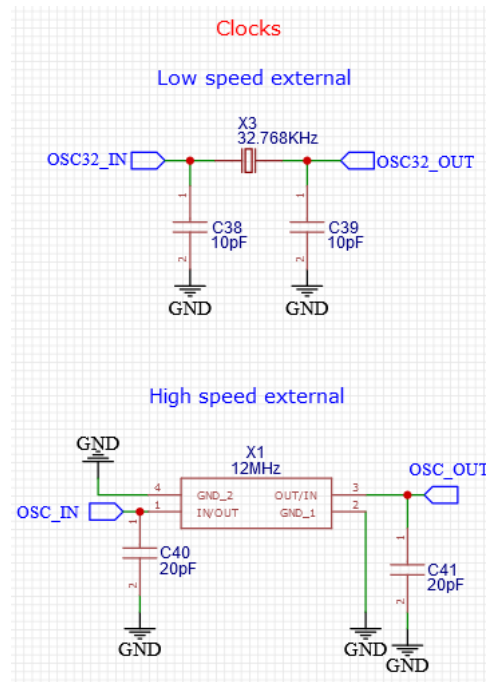


Figure B.3: STM32 reference clocks connections

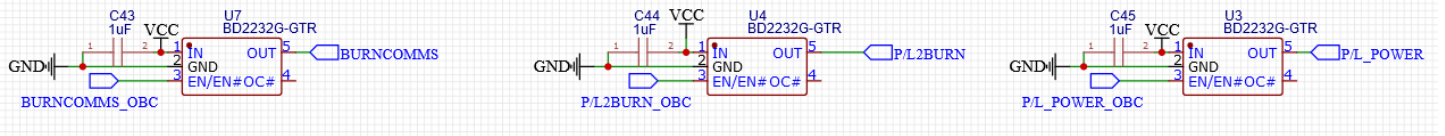


Figure B.4: Power and deployment switches connections

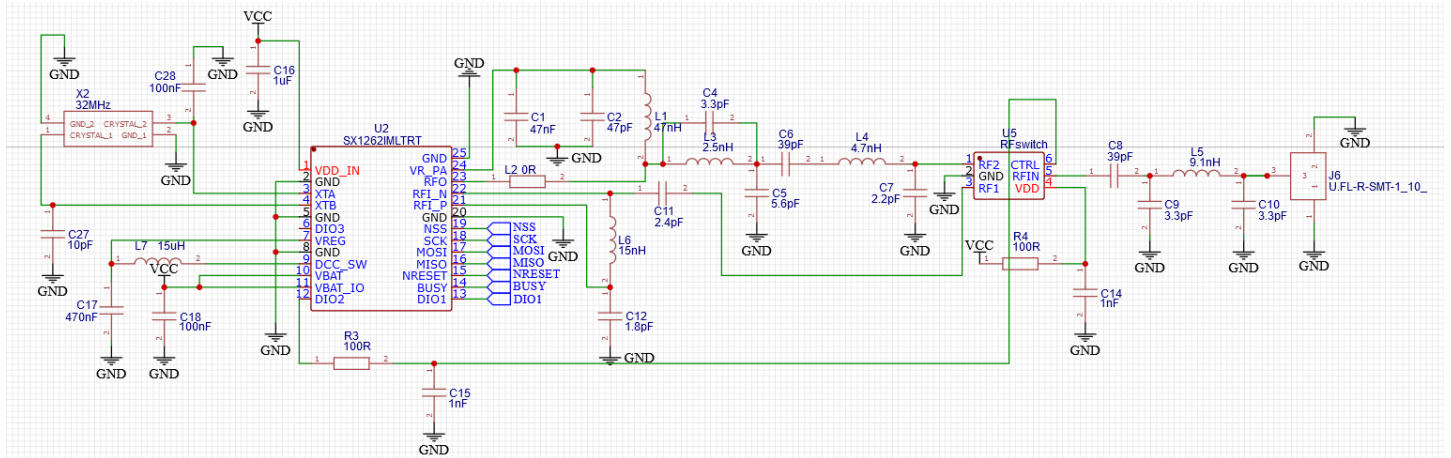
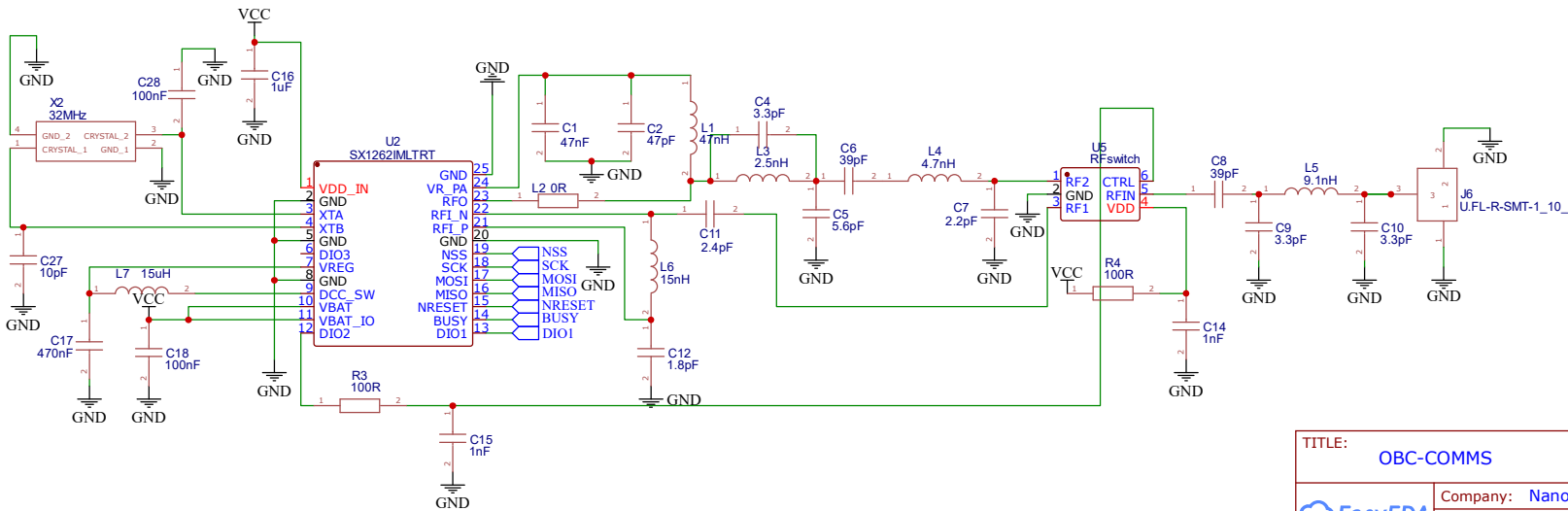
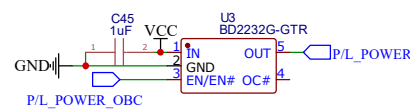
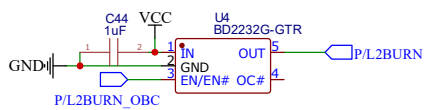
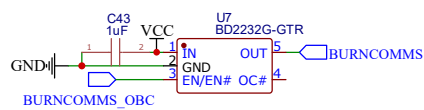
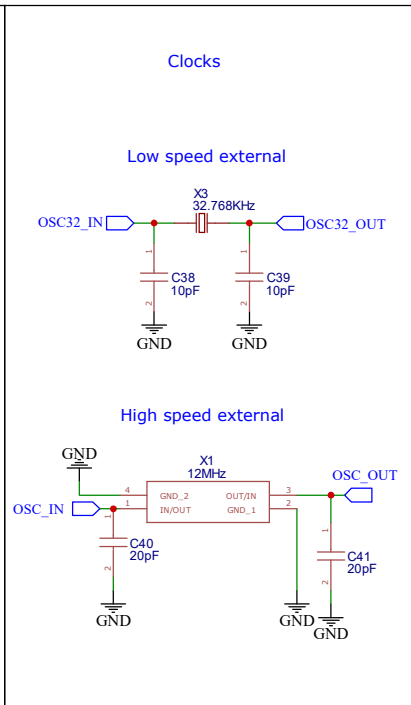
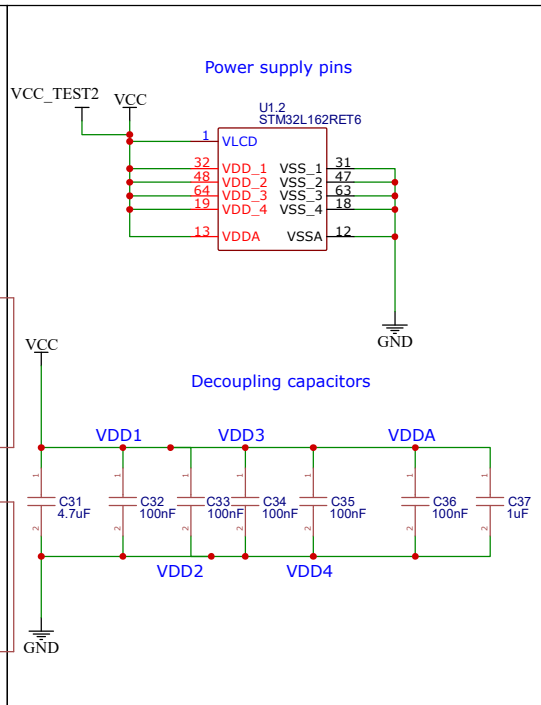
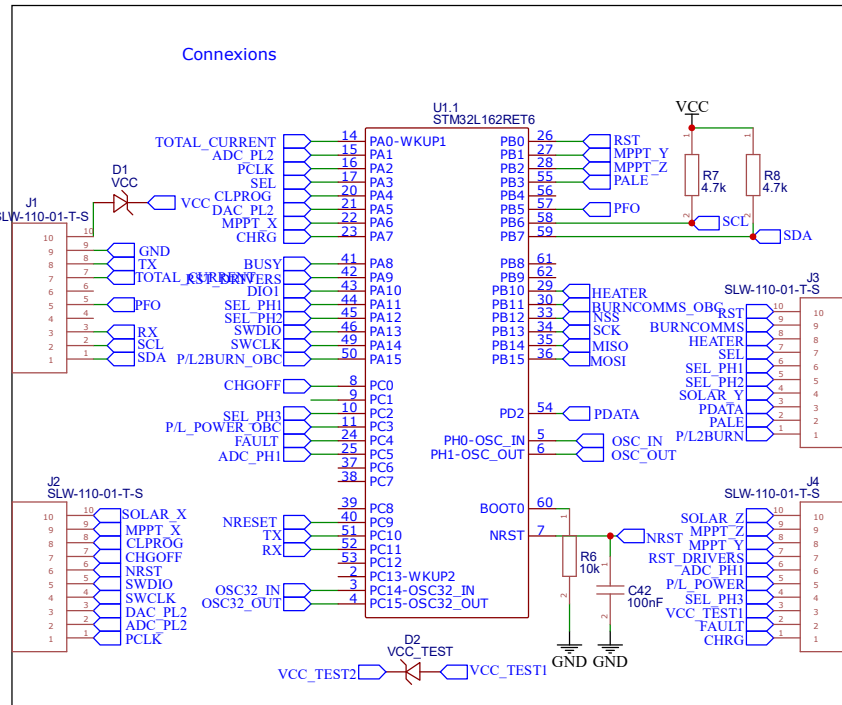


Figure B.5: COMMS schematic connections



TITLE:	OBC-COMMS	REV: 3.0
Company:	NanoSat Lab - Telecos BCN	Sheet: 1/1
Date:	2022-05-28	Drawn By: Daniel Herencia



B.2 JLCPCB design rules

Table B.1: JLCPCB's Capabilities summary [33]

FEATURES	CAPABILITIES
Dielectric material	FR-4 (4.6 dielectric constant)
Board thickness	0.4/0.6/0.8/1.0/1.2/1.6/2.0 mm
Finished outer layer copper	1 oz/2 oz (35 um/70 um)
Drill hole size	0.20-6.30 mm
Min. via hole size	0.2 mm
Min. via diameter	0.45 mm
Min. pad size	1.0 mm
Hole to hole clearance	0.5 mm (diff. nets) / 0.254 mm (same nets)
Pad to pad clearance	0.127 mm
Via to track	0.254 mm
Pad to track	0.2 mm
Minimum trace width	0.09 mm (4-6 layers)
Minimum tarce spacing	0.09 mm (4-6 layers)
Solder mask color	green, red, yellow, blue, white and black

B.3 OBC-COMMS design criteria

The design of the COMMS PCB has been based on Semtech SX1262 Reference Design, which is presented in figures B.6, B.7, B.8 and B.9.

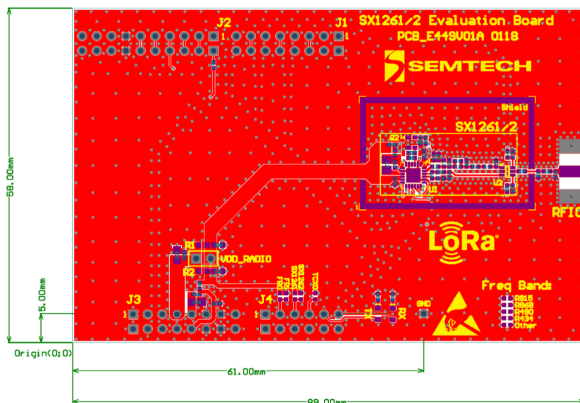


Figure B.6: SX1262 Reference Design Layout – Top Layer. Image from [24]

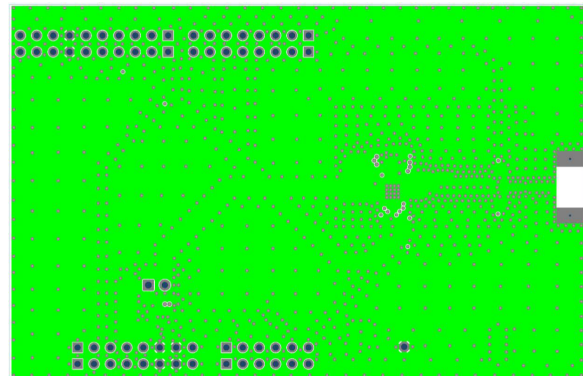


Figure B.7: SX1262 Reference Design Layout – Layer 2. Image from [24]

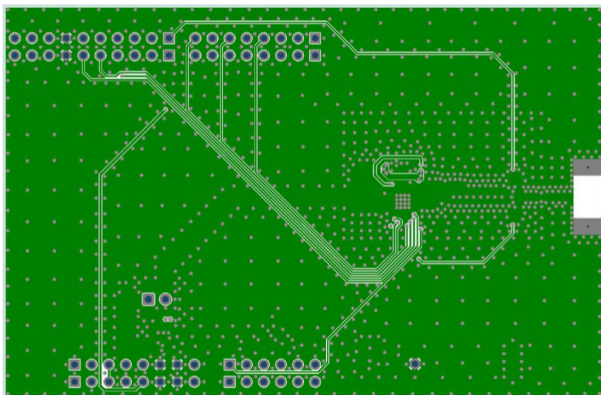


Figure B.8: SX1262 Reference Design Layout – Layer 3. Image from [24]

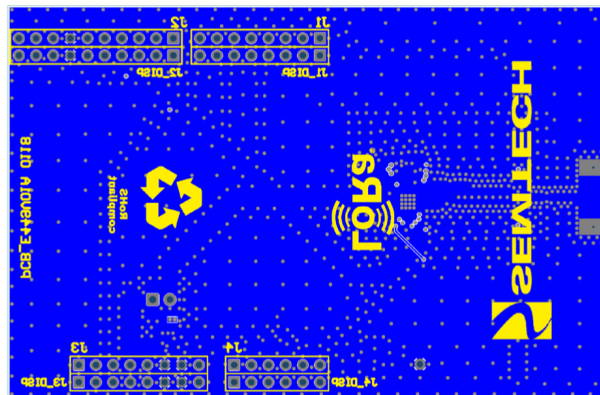


Figure B.9: SX1262 Reference Design Layout – Bottom Layer. Image from [24]

In figures B.10 and B.11 can be seen that the layout of the components is very similar in both designs.

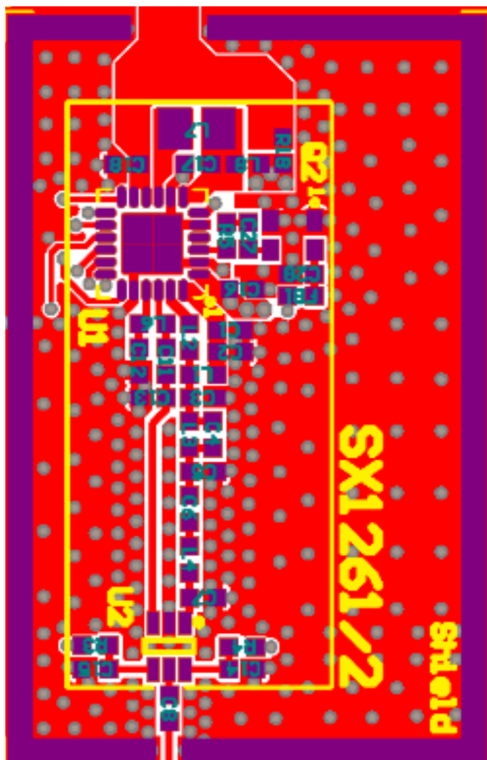


Figure B.10: SX1262 Reference Design RF part transceiver circuit. Image from [24]

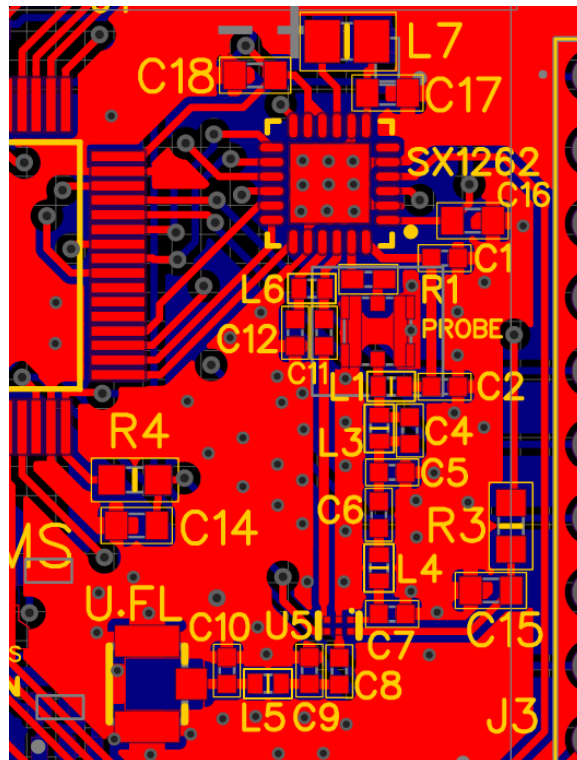


Figure B.11: OBC-COMMS PCB RF part transceiver circuit. Image from [24]

The rules that have been followed in the RF design are the followings:

- All the capacitors, inductors and resistors of the RF part are 0402 in size (1 x 0.5 mm) in order to have a footprint with similar width to the lines. And all the ones that are not in RF lines (for example, C15, R3...) are 0603 (1.6 x 0.8 mm). The only exception is L7.

- The RF lines (tracks) have the minimum length as possible, in order not to lose adaptation. Vias have not been used in the RF part. All the RF lines are grounded coplanar waveguides with a width of 0.35 mm and a track-to-GND clearance of 0.2 mm. However, when the lines are close to a component with very small pads, the width of the track is reduced to avoid interference between lines (for example, when approaching the transceiver or the switch U5).
- In RF circuits, a steep bend (especially of 90°) can act as an antenna and radiate. Thus, making turns with tracks has been avoided and they have been made with components, except for some occasional cases where a 45° turn have been necessary. Also for this reasons, the R1 resistor of $0\ \Omega$ has been maintained for a 90° turn.
- When the probe is connected to a MS-156C coaxial cable, one extreme is blocked (as in open circuit), and the other conducts the RF signal, as explained in figure B.12. When no cable is connected, the probes acts as a $50\ \Omega$ RF line (perfectly matched).

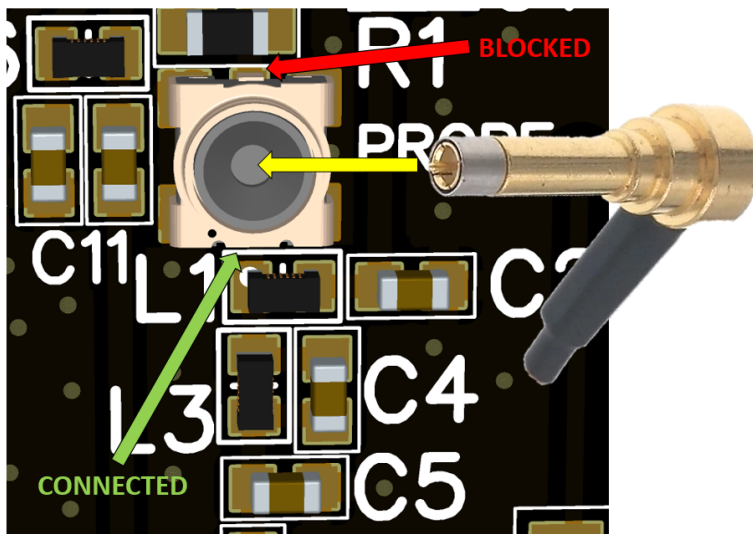


Figure B.12: Probe operation on the OBC-COMMS PCB

Finally, some thermal insulation techniques has been used in the PCB design. The first one is to introduce several ground vias under the transceiver, to distribute quickly the heat to all the layers of the PCB (and also to assure the correct ground reference in the transceiver). This is important, because the transceiver can sometimes get really hot, specially when transmitting at high power. The second technique, which is proposed in Semtech Recommendations for Best Performance document [34], is isolating the crystal oscillator to reduce the frequency drift with respect to the desired clock frequency. Figure B.13 shows Semtech reference design of the transceiver and crystal oscillator. Figure B.14 shows the footprint of the crystal oscillator in OBC-COMMS PCB. It is placed in the bottom layer, to avoid direct heat transfer from the transceiver (and also from the OBC), and the parts that have a dark blue tone have no copper area, to minimize the heat transfer reaching the crystal oscillator. Lastly, figure B.15 shows the ground vias under the transceiver in OBC-COMMS PCB.

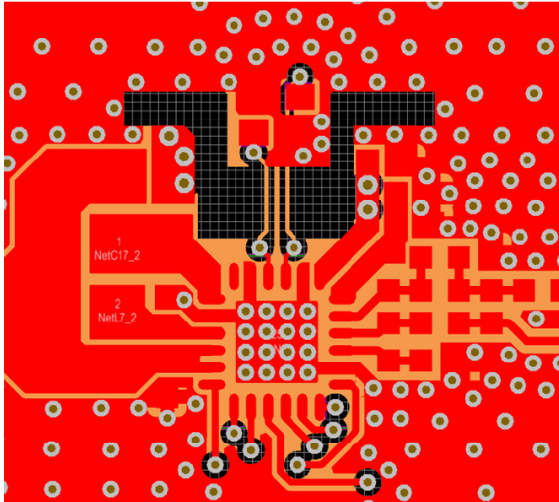


Figure B.13: Semtech Reference Design crystal oscillator and transceiver [24]

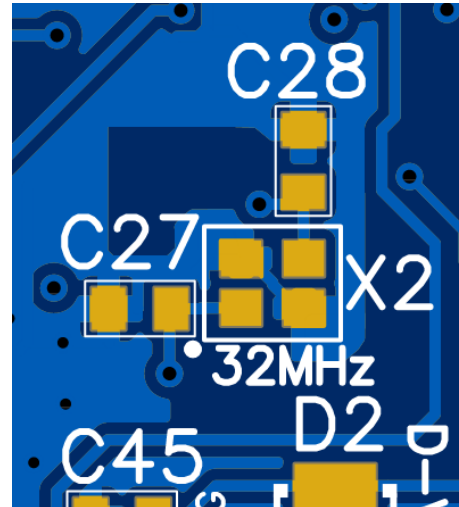


Figure B.14: OBC-COMMS PCB crystal oscillator

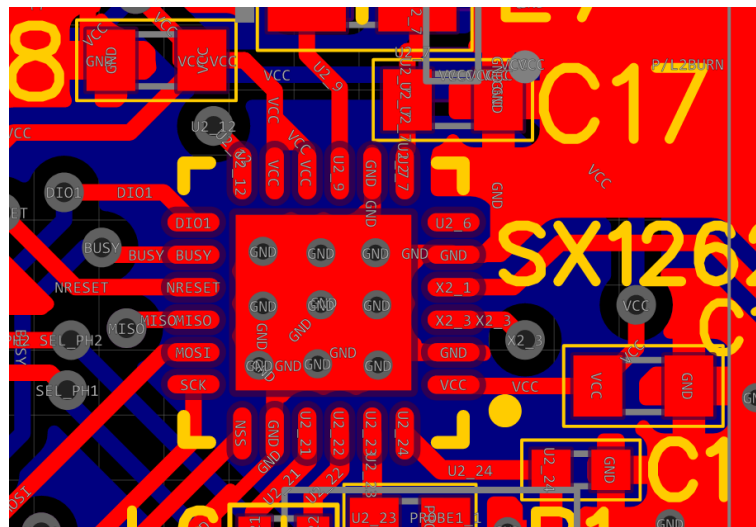


Figure B.15: OBC-COMMS PCB transceiver

One last important thing considered in the design is that the last pin of the component J4 (inferior lateral pin) have to be removed, in order to have enough space to exit with the microcoaxial cable that connects the COMMS circuit with the lateral board where the antenna is placed. This pin is used to connect other PCBs that are below OBC-COMMS. Therefore, it has no utility in this PCB, and can be removed. In figure B.16, the 3D model of the OBC-COMMS PCB with the coaxial cable is presented.

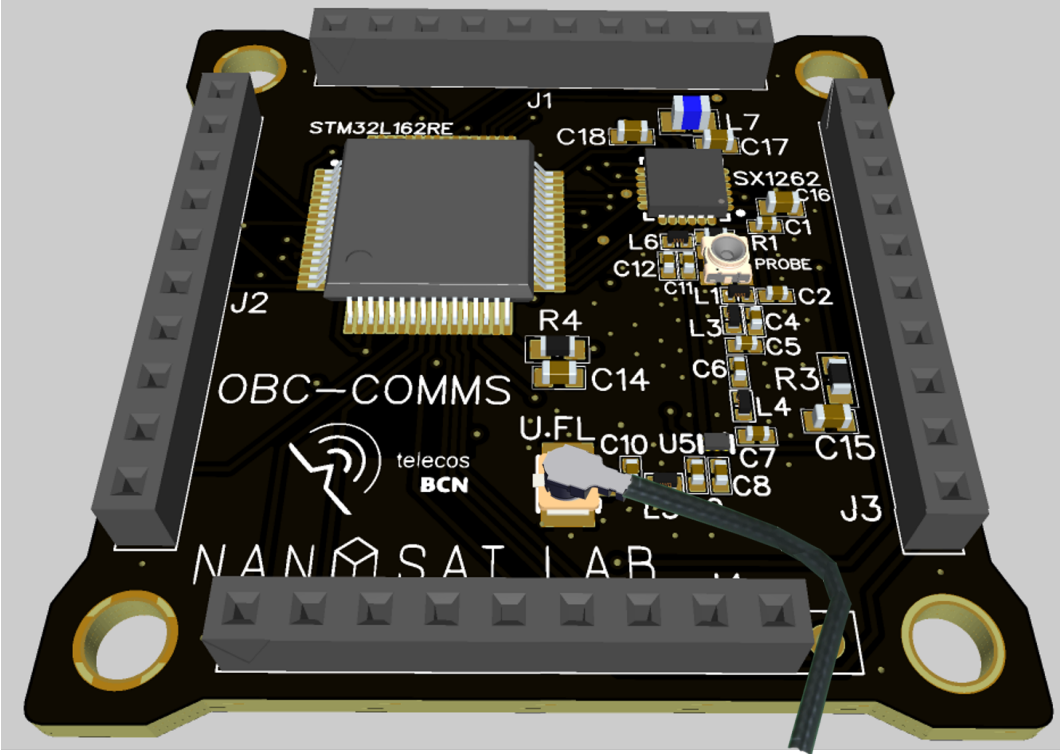


Figure B.16: 3D model of the OBC-COMMS PCB with the coaxial cable

B.4 List of Components

COLORS:	COMMS	OBC				
RefDes	MPN	Geom	Value	Qty	Description	Manufacturer
U2	SX1262IMLTRT	VQFN24 4x4mm	SX1262	1	150MHz to 960MHz Low Power Long Range Transceiver	Semtech
U5	BGS12WN6E6327XTSA1	1.2x0.8mm	BGS12	1	SPDT CIs para conmutadores RF CMOS SWITCH 50 MHz - 6.0 GHz	Infineon
U1	STM32L162RET6	13x13mm	STM32L1	1	MCU ARM Ultra-low-power Arm Cortex-M3 512 Kbytes of Flash 32 MHz CPU	STMicroelectronics
U3	BD2232G-GTR	SSOP-5	BD2232G	1	Small Pkg High Side Sw IC 2.7 V to 5.5 V 145 mOhms 1.3 A	ROHM Semiconductor
U4	BD2232G-GTR	SSOP-5	BD2232G	1	Small Pkg High Side Sw IC 2.7 V to 5.5 V 145 mOhms 1.3 A	ROHM Semiconductor
U7	BD2232G-GTR	SSOP-5	BD2232G	1	Small Pkg High Side Sw IC 2.7 V to 5.5 V 145 mOhms 1.3 A	ROHM Semiconductor
Resistors						
R3	ASC0402-100RFT10	0402	100R	1	Thick Film Resistor ±1%, 1/16W	Vishay/Dale
R4	CRCW0402100RFKED	0402	100R	1	Thick Film Resistor ±1%, 1/16W	Vishay/Dale
R6	CRCW020110K0FNED	0603	10k	1	Thick Film Resistors - SMD 1/20watt 10Kohms 1% 200ppm	Vishay
R7	CRCW02014K70FKED	0603	4.7k	1	Thick Film Resistors - SMD 1/20watt 4.7Kohms 1% 100ppm	Vishay
R8	CRCW02014K70FKED	0603	4.7k	1	Thick Film Resistors - SMD 1/20watt 4.7Kohms 1% 100ppm	Vishay
Capacitors						
C1	GRM155R71C473KA01J	0402	47nF	1	Multilayer ceramic capacitors X7R ±1%, 16V	Murata
C2	GRM1555C1H470FA01D	0402	47pF	1	Multilayer ceramic capacitors C0G ±5%, 50V	Murata
C4	GRM1555C1H3R3BA01D	0402	3.3pF	1	Multilayer ceramic capacitors C0G ±0.1pF, 50V	Murata
C5	GRM1555C1H5R6CA01D	0402	5.6pF	1	Multilayer ceramic capacitors C0G ±0.25pF, 50V	Murata
C6	GRM1555C1H390JA01D	0402	39pF	1	Multilayer ceramic capacitors C0G ±5%, 50V	Murata
C7	GRM1555C1H2R2BA01J	0402	2.2pF	1	Multilayer ceramic capacitors C0G ±0.1pF, 50V	Murata
C8	GRM1555C1H390JA01D	0402	39pF	1	Multilayer ceramic capacitors C0G ±5%, 50V	Murata
C9	GRM1555C1H3R3BA01D	0402	3.3pF	1	Multilayer ceramic capacitors C0G ±0.1pF, 50V	Murata
C10	GRM1555C1H3R3BA01D	0402	3.3pF	1	Multilayer ceramic capacitors C0G ±0.1pF, 50V	Murata
C11	GRM1555C1H2R4WA01D	0402	2.4pF	1	Multilayer ceramic capacitors C0G ±0.1pF, 50V	Murata
C12	GRM1555C1H1R8BA01D	0402	1.8pF	1	Multilayer ceramic capacitors C0G ±0.1pF, 50V	Murata
C14	0603ZD102KAT2A	0603	1nF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 10V 1000pF X5R 10%	Kyocera AVX
C15	0603ZD102KAT2A	0603	1nF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 10V 1000pF X5R 10%	Kyocera AVX
C16	0603D105KAT2A	0603	1uF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 1uF X5R 0603 10%	Kyocera AVX
C17	0603YD474KAT2A	0603	470nF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 16V 0.47uF X5R 0603 10%	Kyocera AVX
C18	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C27	06035A100GAT2A	0603	10pF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 10pF C0G 0603 2%	Kyocera AVX
C28	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C31	GRM185R61C475KE11J	0603	4.7uF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 4.7UF 16V 10% 0603	Murata
C32	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C33	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C34	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C35	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C36	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C37	0603D105KAT2A	0603	1uF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 1uF X5R 0603 10%	Kyocera AVX
C38	06035A100GAT2A	0603	10pF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 10pF C0G 0603 2%	Kyocera AVX
C39	06035A100GAT2A	0603	10pF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50V 10pF C0G 0603 2%	Kyocera AVX

C40	AC0603JRNPO9BN200	0603	20pF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50 V 20 pF C0G 0603 5%	YAGEO
C41	AC0603JRNPO9BN200	0603	20pF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 50 V 20 pF C0G 0603 5%	YAGEO
C42	C0603X103F5JACTU	0603	100nF	1	Multilayer ceramic capacitors MLCC - SMD/SMT 50V 0.01uF U2J 0603 1%	KEMET
C43	06033D105KAT2A	0603	1uF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 1uF X5R 0603 10%	Kyocera AVX
C44	06033D105KAT2A	0603	1uF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 1uF X5R 0603 10%	Kyocera AVX
C45	06033D105KAT2A	0603	1uF	1	Multilayer Ceramic Capacitors MLCC - SMD/SMT 25V 1uF X5R 0603 10%	Kyocera AVX
Inductors						
L1	LQW15AN47NJ00D	0402	47nH	1	Wirewound Inductor ±5%	Murata
L3	LQW15AN2N5C00D	0402	2.5nH	1	Wirewound Inductor ±0.2nH	Murata
L4	LQW15AN4N7C00D	0402	4.7nH	1	Wirewound Inductor ±0.2nH	Murata
L5	LQW15AN9N1H00D	0402	9.1nH	1	Wirewound Inductor ±3%	Murata
L6	LQW15AN15NH00D	0402	15nH	1	Wirewound Inductor ±3%	Murata
L7	LQM21DN150M70L	0805	15uH	1	Fixed Inductors 15uH 250mA Shielded ±20%	Murata
Crystal & TCXO						
X2	SMD02016/4 32.000 MHZ	2.05x1.65mm	32.000MHz	1	SMD02016/4 32.000 MHZ 10/15/-40+85/10PF/40R	NDKPETERMANN-TECHNIK
X1	ECS-120-18-33-JEM-TR3	3.2x2.5mm	12 MHz	1	Crystal 12MHz,CL 18,TOL ±20 ppm ,STAB ±50 ppm,-20 +70 C,ESR 100	ECS
X3	ABS09-32.768KHZ-7-T	4.1x1.5mm	32.768 kHz	1	Crystal 32.768 KHZ 7PF SMD 7pF 20 PPM 70KOHms 1uW	ABRACON
Connectors and Pins						
J6	U.FL-R-SMT-1(10)	U.FL	50 Ohms	1	RF / Coaxial Connector SMT ML REC AU 50 OHM W/ANTI SLDR WCKNG R	Hirose Electric
J5	MS156C320	submin coax female	PROBE	1	RF / Coaxial Connectors SUBMIN COAXIAL RECEP 1.35 MM HT	Hirose Electric
J1	SLW-110-01-T-S	Header 1x10	10 pins	1	10 Pins - Single Row Socket Strip	Samtec
J2	SLW-110-01-T-S	Header 1x10	10 pins	1	10 Pins - Single Row Socket Strip	Samtec
J3	SLW-110-01-T-S	Header 1x10	10 pins	1	10 Pins - Single Row Socket Strip	Samtec
J4	SLW-110-01-T-S	Header 1x10	10 pins	1	10 Pins - Single Row Socket Strip	Samtec
Diodes						
D1	MBRM120ET3G	PowerMITE	1A 20V	1	Schottky Diodes & Rectifiers 1A 20V Low Leakage	onsemi
D2	MBRM120ET3G	PowerMITE	1A 20V	1	Schottky Diodes & Rectifiers 1A 20V Low Leakage	onsemi

C. Budget and Planning

C.1 Budget

Although this project is part of a larger project, in this budget only the costs related with COMMS subsystem will be considered, with the exception of OBC components, that are in the same PCB as COMMS, and they have been used in this work. It is also important to mention that enough components and PCBs were purchased to make 10 satellites (three for IEEE, three to be deployed in orbit and extra components for testing).

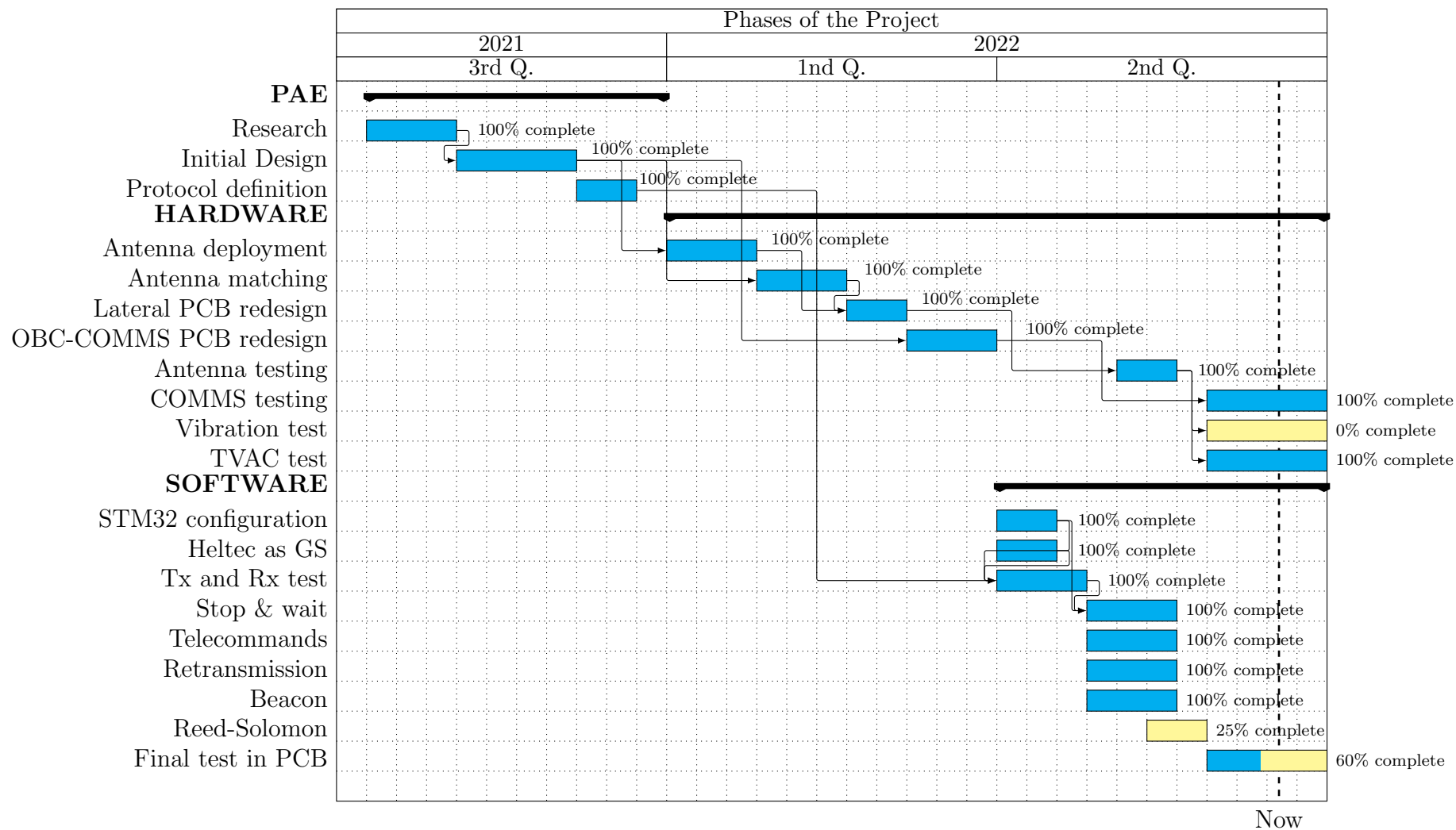
In the following table you can see a breakdown of all the costs that the project has had, which can be classified in material costs (components and PCBs), rent of the facilities that have been used, electricity and internet bills, amortization of laboratory equipment, software licenses, and finally, the salary.

BUDGET TABLE

Component	Value	Quantity	Unit price (€)	Total (€)
06033D105KAT2A	1uF	30	0.091	2.73
06035A100GAT2A	10pF	30	0.273	8.19
0603YD474KAT2A	470nF	10	0.084	0.84
0603ZD102KAT2A	1nF	20	0.084	1.68
ABS09-32.768KHZ-7-T	32.768 kHz	10	1.03	10.3
AC0603JRNPO9BN200	20pF	20	0.087	1.74
ASC0402-100RFT10	100R	20	0.08	1.6
BD2232G-GTR	BD2232G	30	0.732	21.96
BGS12WN6E6327XTSA1	BGS12	10	0.413	4.13
C0603X103F5JACTU	100nF	80	0.566	45.28
CRCW020110K0FNED	10k	10	0.062	0.62
CRCW02014K70FKED	4.7k	20	0.228	4.56
CRCW0402100RFKED	100R	10	0.04	0.4
ECS-120-18-33-JEM-TR3	12 MHz	10	0.333	3.33
GRM1555C1H1R8BA01D	1.8pF	10	0.028	0.28
GRM1555C1H2R2BA01J	2.2pF	10	0.061	0.61
GRM1555C1H2R4WA01D	2.4pF	10	0.076	0.76
GRM1555C1H390JA01D	39pF	20	0.018	0.36
GRM1555C1H3R3BA01D	3.3pF	30	0.052	1.56
GRM1555C1H470FA01D	47pF	10	0.064	0.64
GRM1555C1H5R6CA01D	5.6pF	10	0.039	0.39
GRM155R71C473KA01J	47nF	10	0.054	0.54
GRM185R61C475KE11J	4.7uF	10	0.277	2.77
LQM21DN150M70L	15uH	10	0.201	2.01
LQW15AN15NH00D	15nH	10	0.112	1.12
LQW15AN2N5C00D	2.5nH	10	0.135	1.35
LQW15AN47NJ00D	47nH	10	0.148	1.48
LQW15AN4N7C00D	4.7nH	10	0.114	1.14
LQW15AN9N1H00D	9.1nH	10	0.156	1.56
MBRM120ET3G	1A 20V	20	0.627	12.54
MS156C320	PROBE	10	0.411	4.11
SLW-110-01-T-S	10 pins	40	1.13	45.2
SMD02016/4 32.000 MHZ	32.000MHz	10	0.75	7.5
STM32L162RET6	STM32L1	10	10.05	100.5
SX1262IMLRT	SX1262	10	7.51	75.1
UFL2LPHF6068	RF cable	10	4.54	45.4
U.FL-R-SMT-1(10)	50 Ohms	20	1.01	20.2
Measuring tape	Antenna	10	0.74	7.4
Screw for the antenna	M2	10	0.55	5.5
Nut for the antenna	M2	10	0.08	0.8
Dyneema wire	Roll	1	4.99	4.99
			TOTAL COMPONENTS COST:	453.17
Facility	Location	Days	Price/day (€)	Total (€)
Nanosatlab TVAC	UPC	1	1200	1200
Nanosatlab Internet	UPC	5 month	90 €/month	450
Nanosatlab light	UPC	5 month	130 €/month	650
Anechoic chamber	UPC	1	1200	1200
			TOTAL FACILITIES COST:	3500

PCBs	Company	Quantity	Unit price (€)	Total (€)
OBC-COMMS design 1	JLCPCB	10	0.8	8
Lateral PCB design 1	JLCPCB	10	5.96	59.6
OBC-COMMS final design	JLCPCB	10	0.8	8
Lateral PCB final design	JLCPCB	10	5.96	59.6
TOTAL PCBs COST:				135.2
Equipment (5 month amortization)	Unit price (€)	Quantity	Lifetime (years)	Total (€)
Electronic microscope	150	1	10	6.25
Soldering Station	300	1	15	8.33
Matlab year license	860	1	1	358.33
Vector Network Analyser	15000	1	15	416.67
Spectrum Analyser	10000	1	15	277.78
Signal Generator	1600	1	20	33.33
3D printer	250	1	8	13.02
Power source	200	1	20	4.17
TOTAL AMORTIZATION:				1117.88
Salary	Nº weeks	Hours/week	€/hour	Total (€)
Telecommunication engineer	20	35	9	6300
TOTAL SALARIES:				6300.00
TOTAL COST				11506.25

C.2 GANTT Diagram



D. Link Budget Details

D.1 Montsec Ground Segment

The Ground Segment is located at Montsec mountain (figure D.1), in a rural noise environment, and therefore with very low noise. Currently, there is no antenna installed to work at Europe ISM LoRa frequency (868 MHz). For this reason, for the link budget calculations have been used the properties of the UHF X-Quad antenna (figure D.2). The maximum pointing error of 10° includes both TLE and antenna pointing errors.



Figure D.1: Montsec Ground Segment. Image credit: Kike Herrero (IEEC)



Figure D.2: Montsec UHF antenna. Image from [35]

Despite that this Ground Station is located in the mountain and has visibility at 0° of elevation, the value of minimum elevation has been set to 5° to be conservative and consider the worst case. The same happens with the antenna temperature, where 750 K has been considered.

D.2 PocketQube

All the values of the PocketQube features have been obtained from tests done or from the transceiver datasheet. The transceiver is capable of delivering up to $+22 \text{ dBm}$ under battery supply of 3.3 V. As no configuration to change the polarization have been designed, the monopole antenna will radiate with linear polarization.

To determine the satellite antenna noise, it has been taken into account two factors. The first one is the noise temperature of the Earth, which is about 300 K. However, as the satellite sees free space in the Earth background, and the dark space noise temperature goes from 30 to 150 K, the approximation of 290 K is generally used [36].

From the radiation pattern obtained, the maximum gain of the antenna is 1.15 dB. If the gain with an error of 10° from the maximum is computed, it can be considered an upper limit of 1 dB losses, as seen in figure D.3. The -3 dB beamwidth can also be obtained from the radiation pattern.

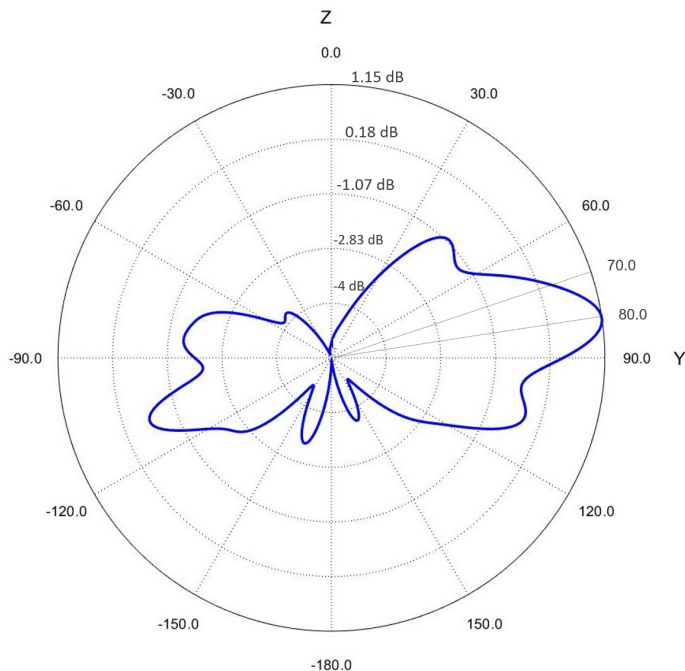


Figure D.3: Radiation pattern cut in the maximum direction

The matching losses of the antenna are negligible (as computed in section 3.1.4, the return losses of the antenna are -26 dB). And the RF losses, as explained in section 3.2.3.1, are 5 dB in a critical case. This value, however, will be revised and improved before the launching of the PocketQube. Nevertheless, a reliable link budget should always consider the worst situation. Figures D.4, D.5 and D.6 show some images of the test done.

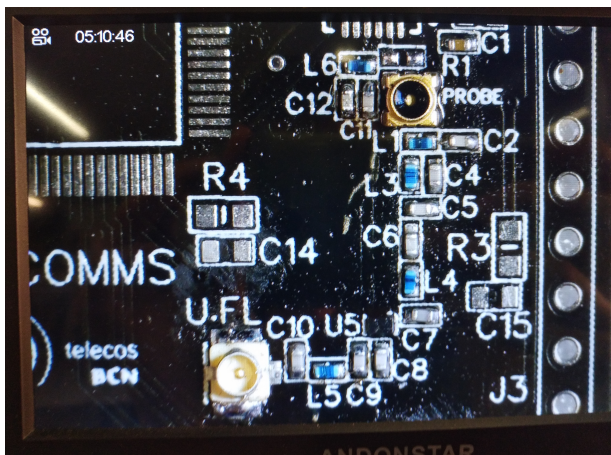


Figure D.4: RF circuit seen through an electronic microscope

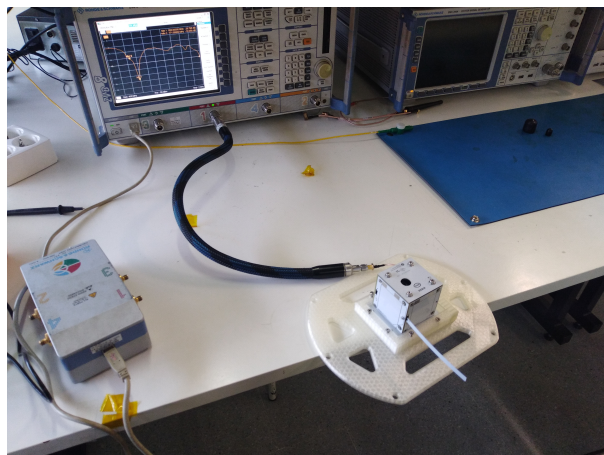


Figure D.5: Antenna matching measurement set up

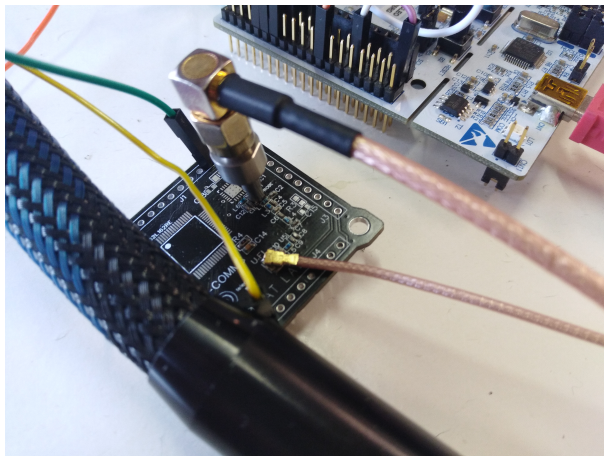


Figure D.6: Measurement of RF losses

Finally, the values of the maximum frequency offset supported by the transceiver, and the sensitivity, can be obtained directly from the datasheet [23].

D.3 Channel Impairments

At the frequency used (868 MHz), most of the atmosphere effect are negligible, for example, the rain attenuation or the gaseous absorption, as can be seen in figures D.7 and D.8. The troposphere scintillation is mainly relevant from 3 GHz [37], so, it has not been considered.

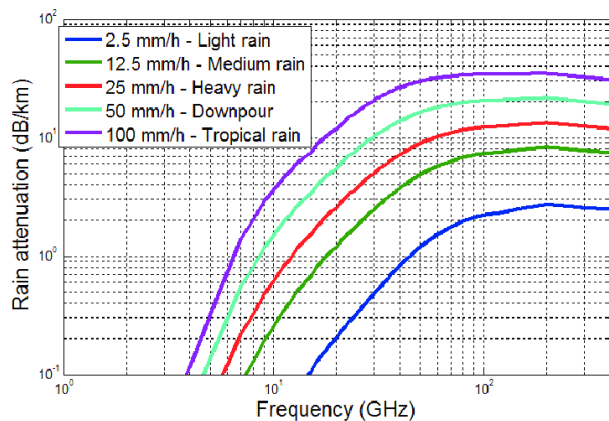


Figure D.7: Rain attenuation as a function of frequency. Image from [38]

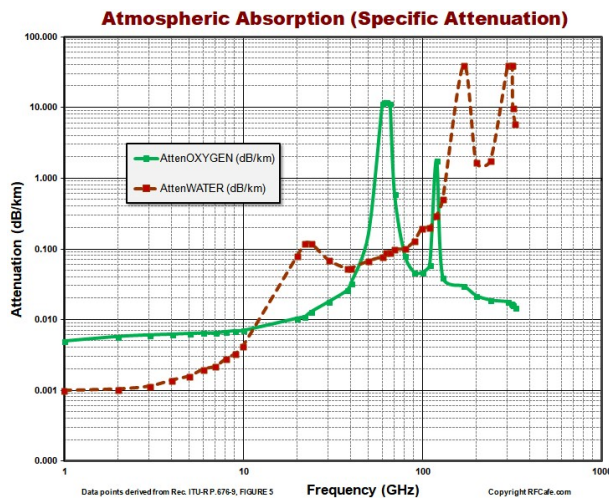


Figure D.8: Gaseous absorption attenuation as a function of frequency. Image from [39]

Finally, apart from the free-space path losses and the polarization losses already mentioned, the only impairment which is significant at 868 MHz is the Ionosphere scintillation. This losses can be defined as fluctuations of the signal due to changes in the ionosphere refraction index, caused by irregular electron concentrations [37]. Despite that in the

location of the Ground Station ionospheric scintillation is not so frequent, as seen in figure D.9, it has to be considered, because it happens although with less frequency. The scintillation effect can produce fadings higher than 15 dB, but they typically recover in 0.4 s, which will affect only one package at most [40]. However, this amount of losses will be critical no matter the SF used in the transceiver, and the packet transmitted will get lost. For this reason, it has no sense considering such high losses in the link budget, and more usual values have been considered. In figure D.10, a table with the duration and time for fadings below 4 dB is presented.

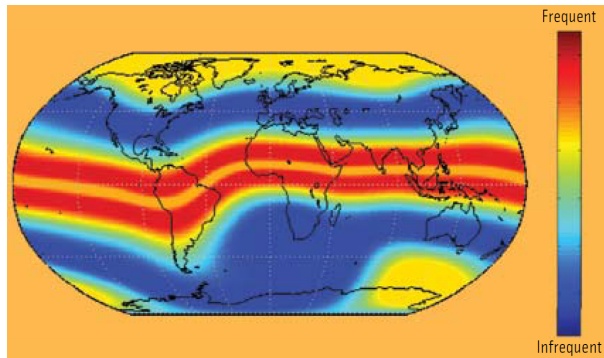


Figure D.9: Frequency of ionospheric disturbances in Earth. Image from [41]

S_4	μ_t (ms)	σ_t (ms)	μ_b (ms)	σ_b (ms)
0.1	64.7	55.5	312.4	198.6
0.2	32.2	29.6	153.6	100.3
0.3	21.1	19.9	99.2	67.9
0.4	15.8	15.5	70.6	51.2
0.5	12.9	15.7	54.1	42.1
0.6	11.0	18.1	43.4	34.8
0.7	10.0	22.4	35.9	29.6
0.8	9.2	25.6	30.1	25.3
0.9	8.4	25.7	25.3	21.8

μ_t : mean duration of fadings
 σ_t : standard deviation of fadings
 μ_b : mean time between fadings
 σ_b : standard deviation of time between fadings

Figure D.10: Duration and time between fadings below 4 dB for the S_4 cases. Image from [40]

High values of S_4 have a short duration and is less probable that they will affect a whole packet. However, lower values of S_4 , despite having longer times between fadings, if they coincide with a transmission, they will affect the link budget. A study done in [42], shows that the peaks of low S_4 values (which are the ones expected in Montsec Ground Station) are usually between ± 1 dB. For this reason, the mean value of ionosphere scintillation losses of 1 dB has been considered in the link budget.

E. SX1262 Internal State Machine

The SX1262 transceiver has internal operating modes and states, as presented in figure E.1.

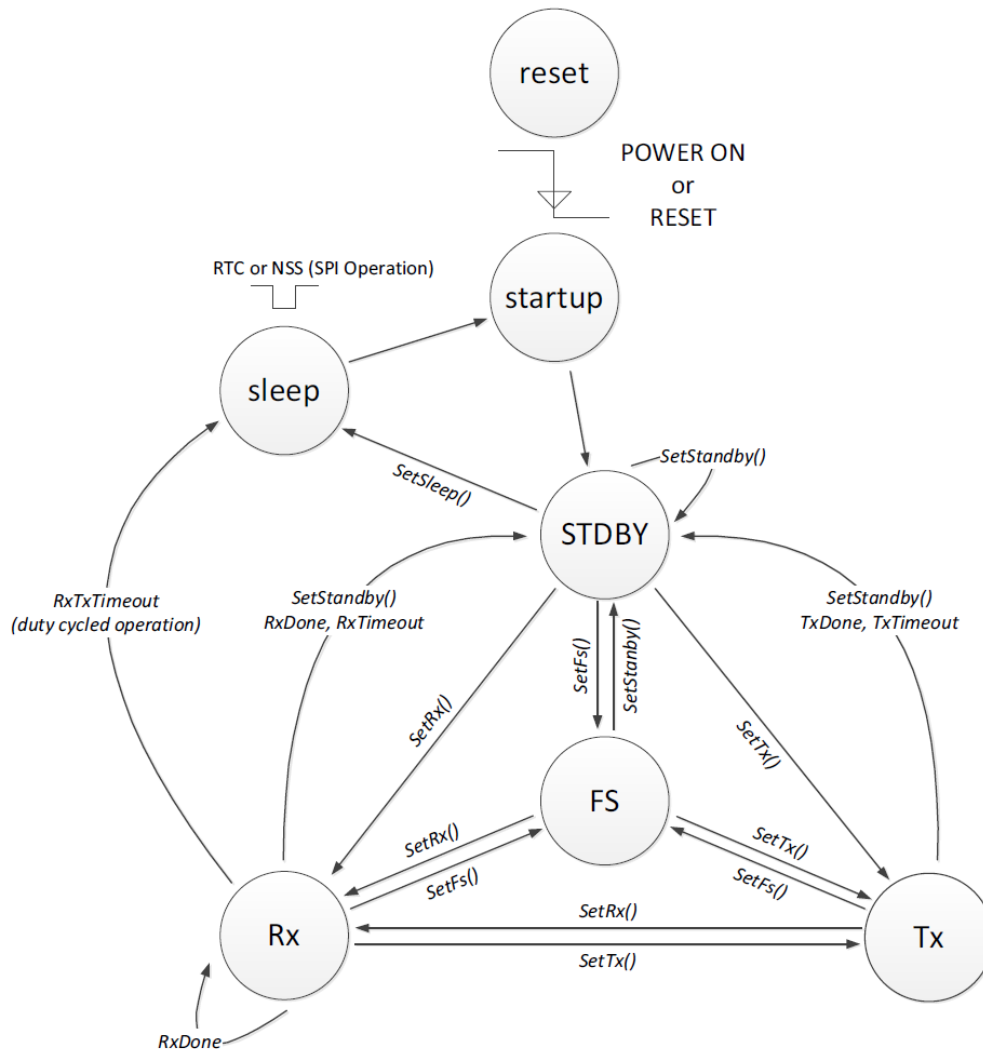


Figure E.1: Transceiver circuit modes. Image from [23]

The main functionalities of each mode are the following:

- *Startup*: the control of the chip is done by the sleep state machine until the digital voltage and RC clock become available. Then, the chip can boot up, and the CPU takes control.
- *Calibrate*: it is called in case of power On or Reset (or with calibration command). It calibrates the internal oscillators with the crystal reference, the PLL and the reception pins.
- *Sleep*: In this mode, most of the radio internal blocks are powered down or in low power mode.

- *STDBY*: In standby mode the host should configure the chip before going to RX or TX modes. It uses the RC13M clock, which has less power consumption.
- *FS*: In Frequency Synthesis mode, PLL and related regulators are switched ON. The BUSY goes low as soon as the PLL is locked or timed out. This mode is used to change between the Tx and Rx frequency.
- *Rx*: receive mode can be configured depending on how we want to operate. It can remain in Rx until another mode is indicated, it can return to STDBY after packet reception or timeout, or it can alternate between Sleep and Rx until there is an IRQ.
- *Tx*: the transceiver ramps-up the power amplifier and transmits the data buffer. The timeout can be used to avoid the transceiver getting stuck if any error occurs.

F. SX1262 library

The library used to manage the transceiver contains the following files:

- `board.c` and `board.h`: these files contain the pin definitions specified for the used stm32 board. In the project they have been modified and adapted to the OBC-COMMS PCB design. These files are also the ones which handle some general function related to stm32, such as calibrations, clocks, IRQs...
- `delay.c` and `delay.h`: they contain delay functions (to delay some seconds or milliseconds a process).
- `fifo.c` and `fifo.h`: they contain the functions to implement a FIFO buffer (First Input, First Output). These functions are used by the radio, uart and board files.
- `gpio.c` and `gpio.h`: generic GPIO driver functions to establish GPIO pin communications between the sx126x and the stm32.
- `gpio-board.c` and `gpio-board.h`: these files adapt the generic gpio files to the specific stm32 board and MCU configuration. The functions included work as an adapter between the GPIO functions and the stm32 functions.
- `pinName-board.h` and `pinName-ioe.h`: these files contain the microcontroller pin definitions.
- `radio.c` and `radio.h`: these files are the ones in charge of driving all the radio operations of the modem, such as the configuration, sending, receiving, computing some parameters related to the radiochannel, manage the Buffer, doing the CAD (Channel Activity Detection), receive the IRQ coming from the transceiver, etc. These files work as an API between our code and the sx126x specific driver files.
- `rtc-board.c` and `rtc-board.h`: these files are the ones who manage the MCU RTC timer and low power modes.
- `serial.h`: definitions of the generic UART driver functions.
- `spi.h`: definitions of the generic SPI driver functions.
- `spi-board.c` and `spi-board.h`: implement the SPI functions for the particular stm32 used. These files make possible the SPI communications between the transceiver and the SPI pins of the stm32.
- `sx126x.c` and `sx126x.h`: these files are the ones that adapt all the radio functions to the particular case of the Semtech SX126x transceivers family. These files are specially programmed for these kinds of transceivers and will not work for any other. It is important to say that all the functions to work with FSK modulation have been removed, and therefore, these files only allow to work with LoRa.
- `sx126x-board.c` and `sx126x-board.h`: these files adapt the sx126x files to the specific stm32 microcontroller, with the proper pin definitions.

- `timer.c` *and* `timer.h`: these files implement the functions related with the timer objects and scheduling management. They are directly related with the `rtc-board` files.
- `uart.c` *and* `uart.h`: these files are the ones that handle the communications of UART configured pins. Despite that UART communications between the SX1262 and the `stm32` pins are not used in the current configuration, these files have been left in case there is a configuration change in the future.
- `uart-board.c` *and* `uart-board.h`: these are used to adapt the `uart` communication files to the `stm32` used.
- `utilities.c` *and* `utilities.h`: these files provide some useful functions such as generating random numbers, coping bytes between non-aligned arrays, conversions between nibbles and hexadecimal...

G. Protocol Tests and Code

To test the protocol and the programmed code, three different devices has been used:

- *Nucleo-L476RG board*: this is a STM32 board with a microprocessor very similar to the one used in OBC-COMMS PCB (pin-compatible, same configuration,...). This board has been used to simulate the whole OBC circuit, with the advantage that it is easier to compile the code and flash the memory than in the OBC-COMMS PCB. Also this board has been chosen because the same code programmed for the PocketQube can be run and debugged without any change, due to its compatibility. See in figure G.1.
- *SX1262MB2CAS*: this is the Semtech SX1262 transceiver evaluation board. Therefore, as it uses the same transceiver, and a circuit very similar to the COMMS PCB, it is the perfect board to test the code and configurations as if it were the COMMS circuit. To do the tests, it has been connected with the Nucleo-L476RG in the same way as in OBC-COMMS PCB. See in figure G.2.
- *Cube Cell HTCC-AB01*: this is a development board which also has a microcontroller unit integrated and a SX1262 transceiver. It also has the advantage of being Arduino compatible, which makes it easier to program. However, its MCU is not the same as the one in the PocketQube, but it can be used to work as if it were the Ground Station. See in figure G.3.

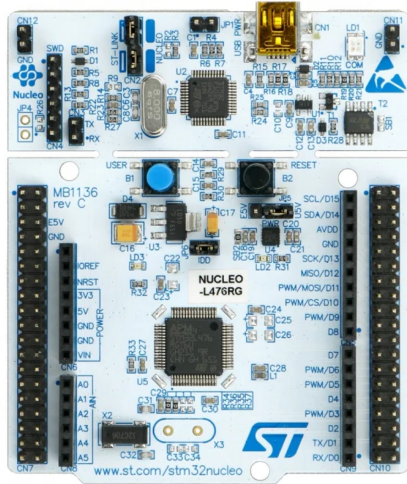


Figure G.1: Nucleo-L476RG board. Image from [43]

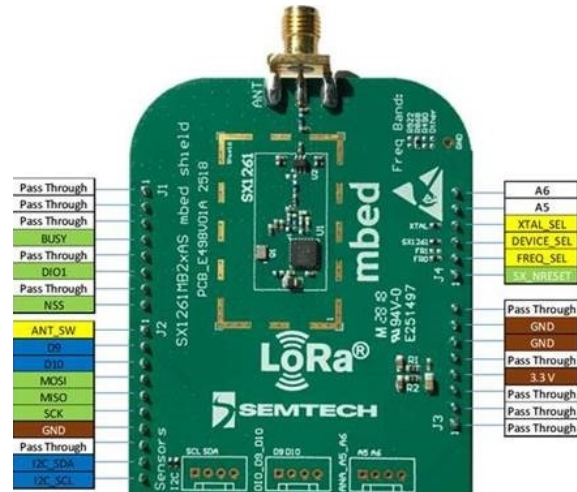


Figure G.2: SX126xMB2xAS board. Image from [44]



Figure G.3: Heltec HTCC-AB01 Dev-Board. Image from [45]

G.1 Tests

All the code programmed, have to be upload and compiled in the Nucleo board, which is also connected to the SX1262 board. However, in order to make the STM32 work correctly, the OBC have to be properly configured and some of its functions programmed, such as the one to read or write to the flash memory, functions to start, stop and handle the timers IRQ, the initialization of all the memory addresses...⁶

G.1.1 STM32 configuration

Before introducing any code in the Nucleo board, the STM32 pins have to be configured depending on its functionality. In figure G.4 is shown the pin type of each of the STM32 pins used by the PocketQube. Apart from declaring the type, some additional parameters have to be added, and it is necessary to indicate which ones require IRQ handling.

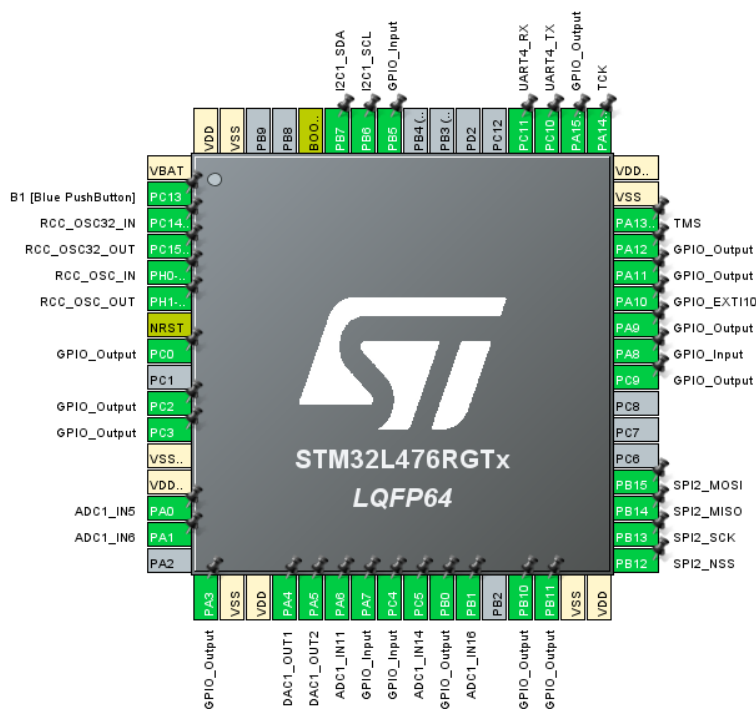


Figure G.4: STM32 pin type declaration

Also, the COMMS protocol uses two timers: one for the beacon (60 seconds timer), and another for the reception timeout in the protocol (500 ms for SF=7). To obtain an IRQ at the desired time, the CPU frequency (80 MHz) is used as reference. Then, a prescaler factor have to be applied to reduce its frequency. After that, a counting period have to be established, which basically means the number of periods that have to pass at the previous frequency before the timeout occurs. Finally, the IRQs have to be enabled. In figure G.5 is shown how the time is computed for the timer of 500 ms.

⁶This functions, despite having been programmed by the author of this work, are out of the scope of this degree thesis and therefore they will not be explained in this document.

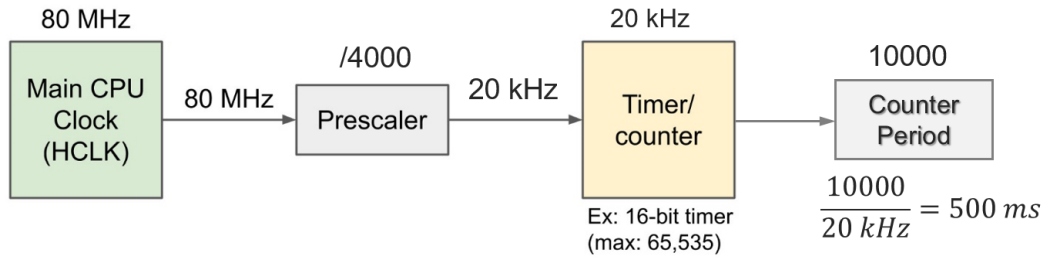


Figure G.5: Protocol timeout timer functioning

For the beacon timer, as 60 seconds is a very long time, a 30 seconds timer has been configured and in the code has been programmed that the beacon is transmitted when the timer ends two times. It has been used a prescaler factor of 48000 and a counter period of 50000.

$$f_{counting} = \frac{f_{MCU}}{prescaler} = \frac{80 \text{ MHz}}{48000} = 1.667 \text{ kHz} \quad (\text{G.1})$$

$$time = \frac{N_{counts}}{f_{counting}} = \frac{50000}{1.667 \text{ kHz}} = 30 \text{ s} \quad (\text{G.2})$$

G.1.2 Transmission and reception

After configuring the OBC, the following steps were followed in order to test the code progressively:

1. Library example test and error corrections. Library adaptation to the used board. Deletion of some unused functions.
2. TX and RX test using the Nucleo board + SX1262 board as if it were the PocketQube, and the Heltec module as Ground Station. As the Heltec module is Arduino compatible, the packets transmitted and received can be seen in the serial monitor.
3. Optimum time between packets to receive at least the 98% of the packets found empirically. For the SF 7, the minimum time was 500 ms. Higher SF need more processing time.
4. Introduction of the functions to write in flash memory and read from it, and test the telecommands one by one to see if they work as expected. To check if the memory operations had been done correctly, the STM32 ST-LINK Utility software⁷ was used.
5. Test of the complete protocol, with telecommand execution request and confirmation. Test adding the beacon. Test the transmission when send data telecommand is sent, and with NACK to acknowledge the 20 packets of the window or request retransmissions.

⁷<https://www.st.com/en/development-tools/stsw-link004.html>

6. Test of the code in OBC-COMMS PCB. This test could not be done because the OBC part of the PCB needed some previous hardware verification that the OBC team had not yet done at the time of writing of the present document.

In figure G.6 is shown a screenshot of the Arduino serial monitor of the Heltec module (which simulates the GS), and a description of the packets transmitted and received. In figure G.7 can be seen the memory addresses of the STM32 where the a photography from the payload 1 is stored. As it can be noticed, the packets received in figure G.6 correspond with the data stored in G.7.

```

Telecommand TX: C8 9D 14 0
Telecommand TX: C8 9D 14 0 Send data telecommand (PIN1, PIN2, telecommand ID, parameter)

Received paquet number 0 Number of the packet received by the GS
0x86, 0x64, 0x0, Mission ID, satellite ID, packet number Data received
0xFF, 0xD8, 0xFF, 0xE0, 0x0, 0x10, 0x4A, 0x46, 0x49, 0x46, 0x0, 0x1, 0x1, 0x0, 0x0, 0x1,
0x0, 0x1, 0x0, 0x0, 0xFF, 0xE2, 0x2, 0x28, 0x49, 0x43, 0x43, 0x5F, 0x50, 0x52, 0x4F, 0x46,
0x49, 0x4C, 0x45, 0x0, 0x1, 0x1, 0x0, 0x0, 0x2, 0x18, 0x0, 0x0, 0x0, 0x0, 0x4, 0x30,
0x0, 0x0, 0x6D, 0x6E, 0x74, 0x72, 0x52, 0x47, 0x42, 0x20, 0x58, 0x59, 0x5A, 0x20, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x61, 0x63, 0x73, 0x70, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0xFF, Final of the packet indicator
Received paquet number 1 2nd packet received
0x86, 0x64, 0x1,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x1, 0x0, 0x0, 0xFF6, 0xD6, 0x0, 0x1,
0x0, 0x0, 0x0, 0x0, 0xD3, 0x2D, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0, 0x9, 0x64, 0x65, 0x73, 0x63, 0x0, 0x0,
0x0, 0xFF0, 0x0, 0x0, 0x0, 0x74, 0x72, 0x58, 0x59, 0x5A, 0x0, 0x0, 0x1, 0x64, 0x0, 0x0,
0xFF,
Received paquet number 2 3rd packet received
0x86, 0x64, 0x2,
0x0, 0x14, 0x67, 0x58, 0x59, 0x5A, 0x0, 0x0, 0x1, 0x78, 0x0, 0x0, 0x0, 0x14, 0x62, 0x58,
0x59, 0x5A, 0x0, 0x0, 0x1, 0x8C, 0x0, 0x0, 0x0, 0x14, 0x72, 0x54, 0x52, 0x43, 0x0, 0x0,
0x1, 0xA0, 0x0, 0x0, 0x0, 0x28, 0x67, 0x54, 0x52, 0x43, 0x0, 0x0, 0x1, 0xA0, 0x0, 0x0,
0x0, 0x28, 0x62, 0x54, 0x52, 0x43, 0x0, 0x0, 0x1, 0xA0, 0x0, 0x0, 0x0, 0x28, 0x77, 0x74,
0x70, 0x74, 0x0, 0x0, 0x1, 0xC8, 0x0, 0x0, 0x0, 0x14, 0x63, 0x70, 0x72, 0x74, 0x0, 0x0,
0x1, 0xDC, 0x0, 0x0, 0x0, 0x3C, 0x6D, 0x6C, 0x75, 0x63, 0x0, 0x0, 0x0, 0x0, 0x0, 0x0,
0xFF,
    
```

Figure G.6: Heltec module serial monitor

Memory display

Address: 0x08010000 Size: 0x1000 Data Width: 32 bits

Device Memory @ 0x08010000 : Binary File

Target memory, Address range: [0x08010000 0x08011000]

Address	0	4	8	C	ASCII
0x08010000	E0FFD8FF	464A1000	01004649	01000001	ÿøÿà..JFIF.....
0x08010010	00000100	2802E2FF	5F434349	464F5250ÿà.(ICC_PROF
0x08010020	00454C49	00000101	00001802	30040000	ILE.....0
0x08010030	6E6D0000	47527274	59582042	0000205A	..mnrRGB XYZ ..
0x08010040	00000000	00000000	63610000	00007073acsp..
0x08010050	00000000	00000000	00000000	00000000
0x08010060	00000000	00000000	00000100	0100D6F6öÖ..
0x08010070	00000000	00002DD3	00000000	00000000Ó-.....
0x08010080	00000000	00000000	00000000	00000000
0x08010090	00000000	00000000	00000000	00000000
0x080100A0	00000000	00000000	65640900	00006373desc..
0x080100B0	0000F000	58727400	00005A59	00006401	.ö...trXYZ...d..
0x080100C0	58671400	00005A59	00007801	58621400	..gXYZ...x...bX
0x080100D0	00005A59	00008C01	54721400	00004352	YZ...Æ....rTRC..
0x080100E0	0000A001	54672800	00004352	0000A001	... (gTRC... ..
0x080100F0	54622800	00004352	0000A001	74772800	.(bTRC... ...(wt
0x08010100	00007470	0000C801	70631400	00007472	pt...Ë....cprt..
0x08010110	0000DC01	6C6D3C00	00006375	00000000	.Û...<mluc.....
0x08010120	00000100	6E650C00	00005355	00005800enUS...X..
0x08010130	73001C00	47005200	00004200	00000000	...s.R.G.B.....

Figure G.7: Flash memory of the STM32

G.2 Code

Finally, in this section is presented the code programmed. The most updated version is in the Github of the author of this final degree thesis⁸. The `Comms.c` file contains the COMMS functions, the state machine and the protocol. The `Comms.h` script is the header file for `Comms.c`. The `GroundStation.ino` is the code programmed for the Heltec module. This code is not a real Ground Station code but a script to test the telecommands and force different situations in the satellite code. Some commented lines are to test different cases.

⁸https://github.com/daniel-herencia/Nucleo-L476RG-COMMS_TEST

G.2.1 Comms.c

```

1 /*
2  * comms.c
3  *
4  * Created on: 23 feb. 2022
5  * Author: Daniel Herencia Ruiz
6  * \code
7  *
8  *
9  * /_____ ) /_____ \ |_____| /_____| |_____| /_____| |_____|
10 * //_____| |_____| | | | | | | \ \ \ / | | | \ \ \ / | | | ( (_____|
11 * ( ( | | | | | | \ \ / / | | | \ \ / / | | | \_____|
12 * \ \_____| |_____| |_____| |_____| |_____| |_____| |_____| )
13 * \_____| \_____| / |_____| |_____| |_____| |_____| |_____| /
14 *
15 *
16 * \endcode
17 */
18
19
20 #include "comms.h"
21
22
23 /***** STATES *****/
24
25 typedef enum //Possible States of the State Machine
26 {
27     LOWPOWER,
28     RX,
29     RX_TIMEOUT,
30     RX_ERROR,
31     TX,
32     TX_TIMEOUT,
33     START_CAD,
34 }States_t;
35
36 typedef enum //CAD states
37 {
38     CAD_FAIL,
39     CAD_SUCCESS,
40     PENDING,
41 }CadRx_t;
42
43 States_t State = LOWPOWER; //Current state of the State Machine
44
45
46 /***** PACKETS *****/
47
48 uint8_t Buffer[BUFFER_SIZE]; //Tx and Rx Buffer
49 uint8_t nack[WINDOW_SIZE]; //To store the last ack/nack received
50 uint8_t last_telecommand[BUFFER_SIZE]; //Last telecommand RX
51
52
53 /***** FLAGS *****/
54
55 uint8_t error_telecommand = false; //To transmit an error packet

```

```

56 uint8_t tx_flag = false;           //To allow transmission
57 uint8_t send_data = false;        //To send data packets to the GS
58 uint8_t beacon_flag = false;      //To transmit the beacon
59 uint8_t nack_flag = false;        //Retransmission necessary
60 uint8_t tle_telecommand = false;   //True when TLE telecommand received
61 uint8_t telecommand_rx = false;    //To indicate that a telecommand has
    been received
62 uint8_t request_execution = false; //To send the request execution packet
63 uint8_t protocol_timeout = false; //True when the protocol timer ends
64 uint8_t reception_ack_mode = false; //True
65 uint8_t contingency = false;       //True if we are in contingency state (
    only RX allowed)
66
67
68 /***** COUNTERS *****/
69
70 uint8_t request_counter = 0;        //Number of request execution packets
    sent (to execute a telecommand order)
71 uint8_t packet_number = 0;         //Data packet number
72 uint8_t num_config = 0;            //Configuration packet number
73 uint8_t num_telemetry = 0;        //Telemetry packet number
74 uint8_t window_packet = 0;        //TX window number
75 uint8_t nack_counter;             //Position of the NACK array (packets
    already retransmitted)
76 uint8_t k=0;                      //Counter for loops
77 uint8_t protocol_timeout_counter = 0; //Number of times that the protocol
    timer (500 ms) has ended
78                                     //This is used to have longer timers
    for higher SF
79
80
81 /***** LoRa PARAMETERS *****/
82 uint8_t SF = LORA_SPREADING_FACTOR; //Spreading Factor
83 uint8_t CR = LORA_CODINGRATE;      //Coding Rate
84 uint16_t time_packets = 500;       //Time between data packets sent in ms
85
86
87 /***** OTHER *****/
88
89 int8_t RssiValue = 0;
90 int8_t SnrValue = 0;
91 CadRx_t CadRx = CAD_FAIL;
92 bool PacketReceived = false;
93 bool RxTimeoutTimerIrqFlag = false;
94 uint16_t channelActivityDetectedCnt = 0;
95 uint16_t RxCorrectCnt = 0;
96 uint16_t RxErrorCnt = 0;
97 uint16_t RxTimeoutCnt = 0;
98 uint16_t SymbTimeoutCnt = 0;
99 int16_t RssiMoy = 0;
100 int8_t SnrMoy = 0;
101
102
103
104 /*****
105 *
106 * Function: configuration
107 * -----

```

```

108 * function to configure the transceiver and the protocol parameters *
109 * *
110 * returns: nothing *
111 * *
112 *****/
113 void configuration(void) {
114
115     uint64_t read_variable; //Read flash function requires variables of 64
        bits
116
117
118     /* Reads the SF, CR and time between packets variables from memory */
119     Read_Flash(SF_ADDR, &read_variable, 1);
120     memcpy(SF, read_variable, sizeof(SF));
121     Read_Flash(CRC_ADDR, &read_variable, 1);
122     memcpy(CR, read_variable, sizeof(CR));
123     Read_Flash(COMMS_TIME_ADDR, &read_variable, 1);
124     memcpy(time_packets, read_variable, sizeof(time_packets));
125
126
127     /* Configuration of the LoRa frequency and TX and RX parameters */
128     Radio.SetChannel( RF_FREQUENCY );
129
130     Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH, SF,
        CR,
131
        LORA_PREAMBLE_LENGTH,
132     LORA_FIX_LENGTH_PAYLOAD_ON,
        true, 0, 0, LORA_IQ_INVERSION_ON, 3000 )
        ;
133
134     Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, SF, CR, 0,
        LORA_PREAMBLE_LENGTH,
135
        LORA_SYMBOL_TIMEOUT,
136     LORA_FIX_LENGTH_PAYLOAD_ON,
        0, true, 0, 0, LORA_IQ_INVERSION_ON,
        true );
137
138     /* Configuration of the CAD parameters */
139     SX126xConfigureCad( CAD_SYMBOL_NUM, CAD_DET_PEAK, CAD_DET_MIN,
        CAD_TIMEOUT_MS);
140     Radio.StartCad( ); //To initialize the CAD process
141
142     State = RX; //To initialize in RX state
143 };
144
145
146 *****/
147 * *
148 * Function: stateMachine *
149 * ----- *
150 * communication process state machine *
151 * States: *
152 * - RX_TIMEOUT: when the reception ends *
153 * - RX_ERROR: when an error in the reception process occurs *
154 * - RX: when a packet has been received or to start a RX process *
155 * - TX: to transmit a packet *
156 * - TX_TIMEOUT: when the transmission ends *
157 * - LOWPOWER: when the transceiver is not transmitting nor receiving *

```

```

158 *
159 * returns: nothing
160 *
161 *****/
162 void StateMachine( void )
163 {
164     /* Target board initialization*/
165     BoardInitMcu( );
166     BoardInitPeriph( );
167
168     /* Radio initialization */
169     RadioEvents.TxDone = OnTxDone;
170     RadioEvents.RxDone = OnRxDone;
171     RadioEvents.TxTimeout = OnTxTimeout;
172     RadioEvents.RxTimeout = OnRxTimeout;
173     RadioEvents.RxError = OnRxError;
174     RadioEvents.CadDone = OnCadDone;
175
176     /* Timer used to restart the CAD */
177     TimerInit( &CADTimeoutTimer, CADTimeoutTimeoutIrq );
178     TimerSetValue( &CADTimeoutTimer, CAD_TIMER_TIMEOUT );
179
180     /* App timer used to check the RX's end */
181     TimerInit( &RxAppTimeoutTimer, RxTimeoutTimerIrq );
182     TimerSetValue( &RxAppTimeoutTimer, RX_TIMER_TIMEOUT );
183
184     Radio.Init( &RadioEvents ); //Initializes the Radio
185
186     configuration(); //Configures the transceiver
187
188     while( 1 ) //The only option to end the state
189     machine is killing COMMS thread (by the OBC)
190     {
191         DelayMs( 1 );
192         Radio.IrqProcess( ); //Checks the interruptions
193
194         switch( State )
195         {
196             case RX_TIMEOUT: //Stops the radio and goes to lowpower
197             {
198                 RxTimeoutCnt++;
199                 Radio.Standby;
200                 State = LOWPOWER;
201                 break;
202             }
203             case RX_ERROR: //Stops the radio and goes to lowpower
204             {
205                 RxErrorCnt++;
206                 PacketReceived = false;
207                 Radio.Standby;
208                 State = LOWPOWER;
209                 break;
210             }
211             case RX:
212             {
213                 if( PacketReceived == true )

```



```

214     if (pin_correct(Buffer[0], Buffer[1])){ //Check if the pin is
correct
215         State = LOWPOWER;
216         if (Buffer[2] == TLE){           //If it is a TLE
telecommand
217             Stop_timer_16();           //This is not necessary, put
here for safety
218             if (!tle_telecommand){     //First TLE packet
219                 tle_telecommand = true;
220                 State = RX;           //Goes to RX the next
221                 telecommand_rx = true;
222             }
223             else{                       //Is the last TLE packet (
there are 2)
224                 tle_telecommand = false;
225                 telecommand_rx = false;
226             }
227             process_telecommand(Buffer[2], Buffer[3]); //Saves the TLE
228         }
229         /* Second telecommand RX consecutively */
230         else if (telecommand_rx){
231             /* 2nd telecommand received equal to the 1st */
232             if (Buffer[2] == last_telecommand[2]){
233                 Stop_timer_16();
234                 /* If it is a sending order */
235                 if (Buffer[2] == SEND_DATA || Buffer[2] == SEND_TELEMETRY
|| Buffer[2] == ACK_DATA || Buffer[2] == SEND_CALIBRATION || Buffer[2]
== SEND_CONFIG){
236                     telecommand_rx = false;
237                     process_telecommand(Buffer[2], Buffer[3]); //Process the
telecommand
238                 }
239                 /* If it is not a sending order, request the execution to
GS */
240                 else {
241                     request_execution = true;
242                     State = TX;
243                     DelayMs(300);
244                 }
245             }
246             /* If the GS confirms the order execution, processes the
telecommand */
247             else if(Buffer[2] == ACK){
248                 Stop_timer_16();
249                 request_counter = 0;
250                 request_execution = false;
251                 reception_ack_mode = false;
252                 telecommand_rx = false;
253                 process_telecommand(last_telecommand[2], last_telecommand
[3]);
254                 State = RX;
255             }
256             /* If the 2nd telecommand received is different from the
first, tx error message */
257             else{
258                 State = TX;
259                 telecommand_rx = false;
260                 error_telecommand = true;

```

```

261         Stop_timer_16();
262         DelayMs(10);
263     }
264 }
265 /* First telecommand RX */
266 else{
267     memcpy( last_telecommand, Buffer, BUFFER_SIZE );
268     last_telecommand[0] = MISSION_ID; //To avoid retransmitting
the PIN
269     last_telecommand[1] = POCKETQUBE_ID;
270     tle_telecommand = false;
271     telecommand_rx = true;
272     State = RX;
273     DelayMs(10);
274     Start_timer_16(); // Start timer to see if we rx another
equal to the first telecommand
275 }
276 /* Pin not correct. If the pin is not correct it is assumed that
the packet comes from another source. The protocol continues ignoring it
*/
277 } else{
278     State = TX;
279     error_telecommand = true;
280     Stop_timer_16();
281     DelayMs(500);
282 }
283     PacketReceived = false; // Reset flag
284 }
285 /* If no packet have been received, restarts the reception
process */
286     else
287     {
288         if (CadRx == CAD_SUCCESS)
289         {
290             channelActivityDetectedCnt++; // Update counter
291             RxTimeoutTimerIrqFlag = false;
292             TimerReset(&RxAppTimeoutTimer); // Start the Rx's's
Timer
293         }
294         else
295         {
296             TimerStart(&CADTimeoutTimer); // Start the CAD's
Timer
297         }
298         Radio.Rx( RX_TIMEOUT_VALUE ); //Basic RX code
299         DelayMs(1);
300         State = LOWPOWER;
301
302         if (reception_ack_mode){
303             reception_ack_mode = false;
304             DelayMs( 300 );
305         }
306     }
307     break;
308 }
309 case TX:
310 {
311     State = LOWPOWER;

```

```

312         /* If the error flag is activated, sends the error message
*/
313     if (error_telecommand) {
314         uint8_t packet_to_send[] = {MISSION_ID,POCKETQUBE_ID,ERROR
};
315         Radio.Send(packet_to_send,sizeof(packet_to_send)); //
Packets sent in pairs
316         DelayMs(100);
317         Radio.Send(packet_to_send,sizeof(packet_to_send));
318         error_telecommand = false;
319     }
320     /* Send request for execute telecommand order if the flag
is activated */
321     else if (request_execution){
322         DelayMs(100);
323         Radio.Send(last_telecommand,3);
324         DelayMs(100);
325         Radio.Send(last_telecommand,3);
326         request_counter++;
327         reception_ack_mode = true;
328         State = RX;
329         Stop_timer_16();
330         Start_timer_16();
331     }
332     /* If send data flag is activated */
333     else if (tx_flag){
334         uint64_t read_photo[12];
335         uint8_t transformed[96];
336         /* We transmit until the window is completed */
337         if (window_packet < WINDOW_SIZE){
338             /* If there are retransmissions pending */
339             if (nack_flag){
340                 if (nack[nack_counter] != 0){ //Retransmit nack packets
until the array is finished
341                     Flash_Read_Data(DATA_ADDR + nack[nack_counter]*96, &
read_photo, sizeof(read_photo));
342                     Buffer[2] = nack[nack_counter]; //Number of the
retransmitted packet
343                     nack_counter++;
344                 } else{ //When all packets have been retransmitted, we
continue with the next one in flash memory
345                     nack_flag = false;
346                     nack_counter = 0;
347                     Flash_Read_Data(DATA_ADDR + packet_number*96, &
read_photo, sizeof(read_photo));
348                     Buffer[2] = packet_number; //Number of the packet
packet_number++;
349                 }
350             }
351         }
352         /* If there are no retransmissions pending, read the data
from flash memory */
353         else{
354             Flash_Read_Data(DATA_ADDR + packet_number*96, &
read_photo, sizeof(read_photo));
355             Buffer[2] = packet_number; //Number of the packet
packet_number++;
356         }
357     }
358     Buffer[0] = MISSION_ID;           //Satellite ID

```

```

359         Buffer[1] = POCKETQUBE_ID;           //PocketQube ID (there
are at least 3)
360         memcpy(&transformed, read_photo, sizeof(transformed));
361         for (uint8_t i=3; i<BUFFER_SIZE-1; i++){
362             Buffer[i] = transformed[i-3];
363         }
364         Buffer[BUFFER_SIZE-1] = 0xFF;       //Final of the packet
indicator
365         window_packet++;
366         State = TX; //We return to TX until the window is
completed
367         DelayMs( (uint16_t) time_packets*3/5 ); //Waits 3/5 of
the time between packets
368                                                     //before sending
and 2/5 after sending
369                                                     //(sometimes read
operations require time)
370         Radio.Send( Buffer, BUFFER_SIZE ); //Send the
packet
371     }
372     /* If the window is completed, goes to RX the ack/nack
packet */
373     else{
374         tx_flag = false;
375         send_data = false;
376         window_packet = 0;
377         State = RX;
378     }
379     DelayMs( (uint16_t) time_packets*2/5 ); //Waits 2/5 of
the time, after sending
380     }
381     /* If the beacon flag is activated, TX the beacon packet */
382     else if (beacon_flag){
383         uint8_t packet_to_send[] = {MISSION_ID, POCKETQUBE_ID, BEACON};
384         Radio.Send(packet_to_send, sizeof(packet_to_send));
385         DelayMs(100);
386         Radio.Send(packet_to_send, sizeof(packet_to_send));
387         beacon_flag = false;
388     }
389     DelayMs( 1 );
390     break;
391 }
392 /* Goes to lowpower state */
393 case TX_TIMEOUT:
394 {
395     State = LOWPOWER;
396     break;
397 }
398 /* LOWPOWER OR DEFAULT CASE */
399 case LOWPOWER:
400 default:
401     /* If any flag is activated, redirects to the corresponding
state */
402     if (error_telecommand || tx_flag){
403         State = TX;
404     }
405     else if (reception_ack_mode || tle_telecommand){
406         State = RX;

```

```

407         }
408         else if (telecommand_rx){ //We have received at least one
telecommand
409             if (request_execution ){ //In this case we have to TX
request or wait for ACK
410                 if (request_counter >= 3){ //If 3 request have been sent
, we send an error message
411                     request_execution = false;
412                     error_telecommand = true;
413                     telecommand_rx = false;
414                     State = TX;
415                     request_counter=0;
416                     Stop_timer_16();
417                 } else if (protocol_timeout){ //if less than 3 have been
sent, TX another request execution
418                     protocol_timeout = false;
419                     State = TX;
420                 } else { //If the timeout has not finished, keep
receiving
421                     State = RX;
422                 }
423             } else{ //We want to Rx the second telecommand
424                 if (protocol_timeout){
425                     PacketReceived = true;
426                     protocol_timeout = false;
427                     DelayMs(1);
428                     Stop_timer_16();
429                 }
430                 State = RX; //If Timeout passes and the 2nd telecommand is not
received,
431                                     //goes to RX and will process the first, as if the
second has been RX
432             }
433         }
434         else if (beacon_flag){
435             State = TX;
436         }
437         else{
438             State = RX;
439         }
440         break;
441     }
442
443     TimerLowPowerHandler( );
444 }
445 }
446
447 /*****
448 *
449 * Function: tx_beacon
450 * -----
451 * Activates the flags to TX the beacon
452 *
453 * returns: nothing
454 *
455 *****/
456 void tx_beacon(void) {
457     if (State == RX || State == LOWPOWER){

```

```

458     if (!reception_ack_mode && !tle_telecommand && !telecommand_rx){
459         State = TX;
460         beacon_flag = true;
461     }
462 }
463 }
464
465 /*****
466 *
467 * Function: OnTxDone
468 * -----
469 * Activate the flag to indicate that the RX timeout has finished
470 *
471 * returns: nothing
472 *
473 *****/
474 void comms_timmer(void) {
475     /* Timeout proportional to SF (high SF require more time) */
476     // SF 7 => 1 timeout (500 ms)
477     // SF 12 => 5 tiemouts (2.5 s)
478     if (protocol_timeout_counter >= SF - 7){
479         protocol_timeout = true;
480         protocol_timeout_counter = 0;
481     } else{
482         protocol_timeout_counter = protocol_timeout_counter + 1;
483     }
484 }
485
486 /*****
487 *
488 * Function: OnTxDone
489 * -----
490 * when the transmission finish correctly goes to LOWPOWER
491 *
492 * returns: nothing
493 *
494 *****/
495 void OnTxDone( void )
496 {
497     Radio.Standby( );
498     if (tx_flag == 1){
499         State = TX;
500     } else{
501         State = LOWPOWER;
502     }
503 }
504
505 /*****
506 *
507 * Function: OnRxDone
508 * -----
509 * processes the information when the reception has been done correctly
510 * calculates the rssi and snr
511 *
512 * payload: information received
513 * size: size of the payload
514 * rssi: rssi value
515 * snr: snr value

```

```

516 *
517 * returns: nothing
518 *
519 *****/
520 void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
521 {
522     Radio.Standby( );
523     uint16_t BufferSize = size;
524     memcpy( Buffer, payload, BufferSize );
525     RssiValue = rssi;
526     SnrValue = snr;
527     PacketReceived = true;
528     RssiMoy = ((RssiMoy * RxCorrectCnt) + RssiValue) / (RxCorrectCnt + 1);
529     ;
530     SnrMoy = (((SnrMoy * RxCorrectCnt) + SnrValue) / (RxCorrectCnt + 1));
531     State = RX;
532 }
533 /*****/
534 *
535 * Function: OnTxTimeout
536 * -----
537 * to process transmission timeout
538 *
539 * returns: nothing
540 *
541 *****/
542 void OnTxTimeout( void )
543 {
544     Radio.Standby( );
545     State = TX_TIMEOUT;
546 }
547
548 /*****/
549 *
550 * Function: OnRxTimeout
551 * -----
552 * to process reception timeout
553 *
554 * returns: nothing
555 *
556 *****/
557 void OnRxTimeout( void )
558 {
559     Radio.Standby( );
560     State = RX;
561     if( RxTimeoutTimerIrqFlag )
562     {
563         State = RX_TIMEOUT;
564     }
565     else
566     {
567         Radio.Rx( RX_TIMEOUT_VALUE ); // Restart Rx
568         SymbTimeoutCnt++; // if we pass here because of
569         Symbol Timeout
570         State = LOWPOWER;
571     }
572 }

```

```

572
573 /*****
574 *
575 * Function: OnRxErro
576 * -----
577 * function called when a reception error occurs
578 *
579 * returns: nothing
580 *
581 *****/
582 void OnRxError( void )
583 {
584     Radio.Standby( );
585     State = RX_ERROR;
586 }
587
588 /*****
589 *
590 * Function: OnCadDone
591 * -----
592 * Function to check if the CAD has been done correctly or not
593 *
594 * channelActivityDetected: boolean that contains the CAD flat
595 *
596 * returns: nothing
597 *
598 *****/
599 void OnCadDone( bool channelActivityDetected)
600 {
601     Radio.Standby( );
602
603     if( channelActivityDetected == true )
604     {
605         CadRx = CAD_SUCCESS;
606     }
607     else
608     {
609         CadRx = CAD_FAIL;
610     }
611     State = RX;
612 }
613
614 /*****
615 *
616 * Function: SX126xConfigureCad
617 * -----
618 * Function Configure the Channel Activity Detection parameters and IRQ
619 *
620 * returns: nothing
621 *
622 *****/
623 void SX126xConfigureCad( RadioLoRaCadSymbols_t cadSymbolNum, uint8_t
        cadDetPeak, uint8_t cadDetMin , uint32_t cadTimeout)
624 {
625     SX126xSetDioIrqParams( IRQ_CAD_DONE | IRQ_CAD_ACTIVITY_DETECTED,
        IRQ_CAD_DONE | IRQ_CAD_ACTIVITY_DETECTED,
626         IRQ_RADIO_NONE, IRQ_RADIO_NONE );

```



```

627     SX126xSetCadParams( cadSymbolNum, cadDetPeak, cadDetMin, LORA_CAD_RX,
        ((cadTimeout * 15.625) / 1000 ));
628 }
629
630 /*****
631 *
632 * Function: CADTimeoutTimeoutIrq
633 * -----
634 * Function called automatically when a CAD IRQ occurs
635 *
636 * returns: nothing
637 *
638 *****/
639 static void CADTimeoutTimeoutIrq( void )
640 {
641     Radio.Standby( );
642     State = LOWPOWER;
643 }
644
645 /*****
646 *
647 * Function: RxTimeoutTimerIrq
648 * -----
649 * Function called automatically when a Timeout interruption occurs
650 *
651 * returns: nothing
652 *
653 *****/
654 static void RxTimeoutTimerIrq( void )
655 {
656     RxTimeoutTimerIrqFlag = true;
657 }
658
659 /*****
660 *
661 * Function: pin_correct
662 * -----
663 * checks if the pin in the telecommand is correct
664 *
665 * pin_1: first byte of the pin
666 * pin_2: second byte of the pin
667 *
668 * returns: true if correct
669 *
670 *****/
671 bool pin_correct(uint8_t pin_1, uint8_t pin_2) {
672     if (pin_1 == PIN1 && pin_2 == PIN2){
673         return true;
674     }
675     return false;
676 }
677
678
679 /*****
680 *
681 * Function: process_telecommand
682 * -----
683 * processes the information contained in the packet depending on

```

```

684 * the telecommand received *
685 * * *
686 * header: number of telecomman *
687 * info: information contained in the received packet *
688 * * *
689 * returns: nothing *
690 * * *
691 *****/
692 void process_telecommand(uint8_t header, uint8_t info) {
693     uint64_t info_write; //Flash_Write_Data functions requires
694                          //uint64_t variables (64 bits) or arrays
695     switch(header) {
696     case RESET2:{
697         HAL_NVIC_SystemReset(); //Resets the STM32
698         break;
699     }
700     case NOMINAL:{ //Writes the new nominal value
701         info_write = info;
702         Flash_Write_Data(NOMINAL_ADDR, &info_write, 1);
703         //xTaskNotify("Task OBC", NOMINAL_NOTI, eSetBits); //Notification to
704         OBC
705         break;
706     }
707     case LOW:{ //Writes the new low value
708         info_write = info;
709         Flash_Write_Data(LOW_ADDR, &info_write, 1);
710         //xTaskNotify("Task OBC", LOW_NOTI, eSetBits); //Notification to OBC
711         break;
712     }
713     case CRITICAL:{ //Writes the new critical value
714         info_write = info;
715         Flash_Write_Data(CRITICAL_ADDR, &info_write, 1);
716         //xTaskNotify("Task OBC", CRITICAL_NOTI, eSetBits); //Notification to
717         OBC
718         break;
719     }
720     case EXIT_LOW_POWER:{ //Notifies the OBC to exit low power state
721         //xTaskNotify("Task OBC", EXIT_LOW_POWER_NOTI, eSetBits); //
722         Notification to OBC
723         break;
724     }
725     case EXIT_CONTINGENCY:{ //Notifies the OBC to exit contingency
726         state
727         //xTaskNotify("Task OBC", EXIT_CONTINGENCY_NOTI, eSetBits); //
728         Notification to OBC
729         break;
730     }
731     case EXIT_SUNSAFE:{ //Notifies the OBC to exit unsafe state
732         //xTaskNotify("Task OBC", EXIT_SUNSAFE_NOTI, eSetBits); //Notification
733         to OBC
734         break;
735     }
736     case SET_TIME:{ //Writes the synchronization time
737         uint8_t time[4];
738         for (k=0; k<4; k++){ //4 bytes variable
739             time[k]=Buffer[k+3];
740         }
741         Flash_Write_Data(TIME_ADDR, &time, sizeof(time));

```

```

736 //xTaskNotify("Task OBC", SETTIME_NOTI, eSetBits); //Notification to
OBC
737 break;
738 }
739 case SET_CONSTANT_KP:{ //Writes the constant KP
740 info_write = info;
741 Flash_Write_Data(KP_ADDR, &info_write, 1);
742 //xTaskNotify("Task OBC", CTEKP_NOTI, eSetBits); //Notification to OBC
743 break;
744 }
745 case TLE:{ //Writes the new TLE in memory
746 uint8_t tle[TLE_PACKET_SIZE];
747 /* TLE arrives in 2 packets of 69 bytes */
748 for (k=0; k<TLE_PACKET_SIZE; k++){
749 tle[k]=Buffer[k+3];
750 }
751 if (tle_telecommand){ //If it is the first TLE packet received
752 Flash_Write_Data(TLE_ADDR1, &tle, sizeof(tle));
753 } else{ //If it is the second TLE packet received
754 Flash_Write_Data(TLE_ADDR2, &tle, sizeof(tle));
755 }
756 //xTaskNotify("Task OBC", TLE_NOTI, eSetBits); //Notification to OBC
757 break;
758 //For high SF, 3 packets will be needed and the code should be adjusted
759 }
760 case SET_GYRO_RES:{ //Writes the gyroscope resolution
761 info_write = info;
762 Flash_Write_Data(GYRO_RES_ADDR, &info_write, 1);
763 //xTaskNotify("Task OBC", GYRORES_NOTI, eSetBits); //Notification to
OBC
764 break;
765 }
766 case SEND_DATA:{ //Activates TX flag and goes to TX state
767 if (!contingency){
768 tx_flag = true;
769 State = TX;
770 send_data = true;
771 }
772 break;
773 }
774 case SEND_TELEMETRY:{
775 /* Reads telemetry information from memory and sends it */
776 uint64_t read_telemetry[5]; //To read from flash 64 bits variables are
needed
777 uint8_t transformed[TELEMETRY_PACKET_SIZE]; //Maybe is better to
use 40 bytes, as multiple of 8
778 if (!contingency){
779 Flash_Read_Data(TELEMETRY_ADDR, &read_telemetry, sizeof(
read_telemetry));
780 Buffer[0] = MISSION_ID; //Satellite ID
781 Buffer[1] = POCKETQUBE_ID; //PocketQube ID (there are at least 3)
782 Buffer[2] = num_telemetry; //Number of the packet
783 memcpy(&transformed, read_telemetry, sizeof(transformed));
784 for (uint8_t i=3; i<TELEMETRY_PACKET_SIZE; i++){
785 Buffer[i] = transformed[i-3];
786 }
787 Buffer[TELEMETRY_PACKET_SIZE+3] = 0xFF; //Final of the packet
indicator

```

```

788     num_telemetry++;
789     DelayMs( 300 );
790     Radio.Send( Buffer, TELEMETRY_PACKET_SIZE+4 ); //Sends the
packet
791     State = RX;
792 }
793 break;
794 }
795 case STOP_SENDING_DATA:{           //Deactivates TX flag
796     tx_flag = false;
797     send_data = false;
798     break;
799 }
800 case ACK_DATA:{                   //Saves the nack data and goes to TX to
retransmit
801     if (!contingency && info != 0){
802         nack_flag = true;
803         memcpy(&nack, Buffer[3], sizeof(nack));
804         tx_flag = true;           //Activates TX flag
805         State = TX;
806         send_data = true;
807     }
808     break;
809 }
810 case SET_SF_CR: {                 //Writes the new SF and CR values and
calls
811                                     //configuration() to reconfigure the
transceiver
812     if (info == 0) SF = 7;
813     else if (info == 1) SF = 8;
814     else if (info == 2) SF = 9;
815     else if (info == 3) SF = 10;
816     else if (info == 4) SF = 11;
817     else if (info == 5) SF = 12;
818     info_write = SF;
819     Flash_Write_Data(SF_ADDR, &info_write, 1);
820     /*4 cases (4/5, 4/6, 4/7,1/2), so we will receive and store 0, 1, 2 or
3*/
821     info_write = Buffer[4];
822     Flash_Write_Data(CRC_ADDR, &info_write, 1);
823     DelayMs(10);
824     configuration();
825     break;
826 }
827 case SEND_CALIBRATION:{           //Writes the new calibration info received
828     /* CALIBRATION PACKET RECEIVED */
829     uint8_t calibration_packet[CALIBRATION_PACKET_SIZE];
830     for (k=0; k<CALIBRATION_PACKET_SIZE; k++){
831         calibration_packet[k]=Buffer[k+3];
832     }
833     Flash_Write_Data(CALIBRATION_ADDR, &calibration_packet, sizeof(
calibration_packet));
834     //xTaskNotify("Task OBC", CALIBRATION_NOTI, eSetBits); //Notification
to OBC
835     break;
836     //For high SF 2 packets will be needed and the code should be adjusted
837 }
838 case CHANGE_TIMEOUT:{

```

```

839     memcpy(&time_packets, Buffer[3], 2);
840     Flash_Write_Data(COMMS_TIME_ADDR, &time_packets, sizeof(time_packets));
841     break;
842 }
843 case TAKE_PHOTO:{           //Writes Payload 1 measurement parameters
844     Flash_Write_Data(PL_TIME_ADDR, &Buffer[3], 4);
845     info_write = Buffer[7];
846     Flash_Write_Data(PHOTO_RESOL_ADDR, &info_write, 1);
847     info_write = Buffer[8];
848     Flash_Write_Data(PHOTO_COMPRESSION_ADDR, &info_write, 1);
849     //xTaskNotify("Task OBC", TAKEPHOTO_NOTI, eSetBits); //Notification to
OBC
850     break;
851 }
852 case TAKE_RF:{             //Writes Payload 2 measurement parameters
853     Flash_Write_Data(PL_TIME_ADDR, &Buffer[3], 8);
854     info_write = Buffer[11];
855     Flash_Write_Data(F_MIN_ADDR, &info_write, 1);
856     info_write = Buffer[12];
857     Flash_Write_Data(F_MAX_ADDR, &info_write, 1);
858     info_write = Buffer[13];
859     Flash_Write_Data(DELTA_F_ADDR, &info_write, 1);
860     info_write = Buffer[14];
861     Flash_Write_Data(INTEGRATION_TIME_ADDR, &info_write, 1);
862     //xTaskNotify("Task OBC", TAKERF_NOTI, eSetBits); //Notification to OBC
863     break;
864 }
865 case SEND_CONFIG:{
866     /* Reads configuration information from memory and sends it */
867     uint64_t read_config[4];
868     uint8_t transformed[CONFIG_PACKET_SIZE];
869     if (!contingency){
870         Flash_Read_Data(CONFIG_PACKET_SIZE, &read_config, sizeof(read_config)
);
871         Buffer[0] = MISSION_ID;           //Satellite ID
872         Buffer[1] = POCKETQUBE_ID;       //PocketQube ID (there are at least 3)
873         Buffer[2] = num_config;          //Number of the packet
874         memcpy(&transformed, read_config, sizeof(transformed));
875         for (uint8_t i=3; i<CONFIG_PACKET_SIZE; i++){
876             Buffer[i] = transformed[i-3];
877         }
878         Buffer[CONFIG_PACKET_SIZE+3] = 0xFF;           //Final of the packet
indicator
879         num_config++;
880         DelayMs( 300 );
881         Radio.Send( Buffer, CONFIG_PACKET_SIZE+4 ); //Sends the packet
882         State = RX;
883     }
884     break;
885 }
886 default:{                 //If the telecommand received does not
exist
887                             //goes to TX and sends an error message
888     State = TX;
889     error_telecommand = true;
890 }
891 }
892 }

```

G.2.2 Comms.h

```

1 /*
2  * comms.h
3  *
4  * Created on: 24 feb. 2022
5  * Author: Daniel Herencia Ruiz
6  */
7
8 #ifndef INC_COMMS_H_
9 #define INC_COMMS_H_
10
11 #include <stdbool.h>
12 #include <stdlib.h>
13 #include <stdio.h>
14 #include <stdint.h>
15 #include <inttypes.h>
16
17 #include <string.h>
18 #include <math.h>
19 #include "board.h"
20 #include "sx126x.h"
21 #include "radio.h"
22 #include "sx126x-board.h"
23
24 #include "flash.h"
25 #include "definitions.h"
26
27 #define RF_FREQUENCY 868000000 // Hz
28
29 #define TX_OUTPUT_POWER 22 // dBm
30
31 #define LORA_BANDWIDTH 0 // [0: 125 kHz,
32 // 1: 250 kHz,
33 // 2: 500 kHz,
34 // 3: Reserved]
35 #define LORA_SPREADING_FACTOR 7 // [SF7..SF12]
36 #define LORA_CODINGRATE 1 // [1: 4/5,
37 // 2: 4/6,
38 // 3: 4/7,
39 // 4: 4/8]
40 #define LORA_PREAMBLE_LENGTH 8//108 // Same for Tx and
41 Rx
42 #define LORA_SYMBOL_TIMEOUT 100 // Symbols
43 #define LORA_FIX_LENGTH_PAYLOAD_ON false
44 #define LORA_IQ_INVERSION_ON false
45 #define LORA_FIX_LENGTH_PAYLOAD_LEN 19
46 #define WINDOW_SIZE 20
47
47 #define RX_TIMEOUT_VALUE 4000
48 #define BUFFER_SIZE 100
49
50 #define TLE_PACKET_SIZE 66
51 #define TELEMETRY_PACKET_SIZE 34
52 #define CALIBRATION_PACKET_SIZE 96
53 #define CONFIG_PACKET_SIZE 30
54

```

```

55 /*!
56 * CAD performance evaluation's parameters
57 */
58 #define RX_FW 1
59 #define TX_FW 0 //TX_FW is only for
    test
60 #define FULL_DBG 1 //Active all traces
61
62 // Apps CAD timer
63 TimerEvent_t CADTimeoutTimer;
64 #define CAD_TIMER_TIMEOUT 1000 //Define de CAD
    timer's timeout here
65
66 TimerEvent_t RxAppTimeoutTimer;
67 #define RX_TIMER_TIMEOUT 4000 //Define de CAD
    timer's timeout here
68
69 //CAD parameters
70 #define CAD_SYMBOL_NUM LORA_CAD_02_SYMBOL
71 #define CAD_DET_PEAK 20
72 #define CAD_DET_MIN 10
73 #define CAD_TIMEOUT_MS 2000
74 #define NB_TRY 10
75
76 #define UPLINK_BUFFER_SIZE 100
77 #define ACK_PAYLOAD_LENGTH 5 //ACK payload data
    length
78 #define CONFIG_SIZE 13
79
80
81 /*!
82 * Radio events function pointer
83 */
84 static RadioEvents_t RadioEvents;
85
86 /*!
87 * \brief Function to be executed on Radio Tx Done event
88 */
89 void OnTxDone( void );
90
91 /*!
92 * \brief Function to be executed on Radio Rx Done event
93 */
94 void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr );
95
96 /*!
97 * \brief Function executed on Radio Tx Timeout event
98 */
99 void OnTxTimeout( void );
100
101 /*!
102 * \brief Function executed on Radio Rx Timeout event
103 */
104 void OnRxTimeout( void );
105
106 /*!
107 * \brief Function executed on Radio Rx Error event
108 */

```

```
109 void OnRxError( void );
110
111 /*!
112 * \brief Function executed on Radio CAD Done event
113 */
114 void OnCadDone( bool channelActivityDetected);
115
116 /*!
117 * \brief Function configuring CAD parameters
118 * \param [in] cadSymbolNum The number of symbol to use for CAD
119 *                               operations
120 *                               [LORA_CAD_01_SYMBOL, LORA_CAD_02_SYMBOL,
121 *                               LORA_CAD_04_SYMBOL, LORA_CAD_08_SYMBOL,
122 *                               LORA_CAD_16_SYMBOL]
123 * \param [in] cadDetPeak Limit for detection of SNR peak used in the
124 *                               CAD
125 * \param [in] cadDetMin Set the minimum symbol recognition for CAD
126 * \param [in] cadTimeout Defines the timeout value to abort the CAD
127 *                               activity
128 */
129 void SX126xConfigureCad( RadioLoRaCadSymbols_t cadSymbolNum, uint8_t
130     cadDetPeak, uint8_t cadDetMin , uint32_t cadTimeout);
131
132 /*!
133 * \brief CAD timeout timer callback
134 */
135 static void CADTimeoutTimeoutIrq( void );
136
137 /*!
138 * \brief Rx timeout timer callback
139 */
140 static void RxTimeoutTimerIrq( void );
141
142 /*!
143 * \brief Average the collected RSSIs during CAD
144 */
145 int8_t AverageCadRssi( void );
146
147 /*!
148 * \brief Get the last good RSSI during CAD
149 */
150 int8_t GetLastCadRssi( void );
151
152 /*!
153 * \brief Display collected RSSIs each ms during CAD
154 */
155 void DisplayCadRssivsTime( void );
156
157 void StateMachine( void );
158
159 void txfunction( void );
160
161 void configuration(void);
162
163 void process_telecommand(uint8_t header, uint8_t info);
164
165 bool pin_correct(uint8_t pin_1, uint8_t pin_2);
166
```



```

163 void tx_beacon(void);
164
165 void comms_timmer(void);
166
167 #endif /* INC_COMMS_H_ */

```

G.2.3 GroundStation.ino

```

1
2 #include "LoRaWan_APP.h"
3 #include "Arduino.h"
4
5 #ifndef LoraWan_RGB
6 #define LoraWan_RGB 0
7 #endif
8
9 /*This are LoRa definitions*/
10 #define RF_FREQUENCY 868000000 // 868 MHz
11 #define TX_OUTPUT_POWER 22 // 22 dBm
12 #define LORA_SPREADING_FACTOR 7//9 // [0x09 --> SF9]
13 #define LORA_BANDWIDTH 0 // Radio.h changes it
    Bandwidths[] = { LORA_BW_125, LORA_BW_250, LORA_BW_500 } // [0x04 -->
    125 kHz]
14 #define LORA_CODINGRATE 1 // [0x01 --> CR=4/5]
15 #define LORA_PREAMBLE_LENGTH 8 // CSS modulations
    usually have 8 preamble symbols
16 #define TX_TIMEOUT_VALUE 3000//340 // Air time Tx
17 #define PACKET_LENGTH 59 // Packet Size
18
19 #define TX_WINDOW_TIMEOUT 13900
20
21 #define LORA_SYMBOL_TIMEOUT 0
22 #define LORA_FIX_LENGTH_PAYLOAD_ON false
23 #define LORA_FIX_LENGTH_PAYLOAD_LEN 38
24 #define LORA_IQ_INVERSION_ON false
25
26 #define RX_TIMEOUT_VALUE 0
27 #define BUFFER_SIZE 100 // Define the payload
    size here
28 #define WINDOW_SIZE 20
29
30 #define CAD_TIMER_TIMEOUT 3000//1000 //Define de CAD timer's
    timeout here
31 #define RX_TIMER_TIMEOUT 3000//4000 //Define de CAD timer's
    timeout here
32 #define CAD_SYMBOL_NUM 0x01
33 #define CAD_DET_PEAK 22
34 #define CAD_DET_MIN 10
35 #define CAD_TIMEOUT_MS 2000
36 #define NB_TRY 10
37
38 //----- TELECOMMANDS -----
39 #define RESET2 0x01
40 #define NOMINAL 0x02
41 #define LOW_POWER 0x03
42 #define CRITICAL 0x04
43 #define EXIT_LOW_POWER 0x05

```



```

92 byte telecommand11_1[] = {0xC8, 0x9D, 0x0B, 0x31, 0x20, 0x34, 0x31, 0x37, 0
    x33, 0x32, 0x55, 0x20, 0x31, 0x36, 0x30,
93 0x35, 0x31, 0x42, 0x20, 0x31, 0x36, 0x32, 0x36, 0x36, 0x2e, 0x33, 0x30, 0
    x31, 0x39, 0x39, 0x31, 0x34, 0x34,
94 0x20, 0x2e, 0x30, 0x30, 0x30, 0x30, 0x32, 0x35, 0x33, 0x31, 0x20, 0x30, 0
    x30, 0x30, 0x30, 0x30, 0x2d, 0x30,
95 0x20, 0x31, 0x30, 0x37, 0x39, 0x35, 0x2d, 0x33, 0x20, 0x30, 0x20, 0x39, 0
    x39, 0x39, 0x38, 0x0a, 0x32, 0x20}; //Total first packet => 69 bytes
    (3 + 66)
96 //Second TLE packet:
97 byte telecommand11_2[] = {0xC8, 0x9D, 0x0B, 0x34, 0x31, 0x37, 0x33, 0x32, 0
    x20, 0x39, 0x37, 0x2e, 0x33, 0x37, 0x33, 0x34, 0x20, 0x31, 0x38, 0x31, 0
    x2e,
98 0x34, 0x35, 0x32, 0x30, 0x20, 0x30, 0x30, 0x31, 0x35, 0x35, 0x33, 0x37, 0
    x20, 0x31, 0x33, 0x31, 0x2e, 0x32,
99 0x32, 0x39, 0x34, 0x20, 0x33, 0x34, 0x39, 0x2e, 0x32, 0x37, 0x32, 0x32, 0
    x20, 0x31, 0x35, 0x2e, 0x32, 0x33,
100 0x37, 0x33, 0x31, 0x30, 0x37, 0x30, 0x20, 0x35, 0x37, 0x32, 0x38, 0xFF}; //
    TOTAL 69 (3 + 65 + ff)
101
102 byte telecommand12[] = {0xC8, 0x9D, 0x0C};
103
104 byte telecommand20[] = {0xC8, 0x9D, 0x14, 0x00};
105 byte telecommand21[] = {0xC8, 0x9D, 0x15};
106 byte telecommand22[] = {0xC8, 0x9D, 0x16};
107 byte telecommand23[] = {0xC8, 0x9D, 0x17};
108 byte telecommand24[] = {0xC8, 0x9D, 0x18};
109 byte telecommand25[] = {0xC8, 0x9D, 0x19};
110
111 byte telecommand30[] = {0xC8, 0x9D, 0x1E};
112
113 byte telecommand40[] = {0xC8, 0x9D, 0x28};
114
115 byte telecommand50[] = {0xC8, 0x9D, 0x32};
116
117 byte photo[] = {};
118 uint32_t air_time;
119 uint8_t Buffer[BUFFER_SIZE];
120 uint8_t reception_mode = 1;
121
122 uint8_t count_packet[] = {0}; //To count how many packets have been sent (
    maximum WINDOW_SIZE)
123 uint8_t count_window[] = {0}; //To count the window number
124 uint8_t count_rtx[] = {0}; //To count the number of retransmitted
    packets
125 uint8_t i = 0; //Auxiliar variable for loop
126
127 uint64_t ack; //Information rx in the ACK
128 uint8_t nack_number; //Number of the current packet to retransmit
129 bool nack = false; //True when retransmission necessary
130 bool full_window = false; //Stop & wait => to know when we reach the
    limit packet of the window
131
132 char txpacket[BUFFER_SIZE]; //Creates the package
133
134 void tx_function(void);
135 void rx_function(void);
136 void packaging(void);

```

```

137 void resetCommsParams(void);
138 void OnTxDone( void );
139 void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr );
140 void OnTxTimeout( void );
141 void OnRxTimeout( void );
142 void OnRxError( void );
143 void OnCadDone( bool channelActivityDetected);
144 static void CADTimeoutTimeoutIrq( void );
145 static void RxTimeoutTimerIrq( void );
146 //void array_to_string(uint8_t array1[], unsigned int len, char string[]);
147
148 /*
149  * STATE MACHINE VARIABLES
150  */
151 typedef enum
152 {
153     LOWPOWER,
154     RX,
155     RX_TIMEOUT,
156     RX_ERROR,
157     TX,
158     TX_TIMEOUT,
159     START_CAD,
160 }States_t;
161
162 typedef enum
163 {
164     CAD_FAIL,
165     CAD_SUCCESS,
166     PENDING,
167 }CadRx_t;
168
169 States_t State;
170
171 int8_t RssiValue = 0;
172 int8_t SnrValue = 0;
173
174 CadRx_t CadRx = CAD_FAIL;
175 bool PacketReceived = false;
176 bool RxTimeoutTimerIrqFlag = false;
177 int16_t RssiMoy = 0;
178 int8_t SnrMoy = 0;
179 uint16_t RxCorrectCnt = 0;
180 uint16_t BufferSize = BUFFER_SIZE;
181 uint16_t PacketCnt = 0;
182 uint16_t rx_counter = 0;
183
184 TimerEvent_t CADTimeoutTimer;
185 TimerEvent_t RxAppTimeoutTimer;
186 /*
187  * ----- END -----
188  */
189
190 void setup() {
191     Serial.begin(115200);
192     RadioEvents.TxDone = OnTxDone;
193     RadioEvents.RxDone = OnRxDone;
194     RadioEvents.TxTimeout = OnTxTimeout;

```

```
195 RadioEvents.RxTimeout = OnRxTimeout;
196 RadioEvents.RxError = OnRxError;
197 RadioEvents.CadDone = OnCadDone;
198
199 Radio.Init( &RadioEvents );
200
201 Radio.SetChannel( RF_FREQUENCY );
202
203 Radio.SetTxConfig( MODEM_LORA, TX_OUTPUT_POWER, 0, LORA_BANDWIDTH,
204                  LORA_SPREADING_FACTOR, LORA_CODINGRATE,
205                  LORA_PREAMBLE_LENGTH, LORA_FIX_LENGTH_PAYLOAD_ON,
206                  true, 0, 0, LORA_IQ_INVERSION_ON, TX_TIMEOUT_VALUE );
207
208 Radio.SetRxConfig( MODEM_LORA, LORA_BANDWIDTH, LORA_SPREADING_FACTOR,
209                  LORA_CODINGRATE, 0, LORA_PREAMBLE_LENGTH,
210                  LORA_SYMBOL_TIMEOUT, LORA_FIX_LENGTH_PAYLOAD_ON,
211                  0, true, 0, 0, LORA_IQ_INVERSION_ON, true );
212
213 air_time = Radio.TimeOnAir( MODEM_LORA , PACKET_LENGTH );
214 ack = 0xFFFFFFFFFFFFFFFF;
215 nack = false;
216 State= TX;
217
218 //Timer used to restart the CAD
219 TimerInit( &CADTimeoutTimer, CADTimeoutTimeoutIrq );
220 TimerSetValue( &CADTimeoutTimer, CAD_TIMER_TIMEOUT );
221
222 //App timer used to check the RX's end
223 TimerInit( &RxAppTimeoutTimer, RxTimeoutTimerIrq );
224 TimerSetValue( &RxAppTimeoutTimer, RX_TIMER_TIMEOUT );
225
226 Radio.StartCad(2); // do the config and lunch first CAD
227 }
228
229
230 void loop() {
231     Radio.IrqProcess( );
232     switch( State )
233     {
234         case RX_TIMEOUT:
235             {
236                 Radio.Standby;
237                 State = LOWPOWER;
238                 break;
239             }
240         case RX_ERROR:
241             {
242                 PacketReceived = false;
243                 Radio.Standby;
244                 State = LOWPOWER;
245                 break;
246             }
247         case RX:
248             {
249                 if( PacketReceived == true )
250                 {
251                     PacketReceived = false; // Reset flag
252                     RxCorrectCnt++; // Update RX counter
```

```

253     #if(FULL_DBG)
254         //printf( "Rx Packet n %d\r\n", PacketCnt );
255     #endif
256         rx_counter = rx_counter + 1;
257         //Serial.print(Buffer[2],HEX);
258         //Serial.println(" Position 3 Received");
259         //Serial.print(last_telecommand[2],HEX);
260         //Serial.println(" Last telecommand send");
261         //if (rx_counter == 5){
262         //Serial.print(Buffer[0],HEX);
263         //Serial.println(" Position 0 Rx");
264         Serial.print("Received paquet number ");
265         Serial.println(Buffer[2],DEC);
266         //for (int i = 3; i < BUFFER_SIZE; i = i + 1) {
267         for (int i = 0; i < BUFFER_SIZE; i = i + 1) {
268             Serial.print("0x");
269             Serial.print(Buffer[i],HEX);
270             Serial.print(", ");
271             if (((i-2)%16 == 0) && i != 3){
272                 Serial.println();
273             }
274         }
275         if(Buffer[2] == 0x41){
276             Serial.println("ERROR MESSAGE RECEIVED");
277         }
278         else if (Buffer[2] == 0x44){
279             Serial.println("BEACON MESSAGE RECEIVED");
280         }
281         else if (Buffer[2] == last_telecommand[2]){
282             delay(100); //Optimum value
283             //delay(1000);
284             Radio.Send( ack_telecommand, 3);
285             Serial.println("ACK SEND");
286             delay(100);
287             Radio.Send( ack_telecommand, 3); //discomment this line
288             Serial.println("ACK SEND");
289         }
290         else if (Buffer[0] == 'E' && Buffer[1] == 'N' && Buffer[2]
== 'D'){
291             reception_mode = 0;
292             delay(1000);
293         }
294         State = LOWPOWER;
295     }
296     else
297     {
298         if (CadRx == CAD_SUCCESS)
299         {
300             //PUT HERE THE CODE TO WITHDRAW THE INFO FROM THE BUFFER
301
302             //channelActivityDetectedCnt++; // Update counter
303         #if(FULL_DBG)
304             printf( "Rxing\r\n");
305         #endif
306             RxTimeoutTimerIrqFlag = false;
307             TimerReset(&RxAppTimeoutTimer); // Start the Rx's's
Timer
308             //Radio.Rx( RX_TIMEOUT_VALUE );

```

```

309         }
310         else
311         {
312             TimerStart(&CADTimeoutTimer); // Start the CAD's
Timer
313         }
314         Radio.Rx( RX_TIMEOUT_VALUE ); //Basic RX code
315         delay(1); //Basic RX code
316         State = LOWPOWER;
317     }
318     break;
319 }
320 case TX:
321 {
322     printf("Send Packet n %d \r\n",PacketCnt);
323     if( PacketCnt == 0xFFFF)
324     {
325         PacketCnt = 0;
326     }
327     else
328     {
329         PacketCnt ++;
330     }
331     //Send Frame
332     delay(300);
333     //Serial.println("I'm going to tx");
334     //tx_function();
335     //Radio.Send( telecommand11_1, sizeof(telecommand11_1) ); //
TEST SEND TLE
336     //delay(500);//TLE
337     //Radio.Send( telecommand11_2, sizeof(telecommand11_2) ); //
TEST SEND TLE
338
339     //Radio.Send( telecommand20, sizeof(telecommand20) ); //TEST
SEND DATA
340     Radio.Send( telecommand2, sizeof(telecommand2) ); //TEST
TELECOMMAND 2 (LOW)
341     for (int i = 0; i < sizeof(telecommand2); i = i + 1) {
342         Serial.print(telecommand2[i],HEX);
343         Serial.print(" ");
344     }
345     Serial.println();
346     delay(300); //Optimum delay for SF = 7 is 300 ms
347     //delay(1000); //Force error paquet => timeout before 2nd
telecommand rx
348     //Radio.Send( telecommand20, sizeof(telecommand20) ); //TEST
SEND DATA
349     Radio.Send( telecommand2, sizeof(telecommand2) ); //TEST
TELECOMMAND 2 (LOW)
350     delay(100);
351     for (int i = 0; i < sizeof(telecommand2); i = i + 1) {
352         Serial.print(telecommand2[i],HEX);
353         last_telecommand[i] = telecommand2[i];
354         Serial.print(" ");
355     }
356     Serial.println();
357     telecommand_send = true;
358     reception_mode = 1;

```

```

359         State = LOWPOWER;
360         Buffer[0] = 'H';
361         break;
362     }
363     case TX_TIMEOUT:
364     {
365         State = LOWPOWER;
366         break;
367     }
368     case LOWPOWER:
369     default:
370         if( reception_mode == 1){
371             State = RX;
372         } else{
373             State = TX;
374         }
375         // Set low power
376         //State = TX;
377         break;
378     }
379     TimerLowPowerHandler( );
380 }
381
382 void tx_function(void) {
383     //configuration();
384     if (!full_window)
385     {
386         packaging(); //Start the TX by packaging all the data that will be
387         transmitted
388         //SX126xSetPayload(); //Aquesta fa el writebuffer, sha de posar
389         direccions com a la pag 48 del datasheet
390         Radio.Send( Buffer, BUFFER_SIZE );
391         //Serial.println("Packet Tx");
392         //array_to_string(Buffer, BUFFER_SIZE, txpacket);
393         /*
394         for (unsigned int i = 0; i < BUFFER_SIZE; i++)
395         {
396             Serial.print(Buffer[i],HEX);
397             Serial.print(" ");
398         }*/
399         Serial.printf("\r\nTX packet \"%s\" \r\n",Buffer);
400         Serial.println();
401         //txpacket[BUFFER_SIZE*3] = '\0';
402         //Serial.println(strlen(txpacket));
403     }
404     Serial.print("Airtime: ");
405     Serial.print(air_time,DEC);
406     Serial.println();
407     if (full_window){
408         reception_mode = 1;
409         //Radio.Standby;
410         State = RX;
411         full_window = false;
412         delay(300);
413         Serial.println("Vuelvo a RX");
414     }
415     //delay(300);

```



```

415 }
416
417 void packaging(void) {
418     //NACK packets at the beginnig of the next window
419
420     if (nack)
421     {
422         while(i<sizeof(ack) && nack_number != i-1)
423         {
424             //function to obtain the packets to retx
425             if(!((ack >> i) & 1)) //When position of the ack & 1 != 1 --> its a 0
426                 --> NACK
427             {
428                 nack_number = i; //Current packet to rtx
429                 //Packet from last window => count_window - 1
430                 //Flash_Read_Data( PHOTO_ADDR + (count_window[0]-1)*WINDOW_SIZE*
431                 BUFFER_SIZE + (nack_number)*BUFFER_SIZE , Buffer , sizeof(Buffer) ); //
432                 Direction in HEX
433                 for (int i = 0; i < BUFFER_SIZE; i = i + 1) {
434                     Buffer[i] = photo[(count_window[0]-1)*WINDOW_SIZE*BUFFER_SIZE + (
435                     nack_number)*BUFFER_SIZE + i];
436                 }
437                 count_rtx[0]++;
438             }
439             i++;
440         }
441         if (i==sizeof(ack)){
442             i=0;
443         }
444         else if (nack_number == i-1){
445             i++;
446         }
447     }
448     else //no NACKS
449     {
450         //Serial.println("Posiciones memoria enviadas: ");
451         //Flash_Read_Data( PHOTO_ADDR + count_window[0]*WINDOW_SIZE*BUFFER_SIZE
452         + (count_packet[0]-count_rtx[0])*BUFFER_SIZE , Buffer , sizeof(Buffer)
453         ); //Direction in HEX
454         for (int i = 0; i < BUFFER_SIZE; i = i + 1) {
455             Buffer[i] = photo[count_window[0]*WINDOW_SIZE*BUFFER_SIZE + (
456             count_packet[0]-count_rtx[0])*BUFFER_SIZE + i];
457             Serial.print(count_window[0]*WINDOW_SIZE*BUFFER_SIZE + (count_packet
458             [0]-count_rtx[0])*BUFFER_SIZE + i,DEC);
459             Serial.print(" ");
460         }
461         //Serial.println("Salgo del for");
462         if (count_packet[0] < WINDOW_SIZE-1)
463         {
464             count_packet[0]++;
465         }
466         else
467         {
468             Serial.println("Ventana Acabada");
469             reception_mode = 1;
470             count_packet[0] = 0;

```

```

465     //count_window[0]++;
466     count_window[0]=0;
467     full_window = true;
468 }
469 }
470 };
471
472 /*This function is called when a new photo is stored in the last photo
    position*/
473 void resetCommsParams(void) {
474     count_packet[0] = 0;
475     count_window[0] = 0;
476     count_rtx[0] = 0;
477 }
478
479 void OnTxDone( void )
480 {
481     Radio.Standby( );
482     State = TX;
483 }
484
485 void OnRxDone( uint8_t *payload, uint16_t size, int16_t rssi, int8_t snr )
486 {
487     Radio.Standby( );
488     BufferSize = size;
489     memcpy( Buffer, payload, BufferSize );
490     RssiValue = rssi;
491     SnrValue = snr;
492     PacketReceived = true;
493     RssiMoy = (((RssiMoy * RxCorrectCnt) + RssiValue) / (RxCorrectCnt + 1))
    ;
494     SnrMoy = (((SnrMoy * RxCorrectCnt) + SnrValue) / (RxCorrectCnt + 1));
495     //Serial.printf("\r\nreceived packet \"%s\" with rssi %d , length %d\r\
    n",Buffer,rssi,size);
496     State = RX;
497 }
498
499 void OnTxTimeout( void )
500 {
501     Radio.Standby( );
502     State = TX_TIMEOUT;
503     Serial.println("OnTxTimeout");
504 }
505
506 void OnRxTimeout( void )
507 {
508     Radio.Standby( );
509     if( RxTimeoutTimerIrqFlag )
510     {
511         State = RX_TIMEOUT;
512     }
513     else
514     {
515         #if(FULL_DBG)
516             printf(".");
517         #endif
518         Radio.Rx( RX_TIMEOUT_VALUE ); // Restart Rx
519         //SymbTimeoutCnt++; // if we pass here because of

```

```
    Symbol Timeout
520     State = LOWPOWER;
521 }
522 Serial.println("OnRxTimeout");
523 }
524
525 void OnRxError( void )
526 {
527     Radio.Standby( );
528     State = RX_ERROR;
529     Serial.println("OnRxError");
530 }
531
532 void OnCadDone( bool channelActivityDetected)
533 {
534     Radio.Standby( );
535
536     if( channelActivityDetected == true )
537     {
538         CadRx = CAD_SUCCESS;
539     }
540     else
541     {
542         CadRx = CAD_FAIL;
543     }
544     State = RX;
545     Serial.println("OnCadDone");
546 }
547
548 static void CADTimeoutTimeoutIrq( void )
549 {
550     Radio.Standby( );
551     //State = START_CAD;
552     State = RX;
553     Serial.println("CADTimeoutIrq");
554 }
555
556 static void RxTimeoutTimerIrq( void )
557 {
558     RxTimeoutTimerIrqFlag = true;
559 }
```