CubeCat-1: Implementation, testing and integration of the communication subsystem

Kattia Flores Pozo

SUPERVISED BY
Adriano Camps, Roger Jové

Universitat Politècnica de Catalunya
Master in Aerospace Science & Technology
July 2012
CubeCat-1: Implementation, testing and integration of the communication subsystem

BY
Kattia Flores Pozo

DIPLOMA THESIS FOR THE DEGREE
Master in Aerospace Science and Technology

AT
Universitat Politècnica de Catalunya

SUPERVISED BY:
Adriano Camps, Roger Jové
Theory and Signal Processing
ABSTRACT

CubeCat-1 is an educational oriented Cubesat – a standard, small form factor satellite project started in 2008 in the UPC in Barcelona, under the supervision of professors Adriano Camps and Juan Ramos, with many students involved. The main objective of the CubeCat-1 project is to get students involved in a hands-on multidisciplinary group frame work.

A primary mission requirement of any satellite is the ability to exchange information with a ground based command station, establishing a reliable link to send telemetry, health status from sensors and scientific payload data and being able to receive commands from Earth.

The goal of this project is to present the design and software implementation of part of the communications subsystem for the Cubesat, continuing the work started by the first CubeCat-1 team (at that time named UPCSat-1) and referenced in the master thesis.

We present:

- The description of the hardware.
- The design and implementation of the software responsible for encoding and decoding telemetry and payload data.
- The Ground segment description and tests performed to guarantee the correct communication with the Cubesat and the integration with the GENSO network.
- Lessons learned, solutions and remaining issues are presented in detail.
Table of Contents

INTRODUCTION ........................................................................................................... 1

CHAPTER 1 FROM THE BEGINNING OF THE CUBESAT TO THE CUBECAT-1 ................ 4
1.1. The CubeSat: The Picosatellite Standard for Research and Education .......... 4
1.2. Launchers and Orbital Deployers ........................................................................ 6
   1.2.1. Orbital Deployers ....................................................................................... 6
   1.2.2. Requirements for deployment .................................................................... 7
   1.2.3. Launch Vehicles ........................................................................................ 7
1.3. Cubesat subsystems: Making it work ................................................................. 7
   1.3.1. Electrical Power Subsystem ................................................................------- 9
   1.3.2. Attitude Control Subsystem ..................................................................... 11
   1.3.3. Thermal Control Subsystem ..................................................................... 12
   1.3.4. On Board Computer ............................................................................... 13
   1.3.5. Structural Subsystem .............................................................................. 14
   1.3.6. Communications subsystem Overview .................................................... 15
   1.3.7. The spectrum available and the HAM Amateur Bands ......................... 16
   1.3.8. The GENSO network ............................................................................ 18
   1.3.9. Modulation: some useful concepts ............................................................ 19
   1.3.10. Coding techniques .................................................................................. 24
1.4. CubeCat-1 Project Overview ............................................................................. 25
   1.4.1. The team ................................................................................................... 25
   1.4.2. Mission objectives .................................................................................... 26
   1.4.3. Scientific missions ................................................................................... 26
       1.4.3.1. Optical camera ................................................................................... 26
       1.4.3.2. Geiger counter ................................................................................... 27
   1.4.4. Technological Demonstrators .................................................................. 28
       1.4.4.1. CubeCat-1 Cellsats .......................................................................... 28
       1.4.4.2. LCD shutter .................................................................................... 28
       1.4.4.3. Peltier Cells ....................................................................................... 28
       1.4.4.4. Resonant Inductive Coupling experiment ........................................ 29
       1.4.4.5. New transistor types ........................................................................... 30
1.5. Goals of this MSc. thesis .................................................................................. 30

CHAPTER 2 SYSTEM REQUIREMENTS ................................................................. 32
2.1. Functionality ...................................................................................................... 32
2.2. General constraints ......................................................................................... 32
2.3. GENSO Recommended Radio Configuration .................................................. 33
2.4. Frequency Bands, maximum power and bandwidth allowed ......................... 35
2.5. Cubesat Standard Regulations ......................................................................... 36
2.6. Doppler shift .................................................................................................... 36
2.7. Derived Requirements ...................................................................................... 37
5.5.3.4. Change default settings ................................................................. 70
5.5.3.5. Enter data mode ............................................................................. 70
5.5.3.6. Read/Write in Serial port ................................................................. 70
5.5.3.7. Read/Write in data bus ................................................................. 70

5.6. Ground Station settings .................................................................... 71

5.7. GENSO Integration ........................................................................... 71
5.7.1. Ground Station Switch ................................................................. 73

5.8. Received Signal ................................................................................. 74

CHAPTER 6 FUTURE WORK AND LESSONS LEARNED ......................... 76

CHAPTER 7 CONCLUSION ........................................................................ 80
List of Figures

Figure 1.1 CubeCat-1, 1U Cubesat ................................................................................. 4
Figure 1.2 QB50, a network of Cubesats, an artist conception................................. 5
Figure 1.3 P-POD deployer, Cal-Poly Standard. .............................................................. 6
Figure 1.4 CubeCat-1 general subsystems scheme ......................................................... 9
Figure 1.5 CubeCat-1 EPS subsystem scheme ............................................................... 11
Figure 1.6 PortuxG20 scheme. [8] .................................................................................. 14
Figure 1.7 a) Pumpkin and b) ISIS structures for 1U Cubesats [10] [11]. .............. 15
Figure 1.8 Cubesat communication subsystem general scheme ......................... 16
Figure 1.9 GENSO amateur stations network block diagram [12] ............................. 19
Figure 1.10 ASK, FSK and PSK modulation techniques ............................................. 20
Figure 1.11 Probability of bit error for non-coherent reception of M-ary FSK[13]..... 22
Figure 1.12 Probability of bit error for non-coherent reception of FSK and GFSK [14]. ............................................................................................................. 23
Figure 1.13 Comparison of the probability of bit error for the different most used techniques, BPSK, MSK and FSK [15] ...................................................... 23
Figure 1.14 Principle of channel encoding....................................................................... 24
Figure 1.15 CubeCat-1, a mission for students. (Named before, UPCSAT)......... 26
Figure 1.16 Left, a picture taken by Masat-1, right: camera chosen to go on board the CubeCat-1 [16]........................................................................................................ 27
Figure 1.17 Geiger counter, to travel on board CubeCat-1 [18]............................... 28
Figure 1.18 Solar panels in the vacuum chamber during a setup to characterize the peltier Cell In the CubeCat-1 laboratory. ......................................................... 29
Figure 2.1 Orbit propagation for the CubeCat-1 done in the STK software, in violet, the passes over the UPC Ground Station ......................................................... 33
Figure 2.2 GENSO basic ground station. ........................................................................ 35
Figure 3.1 Communications subsystem diagram proposed (by Mr. Roger Olivé).... 40
Figure 3.2 ASK Transmitter Module TWS-BS-3 [31] ..................................................... 41
Figure 3.3 GENSO compatible ISIS ground Station installed in the UPC campus .. 42
Figure 3.4 TNC function is to encode and decode AX.25 frames............................... 44
Figure 3.5 TNC7 multi front view. The switch selects between the preset modes and LED’s show the actual status............................................................................. 45
Figure 3.6 CC1101 Radio block diagram [33] ................................................................ 46
Figure 3.7 Development board block diagram, shows the integration of the MCU with the transceiver chip [33]............................................................................. 47
Figure 3.8 MHX 425 radio block diagram [6] ................................................................. 48
Figure 4.1 The link budget analysis takes into account all the gains and losses from the transmitter to the receiver end. ................................................................. 49
Figure 4.2 Link margin in dB to guarantee a BER of $10^{-5}$ using MSK at 9600bps during a.......................................................................................... 54
Figure 4.3 Link margin in dB to guarantee a BER of $10^{-5}$ using 2FSK at 9600bps during a single pass.............................................................. 55
Figure 4.4 Link margin in dB to guarantee a BER of $10^{-5}$ using 2FSK at 12000bps during a ................................................................. 55
Figure 4.5 Uplink performance for the main transceiver at the margin elevation angle ..................................................................................................................... 57
Figure 4.6 Downlink performance for the main transceiver at the margin elevation angle ..................................................................................................................... 58
Figure 5.1 Flow diagram of the software proposed.......................................................... 62
Figure 5.2 SmartRF Studio screenshot. The software is used to configure most of the transceiver settings. ................................................................. 64
Figure 5.3 Flow diagram of the transmission chain merging the transceiver commands and the AX.25 functions. ................................................................. 65
Figure 5.4 Diagram of the software libraries relationship. ................................. 66
Figure 5.5 Flow diagram for the functions of the MHX 425 transceiver. ............... 68
Figure 5.6 TNC data connections diagram. .......................................................... 71
Figure 5.7 Screenshot of the data packets received from ISS in the Ground Station. ........................................................................................................ 72
Figure 5.8 Screenshot of beacon signal received on the VEGA launch day from PWSat in the Ground Station ................................................................. 73
Figure 5.9 Circuit simulation screenshot of the switch. ........................................ 74
Figure 5.10 Screenshot of the Visual interface used in the simulation. ................. 74
Figure 5.11 Packet received in the MixW software ............................................. 75
Figure 6.1 Flow diagram of the reception software on the ground station to allow the retransmission of packets. ................................................................. 77
Figure 6.2 Flow diagram of segmentation of data in the OBC to send it to the transceiver. ........................................................................................................ 78
List of tables

Table 2.1 Maximum power and bandwidths allowed in amateur bands [26] [27] ..... 35
Table 4.1 Ground Station antennas parameters .................................................. 51
Table 4.2 Ground Station cable losses ................................................................. 51
Table 4.3 Orbital parameters ................................................................................. 52
Table 4.4 EbN0 required for the modulation schemes supported by the transceiver 52
Table 4.5 Comparison between performance over time window between the different modulation schemes and data rate ........................................................................ 53
Table 4.6 Downlink system performance for the worst case, 10° of elevation. ....... 56
Table 4.7 Uplink system performance for the worst case, 10° of elevation .......... 56
Table 5.1 AX.25 UI frame format [35] .................................................................. 60
Table 5.2 AX.25 UI Frame, 276 bytes ................................................................. 61
Table 5.3 Point to Point default settings for MHX425 ........................................ 69
INTRODUCTION

A Cubesat is a standard Pico-satellite of 1000 cm³ and a mass of no more than 1.33 kg. The standard came as an initiative of Cal Poly University, Profs. Jordi Puig-Suari and Bob Twiggs, in order to provide a general guideline for the design of Pico-satellites to reduce cost, development time and to increase accessibility to space; therefore, sustaining frequent launches.

CubeCat-1 is an educational oriented Cubesat project started at UPC in Barcelona, under the tuition of Profs. Adriano Camps and Juan Ramos. The main objective of the CubeCat-1 project is to get students involved in a hands-on multidisciplinary group frame works, and at the same time validate different subsystems self-build with COTS components in order to make it less costly and novel.

A primary mission requirement of any satellite is the ability to exchange information with a ground-based command station. Such information includes scientific payload data, sensors data and telemetry data. The implementation of the communication subsystem for a Cubesat is a challenging engineering problem since the size and power are tough constraints to meet.

The goal of this project is to detail the design and the implementation of the communications subsystem for the Cubesat, continuing the work started by the first CubeCat-1 team and referenced in the master thesis. This is, the description of the hardware, the design and implementation of the software responsible for encoding and decoding telemetry and payload data, the Ground segment description and tests done to guarantee the correct communication with the Cubesat and finally to show the lessons learned during the process and to provide the technical documentation for future members of the team about the work performed so far to make it easy to modify, test or improve.

The report gives an introduction of Cubesats and CubeCat-1 in Chapter 1.

Chapter 2 identifies and performs an analysis of the requirements for the communications system.

In Chapter 3 details the system architecture and its practical implementation.

Chapter 4, a link budget analysis is performed.

Chapter 5 shows the software development and tests performed.

Chapter 6 gives a discussion of the lessons learned; future works and recommends improvements for the next version of the system.

Finally Chapter 7 gives general conclusions.

Appendix contains the relevant information to understand the terms and principles discussed in this master thesis.
Chapter 1

FROM THE BEGINNING OF THE CUBESAT TO THE CUBECAT-1

In this chapter, first a brief introduction to the Cubesat standard, the origins, the deployers, and the subsystems general requirements and constraints is given. We focus on the communication subsystem, since this master thesis will detail this one. Second, we describe the CubeCat-1 project, its objectives, the technological demonstrators and the scientific payloads. At last, the main objectives of the master thesis into the framework described are presented.

1.1. The CubeSat: The Picosatellite Standard for Research and Education

In 1999, the CubeSat standard was born at California Polytechnic State University (Cal Poly) under initiative of Profs. Jordi Puig-Suari and Bob Twiggs.

CubeSats are Pico-class satellites, with the smallest ones having the dimensions of a ten centimeter side cube (1U Cubesat, figure 1.1), and a maximum weight of 1.33 kg [1]. Satellites with the dimensions of approximately 10x10x20 centimeters and 10x10x30 centimeters are called 2U and 3U CubeSats respectively.

![Figure 1.1 CubeCat-1, 1U Cubesat](image)
CubeSats give developers standard specifications for size, weight and basic construction, which enable parts to be built as a “one-size-fits-all” type of arrangement. The CubeSat-class space crafts have the advantages of being able to serve as a test bed for new core space technologies and to carry small scientific payloads to be applied to larger space programs, for much lower cost, shorter schedule, and less risk. Due to these features, Cubesats have become an affordable way for educational and scientific initiatives to access space.

Just as an example of the wide variety of missions carried out in Cubesats, it can be mentioned the last in-orbit deployment of seven Cubesats in the VEGA maiden flight on February, this missions involved more than 250 university students from six different countries over the last four years, this were:

- **Xatcobeo** (a collaboration of the University of Vigo and INIA, Spain): a mission to demonstrate software-defined radio and solar panel deployment;

- **Robusta** (University of Montpellier, France): a mission to test and evaluate radiation effects (low dose rate) on bipolar transistor electronic components;

- **e-st@r** (Politecnico di Torino, Italy): demonstration of an active 3-axis attitude determination and control system including an inertial measurement unit;

- **Goliath** (University of Bucharest, Romania): imaging of Earth using a digital camera and in-situ measurement of radiation dose and micrometeoroid flux;

- **PW-Sat** (Warsaw University of Technology, Poland): a mission to test a deployable atmospheric drag augmentation device for de-orbiting CubeSats;

- **MaSat-1** (Budapest University of Technology and Economics): a mission to demonstrate various spacecraft avionics, including a power conditioning system, transceiver and on-board data handling;

- **UniCubeSat GG** (University of Rome La Sapienza, Italy): a mission to study the gravity gradient.

![Figure 1.2 QB50, a network of Cubesats, an artist conception](image-url)
Projects such as the QB50 have to be mentioned as well, which aims to send a swarm of Cubesats at once (50 at least) to study in-situ the temporal and spatial variations of a number of key constituents and parameters in the lower thermosphere (Figure 1.2).

This is how nowadays, the construction of Cubesats has increased, under the motto of “Faster, Cheaper, Better” that can perform missions traditionally assigned to large/medium satellites.

1.2. Launchers and Orbital Deployers

1.2.1. Orbital Deployers

CubeSats are deployed using standardized launch vehicle interfaces (LVI).

Even though there are several deployers so far, the most commonly used is the P-POD (Poly Picosatellite Orbital Deployer) (Figure 1.3) developed by the California Polytechnic State University. The main design goal of the deployer was to provide a standard interface to launch Cubesats safely and in group, protecting the primary load and the launch vehicle.

The mechanism follows a simple principle, it is a 3U (10cmx10cmx30cm) aluminum box of tubular design which uses a spring-loaded system of ejection. The P-POD can deploy 1U, 2U or 3U Cubesats in any combination.

![Figure 1.3 P-POD deployer, Cal-Poly Standard.](image)

There are other deployment systems apart from the P-POD, for example, the T-POD (Tokyo Pico-satellite Orbital Deployer), X-POD (eXperimental Push Out Deployer) designed by the University of Toronto, SPL (Single Pico Launcher designed by Astrotein, a private company, or the ISIS deployers, ISIPOD for one, two, three, or up to six units deployer.
The choice of the deployer will depend on the launcher compatibility and acceptance.

1.2.2. Requirements for deployment

According to the standards, for the Cubesat, and the deployer, any Cubesat aiming to be launched, should meet a list of requirements, mechanical, electrical, operational and legal. These requirements can be found in the website of the developers and the corresponding space agencies.

Especially when thinking on an opportunity launch, like the last VEGA maiden flight, aspects like the frequency band allocation need to be coordinated with the regulation entity to avoid interference with the other payloads [2].

As an example, in the VEGA maiden flight, where seven Cubesats were put in orbit, an exhaustive compliance checks were performed. Some requirements to be met are, Cubesats should not be powered up and should not deploy anything until after 30 min of the deployment from the P-POD.

Concerning the frequencies used by the Cubesats in VEGA launch, they had all different frequencies at least 25 kHz away from each other, which is coordinated previously with the corresponding institutions. Each P-POD deployed the Cubesats almost at the same time (second deployment after 30 sec, and so on the third one). None of the frequencies are repeated since they would interfere with each other by the proximity at which they are launched.

1.2.3. Launch Vehicles

To put a spacecraft in orbit is expensive, it usually costs tens of thousands dollars, and considering the small mass of a Cubesat, it is not worth it as an individual effort, but by placing several units in piggy backs (not used spare space in big launchers) makes sharing the costs more affordable.

In this scenario though, in most of cases, the orbit is not defined until the CubeSat is all set and done. This happens because the launching company will give, of course, priority to the main payload (a bigger satellite for example), and also the final orbit will have less accuracy than the primary load as it will be deployed in a different time.

Also, space agencies like ESA and NASA give some opportunities to educational Cubesat projects called “launch opportunities” when they are launching a scientific mission and have some spare space in the launcher. This was the case for example of the Vega maiden flight on February 13th, 2012, where seven Cubesats were deployed into a LEO orbit free of charge.

1.3. Cubesat subsystems: Making it work

As in any other spacecraft, the Cubesat works as the integration and interaction of some basic systems (see Figure 1.4).
The payload is the satellite mission; this can't work without a platform of minimal functionality. Starting from the structure itself that contains all the subsystems and works as the interface with the launcher.

The electric power supply for all the electronic devices on board requires a subsystem generating, storing and distributing power to the whole system. As the temperature in space varies from -170 °C (in shadow) to +100 °C (in light) a thermal control system is needed to ensure that all the equipment is working within a controlled range of temperatures.

The payload and antennas may need to be pointed to a certain geographical area; therefore the attitude control system will provide this control to the Cubesat. Finally and most important, all the subsystems must be controlled by an on board computer and monitored remotely from Earth, receiving and sending data and commands, to assure this the communications and data handling subsystem is needed. The subsystems interact among them through a common data bus.

The subsystems then can be summarized as:

- The structural subsystem
- The electrical and power subsystem (EPS)
- The communications subsystem (COMMS)
- The attitude control subsystem (ACS)
- Thermal Control Subsystem (TCS)
- Payloads
- The On-board computer (OBH)
1.3.1. **Electrical Power Subsystem**

The Electric Power Subsystem (EPS) is responsible for generating electrical power from the solar panels, storing this energy in batteries and distributing regulated power to the different devices of the Cubesat via common power lines.

The subsystem should have a reliable and robust design ensuring protection and recovery against unwanted peaks, cuts and faults which could cause a loss of the mission. It should be able to handle peak power consumption operating modes, and it should take into account the electronic degeneration performance during the time of...
the mission. For a one unit Cubesat system, the average power generated is 1-1.5 W (Swiss Cube for example generates 1.5 W during daylight) [4].

Constant telemetry should be sent to the control station in order to monitor the correct functionality of the EPS. The telemetry can include readings from the voltage, current, and temperature status of the solar panels, the batteries, the battery charger and the regulators.

In the design, several aspects should be taken into account, just to mention the main ones:

**Solar Panels**

The photo-voltaic system, consisting of the solar panels is the part of the subsystem in charge of collecting energy from the Sun, complying with the Cubesat specifications, solar panels are the only feasible source of energy.

The solar cells efficiency will impose the power factor limit of the whole system, therefore the design and election should take into account many important factors like the efficiency-cost-mass, the lifetime degradation factor of the cells, the area dedicated to the solar panels, the radiation resistance (space validation).

**Batteries**

While solar cells provide energy to the power bus of the system during day light, the energy collected by the solar panels is also stored into the batteries to provide energy to the Cubesat while in dark.

When choosing the batteries, some characteristics to take into account are: the nominal voltage, energy density, the self-discharge rage, the operational temperature range, the total charging life cycles depending of the length of the mission, the mass, dimensions and if they have been space qualified.

**Maximum Power Point Tracker and battery Charger Regulator**

To maximize the energy transfer efficiency the maximum power point tracker makes sure to get the maximum possible power from the solar panels array by setting the proper resistance and maintaining the voltage of the array at its optimum value when the power requirement demands it.

The battery charger regulator protects the batteries from being overcharged and therefore decreasing their lifetime.

**Converters**

These devices are necessary to provide the required voltages to the different devices of the Cubesat, these voltages are usually 5 V or 3 V, from the batteries and solar panels output.

**Kill switch**

That turns off the power system while in the deployer, usually in redundancy mode.
The following diagram shows a general schematic of the EPS used in the CubeCat-1.

![Diagram of EPS subsystem](image)

**Figure 1.5** CubeCat-1 EPS subsystem scheme.

### 1.3.2. Attitude Control Subsystem

The attitude Control main function is to determine and control the Cubesat position in Space compensating any disturbance presence.

The subsystem can be divided in the attitude sensors, the actuators and the data processing unit.

The attitude determination can be performed by two approaches, by using reference sensors or inertial sensors. Reference sensors determine the attitude relative to one or more objects, for example the Earth. Inertial sensors determine the position by integrating the rate of change in attitude, but for this, the initial value for the integration has to be given by a reference sensor anyway.

Sensors can be:

- Earth sensors (sensing thermal emissions from Earth),
- Sun sensors (sensing the angle of incidence of the Sun),
- Magnetometers sensing the Earth's magnetic field,
- Gyroscopes, that sense the spin rate in any direction,
- GPS,
- Star imagers (sensitive cameras and a Stars map but requires extensive data processing),

- Gyroscope sensors, inertial sensors used mainly while fast attitude changes where reference sensors are not anymore coherent.

The attitude can be:

- Stabilized by the gradient of the gravitational force which fixes the orientation of the Cubesat by using only the mass distribution and placing counterweights to increase the difference in moments of inertia and therefore minimizing the misalignment due to unwanted disturbances.

- Permanent magnet stabilization, by placing magnets on the satellite which will tend the Cubesat to align with the magnetic fields of the Earth.

- Magneto torques: consists of three orthogonal copper coils (for three axis control) acting as actuators. When current is applied, magnetic dipole moments are generated, which in interaction with the Earth magnetic field produce control torques. The coils can be wound copper coils or copper etching on PCB panels, first approach result in larger magnetic dipole moment. To sense the Earth magnetic field, a three axis magnetometer is required.

- Fly Wheel Controlled, by transferring unwanted angular momentum to reaction or momentum wheels. For three axis control, three orthogonal fly wheels can be used for fine pointing.

- Gas jets

CubeCat-1 team has chosen to monitor the attitude of the CubeSat using a 9 degrees of freedom sensor (3-Axis gyroscope sensors, 3-Axis magnetometers and 3-Axis accelerometers).

1.3.3. Thermal Control Subsystem

Space is a harsh environment, a satellite orbiting Earth has many heat sources around, depending on the distance to Sun, the solar flux is around 1367 W/m² at summer (to a distance of 1 Astronomic Unit), the albedo is of about the 30 %, the Earth infrared radiation can be represented as a black body emitting at a temperature of 255 K. And while in shadow, the deep space emits 3 K. Therefore the satellite components are exposed to large cyclic changes of temperature which electronics are not able to tolerate.

In order to keep the temperatures of the electronics inside their operating ranges, the thermal control subsystem is in charge of keeping the thermal balance of the satellite this means that the heat absorbed should be equal to the heat emitted.

There are passive and active systems:
The passive approach includes the coating or painting of surfaces, multilayer insulations for non-radiating surfaces, the use of heat pipes, and radiating systems are some of the most common solutions. The active control includes peltier cells and heaters.

While designing, it is recommended to perform simulations of the total time of light and darkness exposure. There are existing tools to do so which perform a very accurate simulation according to the orbital data provided like STK [5] or Thermal Desktop [6].

1.3.4. On Board Computer

The on-board computer is the subsystem in charge of keeping the whole system working in the necessary operation mode. It controls, interfaces and process the data in the main data bus from the payloads and the other subsystems. The core of the subsystem is usually a micro-controller of low power consumption, RAM and flash memories to store software and payload data, real time clocks, oscillators, logic auxiliary, bus connectors interfaces (UART, I2C, SPI), and over-current and voltage protection.

While designing, the hardware components should be preferable space qualified, otherwise, should go through radiation, thermal and vibration tests in advance. The software should be reliable and fault-resistance probed.

The OBC usually fits in a single board along with the main bus; there are several custom integrated models on the market, some already space qualified. For example the Pumpkin motherboards, have used microprocessors from Microchip, Texas Instruments or ARM processors in their different versions.

These companies offer operative systems as well, like SALVO [7], or recently open source software are gaining more popularity for the large community support and because they are free of cost.

In fact, the team has chosen a Linux based On Board Computer, the PortuxG20 (Figure 1.6), which operates at 400 MHz, and has an ARM-9 processor as core. It has most of the common interfaces as Ethernet, USART, USB, Micro SD, JTAG and SPI along with 64 I/O ports, giving the compatibility necessary for all the subsystems.
1.3.5. Structural Subsystem

The objective of the structural subsystem is to provide a modular, sturdy structure that will protect both the Cubesat and the launcher, while providing easy assembly of the components maximizing the usable inner space and minimizing the complexity and cost of the design.

The design of the CubeSat has to meet the standards required by the deployer and launcher. The shape of the structure, as stated in the Cal Poly CubeSat Standard is essentially a cube, with outer dimensions of 10 x 10 x 10 cm$^3$, with 3.0 mm clearance above each face of the cube for mounting exterior components such as antenna, data link and power charger inlet port.

The satellite must have four launch rails along four edges of the cube, allowing for easy ejection from the P-POD launch tube.

To maintain spacing and prevent sticking with other CubeSats, standoff contacts or feet must exist at the ends of these rails; therefore the four rails are extruded by 5 mm on all ends. The center of mass of the CubeSat must be within ±2 cm of the geometric center [9].

The maximum allowable mass of CubeSat is 1.33 kg, and it is desired that the structure be no more than approximately 30% of the total CubeSat mass.

The structural subsystem shall have an external kill switch, and should pass harmonic and random vibration tests. Suggested material for the main satellite structure is Aluminum 7075 or 6061, Stainless Steel, Titanium, Composites, and Honey Comb. [9]

The most widely used structures available in the market are the ones from Pumpkin [10] and ISIS [11].
This ISIS structure faces are totally covered by plates, while Pumpkin one has a “skeleton” design, making it easier to interact with the mounted boards, though it has only one kill switch. Both structural solutions have been put in orbit successfully.

![Figure 1.7 a) Pumpkin and b) ISIS structures for 1U Cubesats [10] [11].](image)

The CubeCat-1 has chosen the ISIS structure since it complies with most of the launchers requirements for having two kill switches and to avoid future compatibility problems.

1.3.6. Communications subsystem Overview

One of the most important parts of any satellite is the communications subsystem, without the ability to communicate with Earth, mission will be lost.

In order to establish a link with Earth, the design of the system depends on:

- The orbit: determines the contact time between the Ground Station and the satellite.

- Data rate needed: How much data is needed to send and how long the contact with the Ground Station is.

- The power constraints: Power available on the Cubesat, which is a strong handicap, needed to establish the link.

- Frequency band selection: Depends on both the data rate needed and the power available and it needs coordination with the regulator entity of the spectrum.

- Gains and losses: From the Ground station, the satellite and along the path.
The general scheme of the communications subsystems is shown in the following figure, an RF front end: the antennas, connectors, RF switches (if needed), an impedance adaption network, following by an amplifying stage (high power amplifiers or low noise amplifiers) and a modulator/demodulator stage. Finally, an encoder/decoder which will interface with the main data bus of the Cubesat.

![Diagram of Cubesat communication subsystem general scheme](image)

**Figure 1.8** Cubesat communication subsystem general scheme

To close the link, the Ground station is on the Earth’s side, with again, the same chain. Antennas (with high sensibility, since power is not a constraint on Earth), rotors (to actively track the Cubesat), amplifiers, a radio transceiver, a modem, and a TNC (Terminal Network Controller) working as decoder/encoder for the data and a PC controlling the tracking.

The Ground Station design depends on the compatibility with the Cubesat communications system design and the coordination with the frequency regulator entities.

In the next section, regulation and frequency allocations issues will be discussed in detail since they are one of the most important criteria while choosing the Ground station equipment, and part of the Cubesat communication subsystem design.

In the following sections, the existing modulation and coding techniques will be briefly described. These concepts are important to understand how the communication is established and to understand the trade-offs of choosing one or another.

### 1.3.7. The spectrum available and the HAM Amateur Bands

Spectrum is a highly demanded resource. Therefore its allocation is regulated and coordinated by an international organization, the International Telecommunications Union (ITU). The ITU has divided the spectrum according to the purpose of the use and the geographical region. The frequency ranges designated for satellites are over the SHF band and it takes a long and costly process to get an allocation.

Since Cubesats are defined as an educational purpose standard, were budget it’s tried to keep as minimum as possible, and time mission development to be short,
most of the Cubesat missions launched have chosen the Amateur band as a practical choice.

**What is an amateur band?**

The purpose of an amateur satellite is defined by IARU as:

“Amateur satellite bands are reserved for the purpose of self-training, intercommunication and technical investigations carried out by amateurs, that is, duly authorized persons interested in radio technique solely with a personal aim and without pecuniary interest. A radio communication service using space stations on Earth satellites for the same purposes as those of the amateur service.”

(http://www.iaru.org/)

The frequency bands available for amateur satellite operations in the region are 144 – 148 MHz, 435 – 438 MHz, 1260 – 1270 MHz and 2400 – 2450 MHz; there are also some higher frequencies available. It is important to note that other services are sharing these frequency bands.

The most demanded sub-bands are the 435-438 MHz and 144-148 MHz ones, which presently are the only space-to-Earth amateur allocation between 144 MHz and 2.4 GHz. Because of the crowding of the existing bands with unmanned amateur satellites and manned space stations, it is desirable to study an expansion of the band. Even though sharing in this band with the Radio allocation Service has been successful over many decades because of the geographic separation and other factors.

**International Amateur Radio Union (IARU)**

Is the organization responsible of regulating the allocation of the radio spectrum among radio amateurs worldwide for a better mutual use of it.

**Why to choose an amateur band?**

- The coordination has the help of the IARU (described later)
- There are no long FCC applications; they can be used without formal notification and application to the ITU.
- Fewer technical restrictions are imposed
- It can provide an expanded network of Earth stations for data downlinks (wide Amateur community)
- There is readily available equipment
- It can provide technical development opportunities for students and grow the rank of highly technical proficient amateurs.
- Licensed amateurs can use any frequency in their bands (rather than being allocated fixed frequencies or channels) and can operate medium to high-powered
equipment on a wide range of frequencies, so long as they meet spurious emission standards.

**Impositions**

- The international communication must be in plain language, meaning that the protocol and framing format must be public available.

- All other communication should be open for use by amateur radio operators worldwide.

- All downlink telemetry data and formats must be unencrypted and published, so anybody with a receiver can find out detailed information about the spacecraft and payload.

**Though there are some drawbacks as well:**

- The control operator should be a licensed amateur operator.

- Amateur radio is licensed on an individual basis, not institutional.

- Recently, in the United States there has been interference from amateur stations to radio location stations, which has been resolved on a case-by-case basis by mitigation techniques or by taking amateur repeaters off the air.

- The allocations of 25 kHz or less do not support high-speed downlinks.

**1.3.8. The GENSO network**

GENSO is a project of the European Space Agency coordinated by ESA's education office. The aim of the project is to form a worldwide network of amateur ground stations interacting through a common standard software platform through Internet in order to support educational space missions.

This network increases the level of access to orbital space crafts by allowing a near global coverage in communication for every educational satellite launched, greatly increasing the return from educational space missions and the opportunities for sending commands to the spacecraft. Having only one ground station gives access to a very small communication window, therefore not being able to send as much data as it would be desirable.
The design and implementation work is being carried out by a distributed set of student and radio amateur teams worldwide and with over 80 educational spacecraft currently planned there is a very large demand for this project.

So far, the university heading the project, the University of Vigo, in Galicia has been in charge of the network, having released on February, 2012 the second version of the GENSO network, having performed improvements on previous bugs, and adding some hardware compatibility features.

More details will be provided in next chapter about the specifications required for the Ground Stations in order to be part of the network.

1.3.9. Modulation: some useful concepts.

To transfer the information through a channel, we need to adapt the signal to the transmission medium. This is the modulation stage of any communications subsystem.

When modulating a signal, the carrier can be phase-modulated, frequency-modulated or amplitude-modulated. This can be either done in a digital or analog way.

It is not possible to make a wireless transmission using a base band signal

Using analog modulations requires larger bandwidths and powers as compared to digital modulations, and yet they have less channel efficiency. Therefore, they are not suitable for Cubesat systems.
In the following the focus will be on digital modulation techniques. Baseband digital signals modulate a continuous wave (CW) high-frequency carrier. Three well known techniques are:

- Amplitude shift keying.
- Frequency shift keying
- Phase shift keying

These main binary modulation methods are illustrated in the following figure:

**Figure 1.10** ASK, FSK and PSK modulation techniques.

**Amplitude Shift Keying (ASK/OOK):** Also known as On-Off keying because depending on the information (1 or 0) the carrier signal is switched on or off.

But during the transmission of 0, the response is not ideal “off” signal, but there is a presence of noise which can be misinterpreted as data. To overcome this problem, '1' and '0' are assigned to two different amplitudes. To get the maximum power it is possible to suppress the carrier and filter one of the side-bands to preserve the bandwidth.

**Frequency Shift Keying (FSK)**

The frequency of the carrier is switched between two values, one representing the '1' and the other representing the '0'.

The system uses two different frequencies for the values 0 and 1 of each bit. If $B$ is the base frequency (the carrier) and $d$ the carrier deviation in frequency, each time a '0' is transmitted, a waveform of frequency $B-d$ (a symbol), and to transmit a 1 it creates a waveform of frequency $B+d$. The receiver just needs to measure the deviation of the signal to the reference frequency $B$ to know which value of the bit
was transmitted. When the modulation rate increases, the difference between the two chosen frequencies also need to be higher, this has a restriction in the bandwidth.

2FSK is the simplest form of FSK, it uses one bit per symbol (‘1’ or ‘0’), but it is possible to use more bits per symbol.

4-FSK uses four different symbols and therefore needs 4 different carrier deviations; in this case each symbol is mapped as a combination of two bits (00, 01, 10, 11). This doubles the signaling rate, but requires a higher received Signal to Noise ratio due to the shorter distance between symbols, which makes the system more sensitive to noise and interferences between symbols.

Therefore between 2-FSK and 4-FSK, the choice between a slow modulation which works even on weak signals and a high speed modulation which requires a stronger received signal. In other words, the higher the signaling rate, the shorter the range.

Gaussian Shift Keying (GFSK): Is a 2-FSK form shaped by a Gaussian filter with a roll off factor of 0.5. The parameters of the Gaussian filter determine how much the basic spectrum of the lateral bands will be narrowed.

Audio Frequency Shift Keying (AFSK): Is a special FSK modulation scheme using two tones, 2200 and 1200Hz, to modulate a binary signal into a carrier wave. It uses audible tones which can be transmitted through electronic circuits carrying sound, it is a simple system but less efficient in both power and bandwidth.

Phase Shift Keying (PSK): Here, the phase of the carrier is discretely varied with respect to either a reference phase or to the phase of the immediately preceding signal element in accordance with the binary data.

PSK systems with only two different phase angles are called Binary -PSK systems (BPSK), in this, the bit rate equal the modulation rate.

QPSK (Quadrature phase shift keying) is one of the most used forms of PSK modulation, it consist on two BPSK systems operating in quadrature. The input bit stream is split in two bit streams, one containing the even numbered bits, and the other the odd numbered bits.

Offset QPSK is similar to QPSK, but the two different streams, even and odd are shifted by an offset equal to the duration of one bit in one stream. In offset QPSK, the possibility of the carrier phase changing state by 180º, is eliminated as only one bit stream can change its state at any time. This reduces interference problems that QPSK has.

Minimum Shift Keying (MSK): Minimum Shift Keying is FSK with a modulation index of 0.5. Therefore the carrier phase of an MSK signal will be advanced or retarded 90º over the course of each bit period to represent either a one or a zero. Due to this exact phase relationship MSK can be considered as either phase or frequency modulation. The result of this exact phase relationship is that MSK can’t practically be generated with a voltage controlled oscillator and a digital waveform. Instead, an IQ modulation technique, as for PSK, is usually implemented.
Coherent demodulation is usually employed for MSK due to the superior bit-error-rate (BER) performance. This is practically achievable, and widely used in real systems, due to the exact phase relationship between each bit.

**Gaussian Minimum Shift Keying (GMSK):** Is a form of continuous-phase FSK, the phase is changed between symbols to provide a constant envelope. It is a popular alternative to QPSK. The RF bandwidth is controlled by the bandwidth of the Gaussian low-pass filter.

**Comparison between modulation Schemes**

The probability of error is a function of the signal to noise ratio (SNR) and the number of bits per symbol used in a modulation scheme (M). The higher the modulation level, more energy per bit is needed, therefore the distance between symbols is smaller, making it more susceptible to inter symbol interferences and having a greater chance of having errors.

\[
\frac{E_b}{N_0} \quad \text{is the energy per bit, it is a relation between the signal to noise ratio, the frequency of bit, and the bandwidth.}
\]

These parameters are useful when comparing techniques.

\[
(SNR_{[dB]} = \frac{E_b}{N_0} + 10 \cdot \log(f_b) - 10 \cdot \log(B))
\]

**Figure 1.11** Probability of bit error for non-coherent reception of M-ary FSK[13].
Figure 1.12 Probability of bit error for non-coherent reception of FSK and GFSK [14].

Figure 1.13 Comparison of the probability of bit error for the different most used techniques, BPSK, MSK and FSK [15].
After having reviewed the relationships between the probability of error and the energy per bit to spectral noise density ratio required it can be said that:

- FSK is the simplest technique to implement, but it has a wider spectrum than the other techniques. By increasing the number of bits per symbol, for example between 2-FSK and 4-FSK, to achieve the same probability of error, less energy is required, but also implementation becomes more complicated.

- GFSK is as simple as FSK, and it has a smaller spectrum due to the Gaussian filtering, but it also has a worse noise immunity as compared to FSK.

- MSK has the better noise immunity compared to 2FSK, 4FSK and GFSK, but the implementation of the modulator and demodulator is more complex.

1.3.10. Coding techniques

Coding the information means adding redundant bits to the information bits in order to be used in the receiver to detect and correct errors. This approach is called Forward Error Correction (FEC).

The code rate is defined as the relationship between the information bits \((n)\) and the total bits sent \((n+s):\) information bits + redundant bits). The codification can be done by:

- Block encoding: Each block of N bits is encoded in a different way by adding \(n\) bits of redundancy. The code bits are generated by a linear combination of the information bits.

- Convolutional encoding: \((n+r)\) bits are generated by the encoder from the \((N-1)\) preceding packets of \(n\) bits of information. The product \(N\) \((n+r)\) defines the constraint length of the code (Figure 1.12).
Under stable propagation conditions and Gaussian noise where random errors are present, convolutional encoding is used; block encoding is preferable under fading conditions where errors occur mostly in bursts.

**Interleaving**

Is a way to improve the performance of the convolutional encoding against burst errors. It consists of rearranging the encoded bits before transmission, in this way consecutive data are separated at least a predefined distance. This protects the information against bursts errors due to a deep channel fading or after decoding a convolutional code.

**Manchester Encoding**

Is a line code, it can be described as a special case of BPSK (Binary Phase Shift Keying) where the data controls the phase of a square wave carrier whose frequency is the data rate.

**Cyclic redundancy check (CRC)**

It is an error detecting code which uses systematic cyclic codes. This technique adds a fixed length check value to a packet of information. CRC codes are generated by polynomials. For each block of data to be sent a fixed short binary sequence is added. On the receiver side, the receiver computed in the same way as the transmitter the code attached to the information and compares it. If it doesn’t match, the receiver can perform a corrective action such as request to resend.

### 1.4. CubeCat-1 Project Overview

#### 1.4.1. The team

The CubeCat-1 project has involved many students since it started; the groups were divided by the subsystems of the Cubesat, working in a cooperative way. Often, one team member would have found himself involved in more than one subsystem. The first report of the project was presented as a final project thesis by Alberto Sanchez and Jordi Serra, here the team tried to state the basics of the designs of the CubeCat-1.

Continuing with this work, this master thesis has focused in solving the communication related tasks. Lately, 12 more students got involved and gave the project a big push, having so far almost completed most of the tasks left.

It is important to mention as well, that the CubeCat-1 project, thanks to the effort of the directors, owns all the testing facilities to simulate an space environment, this includes a thermal vacuum chamber (TVAC) of \(10^{-7}\) mBar with a temperature range going from -190 °C to +100 °C in different profile configurations, a sun simulator with the ability of irradiating a power equivalent twice what the Sun does, and very soon a vibration table facility. With this testing equipment, it will be possible to pre-qualify off-the-shelf components and the whole system to validate it for a future launch.
1.4.2. Mission objectives

The main goal of CubeCat-1 is to be a hands-on satellite experience for students. Since its beginning, it has been fully designed and developed by students. It can be said that two important things will be tested, the system reliability (all the subsystems, EPS, attitude control, communications and thermal control) and the payloads, the scientific projects on board as well as the technology demonstrators (electronics or devices which have never been put in orbit before). In the next sections, the technological demonstrators and scientific payloads going on board the CubeCat-1 will be described briefly.

![CubeCat-1, a mission for students. (Named before, UPCSAT)](image)

1.4.3. Scientific missions

1.4.3.1. Optical camera

A camera installed in a Cubesat face facing towards the Earth will take pictures at a certain time of day by a command sent through the Ground Station or at the same time each day by command of the OBC. The picture will be stored on board and then sent to Earth in the immediate next contact. The purpose of this payload is to show through images the capability of the satellite once in orbit.

The camera chosen is a Link Sprite JPEG color camera of only 32 mmx32 mm size which allows capturing and outputting JPEG images through an UART interface, making it easy to integrate with the main data bus. The resolution available is of VGA/QVGA/160x120 pixels. It has a very low power consumption, (100 mA max) and
a simple command interface. This is important since at a first stage of the project, several tests were performed with other types of cameras, but due to its unstable software interface, it was not possible to continue with them.

The commands allow making changes in the compression rate of the picture, to select the image size, the baud rate, change the power mode and reset the camera.

To protect the camera lenses from radiation, a shutter will be implemented. In the figure below, a picture taken by a similar camera on a Cubesat, by the Masat-1 [16], a Hungarian Cubesat, is shown.

Figure 1.16 Left, a picture taken by Masat-1, right: camera chosen to go on board the CubeCat-1 [16].

1.4.3.2. Geiger counter

A Geiger counter is a device that measures the level of ionizing radiation in a space.

The ionizing radiation can be Alpha (positively charged particles), Beta (negatively charged particles) or Gamma particles (cosmic radiation). Gamma particles on Earth conform what is called background radiation. Solar particle events produce high injection of alpha particles [17].

Since the Cubesat will be exposed to radiation, the objective of this payload is to measure through the counter the effects of radiation particles and collect statistical data during the lifetime of the mission. For example, to detect a solar flare, in which case the counting will be higher, or in case of a single event effect affecting a component. The counter used has a USB interface, with approximately 25 counts per minute for background lectures [18]. It can detect alpha, gamma and beta particles, though it can’t distinguish among them.
1.4.4. Technological Demonstrators

1.4.4.1. CubeCat-1 Cellsats

For the power subsystem design, the team has chosen the Spectrolab dual-junction GaAs cells [20], since they have filed in successful missions, and have the highest efficiency available on the market (≥26%).

But in one of the faces, a solar array of cells designed and built in the Electronics Department clean room [21]. This array has been developed completely by students of the team. Tests performed have shown a lower efficiency than the commercial ones (in 13%), testing them in orbit will give the final results and the degradation with radiation. A characterization has been made by a member of the group, and it can be find in his report [21].

1.4.4.2. LCD shutter

The amount of power received at the satellite from the Sun is much greater when in orbit than in the Earth’s surface. This radiation effect could damage the CCD of the camera; therefore a shutter system using a LCD has been designed.

The shutter is an LCD that, when powered up, becomes transparent, allowing the camera to take the shoot and when powered off, protecting the lens by going dark. This system has already been tested.

Though while doing tests with the camera and the solar simulator, the camera hasn’t suffered malfunction after being exposed, still demonstrating the behavior of a LCD in Space will be an additional scientific data to be tried out.

1.4.4.3. Peltier Cells

The Peltier effect shows that an electrical current will produce heating or cooling at the junction of two dissimilar semiconductors, depending on the direction of the current flow, it could produce heat or freeze. The heat absorbed or created at the
junction is proportional to the electrical current. The proportionality constant is known as the Peltier coefficient. Peltier Cells take advantage of this effect.

The main idea of the experiment is to take advantage of the temperature differences in the Cubesat faces exposed to Sun and dark and measure the current generated due these differences. This is also a part of an energy harvesting experiment.

Since the Peltier cell is small, the surface capturing the radiation is not big enough to produce a notable current, therefore the cell will be connected to the bottom solar panel of the CubeCat-1 to be able to capture the most energy possible of this area (100 cm²).

In the picture below, we can see one of the setups made to characterize the behavior of the cell inside the vacuum chamber attached to the solar panel; the solar simulator was placed in front.

![Solar panels in the vacuum chamber during a setup to characterize the peltier cell in the CubeCat-1 laboratory.](image)

**Figure 1.18** Solar panels in the vacuum chamber during a setup to characterize the peltier cell in the CubeCat-1 laboratory.

### 1.4.4.4. Resonant Inductive Coupling experiment

Electromagnetic Wireless Power Transfer consists of the transmission of electric energy through electromagnetic fields. This transfer can be made using electromagnetic induction as demonstrated by Tesla, the principle of operation of this system is a magnetic inductive coupling between a pair of resonant coils, the system is known as Resonant Inductive Coupling.

The experiment aims at analyzing the performance of these systems in two different scenarios: in-space wireless power transfer for fractionated space crafts (modular network of satellites interacting wireless and cooperatively) and miniaturized active energy harvesting for sensor networks [22].
The CubeCat-1 will allow testing the system in-orbit, observing the effects of near-field plasma in the vicinity of the antennas, and to characterize it in space.

1.4.4.5. New transistor types

New materials with excellent optical, mechanical, and electrical properties can be used in transistors to overcome some of the problems faced by transistors made of conventional semiconductor materials. They are not only smaller, but predictably faster.

Many researchers are currently engaged in developing the hardware underlying future nano devices. Including Nano-antennas (terahertz band in the short range), Nano-transceivers and Nano-processors, still in their very early stage.

CubeCat-1 will allow to test a new type of transistors in-orbit and to characterize their behavior for future applications in space.

1.5. Goals of this MSc. thesis

This M.Sc. thesis aims to provide a description of the CubeCat-1 communications subsystem (Space and Ground segments). The main objective is to show the implementation in software of the link, the hardware architecture description, the problems encountered, and to provide technical documentation for future members of the team about the work done so far to make it easy to modify, test or improve. The main objectives of the M.Sc. thesis are:

- Identify the system requirements
- Describe the hardware architecture
- Perform the link budget calculations to define the system settings.
- Implement a software terminal network controller handling AX.25 format data packets in the Cubesat micro-controller.
- Ground segment: Ability to communicate with CubeCat-1 and to be integrated with the GENSO network.

This report is organized as follows:

Chapter 1 has provided an introduction to the Cubesat concepts, some necessary communications concepts, and the CubeCat-1 project.

Chapter 2 will identifies the requirements for the communication system.

Chapter 3 details the system hardware architecture.

Chapter 4 a link budget analysis of the space segment is calculated.

Chapter 5 details the software development and the tests performed.
Chapter 6 gives a discussion of the lessons learned, future works and recommendations.

Chapter 7 gives the conclusions.

The Appendices contain the codes, manuals and the relevant theory to understand the terms and principles discussed in this master thesis.
Chapter 2
SYSTEM REQUIREMENTS

2.1. Functionality

The CubeCat-1 mission would be useless if it doesn't have the ability to communicate with the Earth. Three primary functions must be guaranteed:

Transmit a tracking signal (a beacon): This is an automated and periodic signal sent from the satellite that allows the ground stations to follow the position of the satellite. The beacon also sends vital basic information of the status of the satellite subsystems, which in the event of losing the ability to send commands to the Cubesat, will keep sending status information of the Cubesat.

Download telemetry data to the Ground Station and receive commands from the Ground Station: This is called establishing a two ways data link: the uplink (from the Ground station to the Cubesat) and downlink (from the Cubesat to Earth).

2.2. General constraints

The satellite orbit determines the schedule of passes over the Ground Station. Cubesats are typically launched in Low Earth Orbits (between 300 and 700 Km above Earth surface) so that the time mission goes no longer than 25 years (a requirement of the space regulations to comply with the debris policies and part of the Cubesat Standard).

Low Earth Orbits (LEO) are characterized then by its short range and high orbital velocity.

The communications window is the amount of time the satellite is visible from the Ground station, since the acquisition of the signal is available until it is lost.

Based on the simulations performed with STK (Satellite Tool Kit) [5] the expected communication window is of approximately 10 mins and a number of passes per day around 4-6.

The following figure shows a simulation made in the STK propagator by the CubeCat-1 team. In violet, the passes over the Ground Station in the UPC.

The orbit height was set to 650 km, the inclination of 98º, therefore the orbital period is 97 min, and it performs 14.7 revolutions per day.
Another constraint is power. The power subsystem is able to collect a maximum average of 2 W. However this has to be shared with the other subsystems, therefore the power should be kept to the minimum possible to establish a reliable link. Finally, and no less important, electronics should be chosen so that they have a wide range of temperature operation and be radiation tolerant. Radiation can cause Single Event Upset (SEU) or software corruption and Single Event Latch-ups (SEL), electric shortcut caused by high energy particles [23].

### 2.3. GENSO Recommended Radio Configuration

The GENSO approach was described in Chapter one, in order to be part of the network, some standard is required and/or recommended. The CubeCat-1 team has considered being part of the network since the conception of the mission, which is how now a day, the group owns its own ground station compatible with the GENSO specifications.

The requirements are:

**-Operating in the VHF and/or UHF amateur bands** allocation for satellite service, this is a recommendation. Since most of the ground stations part of the network have these hardware capabilities. The capacity of each user will determine the capability of the network. The benefits of using the amateur bands have already been explained in Chapter 1.

**Recommendations:**
-**Use the packet protocol AX.25** and the KISS host protocol. This protocol will be discussed in detail in the Chapter 5.

-**Data rates of 1200 bps with AFSK and/or 9600 bps with FSK**

Most amateur TNC’s provide an internal modem for this speed, the two-tone AFSK audio spectrum suits unmodified voice band radios comfortably (audio tones used are from Bell 202 standard, tones frequencies 1200 and 2200 Hz). The 1200 bps TNC’s were widely popularized by their simplicity and low bandwidth required in the 90’s already [24], even though higher speed modems appeared in the market later, this data rate has continued being the most used in the amateur world.

Data rated 9600 kbps was made popular thanks to the G3RUH FSK modem with a +/- 3 kHz shift over FM modulation. G3RUH is an adaptation through digital filtering to make an AFSK modulator able to send a higher bit rate not exceeding the 15 kHz Bandwidth allowed in the amateur band [25]. Although these modes are not mandatory, using them will make the system accessible to more stations.

A typical GENSO Ground Station will have two antennas, UHF and VHF, rotors to track the satellites, a radio transceiver with FM, SSB and CW capabilities, a TNC working at least in the two modes mentioned, and a computer interfaced.
2.4. Frequency Bands, maximum power and bandwidth allowed

Meeting with the International Amateur Radio Union (IARU), Spanish Amateur Radio Union (URE) regulations, the frequency bands designated for satellite communications are shown in the following table along with the maximum bandwidth allowed.

Table 2.1 Maximum power and bandwidths allowed in amateur bands [26] [27].

<table>
<thead>
<tr>
<th>BAND</th>
<th>Assigend Frequency</th>
<th>Max. Bandwidth (Khz)</th>
<th>Max. Power (W)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VHF</td>
<td>145.806-146</td>
<td>12</td>
<td>600</td>
</tr>
<tr>
<td>UHF</td>
<td>435-438</td>
<td>20</td>
<td>200</td>
</tr>
<tr>
<td>SHF</td>
<td>1260-1270</td>
<td>Not specified</td>
<td>10+30dBW PIRE</td>
</tr>
<tr>
<td>S</td>
<td>2400-2450</td>
<td>Not specified</td>
<td>10+30dBW PIRE</td>
</tr>
</tbody>
</table>
2.5. Cubesat Standard Regulations

CubeSats shall meet certain requirements pertaining to integration and operation to meet legal obligations and ensure the safety of other CubeSats.

- CubeSats with batteries shall have the capability to receive a transmitter shutdown command, as per Federal Communications Commission (FCC) regulation.

- All deployables such as booms, antennas, and solar panels shall wait to deploy a minimum of 30 minutes after the CubeSat's deployment switches are activated from P-POD ejection.

- RF transmitters greater than 1 mW shall wait to transmit a minimum of 30 minutes after the CubeSat's deployment switches are activated from P-POD ejection.

- Operators shall obtain and provide documentation of proper licenses for use of frequencies coordinated with the IARU.

- For amateur frequency use, this requires proof of frequency coordination by the International Amateur Radio Union (IARU).

- The orbital decay lifetime of the CubeSats shall be less than 25 years after end of mission life. This is guaranteed by inserting the Cubesat in a LEO orbit.

- Developers shall obtain and provide documentation of approval of an orbital debris mitigation plan from the FCC or local agency.

According to ITU-R regulations the system shall also be able to cease any radio transmissions through commands from the ground station (see Radio Regulations, Vol.1, Art.22, Sec.I, pt. 22.1).

2.6. Doppler shift

The Doppler Effect is a physical effect that produces a frequency shift in the signal received by the receiver due to the movement of the transmitter relative to it. Satellite is orbiting the Earth moving at a relative high speed. This can cause the satellite to transmit out of the assigned bandwidth, if this effect is not taken into account.

Doppler shift is related directly with the period of the orbit, the speed of the satellite and the speed relative to Earth. It can be expressed by the following equation:

\[ \Delta f = f t \cdot \left( \frac{v}{c} \right) \]
System requirements

(Equation 2.1)

Where \( f_t \) is the frequency of transmission, \( v \) is the speed of the satellite relative to the ground station, and \( c \) is the speed of light.

The relative speed can be found by:

\[
v = v_s \cdot \cos(\theta) = v_s \cdot \left( \frac{R}{R+h} \right)
\]

(Equation 2.2)

Where \( v_s \) is the satellite speed, \( R \) is the Earth radius, and \( h \) the height of the orbit relative to the Earth surface.

The Doppler shift in uplink is three times lower at 145MHz, than at 435MHz.

The higher the frequency or the speed is, the more noticeable the effect will be.

Using Equation 2.3, the Doppler effect is +/-2.5 KHz in VHF, and +/- 10 KHz in UHF.

If the Cubesat would transmit at 145.950 MHz, and we were in the worst case (satellite passing at the zenith) when appearing in the horizon, the frequency will be 145.952.5 MHz, when it is at zenith, 145.950 MHz, and when it is disappearing below the horizon, in 145.987.5 MHz. Following the same analogy, if the satellite would transmit in 435.850 MHz the effect would be noticeable between 435.860 MHz and 435.840 MHz.

Transmitter and receiver will need to adjust their frequency either periodically or continuously to compensate for the Doppler shift. Even though, as it can be seen in VHF the Doppler Effect is not critical.

One method of dealing with Doppler shift is to increase the bandwidth of the receiver filters on both ends of the link so that even with the Doppler shift, the modulated transmitted signal is always contained within the receiver bandwidth.

Doppler correction will be performed by the ground station.

2.7. Derived Requirements

-The frequency chosen for the uplink and downlink should be either the UHF (435-438 MHz) or VHF (145.806-146 MHz) amateur bands assigned to satellite operations.

-The system shall provide an uplink and downlink of data rate of at least 1200bps.

-The modulation supported should be AFSK, FSK or GFSK.

-The data packet protocol AX.25 should be supported in receiver and transmitter.

-The beacon should be continuously operating.
- The system shall maintain a link margin of at least 6 dB at elevations greater than 10° [26].

- The bit error rate shall be less than 10^{-5} [30].

- To keep a low BER, an optional capability of fault packages retransmission should be available.

- The system shall preferably not consume more than 1 W of peak power when transmitting.

- The system shall use the minimum power in standby.

- Doppler shifts should be compensated in both the Ground Station and the Cubesat transceiver.

- Design (software and hardware) should be kept as simple as possible.

- In order to mitigate the fact that contact time is short, joining GENSO network would allow having multiple ground stations to track the satellite and download more data than just passing over a single station.

2.8. Design decisions

- Use separate bands for Beacon and data transmitter.

- Use whether UHF or VHF for the downlink, the choice with the best performance in the link budget.

- Choose a transceiver on board which at least two options of modulation, AFSK or FSK and low power consumption. The CC1101 transceiver from Texas was chosen for this, its parameters will be used in the link budget, in the Chapter 4, more details on its characteristics and the reasons why it was chosen will be provided.

- Use the AX.25 packet radio protocol with a simple handshaking layer and a retransmission optional feature in case of errors.

- Implement a simple beacon (Morse code, OOK).

- Join GENSO network as a way of redundancy.

- Implement a software TNC to decode AX.25 on board.

- To keep the communications simple and reliable, use short data bursts.

- Apply redundant hardware in the communications hardware to guarantee communications. Separating beacon from data link for example.
System requirements

- Implement a software system, if possible as independent as possible from the main on board computer for safety, in case of OBC failure.

- Implement by software, low power consume modes.

- Implement by software a simple interface in the Ground Station to receive the data sent by satellite.
Chapter 3 HARDWARE SYSTEM ARCHITECTURE

3.1. System Architecture proposed

The On Board hardware design model proposed by the team is shown in the following figure:

The board

![Diagram of the communication subsystem proposed by Mr. Roger Olivé](image)

**Figure 3.1** Communications subsystem diagram proposed (by Mr. Roger Olivé)

The proposed architecture meets all the requirements, there are two separated links for data and beacon to guarantee reliability. As a main data transceiver the CC1101 transceiver controlled by a MSP430 Texas microchip has been selected.

As a redundant transceiver, a MHX425 radio, to be used in case of total failure of the main transceiver, since the system is encoded and the signal can only be decoded by another same transceiver.

The data transceivers will be explained in detail in section 3.3.

3.1.1. Beacon

A high power transmitter module using ASK has been chosen for its simplicity. This transmitter operates in the band of 433.92 MHz and uses ASK modulation at a rate up to 8 kbps and have an output power of 14 dBm.
The team has decided to have the beacon working on VHF instead of UHF so it can be independent of the main transceiver by having a dedicated antenna and therefore making the system more reliable. To do so, a mixer is used to change the output frequency.

The mixer chosen is the LT5512, it has the capability of being programmable and it has a noise figure of around 10 dB and with a conversion gain of 1 dB.

As a last stage, before the UHF antenna, an amplifier (MGA30889) of 15 dB of gain and 2 dB of Noise Figure.

<table>
<thead>
<tr>
<th>Pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GND</td>
</tr>
<tr>
<td>2</td>
<td>Data In</td>
</tr>
<tr>
<td>3</td>
<td>Vcc</td>
</tr>
<tr>
<td>4</td>
<td>ANT</td>
</tr>
</tbody>
</table>

![Figure 3.2 ASK Transmitter Module TWS-BS-3 [31]](image)

### 3.1.2. High Power Amplifier

The amplifier chose for the downlink in the UHF band is the RF6886, which operates in frequencies from 100 to 1000 MHz. It has more than 50% efficiency and an output of 34.5 dBm. It can operate in a wide range of temperatures (from -40 to +85ºC). The maximum input allowed is 10 dBm, this is important when selecting the transceiver output power.

### 3.1.3. Low Noise Amplifier

This election for the uplink amplifier was the RF2374 of low power consumption, high frequency range, and a gain of 20 dB. The noise figure is of 2dB.

The PCB design has been performed by Mr. Roger Olivé and are not included in this thesis.

### 3.1.4. On board antennas

The choice made was to use half wave crossed-dipoles. The material used is metric tape since it has given very good performance results and is easy to deploy with a simple mechanism of threads burned by a resistance. The design of the antennas
has been performed by Carlos Garcia Del Castillo, and is further explained in his Final Project report.

### 3.2. CubeCat-1 Ground Station

As recommended by the GENSO specifications for hardware equipment, the Ground Station installed in the UPC Campus Nord (an ISIS solution) has the following components:

![Figure 3.3 GENSO compatible ISIS ground Station installed in the UPC campus.](image)

#### 3.2.1. Radio IC-910H

The radio used is an IC-910H and it is capable of detecting FM, SSB and CW modes.

The data interface at the radio provides two audio lines for transmitting and receiving together with a PTT line to key the radio.

Its main characteristics are [32]:

**Band range:**

VHF 144-146 MHz
Hardware system architecture

UHF 430-450 MHz

**Downlink capabilities:**

**Data rate and mode**
- 1200bps AFSK in SSB/FM mode
- 1200bps BPSK in SSB/FM mode
- 9600bps G3RUH FSK in FM mode

**Sensitivity**
- In SSB, CW (10 dB S/N) Less than 0.11uV
- FM (12 dB SINAD) Less than 0.18 uV

**Squelch sensitivity threshold**
- SSB, CW Less than 1.0uV
- FM Less than 0.18uV

**Selectivity (Filter Bandwidth)**
- SSB, CW 2.3 kHz/-6dB
- 4.2 kHz/-60dB
- FM 15 kHz/-6dB
- 30 KHz/-60dB
- FM-N 6.0 kHz/-6dB
- 18.0 kHz/-36dB

**Receive Incremental Tuning variable range**
- 144/430 MHz SSB, CW +/-1.0 kHz
- FM +/-5.0 kHz

**Uplink capabilities**
- 1200 bps AFSK
- 9600 bps G3RUH FSK

**Output Power**
- 144 MHz 5-100 W
430 MHz 5-75 W

**Modulation System**

SSB  Balanced modulation

FM  Variable reactant modulation

**Spurious emission**

144/430 MHz More than 60dB

**Antennas**

UHF  14.1 dBdc, VHF 10.2 dBdc

Noise Figures: VHF/UHF 4.2 dB

### 3.2.2. TNC's

In a packed-based communication network, a TNC works like a modem and it is responsible for coding any data to an AX.25 frame, and provides a raw bit stream for the radio to modulate and transmit. In reception, the TNC is responsible for decoding the received bit stream into an AX.25 frame.

The decoded data is called a High level data link control packet.

![Diagram of TNC function](image)

**Figure 3.4** TNC function is to encode and decode AX.25 frames.

There are two TNC's implemented by hardware, and connected to the data output of the radio.
**TNC 7 Multi**

Has a build in 1200/9600bps modem, and high speed serial interfaces (USB and RS232). Modulations supported are: FSK and AFSK in KISS/SMACK mode.

The TNC has 9 modes presets but from these we are interested in three only, since the other modes are for higher data rates available by adding an optional modem. These three modes are 1200 AFSK, 9600 FSK and Flexnet which is a TCP/IP protocol over AX.25 which allows switching from one mode to the other one via software.

It also important here, while developing a software to take into account the transmitter key up delay of 150 ms specified in the manual.

![TNC7 Multi](image)

**Figure 3.5** TNC7 multi front view. The switch selects between the preset modes and LED’s show the actual status.

It is easy to configure and the LED’s in the front panel help visualizing the function being performed. PTT led flash when transmitting, DCD led flash while receiving data at rate selected, CON led flash when the TNC is transmitting data to the PC and STA on the other way around.

Finally a TNC31, which is a dedicated 9600bps modem, communicates the data to the PC data rate of 19200, no parity, 8bits via a RS232 connection.

### 3.2.3. Geographic location considerations

Due to the geographic location of the Ground Station, 41.39ºN, 2.11ºW, the elevation angle limit for LOS is approximately 10º, since a mountain in front blocks the line of sight for lower elevation angles.

### 3.3. Data transceiver

#### 3.3.1. Micro controller Unit MSP430

The philosophy was to consider the communication system as independent as possible from the other subsystems, therefore to have a dedicated microcontroller...
The MSP430 was chosen because it is one of the MCU's with least power consumption (1.1 uA in standby mode, and 330 uA in active mode) and it has flown in Space successfully already in other Cubesats.

The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 6 μs.

It has two built-in 16-bit timers, a fast 12-bit A/D converter, dual 12-bit D/A converter, one or two universal serial synchronous/asynchronous communication interfaces (USART), I2C, DMA, SPI and 48 I/O pins.

All MSP430 microcontrollers include an Embedded Emulation Module (EEM) allowing advanced debugging and programming through easy to use development tools.

3.3.2. Transceiver Chipcon CC1101

The transceiver system is built around the Texas CC1101 transceiver. The chip is a low-power RF transceiver with the ability to operate in either the 300-348 MHz, 387-464 MHz or 779-928 MHz bands. The baseband modem supports 2-FSK, 2-GFSK, MSK, OOK and Flexible ASK Shaping with a configurable data rate up to 600 kbps, an output power selectable up to +12 dBm and a high sensitivity (up to -116 dBm in 433 MHz). Its Automatic Frequency Compensation (AFC) compensates Doppler shift in reception.

The following figure shows a block diagram of the radio architecture,

*Figure 3.6 CC1101 Radio block diagram [33]*
The received RF signal is amplified by a low-noise amplifier (LNA) and down-converted in quadrature to the intermediate frequency (IF). Demodulation and bit/packet synchronization are performed digitally.

The transmitter part is based on direct synthesis of the RF frequency. The 26-MHz crystal oscillator generates the reference frequency for the synthesizer, as well as clocks for the ADC and the digital part.

The transceiver can be operated in two modes. The first mode is through direct serial control receiving and transmitting a raw bit stream data. This mode allows for a high degree of flexibility of packet handling since the MCU can be programmed to process the received data in any number of ways. The drawbacks of this approach are that the MCU will use much of its resources to manage packet handling since this is a continuous operation. The second mode is to use the built-in functionality of the transceiver chip. This approach allows the system to use the built-in packet handler to receive and transmit data.

3.3.3. Development board EMF430F6137RF900

The development board used scheme is show in the following diagram, it is a MSP430F1612 MCU interfaced with a CC1101 transceiver. It provides a useful integration between the microcontroller, its peripherals, software, and the RF transceiver.

![Figure 3.7 Development board block diagram, shows the integration of the MCU with the transceiver chip [33]]
3.3.4. MHX425 Radio

The MHX425 is a 400 to 450 MHz UHF Frequency Hopping Wireless Modem. It has adjustable hopping patterns, a high sensitivity (-115 dBm), a system gain of 140dB and output power of up to 1 W (437.325 MHz, GFSK modulation) and a data rate up to 19.2 kbps [6].

The radio can be configured either as a slave or master, synchronization is done and then the link is established, the hopping is determined on a network address basis. The radio is configurable via short commands that modify the registers.

Frequency hopping spread spectrum (FHSS) makes the system more robust with respect to interference from other systems operating in the same frequency band.

![Figure 3.8 MHX 425 radio block diagram](image)

A general description of the main characteristics of the hardware chosen has been shown in this chapter, the parameters presented will determine the quality of the communication link which is analyzed in the next chapter.
Chapter 4 **LINK BUDGET**

This chapter will describe the link budget analysis, discusses the key parameters used and presents the results provided by the calculations. The link budget analysis determines whether it is possible to establish a communication link with given systems parameters (power, frequency, data rate and the bandwidth).

The AMSAT / IARU Link Model System excel spreadsheet was used to calculate the link budget in detail. The data link BER was set to $10^{-5}$, minimum requirement for a reliable link.

![Link budget analysis diagram](image)

*Figure 4.1* The link budget analysis takes into account all the gains and losses from the transmitter to the receiver end.

### 4.1. Link main characteristic parameters

We describe very briefly the main relationship between the parameters involved in the link budget calculations.

The power arriving to the receptor can be written expressed in dB as:

$$ P_r = P_t + G_t + G_r - \left(\frac{4\pi R}{\lambda}\right)^2 \text{ [dBW]} $$

**Equation 4.1**
Where:

- $P$: the power transmitted
- $G_t$: the transmitting antenna gain
- $G_r$: the receiving antenna gain
- $\lambda$: wavelength

The product of the power and the gain of the antenna in the transmitter is known as Effective Isotropic Radiated Power (EIRP).

$$EIRP = P_t + G_t \ [\text{dBW}]$$  \hspace{1cm} \text{Equation 4.2}

and the term,

$$\left(\frac{4\pi R}{\lambda}\right)^2 = \text{Free Space path loss}$$  \hspace{1cm} \text{Equation 4.3}

in addition to this lost, there are also other losses due to atmospheric conditions, line losses, or connectors in the transmitter and the receiver.

Thermal Noise or white noise is evenly distributed over the full band, it is proportional to the bandwidth and system’s temperature, and is given by:

$$P_n = k \cdot T_n \cdot B_n$$  \hspace{1cm} \text{Equation 4.4}

where $k$ is the Boltzmann constant, $T_n$ the physical temperature in Kelvins and $B_n$ the noise bandwidth ([Hz]).

The noise figure (NF) is the rate of the signal degradation through a circuit and is given as the relationship between the signal to noise ratio at the input and at the output.

$$NF = S \frac{N_{in}}{N_{out}}$$  \hspace{1cm} \text{Equation 4.5}

The signal to noise ratio is the ratio between the power of the information carrying the signal and the power of the noise present in the channel.

It can be expressed as:

$$SNR[dB] = P_t + G_t + G_r - Lfs - L - P_n$$  \hspace{1cm} \text{Equation 4.6}

This value will determine the threshold of the system, it will indicate whether or not the information from the received signal can be extracted or not.

In digital communications, the figure of merit is the energy per bit to noise power spectral density ratio $\frac{E_b}{N_0}$, it is a normalized version of the SNR. This factor allows comparing between different modulation schemes, such as the ones in figures 1.10, 1.11 and 1.12.
Link Budget

The relationship between the signal to noise ratio and the Eb/No is:

\[ SNR_{[dB]} = \frac{E_b}{N_0} + 10 \cdot \log(f_b) - 10 \cdot \log(B) \quad \text{Equation 4.7} \]

were \( f_b \) is the frequency of symbol, and \( B \) the bandwidth.

The bit error rate (BER) is directly related with the \( \frac{E_b}{N_0} \). To achieve a lower BER more energy per bit should be sent, therefore a higher \( \frac{E_b}{N_0} \) requirement is imposed.

4.2. Parameters of the system

In order to perform the calculations, the main parameters of the system are summarized.

<table>
<thead>
<tr>
<th>Table 4.1 Ground Station antennas parameters.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>UHF Antenna</strong></td>
</tr>
<tr>
<td>Band [Mhz]</td>
</tr>
<tr>
<td>Gain [dBd]</td>
</tr>
<tr>
<td>Front to back [dB]</td>
</tr>
<tr>
<td>Impedance [Ohms]</td>
</tr>
<tr>
<td>Noise Figure [dB]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 4.2 Ground Station cable losses.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Cable</strong></td>
</tr>
<tr>
<td>VHF EE-15</td>
</tr>
<tr>
<td>UHF EE-15</td>
</tr>
<tr>
<td>VHF S-B RG58</td>
</tr>
<tr>
<td>POL VHF RG58</td>
</tr>
<tr>
<td>VHF AC-7</td>
</tr>
<tr>
<td>UHF AC-7</td>
</tr>
<tr>
<td>S-band AC-7</td>
</tr>
<tr>
<td>POL VHF RG58</td>
</tr>
<tr>
<td>AZ Rotor</td>
</tr>
</tbody>
</table>

The following table resumes the orbital parameters used, as the ones used for the simulations in STK, accorded by the Team.
Table 4.3 Orbital parameters.

<table>
<thead>
<tr>
<th>Type</th>
<th>Circular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean Orbit Altitude</td>
<td>650Km</td>
</tr>
<tr>
<td>Inclination</td>
<td>98º</td>
</tr>
<tr>
<td>Period</td>
<td>97.73min</td>
</tr>
<tr>
<td>Uplink Frequency</td>
<td>436Mhz</td>
</tr>
<tr>
<td>Downlink frequency</td>
<td>436.25Mhz</td>
</tr>
</tbody>
</table>

As mentioned before, the $\frac{E_b}{N_0}$ determines the requirements of the system to meet a minimum BER. An optimum link is recommended to have a BER smaller than $10^{-4}$, optimally of $10^{-5}$.

Table 4.4 summarizes the required $\frac{E_b}{N_0}$ for the modulation schemes supported by our transceiver (CC1101).

**Table 4.4** $\frac{E_b}{N_0}$ required for the modulation schemes supported by the transceiver.

<table>
<thead>
<tr>
<th>Modulation</th>
<th>$\frac{E_b}{N_0}$ [dB]</th>
<th>$\frac{E_b}{N_0}$ [dB]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coherent 2FSK</td>
<td>10.4</td>
<td>12</td>
</tr>
<tr>
<td>Non-coherent 2FSK</td>
<td>12.3</td>
<td>13.5</td>
</tr>
<tr>
<td>Coherent 4FSK</td>
<td>9</td>
<td>11</td>
</tr>
<tr>
<td>Non-coherent 4FSK</td>
<td>9.8</td>
<td>10.8</td>
</tr>
<tr>
<td>GFSK</td>
<td>13.2</td>
<td>14.5</td>
</tr>
</tbody>
</table>

4.3. Downlink calculations

4.3.1. Main transceiver (CC1101)

The link budget calculations for the downlink, were done assuming a frequency of 436.250 MHz, and a time window of 10 min.

AMSAT recommends considering a link closed when it has a margin greater than 6 dB. Lower than this, it is considered a marginal link, the link is closed but there are more chances to have errors during the link. Table 4.5 compares the link performance for the different modulations supported by the transceiver and the two rates supported by the Ground Station and the amateur network.
Table 4.5 Comparison between performance over time window between the different modulation schemes and data rate.

<table>
<thead>
<tr>
<th></th>
<th>Optimum Link</th>
<th>Link Closes</th>
<th>Marginal Link</th>
<th>No Link</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Over 10dB</td>
<td>Over 6dB</td>
<td>Less than 6dB</td>
<td>Less than 0 dB</td>
</tr>
<tr>
<td></td>
<td>% window time</td>
<td>% window time</td>
<td>% window time</td>
<td>% window time</td>
</tr>
<tr>
<td>GFSK 9600bps</td>
<td>0</td>
<td>19</td>
<td>53</td>
<td>28</td>
</tr>
<tr>
<td>Non-Coherent 2FSK 9600</td>
<td>0</td>
<td>50</td>
<td>40</td>
<td>10</td>
</tr>
<tr>
<td>MSK 9600</td>
<td>0</td>
<td>71</td>
<td>28</td>
<td>1</td>
</tr>
<tr>
<td>Non-Coherent 4FSK 9600</td>
<td>19</td>
<td>56</td>
<td>25</td>
<td>0</td>
</tr>
</tbody>
</table>

It is also shown the performance using a FEC code, this feature is available in the transceiver, though, it requires the implementation of a FEC decoder on the Ground Station, the performance increases greatly, but the data rate is half of the ones shown. Therefore shown for inter-comparison purposes only.

10° of elevation has been selected as the starting point for the geographic limitations mentioned. On figures 4.3, 4.4 and 4.5 can be graphically seen the performance of the best options for 9600 bps and 1200 bps.
Figure 4.2 Link margin in dB to guarantee a BER of $10^{-5}$ using MSK at 9600bps during a single pass, the optimum margin is achieved ≥70% of the total duration of a single pass.

For 9600 bps, even MSK offers an optimum link 70% of the total pass time, the compatibility with the network of ground stations is reduced, as mentioned in the previous chapters, the recommended configuration is the use of FSK or AFSK. However, this mode has performed well with our receiver so it is considered as one option, the second option is the 2FSK mode, although only 50% of the time the optimum parameters are available.

If the bit rate is set to 1200 bps, the performance of the four modes is noticeable better. For simplicity, 2FSK is selected.

Note that the recommendation to work at 1200bps, was to use AFSK, though, the CC1100 is not designed to transmit AFSK.

AFSK can be interpreted as a FSK with a 1Khz frequency separation (separation between the two tones used in AFSK, 1200hHz and 2200Hz). In the ground station the radio receiving in SSB(Single side band) will map the FSK transmission to audio tones very close to the two AFSK tones, making it possible to decode AX.25. But we also know that by Carlson Rule, for a successful reception, the frequency separation of the received signal must be greater than or equal to twice the bit rate.
Link Budget

Figure 4.3 Link margin in dB to guarantee a BER of $10^{-5}$ using 2FSK at 9600bps during a single pass.

Thus, with a separation of 1 kHz, to transmit a bit rate greater than 500 bits per second it is not guaranteed. There are two options, either increase the frequency separation (no longer AFSK) or reducing the bit rate.

Figure 4.4 Link margin in dB to guarantee a BER of $10^{-5}$ using 2FSK at 1200bps during a single pass.
AX.25 doesn’t impose a requirement in the physical layer, several missions (QuakeSat, Cute-1, UNICubeSAT) have operated at 9600 bps data rates using FSK and AX.25 and MaSat-1 for example, launched with VEGA, uses 2FSK at 1200 bps.

Therefore for the downlink the transceiver will be programmed to work between 9600 bps/2FSK, 9600 bps/MSK and 1200 bps/2FSK.

4.3.2. Redundant transceiver (MHX425)

The downlink parameters and results are shown in the table 4.6, this transceiver has flown successfully in the ITUPsat-1 [34] (September, 2009) using the same parameters.

Table 4.6 Downlink system performance for the worst case, 10º of elevation.

<table>
<thead>
<tr>
<th>Downlink Budget MHX425</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Cubesat Transmitter Power Output</td>
<td>30 dBm / 1W</td>
</tr>
<tr>
<td>Cubesat total transmission Line losses</td>
<td>1.9 dB</td>
</tr>
<tr>
<td>Cubesat Antenna Gain</td>
<td>2.2 dB</td>
</tr>
<tr>
<td>Cubesat pointing losses</td>
<td>0.3 dB</td>
</tr>
<tr>
<td>Polarization Losses</td>
<td>0.2 dB</td>
</tr>
<tr>
<td>Free Path loss (10º elevation)</td>
<td>140 dB</td>
</tr>
<tr>
<td>Atmospheric loss</td>
<td>1.1 dB</td>
</tr>
<tr>
<td>Ionospheric loss</td>
<td>0.4 dB</td>
</tr>
<tr>
<td>GS Pointing Losses</td>
<td>0.7 dB</td>
</tr>
<tr>
<td>GS Antenna Gain</td>
<td>12 dB</td>
</tr>
<tr>
<td>Total transmission line losses</td>
<td>5.1 dB</td>
</tr>
<tr>
<td>System Sensitivity</td>
<td>-115 dB</td>
</tr>
<tr>
<td>System Gain</td>
<td>-105.5 dB</td>
</tr>
<tr>
<td>Link Margin</td>
<td>9.5 dB</td>
</tr>
</tbody>
</table>

Link margin is optimal even in the edge of the acquisition of signal (AOS).

4.4. Uplink calculations

Table 4.7 Uplink system performance for the worst case, 10º of elevation.

<table>
<thead>
<tr>
<th>Uplink Budget MHX425</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>GS Transmitter Power Output</td>
<td>49 dBm / 75W</td>
</tr>
<tr>
<td>GS Total Transmission Line Losses</td>
<td>5.1 dB</td>
</tr>
<tr>
<td>GS Antenna Gain</td>
<td>12 dB</td>
</tr>
<tr>
<td>GS Pointing Losses</td>
<td>0.7 dB</td>
</tr>
<tr>
<td>Polarization Losses</td>
<td>0.2 dB</td>
</tr>
<tr>
<td>Free Path loss (10º elevation)</td>
<td>140 dB</td>
</tr>
<tr>
<td>Atmospheric loss</td>
<td>1.1 dB</td>
</tr>
<tr>
<td>Ionospheric loss</td>
<td>0.4 dB</td>
</tr>
<tr>
<td>Cubesat pointing loss</td>
<td>0.3 dB</td>
</tr>
<tr>
<td>Cubesat Antenna Gain</td>
<td>2.2 dB</td>
</tr>
<tr>
<td>Cubesat total transmission Line losses</td>
<td>0.3 dB</td>
</tr>
<tr>
<td>System Sensitivity</td>
<td>-115 dB</td>
</tr>
<tr>
<td>System Gain</td>
<td>-84.9 dB</td>
</tr>
<tr>
<td>Link Margin</td>
<td>30.1 dB</td>
</tr>
</tbody>
</table>
Figure 4.5 Uplink performance for the main transceiver at the margin elevation angle.
4.5. Beacon

The figure 4.6 summarizes the link budget for the Beacon downlink, as mentioned before the data rate is 8 kbps and modulated with ASK. The link closes with an optimal margin of 7.9 dB at 10° of elevation.

Figure 4.6 Downlink performance for the main transceiver at the margin elevation angle.
Link Budget

In this chapter the performance of the link has been calculated taking into account the hardware characteristics, it has been shown that the critical part of the link is the downlink (from the satellite to the Ground Station).

Even though it is critical, the calculations show that it is possible to use a data rate of 9600 bps using FSK or MSK modulations since the BER required can be achieved over 70% of the time during a single pass.

In the worst case, reducing the data rate to 1200 bps guarantees an optimum link the whole duration of the pass.
Chapter 5 SOFTWARE DEVELOPMENT

5.1. The data link layer

The communication system can be described by the layers model OSI. In our case we have three layers, the physical layer, the data link layer and the application layer.

The physical layer is in charge of transmitting raw bits over the channel, it has been described in the earlier chapters (hardware, modulation/demodulation, RF interface).

The data link layer protocol is the critical part for ensuring a reliable communication by providing frames to the raw bits in order to control and detect errors. The data link protocol should be compatible with the physical layer hardware.

As mentioned before the recommended protocol is the AX.25, used widely in the amateur network and most ground stations TNC's, it is important to mention as well that this protocol is independent of the existence of any upper layer.

5.2. The AX.25 Link Access Protocol

This protocol is based in the High Level data link control (HDLC) frames. It support amateur call names (as an extension of X.25), connected links, connection less links, half or full duplex support and error detection.

The protocol details can be found in [35]. It is rather more a description of the usage than a guide for implementation. It can work in a connection mode or a connectionless mode.

Though, AX.25 in connection oriented mode would shrink the info data throughput and due to the short window contact, sending many acknowledgments and having a handshaking preamble will make a not so efficient use of the pass, therefore we decide to implement a connection less mode.

Implementing a connection less mode uses a special frame for this operation, the Unnumbered Information (UI) frame (Table 5.2). In this type of operation, there are no requests for retransmissions. Without the handshaking activity of a point-to-point connection, collisions may occur but since it is not a full duplex link, the problem is inexistent.

Table 5.1 AX.25 UI frame format [35].

<table>
<thead>
<tr>
<th>Flag</th>
<th>Address</th>
<th>Control</th>
<th>PID</th>
<th>Info</th>
<th>FCS</th>
<th>Flag</th>
</tr>
</thead>
<tbody>
<tr>
<td>01111110</td>
<td>112/224 Bits</td>
<td>8/16 Bits</td>
<td>8 Bits</td>
<td>N*8 Bits</td>
<td>16 Bits</td>
<td>01111110</td>
</tr>
</tbody>
</table>
The Unnumbered Information frame contains PID and information fields and passes information along the link outside the normal information controls. This allows information fields to be exchanged on the link, bypassing flow control.

Every field is transmitted low-order bit first except the Frame Check Sequence. The flag delimits the frames; it is in hexadecimal a 7E.

The address field contains the amateur call-sign, it is important to notice that the amateur radio call-sign information is shifted one bit left. The control field for the link would be as in Table 5.3.

The control field in an unnumbered frame can be one or two octets, for an UI frame, the octet is 00000011.

Table 5.2 AX.25 UI Frame, 276 bytes.

<table>
<thead>
<tr>
<th>Bytes</th>
<th>Octet</th>
<th>ASCII</th>
<th>Hex Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Flag</td>
<td>0x7E</td>
<td></td>
</tr>
<tr>
<td>A1</td>
<td>E</td>
<td>0x45</td>
<td></td>
</tr>
<tr>
<td>A2</td>
<td>A</td>
<td>0x41</td>
<td></td>
</tr>
<tr>
<td>A3</td>
<td>3</td>
<td>0x33</td>
<td></td>
</tr>
<tr>
<td>A4</td>
<td>R</td>
<td>0x52</td>
<td></td>
</tr>
<tr>
<td>A5</td>
<td>C</td>
<td>0x43</td>
<td></td>
</tr>
<tr>
<td>A6</td>
<td>U</td>
<td>0x55</td>
<td></td>
</tr>
<tr>
<td>A7</td>
<td></td>
<td>0x60</td>
<td></td>
</tr>
<tr>
<td>A8</td>
<td>C</td>
<td>0x43</td>
<td></td>
</tr>
<tr>
<td>A9</td>
<td>U</td>
<td>0x55</td>
<td></td>
</tr>
<tr>
<td>A10</td>
<td>B</td>
<td>0x42</td>
<td></td>
</tr>
<tr>
<td>A11</td>
<td>E</td>
<td>0x45</td>
<td></td>
</tr>
<tr>
<td>A12</td>
<td>C</td>
<td>0x43</td>
<td></td>
</tr>
<tr>
<td>A13</td>
<td>A</td>
<td>0x41</td>
<td></td>
</tr>
<tr>
<td>A14</td>
<td></td>
<td>0x67</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>PID</td>
<td>0xF0</td>
<td></td>
</tr>
<tr>
<td>256</td>
<td>Info</td>
<td>xxxx</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>2</td>
<td>FCS</td>
<td>xxxx</td>
<td>xxxxxxxx</td>
</tr>
<tr>
<td>1</td>
<td>Flag</td>
<td>0x7E</td>
<td></td>
</tr>
</tbody>
</table>

The Protocol Identifier (PID) field indicates which layer 3 protocol is used, if any. For the not implemented case, it is F0, and an escape character is FF.

The information field default length is of 256 octets without the insertion of zeros, these info bits are passed transparently. The zero bit stuffing is done to avoid confusing the flags with the information bits, anywhere it is find a group of more than five contiguous “bits” a “0” is inserted after the 5th “1” bit.

The Frame Check Sequence (FCS) is a 16 bits number calculated in both sides of the link to check the integrity of the package. Only in this field, the most-significant bit is sent first.

A frame is said to be invalid when it has less than 136 bits including the flags, it is not bounded by flags or is not an integral number of octets.
Since the frame header and trailer fields has the distinctive bit pattern of 01111110, to avoid this pattern to exist anywhere else in the frame bit stuffing is used.

Anytime the transmitter detects five contiguous ones in a row that is not a part of the head nor tail flag, it will insert a zero. Likewise when the receiver detects five contiguous ones in a row, it will remove any preceding zero. Hence the bit stuffing is applied in all AX.25 frame fields but the head and tail flag.

5.3. The AX.25 implementation functions

To implement the AX.25 protocol, the GNU library “comSC” is used, it contains all the low-level routines for the connectionless mode of AX.25 communications.

5.3.1. Working Principle

For the data transmission, the data is first copied in memory. After having added the header, the checksum is calculated and appended. Finally, the frame is transmitted towards the frequency modulator.

For the data reception, the UI frame is captured at a digital input pin. When the stop flag is received, then the frame is verified by checking the destination address and the validity of the checksum. If the frame is valid, a flag is sent to notice the main program it was a valid frame (Figure 5.1).

![Figure 5.1 Flow diagram of the software proposed.](image)

The code to forward the data to the on board computer is not fixed, if the communication with the main data bus is serial, it is possible to use the same serial communication module as the one used in the MHX transceiver, opening a port,
Software development

writing, and closing. This will depend on the OBC side. Meanwhile, for debug purposes, a printf() function has been used to show the packets.

5.3.2. Functions Description
The functions used in the comSc are briefly described now.

void AX_start(void); This enables the Timer_B interrupts that are used for RX and TX. This should be called during the initialization of the microcontroller in the main file.

5.3.2.1. ax_recv

- void RX_setByteMax(unsigned int size); Sets the maximum number of data bytes in a frame. If you plan to use larger settings than 278, than the memory allocation of RX_Data needs to be modified. The received frame is not required to have the maximum length, but can be shorter.

- void RX_setPassAll(char passAll); When calling RX_setPassAll(1), than the data from all frames are forwarded to the higher layer, independently if the frame passes the validity check or not. This mode is useful for debugging or to bypass the problem when the receiver address is not known.

- void RX_packetReceived(void); This routine is called whenever a frame has been received and decapsulated. The higher layer routines should add a hook to be announced of the arrived data.

5.3.2.2. ax_send

- void TX_start(void); Starts the emission of a frame. The frame needs first to be prepared by TX_prepareFrame.

- void TX_prepareFrame(char *data, unsigned int length); Prepares the data and forms a valid AX.25. The flags are concatenated, the address, and the FCS is calculated and stored.

- void TX_delaySet(unsigned int a); this function delays the emission to give time to the TNC to process the frame.

- void TX_DACSet(unsigned int low, unsigned int high); Sets the values for the DAC to the values low and high. The range is between 0x0000 (0 Volt) and 0x0FFF (3.3 Volt).

- void TX_DACReset(void); Resets the values for the DAC to the default values.

- void TX_sync(void); sends a synchronization signal.

The code for ax_send and ax_recv can be find in the Annexes.
5.3.3. Configuring the Transceiver

The CC1101 can run in a serial synchronous mode, then the transceiver is transparent and the raw data bit stream is clocked in or out from the transceiver. In this mode the microcontroller and transceiver communicates over a simple two-wire serial bus, consisting of a clock and a data line.

The transceiver contains 49 8-bit registers used to configure it. The manufacturer of the device, Texas Instrument has developed a software program SmartRF Studio [ ] (Figure ) to calculate the registers settings depending on the desired RF characteristics. SmartRF Studio was used as a tool to obtain some of the register values. Though some values, as modulation, has some bits set as default which needed to be review in the application note of the transceiver.

![SmartRF Studio screenshot](image)

**Figure 5.2** SmartRF Studio screenshot. The software is used to configure most of the transceiver settings.

The transceiver is by SmartRF Studio configured with the following settings, the RF output is set to 0dB, deviation frequency shift of 3 kHz, data rate of 9600 bps, GFSK or 2FSK in modulation and the rest of the registers in the default settings. Some of them have been changed, the description of each register it is described as a comment in the code.
Figure 5.3 Flow diagram of the transmission chain merging the transceiver commands and the AX.25 functions.
5.4. Program Flow

The software consists of three modules, the comSc that implements the AX.25, the HAL, which controls the radio, and the main program, `synchronous_com.c`.

The Hardware Abstraction Layer (HAL) is the firmware library for the CC1101. It manages the power management module, and the functions to read and write the registers of the radio. It also manages the radio operations modes, TX, RX, IDLE, this commands are called *Strobes*.

The main program is structured with an interrupt vector, for Rx and TX, the program starts transmitting unless there is a RX interruption.

![Diagram of the software libraries relationship.](image)

5.5. Redundant transceiver control

The MHX425 module use a serial interface (RS232) to communicate with the on board computer. It has two different modes of operation and it is necessary to have the ability to work in both modes.

5.5.1. Data mode

The data mode is the normal operating mode, in a Point-to-Point configuration there is one master module and one slave mode. Master will provide synchronization and will send acknowledgements packets to the slave. When configured as slave, the unit will search for synchronization with the master.

5.5.2. Command mode

This mode allow the system to configure and program the unit, configuration is done via one word commands, called “AT commands”.
Software development

There are as well default settings for different network topologies. The topology of the system is a point to point (Cubesat – Ground Station) so interest is focused only in this topology. The MHX425 has a pair of default settings for this mode, all the parameters preset (see Table 5.1) meet the requirements of the system, so it is a good start point. Nonetheless, we can modify the settings by changing the core registers, called the “S_registers”, values anytime.

5.5.3. Flow diagram

To start the unit is quite a simple process, the module can be seen as a black box, raw data enters, and comes out. It is only important to be sure to be working with the correct settings.

The unit will be used as a backup transceiver; therefore, it will be turned on by a command of the On Board Computer when the main transceiver unit fails.

Once turned on, the first thing to do is to establish communication with the unit, this is done through a serial port, set the configuration in the command mode, load the settings, and then start the normal operations.

The figure 5.5 shows the flow diagram of the functions performed. Each function will be briefly explained and the code can be found in the annexes.
Figure 5.5 Flow diagram for the functions of the MHX 425 transceiver.
5.5.3.1. Open serial port

This function opens the serial port connection and configures the connection to the required settings, 9600, 8N1. It returns a file descriptor where the read/write operations to the module will be performed.

5.5.3.2. Enter command mode

To enter the command mode, the more reliable way is to force the command mode no matter in which mode the unit wakes up. This is done through simple steps, sending escape characters and waiting for the response of the unit.

5.5.3.3. Load default settings

Once in command mode, as a first approach the default settings are loaded for a point to point network.

In this configuration there are two sets of settings, for master and slave. It is preferable to have the unit on board set as a slave, since in any case as the Master provides the synchronization; it is easier to reset it on Earth for security reasons.

&F: this AT command resets the module and loads the default factory configurations

&F6: Configures the unit as master.

&F7: Configures the unit as slave.

These configurations set the values shown in table 5.1.

<table>
<thead>
<tr>
<th><strong>Table 5.3 Point to Point default settings for MHX425</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AT&amp;F6-Point-to-Point Master Default Settings</strong></td>
</tr>
<tr>
<td><strong>Parameter</strong></td>
</tr>
<tr>
<td>DCD</td>
</tr>
<tr>
<td>DTR</td>
</tr>
<tr>
<td>Handshaking</td>
</tr>
<tr>
<td>DSR</td>
</tr>
<tr>
<td>Operating Mode</td>
</tr>
<tr>
<td>Serial Baud Rate</td>
</tr>
<tr>
<td>Wireless Link Rate</td>
</tr>
<tr>
<td>Network Address</td>
</tr>
<tr>
<td>Encryption Key</td>
</tr>
<tr>
<td>Output Power</td>
</tr>
<tr>
<td>Data Format</td>
</tr>
<tr>
<td>Packet Retransmissions</td>
</tr>
<tr>
<td>Network Type</td>
</tr>
<tr>
<td>Destination Address</td>
</tr>
<tr>
<td>Repeaters YN</td>
</tr>
<tr>
<td>Serial Channel Mode</td>
</tr>
<tr>
<td>Protocol Type</td>
</tr>
<tr>
<td>Sleep Mode</td>
</tr>
<tr>
<td>Awake Timeout</td>
</tr>
<tr>
<td>Sniff Duration</td>
</tr>
<tr>
<td>Quick Sync Mode</td>
</tr>
<tr>
<td>Quick Sync Timeout</td>
</tr>
</tbody>
</table>
5.5.3.4. Change default settings

After loading the default settings, any setting can be changed at any time, as the radio will wake up whenever it receives some data in the serial port. It will compare the received data with a valid command word to allow changing the settings.

A setting is easily reconfigured by naming the register and assigning a value in the format \( \text{S}xxx=yyy \). This is sent as a string to the radio in the command mode.

The total of the \( S \_\text{registers} \) details can be found in [6] the MHX manual. Only the ones which are critical and useful to meet the requirements are described briefly.

• \( S103 \) Wireless Link rate: there are three values available,
  
  1 115Kbps  
  2 172Kbps* by default  
  3 230Kbps

• \( S107 \) Encryption Phrase is set to the string “default”.

• \( S108 \) Output power level, set to the maximum by default (30 dBm), the link budget said that to guarantee the link, the maximum should be set.

• \( S113 \) number of packet retransmissions set to 5. It can be increased if the link quality is marginal.

• \( S123 \) Reading the \( S \_\text{registers} \) is also useful, it makes possible to known the performance of the link (of the last four hops) by using the command “S123?”

• \( S143 \) Selects the “Sleep mode” for power savings. Along with the default configuration, this mode is set to 1. In this mode, the radio is always sleeping until it receives data in the port.

• \( S149 \) LED dimming, it turns on or off the LEDs output. Since the interest is to save energy, this is set to zero. (100 is the default value, “always on”)

To save the settings, the \( \text{AT\&WA} \) commands are used.

5.5.3.5. Enter data mode

To go from the command mode to the data mode or to normal operations the AT command “\&ATA” is used.

5.5.3.6. Read/Write in Serial port

To read and write in the serial port is as easy as using the read write functions and the file descriptor of the port.

\[
\text{write (port, \&cmd, sizeof(cmd))};
\]

5.5.3.7. Read/Write in data bus

This functions should be implemented in the on board computer, and are used to read and write in the common data bus of the Cubesat.
Software development

The radio pair was tested with the code in the annexes but modify to send the current time stamp for a large period.

**5.6. Ground Station settings**

To receive the signals from the CUBECAT-1, the radio needs to be set to the correct settings. This can be done manually or via the HAM radio application. The radio can be controlled by the PC via a serial port, actually the COM 5 port with a data rate of 19200, 8N1.

The radio can operate in two bands simultaneously, a main band and a sub band in the mode “Satellite”

FSK is not directly supported by the IC-910H but it can be operated in FM for 9600 bps and SSB or FM for 1200 bps.

The received or transmitted data is send through the DATA socket in the rear panel of the radio.

This DATA socket is currently connected to the ISIS switch, where, it is multiplexed to the both TNC's on the Ground Station.

![Diagram of TNC data connections](image)

* Figure 5.6 TNC data connections diagram.

The rotors are connected to COM3 at a data rate of 9600.

The radio, IC-910H is connected to the COM5 at a rate of 19200.

The TNC31 was programmed to be connected at a data rate of 57600.

The TNC7 Multi, is connected to the COM11 at a rate of 38400 in the main transceiver.

**5.7. GENSO Integration**

The integration with GENSO was made by setting up the network with the following settings:
IP GSS : 147.83.68.234

AUS port: 8051

HOST ports: 10810 or 10811

AUS ip : geon.genso.org

Since GENSO is an IP based network, it is necessary to have the TCS working in Flexnet mode, so that the network controller is able to control the radio and set the data rate needed. Flexnet is a set of drivers that allows the use of applications over TCP/IP as clients of a radio packet network.

By the date it was available, the first GENSO release was working for 4 months and haven recorded many passes, some of the longest pass reports came from the ISS (Figure 5.7) and one of the most-significant ones were the beacons from the Cubesats launched on VEGA (Figure 5.8).

**Figure 5.7** Screenshot of the data packets received from ISS in the Ground Station.

Though, the first release had sporadic errors while connecting with the M2 rotors of the ground station. The GENSO network claimed to have the driver fixed in the second release, installed 3 months ago, though this new release besides not supporting the rotors, has deleted the drivers for TNC31 as well.

Even though the TNC31 support is not critical since the Flexnet service is provided by the TNC7, the rotors are still a bug.

To be able to track the satellite not inter-fearing with the GENSO client, it is necessary to do the control directly via a serial interface as Hyper Terminal, because when opening a tracking program as Orbitron interferes with the radio connection...
Software development

with the network. The simplest solution is to initialize the GENSO client with the rotors configured in dummy mode and having at the same time the control over the M2 rotors via a Hyper terminal client. To control the rotors is very simple, there are three basic commands:

\[
\begin{align*}
S & : \text{Stop} \\
E = xx.xx & : \text{Sets the elevation} \\
A = xx.xx & : \text{Sets the azimuth}
\end{align*}
\]

Since it is not possible to start tracking software at the same time as GENSO, one possibility tried was reading from the log file searching for the commands sent to the dummy rotor, but this was slow and ineffective.

![Screenshot of beacon signal received on the VEGA launch day from PWSat in the Ground Station.](image)

**Figure 5.8** Screenshot of beacon signal received on the VEGA launch day from PWSat in the Ground Station.

### 5.7.1. Ground Station Switch

Actually the ground station has the TNC’s connected to work one at 9600 bps and the other at 1200 bps in a dedicated way, even though TNC7 has the ability to work in both modes. The existing switch takes the radio data output and redirects the data either to TNC31 or TNC7. This switch is manual, meaning that it cannot be changed remotely. To solve this issue it was proposed to implement a switch over a simple microcontroller connected to the PC via USB, so that it makes it possible to select remotely either one or other TNC, adding flexibility to the system.

The switch should be able to be changed either manually or remotely.

A diagram of the proposed setup is shown in Figure 5.9, there are 4 positions, a RESET, which puts to ground all the switches, a switch to connect to the TNC31, one to the TNC7 and finally an audio card.

As there is the option of manually change the selection, a feedback is given to the micro to update that information so there is no conflict afterwards.

As a first approach, a simulation was done in PROTEUS software to check the logic of the switch and to test the USB interface. A very simple graphic interface in Visual
controlled the changes in the simulation. The hardware implementation is proposed for further work.

Figure 5.9 Circuit simulation screenshot of the switch.

Figure 5.10 Screenshot of the Visual interface used in the simulation.

5.8. Received Signal

Early tests were performed using the synchronous mode in the transceiver for the downlink sending AX.25 packets containing ASCII text strings (Send_data[]) having been able to decode a testing phrase at 9600bps, with both GMSK and 2FSK using the TNC31 in Kiss mode via an Hyper Terminal and the Mixw software TNC(Figure 5.11).

The ICOM-910H (ground station radio) was setup in satellite mode, the main radio set in FM mode, and in 436 MHz
Software development

Beacon signals (Continuous Wave) were detected with no problem from other satellites in 1200 bps (Figure 5.8). Since the team decided to use a separated beacon transmitter no further tests were made on the CC1101.

The communication has been tested between two CC1101 as well using the test packets of SmartRf, with an attenuator in between and no packets lost.

The point to point network was set between two MHX425 using two serial ports and the testing boards, a Hyper terminal was opened to send and receive data. One was configured as a slave and the other one as a master, and the communication was tested both ways with no packet loss.

Figure 5.11 Packet received in the MixW software.
Chapter 6 FUTURE WORK AND LESSONS LEARNED

MSP-FET Interface

It was important for the tests that the JTAG interface (used for programming of the MCU) and UART interface (used to control and debug the circuit) were functioning properly, after sometime of testing, the MSP-FET would ask for an update, the last version of the MSP-FET firmware is the 0104, but while working on Code Composer, it would sometimes ignore the device, as a solution, it is recommended to downgrade the controller and re install the oldest version, if the problem persists, upgrade the firmware again if not possible with Code Composer, do it via Kickstart.

TNC31 and KISS Protocol

The "asynchronous packet protocol" spoken between the host and TNC is very simple, since its only function is to delimit frames. Each frame is both preceded and followed by a special FEND (Frame End) character, analogous to an HDLC flag. No CRC or checksum is provided. In addition, no RS-232C handshaking signals are employed.

The special characters are:

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
<th>Hex value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FEND</td>
<td>Frame End</td>
<td>C0</td>
</tr>
<tr>
<td>FESC</td>
<td>Frame Escape</td>
<td>DB</td>
</tr>
<tr>
<td>TFEND</td>
<td>Transposed Frame End</td>
<td>DC</td>
</tr>
<tr>
<td>TFESC</td>
<td>Transposed Frame Escape</td>
<td>DD</td>
</tr>
</tbody>
</table>

The reason for both preceding and ending frames with FENDs is to improve performance when there is noise on the asynchronous line. The FEND at the beginning of a frame serves to "flush out" any accumulated garbage into a separate frame (which will be discarded by the upper layer protocol) instead of sticking it on the front of an otherwise good frame. As with back-to-back flags in HDLC, two FEND characters in a row should not be interpreted as delimiting an empty frame.

While making the first tests, the data taken from the serial output had this flags, a post processing of the data received to remove the flags it is not complicated. But a problem encountered is that apparently the TNC31 doesn't only works with KISS mode, but others. When powered up for the first time, the KISS mode was selected along with the serial port data rate. Unfortunately there is no much documentation about this TNC but a single document explaining very briefly initial setups and a procedure to reconfigure some settings. When trying to repeat the tests, the TNC will show a correct function in the LED's (that indicate a correct reception of a packet at 9600 bps) but it won't pass the data to the computer. One possible reason is the TNC being reconfigure to an else mode but KISS. All the possible data rates have been tried, the commands to reconfigure the TNC won't work. A suggestion will be to perform a hard reset on the TNC and reconfigure it again.
Future work and lessons learned

Figure 6.1 Flow diagram of the reception software on the ground station to allow the retransmission of packets.

- Read from serial port
- Search for identifying word
- Search for number header n
  - Packet n received?
    - yes
    - no
      - Save number in vector
        - Counter = Total number of packets send?
          - yes
          - no
          - no
            - Send command to Cubecat
              - ask for retransmission for vector length
In the meanwhile, TNC7 could work for either 9600 or 1200 bps. As a redundant TNC, the MixW software can work as a TNC but requires connecting the data output to the sound card.

**On board computer**

The AX.25 software has been extensively tested sending ASCII strings, but it was not tested reading the data from the on board computer bus. The first On Board Computer was a MSP430 based board from Pumpkin, there code has been written to read data via SPI from a RAM memory, but after many tries, it was not possible to synchronize the clock signal of the memory with the clock of the MSP430.

![Flow diagram of segmentation of data in the OBC to send it to the transceiver.](image)
Future work and lessons learned

Some of the interfaces in the testing board of Pumpkin were not working properly, while connecting the peripheral devices, it was then when the team decided to change the on board computer to the PortuxG20.

The initial approach of the system was different from the actual design, even though the hardware, as the antenna and the transceivers remained the same, the lack of documentation and these issues represent the lessons learned through the course of designing and implementing the communications subsystem.

Application layer

Since the change of the on board computer approach, the integration of the software with the main system was not tested or implemented yet, but as the design of the communications systems requires to be as independent as possible, it is practical to see the transceiver as a black box, the input is a burst of packets of raw data of 256 bytes maximum and that outputs decoded packets.

The retransmissions requests and commands parse will be performed in the on board computer, in this way, processing is not loaded into the microcontroller transceiver, keeping the power consumption to the minimum.

Since the transceiver software works with UI AX.25 packets, for simplicity the length of the packet is fixed to 256 bytes, in this way the decoding is as well easier and the post processing also (since it is known the length of the data to be expected). This implies, though, that the data in the input of the transceiver should be segmented in 256 bytes packets, this should be implemented in the on board computer routine that will send the data to the communications subsystem, the figure 6.2 shows the steps to a suggestion of how to do so. To identify the order of the packets in the ground station, 2 bytes are reserved for a simple numeration of the packets.

In the receptor side, after doing the ax.25 decoding transparently, post processing should be done to read the numeration header, in order to reorder the information or in the case of failure, find the missing packet number, the ground station will save the number of the packets missing, and after receiving all the data, next step is to ask for retransmission of the numbers of packets missing to the Cubesat.

This command should be parsed in the OBC, which should read from the segmented data the numbers required and send them to the transceiver.

It can be seen that the transceiver doesn’t know about retransmission, it is only receiving any data the OBC sends it.

Graphic interface and switch

Since a lot of testing was done in the code, and the integration with the on board computer was not done, a graphic interface dedicated to the CubeCat only was not yet developed, it is to be done for the future members of the team once the command words are defined. The switch proposed could not be implemented in hardware by the time it was finished, it remains as a future task to test the software simulations and have a better control over the ground station.
Chapter 7 **CONCLUSION**

This thesis was an approach to close the link between the Ground Station and the communication subsystem of the CubeCat-1.

The subsystem was defined as a redundant semi-duplex link in the UHF amateur satellite frequency (435-438MHz), and an independent beacon transmitter in VHF, the systems is compatible with the GENSO network.

After describing the hardware choices, the link budget margin calculations helped in defining the data rate to guarantee the availability of the link, of course the approach was to have the maximum data through output, the calculations showed it was possible to transmit at 9600 baud using a GFSK/ 2FSK modulation scheme while upholding the electrical, thermal, regulatory and functional requirements.

The prototype system has been tested with the ground station and the compatibility has been confirmed as much as possible. AX.25 data packets have been transmitted successfully from the prototype system to the ground station.

Designing and implementing a communications subsystem for a Cubesat is a project full of tradeoffs. The mission requirements state that the communication subsystem should be as reliable as possible, and at the same time to have a low power and low cost constraints.

The current communications subsystem design meets these conditions by exploiting low power, commercial-off-the-shelf components, which are both relatively cheap and power efficient, and using low power, programmable microcontrollers to provide, in software, functionality that typically requires additional hardware. The reliability of the subsystem is ensured through a two level approach: First, through hardware redundancy, and second, through software.

The ground station has been working successfully with the GENSO network, since it was connected, although a driver incompatibility with the rotors is a bug and needs an additional control.

The road to success was paved with many hours of debugging and frustration. While having hardware and software incompatibilities and unexpected behaviors, a future design will hopefully take notice of the issues discovered in this design, and correct them in the next satellite.

In the end, however, the design met all the stated requirements and represents an approach to the implementation of a Cubesat communications system.
Conclusions
BIBLIOGRAPHY


[12] GENSO, URL: www.genso.org


[26] IARU, URL: http://www.iaru.org/
[31] “ASK RF Transmitter module Data Sheet”, URL: http://ece.colorado.edu/~ecen2830/components/TWS-BS-3_433.92MHz_ASK_RF_Transmitter_Module_Data_Sheet.pdf
[33] “MSP430TM SoC with RF Core, SLAS554E”, (2009)
[34] ITUPsat-1, URL: http://usu.itu.edu.tr/en/
ACRONYMS

The following acronyms are used in this thesis.

ACK Acknowledged message
ADCS Attitude Determination and Control System
AFSK Audio Frequency Shift Keying
AOS Acquisition of Signal
APRS Automatic Packet Reporting System
ARQ Automatic Repeat Request
BER Bit Error Rate
CCSDS Consultative Committee for Space Data Systems
COMM Communication
CRC Cyclic Redundancy Check
CW Continuous Wave
Cal Poly California Polytechnic State University
EIRP Equivalent Isotropically Radiated Power
EPS Electrical and Power System
FCS Frame-Check Sequence
FEC Forward Error Correction
FSK Frequency Shift Keying
GENSO Global Educational Network for Satellite Operations
GEO Geostationary Earth Orbit
GFSK Gaussian Frequency Shift Keying
HAL Hardware Abstraction Layer
HDLC High-Level Data Link Control
IARU The International Amateur Radio Union
ITU International Telecommunications Union
LEO Low Earth Orbit
LNB Low Noise Block
LOS Loss of Signal
LSB Least Significant Bit
MEO Medium Earth Orbit
MET Maximum Elevation at Transit
MSB Most Significant Bit
MSK Minimum Shift Keying
OBC On-Board Computer
OOK On-Off Keying
OSI Open Systems Interconnection
PCM Pulse-Code Modulation
PER Packet Error Rate
PID Protocol Identifier
RF Radio Frequency
SNR Signal-Noise-Ratio
SSB Single Side-Band
SSID Secondary Station Identifier
TLE Two Line Element
TNC Terminal Node Controller
UI Unnumbered Information
UPC Universitat Politecnica de Cataluna
URE Union de Radioaficionados de Espana
ANEXES

Synchronous_comm.c

#include <stdio.h> /* Standard input/output definitions */
#include <string.h> /* String function definitions */
#include <unistd.h> /* UNIX standard function definitions */
#include <fcntl.h> /* File control definitions */
#include <errno.h> /* Error number definitions */
#include <termios.h> /* POSIX terminal control definitions */

* 'open_port()' - Open serial port 1 and configures it for 9600 8n1 operation
* Returns the file descriptor on success or -1 on error.
*
int open_port{
    int port;                           /* File descriptor for the port */
    struct termios options;
    port = open("/dev/ttyS1", O_RDWR | O_NOCTTY | O_NDELAY); //Slave then send AT&F7 command
    if (port == -1)
    {
        /* Could not open the port */
        fprintf(stderr, "open_port: Unable to open /dev/ttyS1 - %s\n",
                strerror(errno));
    }
    fcntl(port, F_SETFL, FNDELAY);        /* Configure port reading */
    /* Get the current options for the port */
    tcgetattr(port, &options);
    cfsetispeed(&options, B9600);        /* Set the baud rates to 9600 */
    cfsetospeed(&options, B9600);
    /* Enable the receiver and set local mode */
    options.c_cflag |= (CLOCAL | CREAD);
    options.c_cflag &= ~PARENB; /* Mask the character size to 8 bits, no parity */
    options.c_cflag &= ~CSTOPB;
    options.c_cflag &= ~CSIZE;
    options.c_cflag |= CS8;             /* Select 8 data bits */
    options.c_cflag &= ~CRTSCTS;        /* Disable hardware flow control */
    /* Enable data to be processed as raw input */
    options.c_iflag &= ~(ICANON | ECHO | ISIG);

    /* Set the new options for the port */
    tcsetattr(port, TCSANOW, &options);
    return(port);
}

int set_default_config(int port) //Sets default settings for point to point
{
    int port;
    char cmd;
    cmd="AT&F7" //comand to set as slave
    write (port, &cmd, sizeof(cmd));
    cmd="S143=1"; //set the sleep mode to 1, sleep until receiving data
void main()
{
    int port;
    char cmd, chout, rcvd;
    char comm="chsetting";
    char ack="waitingcmd";
    char perf; //variable to save the performance of the link value
    port=open_port(); //open serial port

    sleep(100);
    cmd="+++" //go to command mode
    write (port, &cmd, sizeof(cmd));

    sleep(100);
    cmd="NO CARRIER OK";
    read(port, &chout, sizeof(cmd));
    if(chout==cmd) //verify we are in the command mode
    {
        set_default_config(port); // set default configuration
    }

    while(1)
    {
        read(port, &rcvd, sizeof(rcvd));  //check if we have got a command from GS
        if(rcvd==comm) //we received command to change settings
        {
            write(port, &ack, sizeof(ack)); //Send ack that we are waiting for command
            sleep(100);
            read(port, &rcvd, sizeof(rcvd)); //read command
            sleep(100);
            cmd="+++" /*go to command mode*/
            write (port, &cmd, sizeof(cmd));
            sleep(100);
            cmd="NO CARRIER OK";
            read(port, &chout, sizeof(cmd));
            if(chout==cmd) /*we are into the command mode*/
            {
                write (port, &rcvd, sizeof(rcvd));//change setting
            }
            cmd="AT&WA"//save settings
            write (port, &cmd, sizeof(cmd));
            cmd="ATA"//go back to DATA mode
            write (port, &cmd, sizeof(cmd));
        }

        if(rcvd=="check") //we received command to check the link
        {
            sleep(100);
cmd="+++" //go to command mode
write (port, &cmd, sizeof(cmd));
sleep(100);
cmd="NO CARRIER OK";
read(port, &choutch, sizeof(cmd));
if(chout==cmd) //we are into the command mode
{
    cmd="S123?" //read S_register of link performance
    write (port, &cmd, sizeof(cmd)); //send the command
    sleep(100);
    read(port, &perf, sizeof(perf));
}

/* Close the serial port */

#include "COMSC/ax_rec.h"
#include "COMSC/ax_send.h" //declares variables to handle data
#include "COMSC/ax_misc.h"
#include "COMSC/definitions.h"
#include "Synchronous_comm.h" //declares tx & rx start variables

#define  PATABLE_VAL        (0xC4)          // 315Mhz 10 dBm output 25ma consumption
#define  PATABLE_VAL        (0x51)          // 315Mhz 0 dBm output 15mA consumption

Synchronous_comm.h

/*******************
 * CC430 RF Code Example - Synchronous Transmit
 * RF link between two devices uses synchronous mode.
 * The square wave is output to pin P2.4, which is jumpered to the GDO0
 * radio input on P2.6. Since IOCFG0 == 0x2D, data is taken from the GDO0
 * pin and modulated for the transmission.
 *
 * M. Morales/D. Dang
 * Texas Instruments
 * June 2010
 * Compiled using IAR v4.21.8 and CCS v4.1
 *******************/
```c
//define _PATABLE_VAL (0xC0) //315/433 Mhz Max dBm output 26mA consumption
RF_SETTINGS rfSettings = {
    0x00,  // FSCTRL1  Frequency synthesizer control.
    0x00,  // FSCTRL0  Frequency synthesizer control.
    0x10,  // FREQ2   Frequency control word, high byte.
    0xC4,  // FREQ1   Frequency control word, middle byte.
    0xEC,  // FREQ0   Frequency control word, low byte.
    0xF8,  // MDMCFG4  Modem configuration. Sets the BW of the filter and the data rate, F5 for 1.2K, F8 for 9.6K
    0x83,  // MDMCFG3  Modem configuration.
    0x10,  // MDMCFG2  Modem configuration. Modulation 10 for 2GFSK, 00 for 2FSK, 30 ask, 70 for msk
    0x22,  // MDMCFG1  Modem configuration.
    0xF8,  // MDMCFG0  Modem configuration. Channel Spacing
    0x00,  // CHANNR  Channel number.
    0x07,  // DEVIATN Modem deviation setting (when FSK modulation is enabled).
    0x51,  // FRENDF  Front end RX configuration.
    0x10,  // FRENDD  Front end TX configuration.
    0x18,  // MCSM0   Main Radio Control State Machine configuration.
    0x16,  // FOCCFG  Frequency Offset Compensation Configuration.
    0x6C,  // BSCFG   Bit synchronization Configuration.
    0x03,  // AGCTRL1 AG control.
    0x40,  // AGCTRL0 AG control.
    0x91,  // AGCTRL0 AG control.
    0x10,  // MDMDA0  Frequency synthesizer calibration.
    0x2A,  // MDMDA3  Frequency synthesizer calibration.
    0x00,  // MDMDA1  Frequency synthesizer calibration.
    0x1F,  // MDMDA0  Frequency synthesizer calibration.
    0x59,  // MDMDA9  Frequency synthesizer calibration.
    0x81,  // TEST2   Various test settings.
    0x35,  // TEST1   Various test settings.
    0x09,  // TEST0   Various test settings.
    0x47,  // FIFOTH   RX FIFO and TX FIFO thresholds.
    0x0B,  // IOCFG2  PDO output pin configuration.
    0x2D,  // IOCFG0  PDO output pin configuration. Refer to SmartRF® Studio User Manual for detailed pseudo register explanation.
    0x00,  // PKTCTRL1 Packet automation control.
    0x12,  // PKTCTRL0 Packet automation control.
    0x00,  // ADDR    Device address.
    0xFF,  // PKTLEN  Packet length.
};

extern RF_SETTINGS rfSettings;
extern char TX_running;
unsigned char buttonPressed = 0;

unsigned short AX_crc_ccitt_table[] = {
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xc86c, 0xd7e5, 0xe6af, 0xf587,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0x7f6b, 0x6ae5, 0x5be6, 0x4af1, 0x39f2, 0x28f8,
    0x1210, 0x03b1, 0x3594, 0x241b, 0x58a6, 0x492d, 0x77b2, 0x663f,
    0x9cd9, 0x8450, 0x7371, 0x629e, 0x51af, 0x402d, 0x3948, 0x2865,
    0x1084, 0x0101, 0x3398, 0x2215, 0x56a2, 0x4729, 0x75b4, 0x643b,
    0x9cc6, 0x8d47, 0x7f68, 0x6ae4, 0x5be5, 0x4af2, 0x39f3, 0x28f9,
    0x1211, 0x03b2, 0x3595, 0x2418, 0x58a7, 0x492e, 0x77b3, 0x663a,
    0x9cd8, 0x8451, 0x7372, 0x629f, 0x51af, 0x402e, 0x3949, 0x2866,
    0xe9ff, 0xf876, 0xd9f5, 0xc874, 0xbaf3, 0xae52, 0x9ed9, 0x8f50,
    0xfbef, 0xea66, 0xd8fd, 0xc974, 0x5285, 0x430c, 0x7197, 0x601e,
    0x14a1, 0x0528, 0x37b3, 0x263a,
};```
void main( void )
{
    // Stop watchdog timer to prevent time out reset
    WDTCTL = WDTPW + WDTHOLD;

    //Clock a 12Mhz UCS = Unified Clock System DCO = Digital Controlled oscillator
    __bis_SR_register(SCG0);          // Disable the FLL control loop
    UCSCTL0 = 0x0000;                 // Set lowest possible DCOx, MODx unified clock system
    UCSCTL1 = DCORSEL_5;             // Select DCO range 24MHz operation
    UCSCTL2 = FLLD_1 + 374;          // Set DCO Multiplier for 12MHz
    __bic_SR_register(SCG0);         // Enable the FLL control loop.

    __bic_SR_register(SCG0);         // Enable the FLL control loop.

    // Increase PMMCOREV level to 2 in order to avoid low voltage error
    // when the RF core is enabled
    SetVCore(2);

    InitRadio();
    InitButtonLeds();

    P2SEL |= BIT4+BIT5+BIT6;         // P2.4 (TA1 out), P2.6 (Radio GDO0 (direction controlled by Radio)
P2DIR |= BIT3+BIT4+BIT5+BIT7;                        // P2.4 TA1CCR1 output
P2OUT &= ~BIT7;

PMAPPWD = 0x02D52;                        // Get write-access to port mapping regs
P2MAP4 = PM_MCLK;                     // Map TA1CCR0A output to P2.4
P2MAP6 = PM_RFGDO0;                       // Map GDO0 as an input on P2.6
P2MAP5 = PM_RFGDO2;         //SEND the GDO2 output to the P2.5
PMAPPWD = 0x00;                        // Lock Port mapping

RF1AES |= BIT2;                          // 0 - Falling edge of RFIFG2 Radio core interrupt edge select register
RF1AIFG &= ~BIT2;                         // 0 - Clear a pending interrupt
RF1AIE |= BIT2;                          // Enable the interrupt
Strobe( RF_SIDLE );
TX_delaySet(10);                           //this delay is specified in the TNC manual, it is the minimum to give the frame time to send completely.
__bis_SR_register( GIE );   //GIE = 0x0008 status bit

//Transmit the TX waveform synchronously
while(1)
{
    P1IFG = 0;
    Strobe( RF_STX );                                  //Command to the radio to wake up to start transmitting
    TX_prepareFrame(SendData, sizeof(SendData));  // Prepare AX.25 frame for TX
    TX_sync();               //send synchronization
    TX_start();                 //Start sending the frame
    P1IE |= BIT7;              //Interruptions activated
}

void InitButtonLeds(void)
{
    // Initialize Port J
    PJOUT = 0x00;
    PJDIR = 0xFF;

    // Set up LEDs
    P1OUT &= ~BIT0;
    P1DIR |= BIT0;
P3OUT &= ~BIT6;
P3DIR |= BIT6;
}
void InitRadio(void)
{
    // Set the High-Power Mode Request Enable bit so LPM3 (Low power mode) can be entered
    // with active radio enabled
PMMCTL0_H = 0xA5;
PMMCTL0_L |= PMMHPMRE_L;
PMMCTL0_H = 0x00;

WriteRFSettings(&rfSettings); //Write the RF settings in the radio registers
WriteSinglePATable(PATTLE_VAL); //Set the power module registers

#pragma vector=CC1101_VECTOR
__interrupt void CC1101_ISR(void)
{
  switch(__even_in_range(RF1AIV,32))  // Prioritizing Radio Core Interrupt
  {
    case 0: break;  // No RF core interrupt pending
    case 2: break;  // RFIFG0
    case 4: break;  // RFIFG1
    case 6:
      TX_stop();
      TX_sendBit();
      TX_prepareBit();

      TX_sendSync();

      break;  // RFIFG2
    case 8: break;  // RFIFG3
    case 10: break;  // RFIFG4
    case 12: break;  // RFIFG5
    case 14: break;  // RFIFG6
    case 16: break;  // RFIFG7
    case 18: break;  // RFIFG8
    case 20: break;  // RFIFG9
    case 22: break;  // RFIFG10
    case 24: break;  // RFIFG11
    case 26: break;  // RFIFG12
    case 28: break;  // RFIFG13
    case 30: break;  // RFIFG14
    case 32: break;  // RFIFG15
  }
}

Synchronous_comm.h

#include "cc430x613x.h" //micro specs
#include "RF1A.h" //defines de rf settings variables,
#include "hal_pmm.h" // settings for the Power Management Module

/**************
 * Function Definition
 */
void Transmit(unsigned char *buffer, unsigned char length);
void ReceiveOn(void);
void ReceiveOff(void);
void InitButtonLeds(void);
void InitRadio(void);
AX.25 code

ax_rec.c

/*
 * This file is part of comSC.
 *
 * comSC is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * comSC is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with comSC. If not, see <http://www.gnu.org/licenses/>.
 */

#include <msp430x16x.h>           // generic headers
#include "ax_rec.h"                // headers for ax_rec
//#include "ax_misc.h"               // headers for ax.25
#define __AX25DEBUG__
#define STATE_FRAME_WAIT 0
#define STATE_DATA_REC 1
#define STATE_FLAG_WAIT 2
#define STATE_FLAG_WAIT2 3

// put own address
char RX_Dest[7] = {0x90, 0x84, 0x72, 0x8A, 0x8E, 0x40, 0x60};
// Symbol, Ascii (HEX), Ascii (BIN), Shifted (HEX)
// E, 45, 0100 0101, 90
// A, 41, 0100 0010, 84
// 3, 33, 0011 1001, 72
// R, 52, 0100 0101, 8A
// C, 43, 0100 0111, 8E
// U, 55, 0010 0000, 40
// , 20, 0010 0000, 40
// SSID 1 --> 011 0001 0 --> 0x62
// not last address ^

char RX_recMode;                   // RX_recMode=STATE_FRAME_WAIT: frame waiting mode
  // RX_recMode=STATE_DATA_REC: data reception mode
  // RX_recMode=STATE_FLAG_WAIT: flag waiting mode
char RX_value;                     // current voltage at the input
char RX_lastValue;                 // stores last input value to detect edges
unsigned int RX_counter = 0;       // estimates the time since the last edge
unsigned int RX_bytePointer;      // pointer to the current byte
unsigned int RX_bytePointerMax = 278; // the maximum size of the packet (default 278 bytes)
unsigned int RX_bitPointer;       // Current pointer on the bit
CubeCat-1: Communication subsystem

char RX_Data[278]; // 255 Data + 23 Bytes
 // lokal storage of the entire frame including flags, header, fcs
 // Do not use this memory for external functions as the content is
 // modified during RX.

char RX_passAll; // RX_passAll=0: use normal filtering
 // RX_passAll=1: pass all packets to the next layer, even if erroneous

extern unsigned AX_crc_ccitt_table[];

unsigned int RX_crc; // Store the FCS

// A timing interrupt occured
void RX_interrupt()
{
    unsigned int RX_counter_int; // internal RX_counter_int

    RX_value = (P1IN & 0x08) >> 3; // read new value at P1.3

    #ifdef __AX25DEBUG__ // debug: show current value and RX_recMode
    if (RX_value == 1) P4OUT |= 0x01; else P4OUT &= ~0x01;
    // show received value on P4.0
    switch (RX_recMode)
    {
    case STATE_FRAME_WAIT:
        P4OUT &= ~0x04;
        P4OUT &= ~0x10;
        break;
    case STATE_DATA_REC:
        P4OUT |= 0x04;
        P4OUT &= ~0x10;
        break;
    case STATE_FLAG_WAIT:
        P4OUT &= ~0x04;
        P4OUT |= 0x10;
        break;
    default:
        P4OUT |= 0x04;
        P4OUT |= 0x10;
        break;
    }
    #endif

    RX_counter++; // increase global RX_counter
    RX_counter_int = RX_counter; // store to internal RX_counter_int

    if (RX_lastValue != RX_value) { // test if an edge occurred
        RX_counter = 0; // zero received, reset RX_counter
        // if RX_counter<2 do nothing. Edges are too close.
        if (RX_counter_int > 1) & (RX_counter_int <= 5)) { // 0 received
            RX_recZero();
        }
        if (RX_counter_int > 5) & (RX_counter_int <= 9)) { // 10 received
            RX_recOne();
            RX_recZero();
        }
    }
if ((RX_counter_int > 9) & (RX_counter_int <= 13)) { // 110 received
    RX_recOne();
    RX_recOne();
    RX_recZero();
}
if ((RX_counter_int > 13) & (RX_counter_int <= 17)) { // 1110 received
    RX_recOne();
    RX_recOne();
    RX_recOne();
    RX_recZero();
}
if ((RX_counter_int > 17) & (RX_counter_int <= 21)) { // 11110 received
    RX_recOne();
    RX_recOne();
    RX_recOne();
    RX_recOne();
    RX_recZero();
    // omit the zero!
}
if ((RX_counter_int > 21) & (RX_counter_int <= 25)) { // bit stuffing
    RX_recOne();
    RX_recOne();
    RX_recOne();
    RX_recZero();
    RX_recOne();
    // omit the zero!
}
if ((RX_counter_int > 25) & (RX_counter_int <= 29)) { // flag received
#ifdef __AX25DEBUG__
    P4OUT |= 0x40;                     // Announce rec of flag at P4.6
    P4OUT &= ~0x40;
#endif
    if (RX_recMode == STATE_FRAME_WAIT) {
        RX_recMode = STATE_FLAG_WAIT;
    }
    if (RX_recMode == STATE_DATA_REC) {
        RX_recMode = STATE_FRAME_WAIT;
        RX_Data[RX_bytePointer] = 0x7E;
        RX_frameTest();
    }
    if (RX_recMode == STATE_FLAG_WAIT2) {
        RX_recMode = STATE_FLAG_WAIT;
    }
    // if RX_recMode == 2 do nothing!
} else {
    // too many ones received.
    RX_recMode = STATE_FRAME_WAIT;
}
if (RX_counter > 255) {
    RX_recMode = STATE_FRAME_WAIT; // reset the mode
    RX_counter = 255;
}
RX_lastValue = RX_value;        // store the value for the next iteration
}
/**********************************************
// a one has been received
void RX_recOne() {

if (RX_recMode == STATE_FLAG_WAIT2) {
    RX_Data[0] = 0x7E;
    RX_bytePointer = 1;
    RX_bitPointer = 0;
    RX_recMode = STATE_DATA_REC;
    RX_recZero();
}

if (RX_recMode == STATE_FLAG_WAIT) {
    RX_recMode = STATE_DATA_REC;
}

if (RX_recMode == STATE_DATA_REC) {             // store the 1
    RX_Data[RX_bytePointer] |= (0x01<<RX_bitPointer);
    RX_increasePointer();
}
}

/**********************************************
// a zero has been received
void RX_recZero() {

if (RX_recMode == STATE_FLAG_WAIT2) {
    RX_Data[0] = 0x7E;
    RX_bytePointer = 1;
    RX_bitPointer = 0;
    RX_recMode = STATE_DATA_REC;
    RX_recZero();
}

if (RX_recMode == STATE_FLAG_WAIT) {
    RX_recMode = STATE_FLAG_WAIT2;
}

if (RX_recMode == STATE_DATA_REC) {             // store the 0
    RX_Data[RX_bytePointer] &= ~(0x01<<RX_bitPointer);
    RX_increasePointer();
}
}

/**********************************************
// Set the maximum number of bytes to be received
void RX_setByteMax(unsigned int size) {
    if (size<256) {
        RX_bytePointerMax = size+23;
    }
}

/**********************************************
// When enabled, receive all frames, even if they are erroronous
void RX_setPassAll(char passAll) {

RX_passAll=passAll;
}
// *************************************************
// increase the data pointers. Used in RX_recZero and RX_recOne
void RX_increasePointer(void) {
  RX_bitPointer++;
  if (RX_bitPointer>7) {
    RX_bitPointer=0;
    RX_bytePointer++;
    if (RX_bytePointer>=RX_bytePointerMax) { // frame too long. Exit.
      RX_bytePointer = 0;
      RX_bitPointer = 0;
      RX_recMode = STATE_FRAME_WAIT;
    }
  }
}

// *************************************************
// test the received frame
void RX_frameTest(void) {

  char RX_frameValid = 1;               // be optimistic
  volatile unsigned int length = RX_bytePointer; // copy the length
  char RX_temp_data[278];
  unsigned short temp;
  unsigned int i;                       // generic counter

  __BIS_SR(GIE); // Allow interrupt nesting to allow TX!
  // test if the number of bytes is pair.
  // if (length&0x01) {
  //   RX_frameValid = 0;
  // }
  // Copy the data!
  for (i=0;i<=length;i++) {
    RX_temp_data[i] = RX_Data[i];
  }

  #ifdef __AX25DEBUG__
P5OUT ^= 0x04;
  printf("Packet received: %c",RX_Data);
  #endif

  // Test if the frame is sufficiently long to test for the address and the FCS
  if (length < 23)
  {
    RX_frameValid = 0;
  } else {
    // sufficiently long.
    // address correct?
    // test only 5 first bytes: if required change to test also for the SSID!
    for (i=1;i<6;++i) {
      if (RX_temp_data[i] != RX_Dest[i-1]) {
        RX_frameValid = 0;
      }
    }
if (~RX_passAll) { // check FCS only when RX_passAll is not active.
// FCS valid? Calculate FCS and compare it to received value.
temp = RX_sysCRC(RX_temp_data, length-3);

if (temp & 0x00FF) != (RX_temp_data[length-2] & 0x00FF)) RX_frameValid = 0;
if (temp & 0xFF00) >> 8) != (RX_temp_data[length-1] & 0x00FF)) RX_frameValid = 0;
}

// announce to higher layer if frame is valid (or passAll is enabled)
if (RX_frameValid)
{
    RX_packetReceived();
}

// announce packet reception to the main program.
void RX_packetReceived(void) {
    // add a hook here to announce frame reception to higher layer!
#ifdef __AX25DEBUG__
    P5OUT ^= 0x01;   // Announces frame reception
#endif
}

// Calculate the FCS
unsigned short RX_sysCRC(char *buffer, unsigned short length)
{
    unsigned short crc = 0xffff;
    *buffer++;              //CRC does not take falgs into account
    while (length--)
    {
        crc = (crc >> 8) ^ AX_crc_ccitt_table[(crc ^ *buffer++) & 0xff];
    }
    return crc ^ 0xffff;

ax_rec.h

 ifndef __ax_rec__
define __ax_rec__

// external routines
void RX_setByteMax(unsigned int size); // set the maximum number of data bytes in a frame
void RX_setPassAll(char passAll);       // set/clear passAll flag

// internal routines (place a hook!)
void RX_packetReceived(void);           // announce reception to main program

// internal routines
void RX_fcs(unsigned char msg);
void RX_recOne(void);                   // a one is received
void RX_recZero(void);                  // a zero is received
void RX_interrupt(void);                // interrupt handling for RX
void RX_frameTest(void);                // test the received frame
void RX_increasePointer(void); // increase bitPointer and bytePointer
unsigned short RX_sysCRC(char *frame, unsigned short size_frame);

#endif

ax_send.c

/*
 * This file is part of comSC.
 *
 * comSC is free software: you can redistribute it and/or modify
 * it under the terms of the GNU General Public License as published by
 * the Free Software Foundation, either version 3 of the License, or
 * (at your option) any later version.
 *
 * comSC is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with comSC. If not, see <http://www.gnu.org/licenses/>.
 */

#include "RF1A.h"
#include "ax_send.h" // headers for ax_send, prepares the variables for handling data
//include "ax_mime.h" // headers for ax_mime
#include "definitions.h" //defines the max value of ax.25 & data field size in bytes

#define __AX25DEBUG__

#define SYNC_INACTIVE 0
#define SYNC_LOW 1
#define SYNC_HIGH 2
#define SYNC_ALTERNATE 3

// *********************************************************
// char TX_Header[16] = {0x90, 0x84, 0x72, 0x8A, 0x8E, 0x40, 0x60, 0x90, 0x84, 0x72, 0x8A, 0x8E, 0x62, 0x67, 0x03, 0xF0};
// Symbol, Ascii (HEX), Ascii (BIN), Shifted (HEX)
// E, 45, 0100 010
// A, 41, 0100 0010, 84
// 3, 33, 0011 1001, 72
// R, 52, 0100 0101, 8A
// C, 43, 0100 0111, 8E
// U, 55, 0010 0000, 40
const char TX_Header[16] = {'E' << 1, // dest address
'A' << 1,
'R' << 1,
'C' << 1,
'U' << 1,
0x60, // SSID for destination - earth station
'C' << 1, // source address
'U' << 1,
'B' << 1,
'E' << 1,
'C' << 1,
'A' << 1,
0x67,      // SSID for source - satellite
0x03,      // control byte - for UI or unnumbered frame
0xF0 };    // PID - for no layer 3 protocol implementation

int size_header = sizeof(TX_Header);
unsigned int DAC_lowlevel;
unsigned int DAC_highlevel;

// ******************************************************
// Global Variables
char TX_Data[AX25_MAX];    // 4 Data + 20 Bytes
// local storage of the entire frame including flags, header, fcs
// Do not use this memory for external functions as the content may
// be modified due to bit stuffing!

char TX_Data_to_CRC[16 + 4]; // 16 bytes (Address + control bytes) FIXED (do not modify)
// + 4 bytes Data (modify according to size of your data field)
// The array stores the AX25 header + Data that are used as input for the FCS (CRC) calculation

unsigned short TX_crc;     // Store the FCS -> calculated with for, probar con la tabla si es mas
                          // rapido
char TX_oneCounter;        // counts the number of 1's in a row. When it gets to 5, it is time to
                          // insert a 0. Esto es bit stuffing
unsigned int TX_bytePointer; // pointer to the current byte
unsigned int TX_bytePointerMax; // length of the packet
char TX_bitPointer;         // current pointer on the bit

int TX_DelayCounter;       // Counter for TX_Delay
unsigned int TX_Delay;     // Stores the settings of the TX_Delay
unsigned char TX_output_logiclevel; // stores the output logic level for TX
char TX_running = 0;       // TX (for frame transmission) active?
char TX_stop_bit = 0;
char TX_sync_mode = 0;     // current mode for synchronization emission
unsigned short TX_sync_counter = 0; // counter for the synchronization
unsigned char TX_data_bit; // next data bit to be transmitted.
unsigned char TX_mdm = 0;
volatile char TX_lastShoot = 0; // keep the TX active during the transmission of the last bit

unsigned long state = 0;

extern unsigned short AX_crc_ccitt_table[];
extern unsigned char SendData[];
/* for tests, send next frame immediately
char SendData[] = { 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44,
                   0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44,
                   0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44,
                   0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45,
                   0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45,
                   0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45,
                   0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45, 0x44, 0x54, 0x45,
void TX_DACSet(unsigned int low, unsigned int high) {
    DAC_lowlevel = low;
    DAC_highlevel = high;
}

void TX_DACReset(void) {
    DAC_lowlevel = 0x088D;
    DAC_highlevel = 0x0AD5;
}

void TX_delaySet(unsigned int x) {
    TX_Delay = x;
}

void TX_enablePA(void) {
    //DAC12_1DAT = 0x0C9A;        // Enable the PA (2.6 Volt)
    //DAC12_1DAT = 0x00FF;        // Enable the PA
}

void TX_disablePA(void) {
    //DAC12_1DAT = 0x0000;          // Disable the PA
}

void TX_toggle(void) {
    //TX_output_logiclevel ^= 0x01;   // invert TX_output_logiclevel
    //DAC12_0DAT = TX_output_logiclevel?DAC_highlevel:DAC_lowlevel;
    P2OUT ^= BIT7;  //Invertim el bit
TX_oneCounter = 0;      // output is a zero. Reset counter.
}

// Calculate the FCS
unsigned short sysCRC(char *buffer, unsigned short length)
{
    unsigned short crc = 0xffff;

    *buffer++;          //CRC does not take faags into account
    length = length - 4; //No flags, and no bytes for CRC. Only data and headers

    while (length--)
    {
        crc = (crc >> 8) ^ AX_crc_ccitt_table[(crc ^ *buffer++) & 0xff];
    }
    return crc ^ 0xffff;
}

// Calculate the FCS
void compute_CRC(char *frame, unsigned short size_frame)
{
    unsigned short temp;

    temp = sysCRC(frame, size_frame);
    frame[size_frame - 3] = (temp & 0xff);
    frame[size_frame - 2] = ((temp >> 8) & 0xff);
}

// Start transmission
void TX_start(void) {
    P1OUT ^= BIT0;  //Encenem el LED per indicar què estem enviant
    TX_oneCounter = 0;            // Initialize counter
    TX_bytePointer = 0;           // Initialize bytePointer
    TX_bitPointer = 0;            // Initialize bitPointer
    TX_DelayCounter = TX_Delay;   // Initialize DelayCounter
    //TX_enablePA();                // Enable the PA
    state = 0;
    TX_mdm=0;
    TX_output_logiclevel = 0;     // Ensure the output logiclevel to be low
    P2OUT &= ~BIT7;   //Comencem amb un 0
    RF1AIE |= BIT2;  //Activem les interrupcions
    Strobe( RF_STX );

    TX_running = 1;               // Enable the TX
}

// End transmission
void TX_end(void) {
    RF1AIE &= ~BIT2; //Desactivem ls interrupcions de la radio
    Strobe( RF_SIDLE );
TX_running = 0;                  // stop the TX
P1OUT ^= BIT0;                  //Parem el LED
TX_start();
TX_disablePA();                // disable the PA

// *****************************************************************
// Copy the data bytes into the internal memory
// length: length of the data bytes
void TX_prepareFrame(char *data, unsigned int length_data) {

    int i;
    unsigned short TX_Data_length = (unsigned short) (length_data + 20);

    _BIS_SR(GIE);                // Allow interrupt nesting to allow RX!

    if (length_data > DATA_MAX) length_data = DATA_MAX;       // length limitation not required. char < 257

    TX_bytePointerMax = length_data + 20;         // frame size is data size + 20 bytes

    // Send a flag first
    TX_Data[0] = 0x7E;                     // flag

    // Append the addresses and the control bytes
    for (i = 0; i < size_header; i++) {
        TX_Data[i+1] = TX_Header[i];
    }

    // Append the data
    for (i = 0; i < length_data; i++) {
        TX_Data[i+17] = *data++;       // data
    }

    // Calculate and append the FCS (low byte first)
    compute_CRC(TX_Data, TX_Data_length);

    // Append the final flag
    TX_Data[TX_bytePointerMax-1] = 0x7E;   // flag

}

void TX_stop(void){
    if(TX_stop_bit == 1){
        __delay_cycles(100000);
        RF1AIE &= ~BIT2; //Desactivem ls interrupcions de la radio
        Strobe( RF_SIDLE );
        TX_stop_bit = 0;
        //TX_start();
    }
}

// *****************************************************************
// Send a bit.
// Beware that TX_Data may be changed due to bit stuffing.
// Therefore, never use TX_Data outside of the TX routines to store data.
void TX_sendBit(void) {
TX_toggle();
TX_lastShoot = 0;
}

// prepare the next bit.
void TX_prepareBit(void)
{
    int TX_flag = 0;
    if ((TX_running)&&(TX_sync_mode == 0)) // Only prepare the next bit, if TX is enabled
    {
        TX_data_bit = 1;
        // during TX_Delay send flags all the time
        if (TX_DelayCounter>0)
        {
            TX_data_bit = TX_Data[TX_bytePointer] & 0x01;
            TX_bitPointer++;
            TX_Data[TX_bytePointer] >>= 1;
            if (TX_bitPointer>7)
            {
                TX_bitPointer = 0;
                TX_DelayCounter--;
                TX_Data[TX_bytePointer] = 0x7E; // reload frame
            }
        }
        return;
    }

    // check if a flag is transmitted (no bit stuffing)
    if ((TX_bytePointer==0) || (TX_bytePointer==TX_bytePointerMax-1)) TX_flag = 1;
    //if(TX_bytePointer == (TX_bytePointerMax-4))
    //   TX_flag = TX_flag;

    // update TX_bitPointer and TX_bytePointer
    if (TX_oneCounter>4) // consider if a bit stuffing is required
    {
        // bit stuffing required. Change data and do not increase TX_bitPointer
        TX_oneCounter = 0; // reset TX_oneCounter
        TX_data_bit = 0; // send a 0 next time.
    }
    else
    {
        TX_data_bit = TX_Data[TX_bytePointer] & 0x01; // always take the LSB bit
        TX_Data[TX_bytePointer] >>= 1; // shift data to the right
        TX_bitPointer++; // point to next bit
        if (TX_bitPointer>7)
        {
            TX_bitPointer=0;
            TX_bytePointer++;
        }
    }
}

// prepare the next bit.
// transmit bit
if ((TX_flag==0) && (TX_data_bit == 1)) // send flag. No bit stuffing
{
    TX_oneCounter++;
}

// Request to send a synchronization sequence
void TX_sync(void)
{
    TX_enablePA(); // Enable the PA
    TX_output_logiclevel = 0; // Ensure the output logiclevel to be low
    TX_sync_mode = SYNC_ALTERNATE; // Enable theTX
}

// Toggle the switch signal (to keep the COM connected to the antenna)
void TX_switch_toggle(void)
{
    if ((TX_running | TX_lastShoot == 1) || (TX_sync_mode != 0))
    {
        P1OUT ^= 0x04;
    }
}

// Send the synchronization sequence
void TX_sendSync(void)
{
    switch (TX_sync_mode)
    {
    case SYNC_INACTIVE: // nothing to do
        break;
    case SYNC_LOW: // send low frequency
        P2OUT &= ~BIT7; //P2.7 a 0
        TX_sync_counter++;
        if (TX_sync_counter > 5000)
        {
            TX_sync_counter = 0;
            TX_sync_mode = SYNC_HIGH;
        }
        break;
    case SYNC_HIGH: // send high frequency
        P2OUT |= BIT7; //P2.7 a 1
        TX_sync_counter++;
        if (TX_sync_counter > 5000)
        {
            TX_sync_counter = 0;
            TX_sync_mode = SYNC_ALTERNATE;
        }
        break;
    case SYNC_ALTERNATE: // send alternative signal
    }
P2OUT ^= BIT7; // Inverteix el Bit P2.7
TX_sync_counter++;
if (TX_sync_counter > 5000)
{
    TX_sync_counter = 0;
    TX_sync_mode = SYNC_INACTIVE;
    //TX_disablePA();
    P2OUT &= ~BIT7;     // Ensure the output logiclevel to be low
}
break;
default: // nothing to do
break;
}

ax_send.h

#include "cc430x613x.h"
#include "hal_pmm.h"
#ifndef __ax_send__
#define __ax_send__
#define __ax_send__

// external routines
void TX_start(void);                                // start the TX
void TX_prepareFrame(char *data, unsigned int length);   // prepare the data for TX
void TX_delaySet(unsigned int a);                   // change time for sending 0x7E prior to any frame
void TX_DACSet(unsigned int low, unsigned int high); // set the dac
void TX_DACReset(void);
void TX_sync(void);                                 // send a synchronization signal
void TX_stop(void);

// internal routines
void TX_fcs(unsigned char msg);                     // Calculate the FCS
void TX_toggle(void);                               // Change output value
void TX_end(void);                                  // End of transmission
void TX_sendBit(void);                              // send one bit
void TX_prepareBit(void);                           // prepare next bit
void TX_fcs_bit(int lsb_int);
void TX_sendSync(void);
void TX_enablePA(void);                             // enable power amplifier
void TX_disablePA(void);                            // disable power amplifier
unsigned short sysCRC(char *buffer, unsigned short length);
void compute_CRC(char *frame, unsigned short size_frame);
void TX_switch_toggle(void);

#endif

Definitions.h

#ifndef DEFINITIONS_H_
#define DEFINITIONS_H_
#define DEFINITIONS_H_
#define AX25_MAX   276     // Max number of bytes for an ax25 frame.
#define DATA_MAX   256     // Max number of bytes for data field.
RF1a.c

#define MSPGCC
#define /*DEFINITIONS_H_*/

RF1a.c

#include "RF1A.h"
#include "cc430x613x.h"

// ****************************************************************************
// @fn          Strobe
// @brief       Send a command strobe to the radio. Includes workaround for RF1A7
// @param       unsigned char strobe        The strobe command to be sent
// @return      unsigned char statusByte    The status byte that follows the strobe
// ****************************************************************************

unsigned char Strobe(unsigned char strobe)
{
    unsigned char statusByte = 0;
    unsigned int  gdo_state;

    // Check for valid strobe command
    if((strobe == 0xBD) || ((strobe >= RF_SRES) && (strobe <= RF_SNOP)))
    {
        // Clear the Status read flag
        RF1AIFCTL1 &= ~(RFSTATIFG);

        // Wait for radio to be ready for next instruction
        while( !(RF1AIFCTL1 & RFINSTRIFG));

        // Write the strobe instruction
        if ((strobe > RF_SRES) && (strobe < RF_SNOP))
        {
            gdo_state = ReadSingleReg(IOCFG2);    // buffer IOCFG2 state
            WriteSingleReg(IOCFG2, 0x29);         // chip-ready to GDO2
            RF1AINSTRB = strobe;
            if ( (RF1AIN&0x04)== 0x04 )           // chip at sleep mode
            {
                if ( (strobe == RF_SXOFF) || (strobe == RF_SPWD) || (strobe == RF_SWOR) ) { }
                else
                {
                    while ( (RF1AIN&0x04)== 0x04);  // chip-ready ?
                    // Delay for ~810usec at 1.05MHz CPU clock, see erratum RF1A7
                    __delay_cycles(850);
                }
            }
            WriteSingleReg(IOCFG2, gdo_state);    // restore IOCFG2 setting
        }
        else
        {
            // chip active mode (SRES)
            RF1AINSTRB = strobe;
            statusByte = RF1ASTATB;
        }
    }
    return statusByte;
}
unsigned char ReadSingleReg(unsigned char addr)
{
    unsigned char data_out;
    // Check for valid configuration register address, 0x3E refers to PATABLE
    if ((addr <= 0x2E) || (addr == 0x3E))
        // Send address + Instruction + 1 dummy byte (auto-read)
        RF1AINSTR1B = (addr | RF_SNGLREGRD);
    else
        // Send address + Instruction + 1 dummy byte (auto-read)
        RF1AINSTR1B = (addr | RF_STATREGRD);

    while (!(RF1AIFCTL1 & RFDOUTIFG));
    data_out = RF1ADOUTB;                   // Read data and clears the RFDOUTIFG
    return data_out;
}

void WriteSingleReg(unsigned char addr, unsigned char value)
{
    while (!(RF1AIFCTL1 & RFINSTRIFG));       // Wait for Radio to be ready for next instruction
    RF1AINSTRB = (addr | RF_SNGLREGWR);       // Send address + Instruction
    RF1ADINB = value;                        // Write data in
    __no_operation();
}

void ReadBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count)
{
    unsigned int i;
    if(count > 0)
    {
        while (!(RF1AIFCTL1 & RFINSTRIFG)); // Wait for INSTRIFG
        RF1AINSTR1B = (addr | RF_REGRD);          // Send addr of first conf. reg. to be read
        // ... and the burst-register read instruction
for (i = 0; i < (count-1); i++)
{
    while (!((RFDOUTFIFG&RF1AIFCTL1))); // Wait for the Radio Core to update the RF1ADOUTB reg
    buffer[i] = RF1ADOUT1B; // Read DOUT from Radio Core + clears RFDOUTIFG
        // Also initiates auo-read for next DOUT byte
} buffer[count-1] = RF1ADOUT0B; // Store the last DOUT from Radio Core

} // ***************************************************************************
}

// ***************************************************************************
// @fn          WriteBurstReg
// @brief       Write multiple bytes to the radio registers
// @param       unsigned char addr      Beginning address of burst write
// @param       unsigned char *buffer   Pointer to data table
// @param       unsigned char count     Number of bytes to be written
// @return      none
// ***************************************************************************
void WriteBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count)
{
    unsigned char i;
    if(count > 0)
    {
        while (!((RF1AIFCTL1 & RFINSTRIFG))); // Wait for the Radio to be ready for next instruction
        RF1AINSTRW = ((addr | RF_MUTER)<<8 ) + buffer[0]; // Send address + Instruction
        for (i = 1; i < count; i++)
        {
            RF1ADINB = buffer[i]; // Send data
            while (!((RFDINIFG & RF1AIFCTL1)); // Wait for TX to finish
        } i = RF1ADOUTB; // Reset RFDOUTIFG flag which contains status byte
    }

} // ***************************************************************************
// @fn          ResetRadioCore
// @brief       Reset the radio core using RF_SRES command
// @param       none
// @return      none
// ***************************************************************************
void ResetRadioCore (void)
{
    Strobe(RF_SRES); // Reset the Radio Core
    Strobe(RF_SNOP); // Reset Radio Pointer
}

} // ***************************************************************************
// @fn          WriteRfSettings
// @brief       Write the minimum set of RF configuration register settings
// @param       RFSETTINGS *pRfSettings  Pointer to the structure that holds the rf settings
// @return      none
// ***************************************************************************
void WriteRfSettings(RF_SETTINGS *pRfSettings) {
    WriteSingleReg(FSCTRL1, pRfSettings->fsctrl1);
    WriteSingleReg(FSCTRL0, pRfSettings->fsctrl0);
WriteSingleReg(FREQ2, pRfSettings->freq2);
WriteSingleReg(FREQ1, pRfSettings->freq1);
WriteSingleReg(FREQ0, pRfSettings->freq0);
WriteSingleReg(MDMCFG4, pRfSettings->mdmcfg4);
WriteSingleReg(MDMCFG3, pRfSettings->mdmcfg3);
WriteSingleReg(MDMCFG2, pRfSettings->mdmcfg2);
WriteSingleReg(MDMCFG1, pRfSettings->mdmcfg1);
WriteSingleReg(MDMCFG0, pRfSettings->mdmcfg0);
WriteSingleReg(CHANNR, pRfSettings->channr);
WriteSingleReg(DEVIATN, pRfSettings->deviatn);
WriteSingleReg(FREND1, pRfSettings->frend1);
WriteSingleReg(FREND0, pRfSettings->frend0);
WriteSingleReg(MCSM0, pRfSettings->mcsm0);
WriteSingleReg(FOCCFG, pRfSettings->foccfg);
WriteSingleReg(BSCFG, pRfSettings->bscfg);
WriteSingleReg(AGCCTRL2, pRfSettings->agcctrl2);
WriteSingleReg(AGCCTRL1, pRfSettings->agcctrl1);
WriteSingleReg(AGCCTRL0, pRfSettings->agcctrl0);
WriteSingleReg(FSCAL3, pRfSettings->fscal3);
WriteSingleReg(FSCAL2, pRfSettings->fscal2);
WriteSingleReg(FSCAL1, pRfSettings->fscal1);
WriteSingleReg(FSCAL0, pRfSettings->fscal0);
WriteSingleReg(FSTEST, pRfSettings->fstest);
WriteSingleReg(TEST2, pRfSettings->test2);
WriteSingleReg(TEST1, pRfSettings->test1);
WriteSingleReg(TEST0, pRfSettings->test0);
WriteSingleReg(FIFOTH, pRfSettings->fifoth);
WriteSingleReg(IOCFG2, pRfSettings->iocfg2);
WriteSingleReg(IOCFG0, pRfSettings->iocfg0);
WriteSingleReg(PKTCTRL1, pRfSettings->pktctrl1);
WriteSingleReg(PKTCTRL0, pRfSettings->pktctrl0);
WriteSingleReg(ADDR, pRfSettings->addr);
WriteSingleReg(PKTLEN, pRfSettings->pktlen);

void WriteBurstPATable(unsigned char *buffer, unsigned char count)
{
    while( !(RF1AIFCTL1 & RFINSTRIFG));
    RF1AINSTRW = 0x3E00 + value;  // PA Table single write

    while( !(RF1AIFCTL1 & RFINSTRIFG));
    RF1AINSTRB = RF_SNOP;  // reset PA_Table pointer
}

void WriteSinglePATable(unsigned char value)
{
    while( !(RF1AIFCTL1 & RFINSTRIFG));
    RF1AINSTRW = 0x3E00 + value;  // PA Table single write
}

void WritePATable(void)
{
    // Write data to power table
    // @brief Write data to power table
    // @param unsigned char value Value to write
    // @return none
    // ******************************************************************************
}

// *****************************************************************************
// @fn          WritePATable
// @brief       Write data to power table
// @param       unsigned char value
// @return      none
// *****************************************************************************/
// void WritePATable(unsigned char value)
// {
//     while( !(RF1AIFCTL1 & RFINSTRIFG));
//     RF1AINSTRW = 0x3E00 + value;
//     while( !(RF1AIFCTL1 & RFINSTRIFG));
//     RF1AINSTRB = RF_SNOP;
// }

// *****************************************************************************/
// @fn          WritePATable
// @brief       Write to multiple locations in power table
// @param       unsigned char *buffer Pointer to the table of values to be written
// @param       unsigned char count Number of values to be written
// @return      none
// *****************************************************************************/
// void WriteBurstPATable(unsigned char *buffer, unsigned char count)
volatile char i = 0;

while( !(RF1AIFCTL1 & RFINSTRIFG));
RF1AINSTRW = 0x7E00 + buffer[i]; // PA Table burst write

for (i = 1; i < count; i++)
{
    RF1ADINB = buffer[i]; // Send data
    while (!(RFDINIFG & RF1AIFCTL1)); // Wait for TX to finish
}
i = RF1ADOUTB; // Reset RFDOUTIFG flag which contains status byte

while( !(RF1AIFCTL1 & RFINSTRIFG));
RF1AINSTRB = RF_SNOP; // reset PA Table pointer

RF1a.h

/* **************************************************************************************************
 *                                          Defines
 * **************************************************************************************************
 */
// Chipcon
// Product = CC430Fx13x
// Chip version = A (HW Rev. 0x10)
// Crystal accuracy = 10 ppm
// X-tal frequency = 26 MHz
// RF output power = 0 dBm
// RX filterbandwidth = 101.562500 kHz
// Deviation = 19 kHz
// Datarate = 9.6 kBaud
// Modulation = 2-FSK
// Manchester enable = (0) Manchester disabled
// RF Frequency = 435.999939 MHz
// Channel spacing = 199.951172 kHz
// Channel number = 0
// Optimization = -
// Sync mode = (0) No preamble/sync
// Format of RX/TX data = (1) Serial Synchronous mode, used for backwards compatibility
// CRC operation = (0) CRC disabled for TX and RX
// Forward Error Correction = (0) FEC disabled
// Length configuration = (2) Enable infinite length packets.
// Packetlength = 255
// Preamble count = No preamble
// Append status = 1
// Address check = (0) No address check
// FIFO autoflush = 0
// Device address = 0
// GDO0 signal selection = (45) GDO0_Z_EN_N
// GDO2 signal selection = (11) Serial Clock

typedef struct S_RF_SETTINGS {
    unsigned char fsctrl1; // Frequency synthesizer control.
    unsigned char fsctrl0; // Frequency synthesizer control.
    unsigned char freq2;    // Frequency control word, high byte.
    unsigned char freq1;    // Frequency control word, middle byte.
}
unsigned char freq0; // Frequency control word, low byte.
unsigned char mdmcfg4; // Modem configuration.
unsigned char mdmcfg3; // Modem configuration.
unsigned char mdmcfg2; // Modem configuration.
unsigned char mdmcfg1; // Modem configuration.
unsigned char mdmcfg0; // Modem configuration.
unsigned char channr; // Channel number.
unsigned char deviatn; // Modem deviation setting (when FSK modulation is enabled).
unsigned char frend1; // Front end RX configuration.
unsigned char frend0; // Front end RX configuration.
unsigned char mcsm0; // Main Radio Control State Machine configuration.
unsigned char foccfg; // Frequency Offset Compensation Configuration.
unsigned char bscfg; // Bit synchronization Configuration.
unsigned char agcctrl2; // AGC control.
unsigned char agcctrl1; // AGC control.
unsigned char agcctrl0; // AGC control.
unsigned char fscal3; // Frequency synthesizer calibration.
unsigned char fscal2; // Frequency synthesizer calibration.
unsigned char fscal1; // Frequency synthesizer calibration.
unsigned char fscal0; // Frequency synthesizer calibration.
unsigned char fstest; // Frequency synthesizer calibration control
unsigned char test2; // Various test settings.
unsigned char test1; // Various test settings.
unsigned char test0; // Various test settings.
unsigned char fifothr; // RXFIFO and TXFIFO thresholds.
unsigned char iocfg2; // GDO2 output pin configuration
unsigned char iocfg0; // GDO0 output pin configuration
unsigned char pktctrl1; // Packet automation control.
unsigned char pktctrl0; // Packet automation control.
unsigned char addr; // Device address.
unsigned char pktlen; // Device address.
}
RF_SETTINGS;

void ResetRadioCore (void);
unsigned char Strobe(unsigned char strobe);
unsigned char Strobe(unsigned char strobe);

void WriteRfSettings(RF_SETTINGS *pRfSettings);
void WriteSingleReg(unsigned char addr, unsigned char value);
void WriteBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count);
unsigned char ReadSingleReg(unsigned char addr);
void ReadBurstReg(unsigned char addr, unsigned char *buffer, unsigned char count);
void WriteSinglePATable(unsigned char value);
void WriteBurstPATable(unsigned char *buffer, unsigned char count);

Hal_pmm.c

//****************************************************************************//
// Function Library for setting the PMM
//   File: hal_pmm.c
////
// Texas Instruments
//
// Version 1.2
// 11/24/09
//
// V1.0  Initial Version
// V1.1 Adjustment to UG
// V1.2 Added return values

//****************************************************************************/
/* *******************************************
* THIS PROGRAM IS PROVIDED "AS IS". TI MAKES NO WARRANTIES OR
* REPRESENTATIONS, EITHER EXPRESS, IMPLIED OR STATUTORY,
* INCLUDING ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS
* FOR A PARTICULAR PURPOSE, LACK OF VIRUSES, ACCURACY OR
* COMPLETENESS OF RESPONSES, RESULTS AND LACK OF NEGLIGENCE.
* TI DISCLAIMS ANY WARRANTY OF TITLE, QUIET ENJOYMENT, QUIET
* POSSESSION, AND NON-INFRINGEMENT OF ANY THIRD PARTY
* INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO THE PROGRAM OR
* YOUR USE OF THE PROGRAM.
*
* IN NO EVENT SHALL TI BE LIABLE FOR ANY SPECIAL, INCIDENTAL,
* CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, ON ANY
* THEORY OF LIABILITY AND WHETHER OR NOT TI HAS BEEN ADVISED
* OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT
* OF THIS AGREEMENT, THE PROGRAM, OR YOUR USE OF THE PROGRAM.
* EXCLUDED DAMAGES INCLUDE, BUT ARE NOT LIMITED TO, COST OF
* REMOVAL OR REINSTALLATION, COMPUTER TIME, LABOR COSTS, LOSS
* OF GOODWILL, LOSS OF PROFITS, LOSS OF SAVINGS, OR LOSS OF
* USE OR INTERRUPTION OF BUSINESS. IN NO EVENT WILL TI'S
* AGGREGATE LIABILITY UNDER THIS AGREEMENT OR ARISING OUT OF
* YOUR USE OF THE PROGRAM EXCEED FIVE HUNDRED DOLLARS
* (U.S.$500).
*
* Unless otherwise stated, the Program written and copyrighted
* by Texas Instruments is distributed as "freeware". You may,
* only under TI's copyright in the Program, use and modify the
* Program without any charge or restriction. You may
* distribute to third parties, provided that you transfer a
* copy of this license to the third party and the third party
* agrees to these terms by its first use of the Program. You
* must reproduce the copyright notice and any other legend of
* ownership on each copy or partial copy, of the Program.
*
* You acknowledge and agree that the Program contains
* copyrighted material, trade secrets and other TI proprietary
* information and is protected by copyright laws,
* international copyright treaties, and trade secret laws, as
* well as other intellectual property laws. To protect TI's
* rights in the Program, you agree not to decompile, reverse
* engineer, disassemble or otherwise translate any object code
* versions of the Program to a human-readable form. You agree
* that in no event will you alter, remove or destroy any
* copyright notice included in the Program. TI reserves all
* rights not specifically granted under this license. Except
* as specifically provided herein, nothing in this agreement
* shall be construed as conferring by implication, estoppel,
* or otherwise, upon you, any license or other right under any
* TI patents, copyrights or trade secrets.
*
* You may not use the Program in non-TI devices.
*/
#include "cc430f6137.h"
#include "hal_pmm.h"

#define _HAL_PMM_DISABLE_SVML_
#define _HAL_PMM_DISABLE_SVSL_
#define _HAL_PMM_DISABLE_FULL_PERFORMANCE_

//****************************************************************************//
#ifdef _HAL_PMM_DISABLE_SVML_
#define _HAL_PMM_SVMLE SVMLE
#else
#define _HAL_PMM_SVMLE 0
#endif
#ifdef _HAL_PMM_DISABLE_SVSL_
#define _HAL_PMM_SVSLE SVSLE
#else
#define _HAL_PMM_SVSLE 0
#endif
#ifdef _HAL_PMM_DISABLE_FULL_PERFORMANCE_
#define _HAL_PMM_SVSFP SVSFP
#else
#define _HAL_PMM_SVSFP 0
#endif
//****************************************************************************//
// Set VCore
//****************************************************************************//
unsigned int SetVCore (unsigned char level)
{
    unsigned int actlevel;
    unsigned int status = 0;
    level &= PMMCOREV_3;                       // Set Mask for Max. level
    actlevel = (PMMCTL0 & PMMCOREV_3);         // Get actual VCore
    while (((level != actlevel) && (status == 0)) || (level < actlevel))  // step by step increase or decrease
    {
        if (level > actlevel)
            status = SetVCoreUp(++actlevel);
        else
            status = SetVCoreDown(--actlevel);
    }
    return status;
}
//****************************************************************************//
// Set VCore Up
//****************************************************************************//
unsigned int SetVCoreUp (unsigned char level)
{
    unsigned int PMMRIE_backup,SVSMHCTL_backup;
    // Open PMM registers for write access
    PMMCTL0_H = 0xA5;
    PMMRIE_backup = PMMRIE;
    // Disable dedicated Interrupts to prevent that needed flags will be cleared
    PMMRIE_backup = PMMRIE;
    return status;
}
PMMRIE &= ~(SVSMDLYIE | SVMLVLRIE | SVMHVLRIE | SVMHVLRIE | SVSMHRRL0 * level);

// Set SVM highside to new level and check if a VCore increase is possible
SVSMHCTL_backup = SVSMHCTL;

// Set SVM highside to new level and check if a VCore increase is possible
PMMIFG &= ~(SVSMHIFG | SVSMDLYIFG);
SVSMHCTL = SVMHE | SVMHF | (SVSMHRL0 * level);

// Wait until SVM highside is settled
while ((PMMIFG & SVSMDLYIFG) == 0);

// Disable full-performance mode to save energy
SVSMHCTL &= ~_HAL_PMM_SVSFP;

// Check if a VCore increase is possible
if ((PMMIFG & SVSMHIFG) == SVSMHIFG);

// Vcc is to low for a Vcore increase
  // recover the previous settings
  PMMIFG &= ~SVSMDLYIFG;
  SVSMHCTL = SVSMHCTL_backup;

// Wait until SVM highside is settled
while ((PMMIFG & SVSMDLYIFG) == 0);

// Clear all Flags
PMMIFG &= ~SVSMDLYIFG;

// Set also SVS highside to new level
SVSMDLYIE |= SVSHE | (SVSRL0 * level);

// Set SVM low side to new level
SVSMLCTL = SVMLE | SVMLF | (SVSMLRL0 * level);

// Wait until SVM low side is settled
while ((PMMIFG & SVSMDLYIFG) == 0);

// Clear already set flags
PMMIFG &= ~(SVMLVLRIE | SVMLIFG | SVMLVLRIFG |

// Set VCore to new level
PMMCT0_L = PMMCT0_L * level;

// Wait until new level reached
if (PMMIFG & SVMLVLRIE) while ((PMMIFG & SVMLVLRIE) == 0);

// Set also SVS/SVM low side to new level
PMMIFG &= ~SVMVLRL0 * level;
SVSMLCTL |= SVSLE | (SVSRL0 * level);

// Wait for lowside delay flags
while ((PMMIFG & SVSMDLYIFG) == 0);

// Disable SVS/SVM Low

// Disable full-performance mode to save energy
SVSMLCTL &= ~(_HAL_PMM_DISABLE_SVSL + _HAL_PMM_DISABLE_SVML + _HAL_PMM_SVSFP);

// Clear all Flags
PMMIFG &= ~(SVMLVLRIE | SVMLIFG | SVSMDLYIFG | SVMLVLRIE | SVMLVLRIE | SVMLVLRIE | SVSMDLYIFG);

// backup PMM-Interrupt-Register
PMMRIE = PMMRIE_backup;

// Lock PMM registers for write access
PMMCT0_H = 0x00;

return PMM_STATUS_ERROR;                       // return: voltage not set

// Vcc is high enough for a Vcore increase

// Set also SVS highside to new level
SVSMDLYIE |= SVSHE | (SVSRL0 * level);

// Set SVM low side to new level
SVSMLCTL = SVMLE | SVMLF | (SVSMLRL0 * level);

// Wait until SVM low side is settled
while ((PMMIFG & SVSMDLYIFG) == 0);

// Clear already set flags
PMMIFG &= ~(SVMLVLRIE | SVMLIFG | SVMLVLRIFG |

// Set VCore to new level
PMMCT0_L = PMMCT0_L * level;

// Wait until new level reached
if (PMMIFG & SVMLVLRIE) while ((PMMIFG & SVMLVLRIE) == 0);

// Set also SVS/SVM low side to new level
PMMIFG &= ~SVMVLRL0 * level;
SVSMLCTL |= SVSLE | (SVSRL0 * level);

// wait for lowside delay flags
while ((PMMIFG & SVSMDLYIFG) == 0);

// Disable SVS/SVM Low

// Disable full-performance mode to save energy
SVSMLCTL &= ~(_HAL_PMM_DISABLE_SVSL + _HAL_PMM_DISABLE_SVML + _HAL_PMM_SVSFP);

// Clear all Flags
PMMIFG &= ~(SVMLVLRIE | SVMLIFG | SVSMDLYIFG | SVMLVLRIE | SVMLVLRIE | SVMLVLRIE | SVSMDLYIFG);

// backup PMM-Interrupt-Register
PMMRIE = PMMRIE_backup;

// Lock PMM registers for write access
PMMCT0_H = 0x00;
return PMM_STATUS_OK; // return: OK
}

//****************************************************************************//
// Set VCore down (Independent from the enabled Interrupts in PMMRIE)
//****************************************************************************//
unsigned int SetVCoreDown (unsigned char level)
{
    unsigned int PMMRIE_backup;

    // Open PMM registers for write access
    PMMCTL0_H = 0xA5;

    // Disable dedicated Interrupts to prevent that needed flags will be cleared
    PMMRIE &= ~(SVSMHDLIE | SVSMLDLIE | SVMLVLRIE | SVMHVLRIE | SVMHVLRPE);

    // Set SVM high side and SVM low side to new level
    PMMIFG &= ~((SVMHIFG | SVSMHDLIFG | SVSLIFG | SVSMLDLIFG);
    SVSMHCTL = SVMHE | SVMHFP | (SVSMHRRL0 * level);
    SVSMLCTL = SVSLE | SVSLFP | (SVSMLRL0 * level);
    // Wait until SVM high side and SVM low side is settled
    while ((PMMIFG & SVSMHDLYIFG) == 0 || (PMMIFG & SVSMLDLYIFG) == 0);

    // Set VCore to new level
    PMMCTL0_L = PMMCOREV0 * level;

    // Set also SVS highside and SVS low side to new level
    PMMIFG &= ~((SVSHIFG | SVSMHDLIFG | SVSLIFG | SVSMLDLIFG);
    SVSMHCTL |= SVSHE | SVSHFP | (SVSHRVL0 * level);
    SVSMLCTL |= SVSLE | SVSLFP | (SVSLRVL0 * level);
    // Wait until SVS high side and SVS low side is settled
    while ((PMMIFG & SVSMHDLYIFG) == 0 || (PMMIFG & SVSMLDLYIFG) == 0);
    // Disable full-performance mode to save energy
    SVSMHCTL &= ~HAL_PMM_SVSFP;

    // Disable SVS/SVM Low
    // Disable full-performance mode to save energy
    SVSMLCTL &= ~HAL_PMM_DISABLE_SVSL + HAL_PMM_DISABLE_SVML + HAL_PMM_SVSFP;

    // Clear all Flags
    PMMIFG &= ~(SVHVLRIFG | SVMHIFG | SVSMHDLIFG | SVMLVLRIFG | SVMLIFG | SVSMLDLIFG);
    // backup PMM-Interrupt-Register
    PMMRIE = PMMRIE_backup;
    // Lock PMM registers for write access
    PMMCTL0_H = 0x00;

    if ((PMMIFG & SVMHIFG) == SVMHIFG)
        return PMM_STATUS_ERROR; // Highside is still too low for the adjusted VCore Level
    else return PMM_STATUS_OK; // Return: OK
}
#ifndef __PMM
#define __PMM

#define PMM_STATUS_OK 0
#define PMM_STATUS_ERROR 1

unsigned int SetVCore (unsigned char level);

unsigned int SetVCoreUp (unsigned char level);

unsigned int SetVCoreDown (unsigned char level);
unsigned int SetVCoreDown (unsigned char level);

#endif /* __PMM */

**Visual interface**

Public Class Form1

    Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles MyBase.Load
        ComboBox1.Items.Add("COM1")
        ComboBox1.Items.Add("COM2")
        ComboBox1.Items.Add("COM3")
    End Sub

    Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles ComboBox1.SelectedIndexChanged
        Select Case ComboBox1.SelectedIndex
            Case 0
                SerialPort1.PortName = "COM1"
            Case 1
                SerialPort1.PortName = "COM2"
            Case 2
                SerialPort1.PortName = "COM3"
        End Select
    End Sub

    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button1.Click
        If Not SerialPort1.IsOpen Then
            SerialPort1.Open()
            MsgBox("CONECTADO", MsgBoxStyle.Exclamation)
        End If
    End Sub

    Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button2.Click
        If SerialPort1.IsOpen = True Then
            SerialPort1.Close()
            MsgBox("DESCONECTADO", MsgBoxStyle.Exclamation)
        End If
    End Sub

    Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button3.Click
        Dim AUDIO As Long
        AUDIO = 1
        SerialPort1.Write((AUDIO))
    End Sub

    Private Sub Button4_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles Button4.Click
        Dim TNC31 As Long
        TNC31 = 2
        SerialPort1.Write((TNC31))
    End Sub
Private Sub Button5_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button5.Click
Dim TNC7 As Long
TNC7 = 3
SerialPort1.Write(TNC7)
End Sub

Private Sub Button6_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles Button6.Click
Dim AUDIO As Long
AUDIO = 4
SerialPort1.Write(AUDIO)
End Sub
End Class

CCS compiler PIC 18F2550

#include <18f2550.h> //archivo de cabecera
#include <stdlib.h> //libreria de cabecera
#define use_portb_lcd TRUE //Utilizar el puerto_B para el LCD
#define USB_HID_DEVICE FALSE //deshabilitamos el uso de las directivas HID
#define USB_EP1_TX_ENABLE USB_ENABLE_BULK //turn on EP1(EndPoint1) for IN
#define USB_EP1_RX_ENABLE USB_ENABLE_BULK //turn on EP1(EndPoint1) for OUT
#define USB_EP1_TX_SIZE 1 //size to allocate for the tx endpoint 1 buffer
#define USB_EP1_RX_SIZE 3 //size to allocate for the rx endpoint 1 buffer
#include <pic18_usb.h> //Microchip PIC18Fxx5x Hardware layer for CCS's PIC USB driver
#include <PicUSB.h> //Configuración del USB y los descriptores para este dispositivo
#include <usb.c> //handles usb setup tokens and get descriptor reports

int8 data;
void main(void) {
    //------------------------DE LA PARTE MANUAL------------------------
    unsigned char led = 0; //Inicializamos el LCD
    // desactivando interrupciones globales
    disable_interrupts(globals); // desactivando puertos analogicos
    // configurar salidas y entradas [salidas(0), entradas(1)]
    set_tris_a(0x0001000); // configurando todos salidas excepto pin_a3
    set_tris_b(0x00); // configurando todo como salidas para el LCD
    set_tris_c(0x11111100); // configurando los puertos pin_c0, pin_c1, pin_c2 como entradas y el resto como salidas
    // limpiando todos los puertos
    LIMPIAR_A;
    LIMPIAR_B;
    LIMPIAR_C;
    // estado inicial
    TURN_OFF(SW1);
    TURN_OFF(SW2);
    //-------------------Para el control por medio del USB-------------------
    TURN_OFF(LEDV);
    TURN_ON(LEDR);
    // inicializamos el USB
    usb_init();
    // habilita periferico usb e interrupciones
    // usb_wait_for Enumeration(); // esperamos hasta que el PicUSB sea configurado por el host
    TURN_OFF(LEDR);
    TURN_ON(LEDV); // encendemos led verde
    // while (!usb_connected()) {} // while (TRUE)
    if (usb_enumerated()) // si el PicUSB está configurado
    {
        if (usb_kbhit(1)) // si el endpoint de salida contiene datos del host
        {
            usb_get_packet(1, data, 8); // cojemos el paquete de tamaño 3bytes del EP1 y almacenamos en recibe
        }
        switch (data) {
        case 1: TURN_OFF(SW1);
            TURN_OFF(SW2);
            break;
        case 2: TURN_OFF(SW1);
            TURN_ON(SW2);
            break;
        case 3: TURN_ON(SW1);
            TURN_OFF(SW3);
            break;
        case 4: TURN_ON(SW2);
            TURN_ON(SW3);
break;
default:TURN_OFF(SW1);
    TURN_OFF(SW2);
break; }
}
//----------------------------Para el switching manual---------------------------------
if(SELEC==0)  //si se preciona "seleccionar"
{
    led++;
    switch(led)
    {
    //Al primer pulso:
    case 1:
        LIMPIAR_A;
        TURN_OFF(SW1);
        TURN_ON(SW2);
        break;
    //Al segundo pulso:
    case 2:
        LIMPIAR_A;
        TURN_ON(SW1);
        TURN_OFF(SW2);
        break;
    //Al tercer pulso:
    case 3:
        LIMPIAR_A;
        TURN_ON(SW1);
        TURN_ON(SW3);
        break;
    //Por defecto: siempre aterrado (reset)
    default:
        TURN_OFF(SW1);
        TURN_OFF(SW2);
        led=0;             //limpiamos la variable que cuenta los pulsos
        break;
    }
while ((SELEC==0));
while ((SELEC==1));
}
//-------------------Pantalla LCD-------------------------------------
//Caso por defecto:
    lcd_gotoxy(1,1);
    printf(lcd_putchar," Estado Inicial \n");
    lcd_gotoxy(1,2);
    printf(lcd_putchar,"     (RESET) \n");
//Para los demas casos:
//Si switch en TNC7:
    if (IN_SW1==1 && IN_SW2==1 && IN_SW3==0)
    {
        lcd_gotoxy(1,1);
        printf(lcd_putchar," Enlazado con... \n");
        lcd_gotoxy(1,2);
        printf(lcd_putchar,"      (TNC7) \n");
    }
//Si switch en TNC31:
if (IN_SW1==1 && IN_SW2==0 && IN_SW3==1)
{
    lcd_gotoxy(1,1);
    printf(lcd_putchar," Enlazado con...
    ");
    lcd_gotoxy(1,2);
    printf(lcd_putchar," (TNC31)  
    ");
}
//Si switch en Audio:
if (IN_SW1==0 && IN_SW2==1 && IN_SW3==1)
{
    lcd_gotoxy(1,1);
    printf(lcd_putchar," Enlazado con...
    ");
    lcd_gotoxy(1,2);
    printf(lcd_putchar," Audio  
    ");
}
//Si switch en RESET:
if (IN_SW1==0 && IN_SW2==1 && IN_SW3==0)
{
    lcd_gotoxy(1,1);
    printf(lcd_putchar," Estado Inicial
    ");
    lcd_gotoxy(1,2);
    printf(lcd_putchar," (RESET)  
    ");
}