MASTER THESIS

TITLE: A multi-agent payload management approach for femtosatellite applications

MASTER DEGREE: Master of Science in Telecommunications Engineering and Management

AUTHOR: Lara Navarro Morales

DIRECTOR: Joshua Tristancho Martínez

DATE: September 15, 2011

Título : A multi-agent payload management approach for femtosatellite applications

Autor: Lara Navarro Morales

Director: Joshua Tristancho Martínez

Fecha: September 15, 2011

Resumen

La reducción en tamaño de los componentes electrónicos hace posible la construcción de satélites realmente pequeños como los femtosatélites (satélites con una masa inferior a 100 gramos).
La principal ventaja de este tiepo de satélites es que proporcionan un punto de vista múltiple cuando trabajan en enjambre o dentro una constelación. La complejidad de este tipo de red de sensores, añadido al bajo consumo de potencia y al tamaño reducido de los nodos, require una buena estrategia de gestión de recursos que se pretende presentar en este trabajo.
El paradigma de gestión de agente consiste en un punto de vista simple, con alta calidad, y diversos puntos de vista múltples con una calidad menor. La conmutación de un punto de vista a otro se realiza de forma externa a la red o se lleva a cabo siguiendo una ley básica. Este enfoque permite una buena optimización del ancho de banda. Del mismo modo, permite una distribución de tareas en la red en la que hay un único agente recibiendo, otro transmitiendo y el resto trabajan como nodos.

Palabras clave: Multi agente, Femtosatélite, PicoRover, Micro cámara, System-On-Chip

Title : A multi-agent payload management approach for femtosatellite applications

Author: Lara Navarro Morales

Director: Joshua Tristancho Martínez

Date: September 15, 2011

Overview

The reduction in size of electronic components makes feasible really small satellites like Femtosatellites with are less than 100 grams of mass.
The main advantage of this kind of satellite is the multipoint of view when they work as swarm or a inside a constellation. The complexity of these kind of network sensors in addition to the low power and low size requires a good strategy of management that we want to present in this work.
The paradigm of agent management consists of a single point high quality point of view and multipoint low quality point of view where the switching for the selected point of view is done externally to the network or done by a basic law. This approach allow a good optimization of the bandwidth instead of streaming every points of view in high quality. At the same time, this approach allows a task distribution in the network where there is only one acquiring agent, one transmitting agent and the rest of agents working as a node agent.

Keywords: Multi-agent, Femtosatellite, PicoRover, Micro-camera, System-On-Chip

Acknowledgements

I'm grateful to my parents, my sister and my friends for encouraging and supporting me during this work.

I'm very grateful to my tutor Joshua Tristancho for the opportunity to do this research as well as his help and inspiration in this work.

I want to thank members of the WikiSat research group who helped me with this work developing the femtosatellite, providing the supplies and prototyping the boards, specially Joshua Tristancho, Sonia Pérez and Jordi Gutiérrez.

I'm very grateful to people who helped and supported me, specially Esteve Bardolet, Raquel González, Joan Naudó, Enric Fernández, Roberto Rodríguez, Juan Martínez, Victor Kravchenko and Javier Pérez.

I want to thank the EETAC school for the laboratories and equipment they have provided to support this research.

Special thanks to Luis Izquierdo from the Universidad Nebrija and Ángel Esteban from the D47 company for their interest in this research and the opportunity to participate in the TURIZMAP project funded by AVANZA grant, project code F-00300.

Glossary

ADC	Analog-to-Digital Converter
BIAS	Biased error
BOM	Bill-of-Material
CAD	Computer-Aided Design
CAM	Computer-Aided Manufacturing
CLK	Clock signal
CLKINH	Clock inhibit signal
CNC	Computer Numerical Control
COTS	Commercial-of-the-Shelf
DAQ	Digital Acquisition
DoF	Degrees of Freedom
EEPROM	Electrically Erasable Programmable Read-Only Memory
EMC	Electromagnetic Compatibility
EXTCLK	External clock signal
FIFO	First In First Out
GFKS	Gaussian Frequency-Shift Keying modulation
HAL	Hardware Abstraction Layer
HD	High Definition
HGA	High Gain Antenna
I2C	Inter-Integrated Circuit
IC	Integrated Circuit
IMU	Inertial Measurement Unit
LASER	Light Amplification by Stimulated Emission of Radiation
LEO	Low Earth Orbit
LNA	Low Noise Amplifier
LOS	Line-of-Sight
LSB	Less Significant Bit
MCU	Main Control Unit
MEMS	Micro-Electromechanical System
MSB	Most Significant Bit
OIP3	Output Intercept Point at 3 dB
P1dB	Input power at 1 dB
PA	Power Amplifier
PCB	Printed Circuit Board
PLF	Polarization Loss Factor
PWM	Pulse With Modulation
RF	Radio-Frequency
SAA	South Atlantic Anomaly
SCL	Serial Clock
SDA	Serial Data Signal
SEE	Single Event Effect
SH/LD	Serializer/Load signal
SMD	Surface Mounting Device
SoC	System-on-Chip
SPI	Serial Peripheral Interface bus

SRAM	Shadow RAM memory
TTC	Telemetry, Tracking and Commanding
UART	Universal Asynchronous Receiver-Transmitter

CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

For a given agent network, the paradigm of management-on-the-agent sets that the responsibility of streaming management goes through to the agent and not through the network control. In example, if these agents are a set of wireless cameras, the paradigm of management-on-the-agent consist on a single high quality point of view and many low quality multi-point of view. The switching for the selected point of view is done externally to the network or by a basic law in such a way that the bandwidth is optimized. Instead of streaming every point of view in high quality mode only one is used. An important strength of this approach is that allows a task distribution on the network where there is only one acquiring agent, one transmitting agent and the rest of agents work as a node agent. The term ”point of view” here is used not only in terms of a camera but also in terms of knowing information about how, when and where the point of view has been taken.

Traditional approach for Recording Studios was to stream every single point in high quality to a master control panel where in that moment or later, the switching action is done. Final result is a sequence of these high quality points of view. With this traditional approach, there is a huge waste of resources only affordable if transmission lines are wired. If the transmission line is a wireless link, this traditional approach does not makes sense. The management-on-the-agent paradigm proposes to break with this approach and to make more efficient the use of the resources or available bandwidth.

In [5] a multi-agent adaptive protocol for femtosatellite applications is proposed. Hence, the proposal of this protocol is to define which agent is the Producer and which is the Consumer thus the rest of agents will work as a Relay. Figure 1 shows the three study cases proposed: Direct link case, a Relayed link case and a Multi-path case.

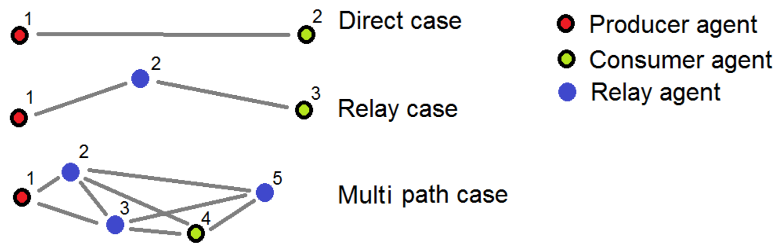


Figure 1: Direct case, Relay case and Multi-path case

The first one,the direct case, is when the Consumer agent is closer enough to the Producer agent. Even if looks like the network is not required, it is not true because the aim of the Multi Agent System is to provide real-time information for each agent’s point of view to the Consumer agent. In this case it can be that the transmission is faster and no delay is introduced. Telemetry, Tracking and Command (TTC) should be sent from every node to the Consumer agent. A relay case is done when the Consumer agent can not reach the Producer directly but through other agent. In this case the streaming is done step by step and thus a delay is introduced. Finally, when multi-path case is considered, delay may very depending on the number of jumps. Infinity loops must be avoided.

Observing these three study cases, some basic ideas can be taken into account:

- TTC information of all nodes should be sent whatever the configuration is.
- The smaller jump number is, the better.
- In multi-path case, lower jump number is preferred.
- Need to avoid infinite loops.
- Need to ignore repeated information.

The *WikiSat* space program consists on implementing a low cost satellite for the N-Prize.

The satellite *WikiSat* is a less than 20 grams femtosatellite that will be the brain of the mission, it is responsible of ignite the rocket, its control, ignite the Stage 2, etc. These capabilities save job and weight because there is no need of a single control system for the rocket, the *WikiSat* is capable of doing these functions by itself.

Then the *WikiSat* is an essential part of the group and everything that makes the group has direct or indirect relation with it. The group have worked to allow that all femtosatellites can establish network and return information once they are in orbit, working like a constellation. There are four versions some of them are showed in Figure 2.

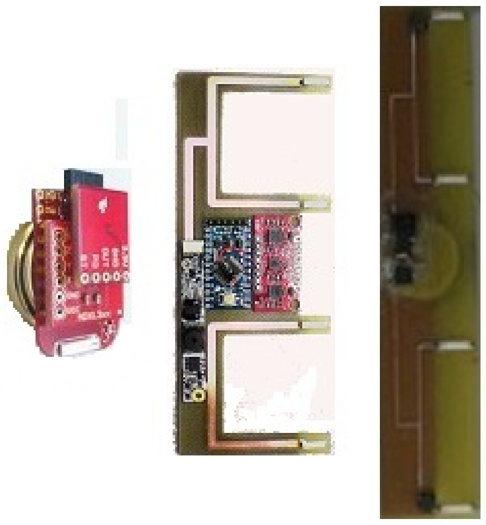


Figure 2: a)WikiSat V1, b)WikiSat V2, c)WikiSat V3

The Chapter 1 presents the state of the art in terms of femtosatellites as well as the technologies used on its design. On Chapter 2 a set of requirements to the femtosatellite technology capable of achieving the N-Prize goals are synthesized while on Chapter 3 the design considerations followed during the development of *WikiSat V4.1* are exposed. The femtosatellite link budget is presented in Chapter 4.

The technical implementation of the *WikiSat V4.1* will be described in the Chapter 5. The component selection and the hardware design is widely explained as well as the integration of the different subsystem and its validation. The Chapter 6 will be focused on the payload implementation.

As the concept of the paradigm of management-on-the-agent was developed enough in [5] this work will be focused on the development of an agent that follows this paradigm.

CHAPTER 1. RELATED WORK

1.1. Satellite classification

In general, a satellite is any object that orbits something else, as, for example, the Moon orbits the Earth. An artificial satellite is a device which has been placed into orbit by human endeavor.

First artificial satellite (Sputnik 1) was launched by the Soviet Union in the latest 1950s. Since then, thousands of satellites[1] have been launched into orbit around the Earth; also some satellites, notably space stations, have been launched in parts and assembled in orbit.

Different purposes can be developed by satellites. Common types include military and civilian Earth observation, communications, navigation, weather and research. Space stations and human spacecraft in orbit are also satellites. Satellite orbits vary greatly, depending on its purpose and are classified in a number of ways. Well-known (overlapping) classes include low Earth orbit, polar orbit, and geostationary orbit.

Satellites are usually semi-independent computer-controlled systems. Satellite subsystems attend many tasks, such as power generation, thermal control, telemetry, attitude control and orbit control. The satellites are usually classified by their mass as shown in Figure 1.1.

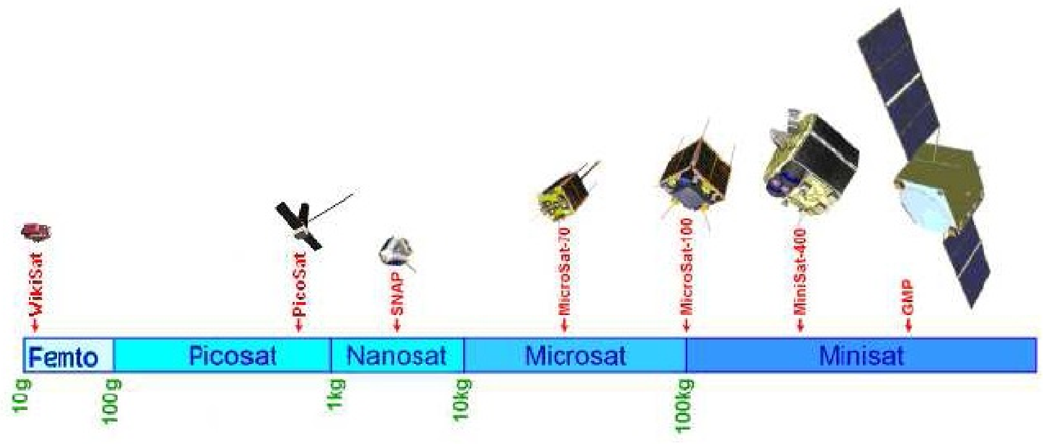


Figure 1.1: Scale of satellites as a function of mass

1.2. Femtosatellites

The idea of a complete satellite that weights less than 100 grams is not new; this category is called femtosatellite. Some authors like Helvajian and others have proposed a complete

[1] http://nssdc.gsfc.nasa.gov/nmc/spacecraftSearch.do

femtosatellite design in [8] and Barnhart in [3]. Also some femtosatellite designs were propose by the authors in [11] but to date, no femtosatellite was launched to the orbit.

This very promising category of satellite and its low weight, can severally reduce the launch cost. Also it is feasible to increment the number of satellites launched in a same event, i.e. swarms of these satellites can record same phenomena from different points of view; in addition it is possible to distribute the work load if they are interconnected as a sensor network. Of course this is only feasible if all the basic subsystems are implemented in such a small size. The key point is to use Commercial-of-the-shelf (COTS) that are commercially available like electronic components, i.e. Micro-Electro-Mechanical Systems (MEMS) but they need to be validated for the space environment. Many sensors had been assembled in a format of MEMS while others can not be implemented with this technology, i.e. huge telescopes or low frequency antennas. Basic knowledge on physics [9] says that antenna size depends on the working frequency while amount of power needed to transmit increases with a power of four of the frequency. The $2.4\,GHz$ frequency was selected for ground short distance applications but also for a Low Earth Orbit (LEO) by some authors like Doerksen in [13] up to $115\,kbps$ with a patch antenna or like Hamrouni proposed in [7] and by Hall in [6] as a download communication system. The wave length is about few centimeters but for a communication link from a LEO orbit with ground is the order of few watts. These basic parameters establish size and power limits for a femtosatellite design that is magnified for the technological limitations.

1.3. Technologies

The improvement of satellites is base on the use of new technologies. Although it is very hard to implement them because the space sector is a very conservative industry. Some technologies that are used in the process of designing a femtosatellite like this are stated bellow.

1.3.1. Micro-Electro-Mechanical Systems

The main improvement for femtosatellites is to based them on Micro-Electromechanical Systems (MEMS) that are available in the domestic market. These components should be validated for space use. Many of them can be used in hard conditions like our femtosatellite is going to resist. One of the main components for the femtosatellite is a 3 axes accelerometer single chip like the LIS331HH from *STMicroelectronics* that is used to guide the launcher during the trajectory. Other critical component is a high accurate, high range, three axes gyro like the ITG-3200 from *Invensense* that is used for short maneuvers for the camera or antenna pointing.

1.3.2. Printed Circuit Boards

The use of Printed Circuit Board (PCB) makes the design and implementation very easy and cheap. This technology is nowadays, very well expanded. It is feasible to design with open tools the whole satellite and for less that 100€a manufacturer will build and assemble your design in only few days. Additionally, the use of micro-strips allows to design and implement in the same platform many kinds of High Gain Antennas (HGA) using a technology of micro ceramic antennas array.

1.3.3. Surface Mount Devices Technology

Based on PCB, the use of Surface Mount Device (SMD) is another improvement in terms of weight saving, size reduction, high shock resistant and robustness. For the femtosatellite development we are interested in the use of these devices because they are easy to assemble during the re-flow. Many electronic components are available in this format, including a complete high definition camera.

1.3.4. CameraCube

CameraCube is a technology that integrates a whole camera (sensor, circuit and lens) inside a cube able to re-flow in a PCB. There is a new growing market in this sens for phone mobiles and i-Pod applications. A good example of this is a high definition camera VW6754 from *STMicroelectronics* that has $5x5x4mm$ and $1,600x1,200$ pixels. These cameras use a technology called wafer, a pin matrix is below the camera with bubbles. When high temperature is applied, these bubbles made of soldering past melt and welds the camera in the PCB. The level of integration is really high.

1.3.5. Inter-Integrated Circuit Bus

The I^2C bus or IIC stands for Inter-Integrated Circuit that is used to attach low-speed peripherals to a Main Control Unit (MCU). Many sensors are based on an Analog to Digital Converter (ADC) that is inside the sensor itself and data is provided in digital format through the I^2C bus. Two wires are only required to connect all the sensor in the femtosatellite. This fact results in a reduction in complexity and space saving in the PCB. Additionally, no calibrations or maintenance are required. The speed of this bus reaches up to 100 kbits per second so this bus is not suitable for the payload data streaming but for the payload control. In our case, the femtosatellite will have a dedicated streaming bus direct from the serializer to the transceiver.

CHAPTER 2. REQUIREMENTS

2.1. Architecture

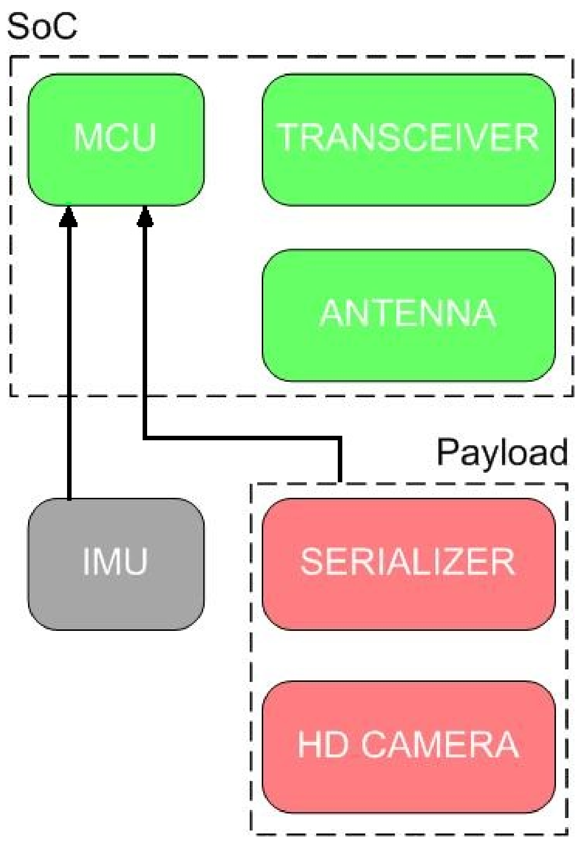


Figure 2.1: Femtosatellite block diagram for an imaging payload

The *WikiSat V4.1* is a Satellite-on-a-board; its design was based on some synergies to achieve such a low cost and low mass challenge. The structure of the satellite is a PCB (Figure 2.1) that works to hold the ceramic antenna array that was proposed by Fernandez-Murcia in [4] and at the same time works as a passive thermal control subsystem. This control is done in the design phase in terms of equilibrium between the incoming heat flow and the exit heat flow but adjusted by the area of two different materials, copper and glass fiber, that have opposite emissivity.

The femtosatellite performances turn around an Inertial Measurement Unit (IMU) designed, implemented and validated in near space by Bardolet Santacreu in [1], this IMU consist of two MEMS: a 3 axis accelerometers in the range of $24g$ with 16 bits of data resolution and a bandwidth of $500Hz$ and the other is a 3 axial gyros in the range of 2,000 degrees per second and a built-in temperature sensor. Both components are connected to a I^2C bus that is connected to the Main Control Unit (MCU) and other satellite devices. The idea of using the same IMU for the satellite and the launcher trajectory control was proposed by Tristancho in [17] as the Space Payload Paradigm is the integration of both, the launcher and the satellite in the design cycle of a space mission was proposed.

The satellite attitude control and the launcher vector control was studied by Navarro-Morcillo in [14] having the same inputs but different output. In order to reduce 10 times the launcher size, a balloon launching ramp was proposed by Bonet-Osorio in [2]. When a launch is done at an altitude of 35 kilometers; all the hard atmosphere and hazards are avoided and a large quantity of propellant is saved, hence safer and more often launches can be done weekly.

2.2. System requirements

In this section the mission requirements, as well as the femtosatellite ones, are presented. The *WikiSat* organization is the one in charge of the development and implementation of the femtosatellite. This organization has implemented on its designs the following directives and policies respect to it:

- Simplicity directive. The KISS rule stands for Keep It Simple and Safe.
- Absolute minimum directive, closer tolerances and a design only for its mission.
- No redundancy policy. Single fault tolerant system.
- Preferred configuration. Complete capabilities in the default configuration.

The system requirements are the following:

S0: The femtosatellite, as a system, shall stay in Low Earth Orbit for at least one week in order to allow the payload purpose before the end of this Master Thesis.

S1: The vector of the launcher used to launch the femtosatellite will be controlled by itself. The mini-launcher should have two stages and the bill of materials must be lower than $1.000£$, the femtosatellite is not included in this price.

S2: The femtosatellite should be operated by a development platform and the Earth control must be open source in order to guarantee the platform transparency.

2.3. High level requirements

HL00: The Power Supply subsystem shall provide electrical power for the computing of the orbit and the tracking.

HL01: The Communication subsystem shall transmit and receive the tracking and payload information.

HL02: The Structure subsystem shall be used protect the femtosatellite components and be used as a thermal path for thermal loads.

HL03: The Attitude determination subsystem shall determine the attitude by inertial means and be helped by optic sensors.

HL04: The Position determination subsystem shall determine the position in the orbit by inertial mean and be helped by optic sensors.

HL05: The Attitude control subsystem shall point the high gain antenna to the Earth in a passive way using the Earth's magnetic field.

HL06: The Tracking subsystem shall transmit its computed position to a ground station only when passing over the ground station's sky.

HL07: The Video recording subsystem shall record pictures and video if required.

2.4. Low level requirements

LL000: The battery shall provide enough power for the whole mission at any time and in peak power conditions for a limited period of time.

LL001: The electrical power subsystem shall be used in short periods of time having an idle model.

LL010: The ground communication link shall be disconnected when the ground electrical power source is not available.

LL011: The ground monitoring function shall be disconnected when the ground electrical power source is not available.

LL012: The downlink subsystem shall transmit the monitoring information from the femtosatellite to the ground station before the launch and using the low gain antenna.

LL013: The uplink subsystem shall receive the configuration information form the ground station to the femtosatellite before the launch and using the low gain antenna.

LL020: The structure shall have a high thermal inertia in order to be used as the thermal control.

LL021: The structure subsystem shall support physical loads up to $500G$.

LL022: The structure subsystem shall work in a temperature range from -150 to 250$^{\circ}C$.

LL023: The structure subsystem shall have a surface with cooling properties in order to e a good heat flow in such a way that the resulting temperature remains inside the range between -40 to 60$^{\circ}C$.

LL030: The attitude determination subsystem shall be calculated from two different sources (optical devices and gyros).

LL040: The position determination subsystem shall be calculated from two different sources (optical devices and accelerometers).

LL041: The position determination subsystem shall guarantee an error less than one degree in latitude and one degree in longitude and the sum of both has to be an area less than $10,000$ square kilometers.

LL050: The attitude control subsystem shall point the high gain antenna towards the ground with an angular accuracy of 5 degrees.

LL051: The attitude control subsystem shall absorb any rotation energy produced by the radiation pressure wind in less than few minutes.

LL060: The tracking subsystem shall transmit the femtosatellite computed position at least once to every ground station in the list of available ground stations.

LL070: The Video recording subsystem shall take pictures with a resolution of at least $1,280$ horizontal by $1,024$ vertical.

LL071: The Video recording subsystem shall take videos with a frame rate of 15 frames per second at least.

LL072: The Video recording subsystem shall have the possibility of compressing pictures and videos using a JPEG compression.

2.5. Additional requirements

Operational Requirements

AD000: The femtosatellite shall be able to receive the launch position in order to align the inertial Measurement Unit before launch.

Safety Requirements

AD001: The femtosatellite battery shall be only used when the femtosatellite is deployed.

Performance Requirements

AD002: The tracking subsystem shall be able to illuminate an area of 200 kilometers in diameter.

Physical and Installation Requirements

AD003: The femtosatellite size shall be lower than 0.2 meters in any direction.

Maintainability Requirements

AD004: The femtosatellite shall be able to keep ready to launch at least for two years without any maintenance action.

Interference Requirements

AD005: The femtosatellite shall be electromagnetically compatible with the mini-launcher.

CHAPTER 3. DESIGN CONSIDERATIONS

The scope of this chapter is to justify how to reduce launch costs based on the list published in [10] by James R. Wertz. This list contains the major trends in the space industry in the US that are:

- Decreasing cost of basic electronics.
- Increased capability of software due to more capable, faster processes.
- Increasing performance capability of electronic packages of a specific size.
- Increasing capability of mission-related software.
- Increasing reliability of basic components.
- Increased use of advanced composite technology to reduce structural masses.
- Increased use of computer technology for decreasing staffing requirements for launch day operations.
- Motivation to use commercial launch providers whenever possible for government projects as well as for commercial projects.
- Increasing political pressure to separate NASA from launch services.
- Establishment of private spaceports to compete with federal ranges.
- Severe budgetary pressure on federal discretionary spending.

The trends outlined above result in decreased size and mass of payloads. Because of this resultant trend, launch cost per unit mass of payload becomes less important than total launch cost for dedicated missions as the last stage becomes the most massive part of the orbited mass. Decreased mass of specific payload packages and the availability of excess payload mass in existing specific launch vehicle configurations drive a tendency to manifest multiple payloads on a given launcher. This makes necessary the existence of some regulations. Current technological, economic and regulatory realities forbid payload delivery to LEO for true costs of less than about $2,000 /kg.

One of the successful strategies employed by *Arianespace* and explained by Mowry in [16] is to make multiple rockets, with different payload interfacing, at one time. This strategy, not only allows to obtain greater probability of launch success, it also maximizes the potential for some cost reductions.

3.1. Documentation program

For the development of this project, a documentation program based on five documents has been created: ConOps document, System requirements document, System design document, Program management plan and Engineering management plan that are summarized following:

WikiSat ConOps Document contains the information related to the utility of the system. This document contains the user manual and is used by clients.

WikiSat System Requirements Document contains the list of requirements that the system shall meet. This document is used by engineers to develop the detailed subsystem document.

WikiSat System Design Document contains the definition of components required for each subsystem. This document is used by engineers to build the system.

WikiSat Program Management Plan contains the planning to build and operate the system. This document is used by engineers to design the mission and by the operator.

WikiSat Engineering Management Plan contains information about the role of each engineer in the organization. This document is used by the organization manager.

3.2. Mechanical considerations

The aerospace science is, by definition, a very demanding sector. One of the challenges of this project is to fit all the aforementioned subsystems in a mass budget of less than 20 grams. Additionally, the femtosatellite should support accelerations up to $500g$ (About $4,900m/s^2$), there is no extra mass for reinforcement. The structure itself must be a monolithic block. The key point is the use of two advanced technologies: PCB and SMD. Printed Circuit Board is a very well known technology that make easy the integration of all the components in a single layer of fiberglass. The Surface Mounted Devices are re-flowed over the copper layers of the PCB and they can resist high loads thanks to the lightness of the components. Extra hardware, like struts or bolts, is not required to increase the toughness. This fact makes the femtosatellite design very light and easy to assemble. The PCB has good dielectric properties and also can be used for thermal control.

3.3. Thermal considerations

There is a heat flow of income heat and outcome heat. The heat flow received by the femtosatellite is determined by the Equation 3.1 and the heat emitted by the femtosatellite is determined by the Equation 3.2; where α is intensity, I is the absorbency, F is the geometric factor, A is the effective area, σ is the Boltzmann constant of $5.67 \cdot 10^{-8} W/(m^2 \cdot K^4)$, T is the effective temperature and ε is the emissivity.

$$\mathbb{Q}_{in} = \alpha \cdot I \cdot F \cdot A \tag{3.1}$$

$$\dot{\mathbb{Q}}_{out} = \sigma \cdot T^4 \cdot \sum(\varepsilon \cdot A) \tag{3.2}$$

Computing the maximum heat flow able to irradiate the femtosatellite (Maximum cooling heat flow) at the maximum operative temperature of 60 $^{\circ}C$ is determined by Equation 3.3 and its value is shown in Equation 3.4.

$$\dot{\mathbb{Q}}_{max} = \sigma \cdot T^4 \cdot \sum(\varepsilon \cdot A) \tag{3.3}$$

$$\dot{\mathbb{Q}}_{max} = 5.67 \cdot 10^{-8} \cdot (273+60)^4 \cdot \sum(0.92 \cdot 0.0042 + 0.15 \cdot 0.0042) = 3.133\,W \tag{3.4}$$

Computing of the total heat flow when the femtosatellite is radiated by the sun the following results are obtained:

$$\dot{\mathbb{Q}}_{total} = \dot{Q}_{sun} + \dot{Q}_{albedo} + \dot{Q}_{IR} + \dot{Q}_{disipated} = 0.895 + 0.878 + 0.707 + 0.500 = 2.980\,W \tag{3.5}$$

$$\dot{\mathbb{Q}}_{sun} = \alpha \cdot I \cdot F \cdot A = 0.15 \cdot 1{,}420 \cdot 1.0 \cdot 0.0042 = 0.895\,W \tag{3.6}$$

$$\dot{\mathbb{Q}}_{albedo} = \alpha \cdot I \cdot F \cdot A = 0.92 \cdot 454.4 \cdot 0.5 \cdot 0.0042 = 0.878\,W \tag{3.7}$$

$$\dot{\mathbb{Q}}_{IR} = \alpha \cdot I \cdot F \cdot A = 0.92 \cdot 244.0 \cdot 0.75 \cdot 0.0042 = 0.707\,W \tag{3.8}$$

$$\dot{\mathbb{Q}}_{dissipated} = 0.5\,W \tag{3.9}$$

Hence, when the femtosatellite is exposed to the sun, the albedo and the infra red, the resultant temperature for the worst case is computed as show in Equation 3.10 which is below the maximum operating temperature.

$$\mathbb{T} = \sqrt[4]{\frac{\dot{Q}_{total}}{\sigma \cdot \sum(\varepsilon \cdot A)}} = \sqrt[4]{\frac{2.980}{5.67 \cdot 10^{-8} \cdot (0.92 \cdot 0.0042 + 0.15 \cdot 0.0042)}} = 329\,K = 56\,^{\circ}C \tag{3.10}$$

In these conditions, when the satellite is in idle, the resultant temperature is $324\,K = 51\,^{\circ}C$.

When the femtosatellite is in the eclipse, it is only exposed to the infra red, the resultant temperature for the worst case is determined by Equation 3.11 being this value inside the operating temperature range. In these same conditions, when the satellite is in idle, the resultant temperature is $253\,K = -20\,°C$.

$$\mathbb{T} = \sqrt[4]{\frac{\dot{Q}_{total}}{\sigma \cdot \sum(\varepsilon \cdot A)}} = \sqrt[4]{\frac{1.044}{5.67 \cdot 10^{-8} \cdot (0.92 \cdot 0.0042 + 0.15 \cdot 0.0042)}} = 253\,K = -20\,°C \tag{3.11}$$

3.3.1. Thermal design discussion

This is a passive thermal control subsystem because the femtosatellite has a good balance between the income heat flow and the outcome heat flow. Any modification in the initial design may vary the working temperature range. In this case the copper area in the PCB back layer should be changed until the femtosatellite temperature range is correct. The copper has a good emissivity property while the fiberglass has a very bad emissivity property. If the PCB back layer has to much copper, the femtosatellite can irradiate the heat and the femtosatellite tends to cool down. Opposite, if the predominant area in the PCB back plate layer is fiberglass, the femtosatellite tends to keep the heat and it warms up.

3.3.2. Thermal cases summary

Following, the Table 3.1 presents the basic mission cases in terms of electrical power and thermal levels. These levels are Idle case and transmitting case correspondig to a standby state and an active state. Both levels have two cases more that are Sun case and Eclipse case. There are four combinations in total under study.

Table 3.1: Thermal budget overview

Maximum cooling heat flow						
	$\dot{Q}_{sun}$	$\dot{Q}_{albedo}$	$\dot{Q}_{IR}$	$\dot{Q}_{disipated}$	$\dot{Q}_{TOTAL}$	Temp. $K\,(°C)$
Cooling	-	-	-	-	3.133	333 (60 $°C$)
Sun case						
	$\dot{Q}_{sun}$	$\dot{Q}_{albedo}$	$\dot{Q}_{IR}$	$\dot{Q}_{disipated}$	$\dot{Q}_{TOTAL}$	Temp. $K\,(°C)$
Idle case	0.895	0.878	0.707	0.330	2.810	324 (51 $°C$)
TX case	0.895	0.878	0.707	0.500	2.980	329 (56 $°C$)
Eclipse case						
	$\dot{Q}_{sun}$	$\dot{Q}_{albedo}$	$\dot{Q}_{IR}$	$\dot{Q}_{disipated}$	$\dot{Q}_{TOTAL}$	Temp. $K\,(°C)$
Idle case	0.000	0.000	0.707	0.330	1.073	253 (-20 $°C$)
TX case	0.000	0.000	0.707	0.500	1.207	262 (-11 $°C$)

3.4. Preferred component list

There is a preferred configuration for a default list of components in this femtosatellite. Table 3.2 presents the mass, the power and the temperature budget for every subsystem composed by these components. Some components are the implementation of one or more subsystems. A detailled Bill of Materials (BOM) is also provided in the Appendix C.

Table 3.2: Femtosatellite subsystems mass, power and temperature budget

Subsystem	**Mass**	**Max. Power**	**Idle**	**Temp. Range**	**Size**
	grams	*mW*	*mW*	$^{\circ}C$	*mm*
Power supply subsystem	6.6	$3V, 610mAh$		-30 to 60	$D24.5$ x 5
Communication subsystem	0.1	100	0.1	-40 to 85	171 x 34
Structure subsystem	2.0	–	–	-116 to 204	30 x 25 x 7
Attitude determination subsystem	1.4	5	0.1	-30 to 70	$D5$ x 2
Position determination subsystem	1.2	65	0.1	-30 to 60	30 x 16 x 3
Attitude control subsystem	0.4	40	30	-150 to 550	1 x 0.25 x 0.25
Tracking subsystem	7.0	500	250	-40 to 85	140 x 25 x 1
Video recording subsystem	1.0	150	150	-20 to 60	11 x 11 x 6
TOTAL	19.7	860	330	-20 **to** 60	140 **x** 30 **x** 7

3.5. Electrical considerations

The whole femtosatellite design is based on a single LiPoly battery. For this reason, it is mandatory to do a mission consumption budget that will be used in the power budget in order to dimension the battery capacity needed. This study is based on a typical Earth observation LEO mission but real power budget will depend on each mission type. Some simulations have been run using the *Moon2.0* simulator and the *LEO* preset. Also, the *Wikisat* ground station network consisting of Denmark ground station, *Barcelona* ground station, *El Arenosillo* ground station and *Maspalomas* ground station have been selected as shown in Figure 3.1. The whole commercial available network is not been used in order to be realistic.Can be shown that there is an average window of $2,000s$ per day where the femtosatellite has a ground station in Line of Sight (LOS). If the mission has a maximum duration of nine days and transmission time is about $256s$, as computed by Gonzalez in [5] there is a total of about $2,300s$ of pure transmitting mode.

In the future, if a mission is demanding higher power, it makes sense to put small solar panels in the back of the satellite where the Sun allows the battery to be charged. The main problem with this is that the area of the femtosatellite is small. Other important issue is that components should have very low consumption, hibernate modes and same voltage to make the power distribution subsystem more efficient.

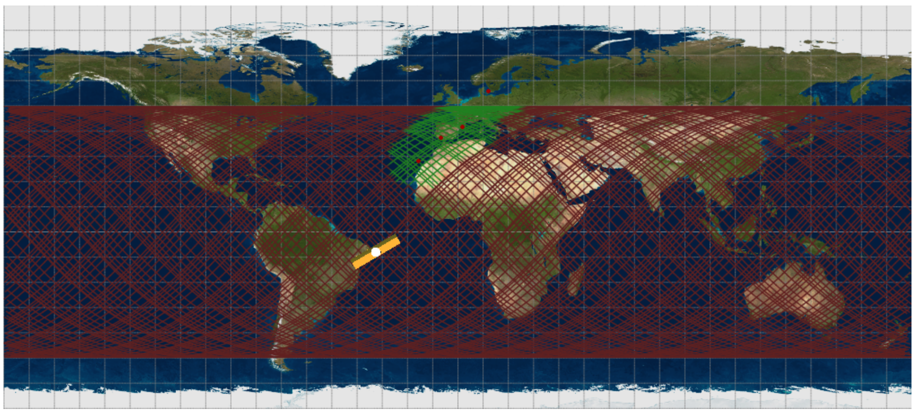


Figure 3.1: Wikisat ground station network and femto-satellite trajectory. (Green available)

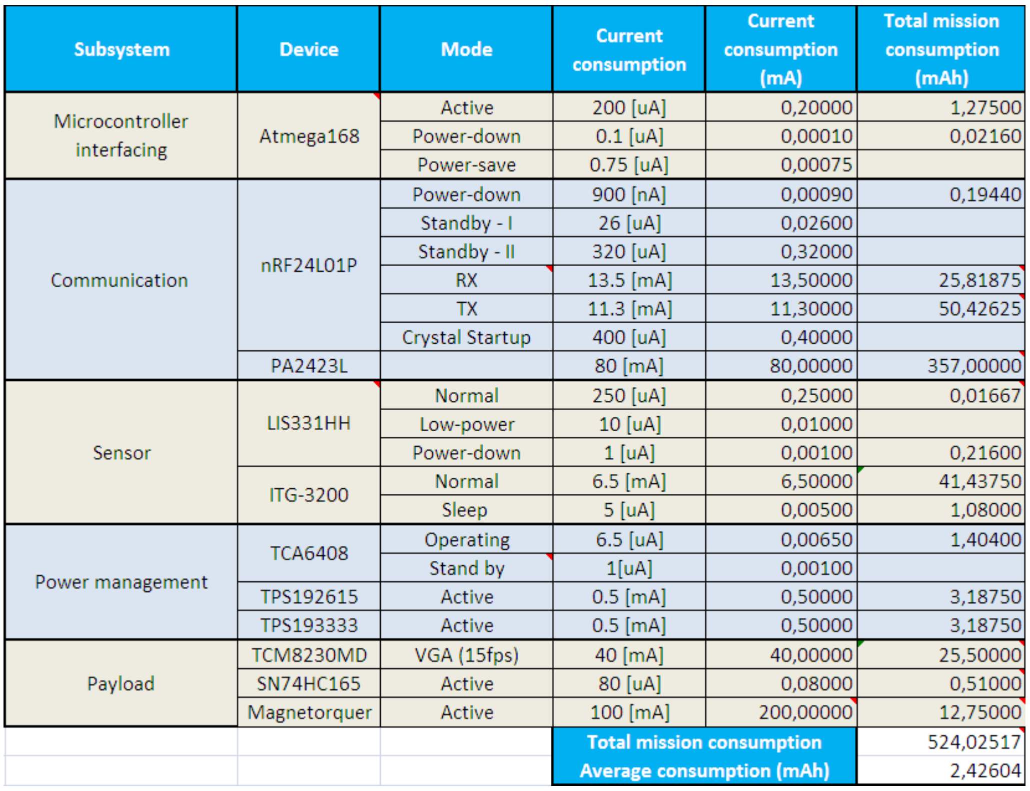
3.5.1. Power budget

There are two working modes: Standby and Active. When Standby mode, also called Idle case, no main work is done by the satellite except for the integration of the attitude an time counter. When Active mode, also called transmitting case, all the subsystems are actives. Magnetorquers could generate a magnetic field in order to have an attitude control to point the high gain antenna towards the ground station or to follow an interesting point for the payload. The transceiver and the power amplifier could transmitt or receive information. Since this femtosatellite uses a batery, all the available power should be used only when necessary. The total mission time when the satellite could stay in active mode during these nine days are about 382 minutes of a total of about $13,000$ minutes. Only 3 percent of the mission time, the femtosatellite will have an important battery leak. The total mission consumption is about $520mAh$ and the total available is $610mAh$. We have to say here that this is only a simulation based on a typical mission. Accurate budget should be done in a future real mission. Finally, the average consumption per hour is $2.4mAh$

Table 3.2 is a summary about the femtosatellite main component consumptions. Some asumptions were done related to the Active mode. Transmittion will take only 70 percent and Reception is the other 30 percent. Pictures are taken only during 10 percent of the Active time meanwhile the attitude control will take only 1 percent of this time. The highest current consumption is due to the magnetorquer and the power amplifier but the power amplifier is used lager time so this component represents the major consumption. For this reason saving modes are recommended like the Burst technology or Data Compression.

3.5.2. Power sources

The power source for this kind of missions, due to the short mission time is strongly recomended to use batteries. 34 percent of the mass budget is for the battery that represents 6.6 grams compared to the tracking subsystem that is the 36 percent of the mass budget in 7.0 grams. In the early designs, i.e. *WikiSat v3* a coin batery was selected having a better consumption to mass ration that the current LiPoly batteries. The main problem that a coin battery has is not the fact that it is no rechargeable but it has a very low maximum power, about $50mAh$, while the total consumption is $610mAh$. LiPoly batteries have higher



Subsystem	Device	Mode	Current consumption	Current consumption (mA)	Total mission consumption (mAh)
Microcontroller interfacing	Atmega168	Active	200 [uA]	0,20000	1,27500
		Power-down	0.1 [uA]	0,00010	0,02160
		Power-save	0.75 [uA]	0,00075	
Communication	nRF24L01P	Power-down	900 [nA]	0,00090	0,19440
		Standby - I	26 [uA]	0,02600	
		Standby - II	320 [uA]	0,32000	
		RX	13.5 [mA]	13,50000	25,81875
		TX	11.3 [mA]	11,30000	50,42625
		Crystal Startup	400 [uA]	0,40000	
	PA2423L		80 [mA]	80,00000	357,00000
Sensor	LIS331HH	Normal	250 [uA]	0,25000	0,01667
		Low-power	10 [uA]	0,01000	
		Power-down	1 [uA]	0,00100	0,21600
	ITG-3200	Normal	6.5 [mA]	6,50000	41,43750
		Sleep	5 [uA]	0,00500	1,08000
Power management	TCA6408	Operating	6.5 [uA]	0,00650	1,40400
		Stand by	1[uA]	0,00100	
	TPS192615	Active	0.5 [mA]	0,50000	3,18750
	TPS193333	Active	0.5 [mA]	0,50000	3,18750
Payload	TCM8230MD	VGA (15fps)	40 [mA]	40,00000	25,50000
	SN74HC165	Active	80 [uA]	0,08000	0,51000
	Magnetorquer	Active	100 [mA]	200,00000	12,75000
			Total mission consumption		524,02517
			Average consumption (mAh)		2,42604

Figure 3.2: Femtosatellite total and average mission consumption

maximum power in the order of $12,000 mAh$ for a short time.

If lager time is required for a very active mission or for a high active mission, small solar cells could be mounted in the back of the antenna in such a way that it could recharge the battery. These solar cells only represents few grams in the mass budget but it increases the complexity of the system.

Other power source that could be considering in femtosatellite formations is to reuse the magnetorquers to pass the energy by induction if femtosatellites are less than one meter closer and if the formation have a harvester satellite.

3.5.3. Distributed vs centralized voltage regulation

There are some considerations to take into account related to the voltage regulation. Each box was designed with a different power need and different voltages. In old satellite designs there were two approaches in order to feed the subsystems: distributed power supply and centralized power supply. A third approach only possible in a complete design like the one proposed by the *WikiSat* team.

Distributed power supply means that there is a single power source with a given voltage. Inside every box, there are few regulators that convert the voltage to every need that this box may has. This approach is really complicated and not so efficient. The main advantage is that it is easy to design a subsystem inside a box. It is easy to have a modular design allowing a complex design. Another advantage is that the heat produced by the regulator is also distributed and is more efficient in terms of thermal distribution.

Centralized power supply has a single regulator for all the devices and for every voltage. The energy transformation is very efficient respect to the distributed approach. The design is really simple but no so modular. A good dimension in terms of power consumption should be done because it is not easy to isolate a box if this box has a problem.

Single power supply is a version of the centralized approach but, in this case, all the devices have the same voltage and there is no need to convert to other voltages. This is the most efficient design in terms of power transformation efficiency. The problem is that there is no way to do a modular design. All the system should be designed completely from the scratch. This is the approach used in the *WikiSat v4.1* except for the Payload that a centralized schema has been used. The rest of components will use $3.3V$ everywhere. This voltage has been chosen instead of $5.0V$ because the LiPoly battery has $3.7V$ and all the components are tolerant to it. There is no need to regulate the power except for some cases like the transmitter that requires a very stable power source and, of course, the payload that has a different approach.

3.5.4. Cosmic radiation study and mitigation

The most important phase when designing a satellite is to study the cosmic radiation study and its effects on the electronic parts. Radiation particles coming from solar wind, like protons and electrons, can be a serious problem for the semiconductor materials on board. In [15], a wide study of this effects on the *WikiSat* in presented. The key points extracted by Molas-Pous during her research are exposed bellow.

- Short periods of radiation are allowed for a small satellites like this if they are in very Low Earth Orbit (LEO)
- The main radiation source apart of the Van Allen electrons and protons, is the so called South Atlantic Anomaly (SAA) that is passing through it every day
- The battery has a good shielding effect over the weak electronic components against the Single Events Effects (SEE) that can polarize a transistor or even destroy it.

3.6. Attitude determination and control subsystem

The attitude determination could be done by gyros but they have a lot of BIAS of drift error and an absolute correction method is required. Gyros can provide a very accurate attitude determination value. In order to correct this BIAS, a magnetometer and a Sun-tracker will be used. Accuracy is low in the order of 10 degrees while gyros can give an accuracy in the order of 0.1 degrees.

The attitude control will be done by magnetorquers that are not really accurate but it is very simple and easy to be assembled in the femtosatellite structure. These magnetorquers are controlled by the IO Expander with a current limit of $100mA$ per coil.

3.6.1. Earth magnetic field sensor

The attitude control is done by four magnetorquers. The control can be done in 2 degrees of freedom. These magnetorquers can generate a magnetic pole that reacts against the Earth magnetic field. The main problem with this technique is that we should know what is the magnetic field surrounding the femtosatellite and the variations of the Earth's magnetic field. WikiSat has a Earth's magnetic field model in the EEPROM memory that is showed in figure 3.3 with information about the declination of the true magnetic North. There is an example of implementation in the Appendix F. The magnetic model is based on the NOAA World Magnetic Model (National Oceanic and Atmospheric Administration). Comparing the sensor reading with the value stored in the model, for a given coordinates, it is possible to know one angle of the femtosatellite attitude. This method is useless in the poles due to the gimbal lock effect.

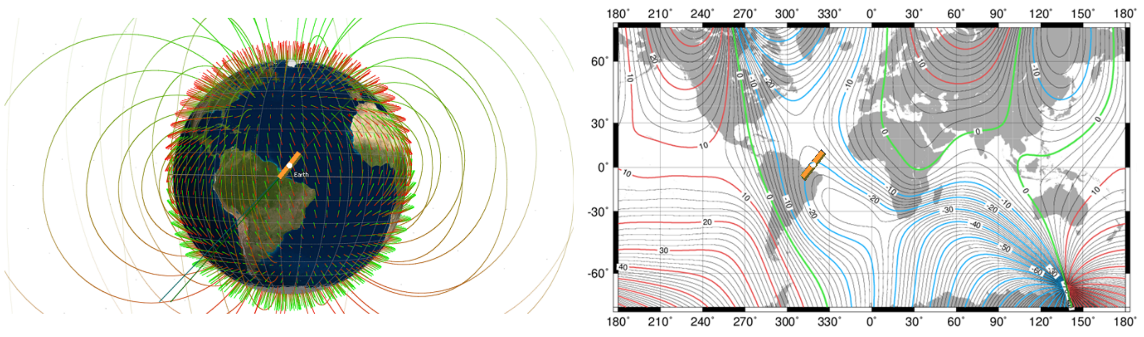


Figure 3.3: Femtosatellite Earth's magnetic field model and the NOAA real declination map

3.6.2. Sun-tracker sensor

We need to solve the other angle in order to know where the femtosatellite should point with the High Gain Antenna or the Payload Sensor. For this reason, a Sun-tracker should be installed in the followinw WikiSat version. This other method is useless in the eclipse that hapens every orbit. Additionally, it is possible to use the Moon if we know the position not only the Sun but the Moon. Typically, femtsatellite missions are more focused on day recording but download could be in any moment. Also, the main needs dor Disaster Management are closer to the Equator.

3.7. Payload areas

Some rules about how to use the available payload areas are defined. In that way, there is no need to worry about basic functionalities such as: COTS components conditioning, Logistics usage of COTS components, Tracking functionalities and Information download to Earth. Additionally, using these rules it make easier to test if the payload is going to affect to the femtosatellite integrity.

The area of the femtosatellite *WikiSat* was optimized during the design. It was fitted to the antenna array in order to reduce the size as much as possible. There are four different zones clearly separated by the microstrips of the antenna. The most suitable part to allocate the circuitry, as show in Figure 3.4 is the central part because in the bottom part is the battery protecting against radiation. It was decided to divide this area in two parts, one for the femtosatellite and the other for the payload. The lateral areas is where the magnetorquers are allocated leaving free space to use as payload if necessary. These lateral areas have two-fold problem: they are exposed to radiation and they need to be conditioned passing lines through the microstrips. When lines are passed through the microstips it has to be done in the bottom part of the satellite in order to no interfere the antenna. It is very important to check that there is no near field interference in those passing points.

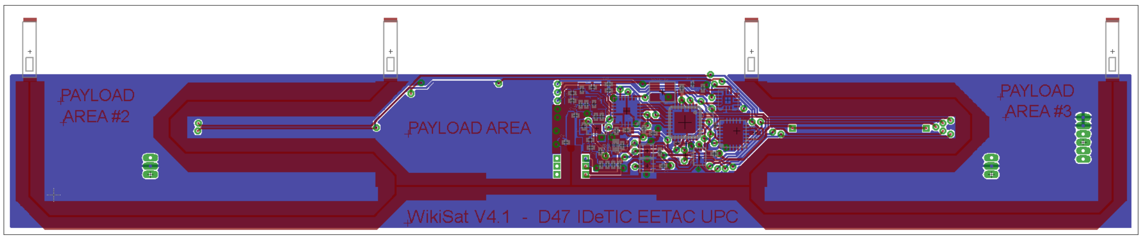


Figure 3.4: WikiSat V4.1 payload areas

3.7.1. PCB constraints

There are some constraints that should be taken into account when designing a PCB:

- Lines can not be thinner tan $0.2mm$ and VIAS shall be at least of $0.6mm$.
- SMD components can not have a footprint lower than 0402 size (0.04×0.02 in, 1.0×0.51 mm).
- In case of needing more layers or SMD components with footprints lower than 0402 size the implication on the manufacturing process will be studied.
- Generally the ”8mil” $(8\mu m)$ rules established on the *micron-20.dru* files of design rules for EAGLE PCB will be followed.
- It is not allowed to pass through the microstrips in the bottom part cutting the copper ground plane. If it is necessary to pass solder wires will be used.
- It is not allowed to pass through the microstrips anyway only using a board with the same characteristics and copper on the other side plus an EMC (Electromagnetic compatibility) study. In that case the same rules about passing lines in the bottom part are applied. Also, it will be necessary a study about electromagnetic interference with the synthetic aperture antenna and the magnetorquers and how can the metallic parts reflexions be and the harmonic generation of the transmitter payload components. It is not allowed to pass closer than $0.4mm$ from the microstrip.
- It is desirable to have a ground plane closer to the microstrip, on the bottom and top sides.

- It is allowed to mount PCB board above the other taking into account that this board can not exceed the payload area minus a $0.4mm$ border. It will be necessary to fix them with resin and use pins to connect the buses, never connectors. The pile of PCB boards never can exceed its area unless the previous rule is followed with its implications and limitations.

3.7.2. Main payload area use

The main payload area is conditioned to easily mount and supply SMD components. Through an I^2C port the state of the tested component can be monitored by the satellite. This bus follows a set of rules that, in case of overload, can endanger the satellite integrity and even the launching. Moreover the power consumption of the payload must fit the power budget as the available power and the consumption capacity are very limited in case of use a battery even using solar cells. An excess in power consumption has an impact on the satellite thermal stability during its entire life cycle. The payload must be turn on only in key moments. Some basic functions are supplied by the satellite as event detection, navigation data and system operation through subsystem supply. Examples of this are: test of new sensors, test of high definition cameras (as the data sending is highly limited), test of the transmitters, etc. Criticality on this area is low, it may can endanger the attitude control capacity of the satellite in orbit if transmitter systems interfere with magnetorquers.

3.7.3. Secondary payload area use

The second payload area has access to several power and data lines. Its use is justified when the receptor is far away from the main payload area. Any specific need of current regulation can be supplied through the available channel, always taking into account the crossing microstrips rules. Examples of use are: Transmitter-Receiver lines (optical or electromagnetic waves). Criticality on this area is moderate, it may endanger the attitude control capacity of the satellite in orbit if transmitter systems interfere with magnetorquers.

3.7.4. Third payload area use

This area is the best place to test power systems as solar cells as it has an SPI bus and direct connection to the battery. Any specific current regulation shall be implemented by the payload user because there is no way to supply the available voltages in the main payload area unless using multi-layer techniques. In that case, implications and cost must be determined. Criticality on this area is very high, it may endanger the attitude control capacity of the satellite in orbit if transmitter systems interfere with magnetorquers.

CHAPTER 4. FEMTOSATELLITE LINK BUDGET

The antenna design has been lead by Fernandez-Murcia and it is widely explained in [4]. In this chapter only the main aspects are exposed.

4.1. Starting point

First of all, an accurate power link budget for the communication is necessary in order to determine which elements will be required to increase the transceivers radio link (initially designed for less than 100 meters). The maximum link distance is fixed at $500\,km$, with a minimum of $250\,km$ and it will be unidirectional from the satellite to the base station. It will be assembled on a Satellite-on-a-Board. It is selected the transceiver nRF24L01P [26] at $2.4\,GHz$ System-on-Chip (SoC). It will work at $250\,kbps$ with a $0\,dBm$ power signal (GFSK modulation).

The system must be upgradeable, ready to be updated with new elements constantly (step-design). It starts as a medium range communication system able to evolve to a long range system. For this reason, some extra components like amplifiers (in both sides of the radio link) are necessary to increase the range. It is mandatory to begin with the computation of how many extra gain (dB) will be necessary for the worst case condition (500 km). For this calculation, the following expression has been used:

$$\mathbb{P}_{rx} = P_{tx} + G_{tx} - e_{tx} - LFS - PLF + G_{rx} - e_{rx} \tag{4.1}$$

A polarization loss factor (PLF) of $3\,dB$ has been considered due to the different polarization of the antennas, one linear and the other circular. This is critical because polarization may change due to satellite movement or any other undesired effect. A circular polarization ($20\,dB$ of gain) Yagi antenna is selected for reception and a $6\,dB$ antenna (the gain expected for the array) for transmission is considered. Finally, $0.5\,dB$ losses due efficiencies have been considered at each side of the link. The operating frequency (f_0) selected is $2.49\,GHz$, but it may change a bit if necessary (selecting a different operating channel).

Equation 4.1 can be easily transformed to estimate the extra gain necessary as shown in Equation 4.2.

$$\mathbb{G}_{extra} = P_{tx} + G_{tx} - e_{tx} - LFS - PLF + G_{rx} - e_{rx} - P_{rx,min} = 50.3\,dB \tag{4.2}$$

As noticed, at least $50.3\,dB$ gain is necessary, without considering noise effects, for a 500 km link. Typical amplifiers offer between 10 and $20\,dB$ of gain, for this reason at least two or three amplifiers will be required. In WikiSat V4.1 design there are one power amplifier and two low noise amplifiers.

4.2. Design

For transmission, a power amplifier (PA) with an adequate gain-consumption trade-off, easy to implement (not too much components) and the better OIP3 [1] and P1dB [2] as possible are the main specifications considered. After a research it has been found a product from SiGe [3] , it is the PA2423L.

For reception, other parameters are more critical (principally its noise figure and gain), therefore is necessary to find out some low noise amplifiers (LNA in advance). Two options have been found and both will we cascade connected with a filter between them to reduce noise. The LNA selected (by its low noise figure and as close to the antenna as possible) is from Analog Devices [4] and its the ADL5521 (it is important to place just before the reception antenna, an amplifier with the lowest noise figure as possible).

Figure 4.1 shows the femtosatellite communication diagram as well as the link budget calculations.

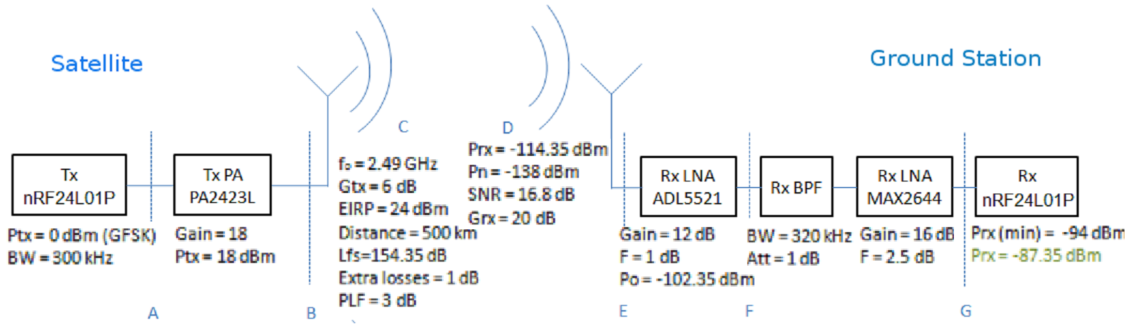


Figure 4.1: Femtosatellite communication diagram block and link budget

4.3. Implementation

Each component, LNA and PA, was tested separately. To do so, a PCB has been manufactured for each one following proposed circuits from manufacturers data sheet with some simplifications.

An initial design of the array configuration for the transmitting antenna is represented in Figure 4.2

The selected antennas are ceramic antennas AT9520 with a $1.5-2dB$ max. gain and a radiation diagram similar to a dipole diagram in 2.4 to $2.5GHz$ band. As 5 to $10dB$ of gain are required a four elements array has been considered. Different theoretical and experimental test have been made, results are detailed in [4].

[1]Output interception point order 3
[2]Input power 1 dB compression point
[3]http://www.sige.com
[4]http://www.analog.com

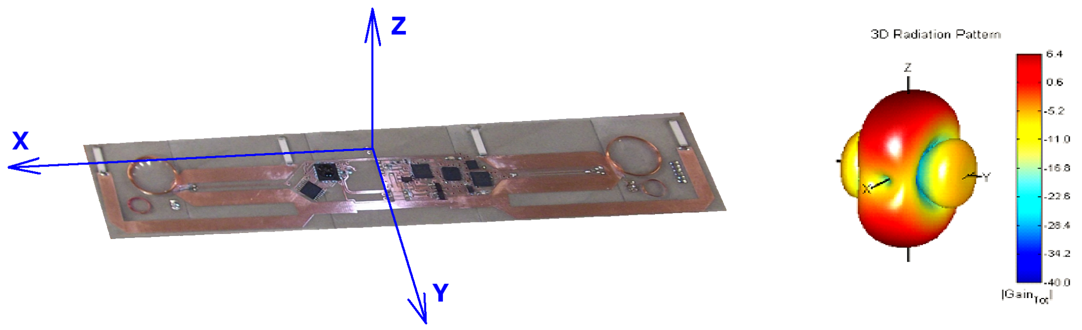


Figure 4.2: a) Array antenna distribution and b) 3D radiation pattern

CHAPTER 5. SYSTEM IMPLEMENTATION

The technical implementation of the *WikiSat v4.1* was split into the following parts: component selection, hardware design, prototyping, integration of the different subsystems and test and validation of the whole system. The aforementioned implementation steps were performed and documented below. As mentioned before, one of the key aims of the technical implementation part was to perform it with minimum expenses and using low-cost in-house manufacturing tools that could be afforded by a particular hobbyist, open source and open hardware programming and debugging tools.

5.1. Component selection

Following the design specification stated above the component selection was performed. The component instances were selected according to the requirements exposed in Chapter 2 taking as well into account their cost characteristics. The summary of the component selection work is summarized in the Table 5.1.

Table 5.1: Femtosatellite component list

Component candidate	Mfg	Model	Size mm	Package Type	Price €
MCU	Atmel	ATMEGA168	$5X5$	MLF32	3.07
Accelerometer	ST	LIS331HH	$3X3$	QFN14	3.64
Rate gyroscope	Invensense	ITG-3200	$4X4$	QFN24	25.39
Transceiver	NORDIC	nRF24L01P	$4X4$	QFN20	3.78
HD camera	TOSHIBA	TCM8230MD	$6X6$	-	9.95
Serializer	Texas Instruments	SN74HC165PW	$5X4.5$	TSSOP	0.27
IO expander	Texas Instruments	TCA6408A	$1.80x2.60$	RSV	1.86
Voltage regulators [1]	Texas Instruments	TPS719XXXX	$2x2$	DRV	1.36
Power amplifier	SiGe Semiconductor	PA2423L	$3.25x1.85$	QFN6	2.63
				Total components cost	53.61

5.2. Hardware design

One of the most important tools in the hardware design process is the CAD software used for schematics and board design, electrical and manufacturing-related rules compliance revision and CAM output synthesis for prototyping or outsourced manufacturing. The study of available open source and freeware tools was performed and the CadSoft Eagle CAD [2] was selected due to its advantages such as freeware license; wide community support and amount of component libraries; user-scripting language support making the CAD highly customizable and others.

[2]Official website: http://www.cadsoft.de/

This software package is able to produce CAM output in GERBER RS-2742 providing though a high compatibility with industrial PCB manufacturing and assembling services as well as in-house prototyping using even a toner-transfer method. It is also important to note that the package is equipped with an auto-routing tool which allows achieving the lowest development time.

The hardware design of the system, as well as the conditioning of all the components, was performed according to the manufacturers recommendations of the implementation methods in the data sheets [19, 22, 18, 26, 20, 24, 28]

5.2.1. Payload subsystem

The payload subsystem is used to accomplish the needs for the specific mission. In this case, as the *Wikisat V4.1* will act as a Sun-tracker, it has an imaging payload. This payload subsystem is composed of a HD camera [20] and a shift register [28] used to convert the parallel output of the camera into a serial list of bits in order to send them to the Ground Station. Payload conditioning circuit is shown in Figure 5.1. The complete system conditioning is shown in Appendix A.

As can be seen, three different voltages are required to feed the camera ($3.3V$, $2.6V$ for the sensor and $1.5V$ for the A/D). Also I^2C signals (SCL, SDA) are needed to start up the camera. Finally, some clock references are used to capture (CLKO) and trasmit (CLK1) the images. The camera output (D0 from D7) is transmitted directly to the shift register. The shift register in supplied by $3.3V$ and needs two clock references (CLK1 and CLK) in order to synchronize with the camera and to serialize the received data.

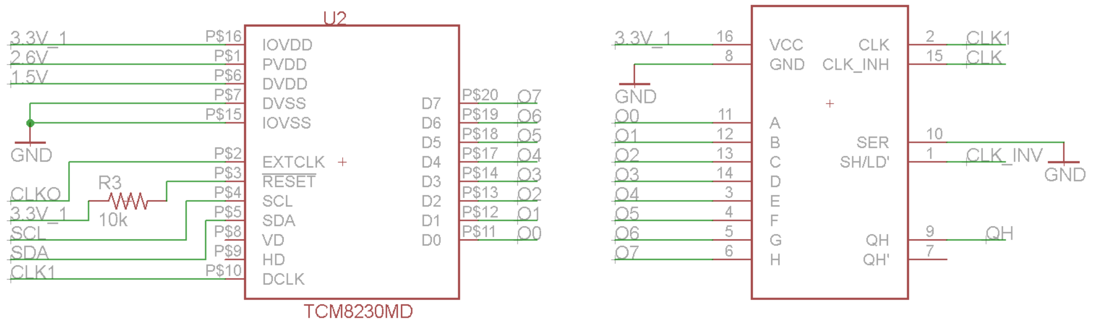


Figure 5.1: Payload subsystem schematic

5.2.2. Microcontroller interfacing

The Main Control Unit is a 8051 compatible processor, $16MHz$ with $32kb$ of RAM and $2kb$ of SRAM. Multiple peripherals are available like: Pulse With Modulators (PWM), General Purpose Input Outputs and Analogic to Digital Converters (ADC). Figure 5.2 show the microcontroller interfacing implemented on the design. As the rest of the system it is supplied by $3.3V$.

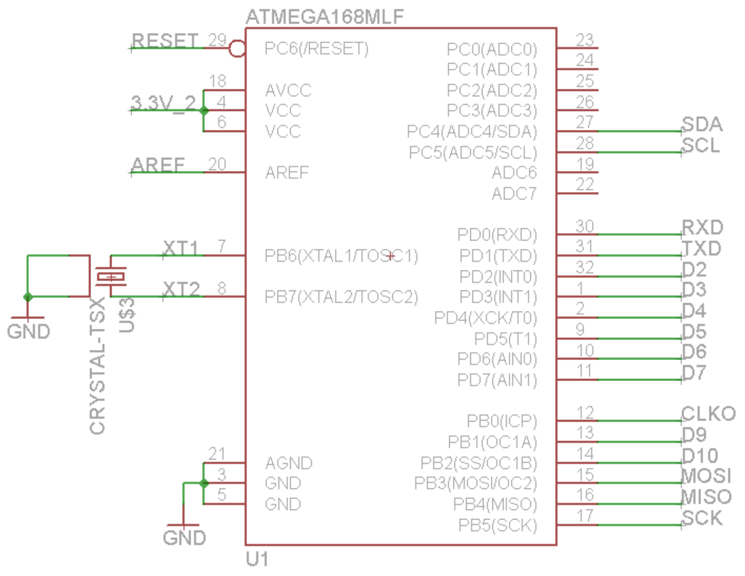


Figure 5.2: Microcontroller interfacing schematic

5.2.3. Communication subsystem

This subsystem is essential for the femtosatellite mission as it is the responsible for the communication with the Ground Station. The main parts of this subsystem are the transceiver [26] and the PA [25]. The conditioning of this part has been design by Fernandez-Murcia in [4] and can be show in Appendix A. Figure 5.3 shows the pin out of both components.

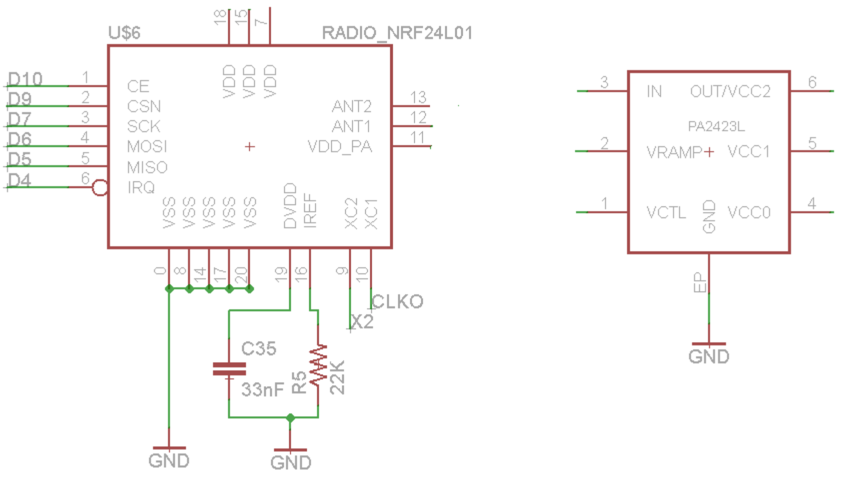


Figure 5.3: Communication subsystem schematic

5.2.4. Sensor subsystem

The key subsystem is the Inertial Measurement Unit (IMU). It has a platform of high accuracy 3 Axes accelerometers 16 bit of data resolution, three gyros and 2000 $^{\circ}/s$ range and temperature sensor for real-time calibration. All the femtosatellites are connected to the I^2C bus for data and configuration. As can be seen in Figure 5.4. Specific details of this subsystem conditioning are explained in [1].

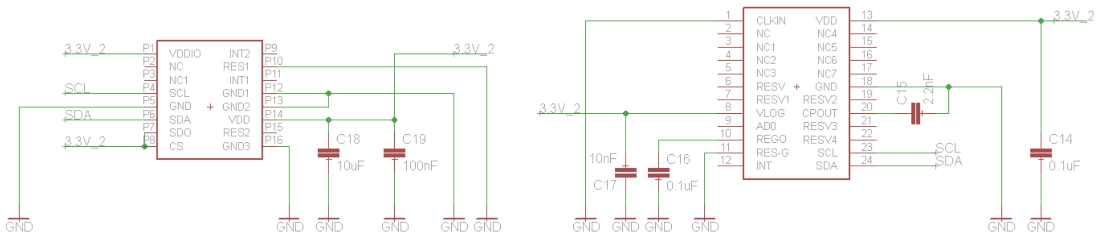


Figure 5.4: Sensor subsystem subsystem schematic

5.2.5. Power management subsystem

The power management subsystem is composed by two voltage regulators [27] and a IOExpander [24]. The IOExpander is in charge of femtosatellite power management to guarantee the power saving. It turns-on or turns-off the different subsystems depending on the needs off a specific moment. It is controlled through the I^2C port by the MCU. The voltage regulators are in charge of the voltage level adaptation to supply the different subsystems; as the main power source is a $3.3V$ LiPoly battery and there are some devices, like the HD camera, that require different voltages to operate. The Figure 5.5 shows the conditioning for these three devices.

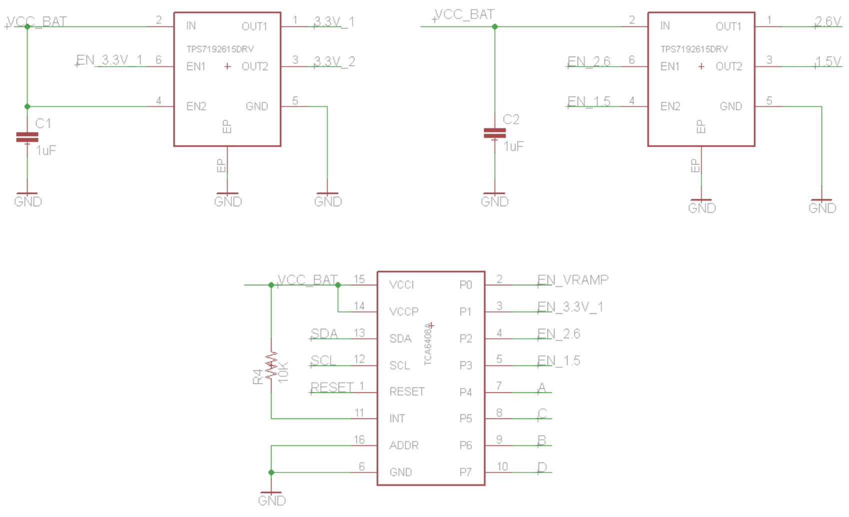


Figure 5.5: Power management subsystem schematic

5.3. Board

One important challenge of the femtosatellite development was to do the board design. This challenge was due, first, to the reduced size of the SMD components (0402) and, secondly, to the constrain of design a satellite as small as possible.

This second constrain was specially limiting when connecting buses like I^2C or SPI that cross the board from right to left side. The same problem was found when connecting the magnetometers. To deal with this problem, kind of ”connector” were design. The idea is to pass over the antenna microstrip. Also, it was very complicated to connect the power supply from the regulator to the payload area. In order to do the power supply connection, it was necessary to design a ”bridge” in such a way that the lines will be connected over the antenna microstrip. These challenges are illustrated in Figure 5.6.

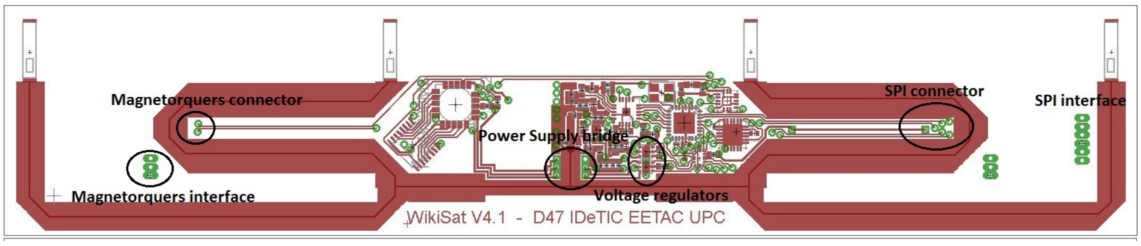


Figure 5.6: *WikiSat V4.1* board challenges

5.4. Integration

5.4.1. PCB prototyping procedure

In order to fulfill the N-Prize requirements of being low-cost and easy to manufacture, it was decided to make a prototype using the technologies available in the EETAC[3]. In the school there are two CNC (Computed-aided Numerical Control) machines available both from LPKF[4].

The first one is shown in Figure 5.7 a) an it is property of EETAC. The ProtoMat H100 is LPKFs top-of-the-line circuit board plotter, ideal for all in-house prototyping applications, including multilayer and RF applications. Due to the small size of the femtosatellite, this machine was not accurate enough for the *WikiSat V4.1* prototyping so it was necessary to use an accurate and precise one, the LPKF Protolaser S.

Figure 5.7 b) shows the LPKF Protolaser S, property of TSC[5] that kindly let us use for the *WikiSat V4.1* prototyping. The LASER system opens up a new dimension in in-house prototyping: it transfers the layout onto the PCB with unprecedented speed and precision easily and with no chemicals.

[3]Escola d’Enginyeria de Telecomunicació i Aeroespacial de Castelldefels, http://eetac.upc.edu/en/
[4]LPKF, http://www.lpkf.com/
[5]Departament de Teoria del Senyal i Comunicacions, http://www.tsc.upc.edu/en.html

Figure 5.7: a) LPKF Protomat H100 and b) LPKF Protolaser S

5.4.2. PCB assembling procedure

The PCB assembling process was done in-house using a set of very basic tools. The process was split into 3 main parts: soldering paste application; components positioning; reflow soldering. An important part of the process was preparing the components for the assembly in the previously defined assembling order. Figure 5.8 shows the *WikiSat V4.1* assembled.

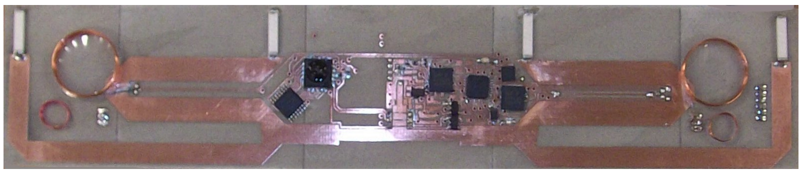


Figure 5.8: WikiSat v4.1 assembled

A low-cost reflow oven shown in Figure 5.9 and based on the consumer toaster device was made in order to ensure a quality reflow soldering. The oven was equipped with a thermoresistive temperature sensor attached to the 8-bit ADC of the oven controller. The reflow oven controller was based on the same ATMEGA168 hardware and *Arduino* firmware. The embedded software allows programming the heat treatment procedure according to the typical requirements of the reflow soldering process (preheating temperature and time, soldering temperature and time and final cooling time). The feedback of the reflow process was provided to the PC terminal in the real time during the soldering process via USB to UART bridge interface. More information on the low-cost assembling tools used in this work can be found in [1].

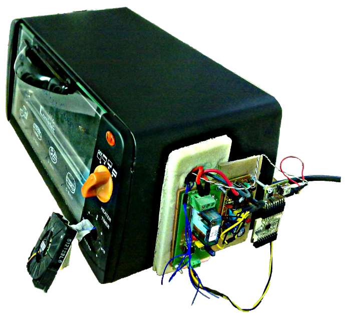


Figure 5.9: Reflow oven

5.5. Test and validation

This chapter will demonstrate the test and evaluation procedures developed in order to assess operability of the device and the results analysis will be stated. Each subsystem has been tested separately from the others.

5.5.1. Microcontroller interfacing subsystem validation

These tests were lead by Kravchenko and they are widely explained in [12]. On this work, Kravchenko presents the flowing experiments:

IMU data acquisition performance: The purpose of this experiment was to perform a study of the DAQ system of sensors and MCU determining the maximum data acquisition and streaming rates through the UART interface varying the methods of reading the FIFO registers of the sensors and I^2C bus parameters. It was also necessary to determine an MCU time cost of a single DAQ operation at the maximum rate. The payload for this experiment was defined as the 3 values of acceleration and 3 values of angular velocity measurements resulting in a message size of 12 bytes.

IMU data processing demo: The purpose of this experiment was to demonstrate the way of interfacing the IMU data with the common scientific tools stating an example of the IMU noise distribution model study in the non-disturbed condition. A binary data stream of 6-DoF[6] (12 bytes) with a 2 bytes synchronization header was recorded for a period of $1\,s$ while keeping the board in a neutral position with minimum of disturbing factors. The δt was not added to the data stream as it was observed to be equal for all measurements in the binary streaming mode if the DAQ and streaming operations are performed at the maximum rate (described in the previous section).

RF output power and offset test: An experiment was carried out in order to ensure that the output power of the transceiver is correct and constant in all the bands as declared in the datasheet. It was also necessary to determine the carrier frequency offset levels in all the bands of the transceiver to ensure its proper operability.

Wireless link test: The wireless link validation procedure was designed in order to ensure the correct functioning of the transceiver as well as the related HAL and control library made for it in various communication modes. The parameters varied were the data-rate, payload size, control check-sum presence and its size, acknowledged and non-acknowledged modes. The variation of the distance was not scope of this test as the antenna and RF front-end may vary depending on the desired satellite architecture though it was observed that with a typical dipole antenna for $2.4\,GHz$ band the $20\,km$ distance may be achieved at least.

[6]DoF in this context stands for “degrees of freedom”.

5.5.2. Sensor subsystem validation

These tests were proposed by Bardolet Santacreu [1]. They were divided in five main aspects:

Radiation: Radiation effects can cause degradation but also failure of the electronic and the electrical systems.

Vibrations: During the launch, the satellite will be working under high vibrations conditions. It is also normal that vibrations on the satellites are tested on ground to validate them. Space related facilities use big electrodynamic shakers and hydraulic shakers to test in a wide range in vibration testing

Temperature: In orbit, the temperature changes are fast and beyond the ranges inside the atmosphere. First, it is necessary to study the thermal budget to know the temperature range that has to withstand the satellite. Afterwards, some tests under the minimum and maximum values can be done to assure that it will not fail under the calculated thermal conditions. The calibrated oven for high temperatures and a freezer for low temperatures can be used. Fast temperature changes must be also tested; but it will be more difficult to reproduce.

Electromagnetic compatibility: Before launching the satellite, it is necessary to be sure that there are no interferences between inner components of the satellite and also from other electromagnetic sources. The electrostatic discharge has to be tested as well. Electromagnetic compatibility tests can be done using an anechoic chamber or a sniffer.

Vacuum: In vacuum conditions, out-gassing occurs in every material. Every day expose materials become weaker. As it happens with under radiation conditions, can be said that for the short lifetime of our satellite (8-9 days) there is no need to worry for the vacuum exposition.

In order to avoid double work, these tests were proposed to be done within the complete satellite.

5.5.3. Communication subsystem validation

These tests were lead by Fernández-Murcia and they are widely explained in [4]. Experimental tests are divided in two main groups:

Laboratory tests: Each circuit and the antenna were test in a RF laboratory with a VNA and a RF generator.

Open area tests: Two simple open area tests (each one longer than previous one) and the final balloon tests (which validates the entire system) were defined:

- $400m$ test. This is only a quick test to assure all works as expected. All was as expected.
- $6,5$ kilometers test. This is critical because determines if all the work done before goes in a good direction or not. After some problems in transceivers feeding, spot and antennas pointing, the tests finished satisfactorily (from 41.26784, 1.923287 to 41.265227, 1.996071). A $7\,dBm$ power transmission achieved using PA2423L [25]. No LNA on reception.

5.5.4. Power management subsystem validation

This subsystem includes the IOExpander [24] and the voltage regulators [27].

For the IOExpander, an evaluation board shown in Figure 5.10 was designer. This board will facilitate connect the device to the $Arduino$[7] in order to undergo some tests. The developed code is shown in Appedix D.1. Due to the lack of time, this board was designed but not build, so this test remain pending for further work.

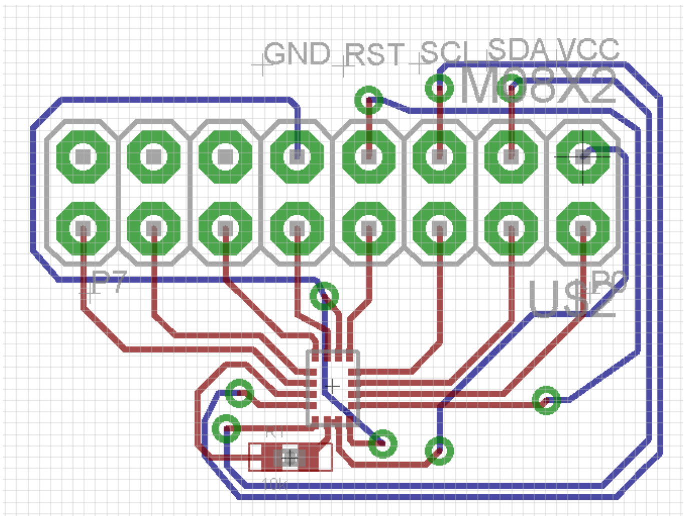


Figure 5.10: IOExpander evaluation board

In order to study the behavior of the regulator a practical case has been used. First of all, as with the rest of IC, an evaluation board was built. This board is shown in Figure 5.11. For this experiment the $TPS7192615$ regulator was selected to supply the HD Camera [20] during its tests. The voltage was directly connected to the $Arduino$ output voltage $(3.3V)$ in order to simulate an scenario as closer as possible to the mission one. Some measurements of the output voltages of the regulator were done and they were the expected ones. As the HD Camera worked properly during the tests, can be assured that this device will perform correctly during the mission.

[7]Seeduino v2.21 with ATmega328P has been used, http://www.seeedstudio.com/depot/seeeduino-v221-atmega-328p-p-669.html

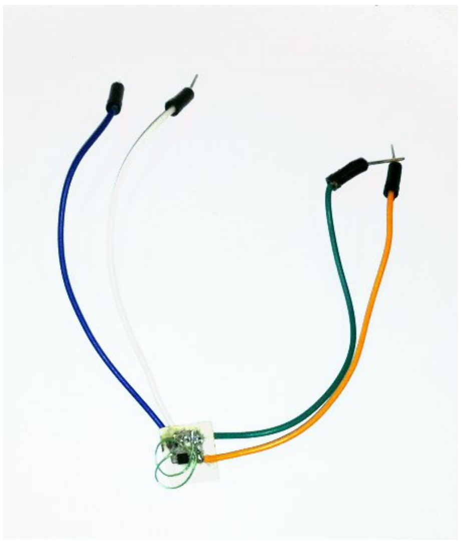


Figure 5.11: Voltage regulator evaluation board

CHAPTER 6. PAYLOAD IMPLEMENTATION

This chapter is going to present how the HD Camera [20] and the Serializer [28] can work together in order to obtain and transmit images.

6.1. Evaluation boards

The first step was to evaluate the behavior of each device independently. To do that, two evaluation boards were designed and built following the same procedure explained in Chapter 5. These boards are show in Figure 6.1. As can be seen, the design of both boards is very simple, only required pins have been connected. Also a connector for protoboard has been added.

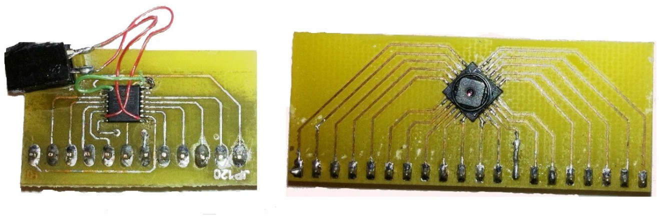


Figure 6.1: a) Serializer evaluation board and b) Camera evaluation board

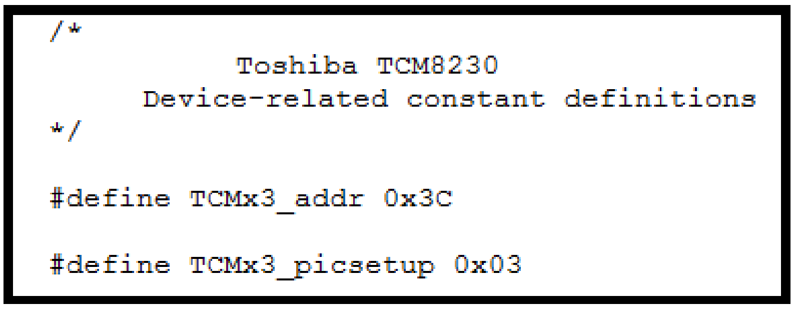
These boards let connect the devices to the *Arduino* in order to undergo some tests.

6.2. Test and validation

In order to test the camera and serializer it is necessary to check if they are going to be able to work together and if they could be managed by the MCU of the satellite. For this reason, all the tests have been done using the a processor of the same family. First, they were tested separately. Once each device was validated, the entire subsystem was validated following the same procedure.

6.2.1. Camera test and validation

It is necessary to send an star-up sequence through the I^2C port to the camera in order to activate it. The first step was to define a library containing the main registers of the camera and then to write a some code to send the required start-up sequence. Figure 6.2 shows the aforementioned library. The specific code is in Appendix E.1.



```
/*
            Toshiba TCM8230
      Device-related constant definitions
*/

#define TCMx3_addr 0x3C

#define TCMx3_picsetup 0x03
```

Figure 6.2: TCM8230MD constant definitions

To check the camera output, a Logic Analyzer[1] has been used. Figure 6.3 shows a frame example. Lines from 0 to 7 correspond to D0 (LSB) to D7 (MSB), lines 8 and 9 are VD (Vertical Detection) and HD (Horizontal Detection) respectively.

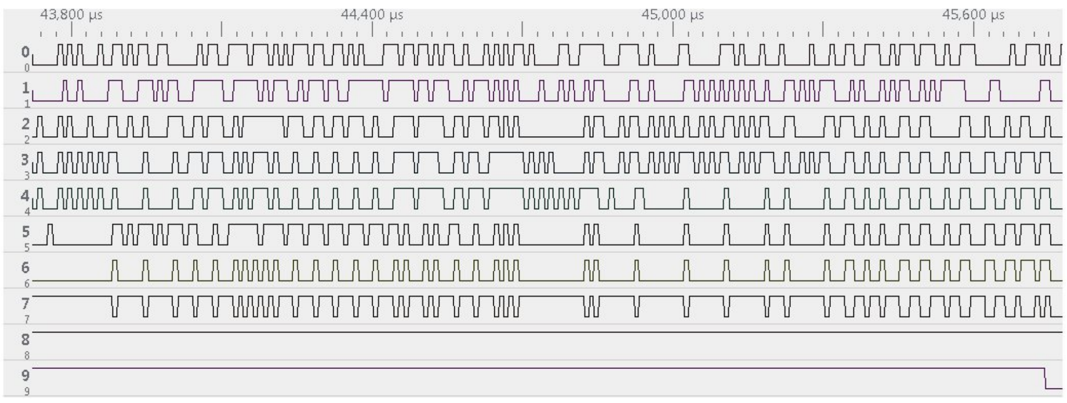


Figure 6.3: TCM8230MD output

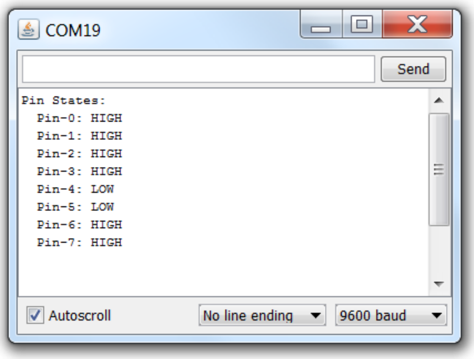
6.2.2. Serializer test and validation

As the camera, the serializer needs a start sequence in order to be activate. In this case, the procedure has been the same. Some code has been written in order to evaluate the performance of the $SN74HC165$ [28]. The code is shown in Appendix E.2. In order to check the correct behavior of the serializer, inputs values have been forced connecting these inputs directly to ground or to $3.3V$(0 to $0V$, 1 to $3.3V$). Figure 6.4 shows the test results.

6.2.3. Payload test and validation

In order to get both devices (camera and serializer), it was necessary to have a clock reference to get them synchronized. To do that, some code able to generate pulse signals

[1]Open Logic Sniffer, http://www.sparkfun.com/products/9857



```
Pin States:
  Pin-0: HIGH
  Pin-1: HIGH
  Pin-2: HIGH
  Pin-3: HIGH
  Pin-4: LOW
  Pin-5: LOW
  Pin-6: HIGH
  Pin-7: HIGH
```

Figure 6.4: Serializer test

was written. This code is show in Appendix E.4. Once the subsystem was integrated, some test were done in lab, using an oscilloscope to display the output of the serializer. Figure 6.5 shows the results obtained when doing these tests.

In a signals $CLKINH$ (yellow) and CLK (blue, clock reference generated) can be shown. $CLKINH$ is eight times slower, that means that, when $CLKINH$ is in low level and the SH/LD signal is in high level, starts the serialization in the first eight clock cycles. When $CLKINH$ is high and SH/LD is low, starts the data load. During these sixteen clock cycles, one byte of each pair is lost due to that, when doing the load, only the last cycle is valid. In order not to loose any byte, a counter will be necessary in order to act as a trigger each eight cycles but this circuit will add complexity to the femtosatellite. Figure 6.5 b just compares the CLK signal with the $EXTCLK$ of the camera. Finally, in c a capture of the output is shown. In this moment, is taken place the serialization of the byte $F3$ (11110011).

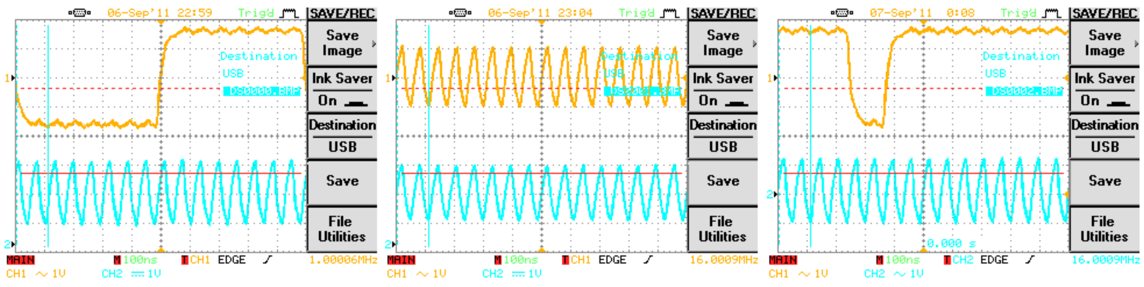


Figure 6.5: a) Clock references, b) Clock comparisons and c) Serializer output

CHAPTER 7. CONCLUSIONS

7.1. General conclusions

During the development of this project the entire *WikiSat* team has explored in a practical case of a new paradigm for the development of very small space missions. This new approach of development can result, if successful, in the birth of a new market for femtosatellites. These kind of missions should be very focused on a single goal, but the system can perform most of the typical applications undertaken by larger mass systems. However, it must be stated that very small satellites will not enter into competition with standard satellites. They could open new markets and opportunities, could be used for education, disaster management and even communications on an extremely low cost and complexity.

It has been demonstrated that it is possible to design, test and build a femtosatellite (with less than 20 grams) that is able to perform a short duration mission. Specifically a Sun-tracker mission has been developed. Likewise, an example of with an imaging payload with an HD Camera have been designed and implementer. Finally, it has been proved that the new space payload paradigm can contribute to a simpler, faster and much cheaper way of producing very small missions.

7.2. Environmental impact

The achievement of this project will deliver a satellite to the space. The launch phase is the most contaminant stage of the mission, but debris will reenter soon into the Earth atmosphere. Low cost and lead-free materials were used in this project in order to minimize environmental impact. To reduce the amount of waste produced during the work some recycled components were used. All the boards with production defects were recycled and reused in other prototypes as well.

This project does not mean an important impact upon the environment because of the reduced sizes and the low orbit used. The estimated time of the re-entry is between 8 and 9 days so it can be assured that there will be no generation of space debris. The small size of the femtosatellite assures that it will be disintegrated during the re-entry. For the mini-launcher, it is necessary to study all cases but the idea is not to generated debris either. Moreover, the pollution generated will be much lower that the generated by a big launcher. So, the use of small mini-launcher not only implies a reduction of cost, but also a reduction in pollution and a lower impact on the environment. This kind of small size technologies also is translated into power savings.

Regarding the economic and social field, the use of low cost technologies for space applications can open a market that has been inaccessible for many years. Governments, companies and organizations unable to afford the costs of space missions may will take benefit from this project.

7.3. Future work

It is also important to mention that further experimentation and research work is required in such fields like image processing and storage. It is necessary to improve these aspects in order to be able to obtain a good quality image fulfilling the system requirements explained in Chapter 2.

Another development challenge will be introduction of new improved devices such:

- A three axes accelerometer and compass single chip like the LSM303DLHC [21] from *STMicroelectronics*. It is going to used to guide the launcher during the trajectory-
- The L3G4200D [23] from *STMicroelectronics*, a three axes gyro that is going to be used for short maneuvers for the camera or antenna pointing.

The *WikiSat* team is already working in the *WikiSat V.5*. This version will include the improvements explained above.

BIBLIOGRAPHY

[1] E. Bardolet. Study of a low cost inertial platfom for a femto-satellite deployed by a mini-launcher, 2010.

[2] Lluis Bonet. High altitude balloon mission design and implementation for a mini-launcher. Master's thesis, Universitat Politècnica de Catalunya, 2011.

[3] et al. D. J. Barnhart. A low-cost femtosatellite to enable distributed space missions. *Acta Astronautica 64*, 2009.

[4] Enric Fernàndez Murcia et al. A synthetic aperture antenna for femto-satellites based on commercial-of-the-shelf. *29th Digital Avionics Systems Conference. Seattle (2011)*, 2011.

[5] R. González and J. Naudó. A multi-agent adaptive protocol for femto-satellite applications. Master's thesis, Universitat Politécnica de Catalunya, 2011.

[6] C. D. Hall. Virginia tech ionospheric scintillation measurement mission. *13th Annual AIAA/USU Conference on Small Satellites*, 1999.

[7] C. Hamroun. Design and prototype of a flight microstrip antennas for pico satellite erpsat-1. *4ht International Conference, Istanbul*, 2009.

[8] H. Helvajian. The fabrication of a 100 g co-orbiting satellite assistant (cosa) using glass ceramic materials and 3-d. *Proceedings of the Ninth International Micromachine/Nanotech Symposium, Tokyo, Japan*, 2003.

[9] H. Herman. *HANDBOOK OF ASTRONAUTICAL ENGINEERING*. Ed. McGraw-Hill, Inc., 1961.

[10] James R. Wertz et. al. Space Mission Engineering: The new SMAD, Chapter 1.4.7. Space Technology Library, 2011.

[11] et al. J. Tristancho. A probe of concept of femto-satellite based on commercial-of-the-shelf. *30th Digital Avionics Systems Conference, USA*, 2011.

[12] Kravchenko, Victor Design and implementation of a femto-satellite technology demonstrator. Master's thesis, Universitat Politècnica de Catalunya, 2011.

[13] et al. K. Doerksen. Design, modeling and evaluation of a 2.4ghz fhss communications system for narcissat. *17th AIAA/USU Conference on Small Satellites, Stanford, CA, USA*, 2003.

[14] Laia Navarro Morcillo. Use of hardware-on-the-loop to test missions for a low cost mini-launcher. Master's thesis, Universitat Politècnica de Catalunya, 2011.

[15] Laia Molas Pous. Efectes de la radiacio sobre els components electronics d'un satelit en orbita al voltant de la terra. Master's thesis, Universitat Politécnica de Catalunya, 2011.

[16] Clayton Mowry. Interview on ”The Space Show”, 2010.

[17] Joshua Tristancho. Implementation of a femto-satellite and a mini-launcher. Master's thesis, Universitat Politècnica de Catalunya, 2010.

[18] ST Electronics. LIS331HH Data Sheet. http://www.sparkfun.com/datasheets/Sensors/Accelerometer/LIS331HH.pdf, Last visit on 09/13/2011.

[19] Atmel. ATMega 168 Data Sheet. http://www.atmel.com/dyn/resources/prod_documents/doc2545.pdf, Last visit on 09/13/2011.

[20] TOSHIBA. TCM8230MD Data Sheet. http://www.sparkfun.com/datasheets/Sensors/Imaging/TCM8230MD.pdf, Last visit on 09/13/2011.

[21] ST Electronics. LSM303DLHC Data Sheet. http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/DM00027543.pdf, Last visit on 09/13/2011.

[22] InvenSense. ITG-3200 Data Sheet. http://invensense.com/mems/gyro/documents/EB-ITG-3200-00-01.1.pdf, Last visit on 09/13/2011.

[23] ST Electronics. L3G4200D Data Sheet. http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00265057.pdf, Last visit on 09/13/2011.

[24] Texas Instruments. TCA6408 Data Sheet. http://www.ti.com/lit/ds/symlink/tca6408.pdf, Last visit on 09/13/2011.

[25] SiGe Semiconductor. PA2423L Data Sheet. http://www.skyworksinc.com/uploads/documents/SiGe/13-DST-01_PA2423L_Brief_Rev_4p1_AP_May-26-2009.pdf, Last visit on 09/13/2011.

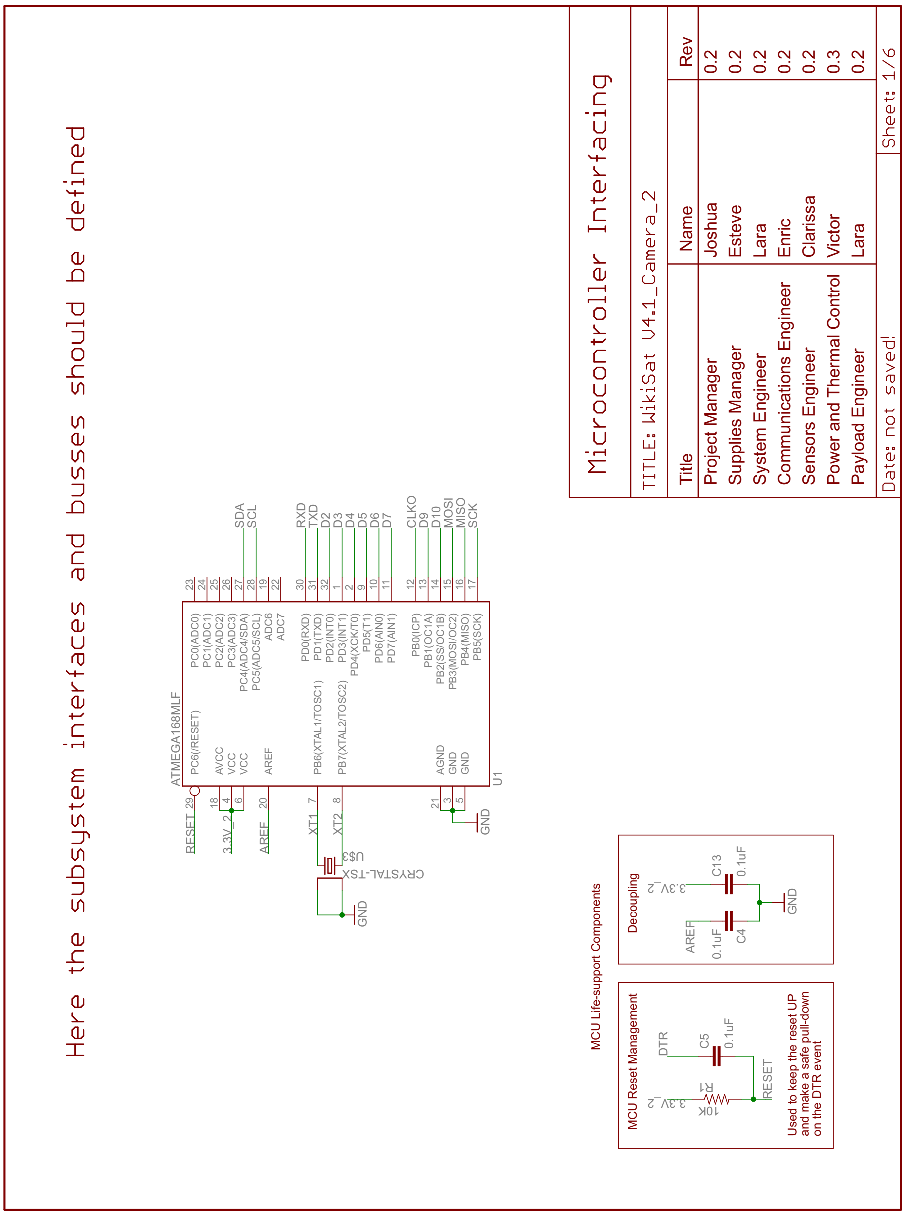
[26] Nordic Semiconductor. nRF24L01+ Data Sheet. http://www.nordicsemi.com/eng/Products/2.4GHz-RF/nRF24L01P, Last visit on 09/13/2011.

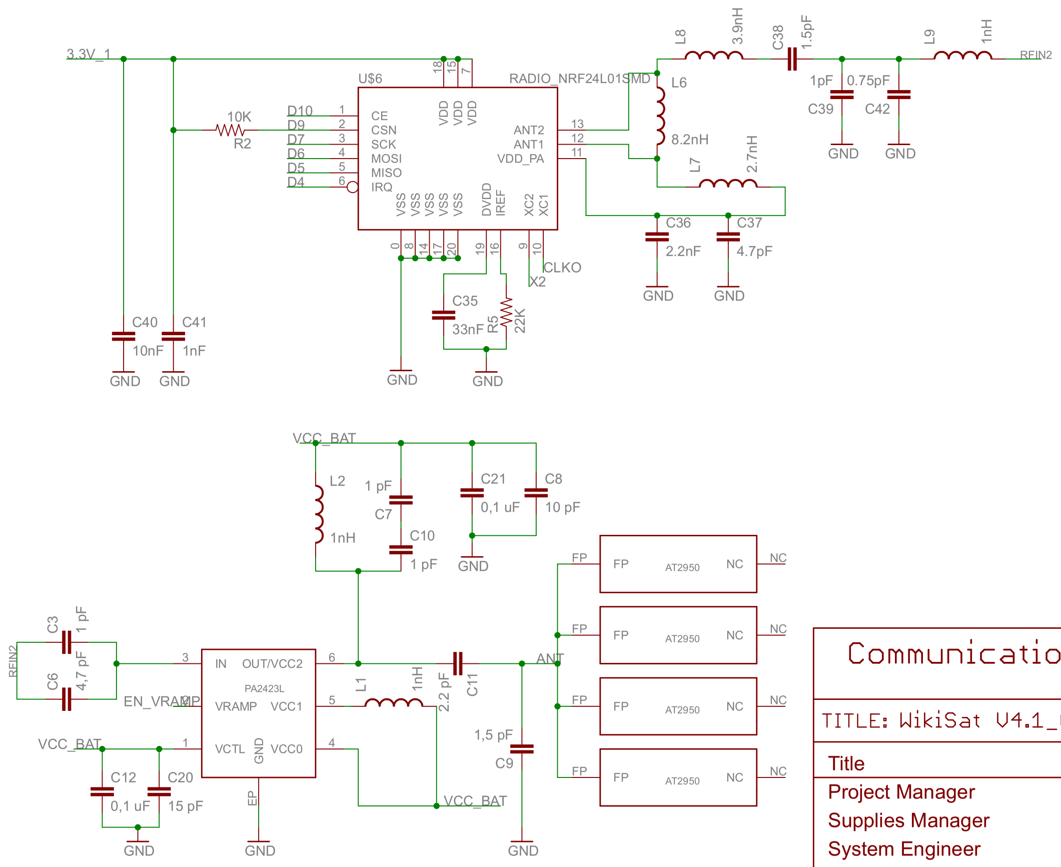
[27] Texas Instruments. TPS719XXXX Data Sheet. http://www.ti.com/lit/ds/symlink/tps71918-12.pdf, Last visit on 09/13/2011.

[28] Texas Instruments. SN74HC165 Data Sheet. http://www.sparkfun.com/datasheets/Components/General/sn74hc165.pdf, Last visit on 09/13/2011.

APPENDIX

APPENDIX A. WIKISAT V4.1 SCHEMATICS



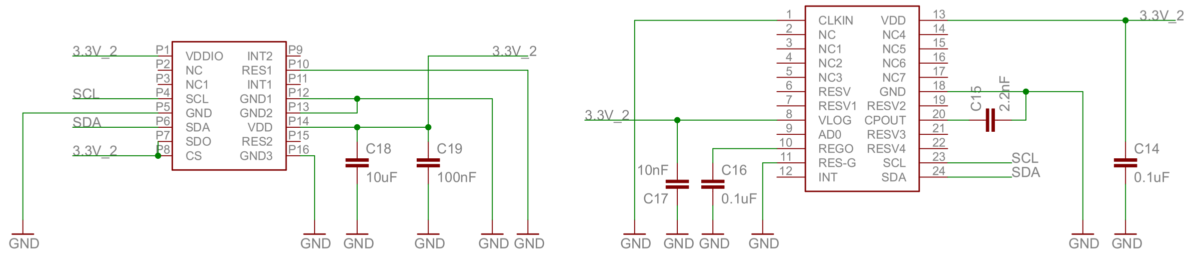


Communication Subsystem

TITLE: WikiSat V4.1_Camera_2

Title	Name	Rev
Project Manager	Joshua	0.2
Supplies Manager	Esteve	0.2
System Engineer	Lara	0.2
Communications Engineer	Enric	0.2
Sensors Engineer	Clarissa	0.2
Power and Thermal Control	Victor	0.3
Payload Engineer	Lara	0.2

Date: not saved! Sheet: 2/6

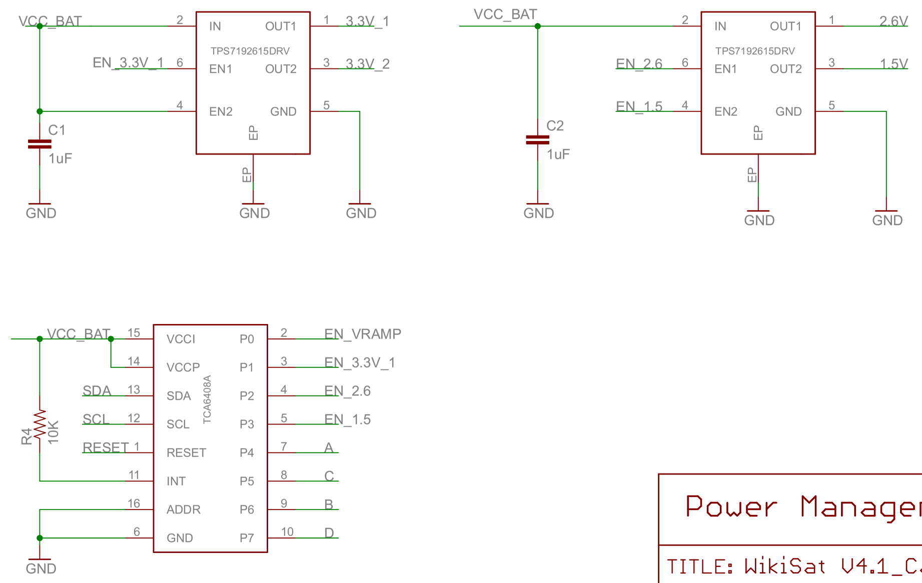

3.3V_2
SCL
SDA
GND
P1 P2 P3 P4 P5 P6 P7 P8
VDDIO NC NC1 SCL GND SDA SDO CS
INT2 RES1 INT1 GND1 GND2 VDD RES2 GND3
P9 P10 P11 P12 P13 P14 P15 P16
C18 10uF
C19 100nF
CLKIN NC NC1 NC2 NC3 RESV RESV1 VLOG AD0 REGO RES-G INT
VDD NC4 NC5 NC6 NC7 GND RESV2 CPOUT RESV3 RESV4 SCL SDA
1 2 3 4 5 6 7 8 9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24
10nF C17
C16 0.1uF
C15 2.2nF
C14 0.1uF

Sensor Subsystem

TITLE: WikiSat V4.1_Camera_2

Title	Name	Rev
Project Manager	Joshua	0.2
Supplies Manager	Esteve	0.2
System Engineer	Lara	0.2
Communications Engineer	Enric	0.2
Sensors Engineer	Clarissa	0.2
Power and Thermal Control	Victor	0.3
Payload Engineer	Lara	0.2

Date: not saved! Sheet: 3/6

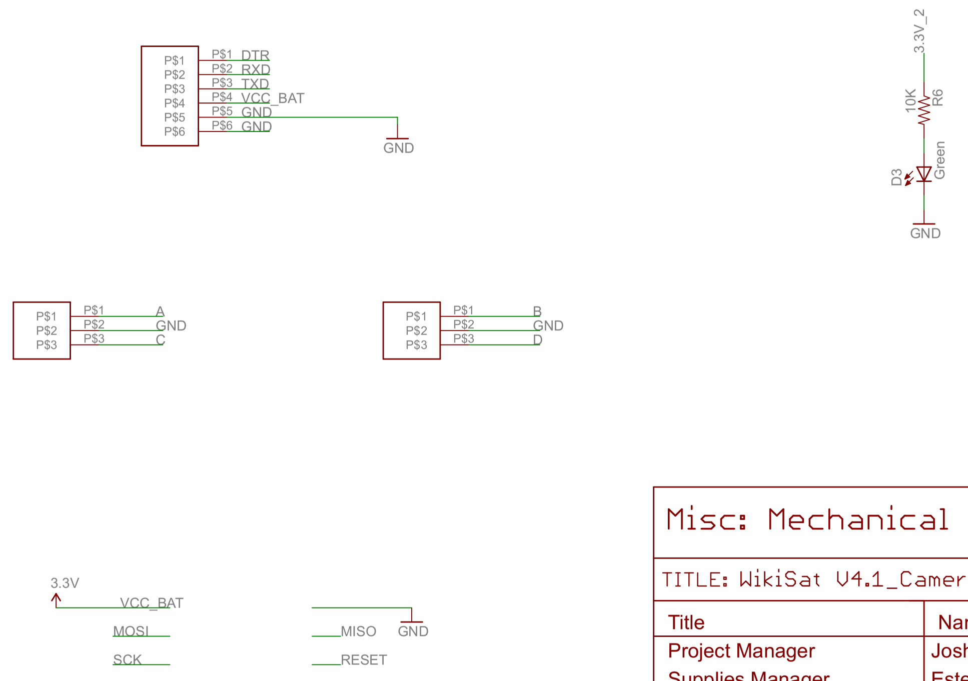


Power Management Subsystem

TITLE: WikiSat V4.1_Camera_2

Title	Name	Rev
Project Manager	Joshua	0.2
Supplies Manager	Esteve	0.2
System Engineer	Lara	0.2
Communications Engineer	Enric	0.2
Sensors Engineer	Clarissa	0.2
Power and Thermal Control	Victor	0.3
Payload Engineer	Lara	0.2

Date: not saved! Sheet: 4/6

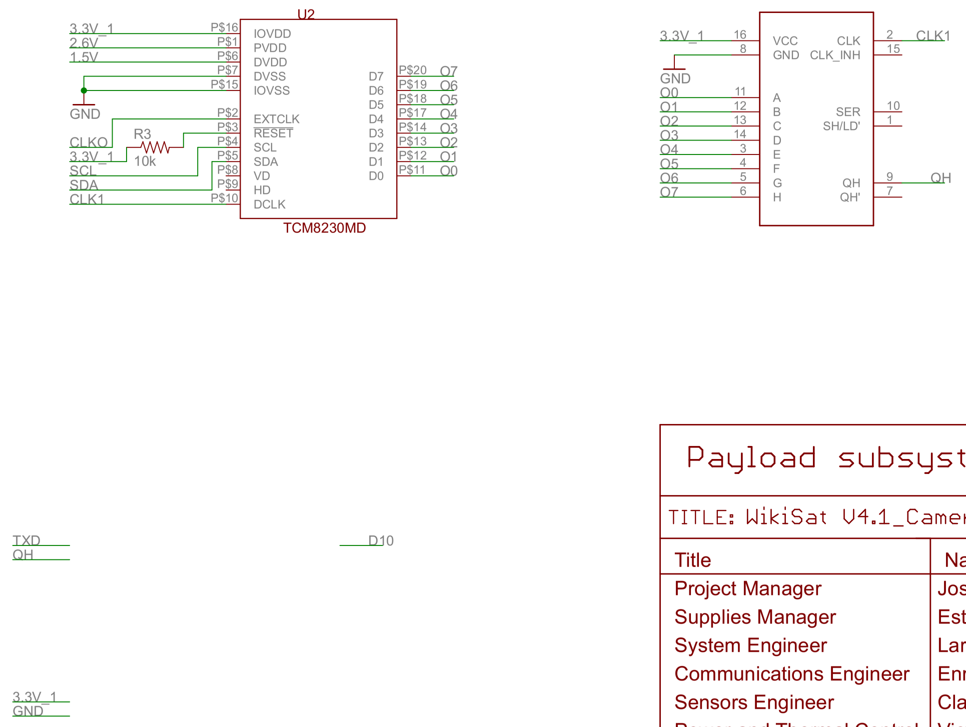


Misc: Mechanical Interfaces

TITLE: WikiSat V4.1_Camera_2

Title	Name	Rev
Project Manager	Joshua	0.2
Supplies Manager	Esteve	0.2
System Engineer	Lara	0.2
Communications Engineer	Enric	0.2
Sensors Engineer	Clarissa	0.2
Power and Thermal Control	Victor	0.3
Payload Engineer	Lara	0.2

Date: not saved! Sheet: 5/6



Payload subsystem

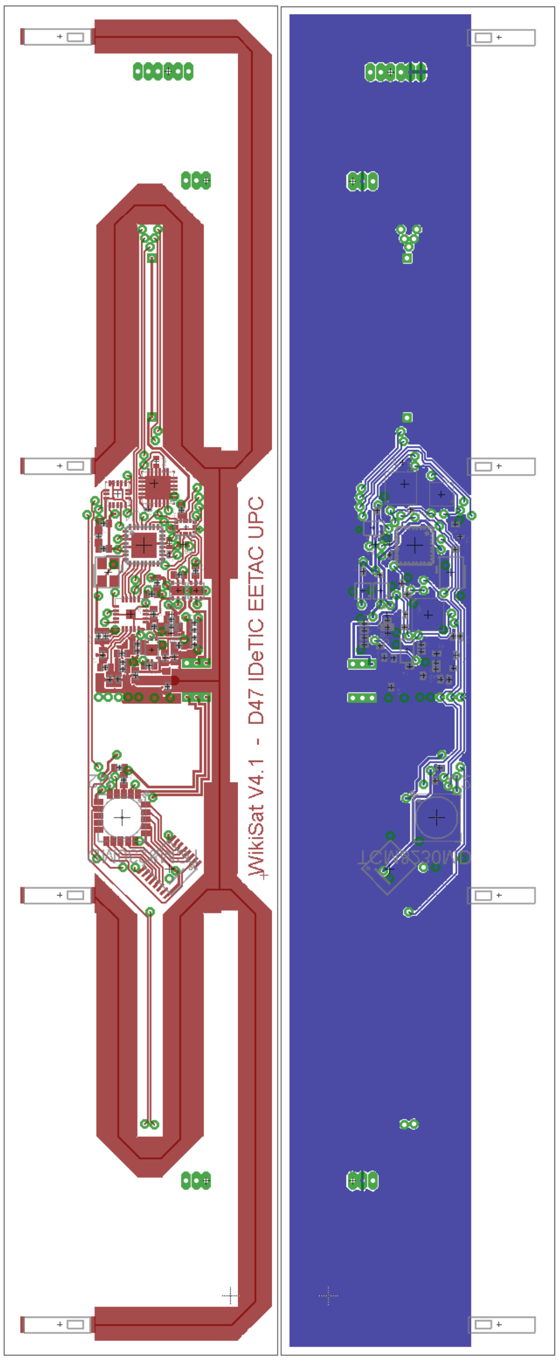
TITLE: WikiSat V4.1_Camera_2

Title	Name	Rev
Project Manager	Joshua	0.2
Supplies Manager	Esteve	0.2
System Engineer	Lara	0.2
Communications Engineer	Enric	0.2
Sensors Engineer	Clarissa	0.2
Power and Thermal Control	Victor	0.3
Payload Engineer	Lara	0.2

Date: not saved!

Sheet: 6/6

APPENDIX B. WIKISAT V4.1 BOARD



APPENDIX C. WIKISAT V4.1 ASSEMBLY FORM

Part	Value	Device	Package
C1	1uF	CAP	0402
C2	1uF	CAP	0402
C3	1 pF	CAP	0402
C4	0.1uF	CAP	0402
C5	0.1uF	CAP	0402
C6	4.7pf	CAP	0402
C7	1pF	CAP	0402
C8	10pF	CAP	0402
C9	1.5pF	CAP	0402
C10	1pF	CAP	0402
C11	2.2pF	CAP	0402
C12	0.1uF	CAP	0402
C13	0.1uF	CAP	0402
C14	0.1uF	CAP	0402
C15	2.2nF	CAP	0402
C16	0.1uF	CAP	0402
C17	10nF	CAP	0402
C18	10uF	CAP	0603
C19	100nF	CAP	0402
C20	15pF	CAP	0402
C21	0.1uF	CAP	0402
C35	33nF	CAP	0402
C36	2.2nF	CAP	0402
C37	4.7pF	CAP	0402
C38	1.5pF	CAP	0402
C39	1pF	CAP	0402
C40	10nF	CAP	0402
C41	1nF	CAP	0402
C42	0.75pF	CAP	0402
D3	Green	LED	0603
L1	1nH	INDUCTOR	0402
L2	1nH	INDUCTOR	0402
L6	8.2nH	INDUCTOR	0402
L7	2.7nH	INDUCTOR	0'402
L8	3.9nH	INDUCTOR	0402
L9	1nH	INDUCTOR	0402
R1	10K	RESISTOR	0201
R2	10K	RESISTOR	0402
R3	10k	RESISTOR	0402
R4	10K	RESISTOR	0402
R5	22K	RESISTOR	0402
R6	10K	RESISTOR	0402

APPENDIX D. POWER MANAGEMENT SUBSYSTEM SOURCE CODE

D.1. TCA6408, Source Code

```
#include <Wire.h>
extern "C"
{
  #include "utility/twi.h"  // from Wire library, so we can do bus scanning
}

/* Sensors */
#define AccID B00011001 // Accelerometer LIS331HH (0x19)
#define AccX 0x29, 0x28
#define AccY 0x2B, 0x2A
#define AccZ 0x2D, 0x2C
#define GyroID B01101001 // Gyro ITG3200 (0x69)
#define GyroT 0x1B, 0x1C
#define GyroX 0x1D, 0x1E
#define GyroY 0x1F, 0x20
#define GyroZ 0x21, 0x22
#define IOeID B01000000 // IO expander write on TCA6408A (0x40)
#define IOeW 0x01 // Write IOe ports
#define IOrID B01000001 // IO expander read on TCA6408A (0x41)
#define IOrW 0x00 // Read IOe configuration
#define IOeEN_VRAMP B00000001 // P0
#define IOeEN_3_3V_1 B00000010 // P1
#define IOeEN_2_6V B00000100 // P2
#define IOeEN_1_5V B00001000 // P3
#define IOeEN_A B00010000 // P4 Magnetorquer A - LEFT UP and Turn +v (Yaw)
#define IOeEN_B B00100000 // P5 Magnetorquer B - RIGHT UP and Turn -v (Yaw)
#define IOeEN_C B01000000 // P6 Magnetorquer C - LEFT DOWN and Turn +u (Pitch)
#define IOeEN_D B10000000 // P7 Magnetorquer D - RIGHT DOWN and Turn -u (Pitch)


// ---------------------------------------------------------------------------
// --------------------------------I2C Bus------------------------------------
// ---------------------------------------------------------------------------
void I2C_SetRegister(int device, int address, int value)
{
 Wire.beginTransmission(device);
 Wire.send(address);
 Wire.send(value);
 Wire.endTransmission();
}
byte I2C_GetRegister(int device, int address)
{
 Wire.beginTransmission(device);
 Wire.send(address);
 Wire.endTransmission();
 Wire.requestFrom(device, 1);
 if(Wire.available()) return Wire.receive();
 return B00000000;
}
int I2C_GetValue(int device, int addressH, int addressL)
{
 return ((unsigned int)(I2C_GetRegister(device, addressH)) << 8) + I2C_GetRegister(device, ↩
     addressL);
}
void I2C_TurnOn(byte mask)
{
  I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | mask);
}
void I2C_TurnOff(byte mask)
{
  I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) &~ mask);
}
// BORRAR
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_A); // Active magnetorquer ↩
     P4 A — LEFT UP and Turn +v (Yaw)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_B); // Active magnetorquer ↩
     P5 B — RIGHT UP and Turn —v (Yaw)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_C); // Active magnetorquer ↩
     P6 C — LEFT DOWN and Turn +u (Pitch)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_D); // Active magnetorquer ↩
     P7 D — RIGHT DOWN and Turn —u (Pitch)

void setup()
{
  Serial.begin(4800); // Remove this function if memory required
  Wire.begin(); // I2C bus for femto—satellite sensors communications
  Serial.println("WIKISAT READY TO LAUNCH"); // Remove this message if memory required
}


void loop()
{
  I2C_TurnOn(IOeEN_A);
  I2C_TurnOff(IOeEN_A);
  delay(1000);
  I2C_TurnOn(IOeEN_B);
  I2C_TurnOff(IOeEN_B)
  delay(1000);
  I2C_TurnOn(IOeEN_C);
  I2C_TurnOff(IOeEN_C)
  delay(1000);
  I2C_TurnOn(IOeEN_D);
  I2C_TurnOff(IOeEN_D)
  delay(1000);
}
```

APPENDIX E. PAYLOAD SUBSYSTEM SOURCE CODE

E.1. TCM8230MD, Source Code

```
#include <Wire.h>

void i2cSetRegister(int device, int adress, int value){
  Wire.beginTransmission(device); Wire.send(adress);
  Wire.send(value); Wire.endTransmission();
}
byte i2cGetRegister(int device, int adress){
  Wire.beginTransmission(device); Wire.send(adress);
  Wire.endTransmission(); Wire.requestFrom(device, 1);
  if(Wire.available()) return Wire.receive(); return B00000000;
}

void setup()
{

  pinMode(10, INPUT);
  pinMode(10, OUTPUT);

  Wire.begin();

  pinMode( 7, OUTPUT);
  digitalWrite( 7, LOW);

  delay(500);
  digitalWrite( 7, HIGH);
  delay(200);
  i2cSetRegister(0x3C, 0x03, B00100000);

void loop()
{
  int buttonState = LOW;
  buttonState = digitalRead(10);

  if (buttonState == HIGH) {
    // turn LED on:
    digitalWrite(13, HIGH);
  }

}
```

E.2. Serializer, Source Code

```
/*
 * SN74HC165N_shift_reg
 *
 * Program to shift in the bit values from a SN74HC165N 8-bit
 * parallel-in/serial-out shift register.
 *
*/

/* How many shift register chips are daisy-chained.
*/
#define NUMBER_OF_SHIFT_CHIPS   1

/* Width of data (how many ext lines).
*/
#define DATA_WIDTH   NUMBER_OF_SHIFT_CHIPS * 8

/* Width of pulse to trigger the shift register to read and latch.
*/
#define PULSE_WIDTH_USEC   5

/* Optional delay between shift register reads.
*/
#define POLL_DELAY_MSEC   1

/* You will need to change the "int" to "long" If the
 * NUMBER_OF_SHIFT_CHIPS is higher than 2.
*/
#define BYTES_VAL_T unsigned int

int ploadPin        = 6;  // Connects to Parallel load pin the 165
int clockEnablePin  = 3;  // Connects to Clock Enable pin the 165
int dataPin         = 2; // Connects to the Q7 pin the 165
int clockPin        = 5; // Connects to the Clock pin the 165

BYTES_VAL_T pinValues;
BYTES_VAL_T oldPinValues;

/* This function is essentially a "shift-in" routine reading the
 * serial Data from the shift register chips and representing
 * the state of those pins in an unsigned integer (or long).
*/
BYTES_VAL_T read_shift_regs()
{
    byte bitVal;
    BYTES_VAL_T bytesVal = 0;

    /* Trigger a parallel Load to latch the state of the data lines,
    */
    digitalWrite(clockEnablePin, HIGH);
    digitalWrite(ploadPin, LOW);
    delayMicroseconds(PULSE_WIDTH_USEC);
    digitalWrite(ploadPin, HIGH);
    digitalWrite(clockEnablePin, LOW);

    /* Loop to read each bit value from the serial out line
     * of the SN74HC165N.
    */
    for(int i = 0; i < DATA_WIDTH; i++)
    {
        bitVal = digitalRead(dataPin);

        /* Set the corresponding bit in bytesVal.
        */
        bytesVal |= (bitVal << ((DATA_WIDTH-1) - i));

        /* Pulse the Clock (rising edge shifts the next bit).
        */
        digitalWrite(clockPin, HIGH);
        delayMicroseconds(PULSE_WIDTH_USEC);
        digitalWrite(clockPin, LOW);
    }

    return(bytesVal);
}

/* Dump the list of zones along with their current status.
*/
void display_pin_values()
{
    Serial.print("Pin States:\r\n");

    for(int i = 0; i < DATA_WIDTH; i++)
    {
        Serial.print("  Pin-");
        Serial.print(i);
        Serial.print(": ");

        if((pinValues >> i) & 1)
            Serial.print("HIGH");
        else
            Serial.print("LOW");

        Serial.print("\r\n");
    }

    Serial.print("\r\n");
}

void setup()
{
    Serial.begin(9600);

    /* Initialize our digital pins...
    */
    pinMode(ploadPin, OUTPUT);
    pinMode(clockEnablePin, OUTPUT);
    pinMode(clockPin, OUTPUT);
    pinMode(dataPin, INPUT);

    digitalWrite(clockPin, LOW);
    digitalWrite(ploadPin, HIGH);

    /* Read in and display the pin states at startup.
    */
    pinValues = read_shift_regs();
    display_pin_values();
    oldPinValues = pinValues;
}

void loop()
{
    /* Read the state of all zones.
    */
    pinValues = read_shift_regs();

    /* If there was a chage in state, display which ones changed.
    */
    if(pinValues != oldPinValues)
    {
        Serial.print("*Pin value change detected*\r\n");
        display_pin_values();
        oldPinValues = pinValues;
    }

    delay(POLL_DELAY_MSEC);
}
```

E.3. Payload, Source Code

```
//HD Camera + Serializer  + Sincronizaation signal
#include <Wire.h>

extern "C"
{
#include <FrequencyTimer2.h>
}
/* How many shift register chips are daisy-chained.
*/
#define NUMBER_OF_SHIFT_CHIPS   1

/* Width of data (how many ext lines).
*/
#define DATA_WIDTH   NUMBER_OF_SHIFT_CHIPS * 8

/* Width of pulse to trigger the shift register to read and latch.
*/
#define PULSE_WIDTH_USEC   5

/* Optional delay between shift register reads.
*/
#define POLL_DELAY_MSEC   1

//#define BYTES_VAL_T unsigned int

void i2cSetRegister(int device, int adress, int value){
  Wire.beginTransmission(device); Wire.send(adress);
  Wire.send(value); Wire.endTransmission();
}
byte i2cGetRegister(int device, int adress){
  Wire.beginTransmission(device); Wire.send(adress);
  Wire.endTransmission(); Wire.requestFrom(device, 1);
  if(Wire.available()) return Wire.receive(); return B00000000;
}


int ploadPin        = 6;  // Connects to Parallel load pin the 165
int clockEnablePin  = 3;  // Connects to Clock Enable pin the 165. Always LOW
int dataPin         = 2; // Connects to the Q7 pin the 165
int clockPin        = 5; // Connects to the Clock pin the 165

unsigned int pinValues;
unsigned int oldPinValues;

/* This function is essentially a "shift-in" routine reading the
 * serial Data from the shift register chips and representing
 * the state of those pins in an unsigned integer (or long).
*/
unsigned int read_shift_regs()
{
    byte bitVal;
    unsigned int bytesVal = 0;

    /* Trigger a parallel Load to latch the state of the data lines,
    */
    digitalWrite(clockEnablePin, HIGH);
    digitalWrite(ploadPin, LOW);
    delayMicroseconds(PULSE_WIDTH_USEC);
    digitalWrite(ploadPin, HIGH);
    digitalWrite(clockEnablePin, LOW);

    /* Loop to read each bit value from the serial out line
     * of the SN74HC165N.
    */
    for(int i = 0; i < DATA_WIDTH; i++)
    {
        bitVal = digitalRead(dataPin);
        /* Set the corresponding bit in bytesVal.
        */
        bytesVal |= (bitVal << ((DATA_WIDTH-1) - i));

        /* Pulse the Clock (rising edge shifts the next bit).
        */
        digitalWrite(clockPin, HIGH);
        delayMicroseconds(PULSE_WIDTH_USEC);
        digitalWrite(clockPin, LOW);
    }

    return(bytesVal);
}

/* Dump the list of zones along with their current status.
*/
void display_pin_values()
{
    Serial.print("Pin States:\r\n");

    for(int i = 0; i < DATA_WIDTH; i++)
    {
        Serial.print("  Pin-");
        Serial.print(i);
        Serial.print(": ");

        if((pinValues >> i) & 1)
            Serial.print("HIGH");
        else
            Serial.print("LOW");

        Serial.print("\r\n");
    }

    Serial.print("\r\n");
}


void setup()
{

  pinMode(10, INPUT);
  pinMode(10, OUTPUT);
  pinMode(11, OUTPUT);
  pinMode(ploadPin, OUTPUT);
  pinMode(clockEnablePin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, INPUT);
  pinMode( 7, OUTPUT);

  Serial.begin(9600);
  Wire.begin();

  //Camera
  digitalWrite( 7, LOW);
  delay(500);
  digitalWrite( 7, HIGH);
  delay(200);
  i2cSetRegister(0x3C, 0x03, B00100000);
  delay(1); //2100 ciclos de EXTCLK

  //Syncronizaion signal

  FrequencyTimer2::disable(); // Turn off toggling of pin 11
  FrequencyTimer2::enable(); // Turn on toggling of pin 11

  //Serializador
  digitalWrite(clockEnablePin, LOW);
  digitalWrite(clockPin, LOW);
  digitalWrite(ploadPin, HIGH);

   /* Read in and display the pin states at startup.
   */
   pinValues = read_shift_regs();
   display_pin_values();
   oldPinValues = pinValues;
}

void loop()
{
     delay(POLL_DELAY_MSEC);

}
```

E.4. Synchronization signal, Source Code

```
extern "C"
{
#include <FrequencyTimer2.h>
}

void setup() {
 pinMode(11, OUTPUT);
 FrequencyTimer2::disable(); // Turn off toggling of pin 11
 FrequencyTimer2::setPeriod(2000); // Set refresh rate (interrupt timeout period)
 FrequencyTimer2::enable(); // Turn on toggling of pin 11
}

void loop()
{
}
```

APPENDIX F. PRELIMINARY FEMTOSATELLITE SOURCE FOR WIKISAT V4.1

```
#include <SPI.h>
#include <math.h>

#define RESET_EEPROM false // WARNING: TRUE WILL ERASE ALL THE EEPROM PERMANENTLY. NEW WAYPOINT ↩
    LIST WILL BE UPDATED

/* Memory Map */
#define CONFIG      0x00
#define EN_AA       0x01
#define EN_RXADDR   0x02
#define SETUP_AW    0x03
#define SETUP_RETR  0x04
#define RF_CH       0x05
#define RF_SETUP    0x06
#define STATUS      0x07
#define OBSERVE_TX  0x08
#define CD          0x09
#define RX_ADDR_P0  0x0A
#define RX_ADDR_P1  0x0B
#define RX_ADDR_P2  0x0C
#define RX_ADDR_P3  0x0D
#define RX_ADDR_P4  0x0E
#define RX_ADDR_P5  0x0F
#define TX_ADDR     0x10
#define RX_PW_P0    0x11
#define RX_PW_P1    0x12
#define RX_PW_P2    0x13
#define RX_PW_P3    0x14
#define RX_PW_P4    0x15
#define RX_PW_P5    0x16
#define FIFO_STATUS 0x17
/* Bit Mnemonics */
#define MASK_RX_DR  6
#define MASK_TX_DS  5
#define MASK_MAX_RT 4
#define EN_CRC      3
#define CRCO        2
#define PWR_UP      1
#define PRIM_RX     0
#define ENAA_P5     5
#define ENAA_P4     4
#define ENAA_P3     3
#define ENAA_P2     2
#define ENAA_P1     1
#define ENAA_P0     0
#define ERX_P5      5
#define ERX_P4      4
#define ERX_P3      3
#define ERX_P2      2
#define ERX_P1      1
#define ERX_P0      0
#define AW          0
#define ARD         4
#define ARC         0
#define PLL_LOCK    4
#define RF_DR       3
#define RF_PWR      1
#define LNA_HCURR   0
#define RX_DR       6
#define TX_DS       5
#define MAX_RT      4
#define RX_P_NO     1
#define TX_FULL     0
#define PLOS_CNT    4
#define ARC_CNT     0
```

```
#define TX_REUSE      6
#define FIFO_FULL     5
#define TX_EMPTY      4
#define RX_FULL       1
#define RX_EMPTY      0
/* Instruction Mnemonics */
#define R_REGISTER      0x00
#define W_REGISTER      0x20
#define REGISTER_MASK 0x1F
#define R_RX_PAYLOAD   0x61
#define W_TX_PAYLOAD   0xA0
#define FLUSH_TX        0xE1
#define FLUSH_RX        0xE2
#define REUSE_TX_PL    0xE3
#define NOP             0xFF
/* Defines. Radio pinout */
#define pCE 10
#define pCSN 9
#define SOFT_MOSI 6
#define SOFT_MISO 5
#define SOFT_SCK 7
#define ceHi() {digitalWrite(pCE, HIGH);}
#define ceLow() {digitalWrite(pCE, LOW);}
#define csnLow() {digitalWrite(pCSN, LOW);}
#define csnHi() {digitalWrite(pCSN, HIGH);}
// DELETE #define AWIP_CONFIG ((1<<EN_CRC) | (0<<CRCO))
/* Femto-satellite constants */
#define PLANET_RADIUS 6378137    // WGS84 equatorial Earth radius
#define A90 1.570796              // 90 degrees in radians
#define RAD 1.745329E-02          // Degrees to radians conversion factor
#define INFINITE 1E30             // Infinite value for calculations
#define INVALID 1E30              // Invalid coordinate value
#define ALT1   32000              // Launch initial altitude in meters when stage 1 ignition (↩
     based on 'LEO N-Prize Balloon East' preset)
#define ALT2   32000              // Launch middle altitude in meters (based on 'LEO N-Prize ↩
     Balloon East' preset)
#define ALT3 250000               // Launch end altitude in meters at orbit (based on 'LEO N-Prize↩
      Balloon East' preset)
#define ANGLE1    90.00           // Launch initial pitch angle in degrees when stage 1 ignition (↩
     based on 'LEO N-Prize Balloon East' preset)
#define ANGLE2    32.55           // Launch middle pitch angle in degrees (based on 'LEO N-Prize ↩
     Balloon East' preset)
#define ANGLE3     0.00           // Launch end pitch angle in degrees at orbit (based on 'LEO N-↩
     Prize Balloon East' preset)
//#define TARGET_ANGLE 33          // Target pitch angle when stage 1 ignition in degrees
#define COVERAGE_DISTANCE 1200000 // Distance from target to start the attitude maneuver
#define IGNITION_ALTITUDE 32000 // Above this altitude in meters the igniter is activated [USED ↩
     ONLY WITH SOUNDING ROCKET]
#define PARACHUTE_ALTITUDE 2000 // Below this altitude in meters the parachute is deployed [USED↩
      ONLY WITH SOUNDING ROCKET]
#define SAFETY_ALTITUDE 3000     // Safety altitude above which the parachute can be deployed [↩
     USED ONLY WITH SOUNDING ROCKET]
#define INTEGRATOR_CYCLES 4      // (1<<4)=16 Two base integrator cycles
/* EEPROM */
//#define APOGEE_ADDR 0            // 128:  NMEA apogee EEPROM Address [USED ONLY WITH SOUNDING ↩
     ROCKET]
//#define IGNITION_ADDR 128        // 128:  NMEA ignition altitude EEPROM Address [USED ONLY WITH↩
      SOUNDING ROCKET]
//#define PARACHUTE_ADDR 256       // 128:  NMEA parachute altitude EEPROM Address [USED ONLY ↩
     WITH SOUNDING ROCKET]
#define DECLINATION_ADDR 0       // 375:  Magnetosphere declination EEPROM Address
#define UNUSED_ADDR 375          // 9:    Unused EEPROM Address
#define IMU_ADDR 384             // 12:   IMU bias parameters EEPROM Address
#define LOCKED_ADDR 396          // 12:   Current target vector EEPROM Address
#define MISSION_ADDR 408         // 1:    Mission status EPROM Address
#define FLAGS_ADDR 409           // 1:    Mission flags EPROM Address
#define MAXG_FLAG B00000001      // 1bit: Accelerometer saturation
#define ATTITUDE_FLAG B00000010 // 1bit: Attitude control active
#define GPS_FLAG B00000100       // 1bit: GPS control active
#define PAYLOAD_FLAG B00001000   // 1bit: Payload On
#define LAT_ADDR 410             // 4:    Current latitude of the satellite EPROM Address
#define LON_ADDR 414             // 4:    Current longitude of the satellite EPROM Address
#define ALT_ADDR 418             // 4:    Current altitude of the orbit EPROM Address in km above↩
```

```
     Sea Level
#define CONTROL_P_ADDR 422      // 4:    Proportional parameter of the PID controller
#define CONTROL_I_ADDR 426      // 4:    Integral parameter of the PID controller
#define CONTROL_D_ADDR 430      // 4:    Derivative parameter of the PID controller
//#define TARGET_U_ADDR 434      // 4:    Target X axis rotation value of the PID controller
//#define TARGET_V_ADDR 438      // 4:    Target Y axis rotation value of the PID controller
//#define TARGET_W_ADDR 442      // 4:    Target Z axis rotation value of the PID controller
#define IOeW_ADDR 446           // IO expander status
#define TARGET_ADDR 447         // 1:    Index of the current target in the station list
#define WAYPOINTLIST_ADDR 448   // 64:  List of 8 ground stations coordinates (in degrees) ↩
    EEPROM Address
#define SUNA_PIN 0              // Analog input pin for the Sun tracker A direction
#define SUNB_PIN 1              // Analog input pin for the Sun tracker B direction
#define SUNC_PIN 2              // Analog input pin for the Sun tracker C direction
#define SUND_PIN 3              // Analog input pin for the Sun tracker D direction
#define SUNE_PIN 4              // Analog input pin for the Sun tracker E direction
#define SUNF_PIN 5              // Analog input pin for the Sun tracker F direction
#define IGNITION_PIN 12         // Output pin number to activate the igniter 5V
#define PARACHUTE_PIN 10        // Output pin number to activate the parachute 5V
#define GPSRX_PIN 11            // Input pin for GPS Rx serial port
#define GPSTX_PIN 13            // Output pin for GPS Tx serial port
#if SAFETY_ALTITUDE <= PARACHUTE_ALTITUDE
#error SAFETY_ALTITUDE must be above PARACHUTE_ALTITUDE
#endif
/* Mission sequence */
#define MISSION_INOP       0 // Any main system fail or GPS not aligned. The satellite is not ↩
    ready to be launched
#define MISSION_RAMP       1 // IMU integrators are active and waiting for balloon release. Next↩
     mission state when first stage burn-in is detected. The balloon from an external GPS or a ↩
    person actives the first stage ignition. The satellite can not active the first stage by it ↩
    self
#define MISSION_STAGE1     2 // IMU integrators are active and vector control is active. Next ↩
    mission state when first stage burnout is detected
#define MISSION_BURNOUT1   3 // IMU integrators are active and vector control is in idle in ↩
    order to keep the attitude. Next mission stage one minute before apogee and detected by low ↩
    vertical speed
#define MISSION_TURN       4 // INOP. IMU integrators are active and a vector control maneuver ↩
    is done to match the apogee plane. Next mission stage when maneuver is completed
#define MISSION_TILT       5 // INOP. IMU integrators are active and a vector control maneuver ↩
    is done to match the heading plane parallel to the ground. Next mission stage when maneuver ↩
    is completed
#define MISSION_SPIN       6 // IMU integrators are active and a spin is given to the launcher. ↩
    At this point an orbital speed prediction is calculated based on the current altitude. If ↩
    GPS is available at orbit, it will be replaced for the real one. Next mission stage when ↩
    spin is achieved and apogee is reached or passed then second stage ignition is activated ↩
    before
#define MISSION_STAGE2     7 // IMU integrators are active while the orbital speed is achieved. ↩
    Next mission state when second stage burnout is detected. If ignition fails, next mission ↩
    stage will be STANDBY
#define MISSION_DEPLOY     8 // IMU integrators are active while satellites are deployed. Next ↩
    mission state when attitude control is achieved. If attitude control fails, next mission ↩
    stage will be STANDBY
#define MISSION_DAMPING    9 // IMU integrators are active while a damping maneuver is done by ↩
    Magnetorquers. Next mission state when satellite attitude is stabilized
#define MISSION_STANDBY   10 // IMU integrators are active but all the functions are hibernated.↩
     Every 10 seconds it looks for any near ground station below 1200 km. Next mission state ↩
    when any ground station is near
#define MISSION_FOLLOWING 11 // IMU integrators are active while the attitude maneuver is done ↩
    to point towards the ground station. Other near stations are checked in between. Next ↩
    mission state when the target is locked. If ground station is out of range, next mission ↩
    stage will be STANDBY
#define MISSION_PAIRING   12 // Same as FOLLOWING but radio-link is tested. Next mission state ↩
    when link is established and new commands are uploaded. If radio-links fails keeps the ↩
    current mode. If a new nearest ground station is available, next mission stage will be ↩
    FOLLOWING. If ground station is out of range, next mission stage will be STANDBY
#define MISSION_DOWNLOAD  13 // Same as FOLLOWING but download is done. If radio-links fails of ↩
    a new nearest ground station is available, next mission stage will be FOLLOWING. If ground ↩
    station is out of range, next mission stage will be STANDBY
/* Sensors */
#define CompwID B00110010 // Compass LSM303DLHC (0x32)
#define CompwW 0x01 // Write Compass ports
#define CompID B00110001 // Compass LSM303DLHC reading (0x33)
#define CompX 0x04, 0x03
```

```
#define CompY 0x06, 0x05
#define CompZ 0x08, 0x07
/*
#define AccID B00110001 // Accelerometer LSM303DLHC reading (0x33)
#define AccX 0x29, 0x28
#define AccY 0x2B, 0x2A
#define AccZ 0x2D, 0x2C
*/
#define AccID B00011001 // Accelerometer LIS331HH (0x19) TO BE REPLACED BY LSM303DLHC
#define AccX 0x29, 0x28
#define AccY 0x2B, 0x2A
#define AccZ 0x2D, 0x2C
#define GyroID B01101001 // Gyro ITG3200 (0x69)
#define GyroT 0x1B, 0x1C
#define GyroX 0x1D, 0x1E
#define GyroY 0x1F, 0x20
#define GyroZ 0x21, 0x22
#define CamID B01111001 // HD camera write on TCM8230MD (0x79)
#define CamrID B01111000 // HD camera read on TCM8230MD (0x78)
#define CamReg02 0x02 // FPS ACF 0 0 0 0 DCLKP ACFDET
#define IOeID B01000000 // IO expander write on TCA6408A (0x40)
#define IOeW 0x01 // Write IOe ports
#define IOrID B01000001 // IO expander read on TCA6408A (0x41)
#define IOrW 0x00 // Read IOe configuration
#define IOeEN_VRAMP B00000001 // P0
#define IOeEN_3_3V_1 B00000010 // P1
#define IOeEN_2_6V B00000100 // P2
#define IOeEN_1_5V B00001000 // P3
#define IOeEN_A B00010000 // P4 Magnetorquer A - LEFT UP and Turn +v (Yaw)
#define IOeEN_B B00100000 // P5 Magnetorquer B - RIGHT UP and Turn -v (Yaw)
#define IOeEN_C B01000000 // P6 Magnetorquer C - LEFT DOWN and Turn +u (Pitch)
#define IOeEN_D B10000000 // P7 Magnetorquer D - RIGHT DOWN and Turn -u (Pitch)
#define IOeEN_GPS B00000010 // P1 GPS Power supply. CAUTION: This signal should be implemented
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_A); // Active magnetorquer ↩
    P4 A - LEFT UP and Turn +v (Yaw)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_B); // Active magnetorquer ↩
    P5 B - RIGHT UP and Turn -v (Yaw)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_C); // Active magnetorquer ↩
    P6 C - LEFT DOWN and Turn +u (Pitch)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_D); // Active magnetorquer ↩
    P7 D - RIGHT DOWN and Turn -u (Pitch)
#define I2C_BUFFER_LENGTH 32
#define TWI_BUFFER_LENGTH 32 // CONSIDER TO USE I2C_BUFFER_LENGTH
#define TWI_READY 0
#define TWI_MRX   1
//#define TWI_SRX  3
#define TWI_MTX   2
#define TWI_STX   4
#define TW_WRITE  0
#define TW_READ   1
#define TW_MT_SLA_NACK  0x20
#define TW_MT_DATA_NACK 0x30
#define I2C_cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
#define I2C_sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
#define I2C_CPU_FREQ 16000000L
#define I2C_TWI_FREQ 100000L
#define I2C_TWI_READY 0
#define URX_ADDR   (byte *)"AWIP0"
#define UTX_ADDR   (byte *)"AWsrv"
#define RF_CHANNEL 75
#define PL_SIZE    24

//GLOBALS
typedef struct tagIMUdata
{
  float i, j, k; // Inertial acceleration [g]/32768*6
  float u, v, w; // Rate of turn. Angular speed [Â?/s]/32768*2000
  float t; // IMU temperature [Â?C]/100
  int d; // Delta time in microseconds [ms]
  float x, y, z; // Position [m]
  float a, b, c; // Compass angular position. Attitude [Â?] WARNING: Compass is not present
  float l, m, n; // Linear speed [m/s]
  float o, p, q; // Target attitude unit vector
}IMUdata;
IMUdata data;

typedef struct tagIMUbias
{
  int i, j, k; // Inertial bias acceleration [g]/32768*6
  int u, v, w; // Bias rate of turn. Bias angular speed [Â?/s]/32768*2000
}IMUbias;
IMUbias bias;

typedef struct tagSUNtracker
{
  float x, y, z; // sun direction from −1.0 to 1.0
}SUNtracker;
SUNtracker sun;

float dt;                           // Current loop time interval in microseconds
float altitude;                     // Current altitude in meters
float speedv;                       // Current vertical speed for apogee detection in meters per↩
      second
float controlP, controlI, controlD; // PID controller parameters
float targetU, targetV, targetW;    // PID controller target values
float outputU, outputV, outputW;    // PID controller output values
float accU, accV, accW;             // PID controller accumulated values
float lastU, lastV, lastW;          // PID controller last values
unsigned long time;                 // Last time stamp in microseconds
unsigned long timeStamp;            // Last time stamp in microseconds
unsigned long timeMagnetorquers;    // Last magnetorquers time stamp in microseconds
unsigned long timeGPS;              // Last GPS reading cycle time stamp in microseconds
boolean MagnetorquersCycle;         // True = This is a magnetorquers cycle
int r;                              // Integral pointer
long i, j, k, u, v, w;              // Align temporary variables
boolean maxg;                       // True = two consecutive values of maximum acceleration ↩
     read
boolean attitude;                   // True = attitude control is active
boolean gpsActive;                  // True = GPS reading is allowed
boolean payloadActive;              // True = Payload is Turned On
float g;                            // Gravity at +/− 6g
float gz;                           // Gravity at +/− 6g in Z axis
float h;                            // Angular at +/− 2000Â?/s
uint8_t nRFstatus;                  // AWIP_nRF24L01 status
byte mission;                       // Mission status
byte IOeWstatus;                    // IO expander status
// PASAR A ARRIBA
static uint8_t i2c_rxBufferIndex;
static uint8_t i2c_rxBufferLength;
static uint8_t i2c_txBufferIndex;
static uint8_t i2c_txBufferLength;
static uint8_t i2c_txAddress;
uint8_t i2c_rxBuffer[I2C_BUFFER_LENGTH];
uint8_t i2c_txBuffer[I2C_BUFFER_LENGTH];
uint8_t i2c_transmitting = 0;
static volatile uint8_t i2c_state;
static volatile uint8_t i2c_error;
static uint8_t i2c_twi_slarw;
static volatile uint8_t i2c_twi_txBufferLength; // CONSIDER TO USE i2c_txBufferLength
static uint8_t i2c_twi_txBuffer[TWI_BUFFER_LENGTH]; // CONSIDER TO USE i2c_txBuffer
static volatile uint8_t i2c_twi_masterBufferIndex; // CONSIDER TO USE i2c_rxBufferIndex
static uint8_t i2c_twi_masterBufferLength; // CONSIDER TO USE i2c_rxBufferLength
static uint8_t i2c_twi_masterBuffer[TWI_BUFFER_LENGTH];  // CONSIDER TO USE i2c_rxBuffer
long gps_baudRate;                  // GPS baud rate
int gps_bitPeriod;                  // GPS bit period in microseconds
byte gps_chk;                       // NMEA sentence checksum
boolean gps_ischk;                  // Checksum is active
byte gps_p[16];                     // Read buffer for one parameter from GPS input
byte gps_k[80];                     // Read buffer for one NMEA sentence from GPS input
byte gps_n;                         // Read buffer pointer
float gps_lat;                      // New gps latitude (WGS84)
float gps_lon;                      // New gps longitude (WGS84)
float gps_alt;                      // New gps altitude (WGS84)
boolean gps_valid;                  // True = GPS fix available and message validated
float mag_dec;                      // New magnetic vector declination (WMM2010)
//float mag_inc;                      // New magnetic vector inclination (WMM2010)
```

```c
// ---------------------------------------------------------------------------
// ---------------------------------I2C Bus-----------------------------------
// ---------------------------------------------------------------------------
void I2C_beginTransmission(uint8_t address)
{
  // indicate that we are transmitting
  i2c_transmitting = 1;
  // set address of targeted slave
  i2c_txAddress = address;
  // reset tx buffer iterator vars
  i2c_txBufferIndex = 0;
  i2c_txBufferLength = 0;
}
uint8_t I2C_endTransmission(void)
{
  // transmit buffer (blocking)
  int8_t ret = I2C_twi_writeTo(i2c_txAddress, i2c_txBuffer, i2c_txBufferLength, 1);
  // reset tx buffer iterator vars
  i2c_txBufferIndex = 0;
  i2c_txBufferLength = 0;
  // indicate that we are done transmitting
  i2c_transmitting = 0;
  return ret;
}

// must be called in:
// slave tx event callback
// or after beginTransmission(address)
void I2C_send(uint8_t data)
{
  if(i2c_transmitting)
  {
  // in master transmitter mode
    // don't bother if buffer is full
    if(i2c_txBufferLength >= I2C_BUFFER_LENGTH) return;
    // put byte in tx buffer
    i2c_txBuffer[i2c_txBufferIndex] = data;
    ++i2c_txBufferIndex;
    // update amount in buffer
    i2c_txBufferLength = i2c_txBufferIndex;
  }
  else
  {
  // in slave send mode
    // reply to master
    I2C_twi_transmit(&data, 1);
  }
}
uint8_t I2C_requestFrom(uint8_t address, uint8_t quantity) // CONSIDER TO REPLACE BY uint8_t ↩
    twi_readFrom(uint8_t address, uint8_t* data, uint8_t length)
{
  // clamp to buffer length
  if(quantity > I2C_BUFFER_LENGTH) quantity = I2C_BUFFER_LENGTH;
  // perform blocking read into buffer
  uint8_t i2c_read = I2C_twi_readFrom(address, i2c_rxBuffer, quantity);
  // set rx buffer iterator vars
  i2c_rxBufferIndex = 0;
  i2c_rxBufferLength = i2c_read;
  return i2c_read;
}
// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
uint8_t I2C_available(void)
{
  return i2c_rxBufferLength - i2c_rxBufferIndex;
}

// must be called in:
// slave rx event callback
// or after requestFrom(address, numBytes)
uint8_t I2C_receive(void)
{
  // default to returning null char
  // for people using with char strings
  uint8_t value = '\0';
  // get each successive byte on each call
  if(i2c_rxBufferIndex < i2c_rxBufferLength)
  {
    value = i2c_rxBuffer[i2c_rxBufferIndex];
    ++i2c_rxBufferIndex;
  }
  return value;
}

/*
 * Function twi_transmit
 * Desc     fills slave tx buffer with data
 *          must be called in slave tx event callback
 * Input    data: pointer to byte array
 *          length: number of bytes in array
 * Output   1 length too long for buffer
 *          2 not slave transmitter
 *          0 ok
 */
uint8_t I2C_twi_transmit(uint8_t* data, uint8_t length)
{
  uint8_t i;
  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length) return 1;
  // ensure we are currently a slave transmitter
  if(TWI_STX != i2c_state) return 2;
  // set length and copy data into tx buffer
  i2c_twi_txBufferLength = length;
  for(i = 0; i < length; ++i) i2c_twi_txBuffer[i] = data[i];
  return 0;
}

/*
 * Function twi_readFrom
 * Desc     attempts to become twi bus master and read a
 *          series of bytes from a device on the bus
 * Input    address: 7bit i2c device address
 *          data: pointer to byte array
 *          length: number of bytes to read into array
 * Output   number of bytes read
 */
uint8_t I2C_twi_readFrom(uint8_t address, uint8_t* data, uint8_t length)
{
  uint8_t i;
  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length) return 0;
  // wait until twi is ready, become master receiver
  while(TWI_READY != i2c_state);
  i2c_state = TWI_MRX;
  // reset error state (0xFF.. no error occured)
  i2c_error = 0xFF;
  // initialize buffer iteration vars
  i2c_twi_masterBufferIndex = 0;
  i2c_twi_masterBufferLength = length - 1;  // This is not intuitive, read on...
  // On receive, the previously configured ACK/NACK setting is transmitted in
  // response to the received byte before the interrupt is signalled.
  // Therefor we must actually set NACK when the _next_ to last byte is
  // received, causing that NACK to be sent in response to receiving the last
  // expected byte of data.

  // build sla+w, slave device address + w bit
  i2c_twi_slarw = TW_READ;
  i2c_twi_slarw |= address << 1;
  // send start condition
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);
  // wait for read operation to complete
  while(TWI_MRX == i2c_state);
  if (i2c_twi_masterBufferIndex < length) length = i2c_twi_masterBufferIndex;
  // copy twi buffer to data
  for(i = 0; i < length; ++i) data[i] = i2c_twi_masterBuffer[i];
  return length;
}

/*
 * Function twi_writeTo
 * Desc     attempts to become twi bus master and write a
 *          series of bytes to a device on the bus
 * Input    address: 7bit i2c device address
 *          data: pointer to byte array
 *          length: number of bytes in array
 *          wait: boolean indicating to wait for write or not
 * Output   0 .. success
 *          1 .. length to long for buffer
 *          2 .. address send, NACK received
 *          3 .. data send, NACK received
 *          4 .. other twi error (lost bus arbitration , bus error, ..)
 */
uint8_t I2C_twi_writeTo(uint8_t address, uint8_t* data, uint8_t length, uint8_t wait)
{
  uint8_t i;
  // ensure data will fit into buffer
  if(TWI_BUFFER_LENGTH < length) return 1;
  // wait until twi is ready, become master transmitter
  while(TWI_READY != i2c_state);
  i2c_state = TWI_MTX;
  // reset error state (0xFF.. no error occured)
  i2c_error = 0xFF;
  // initialize buffer iteration vars
  i2c_twi_masterBufferIndex = 0;
  i2c_twi_masterBufferLength = length;
  // copy data to twi buffer
  for(i = 0; i < length; ++i) i2c_twi_masterBuffer[i] = data[i];
  // build sla+w, slave device address + w bit
  i2c_twi_slarw = TW_WRITE;
  i2c_twi_slarw |= address << 1;
  // send start condition
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);
  // wait for write operation to complete
  while(wait && (TWI_MTX == i2c_state));
  if (i2c_error == 0xFF) return 0; // success
  else if (i2c_error == TW_MT_SLA_NACK) return 2; // error: address send, nack received
  else if (i2c_error == TW_MT_DATA_NACK) return 3; // error: data send, nack received
  else
    return 4;   // other twi error
}

void I2C_SetRegister(int device, int address, int value)
{
 I2C_beginTransmission(device);
 I2C_send(address);
 I2C_send(value);
 I2C_endTransmission();
}

byte I2C_GetRegister(int device, int address)
{
 I2C_beginTransmission(device);
 I2C_send(address);
 I2C_endTransmission();
 I2C_requestFrom(device, 1);
 if(!I2C_available()) return 0;
 return I2C_receive();
}

int I2C_GetValue(int device, int addressH, int addressL)
{
 return ((unsigned int)(I2C_GetRegister(device, addressH)) << 8) + I2C_GetRegister(device, ↩
     addressL);
}
```

```
void I2C_TurnOn(byte mask)
{
  IOeWstatus = I2C_GetRegister(IOrID, IOrW) | mask;
  EEPROMWrite(IOeW_ADDR, 1, &IOeWstatus);
  I2C_SetRegister(IOeID, IOeW,  IOeWstatus);
}

void I2C_TurnOff(byte mask)
{
  IOeWstatus = I2C_GetRegister(IOrID, IOrW) &~ mask;
  EEPROMWrite(IOeW_ADDR, 1, &IOeWstatus);
  I2C_SetRegister(IOeID, IOeW, IOeWstatus);
}
// BORRAR
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_A); // Active magnetorquer ↩
    P4 A - LEFT UP and Turn +v (Yaw)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_B); // Active magnetorquer ↩
    P5 B - RIGHT UP and Turn -v (Yaw)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_C); // Active magnetorquer ↩
    P6 C - LEFT DOWN and Turn +u (Pitch)
// I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_D); // Active magnetorquer ↩
    P7 D - RIGHT DOWN and Turn -u (Pitch)
void I2C_UpdateAxis(float output_axis, byte positive_mask, byte negative_mask)
{
    if (output_axis > 0.1)
    {
      I2C_TurnOff(negative_mask);
      I2C_TurnOn(positive_mask);
    }
    else if (output_axis < 0.1)
    {
      I2C_TurnOff(positive_mask);
      I2C_TurnOn(negative_mask);
    }
    else
    {
      I2C_TurnOff(positive_mask);
      I2C_TurnOff(negative_mask);
    }
}



// ---------------------------------------------------------------------------
// ---------------------------SPI bus------------------------------------------
// ---------------------------------------------------------------------------
unsigned int SPI_readRegister(byte thisRegister)
{
  unsigned int result = 0;   // result to return
  //digitalWrite(chipSelectPin, LOW);
  SPI.transfer(thisRegister & B11111100);
  result = SPI.transfer(0x00);
  //digitalWrite(chipSelectPin, HIGH);
  return(result);
}

// Writes a value in SPI bus register
void SPI_writeRegister(byte thisRegister, byte thisValue)
{
  //digitalWrite(chipSelectPin, LOW);
  SPI.transfer(thisRegister | B00000010); //Send register location
  SPI.transfer(thisValue);  //Send value to record into register
  //digitalWrite(chipSelectPin, HIGH);
}

// Turn On a Control vector nozzle
void SPI_TurnOn(byte mask)
{
  // WARNING: Control vector configuration should be defined. I2C used instead SPI. Need to be ↩
      modified
  I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | mask);
}
```

```
// Turn Off a Control vector nozzle
void SPI_TurnOff(byte mask)
{
  // WARNING: Control vector configuration should be defined. I2C used instead SPI. Need to be ↩
      modified
  I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) &~ mask);
}

// Set a control vector configuration
void SPI_Config(byte maskOn1, byte maskOn2, byte maskOff3, byte maskOff4)
{
  SPI_TurnOff(maskOff3);
  SPI_TurnOff(maskOff4);
  SPI_TurnOn(maskOn1);
  SPI_TurnOn(maskOn2);
}



// ----------------------------------------------------------------------------
// ---------------------------EEPROM-------------------------------------------
// ----------------------------------------------------------------------------

// Returns 1 if EEPROM is ready for a new read/write operation, 0 if not.
#define eeprom_is_ready() bit_is_clear(EECR, EEPE)

//Read one byte from EEPROM address __p.
#define EEPROM_read(address) ((uint8_t) eeprom_read_byte((const uint8_t *) address))
__ATTR_PURE__ static __inline__ uint8_t eeprom_read_byte (const uint8_t *__p)
{
    while (!eeprom_is_ready());
    #if E2END <= 0xFF
      EEARL = (uint8_t)__p;
    #else
      EEAR = (uint16_t)__p;
    #endif
    /* Use inline assembly below as some AVRs have problems with accessing
    EECR with STS instructions. For example, see errata for ATmega64.
    The code below also assumes that EECR and EEDR are in the I/O space.
    */
    uint8_t __result;
    __asm__ __volatile__
    (
      "/* START EEPROM READ CRITICAL SECTION */ \n\t"
      "sbi %1, %2 \n\t"
      "in %0, %3 \n\t"
      "/* END EEPROM READ CRITICAL SECTION */ \n\t"
      : "=r" (__result)
      : "i" (_SFR_IO_ADDR(EECR)),
        "i" (EERE),
        "i" (_SFR_IO_ADDR(EEDR))
    );
    return __result;
}

// Write a byte __value to EEPROM address __p.
#define EEPROM_write(address, value) (eeprom_write_byte((uint8_t *)address, (uint8_t)value))
static __inline__ void eeprom_write_byte (uint8_t *__p, uint8_t __value)
{
  while (!eeprom_is_ready ());

  EECR = 0;     /* Set programming mode: erase and write.   */

  #if E2END <= 0xFF
    EEARL = (unsigned)__p;
  #else
    EEAR = (unsigned)__p;
  #endif
  EEDR = __value;

  __asm__ __volatile__
  (
    "/* START EEPROM WRITE CRITICAL SECTION */\n\t"
    "in r0, %[__sreg]       \n\t"
    "cli                \n\t"
    "sbi    %[__eecr], %[__eempe]   \n\t"
    "sbi    %[__eecr], %[__eepe]    \n\t"
    "out    %[__sreg], r0       \n\t"
    "/* END EEPROM WRITE CRITICAL SECTION */"
    :
    : [__eecr]  "i" (_SFR_IO_ADDR(EECR)),
      [__sreg]  "i" (_SFR_IO_ADDR(SREG)),
      [__eempe] "i" (EEMPE),
      [__eepe]  "i" (EEPE)
    : "r0"
  );
}

// Write a memory block in the EEPROM
#define EEPROMWriteVar(addr, var) EEPROMWrite(addr, sizeof(var), (byte*)&(var))
void EEPROMWrite(int addr, uint8_t length, byte *block)
{
  while(length--)
  {
    EEPROM_write((const uint8_t*)addr, *(block++));
    addr++;
  }
}

// Read a memory block from the EEPROM
#define EEPROMReadVar(addr, var) EEPROMRead(addr, sizeof(var), (byte*)&(var))
void EEPROMRead(int addr, uint8_t length, byte *block)
{
  while(length--)
  {
    *(block++) = EEPROM_read(addr);
    addr++;
  }
}

void EEPROMSetFlag(int addr, byte mask)
{
  uint8_t c = EEPROM_read(addr) | mask;
  EEPROM_write(addr, c);
}

void EEPROMResetFlag(int addr, byte mask)
{
  uint8_t c = EEPROM_read(addr) &~ mask;
  EEPROM_write(addr, c);
}

boolean EEPROMGetFlag(int addr, byte mask)
{
  return EEPROM_read(addr) & mask;
}

/* // NOT IN USE. KEEPED FOR COMPATIBILITY
// Reads a float variable from the specified EEPROM address
float EEPROMReadFloat(int addr)
{
  float num;
  ((byte*)&num)[0] = EEPROM_read(addr++);
  ((byte*)&num)[1] = EEPROM_read(addr++);
  ((byte*)&num)[2] = EEPROM_read(addr++);
  ((byte*)&num)[3] = EEPROM_read(addr++);
  return num;
}

// Writes a float variable to the specified EEPROM address
void EEPROMWriteFloat(int addr, float num)
{
  EEPROM_write(addr++, ((byte*)&num)[0]);
  EEPROM_write(addr++, ((byte*)&num)[1]);
  EEPROM_write(addr++, ((byte*)&num)[2]);
  EEPROM_write(addr++, ((byte*)&num)[3]);
```

```
}
*/
// ---------------------------------------------------------------------------------
// ------------------------------nRF24L01p radio------------------------------------
// ---------------------------------------------------------------------------------
uint8_t nRF_softSpiSend(uint8_t data)
{
  uint8_t buff = 0;
  for (int i = 7; i>=0; i--)
  {
    digitalWrite(SOFT_MOSI, (data>>i&1));
    digitalWrite(SOFT_SCK,HIGH);
    buff = buff | ((digitalRead(SOFT_MISO)&1)<<i);
    digitalWrite(SOFT_SCK,LOW);
  }
  return buff;
}

void nRF_transmitSync(uint8_t *dataout, uint8_t len)
{
  uint8_t i;
  for(i = 0;i < len;i++)
  {
    nRF_softSpiSend(dataout[i]);
  }
}

void nRF_transferSync(uint8_t *dataout, uint8_t *datain, uint8_t len)
{
  uint8_t i;
  for(i = 0;i < len;i++)
  {
    datain[i] = nRF_softSpiSend(dataout[i]);
  }
}

uint8_t nRF_setCmd(uint8_t reg)
{
  csnLow();
  nRFstatus = nRF_softSpiSend(reg);
  csnHi();
  return nRFstatus;
}

uint8_t nRF_setData(uint8_t *data, uint8_t len)
{
  csnLow();
  nRFstatus = nRF_softSpiSend(W_TX_PAYLOAD);
  nRF_transmitSync(data, len);
  csnHi();
}

uint8_t nRF_getData(uint8_t *data, uint8_t len)
{
  csnLow();
  nRFstatus = nRF_softSpiSend(R_RX_PAYLOAD);
  nRF_transferSync(data, data, len);
  csnHi();
  nRF_setReg(STATUS, (1<<RX_DR));
}

uint8_t nRF_setReg(uint8_t reg, uint8_t value)
{
  csnLow();
  nRFstatus = nRF_softSpiSend(W_REGISTER | (REGISTER_MASK & reg));
  nRF_softSpiSend(value);
  csnHi();
  return nRFstatus;
}

uint8_t nRF_setReg(uint8_t reg, uint8_t *value, uint8_t len)
{
  uint8_t stat = 0;
  csnLow();
  nRFstatus = nRF_softSpiSend(W_REGISTER | (REGISTER_MASK & reg));
  nRF_transmitSync(value, len);
  csnHi();
  return nRFstatus;
}

uint8_t nRF_getReg(uint8_t reg)
{
  uint8_t stat = 0;
  csnLow();
  nRFstatus = nRF_softSpiSend(R_REGISTER | (REGISTER_MASK & reg));
  stat = nRF_softSpiSend(reg);
  csnHi();
  return stat;
}

uint8_t nRF_getReg(uint8_t reg, uint8_t *value, uint8_t len)
{
  uint8_t stat = 0;
  csnLow();
  nRFstatus = nRF_softSpiSend(R_REGISTER | (REGISTER_MASK & reg));
  nRF_transferSync(value,value,len);
  csnHi();
  return nRFstatus;
}

void nRF_configure(uint8_t * myaddr, uint8_t * addr, uint8_t channel, uint8_t pl_size)
{
  ceLow();
  nRF_setReg(RX_ADDR_P0, addr, 5);
  nRF_setReg(RX_ADDR_P1, myaddr, 5);
  nRF_setReg(TX_ADDR, addr, 5);
  nRF_setReg(EN_AA, 0x03); //auto_ack enabled for both rx pipes 0 and 1
  nRF_setReg(EN_RXADDR, 0x03); //both pipes P0 and P1 are enabled to recieve
  nRF_setReg(RX_PW_P0, pl_size);
  nRF_setReg(RX_PW_P1, pl_size);
  nRF_setReg(SETUP_RETR, 0x1A);
  nRF_setReg(RF_CH, channel);
  nRF_setReg(RF_SETUP, 0x07);
}

void nRF_stateStandbyI()
{
  ceLow();
  nRF_setReg(CONFIG, 0x0E);
  delayMicroseconds(130);
}

bool nRF_dataReady()
{
  byte tmp = nRF_getReg(STATUS);
  if( nRFstatus & (1 << RX_DR) ) return 1;
  byte fifoStatus = nRF_getReg(FIFO_STATUS);
  return !(fifoStatus & (1 << RX_EMPTY));
}

boolean nRF_sendd(byte* data, byte psize)
{
  //byte stat = nrf.getReg(STATUS);
  //if ((stat & ((1 << TX_DS)  | (1 << MAX_RT)))) return false;

  ceLow();
  nRF_setReg(CONFIG, 0x0e);
  nRF_setCmd(FLUSH_TX);
  nRF_setData(data, psize);
  ceHi();
  delayMicroseconds(10);
  ceLow();
}
```

```
// ---------------------------------------------------------------------------
// ------------------------------------GPS------------------------------------
// ---------------------------------------------------------------------------
// Reads a GPS character
int GPS_read()
{
  unsigned long timeout = 10000000; // WARNING: This value should be validated
  int val = 0;
  int bitDelay = gps_bitPeriod - clockCyclesToMicroseconds(50);
  // one byte of serial data (LSB first)
  // ...--\    /--\/--\/--\/--\/--\/--\/--\/--\/--...
  //      \--/\--/\--/\--/\--/\--/\--/\--/\--/
  //     start  0   1   2   3   4   5   6   7 stop
  while (timeout--)
  {
    if(digitalRead(GPSRX_PIN))
    {
      // confirm that this is a real start bit, not line noise
      if (digitalRead(GPSRX_PIN) == LOW)
      {
        // frame start indicated by a falling edge and low start bit
        // jump to the middle of the low start bit
        delayMicroseconds(bitDelay / 2 - clockCyclesToMicroseconds(50));
        // offset of the bit in the byte: from 0 (LSB) to 7 (MSB)
        for (int offset = 0; offset < 8; offset++)
        {
            // jump to middle of next bit
            delayMicroseconds(bitDelay);
            // read bit
            val |= digitalRead(GPSRX_PIN) << offset;
        }
        delayMicroseconds(gps_bitPeriod);
        return val;
      }
      return -1; // Synchronization lost
    }
  }
  return -2; // Timeout
}

// Reads a character from GPS
byte GPS_Next()
{
  // GPS reading
  byte val = (byte)GPS_read();
  if(val == '$') // Check for a command prefix
  {
    gps_ischk = true; // Activate checksum flag
    gps_chk = 0;      // Reset checksum
    gps_n = 0;        // Reset command buffer pointer
  }
  else if(gps_ischk)
  {
    if(val == '*') gps_ischk = false; else gps_chk ^= val; // Calculate checksum
  }
  if(gps_n < sizeof(gps_k)) gps_k[gps_n++] = val; // Store the byte in the command buffer
  return val;
}

// Reads a parameter from the serial port
int GPS_ReadParam(byte chr = ',')
{
  int i = 0;
  while(true)
  {
    if(i >= sizeof(gps_p))
    {
      return -1; // Buffer overrun
    }
    else if((gps_p[i] = GPS_Next()) == chr)
    {
      return i; // Number of bytes read
    }
    else
    {
      i++;
    }
  }
}

// Returns the value of two byte ASCII hexadecimal number
byte GPS_Check(char *k)
{
  int i;
  byte c;
  // 0xDF = ~0x20 .. 55 = 'A' - 10
  c = k[0]; if(c >= 'A') c = (c & 0xDF) - 55; else c -= '0';
  i = c << 4;
  c = k[1]; if(c >= 'A') c = (c & 0xDF) - 55; else c -= '0';
  i += c;
  return i;
}

// Deactivate GPS updating
void GPS_activate(boolean active)
{
  if(active) EEPROMSetFlag(FLAGS_ADDR, GPS_FLAG); else EEPROMResetFlag(FLAGS_ADDR, GPS_FLAG);
  gpsActive = active;
}



// -----------------------------------------------------------------------------
// ------------------------------Misc functions---------------------------------
// -----------------------------------------------------------------------------

// Converts Cartesian XYZ (in meters) to LLA coordinates (in radians and meters) using a ↩
    spherical model
void XYZ2LLA(float x, float y, float z, float *lat, float *lon, float *alt)
{
  float r = sqrt(x * x + y * y);
  if (r != 0)
  {
    *alt = sqrt(r * r + z * z) - PLANET_RADIUS;
    *lat = atan(z / r);
  }
  else
  {
    *lon = 0;
    if (z < 0)
    {
      *alt = -z - PLANET_RADIUS;
      *lat = -A90;
    }
    else
    {
      *lat = A90;
      *alt = z - PLANET_RADIUS;
    }
  }
}

// Converts LLA coordinates (in radians and meters) to cartesian XYZ (in meters) using a ↩
    spherical model
void LLA2XYZ(float lat, float lon, float alt, float *x, float *y, float *z)
{
  float r = PLANET_RADIUS + alt;
  *z = r * sin(lat);
  r = r * cos(lat);
  *x = -r * sin(lon);
  *y = r * cos(lon);
}

// Calculate the distance (in meters) between two LLA coordinates (in radians and meters) using ↩
    a spherical model
float LLADistance(float lat1, float lon1, float alt1, float lat2, float lon2, float alt2)
```

```
{
  float x1, y1, z1, x2, y2, z2;
  LLA2XYZ(lat1, lon1, alt1, &x1, &y1, &z1);
  LLA2XYZ(lat2, lon2, alt2, &x2, &y2, &z2);
  x1 -= x2;
  y1 -= y2;
  z1 -= z2;
  return sqrt(x1 * x1 + y1 * y1 + z1 * z1);
}

//Converts a string in degrees-minutes format to degrees only
float ToDM(char *str, byte degrees_size)
{
  float n = atof(str + degrees_size) / 60;
  *(str + degrees_size) = 0;
  n += atof(str);
  return n;
}

// Checks if any station is in range
byte StationInRange()
{
  float lat1, lon1, alt1;
  float lat2, lon2;
  float dist = INFINITE;
  byte n = 0;
  int a;
  EEPROMReadVar(LAT_ADDR, lat1);
  EEPROMReadVar(LON_ADDR, lon1);
  EEPROMReadVar(ALT_ADDR, alt1);
  for(byte c = 0; c < 8; c++)
  {
    a = WAYPOINTLIST_ADDR + (c << 4);
    EEPROMReadVar(a, lat2);
    EEPROMReadVar(a + 4, lon2);
    r = LLADistance(lat1, lon1, alt1, lat2, lon2, 0);
    if (r < dist)
    {
      dist = r;
      n = c;
    }
  }
  if(dist > COVERAGE_DISTANCE) n = 0; else n++;
  EEPROMWriteVar(LOCKED_ADDR, n);
  return n;
}

// Closes the control loop activating the magnetorquers
void MagnetorquersUpdate()
{
  // WARNING: Magnetorquers configuration should be defined
  if(MagnetorquersCycle)
  {
    I2C_UpdateAxis(outputU, IOeEN_A, IOeEN_B);
    I2C_UpdateAxis(outputV, IOeEN_C, IOeEN_D);
    //I2C_UpdateAxis(outputW, IOeEN_E, IOeEN_F); // This magnetorquer is not available
  }
  // BORRAR
  // I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_A); // Active ↩
      magnetorquer P4 A - LEFT UP and Turn +v (Yaw)
  // I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_B); // Active ↩
      magnetorquer P5 B - RIGHT UP and Turn -v (Yaw)
  // I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_C); // Active ↩
      magnetorquer P6 C - LEFT DOWN and Turn +u (Pitch)
  // I2C_SetRegister(IOeID, IOeW, I2C_GetRegister(IOrID, IOrW) | IOeEN_D); // Active ↩
      magnetorquer P7 D - RIGHT DOWN and Turn -u (Pitch)
}

// Launch trajectory calculation
void ControlVectorTrajectory()
{
  // Fixed pitch angle mode
  // attitude = false;
  // targetU = TARGET_ANGLE;

  // Pitch target angle calculation as a function of the current altitude and three defined ↩
      points
  if(altitude >= ALT3)
    targetU = ANGLE3;
  else if(altitude >= ALT2)
    targetU = ANGLE2 + (ANGLE3 − ANGLE2) * (altitude − ALT2) / (ALT3 − ALT2);
  else if(altitude >= ALT1)
    targetU = ANGLE1 + (ANGLE2 − ANGLE1) * (altitude − ALT1) / (ALT2 − ALT1);
  else
    targetU = ANGLE1;
}
// Closes the control vector loop activating the control vector nozzles
void ControlVectorUpdate()
{
  // WARNING: Control Vector configuration should be defined
  boolean a, b, c, d;
  if(MagnetorquersCycle)
  {
    // PENDING: It should be a SPI command, not a I2C command
    if (outputW > 0.1)
    {
      SPI_Config(IOeEN_B, IOeEN_C, IOeEN_A, IOeEN_D);
    }
    else if (outputW < 0.1)
    {
      SPI_Config(IOeEN_A, IOeEN_D, IOeEN_B, IOeEN_C);
    }
    else
    {
      a = false;
      b = false;
      c = false;
      d = false;
      if (outputU > 0.1)
      {
        b = true;
        d = true;
      }
      else if (outputU < 0.1)
      {
        a = true;
        c = true;
      }
      if (outputV > 0.1)
      {
        a = true;
        b = true;
      }
      else if (outputV < 0.1)
      {
        c = true;
        d = true;
      }
      if(a)SPI_TurnOn(IOeEN_A); else SPI_TurnOff(IOeEN_A);
      if(b)SPI_TurnOn(IOeEN_B); else SPI_TurnOff(IOeEN_B);
      if(c)SPI_TurnOn(IOeEN_C); else SPI_TurnOff(IOeEN_C);
      if(d)SPI_TurnOn(IOeEN_D); else SPI_TurnOff(IOeEN_D);
    }
  }
}

// Calculates the PID algorithm for the given values
float CalculatePID(float value, float target, float *acc, float *last)
{
  float h = target − value;         // Proportional
  *acc += h * dt;                   // Integral
  float g = (h − *last) / dt;       // Derivative
  g = h * controlP + *acc * controlI + g * controlD;
  *last = h;
  return g;
}
```

```
// Activate attitude control and configure PID controller parameters
void AttitudeOn(float p, float i, float d)
{
  controlP = p;
  controlI = i;
  controlD = d;
  EEPROMWriteVar(CONTROL_P_ADDR, p);
  EEPROMWriteVar(CONTROL_I_ADDR, i);
  EEPROMWriteVar(CONTROL_D_ADDR, d);
  EEPROMSetFlag(FLAGS_ADDR, ATTITUDE_FLAG);
  attitude = true;
}

// Deactivate attitude control
void AttitudeOff()
{
  EEPROMResetFlag(FLAGS_ADDR, ATTITUDE_FLAG);
  attitude = false;
}

// Set the mission status and turn of payload if it is standby
void SetMission(byte n)
{
  mission = n;
  EEPROMWrite(MISSION_ADDR, 1, &mission);
  if(mission == MISSION_STANDBY) Payload_activate(false);
}

// Activates if selected the payload power supply
void Payload_activate(boolean active)
{
  if(active)
  {
    // Turn on payload power supply
    I2C_TurnOn(IOeEN_2_6V);
    I2C_TurnOn(IOeEN_1_5V);
    I2C_TurnOn(IOeEN_3_3V_1);
    // Turn on the radio
    ceLow();
    csnHi();
    nRF_configure(URX_ADDR, UTX_ADDR, RF_CHANNEL, PL_SIZE);
    nRF_stateStandbyI();
    // Start to record because at this point, target is locked
    I2C_SetRegister(CamID, CamReg02, B10000000); // 15fps, 50Hz, normal DCLKP and ACFDET Auto ↩
         mode
  }
  else
  {
    I2C_TurnOff(IOeEN_3_3V_1);
    I2C_TurnOff(IOeEN_2_6V);
    I2C_TurnOff(IOeEN_1_5V);
  }
  payloadActive = active;
}

// ---------------------------------------------------------------------------
// ------------------------------Reset procedure------------------------------
// ---------------------------------------------------------------------------

#if !RESET_EEPROM
// ---------------------------------------------------------------------------
// ------------------------------Main functions-------------------------------
// ---------------------------------------------------------------------------
void setup()
{
  // Starting I2C bus
  i2c_rxBufferIndex = 0;
  i2c_rxBufferLength = 0;
  i2c_txBufferIndex = 0;
  i2c_txBufferLength = 0;
  // initialize two wire interface state
  i2c_state = TWI_READY;
```

```
// activate internal pull-ups for twi
// as per note from atmega8 manual pg167
I2C_sbi(PORTC, 4);
I2C_sbi(PORTC, 5);

// initialize twi prescaler and bit rate
I2C_cbi(TWSR, TWPS0);
I2C_cbi(TWSR, TWPS1);
TWBR = ((I2C_CPU_FREQ / I2C_TWI_FREQ) - 16) / 2;

/* twi bit rate formula from atmega128 manual pg 204
SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
note: TWBR should be 10 or higher for master mode
It is 72 for a 16mhz Wiring board with 100kHz TWI */

// enable twi module, acks, and twi interrupt
TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);

// Initializing I2C sensors
I2C_SetRegister(AccID, 0x20, B00111111); // Initialize at 1000 Hz sampling rate, low pass cut-↩
    off at 780 Hz. Register 32 (0x20) - CTRL_REG1: PM2 PM1 PM0 DR1 DR2 Zen Yen Xen
I2C_SetRegister(AccID, 0x23, B00000000); // From -6g to +6g. g=5461.3f; Register (0x23) - ↩
    CTRL_REG4: BDU BLE FS1 FS0 STsign 0 ST SIM
I2C_SetRegister(GyroID, 0x3E, B00000001); // Internal oscillator. Register 62 (0x3E) â  Power↩
     Management: H_RESET SLEEP STBY_XG STBY_YG STBY_ZG CLK_SEL_Bit2 CLK_SEL_Bit2 CLK_SEL_Bit1 ↩
    CLK_SEL_Bit0
I2C_SetRegister(GyroID, 0x16, B00011011); // Initialize at 1000 Hz sampling rate, DLPF at 188 ↩
    Hz. Register 22 (0x16) â  Digital Low Pass Filter, Full Scale: FS_SEL_Bit4 FS_SEL_Bit3 ↩
    DLPF_CFG_Bit2 DLPF_CFG_Bit1 DLPF_CFG_Bit0
I2C_SetRegister(IOeID, IOeW, IOeWstatus);

// Starting SPI bus
SPI.begin(); // SPI bus for launcher communications

// Starting GPS
gps_baudRate = 4800;
gps_bitPeriod = 1000000 / gps_baudRate;
digitalWrite(GPSTX_PIN, HIGH);
delayMicroseconds(gps_bitPeriod); // if we were low this establishes the end

// Configuring the radio
pinMode(SOFT_MOSI, OUTPUT);
pinMode(SOFT_MISO, INPUT);
pinMode(SOFT_SCK, OUTPUT);
digitalWrite(SOFT_MOSI,LOW);
digitalWrite(SOFT_SCK,LOW);
pinMode(pCE,OUTPUT);
pinMode(pCSN,OUTPUT);

// Configuring EEPROM and mission state recovery
EEPROMReadVar(MISSION_ADDR, mission);
time = micros(); // Time stamp
g = 1.796265E-03; // g = 6.0 * 9.81 / 32768;
gz = 1.704193E-03; // g = 6.0 * 9.81 * (9.81 / 10.34) / 32768;
h = .06103515625; // h = 2000 / 32768;
maxg = EEPROMGetFlag(FLAGS_ADDR, MAXG_FLAG);
attitude = EEPROMGetFlag(FLAGS_ADDR, ATTITUDE_FLAG);
GPS_activate(EEPROMGetFlag(FLAGS_ADDR, GPS_FLAG));
Payload_activate(EEPROMGetFlag(FLAGS_ADDR, PAYLOAD_FLAG));
EEPROMReadVar(CONTROL_P_ADDR, controlP);
EEPROMReadVar(CONTROL_I_ADDR, controlI);
EEPROMReadVar(CONTROL_D_ADDR, controlD);
//EEPROMReadVar(TARGET_U_ADDR, targetU);
//EEPROMReadVar(TARGET_V_ADDR, targetV);
//EEPROMReadVar(TARGET_W_ADDR, targetW);
targetU = 0;
targetV = 0;
targetW = 0;
accU = 0;
accV = 0;
accW = 0;
lastU = 0;
```

```
  lastV = 0;
  lastW = 0;
  timeMagnetorquers = 0;
}

void loop()
{
  byte c;
  float newlat;
  float newlon;
  float newalt;
  byte newsta;

  if(mission < MISSION_DEPLOY)
  {
    // Acceleration filtering
    i = 0;
    j = 0;
    k = 0;
    r = (1 << INTEGRATOR_CYCLES);
    while(r--)
    {
      i += I2C_GetValue(AccID, AccX);
      j += I2C_GetValue(AccID, AccY);
      k += I2C_GetValue(AccID, AccZ);
      //delay(1);
    }

    // Accelerometer saturation detection
    if(k > (32000 << INTEGRATOR_CYCLES))
    {
      if(maxg)
      {
        EEPROMSetFlag(FLAGS_ADDR, MAXG_FLAG);
      }
      maxg = true;
    }
    else
    {
      maxg = false;
    }
    data.i = (float)(i >> INTEGRATOR_CYCLES) * g;
    data.j = (float)(j >> INTEGRATOR_CYCLES) * g;
    data.k = (float)(k >> INTEGRATOR_CYCLES) * gz;
    // Integrators. v=a*dt; e=v*dt
    data.l += data.i * dt; // Speed
    data.m += data.j * dt;
    data.n += data.k * dt;
    data.x += data.l * dt; // Position
    data.y += data.m * dt;
    data.z += data.n * dt;
  }
  else
  {
    // Sun tracker calculation
    sun.x = (analogRead(SUNA_PIN) - analogRead(SUND_PIN)) / 1024; // CAUTION: Sun tracker should↩
         be defined
    sun.y = (analogRead(SUNA_PIN) - analogRead(SUND_PIN)) / 1024;
    sun.z = (analogRead(SUNE_PIN) - analogRead(SUNF_PIN)) / 1024;
  }

  //Gyros bias correction
  data.u = (I2C_GetValue(GyroID, GyroX) - bias.u) * h;
  data.v = (I2C_GetValue(GyroID, GyroY) - bias.v) * h;
  data.w = (I2C_GetValue(GyroID, GyroZ) - bias.w) * h;
  data.t = (((float)(I2C_GetValue(GyroID, GyroT) + 13200)) / 280.0f) * 100 + 3500; // Celsius=((↩
      T+13200)/280)+35
  data.d = micros() - time; if(data.d < 0) data.d = -data.d;
  time += data.d;
  dt = (float)data.d / 1000000.0; // Seconds

  // Integrator c=w*dt
  data.a += data.u * dt; // Attitude WARNING: TO BE REPLACED BY LSM303DLHC COMPASS
  data.b += data.v * dt;
  data.c += data.w * dt;
  /*
  data.a = I2C_GetValue(CompID, CompX); // Magnetic attitude
  data.b = I2C_GetValue(CompID, CompY);
  data.c = I2C_GetValue(CompID, CompZ);
  */

  altitude = 0; // Altitude calculation
  speedv = 0; // Vertical speed calculation

  // Magnetorquers Cycle detection
  timeMagnetorquers += data.d;
  if(timeMagnetorquers > 20000)
  {
    MagnetorquersCycle = true;
    timeMagnetorquers -= 20000;
  }

  // GPS cycle every second
  timeGPS += data.d;
  if(timeGPS > 1000000)
  {
    timeGPS -= 1000000;
    gps_valid = false;
    if(gpsActive)
    {
      if(GPS_Next() == '$') if(GPS_Next() == 'G') if(GPS_Next() == 'P') if(GPS_Next() == 'G') if↩
          (GPS_Next() == 'G') if(GPS_Next() == 'A') if(GPS_Next() == ',') // Check for $GPGGA ↩
          command
      {
        if(GPS_ReadParam() >= 0)      // Time in UTC of position
        if(GPS_ReadParam() > 2)      // Latitude of position
        {
          newlat = ToDM((char*)&gps_p[0], 2); // Parse the latitude and store it in a temporary ↩
              variable before validation
          if(GPS_ReadParam() > 3)      // Latitude N or S
          {
            newlon = ToDM((char*)&gps_p[0], 3); // Parse the longitude and store it in a ↩
                temporary variable before validation
            if(GPS_ReadParam() >= 0)      // Longitude of position
            if(GPS_ReadParam() >= 0)      // Longitude E or W
            if(GPS_ReadParam() >= 0)      // GPS quality                          (0=fix not ↩
                available, 1=Non-differential GPS fix available, 2=Differential GPS (WAAS) fix ↩
                available, 6=Estimated)
            {
              newsta = gps_p[0]; // Parse GPS alignment
              if(GPS_ReadParam() >= 0)      // Number of satellites                  (00 to 12)
              if(GPS_ReadParam() >= 0)      // Horizontal dilution of precision       (0.5 to ↩
                  99.9)
              if(GPS_ReadParam() > 0)       // Antenna altitude above mean sea level  (-9999.9 ↩
                  to 99999.9)
              {
                newalt = atof((char*)&gps_p[0]); // Parse the altitude and store it in a ↩
                    temporary variable before validation
                if(GPS_ReadParam() >= 0)    // Units of antenna altitude, meters      M
                if(GPS_ReadParam() >= 0)    // Height of geoid above ellipsoid        (-999.9 to↩
                    9999.9)
                if(GPS_ReadParam() >= 0)    // Units of geoid height, meters          M
                if(GPS_ReadParam() >= 0)    // Age of differential GPS data, seconds  Null
                if(GPS_ReadParam('*') >= 0) // Differential reference station ID      Null
                if(GPS_ReadParam(10) >= 2)  // Checksum                               *HH (two ↩
                    hex digits representing an 8 bit exclusive OR of all characters between, but↩
                     not including, the '$' and '*')
                {
                  if(gps_chk == GPS_Check((char*)&gps_p[0])) // Checksum validation
                  {
                    gps_lat = newlat;
                    gps_lon = newlon;
                    gps_alt = newalt;
                    gps_valid = (newsta != '0');
                    /*
                    if(gps_valid)
                    {
                      LLA2XYZ(gps_lat, gps_lon, gps_alt, &data.x, &data.y, &data.z); // IMU ↩
                           calibrated
                    }
                    */
                    // Computation of the magnetic vector declination
                    #define wmm(pos) ((char)EEPROM_read(DECLINATION_ADDR + pos))
                    int wmm_lat = (int)gps_lat;
                    int wmm_lon = (int)gps_lon;
                    float wmm_yb = gps_lat - wmm_lat;
                    float wmm_ya = 1.0 - wmm_ya;
                    float wmm_xb = gps_lon - wmm_lon;
                    float wmm_xa = 1.0 - wmm_xa;
                    int wmm_pos = (wmm_lat / 10 + 7) * 25 + (wmm_lon / 15 + 12);
                    mag_dec = ((float)wmm(wmm_pos)) * wmm_xa * wmm_ya +
                              ((float)wmm(wmm_pos + 1)) * wmm_xb * wmm_ya +
                              ((float)wmm(wmm_pos + 25)) * wmm_xa * wmm_yb +
                              ((float)wmm(wmm_pos + 26)) * wmm_xb * wmm_yb;
                  }
                }
              }
            }
          }
        }
      }
    }
  }

  // Attitude control
  if(attitude)
  {
    // PID controller
    outputU = CalculatePID(data.a, targetU, &accU, &lastU); // PID controller for pitch control
    outputV = CalculatePID(data.b, targetV, &accV, &lastV); // PID controller for yaw control
    outputW = CalculatePID(data.c, targetW, &accW, &lastW); // PID controller for roll control
  }

  switch(mission)
  {
    case MISSION_INOP:    // Any main system fail or GPS not aligned. The satellite is not ready↩
         to be launched
      // Reset the configuration to defaults
      if(!gpsActive) GPS_activate(true);
      if(gpsActive)
      {
        SetMission(MISSION_RAMP);
      }
      break;
    case MISSION_RAMP:    // IMU integrators are active and waiting for balloon release. Next ↩
         mission state when first stage burn-in is detected. The balloon from an external GPS or ↩
         a person actives the first stage ignition. The satellite can not active the first stage ↩
         by it self
      if(maxg)
      {
        // Guardar la actitud del los gyros
        // Guardar en la EEPROM las coordenadas GPS
        EEPROMWriteVar(LAT_ADDR, gps_lat); // First known stage coordinates
        EEPROMWriteVar(LON_ADDR, gps_lon);
        EEPROMWriteVar(ALT_ADDR, gps_alt);
        // Guardar el vector magnetico terrestre
        GPS_activate(false);
        AttitudeOn(2, 5, 1);
        SetMission(MISSION_STAGE1);
      }
      break;
    case MISSION_STAGE1:  // IMU integrators are active and vector control is active. Next ↩
         mission state when first stage burnout is detected
      ControlVectorTrajectory();
      ControlVectorUpdate();
      if(!maxg)
      {
        I2C_TurnOn(IOeEN_GPS);
        GPS_activate(true);
        SetMission(MISSION_BURNOUT1);
      }
      break;
    case MISSION_BURNOUT1:// IMU integrators are active and vector control is in idle in order ↩
         to keep the attitude. Next mission stage one minute before apogee and detected by low ↩
         vertical speed
      ControlVectorTrajectory();
      ControlVectorUpdate();
      if(speedv < 80)
      {
        targetU = 0;
        AttitudeOn(2, 5, 1);
        SetMission(MISSION_SPIN);
      }
      break;
//    case MISSION_TURN:    // INOP. IMU integrators are active and a vector control maneuver is↩
      done to match the apogee plane. Next mission stage when maneuver is completed
//      // NOT IMPLEMENTED
//      ControlVectorUpdate();
//      if(speedv < 100)
//      {
//        SetMission(MISSION_TILT);
//      }
//      break;
//    case MISSION_TILT:    // INOP. IMU integrators are active and a vector control maneuver is↩
      done to match the heading plane parallel to the ground. Next mission stage when maneuver is↩
      completed
//      // NOT IMPLEMENTED
//      ControlVectorUpdate();
//      if(true)
//      {
//        SetMission(MISSION_SPIN);
//      }
//      break;
    case MISSION_SPIN:    // IMU integrators are active and a spin is given to the launcher. At ↩
         this point an orbital speed prediction is calculated based on the current altitude. If ↩
         GPS is available at orbit, it will be replaced for the real one. Next mission stage when↩
          spin is achieved and apogee is reached or passed then second stage ignition is ↩
         activated before
      targetW = INFINITE; // Force attitude control to spin
      ControlVectorUpdate();
      if((data.w > 1000) && (digitalRead(IGNITION_PIN) == LOW))
      {
        digitalWrite(IGNITION_PIN, HIGH);
        timeStamp = 0;
      }
      if(maxg)
      {
        targetW = 0; // Stop spin
        SetMission(MISSION_STAGE2);
      }
      else
      {
        timeStamp += data.d;
        if(timeStamp > 10000000) // 10 seconds without ignition goes to STANDBY
        {
          targetW = 0; // Stop spin
          SetMission(MISSION_STANDBY); // CAUTION: Injection sequence failed. Mission time ↩
               reduced
        }
      }
      break;
    case MISSION_STAGE2: // IMU integrators are active while the orbital speed is achieved. Next↩
          mission state when second stage burnout is detected. If ignition fails, next mission ↩
         stage will be STANDBY
      if(!maxg) SetMission(MISSION_DEPLOY); // Orbital speed is not detected, only the burn out ↩
           of the last stage
       break;
    case MISSION_DEPLOY:  // IMU integrators are active while satellites are deployed. Next ↩
         mission state when attitude control is achieved. If attitude control fails, next mission↩
          stage will be STANDBY
      if(MagnetorquersCycle)
      {
```

```
      //EEPROMWriteVar(TARGET_U_ADDR, 0);
      //EEPROMWriteVar(TARGET_V_ADDR, 0);
      //EEPROMWriteVar(TARGET_W_ADDR, 0);
      targetU = 0;
      targetV = 0;
      targetW = 0;
      SetMission(MISSION_DAMPING);
    }
    break;
  case MISSION_DAMPING: // IMU integrators are active while a damping maneuver is done by ↩
       Magnetorquers. Next mission state when satellite attitude is stabilized
    MagnetorquersUpdate();
    if ((abs(targetU-data.a) < 0.1) && (abs(targetV-data.b) < 0.1) && (abs(targetW-data.c) < ↩
        0.1))
    {
      SetMission(MISSION_STANDBY);
    }
    break;
  case MISSION_STANDBY: // IMU integrators are active but all the functions are hibernated. ↩
       Every 10 seconds it looks for any near ground station below 1200 km. Next mission state ↩
       when any ground station is near
    if(MagnetorquersCycle)
    {
      I2C_TurnOff(IOeEN_A);
      I2C_TurnOff(IOeEN_B);
      I2C_TurnOff(IOeEN_C);
      I2C_TurnOff(IOeEN_D);
    }
    if(StationInRange()) SetMission(MISSION_FOLLOWING);
    // HIBERNATE FOR A WHILE;
    break;
  case MISSION_FOLLOWING: // IMU integrators are active while the attitude maneuver is done to↩
        point towards the ground station. Other near stations are checked in between. Next ↩
       mission state when the target is locked. If ground station is out of range, next mission↩
        stage will be STANDBY
    MagnetorquersUpdate();
    if((abs(data.u) < 0.1) && (abs(data.v) < 0.1) && (abs(data.w) < 0.1)) // Target locked
    {
      Payload_activate(true);
      SetMission(MISSION_PAIRING);
    }
    break;
  case MISSION_PAIRING: // Same as FOLLOWING but radio-link is tested. Next mission state when↩
        link is established and new commands are uploaded. If radio-links fails keeps the ↩
       current mode. If a new nearest ground station is available, next mission stage will be ↩
       FOLLOWING. If ground station is out of range, next mission stage will be STANDBY
    MagnetorquersUpdate();
    // Sending request to the locked ground station every second when GPS is active
    if(gpsActive) nRF_sendd((byte *)&data.i, PL_SIZE); while(!((nRF_getReg(STATUS) & ((1 << ↩
        TX_DS) | (1 << MAX_RT))))); nRF_setReg(STATUS, (1 << TX_DS) | (1 << MAX_RT));
    if(nRF_dataReady())
    {
      nRF_getData((byte *) &data.x, PL_SIZE); // New commands available
      SetMission(MISSION_DOWNLOAD);
    }
    else if (false) // Locked ground station out of range
    {
      SetMission(MISSION_STANDBY);
    }
    break;
  case MISSION_DOWNLOAD: // Same as FOLLOWING but download is done. If radio-links fails of a ↩
       new nearest ground station is available, next mission stage will be FOLLOWING. If ground↩
        station is out of range, next mission stage will be STANDBY
    MagnetorquersUpdate();
    //nRF_sendd((byte *)&data.i, PL_SIZE); while(!((nRF_getReg(STATUS) & ((1 << TX_DS) | (1 <<↩
         MAX_RT))))); nRF_setReg(STATUS, (1 << TX_DS) | (1 << MAX_RT));
    if(false) // Locked ground station out of range
    {
      SetMission(MISSION_STANDBY);
    }
    break;
  default:            // Mission phase number out of range. Changes to STANDBY
    SetMission(MISSION_STANDBY);
```

```
      break;
  }
  MagnetorquersCycle = false; // end of cycle
}

#else

void setup()
{
  float val;
  Serial.begin(4800);

  // Used only in RESET procedure
  #define EEPROMWriteWaypoint(pos, lat, lon) \
    val = lat; EEPROMWriteVar(WAYPOINTLIST_ADDR + (pos - 1) * 8, val); \
    val = lon; EEPROMWriteVar(WAYPOINTLIST_ADDR + (pos - 1) * 8 + 4, val);
  // Cycles for alignment procedure
  #define ALIGN_CYCLES 1000

  // EEPROM dumping
  for(int addr = 0; addr < 512; addr++)
  {
    if ((addr % 16) == 0)
    {
      Serial.println(); Serial.print(addr / 1000.0, 3); Serial.print(": ");
    }
    Serial.print(EEPROM.read(addr) / 1000.0, 3);
    Serial.print(" ");
    EEPROM_write(addr, 0); // WARNING: This line will erase all the EEPROM memory
  }

  /*
   * FEMTO-SATELLITE CONTROL PROTOCOL
   * The following code will decide the activity of the femto-satellite
   * The satellite follows the closer waypoint
   * The target waypoint will have preference over the others in the list
   * A waypoint can be a point to record, a ground station or both
   * We recommend to put a list of ground station at the end of the list to ensure the downlink
   * For video record purposes, closer than 50 km waypoints are useless
   * Each waypoint covers 50 km of radius of video
   * Choose a not aligned waypoints distribution because femto-satellite trajectory changes ↩
       every orbit
   * For downlink purposes, we recommend 1,000 km ground stations separation
   * A ground station can not see the femto-satellite beyond 1,100 km of distance
   * Less than 500 km between ground stations is not efficient
   * There is a lost for the downlink time due to the change of ground station
   */

  // New Target waypoint having priority over others
  EEPROM_write(TARGET_ADDR, 0); // 0 = No Target waypoint, {1,...,8} Target waypoint

  // New waypoint list
  // 01: WikiSat,Barcelona,41.275427,1.986917,17.0
  EEPROMWriteWaypoint(1, 41.275427, 1.986917);
  // 02:
  EEPROMWriteWaypoint(2, INVALID, INVALID);
  // 03:
  EEPROMWriteWaypoint(3, INVALID, INVALID);
  // 04:
  EEPROMWriteWaypoint(4, INVALID, INVALID);
  // 05: WikiSat,Barcelona,41.275427,1.986917,17.0
  EEPROMWriteWaypoint(5, 41.275427, 1.986917);
  // 06: WikiSat,El Arenosillo,37.096574,-6.738729,31.0
  EEPROMWriteWaypoint(6, 37.096574, -6.738729);
  // 07: WikiSat,Maspalomas,27.762892,-15.6338072,204.9
  EEPROMWriteWaypoint(7, 27.762892, -15.6338072);
  // 08: WikiSat,OZ7SAT,55.732055,12.394315,33.0
  EEPROMWriteWaypoint(8, 55.732055, 12.394315);

  Serial.println();
  Serial.println("ALIGN SEQUENCE STARTED");
  Serial.println("Satellite should be in the launcher in vertical position");
  Serial.println("Do not connect the satellite to the USB cable");
  data.x = 0; // Position [m]
  data.y = 0;
  data.z = 0;
  data.a = 0; // Angular position. Attitude [Â?]
  data.b = 0;
  data.c = 0;
  data.l = 0; // Inertial speed [m/s]
  data.m = 0;
  data.n = 0;
  i = 0; // Gravity vector
  j = 0;
  k = 0;
  u = 0; // Precession rate of turn
  v = 0;
  w = 0;
  r = ALIGN_CYCLES;
  //Serial.println("IMU Align initialized."); // Remove this message if memory required
  while(r--)
  {
    i += I2C_GetValue(AccID, AccX);
    j += I2C_GetValue(AccID, AccY);
    k += I2C_GetValue(AccID, AccZ);
    u += I2C_GetValue(GyroID, GyroX);
    v += I2C_GetValue(GyroID, GyroY);
    w += I2C_GetValue(GyroID, GyroZ);
  }
  bias.i = i / ALIGN_CYCLES;
  bias.j = j / ALIGN_CYCLES;
  bias.k = k / ALIGN_CYCLES;
  bias.u = u / ALIGN_CYCLES;
  bias.v = v / ALIGN_CYCLES;
  bias.w = w / ALIGN_CYCLES;
  EEPROMWrite(IMU_ADDR, 6, (uint8_t *)&bias.i);

  // Earth magnetic field declination matrix
  // Epoch: 2012, Altitude: 250000m, Latitude step: 10, Longitude step: 15, Latitude range: ↩
      -70,+70
  char wmm[15][25] =
  {
    // -180  -165  -150  -135  -120  -105   -90   -75   -60   -45   -30   -15     0   +15   +30 ↩
           +45   +60   +75   +90  +105  +120  +135  +150  +165  +180
    {  84,   73,   65,   58,   51,   43,   34,   25,   16,    7,   -2,  -11,  -21,  -31,  -43,  ↩
        -55,  -66,  -78,  -91, -105, -124, -128,  127,  102,   84}, // -70
    {  46,   45,   43,   41,   38,   34,   28,   19,   10,    1,   -6,  -13,  -20,  -30,  -40,  ↩
        -51,  -60,  -67,  -72,  -72,  -57,    3,   39,   46,   46}, // -60
    {  30,   30,   30,   30,   29,   27,   23,   14,    3,   -6,  -13,  -17,  -21,  -29,  -38,  ↩
        -47,  -53,  -55,  -50,  -37,  -14,    7,   21,   27,   30}, // -50
    {  22,   22,   22,   22,   22,   21,   18,    9,   -3,  -13,  -19,  -22,  -23,  -26,  -33,  ↩
        -40,  -43,  -40,  -31,  -16,   -3,    7,   14,   19,   22}, // -40
    {  16,   17,   17,   17,   17,   16,   13,    4,   -8,  -18,  -23,  -24,  -22,  -19,  -23,  ↩
        -28,  -30,  -26,  -17,   -7,    0,    6,   11,   14,   16}, // -30
    {  13,   13,   14,   13,   13,   12,    9,    0,  -12,  -20,  -23,  -22,  -17,  -11,  -11,  ↩
        -15,  -17,  -15,   -9,   -2,    2,    5,    8,   11,   13}, // -20
    {  11,   11,   11,   11,   10,    9,    6,   -3,  -14,  -21,  -21,  -17,  -11,   -5,   -3, ↩
          -6,   -9,   -8,   -4,    0,    2,    4,    7,    9,   11}, // -10
    {   9,    9,   10,   10,    9,    8,    4,   -5,  -14,  -19,  -18,  -12,   -6,   -2,    0, ↩
          -1,   -4,   -4,   -2,    0,    1,    2,    5,    8,    9}, //   0
    {   9,    9,    9,   10,    9,    7,    2,   -6,  -15,  -18,  -15,   -8,   -3,    0,    2, ↩
           1,   -1,   -2,   -1,    0,   -1,    0,    3,    7,    9}, // +10
    {   8,    9,   10,   11,   10,    7,    1,   -8,  -15,  -16,  -12,   -6,   -2,    1,    3, ↩
           2,    0,   -1,   -1,   -1,   -2,   -2,    1,    5,    8}, // +20
    {   7,   10,   12,   13,   12,    8,    0,  -10,  -15,  -15,  -11,   -6,   -1,    2,    3, ↩
           3,    2,    1,    0,   -2,   -4,   -5,   -2,    2,    7}, // +30
    {   5,   10,   14,   15,   14,    8,   -1,  -12,  -17,  -15,  -11,   -6,   -1,    2,    4, ↩
           5,    5,    3,    1,   -3,   -7,   -8,   -5,    0,    5}, // +40
    {   5,   11,   15,   18,   16,    9,   -3,  -15,  -20,  -18,  -13,   -7,   -2,    3,    6, ↩
           8,    9,    7,    3,   -3,   -9,  -11,   -8,   -2,    5}, // +50
    {   4,   12,   17,   21,   19,   10,   -7,  -21,  -25,  -23,  -17,  -10,   -3,    3,    8, ↩
          13,   14,   13,    6,   -3,  -11,  -13,  -10,   -3,    4}, // +60
    {   5,   13,   20,   24,   22,    8,  -18,  -33,  -34,  -30,  -22,  -13,   -5,    4,   12, ↩
          18,   22,   21,   13,   -1,  -11,  -14,  -10,   -3,    5}  // +70
  };
  EEPROMWrite(DECLINATION_ADDR, sizeof(wmm), (byte*)&wmm[0][0]);
  Serial.println();
  Serial.println("ALIGN SEQUENCE COMPLETED");
  Serial.println("CAUTION: satellite should be programmed:");
  Serial.println("Change RESET_EEPROM define to false before flight");
  while(true); // Satellite programming is mandatory at this point
}

void loop() {while(true);} // Satellite programming is mandatory at this point

#endif
```