POLITECNICO DI TORINO

Dipartimento di Elettronica e Telecomunicazioni

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Development of a University Femtosatellite

Relatori:

Prof. Leonardo Reyneri

Prof. Claudio Sansoè

Prof. Dante Del Corso

Candidato:

Donato Russo

July 2013

Abstract

Recently, the attention in designing small satellites has rapidly grown by European universities, Politecnico di Torino has adopted several standards and put its effort in developing its own satellite.

AraMis (*Architettura Modulare per Satelliti*) is the evolution of previous project PiCPot and consists in the development of nano and pico satellites orbiting in LEO (Low Earth Orbit) for university purposes.

In particular AraMis aims to build a modular satellite configurable in different shapes in which each module is independent from the other ones and capable of working in stand-alone mode. The communication among the elements, known as tiles, takes place through proper buses.

Each module has a specific role and can work alone or together with other modules to improve the performance of the system. The project is considered as low-cost since it uses COTS components and exploit the re-usability of different units so reducing design costs.

Object of this thesis is to exploit AraMis standard to develop a femtosatellite (a satellite having a weight lower than 100g) having features of a typical satellite with in addition some peculiarities.

The idea is to develop a constellation of femtosatellites able to communicate each other, exchange generic messages, measure the distances inside the constellation and modify the distance according to the user needs. To start with, we consider a constellation made of two satellites placed opposite one to the other.

The system has a modular structure, following the AraMis philosophy, and it is made of a magnetometer, a magnetic actuator, a solenoid and a telecommunication module as main units. It

has a disk shape, where on a surface there is the RF electronics, on the other one there are solar cells for supply system, finally the solenoid is placed around the disk.

The magnetometer and magnetic actuator are taken from AraMis project and adapted to the particular case while the telecommunication module is entirely designed and developed customized for the application. The communication module (named 1B35_Intersatellite_Communication in the thesis) is realized so it can be re-utilized for future works involving a communication between two identical nodes (placed on different satellites or on the same satellite).

The system is based on CC2510 system on chip provided by Texas Instrument and simpliciTi protocol for telecommunication. The main reasons are:

- low cost;
- low power consumption;
- custom protocol (simpliciTi), provided by Texas Instrument easy to use;
- RSSI technique integrated;
- possibility to use 2.4GHz as working frequency;

One of the system purposes is to measure the distance between two satellites, this is achieved through RF techniques (RSSI) and magnetic techniques (generation and measurement of magnetic field). Another important issue is to let satellites get closer or separated, the motion is obtained through attractive and repulsive magnetic force, generated by the actuator and electromagnet (also used in measurement operations).

In this thesis, we are introducing for the first time a femtosatellite in the AraMis project, so no particular initial constraints are given. During the thesis several configurations have been considered and analyzed to find the better solution for the case study. The general structure of the system is completely described and documented by using UML language, for a particular unit (the telecommunication unit) we proceed instead with the hardware and software realization.

As results, we have set up the general structure of the femtosatellite, analyzed several issues and implemented a good basis for future development. One possible application could be using the femtosatellites to transport small objects in space environment.

Finally, another important result is the realization of the telecommunication unit, fully implemented in hardware and software but not tested.

Summary

Index of tables and figures

Introduction

1.1 AraMis

In this paragraph the general context of the thesis and a brief description of AraMis standard will be presented.

1.1.1 Politecnico di Torino and Nanosatellites

In the last years aerospace and satellite market has grown in a remarkable way due to the decreasing in launchers and launches costs used to put the satellites in orbit.

This aspect bring the Politecnico di Torino like other universities and companies worldwide to focus on the implementation of low-cost small satellites. As a consequence different standards were born like CUBESAT and ARAMIS in order to make easier the design and development of these kind of space systems.

Another interesting peculiarity of these university projects is the usage of COTS (Commercial Off The Shelf) elements, i.e. low-cost components easy to find .

1.1.2 AraMis standard and specifications

AraMis (Architettura Modulare per Satelliti) is a project started in 2006 by Politecnico di Torino which aim to design small satellites with a modular structure in order to improve the CubSat standard in which the system is build with an ad-hoc architecture.

The satellites are classified basing on the mass in the following way:

- microsatellite : mass between 10 and 100 Kg
- nanosatellite: mass between 1 and 10 Kg
- picosatellite: mass between 100 g and 1 Kg
- femtosatellite: mass lower than 100 g

AraMis is considered as a nanosatellite, its basic architecture is based on one or more tiles placed on the outer side of the system so that they also play a structural function. Putting together several tiles a cubic or prismatic shape can be built as shown in *Figure 1.1* and *Figure 1.2*.

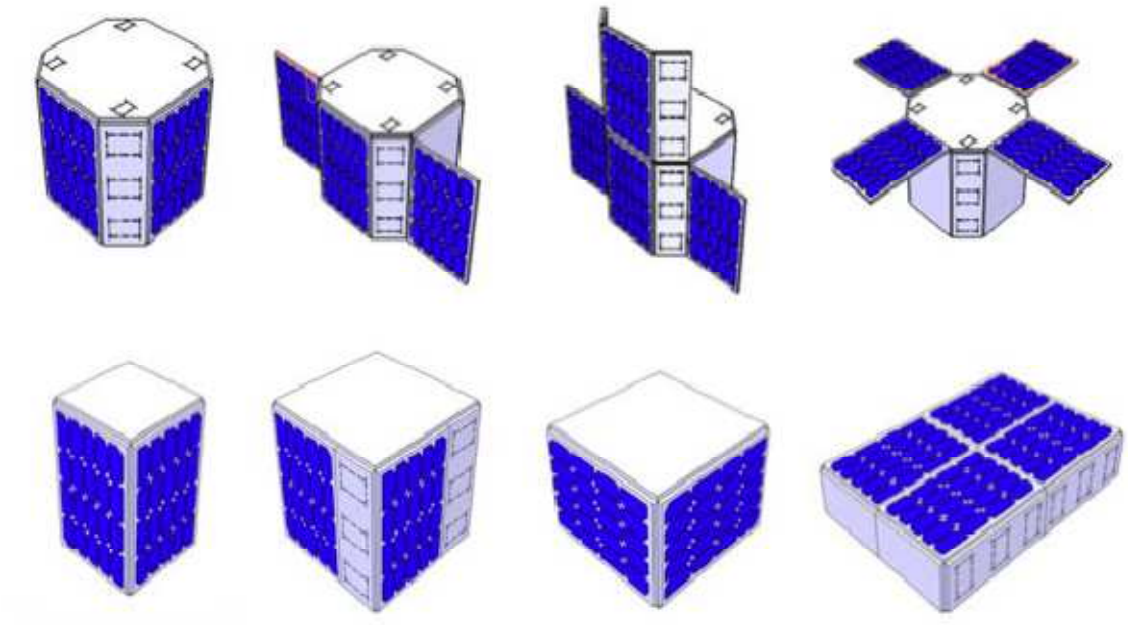


Figure 1.1 Possible configurations of an AraMis satellite.

Figure 1.2 Example of a parallelepiped structure.

The standard modules of AraMis are:

- Power Management Tile - it is used to generate, control and store the energy necessary to supply the whole system. In particular solar panels are chosen which supply the system and charge the batteries through proper circuits. The batteries are used as reserve of energy if the one coming from the sun would not be enough. The structure of the satellite is mainly built by putting together tiles of this kind, they also have microcontrollers which manage the tile and communicate with OBC through buses. An example is shown in *Figure 1.3*.

- Telecommunication Tile - it has the function to let the satellite communicate with the ground segment, it manages the information related to the attitude control coming from the Earth and transfer it to the Power Management Tiles to perform actuation.

- On Board Computer - it is the module containing the processor in charge to coordinate the whole satellite operation. As an example, it makes all calculations related to attitude control and governs the other tiles such that actuation takes place.

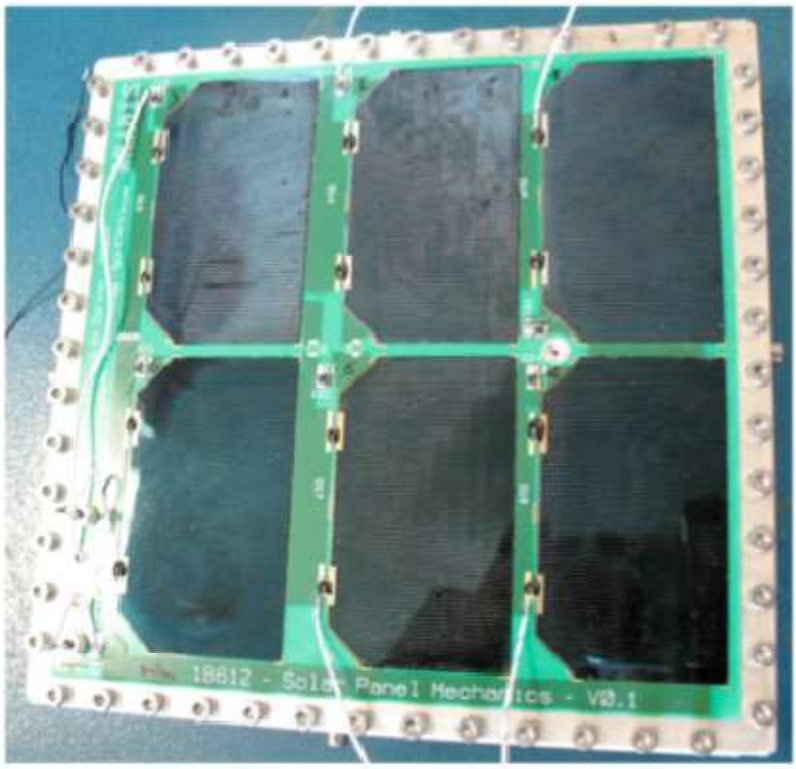


Figure 1.3 Module with a size of 164 x 165 mm with solar cells.

1.2 FemtoSat

The purpose of this thesis is to realize a university femtosatellite placed in a constellation of identical satellites (at least two) able to:

- perform typical operations of a bigger satellite
- let two satellites communicate each other
- measure and control the distance between them.

The system has a modular structure as AraMis standard suggest, in particular it is made of:

- 1B35_Intersatellite_Communcation
- Bk1B221_Magnetometer_Sensor
- Bk1B222_Magnetic_Torque_Actuator

- An electromagnet (solenoid)

The first module is used to implement the communication between two satellites, the functionalities of CC2510 System on Chip are exploited to also perform OBC functions. The magnetometer is used to evaluate the magnetic field around the satellite in a distance measurement operation. Finally the motion of the system is assigned to an actuator unit which has as core system the *Bk1B222_Magnetic_Torque_Actuator* and an electromagnet. The *1B35_Intersatellite_Communcation* is object of this thesis as well and it will be described in the following chapters. The *Bk1B221_Magnetometer_Sensor, Bk1B222_Magnetic_Torque_Actuator* are taken from the AraMis project and used for the case study so exploiting the concept of modularity and re-usability. The electromagnet choice is described in the paragraph *VI.1*

The idea is to design a satellite having a disk shape, in the inner part will be placed the PCB, around the disk there will be the coil used to generate the magnetic field, on one face there will be solar cells while the other side will be placed the RF circuit with the antenna.

In the following picture there is a draft of the structure:

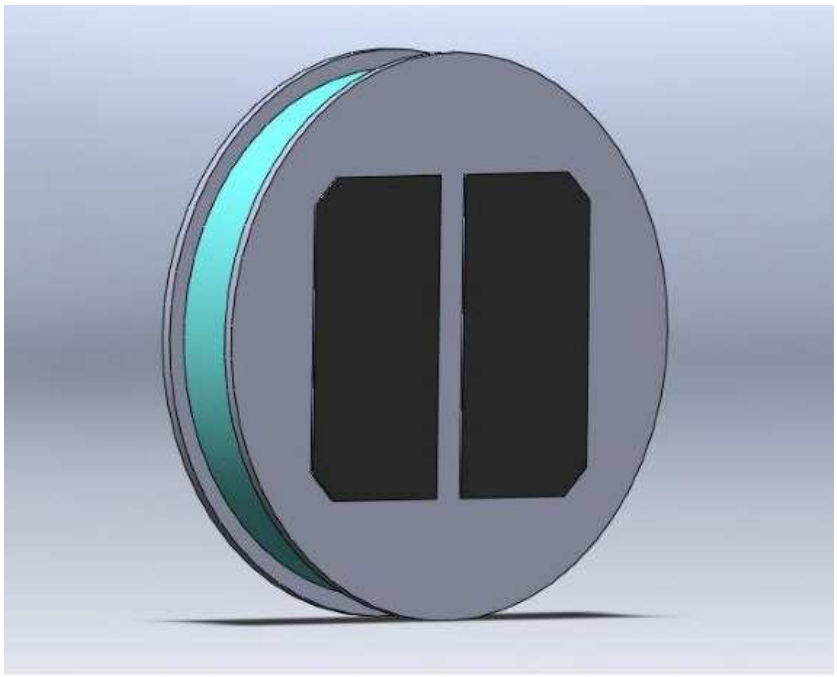


Figure 1.4 - Qualitative representation of the Femtosatellite

There are no particular constraints except the weight that shall be lower than 100g. Some of free parameters that will be evaluated and analyzed during the design are:

- Minimum/Maximum distance
- Satellite dimension
- Solenoid dimension
- Electrical quantities

All this subjects will be presented in chapter *IV.1*.

1.2.1 Operating Environment

In the design of a space system the designer has to face with several issues related to the particular environment as space is. Among them we have:

- temperature;
- pressure;
- ionizing radiotion;
- terrestrial magnetic fields;

Since the femtosatellite has to deal with magnetic field generation and measurement, for sure the latter issue is very important to be considered.

In fact, once the current flow inside the solenoid, it generates its own dipole which forces the system align with the terrestrial magnetic field. We adopt a particular technique to eliminate the contribution of terrestrial magnetic field when we need to isolate the magnetic field generated by the interlocutor satellite (See chapter ...)

Magnetic field around the Earth is of sum of more contributions [4], each one having different origin:

- **Main Field,** generated by fluid nucleus of the Earth;
- **Crustal Field,** generated by magnetized rocks of Earth's crust;
- **External Field,** generated by electrical currents taking place in ionosphere and magnetosphere;

- **Field of electromagnetic induction,** generated by induced currents in the crust and muntle by the external field (it is variable in time);

Among them, the main field generate the 99% of the total magnetic field measurable on the surface. Several studies demonstrates that this field can be compared (at 95%) with the one generated by a magnetic dipole centered in the Earth's center and with a misalignment of 11° 30' with respect to the Earth' s rotation axis.

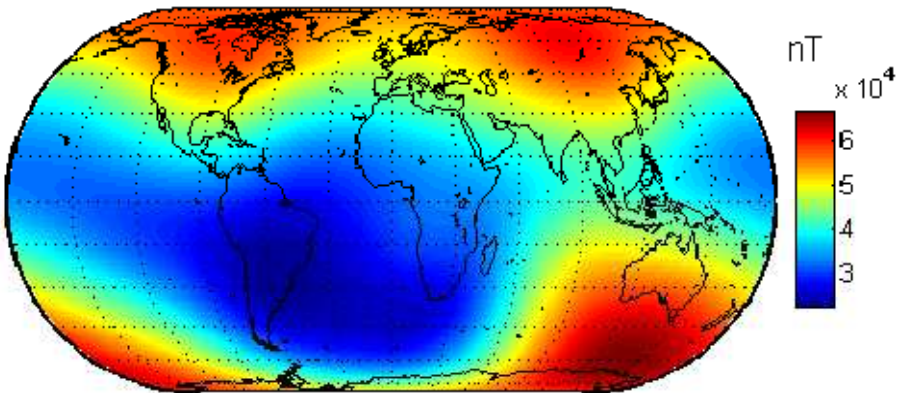


Figure 1.5 - Distribution of terrestrial magnetic field in 2009 [5]

Earth' magnetic field is expressed conventionally through the vector of magnetic induction $\vec{B}$. Its measurement unit is Tesla (T), but in practical applications the Gauss (G) is often use which is equals to 10^{-4} T. The absolute value of the magnetic field on Earth's surface varies from 0.2 G at equator to 0.7 G at poles (the intensity distribution is shown in *Figure 1.5)*

As conclusion, we assume possible values for the field in LEO orbit in the range -0.625 and 0.625G, values used as specification for the magnetometer/magnetic torque actuator chosen in this thesis at design time (see previous chapter)

1.2.2 Managing intersatellite communication

The project started as stand-alone project, then, while the design procedure was proceeding , the necessity to design a module dedicated to the communication between two satellites has born.

This issue is in common with all projects where the communication among more nodes has to take place. Moreover, the two nodes can be placed on the same satellite (inter-board communication) or different ones.

One example sharing this kind of problem is Aram-Dock project [1] developed by another student. We decide to work together in order to design a "stand-alone" module, re-usable in other projects, where specifications have been agreed together: the 1B35_Intersatellite_Communication module.

We decide to divide the work into two parts, hardware and software. The first steps of the design and the description in UML are carried out together. All software contents are taken from Aram-Dock thesis while the hardware is developed in this thesis.

1.3 1B35 Intersatellite Communication Module

1B35 module is used to govern the intersatellite communication in AraMis project.

The communication is based on SimpliciTi protocol and on the CC2510 system on chip (SoC), both provided by Texas Instrument.

CC2510 contains a low-power transceiver, a microcontroller MCU 8051 , 32kB flash programmable memory, 4kB RAM and works at a frequency of 2.4 GHz. The extremely reduced size (6mm x 6mm) and power consumption makes it a perfect partner for building nano and femto satellites.

SimpliciTi is a network protocol used in RF field which requires a very small memory cost (8 kB maximum of flash and 1kB maximum of RAM). It implements two network topologies: pure peer-to-peer and star topology. In 1B35 project a peer-to-peer communication is chosen, the software drivers are based on several calls to an API (Application Programming Interface) interface which allow the connection during run-time. Sleep mode can be applied to extend the duration of functioning.

The main idea is to use more than one 1B35 modules to be placed on different satellites in order to let them communicate each other. For this reason the system can act both as master and slave depending on the use cases and operations which are carried out.

1B35 needs to be configured once by the configurator (one of main actors), operation in which the ID is set. The ID is a necessary reference to correctly address the messages in the communication but also to identify the modules between which a distance is measured. In fact, one of main purposes of 1B35 is to determine the distance with respect to another interlocutor node by using RSSI technique (which derived from Friis theory). SimpliciTi protocol has a perfect structure to accomplish this task since in the body of any kind of message there is a field which contains RSSI value.

1B35 module does not have an on-board system to provide the power supply,it receives power from other modules.

1.4 UML Language

The design of AraMis has been carried on by using UML language (Unified Modeling Language) [2].

The same language is used in this thesis to describe all modules contained in both FemtoSat and 1B35 projects, in this paragraph there is a brief introduction to the UML language.

UML is a visual language born in 1995 for designing software, but it can be optimally adapted to the description of systems made both of hardware and software. It is based on the representation of entities involved in the system functioning and all interactions among them. It offers several advantages, the most important are:

- Make easier the project understanding, even by people external to the project, thanks to a graphical/conceptual representation of the elements that make the system (components, subsystems, signals, functions...) starting from a high level description to a specific one.
- Simplify and improve the description of system functionalities and the specification definition providing a common basis in the approach of designing the units forming the whole system.
- Make exportable the system building blocks (which are independent form each other) such that they can be re-used in other projects so implementing the modularity concept.

The tool chosen to adopt UML is Visual Paradigm for UML [3] which also provides automatic code generation.

Among all possible utilities that UML offers, there are three main types of diagrams which have been used to make the project described in this thesis: use case diagram, class diagram and sequence diagram.

1.4.1 Use Case Diagram

The use case diagram describes functionalities, project specifications and by what/by who those can be played. It is the starting point in the system modeling and contains the following categories of element:

- Actor - it is a generic entity, an human user, another system or the external environment, which interacts with the system under design, asking for the implementation of one or more use cases. There can be more than one actor in a use case diagram and is represented as depicted in *Figure 1.6*

Figure 1.6: UML language - Actor.

- Use case - it is a task which the actor ask to the system i.e. the objectives of the project. It can be called directly to the actor or related to other use cases as shown in *Figure 1.7*

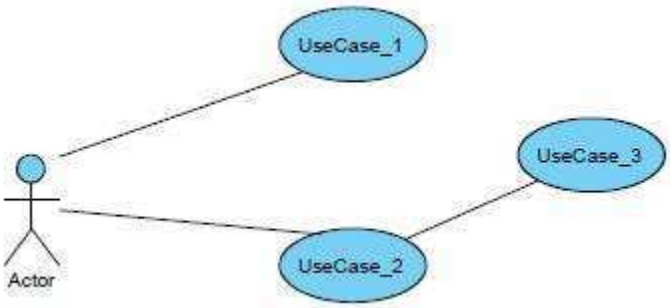


Figure 1.7: UML language - Use cases

- Relations among actors - there exist several kind of relations, e.g. the generalization in which the actor A , from which the arrow starts and points toward actor B, can execute its use cases and all use cases related to B. In this case A is the generalization of B.

- Relations among use cases - in the following some examples:

  - Generalization: similar to the one described for the actors;

  - Inclusion: identified by a dashed arrow (*Figure 1.8*) with a label <<include>>, it indicates that the basic use case includes also the actions that the included use case can execute on the system;

  - Extension: Graphically it is similar to the Inclusion, but with a label <<extend>> and indicates that the extended use case is an optional functionality of the basic use case (*Figure 1.9*);

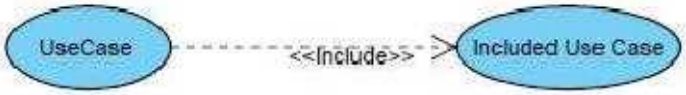


Figure 1.8: UML language - Use case inclusion

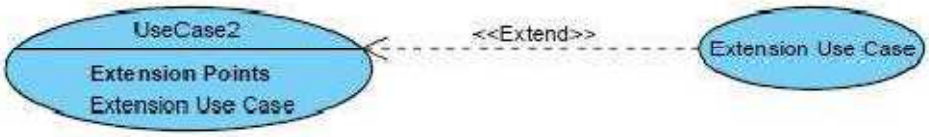


Figure 1.9: UML language - Use case extension

- Association between actor and use case - it is identified by a straight line and indicates which actor has the possibility to put in action specific use cases i.e. by which actor each use case is required; one actor is often associated to more use cases and one uses case can be associated to more than one actor.

Moreover, it's useful to highlight the possibility to add a documentation to each element inside the UML project. This documentation (containing the element description, comments, information useful for the designer) together with the diagrams, the software routines and hardware schematics constitutes the complete description of the project.

1.4.2 Class Diagram

The class diagram is made of **objects** with their associations and it is used to deeply characterize the system under project by describing all its components, hardware, mechanical, software and mixed (hardware/software).

An **object** is an entity belonging to the system which interacts with objects of the same system or other external ones. The interactions among objects is described in the sequence diagrams (illustrated in the next paragraph).

The **class** is the abstraction (generalization) of an object which represents a specific instance (see *Figure 1.10*). The **attributes** and **methods** (named as **operations** in the class) are fundamental features of a class and its instances.

- An **attribute** is a property of the object and it can be logical, physical, etc e. g. if the object is a sensor its attributes can be the sensitivity and power consumption. While in software field the attributes are variables and C language structures. Each attribute can be characterized by a type (for example *int* for integer type), an initial value, the visibility and other specific properties. Another emblematic case is the one in which an attribute is an instance of another class.

- The **methods** indicate which operations the object can perform and how this object interface with the other elements of the system. They are C functions for software objects or signals (wires) for hardware objects. They can be associated to a return value (which returns to the calling object) and to several parameters which it receives from calling object.

A class diagram illustrates the objects and classes in a specific hierarchy, connected through different types of associations. Other than the ones described for use cases, there is another

particular example called *composition*, very useful in modular projects to represent father objects made of various son objects.

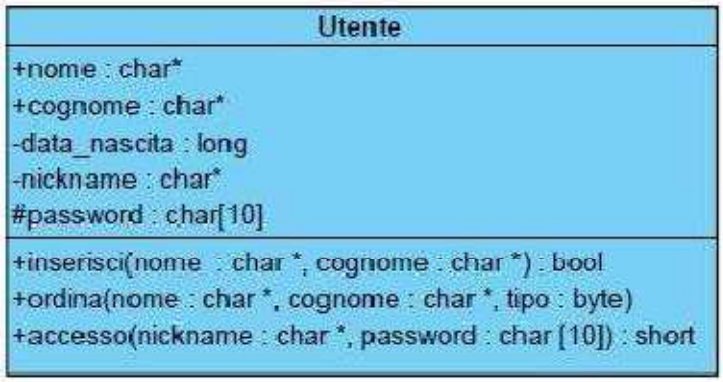


Figure 1.10: UML language - Class example

1.4.3 Sequence Diagram

Sequence diagrams describe the carrying out of actions performed by the system i.e. the use cases. Infact, each of them is split into a set of actions which follow to let the use case takes place. Then the objects ,which make the system, implement the operations through **messages** containing calls to methods. Each object can call a method of another one (if it is visible) or its own methods.

The best representation of this *course of action* is the sequence diagram where the messages are listed in exact timeline through the association of a order number. Each object is related to a *lifeline* in which the time increases if the line is read from the top to the bottom.

There are many other features that describe the operations in the sequence diagram e.g. to implement loops, if-then-else constructs, nested sequence diagrams and so on, In the *Figure 1.11* is presented an example.

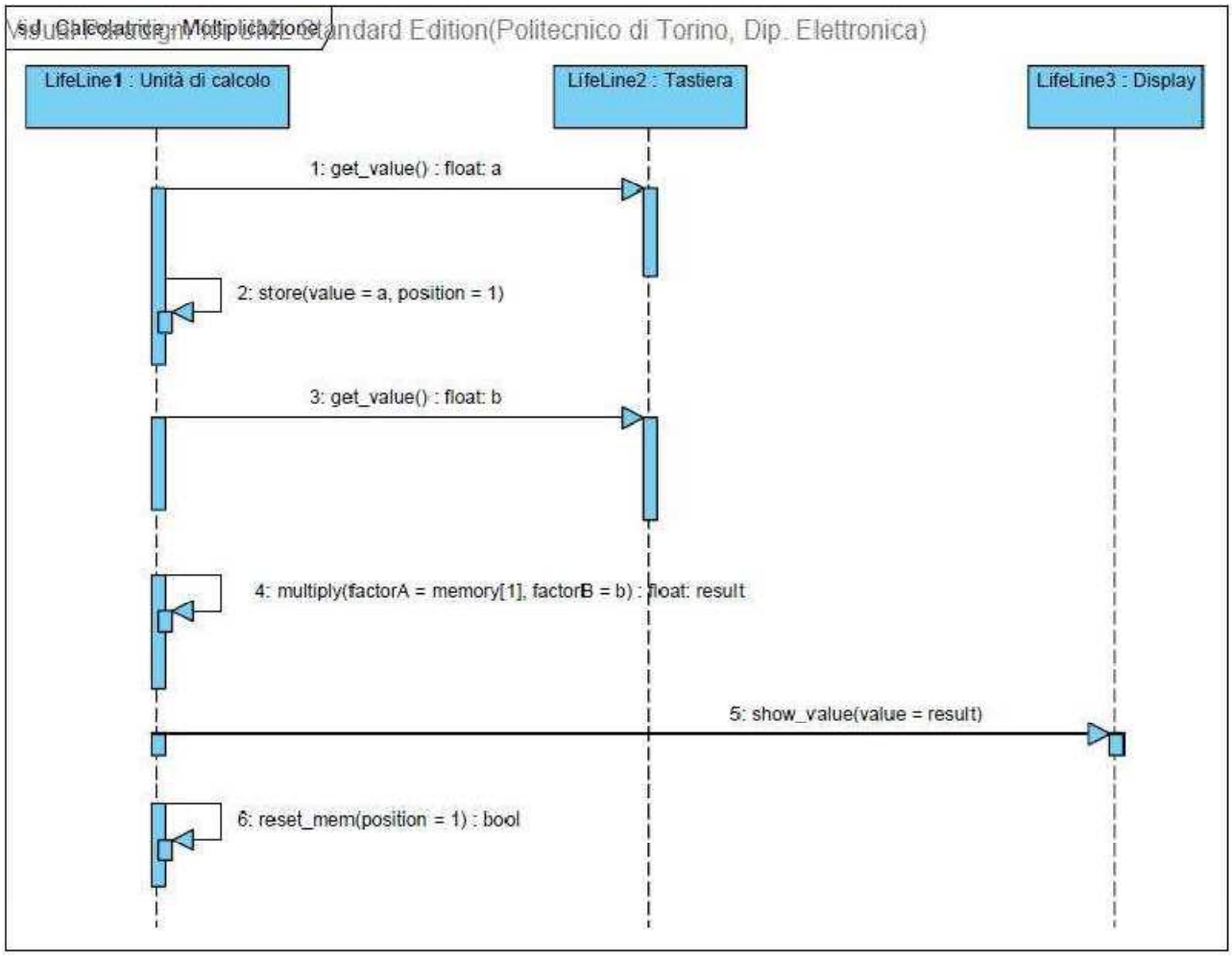


Figure 1.11: UML language -Sequence diagram example

Chapter II

FemtoSat Specifications, Actors and Use Cases

FemtoSat is a very small satellite (with a weight lower than 100g) designed to implement typical operations of bigger satellites with in addition some issues specific for the case study.

In particular, all use cases associated to FemtoSat are divided into the following sets:

- Intersatellite Communication ;
- Distance Measurement
- Distance Control ;

Each of them is shown in the following diagram and refers to several sub-use cases and sub-diagrams described in the chapter.

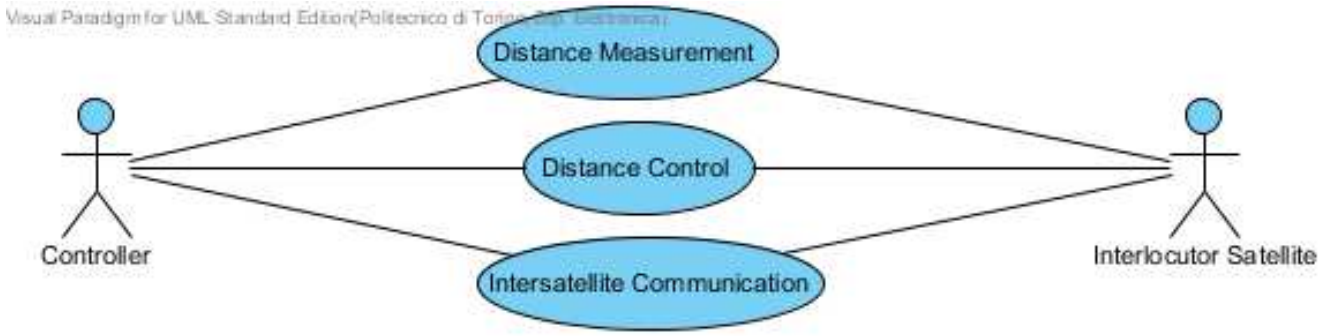


Figure 2.1 FemtoSat main use case diagram

2.1.1 Use Case - Distance Measurement

One of the main function of FemtoSat is the possibility to measure the distance with respect to another satellite (named Interlocutor Satellite). Two techniques are exploited:

- Radio Frequency;
- Magnetic;

The first one is based on RSSI technique and it is used for measurements where the satellites are far away and with a random orientation in the space. This technique cannot be applied if the distance between them is lower than distance threshold 1;

The second technique is suitable if the distance is not so high, (situation in which the magnetic field become small and hard to be measured). Magnetic technique cannot be applied if the distance is greater than distance threshold 2. Furthermore, the two satellites are supposed to be with the two faces opposite one other.

If it is not the case, one rotation shall be applied in order to align the two systems.

In the range of distances between distance threshold 1, distance threshold 2 both techniques can be applied.

Through Get Distance Global, the Controller combines both techniques to obtain a better estimation.

The distance measurement based on radiofrequency technique is implemented in 1B35_Intersatellite_Communication project.

2.1.2 Use Case - Distance Control

The distance between the two satellites can be also changed according to the Controller needs.

The distance can be measured through magnetic and radiofrequency techniques depending on the case, while the relative motion of the two system is implemented by using attractive/repulsive magnetic force generated by two Electromagnets.

Basing on the signal which drives the actuator, we can manage the contribution of Earth magnetic field and magnetic field generated by Interlocutor Satellite.

We can have the following situation:

- Magnetic_Actuator_Subsystem driven by a DC signal: only contribution of Earth magnetic field which produces rotation;
- Magnetic_Actuator_Subsystem s on both satellites driven by a square wave signal with a 50% duty cycle: only contribution of magnetic field generated by Interlocutor Satellite which produces attraction/repulsion.
- Magnetic_Actuator_Subsystem s driven by generic square wave signal: rotation and attraction/repulsion combined;

In the thesis only the second case is implemented, but the project can be easily extended to consider all previously described functions.

2.1.3 Use Case - Intersatellite Communication

Another important functionality of FemtoSat is the possibility to exchange messages with an Interlocutor Satellite.

As anticipated in the Introduction, this need is a common need for all projects in which a communication between two nodes (placed on the same satellite or on different satellites) is implemented.

These features are implemented in 1B35_Intersatellite_Communication module, for this reason all use cases are taken from 1B35_Intersatellite_Communication project, they are just presented in this chapter and fully described in the previous chapter.

2.2 FemtoSat TX/RX

The diagram contains all uses cases related to the communication between FemtoSat and the Interlocutor Satellite. Since the communication operations are performed by 1B35_Intersatellite_Communication module, most of use cases are specified in 1B35_Intersatellite_Communication and then re-used in this diagram.

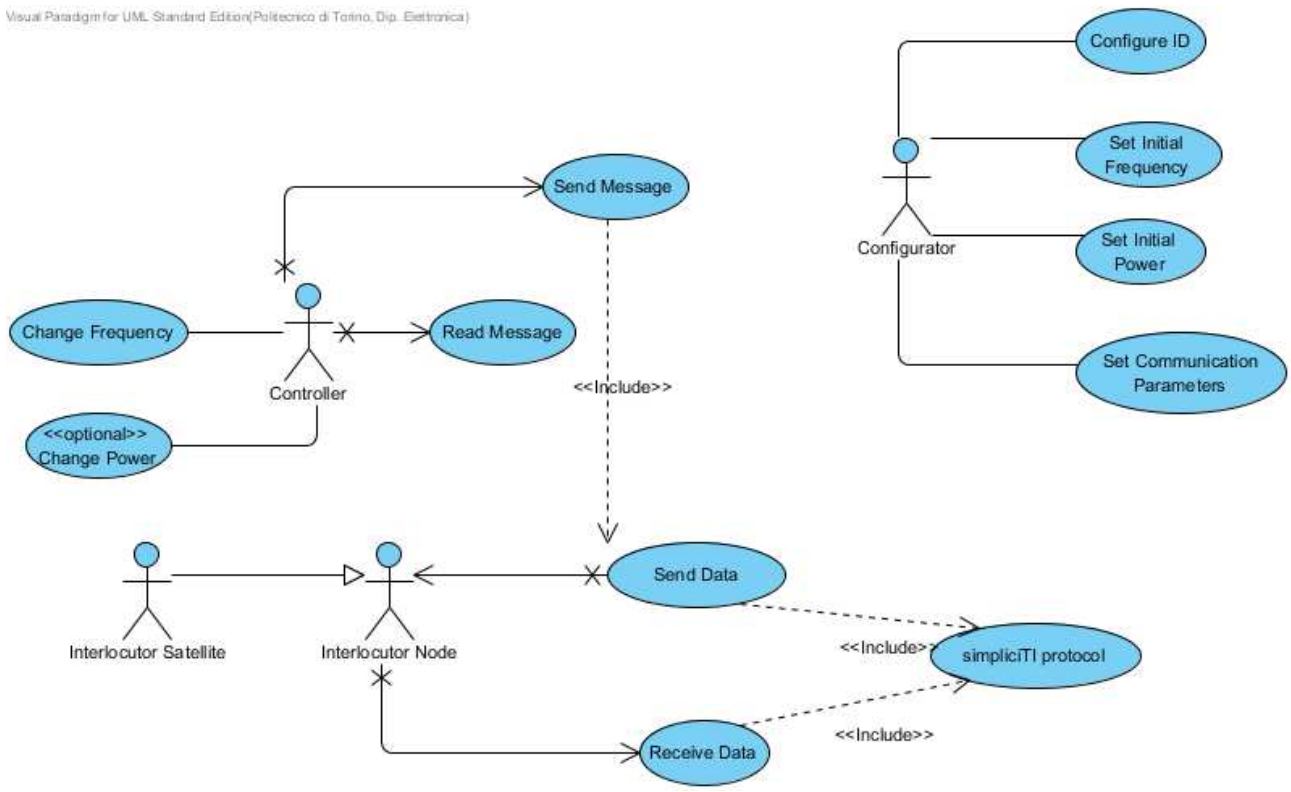


Figure 2.2 - FemtoSat TX/RX use case diagram

2.3 FemtoSat Get Distance RF

The diagram contains all uses cases related to the evaluation of the distance between the FemtoSat and Interlocutor Node by using radiofrequency techniques.

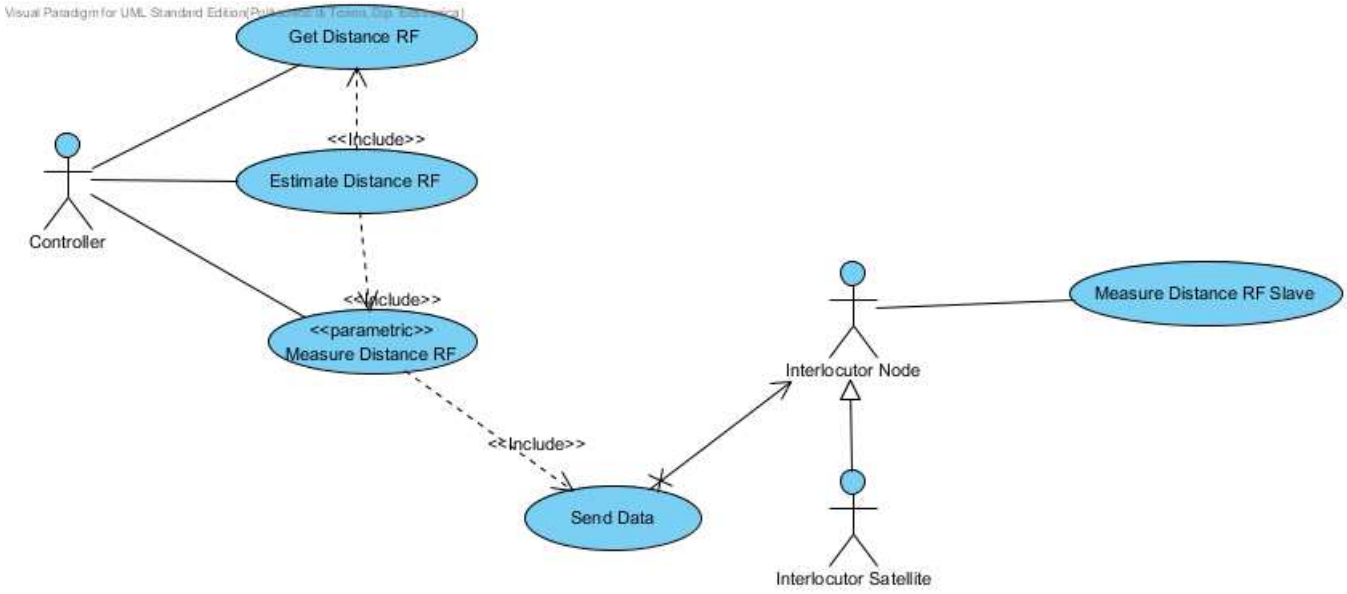


Figure 2.3 - Get Distance RF use case diagram

2.4 FemtoSat Get Distance MAG

The diagram contains all uses cases related to the evaluation of the distance between the FemtoSat and Interlocutor Node by using magnetic techniques.

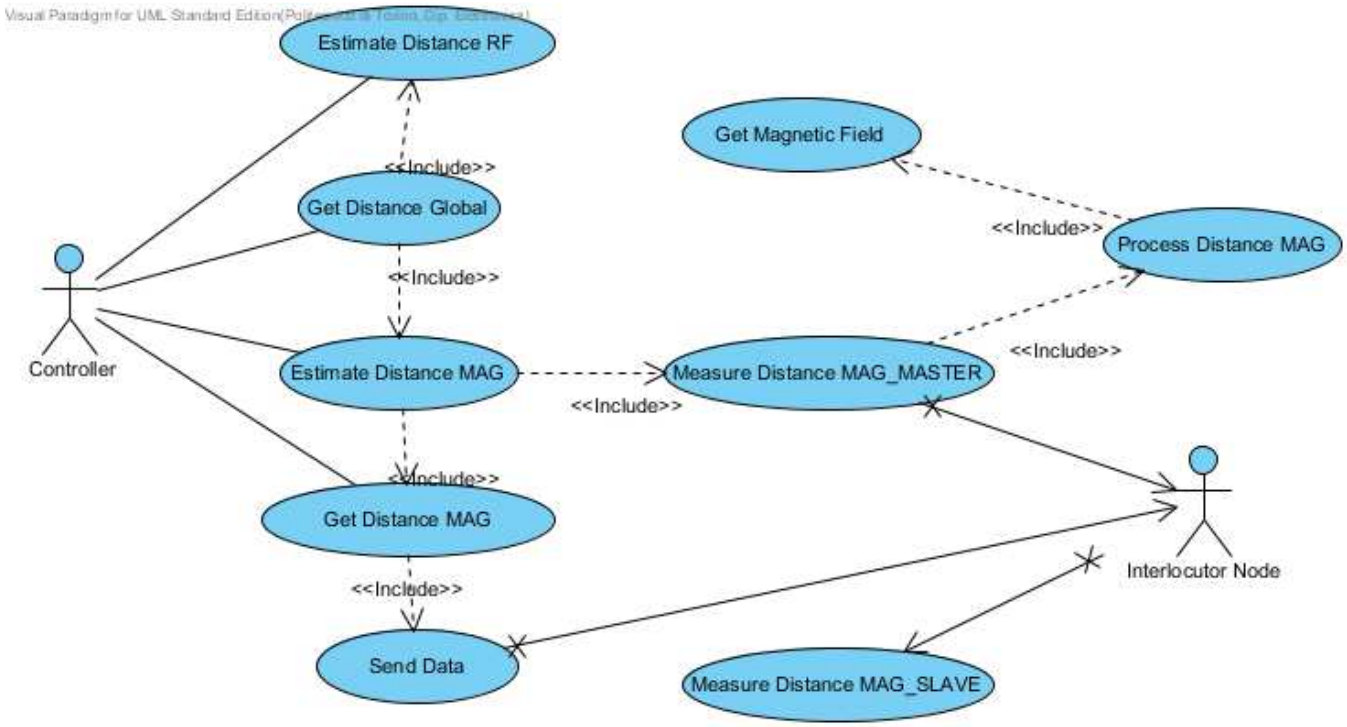


Figure 2.4 FemtoSat Get Distance MAF use case diagram

2.4.1 Use Case - Get Distance Global

The Controller combines Estimate Distance RF and Estimate Distance MAG to obtain a better evaluation of the distance between FemtoSat and Interlocutor Satellite.

First it uses Estimate Distance RF, if dist_RF > = distance_threshold_2 then it consider that value as better estimation so it stores the result in distance variable. If the two satellites are too far away magnetic techniques cannot be applied.

If dist_RF distance_threshold_2 then Controller uses also Estimate Distance MAG to obtain dist_MAG.

Finally an average between dist_RF and dist_MAG is stored in global distance variable.

If dist_RF distance threshold 1 the dist_MAG is stored in global distance variable because the dist_RF value is meaningless.

2.4.2 Use Case - Estimate Distance MAG

First the Controller uses Measure Distance MAG_MASTER to trigger the distance measurement to the right Interlocutor Node identified by an ID given by the actor, then starts a loop in which periodically the value of dist_MAG and its validity are evaluated by using Get Distance MAG .

When distValid is true, it indicates a valid measurement and causes the loop termination.

The Estimate Distance MAG is implemented by calling the estimateDistanceMAG function.

2.4.3 Use Case - Measure Distance MAG_MASTER

The Controller uses Send Message to send TURN_ON_MAGNET_FORWARD command to the Interlocutor Satellite identified by an ID given by the actor.

The Interlocutor Satellite then uses Turn ON Magnet Forward Mode to turn on its Electromagnet in forward mode.

The Controller uses Process Distance MAG to measures the magnetic field generated by the Interlocutor Satellite and process the value to obtain the distance dist_MAG.

Finally, to stop the procedure, the Controller uses Send Message to sends TURN_OFF_MAGNET command to Interlocutor Satellite which uses Turn OFF Magnet for turning off its Electromagnet.

The Measure Distance MAG_MASTER is implemented by calling the measureDistaceMAG function.

2.4.4 Use Case - Measure Distance MAG_SLAVE

The Controller using Read Message receive and interpret TURN_ON_MAGNET_FORWARD command sent by Interlocutor Satellite then uses Turn ON Magnet Forward Mode to turn on its Electromagnet in forward mode.

Finally, to stop the procedure, the Controller uses Turn OFF Magnet for turning off its Electromagnet after having received the corresponding command TURN_OFF_MAGNET from the Interlocutor Satellite.

2.4.5 Use Case - Get Distance MAG

The Controller reads the value of distValid, if true, it reads the other attributes of Distance_MAG.

The Get Distance MAG use case is implemented by calling the getDistanceMAG function.

distValid is reset at the beginning of the measurement and set to true at the end of measurement. All other attributes of Distance_MAG are updated at the end of measurement.

2.4.6 Use Case - Process Distance MAG

First the Controller uses Get Magnetic Field to obtain magX and magY values. Then the following formula [12] shall be manipulated to process those values and evaluate dist_MAG.

$$B(x) = \frac{\mu_0 N_{coil}}{2L} I \left[\frac{L + 2x}{\sqrt{(L + 2x)^2 + 4R^2}} + \frac{L - 2x}{\sqrt{(L - 2x)^2 + 4R^2}} \right] 10^4 [Gauss]$$

The validity of dist_MAG is indicated by distValid flag.

Process Distance MAG is implemented by using processDistanceMAG function.

2.4.7 Use Case - Get Magnetic Field

The Controller starts a sequence of operation necessary to measure the magnetic field by calling getField.

In order to have the correct measurement of the magnetic field, the Voffset shall be evaluated.

First the Controller uses Set Magnetic Sensor HMC1002 after which Vset is read, then it uses Reset Magnetic Sensor HMC1002 after which Vreset is read.

The Voffset is obtained as average of the last measured voltages i.e.:

Voffset = (Vset+Vreset)/2

Finally the components of magnetic field (written in the magX, magY parameters) are calculated as follows:

magX [G]= (MAGN_X-OFFSET_MAGNETIC)[V]/SENS_MAGNETIC[V/T]*10^4;

magY [G]= (MAGN_Y-OFFSET_MAGNETIC)[V]/SENS_MAGNETIC[V/T]*10^4;

2.5 FemtoSat Set Distance

The diagram contains all uses cases related to the control of the distance between the FemtoSat and Interlocutor Node.

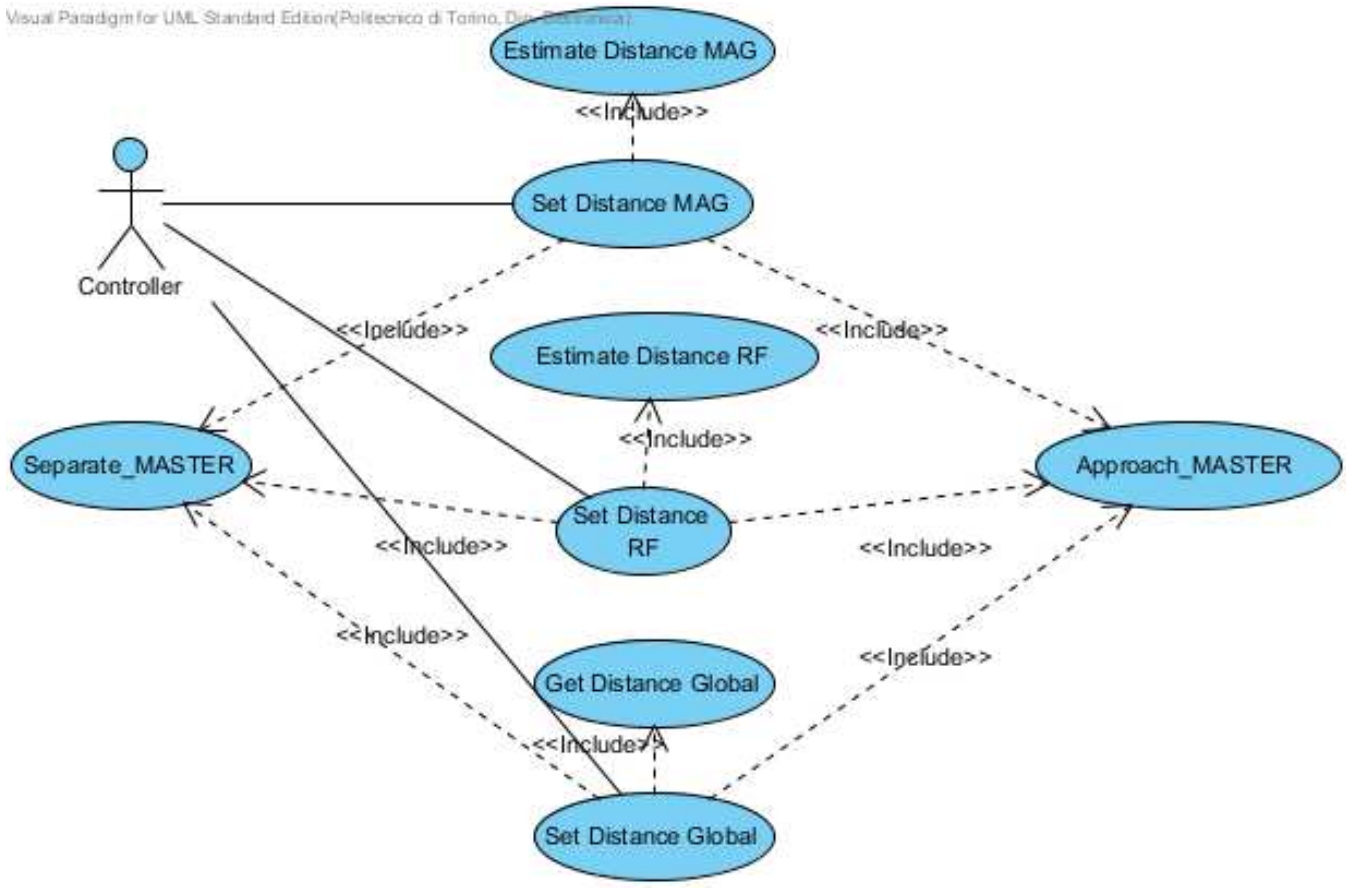


Figure 2.5 - FemtoSat Set Distance use case diagram

2.5.1 Use Case - Set Distance Global

The Controller will perform all needed operations to set the distance between FemtoSat and Interlocutor Satellite to distance_new value.

First it uses Get Distance Global use case to estimate the actual distance (stored in distance variable).

If distance > distance_new, the two satellites shall get closer, the Approach_MASTER use case is then used by Controller.

If distance distance_new, the two satellites shall separate, the Separate_MASTER use case is then used by Controller.

After an Approach_MASTER/Separate_MASTER cycle there is again a Get Distance Global cycle to measure the updated distance; the algorithm is executed in a loop until the measured distance is close enough to distance_new (distance - distance_new distance_error).

The Set Distance Global is implemented by calling setDistanceGlobal function.

2.5.2 Use Case - Set Distance MAG

The Controller will perform all needed operations to set the distance between FemtoSat and Interlocutor Satellite to distance_new value.

First it uses Estimate Distance MAG use case to estimate the actual distance (stored in dist_MAG variable).

If dist_MAG > distance_new, the two satellites shall get closer, the Approach_MASTER use case is then used by Controller.

If dist_MAG distance_new, the two satellites shall separate, the Separate_MASTER use case is then used by Controller.

After an Approach_MASTER/Separate_MASTER cycle there is again a Estimate Distance MAG cycle to measure the updated distance; the algorithm is executed in a loop until the measured distance is close enough to distance_new (dist_MAG - distance_new distance_error).

The Set Distance MAG is implemented by calling setDistanceMAG function.

2.5.3 Use Case - Set Distance RF

The Controller will perform all needed operations to set the distance between FemtoSat and Interlocutor Satellite to distance_new value.

First it uses Estimate Distance MAG use case to estimate the actual distance (stored in dist_RF variable).

If dist_RF > distance_new, the two satellites shall get closer, the Approach_MASTER use case is then used by Controller.

If dist_RF distance_new, the two satellites shall separate, the Separate_MASTER use case is then used by Controller.

After an Approach_MASTER/Separate_MASTER cycle there is again a Estimate Distance RF cycle to measure the updated distance; the algorithm is executed in a loop until the measured distance is close enough to distance_new (dist_RF - distance_new distance_error).

The Set Distance RF is implemented by calling setDistanceRF function.

2.5.4 Use Case - Approach_MASTER

First the Controller uses Set Local Temporal Reference, Set Remote Temporal Reference (related to the Interlocutor Satellite identified by an ID given by the actor) passing timeReference value to provide a time reference.

The Controller uses Send Message to send TURN_ON_MAGNET_AC command and 180 (phase),0.5 (duty cycle) as parameters to the Interlocutor Satellite.

The former uses Turn ON Magnet_AC with a duty cycle of 0.5 and a phase of 0 while the latter uses Turn ON Magnet_AC with a duty cycle of 0.5 and a phase of 180 to turn on their Electromagnets in a synchronous way with opposite phase.

Both Electromagnets stay on for a certain amount of time (Ton_long) so that the two satellites approach each other thanks to the attractive magnetic force. After Ton_long time period both the Controller and Interlocutor Satellite uses Turn OFF Magnet to turn off the Electromagnets.

The two satellites shall stop, so a break cycle is executed i.e. they both use Turn ON Magnet_AC with a duty cycle of 0.5 and a phase of 0 for a Ton_long time period after which the Turn OFF Magnet is used.

The use case is implemented by calling the approach function.

2.5.6 Use Case - Separate_MASTER

First the Controller uses Set Local Temporal Reference, Set Remote Temporal Reference (related to the Interlocutor Satellite identified by an ID given by the actor)passing timeReference value to provide a time reference.

The Controller uses Send Message to send TURN_ON_MAGNET_AC command and 0(phase),0.5(duty cycle) as parameters to the Interlocutor Satellite.

Both the Controller and Interlocutor Satellite use Turn ON Magnet_AC with a duty cycle of 0.5 and a phase of 0 to turn on their Electromagnets in a synchronous way with same phase.

The Electromagnets stay on for a certain amount of time (Ton_long) so that the two satellites separate each other thanks to the repulsive magnetic force. After Ton_long time period both the Controller and Interlocutor Satellite uses Turn OFF Magnet to turn off the Electromagnets.

The two satellites shall stop, so a break cycle is executed i.e. the Interlocutor Satellite uses Turn ON Magnet_ACfor Ton_long time period with a duty cycle of 0.5 and a phase of 180, while the Controller uses Turn ON Magnet_AC for the same time period with a duty cycle of 0.5 and a phase of 0.

Finally, the Turn OFF Magnet is used.

The use case is implemented by calling the separate function.

2.6 FemtoSat Magnetometer Subsystem

All use cases related to the Magnetometer_Subsystem.

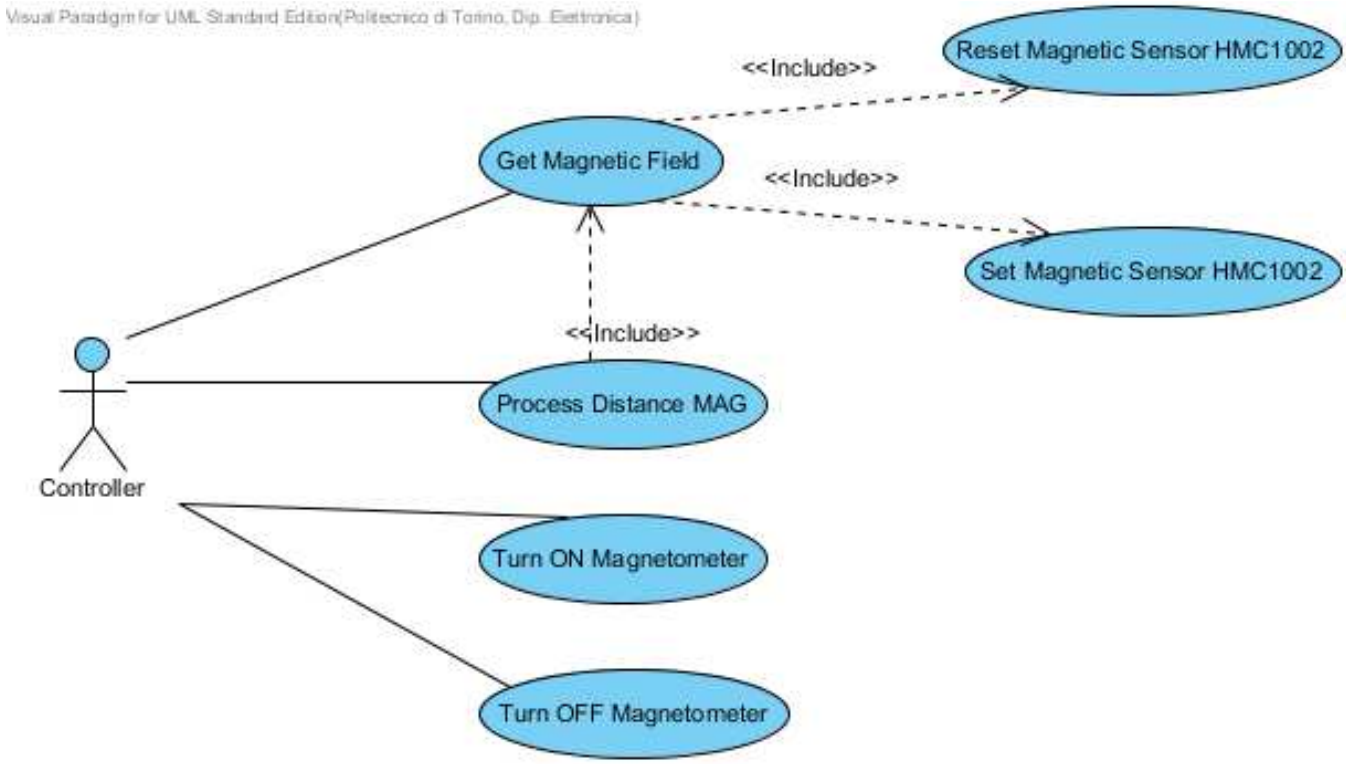


Figure 2.6 - FemtoSat Magnetometer Subsystem

2.6.1 Use Case - Turn OFF Magnetometer

The Controller turns off the Magnetometer_Subsystem by calling turnOFF function of Magnetometer_Sensor_SW_Driver.

2.6.2 Use Case - Turn ON Magnetometer

The Controller turns on the Magnetometer_Subsystem by calling turnON function of Magnetometer_Sensor_SW_Driver.

2.6.3 Use Case - Set Magnetic Sensor HMC1002

The Bk1B221_Magnetometer_Sensor autonomously starts a set procedure for Magnetometer_2_axis_HMC1002 by calling set function.

This operation is necessary for the correct usage of Magnetometer_2_axis_HMC1002 (due to its physical characteristics) and it consists of an impulsive current flowing through the sensor for Tset_reset in a specific direction.

2.6.4 Use Case - Reset Magnetic Sensor HMC1002

The Bk1B221_Magnetometer_Sensor autonomously starts a reset procedure for Magnetometer_2_axis_HMC1002 by calling resetfunction.

This operation is necessary for the correct usage of Magnetometer_2_axis_HMC1002 (due to its physical characteristics) and it consists of an impulsive current flowing through the sensor for Tset_reset in a specific direction.

2.7 FemtoSat Magnetic Actuator

All use cases related to the Magnetic_Actuator_Subsystem.

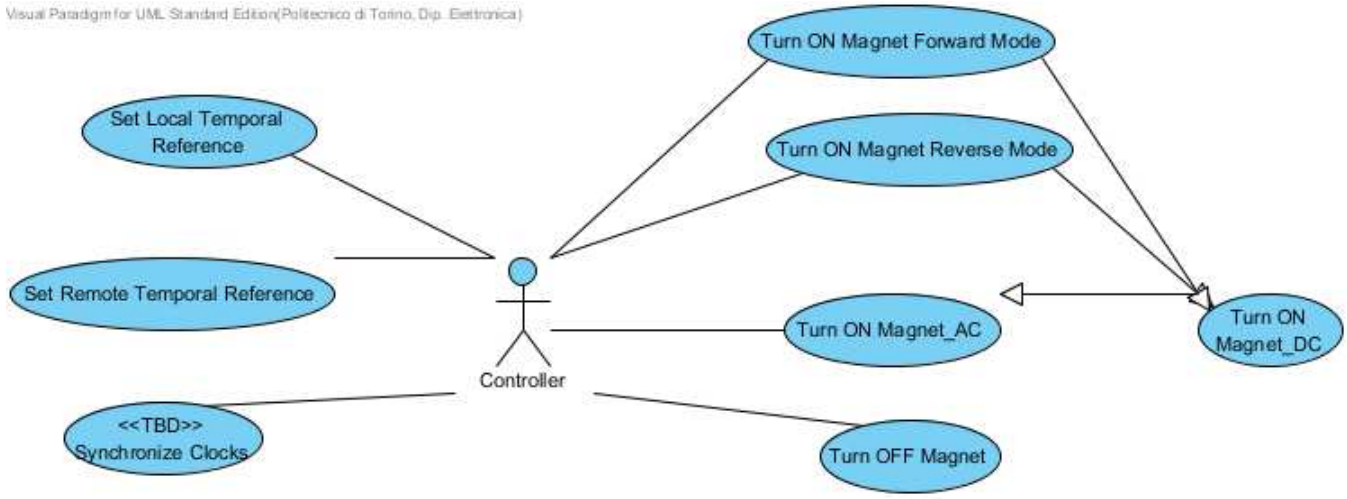


Figure 2.7 - FemtoSat Magneic Actuator use case diagram

2.7.1 Use Case - Set Remote Temporal Reference

The Controller uses Send Message to send SET_TIME_REFERENCE command and timeReference as parameter to the Interlocutor Satellite identified by an ID given by the actor.

The Interlocutor Satellite then uses its own Set Local Temporal Reference passing as parameter the received value.

The Set Remote Temporal Reference is implemented by calling setRemoteTimeReference function.

2.7.2 Use Case - Set Local Temporal Reference

The Controller creates the reference signal which is used in the motion operations of FemtoSat i.e.:

- Separate_MASTER
- Approach_MASTER
-

The reference is a periodic signal having a time period specified in timeReference and configurable with setTimeReference function.

It shall have a period not smaller than CC2510 clock.

2.7.3 Use Case - Turn ON Magnet_AC

The Controller turns on the Magnetic_Actuator_Subsystem driving it with an AC current having a duty cycle and phase reference specified by the actor. The use case is implemented by calling turnON and passing the desired duty cycle and phase reference as parameter.

2.7.4 Use Case - Turn ON Magnet_DC

It is a particular case of Turn ON Magnet_AC in which the duty cycle passed as parameter is 1.

The Controller turns on the Magnetic_Actuator_Subsystem driving it with a continuous current. The use case can be Turn ON Magnet Forward Mode, Turn ON Magnet Reverse Mode depending on the direction of the current.

2.7.5 Use Case - Turn ON Magnet Forward Mode

The Controller turns on the Magnetic_Actuator_Subsystem in forward mode by calling in sequence setForwardMode and turnON functions of Magnetic_Actuator_SW_Driver class.

2.7.6 Use Case - Turn ON Magnet Reverse Mode

The Controller turns on the Magnetic_Actuator_Subsystem in reverse mode by calling in sequence setReverseMode and turnON functions of Magnetic_Actuator_SW_Driver class.

2.7.7 Use Case - Turn OFF Magnet

The Controller turns off the Magnetic_Actuator_Subsystem by calling turnOFF function of Magnetic_Actuator_SW_Driver class.

2.8 FemtoSat Motion_MASTER

The diagram contains all uses cases related to the motion of the FemtoSat and Interlocutor Node needed to set a proper value of distance.

Here the FemtoSat is the master (it starts the use cases) while Interlocutor Node is the slave (it collaborates with the FemtoSat to complete the use cases).

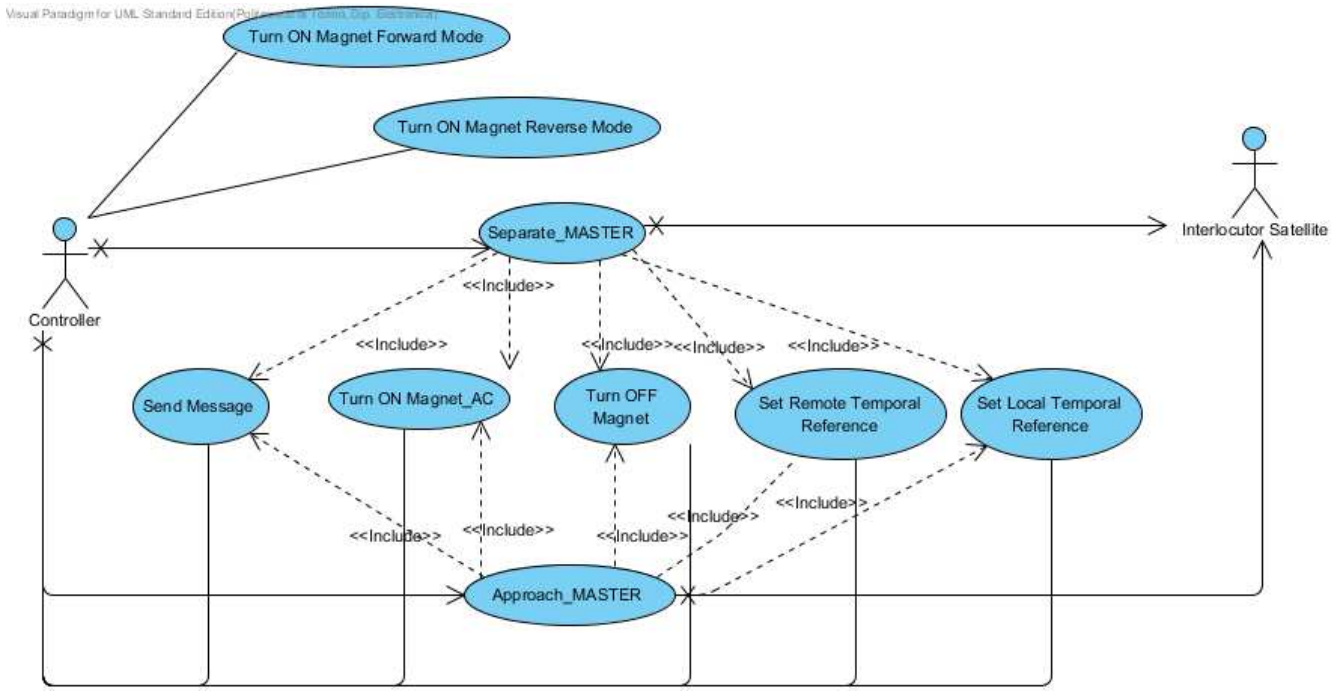


Figure 2.8 - FemtoSat Motion_MASTER

2.9 FemtoSat Motion_SLAVE

The diagram contains all uses cases related to the motion of the FemtoSat and Interlocutor Node needed to set a proper value of distance.

Here the Interlocutor Node is the master (it starts the use cases) while FemtoSat is the slave (it collaborates with the Interlocutor Node to complete the use cases).

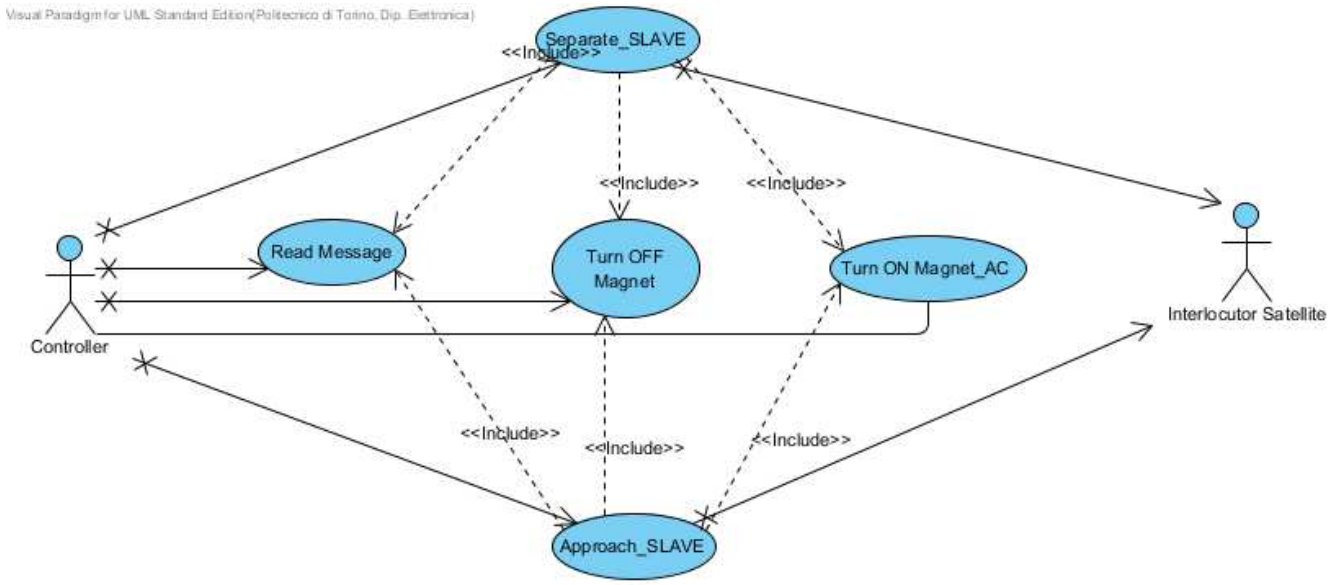


Figure 2.9 - FemtoSat motion_SLAVE use case diagram

2.9.1 Use Case - Approach_SLAVE

The Controller using Read Message receive and interpret SET_TIME_REFERENCE which triggers

Set Local Temporal Reference use case.

Then the Controller, using again Read Message receive and interpret TURN_ON_MAGNET_AC command and 0.5,180 as parameters (duty cycle and phase displacement respectively) sent by Interlocutor Satellite then uses Turn ON Magnet_AC accordingly.

Meantime the Interlocutor Satellite uses Turn ON Magnet_AC with 0.5 as duty cycle and 0 as phase reference.

The two Electromagnets stay on for a certain amount of time (Ton_long) so that the two satellites approach each other thanks to the attractive magnetic force. After Ton_long time period both the Controller and Interlocutor Satellite use Turn OFF Magnet to turn off the Electromagnets.

The two satellites shall stop, so a break cycle is executed i.e. they both use Turn ON Magnet_AC with 0.5 as duty cycle and 180 as phase reference for a Ton_long time period after which the Turn OFF Magnet is used.

2.9.2 Use Case - Separate_SLAVE

The Controller using Read Message receive and interpret SET_TIME_REFERENCE which triggers

Set Local Temporal Reference use case.

Then the Controller, using again Read Message receive and interpret TURN_ON_MAGNET_AC command and 0,0.5 as parameters (phase displacement and duty cycle respectively) sent by Interlocutor Satellite then uses Turn ON Magnet_AC accordingly.

Meantime the Interlocutor Satellite uses Turn ON Magnet_AC with 0.5 as duty cycle and 0 as phase reference.

The two Electromagnets stay on for a certain amount of time (Ton_long) so that the two satellites separate each other thanks to the repulsive magnetic force. After Ton_long time period both the Controller and Interlocutor Satellite use Turn OFF Magnet to turn off the Electromagnets.

The two satellites shall stop, so a break cycle is executed i.e. the Interlocutor Satellite uses Turn ON Magnet_AC with 0.5 as duty cycle and 0 as phase reference for Ton_long time period, while the Controller uses Turn ON Magnet_AC with 0.5 as duty cycle and 180 as phase reference for the same time period.

Finally, the Turn OFF Magnet is used.

Chapter III

FemtoSat Design and Analysis

As stated in previous paragraphs, FemtoSat has a modular structure following AraMis philosophy in which both pre-designed blocks and blocks developed customized for the application are used together.

In particular we can logically divide the whole system into the following units:

- 1B35_Intersatellite_Communication
- Distance Controller
  - Magnetometer_Subsystem
  - Magnetic_Actuator_Subsystem
- Supply Subsystem

The actual implementation of the system is divided into hardware and software:

- FemtoSat_HW
- FemtoSat_SW

The root classes contains all hardware and software classes which are described into corresponding diagrams in the following paragraphs.

1B351_Intersatellite_Communication hardware class and 1B351S_Intersatellite_Communication software class with all subclasses are taken from 1B35_Intersatellite_Communication project [6].

.

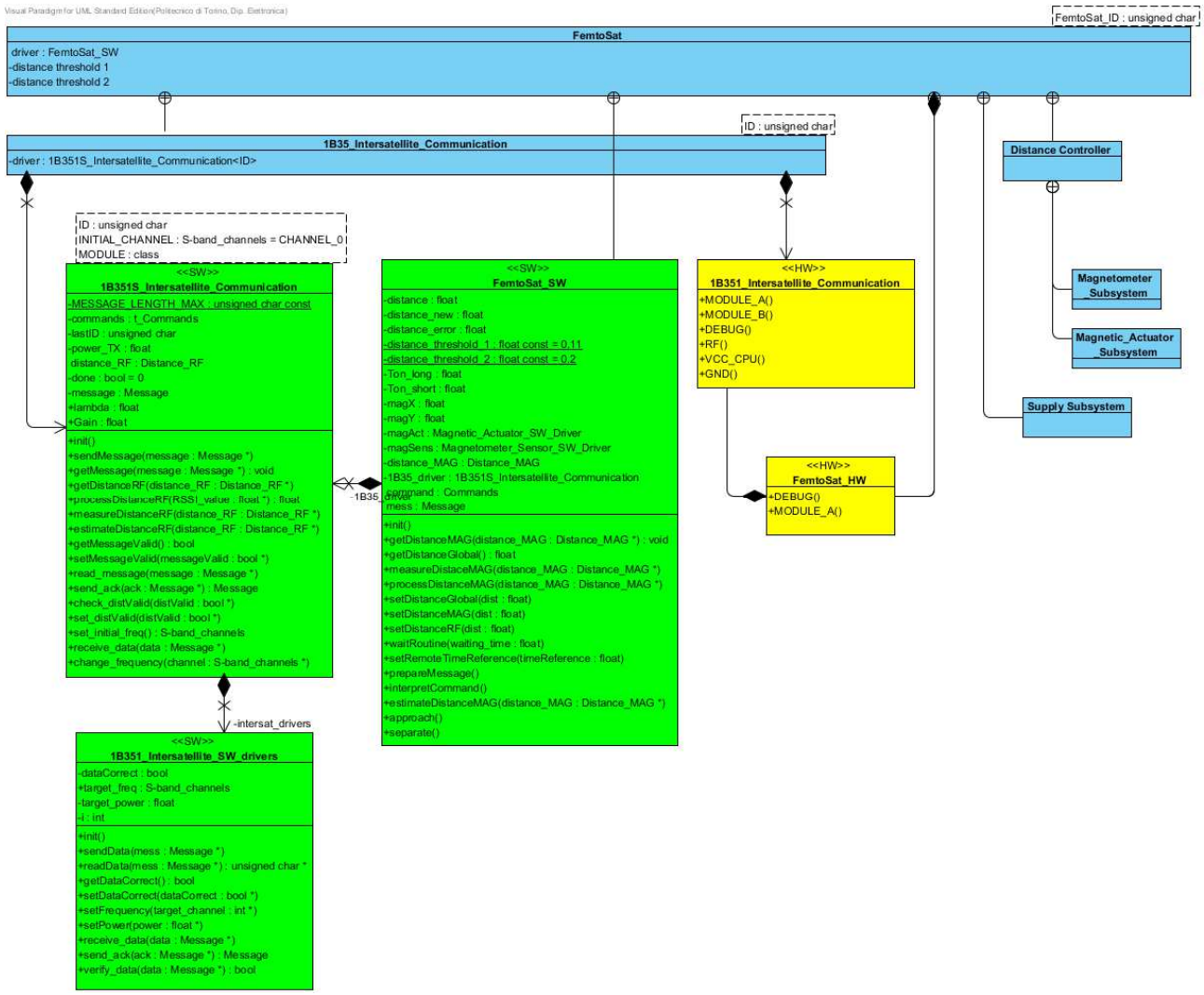


Figure 3.1 - FemtoSat main class diagram

3.1 Design choices

We can start the design procedure by fixing the available power on-board the satellite. The PCB dimensions are not specified so one can considered an quasi-arbitrary case as starting reference: 9X9 cm.

3.1.1 Source Power

Regardless the actual implementation of supply system, we can assume a reasonable solution adopting solar cells at least for a rough estimation of the power available in the system.

We consider triple junction GaAs solar cells provided by AzurSpace (illustrated in *Figure 3.2*) which have the following feature:

- Model: TJ Solar Cell 3G28C;
- Efficiency (η = 28%);
- Vmax = 2.87 V;
- Standard dimensions = 80 mm X 40 mm;

Figure 3.2- TJ Solar Cell 3G28C.

In a 9X9 cm PCB we can use two cells having standard dimensions obtaining a total cells surface of 60.35 cm^2 (*Figure 3.3*). The power can be estimated by the following formula:

$$P_{max} = K_s * Sup * \eta * \eta_s$$

where Ks is the solar constant (1366 W/m^2), Sup is the solar cells surface, η is the solar cells efficiency and η_s is the efficiency of regulation circuits (we can assume an efficiency of 90%).

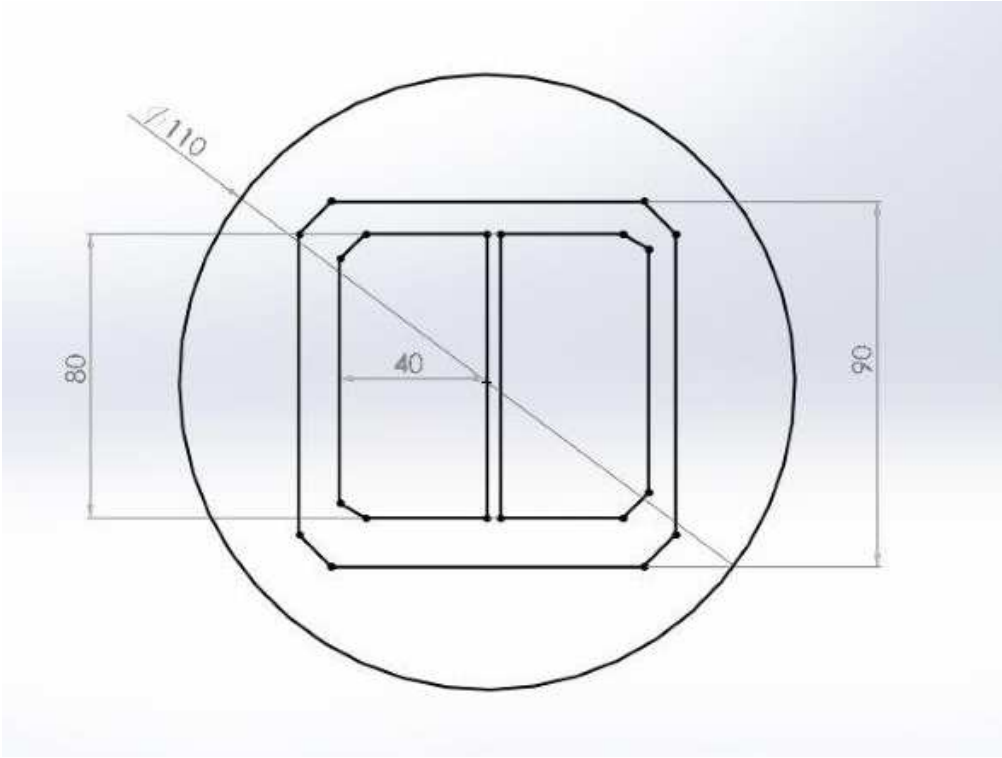


Figure 3.3 - Top sight of the system under design.

The maximum power available in the femtosatellite is then estimated as P_{max} = 2.077 W ≈ 2 W with a voltage supply of about 6 V (if we connect in series two solar cells).

Notice that the actual voltage supply depends on the choice of regulation circuit, anyway the values just derived are a good starting point for the design and analysis procedures.

3.1.2 Magnetometer and Magnetic Actuator

The magnetometer is a module based on 2 axis HMC1002 sensor , with all signal conditioning circuits to provide a magnetic field measurement. To exploit modularity and re-usability which are the main goals of AraMis project, we choose to adopt the magnetometer used in AraMis, the Bk1B221_Magnetometer_Sensor. The module provide a measurement of the magnetic field along X,Y axis expressed in Tesla. More details are shown in *Chapter V.*

The electronic module used to drive the electromagnet is chosen among the modules available in the AraMis repository, that is Bk1B222_Magnetic_Torque_Actuator.

The power dissipated by *Bk1B222_Magnetic_Torque_Actuator* is not fixed a priori by the electronic module, but depends on the voltage supply and on the equivalent resistance made of solenoid and other resistances as showed in the prosecution of this paragraph.

Let's consider the solenoid as an inductive passive load having two connectors through which it is connected to the circuit. This circuit (called **driver**) has to drive the load such that the current can flow in both directions (from connector 1 to connector 2 and viceversa).

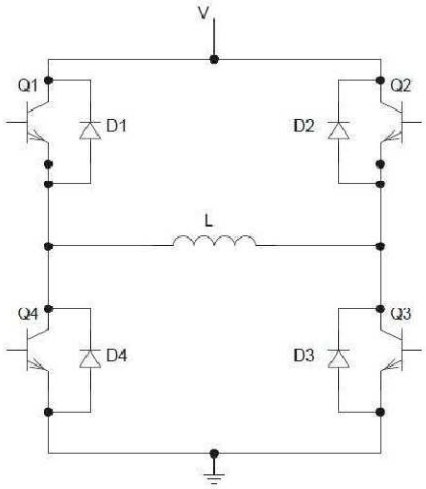


Figure 3.4 - Schematic of an H bridge circuit

The driver will have a typical configuration called **H bridge** (*Figure 3.4)* which allows to drive the solenoid in bidirectional way (thanks to the bridge topology) and discharge the stored current through wheeling diodes.

In *Figure 3.5* (on the left) we can see how, activating transistor Q1 and Q3, the current flow from left to the right in the solenoid while, when Q2 and Q4 are closed, it flows in the opposite direction (*Figure 3.5* on the right).

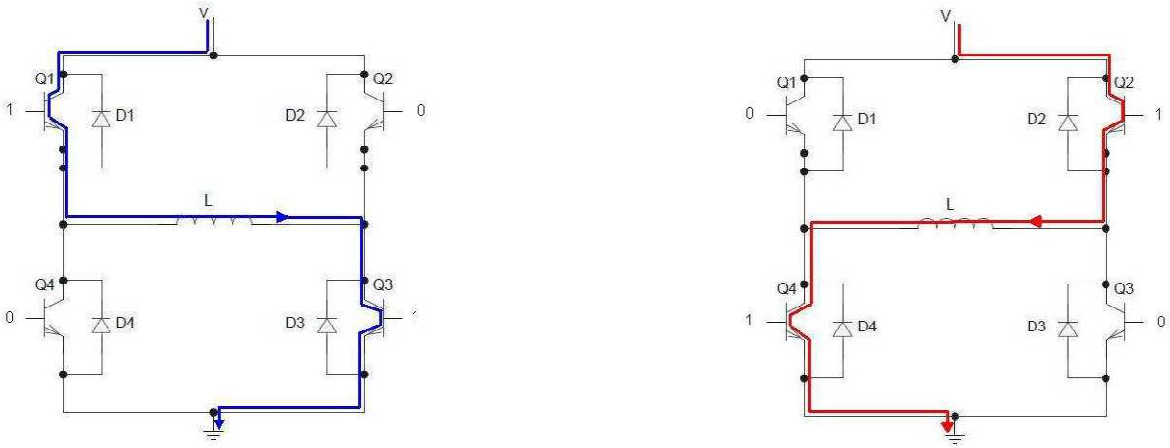


Figure 3.5 - Current paths in a H-bridge

In *Figure 3.6* is shown the path of the current through the wheeling diodes D2 and D4 once all transistors are interdicted, in case the solenoid was previously driven by transistors Q1 and Q3

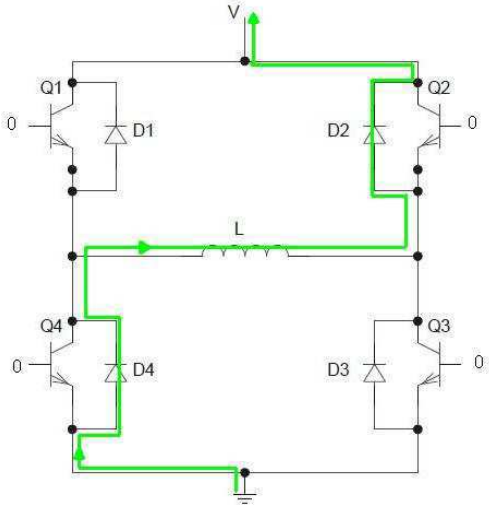


Figure 3.6 - Current flow through wheeling diodes.

On the other hands the current would flow in D1 and D3 if the solenoid had been charged through Q2 and Q4.

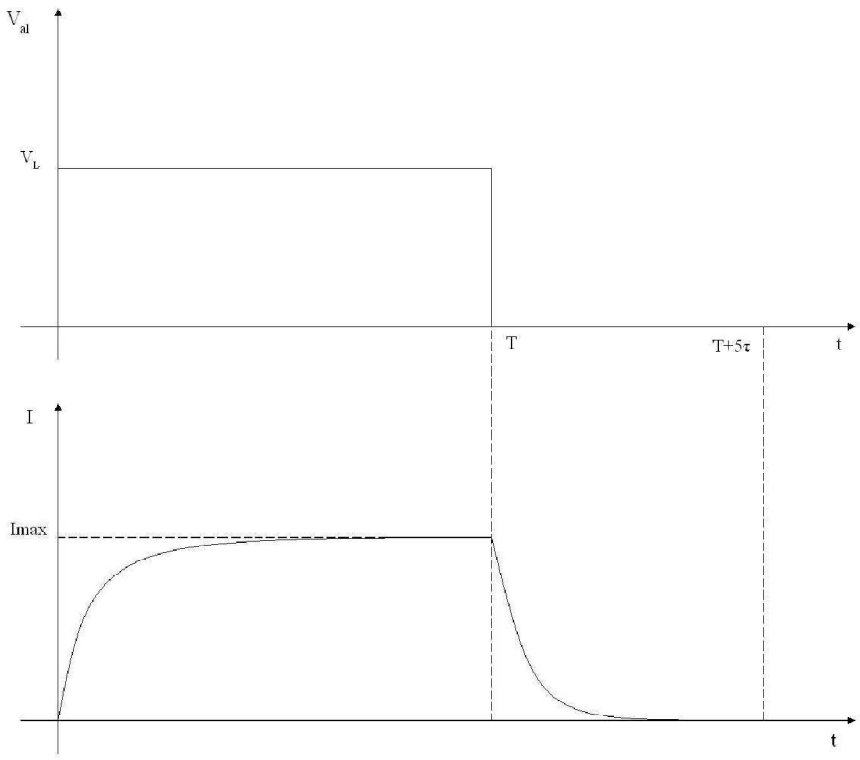


Figure 3.7 - Solenoid charge at constant voltage and discharge

The current and voltage trends are shown in *Figure 3.7* where T is the time in which the solenoid is driven (charged), following a discharge period assumed to be 5τ.

In *Figure 3.8* is showed the equivalent circuit of the H-bridge topology, for t<T, where Rdson and Rsol are respectively the resistive contributions of each MOS in conduction and of the solenoid.

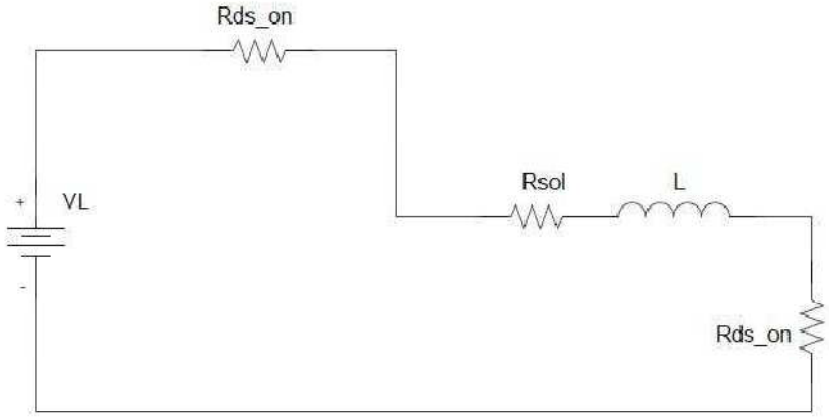


Figure 3.8 - Equivalent circuit of a H-bridge topology driving an inductor

If we call Rtot the equivalent series resistance equals to 2Rdson+Rsol, the circuit becomes the one illustrated in *Figure 3.9*

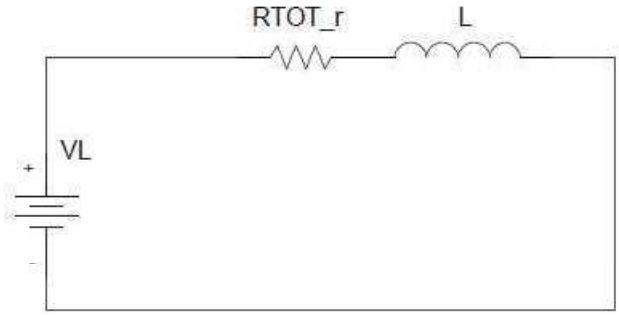


Figure 3.9 - Simplified equivalent circuit of a H-bridge topology driving an inductor

The expression of the current for t<T (graphically shown in *Figure 3.7*) will be :

$$I(t \leq T) = I_{max} - I_{max} exp\left(\frac{-t}{\tau_r}\right)$$

where $I_{max} = V_L / R_{TOT}$ is the maximum current flowing in that circuit and $\tau_r = L/R_{TOT}$ is the time constant related to the inductor charging. Now we can evaluate the average current flowing inside the solenoid in the time interval [0,T] by integrating from time 0 to time T and dividing by T:

$$\bar{I}_{rise} = \frac{1}{T}\int_0^T I(t)dt = I_{max} - \frac{I_{max}\left[-\tau_r exp\left(^{-T}/_{\tau_r}\right) + \tau_r\right]}{T}$$

The equivalent circuit for the discharge (t>T) is similar to the one related to the charge but the equivalent resistance will be $R_{TOT} = 2R_D + R_{sol}$, where R_D is the resistance of a diode in conduction. The expression of the current for t>T is:

$$I(t > T) = I_{max} exp\left(\frac{-t}{\tau_f}\right)$$

then we have:

$$\bar{I}_{fall} = \frac{1}{T}\int_0^{5\tau_f} I(t > T)dt = \frac{\left[I_{max} - I_{max} exp\left(^{-T}/_{\tau_f}\right)\right]\left[-\tau_f \exp(-5) + \tau_f\right]}{T}$$

Having that τ_r and τ_f are negligible, the average currents which the solenoid has to sustain are:

$$\bar{I}_{rise} \cong I_{max} = \frac{V_L}{R_{TOT}}$$

$$\bar{I}_{fall} \cong 0$$

In the following paragraph there is the analysis carried out to choose the solenoid.

3.1.3 Solenoid Choice

In order to choose the solenoid parameters different simulations are carried out by using some Matlab routines available in *Appendix 1*.

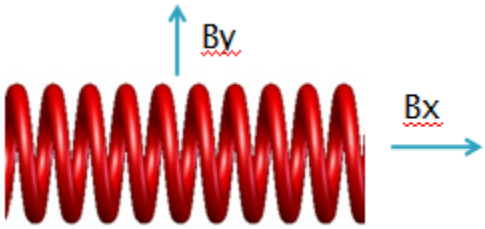


Figure 3.10 - Solenoid representation

Since there are many variables at stake we need to fix some of them and evaluate the other ones, in particular we fixed:

- Solenoid orientation (see *Figure 3.10*)
- Source Power : 2W
- Solenoid Radius : 5.5cm
- Wire diameter : 0.15mm

The elements that we can change in the simulations are

- Number of coils
- Number of solenoids
- Type of connection (series/parallel, see *Figure 3.11* for more details)

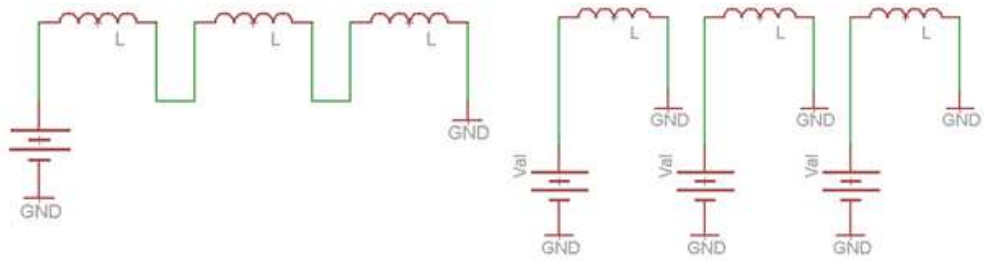


Figure 3.11 - Series connection on the left, parallel connection on the right

The formula for the magnetic field evaluation is:

$$B(x) = \frac{\mu_0 N_{coil}}{2L} I \left[\frac{L + 2x}{(L + 2x)^2 + 4R^2} + \frac{L - 2x}{\sqrt{(L - 2x)^2 + 4R^2}} \right] 10^4 \quad [Gauss]$$

where B is the magnetic field value expressed in Gauss, N is the number of coils, I the current flowing in the solenoid, L the length of the solenoid, R the radius of the solenoid, X the distance from the center of the solenoid.

Since we have to evaluate the distance between the two satellites, in the B function we need to pass as parameter the distance value adding an offset X_0 which consider the dimensions of the solenoid and thickness of HMC1002 sensor as illustrated in *Figure 3.12.* X_0 is then calculated as 2a-b (refer to *Figure 3.12* for the notations).

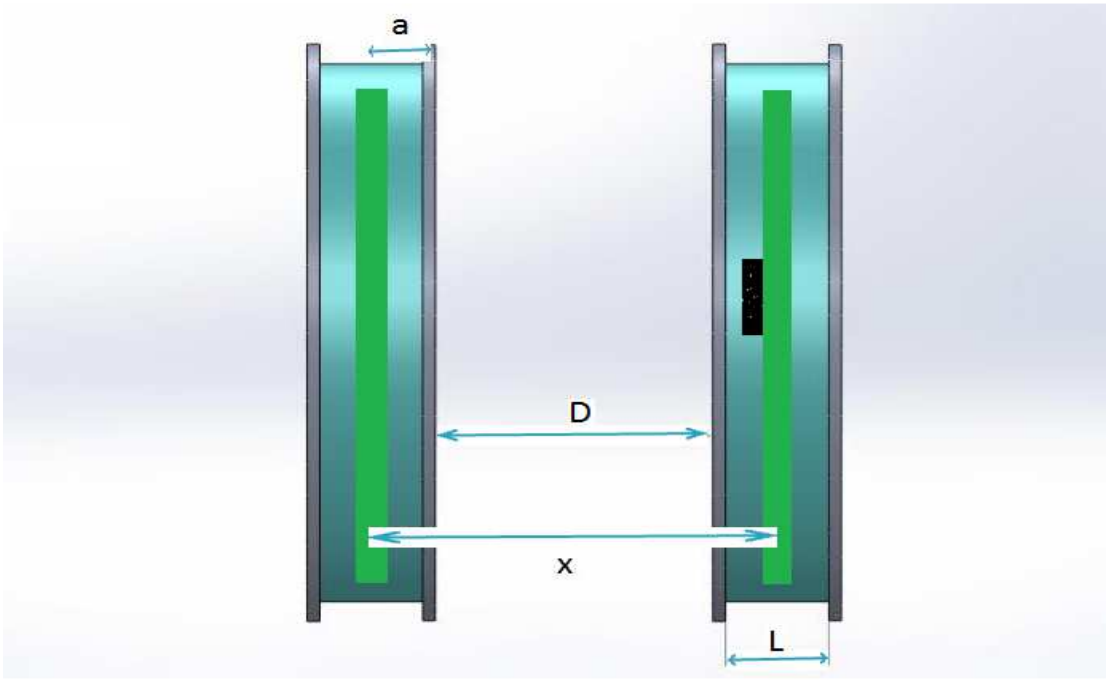


Figure 3.12 - System configuration, X is the distance between the two solenoids center, d is the distance between the two satellites, L is the satellite lentgh, b is the thickness of HMC1002 sensor

Varying the free elements described before, for each simulation we evaluate:

- B(dmin): magnetic field at dmin=1cm;
- B(dmax):magnetic field at dmax=20cm;
- Voltage supply (having fixed the power supply);
- Current flowing in one solenoid;
- Solenoid length;

Finally, the results are shown in *Table 3.1*

Configuration		*Geometrical Parameters*				*Electrical Parameters*			*Magnetic Parameters*	
Number of Solenoids	Connection	Wire diameter [m]	Radius of Solenoid [m]	Ncoils [per solenoid]	Length of a single Solenoid[m]	Power Source [W]	Voltage [V]	Current [A] (flowing in one solenoid)	B|d=dmin=1cm [G]	B|d=dmax=20cm [G]
1		1,50E-04	0,055	50	0,008	2	6,01	0,333	1,705	0,033
1		1,50E-04	0,055	100	0,015	2	8,357	0,239	2,163	0,043
1		1,50E-04	0,055	150	0,023	2	10,176	0,197	2,283	0,048
1		1,50E-04	0,055	200	0,03	2	11,716	0,171	2,223	0,051
1		1,50E-04	0,055	300	0,045	2	14,307	0,14	1,918	0,053
2	Series	1,50E-04	0,055	50	0,008	2	8,357	0,239	2,163	0,043
2	Series	1,50E-04	0,055	100	0,015	2	11,716	0,171	2,223	0,051
2	Series	1,50E-04	0,055	150	0,023	2	14,307	0,14	1,918	0,053
2	Parallel	1,50E-04	0,055	50	0,008	2	4,388	0,228	2,059	0,041
2	Parallel	1,50E-04	0,055	100	0,015	2	6,01	0,166	2,174	0,05
2	Parallel	1,50E-04	0,055	150	0,023	2	7,278	0,137	1,885	0,052
4	Parallel	1,50E-04	0,055	50	0,008	2	3,291	0,152	1,985	0,045
4	Parallel	1,50E-04	0,055	75	0,011	2	3,879	0,129	1,769	0,049

Table 3.1 - Simulation results for solenoid choice.

As we can see, considering one solenoid, as the number of coils increases (up to 150), the magnetic field close to the solenoid increases. If the number of coils continue to increases from 150 coils on, the magnetic field in the solenoid proximity becomes low while increases the distance at which there is a magnetic field not negligible.

Connecting together two or more solenoids in series is meaningless because we have the same performance as the case of one solenoid (unless the total number of coils is the same).

Considering more solenoids in parallel implies a change in the voltage and current provided by the supply (having fixed the power), while for the magnetic behavior there is a degradation of the performance.

One explanation of this behavior can be that as the solenoids resistance decreases (due to the increasing of solenoids connected in parallel) the contributions of parasitics like Rs and Rdson become more significant.

As conclusion, after all considerations preaviously stated, the solenoid parameters are chosen and summarized in the following:

- 2 Solenoids connected in parallel;
- Total number of coils: 200 (100 per solenoid);
- Total length of solenoids (consequently of the satellite) : 3cm;
- Solenoid resistance: Rsol=33.716 Ω ;
- Equivalent resistance of the system: Req=18.058 Ω (Rs=0.2Ω, Rds_on =1Ω);
- Voltage supply: 6V;
- Dissipated power : 2W ;
- Current flowing in one solenoid : I_L =0.166 A;
- Magnetic field at dmin =1cm : B(dmin)=2.174G ;
- Magnetic field at dmax = 20cm : B(dmax)=0.05G ;

3.2 Class Diagrams

3.2.1 FemtoSat main class diagram

Class - FemtoSat

Development of a FemtoSat i.e a satellite with a weigh less than 100g, but having the main features of a bigger one:

- at least one battery with related circuits
- a transceiver system to communicates with an Interlocutor Satellite
- an on-board processor
- a system for determination and control of the distance with respect to an Interlocutor Satellite

Template Parameters: FemtoSat_ID

References

Type	Value
Folder	${Aramis_Progetto}\FemtoSat

Attributes

Signature: driver : FemtoSat_SW

Signature: -distance threshold 1

The threshold under which RF techniques cannot be applied to measure the distance between satellites (Estimate Distance RF use case).

Signature: -distance threshold 2

The threshold under which magnetic techniques can be applied to measure the distance between satellites (Estimate Distance MAG use case).

Class - FemtoSat_HW

Hardware implementation of FemtoSat.

References

Type	Value
Folder	R:\Progetto\FemtoSat

Operations

Signature: DEBUG()

Debug bus for CC2510. Contains: RESET, DEBUG_DATA, DEBUG_CLOCK, 3V3_SUPPLY.

Signature: MODULE_A()

Class - FemtoSat_SW

The class contains all the software design of FemtoSat .

Attributes

Signature: -distance : float

The value of the actual distance between FemtoSat and Interlocutor Satellite. It is the best estimation obtained as a combination of dist_MAG and dist_RF.

The value is expressed in meters.

Signature: -distance_new : float

The value of the new distance between FemtoSat and Interlocutor Satellite provided by the Controller.

The value is expressed in meters.

Signature: -distance_error : float

The accepted difference between distance and distance_new.

The value is expressed in meters.

-distance_threshold_1 : float const = 0.11

The threshold under which RF techniques cannot be applied to measure the distance between satellites (Estimate Distance RF use case).

The value is expressed in meters.

-distance_threshold_2 : float const = 0.2

The threshold under which magnetic techniques can be applied to measure the distance between satellites (Estimate Distance MAG use case).

The value is expressed in meters.

Signature: -Ton_long : float

The time in which the Magnetic_Actuator_Subsystem keeps on the Electromagnet in a motion operation (Approach_MASTER, Separate_MASTER).

Signature: -Ton_short : float

The time in which the Magnetic_Actuator_Subsystem keeps on the Electromagnet in a distance measurement operation (Measure Distance MAG_MASTER).

Signature: -magX : float

Magnetic field component measured along X-axis.

The value is expressed in Gauss.

Signature: -magY : float

Magnetic field component measured along Y-axis.

The value is expressed in Gauss.

Signature: -magAct : Magnetic_Actuator_SW_Driver

An instance of Magnetic_Actuator_SW_Driver

Signature: -magSens : Magnetometer_Sensor_SW_Driver

An instance of Magnetometer_Sensor_SW_Driver

Signature: -distance_MAG : Distance_MAG

An instance of Distance_MAG

Signature: -1B35_driver : 1B351S_Intersatellite_Communication

An instance of 1B351S_Intersatellite_Communication

Signature: command : Commands

An instance of Commands

Signature: mess : Message

An instance of Message.

In FemtoSat_SW the 1B35_Intersatellite_Communication will be used to send a generic Message to the Interlocutor Satellite. This means that the message_body is considered as a generic array of chars to be filled depending on the case.

The basic structure of the message will be:

- a command chosen among Commands, placed in the first byte;

- one or more parameters, typically float number, having a meaning depending on the type of command.

Operations

Signature: init()

Initializes all attributes and calls the initialization functions of the following objects:

- 1B35_driver

- distance_MAG

- magSens

- magAct

Moreover, it sets all attributes to:

- distance = 0.0

- distance_error = 0.0

- distance_new = 0.0

- magX = 0.0

- magY = 0.0

Signature: getDistanceMAG(distance_MAG : Distance_MAG) : void

Returns the instance of Distance_MAG class containing all the following attributes:

- dist_MAG
- ID_MASTER
- ID_SLAVE
- time_MASTER
- time_SLAVE
- distValid

The reference of the Distance_MAG is passed as parameter to the function.

Signature: getDistanceGlobal() : float

It implements Get Distance Global use case. When the distance estimation is completed it is written in distanceattribute.

Signature: measureDistaceMAG(distance_MAG : Distance_MAG)

It implements Measure Distance MAG_MASTER use case. The instance of Distance_MAG attribute is passed as parameter so that all attributes can be set after a distance measurement is carried out.

Signature: processDistanceMAG(distance_MAG : Distance_MAG)

It implements Process Distance MAG use case. The instance of Distance_MAG attribute is passed as parameter so that all attributes can be set after a distance measurement is carried out.

Signature: setDistanceGlobal(dist : float)

It implements Set Distance Global use case. The value of new distance is passed as parameter.

Signature: setDistanceMAG(dist : float)

It implements Set Distance MAG use case. The value of new distance is passed as parameter.

Signature: setDistanceRF(dist : float)

It implements Set Distance RF use case. The value of new distance is passed as parameter.

Signature: waitRoutine(waiting_time : float)

The function uses the timer to provide a waiting cycle of waiting_time seconds.

Signature: setRemoteTimeReference(timeReference : float)

The function used to communicate the value of local timeReference to Interlocutor Satellite in order to change the remote reference accordingly.

Signature: prepareMessage()

Signature: interpretCommand()

Signature: estimateDistanceMAG(distance_MAG : Distance_MAG)

It implements Estimate Distance MAG use case. The instance of Distance_MAG attribute is passed as parameter so that all attributes can be set after a distance measurement is carried out.

Signature: approach()

It implements Approach_MASTER use case.

Signature: separate()

It implements Separate_MASTER use case.

Class - Distance Controller

The system which allows the satellite to modify the distance from the Interlocutor Satellite.

References

Type	Value
File	C:\Users\Donato\Desktop\Sistemi ELT per applicazioni spaziali\Final Project\Report and presentation\Tesina finale.pptx

Class - Magnetic_Actuator_Subsystem

It is the module which allows the generation of a magnetic field for motion and distance measurement purposes.

It is made of some electronic circuits (Bk1B222_Magnetic_Torque_Actuator) used to drive an Electromagnet.

Furthermore, there is a software class containing the driver routines which govern Magnetic_Actuator_Subsystem operations.

Class - Magnetometer_Subsystem

The module which allows the measurement of magnetic field around the FemtoSat. It is based on Bk1B221_Magnetometer_Sensor and a software driver (Magnetometer_Sensor_SW_Driver).

The magnetic field is measured along two directions (magX, magY) and it is made of both contributions of Earth and Interlocutor Satellite.

A procedure shall be implemented to isolate the Interlocutor Satellite field contribution and eliminate the one of the Earth.

Class - Supply Subsystem

All hardware subsystems and components which provide power supply to the FemtoSat.

3.2.2 FemtoSat_HW

The diagram contains all hardware classes of FemtoSat which are:

- FemtoSat_HW
- Bk1B222_Magnetic_Torque_Actuator
- Bk1B221_Magnetometer_Sensor
- 1B351_Intersatellite_Communication
- Electromagnet
- Solenoid
- Antenna

The Bk1B222_Magnetic_Torque_Actuator, Bk1B221_Magnetometer_Sensor are taken from 1B22_Magnetic_Attitude_Subsystem project [7]; the 1B351_Intersatellite_Communication is taken from 1B35_Intersatellite_Communication project [6]; the last components are designed in the current project.

In the root class FemtoSat_HW, we can see the system interface toward the external environment, i.e. two modules MODULE_A and DEBUG which can be used for testing purposes.

For more details refers to schematics in *Appendix IV*.

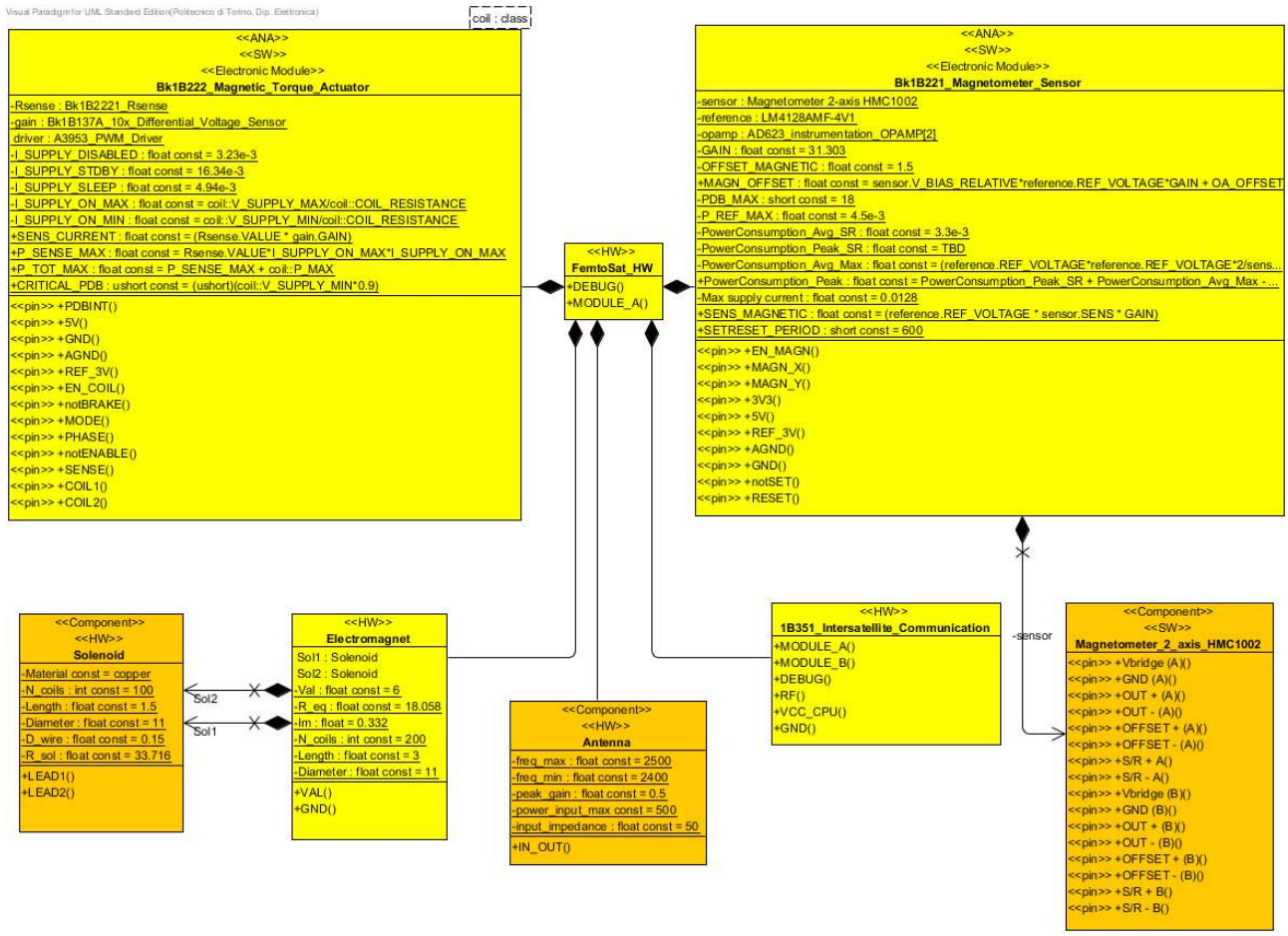


Figure 3.13 - FemtoSat HW class diagram

Class - Bk1B221_Magnetometer_Sensor

The Bk1B221_Magnetometer_Sensor is a 2-axis magnetic sensor for space applications based on a Magnetometer_2_axis_HMC1002 from Honeywell. The bridge-based reading of magnetic field for each of the two orthogonal axes is converted to a single ended voltage signal for each axis, available on the MAGN_X and MAGN_Y outputs.

The circuit operates with two supply voltage (3V3, 5V) and it requires a reference voltage REF_3V.

The Bk1B221_Magnetometer_Sensor can be disabled by pulling down the signal EN_MAGN both to reduce power consumption and to isolate the circuit in case of faults.

The two inputs notSET and RESET are used to trigger the so-called set-reset operation of the magnetic field transducer Magnetometer_2_axis_HMC1002, as detailed in the corresponding datasheet.

References

Type	Value
File	R:\Tesi\Tesi Vico AOCS\Tesi Vico\tesi\AOCS_MagnetometroHW_080416_Vico.doc
Folder	${Aramis_Progetto}\1B_Subsystem_Elements\1B2_Attitude_and_Orbit_Subsystem\1B22_Magnetic_Attitude_Subsystem\1B221_Magnetometer

Attributes

-sensor : Magnetometer 2-axis HMC1002

Biaxial magnetic field sensor.

-reference : LM4128AMF-4V1

Voltage reference which supplies magnetic sensing bridge.

-opamp : AD623_instrumentation_OPAMP[2]

Instrumentation Amplifier used for conditioning circuit of magnetometer.

-GAIN : float const = 31.303

Gain of magnetic field components conditioning circuits.

-OFFSET_MAGNETIC : float const = 1.5

Offset of magnetic field components conditioning circuits, applied on opamp REF pins, in V.

+MAGN_OFFSET : float const = sensor.V_BIAS_RELATIVE*reference.REF_VOLTAGE*GAIN + OA_OFFSET

Global offset of magnetometer sensor on both axes [V], in typical conditions. It depends on sensor typical offset and conditioning circuit gain and offset.

-PDB_MAX : short const = 18

Maximum possible Power Distribution Bus voltage [V].

-P_REF_MAX : float const = 4.5e-3

Maximum dissipated power by each voltage divider on REF_3V, in W (see electric scheme in par. 6.1.6). Calculated value.

-PowerConsumption_Avg_SR : float const = 3.3e-3

Average power consumption of Set/Reset circuit, in W. Theoric value. Based on sensor's datasheet declared value of maximum S/R effective current from power supply (1 mA).

-PowerConsumption_Peak_SR : float const = TBD

Set/Reset circuit peak power consumption, in W.

-PowerConsumption_Avg_Max : float const = (reference.REF_VOLTAGE*reference.REF_VOLTAGE*2/sensor.R_BRIDGE_MIN) + (reference.I_SUPPLY_MAX*PDB_MAX) + 2*(opamp[1].I_SUPPLY_MAX*5) + (2*P_REF_MAX) + PowerConsumption_Avg_SR

Maximum average power consumption of Magnetometer Sensor block, in W. Calculated from maximum values declared in components' datasheet.

Contributions to power dissipation are Magnetometer_2_axis_HMC1002, REF02_5V_Reference, 2x AD623_instrumentation_OPAMP, 2x REF_3V voltage dividers and Set/Reset circuit.

+PowerConsumption_Peak : float const = PowerConsumption_Peak_SR + PowerConsumption_Avg_Max - PowerConsumption_Avg_SR

Peak power consumption of Magnetometer Sensor block in W, including Set/Reset contribution -> PowerConsumption_Peak_SR.

-Max supply current : float const = 0.0128

Maximum supply current of Magnetometer Sensor block (when enabled) in A. Tested with (model element not found) voltage = 14 V.

+SENS_MAGNETIC : float const = (reference.REF_VOLTAGE * sensor.SENS * GAIN)

Sensitivity of conditioned magnetic sensor, in V/T.

+SETRESET_PERIOD : short const = 600

The repetition rate of the System SR Magnetometer, in s.

Operations

Signature: EN_MAGN()

Enables magnetometer, by providing supply to bridge and opamp. Active high. TTL input.

Signature: MAGN_X()

Output voltage for magnetic field along X-axis.

Output voltage is

OFFSET_MAGNETIC [V] + SENS_MAGNETIC [V/T] * MagneticField(x) [T]

Signature: MAGN_Y()

Output voltage for magnetic field along Y-axis.

Output voltage is

OFFSET_MAGNETIC [V] + SENS_MAGNETIC [V/T] * MagneticField(x) [T]

Signature: 3V3()

3.3V supply voltage. Supplies Set/Reset circuit.

Signature: 5V()

5 V supply voltage. Supplies differential OpAmps.

Signature: REF_3V()

3V voltage reference. It feeds a voltage divider whose output is connected to REF pin of each AD623.

Signature: AGND()

Analog ground

Signature: GND()

Power and digital ground.

Signature: notSET()

Triggers a SET pulse on both magnetometer channels when transitioning from high to low. TTL input.

Signature: RESET()

Triggers a RESET pulse on both magnetometer channels when transitioning from low to high. TTL input.

Class - Magnetometer_2_axis_HMC1002

Transit To: Magnetometer_2_axis_HMC1002

References

Type	Value
File	R:\Progetto\1C_Other_Activities\1C5_Documentation_and_Qualification\1C54_Datasheets\hmc1002.pdf

Operations

Signature: Vbridge (A)()

High stability IN supply voltage of 5V for magnetic sensor. Connected to REF_5V.

Signature: GND (A)()

IN/OUT analog GND signal

Signature: OUT + (A)()

OUT Differential magnetic signal x-axis

Signature: OUT - (A)()

OUT Differential magnetic signal x-axis

Signature: OFFSET + (A)()

Signature: OFFSET - (A)()

Signature: S/R + A()

IN to SR_A signal from Set/Reset circuit.

Signature: S/R - A()

OUT signal to GND

Signature: Vbridge (B)()

High stability IN supply voltage of 5V for magnetic sensor. Connected to REF_5V.

Signature: GND (B)()

IN/OUT analog GND signal

Signature: OUT + (B)()

OUT Differential magnetic signal y-axis

Signature: OUT - (B)()

OUT Differential magnetic signal y-axis

Signature: OFFSET + (B)()

Signature: OFFSET - (B)()

Signature: S/R + B()

IN to SR_B signal from Set/Reset circuit.

Signature: S/R - B()

OUT signal to GND

Class - Bk1B222_Magnetic_Torque_Actuator

This is the part of the circuitry that implements generation of a magnetic momentum to execute a command of ACTUATE_MAGNETIC.

Template Parameters: coil

References

Type	Value
Folder	R:\Progetto\1B_Subsystem_Elements\1B2_Attitude_and_Orbit_Subsystem\1B22_Magnetic_Attitude_Subsystem\1B222_Magnetic_Torque_Actuator
File	R:\Tesi\Tesi Vico AOCS\Tesi Vico\tesi\AOCS_SceltеProgettuali_Ruota d'inerzia_080416_Vico.doc

Attributes

-Rsense : Bk1B2221_Rsense

Resistor used to sense coil current.

-gain : Bk1B137A_10x_Differential_Voltage_Sensor

Differential amplifier used for coil current conditioning.

driver : A3953_PWM_Driver

-I_SUPPLY_DISABLED : float const = 3.23e-3

Maximum supply current (A) when disabled. Tested. Maybe due to faulty switch on PDBINT voltage.

-I_SUPPLY_STDBY : float const = 16.34e-3

Maximum supply current (A) when enabled (driver in **standby mode**) but not active, tested.

-I_SUPPLY_SLEEP : float const = 4.94e-3

Maximum supply current (A) when enabled (driver in **sleep mode**) but not active, tested.

-I_SUPPLY_ON_MAX : float const = coil::V_SUPPLY_MAX/coil::COIL_RESISTANCE

Maximum supply current (A) when enabled and active, over all operating conditions.

-I_SUPPLY_ON_MIN : float const = coil::V_SUPPLY_MIN/coil::COIL_RESISTANCE

Minimum supply current (A) when enabled and active, over all operating conditions.

+SENS_CURRENT : float const = (Rsense.VALUE * gain.GAIN)

Sensitivity of SENSE output (conditioned coil current), in V/A

+P_SENSE_MAX : float const = Rsense.VALUE*I_SUPPLY_ON_MAX*I_SUPPLY_ON_MAX

Maximum power dissipated by coil current sensing resistor.

+P_TOT_MAX : float const = P_SENSE_MAX + coil::P_MAX

Maximum power dissipated by Bk1B222_Magnetic_Torque_Actuator block.

+CRITICAL_PDB : ushort const = (ushort)(coil::V_SUPPLY_MIN*0.9)

Critical value of PDB voltage [V]. It is used for error Supervision Xxx use case.

Operations

Signature: PDBINT()

IN supply voltage for solenoid.

Signature: 5V()

IN supply voltage for logic part of solenoid driver

Signature: GND()

IN/OUT ground voltage for digital and power signals

Signature: AGND()

IN/OUT analog groung voltage signal

Signature: REF_3V()

IN reference voltage signal necessary to coil driver to limit maximum load current

Signature: EN_COIL()

When high, enables the whole Bk1B222_Magnetic_Torque_Actuator block by providing both power supplies. When low, both power supplies are removed internally from all analog and digital circuits, by means of PMOS devices.

Signature: notBRAKE()

IN signal to drive solenoid driver. Active low. If low, connects coil wires to ground (turns off both source drivers and turns on both sink drivers), discharging the energy previously cumulated into coil. Used to dynamically brake brush DC motors.

Signature: MODE()

IN signal to drive solenoid driver.

When ENABLE and BRAKE are high ('1'):

- if MODE = '1' driver is in "Sleep Mode" (reduced power consumption)

- if MODE = '0' driver is in Standby

When ENABLE='0' and BRAKE='1':

- if MODE = '1' coil is driven in fast current decay mode

- if MODE = '0' in slow current decay mode (not used)

When BRAKE='0':

- if MODE = '1' coil is discharged in fast current decay mode

- if MODE = '0' coil is discharged without current control

Signature: PHASE()

IN signal to drive solenoid driver. It determines current-flow direction. If PHASE='1' coil is driven forward (current flows from COIL1 output to COIL2 output), if PHASE='0' coil is driven reverse (current flows from COIL2 to COIL1).

Signature: notENABLE()

IN signal to drive solenoid driver. Active low. If low, enables coil driving by allowing current flow into it.

Signature: SENSE()

OUT Current into solenoid signal from driver, conditioned.

Signature: COIL1()

OUT signal to drive the solenoid

Signature: COIL2()

OUT signal to drive the solenoid

Class - Electromagnet

It is the load of the (model element not found) and it is made of the parallel connection of two Solenoids.

When a current flows through it, a magnetic field is produced having a magnitude and sign in accordance to the current itself.

The component of the magnetic field that is meaningful for the case study is the one parallel to the longitudinal axis of the magnet.

Concerning the mechanical arrangement, the component is placed with its longitudinal axis orthogonal with respect to the plane where the PCB lies and it is fixed on the PCB in the middle of the magnet, where there are the terminations of the two Solenoids.

More details are shown in the following picture.

Comments

Current flowing in the magnet	
Documentation	In order to evaluate the current flowing in the Electromagnet (Im) an equivalent circuit has to be considered which is made of a voltage generator (Val) supplying a resistance made of the series of R_eq, (model element not found) and (model element not found). This value is useful to evaluate the power consumption, but to evaluate the magnetic field generated, the current flowing in one Solenoid shall be considered (one half of Im).
Author	Donato
Date Time	15-lug-2012 9.04.21

Attributes

Signature: Sol1 : Solenoid

Signature: Sol2 : Solenoid

-Val : float const = 6

Supply voltage (measurement unit: V).

-R_eq : float const = 18.058

Equivalent resistance (measurement unit: Ohm).

-Im : float = 0.332

Supply current asked by the Electromagnet (measurement unit: A).

-N_coils : int const = 200

Total number of coils.

-Length : float const = 3

Electromagnet length (measurement unit: cm).

-Diameter : float const = 11

Electromagnet diameter (measurement unit: cm).

Operations

Signature: VAL()

Supply voltage.

The signal is connected to LEAD1 pin of Sol1 and LEAD1 pin of Sol2.

Signature: GND()

Ground.

The signal is connected to LEAD2 pin of Sol1 and LEAD2 pin of Sol2.

Class - Solenoid

Solenoid made of a certain number (N_coils) of windings of copper wire.

Attributes

-Material const = copper

Constituting material.

-N_coils : int const = 100

Total number of coils.

-Length : float const = 1.5

Solenoid length (measurement unit: cm).

-Diameter : float const = 11

Solenoid diameter (measurement unit: cm).

-D_wire : float const = 0.15

Wire diameter (measurement unit: mm).

-R_sol : float const = 33.716

Solenoid equivalent resistance (measurement unit: Ohm).

Operations

Signature: LEAD1()

One lead of the Solenoid.

Signature: LEAD2()

One lead of the Solenoid.

Class - Antenna

The antenna is designed for 2.45 GHz applications but, as the datasheet shows, if a proper matching circuit is inserted between the feeding line and the antenna, the S1.1 parameter is smaller than -10dB.

The radiation pattern shows a quasi-isotropic behaviour.

Comments

How to buy	
Documentation	It is available on Digi-key website.
Author	S176277
Date Time	Jun 20, 2013 5:07:25 PM

References

Type	Value
URL	http:\\www.johansontechnology.com\images\stories\ip\rf-antennas\JTI_Antenna-2450AT18A100_10-03.pdf

Attributes

-freq_min : float const = 2400

The measurement is expressed in MHz.

-freq_max : float const = 2500

The measurement is expressed in MHz.

-peak_gain : float const = 0.5

The measurement is expressed in dBi.

-power_input_max const = 500

The measurement is expressed in mW.

-input_impedance : float const = 50

The measurement is expressed in ohm.

Operations

Signature: IN_OUT()

3.2.3 FemtoSat_SW

The diagram contains all software classes of FemtoSat, the main ones are:

- FemtoSat_SW
- 1B351S_Intersatellite_Communication
- Magnetic_Actuator_SW_Driver
- Magnetometer_Sensor_SW_Driver

The 1B351S_Intersatellite_Communication is taken from 1B35_Intersatellite_Communication project [6]; the last components are simplified drivers designed in the current project for pre-existing blocks Bk1B222_Magnetic_Torque_Actuator, Bk1B221_Magnetometer_Sensor described in the hardware section of the chapter.

Figure 3.14 - FemtoSat SW class diagram

Class - FemtoSat_main

The class of the FemtoSat which contains the main.

Transit From: FemtoSat_main

Attributes

Signature: -femtoSat1S : FemtoSat_SW

An instance of FemtoSat_SW.

Operations

Signature: main()

The function which is called at the FemtoSat bootstrap.

Class - Distance_MAG

The class contains all information related to a distance measurement performed with magnetic techniques.

The measurement value is stored in dist_MAG variable, other data are stored in the other attributes:

- distValid
- ID_MASTER
- ID_SLAVE
- time_MASTER
- time_SLAVE

Attributes

Signature: +ID_MASTER : unsigned char

This is the FemtoSat_ID of the FemtoSat master which has requested the measurement.

Signature: +ID_SLAVE : unsigned char

This is the FemtoSat_ID of the Interlocutor Node slave which helps the master in the measurement.

Signature: +dist_MAG : float

Signature: +distValid : bool

The flag which indicates if the dist_MAG value is valid i.e. the measurement is carried out correctly.

Signature: +time_MASTER : float

This is the time relative to the internal clock of the FemtoSat, the value is updated at the end of the measurement.

Signature: +time_SLAVE : float

This is the time relative to the internal clock of the Interlocutor Node, the value is updated at the end of the measurement.

Operations

Signature: init()

Initialises the attributes of the class to:

- dist_MAG = 0.0
- distValid = false
- ID_MASTER = 0
- ID_SLAVE = 0
- time_MASTER = 0.0
- time_SLAVE = 0.0

Class - Message

Is the set of data that can be exchanged between 1B35_Intersatellite_Communication and Interlocutor Node.

Attributes

Signature: +preamble_length : unsigned short

Indicates the length of the preamble of the message, that is to say the length of that part of the message which contains information about the message, not the real message body.

Signature: +sourceID : unsigned char

It's the ID of the source of the message. It's a value chosen among one of the ID present in ID_List.

Signature: +destID : unsigned char

It's the ID of the destination of the message. It is a value among the ones present in ID_List.

Signature: +message_length : unsigned short

It's the length of the message itself.

Signature: +message_body : unsigned char

This attribute contains the main body of the message.

It can contain

- Distance_RF
- one of t_Commands
- a generic message

Signature: +message_valid : bool

It is a flag which specifies if the message is valid or not.

The variable assumes "true" as value when there is an incoming message and the data are correct.

The variable assumes "false" as value either when the incoming message is read by the OBC (Read Message use case) so data become obsolete or when data are corrupted.

It's a boolean value which indicates if the message is correct or corrupted.

- its value is 1 if message is valid
- its value is 0 instead when the message is corrupted

Signature: +power : float

This is power_TX. expressed in Watt.

Signature: +frequency : S-band_channels

Indicates the frequency of transmission.

It's one of the values among the ones listed in S-band_channels

Signature: +RSSI_value : float

Class - Commands

Attributes

Signature: -TURN_ON_MAGNET_FORWARD

Stereotypes: Constant

It is used to trigger Turn ON Magnet Forward Mode use case.

The command doesn't involve any parameter.

Signature: -TURN_ON_MAGNET_REVERSE

Stereotypes: Constant

It is used to trigger Turn ON Magnet Reverse Mode use case.

The command doesn't involve any parameter.

Signature: -TURN_ON_MAGNET_AC

Stereotypes: Constant

It is used to trigger Turn ON Magnet_AC use case.

The command shall be followed by two float parameter which are the dutyCycle and phaseReference.

Signature: -TURN_OFF_MAGNET

Stereotypes: Constant

It is used to trigger Turn OFF Magnet use case.

The command doesn't involve any parameter.

Signature: -SET_TIME_REFERENCE

Stereotypes: Constant

It is used to trigger Set Remote Temporal Reference use case.

The command shall be followed by one float parameter which is timeReference.

Class - Magnetometer_Sensor_SW_Driver

The software driver to govern Magnetometer_Subsystem operations.

Attributes

Attributes

Signature: -Tset_reset : float

Time period in which current flows through Magnetometer_2_axis_HMC1002 in Set Magnetic Sensor HMC1002/Reset Magnetic Sensor HMC1002 use cases. It is measured in us.

Signature: -Vset : float

The voltage read after a Set Magnetic Sensor HMC1002 operation.

Signature: -Vreset : float

The voltage read after a Reset Magnetic Sensor HMC1002operation.

Signature: -Voffset : float

The voltage offset evaluated after a Set Magnetic Sensor HMC1002/ Reset Magnetic Sensor HMC1002 operation.

Signature: -magX : float

Magnetic field component measured along X-axis

The value is expressed in Gauss.

Signature: -magY : float

Magnetic field component measured along Y-axis.

The value is expressed in Gauss.

Operations

Signature: init()

It initializes all attributes of the class to:

- Tset_reset = 0.0;
- Vset = 0.0;
- Vreset = 0.0;
- Voffset = 0.0;
- magX = 0.0;
- magY = 0.0;

Signature: getField(magX : float, magY : float)

Getter method for magX and magY.

Signature: measureField()

The function triggers the field measurement.

Signature: turnON()

The function is used to turns ON the Magnetometer_Subsystem.

Signature: turnOFF()

The function is used to turns ON the Magnetometer_Subsystem.

Signature: set()

The function triggers a set operation to let Magnetometer_2_axis_HMC1002 sensor work properly.

Signature: reset()

The function triggers a reset operation to let Magnetometer_2_axis_HMC1002 sensor work properly.

Class - Magnetic_Actuator_SW_Driver

The software driver to govern Magnetic_Actuator_Subsystem operations.

Attributes

Signature: -timeReference : float

It is the time reference period used in motion operation of FemtoSat, it shall not be lower than CC2510 clock period.

Signature: -phaseReference : float

The phase displacement with respect to the reference signal provided by the master satellite. It is 0 if the master is FemtoSat, it can have a value between 0 and 2pi if the master is the Interlocutor Satellite.

The value is expressed in radiants.

Operations

Signature: turnON(dutyCycle : float, phaseReference : float)

It is used to implement both Turn ON Magnet_AC, Turn ON Magnet_DC use cases basing on the value of dutyCycle that is passed as parameter.

Another parameter passed to the function is the phaseReference with respect to reference signal generated by the master (FemtoSat or Interlocutor Satellite depending on the operation that is running)

Signature: turnOFF()

It is used to turn off the Magnetic_Actuator_Subsystem.

Signature: setForwardMode()

It implements Turn ON Magnet Forward Mode use case.

Signature: setReverseMode()

It implements Turn ON Magnet Reverse Mode.

Signature: getTimeReference() : float

Getter method for timeReference.

Signature: setTimeReference(timeReference : float) : void

The function set the timeReference.

3.2.4 Distance_Controller

The diagram contains a logical arrangement of hardware and software classes in order to identify Magnetic_Actuator_Subsystem and Magnetometer_Subsystem units.

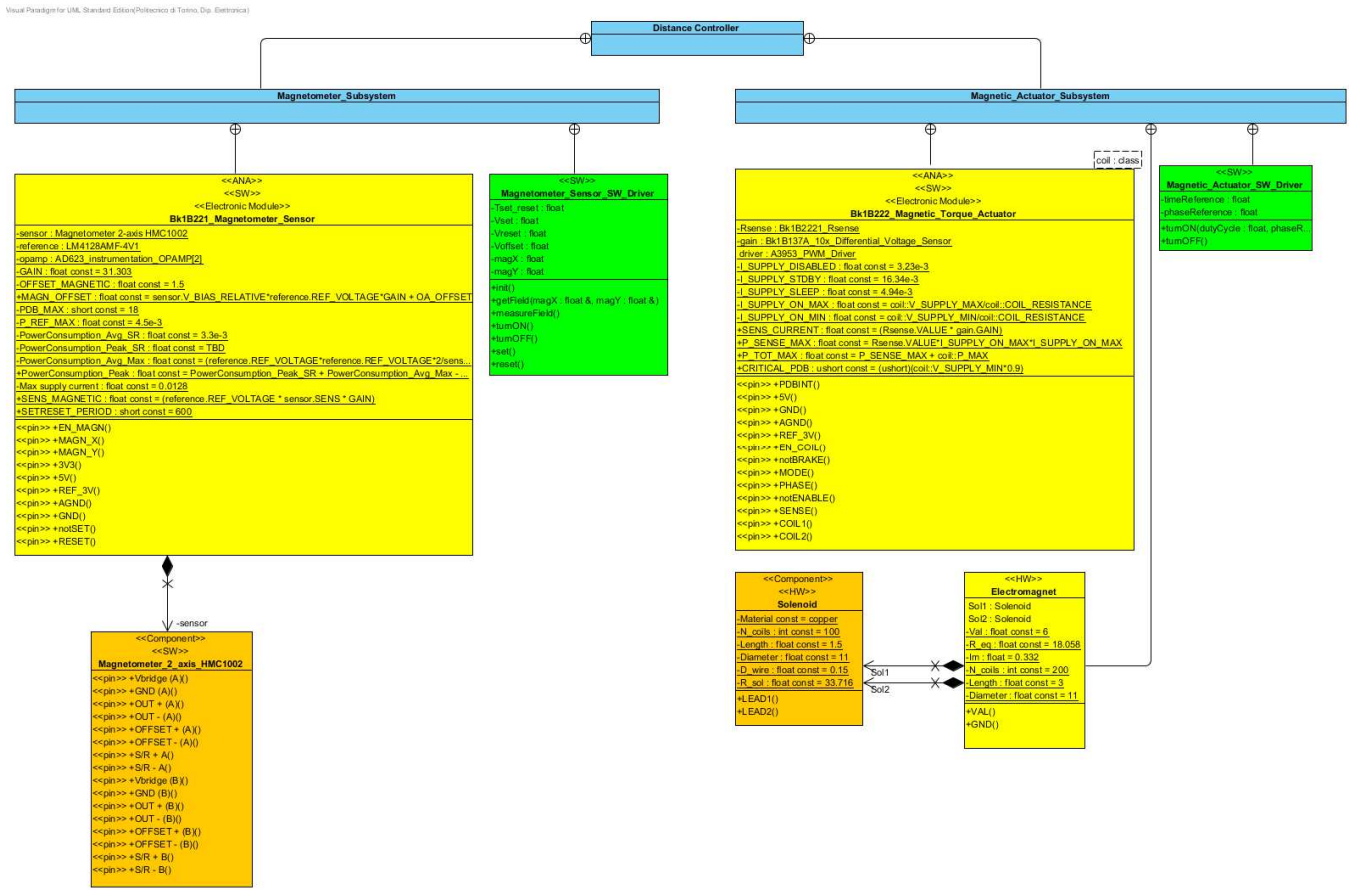


Figure 3.15 - Distance controller class diagram

Class - Magnetic_Actuator_Subsystem

It is the module which allows the generation of a magnetic field for motion and distance measurement purposes.

It is made of some electronic circuits (Bk1B222_Magnetic_Torque_Actuator) used to drive an Electromagnet.

Furthermore, there is a software class containing the driver routines which govern Magnetic_Actuator_Subsystem operations.

Class - Magnetometer_Subsystem

The module which allows the measurement of magnetic field around the FemtoSat. It is based on Bk1B221_Magnetometer_Sensor and a software driver (Magnetometer_Sensor_SW_Driver).

The magnetic field is measured along two directions (magX, magY) and it is made of both contributions of Earth and Interlocutor Satellite.

A procedure shall be implemented to isolate the Interlocutor Satellite field contribution and eliminate the one of the Earth.

3.3 Sequence Diagrams

3.3.1 Get Distance MAG

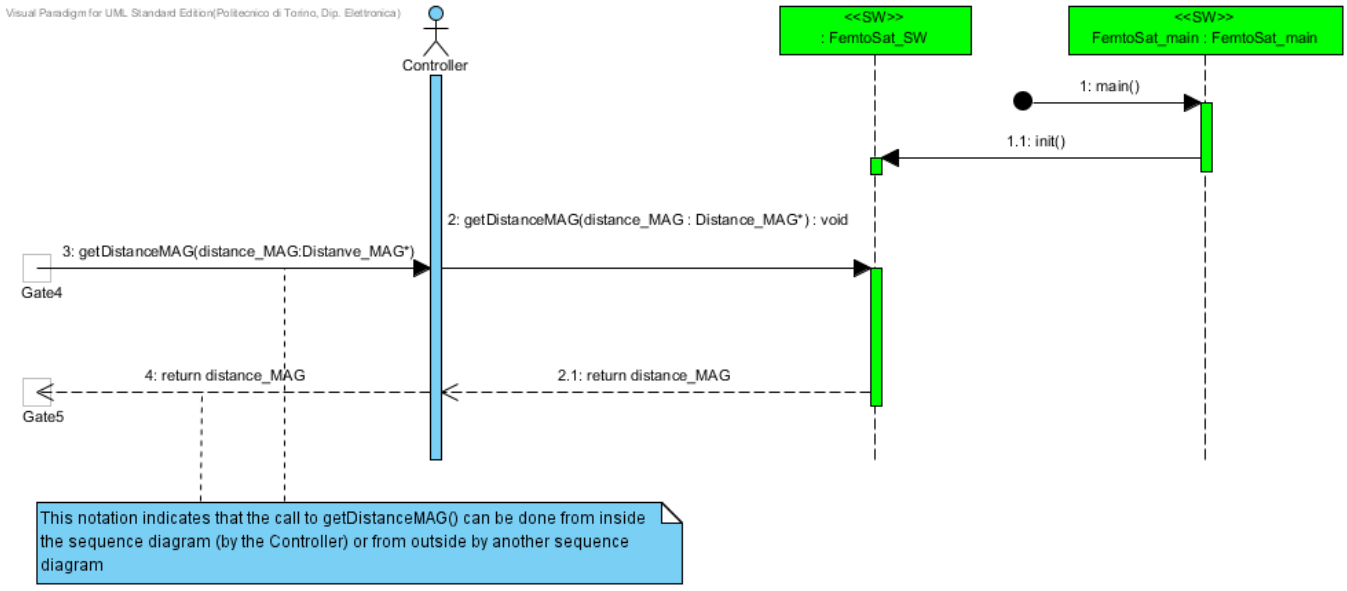


Figure 3.16 - Get Distance MAG sequence diagram

3.3.2 Process Distance MAG

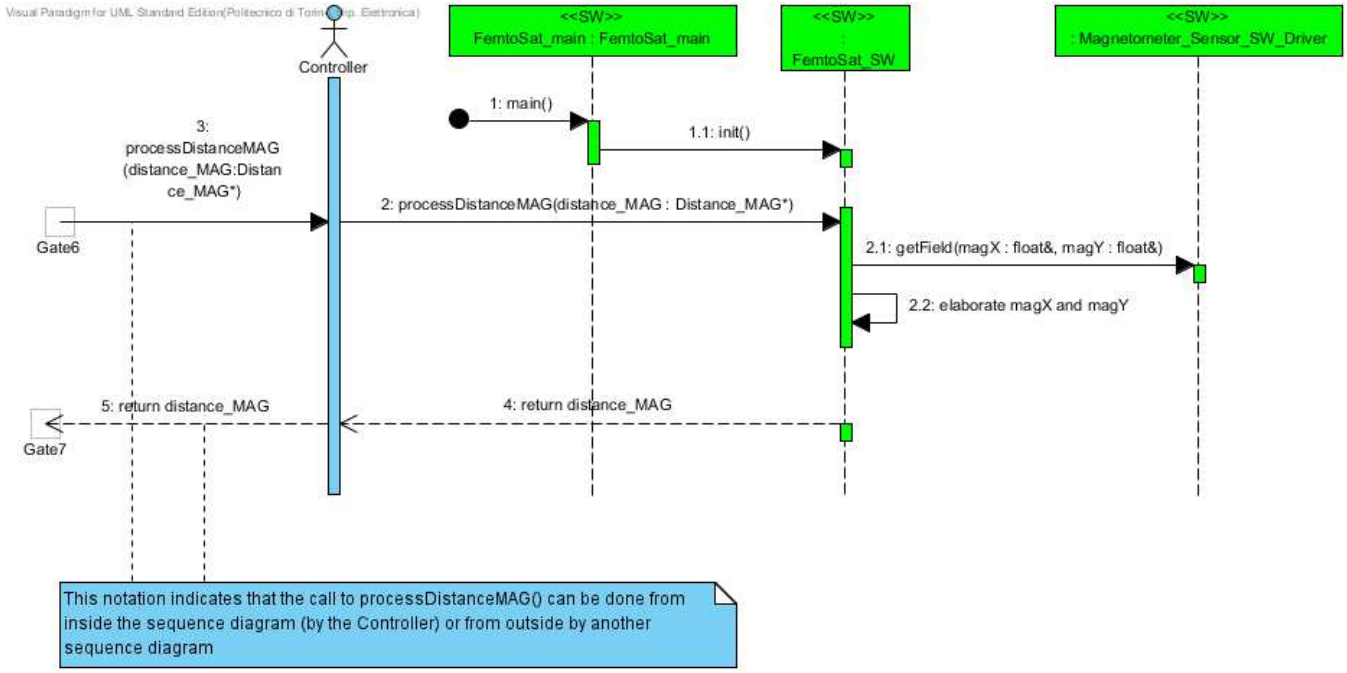


Figure 3.17 - Process Distance MAG sequence diagram

3.3.3 Measure Distance MAG_MASTER

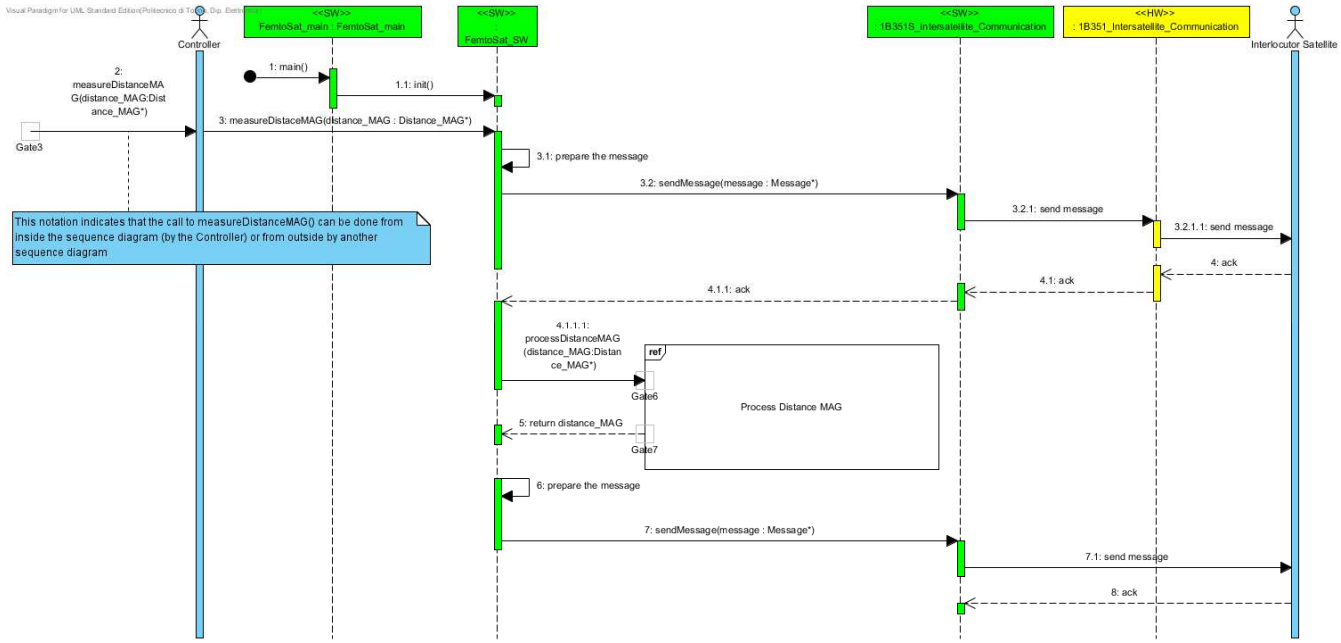


Figure 3.18 - Measure Distance MAG_MASTER sequence diagram

3.3.4 Measure Distance MAG_SLAVE

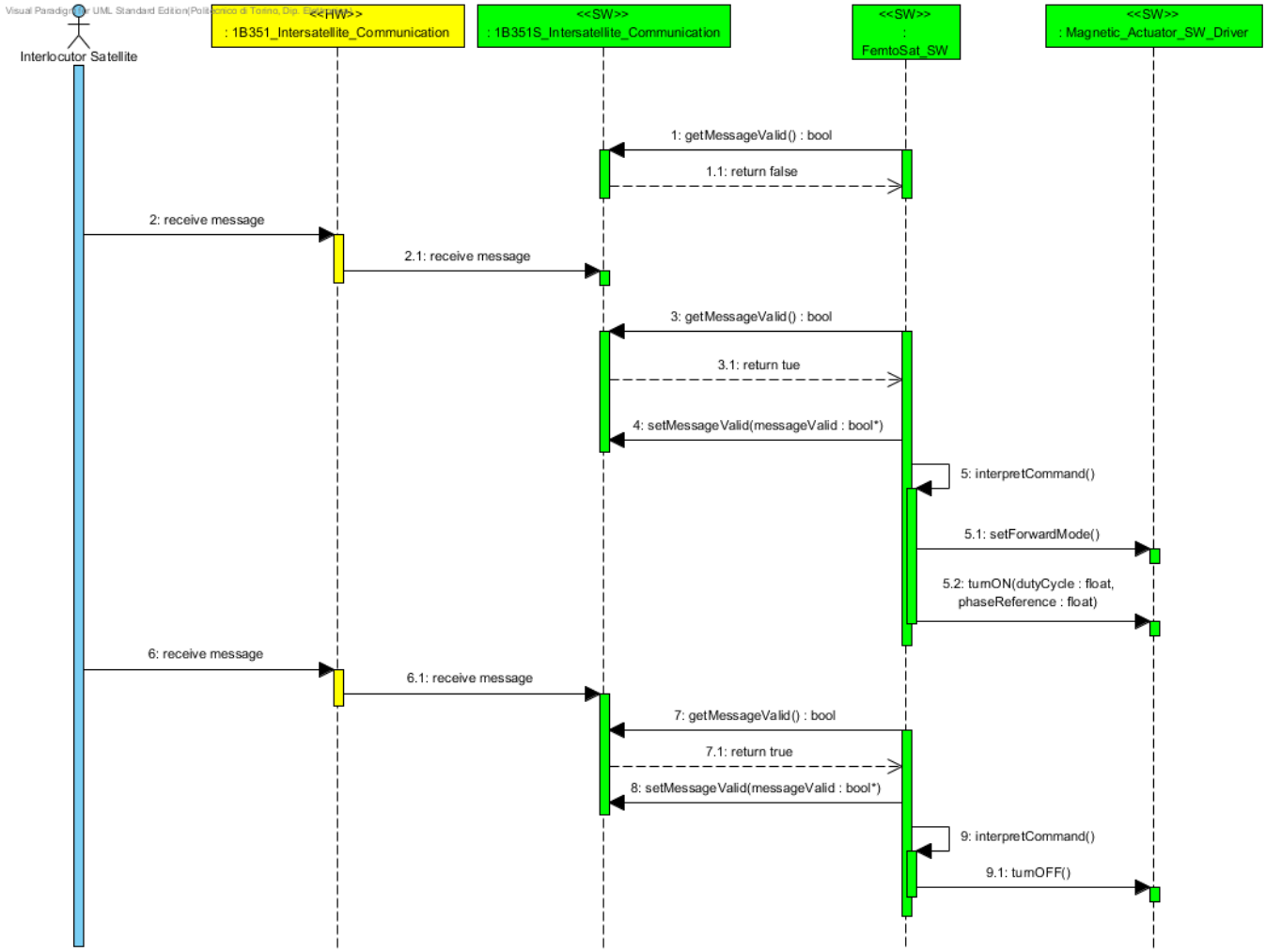


Figure 3.19 - Measure Distance MAG_SLAVE sequence diagram

3.3.5 Estimate Distance MAG

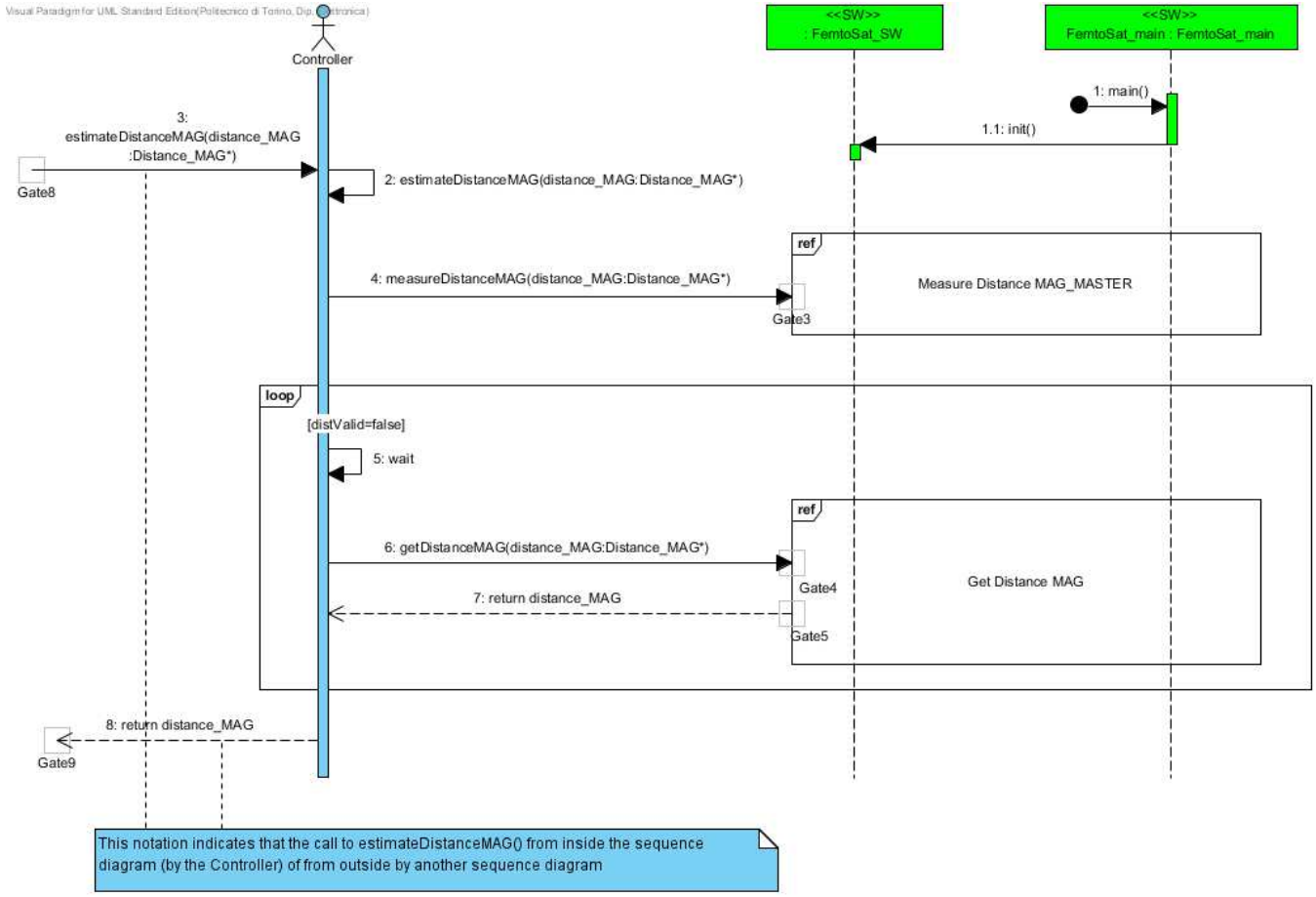


Figure 3.20 - Estimate Distance MAG sequence diagram

3.3.6 Get Distance Global

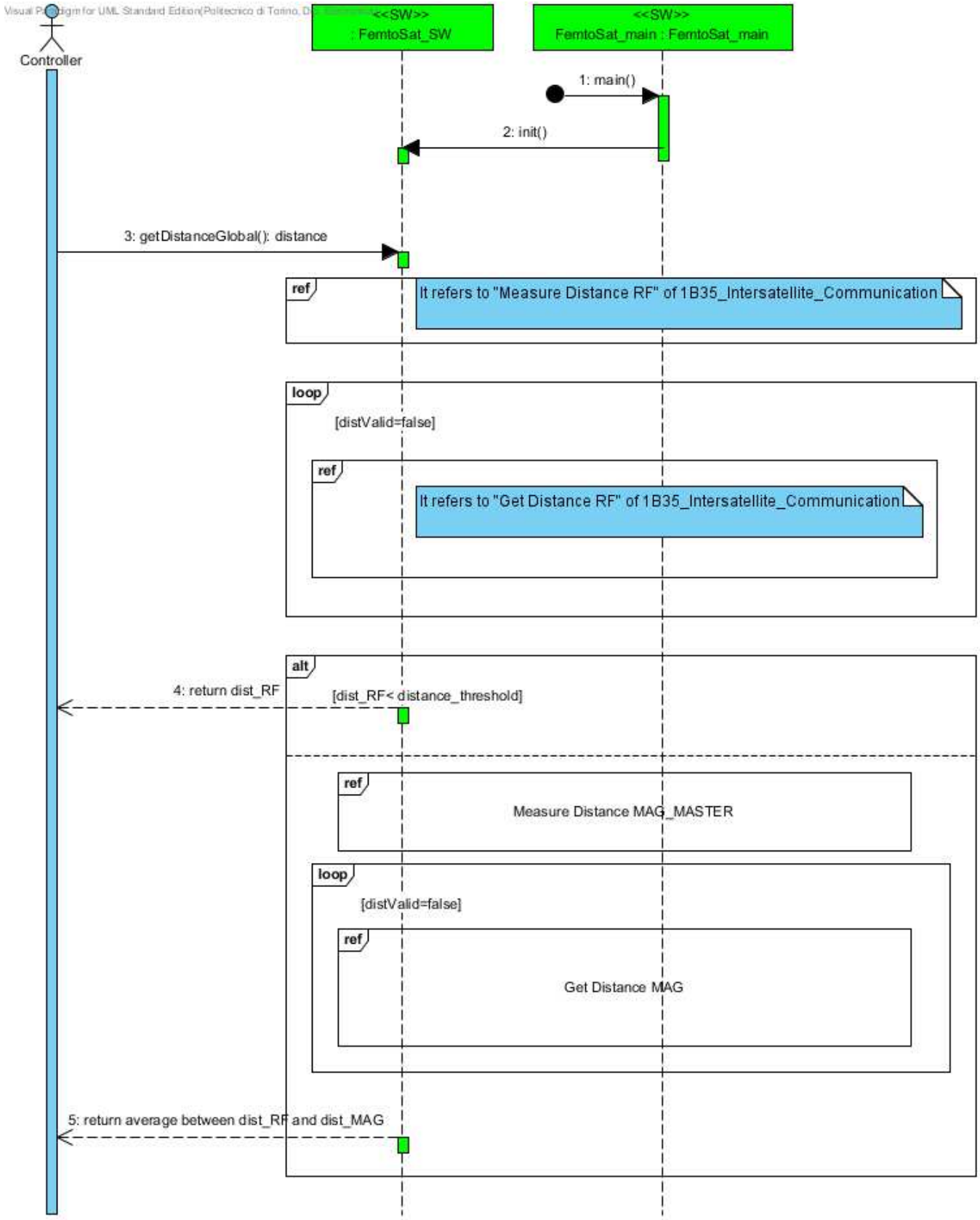


Figure 3.21 - Get Distance Global sequence diagram

3.3.7 Set Distance MAG

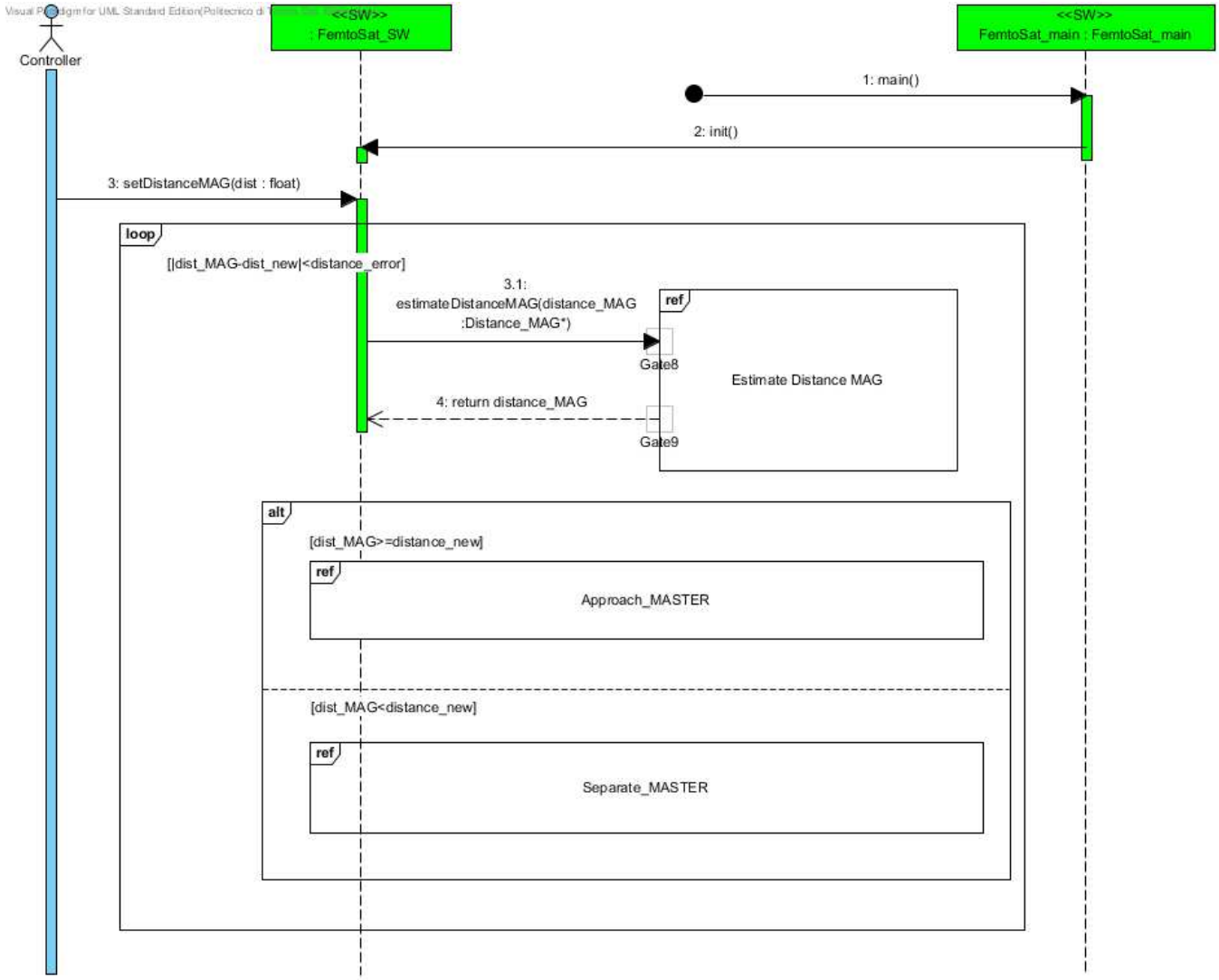


Figure 3.22 - Set Distance MAG sequence diagram

3.3.8 Set Distance RF

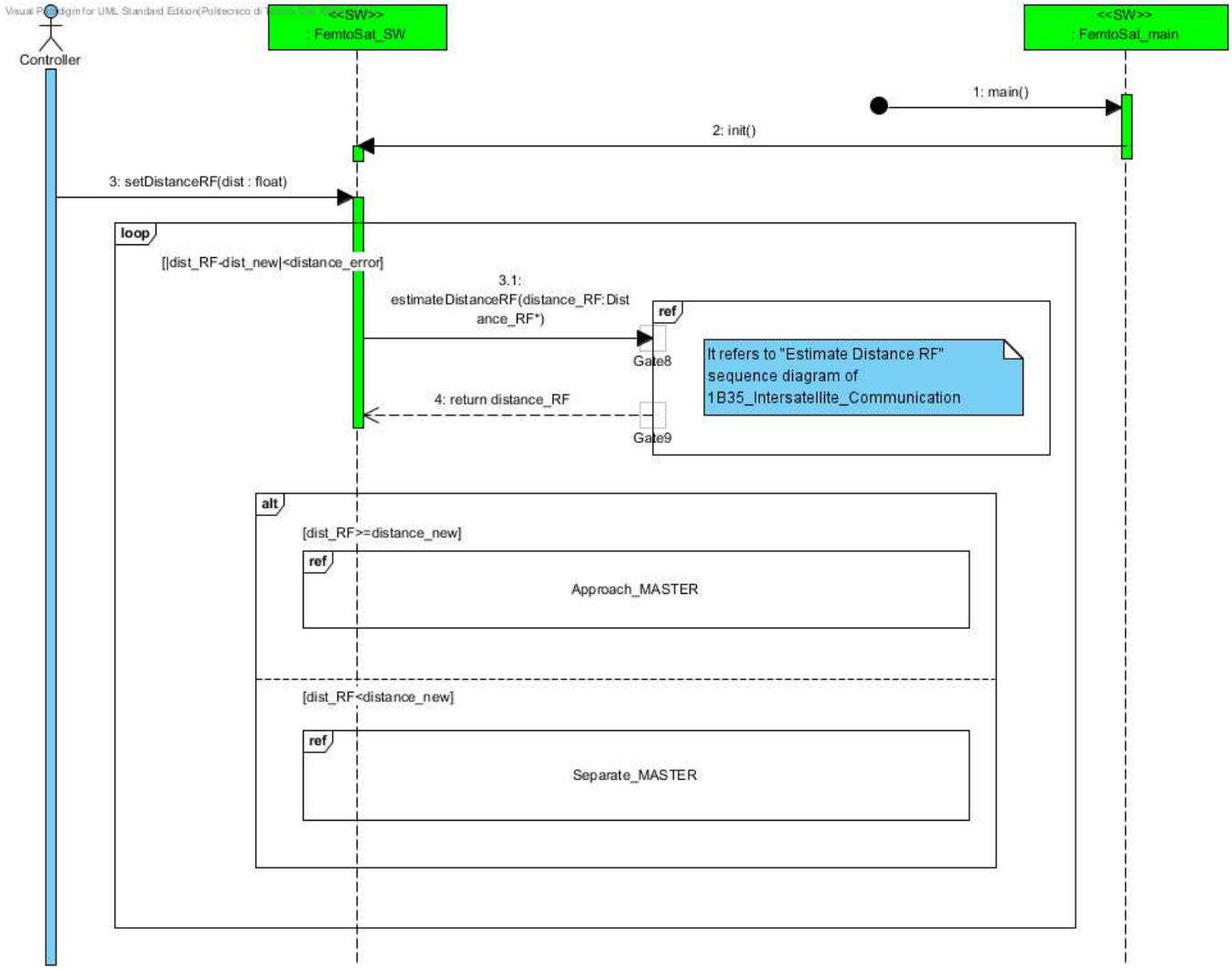


Figure 3.23 - Set Distance RF sequence diagram

3.3.9 Set Distance Global

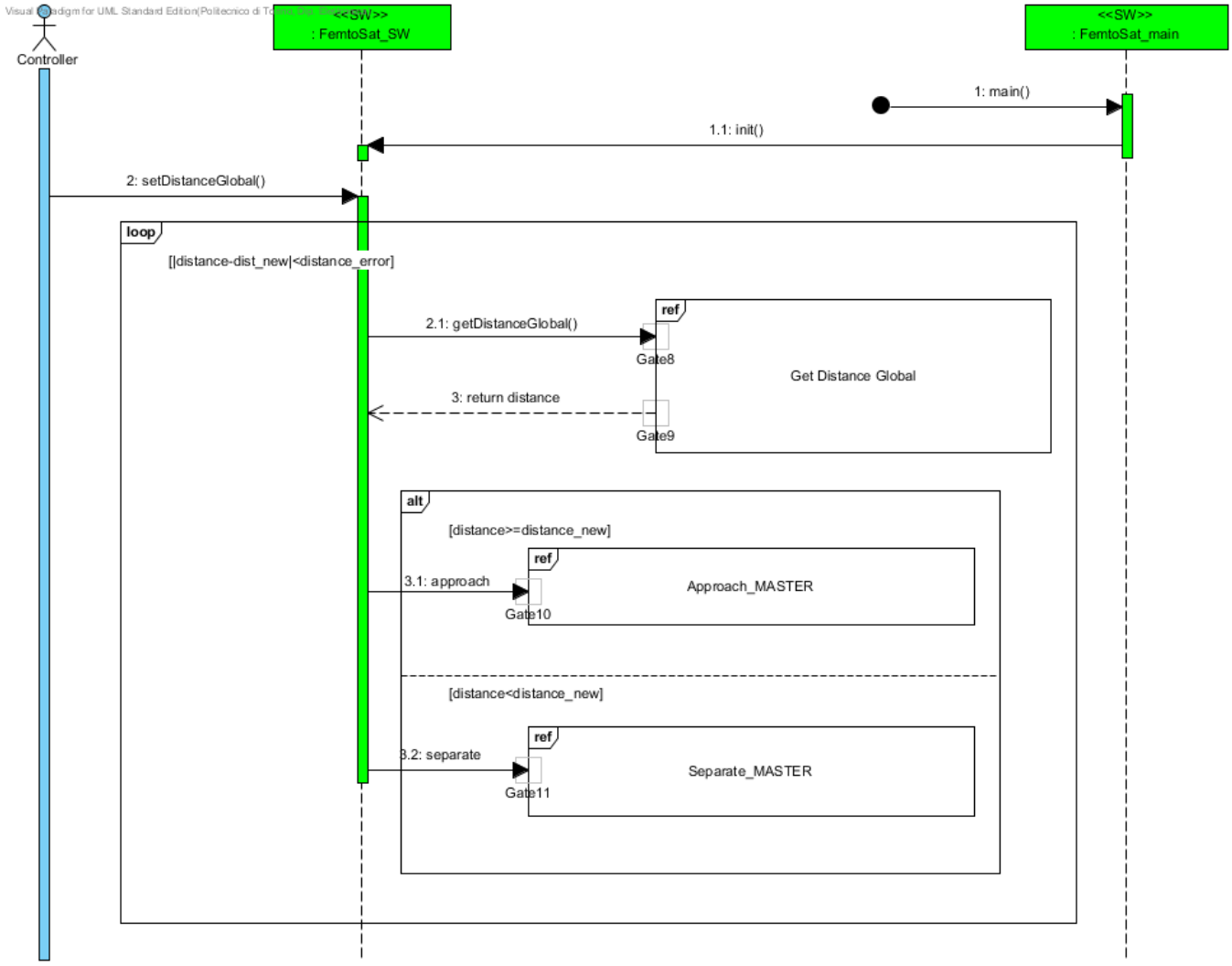


Figure 3.24 - Set Distance Global sequence diagram

3.3.10 Approach_MASTER

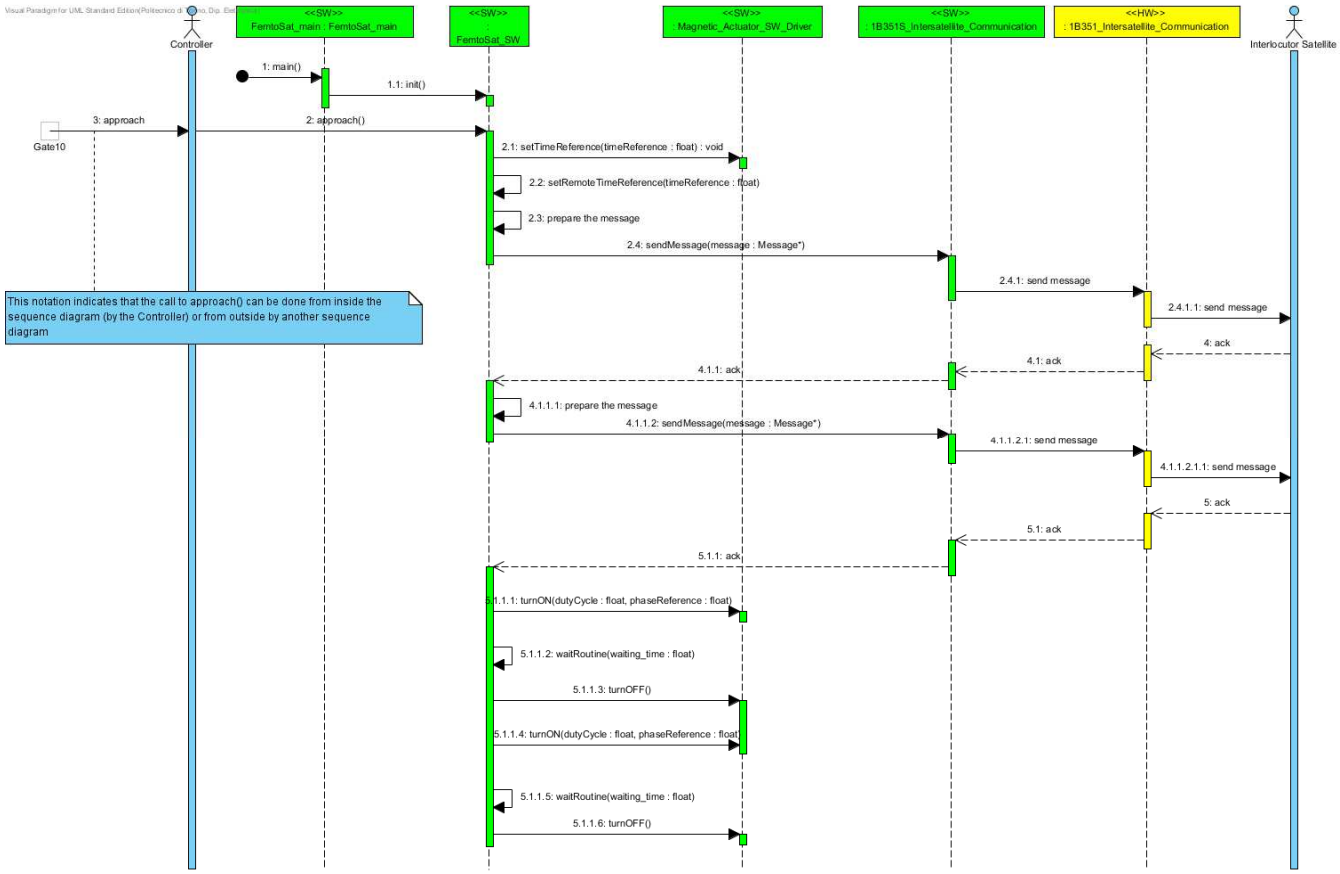


Figure 3.25 - Approach MASTER sequence diagram

3.3.11 Separate_MASTER

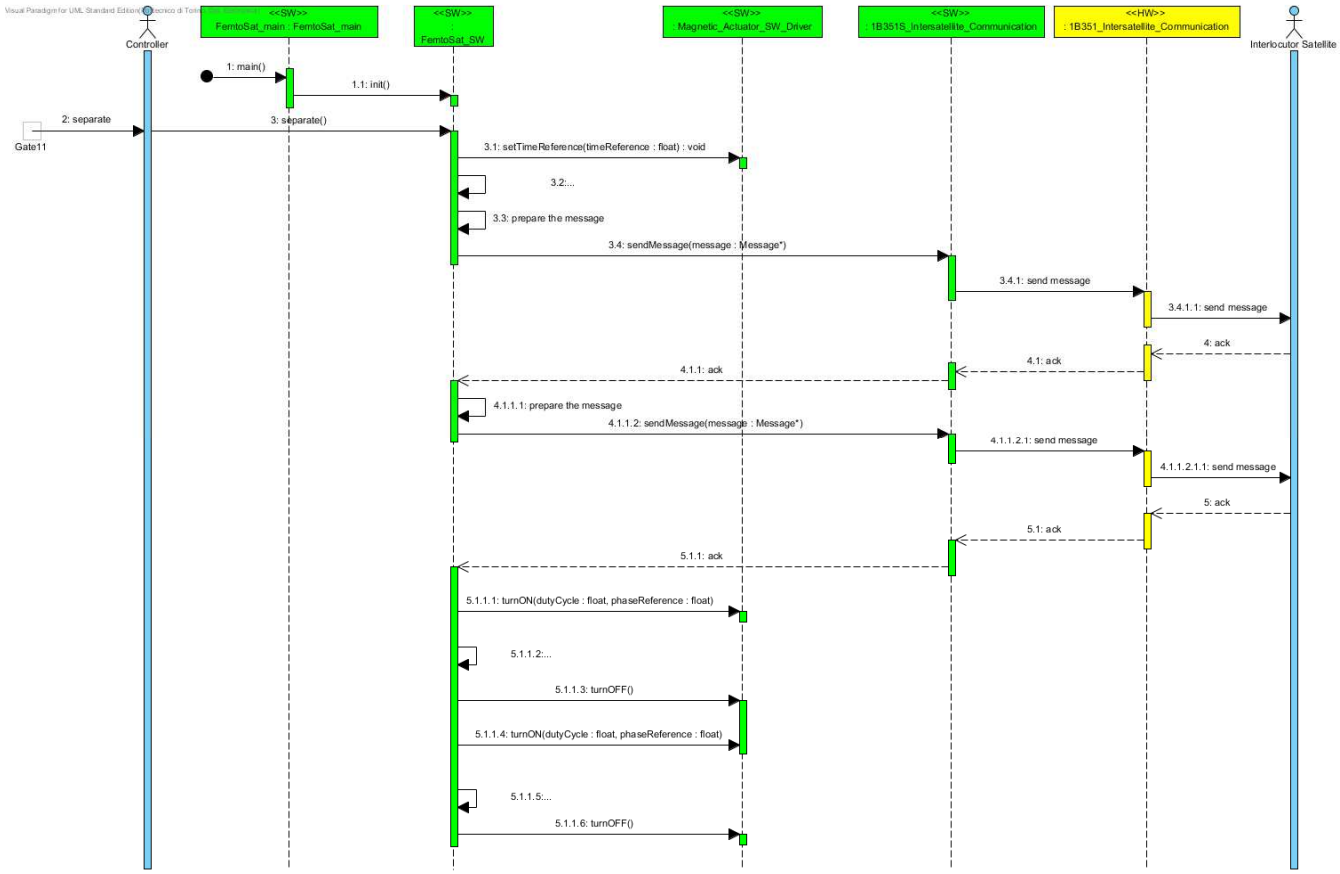


Figure 3.26 - Separate MASTER sequence diagram

3.3.12 Approach_Separate_SLAVE

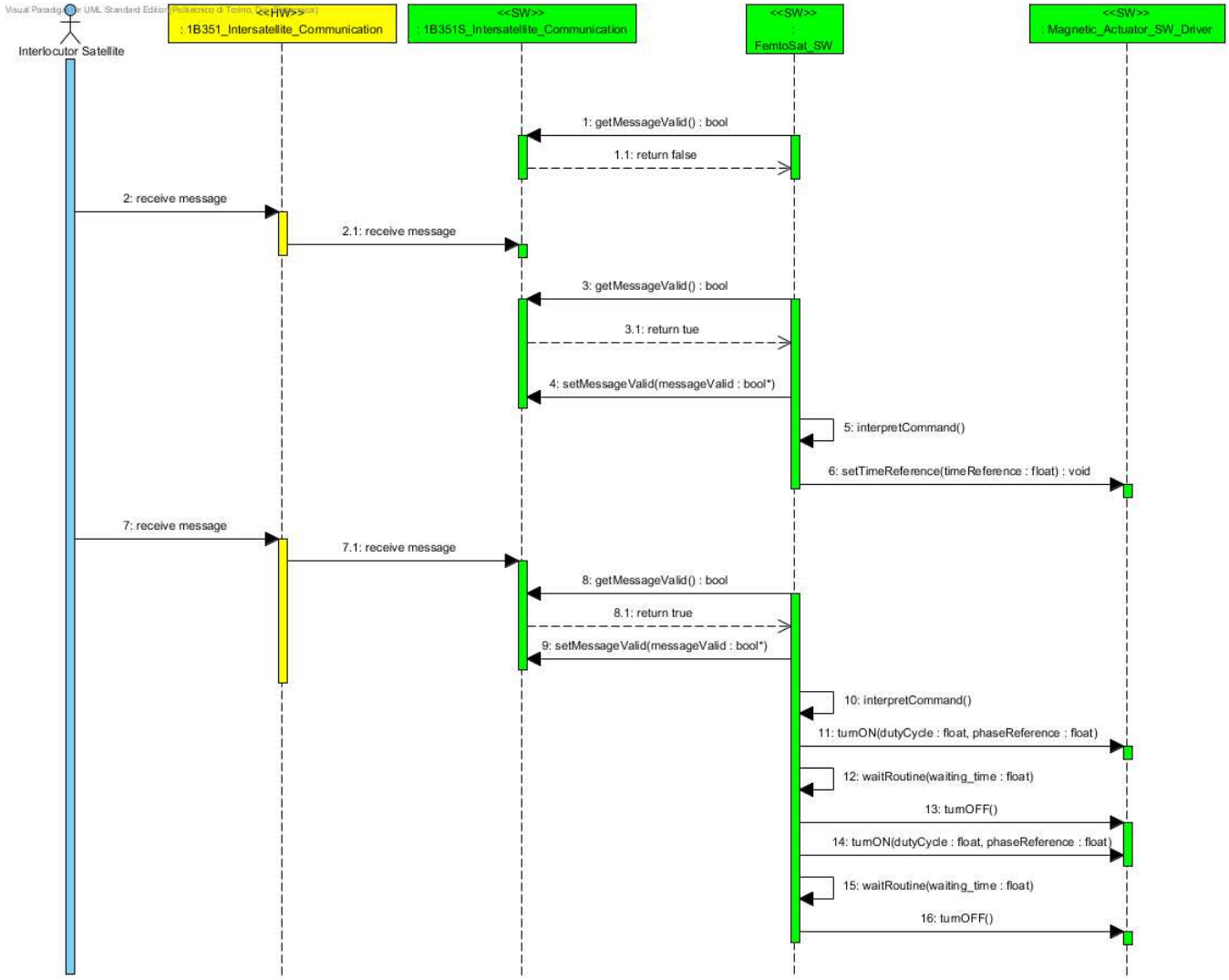


Figure 3.27 - Approach_Separate SLAVE sequence diagram

3.4 FemtoSat Analysis

3.4.1 Magnetic Force

The Attraction and Repulsion between the satellites is the consequence of Attractive/Repulsive force due to magnetic field generated by the electromagnets mounted on the two systems.

In order to analyze the dynamics of the system when the electromagnets are ON it is necessary to estimate the amplitude of magnetic force.

The solenoids can be seen as cylindrical magnets, in the *Figure 3.28* more details are shown.

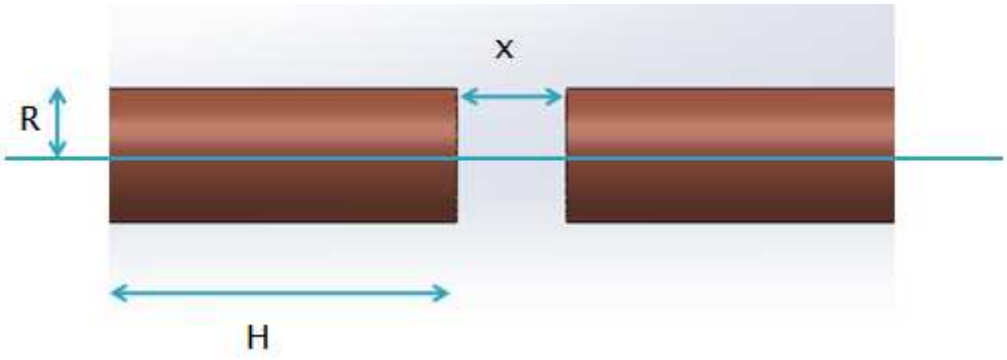


Figure 3.28 - Cylindrical Magnets configuration, each of them can be associated to an equivalent solenoid

The formulas to be used are the following [11]:

$$F(X) = \frac{\pi\mu_0}{4} M^2 R^4 \left[\frac{1}{X^2} + \frac{1}{(X+2H)^2} - \frac{2}{(X+H)^2}\right]$$

$$M = \frac{Nm}{V}$$

where:

- M is the magnetization (or magnetic polarization);
- N is the number of magnetic momentum in the volume
- V is the cylinder's volume
- m is the magnetic momentum of the magnet

- $\mu_0 = 1.256637*10^{-6}$ N/A^2 is the permeability in vacuum

Considering the equivalent solenoid we have:

$$M = \frac{NIA}{V} = \frac{NIA}{AH} = \frac{NI}{H}\left[\frac{A}{m}\right]$$

where

- N is the number of coils
- I is the current flowing inside the solenoid
- A is the solenoid section
- V is the solenoid volume
- H is the solenoid length

3.4.2 Dynamics Analysis

Since the solenoids are driven by a constant current, the magnetic force varies during motion then the acceleration varies as well. We can approximate the motion as uniformly accelerated in small intervals. Time law of uniformly accelerated motion is:

$$x = x_0 + v_0 t + \frac{1}{2}at^2$$

$$v = v_0 + at$$

The equations can be rewritten in the following form:

$$x = x_0 + v_0 t + \frac{1}{2}t\Delta v$$

$$v^2 = v_0^2 + 2a\Delta x$$

Considering a generic i-th interval and making explicit v_i and t_i we obtain:

$$v_i = \sqrt{v_{i-1}^2 + 2ai\Delta x}$$

$$t_i = \frac{\Delta x}{v_{i-1} + \frac{1}{2}(v_i - v_{i-1})}$$

In Appendix II there are some Matlab routines to implement time calculation.

Chapter IV

1B35 Specifications, Actors and Use Cases

4.1 InterSat Communication

It is the main use case diagram, it contains all use cases related to the communication between the OBC and Interlocutor Node via the 1B35_Intersatellite_Communication.

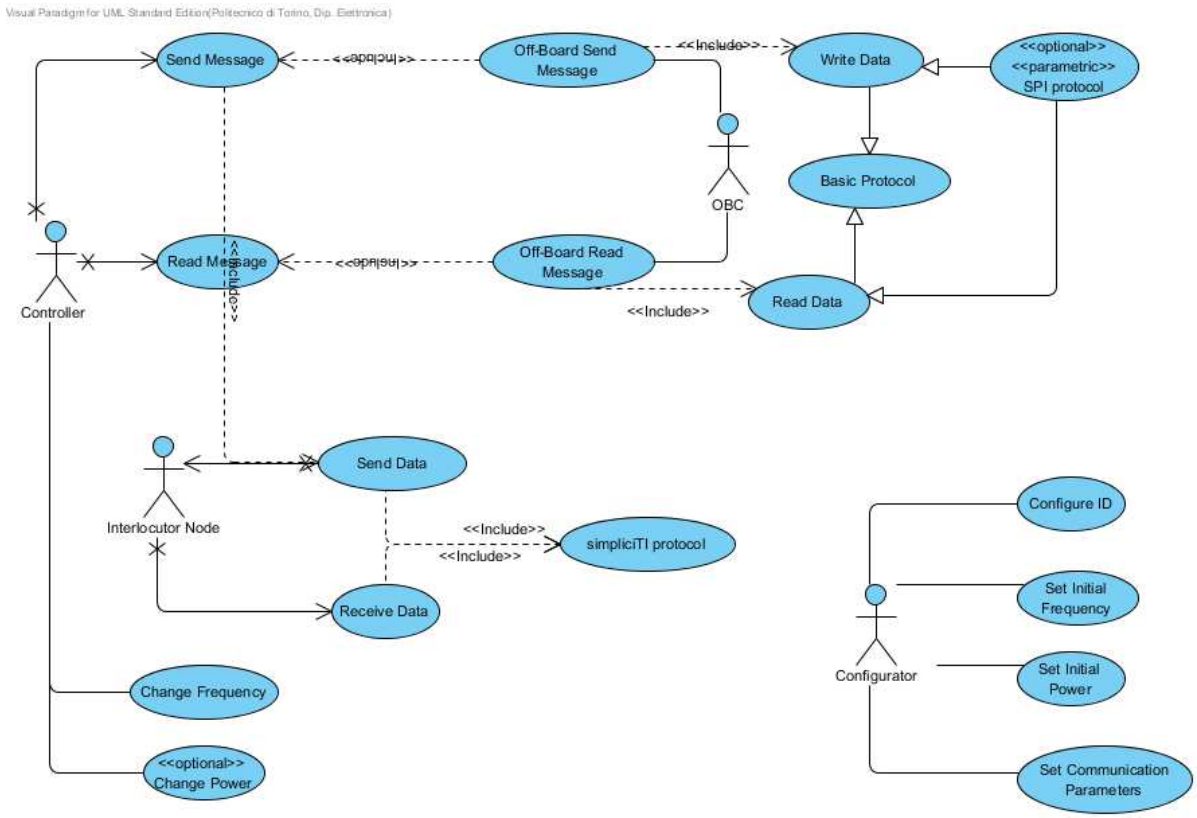


Figure 4.1 - InterSat Communication use case diagram

4.1.1 Use Case - Send Message

The Controller sends data to be transmitted to 1B35_Intersatellite_Communication which uses Send Data use case to communicate with Interlocutor Node identified by an ID specified by the Controller using set_address function.

The Send Message is implemented by calling the sendMessage function.

The use case will create a packet compatible with the simpliciTI protocol.

The reliability of the communication is guaranteed by the reception of an ack. which is then available to the Controller.

4.1.2 Use Case - Send Data

The 1B35_Intersatellite_Communication sends data to Interlocutor Node by means of simpliciTI protocol and the simpliciTI_API.

The reliability of the communication is guaranteed by the reception of an ack (forseen by simpliciTI protocol).

The OBC will pass a proper ID as parameter to

sendData function choosing among the ones available in ID_List basing on the target of communication.

The OBC will write the value of time_MASTER referring to its internal clock.

Messages will be sent at a frequency chosen among one of S-band_channels at programmable power, up to the the maximum power available and it will have the structure described in simpliciTI protocol.

Frequency can be set by the Configurator by means of the Set Initial Frequency and possibly changed by the OBC using Change Frequency.

Power can be set by the Configurator by means of the Set Initial Power and possibly changed by the OBC using Change Power.

4.1.3 Use Case - Read Message

The Controller reads the message provided by 1B35_Intersatellite_Communication using the function getMessage if it is valid. The message is then taken from the message_body. When the variable message_valid is true (high), it indicates that the message is valid.

4.1.4 Use Case - Read Data

The protocol to issue a command which reads between 1B to 256B of Designer's defined data from one Slave to the Master, as defined by the Slave address. Up to 255 Slaves can be addressed separately. It is the Designer's responsibility to ensure that data to be read is already available in the Slave when the command is issued.

4.1.5 Use Case - Receive Data

The 1B35_Intersatellite_Communication receives data sent by an Interlocutor Node by means of simpliciTI protocol and the simpliciTI_API.

Messages will be received at a frequency in the range 2.4-2.4835 GHz initially set by the Configurator by means of the Set Initial Frequency and then changed by the OBC using Change Frequency and it will have the structure described in simpliciTI protocol.

When the Message is received, 1B35_Intersatellite_Communication will use Measure Distance RF Slave to estimate the distance stored in internal variable dist_RF.

The 1B35_Intersatellite_Communication will write the value of time_SLAVE referring to its internal clock.

This use case periodically checks if data is received by the Interlocutor Node, removes preamble, verifies ID and if ID matches then stores it into the internal buffer message_body.

If CRC and length are correct it sets the internal flag message_valid.

The message message_body and the flag message_valid can be read by Controller using the Read Message use case.

Correctness of transmission is granted by an ACK which is sent by 1B35_Intersatellite_Communication.

4.1.6 Use Case - Configure ID

The Configurator set a code, chosen among ID_List, and stores it in ID to identify the 1B35_Intersatellite_Communication involved in actual operations.

4.1.7 Use Case - Set Communication Parameters

The Configurator sets:

- modulation scheme,
- format,
- frequency deviation,
- packet format

4.1.8 Use Case - Set Initial Frequency

The Configurator must choose one of the 4 channels described in S-band_channels in the bandwidth 2.4000 - 2.4835 GHz through which the 1B351_Intersatellite_Communication will communicate at boot (when powered-up and/or reset). The Configurator shall set initial frequency by setting INITIAL_CHANNEL parameter of class 1B351S_Intersatellite_Communication.

4.1.9 Use Case - Change Frequency

For any reason (channel occupied, data not arriving, and so on...) the Controller can change the channel in which the 1B351_Intersatellite_Communication are communicating. By calling change_frequency operation.

4.1.10 Use Case - Set Initial Power

The Configurator will set the initial power_TX of the 1B351_Intersatellite_Communication at the maximum value allowed by calling setPower function.

4.1.11 Use Case - Change Power

Sets the transmitting power of CC2510, as close as possible to the value of power defined by the Controller. The actual power will be the smallest value compatible with the transceiver, possibly not smaller than power.

Namely, if power is less than the maximum allowable, the actual transmission power will be smallest value larger than or equal to power.

This use case will start by calling setPower.

The implementation is optional.

4.1.12 Use Case - simpliciTI protocol

The Tile Processor actor is master, while an AraModule is a slave. Addressing is by means of Slave address per each slave.

The simpliciTI protocol supports the following actions (see the corresponding descriptions):

- Write Data - when the Tile Processor wants to transfer up to 256B of data to an AraModule;
- Read Data - when the Tile Processor wants to read up to 256B of data from an AraModule;
- Command Only - when the Tile Processor wants to deliver a data-less command to an AraModule.

Most data transfers contain:

- 2-24 Bytes Preamble Bytes
- 2-4 Byte Sync word
- an 8-bit data length field
- a 4-Byte Destination Address
- a4-Byte Source Address
- data. Length of data field is (length+1) bytes (1B to 64B)
- a 2-Byte RSSI
- an 16-bit CRC.
- ack/nack (only from Tile Processor to an AraModule)

If an error occurs (either wrong CRC or wrong length or no memory available, etc.) the an AraModule sets the (model element not found) flag.

By calling the Get Module Status use case, the Tile Processor can read details on the last error and clear the (model element not found) flag.

References

Type	Value
Folder	${Aramis_Progetto}\1B_Subsystem_Elements\1B4_On-Board_Data_Handling_Subsystem\1B45_Subsystem_Serial_Data_Bus\SimpliciTI
File	${Aramis_Progetto}\1B_Subsystem_Elements\1B4_On-Board_Data_Handling_Subsystem\1B45_Subsystem_Serial_Data_Bus\SimpliciTI\SimpliciTI Specification.pdf
File	${Aramis_Progetto}\1B_Subsystem_Elements\1B4_On-Board_Data_Handling_Subsystem\1B45_Subsystem_Serial_Data_Bus\SimpliciTI\SWRA221_SimpliciTI API.pdf

4.2 InterSat Distance

It contains all use cases needed to evaluate the intersatellite distance between 1B35_Intersatellite_Communication and the Interlocutor Node.

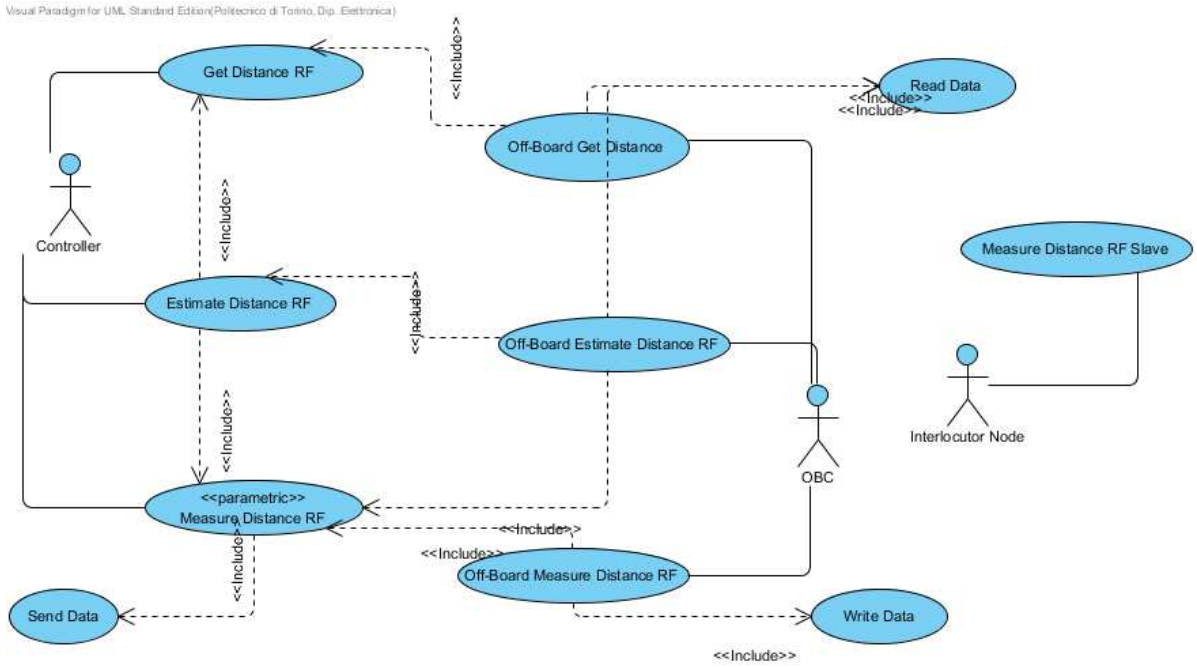


Figure 4.2 - InterSat Distance use case diagram

4.2.1 Use Case - Measure Distance RF

First the Controller uses Send Data where a DUMMY message is sent to the right Interlocutor Node identified by an ID given by the actor.

Through the RSSI technique, the power of the incoming acknowledgement (expected by simpliciTI protocol) is measured and then processed by calling the processDistanceRF function.

The result is stored in the internal memory (dist_RF variable).

The Measure Distance RF is implemented by calling the measureDistanceRF function.

4.2.2 Use Case - Measure Distance RF Slave

When the 1B35_Intersatellite_Communication receives a Message sent by Interlocutor Node it will calculate the actual dist_RF basing on the value of incoming power_TX through processDistanceRF.

4.2.3 Use Case - Get Distance RF

The Controller reads the value of distValid, if high (1), it reads the other attributes of Distance_RF.

The Get Distance RF is implemented by calling the getDistanceRF function.

4.2.4 Use Case - Estimate Distance RF

First the Controller uses Measure Distance RF to trigger the distance measurement to the right Interlocutor Node identified by an ID given by the actor, then starts a loop in which periodically the value of dist_RF and its validity are evaluated by using Get Distance RF .

When distValid is true, it indicates a valid measurement and causes the loop termination.

The Estimate Distance RFis implemented by calling the estimateDistanceRF function.

Chapter V

1B35 Communication Module Design

The general structure of 1B35_Intersatellite_Communication is divided in two parts:

- hardware (1B351_Intersatellite_Communication)
- software (1B351S_Intersatellite_Communication)

which are described in the corresponding diagrams in the next paragraphs. Among all objects used by the software root class, there is an instance of 1B351_Intersatellite_SW_drivers which contains the main functions to send/receive data and which directly interfaces with SimpliciTi API [8].

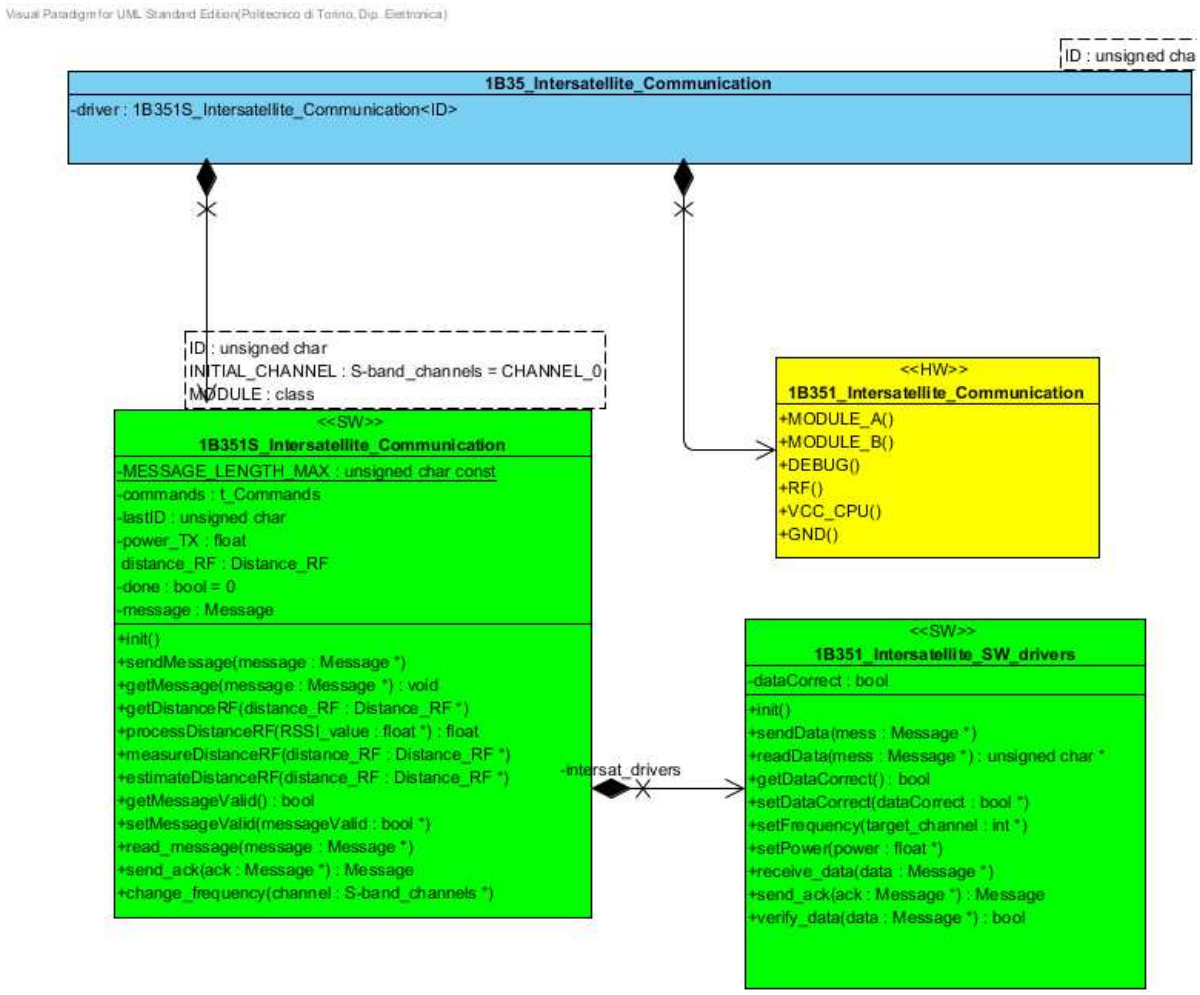


Figure 5.1 - 1B35 Intersatellite Communication main class diagram

5.1 Design choices

1B35 module is developed with the main target of implementing an efficient and reliable intersatellite communication.

Obviously this communication has to be wireless, and respecting AraMiS specifications and philosophy, it should be as light and low-power as possible. In fact it has to be mounted on a very small satellite, which can provide a little amount of electrical power for all of its devices. After studying and considering many wireless devices and protocols, our choice fell on CC2510 from Texas Instruments.

This device is an extremely low power transceiver which is developed for communicating on the 2.4 GHz frequencies, and offers a series of goodpros:

- Very high sensitivity: -103 dBm at 2.4 kBaud
- Programmable data rate up to 500 kBaud
- Programmable output power up to 1dBm for all supported frequencies
- Frequency range 2400 – 2483.5 MHz
- Low current consumption: RX- 17.1 mA @ 2.4 kBaud, TX- 16 mA @ -6 dBm output power
- Digital RSSI support

Last two are the most important features, the low current consumption because of our need of power saving, and the RSSI support because of the final aim of my thesis.

1B35 module in fact, hasn't only the duty of communicating, but also the one of helping the satellites calculating their reciprocal distance. RSSI technique is the best way to do that as we will see later.

CC2510 transceiver is also a perfect match because Texas Instruments already developed a communication protocol based on these devices i.e. the SimpliciTI protocol.

5.1.1 SimpliciTi Protocol

SimpliciTI protocol is intended to support customer development of wireless end user devices in environment in which the network support is simple and the customer desires a simple means to do messaging over air.

The protocol is oriented around application peer-to-peer messaging. In most cases the peers are linked together explicitly. However, simpliciTI protocol will support scenarios in which explicit linking between pairs of devices is neither needed nor desired. The simpliciTI_API provides the means to initialize the network, link devices, and send messages.

SimpliciTI layers

SimpliciTIprotocol is organized in 3 layers : Data Link/Physic, Network, and Application.

Application

This is the only layer that the developer needs to implement. It is where he develops his application (to manage sensors for example), and implements network communication, by using SimpliciTI network APIs or network applications.

Note that it is in this layer that the developer needs to implement reliable transport if required, as there is no Transport layer.

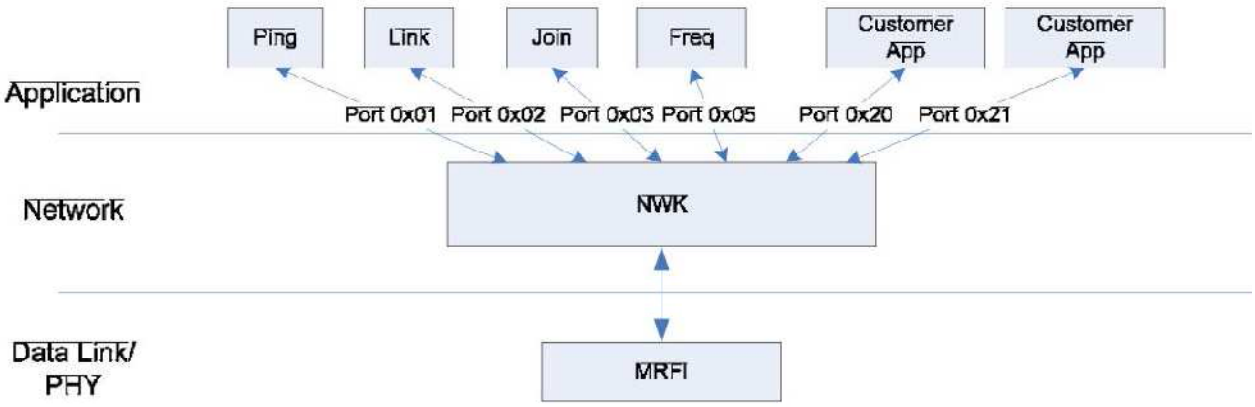


Figure 5.2 - SimpliciTI logical layers

Network

This layer manages the Rx and Tx queues and dispatches frames to their destination. The destination is always an application designated by a Port number.

Network applications are internal peer-to-peer objects intended to manage network. They work on a predefined port and are not intended to be used by the developer (except Ping for debugging purposes)

Their usage depends on the SimpliciTI device type. These applications are:

Ping (Port 0x01): to detect the presence of a specific device.

Link (Port 0x02): to support the link management of two peers.

Join (Port 0x03): to guard entry to the network in topologies with APs.

Security (Port 0x04): to change security information such as encryption keys and encryption context.

Freq (Port 0x05): to perform change channel, change channel request or echo request.

Mgmt (Port 0x06): general management port to be used to manage the device.

Source code files for network layer are located in /Components/simpliciti.

Data Link/Physic

This layer may be divided in 2 entities:

BSP (Board Support Package): to abstract the SPI interface from the NWK layer calls that interactwith the radio (/Components/mrfi);

MRFI (Minimal RF Interface): to encapsulate the differences between supported hardware radios, toward the network layer (/Components/bsp).

SimpliciTIprotocol initialization involves three stages of initialization: board, radio, and stack. Board initialization (BSP) is deliberately separated from the radio and stack initialization. The radio and stack initialization occur as a result of the SimpliciTI initialization call. The board initialization is a separate invocation not considered part of the SimpliciTI API but it is noted here for completeness.

The BSP initialization is partitioned out because customers may already have a BSP for their target devices. Making the BSP initialization explicit in the SimpliciTI distribution makes it easier to port to another target.

Board Initialization

SimpliciTI supports a minimal board-specific BSP. The BSP scope includes GPIO pin configuration for LEDs, switches, and a counter/timer used for protocol chores. It also includes SPI initialization for the dual-chip RF solutions.

Radio Initialization

Radio registers are populated and the radio is placed in the powered, idle state. Most of the radio registers are based on exported code from SmartRF Studio. The default channel is set with the first entry in the channel table.

Stack Initialization

All data structures and network applications are initialized. In addition the stack issues a Join request on behalf of the device. The Join request will fail in topologies in which there is no Access Point. This is expected in this topology and is not an error condition.

We will use SimpliciTI protocol to communicate through CC2510, this device automatically sends RSSI value. This value is fundamental for our target, and to understand it we need to see the RSSI technique in details.

5.1.2 RSSI Technique

Received signal strength indicator (RSSI) is a measurement of the power present in a received radio signal.

RSSI is a generic radio receiver technology metric, which is usually invisible to the user of the device containing the receiver, but is directly known to users of wireless networking of IEEE 802.11 protocol family.

In an IEEE 802.11 system RSSI is the relative received signal strength in a wireless environment. RSSI is an indication of the power level being received by the antenna. Therefore, the higher the RSSI value, the stronger the signal.

This technique is based on Friis equation:

$$\frac{P_r}{P_t} = \frac{G_r}{G_t}\left(\frac{\lambda}{4\pi D}\right)^2$$

From which we get

$$D = \frac{\lambda}{4\pi}\sqrt{\frac{P_t G_r G_t}{P_r}}$$

Now it is clear how things will work: 1B35 Module will implement a communication between the two interlocutor nodes, and each time a message is sent from a node to another, an RSSI value is present in the message itself. This will grant the possibility of getting the distance between the nodes, no matter what the content of the message is.

Also, our system will store a reference of time inside the message. This reference is not absolute, it means that the internal clocks of the nodes don't need to be synchronized, in fact there are two fields in the message, one contains the time referred to the clock of the source of the message, another one contains the time referred to the clock of the destination; since the message also contains the IDs of the source and of the destination, it is simple to obtain the time I need to refer to.

The reference of time is important because I could need to calculate the distance more than once in a small gap of time, in this case it's unuseful to send a new message because it would need time and power, and sometimes we can avoid that. In fact if the last message is very near in time, we can suppose that the node is still in the same position of the point where it sent the last message. If we accept it, we can calculate the distance again without communicating.

5.2 Class Diagrams

5.2.1 -1B35 Intersatellite Communication main class diagram

Class - 1B35_Intersatellite_Communication

The system allows the exchange of signals with the Interlocutor Node. It works either as a transmitter or a receiver and for this reason it has two separate signal processing chain (one for transmission and the other one for reception). Since one section is shared between the two channels, a half-duplex communication is implemented (transmission and reception separated in time and using the same channel).

In Send Data use case , starting from a digital sequence, it properly build an analog signal which travel on the channel (free space in the case study).

Similarly, in the Receive Data use case, it takes the analog signal intercepted by the antenna and it extracts the digital information.

It also provides distance measurement with respect to Interlocutor Node by using (model element not found) technique.

Template Parameters: ID

References

Type	Value
Folder	${Aramis_Progetto}\FemtoSat

Attributes

Signature: -driver : 1B351S_Intersatellite_Communication

Class - 1B351_Intersatellite_Communication

The class contains all the hardware design of the 1B35_Intersatellite_Communication.

It acts as a slave on both MODULE_A and MODULE_B.

Operations

Signature: MODULE_A()

Signature: MODULE_B()

Signature: DEBUG()

Signature: RF()

Signature: VCC_CPU()

Signature: GND()

Class - 1B351S_Intersatellite_Communication

It is the core system of the 1B35_Intersatellite_Communication, it manages all communications. Then basing on priorities it takes decisions, governs and coordinates all modules inside the 1B35_Intersatellite_Communication.

Template Parameters: ID, INITIAL_CHANNEL, MODULE

Attributes

-MESSAGE_LENGTH_MAX : unsigned char const

Message maximum length according to simpliciTI protocol.

Signature: -commands : t_Commands

An instance of t_Commands class.

Signature: -lastID : unsigned char

Stores the ID of the last Interlocutor Node for which the distance has been measured.

Signature: -power_TX : float

It's the value of power chosen by OBC for current transmission.

It's firstly set by Set Initial Power and then changed by Change Power.

Signature: distance_RF : Distance_RF

Signature: -done : bool = 0

Is a flag that tells if the message is still useful or not.

It's just 1 bit.

Signature: -message : Message

Signature: +lambda : float

It's the wavelength referred to the transmitting frequency. The frequency is variable but we can assume the centerband as 2.4 GHz since the variation between a channel and another is relatively small.

Given this we can approximate the lambda with

lambda = c / f = 125 mm

Signature: +Gain : float

Is the gain of the antenna and it is the same in transmission and in reception.

It is indicated in the formulas as Gt or Gr.

Its value is set by Configurator.

Operations

Signature: init()

Initializes internal variables, including:

- lastID = 0;

Configures initial frequency of CC2510

Signature: sendMessage(message : Message)

It performs all operations needed to send a message through the 1B35_Intersatellite_Communication which writes its own ID in sourceID and the ID of the Interlocutor Node to whom the message is sent, in destID; both ids are chosen among ID_List.

The message has the structure described in Message.

Signature: getMessage(message : Message) : void

Returns the last received message from Interlocutor Node.

After the message is received, the 1B351_Intersatellite_Communication checks that the destID inside the message is the same of its own ID

If this is true, basing on preamble_length reads the preamble and verifies that everything is ok.

At last, it reads message_body knowing when to stop by reading the value message_length.

Once the message is verified to be valid, the value message_valid is raised to 1.

Signature: getDistanceRF(distance_RF : Distance_RF)

Returns different parameters related to the distance:

- dist_RF is the actual value of distance between the 1B35_Intersatellite_Communication and the Interlocutor Node
- ID_MASTER is the ID of the master
- ID_SLAVE is the ID of the slave
- power is the value of power of the transmitter
- time_MASTER stores the time referring to the internal clock of the master
- time_SLAVE stores the time referring to the internal clock of the slave, it's not synchronized with the master clock

Signature: processDistanceRF(RSSI_value : float) : float

Starting from the value read by (model element not found) technique it process data,computes distance and stores it in dist_RF and returns dist_RF value. This function is called inside measureDistanceRF

The formula will be

$$D = \frac{\lambda}{4\pi}\sqrt{\frac{P_t G_r G_t}{P_r}}$$

Since we are in open space we can, with an acceptable margin of error, consider only the direct wave in the FRIIS equation:

$$\frac{P_r}{P_t} = G_r G_t \left(\frac{\lambda}{4\pi D}\right)^2$$

To be more precise we would have to calculate at least the reflections on the satellite body and the diffraction due to the solar panels of the satellites themselves. (AGGIUNGI MTIVO)

Signature: measureDistanceRF(distance_RF : Distance_RF)

When the Controller calls this function all operations needed to measure the dist_RF value start (Measure Distance RF use case).

Signature: estimateDistanceRF(distance_RF : Distance_RF)

It combines measureDistanceRF and getDistanceRF functions in order to estimate the value of dist_RF.

The measureDistanceRF triggers the measurement, while getDistanceRF read the value in a loop, until the distValid is true.

Signature: getMessageValid() : bool

Getter method for message_valid

Signature: setMessageValid(messageValid : bool)

Setter method for message_valid

Signature: read_message(message : Message)

Asks through the SPI to read the Message and return the message_body

Signature: send_ack(ack : Message) : Message

Sends an ACK to certificate that the message is sent / received correctly and all required operations are completely done.

Signature: check_distValid(distValid : bool)

Checks the value of distValid

Signature: set_distValid(distValid : bool)

Sets distValid high when the measurement is over in the following conditions:

- ID_MASTER is the same of the 1B351S_Intersatellite_Communication ID
- ID_SLAVE is one among ID_List
- time_MASTER is not to far from actual 1B351S_Intersatellite_Communication time (less than 10 minutes

Signature: set_initial_freq() : S-band_channels

Starts the Set Initial Frequency sequence, passes to the 1B351S_Intersatellite_Communication the band among S-band_channels on which the communication will take place.

Signature: receive_data(data : Message)

Stores Message data

Signature: change_frequency(channel : S-band_channels)

This function chooses a value from S-band_channels

Class - 1B351_Intersatellite_SW_drivers

The class contains all the software routines which works on simpliciTI protocol.

Attributes

Signature: -dataCorrect : bool

It indicates if there are incoming data and if they are correct.

Signature: +target_freq : S-band_channels

the frequency at which the 1B351_Intersatellite_SW_drivers trie to communicate

Signature: -target_power : float

Is the target power passed by the Controller

Signature: -i : int

An int variable, useful for some cycle

Operations

Signature: init()

Initializes the class and the variables needed inside it.

Signature: sendData(mess : Message)

It contains all operations needed to send data through the Send Data use case.

Signature: readData(mess : Message) : unsigned char

Returns the last received data from Interlocutor Node. The message has already been verified for correctness (CRC)

Signature: getDataCorrect() : bool

Getter method for dataCorrect

Signature: setDataCorrect(dataCorrect : bool)

Setter method for dataCorrect

Signature: setFrequency(target_channel : int)

The implementation is optional. The function is used to change the frequency among the ones available in S-band_channels.

Signature: setPower(power : float)

The function is used to set EIRP of the transmitted Message, as close as possible to power (Change Power use case). The actual power will be the smallest value compatible with the transceiver, possibly not smaller than power.

Namely, if power is less than the maximum allowable, the actual transmission power will be smallest value larger than or equal to power.

The implementation is optional.

Signature: receive_data(data : Message)

This function is needed to prepare the 1B35_Intersatellite_Communication to the reception of a packet from the Interlocutor Node.

The 1B351S_Intersatellite_Communication will have to recognize the various parts of the message (preamble, mess...) and will store them in the proper memory locations for future use.

At the end of the operations, data will be verified and a flag dataCorrect will be set through setDataCorrect in order to allow the sending of an ack, that will certify the receiving of the proper message.

Signature: send_ack(ack : Message) : Message

Sends an ACK to certificate that the message is sent / received correctly and all required operations are completely done.

Signature: verify_data(data : Message) : bool

Verifies that data are correct by getting sure that dataCorrect is set high (1).

5.2.2 -1B351 Intersatellite Communication Hardware

The diagram contains all hardware classes of 1B35_Intersatellite_Communication that is:

- 1B351_Intersatellite_Communication
- Bk1B31B1W_OBRF_CC2510
- 1B480_Null_Modem_SPI

1B351_Intersatellite_Communication is the root class, it is taken from 1B35_Intersatellite_Communication project [6]. The module communicates with external environment by means of two standard modules developed for AraMis (MODULE_A, MODULE_B) and a DEBUG interface. It is important to notice that the antenna is an external element not consider in 1B35_Intersatellite_Communication design, for this reason there is an RF pin in the module interface.

Bk1B31B1W_OBRF_CC2510 is taken from 1B31_On_Board_RF_Module project [9], which has as core component CC2510 system on chip with in addition some electronics in order to let the system work.

1B480_Null_Modem_SPI is taken from 1B48_Module_Interface [10] which is used to implement null modem connection to configure 1B35_Intersatellite_Communication as slave.

For more details refers to schematics and PCB layout in *Appendix V*.

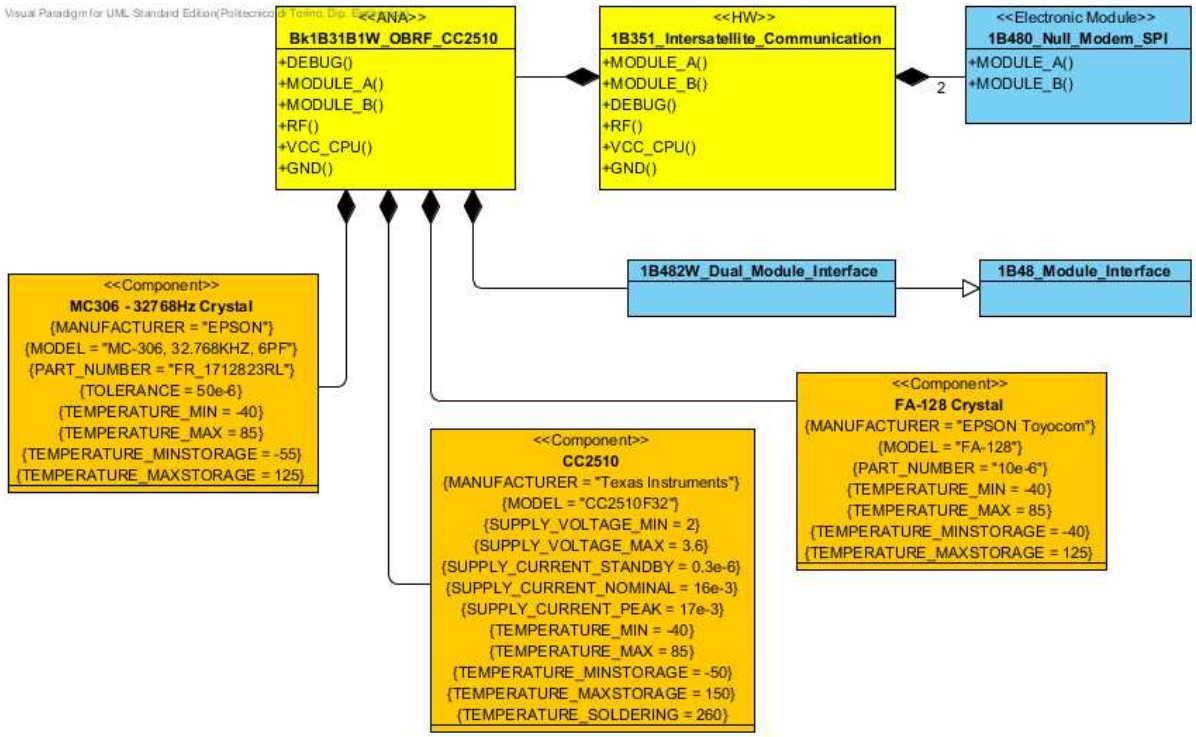


Figure 5.3 - 1B351 Intersatellite Communication Hardware class diagram

Class - Bk1B31B1W_OBRF_CC2510

Transceiver module for 437MHz transmission. It provides FSK modulation/demodulation capabilities for the information bit-stream and includes an external oscillator to achieve low frequency drift.

It acts as a master on both MODULE_A and MODULE_B slots.

References

Type	Value
Folder	${Aramis_Progetto}\1B_Subsystem_Elements\1B3_TT&C_Telecommunication_Subsystem\1B31_On_Board_RF_Module\Bk1B31B1_OBRF_2_4GHz\Bk1B31B1W_OBRF_CC2510

Operations

Signature: DEBUG()

Debug bus for CC2510. Contains: RESET, DEBUG_DATA, DEBUG_CLOCK, 3V3_SUPPLY.

Signature: MODULE_A()

Signature: MODULE_B()

Signature: RF()

Signature: VCC_CPU()

Signature: GND()

Class - CC2510

References

Type	Value
File	${1C54_Datasheets}\cc2510f32.pdf

Class - 1B480_Null_Modem_SPI

Null model circuit for SPI interface to connect two 1B48_Module_Interfaces which are both master.

It connects with simple wires:

- D0_RX_SOMI of MODULE_A with D1_TX_SIMO of MODULE_B;
- D1_TX_SIMO of MODULE_A with D0_RX_SOMI of MODULE_B;
- D4_CLK of MODULE_A with D4_CLK of MODULE_B.

such that the system connected to MODULE_A can talk with the system connected to MODULE_B.

One of them shall be configured as master, while the other shall be configured as slave.

References

Type	Value
Folder	${Progetto}\1B_Subsystem_Elements\1B4_On-Board_Data_Handling_Subsystem\1B48_Module_interface\1B480_Null_Modem_SPI

Operations

Signature: MODULE_A()

Signature: MODULE_B()

Class - 1B48_Module_Interface

References

Type	Value
File	${Aramis_Progetto}\1B_Subsystem_Elements\1B4_On-Board_Data_Handling_Subsystem\1B48_Module_interface\1B481_Module_Interface\1B481 Module Interface ICD V1_1.doc
Folder	${Aramis_Progetto}\1B_Subsystem_Elements\1B4_On-Board_Data_Handling_Subsystem\1B48_Module_interface\1B481_Module_Interface

Class - MC306 - 32768Hz Crystal

References

Type	Value
File	${1C54_Datasheets}\MC406-crystal.PDF

Class - FA-128 Crystal

References

Type	Value
File	${1C54_Datasheets}\FA-128_crystal.pdf

5.2.3 -1B351S Intersatellite Communication Software

The diagram contains all software classes of 1B35_Intersatellite_Communication

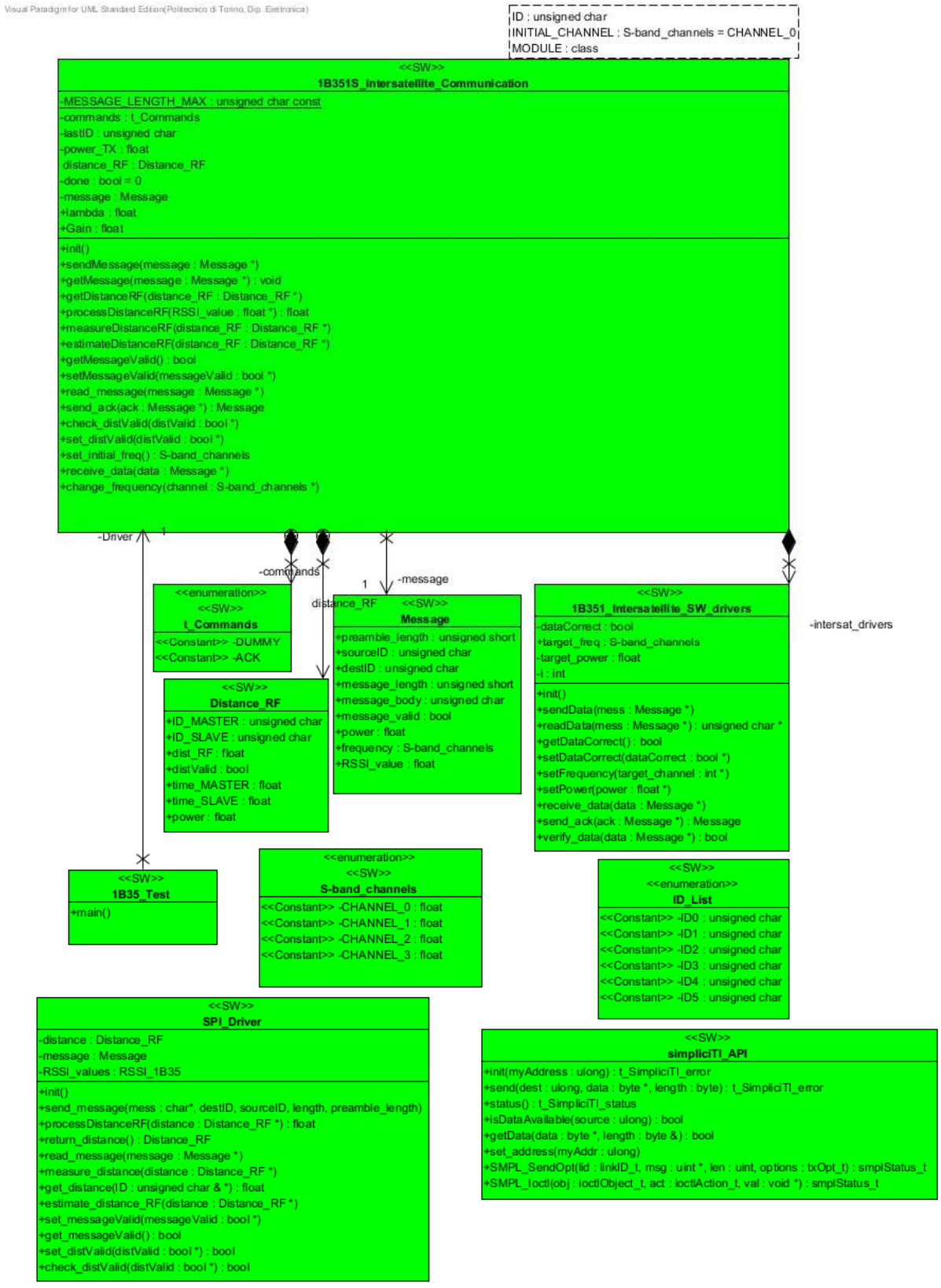


Figure 5.4 - 1B351S Intersatellite Communication Software class diagram

Class - 1B351_Intersatellite_SW_drivers

The class contains all the software routines which works on simpliciTI protocol.

Attributes

Signature: -dataCorrect : bool

It indicates if there are incoming data and if they are correct.

Signature: +target_freq : S-band_channels

the frequency at which the 1B351_Intersatellite_SW_drivers trie to communicate

Signature: -target_power : float

Is the target power passed by the Controller

Signature: -i : int

An int variable, useful for some cycle

Operations

Signature: init()

Initializes the class and the variables needed inside it.

Signature: sendData(mess : Message)

It contains all operations needed to send data through the Send Data use case.

Signature: readData(mess : Message) : unsigned char

Returns the last received data from Interlocutor Node. The message has already been verified for correctness (CRC)

Signature: getDataCorrect() : bool

Getter method for dataCorrect

Signature: setDataCorrect(dataCorrect : bool)

Setter method for dataCorrect

Signature: setFrequency(target_channel : int)

The implementation is optional. The function is used to change the frequency among the ones available in S-band_channels.

Signature: setPower(power : float)

The function is used to set EIRP of the transmitted Message, as close as possible to power (Change Power use case). The actual power will be the smallest value compatible with the transceiver, possibly not smaller than power.

Namely, if power is less than the maximum allowable, the actual transmission power will be smallest value larger than or equal to power.

The implementation is optional.

Signature: receive_data(data : Message)

This function is needed to prepare the 1B35_Intersatellite_Communication to the reception of a packet from the Interlocutor Node.

The 1B351S_Intersatellite_Communication will have to recognize the various parts of the message (preamble, mess...) and will store them in the proper memory locations for future use.

At the end of the operations, data will be verified and a flag dataCorrect will be set through setDataCorrect in order to allow the sending of an ack, that will certify the receiving of the proper message.

Signature: send_ack(ack : Message) : Message

Sends an ACK to certificate that the message is sent / received correctly and all required operations are completely done.

Signature: verify_data(data : Message) : bool

Verifies that data are correct by getting sure that dataCorrect is set high (1).

Class - Message

Is the set of data that can be exchanged between 1B35_Intersatellite_Communication and Interlocutor Node.

Attributes

Signature: +preamble_length : unsigned short

Indicates the length of the preamble of the message, that is to say the length of that part of the message which contains information about the message, not the real message body.

Signature: +sourceID : unsigned char

It's the ID of the source of the message. It's a value chosen among one of the ID present in ID_List.

Signature: +destID : unsigned char

It's the ID of the destination of the message. It is a value among the ones present in ID_List.

Signature: +message_length : unsigned short

It's the length of the message itself.

Signature: +message_body : unsigned char

This attribute contains the main body of the message.

It can contain

- Distance_RF
- one of t_Commands
- a generic message

Signature: +message_valid : bool

It is a flag which specifies if the message is valid or not.

The variable assumes "true" as value when there is an incoming message and the data are correct.

The variable assumes "false" as value either when the incoming message is read by the OBC (Read Message use case) so data become obsolete or when data are corrupted.

It's a boolean value which indicates if the message is correct or corrupted.

- its value is 1 if message is valid
- its value is 0 instead when the message is corrupted

Signature: +power : float

This is power_TX. expressed in Watt.

Signature: +frequency : S-band_channels

Indicates the frequency of transmission.

It's one of the values among the ones listed in S-band_channels

Signature: +RSSI_value : float

Class - Distance_RF

This class contains information related to distance measurement.

Attributes

Signature: +ID_MASTER : unsigned char

This is the ID of the 1B35_Intersatellite_Communication master which has requested the measurement.

Signature: +ID_SLAVE : unsigned char

This is the ID of the Interlocutor Node slave which is receiving the (model element not found).

Signature: +dist_RF : float

Contains the last measured distance as defined by Measure Distance RF use case.

Unit is meters.

Initial value shall be 0, which means that no distance has been measured so far.

Signature: +distValid : bool

The flag which indicates if the dist_RF value is valid i.e. the measurement is carried out correctly.

Signature: +time_MASTER : float

This is the time relative to the internal clock of the 1B35_Intersatellite_Communication expressed in seconds.

Signature: +time_SLAVE : float

This is the time of the internal clock of the Interlocutor Node expressed in seconds.

Signature: +power : float

This is power_TX.

Class - t_Commands

The enumeration contains a list of all possible commands and status notification that the 1B35_Intersatellite_Communication can receive.

Attributes

Signature: -DUMMY

Dummy command.

Signature: -ACK

It indicates that the communication is carried out without errors.

Class - S-band_channels

Is the enumeration list of all possibles channels of frequencies on which the CC2510 can establish a communication.

Attributes

Signature: -CHANNEL_0 : float

This channel refers to channel_3 (see SimpliciTI_Channel_Table_Information.pdf), has a center band in 2.4257 GHz. and a bandwidth of 250KHz, corresponds to the IEEE channel 15

Signature: -CHANNEL_1 : float

This channel refers to channel_103 (see SimpliciTI_Channel_Table_Information.pdf), has a center band in 2.4508 GHz and a bandwidth of 250KHz., corresponds to the IEEE channel 20

Signature: -CHANNEL_2 : float

This channel refers to channel_202 (see SimpliciTI_Channel_Table_Information.pdf), has a center band in 2.4755 GHz. and a bandwidth of 250KHz, corresponds to the IEEE channel 25

Signature: -CHANNEL_3 : float

This channel refers to channel_212 (see SimpliciTI_Channel_Table_Information.pdf), has a center band in 2.4807 GHz and a bandwidth of 250KHz., corresponds to the IEEE channel 26

Class - ID_List

The list of all possible IDs that Configurator can assign to 1B35_Intersatellite_Communication and that 1B35_Intersatellite_Communication can use to address messages.

Attributes

Signature: -ID0 : unsigned char

Signature: -ID1 : unsigned char

Signature: -ID2 : unsigned char

Signature: -ID3 : unsigned char

Signature: -ID4 : unsigned char

Signature: -ID5 : unsigned char

5.3 Sequence Diagrams

5.3.1 Send Message

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then the Controller calls the sendMessage function. As a result of this, 1B351S_Intersatellite_Communication uses Send Data use case with Message as payload of the communication. After data are sent, an ack is returned to 1B351S_Intersatellite_Communication and then to Controller.

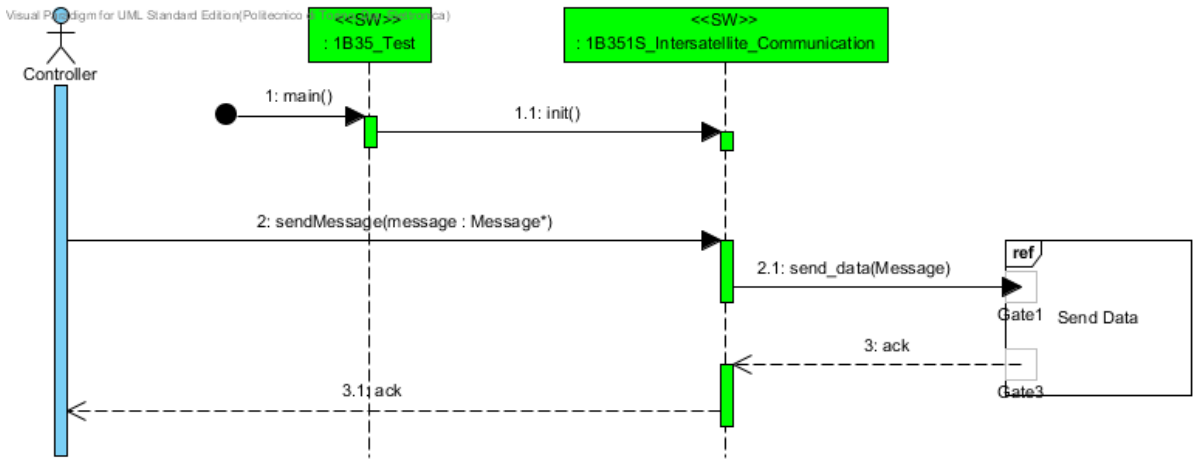


Figure 5.5 - Send Message sequence diagram

5.3.2 Send Data

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then simpliciTI_API is initializes as well. Now the Controller calls sendData, which causes a call to send. From here the hardware 1B351_Intersatellite_Communication is involved in the sequence and uses Bk1B31B1W_OBRF_2_4GHz for sending data to the Interlocutor Node.

When data are received an ack is sent back to the 1B351_Intersatellite_SW_drivers which calls verify_data function and gives the ack back to the Controller

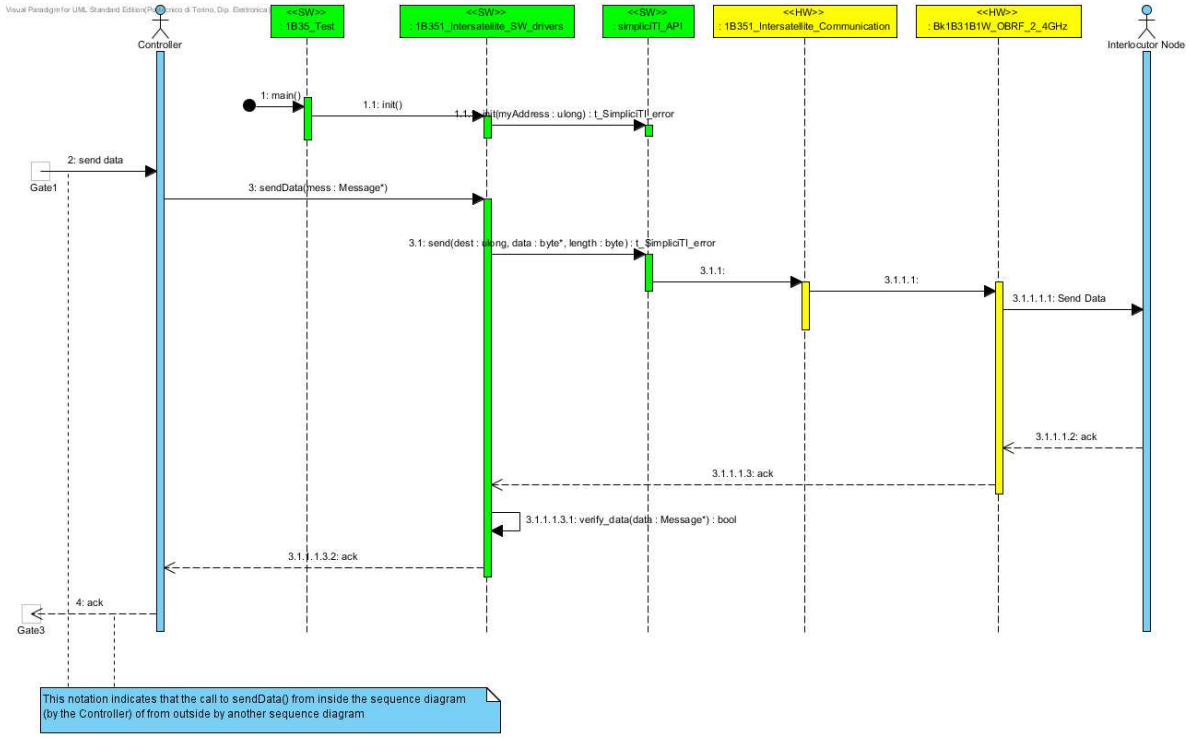


Figure 5.6 - Send Data sequence diagram

5.3.3 Read Message

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then the Controller periodically checks the validity of the message by calling getMessageValid. If message_valid is true then it calls getMessage of 1B351S_Intersatellite_Communication which calls setMessageValid as well passing false (0) as parameter.

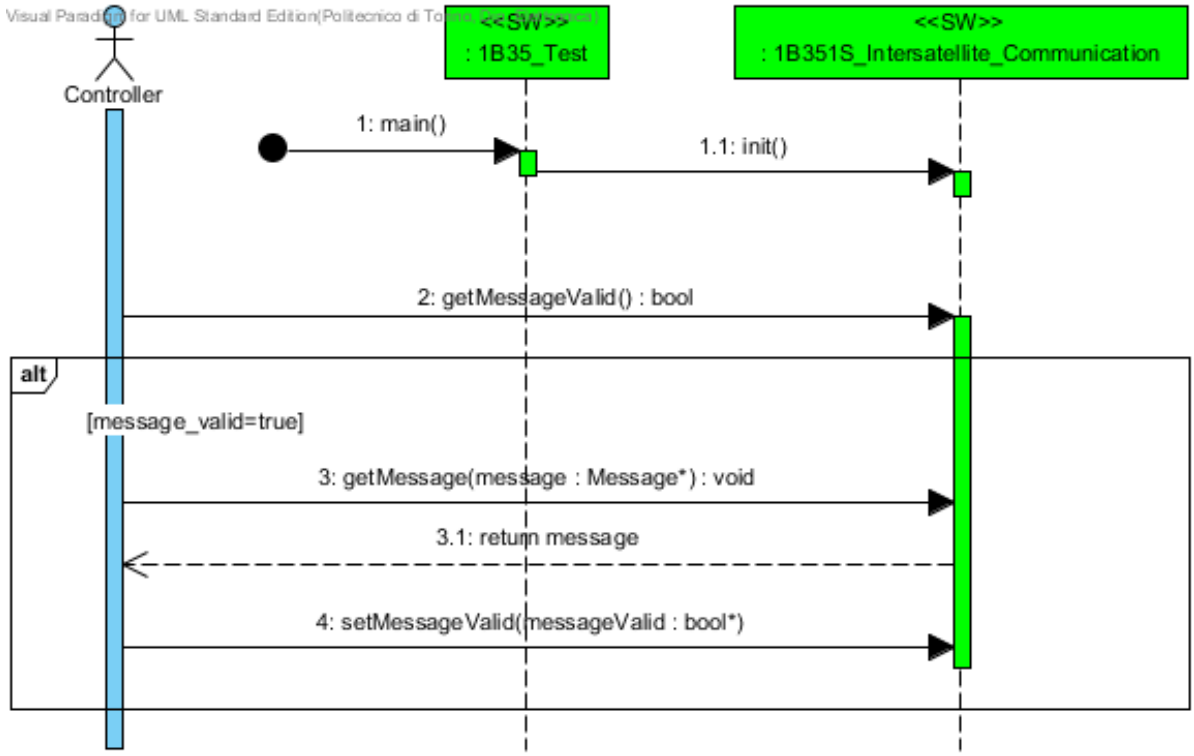


Figure 5.7 - Read Message sequence diagram

5.3.4 Receive Data

The Interlocutor Node is sending data to Bk1B31B1W_OBRF_2_4GHz. When data are received, through 1B351_Intersatellite_Communication, an interrupt is launched by simpliciTI_API that sets high (1) the flag dataCorrect and 1B351_Intersatellite_SW_drivers uses receive_data to write data inside Message.

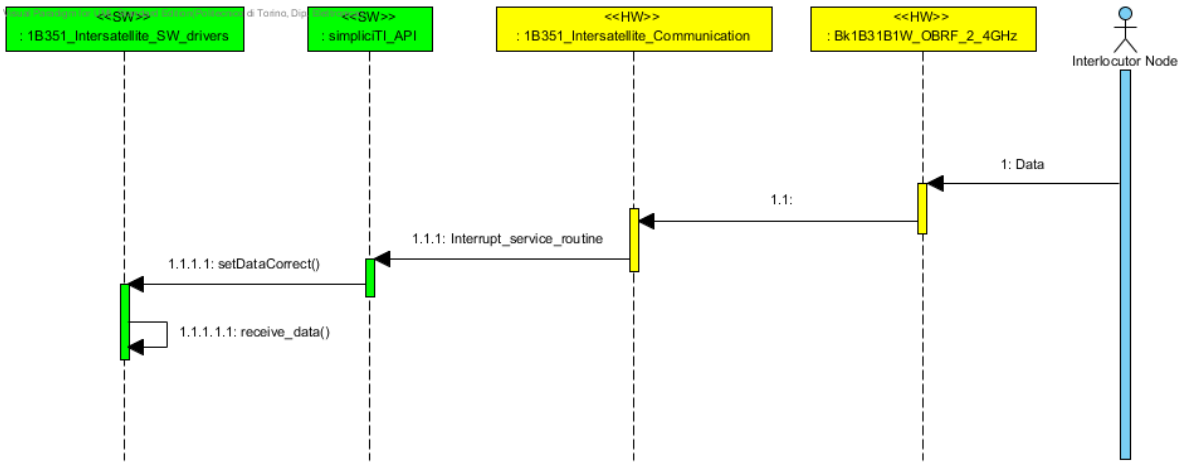


Figure 5.8 - Receive Data sequence diagram

5.3.5 Read Data

1B351S_Intersatellite_Communication periodically checks on 1B351_Intersatellite_SW_drivers if dataCorrect is true.

When it is true, reads the data through readData function and then uses setDataCorrect to set dataCorrect as false (0).

At last, uses setMessageValid function to set the flag messageValid as true (1).

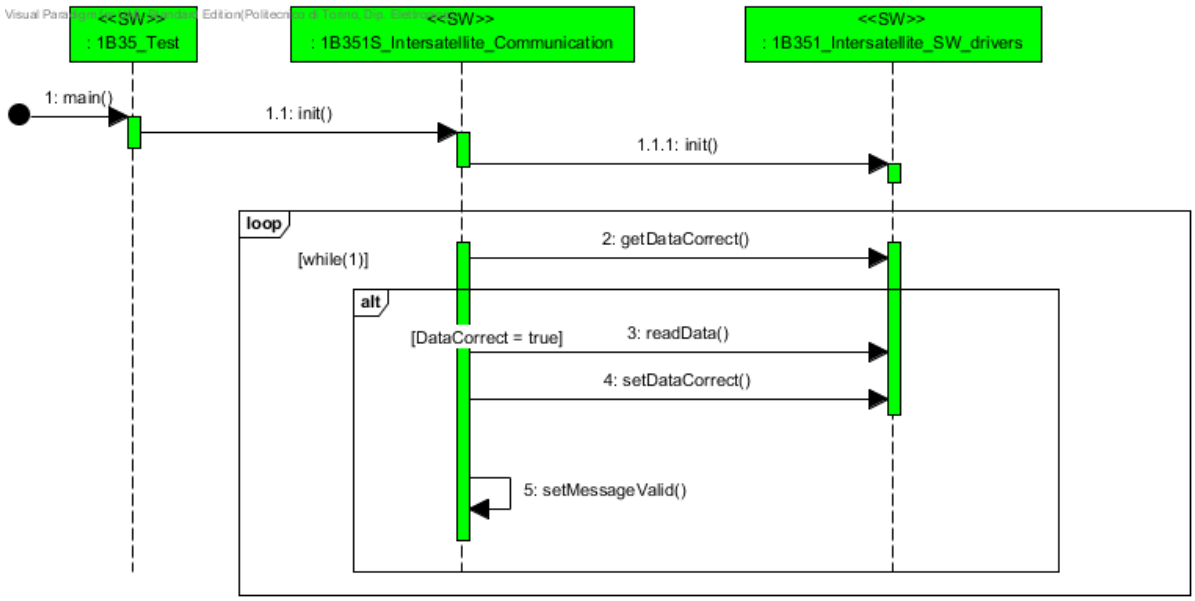


Figure 5.9 - Read Data sequence diagram

5.3.6 Measure Distance RF

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then the Controller can call measureDistanceRF function. 1B351S_Intersatellite_Communication uses Send Data use case and receives an ack. From the message received it can calculate distance with processDistanceRF function and it calls set_distValid function.

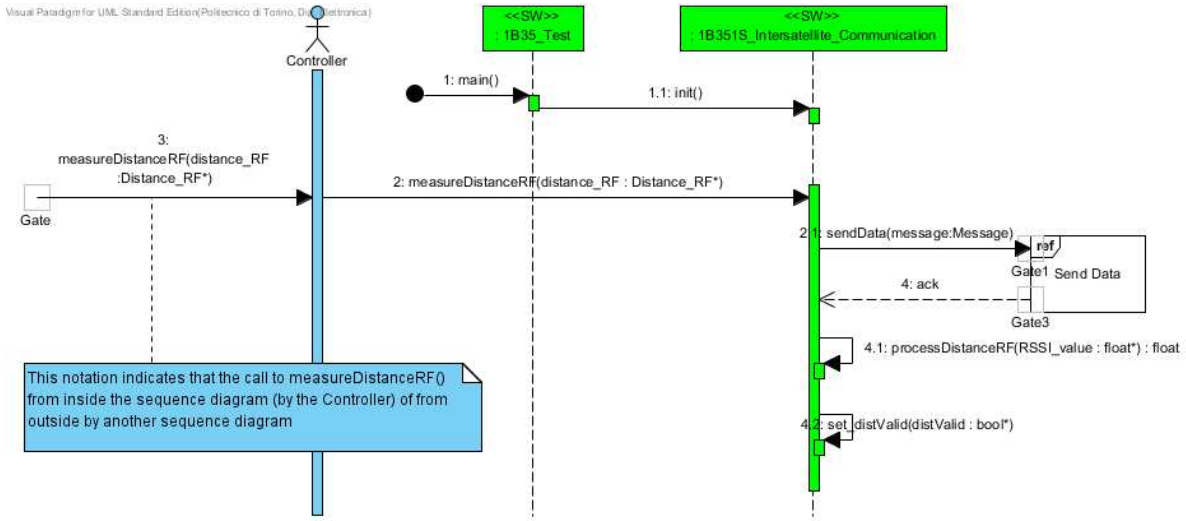


Figure 5.10 - Measure Distance RF sequence diagram

5.3.7 Get Distance RF

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then theController is able to ask the distance through getDistanceRF.

At this point 1B351S_Intersatellite_Communication calls check_distValid function to check the value of distValid. If high (1) returns Distance_RF to the Controller.

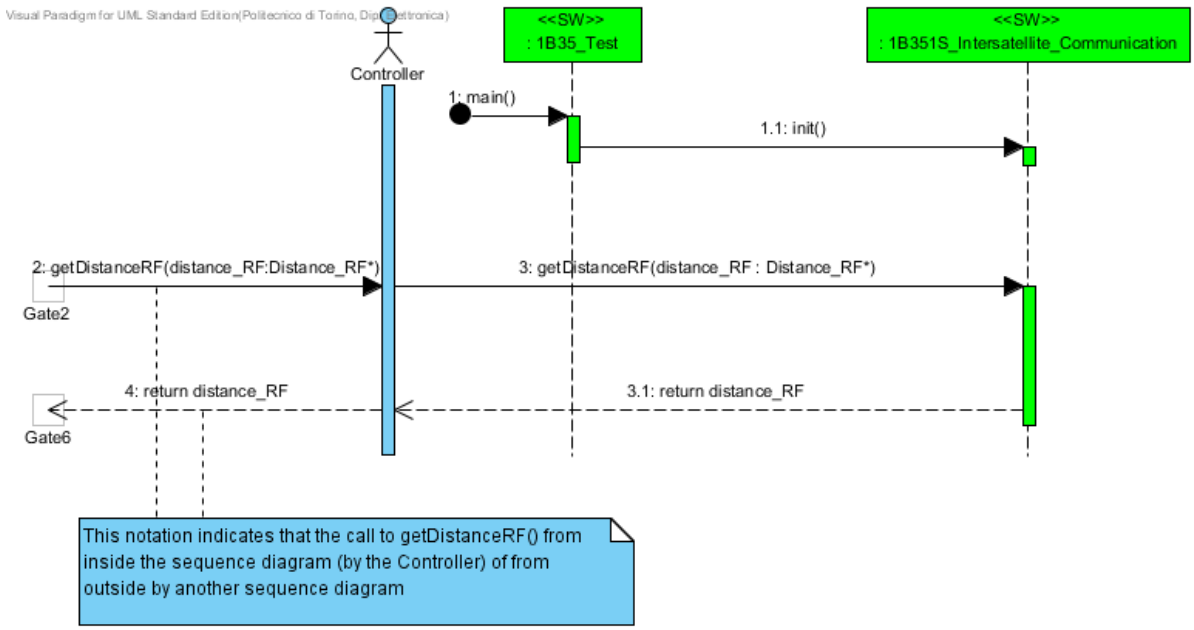


Figure 5.11 - Get Distance RF sequence diagram

5.3.8 Estimate Distance RF

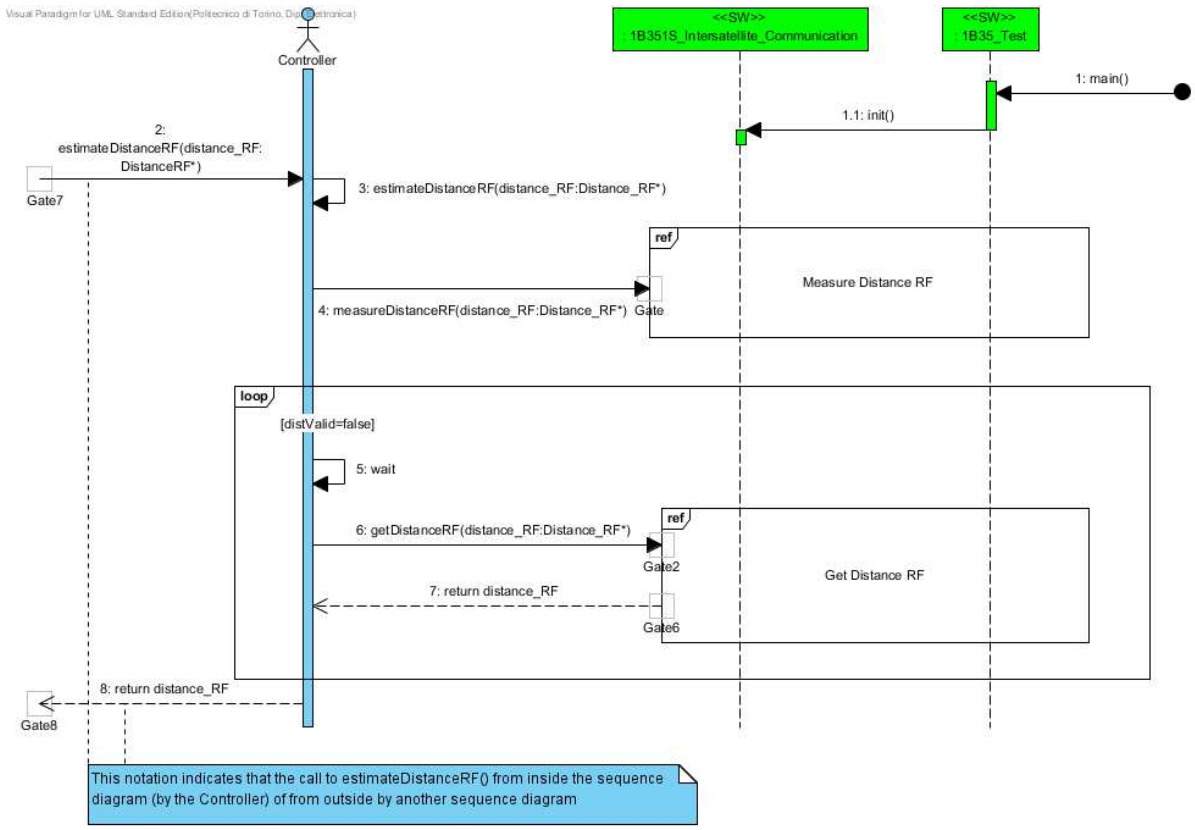


Figure 5.12 - Estimate Distance RF sequence diagram

5.3.9 Set Initial Frequency

The Configurator launches set_initial_freq function .

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then the 1B351S_Intersatellite_Communication calls change_frequency causing the execution of SMPL_Ioctl from simpliciTI_API that updates the registers in CC2510.

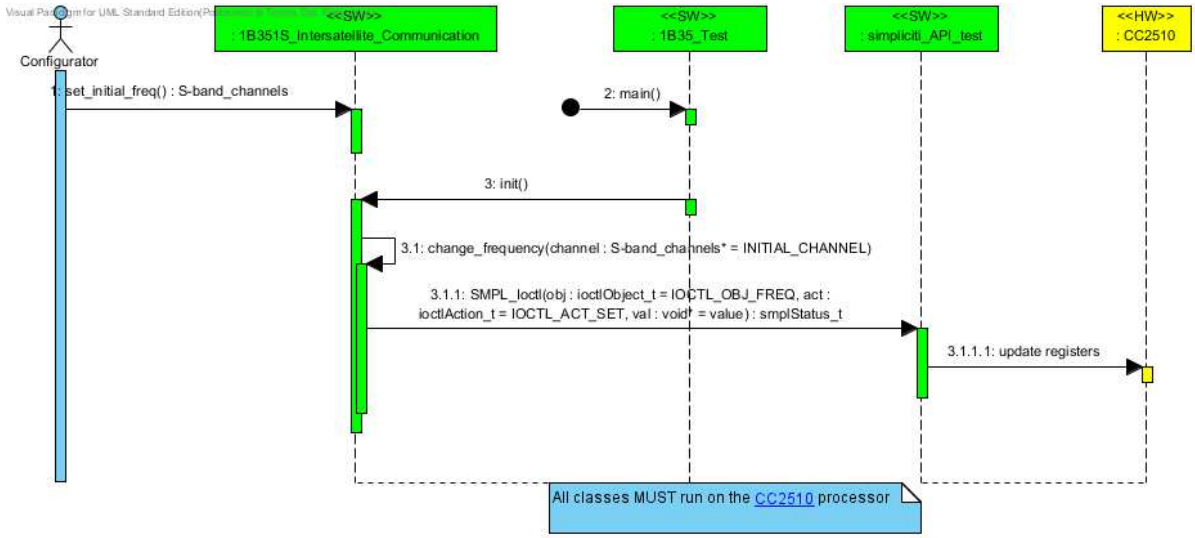


Figure 5.13 - Set initial Frequency sequence diagram

5.3.10 Change Frequency

The 1B35_Test initializes the 1B351S_Intersatellite_Communication calling init function, then the Controller is able to call change_frequency function which causes the call of SMPL_Ioctl from simpliciTI_API which updates registers on the Bk1B31B1W_OBRF_2_4GHz.

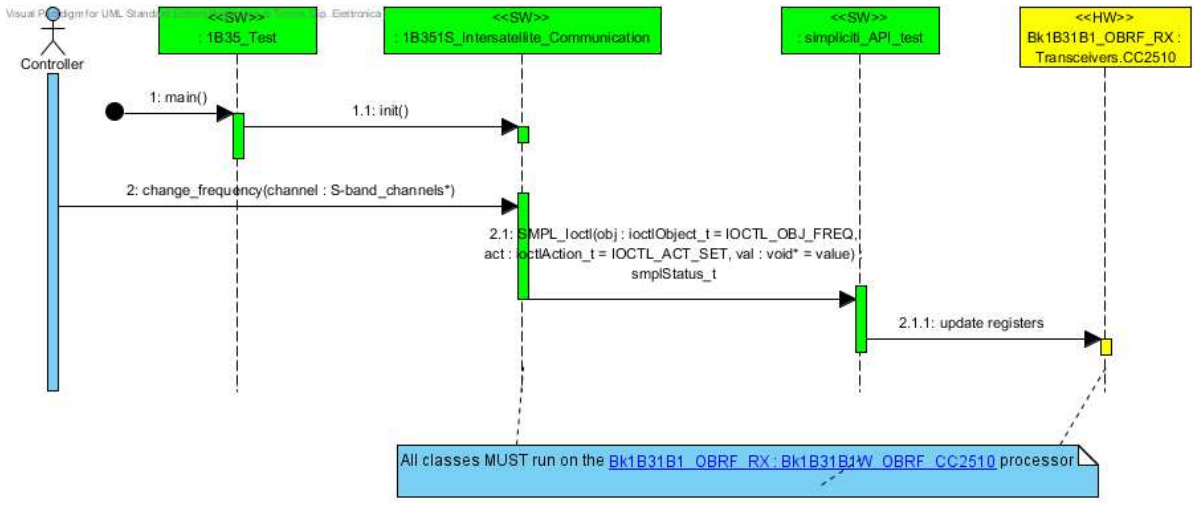


Figure 5.14 - Change Frequency sequence diagram

Conclusion

This thesis has led to the realization of 1B35_InterSatellite_Communication module for managing the communication between two satellites placed in line of sight.

In particular we realized the project description (fully documented in UML), the software routines, the schematics up to the PCB layout.

Due to the complexity of the subject, the FemtoSat project was not completed, but a general setting of the problem is presented. Also in this case the design is carried on by means of UML and different studies have been done to understand particular issues related to the subject:

-Solenoid Choice

-Source power Analysis

-FemtoSat dynamic analysis.

Concerning FemtSat project, this thesis represent a very good basis for future implementations.

Appendix I

Solenoid Choice - Matlab Files

I.1 Function *magnetic_field_1solenoid*

```
%having fixed solenoid parameters, the function evaluate the magnetic field
%generated by one solenoid at distance x.

clc
clear all
close all

%***physical constant***
ro=1.7241e-8;                %[Ohm*m]    copper resistivity
u0=4*pi*10^(-7);            %[T*m/A]  relative magnetic permeability

%***solenoid parameter***
Radius=0.055;                %[m]  coil radius
N=300;                       %number of coils

L_wire=2*pi*Radius*N;        %[m]  solenoid wire length
d_wire=0.15e-3;            %[m]  solenoid wire diameter
S_wire=(d_wire/2)^2*pi;          %[m^2]  solenoid wire cross section
R_sol=ro*L_wire/S_wire;      %[Ohm]  solenoid resistance
L_sol=d_wire*N              %[m]  length of the solenoid

a=L_sol/2;                   %[m]
b=2.5e-3;                     %[m]
x0=2*a-b;                    %[m] initial offset considering the thicketness of
the solenoid and magnetic sensor

%***power, voltage, current on the solenoid***
```

```
P=2;                      %[W] max power coming from power generator
Rdson=1;
Rs=0.2;
Req=R_sol+Rdson+Rs;
V=sqrt(Req*P)            %[V]  voltage supplying the solenoid
I=V/Req            %[A]  current flowing inside one solenoid



d=0.01                    %[m]  distance from the solenoid coil

B_x_gauss=B(d+x0,I,N,L_sol,Radius)

d=0.2                   %[m]  distance from the solenoid coil

B_x_gauss=B(d+x0,I,N,L_sol,Radius)


d=linspace(0,1,10000);
for (i=1:10000)
    y(i)=B(d(i)+x0,I,N,L_sol,Radius);
end
createfigure(d*100,y);
```

I.2 Function *magnetic_field_2solenoid*

```
%having fixed solenoid parameters, the function evaluate the magnetic field
%generated by 2 solenoid in serires at distance x.


clc
clear all
close all

%***physical constant***
ro=1.7241e-8;               %[Ohm*m]    copper resistivity
u0=4*pi*10^(-7);           %[T*m/A]  relative magnetic permeability

%***solenoid parameter***
Radius=0.055;              %[m]  coil radius
N=150;                     %number of coils

L_wire=2*pi*Radius*N;      %[m]  solenoid wire length
d_wire=0.15e-3;           %[m]  solenoid wire diameter
S_wire=(d_wire/2)^2*pi;         %[m^2]  solenoid wire cross section
R_sol=ro*L_wire/S_wire;    %[Ohm]  solenoid resistance
L_sol=d_wire*N             %[m]  length of the solenoid

a=L_sol;                  %[m]
b=2.5e-3;                    %[m]
x0=2*a-b;                  %[m] initial offset considering the thicketness of
the solenoid and magnetic sensor

%***power, voltage, current on the solenoid***
P=2;                      %[W] max power coming from power generator

Rdson=1;
Rs=0.2;
```

```
Req=R_sol*2+Rdson+Rs;
V=sqrt(Req*P)            %[V]  voltage supplying the solenoid
I=V/Req           %[A]  current flowing inside one solenoid



d=0.01                    %[m]  distance from the solenoid coil

B_x_gauss=B(d+x0,I,N*2,L_sol*2,Radius)

d=0.2                   %[m]  distance from the solenoid coil

B_x_gauss=B(d+x0,I,N*2,L_sol*2,Radius)


d=linspace(0,1,1000);
for (i=1:1000)
    y(i)=B(d(i)+x0,I,N*2,L_sol*2,Radius);
end
createfigure(d*100,y);
```

I.3 Function *solenoid_parallel*

```
function [R_sol,Req,V,I,B_dmin,B_dmax] =
solenoid_parallel_Req(Radius,N_coil,P,N_sol)
%UNTITLED Summary of this function goes here
%   Detailed explanation goes here
   %***physical constant***
    ro=1.7241e-8;               %[Ohm*m]    copper resistivity
    u0=4*pi*10^(-7);           %[T*m/A]  relative magnetic permeability

    %***solenoid parameter***
    L_wire=2*pi*Radius*N_coil;       %[m]  solenoid wire length
    d_wire=0.15e-3;           %[m]  solenoid wire diameter
    S_wire=(d_wire/2)^2*pi;         %[m^2]  solenoid wire cross section
    R_sol=ro*L_wire/S_wire;     %[Ohm]  solenoid resistance
    L_sol=d_wire*N_coil;             %[m]  length of the solenoid
    R_sol/N_sol;
    b=2.5e-3;                    %[m]
    x0=2*L_sol*(N_sol/2)-b;                   %[m] initial offset considering
the thicketness of the solenoid and magnetic sensor

    %***power, voltage, current on the solenoid***
    Rdson=1;
    Rs=0.2;
    Req=R_sol/N_sol+Rdson+Rs;
    V=sqrt(Req*P);            %[V]  voltage supplying the solenoid
    I=V/Req/N_sol;            %[A]  current flowing inside one solenoid


d=0.01;                    %[m]  minimum distance considered between two
solenoids

B_dmin=B(d+x0,I,N_coil*N_sol,L_sol*N_sol,Radius);

d=0.2;                    %[m]  minimum distance considered between two
solenoids
```

```
B_dmax=B(d+x0,I,N_coil*N_sol,L_sol*N_sol,Radius);

d=linspace(0,2,100000);
for (i=1:100000)
    y(i)=B(d(i)+x0,I,N_coil*N_sol,L_sol*N_sol,Radius);
end
d1=find(y<0.2,1)*(2/100000)*100;
d2=find(y<0.0005,1)*(2/100000)*100;

%createfigure(x*100,y);

end
```

I.4 Function *B*

```
function [B_x] = B(x,I,N,L_sol,Radius)
%the function return the value of magnetic field starting from physical
%parameter if the solenoid.

   %***physical constant***
   ro=1.7241e-8;                %[Ohm*m]    copper resistivity
   u0=4*pi*10^(-7);            %[T*m/A]  relative magnetic permeability

   B_x=(u0*N*I)/(2*L_sol)*((L_sol+2*x)/sqrt((L_sol+2*x)^2+4*Radius^2)+(L_sol-
2*x)/sqrt((L_sol-2*x)^2+4*Radius^2))*10^4;
end
```

I.5 Function *main*

```
clc
clear all
close all

Radius=0.055;        %[m] solendoid radius
P=2;              %[W] power coming from an ideal source

% N_solenoid=1       %number of solenoids
%
% N_coil=50
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
% N_coil=100
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
% N_coil=150
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
% N_coil=200
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
% N_coil=300
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)


N_solenoid=2       %number of solenoids

N_coil=50
[R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
N_coil=100
[R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
```

```
N_coil=150
[R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)

% N_solenoid=4       %number of solenoids
%
% N_coil=50
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
% N_coil=75
% [R_sol,Req,V,I,B_dmin,B_dmax]=solenoid_parallel(Radius,N_coil,P,N_solenoid)
```

Appendix II

FemtoSat Dynamics - Matlab Files

II.1 Function *main*

```
clc
clear all
close all

%two exaples of calculations%

time_calculation_sat(1,0.01)
time_calculation_sat(0.5,0.01)

time_calculation_sat(0.01,1)
time_calculation_sat(0.01,0.5)
```

II.2 Function *time_calculation_sat*

```
function [ T ] = time_calculation_sat( d_start,d_stop )

%it receives the starting distance and the stopping distance (in meter)
%and returns the time that the satellites need to approach/leave

   u0=4*pi*10^(-7);
   N=150;
   I=0.1731;
   H=0.023;
   R=0.035;
```

```
   m=0.1;

   x_start=d_start/2;
   x_stop=d_stop/2;

   x=linspace(x_start,x_stop,1000);
   v=zeros(1,1000);
   deltaX=abs(x_start-x_stop)/1000;

   for i=2:1000
      F(i)=pi*u0/4*(N*I/H)^2*R^4*(1/((2*x(i))^2)+1/(2*x(i)+2*H)^2-
2/(2*x(i)+H)^2);
      a(i)=F(i)/m;
      v(i)=sqrt(v(i-1)^2+2*a(i)*deltaX);
      t(i)=deltaX/(v(i-1)+0.5*(v(i)-v(i-1)));
   end

   T=sum(t);


end
```

Appendix III

1B35 Design - Software Routines

Class - SPI_Driver

Operations

Signature: init()

Code Body: #include 1B35_main.c

```
void Initialize_SPI_Driver()
{

/*SPI configuration*/

        #define SPIy                SPI1
        #define SPIy_GPIO           GPIOA
        #define SPIy_CLK            RCC_APB2Periph_SPI1
        #define SPIy_GPIO_CLK       RCC_APB2Periph_GPIOA
        #define SPIy_SCKPin         GPIO_Pin_5
        #define SPIy_MISOPin        GPIO_Pin_6
        #define SPIy_MOSIPin        GPIO_Pin_7

/* TODO: implement a system of calling one of the following functions basing on cycles
and passing parameters. */

send_message();
processDistanceRF();
return_distance();
read_message();
```

```
measure_distance();
get_distance();
get_ack();
estimate_distance_RF();
set_messageValid();
set_distValid();
check_distValid();

return 0;
}
```

Signature: send_message(mess : char*, destIDsourceIDlengthpreamble_length)

Code Body:

```
void send_message()
{

        if(message != NULL);
        {
                GPIO_Configuration();
                SPI_Configuration();
                while(message--)
                {
                /* Waiting for transmission to end. */
                while(SPI_I2S_GetFlagStatus(SPIy, SPI_I2S_FLAG_TXE) == RESET)
                {
                }

                /* Writes a byte (bit after bit) inside SPIy Transmit Data Register. */
                SPI_I2S_SendData(SPIy, TxBuffer2[TxCounter2++]);
                }

        RCC_APB2PeriphClockCmd(SPIy_GPIO_CLK | RCC_APB2Periph_AFIO, ENABLE);
        /* Enables SPIy Clock. */
        RCC_APB2PeriphClockCmd(SPIy_CLK, ENABLE);
        }

        else
        {
        }

return 0;
}
```

```
/*  Here is the function that configures GPIO. */
void GPIO_Configuration(void)
{

        /* Here is the configuration of SPI1 Pins Clock, MOSI and MISO as GPIO Ports. */
        GPIO_InitStructure.GPIO_Pin = SPIy_SCKPin | SPIy_MISOPin | SPIy_MOSIPin;
        GPIO_Init(SPIy_GPIO, &GPIO_InitStructure);
}

/*  Here is the function that configures SPI. */
void SPI_Configuration(void)
{
        /* Refers to the library stm32f10x_spi.h */
        SPI_InitTypeDef SPI_InitStructure;
        /* Refers to the library stm32f10x_spi.c */
        SPI_StructInit(&SPI_InitStructure);
        /* Refers to the library stm32f10x_spi.c */
        SPI_I2S_DeInit(SPIy);

        /* Configures all aspects of SPI defining each of its parameters.
         * All the meanings of those values are defined in stm32f10x_spi.h.
         */
        SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
        SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;
        SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
        SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
        SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
        SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
        SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_LSB;

        /* Configures SPIy as SPI with the declared features. */
        SPI_Init (SPIy, &SPI_InitStructure);

        /* Enables SPIy. */
        SPI_Cmd(SPIy, ENABLE);

}
```

Signature: processDistanceRF(distance : Distance_RF) : float

Code Body:

```
void processDistanceRF(S-band_channels f)
{
float _pi = 3.14;
```

```
float lambda;
float P_t;                  /* TBD by Configurator */
float P_r;                  /* TBD by Configurator */
float G_t;                  /* TBD by Configurator */
float G_r;                  /* TBD by Configurator */
int c = 300000;  /* m/s */
S-band_channels f;

lambda = c/f;

distance_RF.dist_RF = ((lambda/(4*_pi))*((P_t*G_r*G_t)/P_r));

}
```

Signature: return_distance() : Distance_RF

Code Body:

```
void return_distance()
{

        if(distance.distValid == 1);
        {
        return DistanceRF=distance;
        }

        else
        {
        }

return 0;
}
```

Signature: read_message(message : Message)

Code Body:

```
void read_message()
{

/* the function need to pass all parameters of the struct t_message from the received
message to the local struct*/

        message.preamble_length = Message.preamble_length;
        message.sourceID = Message.sourceID;
        message.destID = Message.destID;
        message.message_length = Message.message_length;
```

```
        message.message_body = Message.message_body;
        message.message_valid = Message.message_valid;
        message.power = Message.power;
        message.frequency = Message.frequency;

return 0;
}
```

Signature: measure_distance(distance : Distance_RF)

Code Body:

```
void measure_distance()
{

}
```

Signature: get_distance(ID : unsigned char &) : float

Code Body:

```
void get_distance()
{

        if(distance_RF != NULL);
        {
                GPIO_Configuration();
                SPI_Configuration();
                while(distance_RF--)
                {
                /* Waiting for transmission to end. */
                while(SPI_I2S_GetFlagStatus(SPIy, SPI_I2S_FLAG_RXE) == RESET)
                {
                }

                /* Writes a byte (bit after bit) inside SPIy Transmit Data Register. */
                SPI_I2S_GetData(SPIy, RxBuffer2[TxCounter2++]);
                }

        RCC_APB2PeriphClockCmd(SPIy_GPIO_CLK | RCC_APB2Periph_AFIO, ENABLE);
        /* Enables SPIy Clock. */
        RCC_APB2PeriphClockCmd(SPIy_CLK, ENABLE);
        }

        else
        {
        }
```

```
return 0;
}



/*  Here is the function that configures GPIO. */
void GPIO_Configuration(void)
{

        /* Here is the configuration of SPI1 Pins Clock, MOSI and MISO as GPIO Ports. */
        GPIO_InitStructure.GPIO_Pin = SPIy_SCKPin | SPIy_MISOPin | SPIy_MOSIPin;
        GPIO_Init(SPIy_GPIO, &GPIO_InitStructure);
}

/*  Here is the function that configures SPI. */
void SPI_Configuration(void)
{
        /* Refers to the library stm32f10x_spi.h */
        SPI_InitTypeDef SPI_InitStructure;
        /* Refers to the library stm32f10x_spi.c */
        SPI_StructInit(&SPI_InitStructure);
        /* Refers to the library stm32f10x_spi.c */
        SPI_I2S_DeInit(SPIy);

        /* Configures all aspects of SPI defining each of its parameters.
         * All the meanings of those values are defined in stm32f10x_spi.h.
         */
        SPI_InitStructure.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
        SPI_InitStructure.SPI_Mode = SPI_Mode_Slave;
        SPI_InitStructure.SPI_DataSize = SPI_DataSize_8b;
        SPI_InitStructure.SPI_CPOL = SPI_CPOL_High;
        SPI_InitStructure.SPI_CPHA = SPI_CPHA_2Edge;
        SPI_InitStructure.SPI_NSS = SPI_NSS_Soft;
        SPI_InitStructure.SPI_FirstBit = SPI_FirstBit_LSB;

        /* Configures SPIy as SPI with the declared features. */
        SPI_Init (SPIy, &SPI_InitStructure);

        /* Enables SPIy. */
        SPI_Cmd(SPIy, ENABLE);

}
```

Signature: estimate_distance_RF(distance : Distance_RF)

Code Body:

```
void estimate_distance_RF()
{

measure_distance(distance);

dist_RF processDistanceRF(distance);

return 0;
}
```

Signature: set_messageValid(messageValid : bool)

Code Body:

```
void set_messageValid()
{

        if(message.message_body != 0)
        {
        return message.message_body;
        }

        else
        {
        }
}
```

Signature: get_messageValid() : bool

Signature: set_distValid(distValid : bool) : bool

Code Body:

```
void set_distValid()
{

        distance.distValid = 1;

return 0;
}
```

Signature: check_distValid(distValid : bool) : bool

Code Body:

```
void check_distValid()
{
	if(distance.distValid == 1)
	{
	return distance;
	}

	else
	{
	}
}
```

Class - 1B35_Test

Operations

Signature: main()

Code Body: /***************1B35_main.c*******************/

```
#include 1B351_Intersat_SW_drivers.h
#include 1B351S_Intersat_communication.h
#include SimpliciTI_API.h
#include SPI_Driver.h

int main (void)
{

	typedef unsigned char UCHAR, *PUCHAR;

	typedef struct Distance_RF
	{
	uchar ID_MASTER;
	uchar ID_SLAVE;
	float dist_RF;
	bool distValid;
	float time_MASTER;
	float time_SLAVE;
	float power;
	}t_Distance_RF;

	typedef struct Message
	{
	ushort preamble_length;
```

```
        uchar sourceID;
        uchar destID;
        ushort message_length;
        uchar message_body;
        bool message_valid;
        float power;
        S-band_channels frequency;
        }t_Message;

        /* ID_List */

        uchar enum {ID0 = A, ID1 = B, ID2 = C, ID3 = D, ID4 = E, ID5 = F}

        /*S_Band_channels*/

        float enum {CHANNEL_0 = 2.4257, CHANNEL_1 = 2.4508,
                CHANNEL_2 = 2.4755, CHANNEL_3 = 2.4807}

        Initialize_1B351_Intersat_SW_drivers();

        Initialize_1B351S_Intersat_communication();

        Initialize_SPI_Driver();

        while(1)
        {
        }

return 0;
}
```

Class - simpliciTI_API

Operations

Signature: init(myAddress : ulong) : t_SimpliciTI_error

Code Body: BSP_Init();
//SMPL_Ioctl(IOCTL_OBJ_ADDR, IOCTL_ACT_SET, (addr_t*) &linkToAddr);
//SMPL_Init(sRxCallback);

Signature: send(dest : ulong, data : byte, length : byte) : t_SimpliciTI_error

Signature: status() : t_SimpliciTI_status

Signature: isDataAvailable(source : ulong) : bool

Signature: getData(data : byte, length : byte) : bool

Signature: set_address(myAddr : ulong)

Signature: SMPL_SendOpt(lid : linkID_t, msg : uint, len : uint, options : txOpt_t) : smplStatus_t

Signature: SMPL_Ioctl(obj : ioctlObject_t, act : ioctlAction_t, val : void) : smplStatus_t

Class - 1B351_Intersatellite_SW_drivers

Operations

Signature: init()

Code Body: #include 1B35_main.c

```
void Initialize_1B351_Intersat_SW_drivers()
{

S-band_channels target_freq = CHANNEL_0;

/* TODO: implement a system of calling one of the following functions basing on cycles
and passing parameters. */

sendData();
readData();
getDataCorrect();
setDataCorrect();
setFrequency();
setPower();
receive_data();
send_ack();
verify_data();

return 0;
}
```

Signature: sendData(mess : Message)

Code Body:

```
void sendData()
{

}
```

Signature: readData(mess : Message) : unsigned char

Code Body:

```
void readData()
{

        message.preamble_length = Message.preamble_length;
        message.sourceID = Message.sourceID;
        message.destID = Message.destID;
        message.message_length = Message.message_length;
        message.message_body = Message.message_body;
        message.message_valid = Message.message_valid;
        message.power = Message.power;
        message.frequency = Message.frequency;

return 0;
}
```

Signature: getDataCorrect() : bool

Code Body:

```
void getDataCorrect()
{

        if(message.message_body != 0)
        {
        dataCorrect = 1;
        }

return 0;
}
```

Signature: setDataCorrect(dataCorrect : bool)

Code Body:

```
void setDataCorrect()
```

```
{
        if(message.message_body != 0)
        {
        dataCorrect = 1;
        }

return 0;
}
```

Signature: setFrequency(target_channel : int)

Code Body:

```
void setFrequency(target_freq)
{

        if(frequency != target_freq)
        {
        message.frequency = target_freq;
        }

return 0;
}
```

Signature: setPower(power : float)

Code Body:

```
void setPower()
{
int i;

        if(message.power != target_power)
        {
                if(message.power < target_power)
                {
                message.power = message.power + 0.5 ;
                i++;
                }

                elseif(message.power > target_power)
                {
                message.power = message.power - 0.4;
                i++;
                }

                if(i >= 10)
```

```
                {
                }

                else
                {
                return 0;
                }
        }

        else
        {
        {

return 0;
}
```

Signature: receive_data(data : Message)

Code Body:

```
void receive_data()
{

        if(dataCorrect == 1)
        {
        message.preamble_length = Message.preamble_length;
        message.sourceID = Message.sourceID;
        message.destID = Message.destID;
        message.message_length = Message.message_length;
        message.message_body = Message.message_body;
        message.message_valid = Message.message_valid;
        message.power = Message.power;
        message.frequency = Message.frequency;
        }

        else
        {
        }

return 0;
}
```

Signature: send_ack(ack : Message) : Message

Code Body:

```
bool send_ack()
```

```
{
bool ack = 1;

message.message_body = ack;
sendMessage(message);

return 0;
}
```

Signature: verify_data(data : Message) : bool

Code Body:

```
void verify_data()
{

return message.message_valid;
}
```

Class - 1B351S_Intersatellite_Communication

Operations

Signature: init()

Code Body: #include 1B35_main.c

```
void Initialize_1B351S_Intersat_communication()
{

t_Commands commands = NULL;
uchar lastID = Z;
float power_TX = 0;
t_Distance_RF distance_RF = NULL;
bool done = 0;
Message message = NULL;

return 0;
}
```

Signature: sendMessage(message : Message)

Code Body: for(int i=0;i<MESSAGE_LENGTH;i++){

```
  message[i]=mess[i];
}
communication.sendData(message);
```

Signature: getMessage(message : Message) : void

Code Body:

```
void getMessage()
{

return message;
}
```

Signature: getDistanceRF(distance_RF : Distance_RF)

Code Body:

```
void getDistanceRF()
{

return distance_RF;
}
```

Signature: processDistanceRF(RSSI_value : float) : float

Code Body:

```
void processDistanceRF(S-band_channels f)
{
float _pi = 3.14;
float lambda;
float P_t;                  /* TBD by Configurator */
float P_r;                  /* TBD by Configurator */
float G_t;                  /* TBD by Configurator */
float G_r;                  /* TBD by Configurator */
int c = 300000;  /* m/s */
S-band_channels f;

lambda = c/f;

distance_RF.dist_RF = ((lambda/(4*_pi))*((P_t*G_r*G_t)/P_r));

}
```

Signature: measureDistanceRF(distance_RF : Distance_RF)

Code Body: message[0]=commands.DUMMY; //non sono sicuro che gli enumeration funzioni così

```
for (int i=1;i<MESSAGE_LENGTH;i++){
  message[i]=0;
}
sendData(message);

//dopo aver ricevuto l'ack
```

dist=processDistanceRF();

Signature: estimateDistanceRF(distance_RF : Distance_RF)

Code Body:

```
void estimateDistanceRF()
{

}
```

Signature: getMessageValid() : bool

Code Body:

```
void getMessageValid()
{

return message.message_valid;
}
```

Signature: setMessageValid(messageValid : bool)

Code Body:

```
void setMessageValid()
{

        if(message.message_body != NULL)
        {
        message.message_valid = 1;
        }

        else
        {
        }

return 0;
}
```

Signature: read_message(message : Message)

Code Body:

```
void read_message()
{
```

```
        message.preamble_length = Message.preamble_length;
        message.sourceID = Message.sourceID;
        message.destID = Message.destID;
        message.message_length = Message.message_length;
        message.message_body = Message.message_body;
        message.message_valid = Message.message_valid;
        message.power = Message.power;
        message.frequency = Message.frequency;

return 0;
}
```

Signature: send_ack(ack : Message) : Message

Code Body:

```
bool send_ack()
{
bool ack = 1;

message.message_body = ack;
sendMessage(message);

return 0;
}
```

Signature: check_distValid(distValid : bool)

Code Body:

```
void check_distValid()
{

        if(distance_RF.distValid == 1)
        {
        }

        else
        {
        }

return 0;
}
```

Signature: set_distValid(distValid : bool)

Code Body:

```
void set_distValid()
{

        if(distance_RF.dist_RF != 0)
        {
        distance_RF.distValid = 1;
        }

return 0;
}
```

Signature: set_initial_freq() : S-band_channels

Code Body:

```
void set_initial_freq()
{

target_freq = CHANNEL_0;

return 0;
}
```

Signature: receive_data(data : Message)

Code Body:

```
void receive_data()
{

        if(dataCorrect == 1)
        {
        message.preamble_length = Message.preamble_length;
        message.sourceID = Message.sourceID;
        message.destID = Message.destID;
        message.message_length = Message.message_length;
        message.message_body = Message.message_body;
        message.message_valid = Message.message_valid;
        message.power = Message.power;
        message.frequency = Message.frequency;
        }

        else
        {
        }
```

```
return 0;
}
```

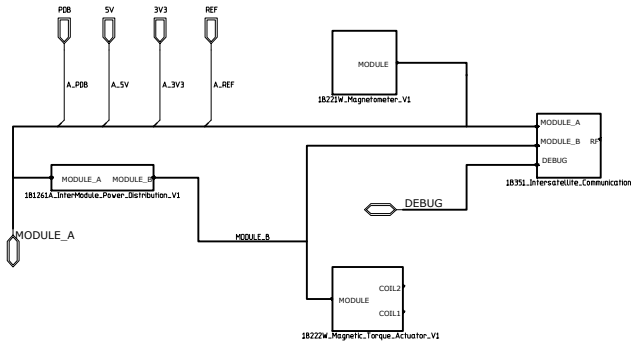
Signature: change_frequency(channel : S-band_channels)

Code Body:

```
void change_frequency()
{
int i;
S-band_channels x;

srand(time(NULL));

i = rand() % 4;

x = CHANNEL_i;

target_freq = x;

return 0;
}
```

Appendix IV

FemtoSat Design - Hardware schemes



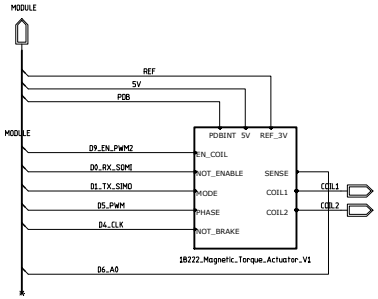
DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED

CONTRACT NO.		Schematic: FemtoSat Root: FemtoSat
APPROVALS	DATE	Project name: FemtoSat
DRAWN	17-Jul-13 6:07:32 PM	Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Lib.lmc
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

HxBND; Vx4LA



DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED

CONTRACT NO.	
APPROVALS	DATE
DRAWN	
CHECKED	
ISSUED	

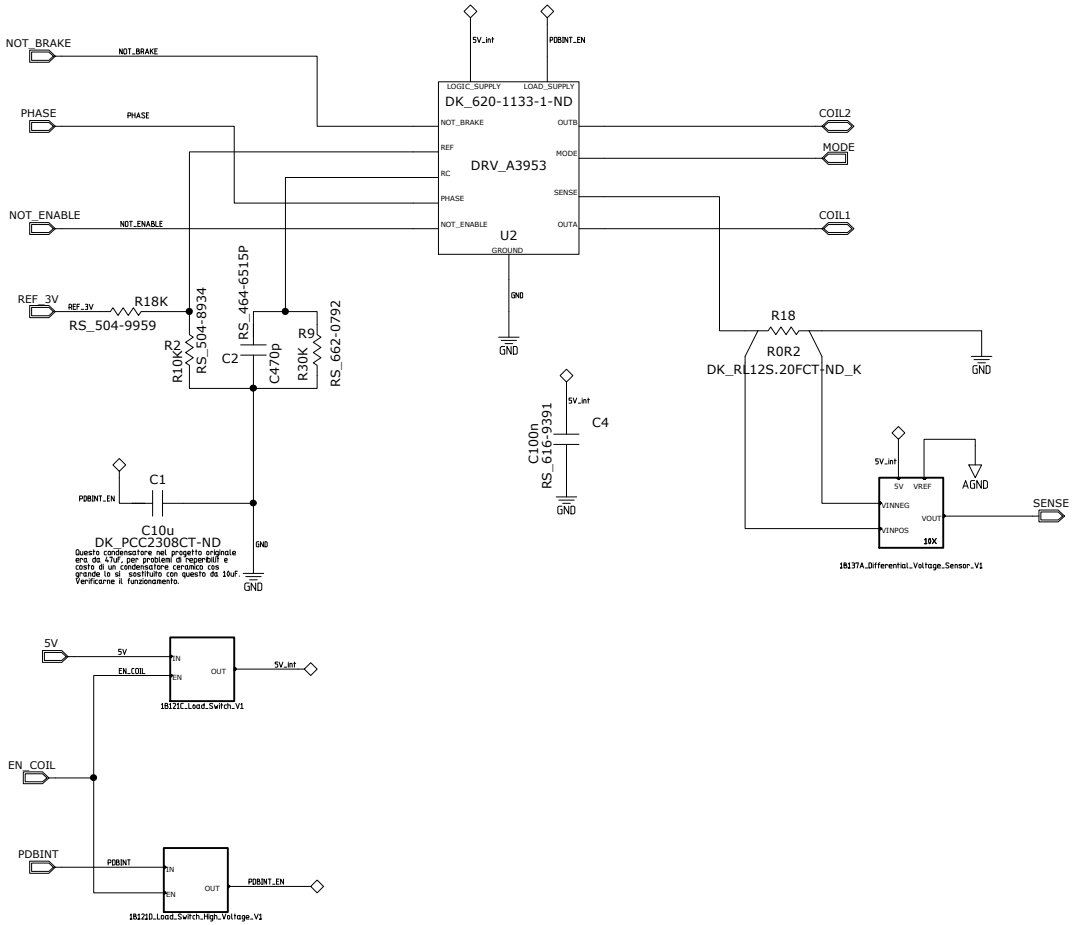
SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

H=BND; V=4LA

DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED



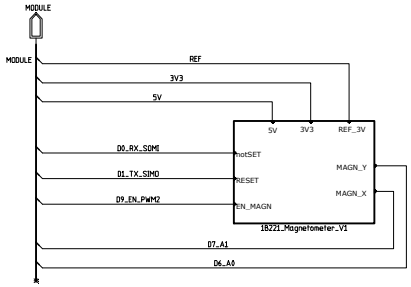
CONTRACT NO.		Schematic: 1B222_Magnetic_Torque_Actuator_V1 Root: FemtoSat
APPROVALS	DATE	
DRAWN	17-Jul-13 6:15:00 PM	Project name: FemtoSat Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Liblmc
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

H=BND; V=4LA

DWG. NO.	SH.	REV

REVISIONS

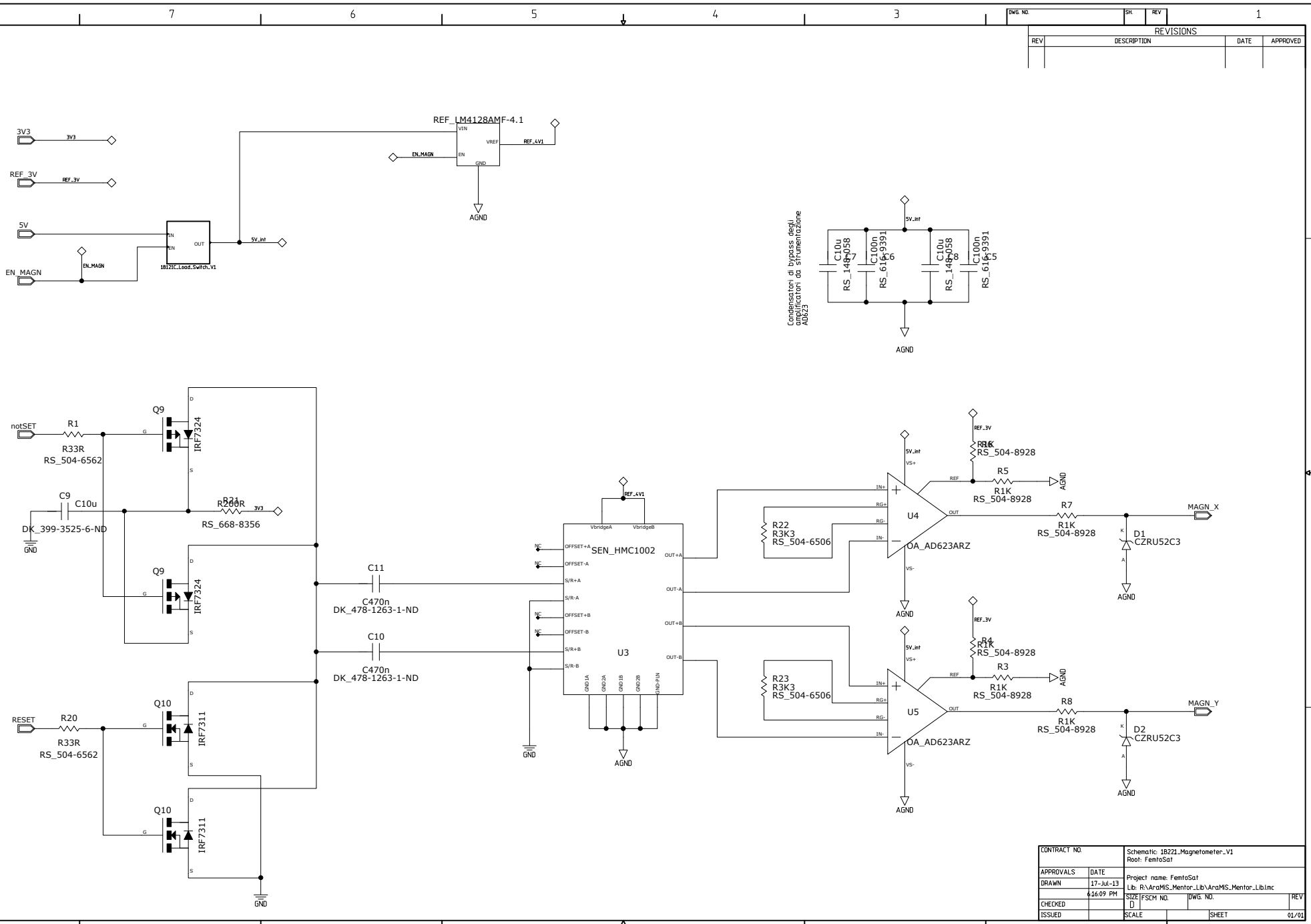
REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		
APPROVALS	DATE	
DRAWN		
CHECKED		
ISSUED		

SIZE	FSCM NO.	DWG. NO.	REV
D			
SCALE		SHEET	

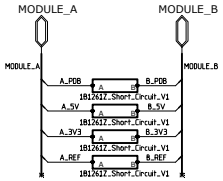
H=BND; V=4LA


DWG. NO.
SH.
REV
REVISIONS
REV
DESCRIPTION
DATE
APPROVED
3V3
REF_3V
5V
EN_MAGN
1B121C_Load_Switch_V1
5V_int
REF_LM4128AMF-4.1
REF_4V1
AGND
Condensatori di bypass degli amplificatori da strumentazione AD623
C10u
RS_146-7058
C100n
RS_616-9391
notSET
RESET
R1
R20
R33R
RS_504-6562
Q9
Q10
IRF7324
IRF7311
C9
C10u
DK_399-3525-6-ND
R21
R220R
RS_668-8356
C11
C10
C470n
DK_478-1263-1-ND
SEN_HMC1002
U3
U4
U5
OA_AD623ARZ
R22
R23
R3K3
RS_504-6506
R16
R4
R1K
RS_504-8928
R5
R3
R7
R8
MAGN_X
MAGN_Y
D1
D2
CZRU52C3
CONTRACT NO.
APPROVALS
DRAWN
17-Jul-13
6:16:09 PM
CHECKED
ISSUED
Schematic: 1B221_Magnetometer_V1
Root: FemtoSat
Project name: FemtoSat
Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Lib.lmc
SIZE
D
FSCM NO.
DWG. NO.
REV
SCALE
SHEET
01/01

DWG. NO.	SH.	REV

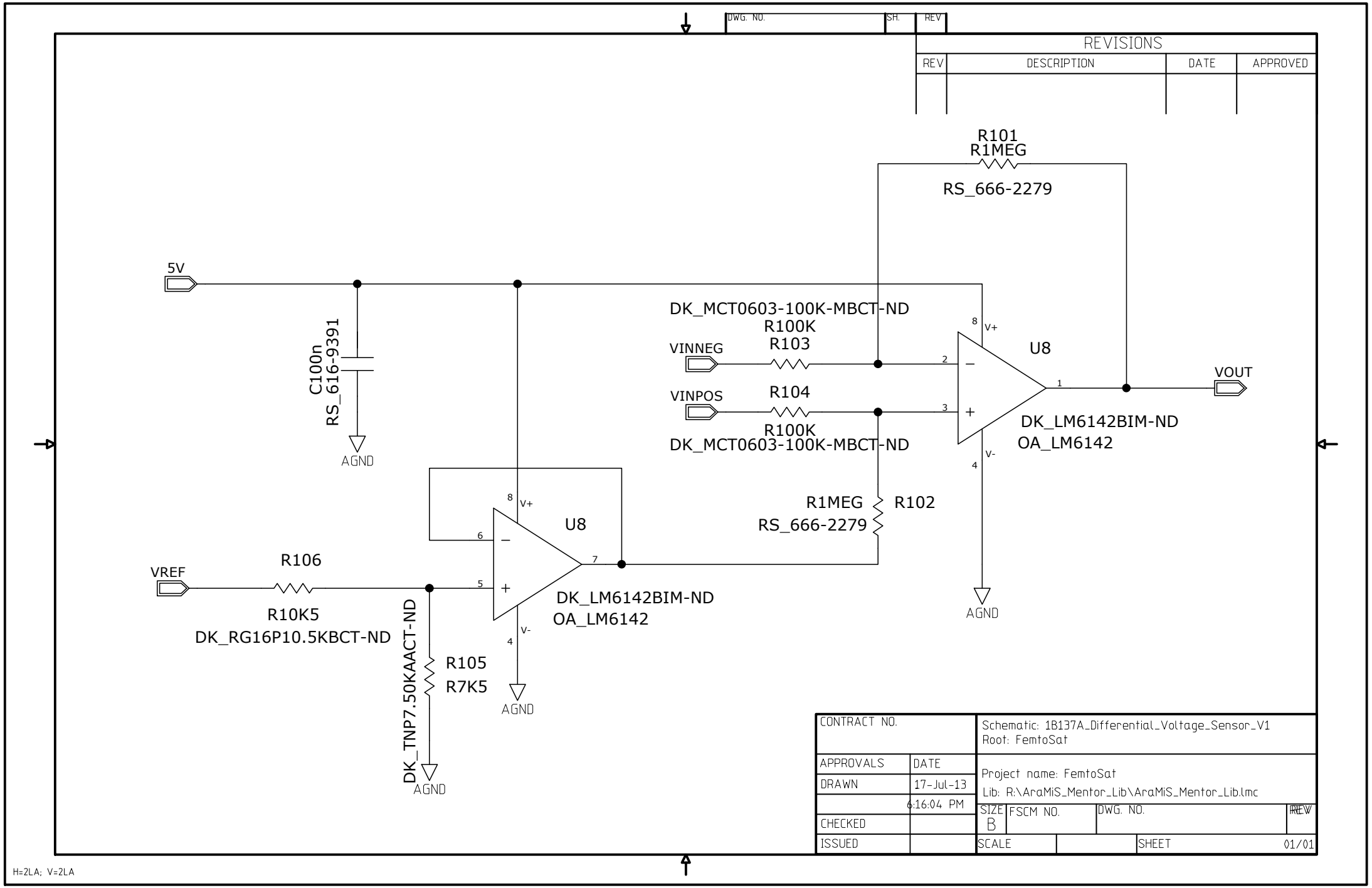
REVISIONS

REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		Schematic: 1B1261A_InterModule_Power_Distribution_V1 Root: FemtoSat
APPROVALS	DATE	Project name: FemtoSat Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Lib.lmc
DRAWN	27-May-13 12:16:47 PM	
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

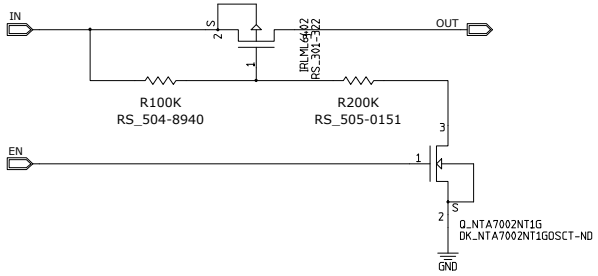
HtoBND; V=4LA


DWG. NO.
SH.
REV
REVISIONS
REV
DESCRIPTION
DATE
APPROVED
R101
R1MEG
RS_666-2279
5V
C100n
RS_616-9391
AGND
DK_MCT0603-100K-MBCT-ND
R100K
R103
VINNEG
VINPOS
R104
R100K
DK_MCT0603-100K-MBCT-ND
U8
V+
V-
DK_LM6142BIM-ND
OA_LM6142
VOUT
R1MEG
R102
RS_666-2279
AGND
VREF
R106
R10K5
DK_RG16P10.5KBCT-ND
R105
R7K5
DK_TNP7.50KAACT-ND
AGND
U8
V+
V-
DK_LM6142BIM-ND
OA_LM6142
AGND
CONTRACT NO.
Schematic: 1B137A_Differential_Voltage_Sensor_V1
Root: FemtoSat
APPROVALS
DATE
DRAWN
17-Jul-13
6:16:04 PM
CHECKED
ISSUED
Project name: FemtoSat
Lib: R:\AraMiS_Mentor_Lib\AraMiS_Mentor_Lib.lmc
SIZE
B
FSCM NO.
DWG. NO.
REV
SCALE
SHEET
01/01
H=2LA; V=2LA

DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED



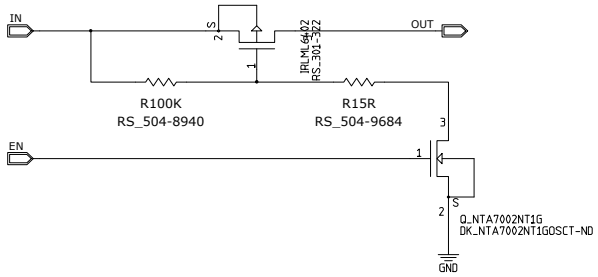
CONTRACT NO.		Schematic: 1B121D_Load_Switch_High_Voltage_V1 Root: FemtoSat
APPROVALS	DATE	
DRAWN	07-May-12	Project name: FemtoSat
	2:00:57 PM	Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Lib.lmc
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

HtmBND; V=4LA

DWG. NO.	SH.	REV

REVISIONS

REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		Schematic: 1B121C_Load_Switch_V1 Root: FemtoSat
APPROVALS	DATE	Project name: FemtoSat Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Liblmc
DRAWN	07-May-12 1:58:15 PM	
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

HtmBND; Vs4LA

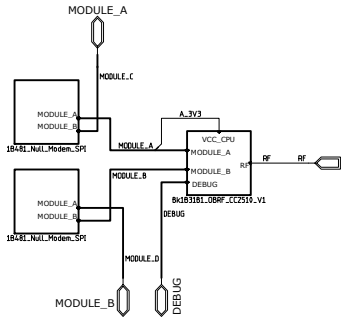
Appendix V

1B35 Design - Hardware schemes

DWG. NO.	SH.	REV

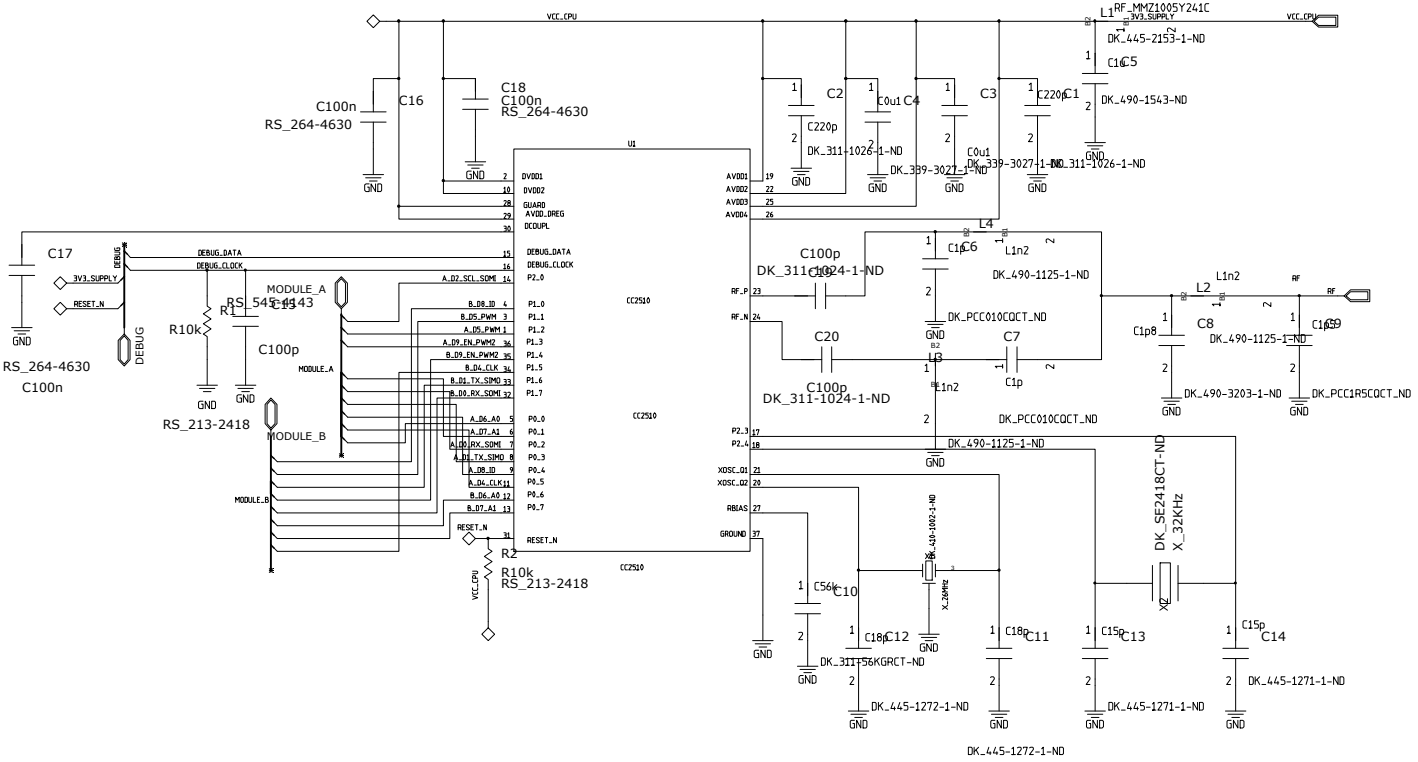
REVISIONS

REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		Schematic: 18351_Intersatellite_Communication Root: 8x183181_OBRF_CC2510_V1
APPROVALS	DATE	Project name: 18351_Intersatellite_Communication Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Lib.lmc
DRAWN	21-Jun-13 2:53:53 PM	
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

H=BND; V=4LA



REVISIONS			
REV	DESCRIPTION	DATE	APPROVED

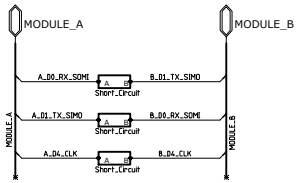
CONTRACT NO.		Schematic: Bk1B3181_OBAF_CC2510_V1 Root: Bk1B3181_OBAF_CC2510_V1
APPROVALS	DATE	Project name: 18351_Intersatellite_Communication
DRAWN	21-Jun-13 3:28:52 PM	Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Lib.lmc
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

H=BND; V=4LA

DWG. NO.	SH.	REV

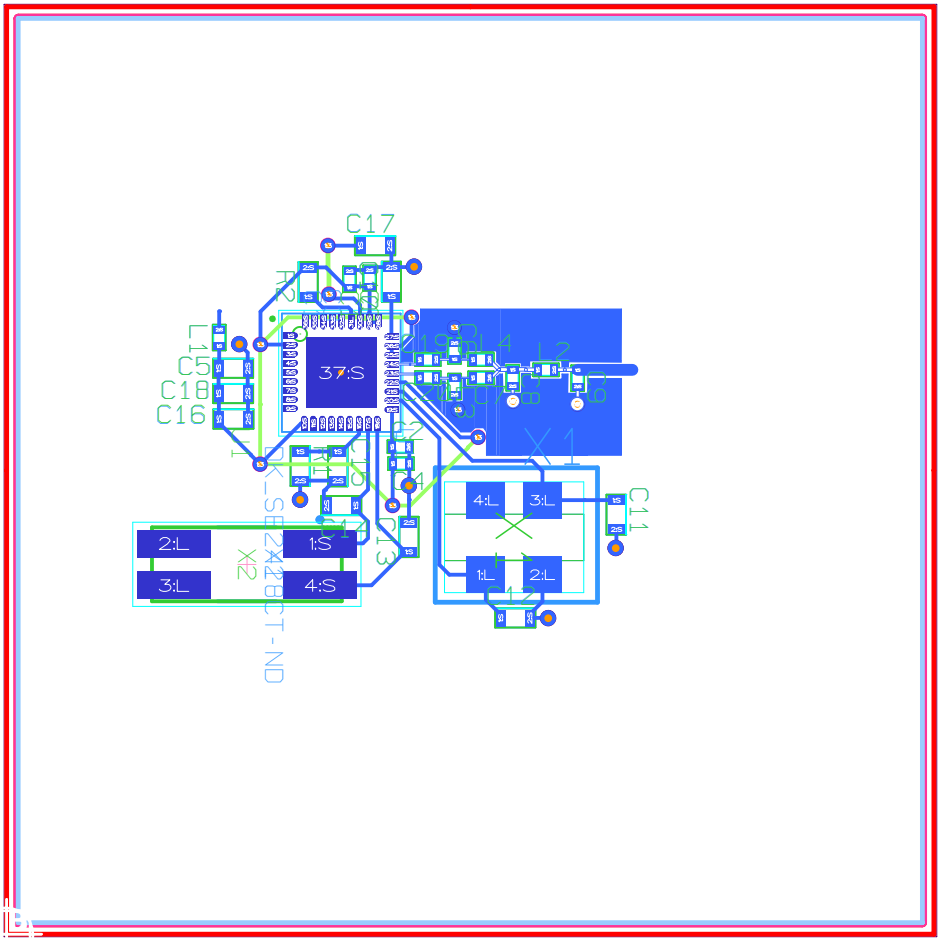
REVISIONS

REV	DESCRIPTION	DATE	APPROVED



CONTRACT NO.		Schematic: 18481_Null_Modem_SPI Root: BK1B3181_OBRF_CC2510_V1
APPROVALS	DATE	Project name: 18351_Intersatellite_Communication Lib: R:\AraMIS_Mentor_Lib\AraMIS_Mentor_Liblmc
DRAWN	20-Jun-13 10:19:26 AM	
CHECKED		SIZE D / FSCM NO. / DWG. NO. / REV
ISSUED		SCALE / SHEET 01/01

HtBND; Vt4LA


C17
C5
C18
C16
37:S
2:L
3:L
1:S
4:S
4:L
3:L
1:L
2:L
C11

Bibliography

1. MONTEFUSCO GIUSEPPE, *Design of a system of determination and communication of distance and mutual flying attitude between two satellites in close-by flight*, Aram-Dock, Master Thesis, Politecnico di Torino, 2013.

2. MARTIN FLOWER *UML Distilled, third edition*, Pearson Education, 2004.

3. VISUAL PARADIGM official web site http://www.visual-paradigm.com

4. R.T. MERRILL, M.W. MCELHINNY, PH.L. MCFADDEN. *The Magnetic Field of the Earth - Paleomagnetism, the Core and the Deep Mantle*. San Diego, California: Academic Press, 1996.

5. *Distribution of geomagnetic field intensity in 2009* http://www.ngdc.noaa.gov/geomag/WMM/WMM2010BetaSoft.shtml

6. DONATO RUSSO, MONTEFUSCO GIUSEPPE, *1B35_Intersatellite_Communication UML Project*, AraMis repository, Politecnico di Torino, 2013

7. DAVIDE MASERA, *1B22_Magnetic_Attitude_Subsystem UML project*, AraMis repository, Politecnico di Torino, 2012

8. RIZWAN MUGHAL, *1B45_Subsystem_Serial_Data UML project*, AraMis repository, Politecnico di Torino, 2013

9. HAIDER ALI, *1B31_On_Board_RF_Module UML project*, AraMis repository, Politecnico di Torino, 2013

10. JUANCARLOS DELOSRIOS, RIZWAN MUNGHAL *1B48_Module_Interface UML project*, AraMis repository, Politecnico di Torino, 2012

11. *Magnet* - https://en.wikipedia.org/wiki/Magnet

12. P.MAZZOLDI, M.NIGRO, C.VOCI, *Fisica, Vol.II Elettromagnetisco Onde*, Edises, 2002