



# Design and implementation of on-board processor and software of student nanosatellite OUFTI-1

Damien Teney

Thesis submitted in partial fulfilment of the requirements  
for the degree of the Master of Computer Science  
in the Faculty of Applied Sciences of the University of Liège

## Board of examiners:

Co-advisors:	Prof. Bernard Boigelot, ULg Prof. Jacques G. Verly, ULg
Examiners:	Mr. Luc Halbach, Spacebel Prof. Gaëtan Kerschen, ULg

Academic year 2008 - 2009  
University of Liège, Belgium  
Faculty of Applied Sciences  
Department of Electrical Engineering and Computer Science





---

# Abstract

OUFTI-1 is the first CubeSat from the University of Liège, as well as the first nanosatellite ever developed in Belgium. OUFTI-1 is being developed within the framework of a long-term, educative program, called Leodium (Liège in Latin), the goal of which is to develop a series of student satellites for scientific experiments. The payload of OUFTI-1 is a repeater for amateur radio, that uses the digital D-STAR protocol.

We developed the hardware and the software of the on-board computer of the OUFTI-1 CubeSat. The hardware uses an innovative solution, using two redundant processor boards, one available off-the-shelf, and one developed specifically for this mission. We developed a mechanism to manage the redundancy offered by the two processors; the goal of this hardware architecture is to extend the lifetime of the global system in the harsh space environment. Several prototypes of the custom board were built and successfully tested.

An original, modular, and robust architecture for the satellite on-board software was designed. It consists of six general-purpose modules that can handle real-time processing of housekeeping measurements, automatic storage of measurements, and execution of telecommands. It also allows one to pre-plan complex mission operations that are subsequently automatically executed. A first implementation of this software, not including the specific details of the OUFTI-1 mission, was realized and tested with success.



**Keywords:** *nano-satellite, CubeSat, OBDH, OBC, software, OUFTI-1*

---

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Subject and objectives . . . . .	1
1.2	Document outline . . . . .	2
<b>2</b>	<b>Context of project</b>	<b>3</b>
2.1	The Leodium initiative . . . . .	3
2.2	The CubeSat concept . . . . .	3
2.3	The OUFTI-1 project . . . . .	4
<b>3</b>	<b>Overview of mission</b>	<b>6</b>
3.1	Objectives of mission . . . . .	6
3.2	Operations of OUFTI-1 satellite system . . . . .	7
3.3	Constraints of mission . . . . .	9
3.3.1	Dimensional constraints . . . . .	9
3.3.2	Launcher constraints . . . . .	9
3.3.3	Orbit constraints . . . . .	11
3.3.4	Environment constraints . . . . .	12
<b>4</b>	<b>Architecture and subsystems of OUFTI-1 satellite</b>	<b>15</b>
4.1	Mechanical structure . . . . .	16
4.2	Electrical power supply . . . . .	17
4.3	Radio-communications . . . . .	18
4.4	Radio beacon . . . . .	18
4.5	Antennas . . . . .	19
4.6	Attitude control . . . . .	20
4.7	Thermal control . . . . .	21
4.8	Experimental electrical power supply . . . . .	21
4.9	On-board computer . . . . .	22
4.10	On-board measurements . . . . .	23
<b>5</b>	<b>Hardware: Requirements of on-board computer (OBC)</b>	<b>25</b>
5.1	Processing power . . . . .	25
5.2	Data flow paths . . . . .	26
5.3	Other concerns . . . . .	26

<b>6</b>	<b>Hardware: Architectures of CubeSat on-board computers</b>	<b>28</b>
6.1	OBC architectures of other CubeSats . . . . .	28
6.1.1	Centralized and distributed architectures . . . . .	28
6.1.2	Processors used by other CubeSats . . . . .	30
6.2	OBC architecture of OUFTI-1 . . . . .	32
6.2.1	Processor of OUFTI-1 . . . . .	32
6.2.2	Peripherals of the OBC of OUFTI-1 . . . . .	33
6.2.3	Data flow paths in OUFTI-1 . . . . .	34
6.2.4	Protection against radiations . . . . .	34
6.3	Overall electrical architecture of OUFTI-1 . . . . .	36
<b>7</b>	<b>Hardware: Design of the OBC electronic boards</b>	<b>37</b>
7.1	OBC1: Detailed analysis of the FM430 board . . . . .	37
7.2	OBC2: Design of the custom board . . . . .	39
7.3	Interboard communication bus . . . . .	41
<b>8</b>	<b>Hardware: Practical realization of the OBC electronic boards</b>	<b>43</b>
8.1	General strategy for prototyping and fabrication . . . . .	43
8.2	Strategy for prototyping and fabrication applied to the OBC . . . .	44
8.2.1	Breadboard models . . . . .	44
8.2.2	Engineering models . . . . .	44
8.2.3	Flight models . . . . .	47
<b>9</b>	<b>Software: Requirements and organization</b>	<b>50</b>
9.1	Identification of software fonctionnalités . . . . .	50
9.2	Organization of software into layers and modules . . . . .	51
9.2.1	Traditional organization of satellite on-board software . . . .	51
9.2.2	Organization of on-board software of OUFTI-1 . . . . .	52
<b>10</b>	<b>Software: Architecture</b>	<b>55</b>
10.1	Scheduling of software fonctionnalités . . . . .	55
10.1.1	Methods of scheduling . . . . .	55
10.1.2	Choice of operating system for the on-board software of OUFTI-1 . . . . .	56
10.1.3	Identification of tasks in the on-board software of OUFTI-1 .	57
10.2	Architecture of software modules . . . . .	57
10.2.1	Initialization of software . . . . .	58
10.2.2	Architecture of the monitor module . . . . .	59
10.2.3	Architecture of communication module . . . . .	61
10.2.4	Architecture of sequencer module . . . . .	62
10.2.5	Architecture of measurement module . . . . .	63
10.2.6	Architecture of log module . . . . .	64
10.2.7	Architecture of clock module . . . . .	65
10.2.8	Architecture of device drivers . . . . .	66

<b>11 Software: Practical implementation</b>	<b>67</b>
11.1 Details of software implementation . . . . .	67
11.2 Practical details of software realization . . . . .	70
11.3 Functional tests of software . . . . .	71
<b>12 Conclusions and future work</b>	<b>73</b>
12.1 Conclusions . . . . .	73
12.2 Ideas for future work . . . . .	74
<b>Bibliography</b>	<b>74</b>
 <b>Appendices</b>	 <b>77</b>
<b>A List of parameters to be measured aboard OUFTI-1</b>	<b>79</b>
<b>B Investigation of the use of two stacked FM430 boards</b>	<b>81</b>
<b>C Overall electrical architecture and data flow paths of OUFTI-1</b>	<b>83</b>
<b>D Allocation of the CubeSat bus</b>	<b>85</b>
<b>E OBC2 electrical schematics originally proposed by author</b>	<b>88</b>
<b>F OBC2 electrical schematics re-encoded by Deltatec</b>	<b>91</b>
<b>G OBC2 engineering model PCB originally proposed by author</b>	<b>95</b>
<b>H OBC2 engineering model PCB designed by Deltatec</b>	<b>97</b>
<b>I Handling of software functions by the application modules</b>	<b>102</b>
<b>J Static architecture of software</b>	<b>105</b>
<b>K Source code of software</b>	<b>107</b>
<b>L Test scenario and results</b>	<b>109</b>

---

# Chapter 1

## Introduction

### 1.1 Subject and objectives

This thesis discusses the development of the on-board processing capabilities of the OUFTI-1 nanosatellite. This satellite is the first developed at the University of Liège, as well as the very first nanosatellite developed in Belgium, under fully-Belgian management. It was developed by two successive waves of students, during the final year of their Master's degree, in 2007 – 2008 and 2008 – 2009.

The on-board computer of the satellite plays a central and thus critical role, and its development required a rigorous and well-structured organization. Several key objectives for this thesis were thus defined at the very beginning of the project.

1. Identify the needs of the OUFTI-1 mission in terms of processing power.
2. Study the solutions adopted by similar nanosatellites.
3. Develop, in collaboration with the teams working on the other subsystems, an appropriate global electrical architecture, identifying the role and the position of the processing unit(s).
4. Design and build the hardware of the on-board computer.
5. Design an overall software architecture and fundamental building blocks.
6. Write and optimize the detailed mission-specific software.

As we will see in the following, the five first objectives were successfully fulfilled. The sixth objective could not be completed, due to some uncertainties remaining in the specifications of the mission, as well as in the details of other subsystems of the satellite. Some abstraction effort was therefore needed, and led to the development of fully-functionnal hardware, and to the development of a solid and

modular software system, that can accommodate modifications of many of its details, if needed later.

## 1.2 Document outline

This thesis presents the mission of the satellite, and then explains the design details of the hardware and software parts of the on-board computer.

More precisely, Chapter 2 presents the context and the motivations of the project. Chapter 3 details the planned mission and its requirements. Chapter 4 gives an overview of the overall design of the satellite. Chapters 5 to 8 describe the development of the hardware part of the on-board computer, with the analysis of requirements, the development of a global architecture, the precise design of the electronic circuits, and finally the practical realization of the electronic boards. Chapters 9 to 11 describe the development of the software part, with an analysis of software requirements, explanations of the software architecture, and details on its practical implementation. Finally, Chapter 12 derives some conclusions and suggests some ideas for future work.

---

# Chapter 2

## Context of project

This chapter presents the context in which the project originated, and it presents the base ideas of the project in a conceptual manner. A more formal and exhaustive definition of the mission will be presented in chapter 3.

### 2.1 The Leodium initiative

Since about 2003, there has been a desire, at the University of Liège (ULg), to develop educational satellites. This initiative was given the name of *Leodium*, which means “Lancement en Orbite de Démonstrations Innovantes d’une Université Multidisciplinaire” (launch into orbit of innovative demonstrations of a multidisciplinary university), and which is also the Latin name of Liège. It started with a participation of the ULg to the European Space Agency’s Student Space Exploration and Technology Initiative (SSETI). In this context, the ULg took part in the European Student Earth Orbiter (ESEO) and European Student Moon Orbiter (ESMO) projects. ULg students worked on various subsystems of these satellites, more precisely on the deployment system of the solar panels of the former, and on a narrow angle camera for the latter.

### 2.2 The CubeSat concept

Not until September 2007 did the project of a completely Belgian satellite become a reality, with the idea of building a *CubeSat*, proposed by Luc Halbach, sales manager of Spacebel.

A CubeSat is a standard type of nanosatellite, for which very precise specifications exist. These specifications, developed in the late nineties at California Polytechnic State University (CalPoly) and at Stanford University, include precise requirements, the most restrictive one being probably the dimensions. Indeed, the satellite must fit in a cube 10 cm on a side, and weight no more than 1kg. Besides dimensions, the CubeSat standard also describes a specific launcher interface, facilitating the integration of CubeSats as secondary payloads on a wide range of commercial launchers.

Although some CubeSats have been built in industry, by large companies like Boeing<sup>1</sup>, most of the CubeSats launched to date were built by universities. The reasons is that they make an ideal educational project. Indeed, they constitute a very challenging multidisciplinary project, while requiring a budget and development time that are reasonable. This gives students the opportunity to work on all the phases of the design of a complete satellite system, from mission design to launch and operation. Besides educational aspects, such satellites also offer the possibility of carrying out innovative and interesting scientific experiments in space.

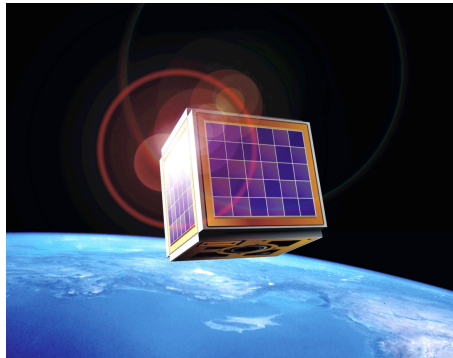


Figure 2.1: Typical CubeSat on orbit (source: AAU CubeSat website).

## 2.3 The OUFTI-1 project

The projected series of nanosatellites of the University of Liège were named OUFTI, where “oufti” is a typical interjection from the region of Liège, that could translate in English into “wow”. A possible interpretation for the OUFTI acronym is “Orbital Utility For Telecommunication Innovation”. The basic idea behind the first satellite of the series, OUFTI-1, is to put on orbit the cutting edge of amateur (digital) radio communications, namely the D-STAR system. In case of success, OUFTI-1

---

<sup>1</sup>The Boeing CubeSat TestBed 1 (CSTB1) mission was launched in April 2007.



would be the world's first satellite to feature the D-STAR radio-communication system.

The idea of using amateur radio bands on orbit is not new: it dates back to the early sixties, with the launch in 1961 of OSCAR-1, the very first satellite carrying amateur radio. As another example, more recently, the International Space Station (ISS) was also equipped with an amateur radio station; it is frequently used by astronauts to communicate with fellow amateur radio operators.

In our case, the advantage of using amateur radio for communicating with the satellite is significant. Indeed, the ham community offers a potentially worldwide tracking network for OUFTI-1. It could prove very interesting to receive data from the satellite while it is, e.g. over Japan or South America, when our tracking station in Belgium lies well beyond the footprint of the satellite. In exchange, the D-STAR capabilities of the satellite will be offered to ham-radio operators, who could then use it as a repeater.

More specifically, the D-STAR system seems appropriate for our project. It is a digital voice and data protocol, developed by the Japanese Amateur Radio League (JARL) in the late nineties. It mainly uses the 2m (VHF) and 70cm (UHF) amateur radio bands, and permits a synchronous transmission of voice and data. Technical considerations, beyond the scope of this document, and detailed in [Pisane08], justify its use in an application like our CubeSat. Furthermore, if the deployment of the D-STAR protocol on OUFTI-1 is successful, this system could be reused on future nanosatellites as the main communication means.

Two remarks are worth marking. First, D-STAR equipment for ground applications is readily available and affordable, which makes the development of our project easier. Second, a large pool of ham-radio operators are already equipped with such equipment throughout the world, which makes the use of D-STAR in our project very interesting for the above reasons.

Finally, let us not forget the novel experimental and educational aspects of the project, which make it challenging and very exciting.

---

# Chapter 3

## Overview of mission

This chapter presents the objectives of the mission, the details of operation of the satellite, and various constraints that are imposed to us. All these elements constitute a set of requirements, that will drive the subsequent design process. Indeed, the entire system will be devised for the sole purpose of satisfying these mission objectives, and for being able to operate under these constraints.

### 3.1 Objectives of mission

The primary objective of the project is educational. The goal is to give students hands-on experience and to give them the opportunity to work on a real space mission during all its phases. The consequence of this first objective is that as many tasks as possible of the project have to be performed by students.

To fulfill this educational objective, the project is to design, build, and operate a CubeSat in space. This primary objective leads to the definition of the primary success criterion, which is to establish a radio contact with the CubeSat once launched in space, and to receive its telemetry.

The secondary objectives of the mission are to use the satellite to test new technologies in space. As indicated before, the main payload of OUFTI-1 will be its telecommunication system, which will use the D-STAR protocol. Two secondary payloads will also be embarked: an experimental electrical power supply (later referred to as *EPS2*), and a new type of high efficiency solar cells, provided by the AzurSpace company.

## 3.2 Operations of OUFTI-1 satellite system

This section presents how the satellite is supposed to behave and how it can be used by on-ground operators.

### D-STAR radio-communications

The D-STAR capability of the satellite is used to offer a *repeater* functionality to any D-STAR equipped ham-radio operator, as long as he lies in the footprint of the satellite. He can then communicate, through our CubeSat, with another ham, or another ground-based repeater, which also has to lie within the footprint of the satellite.

The goal is to allow anyone to use ordinary D-STAR equipment with the satellite. As a consequence, the frequency shift induced by the Doppler effect, inherent to non-geostationary orbits, will have to be compensated on-board. A user in a specific zone will need to request compensation for that zone for a specific time on a dedicated website. The appropriate compensation will then be computed, on the ground, and uploaded to the satellite by our ground station.

The following typical scenario illustrates this procedure.

1. User A in zone A wants to contact, through the satellite, user B, lying in zone B, between times  $t_1$  and  $t_2$ . He therefore fills in a form with this information on the website of our ground station.
2. The computer of the ground station computes the appropriate Doppler shift corrections for the period  $[t_1, t_2]$ .
3. At time  $t_0$  ( $< t_1$ ), the satellite comes in sight of the ground station and radio contact can be established. The computer of the ground station may first request some telemetry to check the health of the satellite and, for example, assess the state of its batteries. Then, if all retrieved data is nominal (after an automatic or manual verification on the ground), the ground station uploads the Doppler shift corrections, together with the order of activating the D-STAR on-board repeater between times  $t_1$  and  $t_2$ .
4. At time  $t_1$ , the D-STAR repeater is activated by the satellite, and user A can then use it to communicate with user B.
5. Between times  $t_1$  and  $t_2$ , aboard the satellite, the frequency of the communication with user A (resp. user B) is regularly modified, according to the correction tables uploaded in point 2, so that the apparent communication frequency with the satellite in zone A (resp. zone B) is constant.
6. At time  $t_2$ , the satellite deactivates the D-STAR repeater.

## Telecommands

In addition to the D-STAR channel, the OUFTI-1 CubeSat is equipped with an independent “maintenance” communication system. This system employs the AX.25 protocol, which is used to transmit telecommands to the satellite, and to retrieve telemetry from it. It is important to mention that this “telecommand” channel is monitored continuously, even during D-STAR operations for example. This ensures that the satellite always stays under control from the ground, and that it can, for example, be shut down at any time. Note that this particular feature is a specific requirement from the ITU Radio Regulations, that we have to comply with (see [IARU06, ITU04]).

## Secondary payloads

The operation of the secondary payloads involves mainly taking regular measurements of various parameters (voltages, currents, and temperatures). These measurements are performed and stored automatically at predefined frequencies, and they are then available for download on the request of a telecommand. It is worth noting that, on the one hand, the experimental solar cells are critical to the operation of the satellite, and are thus used at all times. On the other hand, the second secondary payload, the experimental electrical power supply, is not required for the normal operation of the satellite, and is only activated for testing purposes. It can be switched on and off in predefined conditions (e.g. when crossing a predefined threshold of the battery voltage), or on specific requests sent by telecommands.

## Radio beacon

In order to maximize the chances of success of the primary objectives of the mission (launch a satellite and establish a radio contact with it), the following fail-safe strategy was adopted. In addition to the main precited communication means, an independent radio transmitter, called the beacon, is included on board. It will be active at all times, and its only task will be to take some measurements aboard the satellite, and send them on a rudimentary (e.g. CW modulated) radio channel. This signal could prove to be extremely useful in the event where the main systems would not respond. The measurements sent by the beacon could, in that case, be used to determine the cause of the possible malfunction.

It is important to note that, to be useful in emergency conditions, such a system must be independent from the other main subsystems of the satellite.

## 3.3 Constraints of mission

### 3.3.1 Dimensional constraints

As indicated before, OUFTI-1 will take the form of a CubeSat. This standard imposes external dimensions that correspond to a 10cm cube. Its mass is also limited, to 1 kg. It is worth noting that the shape constraint is only in effect during the launch, and that after accurate timing following the ejection from the launcher, external structures may be deployed. These typically include solar panels and radio antennas. In the case of OUFTI-1, only the antennas need to be deployed.

### 3.3.2 Launcher constraints

The OUFTI-1 CubeSat is planned to be put on orbit by the Vega launcher. Vega is a new expendable launch vehicle, developed by the Italian and the European Space Agencies. Its maiden flight, scheduled for the end of 2009, will bring to orbit, in addition to the main payload, six European CubeSats, including OUFTI-1.

#### Mechanical interface

The CubeSats are fixed to the launcher with a structure that also serves to the deployment. This system is called a *POD*, for *Picosatellite Orbital Deployer*. Several kinds of such devices exist; the one aboard the Vega launcher will be a P-POD, for Poly-POD<sup>1</sup>. It consists of a rectangular box containing three CubeSats stacked on each other. The box comprises a door, that is kept closed during the launch, and that is opened for the ejection; the three CubeSats are then pushed out of the box by a spring-loaded mechanism. The structure of each CubeSat must include small springs at its base, called *separation springs*, to ensure that the three CubeSats of the same POD will separate from each other after ejection (Fig. 3.2).

#### Electrical interface

The CubeSats do not have any electrical interface with the launcher, nor any umbilical connections once integrated on the payload platform of the launcher. Before the integration, the CubeSats can however be connected to some electrical ground support equipment, typically to recharge the batteries.

---

<sup>1</sup>The Poly-POD is named after the California Polytechnic State University (CalPoly), where it was developed.

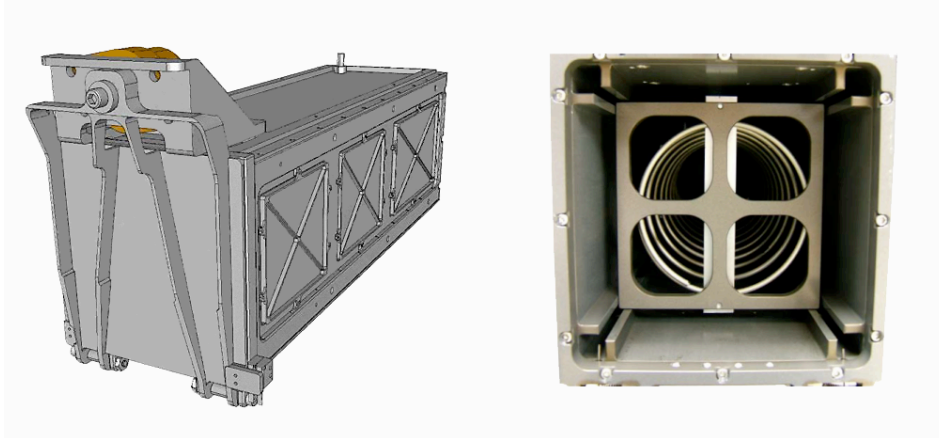


Figure 3.1: P-POD perspective and cross-section (source: [CDS08]).

For safety reasons, the CubeSats cannot be powered on during the launch. A means must therefore be included to switch on the electronics of each CubeSat after its ejection from the POD. This is typically accomplished with one or several so-called *deployment switches*, or *launch switches*. These small switches are placed at the base of the CubeSat (Fig. 3.2), and they are kept (electrically) open during the launch, as a result of the stacking of the three CubeSats inside the POD. Following ejection, the switches close, and their closing turns on the electronics.

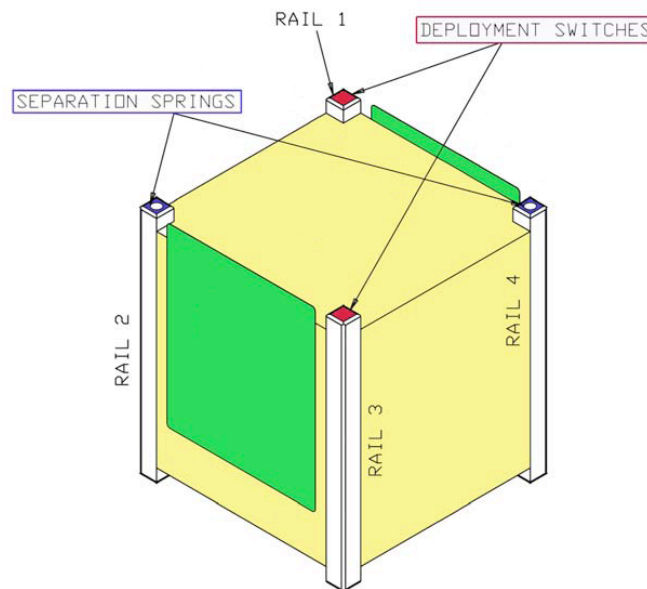


Figure 3.2: Drawing of a CubeSat showing deployment switches and separation springs (source: [CDS08]).

## Ejection and timing

After ejection from the POD, a CubeSat must respect minimal timings before its initial operations. More precisely, external structures (e.g. antennas and/or solar panels) can be deployed no earlier than 15 minutes after ejection. Radio transmissions may start at the same time, but only at a low power. Such low power radio transmissions are typically those of a radio beacon. The primary radio transmitters can only start transmissions 30 minutes after ejection.

### 3.3.3 Orbit constraints

Since the CubeSats are (almost) always embarked on launchers as secondary payloads, they usually have to cope with the orbit chosen for the main payload of the launcher. In our case, the CubeSats will be placed on an elliptical low Earth orbit with the following parameters:

- perigee altitude: 354 km;
- apogee altitude: 1,447 km;
- inclination:  $71^\circ$ .

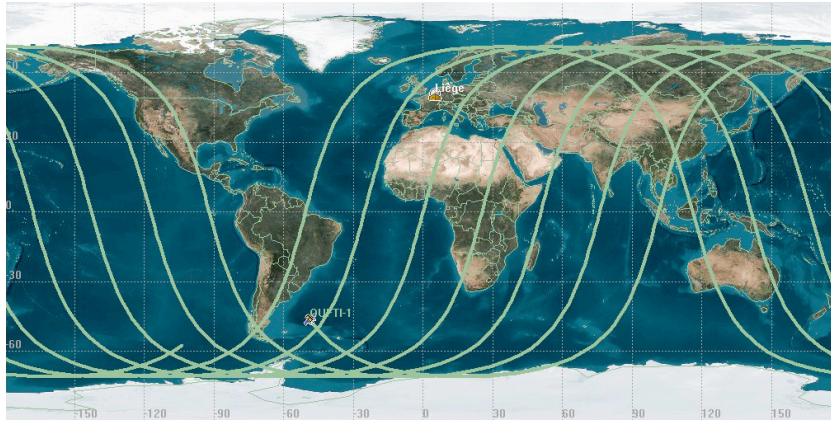


Figure 3.3: Ground track of the orbits of OUFTI-1 for 12 hours, computed for the orbit parameters available in 2008 (source: [Galli08]).

The parameters of the orbit are of great importance, because they determine several characteristics of the mission that can affect the design of several subsystems of the satellite.

## Time in view

The time in view, or access time, correspond to the time during which a particular point on Earth will be within the footprint of the satellite, and therefore able to

communicate with it. In particular, we are interested in the time in view of our main ground station, located in Liège, Belgium.

A complete analysis can be found in [Galli08, Beukelaers09]. It was performed by studying the best and worst cases. In the worst case, the perigee is over our ground station in Liège, Belgium: the speed of the satellite is then the highest over Liège, the time in view is only 30 min/day, and the maximum continuous access time is only about 8 minutes. In the best case, the apogee is over our ground station in Liège, Belgium: the speed of the satellite is then the lowest over Liège, the time in view is 104 min/day, and the maximum continuous access time is about 17 minutes. These numbers are be useful for evaluating the quantity of data that could be transferred between the satellite and the ground station, each day, or at once (see [Evrard09]).

#### **Eclipse duration**

The eclipse duration corresponds to the length of the periods during which the satellite does not receive any sunlight. The satellite thus has to rely on power stored in its batteries. A detailed study was performed in [Galli08]. What is worth keeping in mind is that available on-board power will be limited at all times, and that low-power consumption will often have to be the primary driving factor in the design of the subsystems.

#### **3.3.4 Environment constraints**

Operating electronics in the space environment is particular in several ways. Here are the three main characteristics that will have to be taken into account when designing electronics for the CubeSat.

##### **Thermal environment**

The range of temperatures to which a spacecraft is exposed is very wide. In low Earth orbit, these temperatures can go from  $-180^{\circ}\text{C}$  to  $150^{\circ}\text{C}$ . Moreover, the vacuum prevents the convection cooling of electronic components, which is typically used on Earth. Any component that generates some heat will have to dissipate it in some way. A careful design of the electronic boards, with this problem in mind, is therefore needed. A detailed analysis of this problem is available in [Jacques09].

##### **Mechanical conditions**

During launch, the CubeSat will be subjected to harsh mechanical conditions generated by the launcher: intense vibrations and a strong acceleration. This is



particularly the case with the Vega rocket, since it only uses solid fuel motors for its first three stages. Such motors typically generate even more violent vibrations than the more common liquid fueled motors. It is therefore important to design attachments for the electronic boards and for the electronic components on these boards, that can withstand these conditions. Corresponding care will also be taken in the choice of the electronic components.

A detailed analysis of this problem is available in [Pierlot09].

## Radiations

The most serious concerns when operating electronics in space come from the radiations they are exposed to. High-energy ionizing particles exist in space as part of the natural background, referred to as *galactic cosmic rays*. These particles can cause temporary or permanent damage to microelectronic devices. One distinguishes three different types of problems.

- The **single-event upset** is a change of state of one node in a microelectronic device, caused by ionization in, or close, to this node. The resulting error can be said to be a *soft* error, since it does not involve a permanent change to the physical circuit. For example, in a memory device, a single-event upset could cause the flip of a bit in the memory; however, this bit can later be rewritten to its appropriate value.
- The **single-event latchup** is a temporary short circuit caused by a particle creating a low-impedance path between the power supply rails of the circuit. The recovery from a latchup necessitates a power cycle.
- The **single-event burnout** is a hard error in which the device fails permanently in some way. Various physical causes can lead to a single-event burnout.

Let us note that, besides those single-event effects, any given component can withstand a limited total dose of radiation during its lifetime (called the *total ionizing dose*), before failing permanently.

Around the Earth, the predicted charged particles are usually trapped and concentrated in the Van Allen radiation belts. These two imaginary torusses are maintained around the Earth by its the magnetic field, and the inner one extends from an altitude of 700 km to 10,000 km above the Earth's surface (Fig. 3.4). In our case, the planned orbit goes through this inner belt, and our CubeSat is thus likely to receive high doses of radiation. Protection measures must thus be considered; these measures can be passive (e.g. a careful selection of components or a metallic shielding around some of them) and/or active (e.g. by making the software tolerant to single-event upsets).

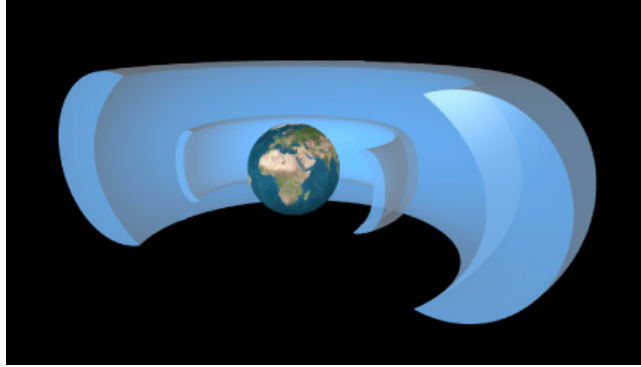


Figure 3.4: Representation of the Van Allen radiation belts (source: ESA Science and Technology website).

A detailed analysis of this problem is available in [Beukelaers09].

---

## Chapter 4

# Architecture and subsystems of OUFTI-1 satellite

This chapter explains how the project is divided into different subsystems, and gives an overview of each of these subsystems.

A project as ambitious as designing a nanosatellite can only be tackled when divided into a series of smaller subprojects. In our case, an additional, practical, constraint, was that these subprojects, were to be handled as much as possible by students, as the subject of their master thesis. A vertical division was therefore carried out, which turned the project into a set of subsystems. The development of these subsystems could therefore be carried out relatively independently, by teams of one or more students.

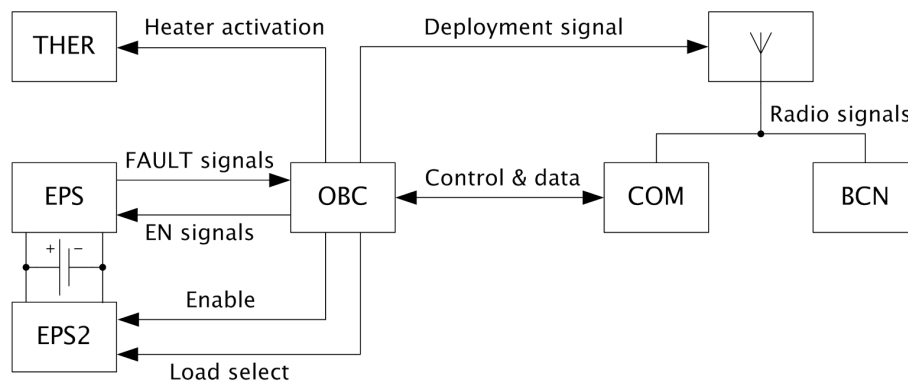


Figure 4.1: Satellite subsystems and information flow between them.

Figure 4.1 gives an overview of the complete system, and illustrates the information flow between the various subsystems. The following sections present in more details each of these subsystems, with their role, the technical solution they use, and

the requirements of their electrical interface. The electrical requirements of the interface of each subsystem are presented in a table, that lists the individual electric signals that are used to interface this subsystem with the other ones.

## 4.1 Mechanical structure

### Role

The mechanical structure of the satellite must provide means of organizing and securing all its elements together (the electrical boards, the other mechanical elements, the electrical connections between them, etc.).

### Means

OUFTI-1 uses Pumpkin's CubeSat kit as its main structure, since this solution shortens the overall development time of the project. The internal configuration of the electronic boards is the one proposed by Pumpkin: the PCBs are stacked "horizontally", and the electrical connections between the boards are accomplished with solid (stackable) interboard connectors. The structure also provides means for attaching the solar panels. These panels will be attached to the sides of the cube, and thus do not need any deployment mechanism. More details on the mechanical structure are available in [Pierlot09].

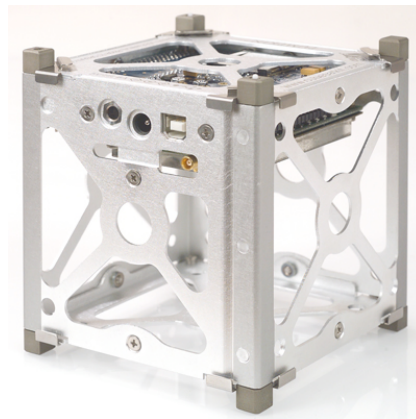


Figure 4.2: CubeSat kit structure used for OUFTI-1 (source: Pumpkin CubeSat kit website).

### Electrical interface

Inputs	Outputs
None	None

## 4.2 Electrical power supply

### Goal

The electrical power supply (EPS) is responsible for producing electrical power, storing it, and distributing it to the various on-board subsystems in the form they need it (i.e. the required current at the appropriate, regulated voltages). The EPS thus comprises solar cells and batteries, and it must also provide a means of recharging these batteries on the ground, before launch. Its proper working is critical to the operations of the satellite, and reliability is thus its first design criterion.

### Means

For simplicity, and thus for reliability reasons, the EPS is chosen to be passive. It thus comprises mainly batteries and power converters, in addition to the solar cells. The management of the use of electrical power aboard the satellite is left to the on-board computer (OBC), that we will introduce in Sect. 4.9. In order to protect the global system against latchups in components of the clients of the EPS, each of these clients is powered through a current-limiting switch. These switches can limit the current to a predefined threshold, and they then set a **FAULT** signal; this signal indicates the faulty condition to the OBC. Each switch also accepts an **EN** (for *enable*) input signal, that can be used by the OBC to switch on and off a particular subsystem. Note that all these switches are conceptually thought about as being part of the EPS, even though they are individually related to other subsystems, and physically located on the boards of these other subsystems. More details on the EPS subsystem are available in [Thirion09].

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"><li>• <b>EN_xxx</b><sup>1</sup> Enable signals (one for each client of the EPS).</li></ul>	<ul style="list-style-type: none"><li>• <b>POWER_3.3V_BUS</b>, <b>POWER_5.0V_BUS</b>, <b>POWER_5.0VBIS_BUS</b> Regulated electrical currents at different voltages;</li><li>• <b>FAULT_xxx</b> Fault signals (one for each client of the EPS).</li></ul>

---

<sup>1</sup>The suffix **xxx** identifies the client.

## 4.3 Radio-communications

### Goal

The radio-communication subsystem (COM) comprises, conceptually, a radio receiver and a radio transceiver. These can be used to receive telecommands, transmit telemetry, and, in our case, act as a repeater for communications between two ham-radio operators.

### Means

The radio-communication subsystem of OUFTI-1 uses the D-STAR protocol. In addition to that, and for various reasons, including safety, both the receiver and the transmitter are capable of using another, non-experimental protocol, the AX.25 protocol. From the point of view of the OBC, let us note that the whole radio-communication system is built with integrated circuits that are digitally controllable. Most of the encoding/decoding work is left to the OBC. More details on the COM subsystem are available in [Henrard09, Mahy09].

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"><li>• POWER_3.3V_BUS, POWER_5.0VBIS_BUS 3.3V and 5.0V currents from EPS.</li><li>• COM_ADFxxx Data and control signals for the (de)modulation circuits (see [Henrard09, Mahy09]).</li><li>• ANT_SIGNAL Radio signal from the antennas.</li></ul>	<ul style="list-style-type: none"><li>• COM_ADFxxx Data and control signals from the (de)modulation circuits (see [Henrard09, Mahy09]).</li><li>• ANT_SIGNAL Radio signal to the antennas.</li></ul>

## 4.4 Radio beacon

### Goal

The beacon (BCN) aboard OUFTI-1 will serve as an emergency radio transmitter, in the case where the satellite's main transmitter does not operate properly. The BCN takes various measurements aboard the satellite, and transmits them via a specific radio channel. These measurement are intended to allow the operator of the ground control station to diagnose the health of the various subsystems.

### Means

The beacon is designed to be as autonomous as possible. In particular, it does not rely on any other subsystem to perform the measurements. Moreover, the development of the BCN was left in care of a completely independant team.

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"> <li>• POWER_3.3V_BUS 3.3V current from EPS.</li> <li>• BCN_INPUTxx<sup>1</sup> Analog signals representing the parameters that the beacon must measure.</li> </ul>	<ul style="list-style-type: none"> <li>• ANT_SIGNAL Radio signal to the antenna.</li> </ul>

## 4.5 Antennas

### Goal

The radio-communication subsystems (COM and BCN) require two antennas, one of 50 cm in length and one of 17 cm. These antennas can be deployed no earlier than 15 minutes after the ejection from the launcher. A system must therefore be devised to hold the antennas in place during the launch, and allow for a reliable deployment once the OBC decides to do so.

### Means

The antennas are made of flat wire, that is folded during the launch, and held in place by a piece of string. For the deployment, an electrical signal feeds a heating element that melts this string, and releases the antennas. For reliability, two identical, redundant release systems are in fact included, and can be triggered with distinct control signals. Note that there is no direct means in that deployment system to check whether the antennas are correctly deployed. However, this could be accomplished by operating the radio-communication system, and measuring the standing wave ratio (SWR). More details on the antennas are available in [Wertz09].

---

<sup>1</sup>The suffix xx identifies, with a number, each analog input of the beacon.

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"> <li>• <b>ANT_SIGNAL</b> Radio signal to the antennas (from COM and BCN).</li> <li>• <b>POWER_?</b> (the necessary voltage has not yet been defined) Current from EPS for the deployment system.</li> <li>• <b>ANTENNA-DEPLOYMENT1_EN, ANTENNA-DEPLOYMENT2_EN</b> Enable signals for the two deployment systems.</li> </ul>	<ul style="list-style-type: none"> <li>• <b>ANT_SIGNAL</b> Radio signal from the antennas (to COM and BCN).</li> </ul>

## 4.6 Attitude control

### Goal

The role of the attitude control subsystem is to keep the satellite in the best possible orientation, with respect to the mission objectives.

### Means

The radio-communication subsystem of OUFTI-1 does not require the spacecraft to be continuously pointed in a precise direction; therefore, the attitude control system was chosen to be passive. It is composed of magnets and rods of hysteretic materials. They will, respectively, align themselves with the Earth magnetic field, and dampen the rotation of the satellite. The choice of a passive system offers the advantages of being lightweight and of not requiring any processing or electrical power. More details on the attitude control system are available in [Hannay09].

### Electrical interface

Inputs	Outputs
None	None



## 4.7 Thermal control

### Goal

The thermal control subsystem (THER) of the satellite must ensure that all its components are within an appropriate temperature range so that they can function properly.

### Means

The thermal behavior of the satellite was analyzed by simulations (see [Jacques09]), and the control system was chosen to be (mainly) passive. It relies on a well thought-out layout of the elements and on a proper choice of materials to ensure that each component will stay within its allowable temperature range at all times. However, the most sensitive components, the batteries, need an additional protection against the cold. They are equipped with small resistive heaters. The activation of these heaters is to be controlled by the OBC. More details on the THER subsystem are available in [Jacques09].

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"><li>• POWER_? (the necessary voltage has not yet been defined) Current from EPS for the deployment system.</li><li>• BATTERY-HEATER_EN Enable signal for the battery heater.</li></ul>	None

## 4.8 Experimental electrical power supply

### Goal

In addition to the main electrical power supply, OUFTI-1 is equipped with a second, innovative, experimental digitally-controlled electrical power supply (EPS2), that was designed in collaboration with Thales Alenia Space ETCA. Its sole purpose is to be tested in space, and it has no vital role aboard the satellite.

### Means

From the point of view of the OBC, the EPS2 can be seen as a black box payload that can be activated on demand. The EPS2 uses current from the battery, and convert it to a voltage of 3.3V. It can then either feed a resistive load, or be

connected to the main EPS power supply rail, in parallel with the output of the main EPS. The choice of the output can also be set by the OBC. More details on the EPS2 subsystem are available in [Ledent09].

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"> <li>• <b>EN_EPS2</b> Enable signal to turn the subsystem on or off.</li> <li>• <b>EPS2_LOAD_SELECT</b> Load select signal to connect the output to a resistive load or to the main EPS power rail.</li> </ul>	<ul style="list-style-type: none"> <li>• <b>POWER_3.3V_BUS</b> Regulated electrical current, fed on the 3.3V power rail (in parallel with the main EPS).</li> </ul>

## 4.9 On-board computer

### Goal

The on-board computer (OBC), in short, can be said to be responsible of all high-level monitoring and control tasks aboard the satellite. More precisely, the following roles are attributed to it.

- Perform the initial satellite operations (antenna deployment, first activation of the other subsystems) according to a predefined sequence.
- Interface with the radio-communication circuits, and perform AX.25 and D-STAR encoding and decoding.
- Handle telecommands received on the uplink channel.
- Perform measurements aboard the satellite (see Sect. 4.10).
- Store relevant measurements until they can be sent to the ground station.
- Respond to telemetry requests by sending present or past (stored) measurements.
- Provide a time reference.
- Perform electrical power supply management, by enabling and disabling other subsystems in predefined conditions (e.g. when battery voltage is too low).
- Perform power cycling in case of latchup in a subsystem (detected with a **FAULT\_xxx** signal).
- Manage the experimental electrical power supply, by enabling and disabling it in predefined conditions.

- Manage the D-STAR payload, by configuring it (e.g. for Doppler compensation) according to data received by specific telecommands.

### Means

They will be discussed beginning with Chap. 5.

### Electrical interface

Inputs	Outputs
<ul style="list-style-type: none"> <li>• POWER_3.3V_BUS, POWER_5.0V_BUS 3.3V and 5.0V currents from EPS.</li> <li>• FAULT_xxx Fault signals (one for each client of the EPS).</li> <li>• COM_ADFxxx Data and control signals from the (de)modulation circuits of the COM subsystem.</li> </ul>	<ul style="list-style-type: none"> <li>• EN_xxx Enable signals (one for each client of the EPS).</li> <li>• COM_ADFxxx Data and control signals for the (de)modulation circuits of the COM subsystem.</li> <li>• BATTERY-HEATER_EN Enable signal for the battery heater.</li> <li>• ANTENNA-DEPLOYMENT1_EN, ANTENNA-DEPLOYMENT2_EN Enable signals for the antenna deployment mechanism.</li> <li>• EN_EPS2 Enable signal for the experimental electrical power supply.</li> <li>• EPS2_LOAD_SELECT Load select signal for the experimental electrical power supply.</li> </ul>

## 4.10 On-board measurements

Taking measurements aboard the satellite is of prime importance, since it is the only way, for the OBC, and for the operators of the ground station, to determine the state of the satellite and of its various subsystems. Each measurements belongs to one of the two possible categories or to both.

First, the *housekeeping parameters* reflect the state of the (generally) most critical components on-board. They are very important because they bring information to the OBC that is used to decide about some actions to perform aboard the satellite.

For example, the temperature of the batteries is a housekeeping parameter, because it is used by the OBC to decide when to activate the battery heater.

Second, the other measurements, that we will call the *science parameters*, are not used in any automatic control loop in the satellite. They are however very interesting for analysis on the ground. They provide information on the behavior of the on-board systems and payloads. These parameters are typically measured automatically and periodically; the samples are stored aboard the satellite, and regularly downloaded for analysis on the ground.

Let us note again that a parameter can belong to both of the aforementioned categories.

The list of measurements to be taken aboard OUFTI-1 was established with the designer(s) of each subsystem. It was important to determine which parameters were the most important. Indeed, the limited capacities of a nanosatellite do not allow for an extremely large number of measurements.

The complete list of selected parameters to be measured is given in Appendix A. Additional information is available in [Evrard09, Henrard09, Ledent09, Mahy09, Thirion09].

---

## Chapter 5

# Hardware: Requirements of on-board computer (OBC)

### 5.1 Processing power

The first criterion for designing a computer for an application as complex as a CubeSat is probably the processing power. Even though other characteristics, such as power consumption, seem to be as important, a processor would be of little interest if it cannot handle all the real-time tasks that need to be accomplished.

In our case, the critical real-time operations are related to the EPS and COM subsystems.

First, the OBC needs to manage the use of electrical power available from the EPS aboard the satellite. In other words, the OBC must enable and disable the various subsystems depending, notably, on the available electrical power. The OBC must also make the decision of switching off faulty subsystems, typically those that draw too much current because of a short circuit. One can easily determine that a trivial implementation of these tasks would need very little processing time on any common microcontroller.

Second, the OBC needs to process in real time data from the COM subsystem. Digital integrated circuits are used for the modulation and the demodulation of the radio signals, but the encoding and the decoding of the data are left to the OBC. In the case of D-STAR, this involves processing, for each communication channel, a 4800 bit-per-second (bps) flow with a number of complex algorithms ((de)scrambling, (de)interleaving, (de)convolution, etc.). In the OUFTI-1 project, the design and the programming of the COM part of the software was chosen to be the responsibility of the COM team, and the corresponding analysis of the required

processing power was therefore performed as part of their work. This analysis comprises, furthermore, the evaluation of the size in memory of the required data structures, such as software buffers. The details are available in [Henrard09, Mahy09]).

## 5.2 Data flow paths

To design an appropriate electrical architecture for our satellite, it is crucial to determine where data must be exchanged in the satellite.

- Data flow paths related to the **EPS subsystem**

It is clear that the computer must be able to read the status of the various subsystems, and be able to send them enabling/disabling commands. It appeared clear that this information would be best exchanged through single digital lines, especially for reliability (as opposed to a bus-based communication). However, we must keep in mind that this solution requires a significant number of I/O lines on the processor.

- Data flow paths related to the **COM subsystem**

There is a potentially large amount of data that could have to be exchanged with the COM subsystem (telemetry in particular). Thus, if one decides to use a dedicated processor for the COM system, a dedicated data flow path would be needed between this COM processor and the OBC's processor. Besides this, it is important to note that the digital circuits used in the COM subsystem need a large number of digital I/O lines, on the order of 20 (see [Henrard09, Mahy09]).

## 5.3 Other concerns

Here are some other concerns, listed below by decreasing order of importance, that must be taken into account in the design of the overall electrical architecture.

- **Reliability**

One important aspect of equipment for space is that it generally cannot be serviced or repaired once in orbit. It is thus crucial to design fail-safe systems, by minimizing the number of single points of failure (SPOF). A SPOF is defined as a part in the system which, if it fails, will prevent the entire system from working. It is therefore important to include redundancy in the system. It is also wise to limit, when possible, the interdependence between the various subsystems (i.e. make a component be able to work alone if another one fails).

- **Protection against radiations**

For similar reliability reasons, it is desirable to have some means of protecting the hardware against the effects of space radiations. Typically, these protections would reduce, at the same time, the probability of soft errors (single-event upsets and latchups) and of fatal events for the components (single-event burnouts).

- **Power consumption**

The power available aboard the satellite is extremely limited. Indeed, there is typically less than 1W available for all the subsystems, including for the radio transmitters. It is therefore very important to take the power consumption into account when selecting components or communication protocols.

- **Analog inputs**

Several measurements will have to be taken during the operation of the satellite (see Sect. 4.10). It is therefore necessary to have the capability of converting several analog signals to digital values.

- **Mass memory**

It is necessary to have a way of storing data aboard the satellite. This memory is to be used to store measurements, and information on the behavior of the various subsystems. Therefore, there is no need for a large capacity or for significant transfer rates.

---

## Chapter 6

# Hardware: Architectures of CubeSat on-board computers

This chapter presents an overview of various solutions adopted for other CubeSats, for their overall electrical architecture and for their on-board computer (OBC). We will then propose a specific architecture adapted to the OUFTI-1 nanosatellite.

### 6.1 OBC architectures of other CubeSats

#### 6.1.1 Centralized and distributed architectures

The electrical architecture of the whole satellite, in terms of information flow, is strongly related to the design of the OBC. Below, we discuss two general trends that can be considered for the design of a nanosatellite.

Figures 6.1 and 6.2 illustrate a centralized architecture and a distributed architecture, respectively. In a centralized architecture, the processing power is concentrated in one node, the OBC. This OBC is connected to all the subsystems it has to interact with. These connections can use either direct lines, or a bus-based system, but this choice will often be dictated by the nature of the components the OBC needs to communicate with. Some advantages and disadvantages are listed in Table 6.1. Let us mention that a centralized architecture is fairly simple to develop and to debug, but may lack some flexibility and — more importantly — redundancy.

In a distributed architecture, each subsystem is typically equipped with its own processor or microcontroller. Information transfer can occur between two subsys-



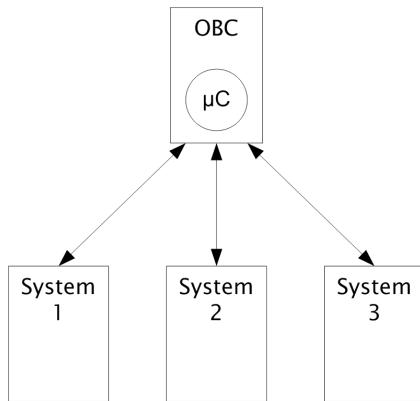


Figure 6.1: Centralized architecture.

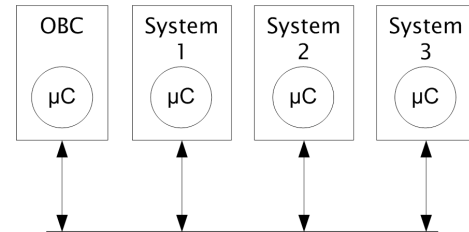


Figure 6.2: Distributed architecture.

<b>Centralized architecture</b>
<ul style="list-style-type: none"> <li>⊕ Less complex, thus probably more reliable</li> <li>⊕ Communication problems between subsystems less likely</li> <li>⊕ Simpler hardware</li> <li>⊖ Large single point of failure (the computer itself)</li> <li>⊖ Neither flexible nor scalable</li> <li>⊖ The processing power has to be shared between subsystems</li> </ul>
<b>Distributed architecture</b>
<ul style="list-style-type: none"> <li>⊕ Modular and scalable</li> <li>⊕ Many possible communication paths (system not entirely dependent on the OBC)</li> <li>⊕ Independent testing of subsystems more practical</li> <li>⊖ More complex hardware and software</li> <li>⊖ One faulty subsystem could perturb the bus and thus all the other subsystems</li> <li>⊖ As a whole, trickier to debug</li> </ul>






Table 6.1: Pros and cons of centralized and distributed architectures.

tems without going through the OBC. The computer's role is typically to handle and supervise the tasks that involve several subsystems. Such an architecture can use point-to-point connections, but it more often uses a bus-based system. The pros and cons of this type of architecture are also given in Table 6.1. The main advantage is that it is more flexible and scalable than a centralized architecture. The different parts of the software may be individually simpler, and can be developed by independant teams, but a rigourous coordination between them is needed. Moreover, the additional intermodule communication code may add significant development time.

Note that the two types of architecture can be combined in various ways, and that hybrid solutions are perfectly possible. In our case, a centralized architecture seemed to be a good choice, since the complexity of the software to be executed on the OBC would still be reasonable. The problem of having a large single point of failure is addressed later.

### 6.1.2 Processors used by other CubeSats

Table 6.2 lists a series of CubeSats together with the main characteristics of their OBC. Note that this survey is largely based on the one realized by the SwissCube team ([SwisscubeCDMSB]).

Processor	Clock	Memory	Power supply
<b>AAUSat-I</b>			
Siemens C161	10 MHz	RAM: 512 kB ROM: 512 kB Flash: 256 kB	5 V 150 mW
<b>AAUSat-II</b>			
Atmel ARM7	8/40 MHz	RAM: 2 MB Flash: 8 MB	3.3 V 80 mW @ 8 MHz 300 mW @ 40 MHz
<b>CanX-1</b>			
Atmel ARM7	40 MHz	RAM: 512 kB ROM: 128 kB Flash: 32 MB	3.3 V 400 mW

---

6. HARDWARE: ARCHITECTURES OF CUBESAT ON-BOARD COMPUTERS

---

Processor	Clock	Memory	Power supply
<b>CP1 and CP2</b>			
PIC 18LF6720	4 MHz	RAM: 4 kB ROM: < 1 kB Flash: 128 kB	3 V < 10 mW
<b>CubeSat XI-IV</b>			
PIC 16F877	4 MHz	RAM: 368 bytes ROM: 32 kB	5 V 10 mW
<b>CUTE-1</b>			
Hitachi H8/300	<i>unknown</i>	RAM: 512 kB ROM: 256 kB Flash: 4 MB	<i>unknown</i>
<b>DTUSat-I</b>			
Atmel AT91M40800	16 MHz	RAM: 1 MB ROM: 16 kB Flash: 2 MB	<i>unknown</i>
<b>KUTESat</b>			
PIC 18F4220	8 MHz	RAM: 512 MB ROM: 4 kB EEPROM: 256 bytes	3.3 V < 10 mW
<b>Mea Huaka's</b>			
Z-World RabbitCore RCM2000	1.8 - 30 MHz	RAM: 368 bytes ROM: 32 kB	5 V Max: 650 mW Typical: 300 mW
<b>MEREOPE</b>			
Freescale MC68HC812A4	8 MHz	RAM: 1 kB EEPROM: 4 kB	5 V 150 mW
<b>nCube</b>			
Atmel AVR ATmega32L	4 MHz	RAM: 2 kB ROM: 32 kB Flash: 1024 bytes	3.3 V 12 mW

---

Processor	Clock	Memory	Power supply
<b>Pumpkin CubeSat Kit FM430 (Delphi C-3, Libertad-1, ...)</b>			
Texas Instruments MSP430	8 MHz	RAM: 5 or 10 kB Flash: 50 or 55 kB	3.3 V or 5 V < 10 mW
<b>Sacred and Ricon</b>			
PIC 16C77	4 MHz	RAM: 64 kB	5 V < 10 mW

Table 6.2: Characteristics of processors and microcontrollers used in other Cube-Sats.

It can be observed from this survey that a very broad range of microcontrollers and microprocessors are used in nanosatellites. These components exhibit very different performances and power requirements. We can therefore conclude that there is no one-fits-all solution to the choice of a processor for the OBC of a CubeSat. This choice will thus essentially be based on the requirements of the mission.

## 6.2 OBC architecture of OUFTI-1

### 6.2.1 Processor of OUFTI-1

In the above overview of solutions, one of the solutions seemed particularly attractive in our situation: Pumpkin’s FM430 flight computer (Fig. 6.3). Its very low power consumption, and the fact that it has already flown several times in space, make it very attractive. It also seemed reasonable to us not to start from scratch, but rather from a proven and off-the-shelf solution. Using an FM430, or at least one of the Texas Instrument’s MSP430 microcontrollers, thus became the baseline.

The COM team had to determine whether the processing capabilities and the memory resources of an MSP430 would be sufficient to process D-STAR data. A preliminary analysis led to a positive conclusion, and allowed us to design a centralized and quite simple architecture. A single processor could indeed take care of all the “intelligent” tasks aboard the satellite. This eliminates the need for complex – and failure-prone – communication interfaces between multiple processors.

As indicated before, the main drawback of a centralized architecture is that it presents a large single point of failure: the processor itself. This issue is addressed,

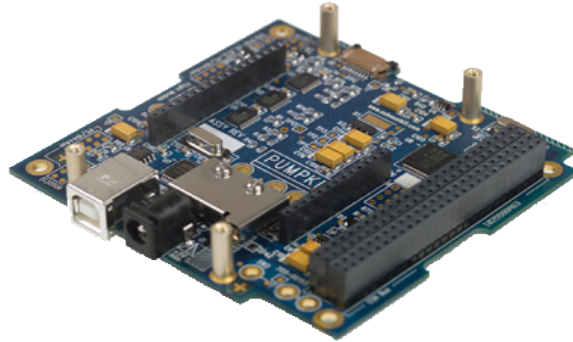


Figure 6.3: Pumpkin’s FM430 (source: Pumpkin CubeSat kit website).

in our case, by adding a second, fully redundant processor to the first one. The strategy consists in the following: as long as one computer, called the *default OBC* (later referred to as OBC2) is working properly, the other one, named the *backup OBC* (later referred to as OBC1) stays silent but monitors the activity of the other one. In the event where the default OBC becomes ineffective, the backup OBC detects this situation, and immediately takes over by starting, by itself, to control the normal sequence of operations.

This approach is practically very effective, and, moreover, very simple to implement. Indeed, at least conceptually and at the hardware level, all the corresponding I/O pins of both processors can be tied together. Unfortunately, Pumpkin’s FM430 boards use a predefined bus standard, namely the CubeSat bus, which includes various signals that cannot be directly connected together between two FM430 board (see Appendix B). Simple approaches were proposed, such as fitting a custom board between two FM430s, and interrupting the problematic signals at that level (e.g. by removing the corresponding physical pins of the connector). Another solution was however chosen. It consists in developing a custom computer board, in place of one of the FM430s. This board would basically be a simplified replica of an FM430, but it would give us more flexibility, as well as the expertise and the experience needed to avoid being limited to use commercial boards (FM430s) in our future CubeSats.

This custom board, that we chose to name OBC2, is chosen to be the default OBC, whereas the FM430, considered more reliable, and named OBC1, is chosen to be the backup OBC.

### 6.2.2 Peripherals of the OBC of OUFTI-1

The MSP430 offers very little internal memory (55 Kb at the maximum), so we decided to include an external storage memory. Pumpkin’s FM430 includes a SD

card socket and electrical interface, but SD cards are relatively power hungry, and their capacity is well beyond our needs. We therefore decided to include a small EEPROM chip, than we can chose to be more adapted to our needs.

The MSP430 includes a 10-bit, 8-channel, analog/digital converter (ADC), but we need, in our application, much more than 8 analog inputs. We therefore chose to use external ADCs. The internal ADC of the MSP430 is not used at all, which allows us to use its inputs pins as digital I/O, rather than as analog inputs.

### 6.2.3 Data flow paths in OUFTI-1

As indicated before, using a centralized architecture allows one to simplify the communication means between the subsystems inside the satellite. The following solutions were adopted.

- Data flow paths related to the **EPS subsystem**  
We use single I/O lines to read the status of, and enable/disable, the various subsystems.
- Data flow paths related to the **COM subsystem**  
The modulation/demodulation integrated circuits are controlled via a set of dedicated I/O lines.
- Data flow paths related to the **EEPROM and ADCs**  
These peripherals where chosen with an I<sup>2</sup>C interface. This makes their physical placement on different boards in the satellite easier (which is desirable for the ADCs).

### 6.2.4 Protection against radiations

Various measures can be taken to limit the impact of radiations on the electronics.

The first approach, traditionnally applied in the space industry, is to use exclusively radiation-hardened, or rad-hard, components. These are, by design, much less vulnerable to radiations. Their availability may however be a problem, and their cost would be prohibitive in a project like ours.

The second approach is to purchase common components in lots, and to perform a screening to select the most resistant ones. This can be done, e.g. by exposing

them to a small source of radiations and by counting the occurrences of non-destructive single-event effects ([Cutler06, Kayali00]). The drawback is that this method could take a lot of time, and needs specialized equipment.

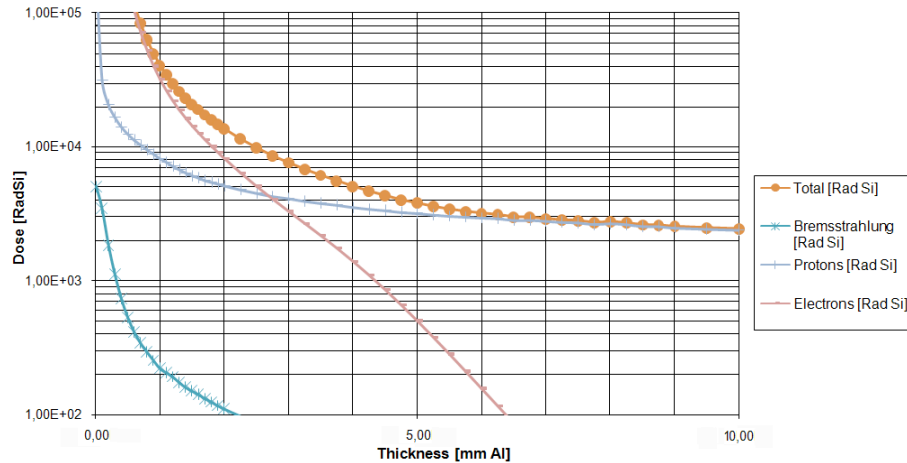


Figure 6.4: Total dose of radiation received during one year aboard OUFTI-1 under various thicknesses of aluminum shielding (source: [Beukelaers09]).

The third approach is to provide an additional external shielding, either to the whole electronic board, or to the most sensitive components. This is usually done with aluminum boxes or screens, since they provide an appreciable shielding against most types of radiations for a minimal weight. Figure 6.4 shows the total received dose aboard OUFTI-1 during one year, under various thicknesses of aluminum shielding; it can be observed that a shielding of 2 or 3 mm already gives a significant protection. As an indicator of usefulness of the various levels of shielding, Table 6.3 gives typical failure doses for various kinds of (non rad-hard) electronic components ([Cutler06]). This third approach was selected for our mission, but, to date, it has not been decided whether this shielding would protect the entire satellite or only particular components.

Components	Failure dose (KRadSi)
Linear IC's	2 – 50
Mixed signal IC's	2 – 30
Flash memories	5 – 15
DRAMs	15 – 50
Microprocessors	15 – 70

Table 6.3: Typical failure doses of various types of commercial off-the-shelf electronic components (source: [Cutler06]).

A detailed analysis of this problem is available in [Beukelaers09]. A discussion on the use of global shielding or local shielding for OUFTI-1 is available in [Pierlot09].

### 6.3 Overall electrical architecture of OUFTI-1

Appendix C shows the overall electrical architecture that results essentially from the various choices that have just been made and justified.



---

# Chapter 7

## Hardware: Design of the OBC electronic boards

This chapter presents in detail the design of the two electronic boards of the OUFTI-1 OBC. As a reminder, the first board, OBC1, is Pumpkin's FM430, which is available off-the-shelf; the second board, OBC2, is a custom board, which we discuss the design of in Sect. 7.2.

### 7.1 OBC1: Detailed analysis of the FM430 board

Before starting to work on the design of the custom board, OBC2, we began by carefully analyzing how the FM430 (OBC1) was designed. This analysis task did not pose much problem since we had at our disposition a block diagram (Fig. 7.1) and all the electrical schematics of this board ([Pumpkin08]).

We now describe the various fonctionnalités of the FM430 board. We also indicate the features that will be carried over from OBC1 to our homemade OBC2.

- **Clocks**

The MSP430 features two internal crystal oscillator circuits. On the FM430, one of the MSP430 oscillators is connected to an external 32.768 KHz watch crystal, and the second is connected to a 7.3728 MHz crystal. Note that the frequency of this second crystal is close to the maximum of 8 MHz allowed by the MSP430. We will include the same crystals on OBC2.

- **JTAG**

The FM430 includes a connector for the JTAG interface of the MSP430. This

Figure 7.1: Pumpkin’s FM430 block diagram (source: [Pumpkin08]).

- **USB connector and electrical interface**

- External 5V power

- **MHX transceiver socket and electrical interface**

The MHX transceiver is an off-the-shelf radio modem, that can be interfaced with the FM430. Since we developed our own radio-communication system, this feature is useless in our case. This feature is thus not included on OBC2.

- **SD card socket**

As explained earlier, we will not be using an SD card for mass storage. This feature is thus not included on OBC2.

## 7.2 OBC2: Design of the custom board

The design of OBC2 was quite straightforward, thanks to its simplicity and to the small number of circuits needed. Here are, however, several points that did require some special attention, particularly for the choice of the components.

- **I/Os**

The corresponding I/Os of OBC1 and OBC2 are connected together. In the case of a malfunction, two connected pins could be configured as outputs and this could lead to a short circuit. To mitigate the consequences of such a situation, we added small 100  $\Omega$  resistors in series with each I/O pin of the OBC2's MSP430.

- **Crystals**

Two crystals are used. They were carefully chosen in our distributor's catalog, so that their temperature-induced frequency drift is as small as possible.

- **Power supply**

Whereas the FM430 needs a 5.0V power supply (the needed 3.3V being converted on-board), OBC2 was chosen to use directly a 3.3V power supply voltage. This was decided together with the EPS team, for reliability reasons (so that the different OBCs use different converters of the EPS). This 3.3V voltage is generated on the EPS board. Therefore, no regulator is needed on OBC2. However, the OBC2 still includes the same current-limiting switch as the one of the FM430 (a *Maxim MAX890*).

- **EEPROM**

We decided to use a small EEPROM as our mass storage memory. After reviewing the available products of several manufacturers, we initially selected the following candidate components (each with an I<sup>2</sup>C interface).

Size	Supply voltage	Operating current	Max I <sup>2</sup> C clock
<b>Atmel AT24C1024</b>			
1024 kbit	2.7 V to 5.5 V	Standby: 3 $\mu$ A Read (max): 2 mA Write (max): 5 mA	400 KHz
<b>Microchip 24LC1025</b>			
1024 kbit	2.5 V to 5.5 V	Standby: 100 nA Read (max): 450 $\mu$ A Write (max): 5 mA	400 KHz
<b>Renesas HN58W241000I</b>			
1024 kbit	2.5 V to 3.6 V	Standby: 1 $\mu$ A Read (max): 1 mA Write (max): 4 mA	1 MHz

The *Microchip 24LC1025* was finally chosen for its low power consumption and good availability.

- **Connectors**

The main interboard connections inside the satellite use the same connector as the one used on the FM430, namely the CubeSat bus (Fig. 7.2). It consists in fact in a pair of stackthrough connectors, available from *Samtec* under the reference *ESQ-126-39-G-D*.

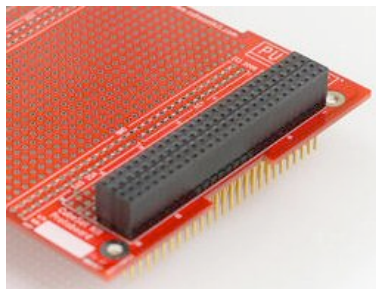


Figure 7.2: Connectors of the CubeSat bus (reference in text)

The JTAG connector is an 8-pin FPC (Flexible Printed Circuit) female connector. A reference is given in the electrical schematics, the *Hirose FH10A-8S-1SHB*, but it appeared that Hirose no longer manufactures the 8-pin version of this product. After some research, we found a similar component, manufactured by *Japan Solderless Terminals*, under the reference *08FMS-1.0SP-TF* (Fig. 7.3).



Figure 7.3: FPC connector used for the JTAG interface (reference in text).

After selecting all the components, we realized the electrical schematics of OBC2, using Altium Designer. They are given in Appendix E.

## 7.3 Interboard communication bus

The electrical connections between the different boards are done through a bus made of “stacktrough” connectors (Sect. 7.2). During the development of the different subsystems, I supervised the allocation of this bus. Although it took a lot of time, effort, and iterations to satisfy every team’s needs, there is little to say on this task, except to state the final decision reached, and make a few remarks.

Here are thus a few remarks on some details that did deserve some special attention.

- Some pins of the CubeSat bus are used on the FM430 for some very specific signals, that are of little use in our application (such as, for example, the signals related to the MHX radio interface). Unfortunately, most of these pins cannot serve for another use, since other signals could interfere with the hardware connected to these pins on the FM430. It is to be noted that, in our future satellites, where no FM430 will most likely be used, these pins would get freed-up and could be used to carry custom signals.
- Some power lines can carry large currents (e.g. the `POWER_XXX_BUS` lines), and each of these lines was thus distributed on two (adjacent) physical pins.
- Some analog signals have to transit through the bus to the radio beacon (BCN) board. These signals were grouped together at one end of the CubeSat bus. This minimizes the interference caused by digital signals carried on other pins of the bus.
- One pin is used to carry each of the I<sup>2</sup>C bus SCL and SDA signals. When routing PCBs, it is a common practice to route a ground track between these two signals. For the same reason, in the allocation of bus, we chose to place

a pin connected to the ground, in-between the pins of the two I<sup>2</sup>C signals. This also makes the routing of the PCB of the OBC2 slightly easier.

The resulting allocation of the bus is summarized in the table given in Appendix D.

---

## Chapter 8

# Hardware: Practical realization of the OBC electronic boards

This chapter presents the practical development of the real electronic boards. The first section presents a general plan for the prototyping and manufacturing stages, which was a common strategy for all the electrical subsystems of the CubeSat. The second section explains how this strategy applies to the particular case of the two on-board computers (OBCs).

### 8.1 General strategy for prototyping and fabrication

A general strategy for the prototyping and the manufacturing of the electrical subsystems was proposed at the beginning of the OUFTI-1 project. It consists in developing different prototypes at the various stages of the design process.

**The breadboard model** is the first one to be built. It may not be complete or fully functional, but it is the one that should serve most of the developments and of the experimental tests. Several iterations of this model are likely to be needed.

**The engineering model** is closer to the final version. Its electrical circuit and the components used should be the same as in the final model, but the board may not satisfy the flight size and weight constraints. Its main purpose is to be interfaced with all the other subsystems, so that a complete prototype of the satellite can be tested.

The **flight model** is the final version of the board. It is ready to be integrated in the CubeSat with the other subsystems. Several instances are likely to be manufactured, e.g. to serve vibration tests, to undergo additional tests when the satellite is on orbit, or for demonstration purposes.

## 8.2 Strategy for prototyping and fabrication applied to the OBC

### 8.2.1 Breadboard models

The CubeSat kit, that was purchased from Pumpkin, includes the FM430, which will be our OBC1, but it also includes a development board, or *devboard*, that served us as the breadboard model of both the OBC1 and the OBC2. This development board is shown in Fig. 8.1. It basically is an enlarged replica of the FM430, with some extra functionalities, such as an RS-232 interface, and connectors to allow powering it with a benchtop power supply. This devboard was very useful, since it allowed us to start experimenting and coding software very early, without having to wait for the fabrication of custom prototypes.

In addition to the devboard provided by Pumpkin, we built a small complementary *test board*. This board, represented in Fig. 8.2 and 8.3, plugs onto the CubeSat bus connector of the devboard, and adds additional testing capabilities to it. It includes several I<sup>2</sup>C devices (an AD7997 ADC and an LM75 temperature sensor), LEDs for visualizing the state of digital outputs and various connectors. These components were obviously not to be implemented in the subsequent prototypes of the OBC, but they allowed us to test various bits of software, such as, for example, the code to handle I<sup>2</sup>C devices.

### 8.2.2 Engineering models

The engineering model that was used for the OBC1 was, again, the development board provided in the CubeSat kit. As we said before, its design is — electrically — very close to the one of the flight module (FM430). Note that it includes the CubeSat bus connector that allows one to plug other boards, for interconnection tests.

For the OBC2, a first engineering model was designed as early as January 2009. I used the Altium Designer<sup>1</sup> software to manually route the PCB. I chose to route

---

<sup>1</sup>See <http://www.altium.com/products/altiumdesigner/>.



## 8. HARDWARE: PRACTICAL REALIZATION OF THE OBC ELECTRONIC BOARDS

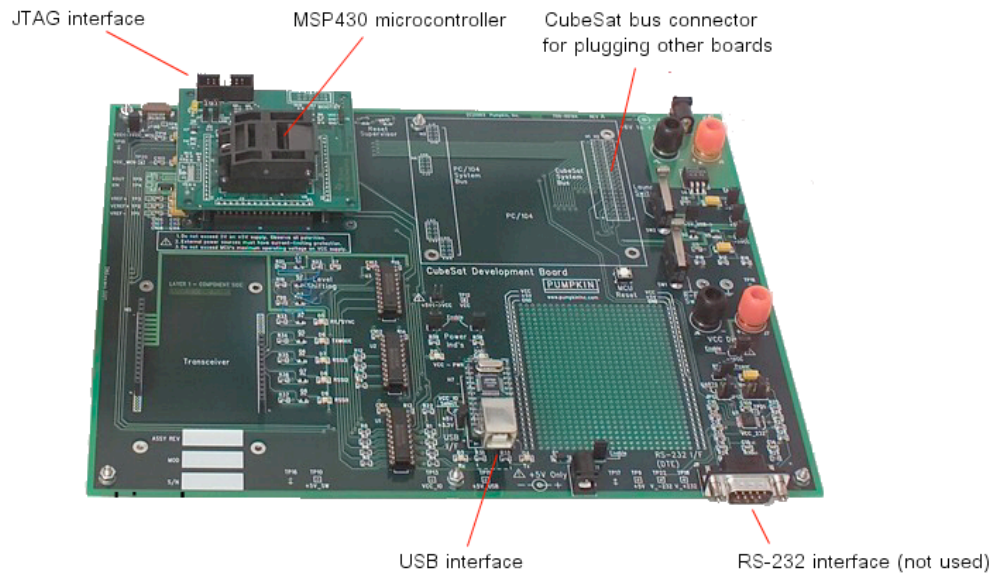


Figure 8.1: Development board provided with the FM430.

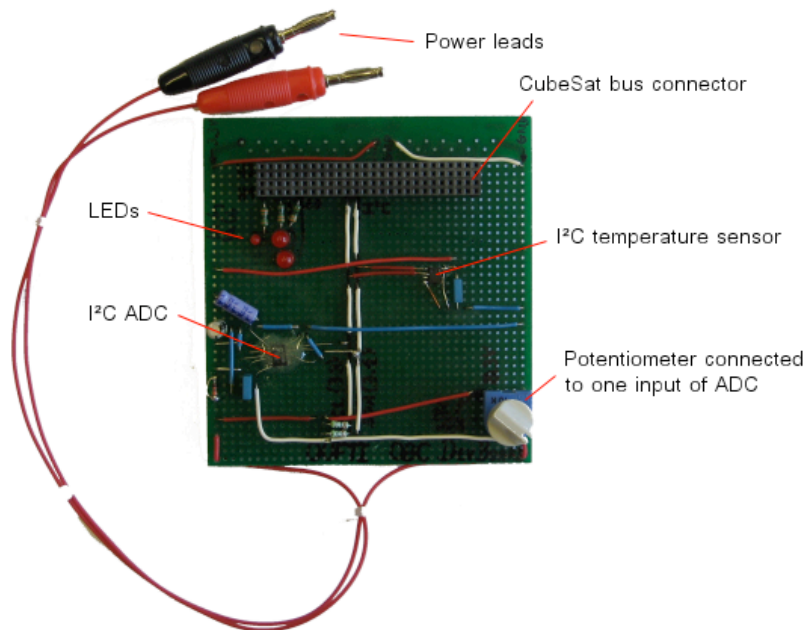


Figure 8.2: Custom test board.

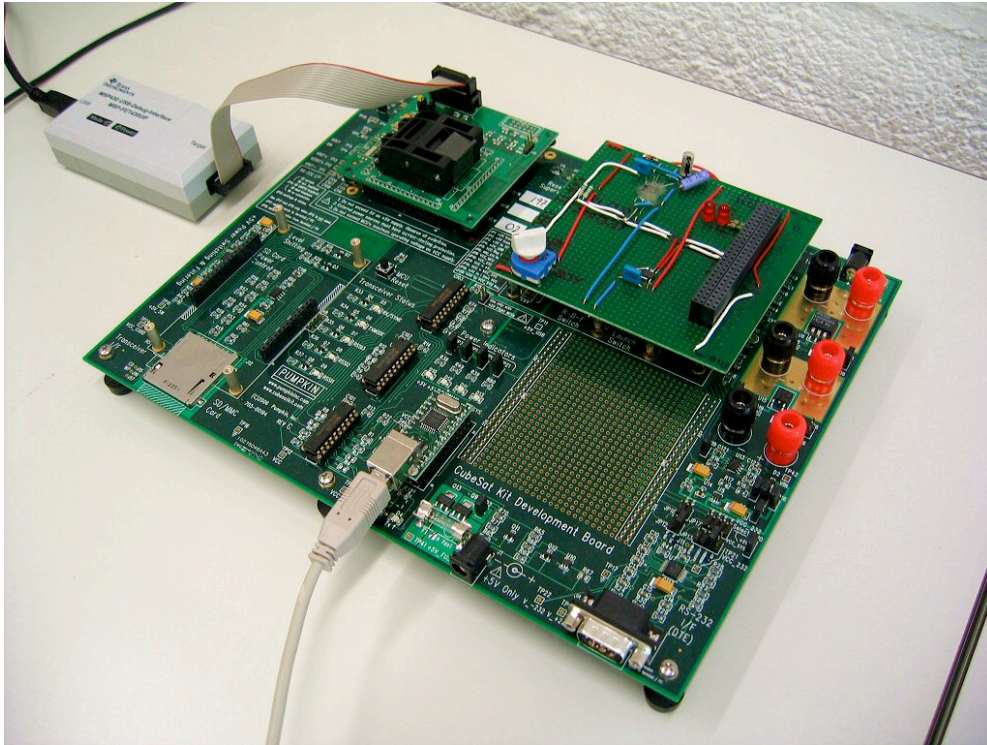


Figure 8.3: Custom test board, plugged onto the development board.

it on only two layers, to make prototyping faster and cheaper. However, with only two layers, routing the 52 I/O signals between the microcontroller and the bus connector was particularly tricky. Some connections were left to be connected with loose wires, which was not a problem, since these connections were not likely to be needed in most of the interconnection tests that this prototype was devised for. Appendix G gives an overview of the design of this PCB, which was manufactured by Olimex (<http://www.olimex.com>), a fast PCB prototyping company I have good experience with. Figure 8.4 shows the prototype at different stages of assembly. Note that I hand soldered all the components of the prototype.

Two months after having designed the first engineering model, it was decided, for the whole project, to get all the electronic boards fabricated and assembled at a local company, Deltatec<sup>1</sup>. This company has experience in the design of hardware for aerospace applications, and their expertise was judged to be a plus for the project.

More precisely, four reasons justifies this strategy. First, we benefit from the experience of Deltatec in the routing and in the assembly of electronic boards, including for space applications. Second, this strategy permits to accelerate the general realization process, and it allowed other teams to test their hardware before completing the write-up of their thesis — although this was not necessary in my

---

<sup>1</sup>Deltatec is located in Ans, Belgium; see <http://www.deltatec.be>.

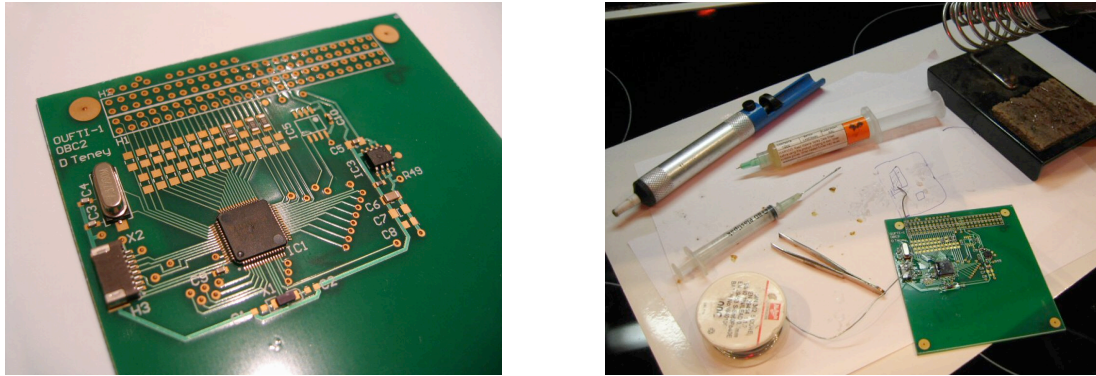


Figure 8.4: First engineering model during assembly.

case, since a homemade prototype was already built before the one of Deltatec. Third, this strategy guarantees a uniformity in the realization and in the quality of the various electronic boards of the satellite. Fourth, it provided experience for the students in interacting with a commercial company, which forced us to prepare a thorough set of specifications, and to take responsibility for the various choices made in the designs.

We provided Deltatec with the schematics of the OBC2, which they re-encoded in their standard electronic design software. The re-encoded schematics can be found in Appendix F. The PBC was then re-routed, on four layers this time. Shorter track widths and clearances were also used, but it is interesting to note that the overall area used on the PCB is roughly the same on the two engineering models. It is also worth noting that the expertise of Deltatec for the routing was particularly useful. They took much more care of the specifics of space equipment during the design of the PCB, such as thermal issues or potential electrical perturbation problems. For example, special pads under the integrated circuits were included, to carry away the heat they can generate. Another common practice in space electronics is to never let floating input or output pins on any integrated circuit. This practice eliminates potential entry points for electrical perturbations (due to radiations for example). The problem of the mechanical resistance to vibrations was addressed by providing each components with pads slightly larger than usual. This allows a larger amount of solder to hold the components, which provides better mechanical fixation points. Figures 8.6 and 8.5 show the engineering model provided by Deltatec.

### 8.2.3 Flight models

The flight model of the OBC1 is the FM430 itself (Fig. 8.7). The flight model of the OBC2 has not been built yet, but it will be provided by Deltatec as well.

## 8. HARDWARE: PRACTICAL REALIZATION OF THE OBC ELECTRONIC BOARDS

---

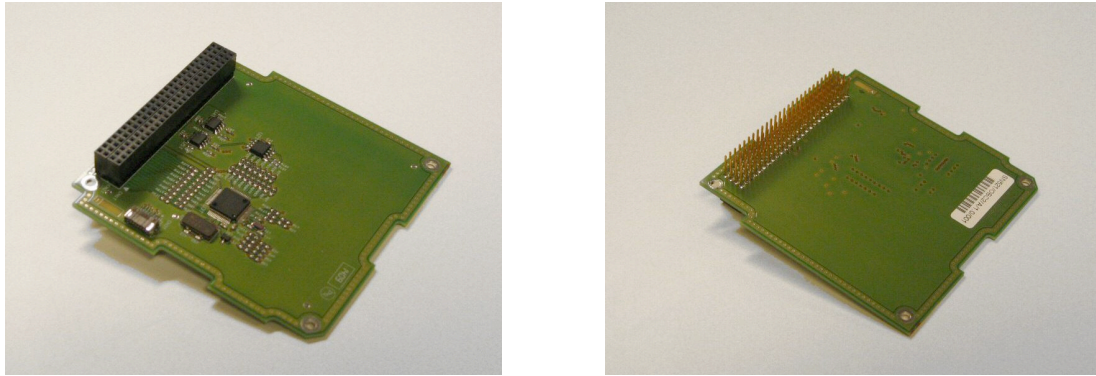


Figure 8.5: Engineering model provided by Deltatec.

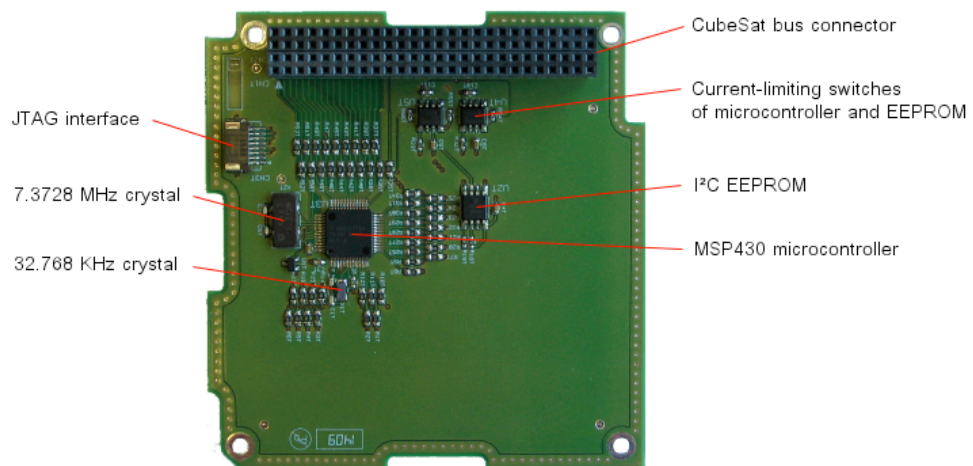


Figure 8.6: Annotated view of the engineering model provided by Deltatec.



## 8. HARDWARE: PRACTICAL REALIZATION OF THE OBC ELECTRONIC BOARDS

---

Since the functional tests of the OBC2 engineering model did not show any particular problem, no major changes are expected to be required for the flight model. Soldering should however be done differently, with a high lead content alloy. This will offer better resistance to the vibrations of the launch, to the large temperature variations encountered in space, and will avoid the accidental growth of *tin whiskers* (see [Drevon05]).

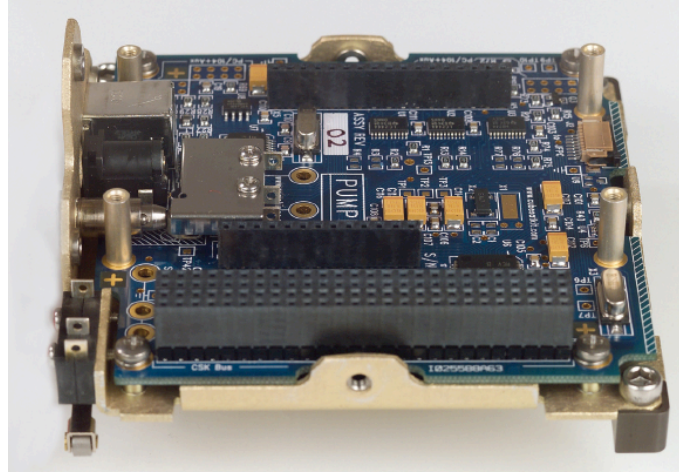


Figure 8.7: Pumpkin's FM430, attached to the bottom panel of the mechanical structure of the CubeSat.

---

## Chapter 9

# Software: Requirements and organization

In this chapter, we introduce the on-board software of OUFTI-1. We first identify the required fonctionnalités of the software, and we then present a partitioning of the software into modules, and justify how these modules can take care of all the required fonctionnalités of the OUFTI-1 system.

### 9.1 Identification of software fonctionnalités

Before designing the architecture of the software of a significant project, one must first determine the exact requirements and the desired behavior of the finished product. In our case, this task was not easy. Indeed, because of the general planning of the project, we had to start thinking the development of the software even though a significant number of unknowns did remain, including at the very top level of the mission requirements and mission implementation<sup>1</sup>. We therefore had to formalize these requirement in a quite abstract way, that could allow a maximum number of variations of these unknown parts. These requirements, or software fonctionnalités, are listed below. Note that they are similar, but not identical, to the general “OBC roles” of Sect. 4.9.

1. Perform the initial satellite operations (antenna deployment, first activation of the other subsystems) according to a predefined sequence.
2. Perform AX.25 and D-STAR encoding and decoding.

---

<sup>1</sup>At the time of this writing, several unknowns remain, mainly in the overall hardware architecture, the strategy for making D-STAR work in space, and the number and location of processors throughout the satellite.

3. Handle telecommands received on the uplink channel.
4. Perform measurements of housekeeping and science parameters aboard the satellite.
5. Store relevant measurements until they can be sent to the ground station.
6. Respond to telemetry requests by sending present or past (stored) measurements.
7. Provide a time reference.
8. Perform power supply management, by enabling and disabling other subsystems in predefined conditions (e.g. a low battery voltage).
9. Perform power cycling in case of latchup in a subsystem (detected with the FAULT\_XXX signals).
10. Manage the experimental electrical power supply (EPS2), by enabling and disabling it in predefined conditions.
11. Manage the D-STAR system, by configuring it (e.g. for Doppler compensation) according to data received via specific telecommands.
12. Keep a log of meaningful events happening aboard the satellite, that can then be retrieved with one or more specific telecommands.
13. Monitor, for the backup processor (OBC1), the activity of the default processor (OBC2) and detect when it stops functioning.

## 9.2 Organization of software into layers and modules

### 9.2.1 Traditional organization of satellite on-board software

Any large or complex software project has to be subdivided into subproblems in order to be implemented efficiently. In the traditional space industry, the on-board software of a satellite system is typically divided relatively to two dimensions (Fig. 9.1, and [Parisis08a, Parisis08b]).

On the one hand, horizontal divisions separate parts of the software relatively to their dependence on hardware. The top layer is strictly mission dependent, and it handles the high-level tasks of the system (electrical power management, thermal management, payload activation, etc.). The bottom layer interfaces directly with the hardware. The intermediate layers provide so-called *common services*, i.e. the services that do not directly depend on the particularities of the mission and of the hardware. These services can include, for example, a file system for data storage, or the basic handling of telecommunications. This separation permits to easily reuse a maximum of the developments from one project to another. For example, the common services are generally necessary to any mission and can then be reused.

The hardware drivers can also very often be reused, as long as the platform of the satellite (all but the mission-dependent subsystems of the satellite) remains essentially the same.

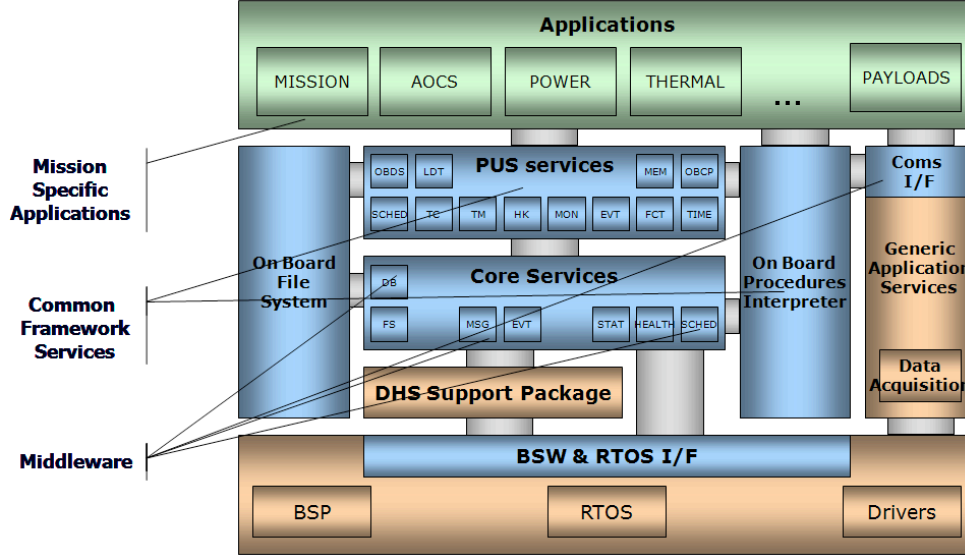


Figure 9.1: Typical organization of the on-board software of a traditional satellite (source: [Paris08a]).

On the other hand, vertical divisions separate parts of the software relatively to the functionalities of the system. This is particularly apparent for the top layer, where, for example, the payload management module is well separated from, say, the power management module. This vertical division allows one to clearly organize the code, and permits easier reuse and independent development of different parts of the software.

### 9.2.2 Organization of on-board software of OUFTI-1

The ideal subdivision scheme that we just presented is conceptually very interesting, but it is not well suited to the OUFTI-1 project. The software that we have to implement is much simpler than the software of a traditional (commercial or defense) satellite. An “overdivision” of the software could make it unnecessarily cumbersome, which would be a problem with the limited memory and processing power available.

For the horizontal division, we still identified a useful partitioning for our software, into two layers (Fig. 9.2). The bottom one, which we can call the *device driver* layer, provides drivers for the physical devices we have to interact with (e.g. the external ADCs, some internal modules of the microcontroller). These



drivers consists in functions that can be called from another layer to interact with these devices. The top one, which we can call the *application layer*, includes all the main functionalities of the system. This application layer is itself vertically divided into different modules. The modules that we defined cannot be deduced directly from the list of functionalities identified in Sect. 9.1, but they allow one to handle all of these functionalities in a very efficient way, while still being very modular. Fig. 9.2 shows which functionalities are handled by each module (the numbering refers to that of Sect. 9.1). Below, we give a basic description of each of these modules.

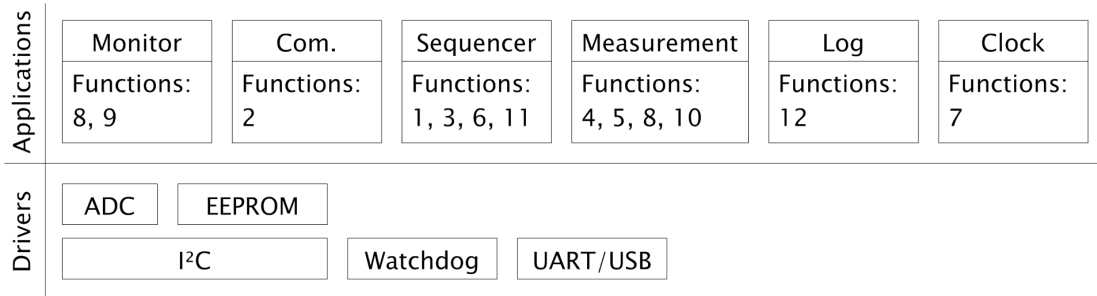


Figure 9.2: Organization of the on-board software of OUFTI-1.

**The monitor module** is responsible for handling the major *status* changes aboard the satellite, i.e. the decision of enabling or disabling the various subsystems. These decisions are taken on the basis of some requests from the other modules (themselves based, e.g. on the measurement of on-board parameters), as well as on the status of the current-limiting switches of the power supply, sensed directly by the monitor module (via the FAULT\_xxx signals, see Sect. 4.2).

**The communication module** encompasses all the functions related to the handling of D-STAR and AX.25 encoding/decoding. It outputs the telecommands that may have been decoded, and takes as input the parameters of the transmission (e.g. the Doppler-effect frequency correction). It can also be fed with telemetry, to be sent on the downlink channel.

**The sequencer module** is a generic module that handles all the telecommands coming out of the communication module, as well as other commands that have been generated on-board by other modules. All these (tele)commands are accompanied by their planned execution time. The role of the sequencer is to execute these commands at the specified time. The (tele)commands that need immediate execution are accompanied by a special tag, and are handled first by the sequencer.

**The measurement module** takes care of performing the measurements of the two kinds of parameters already defined (Sect. 4.10). The housekeeping

parameters, sampled at a relatively high frequency, are processed in real time, in order to request changes of status (i.e. to request to enable or disable specific subsystems) when needed. The requests for change of status are then handled by the monitor module. The science parameters, that are not used in the control loop of the operation of the satellite, are sampled at a lower frequency, and then immediately written into non-volatile memory (the external EEPROM). These measurements can be sent back to Earth later upon request, with a telecommand (handled by the sequencer module).

**The log module** is responsible for storing a list of meaningful events happening aboard the satellite. Any other module can post an event to the log module, that will take care of storing it in the external EEPROM. The log module also provides an interface to access the stored list of recent events; it can be accessed to be sent down to Earth, upon request of a telecommand (handled by the sequencer module).

**The clock module** maintains a time reference, with a resolution of one second. This resolution is appropriate for the needs of the mission. The current value of the clock can be accessed by any other module.

Details and explanations on how each task is handled by these modules are given in Appendix I.

---

# Chapter 10

## Software: Architecture

This chapter presents the architecture of the on-board software of OUFTI-1. We first discuss task scheduling, and we then present the actual architecture proposed for the OUFTI-1 software.

### 10.1 Scheduling of software fonctionnalités

#### 10.1.1 Methods of scheduling

The on-board software of OUFTI-1 presents the characteristics of a real-time system, because of the fact that it has to handle various external events and perform actions with timing constraints. This section presents three alternatives to organize such software, so that all of its fonctionnalités are executed with respect to their timing constraints.

##### **Cyclic executive**

The cyclic executive is the simplest form for implementing real-time software. It consists of one infinite loop in which all the tasks of the software are executed. Delays are typically introduced in the loop to ensure a constant periodic execution. There is thus one main control flow, which can however be completed by the handling of interrupts. This simple organization gives the advantage of being simple to analyze, and limits the needs of synchronization or message-passing mechanisms. The limitations of such a system are however obvious: the rigid structure does not allow one to easily execute functions at different rates, and it does not allow either to assign different priorities to different fonctionnalités of the software. This last feature would be however desirable in our case. For example, telecommunication

functions are far more time-critical than logging functions; indeed, they have to be executed in a timely manner with respect to the baud rate of the transmission, whereas the writing in the log of an event tolerates a delayed execution. This first solution is thus not suitable for our system.

### **Cooperative multitasking**

Multitasking is the method that permits to execute multiple tasks on a single processor, using mechanisms provided by an operating system (OS). Its simplest form is the cooperative multitasking, in which each task has the responsibility of initiating the *context switch* necessary to allow another task to execute. This permits the OS to be relatively simple, and gives the programmer the advantage of choosing where the execution of each task can be interrupted. The drawback is that a faulty task (due to programming errors, hardware failure, code corruption, etc.) can cause the whole system to hang. This could lead to serious problems in a system like ours, and this solution was therefore discarded.

### **Preemptive multitasking**

In a preemptive multitasking environment, it is the responsibility of the OS to decide when to perform a context switch. A task may thus be interrupted at any time. Each task is usually assigned a priority, in order to ensure that the most critical tasks are given a greater share of available processing time.

## **10.1.2 Choice of operating system for the on-board software of OUFTI-1**

After determining that the best suited type of OS for our project was a preemptive multitasking OS, we looked for one such OS that was available, off-the-shelf, and ideally with an architecture port for the MSP430. We found that FreeRTOS<sup>1</sup> presented these characteristics. It is an open-source, lightweight real-time operating system, freely available, and known to be reliable<sup>2</sup>. We examined its exact requirements, especially its memory footprint in ROM and RAM (see [FreeRTOS]), and we concluded that it was ideally suited for our project.

Let us also note that FreeRTOS provides an API for using task synchronization mechanisms, including mutexes and semaphores, as well as the means needed to implement periodic tasks.

---

<sup>1</sup>See <http://www.freertos.org>.

<sup>2</sup>A sister project of FreeRTOS, SafeRTOS, is based on similar components, and is certified for use in safety-critical applications.

### 10.1.3 Identification of tasks in the on-board software of OUFTI-1

Having determined that we will use an OS, we must then identify the tasks to be handled by this OS. We decided to allocate one task to the monitor module, one to the sequencer module, one to the measurement module, and one to the log module. Let us note that, first, the case of the clock module is discussed Sect. 10.2.7. Second, we excluded the communication module from most of the following discussion, due to the uncertainties still related to the hardware implementation of the COM subsystem (details are available in [Henrard09, Mahy09]).

It is generally desirable to limit as much as possible the number of tasks, for simplicity reasons, already mentionned in Sect. 10.1.1, but also to limit the memory and processing ressources needed by the scheduler to handle these tasks. We analyzed all the possibilites of grouping two or more modules of our software in a single task, but none of these was deemed acceptable.

As an example, let us discuss the potential grouping of the monitor and measurement modules. This may look interesting at first sight: the measurement module samples the housekeeping parameters, that are analyzed to generate a `requestedStatus` shared data structure, which is then handled by the monitor module. Grouping these two modules thus also allows one to eliminate this shared data structure. However, the measurement module also has to sample the science parameters, at frequencies other than for the housekeeping parameters. The relatively high frequency of execution of the monitor module does not allow one to perform these measurements in the monitor task, and another task would therefore still be needed. We thus concluded that it was simpler to keep the conceptually simple solution of one separate task for each of the monitor and measurement modules.

The analysis of other possibilities for grouping several tasks together lead to the same type of conclusions, due to the differences in the frequencies of execution of these tasks, or to the differences in the criticality of the fonctionnalités they handle.

## 10.2 Architecture of software modules

This section presents the fonctionning details of each software module. It is advisable to read the following discussion while keeping an eye on the global static structure diagram, in Appendix J.

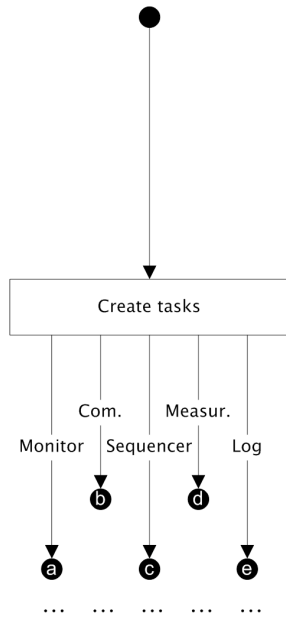


Figure 10.1: Flowchart of the initialization of the software for the default processor (OBC2).

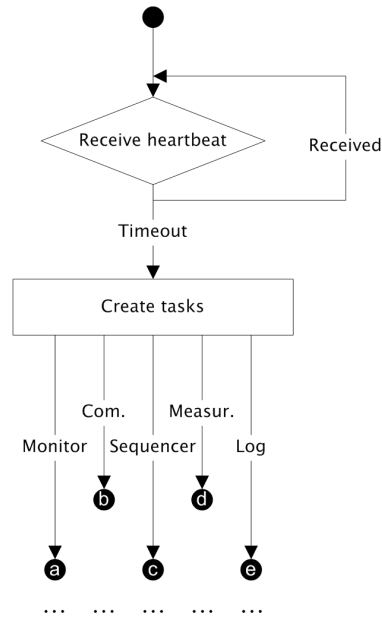


Figure 10.2: Flowchart of the initialization of the software for the backup processor (OBC1).

### 10.2.1 Initialization of software

The initialization part of the software corresponds to the work executed prior to the creation of the tasks and prior to the execution of the scheduler of the OS. This is one of the two main places where the code of the backup processor (OBC1, the FM430) and the code of the default processor (OBC2, the custom board) have to be different. The default processor does not require to do anything particular (Fig. 10.1), but this is where the backup processor monitors the *heartbeat* signal of the default processor (Fig. 10.2). This signal is chosen to be a simple message on the existing I<sup>2</sup>C bus; the complexity of this mechanism seemed perfectly suited to the problem, and, moreover, it does not require to allocate particular I/O lines to this function. This solution furthermore allows us to keep the backup processor in one of its low-power modes most of the time, by using its internal hardware devices to perform most of the monitoring work: the hardware I<sup>2</sup>C circuits integrated into the MSP430 are responsible for receiving the message, and the internal watchdog timer of the MSP430 is responsible for starting the control of operations if no heartbeat message is received during a long time. This allows the backup processor to be active for only a few dozens of cycles per second, keeping its power consumption extremely low.

Note that this is the only part of the software where the sequence of operations is really different between the two processors. The distinction between them will therefore not be made in the rest of the discussion.

### 10.2.2 Architecture of the monitor module

The monitor module is implemented in one specific task. This task mainly consists in a large infinite loop, executed with a constant period, and that performs four jobs (Fig. 10.3).

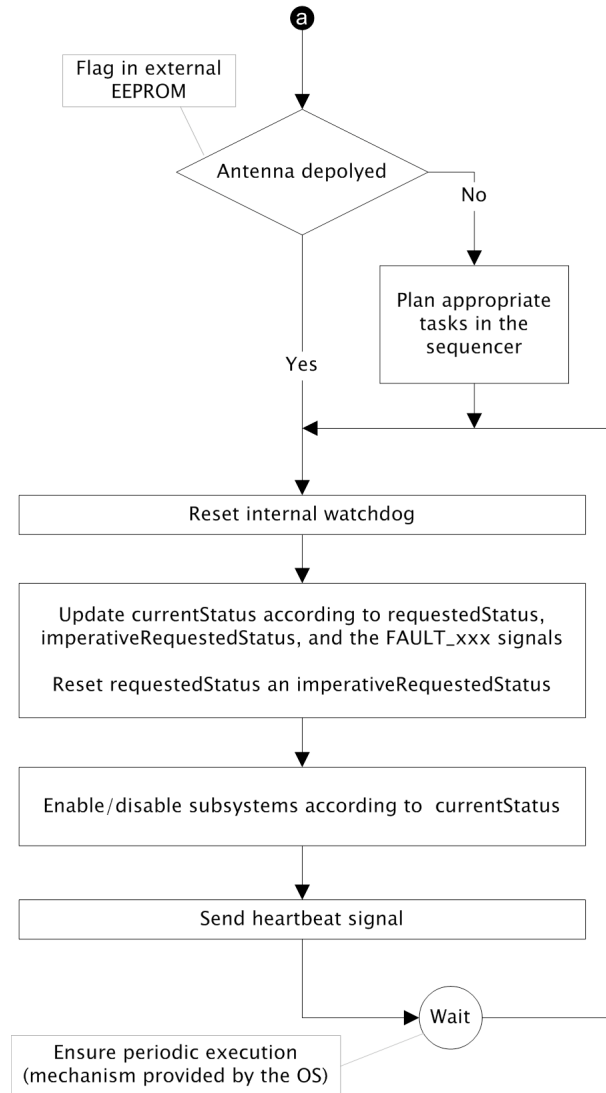


Figure 10.3: Flowchart of the task of the monitor module.

#### Job 1

The internal watchdog timer of the processor is reset. This watchdog is used to recover from fault conditions, such as a hang, caused for example by a single-event upset. The reset of this watchdog timer is performed first, so that the period of execution of this job is not affected by the variations of execution time of the other jobs in the monitor task.

## Job 2

The monitor task updates, in live memory, the status of the subsystems of the satellite. Each subsystem that can be enabled or disabled by the OBC (thereby excluding e.g. the BCN, which is active at all times) has information on its status stored in the live memory of the OBC, in a data structure named **currentStatus**. This status of each subsystem can be any value in the set {on, ready, disabled}. Below follows the definition of each of these values.

- **on** means that the corresponding subsystem is currently powered on.
- **ready** means that the corresponding subsystem is currently not powered on, but is considered in working order. The OBC may automatically turn it on at any time.
- **disabled** means that the corresponding subsystem is temporarily or permanently disabled. Only a telecommand can change this status to “ready” or “on”.

The monitor task updates the current status of the controllable subsystems according to the following information.

- **requestedStatus** This data structure is similar to **currentStatus**, and is modified by the measurement module. A special value, named “noChange”, is possible in addition to “on”, “ready”, and “disabled”; it is used to depict no particular request in the change of status of a subsystem. Note that no information contained in **requestedStatus** may change the status of a subsystem currently set to disabled. This prevents an automatic reactivation of a disabled and perhaps defective subsystem.
- **imperativeRequestedStatus** This table is similar to **requestedStatus**, except that its handling by the monitor module allows to change the status of a disabled subsystem. This permits a telecommand to reactivate a disabled subsystem.
- **FAULT\_xxx lines** These electric signals come from the MAX890 current-limiting switches that equip each subsystem (Chap. 7). If one of these signals is set, the monitor module sets temporarily the status (in **currentStatus**) of the corresponding subsystem to disabled, and attempts to reactivate it, several iterations later. This power cycle is designed to recover from latchups. However, the optimal reaction to such an event has not yet been determined. Additional details are available in [Thirion09].

## Job 3

Once the current status of the subsystems has been updated in memory, the actual activation or deactivation of these subsystems is performed. This is realized by setting or resetting the EN\_xxx lines of the current-limiting switches.



#### Job 4

In the case of the default processor, the heartbeat signal is sent to the other processor. This is the last job to be done, since it is the one that has the potential of creating the largest jitter (due to the use of the — shared — I<sup>2</sup>C module).

Note that, prior to the execution of any of the four aforementioned jobs, the monitor module is responsible for adding to the scheduler the commands related to the initial activation of the satellite. The exact sequence of commands has not yet been decided, but here follows an example that indefinitely triggers the antenna deployment mechanism until success, then turns on the COM subsystem; each planned command is defined by a pair {execution time ; command}.

- {5 min ; Deploy antenna}
- {6 min ; Check whether antenna is correctly deployed}

If the antenna is not correctly deployed, the following tasks are added:

{currentTime + 1 min ; Deploy antenna}

{currentTime + 2 min ; Check whether antenna is correctly deployed}

If the antenna is correctly deployed, a specific flag is set in the external EEPROM, and the following task is added:

{currentTime + 5 min ; set the COM subsystem in `requestedStatus` to “on”}

### 10.2.3 Architecture of communication module

Due to the significant, remaining uncertainties in the hardware of the COM subsystem of the satellite, no precise or fixed choices can be made for the architecture of the related software. A tentative implementation is however discussed in [Henrard09, Mahy09]. What is of more interest to us at this point is the interface of the communication module with the other modules.

As inputs, the communication module needs the values of the parameters needed to configure the radiocommunication channels. The communication module must therefore provide relevant functions to set each of these parameters. This should include, for example, a function to set the current frequency shift for the correction of the Doppler effect. The communication module should also accept a way to feed data on the downlink AX.25 channel, for sending telemetry down to earth. This mechanism has not yet been devised; details are available in [Henrard09, Mahy09].

As outputs, the communication module provides the telecommands that have been received on the AX.25 uplink and decoded. For each received telecommand, the communication module calls a function of the sequencer, that places that particular

telecommand in the list of pending commands, that will later be executed by the sequencer.

### 10.2.4 Architecture of sequencer module

The sequencer module is implemented in one specific task (Fig. 10.4). The role of this task is to execute, at the right time, the commands stored in a shared data structure, named `pendingCommands` (see Appendix J). This data structure can be filled via calls to a specific function, `addCommand()`, provided by the sequencer module. This function takes care of filling the data structure properly. The mutual exclusion of concurrent calls to `addCommand()` is assured by disabling the interrupts during the manipulation of the `pendingCommands` shared data structure; this was preferred over the use of a software solution such as a mutex, since it does not use any memory resources, and since the number of operations in the critical section is very small. The synchronization between the adding of commands, via calls to `addCommand()`, and the use of these commands by the sequencer, is performed using a semaphore. Note that an implementation of semaphores is provided with FreeRTOS.

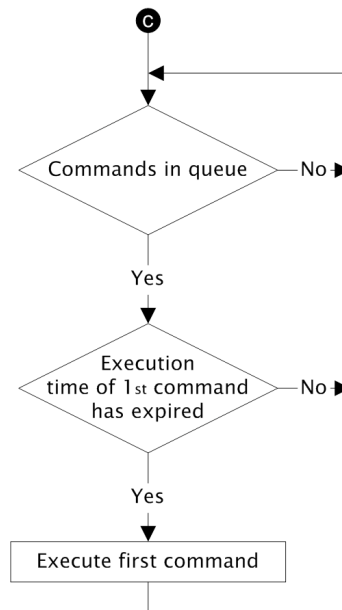


Figure 10.4: Flowchart of the task of the sequencer module.

The data structure used to store the pending commands is chosen to be a fixed-size array. This solution is preferable to a dynamic allocation scheme, and is strongly recommended in a system like ours ([AST06]), since it makes the analysis of the whole software easier, by avoiding a source of uncertainties (the possible failure of memory allocation). The handling in the case of overflow of `pendingCommands` has not been decided yet.

The complete list of actions that can be executed by the sequencer has not yet been established. This list should include as many ways as possible to act on the satellite, since these commands will be the only way for the operator on the ground to control it. We also advise to include generic commands in this list, e.g. to read or write any address in RAM or EEPROM. Such commands should accomodate as many unforeseen situations as possible. Note that some particular commands may need a parameter to specify by which processor (OBC1 or OBC2) they are to be executed. An example of such a processor-specific command would force the default processor (OBC2) to stay idle, so that the backup processor (OBC1) could take over the control of operations.

### 10.2.5 Architecture of measurement module

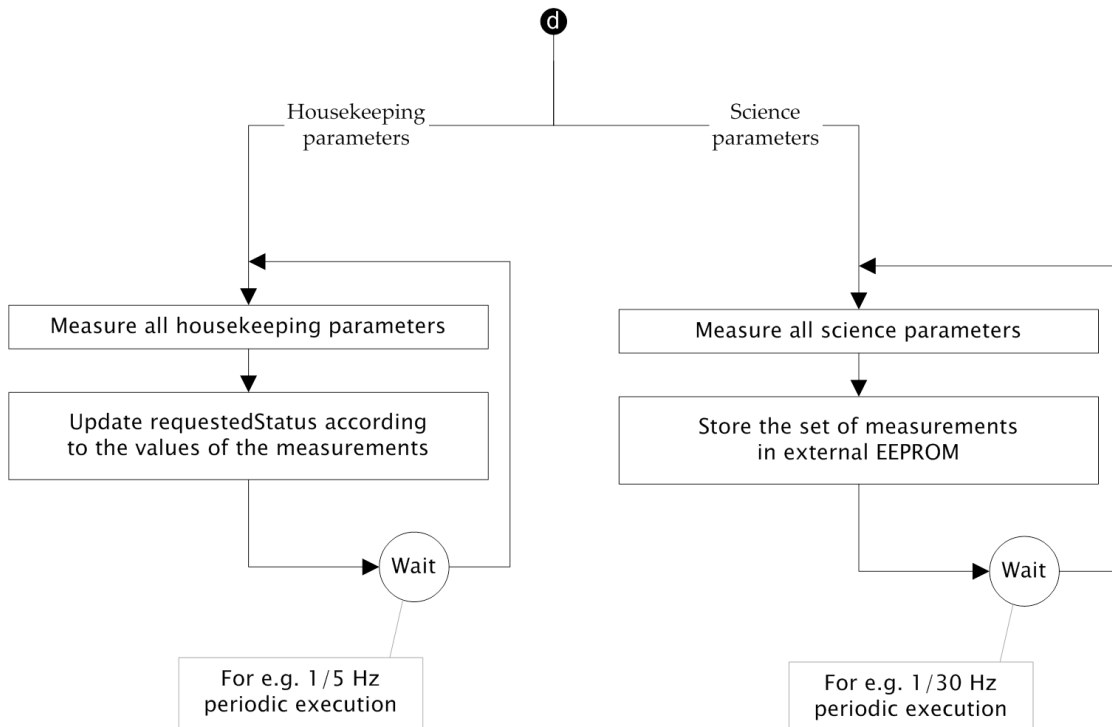


Figure 10.5: Flowchart of the task of the measurement module.

The role of the measurement module is to handle both the housekeeping parameters and the science parameters. Even though all these measurements have to be sampled at different frequencies (Fig. 10.5), this module can be implemented as a single task.

The housekeeping parameters, once sampled, are analyzed by a specific function, that can modify the `requestedStatus` variable (Sect. 10.2.2). For example, the voltage of the batteries, once sampled, is compared to a predefined threshold, and the result of this comparison is used to turn EPS2 on or off, by setting its

value in `requestedStatus` to “on” or “ready”, respectively. Note that the analysis function, which consists only in a comparison here, may be more complex; it could, for example, include an hysteresis in the comparison with a threshold.

The science parameters, once sampled, are stored in the external EEPROM. We use a fixed scheme for the placement in this memory (for each parameter, a fixed range of addresses is used cyclically). Details are available in [Evrard09].

### 10.2.6 Architecture of log module

The implementation of the log module is similar to the one of the sequencer module. Once specific task is used; it takes data from a shared data structure, `pendingEvents`, and it writes it in the external EEPROM. The same synchronization mechanisms — disabling the interrupts for mutual exclusion and a semaphore for synchronization — are used.

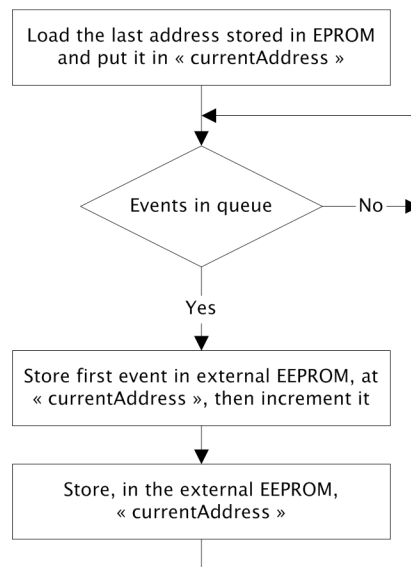


Figure 10.6: Flowchart of the task of the log module.

A static memory placement scheme is also used here. A predetermined range of addresses in the external EEPROM is used cyclically to store the most recent events. Note that the log module also store in the external EEPROM the last position used in this range. This is designed to keep a coherent log even when the backup processor (OBC1) takes over the default processor (OBC2). The events written in the log also mention by which processor they were generated.

## 10.2.7 Architecture of clock module

As a reminder, the only goal of the clock module is to maintain a time reference with a precision of one second, and make it available for any other module that needs it. This naturally translates into a global counter variable, which has to be incremented every second.

The clock module is chosen to be implemented, not with a specific task, but directly in the interrupt service routine (ISR) of a timer of the MSP430. Timer B is used, and its clock uses the external 32.768 KHz crystal. This gives, with the appropriate configuration of this timer, a 1 Hz periodic execution of the ISR. The code executed in this ISR only consists in incrementing a counter, and thus only takes very few cycles (Fig. 10.7).

The counter variable, named `currentTime`, is implemented on 32 bits. This gives a counting capability of  $2^{32}$  seconds, or about 136 years. Since the instructions of the MSP430 can manipulate only 16 bits at a time, operations on `currentTime` may not be atomic. Therefore, when manipulating this variable, we chose to temporarily disable the interrupts, to ensure atomicity of the operations.

Note that the `currentTime` variable can not only be read, be also written. This allows a telecommand to initialize or to resynchronize the on-board clock.

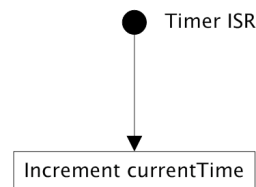


Figure 10.7: Flowchart of the clock module.

Instead of using a hardware timer, the clock module could be implemented using an specific task of the OS. However, this would require more ressources (as does adding any additional task), both in memory and in the processing time of the scheduler. This is thus not a desirable solution.

The clock module could also be quite simply integrated in an existing, periodic, task. However, a very careful — and complicated — analysis of the scheduling of all the tasks would be needed, to show that the chosen task would be schedulable at all times, even in cases of software faults (due to programming errors, hardware failure, code corruption, etc.). If this is not proven, the clock could experience jitter or drift, which is to be avoided, since the clock is the reference for many critical operations aboard the satellite.

The use of a timer and its interrupt routine for the clock module was therefore

deemed the best solution given the information that was available regarding the mission and the subsystems at the time of this writing. Let us however note that this regular interrupt will have to be taken into account when designing and analyzing the execution of the communication module (given its strong real-time constraints).

### 10.2.8 Architecture of device drivers

The device drivers are libraries of functions that allow one to interact, at a high level, with specific hardware devices. For example, a function is provided to reset the internal watchdog timer of the MSP430, and another one is provided to perform a conversion on an external ADC and retrieve the conversion result. Even though the implementation of these functions is relatively straightforward, the device drivers are the part of the software that has needed the most development time. Indeed, interacting with each device requires a good understanding of its functioning, with a thorough analysis of its datasheet.

There are few details of this part of the software that can be analyzed at a high level. However, all the low-level details are explained directly in the source code (Appendix K).

Note that the I<sup>2</sup>C drivers make full use of the corresponding hardware circuits integrated into the MSP430. A software buffer is used, and the interrupts generated by the I<sup>2</sup>C module are used to move data from this software buffer to the hardware module in transmit mode, and vice versa in receive mode. A semaphore is used to report to the calling function when the transfer is complete. Appropriate verifications and timeouts are included to make the software robust to hardware faults (e.g. a device that does not respond).

Note that the UART drivers make also full use of the corresponding hardware circuits integrated into the MSP430. Similarly, a software buffer is used, and the interrupts generated by the UART module allow a transfer to be executed autonomously once the appropriate function has been called.

---

# Chapter 11

## Software: Practical implementation

This chapter describes the software that was implemented. Due to the many uncertainties remaining, at the time of this writing, in several aspects of the mission and in the design of several subsystems, the implemented software is necessarily a first-cut of what the complete software will ultimately be. However, this software implements all the functionalities and modules discussed in Chap. 10. The implemented software was designed to constitute a solid basis, from which the final CubeSat software will ultimately emerge.

### 11.1 Details of software implementation

As discussed in the previous chapter, the communication module could not be developed at this point. Since this module has a central role in the overall software system, one had to simulate it in some way, in order to provide a functional and testable implementation of the system. We chose to use the existing USB interface of the FM430 (Chap. 7) to simulate the input/output capabilities of the COM subsystem. Here is some information, module by module, about the functionalities of the implemented software.

- **Initialization**

The dual-processor redundancy is implemented, as explained in Chap. 10, using I<sup>2</sup>C messages.

- **Monitor module**

All the functionalities of the monitor modules are implemented. However, only one subsystem, EPS2, is currently handled. This is sufficient to test all the functionalities, and the effort needed to handle more subsystems

is very small. In the case of a `FAULT_EPS2` signal, the subsystem is reset via a power cycling. Note that, for simulation and test purposes, the `FAULT_EPS2` signal can be set by a “telecommand” sent to the communication module (through the USB interface). Again, for simulation and test purposes, the `EN_EPS2` is connected, on the test board, to a LED, that permits to directly visualize the activation and deactivation of the EPS2 by the OBC. The value of the EPS2 in `currentStatus` is, by default, set to “disabled”.

Note that the monitor task, in this first implementation, also makes a LED blink on the test board. This LED is different in the case of OBC1 or OBC2, which allows one to clearly visualize which OBC is active at any time.

- **Communication module**

We use the USB interface, which itself uses the UART1 of the MSP430, to receive telecommands, and to send telemetry and status messages. The code for handling the USB interface is integrated into the communication module, so that the transition from using the USB interface to using the actual COM subsystem will be fairly transparent to the other software modules.

A “telecommand” is sent to the OBC by transmitting 4 bytes on the USB interface. The first byte is a start byte, of constant value 0x21 (“!” in ASCII), that indicates to the OBC that the transmission of a telecommand is following. The two following bytes correspond to the planned execution time of the telecommand (in the same format and with the same reference as the on-board clock); if the telecommand needs immediate execution, this value should be set to zero. The last byte, finally, is the identifier of the telecommand itself. Note that, in the final software, some particular telecommands should probably accept parameters, of fixed or variable size; this feature was however not implemented in this first version of the software, to keep it simple and thus easily modifiable.

- **Sequencer module**

The sequencer is implemented as discussed in Chap. 10. The only difference is in the number of commands currently supported. Only commands needed for basic test and demonstration purposes are implemented. Note however that adding a new command is generally a matter of only a few lines of code. The command for deploying the antenna is currently simulated by sending a “Deploy antenna” message on the USB interface.

- **Measurement module**

The measurement module is implemented as discussed in Chap. 10. However, only one housekeeping parameter and one science parameter are currently handled. They are measured respectively on an AD7997 ADC and on an LM75 temperature sensor, both included on the test board (Chap. 8).



The parameter measured by the ADC is a simulation of the battery voltage, and it is used to turn the EPS2 subsystem on and off, by comparing it to a fixed threshold.

The measurements of both parameters are stored in the external EEPROM. A single telecommand (0x6D, or “m” in ASCII) allows one to retrieve the entire set of measurements stored in the external EEPROM. Since no precise requirements for the storage and retrieval of measurements were yet established, this part of the implemented software was kept very simple. For example, there is currently no time information stored with the measurement. This simplicity in the implementation was deliberate, again, to keep the software simple and easily modifiable.

- **Log module**

The clock module is implemented as discussed in Chap. 10. Each event in the log is composed of a 8-bit code, denoting the nature of the event, and of 8 other bits of information, the meaning of which depends on the nature of the event. For example, the event number 3 means a change in the current status of EPS2, and the associated information is the new value of its status.

A single telecommand (0x6C, or “l” in ASCII) allows one to retrieve the entire set of events stored in the log.

- **Clock module**

The clock module is implemented as discussed in Chap. 10.

- **Measurement of processor utilization**

A particularly useful feature for the development of the software was included in this first version. It allows one to monitor the current utilization of the processor of the active OBC (OBC1 or OBC2). It is implemented with a counter variable, incremented in the *idle hook*<sup>1</sup>, and regularly reset, by the sequencer task (although this could have been done in any other periodic task). Each time the counter is reset, its value reflects the amount of available, remaining processing time, between the two last resets. A special version of the software, with all tasks disabled (except the mechanism to reset the counter and show its value) was run first, and the average value of the counter was noted. Then, when running the actual software, the value of the counter can be compared with this previously noted value, and a percentage of “processor utilization” can be deduced.

Practically, the value of the counter is periodically sent in a particular message, on the USB interface; the periodic sending of this message can be enabled and disabled with a telecommand (0x75, or “u” in ASCII).

As a example, the current figure of utilization, in idle conditions (i.e. when

---

<sup>1</sup>The idle hook is the part of the code executed by the OS when no other task is active.

no telecommand is received and when no event has to be written in the log), is around 0.1 %.

Note that, in addition to the information above, the source code of the software (Appendix K) is extensively commented; these comments explain functioning details (such as, for example, the list of possible telecommands, or the code used to represent each type of event in the log), as well as technical implementation details. The interface of each function of each module is also formally specified, in order to allow one to easily reuse and rework on the existing code.

## 11.2 Practical details of software realization

The software is programmed in C language. We use the Rowley CrossWorks<sup>1</sup> compiler, together with the Rowley CrossStudio integrated development environment. The processors of OBC1 and OBC2 are each programmed using their own JTAG interface (Chap. 7) and the appropriate tool from Texas Instruments, the MSP-FET430UIF (Fig. 11.1). This tool connects, on one side, to the JTAG interface of one of our boards, and on the other side to the USB port of a computer. The Rowley CrossWorks compiler allows one to interact directly with this tool, to load a program into the processor, but also to perform debugging operations such as step-by-step execution. For interacting with our boards through their USB interface, we use Hilgraeve Hyperterminal<sup>2</sup> and Eltima Advanced Serial Port Terminal<sup>3</sup>.

Appendix K gives the list of the files that contain the source code, and the number of lines in each of them.



Figure 11.1: Texas Instruments' JTAG interface used to program and debug OBC1 and OBC2.

---

<sup>1</sup>See <http://www.rowley.co.uk/msp430/>.

<sup>2</sup>See <http://www.hilgraeve.com>.

<sup>3</sup>See <http://www.virtualserialport.com/products/serial-port-terminal/>.

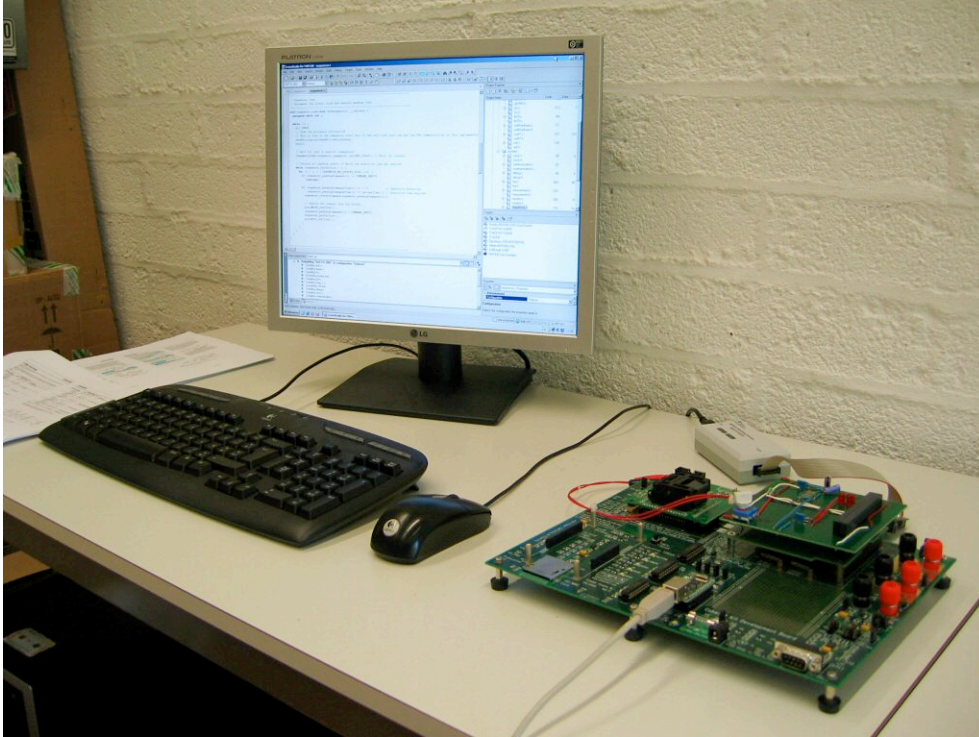


Figure 11.2: Test-bed used for integration tests.

### 11.3 Functional tests of software

The implemented modules were tested in various ways, and showed to behave according to their specifications. The first tests were *unit tests*, where each module is individually tested. Such tests allow one to manually modify the inputs to the module, and monitor its individual behavior. The subsequent tests were *integration tests*. They allow one to test the interaction between the modules, and the global behavior of the system. These tests were particularly important in our case, due to the strong interdependence of the different modules. For example, turning on a subsystem at a preprogrammed time involves the communication module, to receive that command, the sequencer module, to execute the command at the correct time, the clock module, to provide the current time to the sequencer, and finally the monitor module, to actually change the status of the subsystem.

The description and the results of an extensive test scenario can be found in Appendix L. It shows the use of all the main hardware and software features of the OBC system. The test-bed used for integration tests like this one, comprises the OBC1 (either the FM430, or its development board, which are electrically equivalent), the OBC2 (one of the engineering models provided by Deltatec), and the test board (Chap. 8), all plugged onto each other (Fig. 11.2). The USB port of the OBC1 is connected to a control computer, where a terminal software is used to send commands to and to receive messages from the OBC boards.

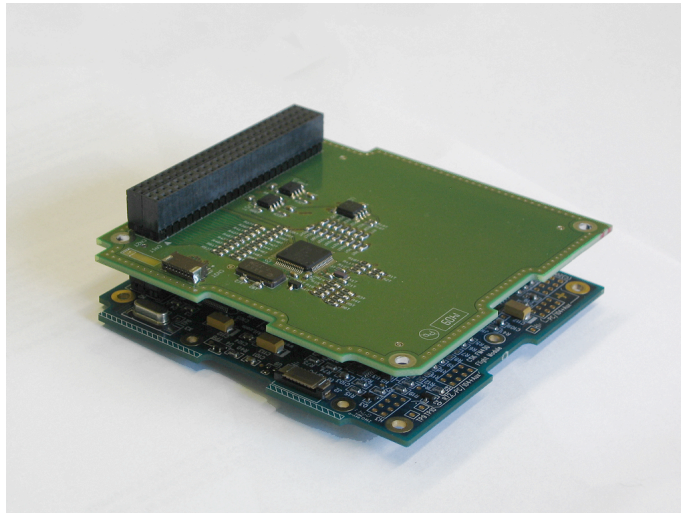


Figure 11.3: Engineering model of the OBC2, provided by Deltatec, plugged onto the OBC1 (Pumkin's FM430).

---

# Chapter 12

## Conclusions and future work

### 12.1 Conclusions

This thesis discussed the development of the hardware and software aspects of the on-board computer (OBC) of the OUFTI-1 CubeSat system.

This work started from scratch, with the identification of the exact requirements of the system. We performed an analysis of the mission and its operations. We centralized and formalized the requirements, in terms of electrical interfaces, of all the other subsystems, and we devised a suitable global electrical architecture for the satellite. We were then able to identify the exact requirements of the OBC, in terms of processing power, electrical interface, storage memory, etc. A robust solution was conceived, which consists in one off-the-shelf microcontroller board (OBC1), and in one custom board (OBC2). The complete design of this custom board was carried out, and a prototype was hand-built and tested successfully. A professional set of specifications for this custom board was then prepared, in order to get the subsequent prototypes manufactured at a specialized local electronics company, Deltatec. Their version of the board was successfully tested as well.

The development of the on-board software of the satellite was performed, starting with the identification of its precise requirements. We proposed an original, modular, and robust software architecture, consisting in six generic modules. Each of these modules was then designed and described in detail. Last but not least, a comprehensive and functional implementation of the software modules was realized, as well as an implementation of software drivers and libraries for the hardware devices to be handled by the OBC of OUFTI-1. The mission-specific details of the software could not be implemented, due to many unknowns in several aspects of the project, still at the time of this writing. However, the whole software was designed to be modular, so that many possible changes can be very easily integrated.

## 12.2 Ideas for future work

On the hardware side, the current prototypes are deemed complete and functional. If no major change in the global project arises, the only task left is to supervise the production of the flight model(s) of the OBC2.

On the software side, most of the development work left concerns mission-specific details. In particular, one will have to formalize a precise and comprehensive list of telecommands, as well as a format for the telemetry to be sent by the satellite. A formal sequence of operations, and/or a definition of satellite operating modes, will also have to be established. These definitions should state when one subsystem or another has to be turned on and off, or e.g. what to do in the event of a malfunction of a given subsystem. These operations would then have to be integrated in the software.

Similarly, the mechanism used to switch between the default and backup processors may also have to be improved. Some cooperation effort is needed, with teams working on the other electrical subsystems, in particular on the EPS, to determine all the possible failure scenarios, and the best corresponding reactions to these scenarios. In the case of the redundancy of the two processors, it is yet to determine whether it is advisable to be able to switch back to the default processor if it proved to be faulty at some time.

Some other details, that have not been precisely defined at the time of this writing, will have to be integrated in the current software architecture. In particular, a high-frequency measurement capability is likely to be needed, for monitoring the functioning of the EPS2 (see [Ledent09]).

Additional features could then be included in the software. Among the most useful ones, we suggest adding mechanisms to detect and correct the effect of single-event upsets in RAM and ROM, as well as adding an on-orbit reprogramming capability.

Finally, a task of major importance, in criticality and in the resources needed, is the testing phase. Its purpose is to assess the correct behavior of the satellite in nominal and off-nominal situations, and to discover as many bugs as possible. A significant time-frame should be planned to complete this critical task successfully.

---

# Bibliography

- [AST06] “Guide to Reusable Launch and Reentry Vehicle Software and Computing System Safety”, FAA/AST (Office of Commercial Space Transportation), July 2006.
- [Beukelaers09] Vincent Beukelaers, “From Mission Analysis to Space Flight Simulation of OUFTI-1 Nanosatellite”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [CDS08] “CubeSat Design Specification”, California Polytechnic State University, San Luis Obispo, CA, USA, February 2008.
- [Cutler06] James Cutler, Space Environment Lecture notes, Stanford University, Space and Systems Development Laboratory, Stanford, CA, USA, August 2006.
- [Dabrowski03] Michael J. Dabrowski, “The Design of a Software System of a Small Space Satellite”, Bachelor of Science thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 2003.
- [Drevon05] Claude Drevon, José Aldeguer, “Conception de cartes pour équipements spatialisables”, Techniques de l’Ingénieur, November 2005.
- [Eggert02] Daniel Eggert, “DTUsat Interboard Communication”, Technical University of Denmark, Denmark, January 2002.
- [Evrard09] Nicolas Evrard, “Développement de l’infrastructure de mesure du nanosatellite OUFTI-1”, Master thesis, Institut Gramme, Liège, Belgium, June 2009.
- [FreeRTOS] FreeRTOS documentation, <http://www.freertos.org>.
- [Galli08] Stefania Galli, “Mission Design for the CubeSat OUFTI-1”, Master thesis, University of Liège, Liège, Belgium, June 2008.
- [Halbach09] Luc Halbach, “OUFTI-1 Mission and System Definition v.1.0”, Private communication, February 2009.

- [Hannay09] Samuel Hannay, “Modeling of the Attitude Determination and Control System of the OUFTI-1 nanosatellite”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [Hardy09] Johan Hardy, “Implémentation du protocole AX.25 à bord du nanosatellite OUFTI-1”, Master thesis, Institut Gramme, Liège, Belgium, June 2009.
- [Henrard09] Renaud Henrard, *Réalisation du système de télécommunication du satellite OUFTI-1*, Master thesis, Institut Supérieur Industriel Liégeois, Liège, Belgium, June 2009.
- [IARU06] International Amateur Radio Union, “Information for developers of satellites: Planning to use frequency bands allocated to the amateur-satellite service”, October 2006.
- [ITU04] International Telecommunication Union, “Radio regulations”, 2004.
- [Jacques09] Lionel Jacques, “Thermal design of the CubeSat OUFTI-1”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [Kayali00] Sammy Kayali, “Electronic parts evaluation activities at JPL”, NASA Jet Propulsion Laboratory (JPL), April 2000.
- [Ledent09] Philippe Ledent, “Design and Implementation of On-board Digitally Controlled Electrical Power Supply of Student Nanosatellite OUFTI-1 of University of Liège”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [Mahy09] François Mahy, “Design and Implementation of On-board Telecommunication System of Student Nanosatellite OUFTI-1 of University of Liège”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [Parisis08a] Paul Parisis, “Avionics architecture”, Private communication, Spacebel, Liège, Belgium, December 2008.
- [Parisis08b] Paul Parisis, “On Board Software Overview”, Private communication, Spacebel, Liège, Belgium, December 2008.
- [Pisane08] Jonathan Pisane, “Design and Implementation of the Terrestrial and Space Telecommunication Elements of the Student Nanosatellite of the University of Liège”, Master thesis, University of Liège, Liège, Belgium, June 2008.
- [Pierlot09] Gauthier Pierlot, “OUFTI-1: Configuration, Structural Integrity and Vibration Testing”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [Pumpkin08] Pumpkin Inc., “CubeSat Kit FM430 Flight Module Manual”, Rev. C, June 2008.



- [Schmidt07] Marco Schmidt, Klaus Schilling, “An extensible on-board data handling software platform for pico satellites”, University of Wuerzburg, Wuerzburg, Germany, December 2007.
- [SwisscubeA] “Swisscube Phase A: Mission, Space, and Ground System Description”, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, June 2006.
- [SwisscubeB] “Swisscube Phase B: Project, Mission, Space, and Ground System Overview”, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, February 2007.
- [SwisscubeCDMSB] “Swisscube Phase B: CDMS”, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, February 2007.
- [SwisscubeFT] “Swisscube CDMS Fonctionnal Tests”, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, December 2007.
- [Thirion09] Pierre Thirion, “Design and Implementation of On-board Electrical Power Supply of Student Nanosatellite OUFTI-1 of University of Liège”, Master thesis, University of Liège, Liège, Belgium, June 2009.
- [Wertz09] Jérôme Wertz, “Conception et réalisation du système de déploiement des antennes du nanosatellite OUFTI-1”, Master thesis, Institut Gramme, Liège, Belgium, June 2009.

---

# Appendices

---

## Appendix A

### List of parameters to be measured aboard OUFTI-1

The following table gives the list of housekeeping and science parameters that were chosen to be measured aboard OUFTI-1 during its normal operations. This table is an excerpt from [Evrard09].

Measurement point	Type	Name
<b>Mechanical structure</b>		
Cube face 1	Temperature	T_F1
Cube face 2	Temperature	T_F2
Cube face 3	Temperature	T_F3
Cube face 4	Temperature	T_F4
Cube face 5	Temperature	T_F5
Cube face 6	Temperature	T_F6
<b>Telecommunication system (COM)</b>		
3.3V current	Current	I_COM3.3
7.2V current	Current	I_COM7.2
Amplifier temperature	Temperature	T_COM7.2
SWR	SWR	SWR_COM

---

A. LIST OF PARAMETERS TO BE MEASURED ABOARD OUFTI-1

---

Measurement point	Type	Name
<b>Radio beacon (BCN)</b>		
Beacon 3.3V current	Current	I_BC3.3
Beacon 7.2V current	Current	I_BC7.2
Beacon amplifier temperature	Temperature	T_BC7.2
<b>Electrical power supply (EPS)</b>		
Solar cell 1 current	Current	I_SC1
Solar cell 2 current	Current	I_SC2
Solar cell 3 current	Current	I_SC3
Solar cell 4 current	Current	I_SC4
Solar cell 5 current	Current	I_SC5
Solar cell total current	Current	I_SCT
Batteries temperature	Temperature	T_BAT
Dissipator temperature	Temperature	T_LM94022
Batteries voltage	Voltage	V_BAT
7V converter temperature	Temperature	T_7.0
3.3V bus	Voltage	V_3.3
5.0V bus	Voltage	V_5.0
7.2V bus	Voltage	V_7.2
<b>Experimental electrical power supply (EPS2)</b>		
3.3V converter output	Voltage	V_EPS2OUT
Converter current	Current	I_EPS2OUT
Digital circuit voltage	Voltage	V_EPS2DIG
Digital circuit current	Current	I_EPS2DIG
Digital circuit temperature	Temperature	T_EPS2DIG
Converter temperature	Temperature	T_EPS2CONV

---

## Appendix B

# Investigation of the use of two stacked FM430 boards

The use of two FM430 board was examined as a means of providing redundancy. Using two identical boards implies that the CubeSat bus connectors of the two boards would be directly connected together, and some careful analysis was needed to determine whether this was compatible with the hardware.

By analyzing the use of each pin of the CubeSat bus (described in the FM430 datasheet), I determined the consequences of stacking two FM430 boards.

- The six 8-bit I/O ports of the MSP430s of both boards would be connected together (since they are all available on the CubeSat bus). This implies that the pins configured as outputs on one board have to be configured as inputs or as high impedance pins on the other board.
- The VREF+ pins of the two MSP430s would be connected together. These pins provide the ADC reference voltage as output. Since this is an output, connecting them together should be avoided. This voltage is not likely to be useful in our application, so the printed circuit board track between one of the MSP430s and that pin on its bus connector could be cut to solve the problem.
- The USB communication lines of the two boards would be connected together. They would be connected at the level between the MSP430 and the buffer. The pins of the MSP430 used for USB transmission (Tx) do not pose any more problems than any other output (see the first point). The pins of the MSP430 used for USB reception (Rx) would be connected together between the buffer and the MSP430. This may be problematic if the two buffers output different values. Each buffer has an output enable (OE) input,

which can be used to set its outputs to high impedance. This OE line is available on the CubeSat bus, and is thus connected between the two boards. A solution is to cut the corresponding track on one of the boards, so that each board may enable its own buffer independently.

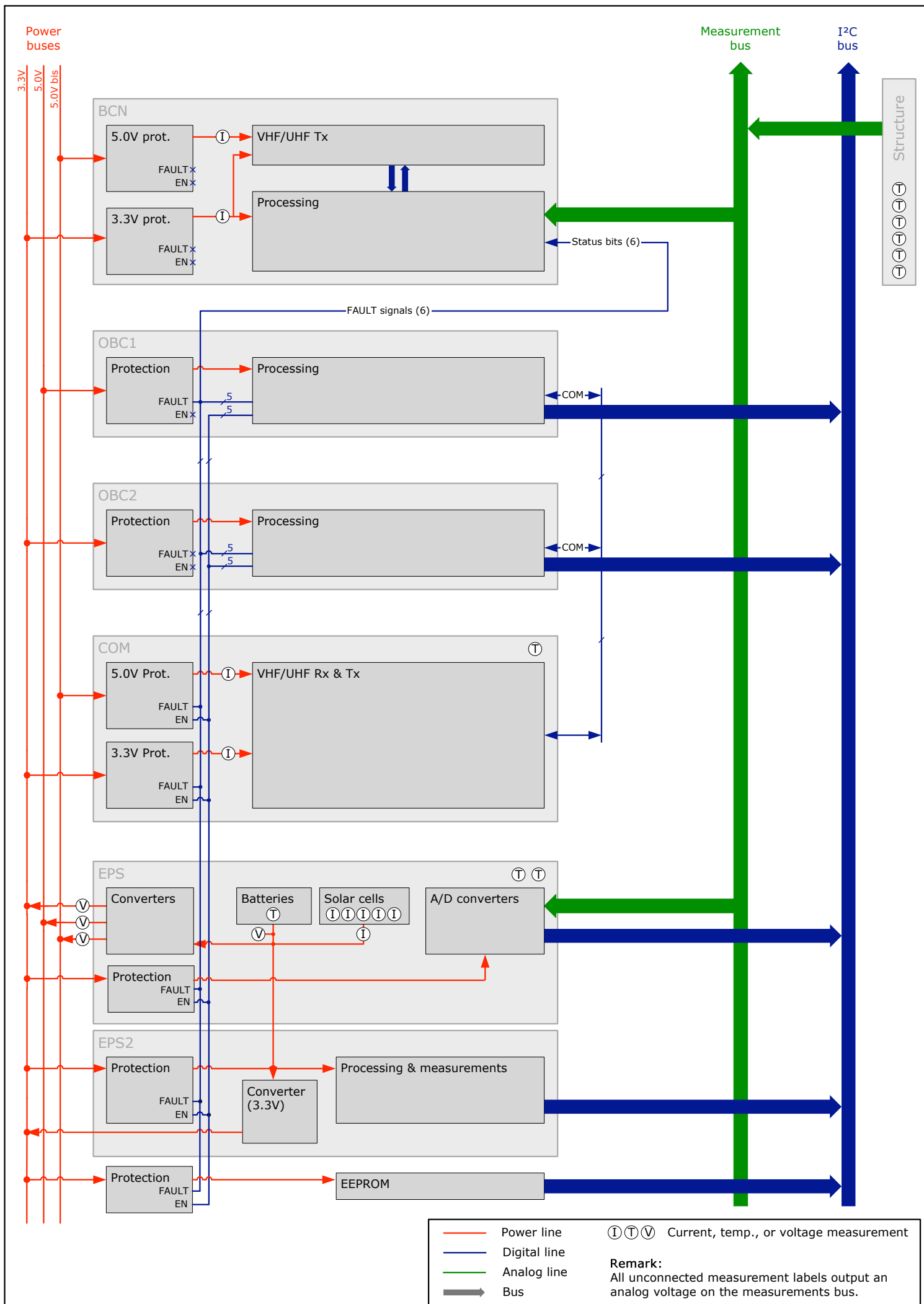
- All the pins of the SD card sockets of both boards would be connected together. This implies that it would be impossible to write to only one of the two cards. This may also pose a problem in the case when a read, occurring on both cards at the same time, returns something different for the two cards. This problem may easily be avoided by using only a single SD card, thus located on only one of the two boards.
- The FAULT signals of both boards would be connected together. These signals come from current-limiting MAX890 integrated circuits (IC). The output of these ICs is an active-low, open-drain signal, so this would not be a problem. However, it would be impossible to know whether a FAULT signal is originating from one board or the other.
- The SENSE and Vcc signals of both boards would be respectively connected together. These are sensing points of the MSP430 supply voltage, that are respectively connected after the current-limiting switch and after the LDO. This should be avoided since we would lose the protection by connecting the SENSE pins, and we may lose some precision in the supply voltage regulation by connecting the Vcc pins. We would then recommend to cut the tracks going to these pins on the bus.
- All other pins on the bus are either not connected, or inputs, so they do not pose any problem.

After this analysis, it appears that directly stacking two FM430 boards is not impossible, but it requires to isolate several signals between the two boards. Let us also mention that physically cutting tracks on these boards is not recommended and may not be feasible at all. Indeed, these boards are multi-layered, and the problematic signals may very well use buried tracks. A solution around this problem, that uses one unmodified FM430 and one custom board, is presented in Chap. 6.

---

## Appendix C

### Overall electrical architecture and data flow paths of OUFTI-1





---

## Appendix D

### Allocation of the CubeSat bus

Bus pin	Description	Use on each board				COM	Name
		EPS1	EPS2	OBC1	OBC2/beacon		
H1 1	Enable signal for the deployment system of the antenna Generated by the OBCs, handled on the EPS board	I: antenna deployment 2	-	P5.7	P5.7	-	ANTENNA-DEPLOYMENT1_EN
H1 2	Signal similar/redundant with ANTENNA-DEPLOYMENT1_EN	I: antenna deployment 2	-	P5.6	P5.6	-	ANTENNA-DEPLOYMENT2_EN
H1 3	Signal to choose the load of the EPS2 (dummy load/3.3V bus)	-	I: load switch	P5.5	P5.5	-	EPS2_LOAD_SELECT
H1 4	COM 3.3V power enable	-	-	P5.4	P5.4	I: EN 3.3V	EN_COM5.0
H1 5	COM 7.2V power enable	-	-	P5.3	P5.3	I: EN 7.2V	EN_COM3.3
H1 6	EPS2 power enable (turn on its 'command' part)	-	I: EN	P5.2	P5.2	-	EN_EPS2
H1 7	EEPROM power enable	-	-	P5.1	P5.1	-	EN_EEPROM
H1 8	Measurements circuits power enable See POWER_3.3V_MEASUREMENTS	I: EN measurements	-	P5.0	P5.0	I: EN EEPROM	EN_MEASUREMENTS
H1 9	COM 7.2V OBC interface details TBD	-	-	P4.7	P4.7	ADF...	COM_ADFxxx
H1 10	Not used	-	-	O: P4.6 I: MXX power enable	O: P4.6	-	-
H1 11	Not used	-	-	O: P4.5 I: SD card power enable	O: P4.5	-	-
H1 12	COM: ADF-OBC interface details TBD	-	-	P4.4	P4.4	ADF...	COM_ADFxxx
H1 13	COM: ADF-OBC interface details TBD	-	-	P4.3	P4.3	ADF...	COM_ADFxxx
H1 14	COM: ADF-OBC interface details TBD	-	-	P4.2	P4.2	ADF...	COM_ADFxxx
H1 15	COM: ADF-OBC interface details TBD	-	-	P4.1	P4.1	ADF...	COM_ADFxxx
H1 16	COM: ADF-OBC interface details TBD	-	-	P4.0	P4.0	ADF...	COM_ADFxxx
H1 17	OBCs UART1 Rx, connected to the USB interface located on OBC1	-	-	P3.7 + USB Rx	P3.7	-	USB_RX
H1 18	OBCs UART1 Tx, connected to the USB interface located on OBC1	-	-	P3.6 + USB Tx	P3.6	-	USB_TX
H1 19	COM: ADF-OBC interface details TBD	-	-	P3.5	P3.5	ADF...	COM_ADFxxx
H1 20	COM: ADF-OBC interface details TBD	-	-	P3.4	P3.4	ADF...	COM_ADFxxx
H1 21	I2C SCL	I/O: I2C Pull-up to 3.3v	I/O: I2C	I/O: P3.3	I/O: P3.3	-	I2C_SCL
H1 22	Not used	-	-	P3.2	P3.2	-	-
H1 23	I2C SDA Kept to GND by software to isolate the two adjacent I2C signals	I/O: I2C Pull-up to 3.3v	I/O: I2C	I/O: P3.1	I/O: P3.1	-	I2C_SDA
H1 24	Not used	-	-	O: P3.0 I: SD card enable pull up to 3.3v	O: P3.0	-	-
H1 25	OBC1 overcurrent fault signal	-	-	O: fault (open collector)	-	-	FAULT_OBC1
H1 26	Not used	-	-	O: MSP430 Vref+	-	-	-
H1 27	Not used	-	-	O: current sense	-	-	-
H1 28	Not used	-	-	I: MSP430 Vref+	-	-	-
H1 29	Not used	-	-	I: reset	-	-	-
H1 30	Not used	-	-	I/O: MSP430 Vref-	-	-	-
H1 31	OBC1 power enable	-	-	I: enable power Pulled low	-	-	EN_OBC1
H1 32	5V power supply fed from the USB connector located on OBC1	I: battery charge power	O: USB 5V (when USB connected)	-	-	-	POWER_5V_USB
H1 33	Not used	-	-	O: 5V SW	-	-	-
H1 34	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 35	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 36	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 37	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 38	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 39	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 40	Not used	-	-	reserved for MXX transceiver	-	-	-
H1 41	Global oscillator clock signal	-	-	-	I: clock	-	TCXO
H1 42	3.3V power supply for measurement circuits; generated on EPS1, the protection is also located on EPS1, but this power supply is used by various measurement circuits aboard the satellite	O: 3.3V (after protection)	I: 3.3V for measurement circuits	-	-	I: 3.3V for measurement circuits	POWER_3.3V_MEASUREMENTS
H1 43	free	-	-	-	-	-	-
H1 44	free	-	-	-	-	-	-
H1 45	free	-	-	-	-	-	-
H1 46	free	-	-	-	-	-	-
H1 47	Beacon analog input	-	-	-	I: beacon analog signal	-	BCN_INPUT1 (I SCT)
H1 48	Beacon analog input	-	-	-	I: beacon analog signal	-	BCN_INPUT2 (I BAT)
H1 49	Beacon analog input	-	-	-	I: beacon analog signal	-	BCN_INPUT3 (I EPS)
H1 50	Beacon analog input	-	-	-	I: beacon analog signal	-	BCN_INPUT4 (I BAT)
H1 51	Beacon analog input	-	-	-	I: beacon analog signal	-	BCN_INPUT5 (V 3.3)
H1 52	Beacon analog input	-	-	-	I: beacon analog signal	-	BCN_INPUT6 (V 5.0)
H2 1	Battery heater enable	-	-	P6.7	P6.7	-	BATTERY-HEATER_EN
H2 2	Enable MXX interface, located on OBC1	-	-	P6.6	P6.6	-	MXX_EN
H2 3	Held high by software to keep it disabled	-	-	P6.5	P6.5	-	USB_RST
H2 4	RST: USB, signal related to the USB interface, located on OBC1	-	-	P6.4	P6.4	-	USB_RI
H2 5	C1S: USB, signal related to the USB interface, located on OBC1	-	-	P6.3	P6.3	-	USB_C1S

H2 6	PWE	USB; signal related to the USB interface, located on OBC1	-	-	P6.2	-	USB_PWE
H2 7	DTX	USB; signal related to the USB interface, located on OBC1	-	-	P6.1	-	USB_DTR
H2 8	RTS	USB; signal related to the USB interface, located on OBC1	-	-	P6.0	-	USB_RTS
H2 9	Enable	signal of the USB interface, located on OBC1	-	-	O: P1.7	-	USB_EN
H2 10	COM 3.3V overcurrent	fault signal	-	-	I: USB control	-	FAULT_COM3.3
H2 11	COM 5.0V overcurrent	fault signal	-	-	P1.6	O: fault (open collector)	FAULT_COM5.0
H2 12	EP52 overcurrent	fault signal (of the command logic)	-	-	P1.5	O: 7.2V fault (open collector)	FAULT_EP52
H2 13	EEPROM overcurrent	fault signal	-	-	P1.4	-	FAULT_EEPROM
H2 14	Measurement circuits overcurrent	fault signal	-	-	P1.3	-	FAULT_MEASUREMENTS
H2 15	COM_ADF_OBC interface details TBD	-	-	-	O: EEPROM fault (open collector)	-	COM_ADFxxx
H2 16	COM_ADF_OBC interface details TBD	-	-	-	P1.2	ADF...	COM_ADFxxx
H2 17	COM_ADF_OBC interface details TBD	-	-	-	P1.1	ADF...	COM_ADFxxx
H2 18	COM_ADF_OBC interface details TBD	-	-	-	P1.0	ADF...	COM_ADFxxx
H2 19	COM_ADF_OBC interface details TBD	-	-	-	P2.7	ADF...	COM_ADFxxx
H2 20	COM_ADF_OBC interface details TBD	-	-	-	P2.6	ADF...	COM_ADFxxx
H2 21	COM_ADF_OBC interface details TBD	-	-	-	P2.5	ADF...	COM_ADFxxx
H2 22	COM_ADF_OBC interface details TBD	-	-	-	P2.4	ADF...	COM_ADFxxx
H2 23	COM_ADF_OBC interface details TBD	-	-	-	P2.3	ADF...	COM_ADFxxx
H2 24	COM_ADF_OBC interface details TBD	-	-	-	P2.2	ADF...	COM_ADFxxx
H2 25	COM_ADF_OBC interface details TBD	-	-	-	P2.1	ADF...	COM_ADFxxx
H2 26	COM_ADF_OBC interface details TBD	-	-	-	P2.0	ADF...	COM_ADFxxx
H2 27	COM_ADF_OBC interface details TBD	-	-	-	I: power (5V)	-	POWER_5V_BUS
H2 28	COM_ADF_OBC interface details TBD	-	-	-	I: power (3.3V)	-	POWER_3.3V_BUS
H2 29	COM_ADF_OBC interface details TBD	-	-	-	I: power (3.3V)	-	POWER_3.3V_BUS
H2 30	COM_ADF_OBC interface details TBD	-	-	-	I: power (3.3V)	-	POWER_3.3V_BUS
H2 31	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 32	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 33	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 34	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 35	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 36	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 37	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 38	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 39	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 40	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 41	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 42	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 43	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 44	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 45	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 46	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 47	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 48	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 49	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 50	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 51	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND
H2 52	COM_ADF_OBC interface details TBD	-	-	-	GND	GND	POWER_DGND

Free

Power

Power-related signals (resets, faults, switches etc.)

Cannot be used (reserved by a single board)

Digital signal

Analog signal

Color code

Abbreviations

I

O

I/O

NC

Input

Output

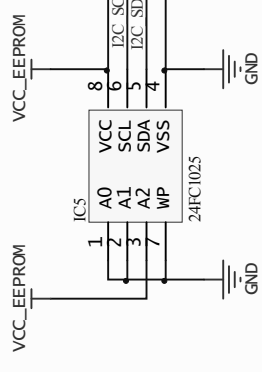
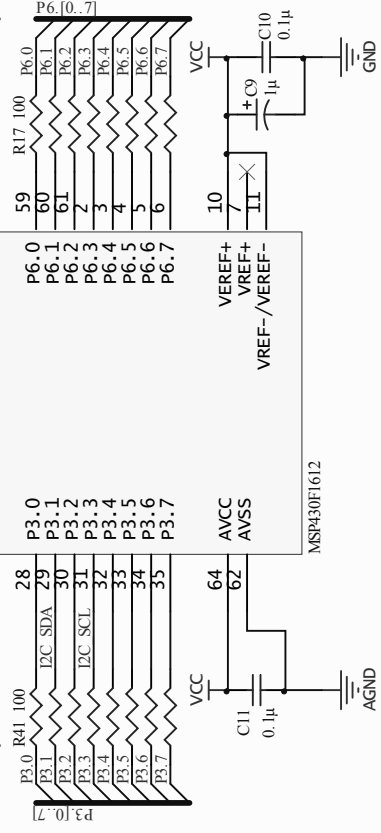
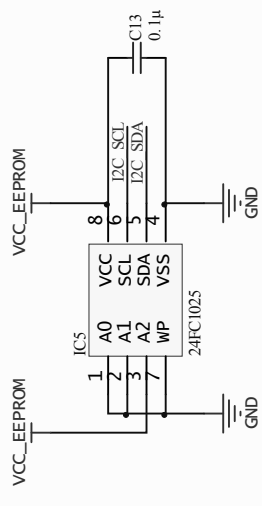
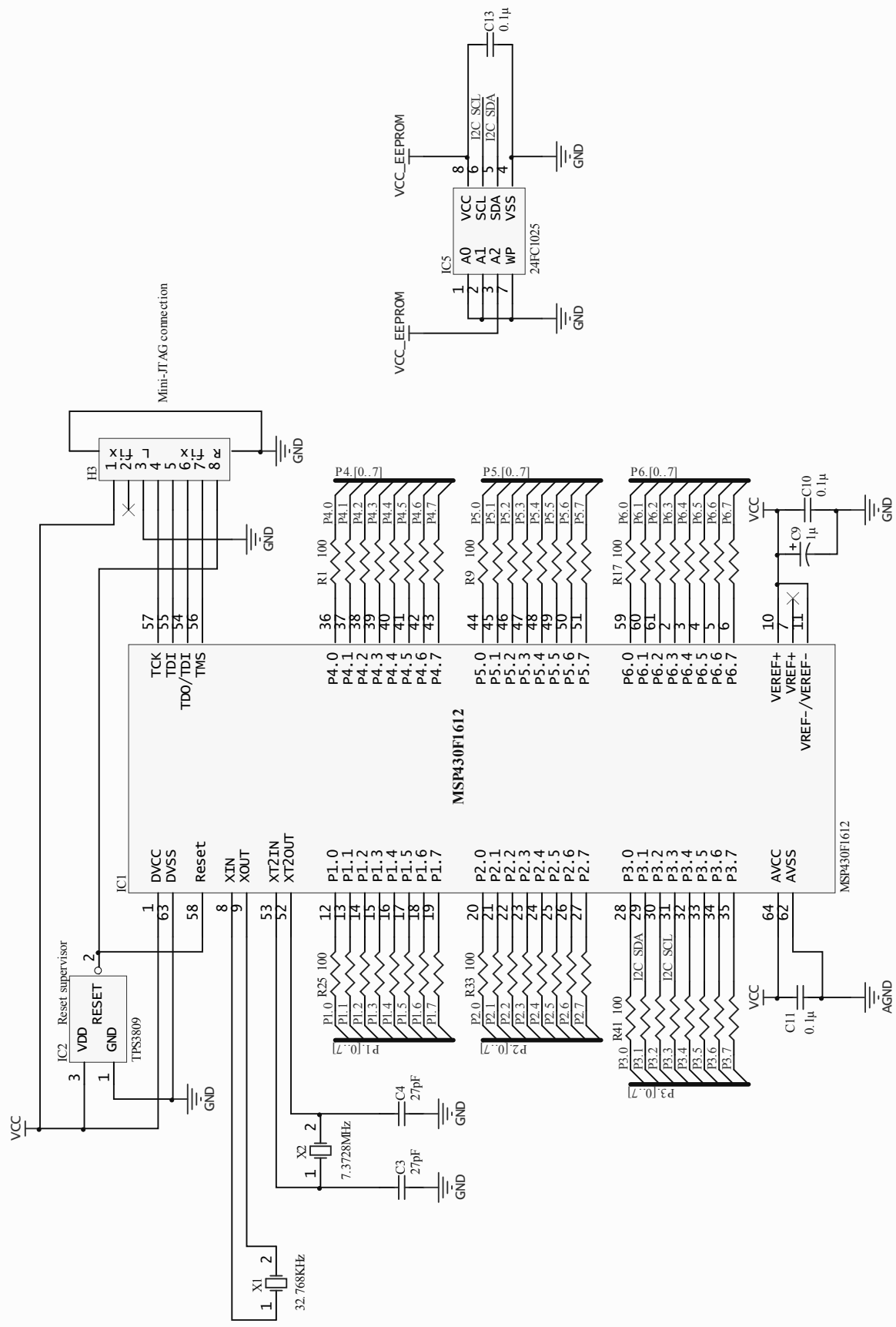
Input/output

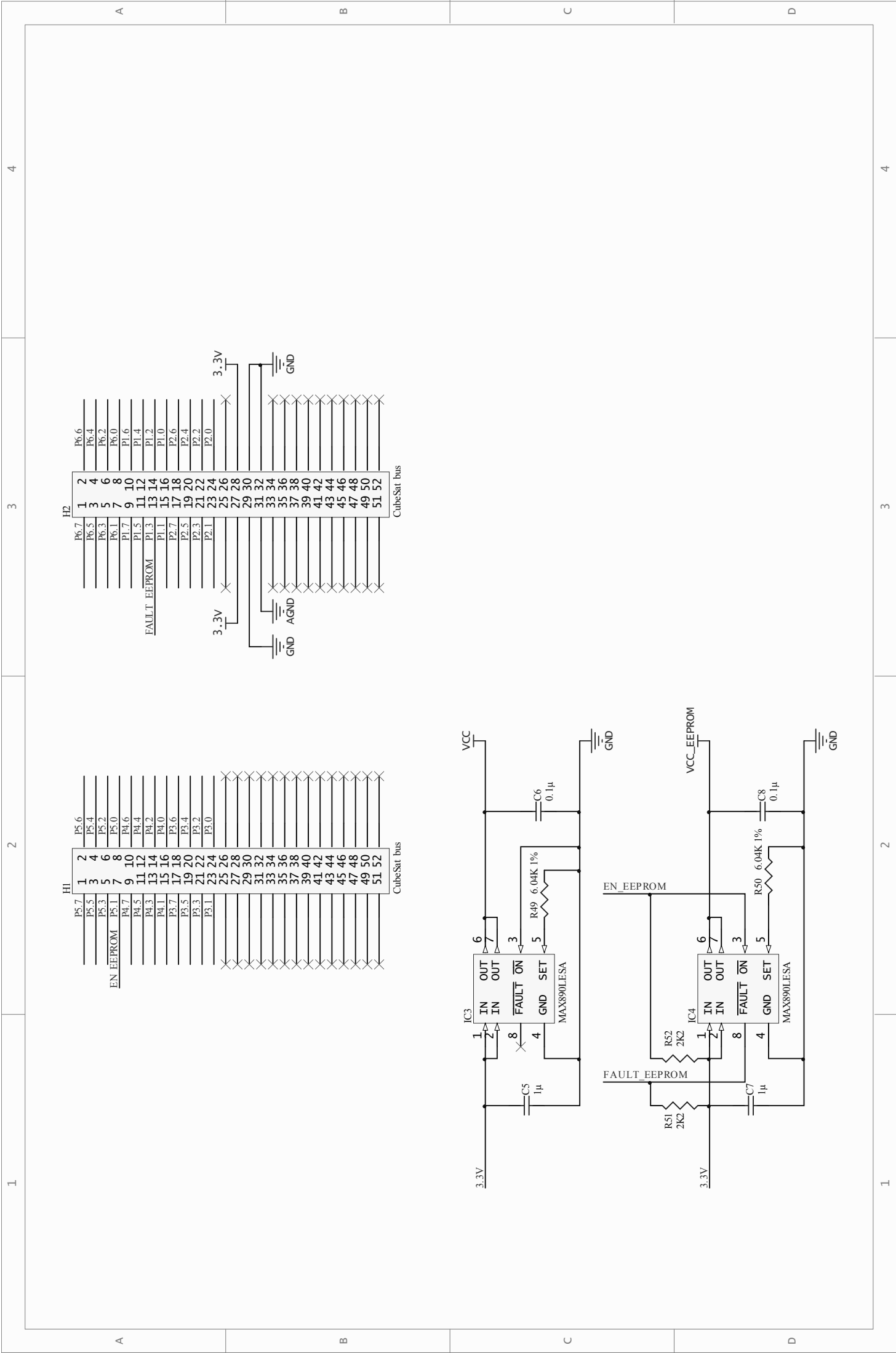
Not connected

---

## Appendix E

OBC2 electrical schematics  
originally proposed by author





---

## Appendix F

OBC2 electrical schematics  
re-encoded by Deltatec



Rue Gilles Magnée 92/6  
B-4430 ANS  
BELGIUM

TEL : (32) 4 239 78 80  
FAX : (32) 4 239 78 89  
e-mail : main@deltatec.be

**CUSTOMER :**

ULG

**DESIGN TOP LEVEL :**

**PROJECT :**

PROJECT NUMBER : 521

PROJECT NAME : OUF TI 1

PROJECT PART : OBC2

**REFERENCE :**

DOCUMENT ID : 20597

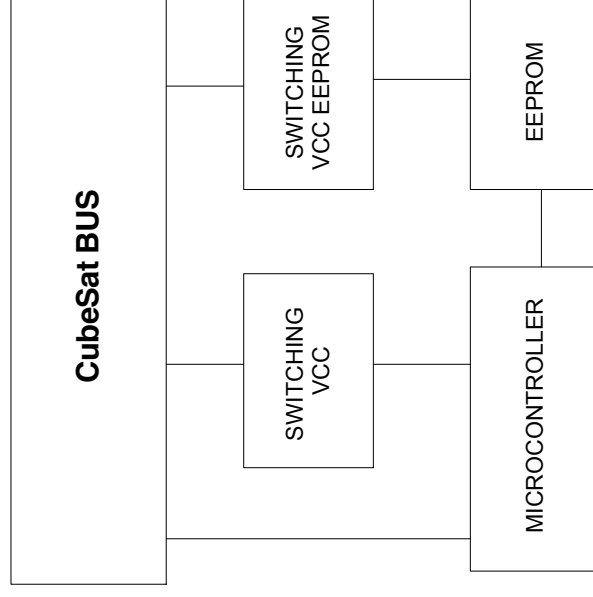
REVISION : 1.0

REDRAW : 0

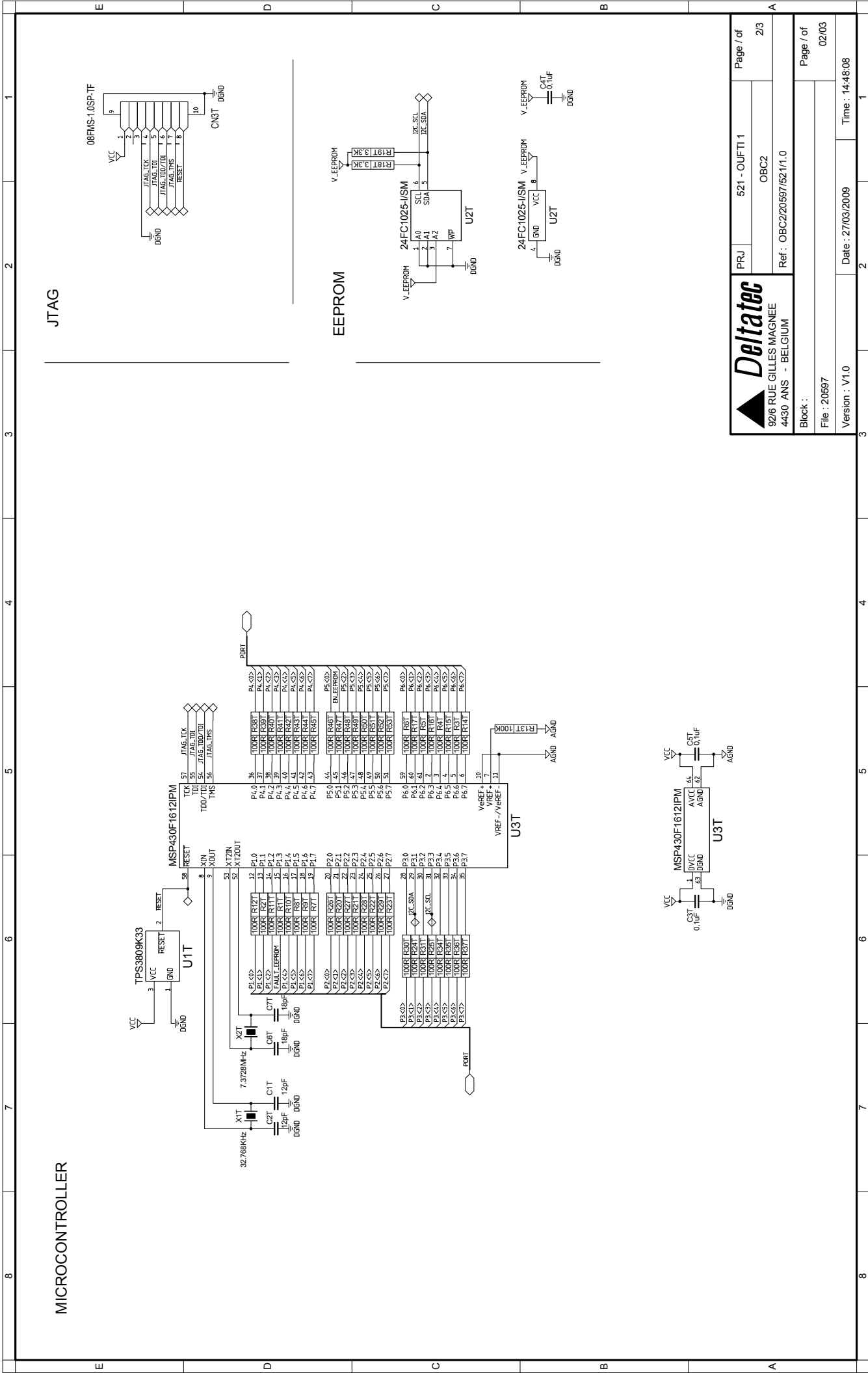
DATE : 27/03/2009

**SHEET COUNT :**

3



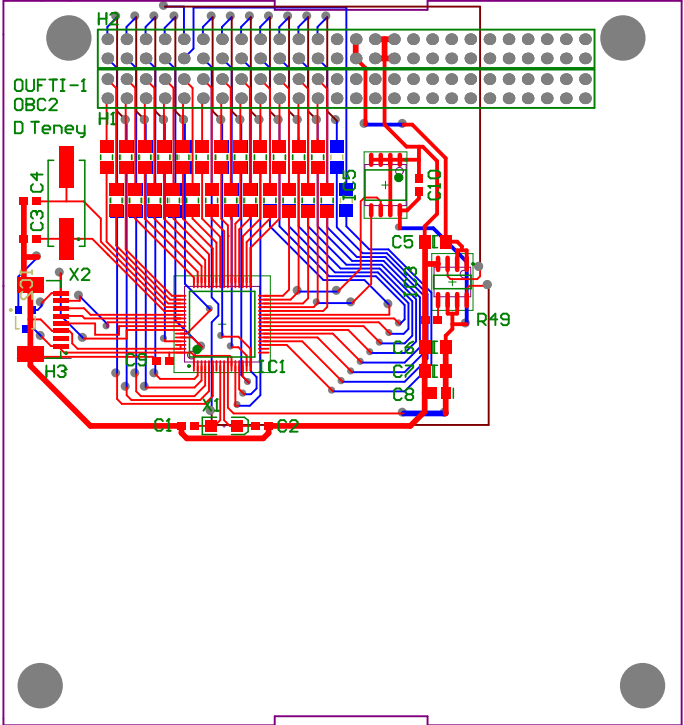





---

## Appendix G

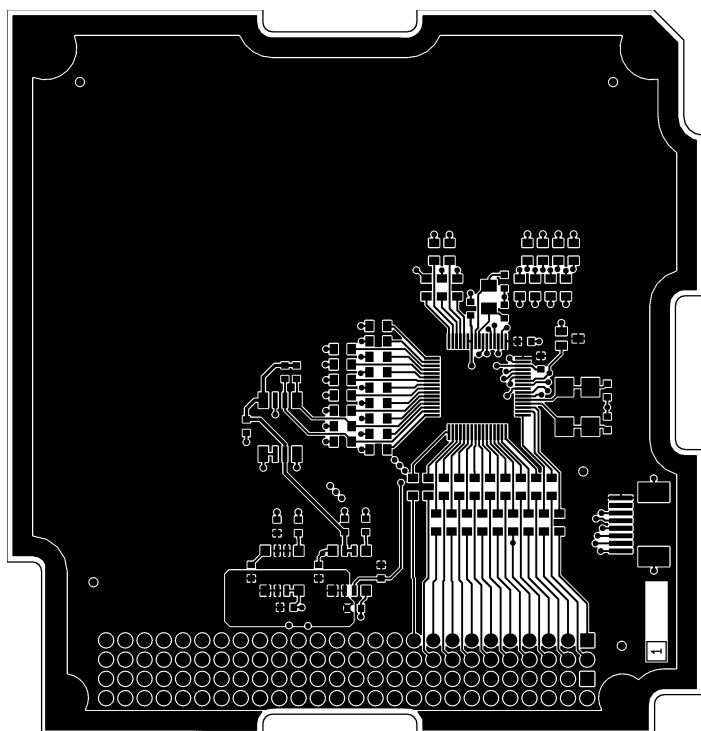
OBC2 engineering model PCB  
originally proposed by author




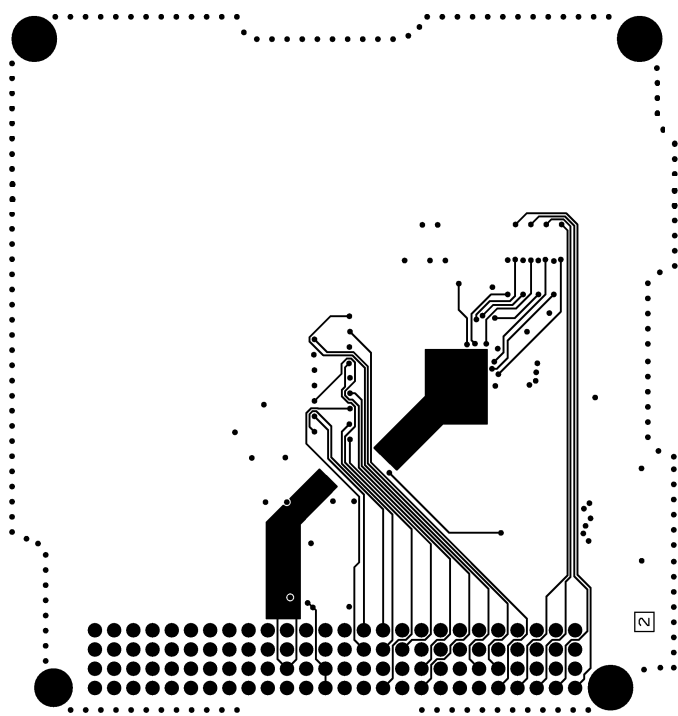
---

## Appendix H

OBC2 engineering model PCB  
designed by Deltatec



	File Name : 20597 _ COMPNT _ 1.00 .GDO	Date : 23/03/09	Version : 1.00
	DIR : P:/DT/PRO/ 521 / OBC2/PCB/	VIEW : TOP	
	Reference : PCD/ 20597 / 521 / 1.00	COMPONENT LAYER	
	Rue Gilles Magnée, 92/6 - 4430 Ans - BELGIUM - Tel. +32 4 239 78 80 - Fax +32 4 239 78 89 - <a href="http://www.deltatec.be">www.deltatec.be</a> - E-mail : <a href="mailto:main@deltatec.be">main@deltatec.be</a>		



File Name : 20597 \_ INTER1 \_ 1.00 .GDO

Date : 23/03/09

Version : 1.00

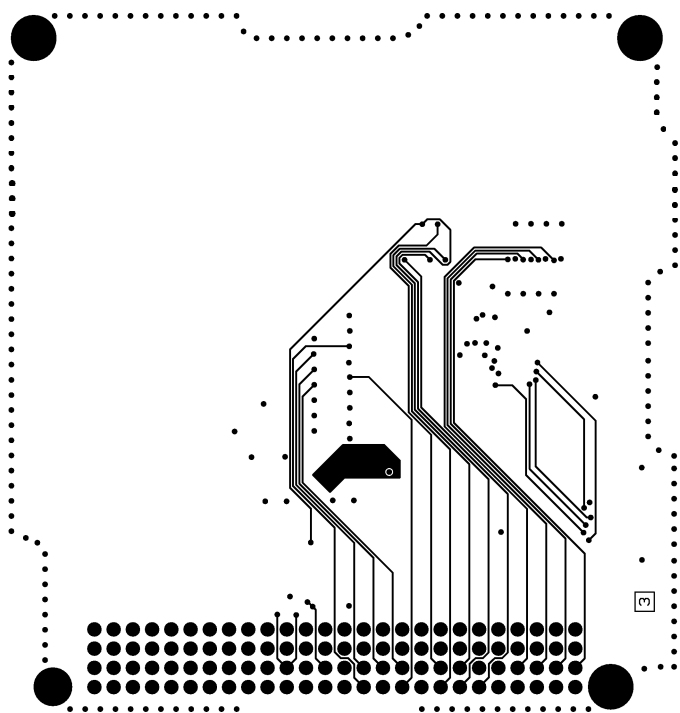
DIR : P:/DT/PRO/ 521 / OBC2/PCB/

VIEW : TOP

Reference : PCD/ 20597 / 521 / 1.00

INTERNAL 1 LAYER

Rue Gilles Magnée, 92/6 - 4430 Ans - BELGIUM - Tel. +32 4 239 78 80 - Fax +32 4 239 78 89 - [www.deltatec.be](http://www.deltatec.be) - E-mail : [main@deltatec.be](mailto:main@deltatec.be)



File Name : 20597 \_ INTER2 \_ 1.00 .GDO

Date : 23/03/09

Version : 1.00

DIR : P:/DT/PRO/ 521 / OBC2/PCB/

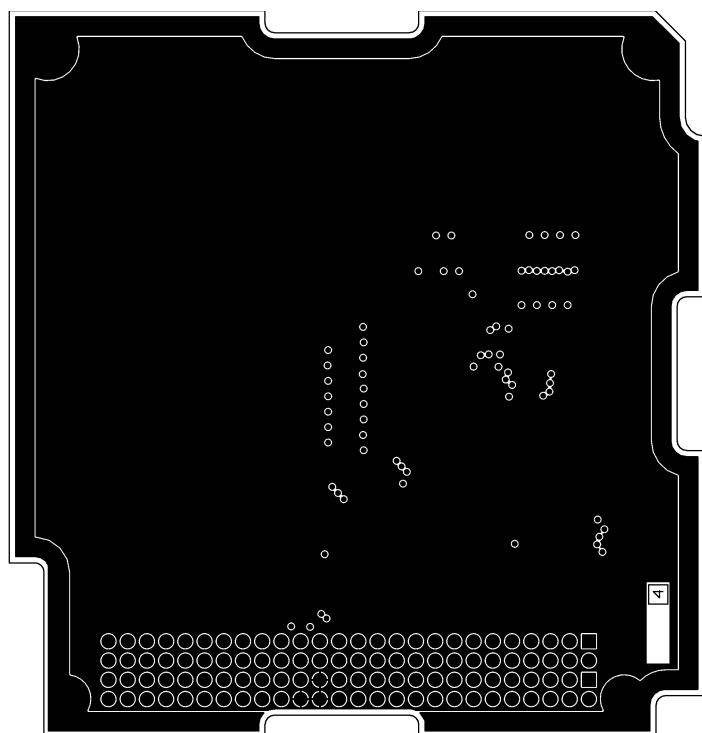
VIEW : TOP


Reference : PCD/ 20597 / 521 / 1.00

INTERNAL 2 LAYER

Rue Gilles Magnée, 92/6 - 4430 Ans - BELGIUM - Tel. +32 4 239 78 80 - Fax +32 4 239 78 89 - [www.deltatec.be](http://www.deltatec.be) - E-mail : [main@deltatec.be](mailto:main@deltatec.be)





	File Name : 20597 _ SOLDER _ 1.00 .GDO	Date : 23/03/09	Version : 1.00
	DIR : P:/DT/PRO/ 521 / OBC2/PCB/	VIEW : TOP	
	Reference : PCD/ 20597 / 521 / 1.00	SOLDER LAYER	
Rue Gilles Magnée, 92/6 - 4430 Ans - BELGIUM - Tel. +32 4 239 78 80 - Fax +32 4 239 78 89 - <a href="http://www.deltatec.be">www.deltatec.be</a> - E-mail : <a href="mailto:main@deltatec.be">main@deltatec.be</a>			

---

# Appendix I

## Handling of software functions by the application modules

This appendix is an addendum to Chap. 9. It explains, at a high level, how each functionality, listed in Sect. 9.1, is handled by the various software modules, listed in Sect. 9.2.2. As a reminder, they consist in the monitor module, the communication module, the sequencer module, the measurement module, the log module, and the clock module.

1. *Perform the initial satellite operations (antenna deployment, first activation of the other subsystems) according to a predefined sequence.*

These operations are executed by the sequencer module. The exact sequence and timing of the operations is not yet determined, but they are to be placed in the initial list of tasks of the sequencer, which will execute them with the programmed timing.

2. *Perform AX.25 and D-STAR encoding and decoding.*

These complex operations are naturally handled by the communication module. The development of this part of the software is the responsibility of the team working on the COM subsystem.

3. *Handle telecommands received on the uplink channel.*

All telecommands are first decoded by the communication module, then placed, with their planned execution time, in the task list of the sequencer. All the telecommands are eventually executed by the sequencer.

4. *Perform measurements of housekeeping and science parameters aboard the satellite.*

The measurements are naturally performed by the measurement module.

5. *Store relevant measurements until they can be sent to the ground station.*

The measurements, once sampled by the measurement module, are written to the external EEPROM.

6. *Respond to telemetry requests by sending current or stored measurements.*

A “telemetry request” command is first decoded by the communication module, then executed by the sequencer (like any other telecommand), which loads the requested data from the memory and puts it in the downlink buffer of the communication module.

7. *Provide a time reference.*

The current time is maintained by the clock module. It provides an interface to get the current time from any other module.

8. *Perform power supply management, by enabling and disabling other subsystems in predefined conditions (e.g. a low battery voltage).*

These conditions are measured by the measurement module, which also immediately applies the appropriate tests on them. If these tests imply changes in the activation of subsystems, these changes are requested, and the monitor module is then responsible for making these changes effective. Note that the monitor may not apply the changes if it was given information that has a higher priority than the changes requested by the measurement module (e.g. a status change request from a telecommand, that has a higher priority than automatic actions).

9. *Perform power cycling in case of latchup in a subsystem (detected with the FAULT signals).*

The status of the FAULT lines is sensed directly by the monitor module, which then takes care to power cycle or disable problematic subsystems.

10. *Manage the experimental electrical power supply, by enabling and disabling it in predefined conditions.*

This functionality is a particular case of point 8.

11. *Manage the D-STAR payload, by configuring it (e.g. for Doppler compensation) according to data received by specific telecommands.*

The communication module provides an interface to configure it from any other module. Therefore, as an example, a planned sequence of frequency corrections

can be uploaded via several telecommands, with appropriate execution times. They are executed by the sequencer, which then reconfigures the communication module as requested.

12. *Keep a log of meaningful events happening aboard the satellite, that can then be retrieved with one or more specific telecommands.*

The log is naturally maintained by the log module. It also provides an interface to retrieve its contents. The mechanism of the retrieval is thus similar to the one used in point 6.

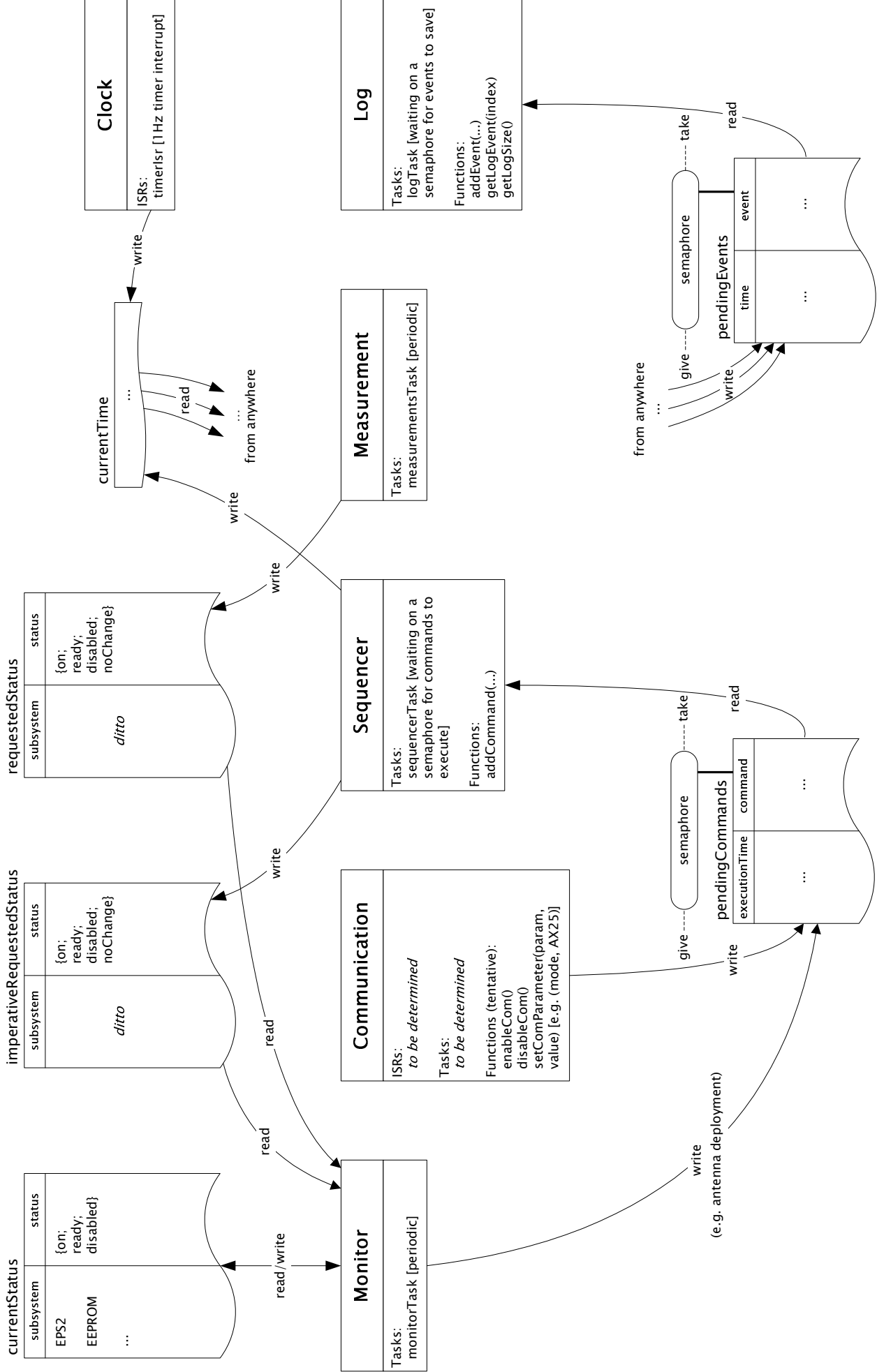
13. *Monitor, for the backup processor (OBC1), monitor the activity of the default processor (OBC2) and detect when it stops functioning.*

This system is implemented using existing communication lines between the two processors: the I<sup>2</sup>C bus. The default processor is the bus master, and regularly sends a predefined I<sup>2</sup>C message to the backup processor, then configured as a slave. If no message is received by the backup processor during a prescribed length of time, it concludes that the default processor has become ineffective, it configures itself as the bus master, and then starts to handle the normal operations.

---

## Appendix J

### Static architecture of software



---

# Appendix K

## Source code of software

The source code of the developed software can be found at the following address:

<http://www.student.montefiore.ulg.ac.be/~teney/thesis/>

Here is the list of the files that contain the source code, and the number of lines in each of them.

/OUFTI1.hzp (CrossStudio project file)	N/A
/src/boolean.h	11 lines
/src/FreeRTOSConfig.h	48 lines
/src/main.c	118 lines
/src/main.h	26 lines
/src/drivers/24xx1025.c	162 lines
/src/drivers/24xx1025.h	15 lines
/src/drivers/ad7997.c	65 lines
/src/drivers/ad7997.h	17 lines
/src/drivers/i2c.c	209 lines
/src/drivers/i2c.h	30 lines
/src/drivers/lm75.c	64 lines
/src/drivers/lm75.h	8 lines
/src/drivers/ouftiHardware.c	101 lines
/src/drivers/ouftiHardware.h	62 lines
/src/drivers/usart1.c	234 lines
/src/drivers/usart1.h	36 lines
/src/drivers/usb.c	77 lines
/src/drivers/usb.h	18 lines

/src/modules/clock.c	39 lines
/src/modules/clock.h	7 lines
/src/modules/communication.c	40 lines
/src/modules/communication.h	8 lines
/src/modules/debug.c	84 lines
/src/modules/debug.h	17 lines
/src/modules/log.c	199 lines
/src/modules/log.h	46 lines
/src/modules/measurement.c	106 lines
/src/modules/measurement.h	35 lines
/src/modules/monitor.c	120 lines
/src/modules/monitor.h	30 lines
/src/modules/sequencer.c	204 lines
/src/modules/sequencer.h	29 lines
Total	1913 lines



---

# Appendix L

## Test scenario and results

Here is the description and the results of an extensive test scenario, that shows the main features of the implemented software. The hardware test-bed used for this test is the OBC1 (either the FM430, or its development board, which are electrically equivalent), the OBC2 (one of the engineering models provided by Deltatec), and the custom test board (Chap. 8), all plugged onto each other. The USB port of the OBC1 is connected to a control computer, where a terminal software is used to send commands to and receive messages from the OBC boards.

Time	Message received (Rx) or transmitted (Tx)	Comments and observations
-----		
0		OBC1 and OBC2 are turned on; the LED of OBC2 blinks and shows that it is active
300	Rx: "Deploy antenna"	After 5 minutes, the antenna is automatically deployed (simulated by this message)
463	Tx: 0x21 0x0000 0x7C	Command to simulate failure of OBC2
463		OBC2 stops any activity, its LED stops blinking
464		OBC1 starts functioning, its LED is now blinking
502	Tx: 0x21 0x0000 0x21	We set the value of the EPS2 in

		currentStatus to ready (it was set to disabled by default; the EPS2 can now be activated automatically by the OBC)
514		We manually set the input of the ADC (by turning the potentiometer on the test board) over the predefined threshold (defined of value 512) for activating the EPS2
514		The LED simulating the EPS2 is turned on
531	Tx: 0x21 0x028A 0x66	We simulate a latchup on EPS2 (the FAULT_EPS2 signal is set by software for the simulation) at time 650 (0x28A)
650		The LED of the EPS2 is turned off
652		The LED of the EPS2 is turned on (this power cycle is designed to recover from the latchup)
649	Tx: 0x21 0x0000 0x6C	We ask to retrieve the log
660	Rx:	The format of each record is: “time;eventType;eventParameters”; here follows a textual explanation of each received record
	0;1;1	Time: 0 seconds Event: "OBC started" Parameter: "OBC = OBC2"
	300;2;0	Time: 300 seconds Event: "Antenna deployed" Parameter: none
	464;1;0	Time: 464 seconds Event: "OBC started" Parameter: "OBC = OBC1" Note that OBC1 does not overwrite records previously stored by OBC2

---

502;3;12	Time: 300 seconds Event: "EPS2 status changed" Parameter: "current status = ready"
514;3;3	Time: 514 seconds Event: "EPS2 status changed" Parameter: "current status = on"
531;4;0	Time: 531 seconds Event: "EPS2 fault detected" Parameter: none
650;4;47	Time: 650 seconds Event: "EPS2 status changed" Parameter: "current status = power cycle in progress"
652;1;0	Time: 652 seconds Event: "EPS2 status changed" Parameter: "new status = on"
0;0;0 0;0;0 0;0;0 (...)	Empty records
698 Tx: 0x21 0x0000 0x6D	We ask to retrieve all stored measurements
698 Rx:	The format of each record is: ‘‘V_BATTERY;T_BATTERY’’
314;23 314;23 314;23 (...)	Identical records
314;24 314;24 429;24	
680;24	Approaching time 514, we start turning the potmeter that simulates the voltage of the battery At time 514, the value of V_BATTERY crosses the threshold of 512 for activating EPS2
680;24	

---

680;24

680;24

(...)

Identical records

680;24

0;0

Empty records

0;0

0;0

(...)