

---

## Development of an on-board computer for a nanosatellite

**Auteur** : Horbach, Amadis

**Promoteur(s)** : Redouté, Jean-Michel

**Faculté** : Faculté des Sciences appliquées

**Diplôme** : Master : ingénieur civil électricien, à finalité spécialisée en "signal processing and intelligent robotics"

**Année académique** : 2020-2021

**URI/URL** : <http://hdl.handle.net/2268.2/11659>

---

*Avertissement à l'attention des usagers :*

*Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.*

*Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.*

---



Master's thesis carried out to obtain the degree of Master of Science in Electrical Engineering by Horbach Amadis

---

# Development of an on-board computer for a nanosatellite

---

University of Liège - School of Engineering and Computer Science

Supervisor: Jean-Michel Redouté

Academic year 2020-2021

# Acknowledgements

I would like to start by thanking Hervé Pierre. Firstly, for proposing this thesis to me. Then, for his follow-up and guidelines during the whole year. And finally, for his proofreading of this thesis. His help was really valuable to me. I would also like to thank Professor Jean-Michel Redouté for giving me and other students the chance to work on such an interesting project.

I would like to express my thanks to the Microsys team for the good atmosphere that always prevailed in the working room and their kind advice in case of need. I would especially like to thank Morgan with whom I was happy to share on the current OUFTI project.

I also would like to thank my friends for their support during this particularly strange and hard period we are all living. You have allowed me to get a bit of fresh air and to go through this work.

Finally, I want to thank my family for their constant support during my studies and particularly during this work. I would especially like to thank my mother, my sister and my uncle for their proofreading of this thesis.

# Abstract

The purpose of this master thesis is to develop the On-Board Computer (OBC) of the University educational CubeSat OUFTI. The OBC is the central part of a CubeSat allowing the control of all the other subsystems. Its development is therefore crucial and must be carried out effectively and efficiently. As this is not the first CubeSat developed by the university, the first thing to do is to analyze the previous ones in order to extract their strengths and more importantly their weaknesses. Thus, this work starts with an introduction of the concept of CubeSat followed by a brief analysis of the main elements that created difficulties in the previous CubeSat of the University.

Once the context of this thesis is introduced, the development of the electric circuit can be detailed. The choice of the components is logically based on the space conditions introduced in the first chapter. The most important part of the circuit is the micro-controller. Indeed, this is the component that will manage and monitor all the other components, whether they are on the OBC board or on one of the other boards in the CubeSat. It is thus essential that it works efficiently and without any failure. Then, the other components of the OBC can be selected and assembled in schematics. All the components are finally placed in a PCB to form the OBC's PCB.

When the hardware of the OBC is finished and its PCB is created, it is important to test that the OBC performs all its required tasks. To this end, a software has to be developed. This software implements a real time operating system architecture. The creation of the code implemented to test the correct functioning of the OBC is then detailed.

The final part of this thesis explores the various tests that have been performed along with the most important results obtained. These tests went from simple voltage measurements to communication with external devices. The tests allow a first validation of the OBC of the university CubeSat.

# List of acronyms

|               |   |
|---------------|---|
| <b>ACK</b>    | Acknowledge   |
| <b>ADC</b>    | Analog-to-Digital Converter                         |
| <b>ADCS</b>   | Attitude Determination and Control Subsystem        |
| <b>AFEC</b>   | Analog Front-End Controller                         |
| <b>ARM</b>    | Advanced RISC Machine                               |
| <b>COTS</b>   | Commercial Off The Shelf                            |
| <b>CPHA</b>   | Clock Phase   |
| <b>CPOL</b>   | Clock Polarity                                      |
| <b>CS</b>     | Chip Select   |
| <b>DDR</b>    | Double Data Rate                                    |
| <b>DOUT</b>   | Data Out  |
| <b>DRAM</b>   | Dynamic Random Access Memory                        |
| <b>DTCM</b>   | Data Tightly Coupled Memory                         |
| <b>ECC</b>    | Error Correction Code                               |
| <b>EEPROM</b> | Electrically Erasable Programmable Read-Only Memory |
| <b>EPROM</b>  | Erasable Programmable Read-Only Memory              |
| <b>EPS</b>    | Electrical Power System                             |
| <b>FRAM</b>   | Ferroelectrical Random Access Memory                |
| <b>GCR</b>    | Galactic Cosmic Rays                                |
| <b>HSMCI</b>  | High Speed Multimedia Card Interface                |
| <b>I2C</b>    | Inter-Integrated Circuit                            |
| <b>ICD</b>    | In-Circuit Debugger/Programmer                      |
| <b>IDE</b>    | Integrated Development Environment                  |
| <b>IO</b>     | Input/Output  |
| <b>ITCM</b>   | Instruction Tightly Coupled Memory                  |
| <b>LEO</b>    | Low Earth Orbit                                     |
| <b>LET</b>    | Linear Energy Transfer                              |
| <b>MCAN</b>   | Master Controller Area Networks                     |
| <b>MCU</b>    | Microcontroller Unit                                |
| <b>MISO</b>   | Master In Slave Out                                 |
| <b>MOSI</b>   | Master Out Slave In                                 |
| <b>MOSFET</b> | Metal-Oxide-Semiconductor Field-Effect Transistor   |
| <b>MSB</b>    | Most Significant Bit                                |
| <b>NASA</b>   | National Aeronautics and Space Administration       |
| <b>NCPHA</b>  | Not Clock Phase                                     |

|               |   |
|---------------|---|
| <b>NVIC</b>   | Nested Vector Interrupt Control                         |
| <b>OBC</b>    | On-Board Computer                                       |
| <b>OS</b>     | Operating System  |
| <b>PCB</b>    | Printed Circuit Board                                   |
| <b>PMC</b>    | Power Managment Controller                              |
| <b>PROM</b>   | Programmable Read-Only Memory                           |
| <b>PZT</b>    | Lead zirconate titanate                                 |
| <b>RAM</b>    | Random Access Memory                                    |
| <b>RMAP</b>   | Remote Memory Access Protocol                           |
| <b>ROM</b>    | Read-Only Memory  |
| <b>RSD</b>    | Redundant Signed Digit                                  |
| <b>RSWDT</b>  | Reinforced Safety Watchdog Timer                        |
| <b>RTC</b>    | Real Time Clock   |
| <b>RTOS</b>   | Real Time Operating System                              |
| <b>RTT</b>    | Real-Time Timer   |
| <b>SAA</b>    | South Atlantic Anomaly                                  |
| <b>SCL</b>    | Serial Clock Line                                       |
| <b>SDA</b>    | Serial Data   |
| <b>SDR</b>    | Single Data Rate  |
| <b>SDRAM</b>  | Synchronous Dynamic Random Access Memory                |
| <b>SDRAMC</b> | Synchronous Dynamic Random Access Memory Controller     |
| <b>SEE</b>    | Single-Event Effects                                    |
| <b>SEL</b>    | Single-Event Latchup                                    |
| <b>SEU</b>    | Single-Event Upset                                      |
| <b>SLC</b>    | Single-Level Cell                                       |
| <b>SPCK</b>   | Serial Peripheral Interface Clock                       |
| <b>SPE</b>    | Solar Particle Events                                   |
| <b>SPI</b>    | Serial Peripheral Interface                             |
| <b>SRAM</b>   | Static Random Access Memory                             |
| <b>TCM</b>    | Tightly Coupled Memory                                  |
| <b>TID</b>    | Total Ionizing Dose                                     |
| <b>TWI</b>    | Two Wire Interface                                      |
| <b>UART</b>   | Universal Asynchronous Receiver/Transmitter             |
| <b>USART</b>  | Universal Synchronous/Asynchronous Reveiver/Transmitter |
| <b>VREFP</b>  | Positive Voltage Reference                              |
| <b>WDT</b>    | Watch Dog Timer   |
| <b>WREN</b>   | Write Enable  |

# Contents

|   |           |
|---|-----------|
| <b>Acknowledgements</b>                         | <b>2</b>  |
| <b>Abstract</b>                                 | <b>3</b>  |
| <b>List of acronyms</b>                         | <b>4</b>  |
| <b>Contents</b>                                 | <b>6</b>  |
| <b>1 Introduction</b>                           | <b>8</b>  |
| 1.1 Project's presentation . . . . .            | 8         |
| 1.2 CubeSat history and architecture . . . . .  | 9         |
| 1.2.1 History . . . . .                         | 9         |
| 1.2.2 Architecture . . . . .                    | 9         |
| 1.2.3 On-Board Computer . . . . .               | 11        |
| 1.3 Space conditions . . . . .                  | 12        |
| 1.3.1 Temperature . . . . .                     | 12        |
| 1.3.2 Radiation . . . . .                       | 12        |
| 1.4 Power management . . . . .                  | 14        |
| <b>2 Hardware design</b>                        | <b>15</b> |
| 2.1 Choice of the microcontroller . . . . .     | 15        |
| 2.1.1 SAMV71 . . . . .                          | 17        |
| 2.2 Selection of the other components . . . . . | 22        |
| 2.2.1 Memories . . . . .                        | 23        |
| 2.2.2 Oscillators . . . . .                     | 31        |
| 2.2.3 Connectors . . . . .                      | 31        |
| 2.2.4 Power monitoring . . . . .                | 33        |
| 2.3 Final PCB presentation . . . . .            | 36        |
| <b>3 Software implementation</b>                | <b>38</b> |
| 3.1 Software architecture . . . . .             | 38        |
| 3.2 MPLAB . . . . .                             | 39        |
| 3.2.1 Harmony v3 . . . . .                      | 39        |
| 3.3 Code implementation . . . . .               | 41        |
| 3.3.1 I2C . . . . .                             | 42        |
| 3.3.2 SPI . . . . .                             | 43        |
| 3.3.3 External I2C bus task . . . . .           | 45        |
| 3.3.4 External ADC (SPI) task . . . . .         | 46        |
| 3.3.5 FRAM (SPI) task . . . . .                 | 47        |

- 3.3.6 User task . . . . . 49
- 4 Hardware and Software testing 52**
  - 4.1 Development board . . . . . 52
  - 4.2 PCB compatible with the development board . . . . . 54
- 5 Conclusion 58**
- Appendices 60**
  - A Final PCB Schematics 61**
  - B Test PCB Schematics 67**
  - C Codes Git link 71**
- References 73**



# Chapter 1

## Introduction

### 1.1 Project's presentation

The University of Liège (ULiège) has a well known history with nanosatellites. It all started in September 2007 when ULiège and the Higher Education Institution of the Province of Liège (HEPL) decided to launch a project aimed at developing a nanosatellite. After several years of development, the nanosatellite named OUFTI-1 was launched and successfully reached its orbit on 25th April 2016. Unfortunately, the nanosatellite encountered a failure quickly after. Even if there is no proof of what caused the failure, radiation is thought to be responsible. In 2017, ULiège and HEPL decided to develop a second nanosatellite named OUFTI-2. However, mainly due to the high power consumption of its microcontroller (MCU), this one was never launched.

In 2020, ULiège and HEPL decided to start a new OUFTI project. This one is meant to be composed of four main sections: (i) an Electrical Power System (EPS), (ii) a telecommunication system, (iii) a scientific payload, and (iv) an On-Board Computer (OBC) controlling the entire nanosatellite. The purpose of this master thesis is to develop the OBC of this nanosatellite. The work is divided into two main tasks.

The first one concerns the hardware of the OBC. It consists in selecting the components and assembling them to produce the OBC schematics. Once the schematics are done, the conception of a Printed Circuit Board (PCB) is also needed. For this task, particular attention must be paid to the choice of components. Indeed, previous OUFTI experiments have shown that it is important to choose components that are (i) resistant to the harsh conditions encountered in space, (ii) efficient in terms of power consumption.

The second task consists in a series of tests to check the proper functioning of some components. Some tests only require the reading of some voltages through external components such as voltmeters but most of the tests require a software. For that reason, a complete software has also to be realized. It is important for this software to be easily readable and well documented as the development of the nanosatellite will continue in the coming years.

## 1.2 CubeSat history and architecture

Traditional satellites are huge, heavy, and expensive. Thanks to the invention of microcontrollers, which integrate multiple components, and the reduction of Printed Circuit Board size, engineers were able to develop a new class of satellites, with lower size and weight, and therefore, with lower power consumption: the nanosatellites.

### 1.2.1 History

CubeSats are nanosatellites with specific characteristics. They are the result of a collaborative work carried out by two university professors, Jordi Puig-Suari, a professor at California Polytechnic State University (Cal Poly), and Bob Twiggs, a professor at Stanford University's Space Systems Development Laboratory (SSDL) [1]. Their aim was to make it possible for engineering students within their universities to work on space programs and so improve their training. To reach their objective, it was necessary to reduce considerably the size and cost of satellites. In 1999, they succeeded with the conception of the first CubeSat. And in 2003, the first CubeSat was launched.

CubeSats were first used within universities as educational tools. But now they are more and more popular in the satellite industry. As they are much smaller and considerably less expensive than conventional satellites, they give affordable access to space and offer a perfect solution for several industrial applications. Therefore, CubeSats are now developed by Government organizations as well as private firms worldwide. What makes CubeSats so popular is their standard size and modularity, which not only reduces production costs but also makes the launch process very efficient, as will be detailed below. In addition to this, as the production time is also greatly reduced, if something goes wrong, a whole mission does not have to be stopped for years, as is the case with traditional satellites.

### 1.2.2 Architecture

In 1999, the CubeSat standard was defined. The basic unit, called 1U, is a 10 cm cube with a maximum mass of 1,33 kg [1]. Thanks to its modularity, different units, based on the standard CubeSat unit (1U), can be combined and so form a larger structure (2U, 3U,...).

Standard dimensions make mass production possible. The use of commercial off-the-shelf (COTS) components for the structure and electronics of CubeSats considerably reduces both costs and production time. The launch of Cubesats is also greatly facilitated by their reduced volume, light weight, and standard dimensions. As a result, they can be installed in a specific container and launched from commercial space flights, which enables more frequent launches and at reduced costs since they can be shared.

As it can be seen in Figure 1.2.1, the main components of a CubeSat are the following:

- **Payload:** it is the reason why the satellite is designed.
- **Attitude Determination and Control System (ADCS):** it controls the orientation of the satellite.
- **On-Board Computer (OBC):** it is the central part of the CubeSat allowing the control of all the other subsystems.
- **Communication system:** it handles the external communications of the CubeSat.
- **Electrical Power System (EPS):** it is responsible for the adequate distribution of power to all the subsystems.
- **Solar panels:** they generate power and reload the EPS batteries.
- **Antenna:** it is used to send and receive CubeSat's communications.

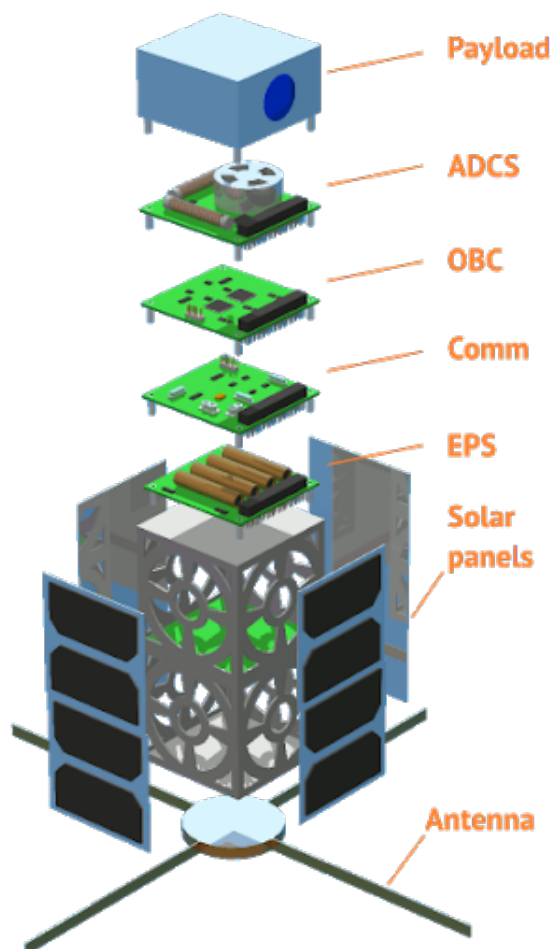


Figure 1.2.1: Representation of the different parts of a CubeSat.

SOURCE : <http://spacebillboard.com/mission/>

### 1.2.3 On-Board Computer

This thesis aims at developing the on-board computer of a CubeSat. The characteristics and role of the OBC will therefore be detailed in this section. During the mission, data will be collected through sensors and be processed. Communication with the ground station will also happen. Although they usually involve other sub-systems, all these operations are linked at some point to the OBC. An example is the voltage of the batteries which is frequently monitored by the EPS board and whose values are then stored in the OBC memory. The OBC can thus be represented at the center of a CubeSat diagram. This can be seen in Figure 1.2.2 with the subsystems introduced in the previous section.

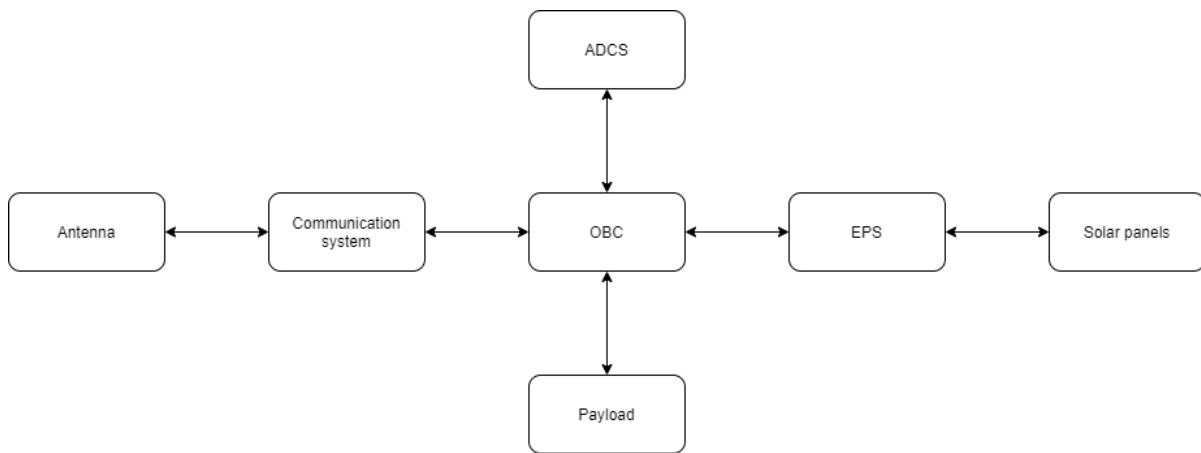


Figure 1.2.2: Block diagram representing the interconnection between the main CubeSat subsystems.

Based on what has been explained above the main components required by an OBC will now be explored. First, a microcontroller is needed. The MCU can be viewed as the brain of the OBC and more generally of the CubeSat. It will handle the different communications happening inside the CubeSat and will process the data. The communications and data processing will be managed through a software that needs to be implemented. Then, connectors are needed to allow communication between the different subsystems and the OBC. It is also important to ensure that the power signals coming from the EPS board are well regulated in order to avoid a system failure. A monitoring of the power consumed by the OBC is also required as power is limited inside the CubeSat. Finally, the OBC will need to have memories in order to store data and the software. Different memory types will be required. Indeed, some information needs to be stored for a long time while other information will need to be processed quickly using volatile memory.

The components selected for the realization of the OBC of this project will be described in the next Chapter. Then, Chapter 3 will detail the software implemented in order to ensure that the OBC performs its operating tasks. Finally, Chapter 4 will present the tests that have been performed to ensure the good working of the OBC. But before that, a presentation of the space conditions that the CubeSat and thus the OBC will encounter is required in order to justify some of the choices made later.

## 1.3 Space conditions

CubeSats operate in Low Earth Orbit (LEO), which means at an altitude comprised between 160 km and 2.000 km. It is therefore important to take into account the particular space environment in LEO while choosing the components used in the OBC. Indeed, the temperature extremes and radiation that electronic systems will be subjected to may cause serious disturbances and failures [2].

In Low Earth Orbit, CubeSats will be exposed to temperature extremes, ranging from  $-55^{\circ}$  to  $+125^{\circ}\text{C}$  in a short period of time, putting a strain on all materials. This could affect the operational functionality of electronic components and limit their lifetime. It is thus essential to use components specially designed to withstand such a range of temperature extremes. Moreover, satellites in LEO are particularly affected by space radiation that can pass through a satellite and consequently damage electronic devices. Shielding can help to protect the components from radiation but it adds weight. Another solution is the use of radiation hardened or radiation tolerant components.

Space conditions in LEO will now be explained in more detail.

### 1.3.1 Temperature

Satellites orbiting the Earth move in and out of the sunlight. Therefore, they are submitted to extreme swings of temperature depending on whether they are facing the Sun or in the shadow. When they are in view of the Sun, as they are not protected by the Earth's atmosphere, they are subjected to solar radiation, and thus to very high temperatures. Thermal effects are also enhanced by the lack of convection in space. On the reverse, when satellites pass into the shadow of the Earth, the absence of atmosphere leads to rapid cooling as a result of the lack of energy being absorbed and of the thermal energy released by the satellites. It is therefore important to use electronic equipment that can withstand these temperature cycles.

### 1.3.2 Radiation

There are three main sources of ionizing radiation that can affect CubeSats and their electronic components. They are depicted in Figure 1.3.1.

Galactic Cosmic Rays (GCR) are high energy protons and heavy ions which are constantly bombarding the Earth. These particles originate outside the solar system. They are made up of atomic nuclei that have been stripped of their surrounding electrons and that are moving almost at the speed of light. These high-energy particles can alter components in electronic integrated circuits.

Energetic solar particles emanate from solar flares and coronal mass ejections. These events occur from time to time on the surface of the Sun. They are unpredictable. During these giant explosions, a large number of particles (protons, electrons, X-rays, and gamma rays) are released into space and can cause light to severe damage to electrical

components. These are known as Solar Particle Events (SPE).

Trapped particles are highly charged particles, emanating mainly from the Solar Wind. They are trapped by the Earth's magnetic field in the Van Allen belts. Over the South Atlantic Ocean, there is a magnetic field anomaly. This region of weaker magnetism is called the South Atlantic Anomaly (SAA). Because the axis of the magnetic field is slightly offset from the one of the Earth, the inner belt is closer to the Earth's surface in this area. This weak point allows charged particles, mainly protons and electrons, to fall closer to the ground and reach satellites operating in LEO. As a result, when they pass through this zone, as CubeSats often do, they are exposed to particularly intense radiation.

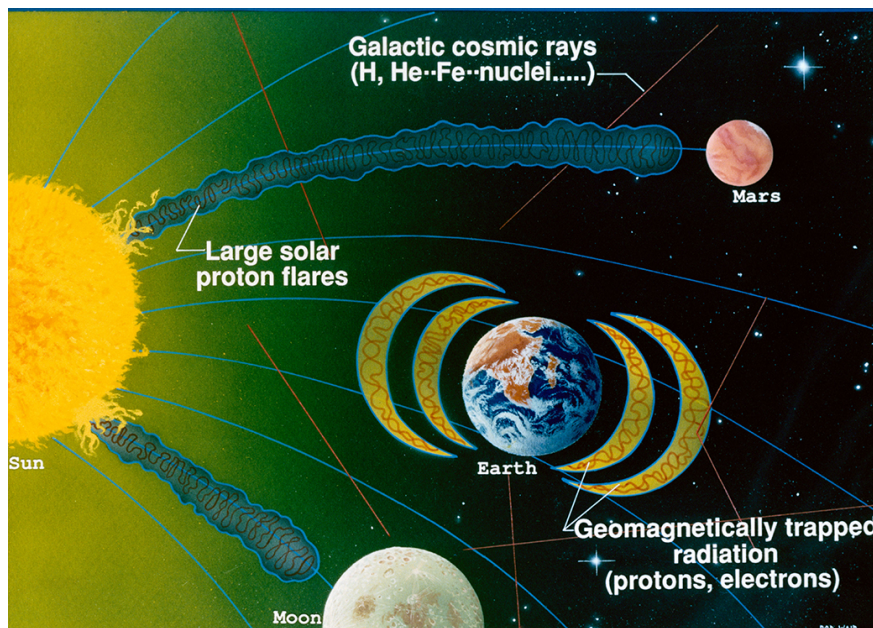


Figure 1.3.1: Main sources of space radiation in LEO.

SOURCE : [https://www.nasa.gov/audience/foreducators/stem-on-station/learning\\_launchers\\_radiation](https://www.nasa.gov/audience/foreducators/stem-on-station/learning_launchers_radiation)

Electronic devices are impacted by space radiation in two different ways. Indeed, radiation effects can be either cumulative or due to a single event. That will now be described.

**Total Ionizing Dose (TID):** Electronic devices are subject to the long-term effects of radiation and are being deteriorated by the charges they have accumulated.

**Single-Event Effects (SEE):** A single charged particle hits and ionizes the material, causing a current pulse that can damage the component. The effects can be reversible (soft errors, which can be mitigated by a reset or an erase operation, for instance) or non-reversible (hard error).

There are different types of SEEs, depending on the location of the hit and its intensity.

The two more common are:

- Single-Event Upset (SEU): In this case an ionizing particle from the radiation reaches a sensitive area of the circuit such as the memory and creates a change of state (bitflips).
- Single-Event Latchup (SEL): An ionizing particle from the radiation creates an overcurrent within the device which can cause irreparable damage if not directly addressed.

## 1.4 Power management

Satellites get all their power from solar panels. When exposed to the Sun, the solar arrays transfer energy to the EPS. Part of this energy is used for operations and part for recharging the batteries. The batteries are required to operate during eclipses, when the CubeSat is in the shadow of the Earth, or to deal with power peaks that are beyond what the solar cells can provide. Due to their small size, CubeSats have a reduced surface area for solar arrays and strict constraints on the subsystem components, including the EPS, that can be installed. As CubeSats are increasingly used for more sophisticated applications, the power needs are constantly growing. Even with the advent of deployable solar panels and more efficient batteries, moderate use and correct management of power are thus crucial. A negative power budget could lead to the failure of a CubeSat mission. In short, energy is precious on a Cubesat. Particular attention must therefore be paid to the choice of low-power components.

# Chapter 2

## Hardware design

The development of the on-board computer starts with the creation of its PCB. To this end, the components constituting the OBC have to be selected and the schematics have to be realized. This chapter will present this hardware part of the OBC's conception. It will start with an explanation about the thinking that was made in order to choose the microcontroller. Then, the chosen MCU will be investigated and some of its characteristics will be detailed. After having presented the MCU, the other components constituting the OBC will be explored. This exploration will start with a review of all the different memory components that have been placed on the OBC. Then, the connectors and oscillators that have been used will be detailed. The last components that will be presented in this chapter are those used to monitor the power of the OBC. Finally, the chapter will end with a presentation of the final OBC's PCB.

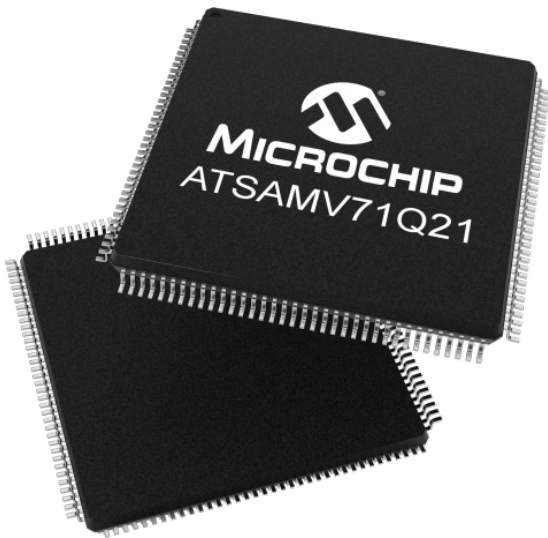
### 2.1 Choice of the microcontroller

The microcontroller is the central part of the OBC. It is therefore important to choose it with particular attention. Indeed, a malfunction of the MCU will lead to the failure of the OBC, and thus, of the whole nanosatellite. The previous chapter introduced the main difficulties encountered in space. Research has been conducted in order to find a MCU consuming relatively low power and being able to survive in the harsh space environment. Two main challenges were encountered during this procedure. First, radiation-resistant MCUs generally consume much more power than more basic MCUs. Second, the price of MCUs capable of surviving in space is much higher than that of MCUs designed for use on Earth. The second point is particularly problematic for the development of a university CubeSat. Indeed, the budget is not unlimited and some tests must be done this year while the OBC will still evolve before being launched one day. It is thus impossible to spend hundreds of euros in order to get a MCU to test the OBC's PCB this year and to have to buy a new one next year for the updated PCB.

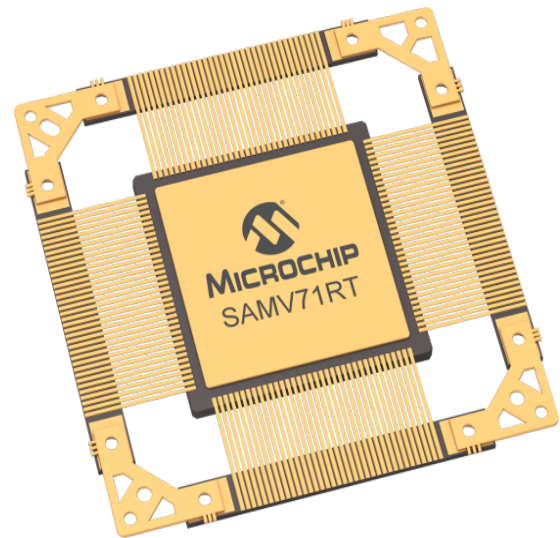
For the reasons mentioned above, the choice has been made to develop the OBC around the SAMV71 MCUs. This choice is motivated by the fact that three different versions of this MCU exist [3]. They are shown in Figure 2.1.1. *Microchip* has developed the SAMV71 to be Arm<sup>®</sup>-based MCUs able to operate properly in space. The standard version, the ATSAMV71Q21, is a COTS device and costs only a few dozen euros. The second one, the SAMV71Q21RT, is the radiation-tolerant version of the ATSAMV71Q21. The third one, the SAMRH71, is the radiation-hardened MCU based on the standard



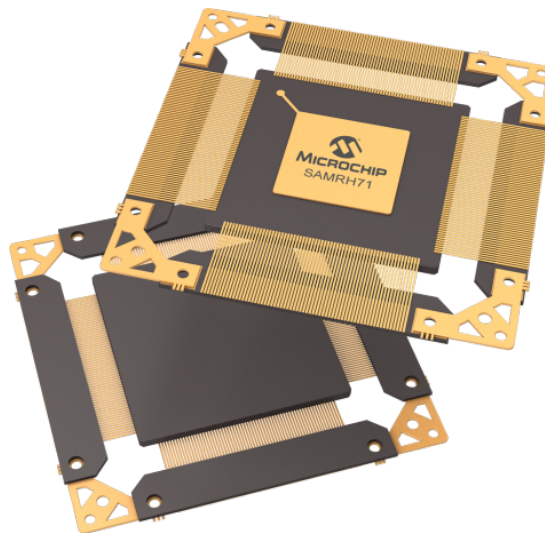
one. It is also worth noting that a development board made around the standard MCU version is available.



(a) ATSAMV71Q21, the standard version of the SAMV71



(b) ATSAMV71Q21RT, the radiation-tolerant version of the SAMV71



(c) SAMRH71, the radiation-hardened version of the SAMV71

Figure 2.1.1: The three versions of the Arm<sup>®</sup>-based SAMV71 developed by *Microchip*.

#### SOURCES:

- (a) <https://www.microchip.com/wwwproducts/en/ATSAMV71Q21>
- (b) <https://www.microchip.com/wwwproducts/en/SAMV71Q21RT>
- (c) <https://www.microchip.com/wwwproducts/en/SAMRH71>

The development board has been used to implement the software and to realize the tests performed during this thesis. For the future tests, the standard version will be placed in the designed OBC's PCB. Finally, the radiation-tolerant version will replace the standard version before the launch of the nanosatellite. The transition from the standard version to the radiation-tolerant version is straightforward since the masks of the latter are the same as those of the former. Some little modifications should be made

to adapt the software to the radiation-hardened version. However, they are negligible compared to the gain obtained from developing the software on the standard version. Moreover, the radiation-tolerant MCU will be the one used when the nanosatellite will be launched. Firstly, because it is well adapted to operate in LEO, which is precisely where the CubeSats operate. Secondly, because it is way cheaper than the radiation-hardened version. The characteristics of the three versions will now be detailed in the next section.

### 2.1.1 SAMV71

As already stated, one of the reasons for which the SAMV71 has been chosen is because it exists in three different versions. Before presenting the space characteristics of the radiation-tolerant and radiation-hardened versions, the main characteristics of the standard version will be described. Figure 2.1.2 represents the block diagram of the ATSAMV71Q21 microcontroller [4]. This gives an overview of the main functions of the MCU in one picture.

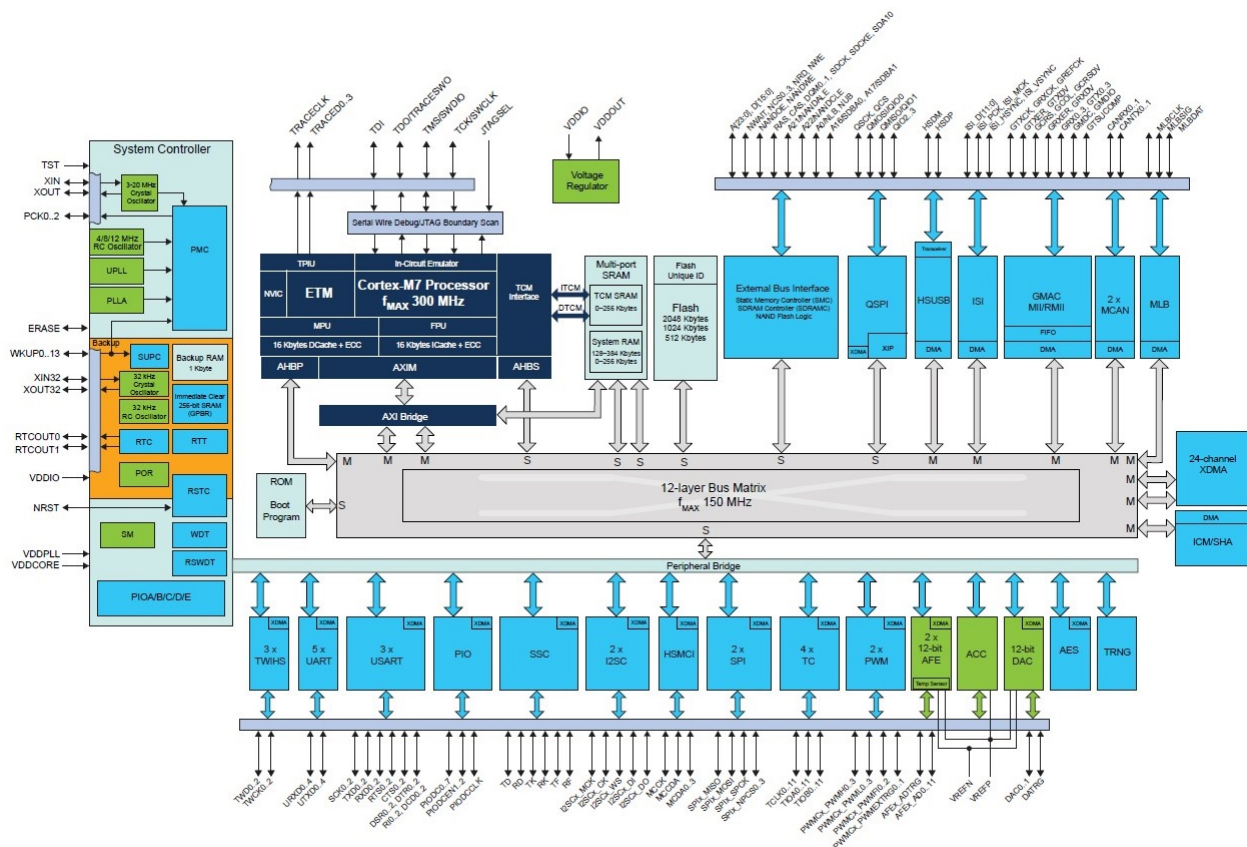


Figure 2.1.2: Block diagram of the ATSAMV71Q21 [4].

#### 2.1.1.1 Core

The main component of a microcontroller is its core. The ATSAMV71Q21 contains an ARM<sup>®</sup> Cortex<sup>®</sup>-M7 core processor running at up to 300 MHz [4]. In addition to being very powerful, this processor offers great flexibility and consumes relatively low power, which makes it ideal for this application. Moreover, it supports Real Time Operating System (RTOS) and there exists a lot of third party tools working with it. One of the main

properties of the Cortex-M7 is the presence of a six-stage, in-order super-scalar pipeline that enables dual-issue processing of instructions. This feature allows the processor to run two instructions in parallel. Another interesting point is that the latency of the Cortex-M7 processor is reduced thanks to the fact that memory accesses are interleaved with computation.

### 2.1.1.2 Memories

The ATSAMV71Q21 has different memories. They will now be detailed in turn [4].

#### Internal SRAM

The MCU contains 384 Kbytes of high speed Static Random Access Memory (SRAM). Four priority levels are available to improve processor efficiency without compromising the high-priority latency-critical requests. Some features make possible to optimize the SRAM performances. First, it is equipped with a Multi-Port SRAM with four ports to ensure optimal bandwidth and latency. Then, the MCU is equipped with an SRAM controller. This one allows the minimization of latency by giving priority to the most urgent requests without neglecting the less urgent ones. Indeed, the latter are executed at the latest during the next cycle.

#### Tightly Coupled Memory (TCM) Interface

A tightly coupled memory (TCM) is integrated into the MCU. Unlike other memories that are accessed at bus speed, TCM operates at processor speed. TCM is therefore used to store critical parts of the code that require the fastest possible processing. It is divided into two parts. The instruction tightly coupled memory (ITCM) is a single 64-bit interface that allows the processor to fetch two 32-bit instructions in a single access and thus use the dual issue property of the pipeline. The data tightly coupled memory (DTCM) consists of two interleaved 32-bit interfaces that makes possible the optimization of concurrent accesses. Finally, the software allows both the configuration of the TCM in 4 different modes and the activation or deactivation of the ITCM and DTCM.

#### Internal ROM

The SAMV71Q21 embeds an internal Read Only Memory (ROM). It is used for the SAM Boot Assistant, In-Application Programming and Fast Flash Programming Interface functions.

#### Backup SRAM

The MCU embeds 1 Kbytes of backup SRAM. This one is only accessible in 32-bit words. A power switch ensures that the contents of the backup SRAM are retained. Indeed, in normal mode, it is powered by VDDCORE. However, in backup mode, when VDDCORE is turned off, the power switch allows the backup SRAM to be powered by VDDIO. To save power, the power switch can be turned off.

#### Flash Memories

The SAMV71 embeds 2084 Kbytes of internal Flash. It is organized in sectors of 128 Kbytes in size. Each sector is organized in pages of 512 bytes. Figure 2.1.3 represents the general organization of the internal Flash. It is interesting to note that the first sector is divided into three smaller sectors, two 8 Kbyte sectors, and one 112 Kbyte sector.

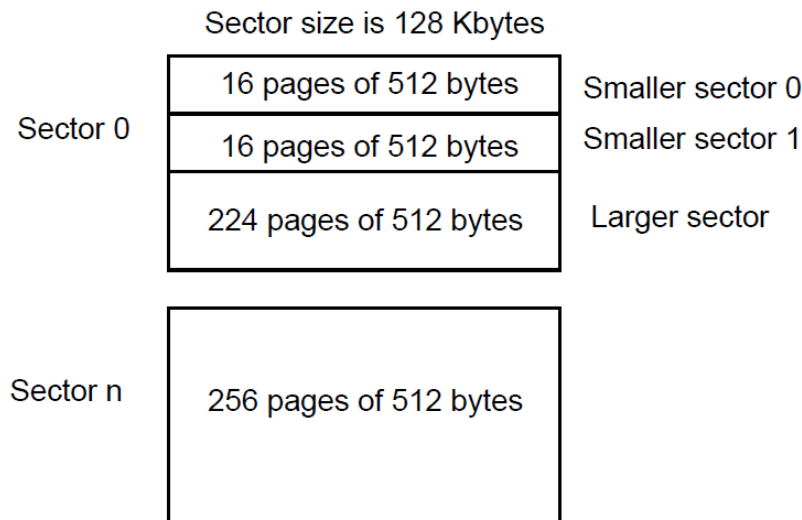


Figure 2.1.3: Flash Sector Organization [4].

The MCU has an Enhanced Embedded Flash Controller that manages the Flash memory. Thanks to it, it is possible to read the Flash and to write the write buffer. It also provides the interface of the Flash block. It uses a complete set of commands to manage the programming, erasing, locking and unlocking sequences of the Flash.

## External Memories

Finally, the MCU includes an Synchronous Dynamic Random Access Memory Controller (SDRAMC). This controller increases the memory capacity by allowing an external 16-bit DRAM device to be connected. The MCU also has an external bus interface giving the possibility to use external memories. All external memories that have been used will be presented in the next section.

### 2.1.1.3 Peripherals

The ATSAMV71Q21 has many peripherals. Therefore, only the most useful ones are detailed below [4].

## The Two-wire Interface

There are three different two-wire interfaces (TWI) on the SAMV71Q21. The TWI is a two-wire bus used for data transfer that is based on a byte-oriented format. The two wires correspond to a clock line and a data line. TWI is I2C compatible and works with two-wire serial Electrically Erasable Programmable Read-Only Memory (EEPROM). An I2C bus has been implemented in the software, its operation will thus be explained in more detail in the following Chapter. TWI can operate at speeds up to 400 kbit/s in fast mode and up to 3.4 Mbit/s in high-speed slave mode only. It uses a configurable baud

rate generator to match the clock rates of a wide range of external components.

### Serial Peripheral Interfaces

Two Serial Peripheral Interfaces (SPI) bus are present on the MCU. It is a synchronous communication system requiring only 4 wires to perform data transfers with external devices. The SPI operating speed is generally way higher than the TWI one. As for the I2C bus, a complete description of the functioning of the SPI bus is presented in the next Chapter.

### Analog Front-End Controllers

The MCU contains two Analog Front-End Controllers (AFEC) that rely on an Analog Front-End cell containing various modules. It can be used to perform different types of data conversion either in differential or single-ended mode and it has a programmable gain stage. The conversion results range from 0V to VREFP which is the positive voltage reference. It has a standard resolution of 12-bit which can be extended up to 16-bit. A digital error correction circuit offers the possibility to reduce integral non-linearity and differential non-linearity errors. This circuit is based on the multi-bit redundant signed digit (RSD) algorithm. The Analog-to-Digital Converter (ADC) of the AFEC has been used in the software of the OBC and some clarifications about it will be given in the next Chapter.

### USART/UART

Three Universal Synchronous Asynchronous Receiver Transceiver (USART) and five Universal Asynchronous Receiver Transmitter (UART) are available on the MCU. The UART enables asynchronous serial communications while the USART enables synchronous and asynchronous serial communications. The remaining part of this explanation will focus on the UART since it is a more widespread system. UART is a serial device-to-device hardware communication protocol. Data is transferred bit by bit using only two wires for its transmitting and receiving ends. Those two wires are Transmitter (Tx) and Receiver (Rx). The data bus on the transmitter and receiver side transfers the data in parallel. However, as in the UART data is sent and received serially, bit by bit, on a single line, it must be converted at the beginning and end of the transfer. This is shown in Figure 2.1.4.

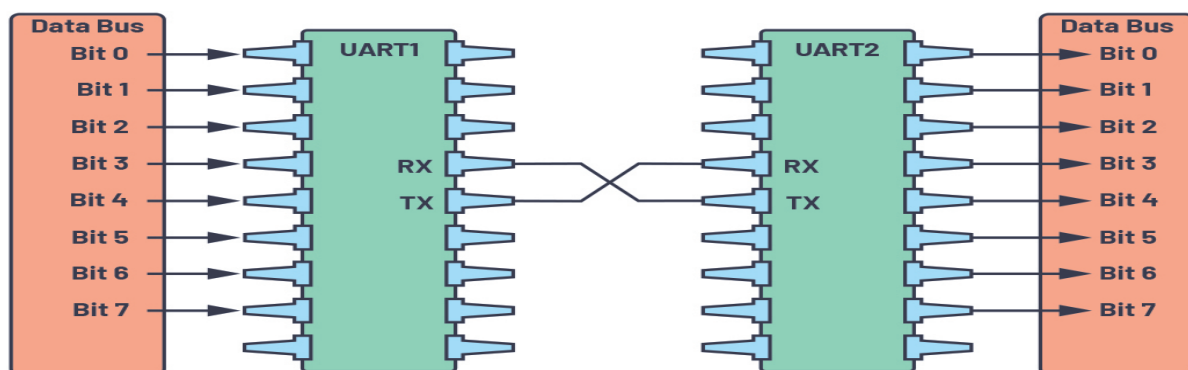


Figure 2.1.4: Representation of UART with data bus [5].

During the development of this thesis, this communication system has only been used as a debugging tool to print messages on the terminal. However, as it is a very useful communication system it will be used in future works based on this one, for example, to communicate with the communication board.

### **Controller Area Network**

Two Master Controller Area Networks (MCAN) can be used on the MCU. CAN is a communication bus standard that allows communication between multiple devices without a host computer. It is a message-based protocol that transmits data sequentially. When multiple devices attempt to transmit at the same time, only the device with the highest priority continues to transmit. Eventually, all devices connected to the bus receive the data, even the sending device.

### **Real-Time Clock**

The Real-Time Clock (RTC) device is developed for very low power consumption. A crystal oscillator provides an accurate 32.768 kHz external clock that allows the RTC to operate perfectly. The oscillator works when the power consumption is critical and the MCU is in power saving mode. The choice of the crystal must therefore be made taking into account its power consumption as well as the effects of temperature on its oscillation frequency. An interesting feature of the RTC is the possibility it offers to program a periodic interrupt. Finally, it also has a complete clock with an alarm and a Gregorian or Persian calendar.

### **Real-Time Timer**

The Real-Time Timer (RTT) is built around a 32-bit counter. It is used to generate a periodic interrupt when the user-programmed value in the counter is reached. In its standard configuration, the RTT uses the 16-bit programmable prescaler from the 32 kHz slow clock. However, it can also be configured to operate with the 1Hz RTC signal to take advantage of the 1Hz calibrated clock. Finally, when it is only necessary to count elapsed seconds, it is possible to disable the slow clock source to reduce power consumption.

### **Power Management Controller**

As explained in the Introduction, power is limited within the CubeSat. The Power Management Controller (PMC) optimizes power consumption by monitoring all system and user peripheral clocks. The PMC can actually enable and disable the clock inputs of both the devices connected to the MCU and the Cortex-M7 processor. The MCU is equipped with two oscillators that can act as clock signals, a slow RC oscillator, and a 32.768 kHz crystal oscillator. To optimize power consumption only one of the two oscillators is used while the other is disabled. By default, at startup, the main RC oscillator operating at 12 MHz is enabled. Finally, one of the main interesting features of the PMC is that it can detect failures of the main crystal oscillator.



#### 2.1.1.4 Other characteristics

The ATSAMV71Q21 possesses some very interesting characteristics that haven't been detailed yet. First, it has an embedded voltage regulator for single-supply operation [4]. In other terms, that means that the microcontroller only needs a single supply voltage of 3.3V. The embedded voltage regulator will convert this 3.3 V tension to a 1.23V tension for the parts of the microcontroller needing it.

As far as reliability is concerned, the MCU features a Power-on-Reset, a Brown-out Detector, a Watchdog Timer (WDT) as well as a Reinforced Safety Watchdog Timer (RSWDT) working independently from the WDT for increased safety. The last two mentioned were used in this project and more details about them will be given in the next chapter.

Finally, it is worth noting that the ATSAMV71Q21 is composed of 114 I/O lines with external interrupt capability.

#### 2.1.1.5 Space characteristics

The second version of the microcontroller, the SAMV71Q21RT [6] is the radiation-tolerant version of the SAMV71. This version is perfectly suited for CubeSats as they operate in LEO. It should therefore be the version of the MCU used in the final design of the OBC that will be sent to space. The first improvement achieved by this device is that it can work in temperatures ranging from -55 to 125°C while the operating temperatures of the ATSAMV71Q21 goe from -40 to 105°C. Besides being able to operate in a wider temperature range than the standard version, the radiation-tolerant version has the following characteristics in terms of radiation tolerance:

- It ensures an accumulated TID of 30Krad (Si) with latch-up immunity;
- It is nondestructive against heavy ions;
- It is completely immune to SEL below a Linear Energy Transfer (LET) Threshold of 60 MeV.cm2 /mg at 125°C.

Finally, the third version, the SAMRH71 [7] is the radiation hardened version of the SAMV71. Even if this version is more suited for deep space applications, its main characteristics will be given. It operates in the same temperature range as the radiation-tolerant version and has the following radiation performances:

- It ensures an accumulated TID of more than 100 Krad (Si);
- It ensures no SEU below an LET Threshold of 62.5 MeV.cm2/mg at 125°C;
- It includes SpaceWire interface with two SpaceWire ports with Integrated RMAP support and embedded SpaceWire router.

## 2.2 Selection of the other components

Now that the microcontroller of the OBC has been presented, the other components required by the OBC will be explored. These include the external components required

by the MCU in order to properly operate as well as some devices used to perform specific tasks. Apart from the MCU, the memories are the most important components of the OBC. Therefore, this point will be examined in detail.

### 2.2.1 Memories

The memories are naturally the first thing investigated after having chosen the MCU. The OBC is the place where everything is stored in a CubeSat. There are two main categories of memories. Volatile memories and non-volatile memories.

The basic difference between these two types of memories is that volatile memories lose their data when they are no longer powered, whereas the data in non-volatile memories is retained even without power [8]. Volatile memories may thus seem useless compared to non-volatile memories. However, non-volatile memories are slower and have a higher cost per bit compared to non-volatile memories. Moreover, volatile memories are more wear tolerant than non-volatile memories. Volatile memories therefore have a great impact on system performances while non-volatile memories have a great impact on the storage capacity of the system.

It is important that the OBC contains both types of memory. It needs non-volatile memories to save data that must be kept for a long time. And, since a lot of computing is done within the OBC, it is essential that it also contains volatile memory for faster and more frequent access.

The different types of technologies existing in these two categories will now be investigated. Volatile memories will be studied first. Then the component chosen in this category will be described. After that, the non-volatile memory systems will be detailed. Finally, the three non-volatile memory components selected for the OBC will be presented.

#### 2.2.1.1 Volatile Memory

As a reminder, volatile memory is the type of memory that loses its data when it is not powered. It is used to store temporary data in order to increase the computation speed and to reduce the workload of non-volatile memories and thus preserve them. Volatile memory can be separated into two main categories: Static Random Access Memory and Dynamic Random Access Memory (DRAM).

##### SRAM

A typical SRAM memory cell consists of six transistors [9]. Four of these are configured as two cross-coupled inverters. This configuration works like flip-flops and provides the two logical states "0" and "1". The other two transistors are used for access monitoring of the memory cell during read and write operations.

##### DRAM

DRAM memory cells, on the other hand, require only a single transistor combined with a capacitor to maintain the state of the cell [9]. The "1" state happens when the capacitor is charged, while the "0" state occurs when the capacitor is discharged. However,



due to charge leakage, the capacitor will discharge itself. For this reason, it is necessary to refresh periodically the cells of DRAM memories. This refreshing process ensures that the state of the capacitor is not lost. The term dynamic comes from the need to refresh, which SRAM does not have. In order to retain their data, DRAM must therefore be both powered and refreshed periodically.

### SDRAM

DRAM operates asynchronously. This means that it processes the information that arrives one by one. Synchronous Dynamic Random Access Memory (SDRAM) constitutes an improvement on DRAM [9]. SDRAM is synchronized to the MCU clock, which makes it much more efficient. Because of its synchronous nature, SDRAM can operate in more complex ways than DRAM. Indeed, it is able to keep two sets of memory addresses opened at the same time. As a result, while DRAM must close one address bank before opening the next, SDRAM can have two address banks opened at the same time. Therefore SDRAM is able to prepare the next instruction while it is performing the current one. It is worth noting that this feature of SDRAM is called pipelining. This characteristic considerably reduces delays and therefore SDRAM is much faster than DRAM.

### Comparison

Now that the different volatile memory technologies have been introduced they need to be compared in order to motivate the choice made. This is done in Table 2.2.1.

| Feature           | SRAM   | SDRAM   |
|-------------------|--|---|
| Cost              | More expensive as they require 6 transistors               | Cheaper as they only require 1 transistor and 1 capacitor |
| Speed             | Faster as they don't need to refresh the cells             | Slower because of the periodic refresh of the cells       |
| Density           | Low (6 transistors per bit)                                | High (1 transistor per bit)                               |
| Capacity          | Lower as fewer cells can fit in the same space             | Larger as more cells can fit in the same space            |
| Power consumption | Lower because they don't need to be periodically refreshed | Larger due to the need to periodically refresh the cells  |

Table 2.2.1: Comparison between the main features of SRAM and SDRAM.

To conclude this comparison, it should be noted that none of these technologies are radiation-tolerant. However, as they store data only for a short period, radiation does usually not have enough time to corrupt their data.

### Final choice

Due to its lower cost and higher capacity, it was decided to select an SDRAM as volatile memory for the OBC. Once the technology is selected, it is still necessary to choose between different types of systems.

Single data rate (SDR) is the first SDRAM that was created. After that, the Double data rate (DDR) SDRAM was developed on the basis of SDR. The main difference between the two is that DDR can access data on both edges of the clock signal, whereas SDR can only access it on one edge. The speed of DDR is therefore twice that of SDR. Further improvements in bandwidth and power consumption were then made with DDR2, DDR3, and DDR4 technologies.

Finally, the MCU includes an SDRAM Controller (SDRAMC) [4]. This controller is compatible with external 16-bit DRAMs and therefore increases the memory capacity of the MCU. SDRAM performances are improved by the fact that the controller keeps track of the active row in each bank. The controller optimizes the power consumption of the external SDRAM device by offering different operating modes such as Self-refresh, Powerdown, and Deep Powerdown. However, as this controller is only compatible with SDR devices, this type of technology was finally chosen despite being a bit older and less advanced than modern DDR technologies.

### **MT48LC16M16A2**

The MT48LC16M16A2 [10] is a 256Mb high-speed CMOS SDRAM. Its operations are burst-oriented. The location at which the memory is accessed and the number of locations to be processed are configured by the user. In order to start an operation, an ACTIVE command must be sent followed by a READ or WRITE command. It is possible to define the length of the writes and reads. The refresh rate of this compound is 64 ms and an automatic refresh mode is provided along with a power-saving mode and a power-down mode.

#### **2.2.1.2 Non-Volatile Memory**

As already mentioned, non-volatile memories retain their data even when they are not powered. This kind of memory is therefore used to store long-term data. An intuitive example is sensor data coming from other subsystems. This data may simply need to be stored in order to be reused by the CubeSat some time later or it can require to be sent to the Earth. As communication between the CubeSat and the Earth is only possible when the CubeSat is at some specific locations, the data should be stored while the CubeSat is in a region where it cannot communicate with the Earth.

There are many technologies for storing data in a non-volatile way. Therefore only the most popular ones will be presented in this thesis. After having explored these technologies, the chosen components will be detailed and their choice will be justified.

### **ROM**

The first non-volatile memory technology investigated is the Read-Only Memory. As the name suggests, it is impossible to write to this type of memory. The data is entered into the device during its creation and cannot be changed afterward. These memories are mainly used for applications where mass production takes place to store boot loaders and application codes.

## PROM

There are several variants of ROM-based memories. The first one is Programmable Read-Only Memory (PROM). This type of devices can only be programmed once. They are also mostly used in mass production, but this time for products with a shorter life span. Indeed, unlike ROMs, they are programmed after they are manufactured. This avoids the need to change the whole production line when updates are required. A different code can simply be programmed after the production.

## EPROM

Erasable Programmable Read-Only Memory (EPROM) is the next type of studied ROM. Unlike the above-mentioned memories, EPROM can be erased and reprogrammed. They have a small glass window. When this window is exposed to ultraviolet light for a certain period, the data is erased and new data can be programmed. These memories can therefore be erased and reprogrammed in the lab but not during operation. They are therefore usually used during the development of applications during software testing. When the software is finished, it is programmed onto a PROM.

## EEPROM

Electrically Erasable Programmable Read-Only Memory is another type of memory based on ROM. Like EPROM these memories can be erased but this time using an electrical signal. This makes them easier to reprogram and more useful as external memories needing to be reprogrammed several times.

## Flash

Finally, the last type of non-volatile memory derived from ROM is Flash EEPROM, more commonly referred to as Flash memory. Like EEPROM, Flash can be erased using an electrical signal [9]. However, the data is erased and reprogrammed faster than with EEPROM, hence the name Flash. Apart from the speed of reprogramming, the difference between EEPROM and Flash is that the former gives the opportunity to reprogram bit-by-bit, whereas the latter works in blocks. There are two main types of Flash memories that must now be introduced: (i) NAND memories and (ii) NOR memories.

NOR Flash memory cells are connected in parallel to the bit lines. This makes it possible to program the NOR memory cells individually. The name of these memories comes from the fact that the cell arrangement resembles a NOR gate.

NAND Flash memory cells, on the other hand, are connected in series. This configuration then resembles a NAND gate. The cell structure of NAND and NOR Flash can be seen in Figure 2.2.1. It must be mentioned that the cell structure of a NAND memory shown in this figure is that of a single-level cell (SLC) that can store one bit of data per cell. In order to increase the storage capacity, NAND operating with multi-level cells or three-level cells have been developed. Nevertheless, while these technologies increase storage capacity, their higher complexity reduces the speed of operations. This increased complexity also leads to more bit errors, making it necessary to implement more complex

error correction codes (ECC).

## TOSHIBA NAND vs. NOR - Cell Structure

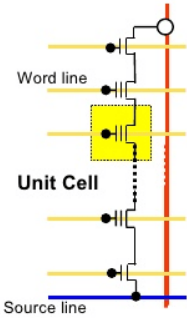
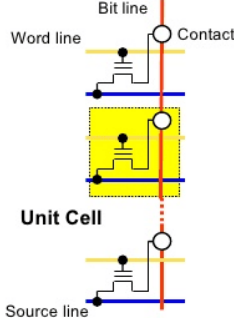
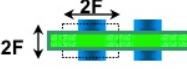
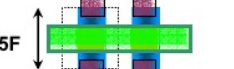
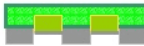

|               | NAND   | NOR  |
|---------------|--|--|
| Cell Array    |   |    |
| Layout        |   |    |
| Cross-section |  |  |
| Cell size     | <b><math>4F^2</math></b>   | <b><math>10F^2</math></b>  |

Figure 2.2.1: Comparison between the cell structure of NOR and NAND memories.

Source : Toshiba America Electronic Components, Inc. November 2003

### NAND & NOR comparison

Now that the cell structure of the two types of Flash memory has been presented, it is time to compare their characteristics [11]. As can be seen in Figure 2.2.1, the serial connection of the NAND memory cells makes them occupy less space than the parallel connection of the NOR memory cells. The two consequences of this are that: (i) NAND devices are provided with higher storage densities as more memory cells can be packed into a single device and (ii) the cost per bit of NAND is lower than that of NOR.

As far as speed is concerned, it all depends on the type of operation performed. The reading speed of NOR memories is faster than that of NAND memories. NOR has a full set of address lines providing random access to each byte of memory. NAND on its side accesses blocks or pages, which makes them slower to read a particular bit. On the other hand, the above-mentioned characteristics mean that the erasing and writing speed of NAND is higher than that of NOR.

With regard to the reliability of the stored data, NOR has better performance than NAND. NAND memories already have some bad memory cells when shipped and the number of bad cells will increase during use. NOR, on the other hand, is shipped without any errors and the number of errors during use is very low.

It is also interesting to consider the differences in power consumption between these two technologies. In active mode NOR consumes more power than NAND. However, in standby mode, the opposite is true and NOR consumes less power than NAND. The most interesting technology in terms of power consumption, therefore, depends on the operating and standby times.

NOR technology is typically chosen for applications that do not require very large storage capacity but require very fast random read access combined with high data reliability. As a result, it is ideally suited for code storage in embedded systems. NAND memory, on the other hand, is more suitable for storing data that requires larger storage capacities and frequent erasing and writing.

Finally, Table 2.2.2 gathers all the information. It consists of a summary of the different studied characteristics of NAND and NOR Flash memories.

| <b>Feature</b>                   | NOR           | NAND          |
|----------------------------------|---------------|---------------|
| <b>Cost per bit</b>              | Higher        | Lower         |
| <b>Storage Capacity</b>          | Lower         | Higher        |
| <b>Read Speed</b>                | Faster        | Slower        |
| <b>Erase Speed</b>               | Slower        | Faster        |
| <b>Write Speed</b>               | Slower        | Faster        |
| <b>Bit flip</b>                  | Less frequent | More frequent |
| <b>Active Power Consumption</b>  | Higher        | Lower         |
| <b>Standby Power Consumption</b> | Lower         | Higher        |

Table 2.2.2: Comparison between the main characteristics of NAND and NOR Flash memories.

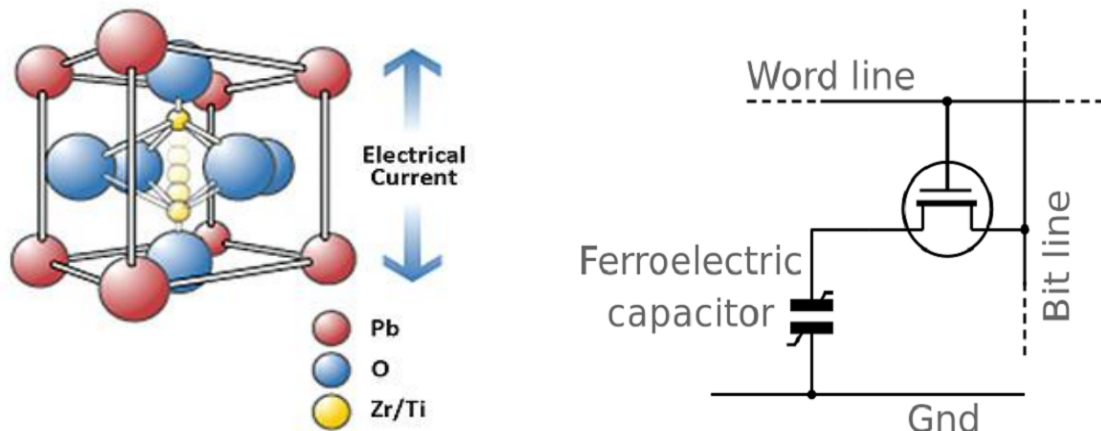
Before proceeding to the details and justifications of the chosen components it is important to introduce one last memory technology that provides some of the advantages of RAM while being a non-volatile memory.

## FRAM

Ferroelectric Random Access Memory (FRAM) is a non-volatile memory that uses a ferroelectric capacitor to store data [12]. This technology is based on the properties of some materials with a perovskite type crystal structure. The most commonly used material in FRAM is lead zirconate titanate (PZT) and its structure is shown in Figure 2.2.2a. The presence of a cation in the center of the PZT structure must be noticed. This cation can take two equal and stable low-energy states. The FRAM technology takes advantage of this cation. Indeed, when an electric field is applied to the PZT the cation moves in the direction of the field. It is then sufficient to apply an electric field in the required direction so that the cation moves and brings the PZT to the desired state. This displacement produces energy in the form of a charge.

The structure of a FRAM memory cell contains the aforementioned ferroelectric capacitor as well as an access transistor. This structure, shown in Figure 2.2.2b, can therefore recall that of the DRAM seen above. It can be noted that the symbol of the capacitor

is not the classical symbol of a linear capacitor since the ferroelectric capacitor has a variable capacitance.



(a) Atomic structure of PZT [12].

(b) Structure of an FRAM memory cell [9].

Figure 2.2.2: The atomic structure of the ferroelectric material used in FRAM (a) and the structure of an FRAM memory cell (b).

Although the name may sound like there is iron in FRAM, this is not the case. The name actually comes from the resemblance between the well-known hysteresis loop in ferromagnetic materials and the curve obtained when the charge of the ferroelectric capacitor is plotted as a function of the applied voltage. FRAM is therefore not affected by magnetic fields. The main advantages and drawbacks of FRAM will now be considered.

The first strength of FRAM is its speed [13]. Writing can be up to 1000 times faster with FRAM than with EEPROM or Flash. Then, data reliability is much better with FRAM than with EEPROM or Flash. Indeed, it can generally perform up to  $10^{15}$  write/erase cycles which is about 100 million times more than for EEPROM.

The FRAM also has some particularly interesting advantages for this project. Indeed, it can modify data with very little current, resulting in very low power consumption. Finally, the FRAM has an essential characteristic. It is radiation resistant which means that it is very unlikely to be affected by the radiation phenomena presented in the Introduction. The other memories studied so far are much more affected by these phenomena which flip the state of a bit.

However, the FRAM has some negative points. The main one is that the ferroelectric properties are lost when the material is too compressed. The memory cells of FRAM can therefore not be as small as those of other memories. FRAM has therefore a lower storage capacity and a higher cost per bit.

### Final choice

It is now time to detail and justify the choices made in terms of non-volatile memory. The advantages of FRAM in the harsh space environment make this technology necessary in this project. Indeed, it is essential to have a reliable device to save important data in the long term while minimizing the risk of errors associated with radiations. However,

the limited storage capacity of FRAM combined with its high purchase price makes it impossible to only use this technology to save non-volatile data. As the density of the chosen FRAM is 2 Mbits for a price of 15€, the choice was made to include 8 FRAMs in the OBC design. This makes it possible to have enough space to store critical data without paying too much or taking up all the space on the OBC's PCB.

In order to complete FRAM with a non-volatile storage device of greater capacity and lower cost, it was decided to use NAND Flash. The decision to use NAND rather than NOR is due to the fact that the desired memory will have to be regularly written and erased. As seen above, NAND is better for this type of operations. As Flash memory is not radiation-tolerant, three NAND were placed on the OBC to increase the reliability. Finally, in order to further increase the total storage capacity it was decided to place two SD cards in the OBC design.

### **FM25V20A**

As mentioned above, the FM25V20A [14] is a 2 Mbits FRAM memory. Since 8 of these devices were placed in the design, the total FRAM memory is 16 Mbits. Communications between the MCU and the FM25V20A take place via the high-speed SPI bus. This means that the high communication speeds of FRAM are fully exploited. In terms of power consumption, the FM25V20A consumes 800  $\mu\text{A}$  at 1 MHz, 100  $\mu\text{A}$  in standby mode, and 3  $\mu\text{A}$  in sleep mode. Although the SPI bus will be explained in the next chapter, it is important to point out that one line of this bus is used to choose which component the MCU communicates with. Each component must therefore have a dedicated selection line. However, by using a demultiplexer, the number of MCU pins required to select a FRAM can be reduced from 8 to 4.

After selecting the FRAM with which communication is taking place, the user can perform a write/read operation to that memory. To do this, the user must send a first byte corresponding to the opcode of the operation to be performed, followed by 3 bytes corresponding to the address where this operation is to be effectuated. The FM25V20A has 256K locations of eight bits of data. The 3 bytes (i.e. 24 bits) of address are thus used to code the 18 bits necessary to differentiate all the addresses. It should be noticed that the first 6 address bits are "don't care" values. Finally, it is interesting to note that the access time to the memory is practically zero, which means that the speed of reading or writing data is the speed of the SPI bus.

### **MT29F2G01ABAGDWB**

MT29F2G01ABAGDWB [15] is a 2GB NAND Flash memory that operates through the SPI bus. It uses SLC technology and therefore prioritizes speed and data reliability over higher capacity.

This device is made up of blocks, each of which has 64 programmable pages. A programmable page consists of 2048 bytes of data storage area and 128 bytes used for memory and error management. This makes a total of 2176 bytes per page. Reading and writing to this memory is done by pages while erasing is done by blocks.

It is also important to talk about data reliability with this component. As mentioned in the presentation of NAND memories, this type of technology is subject to data corruption in the form of inverted bits. In order to reduce these errors, the MT29F2G01ABAGDWB has an error correction code that can detect and correct some of those errors. Finally, the choice to place 3 of these devices on the OBC PCB was not made to triple the storage capacity but to have triple modular redundancy (TMR). The three NAND store the same data and a majority voting system is used to reduce the risk of errors induced by the harsh space conditions against which the NAND is not protected.

### SD Card

For the moment, two sockets that can hold SD cards have been placed on the PCB. The choice of the SD cards used has not yet been made. As the payload and some of the functions that the OBC will have to perform are not yet known this choice is impossible. Indeed, the capacity of the SD card as well as some of its characteristics depend on these elements. However, some research has already been carried out to give an idea of the type of SD card to be chosen for the CubeSat. The results shown in this paper [16] indicate that not all technologies are equal and that it is better to choose an industrial grade SD card over a consumer grade one.

### 2.2.2 Oscillators

The MCU and its peripherals require clock signals to operate. The role of oscillators is to provide a stable clock signal at a predetermined frequency. There are different types of circuits that can act as oscillators, but the ones that offer the most accuracy and stability are quartz crystals. In an OBC it is important that the clock signals are as accurate and stable as possible to avoid all kinds of problems. This is why quartz crystals are chosen as oscillators in the design. Two oscillators are required by the MCU, a slow clock oscillator with a frequency of 32.768 kHz and a fast clock oscillator with a frequency between 3 and 20 MHz.

The MS1V-T1K 32.768KHz [17] has been chosen as slow clock crystal oscillator. It is a square-bodied metal structure containing a high-quality quartz crystal resonator. Its operating temperatures range from -40 to +80°C.

Concerning the high frequency oscillator, the 12MHz ABM3BAIG [18] has been selected. The crystal is placed in a closed ceramic package which ensures high precision and reliability. Its operating temperatures range from -40 to +125°C.

### 2.2.3 Connectors

Connectors can be used to connect different PCBs between them or to connect a PCB to an external device. There are two types of connectors required in this thesis. The first one is a PC/104 used to interconnect the different boards constituting the CubeSat. The second one is a RJ-11 connector used to connect the PCB to an external In-Circuit Debugger/Programmer (ICD). These two connectors are described below.



### 2.2.3.1 PC/104

As it has been presented in the Introduction (Figure 1.2.1), the CubeSat is made of a superposition of different boards. It is essential that some signals are transmitted from one board to another. An intuitive example is the power signals that are generated on the EPS board but must be accessible on all other boards. The PC/104 allows this inter-board communication. Indeed, each board is equipped with one of these connectors and they are stacked on top of each other. In addition to the power signals which are obviously essential, the OBC uses the PC/104 to enable communication between the other boards and the microcontroller. Two examples of communication signals present on the PC/104 of the OBC are the I2C lines and the SPI lines.

### 2.2.3.2 RJ-11 connector

The *MPLAB* ICD 4 [19] is used to program and debug the microcontroller. It is compatible with the *MPLAB* Integrated Development Environment (IDE), which has been used to implement the software of the OBC. The ICD 4 has to be connected to a computer through a USB 2.0 interface to get its power. Once it is powered, the device is connected to an RJ-11 connector placed on the OBC PCB. This connector comprises six pins which are listed below:

1.  $V_{PP}$ : power;
2.  $V_{DD\_TGT}$ : power on target;
3. GND: ground;
4. PGD: Standard Com Data;
5. PGC: Standard Com Clock;
6. Reserved.

Figure 2.2.3 shows an *MPLAB* ICD 4 along with its connections. The only difference between the image and the use of the device in this thesis is obviously that the test interface module has been replaced by the OBC PCB.

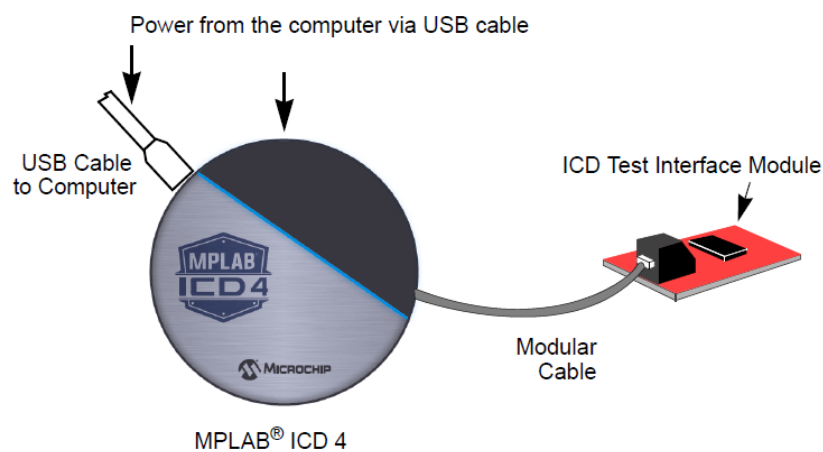


Figure 2.2.3: *MPLAB*® ICD 4 connection example [19].

## 2.2.4 Power monitoring

As it has been explained in the previous section, the OBC board receives its power from the EPS board through the PC/104. All the components present on the OBC board operate at a voltage of 3.3V. For that reason, the only power signals on this board are 3.3V and ground.

### 2.2.4.1 Protection circuit

Although the EPS board is supposed to provide a stable 3.3V signal, it has been decided to place a device to monitor the voltage and current coming from the EPS upstream of the other components of the OBC board. This device is the LTC4368 [20] and has the following features:

- Overvoltage protection;
- Undervoltage protection;
- Forward and backward overcurrent protection.

To perform its protection functions the LTC4368 is connected to a Dual N-Channel Metal-Oxide-Semiconductor Field-Effect Transistor (MOSFET) used as a switch. When the LTC4368 detects a voltage or current outside the defined ranges, it isolates the load from the power supply by switching the MOSFET off. Overvoltage and undervoltage are detected using comparators whose values are set via several external resistors forming a voltage divider. Overcurrents are detected using an external sensing resistor that allows the selection of the overcurrent value. Figure 2.2.4 shows an example application of the LTC4368.

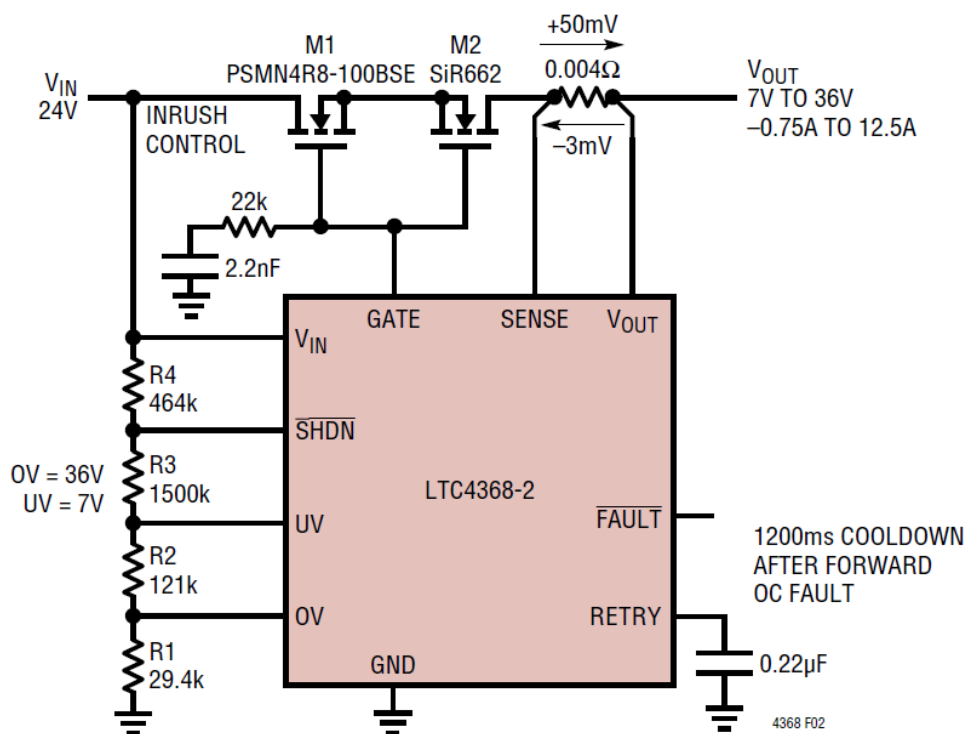


Figure 2.2.4: Example application of the LTC4368 [20].

The choice of the resistor's values for the OBC protection is based on the equations 2.2.1, 2.2.2, 2.2.3, and 2.2.4. The first one is based on  $V_{OS(UV)}$ , which is the maximum tolerable offset voltage at the UV pin, and  $I_{UV}$ , which represents the worst case leakage current at the UV pin.  $I_{UV}$  is given to be equal to 10nA and  $V_{OS(UV)}$  is chosen to be equal to 3mV. It is worth noting that in what follows,  $R_3$  corresponds to  $R_3 + R_4$  from Figure 2.2.4. The first equation is obtained as being:

$$R_1 + R_2 = \frac{V_{OS(UV)}}{I_{UV}} = \frac{3mV}{10nA} = 300k\Omega \quad (2.2.1)$$

The second equation is based on the UV threshold value which is chosen equal to 3V in this application since the minimum supply voltage of the MCU is 3V. It also relies on the comparator rising threshold voltage which is equal to 0.5V. The equation is thus given by:

$$R_3 = \frac{V_{OS(UV)}}{I_{UV}} \cdot \frac{UV_{TH} - 0.5V}{0.5V} = \frac{3mV}{10nA} \cdot \frac{3V - 0.5V}{0.5V} = 1.5M\Omega \quad (2.2.2)$$

The last two equations are based on the OV threshold value, which is chosen equal to 3.6V as the maximum supply voltage of the MCU is 3.6V. These equations are as follows:

$$R_1 = \frac{\left(\frac{V_{OS(UV)}}{I_{UV}}\right) + R_3}{OV_{TH}} \cdot 0.5V = \frac{\left(\frac{3mV}{10nA}\right) + 1.5M\Omega}{3.6V} \cdot 0.5V = 250k\Omega \quad (2.2.3)$$

$$R_2 = \frac{V_{OS(UV)}}{I_{UV}} - R_1 = \frac{3mV}{10nA} - 250k\Omega = 50k\Omega \quad (2.2.4)$$

The equations giving the forward and backward overcurrent are easier. The forward overcurrent threshold ( $I_{OC,FWD}$ ) is determined by the value of the external sense resistor ( $R_{SENSE}$ ). Indeed, the forward current passing through  $R_{SENSE}$  generates a voltage that is compared with a 50mV threshold by an internal comparator. It has been decided to limit the forward current to 0.5A. The value of  $R_{SENSE}$  is thus given by:

$$R_{SENSE} = \frac{50mV}{I_{OC,FWD}} = \frac{50mV}{0.5A} = 0.1\Omega \quad (2.2.5)$$

Concerning the reverse overcurrent, the method is the same. The only difference is that the voltage generated by the reverse current flow through  $R_{SENSE}$  is compared to a threshold of -3mV. The reverse overcurrent threshold ( $I_{OC,REV}$ ) is thus given by:

$$I_{OC,REV} = \frac{-3mV}{R_{SENSE}} = \frac{-3mV}{0.1} = -30mA \quad (2.2.6)$$

To conclude on the use of the LTC4368, Table 2.2.3 resumes the threshold values causing the device to switch the MOSFET off in order to protect the OBC components.

| Type of protection  | Threshold value |
|---------------------|-----------------|
| Undervoltage        | 3V              |
| Overvoltage         | 3.6V            |
| Forward overcurrent | 0.5A            |
| Reverse overcurrent | -30mA           |

Table 2.2.3: Summary of the protection threshold values of the LTC4368.

### 2.2.4.2 Power consumption

In addition to the protection circuit, it was decided to measure the power consumption of the OBC board. For this purpose, the MAX9938 device [21] and the AFEC of the MCU were used. The MAX9938 is a high-side current sense amplifier that amplifies the voltage generated by the current flowing through a sense resistor with a voltage gain of 25V/V. A special characteristic of this device is that its input common-mode voltage is used as supply voltage. Its typical operating circuit is represented in Figure 2.2.5.

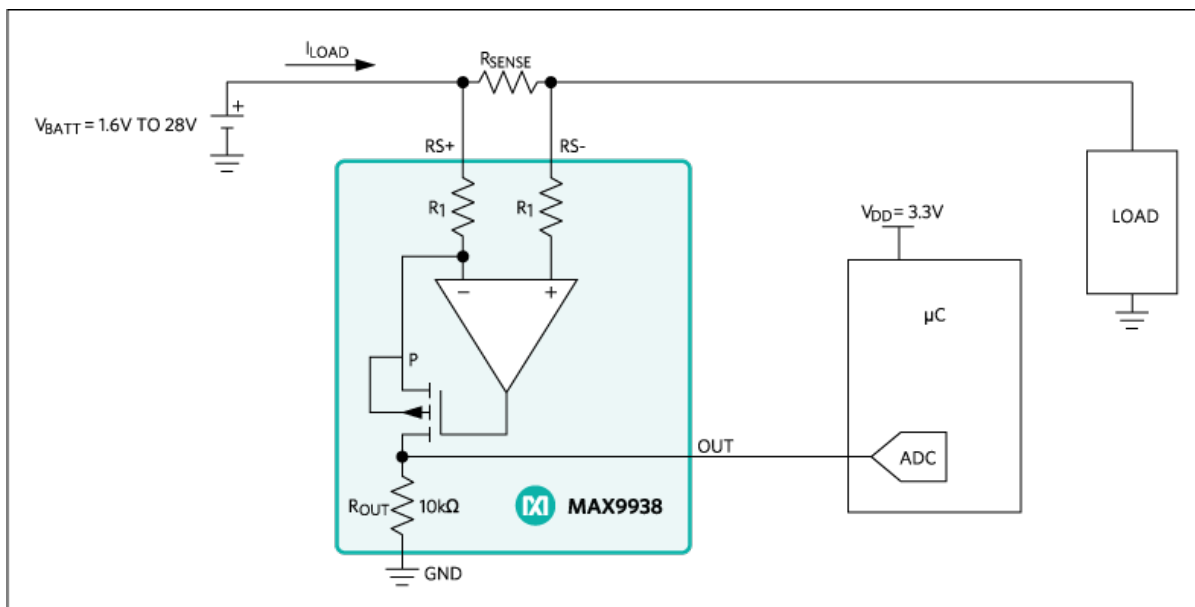


Figure 2.2.5: Typical operating circuit of the MAX9938 [21].

The choice of the  $R_{SENSE}$  had to be done carefully. Indeed, a large value would have caused a lot of power dissipation due to the  $I^2 R_{SENSE}$  losses.  $R_{SENSE}$  was finally chosen equal to  $20m\Omega$ . The output voltage of the MAX9938 goes to the AFEC of the MCU and is converted using its ADC module. A capacitor is placed at the output of the MAX9938. It is used to stabilize the voltage while it is sampled by the ADC as well as to reduce the noise at the output. The power consumption of the OBC can thus be monitored easily.

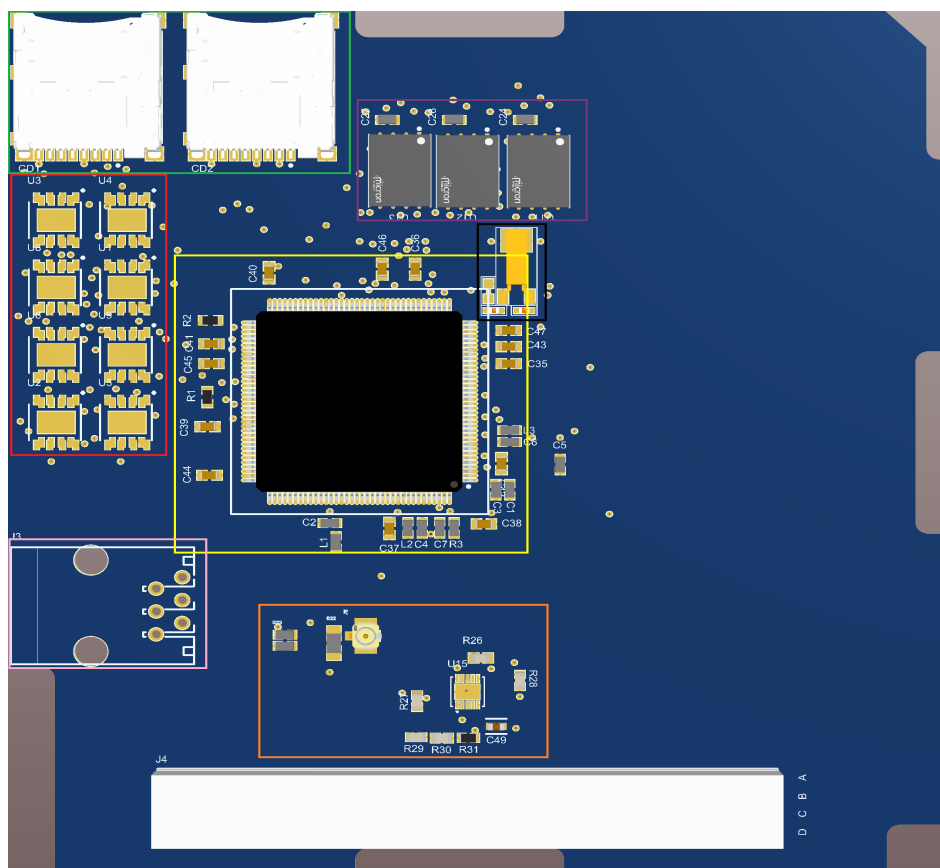
Since the CubeSat project is not yet ready to be tested in its entirety, the results obtained now would not have been representative. Indeed, most of the power consumption of the OBC will come from the tasks related to the other subsystems of the CubeSat. For this reason, no power consumption test was performed during this thesis.

## 2.3 Final PCB presentation

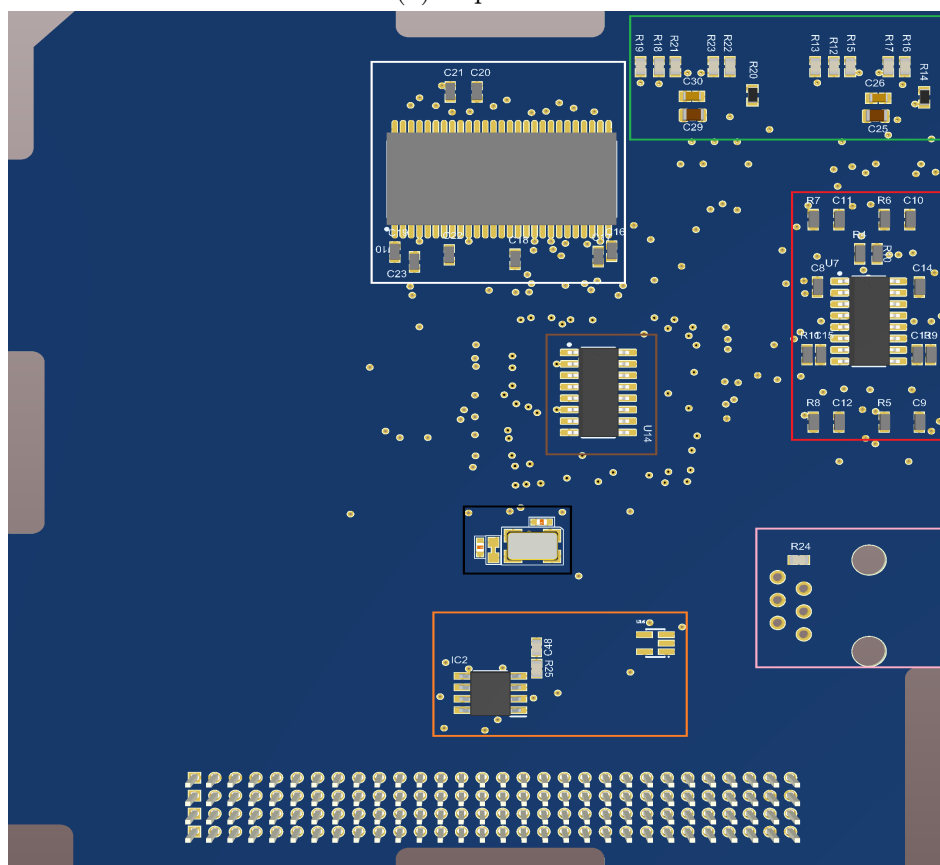
To end the presentation of the hardware part of this project a representation of the final OBC's PCB is given in Figure 2.3.1. It is divided into separate areas that have been highlighted with colored rectangles. The components laying in each rectangle will now be listed.

- The yellow rectangle contains the microcontroller and its passive components;
- The red rectangles include the eight FRAMs along with their passive components and the demultiplexer that has been used with these memories;
- The two black rectangles enclose the two crystal oscillators. The 32.768KHz slow clock crystal oscillator is on the top side while the 12MHz high frequency crystal oscillator is on the bottom side;
- The orange rectangles comprise all the components linked to the power monitoring;
- The white rectangle contains the DRAM and its passive components;
- The green rectangles include the two micro SD card sockets along with their passive elements;
- The purple rectangle comprises the three NAND memories and their passive components;
- The brown rectangle simply contains the demultiplexer used by the micro SD cards and the NAND memories;
- The pink rectangles include the RJ-11 connector and its passive components;
- The big white box and the 120 pins situated at the bottom of respectively the top and bottom views is the PC104 connector.

It is worth noting that a lot of free space has been left on the PCB. This has been done intentionally to allow the addition of other components in this area. Indeed, as the payload of the CubeSat is still to be determined, it might be necessary to add some components in the future development of the OBC.



(a) Top view.



(b) Bottom view.

Figure 2.3.1: 3D top (a) and bottom (b) views of the final OBC's PCB.

# Chapter 3

## Software implementation

The OBC is the subsystem at the center of the CubeSat design that interacts with all the other subsystems. It can thus be viewed as the brain of a CubeSat. However, until now the only link between the OBC and the other subsystems is a mechanical one that happens through the PC/104 connector. A software is therefore required in order to implement the actions of the OBC.

This chapter is dedicated to the introduction of this OBC software. The first thing while designing the software of an embedded application is to select a software architecture. The first section will thus be related to that selection. After that, some explanations will be given about the software program that was used to develop the code. Lastly, the different OBC applications implemented during this thesis will be detailed. This will give an overview of what the OBC can already do and how those tasks have been implemented. It is worth mentioning that all the codes have been implemented using the C programming language.

### 3.1 Software architecture

The choice has been made to develop the software of the OBC following a Real-Time Operating System (RTOS) architecture. An operating system (OS) is the part of the software that manages the interaction between tasks and the system resources [22]. It is composed of a kernel that controls the system and a library of functions that facilitate its use. RTOS is the OS developed for embedded applications.

Some important features of RTOS need to be introduced. An RTOS is made of several tasks each with a given priority and a given state. Interrupts are also available. An interrupt consists in sending a signal asking the processor to suspend the task currently in progress in order to execute an interrupt routine making it possible to execute urgent operations within them. The kernel contains a scheduler that manages the state of the system by assigning the processor to the tasks.

Data communications between different tasks is critical and can be handled with semaphores and message queues. Semaphores are objects having a value  $\geq 0$ . A semaphore can either be given, which increments its value, or taken, which decrements its value. A semaphore can only be taken if its value is  $> 0$ . Message queues are objects used to transfer data between tasks. The number of messages that a queue can store and the size

of a single message are fixed by the user. A task trying to read an empty queue moves to the suspended state.

Finally, there are five possible states for a task:

1. Dormant: The task is not scheduled;
2. Active: The task is currently running;
3. Executable: The task is ready to perform operations but another task is already active;
4. Blocked: The task is suspended and waits for a signal, or for a resource;
5. Interrupted: The task is executing an interrupt routine.

There are a lot of advantages to the use of RTOS. First, the OS dynamically controls the execution of non-urgent tasks through the scheduler and can suspend a task before its completion to execute another one. This allows long computations to be performed while maintaining low latency. Then, the structure of the RTOS makes it easy to add new tasks without having to rethink the whole system. This feature is particularly useful in the case of this project as the payload of the CubeSat is still to be determined and therefore additional tasks will need to be implemented and performed by the OBC.

On the other hand, drawbacks of RTOS include the difficulties introduced by data exchange and the consumption of resources by the OS. In addition, the main difficulty associated with RTOS is the complexity of its implementation. However, this is not a real problem as there already exist open source RTOS that can be used. The one used in this project is FreeRTOS.

## 3.2 MPLAB

*MPLAB X IDE* is a freeware developed by *Microchip* that is widely used for embedded applications. It offers some interesting characteristics such as a debugger and a code configurator, which will be explained in more detail in the next section. It is also worth noting that some examples of codes using the ATSAMV71Q21 are available on the *Microchip* website [23].

### 3.2.1 Harmony v3

Harmony v3 is an embedded software integrated into *MPLAB X IDE* which offers software modules that simplify the development of the code. In order to use it, the first step is to download the necessary modules for the application that needs to be implemented. This is done using the Harmony 3 Content Manager that can be found by going to the **Tools** tab in *MPLAB X IDE* and then selecting **Embedded > MPLAB<sup>®</sup> Harmony 3 Content Manager**. The next step is to launch the Harmony 3 Configurator. This is done by clicking **Tools > Embedded > MPLAB<sup>®</sup> Harmony 3 Configurator**. At that point, the project graph will open. All available modules are then listed under the tab **Available components**. The user must now select the modules needed in his application. Figure 3.2.1 shows the modules used for the implementation of the software of the



OBC. Each module must also be configured with the appropriate values. The most important ones will be detailed in the corresponding subsection of the Code implementation section.

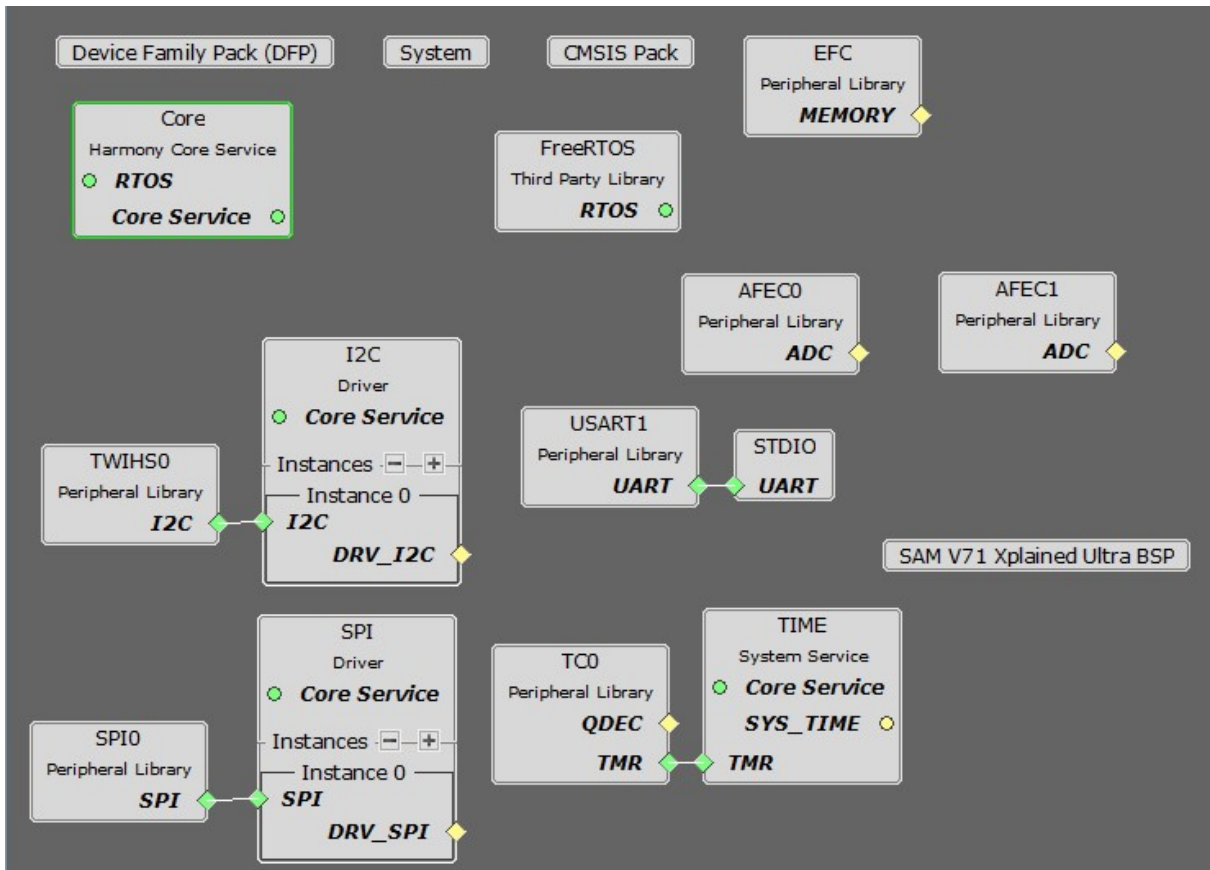


Figure 3.2.1: Project graph

When the project graph is finished, it is necessary to assign the different pins of the microcontroller. This is done by going to the **MHC** tab that is present in *MPLAB X* IDE when the Harmony 3 Configurator is open and then selecting **Tools > Pin Configuration**. The pin diagram of this project is depicted in Figure 3.2.2. This one makes it possible to see that many pins of the MCU are still available. Most of them will be assigned before launch once the other subsystems of the CubeSat will be finished.

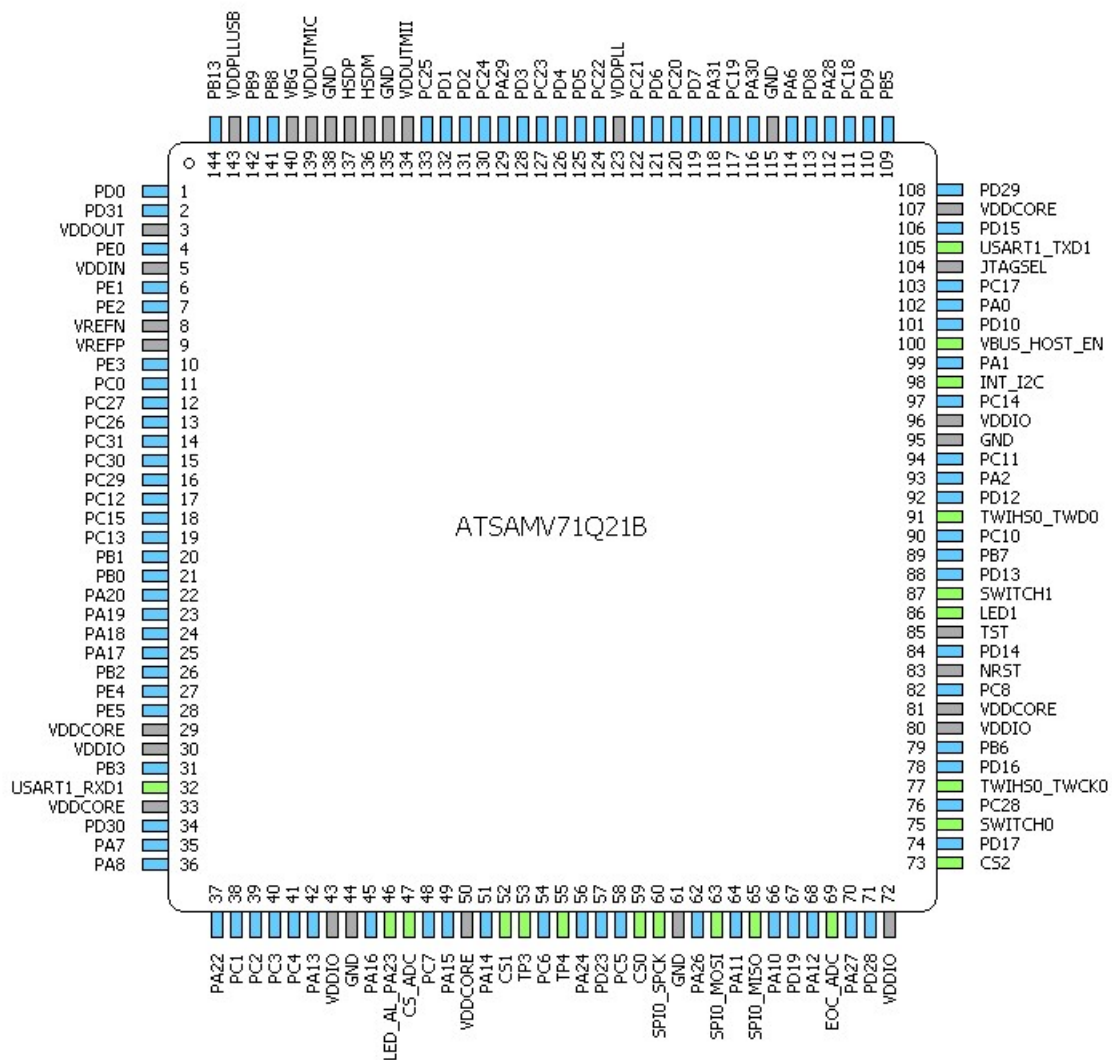


Figure 3.2.2: Pin diagram of the ATSAMV71Q21. The green pins are the ones assigned while the blue ones are available and the grey ones are unavailable.

It is also worth mentioning that some other interesting functionalities are available by clicking **MHC** > **Tools** such as the Clock Configurator or the Nested Vector Interrupt Control (NVIC) Configurator.

Finally, when the project graph and all configurations are completed, the code can be generated by clicking on the "Generate Code" icon. At that point *MPLAB X* IDE generates all the required drivers' code. Those can be found in the project files.

### 3.3 Code implementation

For this project, five tasks were implemented in order to achieve all the objectives of the project. Each task has a set of states in which it can be. The states indicate what the task is currently doing. In addition, a data structure representing the data of the task is associated to each task.

### 3.3.1 I2C

An I2C bus is a serial communication mechanism consisting of a pair of two-way lines: (i) the Serial Data (SDA) line and (ii) the Serial Clock (SCL) line [22] [24]. The default state of both lines when no communication is in progress is high. To do this, the lines are connected to VDD through pull-up resistors. Thus, the devices connected to the I2C bus can only pull the lines down. Master and slave are two important terms in the I2C protocol. The master is the device that generates the clock signal and manages the data transaction with the slaves. The slave on its side is the device responding to the instruction of the master. Regarding the transmission speed, the standard mode speed is 100kHz while the fast mode speed is 400kHz. Figure 3.3.1 represents the structure of an I2C bus with one master and three slave devices.

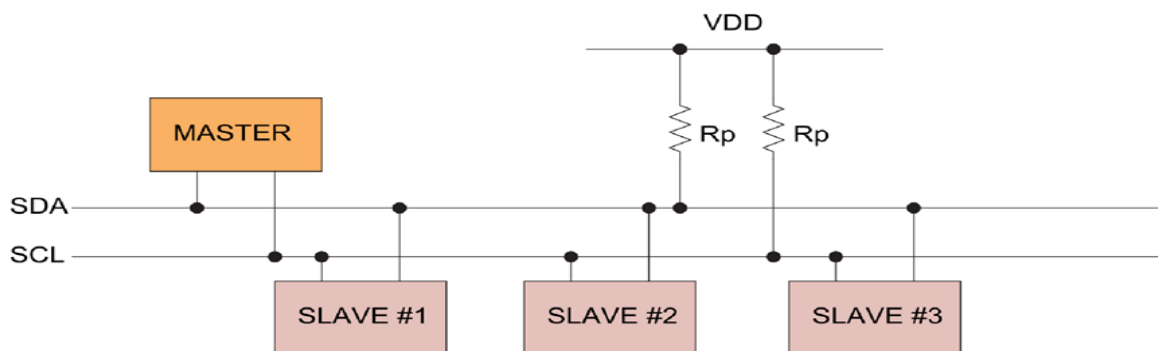


Figure 3.3.1: Structure of an I2C bus with one master and three slave devices [24].

Figure 3.3.2 shows an example of communication on an I2C bus. A communication always starts with a start condition and always ends with a stop condition. These two conditions are sent by the master and correspond to the only two moments when a transition from high to low on SDA is possible while SCL is high. At all other times, SDA changes must take place when SCL is low.

Between the start and stop conditions data are exchanged in 8 bits blocks and Most Significant Bit (MSB) first. The master always sends the first block. It corresponds to the address of the slave and the direction of the communication. Indeed, the first 7 bits represent the address of the slave while the last one indicates whether the master wants to perform a read or a write operation with the slave. A low value of the 8<sup>th</sup> bit means that the master will write data to the slave while a high value of the 8<sup>th</sup> bit means that the master will read data sent by the slave. The other data blocks are thus sent by the master in case of a write operation or by the slave in case of a read operation. Moreover, any number of data blocks can be communicated between the start and stop conditions.

The last essential information about I2C concerns the acknowledge (ACK) and not acknowledge (NACK) bits. Indeed, each block of data is followed by a bit placed on SDA by the receiver. This bit is used as a verification mechanism. The first ACK is sent by the slave which address corresponds to the one sent by the master. In case the master sends an address that does not correspond to any slave, the first block is not acknowledged and the communication stops immediately. Concerning the subsequent blocks, if the receiver does not acknowledge the block the communication is also immediately stopped.

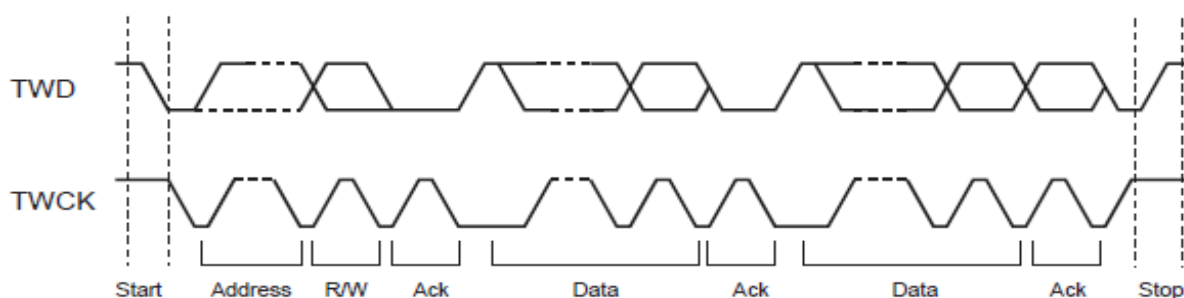


Figure 3.3.2: Example of communication of an I2C bus. Note that in this figure TWD corresponds to SDA in the text while TWCK corresponds to SCL [4].

One of the advantages of I2C buses is that they allow communication between many devices using only two wires. Another advantage is the feedback provided by the ACK/NACK bit. Regarding the disadvantages, the first one is the limited speed of I2C bus communications when compared to the speed of SPI bus communications. Finally, it is important to point out that all the slaves on an I2C bus must have different addresses in order to identify which one the master wants to start a transaction with.

### 3.3.2 SPI

Serial Peripheral Interface (SPI) is a 4-wire synchronous serial communication interface. The four signals used by SPI are the following [25]:

- Serial Peripheral Interface Clock (SPCK);
- Master In Slave Out (MISO);
- Master Out Slave In (MOSI);
- Chip Select (CS).

As the names of the signals indicate, the SPI works with master and slaves as was the case with the I2C bus. The clock signal is generated by the master and is used as a synchronization method. MISO and MOSI are the two lines for data transfer. The first one corresponds to the line on which the slaves send data to the master while the second one is the line on which the master sends data to the slaves. The master can only communicate with one slave at a time. The CS line is used for that purpose. Indeed, the standard state of this line is high and when the master wants to communicate with a slave, it pulls the CS line of that slave down. During communication between a master and a slave, all CS lines other than that of the slave communicating must be high.

It is important to specify that SPCK, MISO, and MOSI lines are shared by all the devices connected to the SPI bus, while every slave has a different CS line. Finally, the CS lines can be connected to a demultiplexer in order not to use too many pins from the master. That method has been applied in this thesis and is depicted in Figure 3.3.3 along with the above-mentioned SPI components.

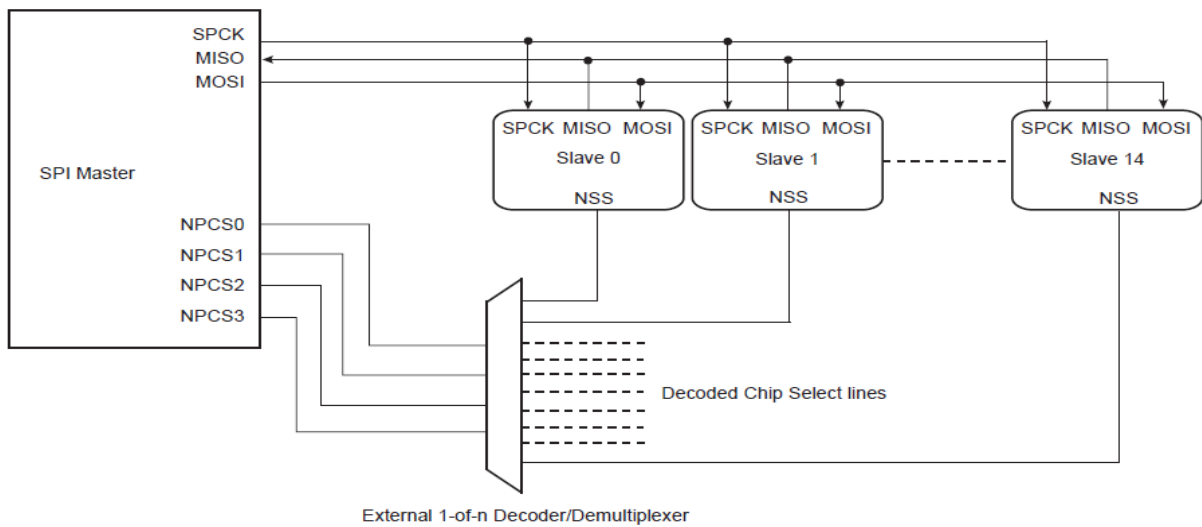


Figure 3.3.3: Representation of an SPI bus working with a demultiplexer used to decode the CS lines [4].

Before explaining the way data are exchanged in SPI, it is necessary to introduce the clock polarity (CPOL), the clock phase (CPHA), and the idle state concepts. The latter corresponds to the start and end of communications when the CS signal switches from high to low and from low to high respectively. CPOL and CPHA on their side relate to a specificity of the SPI bus. Indeed, the master must define a polarity and a phase for the clock signal that satisfy the technical characteristics of the slave. This is done through the CPOL and CPHA bits. The first one defines the polarity of the clock signal in the idle state. The second one determines on which clock edge the master writes on MOSI and on which one the slave writes on MISO. Master and slave writes always occur on opposite clock edges.

To initiate a communication, the master must generate a clock signal and pull down the CS line of the slave with which it wants to initiate a transmission. Another characteristic of SPI is that it is a full-duplex interface. This means that data are transmitted both on MOSI and MISO even when only one line is sending a useful signal. At the end of the communication, the master has to pull the slave CS line back to the high state. An example of SPI communication with  $CPHA = 1$  is shown in Figure 3.3.4. However, it is interesting to note that the selected microcontroller uses NCPHA instead of CPHA where the N simply means Not.

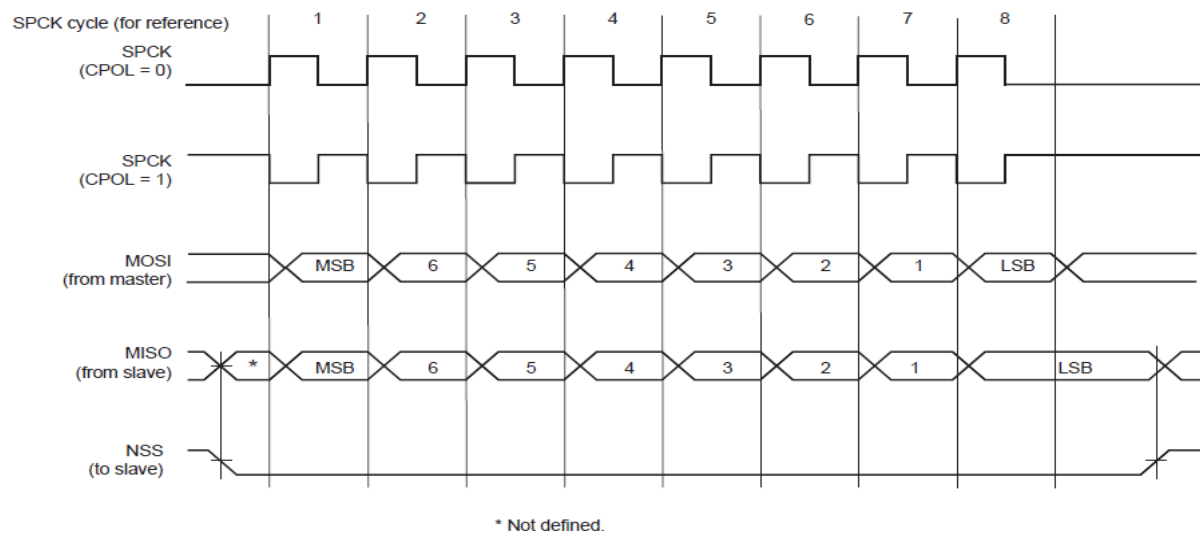


Figure 3.3.4: Example of an SPI communication with  $NCPHA = 0$  [4].

Finally, one of the main advantages of the SPI bus is its speed. Indeed, it can operate at speeds up to a few MHz. On the other side, it requires at least 4 wires and that number can quickly increase with the number of slave devices.

### 3.3.3 External I2C bus task

The role of this task is to enable communication with the TCA9555 [26]. The TCA9555 is a 16-bit I/O expander for the I2C bus. It comprises two ports of 8 pins that can be configured as inputs or outputs. The user can configure the direction of each pin by writing the appropriate bit inside the configuration register. At power on, all pins are configured as inputs. This component also includes an interrupt pin that indicates when a new value appears at one of the inputs. This device is used by the EPS board to monitor different signals.

The initial state of the task performs the configuration of the I2C driver used to communicate with the TCA9555. The I2C driver clock speed is set at 400kHz.

Once the I2C driver is initialized, the task moves to the state in which the TCA9555 is configured. In that state, the task waits for a message on the *I2C\_Queue*. This message is sent by the user task and contains the transfer type, which must be equal to *TRANSFER\_TYPE\_I2C\_CONFIGURATION*, as well as the values of the two configuration registers of the TCA9555. The role of these registers is to configure the directions of the I/O pins. In order to define a pin as an input, the bit corresponding to this pin must be set to 1 in the configuration register. Conversely, when the bit corresponding to a pin is set to 0, the pin is defined as an output. When the message queue has been received, the two configuration registers are initialized with the values defined by the user. The last thing to do in this state is to read the values of both ports before activating the interrupt I2C pin of the MCU connected to the interrupt pin of the TCA9555. Indeed, modifying the configuration registers can cause an interrupt to occur and that is the reason why the MCU I2C interrupt pin is only enabled after the initialization of these registers.

When the initialization of the I2C driver and of the configuration registers is over, the task moves to the read/write state. In this state, the task waits once again for messages on the *I2C\_Queue* coming from the user task. A message contains the type of operation to be performed, one or two data bytes depending on the type of operation to be performed, and the address to which the data must be stored in the case of a read operation. The five different types of messages will be described hereafter.

The first type is a port write request. In this case, the user task sends a data byte and the port in which this byte has to be written. The I2C task then performs the write of that byte in the right port.

The second type of operation is a bit write. The process is similar to the previous case, except that the data byte is replaced by a data bit and the port is replaced by a channel.

The third type is a port read request. Here, the user task sends the number of the port that needs to be read and the address of the FRAM where the result must be stored. The I2C task then performs a read of that port and sends the result to the FRAM.

The fourth type, which corresponds to a read bit, works in the same way as the previous one with the difference that the port is replaced by the channel.

Finally, the fifth type is called by the user task when an interrupt occurs on the MCU I2C interrupt pin. In this case, the user task only sends the FRAM address to which the data have to be stored. The I2C task then reads the two ports of the TCA9555 and the results are stored in the FRAM.

### 3.3.4 External ADC (SPI) task

This task is responsible for performing analog to digital conversions using the MAX1128 [27] and saving their results in the FRAM memory. The MAX1128 is a high speed analog-to-digital converter that has low-power consumption characteristics. Indeed, it only consumes 5.4mW at a speed of 1 Megasamples per second with a 3V power supply. It has 16 ADC channels that can be configured in single ended or differential mode and it can be used with internal or external clock signals. This device is used in the EPS board to monitor some signals.

Its first state is responsible for the initialization of the SPI driver used to communicate with the MAX1128. The SPI driver parameters are the following:

- A baud rate in Hertz of 1MHz.
- Data is changed on the leading edge of SPCK and captured on the following edge of SPCK.;
- Idle High Clock Polarity;
- 8 data bits per transfer;
- Pin PD27 used as chip select pin with active low polarity configuration of the chip select.

When the SPI driver is configured, the task goes to the configuration state in which the MAX1128 is configured by putting the appropriate values in the *ADC Configuration* register. The configuration is the following:

- External single-ended voltage reference;
- Averaging mode off;
- 1 conversion for each requested result;
- Scans channel N and returns 4 results if repeat mode activated;
- Full shutdown power mode;
- Instruction echo on Data OUT (DOUT) disabled.

In full shutdown power mode, all circuits are powered down at the end of each conversion. The falling edge of CS switches the device back on and 2 cycles are needed before valid conversions take place. Finally, it is worth noting that the information in the registers is not lost while the device is in full shutdown.

After the SPI driver and the MAX1128 are configured, conversions can take place. In order to do that, the task enters in the conversion state. In this state, the task waits for a conversion instruction to be received on the *ADC\_Queue*. A conversion instruction is a data structure consisting of the high and low bytes of the *ADC Mode Control* register along with the address where the conversion must be stored in the FRAM.

Concerning the two *ADC Mode Control* registers, they are defined in the *Constant.h* file and configured in the following way:

- The conversions are started manually;
- The user selects the channel for which he wants the conversion;
- The registers are not reset;
- Auto Shutdown mode;
- DOUT is a 16-bit data word starting by a default 0 bit, followed by a 15-bit conversion result led by the MSB.

When the conversion has been performed, the result is read and finally sent to the FRAM address that was previously specified by the user.

### 3.3.5 FRAM (SPI) task

The role of this task is to store data in the FM25V20A [14] FRAM memories and to read it back when needed. Its first state corresponds to the initialization of the SPI driver used to communicate with the FM25V20A memories.

Before detailing the configuration of the driver, it is important to mention that some modifications had to be made in the source files generated by the Harmony v3. Indeed, as explained in section 2.2.1.2, the FRAM chip select is managed with a demultiplexer. It



is therefore necessary to be able to manage the 3 FRAM chip select pins and the demultiplexer enable pin simultaneously. This was not the case in the original codes in which only one chip select could be defined per SPI driver handler.

Another important comment must be made concerning the use of the FRAMs. Each of these memories has addresses ranging from 0h to 3FFFFh. To simplify the use of the code, the memory addresses range from 0h to 23FFF6h which corresponds to  $9 \cdot 3FFFh - 1$ . Thus, the user only needs to provide a memory address within this range and the program is responsible for finding the memory and the internal address of that memory corresponding to the value entered.

Having explained the above, the configuration of the driver can be detailed. It is as follows:

- A baud rate in Hertz of 1MHz;
- Data is changed on the leading edge of SPCK and captured on the following edge of SPCK;
- Idle High Clock Polarity;
- 8 data bits per transfer;
- Pins PA25, PD25, and PA5 used as chip select pins with active low polarity configuration of the chip select.

When the initialization of the driver is done, the task moves to the transfer state. In this state, it waits for a read or write instruction on the *FRAM\_Queue*. In addition to the write or read instruction, the message queue contains the address of the FRAM with which communication is desired, the size of the transfer and the data to be saved in the case of a write. The received address is transferred as a 32 bits integer. At that point, it is therefore decomposed in the 3 addresses of 8 bits needed to communicate with the FM25V20A. The number of the FRAM with which the communication has to take place is retrieved at the same time. Once the correct FRAM is selected and the 3 addresses bytes are known, the task can proceed with the writing or reading of data.

In order to write data to the FM25V20A the first thing to do is to issue the Write Enable (WREN) command. Then the write operation can be performed. It starts with the WRITE opcode followed by the 3 addresses bytes and finally by the data that has to be written in the FRAM. It is worth noting that CS has to be deasserted between the WREN and the WRITE commands.

The reading operations on their side work in the following way. The first byte sent to the FM25V20A corresponds to the READ opcode. This one is followed by the three addresses bytes. After that, the FM25V20A starts sending the data bytes on the MISO line. The number of data bytes sent from the FM25V20A is equal to the number of bytes specified by the user in the message queue when requesting a read operation. It is interesting to note that the MOSI line is ignored while a read operation is in progress.

Finally, the following are some specificities of the FM25V20A, both in write and read mode:

- The addresses continue to be incremented as long as the master continues to transmit a clock signal and CS is low;
- If the last address of the FRAM is reached, the counter will jump back to the first one;
- Data are written/read MSB first;
- A rising edge of CS terminates the current operations.

### 3.3.6 User task

The user task has already been mentioned many times in the previous points. Indeed, it is the task that manages all the other tasks. It was developed mainly but not only to show how the other tasks should be used in the future use of this work. The first notable thing about this task is that the three message queues already mentioned before are created at the initialization of the task. Each queue is created with a number of elements being equal to a constant defined in the *Constants.h* file. The size of an element is itself equal to the size of the data structure used in that queue.

The first state of this task is used as a configuration state. In this state, the different flags and callback functions are initialized. A callback function and a flag are linked to each interrupt. When the interrupt occurs its corresponding callback function is called. This function updates the status of the flag related to it.

Once everything has been initialized, the task moves to its operating state. In this state, the task basically checks the value of different flags in an infinite loop. When the value of a flag is true, the user task performs the application related to this flag, changes the value of the flag to false, and then moves to the next flag. These flags are used for three different things. The first one corresponds to timer interrupts used to send a message to one of the tasks detailed above through the corresponding message queue. This is done in order to show how the other tasks can be used in future works. The other two types of flags relate to the MCU's internal ADC and watchdog timers. They will be described in the following sections.

#### 3.3.6.1 Internal ADC

The second application requiring flags in the user task is linked to the MCU Analog Front-End Controller [4]. Before detailing the role of those flags, some details about the AFEC are needed. The part of the AFEC that is used in this application is the Analog-to-Digital Converter.

This ADC has a standard resolution of 12 bits and integrates a sleep mode which is of great interest in this project. A final point of great value is that the AFEC incorporates a temperature sensor. This one will thus be used in order to monitor the temperature around the MCU. In addition to being used to obtain a voltage proportional to temperature, the MCU's internal ADC is also used to monitor the power consumed by the

OBC. Indeed, as described in section 2.2.4, a MAX9938 is present on the OBC in order to monitor constantly the consumed power.

Now that the AFEC has been detailed, the flags that depend on it can be explained. The first flag simply relies on a timer interrupt used to launch the ADC conversions. The second flag relies on the AFEC driver functions and is true when the specified AFEC channel result is ready to be read. To conclude on the use of the internal ADC, it is important to notice that some manipulations have to be made with the ADC results. The formula 3.3.1 is used to obtain the ADC voltage from the ADC result.

$$\text{ADC\_voltage} = \text{ADC\_result} \times \frac{\text{AFEC\_VREF}}{\text{ADC\_COUNT\_MAX}} \quad (3.3.1)$$

Where the AFEC parameters are as follows:

- Positive reference voltage:  $\text{AFEC\_VREF} = 3.3\text{V}$ ;
- $\text{ADC\_COUNT\_MAX}$  is equal to  $2^{12} - 1$  as the ADC has a resolution of 12 bits.

Finally, the temperature is obtained with the formula 3.3.2.

$$\text{Temperature} = (\text{ADC\_voltage} - 0.72) \times \frac{100}{233} + 25 \quad (3.3.2)$$

Where the temperature sensor parameters are the following:

- Typical  $\text{ADC\_voltage}$  at  $25^\circ\text{C} = 0.72\text{V}$ ;
- Typical temperature Sensitivity =  $2.33\text{mV}/^\circ\text{C}$ .

### 3.3.6.2 Watchdog timers

If the software enters a deadlock, the system may get blocked. The purpose of the Watchdog Timer is to prevent this from happening [4]. For this purpose, a 12-bit counter is used. The user chooses a value for this counter, the maximum value corresponding to 16 seconds, and the counter must be reset before it expires. Otherwise, a total or partial reset of the MCU takes place.

As it is important to prevent any blockage of the system to happen, the Reinforced Safety Watchdog Timer of the MCU is also used. Even if it shares the same features as the WDT, the RSWDT is entirely independent of the WDT. In fact, its clock source is automatically selected to be different from that of the WDT. Thus, if the WDT clock source fails, the RSWDT performs the monitoring of the system. This provides maximum safety regardless of the external operating conditions.

The last application requiring a flag in the user task is related to the two above-mentioned Watchdog Timers of the MCU. That flag is simply linked to a timer interrupt and is used to reset the two WDT. Finally, the Watchdog Timer and Reinforced Safety Watchdog Timer parameters are the following:

- WDT counter value = 1536 which corresponds to 6 seconds;

- RSWDT counter value = 4095 which corresponds to 16 seconds;
- Booth timers are reset every 4 seconds.

# Chapter 4

## Hardware and Software testing

When making a new product, it is always important to make sure that it works properly. However, this is specially critical for a space application. Indeed, while it is not uncommon to see cars called back to the garage for corrections, this is not feasible in this case. If the CubeSat encounters a problem, it is impossible to solve it and the whole mission is lost. The goal of this chapter is to verify that the OBC design discussed in the previous chapters works perfectly.

To start, the development board conceived around the selected MCU will be detailed. Then, the first batch of tests performed with this board to learn how to use it will be described. After that, the PCB designed for testing purposes will be shown. Finally, the components of the latter will be separated into groups according to their function. The different groups will then be presented together with the tests performed within them.

### 4.1 Development board

As it has been explained in section 2.1.1, an Xplained Ultra evaluation kit is available to perform tests on the ATSAMV71Q21 [28]. This evaluation board has been largely used during this thesis, so it is interesting to detail some of its features. An overview of the board is shown in Figure 4.1.1.

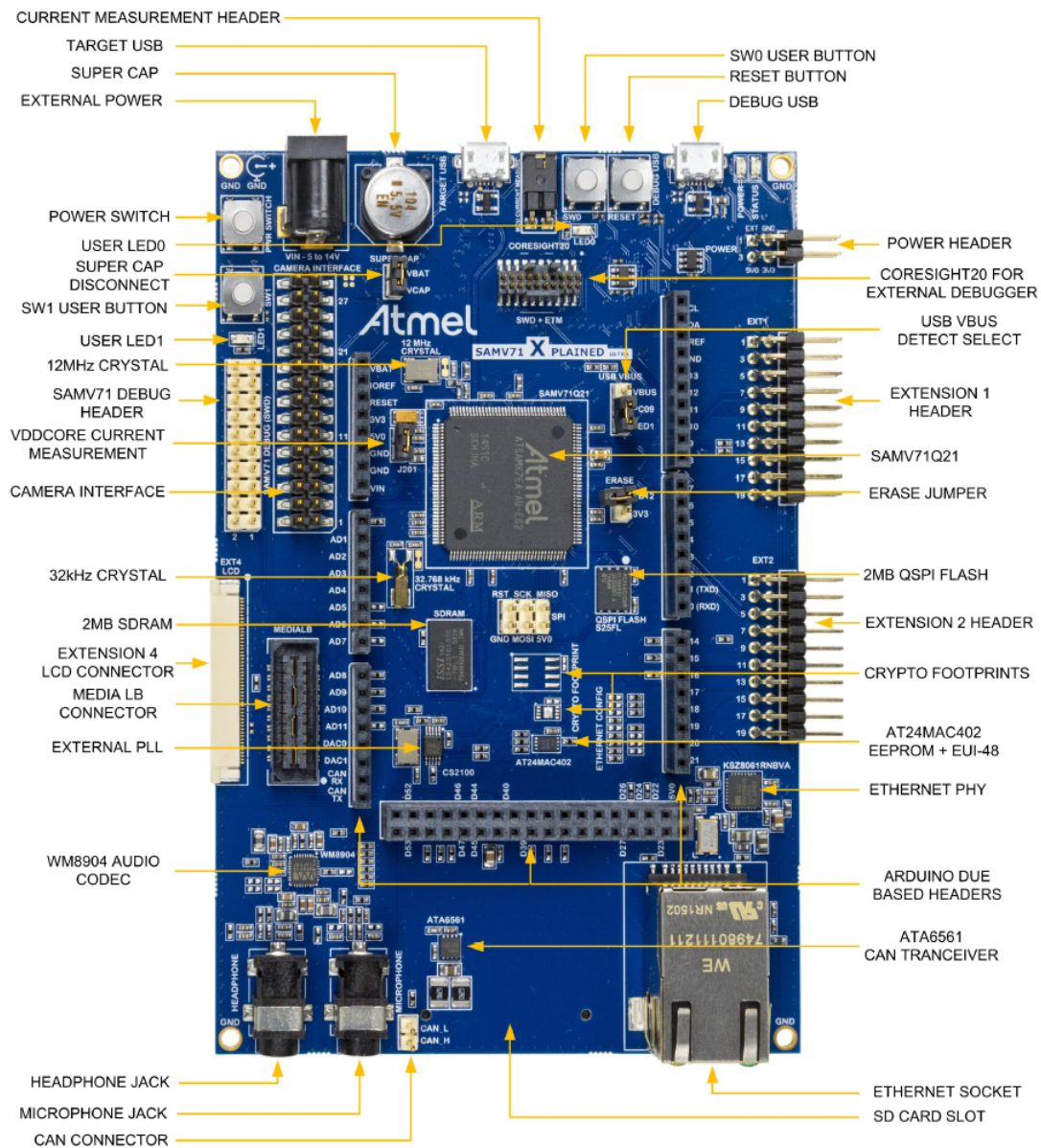


Figure 4.1.1: SAM V71 Xplained Ultra Evaluation Kit Overview [28].

The evaluation board comprises a lot of hardware components, but only the most important ones will be listed below:

- An ATSAMV71Q21 MCU;
- An embedded debugger (EDBG), used to program and debug the MCU;
- Two external crystal oscillators. A 32.768 kHz crystal for the slow clock signals and a 12 MHz for the speed clock signals;
- Three mechanical buttons: one reset button and two configurable buttons.
- Two yellow LEDs that can be programmed by the user;
- An external IS42S16100E-7B1, 512Kx16x2, 144MHz, SDRAM;

- An SD card connector linked to the High Speed Multimedia Card Interface (HSMCI) of the MCU;
- An external EEPROM connected to the I2C bus of the MCU;
- Arduino shield connectors on which the test PCB has been plugged.

Some of the above-mentioned components have been used to develop the software of the OBC. Since the test PCB is based on them, they will be mentioned again in the following section. But prior to that, the first tests performed using only the development board should be presented.

To learn the basic features of the board, the first tests were carried out by following online tutorials such as those presented by *Microchip* [23]. The first thing tested was to turn on and off one of the LEDs by pressing one of the buttons. The next thing was to add the RTOS in the previous application. Once that worked properly, several features of the RTOS were explored.

The first RTOS feature studied was the development of an application using multiple tasks. Then it was necessary to implement the communication between these tasks. Additionally, tests with semaphores and timers were performed. Finally, the last key functionality of RTOS that needed to be tested was related to interrupts.

It is also necessary to point out that while waiting for the reception of the test PCB, experiments were carried out with an EEPROM connected to the development board via a breadboard. That made it possible to carry out preliminary communication tests with the I2C bus. This way, when the test PCB was received, the communication with this bus could be implemented more quickly.

## 4.2 PCB compatible with the development board

A test PCB has been realized in order to develop and test the software of the OBC. It has been conceived to be plugged into the development board. For that reason, it is based on the Arduino Mega shape, as it is the case of the Arduino connectors of the development board. The 3D view of this testing PCB is shown in Figure 4.2.1.

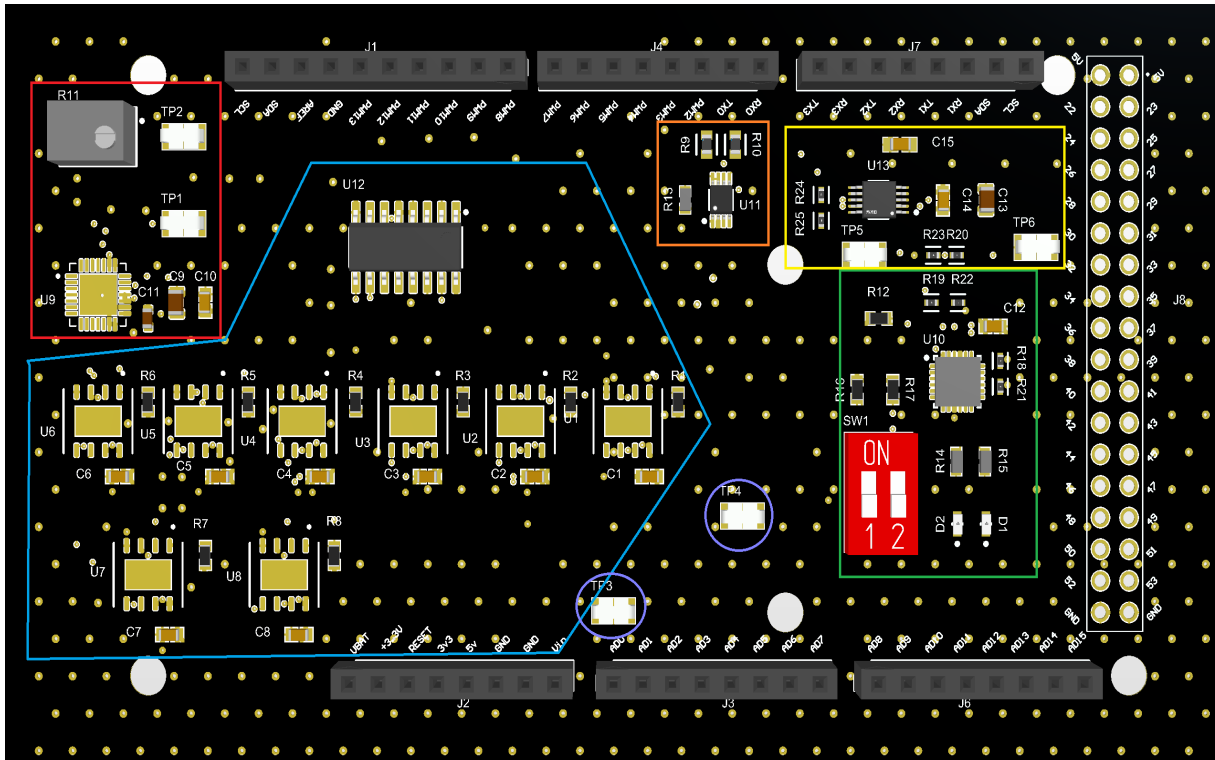


Figure 4.2.1: 3D view of the testing PCB that was plugged into the evaluation board.

The components are divided into different groups that have been highlighted with colored polygons in Figure 4.2.1. Each group will now be detailed.

The red rectangle contains the components linked to the external ADC. Therefore, it includes the MAX1128 [27] along with its required passive components. In addition, there are a potentiometer and two test points. One of the test points is directly connected to a channel of the MAX1128. It gives the opportunity to put any wanted external voltage on it to verify that the ADC result given by the MAX1128 is correct. The other test point is connected to the potentiometer and to another channel of the MAX1128. The potentiometer being itself connected to the 3.3V power source, its output is a fraction of this voltage. This gives an easy way to monitor that the MAX1128 ADC results are valid for different voltage values by just turning the potentiometer button and without the need for an external power source. The MAX1128 was thus tested in different ways.

The blue polygon comprises the components related to FRAMs. This includes the eight FM25V20A [14] with their passive components as well as the demultiplexer. The first test performed with this group was to select the correct FRAM CS line at the output of the demultiplexer. Indeed, as explained in section 2.2.1.2, the OBC manages simultaneously different CS lines with a demultiplexer. As this functionality was not implemented by the SPI driver management codes generated by *MPLAB*, they needed to be adapted. Some changes in these generated codes allowed the use of the demultiplexer for the selection of the FRAM CS line. Then, the first real test with these memories was a simple write of a text to a given address of a memory, followed by a read to the same address of the same memory. Once this writing and reading of text were working perfectly, further tests were carried out. The important function of FRAMs is to store and then read the values obtained by other components, such as the MAX1128 or the TCA9555. The next



test was therefore logically the writing and reading of ADC results from the MAX11128 into the memories. When this second test was working, a last check was made with the writing and reading of values coming from the TCA9555.

The orange rectangle contains the PCA9306 [29] and its passive components. This device is used as a dual bidirectional voltage-level translator for the I2C bus. Indeed, the I2C bus of the MCU has a voltage-level of 3.3V while the devices present in this PCB and with which the MCU communicates via the I2C bus have a voltage-level of 5V. The PCA9306 is consequently only used to enable other components to work. The components in question belong to the two groups that will now be analyzed.

The yellow rectangle includes the AD5272 [30] digital rheostats along with its passive components and two test points. The purpose of that group is to make some I2C communication with the MCU. Figure 4.2.2 shows the functional block diagram of the AD5272. The two test points are connected to the A and W pins of the AD5272. This block operates as follows: an input voltage is applied to one of the two test points and thus to the A or W pin of the AD5272, while the other test point is used to measure the output voltage of the AD5272. The output voltage will thus be a fraction of the input voltage. This fraction depends on the resistance value chosen for the AD5272. This resistance value is programmed through the I2C bus. Therefore, the tests performed consisted in programming the digital rheostats and checking that the output voltage was correct, given this chosen resistance value and the applied input voltage.

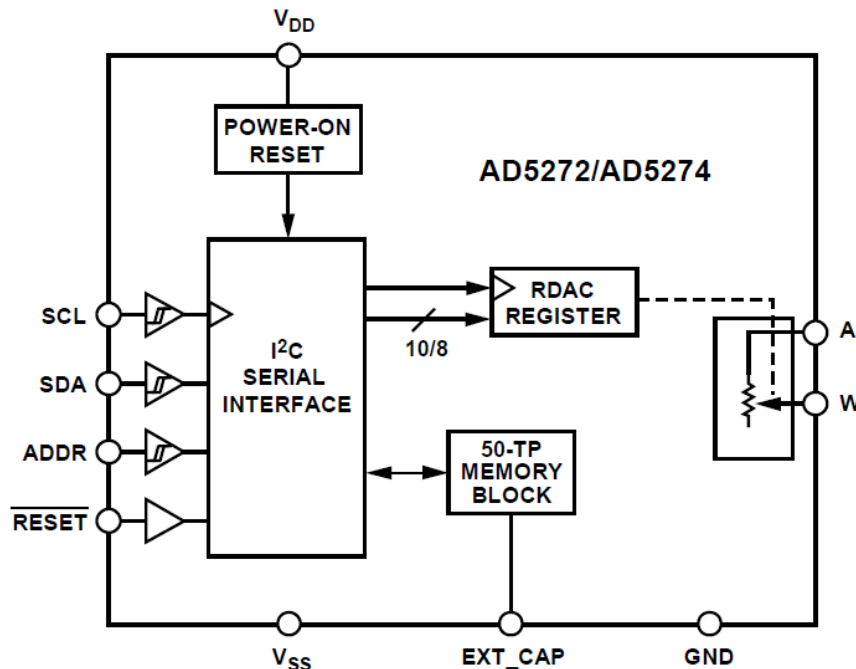


Figure 4.2.2: Functional block diagram of the AD5272 [30].

The green rectangle comprises the TCA9555 [26], its passive components, two LEDs, and two switches. This block involved two kinds of communications through the I2C bus. The first one was the writing of a value to the TCA9555 pins connected to the LEDs. In this case, the MCU was thus the sender and the TCA9555 the receiver. The first thing to do in order to reach that goal was to define the pins connected to the LEDs as outputs.

When the direction of the pins was correctly defined, it was necessary to put a "1" value in the bits corresponding to the two LEDs pins. Once the pins were configured and set as described, the two LEDs turned on. Some more tests were performed in order to turn only one of the LEDs on. The second type of communication was a reading of the values of the two pins connected to the switches. Here, the TCA9555 is the sender and the MCU is the receiver. The methodology was close to that of the first type of communication. Indeed, the pins were configured as input, which is the case by default, and then the value of those pins was read. The read value was then displayed on the terminal debugger to check if it was in accord with the switches positions. The tests have been performed with the switches in all possible configurations in order to be sure that everything was working well.

The two purple circles correspond to test points directly connected to pins of the connectors. They were used as a power source for some tests. Indeed, a simple bit set was sufficient to obtain a 3.3V voltage at those two test points. It was then only necessary to connect these points to the desired location in the circuit. For example, at one of the test points of the variable resistors.

Finally, it is worth noting that the connectors are represented as female connectors while in reality male connectors were soldered to the PCB in order to be compatible with the female connectors of the development board. This is simply because only the CAD models of female connectors were found online, but as their footprints are the same as the ones of the male connectors, it is not an issue.

# Chapter 5

## Conclusion

This project has led to significant progress in the development of the on-board computer of the CubeSat being developed by the University. As far as the hardware is concerned, the choice of the microcontroller and its external components was made. The MCU was chosen for its excellent general performance combined with the fact that it is available in different versions. Indeed, it is a huge advantage to be able to carry out all the development of the OBC with a standard COTS version and simply switch to a radiation-tolerant version when sending it into space. Beyond the choice of the MCU and its external components, the different memories of the OBC were also chosen. This choice took into consideration the harsh space environment as well as the necessary storage capacities and the time spent by data in the memories. Due to the current absence of a payload, it was decided not to make a definitive choice concerning the two SD cards. A first study has been carried out and their location on the PCB is already defined. Only the model that will be the most suitable for the payload, once this one has been determined, remains to be chosen based on the results obtained in this work. A power control system has also been set up. Its objective is twofold. Firstly, it is used to protect the downstream circuits from overvoltage, undervoltage and overcurrent. Secondly, it is used to evaluate the total power consumed by the on-board computer. In addition to the choice of the components, a PCB has been designed. This PCB has been developed so as to leave some space for the possible addition of new components during the future development of this project. Indeed, once the payload will be established, it might be necessary to add some components to the OBC.

In addition to the choice of the components and the realization of the PCB, codes allowing the OBC to carry out some functions have also been developed. These ones are based on an RTOS. This architecture has been chosen because it is the most widespread architecture in the space domain. Moreover, given the current absence of payloads, it was necessary to make sure that future developments would not require modifying the entire code. Using an RTOS prevents this from happening. This software was made using the *MPLAB X IDE*. This program makes it very practical to develop applications around the chosen microcontroller. Indeed, it provides many pre implemented modules such as *FreeRTOS*, an open-source RTOS. Codes enabling communication with the FRAMs located on the OBC as well as with several sensors contained on the EPS board have been implemented. Finally, a code to demonstrate the functioning of the above-mentioned codes has also been implemented to simplify future works related to this thesis. This code also handles some security aspects such as the WDT that is used to ensure that the

program does not get stuck in a deadlock.

A series of tests were also carried out to ensure the proper functioning of the codes implemented for the OBC. In order to perform these tests, the development board designed for the chosen MCU was used. A test PCB created to be compatible with this development board was created for further testing. These tests verified that everything was working as it should.

For the further development of the on-board computer, it will first be necessary to know the payload included in the CubeSat. This will allow to make the final choices of components and to complete the OBC PCB. After that, the OBC software will have to be developed further so that it can perform all the required tasks. The last step will be to replace the standard MCU with the radiation-tolerant version before sending the CubeSat into space. It could also be interesting to include some modern features in the CubeSat. Indeed, technologies such as artificial intelligence and machine learning are increasingly used in CubeSat. For example, ESA has worked on a nanosatellite containing artificial intelligence techniques to send back to earth only the interesting images that have been pre-processed in the satellite. This avoids sending many useless images to earth.

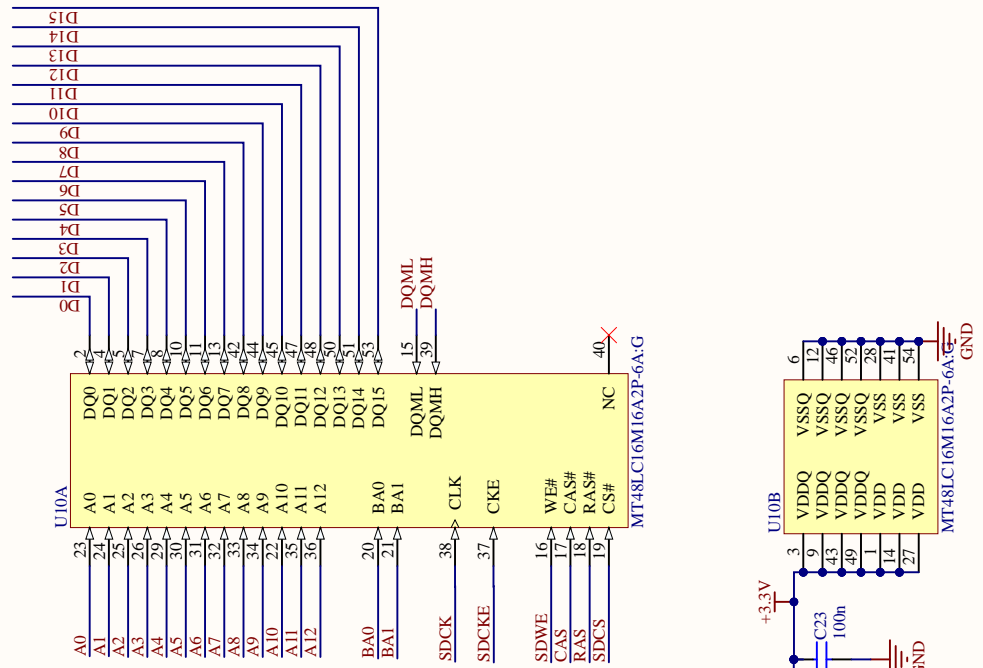
To conclude, this work has been a huge opportunity for me. The space domain is a very interesting domain but quite difficult to access. I am very happy to have had the opportunity to already gain a first experience in this field. Furthermore, this work allowed me to familiarize myself with the Altium Designer software as well as with RTOS-based systems.

# Appendices

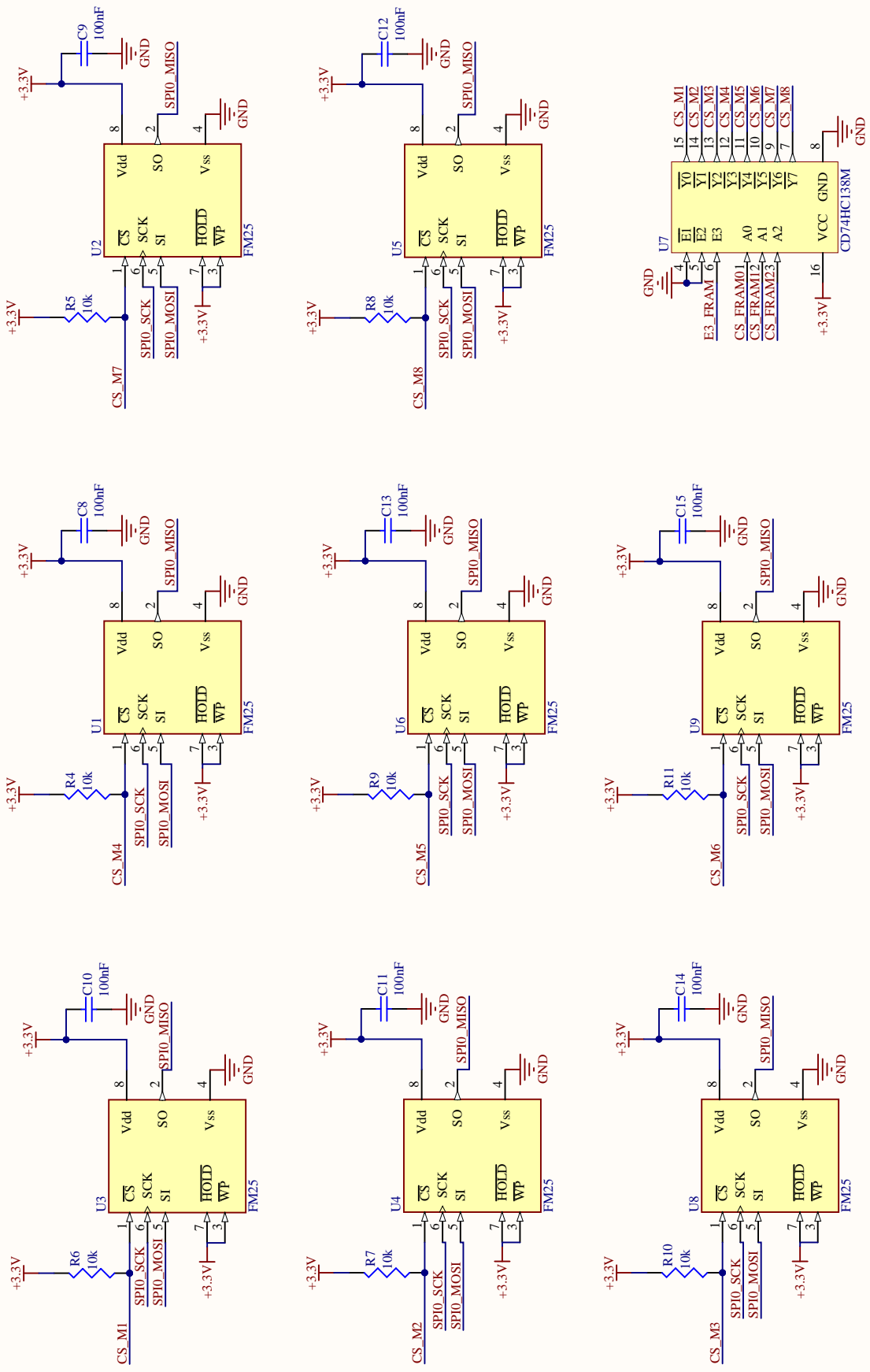
# Appendix A

## Final PCB Schematics

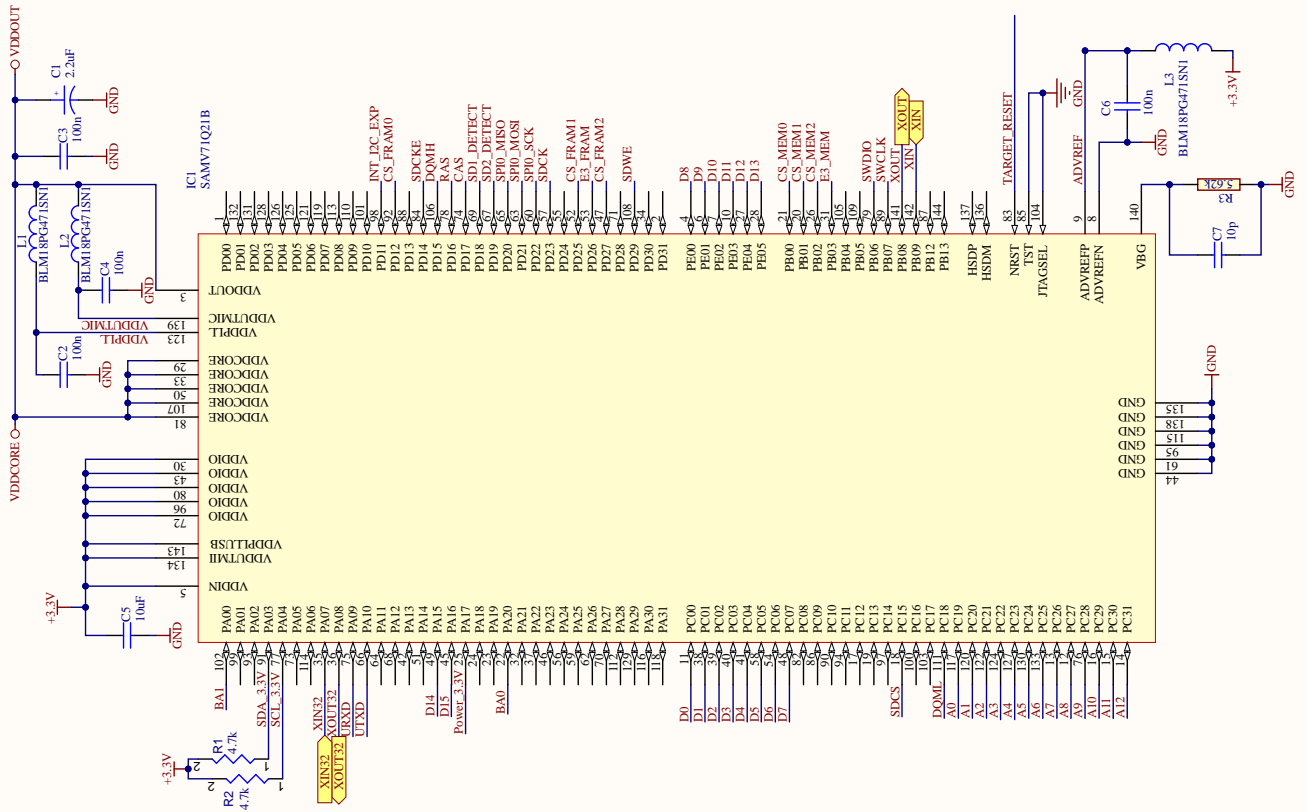
### SDRAM



# FRAM

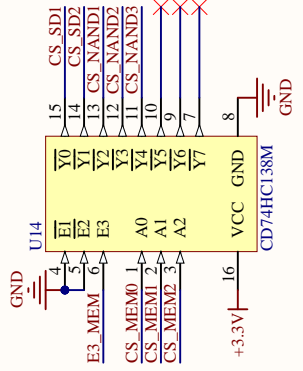
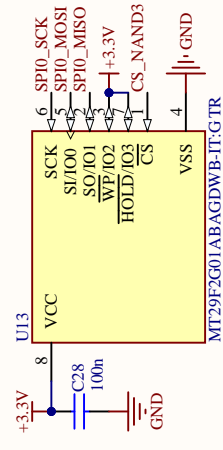
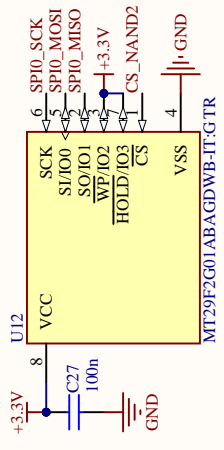
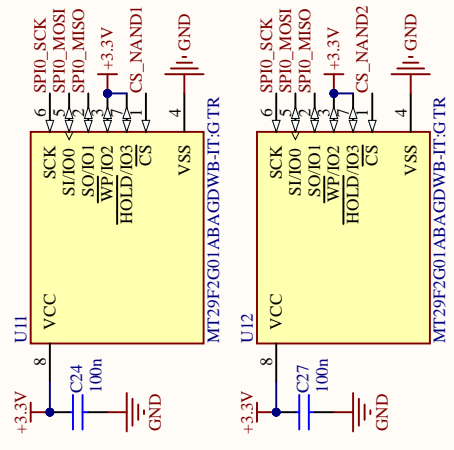
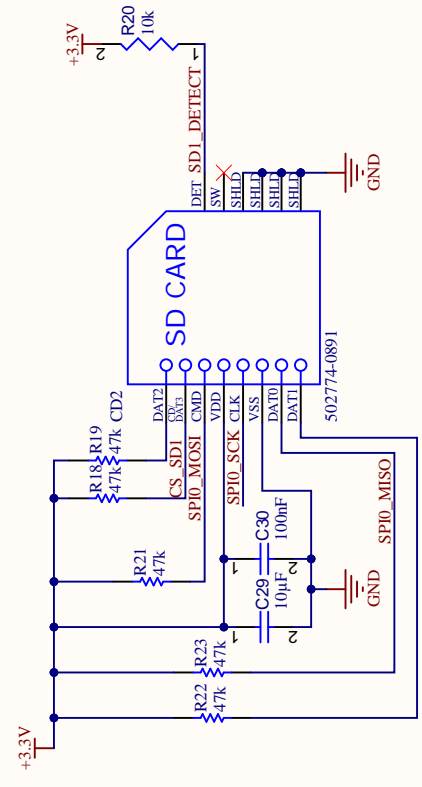
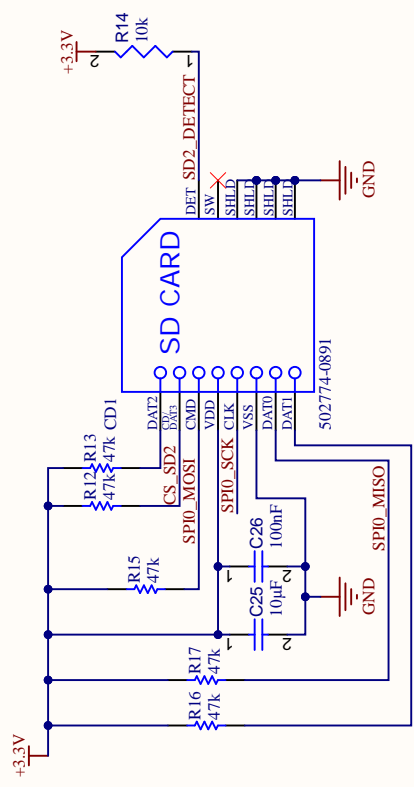


# SAMV71Q21



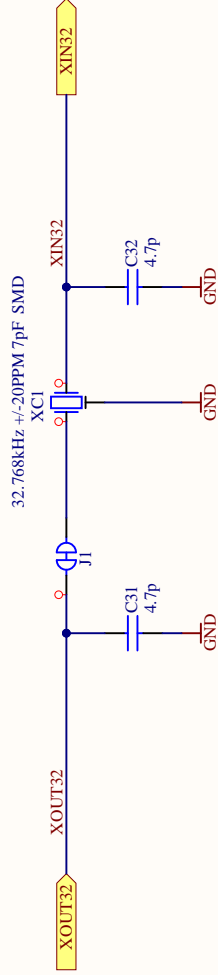


# NAND + SD

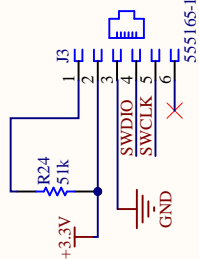
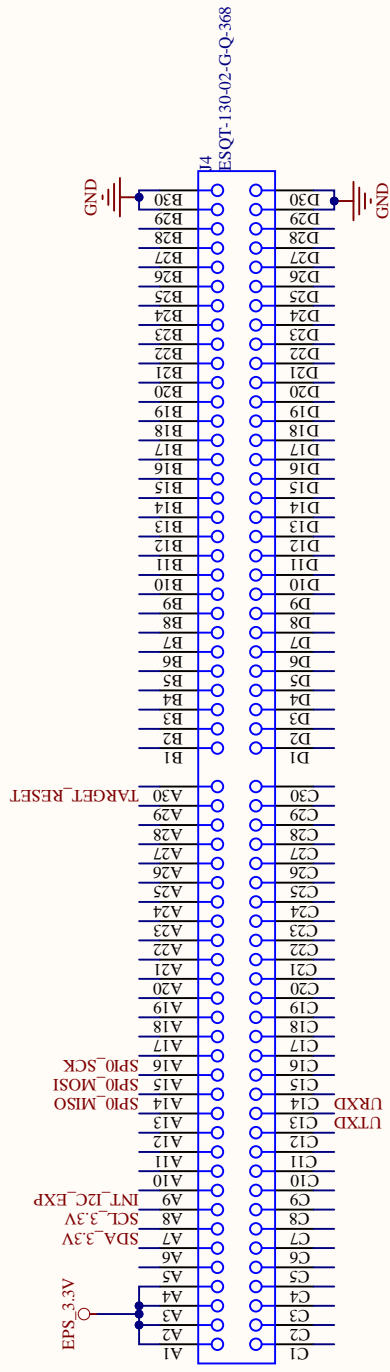
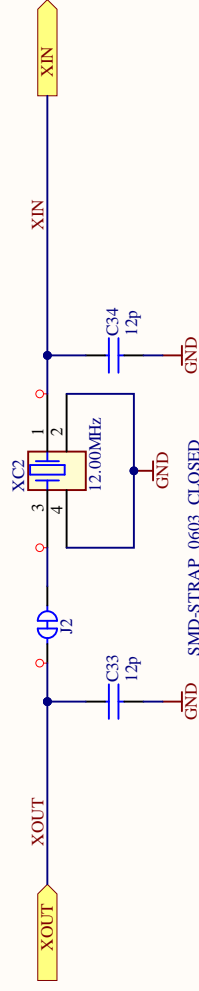


# CRYSTALS + CONNECTORS

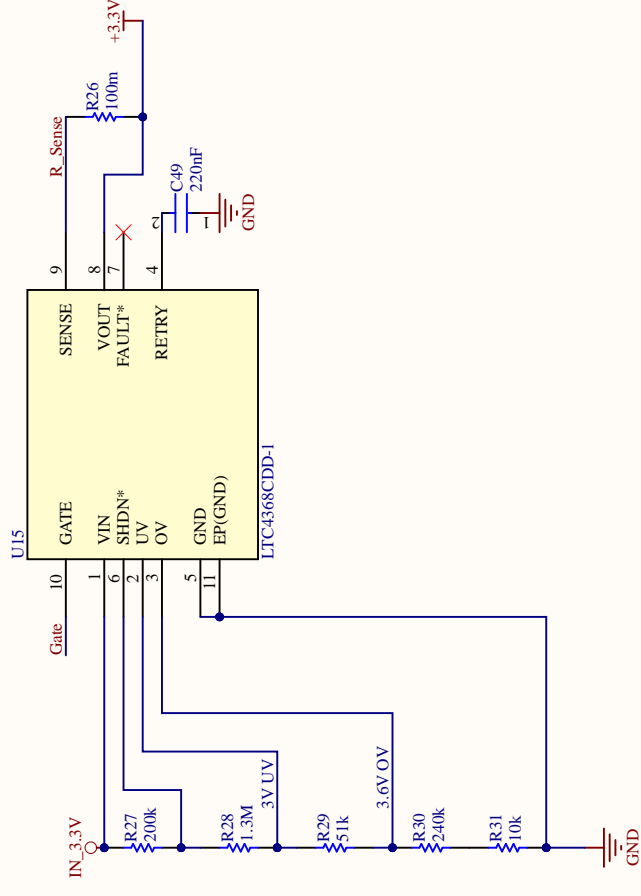
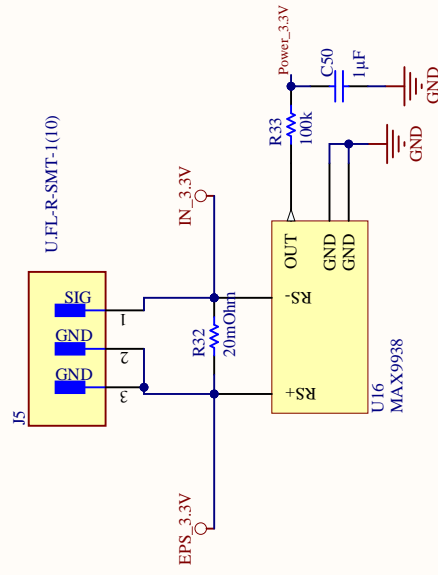
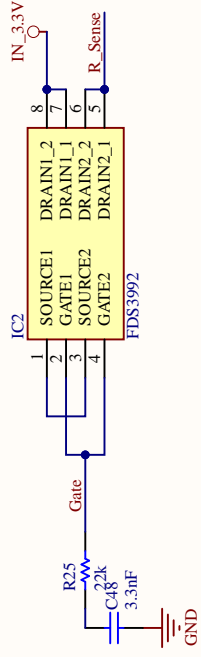
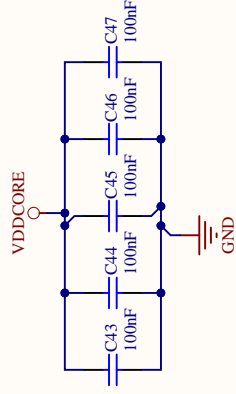
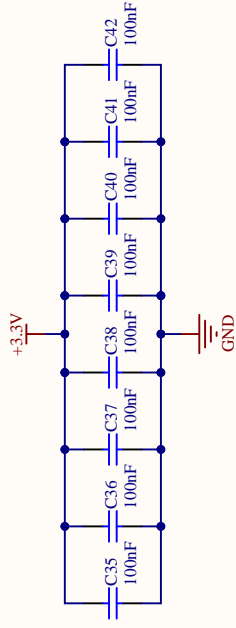
### 32.768 kHz Crystal



### 12 MHz Crystal



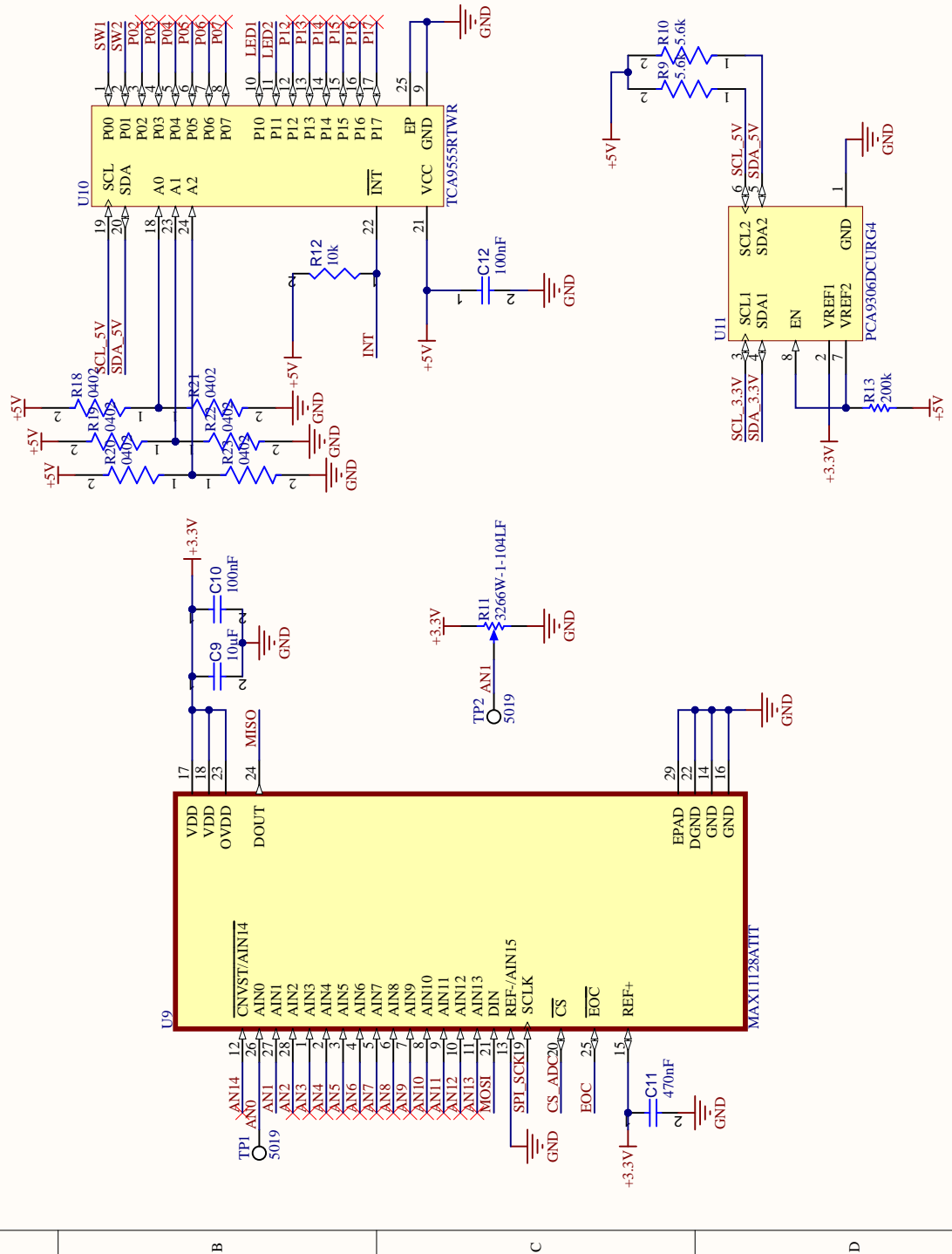
# POWER



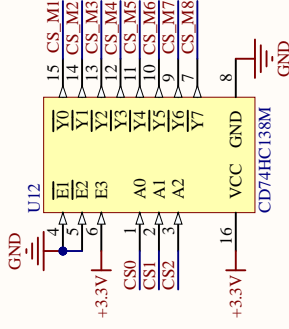
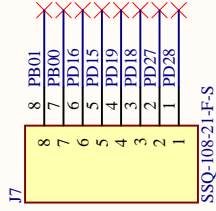
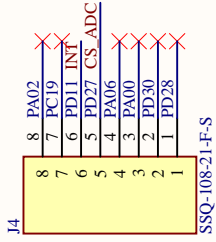
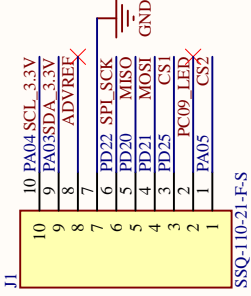
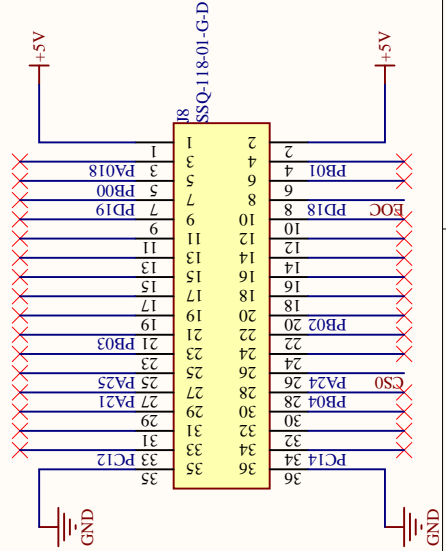
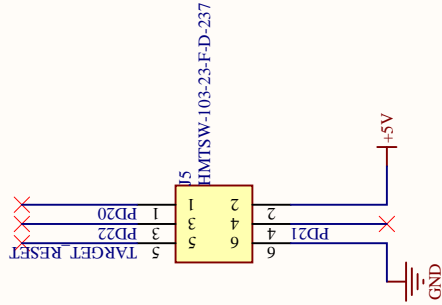
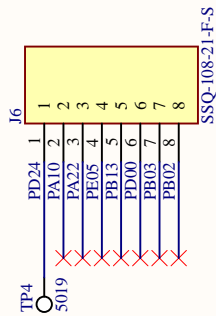
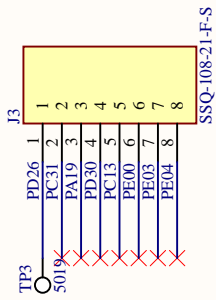
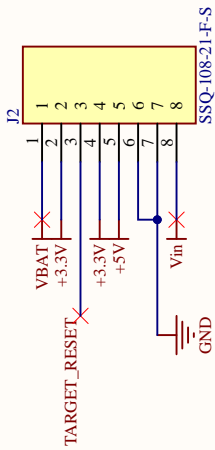
# Appendix B

## Test PCB Schematics

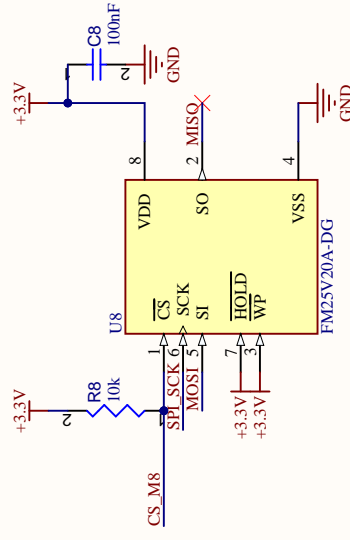
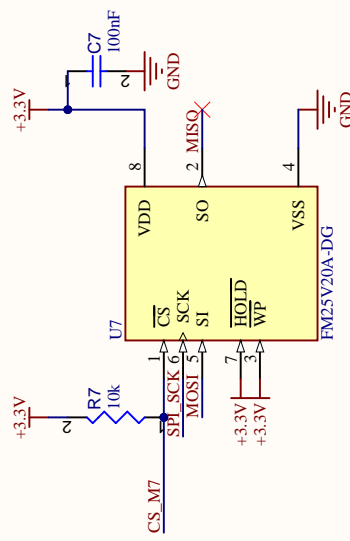
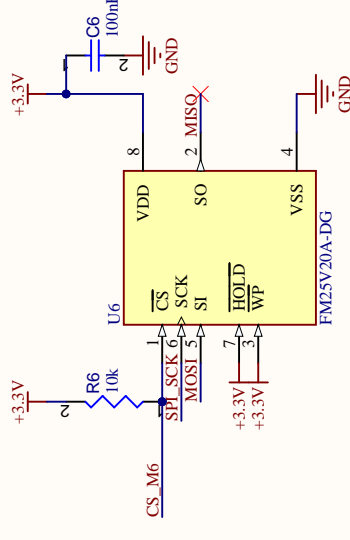
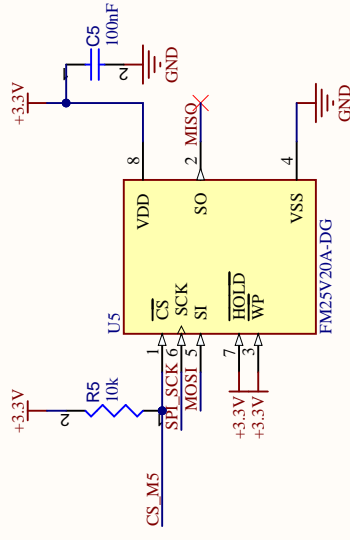
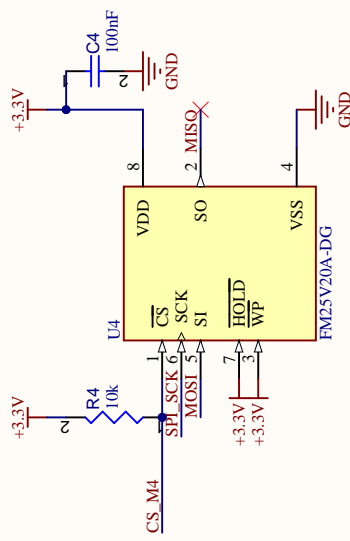
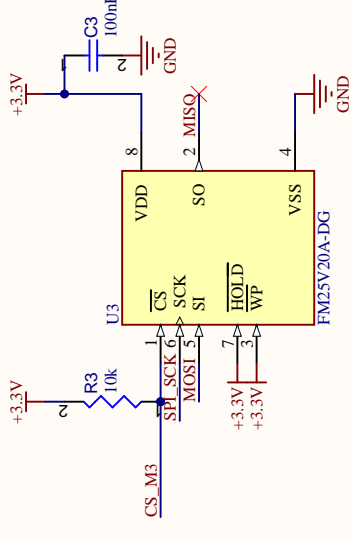
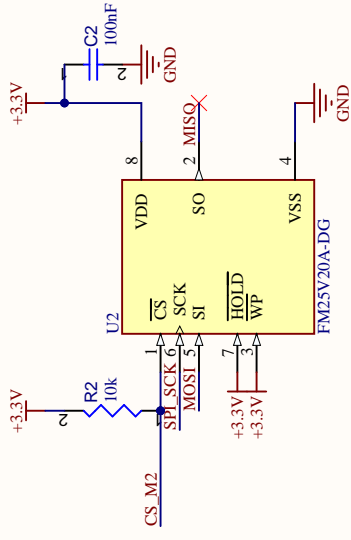
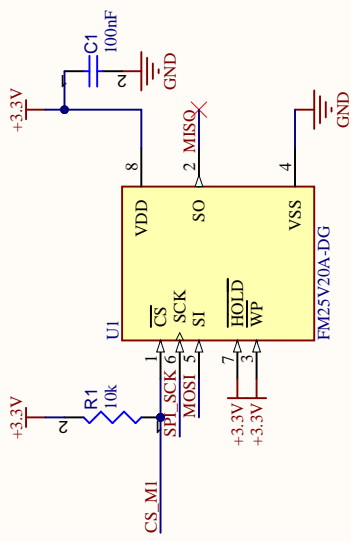
### ADC + I2C BUS EXPANDER



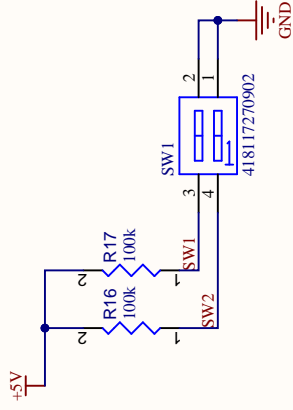
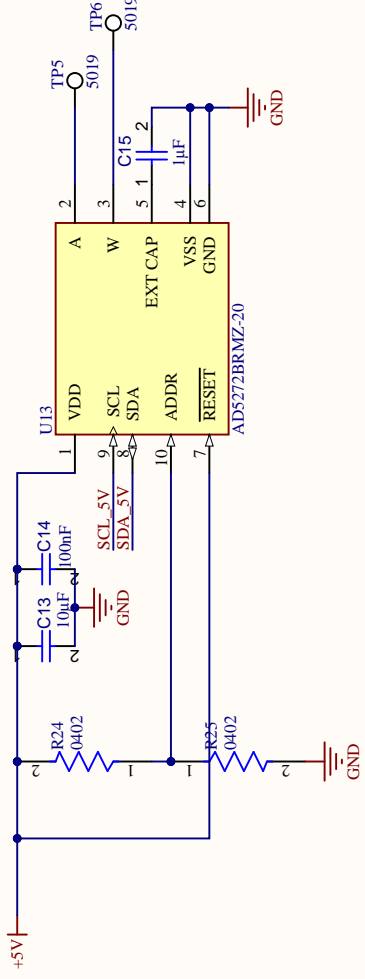
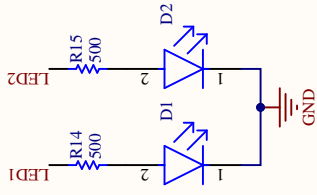
# CONNECTORS



# FRAM



# LEDs + SWITCHES + DIGITAL POTENTIOMETER



# Appendix C

## Codes Git link

<https://gitlab.uliege.be/A.Horbach/tfe>



# Bibliography

- [1] Atkinson et al. “CubeSat 101, Basic Concepts and Processes for First-Time CubeSat Developers”  
In: *NASA* (Oct. 2017).
- [2] The European Space Agency. *Surviving extreme conditions in space*  
URL: [http://www.esa.int/Science\\_Exploration/Space\\_Science/Extreme\\_space/Surviving\\_extreme\\_conditions\\_in\\_space?fbclid=IwAR2P0wWBpsnNyQ-8bpIe44Pn0-m5PUFnWjWQBYV6LUCmprZqbPhpW2MqQMI](http://www.esa.int/Science_Exploration/Space_Science/Extreme_space/Surviving_extreme_conditions_in_space?fbclid=IwAR2P0wWBpsnNyQ-8bpIe44Pn0-m5PUFnWjWQBYV6LUCmprZqbPhpW2MqQMI). (Accessed: 18.11.20).
- [3] Microchip. *Scale Space Applications with COTS-to-Radiation-Tolerant and Radiation-Hardened Arm® Core MCUs*  
URL: <https://www.microchip.com/pressreleasepage/cots-to-radiation-tolerant-and-radiation-hardened-arm-core-mcus>. (Accessed: 15.10.20).
- [4] *32-bit ARM Cortex-M7 MCUs with FPU, Audio and Graphics Interfaces, High-Speed USB, Ethernet, and Advanced Analog*  
ATSAMV71Q21 Datasheet. Rev.E. MICROCHIP TECHNOLOGY. Dec. 2020.
- [5] Eric Peña and Mary Grace Legaspi. “UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter”  
In: *Analog Devices* (Dec. 2020).
- [6] *Radiation-Tolerant 32-bit Arm® Cortex®-M7 MCU*  
SAMV71Q21RT Datasheet. Rev.C. MICROCHIP TECHNOLOGY. Aug. 2019.
- [7] *Rad-Hard 32-bit Arm® Cortex®-M7 Microcontroller for Aerospace Applications*  
SAMRH71 Datasheet. Rev.D. MICROCHIP TECHNOLOGY. May 2020.
- [8] Predictable Designs. *Comparison of Memory Types Available for Your Product*  
URL: <https://predictabledesigns.com/comparison-of-memory-types-available-for-your-product/>. (Accessed: 07.02.21).
- [9] Electronics Notes. *Semiconductor Memory Types & Technologies*  
URL: <https://www.electronics-notes.com/articles/history/>. (Accessed: 23.02.21).
- [10] *256Mb: x4, x8, x16 Automotive SDRAM*  
MT48LC16M16A2 Datasheet. Rev.D. MICRON TECHNOLOGY. June 2018.
- [11] Jagan Singh Meena et al. “Overview of emerging nonvolatile memory technologies”  
In: *Nanoscale Research Letters* 9.1 (Sept. 2014), p. 526. ISSN: 1556-276X. DOI: 10.1186/1556-276X-9-526. URL: <https://doi.org/10.1186/1556-276X-9-526>.
- [12] *F-RAM™ Technology Brief*  
Rev.B. Cypress. June 2016.

- 
- [13] *FRAM FAQs*  
Texas Instruments. 2020.
- [14] *2-Mbit (256 K × 8) Serial (SPI) F-RAM*  
FM25V20A Datasheet. Rev. 1. CYPRESS. Sept. 2019.
- [15] *2Gb 3.3V x1, x2, x4: SPI NAND Flash Memory*  
MT29F2G01ABAGDWB Datasheet. F. MICRON. Sept. 2016.
- [16] R. Kingsbury et al. “TID Tolerance of Popular CubeSat Components”  
In: *2013 IEEE Radiation Effects Data Workshop (REDW)*. 2013, pp. 1–4. DOI: 10.1109/REDW.2013.6658220.
- [17] *Tuning Fork Crystal 30 kHz-200 kHz*  
MS1V-T1K. Rev.4. Micro Crystal. Mar. 2007.
- [18] *Automotive & Industrial Grade Ceramic SMD Miniature Crystal*  
ABM3BAIG Datasheet. Abracon. May 2014.
- [19] Microchip Technology. *MPLAB® ICD 4*  
URL: <https://microchipdeveloper.com/icd4:start>. (Accessed: 23.04.21).
- [20] *100V UV/OV and Reverse Protection Controller with Bidirectional Circuit Breaker*  
LTC4368 Datasheet. Rev.C. ANALOG DEVICES. Apr. 2021.
- [21] *nanoPower, 4-Bump UCSP/SOT23, Precision Current-Sense Amplifier*  
MAX9938. Rev.7. Maxim Integrated. Apr. 2017.
- [22] Bernard Boigelot. *Transparencies of INFO0064 Embedded systems*  
Ed. by University of Liège. 2020. URL: <https://people.montefiore.uliege.be/boigelot/courses/embedded/>.
- [23] Microchip. *Getting Started with Harmony v3 Drivers on SAM E70/S70/V70/V71 MCUs Using FreeRTOS*  
URL: <https://microchipdeveloper.com/harmony3:same70-getting-started-tm-drivers-freertos>. (Accessed: 01.04.2021).
- [24] Sal Afzal. *I2C Primer: What is I2C?*  
Ed. by Analog Devices. 2021.
- [25] Piyu Dhaker. *Introduction to SPI Interface*  
Ed. by Analog Devices. 2021.
- [26] *Low-Voltage 16-Bit I2C and SMBus I/O Expander with Interrupt Output and Configuration Registers*  
TCA9555 Datasheet. Rev.E. TEXAS INSTRUMENTS. July 2016.
- [27] *1Msps, Low-Power, Serial 12-/10-/8-Bit, 4-/8-/16-Channel ADCs*  
MAX11128 Datasheet. Re.4. MAXIM INTEGRATED. Feb. 2019.
- [28] *SMART ARM-based Microcontrollers*  
SAM V71 Xplained Ultra. Rev.42408C. Atmel. Sept. 2015.
- [29] *Dual Bidirectional I2C Bus and SMBus Voltage-Level Translator*  
PCA9306 Datasheet. Rev.M. TEXAS INSTRUMENTS. Apr. 2019.
- [30] *1024-/256-Position, 1% Resistor Tolerance Error, I2C Interface and 50-TP Memory Digital Rheostat*  
AD5272 Datasheet. Rev.D. ANALOG DEVICES. Mar. 2013.
-