Semester Project, Summer 2007

Phase B/C

ADCS System Engineer

ADCS System and Hardware

Prepared by:

Hervé Péter-Contesse

Professor:

Maurice Borgeaud

Advisor:

Muriel Noca

•

EPFL
Lausanne
Switzerland

•

23/06/2007

•

RECORD OF REVISIONS

ISS/REV	Date	Modifications	Created/modified by
1/0	3/06/2007	First release / Draft	Hervé Péter-Contesse
1/1	10/06/2007	Completion	Hervé Péter-Contesse
1/2	18/06/2007	Grammar correction and completion. Requirements added	Hervé Péter-Contesse
1/3	23/06/2007	Grammar correction and completion. Final version	Hervé Péter-Contesse

PROJECT: SwissCube Satellite PHASE: B/C	WP REF: 4600
WP Title: SwissCube ADCS system engineer	**Sheet 1 of 3**
Responsible: Space Center **Collaborator/assistant:** Muriel Noca **Student:** Hervé Péter-Contesse	**Issue Ref: 1** **Issue Date:** **09-03-07**
Start date: 12-03-07 **End date:** 30-06-07	
WP Manager: M. Noca	

Introduction

This Work Package summarizes the work expected from the student during phase B/C (semester's project) of the SwissCube Project. The expected duration of the work is as stated above.

The objectives of this task will be four fold:

- Review the task description;
- Plan a schedule for your work, and review it with your project assistant;
- Perform the tasks and keep the project informed of the status;
- Provide the outputs and deliverables listed at the end of this document.

The student will report for all technical matters to the Lab assistant (when applicable) and project system engineer (assistant) assigned at the beginning of the semester.

The student will have to participate in the design meetings related to his topic (mechanical, electronics, data).

The times open for discussion with the project assistant are:

- Monday afternoons in ELD 014
- Friday afternoons in ELD 014.

Deadlines are summarized here:

- Kick-off meeting (mandatory): March 15, 17h00 in ELD 010
- Mid-term review: May 2, time TBD ELD 010
- Draft report due: June 3
- Final report due: June 17
- Presentation due: July 5

Task Description:

During last semester project, the hardware (sensors, actuators and controller) for the SwissCube attitude determination and control system (ADCS) has been selected, and preliminary tests have been made. The goal of this project is to optimize the hardware electronics PCB, program the micro-controller and test the board with all sensor and actuator components. This project will also require the student to understand the control and determination aspects of the ADCS, verifying the current specifications, providing technical overview of the proposed sensor and actuator characterization tests. This project will be done closely with the project on the determination algorithms.

This task includes :

- Review all documentation regarding the ADCS Hardware part of the ADCS subsystem; Concentrate on the understanding of the perturbation calculations and understanding of the science and project requirements and needs;
- Review current state of the hardware, PCB design and programmed functionalities;
- Program the ADCS micro-controller for each missing functionalities;
- Characterize operations of the magneto-torquers;
- Integrate magnetometers and characterize operations;
- Follow characterization tests of the gyroscopes and sun sensors;
- Provide recommendations for, or new design of, the ADCS HW board;

In support of and in coordination with the attitude determination algorithms, the student shall:

- Identify the Earth magnetic model to use and provide, if necessary a simplification for use in the determination algorithms;
- Provide a model of the sun-sensors, gyroscopes and magnetometers for inclusion into the determination algorithms; (Interface Control Document)
- Provide a model of the magneto-torquers for inclusion into the control algorithms; (Interface Control Document)
- Document all activities.

Inputs:

- "SwissCube Phase B Mission and System Overview" (S3-B-SET-1-2-Mission_System_Overview.pdf);
- "SwissCube Project Specifications Document" (S3-B-SE-1-0-Level_1and2_specifications.pdf);
- "SwissCube System Specifications Document" (S3-B-SE-1-0-Level_3_SSR.pdf);
- "SwissCube Your_Subsystem Specifications Document" (S3-B-SE-1-0-Level_4_SUBSYSTEM_specifications.pdf);
- S3-B-ADCS-1-4-ADCS_HW.pdf;

- S3-B-ADCS-1-3-Magnetic_sensor.pdf;
- SwissCube document templates;
- Reference: "Space Mission Analysis and Design", Larson & Wertz

---

Outputs:

- Performance model of all sensors and actuators;
- Tests of the ADCS micro-controller functionalities;
- Test of the ADCS board with as many sensors and actuators as available;
- Updated design of ADCS-HW PCB;
- Interface control document for the ADCS-HW PCB;

All analyses should include documentation of the assumptions.

Deliverables:

- A final report including a short description of all outputs.
- A presentation at the Delta-PDR that will conclude Phase B.
- A disk containing all analysis and documentation files for records.

1 INTRODUCTION

The main purpose of this report is to present the development, the design and test of the Attitude Control and Determination System (ADCS) hardware of the SwissCube satellite. The main part of this Semester Project was to continue two previous projects (phase B): the 1st has been done by Bastien Despont on the whole ADCS system and hardware (see [R1]) and the 2nd by Vasco Vitanov on the magnetometers (see [R2]). A second part was to develop an Earth's Magnetic Field model, sensors and actuators models in order to use them in the control and determination algorithms. The part concerning the Earth's magnetic field model is not presented in this report, but in [R3].

Based on the CubeSat program started by the Stanford University and the California Polytechnic State University (CalPoly) the SwissCube is the first entirely Swiss picosatellite program. The primary objective of developing this satellite is to provide a dynamic and realistic learning environment for undergraduates, graduates and to improve the development of small satellite technologies. The secondary objective is to house a science payload in order to take optical measurements and characterize the Nightglow phenomenon (see Figure 1-1) over all latitudes and longitudes for at least a period of 3 months, with extended science mission duration up to 1 year.

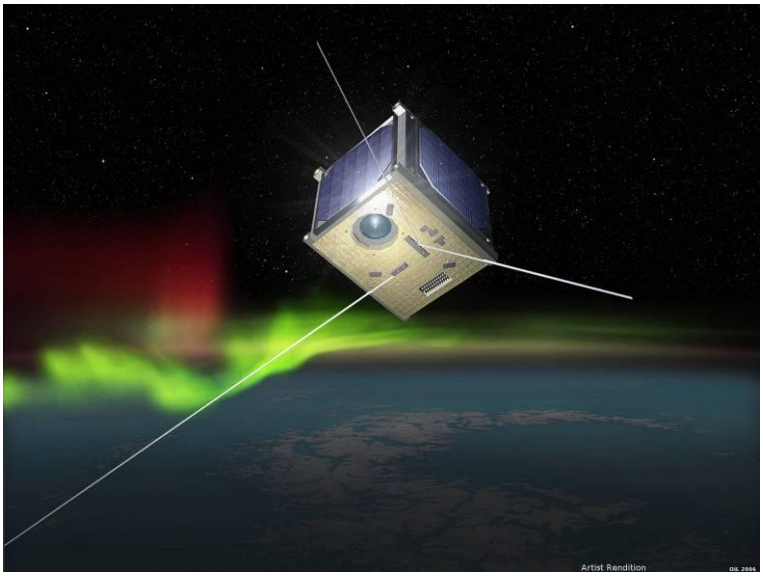


Figure 1-1 : Artist View of the SwissCube

The ADCS must determine the position, velocity and orientation and also control the satellite. The determination is the most important point, because we must to know where the payload is pointing to characterise the nightglow phenomenon. The major issue of the control is to reduce the spinning rate of the satellite after the launch and, if it is technically possible, to orient the payload in a precise direction in order to take photographs.

The ADCS algorithms run on the CDMS microcontroller (which is the main computer of the SwissCube). The ADCS board itself is in charge of the sensor readings and actuators control.

Figure 1-2 shows the architecture of the ADCS.

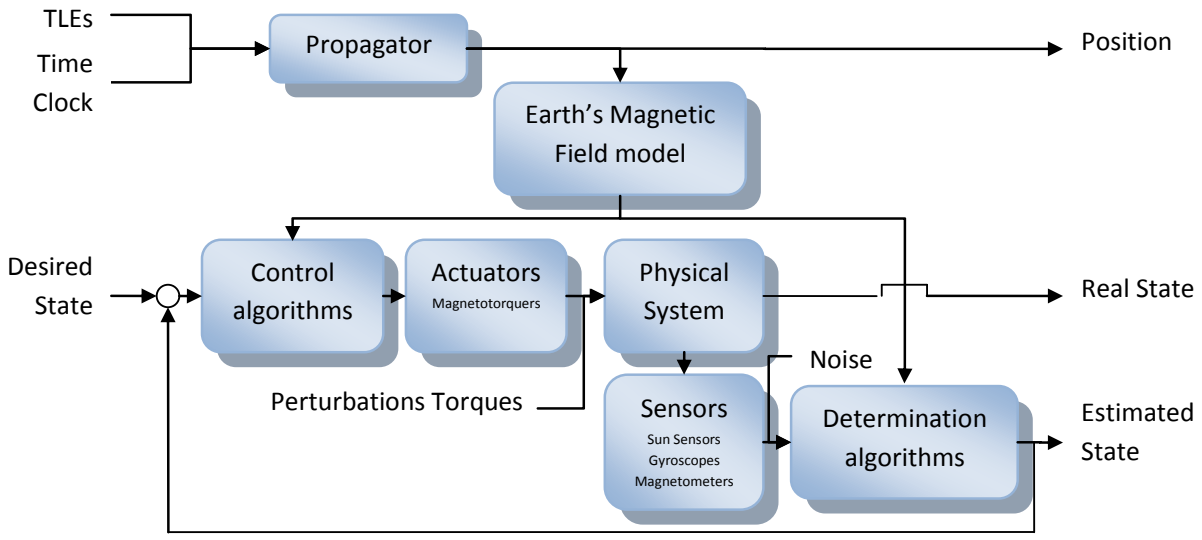


Figure 1-2 : ADCS functional diagram

The sensors used for the ADCS system are:

- A 3-axis magnetometer (MM) to measure the Earth's magnetic field (EMF) intensity and direction
- A 3-axis gyroscopes (GYR) to measure the spinning rate for each axis
- 6 Sun sensors (SS) to find the direction of the sun
- Temperature sensors to compensate the temperature drift of the other sensors

The actuators are:

- 3 (or 2) magnetotorquers (or coils; MT) to produce a torque thanks to their interaction with the Earth's magnetic field
- *Passive actuators such as permanents magnets* (not yet decided)

The algorithms can be separated in 4 main parts:

- The propagator which computes the position and the orbit of the satellite
- The Earth's magnetic field model which compute the magnetic field intensity and direction
- The control algorithms
- The determinations algorithms to filter the sensors measurements (with an Extended Kalman Filter) and then determine the satellite

This report will mainly present the design and test of the ADCS main board, magnetometer and magnetotorquers. The design and test of the gyroscopes, sun sensors, propagator and determination algorithms have been done by other students and are respectively presented in [R4], [R5], [R6] and [R7].

2 DESIGN REQUIREMENTS

ADCS requirements, level 4 and 5

Level 4

1. Functional

1.0 General

4_ADCS_10_01 **Primary function**

The ADCS shall provide an interface with CDMS. This includes the following functions:
Receive cmd from and send TM to CDMS
Power and command actuators
Power, command and receive TM, store and format data from sensors

To fulfill mission objective 3

3_SSR_10_02

4_ADCS_10_02 **Determination**

The ADCS subsystem shall provide determination of the attitude of the satellite.

To fulfill science objective

3_SSR_24_02

4_ADCS_10_03 **HK**

The ADCS shall provide HK to determine its health.

To monitor health of the Space system

3_SSR_10_06

4_ADCS_10_04 **Attitude control**

The ADCS shall provide a control for the attitude of the satellite

Pointing stability

3_SSR_24_04

2. Mission and Performance

2.1 Unit modes

4_ADCS_21_01 **Subsystem modes**

The ADCS system shall have the following modes: "NOMINAL", "SENSOR", "STAND-BY" and "OFF" modes.

To fulfill its function

4_ADCS_10_01

4_ADCS_21_02 **"OFF" mode**

In this mode the whole subsystem shall be turned off.

OFF mode definition

4_ADCS_21_01

4_ADCS_21_03 **"STAND-BY" mode**

In this mode only the microcontroller shall be on and waiting for a command.

Stand-By mode definition

4_ADCS_21_01

4_ADCS_21_04 **"SENSOR" mode**

In this mode only the microcontroller and all the sensors shall be turned on. Actuators shall be switched off.

Attitude determination

4_ADCS_10_02

4_ADCS_21_05 **"NOMINAL" mode**

In this mode the microcontroller, the sensors and the actuators shall be on.

NOMINAL mode definition
Attitude determination and control.

4_ADCS_10_04

2.3 Unit state H/W performance

4_ADCS_23_01 **Consumption in "OFF" mode**

The ADCS shall not consume any power in "OFF" mode.

SSR power budget

4_ADCS_21_02

4_ADCS_23_02 **Consumption in "STAND-BY" mode**

The ADCS shall consume less than [30] mW in "STAND-BY" mode.

SSR power budget

4_ADCS_21_03

4_ADCS_23_03 **Consumption in "SENSOR" mode**

The ADCS shall consume less than [90] mW in "SENSOR" mode.

SSR power budget

4_ADCS_21_04

4_ADCS_23_04 **Consumption in "NOMINAL" mode**

The ADCS shall consume less than [250] mW in "NOMINAL" mode.

SSR power budget

4_ADCS_21_05

4_ADCS_23_06 **Attitude determination**

The ADCS subsystem shall provide determination data of the attitude of the satellite with an accuracy of [10°] for each

MSR

3_SSR_24_02

4_ADCS_23_07 **Attitude control**

The ADCS subsystem shall provide a control for the attitude of the satellite with a stability [3°] deg/s in any direction.

To ensure success of science mission (SwissCube MEMO, Pointing Accuracy and Stability Requirements by Daniel Hakansson)

3_SSR_24_04

2.4 Unit state S/W performance

4_ADCS_24_01 **Command data**

The ADCS shall accept the command signal for the actuators control electronics at least once every [10] seconds.

Control algorithm

4_ADCS_10_01

4_ADCS_24_02 **Data transmission**

The ADCS shall be able to send attitude and HK sensors data at specific requests from the CDMS.

To fulfill its function

4_ADCS_10_01

2.5 Reliability and redundancy

4_ADCS_25_02 **Latch up protection**

The ADCS shall be designed with separate latch-up protection circuits.

To mitigate SEL.

3_SSR_25_03

4_ADCS_25_03 **SEU**

ADCS H/W and S/W design for critical functions shall mitigate possible SEUs.

Protection

3_SSR_25_01

4_ADCS_25_04 **Tests**

Reliability of the electrical systems shall be demonstrated by tests.

Tests requirements

3_SSR_25_02

3. Design

3.1 Constraints

4_ADCS_31_01 **Outgassing**

The ADCS materials shall have a Total Mass Loss (TML) ≤ 1 % and a Collected Volatile Condensable Material (CVCM) ≤ 0.1 %.

Arianespace user's manual.

3_SSR_31_15

4_ADCS_31_02 **Launch date**

The ADCS shall be ready for integration in the Engineering Qualification Model by Decembre 2007.

Launch date deadline

3_SSR_31_09

4_ADCS_31_03 **Contamination**

Nasa approved materials shall be used whenever possible to prevent contamination of other spacecraft during integration, testing and lauch.

CalPoly spec.

3_SSR_31_13

3.2 Thermal

4_ADCS_32 _01 **Temperature**

The ADCS shall be capable of measuring its temperature.

Thermal analysis.

3_SSR_32_03

4_ADCS_32 _02 **Temperature control**

The ADCS shall have a passive temperature control.

Thermal analysis.

3_SSR_32_03

4_ADCS_32_03 **Thermal design**

The thermal design of the ADCS board shall ensure that all components are maintained within their qualification temperature range throughout the lifetime of the subsystem.

Thermal Analysis

3_SSR_32_02

3.3 Maintainability

4_ADCS_33 _01 **Electrostatic sensibility**

The ADCS shall be handled with precaution against electrostatic discharges.

Manufacturer recommendation

3_SSR_33_01

4_ADCS_33_02 **Maintenance during storage and ground life**

The ADCS shall be designed to require no maintenance during storage and ground life. If ground maintenance during storage or ground operation cannot be avoided, the maintenance requirements shall be documented.

To survive storage and ground life

3_SSR_33_01

4. Interfaces

4.1 Structural

4_ADCS_41_01 **Attachment**

The ADCS shall be attached on the frame.

S&C

4_SC_10_02

4.2 Thermal

4_ADCS_42_01 **Thermal interfaces**

Thermal interfaces shall be optimized considering the whole satellite thermal design.

Thermal analysis

3_SSR_32_02

4.3 Electrical

4_ADCS_43_01 **Supply voltage**

The ADCS shall use [3.3V] +/- [7%].

EPS.

3_SSR_31_16

4_ADCS_43_02 **Current**

The ADCS shall use less than [150] mA.

EPS.

3_SSR_31_17

4_ADCS_43_03 **Connectors**

The ADCS shall use connectors compatible with the main data bus and the power bus.

SSR.

4_ADCS_10_01

4.4 Data interfaces

4_ADCS_44_01 **Data bus compatibility**

The ADCS shall be capable of communicating with the main data bus.

Same data bus for the whole satellite.

4_ADCS_10_01

4_ADCS_44_02 **TM**

The ADCS generate TM with all the sensors.

To provide housekeeping data

4_ADCS_10_01

4.5 Physical properties

4_ADCS_45_01 **Size**

The ADCS shall be adapted to the structure as shown on the picture below.

S&C.
Detail shall be discussed with mechanical engineer.

4_SC_10_02

4_ADCS_45_02 **Mass**

The ADCS mass shall weight less than [120] grams.

Mass budget.

3_SSR_45_01

5. Environmental

5.0 General

4_ADCS_50_01 **Environment**

The ADCS operates under the environment constraints described in the SwissCube Environment Requirements document S3-B-STRU-1-3-Launch Environment.

Launch environment document S3-B-STRU-1-3-Launch Environment
G. Röthlisberger Space Environment at earth distances between [400 and 1000 km]

3_SSR_50_01

5.1 Thermal

4_ADCS_51_01 **Qualification temperature range**

The ADCS subsystem shall survive in its qualification temperature range.

Thermal analysis

3_SSR_32_02

5.2 Static and dynamic loads

4_ADCS_52_01 **Acceleration**

The ADCS shall withstand a maximal acceleration of [10,4] g including margins.

Launch environment constraints

3_SSR_52_02

5.3 Vacuum

4_ADCS_53_01 **Vacuum**

The ADCS subsystem shall operate under vacuum conditions.

Space Environement

3_SSR_53_01

5.4 Radiation

4_ADCS_54_01 **Total dose**

The ADCS shall support a TID of maximum [20]kRad.

Analysis using ESA Spenvis Tool.
This is the value for 1 year in orbit.

3_SSR_54_02

6. Operational

6.1 Autonomy

4_ADCS_61_01 **Life time**

The ADCS shall be designed to operate during [4] months including commissioning and nominal activities. The lifetime can be extended to [1] year.

Mission duration

3_SSR_61_01

6.2 Control

4_ADCS_62_01 **Cmd reception**

The ADCS shall be able to receive cmd from the CDMS at all times when not in the "OFF" mode. [TBC]

To control the attitude of the space system

4_ADCS_10_01

6.3 Failure management

4_ADCS_63_01 **Failure propagation**

Failure of one part or element of the ADCS shall not result in consequential damage to the equipments or other satellite components

To minimize failure propagation

3_SSR_63_01

4_ADCS_63_02 **Recovery plans**

For the nominal phase, possible failure scenarios and recovery plans shall be elaborated.

To ensure mission duration objectives

3_SSR_63_03

Level 5
1. Functional

1.0 General

5_ADCS/ACT_10_0 **Actuators**

The actuators shall provide torque to control the attitude of the Space System.

To perform attitude control.
Actuators include: magnetotorquers.

4_ADCS_10_04

5_ADCS/ACT_10_0 **Magnetotorquer axis control**

The magnetotorquers shall provide controllability on all 3 axis

To provide attitude control

4_ADCS_10_04

5_ADCS/CTL_10_0 **Controller signals**

The controller shall generate control signals of the actuators.

Primary function

4_ADCS_10_04

5_ADCS/CTL_10_0 **Detumbling**

The controller shall provide signals to detumble the satellite after leaving the P-POD or after any other event needing restabilisationfor far to the nominal attitude.

Lowering rotational speed and angular error

4_ADCS_10_04

5_ADCS/CTL_10_0 **Nominal Control**

The controller shall provide signals to keep the satellite near the origin (nominal attitude) when already there.

Precision and perturbation rejection

4_ADCS_10_04

5_ADCS/DTS_10_0 **Determination sensors**

The ADCS determination sensors shall provide data for determination of the attitude of the satellite.

ADCS determination functions.
These sensors includes : magnetometers, gyroscopes and sun sensors.

4_ADCS_10_02

5_ADCS/DTS_10_0 **Sun Sensors axis measurement**
Sun sensors shall provide measurements on all 3 axis.
To determine the attitude

4_ADCS_10_02

5_ADCS/DTS_10_0 **Gyroscopes axis measurements**
Gyroscopes shall provide measurements on all 3 axis.
To determine the attitude

4_ADCS_10_02

5_ADCS/DTS_10_0 **Magnetometers axis measurements**
The magnetic sensors shall provide measurements on all 3 axis.
To provide attitude determination.

4_ADCS_10_02

5_ADCS/HKS_10_0 **HK sensors**
The ADCS HK sensors shall provide data to determine the health of the ADCS subsystem.
To fulfill ADCS function
These sensors include Temperature sensors.

4_ADCS_10_03

5_ADCS/MCU_10_01 **ADCS microncontroller function I**
The ADCS MCU shall receive commands from CDMS.
Definition of ADCS function

4_ADCS_10_01

5_ADCS/MCU_10_02 **ADCS Microcontroller function II**
The microcontroller shall turn off and on the sensors.
ADCS determination function

4_ADCS_10_02

5_ADCS/MCU_10_03 **ADCS microcontroller function III**
The microcontroller shall collect sensor data.
ADCS function

4_ADCS_10_02

5_ADCS/MCU_10_04 **ADCS microcontroller function IV**
The microcontroller shall format the data from sensor.
ADCS function

4_ADCS_10_02

5_ADCS/MCU_10_05 **ADCS microcontroller function V**

The microcontroller shall send TM to CDMS.

ADCS function

4_ADCS_10_01

5_ADCS/MCU_10_06 **ADCS microcontroller function VI**

The microcontroller shall command the actuators.

ADCS attitude control function

4_ADCS_10_04

5_ADCS/PCB_10_0 **PCB**

The ADCS PCB shall provide a mechanical and electrical support for all the electronics.

To perform ADCS function

4_ADCS_10_01

2. Mission and Performance

2.3 Unit state H/W performance

5_ADCS/ACT_23_0 **Magnetotorquers consumption in "NOMINAL" mode**

The mean magnetotorquer power consumption in "NOMINAL" mode shall be less than [50] mW per magnetotorquer.

Power budget

4_ADCS_23_04

5_ADCS/ACT_23_0 **Magnetotorquers consumption in "SENSOR" mode**

The magnetotorquers shall not consume any power in "SENSOR" mode.

Power budget

4_ADCS_23_03

5_ADCS/ACT_23_0 **Magnetotorquers consumption in "STAND-BY" mode**

The magnetotorquers shall not consume any power in "STAND-BY " mode.

Power budget

4_ADCS_23_02

5_ADCS/ACT_23_0 **Magnetotorquers consumption in "OFF" mode**

The magnetotorquers shall not consume any power in "OFF" mode.

Power budget

4_ADCS_23_01

5_ADCS/ACT_23_0 **Magnetotorquers magnetic moment**

The magnetotorquers shall generate a magnetic moment of at least [0.0285] Am2.

AHW report phase B
To control perturbations torques

4_ADCS_10_04

5_ADCS/DTS_23_0 **Determination sensor power consumption in**

The mean sensors power consumption in "NOMINAL" mode shall be less than [60] mW.

Power budget

4_ADCS_23_04

5_ADCS/DTS_23_0 **Determination sensors consumption in "STAND-BY"**

The determination sensors shall not consume any power in "STAND-BY " mode.

Power budget

4_ADCS_23_02

5_ADCS/DTS_23_0 **Determination sensors consumption in "OFF" mode.**

The determination sensors shall not consume any power in "OFF" mode.

Power budget

4_ADCS_23_01

5_ADCS/DTS_23_0 **Magnetometers measurement range**

Magnetometers shall be capable of measuring magnetic fields in the range of [-60] µT and [60] µT with an accuracy of +/- [1]%.

Sensor performances; maximal magnetic field over the poles according to "S3-C-ADCS-1-2-Earth's Magnetic Field Model" plus a margin of 5µT.

4_ADCS_10_02

5_ADCS/DTS_23_0 **Magnetometers measurement resolution**

The magnetic sensor shall have a minimum resolution of [1 10-6]T.

Sensor performances.

4_ADCS_10_02

5_ADCS/DTS_23_0 **Sun sensors measurement range**

Sun sensors shall be capable of measuring the direction of the sun in the range of [0] deg and [60] deg with an accuracy of +/- [10]%.

Sensor performances.

4_ADCS_10_02

5_ADCS/DTS_23_0 **Gyroscopes measurement range**

Gyroscopes shall be capable of measuring angular rate in the range of [0.01] deg/s and [180] deg/s with an accuracy of +/- [1]%.

Sensor performances.

4_ADCS_10_02

5_ADCS/MCU_23_01 **MCU consumption in "OFF" mode**

The microcontroller shall not consume any power in "OFF" mode.

Power budget

4_ADCS_23_01

5_ADCS/MCU_23_02 **MCU consumption in "STAND-BY" mode**

The mean microcontroller power consumption shall be less than [30] mW in "STAND-BY" mode.

Power budget

4_ADCS_23_02

5_ADCS/MCU_23_03 **MCU consumption in "SENSOR" mode**

The mean microcontroller power consumption shall be less than [30] mW in "SENSOR" mode.

Power budget

4_ADCS_23_03

5_ADCS/MCU_23_04 **MCU consumption in "NOMINAL" mode**

The mean microcontroller power consumption shall be less than [30] mW in "NOMINAL" mode.

Power budget

4_ADCS_23_04

2.5 Reliability and redundancy

5_ADCS/ACT_25_0 **Actuators reliability and redundancy**

The actuators shall comply to 4_ADCS_25_02.

To mitigate SEL.

4_ADCS_25_02

5_ADCS/ACT_25_0 **Actuators reliability and redundancy**

The actuators shall comply to 4_ADCS_25_03.

Level 4 ADCS requirements

4_ADCS_25_03

5_ADCS/ACT_25_0 **Actuators reliability and redundancy**
The actuators shall comply to 4_ADCS_25_04.
Level 4 ADCS requirements
4_ADCS_25_04

5_ADCS/DTS_25_0 **Determination sensors reliability and redundancy**
The determination sensors shall comply to 4_ADCS_25_02.
To mitigate SEL.
4_ADCS_25_02

5_ADCS/DTS_25_0 **Determination sensors reliability and redundancy**
The determination sensors shall comply to 4_ADCS_25_03.
Level 4 requirements
4_ADCS_25_03

5_ADCS/DTS_25_0 **Determination sensors reliability and redundancy**
The determination sensors shall comply to 4_ADCS_25_04
Level 4 requirements
4_ADCS_25_04

5_ADCS/MCU_25_01 **MCU Reliability and redundancy**
The microcontroller shall comply to 4_ADCS_25_02.
To mitigate SEL.
4_ADCS_25_02

5_ADCS/MCU_25_02 **MCU Reliability and redundancy**
The microcontroller shall comply to 4_ADCS_25_03.
Level 4 ADCS requirements
4_ADCS_25_03

5_ADCS/MCU_25_03 **MCU Reliability and redundancy**
The microcontroller shall comply to 4_ADCS_25_04.
Level 4 ADCS requirements
4_ADCS_25_04

5_ADCS/PCB_25_0 **PCB reliability and redundancy**
The microcontroller shall comply to 4_ADCS_25_02.
To mitigate SEL.
4_ADCS_25_02

5_ADCS/PCB_25_0 **PCB reliability and redundancy**
The microcontroller shall comply to 4_ADCS_25_03.
Level 4 ADCS requirements
4_ADCS_25_03

5_ADCS/PCB_25_0 **PCB reliability and redundancy**
The microcontroller shall comply to 4_ADCS_25_04.
Level 4 ADCS requirements
4_ADCS_25_04

3. Design

3.1 Constraints

5_ADCS/ACT_31_0 **Magnetotorquers design constraints**
The magnetotorquers shall comply to 4_ADCS_31_01.
Arina user's manual
4_ADCS_31_01

5_ADCS/ACT_31_0 **Magnetotorquers design constraints**
The magnetotorquers shall comply to 4_ADCS_31_02.
Launch date
4_ADCS_31_02

5_ADCS/ACT_31_0 **Magnetotorquers design constraints**
The magnetotorquers shall comply to 4_ADCS_31_03.
Level 4 ADCS requirements
4_ADCS_31_03

5_ADCS/DTS_31_0 **Determination sensors design constraints**
The Determination sensors shall comply to 4_ADCS_31_01.
Arina user's manual
4_ADCS_31_01

5_ADCS/DTS_31_0 **Determination sensors design**
The Determination sensors shall comply to 4_ADCS_31_02.
Launch date
4_ADCS_31_02

5_ADCS/DTS_31_0 **Determination sensors design**
The Determination sensors shall comply to 4_ADCS_31_03.
Level 4 ADCS requirements
4_ADCS_31_03

5_ADCS/MCU_31_01 **MCU design constraints**
The microcontroller shall comply to 4_ADCS_31_01.
Arina user's manual
4_ADCS_31_01

5_ADCS/MCU_31_02 **MCU design constraints**
The microcontroller shall comply to 4_ADCS_31_02.

Launch date

4_ADCS_31_02

5_ADCS/MCU_31_03 **MCU design constraints**
The microcontroller shall comply to 4_ADCS_31_03.
Level 4 ADCS requirements

4_ADCS_31_03

5_ADCS/PCB_31_0 **PCB design constraints**
The ADCS PCB shall comply to 4_ADCS_31_01.
Arina user's manual

4_ADCS_31_01

5_ADCS/PCB_31_0 **PCB design constraints**
The ADCS PCB shall comply to 4_ADCS_31_02.
Launch date

4_ADCS_31_02

5_ADCS/PCB_31_0 **PCB design constraints**
The ADCS PCB shall comply to 4_ADCS_31_03.
Level 4 ADCS requirements

4_ADCS_31_03

3.2 Thermal

5_ADCS/ACT_32_0 **Actuators thermal design**
The actuators shall comply to 4_ADCS_32_01.
Level 4 ADCS specifications

4_ADCS_32_01

5_ADCS/ACT_32_0 **Actuators thermal design**
The actuators shall comply to 4_ADCS_32_02.
Level 4 ADCS specifications

4_ADCS_32_02

5_ADCS/ACT_32_0 **Actuators thermal design**
The actuators shall comply to 4_ADCS_32_03.
Thermal analysis

4_ADCS_32_03

5_ADCS/DTS_32_0 **Determination sensors thermal design**
The determination sensors shall comply to 4_ADCS_32_01.
Level 4 ADCS requirements

4_ADCS_32_01

5_ADCS/DTS_32_0 **Determination sensors thermal**

The determination sensors shall comply to 4_ADCS_32_02.

Level 4 ADCS requirements

4_ADCS_32_02

5_ADCS/DTS_32_0 **Determination sensors thermal**

The determination sensors shall comply to 4_ADCS_32_03.

Thermal analysis

4_ADCS_32_03

5_ADCS/MCU_32_01 **MCU thermal design**

The microcontroller shall comply to 4_ADCS_32_01.

Level 4 ADCS requirements

4_ADCS_32_01

5_ADCS/MCU_32_02 **MCU thermal design**

The microcontroller shall comply to 4_ADCS_32_02.

Level 4 ADCS requirements

4_ADCS_32_02

5_ADCS/MCU_32_03 **MCU thermal design**

The microcontroller shall comply to 4_ADCS_32_03.

Thermal analysis

4_ADCS_32_03

5_ADCS/PCB_32_0 **PCB thermal design**

The PCB shall comply to 4_ADCS_32_01.

Level 4 ADCS specifications

4_ADCS_32_01

5_ADCS/PCB_32_0 **PCB thermal design**

The PCB shall comply to 4_ADCS_32_02.

Level 4 ADCS specifications

4_ADCS_32_02

5_ADCS/PCB_32_0 **PCB thermal design**

The PCB shall comply to 4_ADCS_32_03.

Thermal analysis

4_ADCS_32_03

3.3 Maintainability

5_ADCS/ACT_33_0 **Actuators maintainability**

The actuators shall comply to 4_ADCS_33_01.

Level 4 specifications

4_ADCS_33_01

5_ADCS/ACT_33_0 **Actuators maintainability**
The actuators shall comply to 4_ADCS_33_02.
To survive storage and ground life
4_ADCS_33_02

5_ADCS/DTS_33_0 **Determination sensors maintainability**
The determination sensors shall comply to 4_ADCS_33_01.
Level 4 specifications
4_ADCS_33_01

5_ADCS/DTS_33_0 **Determination sensors**
The determination sensors shall comply to 4_ADCS_33_02.
To survive storage and ground life
4_ADCS_33_02

5_ADCS/MCU_33_01 **MCU maintainability**
The microcontroller shall comply to 4_ADCS_33_01.
Level 4 ADCS specifications.
4_ADCS_33_01

5_ADCS/MCU_33_02 **MCU maintainability**
The microcontroller shall comply to 4_ADCS_33_02.
To survive storage and ground life
4_ADCS_33_02

5_ADCS/PCB_33_0 **PCB maintainability**
The PCB shall comply to 4_ADCS_33_01.
Level 4 specifications
4_ADCS_33_01

5_ADCS/PCB_33_0 **PCB maintainability**
The PCB shall comply to 4_ADCS_33_02.
To survive storage and ground life
4_ADCS_33_02

4. Interfaces

4.1 Structural

5_ADCS/ACT_41_0 **Magnetotorquers**
Magnetotorquers shall be located in perpendicular planes.
Action on three axes.
5_ADCS/ACT_10_02

5_ADCS/DTS_41_0 **Sun sensor location**
Sun sensors shall be mounted on the 6 faces, 1 sensor per face

Need 3 sensors to determine direction of the Sun at all times, need 6 sensor heads to cover all possible directions.

5_ADCS/DTS_10_02

5_ADCS/DTS_41_0 **Magnetometer location**

The magnetometers shall be placed at a distance at least greater than [xx]mm from the magnetotorquers plane.

In order to avoid as well as it is possible disturbances.

5_ADCS/PCB_41_0 **PCB placement**

The placement of the PCB shall be optimized to be as distant as possible from the magnetotorquers.

Magnetic perturbations on magnetometer

4.2 Thermal

5_ADCS/ACT_42_0 **Magnetotorquers**

Magnetotorquers shall be thermally connected to the frame.

TH

4_ADCS_32_02

5_ADCS/PCB_42_0 **PCB**

The PCB shall be thermally connected to the frame.

TH

4_ADCS_32_02

4.3 Electrical

5_ADCS/ACT_43_0 **Magnetotorquers supply voltage**

The magnetotorquers and their electronic shall use 3.3V [+/- 7%].

ADCS alimentation

4_ADCS_43_01

5_ADCS/ACT_43_0 **Magnetotorquers connection**

The magnetotorquers shall be connected to ADCS board.

Decrease number of interfaces.

4_SC_10_02

5_ADCS/DTS_43_0 **Gyroscopes supply voltage**

The gyroscopes and their electronic shall use 3.0V[+/-2%].

ADCS alimentation

4_ADCS_43_01

5_ADCS/DTS_43_0 **Magnetometers supply voltage**

The magnetometers and their electronic shall use 3.3V[+/-7%].

ADCS alimentation

4_ADCS_43_01

5_ADCS/DTS_43_0 **Sun sensors supply voltage**

The sun sensors and their electronic shall use 3.0V[+/-2%].

ADCS alimentation

4_ADCS_43_01

5_ADCS/DTS_43_0 **Sun sensors connection**

The sun sensors shall be connected to ADCS board.

Decrease number of interfaces.

4_SC_10_02

5_ADCS/MCU_43_01 **Microcontroller supply voltage**

The microcontroller and its electronic shall use 3.3V[+/-

ADCS alimentation

4_ADCS_43_01

5_ADCS/PCB_43_0 **Board supply**

The main board shall be connected to the power bus and use use 3.3V[+/-7%].

ADCS alimentation

4_ADCS_43_01

4.4 Data interfaces

5_ADCS/CTL_44_0 **Controller input data**

The controller shall be feed with the current state variables estimations (rotational speed vector and quaternion)

FSW

5_ADCS/DTS_44_0 **Magnetometer master clock frequency**

The magnetometer shall use a master clock frequency in the range of [5] to [7.35] MHz.

S3_Phase_B-C-ADCS-1-3-ADCS_HW_and_System report.

5_ADCS/MCU_44_01 **Microcontroller main bus interface**

The microcontroller shall be capable of communicating with the main data bus.

Same data bus for the whole satellite

4_ADCS_44_01

5_ADCS/MCU_44_02 **Microcontroller sensor interface**

The microcontroller shall be capable of communicating with each sensor.

Anolog signal reading, and serial data interface reading capability.

5_ADCS/MCU_10_02

5_ADCS/MCU_44_03 **TM**

The microcontroller shall generate TM with all the sensors.

To provide housekeeping data

4_ADCS_44_02

4.5 Physical properties

5_ADCS/ACT_45_0 **Magnetotorquer mass**

Each coil shall weight less than [28] grams.

Mass budget.

4_ADCS_45_02

5_ADCS/ACT_45_0 **Magnetotorquer size**

Magnetotorquers external dimensions shall not exceed [70x80x5] mm on X, Y and Z axis in System Reference Frame.

S&C.

4_ADCS_45_01

5_ADCS/DTS_45_0 **Sun sensors dimensions**

The external sun sensors PCB dimensions shall not exceed [20x16x3.5] mm.

S&C.

4_ADCS_45_01

5_ADCS/PCB_45_0 **Card dimensions**

The external ADCS card dimensions shall not exceed [80x85x10] mm, and the value of [10] shall not be reached on the entire surface.

S&C.

4_ADCS_45_01

5_ADCS/PCB_45_0 **Card mass**

The ADCS card (PCB and components) shall weight less than [34] g.

Mass budget

4_ADCS_45_02

5. Environmental

5.1 Thermal

5_ADCS/ACT_51_0 **Thermal environement**

The magnetotorquers shall survive the qualification temperature range [-45°C to +70°C].

Thermal analysis

4_ADCS_51_01

5_ADCS/DTS_51_0 **Thermal Environement**

The gyroscopes and magnetometers shall survive the qualification temperature range [-30°C to +60°C].

Thermal analysis

4_ADCS_51_01

5_ADCS/DTS_51_0 **Thermal Environment**

The sun sensors shall survive the qualification temperature range [-45°C to +70°C].

Thermal analysis

4_ADCS_51_01

5_ADCS/MCU_51_01 **Thermal environment**

The microcontroller shall survive the qualification temperature range [-30°C to +60°C].

Thermal analysis

4_ADCS_51_01

5_ADCS/PCB_51_0 **Thermal environment**

The PCB shall survive the qualification temperature range [-[-30°C to +60°C].

The current temperature range is designed as [-20] to [60]°C

4_ADCS_51_01

5.2 Static and dynamic loads

5_ADCS/ACT_52_0 **Acceleration**

The magnetotorquers shall comply to 4_ADCS_52_01.

Level 4

4_ADCS_52_01

5_ADCS/DTS_52_0 **Acceleration**

The determination sensors shall comply to 4_ADCS_52_01.

Level 4

4_ADCS_52_01

5_ADCS/MCU_52_01 **Acceleration**

The microcontroller shall comply to 4_ADCS_52_01.

Level 4

4_ADCS_52_01

5_ADCS/PCB_52_0 **Acceleration**

The determination sensors shall comply to 4_ADCS_52_01.

Level 4

4_ADCS_52_01

5.3 Vacuum

5_ADCS/ACT_53_0 **Vacuum**

The magnetotorquers shall be able to operate under vacuum conditions.

Space Environement

4_ADCS_53_01

5_ADCS/DTS_53_0 **Vacuum**

The determination sensors shall be able to operate under vacuum conditions.

Space Environement

4_ADCS_53_01

5_ADCS/MCU_53_01 **Vacuum**

The microcontroller shall be able to operate under vacuum conditions.

Space environement

4_ADCS_53_01

5_ADCS/PCB_53_0 **Vacuum**

The PCB shall be able to operate under vacuum conditions.

Space Environement

4_ADCS_53_01

5.4 Radiation

5_ADCS/ACT_54_0 **Total dose**

The magnetotorquers shall support a TID of maximum [20]kRad.

Analysis using ESA Spenvis Tool.
This is the value for 1 year in orbit.

4_ADCS_54_01

5_ADCS/DTS_54_0 **Total dose**

The determination sensors shall support a TID of maximum [20]kRad.

Analysis using ESA Spenvis Tool.
This is the value for 1 year in orbit.

4_ADCS_54_01

5_ADCS/MCU_54_01 **Total dose**

The ADCS MCU shall support a TID of maximum

Analysis using ESA Spenvis Tool.
This is the value for 1 year in orbit.

4_ADCS_54_01

5_ADCS/PCB_54_0 **Total dose**

The PCB shall support a TID of maximum [20]kRad.

Analysis using ESA Spenvis Tool.
This is the value for 1 year in orbit.

4_ADCS_54_01

6. Operational

6.1 Autonomy

5_ADCS/ACT_61_0 **Life time**

The ADCS magnetotorquers shall be designed to operate during [4] months including commissioning and nominal activities. The lifetime can be extended to [1] year.

Mission duration

4_ADCS_61_01

5_ADCS/DTS_61_0 **Life time**

The ADCS determinations sensors shall be designed to operate during [4] months including commissioning and nominal activities. The lifetime can be extended to [1] year.

Mission duration

4_ADCS_61_01

5_ADCS/MCU_61_01 **Life time**

The MCU shall be designed to operate during [4] months including commissioning and nominal activities. The lifetime can be extended to [1] year.

Mission duration

4_ADCS_61_01

5_ADCS/PCB_61_0 **Life time**

The ADCS PCB shall be designed to operate during [4] months including commissioning and nominal activities. The lifetime can be extended to [1] year.

Mission duration

4_ADCS_61_01

6.2 Control

5_ADCS/ACT_62_0 **Magnetic incompatibility**

The magnetotorquers shall be turned off during measurements with magnetometers.

To avoid disturbances

5_ADCS/DTS_23_06

5_ADCS/CTL_62_0 **Controller switch-on time**

The attitude controller shall be switched on just after antenna deployment.

B. Graf
It must be after meaningful state variables are available from the

4_ADCS_10_04

5_ADCS/MCU_62_01 **Microcontroller**

The microcontroller must be able to receive a data request from CDMS at all times when not in the "OFF" mode.

To perform attitude determination and control

4_ADCS_62_01

3 DESIGN ASSUMPTIONS AND APPROACH

3.1 Approach

First of all a complete review of the phase B documentation has been done; a greater attention has been paid to the reports about the ADCS hardware and the magnetometer report (see [R1] and [R2]).

Then the MT have been redesigned. In order to do that, a review of the disturbances calculation has been performed to ensure a design with right values. Before building some MT, they have been recalculated with the new parameters (dimensions).

Because there were still some mistakes on the previous ADCS main board, a revision has been done in order to test its functionalities, particularly the sensors and the actuators.

Then the microcontroller programming was started; tests and measurements have been done with the MM, MT, SS and gyroscopes.

In parallel some other tasks have been performed, such as the development of an Earth's magnetic field model (see [R3]), the development of some sensors and actuators model and a study of the possibility to use permanents magnets to control the satellite.

At the end, a new complete review of the ADCS board has been started on in order to finalize its design for the SwissCube Integration Model.

It is important to mention that the hardware was not selected (in phase B) only on the performances, but on availability, compatibility with the system (voltage) and physical characteristics such as size and mass criteria.

3.2 Disturbances

The disturbances analysis has been refined at the beginning of phase B (see [R1]) and then checked.

The disturbances torques were separately calculated in the very worst case for every altitude between 400km and 1000km. Very worst case means that each parameter was taken at its maximal value. For this reason no margin was added at this point. The major disturbance factor is aerodynamic up to 600km. Figure 3-1 summarized the results (see Appendix E.1 for Matlab scripts).

Regarding the dimensioning of the actuators, twice the worst case was taken. The worst case happens at the altitude of 400km. The torque that the actuators shall produce is $2 \cdot 3.6e\text{-}7 = 7.2e\text{-}7$Nm (see section 4.3.1 for more details). For example, the torque used for the actuators dimensioning is twelve times greater than the disturbance torque at the altitude of 700km [R1].

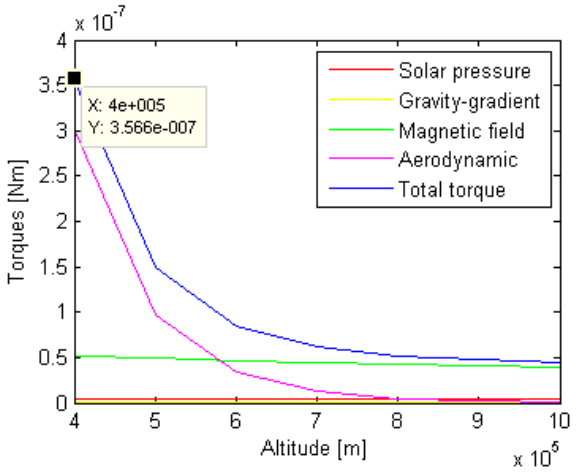


Figure 3-1 : Disturbances in function of altitude

3.3 Hardware assumptions

- The ADCS microcontroller will not support determination and control algorithms. It will be used to collect the sensors values and to store them onboard until the main controller (CDMS) will use them. It will also be used to control the actuators once the main computer has calculated the command values.
- All data are sent to the ADCS using the main I^2C bus. These data mainly come from the CDMS and the EPS.
- The ADCS board has only one supply voltage of 3.3V thanks to the EPS design. Thus its components must comply with that.
- The ADCS is not a critical system, therefore no redundancy is needed. But to ensure reliable operation, current limitations must be implemented for each electrical wire going to a sensor or an actuator. This prevents the shutdown of the whole ADCS board if short circuits occur (thanks to EPS overload protection).
- The MT will be glued inside the faces of the satellite. They will be then subject to high temperature variation. According to values found in [R11] plus a margin of 10°C, the MT should sustain a temperature between -45°C and +70°C.
- With a margin of 20°C, the temperature range for the ADCS board will be -30°C to +60°C (it is inside the satellite).

Figure 3-2 shows the exploded view of the actual mechanical parts of the satellite. We can see that the MT are located on the –X, +Y and –Z sides.

Figure 3-3 shows the last version of the ADCS electrical block diagram. It explains the architecture of the ADCS.

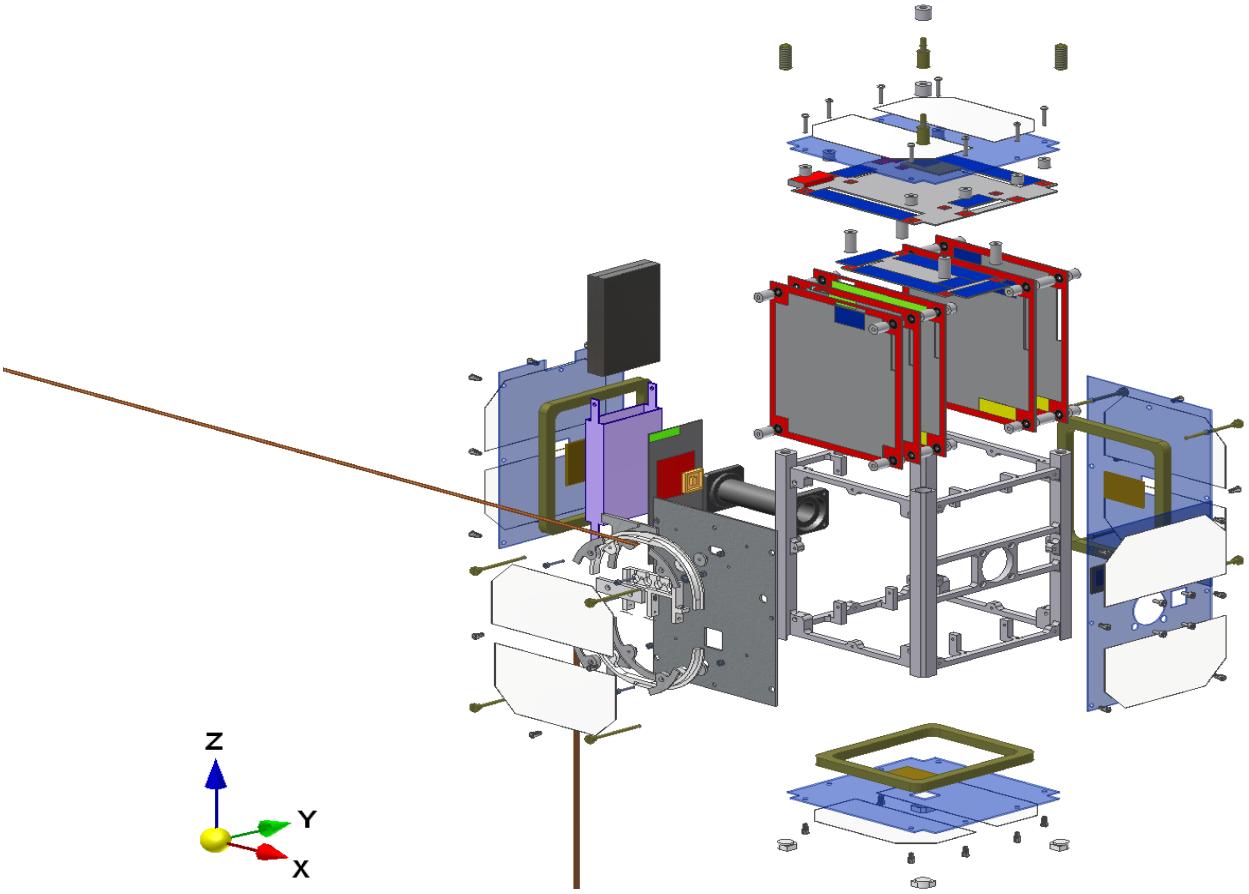


Figure 3-2 : Swisscube exploded view and frame of reference

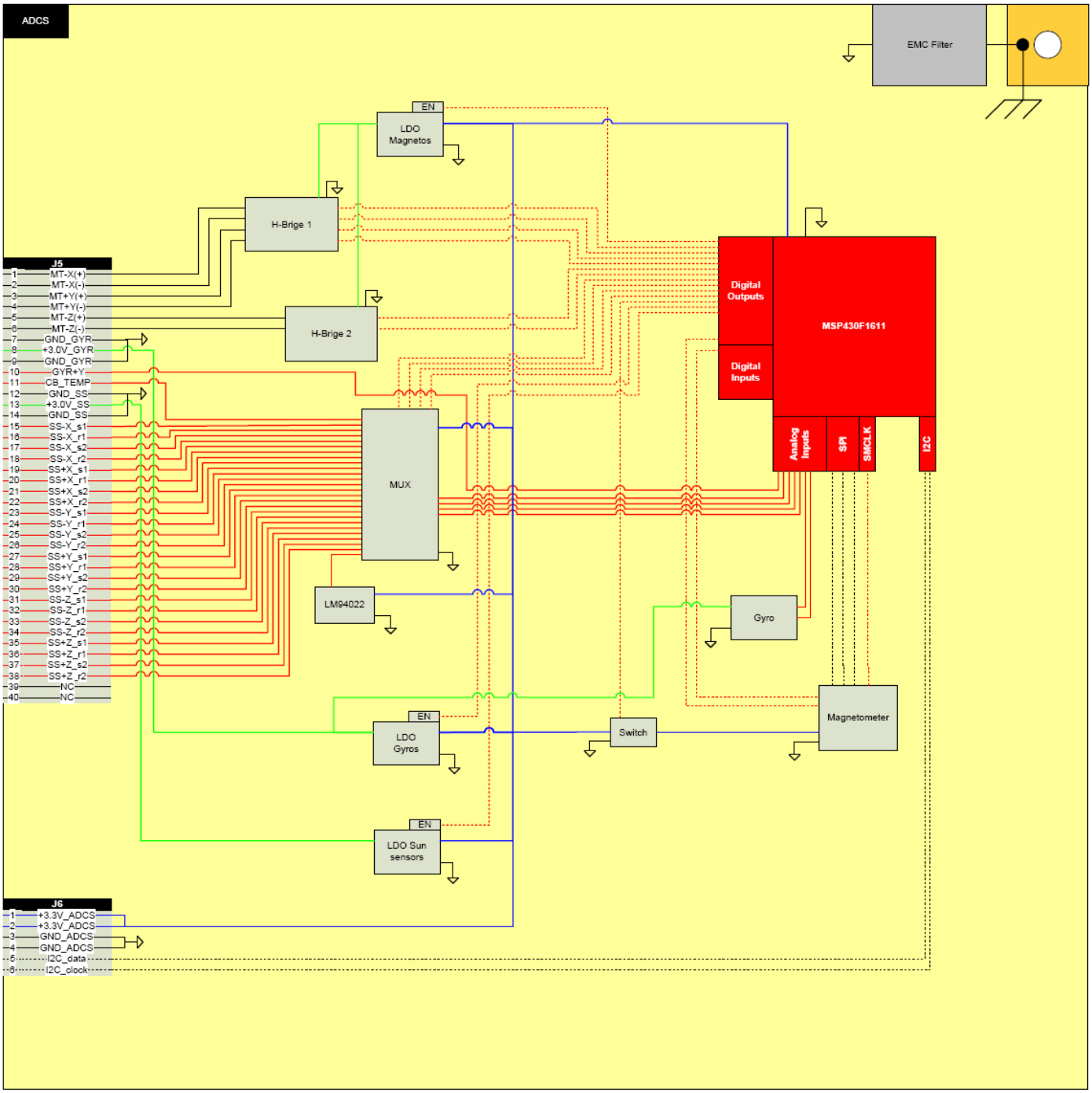


Figure 3-3 : ADCS present electrical block diagram (this version has not been built yet)

4 TECHNICAL DESCRIPTION

4.1 ADCS main board

4.1.1 Board design

The first version of the ADCS board was designed by Bastien Despont in the previous project (see [R1] for more details). All components were chosen and a PCB was built. The preliminary tests indicated there were some errors on the PCB, but thanks to the help of an adapter the operation of the microcontroller could be tested. No other functional tests were done on this board during the previous project.

At the beginning of this project, a first review of this board has been done in order to correct some errors, mainly in the programming interface, the PWM/MT outputs and the MM connections. The electrical circuit and the component placement on the PCB can be found in Appendix A.1. This board is still a test board; so many components such as jumpers will be suppressed in the next revision. The board is shown in Figure 4-1:

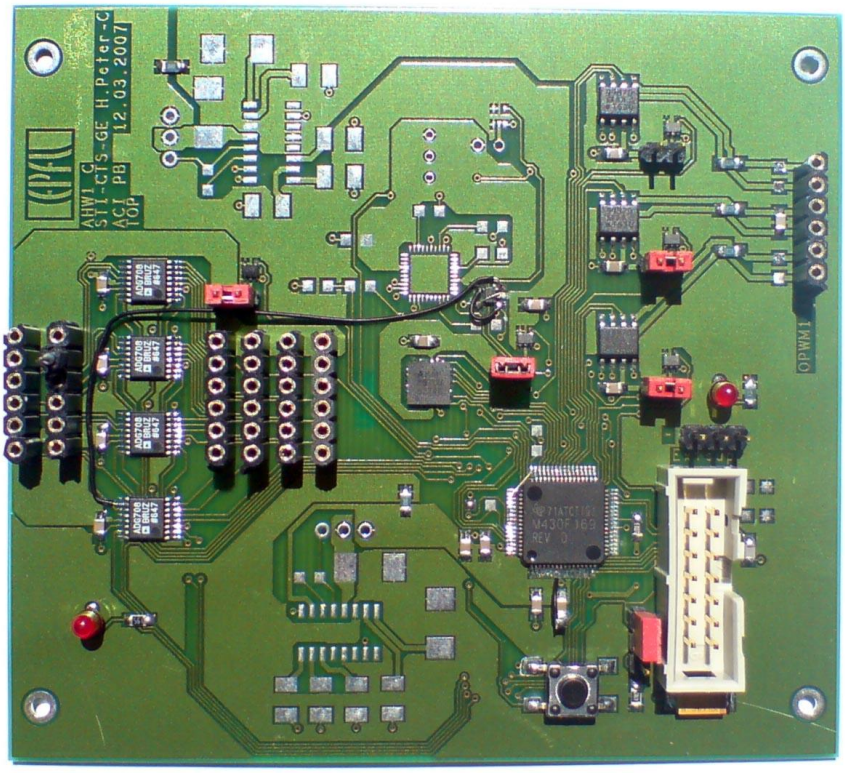


Figure 4-1 : Current ADCS board with some error corrections (see section 4.1.4)

A power supply connector (EXT_SUPPLY) has been added in order to be able to supply the board with an external stabilized power supply. It is also possible to use the power provided through the programmer/debugger by changing a jumper position on the board (POWER_TYPE). Be **careful with this jumper** (it must be in the right position for each power supply possibility) **and the polarity on the power supply connector because there is no protection.** When using the parallel TI programmer (MSP-FET430PIF), the supplied voltage is always 2.85V. This voltage can be set in the project properties (Project→Properties→Debug Properties→Target Voltage) when

using the USB TI programmer (MSP-FET430UIF) and Code Composer Essentials v2. The available power is small in these two cases.

Two LED were also added in order to give very helpful visual information; the first (LED0) indicates there is a power supply connected and the second (LED1) is connected to a microcontroller digital output, which it is very useful for the programming and debugging. **I strongly recommend keeping these LED in the next design** (they simply will not be mounted on the final satellite board).

Two 0Ω resistors were added in order to be able to bypass neatly the latch-up protections to assure correct operation of the board for the tests. The connector names were also printed in order to facilitate the use of the board.

4.1.2 MSP430F16x microcontrollers properties

The microcontroller used on this board is currently an MSP430F169 from Texas Instruments, but an MSP430F1611 can replace it without any trouble or modifications (this one will be used for all the satellite subsystems).

The internal oscillator of the MSP (DCO; for Digitally Controlled Oscillator) is used to provide the clock, because high frequency oscillators and crystals have really high power consumption. Moreover they are breakable components and it is difficult to find some ones satisfying the temperature requirements.

The DCO frequency can be set by software, but the nominal frequency has a big uncertainty (≈15%) and very strong temperature dependency (-0.4%/°C). The maximum frequency is also limited to about 5MHz. In order to improve these characteristics, an external resistor (R_{osc}) can be used to supply the DCO. A precision resistor with a 100kΩ value is recommended (a 0.1% 25ppm/°C resistor is used in our case), but because there is almost no information available on influence of the resistor on the frequency setting, some tests must be done (see section 5.1.1). With this resistor the frequency dependence in temperature diminishes to about -0.1%/°C (see Figure 4-2) and the maximum frequency is increased.

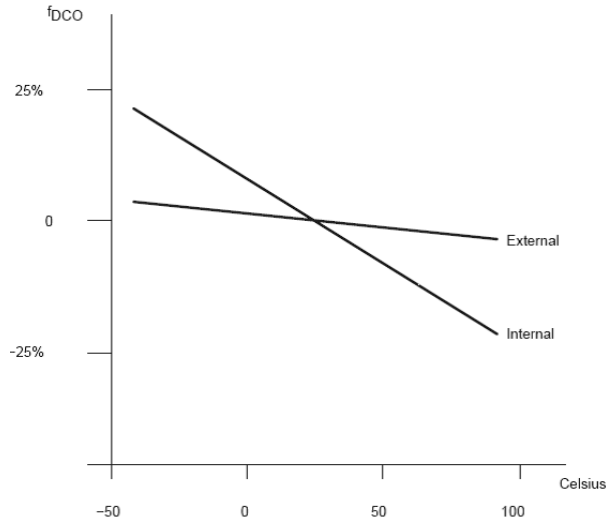


Figure 4-2 : DCO frequency variation versus temperature and external or internal resistor [R16]

But keep in mind that the nominal error is still big (15%) and the maximum frequency the microcontroller can withstand is limited by the voltage; although the DCO can go faster (≈10MHz), for a 3.3V supply we have $\mathbf{f_{MCLK\_max}=7.35MHz}$ and for a 3.0V it is reduced to 6.7MHz

(see datasheet). **Be careful not to exceed this value because of the temperature drift!** An error has been done in the previous project concerning this frequency. Increasing the CPU frequency also increase the power consumption.

In the case of the ADCS board, a frequency as stable and as high as possible is needed for the MCLK of the MM (see section 4.2.1).

Although all subsystem must know the time, a low frequency watch crystal is not needed, because the time will be synchronized regularly for the whole satellite using I^2C messages. For small time periods the committed error (because of the DCO use) will be small. It is possible to correct the time clock temperature drift (and not the DCO frequency) by changing the period of the clock timer (see section 4.1.3) when reading the temperature with a sensor.

The microcontroller has an internal 12-bit analog to digital converter (ADC). It is mainly used to measure the sun sensors, gyroscopes and temperature sensors signals. This represents about 30 different signals. Because the microcontroller has only 8 analog inputs, 4 external analog multiplexers AD708 are used. They have an 8Ω conduction resistance and a 30ns switching time. This time is smaller than the microcontroller clock period (if $f_{MCLK} \leq 33MHz$), so no software waiting time is needed when switching from an input to another. But if low-pass filter are present between the multiplexers and the microcontroller (this is not the case for the moment), it is necessary to wait until the capacitor is fully charge before starting the AD conversion.

For the moment, the internal MSP 2.5V voltage reference is used for the ADC. An external capacitor is needed on pin 7 to ensure its correct operations. It has been forgotten in the ADCS board. **This reference has a quite bad accuracy (4%, ±100ppm/°C), therefore it could be useful to have another more precise external voltage reference in the next design,** because this error is reported to the sensors measurements.

Figure 4-3 shows the ADC equivalent circuit. Some time is needed to charge correctly the input capacitor before starting the conversion.

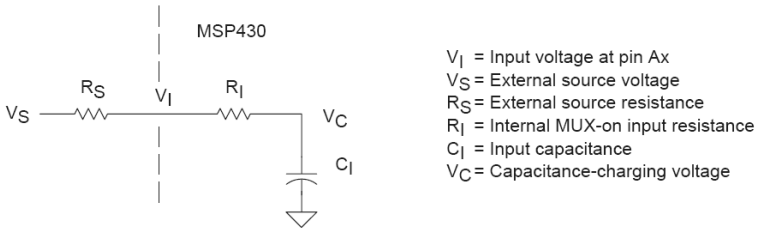


Figure 4-3 : Analog input equivalent circuit of the MSP430 ADC [R16]

This sampling time can be set in software. It is related to the external source resistance and its minimum value can be computed using this formula:

$$t_{sample} > (R_s + R_I) \cdot \ln(2^{13}) \cdot C_I + 800ns \quad (3.1)$$

With R_I=2kΩ and C_I=40pF. For R_s=0, t_{sample}>1.5µs. To measure the internal temperature sensor voltage, the sampling time must be >30µs.

4.1.3 Software

The first version of the ADCS software has been programmed during this project. Recommendations for microcontroller programming in C code can be found in Appendix E.3.1.

This software is able to do measurements with all sensors (MM, SS, GYR and temperatures sensors) and can generate 6 PWM outputs for the MT. **None temperature drift is corrected for the moment** and all sensors and actuators are switched on at the beginning of the program. They should be **switched off when it is not necessary** to reduce the power consumption. **Precautions must be taken with the startup time of each component!** For example the LM94022 startup time is about 10ms.

The microcontroller stays almost all the time in low-power mode in order to reduce power consumption. **Low power mode LPM0 must be used**, because all over modes switches off the DCO, so the microcontroller will never wake up (LPM1=LPM0 if DCO is used). Timer A is used as the ADCS time clock and to wake up the microcontroller with a regular time period. Interrupts also wake up the microcontroller, but it will automatically return in sleep mode after the interrupt service routine has been executed if there is no LPM0_EXIT command.

The architecture of this program is shown in Figure 4-4.

The I^2C and communication protocol have not been implemented for the moment.

The source code can be found in Appendix E.3.

Measurements can be stored in the microcontroller non volatile Flash memory using functions provided in Appendix E.4. Although the complete procedure is quite complicated, this is very useful to do measurements and recover them later without having the programmer connected (for example if the PCB is in a vacuum chamber or on a rotation stage).

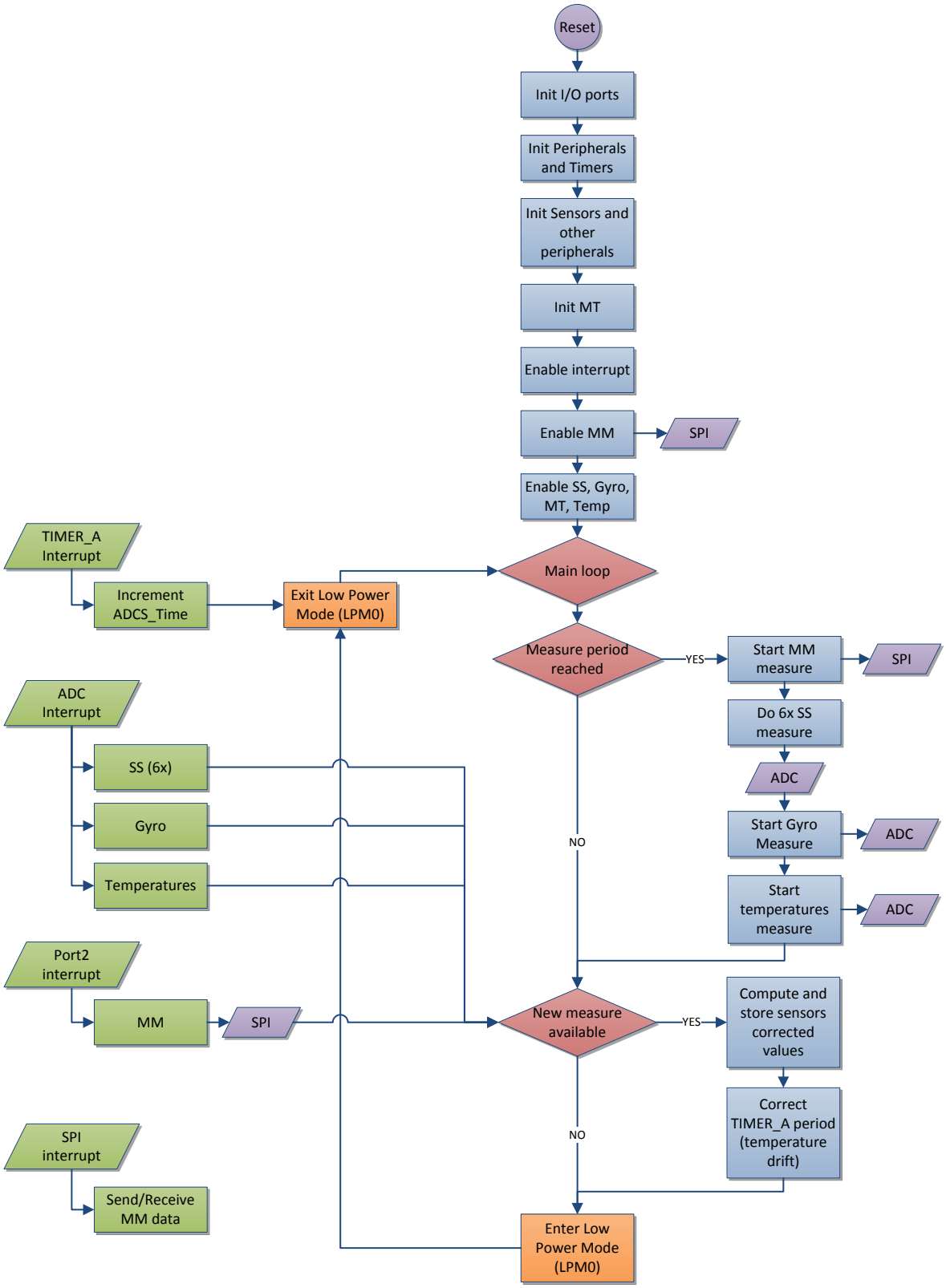


Figure 4-4 : ADCS actual test software architecture

4.1.4 Second ADCS board revision

A second revision of the ADCS board has been started in order to finalize the design for the SwissCube Integration Model. The actual ADCS electrical block diagram is shown in Figure 3-3.

The main changes and errors corrections are:

- Introduction of LDO regulators to supply sensors and actuators in order to stabilize the voltage and to limit the current in case a short circuit occur.
- The temperature sensors are now LM94022.
- The multiplexers are used to read temperature sensors.
- A 10µF capacitor should be placed between MSP pin7 and GND to ensure ADC correct operation.
- The MCLK signal for the magnetometer must use the SMCLK signal from the MSP (pin 49), because the MCLK from pin 48 is stopped when the microcontroller is in sleep mode.
- All jumpers must be removed.

Other changes must still be investigated:

- An external voltage reference should be used for the ADC.
- A circuit to measure the MT currents could be added (see section 4.3.1.4).
- Low-pass filters could be added before or after the multiplexers in order to reduce the noise on the SS (the wires are quite long and they are not shielded); we have $R_{on\_AD708} < 12\Omega$ and $R_{out\_AD8552} \approx 30\Omega$@3mA, thus a simple 1µF capacitor after the multiplexer connected to ground would create a low-pass filter with a cut-off frequency around 5kHz, but it will be necessary to wait until this capacitor is fully charge when switching the multiplexers channels.

The PCB layout will have to be completely redesigned for a 6-layer IS-420 substrate. The components and connexions must be placed according to the new available space on the board (see Appendix A.3).

4.2 Sensors

4.2.1 Magnetometer

4.2.1.1 Characteristics and parameters

The selected magnetometer (MM) is a 3-axis high sensitivity Hall device: the AK8970N from Asahi Kasei. It is a digital sensor and it uses an SPI interface to communicate with the microcontroller. Thanks to that many parameters are configurable:

- The internal amplifier gain, therefore the sensitivity.
- The offset using the internal digital to analog converter.
- The integrator time, therefore the sensitivity, noise and measurement time.

All this parameters are stored in internal registers (not in the EEPROM), so they have to be resend each time the MM is shut down.

This sensor has an internal temperature sensor, an EEPROM to store factory settings (only) and needs a high frequency clock signal for the measurements (MCLK). It can also generate an interrupt using a digital output (INT) when the measure is complete. It uses **8-bit** values, so the resolution, measurement range or offset are limited (256 different values). Figure 4-5 shows its architecture:

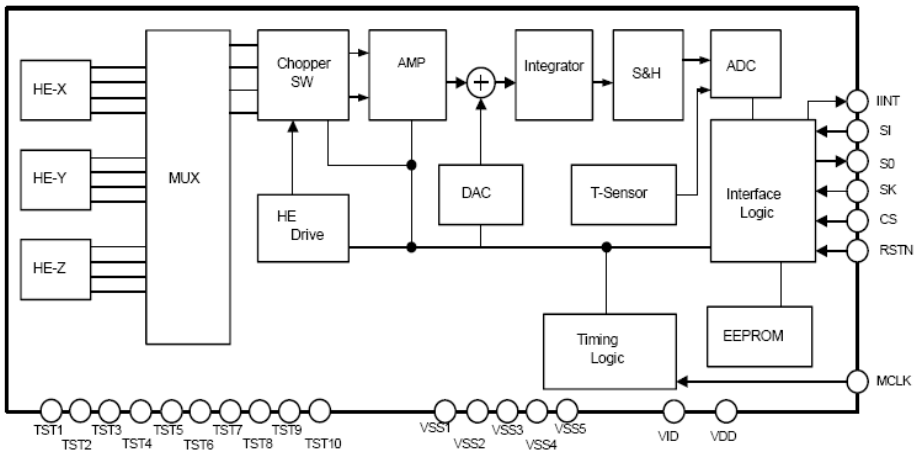


Figure 4-5 : AK8970N block diagram

This sensor has been characterized during the previous semester project using a LabView interface (see [R2]). Here is a summary of this report and the remaining tests:

- This sensor works with a MCLK frequency greater than 3MHz, although the datasheet indicates a minimum frequency of 11MHz.
- This sensor has a good linearity for all axis.
- There is a **strong offset drift with temperature which is different for each axis**. The **sensitivity also changes with the temperature**. The measured noise was quite big for a specific temperature and specific axis (40°C, X and Z axis only). This is quite strange and some measurements should be done with another sensor.
- Some vacuum tests have been done and conclude there is a quite big offset drift. Further tests should be done because the test conditions were not properly defined.
- An integrator time setting of "20ms" reduce the noise level for an identical sensitivity as with the "10ms" setting.
- All tests have been done using a LabView interface and a sine wave MCLK. So the MM must be interfaced with the ADCS microcontroller and the whole system operation must be characterized (the microcontroller generates a rectangular wave MCLK).
- There was an **error in the requirements**: the magnetic field rang is ±60µT and not ±30µT (see [R3], margin 5µT)! Thus some conclusions were wrong, especially those concerning the gain and the measurement resolution; **the maximum gain/sensitivity cannot be used.**
- A MCLK frequency of 5MHz has been advised because it seems to provide a higher sensitivity. Because of the error in the requirements, this is no more relevant. A higher frequency is desirable because it is less fathered to the nominal MCLK value (11MHz) and it can reduce the measurement time. As we have seen in the previous section, the microcontroller cannot generate a frequency higher than 7.35MHz.
- The **internal temperature sensor** and the **interrupt capability** (INT pin) have not been tested.
- The **measurement time** has not been measured.
- The **offset configuration** has not been characterized. This is important because a permanent magnet could be used to control the satellite, therefore it could saturate the MM. This setting must also be used to correct the temperature drift.

4.2.1.2 Electrical circuit

The MM is not supplied through an LDO and uses the 3.3V ADCS power supply. Although an LDO reduce the noise and stabilize the voltage, the SPI interface may not work properly, because the MM would generate 3.0V signals and the microcontroller 3.3V signals. Moreover the maximum voltage which can be applied on the MM pins is Vdd+0.3V=3.3V, so there is no margin.

The MM can currently be completely switched off by a NC7SZ66 switch (a simple MOS transistor could also have been used). This functionality is not really useful, because the MM stays in low-power mode and has a power consumption of only 5µA when it performs no operations (it is 13mA when measuring).

4.2.1.3 Software

Software to configure and do measurements with the MM has been programmed during this project. The SPI interface of the microcontroller has been programmed in order to send commands and settings to the MM and then receive the measurements results. The clock signal is also provided by the microcontroller when it is needed (microcontroller SMCLK output, MM MCLK input). An interrupt is generated when the measurements are completed (the main program runs in parallel with the measurements).

The MCLK frequency used should be configured into the MM with a specific command (register MD2, address 0xF4). Because the frequency used is theoretically not supported by the MM, the right MM setting does not exist. Therefore the setting for the lowest frequency has been used; D0-D3=0000 which correspond to an 11.2896MHz MCLK (we have ≈6.5MHz).

The overall software works well, but it must be completed about some particular points:

- **The MT must be switched off when the MM is measuring** in order to simplify the offset correction and reduce the MM errors.
- The offset and sensibility drifts corrections, because of the temperature variation, have not been implemented for the moment.
- The actual software uses some "while loops" to wait until the SPI has sent a command to the MM; a **maximum waiting time** should be implemented, because if the MM does not respond (for example because of a malfunction) the overall ADCS program execution is **stuck** in this loop. So this should be completed in order to have a better reliability.

The source code can be found in Appendix E.3.3.

4.2.1.4 Model for determination algorithms

A MM model has been done to be used in the control and determination algorithms simulations. It simply reproduces the way the MM do the measurements: all 3-axis are measured one after the other. This effect cannot be completely neglected if the dynamic is very high (in principle it is not our case), because as we will see in section 5.2.1, the measurement time is quite long.

4.2.2 Gyroscopes

The MEMs gyroscopes IDG-1000 or IDG-300 from InvenSense will be used. They work with a 2.7V to 3.3V power supply and have a power consumption of 8.5mA. Each chip contains a 2-axis gyroscope, so **2 chips** are necessary to obtain measurement in the 3 axis of rotation.

The 1st chip is located on the **ADCS board** and measures the **+X and +Z** angular rate. The 2nd is on the **Connection Board** which is perpendicular to the ADCS board and measures the **+Y** angular

rate. Since the measures are temperature dependant, there is a temperature sensor LM94022 near each gyroscope.

A single 3V low-noise LDO regulator TPS79330 is used as power supply for the two GYR. It acts like a switch, stabilizes the voltage, reduce the noise and create a current limitation if a short circuit occur on the wires used to supply the 2nd gyroscope (I_{sc}≤600mA). According to the datasheets, the **startup time** is about **400ms** for the IDG-1000, 200ms for the IDG-300 (and **100µs** for the LDO). This is very long and precautions must be taken when switching on and off these sensors to reduce power consumption (be also careful if the measurement rate is faster than 1Hz). **These timings should be verified with measurements** (see [R4]). The **startup power consumption** should also be checked in order to see if it is advantageous or not to switch on/off the GYR.

The analog outputs are read directly with the 12-bit ADC of the ADCS microcontroller (there is no multiplexer). This ADC can only measure a voltage between 0 and 2.5V, thus a part of the measure range can be lost in some particular cases (the gyroscope output is between 0 and 3V), but this also increase the sensitivity. A low-pass filter is added in order to lower the noise; its **cut-off frequency must be adapted** to the dynamic of the satellite to obtain an efficient filtering. For more details about the GYR, see [R4].

4.2.2.1 Software

The gyroscope software has been programmed; all 3 analog GYR signals are measured one after the others as quickly as possible using the ADC in sequence mode. When these measures are started, the main program continues to run in parallel and then the results are recovered.

The temperature drift correction has not been implemented for the moment.

Be careful to configure correctly the ADC **sampling time** according to the impedance value of gyroscope low-pass filter (see equation (3.1)).

The source code can be found in Appendix E.3.5.

4.2.3 Sun Sensors

4.2.3.1 Description

The sun sensors (SS) are provided by DTU. They can measure the sun direction on 2 axis (2 angles). One sun sensor stands on each face of the satellite (6 at all). Their output is a current which is converted into an analog voltage with an operational amplifier (AD8552). Each sun sensor has 4 outputs: 2 references related to the ambient luminosity and other conditions and 2 signals related to the sun direction.

The 24 signals are read trough 4 analog multiplexers AD708 with the 12-bit ADC of the ADCS microcontroller. The ADC can only measure a voltage between 0 and 2.5V, so the current-voltage converter gain must be adapted. To ensure a better thermal behavior, the converter **feedback resistor** must be a **high precision resistor** (0.1%, 25ppm/°C).

Because the SS are passives sensors, only the operational amplifiers need power; it represents a peak current about 26mA for the 24 amplifiers. This is supplied through a 3V low-noise LDO regulator TPS79330 which stabilize the voltage and limits the short-circuit current for the power wires (I_{sc}≤600mA). The current limit for the signal wires is smaller due to the AD8552 short-circuit current (I_{sc}≤30mA@3V).

The **startup time** is about **10µs** for the AD8552 and **100µs** for the LDO. **These timings should be verified with measurements and must be taken into account in software in order to reduce the average power consumption.**

For more details about the SS, see [R5].

4.2.3.2 Software

As the ADC is used, the SS software is very similar to the one for the GYR and the temperatures sensors.

The main difference is that all 4 signals are measured in a sequence, then the multiplexers are switched and the next 4 signals are measured. This operation is done for each SS. Thus the measurement is longer and the program waits until all SS measures are completed before returning in the main loop.

Be careful to configure correctly the ADC sampling time according to the AD8552 output impedance value: according to equation (3.1) and with R_s=40Ω, the ADC sampling time is $\mathbf{t_{sample}>1.5\mu s}$.

The source code can be found in Appendix E.3.6.

4.3 Actuators

Since the 1-axis inertial wheel has been suppressed, because of control problems, the magnetotorquers are the main actuators and the only active actuators of the SwissCube.

Because we have some trouble controlling the satellite, the possibility of using passive actuators, such as permanents magnets, has also been studied. This will be presented in this section (4.3.2). Others passive control methods exist, but they cannot be applied for the SwissCube:

- Gravity gradient: this technique uses the gravity force. A torque is produced if the satellite has an asymmetric mass repartition (the moment of inertia along one axis is larger than along the others). Because of the high symmetry of a CubeSat, this torque is very small and thus it can't be used to constrain orientation of the satellite.
- Another technique is to let spin the satellite along one axis. This is often used in conjunction with the gravity gradient. This method cannot be used too because it requires a high spinning rate and ensuring a good quality of the images taken by the SwissCube's camera becomes difficult.

4.3.1 Magnetotorquers

4.3.1.1 Design

The MT are coils which simply interact with the Earth's magnetic field (EMF). There are 3 identical MT on the satellite (to simplify the design and control). It's also possible to suppress one to gain mass if a magnet is used.

The MT are characterized with their dipole magnetic moment:

$$\vec{\mu} = NI\vec{A} \ [Am^2] \tag{3.2}$$

I is the current [A], N the number of turns and $\vec{A}$ the surface vector of the coil.

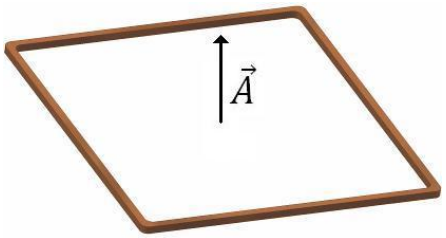


Figure 4-6 : Magnetotorquer schematic [R1]

The produced torque is expressed by:

$$\vec{M} = \vec{\mu} \times \overrightarrow{B_{earth}} \ [Nm] \tag{3.3}$$

With $\boldsymbol{B_{earth} \approx 25\mu T}$ at 400km altitude. This is a mean value; **keep in mind the minimum local value is really 19.5µT** (see [R3]), so the torque could be smaller in particular locations.

The MT are designed to be able to produce the double of the maximum perturbation torque (=2·3.6·10^{-7}Nm). The worst case for the design is for an altitude of 400km and at the equator:

- The perturbations are the biggest for 400km altitude (see Figure 3-1).
- The EMF is smaller at the equator.
- Although the EMF decreases if the altitude increases, its variation is smaller than the one of the perturbations, so the worst case is really for 400km altitude.

The current in the coil is determined using the available power and the voltage. To reduce the total mass of the magnetotorquer, the enclosed area should be maximized and the number of turns minimized. The maximum current density was set to 2A/mm^2, even if it is possible to go up to 8A/mm^2. This is a security factor to avoid a wire burning due to the absence of convection. The wire used is a CAB-200 non space-qualified copper wire with a diameter of 150µm. An aluminium wire is not useful as explained in [R1].

The MT are glued on the outer panels of the satellite. **To ensure the mechanical solidity, precise outer dimension, acceptable outgassing properties and to protect the coil, the whole magnetotorquer is moulded with a space-qualified epoxy resin.**

The coils are designed iteratively using a Matlab script (heart.m, results in Coil_results.m, see Appendix E.2). The area A used to compute the coil is the inner area, thus the coil will produce a slightly greater torque than twice the perturbation torque.

Maximal disturbance torque (design margin of 2)	0.72·10^{-6}Nm
Magnetic field at 400km (mean value)	25µT
Voltage Vcc	3.3V or 3.0V
Available Power P_0	50mW
Nominal Current I_0	15mA
Maximum current density J_{max}	2·10^6A/m^2
Copper resistivity ρ_0 (at 20°C)	1.72·10^{-8}Ωm
Temperature change coefficient α	3.9·$10^{-3}K^{-1}$
Magnetic dipole moment µ	2.85·$10^{-2}Am^2$
Wire	CAB-200

Wire diameter	150µm
Filling rate (measured on the previous MT)	0.4
Outer maximum dimensions	70x80x5mm
Epoxy thickness	0.2 and 0.5mm

Table 4-1 : Magnetotorquer parameters and requirements

Outer dimensions (without epoxy) l_{out} x L_{out} x w	69x79x4.6mm
Inner dimensions (without epoxy) l_{in} x L_{in}	60.8x70.8mm
Mean dimensions l x L	64.9x74.9mm
Cross-section dimensions (without epoxy) h x w	4.1x4.6mm
Number of turns N	427
Coil resistance R_0 (at 20°C)	116Ω

Table 4-2 : Actual Magnetotorquers theoretical characteristics

Because the wire has a standard diameter, the current in the coil can be greater than 15mA (up to 140mA). Thus **the torque produced by the magnetotorquer is actually only limited by the power consumption and voltage.**

The resistance of the coil is computed using the following equation:

$$R = N\,\rho_0(1-\alpha(T-T_0))\frac{l_{cu}}{S_{wire}} \tag{3.4}$$

Because the temperature variation is very high (T∈[-45, +70]°C on the panels), the coil resistance varies from 85Ω up to 140Ω ! This variation must be compensated.

A resistor R_{MT} must be put in series with the coil to ensure the right current (15mA). This can also be done or improved using software.

4.3.1.2 Coil resistance drift compensation

1. The 1st idea was to use an NTC thermistor for R_{MT} and to put it on the panel of the satellite near the MT to compensate its variation. But this type of component uses semiconductors and has a strongly non linear variation for temperatures below 0°C (see Figure 4-7):

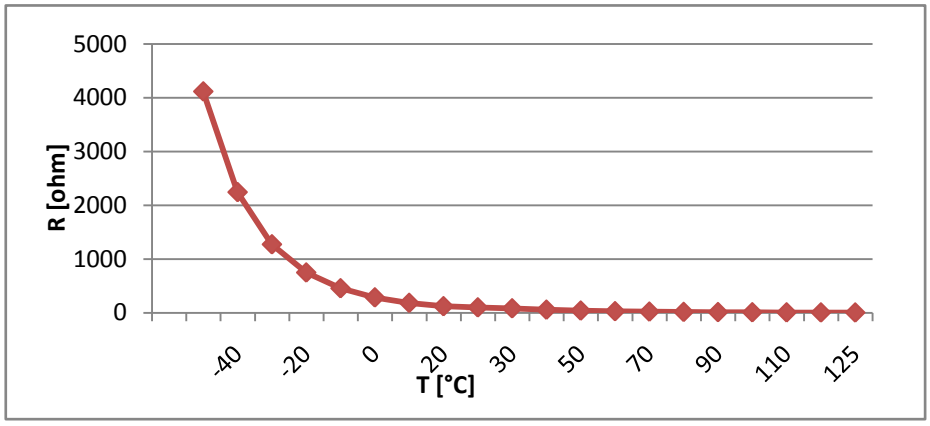


Figure 4-7 : Resistance of a 100Ω EPCOS NTC versus temperature

Therefore this type of resistor cannot be use.

2. As temperature sensors are present on the outer panels of the satellite, a 2nd option is to use their measure to adapt the MT PWM rate (and the mean current) dynamically as a function of temperature. Thus R_{MT} is computed using the maximum coil resistance:

$$R_{MT\ normalized} \leq R_{MT} = \frac{Vcc}{I_0} - R_{max} \quad (3.5)$$

For Vcc=3V, I_0=15mA and R_{max}=140Ω, we obtain R_{MT}=61Ω and **$R_{MT\ normalized}$=56Ω.**

This solution is very simple and it does not need additional hardware, but it has some disadvantages: it does not measure the current, the temperature might not be completely homogenous and it introduces some time constants (due to sensors and MT thermal conductions and thermal capacities).

These temperatures sensors belong to the EPS system, so the temperatures have to be sent to the ADCS using the I^2C main bus.

3. The 3rd option is to measure directly the current using an operational amplifier measuring the voltage between the R_{MT} terminals for each of the 3 MT. These 3 signals are connected to the multiplexers (AD708) and are measured by the 12-bit ADC of the ADCS microcontroller. Then a simple P or PID regulator controls the PWM rate to adapt the current and the produced torque.

This solution is more complex and requires more hardware and so a higher power consumption. Cautions must also be taken with the number of PWM steps (see 4.3.1.5).

For the moment, the 2nd option has been chosen. To ensure good temperature homogeneity, a thermally conductive epoxy should be used. Another problem has appeared here: thermally conductive space-qualified epoxies have a high viscosity. Then it is hard to mould the coil and some bubble may stay in the resin. This can degrade the outgassing properties.

4.3.1.3 MT manufacturing

The MT are quite complex to build. They have been built by Roland Dupuis from the AEM (Atelier d'électromécanique) at the EPFL (see chapter 8).

First, the winding is done using a specific winding stand with the coil dimensions (see Appendix B.2). Then, the coil is heated in an oven to bond the wires together (see Figure 4-10).

Some MT have been moulded with the EPO-TEK 920 epoxy (see Figure 4-8 and Figure 4-9) provided by Polyscience AG. After have mixed and dropped the resin, the mould must be put into vacuum (for about 1h) to decrease the number and size of bubbles. Then the epoxy is cured in an oven (about 80°C during 5-6h; the lower the temperature, the smaller the remaining constrains).

At the end, we obtain MT with precise outer dimension (except on the top because the mould is not closed; see Appendix B.3). It is very rigid. The resin becomes very hard, but because of that it is slightly brittle. Unfortunately, many bubbles remain in the resin. **The final mass is 28g per magnetotorquer.**

Figure 4-8 : Finished/moulded MT view. The small PCB prevents the break of the copper wires during the moulding and facilitates the soldering.

Figure 4-9 : MT detailed view. A break can be observed in the corner.

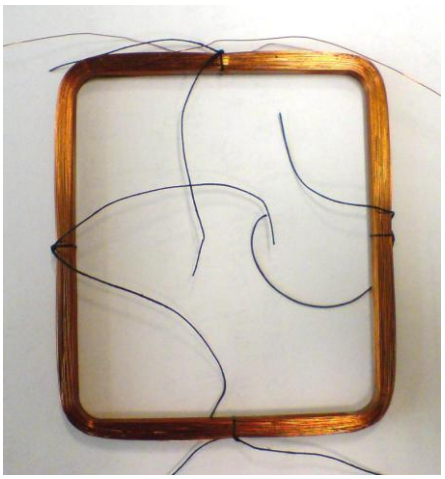


Figure 4-10 : Coil without epoxy resin. The small wires help to maintain the winding. They are also used to centre the coil in the mould.

4.3.1.4 Electrical circuit

The MT are powered through H-Bridges: the microcontroller generates PWM signal and the current is amplified using the H-Bridges. Figure 4-11 shows the schematic:

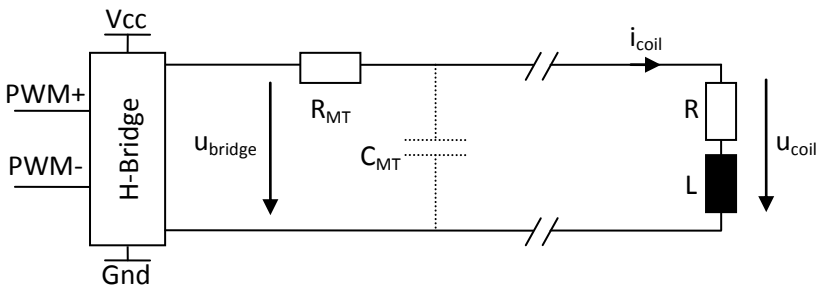


Figure 4-11 : MT electrical schematic

R_{MT} limits the maximum current. An optional filter is made using the capacitor C_{MT}; if it is put on the ADCS board, it lowers the current variation in the wires between the PCB and the MT (create a 2nd order filter for current and 1st order for voltage, see section 5.4.2).

Simple or double PWM can be used:

- For simple PWM, only one of the PWM+ and PWM- signals is a PWM. The other is kept low. The current direction is set inverting the 2 outputs functions.
- For double PWM, the PWM+ and PWM- signals are complementary: one is low and the other is high. If the PWM rate is higher than 50%, the mean current is in one direction and if it is smaller than 50% the current is in the other direction. This PWM mode is often used to control motors because it allows active breaking.

In the previous project, the Si-9987 H-Bridges were selected, but its datasheet mentioned a minimal operating of 3.8V and we will have only 3.0V in the new design because the power will be supplied through an LDO TPS79330 to limits the maximum current in case of a short circuit in a wire. Although these H-Bridges seems to work up to 2V@20°C (see section 5.4.2), this has not been tested for different temperatures and it is probably not a good idea to use this component.

It is very hard to find low-voltage and low-power H-Bridges, but finally another one has been found: the A3901 from Allegro Microsystems. This component contains 2 H-Bridge per package, so only 2 chips are needed. But it is important to mention **its minimal operating temperature is -20°C** according to the datasheet which does not satisfy the -30°C of the requirements. A margin of 20°C has been taken for the requirements with the values given in the Thermal Report [R11]. Therefore I think it would not be a problem. **Measurements should be done to check if this component still works at -30°C.**

In case of option 3 for the resistance drift, the 3 MT currents can be measured through the 3 R_{MT} using 3 differential amplifiers with offset circuits shown in Figure 4-12:

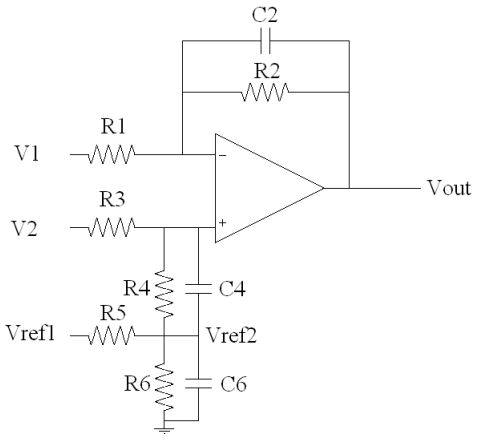


Figure 4-12 : Current measure circuit proposal

With R_1=R_3, R_2=R_4 and C_2=C_4 we obtain the following equation for the output voltage:

$$\underline{v_{out}} = \frac{Z_2}{R_1}\left(\underline{v_2} - \underline{v_1}\right) + V_{ref\,2} = \frac{R_2}{R_1}\frac{1}{1 + j\omega R_2 C_2}\left(\underline{v_2} - \underline{v_1}\right) + V_{ref\,2} \qquad (3.6)$$

Thus with V_{ref1}=2.5V, the reference voltage for the ADC, and V_{ref2}=V_{ref1}/2=1.25V, negative and positive currents can be measured. C_2 and C_4 create a first order optional filter and C_6 stabilizes V_{ref2}.

The ratio R_2/R_1 sets the gain and must be chosen according to the R_{MT} value to ensure an output range of 2.5V. **Caution must be taken choosing the resistors values to ensure small leakage currents and measurement errors**: we must have $i_{coil} \gg i_{R1,3}$ and $i_{R5} \gg i_{R4}$. These currents must be small to have a small current consumption; be careful with the input impedance of the operational amplifier if resistors value is high!

The 3 output voltages v_{out} can be measured using the microcontroller 12-bit ADC through the analog multiplexers (3 different multiplexers, input S8).

As we see the current measure adds a lot of components (3 times this circuit), and it is why the 2nd option has been chosen for the moment. Maybe some chips directly do this circuit function (differential amplifier with offset) with higher input impedance (see instrumentation amplifiers). **Further research must be done on this circuit**.

4.3.1.5 Software

The 6 PWM outputs are made using the Timer_B7 Compare peripherals of the microcontroller.

The PWM frequency must be chosen to ensure an almost constant current in the coils and to lower the perturbations in other lines. To avoid mechanical resonance too, the PWM frequency must be greater than 1kHz, because it correspond to the higher frame natural frequency (see [R15]).

The maximum frequency is limited by the DCO frequency (f_{MCLK}) and the number of increments (or steps) N:

$$f_{PWM} \leq \frac{f_{MCLK}}{2N} \tag{3.7}$$

With f_{MCLK}=6.5MHz and N=100, we obtain $f_{PWM} \leq$32.5kHz. N defines the number of output possible states and then the precision of the PWM setting. This number must not be too small, particularly if a current regulator is used!

Symmetric pulses are currently used for the PWM (Timer_B Up/Down mode), but the maximum frequency can be 2 times greater if non-symmetric pulses are used (Time_B Up mode).

The actual software does not compensate the temperature variation (it does not provide a current regulator either for the 2nd option). **Be careful with the MM; the MT must be switched off while the MM is measuring.** Because this time is quite long, the PWM rate should be corrected to ensure the accurate mean current according to the measure frequency.

The source code can be found in Appendix E.3.4.

4.3.1.6 Model for control algorithms

A simple model has been done with Matlab with the actual MT characteristics (see Table 4-1 and Table 4-2). It considers 3 perpendicular coils and simply adds a Gaussian noise to take into account the small current uncertainty (because of temperature variation which is supposed to be not perfectly corrected). The current is supposed to be a DC current which amplitude is linearly dependant of the PWM rate.

4.3.2 Magnets

Because we have some trouble to correctly control the satellite, other possible controls methods than MT alone have been investigated; this section will presents some results of the researches done about the possibility to use permanents magnets.

A permanent magnet produce the same effect as a MT always switched on, so the satellite (and the magnet) will be aligned with the Earth's magnetic field lines(see Figure 4-13) and will spin around the magnet axis.

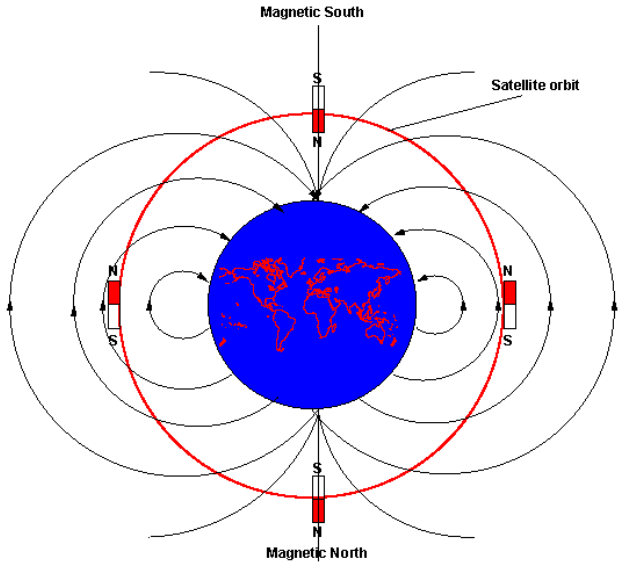


Figure 4-13 : Passive attitude control using a permanent magnet [R13]

Precautions must be taken when choosing the position of the magnet in the satellite, because we must ensure the payload will be able to take goods pictures and the antennas must be in the right direction to guarantee a sufficient gain to communicate with the ground station. A **magnet parallel to the satellite Z axis** (see Figure 3-2) seems to be the best solution.

The main advantages of this method are:

- **No power consumption**; a MT could even be suppressed to decrease the overall mass and power consumption.
- The satellite is always passively controlled when the ADCS is switched off, for example to reduce the power consumption when it is in the eclipse period of the orbit or to recharge the batteries when the satellite is in safe mode.
- The torque produced by a magnet can be greater than the one of the MT (and for a lower mass), so this could help to reject the perturbations.

The main drawbacks are:

- The MM could saturate because of the high magnetic field produced by the magnet. But as we will see in section 5.2.2, this will not be a problem in our case. An offset will simply be added to the MM measurements; this will slightly increase the MM compensation software complexity. This offset must be measured very precisely in order to guarantee the MM precision.
- The satellite will spin around the magnet axis; this rotation might be hard to control. It must not be too high in order to not degrade the payload performances.

Because a permanent magnet does not dissipate energy, the satellite will also oscillate around the Earth's magnetic field lines. This phenomenon can be controlled or minimized using passive hysteresis materials (HyMu80) or the MT (see [R13] and [R14]). The main trouble here is the MT torque will be very small/null because they are almost perpendicular to the Earth's magnetic field if the oscillation amplitude is small. But an advantage of the MT is that because they are actively controlled, they provide a faster detumbling.

Regarding the design of the magnet, some points must also be taken into account: it must survive the space conditions (temperature, demagnetization,…), it must satisfy the launcher requirements (especially it must not interfere with the P-POD and others CubeSats) and the oscillation motion must not excite the structure modes of the satellite.

The equivalent dipole magnetic moment of a magnet can be computed using this equation [R13]:

$$\mu = \frac{B_i V}{\mu_0} \ [Am^2] \tag{3.8}$$

With B_i = Intrinsic induction of magnetic material (same as remanence for fully magnetized materials) [T], V = magnet volume [m^3] and μ_0 = vacuum permeability = $4\pi \cdot 10^{-7}$ [Tm/A].

The magnetic material can be for example Neodymium N35 ($B_i \approx 1.2$T) or AlNiCo-5 ($B_i = 1.28$T). **A simple cylindrical magnet in AlNiCo-5 with Ø6x5mm dimensions will produce a dipole magnetic moment of $576 \cdot 10^{-3}$Am^2 which is about 20 times the one of the MT; its weight is only 4g!**

See [R3] for more information about the Earth's magnetic field, [R13] and [R14] about the passive magnetic stabilization. The free CUBESIM software can be used to design the magnets and compute the oscillation of the satellite.

4.4 Power budget

The ADCS power budget has been corrected and completed according to the new electrical schematic and components.

The ADCS microcontroller is assumed to be in sleep mode (LPM0) when it is not used.

All sensors are assumed to be switched off when no measure is needed. The LDO regulators are not directly taken into account because they simply dissipate overpower and all sensors have the same power consumption with a 3.0V power supply than with a 3.3V. So the power supply can be considered to be 3.3V for all components.

The MT are assumed to be always switched on with their maximum power consumption.

This budget shows the maximum peak currents in order to choose the wires and LDO, and check if the EPS can supply this current. The datasheet typical and maximal values are used for each component in order to have more details.

Part/function	Number	Voltage [V]	Typ Current [µA]		Max Current [µA]		ON Time [ms]	ON frequency [Hz]	Duty cycle	Max Peak Current [mA]	Mean Max Current [mA]	Max Peak Power [mW]	Mean Typ Power [mW]	Mean Max Power [mW]
			ON	OFF	ON	OFF								
Microcontroller :														
MSP430F1611 @7MHz and LPM0	1	3.3	3563	588	4263	693	5	100	0.5	4.26	2.48	14.07	6.85	8.18
Sensors :														
- Magnetometer														
AK8970N	1	3.3	8500	0.1	13000	5	140	1	0.14	13	1.8243	42.9	3.92728	6.02019
NC7SZ66	1	3.3	0.05	0.05	10	10	140	1	0.14	0.01	0.01	0.033	0.00017	0.033
- Sun Sensors														
DTU Sun Sensor	6	0	0	0	0	0	1	1	0.001	0	0	0	0	0
AD8552	12	3.3	1900	0	2150	0	1	1	0.001	25.8	0.0258	85.14	0.07524	0.08514
ADG708	4	3.3	0.0001	0	1	0	1000	1	1	0.004	0.004	0.0132	1.3E-06	0.0132
- Gyroscopes														
IDG1000	2	3.3	8500	0	10500	0	600	1	0.6	21	12.6	69.3	33.66	41.58
- Temperature														
LM94022	2	3.3	5.4	0	9	0	20	1	0.02	0.018	0.00036	0.0594	0.00071	0.00119
								Total sensors :		**59.8**	**14.5**	**197.4**	**37.7**	**47.7**
Actuators :														
- Magnetotorquers														
Coil	3	3.3	15000	0	15000	0	1000	1	1	45	45	148.5	148.5	148.5
H-Bridge	2	3.3	600	0.1	600	0.1	1000	1	1	1.2	1.2	3.96	3.96	3.96
								Total actuators :		**46.2**	**46.2**	**152.5**	**152.5**	**152.5**
								Total :		**110**	**63**	**364**	**197**	**208**

Table 4-3 : ADCS power budget

We can observe the power consumption comply with all ADCS requirements.

5 TESTS

5.1 ADCS main board

5.1.1 DCO frequency

As we have seen in section 4.1.2, an external resistor must be used to provide current to the DCO of the microcontroller to reduce the frequency drift due to the temperature. Because there is no information in the datasheets (and others TI documents) about the influence of the DCO and RSEL registers values when the external resistor is used (DCOR=1), that must be measured to select the operating frequency.

These measures have simply been done using an oscilloscope connected to the SMCLK output pin (pin 49). This output is enabled by the software when an MM measure is started. To ensure a correct precision, the period has been measured on several clock edges (to compute a mean period). These measures have been done with 2 different microcontrollers. The results are shown in Table 5-1.

1) MSP430F169 of the old ADCS board (built by Bastien Despont)

RSEL	DCO							
	0	1	2	3	4	5	6	7
0	2.99E+05							
1								
2								
3								
4				1.89E+06				
5				3.17E+06				
6				4.52E+06	4.95E+06	5.52E+06	**6.20E+06**	6.99E+06
7								

2) MSP430F169 of the actual ADCS test board :

RSEL	DCO							
	0	1	2	3	4	5	**6**	7
0	2.99E+05							
1								
2								
3								
4								
5	2.50E+06			3.30E+06		3.97E+00	4.49E+06	5.08E+06
6			4.23E+06	4.65E+06	5.06E+06	5.81E+06	**6.50E+06**	7.38E+06
7								

Table 5-1 : DCO oscillator frequency [Hz] versus some DCO and RSEL register values. The external resistor Rosc=100kΩ (0.1%, 25ppm/°C, 0805) is used (DCOR =1), the temperature is about 20-25°C and Vcc=3.3V.

We see there is a quite big variation of the DCO frequency between the 2 microcontrollers (<15% according to the datasheet).

Because the maximum MCLK/CPU frequency is 7.35MHz with a 3.3V power supply, the temperature drift -0.1%/°C (according to the datasheet) and the minimum temperature -30°C, we see the maximum parameters are DCO=RSEL=6. **To ensure a small margin I would recommend using DCO=5 and RSEL=6 if the frequency is not measured.** If the CPU speed is not important for the subsystem, a lower frequency can be used to reduce the power consumption.

To compensate correctly the DCO temperature drift for the time clock drift and check if this variation is acceptable for MM (can't be compensated here!), the **DCO frequency should also be measured versus temperature.**

5.2 Magnetometer

5.2.1 Measurement time

The MM measurement time was not measured in the previous project (see [R2]). Because the used MCLK frequency will not be in the MM nominal range (11MHz to 26MHz), the measurement time should be measured to ensure it is not too high for our purpose; indeed if this value is about 0.5-1s, this could be a problem because it reduce the dynamic (by providing an average value for this period) and limits the measurement rate.

This value has simply been measured by looking how long the MCLK (MSP pin 49) is active with an oscilloscope; it is started by the software when an MM measure is started and stopped when the MM generates an interrupt to say the measure is complete. The MM can be configured with an integrator operating time of "10ms" or "20ms". The second value will be preferred because it reduces the sensor noise. I have been able to make work the MM with a 2.5MHz MCLK, although [R2] indicates it does not work below 3MHz. So we see the datasheet takes a very big margin ($f_{min\ nominal}$=11MHz).

MCKL [MHz]	Measurement time [ms]	
	Int. Time 20ms	Int. Time 10ms
2.5	330	
3.3	256	
3.97	208	
4.23	200	86
4.49	190	77
4.65	180	70
5.06	162	60
5.8	146	
6.5	131	
7.38	117	

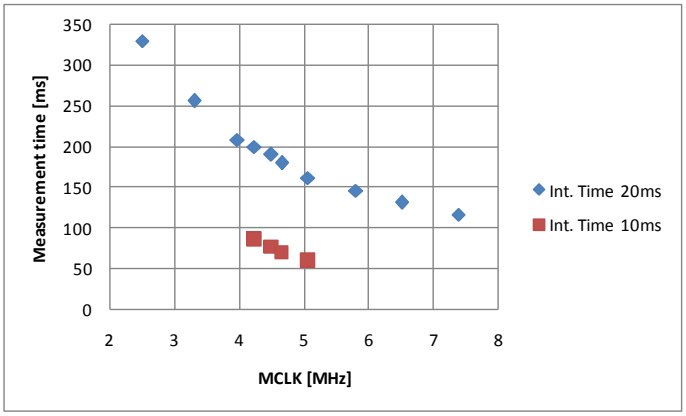


Figure 5-1 : MM measurement time versus MCLK clock frequency

As we can see in Figure 5-1, **the measurement time is 130ms** with a 6.5MHz MCLK and a "20ms" integrator time (at ≈20°C). This is completely acceptable, because the satellite spinning rate will be <1°/s; it will rotate about 0.13° during an MM measure which is below the expected MM precision. We also observe the measurement time variation is not linear for low MCLK frequency and becomes to be rather high.

5.2.2 Offset configuration

The offset of the MM output can be configured with a specific 8-bit register for each axis (be careful: the command is quite strange, see Figure 5-2), but the datasheet does not indicate clearly the offset variation as a function of the value put in these registers. No measure was done about that during the previous project.

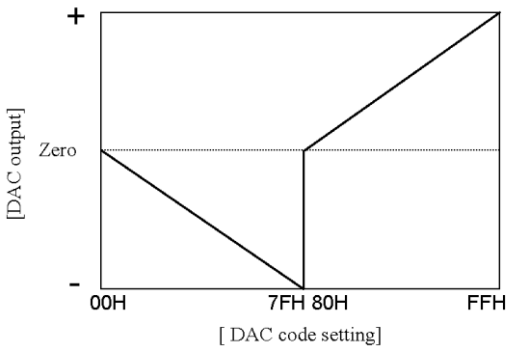


Figure 5-2 : MM offset setting command versus offset

This characteristic is important because if a permanent magnet is put in the satellite, it will generate a high magnetic field which could saturate the MM. Another important point is the MM offset variation due to the temperature must be compensated in real-time in the MM itself by adjusting the offset register to guarantee the measurement range. If the output change is too small (1bit/offsetbit), we will not be able to correct the offset drift because all values are 8-bit values, and if it is too big, it will decrease the MM resolution (see equation (4.1)).

To do these measures, the ADCS board with the MM has simply been kept in the same position and in the same environment; the constant ambient field is measured with the MM. The offset configuration value is changed and the MM output mean value is written. This procedure does not allow finding the 0 offset (output value for a 0T magnetic field). Figure 5-3 shows the measurements results:

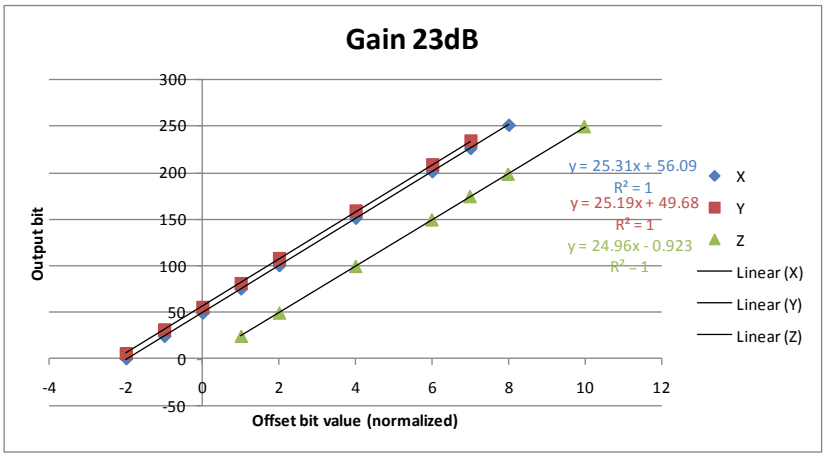


Figure 5-3 : MM output value versus offset setting value. Vcc=3.3V, MCLK=5MHz, gain=23dB, integrator_time=20ms and T≈20°C.

The output offset variation is **$Offset_{sensitivity}$≈25.1bit/offsetbit for each sensor axis.** This value does not change when the MM gain is modified, which is perfectly logical regarding to the sensor architecture (see Figure 4-5). It does not change either when the MCLK frequency is modified; it means that the MCLK mainly influence the MM Chopper SW and influence feebly the MM ADC and SH electronic.

The variation of the offset setting sensitivity versus temperature has not been measured.

5.2.3 Sensitivity and gain

Thanks to the previous measurement, the maximum MM sensitivity can be computed:

$$s = \frac{2B_{max}}{(2^8 - 1) - O_{sensitivity} - Margin} = \frac{2 \cdot 60\mu T}{255 - 25 - 5} = 0.533\,\frac{\mu T}{bit} \quad (4.1)$$

$$or \quad \frac{1}{s} = 1.875\,\frac{bit}{\mu T} \quad (4.2)$$

According to Figure 14 on page 16 in [R2], the MM sensitivity is maximal between 20°C and 50°C. Therefore we can try to obtain the sensitivity computed above at ambient temperature changing the MM gain setting.

To do that, the ambient magnetic field has been measured for different orientation with the MM and using a Gauss-meter as reference; the *MiliGauss,* lent by Pavel Kejik from the LMIS3. It can measure a magnetic field with a resolution of 1mGauss=100nT (it is the probe with the highest resolution I could found). It is quite difficult to put the MiliGauss probe exactly in the same position and orientation of the MM, so some errors are committed.

The results are summarized in table Table 5-2:

Axis	Gain		Sensitivity [bit/uT]	Zero offset [bit]
	Reg. setting	dB		
X	0x0e	22.4	1.739	77.4
	0x0f	22.7	1.851	75.9
	0x10	23	1.909	78.2
Y	0x0f	22.7	1.757	39
Z	0x0f	22.7	1.683	40.6
	0x10	23	1.722	38.8

Table 5-2 : Summary of the gain settings measures. Vcc=3.3V, MCLK=6.5MHz, integrator_time=20ms and T≈20°C.

The maximum measured magnetic field was around 25µT, which is quite small and so introduces a higher unknown on the sensitivity. Unfortunately, only a few gains could be tested because of time constrains.

We can see the gain setting must be tuned to obtain the right sensitivity for each axis. The gain setting could also be used to compensate in real-time the sensitivity drift caused by temperature variation. The zero offset values are quite strange compared to these obtained in [R2], because they are not near 256/2=128. Maybe there was an error with the offset setting during these measurements.

Thus further measurements must be done.

5.2.4 Vacuum

During the previous project, a simple vacuum test was done and concludes that the offset changes under vacuum conditions. This result has to be verified because these tests may have not been done in proper conditions.

Determine and characterize the MM offset change in vacuum conditions is very important, because it will work in vacuum on the satellite.

To do this test, the complete board has simply been put in a vacuum chamber. The MM measures the ambient magnetic field, so the environment must not change during the measurements. Because it was not possible to connect the board to the debugger (there was not enough wires), all measurements have been stored in the microcontroller Flash memory and revered later with the debugger. The detailed procedure description can be found in Appendix C.1.

The main test result is shown in Figure 5-4:

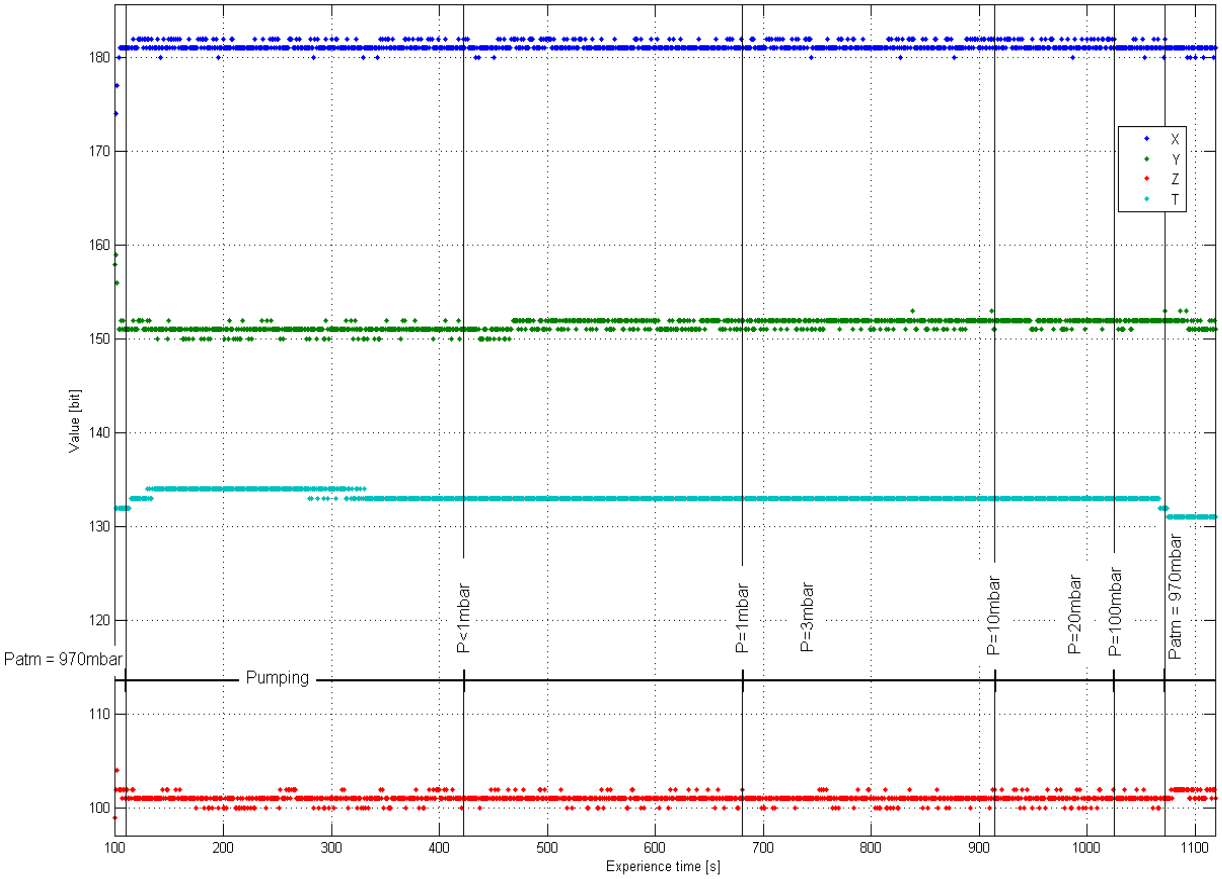


Figure 5-4 : Measures under vacuum conditions

We can observe **there is no significant offset variation** due to the vacuum conditions. The temperature slightly diminish in vacuum (the sensor sensitivity is negative) and this is perfectly logic. But keep in mind the minimum pressure was only ≈1mbar during this experiment.

Another interesting result of this experiment is that the **ADCS time clock error is around 30s for a ≈20mn working time**. So this error is clearly acceptable; but keep in mind the temperature was constant during this experiment.

5.2.5 Integration test with EPS

A very basic integration test has been done with the ADCS and the EPS boards; the ADCS board has been supplied with the EPS board and a battery.

The EPS board uses inductances, voltage converters and can generate high currents (≤1A), thus it could probably generate magnetic disturbances.

The ADCS and EPS board have been placed close to each other (≈1-2cm) and an offset change of about 2 bits has been observed on the MM (with sensitivity≈1.85bit/µT) when the EPS supplied a

high current; this is quite small so we can deduce there will not have any trouble with the MM and EPS because they are separated by more than 3cm.

5.3 Sun sensors

A very simple test has been done with the sun sensors and the ADCS board; the goal was to demonstrate the SS signals can be measured using the ADCS board. The sun sensor PCB (AHW3_4, see electrical schematic in Appendix A.2) has been connected to the ADCS board which was supplied with a 3.3V power supply. A flashlight has simply been used to change the luminosity on the SS.

The result is that the variation of the ambient luminosity can be seen with the microcontroller; the measured voltage increase if the luminosity increase; especially for the reference signals. It is hard to say something about the relationship between the direction of the light, the 2 SS output signals and the 2 SS output references with this experiment. The noise seems to have a standard deviation in the range of 0.7 to 1mV for the output signals and around 0.3mV for the output references. This is small; actually it is in the range of the ADC resolution (0.6mV).

5.4 Magnetotorquers

5.4.1 Impedance measurement

The magnetotorquer impedance characteristic has been measured with an Impedance Analyser (Agilent 4294A) at ambient temperature (20°C). These measurements give us the magnetotorquer equivalent electrical circuit, thus we can know the frequency response and determine an adapted PWM frequency.

The impedance has been measured for a frequency between 1kHz and 50kHz (see Figure 5-5).

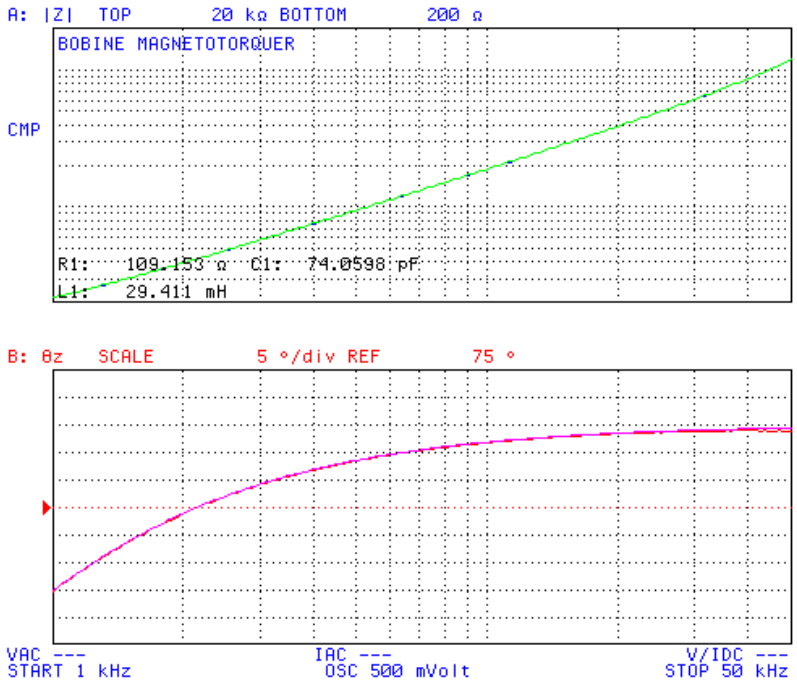


Figure 5-5 : Magnetotorquer impedance (norm and angle) at 20°C versus frequency

As we could expect, the equivalent circuit is mainly composed of a series resistor with an inductance (see Figure 5-6). A small capacitor is also present (due to the wire coating and space between the wires), but it can be neglected.

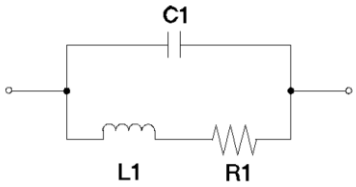


Figure 5-6 : MT equivalent circuit

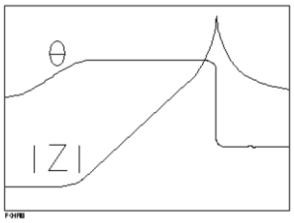


Figure 5-7 : Theoretical impedance (norm and angle) versus frequency of the equivalent circuit

R1 = R_0	109.2Ω
L1 = L	29.4mH
C1	74.1pF

Table 5-3 : MT measured characteristics

5.4.2 PWM rates and filters

The magnetotorquer transfer function can be calculated thanks to the parameters given in Table 5-3 (C1 is neglected). The nomenclature is referred to Figure 4-11 here.

The more interesting transfer function is the one for the current (see Figure 5-8), because there is a direct relationship with the produced torque. The coil itself is an order 1 low-pass filter with a cut-off frequency $\mathbf{f_{coil} = 590Hz}$. When C_{MT} is added, the current filter becomes an order 2 low-pass filter. Thus a good value is $\mathbf{C_{MT}=6.8\mu F}$, because the two cut-off frequency have the same value. **A greater value can also be used** (if it is greater, the cut-off frequency will be lower). The Matlab script "calc_transfert_function.m" has been created to compute these parameters.

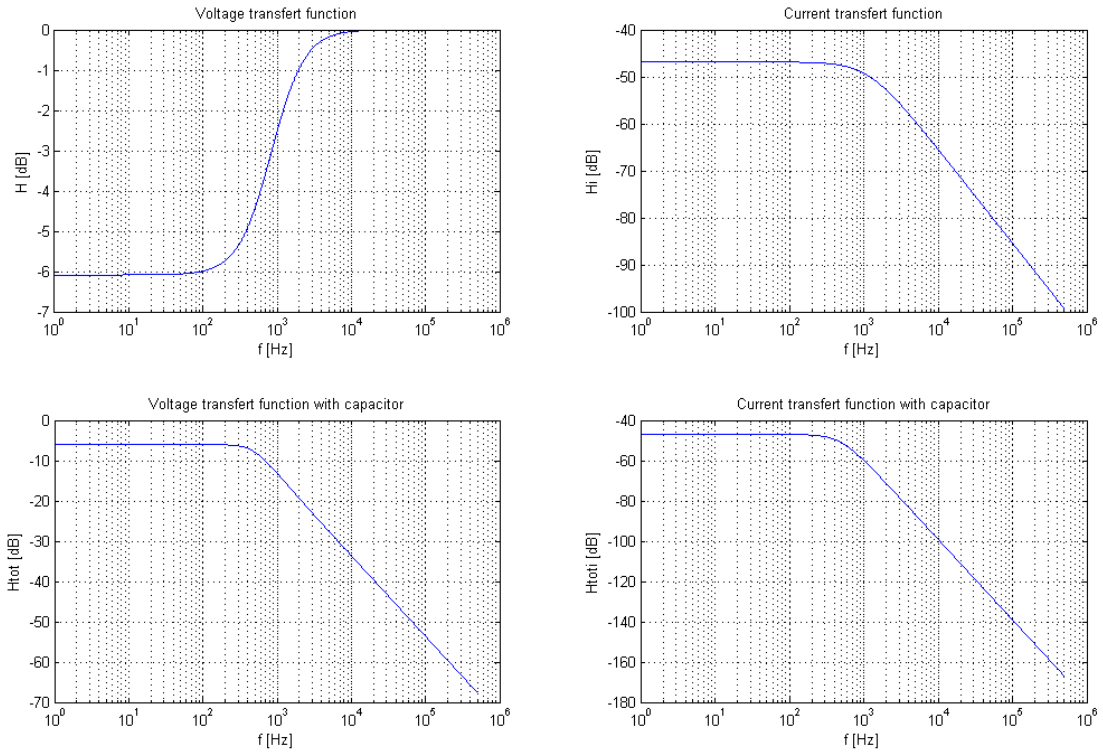
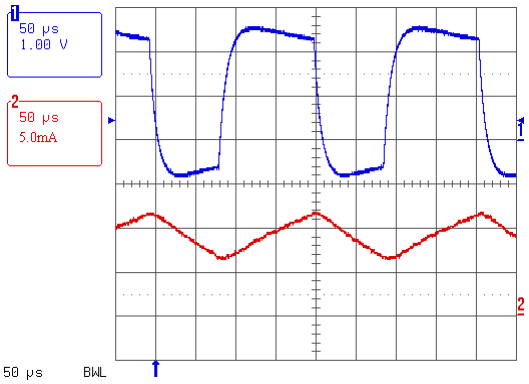


Figure 5-8 : MT current and voltage transfer functions ($\frac{U_{coil}}{U_{bridge}}$ and $\frac{I_{coil}}{U_{bridge}}$), top without C_{MT}, bottom with $C_{MT}=6.8\mu F$

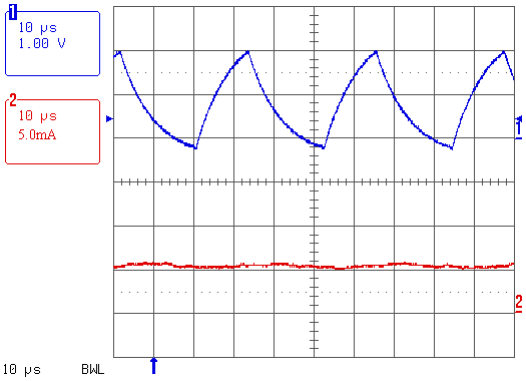
Therefore the PWM frequency should be at least 5 or 10 times greater than f_{coil} to ensure small current variations : **f_{PWM}> ~5kHz**.

The following tests have been done with the Si9987 H-Bridges. Although its datasheet indicates a minimum voltage of 3.8V, they work with the 3.3V power supply (up to 2V at 20°C!). It is probably because the currents are small. Their operation has not been tested for other temperature.

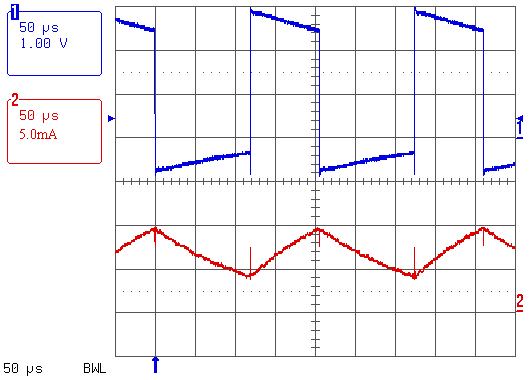
Figure 5-9 shows current and voltage in the coil (and the wires from ADCS board to MT) for different parameters (do not forget to use differential probes). Although the value used for C_{MT} was 100nF, the results should be similar with a higher value (see Appendix B.4 for the transfert functions).



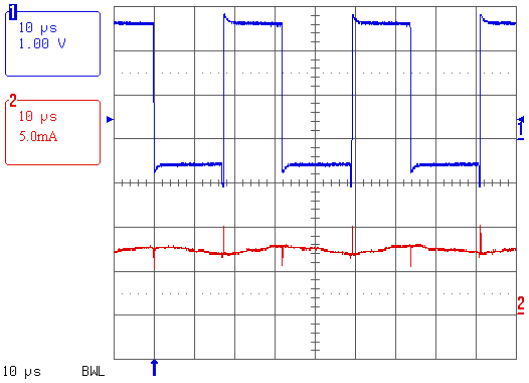
(a) simple PWM, f=5kHz, C_{MT}=100nF



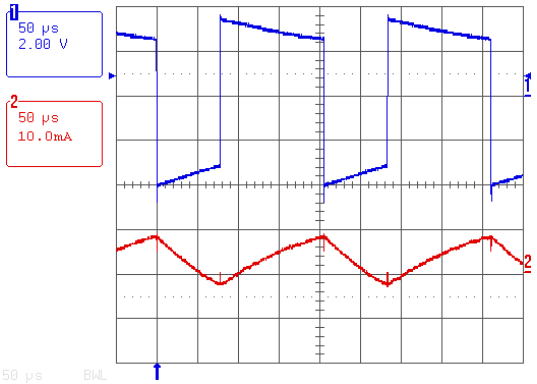
(b) simple PWM, f=32kHz, C_{MT}=100nF



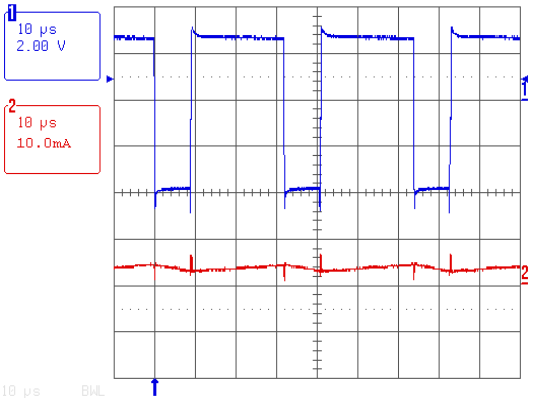
(c) simple PWM, f=5kHz, no capacitor



(d) simple PWM, f=32kHz, no capacitor



(e) double PWM, f=5kHz, no capacitor



(f) double PWM, f=32kHz, no capacitor

Figure 5-9 : MT voltage u_{coil} and current i_{coil} versus frequency, C_{MT} and PWM type

We can observe the current and voltage are smoother if C_{MT} is present (subfigure (a) and (b)) which is perfectly logical. In this case, the current is almost constant for a 32kHz frequency. We also see that there is no small spike due to the commutation of the transistors in the H-bridge. These signals are exactly the ones which stand in the long wires between the ADCS board and the MT.

Therefore adding C_{MT} could help to reduce the noise induced in other lines close to the MT lines, although it is not useful for the MT themselves. Somes further tests should be done to determine if it is better to use a high or a low frequency (~5kHz or ~32kHz). Using a higher frequency than 32kHz should probably not be a good idea, because the MT current will remain constant, but it could generate more noise due to the voltage quick variation.

Although double PWM are often used to control motors, in our case it is also not a good idea to use it because it double the current and voltage variation and so the induced noise in close lines.

5.4.3 Outgassing

Outgassing tests must be done for the MT to ensure they satisfy the contaminations requirements, because the winding wires are not space-qualified. It is also useful to observe the MT mechanical resistance to vacuum conditions, because there is still air between the wires and some bubbles due to the high viscosity of the resin used (EPO-TEK 920).

The outgassing detailed procedure has been prepared by Marie Dumont and can be found in Appendix C.2; the tested MT must stay for at least 24h with controlled temperature and humidity (22±3°C, 55%RH±10%), then it is weighted with a high-precision balance, put in a vacuum chamber for at least 24h (at least 10e-4 Pa, ± 10 %, 125±3°C) and finally weighted a second times.

Unfortunately this test could not have been done during the project, because of a malfunction in the RUAG Nyon vacuum chamber facilities. It will be done as soon as the problem is solved.

One MT has been weighted after having stayed for 24h in a controlled environment: m=27.7544g.

6 RECOMMENDATIONS

A lot of tests remain to be done with all sensors. Regarding the MM, a precise and complete temperature characterisation must be done **with the ADCS board** in order to know:

- The zero offset drift,
- The sensitivity variation,
- The offset setting sensitivity drift (if there is one).
- The noise level must be checked for each axis (see [R2]).
- The internal temperature sensor must be characterized.
- The DCO frequency should also be measured in order to check its variation to correct the time clock.

These tests can be done in the RUAG Nyon vacuum chamber facilities; they gave us authorization to put the ADCS test board in the vacuum chamber (because of contamination problems). These facilities use a cryogenic liquid and a heater to control the temperature using mainly thermal conduction, so they should not generate magnetic perturbations (an electrical resistor or IR heater cannot be used with the MM!). According to the current measures and MM parameters, the precision requirements will be hard to satisfy.

Then the temperature compensation should be implemented in the MM software and validated. One very big problem is that almost each sensor has different characteristics, **so all sensors of the final satellite must be characterized in offset and sensitivity.** The DCO frequency of each MSP should also be measured if it is necessary.

Concerning the MT, outgassing and mechanical resistance tests must be done to check if the present design complies with all requirements and resists to the launcher vibrations, vacuum… or not;

- If it is the case, thermal conduction characteristics should be checked in order to know if it is possible to measure only the temperature of satellite faces to control the MT current in order to satisfy the torque precision requirements (which are currently not defined).
- In the contrary case, another less viscous resin could be used or other MT design solutions could be investigated; for example they could be built using PCB stack. The MT current measure circuit and regulator should also be designed and implemented.

Measurements with the MT H-bridge should be done to check if these components still work at low temperatures (-30°C). Do not forget that the maximum torque is currently only limited by the MT power consumption, so it can potentially be increased (up to 1.8x with 3.0V). Different PWM frequencies should be tested in the Integration Model in order to limit the noise induced on the analog signals lines (SS, GYR, …). The MT produced magnetic should also be characterized.

In the matter of the SS, do not forget to use **precision resistors** for the amplifiers to lower the temperature dependency. **All wires** from the Connection Board could be **enlaced** to reduce the noise.

The new version of the ADCS board should be finalized. Some investigations remain to be done about the ADC reference voltage, the low-pass filters for the analog signals, the latch-up protections and the MT current measurement. Do not forget to put a **measurement pad** to measure in safety the microcontroller **MCLK frequency (pin 49)**. Keep also the useful debug LED.

The ADCS software must be completed; be careful with the starting times in order to reduce power consumption. **Do not forget the MT should not be on when MM is measuring.**

7 CONCLUSION

During this project, almost all functionalities of the ADCS board were integrated and tested (magnetometers, magnetotorquers, sun sensors, gyroscopes and temperature sensors, but not I2C); the first version of the ADCS microcontroller software has been written and a second iteration has been done about the ADCS hardware. Recommendations were proposed for a third revision about the hardware. Although there is still a lot of work and tests to carry out, especially about temperature tests and compensation software, the present ADCS systems is able to provide basic sensors readings and actuators control.

But we still do not know if the whole ADCS system and hardware will comply with the determination and control precision requirements. Many efforts must be put in this direction at the beginning of the next projects.

An efficient calibration methods remains to be found for the gyroscopes. A detailed and complete model of the sun sensors (taking into account the 6 sensors) must be created; for that a model providing the direction of the sun as a function of the date, satellite orbit and position, therefore linked to the propagator, should probably be built.

According to the last news from the Automatic control Laboratory, the presence of a permanent magnet will not help to control the satellite, but on the contrary it will reduce its controllability; the magnetotorquers will always stay nearly perpendicular to the Earth's magnetic field lines and will have few effects. Therefore we do not have an efficient method to control the SwissCube for the moment. This is a major issue for the ADCS.

Lausanne, 23/06/2007

Hervé Péter-Contesse

8 ACKNOWLEDGMENTS

I would like to thank Peter Bruelmeier and André Badertscher for their great help about the PCB layout, components soldering and PCB error corrections.

I give thanks to Roland Dupuis for having winded and molded the magnetotorquers. I would never have been able to build them without him.

Many thanks to Paolo Germano and to the LAI team for having provided to me a working place in their lab, all intruments I needed and some advices, although the fact I was not working for the LAI.

Thanks to the SwissCube team for the nice working atmosphere and their help.

9 CONTACTS

Peter Bruelmeier	peter.bruehlmeier@epfl.ch	ACORT - PCB layout
André Badertscher	andre.badertscher@epfl.ch	ACORT - SMD component soldering
Roland Dupuis	roland.dupuis@epfl.ch	AEM - Electromechanical workshop, MT
Jean-Paul Brugger	jean-paul.brugger@epfl.ch	AEM - Electromechanical workshop (chief)
Beatrice Iten	b.iten@polyscience.ch	Polyscience AG - EPO-TEK supplier
Robert Owen	rcwo@ti.com	TI - University Programme Manager / Supplier for the 40x MSP430F1611
Paolo Germano	paolo.germano@epfl.ch	LAI - Magnetic circuits
Pavel Kejik	pavel.kejik@epfl.ch	LMIS3 - Magnetometers

10 REFERENCES

[R1] Bastien Despont, *S3-B-ADCS-1-4-AHWreport*, EPFL, February 2007

[R2] Vasko Vitanov, *S3-B-ADCS-1-3-Magnetic_sensor,* EPFL, February 2007

[R3] Hervé Péter-Contesse, *S3-C-ADCS-1-2-Earth's Magnetic Field Model*, EPFL, May 2007

[R4] Kaspar Jenni, *S3_Phase_B-C-ADCS-1-3-Gyroscopes*, EPFL, May 2007

[R5] Andres Schmocker, *S3_Phase_B-C-ADCS-Sun sensors*, EPFL, May 2007

[R6] Daniel Hakansson, *S3_Phase_B-C_Mission_design*, EPFL, May 2007

[R7] Jordi Martin-Benet, *S3_Phase_B-C-ADCS-SwissCube Attitude Determination Algorithm Design and Validation*, EPFL, May 2007

[R8] M. Noca, *SwissCube Project Specifications*, EPFL, January 2006

[R9] Dumont Marie, *S3_B_SE_1_1_specrules*, EPFL, February 2007

[R10] R. Krpoun, M. Noca, N. Scheidegger, *S3-B-SET-1-2-Mission_System_Overview,* EPFL, February 2007

[R11] Oscar de la Torre, *S3-B-TCS-1-1-Thermal_Management*, EPFL, February 2007

[R12] Torben Graversen, Michael Kvist Frederiksen, Søren Vejlgaard Vedstesen, *Attitude Control system for AAU CubeSat*, Aalborg University, June 2002

[R13] Lars Alminde, *Semi-Active Attitude Control and Off-line Attitude Determination for SSETI-Express*, Aalborg University, June 2004

[R14] Jean-Francois Levesque, *Passive Magnetic Attitude Stabilization using Hysteresis Materials*, SIgMA, Sherbrooke University

[R15] Guillaume Roethlisberger, *S3-B-STRU-1-4-StructureConfiguration*, EPFL, February 2007

[R16] Texas Instruments, *MSP430x1xx Family User's Guide (slau049f)*, TI, 2006

11 ABBREVIATED TERMS

ADC	Analog to Digital Converter
ADCS	Attitude Determination and Control System
CB	Connection Board
CDMS	Command Data Management System
DCO	Digitally Controlled Oscillator
EMF	Earth's Magnetic Field
EPS	Electrical Power System
GYR	Gyroscope
ICD	Interface Control Document
LDO	Low Dropout linear voltage regulator
LUT	Look-up Table
MM	Magnetometer
MSP	Mixed Signal Processor
MT	Magnetotorquer
PWM	Pulse Width Modulation
RMS	Root Mean Square
SS	Sun Sensors
STK	Satellite Tool Kit
SV	Secular Variation of the Earth's magnetic field
TC	Telecommand
TL	Telemetry
TLE	Two Line Element from NORAD

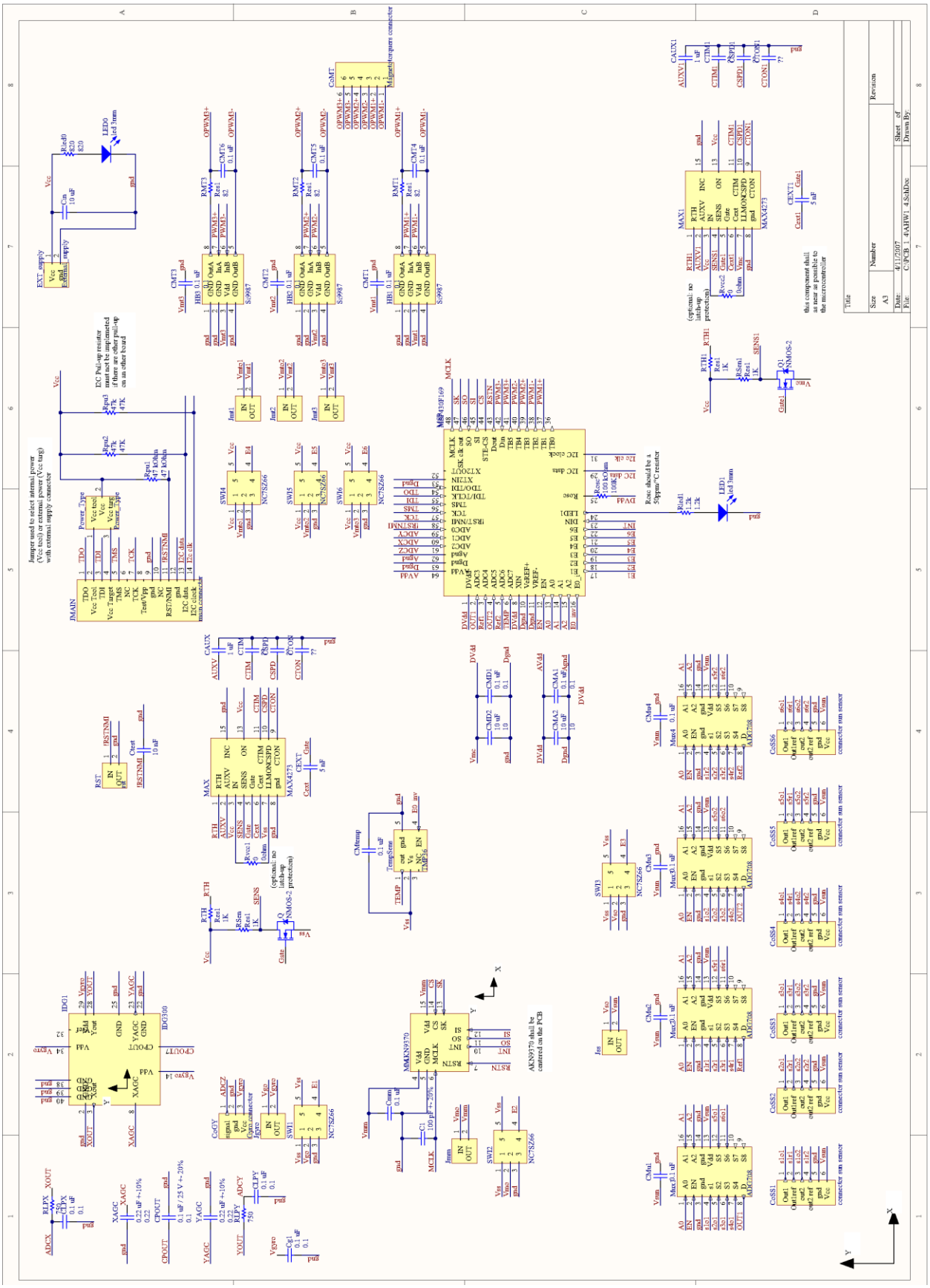
Appendix A ADCS board electrical schematic and PCB

A.1 AHW1_4

A.1.1 Electrical schematic

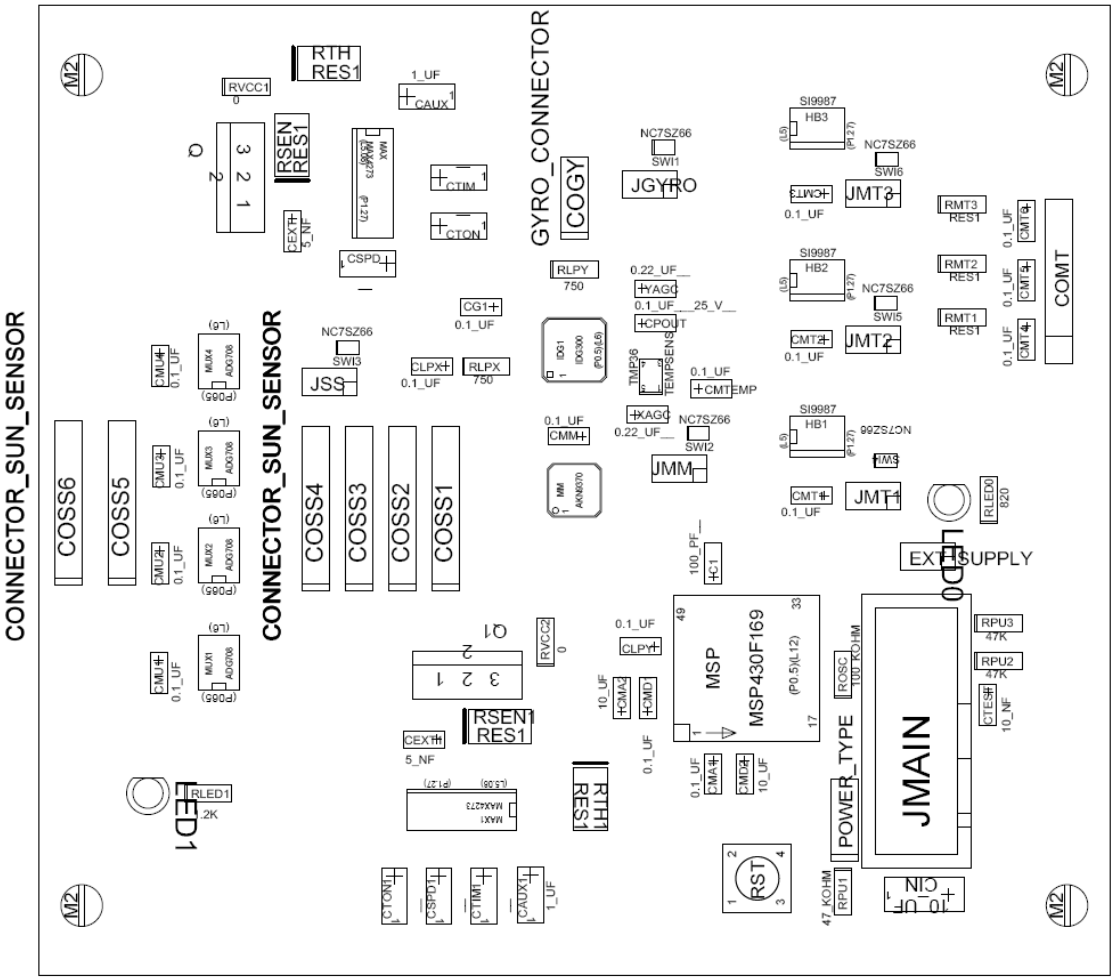
Remaining errors:

- A 10µF capacitor should be placed between MSP430 pin7 and GND to ensure ADC correct operation.
- The MCLK signal for the magnetometer must use the SMCLK signal from the MSP430 (pin 49), because the MCLK from pin48 is stopped when the microcontroller is in sleep mode.



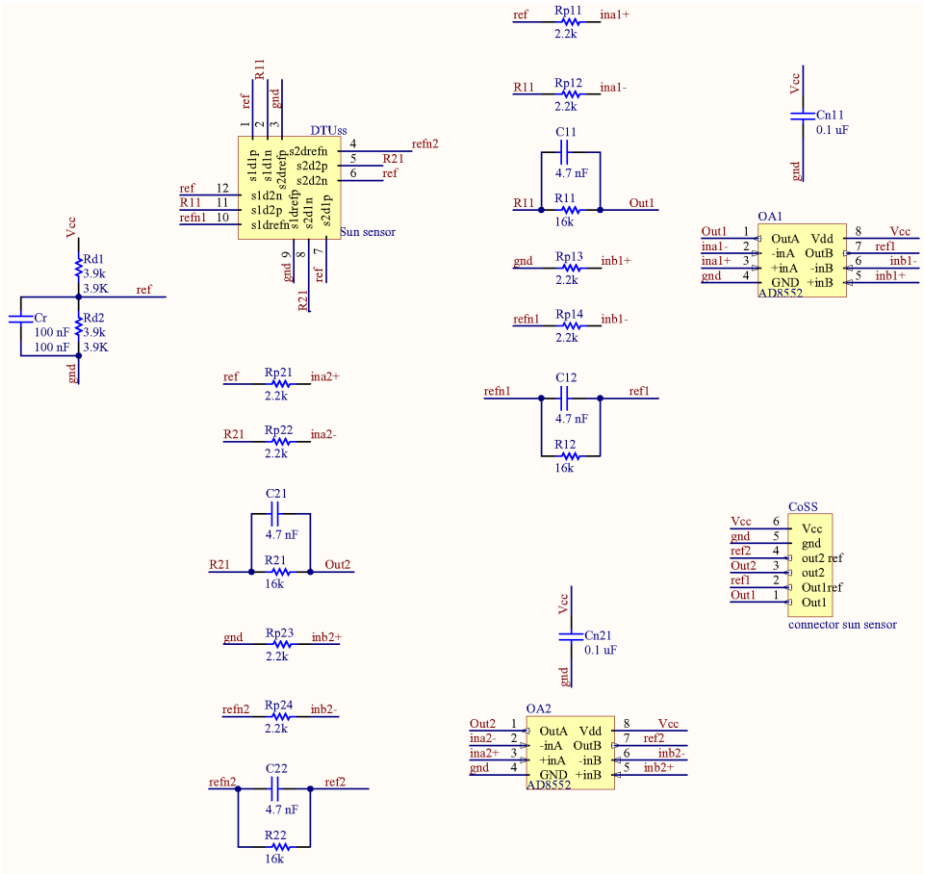
A.1.2 **PCB**

Component placement:

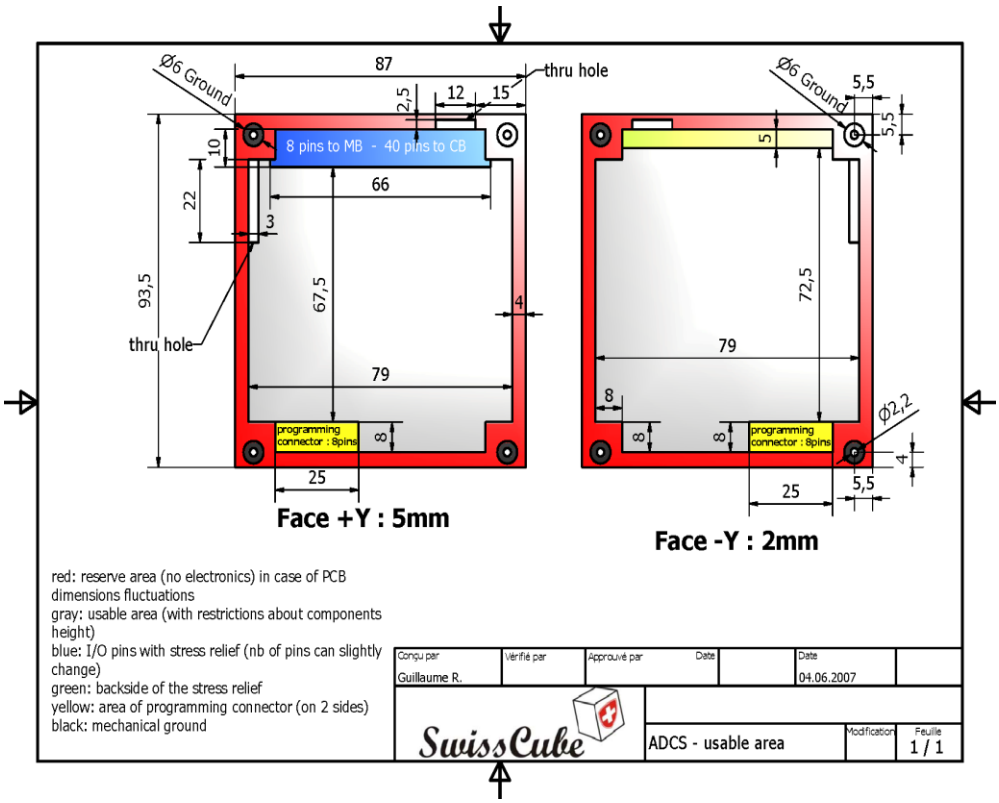


The file containing the complete layout is available on the CD-ROM (ahw1_c_5.pdf).

A.2 Sun sensors electrical schematic

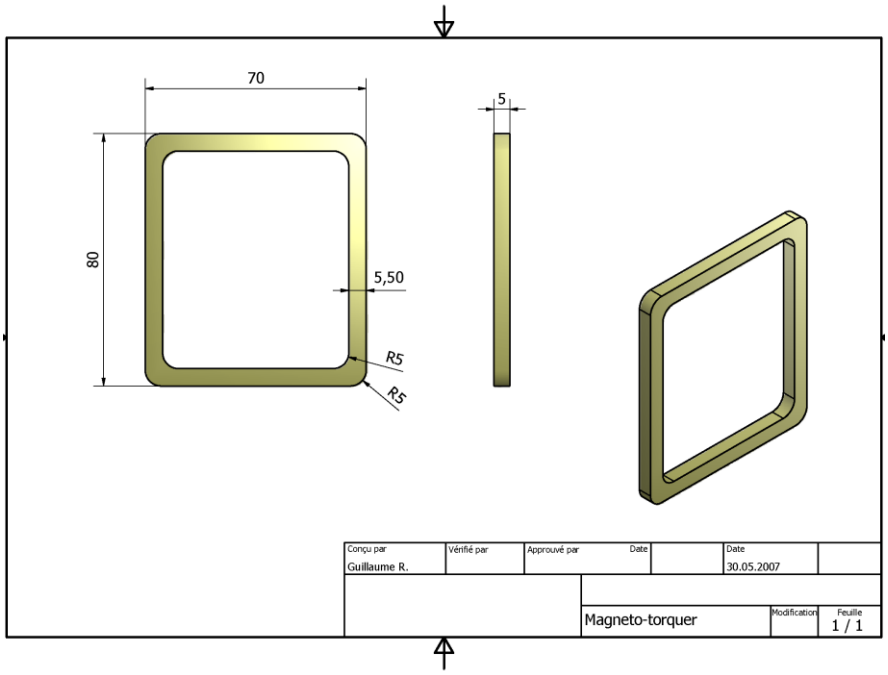


A.3 ADCS PCB usable area for the next board



Appendix B Magnetotorquers drawings

B.1 Outer dimensions



B.2 Winding parts

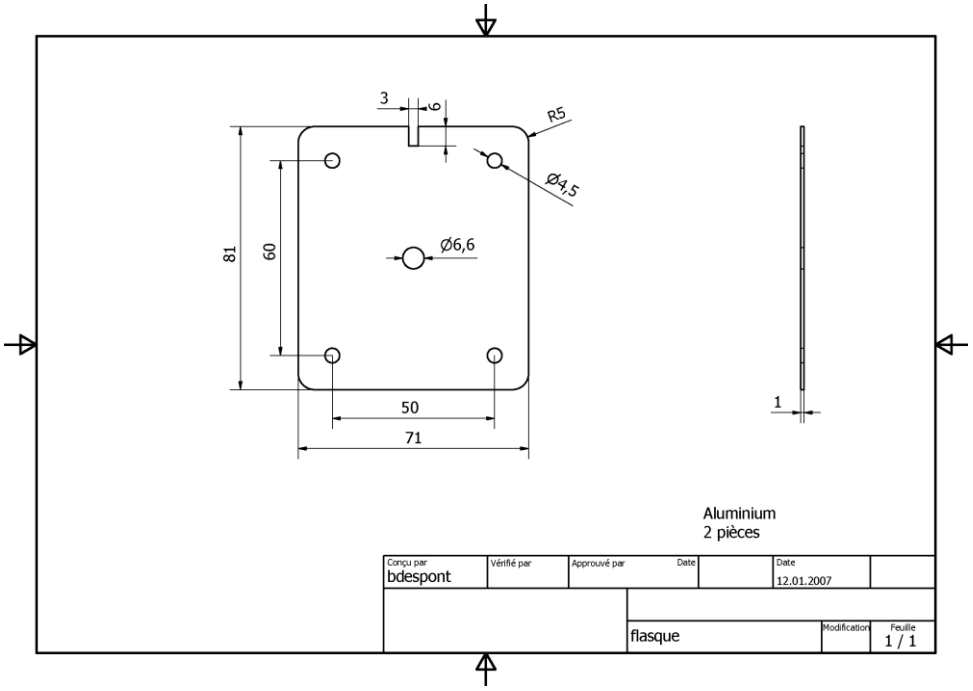


Figure 11-1: Coil winding flange

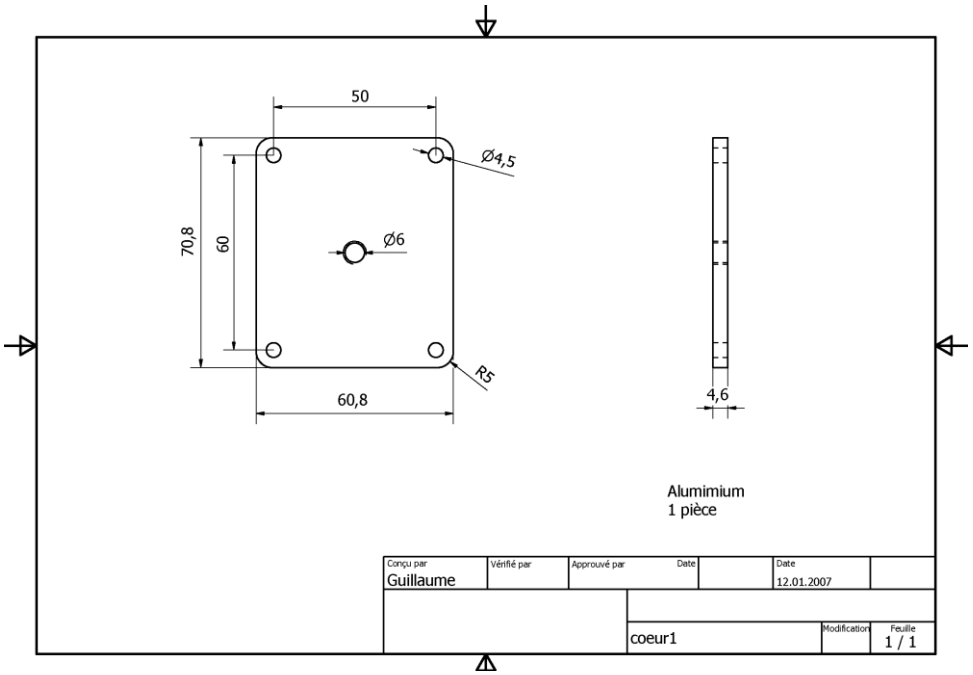
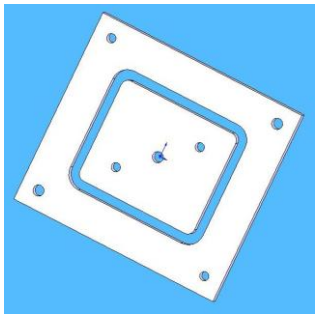


Figure 11-2 : Coil winding heart

B.3 Mould parts



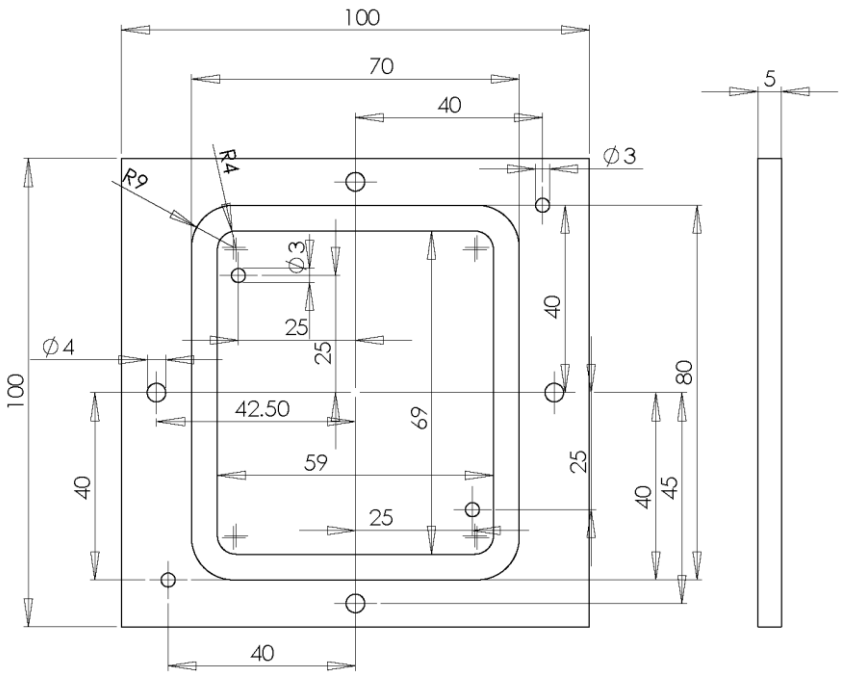


Figure 11-3 : Two mould parts. The last part is a simple plate with the holes for the pins.

B.4 Transfer function

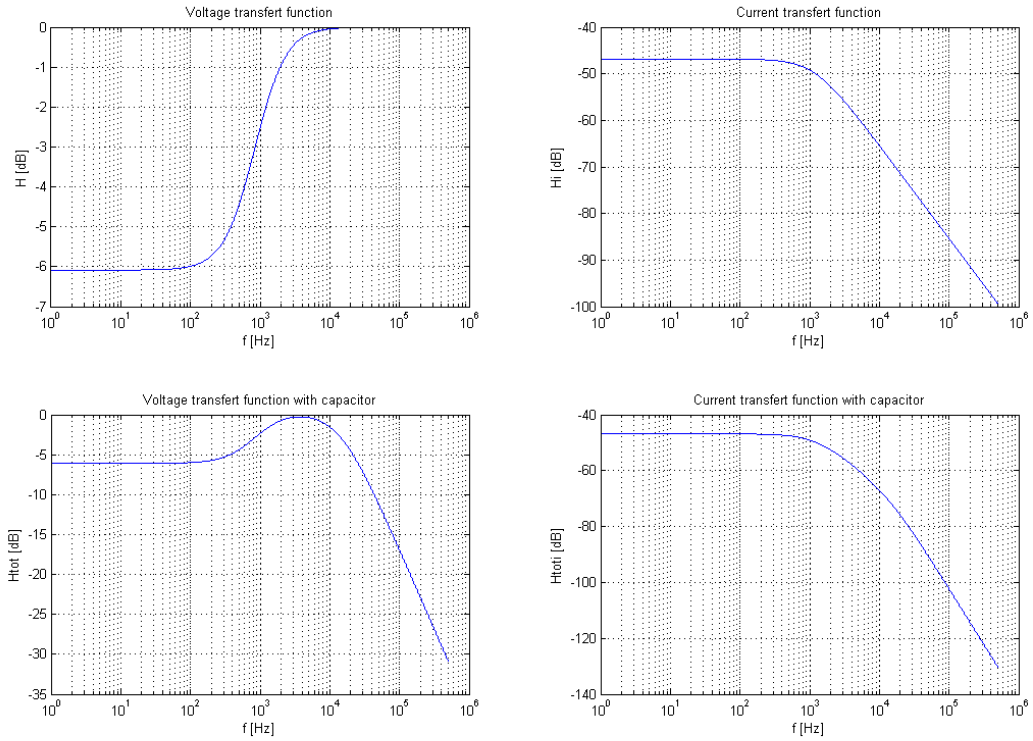


Figure 11-4 : MT voltage and transfer function for C_{MT} = 100nF

Appendix C Test procedures

C.1 Magnetometer Vacuum test

Additional information and measurement results can be found in the file "Results_vacuum_test_22mai07.doc"

C.1.1 Hardware

The magnetometer is tested using the ADCS main board (which is still a test board), so the whole board is put in a vacuum chamber. The board is powered using a stabilized power supply at 3.3V. The power supply wires are the only wires which go into the chamber, so all measurements are stored into the internal flash memory of the microcontroller.

The vacuum chamber is the one of Samuel at the CSEM in Neuchâtel. A barometer which can measure between 1bar and 1mbar has been used. The vacuum pump seems to go up to ~1e-5mbar but this pressure has never been reached, because it could not be measured.

A watch with a chronometer has been used to measure the time.

C.1.2 Procedure

1. Clean the chamber and put an appropriate gel on the seal

2. Put the ADCS board in the chamber with the power supply and the programmer/debugger connected
3. Put the top on the chamber without screwing it down
4. Adjust the MM gain to a desired value and program the microcontroller
5. Start the program (measures) and adjust the MM offset to obtain an output value around 128 for each axis.
6. Re-program the microcontroller (the program can only be launched once!)
7. Start the program, the chronometer and disconnect the debugger
8. Close the chamber by screwing the top and wait a moment (~1mn) → reference values at ambient pressure
9. Switch on the vacuum pump
10. Stop the pump when the pressure is under 1mbar
11. Open the valve slowly to increase the pressure, then wait ~1mn
12. Repeat 11 until ambient pressure is reached
13. Wait ~ 1mn
14. Unscrew the top
15. Stop the measurements using the reset push button of the board
16. Connect the debugger without programming the memory and recover the measurements using the memory view (column space 1 or 2 and unsigned integer rendering). Paste the memory view containing the data in a text file.
17. Begin at 2 if another measure must be done

All actions since 7 have been recorded with the time and the pressure. The environment around the vacuum chamber should not change during the experiment, because we measure the ambient magnetic field!

The temperature was about 25°C, even thought it has not been measured (except by the internal temperature sensor of MM).

C.2 Magnetotorquer outgassing

The magnetotorquer outgassing procedure has been created by Marie Dumont.
See *S3-C-1-0-Outgassing_magnetotorquers*.

Appendix D Interface Control Documents

D.1 Electrical

PIN #		PIN NAME	TYPE	I/O	Voltage [V]	Mean max Current [mA]	Peak max current [mA]	To (in case of output)	From (in case of input)	Purpose / Description
J5	1	MT-X(+)	Power	O	0 - 3		15	MT -X		Magnetotorquers PWM signals and power
	2	MT-X(-)	Power	O	0 - 3		15	MT -X		
	3	MT+Y(+)	Power	O	0 - 3		15	MT +Y		
	4	MT+Y(-)	Power	O	0 - 3		15	MT +Y		
	5	MT-Z(+)	Power	O	0 - 3		15	MT -Z		
	6	MT-Z(-)	Power	O	0 - 3		15	MT -Z		
	7	GND_GYR	Power	O	0	6.3	10.5	GYR +Y		Stabilized power supply for Connection Board GYR and temperature sensor
	8	V_GYR	Power	O	3	12.6	21	GYR +Y		
	9	GND_GYR	Power	O	0	6.3	10.5	GYR +Y		
	10	GYR+Y	Analog	I	0-2.5				GYR +Y/CB	CB GYR analog signal
	11	CB_TEMP	Analog	I	0-2.5				CB	Connection Board temperature sensor
	12	GND_SS	Power	O	0	0.0129	12.9	SS		Stabilized power supply for all sun sensors
	13	V_SS	Power	O	3	0.0258	25.8	SS		
	14	GND_SS	Power	O	0	0.0129	12.9	SS		
	15	SS-X_s1	Analog	I	0-2.5				SS -X	Sun sensors analog outputs: 2 signals and 2 references per sun sensor
	16	SS-X_r1	Analog	I	0-2.5				SS -X	
	17	SS-X_s2	Analog	I	0-2.5				SS -X	
	18	SS-X_r2	Analog	I	0-2.5				SS -X	
	19	SS+X_s1	Analog	I	0-2.5				SS +X	
	20	SS+X_r1	Analog	I	0-2.5				SS +X	
	21	SS+X_s2	Analog	I	0-2.5				SS +X	
	22	SS+X_r2	Analog	I	0-2.5				SS +X	
	23	SS-Y_s1	Analog	I	0-2.5				SS -Y	
	24	SS-Y_r1	Analog	I	0-2.5				SS -Y	
	25	SS-Y_s2	Analog	I	0-2.5				SS -Y	
	26	SS-Y_r2	Analog	I	0-2.5				SS -Y	
	27	SS+Y_s1	Analog	I	0-2.5				SS +Y	
	28	SS+Y_r1	Analog	I	0-2.5				SS +Y	
	29	SS+Y_s2	Analog	I	0-2.5				SS +Y	
	30	SS+Y_r2	Analog	I	0-2.5				SS +Y	
	31	SS-Z_s1	Analog	I	0-2.5				SS -Z	
	32	SS-Z_r1	Analog	I	0-2.5				SS -Z	
	33	SS-Z_s2	Analog	I	0-2.5				SS -Z	
	34	SS-Z_r2	Analog	I	0-2.5				SS -Z	
	35	SS+Z_s1	Analog	I	0-2.5				SS +Z	
	36	SS+Z_r1	Analog	I	0-2.5				SS +Z	
	37	SS+Z_s2	Analog	I	0-2.5				SS +Z	
	38	SS+Z_r2	Analog	I	0-2.5				SS +Z	
	39	NC	Unused							Reserve
	40	NC	Unused							Reserve
J6	1	Vcc	Power	I	3.3	32	55		EPS	ADCS Board, sensors and acutators power supply
	2	Vcc	Power	I	3.3	32	55		EPS	
	3	GND	Power	I	0	32	55		EPS	
	4	GND	Power	I	0	32	55		EPS	
	5	I2C_data	Digital	IO	0 - 3.3				Main bus	Main communication Bus
	6	I2C_clock	Digital	IO	0 - 3.3				Main bus	
Jmec	1	GND_FRA	Thermal	I					Mec	Frame ground for thermal dissipation

D.2 Data

Data direction	Description	Data type	Data length [bits]	Units	Nominal range	Max values (due to data type)
ADCS --> CDMS	Magnetometer X,Y and Z	3x int16 + 1x Uint32 (TS)	80	1e-7 Tesla	± 60.0µT	± 3276.7µT
	Sun Sensors -X,+X,-Y,+Y,-Z and +Z	12x int16 + 1x Uint32 (TS)	224	0.1 Degree	± 70.0°	± 3276.7°
	Gyroscopes X,Y and Z	3x int16 + 1x Uint32 (TS)	80	1e-3°/s	± 30.000°/s	± 32.767°/s
	ADCS Temperature	1x int8	8	°C	-30 to + 60°C	± 127°C
	Connection Board temperature	1x int8	8	°C	-30 to + 60°C	± 127°C
CDMS --> ADCS	Magnetotorquers Torque/rate X,Y and Z	3x int8	24	% from max value	± 100%	± 127%
EPS --> ADCS	Panels Temperature -X,+Y and -Z	1x int8	8	°C	-45 to + 70°C	± 127°C

TS = Time Stamp, Uint16 = 16 bit unsigned interger

Appendix E Software

E.1 Perturbation calculation

The perturbations have been calculated using the Matlab script "disturbances.m" from Bastien Despond.

E.2 Magnetotorquer design

The magnetotorquers have been designed using the completed Matlab script "heart.m" and "dimensioning.m" (created by Bastien Despond). The dimensions results are in "Coil_Result.m".

The scripts "calc_transfert_function.m", "calc_impedance.m" and "calc_resistance_drift.m" have been created to compute the electrical coil parameters.

E.3 ADCS SW

E.3.1 Microcontroller programming in C recommendations

This appendix explains the basics for programming microcontrollers in C code (for use on the 16bit-MSP430 and 32bit-ARM7 microcontrollers).

- Microcontrollers have a small computation capability: divisions and multiplications takes many CPU cycles, so their number should be minimized. Additions, subtractions, shift,… takes few CPU cycles (1 or 2).
- To do a multiplication by 2^x , type : $a = b \ll x$
- To do a division by 2^x , type : $a = b \gg x$
- **Never use floating point numbers (*double* or *float*)!** The microcontrollers have no FPU (floating point unit) and all floating point operations are emulated in software with the C

compiler. For example, a simple addition with 2 floats will take about 250 CPU cycle and only 1 or 2 cycles with integer.

- **Integers should be used!** For the MSP430: if the number can be stored in a 16bit integer (*int* or *short int*), don't use a 32bit integer (*long int*)! For the ARM7, all computations can be done with 32bit integer, except for tables, where you should minimize the memory size using 8bit (*char*) or 16bit integer if it's possible.
- **Always use look-up tables** to compute a sinus, cosines or other complex functions!

E.3.2 Main file and General parameters

```
//*****************************************************************************
/*
* File : main.c
* Created by Hervé Péter-Contesse
* April 2007
*
* SwissCube project - ADCS :
*
* Main program for testing all functionalities of the ADCS microcontroller and
* do measurements with all sensors in order to caracterize their operations
*/
//-----------------------------------------------------------------------------
/*
 * Microcontroller Pin configuration :
 * - LED1 P2.4
 *
 * - Temperature sensor LM94022
 *       E0               P1.4
 *
 * - Magnetotorquers
 *  -- MT1-X
 *       E4               P2.0
 *       PWM1+    P4.1
 *       PWM1-    P4.2
 *  -- MT2+Y
 *       E5               P2.1     --> will be supressed !!!
 *       PWM2+    P4.3
 *       PWM2-    P4.4
 *  -- MT3-Z
 *       E6               P2.2     --> will be supressed !!!
 *       PWM3+    P4.5
 *       PWM3-    P4.6
 *
 * - Magnetometer AK8970N (USART1)
 *       E2               P1.6
 *       INT      P2.3
 *       RSTN     P4.7
 *       CS               P5.0
 *       SI               P5.1
 *       SO               P5.2
 *       SK               P5.3
 *       MCLK     P5.5
 *
 * - Multiplexers
 *  EN           P1.0 : enable Multiplexers
 *  A0           P1.1 : change gate
 *  A1           P1.2
 *  A2           P1.3
 *
 * - Sun sensors
 *  E3           P1.7
 *
 * - gyroscopes
 *  E1           P1.5
 *
 * - Analog Input
 *  GYR+X        P6.0/A0 !!!!!!!!!!!!!
 *  GYR+Y        P6.1/A1 !!!!!!!!!!!!!
 *  GYR+Z        P6.2/A2 !!!!!!!!!!!!!
 *  SS#_s1       P6.3/A3  : sunsensors 1-6 signal 1
 *  SS#_r1       P6.4/A4 : sunsensors 1-6 reference 1
 *  SS# s2       P6.5/A5
 *  SS#_r2       P6.6/A6
 *  Temp P6.3/A3  : ADCS board temperature
 *  CB_temp P6.4/A3 : Connection board temperature
 *  (I_MT-X      P6.3/A3 : magnetotorquer current !!!!!!!!!!!)
 *  (I_MT+Y      P6.3/A4 !!!!!!!!!!!!)
```

```
 *  (I_MT-Z      P6.3/A5 !!!!!!!!!!!!)
 *
 */
 //-------------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_general_parameters_and_functions.h"
#include "adcs_magnetometer.h"
#include "adcs_magnetotorquers.h"
#include "adcs_sunsensors.h"
#include "adcs_gyroscopes.h"
#include "adcs_temperature.h"

#include "flash.h"

//-------------------------------------------------------------------------------
// Global variables declaration
//-------------------------------------------------------------------------------

TIME ADCSTime = 0;                // time counter, incremented each T_TIMERA0
unsigned short int fmclk = FMCLK0;        // actual MCLK clock frequency [kHz], to be adapted with temperature
char ADCSBoard_temp = TEMP0;      //[°C] ADCS board temperature (external sensor)
char CB_temp = TEMP0;                     //[°C] Connection board temperature

unsigned char offset = 0x80;      // must be removed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
unsigned char gain = 0x00;                //0x00 = 18.2dB, 0x10 = 23dB, 0x1F = 27.5dB // must be removed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//unsigned char dco = 0x06;               // must be removed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

char pwmX =0;                                     // must be removed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
unsigned int i=0;                                 // must be removed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

//-------------------------------------------------------------------------------
// Internal functions declaration
//-------------------------------------------------------------------------------

void ADCSTime_correct_temp_drift(char temperature);

//-------------------------------------------------------------------------------
// MAIN PROGRAM
//-------------------------------------------------------------------------------
void main(void)
{
        //-------------------------------------------------------------------------------
        // INITIALISATIONS
        //-------------------------------------------------------------------------------

        WDTCTL = WDTPW + WDTHOLD;// Stop WDT

        //-------------------------------------------------------------------------------
        // Configure Clock system : max frequency 7.35MHz!!!
        //   ~6.5MHz frequency with Rosc 100kohm 25ppm/°C
        DCOCTL=DCO1+DCO2;         // DCO control register
        BCSCTL1=XT2OFF+RSEL1+RSEL2;               // Basic clock system control register 1 : disable XT2
        BCSCTL2=DCOR;             // Basic clock system control register 2 : use DCO for all and Rosc

        //-------------------------------------------------------------------------------
        // Configure all Pins

        // - Sun sensors (SS) and Enable Power (E) pins (E0-E6 ; P1.4-P1.7 and P2.0-P2.2)
        P1SEL = 0x00;    // all IO for SS and E
        P1OUT = 0x00;    // all 0
        P1DIR = 0xFF;    // all ouput for MUX and Enable pins E...

        P2SEL = 0x00;    // all IO
        P2OUT = 0x00;
        P2DIR = 0xD7;    // all output except P2.3(INT MM) and P2.5 (Rosc)

        // - I2C Pins
        //P3.....................

        // - Magnetotorquers (and MM RSTN P4.7)
        P4SEL = 0x7E;    // P4.0 and P4.7 IO, other peripherals !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        P4OUT = 0x00;    // RSTN is not set for the moment (MM reset)
        P4DIR = 0xFF;    // all output
        //P4DIR &= ~0x40;         //P4.6 input for INT for old Bastien PCB1.2
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        // - Magnetometer (MM) pin
        P5SEL = 0x00;    // all IO for the moment (if not, the MM is powered with SPI ! )
        P5OUT = 0x00;    // all L
        P5DIR = 0xFF;    // all IO pin output

        // - Analog pins (Gyroscopes, Sun sensors and temperature sensor)
        P6SEL = 0xFF;    // all peripheral : Analog input
```

```
        //--------------------------------------------------------------------------------
        // Configure Peripherals and init sensors

        //- Timer A for time stamp and clock
        TACCR0 = (T_TIMERA0*(unsigned long)fmclk)>>3;      // Timer period : an interrupt each T_TIMERA0
        TACCTL0 = CCIE;                                                                    // CCR0 interrupt
enabled
        TACTL = TASSEL_2 + MC_1 + ID_3;                                      // SMCLK, up mode, divider 8 //
start timer

        //- ADC12 for Sun sensors, gyroscopes and temperature sensor (and MT current measure?)
        // -- Define conversion sequences
        ADC12MCTL0 = SREF_1+INCH_0;               // ref+=Vref+, channel = A0, GYR+X ; gyroscopes
        ADC12MCTL1 = SREF_1+INCH_1;               // ref+=Vref+, channel = A1, GYR+Y
        ADC12MCTL2 = SREF_1+INCH_2+EOS;   // ref+=Vref+, channel = A2, GYR+Z

        ADC12MCTL3 = SREF_1+INCH_3;                       // ref+=Vref+, channel = A3, SS#_s1 ; sun sensors
        ADC12MCTL4 = SREF_1+INCH_4;                       // ref+=Vref+, channel = A4, SS#_r1
        ADC12MCTL5 = SREF_1+INCH_5;                       // ref+=Vref+, channel = A5, SS#_s2
        ADC12MCTL6 = SREF_1+INCH_6+EOS;           // ref+=Vref+, channel = A6, SS#_r2

        //ADC12MCTL7
        ADC12MCTL8 = SREF_1+INCH_3;                       // ref+=Vref+, channel = A3, Temp  ;
external_temperatures
        ADC12MCTL9 = SREF_1+INCH_4;                       // ref+=Vref+, channel = A4, CB_temp
        ADC12MCTL10 = SREF_1+INCH_10+EOS; // ref+=Vref+, channel = int_temp, internal_temperature

        //ADC12MCTL11 = SREF_1+INCH_3;            // ref+=Vref+, channel = A3, I_MT-X ; magnetotorquer current
        //ADC12MCTL12 = SREF_1+INCH_4;            // ref+=Vref+, channel = A4, I_MT+Y
        //ADC12MCTL13 = SREF_1+INCH_5+EOS;        // ref+=Vref+, channel = A5, I_MT-Z

        // -- Switch on ADC and internal reference
        ADC12CTL0 = ADC12ON+REFON+REF2_5V+SHT0_8+SHT1_8+MSC;       // ADC on,Vref+ on, Vref+=2.5V,

                // Sampling_time=256/fmclk=40us for all !!!!!!!!!

                // multiple sample as soon as possible
        ADC12CTL1 = SHP+ADC12SSEL_3+CONSEQ_1;     // (cstartadr = 0), (Sampling started with Sofware),Sample
pulse mode
                                                                                   // clk div =1, clk
= SMCLK, sequence mode = sequence of channel
    ADC12IE = BIT2+BIT6+BITA; //+BITD;    // enable interrupt for gyro, SS and temp measures //magnetometer
current
        wait_microsecond(20000); // wait ADC reference voltage stabilization

        // - Magnetometer
        init_magnetometer_and_SPI();

        // - Magnetotorquers
        init_magnetotorquers_and_timerB();

        //--------------------------------------------------------------------------------
        // Enable general interrupts (set GIE bit)
        _EINT();

//      init_flash();            // init flash registers
//      launch_the_program_only_once((char *)INFO_MEM_ADR);

        // Enable sensors
        enable_magnetometer(1);
        enable_gyroscopes(1);
        enable_sunsensors(1);
        enable_ext_ADCSBtemp_sens(1);


        // Enable actuators
        enable_magnetotorquers(1);


        P1OUT |= BIT0; // enable multiplexers (will be removed in the new adcs board)
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        //--------------------------------------------------------------------------------
        // MAIN LOOP
        //--------------------------------------------------------------------------------
        while(1) //
        {
                if( !(ADCSTime%((1000*T_MEASURE)/T_TIMERA0)) )     // F_MEASURE
                //if( !(ADCSTime%(1000/(F_MEASURE*T_TIMERA0))) )   // F_MEASURE
                {
//                      DCOCTL=dco<<5;
//                      spi_send_command(0xF5,offset);    // MM X offset
//                      spi_send_command(0xF6,offset);    // MM Y offset
//                      spi_send_command(0xF7,offset);    // MM Z offset
//                      spi_send_command(0xF8,gain);      // MM X gain 0x00 = 18.2dB, 0x10 = 23dB, 0x1F =
27.5dB
```

```
//                      spi_send_command(0xF9,gain);      // MM Y gain
//                      spi_send_command(0xFA,gain);      // MM Z gain

                        start_magnetometer_measure(ADCSTime);     // start MM measure first, because it's the
longest measure!
                        sunsensors measure(ADCSTime);
                        gyroscopes_start_measure(ADCSTime); // must be done after sun sensor
??????????????!!!!!!!!!!!
                        start_temperature_measure(ADCSTime);

                        _LED1TOGGLE;
                }

                if(magnetometer_status() == 2)    // read magnetometer corrected measures
                {
                        unsigned char XYZT[4];
                        TIME timeStamp;

                        read_magnetometer_measure(0,XYZT,&timeStamp);

//                      write_flash_char(XYZT[0],(char*)(ENDOF_PROG_ADR + i ));    //write X
//                      write_flash_char(XYZT[1],(char*)(ENDOF_PROG_ADR + 1*FLASH_TAB_SIZE + i ));  //write Y
//                      write_flash_char(XYZT[2],(char*)(ENDOF_PROG_ADR + 2*FLASH_TAB_SIZE + i ));  //write Z
//                      write_flash_char(XYZT[3],(char*)(ENDOF_PROG_ADR + 3*FLASH_TAB_SIZE + i ));  //write T
//                      write_flash_char(temperature,(char*)(ENDOF_PROG_ADR + 4*FLASH_TAB_SIZE + i ));
        //write Temp
//                      write_flash_int(i,(int*)(ENDOF_PROG_ADR + 5*FLASH_TAB_SIZE + 2*i ));        //write
meas number
//
//                      i++;
//                      if(i==FLASH_TAB_SIZE)
//                              while(1) // stuck the program here ! Because end of free memory has been
reached
//                              {
//                                      _LED1OFF;
//                                      LPM0;
//                              }
                }

                if(gyroscopes_status() == 2)      // read gyroscopes corrected measures
                {
                        int gyrXYZ_[3];
                        TIME timeStamp1;

                        read_gyroscopes_measure(0, gyrXYZ_,&timeStamp1,    ADCSBoard_temp,CB_temp);
                }

                if(sunsensors_status() == 2)      // read sunsensors corrected measures
                {
                        int ss_angle[6][2];
                        TIME timeStamp2;
                        read_sunsensors_measure(0, ss_angle, &timeStamp2);
                }

                if(temperature_status() == 2)     // read temperature corrected measures
                {
                        char T_int_;
                        TIME timeStamp3;

                        read_temperature_measure(0, &ADCSBoard_temp, &CB_temp, &T_int_, &timeStamp3);
                        ADCSTime_correct_temp_drift(ADCSBoard_temp);
                }

                if( !(ADCSTime%(1000/(4*T_TIMERA0))) )     //4Hz, TO TEST MAGNETOTORQUERS, must be removed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                {
                        set_PWM_magnetotorquerX(pwmX);    // the PWM rate is incremented
                        set PWM magnetotorquerY(pwmX);
                        set PWM magnetotorquerZ(pwmX);
                        pwmX +=1;
                        if (pwmX > 100) pwmX = 0;
                }


                LPM0;    // Enter low power mode
        }
}


//-------------------------------------------------------------------------------
// Compute DCO Clock freqency temperature drift
//-------------------------------------------------------------------------------
void ADCSTime_correct_temp_drift(char temperature) // fucntion to be verified
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
{
        fmclk = FMCLK0 + (((long)(temperature-TEMP0))*FMCLK_T_DRIFT*FMCLK0)/1000000;
```

```
        TACCR0 = (T_TIMERA0*(unsigned long)fmclk)>>3;      // Set TimerA0 compare, an interrupt each T_TIMERA0

}


//-------------------------------------------------------------------------------
// Timer A0 interrupt service routine
//-------------------------------------------------------------------------------
__interrupt void Timer_A_int (void);
TIMERA0_ISR(Timer_A_int)
__interrupt void Timer_A_int (void)
{
        ADCSTime++;
        LPM0_EXIT;       // exit low power mode --> main loop
}

//-------------------------------------------------------------------------------
// Analog to digital ADC12 interrupt service routine
//-------------------------------------------------------------------------------
__interrupt void adc_int (void);
ADC12_ISR(adc_int)
__interrupt void adc_int (void)
{
        //look for interrupt source
        if(ADC12IV == 0x0A)       // gyroscopes
                gyroscopes_interrupt();
        else if(ADC12IV == 0x12) // sun sensors
        {
                sunsensors_interrupt();
                LPM0_EXIT;       // exit low power mode --> return in "sunsensors_measure"
        }
        else if(ADC12IV == 0x1A) // temperatures
                temperature_interrupt();

        //else if(ADC12IV == 0x20)        // magnetotorquer current
}
//*****************************************************************************
/*
* File : adcs_general_parameters_and_init_msp.h
* Created by Hervé Péter-Contesse
* April 2007
*
* Header file for use of the ADCS microcontroller (SwissCube project).
* It contains all general and global paramters wich can be device specific --> have to be measured !!!
*/
//-------------------------------------------------------------------------------

#ifndef ADCS_GENERAL_PARAM_H_
#define ADCS_GENERAL_PARAM_H_

//-------------------------------------------------------------------------------
// TIMING PARAMETERS
#define FMCLK0   6500     // [kHz] !!!!!!! nominal MCLK clock frequency DEVICE SPECIFIC, NEED TO BE MEASURED
!!!!!!!!
                                        // max frequency 7.35MHz@3.3V !!!
#define TEMP0    25               // [°C] temperature with which FMCLK0 has been measured
#define  FMCLK_T_DRIFT (-1000)     // [ppm/°C]      MCLK temperature drift
extern unsigned short int fmclk;  // actual MCLK clock frequency [kHz], to be adapted with temperature

#define CYCLE    10               // number of MCLK period per while/for loop turn (measured)
void wait_microsecond(unsigned int us);    // wait function for short wait (>2us and < T_TIMERA0)


#define T_TIMERA0        50  // [ms] TIMER A0 interrupt perdiod/frequency --> used as ACDS clock
                                        // !!! must be smaller than 70ms, because of overflow due to
temperature drift !!!

//#define F_MEASURE      2        // [Hz] Measurement frequency/rate ( >=1 !!)
#define T_MEASURE        1        // [s] Measurement frequency/rate  (>=1 !!)

typedef unsigned long TIME;       // deffine the TIME type for use in every files.
extern TIME ADCSTime;     // time for ADCS board, time counter, incremented each T_TIMERA0
                                        // --> unit = [T_TIMERA0]

//-------------------------------------------------------------------------------

#define N_MEASURE        10       // Number of stored measures for all sensors

//-------------------------------------------------------------------------------
// ADRESSES FOR FLASH WRITE
// used for the measures only
// ONLY VALID FOR MSP430F169 AND DEPENDS ON THE PROGRAM SIZE TOO !

#define INFO_MEM_ADR     (unsigned int)0x1000
#define ENDOF_PROG_ADR   (unsigned int)0x2400      // end of program adress
#define ENDOF_MEM_ADR    (unsigned int)0xFE0F      // maximum adress which can be written
```

```
#define FLASH_TAB_SIZE    (unsigned int)8336               // table size


//-------------------------------------------------------------------------------
// DEBUG LED
#define _LED1ON           P2OUT|=0x10
#define _LED1OFF          P2OUT&=~0x10
#define _LED1TOGGLE P2OUT^=0x10

// other
unsigned int adc2volt(unsigned int adc); //convert in 1e-4 volt ; must be supressed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!


#endif /*ADCS_GENERAL_PARAM_H_*/

//*******************************************************************************
/*
* File : adcs_general_parameters_and_functions.c.h
* Created by Hervé Péter-Contesse
* April 2007
*
* Source file for use of the ADCS microcontroller (SwissCube project).
* It contains all general and global parameters wich can be device specific --> have to be measured !!!
*/
//-------------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_general_parameters_and_functions.h"


void wait_microsecond(unsigned int us)     // wait function for short wait (>2us and < T_TIMERA0)
{
        unsigned int i,limit;
        limit = ((long)us*FMCLK0)/CYCLE/1000;
        for(i=0;i<limit;i++);
}

unsigned int adc2volt(unsigned int adc)
{
        return ((unsigned long)adc*(unsigned long)25000)>>12;
}
```

E.3.3 Magnetometer

```
//*******************************************************************************
/*
* File : adcs_magnetometer.h
* Created by Hervé Péter-Contesse
* April 2007
*
* Header file for use of the ADCS magnetometer (SwissCube project).
* This file provide all functions to initialize and read the magnetometers sensors.
* It uses an SPI interface (USART1), IO port and other interrupts.
*/
//-------------------------------------------------------------------------------
/*
 * Pin configuration :
 * E2 =  P1.6
 * INT =         P2.3
 * RSTN =        P4.7
 * CS =  P5.0
 * SI =  P5.1
 * SO =  P5.2
 * SK =  P5.3
 * MCLK =        P5.5
 */
//-------------------------------------------------------------------------------

#ifndef ADCS_MAGNETOMETER_H_
#define ADCS_MAGNETOMETER_H_

#include "adcs_general_parameters_and_functions.h"


void init_magnetometer_and_SPI(void); // initialisations
void enable_magnetometer(char on_off); // 1=enable, 0=disable
void start_magnetometer_measure(TIME time_stamp_); // start measure : Be carefull, measurement time is about
170ms

int read_magnetometer_measure(unsigned char index, unsigned char XYZTmm_[4], TIME *time_stamp_);
        // index = 0 -> last measure (most recent), index = 1 -> previous measure, ...
```

```
        // be carefull with index ! it must be lower than N_MEASURE-1 ! return -1 if index>N_MEASURE-1, 0 else

int magnetometer_status(void);    // return 1 if measuring, 2 if a new measure is available and 0 else.
                                                                    // a call of function
read_magnetometer_measure set status to 0 if index=0.


void spi_send_command(char adress,char value); // must be removed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

#endif /*ADCS_MAGNETOMETER_H_*/


//*****************************************************************************
/*
* File : adcs_magnetometer.c
* Created by Hervé Péter-Contesse
* April 2007
*
* Source file for use of the ADCS magnetometer (SwissCube project).
* This file provide all functions to initialize and read the magnetometers sensors.
* It uses an SPI interface (USART1), IO port and other interrupts.
*/
//-----------------------------------------------------------------------------
/*
 * Pin configuration :
 * E2 =  P1.6
 * INT =         P2.3
 * RSTN =        P4.7
 * CS =  P5.0
 * SI =  P5.1
 * SO =  P5.2
 * SK =  P5.3
 * MCLK =        P5.5
 */
//-----------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_magnetometer.h"

//-----------------------------------------------------------------------------
// Global variables declaration
//-----------------------------------------------------------------------------
// measure variables
unsigned char XYZTmm[4][N_MEASURE];               // measures circular buffer
TIME time_stamp[N_MEASURE];               // time stamp for each measure
char measi_mm = 0;                                      // measure index
char measf_mm = 0;                                // flag, indicates if the MM is curretly measuring

// SPI variables
#define RXBUFSIZE 8               //must be a 2 multiple
unsigned char rxbuf[RXBUFSIZE];   // receive buffer
char rxbufi = 0;          // rxbuf index


//-----------------------------------------------------------------------------
// Internal functions declaration
//-----------------------------------------------------------------------------

void init_mm_parameters(void);
void spi_ask_meas_res(char adress);
//void spi_send_command(char adress,char value);

//-----------------------------------------------------------------------------
// External functions definition
//-----------------------------------------------------------------------------

void init_magnetometer_and_SPI(void) // initialisations
{
        // -- SPI interface (USART1), 3-pin mode
        U1CTL = CHAR + SYNC + MM + SWRST; // 8-bit + SPI + Master + reset
        U1TCTL = SSEL1 + STC;             // (L when idle) + SMCLK + 3-wire
        U1BR0 = 0x02; U1BR1 = 0x00;       // SPICLK = SMCLK/2 (maximum clock speed !!!)
        U1MCTL = 0x00;   // no modulator used in SPI mode
        ME2 |= USPIE1;      // Module enable
        U1CTL &= ~SWRST;          // SPI enable (clear reset bit)
        IE2 |= URXIE1;      // RX and TX interrupt enable

        // -- INT pin P2.3 for measure complete
        P2IES = 0;       // interrupt on rising edge
        P2IFG = 0;       // clear interrupt flag
        P2IE = BIT3;     // enable interrupt
}

void enable_magnetometer(char on_off) // enable or disable --> must do some init!
{
        if(on_off)       // ON
```

```
        {
                // all pins, included SPI pins, are configured only now, because they can source power into MM
                // (even if E2 = 0!)
                P1OUT |= 0x40;   // Power on
                P4OUT |= 0x80;   // set RSTN (not MM reset)
                P5SEL = 0x0E;    // config SPI IO/peripherals (not MCLK P5.5), CS=IO

                init_mm_parameters();    // init
        }
        else if(!on_off) // OFF
        {
                P5SEL = 0x00;    // Stop MCLK out (if not stopped yet) and SPI
                P4OUT &= ~0x80;  // clear RSTN
                P1OUT &= ~0x40;  // Power off
        }
}

void start_magnetometer_measure(TIME time_stamp_)  // measurement time is about 170ms for MCLK=5MHz
{
        P5SEL |= 0x20;   // Start MCLK out
        spi_send_command(0xF3,0x00);      // send measure command

        time_stamp[measi_mm] = time_stamp_;       // store time stamp
        measf_mm = 1; // measuring
}

int read_magnetometer_measure(unsigned char index, unsigned char XYZTmm_[4], TIME *time_stamp_)
{
        char i;

        if(index > N_MEASURE -1) return -1;       // error with index

        i=measi_mm-1-(char)index;
        if(i<0) i += N_MEASURE;

        XYZTmm_[0]=XYZTmm[0][i]; XYZTmm_[1]=XYZTmm[1][i];
        XYZTmm_[2]=XYZTmm[2][i]; XYZTmm_[3]=XYZTmm[3][i];
        *time_stamp_ = time_stamp[i];

        if(!index)       // if last measure has been read
                measf_mm=0; // no new measure available

        return 0;
}

int magnetometer_status(void)
{
        return measf_mm;
}

//-------------------------------------------------------------------------------
// Internal functions definition
//-------------------------------------------------------------------------------

// init
void init_mm_parameters(void)
{
//      int i,j;         // clear table !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
//      for(i=0;i<4;i++)
//              for(j=0;j<N_MEASURE;j++)
//                      XYZTmm[i][j] = 0x00;

        spi_send_command(0xF4,0xF0);      // MM MCLK setting, int. time 20ms, rectangular wave (0x80 = 10ms,
0xF0 = 20ms)

        spi_send_command(0xF5,0x00);      // MM X offset
        spi_send_command(0xF6,0x00);      // MM Y offset
        spi_send_command(0xF7,0x00);      // MM Z offset

        spi_send_command(0xF8,0x0F);      // MM X gain 0x00 = 18.2dB, 0x10 = 23dB, 0x1F = 27.5dB
        spi_send_command(0xF9,0x10);      // MM Y gain
        spi_send_command(0xFA,0x11);      // MM Z gain
//      spi_send_command(0xF8,0x00);      // MM X gain 0x00 = 18.2dB, 0x10 = 23dB, 0x1F = 27.5dB
//      spi_send_command(0xF9,0x00);      // MM Y gain
//      spi_send_command(0xFA,0x00);      // MM Z gain
}

// read measures result
void spi_ask_meas_res(char adress)
{
        int saverxbufi;

        P5OUT |= 0x01;   // CS H
        while((IFG2 & UTXIFG1) == 0);     // wait until U1TXBUF is free
        U1TXBUF = adress;         // send command
        while((IFG2 & UTXIFG1) == 0);     // wait until U1TXBUF is free
```

```
        rxbufi = 0;
        U1TXBUF = 0x00;           // send nothing to receive data
        while((IFG2 & UTXIFG1) == 0);     // wait until U1TXBUF is free
        U1TXBUF = 0x00;           // send nothing to receive data

        saverxbufi = rxbufi;
        while(saverxbufi == rxbufi) ;     // wait until send is finished // a maximum time should be allowed
here !!!!!!!!!!!!!!!!!!!!!!!!
        P5OUT &= ~0x01;                   // CS L
}

// send a general command to the MM
void spi_send_command(char adress,char value)
{
        int saverxbufi;

        while(measf_mm); // check if mesure is complete
        P5OUT |= 0x01;            // CS H
        while((IFG2 & UTXIFG1) == 0);     // wait until U1TXBUF is free
        U1TXBUF = adress;         // 1st Byte
        while((IFG2 & UTXIFG1) == 0);
        U1TXBUF = value; // 2nd Byte

        saverxbufi = rxbufi;
        while(saverxbufi == rxbufi);      // wait until send is finished // a maximum time should be allowed
here !!!!!!!!!!!!!!!!!!!!!!!!
        P5OUT &= ~0x01;  // CS L
}

//-------------------------------------------------------------------------------
// Interrupts
//-------------------------------------------------------------------------------

// - Measure complete interrupt
__interrupt void measure_complete_int (void);      // Usually not a good idea to have long ISR, but in
PORT2_ISR(measure_complete_int)                                  // this case the interrupt is not called
often
__interrupt void measure_complete_int (void)
{
        P2IFG &= ~BIT3;  // clear interrupt flag, must be done before entering other interrupts !!!
        _EINT();         // enable interrupts, for SPI !
        P5SEL &= ~0x20;  // Stop MCLK out

        spi ask meas res(0xF0);   // ask for X and Y measures
        XYZTmm[0][measi_mm] = rxbuf[1]; XYZTmm[1][measi_mm] = rxbuf[2]; // store measurements in table

        spi_ask_meas_res(0xF1);   // ask for Z and Temperature measures
        XYZTmm[2][measi_mm] = rxbuf[1]; XYZTmm[3][measi_mm] = rxbuf[2]; // store measurements in table

        measi_mm = (measi_mm+1)%N_MEASURE;
        measf_mm=2;               // new measure available
}

// - SPI Bytes receive interrupt
// this interrupt is called each time a byte has been sent, because a byte
// is received in the same time
__interrupt void SPI1_rx (void);
USART1RX_ISR(SPI1_rx)
__interrupt void SPI1_rx (void)
{
        rxbuf[rxbufi] = U1RXBUF;
        rxbufi = (++rxbufi)&(RXBUFSIZE-1);        //circular buffer
}
```

E.3.4 Magnetotorquers

```
//*******************************************************************************
/*
* File : adcs_magnetotorquers.h
* Created by Hervé Péter-Contesse
* May 2007
*
* Header file for use on the ADCS microcontroller (SwissCube project) to control
* the magnetotorquers.
* This file provide all functions to initialize and change the PWM settings for
* the 3 magnetotorquers.
*/
//-------------------------------------------------------------------------------
/*
 * Pin configuration :
 *  -- MT1-X
 *       E4               P2.0
 *       PWM1+    P4.1
 *       PWM1-    P4.2
```

```
 *  -- MT2+Y
 *       E5                P2.1     --> will be supressed !!!
 *       PWM2+    P4.3
 *       PWM2-    P4.4
 *  -- MT3-Z
 *       E6                P2.2     --> will be supressed !!!
 *       PWM3+    P4.5
 *       PWM3-    P4.6
 */
//-------------------------------------------------------------------------------

#ifndef MAGNETOTORQUERS_H_
#define MAGNETOTORQUERS_H_

void init_magnetotorquers_and_timerB(void);        // initializations
void enable_magnetotorquers(char on_off); // 1=enable, 0=disable

int set_PWM_magnetotorquerXYZ(char pwmX, char pwmY, char pwmZ);     // set PWM ratio for all magnetotorquers
                                                                    // values are between [-
100,100]% ; negatives values simply invert
                                                                    // the current in the
magnetotorquers ; 0 = no torque

int set_PWM_magnetotorquerX(char pwmX);    // set PWM ratio for magnetotorquer X only
int set_PWM_magnetotorquerY(char pwmY);
int set_PWM_magnetotorquerZ(char pwmZ);

void correct_magnetotorquers_temperature_drift(char tempX, char tempY, char tempZ);


#endif /*MAGNETOTORQUERS_H_*/


//*******************************************************************************
/*
* File : adcs_magnetotorquers.c
* Created by Hervé Péter-Contesse
* May 2007
*
* Source file for use on the ADCS microcontroller (SwissCube project) to control
* the magnetotorquers.
* This file provide all functions to initialize and change the PWM settings for
* the 3 magnetotorquers.
*/
//-------------------------------------------------------------------------------
/*
 * Pin configuration :
 *  -- MT1-X
 *       E4                P2.0
 *       PWM1+    P4.1
 *       PWM1-    P4.2
 *  -- MT2+Y
 *       E5                P2.1     --> will be supressed !!!
 *       PWM2+    P4.3
 *       PWM2-    P4.4
 *  -- MT3-Z
 *       E6                P2.2     --> will be supressed !!!
 *       PWM3+    P4.5
 *       PWM3-    P4.6
 */
//-------------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_general_parameters_and_functions.h"
#include "adcs_magnetotorquers.h"

#define FPWM0 32000      // [Hz] PWM frequency/TimerB frequency,   !!!!!!!!!!!!!!!!!!! mettre en kHz
                                    // Max PWM freq = FMCLK0/2/100 = 32.5kHz for FMCLK0 =6.5MHz
!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                                    // Must be >500Hz (because of the MT cut frequency)
                                    // It is better if it's >>1.1kHz because it's the max frame mecanical
frequency
                                    // the frequency drift due to temperature is not corrected here,
because it isn't important

#define OUT_PLUS          (OUTMOD_2 + CLLD_2)
#define OUT_MINUS         CLLD_2

//-------------------------------------------------------------------------------
// Global variables declaration
//-------------------------------------------------------------------------------

unsigned long fpwm = FPWM0;       // must be supressed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
char pwmX_gl = 0, pwmY_gl = 0, pwmZ_gl = 0;

//-------------------------------------------------------------------------------
```

```
// Internal functions declaration
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// External functions definition
//---------------------------------------------------------------------------

void init_magnetotorquers_and_timerB(void)// initialization
{
        TBCCTL1 = TBCCTL2 = TBCCTL3 = TBCCTL4 = TBCCTL5 = TBCCTL6 = 0 ;
        TBCCR0 = (((unsigned long)FMCLK0*1000)/FPWM0)>>1;  // timer frequency / PWM frequency
        !!!!!!!!!!!!!!!! changer en kHz
        TBCTL = TBSSEL_2 + ID_0 + MC_3;   // (16-bit timer), SMCLK, divider 1,
                                                                                        // up/down
mode --> div tot = 2,  start timer
                                                                                        // max PWM
freq = FMCLK0/2/100
}
void enable_magnetotorquers(char on_off) // enable 1 or disable 0
{
        if(on_off)       // MT ON
        {
                P2OUT |= BIT0 + BIT1 + BIT2;      // Power on H-bridge
                //TBCCTL1 = TBCCTL2 = CLLD1;      //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        }
        if(!on_off)      // MT OFF
        {
                // out mode 0 = OUT, OUT = Low for mode 0 --> stop PWM
                TBCCTL1 = TBCCTL2 = TBCCTL3 = TBCCTL4 = TBCCTL5 = TBCCTL6 = 0;
        //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
                P2OUT &= ~(BIT0 + BIT1 + BIT2);   // Power off H-bridge
        }
}

int set_PWM_magnetotorquerXYZ(char pwmX, char pwmY, char pwmZ)
{
        return 0;
}

int set_PWM_magnetotorquerX(char pwmX)     // set PWM ratio for magnetotorquer X only
{
        if(pwmX > 100 || pwmX < -100)
                return -1;
        else if(pwmX>=0) // set current direction
        {
                // for simple PWM
                TBCCTL1 = OUTMOD_2; // out mode 2 toggle/reset, output Low for mode 0
                TBCCTL2 = 0;      // out mode 0, output Low
                TBCCR1 = ((unsigned long)TBCCR0*pwmX)/100;// set ratio     // put in correct temp !!!!!!!

                // for double PWM
//              TBCCTL2 = OUTMOD_6;
//              TBCCR2 = TBCCR1;
        }
        else
        {
                TBCCTL1 = 0;      // out mode 0, output Low
                TBCCTL2 = OUTMOD_2; // out mode 2 toggle/reset, output Low for mode 0
                TBCCR2 = ((unsigned long)TBCCR0*(-pwmX))/100;      // set ratio     // put in correct temp
!!!!!!!
        }

        TBCCR0 = (((unsigned long)FMCLK0*1000)/fpwm)>>1;   // to change the frequence manually, must be
supressed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        return 0;
}
int set_PWM_magnetotorquerY(char pwmY)
{
        if(pwmY > 100 || pwmY < -100)
                return -1;
        else if(pwmY>=0) // set current direction
        {
                TBCCTL3 = OUTMOD 2; // out mode 2 toggle/reset, output Low for mode 0
                TBCCTL4 = 0;      // out mode 0, output Low
                TBCCR3 = ((unsigned long)TBCCR0*pwmY)/100;// set ratio     // put in correct temp !!!!!!!
        }
        else
        {
                TBCCTL3 = 0;      // out mode 0, output Low
                TBCCTL4 = OUTMOD 2; // out mode 2 toggle/reset, output Low for mode 0
                TBCCR4 = ((unsigned long)TBCCR0*(-pwmY))/100;      // set ratio     // put in correct temp
!!!!!!!
        }
```

```
        TBCCR0 = (((unsigned long)FMCLK0*1000)/fpwm)>>1;   // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        return 0;
}

int set_PWM_magnetotorquerZ(char pwmZ)
{
        if(pwmZ > 100 || pwmZ < -100)
                return -1;
        else if(pwmZ>=0) // set current direction
        {
                TBCCTL5 = OUTMOD_2; // out mode 2 toggle/reset, output Low for mode 0
                TBCCTL6 = 0;      // out mode 0, output Low
                TBCCR5 = ((unsigned long)TBCCR0*pwmZ)/100;// set ratio     // put in correct temp !!!!!!!
        }
        else
        {
                TBCCTL5 = 0;      // out mode 0, output Low
                TBCCTL6 = OUTMOD_2; // out mode 2 toggle/reset, output Low for mode 0
                TBCCR6 = ((unsigned long)TBCCR0*(-pwmZ))/100;      // set ratio     // put in correct temp
!!!!!!!
        }

        TBCCR0 = (((unsigned long)FMCLK0*1000)/fpwm)>>1;   // !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

        return 0;
}


void correct_magnetotorquers_temperature_drift(char tempX, char tempY, char tempZ)
{

}

//---------------------------------------------------------------------------
// Internal functions definition
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Interrupts
//---------------------------------------------------------------------------
// NO INTERRUPT NEEDED !!!
```

E.3.5 Gyroscopes

```
//*****************************************************************************
/*
* File : adcs_gyroscopes.h
* Created by Hervé Péter-Contesse
* June 2007
*
* Header file for use on the ADCS microcontroller (SwissCube project) to do
* measurement with the gyroscopes.
* This file provide all functions to initialize the 12-bit ADC of the MSP and to
* use it to measure the 3 analog signals from the 2 IDG 1000??? gyroscopes
*/
//---------------------------------------------------------------------------
 /*
 * Pin configuration :
 * - gyroscopes
 *  E1           P1.5
 *
 * - Analog Input
 *  GYR+X        P6.0/A0 !!!!!!!!!!!!!
 *  GYR+Y        P6.1/A1 !!!!!!!!!!!!!
 *  GYR+Z        P6.2/A2 !!!!!!!!!!!!!
 */
//---------------------------------------------------------------------------

#ifndef ADCS_GYROSCOPES_H_
#define ADCS_GYROSCOPES_H_


void enable_gyroscopes(char on_off);      // 1=enable, 0=disable,
                                                                        // !!!!!! be carefull, the
gyroscope startup time is enormous !!!!!!
                                                                        // !!!!!! about 400-500ms
!!!!!!
                                                                        // !!!!!! be sure this
waiting time is reached before starting measure!!!!!!
void gyroscopes_start_measure(TIME time_stamp_);   // start the ADC conversion of the gyroscopes signals
```

```
int read_gyroscopes_measure(unsigned char index,                                    // read the corrected
gyroscopes
                                                    int gyrXYZ_[3], TIME *time_stamp_,
        // values in °/s,
                                                    char ADCS_temp_XY, char CB_temp_Z);       // actual
temperatures

void gyroscopes_interrupt(void);  // function called in the interrupt ADC service routine to catch
                                                                        // the gyroscopes measures
int gyroscopes_status(void);      // return 1 if measuring, 2 if a new measure is available and 0 else.
                                                        // a call of function
read_gyroscopes_measure set status to 0 if index=0.


#endif /*ADCS_GYROSCOPES_H_*/


//****************************************************************************
/*
* File : adcs gyroscopes.h
* Created by Hervé Péter-Contesse
* June 2007
*
* Header file for use on the ADCS microcontroller (SwissCube project) to do
* measurement with the gyroscopes.
* This file provide all functions to initialize the 12-bit ADC of the MSP and to
* use it to measure the 3 analog signals from the 2 IDG 1000??? gyroscopes
*/
//----------------------------------------------------------------------------
 /*
 * Pin configuration :
 * - gyroscopes
 *  E1           P1.5
 *
 * - Analog Input
 *  GYR+X        P6.0/A0 !!!!!!!!!!!!!
 *  GYR+Y        P6.1/A1 !!!!!!!!!!!!!
 *  GYR+Z        P6.2/A2 !!!!!!!!!!!!!
 */
//----------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_general_parameters_and_functions.h"
#include "adcs_gyroscopes.h"


//----------------------------------------------------------------------------
// Global variables declaration
//----------------------------------------------------------------------------
unsigned int gyrXYZ[3][N_MEASURE];        // measures circular buffer !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

TIME time stamp gyr[N MEASURE];           // time stamp for each measure
char measi_gyr = 0;                                     // measure index
char measf_gyr = 0;                               // flag, indicates if the MM is curretly measuring


unsigned int a_gyr[3][N_MEASURE]; // must be supressed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

//----------------------------------------------------------------------------
// Internal functions declaration
//----------------------------------------------------------------------------


//----------------------------------------------------------------------------
// External functions definition
//----------------------------------------------------------------------------
void enable_gyroscopes(char on_off) // enable or disable, not init needed because the ADC is configured
                                                                        // when the measure is started
{
        if(on_off)                          // ON
                P1OUT |= BIT5;
        else if(!on_off) // OFF
                P1OUT &= ~BIT5;
}

void gyroscopes_start_measure(TIME time_stamp_)
{
        time stamp gyr[measi gyr] = time_stamp_;  // store time stamp
        measf_gyr = 1; // measuring

        while(ADC12CTL1 & BUSY); // wait in case ADC in doing another convertion

        ADC12CTL1 = CSTARTADD_0+SHP+ADC12SSEL_3+CONSEQ_1;  // cstartadr = 0, (Sampling started with
Sofware),Sample pulse mode
                                                                                // clk div =1, clk
= SMCLK, sequence mode = sequence of channel
```

```
        ADC12CTL0 |= ENC+ADC12SC; // start conversion !
        ADC12CTL0 &= ~ENC;        // clear enc bit (sequence conversion continue)
}

int read_gyroscopes_measure(unsigned char index, int gyrXYZ_[3], TIME *time_stamp_,
                                                    char ADCS_temp_XY, char CB_temp_Z)        // must
correct temperature drift !!!!!!!!!!!!
{
        char i;

        if(index > N_MEASURE -1) return -1;       // error with index

        i=measi_gyr-1-(char)index;        // check borders
        if(i<0) i += N_MEASURE;

        // return measures, must do convertions bit-->°/s here !!!!!!!!!!!!!!!!!!!!
        gyrXYZ [0] = gyrXYZ[0][i]; gyrXYZ_[1] = gyrXYZ[1][i]; gyrXYZ_[2] = gyrXYZ[2][i];
        *time_stamp_ = time_stamp_gyr[i];

        if(!index)       // if last measure has been read
                measf_gyr=0; // no new measure available

        return 0;
}

void gyroscopes_interrupt(void)
{
        gyrXYZ[0][measi_gyr] = ADC12MEM0; // signal1
        gyrXYZ[1][measi_gyr] = ADC12MEM1; // ref1
        gyrXYZ[2][measi_gyr] = ADC12MEM2; // signal2

        a_gyr[0][measi_gyr] = adc2volt(gyrXYZ[0][measi_gyr]);      // must be supressed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        a_gyr[1][measi_gyr] = adc2volt(gyrXYZ[1][measi_gyr]);      // !!!
        a_gyr[2][measi_gyr] = adc2volt(gyrXYZ[2][measi_gyr]);      // !!!

        measi_gyr = (measi_gyr+1)%N_MEASURE;      // increment measure index
        measf_gyr=2;     // new gyroscope measure available
}

int gyroscopes_status(void)
{
        return measf_gyr;
}

//---------------------------------------------------------------------------
// Internal functions definition
//---------------------------------------------------------------------------


//---------------------------------------------------------------------------
// Interrupts
//---------------------------------------------------------------------------
//NO INTERRUPT HERE
```

E.3.6 Sun sensors

```
//***************************************************************************
/*
* File : adcs sun sensors.h
* Created by Hervé Péter-Contesse
* June 2007
*
* Header file for use on the ADCS microcontroller (SwissCube project) to do
* measurement with the sun sensors.
* This file provide all functions to initialize the 12-bit ADC of the MSP and to
* use it to measure the 24 analog signals from the 6 DTU sun sensors
*/
//---------------------------------------------------------------------------
/*
 * Pin configuration :
 *
 * - Multiplexers
 *  EN          P1.0 : enable Multiplexers --> will be supressed !!
 *  A0          P1.1 : change gate
 *  A1          P1.2
 *  A2          P1.3
 *
 * - Sun sensors
 *  E3          P1.7
 *
 * - Analog Input
 *  SS#_s1      P6.3/A3  : sunsensors 1-6 signal 1
 *  SS#_r1      P6.4/A4 : sunsensors 1-6 reference 1
```

```
 *  SS#_s2       P6.5/A5
 *  SS#_r2       P6.6/A6
 *
 */
//---------------------------------------------------------------------------

#ifndef ADCS_SUN_SENSORS_H_
#define ADCS_SUN_SENSORS_H_

void enable_sunsensors(char on_off);      // 1=enable, 0=disable,
                                                                    // !!!!!! be carefull with
startup time !!!!!!

void sunsensors_measure(TIME time_stamp_);        // make all ADC conversion to measure the gyroscopes signals
                                                                    // and changes the
multiplexers outputs.
                                                                    //!!!! This
function does not start the measure but DO the     !!!!
                                                                    //!!!! measures so
it is quite long to execute                              !!!!

int read_sunsensors_measure(unsigned char index, int ss_angle[6][2], TIME *time_stamp_); // read the
                                                                    // corrected sunsensors angle in °

void sunsensors_interrupt(void);  // function called in the interrupt ADC service routine to catch
                                                                    // the sunsensors measures
int sunsensors_status(void);      // return 1 if measuring, 2 if a new measure is available and 0 else.
                                                                    // a call of function
read_sunsenors_measure set status to 0 if index=0.


#endif /*ADCS_SUN_SENSORS_H_*/


//***************************************************************************
/*
* File : adcs_sun_sensors.c
* Created by Hervé Péter-Contesse
* June 2007
*
* Source file for use on the ADCS microcontroller (SwissCube project) to do
* measurement with the sun sensors.
* This file provide all functions to initialize the 12-bit ADC of the MSP and to
* use it to measure the 24 analog signals from the 6 DTU sun sensors
*/
//---------------------------------------------------------------------------
/*
 * Pin configuration :
 *
 * - Multiplexers
 *  EN           P1.0 : enable Multiplexers --> will be supressed !!
 *  A0           P1.1 : change gate
 *  A1           P1.2
 *  A2           P1.3
 *
 * - Sun sensors
 *  E3           P1.7
 *
 * - Analog Input
 *  SS#_s1       P6.3/A3  : sunsensors 1-6 signal 1
 *  SS#_r1       P6.4/A4 : sunsensors 1-6 reference 1
 *  SS#_s2       P6.5/A5
 *  SS#_r2       P6.6/A6
 *
 */
//---------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_general_parameters_and_functions.h"
#include "adcs_sunsensors.h"


//---------------------------------------------------------------------------
// Global variables declaration
//---------------------------------------------------------------------------
unsigned int ss[6][4][N_MEASURE]; // measures circular buffer
TIME time_stamp_ss[N_MEASURE];    // time stamp for each measure

char measi_ss = 0;                              // measure index
char ss_index = 0;                              // sun sensor index (atual selected ss)
char measf_ss = 0;                              // flag, indicates if the MM is curretly measuring

unsigned int a[6][4][N_MEASURE]; // must be supressed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

//---------------------------------------------------------------------------
// Internal functions declaration
```

```
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// External functions definition
//-----------------------------------------------------------------------------

void enable_sunsensors(char on_off) // enable or disable, not init needed because the ADC is configured
                                                                        // when the measure is started
{
        if(on_off)                      // ON
                P1OUT |= BIT7;
        else if(!on_off) // OFF
                P1OUT &= ~BIT7;
}

void sunsensors_measure(TIME time_stamp_)
{
        unsigned int mux;
        ss_index = 0;

        time_stamp_ss[measi_ss] = time_stamp_;    // store time stamp
        measf_ss = 1; // measuring

        while(ADC12CTL1 & BUSY); // wait in case ADC in doing another convertion

        while(ss_index <6)      // do/wait until all measures are complete for the 6 sun sensors
        {
                int save_index = ss_index;

                mux = ss index <<1;     // change multiplexers inputs : sunsensor 0 to 5
                P1OUT |= mux;
                P1OUT &= mux|0xF1;
                //!! a wait is needed here if there are capacitors after the mux !!!

                ADC12CTL1 = CSTARTADD_3+SHP+ADC12SSEL_3+CONSEQ_1;  // cstartadr = 3, (Sampling started with
Sofware),Sample pulse mode
                                                                                // clk div =1, clk
= SMCLK, sequence mode = sequence of channel
                ADC12CTL0 |= ENC+ADC12SC;// start conversion !

                while(save_index == ss_index) LPM0;     // wait until the 4 measures are complete for 1
sunsensor
        }
        ADC12CTL0 &= ~ENC;      // clear enc bit --> the ADC parameters can be modified

        measi_ss = (measi_ss+1)%N_MEASURE;      // increment measure index
        measf_ss=2;             // new measure available
}

int read_sunsensors_measure(unsigned char index, int ss_angle[6][2], TIME *time_stamp_)
{
        char i,j;

        if(index > N_MEASURE -1) return -1;     // error with index

        i=measi_ss-1-(char)index;// check borders
        if(i<0) i += N_MEASURE;

        for(j=0;j<6;j++)                // read all measures
        {
                ss angle[j][0] = ss[j][0][i]/ss[j][1][i]; // must be completed !!!!!!!!!!!!!!!!!!
                ss_angle[j][1] = ss[j][2][i]/ss[j][3][i];
        }

        *time_stamp_ = time_stamp_ss[i];

        if(!index)      // if last measure has been read
                measf_ss=0; // no new measure available

        return 0;
}

void sunsensors_interrupt(void)
{
        ss[ss_index][0][measi_ss] = ADC12MEM3;    // signal1
        ss[ss_index][1][measi_ss] = ADC12MEM4;    // ref1
        ss[ss_index][2][measi_ss] = ADC12MEM5;    // signal2
        ss[ss_index][3][measi_ss] = ADC12MEM6;    // ref2

        a[ss index][0][measi ss] = adc2volt(ss[ss_index][0][measi_ss]);    // must be supressed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        a[ss index][1][measi ss] = adc2volt(ss[ss index][1][measi ss]); // !!!
        a[ss_index][2][measi_ss] = adc2volt(ss[ss_index][2][measi_ss]);    // !!!
        a[ss_index][3][measi_ss] = adc2volt(ss[ss_index][3][measi_ss]);    // !!!
```

```
        ss_index++;
}

int sunsensors_status(void)
{
        return measf_ss;
}

//-----------------------------------------------------------------------------
// Internal functions definition
//-----------------------------------------------------------------------------


//-----------------------------------------------------------------------------
// Interrupts
//-----------------------------------------------------------------------------
//NO INTERRUPT HERE
```

E.3.7 Temperature sensors

```
//*****************************************************************************
/*
* File : adcs_temperature.h
* Created by Hervé Péter-Contesse
* June 2007
*
* Header file for use on the ADCS microcontroller (SwissCube project) to do
* measurement with the temperature sensors (MSP internal, external LM94022 on
* ADCS board and external LM94022 on connection board)
*/
//-----------------------------------------------------------------------------
 /*
 * Pin configuration :
 * - enable ADCS temperature sensor
 *  E0           P1.4
 *
 * - Analog Input
 *  Temp P6.3/A3  : ADCS board temperature
 *  CB_temp P6.4/A3 : Connection board temperature
 */
//-----------------------------------------------------------------------------

#ifndef ADCS_TEMPERATURE_H_
#define ADCS_TEMPERATURE_H_

void enable_ext_ADCSBtemp_sens(char on_off);       // 1=enable, 0=disable, ADCS board temperature sensor
                                                                                                    // !!!! be
carefull with statrup time !!!!

void start_temperature_measure(TIME time_stamp_);

int read_temperature_measure(unsigned char index,                          // read temperatres measures
                                                        char *T_ext_, char *T_CB_,                  // [°C]
LM94022 ADCS board and connection board
                                                        char *T_int_, TIME *time_stamp_);           // [°C]
internal MSP sensor
void temperature_interrupt(void);
int temperature_status(void);

#endif /*ADCS_TEMPERATURE_H_*/


//*****************************************************************************
/*
* File : adcs_temperature.c
* Created by Hervé Péter-Contesse
* June 2007
*
* Source file for use on the ADCS microcontroller (SwissCube project) to do
* measurement with the temperature sensors (MSP internal and external)
*/
//-----------------------------------------------------------------------------
 /*
 * Pin configuration :
 * - enable ADCS temperature sensor
 *  E0           P1.4
 *
 * - Analog Input
 *  Temp P6.3/A3  : ADCS board temperature
 *  CB_temp P6.4/A3 : Connection board temperature
 */
//-----------------------------------------------------------------------------

#include <msp430x16x.h>
```

```
#include "adcs_general_parameters_and_functions.h"
#include "adcs_temperature.h"


//------------------------------------------------------------------------------
// Global variables declaration
//------------------------------------------------------------------------------
unsigned int T_ext[N_MEASURE];    // measures circular buffer
unsigned int T_CB[N_MEASURE];
unsigned int T_int[N_MEASURE];
TIME time_stamp_T[N_MEASURE];     // time stamp for each measure

char measi_T = 0;                                 // measure index
char measf_T = 0;                                 // flag, indicates if the MM is curretly measuring

unsigned int a_T[3][N_MEASURE]; // must be supressed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
signed int temp_degree[N_MEASURE]; // must be supressed !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!

//------------------------------------------------------------------------------
// Internal functions declaration
//------------------------------------------------------------------------------

char volt2temp_LM94022(int volt); // must be optimized with look-up table !!!!!!!!!!!!!!!!!!!!!!!

//------------------------------------------------------------------------------
// External functions definition
//------------------------------------------------------------------------------

void enable_ext_ADCSBtemp_sens(char on_off)
{
        if(on_off)                          // ON
                P1OUT |= BIT4;
        else if(!on_off) // OFF
                P1OUT &= ~BIT4;
}

void start_temperature_measure(TIME time_stamp_)
{
        time_stamp_T[measi_T] = time_stamp_;      // store time stamp
        measf_T = 1; // measuring

        while(ADC12CTL1 & BUSY); // wait in case ADC in doing another convertion

        P1OUT |= 0x06<<1;         // change multiplexers inputs : temperature ext and CB
        P1OUT &= (0x06<<1)|0xF1;
        //!! a wait is needed here if there are capacitors after the mux !!!

        ADC12CTL1 = CSTARTADD_8+SHP+ADC12SSEL_3+CONSEQ_1;  // cstartadr = 0, (Sampling started with
Sofware),Sample pulse mode
                                                                                    // clk div =1, clk
= SMCLK, sequence mode = sequence of channel
        ADC12CTL0 |= ENC+ADC12SC; // start conversion !
        ADC12CTL0 &= ~ENC;        // clear enc bit (sequence conversion continue)

}

int read_temperature_measure(unsigned char index, char *T_ext_,
                                                        char *T_CB_, char *T_int_, TIME *time_stamp_)
{
        char i;

        if(index > N_MEASURE -1) return -1;       // error with index

        i=measi_T-1-(char)index;  // check borders
        if(i<0) i += N_MEASURE;

        *T_ext_ = temp_degree[i];
        //*T_CB =
        //*T_int_ =

        *time_stamp_ = time_stamp_T[i];

        if(!index)       // if last measure has been read
                measf_T=0; // no new measure available

        return 0;
}

void temperature_interrupt(void)
{
        T_ext[measi_T] = ADC12MEM8;               // external temp
        T_CB[measi_T] = ADC12MEM9;                // connnection board temp
        T_int[measi_T] = ADC12MEM10;      // internal temp

        a_T[0][measi_T] = adc2volt(T_ext[measi_T]);       // must be supressed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

```
        temp_degree[measi_T] = volt2temp_LM94022(a_T[0][measi_T]); //temp 1e-1degree // must be supressed
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        //a_T[1][measi_T] = adc2volt(T_CB[measi_T]);       // !!!
        a_T[2][measi_T] = adc2volt(T_int[measi_T]);       // !!!

        measi_T = (measi_T+1)%N_MEASURE;  // increment measure index
        measf_T = 2;     // new gyroscope measure available
}

int temperature_status(void)
{
        return measf_T;
}

//---------------------------------------------------------------------------
// Internal functions definition
//---------------------------------------------------------------------------
char volt2temp_LM94022(int volt)  // should use a look up table with the ADC 12-bit value
directly!!!!!!!!!!!!!!!
{                                                                    // for the momement, T is in [°C],
volt in 1e-4[V]
        if(volt<11590)
                return ((volt-11590)*5)/(-422)+50;
        else if(volt<15670)
                return ((volt-15670)*5)/(-408);
        else if(volt<19550)
                return ((volt-19550)*5)/(-388)-50;
        else
                return -51;
}

//---------------------------------------------------------------------------
// Interrupts
//---------------------------------------------------------------------------
//NO INTERRUPT HERE
```

E.4 Write in Flash memory

```
//*****************************************************************************
/*
* File : adcs_magnetometer.h
* Created by Hervé Péter-Contesse
* May 2007
*
* Header file for use to read/write flash memory of the microcontroller (SwissCube project).
* This file provide all functions to initialize, read and write into the flash memory.
* The data or measurement can be stored into the flash memory, and get back using the debugger :
*
* !!! To do this : you have to connect the debugger without erasing the flash memory!!!
* !!! With Code composer Essential v2, do :
                        !!!
* !!! project-->proprieties-->debug properties-->supress download and connect to target      !!!
* !!! then run and use the memory view to get back the results
        !!!
*/
//---------------------------------------------------------------------------

#ifndef FLASH_H_
#define FLASH_H_

void init_flash(void);            // init flash registers
void launch_the_program_only_once(char *flag_adress);      // If this function is called at the beginning of

        // the main function, the program can only be started

        // once ! It write/check a flag in Flash memory

        // Usefull to store measurement in Flash !!!

        // You have to reload the program to run the program !

// the following function are used to write and erase flash :
// !!! be carefull with the adress : you can erase the running program itself !!!
void erase_flash_segment(char* adress);   // erase a complete flash segment
void write_flash_int(int value, int* adress);
void write_flash_char(char value, char* adress);


#endif /*FLASH_H_*/


//*****************************************************************************
```

```
/*
* File : flash.c
* Created by Hervé Péter-Contesse
* May 2007
*
* Source file for use to read/write flash memory of the microcontroller (SwissCube project).
* This file provide all functions to initialize, read and write into the flash memory.
* The data or measurement can be stored into the flash memory, and get back using the debugger :
*
* !!! To do this : you have to connect the debugger without erasing the flash memory!!!
* !!! With Code composer Essential v2, do :
                        !!!
* !!! project-->proprieties-->debug properties-->supress download and connect to target      !!!
* !!! then run and use the memory view to get back the results
        !!!
*/
//-------------------------------------------------------------------------------

#include <msp430x16x.h>
#include "adcs_general_parameters_and_functions.h" //needed for "_LED1ON"
#include "flash.h"

#define F_FLASH 360      //[kHz] flash frequency generator, must be between 257kHz and 476kHz


void init_flash(void)
{
        int clk_divider = FMCLK0/F_FLASH;
        FCTL2 = FWKEY + FSSEL0 +clk_divider;
}


void launch_the_program_only_once(char* flag_adress)
{
        unsigned char i; // this line is necessary !!
        i = *flag_adress;         // this line is necessary too!!
        if(i == 0xFF)    // 1st program launch
                write_flash_char(0x01,flag_adress); // write flag in flash
        else
                while(1) // stuck the program here !
                {
                        _LED1ON;
                        LPM0;
                }
}

void erase_flash_segment(char* adress)
{
        _DINT();                          // disable interrupt
        FCTL1 = FWKEY + ERASE;    // Set Erase bit
        FCTL3 = FWKEY;           // Clear Lock bit

        *adress = 0;             // Dummy write to erase Flash seg

        FCTL1 = FWKEY;           // Clear ERASE bit
        FCTL3 = FWKEY + LOCK;    // Set LOCK bit
        _EINT();                          // enable interrupt
}

void write_flash_int(int value, int* adress)
{
         DINT();                          // disable interrupt
        FCTL1 = FWKEY + WRT;     // Set WRT bit for write operation
        FCTL3 = FWKEY;           // Clear Lock bit

        *adress = value;         // write in Flash

        FCTL1 = FWKEY;           // Clear WRT bit
        FCTL3 = FWKEY + LOCK;    // Set LOCK bit
        _EINT();                          // enable interrupt
}

void write_flash_char(char value, char* adress)
{
         DINT();                          // disable interrupt
        FCTL1 = FWKEY + WRT;     // Set WRT bit for write operation
        FCTL3 = FWKEY;           // Clear Lock bit

        *adress = value;         // write in Flash

        FCTL1 = FWKEY;           // Clear WRT bit
        FCTL3 = FWKEY + LOCK;    // Set LOCK bit
        _EINT();                          // enable interrupt
}
```

E.5 Magnetometer and magnetotorquers model

E.5.1 Matlab Magnetometer model

```
% magnetometer_model.m
% Created by Hervé Péter-Contesse
% March 2007
% Modified by
%-------------------------------------------------------------------------
%
% Magnetometer model for determination and control algorithms
% (The magnotometer used is an AK8970N and return a result on 8bits)
%
% function measuredEMF = magnetometer_model(EMF)
%
% - h is the time period [s] (time space between two EMF values/row in input EMF
% vector) : must be SMALLER than MEASUREMENT TIME/3 (belongs to
% [40e-3;170e-3]/3 [s])! (see value below).  IF THIS IS NOT POSSIBLE (for
% computation time?), put h=0 and use T only, but this model would not take
% into account the measurement time and measurement order
%
% - T is the measurement period [s] : a measure is done every t=k*T, k=0 for
% the fist input EMF vector row (t=0). Must be GREATER than MEASUREMENT
% TIME (belongs to [40e-3;170e-3] [s])! (see value below)
%
% - EMF is 3-by-n matrix containing the X,Y,Z (row) magnetic field
% components [nT] for the time tk = k*h, where h is the sampling time
% period and k<=n-1.
%
% - measEMF is a 3-by-floor(n*h/T) and return the XYZ magnetic components measured by
% the magnetometer [nT] (each component is handled separetely ; no
% interaction)
%
% This model doesn't take into account the linearity or the error due to
% the temperature variation (it just add noise for that), we assume theses
% corrections are made by the ADCS microcontroller.


function measEMF = magnetometer_model(h,T,EMF)

% Magnetometer parameters (these are experimental parameters !!!!)

measurement_time = 150e-3;  % [s] time taken by one measure (XYZ), belongs to [40e-3;170e-3]
sensitivity = 1.80e-3;      % [bit/nT] Sensitivity of the sensor (gain 23dB, 20ms = sens = 1.95e-3 ;  gain
18.2dB, 20ms sens = 1.1e-3)
                            % instanteneous, but is averaged during the measure
std_noise =  1.25 + 1.5;      % [bit] Standard deviation of the white noise

saturation_p = 127;         % [bit] Saturation (8bit)
saturation_m = -128;


if measurement_time > T
    error(['T must be greater than measurement_time=' num2str(measurement_time) 's']);
end

% Transfert function

[m n] = size(EMF);

if h == 0   % simplifed case
    EMF_ = EMF;

else        % normal case, take intergration time into account
    if h> measurement_time
        error(['h must be smaller than measurement_time/3=' num2str(measurement_time/3) 's , or put h=0']);
    end

    n_ = floor(n*h/T);
    EMF_ = zeros([m n_]);
    for k=1:n_ % compute magnetometer integration time
        for i=1:m
            p = floor((k-1)*T/h+1) + floor(measurement_time/3*(i-1)/h);
            q = floor((k-1)*T/h+1) + floor(measurement_time/3*i/h);

            EMF_(i,k) = mean(EMF(i,p:q));
        end
    end
end

[m n] = size(EMF_);
Nc = round(sensitivity*EMF_ + std_noise*randn([m n]));    % convert to bit and add noise
```

```
for i=1:m   % apply saturation for XYZ
    for j=1:n
        if Nc(i,j) > saturation_p
            Nc(i,j) = saturation_p;
        elseif Nc(i,j) < saturation_m
            Nc(i,j) = saturation_m;
        end
    end
end

measEMF = Nc/sensitivity;   % re-convert to magnetic field
```

E.5.2 Matlab Magnetotorquer model

```
% magnetometer_model.m
% Created by Hervé Péter-Contesse
% March 2007
% Modified by
%-------------------------------------------------------------------------

function [M] = magnetotorquer_model(ratio, B)
%
% Magnetotorquer model for determination and control algorithms.
% There are 3 perpendicular magnetotorquers : one for each X, Y and Z.
% Their torque can be modified changing the PWM ratio.
%
% This model assume the magnetotorquer temperature drift is corrected with
% the ADCS microcontroller with a knon precision (current and torque are
% not known exactly ; there is noise)
%
% function [M] = magnetotorquer_model(ratio, B)
%
% Inputs:
%   - ratio : 3-element vector : PWM ratio /Power ratio for each X,Y and Z
%     magnetotorquer. The value are in [-100,100]% (interger number!)
%
%   - B : 3-element vector : earth magnetic field vector in the satellite
%     fixed referential [Tesla]
%
% Outputs:
%   - M : 3-element vector : torque vector in the satellite fixed
%     referential[Nm]


% Magnetotorquer parameters:
I = 15e-3;      % [A] maximum current
Ierror = 5/100; % [%] current precision
N = 427;        % number of turns
l = 64.9e-3;    % [m] mean dimension of the small side
L = 74.9e-3;    % [m] mean dimension of the big side

A = l*L;        % [m^2] enclosed area
mu = N*I*A;     % [Am^2] maximum magnetic dipole moment

maxB = 60e-6;   % [T] maximum earth's magnetic field at 400km

% Compute torque
if any(abs(ratio)>100)
    error(['ratio not in the range : ' num2str(ratio)]);
end
if norm(B)>maxB
    error(['earths magnetic field is too big (not in range) : ' num2str(B)]);
end

[m,n] = size(ratio);
ratio = round(ratio);
mu_v = mu*(1+Ierror*randn([m n])).*ratio/100;

M = cross(mu_v, B);
```

Appendix F Other

All other information, components datasheets, source codes… can be found in the project CD-ROM.