Haute Ecole Spécialisée de Suisse occidentale
Katholieke Hogeschool Sint-Lieven

Inertial Platform for CubeSat

Master thesis to obtain the degree of
Master in Industrial Sciences:
Electronics-ICT

Presented by:
Dries Van de Winkel
Maarten Craeynest

Promotors:
Prof. Christophe Bianchi
Prof. François Corthay

Academic year: 2008-2009

Acknowledgements

We would especially like to thank François Corthay, Christophe Bianchi, Joseph Moerschell, Christian Costa, Hans-Peter Biner, Pascal Sartoretti, Lucio Kilcher, Olivier Walpen, Steve Gallay, Silvan Zahno, Pierre Pompili and everybody of the mechanical departement for their support and help during the accomplishment of this Master thesis.

Abstract (English)

Names: Dries Van de Winkel & Maarten Craeynest

Title: Inertial Platform for CubeSat

The infotronics department of HES-SO Valais is preparing a Master course on high performance signal acquisition and processing. This course will be illustrated with an example linked to a very small satellite, a CubeSat named SwissCube.
An Attitude Control and Determination System (ACDS) will be implemented.

The goal of this thesis is to realize a demonstrator of a one liter cube levitating in a magnetic field. The weight of this cube shall not exceed 400 g. This cube shall be able to control its vertical position (translation in the vertical axe Z) and its azimuth (rotation in the horizontal plane x,y) in order to secure its stability.
The project is composed of two parts: the horizontal one which is responsible for the rotation of the cube and the angle measurement, the vertical one which implements a distance measurement between the magnetic ceiling and the cube as well as a magnetic field regulation in order to control its vertical position.

No wire will be used, which indicates that the cube itself will send the necessary correction information to the control unit throughout a Radio Frequency (RF) link.
Like a real CubeSat, this cube should also have low power consumption: only one or two charges of the internal accumulator a day shall be necessary.

Keywords: CubeSat, RF-link, regulation, optical measurement, magnetic field

Abstract (Nederlands)

Namen: Dries Van de Winkel & Maarten Craeynest

Titel: Traagheidsplatform voor CubeSat

Het departement infotronics van de HES-SO Valais bereidt een Master cursus voor die het onderwerp 'high performance signal acquisition and processing' behandelt. Deze lessen zullen geïllustreerd worden aan de hand van een voorbeeld dat gelinkt is aan een heel kleine satelliet, een CubeSat, SwissCube genaamd.
Een 'Attitude Control and Determination System' (ACDS) zal geïmplementeerd worden.

Het doel van deze Master thesis is het realiseren van een demonstratiemodel: een kubus met een volume van één liter zweeft in een magnetisch veld. De massa van de kubus zal kleiner zijn dan 400 g. De kubus zal in staat zijn om zijn verticale positie te regelen (een translatie langs de verticale Z-as) en zijn azimut (rotatie in het horizontale X,Y-vlak) om op deze manier zijn stabiliteit te bewaren.
Het project wordt onderverdeeld in twee delen: de horizontale component is verantwoordelijk voor de rotatie van de kubus en de hoekmeting, de verticale component zorgt voor een afstandsmeting van het plafond tot de kubus alsook voor een regeling van het magnetisch veld om zo zijn verticale positie te regelen.

Aangezien het om een draadloos systeem gaat, wordt de corrigerende informatie door de kubus zelf doorgestuurd via een Radio Frequency (RF) link naar de besturing.
Zoals een echte CubeSat, moet ook deze kubus een laag vermogenverbuik hebben: er zullen slechts één tot twee oplaadbeurten per dag nodig zijn.

Sleutelwoorden: CubeSat, RF-link, regeltechniek, optische metingen, magnetisch veld

Abstract (Français)

Noms: Dries Van de Winkel & Maarten Craeynest

Titre: Plate-forme inertielle pour CubeSat

Le département Infotronics de la HES-SO Valais prépare un cours Master dans le domaine de l'acquisition et du traitement de signal à haute performance. Ce cours est illustré à l'aide d'un exemple basé sur un très petit satellite respectant le standard CubeSat et appelé SwissCube. Un Système de mesure et de contrôle d'attitude doit en outre être développé.

Le but de ce projet est de réaliser un démonstrateur formé d'un cube de 1 litre en lévitation dans un champ magnétique. Le poids de ce cube ne doit pas dépassé 400 g. Ce cube doit être capable de contrôlé sa position vertical (translation selon l'axe Z) et son azimut (rotation dans le plan horizontal x,y) afin d'assurer sa stabilité.
Le projet se compose de deux parties: la partie horizontale qui est responsable de la rotation du cube et de la mesure de son angle, la partie verticale qui implémente une mesure de distance entre le plafond magnétique et le cube ainsi qu'une régulation du champ magnétique de manière à contrôler sa position verticale.

S'agissant d'un système sans-fil, le cube envoie l'information de correction nécessaire à l'unité de contrôle en utilisant une liaison Radio Fréquence (RF).
Comme un vrai CubeSat, le cube doit avoir une consommation d'énergie faible: seulement un ou deux chargements par jour de la batterie seront possibles.

Mots-clés: CubeSat, liaison RF, régulation, mesure optique, champ magnétique

Contents

List of Figures

Acronyms

AC: Alternating Current
ADC: Analog-to-digital Converter
CRC: Cyclic Redundancy Check
DC: Direct Current
EMF: Electromagnetic Force
ESR: Equivalent Series Resistance
FIFO: First In, First Out
GND: Ground
GPIO: General Purpose Input Output
ISR: Interrupt Service Routine
LED: Light Emitting Diode
LPF: Lowpass Filter
LSb: Least Significant Bit
LSB: Least Significant Byte
MCU: Microcontroller Unit
MEMS: Micro-Electro-Mechanical Systems
MIPs: Million Instructions per Second
MOSFET: Metal-Oxide-Semiconductor Field-Effect Transistor
MSb: Most Significant Bit
MSB: Most Significant Byte
MSSP: Master Synchronous Serial Port
OPAMP: Operational Amplifier
PCB: Printed Circuit Board
PIC: Peripheral Interface Controller
PLL: Phase-locked Loop
PSD: Position Sensitive Device
ptp: Peak-to-peak
PWM: Pulse-width Modulation
RF: Radio Frequency
rpm: Rotations Per Minute

RX:	Receiving
SMD:	Surface Mount Device
SPI:	Serial Pheripheral Interface
TFF:	Transfer Function
TX:	Transmission
WLAN:	Wireless Local Area Network

Chapter 1

Introduction

The infotronics department of HES-SO Valais is preparing a Master course on high performance signal acquisition and processing. This course will be illustrated with an example linked to a very small satellite, a CubeSat named SwissCube.
The goal of this Master thesis is to realize a demonstrator of an object levitating in a magnetic field.
The idea of realizing such a demonstrator is based on an existing levitation system designed at HES-SO. In the existing system a magnetized object levitates in a magnetic field at a fixed distance from the ceiling (bottom of the base station). The analog processing to keep the object floating is integrated in the base station, the object itself contains no electronics.
In our demonstrator, we want the object to float at a variable distance from the ceiling. Therefore, the object has to measure its distance to the *base station*. The object shall also be able to rotate and to measure its angle. Communication between the object and the base station shall be done wireless. Because our object contains electronics, an internal accu is necessary. This accu shall be charged as less as possible, so power consumption inside the object is very important. The levitating object in our demonstrator has the same shape as the real SwissCube: a *cube* with a volume of one liter.

Analysis

After analysing this assignment, the most important aspects are:

- Power Consumption and Charging
- Levitation (Magnetism)
- Distance Measurement (Optics)
- Rotation
- Regulation
- Wireless Communication

Power Consumption and Charging
To provide the cube with the necessary energy to accomplish all its tasks and to charge its accumulator (battery), we need a system that delivers this energy. Because we do not want to interrupt the operation of the cube to charge it, we decide to use power induction. This technique can transfer energy through the air over a certain distance and is based on inductive coupling: a primary (sender) coil works as a transducer as it transforms electrical energy into a magnetic field. A secondary (receiver) coil placed in this magnetic field will also work as transducer, but now transforming magnetic energy into electrical energy again. The sender coil will be placed inside the base station and thus the receiver coil inside the cube.
Power consumption is also an important aspect of our design. This consumption has to be as low as possible, because the amount of energy that can be transfered with power induction is limited.

Levitation (Magnetism)
To levitate the cube in the air, we use a magnetic field. This magnetic field is created inside the base station and will change, depending on the height of the cube. The existing levitation system (previously designed at HES-SO) forms a good example of how to make this: a small current is amplified using a power transistor and this current is lead to the levitation coil, the result is a magnetic field. In this system, the controlling of the magnetic field is done in an analog way. We will do this in a digital way using a processing unit.

Distance Measurement (Optics)
A distance measurement from the cube to the bottom of the base station has to be done. This information will be used to control and change the magnetic field. When the cube is further from the base station, a greater distance is measured and so the magnetic field should be stronger.

As our control loop is 1 ms (see later), we need a fast measurement. Besides that, we also want a very precise measurement as one should not be able to see the cube vibrating in the air.
Classical methods to do a distance measurement, such as a PSD or an ultrasonic device, do not meet the requirements stated above as they are not accurate enough or their response time is too slow.
A method using a laser, a MEMS-mirror and two optical sensors does meet the requirements we want.

Rotation
The cube has to rotate in the magnetic field. A motor with an interia wheel inside the cube will make this rotation possible. Information about the rotational movement of the cube has to be collected to regulate the speed and direction of the cube. This information will be provided by a single-chip gyroscopic system.

Regulation
In order to control the horizontal and vertical movement of the cube, a closed control loop will be used. This regulation system will be, as much as possible, implemented in the cube.
A basic control loop consists of three main elements:

- A system
- A sensing device
- A controller

In the control loop for both the horizontal and the vertical movement, the cube represents the system. However, the system output variable differs: for the horizontal movement this is the *rotational speed*, for the vertical movement the *altitude* of the cube.
In both control loops, a processing unit acts as the controller, meaning that this element of the control loop is implemented in software. The software analyses the output of the sensing device, and commands an input for the system to be controlled.
For the horizontal movement, a motor inside the cube steers the system. For the vertical movement, the element that steers the system needs to be implemented outside the cube: the levitation coil inside the base station.
After analysing the existing levitation system, we decide to make the control loop timing not longer than 1 ms. This value is chosen so that the cube remains stable and does not vibrate in the air.

Wireless Communication
As the distance measurement is used to control the magnetic field, this information, and also

other information and commands, have to be sent from the cube to the base station and vice versa. This is done wireless. There are a lot of technologies available on the market to realise a wireless connection like Infrared, Bluetooth, WLAN, etc. We will use the worldwide 2.4 GHz band to establish the wireless communication.

All the aspects mentioned above, will work together in one system:

- The heart of the cube and the base station is a processing unit. This processing unit has to organize everything and spread the tasks that have to be done.
- A rotation and distance measurement has to be done. This information goes to the processing unit and is used to adjust the rotation of the cube and its distance to the ceiling.
- A transceiver takes care of the wireless communication between the cube and the base station.
- All the elements inside the cube and the base station need a proper alimentation.

A global image of the system can be found in chapter 3.

Course

The course of this report is the following:

We start with the Requirements and Global System Architecture of our system.

Next, chapter 4 ‘Hardware’ fills in all the elements of the Global System Architecture. The first section talks about the elements inside the Base Station: *Levitation*, *Power Induction* and *User Interface*. The next section discusses the different components inside the Cube: *Rotation*, *Distance Measurement*, *Charging and Power Supply* and as last *Power Consumption*. Elements that appear in both the base station and the cube, namely *Processing Unit* and *Wireless Communication*, are also explained in this chapter. A Detailed System Architecture and some comments on the PCB Design close this chapter.

In chapter 5 ‘Software’ we keep the same order as in the previous chapter. We first discuss all the necessary software inside the Base Station: *Startup*, *Synchronisation*, *Levitation*, *Power Induction* and the *User Interface*. Next, the different algorithms implemented in the Cube are discussed: *Startup*, *Synchronisation*, *Rotation*, *Distance Measurement* and *Charging*. The last section of this chapter talks about the *Wireless Communication*.

The Results of this project and the final Specifications are given in chapter 6.

Conclusions and Future Work form the final chapter of this report.

Chapter 2

Requirements

The designed system has to meet certain requirements: some of these are based on the specifications of the real SwissCube, others are necessary to obtain a stable and well working system. Furthermore, some requirements and specifications are stated as to the capabilities of the system.

Specifications inherited from SwissCube:

- The cube has a volume of 1 l.
- Power consumption in the cube does not exceed 1 W.

For the system to be stable, following requirements are postulated:

- The cube must weigh 400 g or less.
- The behaviour of the cube is controlled by a 1 ms control loop.
- A distance measurement with a precision of 0.2 mm is needed.

Other specifications of the system:

- The cube can obtain a rotational speed of 8 rpm and more.
- The cube obtains this speed in 5 s at most.
- The rotational speed is maintained with a precision of $1°$/s.
- The vertical position of the cube lies between 3 cm to 20 cm from the ceiling.
- Half-duplex wireless communication is used.
- The accu in the cube is charged on-line.

Chapter 3

Global System Architecture

Figure 3.1 shows the global system architecture as a result of the analysis in the Introduction. A detailed version of this architecture is given at the end of chapter 4 'Hardware'.

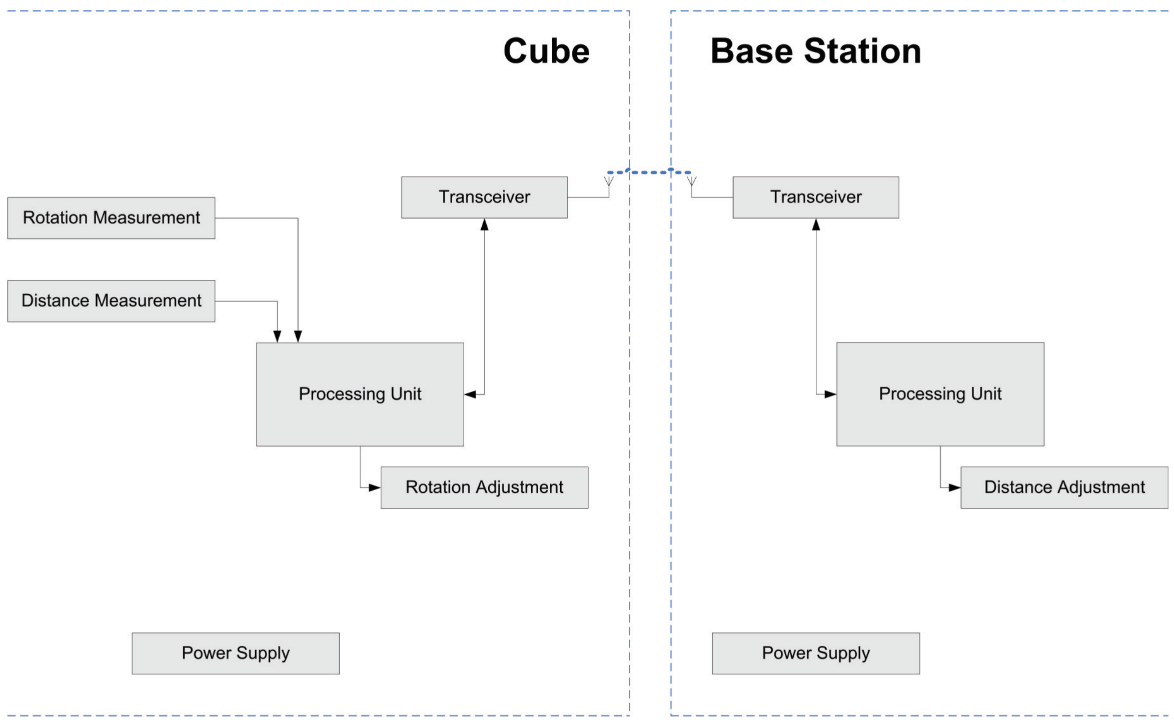


Figure 3.1: Global System Architecture

Chapter 4

Hardware

In this chapter, the hardware of the system is discussed. Each section describes the function of the different modules, and explains the selection of components to accomplish these tasks. This discussion is summarized in the detailed system architecture (section 4.4 on page 37). The schematics are included in appendix B on page 71.
Section 4.5, at the end of this chapter, gives some considerations and constraints for the PCB design. The final PCB layout for the different boards is included in appendix C on page 96.

4.1 Base Station

Although the cube is the core element of the system - and thus containing most of the functionality - certain functions and aspects need to be taken care of outside the cube. With the existing levitation system in mind, these functions are grouped together in the so-called base station.

- The cube has to levitate in a magnetic field, which must be generated in the base station.
- To provide the cube's hardware with energy, there is an accumulator inside, which has to be recharged when its charge is running low. The energy to recharge the accu will be provided by the base station.

4.1.1 Levitation

In order to levitate the cube in the air, magnetism is used. Two magnetic poles of opposite polarity apply an attractive force on each other, which makes it possible to create an upward force on the cube. In the cube, permanent magnets are used, but in the base station, we need to form a variable magnetic field to establish a regulation system. This is done by using a cylindrical wound coil. By sending a DC-current through the coil, a magnetic field is created, with a magnitude depending on the amount of current.
To send and control high DC-currents through the coil, a bipolar power transistor is used,

amplifying the base current to a collector current, which is the current through the coil. Since the MCU has to control the amount of collector current through the coil, we need a circuit that transforms a MCU-signal into a base current. A PWM module of the MCU can be used together with a LPF to create a discrete analog voltage. To convert this voltage into a current, a resistor is used.

PWM Filter

The MCU delivers a PWM signal, of which the duty cycle corresponds to the wanted strength of the levitation magnetic field. When this signal is sent to a LPF, a DC voltage remains. This DC voltage, being the mean value of the PWM signal, is proportional to the duty cycle δ. The stronger we want the magnetic field, the higher this voltage will be.

The LPF is implemented in the Sallen&Key-topology, consisting of a R and C circuit built around an operational amplifier. The main advantage of this topology is the high input impedance and low output impedance of the OPAMP, so that the filter characteristics are not influenced by the current flowing to the base of the transistor: the OPAMP acts as a buffer between the filter and the transistor.

We want a filter with a low cut-off frequency f_c, since we are only interested in the DC-value of the PWM signal. However, the cut-off frequency should be high enough to make sure that the output of the filter has a sufficient fast response to fast changes in the duty cycle δ. A cut-off frequency of 1 kHz (frequency of the control loop) is chosen, on condition that the PWM frequency is 10 kHz or higher.
The quality factor Q of the filter is less important here, but we choose to implement a Butterworth filter (having a maximal flat frequency response in the passband), this means that $Q = \frac{1}{\sqrt{2}}$.
The configuration of the filter is shown in figure 4.1, the theoretical values for the R & C components are calculated in appendix A.1. In the figure below, the chosen component values from the E12 series are shown.

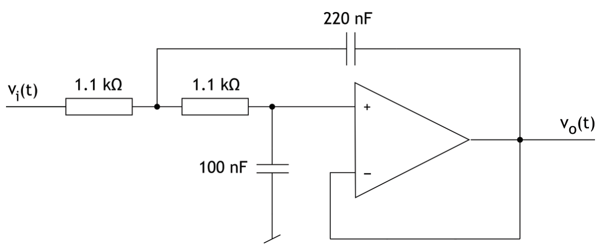


Figure 4.1: Configuration of the PWM filter

Figure 4.2 shows the input and output of the PWM filter. The input is a PWM signal with duty cycle $\delta = 20\%$ (the high level is 5 V), the output is a DC-signal of $20\% \cdot 5\,\mathrm{V} = 1\,\mathrm{V}$. Note that the frequency of the PWM signal is higher than 10 kHz.

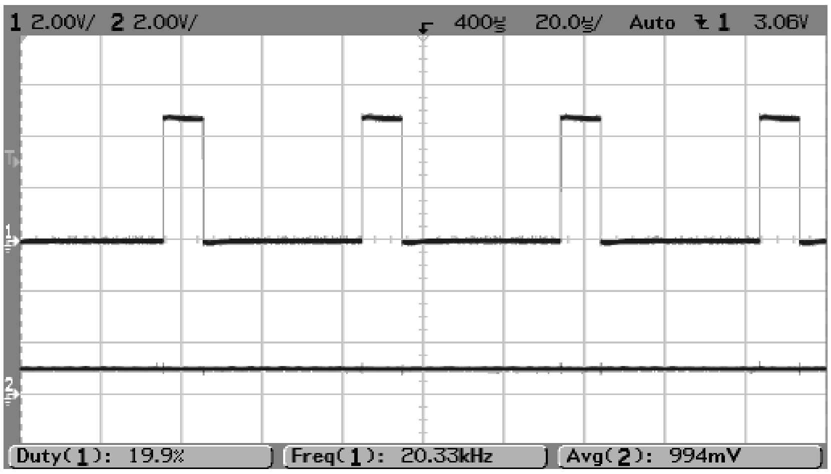


Figure 4.2: Input and output of the PWM filter, $\delta = 20\%$

Drive Circuit

The output of the Sallen&Key filter is a DC voltage. To control the levitational magnetic field, a certain DC current is to be sent through the coil. A bipolar power transistor is used to amplify the commanding current. A transconductance is needed, to convert the voltage output of the filter into a current, the base current of the transistor. This can simply be done using a resistor.

We need to make sure that the current through the coil does not exceed a certain value. This could be done by limiting the base current of the power transistor, but since the forward current gain β is temperature dependent, it is better to watch the actual collector current. For this, a small sensing resistor is used. By placing 4 resistors of $0.1\Omega, 1\,\mathrm{W}$ in parallel, we have an equivalent resistor of $0.025\Omega, 4\,\mathrm{W}$. When a current of 8 A (this value is explained in section 4.1.1 on page 11) flows through this equivalent resistor, a voltage of 0.2 V is seen. We need to place resistors in parallel so that they can withstand the $0.025\,\Omega \cdot (8\,\mathrm{A})^2 = 1.6\,\mathrm{W}$ that will be dissipated at 8 A.
The voltage over the parallel resistors is compared with a 0.2 V reference voltage. When the sensing voltage is higher than the reference voltage, the comparator output switches to a high level, allowing a N-channel enhancement MOSFET to lead away the base current of the power transistor. To prevent oscillation, a 10 nF capacitor is placed between the comparator output and ground. When current limiting, the comparator output stabilises at 3 V. This voltage is compared with a 1 V reference voltage, to switch on a LED that indicates current limiting.

When a large current flows through the coil, which has a certain resistance, there will be a voltage over the coil. To make sure that enough voltage is left for the transistor to work properly, we need a supply voltage of at least 40 V.
The power transistor dissipates a lot of heat, so it is placed on a cooler.

Tests have been done at HES-SO, using a power MOSFET instead of a bipolar transistor. In this setting, the PWM filter and the transconductance resistor are left out, so the PWM signal is directly placed on the gate of the MOSFET. This method has an easier implementation, and less heat dissipation, but has not proven its results.

Levitation Coil

The last stage of the 'Levitation' section is the element that creates the actual magnetic field. When we use a coil that is wound in a cylindrical form, we obtain a magnetic field that is similar to that of a bar magnet. Together with a magnet (with opposite polarity) inside the cube, this will give the appropriate force that keeps the cube floating.

When we simplify the cube as a point mass of 400 g, we can calculate the required force with Newton's second law:

$$\begin{aligned} F &= m \cdot a \\ F &= 0.4\,\text{kg} \cdot 9.81\,\frac{\text{N}}{\text{kg}} \\ &= 3.924\,\text{N} \end{aligned}$$

According to the Biot-Savart law for a cylindrical coil:

$$B = \frac{\mu_0 \cdot I \cdot R^2}{2 \cdot (R^2 + z^2)^{\frac{3}{2}}}$$

the magnetic field density B is proportional to $\frac{1}{z^3}$. So increasing the distance z to the coil, along the vertical axis, will decrease the magnetic field with a factor z^3.
The simulation program *Maxwell 12* gives us the possibility to calculate the number of necessary windings and to see what the maximum distance is the cube can achieve.
While using this program, we have to keep some constraints in mind:

- We have to make a movable construction, so the weight and dimensions of the coil are important.
- The maximum current through a wire of $2\,\text{mm}^2$ is 8 A.
- The magnet inside the cube can not be too heavy, because the cube's mass is limited to 400 g.

A force of 4.41 N on the magnet inside the cube is found at a distance of 80 mm. We decide to take this value as a limit because we already need 2100 windings at 8 A.
Figure 4.3 gives the final result in *Maxwell 12*.

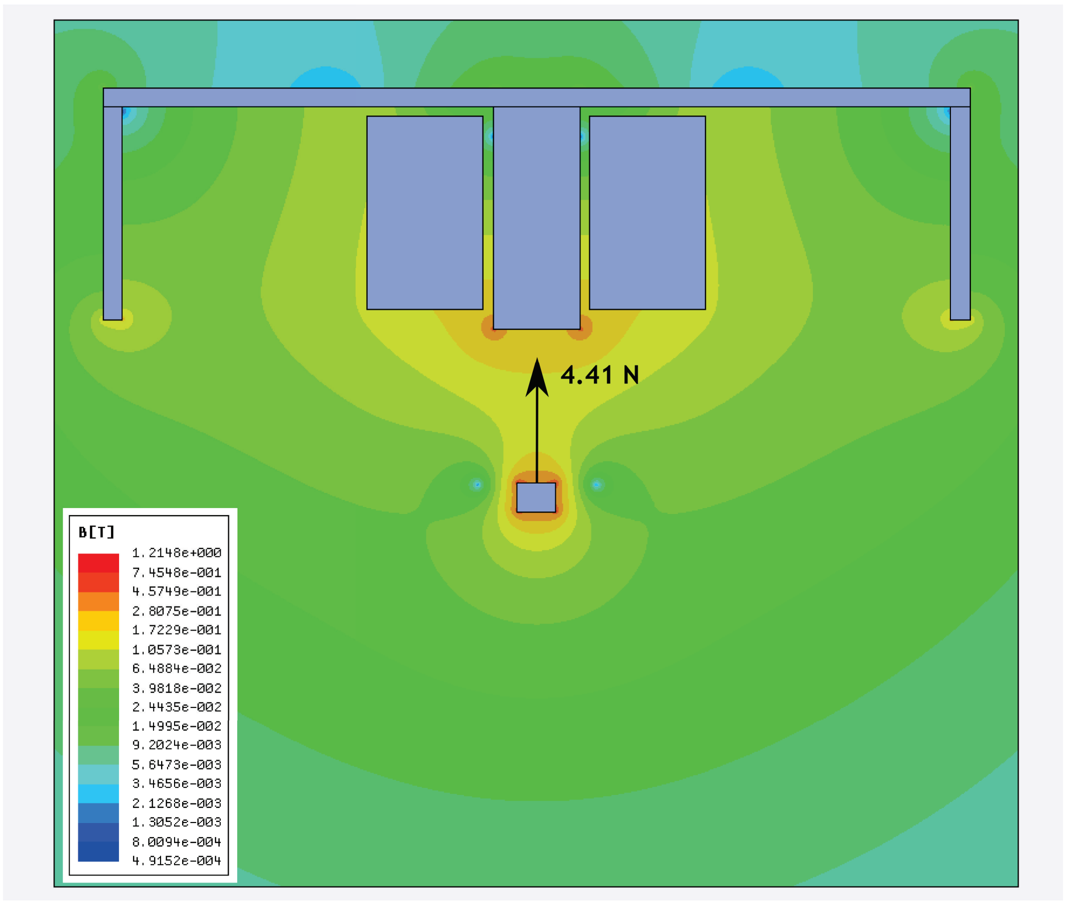


Figure 4.3: Simulation in *Maxwell 12*

Placing a U-shaped bar on top of the coil makes sure that the magnetic field lines are concentrated towards the magnet. The following materials are used during simulation: Steel (the U-shaped bar and the core of the coil), NdFe35 (the magnet) and Copper (the coil).
The radius of the coil varies from 30 to 90 mm, which results in a total length l for the copper wire of:

$$l = 2\pi \cdot r \cdot \alpha = 2\pi \cdot \frac{0.03 + 0.09}{2} \cdot 2100 = 791.68\,\mathrm{m}$$

With a volumetric mass density ρ for copper of $8,96\frac{\mathrm{g}}{\mathrm{cm}^3}$ we find a mass of:

$$m = V \cdot \rho = 0.02\,\mathrm{cm}^2 \cdot 79168\,\mathrm{cm} \cdot 8,96\frac{\mathrm{g}}{\mathrm{cm}^3} = 14186\,\mathrm{g} = 14.186\,\mathrm{kg}$$

The steel core has a $\rho = 7,85\frac{\text{g}}{\text{cm}^3}$:

$$m = V \cdot \rho = \pi \left(\frac{4.5\,\text{cm}}{2}\right)^2 \cdot 12\,\text{cm} \cdot 7,85\frac{\text{g}}{\text{cm}^3} = 1498\,\text{g} = 1.498\,\text{kg}$$

This results in a total mass for the levitation coil of 15.684 kg.
A mechanical drawing of the levitation coil is included in appendix F. This coil is constructed in the mechanical department and has the following characteristics: inductance $L = 444\,\text{mH}$, resistance $R = 6.5\,\Omega$.

4.1.2 Power Induction

To provide the cube with the necessary energy to accomplish all its tasks and to charge its accumulator (battery), we need a system that delivers this energy. This is done with power induction.
Power induction is based on inductive coupling: a primary (sender) coil works as a transducer as it transforms electrical energy into a magnetic field. A secondary (receiver) coil placed in this magnetic field will also work as transducer, but now transforming magnetic energy into electrical again.
This non-resonant induction method is inefficient and wastes much of the primary energy. Therefore it is important that we create a resonance circuit with the same resonance frequency on both sides. This resonance improves efficiency dramatically.
Figure 4.4 shows the principle of power induction: the left-hand circuit is placed inside the base station and discussed in this section. The right-hand one is placed inside the cube and is discussed in section 4.2.3 on page 28.

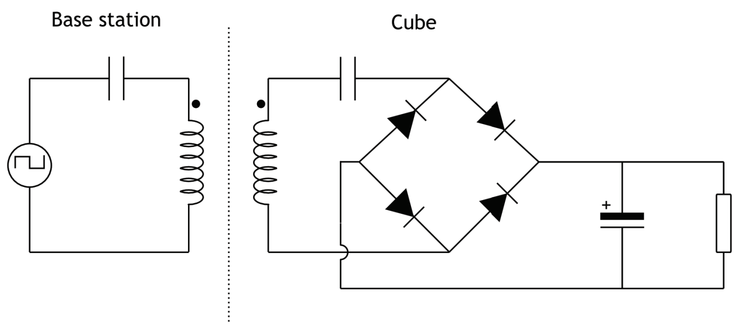


Figure 4.4: Power induction

Resonance Circuit

Faraday's Law gives the induced EMF on a coil placed in a magnetic field:

$$\epsilon = -N \cdot \frac{\mathrm{d}\,\Phi_B}{\mathrm{d}\,t} \tag{4.1}$$

In this formula, N is the number of windings and $\frac{\mathrm{d}\,\Phi_B}{\mathrm{d}\,t}$ is the variation of the flux.

We can influence these two elements to become a higher EMF: increasing the frequency and the number of windings will result in a higher induced EMF on the receiver coil (i.e. the coil inside the cube).
The configuration of the levitation coil with a U-shaped bar on top of it (section 4.1.1), introduces many Foucault currents at higher frequencies, which decreases efficiency. In other words, there are too many losses when we want to realize power induction with the levitation coil. So this coil can not be used for power induction, we need a secondary coil inside the base station.

As stated in the introduction it is very important to create a resonance circuit so that the efficiency of power induction improves. A series resonance circuit exists of two basic components placed in series: an inductor (i.e. the coil) and a capacitor.
The resonance frequency f_{res} of this circuit is given by the following equation:

$$f_{res} = \sqrt{\frac{1}{(2\pi)^2 \cdot L \cdot C}} \tag{4.2}$$

The quality factor Q of this series resonance circuit is given by the following equation:

$$Q = \frac{\sqrt{\frac{L}{C}}}{R} \tag{4.3}$$

As one can see in (4.3), the Q factor is proportional with $\sqrt{\frac{L}{C}}$. Because we want the Q factor as high as possible, we combine a large inductance with a small capacity.
When we construct a coil of 100 windings, we measure an inductance $L = 3.79\,\mathrm{mH}$. If we are working with a resonance frequency of approximately 20 kHz, we can now calculate the necessary capacity to create resonance, using (4.2):

$$\begin{aligned} C &= \frac{1}{(2\pi \cdot f_{res})^2 \cdot L} \\ &= \frac{1}{(2\pi \cdot 20 \cdot 10^3\,\mathrm{Hz})^2 \cdot 3.79 \cdot 10^{-3}\,\mathrm{H}} \\ &= 16.71\,\mathrm{nF} \end{aligned} \tag{4.4}$$

We want the capacitor to have a low ESR, because according to (4.3) this has a positive influence on the Q factor. This capacitor also has to be able to resist a maximum reactive current of 8 A ptp that could be introduced in the circuit at resonance.

Drive Circuit

The drive circuit for the power induction was already designed at HES-SO. Two N-channel enhancement MOSFETs in half-bridge configuration are opened or closed by a bootstrap

driver. This half-bridge places a square voltage over the serie LC circuit, as shown in the left part of figure 4.4 on page 13: the LC circuit is, alternatingly, connected to the supply voltage and to ground.
The square input signal, generated by the MCU or function generator, is inverted so that two complementary signals are present. At the same time, a signal conditioning circuit is present, making sure that the two square waves are never high at the same time. This would result in two closed MOSFETs, creating a direct pad from the supply to ground, this is of course not desirable.
The bootstrap driver is used to convert the logic square signals into an appriopriate signal level for the MOSFETs.

4.1.3 Power Supply

As we saw in paragraphs 'Drive circuit' and 'Levitation coil' of section 4.1.1 we need a voltage of at least 40 V and an active current of 8 A.
In the 'Power Induction' section we saw that we also need a certain current to obtain a sufficient transfer of energy. This active current is less than 2 A.
Since 48 V is an industrial standard, we decide to use a power supply that delivers an active current of 10 A at 48 V. The *Lambda DPP-480* power supply is ordered and is placed inside the base station.

4.1.4 User Interface

A user interface is provided in the base station. This interface gives the user the possibility to control the cube. There are five different commands to give:

1. *stop*, the cube stops turning.
2. *up*, the cube goes up.
3. *down*, the cube goes down.
4. *left*, the cube turns left.
5. *right*, the cube turns right.

All commands are sent to and processed inside the cube.
Commands 2 and 3 have an influence on the distance (height) of the cube and hence the strength of the magnetic field will be influenced.
Commands 1, 4 and 5 have an influence on the rotational speed and the direction of the cube. Giving the same command multiple times after each other will accumulate the effect: for example, the cube goes faster in the left direction if the *left* command is given multiple times after each other.

Five buttons are provided to give these five commands. When a button is pressed, it is connected to GND and the MCU detects the corresponding command: the commands are falling-edge interrupts.

4.2 Cube

In this paragraph we discuss the core element of the system, i.e. the cube.
We will describe the different elements that are implemented in the cube:

- The cube must be able to rotate and collect information about its speed and direction of rotation.
- A distance measurement has to be done by the cube.
- An internal accu must be charged on-line using power induction.
- The cube must have a low power consumption, since the transferable energy using power induction is limited.

4.2.1 Rotation

When suspended in the magnetic field, the cube must be able to rotate at a certain speed. To start rotating, an angular acceleration α must be given to the cube. This acceleration originates from the applied torque τ, according to:

$$\tau = I \cdot \alpha \tag{4.5}$$

where I is the *Moment of Inertia*, also referred to as *inertia*. The torque applied to the cube, divided by the inertia of the cube, will determine the angular acceleration of the cube. Note that (4.5) is the rotational equivalent of $F = m \cdot a$ for translation. In this last formula, the mass m (just like the inertia I) stands for the capability to resist a change of movement.

The principle to apply the desired torque on the cube, is based on Newton's third law:

"To every action there is an equal and opposite reaction."

When we fix a motor in the cube, and send current through its terminals, this current will result in a torque on the rotor of the motor, making the rotor rotate. However, we are interested in the rotation of the cube. According to the law stated above, by applying a torque on the rotor *(action)*, there will be an equal and opposite torque *(reaction)* on the stator of the motor, and via its mounting screws also on the cube.

Motor

Certain requirements regarding the rotation of the cube are stated in chapter 2:

- The cube can obtain a rotational speed of 8 rpm and more.
- The cube obtains this speed in 5 s at most.

These requirements are used to select an appriopriate motor, the calculations can be found in appendix A.2. A similar motor as the one found in the calculations was at our disposal at HES-SO, and was able to meet the requirements regarding the speed and acceleration of the cube. The measured *no load current* of this motor is 18 mA.

Drive Circuit

Now that an appriopriate motor is selected, we need to drive it. In order to stabilize the cube at a certain speed, it needs to be accelerated and decelerated. This is possible with a H-bridge (full bridge), by sending current to the motor in two directions. The configuration of a H-bridge is shown in figure 4.5.

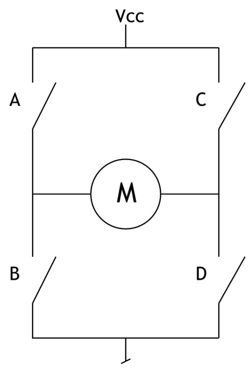


Figure 4.5: Configuration of an H-bridge

By closing switches A and D, the motor turns in one direction, by closing switches B and C, it turns in the other direction.

As switches, MOSFETs are used. This type of transistor is voltage-driven and draws a very small current to its gate. Switches B and D, at the bottom side of the H-bridge, are N-channel enhancement MOSFETs, they need a high voltage at their gates to switch *on*. Switches A and C, at the top side of the H-bridge, are P-channel enhancement MOSFETs, they need a low voltage at their gates to switch *on*.
A careful selection is required: even when the MOSFETs act as closed switches, they still have a certain resistance (the drain-source resistance $R_{DS}(ON)$). When this resistance is too high, and the motor needs its starting current to begin rotating, the voltage for the H-bridge will be placed over the MOSFETs, and no voltage will remain over the terminals of the motor. This means the motor will not start rotating.
The chosen MOSFETs have a $R_{DS}(ON)$ of 0.02 Ω, they only take a voltage of 6 mV when 300 mA (starting current when a rather large load is attached) is flowing to the motor.
The supply for this H-bridge is V_{BAT}, so the motor works at a nominal voltage of 3.7 V (a lithium polymer battery is used, see section 4.2.3).

With this H-bridge, it is possible to set the motor in the four following states: *turn left*, *turn right*, *brake* and *run free*. Turning left or right happens by sending current through the motor in a particular direction. Free running happens by opening all switches so no current flows through the motor terminals. Braking happens by placing 0 V over the terminals of the motor, this is done by closing the upper switches.

By using 2 digital bits (D_0 and D_1), each of these 4 states can be represented. The table in figure 4.6 shows the chosen correspondence between the state bits, and the position of the switches to effectuate the chosen state (a `C` represents a closed switch, a `O` represents an open switch).

state	D_1	D_0	A	B	C	D
brake	0	0	C	O	C	O
right	0	1	O	C	C	O
left	1	0	C	O	O	C
free	1	1	O	O	O	O

Figure 4.6: The 4 states for the motor: position of the switches

To close the lower N-channel MOSFETs (to let them conduct), a logic high is placed on the gate. To close the upper P-channel MOSFETs, a logic low is placed on the gate. The table in figure 4.7 shows the logic levels at the gates of the MOSFETs to effectuate the chosen state.

state	D_1	D_0	$GATE_A$	$GATE_B$	$GATE_C$	$GATE_D$
brake	0	0	0	0	0	0
right	0	1	1	1	0	0
left	1	0	0	0	1	1
free	1	1	1	0	1	0

Figure 4.7: The 4 states for the motor: logic level at the gates

With some logic ports, the digital bits coming from the MCU can be converted to the logic levels for the gates. When a PWM signal is added to the signal at the gates of the lower MOSFETs, to control the speed of the motor, we see the following logic functions to be implemented:

$$\begin{aligned}
\mathrm{GATE_A} &= \mathrm{D_0} \\
\mathrm{GATE_C} &= \mathrm{D_1} \\
\mathrm{GATE_B} &= \overline{\mathrm{D_1}} \cdot \mathrm{D_0} \cdot \mathrm{PWM} \\
\mathrm{GATE_D} &= \mathrm{D_1} \cdot \overline{\mathrm{D_0}} \cdot \mathrm{PWM}
\end{aligned}$$

Inertia Wheel

Equation (4.5) on page 17 shows us that, for a given torque applied on an object, the acceleration depends on the inertia of the object. When no load is attached to the rotor of the motor, there is only the small inertia of the rotor: a given torque will result in a high acceleration, and the rotor will quickly arrive at its no load speed. At that point, no torque is applied to the rotor, and thus no reaction torque on the cube. This small time of torque applied to the cube (with its high inertia), results in a very small and shortly lasting acceleration: the cube does not start turning.

If we want the reaction torque to be applied to the cube for a longer time, so that it can start turning visibly, we make use of two aspects:

- When a load is attached to the rotor, the inertia increases. This results in a lower acceleration, and thus a longer time of applied torque before the motor arrives at its no load speed. Since increasing the inertia is the main goal of this load, it is also called an *inertia wheel*.
- Depending on the shape of the load, there will be friction in the air. This is shown in figure 4.8. When the motor arrives at speed ω_f, the applied torque will be equal in size to the friction torque τ_f. The motor will keep compensating for this friction, and a remaining reaction torque will keep being applied to the cube.

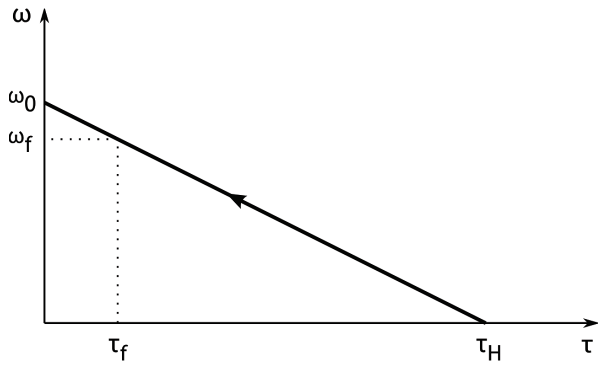


Figure 4.8: Behaviour of the motor for constant voltage, friction, accelerating

In most applications involving a motor, inertia matching is important to obtain a sufficient energy efficiency η. This means that the ratio of the reflected load inertia (equals the load inertia when no gearhead is used) to the rotor inertia should not exceed 10:1, ideal is to obtain a ratio of 1:1.
However, in our application this is not the most important design item. We want the behaviour of the cube, in terms of rotation, to be sufficiently responsive to given commands. For example, when the cube is hanging still and a 'turn left' command is given, we want the

cube to obtain a speed of 8 rpm in 5 s at most. Choosing the right inertia for the wheel makes it possible to obtain this responsitivity.
Several tests with different inertia wheels have been done, these tests can be found in appendix A.3. An optimal wheel inertia (optimal for the setting of this project: a particular inertia of the cube, a particular choice of motor, ...) is found: $11.88\,\mathrm{kg}\cdot\mathrm{m}^2\,10^{-7}$.
The resulting graph is also shown in figure 4.9. The *'Time to make 5 rotations, starting from standstill'* curve proves the first item stated above: a higher wheel inertia results in a better acceleration of the cube.
The *'Direction change time'* curve indicates the influence of friction (second item): when a torque is applied to a high inertia, the acceleration is not very high, and the wheel does not obtain a high speed. It causes no movement of the air around it. The smaller inertia turns fast and causes a lot of wind movement (if the wheel is sufficiently large). When changing the direction of the wheel, there will be a lot of friction in the air, resulting in a better responsitivity of the cube.
Because of this influence of friction, two wheels with the same inertia but different shapes are likely to give different results. A cylindrical wheel with an inertia of $11.88\,\mathrm{kg}\cdot\mathrm{m}^2\,10^{-7}$ will probably result in a lower acceleration than the chosen bar of the same inertia. This has not been tested.

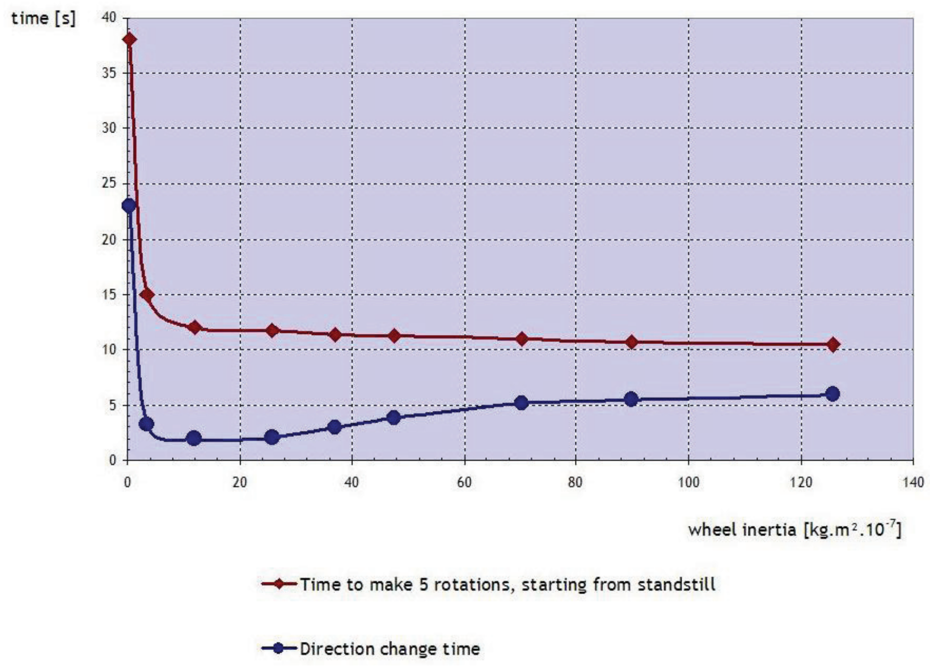


Figure 4.9: Timing values of cube responsitivity in function of wheel inertia

For testing purposes in the final system, an optical encoder is provided, this makes it possible to detect the speed of the inertia wheel. The used encoder is the *Agilent AEDR-8100-1P2*[7].

The mechanical drawing for the final inertia wheel can be found in appendix F.

Rotation Measurement

In order to measure the rotational speed of the cube, certain methods are available. A first one is the use of an optical encoder, which measures speed by detecting the amount of fixed distance holes in an encoding wheel, over a certain time.
Another possibility is using the MEMS technology. Many MEMS sensors exist, including linear acceleration sensors, angular acceleration sensors and angular rate sensors. Since this last type directly gives the needed information, while the others involve another calculation, it is good to choose the angular rate sensor.
An angular rate sensor contains a sensing element inside, sensing the *yaw rate* of the chip, and outputs this rotational speed through an analog voltage.

The position of the angular rate sensor on the PCB is not critical nor constrained. When the chip would be placed so that the sensing element is exactly on the center of rotation of the cube, the sensing element would only undergo a rotational movement, not a translation. When the chip has another position on the PCB, a translational movement is added, but the rotational movement stays the same.

In this project, the *LISY300AL Yaw Rate Gyroscope*[6] is used. It detects rotational speeds in the range $\pm 300°/\mathrm{s}$, and has a sensitivity of $3.3\,\frac{\mathrm{mV}}{°/\mathrm{s}}$.
The analog output needs to be converted into a digital word in order to do the necessary calculations. This can be done with the MCU's internal 10-bit ADC, for which several reference ranges are available. When we choose 3.3 V and 0 V as reference voltages, the ADC has a resolution of 3.3 mV (with the 10-bit precision of the ADC, 1024 different analog values can be distinguished within the chosen reference range). Taking the specified sensitivity of the ARS into account, we arrive at a speed measurement with a precision of $1°/\mathrm{s}$.
The requirements in chapter 2 stated:

- The cube can obtain a rotational speed of $8\,\mathrm{rpm} = 48°/\mathrm{s}$ and more.
- The rotational speed is maintained with a precision of $1°/\mathrm{s}$.

The chosen angular rate sensor meets this requirements, regarding the *measurement* of rotation.

4.2.2 Distance Measurement

Along the vertical axis, the cube has to keep a certain distance to the base station. To hold and stabilize its position, we need to measure the distance from the cube to the base station and process the result in a control loop.
As our control loop is 1 ms, we need a fast measurement. Besides that, we also want a very precise measurement as one should not be able to see the cube vibrating in the air.
Classical methods to do a distance measurement, such as a PSD or an ultrasonic device, do not meet the requirements stated above as they are not accurate enough or their response time is too slow.
A method using a laser, a MEMS-mirror and two optical sensors does meet the requirements we want.

Principle

Figure 4.10 shows the configuration of the distance measurement.

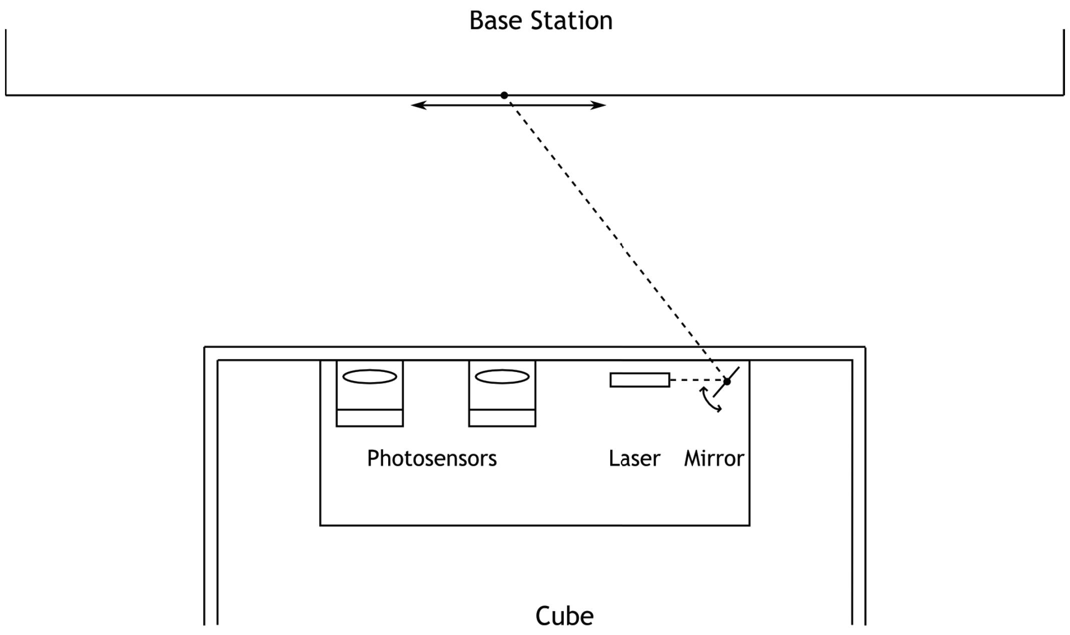


Figure 4.10: Configuration of the distance measurement

A laserdiode emits a laserbeam on a MEMS-mirror. This MEMS-mirror vibrates on its resonance frequency f_{res} and projects the laserbeam on the bottom side of the base station. During projection, the laserbeam makes a round-trip every $T_{res} = \frac{1}{f_{res}}$, but for the human eye this projection is detected as one line.
As one can see in figure 4.10, the right photosensor is the first to detect the laserbeam, followed by the left photosensor. These detections result in pulses generated by the photosensors and these pulses are first amplified and then processed by a comparator. A first order LPF is

placed before the inverting input of the comparator, to make sure we compare the original signal (non-inverting input) with the correct reference DC-voltage.
On the output terminal of the comparator we now have a 5 V pulse for every detection of the photosensor. This 5 V pulse is led to the MCU and is used as an interrupt for an internal timer.
Finally, the time that is passed by between two detections (i.e. interrupts) is related to the cube's distance to the base station: we made a distance measurement system using optical equipment combined with time measurement.

Operation

We now give more details about the different components mentioned in the section above and describe how the distance measurement works.

The laserdiode emits a red laserbeam with a wavelength of 630 nm. To stabilize the laser and to prevent overheating we use the *iC-WK chip* from *iC-Haus*[3] which is specially designed for this task.

The used MEMS-mirror is a very small mirror that vibrates at its resonance frequency $f_{res} = 3.626\,\text{kHz}$. This vibration is not a movement in the vertical direction, but a rotating movement around its horizontal axis. We use the term 'rotating *movement*' because the mirror never makes a 360° rotation. The mirror has the largest excitation when it works at its resonance frequency. The cube's movement along the vertical axis is within a range of 3 to 8 cm to the bottom side of the base station. In this range, both photosensors always have to detect the projected laserbeam and so we want the amplitude of the projected line as large as possible. Hence, we have to work at f_{res} because this results in the largest amplitude.
An H-brige is used to drive the mirror. This H-bridge is necessary to let the current flow through the mirror in two directions alternatingly, in every T_{res}. The switches of the H-bridge are controlled by the MCU: a square wave signal of $f_{res} = 3.626\,\text{kHz}$ is given to operate the H-bridge. One extra MOSFET is provided as inverter to assure a correct operation. For this application there are only two states necessary instead of four states to drive the motor: *left* and *right* are used.

To guarantee the correct operation of the mirror and to realize a sufficient amplitude, the current through the mirror should be in a range of 5 to 10 mA.
To obtain a current of 7 mA at 3.3 V, a total resistance of 471 Ω is needed. As the internal resistance of the mirror is 40 Ω and the resistance of the MOSFET is 6 Ω, a resistance of $471 - 46 = 425\,\Omega$ has to be added to the H-bridge. A resistance of 430 Ω is chosen.
Because the MEMS-mirror is a prototype, a feedback signal (MEMS_FB) is provided to implement a stabilisation algoritm (software). As our mirror is stable enough, we decide not to

implement this. This feedback signal could also be used to automaticaly find the resonance frequency.

Two photosensors - together with two lenses - are placed under the projected laserbeam. These lenses are necessary to minimize the effects of ambient light: the photosensors are only focused on the laserbeam. As photosensors we decide to take photodiodes, because these are faster than phototransistors.
The photodiode in a non-biased configuration is shown is figure 4.11. This configuration is preferable to a biased configuration because of the lower noise and higher sensitivity.

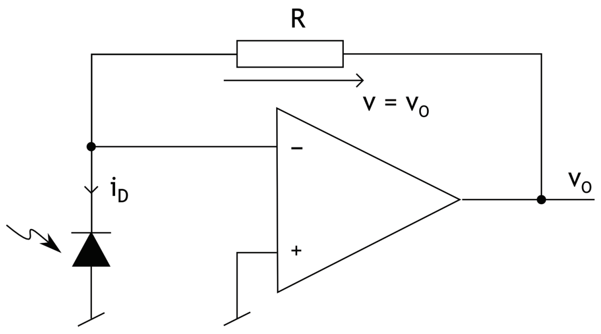


Figure 4.11: Photodiode in a non-biased configuration

The amplifier in figure 4.11 works as a current-to-voltage amplifier. This amplifier is a rail-to-rail op-amp supplied with 0 and 5 V. We choose this kind of op-amp because of the full output range (i.e. 0 to 5 V) we now have at our disposal.

The output of figure 4.11 goes to the inverting and non-inverting input of a comparator. On the inverting input, the signal is filtered by a first order LPF with cut-off frequency:

$$f_c = \frac{1}{2\pi \cdot R \cdot C} \tag{4.6}$$

A frequency $f_c = 800\,\text{Hz}$ is chosen as we want to filter out the pulses generated by the sensors (2 pulses per sensor every $T_{res} = \frac{1}{f_{res}} = \frac{1}{3.626\,\text{kHz}}$) and keep in the ambient light (e.g. fluorescent lighting of 100 Hz). After this filtering a DC-value is left, used to compare the original signal on the non-inverting terminal with. This is also shown in figures 4.12 and 4.13: channel 1 shows the non-inverting input (i.e. the original signal), channel 2 shows the filtered inverting input.

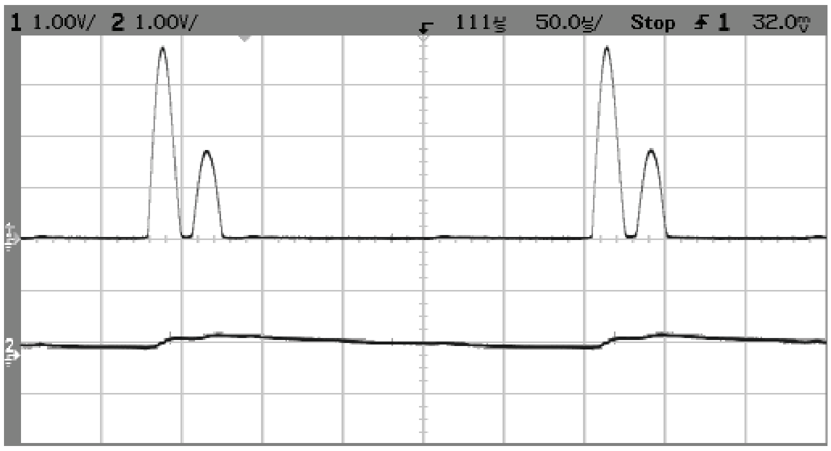


Figure 4.12: Inputs for the comparator: filtered and unfiltered sensor signal

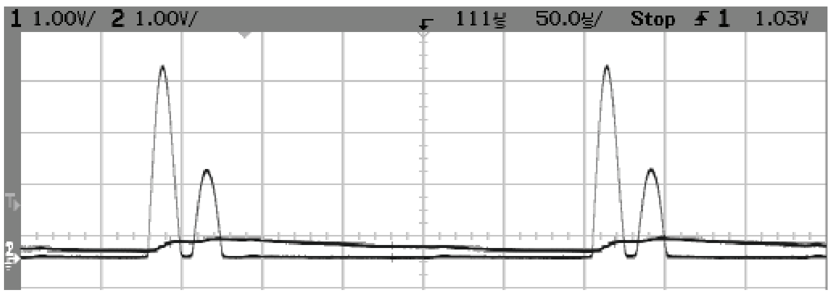


Figure 4.13: Inputs for the comparator: filtered and unfiltered sensor signal

Equation 4.6 is used to calculate the necessary resistance for a capacitor of 1 nF and a wanted cut-off frequency of 800 Hz:

$$\begin{aligned} R &= \frac{1}{2\pi \cdot f_c \cdot C} \\ &= \frac{1}{2\pi \cdot 800 \cdot 1 \cdot 10^{-9}} \\ &\approx 200\,\mathrm{k\Omega} \end{aligned}$$

The output terminal of the comparator is high (5 V) when the original signal (on the non-inverting input) raises above the filtered signal (i.e. the DC-value on the inverting input). This is shown in figure 4.14: channel 1 shows the output from the photodiode circuit of figure 4.11. Channel 2 shows the output terminal from the comparator, this signal goes to an interrupt pin of the MCU.

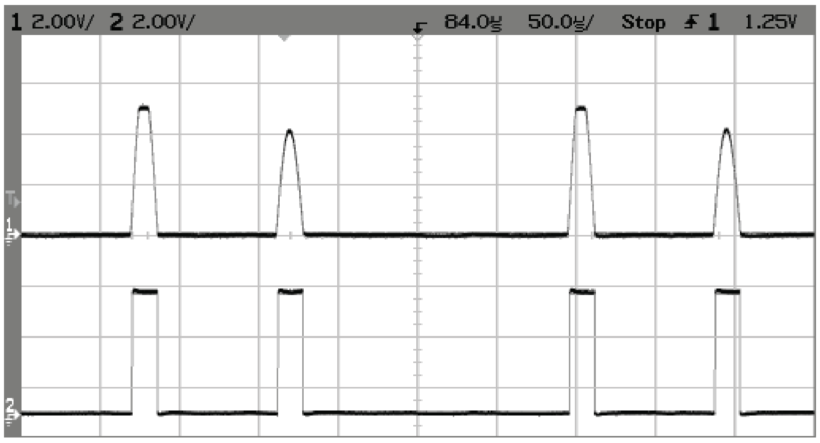


Figure 4.14: Sensor signal and comparator output

For each sensor, there is one interrupt pin available on the MCU. An internal timer is started when the first interrupt (from the first sensor) is detected and stopped when the second interrupt (from the second sensor) is detected. The time that is passed by between these two interrupts is related to the distance from the cube to the bottom of the base station. This relation is shown in figure 4.15

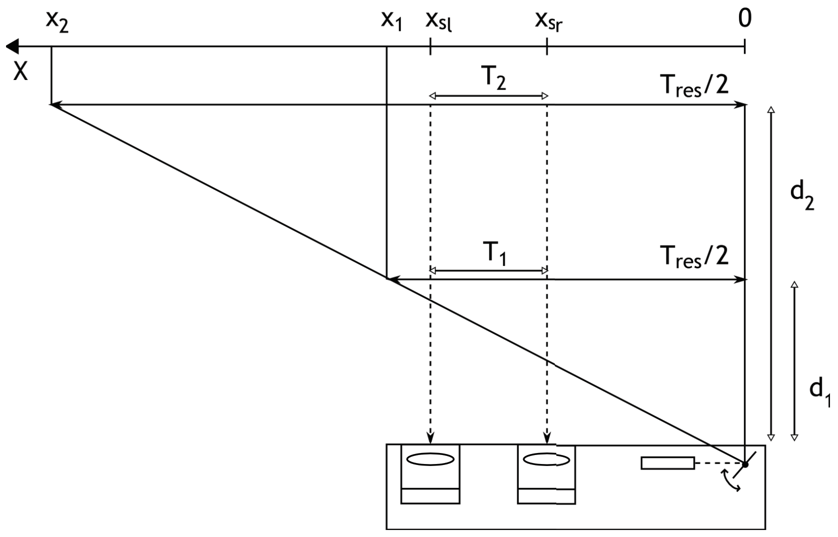


Figure 4.15: Relation time-distance

Situation 1: The cube is at a distance d_1 from the bottom of the base station. The laser beam travels between point $x = 0$ and $x = x_1$ and the time that is passed by between 0 and x_1 is $\frac{T_{res}}{2}$. The right photosensor detects the signal at $x = x_{s_r}$ and the left photosensor at $x = x_{s_l}$. The time that is passed by between these two detections is T_1.
Situation 2: The cube is at a distance d_2 from the bottom of the base station. The laser beam

travels between point $x = 0$ and $x = x_2$ and the time that is passed by between 0 and x_2 is $\frac{T_{res}}{2}$ (so the same time as in *Situation 1*). The right photosensor detects the signal at $x = x_{s_r}$ and the left photosensor at $x = x_{s_l}$. The time that is passed by between these two detections is T_2.

The relationship between these two situations is the following:

$$\frac{T_1}{\frac{T_{res}}{2}} > \frac{T_2}{\frac{T_{res}}{2}}$$

This means that $T_1 > T_2$ and that a larger distance results in a smaller elapsed time between two detections of the photodiodes.
Note that the distance measurement is not an absolute one. The relationship between the time measurement and the absolute distance to the ceiling is a complicated function. The described distance measurement method only indicates a positive or negative variation of the distance. The measured timing value is processed by the control loop.
Appendix A.4 explains the necessary operating frequency for the MCU to obtain a distance measurement resolution of 0.2 mm.

4.2.3 Charging and Power Supply

Power Induction

Figure 4.4 on page 13 gives the principle of power induction. The magnetic field that is sent out by the base station, contains magnetic energy that we want to convert to electrical energy again. So the receiving coil works as a transducer, converting magnetic to electrical energy.

As stated in section 4.1.2 it is important to work at resonance frequency so efficiency increases a lot. Hence, also inside the cube we have to build a resonance circuit that has the same f_{res} as the resonance circuit inside the base station, see (4.2) on page 14.
The quality factor Q of this series resonance circuit is given by (4.3). As we want the Q factor as high as possible, we want a large inductance and a small capacity. To construct the coil (i.e. the inductor), there are two possibilities: one is to print traces on the PCB, the other is to use copper wire. We choose to make a coil with copper wire, because the inductance of printed traces is not high enough.

When we construct a coil of 100 windings of small copper wire, we measure an inductance L = 1.53 mH. If we are working with a resonance frequency of approximately 20 kHz, we can now use (4.4) to calculate the necessary capacity to create resonance:

$$C = \frac{1}{(2\pi \cdot 20 \cdot 10^3\,\mathrm{Hz})^2 \cdot 1.53 \cdot 10^{-3}\,\mathrm{H}} = 41.39\,\mathrm{nF}$$

We decide to use one capacitor of 39 nF inside the cube (nearest value E12 series), this choice results in an increase of f_{res} (4.2):

$$f_{res} = \sqrt{\frac{1}{(2\pi)^2 \cdot 1.53 \cdot 10^{-3}\,\mathrm{H} \cdot 39 \cdot 10^{-9}\,\mathrm{F}}} = 20.604\,\mathrm{kHz}$$

This increase of f_{res} has also an influence on the resonance circuit inside the base station: a new capacity has to be calculated for the base station, in accordance with (4.4):

$$C = \frac{1}{(2\pi \cdot 20.604 \cdot 10^3\,\mathrm{Hz})^2 \cdot 3.79 \cdot 10^{-3}\,\mathrm{H}} = 15.74\,\mathrm{nF}$$

Appendix F includes a mechanical drawing of the coil frame placed inside the cube. Note that the used material is PVC, because an aluminium frame influences the inductance in a negative way.

To convert the AC signal of the power induction into a DC signal, we place a diode rectifier and a smoothing capacitor after the resonance circuit.

We also provide a 7.5 V (5 W) Zener diode to be sure that the induced EMF inside the cube never exceeds the maximum input voltage (i.e. 8 V) of the charger chip (see next paragraph).

Charger & LDO

An internal accu (battery) will provide the hardware with the necessary energy. Because the cube's mass is limited to 400 g, we want a lightweight accu that delivers a sufficient amount of energy.
A *Lithium Polymer Accu* is a good choice to do this task, as it combines a large energy density with a small mass.
Because the charging process of this kind of accu is rather critical, we provide the *LTC4063* charging chip from *Linear Technology* [2] to make sure this is done in a decent way.
Figure 4.16 shows a typical application of the LTC4063 (see datasheet).

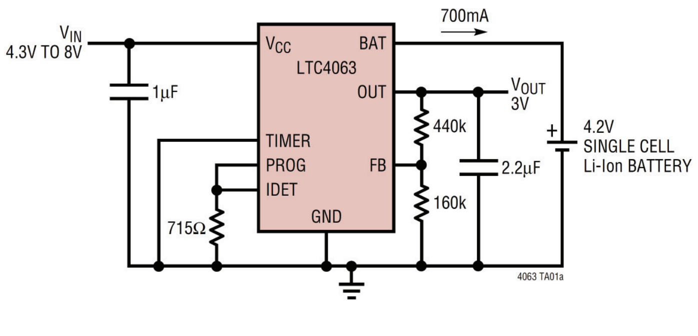


Figure 4.16: Configuration of the charger and LDO regulator

The LTC4063 has also an LDO regulator. This LDO regulates an output voltage between 1.2 V and 4.2 V at up to 100 mA load current. In our system, this output voltage is set to 3.3 V using external resistors. The choice of these two external resistors is given by the following formula (see datasheet):

$$V_{OUT} = 800\,\mathrm{mV} \cdot \left(1 + \frac{R_2}{R_1}\right)$$

In order to maintain stability under light load conditions, the maximum recommended value of R_1 is 160 kΩ. As we want an output voltage of 3.3 V, this results in an $R_2 = 500\,\mathrm{k\Omega}$.

The LTC4063 can terminate a charge cycle using several methods. When using the *C/10 Current Detection/Termination*, R_{PROG} is given by the following equation:

$$R_{PROG} = \frac{500\,\mathrm{V}}{I_{CHG}}$$

An I_{CHG} of 200 mA results in an R_{PROG} of 2.5 kΩ.

Step-up converter

Several modules and components in the system operate at 5 V. The accu has a nominal terminal voltage of 3.7 V, so we cannot connect the 5 V-components directly to the accu. Therefore, a step-up DC/DC converter is needed, capable of delivering 5 V from the lithium polymer battery. The *Linear Technology LTC3539-2 DC/DC converter*[1] is chosen to do this task. It has an efficiency of 80% and more, for load currents of resp. 10 mA and more.
A step-up conversion from a lithium polymer battery to 5 V is a common scenario, so the datasheet of this chip shows the correct circuit to do this (figure 4.17).

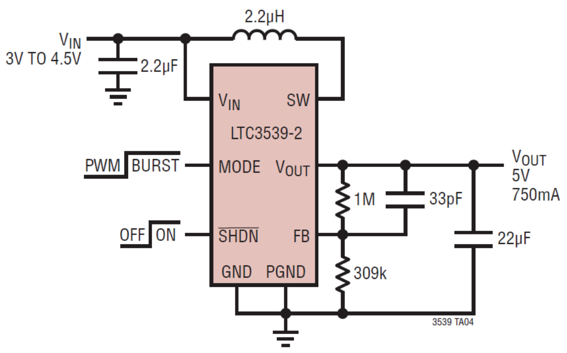


Figure 4.17: Configuration of the step-up converter for 3.7 V to 5 V operation

The output voltage is determined by the voltage divider resistors (R_1 and R_2) at the output of the chip (see datasheet):

$$\begin{aligned}
V_{OUT} &= 1.20\,\mathrm{V} \cdot \left(1 + \frac{R_2}{R_1}\right) \\
&= 1.20\,\mathrm{V} \cdot \left(1 + \frac{1 \cdot 10^6\,\Omega}{309 \cdot 10^3\,\Omega}\right) \\
&\approx 5\,\mathrm{V}
\end{aligned}$$

4.2.4 Power Consumption

All the hardware in the cube is in essence powered from the battery, so low power consumption is an important item here, to ensure long battery autonomy. The total current drawn from the battery terminals is measured when all modules are enabled.

$$\begin{aligned}
i_{\mathrm{bat}} &= 260\,\mathrm{mA} \\
V_{\mathrm{bat,nom}} &= 3.7\,\mathrm{V} \\
\Rightarrow P &= i_{\mathrm{bat}} \cdot V_{\mathrm{bat,nom}} = 0.96\,\mathrm{W}
\end{aligned}$$

Of the total current given above, 60 mA is used by the laser. The power consumed by the motor depends on the duty cycle of the PWM signal. The total current given above is when the duty cycle $\delta = 20\%$, the motor itself then uses 70 mA. When the motor works at $\delta = 100\%$, a total current up to 900 mA has been measured.
From this, we can conclude that the requirement of a power consumption less than 1 W (see chapter 2) is met, on condition that the motor does not need to turn at full speed too often. This is the case when the commanded speed of the cube does not change often.

4.3 Processing and Wireless Communication

4.3.1 Processing Unit

To deal with all the outgoing and incoming signals of the various components, the cube and the base station both need a manager, i.e. a processing unit.
This processing unit has to organize everything and spread the tasks which have to be done. There are a lot of candidates available on the market to do this job, thus a well-considered choice is necessary depending on all the requirements.

Requirements

Our MCU has to able to provide with the following items:

- Because everything inside the cube has to be low power, we need an MCU with *low power consumption.*
- The motor inside the cube and the levitation system inside the base station both need a *PWM* signal.
- The angular rate sensor its value will be processed with an *ADC* of at least 10 bits.
- To communicate with the wireless module a *SPI bus* is required. The maximum speed for the nRF24L01 wireless module is 8 Mbps.
- We need *four different timers* inside the cube: one to control the mirror, one for the photosensors, one for the PWM and one for the main control loop.
- *18 GPIOs* are necessary to handle several control signals.
- *Six interrupt entries* are necessary for the different interrupt signals (e.g. photosensors).
- To obtain a distance measurement resolution of 0.2 mm we need an operating frequency of at least *36.4* MHz (see appendix A.4).

PIC18F Family

The 8-bit *PIC18LF6585* from *Microchip* [4] is chosen as MCU because it can provide with all the requirements stated above.
To do any tests we can use a test board with a PIC18LF6680 on it, designed at HES-SO. The only difference with the chosen MCU is 48 kB program memory against 64 kB for the testing MCU.

Configuration

- To establish oscillation we connect to the OSC1 and OSC2 pins a crystal oscillator of 10 MHz. When we enable the PLL inside the PIC, this frequency is multiplied by 4, thus a frequency of 40 MHz is achieved. The PIC needs per instruction cycle 4 oscillator periods. So at a frequency of 40 MHz, the PIC can work at a maximum operation speed of 10 MIPS or 100 ns per instruction cycle.

- Because we need to work at a frequency of 40 MHz, the MCU has to work on a supply voltage of 5 V.

- Two input signals coming from the nRF24L01 wireless module have an output high voltage V_{OH} between $V_{DD} - 0.3\,\mathrm{V}$ and V_{DD}. As the nRF24L01 works on a V_{DD} of 3.3 V and the PIC on a V_{DD} of 5 V, we have to provide an adaption so these signals are correctly interpreted by the PIC. This adaption is done using twice two MOSFETs: they pull up the signals from the nRF24L01 to a correct high level for the PIC, i.e. 5 V. Note that this adaption is only necessary when the input terminal of the PIC has a 'Schmitt Trigger Buffer'.

- Not all input signals for the PIC are digital ones. When we want to use an input pin as analog one (e.g. ADC conversion), we have to make sure that the source impedance not exceeds 2.5 kΩ. Also for the voltage references ($V_{ref}+$ and $V_{ref}-$) of the ADC convertor, the source impedance must be less than 20 Ω. To make sure we are below these values, we provide an OPAMP for every analog input signal so the source impedance is approximately zero.

- Two input signals could exceed the maximum MCU input voltage of 5.5 V: the charging indication signal and the indication signal for power induction have a maximum voltage of 7.5 V (limited by the 5 W Zener diode). These signals are processed by two comparator OPAMPs LM393, one for each signal. A high (5 V) output voltage is given when the input signal drops below 4.3 V. This reference voltage is chosen because the LTC4063 charger IC starts charging from 4.3 V (see figure 4.16 on page 29).

- 5 LEDs are connected to the MCU and are used for testing and indicating events.

4.3.2 Wireless Communication

Since the cube has to send information and commands to the base station and vice versa, a wireless connection between these two is necessary. There are a lot of technologies available on the market to realize a wireless connection like Infrared, Bluetooth, WLAN, etc.
A detailed research of all these possibilities is not a goal of this Master thesis, though some important issues have to be taken into account when choosing a certain technology: low power consumption, a light protocol and an easy communication with an MCU.

In these subjects, the infotronics department achieved good results with the *nRF24L01* chip from *Nordic Semiconductor* [5]. This is a single chip tranceiver that works on the worldwide 2.4 GHz band, has low power consumption and makes an easy communication possible with an MCU using a SPI bus. An automatic packet handling feature is also included.
Taking all this into account, the nRF24L01 chip is a good choice for our application.

Specifications

The most important specifications of the nRF24L01 chip are the following:

- Worldwide 2.4 GHz band operation
- 1 and 2 Mbps air data rate
- Transmitter: 11.3 mA at 0 dBm output power
- Receiver: 12.3 mA at 2 Mbps
- 1.9 to 3.6 V supply range
- 22 µA Standby-I mode, 900 nA power down mode
- 4-pin hardware SPI
- Automatic packet handling with Enhanced ShockBurst^{TM}

Block Diagram

Figure 4.18 shows the block diagram of the nRF24L01.

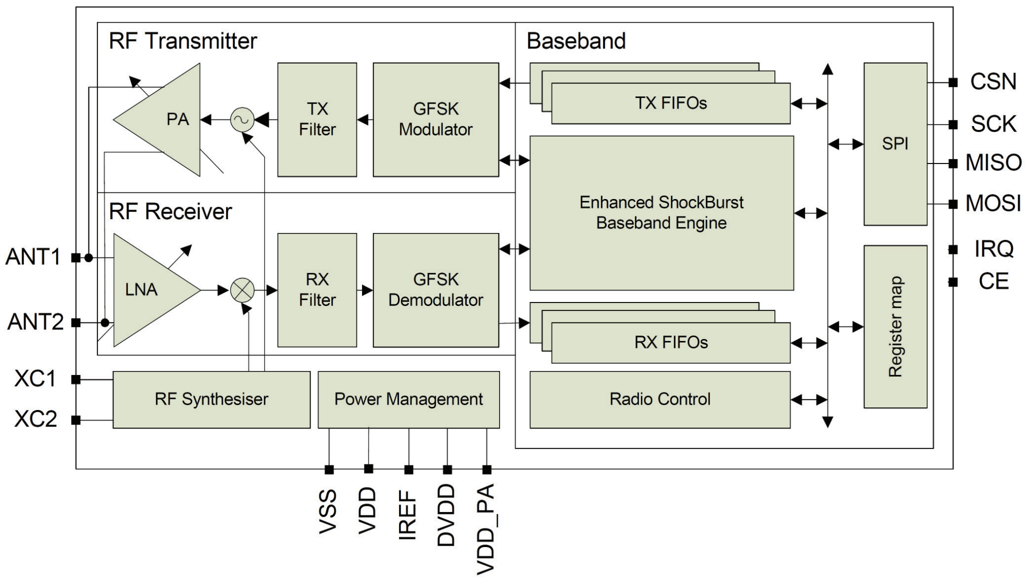


Figure 4.18: Block diagram of the nRF24L01

We discuss the most relevant parts for our application:

- There are 3 separate 32 bytes TX and RX FIFOs. These internal FIFOs ensure a smooth data flow between the radio front end and the system's MCU.
- The embedded baseband protocol engine (Enhanced ShockBurst^{TM}) is based on packet communication and supports various modes from manual operation to advanced autonomous protocol operation.
- The nRF24L01 is configured and operated through a SPI. Through this interface the register map is available.
- The register map contains all the configuration registers in the nRF24L01 and is accessible in all operation modes of the chip.
- The 6 pins on the right side form together the 'data and control interface' and gives access to all the features in the nRF24L01. These pins are 5 V tolerant digital signals.
  IRQ (Interrupt Request): this signal is active low and is controlled by three maskable interrupt sources.
  CE (Chip Enable): this signal is active high and is used to activate the chip in RX or TX mode.
  CSN (Chip Select Not): this SPI signal is active low and is used to indicate the start of a SPI operation.
  SCK (Serial Clock): this SPI signal is a clock formed by the master, i.e. the MCU.
  MOSI (Master Output, Slave Input): this SPI signal is the incoming data from the master.
  MISO (Master Input, Slave Output): this SPI signal is the outgoing data to the master.
  Section 5.3.1 'SPI' on page 49 in chapter 5 gives further information about these signals and how to use them.

Enhanced ShockBurst^{TM}

Incoming and outgoing data (both through the ether and SPI bus) are handled by the Enhanced ShockBurst^{TM} module. This module features automatic packet assembly and timing, automatic acknowledgement and re-transmissions of packets.
Enhanced ShockBurst^{TM} uses ShockBurst^{TM} for automatic packet handling and timing. During transmit, ShockBurst^{TM} assembles the packet and clocks the bits in the data packet into the transmitter for transmission.
During receive, ShockBurst^{TM} constantly searches for a valid address in the demodulated signal. When ShockBurst^{TM} finds a valid address, it processes the rest of the packet and validates it by CRC. If the packet is valid the payload is moved into the RX FIFO.

Schematic

Previously, a PCB is designed at HES-SO to do different tests with this chip. We can use this PCB as plug-in module on our PCBs, only an appriopate connection should be placed.
Appendix B includes the schematic of the plug-in PCB designed at HES-SO:
As one can see, only 9 pins of the nRF24L01 are connected with our PCB (connections J2 and J3).
Although the nRF24L01 normally works on a supply voltage of 3.3 V, we have to connect it with 5 V because this supply voltage is the only one available on the connectors.

4.4 Detailed System Architecture

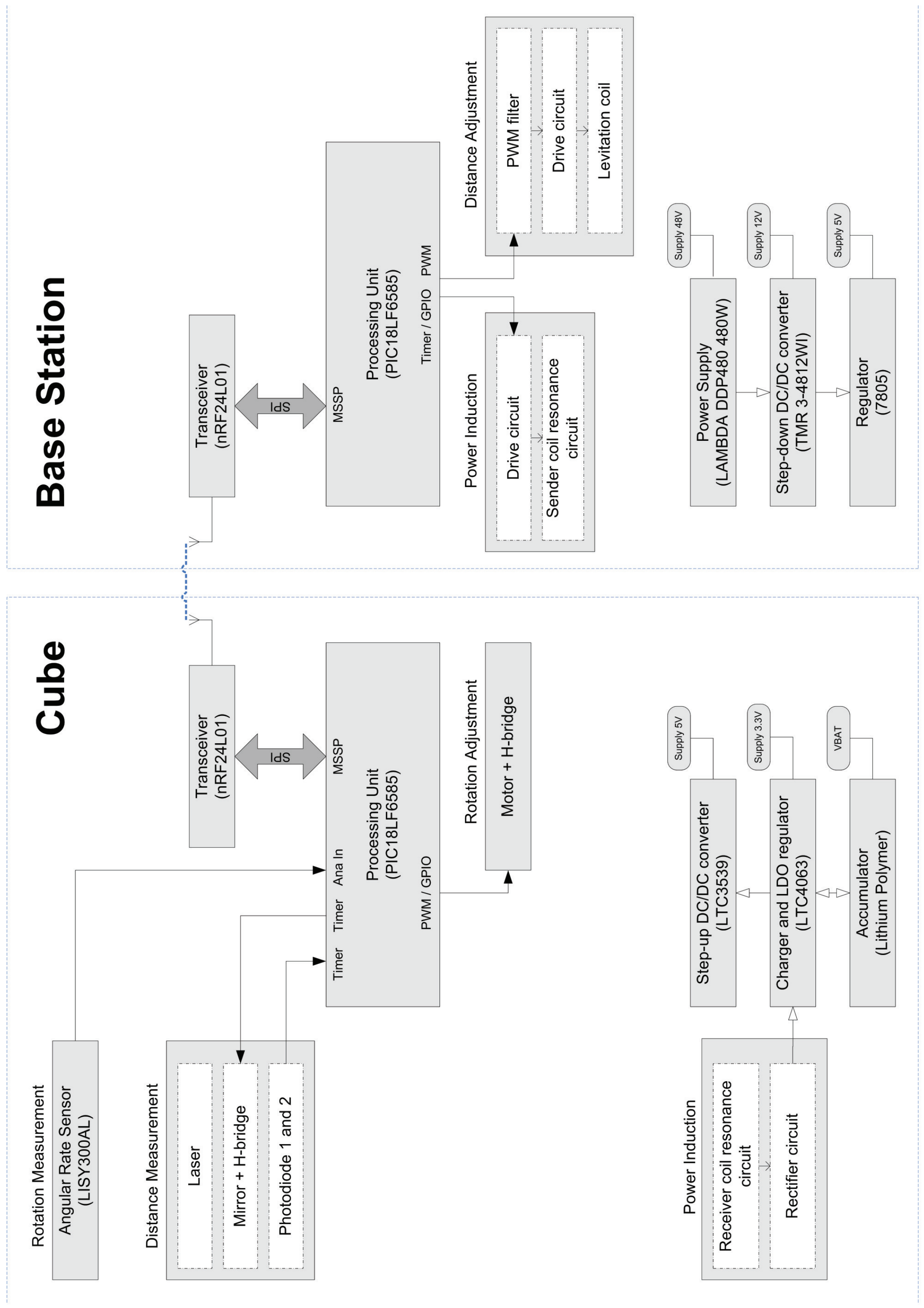


Figure 4.19: Detailed architecture of the system

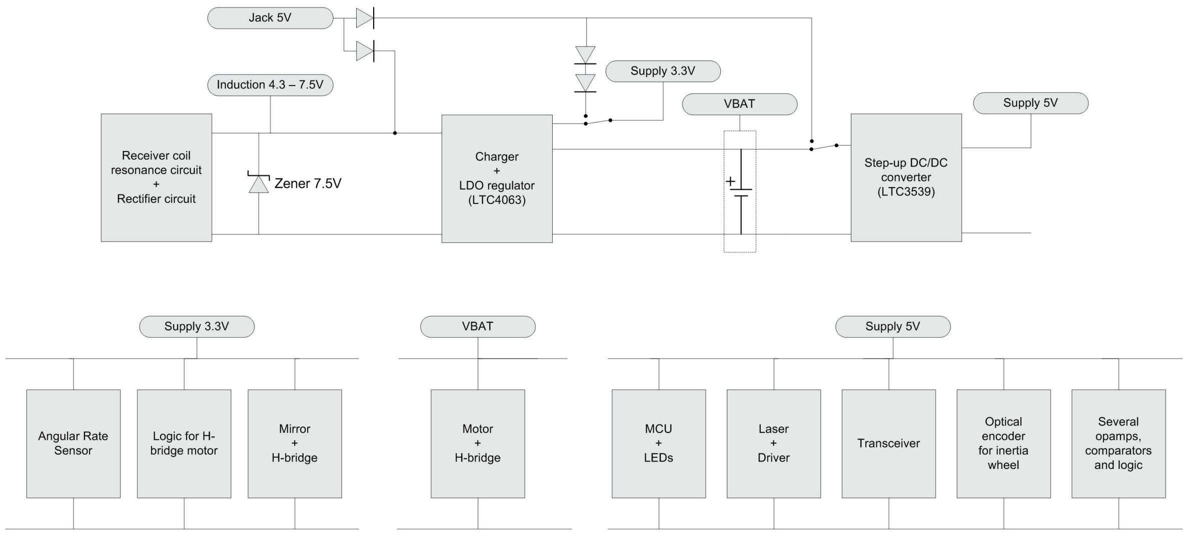


Figure 4.20: Alimentation in the cube

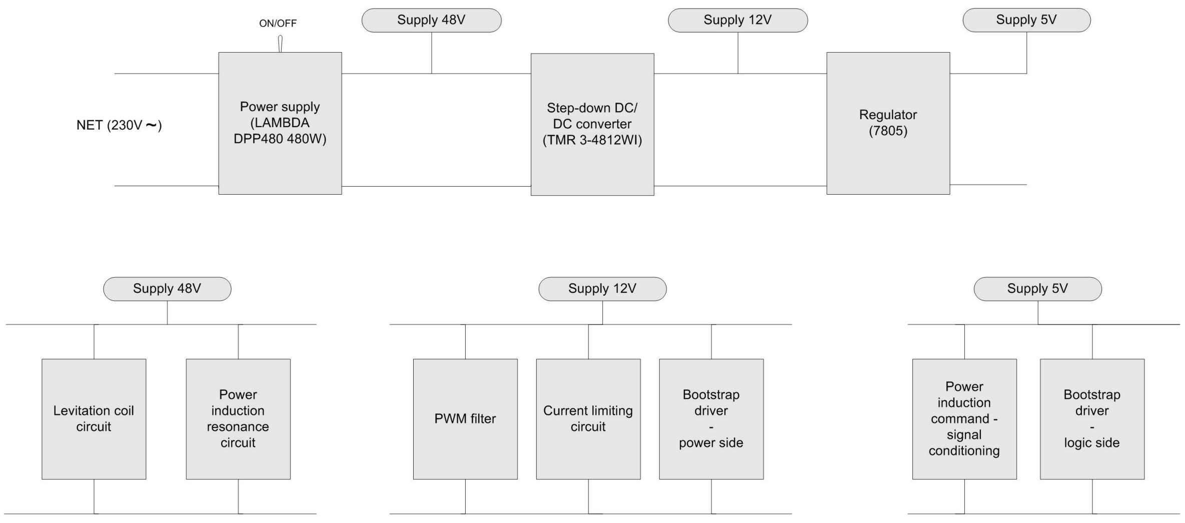

ON/OFF
Supply 48V
Supply 12V
Supply 5V
NET (230V ~)
Power supply (LAMBDA DPP480 480W)
Step-down DC/DC converter (TMR 3-4812WI)
Regulator (7805)
Supply 48V
Levitation coil circuit
Power induction resonance circuit
Supply 12V
PWM filter
Current limiting circuit
Bootstrap driver - power side
Supply 5V
Power induction command - signal conditioning
Bootstrap driver - logic side

Figure 4.21: Alimentation in the base station

4.5 PCB Design

This paragraph gives a brief discussion about the PCB design for the different PCBs inside the base station and the cube. The different PCB layouts can be found in appendix C.

4.5.1 Base Station

Inside the base station there are five PCBs:

1. One PCB operates as *user interface*. This PCB contains 5 *buttons* to give the 5 different commands as discussed in section 4.1.4. A connector is provided to lead the commands to the PCB that contains the MCU and wireless module.
2. A second PCB contains the *MCU* and the *wireless module*. This PCB has the same layout as the first PCB inside the cube. This decision is made to simplify PCB development.
   As stated above, this PCB is connected with the user interface PCB.
   A second connector is provided to connect this PCB with the PCB containing the drive circuit for the levitation.
   A third connector is present to plug in the wireless module (designed at HES-SO).
3. Next, we have the PCB containing the *drive circuit for the levitation*. Large traces are necessary on this PCB to withstand the large currents. For testing purposes, this PCB contains a BNC connector so a function generator can be connected.
4. The forth PCB contains the *power transistor* of the levitation drive circuit. This choice is made because of the heat that is generated inside this transistor. Hence, also a large *cooler* on this PCB is provided to cool down the power transistor.
5. The final PCB contains the *drive circuit for the power induction*. Large traces are necessary on this PCB to withstand the large currents.

4.5.2 Cube

Inside the cube there are three PCBs:

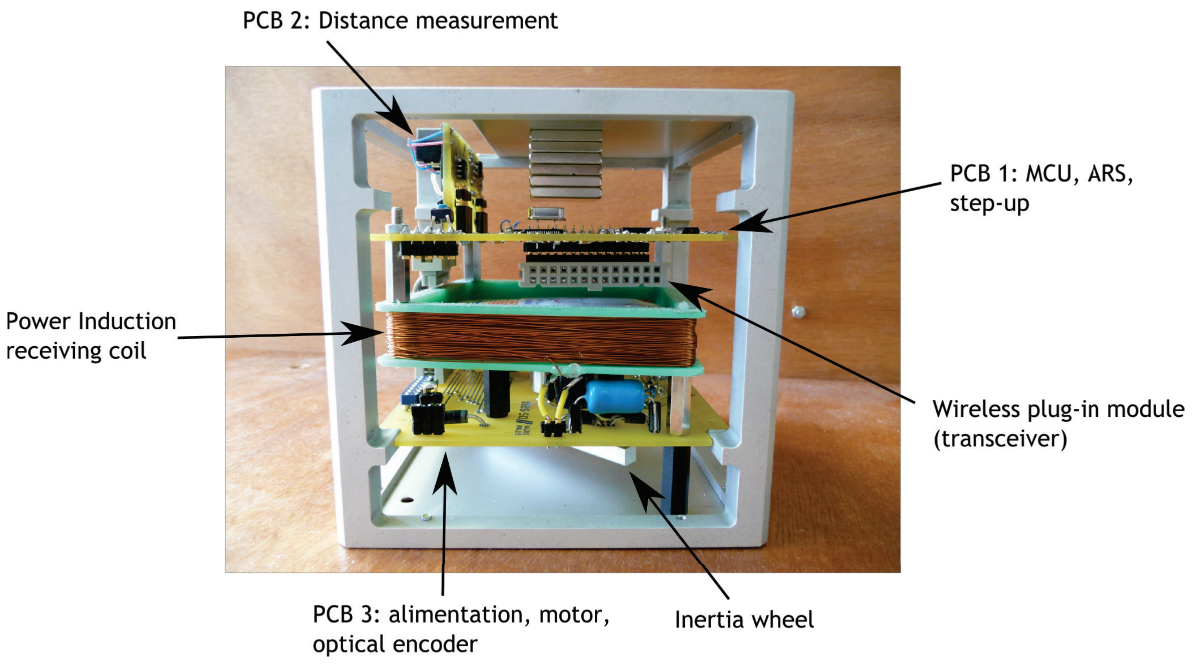
1. A first PCB contains mainly the *MCU*, the *wireless module*, the *angular rate sensor* and the *step-up converter*. This PCB has the same layout as the second PCB inside the base station (see previous section). This decision is made to simplify PCB development.
   A first connector is provided to connect with the third PCB of the cube. A second connector is used to plug in the second PCB containing the laser and mirror system. Finally, a third connector is present to plug in the wireless module (designed at HES-SO).

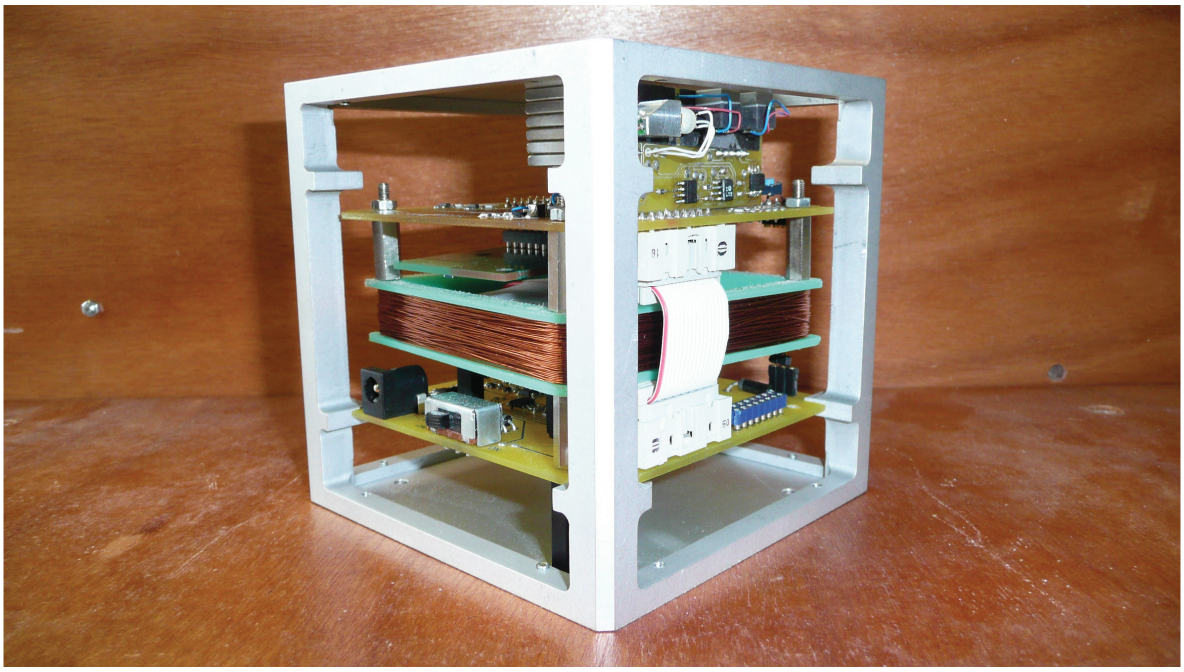
This first PCB is placed inside the cube, at a certain distance from the upper cover (see pictures below).

2. Next, we have the distance measurement PCB with principly the *laser*, *mirror* and *photodiodes* on it. This PCB is also designed as plug-in module and is placed vertical on the first PCB.

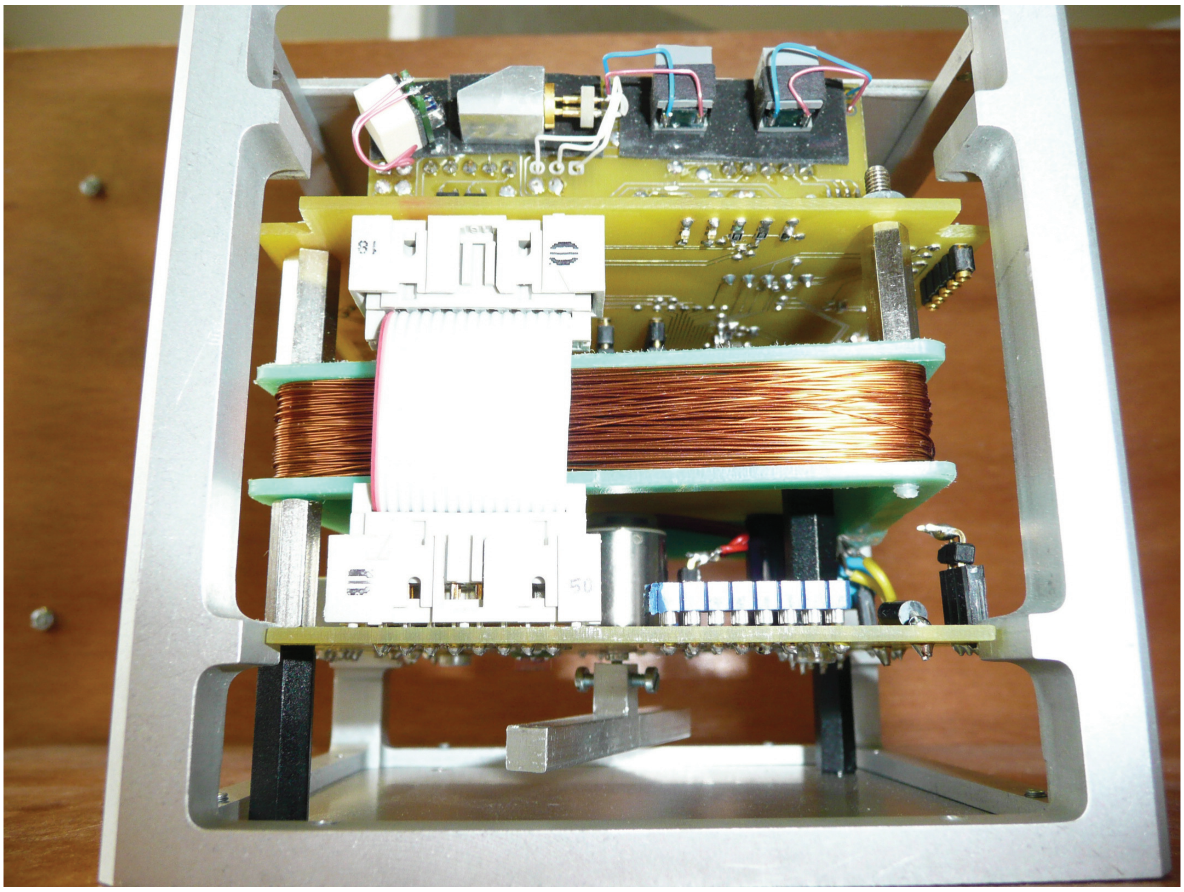
3. A third PCB contains in essence the *motor*, *charger*, *optical encoder* and *accumulator*. The motor is placed in the middle and on top of the PCB, with the inertia wheel mounted on the rotor under the PCB.
   The accumulator is placed on a mezzanine above the motor, so the middle point of the accu lies on the verical axis of the cube: this has a possitive influence on the equilibrium of the cube. A connector is provided to connect this PCB with the first (upper) PCB. This third PCB is placed above the lower cover.
   The *power induction* circuit is also present on this PCB, but the receiving coil is placed between the first and third PCB.

The dimensions of this PCBs are much more critical than the ones inside the base station: everything should fit precisely in the cube of 1 l.
The following pictures show the cube with the different PCBs inside it.


PCB 2: Distance measurement
PCB 1: MCU, ARS, step-up
Power Induction receiving coil
Wireless plug-in module (transceiver)
PCB 3: alimentation, motor, optical encoder
Inertia wheel



18
50



18
50

Chapter 5

Software

The processing unit for this system is the PIC18LF6585 microcontroller. This chapter describes and discusses the software that is written for both the base station and the cube.
The source code is included in appendix E.
In the source code, the workname for the base station that is used during the project, namely 'CAGE'*, is used.*

5.1 Base Station

The software in the base station takes care of the following items:

- Startup
- Synchronisation
- Levitation (distance adjustment)
- Power induction
- User interface (buttons)

When the `main()` function of the program starts, it calls some startup routines, and then enters a continuous loop of data processing. All of the items listed above, except the *startup* routine, are part of the continuous loop.
As section 5.2 will explain, the cube measures its distance to the ceiling, uses this measurement for some calculations, and sends the result (the desired PWM duty cycle, corresponding with the desired strength of the magnetic field) to the base station. The cube also informs the base station whether the power induction should be turned *on* or *off*. This set of data is used by the base station to perform its tasks.
When received data has been analysed and processed in the `analyseReceivedData()` function, the program goes to sendmode and collects the possible data to send to the cube. This

happens in the `setDataToSend()` function. Calling the `sendDataToCube()` function provides the outgoing package with the right addresses, and the data is sent. The data to be sent are the commands given by the user through the user interface (button board).

5.1.1 Startup

At startup, the MCU has a few tasks to do, prior to starting the continuous loop. These startup functions are called in the following order:

- `init()`: initialisation of several ports and peripheral modules. The general purpose I/O ports are set in the correct direction (input or output), and a default value is given to each output pin. The different modules, such as the *Capture/Compare/PWM module* are set up for correct operation. The relevant interrupts are enabled, and set to the desired priority level. Unused modules are disabled.
- `Init_NRF_RX()`: initialisation of the wireless transceiver as receiver. This function is discussed in section 5.3.2
- `start()`: the power induction timer and levitation PWM timer are activated. A blue LED indicates the working of the MCU.

After these startup functions, the *General Interrupt Enable High* (GIEH) and *General Interrupt Enable Low* (GIEL) bits are set, enabling all unmasked high and low priority interrupts, and the program start its main action.

5.1.2 Synchronisation

When the program enters the continuous loop, the transceiver is configured as receiver. At this point, it waits until a data package from the cube arrives (arrival of the data is flagged by the INT3IF bit). This data is analysed in the `analyseReceivedData()` function, and used to command the levitation PWM signal and the state of the power induction sending resonance circuit (*on* or *off*).
Then, the base station sends data to the cube, containing commands about the desired behaviour of the cube (see section 5.1.5). Obviously, the outgoing data array only contains a valid command if a button was pressed in this or the previous loop. As soon as the same INT3IF flag is set, confirming that the data is sent, the program returns to the beginning of the loop, where it waits for data to arrive.
This approach makes sure that the base station is synchronised with the cube, which runs on a control loop cycle of 1 ms. Note that the `main()` loop in the base station polls the INT3IF interrupt flag in an active way. The related interrupt is masked, and no softwareflags are used to exit the data arrival waiting loop. This means that the program in the base station only loops through the main routine when the cube is able to send data to the base station. If the

cube is not able to send data to the base station, the program in the base station is stuck in a waiting routine, but anyhow there would be no data to base its actions on.

5.1.3 Levitation

The data that is sent from the cube to the base station contains the result of the distance calculation. This calculation result is interpreted as the duty cycle of the levitation coil-driving PWM signal. As soon as this value is extracted from the incoming data array, the PWM signal (present on the LEVIT_PWM pin) is updated with the correct duty cycle.

5.1.4 Power Induction

When the MCU is resetted, power induction is enabled (so that the cube can sense proximity to the base station). This means that a square wave of fixed and precise frequency (resonance frequency f_{res} is sent to the power induction board. This square wave signal is generated using a timer (*Timer0*), whose overflow interrupt is set to high priority. At every overflow, the PI_PWM BIT IS TOGGLED, and the timer count that corresponds to f_{res} is resetted.
When analysing the data received from the cube, the program looks for a command about the power induction. If the byte at the correct index of the incoming array commands *power induction on or off*, the used timer is enabled or disabled.

5.1.5 User Interface

The base station is provided with a user interface, a board with five buttons on it (see section 4.1.4). This board is connected with the MCU board of the base station: each button is connected to an interrupt pin of the MCU. For each of these pins, an internal weak pull-up is enabled. Since pressing a button overrides the weak pull-up with a stronger logic low, an interrupt will occur (falling edge). In the ISR, these interrupts are served and the relevant softwareflags are set.
In the `setDataToSend()` function, which collects the data that has to be sent to the cube, these softwareflags are checked to determine if and which button has been pressed. Using a `bounceCounter` variable inhibits button presses for 300 ms after a previous press. By doing this, the bounce of the buttons can not be falsely interpreted as multiple presses.

5.2 Cube

The software in the cube takes care of the following items:

- Startup
- A control loop cycle of 1 ms
- Synchronisation
- Distance measurement
- Rotation measurement and adjustment
- Charging control

The program starts with some initialisations and user indications, in the startup routines. It then enters an continuous control loop, which is restarted every 1 ms. This loop takes care of the following sequence:

- Configure the transceiver as sender.
- Collect data regarding battery state, rotational speed, and distance to the ceiling. When data about the current rotational speed is collected, the adjustment part of the horizontal regulation system, is also done inside the cube.
- Send data to the base station, regarding the desired state of the power induction (*on* or *off*), and the desired strength of the levitation magnetic field (the adjustment part of the vertical regulation system).
- Configure the transceiver as receiver.
- Wait the remaining control loop cycle time for data possibly arriving.

5.2.1 Startup

At startup, the MCU has a few tasks to do, prior to starting the continuous control loop. These startup functions are called in the following order:

- `init()`: initialisation of several ports and peripheral modules. The general purpose I/O ports are set in the correct direction (input or output), and a default value is given to each output pin. The different modules, such as the *Capture/Compare/PWM module* are set up for correct operation. The relevant interrupts are enabled, and set to the desired priority level. Unused modules are disabled.
- `Init_NRF_TX()`: initialisation of the wireless transceiver as sender. This function is discussed in section 5.3.2

- `batteryIndication()`: using the internal comparator and voltage reference module of the MCU, the voltage over the battery is checked. When the charge of the battery is sufficiently high, a blue LED blinks a few times. If not, a red LED blinks. Between activation of the comparator and voltage reference module, and the actual use of it, an `wait(delay)` function is called, making sure that the modules have time to stabilise.
- `setARSZero()`: since it is possible that the analog voltage, given by the angular rate sensor when the cube is not rotating, differs from the expected center voltage of 1.65 V, the exact binary word (given by the ADC of the MCU) that corresponds with a rotational speed of 0°/s needs to be determined. This could be done by watching the ADC output in debugger mode, but by calling this function at startup, the zero point of the ARS is determined automatically. Note that the cube needs to be stable (not rotating) at the time this function is called, for the function the set the variable `ARSZero` at a correct value.
- `waitForPositioning()`: the program waits (while blinking an orange LED) until the rectified power induction voltage is detected. By doing this, the modules in the cube are only activated as soon as it is brought near the base station.
- `start()`: activation of the the laser, the mirror, angular rate sensor, WM module for the motor, . . . A blue LED indicates the working of the MCU.

After these startup functions, the *General Interrupt Enable High* (GIEH) and *General Interrupt Enable Low* (GIEL) bits are set, enabling all unmasked high and low priority interrupts. The timer that takes care of the 1 ms control loop is enabled, and the program start its main action.

5.2.2 Synchronisation

Within the 1 ms control loop, the cube acts as sender and receiver alternatingly. After collecting the correct data in the `powerCheck()`, `rotation()` and `distance()` function, this data is sent. The program then waits for data possibly arriving. The control loop timer has priority over the communication services: when the 1 ms is over, the program is forced to restart the control loop. Since the calculations in the cube only take 30% of the control loop time, the collected data can be sent every cycle.
By sending data every cycle, the cube makes sure that the base station can be synchronised with it: the program in the base station will take over the 1 ms loop by waiting for the data to arrive.
Since both communication points are synchronised, and the base station goes in send mode when the cube goes to receive mode, the cube can receive commands from the base station - if a valid command was given by the user.

5.2.3 Rotation

The motor and H-bridge, together with the angular rate sensor are 2 of the 3 major elements of a control loop. The 3rd element, the controller, is implemented in software. The MCU calculates the error between the wanted rotational speed of the cube and the measured rotational speed, and uses this value in a P-controller. This proportional controller calculates the excitation signal to be sent to the motor, in order to adjust its speed when necessary. This will keep the cube rotating at a constant speed.
The analog signal given by the ARS, corresponding to ω_c, has to be converted into an equivalent digital value that can be compared to the `wantedSpeed` variable. This is done by the internal ADC of the MCU.
The excitation signal for the motor is a PWM signal. By changing the duty cycle δ of this signal, the speed of the motor can be adjusted. Two bits represent the state of the motor: *turn left*, *turn right*, *run free* or *brake*.

5.2.4 Distance Measurement

The distance between the cube and the bottom of the base station is measured by looking at the time between the pulses of the two photodiodes, as explained in section 4.2.2. The output of the comparator after each photodiodes is connected with an interrupt pin of the MCU. The interrupt from the first photodiode (the right one in figure 4.10 on page 23) starts a timer. When the interrupt from the second photodiode arrives, the count of the timer at that moment is saved.
Since the hardware for the distance measurement needs optimalisation, the calculation of, and the controller for the distance have not been implemented in the software. For the controller, a PD-controller is proposed. The differential term of the controller reacts on fast changes in the distance, for example when the cube is suddenly going down, which could cause it to fall.

5.2.5 Charging

Since the charge of the battery is a slowly changing variable, there is no need to check it every millisecond. It suffices to verify the state of the battery once a minute. Therefore, a variable `powerCheckCounter` is introduced, counting the number of control loop cycles that have passed. Every time this 16-bit variable reaches the set `CHECK_POWER` value, the VBAT_DIV signal is compared with an internal reference value. Note that the the program starts up the comparator and voltage reference module in the control loop cycle at `counterPreciezeNaam = CHECK_POWER - 1`. This is necessary because the comparator and voltage reference module both need 10 µs settling time before the outputs are guarenteed to be valid. This approach is more efficient than enabling the modules and waiting (active) for the 10 µs to pass.
When the battery voltage is below the reference voltage, the following steps are taken: the MCU checks if power induction is enabled. If so, and the charging indicates that the charge

cycle has been completed, power induction can be switched off. If not, and the battery charge is low, power induction should be switched on.

5.3 Wireless Communication

5.3.1 SPI

As already seen in section 4.3.2 on page 33 we communicate with the wireless module using a SPI bus.
This synchronous serial bus gives us the possibility to send data to the nRF24L01, to program this chip and to read data from it.
In general, the SPI bus uses four signals to realize its communication:

- MOSI (Master Output, Slave Input): the incoming data from the slave.
- MISO (Master Input, Slave Output): the outgoing data to the slave.
- SCK (Serial Clock): the clock formed by the master.
- CS (Chip Select): used to select the slave.

The acronyms above make clear that the bus works in a master-slave mode. This means that one device is the master and the other is the slave. In our application the PIC MCU is the master, this master is responsible for the clock signal. Hence, the nRF24L01 is the slave.

Figure 5.1 shows the SPI configuration in our system. Note that the PIC uses different names for the MOSI and MISO signals, i.e. SDO (Serial Data Out) and SDI (Serial Data In). CS is replaced by CSN, which makes it an active low signal.

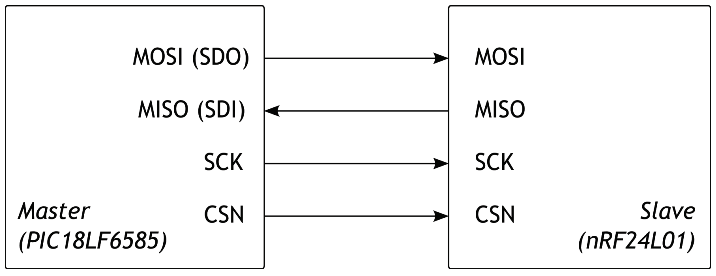


Figure 5.1: SPI configuration

Configuration

The SPI module is configured on the PIC MCU using the MSSP module. There are two registers available: SSPCON1 (Control Register 1) and SSPSTAT (Status Register). These registers are written to during initialisation (`init()`) of the main program.
To enable the serial port, the SSPEN must be set: this configures the SDI, SDO and SCK as serial port pins. CSN is a GPIO controlled by software. Their data directions are pogrammed as follows:

- SDI is automatically controlled by the SPI module.
- SDO as output.
- SCK as output.
- CSN as output.

The clock for the SPI operation (bit 3-0 SSPCON1) is set up as $F_{OSC}/16$: as the maximum SPI data rate of the nRF24L01 is 8 Mbps and the PIC works on a frequency of $F_{OSC} = 40\,\text{MHz}$, we decide to set up the clock as $\frac{40\,\text{MHz}}{16} = 2.5\,\text{MHz}$. This corresponds to a data rate of 2.5 Mbps.

Operation

In general, a SPI operation is started with the configuration of the clock by the master. The master then pulls the chip select (CSN) low for the desired chip, this is the indication for the slave that a SPI operation is started.
During each SPI clock cycle, a full duplex data transmission occurs:

- The master sends a bit on the MOSI line, the slave reads it from that same line.
- The slave sends a bit on the MISO line, the master reads it from that same line.

To make such an operation is possible on the PIC, the MSSP module consists of two registers: SSPBUF (Serial Receive/Transmit Buffer Register) and SSPSR (Shift Register). The last register is not directly accessible. SSPSR is the shift register used for shifting data in or out the PIC. SSPBUF is the buffer register to which data bytes are written to or read from.

Once the SPI module is well configured, the following actions take place to send a byte:

1. The program makes the CSN signal low.
2. The program places the data byte in the SSPBUF register (the data is automatically shifted to the SSPSR register).
3. The program waits until the SSPBUF is empty, i.e the SSPIF flag is set.
4. The program clears the SSPIF flag.
5. The program makes the CSN signal high again.

If there are x bytes to send, actions 2 to 4 have to be repeated x times.

The following actions take place to receive a byte:

1. The program makes the CSN signal low.

2. The program places a dummy byte in the SSPBUF register.
3. The program waits until the SSPBUF is full, i.e the BF flag is set.
4. The program places the incoming data from the SSPBUF register in a variable.
5. The program makes the CSN signal high again.

If there are y bytes to receive, actions 2 to 4 have to be repeated y times.

5.3.2 Wireless Module

We now take a closer look at the configuration and operation of the nRF24L01.

Configuration

We configure and control the nRF24L01 by accessing the register map through the SPI by using read and write commands.
The four most important commands are the following:

- R_REGISTER (binary: 000bbbbb). To read a register, bbbbb = 5 bit Register Map Address.
- W_REGISTER (binary: 001bbbbb). To write to a register, bbbbb = 5 bit Register Map Address.
- R_RX_PAYLOAD (binary: 01100001). To read the RX-payload, used in RX mode. The payload is deleted from FIFO after it is read.
- W_TX_PAYLOAD (binary: 10100000). To write the TX-payload, used in TX mode.

During initialisation (`init_NRF_TX()` or `init_NRF_RX()`) of the nRF24L01 we configure several registers:

- CONFIG. Configuration Register. The data received (RX_DR) and data transmitted (TX_DS) interrupts are not masked: they will be reflected on the IRQ pin. 2 bytes CRC is enabled. To enter stand-by mode, the PWR_UP bit is set high. The PRIM_RX bit is set high or low depending on the transceiver pogrammed as a receiver or transmitter.
- EN_AA. Enable 'Auto Acknowledgment' Function. This function is disabled because of possible timing problems during the 1 ms control loop. When this function is enabled, the nRF24L01 only gives a TX_DS interrupt when it receives an acknowledgment of the receiver. We want the nRF24L01 to go in send mode as well as in receive mode during the 1 ms control loop: this is not guaranteed when EN_AA is on.
- EN_RXADDR. Enable RX Addresses. Data pipe 0 is enabled.

- SETUP_AW. Setup Address Width. A 5 bytes address width is chosen.
- SETUP_RETR. Setup of Automatic Retransmission. Retransmission is disabled because of possible timing problems during the 1 ms control loop (see EN_AA).
- RF_CH. RF Channel. The RF channel is set to 2410 MHz.
- RF_SETUP. RF Setup Register. An air data rate of 1 Mbps is chosen. The RF output power in TX mode is set to 0 dBm so a larger distance can be reached.

The header file `define_communications.h` is implemented to link (define) the names to the corresponding hex value.

Operation

As seen in paragraph 5.2, the transceiver will be first configured as tranmitter and then as receiver during the 1 ms control loop.
We now discuss the operation of the transceiver as transmitter or receiver.

As a transmitter:

- To send data from the cube to the base station and vice versa, the transceiver must be configured as transmitter. This is done by clearing the PRIM_RX bit in the CONFIG Register.
- All the data (information about Levitation, Motor Speed and Direction, Power Induction) that has to be sent, is placed in an array (`NRF_output_array`).
- When all the data is ready to be sent, the function `SendData()` is called in the main program.
- Inside the function `SendData()` a function `NRF_TxData()` is called. This function has the `NRF_output_array` as argument and this `NRF_output_array` will be placed in the SSPBUF as described in section 5.3.1 (using first the W_TX_PAYLOAD command).
- Next, the NRF_CE signal is set high for at least 10 µs. This is the signal for the nRF24L01 to send the packet with all the data (payload) in it. This packet (see paragraph 5.3.3) is automatically composed by the nRF24L01.
- When the packet is sent, the transceiver sets the TX_DS interrupt flag high. This interrupt flag is cleared by writing a 1 to the TX_DS bit of the STATUS Register.

As a receiver:

- To receive data from the cube or the base station and vice versa, the transceiver must be configured as receiver. This is done by setting the PRIM_RX bit in the CONFIG Register.
- When in receiving mode, we make the NRF_CE signal high: this is the signal for the nRF24L01 to start listening for packets.
- If a packet is detected, the transceiver sets the RX_DR interrupt flag high. When this interrupt occures, the function `AnalyseReceivedData()` is called in the main program.
- Inside the function `AnalyseReceivedData()` first the NRF_CE signal is made low so the nRF24L01 stops listening for packets.
- Next, a function `NRF_RxData()` is called. This function has an array (`NRF_input_array`) as argument and the incoming data from the SSPBUF will be placed in the `NRF_input_array` as described in section 5.3.1 (using first the R_RX_PAYLOAD command).
- Now all the data (information about Levitation, Motor Speed and Direction, Power Induction) can be read from `NRF_input_array` array.
- Finally, the interrupt flag RX_DR is cleared by writing a 1 to the RX_DR bit of the STATUS Register.

5.3.3 Protocol

Packet

Figure 5.2 shows an Enhanced ShockBurst^{TM} packet with payload (0-32 bytes).

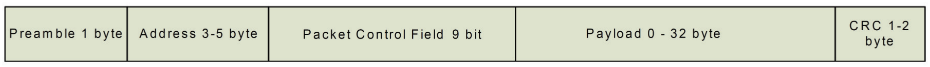


Figure 5.2: An Enhanced ShockBurst^{TM} packet with payload (0-32 bytes)

Preamble The preamble is a bit sequence used to detect 0 and 1 levels in the receiver.
Address This is the address for the receiver, we use a 5 bytes address width. This address ensures that the correct packets are detected by the receiver.
Packet Control Field The packet control field contains a 6 bit payload length field, a 2 bit PID (Packet Identity) field and a 1 bit NO_ACK flag.
Payload The payload is 5 bytes wide and is transmitted on-air as it is uploaded (unmodified) to the device. We implement our own high-level protocol, integrated in this payload.
CRC The CRC is the error detection mechanism in the packet. It is 2 bytes and is calculated over the address, Packet Control Field, and Payload.

High Level Protocol

As seen in previous sections, the payload exists of 5 bytes. This is divided as follow:

1. The first byte contains the Source Address.
2. The second byte contains the Destination Address.
   (We decide to give the cube address 0xB0 and the base station address 0xB5)
3. The third and forth byte contain information about the Levitation. As the PWM resolution is 10 bits, only the 2 LSbs of the forth byte are used.
4. The fifth byte contains information about the Power Induction or Motor Speed and Direction. This depends on what the destination is: if the base station is the destination, this byte is used for Power Induction. If the cube is the destination, this byte is used for Motor Speed and Direction.

5.3.4 Synchronisation

The implemented code makes a synchronisation possible between the cube and other communication points like the base station.
The wireless module inside the cube starts in transmit mode and sends out its data. It will go in receiver mode when the data is sent out (interrupt TX_DS occurred). After a 1 ms, it goes in transmit mode again no mather if it received something or not. This cycle is done every 1 ms (time of control loop).
The wireless module inside the base station starts in receive mode and checks for packets. It will only go in transmit mode when data is received (interrupt RX_DR occurred). Once the data is transmitted (interrupt TX_DS occurred) the base station goes in receive mode again.

When the base station is turned on and the cube is still off, it will wait to start sending out until it received data. Once the cube is turned on, the base station will receive data and this is the signal to send data back to the cube.
On the other hand, when the cube is turned on and the base station is still off, the cube will always send out data. This means the cube will also go in receiver mode during its 1 ms control loop. Once the base station is turned on, the cube receives data from the base station. So when they are both turned on, a synchronous communication is started where the cube is the master and the base station is the slave.

Figure 5.3 shows the synchronisation between the cube and the base station. Channel 1 shows the status of the cube, channel 2 shows the status of the base station. For both channels, a high level indicates that the program is in send mode. A low level indicates receiving mode. It is clear that the base station enters send mode for a short period when the cube is done sending data. We can also see that the base station inherits the 1 ms loop of the cube.

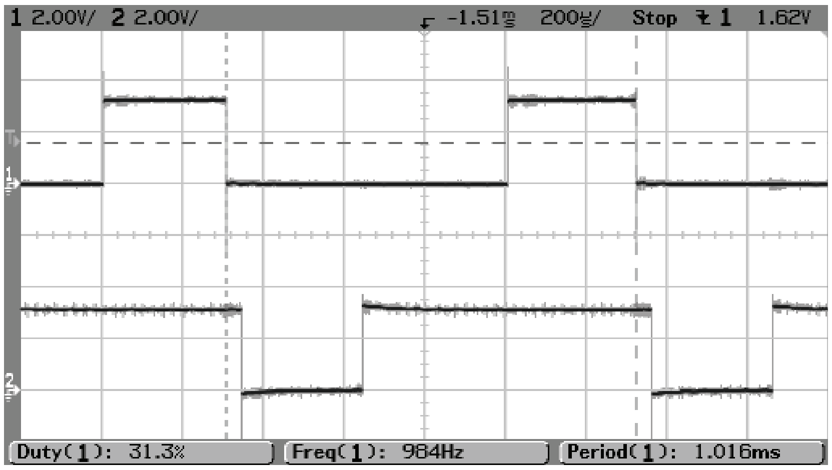


Figure 5.3: Synchronisation of the programs in the cube and base station

Chapter 6

Results and Specifications

In chapter 2 'Requirements' on page 6 all the requirements of our system are listed.
We now take a look at each of these requirements and describe what the results and specifications are that we obtained.

Specifications inherited from SwissCube:

- *The cube has a volume of 1 l.*
  The cube its volume is 1 l, but a small magnet is attached on top of the cube. This magnet could be placed inside the cube.
- *Power consumption in the cube does not exceed 1 W.*
  This requirement has been accomplished on the following condition: the cube's speed should not change to often, because then the motor asks a lot of power from the accu.

For the system to be stable, following requirements are postulated:

- *The cube must weigh 400 g or less.*
  The cube has a final weight of 475 g.
- *The behaviour of the cube is controlled by a 1 ms control loop.*
  We implemented a control loop of 1 ms. This control loop regulates the horizontal movement. Another software algorithm must be implemented to regulate the vertical movement of the cube. There should be more than enough time left from the 1 ms to do this, because there is only about 30% of the time occupied so far.
- *A distance measurement with a precision of 0.2 mm is needed.*
  A principle to do a distance measurement using optical equipment has been worked out. This is not yet implemented in the final system.

Other specifications of the system:

- *The cube can obtain a rotational speed of 8 rpm and more.*
  This has been accomplished: we achieve a rotational speed of 60 rpm.

- *The cube obtains this speed in 5 s at most.*
  A speed of rotation of approximately 10 rpm is obtained in 5 s.

- *The rotational speed is maintained with a precision of* $1^\circ/\mathrm{s}$.
  This has been accomplished: the angular rate sensor is capable of detecting the speed of the cube with the required precision.

- *The vertical position of the cube lies between 3 cm to 20 cm from the ceiling.*
  The vertical position lies between 3 cm and 8 cm. The maximum distance is reduced because of the dimensions and weight of the levitation coil.

- *Half-duplex wireless communication is used.*
  A wireless connection and communication between the cube and the base station has been established. Communication in both directions is possible.

- *The accu in the cube is charged on-line.*
  Power induction makes a transfer of necessary energy possible from the base station to the cube. Because this energy is transfered through the air, the cube can be charged while operating (floating).

Chapter 7

Conclusions and Future Work

When looking at the different aspects as stated in the Introduction on page 1, we can now formulate several conclusions about these aspects:

Power Consumption and Charging
The cube has a low power consumption and can be charged during operation (when floating in the air) using power induction. The accu autonomy has not been tested over longer periods yet. When the cube accelerates, the power consumption is at its maximum. Once the cube is turning at a constant speed, this power consumption is less.

Power induction is a good technique to provide the cube with energy. Unfortunately, the levitation coil has a negative influence on the efficiency of power induction. More research has to be done to improve this efficiency when the levitation coil is present.
The power consumption inside the base station is high, a lot of cooling is necessary. Although this power consumption is less important than the power consumption inside the cube, for safety reasons it would be an inprovement to reduce this power consumption.

Levitation (Magnetism)
The goal of changing the strength of a magnetic field using a digital signal has been reached. Depending on the duty cycle of the PWM signal given by the MCU this magnetic field changes. The cube can send a packet to the base station to change the duty cycle of the PWM signal. Note that this is an arbitray packet and is not the result of the control loop that regulates the vertical movement of the cube (see later).
The levitation module can be tested as stand-alone module when a function generator is connected.

Distance Measurement (Optics)
An optical system to do a distance measurement has been worked out, but not been finished

nor implemented. A better hardware design has to be done to prevent spikes and noise, so no other interrupt pulses will be detected by the internal timer of the MCU (results are better but not perfect when the room is darkened so ambient light is eliminated).
Although the MEMS-mirror is a prototype, the stability is good. A software algorithm can be implemented to improve the stability, but at first sight this is not obligated.
The operating frequency of the MCU is also a critical point because of the wanted resolution of the distance measurement.

Rotation
The cube rotates in the magnetic field using a small motor with an interia wheel mounted on the rotor. Information about the speed and direction of the cube can be collected, using the angular rate sensor. The optical encoder can be used to do tests and to collect information about the speed of the inertia wheel.
An angle measurement of the cube has not been implemented, because this seemed not necessary to control its rotational movement.
More detailed research can be done on the behaviour of the cube when the frame of the cube is closed. In that case, the air current will have another influence on the speed and behaviour of the cube than when its frame is open.

Regulation
The regulation of the rotational movement of the cube is succeeded. The motor will slow down, speed up or turn direction depending on the information given by the angular rate sensor. This information is processed by a control loop of 1 ms, and this control loop regulates the motor.
A software algorithm to do the regulation of the vertical movement must be implemented. Because the distance measurement was not accomplished, it was impossible to implement the controller that manages this regulation. There is more than enough time left from the 1 ms to do the implementation, because there is only about 30% of the time occupied so far.

Wireless Communication
A wireless connection is set up between the cube and the base station. They can both send and receive information or commands about the power induction, levitation and speed of rotation during the 1 ms control loop. The cube and the base station are also synchronised by means of this wireless connection. The wireless module can be easily plugged in or out.
The wireless connection can be improved by ensuring that every packet arrives and this over greater distances.

Appendix A

Calculations

A.1 PWM Filter for Levitation

Figure A.1 shows the generalised form of the Sallen&Key topology (second order filter), in the Laplace domain.

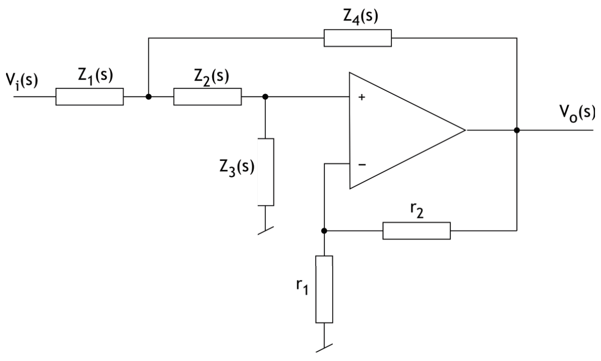


Figure A.1: Generalised form of the Sallen&Key topology

The TFF of this filter is:

$$H(s) = \frac{k}{\frac{Z_1 Z_2}{Z_3 Z_4} + \frac{Z_1}{Z_3} + \frac{Z_2}{Z_3} + \frac{Z_1}{Z_4}(1-k) + 1}$$

with $k = 1 + \frac{r_2}{r_1}$ the gain in the passband.

To implement a LPF, Z_1 and Z_2 are replaced by resistors (resp. R_1 and R_2), Z_3 and Z_4 by capacitors (resp. C_1 and C_2). Choosing $r_1 = \infty$ and $r_2 = 0$ makes it a unity-gain filter. The TFF becomes:

$$\begin{aligned} H(s) &= \frac{1}{R_1R_2C_1C_2 \cdot s^2 + R_1C_1 \cdot s + R_2C_1 \cdot s + 1} \\ &= \frac{\frac{1}{R_1R_2C_1C_2}}{s^2 + \left(\frac{1}{R_2C_2} + \frac{1}{R_1C_2}\right) \cdot s + \frac{1}{R_1R_2C_1C_2}} \end{aligned} \tag{A.1}$$

The general transfer function for a second order, unity gain LPF is:

$$H_{LPF}(s) = \frac{\omega_c^2}{s^2 + \frac{\omega_c}{Q} \cdot s + \omega_c^2} \tag{A.2}$$

When (A.1) and (A.2) are identified, we see the following relationships between the R & C components and the parameters of the LPF:

$$\begin{aligned} \omega_c^2 &= \frac{1}{R_1R_2C_1C_2} \\ \frac{\omega_c}{Q} &= \frac{1}{R_1C_2} + \frac{1}{R_2C_2} \end{aligned}$$

Or:

$$f_c = \frac{1}{2\pi} \cdot \sqrt{\frac{1}{R_1R_2C_1C_2}} \tag{A.3}$$

$$Q = \sqrt{\frac{R_1R_2C_2}{C_1}} \cdot \frac{1}{R_1 + R_2} \tag{A.4}$$

In order to simplify the calculations, we set the components as ratios:

$$\begin{aligned} R_1 &= mR \\ R_2 &= R \\ C_1 &= C \\ C_2 &= nC \end{aligned} \tag{A.5}$$

Equations (A.3) and (A.4) now become:

$$\begin{aligned} f_c &= \frac{1}{2\pi} \cdot \sqrt{\frac{1}{mn \cdot R^2C^2}} = \frac{1}{2\pi \cdot RC \cdot \sqrt{mn}} \\ Q &= \sqrt{\frac{mn \cdot R^2C}{C}} \cdot \frac{1}{mR + R} = \frac{\sqrt{mn}}{m+1} \end{aligned}$$

The calculation of the filter is now done by calculating m, n, R and C.
We want a filter with a low cut-off frequency f_c, since we are only interested in the DC-value

of the signal. However, the cut-off frequency should be high enough to make sure that the output of the filter has a sufficient fast response to fast changes in the duty cycle δ. A cut-off frequency of 1 kHz (frequency of the control loop) is chosen, on condition that the PWM frequency is 10 kHz or higher.
The Q-factor of the filter is less important here, but we choose to implement a Butterworth filter (having a maximal flat frequency response in the passband), this means that $Q = \frac{1}{\sqrt{2}}$. So:

$$f_c = 1\,\text{kHz}$$
$$Q = \frac{1}{\sqrt{2}}$$

A Q-factor of $\frac{1}{\sqrt{2}}$ can be obtained by choosing $m = 1$ and $n = 2$. Then f_c becomes:

$$\begin{aligned} f_c &= \frac{1}{2\pi \cdot RC \cdot \sqrt{2}} \\ &= 1\,\text{kHz} \end{aligned}$$

By choosing $C = 100\,\text{nF}$, we find the correct value for R:

$$\begin{aligned} R &= \frac{1}{2\pi \cdot f_c \cdot C \cdot \sqrt{2}} \\ &= \frac{1}{2\pi \cdot 1\,10^3\,\text{Hz} \cdot 100\,10^{-9}\,\text{F} \cdot \sqrt{2}} \\ &= 1.125\,\text{k}\Omega \end{aligned}$$

By substituting the ratios we supposed in (A.5), we find the configuration and theoretical component values shown in figure A.2

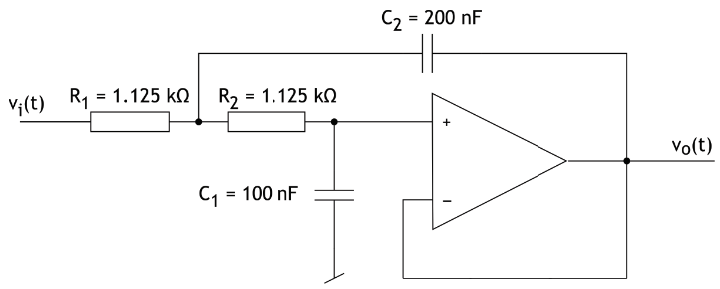


Figure A.2: Theoretical component values for LPF with $f_c = 1\,\text{kHz}$ and $Q = \frac{1}{\sqrt{2}}$

A.2 Motor

Figure A.3 shows the typical behaviour of a DC motor. When the motor is accelerated (from standstill) using a constant voltage, it starts with a torque equal to the *stall torque* τ_H, drawing the *starting current* i_A. When it is gaining speed (ω), the current and the torque decrease. In an ideal scenario, when there is no friction in the bearings and the air, the motor arrives at its no load speed ω_0, using no current.

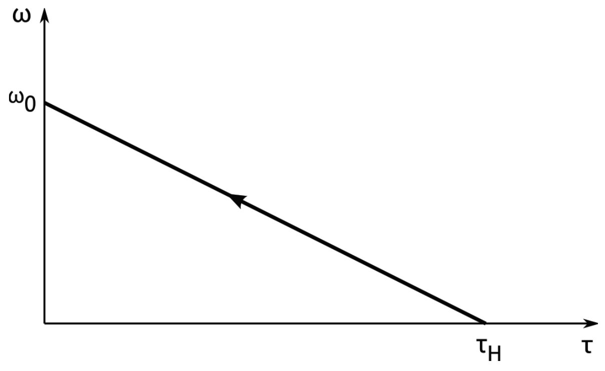


Figure A.3: Typical behaviour of a DC motor for constant voltage, no friction, accelerating

To get an indication of the power that the motor should be able to deliver, the following calculations are done with the assumption that there is no friction in the bearings and the air. However, there will actually be friction. These calculations are merely to determine the order of some parameters, so this friction is disregarded.

A distinction should be made between two basic modes of acceleration. Acceleration at constant terminal voltage and constant terminal current is shown in figures A.4 and A.5, respectively. The acceleration at $t = 0$ $\left(\frac{\mathrm{d}\,\omega_c}{\mathrm{d}\,t}(0)\right)$ is the same for both modes.

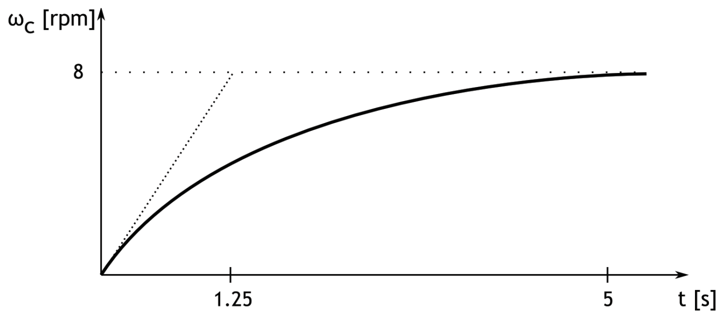


Figure A.4: Rotational speed of the cube in function of time: constant voltage

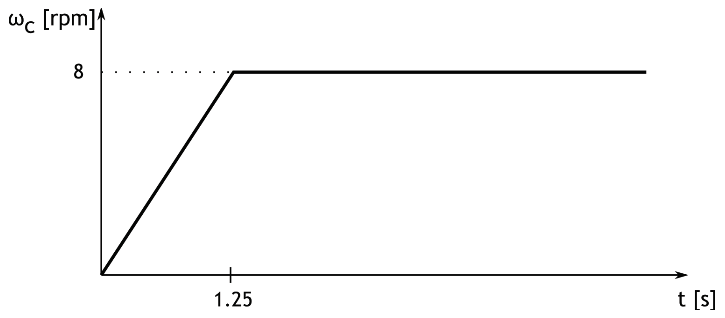


Figure A.5: Rotational speed of the cube in function of time: constant current

A speed of 8 rpm is to be obtained in 5 s, working with constant voltage. Figure A.4 shows that the $\omega(t)$ curve for this mode is an exponential function. This exponential curve reaches 98% of its endpoint in $4 \cdot \tau$ (here is τ the time constant of the exponential curve). If we were working with constant current, the curve would reach its endpoint in $1 \cdot \tau$. Thus, the power needed to obtain 8 rpm in 5 s with constant voltage is the same as the power needed for reaching 8 rpm in $\frac{5\,\mathrm{s}}{4} = 1.25\,\mathrm{s}$ with constant current. Constant current mode simplifies the calculations, so this mode is used.

With the linear acceleration of the constant current mode (and a starting speed of 0 rad/s), we find the needed acceleration of the cube, to arrive at a speed of 8 rpm = 0.84 rad/s in 1.25 s:

$$\begin{aligned} \omega_c &= \alpha_c \cdot t \\ \Leftrightarrow \alpha_c &= \frac{\omega_c}{t} \\ &= \frac{0.84\,\mathrm{rad/s}}{1.25\,\mathrm{s}} = 0.672\,\mathrm{rad/s}^2 \end{aligned}$$

Supposing that the inertia of the cube is (using the formula for the inertia of a cylinder rotating around its axis):

$$\begin{aligned} I_c &= \frac{1}{2} \cdot m_c \cdot R_c^2 \\ &= \frac{1}{2} \cdot 0.4\,\mathrm{kg} \cdot 0.05\,\mathrm{m}^2 \\ &= 0.0005\,\mathrm{kg} \cdot \mathrm{m}^2 \end{aligned}$$

Then we can find the stall torque that the motor should be able to deliver in order to bring the cube from standstill to 8 rpm in 1.25 s:

$$\begin{aligned} \tau &= I_c \cdot \alpha \\ &= 0.0005\,\mathrm{kg} \cdot \mathrm{m}^2 \cdot 0.672\,\mathrm{rad/s}^2 \\ &= 0.336\,\mathrm{mNm} \\ &= \tau_H \end{aligned}$$

The mechanical power of a motor is given by:

$$P_{mech} = \tau \cdot \omega$$

With the stall torque τ_H found above, and the desired speed of the cube ω_c, the needed power for the motor is:

$$\begin{aligned} &= \tau_H \cdot \omega_c \\ &= 0.336\,\mathrm{mNm} \cdot 0.84\,\mathrm{rad/s} \\ &= 0.28\,\mathrm{mW} \end{aligned}$$

The stall torque is applied when the speed is zero. When the speed is higher, the torque is lower. Since these two maximums never occur together, this means that the actual needed power will be lower than this absolute maximum value (still assuming that there is no friction). The numeric values for τ_H and P_{mech} let us choose an appriopriate motor. Of the motors that meet these requirements, the one with the lowest no load current is the best choice.

A.3 Inertia Wheel

To determine the optimal inertia of the wheel, different inertias have been tested. Different bars have been used, with or without weigths at a certain distance of the center of the bar. For these tests, the cube was hanging in the existing levitation system, the motor was operating at 3 V. A H-bridge made it possible to change the direction of the motor - the switches of the bridge were not modulated, this corresponds to a PWM duty cycle $\delta = 100\%$.

The inertias of the used wheels are calculated with the following assumptions: the bar is seen as a rod with an infinite small radius, a mass m_b and length l_b, for which the inertia is calculated as $I_b = \frac{1}{12} \cdot m_b \cdot l_b^2$. The weights are simplified to point masses with a certain mass m_w at a distance r_w from the center of rotation, the inertia is calculated as $I_w = m_w \cdot r_w^2$. The total inertia of the wheel is given by:

$$\begin{aligned} I &= I_b + 2 \cdot I_w \\ &= \frac{1}{12} \cdot m_b \cdot l_b^2 + 2 \cdot m_w \cdot r_w^2 \end{aligned}$$

The table in figure A.6 shows the inertia for each wheel.

Mass bar	Length bar	Mass weights (each)	Weigths dist to center	**Inertia**	
[g]	[mm]	[g]	[mm]	[$g \cdot mm^2$]	[$kg \cdot m^2$ 10^{-7}]
4.83	80	5.56	30	12584	125.84
4.83	80	5.56	24	8981	89.81
4.83	80	5.56	20	7024	70.24
4.83	80	5.56	14	4755	47.55
4.83	80	5.56	10	3688	36.88
4.83	80	0	x	2576	25.76
3.96	60	0	x	1188	11.88
2.64	40	0	x	352	3.52
1.35	20	0	x	45	0.45

Figure A.6: Calculation of the different inertias

Then, for every wheel with its own inertia, two timing values are measured:

- The time that the cube needs to make five rotations, when it started from standstill [s].
- The time between a ‘direction change’ command, and the actual direction change of the cube [s]. The ‘direction change’ command is given after a 5 s acceleration.

Inertia $[\mathrm{kg} \cdot \mathrm{m}^2\, 10^{-7}]$	Time to make **5 rotations**, starting from standstill [s]	**Direction change** time [s]
125.84	10.5	6
89.81	10.7	5.5
70.24	11	5.2
47.55	11.2	3.8
36.88	11.3	2.9
25.76	11.7	2.1
11.88	12	2
3.52	15	3.2
0.45	38	23

Figure A.7: Timing values of cube responsitivity in function of wheel inertia: table

Figure A.8 shows these timing values in a graph.

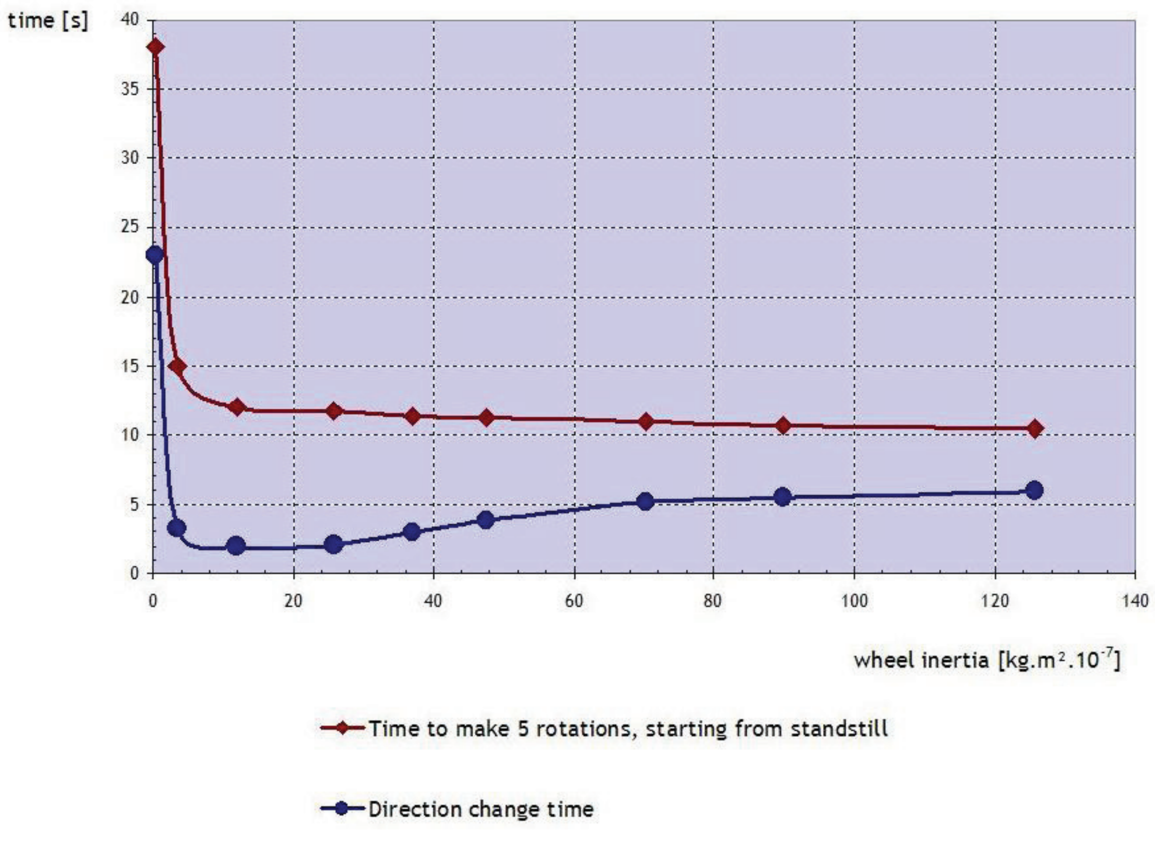


Figure A.8: Timing values of cube responsitivity in function of wheel inertia: graph

From the table in figure A.7 and the graph in figure A.8, we see that the cube has a better starting time (better acceleration) for higher load inertias (refer to 'Inertia Wheel' in section 4.2.1). For load inertias of less than 3.52 $\text{kg} \cdot \text{m}^2\, 10^{-7}$, the starting time is too high. For inertias of $11.88\,\text{kg} \cdot \text{m}^2\, 10^{-7}$ or higher, the starting time is sufficient, but only small improvements are seen for increasing inertias.
To change the direction of the cube, an optimal timing is found for a load inertia of $11.88\,\text{kg} \cdot \text{m}^2\, 10^{-7}$. Direction change time increases for both higher and lower inertias. From this fact, together with the observation that starting times do not improve much for higher inertias, we can conclude that the wheel with inertia $11.88\,\text{kg} \cdot \text{m}^2\, 10^{-7}$ is the best choice.

As discussed in section 4.2.1, the speed of rotation and angular acceleration of the cube are also dependent from the friction in the air. This means that two wheels with the same inertia but different shapes are likely to give different results. A cylindrical wheel with an inertia of $11.88\,\text{kg} \cdot \text{m}^2\, 10^{-7}$ will probably result in a lower acceleration than the chosen bar of the same inertia. This has not been tested.

A.4 Timing Resolution of Distance Measurement

In order to obtain a distance measurement precision of 0.2 mm, a certain timing resolution is needed.

The time for the laserspot to travel from the first sensor to the second is related to the distance between the cube and the ceiling, so we can calculate the needed operating frequency of the MCU to arrive at the stated distance measurement precision. Figure A.9 shows the setting, and the *Maple* sheet on the next page shows the calculation and the result.

In the calculation, a mirror with an amplitude of 40°, a resonance frequency f_{res} of 3.626 kHz and a center angle α_0 of 70° is presupposed. The sensors are placed at 3 cm and 5.5 cm from the mirror.

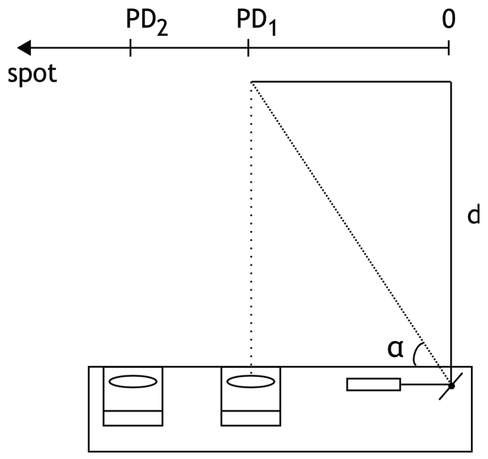


Figure A.9: Setting for the distance measurement timing resolution calculation

The result of the calculation indicates a desired timing resolution of 110 ns. For a PIC MCU, the period of an instruction cycle (the highest obtainable timing resolution) is four times the operating period. This means we need an operating period of maximum $\frac{110\,\text{ns}}{4} = 27.5\,\text{ns}$ or an operating frequency of minimum $\frac{1}{27.5\,\text{ns}} = 36.4\,\text{MHz}$.

restart : *with*(*plots*) :

Angle between the laserbeam and the horizontal plane in function of time is a sinusoidal function with amplitude 20°, frequency 3.626kHz and a center angle of 70°

alpha := $t \rightarrow \frac{20 \cdot 2 \cdot \mathrm{Pi}}{360} \cdot \cos(2 \cdot \mathrm{Pi} \cdot 3.626\mathrm{e}3 \cdot t) + \frac{70 \cdot 2 \cdot \mathrm{Pi}}{360}$;

$$t \rightarrow \frac{1}{9} \pi \cos(2 \pi \cdot 3626.\, t) + \frac{7}{18} \pi \tag{1}$$

The position of the laserspot on the ceiling is given by: spot = distance from ceiling/tangent of the angle of the beam,
so the distance from the ceiling is given by

d := (t, spot) $\rightarrow$ *spot*$\cdot$tan(alpha(t));

$$(t, spot) \rightarrow spot \tan(\alpha(t)) \tag{2}$$

position of the photodiodes

PD1 := 0.03; PD2 := 0.055;

$$0.03$$
$$0.055 \tag{3}$$

The spot passes the photodiodes faster when the cube is at a greater distance.
This means the timing is the most critical when the cube is at 8cm. When the cube rises from 8cm to 7.98cm, the time to travel between the two sensors changes with a value of:

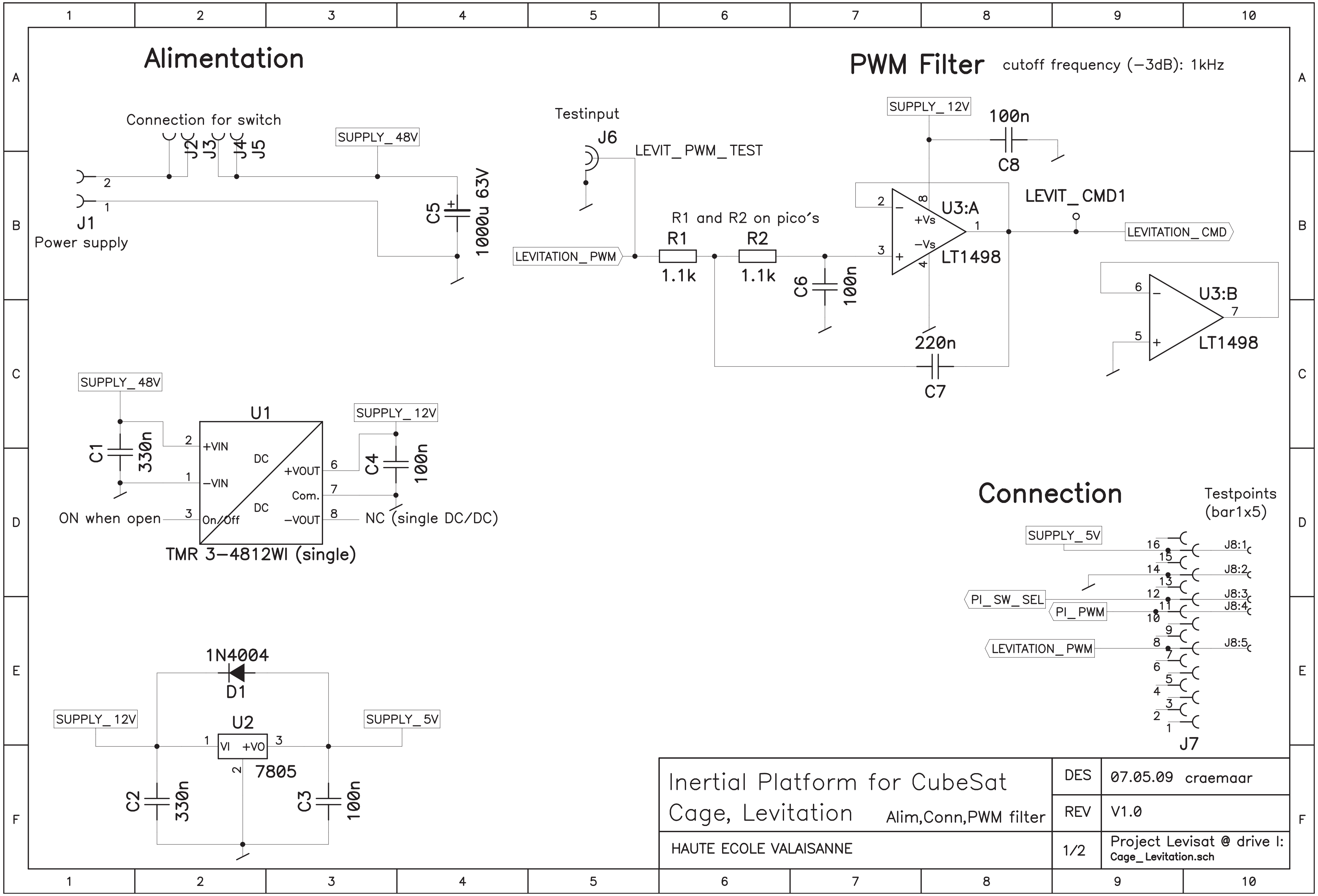
(*solve*(*d*(*t*, *PD2*) = 0.0798, *t*) − *solve*(*d*(*t*, *PD1*) = 0.0798, *t*)) − (*solve*(*d*(*t*, *PD2*) = 0.08, *t*) − *solve*(*d*(*t*, *PD1*) = 0.08, *t*));

$$1.1050796\ 10^{-7} \tag{4}$$

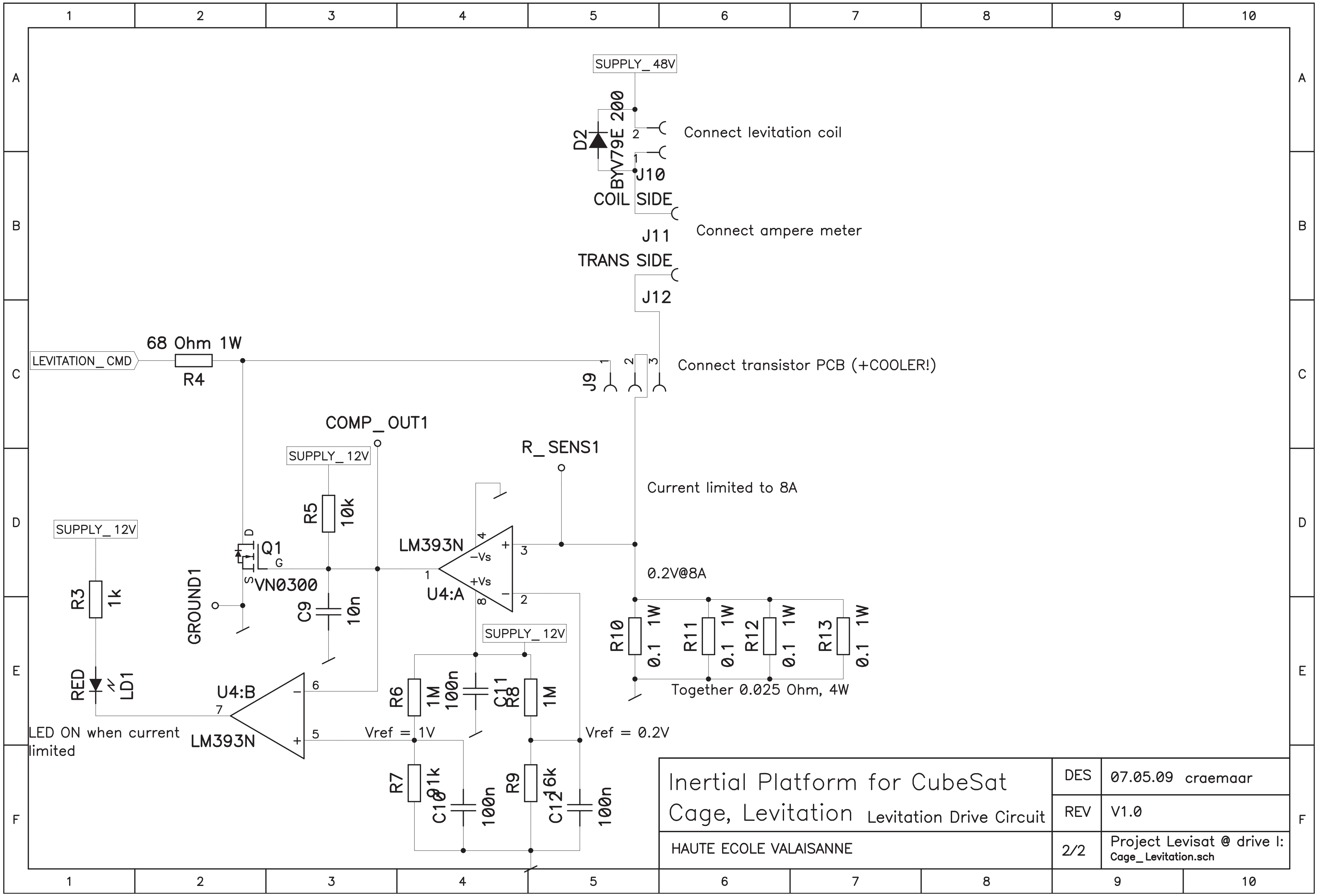
The result above gives the needed timing resolution to determine the distance at a precision of 0.2mm: 110ns

Appendix B

Schematics



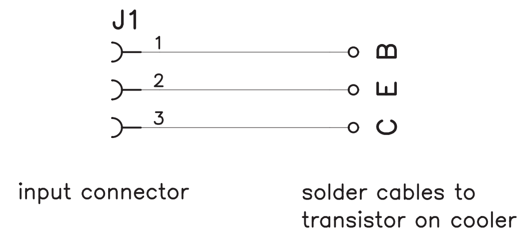
Inertial Platform for CubeSat Cage, Levitation Alim,Conn,PWM filter	DES	07.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/2	Project Levisat @ drive I: Cage_Levitation.sch



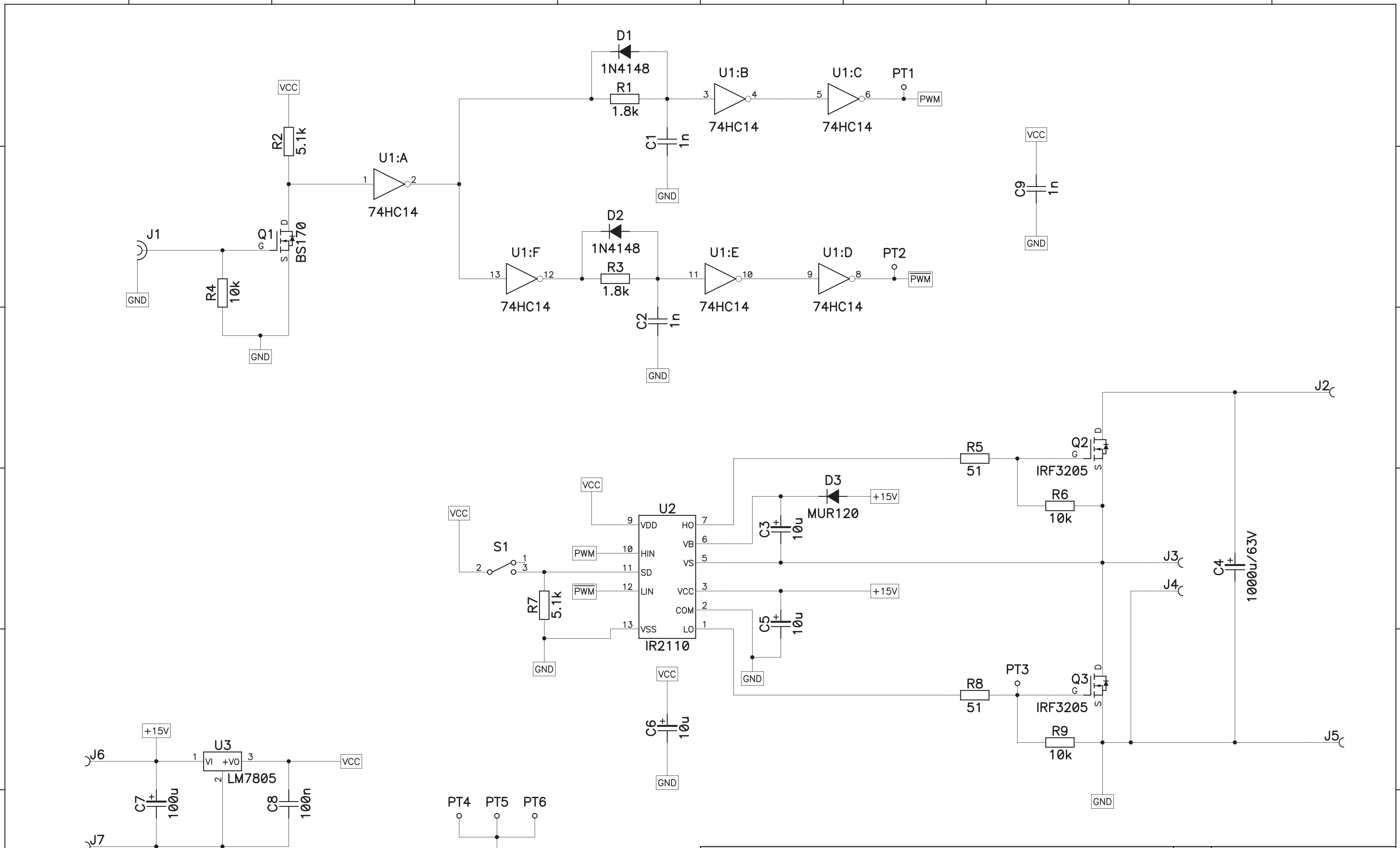
Inertial Platform for CubeSat Cage, Levitation Levitation Drive Circuit

HAUTE ECOLE VALAISANNE

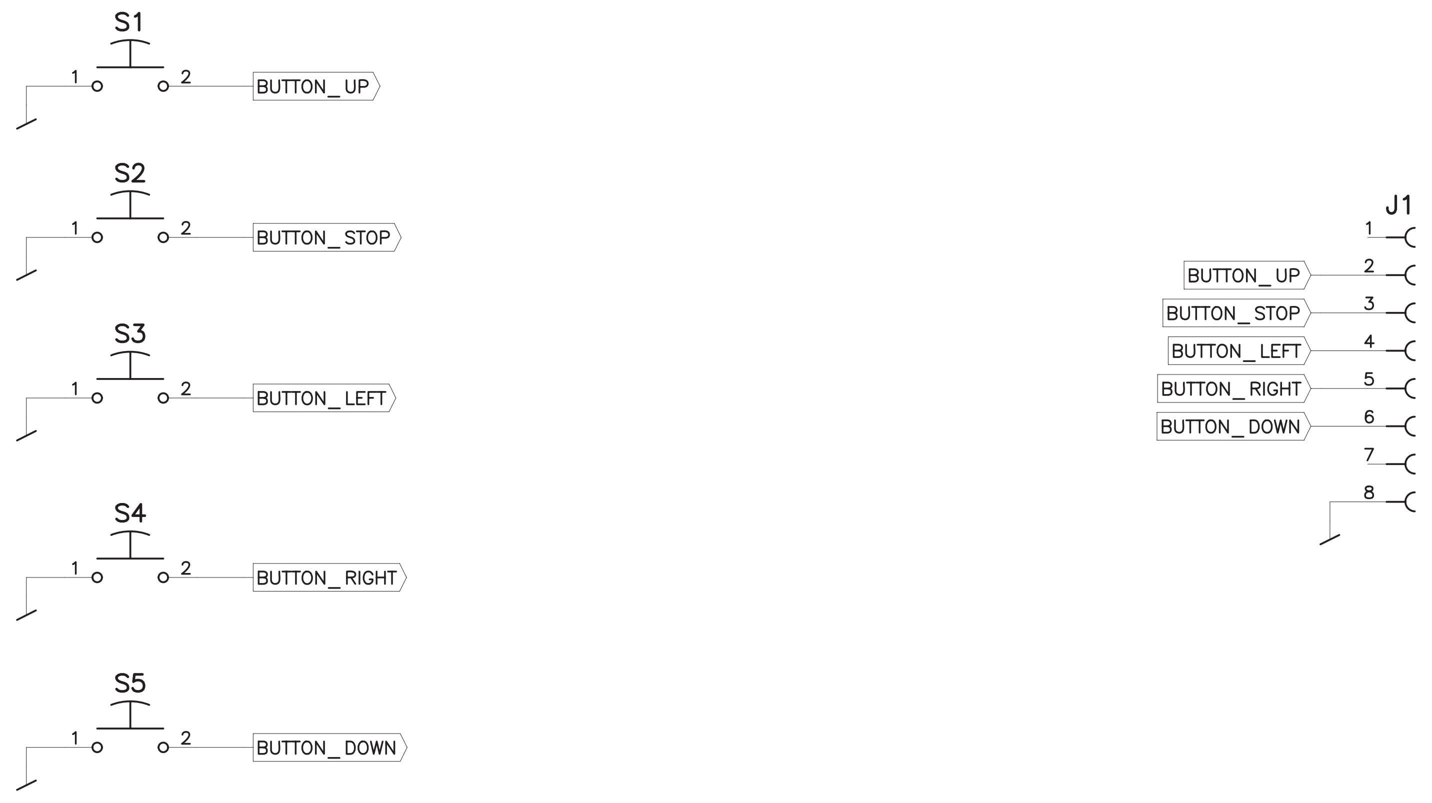
DES	07.05.09 craemaar
REV	V1.0
2/2	Project Levisat @ drive I: Cage_Levitation.sch



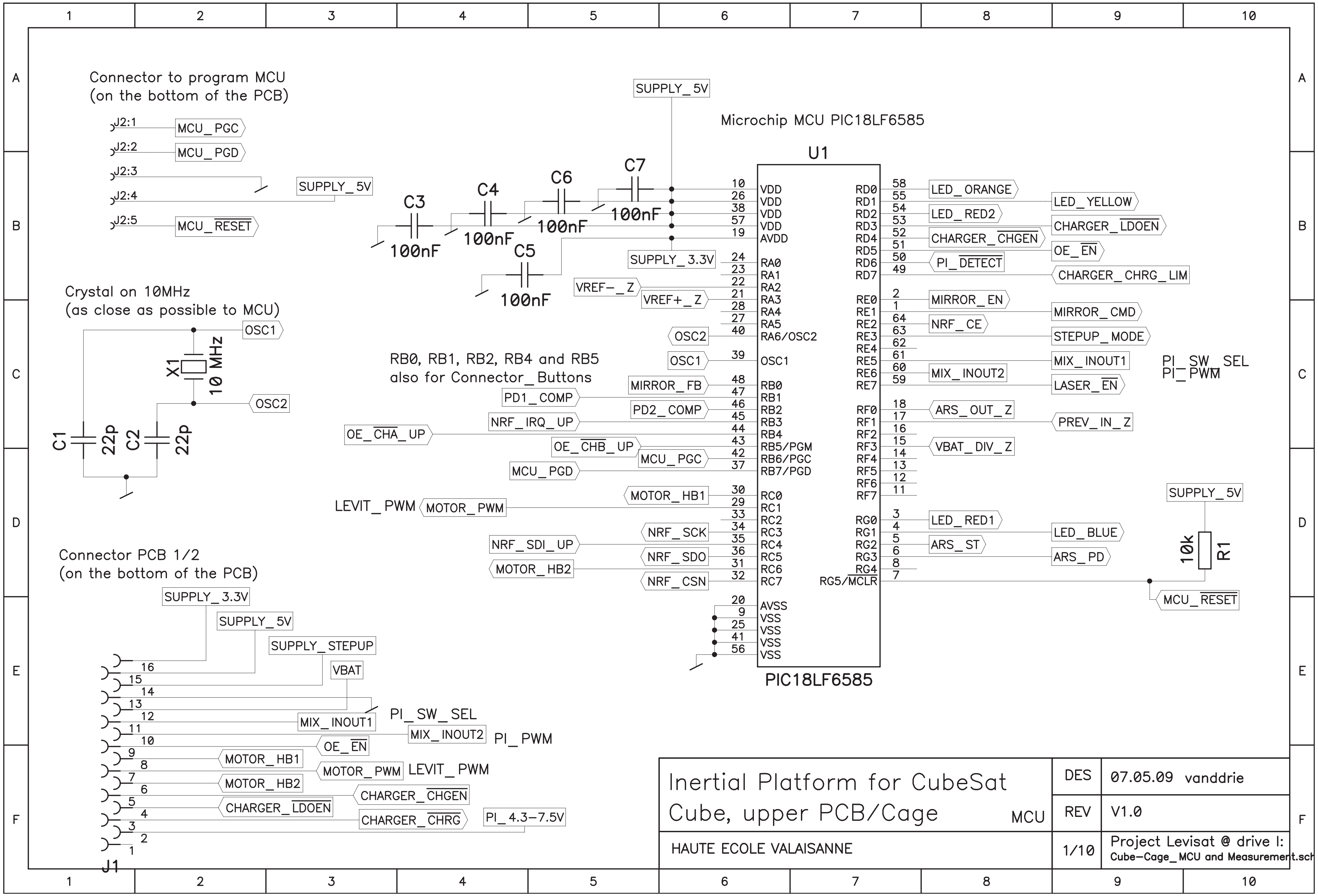
Inertial Platform for CubeSat Cage, Transistor PCB Connect	DES	30.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/1	Project Levisat @ drive I: Cage_Cooler and Power Transistor.sch



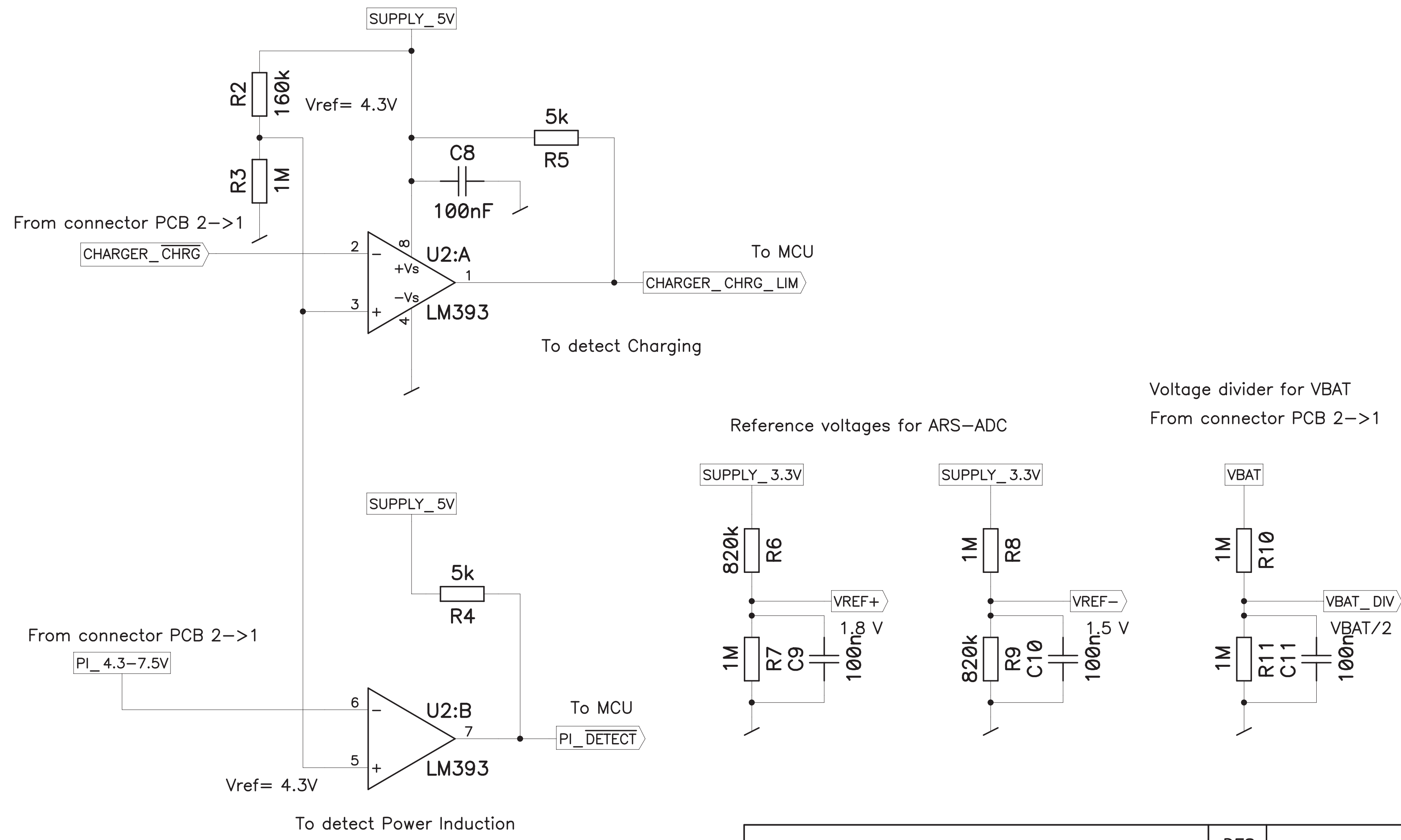
Inertial Platform for CubeSat Cage, Power Induction — Sheet1	DES	24.03.2003 ARC
	REV	1.0
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	1/1	Project Levisat @ drive I: Cage_ Power Induction.sch



Inertial Platform for CubeSat Cage, Button PCB Buttons	DES	18.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/1	Project Levisat @ drive I: Cage_Buttons.sch

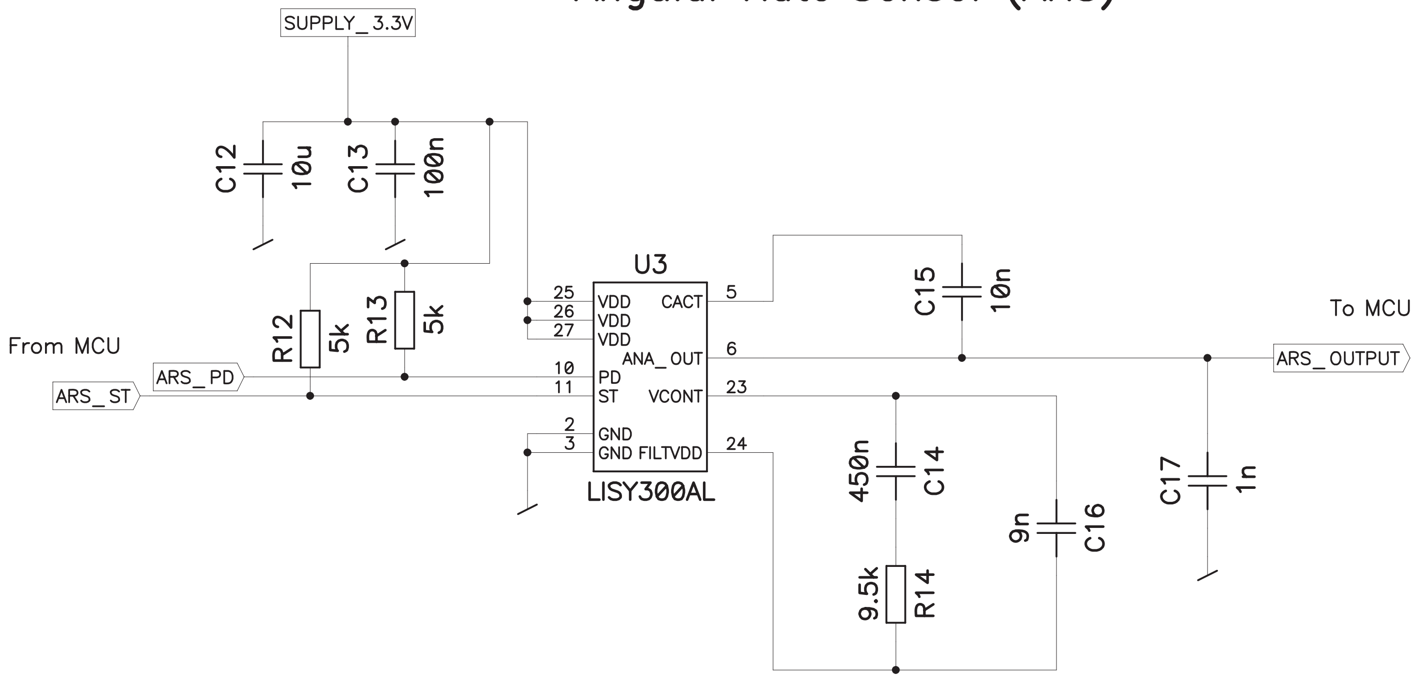


Inertial Platform for CubeSat Cube, upper PCB/Cage MCU	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/10	Project Levisat @ drive I: Cube-Cage_MCU and Measurement.sch

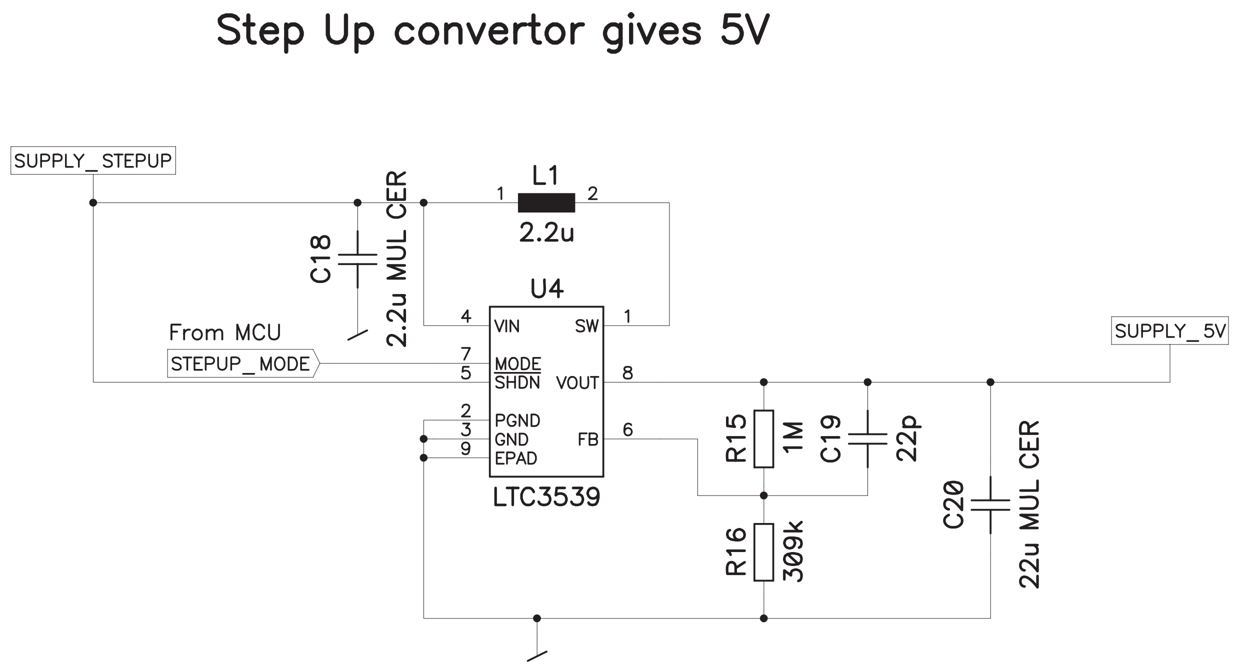


Inertial Platform for CubeSat Cube, upper PCB/Cage Limiter	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	2/10	Project Levisat @ drive I: Cube−Cage_MCU and Measurement.sch

Angular Rate Sensor (ARS)

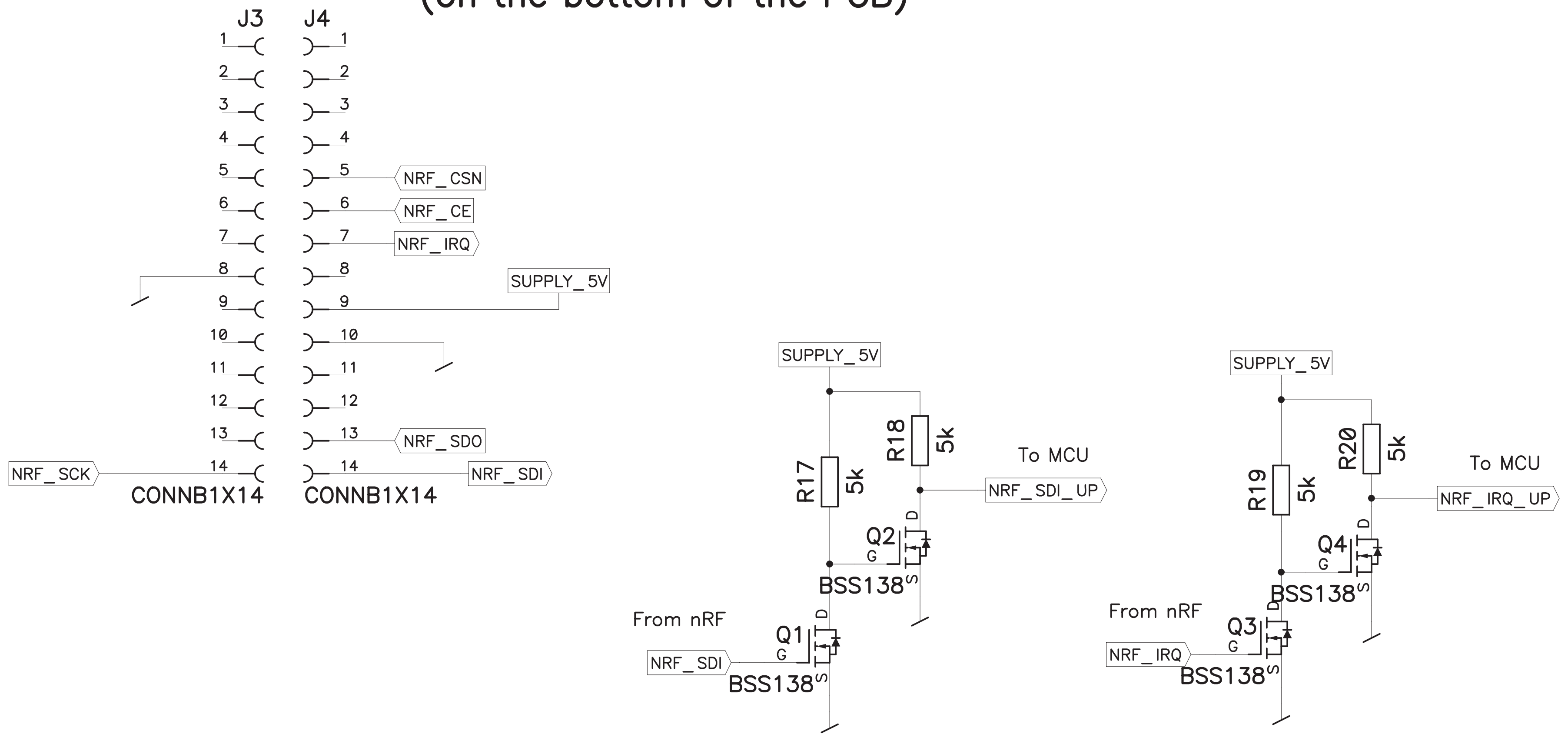


Inertial Platform for CubeSat Cube, upper PCB/Cage ARS	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	3/10	Project Levisat @ drive I: Cube-Cage_MCU and Measurement.sch



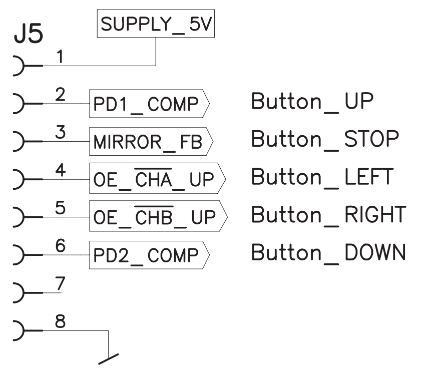
Inertial Platform for CubeSat Cube, upper PCB/Cage Stepup	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	4/10	Project Levisat @ drive I: Cube–Cage_ MCU and Measurement.sch

Connector for Plug–in "Wireless module" (on the bottom of the PCB)



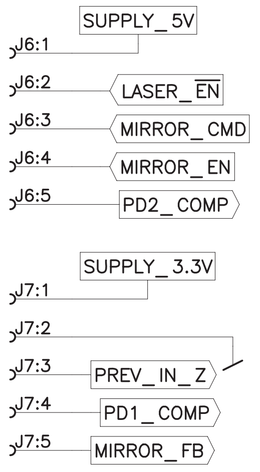
Inertial Platform for CubeSat Cube, upper PCB/Cage nRF	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	5/10	Project Levisat @ drive I: Cube–Cage_MCU and Measurement.sch

Connector for "Button PCB"

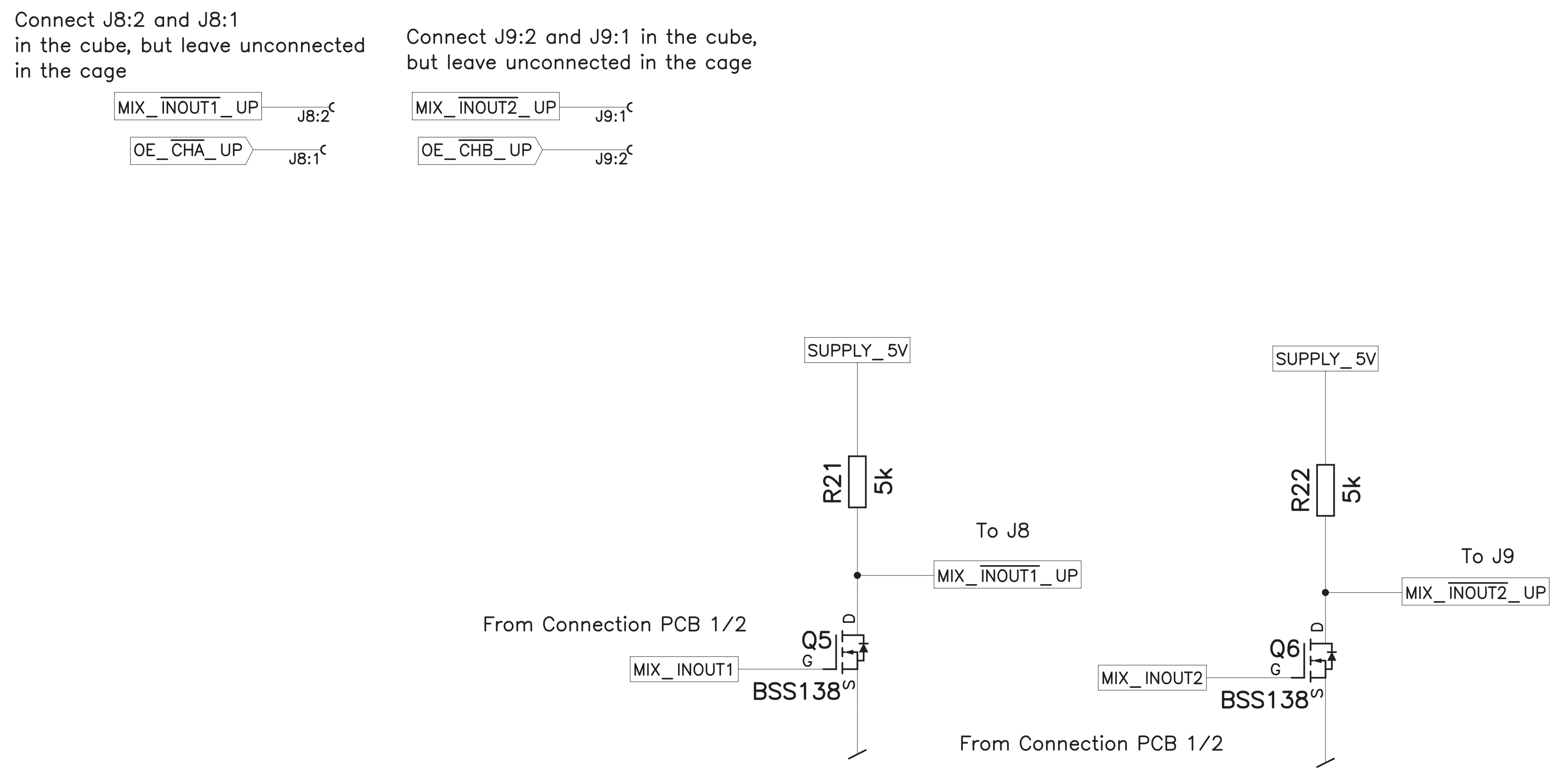


Inertial Platform for CubeSat Cube, upper PCB/Cage Connector_Buttons	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	6/10	Project Levisat @ drive I: Cube–Cage_MCU and Measurement.sch

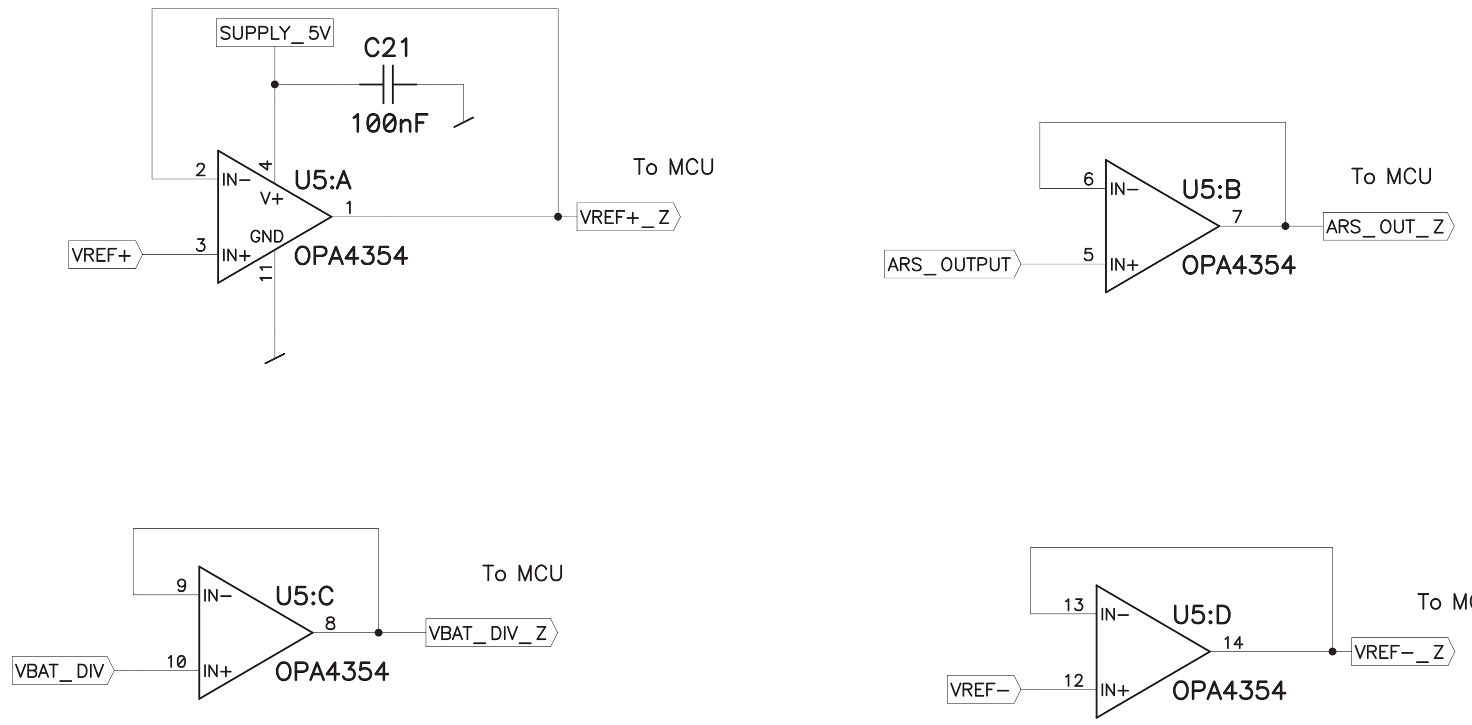
2 connectors for one and the same Plug–in "MEMS–Mirror Module"
(on top of the PCB)



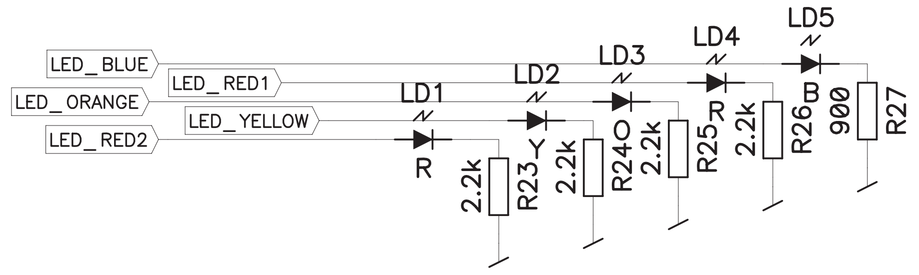
Inertial Platform for CubeSat Cube, upper PCB/Cage MEMS_mirror	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	7/10	Project Levisat @ drive I: Cube–Cage_MCU and Measurement.sch


Connect J8:2 and J8:1
in the cube, but leave unconnected
in the cage
MIX_$\overline{\text{INOUT1}}$_UP
J8:2
OE_$\overline{\text{CHA}}$_UP
J8:1
Connect J9:2 and J9:1 in the cube,
but leave unconnected in the cage
MIX_$\overline{\text{INOUT2}}$_UP
J9:1
OE_$\overline{\text{CHB}}$_UP
J9:2
SUPPLY_5V
R21
5k
To J8
MIX_$\overline{\text{INOUT1}}$_UP
From Connection PCB 1/2
Q5
BSS138
MIX_INOUT1
SUPPLY_5V
R22
5k
To J9
MIX_$\overline{\text{INOUT2}}$_UP
Q6
BSS138
MIX_INOUT2
From Connection PCB 1/2

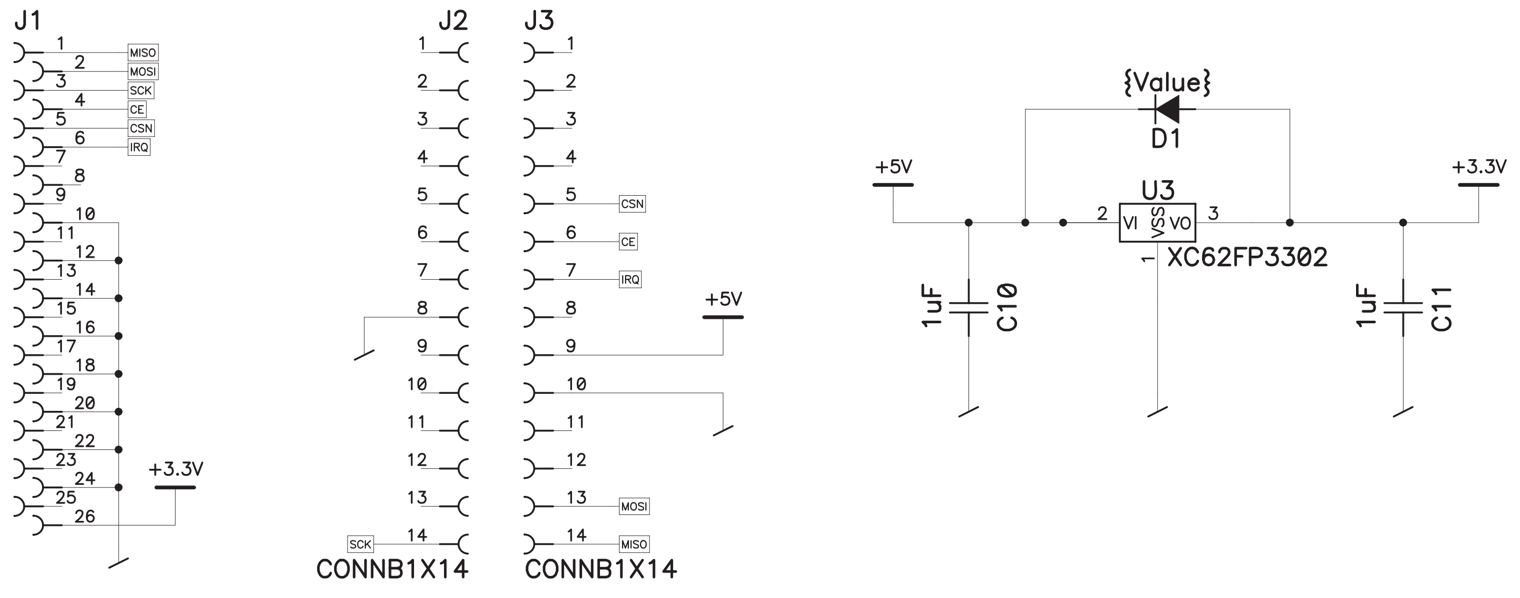
Inertial Platform for CubeSat Cube, upper PCB/Cage Corrections		DES	07.05.09 vanddrie
		REV	V1.0
HAUTE ECOLE VALAISANNE		8/10	Project Levisat @ drive I: Cube–Cage_MCU and Measurement.sch



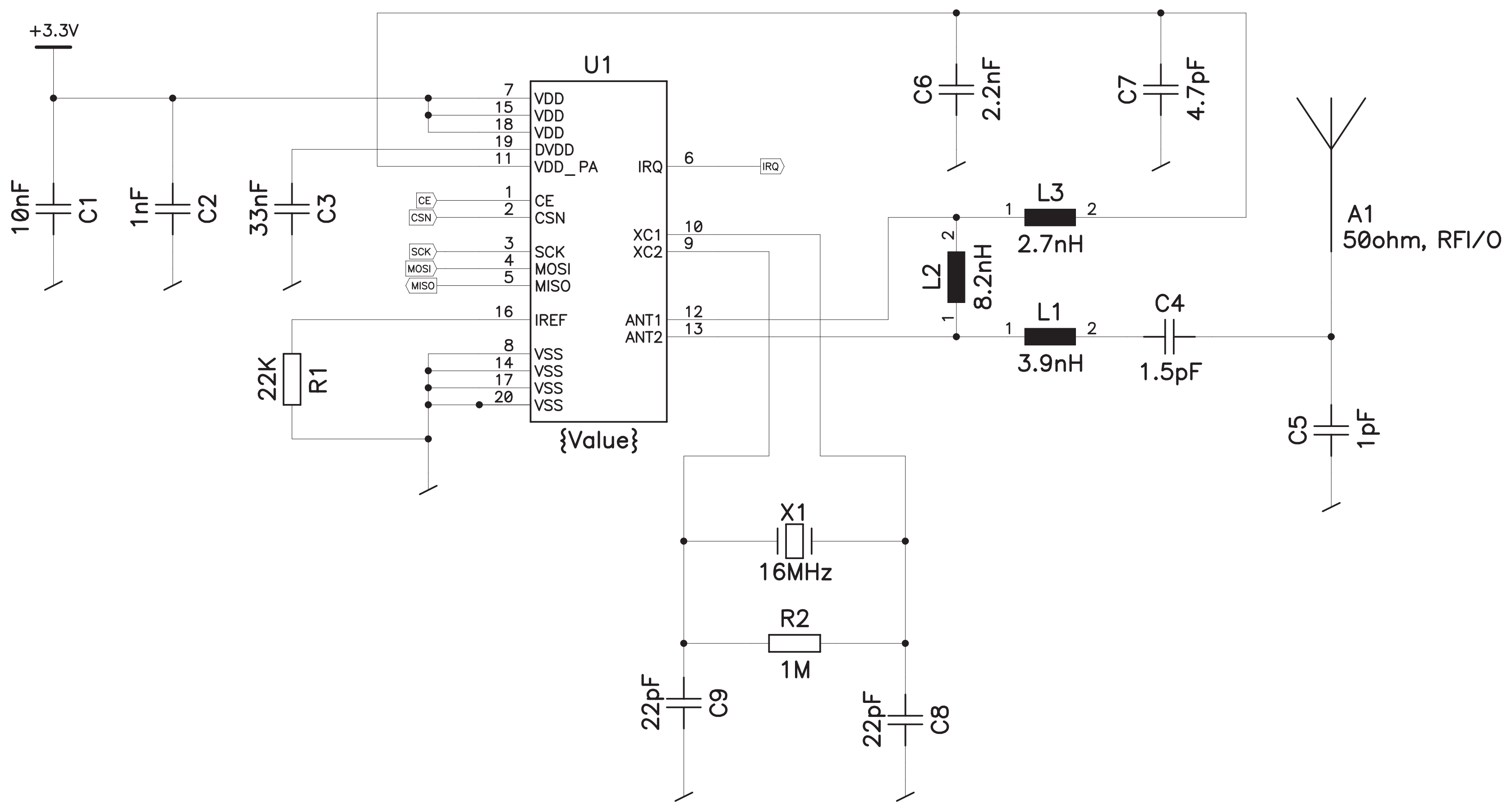
Inertial Platform for CubeSat Cube, upper PCB/Cage Impedance convert		DES	07.05.09 vanddrie
		REV	V1.0
HAUTE ECOLE VALAISANNE		9/10	Project Levisat @ drive I: Cube-Cage_MCU and Measurement.sch



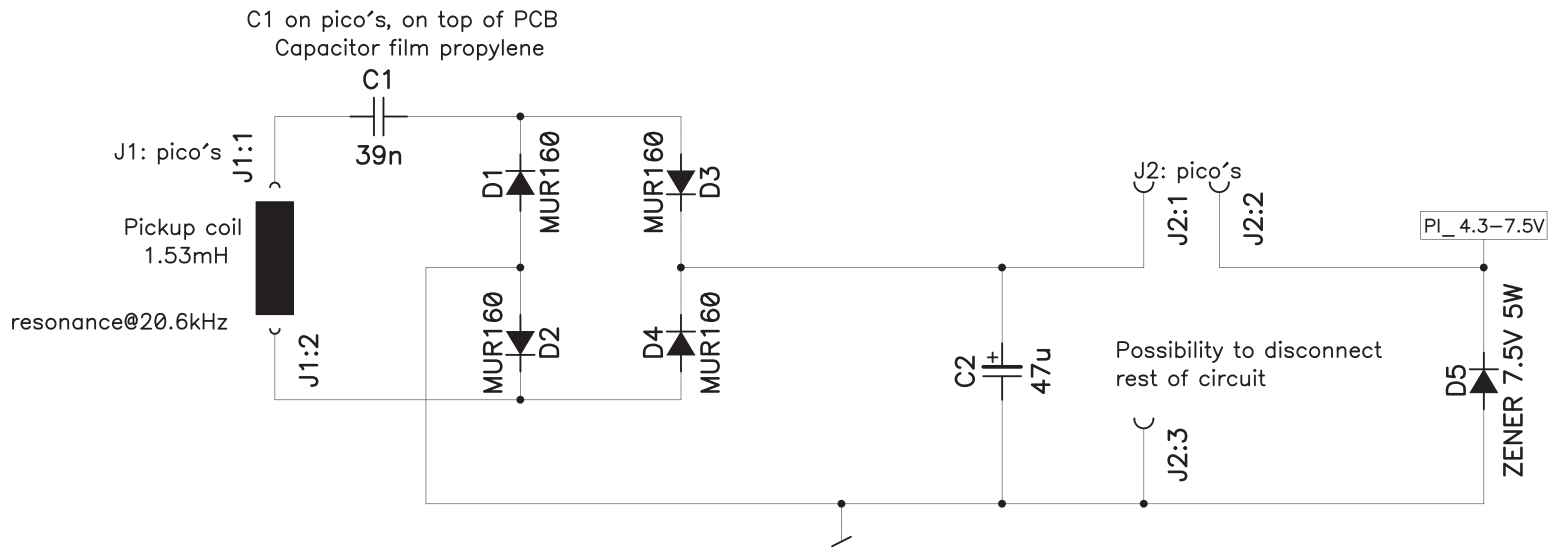
Inertial Platform for CubeSat Cube, upper PCB/Cage LEDs	DES	07.05.09 vanddrie
	REV	V1.0
HAUTE ECOLE VALAISANNE	10/10	Project Levisat @ drive I: Cube-Cage_MCU and Measurement.sch



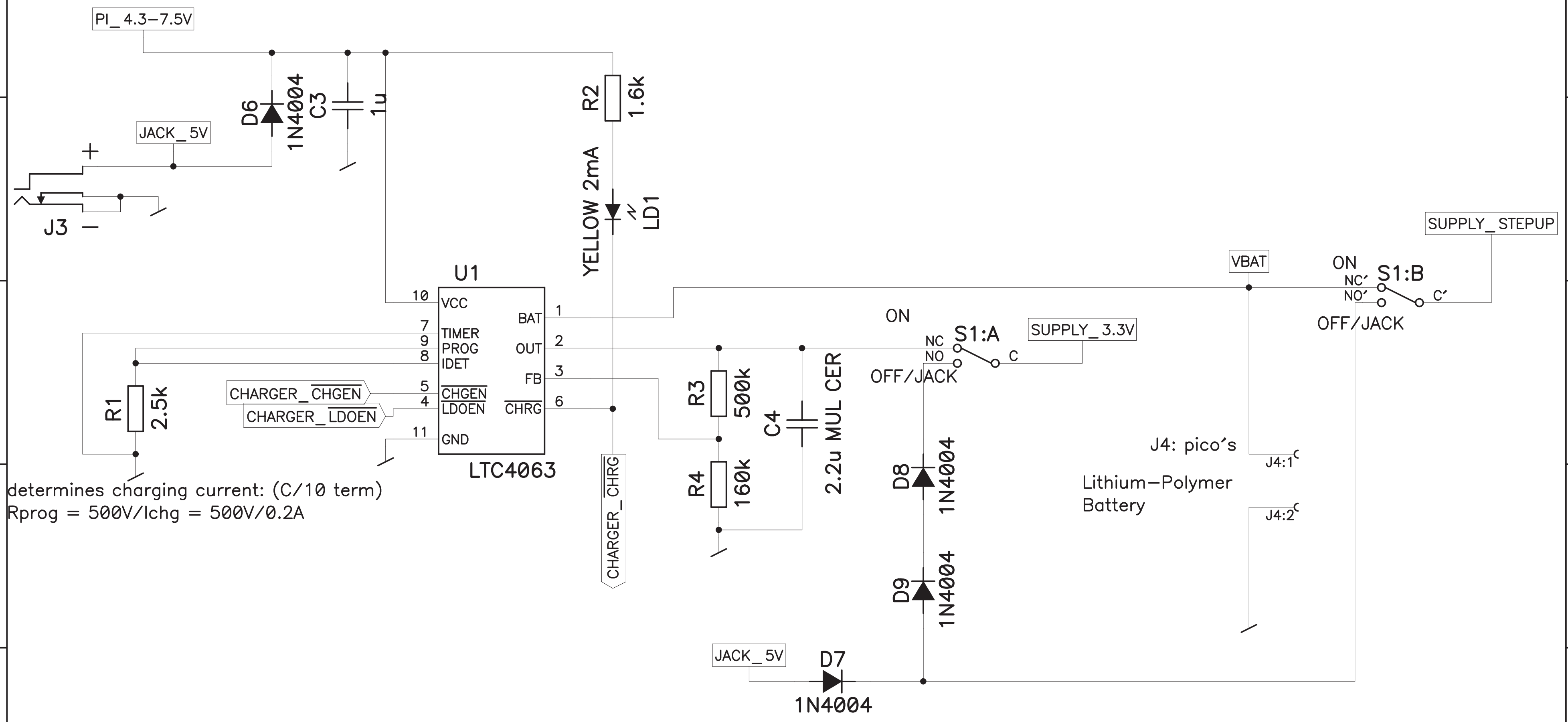
Inertial Platform for CubeSat Cube–Cage, Transceiver Power	DES	28.08.07smar
	REV	v1.0
HAUTE ECOLE VALAISANNE	1/2	Project Levisat @ drive I: Cube–Cage_Transceiver.sch



Inertial Platform for CubeSat Cube–Cage, Transceiver nRF	DES	28.08.07smar
	REV	v1.0
HAUTE ECOLE VALAISANNE	2/2	Project Levisat @ drive I: Cube–Cage_Transceiver.sch



Inertial Platform for CubeSat Cube, lower PCB — Power Induction	DES	07.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/4	Project Levisat @ drive I: Cube_Alim and Motor.sch



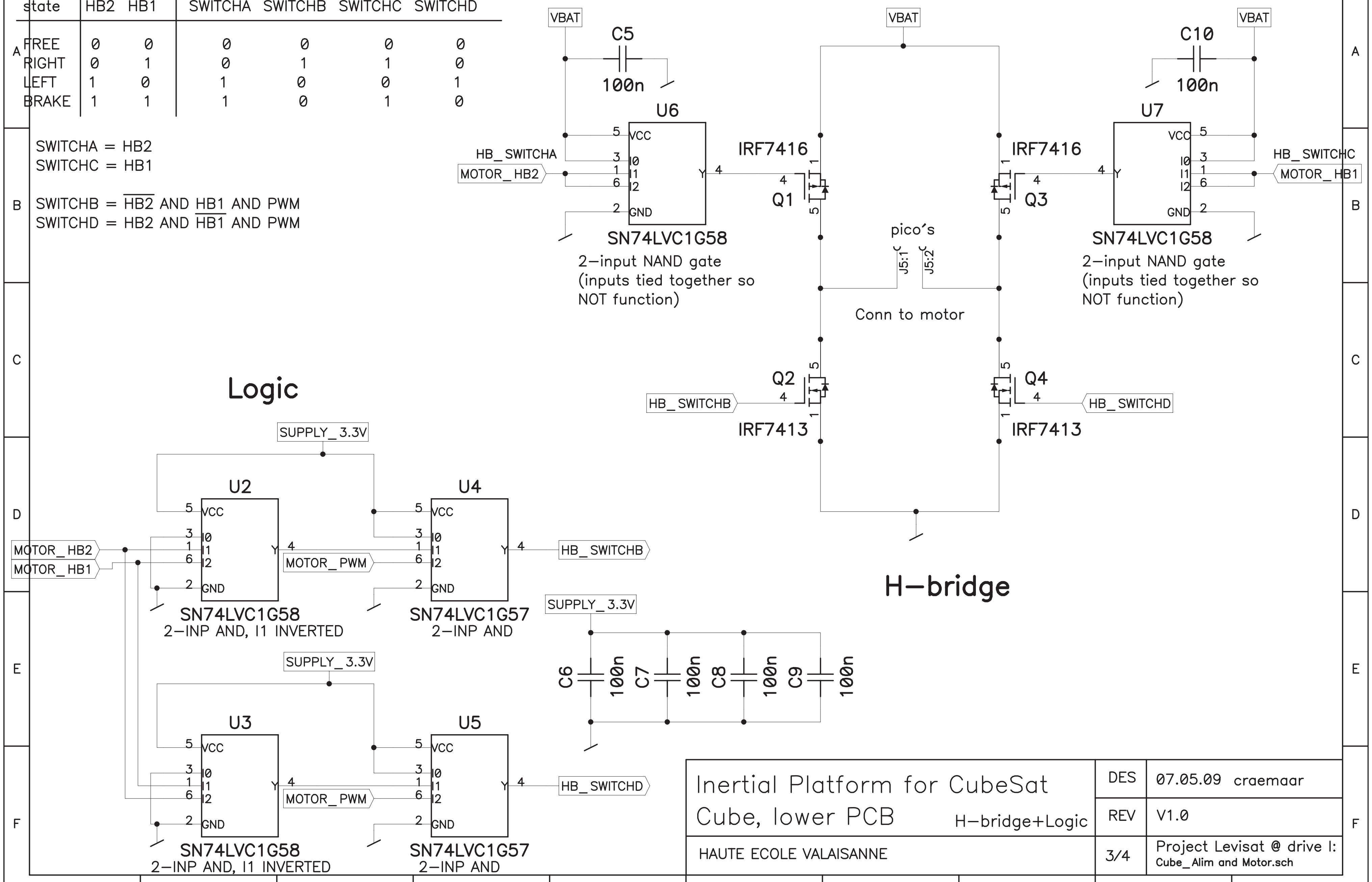
determines charging current: (C/10 term)
Rprog = 500V/Ichg = 500V/0.2A

Inertial Platform for CubeSat Cube, lower PCB Charger	DES	07.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	2/4	Project Levisat @ drive I: Cube_Alim and Motor.sch

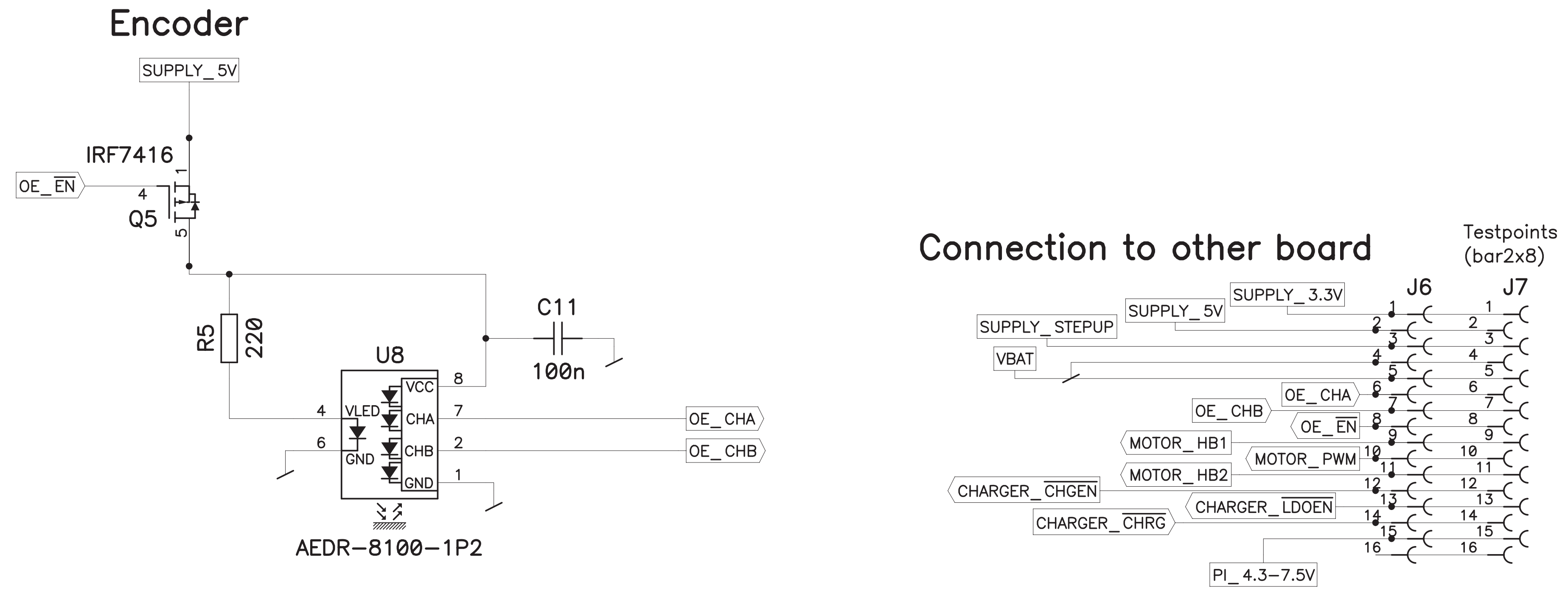
state	HB2	HB1	SWITCHA	SWITCHB	SWITCHC	SWITCHD
FREE	0	0	0	0	0	0
RIGHT	0	1	0	1	1	0
LEFT	1	0	1	0	0	1
BRAKE	1	1	1	0	1	0

SWITCHA = HB2
SWITCHC = HB1

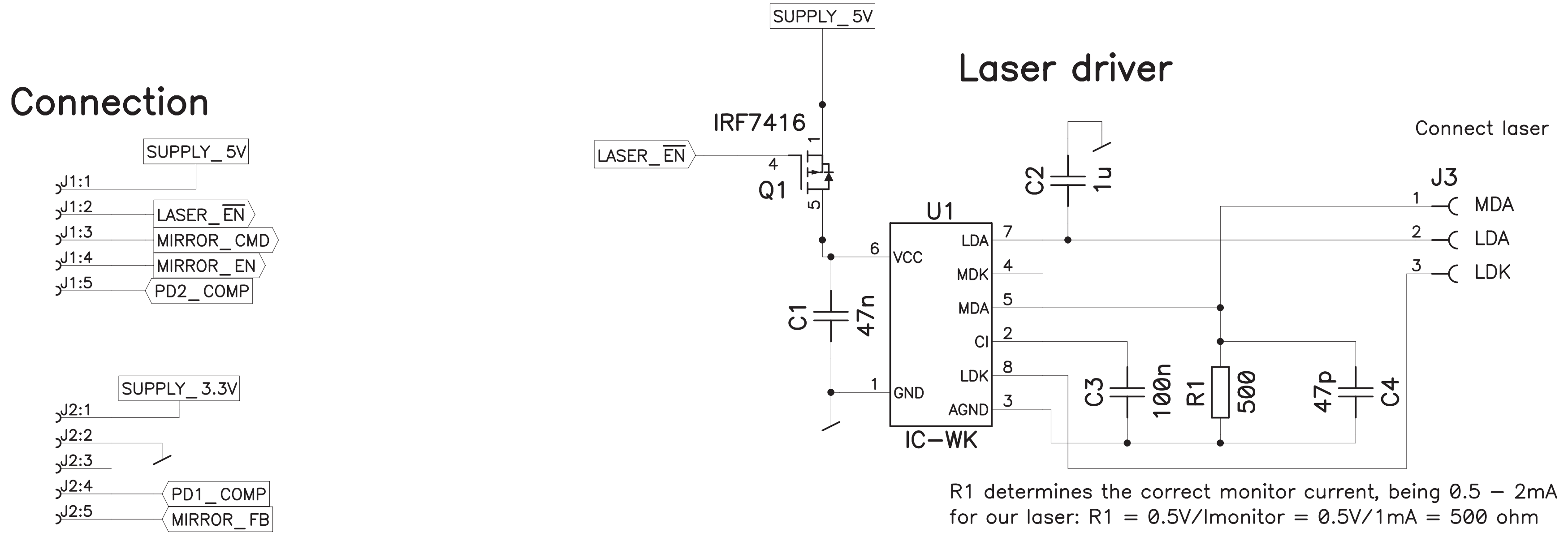
SWITCHB = $\overline{\text{HB2}}$ AND HB1 AND PWM
SWITCHD = HB2 AND $\overline{\text{HB1}}$ AND PWM



Inertial Platform for CubeSat Cube, lower PCB — H–bridge+Logic	DES	07.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	3/4	Project Levisat @ drive I: Cube_Alim and Motor.sch

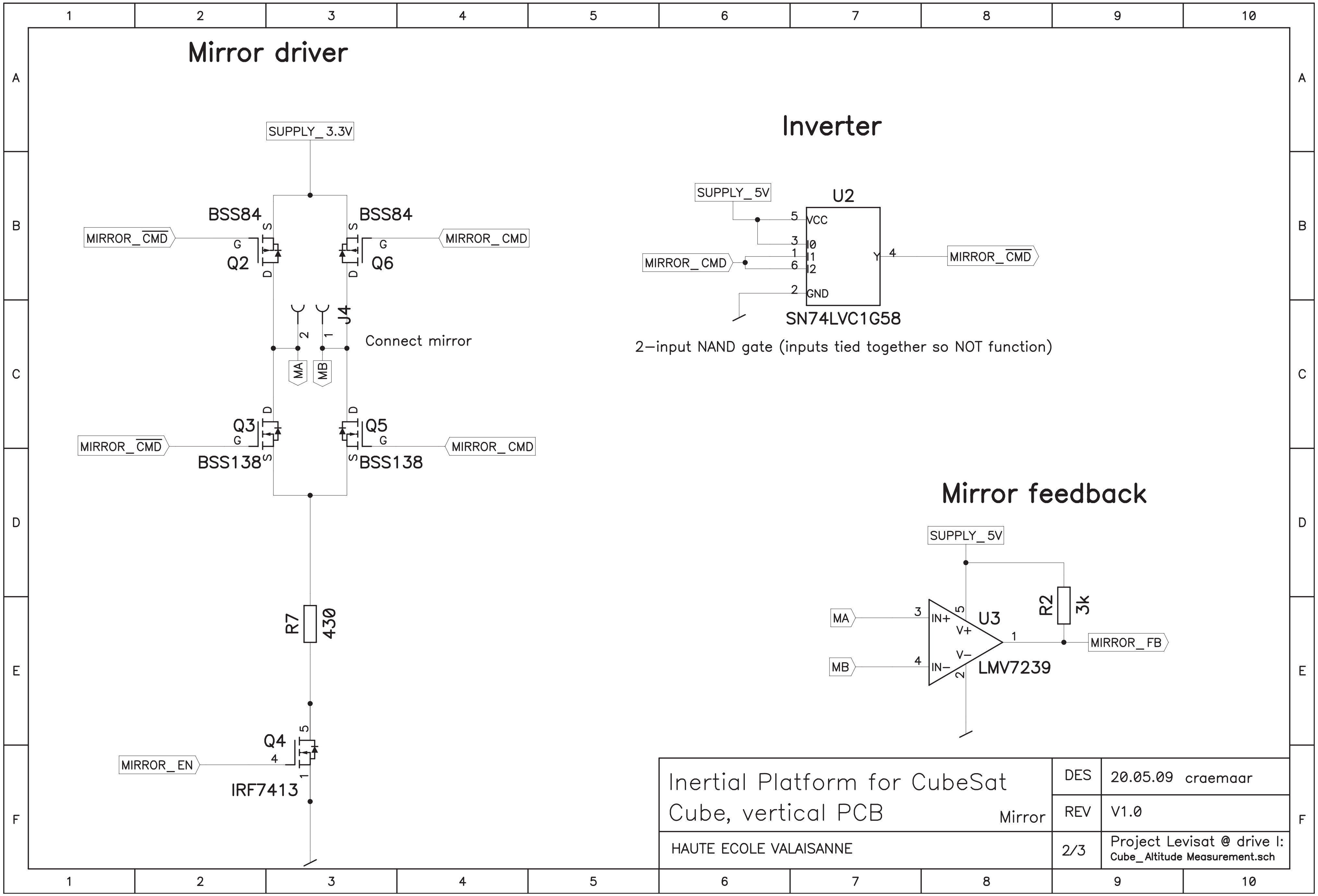


Inertial Platform for CubeSat Cube, lower PCB Encoder	DES	07.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	4/4	Project Levisat @ drive I: Cube_ Alim and Motor.sch

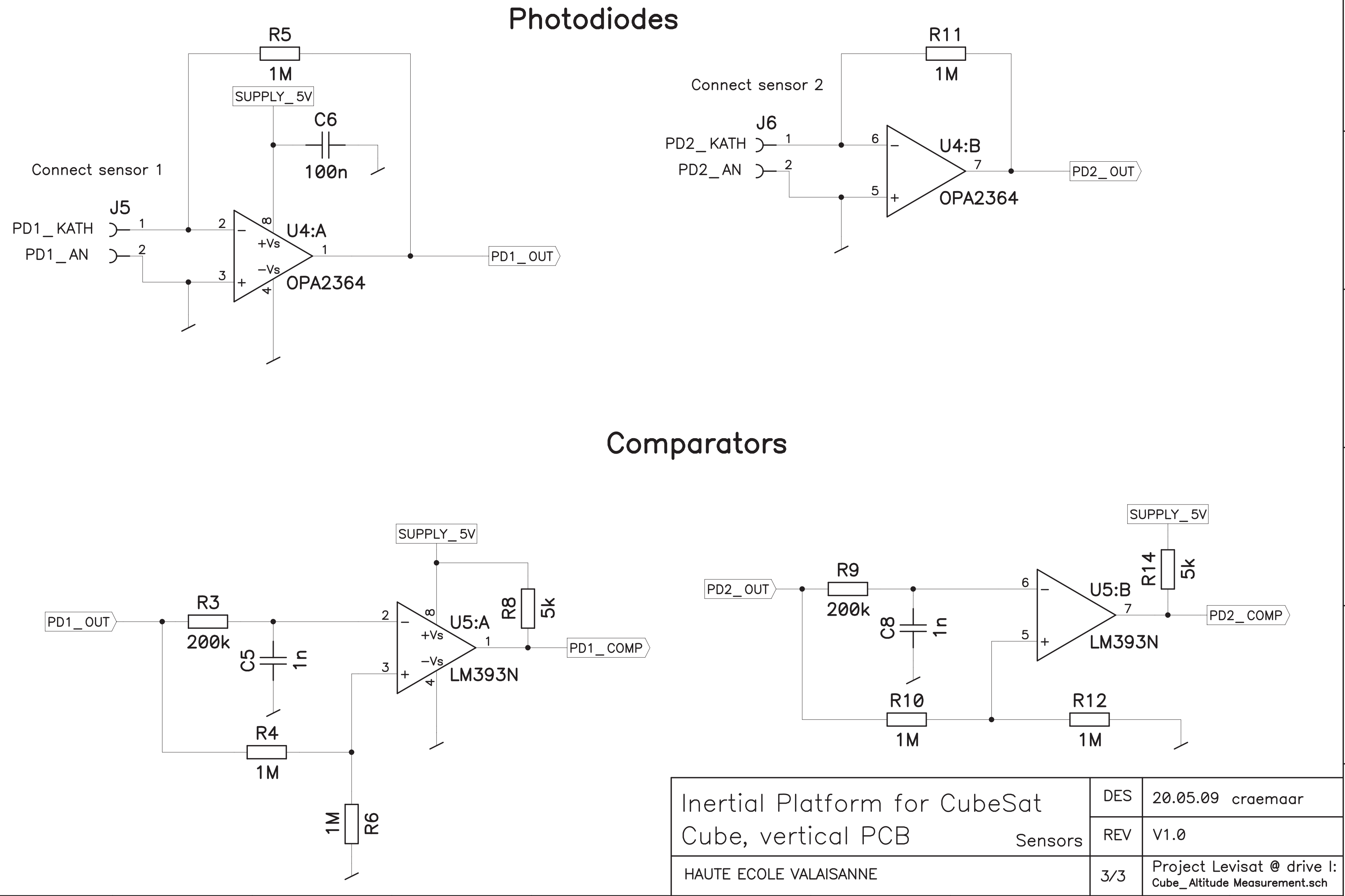


R1 determines the correct monitor current, being 0.5 – 2mA
for our laser: R1 = 0.5V/Imonitor = 0.5V/1mA = 500 ohm

Inertial Platform for CubeSat Cube, vertical PCB — Laser	DES	20.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	1/3	Project Levisat @ drive I: Cube_Altitude Measurement.sch



Inertial Platform for CubeSat Cube, vertical PCB	Mirror	DES	20.05.09 craemaar
		REV	V1.0
HAUTE ECOLE VALAISANNE		2/3	Project Levisat @ drive I: Cube_Altitude Measurement.sch

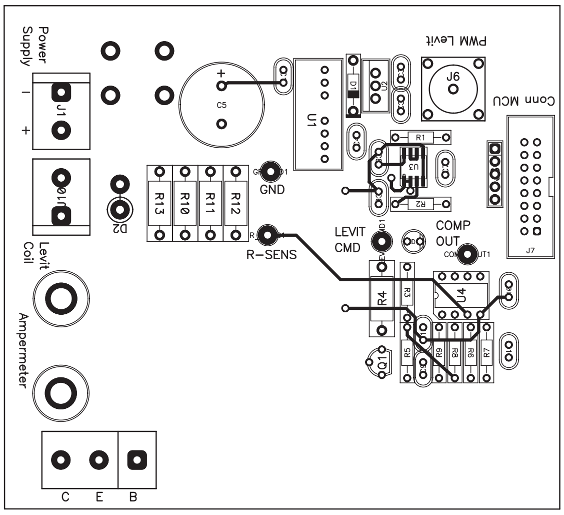


Inertial Platform for CubeSat Cube, vertical PCB Sensors	DES	20.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	3/3	Project Levisat @ drive I: Cube_ Altitude Measurement.sch

Appendix C

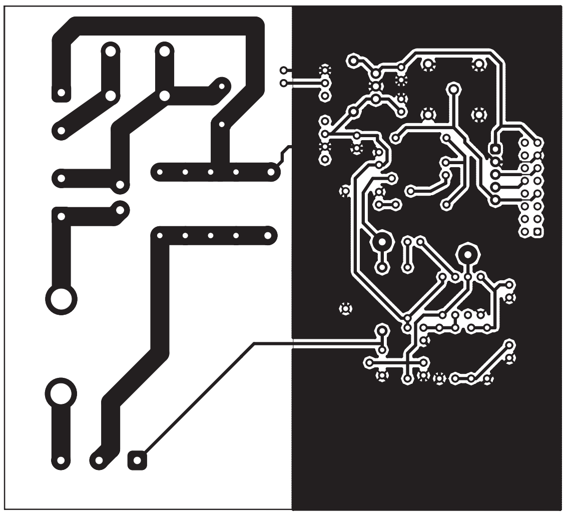
PCB Layout

TOP



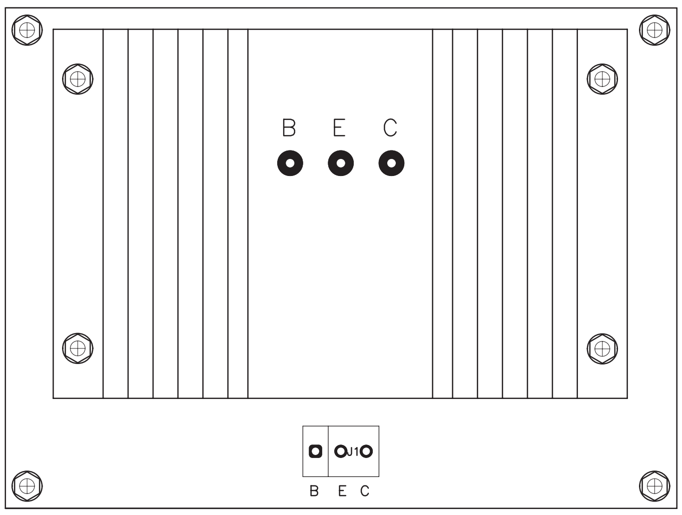
Inertial Platform for CubeSat		02.06.09 vanddrie
Cage, Levitation	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cage_Levitation.pcb	

BOTTOM



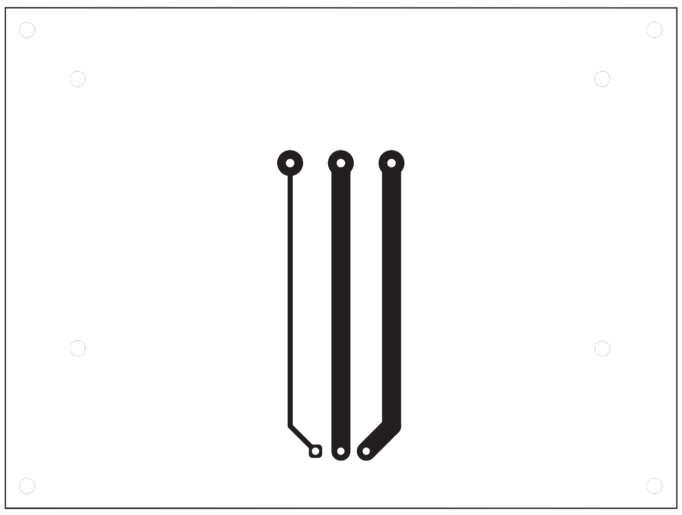
Inertial Platform for CubeSat		02.06.09 vanddrie
Cage, Levitation	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cage_Levitation.pcb	

TOP



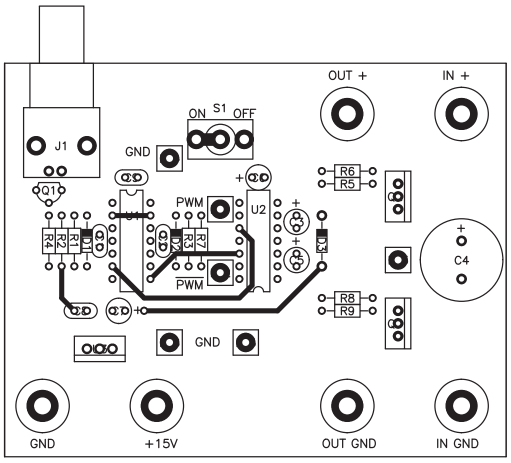
Inertial Platform for CubeSat Cage, Transistor PCB	DES	30.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cage_ Cooler and Power Transistor.pcb	

BOTTOM



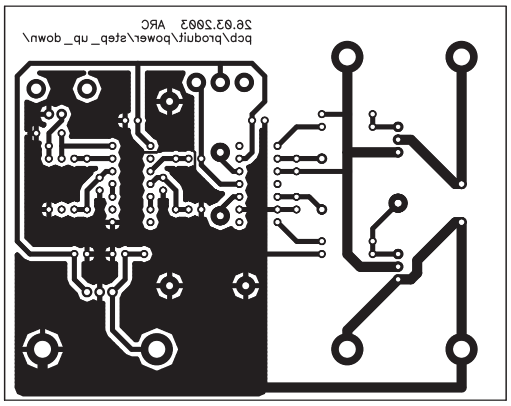
Inertial Platform for CubeSat Cage, Transistor PCB	DES	30.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cage_ Cooler and Power Transistor.pcb	

TOP



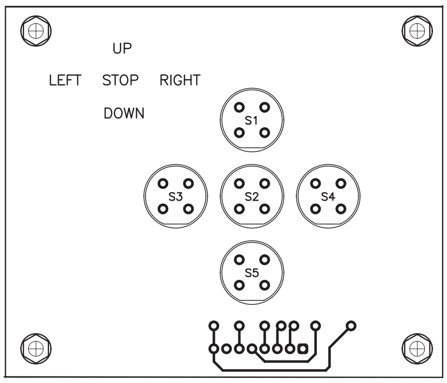
Inertial Platform for CubeSat	DES	26.03.2003 ARC
Cage, Power Induction	REV	V1.0
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	Project Levisat @ drive I: Cage_Power Induction.pcb	

BOTTOM



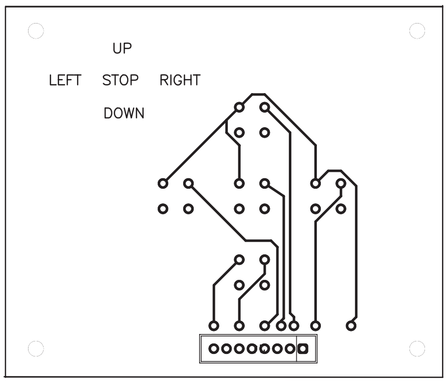
Inertial Platform for CubeSat	DES	26.03.2003 ARC
Cage, Power Induction	REV	V1.0
HAUTE ECOLE VALAISANNE – ECOLE D'INGENIEURS	Project Levisat @ drive I: Cage_Power Induction.pcb	

TOP



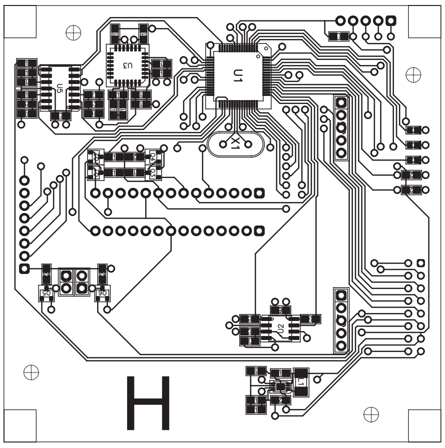
Inertial Platform for CubeSat Cage, Button PCB	DES	18.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cage_ Buttons.pcb	

BOTTOM



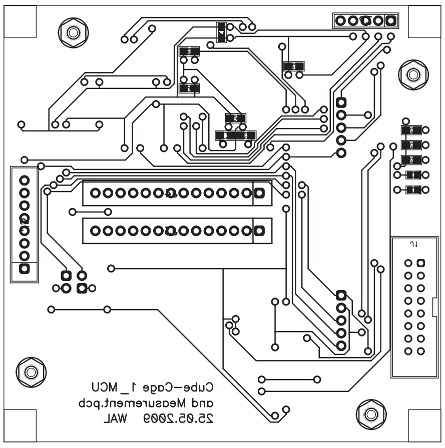
Inertial Platform for CubeSat Cage, Button PCB	DES	18.05.09 craemaar
	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cage_ Buttons.pcb	

TOP



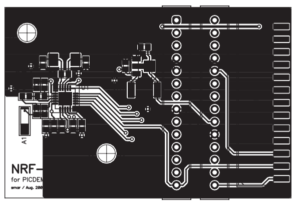
Inertial Platform for CubeSat Cube, upper PCB/Cage, signal PCB	DES	25.05.2009 WAL
	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube–Cage_ MCU and Measurement.pcb	

BOTTOM



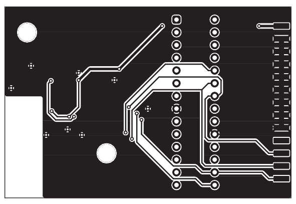
Inertial Platform for CubeSat Cube, upper PCB/Cage, signal PCB	DES	25.05.2009 WAL
	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube–Cage_ MCU and Measurement.pcb	

TOP



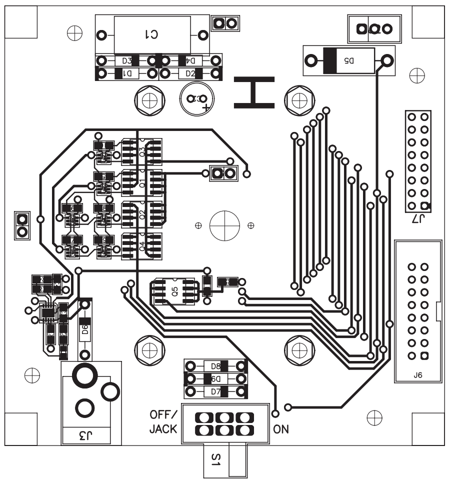
Inertial Platform for CubeSat Transceiver	DES	29.08.2007 smar
	REV	v1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube-Cage_Transceiver.pcb	

BOTTOM



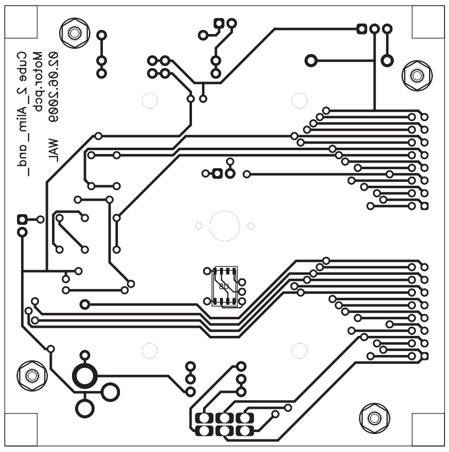
Inertial Platform for CubeSat Transceiver	DES	29.08.2007 smar
	REV	v1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube-Cage_Transceiver.pcb	

TOP



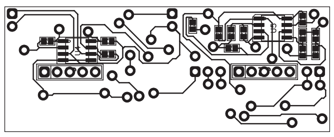
Inertial Platform for CubeSat	DES	02.06.2009	WAL
Cube, lower PCB	REV	V1.0	
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube_Alim and Motor.pcb		

BOTTOM



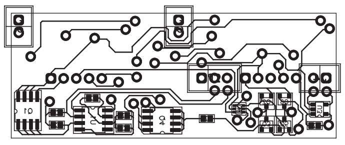
Inertial Platform for CubeSat	DES	02.06.2009	WAL
Cube, lower PCB	REV	V1.0	
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube_Alim and Motor.pcb		

TOP



Inertial Platform for CubeSat	DES	20.05.09 craemaar
Cube, vertical PCB	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube_Altitude Measurement.pcb	

BOTTOM



Inertial Platform for CubeSat	DES	20.05.09 craemaar
Cube, vertical PCB	REV	V1.0
HAUTE ECOLE VALAISANNE	Project Levisat @ drive I: Cube_Altitude Measurement.pcb	

Appendix D

Bill of Materials

```
P-CAD Bill of Materials        I:\03_Technique\...\Cage_Levitation.sch

 ======================================================================

 Count  ComponentName   RefDes          PatternName     Value           Description
 ------ --------------- --------------- --------------- --------------- ---------------

      1 2N7000          Q1              TO92            VN0300          Mosfet-N Silico
                                                                        nix - Farnell 3
                                                                        52-718
      1 BAR1X5          J8              BAR1X5
      1 BNC-PCB-D       J6              BNC-PCB-D       LEVIT PWM TEST
      1 BORNE-4MM-PCB   J11             BORNE-4MM-PCB   COIL SIDE
      1 BORNE-4MM-PCB   J12             BORNE-4MM-PCB   TRANS SIDE
      1 CAP100          C9              CAP100          10n             Capacitor
      7 CAP100          C3              CAP100          100n            Capacitor
                        C4
                        C6
                        C8
                        C10
                        C11
                        C12
      1 CAP100          C7              CAP100          220n            Capacitor
      2 CAP100          C1              CAP100          330n            Capacitor
                        C2
      1 CAP300RP        C5              CAP300RP        1000u 63V       Capacitor
      4 CONN1X1         J2              CONN1X1P        {Value}
                        J3
                        J4
                        J5
      1 CONNB2X8        J7              CONN2X8
      1 D200-UP-DIAM200
                        D2              DIODE-UP-DIAM20
                                        0               BYV79E 200
      1 D400            D1              D400            1N4004
      1 DC/DC-TMR-DUAL  U1              DC/DC-TMR-DUAL  TMR 3-4812WI (s
                                                        ingle)
      2 GMKDS3/2-7.62   J1              GMKDS3/2-7.62
                        J10
      1 GMKDS3/3-7.62   J9              GMKDS3/3-7.62
      1 LED-3MM         LD1             LED-3MM         RED
      1 LM393NO8E       U4              DIP8            LM393N
      1 OPA2350         U3              SO8             LT1498          OPA2350 high-sp
                                                                        eed  single, ra
                                                                        il to rail, amp
                                                                        li op
      1 PTEST           GROUND1         CONN1X1M        GROUND
      1 PTEST           COMP OUT1       CONN1X1M        LIMIT
      1 PTEST           R SENS1         CONN1X1M        R SENS
      1 PTEST           LEVIT CMD1      CONN1X1M
      1 R400            R3              R400            1k              Resistor
      2 R400            R6              R400            1M              Resistor
                        R8
      2 R400            R1              R400            1.1k            Resistor
                        R2
      1 R400            R5              R400            10k             Resistor
      1 R400            R9              R400            16k             Resistor
      1 R400            R7              R400            91k             Resistor
      4 R500            R10             R500            0.1  1W
                        R11
                        R12
                        R13
      1 R500            R4              R500            68 Ohm 1W
      1 REGPOS-UP       U2              TO220-UP        7805            3-Terminal Posi
                                                                        tive REGULATOR
```

```
P-CAD Bill of Materials           Cage_Cooler and Power Transistor.sch

 ======================================================================

 Count  ComponentName   RefDes          PatternName     Value           Description
 ------ --------------- --------------- --------------- --------------- ---------------

      3 PTEST           B               CONN1X1M
                        C
                        E
      1 WEIDB1X3        J1              WEIDB1X3
```

```
P-CAD Bill of Materials               I:\...\Cage_Power Induction.sch

 ======================================================================

 Count  ComponentName   RefDes          PatternName     Value           Description
 ------ --------------- --------------- --------------- --------------- ---------------

      1 74HC14          U1              DIP14                           Hex Schmitt-Tri
                                                                        gger INVERTERS
      1 BNC-PCB-C       J1              BNC-PCB-C
      6 BORNE-4MM-PCB   J2              BORNE-4MM-PCB
                        J3
                        J4
                        J5
                        J6
                        J7
      1 BS170           Q1              TO92
      3 CAP100          C1              CAP100          1n              Capacitor
                        C2
                        C9
      1 CAP100          C8              CAP100          100n            Capacitor
      3 CAP100RP        C3              CAP100RP        10u
                        C5
                        C6
      1 CAP100RP        C7              CAP100RP        100u
      1 CAP300RP        C4              CAP300RP        1000u/63V       Capacitor
      2 D400            D1              D400            1N4148
                        D2
      1 D400            D3              D400            MUR120
      1 IR2110          U2              DIP14
      6 PTEST           PT1             CONN1X1M
                        PT2
                        PT3
                        PT4
                        PT5
                        PT6
      2 R400            R1              R400            1.8k            Resistor
                        R3
      2 R400            R2              R400            5.1k            Resistor
                        R7
      3 R400            R4              R400            10k             Resistor
                        R6
                        R9
      2 R400            R5              R400            51              Resistor
                        R8
      1 REGPOS-UP       U3              TO220-UP        LM7805          3-Terminal Posi
                                                                        tive REGULATOR
      2 SUP75N06        Q2              TO220-UP        IRF3205
                        Q3
      1 SW-SCH3-M2012   S1              SW-SCH3-M2012   {Value}         Switch schema 3
                                                                         - on on 1 pole
                                                                        s - Nikkai - Di
                                                                        strelec 20 02 3
                                                                        0
```

```
 ----------------------------------------------------------------------
 12-Jun-09  09:53                                            Page    1
```

```
P-CAD Bill of Materials           I:\03_Technique\...\Cage_Buttons.sch

 ======================================================================

 Count  ComponentName   RefDes          PatternName     Value          Description
 ------ --------------- --------------- --------------- -------------- ---------------

      1 CONNB1X8        J1              CONN1X8
      5 PUSHBUTT-NO-07  S1              PUSHBUTTON-07   {Value}
                        S2
                        S3
                        S4
                        S5
```

```
P-CAD Bill of Materials       I:\...\Cube-Cage_MCU and Measurement.sch


======================================================================
Count  ComponentName   RefDes          PatternName     Value           Description
------ --------------- --------------- --------------- --------------- ---------------

     2 BAR1X2          J8              BAR1X2
                       J9
     3 BAR1X5          J2              BAR1X5
                       J6
                       J7
     6 BSS138          Q1              SOT23           BSS138          N-Channel Logic
                                                                        Level Enhancem
                                                                       ent Mode Field
                                                                       Effect Transist
                                                                       or
                       Q2
                       Q3
                       Q4
                       Q5
                       Q6
     1 CAPS0805        C17             CC0805          1n              Capacitor
     1 CAPS0805        C18             CC0805          2.2u MUL CER    Capacitor
     1 CAPS0805        C16             CC0805          9n              Capacitor
     1 CAPS0805        C15             CC0805          10n             Capacitor
     1 CAPS0805        C12             CC0805          10u             Capacitor
     3 CAPS0805        C1              CC0805          22p             Capacitor
                       C2
                       C19
     1 CAPS0805        C20             CC0805          22u MUL CER     Capacitor
     4 CAPS0805        C9              CC0805          100n            Capacitor
                       C10
                       C11
                       C13
     7 CAPS0805        C3              CC0805          100nF           Capacitor
                       C4
                       C5
                       C6
                       C7
                       C8
                       C21
     1 CAPS0805        C14             CC0805          450n            Capacitor
     1 CONNB1X8        J5              CONN1X8
     1 CONNB1X14       J3              CONN1X14        {Value}
     1 CONNB1X14       J4              CONN1X14
     1 CONNB2X8        J1              CONN2X8
     1 CRYSTAL-UP      X1              CRYSTAL-UP      10 MHz
     1 L-1210          L1              L1210           2.2u            Self de choc SM
                                                                       D - Murata
     1 LED0603         LD5             LED SMD0603     B               Led smd - Stanl
                                                                       ey Distrelec N°
                                                                        25 31 00
     1 LED0603         LD3             LED SMD0603     O               Led smd - Stanl
                                                                       ey Distrelec N°
                                                                        25 31 00
     2 LED0603         LD1             LED SMD0603     R               Led smd - Stanl
                                                                       ey Distrelec N°
                                                                        25 31 00
                       LD4
     1 LED0603         LD2             LED SMD0603     Y               Led smd - Stanl
                                                                       ey Distrelec N°
                                                                        25 31 00
     1 LISY300AL       U3              LGA28-0.8MM-7X7
                                       MM              LISY300AL       +2.7V to +5.5V,
                                                                       140uA,  Rail-to
                                                                       -Rail Output 8-
                                                                       Bit DAC - Analo
                                                                       g Devices
     1 LM6142BIM       U2              SO8             LM393           LM6142 Dual Hig
                                                                       ht Speed/Low Po
                                                                       wer 17 MHz Rail
                                                                       -to-Rail Input-
                                                                       Output Operatio
                                                                       nal Amplifiers
     1 LTC3539EDCB-X   U4              DFN8 2X3 P0.45  LTC3539
----------------------------------------------------------------------

12-Jun-09  09:50                                        Page    1
```

```
P-CAD Bill of Materials       I:\...\Cube-Cage_MCU and Measurement.sch

======================================================================
     1 OPA4350         U5              SO14            OPA4354         High-Speed, Sin
                                                                       gle-Supply, Rai
                                                                       l-to-Rail Opera
                                                                       tional Amplifie
                                                                       r - Burr-Brown
     1 PIC18F6585-I/PT
                       U1              TQFP64          PIC18LF6585     8-Bit CMOS Flas
                                                                       h Microcontroll
                                                                       er
     6 RS0805          R3              RC0805          1M
                       R7
                       R8
                       R10
                       R11
                       R15
     4 RS0805          R23             RC0805          2.2k
                       R24
                       R25
                       R26
    10 RS0805          R4              RC0805          5k
                       R5
                       R12
                       R13
                       R17
                       R18
                       R19
                       R20
                       R21
                       R22
     1 RS0805          R14             RC0805          9.5k
     1 RS0805          R1              RC0805          10k
     1 RS0805          R2              RC0805          160k
     1 RS0805          R16             RC0805          309k
     2 RS0805          R6              RC0805          820k
                       R9
     1 RS0805          R27             RC0805          900
----------------------------------------------------------------------

12-Jun-09  09:50                                        Page    2
```

```
======================================================================

Count  ComponentName   RefDes          PatternName     Value           Description
------ --------------- --------------- --------------- --------------- ---------------

     1 ANTENNA_2.45GHZ
       _3216           A1              ANTENNA_2.45GHZ
                                        _3216          50ohm, RFI/O
     1 CAPS0603        C2              RC0603          1nF             Capacitor
     1 CAPS0603        C5              RC0603          1pF             Capacitor
     2 CAPS0603        C10             RC0603          1uF             Capacitor
                       C11
     1 CAPS0603        C4              RC0603          1.5pF           Capacitor
     1 CAPS0603        C6              RC0603          2.2nF           Capacitor
     1 CAPS0603        C7              RC0603          4.7pF           Capacitor
     1 CAPS0603        C1              RC0603          10nF            Capacitor
     2 CAPS0603        C8              RC0603          22pF            Capacitor
                       C9
     1 CAPS0603        C3              RC0603          33nF            Capacitor
     2 CONNB1X14       J2              CONN1X14
                       J3
     1 CONNB2X13PRINT  J1              CONN2X13PRINT
     1 CRYSTAL-CS10    X1              CRYSTAL-CS10    16MHz
     1 D-MELF          D1              MELF-DIODE      {Value}         Diode SMD boiti
                                                                       er MELF 5 x 2.5
                                                                       mm
     1 L-0603          L3              L0603           2.7nH           Self de choc SM
                                                                       D - Murata
     1 L-0603          L1              L0603           3.9nH           Self de choc SM
                                                                       D - Murata
     1 L-0603          L2              L0603           8.2nH           Self de choc SM
                                                                       D - Murata
     1 NRF24L01        U1              QFN20 4X4       {Value}         +2.7V to +5.5V,
                                                                       140uA,  Rail-to
                                                                       -Rail Output 8-
                                                                       Bit DAC - Analo
                                                                       g Devices
     1 RS0603          R2              RC0603          1M
     1 RS0603          R1              RC0603          22K
     1 XC6206PXXP      U3              SOT-89-3        XC62FP3302      High-Precision
                                                                       Voltage Regulat
                                                                       or - Seiko Inst
                                                                       ruments
```

```
======================================================================

Count  ComponentName   RefDes          PatternName     Value           Description
------ --------------- --------------- --------------- --------------- ---------------

     1 AEDR-8100       U8              AEDR-8100       AEDR-8100-1P2   Optocoupler
     1 BAR1X2          J1              BAR1X2          {Value}
     2 BAR1X2          J4              BAR1X2
                       J5
     1 BAR2X8MD        J7              BAR2X8-MD       BAR2x8
     1 CAP2.5MM-DIAM6.
       3               C2              CAP2.5MM-DIAM6.
                                       3               47u
     1 CAP800          C1              CAP800          39n             Capacitor
     1 CAPS0805        C3              CC0805          1u              Capacitor
     1 CAPS0805        C4              CC0805          2.2u MUL CER    Capacitor
     7 CAPS0805        C5              CC0805          100n            Capacitor
                       C6
                       C7
                       C8
                       C9
                       C10
                       C11
     1 CONN1X3         J2              CONN1X3         {Value}
     1 CONNB2X8        J6              CONN2X8
     4 D400            D6              D400            1N4004
                       D7
                       D8
                       D9
     4 D400            D1              D400            MUR160
                       D2
                       D3
                       D4
     1 D600            D5              D600            ZENER 7.5V 5W
     1 DC10B           J3              DC10B           {Value}
     3 IRF7416         Q1              SO8             IRF7416         Exfet Power Mos
                                                                       fet
                       Q3
                       Q5
     2 IRF7455         Q2              SO8             IRF7413         Exfet Power Mos
                                                                       fet
                       Q4
     1 LED0603         LD1             LED SMD0603     YELLOW 2mA      Led smd - Stanl
                                                                       ey Distrelec N°
                                                                        25 31 00
     1 LTC4063         U1              DFN10 3X3
     2 NC7SZ57         U4              SC70            SN74LVC1G57
                       U5
     4 NC7SZ57         U2              SC70            SN74LVC1G58
                       U3
                       U6
                       U7
     1 RS0805          R2              RC0805          1.6k
     1 RS0805          R1              RC0805          2.5k
     1 RS0805          R4              RC0805          160k
     1 RS0805          R5              RC0805          220
     1 RS0805          R3              RC0805          500k
     1 SW-SCH3-01-2P   S1              SWITCH-02       {Value}
```

```
P-CAD Bill of Materials           I:\...\Cube_Altitude Measurement.sch

 ======================================================================

 Count  ComponentName   RefDes          PatternName     Value           Description
 ------ --------------- --------------- --------------- --------------- ---------------

      2 BAR1X5          J1              BAR1X5
                        J2
      2 BSS84           Q2              SOT23           BSS84           N-Channel Enhan
                                                                        cement Mode Ver
                                                                        tical D-MOS Tra
                                                                        nsistor
                        Q6
      2 BSS138          Q3              SOT23           BSS138          N-Channel Logic
                                                                         Level Enhancem
                                                                        ent Mode Field
                                                                        Effect Transist
                                                                        or
                        Q5
      2 CAPS0603        C5              RC0603          1n              Capacitor
                        C8
      1 CAPS0603        C2              RC0603          1u              Capacitor
      1 CAPS0603        C1              RC0603          47n             Capacitor
      1 CAPS0603        C4              RC0603          47p             Capacitor
      2 CAPS0603        C3              RC0603          100n            Capacitor
                        C6
      3 CONNB1X2        J4              CONN1X2
                        J5
                        J6
      1 CONNB1X3        J3              CONN1X3
      1 IC-WK           U1              SO8
      1 IRF7416         Q1              SO8             IRF7416         Exfet Power Mos
                                                                        fet
      1 IRF7455         Q4              SO8             IRF7413         Exfet Power Mos
                                                                        fet
      1 LM6142BIM       U5              SO8             LM393N          LM6142 Dual Hig
                                                                        ht Speed/Low Po
                                                                        wer 17 MHz Rail
                                                                        -to-Rail Input-
                                                                        Output Operatio
                                                                        nal Amplifiers
      1 LMV7239         U3              SOT23-5         LMV7239
      1 NC7SZ57         U2              SC70            SN74LVC1G58
      1 OPA2350         U4              SO8             OPA2364         OPA2350 high-sp
                                                                        eed  single, ra
                                                                        il to rail, amp
                                                                        li op
      6 RS0603          R4              RC0603          1M
                        R5
                        R6
                        R10
                        R11
                        R12
      1 RS0603          R2              RC0603          3k
      2 RS0603          R8              RC0603          5k
                        R14
      2 RS0603          R3              RC0603          200k
                        R9
      1 RS0603          R7              RC0603          430
      1 RS0603          R1              RC0603          500





 ----------------------------------------------------------------------
 12-Jun-09  09:47                                            Page    1
```

Appendix E

Source Code

```c
/******************************************************************************/
/******************************************************************************/
/** FILENAME        : code_cage.c                                           **/
/** FUNCTION        : Main program for the base station of the              **/
/**                   'Inertial Platform for CubeSat' project               **/
/** --------------------------------------------------------------------- **/
/** DATE            : 12.06.2009                                            **/
/** AUTHOR          : Maarten Craeynest, Dries Van de Winkel                **/
/** VERSION         : V1.0                                                  **/
/** DESCRIPTION     : This program takes care of the main tasks to be       **/
/**                   performed in the base station, except the functions   **/
/**                   for the transceiver (nRF) module. The power induction **/
/**                   signal is generated, the levitation PWM signal,       **/
/**                   and a user interface is served. This program is       **/
/**                   automatically synchronized with the control loop in   **/
/**                   the cube via the wireless communication. This program **/
/**                   waits for data from the cube, switches to sendmode    **/
/**                   when that data has arrived, sends out its commands,   **/
/**                   and than switches back to receiving mode. It then     **/
/**                   keeps waiting for the next set of data to arrive.     **/
/** --------------------------------------------------------------------- **/
/** PROCESSOR       : PIC18LF6585                                           **/
/** --------------------------------------------------------------------- **/
/** USES            : pic18.h                                               **/
/**                   pic18fxx8x.h                                          **/
/**                   define_ports_cage.h                                   **/
/**                   define_communication.h                                **/
/**                   declare_functions_NRF.h                               **/
/**                   nrf.c                                                 **/
/** --------------------------------------------------------------------- **/
/** USED BY         : /                                                     **/
/******************************************************************************/
/******************************************************************************/

//includes & defines
#include <pic18.h>
#include <pic18fxx8x.h>
#include "define_ports_cage.h"
#include "define_communication.h"
#include "declare_functions_NRF.h"

#define     IN      1
#define     OUT     0

//method predeclaration
void init();
void start();

void analyseReceivedData();
void setDataToSend();
void sendDataToCube();

void wait(unsigned long delay);

//variable declaration
unsigned char PIswSelHigh = 0xFF, PIswSelLow = 0x26;    // resonance frequency of the P
I circuit is 24.65kHz when cube and levitation coil are present

unsigned char NRF_output_array[PACKET_LENGTH]; // array for the outgoing data for wirel
ess communication
unsigned char NRF_input_array[PACKET_LENGTH];  // array for the incoming data for wirel
ess communication

unsigned int bounceCounter = 0; // a counter used to overcome the bounce of the buttons

//softwareflags
unsigned char stopFlag = 0;
unsigned char rightFlag = 0;
```

```c
unsigned char leftFlag = 0;
unsigned char upFlag = 0;
unsigned char downFlag = 0;
unsigned char buttonFlag = 0;
unsigned char ignoreButton = 0;

/**********************************************************************
 * FUNCTION         : HP_ISR()                                        *
 * OVERVIEW         : service routine for the high priority           *
 *                    interrupts                                      *
 **********************************************************************/
void interrupt HP_ISR()
{
    //Signal for Power Induction (must be a precise freq: high priority)
    if(TMR0IF == 1)             // this timer gives the correct frequency for PI_SW_SEL
, so the resonance frequency for the power induction
    {
        TMR0IF = 0;             // clear Timer0 overflow flag

        TMR0H = PIswSelHigh;    // | reset timer count to start a new period
        TMR0L = PIswSelLow;     // |
        PI_SW_SEL = !PI_SW_SEL; // toggle the output bit (=select the other pair of swi
tches)
    }

    //Button
    if(INT0IF == 1) // 'stop' button
    {
        INT0IF = 0;
        if(ignoreButton == 0)
        {
            buttonFlag = 1;
            stopFlag = 1;
            bounceCounter = 0;
        }
    }
}

/**********************************************************************
 * FUNCTION         : LP_ISR()                                        *
 * OVERVIEW         : service routine for the low priority            *
 *                    interrupts                                      *
 **********************************************************************/
void interrupt low_priority LP_ISR()
{
    //Buttons
    if(INT1IF == 1) // 'up' button
    {
        INT1IF = 0;
        if(ignoreButton == 0)
        {
            buttonFlag = 1;
            upFlag = 1;
            bounceCounter = 0;
        }
    }
    if(INT2IF == 1) // 'down' button
    {
        INT2IF = 0;
        if(ignoreButton == 0)
        {
            buttonFlag = 1;
            downFlag = 1;
            bounceCounter = 0;
        }
    }
    if(RBIF == 1)   // 'left' and 'right' button
    {
        RBIF = 0;
        if(ignoreButton == 0)
        {
            buttonFlag = 1;
            if(RB4 == 0)    // 'left' button
            {
                leftFlag = 1;
            }
            if(RB5 == 0)    // 'right' button
            {
                rightFlag = 1;
            }
            bounceCounter = 0;
        }
    }
}

 /**********************************************************************
 * FUNCTION         : init()                                           *
 * OVERVIEW         : initialise ports, peripheral modules             *
 *                    and interrupts                                   *
 ***********************************************************************/
void init()
{
    ////////////////////////////////////////////////////////////
    // Port directions and default pin values for the outputs //
    ////////////////////////////////////////////////////////////

    //Power Induction
    PI_SW_SEL = 0;
    PI_SW_SEL_DIR = OUT;
    PI_PWM = 0;
    PI_PWM_DIR = OUT;

    //Levitation
    LEVIT_PWM = 0;
    LEVIT_PWM_DIR = OUT;

    //Buttons
    BUTTON_DOWN_DIR = IN;
    BUTTON_UP_DIR = IN;
    BUTTON_LEFT_DIR = IN;
    BUTTON_RIGHT_DIR = IN;
    BUTTON_STOP_DIR = IN;

    //Transceiver
    NRF_CE = 0;
    NRF_CE_DIR = OUT;
    NRF_CSN = 1;
    NRF_CSN_DIR = OUT;
    NRF_SDO = 0;
    NRF_SDO_DIR = OUT;
    NRF_SCK = 0;
    NRF_SCK_DIR = OUT;
    NRF_IRQ_UP_DIR = IN;
    TRISF7 = 1;

    //LEDS
    LED_BLUE = 0;
    LED_BLUE_DIR = OUT;
    LED_YELLOW = 0;
    LED_YELLOW_DIR = OUT;
    LED_ORANGE = 0;
    LED_ORANGE_DIR = OUT;
    LED_RED1 = 0;
    LED_RED1_DIR = OUT;
    LED_RED2 = 0;
    LED_RED2_DIR = OUT;
```

```c

    /////////////
    // General // (interrupts enable/disable for some of these modules: see lower)
    /////////////

    // Set up timers
    /////////
        //Timer0 (power induction square wave)
    TMR0ON = 0;     // stop Timer0
    T08BIT = 0;     // Timer0 as 16-bit timer/counter
    T0CS = 0;       // clock source for Timer0 = internal instruction cycle clock
    PSA = 1;        // no prescaler assigned (incorrect in block diagram in the datashe
et)
    TMR0H = 0x00;   // | reset timer count
    TMR0L = 0x00;   // | when writing to TMR register: first write to high, then to low
 byte (vice versa when reading)
    TMR0IF = 0;     // clear Timer0 overflow flag
        //Timer1 (to add PWM to the power induction - if desired)
    TMR1ON = 0;     // stop Timer1
    T1RD16 = 1;     // read/write in one 16-bit operation
    T1CKPS1 = 0;    // | 1:1 prescaler
    T1CKPS0 = 0;    // |
    T1OSCEN = 0;    // disable Timer1 oscillator
    TMR1CS = 0;     // internal clock as clock source
    TMR1H = 0x00;   // | reset timer count
    TMR1L = 0x00;   // | when writing to TMR register: first write to high, then to low
 byte (vice versa when reading)
    TMR1IF = 0;     // clear Timer1 overflow flag
        //Timer2 (used by PWM module for levitation)
    TMR2ON = 0;     // stop Timer2
    T2CKPS1 = 0;    // | 1:1 prescaler (the prescaler influences the PWM period and dut
y cycle)
    T2CKPS0 = 0;    // |_
    T2OUTPS3 = 0;   // |
    T2OUTPS2 = 0;   // | 1:1 postscaler (not really relevant here)
    T2OUTPS1 = 0;   // |
    T2OUTPS0 = 0;   // |
    TMR2 = 0x00;    // reset timer count
    TMR2IF = 0;     // clear Timer2 overflow flag
        //Timer3 (not used)
    TMR3ON = 0;     // stop Timer3

    // Set up CCP modules
    ////////////
        //CCP1
    //CCP1 (enhanced CCP) is disabled at startup, we don't need it in the base station
        //CCP2 (for levitation)
    CCPR2L = 0x00;  // |
    DC2B1 = 0;      // | start with a duty cycle of 0
    DC2B0 = 0;      // |_
    CCP2M3 = 1;     // | CCP2 module in PWM mode
    CCP2M2 = 1;     // |
    PR2 = 0xC0;     // PWM period (PWM frequency higher than 10kHz)

    // Set up external interrupts
    ///////////
        //PortB Pull-up
    RBPU = 0;       // enable internal weak pull-ups for portB (pins that are defined a
s input have the weak pull-up enabled)
        //External interrupt 0 (stop button)
    INTEDG0 = 0;    // interrupt on falling edge
    INT0IF = 0;     // clear ext int 0 flag
        //External interrupt 1 (up button)
    INTEDG1 = 0;    // interrupt on falling edge
    INT1IF = 0;     // clear ext int 1 flag
        //External interrupt 2 (down button)
    INTEDG2 = 0;    // interrupt on falling edge
    INT2IF = 0;     // clear ext int 2 flag
        //External interrupt 3 (NRF_IRQ_UP - transceiver, but interrupt not enabled (ac
tive polling of INT3IF))
    INTEDG3 = 0;    // interrupt on falling edge
    INT3IF = 0;     // clear ext int 3 flag
        //RB Port Change interrupt (left and right button)
    RBIF = 0;       // clear portB change flag

    // Set up MSSP module
    ////////////////////
    CKP = 0;    // idle state for clock is a low level
    CKE = 1;    // data transmitted on rising edge of serial clock SCK (dependant on Cl
ock Polarity, see CKP bit in SSPCON1 register)
    SMP = 1;    // data sampled at end of data output time (dependant on SPI Master or
Slave Mode, see SSPM bits in SSPCON1 register)
    WCOL = 0;   // clear collision flag
    SSPEN = 1;  // enable serial port
    SSPM3 = 0;  // | SPI Master mode
    SSPM2 = 0;  // |_
    SSPM1 = 0;  // | clock = Fosc/16
    SSPM0 = 1;  // |


    ///////////////
    // Interrupts //
    ///////////////

        //General (the Global Interrupt Enable bit is set/cleared in main() )
    PEIE = 1;       // enable all unmasked peripheral interrupts
    IPEN = 1;       // enable priority levels on interrupts
        //Interrupt enable/disable
    TMR0IE = 1;     // enable Timer0 overflow interrupt
    TMR1IE = 0;     // disable Timer1 overflow interrupt
    TMR2IE = 0;     // disable Timer2 overflow interrupt
    TMR3IE = 0;     // disable Timer3 overflow interrupt
    INT0IE = 1;     // enable ext int 0
    INT1IE = 1;     // enable ext int 1
    INT2IE = 1;     // enable ext int 2
    INT3IE = 0;     // disable ext int 3 (active polling of the INT3IF)
    RBIE = 1;       // enable portB change interrupt
    ADIE = 0;       // disable A/D converter interrupt
    CMIE = 0;       // disable comparator interrupt
        //Priorities
    TMR0IP = 1;     // Timer0 int has high priority
    TMR1IP = 1;     // Timer1 int has high priority
    TMR2IP = 0;     // Timer2 int has low priority
    TMR3IP = 0;     // Timer3 int has low priority
                    // ext int 0 has always high priority
    INT1IP = 0;     // ext int 1 has low priority
    INT2IP = 0;     // ext int 2 has low priority
    INT3IP = 0;     // ext int 3 has low priority
    RBIP = 0;       // RB port change int has low priority
    ADIP = 0;       // A/D converter int has low priority
    CMIP = 0;       // comparator int has low priority


    ///////////////////////////
    // Disable unused modules //
    ///////////////////////////
    PSPMODE = 0;    // disable Parallel Slave Port mode, portD is general purpose I/O
}

/********************************************************************
 * FUNCTION         : start()                                       *
 * OVERVIEW         : start the needed modules                      *
 ********************************************************************/
void start()
{
```

```c
    LED_BLUE = 1;   // to indicate 'start'

    TMR0ON = 1;     // start Timer0 (for power induction)
    PI_PWM = 1;     // corresponds to a PWM signal with duty cycle 100% for the power i
nduction (no current controlling) (use Timer1 when current controlling is desired)

    CCPR2L = 0x00;  // duty cycle of levitation PWM starts at zero
    TMR2ON = 1;     // enable Timer2 (to enable PWM for levitation)
}

/********************************************************************
 * FUNCTION         : analyseReceivedData()                         *
 * OVERVIEW         : get incoming data from the transceiver and    *
 *                    place it in the input array, then analyse     *
 ********************************************************************/
void analyseReceivedData()
{
    // get data from transceiver
    NRF_CE = 0;
    NRF_RxData(NRF_input_array);    // read the data from the RX FIFO

    NRF_output_array[0] = 0x4E;     // clear interrupt RX_DR
    NRF_W_Reg(STATUS, NRF_output_array, 1);

    // analysa data
    if(NRF_input_array[DEST_ADDRESS_INDEX] == CAGE_ADDRESS)
    {
        if(NRF_input_array[CAGE_PI_INDEX] == PI_ON)
        {
            TMR0ON = 1; // enable power induction
        }
        if(NRF_input_array[CAGE_PI_INDEX] == PI_OFF)
        {
            TMR0ON = 0;     // | disable power induction
            PI_SW_SEL = 0;  // |
        }

        CCPR2L = NRF_input_array[CAGE_LEVIT_H_INDEX]; // set correct duty cycle for lev
itation PWM signal
    }
}

/********************************************************************
 * FUNCTION         : setDataToSend()                               *
 * OVERVIEW         : collect data that needs to be sent, and place *
 *                    it in the array of outgoing data. Determines  *
 *                    if and which button was pressed, while making *
 *                    sure that the bounce of the buttons gives no  *
 *                    erroneous commands                            *
 *                                                                  *
 ********************************************************************/
void setDataToSend()
{
    NRF_output_array[CUBE_LEVIT_INDEX] = NO_CMD;
    NRF_output_array[CUBE_ROT_INDEX] = NO_CMD;

    bounceCounter++;
    if(bounceCounter == 300) // if a button is pressed, all buttons are ignored for the
next 300ms
    {
        bounceCounter = 0;
        ignoreButton = 0;
    }

    if(ignoreButton == 0 && buttonFlag == 1)
    {
        if(stopFlag == 1)
        {
            ignoreButton = 1;
            NRF_output_array[CUBE_ROT_INDEX] = STOP_CMD;
            LED_YELLOW = !LED_YELLOW;
            stopFlag = 0;
        }

        if(upFlag == 1)
        {
            ignoreButton = 1;
            NRF_output_array[CUBE_LEVIT_INDEX] = UP_CMD;
            LED_YELLOW = !LED_YELLOW;
            upFlag = 0;
        }

        if(downFlag == 1)
        {
            ignoreButton = 1;
            NRF_output_array[CUBE_LEVIT_INDEX] = DOWN_CMD;
            LED_YELLOW = !LED_YELLOW;
            downFlag = 0;
        }

        if(leftFlag == 1)
        {
            ignoreButton = 1;
            NRF_output_array[CUBE_ROT_INDEX] = LEFT_CMD;
            LED_YELLOW = !LED_YELLOW;
            leftFlag = 0;
        }

        if(rightFlag == 1)
        {
            ignoreButton = 1;
            NRF_output_array[CUBE_ROT_INDEX] = RIGHT_CMD;
            LED_YELLOW = !LED_YELLOW;
            rightFlag = 0;
        }

        buttonFlag = 0;
    }
}

/*********************************************************************
 * FUNCTION         : sendDataToCube()                               *
 * OVERVIEW         : set the correct source and destinations address *
 *                    and send the collected data to the cube         *
                                                        *
 *********************************************************************/
void sendDataToCube()
{
    int i;

    NRF_output_array[SRC_ADDRESS_INDEX] = CAGE_ADDRESS;
    NRF_output_array[DEST_ADDRESS_INDEX] = CUBE_ADDRESS;

    NRF_TxData(NRF_output_array);
    NRF_CE = 1;

    for(i=0;i<=12;i++)
    {
        //NRF_CE high for 15us (must be at least 10us)
    }

    NRF_CE = 0;
}

/*********************************************************************
 * FUNCTION         : wait(delay)                                    *
```

```c
 * OVERVIEW         : active waiting - duration determined by 'delay' *
 *********************************************************************/
void wait(unsigned long delay)
{
    unsigned long delayCounter = 0;

    while(delayCounter < delay)
    {
        delayCounter++;
    }
}

/*********************************************************************
 * FUNCTION         : main()                                         *
 * OVERVIEW         : main function of the program. Calls the        *
 *                    functions above, and synchronises with the     *
 *                    cube via wireless communication                *
                                                        *
 *********************************************************************/
void main()
{
    GIE = 0;                // disable all interrupts

    init();                 // init ports, interrupts, timers,...
    Init_NRF_RX();          // init transceiver as receiver

    start();                // start the necessacery modules

    GIEH = 1;               // |
    GIEL = 1;               // | enable unmasked interrupts, both low and high priority

    while(1)
    {
        // go to receiver mode
        LED_RED1 = 1;
        NRF_FlushRX();
        NRF_output_array[0] = 0x1B;     // as receiver
        NRF_W_Reg(CONFIG, NRF_output_array, 1);
        NRF_CE = 1;


        while(INT3IF == 0)
        {
            // wait for data to arrive
        }
        if(INT3IF == 1)
        {
            INT3IF = 0;
            analyseReceivedData();
        }
        LED_RED1 = 0;


        // go to sendmode
        LED_ORANGE = 1;
        NRF_FlushTX();
        NRF_output_array[0] = 0x1A;     // as sender
        NRF_W_Reg(CONFIG, NRF_output_array, 1);
        NRF_CE = 0;

        setDataToSend();

        sendDataToCube();
        while(INT3IF == 0)
        {
            // wait until sure that data is sent
        }
        if(INT3IF == 1)
        {
            INT3IF = 0;
            NRF_output_array[0] = 0x2E;     // clear interrupt TX_DS
            NRF_W_Reg(STATUS, NRF_output_array, 1);
        }
        LED_ORANGE = 0;
    }
}
```

```
/*****************************************************************************/
/*****************************************************************************/
/** FILENAME        : define_ports_cage.h                                   **/
/** FUNCTION        : defines for the cage source code                      **/
/** ----------------------------------------------------------------------- **/
/** DATE            : 12.06.2009                                            **/
/** AUTHOR          : Maarten Craeynest, Dries Van de Winkel                **/
/** VERSION         : V1.0                                                  **/
/** DESCRIPTION     : /                                                     **/
/** ----------------------------------------------------------------------- **/
/** USES            : /                                                     **/
/** ----------------------------------------------------------------------- **/
/** USED BY         : code_cage.c                                           **/
/*****************************************************************************/
/*****************************************************************************/

//meaningful names for used pins
#define     PI_SW_SEL           RE5
#define     PI_PWM              RE6

#define     LEVIT_PWM           RC1

#define     BUTTON_STOP         RB0
#define     BUTTON_UP           RB1
#define     BUTTON_DOWN         RB2
#define     BUTTON_LEFT         RB4
#define     BUTTON_RIGHT        RB5

#define     LED_RED1            RG0
#define     LED_BLUE            RG1
#define     LED_ORANGE          RD0
#define     LED_YELLOW          RD1
#define     LED_RED2            RD2


//similar names for direction registers of these pins
#define     PI_SW_SEL_DIR           TRISE5
#define     PI_PWM_DIR              TRISE6

#define     LEVIT_PWM_DIR           TRISC1

#define     BUTTON_STOP_DIR         TRISB0
#define     BUTTON_UP_DIR           TRISB1
#define     BUTTON_DOWN_DIR         TRISB2
#define     BUTTON_LEFT_DIR         TRISB4
#define     BUTTON_RIGHT_DIR        TRISB5

#define     LED_RED1_DIR            TRISG0
#define     LED_BLUE_DIR            TRISG1
#define     LED_ORANGE_DIR          TRISD0
#define     LED_YELLOW_DIR          TRISD1
#define     LED_RED2_DIR            TRISD2
```

```c
/*****************************************************************************/
/*****************************************************************************/
/** FILENAME        : code_cube.c                                          **/
/** FUNCTION        : Main program for the cube of the                     **/
/**                   'Inertial Platform for CubeSat' project              **/
/** -------------------------------------------------------------------- **/
/** DATE            : 12.06.2009                                           **/
/** AUTHOR          : Maarten Craeynest, Dries Van de Winkel               **/
/** VERSION         : V1.0                                                 **/
/** DESCRIPTION     : This program takes care of the main tasks to be      **/
/**                   performed in the cube, except the functions          **/
/**                   for the transceiver (nRF) module. Calculations       **/
/**                   concerning the battery status, distance to the base  **/
/**                   station and rotation are done. Data is sent to the   **/
/**                   base station, then the cube waits for commands to    **/
/**                   arrive from the base station. The control loop cycle **/
/**                   of calculations, sending, and receiving restarts     **/
/**                   every 1 ms.                                          **/
/** -------------------------------------------------------------------- **/
/** PROCESSOR       : PIC18LF6585                                          **/
/** -------------------------------------------------------------------- **/
/** USES            : pic18.h                                              **/
/**                   pic18fxx8x.h                                         **/
/**                   define_ports_cube.h                                  **/
/**                   define_communication.h                               **/
/**                   declare_functions_NRF.h                              **/
/**                   nrf.c                                                **/
/** -------------------------------------------------------------------- **/
/** USED BY         : /                                                    **/
/*****************************************************************************/
/*****************************************************************************/

//includes & defines
#include <pic18.h>
#include <pic18fxx8x.h>
#include "define_ports_cube.h"
#include "define_communication.h"
#include "declare_functions_NRF.h"

#define     IN      1
#define     OUT     0

//method predeclaration
void init();
void start();

void batteryIndication();
void setARSZero();
void waitForPositioning();

void powerCheck();
void rotation();
void distance();

void analyseReceivedData();
void sendDataToCage();

void wait(unsigned long delay);

//variable declaration
signed int wantedSpeed = 0, currentSpeed, speedError;   // initialise wantedSpeed to ze
ro
signed int speedProp, speedKp = 2, speedCntrlOut;
signed int rotationChangeStep = 15;

signed int wantedDistance, currentDistance, distanceError;
signed int distanceCntrlOut;
signed int distanceChangeStep = 5;
```

```c
unsigned int ARSZero;   //0x7D in tests

unsigned short powerCheckCounter = 0;
static unsigned short CHECK_POWER = 60000;

unsigned char mirrorTimerHigh = 0xFA, mirrorTimerLow = 0xA9;    // the resonance freque
ncy of the mirror is 3.63 kHz
unsigned char sensorTimerHigh, sensorTimerLow;

unsigned char NRF_output_array[PACKET_LENGTH];
unsigned char NRF_input_array[PACKET_LENGTH];

//softwareflags
unsigned char okForSensor2 = 0;

/**********************************************************************
 * FUNCTION         : HP_ISR()                                        *
 * OVERVIEW         : service routine for the high priority           *
 *                    interrupts                                      *
 **********************************************************************/
void interrupt HP_ISR()
{
    //Signal for mirror (must be a precise frequency: high priority)
    if(TMR1IF == 1)
    {
        TMR1IF = 0;
        TMR1H = mirrorTimerHigh;
        TMR1L = mirrorTimerLow;
        MIRROR_CMD = !MIRROR_CMD;
    }
}

/**********************************************************************
 * FUNCTION         : LP_ISR()                                        *
 * OVERVIEW         : service routine for the low priority            *
 *                    interrupts                                      *
 **********************************************************************/
void interrupt low_priority LP_ISR()
{
    //Interrupt from sensor 1 (close to the mirror)
    if(INT1IF == 1)
    {
        INT1IF = 0;     // clear ext int 1 flag

        TMR3H = 0x00;   // | reset timer count
        TMR3L = 0x00;   // |
        okForSensor2 = 1;
    }

    //Interrupt from sensor 2 (far from the mirror)
    if(INT2IF == 1)
    {
        INT2IF = 0;     // clear ext int 2 flag

        if(okForSensor2 == 1) //The current timercount is only valid if the previous se
nsor-interrupt was from sensor 1.
        {
            sensorTimerLow = TMR3L;     // | save timer count
            sensorTimerHigh = TMR3H;    // |
            okForSensor2 = 0;
        }
    }
}

/**********************************************************************
 * FUNCTION         : init()                                          *
 * OVERVIEW         : initialise ports, peripheral modules            *
 *                    and interrupts                                     *
 ***********************************************************************/
void init()
{
    ////////////////////////////////////////////////////////////
    // Port directions and default pin values for the outputs //
    ////////////////////////////////////////////////////////////

    //Power Induction detect and battery check pins
    PI__DETECT_DIR = IN;
    VBAT_DIV_Z_DIR = IN;

    //Charger
    CHARGER__CHGEN = 0; // at startup charging is enabled
    CHARGER__CHGEN_DIR = OUT;
    CHARGER__LDOEN = 0; // activate LDO (3.3V) at startup
    CHARGER__LDOEN_DIR = OUT;
    CHARGER_CHRG_LIM_DIR = IN;

    //Step-up converter
    STEPUP_MODE = 0;    // fixed frequency stepup mode (for rather high load current)

    STEPUP_MODE_DIR = OUT;  // output

    //Angular Rate Sensor
    ARS_PD = 1; // ARS disabled at startup (PD = power down), will be enabled when need
ed
    ARS_PD_DIR = OUT;
    ARS_ST = 0; // ST = system test, only for testing purposes, so disabled at startup
    ARS_ST_DIR = OUT;
    ARS_OUT_Z_DIR = IN;
    VREFP_Z = IN;   // | alternate reference voltages for the ADC
    VREFN_Z = IN;   // |

    //Optical Encoder
    OE__EN = 1;         // disabled at startup (optical encoder for testing purposes)
    OE__EN_DIR = OUT;
    OE__CHA_UP_DIR = IN;
    OE__CHB_UP_DIR = IN;

    //Motor
    MOTOR_HB1 = 0;  // | 'free state' at startup
    MOTOR_HB2 = 0;  // |
    MOTOR_HB1_DIR = OUT;
    MOTOR_HB2_DIR = OUT;
    MOTOR_PWM = 0;
    MOTOR_PWM_DIR = OUT;

    //Mirror and laser
    MIRROR_CMD = 0;
    MIRROR_CMD_DIR = OUT;
    MIRROR_EN = 0;
    MIRROR_EN_DIR = OUT;
    LASER__EN = 1;      // laser disabled at startup (active-low)
    LASER__EN_DIR = OUT;
    PD1_COMP_DIR = IN;
    PD2_COMP_DIR = IN;
    MIRROR_FB_DIR = IN;

    //Transceiver
    NRF_CE = 0;
    NRF_CE_DIR = OUT;
    NRF_CSN = 1;
    NRF_CSN_DIR = OUT;
    NRF_SDO = 0;
    NRF_SDO_DIR = OUT;
    NRF_SCK = 0;
    NRF_SCK_DIR = OUT;
```

```c
    NRF_IRQ_UP_DIR = IN;
    TRISF7 = 1;

    //LEDS
    LED_BLUE = 0;
    LED_BLUE_DIR = OUT;
    LED_YELLOW = 0;
    LED_YELLOW_DIR = OUT;
    LED_ORANGE = 0;
    LED_ORANGE_DIR = OUT;
    LED_RED1 = 0;
    LED_RED1_DIR = OUT;
    LED_RED2 = 0;
    LED_RED2_DIR = OUT;

    //MIXINOUT
    MIX_INOUT1_DIR = IN;    // | Those pins are not used in the cube, but they must be
input!! (otherwise, they disturb the incoming optical encoder channels A and B)
    MIX_INOUT2_DIR = IN;    // |


    /////////////
    // General // (interrupts enable/disable for some of these modules: see lower)
    /////////////

    // Set up timers
    /////////
        //Timer0 (control loop timer -- 1ms)
    TMR0ON = 0;     // stop Timer0
    T08BIT = 0;     // Timer0 as 16-bit timer/counter
    T0CS = 0;       // clock source for Timer0 = internal instruction cycle clock
    PSA = 1;        // no prescaler assigned (incorrect in block diagram in the datashe
et)
    TMR0H = 0x00;   // | reset timer count
    TMR0L = 0x00;   // | when writing to TMR register: first write to high, then to low
 byte (vice versa when reading)
    TMR0IF = 0;     // clear Timer0 overflow flag
        //Timer1 (mirror command and optional enable+feedback)
    TMR1ON = 0;     // stop Timer1
    T1RD16 = 1;     // read/write in one 16-bit operation (to do that: first write to h
igh, then to low byte (vice versa when reading))
    T1CKPS1 = 0;    // | 1:1 prescaler
    T1CKPS0 = 0;    // |
    T1OSCEN = 0;    // disable Timer1 oscillator
    TMR1CS = 0;     // internal clock as clock source
    TMR1H = 0x00;   // | reset timer count
    TMR1L = 0x00;   // | when writing to TMR register: first write to high, then to low
 byte (vice versa when reading)
    TMR1IF = 0;     // clear Timer1 overflow flag
        //Timer2 (used by PWM module for motor)
    TMR2ON = 0;     // stop Timer2
    T2CKPS1 = 0;    // | 1:1 prescaler (the prescaler influences the PWM period and dut
y cycle)
    T2CKPS0 = 0;    // |_
    T2OUTPS3 = 0;   // |
    T2OUTPS2 = 0;   // | 1:1 postscaler (not really relevant here)
    T2OUTPS1 = 0;   // |
    T2OUTPS0 = 0;   // |
    TMR2 = 0x00;    // reset timer count
    TMR2IF = 0;     // clear Timer2 overflow flag
        //Timer3 (distance sensors)
    TMR3ON = 0;     // stop Timer3
    T3RD16 = 1;     // read/write in one 16-bit operation (to do that: first write to h
igh, then to low byte (vice versa when reading))
    T3CKPS1 = 0;    // | 1:1 prescaler
    T3CKPS0 = 0;    // |
    TMR3CS = 0;     // use internal clock (Fosc/4) as clock source
    TMR3IF = 0;     // clear Timer3 overflow flag
    // Set up CCP modules
    ////////////
        //CCP1
    //CCP1 (enhanced CCP) is disabled at startup, we don't need it in the cube
        //CCP2 (for motor)
    CCPR2L = 0x00;  // |
    DC2B1 = 0;      // | start with a duty cycle of 0
    DC2B0 = 0;      // |_
    CCP2M3 = 1;     // | CCP2 module in PWM mode
    CCP2M2 = 1;     // |
    PR2 = 0xCF;     // PWM period

    // Set up external interrupts
    ////////////
        //PortB Pull-up
    RBPU = 1;       // disable internal weak pull-ups for portB
        //External interrupt 0 (mirror feedback)
    INTEDG0 = 1;    // interrupt on rising edge
    INT0IF = 0;     // clear ext int 0 flag
        //External interrupt 1 (sensor 1 comparator output)
    INTEDG1 = 1;    // interrupt on rising edge
    INT1IF = 0;     // clear ext int 1 flag
        //External interrupt 2 (sensor 2 comparator output)
    INTEDG2 = 1;    // interrupt on rising edge
    INT2IF = 0;     // clear ext int 2 flag
        //External interrupt 3 (NRF_IRQ_UP - transceiver, but interrupt not enabled (ac
tive polling of INT3IF))
    INTEDG3 = 0;    // interrupt on falling edge
    INT3IF = 0;     // clear ext int 3 flag
        //RB Port Change interrupt
    RBIF = 0;       // clear portB change flag


    // Set up analog pins, ADC, comparator and voltage reference module
    //////////////////
        //Define which pins are analog and which are digital
    PCFG3 = 0;      // |
    PCFG2 = 1;      // | AN0-AN8 configured as analog
    PCFG1 = 1;      // | AN9-AN11 configured as digital
    PCFG0 = 0;      // |
        //Set up ADC
    ADON = 0;       // ADC module is disabled and consumes no current
    CHS3 = 0;       // |
    CHS2 = 1;       // | select analog channel 5 (channel where ARS_OUTPUT is)
    CHS1 = 0;       // |
    CHS0 = 1;       // |_
    VCFG1 = 0;      // | AVDD and AVSS as reference voltages (at some point, this will
be changed, but only temporary)
    VCFG0 = 0;      // |
    ADFM = 0;       // A/D result left justified
    ACQT2 = 1;      // |                    |
    ACQT1 = 1;      // | A/D acquisition time: 20*Tad
    ACQT0 = 1;      // |_                   | enough time to acquire, but yet less then
 1ms (acquisition time = 20 * Tosc * 64 = 20 * 25ns * 64 = 32us)
    ADCS2 = 1;      // |                    |
    ADCS1 = 1;      // | A/D conversion clock: Fosc/64
    ADCS0 = 0;      // |                    |
    ADIF = 0;       // clear the A/D converter interrupt flag
        //Set up comparator
    CM2 = 1;        // |
    CM1 = 1;        // | comparators off at startup (to save current) (will be enabled
right away)
    CM0 = 1;        // |_
    TRISF4 = OUT;   // |
    TRISF5 = OUT;   // | set the unused comparator pins as output, because we are worki
ng with comparator interrupt, so we don't want these pins floating inputs
    TRISF6 = OUT;   // |
```

```
    CIS = 1;        // when using the '4 muxed input, comparison with internal CVref' m
ode, RF3/AN8 pin is connected to comparator 2 inverting input
    C2INV = 1;      // comparator 2 output is inverted (because our VBAT_DIV_Z pin is c
onnected to the INVERTING input of comp 2: by inverting the output, we undo this fact)
    CMIF = 0;       // clear comparator interrupt flag
        //Set up voltage reference module (used for comparator on VBAT_DIV_Z)
    CVREN = 0;      // disable voltage reference module at startup (to save current) (w
ill be enabled right away)
    CVROE = 0;      // reference voltage disconnected from RF5 pin
    CVRR = 1;       // Vref range selection: between 0*Vdd and 0.625*Vdd, in 24 steps
    CVRSS = 0;      // use Vdd and Vss as rails for voltage divider
    CVR3 = 1;       // |
    CVR2 = 0;       // | value selection = 9 -> Vref = 9/24 * Vsrc = 0.375 * 5 = 1.875
V (which corresponds to battery voltage of 3.75V, because it is divided by two)
    CVR1 = 0;       // |
    CVR0 = 1;       // |

    // Set up MSSP module (SPI operation)
    /////////////////////
    CKP = 0;    // idle state for clock is a low level
    CKE = 1;    // data transmitted on rising edge of serial clock SCK (dependant on Cl
ock Polarity, see CKP bit in SSPCON1 register)
    SMP = 1;    // data sampled at end of data output time (dependant on SPI Master or
Slave Mode, see SSPM bits in SSPCON1 register)
    WCOL = 0;   // clear collision flag
    SSPEN = 1;  // enable serial port
    SSPM3 = 0;  // | SPI Master mode
    SSPM2 = 0;  // |_
    SSPM1 = 0;  // | clock = Fosc/16
    SSPM0 = 1;  // |


    ///////////////
    // Interrupts //
    ///////////////

        //General (the Global Interrupt Enable bit is set/cleared in main() )
    PEIE = 1;       // enable all unmasked peripheral interrupts (see also IPEN bit!!)
    IPEN = 1;       // disable priority levels on interrupts
        //Interrupt enable/disable
    TMR0IE = 0;     // disable Timer0 overflow interrupt (active polling of TMR0IF)
    TMR1IE = 1;     // enable Timer1 overflow interrupt
    TMR2IE = 0;     // disable Timer2 overflow interrupt
    TMR3IE = 0;     // disable Timer3 overflow interrupt
    INT0IE = 0;     // disable ext int 0 (mirror feedback signal not yet needed)
    INT1IE = 1;     // enable ext int 1 (first sensor, PD1_COMP)
    INT2IE = 1;     // enable ext int 2 (second sensor, PD2_COMP)
    INT3IE = 0;     // disable ext int 3 (Nordic transceiver)
    RBIE = 0;       // disable portB change interrupt
    ADIE = 0;       // disable A/D converter interrupt
    CMIE = 0;       // disable comparator interrupt
        //Priorities
    TMR0IP = 0;     // Timer0 int has low priority
    TMR1IP = 1;     // Timer1 int has high priority (precise resonance freq of mirror!)
    TMR2IP = 0;     // Timer2 int has low priority
    TMR3IP = 0;     // Timer3 int has low priority
                    // ext int 0 has always high priority
    INT1IP = 0;     // ext int 1 has low priority
    INT2IP = 0;     // ext int 2 has low priority
    INT3IP = 0;     // ext int 3 has low priority
    RBIP = 0;       // RB port change int has low priority
    ADIP = 0;       // A/D converter int has low priority
    CMIP = 0;       // comparator int has low priority


    ///////////////////////////
    // Disable unused modules //
    ///////////////////////////
    PSPMODE = 0;    // disable Parallel Slave Port mode, portD is general purpose I/O
}

/*********************************************************************
 * FUNCTION         : start()                                        *
 * OVERVIEW         : start the needed modules                       *
 *********************************************************************/
 void start()
{
    LED_BLUE = 1;

    ARS_PD = 0;     // enable Angular Rate Sensor
    ADON = 1;       // enable A/D converter module (in the first control loop cycle, it
 probably will not give a valid value, but that won't be a problem) or just don't look
in first cycle??

    TMR2ON = 1;     // must be on for the PWM module to work (motor)

    LASER__EN = 0;  // enable laser (active-low)
    wait(100);      // wait; or else the laser is not activated properly

    TMR1ON = 1;     // start Timer1 (mirror timer)
    MIRROR_EN = 1;  // enable mirror

    TMR3ON = 1;     // start Timer3 (for distance sensors)
}

/*********************************************************************
 * FUNCTION         : batteryIndication()                            *
 * OVERVIEW         : makes the blue led blink 5 times when battery  *
 *                    charge is OK, red led if too low               *
 *********************************************************************/
void batteryIndication()
{
    int i;

    CVREN = 1;      // enable voltage reference module
    CM0 = 0;        // enable comparator (4 muxed input mode) (there are 3 bits that de
fine the comparator mode, but only this one changes when switching between 'off' and '4
 muxed input mode')

    wait(10000);    // wait for comparator and reference module to become stable

    if(C2OUT == 0)
    {
        for(i = 0; i < 10; i++)
        {
            LED_RED1 = !LED_RED1;  // blink led
            wait(100000);
        }
    }
    else
    {
        for(i = 0; i < 10; i++)
        {
            LED_BLUE = !LED_BLUE;  // blink led
            wait(100000);
        }
    }

    CM0 = 1;        // disable comparators (there are 3 bits that define the comparator
 mode, but only this one changes when switching between 'off' and '4 muxed input mode')
    CVREN = 0;      // disable voltage reference module
}

/*********************************************************************
 * FUNCTION         : setARSZero()                                   *
 * OVERVIEW         : determine the digital word that is equivalent  *
```

```
 *                    to the analog output of the Angular Rate       *
 *                    Sensor when the cube is not turning            *
 *********************************************************************/
void setARSZero()
{
    ARS_PD = 0;     // enable Angular Rate Sensor
    ADON = 1;       // enable A/D converter module

    wait(10000);    // wait for A/D converter to acquire signal

    GODONE = 1;
    while(GODONE == 1)
    {
        // wait for conversion to complete
    }
    ARSZero = ADRESH;  // save result of the conversion

    ADON = 0;
}

/*********************************************************************
 * FUNCTION         : waitForPositioning()                           *
 * OVERVIEW         : wait until cube is brought near the cage,      *
 *                    this is indicated by a detection of the        *
 *                    (active low) power induction detect signal     *
 *********************************************************************/
void waitForPositioning()
{
    while(PI__DETECT == 1)
    {
        LED_ORANGE = !LED_ORANGE;
        wait(200000);               // just wait for PI_DETECT to become 1, while blink
ing the orange LED
    }

    LED_ORANGE = 0;
}

/*********************************************************************
 * FUNCTION         : powerCheck()                                   *
 * OVERVIEW         : check the charge of the battery every 60       *
 *                    seconds (see CHECK_POWER constant). Prepare    *
 *                    command PI 'on' or 'off' to be sent, depending *
 *                    on charging status, detection of power         *
 *                    induction, and charge of the battery.          *
 *********************************************************************/
void powerCheck()
{
    NRF_output_array[CAGE_PI_INDEX] = NO_CMD; // initialise PI command to 'invalid'

    if(powerCheckCounter == CHECK_POWER - 1)  // enable modules in the cycle before the
 cycle where battery is checked (they need some time to become stable)
    {
        CVREN = 1;      // enable voltage reference module
        CM0 = 0;        // enable comparator (there are 3 bits that define the comparat
or mode, but only this one changes when switching between 'off' and '4 muxed input mode
')

        // leave PI command to 'invalid' (don't override)
    }

    if(powerCheckCounter == CHECK_POWER)    // not 'else', because powerCheckCounter co
uld have another value than the two that are checked here (it's not a bool), so if ther
e would be an 'else' here, there would be an 'if' inside anyway to see if the counter =
= ..., so there's really no need to place the 'else'
    {
        if(PI__DETECT == 1)
        {
```

```c
            if(CHARGER_CHRG_LIM == 0)
            {
                NRF_output_array[CAGE_PI_INDEX] = PI_OFF;   // prepare the 'power induc
tion off' command to be sent
            }
        }
        else
        {
            if(C2OUT == 0)
            {
                NRF_output_array[CAGE_PI_INDEX] = PI_ON;   // prepare the 'power induct
ion on' command to be sent
            }
        }

        CM0 = 1;    // disable comparators (there are 3 bits that define the comparator
 mode, but only this one changes when switching between 'off' and '4 muxed input mode')
        CVREN = 0;  // disable voltage reference module

        powerCheckCounter = 0; // reset counter

    }

    powerCheckCounter++;
}

/**********************************************************************
 * FUNCTION         : rotation()                                      *
 * OVERVIEW         : get result of the A/D conversion on the ARS     *
 *                    output. The proportional controller acts on     *
 *                    the motor to adjust the rotational speed        *
 **********************************************************************/
void rotation()
{
    if(GODONE == 0)      // make sure the A/D conversion process is completed before ent
ering the horizontal control function (if not, the rotation is skipped until the next c
ontrol loop cycle)
    {
        currentSpeed = (signed int) (ADRESH - ARSZero);

        speedError = wantedSpeed - currentSpeed;
        speedProp = speedError * speedKp;
        speedCntrlOut = speedProp;

        if(speedCntrlOut < 0)
        {
            MOTOR_HB1 = 0; // | turn left
            MOTOR_HB2 = 1; // |
            CCPR2L = (unsigned char) speedCntrlOut;
        }
        else
        {
            MOTOR_HB1 = 1; // | turn right
            MOTOR_HB2 = 0; // |
            CCPR2L = (unsigned char) (-speedCntrlOut);
        }

        GODONE = 1;      // start the A/D conversion process (by doing this here, it's s
ure that the conversion will be finished in the next control loop cycle) (if this instr
uction would be at the top of the instruction cycle while-loop, it is not sure that the
 2 functions between this instruction and rotation() would be long enough for the job t
o be finished, and then we would have to wait active, which is loss of precious time)

        if(ADRESH > ARSZero)    // the leds indicate direction in which the cube is tur
ning (to check proper working of ARS)
        {
            LED_RED2 = 1;
            LED_YELLOW = 0;
```

```c
        }
        else
        {
            LED_RED2 = 0;
            LED_YELLOW = 1;
        }
    }
}

/**********************************************************************
 * FUNCTION         : distance()                                      *
 * OVERVIEW         : do calculations concerning the distance         *
 *                    and the control loop for levitation. Prepare    *
 *                    data to be sent to base station to act on the   *
 *                    levitation coil (control loop and calculations  *
 *                    yet to be implemented)                          *
 **********************************************************************/
void distance()
{
    // distance calculations using the sensorTimerHigh and sensorTimerLow values
    // distance control value calculation, using wantedDistance and currentDistance

    // send controller output to cage
    NRF_output_array[CAGE_LEVIT_H_INDEX] = distanceCntrlOut;
}

/**********************************************************************
 * FUNCTION         : analyseReceivedData()                           *
 * OVERVIEW         : get incoming data from the transceiver and      *
 *                    place it in the input array, then analyse       *
 **********************************************************************/
void analyseReceivedData()
{
    // get data from transceiver
    NRF_CE = 0;
    NRF_RxData(NRF_input_array);    // read the data from the RX FIFO

    NRF_output_array[0] = 0x4E;     // clear interrupt RX_DR
    NRF_W_Reg(STATUS, NRF_output_array, 1);

    // analysa data
    if(NRF_input_array[DEST_ADDRESS_INDEX] == CUBE_ADDRESS)
    {
        if(NRF_input_array[CUBE_ROT_INDEX] == STOP_CMD)
        {
            wantedSpeed = 0;
        }
        if(NRF_input_array[CUBE_ROT_INDEX] == LEFT_CMD)
        {
            if(wantedSpeed > 0)
            {
                wantedSpeed = (-rotationChangeStep);
            }
            else
            {
                wantedSpeed = wantedSpeed - rotationChangeStep;
            }
        }
        if(NRF_input_array[CUBE_ROT_INDEX] == RIGHT_CMD)
        {
            if(wantedSpeed < 0)
            {
                wantedSpeed = rotationChangeStep;
            }
            else
            {
                wantedSpeed = wantedSpeed + rotationChangeStep;
            }
        }

        if(NRF_input_array[CUBE_LEVIT_INDEX] == UP_CMD)
        {
            wantedDistance = wantedDistance - distanceChangeStep;
        }
        if(NRF_input_array[CUBE_ROT_INDEX] == DOWN_CMD)
        {
            wantedDistance = wantedDistance + distanceChangeStep;
        }
    }
}

/*********************************************************************
 * FUNCTION         : sendDataToCage()                               *
 * OVERVIEW         : set the correct source and destinations address *
 *                    and send the collected data to the cube. Data  *
 *                    is collected in the powerCheck(), rotation(),   *
 *                    and distance() functions                        *
                                                    *
 *********************************************************************/
void sendDataToCage()
{
    int i;

    NRF_output_array[SRC_ADDRESS_INDEX] = CUBE_ADDRESS;
    NRF_output_array[DEST_ADDRESS_INDEX] = CAGE_ADDRESS;

    NRF_TxData(NRF_output_array);
    NRF_CE = 1;

    for(i=0;i<=12;i++)
    {
        //NRF_CE high for 15us (must be at least 10us)
    }

    NRF_CE = 0;
}

/*********************************************************************
 * FUNCTION         : wait(delay)                                    *
 * OVERVIEW         : active waiting - duration determined by 'delay' *
 *********************************************************************/
void wait(unsigned long delay)
{
    unsigned long delayCounter = 0;

    while(delayCounter < delay)
    {
        delayCounter++;
    }
}

/*********************************************************************
 * FUNCTION         : main()                                         *
 * OVERVIEW         : main function of the program. Calls the        *
 *                    functions above, and synchronises the          *
 *                    surrounding communication points (base station) *
 *                    via wireless communication                     *
                                                    *
 *********************************************************************/
void main()
{
    GIE = 0;                // disable ALL interrupts

    init();                 // init ports, interrupts, timers, ...
    Init_NRF_TX();          // init transceiver as sender
```

```
    batteryIndication();    // indication of the charge of the battery
    setARSZero();           // determine center of ARS
    waitForPositioning();   // wait until cube is brought near the base station
    start();                // start the necessacery modules (except for the control lo
op timer)

    GIEH = 1;               // enable all high priority interrupts
    GIEL = 1;               // enable all low priority interrupts
    TMR0ON = 1;             // start the timer that takes care of the 1ms control loop
(this timer has a low priority interrupt, the mirror is more critical)
    // control loop cycle is implemented with active polling, not interrupt driven (pos
sible to do it this way since calculations are finished before 1ms has passed)

    while(1)
    {
        // go to sendmode
        LED_ORANGE = 1;
        NRF_FlushTX();
        NRF_output_array[0] = 0x1A;     // as sender
        NRF_W_Reg(CONFIG, NRF_output_array, 1);
        NRF_CE = 0;

        powerCheck();       // how is the battery charge and charging status
        rotation();         // rotation calculations and adjustment
        distance();         // distance calculations as close as possible to nrf_Tx, so
 that it's results are sent fast enough

        sendDataToCage();
        while(INT3IF == 0 && TMR0IF == 0)
        {
            // wait until sure that data is sent, or control loop cycle is over
        }
        if(INT3IF == 1 && TMR0IF == 0)
        {
            INT3IF = 0;
            NRF_output_array[0] = 0x2E;     // clear interrupt TX_DS
            NRF_W_Reg(STATUS, NRF_output_array, 1);

        }
        LED_ORANGE = 0;


        // go to receiver mode, only if control loop cycle has not passed
        LED_RED1 = 1;
        if(TMR0IF == 0)
        {
            NRF_FlushRX();
            NRF_output_array[0] = 0x1B;     // as receiver
            NRF_W_Reg(CONFIG, NRF_output_array, 1);
            NRF_CE = 1;
        }

        while(INT3IF == 0 && TMR0IF == 0)
        {
            // wait for data to arrive, or for control loop cycle to end
        }
        if(INT3IF == 1 && TMR0IF == 0)
        {
            INT3IF = 0;
            analyseReceivedData();
            while(TMR0IF == 0)
            {
                // wait for rest of control loop cycle
            }
        }
        LED_RED1 = 0;

        TMR0H = 0xD8;   // | 1ms = 1'000'000ns = 10'000*Tcycle so we need to count 10'0
00 times: 0xD8EF is 10'000 counts less than 0xFFFF
        TMR0L = 0xEF;   // | (Instruction cycle = Tcycle = Tosc*4 = 100ns) (Our crystal
 is 10MHz, but multiplied by 4 with PLL, so Tosc = 25ns)
        TMR0IF = 0;
    }
}
```

```c
/*****************************************************************************/
/*****************************************************************************/
/** FILENAME        : define_ports_cube.h                                    **/
/** FUNCTION        : defines for the cube source code                       **/
/** ---------------------------------------------------------------------- **/
/** DATE            : 12.06.2009                                             **/
/** AUTHOR          : Maarten Craeynest, Dries Van de Winkel                 **/
/** VERSION         : V1.0                                                   **/
/** DESCRIPTION     : /                                                      **/
/** ---------------------------------------------------------------------- **/
/** USES            : /                                                      **/
/** ---------------------------------------------------------------------- **/
/** USED BY         : code_cube.c                                            **/
/*****************************************************************************/
/*****************************************************************************/

//meaningful names for used pins
#define     PI__DETECT          RD6
#define     VBAT_DIV_Z          RF3

#define     CHARGER__CHGEN      RD4
#define     CHARGER__LDOEN      RD3
#define     CHARGER_CHRG_LIM    RD7

#define     STEPUP_MODE         RE3

#define     ARS_PD              RG3
#define     ARS_ST              RG2
#define     ARS_OUT_Z           RF0
#define     VREFP_Z             RA3
#define     VREFN_Z             RA2

#define     OE__EN              RD5
#define     OE__CHA_UP          RB4
#define     OE__CHB_UP          RB5

#define     MOTOR_HB1           RC0
#define     MOTOR_HB2           RC6
#define     MOTOR_PWM           RC1

#define     LASER__EN           RE7
#define     MIRROR_CMD          RE1
#define     MIRROR_EN           RE0
#define     MIRROR_FB           RB0
#define     PD1_COMP            RB1
#define     PD2_COMP            RB2

#define     MIX_INOUT1          RE5
#define     MIX_INOUT2          RE6

#define     LED_RED1            RG0
#define     LED_BLUE            RG1
#define     LED_ORANGE          RD0
#define     LED_YELLOW          RD1
#define     LED_RED2            RD2


//similar names for direction registers of these pins
#define     PI__DETECT_DIR          TRISD6
#define     VBAT_DIV_Z_DIR          TRISF3

#define     CHARGER__CHGEN_DIR      TRISD4
#define     CHARGER__LDOEN_DIR      TRISD3
#define     CHARGER_CHRG_LIM_DIR    TRISD7

#define     STEPUP_MODE_DIR         TRISE3

#define     ARS_PD_DIR              TRISG3
```

```c
#define     ARS_ST_DIR              TRISG2
#define     ARS_OUT_Z_DIR           TRISF0
#define     VREFP_Z_DIR             TIRSA3
#define     VREFN_Z_DIR             TRISA2

#define     OE__EN_DIR              TRISD5
#define     OE__CHA_UP_DIR          TRISB4
#define     OE__CHB_UP_DIR          TRISB5

#define     MOTOR_HB1_DIR           TRISC0
#define     MOTOR_HB2_DIR           TRISC6
#define     MOTOR_PWM_DIR           TRISC1

#define     LASER__EN_DIR           TRISE7
#define     MIRROR_CMD_DIR          TRISE1
#define     MIRROR_EN_DIR           TRISE0
#define     MIRROR_FB_DIR           TRISB0
#define     PD1_COMP_DIR            TRISB1
#define     PD2_COMP_DIR            TRISB2

#define     MIX_INOUT1_DIR          TRISE5
#define     MIX_INOUT2_DIR          TRISE6

#define     LED_RED1_DIR            TRISG0
#define     LED_BLUE_DIR            TRISG1
#define     LED_ORANGE_DIR          TRISD0
#define     LED_YELLOW_DIR          TRISD1
#define     LED_RED2_DIR            TRISD2
```

```c
/*----------------------------------------------------------------------------------*/
/* FILENAME   : declare_functions_NRF.h                                              */
/* FUNCTION   : Header file, declares functions to configure and control             */
/*              the nRF24L01                                                         */
/*----------------------------------------------------------------------------------*/
/* DATE       : 18.05.2009                                                           */
/* AUTHOR     : Dries Van de Winkel                                                  */
/*----------------------------------------------------------------------------------*/
/* USED HARDWARE: Nordic nRF24L01                                                    */
/*----------------------------------------------------------------------------------*/
/* PROCESSOR  : PIC18LF6585                                                          */
/*----------------------------------------------------------------------------------*/
/* USES  : /                                                                         */
/*----------------------------------------------------------------------------------*/
/* USED BY  :   nrf.c                                                                */
/*              code_cube.c                                                          */
/*              code_cage.c                                                          */
/*----------------------------------------------------------------------------------*/

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : NRF_FlushTX                                                        */
/*----------------------------------------------------------------------------------*/
void NRF_FlushTX();

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : NRF_FlushRX                                                        */
/*----------------------------------------------------------------------------------*/
void NRF_FlushRX();

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : Init_NRF_TX                                                        */
/*----------------------------------------------------------------------------------*/
void Init_NRF_TX();

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : Init_NRF_RX                                                        */
/*----------------------------------------------------------------------------------*/
void Init_NRF_RX();

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : NRF_RxData                                                         */
/*----------------------------------------------------------------------------------*/
void NRF_RxData(unsigned char *);

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : NRF_R_Reg                                                          */
/*----------------------------------------------------------------------------------*/
void NRF_R_Reg(unsigned char,unsigned char *, unsigned char);

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : NRF_TxData                                                         */
/*----------------------------------------------------------------------------------*/
void NRF_TxData(unsigned char *);

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : NRF_W_Reg                                                          */
/*----------------------------------------------------------------------------------*/
void NRF_W_Reg(unsigned char, unsigned char *, unsigned char);
```

```c
/*----------------------------------------------------------------------------------*/
/* FILENAME    : nrf.c                                                              */
/* FUNCTION    : Implements different functions to configure and control            */

/*               the nRF24L01                                                       */
/*----------------------------------------------------------------------------------*/
/* DATE        : 18.05.2009                                                         */
/* AUTHOR      : Dries Van de Winkel                                                */
/*----------------------------------------------------------------------------------*/
/* USED HARDWARE: Nordic nRF24L01                                                   */
/*----------------------------------------------------------------------------------*/
/* PROCESSOR   : PIC18LF6585                                                        */
/*----------------------------------------------------------------------------------*/
/* USES   :      define_communication.h                                             */
/*               declare_functions_NRF.h                                            */
/*----------------------------------------------------------------------------------*/
/* USED BY   :   code_cube.c                                                        */
/*               code_cage.c                                                        */
/*----------------------------------------------------------------------------------*/
#include "define_communication.h"
#include "declare_functions_NRF.h"

/*----------------------------------------------------------------------------------*/
/* FUNCTION      : NRF_FlushTX                                                      */
/* COMMENTS      : Flush TX FIFO, used in TX Mode                                   */
/*----------------------------------------------------------------------------------*/
void NRF_FlushTX()
{
    SSPIF = 0;                              // clear SSPIF interrupt flag
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = FLUSH_TX;                      // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
    SSPIF = 0;                              // clear SSPIF interrupt flag
    NRF_CSN = 1;                            // indicates end of SPI operation
}

/*----------------------------------------------------------------------------------*/
/* FUNCTION      : NRF_FlushRX                                                      */
/* COMMENTS      : Flush RX FIFO, used in RX Mode                                   */
/*----------------------------------------------------------------------------------*/
void NRF_FlushRX()
{
    SSPIF = 0;                              // clear SSPIF interrupt flag
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = FLUSH_RX;                      // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
    SSPIF = 0;                              // clear SSPIF interrupt flag
    NRF_CSN = 1;                            // indicates end of SPI operation
}

/*----------------------------------------------------------------------------------*/
/* FUNCTION      : Init_NRF_RX                                                      */
/* COMMENTS      : Initialise the nRF with the following settings (CONFIG register): */
/*                   MASK_RX_DR = 0                                                 */
/*                   MASK_TX_DS = 0                                                 */
/*                   MASK_MAX_RT = 1                                                */
/*                   EN_CRC = 1                                                     */
/*                   CRCO = 0 (1 byte crc)                                          */
/*                   PWR_UP = 1  (in Standby)                                       */
/*                   PRIM_RX = 1 (PRX)                                              */
/*----------------------------------------------------------------------------------*/
void Init_NRF_RX()
{
    NRF_FlushRX();                          // clear RX FIFOs

    // CONFIG
```

```c
    SSPIF = 0;                              // clear SSPIF interrupt flag
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + CONFIG;           // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x1B;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag

    // EN_AA every Auto Ack disabled (to save time)
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + EN_AA;            // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x00;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag

    // EN_RXADDR data pipe 0 enbabled
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + EN_RXADDR;        // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x01;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag

    // SETUP_AW address width of 5 bytes
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + SETUP_AW;         // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x03;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag

    // SETUP_RETR retransmission disabled  (to save time)
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + SETUP_RETR;       // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x00;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag

    // RF_CH: 2400 + 10 = 2410 MHz
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + RF_CH;            // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x0A;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag

    // RF_SETUP: 1Mbps, 0dBm
    NRF_CSN = 1;                            // every new instruction must be started
    NRF_CSN = 0;                            // by a high to low transition on CSN
    SSPBUF = W_REGISTER + RF_SETUP;         // transmit Command over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                          // clear SSPIF interrupt flag
    SSPBUF = 0x07;                          // transmit Data over SPI
        while(SSPIF == 0){}                 // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // STATUS: Clear RX_DR interrupt
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + STATUS;       // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x4E;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // RX_PW_P0 number of bytes in RX payload
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + RX_PW_P0;     // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = PACKET_LENGTH;             // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    NRF_CSN = 1;                        // indicates end of SPI operation

    NRF_CE = 1;                         // start listening for packets
}

/*----------------------------------------------------------------------------------*/
/* FUNCTION     : Init_NRF_TX                                                       */
/* COMMENTS     : Initialise the nRF with the following settings(CONFIG register):  */
/*                  MASK_RX_DR = 0                                                  */
/*                  MASK_TX_DS = 0                                                  */
/*                  MASK_MAX_RT = 1                                                 */
/*                  EN_CRC = 1                                                      */
/*                  CRCO = 0 (1 byte crc)                                           */
/*                  PWR_UP = 1  (in Standby)                                        */
/*                  PRIM_RX = 0 (PTX)                                               */
/*----------------------------------------------------------------------------------*/
void Init_NRF_TX()
{

    NRF_FlushTX();                      // clear TX FIFOs

    // CONFIG
    SSPIF = 0;                          // clear SSPIF interrupt flag
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + CONFIG;       // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x1A;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // EN_AA every Auto Ack disabled (to save time)
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + EN_AA;        // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x00;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // EN_RXADDR data pipe 0 enbabled
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + EN_RXADDR;    // transmit Command over SPI
```

```c
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x01;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // SETUP_AW address width of 5 bytes
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + SETUP_AW;     // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x03;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // SETUP_RETR retransmission disabled  (to save time)
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + SETUP_RETR;   // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x00;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // RF_CH: 2400 + 10 = 2410 MHz
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + RF_CH;        // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x0A;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // RF_SETUP: 1Mbps, 0dBm
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + RF_SETUP;     // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x07;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // STATUS: Clear TX_DS interrupt
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + STATUS;       // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = 0x2E;                      // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    // RX_PW_P0 number of bytes in RX payload
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + RX_PW_P0;     // transmit Command over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
    SSPBUF = PACKET_LENGTH;             // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag

    NRF_CSN = 1;                        // indicates end of SPI operation
}
```

```c
/*----------------------------------------------------------------------------*/
/* FUNCTION     : NRF_RxData                                                  */
/* COMMENTS     : This function clocks out the received data from the RX FIFO.      */
/*                Syntax:                                                     */
/*                NRF_RxData(unsigned char *r_data) ->                        */
/*                      r_data = Array to save received data.                 */
/*----------------------------------------------------------------------------*/
void NRF_RxData(unsigned char *r_data)
{
    unsigned char number_of_bytes = 0;  // number_of_bytes
    number_of_bytes = PACKET_LENGTH;

    SSPIF = 0;                          // clear SSPIF interrupt flag
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = R_RX_PAYLOAD;              // transmit Command over SPI
    while(SSPIF == 0){}                 // wait until SSPBUF is empty
    SSPIF = 0;                          // clear SSPIF interrupt flag

    while(number_of_bytes != 0)         // loop until number_of_bytes is zero
    {
        SSPBUF = DUMMY_BYTE;            // transmit dummy byte to receive data
        while(BF == 0){}                // wait until SSPBUF is full
        *r_data++ = SSPBUF;             // save data in r_data array
        number_of_bytes--;              // decrement number_of_bytes
    }
    NRF_CSN = 1;                        // indicates end of SPI operation
}

/*----------------------------------------------------------------------------*/
/* FUNCTION     : NRF_R_Reg                                                   */
/* COMMENTS     : This function reads the chosen register and saves it in r_data.   */
/*                Syntax:                                                     */
/*                NRF_R_Reg(Register, Array to save read data, size of register)    */
/*----------------------------------------------------------------------------*/
void NRF_R_Reg(unsigned char reg, unsigned char *r_data, unsigned char size)
{
    SSPIF = 0;                          // clear SSPIF interrupt flag
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = R_REGISTER + reg;          // transmit Command over SPI
    while(SSPIF == 0){}                 // wait until SSPBUF is empty
    SSPIF = 0;                          // clear SSPIF interrupt flag

    while(size != 0)                    // loop until size is zero
    {
        SSPBUF = DUMMY_BYTE;            // transmit dummy byte to receive data
        while(BF == 0){}                // wait until SSPBUF is full
        *r_data++ = SSPBUF;             // save data in r_data array
        size--;                         // decrement size
    }
    NRF_CSN = 1;                        // indicates end of SPI operation
}

/*----------------------------------------------------------------------------*/
/* FUNCTION     : NRF_TxData                                                  */
/* COMMENTS     : This function writes data into the TX data payload (TX FIFO).     */
/*                Syntax:                                                     */
/*                NRF_TxData(unsigned char *t_data) ->                        */
/*                  t_data = Array with data to send                          */
/*----------------------------------------------------------------------------*/
void NRF_TxData(unsigned char *t_data)
{
    unsigned char number_of_bytes = 0;  // number_of_bytes
    number_of_bytes = PACKET_LENGTH;
```

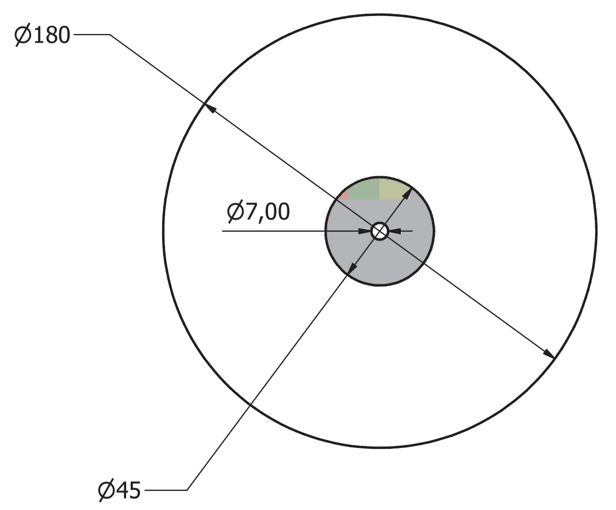
```c
    SSPIF = 0;                          // clear SSPIF interrupt flag
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_TX_PAYLOAD;              // transmit Command over SPI
    while(SSPIF == 0){}                 // wait until SSPBUF is empty
    SSPIF = 0;                          // clear SSPIF interrupt flag

    while(number_of_bytes != 0)         // loop until number_of_bytes is zero
    {
        SSPBUF = *t_data++;             // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
        number_of_bytes--;              // decrement number_of_bytes
    }
    NRF_CSN = 1;                        // indicates end of SPI operation
}

/*----------------------------------------------------------------------------*/
/* FUNCTION     : NRF_W_Reg                                                   */
/* COMMENTS     : This function writes a value into the chosen register from the nRF*/
/*                Syntax:                                                     */
/*                NRF_W_Reg(Register, Array with data to write, size of register)   */
/*----------------------------------------------------------------------------*/
void NRF_W_Reg(unsigned char reg, unsigned char *t_data, unsigned char size)
{
    SSPIF = 0;                          // clear SSPIF interrupt flag
    NRF_CSN = 1;                        // every new instruction must be started
    NRF_CSN = 0;                        // by a high to low transition on CSN
    SSPBUF = W_REGISTER + reg;          // transmit Command over SPI
    while(SSPIF == 0){}                 // wait until SSPBUF is empty
    SSPIF = 0;                          // clear SSPIF interrupt flag

    while(size != 0)                    // loop until size is zero
    {
        SSPBUF = *t_data++;             // transmit Data over SPI
        while(SSPIF == 0){}             // wait until SSPBUF is empty
        SSPIF = 0;                      // clear SSPIF interrupt flag
        size--;                         // decrement size
    }
    NRF_CSN = 1;                        // indicates end of SPI operation
}
```

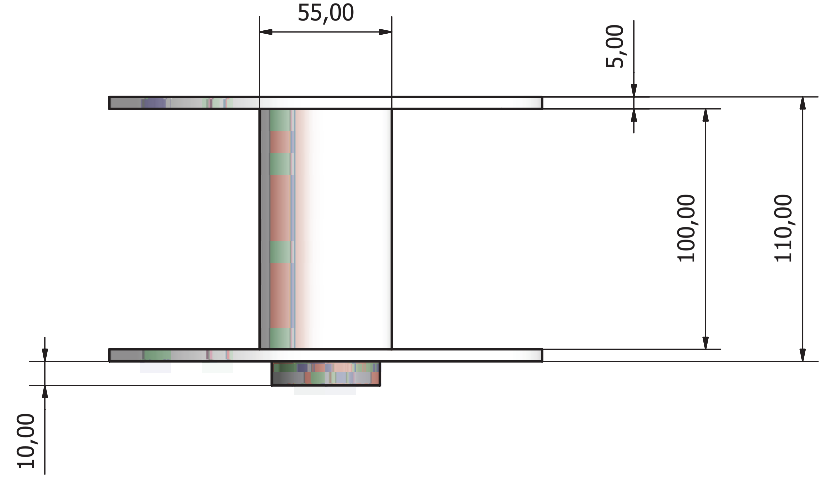
```c
/*-------------------------------------------------------------------------------*/
/* FILENAME   : define_communications.h                                          */
/* FUNCTION   : Header file, defines ports and registers involving operation of  */
/*              the nRF24L01                                                      */
/*-------------------------------------------------------------------------------*/
/* DATE       : 18.05.2009                                                        */
/* AUTHOR     : Dries Van de Winkel                                               */
/*-------------------------------------------------------------------------------*/
/* USED HARDWARE: Nordic nRF24L01                                                 */
/*-------------------------------------------------------------------------------*/
/* PROCESSOR  : PIC18LF6585                                                       */
/*-------------------------------------------------------------------------------*/
/* USES  : /                                                                      */
/*-------------------------------------------------------------------------------*/
/* USED BY  :   nrf.c                                                             */
/*              code_cube.c                                                       */
/*              code_cage.c                                                       */
/*-------------------------------------------------------------------------------*/

/*-------------------------------------------------------------------------------*/
/* Number of bytes in one payload                                                 */
/*-------------------------------------------------------------------------------*/
#define     PACKET_LENGTH       5

/*-------------------------------------------------------------------------------*/
/* Define general MCU ports for communication                                     */
/*-------------------------------------------------------------------------------*/
#define     NRF_CSN             RC7
#define     NRF_CE              RE2
#define     NRF_SCK             RC3
#define     NRF_SDI_UP          RC4
#define     NRF_SDO             RC5
#define     NRF_IRQ_UP          RB3

#define     NRF_CSN_DIR         TRISC7
#define     NRF_CE_DIR          TRISE2
#define     NRF_SCK_DIR         TRISC3
#define     NRF_SDI_UP_DIR      TRISC4
#define     NRF_SDO_DIR         TRISC5
#define     NRF_IRQ_UP_DIR      TRISB3

/*-------------------------------------------------------------------------------*/
/* Define names for memory map addresses                                          */
/*-------------------------------------------------------------------------------*/
#define     CONFIG          0x00
#define     EN_AA           0x01
#define     EN_RXADDR       0x02
#define     SETUP_AW        0x03
#define     SETUP_RETR      0x04
#define     RF_CH           0x05
#define     RF_SETUP        0x06
#define     STATUS          0x07
#define     OBSERVE_TX      0x08
#define     CD              0x09
#define     RX_ADDR_P0      0x0A
#define     RX_ADDR_P1      0x0B
#define     RX_ADDR_P2      0x0C
#define     RX_ADDR_P3      0x0D
#define     RX_ADDR_P4      0x0E
#define     RX_ADDR_P5      0x0F
#define     TX_ADDR         0x10
#define     RX_PW_P0        0x11
#define     RX_PW_P1        0x12
#define     RX_PW_P2        0x13
#define     RX_PW_P3        0x14
#define     RX_PW_P4        0x15
#define     RX_PW_P5        0x16
#define     FIFO_STATUS     0x17
```

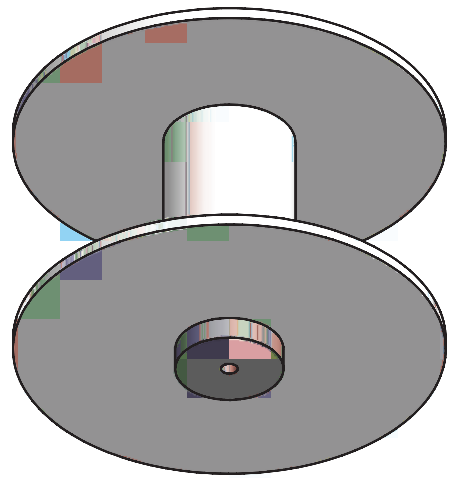
```c
/*-------------------------------------------------------------------------------*/
/* Define name for SPI dummy byte (receive mode)                                  */
                */
/*-------------------------------------------------------------------------------*/
#define     DUMMY_BYTE      0xAA

/*-------------------------------------------------------------------------------*/
/* Define names for SPI Commands                                                  */
/*-------------------------------------------------------------------------------*/
#define     R_REGISTER      0x00
#define     W_REGISTER      0x20
#define     R_RX_PAYLOAD    0x61
#define     W_TX_PAYLOAD    0xA0
#define     FLUSH_TX        0xE1
#define     FLUSH_RX        0xE2
#define     NO_OPERATION    0xFF

/*-------------------------------------------------------------------------------*/
/* Define names for Protocol                                                      */
/*-------------------------------------------------------------------------------*/

/* First byte of protocol contains address for cube and cage (base station)      */
#define     CAGE_ADDRESS        0xB5    //4 bits address
#define     CUBE_ADDRESS        0xB0    //4 bits address
#define     SRC_ADDRESS_INDEX   0
#define     DEST_ADDRESS_INDEX  1

/* Second (and third) byte of protocol contain(s) data for levitation operation.  */
#define     CAGE_LEVIT_H_INDEX  2
#define     CAGE_LEVIT_L_INDEX  3
#define     CUBE_LEVIT_INDEX    2

/* Depending on the configuration of cube and cage (as sender or receiver):      */
/* Third byte of protocol contains 1) data for motor speed and direction          */
/* (cube as receiver) or 2) data for power induction (cage as receiver)           */
#define     CAGE_PI_INDEX       4
#define     CUBE_ROT_INDEX      4

#define     NO_CMD              0x0F

#define     UP_CMD              0x11
#define     DOWN_CMD            0x22
#define     STOP_CMD            0x33
#define     LEFT_CMD            0x44
#define     RIGHT_CMD           0x55

#define     PI_ON               0x66
#define     PI_OFF              0x77
```

Appendix F

Mechanical Parts



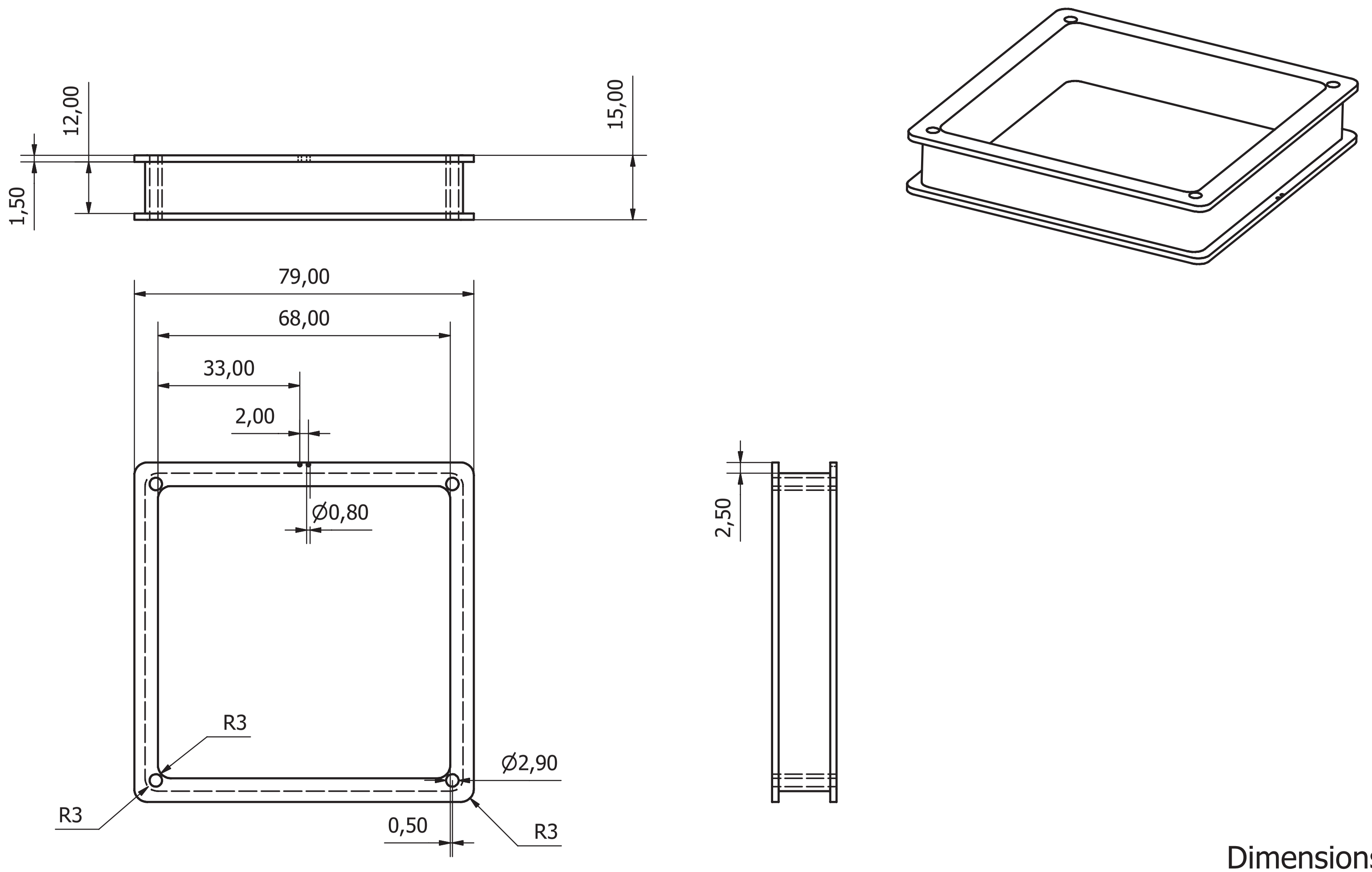




Dimensions in mm

Material: PVC and Iron

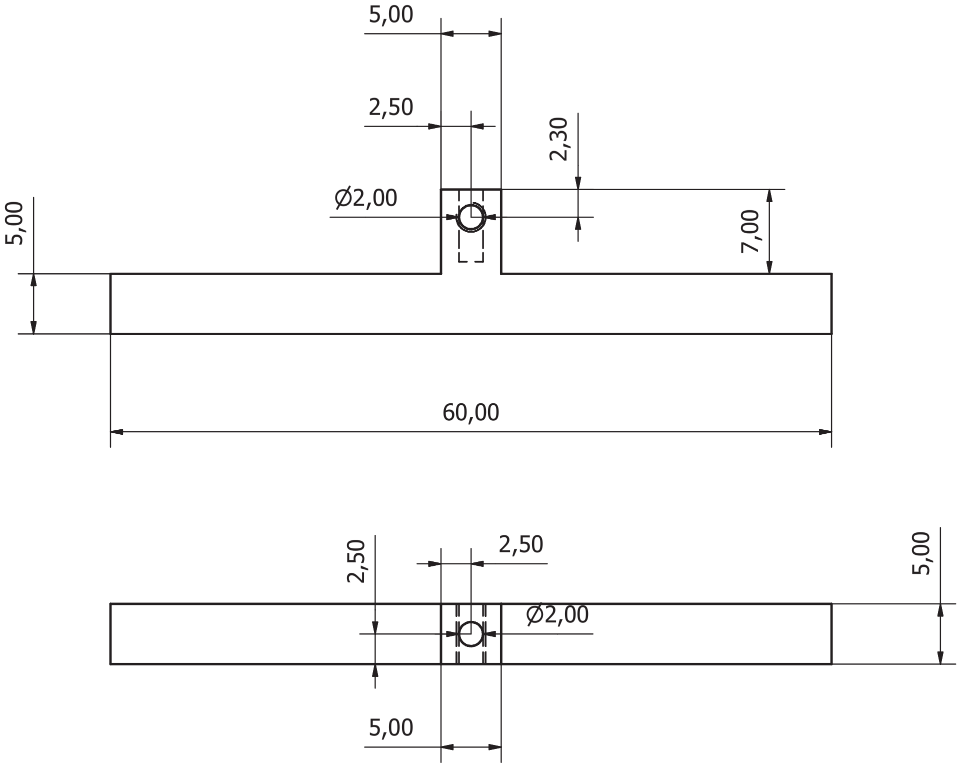
Designed by	Checked by	Approved by	Date		Date	
vanddrie					18.05.2009	
Inertial Platform for CubeSat			Frame for Levitation Coil			
			HES-SO Valais		Edition	Sheet 1 / 1

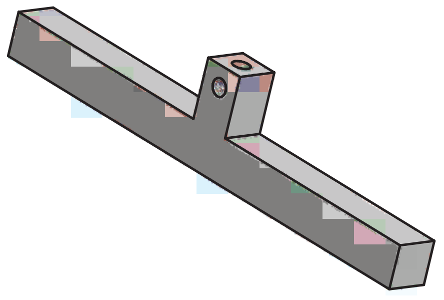


Dimensions in mm

Material: PVC

Designed by	Checked by	Approved by	Date	Date
craemaar				15.05.2009
Inertial Platform for CubeSat	Frame for Power Induction receiving coil			
	HES-SO Valais	Edition	Sheet 1 / 1	



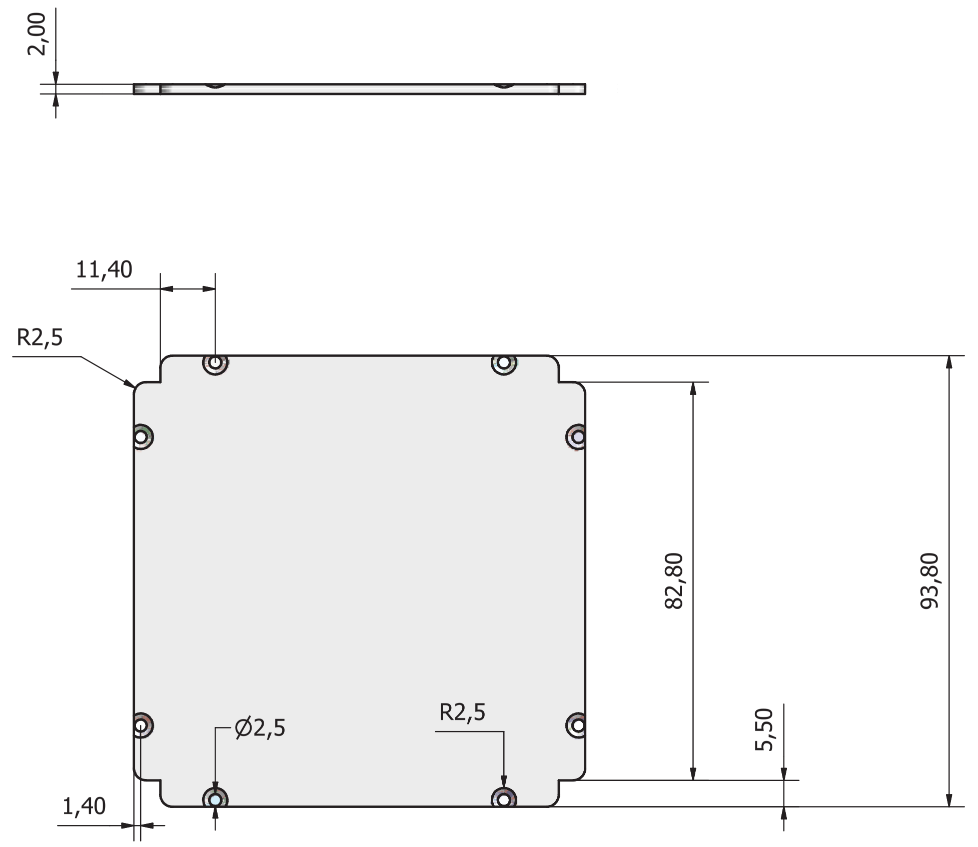


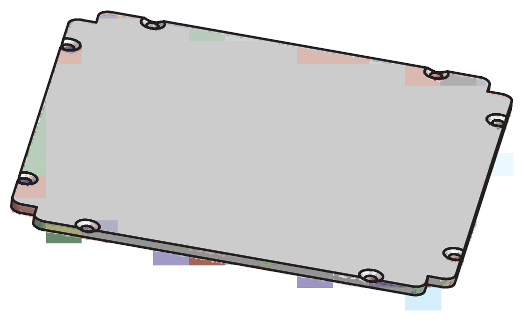
Material: Alu

Dimensions in mm

Designed by	Checked by	Approved by	Date	Date
craemaar				03.06.2009

	Edition	Sheet
Wheel		1 / 1





Dimensions in mm

Material: Alu

Designed by	Checked by	Approved by	Date	Date	
craemaar				18.05.2009	
Inertial Platform for CubeSat			Cube top and bottom lid		
			HES-SO Valais	Edition	Sheet 1 / 1

Bibliography

[1] Linear Technology Corporation. *LTC3539/LTC3539-2, 2A, 1MHz/2MHz Synchronous Step-Up DC/DC Converters*. `http://www.linear.com`.

[2] Linear Technology Corporation. *LTC4063, Standalone Linear Li-Ion Charger with Micropower Low Dropout Linear Regulator*. `http://www.linear.com`.

[3] iC Haus. *iC-WK, iC-WKL, 2.4*V *CW Laser Diode Driver*. `http://www.ichaus.com`.

[4] Microchip. *PIC18F6585/8585/6680/8680, 64/68/80-Pin High-Performance, 64-Kbyte Enhanced Flash Microcontrollers with ECAN Module*. `http://www.micrchip.com`.

[5] Nordic Semiconductor. *nRF24L01, Single Chip 2.4GHz Transceiver*. `http://www.nordicsemi.no`.

[6] STMicroelectronics. *LISY300AL, MEMS inertial sensor: single-axis* ±300°/s *analog output yaw rate gyroscope*. `http://www.st.com`.

[7] Agilent Technologies. *AEDR-8000 Series Encoders Reflective Surface Mount Optical Encoder*. `http://www.agilent.com/semiconductors`.