

Projets Industriels – Elec4

Année scolaire 2019-2020

Project Documentation

Intelligent solar generator for Cubesat

Version 1.0

**Students : Brunin Camille, Combail Quentin,
Cocogne Romain**

Tutors: Florentin Millour

Company: Lab Lagrange

THANKS

First of all, we would like to thank our project tutor, M. Florentin MILLOUR, for his time, patience and cheerfulness.

Thanks to M. Olivier PREIS too for having brought us to the observatory, it was a really rewarding experience.

Finally, thanks to M. JACOB and M. LANGELLA for their help and educational support.

TABLE OF CONTENT

I. Introduction	5
II. Architecture	6
II.1. Block Diagram	6
II.2. The solar panel	6
II.3. The main sensors.....	6
II.4. The microcontroller.....	7
II.5. The power supply	7
III. Plan and Tasks	8
III.1. Objectives	8
III.2. Reality	8
IV. BOM (Bill of material).....	10
V. Estimated budget	11
VI. Components specifications.....	13
VI.1. MICROCONTROLLER.....	13
VI.2. SENSORS.....	13
VI.2.1. Solar sensor	13
VI.2.2. Magnetometer.....	13
VI.2.3. Horizon sensor.....	14
VI.2.4. Temperature sensor	15
VI.2.5. Voltmeter.....	15
VI.2.6. Amperemeter	15
VI.3. SOLAR CELLS.....	16
VII. Development.....	17
VII.1. Solar cells	17
VII.1.1. Power characteristics.....	17
VII.1.2. Voltage Reference	17
VII.1.3. Voltage regulator.....	17
VII.1.4. Amperemeter.....	18
VII.2. Microcontroller and sensors interfacing.....	18
VII.2.1. Code development.....	19
VII.3. Sensor calibration.....	20
VII.3.1. LM75A – Temperature sensor	20
VII.3.2. LSM9DS1 – Magnetometer	20
VII.3.3. AMG88xx Thermal Camera	21

VII.4. Communication with central unit.....	22
VII.4.1. Hardware setup.....	22
VII.4.2. Software setup.....	23
VII.5. Prototypes.....	24
VII.5.1. Prototype 1	24
VII.5.2. Prototype 2	26
VIII. Challenges	28
VIII.1. Technological challenges.....	28
VIII.2. Adapting to the lockdown.....	28
IX. Current state of the project	29
IX.1. Parts that are functional	29
IX.1.1. Solar cells electronic schematic.....	29
IX.1.2. Thermometer, magnetic sensor	29
IX.1.3. 8x8 thermal camera	29
IX.1.4. Communication with the central unit	29
IX.1.5. PCB.....	30
IX.2. Parts that require further work.....	30
IX.2.1. Using the ultra-low power features of the microcontroller	30
IX.2.2. 32x24 Thermal camera	30
IX.2.3. Solar sensors	30
IX.2.4. Network of sensors	30
X. References	31
XI. Annexes	32
XI.1. Code.....	32
XI.1.1. Thermal Camera	32
XI.1.2. Magnetic sensor calibration.....	36
XI.1.3. USCI_I2C library	37
XI.2. Schematics	39
XI.2.1. Prototype 2.....	39

I.Introduction

NiceCube is a nanosatellite project that will allow the *Lab Lagrange* to do technological demonstrations by performing optical communications between earth and the satellite. **NiceCube** is a *CubeSat* type of satellite. It is relatively small, lightweight, and rectangle shaped. The challenges are to make **NiceCube** self-sufficient in terms of energy, and to gather data from its environment.

A satellite can be powered by different types of generators. The most common and easy-to-use type of generators are solar panels, because they take advantage of the satellite's long exposure to sunlight in space. They can also be easily integrated in an electronic system. This is what we decide to use to power **NiceCube**.

Our mission is to conceive a smart solar panel that powers the whole satellite. It will be used as a solar generator and will be able to give us useful real-time information about the satellite. Indeed, we need to measure the current and voltage output of the solar panel, as well as its temperature, the relative location of the sun, and the earth horizon orientation. These measurements are needed to manage the solar cells and to determine the satellite's orientation. Moreover, the solar cells will need to be electrically protected and individually deactivated if damaged.

In this document we will first go through the requirements of the project, then we will explain how we organized and compare what we planned to do and what we did. Finally, we will go over all the components and prototype's specifications.

II. Architecture

II.1. Block Diagram

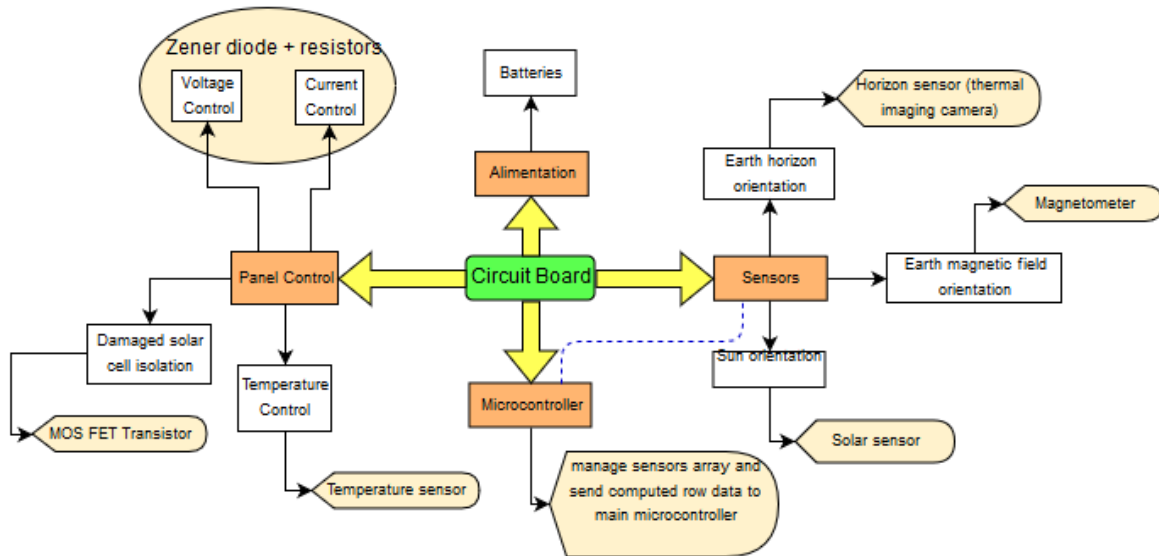


Figure II.1 Architecture block diagram

The circuit board will be organized into four different parts described below.

II.2. The solar panel

The first and core part of our project is the solar panel. The panel control is essential to make sure that the satellite is powered correctly. The panel is a “smart” panel, meaning it can manage itself.

The panel is made of several banks wired in parallel, with each bank made of multiple solar cells wired in series. The voltage output is determined by the number individual cells in each bank, and the current output is set by the number of banks on the panel.

Current and voltage measurements are performed to check if the cells are furnishing the required power for the whole satellite, or to detect a failure in the cell banks. We monitor the temperature of the cells as well with temperature sensors.

If one or several cell banks are damaged, they can be electrically disconnected from the circuit with MOSFET transistors. The command of these transistors is handled by the microcontroller.

II.3. The main sensors

This is the part of the satellite that is used to gather data from the satellite’s environment. It includes three different types of sensors that are used to determine the current orientation of the satellite.

This part is composed of the following sensors:

- A solar sensor made of two 1-dimensional *PSDs* (Positional Sensitive Detectors) used to determine the position of the sun relative to the satellite
- A magnetic field sensor used to detect earth’s magnetic field and its relative direction
- A thermal imaging camera used to deduce earth’s horizon orientation from infrared radiation.

These measurements are computed to deduce three vectors aligned with:

- The orientation of the sun
- Earth's horizon
- Earth's magnetic field

These vectors are then used by the satellite's central unit to regulate its orientation.

II.4. The microcontroller

The microcontroller's duty is to manage the solar panel and the sensors. Its main tasks are:

- Managing the solar panel: checking the status of the solar banks through current and voltage measurements. Opening or closing the MOSFET transistors when a cell is damaged.
- Receiving commands from the central unit and sending back the required data
- Managing communication between the sensors

The sensors are accessed using an I²C bus in which the microcontroller acts as the master unit. Likewise, the communication with the central unit uses I²C in which the microcontroller is the slave.

The central unit sends a message containing an 8-bit command and an eventual 8-bit parameter. Upon receiving the command, the microcontroller executes it and holds the response in the buffer. The central unit then asks to receive the processed data, and the microcontroller starts transmitting. Data is transmitted in bytes, *LSB* first.

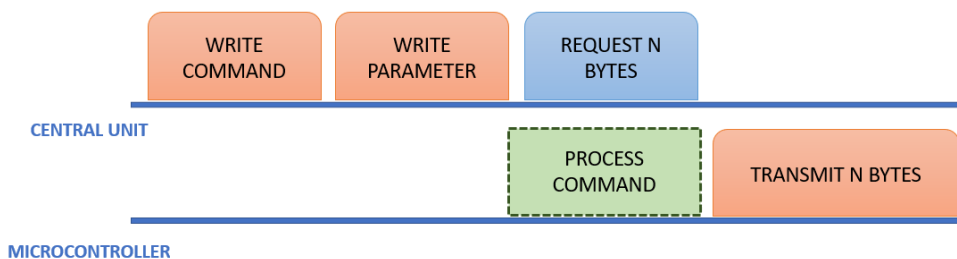


Figure II.2 Communication protocol between the microcontroller and the central unit

Below is a list of possible commands:

- **READ_SENSOR**: asks for a sensor's value. The sensor is specified with a parameter corresponding to a sensor's id. There is also a specific parameter to ask for a measurement from every sensor.
- **LIST_SENSOR**: asks for a list of available and functioning sensors
- **CONROLLER_STATUS**: asks the controller to perform a self-status check and to report any malfunction, such as defective I/Os or unusual temperature.
- **CELLS_STATUS**: asks the controller to perform a status check on the solar cells.

II.5. The power supply

This part would be the batteries, charged by our solar panel. However, we are not in charge of the batteries part. We just need to supply enough power to be able to charge them.

Our system needs to be self-sufficient, which mean we must design our system so it can be powered entirely by the solar generator.

III. Plan and Tasks

III.1. Objectives

The first semester was dedicated to defining the specifications of the project, describing a technical solution, and organizing the development. Tasks were performed by Camille Brunin and Romain Cocogne.

We planned our work as described in this Gantt chart.

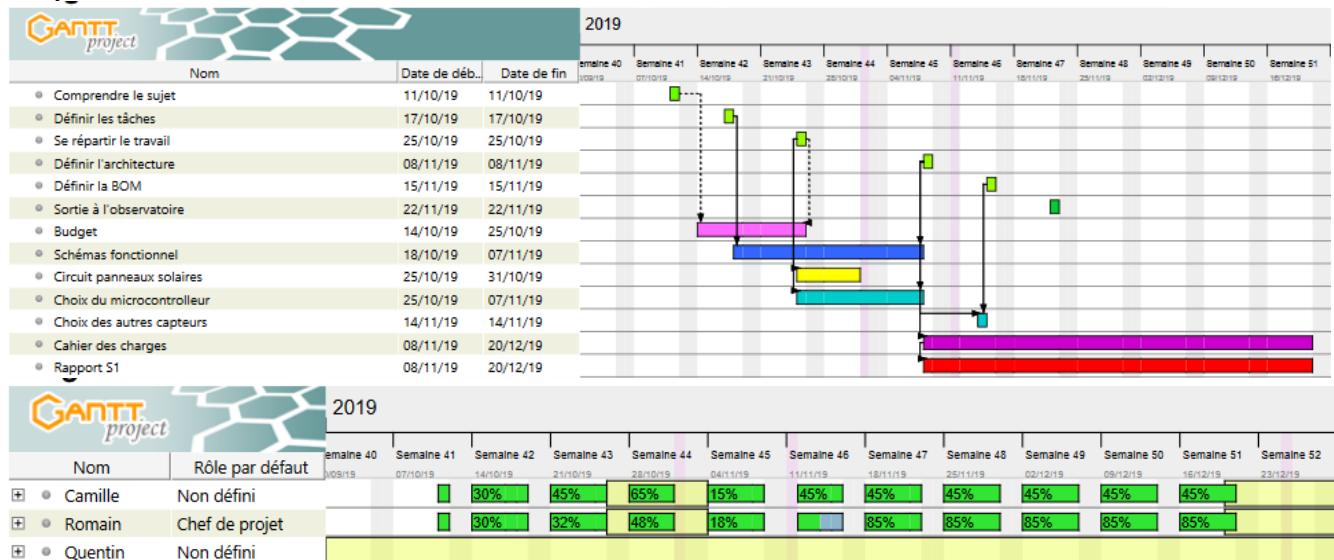


Figure III.1 Semester 1 planned Gantt chart

During the second semester, we developed and adapted the solutions established during the first semester. Tasks were performed by Romain Cocogne and Quentin Combai.

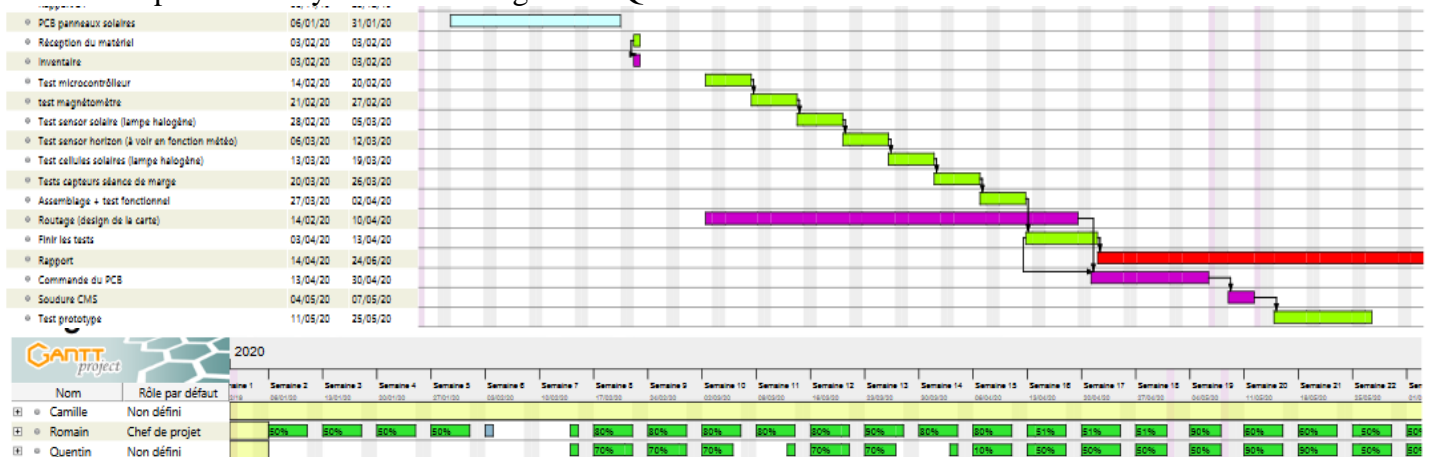


Figure III.2 Semester 2 planned Gantt chart

III.2. Reality

For the first semester, we managed to complete all planned tasks on schedule. We did re-shuffle the tasks because we were able to work on tasks in parallel, but overall the planning was accurate. You can see on the resources section that some of them were overloaded, but in reality, the workload was equally supported by both resources. This kind of temporary help can't be transcribed into the Gantt software.

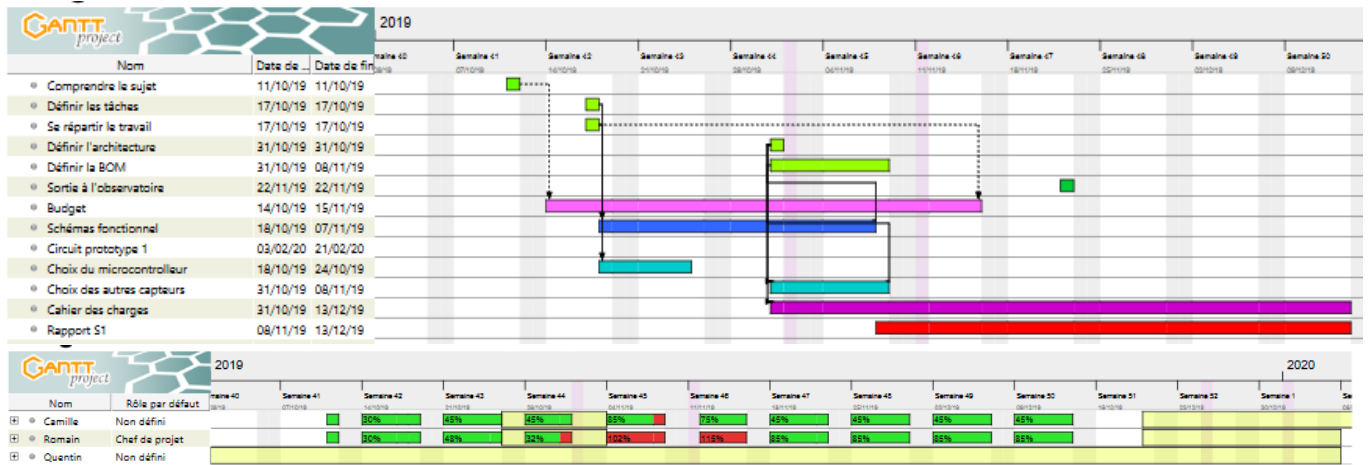


Figure III.3 Semester 1 real Gantt chart

The second semester did not plan out as scheduled. Due to global lockdown caused by COVID-19 pandemic, we re-organized ourselves and changed our objectives, as described in the [Adapting to the lockdown](#) section. Overall, Quentin was responsible for most of the software development and Romain was responsible for the hardware development.

We also had one more prototype to do compared to what was planned in semester 1, so we had to squeeze some time for his development. We did have to remove some objectives such as working on the PSD to still be on schedule.

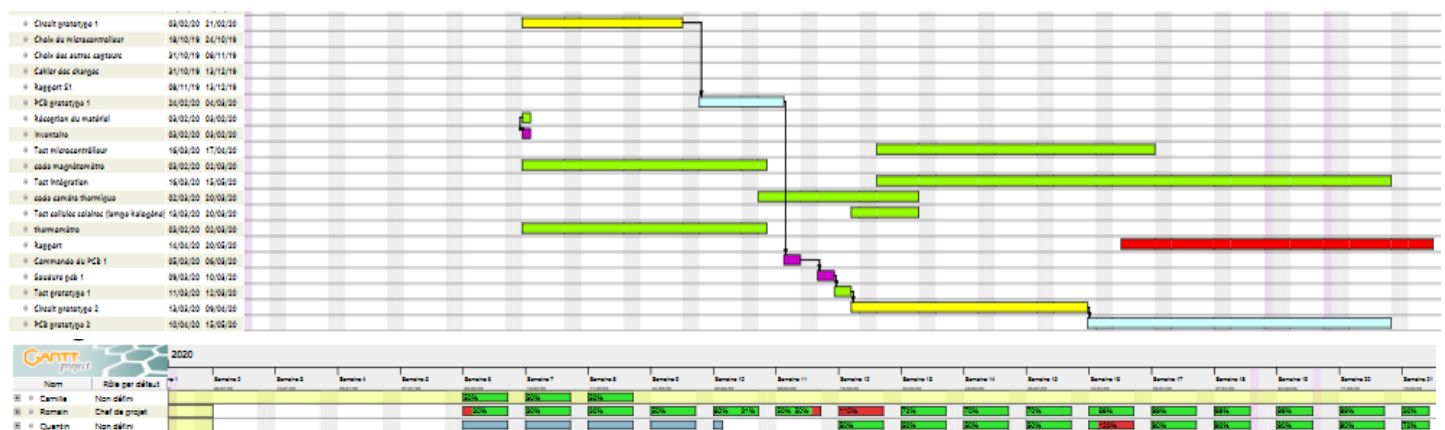


Figure III.4 Semester 2 real Gantt chart

V.Estimated budget

During the first semester, we were able to estimate the budget by taking into account all of the factors as if we really started this project from scratch. First, we estimated our salaries as engineers and our tutor's salary. It gave us a total of 6 007€.

Salaire équipe ingénieur débutant						
Salaire moyen ingénieur (brut) / mois	3,200€					
Charges patronales	42%					nb de journées
Total/mois	4,544€					
Total/heure	34€		Nombre de séances	Semestre 1	8	4
Total sur année (pour projet) par personne	2,045€			Semestre 2	6	3
Total sur année (pour projet)	4,090€		total des jours de travail			7
Salaire encadrant chercheur						
Salaire brut /mois	3,000€					
Charges patronales	42%					
Total /mois	4,260€					
Total/heure	32€					
Total sur année (pour projet)	1,917€					
Total des totaux	6,007€					

Figure V.1 *Estimated salaries*

Then, we estimated the travel costs for us and our project tutor, which led us to a total of 1 299.96€.

		Barème fiscal		
		Etudiants		Encadrant
Chevaux fiscaux 2019	Prix /km	Camille	Romain	
5	0.543	Aller/Retour	18.6304	57.12€
6	0.568	Total (12 séances)	223.5648	685.44€
7	0.595	Total des totaux	1,299.96€	

Figure V.2 *Estimated travel costs*

After that, we had to calculate the costs of the material that we are going to buy and rent, for a total of $2207.56 + 117 = 2324.56€$.

VI. Components specifications

VI.1.MICROCONTROLLER

The microcontroller needs to manage the sensor network, control the well-being of the solar cells, and communicate with the satellite's central unit. It needs to embed at least two hardware I²C modules, so that it can manage one I²C bus for sensor communication, and another bus to exchange data with the central unit. It must have at least 6 analog pins for current and voltage measurements. Most importantly, the microcontroller needs to be optimized for ultra-low power consumption.

Following the advice of our tutor Florentin Millour, we looked towards the [MSP430 microcontrollers](#) from Texas Instruments. This family of microcontrollers is known for its low power capabilities and is widely used in similar projects where reducing consumption is essential.

The [MSP430FR59XX](#) microcomputer family has the following characteristics:

- Up to 256KB of Ferroelectric Random Access Memory (*FRAM*) with ultra-low power writes
- Optimized Ultra-Low-Power Modes, 350 nA consumption when on standby with a Real Time Clock
- Multifunction I/O ports, 40 to 68 depending on the model
- Up to 4 hardware I²C modules

The [MSP430FR59XX](#) satisfies all of the aforementioned requirements.

VI.2.SENSORS

VI.2.1.Solar sensor

We need to sense the satellite's orientation compared to the sun. Knowing the required angle precision between the satellite and the antenna is 10°, the sensor needs a 3° precision in the worst case. The technology used is a masked one-dimensional PSD (depth of the mask is 1.5mm), so the corresponding length precision of the diode is 78µm maximum.

The S3274-05 model gets a 35µm maximum precision, which is under the specification. This PSD needs an under 300µm wide light ray. The R1DS3N mask has a 10µm wide strip line.

VI.2.2.Magnetometer

We need to measure the satellite's inclination compared to the earth's surface. We will use a magnetometer to sense the earth's magnetic field.

As the solar sensor, we need a 3° precision in the worst case.

Knowing the earth's magnetic field of 0.5Gauss, we found a maximum error of 26mGauss.

The LSM9DS1's precision is 0.14mGauss for 4Gauss magnetic fields and under. This corresponds to a 0.02° error. We are well under the specifications.

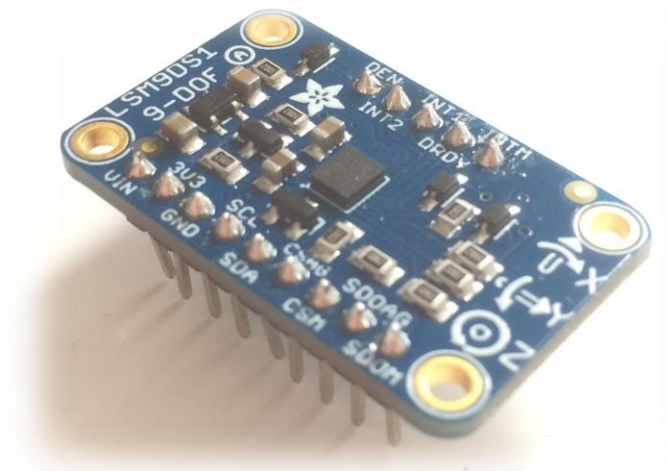


Figure VI.1 *LSM9DS1 magnetic sensor on a prototyping board from Adafruit.*

VI.2.3. Horizon sensor

We need another vector to angle the satellite compared to the earth, so we need to use a horizon sensor. Just like the other sensor, this one needs a 3° precision in the worst case.

The MLX90641 is a 16×12 pixel thermal camera with a field of view of 110° , giving us a $9^\circ/\text{pixel}$ precision. It means we have a 2.6° precision in the camera. Although we can use this camera, further tests need to be done to see if we are not under the 3° error. If that's the case, a more power costly camera could be considered. The MLX90640, for example, has a resolution of 32×24 pixels.

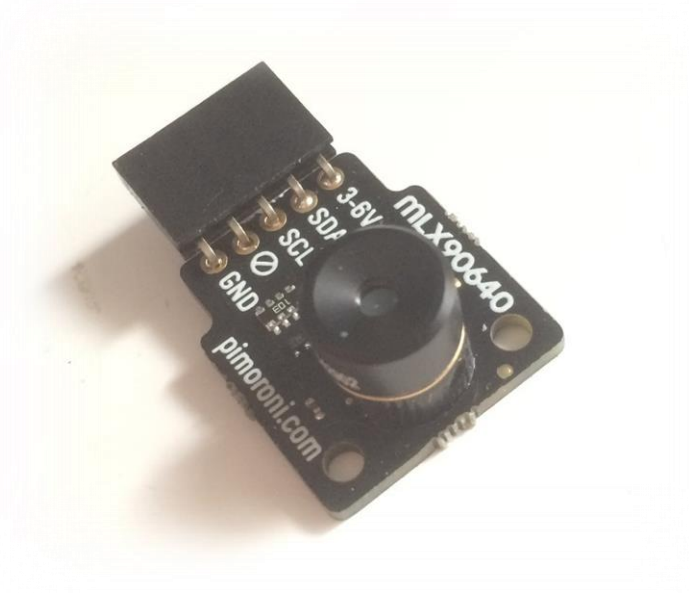


Figure VI.2 *MLX90640 32x24 thermal camera*

VI.2.4. Temperature sensor

This sensor is needed to control the panel's temperature. We can put a sensor in the center of the panel and another sensor opposite to the microcontroller to check if there is no overheating. There is no need for a big precision, as we want to measure big temperature variation. A 1°C precision is enough for us. The LM75A has two resolution modes, 1°C and 0.125°C .

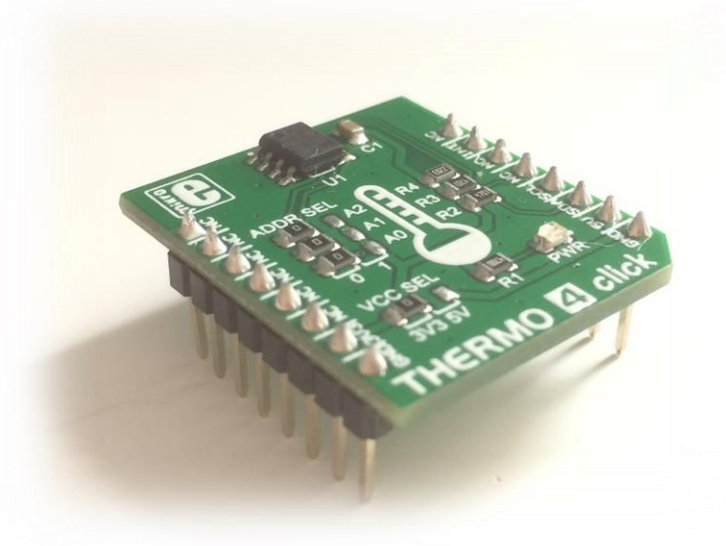


Figure VI.3 LM75A temperature sensor on a prototyping board from Mikroe Electronika

VI.2.5. Voltmeter

We need to measure the volt variation (drops and peaks) to check the state of our solar cells. We will use the analog pin of our micro-controller, with a max voltage input of 3.3V. To get maximum precision over the measurement, we take the voltage reference at 1.15V.

To make a voltage reference, we need to use a Zener Diode. As there is no Diode with a Zener voltage under 1.8V, we take 1.8V as a reference. With this, we can detect a voltage drop under 2V, which is enough to detect one solar cell's failure.

VI.2.6. Amperemeter

To complement the voltage measurement, we need a current variation measurement. We will use resistors in series with the solar cells and measure the resulting voltage. As the solar cells current is around 14mA, the analog pin of our microcontroller will be enough to detect a current drop.

VI.3.SOLAR CELLS

The system needs to be compatible with the CubeSat structure, meaning a 10x10cm² surface with truncated angles.

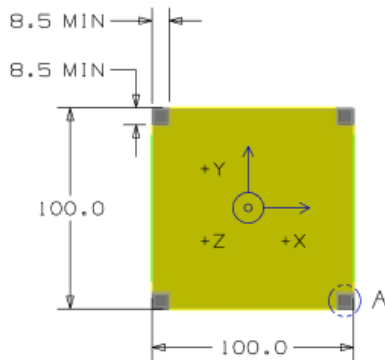


Figure VI.4 Dimensions of the topside of a CubeSat structure, from CubeSat datasheet

The yellow plane above is the available space for our system.

Following the advice of our tutor, we use TrisolX solar cells. TrisolX cells are relatively cheap and are optimized for space applications. They are often used for similar projects.

Each one is approximately 2.6cm². We use a 5mm margin around each cell. Considering the cells will be using 60% of the total panel surface, we can place a total of 12 cells onto our panel.

After testing, the ratio will be adjusted and more panels will be added, but for the prototypes, this ratio will be enough.

The batteries' control system is [GOMSpace, nanopower p31u](#). As stated in the datasheet, we need to deliver between 4.2V and 8.5V. Knowing the cells are 2.33V max each, we will put 4 cells in series to deliver the correct voltage. As we can use 12 cells, we will make a 4x3 cell matrix.

To protect each line, we put a Schottky diode in series with the 4 cells.

Here is a schematic of one solar bank configuration taken from [Prototype 1](#) schematic.

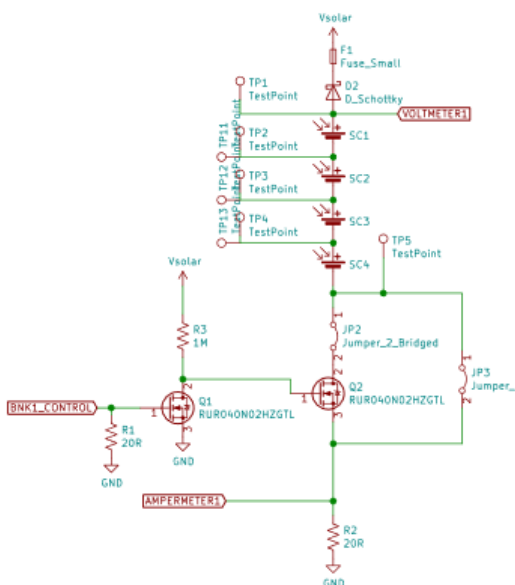


Figure VI.5 Solar bank schematic

VII.Development

VII.1.Solar cells

VII.1.1.Power characteristics

We need to know the best load to put on the solar bank to maximize the power output. To do that, we try different load and measure voltage and current output. In our testing setup, we used a 400W halogen lamp. Halogen lamps provide a luminous spectrum very similar to the sunlight spectrum, and we want to get as close as possible to that spectrum to get realistic tests. The lamp was placed at 10cm from the solar cell testing board. We computed the results into the following curve.

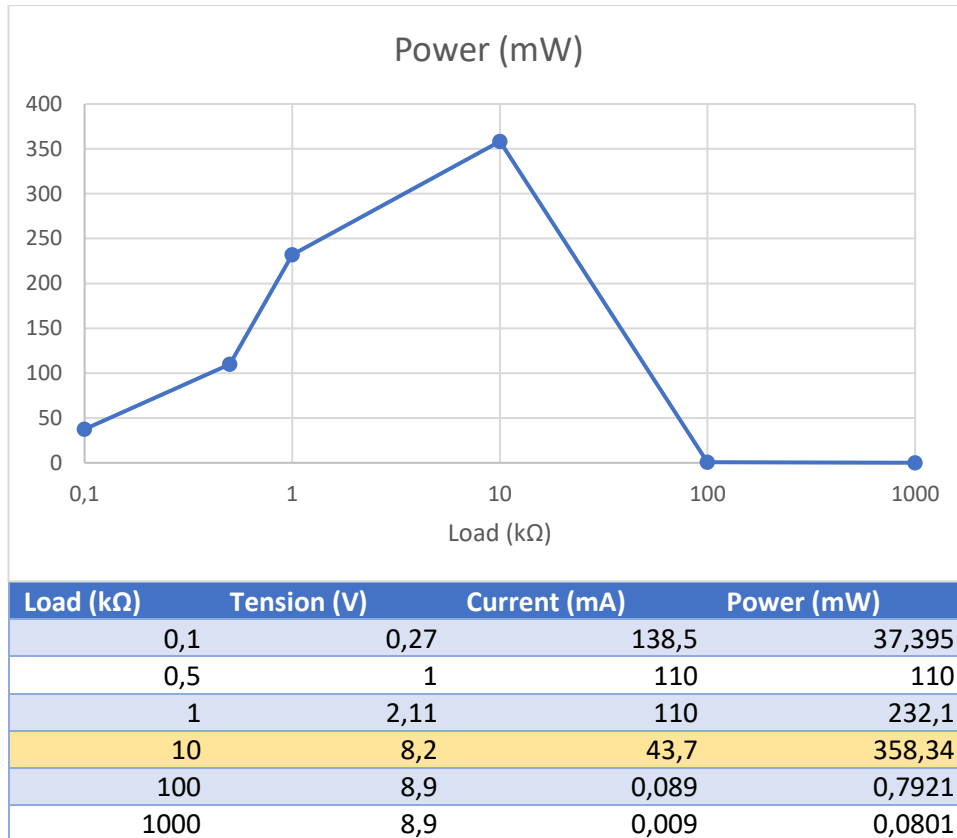


Figure VII.1 Power generation of one solar cell relatively to circuit load.

We can see the optimum load of 10k can provide us with 350mW of power.

VII.1.2.Voltage Reference

At first, we tried to use a Zener diode to provide constant output voltage, but as explained in [Prototype 1](#) category, the result was not satisfactory. We chose to add a dedicated circuit.

We took the **MCP1501**. It provides a constant **1.25V** as reference, with 0.08% of precision. It is possible to shut down the device using the SHDN pin, so we can reduce power consumption.

VII.1.3.Voltage regulator

To power all sensors and the microcontroller, we needed to add a 3.3V voltage regulator. We took the NCP59302DSADJR4G adjustable voltage regulator for his low dropout voltage, its wide output current range and its supply voltage working range (still working with 1.2V input).

VII.1.4.Amperemeter

To measure each solar bank's current input, we first used a shut 20Ω resistor. As explained in [Prototype 2](#), this method doesn't provide a stable enough output, so we added a dedicated circuit instead.

We chose the [INA3221](#) triple-channel current and voltage monitor because of his I2C compatibility, the ability to monitor 3 solar banks at the same time and his programable alert and warning outputs, allowing us to deport current monitoring to the sensor.

The shut resistors are in turn way smaller, consuming less power from the solar panels.



Figure VII.2. *INA3221 TI current and voltage monitor*

VII.2.Microcontroller and sensors interfacing

We were unable to get a micro controller from the MSP430FR59XX family. Instead, we used a MSP430G2553 from Texas Instrument, which provides similar features regarding ultra-low power modes but offers less I/O ports and only one hardware I²C module.

All the sensors we are using communicate their data through I2C busses. The MSP430G2553 has one built-in I2C hardware module that we can use. To develop, calibrate, and prototype our system, we used the MSP430 LaunchPad from Texas Instruments. The LaunchPad is a development board that allows us to easily program our MSP430G2553 microcontroller using a USB connection.

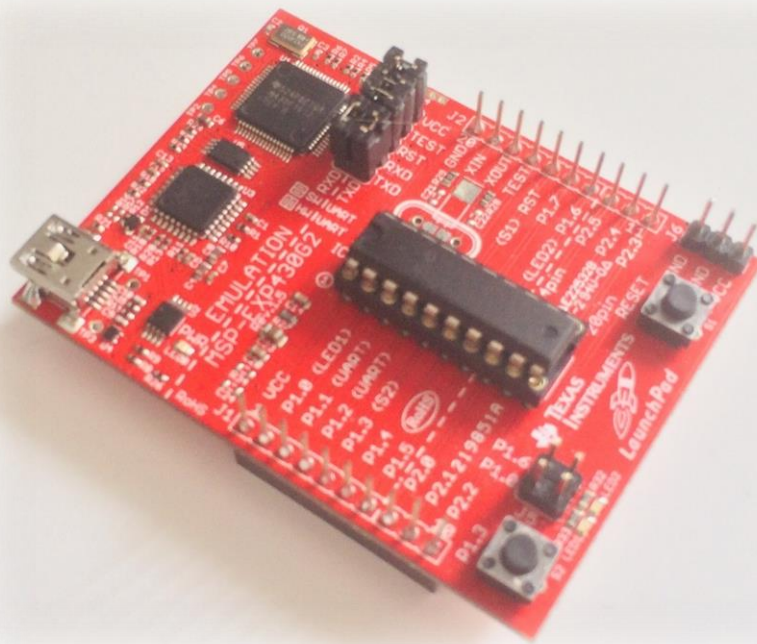


Figure VII.3. *MSP430 Launchpad with an MSP430G2553 microcontroller*

VII.2.1.Code development

To program the MSP430G2553 and use its hardware modules we used the Energia IDE. Energia is a modified version of the Arduino IDE that comes with custom libraries for specific use with development boards from Texas Instruments, such as our MSP430 Launchpad. This is useful for prototyping because we have access to the standard Arduino libraries that we are used to using. The program is written in a simplified C++ language, identical to the Arduino syntax.

To establish the I2C communication we use the Wire library from the standard Arduino libraries. We developed one code per sensor for independent testing and calibration.

We also need to use the ultra-low power capabilities of the MSP430G2553. This requires a lower level of access than what Energia provides us. We used the Code Composer Studio (CCS) IDE developed by Texas Instruments for their LaunchPad. With CCS we program the microcontroller at Register-Transfer Level using C code. Moreover, this IDE provides useful power optimization advice upon compilation. We can also import code made with Energia into CCS. With this environment, we can first develop a prototype using a higher-level language, and then modify it at a lower level for fine-tuning.

The Wire library was useful for the first functional tests of the sensors, but it is relatively slow, and the library is code heavy. We decided to use the [USCI I2C master library](#) from TI application examples. It is a light I2C library written in C, we had to tweak it to make it work with our specific model of microcontroller.

VII.3.Sensor calibration

Each of our sensor might suffer from a bias, or a distortion, in the measurements. This can happen due to an imprecision during the manufacturing, or a physical phenomenon. This bias is generally small, but we need to take it into account nonetheless, which means we need to be able to measure it. In this section we explain the protocols used to calibrate our sensors.

VII.3.1.LM75A – Temperature sensor

To calibrate the LM75A, we need to measure the temperature of an object whose temperature is known to us. We can then compare the theoretical value to the measured value. The easiest example of such an object is water that is changing phase. We know that the temperature of water stays constant during the transition between two states. We also know the temperatures at which water changes phases. So, if we measure the temperature of water during a phase transition, we should get a consistent result. We decide to measure the temperature of ice cubes that are melting.

We assume the atmospheric pressure during the measurements is equal to 1 bar, which means the temperature of water is 0°C when melting.

Measured temperature of the ice-water mixture: 0.25°C with 0.125°C precision

We determined that our LM75A sensor has a static offset of +0.25°C

VII.3.2.LSM9DS1 – Magnetometer

The magnetometer measures the intensity of the magnetic field along 3 directional axes. We can assume that the axes are aligned as they should be, meaning that they form an orthogonal 3D base. With that assumption, we only need to measure the static offset on each axis.

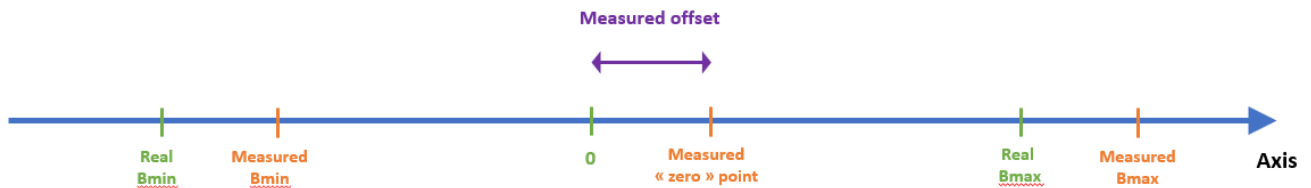


Figure VII.4 *Magnetometer offset schematic*

To measure the offset on one axis, a simple test consists of finding the minimum and maximum values on the axis. The offset is the mean value of those extrema. In order to calibrate the LSM9DS1, we need to measure the extrema values on each axis.

The code used to calibrate the sensor can be found in the [Annex section](#). It performs continuous measurements of the magnetic field on each axis and displays the maximum and minimum values on each axis so far. It also displays the resulting offset of those measurements. While this code runs, we rotate the sensor in every direction until the offset readings are stable.

We got the following values:

X offset: +122.5 mGauss

Y offset: +143.36 mGauss

Z offset: +56.35 mGauss

VII.3.3.AMG88xx Thermal Camera

Due to time constrain, we only managed to make a working test code with the AMG8833 camera, an 8x8 pixels thermal camera with an *Adafruit* library. We managed to test the horizon's detection algorithm with this camera.

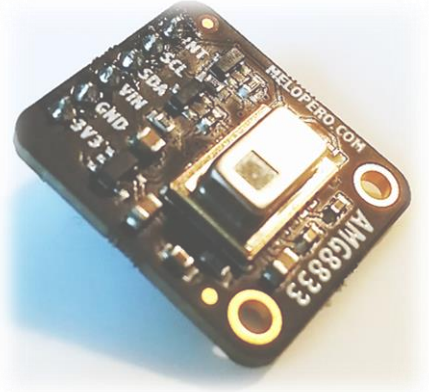


Figure VII.3 AMG8833 thermal camera

We need to extrapolate the position of the satellite with the output of the camera, which is an 8x8 pixel array. To get the horizon, we made a linear regression using the matrix formula

$$(INDEX^T * INDEX)^{-1} * INDEX^T * V \quad (VII.3.3.1)$$

with $V = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \end{pmatrix}$, the matrix filled with the pixel position of detected horizon, and $INDEX = \begin{pmatrix} 1 & 1 \\ 2 & 1 \\ \vdots & \vdots \end{pmatrix}$ with the same length of V .

We get the matrix $C = \begin{pmatrix} a \\ b \end{pmatrix}$, with a, b the coefficients of the linear expression of the horizon

$$y = a * x + b \quad (VII.3.3.2)$$

We then need to calculate corresponding angles of the satellite position compared to the horizon. We choose the angles α and β as described in the following schematic.

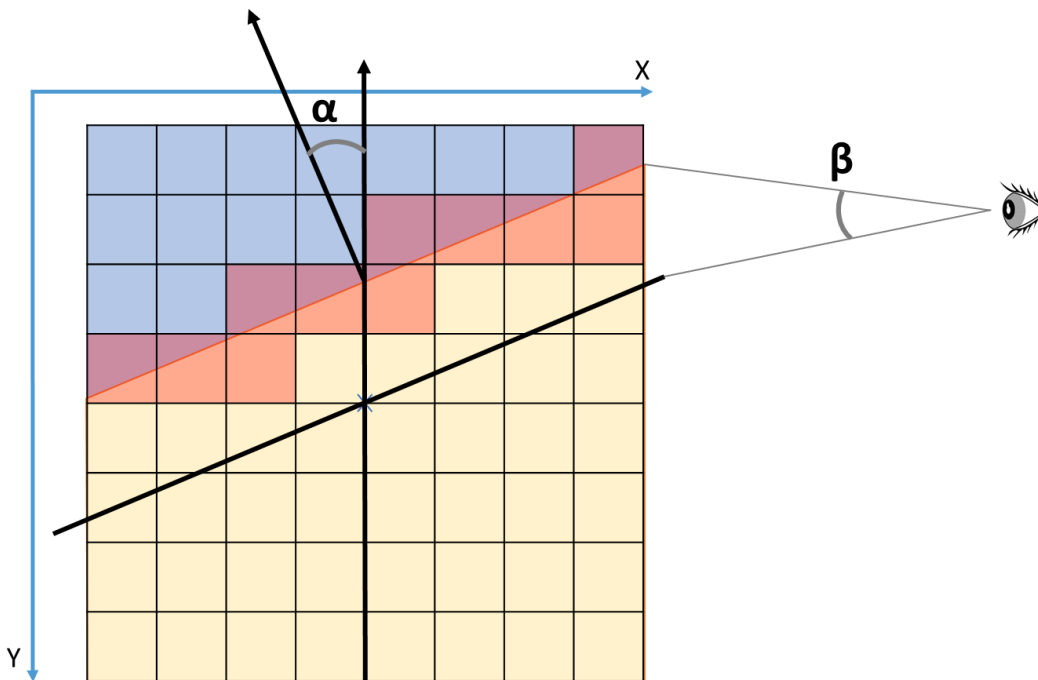


Figure VII.4 Relation between horizon and angles

From our linear expression, we can deduce α and β with the following formula in the XY plane (3.5 corresponds to the center of the screen).

$$\alpha_{xy} = \tan^{-1} \left(\frac{-1}{a} \right) \quad (\text{VII.3.3.3})$$

$$\beta_{xy} = \frac{\pi}{6} * \frac{1}{3.5} * \frac{|3.5(1-a)-b|}{\sqrt{a^2+1}} \quad (\text{VII.3.3.4})$$

And in the YX plane.

$$\alpha_{yx} = \frac{\pi}{2} - \tan^{-1} \left(\frac{-1}{a} \right) \quad (\text{VII.3.3.5})$$

$$\beta_{yx} = \frac{\pi}{6} * \frac{1}{3.5} * \frac{|3.5(1-a)-b|}{\sqrt{a^2+1}} \quad (\text{VII.3.3.6})$$

We did the operation in both planes as, depending of the orientation of the satellite, one projection can be more precise than the other (we have more points in the matrix). We then need to compute the average value.

$$\alpha = \frac{\alpha_{yx} * nbpoints_{yx} + \alpha_{xy} * nbpoints_{xy}}{nbpoints_{yx} + nbpoints_{xy}} \quad (\text{VII.3.3.7})$$

$$\beta = \frac{\beta_{yx} * nbpoints_{yx} + \beta_{xy} * nbpoints_{xy}}{nbpoints_{yx} + nbpoints_{xy}} \quad (\text{VII.3.3.8})$$

As you can see, we need to guard against limits values, such as $a=0$. We can explicitly calculate those values when we do the code implementation.

You can see the translation of these equations to an Arduino test code in the [Annexes, Code, Thermal Camera](#) section.

VII.4. Communication with central unit

VII.4.1. Hardware setup

We do not have a lot of specifics about the central unit. But we know that it requires the values read on the sensors and communicates with the microcontroller on an I2C bus. To develop that communication, we used an Arduino Uno R3 board with an ATMEGA328p controller to act as the central unit. We used 10k Ω pull-up resistors on the I2C lines.

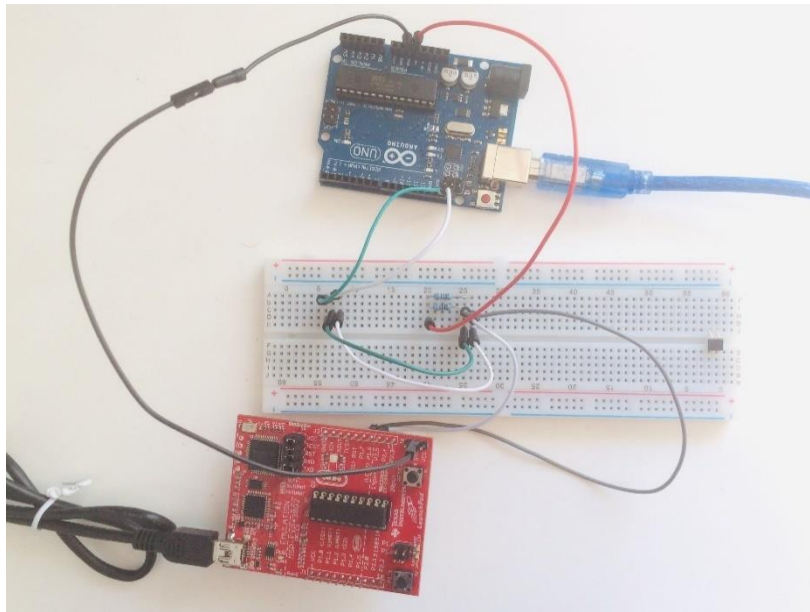


Figure VII.5 I2C communication between an MSP430 launchpad and an Arduino Uno R3.

Because we are using the MSP430G2553 model instead of the MSP430FR59XX, we only have access to one hardware I2C. Therefore, we can't test the sensor network and the communication channel with the central unit at the same time, as they require separate I2C buses. We considered using a virtual I2C interface to solve this problem, but we could not find any library for that purpose in C for MSP430 microcontrollers. Moreover, trying to develop our own virtual I2C library would have been time-consuming. We decided to test the communication protocol and the sensor network separately.

VII.4.2. Software setup

The central unit acts as the master on that I2C bus, and the MPS430 microcontroller acts as a slave. The master sends a command to indicate, for example, that it needs a value from the sensors. The microcontroller receives the command, reads one or multiple values from the sensors, processes the values if needed (ex: linear regression to detect the earth horizon as shown in VIII.3.3), and sends the result back to the central unit.

To make the MSP430G2253 act as a slave in the I2C communication we based our program on the [*TI USCI I2C slave*](#) firmware example from Texas Instruments and modified it to make it compatible with our specific model of microcontroller. This small library is written in C and is very light (~100 lines of code). To make the Arduino act as a master we used the Wire library. We wrote a demo program in which the master tests all the available commands.

VII.5.Prototypes

VII.5.1.Prototype 1

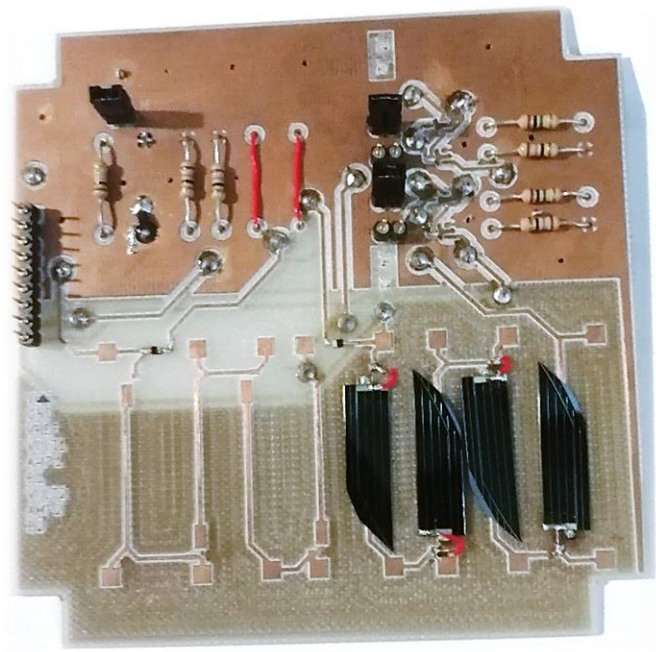


Figure VII.6. First prototype PCB

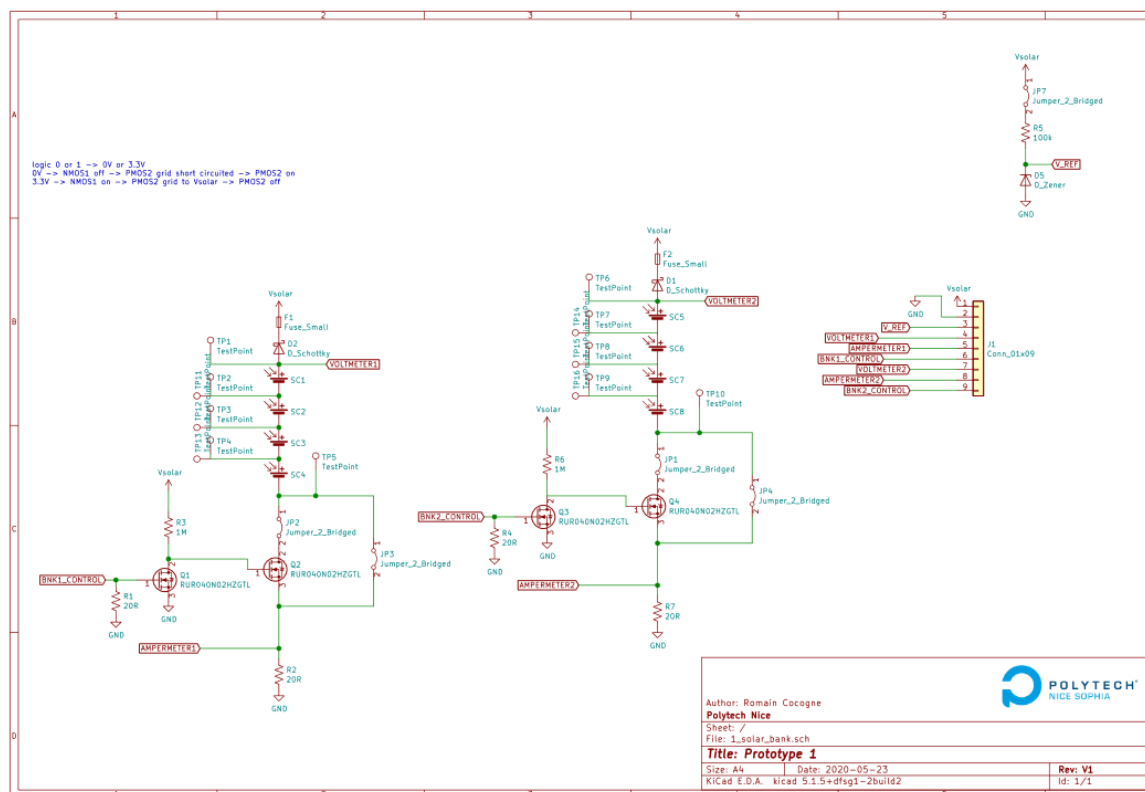


Figure VII.7. First prototype schematic

The first prototype aims at testing only some functionalities of the solar panel.

We want to use this prototype to establish $f(I) = V$ of one solar bank, check if the cut-of circuit of the bank works and if V_{ref} is stable enough.

This prototype highlighted that the polarity of our cells was inverted, so the voltage output was negative.

The voltage reference was not stable enough with wide variation of input voltage, with around 30% of deviation. We revised the [schematic in Prototype 2](#) to integrate a dedicated reference circuit, explained in the [Voltage Reference](#) part.

The footprint of the solar cells was not well adapted and caused some cells to break, so we revised the footprint as seen in the [layout of prototype 2](#).

We tested the amperemeter function and the result was not to our satisfaction. The resistor output was not stable enough to extrapolate running current. We needed a dedicated circuit to measure current with precision. We revised the [schematic in Prototype 2](#) to integrate a dedicated amperemeter circuit, explained in the [Amperemeter](#) part.

This prototype was useful to compute the [power characteristics curve](#) and find the optimum load to put on the solar panels.

We didn't manage to test the cut-of circuit because of delays in components delivery. Before printing the second prototype PCB, it is advised to test this feature. We nonetheless did a Spice simulation and the result was to our satisfaction. You can see below the result with V_3 in input (green trace) and I_{R4} in output (red trace). The circuit lets the current flow through the load if the input is a logic '0' and cut the current if the input is a logic '1'. This configuration allows the microcontroller to boot up before choosing which solar bank to cut.

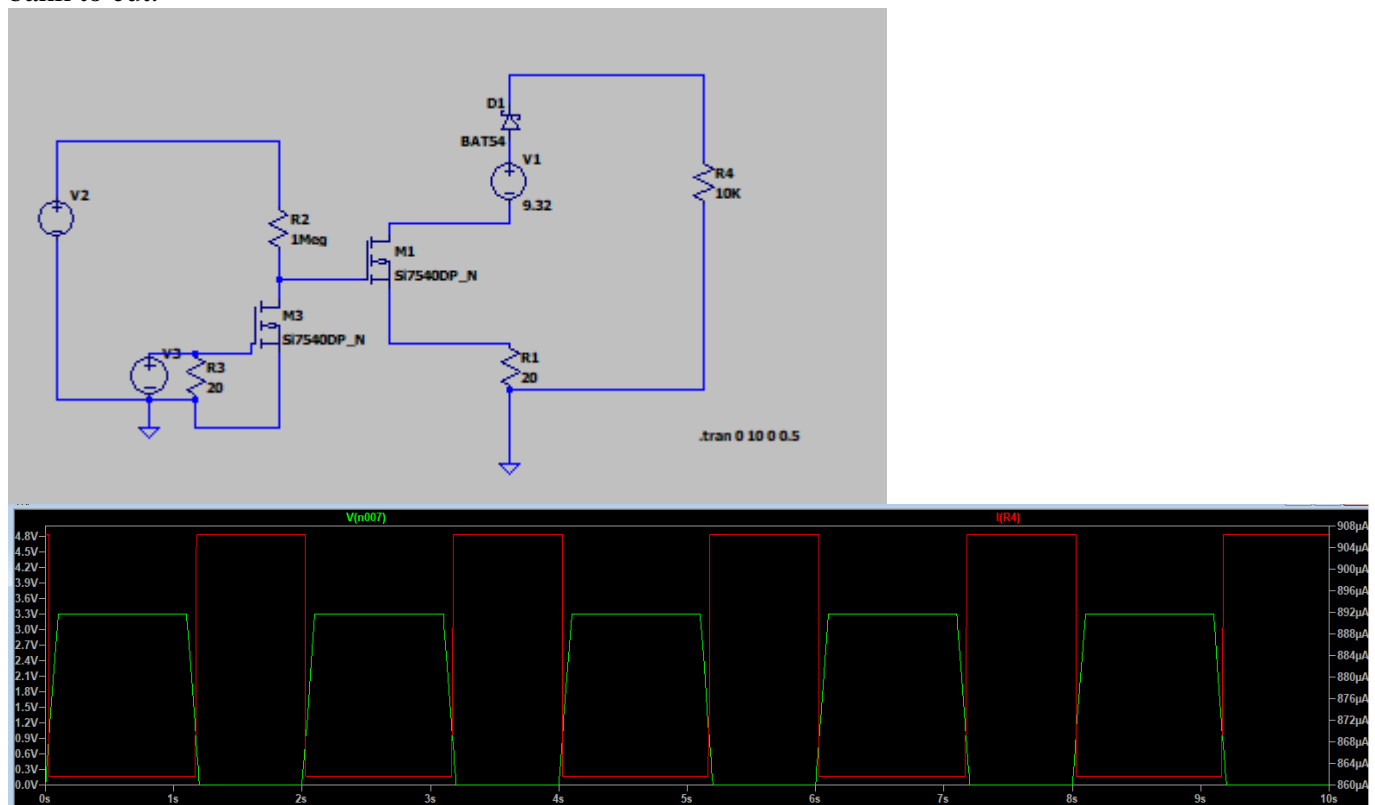


Figure VII.8. *Simulation of cut-of circuit*

VII.5.2.Prototype 2

For this prototype, we implemented all changes pointed out in [prototype 1](#). You can study the schematics following this link to the [Annexes, Schematics](#) part.

We wanted to test the I2C bus and the working of our sensors. This prototype can also be used to test the code integration and the main software.

Due to time constrains, we didn't print the PCB, but we finished the layout. All the passives components are in 0805 dimensions to ease the soldering further prototypes can use smaller packages to gain space. We have a 2 layers board to reduce production cost, but *in fine* it should be best to go with a 6 layers board with internal layers as ground or power plane. This will keep all sensitive components protected from space hazards.

We added one more solar bank to test the cube layout you can see below. This should be an optimized layout according to **FigureVII.11**.

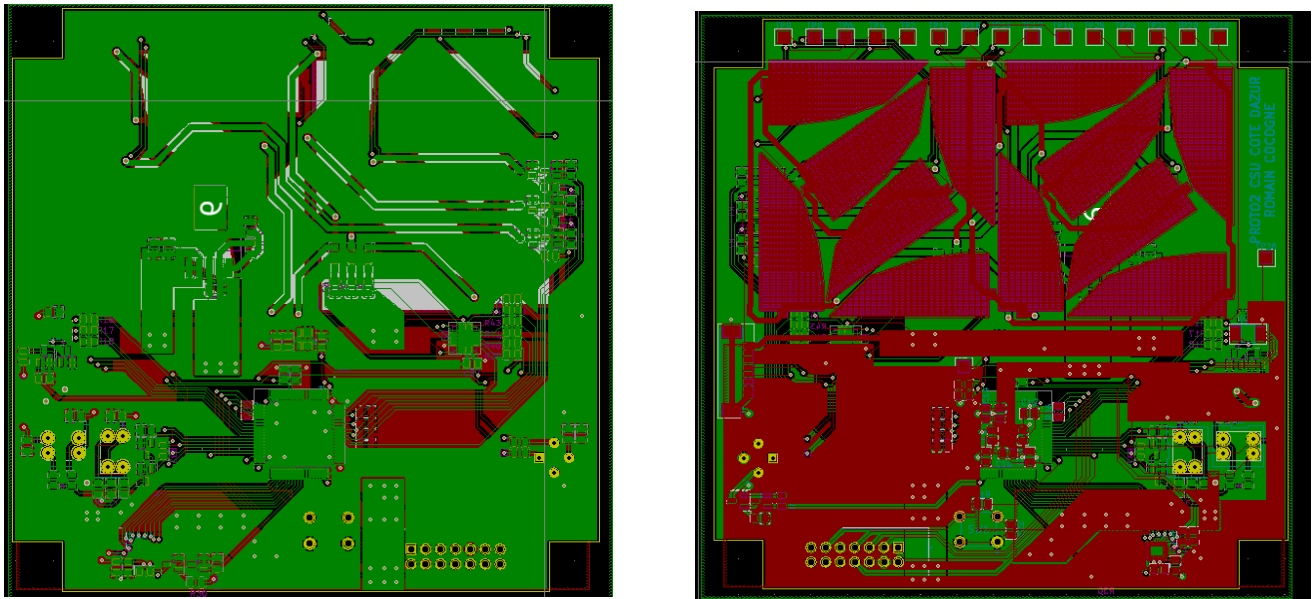


Figure VII.11. *Second prototype layout*

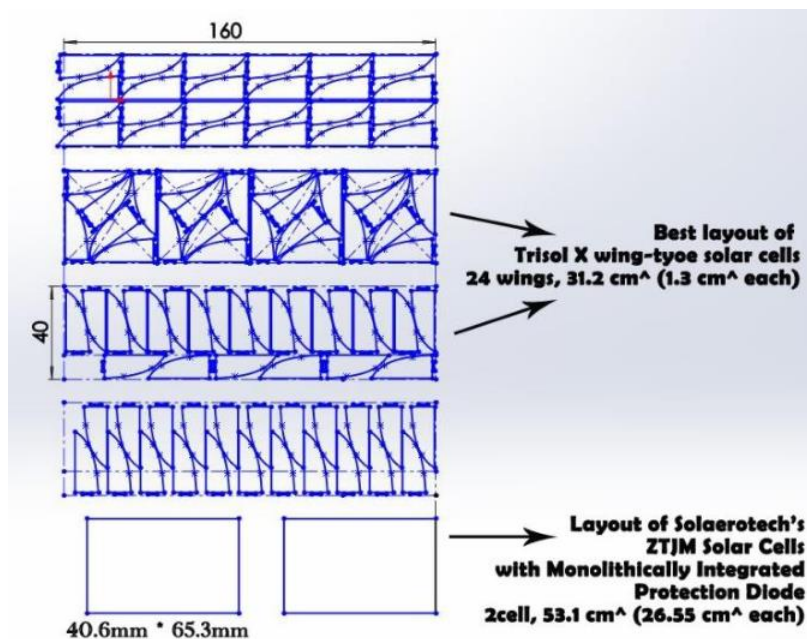


Figure VII.9. *TrisolX layout comparison*

The prototype is delivering the solar bank voltage as well as the I2C bus pins. This bus can't be used to test I2C communication with the CubeSat's main microcontroller. Its only use is to monitor the communication between our microcontroller and the sensors array.

The microcontroller is programmable through a JTAG connection.

To measure all the different voltages on this board, you can use the tests pads marked on the front face.

Here are some improvements and new features to add in the next prototype.

- Implement an EEPROM next to the microcontroller to keep permanent logs of sensors and panels state
- Add LED indicators to check if the board is powered up
- Add access pin to the microcontroller
- Add a second I2C bus output to try communicating with the main microcontroller

VIII.Challenges

VIII.1. Technological challenges

One of the challenges that we faced was that we needed to learn how to use the MSP430 microcontroller and its embedded hardware modules. Setting up an efficient programming environment for prototyping took longer than we expected. The Launchpad development board does not benefit from a large community support, unlike Arduino boards for example. We could not always find existing libraries to answer our needs, which led us to rewrite libraries using register transfer operations. This made the development take more time than what we originally expected.

The routing of the second prototype was also a big challenge. Indeed, the board is inside a 100cm² square and a lot of components needs to fit in there. The 2 layers constrain did add a lot of complexity because some tracks were intertwining and crossing each other most of the time. Changing the orientation and disposition of the sensors helped resolve most of the problems, but sometimes we had no other choice but to modify the microcontroller pinout.

VIII.2. Adapting to the lockdown

The global lockdown that started on March 16th in France in reaction to the COVID-19 pandemic was a major slowdown in the development of our project, due to its hardware nature. We had to reconsider our objectives and our organization.

We split the components so that we could work on separate parts and continue the development. Romain kept the solar cells, the halogen lamp, the first prototype, and soldering materials to be able to perform more tests and develop the schematic and routing for the second prototype. Quentin kept the microcontroller development board, the thermometer, and the magnetic sensor to develop the sensor interfacing and the I2C communication. Regarding the thermal cameras, we both kept one model, the 8x8 AMG88 for Romain and the 32x24 MLX90640 for Quentin. We also both kept one photo diode to try to develop the solar sensor. Fortunately, we had some electronic components at our homes, as well as Arduino boards that were very useful for prototyping.

To keep in touch with our tutor Florentin Millour, we organized weekly group calls using Skype and Zoom. It was essential to keep Mr. Millour informed on the advancements of the project, and to get his advice on where to put our focus in the development.

We tried to issue a new RS order, but due to unforeseen delays caused by current circumstances, we were not able to receive the components in time. Our tutor did provide us with some of his personal hardware such as his own MSP430 development board but there was some mismatch between the microcontrollers model and some features were not tested.

We ultimately had to lower our objectives. As we did not have access to the electronic lab anymore, we were unable to produce our second prototype. Instead we decided to focus on making the core parts of the project functional.

IX. Current state of the project

The core parts of our project, like the schematic for the solar cells or the I2C communication, are working as intended. The next step in the project would be to produce a prototype that links those core parts to test their interactions. We could not achieve all the objectives that we had planned. However, we provided a baseline of the different parts so that the project can be continued in the future.

In the following sections we summarize which parts of the project we consider functional, and which parts require further work.

IX.1. Parts that are functional

IX.1.1. Solar cells electronic schematic

We designed an electronic schematic to manage the solar cells. The solar panel groups the solar cells into banks that deliver an 8.2V output with a max current of 43,7mA per bank. Individual banks can be disconnected by toggling an input on MOSFET transistors.

Simulations of this electronic system were performed on LTSpice and were successful. However, we could not perform practical tests as we could not access the equipment to produce a prototype. Further testing would be needed to ensure that the system works as intended with the microcontroller ports.

IX.1.2. Thermometer, magnetic sensor

We wrote functional programs in C for both the thermometer and the magnetic sensor. The program files are light and use direct register access on the hardware I2C module, making the execution relatively fast. As of now, each sensor has its own program in separate CCS project directories, but the two programs can be easily combined assuming the sensors are on the same I2C bus. The library used to exploit the I2C module can be used as a baseline to write programs for the other sensors.

IX.1.3. 8x8 thermal camera

We performed tests on the AMG8833 8x8 thermal camera and implemented the algorithm described in [Sensor calibration Section](#). Those first tests were successful in detecting the horizon. However, this sensor was only tested on an Arduino Uno R3 board, using a specific library from Adafruit written in C++. Tests on the MSP430 Launchpad could not be performed due to the lockdown.

Moreover, further tests need to be done to measure the precision of the resulting values α and β and determine if the resolution provided by an 8x8 grid is enough.

IX.1.4. Communication with the central unit

The communication protocol between the microcontroller and the central unit is functional. As of now, four commands are implemented, with one command accepting one parameter. More specifications on the central unit could require adding more commands to the command set. This can be done easily as the code structure that handles commands on the microcontroller is flexible. However, the code should be tested with a network of sensors using a MSP430FR59XX microcontroller that has multiple hardware I2C modules.

IX.1.5. PCB

We managed to complete one working prototype validating the following requirements:

- The power output
- The circuit protection of solar cells

The other hardware requirements were not tested but the groundwork is already done. The PCB layout of a nearly full featured board is ready to be printed and tested. The layout was checked by the website [JLCPCB](#) and deemed as sound.

IX.2. Parts that require further work

IX.2.1. Using the ultra-low power features of the microcontroller

We were not able to measure the consumption of the microcontroller when it is in function. We also did not exploit the low-power modes in our prototypes, as we wanted to make the design functional before trying to reduce power consumption. However, the environment we used to develop our code, Code Composer Studio, has an “Optimization advice” feature that helps reduce power consumption and optimize computation time. Once we have a working prototype, we can quickly optimize power consumption using this feature

IX.2.2. 32x24 Thermal camera

As mentioned above, the 8x8 camera is functional but might not meet our requirements regarding precision. We attempted to use the 32x24 MLX90640 camera and tried prototyping using an Arduino library from Adafruit. However, the program for that camera could not be tested due to memory limitations. The temporary storage that we need for the linear regression algorithm is taking too much space. To use this camera, we would need to write the sensor interfacing program in C and tweak the algorithm so that it uses less temporary storage.

IX.2.3. Solar sensors

We could not design a program to test the solar sensors due to a lack of time. This part needs to be developed as it is an essential sensor to determine the orientation of the satellite.

IX.2.4. Network of sensors

We need to write a program that can read data from multiple sensors on the same I2C bus. We would only need to combine the programs that read one sensor, and make sure each address is assigned to only one sensor. Furthermore, we need to test the sensor network when the microcontroller also communicates with the central unit. We want to make sure that reading and processing the sensor values is fast enough to provide real-time reading to the central unit.

X. References

You can find all datasheets and source documents, as well as original Kicad design and Arduino code on the Github of this project.

<https://github.com/RomainCocogne/NanoSat.git>

Other links:

Texas Instrument, “MSP430FR599x, MSP430FR596x Mixed-Signal Microcontrollers datasheet”, [SLASE54C]

<https://www.ti.com/lit/ds/symlink/msp430fr5949.pdf?ts=1590224343299>

Texas Instruments, “MSP430FR58xx, MSP430FR59xx, and MSP430FR6xx Family User Guide”, [SLAU367P]

<https://www.ti.com/lit/ug/slau367p/slau367p.pdf?ts=1590224617008>

Texas Instruments; “Using the USCI I2C Master” [SLAA382A]

<http://www.ti.com/lit/an/slaa382a/slaa382a.pdf>

Texas Instruments; “Using the USCI I2C Slave” [SLAA383]

<https://datasheet.datasheetarchive.com/originals/library/Datasheet-080/DASF004576.pdf>

CubeSat design specification

https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf

TrisolX layout

<https://d3i71xaburhd42.cloudfront.net/6ffac9621ad9e26aa4264a1be23f66f674bda921/10-Figure14-1.png>

GOMspace nanopower p31u, Power supply

<https://gomspace.com/UserFiles/Subsystems/datasheet/gs-ds-nanopower-p31u-27.pdf>

XI. Annexes

XI.1. Code.....	32
XI.1.1. Thermal Camera	32
XI.1.2. Magnetic sensor calibration.....	36
XI.1.3. USCI_I2C library	37
XI.1.3.1. USCI_I2C_master library	37
XI.1.3.2. USCI_I2C_slave library	38
XI.2. Schematics	39
XI.2.1. Prototype 2.....	39

XI.1. Code

XI.1.1. Thermal Camera

amg8833_main	LinearRegression_mtx.h	Regression_lineaire.cpp	Regression_lineaire.h
--------------	------------------------	-------------------------	-----------------------

```
#include <MatrixMath.h>
#include <Wire.h>
#include <Adafruit_AMG88xx.h>
#include <math.h>

Adafruit_AMG88xx amg;

const uint8_t ROW_SIZE = 8;
const uint8_t COL_SIZE = 8;
const float THRESHOLD = 10;

float pixels[ROW_SIZE][COL_SIZE];
LinearRegression horizon(0, COL_SIZE);
mtx_type horizonMTX_x[ROW_SIZE];
mtx_type horizonMTX_y[COL_SIZE];

uint8_t rank_mat_x, rank_mat_y;

void setup() {
    Serial.begin(9600);
    Serial.println(F("\nAMG88xx pixels"));

    bool status;

    // default settings
    status = amg.begin();
    if (!status) {
        Serial.println("Could not find a valid AMG88xx sensor, check wiring!");
        while (!(status = amg.begin()));
    }

    Serial.println("-- Pixels Test --");

    Serial.println();

    delay(100); // let sensor boot up
}
```

Figure XI.1. *Main test code AMG8833, part 1*


```

void loop() {
    //read all the pixels
    amg.readPixels((float*)pixels);
    empty_coeff();

    find_horizon();
    //print_pixels();

    print_angles();

    //delay a second
    delay(500);
}

void find_horizon() {
    for (uint8_t x = 0; x < ROW_SIZE - 1; ++x) {
        for (uint8_t y = 0; y < COL_SIZE - 1; ++y) {
            if ((pixels[x + 1][y]) < 0 && (pixels[x][y]) > 0) {
                horizon.learn(x + 1, y);
                horizonMTX_x[x + 1] = -y;
                ++rank_mat_x;
            }
            if ((pixels[x][y + 1]) < 0 && pixels[x][y] > 0) {
                horizon.learn(x, y + 1);
                horizonMTX_y[y + 1] = -x;
                ++rank_mat_y;
            }
            if ((pixels[x + 1][y]) > 0 && (pixels[x][y]) < 0) {
                horizon.learn(x, y);
                horizonMTX_x[x] = y;
                ++rank_mat_x;
            }
            if ((pixels[x][y + 1]) > 0 && pixels[x][y] < 0) {
                horizon.learn(x, y);
                horizonMTX_y[y] = x;
                ++rank_mat_y;
            }
        }
    }
}

```

Figure XI.2. Main test code AMG8833, part 2

```

void empty_coeff () {
    horizon.reset();
    rank_mat_x = 0;
    rank_mat_y = 0;
}

void print_pixels () {
    Serial.print("[");
    for (uint8_t x = 0; x < ROW_SIZE; ++x) {
        for (uint8_t y = 0; y < COL_SIZE; ++y) {
            Serial.print(pixels[x][y]);
            Serial.print(", ");
        }
        Serial.println();
    }
    Serial.println("]");
    Serial.println();
}

void print_angles() {

    mtx_type coeffs_x[2] = {0, 0};
    mtx_type coeffs_y[2] = {0, 0};
    if (rank_mat_x > 1) {
        LinearRegressionMTX linear_horizon (horizonMTX_x, rank_mat_x);
        linear_horizon.getCoeffs(coeffs_x);
    }
    if (rank_mat_y > 1) {
        LinearRegressionMTX linear_horizon (horizonMTX_y, rank_mat_y);
        linear_horizon.getCoeffs(coeffs_y);
    }

    double alpha = 0;
    double beta = 0;

    if (coeffs_x[0] == 0) {
        alpha = PI / 2;
        beta = PI / 6 / 3.5 * (3.5 * (1 - coeffs_y[0]) - coeffs_y[1]) / (sqrt(coeffs_y[0] * coeffs_y[0] + 1));
    }
    else if (coeffs_y[0] == 0) {
        alpha = 0;
        beta = PI / 6 / 3.5 * (3.5 * (1 - coeffs_x[0]) - coeffs_x[1]) / (sqrt(coeffs_x[0] * coeffs_x[0] + 1));
    }
    else if (coeffs_x[0] != 0 && coeffs_y[0] != 0) {
        alpha = ((PI / 2 - atan2(1, coeffs_y[0]) * rank_mat_x) / (rank_mat_y + rank_mat_x);
        beta = PI / 6 / 3.5 * ((3.5 * (1 - coeffs_x[0]) - coeffs_x[1]) / (sqrt(coeffs_x[0] * coeffs_x[0] + 1)) * rank_mat_x
            + (3.5 * (1 - coeffs_y[0]) - coeffs_y[1]) / (sqrt(coeffs_y[0] * coeffs_y[0] + 1)) * rank_mat_y)
            / (rank_mat_y + rank_mat_x);
    }

    Serial.println("[ ");
    Serial.print("\tangle alpha : "); Serial.println(alpha * 180 / PI);
    Serial.print("\thorizontal : "); Serial.println(beta * 180 / PI);
    Serial.println("]");
}

```

Figure XI.3. Main test code AMG8833, part 3

amg8833_main	LinearRegression_mbx.h	Regression_lineaire.cpp	Regression_lineaire.h
--------------	------------------------	-------------------------	-----------------------

```

#pragma once
#include <MatrixMath.h>

class LinearRegressionMTX {
private:
    mtx_type coeffs_[2];

    void compute_coeffs (mtx_type values[], uint8_t INDEX_SIZE) {
        mtx_type index[INDEX_SIZE][2];
        for (int i = 0; i < INDEX_SIZE; ++i) {
            index[i][0] = i;
            index[i][1] = 1;
        }

        mtx_type index_transpose[2][INDEX_SIZE];
        Matrix.Transpose((mtx_type*)index, INDEX_SIZE, 2, (mtx_type*)index_transpose);

        mtx_type indexprod[2][2];
        Matrix.Multiply((mtx_type*)index_transpose, (mtx_type*)index, 2, INDEX_SIZE, 2, (mtx_type*)indexprod);
        Matrix.Invert((mtx_type*)indexprod, 2);

        mtx_type indexprod2[2][INDEX_SIZE];
        Matrix.Multiply((mtx_type*)indexprod, (mtx_type*)index_transpose, 2, 2, INDEX_SIZE, (mtx_type*)indexprod2);

        Matrix.Multiply((mtx_type*)indexprod2, (mtx_type*)values, 2, INDEX_SIZE, 1, (mtx_type*)coeffs_);

        //Matrix.Print((mtx_type*)values, INDEX_SIZE, 1, "values");
        //Matrix.Print((mtx_type*)indexprod2, 2, INDEX_SIZE, "indexprod2");
        //Matrix.Print((mtx_type*)coeffs_, 2, 1, "coeffs");
    }

public:
    LinearRegressionMTX() {}
    LinearRegressionMTX(mtx_type values[], uint8_t INDEX_SIZE) {
        compute_coeffs(values, INDEX_SIZE);
    }

    void resetCoeffs() {
        coeffs_[0] = 0;
        coeffs_[1] = 0;
    }

    void calculate(mtx_type values[], uint8_t INDEX_SIZE) {
        compute_coeffs(values, INDEX_SIZE);
    }

    void getCoeffs(mtx_type coeffs[2]) {
        coeffs[0] = coeffs_[0];
        coeffs[1] = coeffs_[1];
    }
};

```

Figure XI.4. *Linear Regression class test code AMG8833*

XI.1.2.Magnetic sensor calibration

```
void loop() {
  for (int i=0; i<10; ++i){
    // Begin transmission with this slave device
    Wire.beginTransmission(SLAVE_ADDR);
    // Ask to access the values registers
    Wire.write(OUT_X_L); //Write value in Tx buffer
    if ( Wire.endTransmission() == 0){ //Send value and check for success
      // Get the value of the registers we asked for
      Wire.requestFrom(SLAVE_ADDR, (uint8_t) 6);
      data = Wire.read(); // LSB
      data |= Wire.read() << 8; // MSB
      B_x = data*0.14;
      data = Wire.read(); // LSB
      data |= Wire.read() << 8; // MSB
      B_y = data*0.14;
      data = Wire.read(); // LSB
      data |= Wire.read() << 8; // MSB
      B_z = data*0.14;
    }

    //Update max for each coordinate
    if (B_z > B_zmax)
      B_zmax = B_z;
    if (B_y > B_ymax)
      B_ymax = B_y;
    if (B_x > B_xmax)
      B_xmax = B_x;

    //Update min for each coordinate
    if (B_z < B_zmin)
      B_zmin = B_z;
    if (B_y < B_ymin)
      B_ymin = B_y;
    if (B_x < B_xmin)
      B_xmin = B_x;

    delay(10);
  }

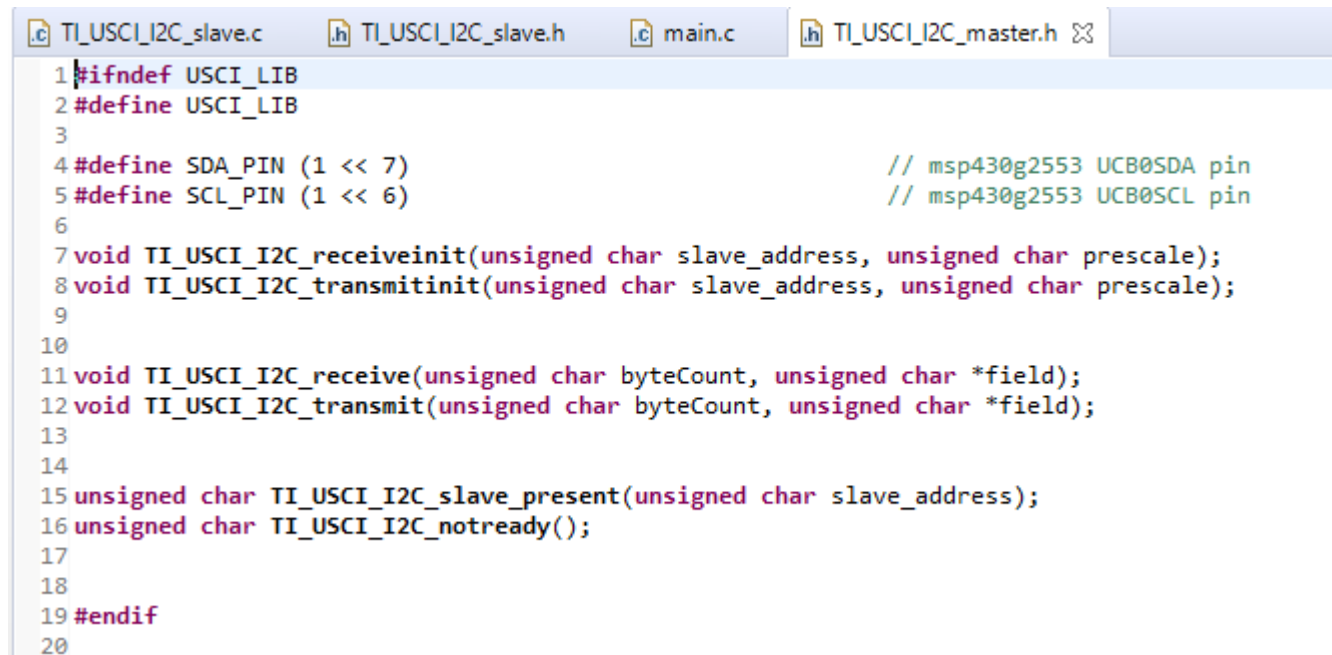
  //Calculate offset
  off_x = (B_xmax + B_xmin)*0.5;
  off_y = (B_ymax + B_ymin)*0.5;
  off_z = (B_zmax + B_zmin)*0.5;

  Serial.print("Offset sur x : "); Serial.print(off_x);
  Serial.print(" Offset sur y : "); Serial.print(off_y);
  Serial.print(" Offset sur z : "); Serial.println(off_z);
}
```

Figure XI.5 Calibration program for the LSM9DS1 magnetic sensor

XI.1.3. USCI_I2C library

XI.1.3.1. USCI_I2C_master library



```
1 #ifndef USCI_LIB
2 #define USCI_LIB
3
4 #define SDA_PIN (1 << 7) // msp430g2553 UCB0SDA pin
5 #define SCL_PIN (1 << 6) // msp430g2553 UCB0SCL pin
6
7 void TI_USCI_I2C_receiveinit(unsigned char slave_address, unsigned char prescale);
8 void TI_USCI_I2C_transmitinit(unsigned char slave_address, unsigned char prescale);
9
10
11 void TI_USCI_I2C_receive(unsigned char byteCount, unsigned char *field);
12 void TI_USCI_I2C_transmit(unsigned char byteCount, unsigned char *field);
13
14
15 unsigned char TI_USCI_I2C_slave_present(unsigned char slave_address);
16 unsigned char TI_USCI_I2C_notready();
17
18
19 #endif
20
```

Figure XI.6 *USCI_I2C_master library interface*

Functional description of the library:

TI_USCI_I2C_receiveinit: Configures the I2C module in receiver mode to receive a message from a slave

TI_USCI_I2C_transmitinit: Configures the I2C module in transmitter mode to send a message to a slave

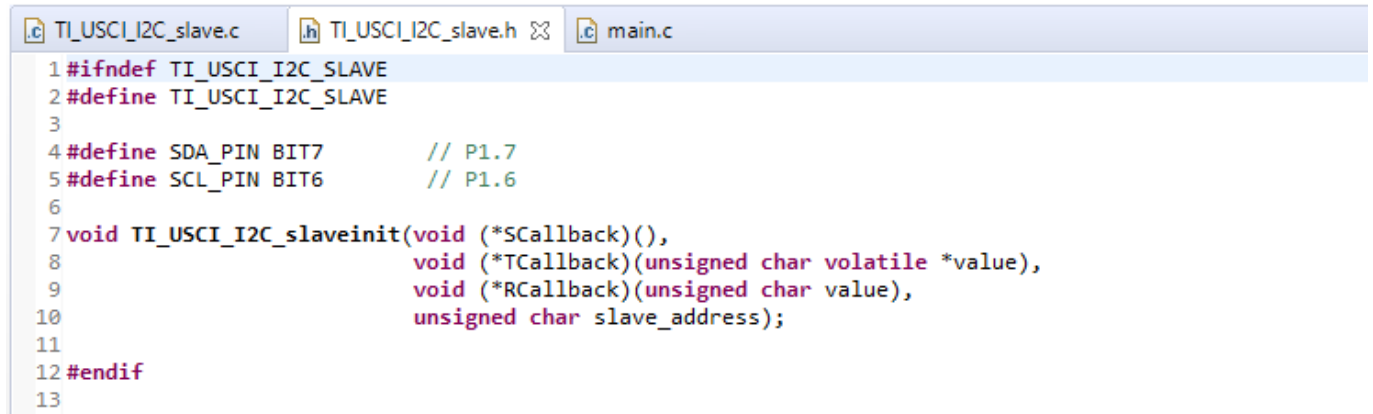
TI_USCI_I2C_receive: Stores *byteCount* received bytes at the memory space pointed by field

TI_USCI_I2C_transmit: Transmits *byteCount* bytes from the memory space pointed by field

TI_USCI_I2C_slave_present: Checks if the address *slave_address* corresponds to a slave on the bus.

TI_USCI_I2C_not_ready: Checks if the bus is busy

XI.1.3.2.USCI_I2C_slave library



```
1 #ifndef TI_USCI_I2C_SLAVE
2 #define TI_USCI_I2C_SLAVE
3
4 #define SDA_PIN BIT7          // P1.7
5 #define SCL_PIN BIT6          // P1.6
6
7 void TI_USCI_I2C_slaveinit(void (*SCallback)(),
8                             void (*TCallback)(unsigned char volatile *value),
9                             void (*RCallback)(unsigned char value),
10                             unsigned char slave_address);
11
12 #endif
13
```

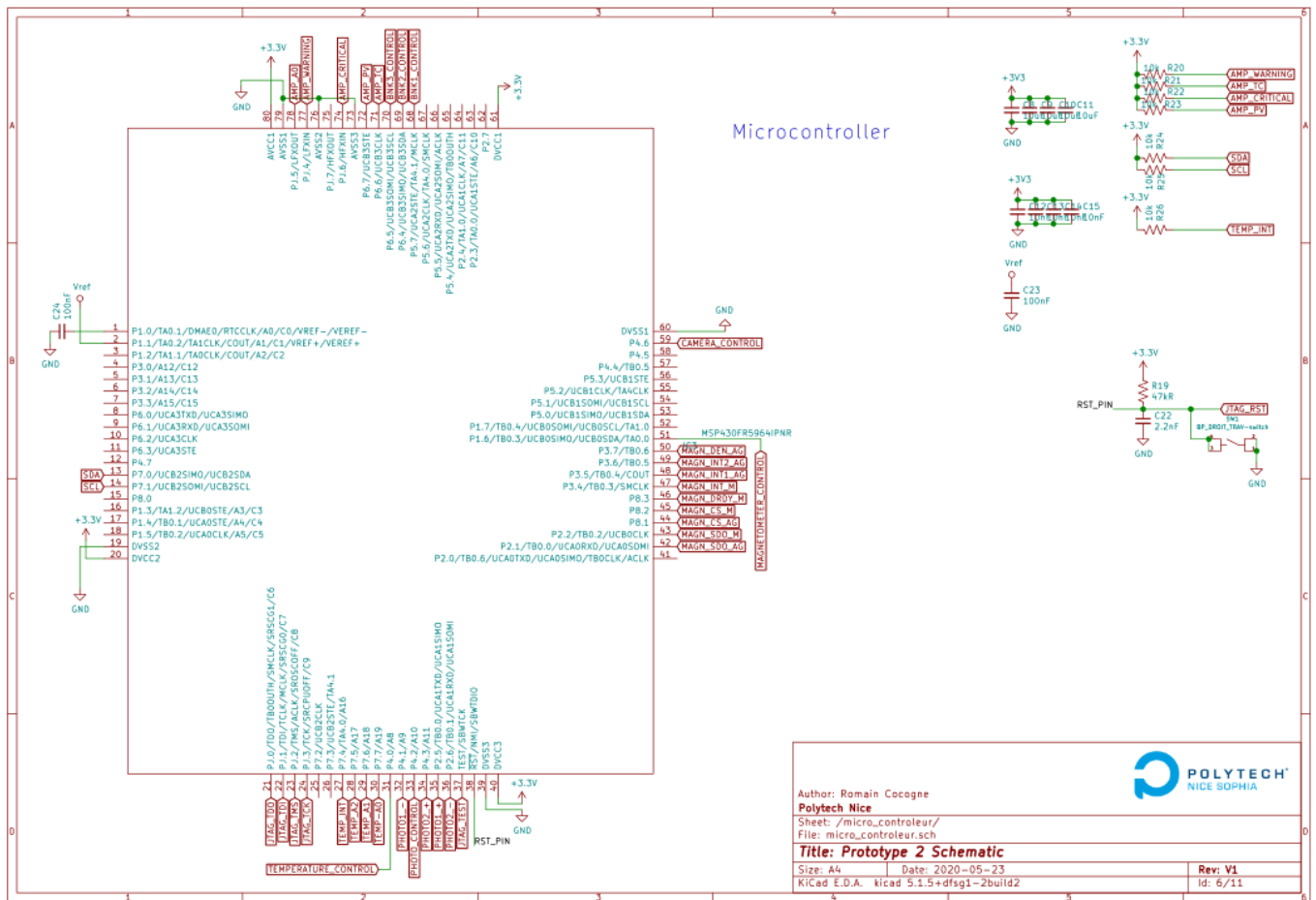
Figure XI.7 *USCI_I2C_slave library interface*

Functional description of the library:

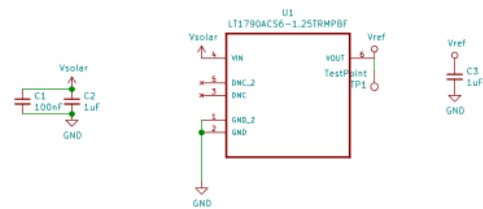
TI_USCI_I2C_slavetinit: Initializes the I2C module in slave mode with the address *slave_address*.

- *SCallback*: function called when a start condition is detected on the bus
- *TCallback*: function called for every byte of data requested by the master
- *RCallback*: function called for every byte of data received from the master

XI.2.1.Prototype 2



Voltage reference



Author: Romain Cocagne
Polytech Nice

Sheet: /ref_tension/
File: ref_tension_1-25V.sch

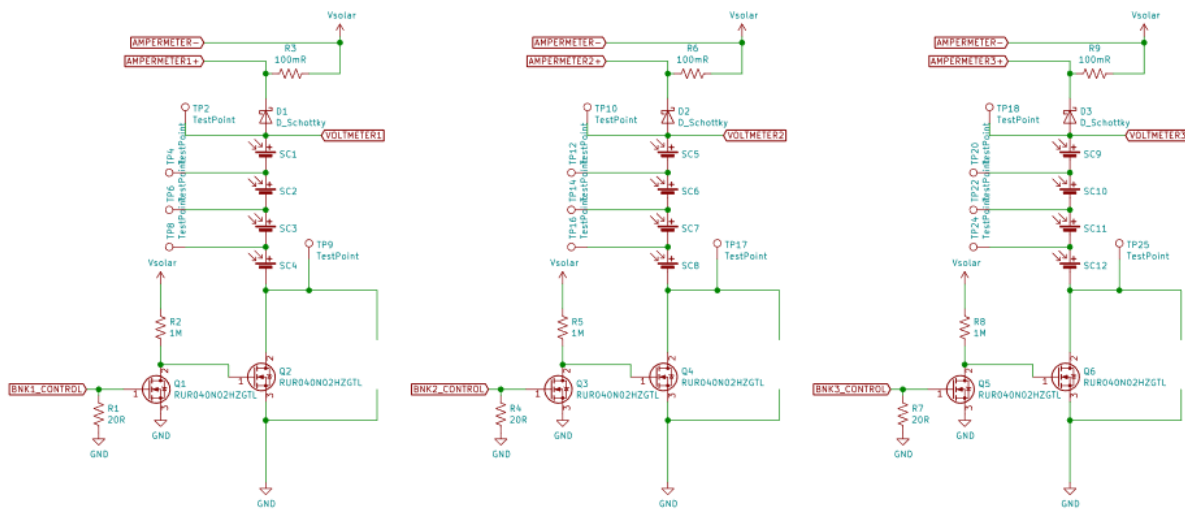
Title: **Prototype 2 Schematic**

Size: A4 Date: 2020-05-23
KiCad E.D.A. kicad 5.1.5+dfsg1-2build2

Rev: V1
Id: 2/11



logic 0 or 1 -> 0V or 3.3V
0V -> NMOS1 off -> PMOS2 grid short circuited -> PMOS2 on
3.3V -> NMOS1 on -> PMOS2 grid to Vsolar -> PMOS2 off



Solar Bank

Author: Romain Cocagne
Polytech Nice

Sheet: /solar_rows/
File: solar_rows.sch

Title: **Prototype 2 Schematic**

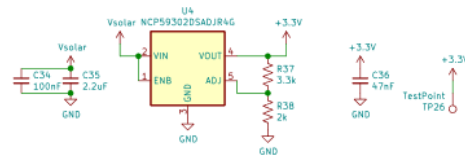
Size: A4 Date: 2020-05-23
KiCad E.D.A. kicad 5.1.5+dfsg1-2build2

Rev: V1
Id: 3/11



The schematic diagram illustrates the photo diode circuit. It features a 3.3V supply connected to a 10k resistor (R33) and a 100nF capacitor (C30). The photo diode (Q13) is connected to the output signal (PHOTO1, PHOTO2) through a 10k resistor (R35) and a 100nF capacitor (C32). The output signal is connected to the output signal (PHOTO1, PHOTO2) through a 10k resistor (R36).

Voltage regulator



Author: Romain Cocogne
Polytech Nice

Sheet: /regulateur_tension/
File: regulateur_tension.sch

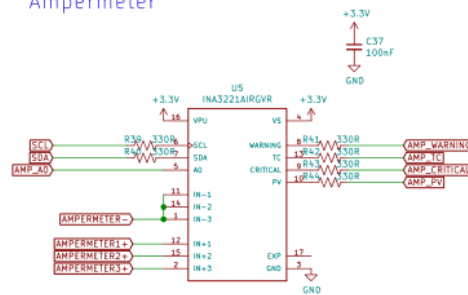
Title: Prototype 2 Schematic

Size: A4 Date: 2020-05-23
KiCad E.D.A. kicad 5.1.5+dfsg1-2build2

Rev: V1
Id: 9/11



Amperemeter



Author: Romain Cocogne
Polytech Nice

Sheet: /amperemeter/
File: amperemeter.sch

Title: Prototype 2 Schematic

Size: A4 Date: 2020-05-23
KiCad E.D.A. kicad 5.1.5+dfsg1-2build2

Rev: V1
Id: 10/11



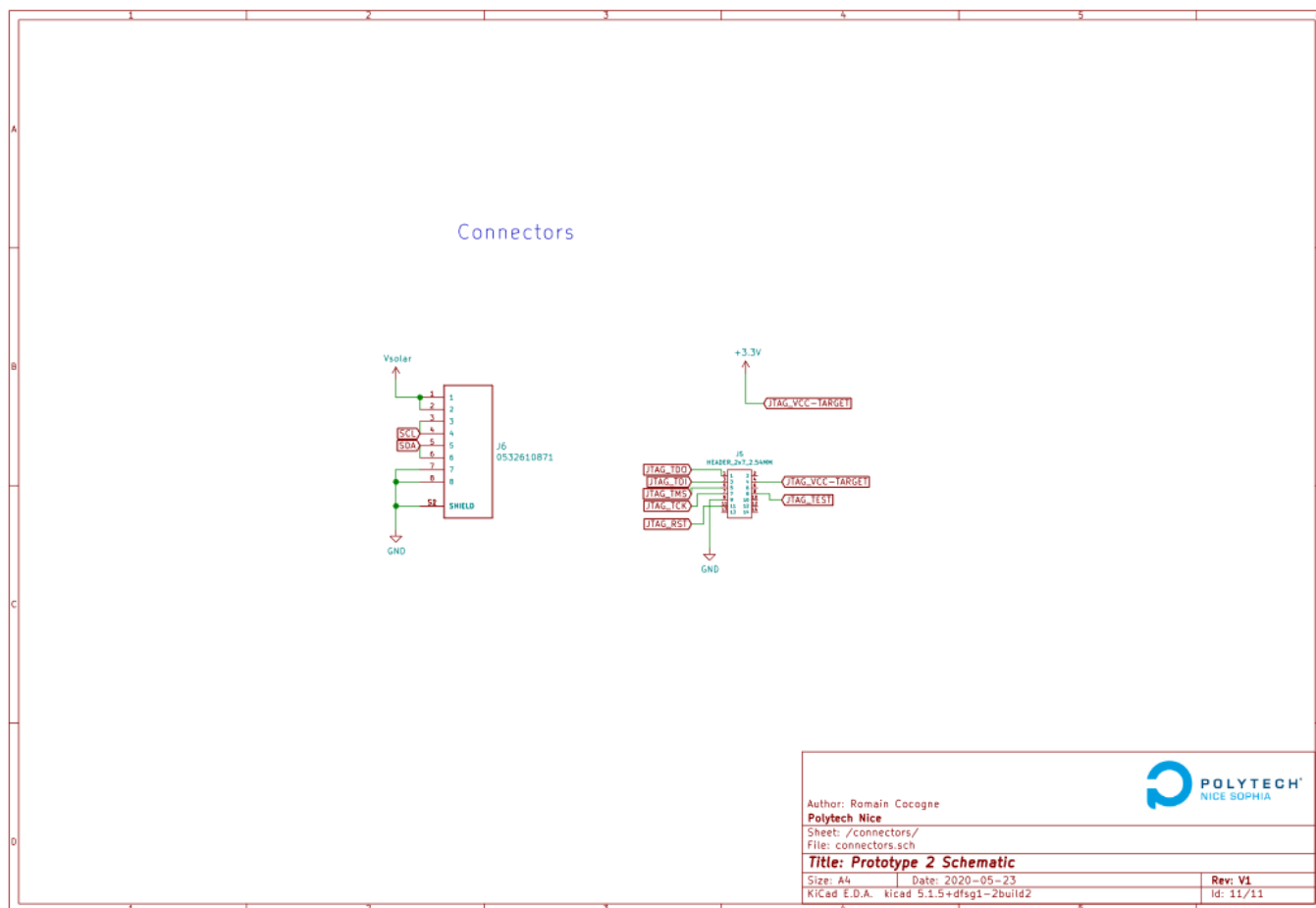


Figure XI.2. *Second prototype schematic*