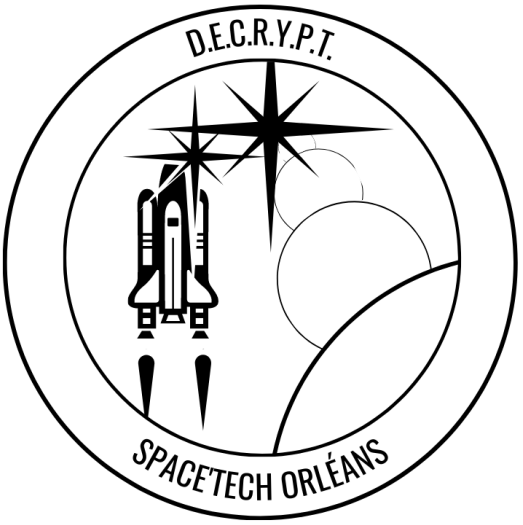
POLYTECH ORLÉANS
École polytechnique de l'Université d'Orléans

UNIVERSITE D'ORLEANS

DATASHEET

moDular and Embedded platform for Control and telemetRY of aerosPace sysTems (DECRYPT)



15/01/2022

TABLE OF CONTENTS

Table of Figures

Table of Tables

1 General presentation of the project

The DECRYPT (moDular and Embedded platform for Control and telemetRY of aerospace sysTems) is a modular and universal embedded system for mini and experimental rockets of the students association Space'Tech Orléans.

This association has for main goal the launch of mini and experimental rockets during an international event, the C'Space. The whole process, from design to launch is carried out by our members.

A main aspect of rocket conception is the embedded system. Figure 1 shows how electronics are usually used during flights.

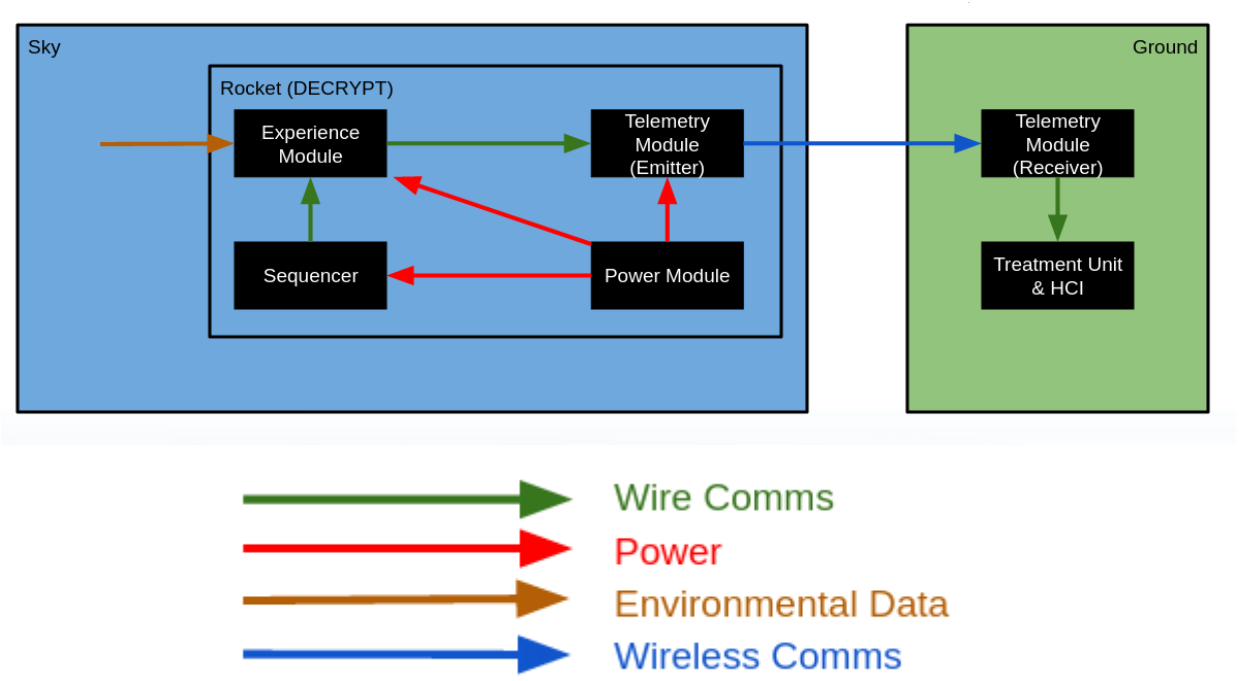


Figure 1 : Global architecture

During the flight, the Sequencer counts time from lift-off and manages flight phases by freeing the parachute at climax. It also sends flight statuses (awaiting lift-off, ascending, descending) via UART transmission to the Experience Module (EXP MOD). Note that the modules should be electrically isolated from each other (via an optocoupler for instance).
The EXP MOD continuously reads from sensors, processes and stores this data for scientific studies. It also sends some of those values to the Emitter Telemetry Module (TELEM MOD) for live monitoring of the experience. These three modules are powered through the Power Module (POW MOD).
The emitter then sends experimental data to the receiver through RF transmission. The receiver is wired to the Human-Computer Interface (HCI) for display of the date.

The following datasheet will provide technical details for the POW MOD, the EXP MOD, the TELEM MOD and the HCI.

The three on-board modules have been designed to be stackable (it forms a 3.5x6cm cylinder), to minimize space occupation in such narrow spaces.

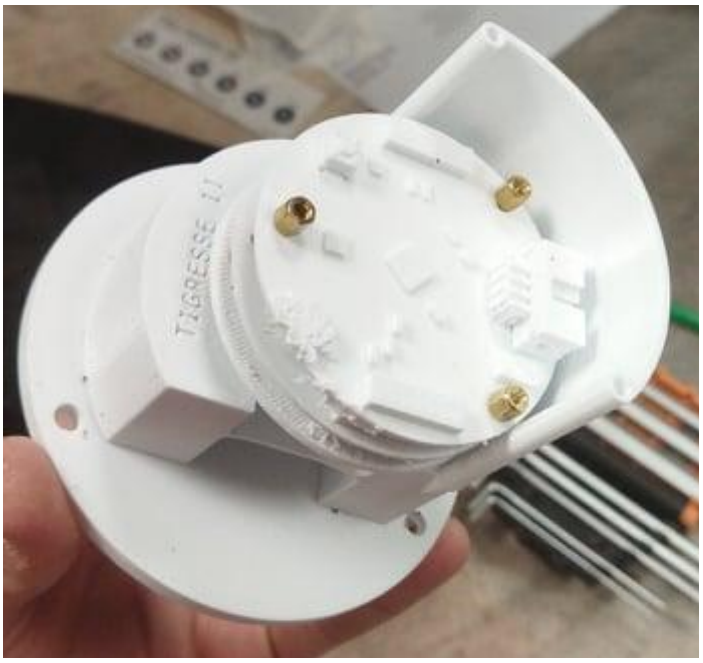


Figure 2 : 3D model of the stacked modules

Pin headers have been placed to transport signals and power lines to all three boards. Figure 3 shows the pin-out of those shared lines.

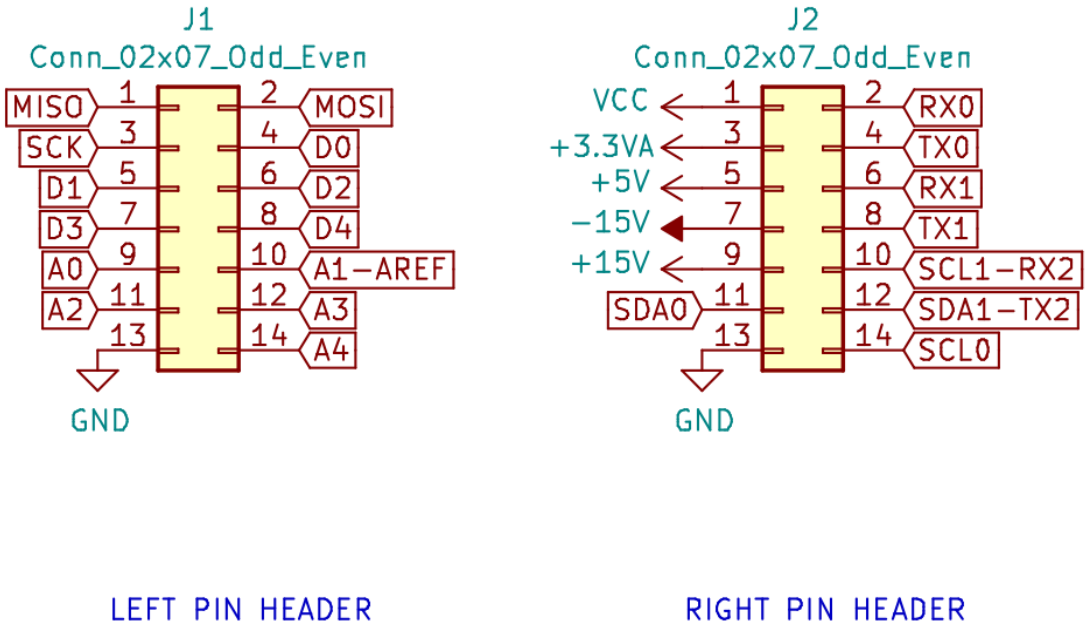


Figure 3 : Pin-out of the connectors

The UART0 port (of the EXP MOD) is designed to be wired to the SEQ MOD for isolated transmission (with an optocoupler) of the flight statuses. Another plug has been added on the POW MOD to facilitate wiring and/or soldering.

The UART1 port (of the EXP MOD) is wired to the TELEM MOD (Emitter) for transmission of flight statuses, experience data and timestamps.

All the power pins (VCC, +3.3V, 5V, +/-15V, GND) can be used for sensor wiring, and some of them are used by certain boards. GND is the common ground between the Power, Experience and Telemetry (Emitter). It is not recommended to share this ground with the Sequencer, in regards to the mini/experimental rocket context.

The other pins are mainly used by the EXP MOD for sensor wiring.

Of course, in order to realize this spatial optimization, all three boards need to have the same mechanical dimensions. Figure 4 shows the position of mounting holes and board dimensions.

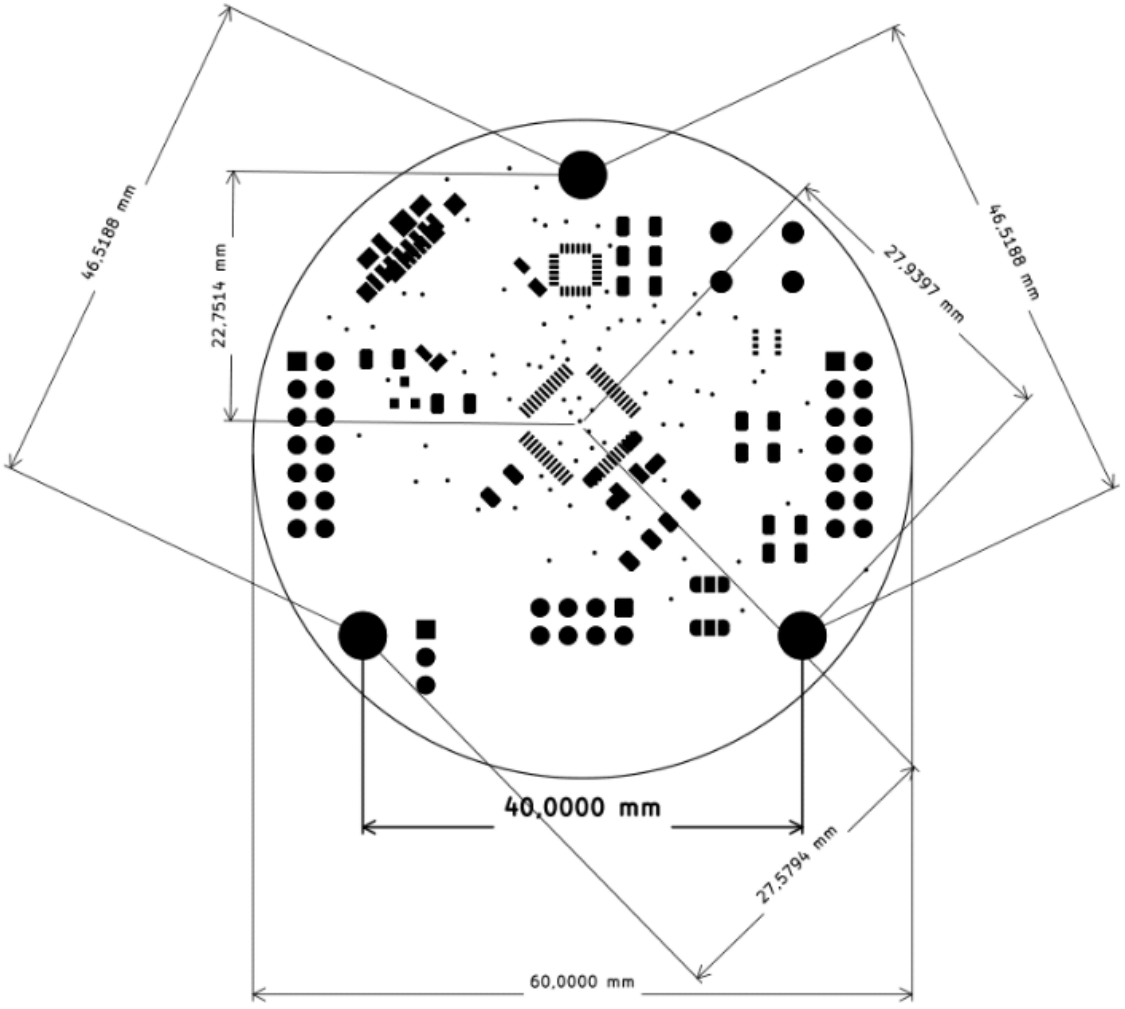


Figure 4 : Shared mechanical dimensions of the boards

2 Power Module

2.1 Introduction

The Power Module has to power the TELEM MOD and the EXP MOD, and hypothetical operational amplifiers that could be used in the experiment.
It also provides a more easy way to wire the isolated UART transmission from the Sequencer.

2.2 Power Signals

VCC : Main power source (often a 9V battery).
+5V : Regulated VCC power income. Can theoretically provide 1.5A (given that the linear regulator is cooled).
+3.3V : Regulated 5V power income. Can theoretically provide 800mA (given that the linear regulator is cooled).
+/-15V : Power incomes for the operational amplifiers. Voltage inverter are not provided in this version.

2.3 Schematics

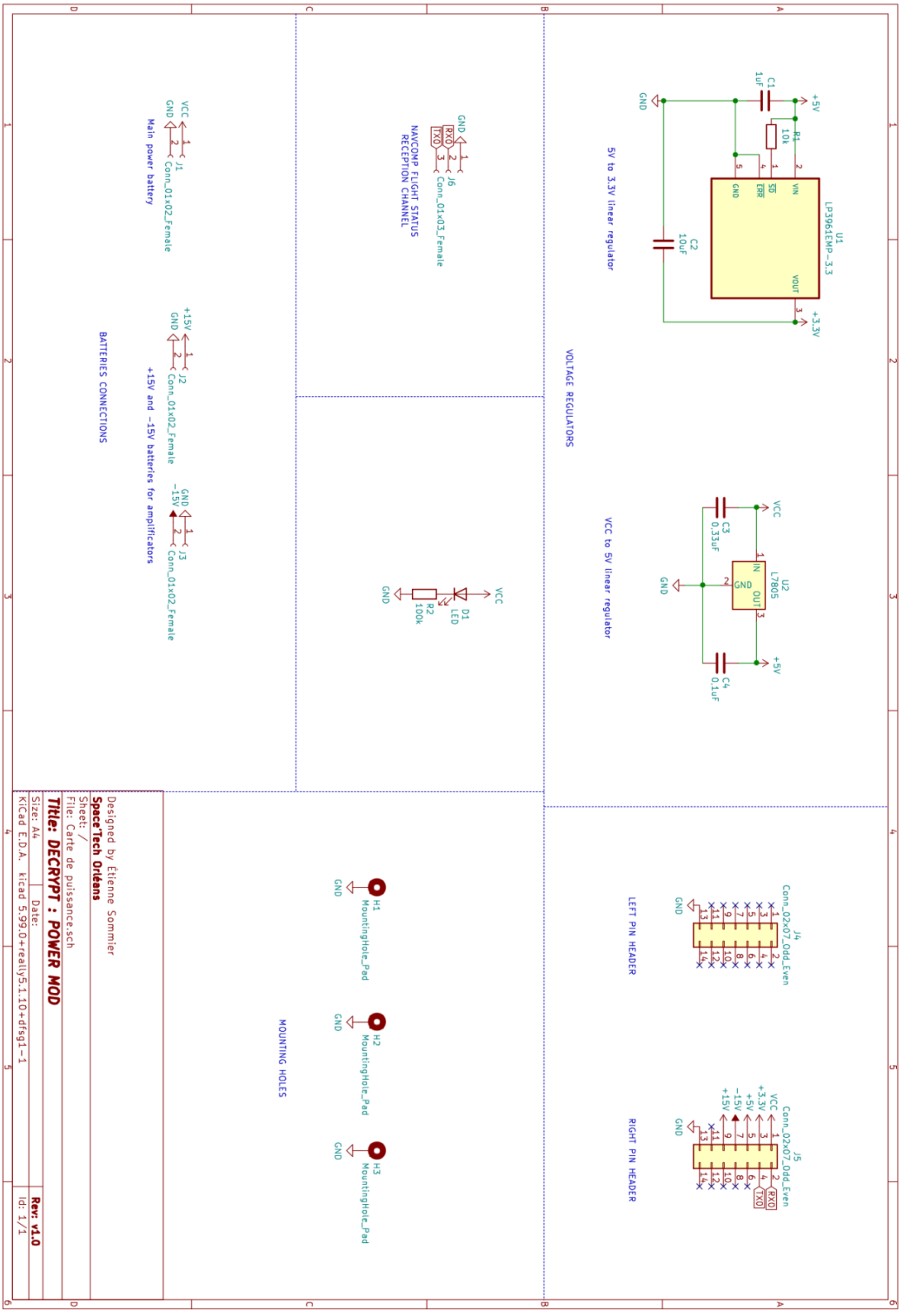


Figure 5 : Power Module full schematics

2.4 Layout

2.4.1 Front Layer

The front layer is just a common ground plane.

2.4.2 Back Layer

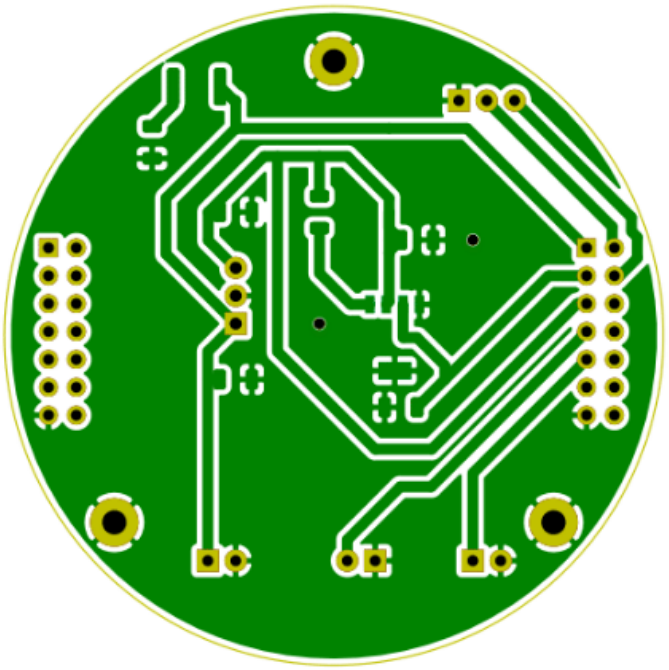


Figure 6 : Back layer of the POW MOD

3 Experience Module

3.1 Introduction and Features

The DECRYPT : Experience Module, often referred to as EXP MOD, is an embedded system for mini and experimental rockets, that aims to be as modular as possible, to be reusable and to adapt to every need.
Its goal is to acquire environmental data, process and store it for post-flight studies. It can even transmit this data to thel telemetry system. This board is a powerful ally in the experimental rocket context, and can be adapted to a large panel of projects.
The 48MHz ATSAMD21 microcontroller mounted on the board is a powerful and versatile chip which can quickly acquire data from multiple sensors via multiples communication protocols, process and filter those data, and then store them on the on-board micro SD card (or any other memory correctly wired to the board).
The module's global consumption of power is under 166 mA (all components can be set to sleep modes), its size makes it easy to integrate on most mini/experimental rockets.

Programming can be done through the micro-USB-B port and the arduino IDE, with the help of the ATSAMD21 libraries developed by Arduino and Adafruit.
The module can be powered either through USB for development or through an external power source (such as the Power Module).

The major improvements since the last version (proof of concept) are an optimized layout, price and consumption by using dedicated components instead of a development board, more connections available, embedded sensors, and improved mechanical integration.

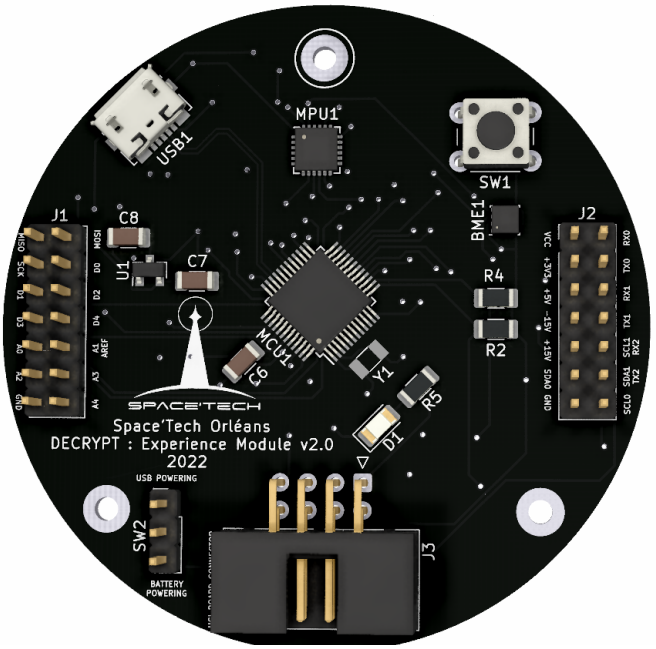


Figure 7 : Display of the EXP MOD

3.2 Signal Description List

3.2.1 Power Signals

VCC : External power source
+3.3VA : Regulated external power source
+3.3V : Either the regulated external power source or regulated USB power
+5V : Regulated external power source
+/-15V : External power sources for amplifiers
GND : General ground

3.2.2 General Purpose Inputs Outputs (GPIOs)

[D0:D4] : Digital pins of the microcontroller. Should be used for sensor monitoring/control.
A0, A2, A3, A4 : Analog pins of the microcontroller. Should be used for sensor monitoring/control
A1-AREF : Either analog input/output from the microcontroller, or voltage reference for the ADCs inside the microcontroller

3.2.3 Communications

The EXP MOD offers three communications protocols (I²C, SPI and UART), using the SERCOMs ports (i.e. SERial COMmunications) of the ATSAMD21 controller. Each SERCOM port consists of 4 pads (one pin per pad), shares the same clocks and can be configured as SPI, UART, or I²C protocol. See table X for repartition and use of the SERCOMs ports in the EXP MOD context.

SERCOM PORT	BUS	PAD0	PAD1	PAD2	PAD3
SERCOM0	I²C1 or UART2	PA08	PA09	PA10	PA11
SERCOM1	UART0	PA16	PA17	PA18	PA19
SERCOM2	SPI0	PA12	PA13	PA14	PA15
SERCOM3	I²C0	PA22	PA23	PA20	PA21
SERCOM4	UART1	PB08	PB09	PB10	PB11
SERCOM5	Unusable	PB02	PB03	PB22	PB23

Table 1 : SERCOMs ports wiring on the EXP MOD board

For pads repartition per protocol, see table X.

PROTOCOL	PAD0	PAD1	PAD2	PAD3
UART	X	X	TX	RX
I²C	SDA	SCL	X	X
SPI	MISO	MOSI	SCK	X

Table 2 : Pads repartition per protocol on the ATSAMD21G18-A

The X does not refer to unusable pins but to other functions either useless or serving other purposes in the EXP MOD context.

SPI PROTOCOL

MOSI : Master Out Slave In, often referred to as SDO (Serial Data Out). Datas coming from the master/controller (ie the microcontroller) to the slave/peripheral (ie the sensors).

MISO : Master In Slave Out, often referred to as SDI (Serial Data In). Datas coming from the slave/peripheral to the master/controller.

SCK : Serial Clock. Control the baud rate of the data lines.

CSn : Chip Select Pins, almost always active when voltage is at low voltage level. One reserved Chip Select pin is used for the micro-SD card (PA07 of the microcontroller), but Analog and Digital pins (see the GPIOs section above) can be used as Chip Select pins.

UART PROTOCOL

RX0/TX0 : Respectively receiver and transmitter lines of the UART0 port.
RX1/TX1 : Respectively receiver and transmitter lines of the UART1 port.

I²C PROTOCOL

SDA0 : Serial Data line, shared by all the peripherals on the bus.
SCL0 : Serial Clock, leads the baud rate for data transmission.

One additional bus can be configured as either I²C or UART protocol. The user simply needs to wire the right tracks using the JP1 and JP2 solder pads (both tracks of the chosen protocol need to be wired in order for it to work). Visual indications are displayed on the board.

For the clocks of the SERCOMs port, see table X, inspired by the ATSAMD21 datasheet.

PROTOCOL	BAUD RATE	BAUD REGISTER VALUE
SPI and I²C	$f_{BAUD} = \frac{f_{ref}}{16}\left(1 - \frac{BAUD}{65536}\right)$	$BAUD = 65536\left(1 - 16\frac{f_{BAUD}}{f_{ref}}\right)$
UART	$f_{BAUD} = \frac{f_{ref}}{2(BAUD+1)}$	$BAUD = \frac{f_{ref}}{2f_{BAUD}} - 1$

Table 3 : SERCOMs ports clock baud rate and register values on the ATSAMD21G18-A

3.2.4 Human-Computer Interface

Note that the HCI IDC8 connector on the EXP MOD does not refer to the HCI Module, but to a simple board containing LEDs and push buttons to initialize and use the EXP MOD.

[LED0:LED2] : Outputs from the microcontroller
[PB0:PB1] : Inputs from the microcontroller.
These ports should be used to interface an HCI composed of LEDs and buttons, to initiate, test, and engage flight mode.

3.3 Recommended Programming

The microcontroller can be programmed with the Arduino IDE. Simply add this URL into the Additional Board Manager URLs text box (under File > Preferences) :
https://adafruit.github.io/arduino-board-index/package_adafruit_index.json
Proceed to Boards Manager (under Tools > Board), search for Adafruit SAMD Boards and install the package. Then select the Adafruit Feather M0 board (under Tools > Board > Adafruit SAMD Boards), and star coding.

You can use the following online tutorial to program the communication ports :
https://learn.adafruit.com/using-atsamd21-sercom-to-add-more-spi-i2c-serial-ports/overview

You can use the MPU-6050, BME280 and SD libraries as well for all the peripherals (maybe need some adjustments to the M0/ATSAMD21 architecture).

A more detailed tutorial will be provided on a future version of this datasheet.

The algorithm on figure 8 is a simple and efficient method for acquisition, processing and transmission of data to the TELEM MOD.

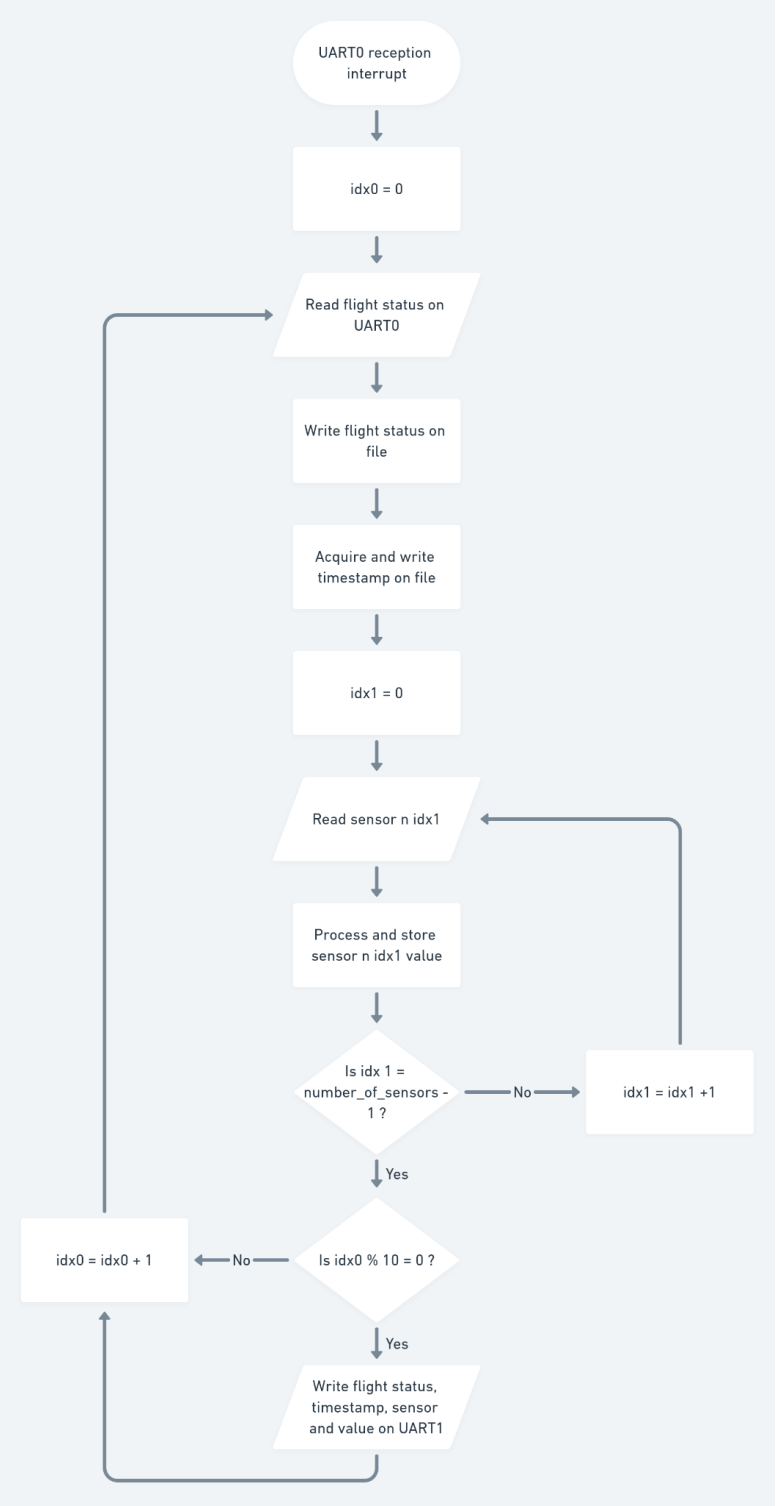


Figure 8 : Recommended algorithm for EXP MOD implementation

3.4 Electrical Characteristics

3.4.1 Absolute Maximum Ratings

SYMBOL	DESCRIPTION	MIN.	TYP.	MAX.	Units
3.3VA[1]	3.3V external source	2.375	3.3	3.48	V
3.3VP[1]	3.3V regulated USB Power	3.17	3.3	3.43	V
VIL	Voltage Input Low-level, GPIO pin			0.3*3.3V	V
VIH	Voltage Input High-level, GPIO pin	0.55*3.3V			V
VOL	Voltage Output Low-level, GPIO pin		0.1*3.3V	0.2*3.3V	V
VOH	Voltage Output High-level, GPIO pin	0.8*3.3V	0.9*3.3V		V
IOL	Current Output Low-level, GPIO pin		10		mA
IOH	Current Output High-level, GPIO pin		7		mA

Table 4 : Absolute Maximum Ratings of the EXP MOD

[1] : The voltages coming out/in of the pins of the microcontroller depend on the 3.3V voltage, which can be either 3.3VA or 3.3VP.

3.4.2 Power Consumption at 25°C

Components	COMMENTS	MIN.	TYP.	MAX.	Units
MPU-6050	“Continuous” use		3.9		mA
MPU-6050	Low-Power mode		140		µA
BME280	“Continuous” use		1.504		mA
BME280	Low-Power mode		0.3		µA
Micro-SD card	SPI mode, R/W		30	100	mA
Micro-SD card	SPI mode, stand-by		0.4		mA
ATSAMD21G 18A	Normal use, estimated value		130		mA

ATSAMD21G 18A	IDLE0 (Low-Power mode)	1.89	2.04	2.20	mA
Total	Normal use		165.404		mA
Total	Low-Power and/or stand-by		2.5803		mA

Table 5 : Power Consumption at 25°C of the EXP MOD

3.5 Schematics

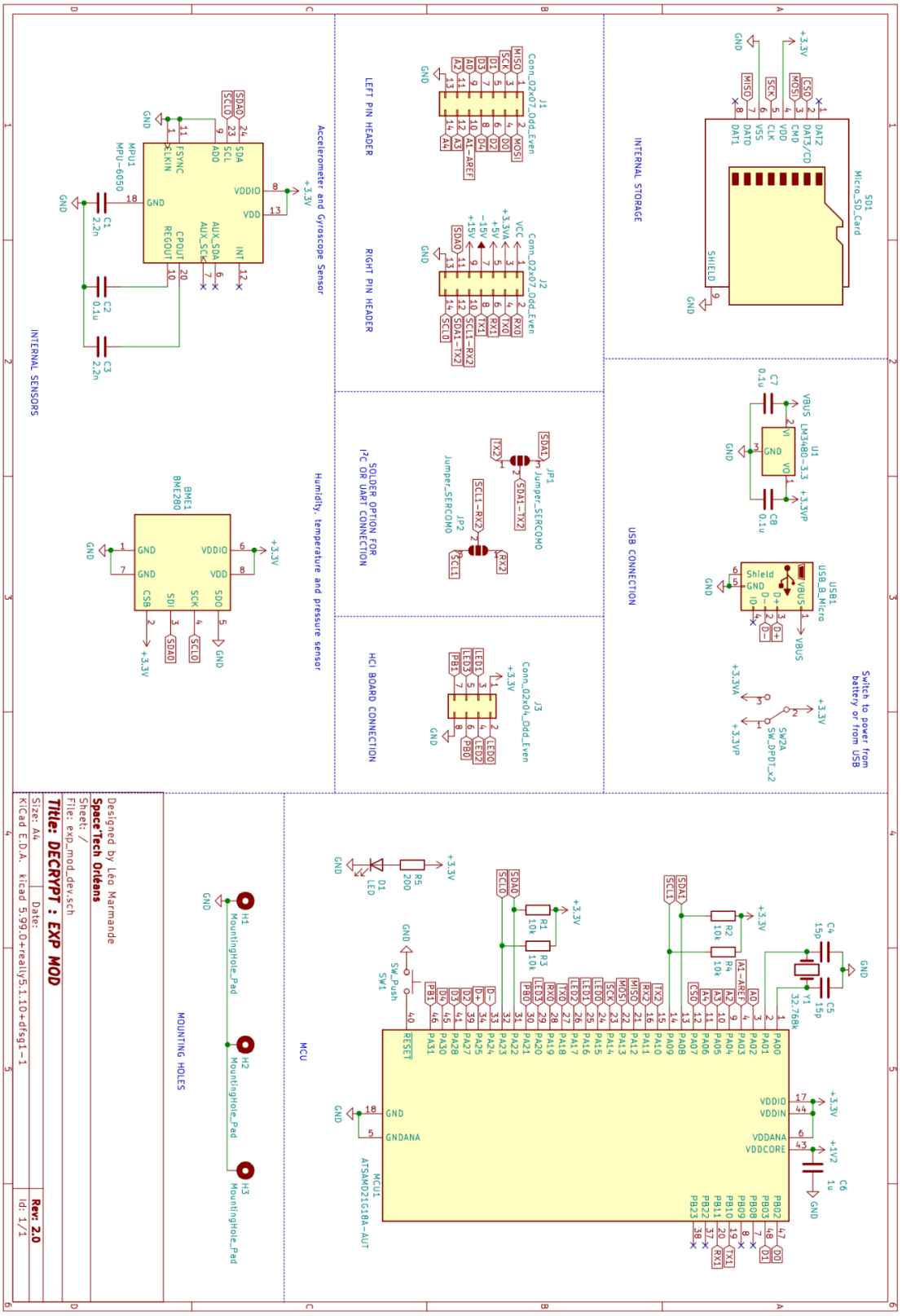


Figure 9 : Full schematics of the EXP MOD

The EXP MOD is centered around the ATSAMD21G18A microcontroller, clocked at 42MHz with a 32.768kHz crystal oscillator. This controller is wired to a micro-USB-B port for development, to a micro-SD port through SPI protocol for data storage, and to two sensors (through I²C protocol) : the MPU-6050 from InvenSense, which is a low-power and precise 6-axis accelerometer and gyroscope, and the Bosch BME280, a precise humidity, pressure and temperature sensor. The quartz has been implemented considering a 5pF parasitic capacitance in the circuit.

As secondary features, the EXP MOD contains two 2x07 pin headers, one IDC8 port (2.54mm standard), and two solder pads for configuration of a SERCOM port.

For power income, the EXP MOD offers two possibilities : powering from the USB port through a 3.3V linear regulator, or from an external power source (for example the POW MOD). The user can easily switch between those two power sources with either a 3x1 pin header with a jumper or a switch (2.54mm standard) soldered on the SW2 pads. This is represented by a simple switch on the schematics sheet.

3.6 Layout

3.6.1 Front Layer

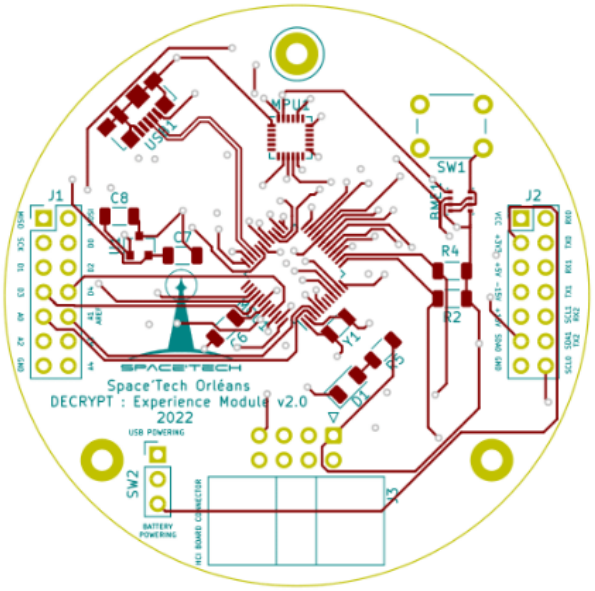


Figure 10 : Front layer tracks, pads and silkscreen of the EXP MOD

3.6.2 1st Interior Layout

The first interior layout is only composed of a common ground plane.

3.6.3 2nd Interior Layer

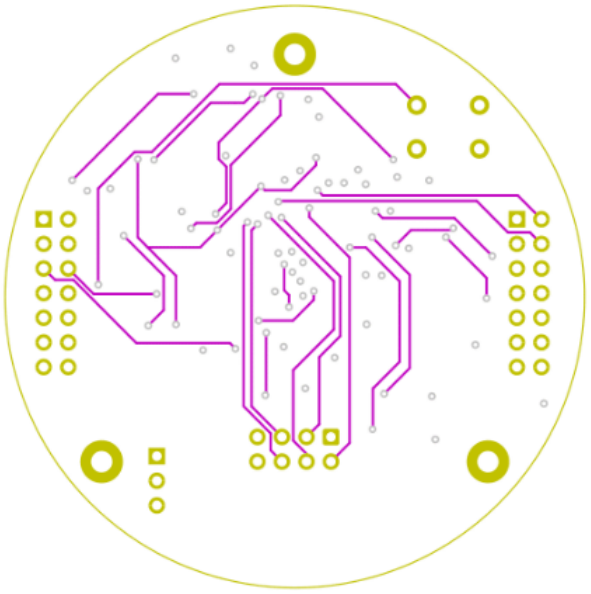


Figure 11 : 2nd interior layer tracks, pads and silkscreen of the EXP MOD

3.6.4 Back Layer

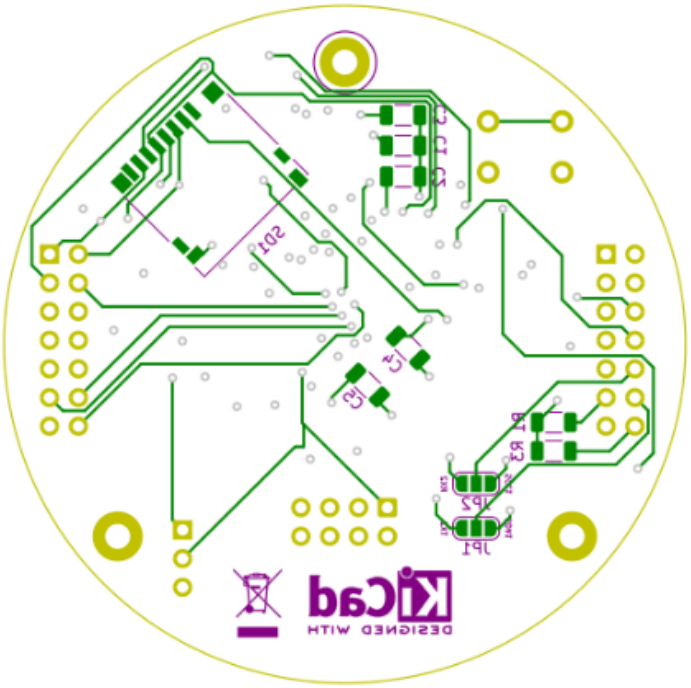


Figure 12 : Back layer tracks, pads and silkscreen of the EXP MOD

The 1st interior layer is a ground plane. All pins are at 2.54 mm from one another. Capacitors, resistors and the LED are all 1206 (3216 Metric) packages. Vias have a 0.8mm diameter, tracks are 0.250mm large, and mounting holes are M2 standard.

Notice that one of the mounting holes is circled by a white line : all three stackable modules (POW, EXP and TELEM MODs) have the same circle on the silkscreen.

4 Telemetry Module

4.1 Telemetry Introduction

In this project, telemetry is the communication between a rocket and the launch center. This is a misuse of language, indeed, telemetry is a distance measurement technique. However we respect the terms used by the organization which manages the launch of the rockets.

4.1.1 LoRa Communications

To realize this communication, LoRa technology (for Long Range) has been chosen. These radio communications are made with a low data rate, over a great distance. This protocol respects the specifications given by the association, which requires communications with a minimum distance of 7km. This technology is very common in the world of the Internet of Things (IOT).

4.1.1.1 How LoRa communications work

LoRa is a Chirp Spread Spectrum modulation. A bandwidth greater than ideal is used, which allows operation with a weak or high noise signal. The spread spectrum consists in repeating several times the transmitted message at different frequencies.

The LoRa physical frame is composed of 4 parts. A preamble, a header, the data and eventually a rate code.

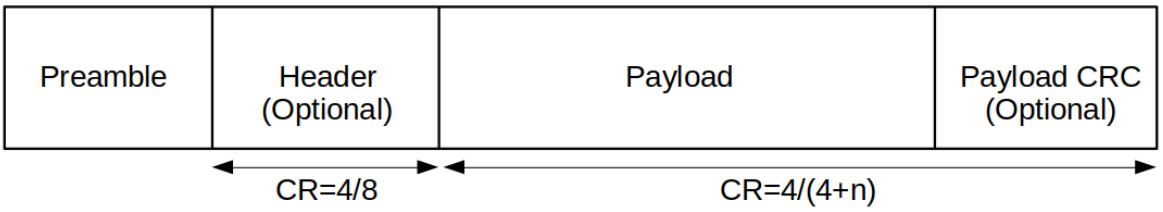


Figure XX: Format of a LoRa frame

4.1.1.2 Communication Characteristics

Many characteristics allow to detail the communication realized. It is possible to modify some of them in order to increase the distance but decrease the amount of data sent, and the other way around.

Frequency	868 MHz
Bandwidth	125 kHz
Output power	13 dBm in transmission
Spreading factor	12
Sensitivity	-111 dBm
Nominal Rb	293 bps (PER = 1%)
Theoretical distance	46 km
Experimental distance	5615 km

Table XX: Characteristics of communication

Another important point of this communication is the antenna, which must be as powerful as possible while being small enough to fit in a rocket without taking much space. The chosen antenna is easy to integrate in the rocket.

Figure XX: Photo of the used antenna

Polarization	Linear
Radiation pattern	Omnidirectional
Impedance	50 Ohm
Gain	0.3 dBi
Total efficiency	>55%

Table XX: Characteristics of the antenna at 868MHz

4.1.2 Proof of Concept

The proof of concept has been made using two TTGO modules. These modules include LCD Screen for data display.

Figure XX: Photo of LoRa module TTGO

To establish connection between the modules, the Sandeep Mistry library has been used.
Two programs have been made.
The first program uses a counter (which starts at 0 and increments by one every 5 seconds) and sends the indentation to the receiver.

```ino
//Libraries for LoRa
#include <SPI.h>
#include <LoRa.h>


//define the pins used by the LoRa transceiver module
#define SCK 5
#define MISO 19
#define MOSI 27
#define SS 18
#define RST 14
#define DIO0 26

#define BAND 866E6 //for French

//packet counter
int counter = 0;

void setup() {

  //SPI LoRa pins
  SPI.begin(SCK, MISO, MOSI, SS);
  //setup LoRa transceiver module
  LoRa.setPins(SS, RST, DIO0);
}

void loop() {

  //Send LoRa packet to receiver
  LoRa.beginPacket();
  LoRa.print("hello ");
  LoRa.print(counter);
  LoRa.endPacket();

  counter++;

  delay(5000);
}
```

Figure XX: Data sending program

The second program receives the data sent by the first.

```
//Libraries for LoRa
#include <SPI.h>
#include <LoRa.h>


//define the pins used by the LoRa transceiver module
#define SCK 5
#define MISO 19
#define MOSI 27
#define SS 18
#define RST 14
#define DIO0 26

#define BAND 868E6

String LoRaData;

void setup() {

  //SPI LoRa pins
  SPI.begin(SCK, MISO, MOSI, SS);
  //setup LoRa transceiver module
  LoRa.setPins(SS, RST, DIO0);
}

void loop() {
    //read packet
    while (LoRa.available()) {
      LoRaData = LoRa.readString();
    }
}
```

Figure XX: Data reception program

These two boards can be used in a rocket, however, the following board is an optimized and dedicated way to send data to the ground station.

4.2 Hardware

As previously described, the LoRa communication is composed of two parts, the emitter and the receiver, which are separated into two distinct boards. A first one in the rocket which can be integrated and stacked with the other modules (i.e. with the same mechanical characteristics). The second board, present in the Human-Computer Interface must be as small as possible but has no specific shape.

4.2.1 Boards Overview

The actual prototype has been realized on a perforated board, however it was not possible to connect an antenna to these boards. PCBs were then made chemically, but these are not functional because the pins are too small and the alignment was wrong. For these boards to be able to be put into a rocket, they have to be strong and the right size. That's why PCBs made by professionals will soon be ordered by Space'Tech, using the definitive schematics.

4.2.1.1 Components Characteristics

The most important component to realize the communication are the two LoRa modules, there must be one in each board. The choice is the RFM95W-868S2 module, which allows integration on a prototyping board. It sends its data at a frequency of 868MHz, which is a free band and which is authorized to use on the military launch base.

To control the sending of the data, a microcontroller is added on the emitter. For reasons of supply difficulty, the ATmega328P has been chosen, but is still perfect for this application. Its job is to receive data from the EXP MOD, sample them and send them to the LoRa module.

The LED connected between the input voltage and the ground allows a luminous indication of the board alimentation. In addition to that, a reset button is present on both boards to reset them.

4.2.1.2 The connections between components

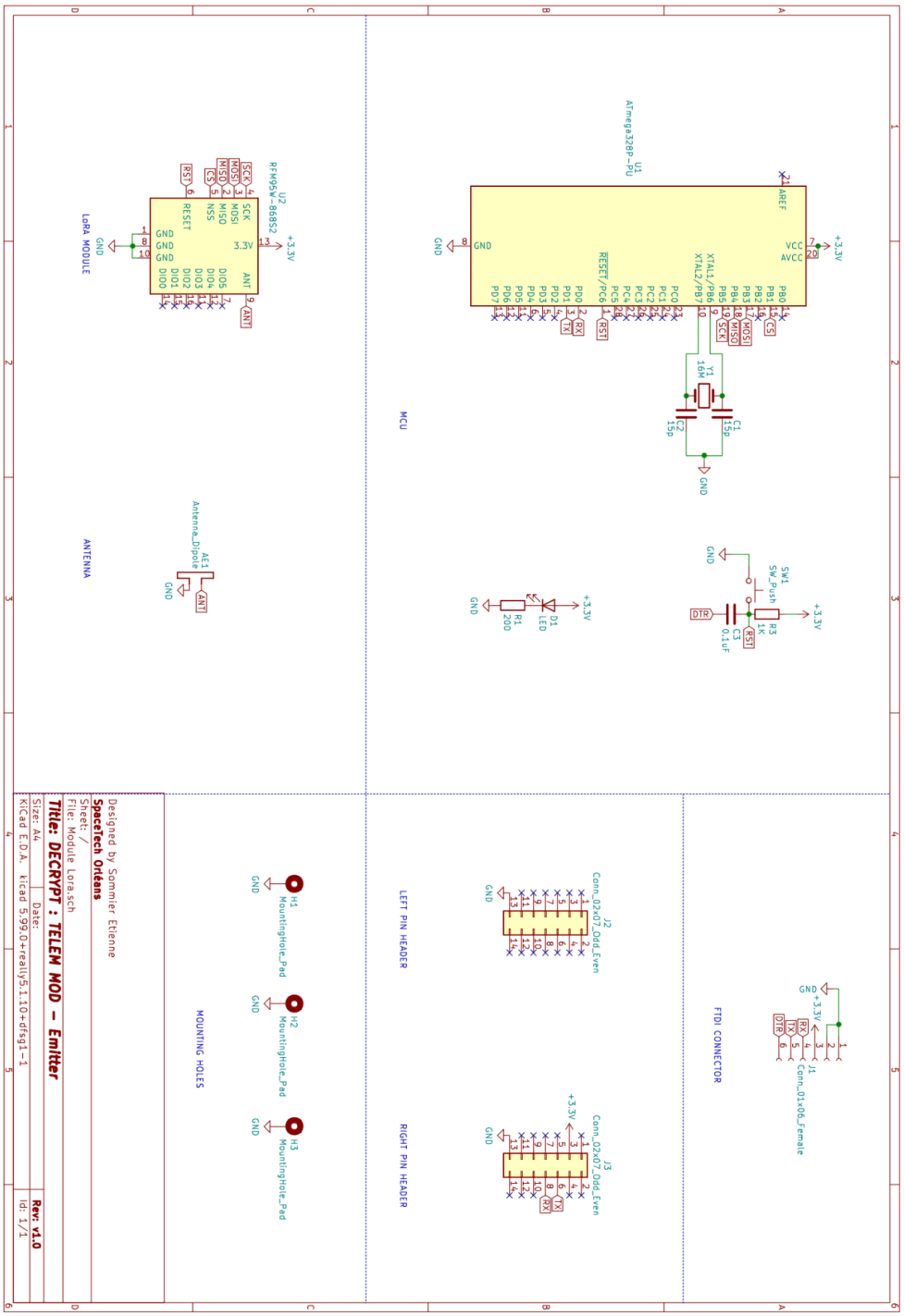


Figure XX: Full schematic of the sending module

The microcontroller needs a timer to operate, so a quartz is connected between pins 9 and 10. Two capacitors are added for the proper functioning of it.
The LoRa module is directly connected to an antenna which aims to amplify the received or emitted signal.

The connection between the LoRa module and the microcontroller is the SPI protocol. There are therefore 4 connections to make the communication between the two, in addition to the ground and power supply :

Pin of ATmega328P	Pin of RFM95W-868V2
PB5	SCK
PB3	MOSI
PB4	MISO
PB1	NSS
Vcc	3V3
GND	GND

Table XX: Connection panel

Finally, to be able to program the microcontroller, 6 pin headers allow connection between the board and a computer in USB through a FTDI Interface.

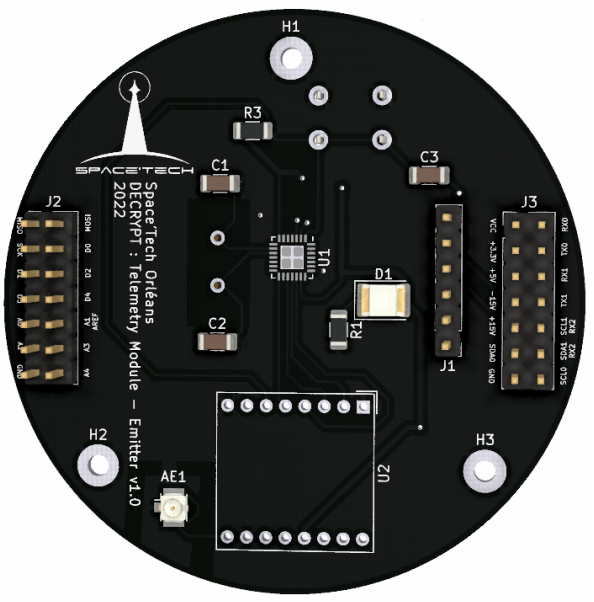


Figure XX: Representation of the realized PCB

The connections of the receiving card are much simpler, the LoRa module is directly connected to pin headers that will allow reception of the data directly on the Raspberry used for the Human-Computer Interface.

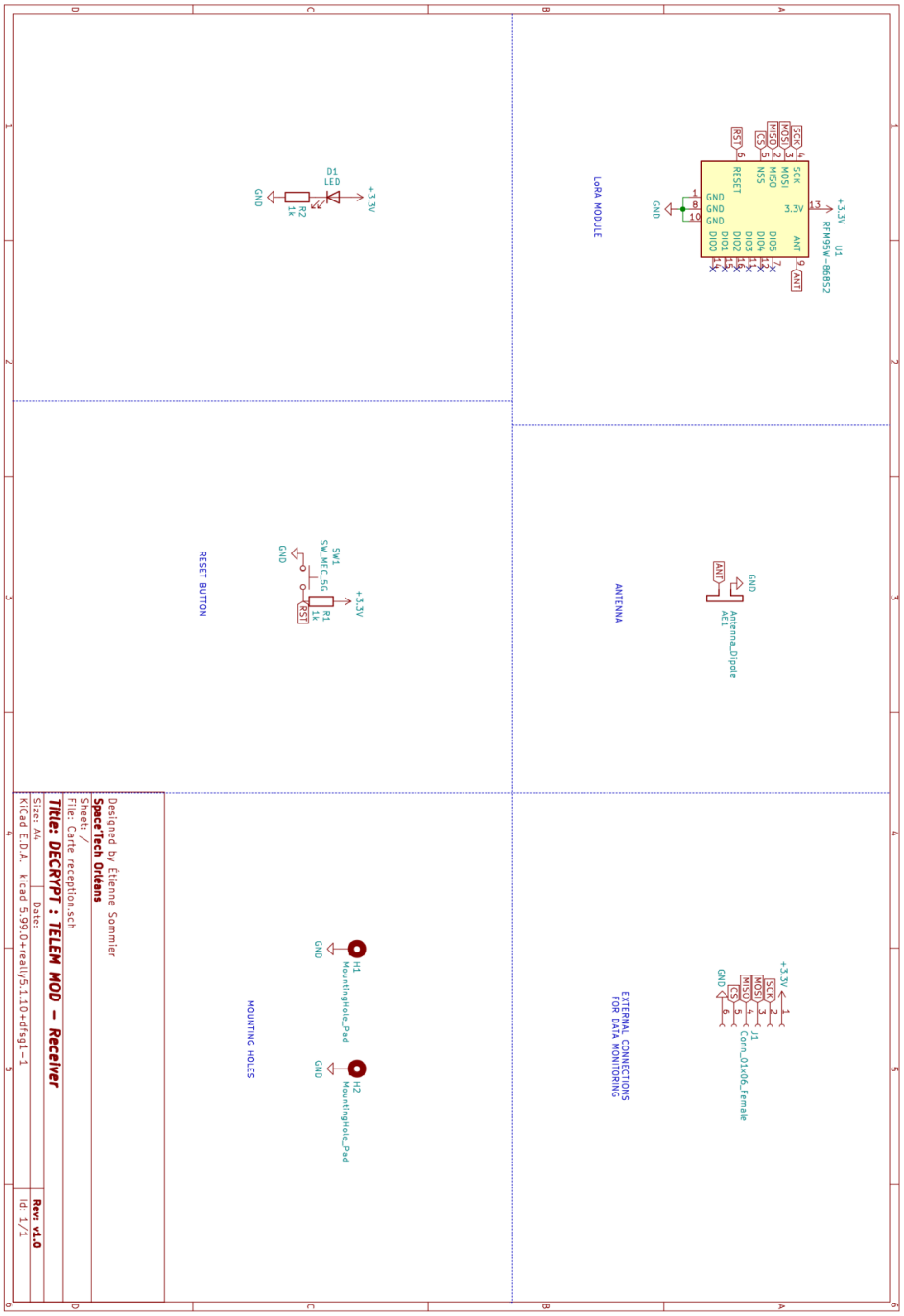


Figure XX: Full schematic of the reception module

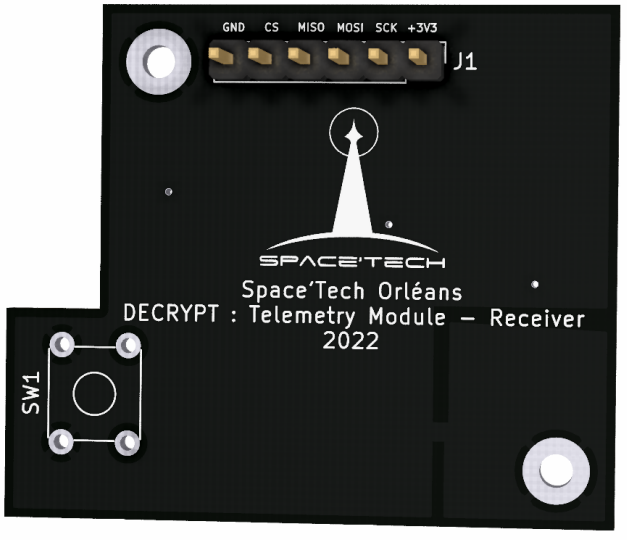


Figure XX: Display of the Receiver

4.2.2 Boards Characteristics

4.2.2.1 Power consumption at 25°C

4.2.2.1.1 Emitter module

Components	COMMENTS	MIN.	TYP.	MAX	Units
ATmega328P	“Continuous” use		1.7	2.5	mA
ATmega328P	Low-power mode		0.9		µA
RFM95W	Transmit mode with impedance matching		120		mA
RFM95W	Standby mode		1.6	1.8	mA
Total	**Normal use**		**121.7**		**mA**
Total	**Standby mode**		**1.6009**		**mA**

Table XX: Consumption of the sending module

4.2.2.1.2 Receiver module

Components	COMMENTS	MIN.	TYP.	MAX.	Units
RFM95W	Receive mode		12.1		mA
RFM95W	Standby mode		1.6		mA
Total	**Normal use**		**12.1**		**mA**
Total	**Standby mode**		**1.6**		**MA**

Table XX: Consumption of the receive module

4.2.2.2 Layouts

4.2.2.2.1 Emitter Module

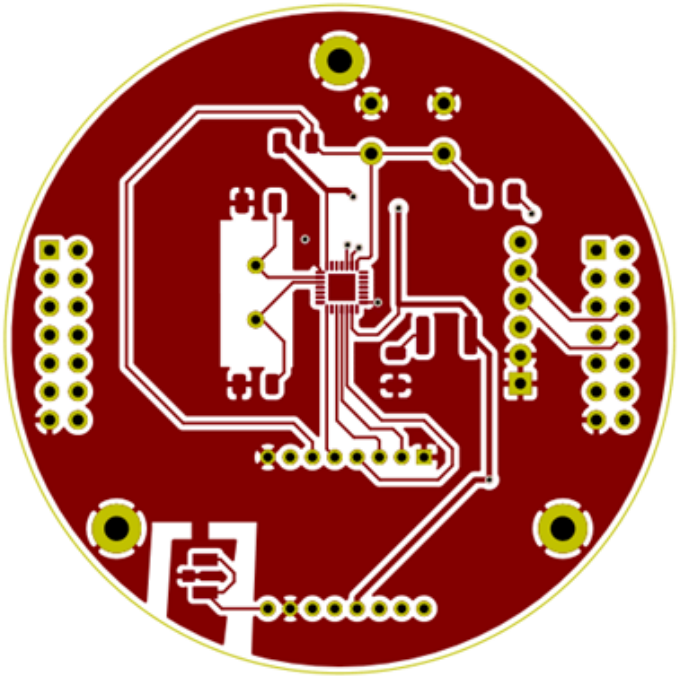


Figure XX : Front layer tracks

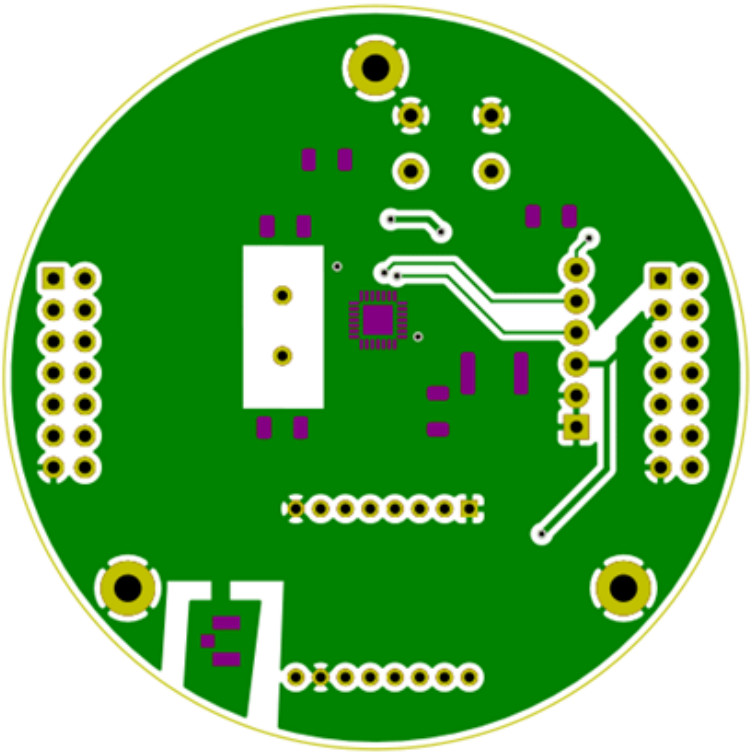


Figure XX : Back layer tracks

4.2.2.2.2 Receiver Module

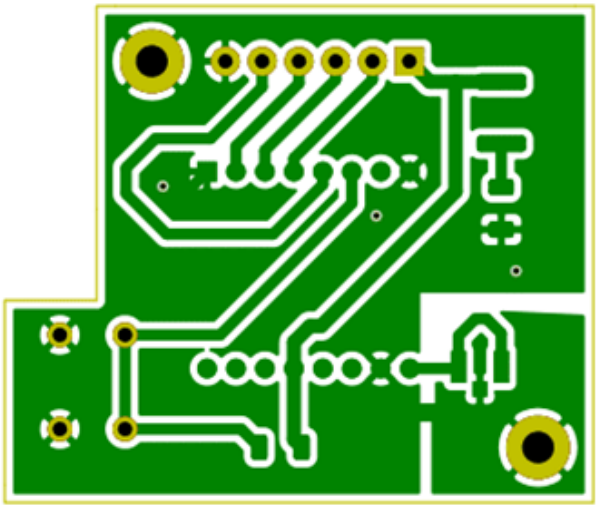


Figure XX: Front layer tracks

5 Human-Computer Interface (HCI) Module

5.1 Presentation

The last part of the DECRYPT project consists in the realization of a man-machine interface in order to establish a visual link between the user and the experimental rocket, so they are able to visualize at regular intervals the data relating to the embedded experiment in the rocket, but also the flight data such as the altitude, the pressure, the acceleration or any other data likely to be requested by the user. The second objective is also to be able to transport this interface in a case, and therefore to create a hardware structure that is sufficiently powerful to create an interface and that is easy to transport.

5.2 Materials and Software Used

5.2.1 Software

In order for the man-machine interface to be modified by the user, it is necessary to write in a programming language understandable by all. For this reason, the language used will be Python. Python has the advantage of having access to a multitude of libraries for databases, e-mail and image manipulation. Moreover, it is a language that is closer to an intuitive language, unlike others such as Java or C++, which allows any user without a deep knowledge of programming to understand and modify its code. The final advantage to note for Python is that it is easily extensible to other languages, such as C++ under the QT interface.

To structure the interface, we will use Qt as a programming language (via the PyQT5 module). This choice is justified by two reasons:

- The first one is justified by the impossibility to interact with all the secondary windows of the graphical interfaces with the Tkinter library (which is the graphical interface module specific to Python) while keeping these windows in the foreground (forced to manage each window separately)

-The second one is justified by the possibility to modulate this interface in the future without modifying the associated python code. If the user wants to make a more advanced architecture in QtDesigner, he will not need to modify the code written in Python, something impossible with the Tkinter library.

In order to receive the different data sent by the rocket sensors, the use of a database is necessary. However, as the internet connection seems to be uncertain at the launch site (military base), the choice of the database must take into account the use of a local database and hosting on a web server (to avoid connection problems) to store the data. For this reason, the choice has been SQLITE, which is a database system that does not require a server. This system also allows to centralize all the data in a single file created automatically.

For the interface, we will use the DB Browser SQLITE software

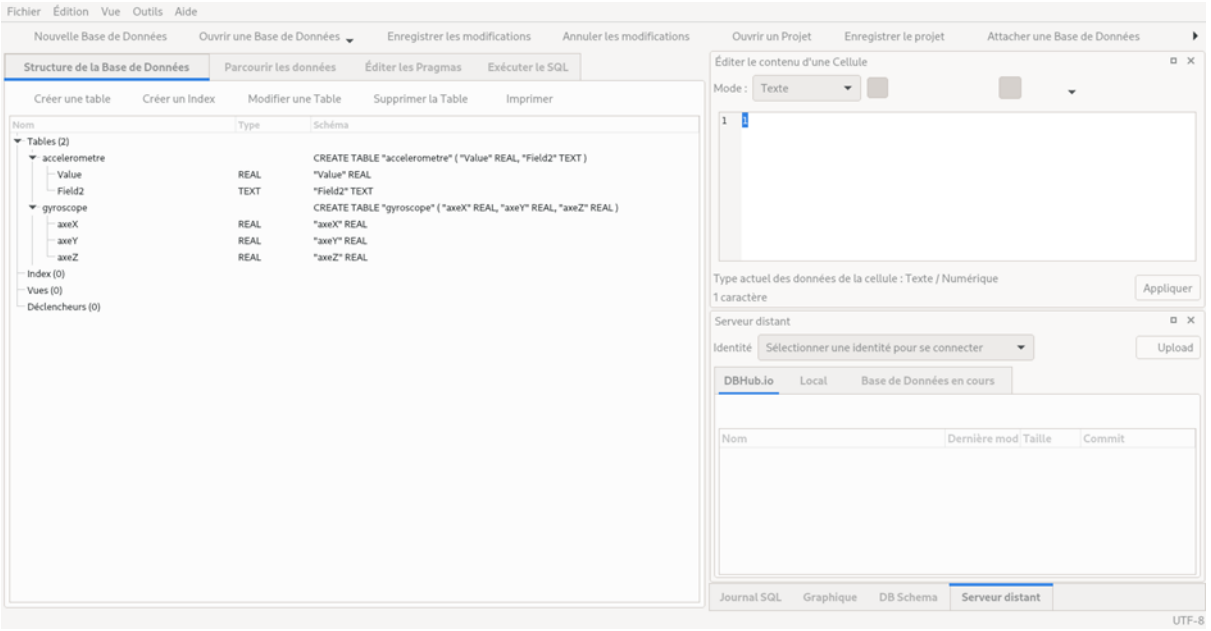


Figure X: DB Browser SQLite software interface

From now on, the interface will be based on Visual Studio Code, which is a code editor that allows the execution of various programs. This choice is justified by the fact that Visual Studio Code allows applications to be debugged directly in the editor thanks to a system of breakpoints and through a debugging console that allows problems to be resolved directly from the editor. In addition, visually, it will be easier for the user to make changes to their code according to their objectives (only a graphical window, non-dynamic graphics, for example). In addition, for the user, their code changes can be transferred to the Raspberry much more easily as the Raspberry will also have Visual Studio Code as a code editor to support the interface.

5.2.2 Hardware

The objective of the hardware part is to propose a man-machine interface that can be easily transported (as in a briefcase for example) and whose implementation would be quick. Moreover, the HCI must be usable by the whole association and therefore by its members. Thus the main element of this interface will be a Raspberry Pi 4, Model B 4GB RAM, which will act as a computer. Moreover, in terms of size for transport and price, it remains much more interesting than a classic laptop. The Raspberry will have a screen to display the interface and a keyboard and mouse to allow interaction. You will also need to remember to take a power source with you for the transportable Raspberry (a battery, for instance).

It is important to configure the Raspberry before using the interface. The SPI interface must be configured to receive data over an SPI link from the LoRA receiver module.

The steps are as follows:

- In the terminal, run the following command "sudo raspi-config" to open the raspberry configuration interface.

- Select "Interfacing Options".

- Select SPI and then "Yes".

- Reboot the Raspberry

In addition, to download different software on the raspberry, simply open a terminal and issue the following command: sudo apt install "see software name".For example, to install VSCode on Raspberry, you just have to enter :

- sudo apt install code

However, the installation of python modules are not present during the installation of VSCode, it will be necessary to use the command below :

-pip3 install "module name".

For data storage, it is necessary to use an SD card (the currently used one is 16 GB). However, an SD card of 4 or 5 GB is already more than sufficient for the human-computer interface.

5.3 Structure of the Interface

This part is devoted to explaining the various functions that allow the interface to be formatted.

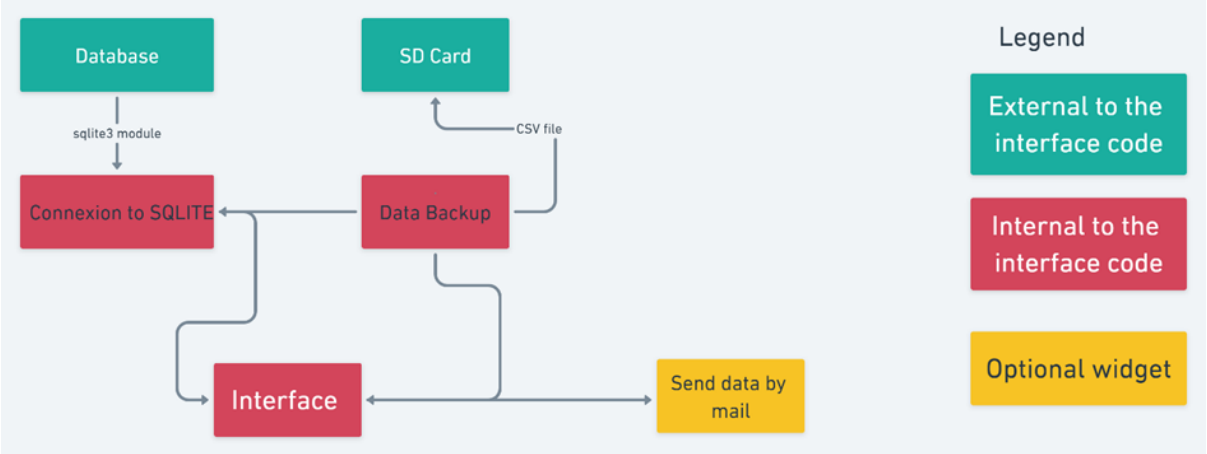


Figure X : Software algorigram

5.3.1 Connection to the database

The interface presented to the user is the assembly of several functions, all having a distinct part:

The first is that of retrieving data from the database. The objective behind the execution of the program is to be able to connect to the database and to be able to execute the contents of each table in the database.

Therefore, the library used is sqlite3, which offers optimal performance for code execution.

By executing each table, the user is able to retrieve the different values, but they are of type "connection" and thus unreadable under VS Code. To remedy this, it is sufficient to store them in different statistical tables.

At the end of the data recovery, it is essential to close the connection to the database so as not to modify it afterwards and not to slow down the execution speed of the program which uses the RAM of the Raspberry.

5.3.2 Data Recording

For saving the files, the most relevant solution was to save the data in ".csv" format by importing the csv module to allow a wider range of use and to use the file again as a text file in other applications such as Matlab for post-processing the data.

It is also possible to record dynamic graphs in mp4 format to follow the evolution of the different curves.

In agreement with the customer (Space'Tech Orléans), an option to send the file by email has been added in the code to allow a backup of the data if the SD card was defective or if the recording was not successful. This requires an internet connection but does not affect the sending of the data. To do this, a project-specific email address had to be created so that the connection between the library allowing the port connection and the recipient could take place.

The csv files are saved on the SD card after running the program.

5.3.3 Interface

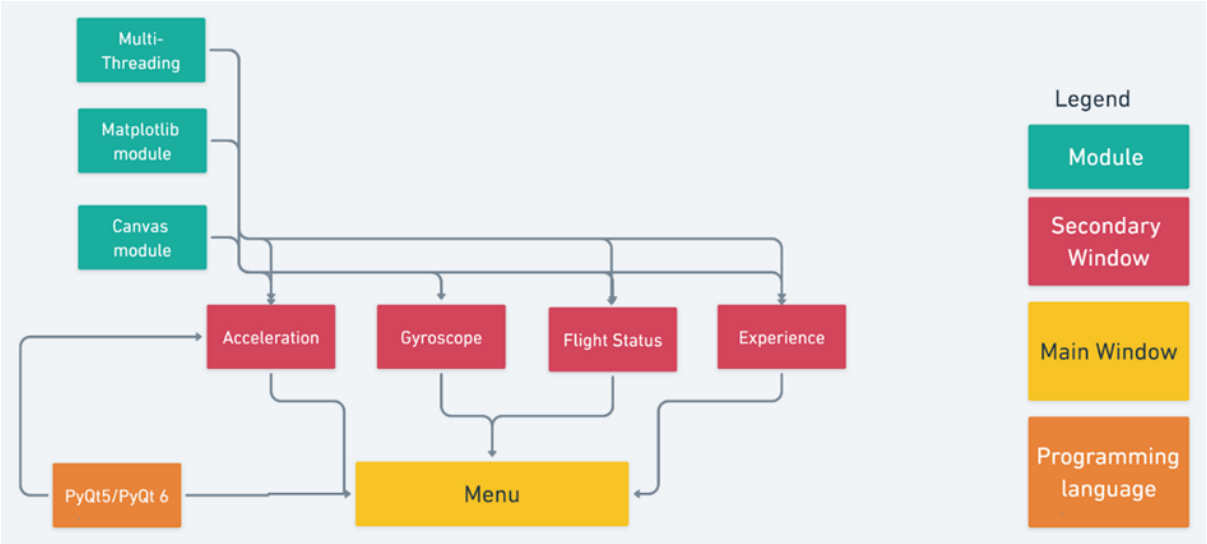


Figure X : Interface flowchart

The interface is coded with the PyQt module (version 5). This library allows the use of python modules like matplotlib or timer under the QT architecture. The use of PyQT6 is necessary to allow the use of the navigationtoolbar and canvas modules in python3 because these modules use certain packages not accessible on PyQt5. Nevertheless, the architecture of the program is well done under this last module.

The visual interface is composed of a first main window in which the user will link all the secondary windows corresponding to the different data of the sensors present in the rocket via buttons. In order to close the application, a button allowing the user to exit the interface is possible. Thus, there is nothing to prevent the user from creating as many buttons as there is data per sensor to be retrieved.

In the secondary windows are incorporated the modules matplotlib, timer, threading, figure canvas and navigationtoolbar.

The first one allows the creation of graphs in two dimensions (if needed in three dimensions, the 3D extension must be added). The threading module allows to open and close the window automatically at an interval given by the timer module, which will allow the update at a given time. The canvas and navigationtoolbar modules are two libraries which allow the display of figures but also a menu bar which allows the recording of images, the possibility of editing the curves if necessary as the user wishes (title, name of the axes, modify the scale of the axes, color of the curve, etc.)

Transferring the libraries to the Raspberry is not a problem, except for the use of PyQt6 which is not present on raspberry. To get around the problem, we had to downgrade the latest version of numpy to the version indicated in the appendix and add the package: "libatlas-base-dev". These two modifications under raspberry are an alternative to PyQt6.

To optimize the transfer of code from the computer to the raspberry, it is possible to use the VNC Viewer software which allows it to connect to the Raspberry and to be an alternative to the screen, mouse and keyboard configuration.

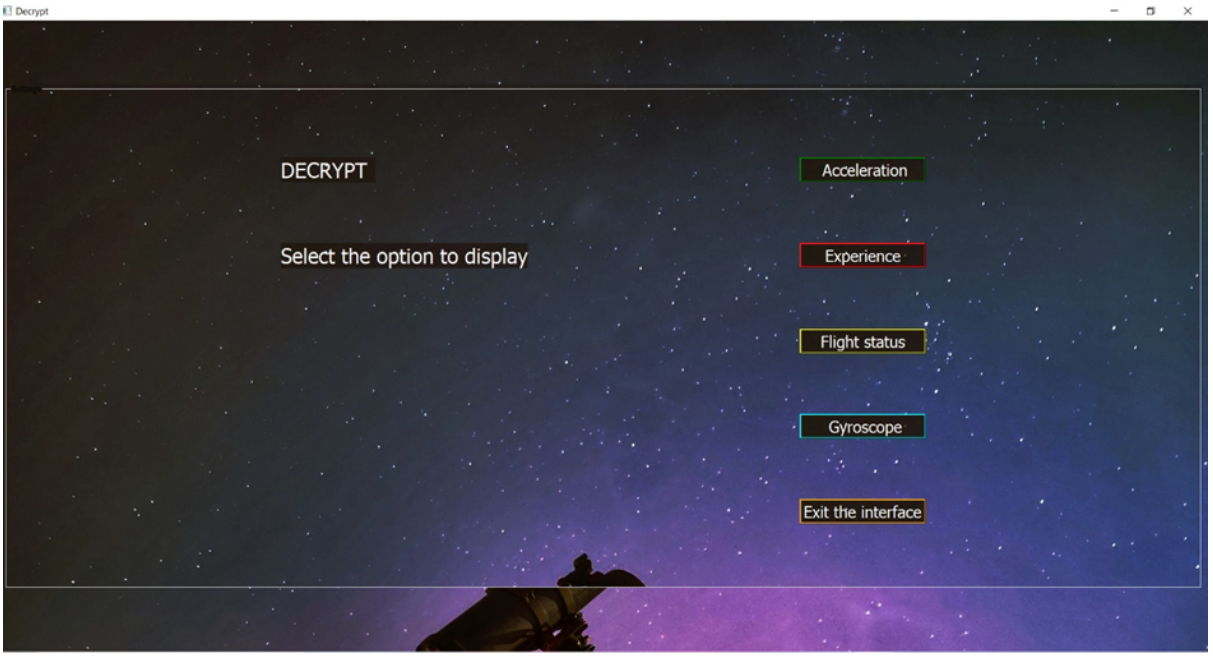


Figure X : Software main screen

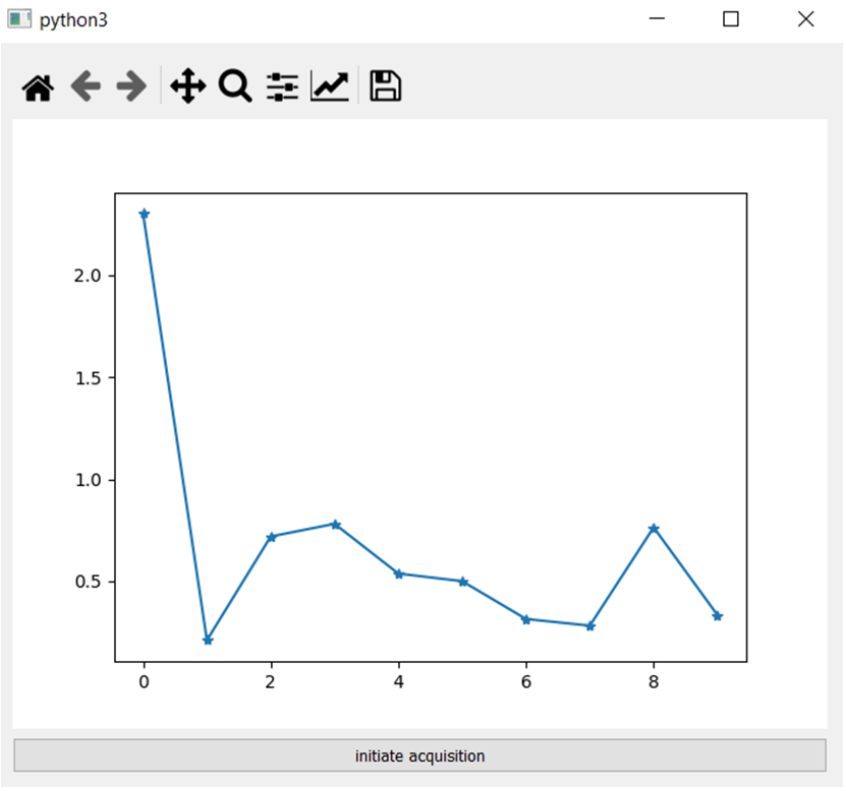


Figure X : Example of Live graphic display

Technical Related Appendices

Appendix A : Used Software and Libraries

VsCode : 1.61.2
Python : 3.9.2
DB Browser for SQlite : 3.12.99
PyQt5 : 5.15.2
PyQt6: 6.2.2 (for VsCode on the computer)
module Matplotlib : 3.15.1
module numpy:1.22.0 + libat las-base-dev → PyQt6 (for Raspberry)
module time 0.2.0
module Matplotlib-backend-qt quick : 0.0.8
module Sqlite 3.12.1

Appendix B : Database Connection Code

The following code is used to connect to retrieve values from **one** datasheet table (with each table linked to a range of values from **one** sensor)

CONNEXION_SQLITE.py > ...

```python
import sqlite3
import numpy as np


table_accelero=[]
#table_gyroscopex=[]
#table_gyroscopey=[]
#table_gyroscope=[]
#table_etatdevol=[]
#table_experiences=[]
#table_altitude=[]

try:
    connection= sqlite3.connect('decrpyt.db') #connexion to the database
    cursor = connection.cursor()#allows the execution of commands
    base_accelero = cursor.execute('SELECT * FROM Accelero')

    for row in base_accelero.fetchall(): #in each row of the database

        table_accelero.append(row[0])#adds the value
finally:
    cursor.close()
    connection.close()
```

Appendix C : Data Recording

SAVE_DATA.py > ...

```python
import csv
import os #operating system
from CONNEXION_SQLITE import table_accelero, table_gyroscope,table_etatdevol,table_experiences




def AcceleroDataRecovery(x):
    with open('Accelero data recovery.csv', 'a') as csvfile:
        writer = csv.writer(csvfile, delimiter='\t') #returns the string object in the csv file
        writer.writerows(zip(x)) #The zip() function returns a tuple iterator based on iterable objects.


def ExperienceDataRecovery(y):
    with open('Experience data recovery.csv', 'w') as csvfile:
        writer = csv.writer(csvfile, delimiter='\t')
        writer.writerows(zip(y))


def FlightDataRecovery(z):
    with open('Flight data recovery.csv', 'w') as csvfile:
        writer = csv.writer(csvfile, delimiter='\t')
        writer.writerows(zip(z))
```

Appendix D : Interface code

The following code is the structure needed to display the main window and only the data from **one** sensor. It is sufficient to multiply the code tenfold for several codes

```python
import numpy as np
import sys, os
from PyQt5.QtWidgets import *
from PyQt5.QtCore import *
from PyQt5.QtGui import *
from PyQt5 import QtCore, QtGui, QtWidgets #import different modules of python
from matplotlib.backends.backend_qt5agg import FigureCanvasQTAgg as FigureCanvas #allows you to use the matplotlib menus with pyqt5
from matplotlib.backends.backend_qt5agg import NavigationToolbar2QT as NavigationToolbar #allows you to use the navigation for matplotlibmenu
import matplotlib.pyplot as plt #displaygraph
import random
import threading
import time #allows updating
from SAVE_DATA import AcceleroDataRecovery,GyroscopeDataRecovery,ExperienceDataRecovery,FlightDataRecovery
from CONNEXION_SQLITE import table_gyroscope,table_accelero,table_etatdevol,table_experiences
import csv #allow the creation of file (.csv)
from PyQt5.QtGui import QPixmap #allow put picture
```

```python
class Menu(QMainWindow):
    def __init__(self):
        super().__init__()
        self.Main()


    def Main(self):

        self.setWindowTitle("Decrypt",)
        self.setStyleSheet("background-image: url(photo.jpg)") #the photo must be in the folder,
        #or you have to put the address to it. This is an additional option


        QBoxMain = QVBoxLayout() #allows you to create a zone in the main class to place
        #the different buttons linked to the secondary classes
        widgetMain = QWidget()
        widgetMain.setLayout(QBoxMain)
        Box = QGroupBox("Settings")
        Box.setMaximumHeight(800)

     ###################################
        Label1 = QLabel("DECRYPT ")
        Label1.setFont(QFont('ArialBlack', 20))
        Label1.setText("<font color=\"#f7f7f7\">DECRYPT</font>")
        Label1.show()
        Label1.setMaximumWidth(150)

        Label2 = QLabel("Select the option to display  ")
        Label2.setFont(QFont('ArialBlack', 20))
        Label2.show()
        Label2.setMaximumWidth(400)
        Label2.setText("<font color=\"#f7f7f7\">Select the option to display</font>")
       ###################################
    #Formatting of the different labels

        Experience = QPushButton("Experience") #allows the creation of a button
        Experience.setMaximumWidth(200) #allows you to enter the code in a delimited zone of a length to be defined
```

```
        Experience.setStyleSheet("background-color: red; color: white")
        Experience.setFont(QFont('ArialBlack', 15))
        Experience.clicked.connect(self.ExperienceFunction) #Allows you to click on the button


        Close= QPushButton("Exit the interface")
        Close.setMaximumWidth(520)
        Close.setFont(QFont('ArialBlack', 15))
        Close.setStyleSheet("background-color: orange; color : white")
        Close.clicked.connect(self.CloseInterface) #Allows to exit the interface and close the main windows




        QBoxMain.addWidget(Box)
        parametres_layout = QGridLayout(Box)#allows the elements displayed in the main window to be divided into a grid for greater rigour
        parametres_layout.addWidget(Experience, 2, 5)#specifies the location in the grid
        parametres_layout.addWidget(Close, 6, 5)
        parametres_layout.addWidget(Label1, 1, 0)
        parametres_layout.addWidget(Label2, 2, 0)

        self.setCentralWidget(widgetMain) #allows you to centralise the different buttons in the foreground of the main window

    def ExperienceFunction(self):#intermediate function calling the secondary class and linking to the previously created button
            self.connexion = Experiencedata()
            self.connexion.show()


    def CloseInterface(self):#intermediate function calling the secondary class and linking to
        #the previously created button to stop the programm
        self.interfaceclose=CloseTool()
        self.interfaceclose.show()
```

```
class Experiencedata(QWidget):
    def __init__(self):
        super().__init__()
        self.buttonpause = QPushButton('Pause the acquisition')
        self.buttonpause.clicked.connect(self.startThread)
        #When the function is started, the thread is started automatically.
        # If you press it again, it stops the thread and therefore the data acquisition. Pressing it again reactivates the thread
        self.graph = plt.figure() #display the figure

        self.figurecanvas = FigureCanvas(self.graph)
        self.navigationtoolbar = NavigationToolbar(self.figurecanvas, self)


        Layout1 = QVBoxLayout()

        Layout1.addWidget(self.figurecanvas)
        Layout1.addWidget(self.navigationtoolbar)
        Layout1.addWidget(self.buttonpause)
        self.setLayout(Layout1)
        self.index = 0
        self.refresh = True
        self.thread = threading.Thread(target=self.plotLoop, args=())
        self.thread.start();


    def plotLoop(self): #allows data to be updated at a given interval
        while self.refresh:
            self.plot() #displays the graph and updates itself every interval
            time.sleep(1) #needs to give a interval to update

    def startThread(self): #allows you to overwrite the current window and reopen one at the given interval
        if(self.refresh):
            self.refresh = False
            self.thread.join()
        else:
            self.refresh = True
            self.thread = threading.Thread(target=self.plotLoop, args=())
```

```
133
134     def DisplayData(self):
135
136         data = table_experiences #use the table containing the data stored in the database
137         self.index = (self.index+1)%len(data)#moves on to the next term
138         data[self.index] = np.random.randn()#for this example, calls the function random
139         self.graph.clear() #allows to restore the axes and to avoid overlapping when the ordinates and abscissas are too far apart
140         ax = self.graph.add_subplot(444)#allows you to position the location of the graph in the window
141         ax.plot(data, '*-')
142         self.canvas.draw() #display the navigationtoolbar
143     ##########################################################################################"""
144
145 class CloseTool(QWidget):
146     def __init__(self):
147         super().__init__()
148         self.pushButton = QPushButton('Quit', self)
149         self.pushButton.clicked.connect(Menu.quit)
150
151
152
153
154 ##############################################################################""
155
156 def main():
157     app = QApplication(sys.argv)
158     ex = Menu()
159     ex.showMaximized()
160     sys.exit(app.exec_())
161
162
163 if __name__ == "__main__":
164     main()
165
166
167
```

Appendix E : Possible Ameliorations and Project Continuity

In order to make the boards programmable, Serial Wire Debug ports will be routed on the EXP MOD and the TELEM MOD (Emitter). These ports will allow us to burn proper bootloaders on the microcontrollers to establish a connection with the Arduino IDE.
It may be possible, once the full project is up and running well, to optimize the layout and reduce boards diameters.
Electro-Magnetic Compatibility should be studied too.

Concerning the EXP MOD, USB port protection and a flash memory could replace the SD Card.
Concerning the HCI, it remains to test the reception of the data by the SPI protocol of the telemetry via the Raspberry.

Connections between all the modules are still to be made.

The whole DECRYPT project has encountered a lot of interest in the association, and is sure to be continued.

Project Related Appendices

Appendix F : Self-Evaluations

Léo Marmande (EXP MOD) :

Field	Previous capabilities	New capabilities
Circuit design and KiCAD	Beginner	Intermediate
Power Consumption establishment	None	Taking power management into account in the design
Oscillators and parasitic capacitance	None	Basic quartz implementation
Digital Electronics	Development boards-based design	Microcontroller approach
Architecture Engineering	Unknown field	Basic Capella-based approach
Documentation	Basic	Full datasheet writing, better marketing
Communications Protocol	Beginner	Better knowledge of the theory, low-level approach

Étienne Sommier (TELEM MOD) :

Field	Previous capabilities	New capabilities
Circuit design and KiCAD	Beginner	Intermediate

Antenna	None	Basic antenna implementation and dimensionning
LoRa communication protocol	None	Basic design of Emitter/Receiver and RF theory
Documentation	Basic	Full datasheet writing, better marketing
Architecture Engineering	Unknown field	Basic Capella-based approach

Pierre-André Lhermite (HCI) :

Field	Previous capabilities	New capabilities
Programming in Python	Beginner	Advanced knowledge of Python
Programming in PyQT	None	Intermediate
Using a Raspberry/Hardware part	None	Able to use a Raspberry with ease
Computer program documentation	Beginner	Find information faster, better marketing
Knowledge of databases	None	Beginner
Architecture Engineering	None	Basic Capella-based approach

Appendix G : Technical Resumes

Spécialité Génie Physique et Systèmes Embarqués — Polytech Orléans

FICHE COMPETENCES / C.V.

LHERMITE Pierre-André
ELEVE INGENIEUR en Génie Physique et Systèmes Embarqués

Description du poste/métier envisagé :

Assister un ingénieur dans ses différentes missions et veiller à la préparation et l'exécution des tâches réalisées.

Projets, stages et mobilité intern. réalisés :

- 2020 (6 mois) TIPE de fin de CPGE sur un algorithme de calcul de puissance pour la station spatiale internationale
- 2021(4mois) ¨Projet d'études ingénieur sur la simulation d'éclairement d'un bâtiment.
- 2021-2022 (5mois) Projet DECRYPT : réalisation d'une IHM pour un système embarqué modulaire.
- 2021-2022 (5 mois) Projet STM32 : acquisition de données avec une base de données (horodatation) via un microcontrôleur STM32
- Projets associatifs : • Mini fusée Sunlight: mesures d 'accélération-vol balistique
  • Fusée Expérimentale Starlight : création de plasma par décharge électriques permettant le contrôle d'écoulement sur un vol de fusée (en cours de développement)

Soft-Skills :

Rigueur	●●●●●○
Curiosité	●●●●○○
Créativité	●●○○○○
Adaptabilité	●●●●○○
Autonomie	●●●●○○
Coopératif	●●●●●●

Langues :

TOEIC	800
Anglais	●●●●○○
Allemand	●●●○○○
Français	●●●●●●

Compétences

Techniques :

Acquises :

- Programmation en C++, Python, QT Creator,Arduino , Android Studio, Java
- Utilisation de carte Arduino et de carte STM32
- Utilisation de matériel optique (goniomètre, interféromètre de Michelson)
- …

A conforter :

- Utilisation plus poussée des logiciels Mbed et IDECube

Méthodologiques :

Acquises :

- Gestion de projet (Capella)
- Gestion administrative (secrétaire général d'association)
- Travail en équipe

A conforter :

- Utilisation plus poussée de Capella
- Gestion d'équipe en tant que leader

Théoriques :

Acquises :

- Mathématiques
- Mécaniques de l'objet
- Chimie thermodynamique
- Lois de l'électronique et de l'électromagnétisme
- Génie Physique
- Architectures processeurs

A conforter :

- Liaisons séries
- Interruptions/timers

Date de mise à jour 14/01/2022

École polytechnique de l'université d'Orléans

Premier réseau français des écoles d'ingénieurs polytechniques des universités

8 rue Léonard de Vinci
45072 Orléans cedex 2
France
Tél. +33 (0)2 38 41 70 50
Fax. +33 (0)2 38 41 70 63

Spécialité Génie Physique et Systèmes Embarqués — Polytech Orléans

FICHE COMPETENCES / C.V.

MARMANDE Léo
ÉTUDIANT INGÉNIEUR en « Génie Physique et Systèmes Embarqués »

Description du poste/métier envisagé :

Ingénieur en systèmes embarqués appliqués au traitement de signal dans le domaine du spatiale

Projets, stages et mobilité intern. réalisés :

- 2018 (6 mois) ElectroBarman : machine à cocktail contrôlée par Bluetooth
- 2019-2020 (6 mois) GPS : géolocalisation et traçage sur carte du chemin parcouru
- 2021 (6 mois) Luminaire connecté avec asservissement
- 2022 (6 mois, sujet à extension) DECRYPT, électronique d'expérience et télémétrie universelle
- 2020-2022 : mini-fusée et fusées expérimentales (projets associatifs)

Soft-Skills :

Pédagogue	●●●●●●
Curiosité	●●●●●●
Créativité	●●●●○○
Adaptabilité	●●●●○○
Autonomie	●●●●○○
Coopératif	●●●●●○

Langues :

TOEIC	990
Anglais	●●●●●●
Français	●●●●●●
Allemand	●○○○○○

Compétences

Techniques :

Acquises :

- Design de circuit analogiques et numériques
- Programmation C « low-level »
- Preuves de concept avec cartes de développement
- Consommation électrique

A conforter :

- Approche « low-level » des microcontrôleurs (registres et fonctions internes)
- CEM
- Intégration de quartz

Méthodologiques :

Acquises :

- Rédaction de datasheet
- Approche « système »
- Bon relationnel

A conforter :

- Capella
- Capacité de plannification et de prévision
- Gestion d'équipe

Théoriques :

Acquises :

- Niveau intermédiare en Traitement du Signal analogique et numérique
- Filtrage

A conforter :

- TDS et Filtrage avancés
- Aspects et considérations physiques dans l'électronique (impédance, etc)
-

Date de mise à jour 06/07/17

UNIVERSITE D'ORLEANS
POLYTECH
École polytechnique de l'université d'Orléans
Premier réseau français des écoles d'ingénieurs polytechniques des universités
8 rue Léonard de Vinci
45072 Orléans cedex 2
France
Tél. +33 (0)2 38 41 70 50
Fax. +33 (0)2 38 41 70 63