Projet RedFly

Rapport de vol

ESILV Pôle Léonard de Vinci

Contents

1 Introduction

L'association LéoFly du pôle universitaire Léonard de Vinci participe une fois de plus cette année à la campagne de lancement C'Space organisée par l'association Planète Sciences et le CNES pour l'édition 2025. Le projet de cette année consiste à réaliser une fusée expérimentale de type Fusex en embarquant plusieurs expériences pour le vol. Pour le lancement de la 7è Fusex de l'association, les deux systèmes expérimentaux sont les ailerons réalisés avec une nouvelle méthode et l'embarquement d'un nouveau système de caméras à son bord.

1.1 Équipe

Le projet RedFly à nécéssité la collaboration de plusieurs équipes travaillant sur plusieurs systèmes différents :

Figure 1: Les membres présents au C'Space 2025

Equipe mécanique

- Maxime Tardieu - Chef de projet
- Matthieu Pecoraro - Chef de projet
- Mathis Pile
- Ethan Anoufa
- Mohamad Jaafar
- Mathilde Bournon
- Manon Gaudillère
- Guillaume Scheid
- Elisabeth Cognet
- Mathilde Haven
- Immanuella Castro
- Margaux Coudeiras

- Balthazar Sanchez
- Ludovic Bocquillon
- Mathias Couder
- Florian Cheron
- Camille David
- Mathis Le Texier
- Charles Léonelli Wendlings
- Eden Elfassy
- Mauricio Espinoza Mayer

Equipe usinage

- Jules Amardeilh - Chef de projet
- Gabriel Ghanem - Chef de projet
- Emile Courgelongue
- Terence Roumillac
- Mathéo Baradat
- Léontine Quénu

Equipe électronique

- Julien Finkler - Chef de projet
- Charles Boullay
- Thomas Lampure
- Tom Alglave
- Thibaut Reboul
- Antoine Lesort
- Lea Montaron
- Alexia Roulet
- Abdul Benyzid
- Gaspard Moulin
- Madeleine Froget

1.2 Acteurs du projet

- LéoFly

LéoFly est une association d'aéronautique et d'aérospatiale fondée en novembre 2015. Elle a pour objectif de rassembler les étudiants du Pôle Léonard de Vinci (à Paris La Défense) autour de leur passion commune. Pour ce faire, les projets proposés aux étudiants s'étendent sur plusieurs domaine, l'astromodélisme, l'aéronautique, les systèmes embarqués, ect, dont certains sont en collaboration avec le CNES et Planete Sciences lors du C'space ainsi que des conférences et des visites dans le domaine de l'aéronautique ou du spatial.

- Centre National d'Etudes Spatiales (CNES)

Le Centre National d'Etudes Spatiales (CNES) est un établissement public à caractère industriel et commercial chargé d'élaborer et de proposer au gouvernement le programme spatial français et de le mettre en œuvre.

- Planète Sciences

Planète Sciences est une association à but non lucratif proposant aux jeunes passionnés des activités scientifiques et techniques expérimentales, grâce à différents projets durant leurs études. Cette année, Planète Sciences nous a encadré grâce à 3 réunions techniques et de contrôle, ce sont les «Rencontre Club Espace »dites «RCE». Les bénévoles sont également présents sur le Camp de Ger (lieu de la campagne C'Space pour l'édition 2023) pour nous apporter des conseils et pour effectuer les vérifications techniques imposées par le cahier des charges avant de procéder au décollage.

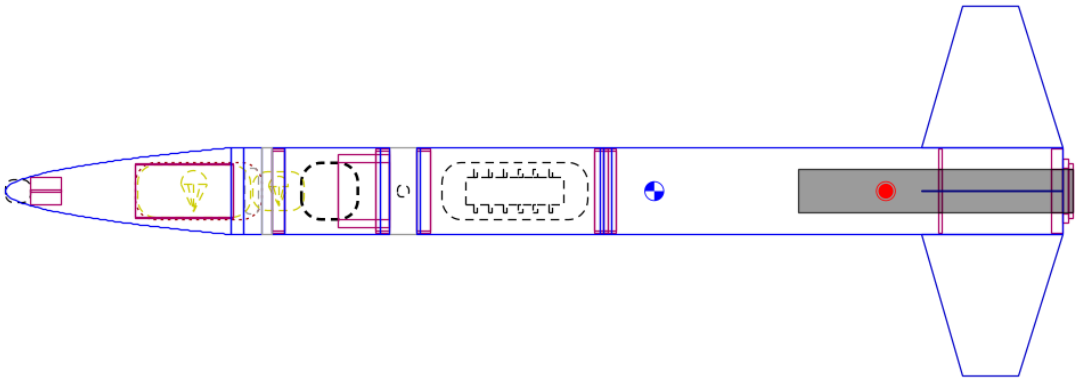

Figure 2: Modélisation OpenRocket de RedFly

2 Partie Mécanique

2.1 Architecture globale

RedFly possède un système d'ejection de parachute par l'ogive. La fusée est donc partagé d'un coté entre l'ogive, dans laquelle est stockée le parachute, et le tube ou corps principal. Le corps principal est divisé en 3 tubes en fibres de carbone ou verre reliés par des bagues en aluminium.

La fusée a pour dimensions principales 189cm de long et un mono-diamètre de 154mm de large. L'ogive fait 400mm de long et les ailerons font 30cm d'envergure et 25cm d'empennage de forme générale NACA.

Les tubes sont en fibres de carbone de 2mm d'épaisseur avec un diamètre interne de 150mm et ont étés sous-traités par l'entreprise Mateduc Composites. Les 3 sections de tubes sont le tube CubeSat, de 15cm de long, le tube Electronique de 30cm de long en fibre de verre ainsi que le tube Propulseur de 80cm de long. Entre les tubes CubeSat et Electronique se trouve la bague Electronique qui correspond à une partie spécifique du projet. La dernière des 6 parties mécaniques du projet est le mécanisme de séparation.

Le choix des fibres de carbone et verre est motivé pour assurer une très grande solidité et créer un assemblage qui présente une flèche négligeable. Le tube est réalisé à partir de fibre T700 avec une couche en orientation transversale et deux allers-retours à +/- 20° sur un diamètre intérieur de 150 mm et une épaisseur de 2 mm. La soutraitance est nécéssaire car nous n'avons pas à disposition un atelier assez grand pour permettre de créer des pièces composites de grandes dimensions.

2.2 Expérience principale : Ailerons

L'expérience principale de RedFly est le processus de fabrication des ailerons qui est nouveau dans l'association et qui permet d'allier une solidité qui se rapproche de celle d'un set en monobloc composite en gardant un poids très léger.

Afin de mieux caractériser leur comportement en vol, nous intégrons des cartes de capteurs de vibration dans les ailerons afin de pouvoir caractériser quelles sont les fréquences de résonnance aux différentes phases du vol. Cela nous permettra de pouvoir déterminer si le procédé utilisé pour la fabrication des ailerons est adaptée à l'utilisation d'un propulseur Pro75.

Nous avons opté pour un système avec la carte du capteur directement intégrée dans l'aileron afin d'avoir une récupération des données sur place et une fréquence d'échantillonage la plus élevée possible. L'utilisation d'un capteur bandelette logée entre le plastique et la couche en carbone à été écarté car ne permet pas de disposer des informations de vibrations dans les 3 dimensions.

Nous sommes donc partis sur une approche d'intégrer l'emplacement du capteur dans le bout de

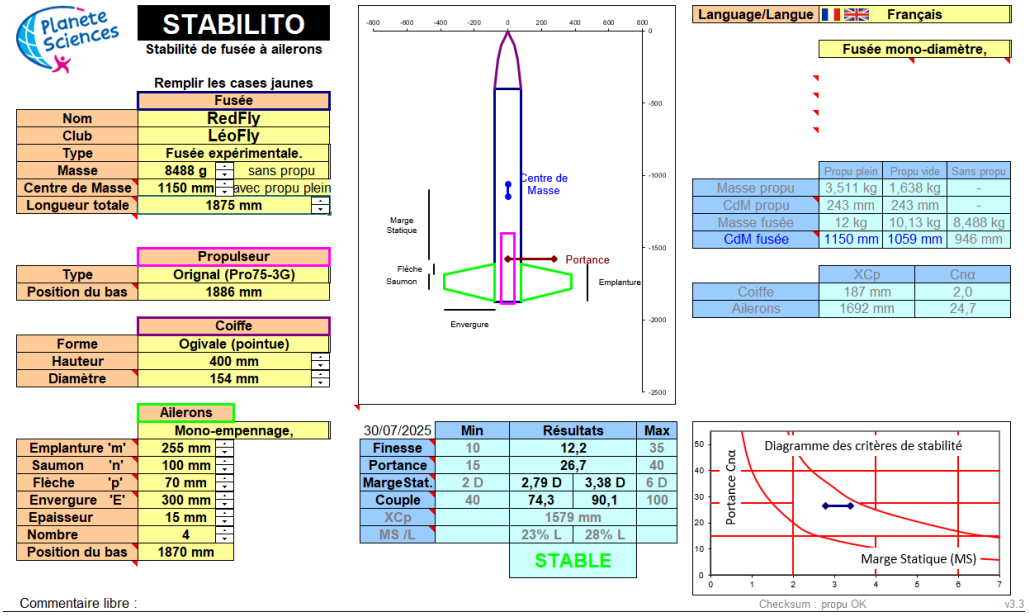

Figure 3: Dimensions caractéristiques de la fusée

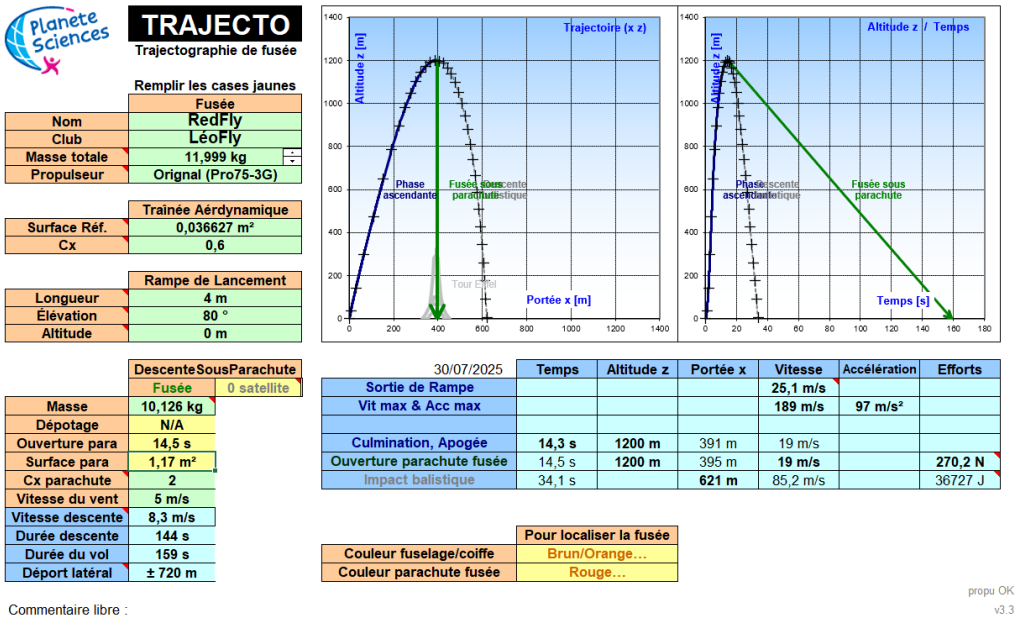

Figure 4: Trajectoire théorique de la fusée

l'aileron dans une poche dimensionnée pour. Une rainure de la taille de quatre fils parcours la longueur interne de l'aileron.

2.2.1 Fabrication

La forme interne de l'aileron a été imprimé avec du filament Nylon-CF, c'est-à-dire du nylon renforcé avec des fibres de carbone. Ce filament offre de très bonnes resistances à la déformation et une adhésion inter-couches excellente.

La feuille de carbone utilisée est une feuille en treillis classique 45/45° de 180g/m². Nous avons

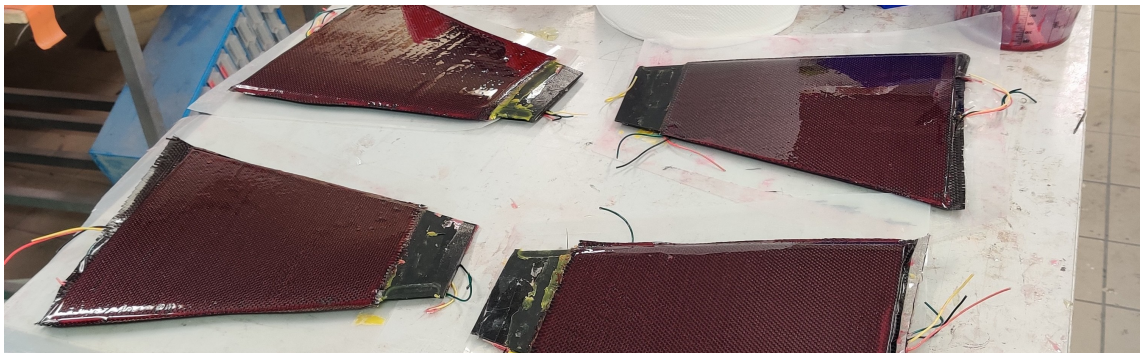

Figure 5: Ailerons à la fin de l'étape de pose de la résine

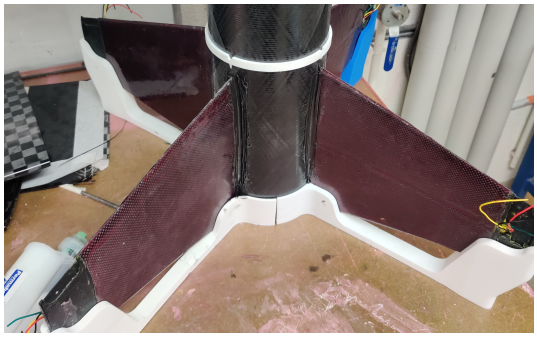

Figure 6: Ailerons entrain d'être collés sur le tube

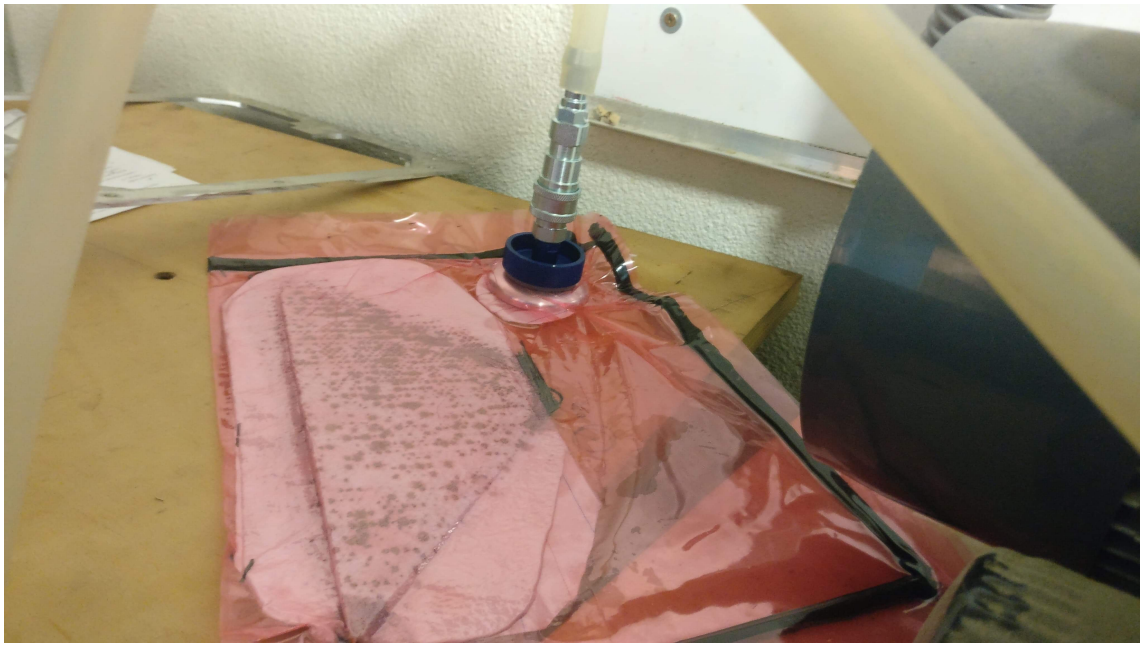

Figure 7: Exemple de mise sous vide pour plaquer le carbone imbibé de résine sur le corps imprimé en 3D

également ajouté du colorant rouge dans la résine afin d'élargir l'identité visuelle de la fusée jusqu'au ailerons.

Nous mesurons un poids de 1412g pour les 4 ailerons combinés, une réduction de 20% par rapport

à l'année précédente. Avec de cales imprimées aux formes des ailerons nous pouvons les aligner avec une grande précision sur le tube. Nous utilisons de la colle structurelle bicomposants spéciale carbone/carbone afin de coller les ailerons sur le tube en rajoutant un congé de 2cm pour répartir les efforts sur une plas grande surface de contact de colle.

Contrairement à la feuille de carbone, la colle n'est pas flexible quand elle est sèche et représente donc le composant qui va céder en cas d'effort trop intense. Nous avons donc fait des tests et avons trouvé qu'avec un congé de 1,5cm, la colle cède aux alentours de 210kg excercé sur l'ensemble du tube, soit 110kg par ailerons. Nous avons décidé de partir sur 2cm de congé de colle structurelle pour encore augmenter la solidité.

A la fin de l'opération de colle, nous avons utilisé une centaine de grammes de colle, et en comparant sur le système monobloc de l'année précédente, nous passons de 4,3kg utilisé pour les ailerons à 1,6kg. En prenant une marge de sécurité de 4, nous avons significativement allégé le système d'ailerons. Le comportement de la fusée en vol ainsi que les données de vibrations viendront caractériser si des fréquences de raisonnance dangereuses sont présentes dans les ailerons.

2.3 Bagues en aluminium

Afin de connecter les différents tubes et parties ded la fusée nos utilisons des bagues en aluminium usinées dans l'atelier du pôle universitaire. L'aluminium est un matériau très resistant pour son poid et facile à usiner en machine. Nous avons choisi d'utiliser l'aliage 7075 qui possède des coefficients de compression et d'elasticité deux fois supérieurs aux autres aliages, ce qui en fait un choix privilégié. Les bagues permettent ainsi de connecter les tubes entre eux et offrir une flèche totale théorique inférieure à 50µm sur les tests de flèche. Les bagues viennent finalement être collées à la colle structurelle composite/métal aux tubes afin de minimiser le nombre de vis nécéssaires à l'assemblage.

2.3.1 Bague de reprise

Nous avons cette année décidé de créer un modèle de bague de reprise pouvant être utilisée sur plusieurs calibres de moteurs, du Pro54 au Pro98. Elle est donc divisée en 2 parties, l'une avec un diamètre interne très large qui est collée au tube, et une deuxième partie qui vient adapter le diamaètre du propulseur sur la reprise en se posant sur la première et est retenue par plusieurs vis.

La retension vient se visser autour du moteur pour permettre une installation la plus simplifiée possible pour les pyrotechniciens.

2.3.2 Bagues de jonction

Pour joindre les sections de tube ensemble, nous avons créé un set de deux bagues filletées de diamètre M148 afin que deux sections de tubes puissent se visser ensemble de façon universelles. Le pas de filetage choisi est 1mm, mais les 1ers essais montrent que cela nécéssite beaucoup de tours pour visser et qu'un pas de 2 voire d'1,5mm seraient surement plus pratique pour l'assemblage.

2.3.3 Bague électronique

La bague électronique est l'emplacement de la fusée sur lequel le système électronique de la fusée peut interagir avec l'extérieur.

On retrouve dessus:

- **Caméras** : 4 emplacements dans lesquels sont placés les caméras pour filmer le vol
- **Clés de sécurité** : 2 emplacements dans lesquels viennent s'insérer les 2 clés de sécurité des systèmes électroniques (systèmes séquenceur et expériences)

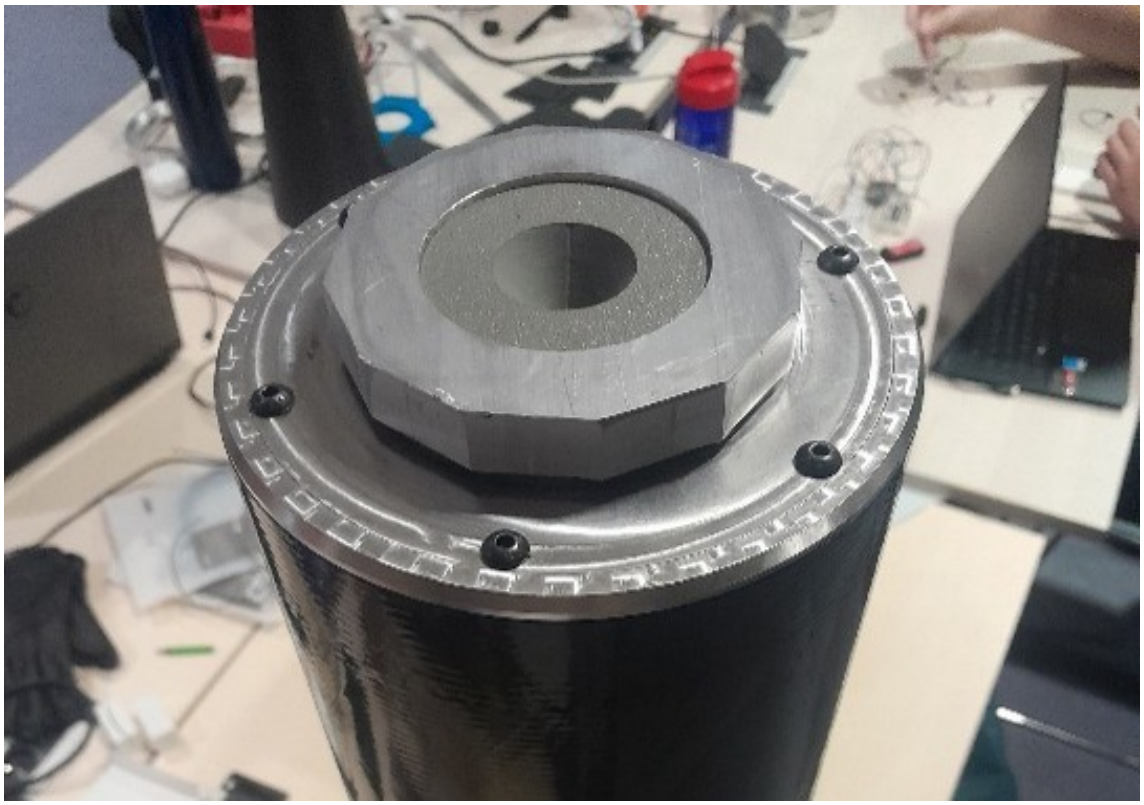

Figure 8: Assemblage de la reprise propulseur avec sa rétention autour d'un Pro75 factice

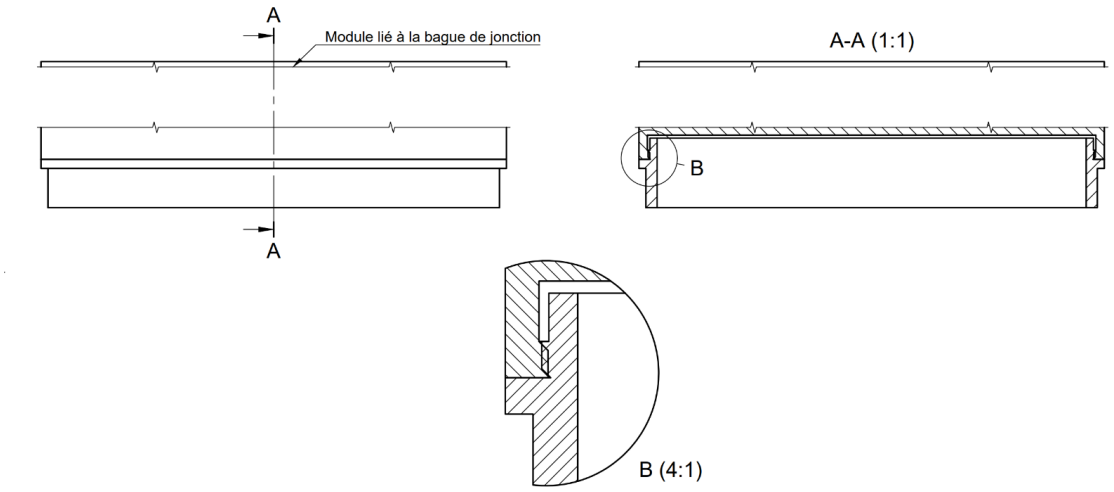

Figure 9: Plan des bagues de jonction

- **Boîtier jack** : Emplacement pour une carte avec un connecteur Jack femelle pour détecter le décollage à l'arrachage d'un cable jack arrimé à la cage de lancement
- **Carte LED** : Un emplacement est destiné à pouvoir voir les LEDs d'une carte interne depuis l'exterieur de la fusée pour connaître l'état des systèmes électroniques.

2.4 Rack électronique

Le rack électronique de RedFly est directement attaché sur la bague électronique. Ce choix a été fait pour ne pas avoir de problème avec la torsion des cables lors de l'emboitement des partes entres elles avec les bagues de jonction filletées!. Il est constitué des cartes et des batteries fixées dans des supports compatibles rails DIN.

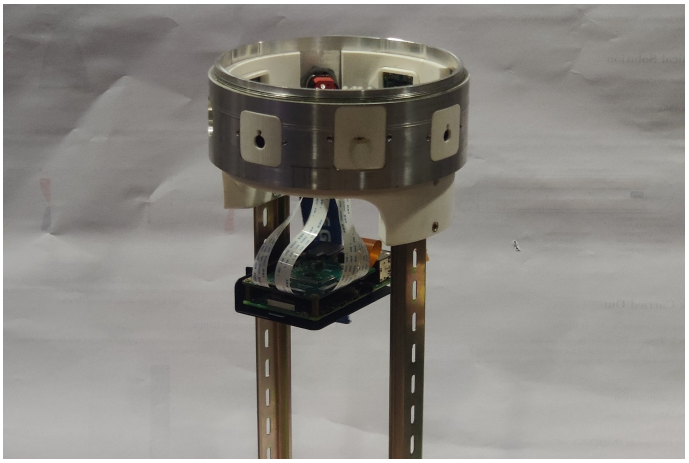

Figure 10: Photo des rails DIN accrochés dans la bague électronique avec une carte retenue dans son support

Les supports de cartes ont étés soit achetés spécialement pour leur fonctionnalité de maintient en position au rail par une vis, soit imprimés en PETG avec des inserts de vis de maintient. Les différentes cartes électroniques sont donc maintenues horizontalement sur des entretoises, les unes au dessus des autres. Les batteries 2s et 3s ont étés emboités dans des cavités prévues dans les supports imprimés en 3D.

2.5 Séparation

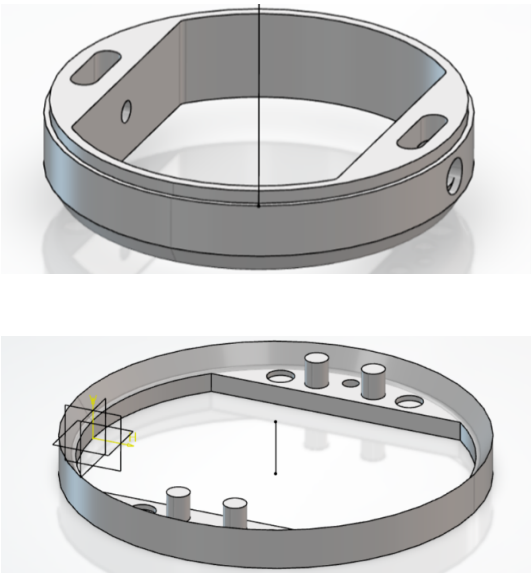

Figure 11: Bagues haute et basse de la séparation

La séparation de RedFly est horizontale au niveau de l'ogive. Elle est constituée de 2 bagues en aluminium emboitées l'une dans l'autre et repoussées par 4 ressorts entre les deux.

Pour créer la tension nécessaire pour relier les 2 bagues entre elles, nous avons fixé un corde Dyneema de 1,5mm de diamètre. Nous venons l'attacher avec un nœud dans la bague du bas, elle passe dans la bague du haut puis est rattaché à une vis fixée dans la partie basse. On vient ensuite serrer la vis avec une clé Allen qui met en tension le cable et ressert les parties entre elles. La vis est applatie d'un côté et est ainsi bloquée en place pour assurer la sécurité du système.

A l'apogée le séquenceur viens faire chauffer à blanc une aiguille de nichrome plantée au cœur de la corde qui la sectionne et libère la tension du système.

2.6 Parachute

Le parachute est de 91cm de diamètre acheté chez Fruity Chutes est stocké dans la coiffe de l'ogive et la longe à laquelle il est attaché est tirée par l'ogive lors de son éjection. Il est relié à l'ogive par une corde Dyneema et au tube principal par une corde d'escalade attachée dans l'émerillon de reprise. La corde est nouée par un double nœud de pêcheur à l'émerillon qui est collé à la colle structurelle sur le tube. Il resiste en essai à 90kg la limite du matériel disponible.

2.7 Usinage

Figure 12: Centre d'usinage du Pôle Léonard de Vinci

L'ensemble des pièces en aluminium à été usiné sur le centre 3 axes à commande numérique (CNC) de l'école. L'alliage utilisé est le 7075 pour ses propriétés de résistances et légèretés. La précision mesurée sur les pièces de tests est de 10µm environ et la machine et la maéthode utilisée sont donc adaptées à l'utilisation pour des pièces mécanique de prototype comme notre fusée.

Une très grande proportion des bruts utilisé se présentent sous la forme de plaques de 23mm de large environ et les bagues ont été modélisées dans cette contrainte d'épaisseur!. La bague electronique à elle été fabriquée dans un brut cylindrique de 160mm de diamètre et 100mm de longueur

Les morceaux de bruts sont découpés à la scie à ruban avant d'êtres directement calés dans la CNC pour être usinés. La programation de l'usinage a été réalisée sur Fusion360 avec les dimensions des bruts découpés.

Chaque programme comporte 2 opérations pour chaque face de la plaque usinée et incorpore une passe de finition.

La bague électronique a nécéssité l'utilisation d'un diviseur d'angle pour réaliser les inserts de chaque partie verticale.

Figure 13: Usinage de la bague électronique

2.8 Impression 3D

Chaque pièce usinée a d'abord été imprimé en 3D pour vérifier sont intégration. La précision d'environ 0,1mm des machine ne permet pas de tester les contraintes mécaniques mais donnent tout de même une idée de la pertinence de la modélisation.

Nous avons utilisé pour le prototypage du filament PLA sur des imprimantes Bambulab A1 mini et pour les pièces finales du PETG.

Les ailerons ont étés imprimés sur une Bambulab P1S avec du filament Nylon-CF en chambre chauffée tout du long de l'impression.

3 Partie électronique

3.1 Expérience : Capteur de vibration

L'expérience principale de la fusée RedFly est l'incorporation d'un capteur de vibrations dans chacun des ailerons de la fusée.

Un rapport détaillé sur la fabrication desdits capteurs est inclu en annexe de ce rapport et sur SCAE pour plus de confort.

3.1.1 Objectif de l'expérience

La mesure des vibrations des ailerons a lieu dans le cadre de l'essai par LéoFly d'une nouvelle méthode de fabrication des ailerons composites. L'objectif est de détecter au cours du vol la présence de vibrations parasites induites par les efforts de l'air sur la surface des ailerons.

3.1.2 Calibration

Chaque capteur est intégré dans le plan de corde des ailerons. Les 3 axes de mesures sont donc X en direction normale à la fusée, Y normale à la surface des ailerons et Z dans la direction de la pointe de la fusée. Les données sont enregistrées sur une carte SD en format .csv pour être lues facilement après récupération.

La calibration initiale des capteurs se fait en branchant la carte à une alimentation. Nous posons ensuite la carte sur ses différentes faces pour lire les données ±1G sur chaque axe. Une rapide secousse des carte nous montre bien un pic sur le graphique des accélérations.

Les différents tests que nous avons fait nous montrent que la carte peut de façon stable et sécurisée enregistrer jusqu'à 300Hz les échantillons d'accélération. On peut donc mesurer les vibrations jusqu'à une fréquence de 150Hz.

La dimension Z est celle qui nous intéresse le moins dans notre cas car les effets aérodynamiques que nous recherchons sont perpendiculaires à cet axe et masqués par les vibrations induites par le propulseur. Nous limitons donc le capteur à 20G, un facteur de 2 au dessus de l'accélération produite par le moteur au cours du décollage. Ainsi nous optimisons la résolution des observations du capteur.

Nous pourrons également quantifier la rotation de la fusée avec la différence des accélération a_1 et a_2 en X de deux capteurs oposés avec la formule dérivée de la force centriguge $\dot{\omega} = \sqrt{(\frac{a_1+a_2}{2\times 0,2m})}$.

Pour étudier les vibrations nous enlèrerons donc à chaque échantillon les forces centrifuges et de gravité pour se restreindre aux effets aérodynamiques du vol.

3.2 Séquenceur

Un rapport détaillé sur la conception du séquenceur a été inclu en annexe de ce rapport et sur SCAE pour plus de confort.

Le but général du séquenceur cette année est d'être universel, petit et relativement peu cher à produire. En attendant d'intégrer des capteurs de détection d'apogée sur les prochaines versions, nous avons fait le choix de faire un PCB minuteur pour l'éjection du parachute.

3.3 Caméras

Nous voulons pour la communication autour du projet de cette année avoir une vidéo 360° du décollage et du vol de la fusée. Pour ce faire nous avons choisi de filmer le vol avec 4 caméras qui pointent cardinalement sur les côtés de la fusée. Nous aurons ainsi une vision 360° horizontale et 120° verticale du vol.

Nous avons choisi l'option d'un module pour Raspberry Pi avec directement 4 cartes caméras reliées par des cables rubans. Ce module a été acheté en ligne chez Arducam.

Les tests que nous avons réalisés sur la carte Raspberry nous on montré que la fusion des 4 caméras en vol n'était pas possible avec la puissance insuffisante de la Ram de l'ordinateur de bord. Nous regrouperons les 4 vidéos après le vol sur un logiciel de montage vidéo.

3.4 Système général

L'électronique de la fusée est consituée de deux circuits séparés, le circuit séquenceur et le circuit expérience. Chacun de ces circuit posséde sa propre alimentation, sa propre clé d'armement et sa propre détection de décollage via un port jack avec 4 pins.

Nous avons utilisé pour l'alimentation des batteries LiPo 2s2p et 3p1s achetées dans le commerce avec BMS intégré.

N.B. pour les prochains projets LéoFly : Normes Européennes ≠ BMS intégré. Faire très attention au moment de l'achat. Toujours vérifier la présence de BMS avant ET après avoir reçu les batteries.

4 Vol

4.1 Déroulement du vol

Les préparatifs fait entierement en zone club se sont parfaitement bien déroulé et ont permis de passer un temps minimum en tente club. La mise en rampe s'est très bien passé sans erreur dans la chronologie.

Les conditions climatiques donnent un vent modéré qui vient face à la direction de la rampe. Avec son parachute légerement surdimensionné et sa vitesse de retombée de 8m/s, la direction des vols propose de passer l'inclinaison de la rampe de 80° à 75° pour éviter tout risque de trajectoire dangereuse en cas de bourasques plus fortes que prévues. La modification est validé par Planète Sciences et LéoFly.

RedFly décolle le mercredi 9 juillet 2025 aux alentours de 16h sous un ciel bleu et totalement dégagé. La trajectoire est très droite et la séparation de l'ogive s'effectue bien aux 14,5s théoriques.

Le parachute se déploie d'un coup très sec et arrache l'émérillon du tube dans lequel il était collé. La violence du choc fait se détacher le tube du reste de la fusée et romp également la suspente qui relie le parachute à l'ogive.

Le reste du corps finit en trajectoire ballistique et le tube arraché et l'ogive s'éparpillent de leur cotés respectifs. Le parachute non endommagé fini de retomber 5min plus tard.

Les positions GPS approxiamatives de retombée du corps principal et du parachute sont notées.

4.2 Analyse du vol

A l'aide de la video fournie par le tracker DGA, de sa position GPS, des angles qu'il indique et de la direction de lancement, nous avons pu retracer la trajectoire suivie par la fusée au cours du vol.

Nous apprenons aussi que l'ouverture du parachute s'est faite à 40,5m/s au lieu des 20m/s espérés. La force exercée sur le parachute à alors quadruplé par rapport à l'estimation du stabtraj. Deux facteurs sont à retenir parmis ceux qui ont mené à l'arrachage du parachute.

Le premier est la taille et le couple particulièrement importants des ailerons par rapport à sa hauteur. Leur authorité à créé une trajectoire très droite pas modélisée sur les logiciels StabTraj et OpenRocket, qui a légerement augmenté la vitesse de la fusée à l'apogée du vol.

Le second est le passage d'une inclinaison de 80° à 75° de la rampe de lancement. La vérification avec le Stabtraj nous confirme que cela augmente fortement la vitesse à l'apogée et double la force de déploiement du parachute.

On peut également constater une rotation très faible (<1 tour/s) sur les vidéo de suivi du vol, ce qui nous conforte dans la méthode de collage des ailerons avec des calles sur mesure.

4.3 Récupération

La trajectoire calculée grace au tracker DGA nous a permit de calculer la position d'impact ballistique du corps avec une plus grande précision que les observateurs manuels, avec un décalage réel d'une dizaine de mètres par rapport à la position calculée sur Python. (NB : pour utiliser cette méthode de récupération c'est très pratique de s'aider des pics des Pyrénées pour tracer une ligne très précise avec l'emplacement du tracker.)

Le parachute est retrouvé grace aux coordonnées données par les spotters. Le tube vide arraché est retrouvé par une autre équipe à une trentaine de mètres de l'impact du corps, et l'ogive n'est pas à ce jour retrouvée.

L'impact sur un sol dur viens très fortement impacter le rack élec qui est le premier élement interne à 40cm de profondeur dans le tube à rencontrer le sol. La carte SD non judicieusement placée sur le dessus de la carte Raspberry est retrouvée fracturée et non utilisable pour récupérer les données.

La violence du choc vient arracher le tube propulseur du reste planté et rebondi à 3m de là, l'aileron qui prend contact lors du rebond est détaché du tube mais ne se brise pas.

4.4 Analyse des résultats

La destruction des cartes ne permet de récupérer les données du vol donc nous devrons nous contenter des informations que l'on a.

Pour les caméras, l'utilisation d'un système Raspberry avec quatre caméras déjà intégrées a permis de ne pas perdre de temps pour la conception globale du système, mais à nécéssité beaucoup plus de debug qu'initialement prévu. L'architecture non modulaire du système ne présentait pas assez de puissance pour combiner directement les quatre flux en un. Une approche maison, bien que plus lente aurait permis plus de flexibilité pour implémenter directement le flux 360°. Nous n'avons suite à la destruction des données pas de confirmation du système de buffer en condition de vol.

Pour les ailerons, nous n'avons donc pas les données, mais plusieurs élements viennent nous montrer qu'ils ont parfaitement remplis leur rôle et que la nouvelle méthode de fabrication est très pertinente. L'observation du vol nous montre que les ailerons n'induisent pas de modification de trajectoire suite à des vibrations. On observe que lors du déploiement du parachute, la fusée se cabre à 90° avant d'arracher le tube. En 0,3s et moins de 3 oscillations la fusée est de nouveau avec son attitude originale. Cela nous confirme leur robustesse en vol. Egalement à l'impact, un aileron pris l'impact de plein fouet du poids de son tube et du casing propulseur pour un total de 3,5kg à environ 200G. La colle tenant l'aileron a cédé mais il ne s'est pas fracturé. Cela rejoint les conclusions que l'ont avait eu après les différents tests de solidité que l'aileron était très solide en lui même. Nous continuerons donc d'utiliser la méthode de la forme NACA en Nylon-CF entourée d'une unique couche de carbone.

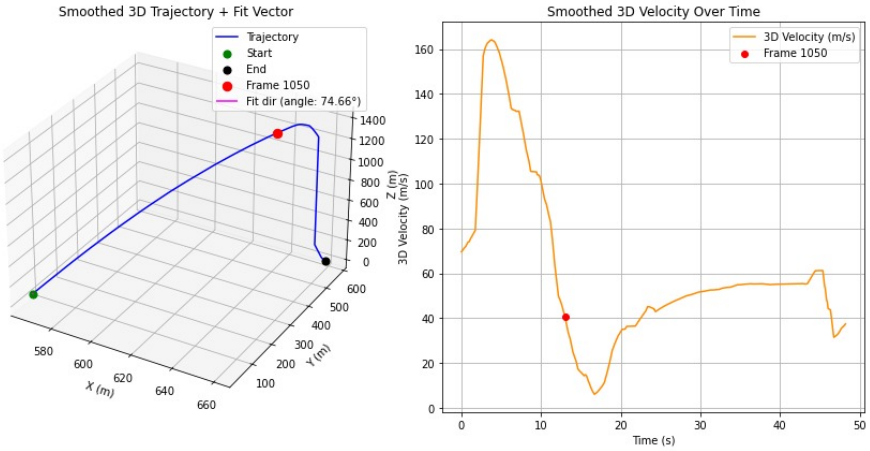

Figure 14: Trajectoire et vitesse calculée avec un script Python

5 Conclusion

La reprise à zéro du projet de l'année dernière à permis à LéoFly de repartir sur de bonnes bases cette année pour aboutir à un lancement et tout de même un vol assez réussi. Les différents systèmes élaborés au long de l'année sont robustes et pourront servir d'une très bonne base de départ pour la prochaine Fusex de l'association. Le déroulé du vol qui a fini ballistique met l'accent sur la nécéssiter de faire tout particulièrement attention à l'intégration des systèmes de parachutes. Ll faut prévoir des tests avec un facteur de sécurité d'au moins 5 pour des élements compliqués à prédire comme le déploiment du parachute et vérifier tout changement fait au pllan de vol. La gestion du projet avec une forme de plusieurs réunions courtes de prototypage par semaine s'est avérée très bénéfique et est également à souligner

La réussite technique et gestionnelle de RedFly va permettre à LéoFly de revenir l'année prochaine et les années suivantes avec des projets encores plus aboutis et une plus grande maitrise de tous les élements tous assurer un bon vol.

6 Remerciements

Nous tenons tous d'abord remercier chalereusement l'association Planete Sciences ainsi que tous les super bénévoles qui nous accompagnés tout au long de cette aventure magnifique du C'space.
Nous remercions également tous nos sponsors pour leur très grand soutien.
On oublie pas les autres assos étudiantes vachement sympas présentes au C'space.
Et un super merci à tous les anciens de l'asso qui nous ont accompagnés tout au long de cette année !!

Figure 15: Merci aux bénévoles, militaires et autres assos qui nous ont aidés à déterrer RedFly !

Merci le CNES et PlaneteScience !

7 Annexe

- Rapport PCB capteur de vibration
- Rapport PCB mini séquenceur

PCB ailerons Redfly

Design Report

ESILV Pôle Léonard de Vinci

Contents

1 Introduction

Ce document rend compte de la conception, l'évolution et de l'état final du pcb s'occupant de l'expérience des vibrations dans les ailerons de la fusée Redfly destinée à décoller lors de la compétition C'Space. Ce PCB à été conçu chez Léofly par Julien FINKLER dans le cadre associatif pour une expérience de rigidité des ailerons en carbone menée par l'équipe mécanique chargée des ailerons à Léofly.

Une étude de rigidité est menée par l'équipe mécanique de LéoFly s'occupant de la fusée Redfly, il s'agit de tester la flexion d'un aileron aucours d'un vol. Il s'agira de comparer les résultats récupérés au cours du vol avec une simulation menée au préalable. La donnée à étudier en question est l'accelération normale à l'aileron.

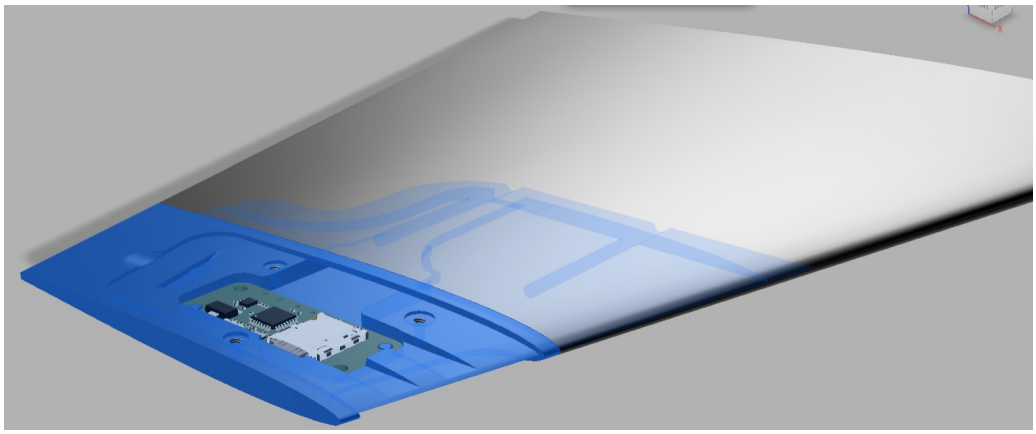

Figure 1: modélisation de l'aileron et du PCB inséré dedans

2 Cahier des charges

Il s'agit de créer un PCB suffisement petit pour tenir dans un aileron de la fusée redfly et doit pouvoir enregister au cours du vol les donnée d'un accéléromètre mesurant la vibration subit par l'aileron. Cet enregistrement doit pouvoir être déclenché au décollage de la fusée par un processus automatisé et dépendant de l'état de la fusée et les données doivent pouvoir être stockées sur le PCB et être récupérées après le vol.

2.1 Contrainte de taille

- La surface du PCB n'est pas très contraignante, il faut juste respecter les dimension surfacique de l'aileron pour ne pas dépasser. Nous avons tout de même cherchés à faire la plus petite possible imposée par la taille des composants choisis.
- L'épaisseur du PCB quant à elle doit être minimisée à tout prix pour avoir l'aileron le plus fin possible. Pour aller dans ce sens des composants particuliers sont choisis pour minimiser cette épaisseur et la PCB est est conçue sur un seul côté.

2.2 Contrainte électrique

- Le PCB doit être alimenté pour répondre au besoin de l'électronique. En l'occurence ici on cherche à tout alimenter à un minimum de 3.3V.
- Le PCB doit pouvoir être programmable depuis l'IDE Arduino tout en minimisant l'espace occupé par le protocole de programmation. L'idéal et de ne pas avoir de port USB, celui-ci occupe trop de place pour être incorporé dans un aileron.
- Le PCB doit pouvoir détecter le décollage de la fusée et ainsi déclencher l'enregistrement ou la transmission des données

3 Choix des composants

3.1 Liste des composants majeurs

Composant	Rôle	Voltages possibles	informations complémentaires
ATmega328p-AU	MCU	V_{IN} = 3.3V / 5V	3.3V -> 8MHz / 5V -> 16MHz
IIS3DWB	Accéléromètre	V_{IN} = 3.3V	SPI -> 3-axes / I^2C -> single-axis
AMS1117-3.3	LDO	V_{OUT} = 3.3V	NaN
MicroSD DM3AT	Stockage	V_{IN} = 3.3V	SPI uniquement
Crystal-4pin	Quartz	NaN	8MHz 9pF

Table 1: Table des composants

3.2 Choix du MCU

L'ATmega328p-AU est un choix de familiarité, il permet de programmer sur l'IDE arduino sans problème, il est très bien renseignéet marche sur une très grande plage de voltage qui la rend très flexible au différents composants, il possède en effet deux configurations intéréssantes :

- V_{alim} = 5V : Cette configuration est celle la plus utilisée, c'est celle qu'arduino utilise sur leurs PCB. Elle permet au MCU de fonctionner avec une fréquence de 16MHz. Néanmoins cette configuration impose un fonctionnement en 5V qui n'est pas compatible avec la plupart des composant récents qui marche en grande majorité à un voltage de 3.3V.
- V_{alim} = 3.3V : Cette cofiguration impose malheureusement un ralentissement du MCU à une fréquence de 8MHz. Néanmoins ce voltage nous permet d'éviter d'utiliser un levelschifter pour pouvoir faire fonctionner notre capteur et notre lecteur de carte SD.

3.3 Considérations sur le voltage

Par rapport à la répartition des différents voltages disponible et des composants qui nous intéressent. Dans le but d'économiser de la place, nous avons alors imposé le voltage de toute la carte à 3.3V. Cela nous permet d'éviter d'avoir 2 LDOs différents et un ou plusieurs levelschifters. Ce choix nous permet donc d'économiser beaucoup de place sur le PCB.

3.4 Absence de pin et composants de programmation

Il existe une méthode pour pouvoir programmer un PCB sans port USB. Cette méthode est fournie par arduino grâce à leurs cartes distribuées dans le commerce et grâce à leur IDE. Il s'agit de la méthode "ArduinoAsISP", elle prend avantage de la nésséctié d'avoir sur toute carte 6 pin dit bootloader qui utilise le protocole SPI d'une arduino et de celui de notre PCB.

(Voir la section "Choix de la méthode de Programmation")

4 Design du PCB

4.1 Schéma électrique

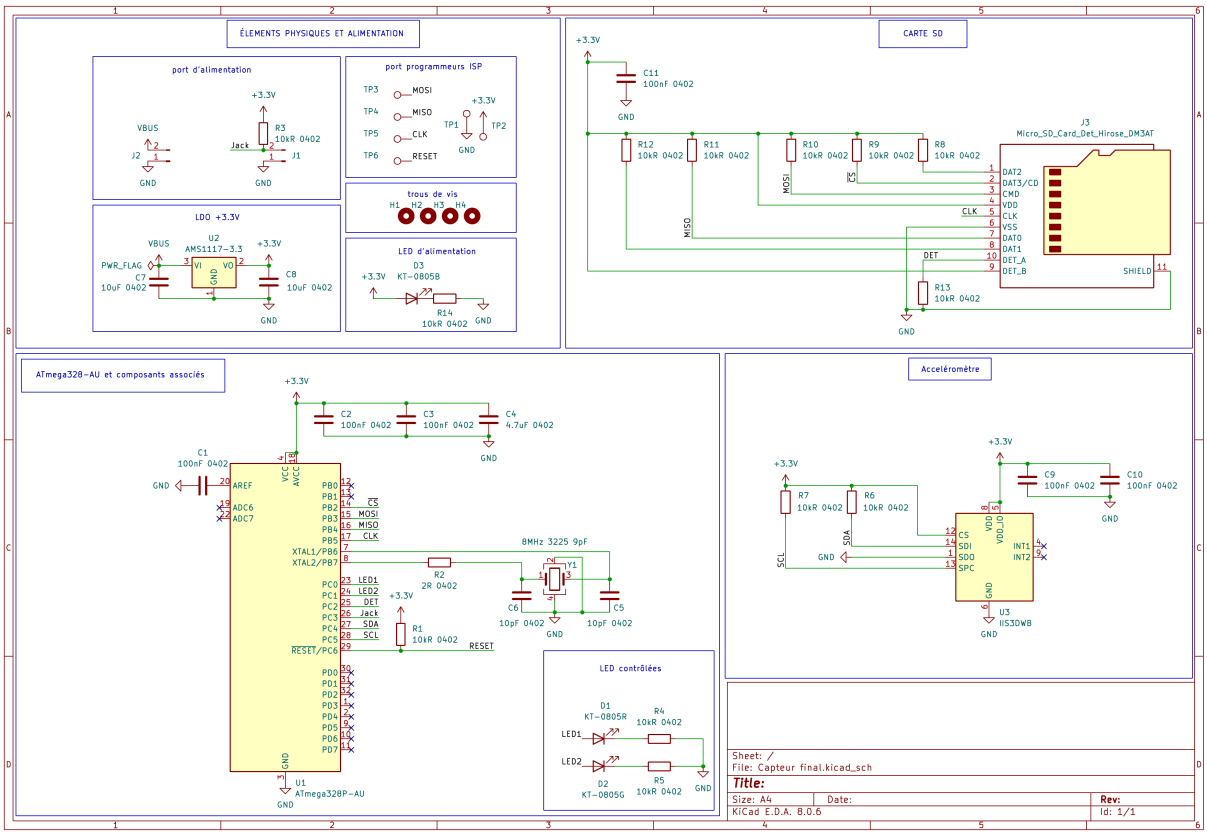

Figure 2: schéma électrique du PCB

Ci-dessus on voit le schéma électrique du PCB. On peut voir que la carte SD est connectée à l'ATmega grâce au protocole SPI, que l'accéléromètre est connecté en I^2C à l'ATmega. On y voit aussi les trois LEDs la bleu indiquant la mise sous tension pour et les LEDs rouges et vertes pilotables par l'ATmega. Le reste du design électrique est un application rigoureuse de ce que nous on indiqués les datasheets des composants respectifs.

4.2 couches de cuivre

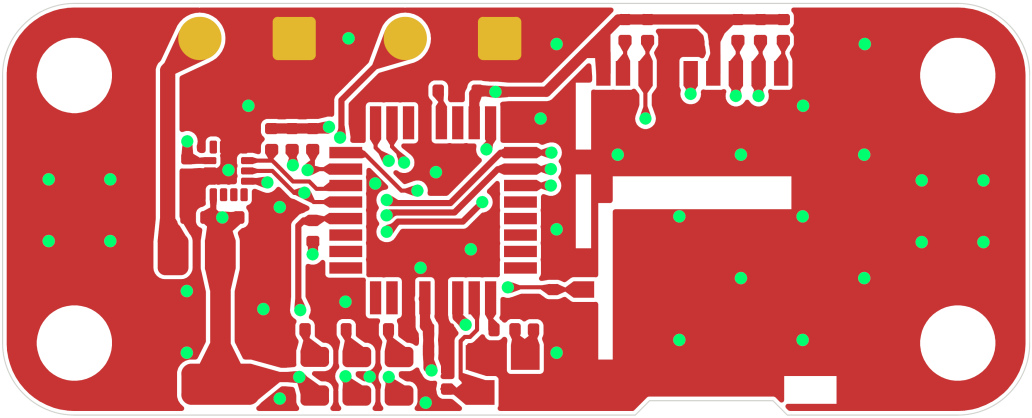

Figure 3: Cuivre du dessus

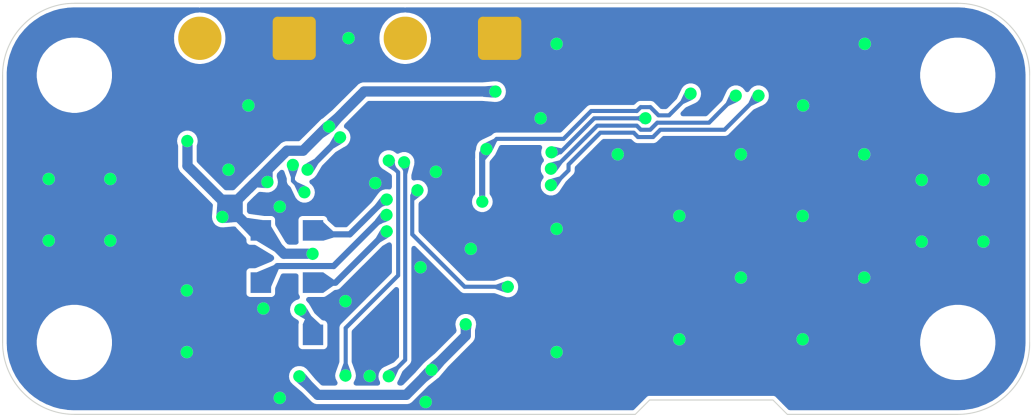

Figure 4: Cuivre du dessous

4.3 Placement des composants

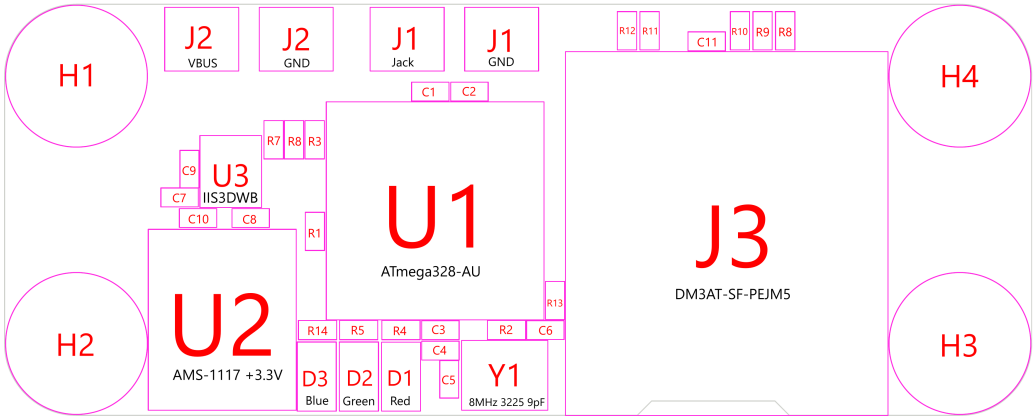

Figure 5: Placement au dessus

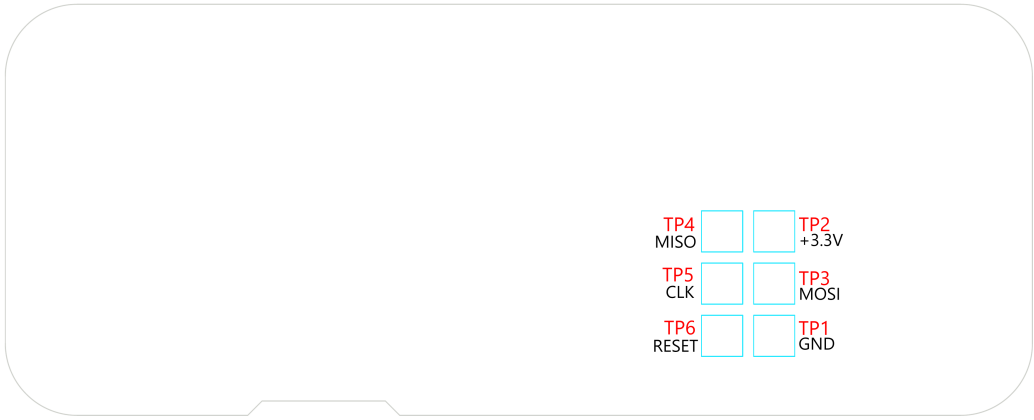

Figure 6: Placement au dessous

4.4 Visuels

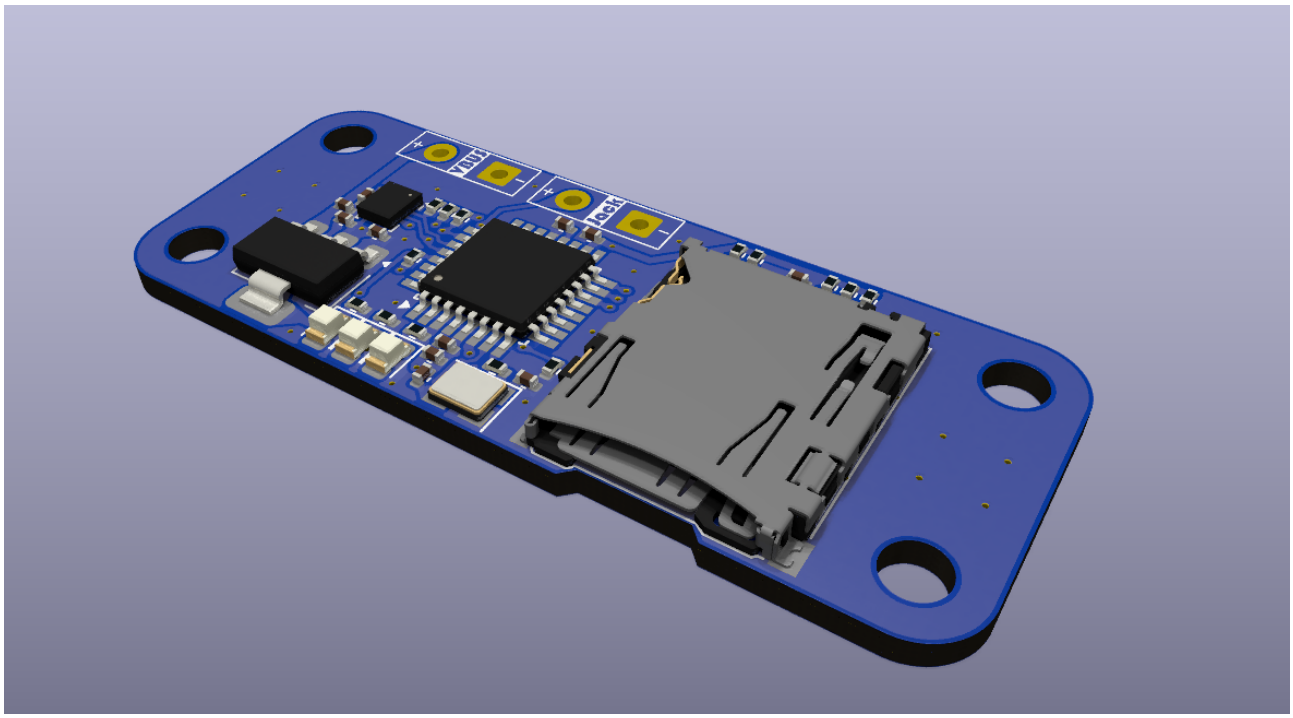

Figure 7: Aspect par dessus

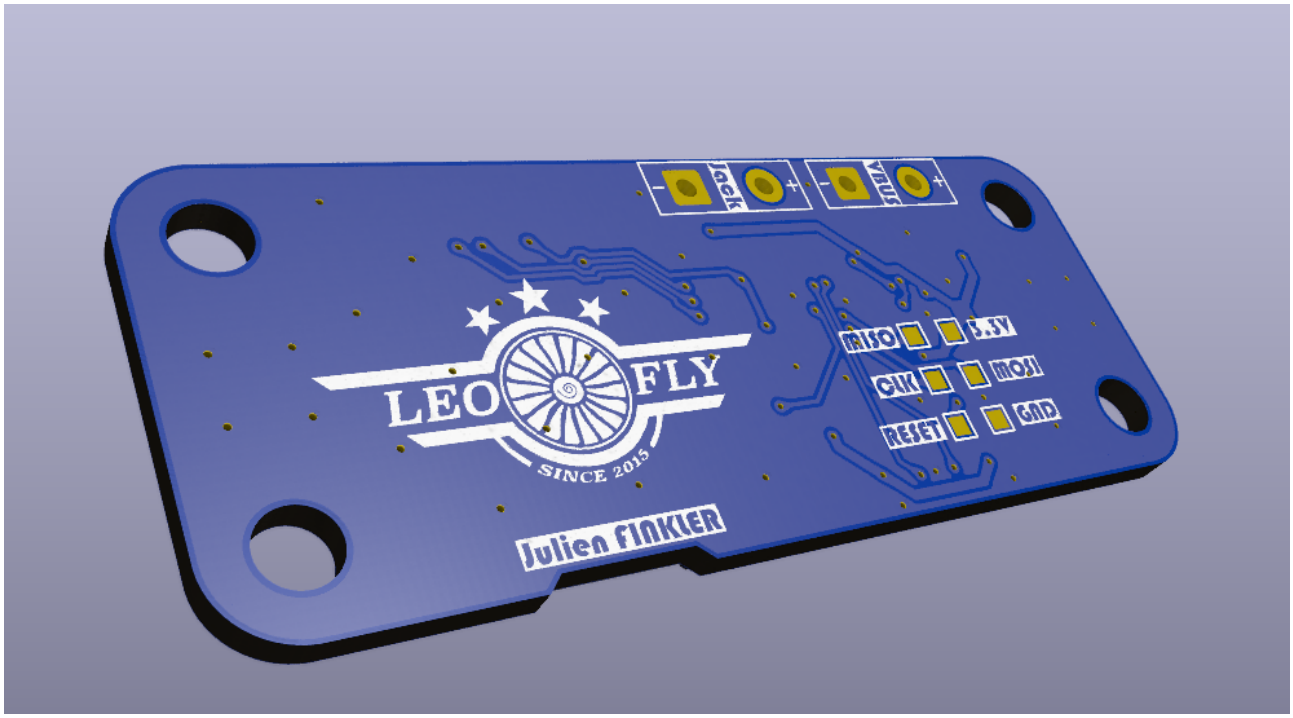

Figure 8: Aspect par dessous

4.5 BOM

Code	Composant	Rôle	Infos
U1	ATmega328p-AU	MCU	3.3V / 8MHz
U2	AMS1117-3.3	LDO	VBUS => +3.3V
U3	IIS3DWB	accéléromètre	I²C
J1, J2	01x02 Conn	Connecteurs	Jack / VBUS
J3	DM3AT-SF-PEJM5	Lecteur MicroSD	SPI
Y1	Crystal 4-pin	Quartz	8MHz 3225 9pF
H1, H2, H3, H4	Mounting-Hole	Supports	M3
D1, D2, D3	R/G/B LEDs	control LEDs	KT-0805
R14, R4, R5	Resistor	Resistance LED	10kΩ 0402
R2	Resistor	Resistance de stabilisation	2Ω 0402
R1, R3, R6, R7, R8, R9, R10, R11, R12	Resistor	Pull-up resistor	10kΩ 0402
R13	Resistor	Pull-down resistor	10kΩ 0402
C1	Capacitor	AREF filter	100nF 0402
C4	Capacitor	MCU Voltage filter	4.7μF 0402
C7, C8	Capacitor	LDO Voltage filter	10μF 0402
C2, C3, C9, C10, C11	Capacitor	Generic Voltage filter	100nF 0402
TP1, TP2, TP3, TP4, TP5, TP6	Test points	Bootloader	Test pad 1.0x1.0mm

5 Choix de la méthode de Programmation

5.1 Burn Bootloader

Sur tous Les nouveaux PCB et MCU il faut effectuer la procédure "burn bootloader" qui permet de programmer n'importe quelle carte grâce à l'IDE arduino. Ce processus nécéssite alors 6 pins accessible correspondants au SPI. On peut alors se servir de cette obligation de connecteurs et les utiliser pour la méthode arduino as ISP proposée par arduino eux mêmes.

5.2 Arduino as ISP

Cette procédure est disponible est adaptable sur l'IDE arduino.et ne nécessite qu'une arduino connecté par USB sur l'ordinateur sur lequel on programe et de relier les ports SPIs de l'arduino au bootloader de la carte.

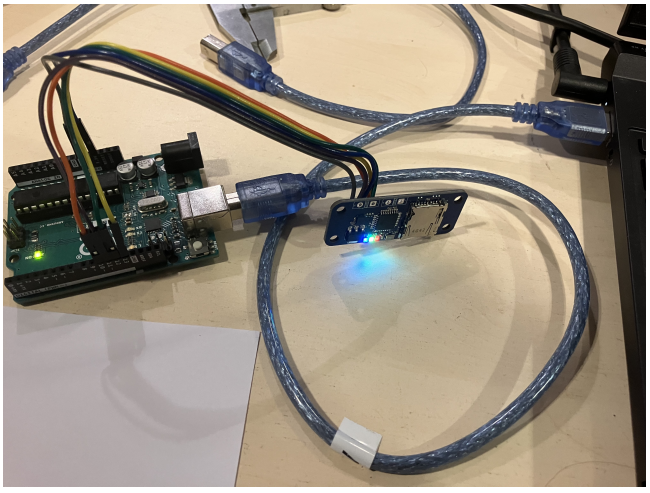

Figure 9: Processus de bootloading

6 Software

(Ce code est provisoire et comporte toujours quelques problèmes, il est amené à changer.)

6.1 Initialisations

```
#include <SPI.h>
#include <SD.h>
#include <Wire.h>

#define PIN_LED_R 14
#define PIN_LED_G 15
#define CS_PIN 10
#define DET_PIN 16

const uint8_t IIS3DWB_ADDRESS = 0x6A;
const uint8_t CRTL_1 = 0x10;
const uint8_t CRTL_6 = 0x15;
const uint8_t CRTL_7 = 0x16;
const uint8_t WAKE_UP_SRC = 0x1B;
const uint8_t OUT_Z_L = 0x2C;
const uint8_t OUT_Z_H = 0x2D;

File fichier;

void Wregister(uint8_t registre, uint8_t valeur) {
  Wire.beginTransmission(IIS3DWB_ADDRESS);
  Wire.write(registre);
  Wire.write(valeur);
  Wire.endTransmission();
}

byte Rregister(uint8_t registre) {
  Wire.beginTransmission(IIS3DWB_ADDRESS);
  Wire.write(registre);
  Wire.endTransmission();
  Wire.requestFrom(IIS3DWB_ADDRESS, 1, true);
  byte resultat;
  while(Wire.available()) {
    resultat = Wire.read();
  }
  return resultat;
}
```

6.2 Setup

```
void setup() {
  pinMode(DET_PIN, INPUT);
  pinMode(PIN_LED_R, OUTPUT);
  pinMode(PIN_LED_G, OUTPUT);

  // Attends qu'une carte SD soit insérer
  while(digitalRead(DET_PIN) != HIGH) {
    digitalWrite(PIN_LED_R, HIGH);
    digitalWrite(PIN_LED_G, HIGH);
    delay(50);
    digitalWrite(PIN_LED_R, LOW);
    digitalWrite(PIN_LED_G, LOW);
    delay(50);
  }
  delay(1000);

  // SETUP CARTE SD
  if (!SD.begin(CS_PIN)) {
    digitalWrite(PIN_LED_R, HIGH);
    while (1) {}
  }
  fichier = SD.open("logs.txt", FILE_WRITE);
  if (!fichier) {
    digitalWrite(PIN_LED_R, HIGH);
    while(1) {}
  }
  fichier.close();

  // Carte SD setup correctement
  digitalWrite(PIN_LED_R, HIGH);
  delay(100);
  digitalWrite(PIN_LED_R, LOW);


  // SETUP I²C
  byte error;
  Wire.begin();
  Wire.beginTransmission(IIS3DWB_ADDRESS);
  error = Wire.endTransmission();
  if (error != 0) {
    digitalWrite(PIN_LED_G, HIGH);
    while(1) {}
  }

  // écriture sur les registres
  fichier = SD.open("logs.txt", FILE_WRITE);

  // logs
  fichier.print("lecture du registre n°1 : ");
  fichier.print(Rregister(CRTL_1));
  fichier.print(" -> ");
```

```
  // endormissement et selection 16g
  Wregister(CRTL_1, 0x04);
  fichier.println(Rregister(CRTL_1));

  // logs
  fichier.print("lecture du registre n°2 : ");
  fichier.print(Rregister(CRTL_6));
  fichier.print(" -> ");

  // selection axe Z et no offset
  Wregister(CRTL_6, 0x03);
  fichier.println(Rregister(CRTL_6));

  // logs
  fichier.print("lecture du registre n°3 : ");
  fichier.print(Rregister(CRTL_1));
  fichier.print(" -> ");

  // réveil et 16g
  Wregister(CRTL_1, 0xA4);
  fichier.println(Rregister(CRTL_1));
  fichier.close();

  // I²C setup correctement
  digitalWrite(PIN_LED_G,HIGH);
  delay(100);
  digitalWrite(PIN_LED_G,LOW);
  fichier = SD.open("donnees.csv",FILE_WRITE);
  fichier.println("Temps;Low_Z;Low_Z_BIN;High_Z;High_Z_BIN;donnees");
  Wire.beginTransmission(IIS3DWB_ADDRESS);
  digitalWrite(PIN_LED_R,HIGH);
  delay(1000);
  digitalWrite(PIN_LED_R,LOW);
}
```

6.3 Loop

```
unsigned long debut = millis();
unsigned long actuel = millis();
bool compte = true;
bool first_loop = true;

void loop() {
  // put your main code here, to run repeatedly:

  // BLINK
  digitalWrite(PIN_LED_G,compte);
  compte = !compte;

  // first loop
  if (first_loop) {
    debut = millis();
    actuel = millis();
    first_loop = false;
  }

  actuel = millis();

  // timer 5sec
  if (actuel - debut > 5000 ) {
    fichier.close();
    Wire.endTransmission();
    while(1) {
      digitalWrite(PIN_LED_R,HIGH);
      digitalWrite(PIN_LED_G,HIGH);
      delay(50);
      digitalWrite(PIN_LED_R,LOW);
      digitalWrite(PIN_LED_G,LOW);
      delay(50);
    }

  // écriture
  } else {
      fichier.print(actuel - debut);
      byte Low = Rregister(OUT_Z_L);
      byte High = Rregister(OUT_Z_H);
      fichier.print(";");
      fichier.print(Low);
      fichier.print(";");
      fichier.print(Low, BIN);
      fichier.print(";");
      fichier.print(High);
      fichier.print(";");
      fichier.print(High, BIN);
      fichier.print(";");
      fichier.println(High*256 + Low);
  }
}
```

7 Evolution et Changements de design

- ESP32 avec flash intégrée
- Carte SD seulement pour écrire les données post-vol
- Capteur IIS3DWB en SPI
- Utilisation du FIFO du capteur

8 Sources et Datasheets

- Datasheet ATmega328p-AU :
  `https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P Datasheet.pdf`
- Datasheet IIS3DWB :
  `https://www.st.com/resource/en/datasheet/iis3dwb.pdf&ved=2ahUKEwjh-aGy97OLAxXEWEEAHUD6JO4QFnoECB4Q usg=AOvVaw2oWmJWRapUXw6F-mjiMNbC`
- Datasheet AMS1117-3.3V :
  `http://www.advanced-monolithic.com/pdf/ds1117.pdf`
- Documentation Arduino as ISP :
  `https://docs.arduino.cc/built-in-examples/arduino-isp/ArduinoISP/`

Mini-Sequencer

Design Report

ESILV Pôle Léonard de Vinci

Contents

1 Introduction

Dans le cadre de la campagne de lancement CSPACE, la conception du séquenceur embarqué pour notre modèle de fusée doit respecter un certain nombre d'exigences techniques imposées par le cahier des charges. Ces contraintes assurent la compatibilité et la sécurité des systèmes électroniques embarqués lors du lancement.

Les exigences à respecter sont les suivantes :

- **SEQ1 :** Aucune liaison électrique, autre que la masse électrique, n'est autorisée entre les séquenceurs et entre chaque séquenceur et tout autre système électrique embarqué.
- **SEQ2 :** Le séquenceur doit avoir une autonomie d'au moins une heure et son activation doit se faire sur la rampe de lancement.
- **SEQ3 :** Le séquenceur doit disposer de la puissance nécessaire pour déclencher le mécanisme de séparation.
- **SEQ4 :** Une signalisation claire doit être mise en place pour indiquer explicitement trois états distincts :
  - Séquenceur sous tension ou hors tension.
  - Séquenceur actif (fusée ayant décollé) ou inactif (fusée en attente de décollage).
  - Actionneur actif (séparation commandée) ou inactif (séparation non commandée).

En plus de ces exigences obligatoires, nous avons défini nos propres critères de conception afin d'assurer la modularité et la simplicité d'utilisation du séquenceur. Ainsi, notre séquenceur est conçu pour :

- Être capable d'opérer un servo ou un MOSFET de puissance (pour les systèmes de séparation "type Nichrome") de manière simple et modulaire.
- Fournir une alimentation 5V via un régulateur LDO embarqué, permettant ainsi l'utilisation du servo lors des tests au sol ou en conditions de faible puissance.
- Offrir un mécanisme simple pour passer d'une alimentation par le LDO à une alimentation par une batterie externe distincte de celle alimentant le LDO et le reste de la carte.

Cette approche garantit un séquenceur flexible et efficace, tout en respectant les contraintes imposées par la campagne de lancement CSPACE.

2 Hardware Design

2.1 Microcontrôleur et circuits de support

2.1.1 Choix du microcontrôleur Atmega328P

Le choix du microcontrôleur Atmega328P-AU repose sur plusieurs critères essentiels pour notre séquenceur :

- **Documentation étendue :** L'Atmega328P bénéficie d'une large documentation, en particulier grâce à son utilisation dans de nombreux produits, notamment les cartes *Arduino Nano*.
- **Simplicité du circuit :** Son schéma de mise en œuvre est relativement simple, nécessitant peu de composants passifs, ce qui réduit la complexité globale du PCB.
- **Tension d'alimentation unique :** Dans notre cas, il fonctionne sous une tension unique de 5 V, ce qui simplifie l'intégration avec les autres composants de la carte.
- **Facilité de programmation et d'interfaçage :** Il peut être programmé et communiqué facilement via une liaison *UART-USB*, permettant un développement et un débogage simplifiés.
- **Format compact :** Le boîtier TQFP-32 permet une intégration efficace sur notre PCB tout en optimisant l'espace disponible.

2.1.2 Schéma du circuit et composants associés

Le schéma du circuit basé sur l'Atmega328P-AU est présenté ci-dessous :

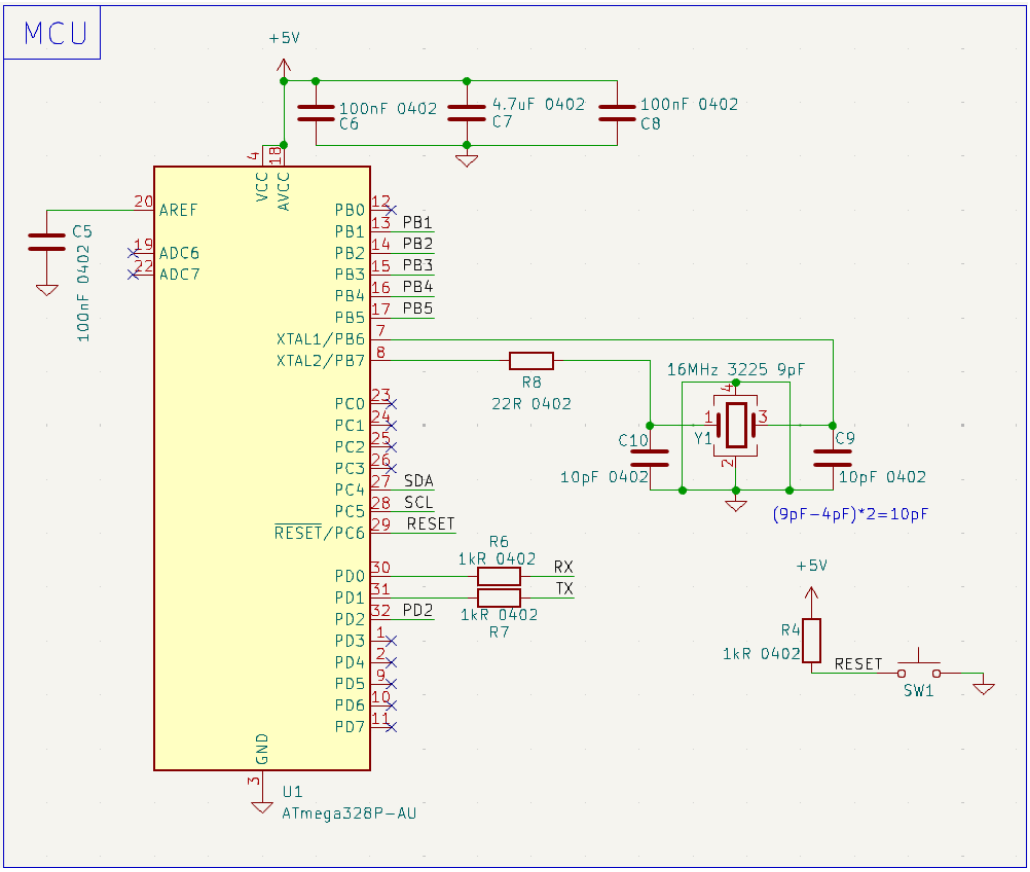

Figure 1: Schéma du microcontrôleur Atmega328P-AU et de ses composants associés.

Ce circuit comprend plusieurs éléments essentiels pour assurer le bon fonctionnement du microcontrôleur :

2.1.3 Condensateurs de découplage

Les condensateurs de découplage jouent un rôle clé dans la stabilisation de l'alimentation du microcontrôleur :

- **Proximité :** Les condensateurs C5, C6 et C8 sont de 100 nF au format 0402 et sont placés à proximité des broches d'alimentation du microcontrôleur pour réduire le bruit haute fréquence et assurer un bon découplage local.
- **Réserve d'énergie :** C7 est un condensateur de 4.7 µF au format 0402, utilisé pour fournir une réserve d'énergie en cas de variations de charge plus importantes, garantissant ainsi une alimentation stable.

2.1.4 Horloge externe 16 MHz

Nous avons choisi une fréquence de 16 MHz, qui représente la limite supérieure de fonctionnement de l'Atmega328P-AU à 5 V, garantissant ainsi des performances optimales.

L'oscillateur externe (Y1) est un cristal au format 3225 avec une capacité de charge de 9 pF. Les condensateurs associés (C9 et C10) ont été calculés selon la formule trouvée dans la documentation du fabricant :

$$C_{load} = (C_{crystal} - C_{stray}) \times 2 \tag{1}$$

En prenant $C_{crystal} = 9\,\text{pF}$ et une capacité parasite estimée de $C_{stray} = 4\,\text{pF}$, nous obtenons :

$$C_{load} = (9 - 4) \times 2 = 10\,\text{pF} \tag{2}$$

Un résistor en série de 22 Ω est également utilisé pour limiter les harmoniques de l'oscillateur et améliorer la stabilité du signal d'horloge.

2.1.5 Broches ISP pour bootloading et programmation

Les broches ISP utilisées pour la programmation de l'Atmega328P-AU sont PB3 (MISO), PB4 (MOSI) et PB5 (SCK). Les connecteurs associés sont référencés dans le schéma global (voir Figure **??**).

2.1.6 Autres broches et fonctionnalités secondaires

Le circuit expose plusieurs broches pour différentes fonctionnalités, incluant :

- **Communication I2C :** Les broches SDA et SCL sont utilisées pour l'interfaçage avec d'autres périphériques et sont connectées à des headers visibles dans la section dédiée à l'interface I2C.
- **Entrée PWM :** La sortie PWM principale est PB3 et est expliquée dans la section dédiée à la broche servo-moteur.
- **MOSFET de puissance :** La commande du MOSFET est assurée par PB2 et est détaillée dans sa propre section.
- **Buzzer :** La broche PB1 est utilisée pour piloter un buzzer, avec des détails donnés dans la section correspondante.
- **Liaison UART :** La communication UART-USB est expliquée dans la section 2.3
- **Détection de lancement (Jack) :** La broche PD2 est utilisée pour la détection du lancement et est expliquée dans sa propre section.
- **GPIO auxiliaire :** Une broche supplémentaire, PB5, est exposée pour des besoins futurs et est visible dans le schéma global (voir Figure 2).

2.1.7 Bouton de réinitialisation

Le bouton de réinitialisation utilisé est un **PTS636 SK25J SMTR LFS**. Ce choix repose sur plusieurs critères :

- **Format compact :** Son empreinte réduite permet une intégration facile sur le PCB.
- **Robustesse :** Avec une force d'actionnement de 250 gf ± 50 gf, il est suffisamment résistant aux vibrations et accélérations du lancement.

En prenant en compte une masse estimée du bouton de 5 mg, l'accélération requise pour provoquer un appui involontaire est donnée par :

$$a = \frac{F}{m} \tag{3}$$

Ce qui donne une plage d'accélération estimée entre 40 000 g et 60 000 g, bien au-delà des forces attendues en vol, garantissant ainsi l'absence de déclenchements intempestifs.

Ce schéma assure une configuration complète et optimisée pour les besoins du séquenceur, tout en restant simple et modulaire.

2.1.8 Application du circuit

Avec toutes ces considérations voici l'application du circuit du microcontrôleur et de ses composants passifs.

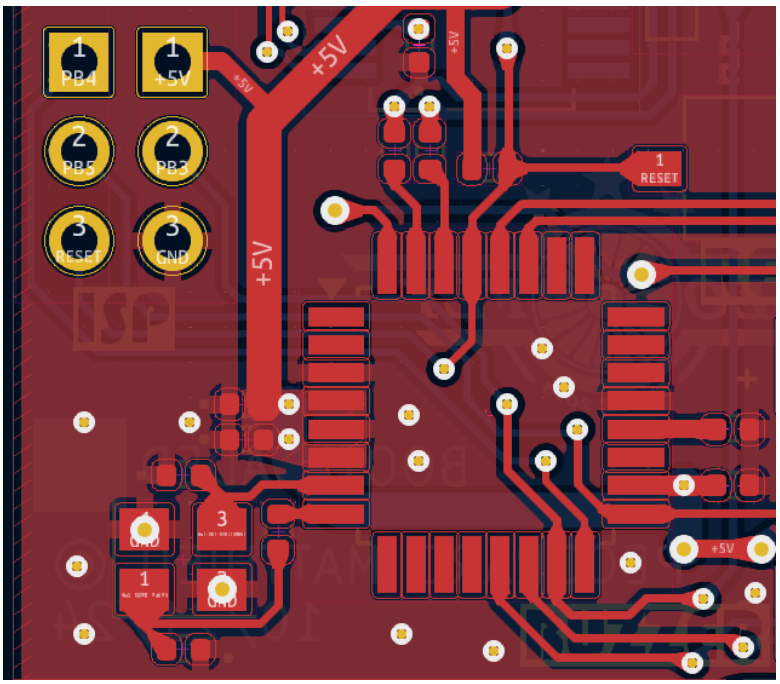

Figure 2: Application du circuit du MCU

2.2 Alimentation du système

L'alimentation du séquenceur repose sur un régulateur linéaire **LM1084-5.0** qui fournit une tension stabilisée de 5 V. Ce régulateur de tension à faible chute (*LDO*) permet une alimentation fiable et stable pour l'ensemble des composants du circuit, y compris le microcontrôleur et potentiellement un servo si celui-ci est connecté à la carte.

2.2.1 Caractéristiques et justification du choix du régulateur

Le **LM1084-5.0** est capable de fournir un courant maximal de 5 A, ce qui est largement suffisant pour alimenter le microcontrôleur et les périphériques associés, y compris un servo moteur de faible à moyenne puissance. L'utilisation de ce régulateur garantit une alimentation stable même sous des charges variables.

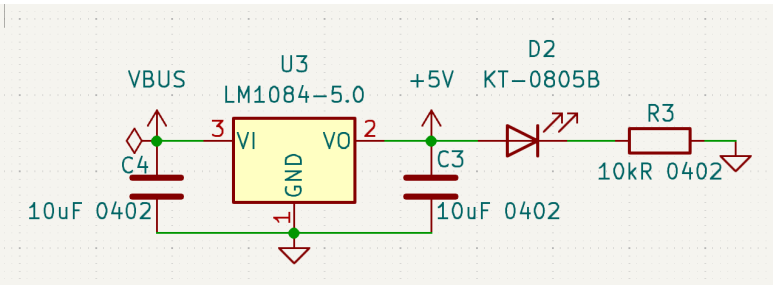

Figure 3: Schéma du circuit d'alimentation basé sur le LM1084-5.0.

2.2.2 Entrée d'alimentation et connecteur

L'entrée d'alimentation est réalisée via un connecteur **XT30**, capable de supporter un courant maximal de 30 A. Ce choix est motivé par sa robustesse mécanique et électrique, ce qui le rend idéal pour un environnement soumis à des vibrations et des chocs importants, comme celui d'un modèle de fusée.

Le **LM1084-5.0** supporte une plage de tension d'entrée typique comprise entre 6.5 V et 25 V, ce qui permet d'utiliser diverses sources d'alimentation telles que :

- Une batterie alcaline de 9 V.
- Une batterie LiPo 2S (7.4 V).
- Une alimentation 12V pour des tests au sol.

2.2.3 Composants supplémentaires

Conformément aux recommandations de la fiche technique du **LM1084-5.0**, des condensateurs de découplage de 10 µF sont placés aux bornes du régulateur pour stabiliser la tension et filtrer les bruits électriques. De plus, une LED est ajoutée en sortie pour indiquer la présence de l'alimentation régulée.

2.2.4 Application du circuit

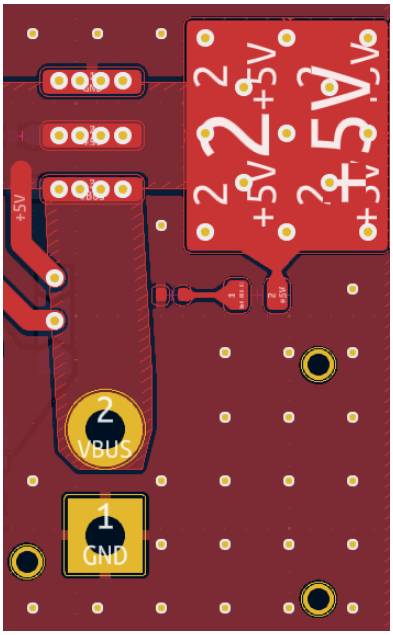

Figure 4: Application du circuit du LDO

2.3 Interface USB-UART

L'interface USB-UART permet de connecter le séquenceur à un ordinateur ou tout autre périphérique compatible, facilitant ainsi la programmation, la configuration et le débogage du système.

2.3.1 Choix du connecteur USB Type-C

Le standard **USB Type-C** a été retenu pour cette interface en raison de sa large adoption et de ses nombreux avantages techniques. Contrairement aux connecteurs USB plus anciens, l'USB-C :

- Est réversible, facilitant ainsi l'insertion.
- Prend en charge différentes tensions d'entrée via les broches **CC1** et **CC2**, qui permettent d'indiquer au dispositif hôte la puissance requise.
- Est mécaniquement robuste et conçu pour supporter un nombre élevé de cycles d'insertion.
- Assure une compatibilité avec divers périphériques grâce à son adoption généralisée.

Les broches **D+** et **D-** du connecteur USB Type-C sont utilisées pour la communication de données USB 2.0, assurant une compatibilité avec les interfaces UART standards.

2.3.2 Utilisation du convertisseur USB-UART FT232RL

Le circuit utilise le **FT232RL**, un convertisseur USB vers UART largement utilisé, notamment dans les cartes *Arduino Nano*. Ce choix est justifié par :

- Sa documentation complète et bien référencée.
- Son support natif par de nombreux systèmes d'exploitation sans nécessiter de pilotes supplémentaires.
- Son intégration simple avec les microcontrôleurs nécessitant une liaison série pour la programmation et la communication.

2.3.3 Présentation du circuit

Le schéma du circuit USB-UART est présenté ci-dessous :

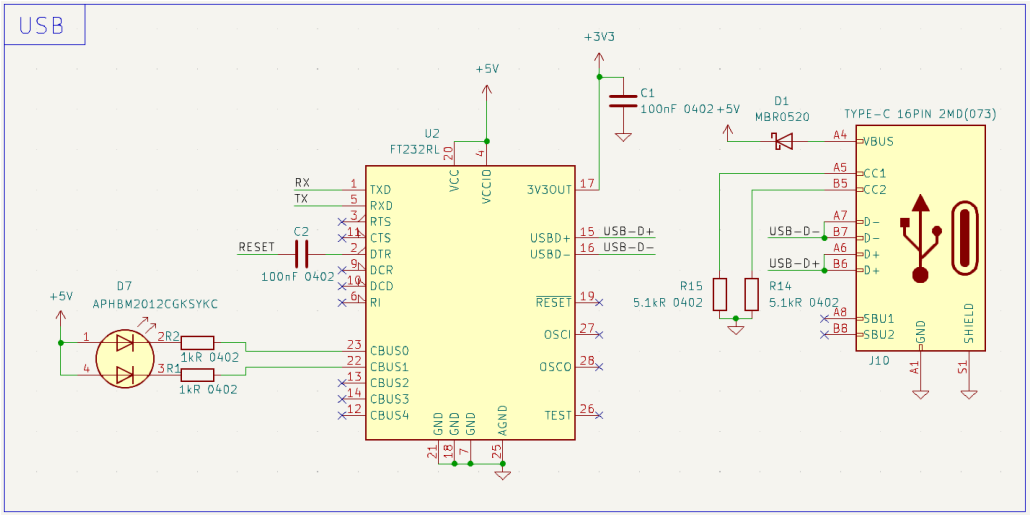

Figure 5: Schéma de l'interface USB-UART basé sur le FT232RL.

Le circuit intègre plusieurs éléments clés :

- **Convertisseur FT232RL** : Assure la conversion des signaux USB en UART.
- **Connecteur USB Type-C** : Interface physique avec l'ordinateur.
- **Résistances pull-down de 5.1kΩ** sur les broches CC1 et CC2 : Indiquent au dispositif hôte qu'il s'agit d'un périphérique passif alimenté en 5V.
- **Diode de protection** : Protège contre d'éventuelles inversions de tension sur VBUS.

2.3.4 Indicateurs d'activité TX/RX

Les broches **CBUS0** et **CBUS1** du FT232RL sont configurées comme sorties drain pour indiquer l'activité de transmission et de réception des données UART. Ces sorties sont connectées à une LED bicolore permettant de visualiser les échanges de données :

- Lorsque CBUS0 est activé, il indique une transmission en cours (TX).
- Lorsque CBUS1 est activé, il indique une réception en cours (RX).

Cela permet une indication visuelle efficace de l'activité de communication sans interférer avec la transmission des données.

2.3.5 Application du circuit

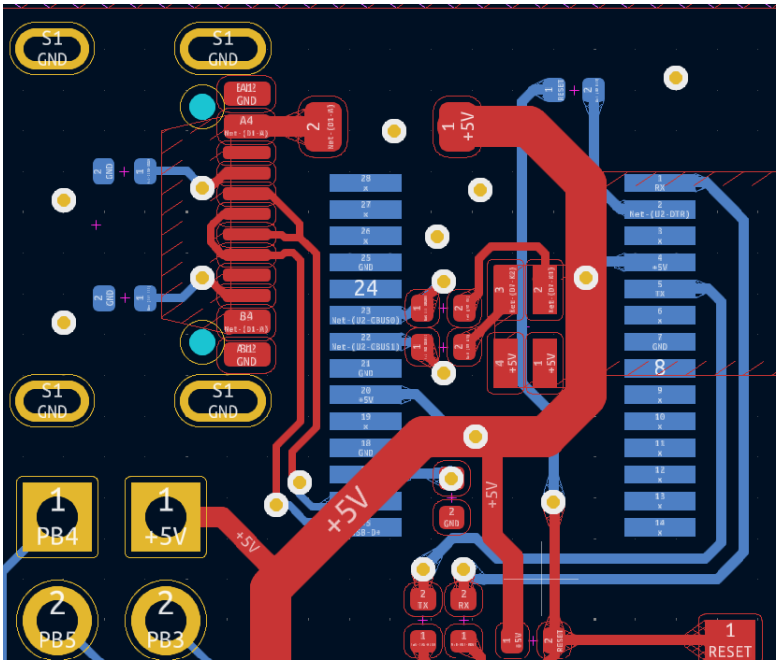

Figure 6: Application du circuit sur le pcb

3 Software

3.1 Architecture générale

La programmation de la carte MiniSequenceur s'effectue à l'aide d'un code universel développé en interne, permettant à des fusées de tout gabarit et de tout type de séparation de fonctionner avec cette carte.
L'architecture du code repose sur une simple machine à états, dont les définitions sont présentées à la Figure 7 et détaillées dans la section consacrée à `StateHandler.h`.

```
1    enum RocketState {
2        NONE,
3        IDLE,
4        LAUNCH_DETECTED,
5        SEPARATION,
6        POST_SEPARATION
7    };
```

Figure 7: Définition des différents états de la machine à états (voir section `StateHandler.h`).

Selon les différentes phases du vol, la fusée se trouve dans l'un de ces états. Lors d'événements programmés, celle-ci change d'état : le comparateur asynchrone associé au nouvel état exécute alors les actions correspondantes jusqu'au prochain changement d'état.

Pour mettre en œuvre cette architecture, le code est divisé en différents fichiers `.cpp` et `.h` associés, chacun ayant pour rôle de gérer une fonctionnalité spécifique de l'architecture.

Voici leurs fonctions respectives, détaillées dans les sections correspondantes de ce document :

1. `Config.h` (modifiable) : permet de configurer les différents paramètres du vol (associé à `Config.cpp`).
2. `Buzzer.cpp` (associé à `Buzzer.h`) : assure la gestion asynchrone du PWM du buzzer embarqué.
3. `IS31FL3193.cpp` (associé à `IS31FL3193.h`) : bibliothèque I2C sur mesure pour la carte accessoire de gestion des LED.
4. `StateHandler.cpp` (associé à `StateHandler.h`) : gère les états et les actions associées.
5. `Main.ino` : boucle principale (pas de `setup`, celui-ci est exécuté à la mise sous tension en état `NONE` par `StateHandler.cpp`).

3.2 Config

3.2.1 Config.h

```
#ifndef CONFIG_H
#define CONFIG_H

#include <Arduino.h>

/*
    ============================================
    CONFIGURATION HEADER FILE (Config.h)
    ============================================

    This file contains all configuration settings
    for the Mini Sequencer Card.

    --------------------------------------------
    FILE CONTENTS
    --------------------------------------------

    - CONFIGURABLE FLIGHT VARIABLES
    - PIN DEFINITIONS
    - CONSTANTS

    --------------------------------------------
    AUTHOR: PECORARO Matthieu
    DATE:   17/01/25
    ============================================
*/

/*
    ============================================
    CONFIGURABLE FLIGHT VARIABLES
    ============================================
*/

// Timing Configuration (in milliseconds)
#define TIME_TO_SEPARATION 13000  // Delay before separation
#define SERIAL_INTERVAL 500 // Delay between serial communication attemps (USB Debug Only)

// Separation Configuration
#define SEPARATION_METHOD MOSFET_SEPARATION // Type of separation (SERVO_SEPARATION / MOSFET_SEPARATION)
#define TIME_TO_POST_SEPARATION 5000  // Time after separation before moving to next state

// Servo Separation (only applicable if SEPARATION_METHOD = SERVO_SEPARATION)
#define SERVO_POSITION_INIT 40 // Initial Servo position
#define SERVO_POSITION_SEPARATION 100 // Servo position at separation
#define RESET_SERVO_POST_SEPARATION true

/*
    ============================================
    PIN DEFINITIONS
    ============================================
*/

// Outputs
#define PIN_BUZZER 9
```

```
#define PIN_MOSFET_CTRL 13
// Inputs
#define PIN_JACK_DETECT 2
#define PIN_AUX_OUTPUT 13
#define PIN_SERVO_CTRL 11

/*
    ========================================
    CONSTANTS
    ========================================
*/

// Jack Pin States
#define PLUGGED HIGH
#define UNPLUGGED LOW

// Separation Methods
#define SERVO_SEPARATION 0
#define MOSFET_SEPARATION 1

void initPinModes();

#endif // CONFIG_H
```

3.2.2 Config.cpp

```
#include "Config.h"

void initPinModes() {
  pinMode(PIN_BUZZER, OUTPUT);
  pinMode(PIN_MOSFET_CTRL, OUTPUT);
  pinMode(PIN_JACK_DETECT, INPUT);
  pinMode(PIN_AUX_OUTPUT, OUTPUT);
  pinMode(PIN_SERVO_CTRL, OUTPUT);
  digitalWrite(PIN_MOSFET_CTRL, LOW);
}
```

3.3 Buzzer

3.3.1 Buzzer.h

```h
#ifndef BUZZER_H
#define BUZZER_H

#include "Config.h"
#include <Arduino.h>

// Structure to manage buzzer settings and states
struct BuzzerSettings {
    unsigned long lastBuzzerTime = 0;
    int buzzerState = 0;        // 0: Off, 1: On during short interval, 2: Long interval
    int repeatCount = 0;
    int maxRepeats = 0;
    unsigned long onInterval = 0;
    unsigned long offShortInterval = 0;
    unsigned long offLongInterval = 0;
    unsigned int buzzerFrequency = 0;
};

// Function Declarations
void startBuzzerSequence(int frequency, unsigned long onTime, unsigned long shortOff, unsigned long longOff, int repeats);
void updateBuzzerSequence();

#endif // BUZZER_H
```

3.3.2 Buzzer.cpp

```cpp
#include "Buzzer.h"

BuzzerSettings buzzer;  // Create an instance of BuzzerSettings

// Function to initialize the buzzer sequence
void startBuzzerSequence(int frequency, unsigned long onTime, unsigned long shortOff, unsigned long longOff, int repeats) {
  buzzer.buzzerFrequency = frequency;
  buzzer.onInterval = onTime;
  buzzer.offShortInterval = shortOff;
  buzzer.offLongInterval = longOff;
  buzzer.maxRepeats = repeats;
  buzzer.repeatCount = 0;
  buzzer.buzzerState = 1;            // Start with the first ON interval
  buzzer.lastBuzzerTime = millis();  // Initialize timing
}

// Function to update the buzzer state (call this in the loop)
void updateBuzzerSequence() {
  unsigned long currentTime = millis();

  switch (buzzer.buzzerState) {
    case 0:  // Idle (Off)
      noTone(PIN_BUZZER);
      break;

    case 1:  // Tone ON
      if (currentTime - buzzer.lastBuzzerTime >= buzzer.onInterval) {
```

```
28          noTone(PIN_BUZZER);  // Stop the tone
29          buzzer.lastBuzzerTime = currentTime;
30          buzzer.buzzerState = 2;  // Move to short OFF interval
31        } else {
32          tone(PIN_BUZZER, buzzer.buzzerFrequency);  // Play the tone
33        }
34        break;
35
36      case 2:  // Short OFF interval
37        if (currentTime - buzzer.lastBuzzerTime >= buzzer.offShortInterval) {
38          buzzer.lastBuzzerTime = currentTime;
39          if (buzzer.repeatCount < buzzer.maxRepeats - 1) {
40            buzzer.repeatCount++;
41            buzzer.buzzerState = 1;  // Repeat the tone
42          } else {
43            buzzer.buzzerState = 3;  // Move to long OFF interval
44          }
45        }
46        break;
47
48      case 3:  // Long OFF interval
49        if (currentTime - buzzer.lastBuzzerTime >= buzzer.offLongInterval) {
50          buzzer.lastBuzzerTime = currentTime;
51          buzzer.repeatCount = 0;  // Reset repeat count
52          buzzer.buzzerState = 1;  // Start the next cycle
53        }
54        break;
55    }
56  }
```

3.4 IS31FL3193

3.4.1 IS31FL3193.h

```h
#ifndef IS31FL3193_H
#define IS31FL3193_H

#include <Wire.h>
#include <Arduino.h>

// I2C Address for IS31FL3193
#define IS31FL3193_ADDRESS 0x68

// Register Definitions
#define IS31FL3193_REG_SHUTDOWN 0x00
#define IS31FL3193_REG_PWM_R 0x04
#define IS31FL3193_REG_PWM_G 0x05
#define IS31FL3193_REG_PWM_B 0x06
#define IS31FL3193_REG_LED_CONTROL 0x1D
#define IS31FL3193_REG_UPDATE 0x07

// Function Declarations
void initLedDriver();
void setRgbColor(uint8_t red, uint8_t green, uint8_t blue);
void debugLedRegisters();

#endif // IS31FL3193_H
```

3.4.2 IS31FL3193.cpp

```cpp
#include "IS31FL3193.h"

void initLedDriver() {
  Serial.println("Initializing IS31FL3193...");

  // Enable the chip (exit shutdown mode)
  Wire.beginTransmission(IS31FL3193_ADDRESS);
  Wire.write(IS31FL3193_REG_SHUTDOWN);
  Wire.write(0x20);  // Enable channels
  if (Wire.endTransmission() == 0) {
    Serial.println("Device enabled.");
  } else {
    Serial.println("Failed to enable the device. Check connections.");
    return;
  }

  // Enable RGB channels (OUT1, OUT2, OUT3)
  Wire.beginTransmission(IS31FL3193_ADDRESS);
  Wire.write(IS31FL3193_REG_LED_CONTROL);
  Wire.write(0x07);  // Enable all three channels
  if (Wire.endTransmission() == 0) {
    Serial.println("Channels enabled.");
  } else {
    Serial.println("Failed to enable channels. Check connections.");
  }

  Wire.beginTransmission(IS31FL3193_ADDRESS);
  Wire.write(IS31FL3193_REG_UPDATE);
  Wire.write(0x00);  // Enable channels
}
```

```
31
32 // Function to set RGB colors
33 void setRgbColor(uint8_t red, uint8_t green, uint8_t blue) {
34   Wire.beginTransmission(IS31FL3193_ADDRESS);
35   Wire.write(IS31FL3193_REG_PWM_R);  // Start at the Red PWM register
36   Wire.write(red);         // Red brightness
37   Wire.write(green);       // Green brightness
38   Wire.write(blue);        // Blue brightness
39   if (Wire.endTransmission() != 0) {
40     Serial.println("Failed to set RGB values. Check I2C.");
41     return;
42   }
43
44   // Update the PWM registers
45   Wire.beginTransmission(IS31FL3193_ADDRESS);
46   Wire.write(IS31FL3193_REG_UPDATE);
47   Wire.write(0x00);  // Trigger an update
48   if (Wire.endTransmission() != 0) {
49     Serial.println("Failed to update RGB values.");
50   }
51 }
52
53 void debugLedRegisters() {
54     Serial.println("Reading LED driver register values...");
55     for (uint8_t reg = 0x00; reg <= 0x1D; reg++) {
56         Wire.beginTransmission(IS31FL3193_ADDRESS);
57         Wire.write(reg);
58         Wire.endTransmission();
59         Wire.requestFrom(IS31FL3193_ADDRESS, 1);
60         if (Wire.available()) {
61             uint8_t value = Wire.read();
62             Serial.print("Register 0x");
63             Serial.print(reg, HEX);
64             Serial.print(": 0x");
65             Serial.println(value, HEX);
66         }
67     }
68 }
```

3.5 StateHandler

3.5.1 StateHandler.h

```h
#ifndef STATEHANDLER_H
#define STATEHANDLER_H

#include <Arduino.h>
#include <Servo.h>
#include "Config.h"
#include "IS31FL3193.h"
#include "Buzzer.h"

enum RocketState {
    NONE,
    IDLE,
    LAUNCH_DETECTED,
    SEPARATION,
    POST_SEPARATION
};

// Function Declarations
void initializeSystem();

void updateState();
void handleStateChange();
void setStateColor(RocketState state);
void triggerSeparation();
void triggerPostSeparation();
void sendStateOverSerial(RocketState state);
void updateSerialCommunication();

#endif // STATEHANDLER_H
```

3.5.2 StateHandler.cpp

```cpp
#include "StateHandler.h"

RocketState previousState = NONE;  // Track the previous state to detect state changes
RocketState currentState = NONE;  // Start in the NONE state -> Initialize

Servo servo;
long stateStartTime = 0;

void initializeSystem(){
  Wire.begin();

  Serial.begin(9600);
  servo.attach(PIN_SERVO_CTRL);
  servo.write(SERVO_POSITION_INIT);
  initLedDriver();
  initPinModes();
  digitalWrite(PIN_MOSFET_CTRL, LOW);
}

void updateState() {
  switch (currentState) {
    case NONE:
      currentState = IDLE;
```

```
      stateStartTime = millis();
      break;

    case IDLE:
      // Check for launch detection (jack pin unplugged)
      if (digitalRead(PIN_JACK_DETECT) == UNPLUGGED & previousState != NONE) {
        currentState = LAUNCH_DETECTED;
        sendStateOverSerial(currentState);
        stateStartTime = millis();
      }
      break;

    case LAUNCH_DETECTED:
      // Wait for the predefined delay
      if (millis() - stateStartTime > TIME_TO_SEPARATION) {
        currentState = SEPARATION;
        sendStateOverSerial(currentState);
        stateStartTime = millis();
      }
      break;

    case SEPARATION:
      if (millis() - stateStartTime > TIME_TO_POST_SEPARATION) {
        currentState = POST_SEPARATION;
        sendStateOverSerial(currentState);
        stateStartTime = millis();
        break;
      }
      break;

    case POST_SEPARATION:
      // No change untill RESET
      break;
  }
}

// Function to handle state-based buzzer activation
void handleStateChange() {
  // Check if the state has changed
  if (currentState != previousState) {
    // State has changed, activate the buzzer with specific intervals for the new
        state
    switch (currentState) {
      case IDLE: // System startup
        initializeSystem(); // Initiliaze system
        startBuzzerSequence(1000, 100, 300, 5000, 2);  // Short beep for IDLE
        setStateColor(currentState);
        break;

      case LAUNCH_DETECTED:
        startBuzzerSequence(1000, 100, 100, 200, 3);  // Medium beep for launch
            detected
        setStateColor(currentState);
        break;

      case SEPARATION:
        triggerSeparation();
        startBuzzerSequence(1000, 1000, 100, 100, 4);  // Long beep for separation
        setStateColor(currentState);
        break;

```

```
      case POST_SEPARATION:
        triggerPostSeparation();
        startBuzzerSequence(500, 1000, 200, 5000, 2);  // Different pattern for post-separation
        setStateColor(currentState);
        break;

      default:
        // Default action if state is unknown
        noTone(PIN_BUZZER);
        setStateColor(currentState);
        break;
    }
    // Update the previous state to the current state
    previousState = currentState;
  }
}

// Function to set state-based colors
void setStateColor(RocketState state) {
  switch (state) {
    case IDLE:
      setRgbColor(0, 0, 255);  // Green for IDLE
      break;

    case LAUNCH_DETECTED:
      setRgbColor(0, 255, 255);  // Yellow for launch detected
      break;

    case SEPARATION:
      setRgbColor(255, 255, 255);  // Red for separation
      break;

    case POST_SEPARATION:
      setRgbColor(255, 0, 0);  // Pure Red for post-separation
      break;

    default:
      setRgbColor(0, 0, 0);  // Off for unknown state
      break;
  }
}

void triggerSeparation(){
  switch (SEPARATION_METHOD) {
    case SERVO_SEPARATION :
      servo.write(SERVO_POSITION_SEPARATION);
      break;
    case MOSFET_SEPARATION :
      digitalWrite(PIN_MOSFET_CTRL, HIGH);
      break;
  }
}

void triggerPostSeparation(){
  switch (SEPARATION_METHOD) {
    case SERVO_SEPARATION :
      if (RESET_SERVO_POST_SEPARATION){
        servo.write(SERVO_POSITION_INIT); // Reinitialize Servo if configured
      }
      break;
```

```
    case MOSFET_SEPARATION :
      digitalWrite(PIN_MOSFET_CTRL, LOW);
      break;
  }
}

// Variables for managing serial communication timing
long lastSerialTime = 0;
// Function to send the current state and important info over Serial
void sendStateOverSerial(RocketState state) {
  // Get the current state as a string
  const char* stateName;
  switch (state) {
    case IDLE:
      stateName = "IDLE";
      break;
    case LAUNCH_DETECTED:
      stateName = "LAUNCH_DETECTED";
      break;
    case SEPARATION:
      stateName = "SEPARATION";
      break;
    case POST_SEPARATION:
      stateName = "POST_SEPARATION";
      break;
    default:
      stateName = "UNKNOWN";
      break;
  }

  // Calculate time into launch if applicable
  long timeIntoLaunch = millis() - stateStartTime;

  // Print the state and relevant details
  Serial.print("State: ");
  Serial.println(stateName);
  // Jack state
  Serial.print("Jack State: ");
  Serial.println(digitalRead(PIN_JACK_DETECT) == PLUGGED ? "PLUGGED" : "UNPLUGGED");
  // Time into launch
  Serial.print("Time Into State (ms): ");
  Serial.println(timeIntoLaunch);
  // Time into launch
  Serial.print("Millis (ms): ");
  Serial.println(millis());
  // Time into launch
  Serial.print("State Start (ms): ");
  Serial.println(stateStartTime);

  Serial.println();
}
// Function to update serial communication asynchronously
void updateSerialCommunication() {
  if (millis() - lastSerialTime >= SERIAL_INTERVAL) {
    lastSerialTime = millis();  // Update the last sent time
    sendStateOverSerial(currentState);
  }
}
```

3.6 Main.ino

```
#include "StateHandler.h"

void setup() {
}

void loop() {
  updateState();
  handleStateChange();
  updateSerialCommunication();
  updateBuzzerSequence();
}
```

4 Annex

- Design Files (Kicad 9): https://devinci-my.sharepoint.com/personal/matthieu_pecoraro_edu_devinci_fr/_layouts/15/guestaccess.aspx?share=ETnXWXaeamxIvn59a_L8oiUBylDcDPXiuulOgeFaUDfTiQ&e=WsSy8p