Axelle

Dossier de cloture.

CLES-FACIL

INSA LYON

Table des matières.

Remerciements.

Tous nos remerciements vont à tous ceux qui, de près ou de loin, ont contribué à la réalisation de ce projet.

- L'INSA de Lyon, notre école, qui chaque année nous soutient :
    - Le Comité de la Vie Associative (CVA),
    - La Direction de l'Information et de la Communication (DIC),
    - Les départements de Génie Mécanique et Développement (GMD), Génie Electrique (GE), Informatique (IF), et Génie Energétique (GEN).
    - Le Centre d'Etudes et de Réalisation de Prototypes (CEREP),
- Les sociétés Hexcel Fabrics (Fibre de carbone), SETS (Traitement de surface : chrome.), Varta (Piles), Le Cri du Kangourou (CDK)(parachutes).
- Les sociétés Analog Devices, Maxim, Texas instruments, qui proposent de très bons échantillons.

Nous tenons à remercier également nos partenaires :

- Le CNES pour la mise en œuvre de nos projets.
- Planète Sciences, pour l'organisation de la campagne de lancements, son soutien tout au long de l'année et ses actions pour les jeunes et la science.

A titre plus personnel, nous tenons à remercier :

- Francis Lésel, notre suiveur,
- Pierre Da Crouz (Pierre De Cascade) pour sa visite au club vers la fin du projet, ainsi les membres de Cascade, jamais à court de Kréma Régalad,
- Gilbert Chaléroux, qui nous a donné un coup de main pour la réalisation du tube de Pitot, pour la matrice du cône et autres pièces mécaniques.
- Denis Psomiadès pour ses conseils.

http://www.insa-lyon.fr/Associations/ClesFacil/

Nicolas Chaléroux
nico@sdbdc.com

Projet : Axelle

12 décembre 2002
Version 3.0

Dossier de Définition

Table des mises à jour du document

Version	Date	Objet de la mise à jour	Auteur
1	12/12/02	Création du document	Nicolas Chaléroux
2	12/10/02	Intégration dans le modèle de mise en page	Nicolas Chaléroux
2	12/10/02	Relecture	Jérôme Hamm Louise Pontal

Sommaire

1. Besoin en matériel

Propulseur : OUI (Chamois à priori)
Virole : NON
Emetteur : OUI (KiMi)
Un récepteur GPS de poche pourra être demandé occasionnellement par le club pour réaliser la mise en œuvre du système de localisation du module.

2. Les objectifs

2.1. L'objectif principal

Nous souhaitons comparer différentes méthodes de mesure de l'accélération.

Pour cela nous mettrons en œuvre les moyens suivant :

- Pour des mesures d'accélération directes :
  - ◇ Un accéléromètre du commerce
  - ◇ Un accéléromètre à jauges de déformation
- Pour des mesures d'accélération par l'intermédiaire de la vitesse :
  - ◇ Un anémomètre à fil chaud[1]
  - ◇ Un tube de Pitot

2.2. Les objectifs secondaires

Ils sont de deux ordres :

- Augmenter significativement la fiabilité du système de récupération des données
- Et augmenter également la qualité des mesures que nous allons effectuer

Voici les moyens que nous comptons utiliser pour répondre aux objectifs secondaires :

- Concernant la fiabilité : Nous mettrons la mémoire dans **un module** qui sera largué à culmination par un système d'ouverture totalement indépendant de l'ouverture du parachute. Ainsi nous doublons nos chances de récupérer la mémoire. Afin de mettre toutes les chances de notre coté, nous émettrons également nos données. Les esprits tordus nous opposerons que nous risquons de perdre[2] notre module, avec la mémoire, car ce dernier sera petit. Afin de répondre à ce problème, un **GPS** se trouvera dans le module et nous donnera la position de ce dernier pendant la descente. Ceci nécessite d'intégrer l'émetteur et le modulateur dans le module. En plus de cela, nous adoptons une nouvelle démarche au niveau

[1] Alias système BOG
[2] Sissonne comportant plus de végétation que Millau

des alimentations cette année. En effet, nous comptons répartir les alimentations afin de ne plus dépendre d'une seule alimentation. Ceci reprend le vieil adage qui dit qu'il ne faut pas mettre tous ses œufs dans le même panier.

- Concernant la qualité : Nous avons commencé notre réflexion par l'évaluation des sources de non qualité de nos mesures. Cela peut donc venir de trois endroits :

  - ◊ La source : le capteur est mauvais (mal dimensionné, mal mis en œuvre), cela a une solution assez simple : mieux concevoir (simple à identifier, mais moins à réaliser)
  - ◊ Le transport de l'information : les grands fils analogique de transport ont beaucoup de chances de se faire perturber par un agent externe[3]. Afin de pallier à ce problème nous avons décidé de réduire le nombre de fils en adoptant une logique de bus. De plus, les signaux seront transmis en numérique.
  - ◊ Acquisition de l'information : La conversion Analogique / Numérique est un vaste sujet qui peut réserver des pièges[4]. Comme nous voulons adopter une logique de bus cette fonction de conversion sera déportée sur chaque carte capteur.

On peut noter que la logique de bus nous apporte un autre avantage en dehors de nos objectifs. Elle limite le nombre de connexions entre la fusée et le module. Ces connexions devant être rompues lors du largage, moins il y en a, mieux c'est !

2.3. Objectifs tertiaires

Afin de ne jamais tomber à cours de travail et parce qu'on est jamais à cours d'imagination voici deux objectifs de troisième ordre :

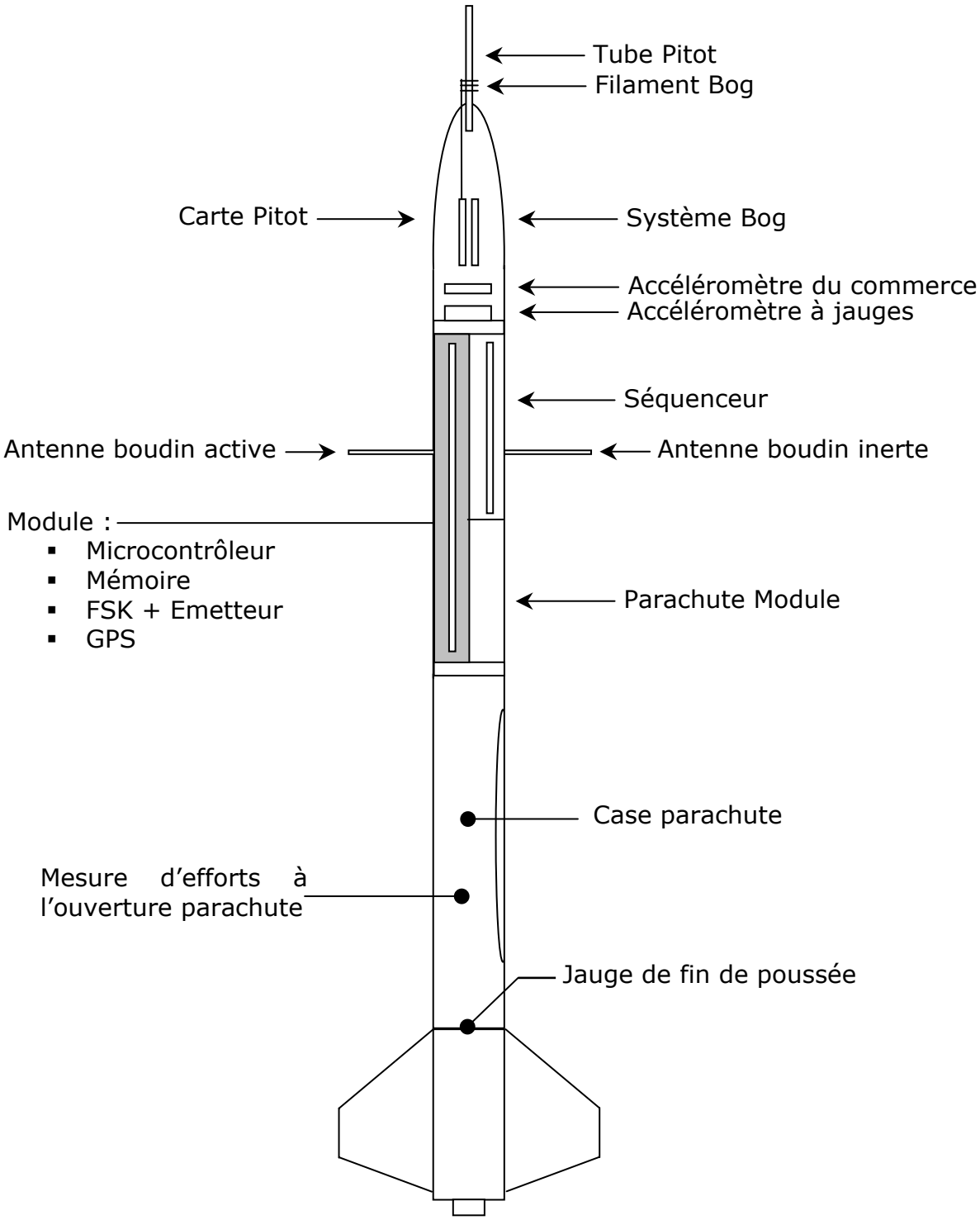
- Détecter la fin de la poussée
- Mesurer les efforts sur la sangle parachute à l'ouverture de ce dernier

Pour nous permettre de mieux mesurer le moment de fin de poussée nous pensons mettre une jauge de contrainte sur la bague de poussée. Cette mesure nous permettra de mieux appréhender l'interprétation des courbes d'accélération.

La mesure du choc sur la sangle du parachute n'est pas connectée au reste de notre expérience. Nous voudrions juste savoir de quel ordre est le majorant que donne la formule du cahier des charges. Ce capteur sera encore à base de jauge de contrainte par contre il sera entièrement autonome. En effet, lorsque l'événement que nous voulons observer aura lieu, la fusée ne comportera déjà plus de microcontrôleur ni de mémoire : le module aura été éjecté. Ce système a juste à enregistrer la valeur maximale de l'effort subi.

[3] Au moins l'émetteur dans notre cas
[4] Repliement de spectre…

3. Schéma d'intégration



4. Identification des risques

4.1. Aux niveaux des capteurs

L'accéléromètre du commerce relève de la mise en œuvre d'un capteur industriel tout encapsulé ce qui ne devrait pas poser de problème majeur.

L'accéléromètre à jauge est plus délicat car dans notre expérience passée[5], les jauges et les amplificateurs à fort gain nous ont posé de réels problèmes.

C'est un peu la même situation pour l'anémomètre à fil chaud. La complexité du montage et des problèmes techniques au festival, alors que le concepteur était absent, ont empêché ce capteur d'être embarqué.

Enfin l'identification des risques sur un capteur, tel que le Pitot, semble peut-être inutile mais le projet précédent nous a montré que nous n'étions pas forcement capable de mettre parfaitement en œuvre ce capteur.

4.2. Autres organes électroniques

Tout comme l'accéléromètre du commerce, la mise en œuvre du GPS relève de l'intégration mais l'équipe actuelle n'a jamais utilisé ce genre de capteur. Le risque vient donc de la nouveauté !

La logique de bus : ceci est un point très sensible. En effet, s'il y a un problème sur le bus, nous n'aurons aucune valeur. La mise en œuvre sera donc très soignée. Malgré cela, une solution pour shunter le bus sera prévue : accès sur les cartes au signal analogique et convertisseurs sur la carte microcontrôleur. Le schéma de câblage sera alors totalement différent mais il faudra le prendre en compte pour pallier au pire des cas.

La conversion A/N déportée : que la conversion soit sur la carte microcontrôleur, comme nous le faisions jusqu'à maintenant, ou sur les cartes capteurs n'est pas très important, l'important étant que cela nous impose de changer de technologie. En effet, avant nous utilisions des convertisseurs internes au microcontrôleur, mais maintenant il s'agira d'entités indépendantes qu'il faudra commander à distance.

Les connexions électriques fusée/module : Dans le cas du bus nous aurons trois connexions entre la fusée et le module, en mode dégradé ce sera plutôt de l'ordre de 20. Ces connexions doivent être d'un contact parfait (fils du bus) qui résiste aux vibrations et à l'accélération et également à force de retrait quasi nulle pour permettre le largage du module.

Le séquenceur : Partie critique par excellence faisant partie de la famille des tartes à la crème car toute fusée est confrontée à ce problème. La simplicité

[5] Avec le projet ELA en 2002

de fabrication reste ici le maître mot, somme toute, cette année nous veillerons à bien faire deux minuteries totalement indépendantes[6] pour le module et la porte parachute.

La répartition des alimentations : ceci est plutôt une bonne chose, l'avantage majeur étant que la fonction sera répartie sur plusieurs systèmes physiques. Mais encore une fois je précise qu'il s'agit de quelque chose de totalement nouveau pour nous et donc, par voie de conséquence, risqué. Au même titre que le séquenceur, les alimentations sont des organes critiques par excellence, il s'agit donc de ne laisser aucune chance à Murphy.

4.3. Organes mécaniques

Le système de largage : ceci étant une petite critique au plus haut point nous n'avons pas cherché à être extraordinaires. L'équipe actuelle a déjà une bonne expérience dans le domaine des séparations en vol avec le projet Cerbère[7] en 2001 et le système retenu est ancien au club et a déjà fait ses preuves sur trois[8] FUSEX.

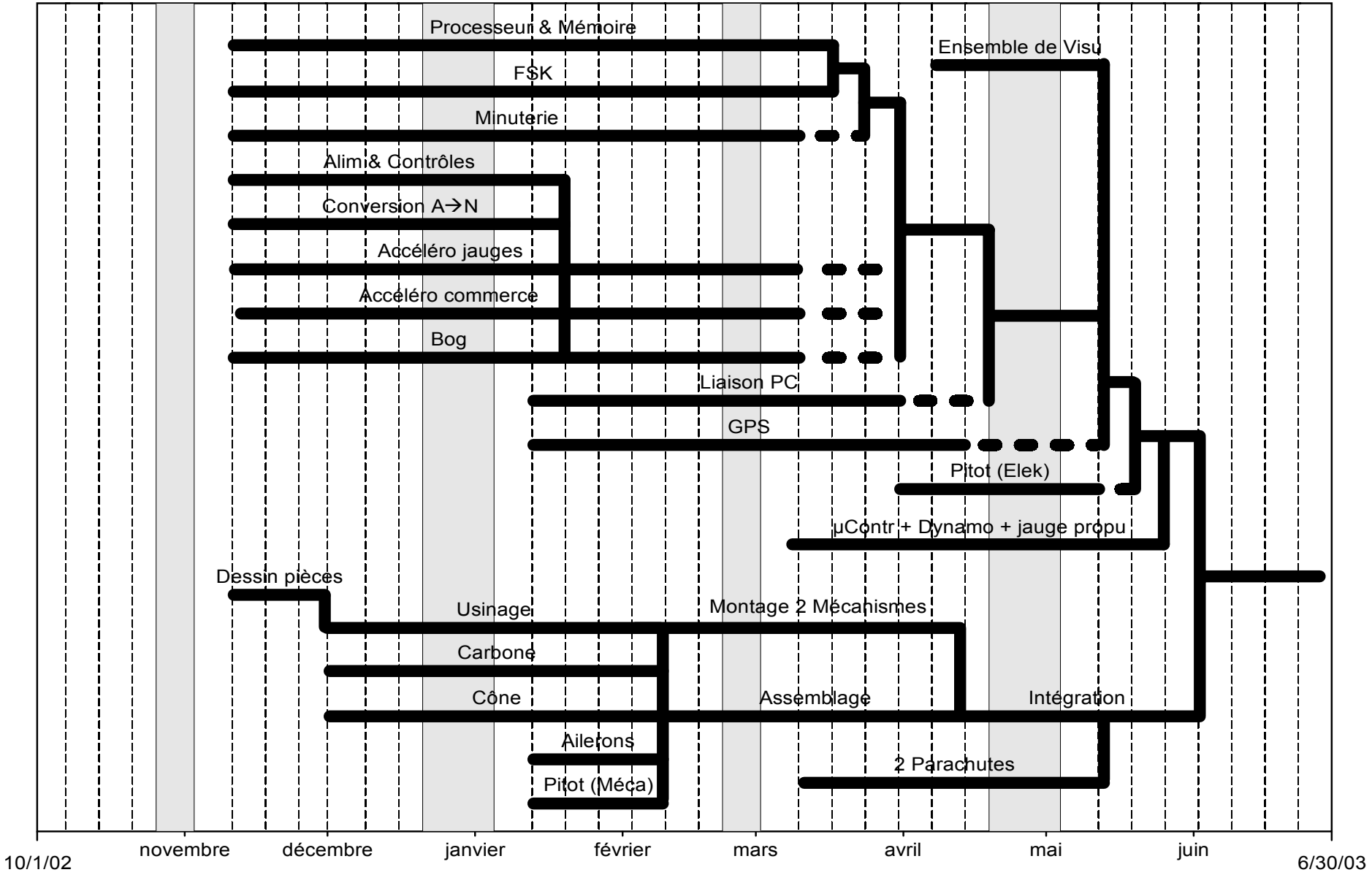
5. Demande de dérogation

Le club demande à pouvoir faire descendre relativement lentement (2 $m.s^{-1}$) son module (2Kg) pour que le GPS puisse se recaler sur les satellites.

6. Annexe

6.1. Planning

[6] Condition sine qua non pour doubler la fiabilité de notre système de récupération des données
[7] Séparation de deux modules
[8] PSO (1995), Pauline (1997) et Nausicaa (2001)


Processeur & Mémoire
FSK
Minuterie
Alimi& Contrôles
Conversion A→N
Accéléro jauges
Accéléro commerce
Bog
Ensemble de Visu
Liaison PC
GPS
Pitot (Elek)
uContr + Dynamo + jauge propu
Dessin pièces
Usinage
Montage 2 Mécanismes
Carbone
Cône
Assemblage
Intégration
Ailerons
Pitot (Méca)
2 Parachutes
10/1/02
novembre
décembre
janvier
février
mars
avril
mai
juin
6/30/03

http://www.insa-lyon.fr/Insa/Associations/ClesFacil/

Nicolas Chaléroux : *nico@sdbdc.org*
(Conception et CAO électronique, conception mécanique)
Vincent Haluska : *haluska@club-internet.fr*
(Etalonneur)
Pierre Fayet: *fayetpierre@aol.com*
(Conception électronique, réalisation électronique)
Laurent Ruet: *laurent_ruet@hotmail.com*
(Garde fou pour électroniciens)

Projet : Axelle

15 Decembre 2003
Version 13.0

01. Mesure de vitesse par tube de Pitot.

1. Capteur Pitot

La fusée aXelle avait pour but, rappelons le, de comparer différentes méthodes de mesure de l'accélération. La mesure d'accélération via un tube de Pitot était précisément un des moyens utilisés pour mesurer ce paramètre.

Nous pouvons qualifier la méthode d'assez ancienne puisque le tube de Pitot fut inventé par le français Henri Pitot (1695-1771) au 18ème siècle. La « machine à mesurer la vitesse des eaux courantes » fût ensuite améliorée par Henry Darcy (1803 à 1858). C'est à lui que l'on accorde la découverte de la forme moderne du tube de Pitot, il fut le premier chercheur à postuler l'existence d'une couche limite dans l'écoulement d'un fluide, il contribua au développement de l'équation de Darcy-Weisbach pour la résistance de l'écoulement dans un tuyau.

Ce chapitre va décrire la mise en œuvre d'un capteur à base de tube de Pitot, du point de vue mécanique mais également électronique. Afin de développer la démarche complètement, nous verrons à la fin les résultats qui sont issus de cette expérience.

Note : *Le capteur tube de Pitot a été un des derniers conçu et réalisé durant l'année du projet mais figure parmi premiers présentés de ce rapport. Ceci est du à sa relative simplicité qui nous permettra d'introduire quelques parties techniques qui sont également reproduites dans d'autres capteurs plus « complexes ».*

1.1. Objectifs

L'objectif premier est de réaliser une mesure de l'accélération de façon maîtrisée. C'est-à-dire en ayant conscience des limites de résolution et de ses facteurs limitants. Nous commençons donc ce document par lister des contraintes que nous avons du prendre en compte lors de la conception et la réalisation de ce capteur :

- L'électronique devait pouvoir être contenue dans une boite blindée électromagnétiquement de dimensions : 112×62×31mm.
- Les communications électriques avec l'extérieur de la boite devaient passer par un connecteur DB9 plus ou moins normalisé. Les 9 fils de communication comprenaient donc : source d'énergie, bus de données, feedback alimentation.
- Les données devaient être mises à disposition via le bus I²C adopté dans toute la fusée.
- Nous devions utiliser le moins de piles possible afin de limiter au maximum le poids (comme toujours…)
- Nous devions minimiser la longueur de tuyau entre le tube (partie mécanique) et le capteur (partie électronique)
- Enfin, une dernière contrainte et nous verrons qu'elle a son importance, ce capteur ne devait vraiment pas coûter cher. En effet, nous étions en fin d'année et le budget commençait à être serré.

En résumé, ces contraintes sont de trois ordres, à ne pas oublier : intégration, interfaçage et coût. Il ne faut pas oublier non plus que ce sont ces contraintes qui ont orienté notre projet, il est certain qu'un tube de Pitot réalisé par l'ONERA[1] répondra à d'autres contraintes et sera donc réalisé différemment.

[1] ONERA : Office National d'Etudes et de Recherches Aérospatiales

1.2. Principe du tube de Pitot

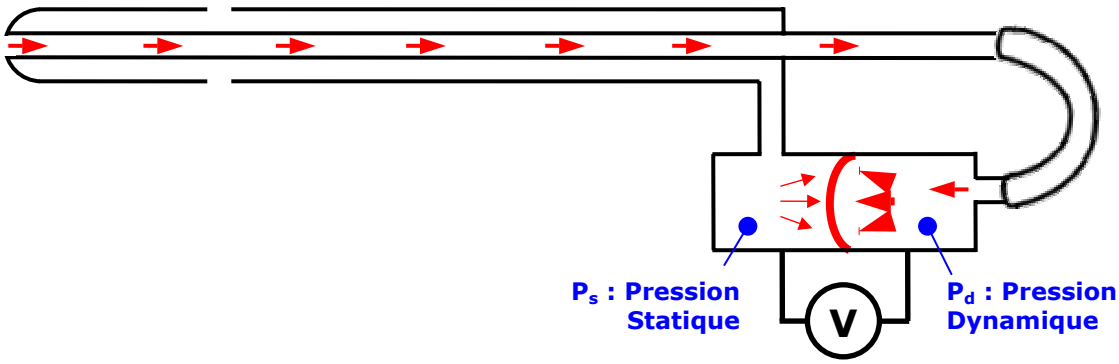


Figure 1 : Schéma de principe du tube de Pitot

Le tube de Pitot a pour principe essentiel de mesurer la différence de pression entre l'air du vent relatif créé par le déplacement de la fusée et l'air ambiant. Pour cela, un tube est placé en tête de fusée (milieu ne subissant pas de perturbation du flux d'air) avec deux prises d'air. Une directe située à la pointe du tube pour capter la pression due à la vitesse que nous nommerons dorénavant : pression dynamique. La deuxième prise d'air ne doit pas subir les effets de la vitesse, elle est donc perpendiculaire au mouvement et capte la pression statique. Cette dernière est souvent négligée lors de la mise en œuvre d'un tube de Pitot, pourtant c'est elle qui permettra de rendre la mesure indépendante de la pression atmosphérique.

Note : *Ma dernière remarque en amène une autre : le schéma ci-dessus présente le dispositif utilisé dans aXelle mais peut très bien être réalisé à partir de deux capteurs de pression absolue. (un pour chaque pression) Il s'agira alors de réaliser la soustraction plus tard dans la chaîne de mesure. Il est même possible de réaliser cette soustraction lors de l'exploitation des résultats, afin de faire d'une pierre deux coups, la pression statique donnant alors accès à l'altitude…*

Observons maintenant les équations qui sont à la base de ce capteur :

$$P_d = P_s + \frac{1}{2}\rho V^2 \qquad \textbf{(Equation 1 dite de Bernoulli)}$$

P_d *et* P_s : Pressions dynamique et statique en Pa

ρ : étant la masse volumique de l'air exprimée en $kg.m^{-3}$

V : étant la vitesse de déplacement exprimée en $m.s^{-1}$

Comme nous pouvons le voir au travers de cette équation le tube de Pitot ne donne accès en première approche qu'à la vitesse. Notons, dès à présent, que nous avons fait l'hypothèse que ρ était constant et égale à 1.225 $Kg.m^{-3}$ durant tout le vol. Nous avons également considéré que l'écoulement gazeux était incompressible. Cette dernière hypothèse est vraie pour des vitesses inférieures à Mach 0.3 et fausse pour des vitesses supérieures à Mach 0.85. Nous verrons dans la partie dimensionnement que notre fusée devrait atteindre la vitesse maximale de Mach 0.5. Cette vitesse n'est pas dans la plage la plus parfaite pour l'application de la formule mais n'est pas non plus dans la plage ou la théorie est complètement inapplicable. Pour avoir accès à

l'accélération, il faudra bien entendu utiliser le fait que cette dernière est la dérivée première de la vitesse :

$$\gamma = \frac{dv}{dt} = \frac{d^2x}{d^2t}$$ ***(Equation 2 : Loi de la cinématique)***

Notre démarche qui consiste à comparer plusieurs méthodes de mesure de l'accélération nous amène avec ce capteur à mesurer la vitesse puis à remonter à l'accélération lors de l'exploitation des résultats. Ceci est un exemple typique ou la grandeur étudiée n'est pas la grandeur mesurée, souvent les équipes se retrouvent dépourvues lorqu'elles rencontrent cette question dans le dossier CNES ou tout autres rapport de suivi pour Planète Sciences. Cette étape « supplémentaire » peut entraîner un surcroît d'erreurs mais d'un autre coté ceci permet aussi d'avoir une méthode dont le principe est totalement différent des autres au niveau de la mesure physique. L'idée principale est de dire que cette méthode a des biais (comme toutes les méthodes d'ailleurs) mais que comme elle est basée sur un principe très différent, elle a de fortes chances de ne pas avoir les mêmes biais que les autres méthodes. Ainsi nous pouvons globalement espérer mieux percevoir le phénomène grâce à la multiplication des capteurs.

En conclusion de ce paragraphe général, et afin de vous mettre un peu l'eau à la bouche, je pense qu'il est important de rappeler l'étendue des utilisations du tube de Pitot : Aéronautique (civile, loisirs, militaire), Formule1 (caché sur les suspensions avant), débitmètre, vitesse des souffleries… Nous pouvons aussi ajouter à ce palmarès l'immense majorité des Fusex[2] conçues sous couvert des activités de Planètes Sciences.

1.3. Mécanique

Le dimensionnement mécanique a été réalisé à partir de la notice technique [1]. Les dimensions effectivement retenues sont reportées sur le schéma suivant :

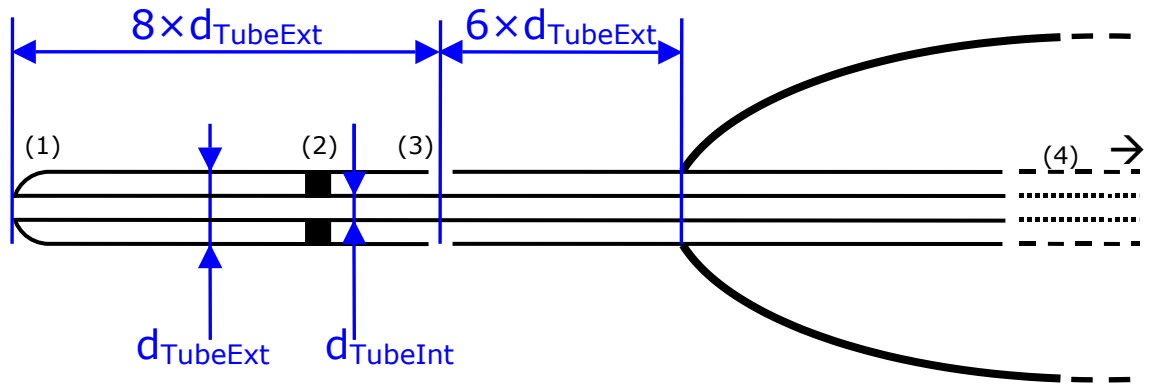


Figure 2 : Dimensionnement mécanique du tube de Pitot

Dans le cas d'aXelle le diamètre extérieur du tube extérieur ($d_{TubeExt}$) était de 10mm, son diamètre interne était de 8mm. Le tube intérieur avait lui un diamètre extérieur ($d_{TubeInt}$) de 6mm et 5mm pour l'intérieur.

[2] Fusex = Fusée Expérimentale

La fabrication d'un tube de Pitot demande le plus grand soin, en effet la moindre entrave au passage de l'air (bosse, colle, soudure, bavure) viendrait rapidement perturber les mesures. Je profite donc de cette partie pour remercier mon père qui a plus que contribué à la réalisation de cette pièce mécanique. J'espère qu'il ne m'en voudra pas de révéler ici quelques secrets de fabrication :

(1) La fermeture du tube extérieur au bout de l'ensemble. Ceci est un point délicat car ce bouchage ne doit pas empiéter sur le tube interne sous peine de fausser la mesure de la pression dynamique. La notice technique présente un projet qui a usiné une pièce spécialement pour cette fonction. Sur le projet aXelle, nous avons utilisé de la pâte « soudure à froid » : pâte à modeler du bricoleur qui répare tout en un clin d'œil.

(2) Il est essentiel d'assurer le fait que les deux tubes resteront à distance constante l'un dans l'autre et que surtout le tube interne ne viendra pas toucher le tube externe. A cette fin nous avons placé deux petites bagues entre les deux tubes, il s'agissait de petits bouts de tube flexible de 5mm de long. Par contre il faut faire très attention à l'endroit où l'on place ces petites « entretoises ». En effet, il ne faut pas obstruer le passage de l'air assurant l'équilibre de la pression statique. Nous avons placé la première « entretoise » au niveau du repère (3) c'est-à-dire au dessus des trous de prise d'air statique. La deuxième a été placée tout en bas sous la sortie de cette prise d'air. Un schéma valant mille mots, je vais ajouter une figure à mes propos :

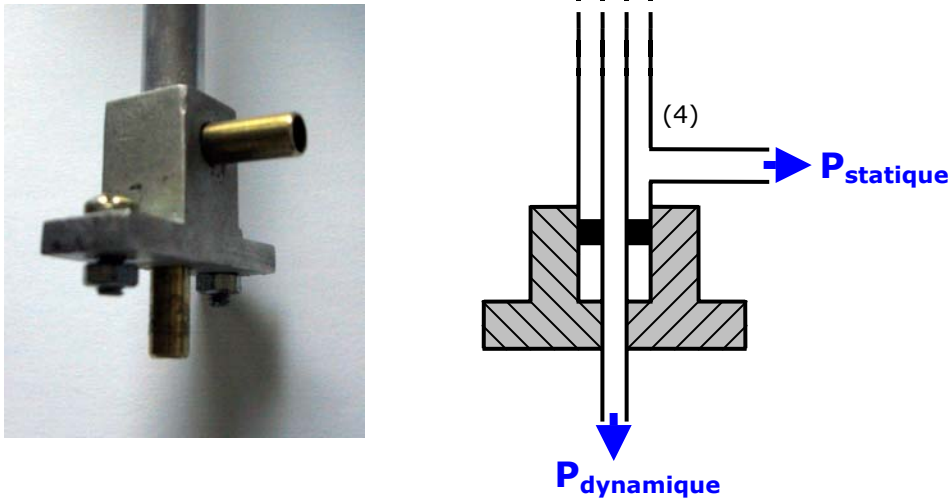


Figure 3 : Fixation du tube de Pitot

La deuxième « entretoise » est encore représentée sur cette figure par les petits carrés noirs autour du tube interne. Le dessin ci-dessus donne également une très bonne idée de la pièce réalisée pour la fixation du tube (pièce grisée en hachures). La forme de cette pièce assure une bonne fixation du tube, l'étanchéité entre les deux tubes, le raccordement facile des tuyaux flexibles en vue de joindre le capteur de pression.

---

(1) (2) sont des renvois à la **Figure 2** page 4

(3) La prise d'air statique est en fait réalisée grâce à 6 petits trous (diamètre 1mm) répartis de façon homogène sur la circonférence du tube. Il est très difficile de percer de si petits trous sur un tube ayant un petit rayon de courbure (1 cm dans le cas de aXelle). Il a donc fallu utiliser un outillage spécial réalisé à partir d'un boulon hexagonal (photos ci-contre). En effet, ceci permet d'avoir la bonne répartition des angles et de guider le foret lors de la phase d'attaque.

(4) La longueur de tube à l'intérieur du cône a peu d'importance, mais il est toujours bon d'essayer de faire au plus court et de faciliter l'intégration. Dans le cas où le tube de Pitot fait également office d'antenne la longueur sous cône peut être utilisée pour compléter la longueur d'antenne extérieure. Ceci sous-entend bien sûr que le cône ne soit pas un élément conducteur comme le carbone dans le cas de aXelle. (Voir également les problèmes de CEM[3] pour le rayonnement à l'intérieur de la fusée)
Il est également très important de penser dès la conception aux raccords des tuyaux souples qui iront au capteur de pression. Une autre chose à garder à l'esprit est le fait que les tuyaux souples ne sont pas si souples que cela, en effet, les rayons de courbure restent de l'ordre de 5 à 10 cm ce qui peut poser des problèmes lors de l'intégration finale.

Voici enfin trois photos de l'ensemble mécanique Pitot de aXelle :

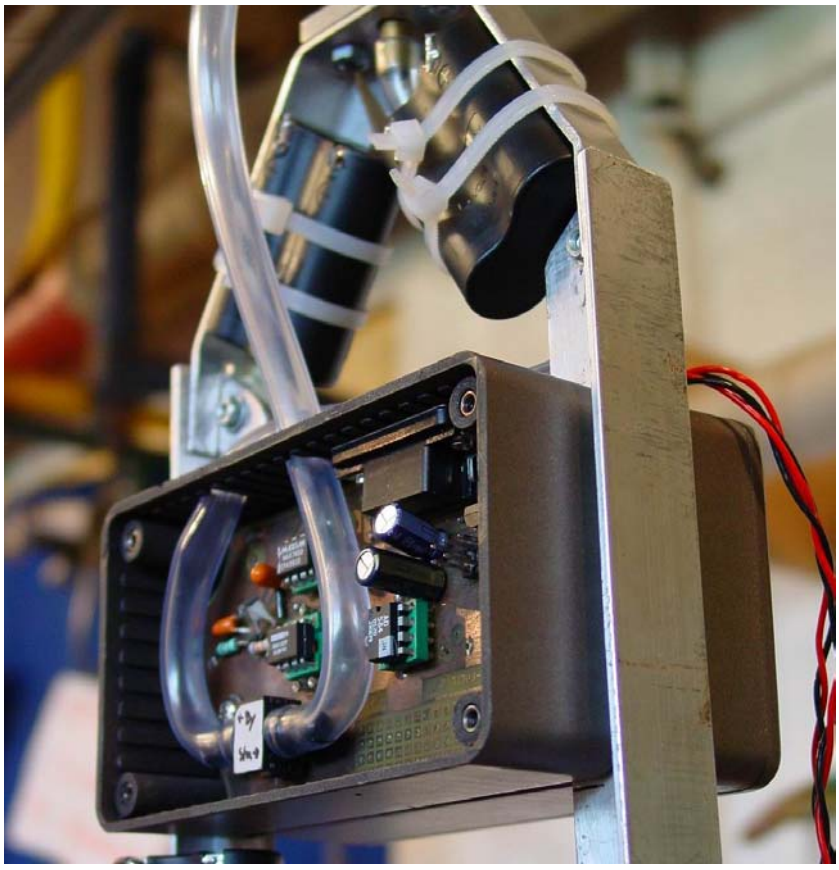


Figure 4 : Vue de la carte électronique.

(3) (4) sont des renvois à la figure 2 page 4

[3] **CEM :** Compatibilité Electromagnétique

Figure 5 : Vue du tube de Pitot et de la carte.

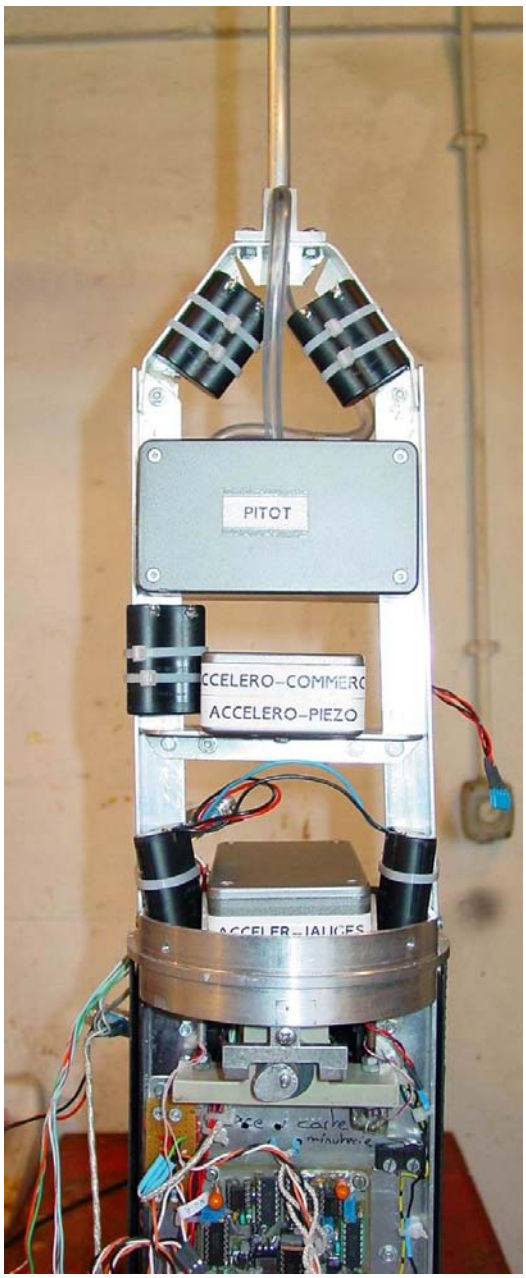


Figure 6 : La boîte une fois refermée.

En conclusion de cette partie, il est important de noter que la réalisation de la partie mécanique d'un tube de Pitot demande beaucoup de temps étant donné l'extrême soin qu'il faut apporter à toutes les opérations.

1.4. Electronique

1.4.1. Le capteur

Les contraintes que nous devions respecter pour le choix (dimensionnement) de notre capteur de température étaient les suivantes :

- intégrable dans une boîte blindée de 112×62×31mm ce qui excluait l'utilisation de deux capteurs absolus. (pourtant disponibles au club). De plus, nous préférions travailler avec un capteur différentiel.
- On devait pouvoir mesurer la pression correspondante à la vitesse maximale de la fusée. Cette vitesse maximale nous est donnée par Trajec : 170m.s^{-1}. En injectant cette valeur dans l'équation 1, on obtient :

$$\Delta P_{\max} = max(P_d - P_s) = \frac{1}{2}\rho V_{\max}^2 \qquad avec \qquad \rho = 1{,}225\, kg.m^{-3} \qquad et \qquad V_{\max} = 170\, m.s^{-1}$$

Unités : Pa=[m^{-1}][kg][s^{-2}]= $\rho \times V^2$ =[kg][m^{-3}]×[m]^2[s^{-1}]^2

Soit : $\Delta P \max \approx 17700 Pa$

- Réutiliser un capteur dont on dispose déjà au club afin de limiter les frais. Un capteur de pression qui répond à nos besoins coûte environ 30€ !

Finalement notre choix s'est porté sur un capteur issu de la série 26PC disponible chez Honeywell (http://www.honeywell.com). Ce capteur était un reste du sponsoring qu'avait fait Honeywell à l'occasion de la fusée Nausicaa (Millau 2001). Cette dernière embarquait une vingtaine de capteurs de ce type afin d'étudier la couche d'air sur le cône de la fusée.

Le principe de ce capteur différentiel est basé sur un pont de Wheatstone à jauges de contraintes. Voici un extrait des grandes caractéristiques de ce capteur. Peu de documentation est disponible à propos de ce capteur vous pouvez cependant vous reporter à la référence [2] qui est une référence particulière de la série 26PC (version 100psi).

- L'alimentation doit se faire entre **10Vdc typ. et 16Vdc max.**, ceci nous oblige donc à utiliser deux piles mais ce n'est pas vraiment un problème car nous verrons que la consommation est très faible et qu'il est donc possible d'alimenter ce capteur depuis des piles servant à d'autres dispositifs de la fusée.
- Le capteur est compensé en température sur la plage **0°C à 50°C.** Il faut bien faire attention à ce paramètre et surtout à ne pas le confondre avec la plage de fonctionnement qui est de **-40°C à 85°C.** Une fusée ne sera jamais exposée à une température inférieure à 10°C (en hiver dans le sous-sol du CLES-FACIL en cas de panne de chauffage) mais par contre pour la température supérieure la question est plus délicate. En effet, le cahier des charges stipule que la fusée doit pouvoir rester 45 minutes en rampe sans que cela ne l'affecte. (autonomie et qualité des expériences). Les lancements se déroulant généralement en plein été et le CLES-FACIL ayant une forte tendance à utiliser des coques carbone noires, quelle peut-être la température atteinte à l'intérieure de la fusée dans des conditions de plein soleil ? Nous estimons cette température à environ 50°C voire plus. Notre capteur est donc à sa limite de compensation, mais de toute façon le défaut de linéarité est de **0.20% max.** ce qui nous laisse confiants.

- Notre capteur n'intègre pas d'amplificateur et a une sensibilité de **16.7 mV.psi^{-1}**. Ceci signifie qu'il faudra prévoir un montage offrant un gain de l'ordre de 100 afin d'obtenir une plage de sortie compatible avec celle d'entrée du convertisseur analogique/numérique (0-5V).
- Ce capteur est **utilisable en positif et négatif (vacuum or positive pressure)**, ce qui n'est vraiment pas une obligation. (Dans notre cas, même lors de la décélération la variation de pression est positive par principe). Nous n'utiliserons donc qu'un seul sens de variation du capteur, ceci est un élément important à retenir afin de bien concevoir l'étage d'amplification.
- **L'offset est de 0mV typ.** ce qui signifie qu'une différence de pression nulle entre chaque coté de la membrane se traduit par une tension nulle aux bornes du capteur. Il s'agit encore d'un point dimensionnant pour l'étage d'amplification que nous verrons par la suite.
- **Le temps de réponse est de 1ms max.** ce qui signifie qu'un échantillonnage au-delà de 1kHz serait totalement superflu. Nous pensons que le tube de Pitot global a une inertie assez grande (temps de réponse long) à cause des effets de la mécaniques des fluides. Nous voyons ici que le capteur de pression ne sera donc pas un facteur limitant pour la fréquence d'acquisition.
- **Le poids du capteur est de 2g** ce qui est parfait pour nos applications, car ne l'oublions pas la chasse au poids se fait partout. Les pièces mécaniques et les piles sont toujours très surveillées mais le poids des composants et de la connectique n'est pas négligeable dans une fusée comme aXelle.
- Deux paramètres importants à observer sont les résistances aux chocs et aux vibrations. Le choc peut être dimensionné par l'accélération maximum de la fusée (8.5g d'après Trajec) ou celui d'une chute de carte dû à une fausse manipulation de l'opérateur ou d'un de ses facétieux collègues ;o) dans ce cas il est bon de prévoir 50g. Pour les vibrations, c'est la grande inconnue, les quelques fusées expérimentales lancées afin d'étudier ce phénomène n'ont jamais données grandes conclusions. Le **capteur résistant à 150g (choc et vibration)** dans les deux exemples de sollicitations présentées ici, nous sommes confiants.
- Impédance d'entrée : 7.5kΩ typ. et Impédance de sortie : 2.5kΩ typ. Nous sommes dans le cas d'un pont de Wheatstone. Or, quiconque connaissant le principe de ce pont peut légitimement se demander pourquoi ces deux impédances ne sont pas égales comme on pourrait le croire à priori. En fait nous avons affaire à un capteur un peu plus élaboré que cela, puisqu'il comporte une résistance de plus ! Regardons un peu le pont de Wheatstone en détail :

  Si l'on prend une résistance du pont, toute seule : sa valeur va varier de façon linéaire en fonction des variations du paramètre qu'elle est censée mesurer (on appelle cela « mesurande »), selon la loi, toute simple : $R_c(m) = R_{c0} + s\Delta m$, où R_{c0} représente la valeur de la résistance au repos, sans qu'aucune contrainte ne lui soit appliquée, s : coefficient de proportionnalité et Δm : variation du mesurande. On voit que cette loi est linéaire.

  Maintenant, si l'on insère cette résistance dans un pont de Wheatstone classique (rappel : 4 résistances, ou plutôt 4 jauges dans notre cas en opposition deux à deux) on aura en sortie du pont une tension $Vm = E\, s\Delta m / R_{c0}$ et la relation est toujours linéaire. E : tension d'alimentation du pont : celle-ci doit être très précise car comme on le voit dans la formule nous ne pourrons pas savoir dans le cas ou elle dérive s'il s'agit d'une variation de tension ou de mesurande. Seulement voilà : Il faut tenir compte de ce que les métrologues nomment « grandeurs d'influence » : Température, humidité, rayonnements et surtout le temps (c'est peut-être le plus important), génératrices de dérives et donc d'erreurs si elles ne sont pas prises en compte. Intéressons-nous aux variations du paramètre de température (θ): R_{c0} n'est plus constant et vaut : $R_{c0} = R_0 (1 + \alpha\theta)$ où θ

représente la température. De plus, le paramètre s vaut désormais : $s = s_0 (1 + \beta\theta)$. En définitive, on a en sortie du pont :

$$Vm = \frac{E\,\Delta m\,S_0(1+\beta\theta)}{R_0(1+\alpha\theta)}$$

Et ça, ce n'est pas du tout linéaire.

Si, par manque de chance, et c'est le cas le plus fréquent, α est différent de β, il faut compenser la variation de Vm en fonction de θ en faisant varier la tension d'alimentation de façon opposée. On insère donc une résistance R_s en série avec le pont, c'est ce qui fait que l'impédance d'entrée et l'impédance de sortie du capteur sont différentes.

On va donc se retrouver avec une résistance $R_s(\theta) = R_{s0} (1 + \gamma\theta)$ en série avec le pont dont l'impédance d'entrée est toujours $R_{c0} = R_0 (1 + \alpha\theta)$. La tension de sortie va donc dépendre de trois paramètres : α, β, γ. Seul γ peut être choisi puisqu'il dépend du matériau de la résistance série que l'on vient d'ajouter. Si on écrit Vm en fonction de ces trois paramètres (je ne le mettrai pas ici, rappelons seulement que la tension d'alimentation du pont a changé puisqu'on a placé une résistance en série sur son alimentation initiale), on s'aperçoit que l'on peut simplifier les termes non-linéaires si on a $\beta = (\alpha R_0 + \gamma R_{s0}) / (R_0 + R_{s0})$ et, comme on choisit nous-mêmes R_{s0} (je rappelle que c'est la résistance série que je viens d'ajouter), on trouvera en renversant l'expression précédente : $R_{s0} = (\alpha - \beta)/(\beta - \gamma)$. Comme R_{s0} doit toujours rester positive, on choisira $\alpha>\beta>\gamma$ ou bien $\alpha<\beta<\gamma$.

Deux remarques importantes : Cette méthode de stabilisation en température fait diminuer la sensibilité du capteur, de plus il faut que la résistance R_{s0} soit à la même température que le pont que l'on souhaite compenser, ce qui n'est pas toujours facile à réaliser mais cela est par contre vérifié pour les capteurs intégrés comme celui utilisé pour le tube de Pitot.

- Enfin, je termine par le plus important des paramètres : la plage de mesure qui est sur notre capteur de **±2psi** soit **±13790Pa** soit encore **150m.s^{-1}**. Comme nous le remarquons nous sommes en dessous des 170m.s^{-1} retenus au paragraphe 1.4.1.

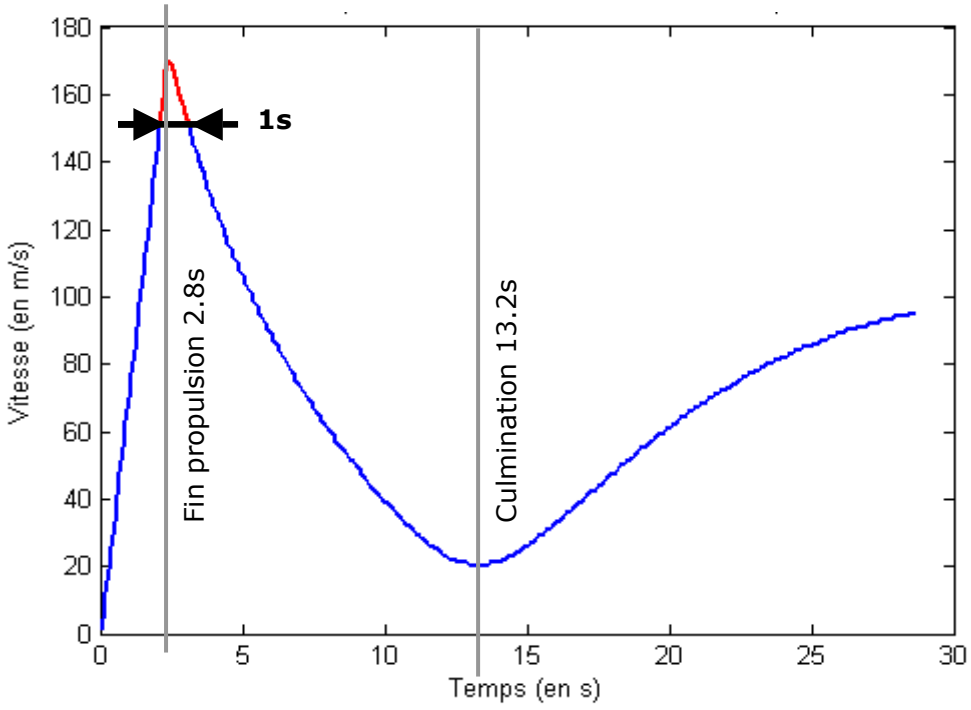


Figure 7 : Estimation par Trajec de la vitesse de la fusée.

Comme nous le voyons, la plage de mesure de notre capteur en ne nous posera problème que durant 1s de vol, partie du vol ou les plus hautes vitesse sont atteintes. Ce compromis nous apparaît tout a fait acceptable étant donné le prix que représenterait l'achat d'un autre capteur au lieu de réutiliser celui-ci.

1.4.2. La chaîne d'acquisition

Les chaînes analogiques utilisées dans aXelle reprenaient toutes le schéma classique, à savoir : Capteur → Amplificateur → Filtre → Convertisseur analogique/numérique. Cette chaîne était intégrée dans une boîte blindée électromagnétiquement et seuls des signaux numériques transitaient à l'extérieur, ces derniers étant moins sensibles aux perturbations que les signaux analogiques.

Note : *Il est toujours préférable de faire voyager l'information sous forme numérique plutôt que sous forme analogique. Ceci a été, pour nous, un grand enseignement du projet ELA (projet 2002). En effet, un signal numérique traduit un état logique correspondant physiquement à 0V ou 5V par exemple. Le signal analogique est un codage continu, par exemple : Valeur (en Volt) = a×Accélération (en g)+b. Les problèmes apparaissent lorsque l'on se trouve dans un cas non nominal, autrement nommé : « la vraie vie » ! En effet, si une perturbation intervient (émetteur de la fusée, émetteur du « plan d'op' », alimentations à découpage de la fusée, quartz du processeur …) elle sera alors captée par les dispositifs analogiques, le plus souvent par l'intermédiaire des fils de liaison de la carte vers le reste de la fusée, ou parfois même par certaines pistes du circuit imprimé. Par la suite, cette perturbation est amplifiée et traitée comme un signal analogique supplémentaire, qui s'ajoute au signal de mesure et devient donc gênant pour la mesure. C'est pourquoi l'on a intérêt à filtrer les tensions d'un circuit le plus souvent possible et à blinder convenablement les parties analogiques de la fusée. La fréquence du signal perturbateur est un paramètre à prendre en compte, surtout lors du dimensionnement des fils et des pistes du circuit : en effet, si la longueur d'onde d'un signal perturbateur est du même ordre de grandeur que la longueur des fils d'alimentation (au hasard), ces fils se transforment en antenne et apportent un signal perturbateur en plus de la tension d'alimentation du circuit. On retiendra que plus les liaisons sont courtes, plus leur fréquence de résonance est élevée. Ainsi, pour la fusée, l'émetteur étant la principale source de perturbations, on prendra des fils de longueur bien inférieure à 56 cm (qui est la longueur d'onde de la fréquence de l'émetteur). En faisant voyager l'information sous forme numérique, on rend la liaison moins sensible aux perturbations, car comme l'information est codée, le décodage est toujours possible sauf en cas de très fortes perturbations, ce qui n'est pas le cas ici vu la puissance de l'émetteur. Dans l'industrie, ces problèmes de Compatibilté Electro Magnétique (CEM) sont parfois bien plus difficiles à résoudre et peuvent avoir des conséquences désastreuses : déclechements inopinés d'appareils (compteurs d'impulsions…), transmissions de données faussées, claquage de composants sensibles…*

La chaîne citée ci-dessus, véritable cœur de la carte est assez simple. Tous les composants permettant à cette carte de fonctionner y ont été intégrés. En effet, pour respecter la règle qui consiste à n'avoir que des communications numériques avec l'extérieur de la boîte blindée nous avons également intégré les alimentations dans la boîboîte. Dans tout les cas il ne faut jamais perdre de vue la chaîne analogique simple citée ci-dessus qui est l'objectif de la boîte. Ensuite il faut toujours se demander a quoi sert ce composant ? C'est une partie de la chaîne ? - Oui/Non – Dans le cas négatif, il permet alors à un composant de la chaîne de fonctionner. Le secret pour lire les schémas ci-dessous, petit scarabée c'est de ne jamais oublier : Capteur → Amplificateur → Filtre → Convertisseur.

Le schéma suivant présente une vue fonctionnelle complète de la carte Pitot. On voit très bien apparaître en haut (de gauche à droite) la fameuse chaîne analogique et

ensuite en parallèle tout ce dont cette chaîne à besoin pour fonctionner parfaitement. Il s'agit essentiellement des alimentations et références de tension. Une partie de ce rapport est dédié à ce sujet, nous nous ne y attarderons donc pas ici.

Les rectangles autour des fonctions citées figurent les limites du découpage du projet dans le logiciel de CAO : Protel. Les paragraphes suivants présenteront les schémas électriques de chacun de ces rectangles. Afin de faciliter la localisation et la lecture des schémas la figure ci-dessous reprend les noms exacts des interfaces entre les fichiers Protel.

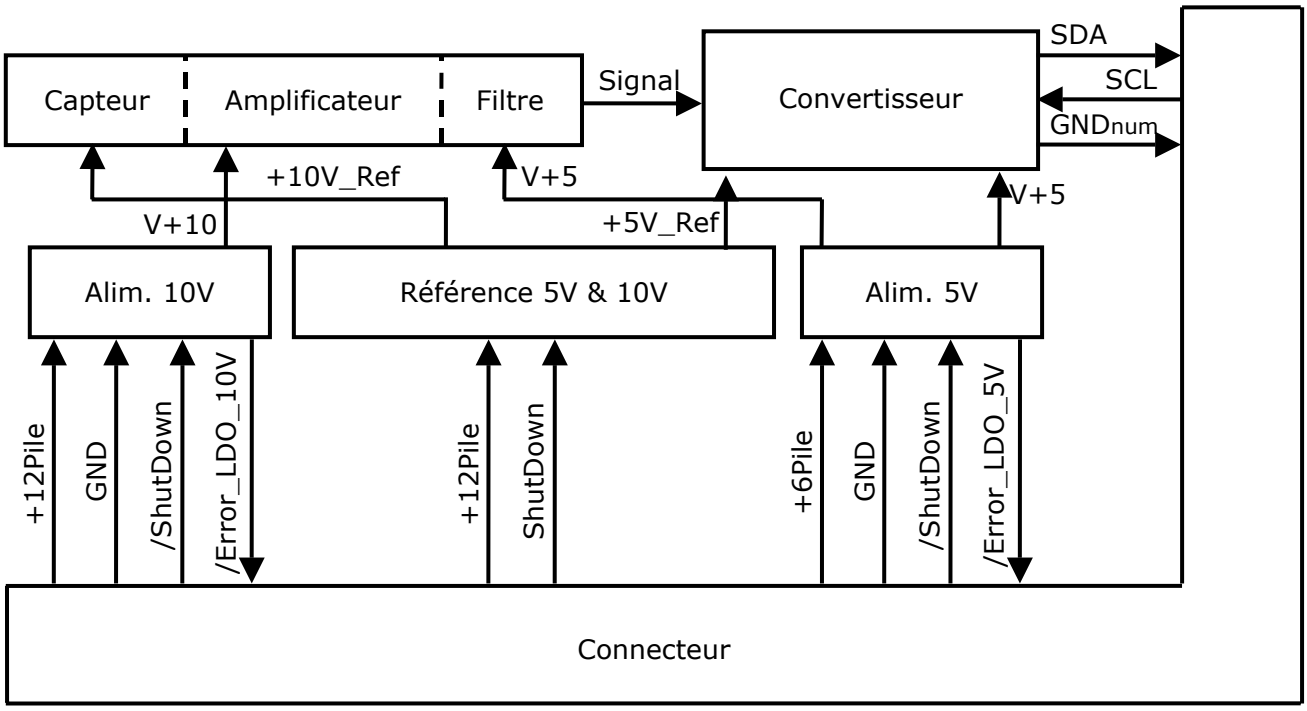


Figure 8 : Synoptique de la carte Pitot

1.4.3. Schémas électroniques

1.4.3.1. Convertisseur Analogique-Numérique.

La puce retenue pour assurer cette fonction est ici un convertisseur à 1 voie : ADS7823 de chez Texas Instruments. Les références de la datasheet de ce composant se trouvent dans le paragraphe bibliographie. Un paragraphe complet de ce rapport est consacré à la conversion analogique numérique je vous y renvoie donc pour plus d'information. On voit ici transpirer les traces de notre organisation de projet : une personne fait la carte, une personne est spécialisée dans les alimentations et apporte des solutions aux besoins particuliers de chaque carte, une personne est plus dédiée numérique avec la mise en œuvre de la conversion. Cette organisation a très bien fonctionné car nous avons communiqué en permanence et su chacun se remettre en cause : ne pas demander une alimentation symétrique si en réfléchissant un peu plus et surtout à plusieurs (Mr Carte et Mr Alim au moins) on peut trouver un solution moins contraignante pour l'autre. Au final c'est comme toujours un subtil compromis entre toute les parties. Un paragraphe de ce dossier est également consacré à la gestion de projet…

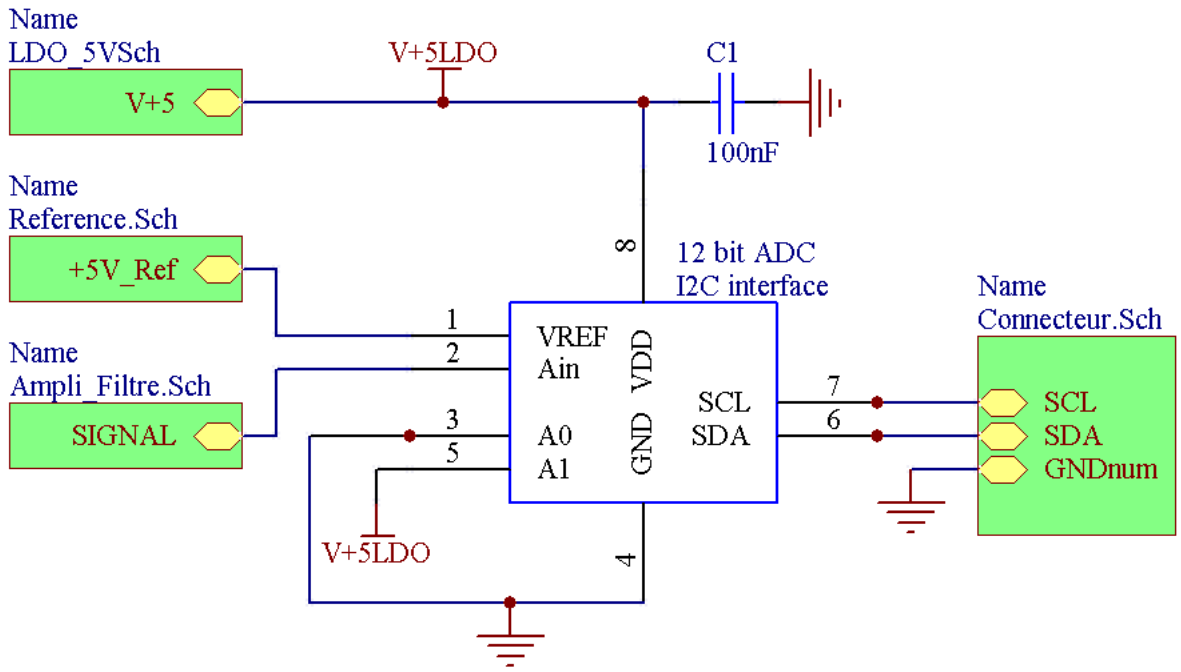


Figure 9 : Schéma du convertisseur analogique/numérique

1.4.3.2. Amplificateur et filtre

Le schéma du convertisseur contenait 1 multipattes et une capacité, celui-ci a un peu plus de composant mais se comprend toujours facilement. L'amplificateur (INA122 de Texas Instruments/BURR-BROWN) a été spécialement choisi car il fonctionne en tension non symétrique. (Une alimentation +10V -10V aurait impliqué 4piles dans la fusée car on ne voulait pas faire de montages élévateurs de tension). Le capteur de pression avait une plage de mesure de ±2psi ce qui se traduisait par une tension de sortie ±33.4 mV. L'amplificateur n'ayant pas d'alimentation symétrique est incapable d'amplifier les tensions négatives et même de les laisser passer en l'état. Ceci n'est pas un problème car comme nous l'avons vu il ne peut physiquement pas y avoir de dépression dans le tube de Pitot et donc pas de tension négative (attention à bien raccorder les prises d'air statique et dynamique afin d'évoluer dans le domaine positif en permanence et non l'inverse).

Le deuxième maillon de la chaîne est le filtre. Il s'agit d'un filtre du 8ème ordre tout intégré reposant sur une technologie dite « à capacités commutées » (MAX7400 de Maxim). Le réglage de la fréquence de coupure se fait par l'intermédiaire d'une capacité externe. Nous avions pour objectif d'échantillonner à 60Hz ce qui nous a fait choisir une fréquence d'échantillonage de 20Hz maximum qui s'est retrouvée être 17.3Hz à cause du nombre limité de valeurs de capacités disponibles sur le marché. (encore une fois bienvenue dans la vie.com)

Une fois le signal sorti de ces deux étapes, (et oui il a passé toutes les épreuves avec succès) il est maintenant possible de passer à l'acquisition analogique/numérique (voir paragraphe précédent).

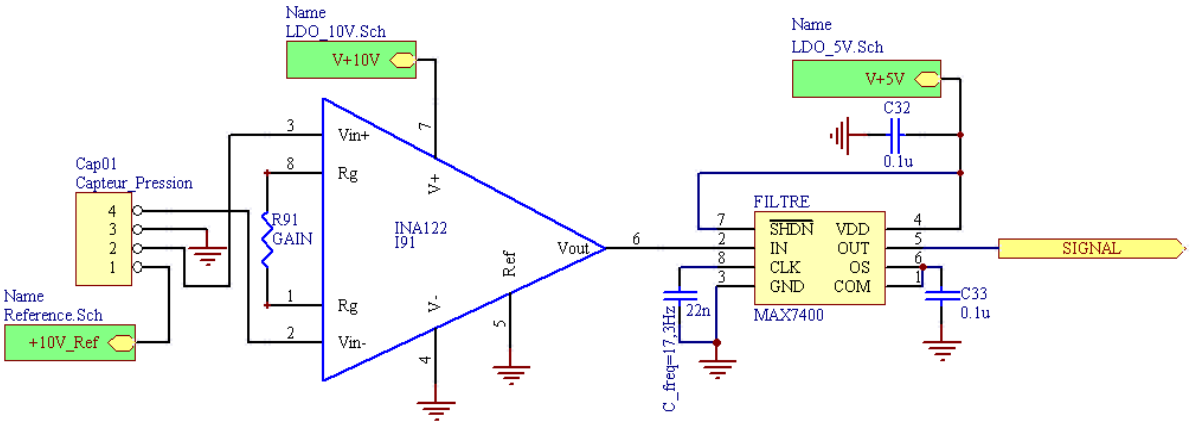


Figure 10 : Schéma capteur+amplificateur+filtre

Ce compte-rendu prend de plus en plus des allures techniques ici mais ce n'est pas une raison pour oublier qu'il doit retranscrire le projet dans son ensemble et ceci passe aussi par les bons moments qu'on a passés lors de ce projet : Par exemple, le test de la carte Pitot, et plus précisément de la partie filtre dont il est question dans ce paragraphe, lorsque nous étions en train de regarder le signal en sortie d'amplificateur (pin 6 de ce dernier) avec l'oscilloscope. Le protocole de test était simple : souffler dans le tuyau et regarder les effets sur l'écran. La carte a fonctionné quasiment du premier coup (merci Pierre), il faut dire que c'était la dernière et qu'elle était relativement plus simple que les autres. Une fois les tests finis, nous parlions de notre affaire et avons remarqué que l'oscilloscope (le capteur de pression en fait) réagissait à nos voix. En effet, les petites variations d'air dues au déplacement du son dans l'air était perçue par le capteur et ensuite amplifiées (faut dire qu'on avait un peu forcé sur le gain. Même qu'à Sissone Pierre a été tout étonné : On n'avait pas remis la bonne résistance de gain). L'idée nous est alors venu de voir ce phénomène à la sortie du filtre. Les fréquences audibles utilisées par la voix étant bien supérieures à 17.3Hz nous ne devions rien voir en sortie de filtre. Ceci s'est bien confirmé par la coupure de nos voix alors que nous voyions bien le signal lent lorsque nous soufflions dans le tuyau. Tout marchait parfaitement ce qui a encore ajouté à l'excitation collective malgré l'heure avancée.

1.4.3.3. Connecteur

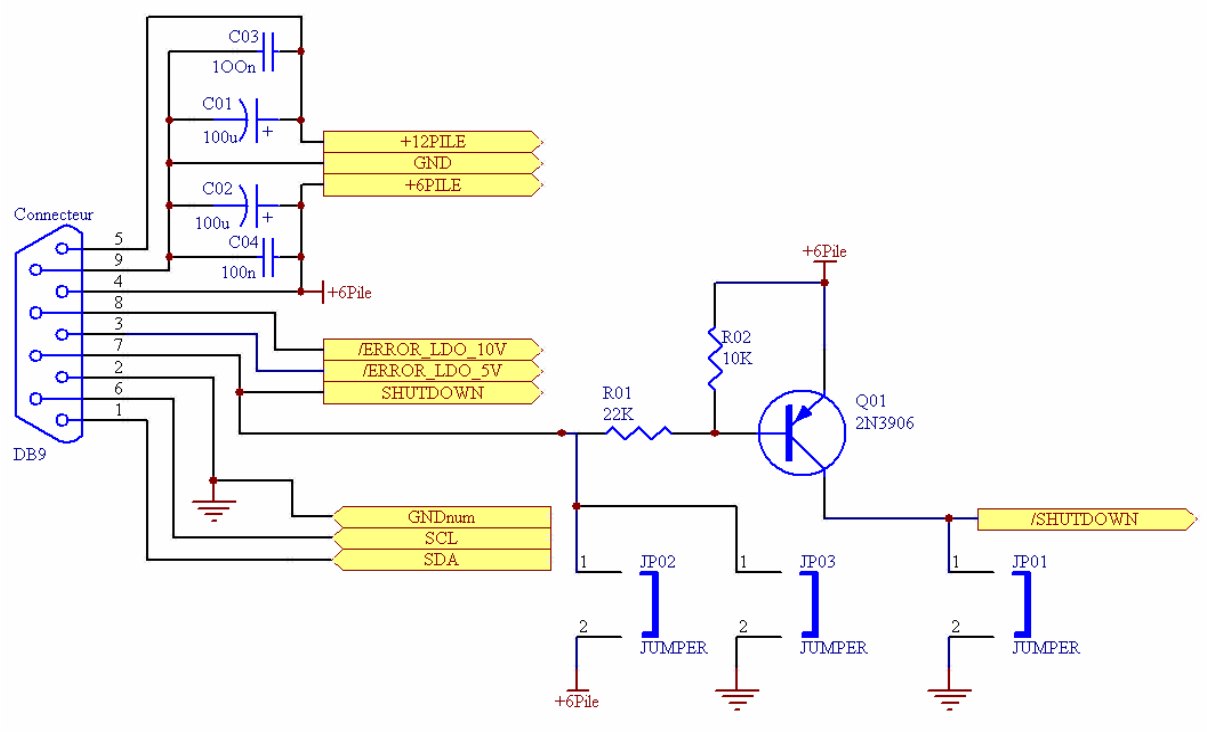


Figure 11 : connecteur d'interface de la boite blindée

Nota : La valeur de résistance R01 a été augmentée par rapport à celle indiquée sur ce schéma, de façon à garantir une bonne saturation du transistor.

Ce schéma montre le câblage du connecteur DB9 utilisé pour rendre la boite blindée communicante avec le reste de la fusée. On voit également apparaître un petit schéma à base de transistor qui permet de générer un signal inverse à la commande shutdown. Ce signal est utilisé pour activer ou désactiver les régulateurs de tension (attention pas les références qui réagissent au signal logique positif). Enfin des jumpers sont également présents sur la carte afin de shunter ce système de transistor pour les phases de développement et de mise au point.

1.4.3.4. LDO10V

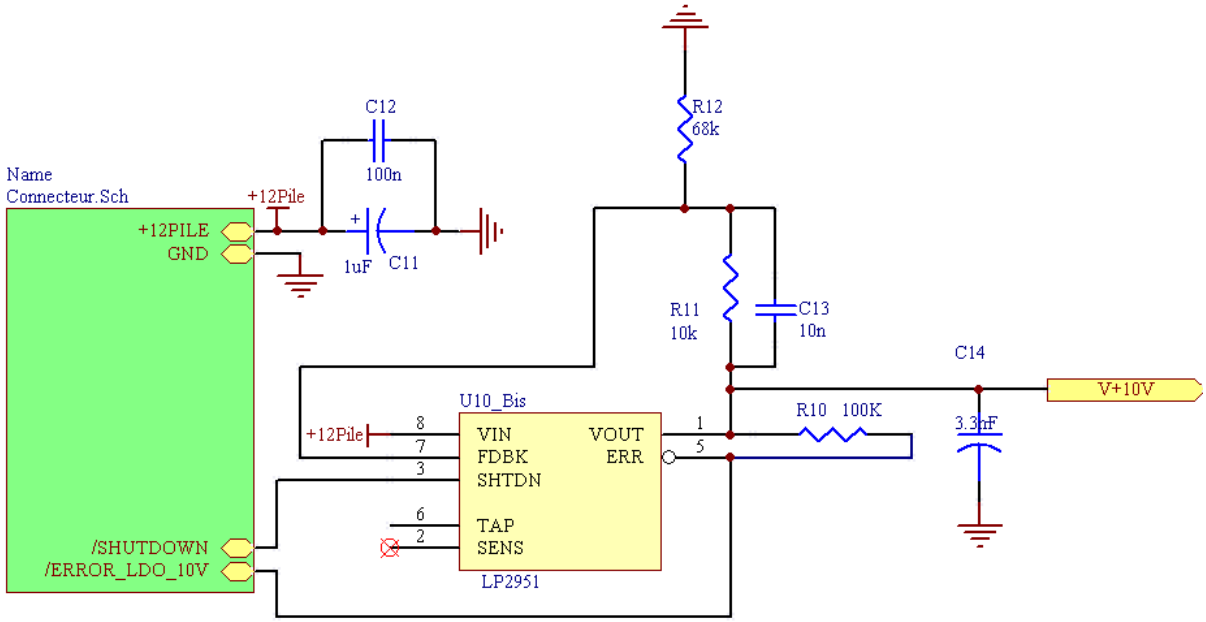


Figure 12 : régulateur de tension +10V

Voici le schéma du régulateur utilisé pour générer le +10V nécessaire au fonctionnement de l'amplificateur unipolaire utilisé dans ce circuit. Tous les détails, subtilités et autres astuces sont donnés dans le paragraphe de ce dossier qui concerne les alimentations.

1.4.3.5. LDO5V

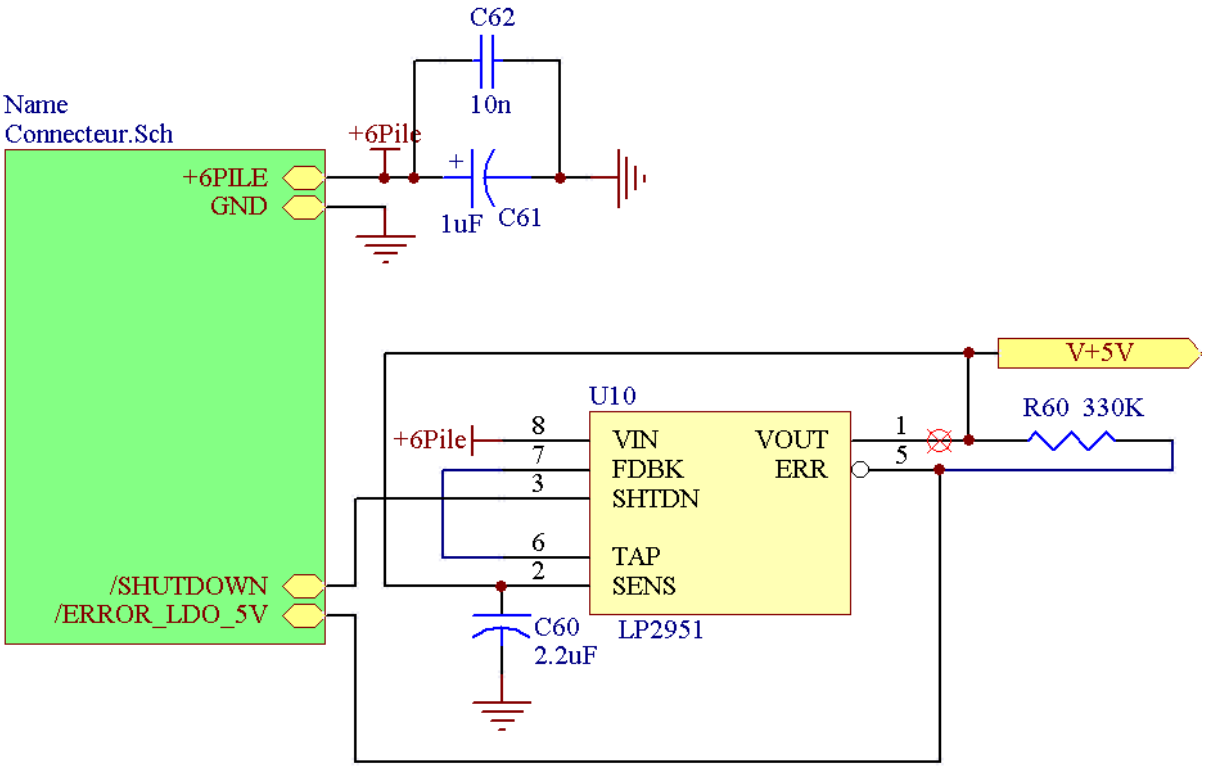


Figure 13 : régulateur de tension +5V

Cette alimentation reprend exactement le même composant de base que l'alimentation +10V. Elle a cette fois pour tâche d'alimenter le filtre et le convertisseur. Comme dans le paragraphe précédent je vous renvoie au paragraphe sur les alimentations pour plus de détails.

1.4.3.6. Référence 5V et 10V

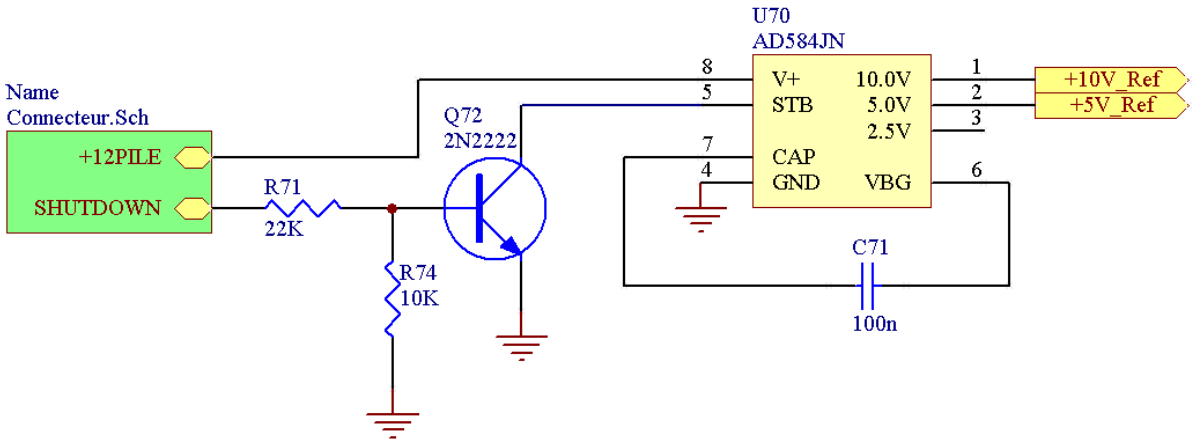


Figure 14 : régulateur de tension +10V

Pour changer un peu, nous parlons maintenant de référence de tension qui rappelons-le n'ont pas vocation à débiter du courant mais seulement à maintenir des potentiels de façon très précise. Les deux références sont élaborées à partir du même composant : le célébrissime AD584. Malgré les quelques tours de cochon (voir paragraphe alimentations) que celui-ci nous a joué il faut admettre qu'il est particulièrement adapté à notre besoin ici. La référence de tension +10V est réservée au capteur lui-même alors que la référence +5V est dédiée au convertisseur. Le fait que les deux références soient issues de la même puce présente un avantage évident pour la fabrication mais ce n'est pas tout. En effet, on peut espérer que si une référence varie (dans la limite de ses quelques dixièmes pourcents autorisés) l'autre variera de la même façon. Ainsi les micro-variations pourront se compenser quelque peu : le capteur codera sa tension à partir d'une alimentation 10V (+/- 30 mV) mais le convertisseur aura alors une plage de codage réduite du même facteur c'est-à-dire 0 à 5V (+/- 15mV). (Données Analog Devices pour l'AD584J, c'est à dire la pire et celle que nous avions en samples, meuh bon ça marche quand même pas mal). En plus c'est très logique : ça découle de la relation du pont diviseur qui est en sortie de l'AD584 : quand le 10V varie de 30mV et ben le 5V lui il varie de 15 mV, puisque le coefficient diviseur s'applique aussi à la variation de la tension. J'ai fait le calcul pour voir : c'est très simple.

1.4.4. Bilan de consommation

Le tableau suivant présente le bilan de consommation de la carte Pitot. Après réalisation de ce bilan, il est possible d'estimer l'autonomie de cette expérience, qui elle-même entrera en compte dans le calcul de l'autonomie de la fusée.

	Consommation moyenne	Conso. Max théorique
LDO +10V (Ampli)	60 µA	85 µA
LDO +5V (Filtre + Convertisseur)	2 mA (Filtre) + 240µA (Convertisseur en mode « Standard » sous 5V)	3.5 mA (Filtre) + 380µA (Convertisseur en mode « hi-speed » sous 5V)
Référence +5V (Convertisseur)	20 µA	
Référence +10V (Capteur)	1.33 mA	

Figure 15 : bilan de consommation interne de la carte, d'après les données constructeurs.

Après mesures, la carte consommait 2.24mA sur le 12V c'est-à-dire sur deux piles 6V en série et 4.34mA sur le 6V (une des deux piles). Si cette dernière consommation se situe plutôt dans le haut de la fourchette que nous venons de donner, c'est qu'il ne faut pas oublier d'inclure la consommation du transistor 3906 assurant la commande des régulateurs et du transistor 2222 pour la commande de la référence AD584. En effet ces consommations ne sont pas négligeables : un transistor en saturation « pompe » du courant, de plus celui-ci dépend de la température, de la qualité du composant et aussi –à moindre échelle tout de même- du vieillissement du transistor choisi. Ce système de commande a fonctionné sans faillir, néanmoins il sera vraisemblablement à l'avenir remplacé par une logique CMOS à cause de cette consommation un peu gênante.

Pour déterminer l'autonomie de l'expérience nous avons besoin de la capacité initiale des piles (le vol s'effectuant avec des piles neuves) mais aussi de la consommation des autres expériences « pompant » sur ces piles, dans notre cas : le système BOG. Le schéma de câblage était le suivant :

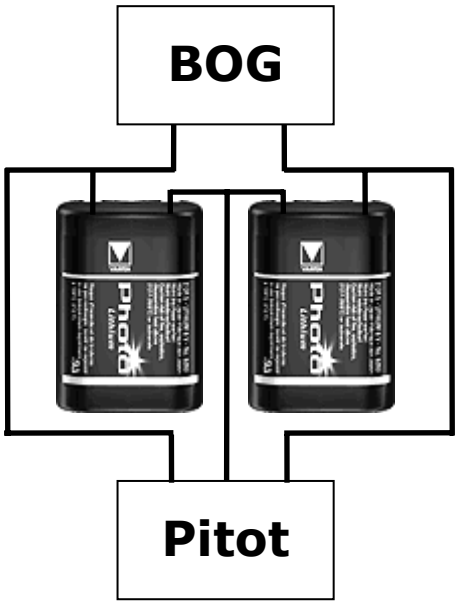


Figure 16 : câblage des piles de l'expérience Pitot

	Pile n°1 (6V)	Pile n°2 (6V et 12V)
Capacité initiale des piles	1.3 A.h	1.3 A.h
Consommation carte Pitot	4.34 mA	2.24 mA
Consommation carte BOG	Env.2 mA	50mA(chauffage)
Autonomie de ces expériences	25 heures environ	

Figure 17 : calcul de l'autonomie de l'expérience Pitot

Comme on peut le constater, l'autonomie de notre expérience est bien supérieure à la limite des 45 minutes minimum imposée par le cahier des charges. Il faut pourtant garder à l'esprit que ce sera la partie électronique ayant la plus faible autonomie qui définira la véritable autonomie de la fusée (principe du maillon faible comme le dirait Laurence B).

1.5. Etalonnage

On applique sur une membrane la pression atmosphérique (statique) d'un coté et la pression due à la vitesse (dynamique) de l'autre.

Après calcul et dimensionnement du tube de Pitot, on sait qu'une pression relative de 17700 Pa subie par la voie dynamique correspond à la vitesse maximale (170 $m.s^{-1}$ = 612 $km.h^{-1}$) de la fusée : vitesse qui ne peut pas être atteinte lors de l'étalonnage voiture (même avec une BX survitaminée au LHM).

On utilise donc une seringue sur laquelle on viendra appuyer pour augmenter la pression. En faisant l'hypothèse (très raisonnable) que l'air est un gaz parfait, on obtient l'égalité suivante :

$$P \times V = n \times R \times T = \text{constante} \quad \textbf{\textit{(Equation 3)}}$$

Unités : $[m^{-1}][kg][s^{-2}] \times [m]^3 = cst$

Où n.R.T est constant, P est la pression du gaz et V le volume occupé par celui-ci. Donc, une augmentation de pression se traduit par une diminution du volume du gaz. Il faut donc savoir de combien on doit diminuer le volume occupé par l'air dans la seringue (et tuyaux) pour augmenter la pression de 17700 Pa.

Pour cela, on doit connaître le volume contenu entre le poussoir de la seringue et la membrane du capteur. C'est un tuyau flexible qui relie la seringue au capteur (Note : Pour s'affranchir des fuites éventuelles, on utilise de la pâte isolante entre les différents éléments.)

Le volume d'air total (volume de référence) contenu dans la seringue et dans le tuyau est estimé à 18,35 ml. A partir de l'équation 3 on peut écrire :

$$(P_{ref} + \Delta P) \times (V_{ref} - \Delta V) = cst$$

Avec :

P_{ref} : 1.10^5 Pa pression terrestre (note pour un futur projet : prévoir le matériel pour mesurer la pression atmosphérique du moment)

ΔP : la variation de pression imposée à l'étalonnage

V_{ref} : volume initial contenu dans l'ensemble seringue + tuyau et

ΔV : variation de volume nécessaire dans la seringue pour l'augmentation de pression desirée

Après calculs, on obtient : ΔV_{Max} = Vref / 5 (on a pris une marge Vitesse Max = 200 $m.s^{-1}$ soit 25000Pa) Soit ΔV_{Max} = 3,67 ml.

Le gain de l'amplificateur doit alors être ajusté afin d'obtenir une tension de sortie comprise entre 0 et 5 Volts.

Le gain de l'amplificateur est donné par la formule suivante :

$$G = 5 + 200.10^3 / Rg \quad \textbf{\textit{(Equation 4)}}$$

Où Rg est la résistance de gain (en ohm).

La sensibilité du capteur est de 16,7 $mV.psi^{-1}$, soit 60,54 mV pour une pression relative de 25000 Pa. Pour une tension de sortie de 5 V, le gain est donc de ×82.

Note : Rappelons que pour passer d'une pression en bar à une pression en psi, il faut multiplier par un facteur 14,503.

A partir de l'**Equation 4**, on déduit la résistance de gain Rg = 2 577 ohm. (Soit 2,7 kohm normalisé).

L'étalonnage se fait alors sur 5 points, d'un volume réduit de 0 ml à un volume réduit de 3,67 ml. La tension analogique se prend en sortie d'amplificateur et les valeurs numériques correspondantes sont traitées (acquisition sur quelques seconds et moyennage) sur portable, via un bus I²C. Les résultats sont reportés dans le tableau et le graphe ci-dessous.

Diminution du volume (ml)	Tension sortie ampli (V)	valeur numérique	Vitesse en $m.s^{-1}$
0	0,08	20	0
1	1,41	1213	97
2	2,68	2415	141.3
3	4,16	3560	178.6
3,67	4,96	4072	202
Saturation à	5	4096	

Figure 18 : Valeur étalonnage tube Pitot

La relation donnant la vitesse, est obtenue à partir de :

$P_{ref} \times V_{ref} = (P_{ref} + \Delta P)(V_{ref} - \Delta V)$ et $\Delta P = \frac{1}{2}\rho V_{itesse}^{\ 2}$

$$Vitesse = \sqrt{\frac{2}{\rho} \times \left(\frac{(P_{ref} \times V_{ref})}{(V_{ref} - \Delta V)} - P_{ref} \right)}$$

$$Vitesse = \sqrt{\frac{1}{[kg][m]^{-3}} \times \left(\frac{[m]^{-1}[kg][s]^{-2} \times [m]^3)}{[m]^3} - [m]^{-1}[kg][s]^{-2} \right)} = \sqrt{\frac{[m]^{-1}[kg][s]^{-2}}{[Kg][m]^{-3}}} = \sqrt{[m]^2[s]^{-2}}$$

Ensuite nous cherchons la relation qui lie ΔV à la valeur Numérique Val_{12b} afin de pouvoir remplacer ΔV dans l'équation précédente.

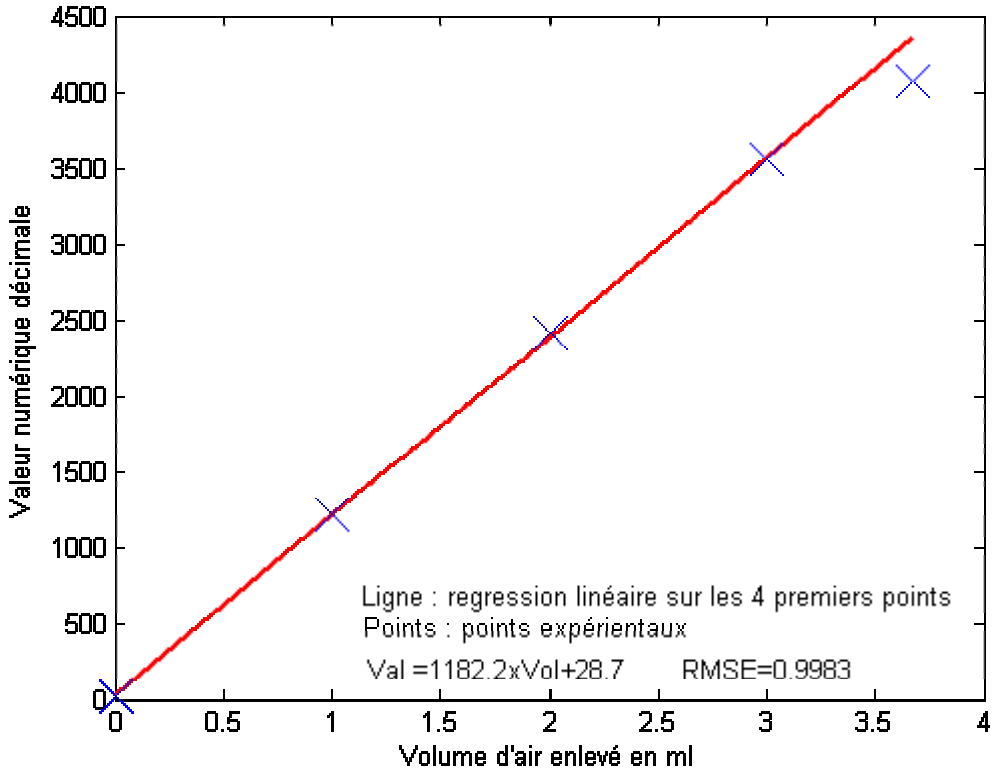


Figure 19 : Courbe figurant l'évolution des valeurs numériques en fin de chaîne d'acquisition en fonction de la réduction de volume imposée.

La régression linéaire suivante montre clairement une très bonne linéarité de nos 4 premières observations puis une légère dégradation à la fin. Ceci a deux explications :

- Il est beaucoup moins facile de réaliser une réduction de volume de 3.67ml qu'une de 1, 2 ou 3 grâce aux graduations.
- Le capteur arrive alors en limite de plage de mesure et n'est plus linéaire. En effet, une réduction de 3.67ml équivaut à une pression de … 3.7psi pour un capteur de dynamique symétrique de 2psi !

Au final, on a donc :

$$Vitesse = \sqrt{\frac{2}{\rho} \times \left(\frac{(P_{ref} \times V_{ref})}{(V_{ref} - (\frac{Val_{12b} - 28,7}{1182,2 \times 10^6})} - P_{ref} \right)}$$

Avec :

ρ =1.225 Kg.m^{-3}

Vref = 18.35.10^{-6}m^3

Pref = 1.10^5 Pa

Val12b = valeur mesurée par la chaîne exprimée en décimal (12bits)

1.6. Exploitation des résultats du Pitot

Le vol nominal de aXelle nous a permis de recueillir : les valeurs la FSK et de la mémoire (MEM). Des problèmes (voir chapitre 08) de fréquence d'acquisition nous ont obligés à sous échantillonner les valeurs de la mémoire pour correspondre à celles de la FSK. Nous disposons donc de 2 jeux de 246 valeurs chacun. La fréquence d'échantillonnage est pour ces deux jeux de : 20 Hz soit les 12.15 secondes de vol avant le largage du module. Nous disposons également des valeurs théoriques de Trajec : 290 échantillons sur tout le vol balistique (cas où le parachute ne s'ouvrirait pas) à une fréquence de 10Hz. Il n'y a donc que les 125 premières données de Trajec qui correspondent à nos mesures.

Note : le pas de calcul de Trajec est configurable, nous aurions donc pu avoir plus ou moins de valeurs ou autant que l'expérience.

Les valeurs de la FSK étaient transmises sur 8 bits pour des raisons évidentes de débit. Une fois les valeurs remises à la même échelle (×16) on remarque que 28 valeurs (9.7%) diffèrent de plus de 16 en décimal, alors qu'en théorie les valeurs de la FSK ne devraient pas différer de plus de 16 par rapport aux données de la mémoire, l'erreur entre les deux étant dûe uniquement au fait que l'on enregistre une valeur sur 12 bits et que l'on transmet une valeur en 8 bits par la FSK. Nous remarquons rapidement qu'il semble y avoir plus d'erreurs au début du vol (2.5 premières secondes) puis une constance dans le taux d'erreur (entre 2.5sec et 8.5sec) et enfin une raréfaction sur la fin. De plus, le signe de l'erreur : ($Valeur_{FSK}$ - $Valeur_{MEM}$) semble également lié à ces périodes. Une explication pourrait être les contraintes subies par l'antenne boudin lors de la phase de poussée du vol. (cf – photo en **Figure 17**)

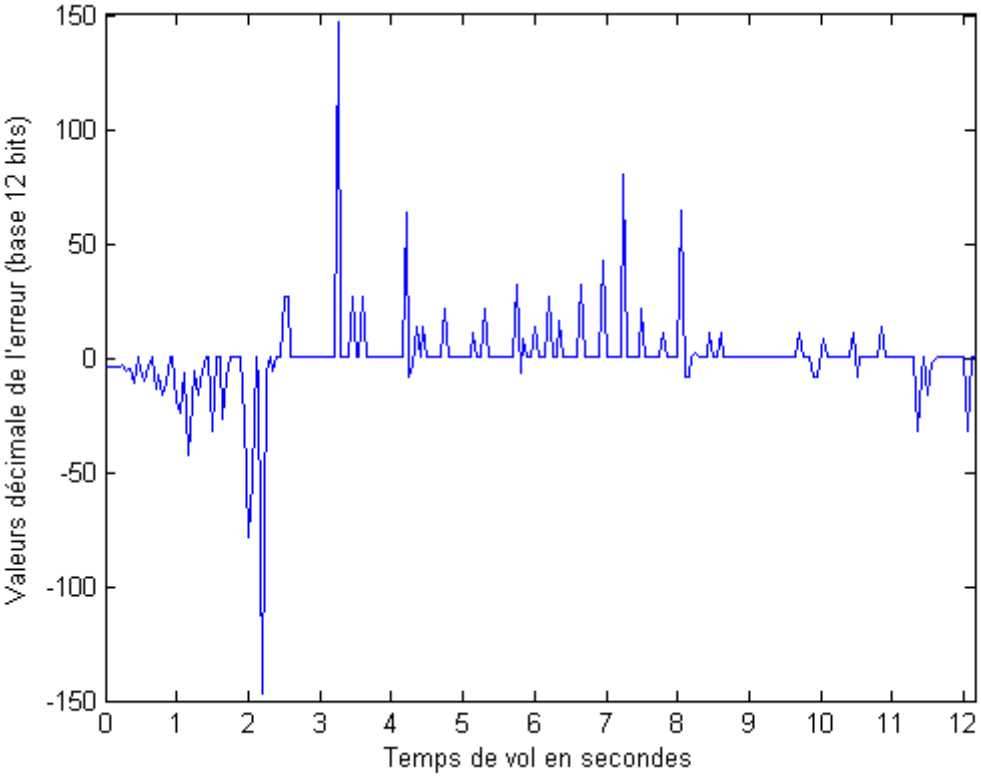


Figure 20 : Les erreurs de transmissions (Valeur FSK – Valeurs MEM) en fonction du temps

Figure 21 : Décollage aXelle – déformation de l'antenne de transmission

Cette même antenne doit se retrouver tordue en sens inverse lors de la décélération. Or, on sait que la peau en carbone de la fusée est suffisament conductrice

pour jouer un rôle de plan de masse. Lorsque l'antenne se tord et s'approche du carbone, son lobe d'émission est alors déformé. De plus, comme la fusée tourne sur elle-même lors du vol, aXelle ne disposant que d'une seule antenne boudin, le « cône » d'émission n'est pas toujours tourné vers l'antenne réceptrice. Une bonne video du vol de la fusée permettrait de tenter une comparaison entre position de l'antenne et erreurs.

Techniquement, nous pouvons conclure que la FSK et la mémoire ont parfaitement donnée satisfaction, mais observons maintenant le phénomène physique. (Saint graal de tout expérimentateur, le pourquoi de toute cette aventure)

1.6.1. Résultats

La grandeur physique que traduit le tube de Pitot est la vitesse. Cette première courbe met donc en parallèle les résultats de notre capteur tube de Pitot (expérience) et la courbe de vitesse donnée par Trajec (théorie).

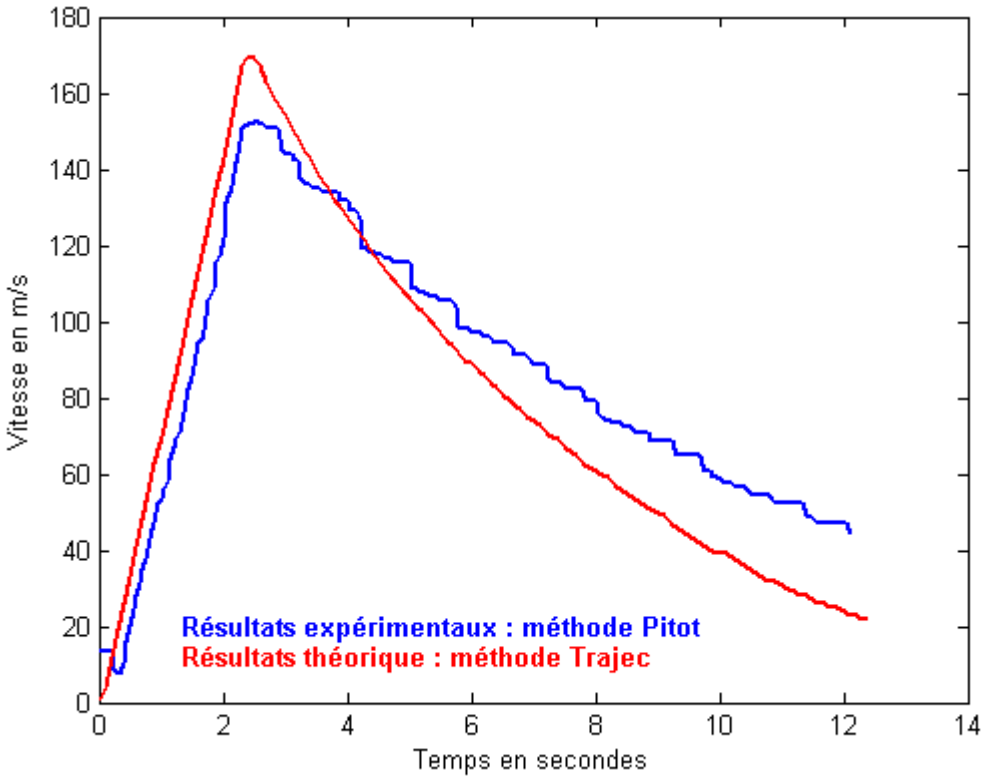


Figure 22 : Evolution de la vitesse en fonction du temps et comparaison théorie/expérience

On remarque que l'allure des courbes est assez similaire, surtout sur le début lors de la phase d'accélération. Les dynamiques sont également proches, vitesse max de 170m.s^{-1} pour l'une et 150m.s^{-1} pour l'autre.

1. La légère accroche dans la courbe sur les 8 premiers échantillons correspond à un léger problème d'étalonnage. En effet, la formule pour retrouver les vitesses comprend une racine carrée et pour les premières valeurs la racine carrée doit être appliquée à des valeurs négatives d'après l'étalonnage, ce qui est bien entendu impossible.
2. On remarque également que la courbe est plus saccadée lors de la descente (décélération) que lors de la monté (accélération) et que la décélération est moins forte que la théorie ne le prévoit. Une des explications pourrait être que notre Cx était meilleur que le Cx entré dans Trajec.

Note : Ce chiffre pourtant très influent pour les calculs est toujours saisi d'après la méthode du hasard constant c'est-à-dire égal à 0.8.

Afin de vérifier cette hypothèse et pour voir l'influence du Cx sur la vitesse de la fusée nous réalisons plusieurs simulations Trajec avec comme seul paramètre variable le Cx. (Les autres paramètres sont pris égaux au paramètres intrinsèques de la fusée).

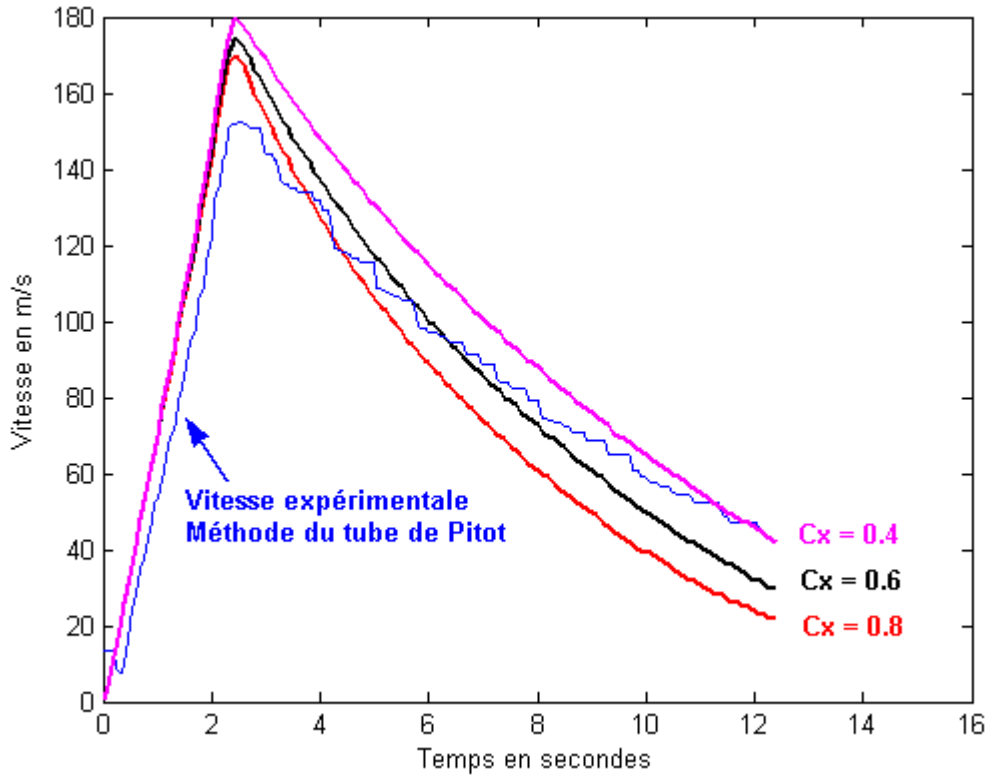


Figure 23 : courbes théoriques de vitesse pour différentes valeurs de Cx

L'erreur de décélération n'est pas compatible avec l'hypothèse de la saisie d'un mauvais Cx. En effet, nous avons réalisé des simulations dans la plage de Cx qui nous semblait réaliste mais l'évolution de ces dernières ne montre aucune convergence vers nos résultats. Il nous faut donc faire une nouvelle hypothèse que voici (en espérant ne pas être tués par Bernoulli en personne avant la fin de cette explication) : Nous pensons qu'un tube de Pitot marche mieux en accélération qu'en décélération. Cette pensé est stimulée par un exemple simple : une bouteille d'eau se remplit facilement alors qu'elle se vide par saccades à cause des problèmes d'entrée d'air. Et bien, notre hypothèse retranscrit ce phénomène au tube de Pitot : accélération = remplissage = sans retard et sans à-coups, décélération = vidage = retard et à-coups. De plus les tubes flexibles en plastique ne doivent qu'amplifier le phénomène. Au final, cette hypothèse nous oriente vers une décélération plus lente et avec des saccades comme nous pouvons le voir sur les graphiques.

3 Enfin, il est vrai que le tube de Pitot ne fonctionne pas bien pour des vitesses faibles (20-30 $m.s^{-1}$), mais ce n'est pas notre terrain de jeu ici. Par contre, un tube de Pitot est très bon pour mesurer un écoulement stationnaire mais pas un écoulement transitoire. Notre pression (vitesse) n'étant jamais fixée et stable nous ne sommes jamais dans les conditions idéales pour ce capteur (état stable). Le flux d'air subit une certaine inertie du système : la pression locale à l'entrée du tube va être instantanée, mais la pression globale dans le

tube et au niveau du capteur aura plus d'inertie. De tels défaut n'apparaissent pas dans l'anémomètre à fil chaud (système Bog) que nous avons mis en œuvre : Il n'y a pas de modification de l'inertie, le capteur est au cœur du flux.

Dans l'objectif global de la fusée, l'expérience première était de comparer différentes méthodes de mesure d'accélération. Même si nous savons déjà que cette méthode de mesure de vitesse par Pitot n'est pas excellente pour une mesure d'accélération, allons tout de même au bout de notre démarche et observons la dérivée de cette courbe de vitesse.

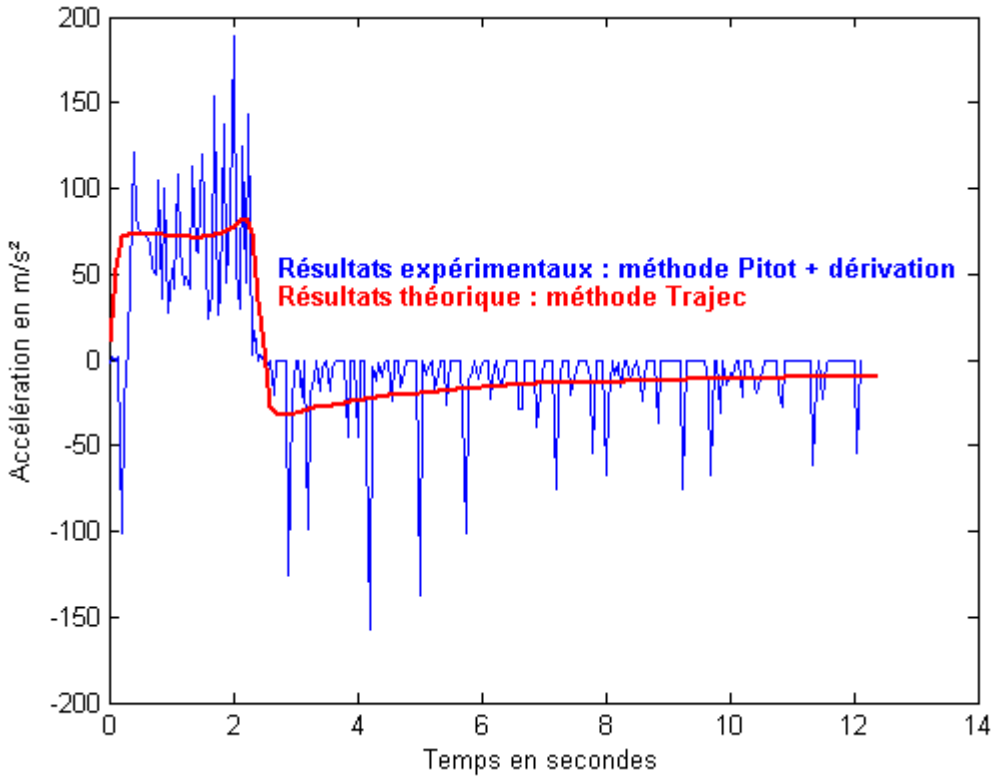


Figure 24 : Evolution de l'accélération et comparaison théorie/expérience

Comme le montre la courbe, la dérivation donne de très mauvais résultats. En effet, nous avons totalement négligé le fait que pour pouvoir dériver une courbe et avoir encore des résultats exploitables il fallait que cette dernière soit échantillonnée beaucoup plus rapidement (l'objectif réel est de minimiser les dérivées locales). Le « rapidement » dépend de la rapidité de variation du phénomène que l'on souhaite observer et de la qualité de la chaîne d'acquisition (Note pour un autre projet : dans le cas de l'observation d'un phénomène au travers de son intégrale, il faudra faire très attention) De plus, cette mauvaise dérivabilité confirme la mauvaise mesure en instantané.

Somme toute et bien que les valeurs ne soient pas exploitables (même avec un filtrage d'ordre 10), on remarque que l'allure générale de la courbe théorique est respectée. La méthode de mesure d'accélération par tube de Pitot ne pourra donc pas participer à l'exploitation finale des résultats qui visera à comparer l'ensemble des méthodes.

1.6.2. Conclusion

Un capteur tube de Pitot est relativement simple à mettre en œuvre du point de vue électronique. Par contre, c'est un capteur qui a une forte composante mécanique qui doit être réalisée avec le plus grand soin. Cette même partie mécanique demande de bien prévoir l'intégration à l'avance car elle est assez imposante avec des circulations de tuyaux pas toujours évidentes.

Les résultats ont montré que cette méthode n'était pas parfaite pour la mesure des vitesses instantanées mais que malgré tout l'allure globale était bien respectée. Par contre lorsque nous avons voulu utiliser cette méthode pour observer l'accélération nous nous avons été confrontés à un problème de sous-échantillonnage. S'il y a bien un conseil à retenir de aXelle c'est qu'une fusée est un tout : un seul élément de la chaîne d'exploitation ou de la fusée laissé au hasard (pas assez d'attention) et on le paye 100 fois à l'exploitation des résultats ou à la campagne !

1.7. Bibliographie

[1] Note technique Planète Sciences : Tube de Pitot (Projet Cyrius club ESO) – Edition Octobre 1999 - sur CD accompagnant ce rapport ou sur le CD technique secteur espace ou encore en ligne à l'adresse suivante :

http://www.planete-sciences.org/espace/publications/fichiers/tube_de_pitot.pdf

[2] Datasheet sommaire de la série de capteurs de pression différentiel utilisé sur aXelle – en ligne à l'adresse suivante :

http://catalog.sensing.honeywell.com/printfriendly.asp?FAM=Pressure&PN=26PCFFA6D

[3] Datasheet ADS 7823 convertisseur 12bits avec interface I²C, basse conso :

http://focus.ti.com/lit/ds/symlink/ads7823.pdf

[4] Datasheet INA122 amplificateur d'instrumentation bipolaire utilisé en alimentation unipolaire dans notre cas :

http://focus.ti.com/lit/ds/symlink/ina122.pdf

[5] Datasheet MAX7400 filtre du huitième ordre – technologie capacités commutées – très facile d'utilisation pour un résultat dont Francis Lesel se souvient encore …

http://pdfserv.maxim-ic.com/en/ds/MAX7400-MAX7407.pdf

[6] Datasheet LP2951 – régulateur de tension 3.3V/5V - 100mA – commandable et avec retour d'information sur le statut. Technologie LDO : faible tension de chute

http://www.onsemi.com/pub/Collateral/LP2950-D.PDF

[7] Datasheet AD584 – référence de tension : 2.5V/5V/7.5V/10V – un incontournable dont il faut se méfier…

http://www.analog.com/UploadedFiles/Data_Sheets/115302565AD584_b.pdf

[8] Datasheet 2N2222 – transistor le plus célère de la planète en type NPN : multi-usage, multi-connerie, multi-montage, multi-bidouille… Utilisé ici pour le pilotage logique des régulateurs de tension.

http://www.semiconductors.philips.com/acrobat/datasheets/2N2222_CNV_2.pdf

[9] Datasheet 2N3906 – il est au 2N2222 ce que l'antimatière est à la matière : transistor baroudeur type PNP. Il est utilisé aussi ici pour les commandes logiques

http://www.semiconductors.philips.com/acrobat/datasheets/2N3906_3.pdf

Note : La liste des composants qui figure dans cette liste de datasheets constitue quasiment l'exclusivité de l'alphabet génétique d'aXelle pour l'électronique. On commence à bien avoir éprouvé ces petites bestioles, n'hésitez surtout pas à vous en resservir s'ils correspondent à vos besoins.

http://www.insa-lyon.fr/Associations/ClesFacil/

Nicolas Chaléroux : nico@sdbdc.org

(Conception mécanique, conception et CAO électronique, observateur réalisation mécanique)

Pierre Fayet : fayetpierre@aol.com

(Conception électronique, réalisation électronique)

Laurent Ruet : laurent_ruet@hotmail.com

(Conception mécanique, exploitation des résultats)

Projet : Axelle

5 Décembre 2003

Version 5.0

02. Accéléromètre à Jauges

L'accéléromètre à jauges tient une place particulière dans le cadre expérimental global d'aXelle. C'est un capteur entièrement fait main qui a pour ambition une très grande précision. Nous avons cherché à maîtriser chaque paramètre du capteur qu'il soit physique ou électrique. Ce capteur met en œuvre des méthodes classiques de métrologie en se basant sur des jauges de contraintes. Il faut bien sûr voir dans ce capteur un réinvestissement des techniques apprises durant le projet ELA 2002 à propos de ces fameuses jauges de contraintes.

1. Objectifs

L'objectif est toujours de mesurer l'accélération et de mettre en œuvre un autre moyen de mesure de ce phénomène. Les contraintes principales à mettre en regard avec nos « ambitions » sont toujours :

- de pouvoir intégrer ce capteur dans la fusée à l'intérieur d'une boite blindée électromagnétiquement.
- d'assurer une autonomie suffisante afin que ce capteur ne soit pas le maillon limitant de l'autonomie globale de la fusée.
- d'avoir un capteur respectant la philosophie de la fusée : nos boites blindées sont nos capteurs et communiquent avec le reste de la fusée suivant des interfaces prédéfinies : Bus I²C, commande de ON/OFF, retour d'information sur l'état des bateries…
- de garder durant tout notre travail gardé un œil sur le budget afin de pouvoir mener tout au long de l'année les deux projets expérimentaux du club de front : Giro le ballon et aXelle la Fusex.

Contrairement aux autres capteurs, la théorie mécanique a une place beaucoup plus importante. Ceci tient au fait que pour une fois elle ne nous est pas dissimulée dans un capteur tout intégré du marché. Nous avons donc dû aller un cran plus loin et travailler ensemble étroitement : électronicien et mécanicien.

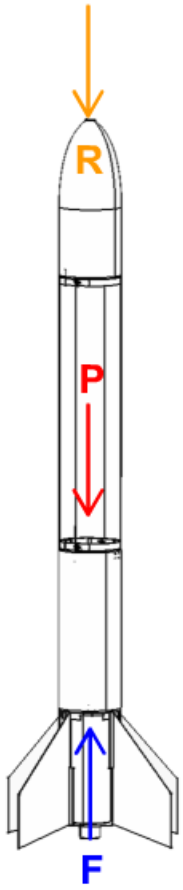
2. Mécanique

2.1 Introduction

Mécaniquement, le capteur mis en place à l'aide de jauges de contraintes a pour but d'utiliser les efforts que subissent les différentes pièces de la fusée pendant le vol. Nous allons voir que ces efforts sont directement liés à l'accélération subie par la fusée par des relations très simples. Ces efforts créent par la même des déformations que nous somme capable de mesurer, afin de retrouver l'accélération recherchée.

2.1.1 Bilan des forces

Nous allons voir ici que les efforts, ou forces internes, dans la fusée sont directement liés à l'accélération de cette dernière. Dans un premier temps, nous allons nous habituer à ce type de calcul très simple par un bilan global :



3 forces agissent sur l'ensemble de la fusée :

- Force de poussée
- Poids de la fusée
- Résistance de l'air

On effectue ensuite un simple bilan des forces

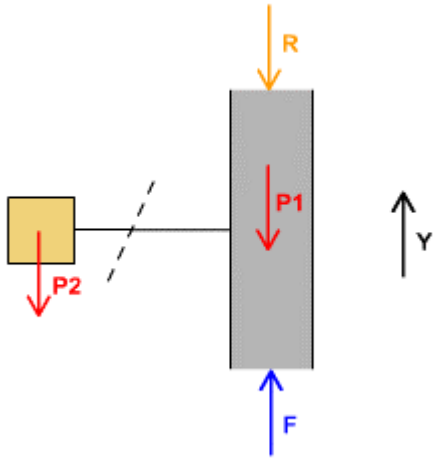
$$F - P - R = F - Mg - \frac{1}{2}\rho SCxV^2 = M\gamma$$

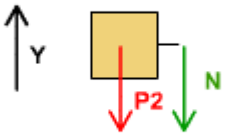
où :

- M : Masse de la fusée
- g : constante de gravitation
- S : Section normale
- Cx : Coefficient de résistance de l'air
- ρ : masse volumique de l'air
- γ : accélération de la fusée

Nous allons donc nous intéresser de plus près aux efforts internes à la fusée et notamment dans le cas de notre capteur. Ce capteur est composé d'une petite masse reliée à la fusée par une lamelle en métal. C'est sur cette lamelle que nous cherchons à déterminer les efforts.

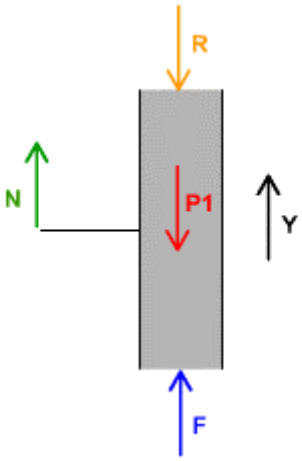
La technique permettant de déterminer les efforts internes consiste à découper la structure en deux et à faire le bilan des forces de chaque coté de ce découpage afin de déterminer les efforts internes à l'endroit du découpage. Dans notre cas nous ne nous intéresserons qu'aux forces et non aux couples.





$$-N - P_2 = m_2\gamma$$

$$N = -m_2(g+\gamma)$$



$$N + F - R - P_1 = m_1\gamma$$

$$N = m_1\gamma - F + R + Mg - m_2 g$$

$$N = m_1\gamma - M\gamma - m_2 g$$

$$N = -m_2(g+\gamma)$$

2.1.2 Relation entre déformation et accélération

ATTENTION : désormais nous noterons « a » l'accélération globale de la fusée (γ+g).

L'effort sur la lamelle a donc été calculé et est donc directement proportionnel à l'accélération de la fusée « a » et à la masse de la masselotte. Ces efforts ne sont pourtant pas directement mesurables, en effet, nous ne pouvons mesurer uniquement les déformations induites par ces efforts. Nous allons donc nous intéresser au calcul de cette déformation :

Il s'agit d'un simple calcul de RdM [1]:

Force	$F = m_2 a$
Moment fléchissant	$m_2 f = \frac{FL}{2}$
Contrainte sur la lame	$\sigma = \frac{m_2 f}{Iyy}\frac{H}{2} = \frac{3 \cdot m_2 \cdot L \cdot a}{B \cdot H^2}$
Déformation unitaire des jauges	$\varepsilon = \frac{\sigma}{E} = \frac{3 \cdot m_2 \cdot L \cdot a}{E \cdot B \cdot H^2}$

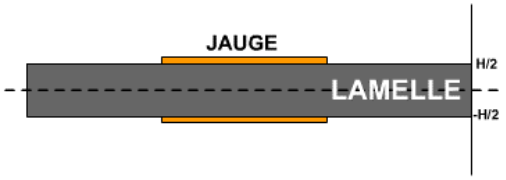
Avec

- m2 : Masse de la masselotte
- L : Longueur de la lamelle
- a : accélération à mesurer
- E : Module d'Young
- B : Largeur de la lamelle
- H : Epaisseur de la lamelle

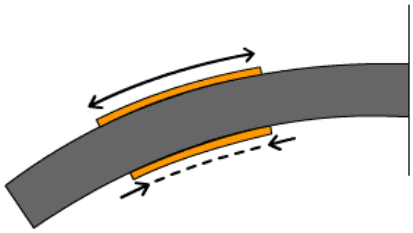
La déformation que nous allons mesurer dépend donc bien de l'accélération, mais aussi de la géométrie du capteur et des matériaux utilisés. Cette déformation peut être mesurée grâce à des jauges de contraintes collés sur la surface de la lamelle.

Remarque :

Il est important de remarquer la présence de H/2 dans l'expression de la contrainte dans la lame. Ce terme correspond à la position pour laquelle nous calculons la contrainte, et ce par rapport à la ligne moyenne. Ici H/2 correspond donc à la surface supérieure de la lamelle (-H/2 correspondrait à la surface inférieure).



D'autre part, il faut noter qu'en mesurant la déformation à la surface de la lamelle, il nous est impossible de savoir si nous mesurons la flexion ou la traction/compression. Afin de pallier à ce problème, nous allons positionner une jauge sur les deux surfaces de la lamelle. En faisant la soustraction entre les deux mesures, nous ne garderons plus que la flexion. Nous multiplions aussi la sensibilité par deux avec cette méthode.



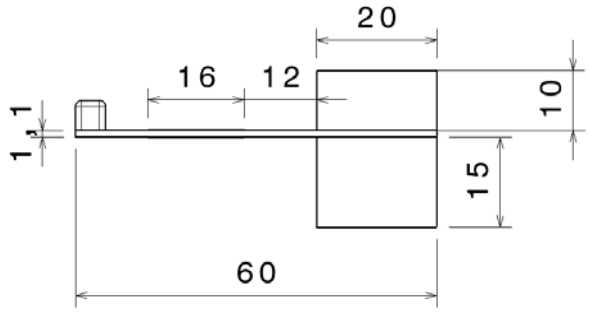
[1] RdM : Résistance des Matériaux

2.1.3 Dimensionnement

Nous allons donc choisir ces différents paramètres de façon à optimiser le capteur en respectant différents critères :

- Rester dans le domaine linéaire
- Minimiser le gain électronique
- Faciliter le montage
- Minimiser le coût

On choisit donc une lamelle en aluminium avec la géométrie suivante :



Longueur	L = 42 mm
Epaisseur	H = 1.1 mm
Largeur	B = 12 mm
Module de Young	E = 69 000 MPa
Masse	M = 20 g

2.1.4 Sensibilité du capteur

Le dimensionnement présenté ci-dessous est la somme de beaucoup de compromis. En effet, nous n'avons que peu insisté sur le fait que tous ces calculs ne sont valables que dans le domaine élastique de déformation : lorsqu'une pièces subit une déformation elle revient exactement et sans déformations à sa forme initiale. En d'autres termes, l'accélération de la fusée (estimée avec Trajec dans le domaine -5G +10G) ne doit pas faire « plier » notre lamelle ! Il faut donc que celle-ci soit résistante, mais pas trop sinon les déformations seront trop minimes et nous ne pourrons plus les mesurer avec nos instruments : les jauges de contraintes. Par manque de temps (lors de la rédaction de ce rapport) nous ne poserons pas ici formellement la démarche qui nous a conduit à ce compromis mais je vais essayer d'en présenter les principaux éléments :

Les jauges de contraintes sont parfaitement caractérisées par un certificat qui est vendu avec lors de l'achat des jauges. Ces éléments sont propres à chaque jauge mais l'ordre de grandeur est fonction du type de jauges : le département GMD nous fournissait des jauges à aluminium, ce qui a directement fixé le paramètre du matériau à utiliser (d'où le module de Young de la figure ci-dessus)

La deuxième particularité était de pouvoir être « facilement » collage par des humains. En effet, le choix de jauges de contrainte doit également se faire en fonction du protocole à mettre en œuvre pour réaliser un collage de qualité. Nous avions un type de jauges facile à coller mais la procédure déroulait tout de même 10 étapes et faisait intervenir au moins 4 produits différents ! Il est très important de suivre scrupuleusement le protocole de collage d'une jauge sinon le risque est de voir ses jauges traduire un peut tout et n'importe quoi mais sûrement pas les déformations de surface de la pièce à étudier (ici notre lamelle).

Une fois le choix de la jauge fixé nous disposons grâce à leur certificat de fabrication d'indications importantes comme par exemple le coefficient d'élongation, la

variation de résistance la jauge en fonction de l'allongement … Il a donc fallu définir la lamelle en fonction de tous ces critères :

- Les jauges doivent travailler uniquement dans le domaine d'élongation et donc de contraintes pour lequel elles ont été fabriquées.
- La Lamelle doit être en aluminium car la compensation thermique des jauges qui nous ont été fournies est valable pour ce matériau. Il est toujours mieux d'utiliser l'aluminium que de la fonte dans une fusée mais d'un autre coté pour ce capteur on a pas hésité à mettre 20g de plomb au bout :o)
- Les déformations que subira la lamelle doivent rester dans le domaine élastique
- La dynamique de mesure des jauges doit être maximale : le maximum de déformation (10G) doit correspondre au maximum d'élongation (donc de variation de résistance) des jauges afin d'avoir le meilleur rapport signal sur bruit en sortie de capteur.

Encore une fois le célèbre principe du compromis est donc à appliquer au dimensionnement de ce capteur. Ce paraphe aurait mérité d'être illustré par les formules et résultats intermédiaires qui nous ont amené à faire ces choix mais les calculs ont malheureusement été faits il y a maintenant 1,5 an et le temps presse pour rendre le compte rendu d'expérience. Somme toute peut-être aurai-je le courage de reprendre cette partie de compte rendu, donc tenez-vous au courant auprès du Cles-Facil pour savoir si une version postérieure de cette documentation existe.

Notes :

- si beaucoup de monde la réclame il est sûr que je ne pourrai laisser autant de questions en suspens trop longtemps et que ça me donnera le coup de pied au postérieur nécessaire à terminer le travail.
- Toute la théorie et les formules (qui devraient figurer ici) sont extraites des polycopiés INSA : TP premier cycle et TP sur les systèmes de mesure et les déformations de GMD.

2.1.5 réalisation mécanique du capteur

Ci-dessous une photo du capteur déjà instrumenté. En raison de sa forme si particulière cette pièce à très vite pris le surnom de « plongeoir ». L'extrémité droite du plongeoir est normalement pourvue de sa masselotte de » 20g. Contrairement au plan ci-dessus la masselotte de 20g est plus volumineuse malgré le fait que nous l'ayons faite en plomb. Le choix de ce matériau à été fait sur des critères de densité et de malléabilité pour sa mise en forme. Afin de simplifier les calculs nous avons souvent été amenés à assimiler comme ponctuelle la masselotte ce qui nous a conduit à s'assurer que le centre de gravité de notre masselotte (petit cube d'environ 10x10mm) était bien à la distance prévue du pied du plongeoir. La masselotte avait globalement une forme comme celle décrite en Figure 2.

Lors de la réalisation une attention particulière à été portée au niveau de la jointure lamelle/pied du plongeoir. En effet, cette jointure doit se faire avec des bords parfaitement « vifs » afin que tous les efforts subis par la lamelle se traduisent bien par une petite déformation de cette dernière. (et non par un quelconque jeu d'assemblage entre le pied et la lamelle.

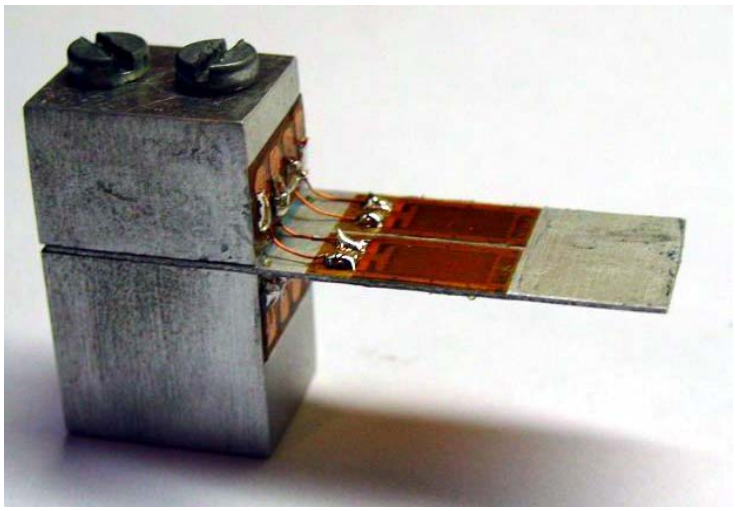


Figure 1 : Plongeoir équipé pour la mesure

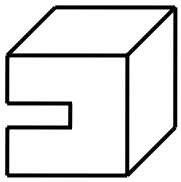


Figure 2 : Masselotte de 20g donc le centre de gravité était au bout du plongeoir

3. Electronique

3.1 Le capteur

Le schéma ci-dessous présente la transition entre mécanique et électronique en présentant la chaîne des phénomènes qui varient ou nous allons intervenir pour réaliser notre mesure.

On peut voir qu'il existe de nombreuses étapes intermédiaires entre les effets du phénomène physique et la sortie d'une grandeur mesurable. Toute ces étapes intermédiaires introduisent des erreurs et incertitudes de mesure dont il faut avoir conscience et qu'il faut maîtriser. Pour la plupart d'entre elles nous en avons déjà parlé plus haut : par exemple l'interface déformations lamelles / déformations jauges est directement dépendante de ce fameux collage des jauges !

Tous ces « petits défauts » sont inévitables et doivent pouvoir être pris en compte par l'étalonnage. Et si ce n'est pas le cas il faut véritablement limiter ce défaut de façon à le rendre négligeable devant les autres. Prenons deux exemples :

Le centre de gravité de la masselotte n'est pas exactement au bout de la lamelle (interface variation d'accélération / déformation lamelle du schéma ci-dessous) : ce n'est pas très grave car les valeurs de l'étalonnage corrigeront cette imperfection.

La réponse des jauges varies en fonction de la température ! (interface Déformation jauges / variation de résistance) Ce problème est très grave car nous n'étalonnerons pas notre capteur dans les mêmes conditions de température que celles

rencontrées durant le vol. Il faut donc choisir **dès la conception** des jauges auto compensées en température et que la plage de cette auto compensation soit compatible avec les températures rencontrées lors du vol.

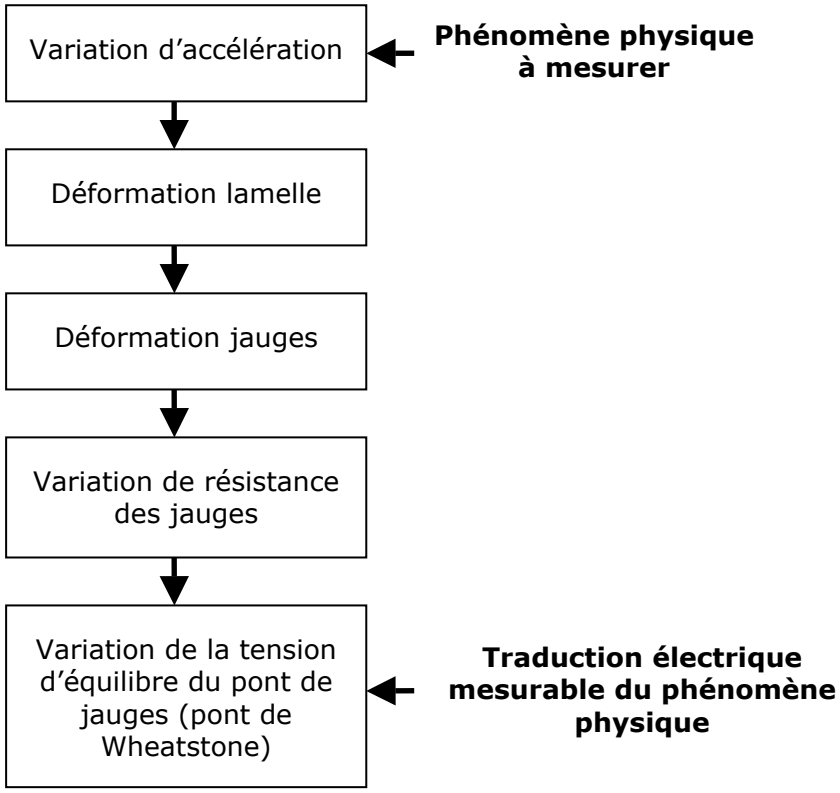


Figure 3 : traduction du phénomène physique en grandeur mesurable

Dans le paragraphe « 2.1.4 Sensibilité du capteur » nous avons signalé qu'il était important d'avoir un maximum de dynamique au niveau des jauges de contraintes. C'est la raison principale qui nous ont fait choisir de réaliser un pont de Wheatstone complet avec 4 jauges au lieu de mettre (3 résistances pures et une jauges ou 2 résistances pures et 2 jauges). En effet, la sensibilité de notre capteur est ainsi multiplié par 4. Ainsi pour une tension d'alimentation du pont de jauge réalisée en symétrique (-6V,+5V) nous nous observions des variations en sortie de notre plongeoir de l'ordre de ±5mV (Accélération de décélération). Encore une fois un bon gros rappel (démontrer les formules de sensibilités…) sur le pont de Wheatstone serait super utile aux gens qui prendront le temps de lire cette documentation mais encore une fois le temps me manque !

L'ordre de grandeur de ces valeurs avait bien sûr été calculé dès la conception du capteur afin de pouvoir ensuite dimensionner le chaîne de traitement du signal : facteur d'amplification …

La photo ci-dessous figure une face de la lamelle supportant deux jauges de contraintes (les deux autres étant sur l'autre face).

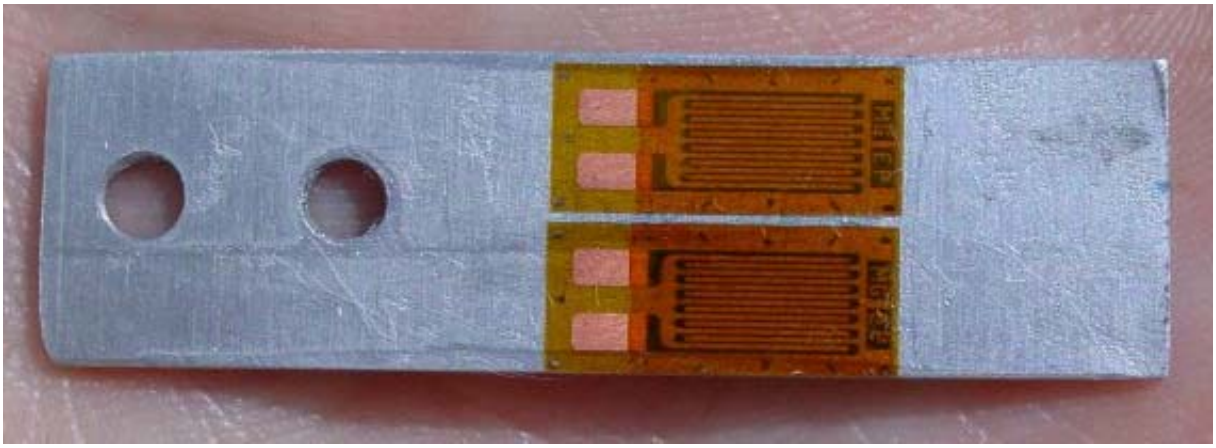


Figure 4 : Jauges collées sur une face

3.2 La chaîne d'acquisition

Tout comme le capteur Pitot la chaîne de base est toujours constituée de 4 maillons principaux : capteur (longuement décrit dans les paragraphes précédents) → Amplificateur → Filtre → Convertisseur analogique/numérique. On n'a rien inventé de ce point de vue là, en effet aXelle se résume souvent à l'application de principes de bases avec des composants récents en essayant de faire attention à tous les maillons.

Nos contraintes principales étaient comme pour le tube de Pitot l'intégration de l'électronique. En effet, nous disposions encore pour ce capteur d'une petite boite blindée électromagnétiquement (voir explication sur ce choix technique dans la partie tube Pitot Chapitre 01). Nous avons choisi de mettre tout le capteur dans cette boite : mécanique et électronique. La mécanique est fortement liée (par de la colle d'ailleurs) à l'électronique puisqu'elle porte les jauges de contraintes. Déporter la mécanique serait revenu à faire apparaître des fils de liaison pour les jauges. Le signal circulant dans ces fils est précisément le signal porteur de toute l'information et c'est également dans ces fils qu'il est le plus vulnérable, au cause de sa très faible amplitude (environ 5mV). Il nous fallait donc protéger ces fils, nous souhaitions éviter la mauvaise expérience sur ELA (projet 2002). Cela revenait à complexifier encore un peu plus le capteur. Ce capteur est donc le plus complexe (en nombre de composants), il doit s'intégrer dans le même espace que les autres mais doit en plus cohabiter avec la partie mécanique : le plongeoir, attention les yeux en ouvrant la boîte les composants sont tellement tassés qu'ils pourraient sauter tout seuls.

Le schéma suivant présente une vue fonctionnelle complète de la carte jauges. Encore une fois ce n'est pas très compliqué quand on repère les fonctions de base de la chaîne d'acquisition et que l'on cherche à quelle partie **fonctionnelle** appartient ce composant. : « tiens un multipattes, oula ça doit être compliqué je laisse tomber ! » non pas du tout, il faut plutôt chercher d'où commencer l'investigation et chercher qui sert à quoi ! Nus avons conçu la carte par étapes : nous n'avons pas jeté tous les composants au hasard sur la plaque d'époxy. Nous avons eu besoin d'un amplificateur qui avait telles caractéristiques : nous l'avons trouvé, ensuite pour le faire marcher il nous fallait tel et tel type d'alimentation avec telles caractéristiques… L'électronique n'est pas non plus un jeu de lego, mais quasiment à notre niveau d'utilisateur. Le savoir faire du club est de bien choisir les briques de ce lego, **celles qui répondent bien à nos besoins.** Faire le bon choix des briques c'est comprendre (au minimum) comment elles fonctionnent, **connaître ses besoins,** adapter les composants à nos besoins le cas échéants.

Les rectangles autour des fonctions citées figurent les limites du découpage du projet dans le logiciel de CAO : Protel. Les paragraphes suivants présenteront les schémas électriques de chacun de ces rectangles. Afin de faciliter la localisation et la

lecture des schémas la figure ci-dessous reprend les noms exacts des interfaces entre les fichiers Protel. Ce schéma est donc la carte à grande échelle de la carte jauge :

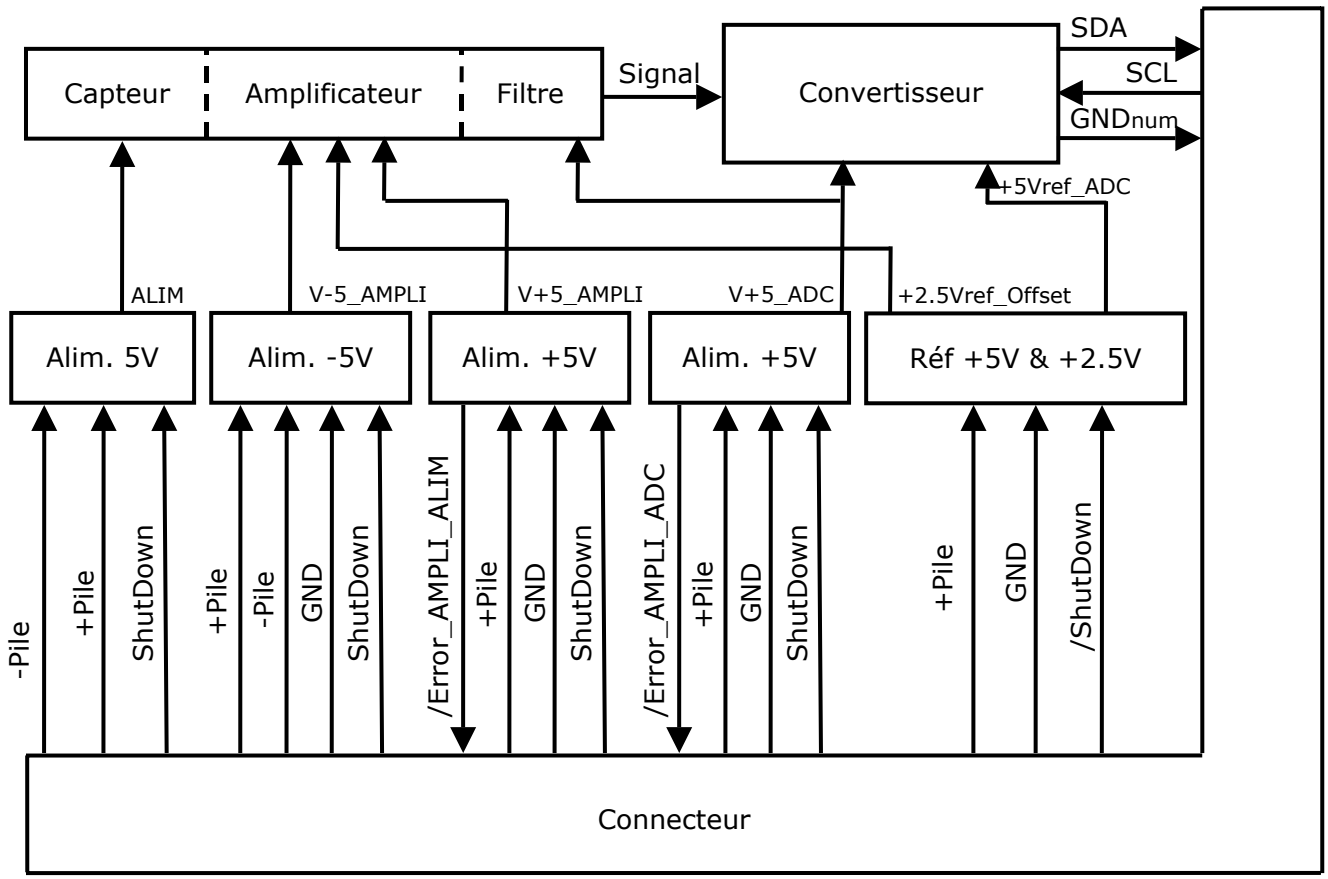


Figure 5 : Synoptique de la carte jauges

3.3 Schémas électroniques

Les cartes détaillés à petite échelle de la carte jauge … bon voyage avec le CLES-FACIL prochain arrêt Pluton la minuscule.

3.1.1 Convertisseur Analogique-Numérique.

La puce retenue pour assurer cette fonction est comme pour le tube de Pitot (Chapitre 01) un convertisseur à 1 voie : **ADS7823** de chez Texas Instruments. Je ne vais donc pas m'étendre sur le sujet. Si je remets le schéma à la suite de ce petit paragraphe c'est pour le nom des interfaces car sinon c'est exactement la même chose que pour le Pitot. De plus, vous pouvez vous reporter à la partie de ce rapport dédiée à la conversion analogique/numérique. (Chapitre 06).

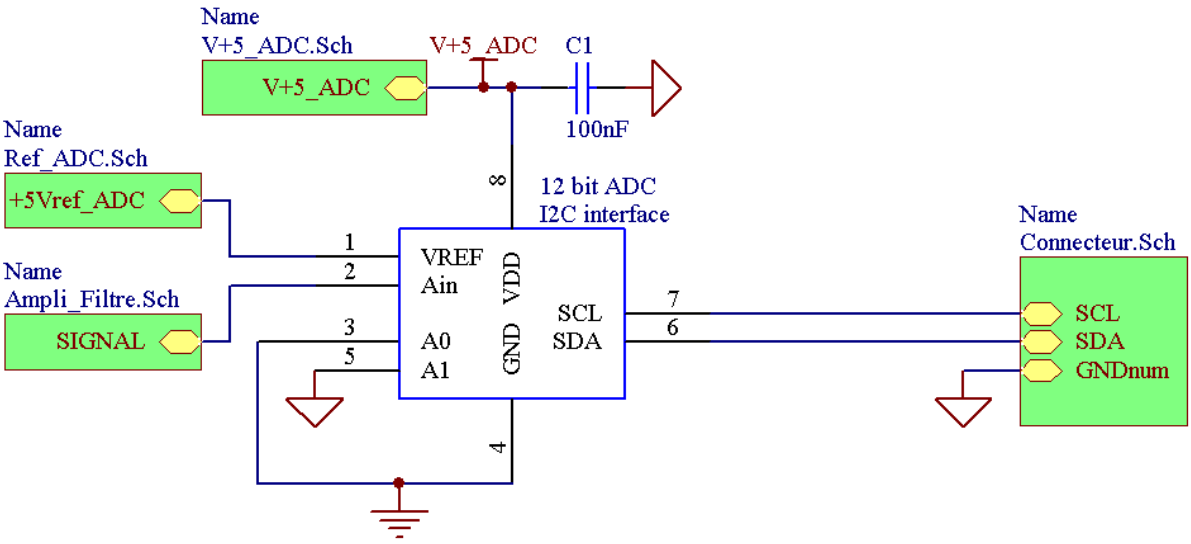


Figure 6 : Schéma du convertisseur analogique/numérique

3.1.2 Amplificateur et filtre

Je ne vais pas vous faire le coup à chaque fois de vous renvoyer à la documentation Pitot car j'aurais ici à moitié tort. En effet, le filtre est le même que celui utilisé pour le capteur précédent (on est méchant avec Francis de toujours réutiliser ce composant) par contre l'amplificateur est, lui, totalement différent.

Il s'agit d'un amplificateur d'instrumentation, c'est-à-dire un super amplificateur fait à partir de trois amplificateurs de base. Donc c'est un peu méga balaise et pour les boss exclusivement. Mais non, pas du tout. Car Mr BURR-BROWN (encore lui comme pour le Pitot) nous a mis tout ça dans un petit 8 pattes facile à utiliser :

- 1 patte de masse
- 2 d'alimentation (positif et négatif)
- 2 pour le signal (et oui celui-ci n'est pas forcement référencé à la masse)
- 2 pattes pour souder la résistance correspondante au facteur d'amplification désiré. La relation simple entre les deux grandeurs et données dans la datasheet.
- 1 patte pour réaliser un offset du signal. C'est encore une économie de sueur, il aurait fallu le faire nous même et là Mr BURR-BROWN y a déjà pensé pour nous. Comme quoi on ne fait peut-être pas des choses si exceptionnelles au CLES-FACIL, ou alors tout le monde fait des composants pour fusées lyonnaises …

Et voilà, alors quel est cet animal incroyable à 8 pattes : l'INA114. Il ne faut pas le chercher chez BURR-BROWN mais plutôt chez Texas Instrument car ces deux firmes ne font plus qu'une maintenant.

Le choix d'un amplificateur symétrique est justifié par le fait que notre plongeoir générera des deltas de tension positifs et négatifs correspondant aux phases d'accélération ou de décélération. (attention les positifs ne sont pas forcement l'accélération et inversement, il faut pour vérifier ce point vous reporter à la section sur l'étalonnage.)

L'INA114 permet également d'injecter un offset, ce qui nous arrange beaucoup. En effet, les variations de tension produites par les jauges seront centrées autour de 0. On peut donner la plage [-5mV ;+5mV] pour fixer les idées. En amplifiant cette plage d'un facteur 500 on disposera d'une plage de travail de [-2.5V ;+2.5V] ce qui pose un

problème pour l'étage fonctionnel suivant : le convertisseur analogique numérique. En effet, ce dernier ne peut convertir que des tensions comprises dans la plage [0 ;5V]. Mais vous l'avez déjà compris, grâce à un offset de 2.5V en sortie d'amplificateur nous rendons la plage de valeur compatible avec la plage de travail du convertisseur. Le zéro est donc au milieu de la plage de conversion (2048 car notre convertisseur est 12bits) Mais à quoi correspond ce zéro ? Au moment où les contraintes sur les jauges sont identiques des deux cotés c'est-à-dire 0G. Et donc, lorsque notre capteur sera posé sur la table il sera soumis à 1G (gravité terrestre) et il ne faudra donc pas nous attendre à lire 2048 dans ce cas. Nous verrons plus loin que ce petit raisonnement est énormément simplifié et que beaucoup de paramètres viennent le complexifier, somme toute il est important de le garder à l'esprit.

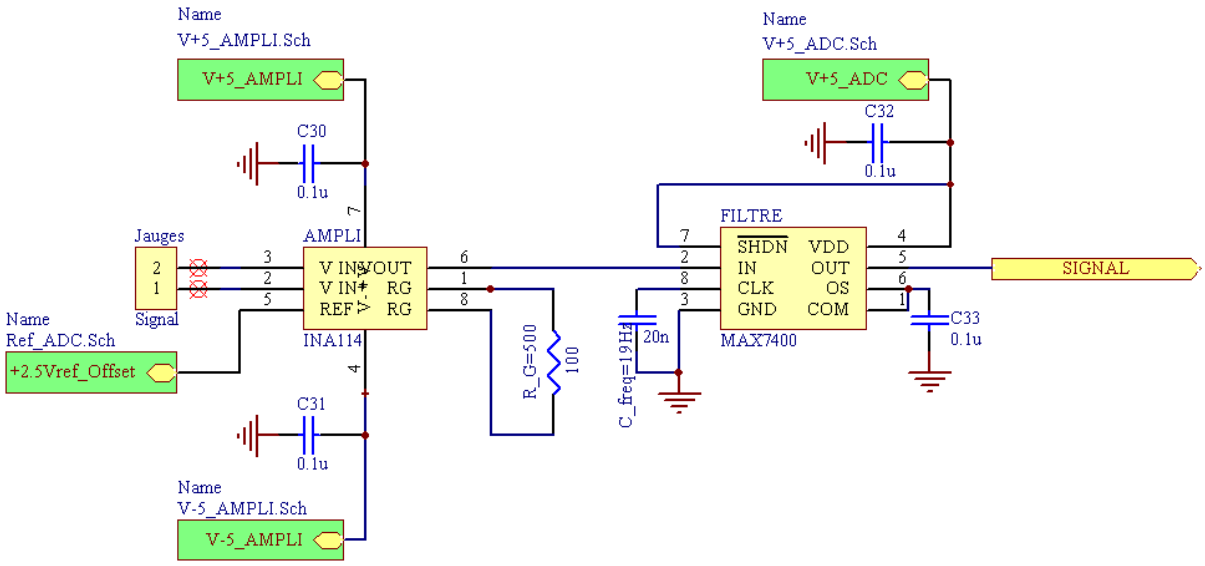


Figure 7 : Schéma amplificateur+filtre

3.1.3 Connecteur

Le principe est toujours le même : garder la boite blindée le plus étanche possible aux perturbations. Ainsi le capteur ne peut pas être perturbé depuis l'extérieur mais il ne peut pas non plus aller mettre le souk ailleurs. Limiter les pollutions électromagnétiques dès la source est souvent négligé au profit du « je me protège des autres » mais c'est pourtant aussi essentiel.

Nous utilisons donc pour ce capteur les 9 connexions offertes par le connecteur DB9 que nous avons choisi. L'emplacement des signaux est dans la mesure du possible compatible entre chaque boite (alimentation, bus I²C, commandes ON/OFF, retour d'info).

Le schéma suivant fait également apparaître un transistor nécessaire à l'inversion des commandes logique d'allumage des régulateurs de tension. Cette carte contient deux régulateurs commandables à distance et ayant des capacités de retour d'information sur l'état d'usure des piles : ce qui explique qu'il faille deux interrupteurs pour allumer complètement la carte et qu'elle ait deux retours distincts sur l'état de ses alimentations.

Même si cette politique est louable car permettant une localisation rapide des problèmes elle contribue à l'encombrement en fils, interrupteurs, LED. Il aurait à mon avis fallu factoriser l'allumage au niveau de la carte ainsi que le retour d'information. Depuis l'extérieur : un bouton et un voyant. En cas de problème, on ouvre la boite et la on regarde laquelle des deux LED rouge CMS (par exemple) est allumée pour trouver le régulateur responsable de l'échec de la mise sous tension. Dans tout les cas il faut toujours ouvrir la boite alors autant ne pas faire sortir d'information inutile.

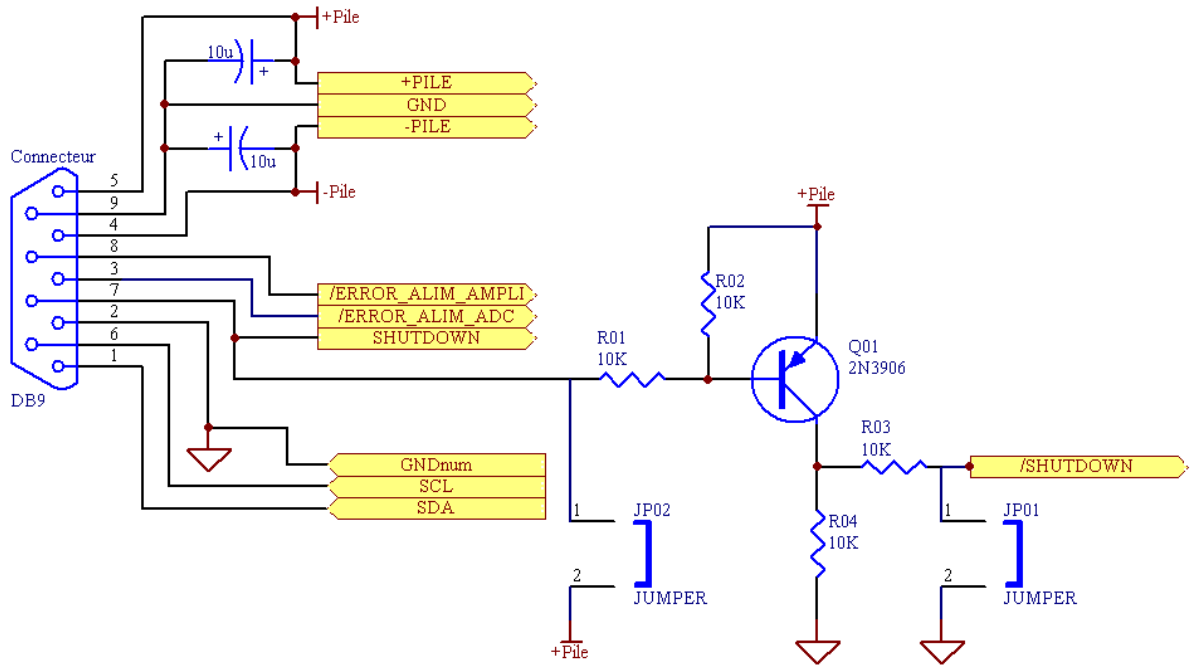


Figure 8 : connecteur d'interface de la boite blindée

Nota : La valeur de résistance R01 a été augmentée par rapport à celle indiquée sur ce schéma, de façon à garantir une bonne saturation du transistor. Sa valeur se situe en fait vers 22 Kohms.

3.1.4 Alimentation amplificateur : +5V

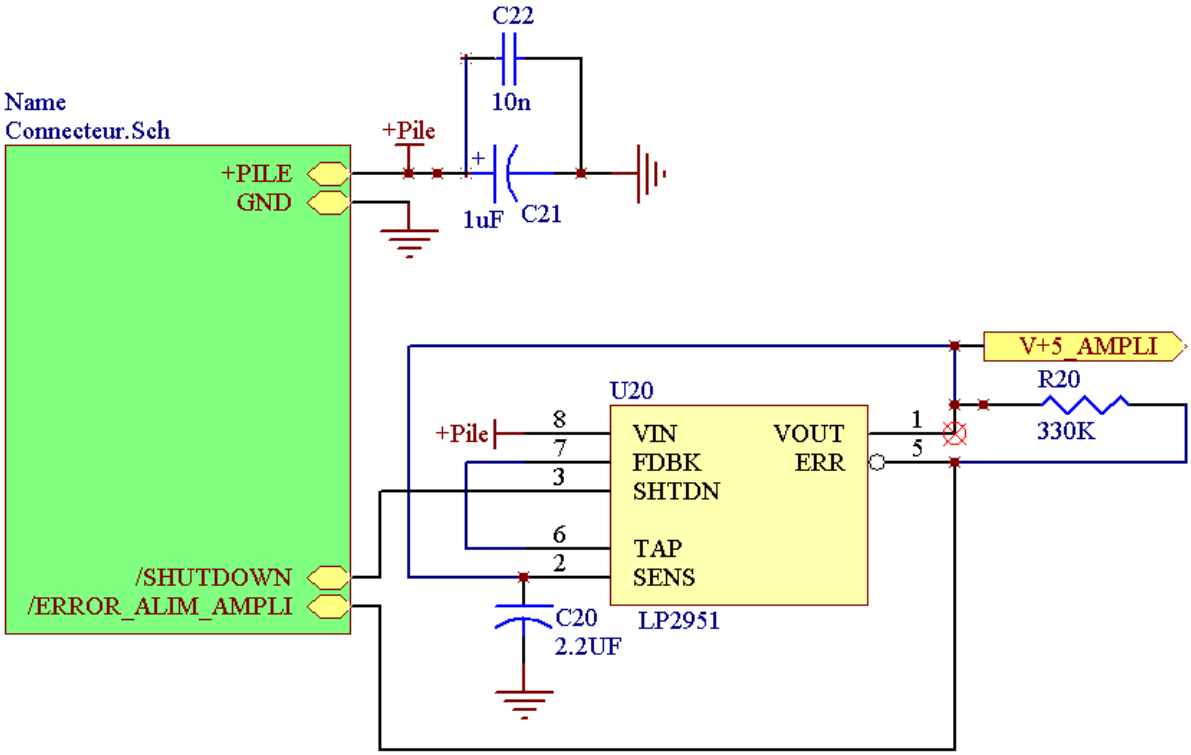


Figure 9 : régulateur de tension +5V

Voici le schéma du régulateur utilisé pour générer le +5V nécessaire au fonctionnement de l'amplificateur bipolaire utilisé dans ce circuit. Le paragraphe d'après présente la deuxième partie de l'alimentation de l'amplificateur : le -5V. Le composant principal est un LDO : régulateur à faible tension de chute ayant pour référence LP2951. Pour tous les détails sur cette alimentation (capacité, précision …) je vous invite a vous reporter à la partie de ce document traitant précisément des alimentations.

3.1.5 Alimentation amplificateur : -5V

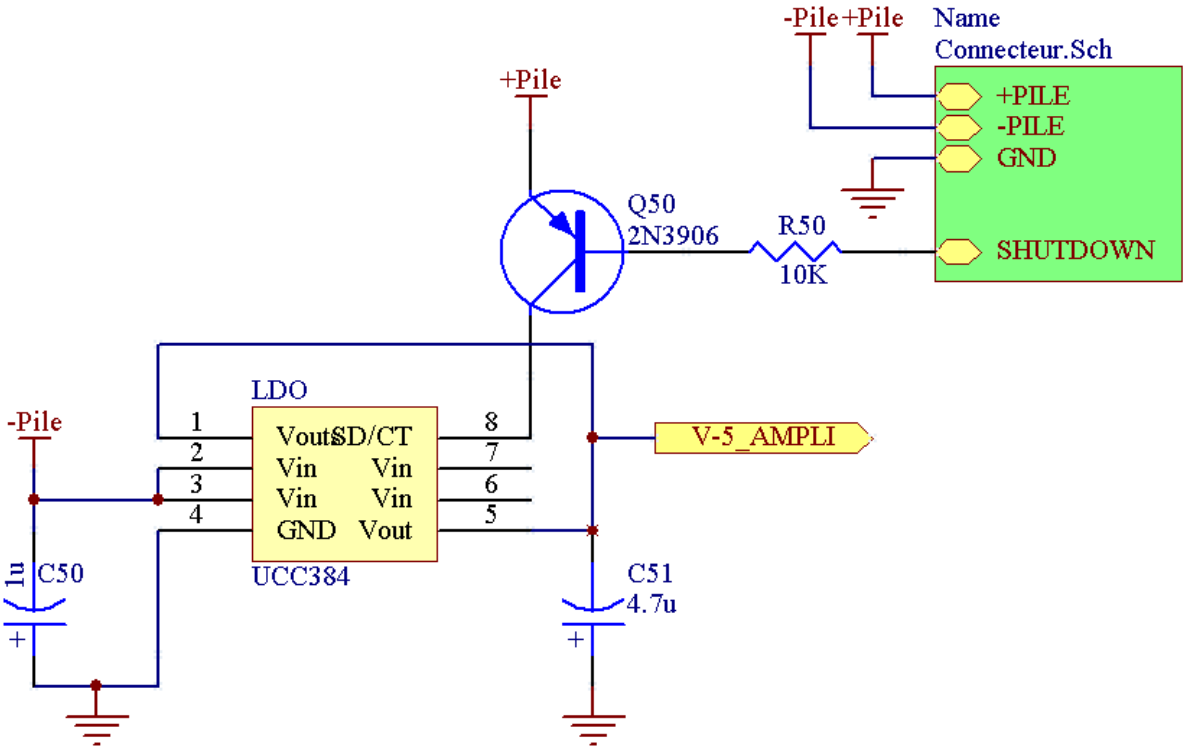


Figure 10 : régulateur de tension -5V

Il s'agit cette fois d'une alimentation « négative » (toujours un peu plus délicate) pour laquelle nous partons des pôles [-6V;+6V] des deux piles dont nous disposons pour ce schéma. Comme dans le paragraphe précédent je vous renvoie au paragraphe sur les alimentations pour plus de détails.

3.1.6 Régulateur +5V pour l'ADC

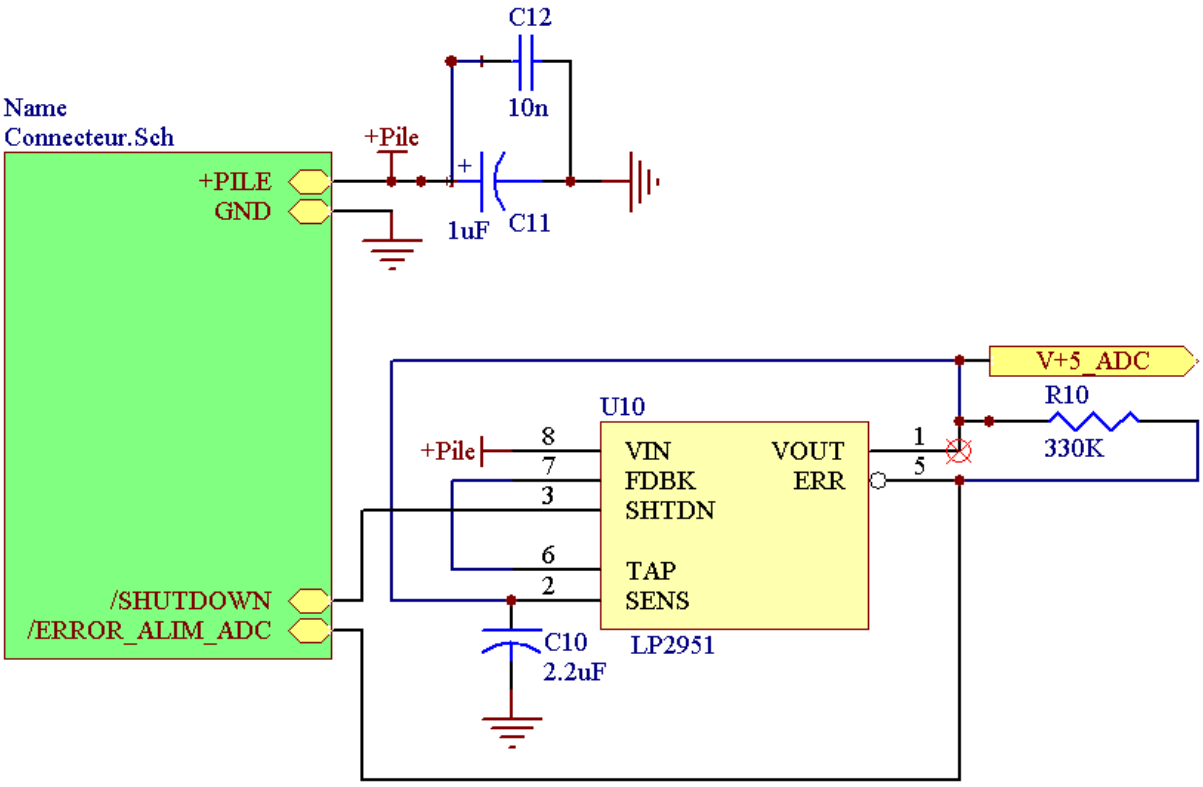


Figure 11 : régulateur de tension +5V

Alors là les loulous, on peut dire qu'on fait du remplissage car si vous avez suivi les épisodes précédents vous vous rendez compte qu'il s'agit exactement du même schéma que pour le régulateur +5V d'alimentation positive de l'amplificateur d'instrumentation. On aime les LP2951 (en plus il sont faciles à router) ! Comme d'habitude pour tous les détails, rendez-vous à la section alimentation de ce pavé.

3.1.7 1 composant : 2 références !

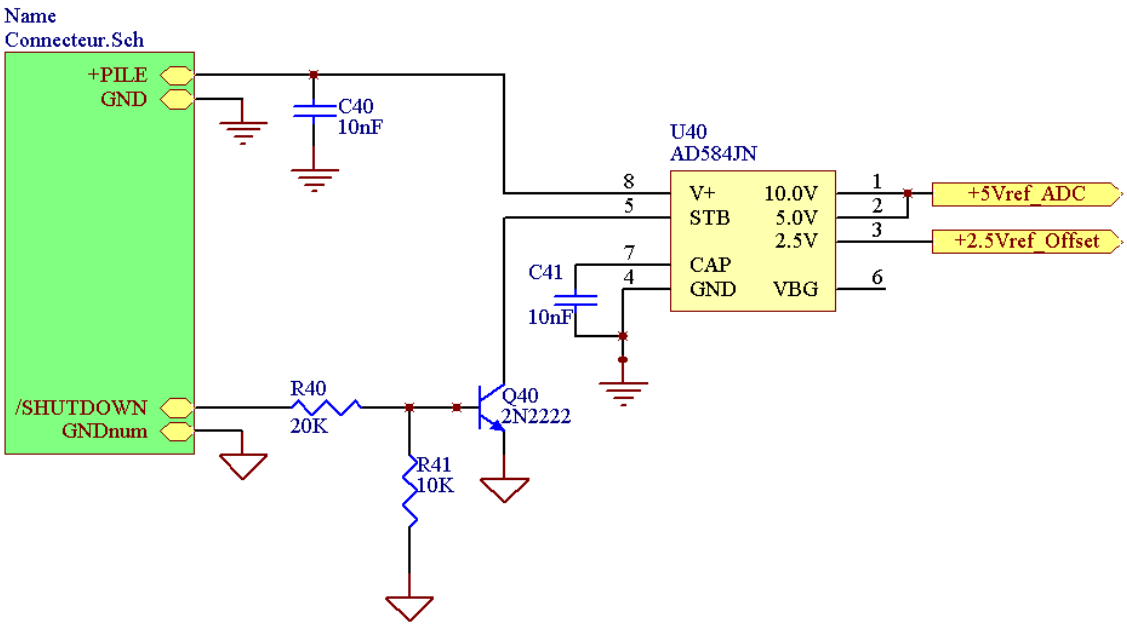


Figure 12 : référence de tension +2.5V et +5V

Nous avons déjà parlé de la différence entre régulateur et référence de tension dans la documentation et dans tous les cas reportez-vous à la documentation sur les alimentations qui comprend également un très bon descriptif de ces références de tension.

Avec le schéma présenté ci-dessus on génère une référence +2.5V qui va nous permettre de faire l'offset du signal au niveau de l'amplificateur vu ci-dessus. On génère également une référence de +5V qui servira de point de « repère » à l'ADC : c'est lorsque l'ADC verra le même +5V que cette référence qu'il nous codera un 4096 ! Le composant utilisé pour faire tout cela est un ancêtre mais toujours bien utile : l'AD584. Voir les précautions d'emploi de ce composant dans le paragraphe de ce document dédié aux alimentations car nous avons rencontré quelques surprises avec.

3.1.7.1 Référence pour alimenter le pont de jauges !

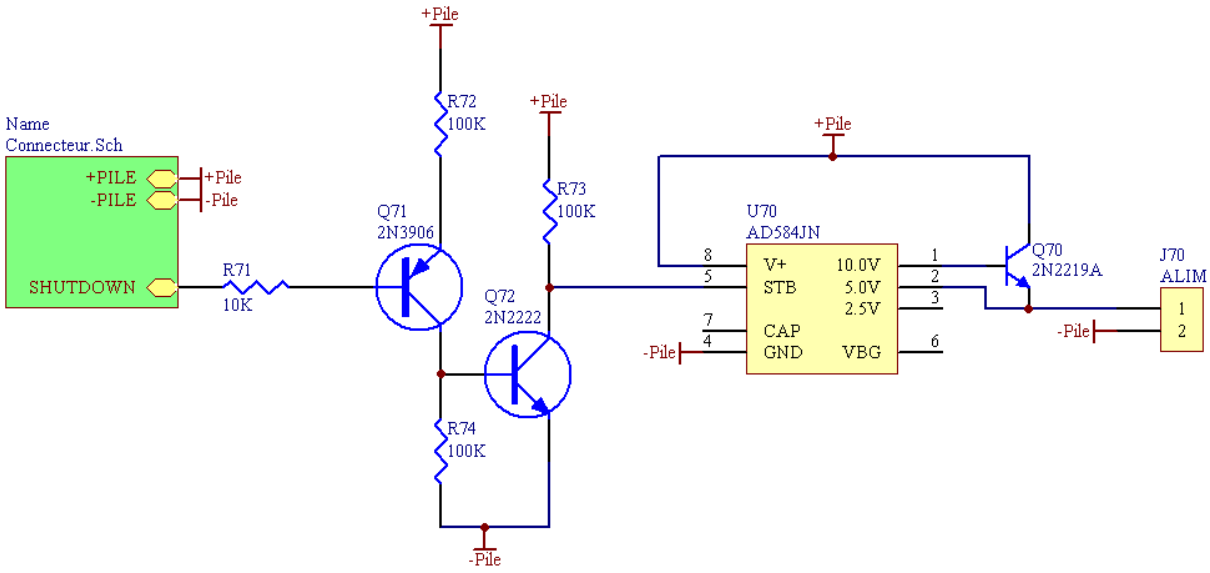


Figure 13 : référence de tension 5V

Ce montage permet de maintenir une différence de potentiel de 5V aux bornes du pont de jauges. Pour plus de précisions sur ce montage un peu exotique – mais qui marche - se reporter au chapitre 10 concernant les alimentations.

3.2 Bilan de consommation

Le tableau suivant présente le bilan de consommation de la carte Jauges. Comme pour le Pitot la réalisation de ce bilan permet d'estimer l'autonomie de cette expérience, qui elle-même entrera en compte dans le calcul de l'autonomie de la fusée.

	Consommation moyenne	Conso. Max théorique
LDO +5V (Amplificateur)	1mA	
LDO -5V (Amplificateur)	1mA	
LDO +5V (ADC)	1mA	
Référence AD584 +2.5V & +5V (Offset et ADC)	N.C.	
Référence + (Pont de Wheatstone)	44 mA	

Figure 14 : bilan de consommation interne de la carte, d'après les données constructeurs.

Pour déterminer l'autonomie de l'expérience nous avons besoin de la capacité initiale des piles (le vol s'effectuant avec des piles neuves) :

Figure 15 : câblage des piles de l'expérience accéléro jauges

	Pile n°1 (-6V)	Pile n°2 (+6V)
Capacité initiale des piles	1.3 A.h	1.3 A.h
Consommation carte Jauges	44.6 mA	47.5 mA
Autonomie de ces expériences	27 heures environ	

Figure 16 : calcul de l'autonomie de l'expérience Pitot

Comme on peut le constater, l'autonomie de notre expérience est bien supérieure à la limite des 45minutes minimum imposée par le cahier des charges. Il faut pourtant garder à l'esprit que ce sera la partie électronique ayant la plus faible autonomie qui définira la véritable autonomie de la fusée (principe de la grosse dame de TF1).

3.3 Etalonnage

Le principe d'étalonnage est simple. Lorsque la fusée est verticale, elle est uniquement soumise à l'accélération terrestre : $9.81 m.s^{-2}$. Cette accélération génère une force au niveau de la lamelle au travers de son action sur la masselotte de 20g placée en bout de lamelle. Donc 20g correspondent à 1G, si on double la masse de la masselotte au bout de notre lamelle nous simulerons une accélération double et ainsi de suite jusqu'à 10G. Pour l'étalonnage en décélération il suffit de réaliser la même opération mais avec la boite dans l'autre sens. Un schéma rendra sûrement les choses plus claires mais ce n'est vraiment pas difficile.

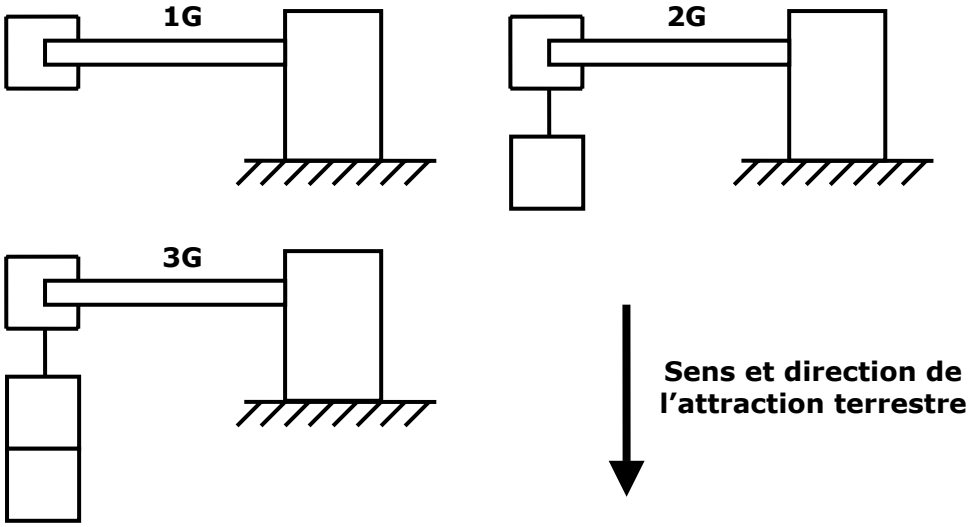


Figure 17 : Etalonnage du système pour les accélérations « positives »

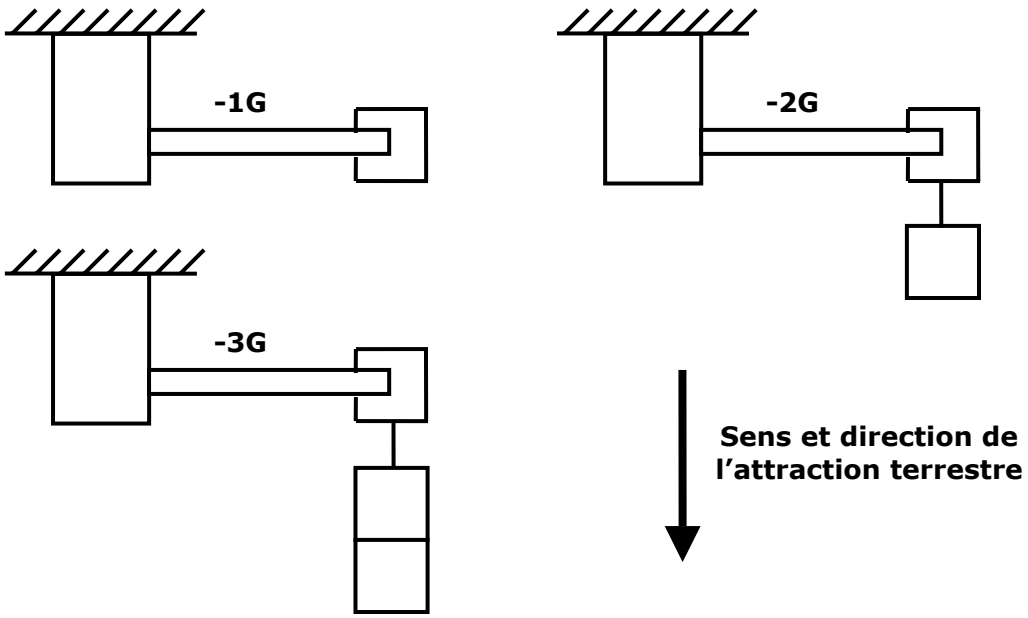


Figure 18 : Etalonnage du système pour les accélérations « négatives »

Bien entendu pour obtenir le point 0G il suffisait de mettre le plongeoir sur sa trace afin que l'attraction terrestre ne vienne pas déformer la lamelle.

Le tableau suivant résume les différents points d'étalonnage que nous avons réalisés. Il s'agit pour chaque point de moyennes sur plusieurs centaines de valeurs au travers de notre espion I²C. En effet, une fois fermée notre boite ne laissait plus transpirer de valeurs analogiques et communiquait déjà en numérique via le bus I²C. Il est très important que l'étalonnage intègre toute la chaîne de mesure. Si on étalonne avec les valeurs analogiques le biais du convertisseur n'est pas pris en compte. Malgré nos efforts, il est regrettable de ne pas avoir intégré le processeur et le stockage mémoire dans cette phase d'étalonnage. Nous le verrons plus tard, lorsque qu'un doute survient sur une mesure/ un capteur on doit être parfaitement certain de toute sa chaîne afin de savoir si on doit chercher une explication physique au phénomène bizarre que l'on observe (prix Nobel nous voila) ou un simple défaut d'électronique (Prix Lesel nous voila :op)

Etalonnage jauges	
Accélération (m/s-2)	Valeur mesurée
10	2179,01
20	2311,39
25	2352,15
30	2448,02
35	2464,22
40	2561,47
45	2573,52
50	2650,25
55	2766,36
60	2740,97
65	2792,88
70	2761,15
75	2812,11
80	2829,97
85	2841,59

Figure 19 : Valeurs de l'étalonnage jauges

On remarque que sur la Figure 19, seule des valeurs positives d'accélération on été testés. Ceci est du à un problème de temps lors de la campagne, comme d'habitude : nous extrapolerons donc nos résultats pour les résultats. D'un autre coté une prouesse de cette année à été de faire plus de 2 points d'étalonnage pour trouver une droite (ceci nous permet de ne pas avoir une extrapolation trop stupide pour les valeurs de décélération) Comme on peut le remarquer la gravité terrestre est prise égale à $10 m.s^{-2}$ pour simplifier les calculs. (Bon ok, on aurait pu se casser le cul ! Vous corrigerez les valeurs de vous-même -1.9% ! D'un autre coté nous n'avions pas de référence pour mesurer l'attraction terrestre à Sissonne alors pourquoi pas 10G.

D'un coup de baguette magique vous allez trouver ci-dessous le graphe qui correspond à ces valeurs, ainsi que la régression linéaire qui va nous donner la loi d'étalonnage tant attendue.

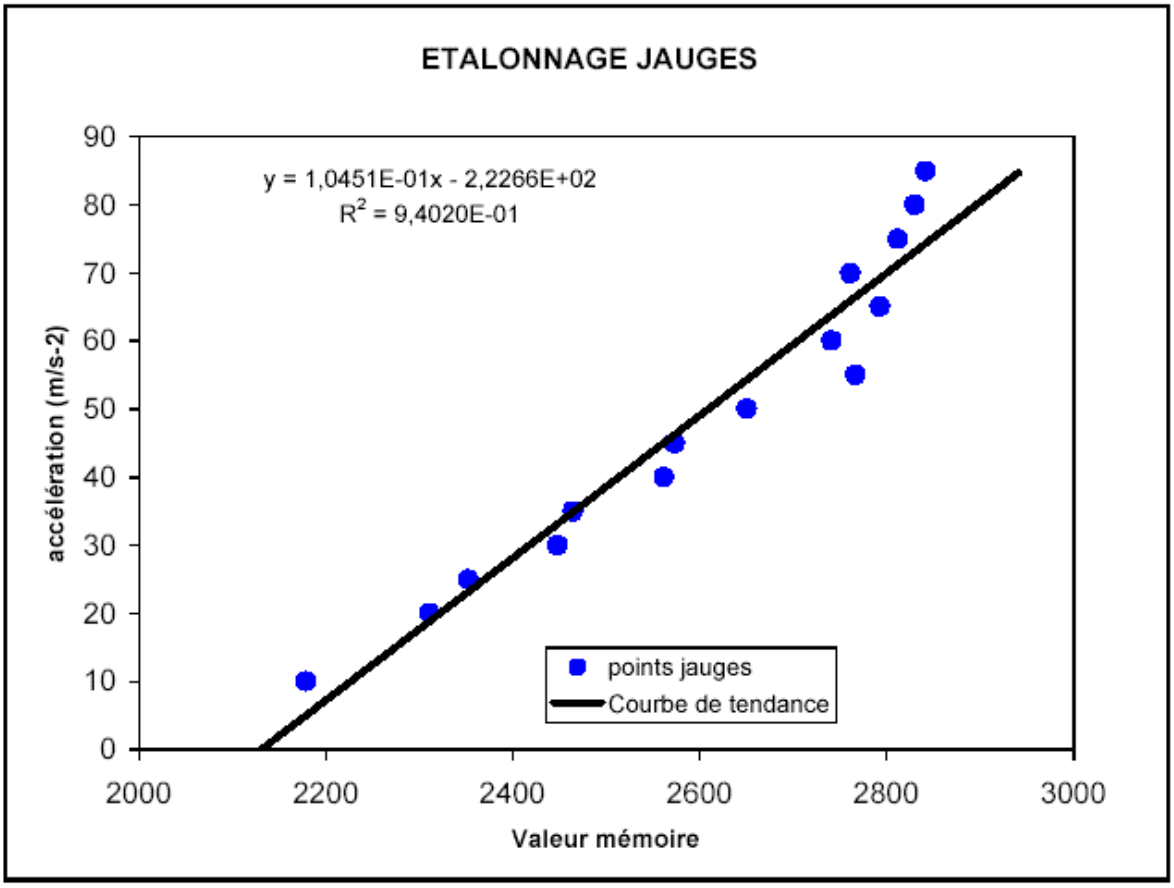


Figure 20 : Courbe de tendance permettant d'obtenir la loi d'étalonnage du capteur

Nous pouvons d'ores et déjà faire plusieurs remarques sur cette courbe :

- La linéarité est assez bonne R^2=0.94. Rappelons que tout le dimensionnement mécanique à été fait pour justement respecter cette règle de linéarité. Nous savions par hypothèse que nous devions faire une régression linéaire et pas une régression par rapport à une courbe sinus mâle.
- Les défauts de linéarité apparaissent surtout à la fin : ce qui est normal car nous approchons de plus en plus des limites élastiques du matériau et des plages de fonctionnement des jauges.
- 7,5G sont codés en 700 valeurs numérique ce qui n'est pas parfait il aurait mieux valu utiliser plus la dynamique du convertisseur à savoir 12bits/2 c'est-à-dire 2048 valeurs. Ceci vient de plusieurs problèmes rencontrés au niveau de l'amplificateur mais n'est pas dramatique.

Au final, on a donc :

Accélération (en m.s^{-2}) *= 0.10451 x* ***Valeur numérique (12 bits)*** *– 222,66*

4. Exploitation des résultats des jauges.

Le vol nominal de aXelle nous a permis de recueillir les valeurs la FSK et de la mémoire (MEM). Des problèmes (Voir Chapitre 08) de fréquence d'acquisition nous ont obligés à sous-échantillonner les valeurs de la mémoire pour correspondre à celles de la FSK. Nous disposons donc de 2 jeux de 246 valeurs chacun. La fréquence d'échantillonnage est pour ces deux jeux de : 20 Hz soit les 12.15 secondes de vol avant

le largage du module. Nous disposons également des valeurs théoriques de Trajec : 290 échantillons sur tout le vol balistique (cas ou le parachute ne s'ouvrirait pas) à une fréquence de 10Hz. Il n'y a donc que les 125 premières données de Trajec qui correspondent à nos mesures.

Note : le pas de calcul de Trajec est configurable, nous aurions donc pu avoir plus de valeurs ou autant que l'expérience.

Les valeurs de la FSK étaient transmises sur 8 bits pour des raisons évidentes de débit. Une fois les valeurs remises à la même échelle (×16) on remarque que 28 valeurs (9.7%) diffèrent de plus de 16 en décimal (plus petite granularité de la FSK). Nous remarquons rapidement qu'il semble y avoir plus d'erreurs au début du vol (2.5 premières secondes) puis une constance dans le taux d'erreur (entre 2.5sec et 8.5sec) et enfin une raréfaction sur la fin. De plus, le signe de l'erreur : ($Valeur_{FSK}$ - $Valeur_{MEM}$) semble également lié à ces périodes. Une explication pourrait être les contraintes subies par l'antenne boudin lors de la phase de poussée du vol.

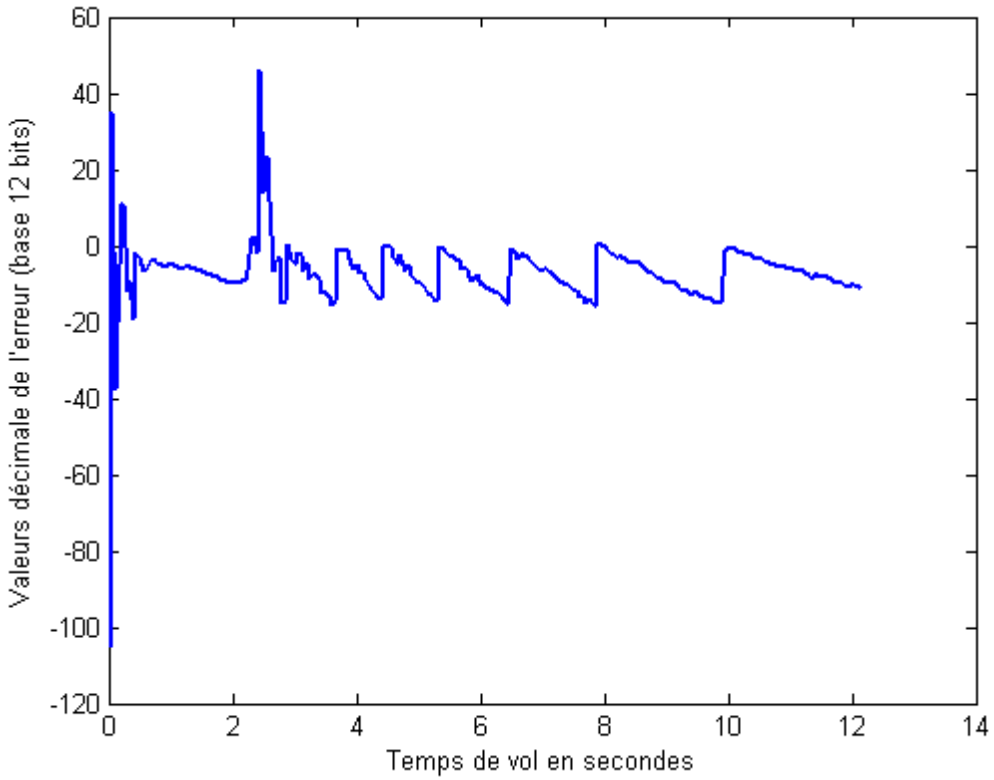


Figure 21 : Les erreurs de transmissions (Valeur FSK – Valeurs MEM) en fonction du temps

Nous avions observé des erreurs du même ordre de grandeur pour les données du tube de Pitot (max 150 alors qu'ici nous n'avons que 100). Nous avions fait plusieurs hypothèse pour expliquer ce phénomène. Il apparaît ici clairement que ce phénomène est périodique. Ceci rejoint l'hypothèse de la rotation de la fusée. En effet, nous ne disposions à bord d'aXelle que d'une seule antenne boudin disposée suivant un rayon de la fusée. Pour parfaitement, confirmer cette hypothèse il nous faudrait regarder une vidéo du vol de aXelle et vérifier que celle-ci à une vitesse de rotation d'environ 1s puis de 2s suivant le moment du vol. Le fait que la vitesse de rotation diminue avec le temps est parfaitement logique puisque la vélocité de la fusée diminue également. Les forces s'appliquant sur le défaut d'équerrage des ailerons et qui produise la rotation sont directement liées à la vitesse de la fusée.

4.1 Résultats

La grandeur physique que traduit notre capteur est directement l'accélération : objet de toutes nos attentions dans ces fusées.

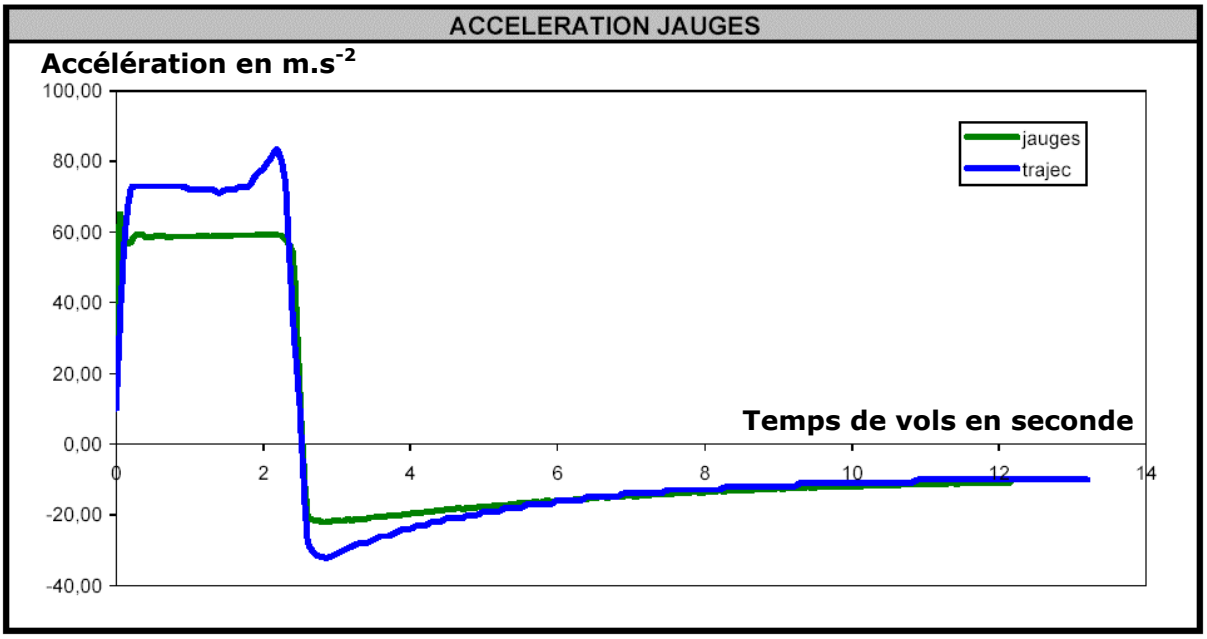


Figure 22 : Evolution de l'accélération en fonction du temps et comparaison théorie/expérience

On remarque que l'allure des courbes est parfaitement similaire somme toute on constate un petit rapport d'échelle. Ce rapport d'échelle peut avoir plusieurs explications possibles :

- Notre capteur est saturé (conforté par le fait qu'on ne voit pas le petit pic des 2sec). Cette saturation peut être de 2 ordres :
  - Mécanique : la masselotte arrive en butés basse : elle touche la plaque de composants électroniques. Ceci est improbable car nous sommes à 60m.s^{-2} et nous avons réussi à étalonner jusqu'à 85 m.s^{-2}.
  - Electronique : le gain de l'amplificateur pourrait plafonner. Encore une fois impossible car ce phénomène aurait du intervenir lors de l'étalonnage. De plus, le tout début de la courbe montre que l'amplificateur n'a pas saturé.
- Le fait que nous ne voyions pas le pic des 2sec s'explique parfaitement si on prend en compte le fait que notre capteur a une inertie importante à cause notamment de la masselotte. Il n'est donc pas très sensible aux petits soubresauts de ce genre. C'est un filtre passe-bas !
- Nous observons une réelle différence entre les caractéristiques théoriques et les caractéristiques réelles du propulseur chamois. Ceci nous semble être la seule explication plausible au vu des résultats. Trajec surestimerait donc d'environ 15% les accélérations extrêmes subies par la fusée. Les calculs du logiciel se basant sur les courbes de poussées théoriques du propulseur chamois.

4.2 Conclusion

Nous sommes très contents des résultats de ce capteur maison. En effet, il va pouvoir tenir une place de choix dans les comparaisons de valeurs entre nos 5 capteurs surtout que c'est également lui qui a la meilleure résolution au niveau de la chaîne d'acquisition des données avec 11bits significatifs. Ce capteur nous a parmi de mettre en évidence l'influence de la rotation de la fusée sur la transmission des données.

Plus généralement, c'est la démarche mise en œuvre pour réaliser notre propre capteur d'accélération qui nous satisfait beaucoup, d'autant plus que les résultats sont au rendez-vous. La réflexion menée de concert entre électronicien et mécanicien lors de la conception/dimensionnement du capteur à vraiment montré qu'une équipe doit véritablement être soudée et réfléchir collectivement.

5. Bibliographie

[1] Datasheet ADS 7823 convertisseur 12bits avec interface I²C, basse conso :

http://focus.ti.com/lit/ds/symlink/ads7823.pdf

[2] Datasheet INA114 amplificateur d'instrumentation bipolaire utilisé en unipolaire dans notre cas :

http://focus.ti.com/lit/ds/symlink/ina114.pdf

[3] Datasheet MAX7400 filtre du huitième ordre – technologie capacités commutées – très facile d'utilisation pour un résultat dont Francis Lesel se souvient encore …

http://pdfserv.maxim-ic.com/en/ds/MAX7400-MAX7407.pdf

[4] Datasheet LP2951 – régulateur de tension 3.3V/5V - 100mA – commandable et avec retour d'information sur le statut. Technologie LDO : faible tension de chute

http://www.onsemi.com/pub/Collateral/LP2950-D.PDF

[5] Datasheet AD584 – référence de tension : 2.5V/5V/7.5V/10V – un incontournable dont il faut se méfier…

http://www.analog.com/UploadedFiles/Data_Sheets/115302565AD584_b.pdf

[6] Datasheet 2N3906 – il est au 2N2222 ce que l'antimatière est à la matière : transistor baroudeur type PNP. Il est utilisé aussi ici pour les commandes logiques

http://www.semiconductors.philips.com/acrobat/datasheets/2N3906_3.pdf

[7] Datasheet UCC384. Texas Instruments. Référence low drop out de tension négative. Technologie CMOS de dernière génération.

http://focus.ti.com/lit/ds/symlink/UCC384.pdf

Note : Seuls l'UCC384 et l'INA114 (grand frère de l'INA122) sont des petits nouveaux dans cette liste très fermée des composants approuvés CLES-FACIL.

http://www.insa-lyon.fr/Insa/Associations/ClesFacil/

Pierre Fayet
fayetpierre@aol.com
Nicolas Chaléroux
nico@sdbdc.com

Projet : Axelle

11 Novembre 2003
Version 2.0

Accéléromètre ADXL 150.

I. Introduction.

L'objectif de ce document est de décrire la mise en œuvre d'un système de mesure d'accélération unidimensionnnel à partir d'un capteur simple disponible chez Analog Devices : L'ADXL 150. On y trouvera donc les schémas électroniques s'y rapportant, quelques photos présentant la carte au cours de sa réalisation et enfin la procédure d'étalonnage et les résultats des mesures effectuées par ce capteur.

II. Cahier des charges.

Il s'agit de réaliser un dispositif de mesure le plus fiable possible. Concrètement cela signifie que l'on attend de ce système un fonctionnement sans panne, mais il faut aussi que les mesures soient précises et renouvellables. Il faut également songer au fait qu'un capteur tel que l'ADXL 150 nécessite un positionnement particulier qu'il faudra prendre en compte lors de son intégration dans la fusée.

L'objectif de cette expérience est de mesurer l'accélération dans l'axe vertical de la fusée.

III. Le capteur.

Il est ici nécessaire de résumer les principales caractéristiques de l'ADXL 150 et aussi d'en rappeler le principe de fonctionnement.

III.1 Caractéristiques techniques

- Plage dynamique : 80dB
- Etendue de mesure : +- 50g, avec possibilté d'adaptation pour d'autres étendues de mesure.
- Non-Linéarité 0.2% de la pleine échelle,
- Sensibilité 38mV/g.
- Réponse en fréquence (@-3dB) : 1000Hz.
- Tension d'alimentation : 4 à 6V.
- Consommation: 1.8 à 3 mA.

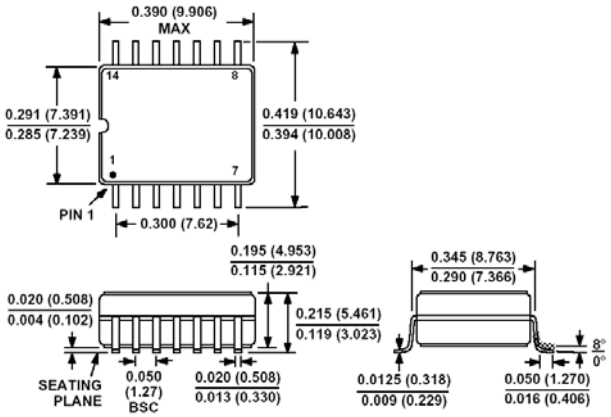
L' alimentation de ce composant ne pose pas de problème. Nous aurons une dynamique de ±25g ce qui permettra de voir sur les courbes d'éventuels à-coups du propulseur. La résolution théorique sera de l'ordre de 10mg que nous acquièrerons avec un convertisseur 12bits. Ce capteur commercial étant a priori le plus fiable nous l'échantillonnerons au maximum que nous pouvons.

A propos… Tant que je suis dans les les caractériqtiques techniques, on remarquera que ce capteur est assez compact :

OUTLINE DIMENSIONS

Dimensions shown in inches and (mm).

14-Lead Cerpac
(QC-14)



III.2 Principe de fonctionnement

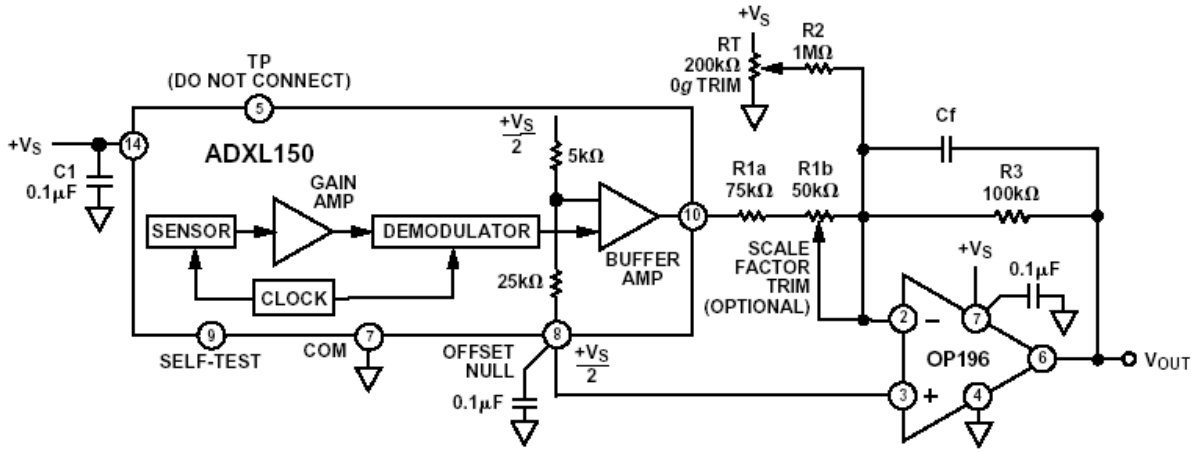


Schéma synoptique de l'accéléromètre ADXL150.

On voit sur ce synoptique que la nature même du capteur n'est pas spécifiée. Ce point est éclairci un peu plus loin dans la datasheet sans toutefois en révéler tous les détails. Il s'agit en effet d'une technologie propriétaire et l'on comprend qu'un constructeur comme Analog Devices reste relativement évasif sur ce sujet. Il s'agit en fait d'un capteur capacitif, qui comporte 42 microstructures élémentaires dont le principe est résumé ci-dessous :

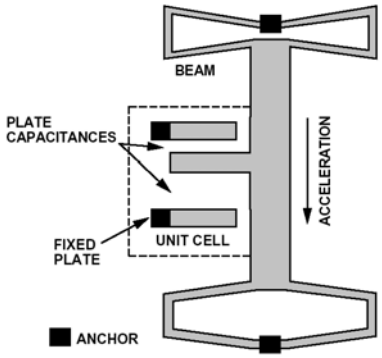
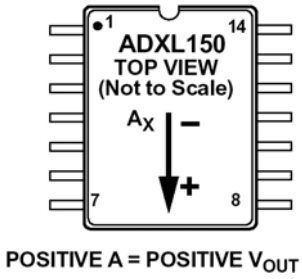


Schéma de principe d'une cellule élémentaire.

Cette figure aide à comprendre la figure 1. En fait les variations de capacité sont mesurées de manière classique et l'information sur l'accélération est convertie analogiquement en fréquence, ce qui justifie la présence d'une horloge (« clock ») sur le synoptique de la figure 1. L'information est ensuite convertie en tension par un convertisseur fréquence-tension. C'est une technique habituelle de mesure de capacité : en insérant une capacité variable dans un circuit oscillant, on fait varier la fréquence d'oscillation du système autour d'une fréquence de repos f_0.

Là où ce capteur est beaucoup moins classique, c'est dans la technologie employée pour réaliser cette capacité élémentaire variable présentée en figure 2, dans un volume extrèmement restreint. C'est là qu'intervient tout le savoir-faire du constructeur.

Comme nous le disions, ce capteur est sensible à son sens d'orientation. La figure ci-dessous indique dans quel sens une accélération positive peut être mesurée.

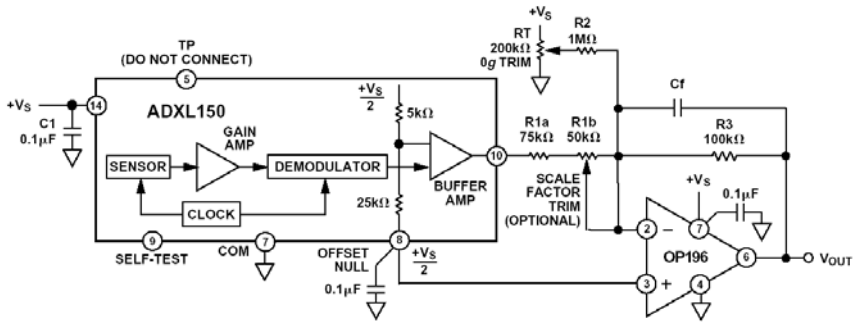


Orientation du capteur.

Il faudra donc implanter ce composant verticalement dans la fusée puisque l'on souhaite mesurer l'accélération de la fusée selon son axe vertical. On verra plus loin que ce capteur a pour cela été implanté sur un morceau de circuit imprimé lui-même solidaire d'une équerre en aluminium, qui, au passage, s'est révélé être un bon plan de masse et a amélioré le nombre de bits significatifs de la mesure. (Voir le chapitre 06 : Conversion Analogique-Numérique).

IV. Le circuit électronique adopté.

Il suffit de lire la datasheet et pour une fois on n'a pas besoin de se casser la tête. En prime le filtre (1er ordre) est donné aussi :

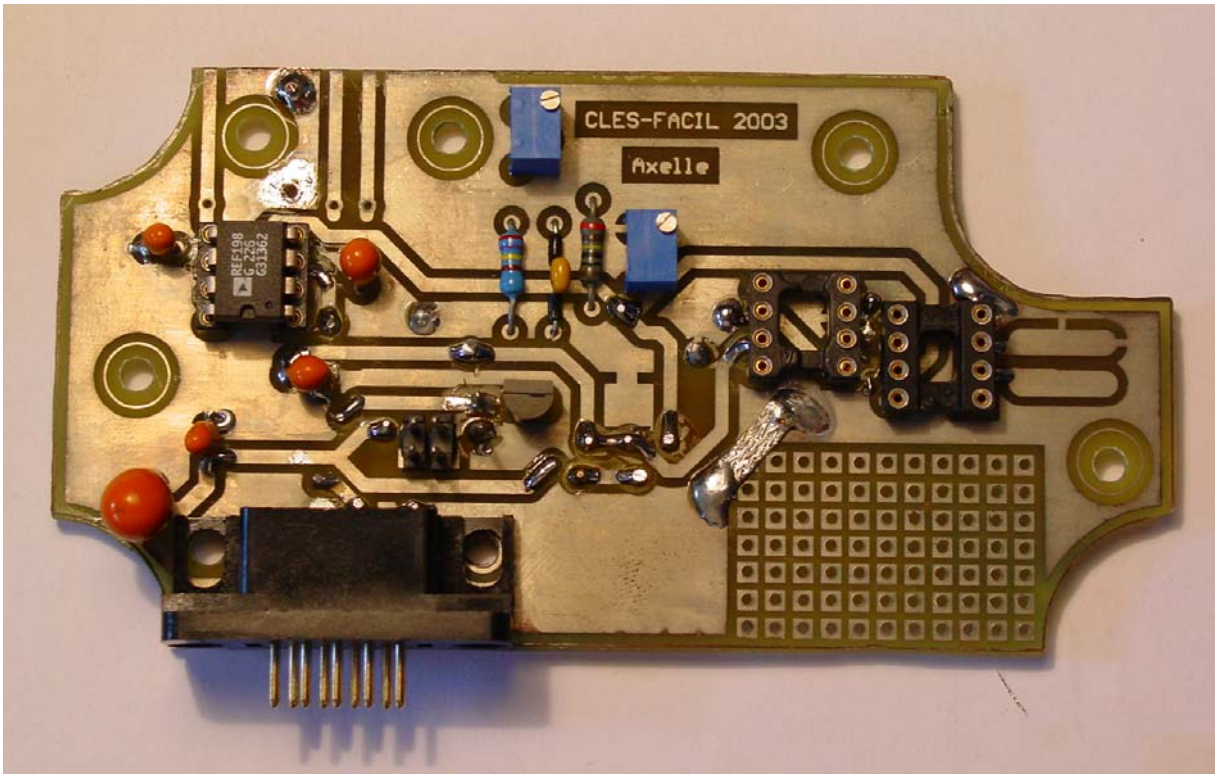


DESIRED OUTPUT SCALE FACTOR	F.S. RANGE	EXT AMP GAIN	R3 VALUE	Cf (µF) 100Hz	Cf (µF) 30Hz	Cf (µF) 10Hz
76mV/*g*	±25*g*	2.0	200kΩ	0.0082	0.027	0.082
100mV/*g*	±20*g*	2.6	261kΩ	0.0056	0.022	0.056
200mV/*g*	±10*g*	5.3	536kΩ	0.0033	0.010	0.033
400mV/*g*	±5*g*	10.5	1MΩ	0.0015	0.0056	0.015

Figure 20. One-Pole Post Filter Circuit with SF and Zero g Offset Trims

Ce montage permet d'ajuster l'offset du montage et éventuellement de choisir un coefficient d'amplification de manière à avoir la plus large dynamique possible. Il a été implanté tel quel dans la fusée avec une sensibilité de 100mV/g et un filtrage à 100Hz.

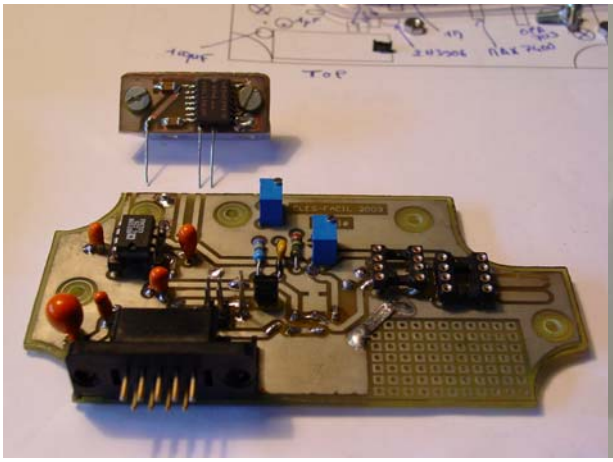
Voici la photo de la carte une fois réalisée :

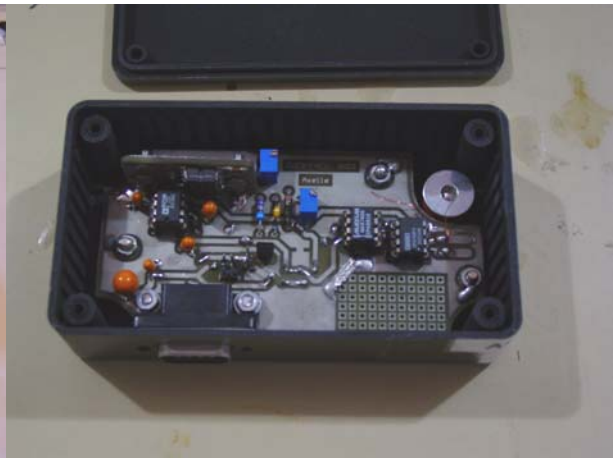


Cette carte est aussi conçue pour recevoir (partie de droite) le capteur piézo, c'est pourquoi un des angles (devinez lequel) n'a pas la même forme que les autres. On distingue aussi la référence de tension REF198 utilisée pour le convertisseur A/N et pour l'alimentation du système piézo. Le capteur ADXL150, l'ampli op 196 et le convertisseur sont quant à eux alimentés en +5V par un LDO LP2951. Tous ces composants (CMS) sauf le capteur ADXL150 sont implantés sous la carte et… j'ai pas la photo. On peut voir en haut à gauche de la carte l'emplacement prévu pour accueillir le capteur ADXL150 monté verticalement sur son équerre. On voit aussi en bas à droite une zone emplie de pastilles au cas où il faille ajouter un composant sur la carte. Le morceau de tresse quant à lui améliore l'équipotentialité du plan de masse. Pour le reste, les concepts chers au CLES-FACIL on été repris : implantation dans une boîte, alimentations commandées de l'extérieur etc…

Note : le convertisseur A/N n'est pas l'ADS7823 mais l'ADS7828 , car il dispose de huit entrées analogiques pour une seule adresse I2C.

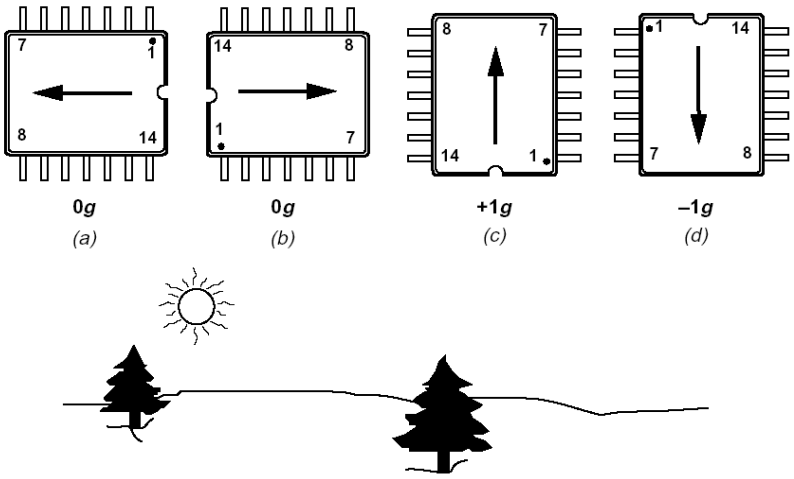
Voici encore deux autres vues du montage : l'une avec le capteur et l'autre avec la carte intégrée dans sa boîte. Remarquer aussi l'intégration du capteur piézoélectrique.





V. Etalonnage.

L'étalonnage de ce capteur est fait à l'aide de trois points –1g, 0g et 1g.
On utilise la méthode d'auto-étalonnage proposée par AD :

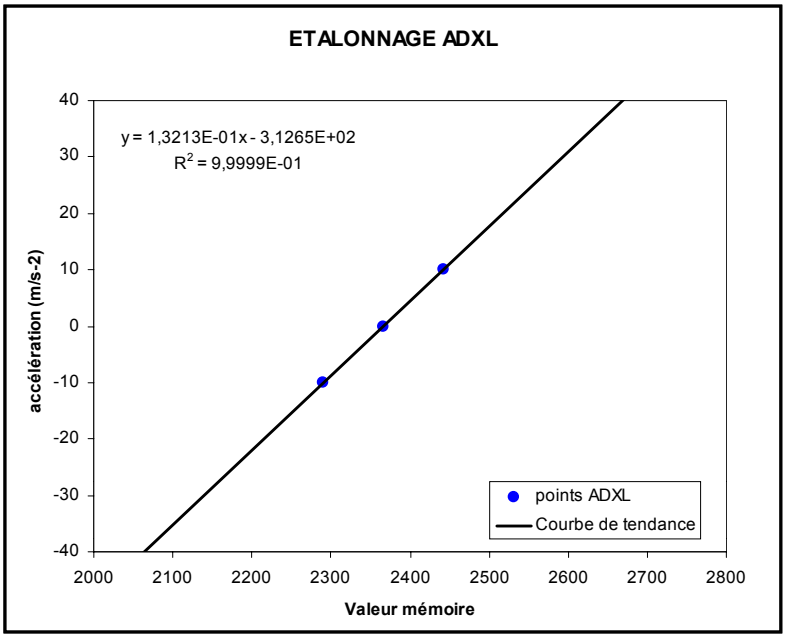


En tournant la boîte dans différents sens, on peut ainsi étalonner notre capteur. En pratique, il est préférable, ne serait-ce que pour des raisons de logique, de procéder dans l'ordre suivant : -1g (capteur tête en bas), 0g (capteur sur le côté, peu importe lequel) et 1g (capteur en position verticale). Il convient, bien sûr, de faire cela dans un lieu calme et d'agir avec des gestes précis. Cet étalonnage a été effectué deux fois à plusieurs mois d'écart, avec deux systèmes d'acquisition différents, mais en appliquant le même protocole. Elles ont donné sensiblement les mêmes valeurs. Les valeurs présentées ci-dessous sont celles qui ont été retenues pour l'exploitation globale des résultats.

Etalonnage ADXL	
Accélération (m/s-2)	Valeur mesurée
-10	2290,56
0	2365,92
10	2441,92

L'autre système d'acquisition par lequel il a aussi été possible de procéder aux essais et à l'étalonnage de ce capteur était constitué d'un ordinateur portable dont le port parallèle émulait le bus I2C.

La courbe d'étalonnage est donc la suivante :



En appliquant une approximation linéaire, on remarque une corrélation parfaite entre les points, ce qui nous permet d'obtenir l'équation d'étalonnage avec une très grande précision.

VI. Les résultats.

Voir les courbes données au chapitre 5 de ce compte-rendu d'expérience.

La courbe est offsetée et on n'a pas encore compris pourquoi. Cette courbe est pourtant conforme à celle donnée par Trajec en ce qui concerne l'allure générale, mais offsetée alors que l'étalonnage permet en théorie de prendre en compte des accélérations négatives, ou des décélérations en d'autres termes. Evidemment si l'on cherche à intégrer ou à dériver la courbe obtenue, c'est encore pire puisque les erreurs sont encore plus grossières.

VII. Conclusion.

C'est un système simple, séduisant, facile à mettre en œuvre, adapté à toutes sortes de problèmes et de plus en plus utilisé pour les fusex, mais… y'a quelque chose qui cloche. Nous nous sommes promis de tirer cela au clair malgré la clôture du dossier Axelle. Peut-être finirons-nous enfin par expliquer ce qui pour l'instant reste à élucider.

http://www.insa-lyon.fr/Insa/Associations/ClesFacil/

Pierre Fayet: fayetpierre@aol.com

Projet : Axelle

10 Décembre 2003

Version 2.0

04. Accéléromètre piézoélectrique.

En cours d'année le CLES-FACIL a reçu une proposition émanant du département de Génie Electrique de l'INSA. Il s'agissait en effet de mettre en œuvre un capteur d'accélération de type piézoélectrique.
Les composants piézoélectriques ont pour principale caractéristique de générer une tension à leurs bornes lorsqu'on leur applique une contrainte mécanique externe et inversement : si on leur applique une tension, leur forme mécanique varie également. La première propriété est du reste utilisée pour certains microphones…

Calcul et dimensionnement de ce capteur.

Le capteur mis à notre disposition est caractérisé par deux paramètres :
d_{33} : 257pc/N, ce qui définit la fonction de transfert de ce capteur, et sa capacité équivalente : C=3.05nF.

On peut alors déduire la charge du composant en fonction de la contrainte mécanique qui lui est appliquée.

$$Q = d_{33} F$$

Avec F : force appliquée au capteur, en Newtons (N).
On sait que la fusée subit une accélération variant entre 0 et 10g. On choisit de coller une masse de deux grammes sur ce capteur, afin de pouvoir mieux visualiser l'accélération. En effet, cette masse permet d'adapter l'étendue de mesure de ce capteur aux contraintes se trouvant dans la fusée.

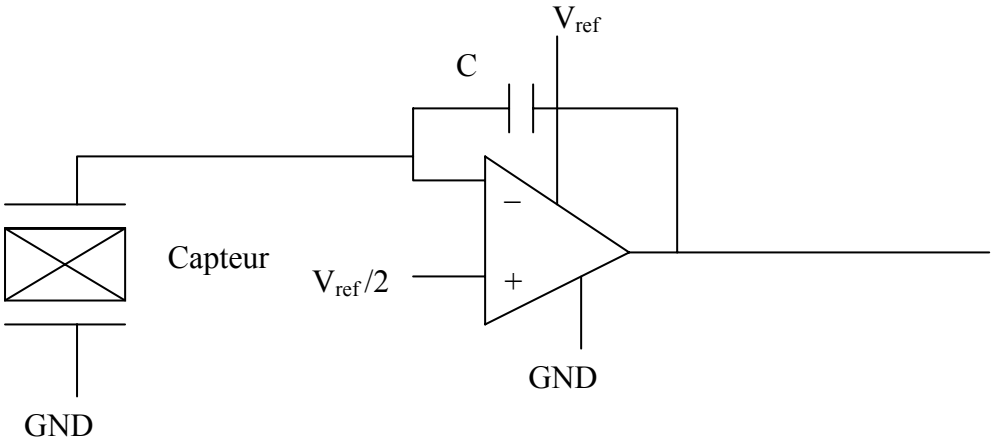
Pour l'accélération maximale la charge sera:

$$Q_{max} = d_{33} M g = 250.10^{-12} \times 2.10^{-3} \times 10 \times 9.81 = 4.905.10^{-11} C$$

Soit 49 pC.

L'amplificateur de charge.

Un amplificateur de charge est un montage électronique voisin du classique montage intégrateur permettant d'obtenir en sortie de montage l'image de la charge présente en entrée.



Or on sait que Q=C.V ce qui permet de déduire C = 10pF environ, pour une tension V = Vref = 4.096V, qui est la tension fournie par la référence Analog Devices REF 198. L'amplificateur opérationnel choisi pour la réalisation de ce montage est l'OPA 703 (Burr-Brown), en raison de son très faible courant de décalage (de l'ordre de quelques pA).

Le principal défaut de ce montage, ici dans sa version unipolaire est que la tension d'offset, qui est donc égale à $V_{ref}/2$ charge le condensateur jusqu'à ce que la saturation soit atteinte, c'est pourquoi j'ai ajouté en parallèle sur le condensateur C une résistance de forte valeur (10MΩ), de façon à éviter la saturation. On introduit donc une constante de temps dans ce système de mesure qui fait qu'après un choc sur le capteur, la tension en sortie de l'amplificateur de charge revient à Vcc/2 en très peu de temps, compte tenu de la faible valeur du condensateur C. On pourrait envisager d'augmenter la valeur de la résistance en parallèle avec C, mais en général de fortes valeurs sont assez difficiles à trouver et leur mise en oeuvre nécessite quelques précautions en termes de CEM. En effet une forte impédance dans un circuit est assez mal immunisée contre un signal perturbateur.

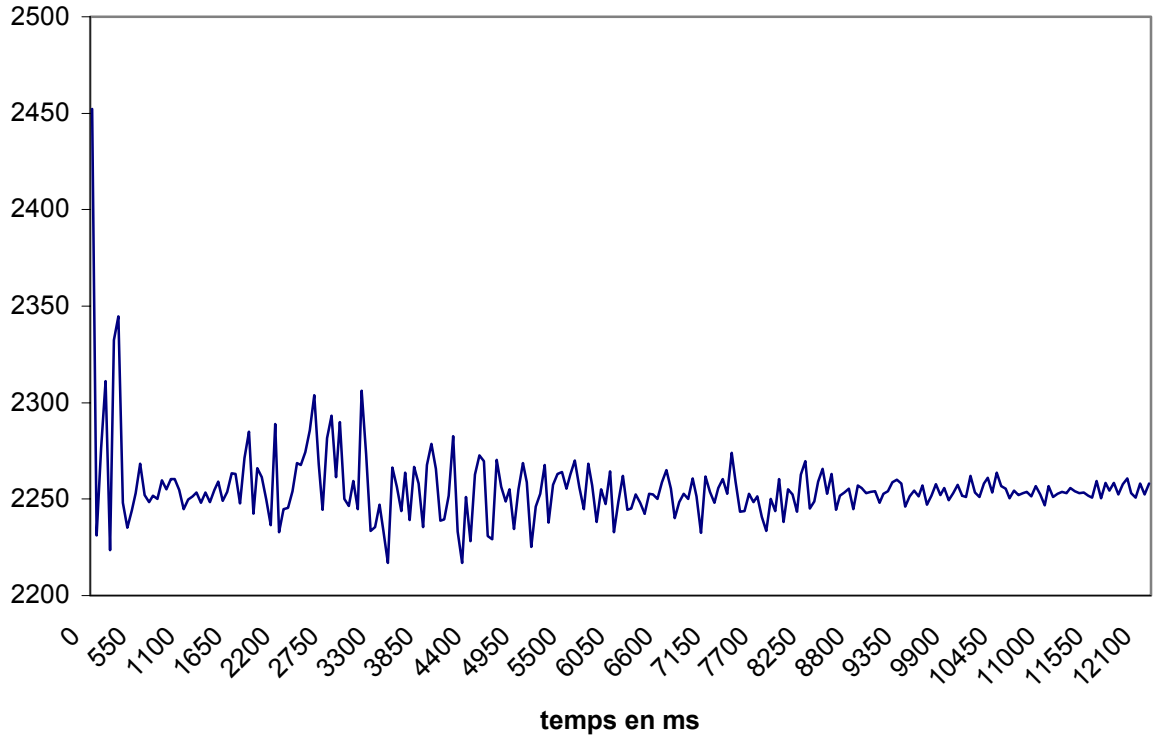
Le capteur est collé à l'Araldite sur un support plastique, lui-même collé au fond de la boîte destinée à abriter la carte qui comporte à la fois ce capteur et aussi le capteur ADXL 150. La sortie de l'ampli de charge est ensuite directement reliée à l'entrée d'un filtre à capacités commutées MAX7400 que nous avons déjà évoqué dans d'autres chapitres de ce document aussi je ne m'apesantirai pas davantage sur ces composants. La fréquence de coupure utilisée était de 17Hz environ comme pour les autres capteurs

L'étalonnage.

Compte tenu de la nature de ce capteur, et de sa constante de temps, dûe, je le rappelle, à la mise en parallèle d'une résistance et d'un condenstateur dans la boucle de réaction de l'AOP, il n'est pas possible de procéder à un étalonnage de ce capteur. On peut tout au plus s'assurer de son bon fonctionnement en tapotant la boîte et en observant la tension en sortie de montage à l'oscilloscope. Détail amusant : en augmentant la masse posée sur le capteur, il a été possible de visualiser les effets de la voix des personnes présentes dans la pièce où l'essai a eu lieu (comme pour le Pitot). Cela se comprend aisément si l'on songe que des capteurs de cette famille on été autrefois (ce n'est plus guère à la mode) utilisés pour faire des capteurs microphoniques

et aussi pour la lecture des disques vinyles. En général ces capteurs sont bien adaptés pour des systèmes basses fréquences et se comportent mécaniquement comme un filtre passe - bas.

Les résultats :
Voici la courbe obtenue lors du vol :



Cette courbe n'est guère exploitable en l'état. Elle montre pourtant que des vibrations mécaniques se sont propagées dans la fusée lors du vol, mais l'information apportée par ce capteur est à prendre avec réserve.

Pour une interprétation dynamique des phénomènes, il faudrait connaître la fréquence de résonance du capteur (cela serait faisable à condition de faire un essai sur table vibrante en traçant la courbe de réponse de ce capteur) et tenir compte également du filtre anti-repliement dont la fréquence de coupure est assez basse. Il est probable qu'une fréquence d'échantillonage plus élevée soit nécéssaire, car il ya de fortes chances pour que la fréquence des vibrations internes à la fusée soit bien supérieure à 20Hz.

Pour une interprétation en mode quasi statique des phénomènes, c'est à dire à très basse fréquence, (cela permettrait en théorie d'avoir une idée globale des différentes phases de vol) il est évident que le montage proposé ci-dessus ne convient pas et ce pour deux raisons :

- L'alimentation unipolaire du montage induit une tension d'offset franchement nuisible à la mesure, puisqu'elle charge le condensateur de l'amplificateur de charge qui est censé récupérer la charge provenant du capteur soumis à une contrainte mécanique.
- La seconde raison est le corollaire de la précédente : le fait d'ajouter une résistance en parallèle sur le condensateur de l'ampli de charge est néfaste à la mesure, puisque l'on introduit dans le système une constante de temps.

On peut remédier à ces problèmes en mettant en œuvre les deux dispositions suivantes :

- Alimentation symétrique régulée pour l'AOP du montage. Choisir un AOP avec une très faible tension d'offset, ou bien un AOP dont l'offset est réglable via un potentiomètre. Cet AOP doit avoir un très faible courant de décalage (biais current : Sur ce point l'OPA703 est très bon).
- Munir la carte d'un système permettant à intervalles réguliers de « vider » le condensateur C, provoquant ainsi un reset système, par exemple un transistor MOS fonctionnant en commutation en parallèle sur ledit condensateur. Cela n'était pas possible pour Axelle, car il aurait fallu que ce capteur soit intégré à la conception de la fusée dès le départ, de façon à ce que le microcontrolleur qui gère les mesures de la fusée puisse également gérer la commande du transistor via le bus I2C. Cela aurait impliqué une architecture légèrement différente, bienqu'assez simple, mais qui ne pouvait en aucun cas être intégrée en cours d'année.

Conclusion :

Ce type de capteur a tout à fait sa place dans une fusée. Néanmoins, si sa mise en œuvre est assez simple, en revanche il faut réfléchir de plus près à la nature même de l'expérience que l'on souhaite réaliser et effectuer un certain nombre d'essais et de mesures préliminaires: s'agit-il de mesures en statique ? en dynamique ? Ce type de capteur est - il avantageux par rapport à un capteur plus évolué disponible dans le commerce (hormis le coût) ? Autant de questions auxquelles il convient de répondre minutieusement avant de se lancer dans une réalisation de ce genre.

http://www.insa-lyon.fr/Insa/Associations/ClesFacil/

Laurent Ruet: laurent_ruet@hotmail.com

Projet : Axelle

15 Novembre 2003

Version 3.0

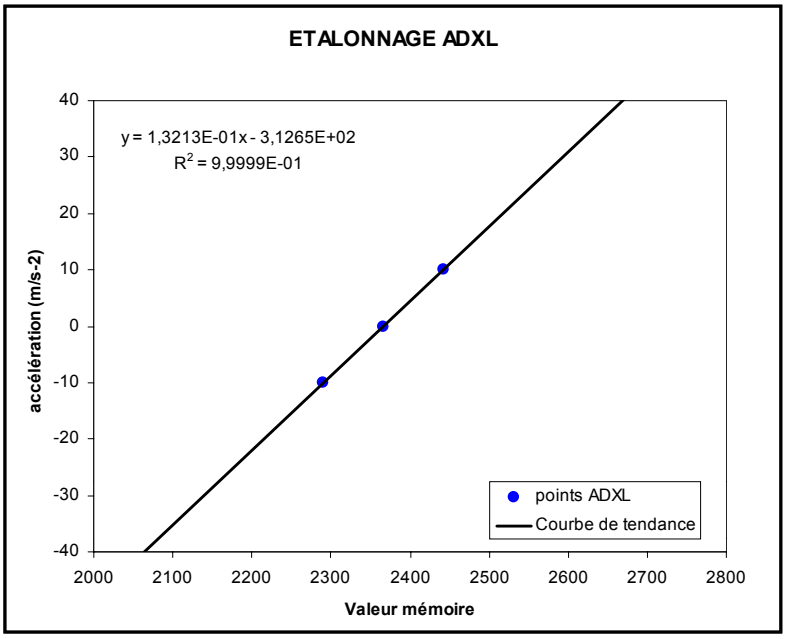
05. Exploitation globale des résultats.

1. L'ADXL

1.1. Etalonnage

L'étalonnage de ce capteur d'accélération se fait sur 3 points, les seuls réalisables facilement, à savoir 3 mesures à -10, 0 et 10m.s-². On mesure la valeur transmise au processeur et on établit une correspondance entre cette valeur et la valeur physique :

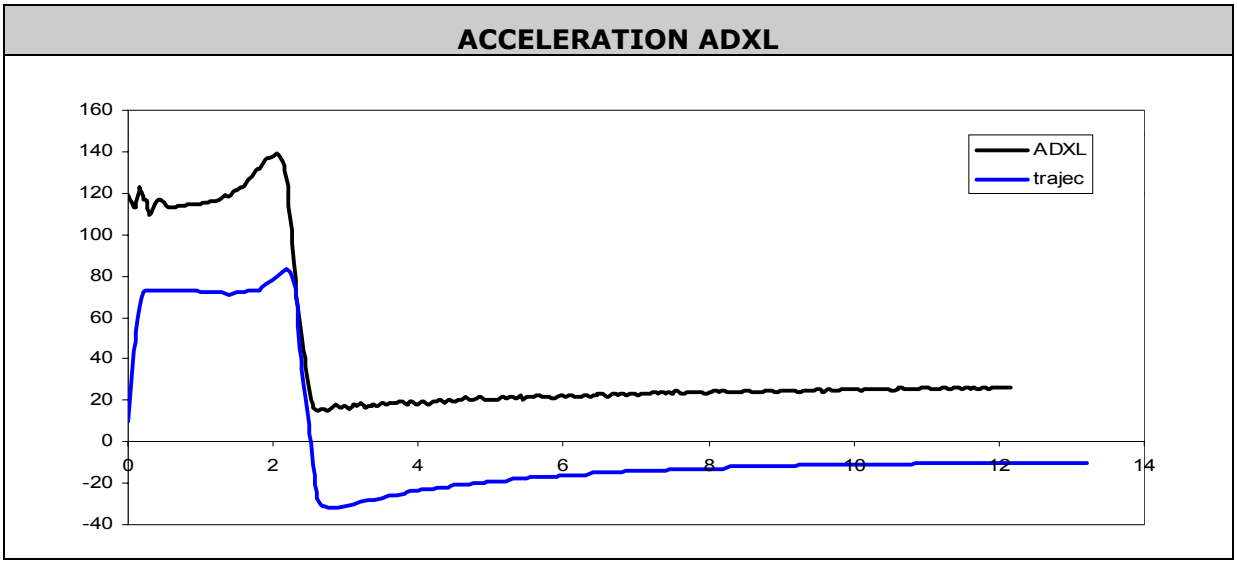
Etalonnage ADXL	
Accélération (m/s-2)	Valeur mesurée
-10	2290,56
0	2365,92
10	2441,92



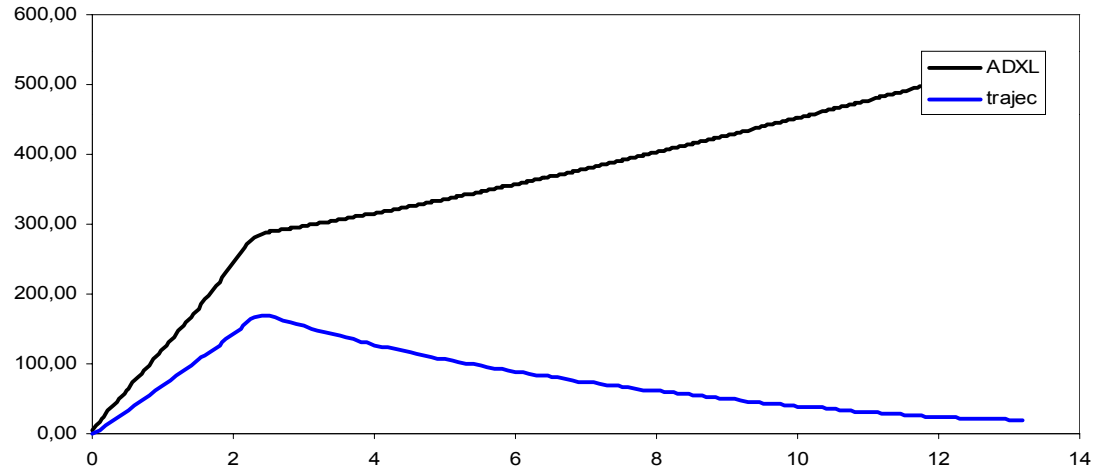
En appliquant une approximation linéaire, on remarque une corrélation parfaite entre les points, ce qui nous permet d'obtenir l'équation d'étalonnage avec une très grande précision.

1.2. Résultats et commentaires

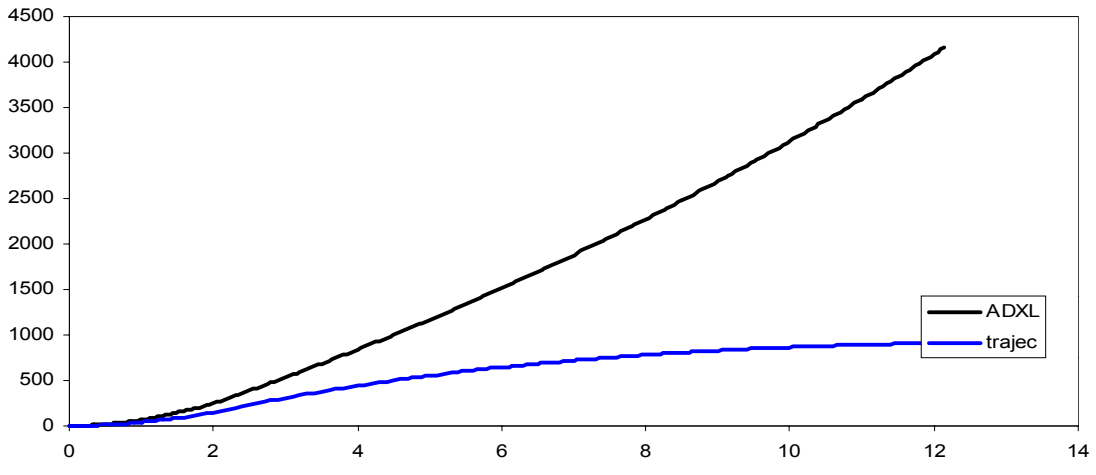
Les résultats de l'ADXL sont comparés avec ceux donnés par trajec :



VITESSE ADXL



ALTITUDE ADXL



On voit évidemment au premier coup d'œil que ces courbes sont inexactes. Il faut savoir que la vitesse et l'altitude sont issues de l'intégration de la valeur mesurée, ce qui renforce cette erreur.

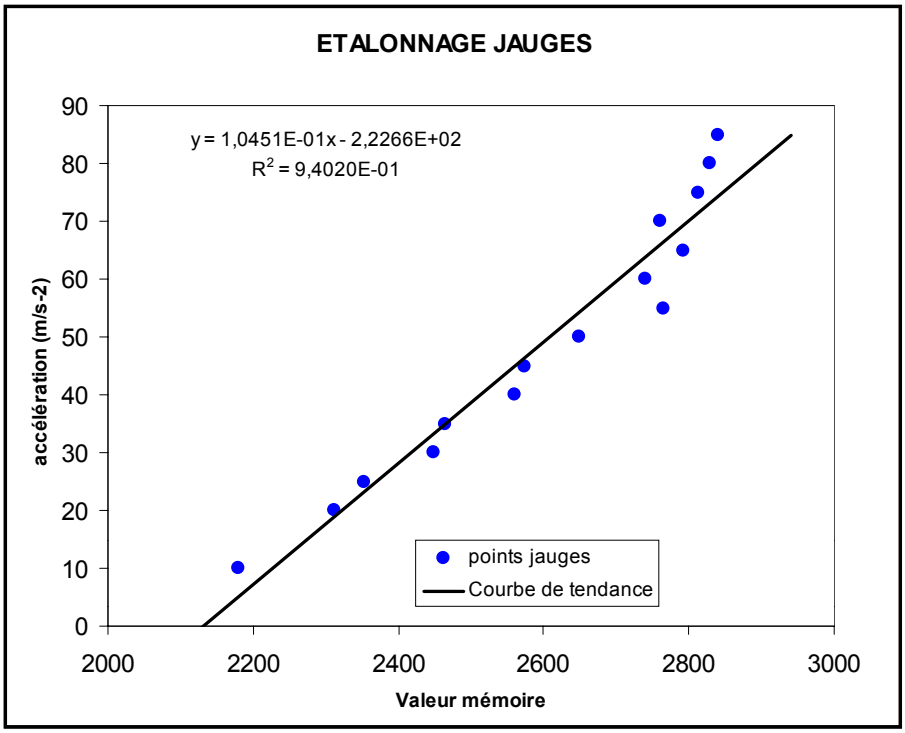
Dans le cas de l'accélération, on remarque que la forme de la courbe est très bonne mais qu'il y a une sorte de décalage vers le haut. On pense donc principalement à un problème d'étalonnage qu'il nous a été toutefois impossible d'identifier clairement, ces courbes sont donc inexploitables.

2. Les jauges de contrainte

2.1. Etalonnage

Le capteur à jauges de contrainte peut être étalonné sur toute sa gamme d'utilisation grâce à un système de masselottes représentant les diverses accélérations que subira la lamelle. On va donc effectuer une dizaine de mesures :

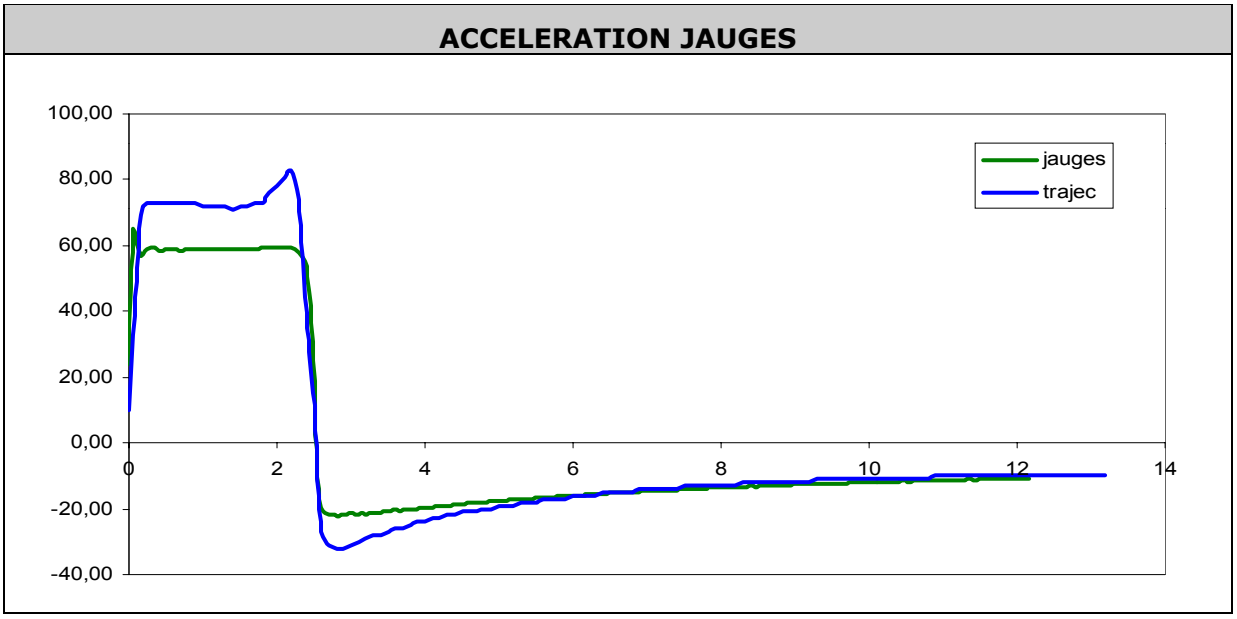
Etalonnage jauges	
Accélération (m/s-2)	**Valeur mesurée**
10	2179,01
20	2311,39
25	2352,15
30	2448,02
35	2464,22
40	2561,47
45	2573,52
50	2650,25
55	2766,36
60	2740,97
65	2792,88
70	2761,15
75	2812,11
80	2829,97
85	2841,59

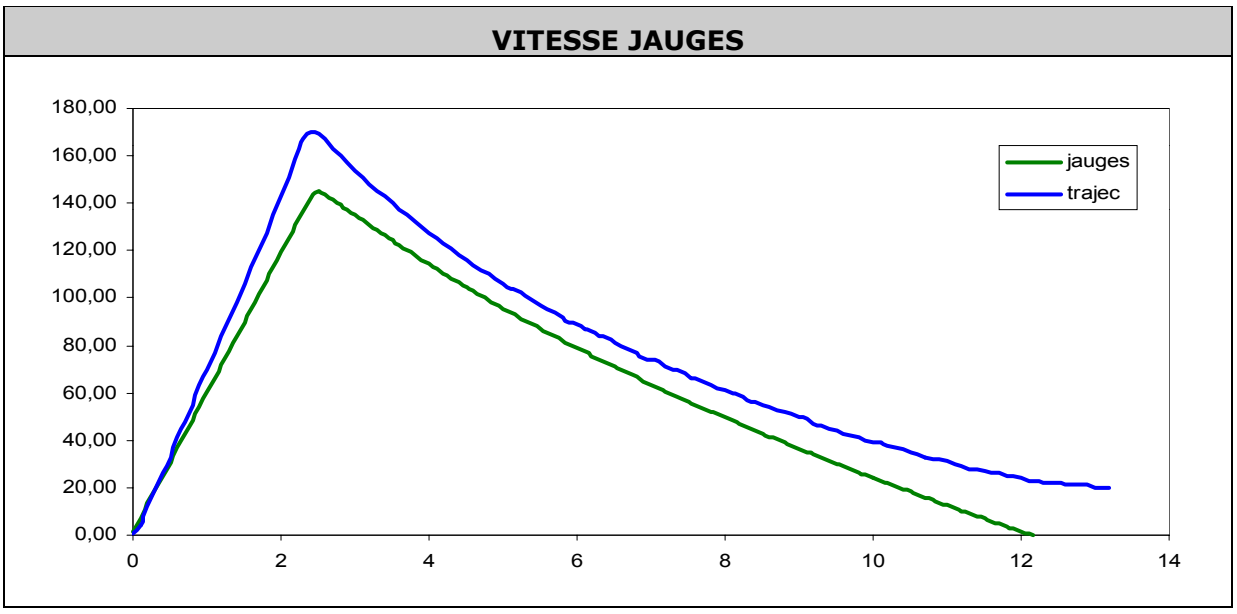


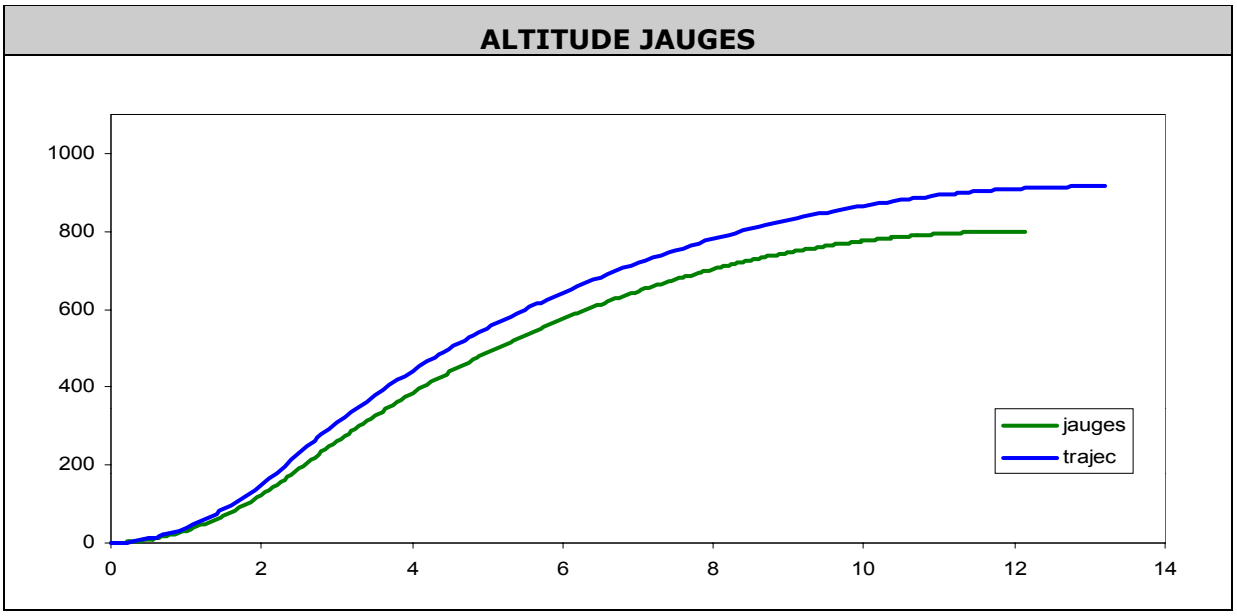
La disposition des points laisse supposer un comportement qui n'est pas parfaitement linéaire, en effet on remarque une sorte de saturation pour les grandes valeurs d'accélération et on utiliserait plus naturellement une forme exponentielle à la courbe d'étalonnage. Toutefois, le phénomène physique, on le sait, est parfaitement linéaire (voir partie jauges), d'autre part, appliquer une équation non linéaire est très complexe notamment au niveau des valeurs. Nous resterons donc sur une forme linéaire et le coefficient de corrélation est de toute façon très satisfaisant.

2.2. Résultats et commentaires

Les résultats du capteur à jauges de contrainte sont comparés avec ceux donnés par trajec :







Tout comme le capteur précédent, il ne faut pas oublier que l'on a mesuré ici l'accélération et que l'on a intégré la mesure pour obtenir vitesse et altitude.

On s'intéresse donc en particulier à l'accélération. On remarque tout d'abord, dans les premières millisecondes, quelques oscillations, pendant cette période, la fusée parcourt la rampe et en sort, il est donc très probable que les frottements générés par les patins, la sortie de rampe et la prise au vent soudaine, génèrent des perturbations et des vibrations, une partie de ces perturbations peut alors influencer directement dans l'axe de la fusée et être mesurés par nos capteurs.

On mesure ensuite un palier d'accélération, bien que logique, ce palier est toutefois suspect lorsqu'on le compare à la valeur de trajec. Il est difficile de juger quelle valeur est la plus correcte (nous reviendrons sur ce point plus tard) mais il est toutefois étrange de ne pas visualiser une augmentation brutale de l'accélération en fin de propulsion (2,5 sec), en effet cette accélération brutale que montre trajec est normale et commune aux moteurs à poudre (perte de poids soudaine par combustion brutale de toute la poudre restante). Notre capteur semble saturer, nous avions déjà observé ce phénomène lors de l'étalonnage. Il nous est très difficile d'expliquer cette saturation, physiquement, elle est illogique tout comme électroniquement, il n'y a aucune raison qu'elle ne se produise.

La chute d'accélération se produit au même instant pour la mesure et la simulation, preuve que le temps de poussée indiquée pour le chamois est parfaitement exact.

Nous remarquons enfin que la décélération est plus faible pour la mesure que pour la simulation trajec. Il peut y avoir plusieurs raisons à ce phénomène, en dehors d'une légère erreur de mesure, on peut aussi imaginer que les frottements de l'air sont mal gérés par trajec.

Enfin, la décélération se stabilise bien autour de -1g, la fusée n'est alors quasiment plus soumise à la résistance de l'air à cause de sa faible vitesse, et va atteindre son apogée (où les mesures s'arrêtent).

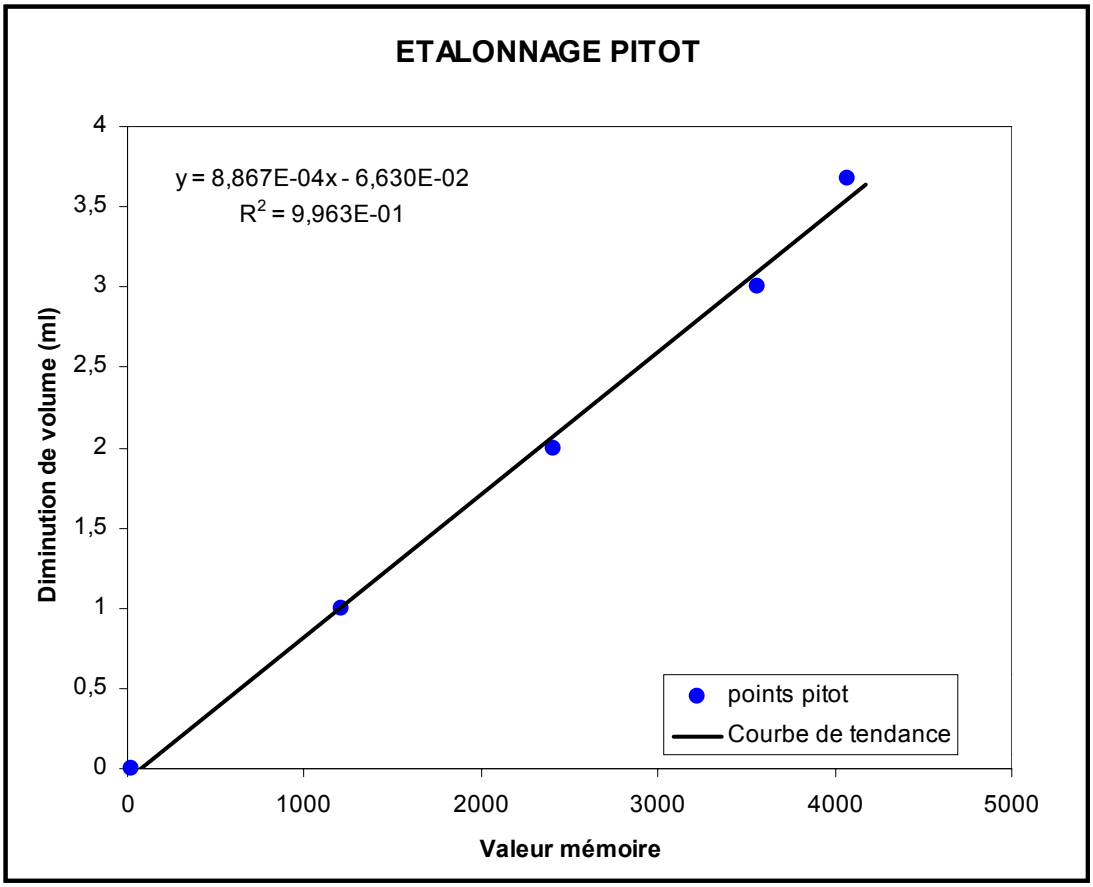
3. Le tube de pitot

3.1. Etalonnage

Le tube de Pitot ne peut pas (ou difficilement) être étalonné à partir de la vitesse. Il est beaucoup plus facile de mesurer la différence de pression au niveau du capteur, on utilise ensuite des lois physiques pour remontrer à la vitesse et obtenir l'équation d'étalonnage :

Etalonnage pitot	
Diminution Volume (ml)	Valeur mesurée
0	20
1	1213
2	2415
3	3560
3,67	4072

Pref	1,00E+05
Vref	1,835E-05
ro	1,225

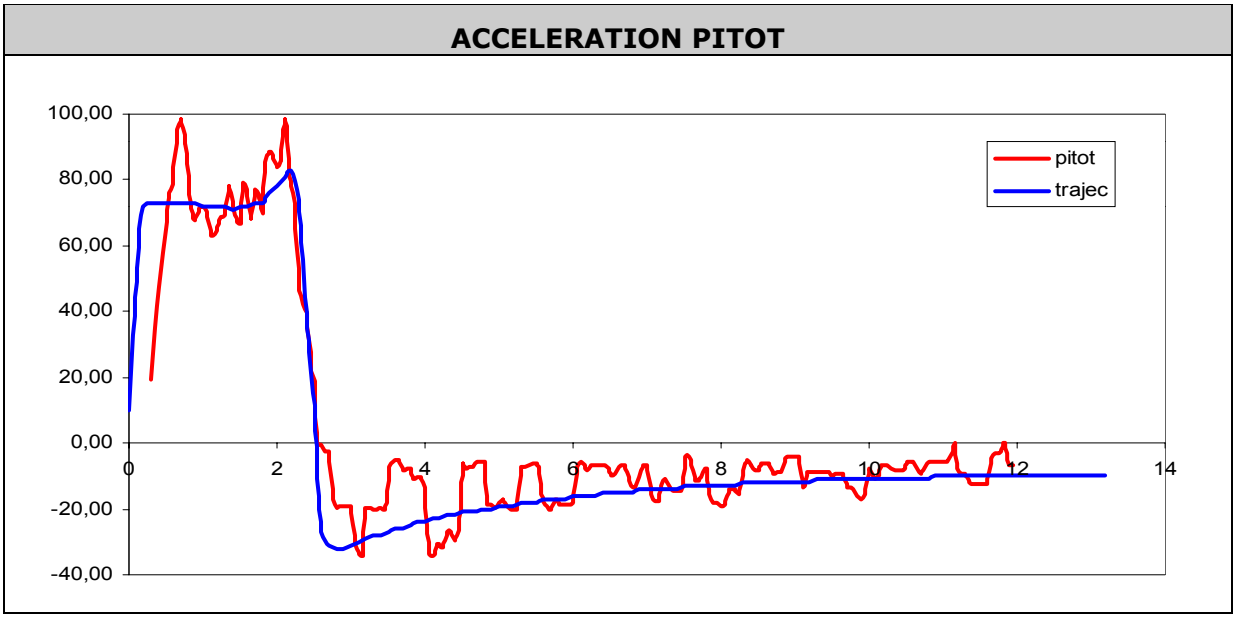


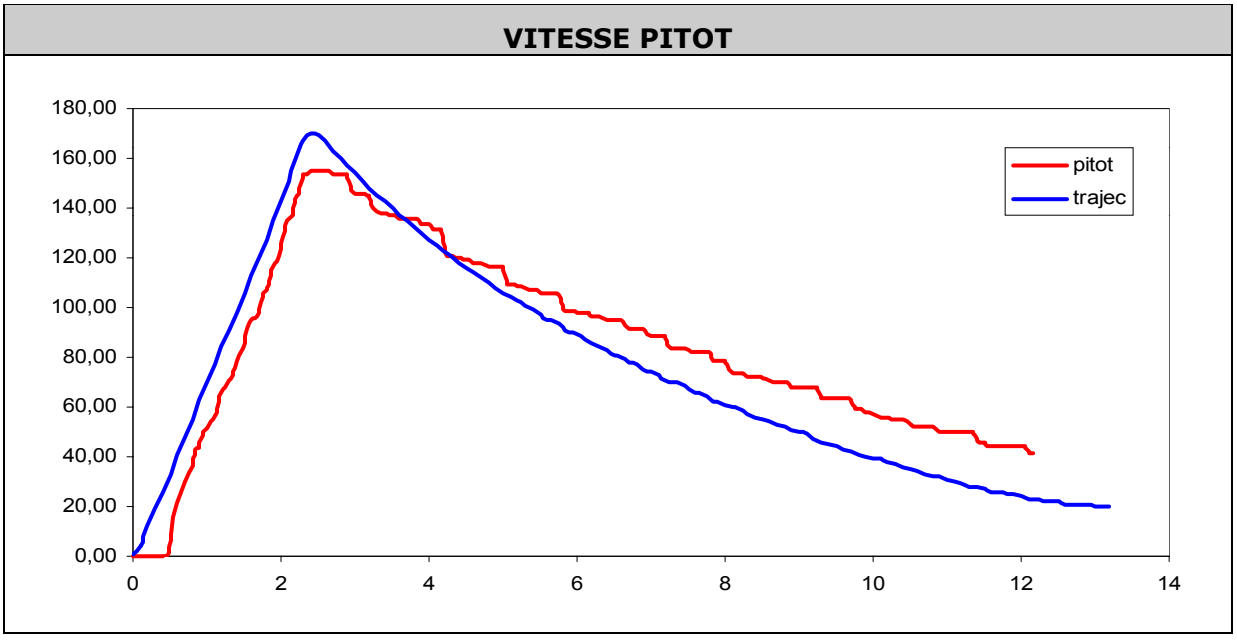
La corrélation est excellente et nous pouvons donc remonter à l'équation d'étalonnage sans difficultés :

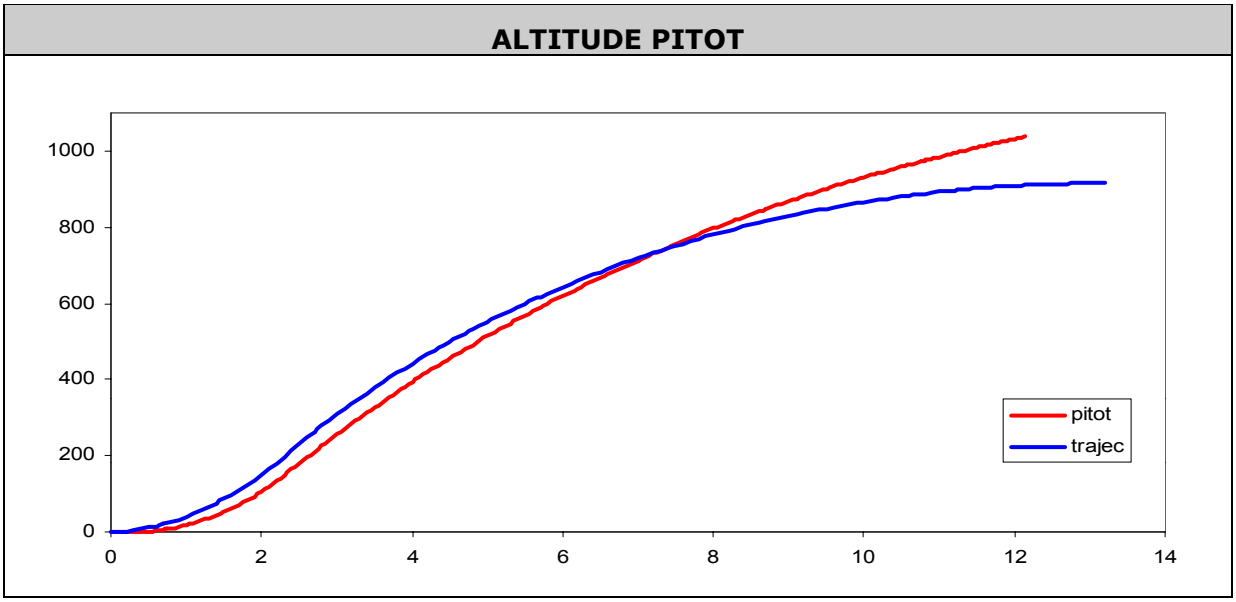
$$V = \sqrt{\frac{2}{\rho} \times \left(\frac{(P_{ref} \times V_{ref})}{(V_{ref} - (8.867 \times 10^{-4} \times v_{mesuré} - 6.63 \times 10^{-2}) \times 10^{-6}} - P_{ref} \right)}$$

3.2. Résultats et commentaires

Les résultats tube de Pitot sont comparés avec ceux donnés par trajec :







Dans ce cas, c'est la vitesse qui est mesurée. L'accélération est obtenue par dérivation, ce qui fait donc perdre de la qualité à la mesure (très nettement visible sur les courbes) tandis que l'altitude est obtenue par intégration (gain de qualité).

On voit que d'allure générale, la vitesse obtenue par la mesure est très proche de celle obtenue par simulation informatique. Il y a aussi un autre phénomène très intéressant à observer sur cette courbe : Le tube de Pitot est reconnu pour être un assez mauvais capteur dynamique, c'est-à-dire que dans un flux d'air dont la vitesse évolue, il mesure très mal les variations de vitesse, il est au contraire beaucoup plus efficace lorsque la vitesse du flux s'est stabilisée. On observe exactement ce phénomène ici, le Pitot engendre un retard sur la mesure, lors de la prise de vitesse, on remarque que la mesure a un léger retard d'environ 0,5 secondes, de même lors de la diminution de vitesse, avec un retard un peu plus important.

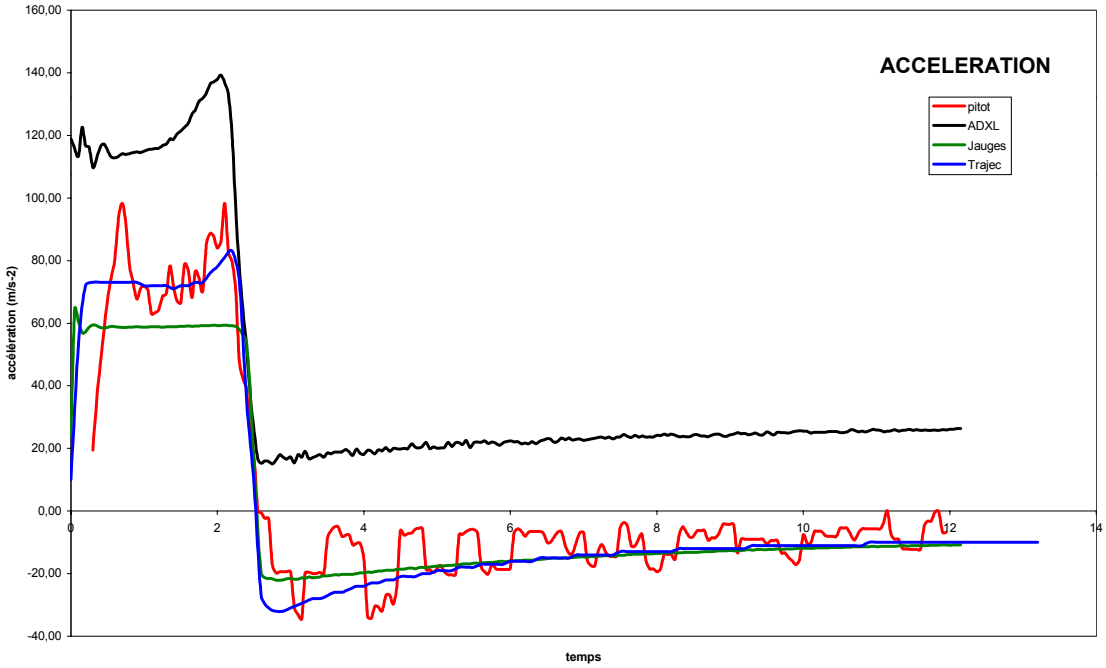
A quoi peut être du ce retard ? Comme nous l'avons vu, le Pitot ne mesure pas directement la vitesse mais la pression. Cette pression est due à la vitesse et ne varie pas instantanément avec celle-ci, surtout au fond du tube de Pitot et du tube en plastique. Lorsque le capteur « voit » une certaine pression, celle-ci ne correspond déjà plus à celle de la fusée car elle évolue très rapidement.

Pour mesurer des vitesse instantanément, il faut utiliser des capteurs qui ne sont pas soumis aux inerties physiques (pression températures) mais à des inerties électroniques qui sont beaucoup plus faibles (voir système anémomètre à fil chaud bog).

4. Exploitation générale

On va tracer toutes les mesures sur le même graphique :

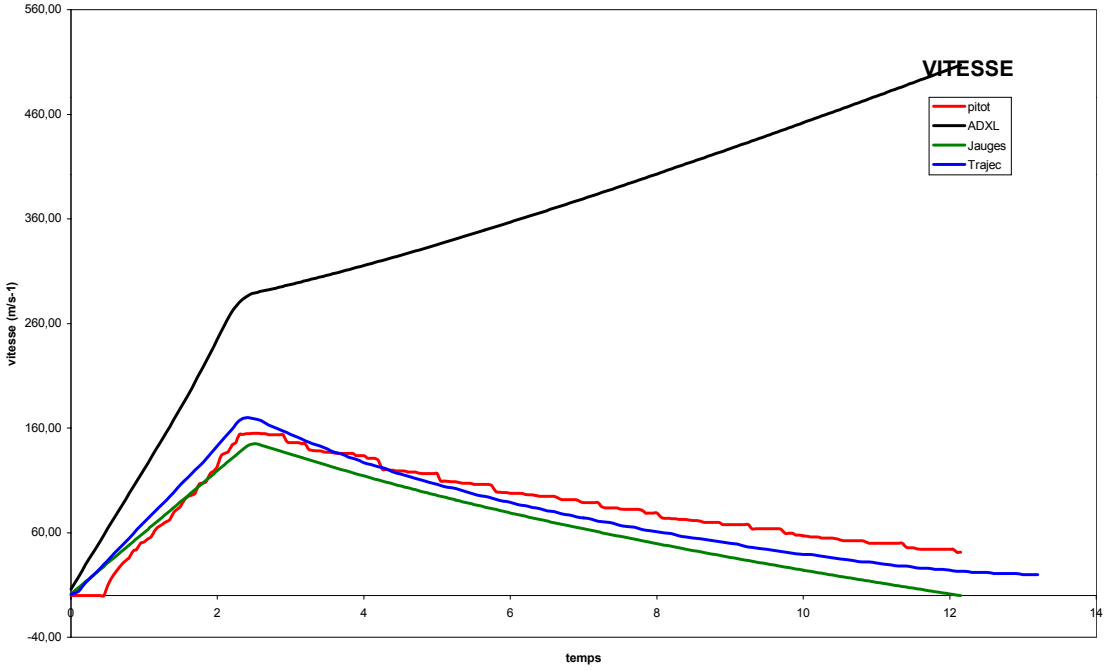
4.1. Accélération



Dans ce cas, l'accélération donnée par le tube de Pitot n'est pas une donnée très fiable (voir plus haut). Nous nous attarderons surtout à comparer les 3 autres courbes. Malheureusement, l'absence d'une mesure convenable par le capteur ADXL rend l'exploitation très difficile. Il nous est par exemple impossible de conclure sur l'accélération maximale (6g pour les jauges et 8g pour trajec).

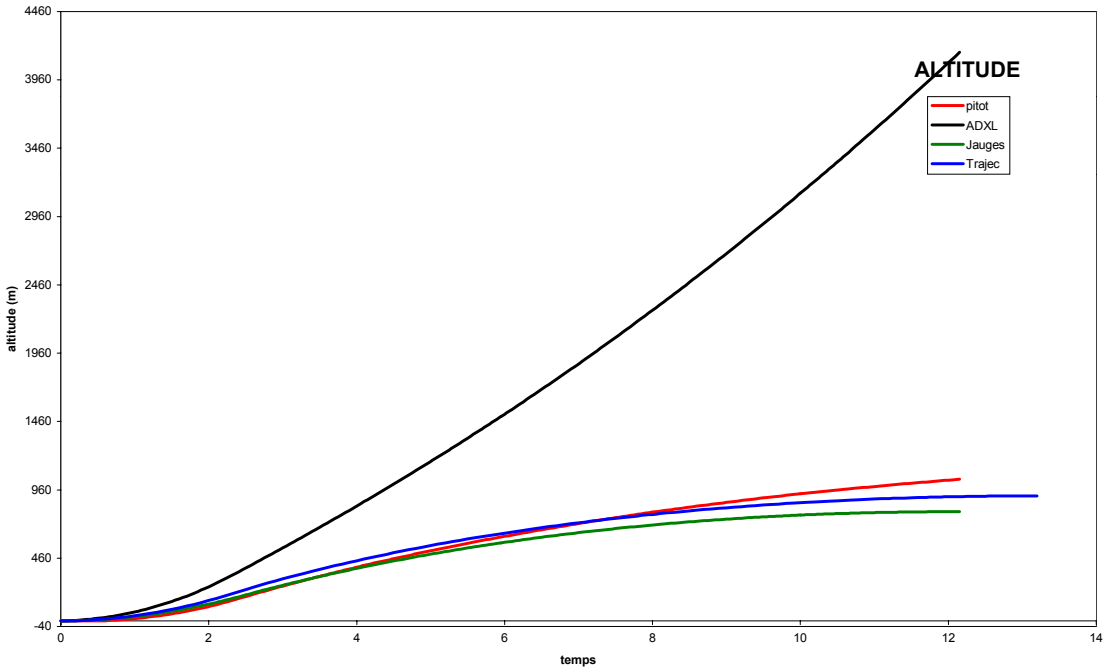
En revanche, on peut noter qu'au niveau de la décélération, toutes les courbes de mesure (Pitot compris) présentent une pente plus douce, ce qui laisse à penser que trajec surestime le coefficient de pénétration dans l'air de la fusée ou gère mal le phénomène de résistance aérodynamique.

4.2. Vitesse



Ici encore, l'exploitation est difficile du fait de la mesure erronée de l'ADXL, cela ne permet pas de conclure sur telle ou telle évolution de vitesse. On remarque toutefois une vitesse maximale un peu plus faible pour les mesures que pour la simulation.

4.3. Altitude



5. Conclusion

Les mesures effectuées sont de plutôt bonne qualité pour des capteurs non industriels conçus et fabriqués par des étudiants en un temps réduit. On remarque certes certains écarts par rapport aux simulations trajec, mais la plupart de ces écarts sont explicables et il faut de toute façon se laisser une marge de calage entre le modèle et la réalité. Les mesures effectuées par les capteurs Pitot et à jauges de contraintes sont très acceptables et nul doute que quelques années de tests et d'évolution rendront ces mesures encore meilleures.

Le problème reste toutefois le capteur ADXL, dont le problème d'étalonnage ou d'exploitation n'a pas été résolu. Cela est d'autant plus gênant que ce capteur était considéré comme important pour l'exploitation, le capteur étant en partie industriel, cela aurait permis une comparaison très objective avec les autres mesures et simulations.

CLES-FACIL

http://www.insa-lyon.fr/Associations/ClesFacil/

Jérôme HAMM
jerome.hamm@sdbdc.org

Projet : Axelle

31 Août 2003
Version 2.0

06. La conversion analogique / numérique.

Les systèmes de stockage de données et de télémesure d'aXelle sont totalement numériques. C'est pourquoi les grandeurs électriques fournies par ses différents capteurs doivent être échantillonnées et numérisées.

1. Le concept des boîtes-capteurs :

Cette année, nous avons voulu produire des mesures de qualité, et pour ce faire nous avons travaillé sur plusieurs points. Nous avons tout d'abord décidé de répartir les différentes alimentations de la fusée. Puis nous avons pensé placer chaque capteur dans une boîte blindée électromagnétiquement. Ceci assurait de l'immunité des mesures aux perturbations EM, au moins au point de production de ces mesures. Comme est il est plus difficile de transporter des signaux analogiques que numériques (comme nous avons pu, à nos dépends, nous en rendre compte sur ELA), nous avons introduit des convertisseurs analogiques-numériques dans chaque boîte.

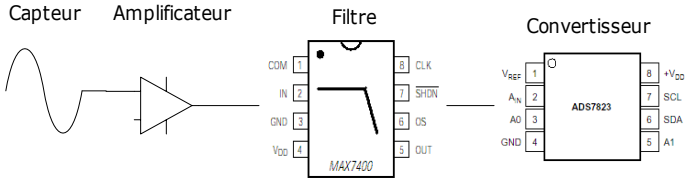


Figure 1 : Chaîne d'acquisition.

Chaque boîte est devenue un capteur à sortie numérique. Afin de limiter le nombre de câbles dans la fusée, nous avons opté pour un bus série. Plusieurs solutions ont été envisagées (CAN, RS232) avant que nous options pour le bus I²C. Bien que ce soit un bus qui est sensé relier des composants sur la même carte, il a plusieurs avantages sur les autres. Premièrement, c'est un bus maître-esclave multipoint, c'est-à-dire que c'est un seul composant qui contrôle le bus et que les capteurs ne font que lui répondre ; de plus on peut y brancher plusieurs capteurs et les différencier par leur adresse. Le débit de 100 kbits/s est également convenable, à cette vitesse, des tests de plus de 2,50 m ont été faits, ce qui permet de traverser toute la fusée sans aucun problème (nous ne connaissions pas la taille de celle-ci tout au début de nos tests).

2. Le Bus I²C :

Le bus I²C se compose de 3 fils, l'horloge, la donnée et la masse pour le retour du courant. Ses niveaux sont 5V et 0V, les deux signaux sont munis de résistances de *pull-up* (de 10kΩ) et fonctionne en *wired-and*, c'est-à-dire qu'un 1 est représenté par le fait de laisser le bus flottant (haute impédance pour le composant) et un 0 est représenté par le fait que le composant court-circuite la ligne à la masse.

Le signal d'horloge est contrôlé par le maître, c'est lui qui définit le rythme de transmission des données. Le signal de données est modifié lorsque l'horloge est à 0 et lu lorsque l'horloge est à 1. Les adresses des différents capteurs sont les suivantes :

1001010X : Pitot
1001011X : Bog
1001001X : Accéléro.com et piézoélectrique
1001000X : Accéléromètre à jauges

3. Les convertisseurs :

Nous voulions pour le projet aXelle avoir des mesures sur 10 bits, nous pensions que c'était un bon compromis entre la difficulté de mise en œuvre et la résolution des mesures. Malheureusement, nous n'avons pas trouvé de convertisseurs convenables à cette résolution, nous nous sommes donc repliés sur des modèles à 12 bits ; de plus il est plus aisé d'obtenir 10 bits significatifs d'un convertisseur à 12 bits qu'à 10 (la mise en œuvre, obtenir le nombre maximal de bits significatifs est quasiment impossible). Ces convertisseurs, qui existent en deux versions - à 8 entrées (ADS7828) ou à une seule entrée (ADS7823), ils sont produits par Texas Instruments – disposent de deux pattes d'adressage, ce qui nous a permis d'en placer 4 sur le bus. Deux capteurs (Accéléro.com et piézo) partagent le même convertisseur à 8 voies, alors que tous les autres capteurs sont équipés d'un modèle à une seule entrée. Les convertisseurs sont alimentés en 5V et leur référence est de 5V ou 4,096V suivant le capteur. A cause de la structure du programme, notre vitesse d'échantillonnage n'est que de 48 acquisitions par seconde.

3.1. Accéléromètre à jauges :

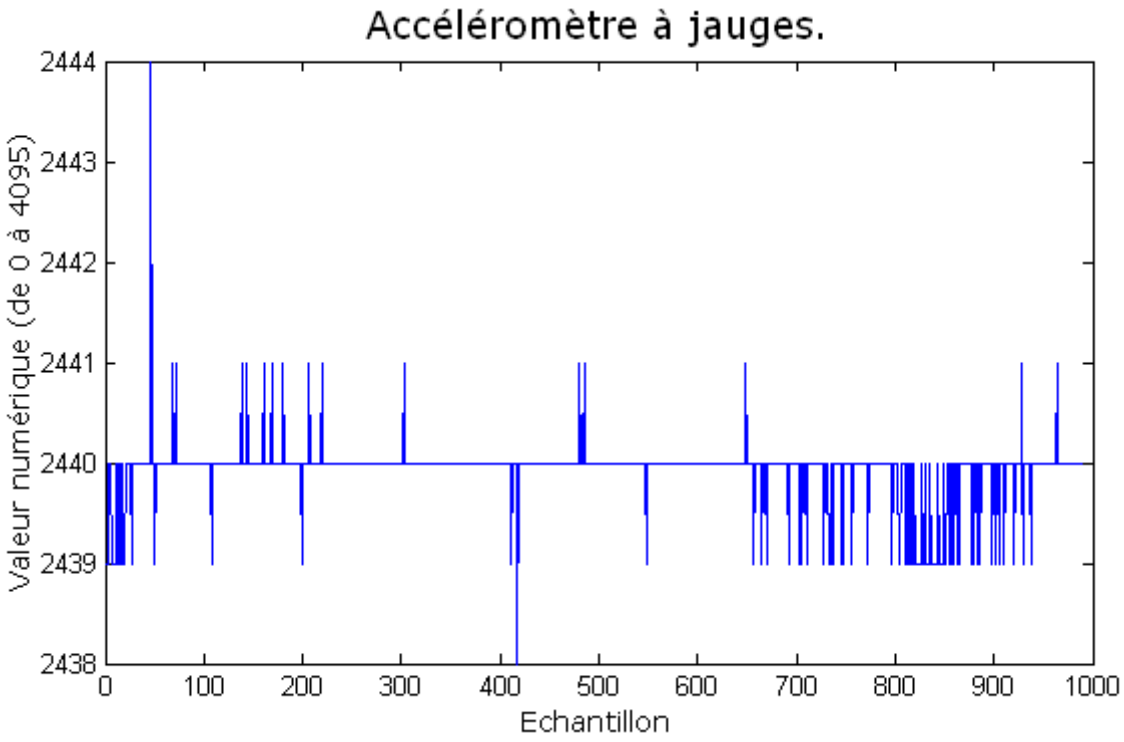


Figure 2 : Estimation du bruit sur la chaîne de mesure des jauges.

Bruit sur la chaîne de mesure des jauges :

Moyenne :	2439,9	Min :	2438
Variance :	0,1182	Max :	2444

Calcul de la résolution d'un capteur :

Pour ce faire, il faut tout d'abord réaliser un nombre suffisant de mesures (« un échantillon, c'est mille », dit toujours mon oncle professeur de mathématiques en lycée) d'une grandeur constante. Ensuite, construire un histogramme des valeurs : les valeurs seront proches de la grandeur physique réelle mesurée (celle-ci est statistiquement proche de la moyenne), avec une certaine dispersion. Ensuite, il suffit de prendre 95% des valeurs obtenues (pour avoir un intervalle de confiance de 95%, plus généralement X% pour avoir un intervalle de confiance de X%), et de celles-ci prendre le min et le max. La résolution de ce capteur est ainsi la distance entre le min et le max sur l'étendue de mesure. La résolution en bits est donnée par $\log_2\left(\frac{4096}{max - min}\right) = 12 - \log_2(max - min)$ (pour un capteur 12 bits).

$$\frac{2441 - 2439}{4096} = 0{,}048\% \Rightarrow 11 bits$$

Equation 3-1 : Résolution de la chaîne de mesures.

La résolution de ce capteur est de 11 bits, avec un intervalle de confiance de 99,8%, ce qui mieux que ce que nous attendions.

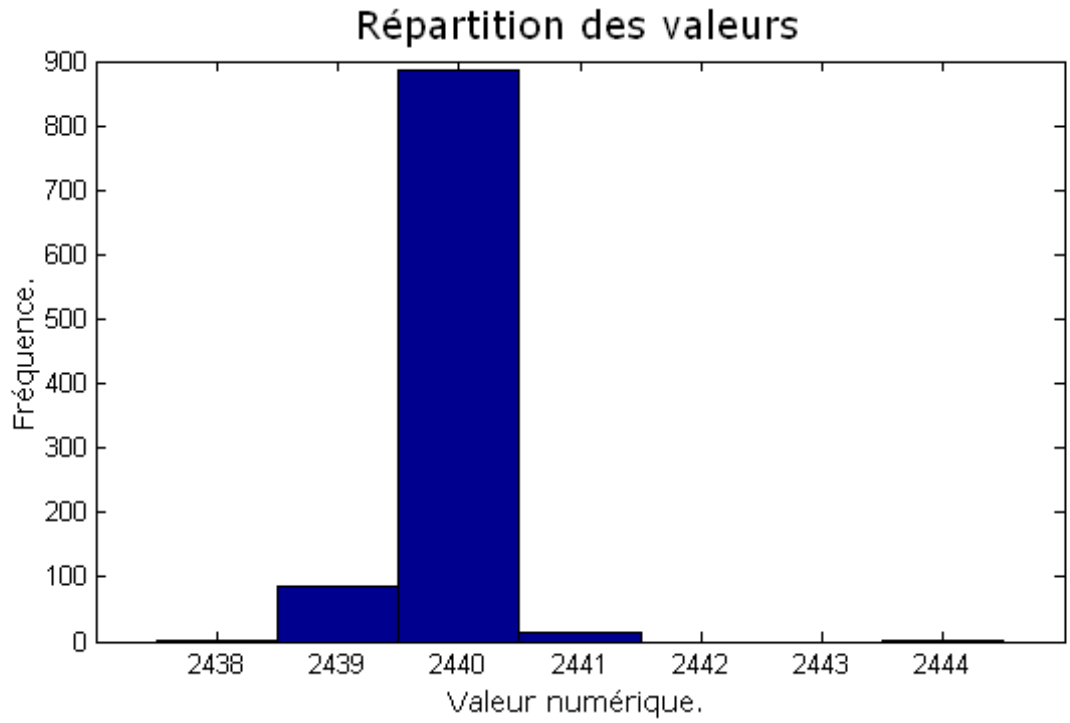


Figure 3 : Répartition des valeurs lors d'une acquisition constante des jauges.

3.2. Accéléromètre du commerce :

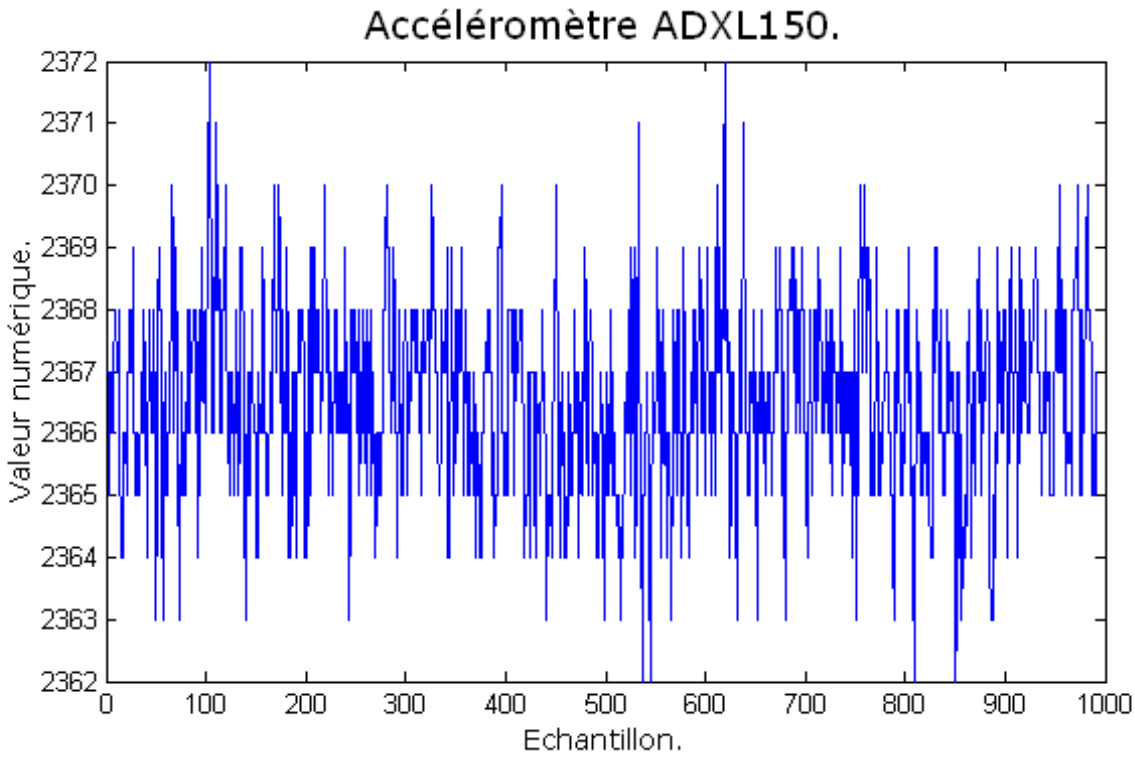


Figure 4 : Estimation du bruit sur la chaîne de l'accéléromètre du commerce.

Bruit sur la chaîne de mesure du capteur commercial :

Moyenne :	2366,6	Min :	2362
Variance :	2,406	Max :	2372

$$\frac{2369 - 2364}{4096} = 0{,}12\% \Rightarrow 9{,}68 bits$$

Equation 3-2 : Résolution de la chaîne de mesures.

La résolution de la chaîne de mesure de l'accéléromètre du commerce est d'un peu plus de 9,5 bits (avec un intervalle de confiance de 95,16%). Ceci est légèrement en dessous des attentes mais est totalement acceptable.

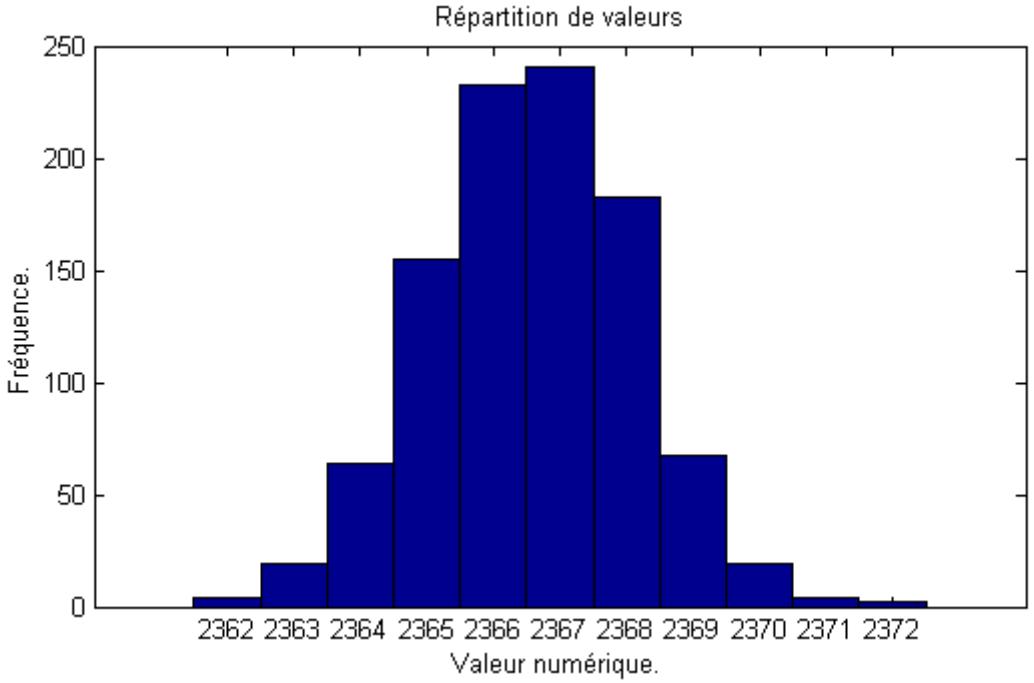


Figure 5 : Répartition des valeurs lors d'une acquisition constante de l'ADXL150.

3.3. Les autres capteurs :

Les mesures n'ont pas été effectuées pour le capteur piézoélectrique, ni pour le système à fil chaud. En effet, aucun étalonnage n'a été réalisé pour eux, du fait de sa difficulté de mise en œuvre pour le capteur à fil chaud ; et du fait de sa spécificité pour le capteur piézoélectrique les autres capteurs jouent en fait le rôle d'étalon pour eux (cf. : les documents sur les capteurs pour plus de détails).

Les fichiers d'étalonnage du Pitot n'ayant pas été conservés, il est impossible de faire le même travail avec ce capteur.

Néanmoins, on peut très raisonnablement présumer que ces autres chaînes de mesures ont une résolution voisine de celles obtenues ci-dessus.

4. Les filtres

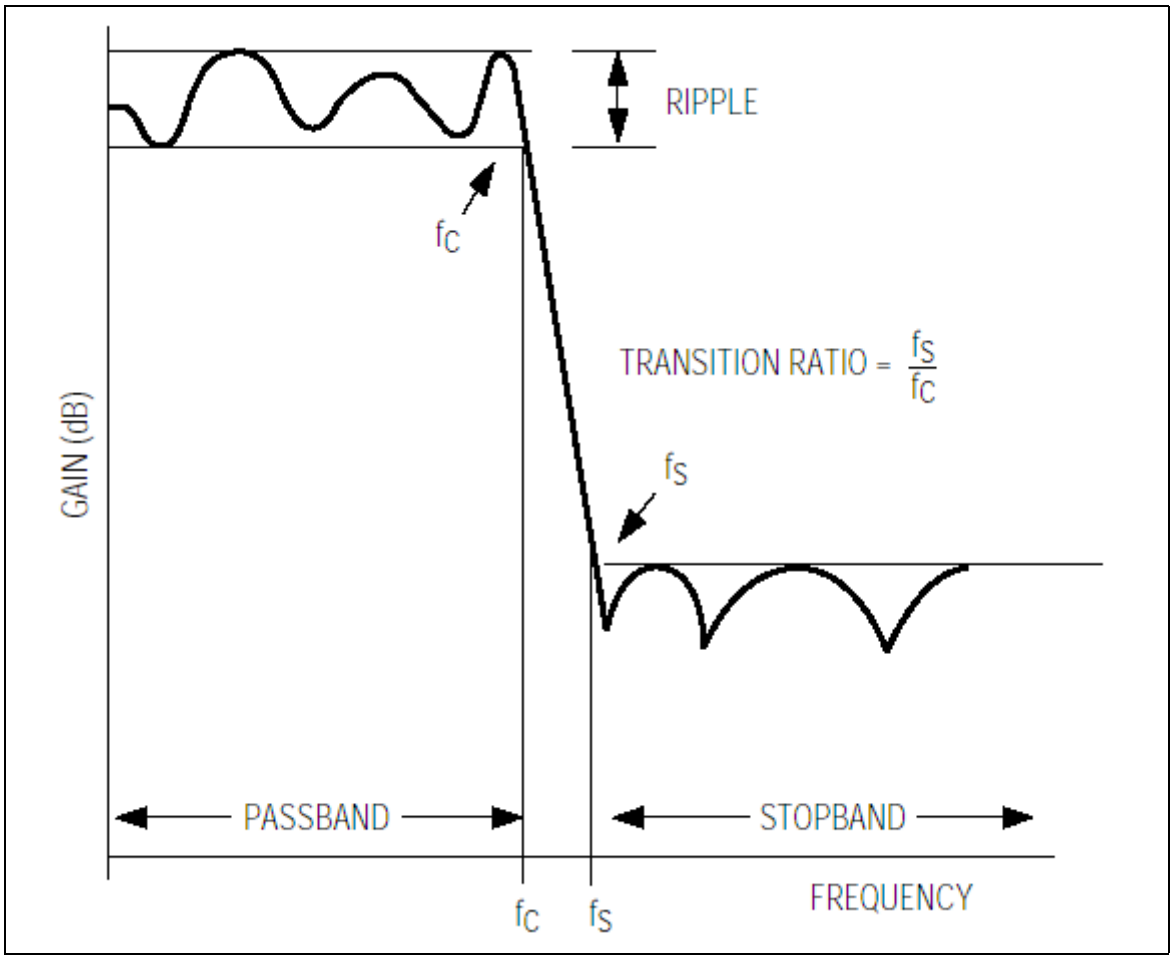


Figure 6 : Réponse fréquentielle théorique du filtre.

Pour satisfaire au théorème de Shannon, nous avons placé des filtres anti-repliement devant les échantillonneurs. Le rôle de ces filtres est bien sûr d'éviter les repliements de spectre. Pour garantir l'intégrité du signal numérisé, il faut théoriquement n'avoir aucune fréquence de valeur supérieure à la moitié de la fréquence d'échantillonnage. Pour garantir ceci, il faut que l'énergie contenue dans les énergies supérieures à cette fréquence limite soient inférieures à ce que peut détecter le convertisseur. C'est-à-dire que ce résidu d'énergie ne doit pas pouvoir faire commuter le bit le moins significatif (*LSB* ou *Least Significant Bit*). Ainsi, la fonction de transfert du filtre pour toutes les fréquences au-delà de la limite doit être une atténuation d'au moins $-20.\log_{10}(1/4096) \approx -72\,\text{dB}$. Ceci est bien vérifié si nous plaçons cette fréquence limite à f_S (cf. Figure **6** : Réponse fréquentielle théorique du filtre.) le gain étant alors de -82dB. De plus, en connaissant le rayon du filtre ($r = f_S/f_C = 1{,}5$), on peut établir que la fréquence de coupure du filtre f_C (à 3dB) doit se trouver au tiers de la fréquence d'échantillonnage ($f_{éch}/(2.r)$).

http://www.insa-lyon.fr/Associations/ClesFacil/

Carlos Correia Da Silva
cayottes@hotmail.com
Pierre Fayet
fayetpierre@aol.com
Pierre-Loïc Ropars
Pierre-loic.ropars@insa-lyon.fr

Projet : Axelle

19 juillet 2003

Version 1.3

07. Carte microcontrôleur & mémoire. Modulation FSK .

1. Description fonctionnelle

La fusée comporte différents capteurs pour nous permettre d'observer l'accélération pendant le vol. Les données acquises par ces capteurs sont ensuite traitées par le microcontrôleur pour être stocké à bord et émises par modulation de fréquence (modulation FSK : Frequency Shift Keying). Cette double sauvegarde permettra d'avoir une certaine sécurité des données qui seront dans les deux cas dépouillées, comparées, analysées et interprétées.

Les capteurs et les minuteries sont situés dans le corps de la fusée, tandis que le microcontrôleur, l'émetteur FSK et le récepteur GPS sont dans le module qui se sépare de la fusée à culmination. Il y a deux minuteries : une pour l'ouverture du module, et une pour l'ouverture de la case parachute.

1.1. Acquisition et traitement des données

Il est réalisé par un processeur *Microchip PIC 16F876A* qui acquière numériquement toutes les données par un bus I²C.

Le microcontrôleur est le maître du bus I²C et a l'initiative de la communication avec les esclaves :

- Capteurs : envoi d'un mot de commande, puis lecture des données sur 12 bits.
- Minuteries : lecture directe du registre.
- Mémoires : envoi d'un mot de commande donnant l'adresse d'accès aux données, puis lecture.

1.2. Mémorisation des données

On utilise quatre mémoires EEPROM. Le composant est un Microchip 24AA515 dont voici les grandes caractéristiques :

- Capacité utile : 64 ko.
- Bus : I²C.

1.3. Emission FSK

L'émission se fait par le port série TxD et est traitée par une autre partie de la carte.

1.4. Réception GPS

La réception se fait par le port série RxD et est générée par une autre carte indépendante.

1.5. Signaux ouverture parachute et décollage

Les signaux sont récupérées par lecture du registre de la minuterie.

	Valeur hexadécimale	Valeur binaire
Minuterie éteinte	0xC7	11000111
Minuterie module sous tension	0x80	10000000
Minuterie parachute sous tension	0x01	00000001
2 minuteries sous tension	0x47	01000111
Décollage	0x45	01000101
Ouverture module	0x05	00000101
Ouverture case parachute	0x01	00000001
Pas de réponse	0xFF	11111111

Figure 1 Interprétation des valeurs du registres des minuteries

Pour plus de détails, se reporter à la documentation de la minuterie.

2. Réalisation

La carte a été réalisée en un seul morceau pour des raisons de place dans la fusée, pour limiter le nombre de connecteurs et réduire les problèmes de compatibilité électromagnétique. Elle est composée des éléments suivants :

- Microcontrôleur et mémoires
- Alimentation microcontrôleur
- Modulation FSK
- Alimentation FSK
- Alimentation GPS

2.1. Schéma logique

Microcontrôleur & mémoires

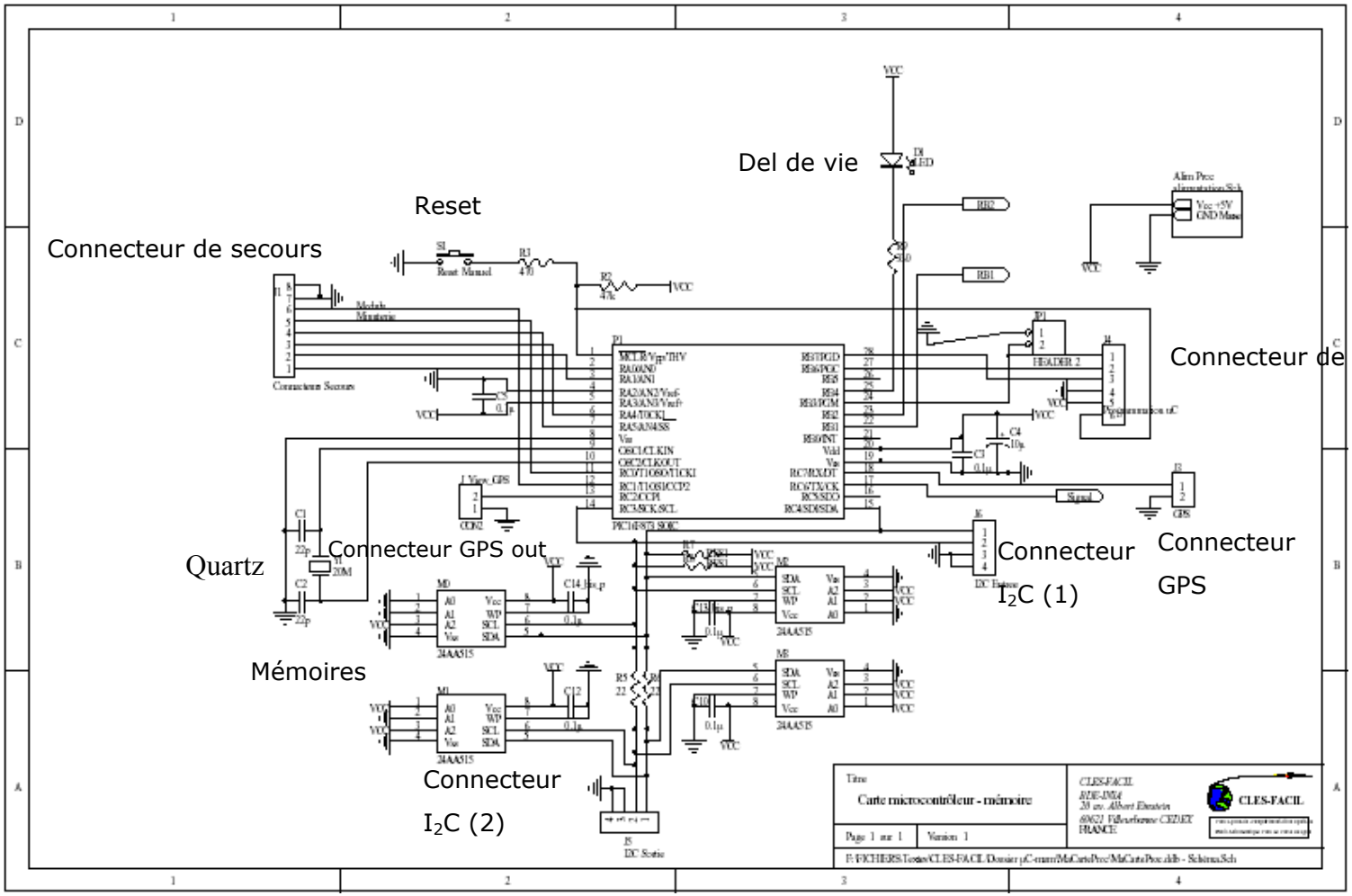


Figure 2 Microcontrôleur & mémoires

Microcontrôleur

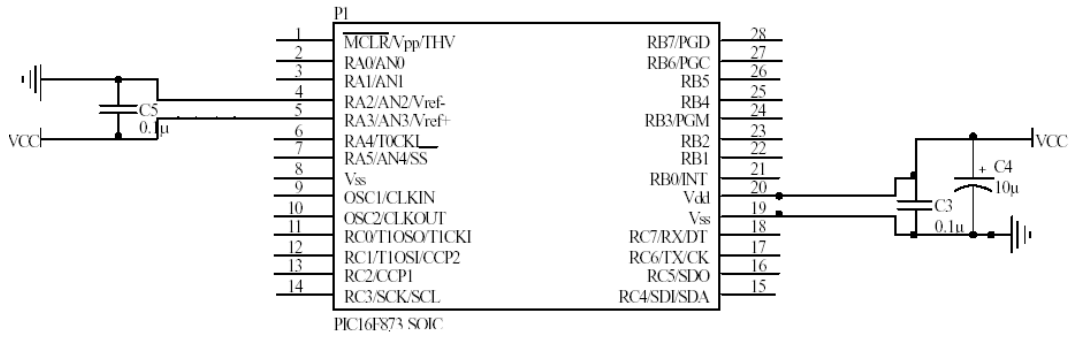


Figure 3 Microcontrôleur

Mémoires

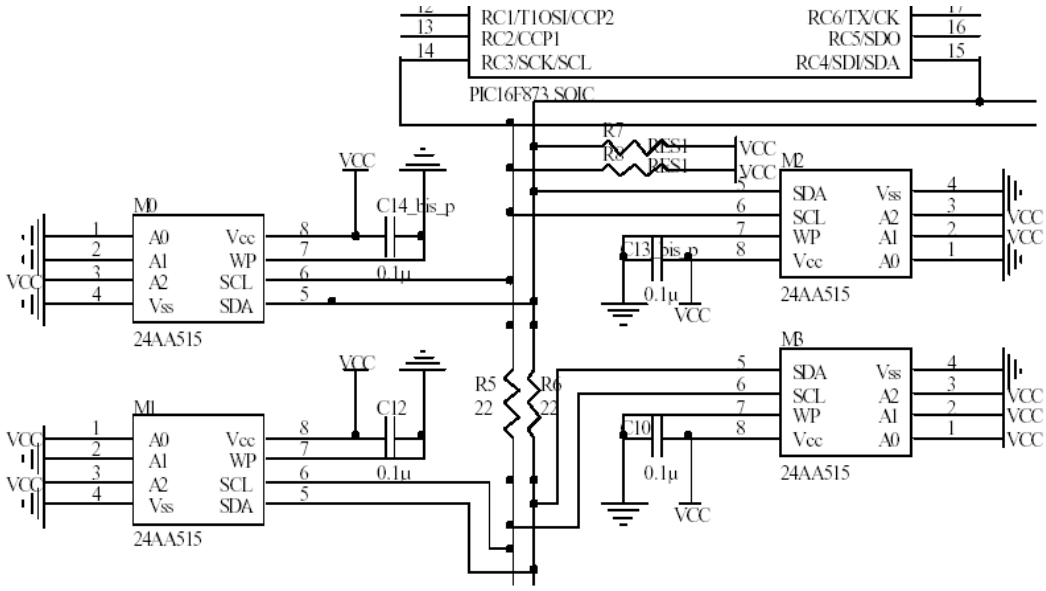


Figure 4 Mémoires

Reset

Le reset est déclenché par l'appui sur le bouton poussoir ou par lors de la programmation du microcontrôleur (par l'intermédiaire du connecteur de programmation).

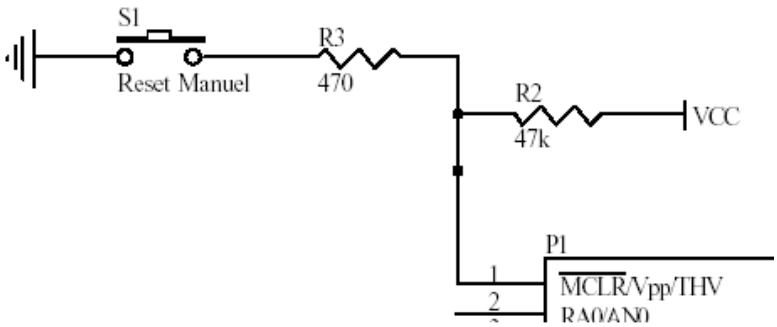


Figure 5 Reset

Quartz

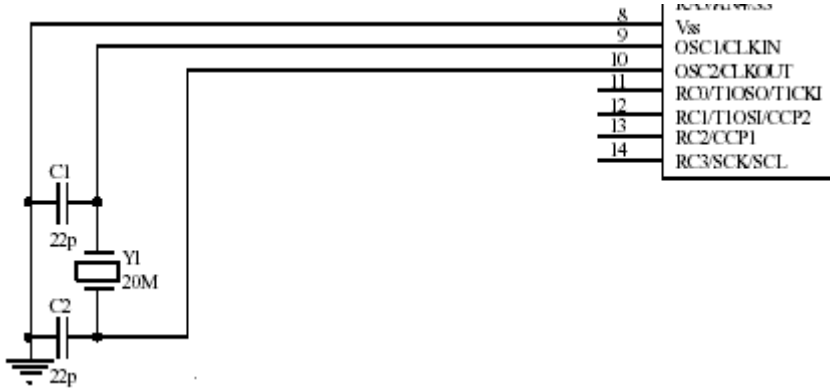


Figure 6 Quartz

DEL de vie

La DEL de vie n'a pour but que de signaler le fonctionnement du microcontrôleur.

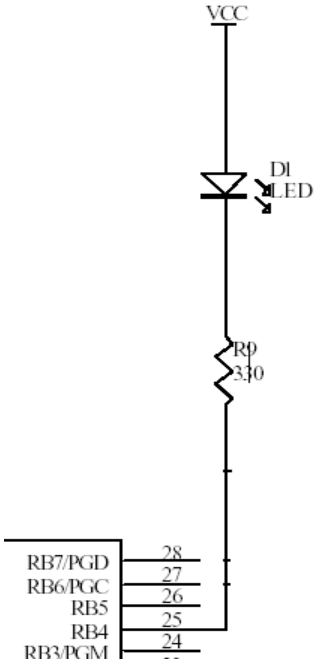


Figure 7 DEL de vie

Connecteur I²C

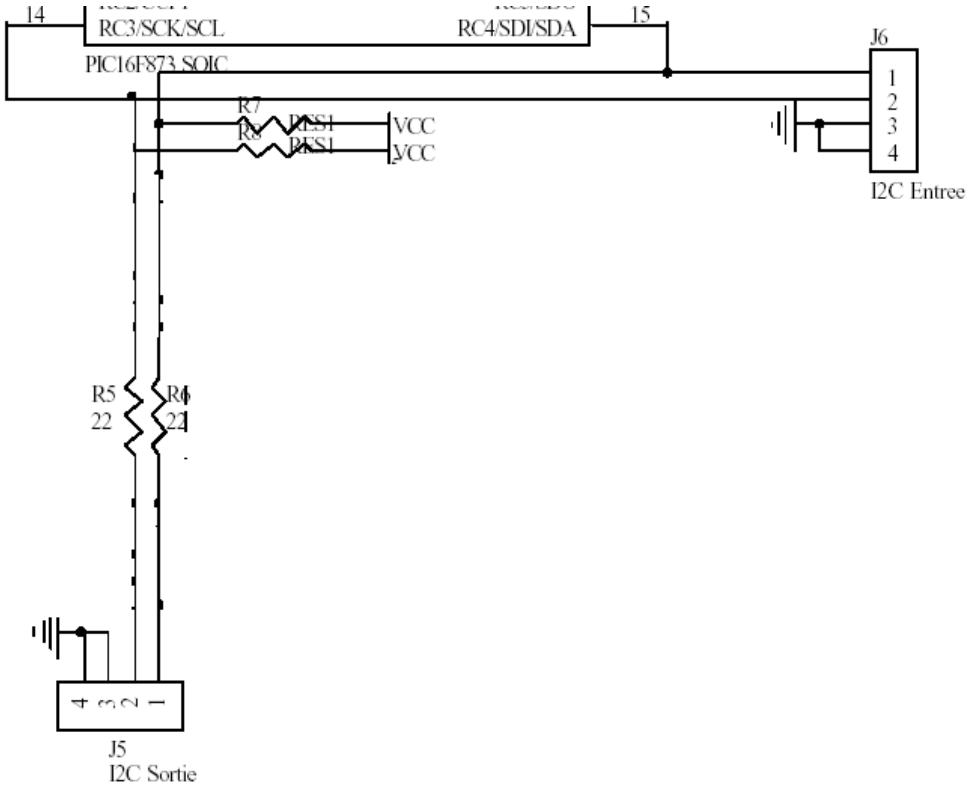


Figure 8 Connecteur I²C

Connecteur de secours

Le connecteur de secours ne devait servir que dans le cas où le bus I²C ne fonctionnerais pas entre les capteurs et le microcontrôleur. Il n'a jamais servi.

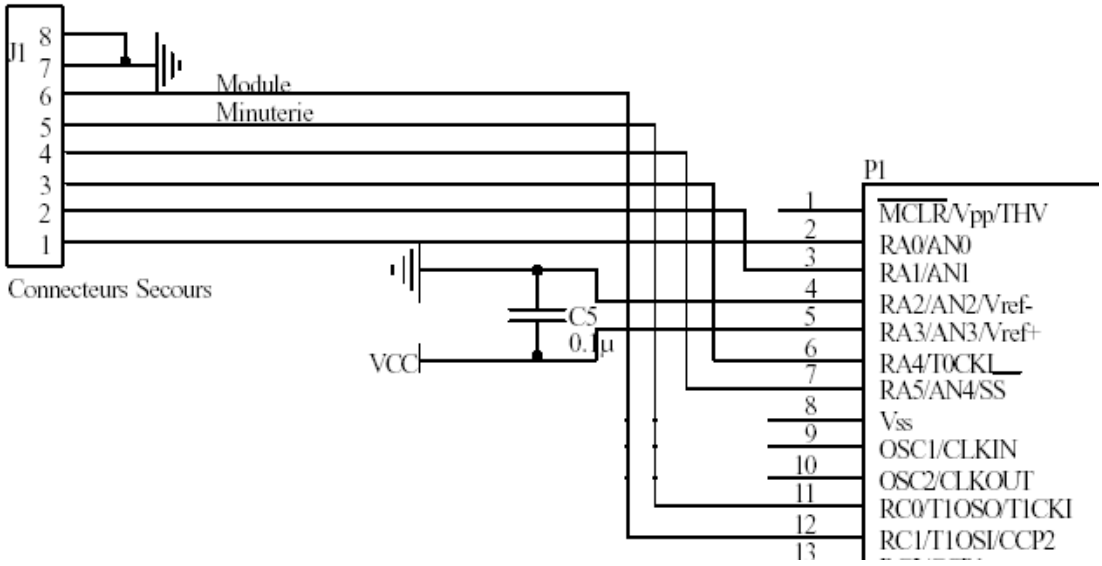


Figure 9 Connecteur de secours

Connecteur de programmation

Il est utiliser pour programmer le microcontrôleur.

Attention : bien vérifier l'alimentation du microcontrôleur lors de la programmation. Pour plus de détails, se référer à la carte de programmation du microcontrôleur. Cette carte est extérieure à la fusée.

Dans le cas du microcontrôleur PIC 16F876A, il faut l'alimenter pendant la programmation.

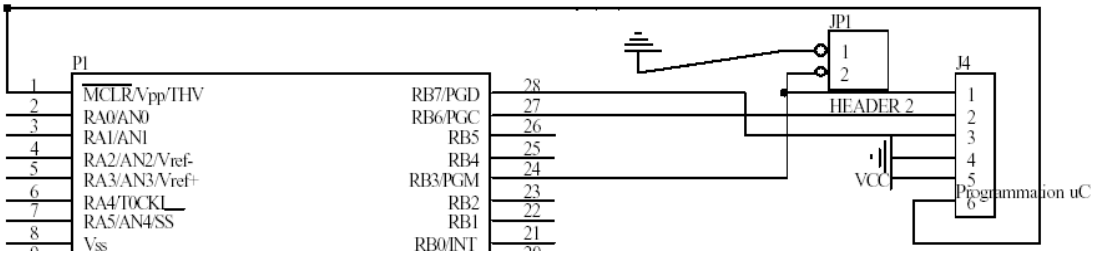


Figure 10 Connecteur de programmation

Connecteur GPS

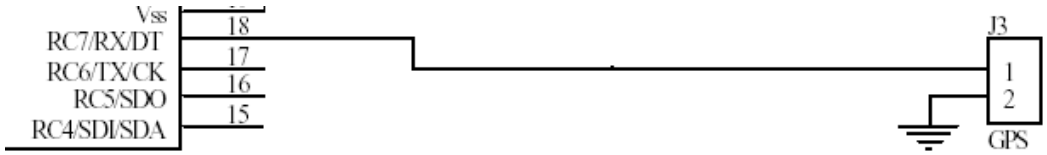


Figure 11 Connecteur GPS

Connecteur GPS out

Ce connecteur devait servir pour l'initialisation du GPS. La présence d'une pile de sauvegarde pour celui-ci rend l'utilisation de ce connecteur facultative, l'initialisation étant faite par une autre carte extérieure à la fusée.

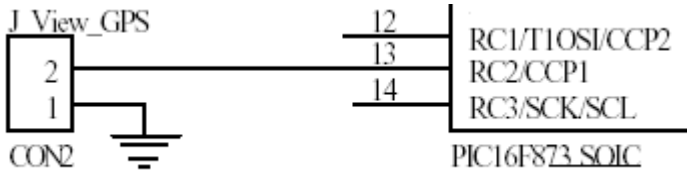


Figure 12 Connecteur GPS out

Signal FSK

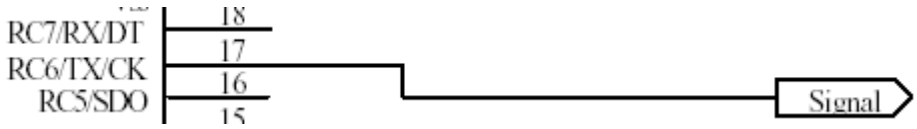


Figure 13 Signal FSK

Alimentation microcontrôleur

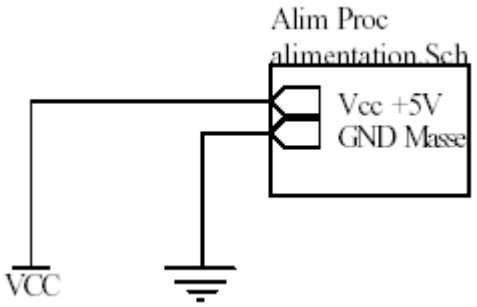


Figure 14 Alimentation microcontrôleur

Un problème de vidage de condensateur conduisit à l'alimentation pendant quelques minutes du microcontrôleur, après l'alimentation de celui-ci.

Cela causa de nombreux problème lorsque nous voulions reprogrammer le microcontrôleur, après le test d'un programme.

Pour compenser cela, il faut réaliser l'opération suivante qui devait scrupuleusement être effectuée selon ce protocole.

Après avoir arrêter toutes les alimentations de la carte microcontrôleur, mémoire et FSK, nous mettions le microcontrôleur en court-circuit pour décharger ce condensateur.

Contrôle Alimentation GPS & Alimentation FSK

- RB1 : Alimentation GPS
  - **Actif niveau haut : 1**
  - Inactif niveau bas : 0
- RB2 : Alimentation FSK
  - **Actif niveau bas : 0**
  - Inactif niveau haut : 1

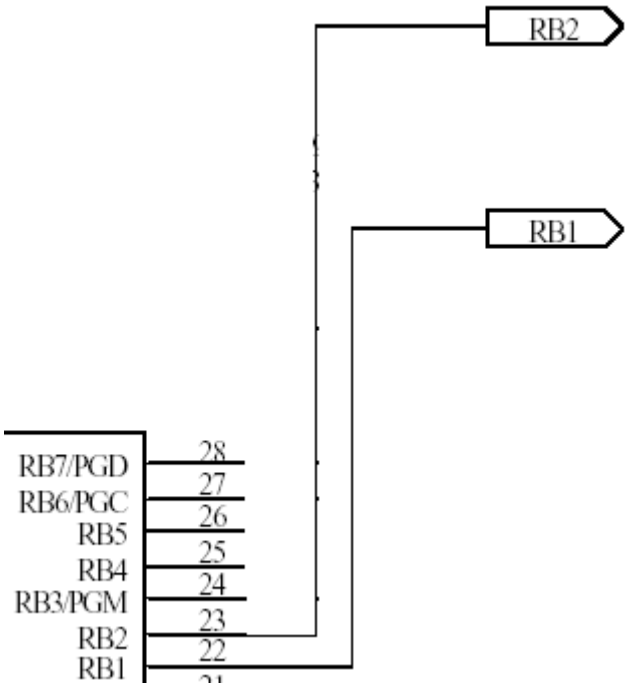


Figure 15 Contrôle Alimentation GPS & Alimentation FSK

Alimentation microcontrôleur

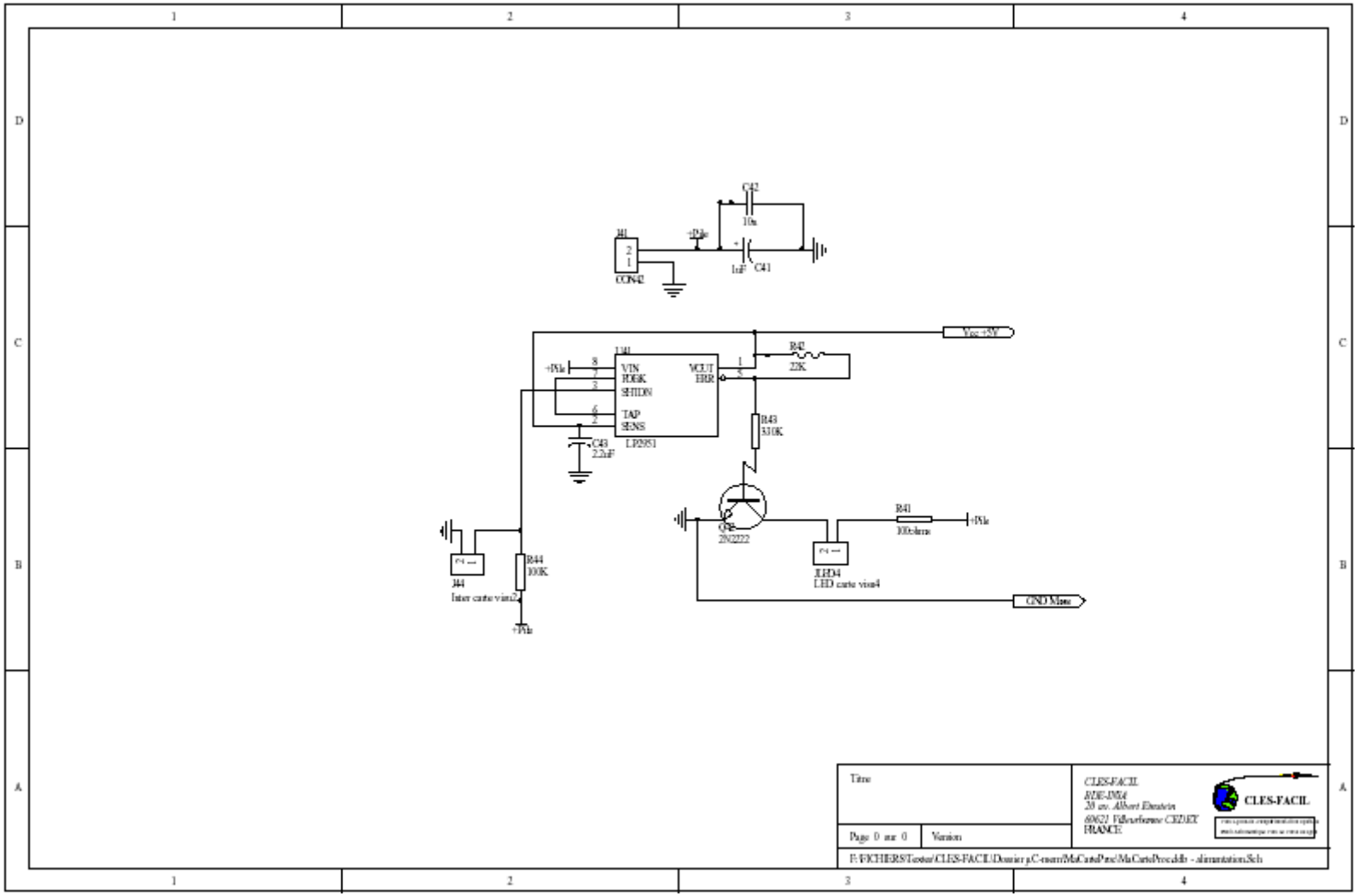


Figure 16 Alimentation microcontrôleur

Modulation FSK

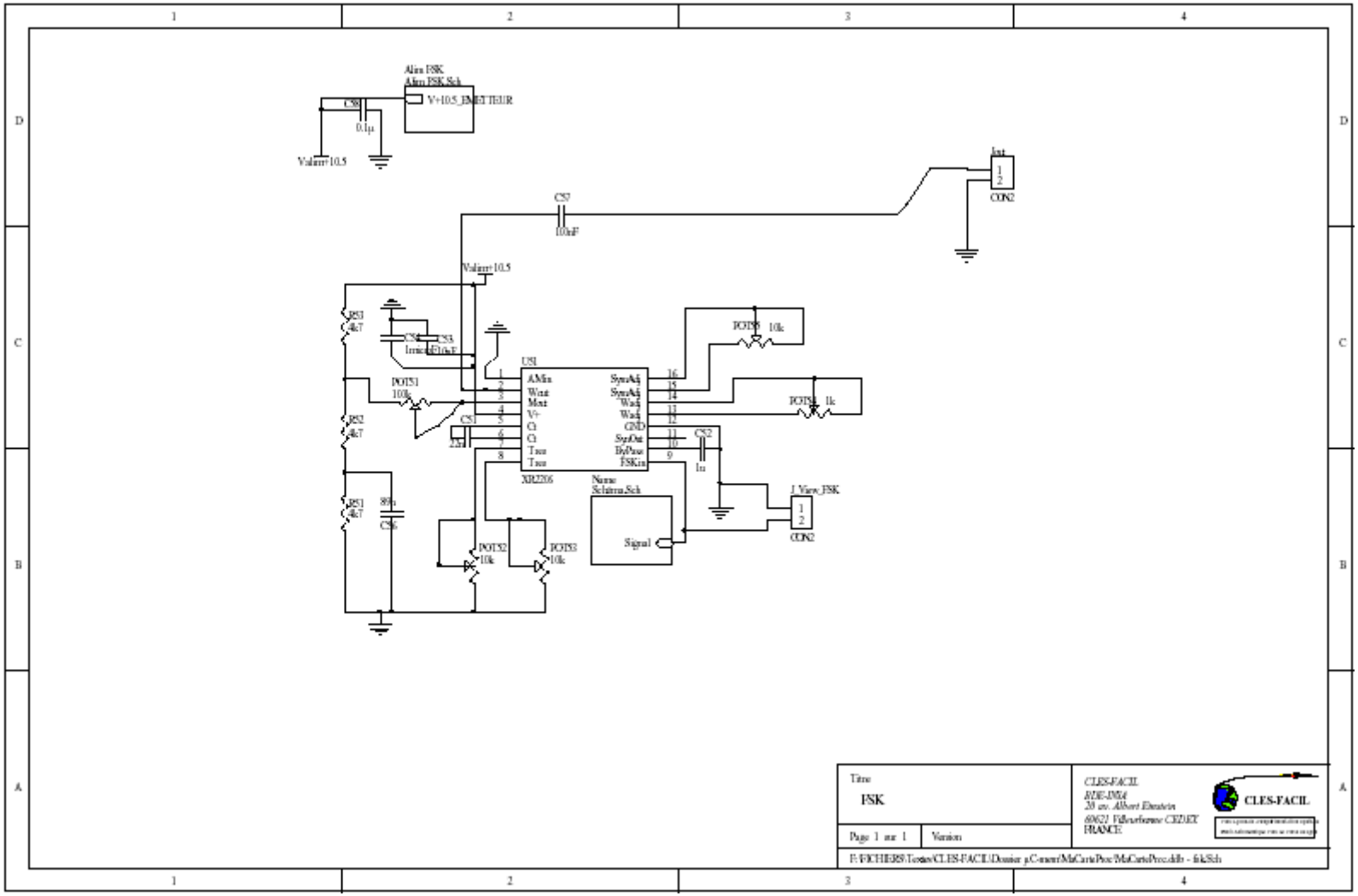


Figure 17 Modulation FSK

Alimentation modulation FSK

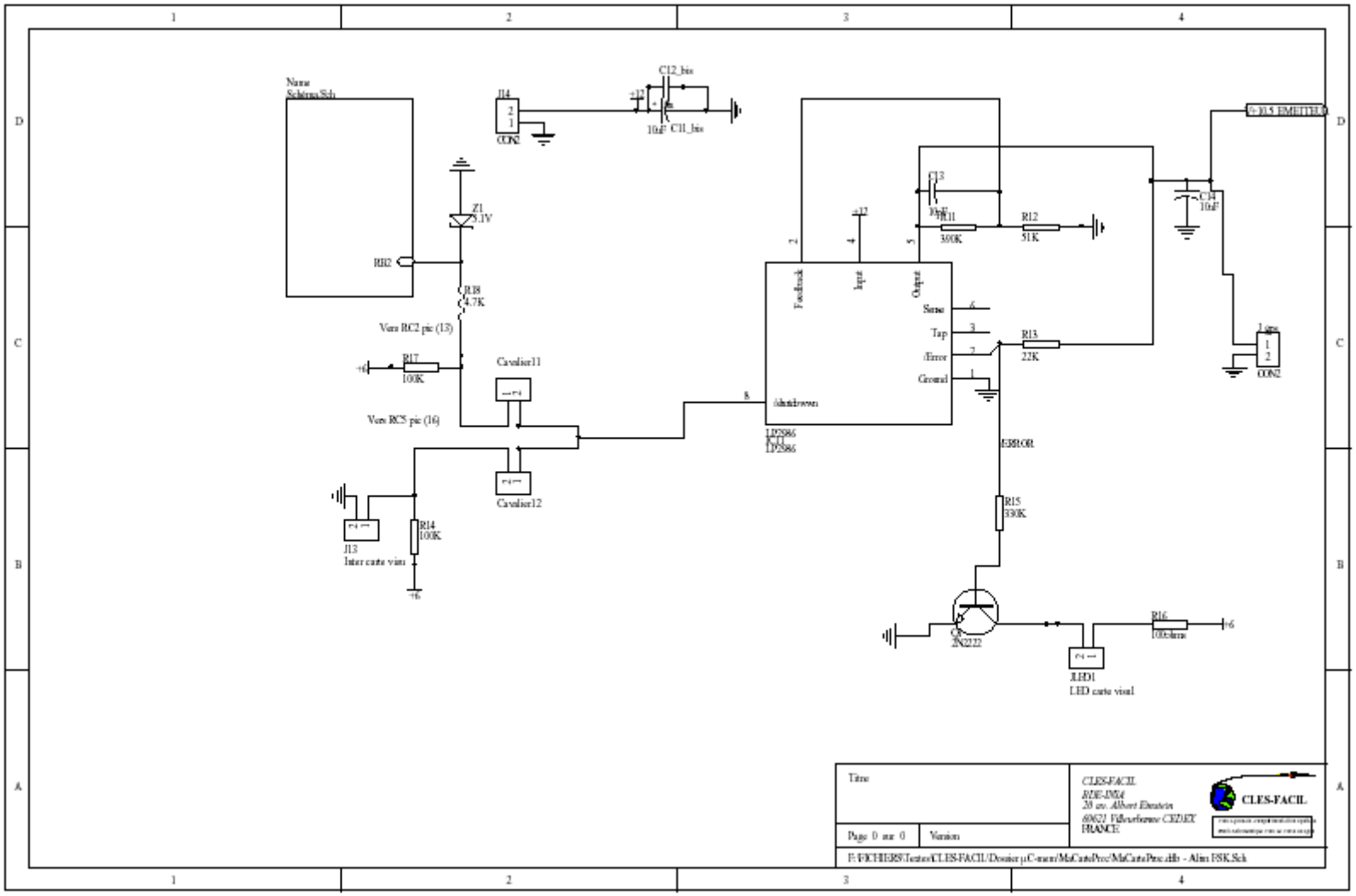


Figure 18 Alimentation modulation FSK

Alimentation GPS

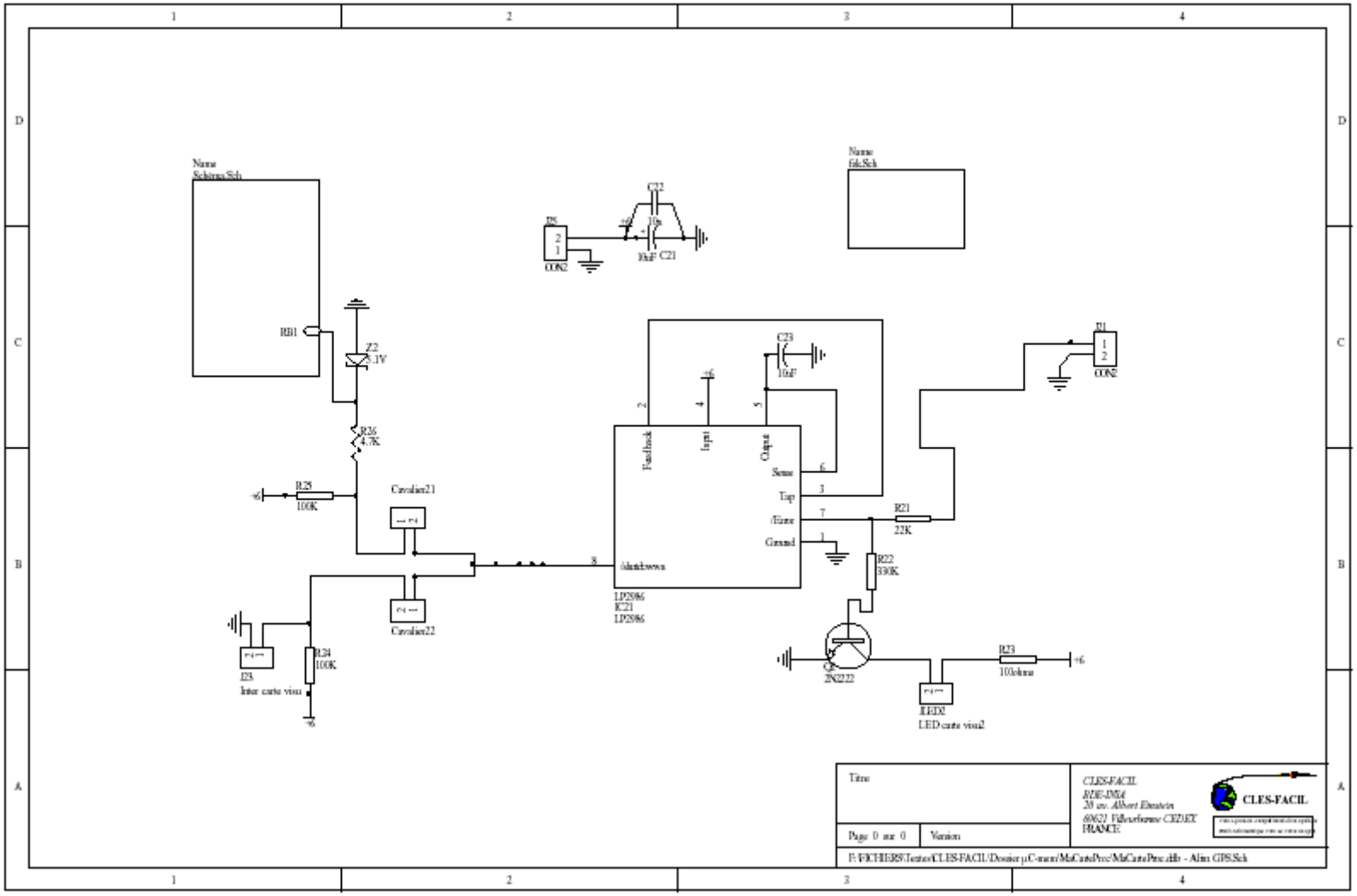


Figure 19 Alimentation GPS

2.2. Routage

Vue de dessus

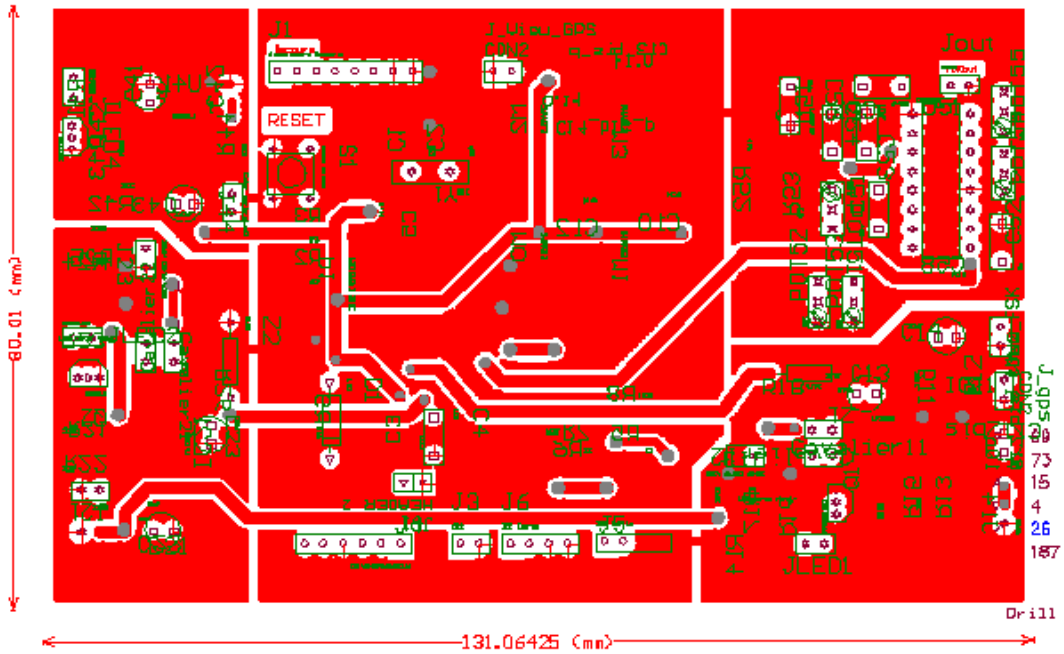


Figure 20 Carte vue de dessus

Vue de dessous

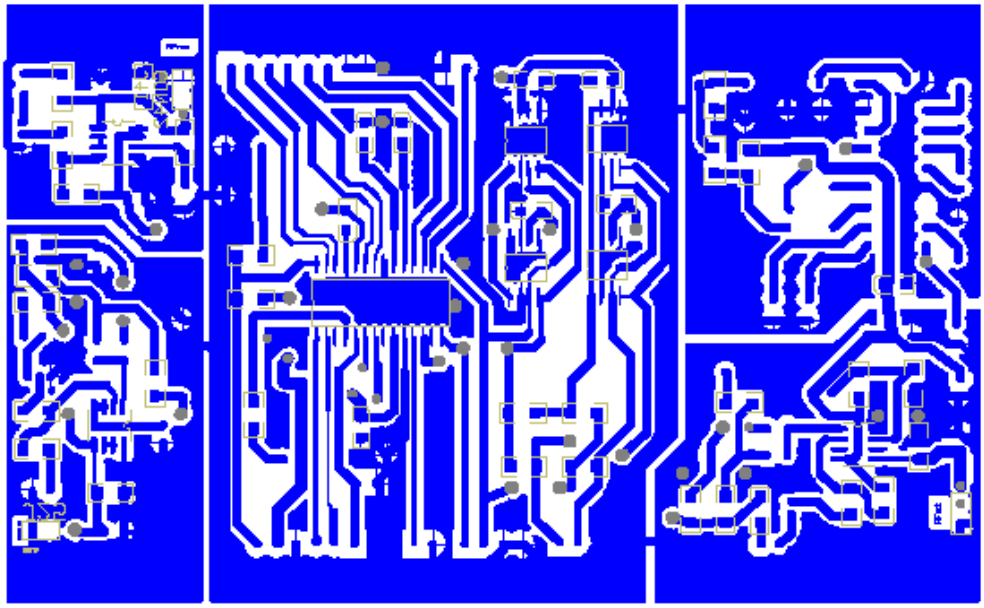


Figure 21 Carte vue de dessous

Connecteurs

Les connecteurs sont tous représentés selon la figure suivante.

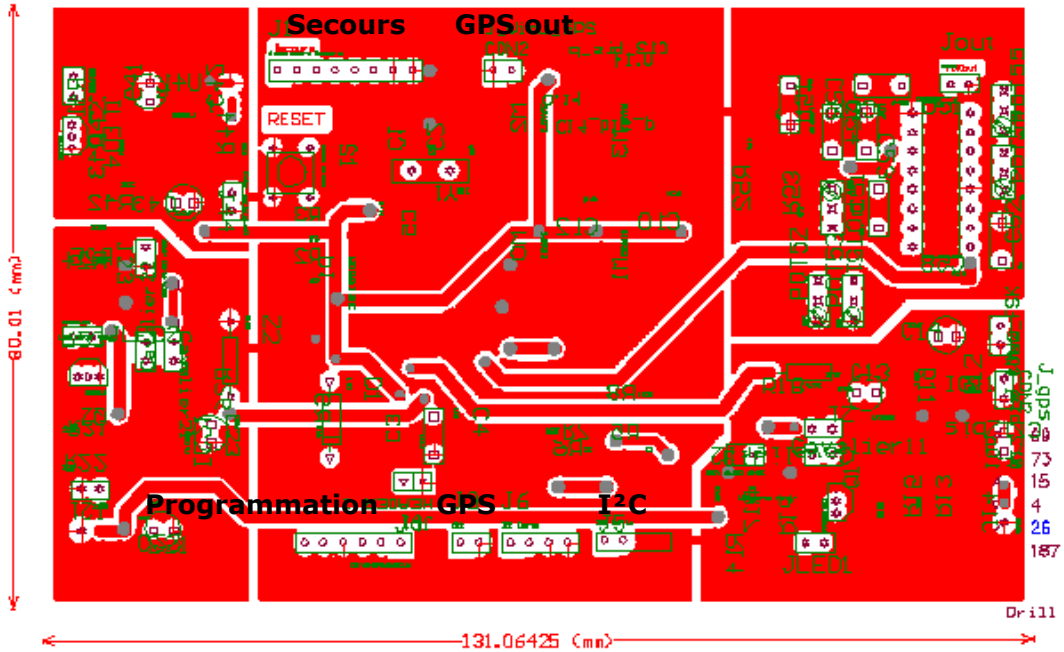


Figure 22 Connecteurs

Connecteur GPS

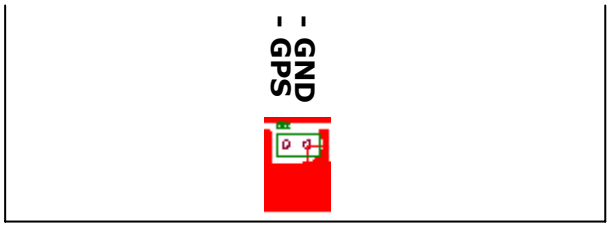


Figure 23 Connecteur GPS

Connecteur GPS out

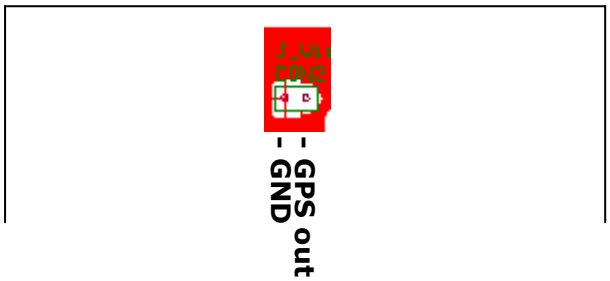


Figure 24 Connecteur GPS out

Connecteurs I²C

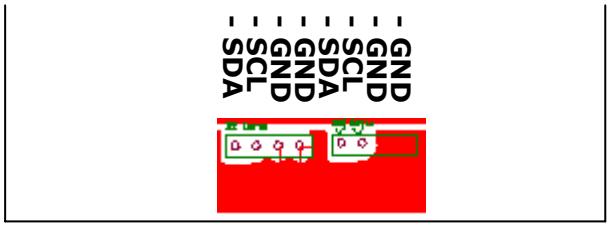


Figure 25 Connecteurs I²C

Connecteur de programmation

Figure 26 Connecteur de programmation

Connecteur de secours

Figure 27 Connecteur de secours

CLES-FACIL

http://www.insa-lyon.fr/Associations/ClesFacil/

Pierre-Loïc Ropars

Pierre-loic.ropars@insa-lyon.fr

Projet : Axelle

19 juillet 2003

Version 1.3

08. Programme embarqué.

1. Description fonctionnelle

La fusée comporte différents capteurs pour nous permettre d'observer l'accélération pendant le vol. Les données acquises par ces capteurs sont ensuite traitées par le microcontrôleur pour être stocké à bord et émises par modulation de fréquence (modulation FSK : Frequency Shift Keying). Cette double sauvegarde permettra d'avoir une certaine sécurité des données qui seront dans les deux cas dépouillées, comparées, analysées et interprétées.

Les capteurs et les minuteries sont situés dans le corps de la fusée, tandis que le microcontrôleur, l'émetteur FSK et le récepteur GPS sont dans le module qui se sépare de la fusée à culmination. Il y a deux minuteries : une pour l'ouverture du module, et une pour l'ouverture de la case parachute.

1.1. Acquisition et traitement des données

Il est réalisé par un processeur *Microchip PIC 16F876A* qui acquière numériquement toutes les données par un bus I²C.

Le microcontrôleur est le maître du bus I²C et a l'initiative de la communication avec les esclaves :

- Capteurs : envoi d'un mot de commande, puis lecture des données sur 12 bits.
- Minuteries : lecture directe du registre.
- Mémoires : envoi d'un mot de commande donnant l'adresse d'accès aux données, puis lecture.

1.2. Mémorisation des données

On utilise quatre mémoires EEPROM. Le composant est un Microchip 24AA515 dont voici les grandes caractéristiques :

- Capacité utile : 64 ko.
- Bus : I²C.

1.3. Emission FSK

L'émission se fait par le port série TxD et est traitée par une autre partie de la carte. Les protocoles de télémesure sont décrits dans la documentation Format télémesure Axelle 2003. Toute valeur égale à 0xff sera modifié à 0xfe pour éviter la confusion avec l'octet de synchronisation de début de trame.

Pour l'émission des position GPS pendant la phase de descente, la chaîne suivante en envoyé :
« *gps<ESPACE>hhmmss.ss,ddmm.mmmm,n,dddmm.mmmm,e,ss,a.a,z<CR><LF>* »
(voir Réception GPS pour la signification des champs en italique).

Un fanion est émis pour signaler le début de la trame.

La longueur de la trame est de :

- 1 octets pour le début de trame
- 4 octets pour « gps<ESPACE> »
- 43 octets pour la position
- 2 octets pour « <CR><LF> »

Soit 50 octets de 10 octets (1 bit de start et 1 bit de stop).

1.4. Réception GPS

Le GPS est un GT Plus Oncore de Motorola.

La réception se fait par le port série RxD et est générée par une autre carte indépendante. Les protocoles sont les mêmes que pour l'émission à l'exception du format de trame qui utilise le protocole NMEA.

Le microcontrôleur reçoit toutes les secondes la trame :
*$GPGGA,hhmmss.ss,ddmm.mmmm,n,dddmm.mmmm,e,q,ss,y.y,a.a,z,g.g,z,t.t,iiii*CC<CR><LF>*

Champ	Description	Valeurs
hhmmss.ss	**UTC of position fix**	
hh	*hours*	*00 .. 24*
mm	*minutes*	*00 .. 59*
ss.ss	*seconds*	*00.00 .. 59.99*
ddmm.mmmm,n	**latitude**	
dd	*degrees*	*00 .. 90*
mm.mmmm	*minutes*	*00.0000 .. 59.9999*
n	*direction*	*N = north* *S = south*
dddmm.mmmm,e	**longitude**	
ddd	*degrees*	*000 .. 180*
mm.mmmm	*minutes*	*00.0000 .. 59.9999*
e	*direction*	*E = east* *W = west*
q	**GPS status indicator**	**0 = GPS not available** **1 = GPS available** **2 = GPS differential fix**
ss	***number of sats being used***	***0 .. 12***
y.y	**HDOP**	
a.a,z	**antenna height**	
a.a	*height*	
z	*units*	*M = meters*
g.g,z	**geoidal separation**	
g.g	height	
z	units	M = meters
t.t	**age of differential data**	
iiii	**differential reference station ID**	**0000 .. 1023**
CC	**checksum**	

Figure 1 Positionnement GPS

Seuls les champs en italiques ont été exploités.

1.5. Signaux ouverture parachute et décollage

Les signaux sont récupérées par lecture du registre de la minuterie.

	Valeur hexadécimale	Valeur binaire
Minuterie éteinte	0xC7	11000111
Minuterie module sous tension	0x80	10000000
Minuterie parachute sous tension	0x01	00000001
2 minuteries sous tension	0x47	01000111
Décollage	0x45	01000101
Ouverture module	0x05	00000101
Ouverture case parachute	0x01	00000001
Pas de réponse	0xFF	11111111

Figure 2 Interprétation des valeurs du registres des minuteries

Pour plus de détails, se reporter à la documentation de la minuterie.

1.6. Adresses des périphériques sur le bus I²C

	Adresse	Mot de commande
Capteur accelero-jauges	0x90	0x00
Capteur accelero.com	0x92	0x84
Capteur Pitot	0x94	0x00
Capteur Bog	0x96	0x00
Capteur accéléro-piezo	0x92	0xf4
Minuterie	0x70	0x00
Mémoire 1	0xA0	N/A
Mémoire 2	0xA2	N/A
Mémoire 3	0xA4	N/A
Mémoire 4	0xA6	N/A

Figure 3 Adresses des périphériques I²C

Les adresses données sont les adresses I²C de lecture. Pour obtenir les adresses I²C en écriture, il faut ajouter 1 à l'adresse de lecture.

Le mot de commande de la mémoire

2. Réalisation

Le programme est exécuté sur la carte microcontrôleur, mémoire, et modulation FSK.

Le code est en c, compilé par pic-c.

2.1. Programme embarqué

Il est décomposé en phases :

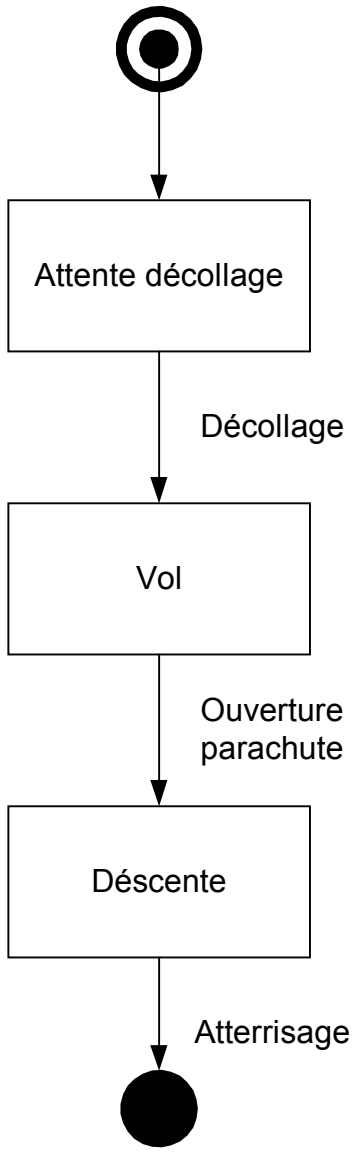


Figure 4 Diagramme d'état du programme

Initialisation

Le programme maintient l'alimentation du GPS arrêtée et met en marche la FSK.

Attente décollage

Le programme émet en continu l'état de la minuterie et le texte : « Axelle --- CLES-FACIL 2003 --- minuterie <%x> --- », où %x est l'état de la minuterie en hexadécimal.

Le passage de l'attente décollage au vol se fait par scrutation de la minuterie, lorsque celle-ci signal le décollage.

Vol

Le programme récupère les valeurs sur 12 bits de tous les capteurs en 6 séries, met les 12 bits de chaque capteurs en mémoire, et émet les 8 bits de poids fort de la 1^{e} série.

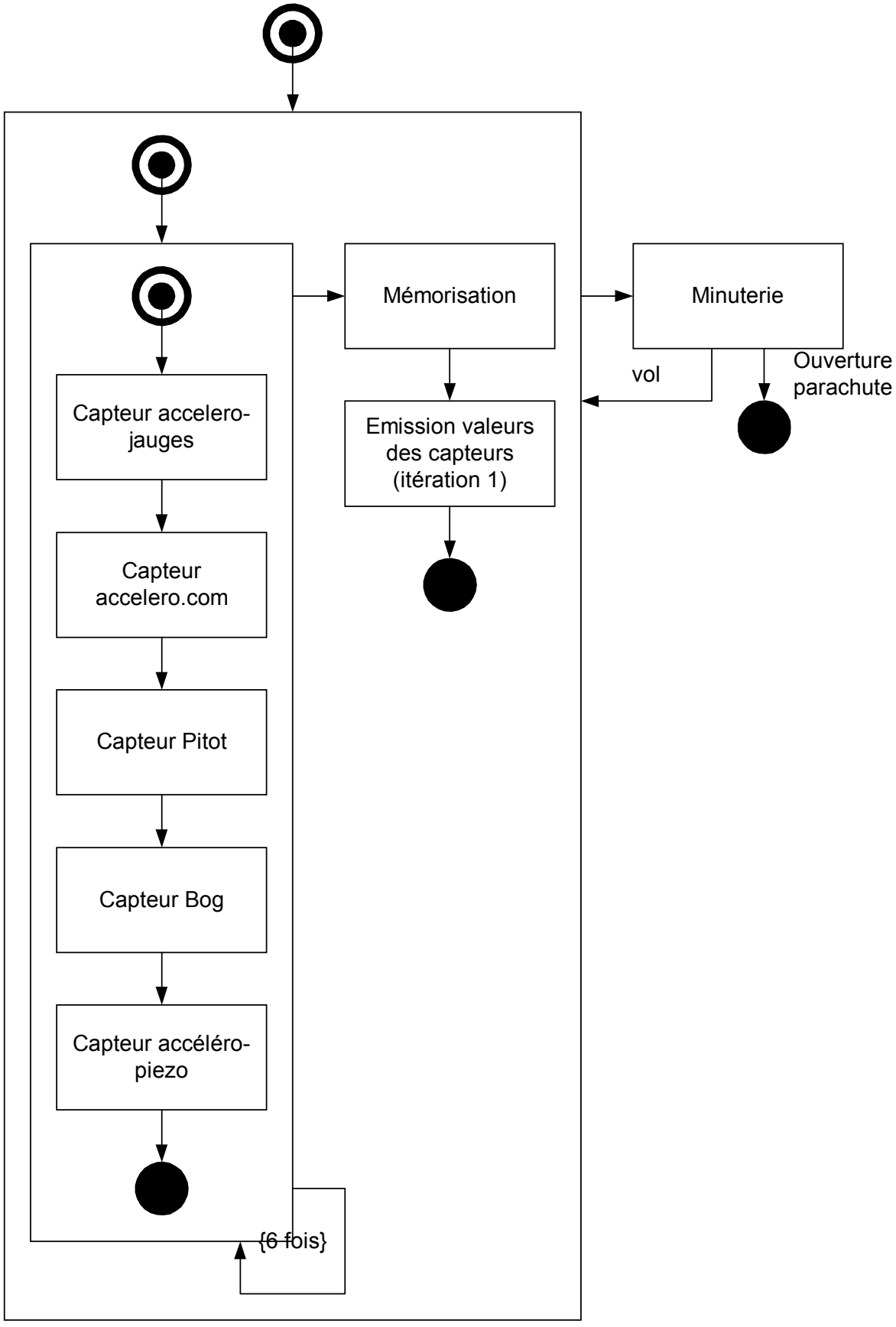


Figure 5 Diagramme d'état de la phase de vol

Le passage du vol à la descente se fait par scrutation de la minuterie, lorsque celle-ci signal l'ouverture du parachute.

Descente

Le programme récupère sa position par le récepteur GPS, ne conserve que les données significatives (voir Réception GPS), les émet par FSK et les met en mémoire.

Le passage de la descente à la fin de vol se fait un décompte de 20 minutes.

Fin de vol

Le programme arrête l'alimentation du GPS, l'alimentation de la FSK.
Le programme boucle ensuite sans fin jusqu'à l'arrêt de son alimentation.

Mémorisation des données

Pour éviter de perdre beaucoup de données consécutives en cas de défaillance d'une mémoire, l'écriture se fait consécutivement dans l'ordre suivant :

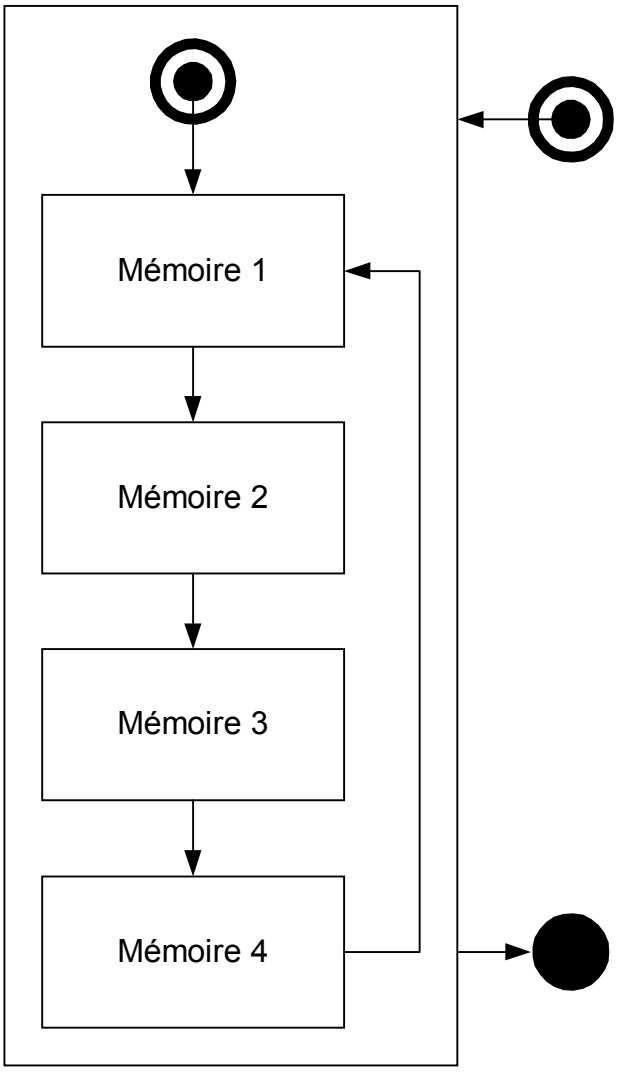


Figure 6 Ordre de mémorisation des pages

Transition capteurs GPS

4 pages ne comportant que des 1 en binaire sont écrites pour marquer la séparation entre les capteurs et le GPS.

Valeurs des capteurs

La minuterie est codée sur 8 bits.
Les valeurs des capteurs étaient codées sur 16 bits dont 12 significatifs.

1 octet	Minuterie	1 octet	Capteur accelero-jauges (itération 4) 2/2
2 octets	Capteur accelero-jauges (itération 1)	2 octets	Capteur accelero.com (itération 4)
2 octets	Capteur accelero.com (itération 1)	2 octets	Capteur Pitot (itération 4)
2 octets	Capteur Pitot (itération 1)	2 octets	Capteur Bog (itération 4)
2 octets	Capteur Bog (itération 1)	2 octets	Capteur accéléro-piezo (itération 4)
2 octets	Capteur accéléro-piezo (itération 2)	2 octets	Capteur accelero-jauges (itération 5)
2 octets	Capteur accelero-jauges (itération 2)	2 octets	Capteur accelero.com (itération 5)
2 octets	Capteur accelero.com (itération 2)	2 octets	Capteur Pitot (itération 5)
2 octets	Capteur Pitot (itération 2)	2 octets	Capteur Bog (itération 5)
2 octets	Capteur Bog (itération 2)	2 octets	Capteur accéléro-piezo (itération 5)
2 octets	Capteur accéléro-piezo (itération 2)	2 octets	Capteur accelero-jauges (itération 6)
2 octets	Capteur accelero-jauges (itération 3)	2 octets	Capteur accelero.com (itération 6)
2 octets	Capteur accelero.com (itération 3)	2 octets	Capteur Pitot (itération 6)
2 octets	Capteur Pitot (itération 3)	2 octets	Capteur Bog (itération 6)
2 octets	Capteur Bog (itération 3)	2 octets	Capteur accéléro-piezo (itération 6)
2 octets	Capteur accéléro-piezo (itération 3)	3 octets	N/A
1 octet	Capteur accelero-jauges (itération 4) 1/2		

1 page
64 octets

Figure 7 Enregistrement des capteurs en mémoire

Position GPS

La trame GPS émise par la FSK est enregistrée telle quelle en mémoire.
Chaque trame est enregistrée dans une page différente.

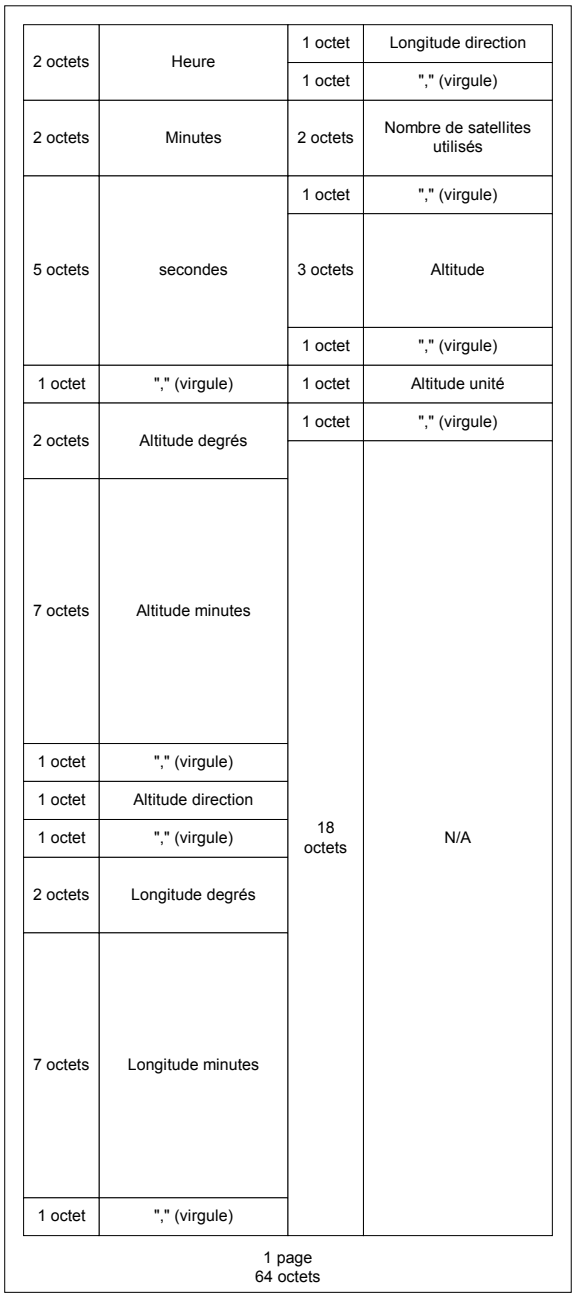


Figure 8 Enregistrement de la position GPS en mémoire

2.2. Extraction des données

L'extraction des données doit permettre de présenter les données pour quelles soient exploitables par un tableur ou par un logiciel de mathématique tel que matlab.

Il est nécessaire de reprogrammer le microcontrôleur pour pouvoir extraire les données.

Les données présentées ci-après correspondent à un cycle de 6 série d'acquisition des 5 capteurs.

Minuterie	Capteur accelero-jauges	Capteur accelero.com	Capteur Pitot	Capteur Bog	Capteur accéléro-piezo
Valeur	Valeur série 1	Valeur série 1	Valeur série 1	Valeur série 1	Valeur série 1

Minuterie	Capteur accelero-jauges	Capteur accelero.com	Capteur Pitot	Capteur Bog	Capteur accéléro-piezo
N/A	Valeur série 2	Valeur série 2	Valeur série 2	Valeur série 2	Valeur série 2
N/A	Valeur série 3	Valeur série 3	Valeur série 3	Valeur série 3	Valeur série 3
N/A	Valeur série 4	Valeur série 4	Valeur série 4	Valeur série 4	Valeur série 4
N/A	Valeur série 5	Valeur série 5	Valeur série 5	Valeur série 5	Valeur série 5
N/A	Valeur série 6	Valeur série 6	Valeur série 6	Valeur série 6	Valeur série 6

Figure 9 Format des données des capteurs après extraction

Chaque champ est séparé par une virgule « , ».

Les données de positions GPS ont le même format que les positions émises par FSK (voir réception GPS, et mémorisation des données, position GPS) : *gps hhmmss.ss,ddmm.mmmm,n,dddmm.mmmm,e,ss,a.a,z<CR><LF>*.

Pour l'extraction des données GPS, nous avons récupéré les valeurs comme s'il s'agissait de capteurs et une conversion a été réalisée à l'aide d'un script php pour obtenir des données « lisibles ». Cela nous a éviter de reprogrammer le microcontrôleur pour réaliser une nouvelle extraction.

2.3. Protocole I²C

Mémoire 24AA515

L'adressage de la mémoire est particulier. En effet dans l'adresse de la mémoire, il faut renseigner la page de mémoire qui sera lue ou écrite : bit B0.

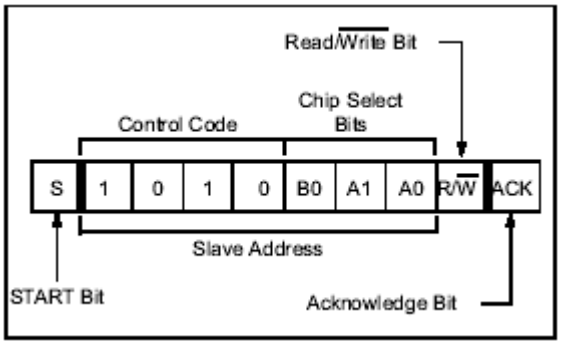


Figure 10 Format du bit de contrôle

L'adressage en mémoire se fait en donnant après l'adresse I²C de la mémoire, l'adresse de la zone mémoire qui sera lue ou écrite. Elle se fait sur 15 bits, le 15^{e} bit de poids le plus fort étant B0.

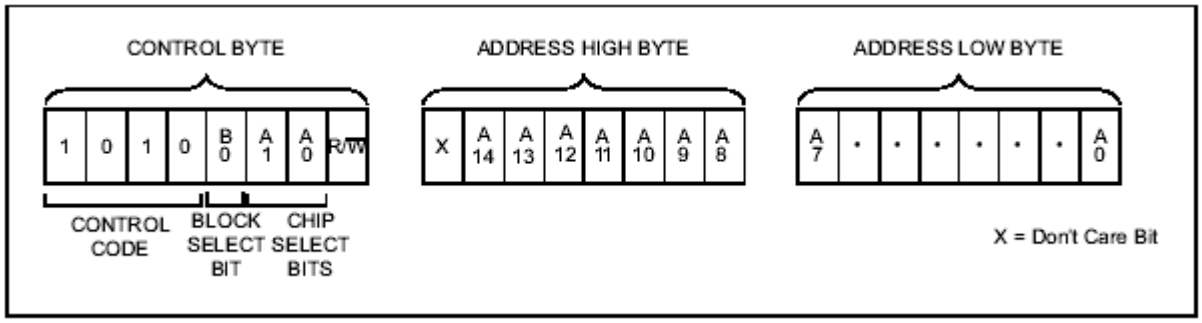


Figure 11 Affectation des bits

L'écriture d'un octet se fait de la manière suivante :

- Ecriture de start
- Ecriture de l'adresse d'écriture de la mémoire
- Ecriture de l'adresse mémoire de poids fort
- Ecriture de l'adresse mémoire de poids faible
- Ecriture de l'octet
- Ecriture de stop

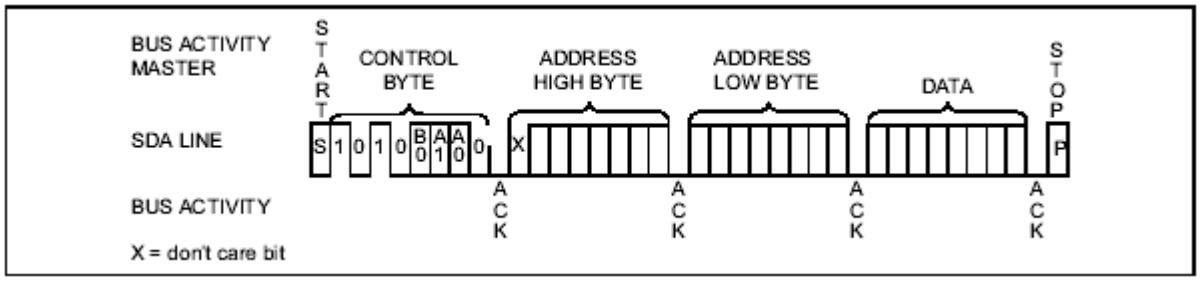


Figure 12 Ecriture d'un octet

L'écriture d'une page (1 à 64 octets) se fait de la manière suivante :

- Ecriture de start
- Ecriture de l'adresse d'écriture de la mémoire
- Ecriture de l'adresse mémoire de poids fort
- Ecriture de l'adresse mémoire de poids faible
- Ecriture des octets
- Ecriture de stop

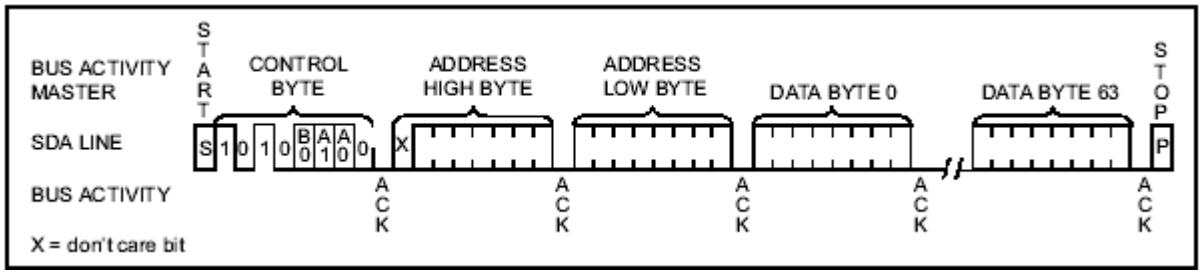


Figure 13 Ecriture d'une page

Une page commence à une adresse multiple de 64 (ex. : 0, 64, 128…).
L'écriture d'une plage implique l'utilisation de toute la page de 64 octets, même si tous les octets ne sont pas écrits. Il faut ensuite passer à la page suivante.

La lecture d'un octet se fait de la manière suivante :

- Ecriture de start
- Ecriture de l'adresse d'écriture de la mémoire
- Ecriture de l'adresse mémoire de poids fort
- Ecriture de l'adresse mémoire de poids faible
- Ecriture de l'adresse de lecture de la mémoire
- Lecture de l'octet
- Ecriture de stop

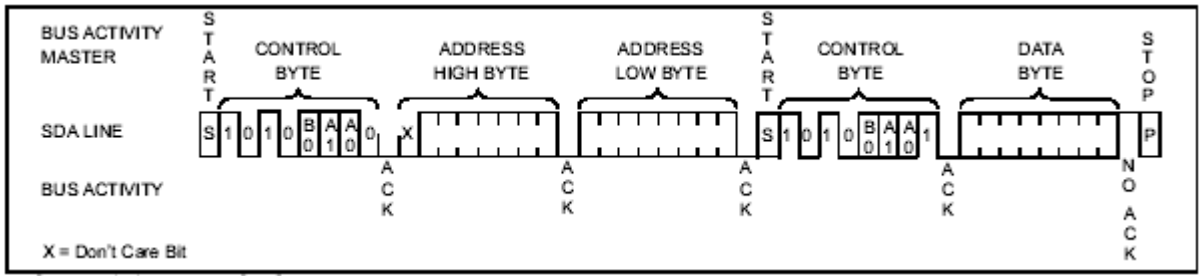


Figure 14 Lecture directe d'un octet

La lecture d'une page (1 à 64 octets) se fait de la manière suivante :

- Ecriture de start
- Ecriture de l'adresse d'écriture de la mémoire
- Ecriture de l'adresse mémoire de poids fort
- Ecriture de l'adresse mémoire de poids faible
- Ecriture de l'adresse de lecture de la mémoire
- Lecture des octets
- Ecriture de stop

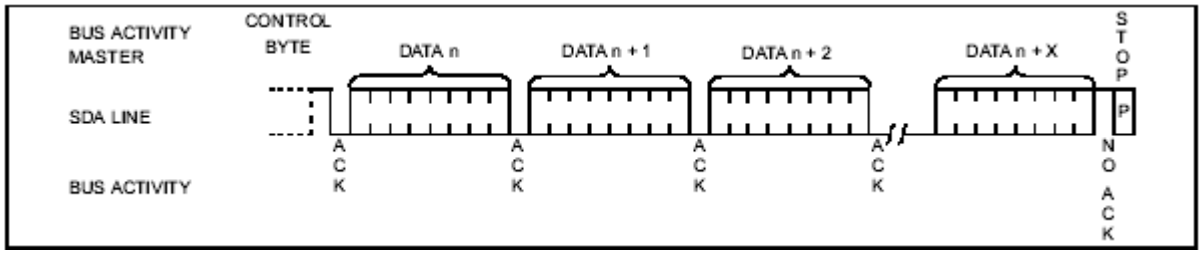


Figure 15 Lecture séquentielle

Capteur

La lecture de chaque capteur (la minuterie doit être considérée comme un capteur) suit le protocole suivant :

- Ecriture de start
- Ecriture de l'adresse d'écriture du capteur
- Ecriture du mot de commande
- Start
- Ecriture de l'adresse de lecture du capteur
- Lecture des bits de poids forts
- Lecture des bits de poids faibles
- Ecriture de stop
- Mise en forme de la valeur (pas d'utilisation du bus I²C)

3. Problèmes rencontrés

L'architecture du programme ne posa pas de problème.

Par contre, la mise en mémoire, et surtout l'extraction posèrent de nombreux problèmes.

L'ordre d'écriture de l'adresse d'écriture en mémoire à été inversé.

De plus, l'inversions des bits de poids forts et de poids faibles en mémoire posait des problèmes à l'extraction.

De nombreux problèmes dans le codage faisaientt que nous ne savions pas si les valeurs acquises étaient justes ou non.

Ces problèmes ont été résolu pendant la campagne.

Le programme d'extraction bouclait sans fin, l'arrêt de l'alimentation au bout de 2 heures montra que les données avaient été extraites plusieurs fois. Il fallu ensuite éliminer les doublons.

L'échantillonnage des valeurs n'était pas régulier. En effet, non avions un cycle de 6 acquisitions des 5 capteurs, une mémorisation de toutes ces valeurs, et l'émission par FSK des valeurs de la 1^{e} acquisition. Or, les valeurs en mémoire n'étaient pas réparties de façon homogène dans le temps, mais par salves.

Pour résoudre ce problème, il aurait fallu :

- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre le caractère de synchronisation de la trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre les 8 premiers bits du numéro de trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre les 8 seconds bits du numéro de trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre la valeur de la minuterie (8 bits) de la trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre la valeur du 1er capteurs (8 bits) de la trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre la valeur du 2^{e} capteurs (8 bits) de la trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre la valeur du 3^{e} capteurs (8 bits) de la trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre la valeur du 4^{e} capteurs (8 bits) de la trame FSK
- faire l'acquisition des 5 capteurs
- mettre en mémoire ces valeurs
- émettre la valeur du 5^{e} capteurs (8 bits) de la trame FSK

Ce problème n'a pas été résolu lors de la campagne en raison du manque de temps pour valider les nouveaux programmes.

L'émission FSK et la réception GPS sur le même port série USART à conduit à utiliser les paramètres du moins disant, c'est à dire 4800 bauds alors que la FSK pouvait émettre à 9600 bauds, et 1 bit de stop alors que l'utilisation de 2 bits nous aurait permis d'utiliser le protocole SNR utilisé par Planète Sciences pour la télémesure.

4. Conclusion

Tout ce qui n'a pas été suffisamment testé avant la campagne n'a pas marché.

C'est pourquoi, il faut être plus rigoureux dans les tests avant la campagne, lorsqu'on a encore du temps, et aussi bien réfléchir à l'échantillonnage. Ce dernier point étant une conséquence du précédent.

Cependant, l'architecture en bus I²C à donner toute satisfaction et est à reconduire pour les projets suivants.

5. *Annexes*

5.1. Programme embarqué

axelle.h

```
/*****************************************************************************/
/* axelle.h                                                                  */
/*                                                                           */
/* Description : programme de test de la fusee,  sous l'environnement de     */
/*    developpement MPLAB IDE avec le compilateur c pic-c.                   */
/* Auteur : Pierre-Loic ROPARS                                               */
/*          Pierre-Loic.ROPARS@insa-lyon.fr                                  */
/* 09/11/2002 Creation                                                       */
/* 22/03/2003 Modification : ajout fonctions programme fusee                 */
/*****************************************************************************/

/* Declaration du processeur */

#include <16f876.h>          /* Microcontroleur */

#fuses HS,NOWDT,NOPROTECT
//#fuses XT,NOWDT,NOPROTECT
//#use delay(clock=4000000)   /* Horloge du microcontroleur */
#use delay(clock=20000000)    /* Horloge du microcontroleur */

/* Inclusions */
#include <string.h>

#define FSK    0x01
#define GPS    0x02

/*#define LIAISON_GPS rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS, STREAM=GPS)*/
/*#define LIAISON_GPS rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS, STREAM=GPS)*/
#ifdef PC      /* PC */
/*      #define LIAISON_FSK   rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS,
STREAM=FSK)*/
#else          /* FSK */
       #define LIAISON_FSK   rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS,
STREAM=FSK)
/*      #define LIAISON_FSK   rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7, BITS=9, ERRORS,
STREAM=FSK)*/
/*      #define LIAISON_FSK   rs232(baud=9600, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS,
STREAM=FSK)*/
#endif

/* Liaison GPS */
/*#use LIAISON_GPS  // Jumpers: 8 to 11, 7 to 12*/
#use rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS, STREAM=GPS)
/* Liaison FSK */
/*#use LIAISON_FSK  // Jumpers: 8 to 11, 7 to 12*/
#use rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7, BITS=8, ERRORS, STREAM=GPS)

/* Entree/Sortie Port B */
#use standard_io(B)

/* I2C */
#use I2C(MASTER, sda=PIN_C4, scl=PIN_C3, SLOW, FORCE_HW)

/* Constantes */
//#define NB_TEST             10
//#define LONGUEUR_BUFFER 10
#define LONGUEUR_GPS  41
#define TEMPS_DESCENTE1200    /* 20 minutes */

#define NB_MEMOIRE    4       /* Nombre de memoires I2C */
#define TAILLE_MEMOIRE65535   /* Taille d'une memoire en octets */
                              /* 64ko = 65536o */
                              /* 1o non utilisé : taille des varaibles */
#define MAX_TRAME_MEM 64      /* Nombre d'octets d'une trame pour la memoire I2C */
```

```
#define FSK_START     (short)0x00    /* FSK - bit de start */
#define FSK_STOP      (short)0x01    /* FSK - bit de stop */
#define FSK_TAILLE_CAPTEUR    10     /* FSK - taille en bit pour un capteur */
                                     /* Comprend bit de start, donnees , bit de stop */
#define NB_CAPTEUR    5

/* Identifiant I2C des capteurs */
#define I2C_JAUGE     0x90    /* Capteur accelero-jauges */
#define I2C_COMMERCE  0x92    /* Capteur accelero.com */
#define I2C_PITOT     0x94    /* Capteur pitot */
#define I2C_BOG               0x96    /* Capteur Bog */
#define I2C_PIEZO     0x92    /* Capteur accelero-piezo */
#define I2C_MINUTERIE 0x70    /* Minuterie */

/* Mot de commande des capteurs I2C */
#define MOT_JAUGE     0x00    /* Capteur accelero-jauges */
#define MOT_COMMERCE  0x84    /* Capteur accelero.com */
#define MOT_PITOT     0x00    /* Capteur pitot */
#define MOT_BOG               0x00    /* Capteur Bog */
#define MOT_PIEZO     0xf4    /* Capteur accelero-piezo */
#define MOT_MINUTERIE 0x00    /* Minuterie */

/* Etat minuterie */
#define MINUTERIE_ETEINTE             0xc7    /* Minuterie eteinte */
#define MINUTERIE_MODULE_TENSION      0x80    /* Minuterie Module (1) sous tension */
       /* valeur non significative */
#define MINUTERIE_PARACHUTE_TENSION   0x01    /* Minuterie Parachute (2) sous tension */  /*
valeur non significative */
#define MINUTERIE_2_TENSION           0x47    /* Les 2 Minuteries sous tension */
#define MINUTERIE_DECOLLAGE           0x45    /* Minuterie 2 a detectée le décollage */
#define MINUTERIE_OUVERTURE_MODULE    0x05    /* Le module est libéré */
#define MINUTERIE_OUVERTURE_PARACHUTE 0x01    /* La case parachute est libérée */
#define MINUTERIE_PAS_REPONSE         0xff    /* Pas de reponse */

/* Identifiant I2C des memoires */
#define I2C_MEM1      0xA0    /* Memoire 1 */
#define I2C_MEM2      0xA2    /* Memoire 2 */
#define I2C_MEM3      0xA4    /* Memoire 3 */
#define I2C_MEM4      0xA6    /* Memoire 4 */

/* Patte de controle alimentation */
#define ALIM_GPS      PIN_B1  /* Alimentation Recepteur GPS */
#define ALIM_FSK      PIN_B2  /* Alimentation Emetteur FSK */
#define DEL           PIN_B4  /* Diode Electro Luminescente */

void main(void);
/*****************************************************************************/
/* Description : programme de test                                           */
/* Parametres  : aucun                                                       */
/* Retour      : aucun                                                       */
/*****************************************************************************/

int initGPS();
/*****************************************************************************/
/* Description : initialisation du recepteur GPS                             */
/* Parametres  : aucun                                                       */
/* Retour      : <false> initialisation non effectuee                        */
/*               <true> initialisation effectuee                             */
/*****************************************************************************/

int initEmetteur(int frequence);
/*****************************************************************************/
/* Description : initialisation de la porteuse de l'emetteur                 */
/* Parametres  : <frequence> frequence de la porteuse                        */
/* Retour      : <false> initialisation non effectuee                        */
/*               <true> initialisation effectuee                             */
/*****************************************************************************/

int recupCapteur(byte numCapteur, byte motCommande, unsigned long int * data);
/*****************************************************************************/
/* Description : recuperation des donnees du capteur                         */
/* Parametres  : <numCapteur> adresse I2C du capteur a recuperer             */
/*               <motCommande> mot de commande au peripherique               */
/*               <data> donnees envoyees par le capteur                      */
/* Retour      : <false> capteur absent                                      */
/*               <true> capteur present                                      */
/*****************************************************************************/

int ecritureMem(byte * buff, unsigned int longueur);
```

```
/*****************************************************************************/
/* Description : ecriture de la trame sur la memoire                         */
/* Parametres  : <buff> donnees a ecrire                                     */
/*               <longueur> taille de buff                                   */
/* Retour      : <false> erreur d'ecriture                                   */
/*               <true> donnees ecrites                                      */
/*****************************************************************************/

int ecritureEmetteur(byte * buff, unsigned int longueur);
/*****************************************************************************/
/* Description : ecriture (emission) de la trame sur l'emetteur              */
/* Parametres  : <buff> donnees a ecrire                                     */
/*               <longueur> taille de buff                                   */
/* Retour      : <false> erreur d'ecriture                                   */
/*               <true> donnees ecrites                                      */
/*****************************************************************************/

int lectureGPS(char * buff);
/*****************************************************************************/
/* Description : recuperation de la position par GPS                         */
/* Parametres  : <buff> position du module                                   */
/* Retour      : <false> erreur de lecture                                   */
/*               <true> pas d'erreur                                         */
/*****************************************************************************/

int arretEmission();
/*****************************************************************************/
/* Description : arret de l'emission de l'emetteur                           */
/* Parametres  : aucun                                                       */
/* Retour      : <false> erreur                                              */
/*               <true> emission arretee                                     */
/*****************************************************************************/

int marcheEmission();
/*****************************************************************************/
/* Description : marche de l'emission de l'emetteur                          */
/* Parametres  : aucun                                                       */
/* Retour      : <false> erreur                                              */
/*               <true> emission en marche                                   */
/*****************************************************************************/

int ouvertureModule(unsigned byte * dataMinuterie);
/*****************************************************************************/
/* Description : ouverture du module                                         */
/* Parametres  : <dataMinuterie> etat minuterie                              */
/* Retour      : <false> module fermee                                       */
/*               <true> module ouvert                                        */
/*****************************************************************************/

int attenteDecollage(unsigned byte * dataMinuterie);
/*****************************************************************************/
/* Description : decollage de la fusee                                       */
/* Parametres  : <dataMinuterie> etat minuterie                              */
/* Retour      : <false> fusee non decollee                                  */
/*               <true> fusee decollee                                       */
/*****************************************************************************/

/* Fin de test.h ************************************************************/
```

axelle.c

```
/*****************************************************************************/
/* test.c                                                                    */
/*                                                                           */
/* Description : programme de test de la fusee,  sous l'environnement de     */
/*    developpement MPLAB IDE avec le compilateur c pic-c.                   */
/* Auteur : Pierre-Loic ROPARS                                               */
/*          Pierre-Loic.ROPARS@insa-lyon.fr                                  */
/* 09/11/2002 Creation                                                       */
/* 22/03/2003 Modification : ajout fonctions programme fusee                 */
/*****************************************************************************/

/* Inclusions */
#include "axelle.h"
```

```
//#define TEST
//#define MEMORY_COMPTEUR

/* Fonctions locales */
unsigned byte constructionTrameFsk(unsigned long * data);
void lectureMotMem(byte mem, unsigned long int address);
char timed_GPSgetc();
void vol();
void descente();

/* 9e bit liaison RS232 - Emission FSK */
//#bit ninth_bit = 0xff.7

/* variables globales */
//int etatGPS;
unsigned long int espaceUtiliseMem[NB_MEMOIRE] = { 0, 0, 0, 0 };
       /* Espace libre pour les memoires */
unsigned byte adresseMemoire[NB_MEMOIRE] =
       { I2C_MEM1, I2C_MEM2, I2C_MEM3, I2C_MEM4 };
       /* Adresse I2C des memoires */

void main(void)
/*****************************************************************************/
/* Description : phase d'attente decollage :                                 */
/*                   - initialisation                                        */
/*               phase de vol :                                              */
/*                   - acquisition des valeurs des capteurs                  */
/*                   - copie en memoire                                      */
/*                   - envoie par l'emetteur                                 */
/*               phase de descente :                                         */
/*                   - acquisition de la position du module                  */
/*                   - copie en memoire                                      */
/*                   - envoie par l'emetteur                                 */
/*               phase au sol :                                              */
/*                   - arret alimentation GPS                                */
/*                   - arret alimentation FSK                                */
/* Parametres  : aucun                                                       */
/* Retour      : aucun                                                       */
/*****************************************************************************/
{
       /* variables locales */
       //volatile int longueur = 0;
       //unsigned long int data2;
       //unsigned long int i;

       /*volatile unsigned long int data[(NB_CAPTEUR*6)+1];*/
       /*volatile unsigned byte dataFsk[NB_CAPTEUR+1];*/
       volatile unsigned byte dataMinuterie;

output_low(DEL);

       /* phase d'initialisation */
       output_low(ALIM_GPS); /* Arret Alimentation GPS */
       marcheEmission();     /* Alimentation FSK */
//     ninth_bit=1;          /* Emission : 2e bit de stop */

//     etatGPS = false;
       /*etatGPS = initGPS();*/

       /* attente decollage */
#ifdef TEST
       fprintf(FSK, "*** Attente decollage ****\n\r");
#endif
       while(!attenteDecollage(&dataMinuterie))
       {
              putc(0xff);
              printf("Axelle --- CLES-FACIL 2003 --- ");
              printf("minuterie <%x> --- ", dataMinuterie);
       }

       /* phase de vol */
#ifdef TEST
       fprintf(FSK, "*** Phase de vol *********\n\r");
#endif

       vol();
//     while(!ouvertureModule(&dataMinuterie))     /* attente ouverture module */
//     {
//printf(" minuterie <%x>", dataMinuterie);
```

```
//output_high(DEL);
//
//              data[0] = dataMinuterie;      /* etat minuterie */
//
//              /* lecture capteurs */
//              longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[1]);
//              longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[2]);
//              longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[3]);
//              longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[4]);
//              longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[5]);
//
//              longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[6]);
//              longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[7]);
//              longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[8]);
//              longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[9]);
//              longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[10]);
//
//              longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[11]);
//              longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[12]);
//              longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[13]);
//              longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[14]);
//              longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[15]);
//
//output_low(DEL);
//              longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[16]);
//              longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[17]);
//              longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[18]);
//              longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[19]);
//              longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[20]);
//
//              longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[21]);
//              longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[22]);
//              longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[23]);
//              longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[24]);
//              longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[25]);
//
//              longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[26]);
//              longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[27]);
//              longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[28]);
//              longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[29]);
//              longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[30]);
//
//              /* construction trame FSK */
//              dataFsk[0] = constructionTrameFsk(&data[0]);        /* minuterie */
//              dataFsk[1] = constructionTrameFsk(&data[1]);        /* I2C_JAUGE */
//              dataFsk[2] = constructionTrameFsk(&data[2]);        /* I2C_COMMERCE */
//              dataFsk[3] = constructionTrameFsk(&data[3]);        /* I2C_PITOT */
//              dataFsk[4] = constructionTrameFsk(&data[4]);        /* I2C_BOG */
//              dataFsk[5] = constructionTrameFsk(&data[5]);        /* I2C_PIEZO */
//
//              /* ecriture FSK */
//              ecritureEmetteur(dataFsk, NB_CAPTEUR+1);
//
//              /* ecriture memoire */
//              ecritureMem(data, 4 * ((NB_CAPTEUR*6)+1));
//      }

output_high(DEL);

        /* phase de descente du module */
//#ifdef TEST
        fprintf(FSK, "%x*** Phase de descente ***\r\n", 0xff);
//#endif
        output_high(ALIM_GPS);/* Alimentation GPS */

        descente();
//      for(i=0; i<TEMPS_DESCENTE; i++)
//      {
//#ifdef TEST
//      printf("*** Attente satellite ***\n\r");
//#endif
//output_high(DEL);
//              /* construction trame */
//              lectureGPS(&data2);
//output_low(DEL);
//      }

        /* test de fin de vol */
        output_low(ALIM_GPS); /* Arret Alimentation GPS */
```

```
        while (!arretEmission())
        {       /* Vide */
        }

#ifdef TEST
        fprintf(FSK, "*** Fin de vol ***********\n\r");
#endif

        /* phase fin de vol */
        for (;;)        /* boucle sans fin */
        {       /* Vide */
        }
}
/* Fin de main */

int initGPS()
/*****************************************************************************/
/* Description : initialisation du recepteur GPS                             */
/* Parametres  : aucun                                                       */
/* Retour      : <false> initialisation non effectuee                        */
/*               <true> initialisation effectuee                             */
/*****************************************************************************/
{
        fprintf(GPS, "@@Eq\1\65\r\n");

        return true;
}
/* Fin de initGPS */

int initEmetteur(int frequence)
/*****************************************************************************/
/* Description : initialisation de la porteuse de l'emetteur                 */
/* Parametres  : <frequence> frequence de la porteuse                        */
/* Retour      : <false> initialisation non effectuee                        */
/*               <true> initialisation effectuee                             */
/*****************************************************************************/
{
        return true;
}
/* Fin de initEmetteur */

int recupCapteur(byte numCapteur, byte motCommande, unsigned long int * data)
/*****************************************************************************/
/* Description : recuperation des donnees du capteur                         */
/* Parametres  : <numCapteur> adresse I2C du capteur a recuperer             */
/*               <motCommande> mot de commande au peripherique               */
/*               <data> donnees envoyees par le capteur                      */
/* Retour      : <false> capteur absent                                      */
/*               <true> capteur present                                      */
/*****************************************************************************/
{
        /* Variables locales */
        unsigned byte data1;  /* bits de poids fort */
        unsigned byte data2;  /* bits de poids faible */

        int etatACK = 1;      /* <0> Acquitement - <1> Non Acquitement */

        /* Lecture */
        i2c_start();
        etatACK = i2c_write(numCapteur);
#ifdef MAP
        fprintf(FSK, "etatACK 1<%d>\n\r", etatACK);
#endif
        etatACK = i2c_write((byte)(motCommande));    /* Commande */
#ifdef MAP
        fprintf(FSK, "etatACK 2<%d>\n\r", etatACK);
#endif
        i2c_start();
        etatACK = i2c_write(numCapteur|0x01);
        data1=i2c_read(1);            /* Lecture poids fort */
        data2=i2c_read(0);            /* Lecture poids faible */
        i2c_stop();

        /* Mise en forme */
        *data = ( ( (unsigned long int)data1 << 8 ) & 0xff00 ) | data2;

#ifdef MAP
        fprintf(FSK, "Capteur <%x> %x%x = %lu\n\r", numCapteur, data1, data2, *data);
#endif
```

```
	/* AJOUTER TEST DE LECTURE */
	return true;
}
/* Fin de recupCapteur */

int ecritureMem(byte * buff, unsigned int longueur)
/*****************************************************************************/
/* Description : ecriture de la trame sur la memoire                         */
/* Parametres  : <buff> donnees a ecrire                                     */
/*               <longueur> taille de buff                                   */
/* Retour      : <false> erreur d'ecriture                                   */
/*               <true> donnees ecrites                                      */
/*****************************************************************************/
{
	/* Variables statiques */
	static unsigned byte memoireAEcrire = 3;

	/* Variables locales */
	byte adresseFaible;   /* Adresse de poids faible */
	byte adresseFort;     /* Adresse de poids fort */
	byte page;            /* bit B0 de la memoire : page */

	int etatACK = 1;      /* <0> Acquitement - <1> Non Acquitement */

	unsigned int j;

	int retour = false;

#ifdef MEMORY_COMPTEUR
	static unsigned int mem_compteur = 0;

	buff[0] = mem_compteur++;
#endif

	/* Test de longueur maximal */
	if (longueur>MAX_TRAME_MEM)
	{
		return retour;
	}

	/* On ecrit par cycle 1/4 */
	if( (++memoireAEcrire)==4 )
	{
		memoireAEcrire = 0;
	}

	if ( (espaceUtiliseMem[memoireAEcrire] + longueur) < TAILLE_MEMOIRE )
	{
		/* Conversion de l'adresse */
		adresseFaible = espaceUtiliseMem[memoireAEcrire];
			/* Recuperation des bits de poids faible */
		adresseFort = (espaceUtiliseMem[memoireAEcrire] >> 8) & 0x7f;
			/* Recuperation des bits de poids fort sauf le 15e */
		page = ( (espaceUtiliseMem[memoireAEcrire] >> 15) & 0x01 ) << 3;
			/* Recuperation du 15e bit */
#ifdef MAP
	fprintf(FSK, "adresseFaible <%x>\n\r", adresseFaible);
	fprintf(FSK, "adresseFort <%x>\n\r", adresseFort);
	fprintf(FSK, "page <%x>\n\r", page);
	fprintf(FSK, "adresseMemoire[memoireAEcrire] <%x>\n\r",
adresseMemoire[memoireAEcrire]);
	fprintf(FSK, "adresseMemoire[memoireAEcrire] | page <%x>\n\r",
(adresseMemoire[memoireAEcrire] | page) );
#endif

		/* Ecriture */
		i2c_start();
		etatACK = i2c_write(adresseMemoire[memoireAEcrire] | page);
			/* Peripherique */
#ifdef MAP
	fprintf(FSK, "etatACK 1<%d>\n\r", etatACK);
#endif
		etatACK = i2c_write(adresseFort);
			/* Commande poids fort */
#ifdef MAP
	fprintf(FSK, "etatACK 2<%d>\n\r", etatACK);
#endif
		etatACK = i2c_write(adresseFaible);
```

```
                        /* Commande poids faible */
#ifdef MAP
        fprintf(FSK, "etatACK 3<%d>\n\r", etatACK);
#endif
                for (j=0;j<longueur;j++)      /* Donnee */
                {
                        etatACK = i2c_write(buff[j]);
#ifdef MAP
        fprintf(FSK, "Recu <%x>\n\r", buff[j]);
        fprintf(FSK, "etatACK 4-%u<%d>\n\r", j, etatACK);
#endif
                }
                i2c_stop();
                //delay_ms(11);

                /* Nouvel espace libre en memoire */
                if (longueur==1)
                {
                        espaceUtiliseMem[memoireAEcrire]++;
                }
                else
                {
                        espaceUtiliseMem[memoireAEcrire] += 64;
                }

                /* Test de verification d'ecriture */
                if (etatACK==0)
                {
                        retour = true;
                }
                break;
        }
        return retour;
}
/* Fin de ecritureMem */

int ecritureEmetteur(byte * buff, unsigned int longueur)
/*****************************************************************************/
/* Description : ecriture (emission) de la trame sur l'emetteur             */
/* Parametres  : <buff> donnees a ecrire                                     */
/*               <longueur> taille de buff                                   */
/* Retour      : <false> erreur d'ecriture                                   */
/*               <true> donnees ecrites                                      */
/*****************************************************************************/
{
        /* Variables locales */
        static unsigned long int numTrame = 0;
        unsigned int numTrameFort;
        unsigned int numTrameFaible;
        byte chekSum = 0;
        unsigned int i;

        /* Emission FSK */
        fputc(0xff, FSK);                   /* Debut de trame - Mot de synchronisation */
        chekSum ^= 0xff;
#ifdef MAP
        fprintf(FSK, "Trame - %lx\n\r", numTrame);  /* Numero de trame */
#else
        numTrameFort = (numTrame >> 8) & 0xff;
        numTrameFaible = numTrame & 0xff;
        fputc(numTrameFaible, FSK);                 /* Numero de trame */
        fputc(numTrameFort, FSK);                   /* Numero de trame */
#endif
        chekSum ^= numTrameFort;
        chekSum ^= numTrameFaible;
        for (i=0;i<longueur;i++)
        {
                /*fprintf(FSK, "%x", buff[i]);*/
        fputc(buff[i], FSK);
                chekSum ^= buff[i];
        }
        chekSum /= 2;
        fputc(chekSum, FSK);                        /* Fin de trame - Cheksum */
        numTrame++;
        return true;
}
/* Fin de ecritureEmetteur */

int lectureGPS(char * buff)
```

```
/*****************************************************************************/
/* Description : recuperation de la position par GPS                         */
/* Parametres  : <buff> position du module                                   */
/* Retour      : <false> erreur de lecture                                   */
/*               <true> pas d'erreur                                         */
/*****************************************************************************/
{
	/* variables locales */
	char charGps[LONGUEUR_GPS];
	char c;
	unsigned int i;
	unsigned int j;

	i=0;
	while(true)
	{
		if(timed_GPSgetc()=='$')
		{
			if(timed_GPSgetc()=='G')
			{
				if(timed_GPSgetc()=='P')
				{
					if(timed_GPSgetc()=='G')
					{
						if(timed_GPSgetc()=='G')
						{
							if(timed_GPSgetc()=='A')
							{
								if(timed_GPSgetc()==',')
								{
									break;
								}
							}
						}
					}
				}
			}
		}
		i++;
		if(i>500)
		{
			return false;
		}
	}

	i=0;
	/* heure */
	while( (c = timed_GPSgetc()) != ',' )
	{
		charGps[i++] = c;
	}
	charGps[i++] = ',';

	/* latitude */
	while( (c = timed_GPSgetc()) != ',' )
	{
		charGps[i++] = c;
	}
	charGps[i++] = ',';
	while( (c = timed_GPSgetc()) != ',' )
	{
		charGps[i++] = c;
	}
	charGps[i++] = ',';

	/* longitude */
	while( (c = timed_GPSgetc()) != ',' )
	{
		charGps[i++] = c;
	}
	charGps[i++] = ',';
	while( (c = timed_GPSgetc()) != ',' )
	{
		charGps[i++] = c;
	}
	charGps[i++] = ',';

	/* gps indicator */
	while( (c = timed_GPSgetc()) != ',' )
```

```
        {
                /* vide */
        }

        /* nombre de satellite */
        while( (c = timed_GPSgetc()) != ',' )
        {
                charGps[i++] = c;
        }
        charGps[i++] = ',';

        /* HDOP */
        while( (c = timed_GPSgetc()) != ',' )
        {
                /* vide */
        }

        /* altitude */
        while( (c = timed_GPSgetc()) != ',' )
        {
                charGps[i++] = c;
        }

        putc(0xff);           /* debut de trame */
        printf("gps ");
        for(j=0;j<i;j++)
        {
                fprintf(FSK, "%c", charGps[j]);
        }
#ifdef TEST
        fprintf(FSK, "\r\n");
#endif
        ecritureMem(charGps, i);

        return true;
}
/* Fin de lectureGPS */

int arretEmission()
/*****************************************************************************/
/* Description : arret de l'emission de l'emetteur                           */
/* Parametres  : aucun                                                       */
/* Retour      : <false> erreur                                              */
/*               <true> emission arretee                                     */
/*****************************************************************************/
{
        output_high(ALIM_FSK);

        return true;
}
/* Fin de arretEmission */

int marcheEmission()
/*****************************************************************************/
/* Description : marche de l'emission de l'emetteur                          */
/* Parametres  : aucun                                                       */
/* Retour      : <false> erreur                                              */
/*               <true> emission en marche                                   */
/*****************************************************************************/
{
        output_low(ALIM_FSK);

        return true;
}
/* Fin de marcheEmission */

int ouvertureModule(unsigned byte * dataMinuterie)
/*****************************************************************************/
/* Description : ouverture du module                                         */
/* Parametres  : <dataMinuterie> etat minuterie                              */
/* Retour      : <false> module fermee                                       */
/*               <true> module ouvert                                        */
/*****************************************************************************/
{
        static unsigned long int i = 0;

        /* Variables locales */
        unsigned byte data;   /* minuterie */
        int retour = false;
```

```
/*int ack=1;*/

	/* Lecture */
	i2c_start();
	/*ack=*/i2c_write(I2C_MINUTERIE|0x01);
//printf("ack %x", ack);
	data=i2c_read(0);		/* Lecture poids faible */
	i2c_stop();
	*dataMinuterie = data;

//fprintf(FSK, " minuterie <%x>", data);

/* test */
#ifdef TEST
	i++;
	if (i>400)
	/*if (i>4000)*/
	{
		retour = true;
	}
#ifdef MAP
	else
	{
		if(i%100==0)
		{
			fprintf(FSK, "%lu\n\r", i);
		}
	}
#endif
/* test */
#else
	if ( (data==MINUTERIE_OUVERTURE_MODULE) || (data==MINUTERIE_PAS_REPONSE) )
	{
		retour = true;
	}
#endif
	return retour;
}
/* Fin de ouvertureModule */

int attenteDecollage(unsigned byte * dataMinuterie)
/*****************************************************************************/
/* Description : decollage de la fusee                                       */
/* Parametres  : <dataMinuterie> etat minuterie                              */
/* Retour      : <false> fusee non decollee                                  */
/*               <true> fusee decollee                                       */
/*****************************************************************************/
{
	static unsigned long int i = 0;

	/* Variables locales */
	unsigned byte data;   /* minuterie */
	int retour = false;

	/* Lecture */
	i2c_start();
	i2c_write(I2C_MINUTERIE|0x01);
	data=i2c_read(0);		/* Lecture poids faible */
	i2c_stop();
	*dataMinuterie = data;

/* test */
#ifdef TEST
	if (i>100)
	//if (i>4000)
	{
		retour = true;
	}
	i++;
/* test */
#else
	if (data==MINUTERIE_DECOLLAGE)
	{
		retour = true;
	}
#endif
//fprintf(FSK, " minuterie <%x>", data);

	return retour;
```

```
}
/* Fin de attenteDecollage */

unsigned byte constructionTrameFsk(unsigned long * data)
{
	/* variables locales */
	unsigned byte retour;

	retour = *data>>4;
	if (retour==0xff)
	{
		retour = 0xfe;
	}
	return retour;
}

char timed_GPSgetc() {

   long timeout;
   long timeout_error;

   timeout_error=FALSE;
   timeout=0;
   while(!kbhit()&&(++timeout<50000))   // 1/2 second
       delay_us(10);
   if(kbhit())
               return(fgetc(GPS));
   else {
               timeout_error=TRUE;
               return(0);
   }
}

void vol()
{
	/* variables locales */
	volatile int longueur = 0;

	volatile unsigned long int data[(NB_CAPTEUR*6)+1];
	volatile unsigned byte dataFsk[NB_CAPTEUR+1];
	volatile unsigned byte dataMinuterie;

	unsigned int i;

	while(!ouvertureModule(&dataMinuterie))     /* attente ouverture module */
	{
#ifdef TEST
	printf(" minuterie <%x>", dataMinuterie);
#endif
output_high(DEL);

		data[0] = dataMinuterie;     /* etat minuterie */

		/* lecture capteurs */
		longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[1]);
		longueur = recupCapteur(I2C_COMMERCE,        MOT_COMMERCE,  &data[2]);
		longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[3]);
		longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[4]);
		longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[5]);

		longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[6]);
		longueur = recupCapteur(I2C_COMMERCE,        MOT_COMMERCE,  &data[7]);
		longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[8]);
		longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[9]);
		longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[10]);

		longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[11]);
		longueur = recupCapteur(I2C_COMMERCE,        MOT_COMMERCE,  &data[12]);
		longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[13]);
		longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[14]);
		longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[15]);

output_low(DEL);
		longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[16]);
		longueur = recupCapteur(I2C_COMMERCE,        MOT_COMMERCE,  &data[17]);
		longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[18]);
		longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[19]);
		longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[20]);
```

```
                longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[21]);
                longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[22]);
                longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[23]);
                longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[24]);
                longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[25]);

                longueur = recupCapteur(I2C_JAUGE,   MOT_JAUGE,     &data[26]);
                longueur = recupCapteur(I2C_COMMERCE,       MOT_COMMERCE,  &data[27]);
                longueur = recupCapteur(I2C_PITOT,   MOT_PITOT,     &data[28]);
                longueur = recupCapteur(I2C_BOG,     MOT_BOG,       &data[29]);
                longueur = recupCapteur(I2C_PIEZO,   MOT_PIEZO,     &data[30]);

                /* construction trame FSK */
                if (dataMinuterie==0xff)
                {
                        dataMinuterie = 0xfe;
                }
                dataFsk[0] = dataMinuterie;                         /* minuterie */
                dataFsk[1] = constructionTrameFsk(&data[1]);        /* I2C_JAUGE */
                dataFsk[2] = constructionTrameFsk(&data[2]);        /* I2C_COMMERCE */
                dataFsk[3] = constructionTrameFsk(&data[3]);        /* I2C_PITOT */
                dataFsk[4] = constructionTrameFsk(&data[4]);        /* I2C_BOG */
                dataFsk[5] = constructionTrameFsk(&data[5]);        /* I2C_PIEZO */

                /* ecriture FSK */
                ecritureEmetteur(dataFsk, NB_CAPTEUR+1);

                /* ecriture memoire */
                ecritureMem(data, sizeof(unsigned long int) * ((NB_CAPTEUR*6)+1));
        }

        /* passage en mode GPS */
        for(i=0;i<(NB_CAPTEUR*6)+1;i++)
        {
                data[i] = 0xff;
        }
        ecritureMem(data, sizeof(unsigned long int) * ((NB_CAPTEUR*6)+1));
        ecritureMem(data, sizeof(unsigned long int) * ((NB_CAPTEUR*6)+1));
        ecritureMem(data, sizeof(unsigned long int) * ((NB_CAPTEUR*6)+1));
        ecritureMem(data, sizeof(unsigned long int) * ((NB_CAPTEUR*6)+1));
}

void descente()
{
        unsigned long int data2;
        unsigned long int i;

        for(i=0; i<TEMPS_DESCENTE; i++)
        {
#ifdef TEST
        printf("*** Attente satellite ***\n\r");
#endif
output_high(DEL);
                /* construction trame */
                lectureGPS(&data2);
output_low(DEL);
        }
}

/* Fin de axelle.c ************************************************************/
```

5.2. Extraction des données en mémoire

test_mem.c

```
/* test_mem.c *****************************************************************/

//#define MAP

/* Declaration du processeur */
#include <16f876.h>          /* Microcontroleur */
```

```
#fuses HS,NOWDT,NOPROTECT
#use delay(clock=20000000)    /* Horloge du microcontroleur */

/* Liaison FSK */
       #use rs232(baud=4800, xmit=PIN_C6, rcv=PIN_C7, BITS=8)  // Jumpers: 8 to 11, 7 to 12
/* 9e bit liaison RS232 - Emission FSK */

#define TABULATION    ","

/* I2C */
#use I2C(MASTER, sda=PIN_C4, scl=PIN_C3, SLOW, FORCE_HW)
/* Identifiant I2C des memoires */
#define I2C_MEM1      0xA0    /* Memoire 1 */
#define I2C_MEM2      0xA2    /* Memoire 2 */
#define I2C_MEM3      0xA4    /* Memoire 3 */
#define I2C_MEM4      0xA6    /* Memoire 4 */

#define NB_MEMOIRE    4       /* Nombre de memoires I2C */
#define TAILLE_MEMOIRE 65535  /* Taille d'une memoire en octets */

#define LONGUEUR      64

#use standard_io(C)

void ecriture(unsigned long int adresse, byte mem, byte * data);
void lecture(unsigned long int adresse, byte mem, byte * data);
void lectureMotMem(byte mem, unsigned long int address);

unsigned long int espaceUtiliseMem[NB_MEMOIRE] = { 0, 0, 0, 0 };
       /* Espace libre pour les memoires */
unsigned byte adresseMemoire[NB_MEMOIRE] =
       { I2C_MEM1, I2C_MEM2, I2C_MEM3, I2C_MEM4 };
       /* Adresse I2C des memoires */

void main(void)
{
       /* variables locales */
       unsigned long int i;

set_tris_c(0xe7);
output_high(PIN_C3);
output_high(PIN_C4);

       printf("%%commence\r\n");

       for(i=0;i<65535;i+=64)
       {
#ifdef MAP
       printf("%%mot <%ld>\r\n", i);
#endif
              lectureMotMem(0, i);
              lectureMotMem(1, i);
              lectureMotMem(2, i);
              lectureMotMem(3, i);
       }
       printf("\r\n");

       printf("%%fini\r\n");
       for(;;) {}
}

void ecriture(unsigned long int adresse, byte mem, byte * data)
{
       byte adresseFaible;   /* Adresse de poids faible */
       byte adresseFort;     /* Adresse de poids fort */
       byte page;            /* bit B0 de la memoire : page */

       int etatACK = 1;      /* <0> Acquitement - <1> Non Acquitement */

       unsigned int i;

       /* Conversion de l'adresse */
       adresseFaible = adresse;
              /* Recuperation des bits de poids faible */
       adresseFort = (adresse >> 8) & 0x7f;
              /* Recuperation des bits de poids fort sauf le 15e */
       page = ( (adresse >> 15) & 0x01 ) << 3;
              /* Recuperation du 15e bit */
#ifndef MAP
```

```
        printf("adresseFaible <%x>\n\r", adresseFaible);
        printf("adresseFort <%x>\n\r", adresseFort);
        printf("page <%x>\n\r", page);
        printf("mem <%x>\n\r", mem);
        printf("mem | page <%x>\n\r", (mem | page) );
#endif

        /* Ecriture */
        i2c_start();
        etatACK = i2c_write(mem | page);
                /* Peripherique */
#ifdef MAP
        printf("etatACK 1<%d>\n\r", etatACK);
#endif
        etatACK = i2c_write(adresseFort);
                /* Commande poids fort */
#ifdef MAP
        printf("etatACK 2<%d>\n\r", etatACK);
#endif
        etatACK = i2c_write(adresseFaible);
                /* Commande poids faible */
#ifdef MAP
        printf("etatACK 3<%d>\n\r", etatACK);
#endif
        for (i=0;i<LONGUEUR;i++)      /* Donnee */
        {
                etatACK = i2c_write(data[i]);
#ifdef MAP
        printf("Recu <%x>\n\r", data[i]);
        printf("etatACK 4-%u<%d>\n\r", i, etatACK);
#endif
        }
        i2c_stop();
        delay_ms(11);
}

void lecture(unsigned long int adresse, byte mem, byte * data)
{
        byte adresseFaible;   /* Adresse de poids faible */
        byte adresseFort;     /* Adresse de poids fort */
        byte page;            /* bit B0 de la memoire : page */

        unsigned int i;

        /* Conversion de l'adresse */
        adresseFaible = adresse;
                /* Recuperation des bits de poids faible */
        adresseFort = (adresse >> 8) & 0x7f;
                /* Recuperation des bits de poids fort sauf le 15e */
        page = ( (adresse >> 15) & 0x01 ) << 3;
                        /* Recuperation du 15e bit */

#ifndef MAP
        printf("adresseFaible <%x>\n\r", adresseFaible);
        printf("adresseFort <%x>\n\r", adresseFort);
        printf("page <%x>\n\r", page);
        printf("mem <%x>\n\r", mem);
        printf("mem | page <%x>\n\r", (mem | page) );
#endif

        i2c_start();
        i2c_write(mem | page);              /* Peripherique */
        i2c_write((byte)(adresseFort));             /* Commande poids fort */
        i2c_write((byte)(adresseFaible));   /* Commande poids faible */
        i2c_start();
        i2c_write((mem | page)|0x01);       /* Peripherique */
        for (i=0;i<LONGUEUR-1;i++)
        {
                printf("Lect i2c <%x> devrait etre <%x>\n\r", i2c_read(1), data[i]);
        }
        printf("Lect i2c <%x> devrait etre <%x>\n\r", i2c_read(0), data[i]);
        i2c_stop();
}

void lectureMotMem(byte mem, unsigned long int address)
{
        /* Variables locales */
        byte adresseFaible;   /* Adresse de poids faible */
        byte adresseFort;     /* Adresse de poids fort */
```

```
        byte page;            /* bit B0 de la memoire : page */

        int i;                /* Numero de la memoire */
        int k;
        int l;

        int ack = 1;

        unsigned int donnee;
        unsigned long int valeur;

        i=mem;

        /* Conversion de l'adresse */
        adresseFaible = address;
                /* Recuperation des bits de poids faible */
        adresseFort = (address >> 8) & 0x7f;
                /* Recuperation des bits de poids fort sauf le 15e */
        page = ( (address >> 15) & 0x01 ) << 3;
                /* Recuperation du 15e bit */

#ifdef MAP
        printf("adresseFaible <%x>\n\r", adresseFaible);
        printf("adresseFort <%x>\n\r", adresseFort);
        printf("page <%x>\n\r", page);
        printf("adresseMemoire[i] <%x>\n\r", adresseMemoire[i]);
        printf("adresseMemoire[i] | page <%x>\n\r", (adresseMemoire[i] | page) );
#endif

        i2c_start();
        ack=i2c_write(adresseMemoire[i] | page);            /* Peripherique */
#ifdef MAP
        printf("ack<%d>", ack);
#endif
        ack=i2c_write((byte)(adresseFort));         /* Commande poids fort */
#ifdef MAP
        printf("ack<%d>", ack);
#endif
        ack=i2c_write((byte)(adresseFaible));       /* Commande poids faible */
#ifdef MAP
        printf("ack<%d>", ack);
#endif
        i2c_start();
        i2c_write((adresseMemoire[i] | page)|0x01); /* Peripherique */
        for (k=0;k<64-10-2;k+=10)
        {
#ifdef MAP
        printf("%%1-%d\r\n", k);
        #endif
                if(k==0)
                {
                        /* minuterie */
                        donnee = i2c_read(1);
                        valeur = ((unsigned long int)donnee)<<8;
                        donnee = i2c_read(1);
                        valeur += donnee;
                        k += 2;
                }
                else
                {
                        /* pas minuterie */
                        valeur = 12345;
                }

                printf("%lu", valeur);
                printf(TABULATION);
                for(l=0; l<5; l++, valeur=0)
                {
                        donnee = i2c_read(1);
                        valeur = donnee;                            /* inversion : poids faibles
*/
                        donnee = i2c_read(1);
                        valeur += ((unsigned long int)donnee)<<8;   /* inversion : poids forts
*/
                        printf("%lu", valeur);
                        printf(TABULATION);
                }
                printf("\r\n");
```

```
        }

        /* pas minuterie */
        valeur = 12345;
        printf("%lu", valeur);
        printf(TABULATION);

        for(l=0; l<5-1; l++, valeur=0)
        {
#ifdef MAP
        printf("%%2-%d\r\n",l);
#endif
                donnee = i2c_read(1);
                valeur = donnee;                            /* inversion : poids faibles */
                donnee = i2c_read(1);
                valeur += ((unsigned long int)donnee)<<8;   /* inversion : poids forts */
                printf("%lu", valeur);
                printf(TABULATION);
        }
#ifdef MAP
        printf("%%3-%d\r\n", l);
#endif
        donnee = i2c_read(1);
        valeur = donnee;                            /* inversion : poids faibles */
        donnee = i2c_read(0);
        valeur += ((unsigned long int)donnee)<<8;   /* inversion : poids forts */
        printf("%lu", valeur);
        printf("\r\n");
        i2c_stop();
#ifdef MAP
        printf("%%4-fin mot\r\n");
#endif
}

/* Fin de test_mem.c ************************************************************/
```

gps.php

```
<?php

//$tmp = dechex(12599);

//echo $tmp.'<br>';

$fp = fopen("gps.txt", "r");

$fp2 = fopen("gps-traite.txt", "w");
echo "fichier ouvert<br>";

//for ($i=0; $i<strlen($tmp); $i+=2)
//{
//    echo hexdec($tmp[$i]).'<br>';
//    echo hexdec($tmp[$i+1]).'<br>';
//    echo chr(hexdec($tmp[$i].$tmp[$i+1]));
//}

$val1 = "";
$val2 = "";
$val3 = "";
$val4 = "";
while ($data = fgetcsv ($fp, 1000, ","))
{
    //for ($i=0; $i<strlen($data); $i+=2)
    //{
    //    echo hexdec($data[$i]).'<br>';
    //    echo hexdec($data[$i+1]).'<br>';
    //    echo chr(hexdec($data[$i].$data[$i+1]));
    //}
    $num = count ($data);
    //print "<p> $num champs dans la ligne $row: <br>";
    $row++;
    for ($c=0; $c<$num; $c++)
    {
        if (strcmp($data[$c], "12345")!=0)
        {
```

```
$tmp = $data[$c];
$tmp = sprintf("%04x", $tmp);
//printf("%04x", $tmp);
            //print $data[$c] . "<br>";
//            $tmp = dechex($data[$c]);
            //echo "<b>".$tmp."</b><br>";

            for ($i=0; $i<strlen($tmp); $i+=4)
            {
//            //    echo hexdec($tmp[$i]).'<br>';
//            //    echo hexdec($tmp[$i+1]).'<br>';
              //echo chr(hexdec($tmp[$i].$tmp[$i+1]));
              $val4 = $val3;
              $val3 = $val2;
              $val2 = sprintf("% 1s", chr(hexdec($tmp[$i].$tmp[$i+1])));
              $val1 = sprintf("% 1s", chr(hexdec($tmp[$i+2].$tmp[$i+3])));

              if($c==0)
              {
                  fwrite($fp2, $val2);
                  fwrite($fp2, $val1);
              }
              else
              {
                  fwrite($fp2, $val1);
                  fwrite($fp2, $val2);
              }
//            //    echo $tmp[$i];
              if( ($val4==chr(0)) && ($val3==chr(0)) && ($val2==chr(0)) &&
($val1==chr(255)))
              {
                  //echo "<br>pl<br>";
                  fwrite($fp2, "\n");
              }
            }
        }
    }
}

fclose($fp);
fclose($fp2);
echo "<br>fichier fermé<br>";

?>
```

http://www.insa-lyon.fr/Associations/ClesFacil/

Jérôme Hamm
Jerome.hamm@insa-lyon.fr

Projet : Axelle

24 avril 2003
Version 2.1

09. Format télémesure Axelle 2003

L'expérience du projet Axelle repose sur les mesures de cinq capteurs, les valeurs acquises seront sauvegardées à bord de la fusée (EEPROM) et plus précisément à bord du module qui sera éjecté à culmination. Dans un soucis de plus grande fiabilité encore, nous émettrons la plus grande quantité possible de ces données à travers un KiwiMillenium (A son débit maximum qui est de 4800Bauds).

Le format utilisé respectera la spécification suivante, notre format de trames étant encapsulé dans le format SNR.

FF	N°trame	Minut	Cap1	Cap2	Cap3	Cap4	Cap5	CRC

Figure 1 : Format trame projet Axelle 2003

Les valeurs des capteurs sont échantillonnées sur 12 bits mais transmises sur 8 bits uniquement afin de préserver la bande passante. Il est à noter que les valeurs sur 12 bits seront stockées dans mémoire à bord du module.

Estimation du nombre de trames reçues :

Avec les hypothèses suivante :

- 1 bit de start, 1 bit de stop, octet d'information à 8 bits
- 2 octets pour le numéro de trame
- 1 octet pour la minuterie
- 1 octet par capteur
- 1 octet pour le CRC

soit une trame de 10 octets de 10 bits (8 bits d'info + 2 bits de contrôle)

Au total, nous prévoyons de transmettre (4800/90) 500 trames par seconde soit la moitié de nos acquisitions.

http://www.insa-lyon.fr/Insa/Associations/ClesFacil/

Pierre Fayet

fayetpierre@aol.com

Projet : Axelle

09 Octobre 2003
Version 3.0

10. Alimentations.

I) Introduction.

Le précédent projet (ELA) réalisé par le CLES-FACIL en 2001-2002 a montré l'importance capitale de l'alimentation des circuits electroniques d'une fusée. C'est en effet une infrastructure indispensable mais dont l'importance est parfois sous-estimée. De fortes contraintes y sont associées.

Ces contraintes sont de deux ordres :

- Le matériel à réaliser doit être de qualité « spatiale » : robuste, mais limité en poids, en autonomie, en nombre de tensions disponibles, ce qui n'est pas le cas dans l'industrie courante où l'on dispose des tensions du secteur, transformables à volonté par les moyens les plus divers.
- Il faut réaliser du matériel de mesure. Dans le cas d'une fusée, celui-ci doit être fiable, précis, convenablement dimensionné (on y reviendra dans une prochaine doc à paraître sur les systèmes de mesure embarqués et leur étalonnage). En ce qui concerne plus particulièrement les alimentations, cela signifie qu'il faut produire les tensions et les courants adaptés à chaque système de mesure.

Ces deux contraintes antagonistes permettent de comprendre toute l'ampleur du problème : Comment obtenir d'une part une large gamme de tensions destinées aux capteurs de mesure et à leurs circuits de traitement, tout en réduisant le nombre de piles (chères, lourdes, et resistant parfois mal aux accélérations), sans trop toucher à l'autonomie globale de l'ensemble ?

II) Description fonctionnelle et cahier des charges.

1) Comment résoudre la quadrature du cercle…

La question posée ci-dessus amène une remarque : le système d'alimentation parfait n'existe pas (cela se saurait), par contre on peut toujours trouver un compromis, moyennant certaines concessions.

L'idée forte qui est à la base de cette étude est de supprimer le concept d'alimentation centrale. En effet dans les précédents projets, les tensions d'alimentation des circuits étaient générés soit par des régulateurs, soit par des alimentations à découpage centraux, pour être ensuite dispatchés vers les différents circuits. Ce système présente quatre grands défauts :

- En cas de panne, ou de court-circuit sur une carte, plus aucun circuit n'est alimenté. Peu de moyens sont disponibles pour permettre à l'utilisateur de contrôler facilement l'alimentation de certains secteurs de la fusée.
- Problèmes de CEM : rayonnement des selfs pour les alims à découpage, prolifération des fils, etc…
- Toute la puissance electrique de la fusée est délivrée par le même ensemble de piles.

- Sur le plan électrique, les circuits sont obligatoirement alimentés par des tensions standard (ce qui n'est pas si mauvais), par contre, les capteurs de mesure nécessitent le plus souvent des tensions d'alimentation autres que ces tensions standard, (capteurs spécifiques, raisons de sensibilité…) et il faut alors transformer les tensions disponibles un peu partout. Finalement, les capteurs en bout de chaîne finissent par être alimentés par des tensions bruitées, polluées et peu stables. Les mesures en sont affectées, alors qu' elles sont, au fond, la raison d'être d'un projet. C'est à mon sens très dommage. J'ajouterai à cela le fait que des courants d'intensités très diverses sont nécessaires pour les différents secteurs de le fusée. Typiquement, les ordres de grandeurs vont de quelques dizaines de microampères à 0.5 Ampère. On devine évidemment que les moyens mis en œuvre pour obtenir ces courants seront très variés.

2) Concepts de base.

Concernant l'architecture globale du projet, au-delà du strict domaine des alimentations, l'idée qui gouverne l'ensemble du projet « axelle » est l'absence de véritable point central. (à l'exception du microcontrolleur chargé entre autres de collecter les données des capteurs de la fusée et de génerer les trames de télémesure). Chaque poste de mesure est complètement autonome, muni de son propre convertisseur A/N (I_2C), et de son propre système d'alimentation, lui aussi totalement indépendant des autres.

En ce qui concerne les postes de mesure, cela présente l'avantage pour l'utilisateur d'apporter une standardisation interne : On retrouvera en effet les mêmes connecteurs (et le même brochage) pour chacun des systèmes de mesure de la fusée.

Les deux autres avantages apportés par cette architecture sont :

- Une fiabilité accrue, une éventuelle panne d'un circuit n'affectant pas le fonctionnement des autres ;
- Moins de fils, ce qui se fait au détriment de la complexité de certaines cartes, mais il est bien plus facile de gérer la complexité d'un circuit imprimé conçu par C.A.O que celle d'un cablage « en l'air » (« 3D ») où les conditions minimales de CEM ne sont pas remplies, sans oublier bien entendu les risques certains de cassures de fils, ruptures de soudures, etc..., sachant que lorsque l'on entre dans de telles considérations, tout devient possible.

Pour en revenir au domaine qui nous interesse, on voit que les alimentations seront réparties, selon chaque secteur de la fusée (nous y reviendrons au chapitre « répartition »), ce qui implique également une répartition des piles alimentant chaque système.

On privilégiera l'utilisation quasi-systématique de régulateurs linéaires à faible tension de chute pour des raisons de stabilité en tension, d'échauffement, de facilité de mise en œuvre.

Qui dit répartition des alims à divers endroits, dit problème en ce qui concerne la commande et le contrôle de celles-ci. Il faut donc prévoir un système

permettant la mise en marche de l'alimentation concernée et son arrêt éventuel en cas de problème, ainsi que son contrôle en fonctionnement normal.

Sans oublier le principal : Tout cela doit rester le plus fiable possible.

Remarque :

Pour plus de clarté, j'utiliserai par la suite le terme « alimentation » ou « alim » pour désigner un ensemble constitué d'un régulateur (ou éventuellement d'une référence de tension dans le cas d'un capteur) ET de son système de commande.

Le terme de « pile » restera reservé exclusivement à un dispositif permettant la conversion de l'énergie chimique en énergie électrique. En effet je me suis aperçu, notamment au cours de discussions avec l'équipe « méca » du CLES-FACIL, de la nécessité de faire cette distinction, et ce, pour une raison fort simple : L'intégration d'un régulateur et des quelques composants qui y sont associés sur un circuit imprimé ne pose aucun problème si elle a été prévue suffisament tôt. Cela reste dans le domaine de l'électronicien. Par contre, l'intégration d'une pile dans la fusée ne peut se faire qu'avec l'avis d'un de nos mécaniciens. Les piles sont lourdes et volumineuses, subissent l'accélération de la fusée, bref, elles sont (je suis ?) la bête noire des mécaniciens du CLES. C'est un sujet à ne pas prendre à la légère. –c'est le cas de le dire.

3) Cahier des charges.

3.a) Autonomie.

Sur le plan de l'autonomie, deux points sont à relever

- L'autonomie globale doit respecter les valeurs imposées par Planète Sciences (Ex. ANSTJ) : 40 minutes à partir de la mise en rampe de la fusée.
- Une certaine cohérence est requise : chaque sous-système de la fusée doit avoir à peu près la même autonomie, c'est-à-dire que les courants délivrés par chaque ensemble de piles doivent être sensiblement égaux. C'est surtout une question de confort pour l'utillisateur qui, à partir de l'état d'une pile, doit avoir une idée d'ensemble de l'état d'usure des piles de la fusée.

3.b) Technologie des piles.

On préfèrere de loin la technologie Lithium. Plus compactes, de capacité accrue, pouvant débiter des courants élevés sous des tensions acceptables. On verra plus loin que leur courbe caractéristique est très interessante pour notre application. Ce type de pile tend d'ailleurs à se démocratiser sur le maché grand public malgré un coût encore dissuasif pour des applications demandant une forte puissance (Photo…).

3.c) Régulateurs.

Tous les régulateurs utilisés dans ce projet sont à faible tension de chute (Low Drop Out ou L.D.O. pour les intimes) qui peuvent délivrer leurs tensions et courants nominaux pour un état d'usure avancé de la (ou les) pile(s) sur la (les)quelle(s) ils débitent. Cela sera étudié en détail plus loin.
Leur mise œuvre doit prendre en compte d'une part les spécificités de ces composants (cf. Datasheets) et d'autre part les risques de pollution des tensions (CEM) et de dérive en température existant dans la fusée.
En outre ces régulateurs doivent être commandables (Marche/Arrêt) et comporter si possible un comparateur intégré permettant de nous renseigner sur l'état de la tension de sortie. (un simple bit suffit : **1** : O.K, **0** : K.O, par exemple).

3.d) Commande.

Le système de commande devra être fiable. Il va de soi qu'il doit être compatible à 100% avec la logique de commande du régulateur choisi.

NB : La définition de projet CLES-FACIL 2002-2003 a prévu d'intégrer une interface PC compatible USB qui permettra d'une part la lecture des données de mesures sur le bus I²C central et, éventuellement, le contrôle centralisé des alimentations, la fusée étant au sol.

Caractéristiques globales du système de commande:

- Doit permettre la mise en marche et l'arrêt à distance du régulateur dont il a la responsabilité.
- Doit permettre le contrôle à distance du bon (ou du mauvais) comportement du régulateur dont il a la responsabilité.
- Eventuellement, peut prendre l'initiative d'arrêter automatiquement une alimentation défaillante avant d'en avertir l'utilisateur.
- L'interface avec l'utilisateur doit être la plus simple possible, que ce soit au niveau de la visualisation du fonctionnement (LEDS), ou bien au niveau de la commande de mise en marche / arrêt. D'autre part, il ne faut pas sous estimer l'importance d'une visualisation efficace. C'est imposé par le cahier des charges du CNES et l'expérience montre que de précieuses heures auraient pu être gagnées lors des essais sur le précédent projet si celui-ci avait disposé d'une meileure visualisation.

3.e) Tensions et courants délivrés.

Il va de soi que les tensions délivrées devront correspondre aux besoins de chaque secteur de la fusée. On essayera autant que possible de se ramener à des tensions « standard » (-5V,+5V, 10.5V…). Ce point là sera étudié en détail dans le paragraphe suivant.

III) Répartition. Choix des tensions d'alimentation.

Le projet se décompose en deux parties. Les différents dispositifs de mesure sont placés dans le cône de la fusée. La fusée comprend également un module détachable qui contient un microcontrolleur centralisant les données de mesure, un récepteur GPS et un ensemble émetteur + modulateur.
Il ne sera pas tenu compte ici de l'alimentation de la minuterie, celle-ci étant complètement autonome.
Le tableau ci-dessous résume les différentes tensions d'alimentations utilisées.

	-5V	+5V	+10.6V*
Amplis d'instrumentation	X	X	
Convertisseurs A/N		X	
Microcontrôleur		X	
Récepteur GPS		X	
Emetteur + Modulateur			X

*: Voir les raisons de ce choix en IV.2.d

Certaines tensions référencées sont nécessaires. Voir tableau ci-dessous.

	+2.5Vref	+5Vref
Convertisseurs A/N		X
Capteurs		X
Offset ampli instru.	X	

En ce qui concerne les références de tensions pour capteurs, certains courants peuvent être assez grands (jauges de contraintes).

IV) Conception.

1) Complémentarité entre régulateur et pile.

Après comparaison des performances entre piles lithium et piles alcalines, il s'est avéré que les piles lithium avaient un très gros avantage : Elles présentent une caractéristique tension/temps -à courant débité constant- presque plate, avant que leur tension ne chute brutalement. Ceci peut être exploité pour augmenter l'autonomie d'un système. En effet un régulateur classique (78XX) a besoin pour fonctionner d'une tension correspondant à sa tension nominale, PLUS une certaine tension qui est en quelque sorte réservée à son propre fonctionnement. Le défaut en est que si le régulateur est alimenté par une tension largement supérieure à sa tension nominale (ex. 10V pour 5V), il va chauffer et dépenser une énergie que l'on préfèrerait utiliser autrement…

Le régulateur Low Drop Out a l'avantage de nécessiter un dépassement de sa tension nominale très faible pour fonctionner (quelques millivolts à quelques

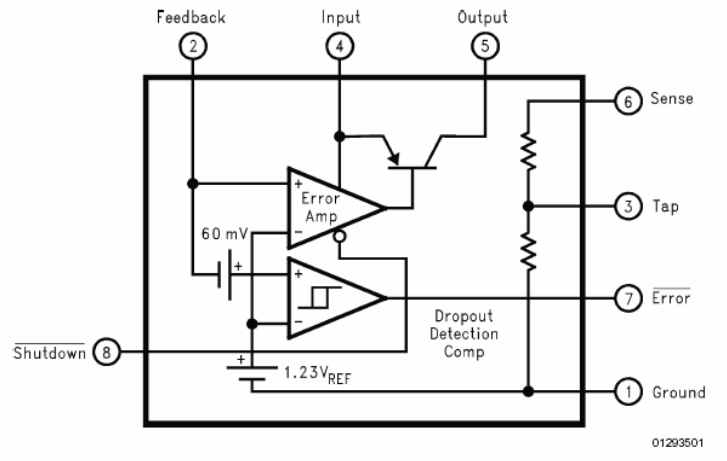
centaines de millivolts en fonction du courant qu'il doit débiter vers le circuit externe). Il dissipe peu d'énergie et exploite donc au maximum la caractéristique de la pile qui l'alimente. En effet, la caractéristique d'une pile lithium chutant brutalement au-delà d'un certain seuil, le régulateur ne pouvant plus fonctionner normalement va indiquer qu'il y a erreur et l'on saura alors de manière certaine que la pile correspondante est à remplacer.

Ainsi, on voit que le choix d'un type de régulateur et d'un type de pile ne peuvent être faits indépendamment.

2) Choix des régulateurs.

2.a) Architecture d'un LDO.

Le plus simple pour notre application était d'employer des LDO intégrant directement les fonctions marche / Arret et indicateur d'erreur.
Voici ci-dessous le schéma-bloc d'un régulateur LDO de chez National Semiconductors.



Intéressons-nous tout d'abord au fonctionnement de la partie régulation proprement dite. La régulation de la tension est obtenue à partir de l'ensemble transistor P avec son amplificateur d'erreur. La tension présente sur l'entrée « feedback » est comparée à une référence de 1.23V . Cette différence est alors renvoyée sur la base du transistor P qui se comporte alors en résistance variable. Pour obtenir un asservissement efficace de ce système, il faut évidemment que celui-ci soit bouclé. C'est le rôle du pont diviseur qui se trouve entre les pattes 1, 3 et 6. En reliant la patte « Sense » à la patte « output » et en reliant la patte « tap » à la patte « feedback », on obtient un système bouclé qui va réagir en fonction des variations de la tension de sortie. En effet, la tension présente sur la patte « tap » va alors commander la base du transistor par l'intermédiare du comparateur. Les ponts diviseurs intégrés dans ce genre de régulateurs sont en général calculés pour obtenir des tensions précises (5V, 3.3V), mais il est tout à fait possible de ne pas utiliser le pont diviseur interne. Un choix judicieux de résistances externes permet d'obtenir toutes sortes de tensions, fixes ou ajustables. On en verra un exemple dans le cas de l'alimentation de l'émetteur.

La patte « shutdown », elle, agit directement sur le comparateur en annulant sa tension de sortie. Le transistor se bloque quasi instantanément (au temps de recombinaison des charges dans le Silicium près) et le régulateur se met en mode de veille.

L'indicateur « error » est lui aussi généré par un simple comparateur. Dans l'exemple donné ci-dessus, on voit que la sortie du comparateur passe à zéro quand la tension de sortie est de 5% inférieure à sa tension nominale. Cette valeur est fixée par le rapport 0.06/1.235, rapport des tensions d'offset présentes à l'entrée du comparateur. La sortie de ce comparateur est une sortie en collecteur ouvert. C'est pourquoi la présence d'une résistance pull-up est nécessaire (valeur : de 100K à 1 Mohm)

Quelques précautions sont à prendre lors de le mise en œuvre de ces composants . En effet il faut impérativement penser à placer des condensateurs en entrée et en sortie de manière à assurer la stabilité du système et éviter que celui-ci ne se mette à osciller. Le choix de la valeur du condensateur importe peu du moment qu'elle soit suffisament grande. En général on utilise un condensateur au tantale, à cause aussi de sa résistance série, qui impose une certaine constate de temps au système, ce qui permet de le stabiliser. D'autres choix de condensateurs restent possibles, une visualisation de la tension de sortie à l'oscilloscope permettant de toute façon de s'apercevoir d'une éventuelle oscillation du circuit. A titre d'information il existe désormais des régulateurs de dernière génération qui permettent de se passer de ces condensateurs en entrée et en sortie.

Puissance dissipée.

Les boîtiers de ces régulateurs sont en général très petits (SO8) et ne peuvent dissiper une énergie trop grande. Le plus souvent ces composants sont munis d'un dispositif de protection thermique qui force le passage en mode veille en cas de surchauffe et les préserve de la destruction.
Le calcul de la puissance disspée se fait de la manière suivante :

$$P = (V_{in} - V_{out}) \cdot I_{out}$$

Cette relation motre que l'essentiel de l'énergie est dissipée au niveau du transistor et que les autres courants (alimentation des comparateurs) sont négligés devant le courant de sortie. En effet ($V_{in} - V_{out}$) représente la différence de potentiel régnant entre l'émetteur et le collecteur du transistor et I_{out} est le courant qui traverse ce transistor.
Or on sait que l'énergie pouvant être dissipée par le boîtier SO8 n'est pas illimitée, et la relation ci-dessus amène une remarque : reduire l'écart ($V_{in} - V_{out}$) améliore le rendement du système. Par contre il faut tenir compte du fait que Vin est variable et que, donc, le choix de Vout va conditionner l'autonomie du système.
Le calcul de la puissance maximale pouvant être dissipée est fonction de la température ambiante et de paramètres inhérents au système. Elle est donnée par la relation :

$$Pmax = (T_{J(MAX)} - T_A) / \theta_{J-A}$$

$T_{J(MAX)}$ et θ_{J-A} sont donnés par le constucteur, T_A est la température ambiante.

Je traiterai en détail le calcul de la puissance dissipée dans le cas de l'alimentation de l'émetteur. (Voir 2.d)

A titre d'information, les dernières générations de régulateurs LDO (Texas Instruments…) voient leur transistor bipolaire de sortie remplacé par un transistor MOS. Le transistor se comporte comme une résistance variable, mais de valeur plus faible que celle d'un transistor bipolaire. La tension de déchet est donc moindre, ainsi que l'échauffement. Cela permet de réguler des courants plus importants à taille égale (boitier SO8). (exemple : TPS 7225 Texas Instruments). Un régulateur de ce type est utilisé pour la regulation des tensions de –5V de la fusée (UCC384).

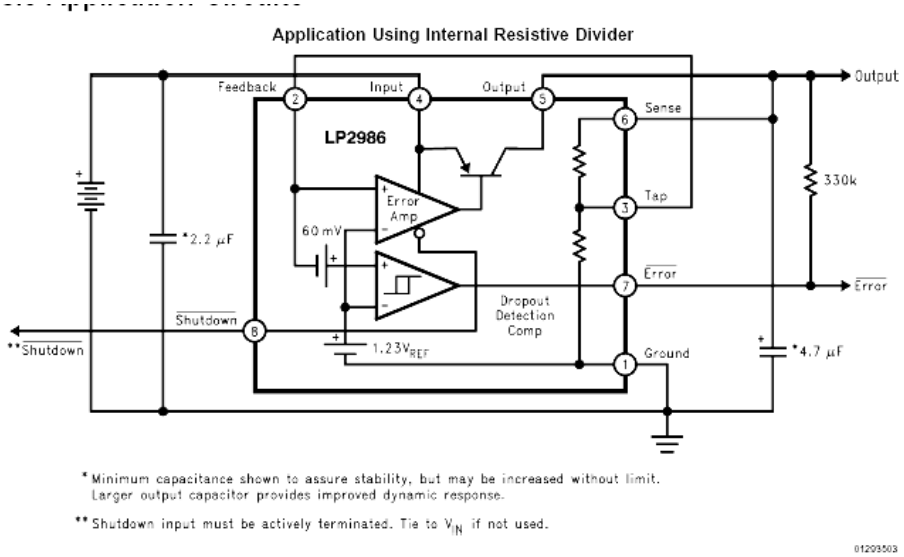
Pour terminer, voici quelques ordres de grandeur habituellement rencontrés :
Tension de déchet (drop out): quelques centaines de millivolts à pleine charge.
Précision : 1% pour une température de 25°C.

2.b) Tensions de +5 Volts .

→Microcontrôleur, Amplificateurs d'instrumentation, Filtres, récepteur GPS.

Pour le régulateur mon choix s'est porté sur le modèle LP2951 de National Semiconductors. (Pour l'anecdote, Le CLES-FACIL en disposait d'un stock d'environ 200 pièces). Ces régulateurs sont des régulateurs LDO commandables.

La fonction « error » est utilisée pour commander l'écairement d'une DEL soit via un transistor 2N2222 fonctionnant en commutation, soit via une porte inverseuse pouvant gérer directement l'éclairement d'une LED (30mA). Ainsi, on peut vérifier le fonctionnement de chaque secteur de la fusée.(cf.chap. « la carte de visualisation »).



NB : Pour le récepteur GPS, j'ai choisi le régulateur LP2986 en raison de son courant de sortie plus élevé (c'est le schéma ci-dessus. Le schéma du LP2951 ne diffère que de quelques valeurs de composants.)

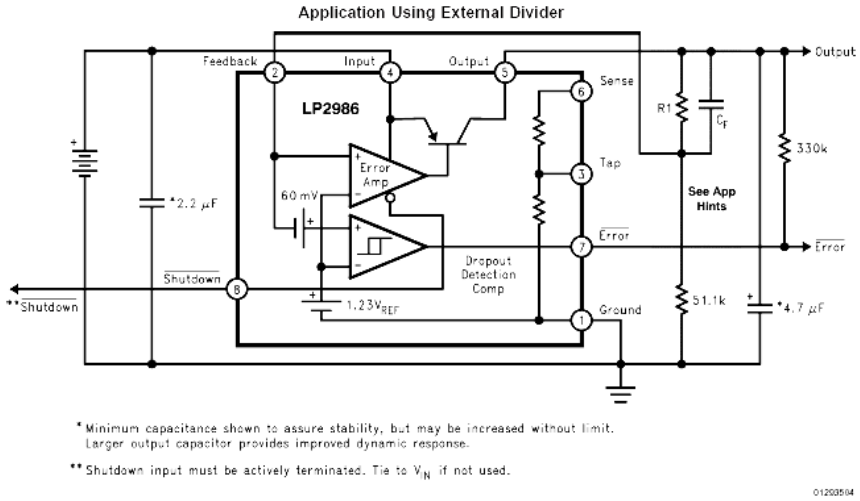
2.c) Tensions de –5 Volts

→Amplificateurs d'instrumentation.

J'ai employé le régulateur UCC284 (ou UCC384, c'est le même !), Texas Instruments, dans un montage standard. Attention toutefois au sens des condensateurs d'entrée et de sortie : Nous sommes dans le cas de tensions négatives.

2.d) Alimentation de l'émetteur.

Voici un exemple de montage particulièrement intéressant, faisant appel à un pont diviseur de tensions externe dans la boucle de régulation du LDO, et qui se prête bien à un calcul de puissances dissipées.



Le calcul de la tension de sortie se fait grâce à la formule :

$$V_{out} = 1.23\ (1 + R1/51.1K)$$

En pratique, il ne faut pas oublier que seules certaines valeurs standard de résistance sont disponibles. Dans le cas de l'émetteur, le problème était le suivant :

La puissance maximale pouvant être dissipée par le boîtier SO8 est de 375mW. Pour ce calcul on a pris une température ambiante de 60°C, $T_{J(MAX)}$ est de 125°C pour le LP2986 et θ_{J-A} de 160 °C/W

Il faut donc trouver une tension de sortie qui représente un bon compromis : voisine de 12V : la pile va s'user et le drapeau « error » va indiquer une usure de la pile au bout de peu de temps. Avantage : la puissance dissipée

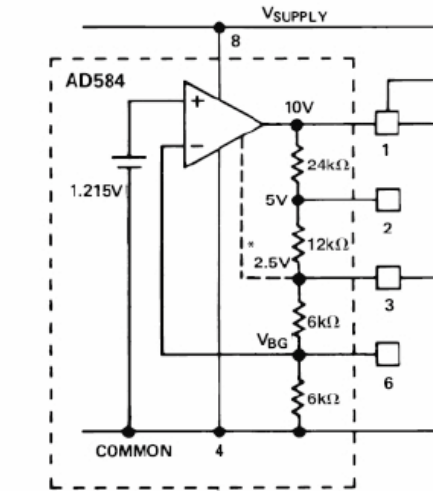
serait moindre puisque la d.d.p aux bornes du transistor P du régulateur serait assez faible. Si la tension de sortie est trop faible, à courant égal, la dissipation deviendrait assez conséquente, sans oublier que la puissance rayonnée par l'émetteur est directement proportionelle à la tension qui lui est appliquée.

J'ai choisi R1 = 390 Kohms, (valeur standard) ce qui donne une tension de sortie de 10.6 Volts. La puissance disspée par le transistor du régulateur serait alors de (12 – 10.6) . 0.2 = 273 mW en supposant que la charge débite un courant de 200mA et que la tension fournie par les piles est effectivement de 12 Volts. On voit que pour cette tension de sortie, la puissance dissipée est inférieure à la puissance maximale pouvant être dissipée par les boitiers SO8 (environ 73%), En réalité, on se trouve plutôt autour des 60%, sachant que j'ai pris un peu de marge sur le courant demandé par l'émetteur. Pour limiter l'échauffement du régulateur il est par contre indispensable que le routage du circuit imprimé tienne compte du fait que ce régulateur va chauffer. Un large plan de masse est à prévoir sous le composant, et on évitera pour une fois de placer un frein thermique au niveau de la patte de masse du régulateur.

Le condensateur Cf présent sur le schéma est un condensateur de compensation, qui permet de placer un zéro dans la fonction de transfert du système en boucle fermée pour une fréquence voisine de 50 KHz. Il se calcule avec la formule classique : $f = 1/2\pi R1C$. On trouve Cf de l'ordre de 10pF dans cet exemple.

3) Références de tension.

J'ai employé les références traditionelles du CLES-FACIL : AD584 de chez Analog Devices. Ce composant dispose de plusieurs tensions de sortie, d'une patte « strobe » permettant de le commander en marche/arrêt. Plus précis qu'un LDO (0.3% pour l'AD584J, 30ppm dans le pire des cas), il peut délivrer des courants allant jusqu'à 10mA et être interfacé avec un transistor de puissance pour des courants plus importants. Son synoptique rappelle un peu celui d'un LDO classique. C'est en tout cas le même concept qui est employé.



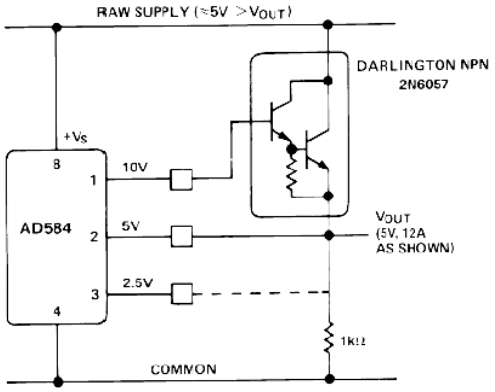
On voit qu'il ne faut pas demander un courant de sortie trop important, au risque de déséquilibrer le système et de rendre la régulation impossible.

3.a) références pour convertisseurs.

Les courants consommés par ces composants étant extrêmement faibles, il suffit de connecter la patte Vref de ces composants à la tension de référence désirée disponible sur l'AD584.

3.b) références pour les jauges de contraintes.

Un rapide calcul : quatre jauges de contrainte montées en pont de Wheatstone ont un résistance équivalente de 120 Ohms. Sous 5 Volts, cela correspond à un courant de 41.7mA. Il faut donc adapter l'AD 584 pour pouvoir fournir ce courant. J'ai donc créé un montage s'inspirant du montage suivant :



N'ayant nullement besoin d'un montage pouvant délivrer 12A en sortie, j'ai simplement remplacé le transistor darlington par un transistor 2N2219, qui peut supporter un courant de 40 mA sans problème. La puissance fournie aux jauges est de 41.7 x 5 = 209mW, soit un peu moins d'un quart de watt. Pour plus de sécurité, ce transistor est enduit de silicone et muni d'une collerette de refroidissement. Il a été testé pendant de nombreuses heures lors des différents essais et lors de l'etalonnage de la carte. Aucun disfonctionnement n'est survenu.

Le défaut principal de ce montage est de nécessiter une tension d'entrée assez conséquente. Comme aucune tension supérieure à six volts n'était disponible sur les cartes de mesure, ce problème a été résolu en connectant directement l'alimentation de ce montage entre –6 et +6 Volts (La tension de -6Volts étant régulée par ailleurs en –5V pour l'alimentation des Aops de mesure.) La tension de sortie n'est alors plus référencée par rapport à la masse mais par rapport au –6V. Les jauges voient toujours une différence de potentiel de 5V à leurs bornes, même si la tension de -6V varie. C'est un montage flottant. Par contre cette technique a l'inconvénient de faire naître une tension de mode commun en sortie du pont de Wheatstone, qui plus est variable en fonction de l'état d'usure des piles, surtout celle assurant l'alimentation négative du circuit, mais ce n'est pas si gênant si l'on choisit un amplificateur d'instrumentation ayant un bon taux de réjection en mode commun.

3.c) références « low drop out ».

On a déjà évoqué le fait que la référence AD584 ne peut, seule, débiter un courant trop important et qu'elle nécessite une tension d'alimentation nettement supérieure (c'est-à-dire de l'ordre du volt) à la tension référencée que l'on souhaite obtenir en sortie. C'est là deux défauts majeurs qui dans un cas particulier d'utilisation sur Axelle s'est transformé en un redoutable piège, heureusement décelé lors des tous premiers essais de la carte finale.

Il s'agissait de génerer, via un pont diviseur, une tension d'offset pour l'amplificateur d'instrumentation de l'accéléromètre à lamelle mobile, afin de ramener sa tension de sortie dans une plage de tensions acceptable pour le convertisseur A/N, lui-même utilisant cette même référence pour assurer sa conversion. Or, l'entrée « Vref » de l'ampli INA114 retenu pour cette application présentait une impédance d'entrée relativement basse. L'idée était de diviser la tension de référence du convertisseur A/N à travers un pont diviseur résistif de suffisament bonne qualité (dérive en température) et donc de récupérer cette tension en sortie de pont pour générer l'offset de l'INA114. On peut choisir pour le pont deux résistances de valeur suffisament élevées afin de ne pas trop utiliser de courant et donc de permettre un bon fonctionnement de l'AD584 qui est plutôt chatouilleux sur ce point. Seulement voilà : l'impédance d'entrée de la patte « Vref » est basse. Il y a donc désadaptation d'impédance et la valeur de tension obtenue sur cette patte n'est absolument pas conforme à la tension désirée et donnée par le calcul. On pense alors spontanément à diminuer les valeurs des résistances du pont diviseur, quitte à consommer un peu plus, afin de se rapprocher de l'impédance d'entrée de « Vref » mais on se retrouve alors face au fait que l'AD584 n'est pas conçu pour des courants trop élevés. De plus, en alimentant ce composant à partir de +6V pour générer +5V, on se trouve à la limite de fonctionnement de celui-ci. Il faut se rendre à l'évidence : l'AD584 convient parfaitement sous des tensions d'alimentation suffisamment élevées, on en trouve à divers endroits de la fusée, mais là il faut trouver autre chose.

On a besoin d'une référence à faible tension de déchet, mais devant débiter des courants suffisament importants. Analog Devices propose les séries REF19X qui correspondent à ce besoin. Pour simplifier, nous dirons que tous les concepts abordés plus haut pour les régulateurs LDO restent valables ici, ces composants ne sont en somme que des régulateurs de précision. Ordres de grandeurs pour la référence REF198 : tension de sortie 4.096Volts ; courant de sortie maximal :30mA ; tension de déchet maximale : 0.5V. Curieusement Analog Devices ne propose pas pour le moment ce composant en version 5V, ce qui est fort dommage. Nous nous sommes donc contentés de la version 4.096V (0.5%), ce qui a peu d'importance puisque cette référence sert aussi pour le convertisseur A/N. Ce composant, d'une mise en œuvre très simple et convenant à de nombreuses applications nous a donné entière satisfaction et nous a permis de rsoudre le problème. Il a également été utilisé pour l'accéléromètre ADXL150 (du même constructeur) embarqué dans Axelle.

4) Commande.

La commande « marche - arrêt » des régulateurs et des références est faite à l'aide de transistors 2222 et 3906 fonctionnant en commutation. Une convention a été adoptée. Pour les cartes de mesure, chacune est pilotée par un seul fil, un état haut sur ce fil provoque la mise en marche et un état bas l'arrêt de l'ensemble des alimentations de la carte. Ces états seront générés par de simples switches disposés ensemble sur une même carte qui portera également les LEDs de visualisation. L'avantage de ce système est la possibilité de mettre sous tension des montages à distance, sans avoir à transporter d'énergie : les circuits « de puissance » et les circuits de commande sont distincts. Plus besoin donc de « gros » interrupteur. Surtout, il n'est plus nécessaire de transporter l'energie au travers de la fusée par de longs fils – parasitables à souhait - pour assurer la mise sous tension de tel ou tel montage.

On a des régulateurs fonctionnant avec des logiques diverses et sous des tensions diverses, aussi serait-il fastidieux et inutile de dresser ici l'inventaire des différents montages à base de transistors 2N2222, 2N3906, trucs et astuces employés pour faire fonctionner l'ensemble. Toujours est-il que les cartes réalisées fonctionnent réellement sans encombre. Une précaution supplémentaire a été prise : les cartes comportent des jumpers que l'on peut court-circuiter pour forcer le fonctionnement du système en cas de problème.

En ce qui concerne le module, la commande est générée par le microcontrolleur, ce qui permet notamment l'extinction de l'émetteur au bout d'un certain temps, comme le préconise le cahier des charges de la fusée, mais aussi la mise en marche et l'arrêt du récepteur GPS. Naturellement, une commande manuelle a été prévue, ne serait-ce que pour une utilisation lors d'essais.

5) Choix des piles.

Il restait à choisir un modèle de pile lithium dans une gamme de prix raisonnable, si possible facile à se procurer (donc relativement standard) correspondant à nos besoins en ce qui concerne leur tension nominale et leur capacité. Deux possibilités sont apparues : soit utiliser des piles SAFT LSH 26180 (deux éléments en série permettant d'obtenir une tension de 7.2V), soit d'utiliser les modèles 2CR5 VARTA ou Ultra 223 Duracell, qui sont des piles photo de tension nominale 6V et de capacité nominale de 1.3Ah pour un poids de 42 grammes par pièce. Une demande de sponsoring a alors été faite et le CLES–FACIL a reçu peu après un lot de 30 piles 2CR5 VARTA. Je tiens ici à remercier cette firme.

Des tests sur la capacité des piles n'ont pu être menés par manque de temps, de plus ils auraient nécessité le sacrifice d'une ou deux précieuses piles, néanmoins celles-ci se sont révélées très puissantes. Il a en effet été possible de faire fonctionner l'émetteur lors d'essais pendant près d'une dizaine d'heures sans que la puissance de l'émission en soit trop affectée.

V) Réalisation.

Le prototypage et la réalisation ont été faits dans les locaux du CLES-FACIL, le routage des circuits s'effectuant sous Protel 99SE. La réalisation n'a été rendue possible que par un travail en équipe régulier, puisque les alimentations sont réparties selon différents secteurs.

VI) Bilan de consommation et autonomie.

Ce paragraphe a été établi à partir mesures de courants effectués le 29 Juillet 2003 sur la fusée afin d'en dresser le bilan de consommation global. Ces résultats sont a considérer avec une certaine prudence et ce, pour plusieurs raisons :

- Incertitudes de mesure, certains courants étant assez faibles et difficilement mesurables avec précision ;
- Calculs d'autonomie effectués de manière approximative à partir de la capacité nominale des piles donnée par le constructeur. Nous ne disposions malheureusement pas de leur caractéristique précise, ce qui aurait permis un calcul plus fiable, nottament pour les courants les plus forts (GPS, Emetteur...).

Néanmoins les ordres de grandeur donnés dans cette étude sont réalistes et nous autorisent la plus grande confiance en ce qui concerne l'autonomie de la fusée.

On trouvera page suivante le document tel qu'il a été rédigé au cours de la campagne Sissone 2003. Ce dernier inclut également les calculs d'autonomie de la minuterie et de la carte de visualisation qui étaient alimentées par des piles 9V et dont les alimentations n'ont pas été traitées en détail dans cet exposé. Elles sont en fait réalisées de manière classique à l'aide d'un régulateur 7805 pour la visualisation et LP2951 pour la minuterie. (Se reporter aux chapitres correspondants pour plus de précisions).

Bilan de consommation électrique de la fusée

Capteurs :

- **Accéléromètre à lamelle :**

Piles lithium +6V/-6V en configuration symétrique.

Consommation en mode veille :

Fil positif de l'alimentation :1,26 mA.
Autonomie 27 heures environ.

Fil négatif de l'alimentation : 1mA.
Autonomie 29 heures environ.

Consommation en fonctionnement normal :

Fil positif de l'alimentation : 47,5 mA
Fil négatif de l'alimentation : 44,6 mA

Consommation dûe essentiellement à la consommation du pont de Wheatstone des jauges piézorésistives.

- **Accéléromètre du commerce/ Capteur piézoélectrique :**

Consommation pile 6 Volts : 4,45 mA, soit une autonomie de 292 heures (12 jours). Il n'est pas envisagé de remplacement de cette pile pour le vol final.

- **Tube de Pitot :**

Deux piles Lithium 6V en configuration +6/+12 V, également utilisées pour alimenter le dispositf « Bog ».

Mode veille :
Mesure fil 12V : 0,170 mA.
Mesure fil 6V : 1,15 mA.

Mode normal :

Fil 12V : 2,24 mA.
Fil 6V : 4,34 mA.

- **Bog :**

Mode Veille :
Fil 12 V : 0,170 mA.
Fil 6 V : 0,711 mA.

Mode normal :

En moyenne 50 mA sur les deux piles, cette consommation variant en fonction du chauffage du filament du dispositif.

Autonomie de l'ensemble : environ 26 h.

Pour les dispositifs de mesure, une autonomie très supérieure à celle requise par le cahier des charges est obtenue.

Carte de visualisation :

Quatre piles 9V reliées en parallèle.
Consommation avec toutes les LEDs de visualisation en rouge + une LED séquenceur: 300 mA.
Consommation en rampe : LEDs vertes allumées, plus une LED séquenceur : 280 mA.

Autonomie de l'ensemble : 1 heure sur une seule pile, donc largement supérieure à une heure avec quatre piles en parallèle (sans pour autant atteindre quatre heures !)

Séquenceur :

Deux minuteries indépendantes alimentées chacune par une pile 9V.

Pile 1 :

En veille : 0,5 mA.
Attente : 36 mA.
Moteur : 76 mA.

Pile 2 :

En veille : 0,07 mA.
Attente : 35 mA.
Moteur 170 mA.

Autonomies estimées, en attente dans la rampe : dix heures environ.

Module :

GPS : 1 pile Lithium 6V.
Consommation récepteur GPS : 197 mA. Autonomie : 6h30.

Emetteur Toucan : deux piles Lithium 6V en série.
Consommation émetteur + modulateur XR2206 : 170 mA. Autonomie 6h30.

Processeur PIC : Une pile Lithium 6V : Consommation : 5 à 10 mA. Autonomie 260 h.
Il n'est pas envisagé de remplacement de cette pile pour le vol final.

VII) Conclusion.

On se trouve face à un système d'alimentation homogène, dimensionné au cas par cas en fonction des besoins en courants et tensions nécessaires aux différentes cartes, doté qui plus est d'un certain confort d'utilisation en ce qui concerne le système de visualisation du fonctionnement du système.

La fiabilité de ce système a dépassé mes espérances : aucun incident à ce sujet en campagne n'est à déplorer, le système de visualisation a été particulièrement utile lorsque, lors d'essais, le moment de remplacer les piles (émetteur, GPS) était venu. Ce système fonctionne encore sans faillir au moment où j'écris ces lignes (Oct. 2003) lors de démonstrations faites aux visiteurs du club. Il sera peut-être utile à l'avenir d'améliorer ce système au niveau de sa commande, quelque peu archaïque : remplacement des inverseurs logiques à base de transistors 2N2222 et 2N3906 par une logique CMOS, intégrée et moins « énergivore ». Ceci est néanmoins un point de détail.

VIII) Annexes.

- LP2986, LP2951, LDO regulators, Datasheet, National Semiconductors.
- UCC384, LDO regulator, Datasheet, Texas Instruments.
- AD584, Pin programmable precision voltage reference, datasheet, Analog Devices.
- REF198, precision micropower, low dropout voltage references, datasheet, Analog Devices.

Ces datasheets sont disponibles sur les sites web des différents fabricants.

CLES-FACIL

INSA LYON

http://www.insa-lyon.fr/Associations/ClesFacil/

Jérôme HAMM
jerome.hamm@sdbdc.org

Projet : Axelle

31 Septembre 2003
Version 2.0

11. La carte de visualisation.

I. Principe général :

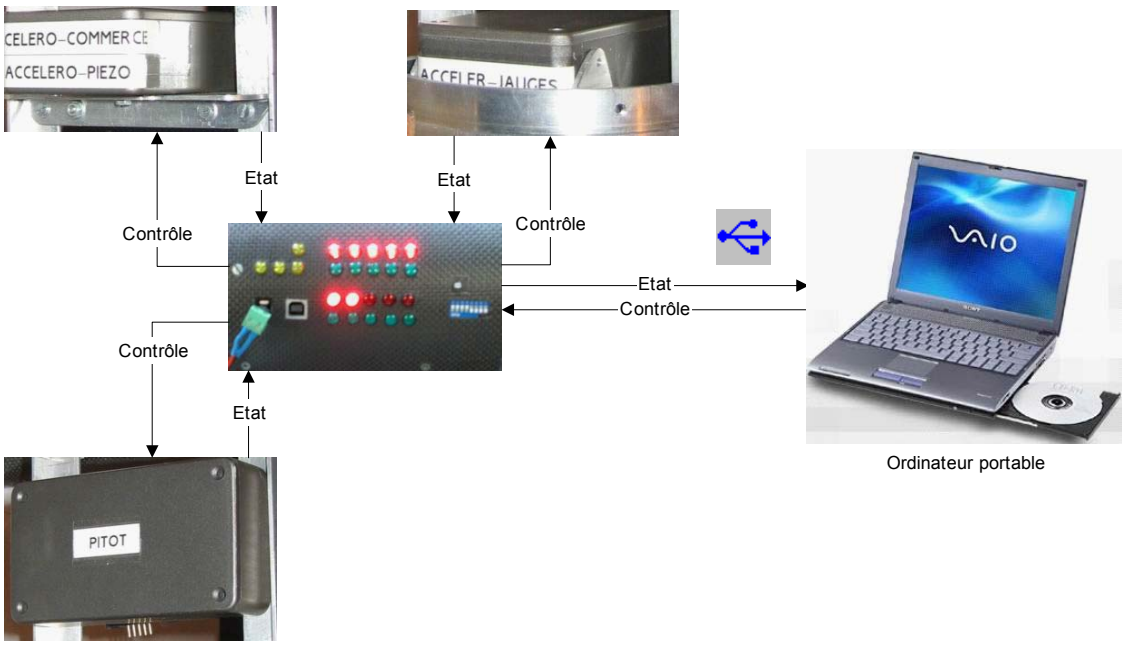


Figure 1 : schéma de principe général du contrôle des alimentations.

L'idée force de cette année a été de répartir les différentes alimentations, et les déporter à proximité des différents capteurs. Ce concept nous a été dicté par les difficultés que nous avions eues l'an dernier avec l'alimentation centralisée d'ELA. Un autre aspect qui nous a fait perdre beaucoup de temps, provenait du fait que nous n'avions pas de moyen de contrôle simple sur ces alimentations. Nous avons donc décidé de placer des DEL reflétant l'état des différents éléments. Ainsi toutes les alimentations disposent de deux voyants lumineux (l'un vert, l'autre rouge) indiquant quand le capteur correspondant est en état de marche, et quand il est éteint ou défaillant.

Pour le développement (et du fait de la consommation électrique importante de certains capteurs), nous avons choisi de pouvoir allumer et éteindre indépendamment tous les capteurs, ce rôle était également joué par cette carte.

Cette carte supporte également les diodes de visualisation de l'état des deux séquenceurs et leur allumage.

A cause de la grande consommation électrique des différentes DEL haute luminosité (30 mA), un système de connecteur Jack permet de couper l'alimentation des piles et d'utiliser une alimentation externe.

De plus, la carte dispose d'un microcontrôleur qui possède une interface USB, ce qui permet à un PC (le driver a été fait pour Windows XP) de *monitorer* en temps réel l'état des différentes alimentations. A l'origine, on devait pouvoir contrôler leur mise en marche également, mais certains problèmes de commande nous en ont empêché.

II. Principe de l'alimentation externe :

Figure 2 : Prise Jack nue, alimentation par les piles.

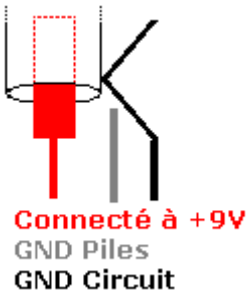


Figure 3 : Prise nourrice branchée, alimentation externe.

L'insertion de la prise externe coupe le chemin de masse reliant les piles à la carte, ainsi si on utilise une alimentation externe, le courant provient de celle-ci, sinon, les DEL sont simplement éteintes. C'est dans ces conditions que nous transportions la fusée. Lors de la mise en place du propulseur, c'est dans cette configuration que la carte était, le connecteur dépassant du corps. Les manipulations ont exercé un couple sur le connecteur, et la prise a été endommagée.

Figure 4 : Connecteur endommagé.

Lors de la mise en place du propulseur, à cause d'une trop grande précipitation, ce connecteur a été endommagé (en fait, la prise externe était dessus pour couper l'alimentation durant le transport, et elle a été tordue, et a fait levier à l'intérieur du connecteur, ce qui l'a endommagé). On voit ici que le bouclage de la masse des piles n'est plus assuré, c'est pourquoi la visualisation n'a plus fonctionnée une fois la fusée en rampe. Heureusement, les dommages étaient limités à l'affichage, le contrôle, c'est-à-dire l'allumage des différents capteurs et des séquenceurs fonctionnait toujours. Ceci a été vérifié grâce à notre télémétrie. En effet, nous envoyions l'état du séquenceur par la FSK, ce qui nous a permis de vérifier que tout était nominal.

III. Principe de fonctionnement des DEL :

Les diodes étaient alimentées sous 5V et 30mA, ce courant leur était fourni par des portes inverseuses à quatre entrées. En en plaçant deux d'entre elles en succession, on obtient le fonctionnement décrit auparavant (diode rouges et vertes allumées alternativement pour chaque alim). Ces *drivers* sont contrôlés directement par les signaux de *feedback* renvoyés par les différents boîtiers.

Photo 1 : La tuile comportant toutes les DEL.

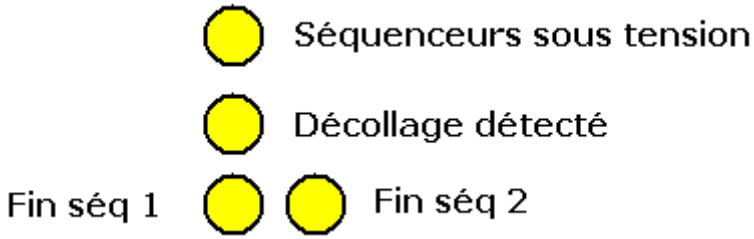


Figure 5 : DEL de contrôle de l'état du séquenceur.

IV. Monitoring USB sur PC :

Un microcontrôleur disposant d'une interface USB permet de surveiller et de contrôler les différentes alimentations depuis un PC. Des *buffers* I²C permettent au microcontrôleur de connaître l'état de chaque module et de les commander.

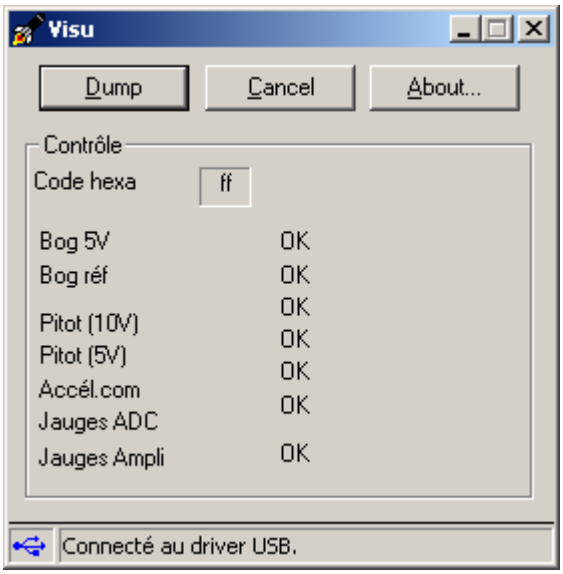


Figure 6 : Logiciel de contrôle des alimentations.

Ce programme de contrôle peut être trouvé dans le répertoire \usb\Axelle\PC\visu ; c'est un projet ATL Visual Studio.

http://www.insa-lyon.fr/Associations/ClesFacil/

Jérôme HAMM
jerome.hamm@sdbdc.org

Projet : Axelle

31 Août 2003
Version 2.0

12. Le contrôle USB.

L'interfaçage de la fusée avec le PC par USB a nécessité le développement de trois logiciels. Ce document les détaillera un par un, et devrait pouvoir aider le lecteur à réaliser un système d'interfaçage complet. Le premier module décrit, le plus simple à mettre en œuvre, et le plus classique, sera l'interface utilisateur sur PC. Ensuite sera examiné le driver sur PC également. Nous terminerons par la programmation embarquée du microcontrôleur.

I. L'interface utilisateur (sur PC sous Windows)

Cette partie est réellement la plus simple de tout le processus de développement.

A - Démarrage du driver

Pour obtenir un accès au driver, il faut utiliser la fonction `CreateFile`, et lui passer le nom du driver.

```
hAxelle = CreateFile(
        "\\\\.\\USB#Vid_1500&Pid_1500#5&e11c678&0&1#{36fc9e60-c465-11cf-8056-444553540000}",
        GENERIC_ALL,                    // Type d'accès (lecture et écriture)
        NULL,                           // Mode de partage (aucun)
        NULL,                           // Sécurité (héritage par un processus fils)
        OPEN_ALWAYS,                    // Type de création (ouvrir seulement s'il existe)
        NULL,                           // Attributs (pour la création d'un nvo fichier)
        NULL);                          // Référence pour les opérations asynchrones.
```

Code 1 : Obtension d'une référence sur le driver.

Cette fonction renvoie un *handle* de fichier sur le driver (en tout point identique à une ouverture de fichier disque, ou d'un port série par exemple). Pour obtenir plus de détails sur les paramètres, se référer à la documentation Microsoft, par exemple sur http://msdn.microsoft.com).

Le nom est généré automatiquement par le système d'exploitation (cf. : la partie sur le driver). Il est possible de retrouver le nom automatiquement, mais je n'ai pas passé de temps à chercher comment (je crois me souvenir qu'il y a un exemple de ceci dans le *DDK*[1] de Microsoft). Une fois cette référence obtenue, nous allons pouvoir travailler avec le driver et échanger des informations avec le microcontrôleur USB.

[1] Driver Development Kit pour Windows 2000.

B - Dialogue avec le microcontrôleur

On peut utiliser trois fonctions pour échanger des informations avec le driver qui relaie les informations au microcontrôleur :

- *ReadFile* : on peut utiliser la fonction ReadFile pour recevoir des informations du microcontrôleur (elle n'a pas été implémentée dans le driver d'aXelle).

```
ReadFile(
        hAxelle,                        // Référence du fichier
        pOBuffer,                       // Tampon qui contiendra les informations lues
        8,                              // Taille du tampon en octets
        &len,                           // Nombre d'octets réellement lus
        NULL);                          // Référence pour les opérations asynchrones
```

Code 2 : Obtention d'informations du driver.

- *WriteFile* : cette fonction peut être utilisé pour envoyer des informations au driver (elle n'a pas non plus été implémentée pour aXelle).

```
WriteFile(
        hAxelle,                        // Référence du fichier
        pOBuffer,                       // Tampon qui contiendra les informations lues
        8,                                  // Taille du tampon en octets
        &len,                           // Nombre d'octets réellement lus
        NULL);                          // Référence pour les opérations asynchrones.
```

Code 3 : Envoi d'informations au driver.

- *DeviceIOControl* : cette fonction permet une interaction plus avancée du programme utilisateur avec le driver. On peut notamment envoyer et recevoir des données dans la même opération. Il est également possible d'effectuer plusieurs types d'opérations grâce au code d'opération passé en paramètre.

```
DeviceIoControl(
        hAxelle,                        // Référence du fichier
        22,                                 // Type de transfert demandé
        NULL,                           // Tampon d'entrée
        0,                                  // Taille du tampon d'entrée
        pOBuffer,                       // Tampon de sortie
        100,                            // Taille du tampon de sortie
        &len,                           // Nombre d'octets retournés
        NULL);                          // Référence pour les opérations asynchrones.
```

Code 4 : Echange de données avec le driver.

C - Fin de l'utilisation du driver

Lorsque l'on a fini d'utiliser le driver (et pouvoir débrancher la fusée sans plantage), il faut le libérer, ceci se fait grâce à la fonction *CloseHandle* (c'est également effectué automatiquement par le système d'exploitation à la fin du *process* – par exemple quand on quitte l'application – qui a demandé l'accès au *driver*).

```
CloseHandle(hAxelle);           // Référence du fichier.
```

Code 5 : Libération du driver.

D - Programme de visualisation de l'état d'aXelle

Le programme de visualisation de l'état de la fusée a été réalisé sous Visual Studio, avec la bibliothèque *Windows Template Library* qui encapsule la plupart des fonctions d'interface graphique de Windows. Les points intéressants sont les suivants :

```
LRESULT CMainDlg::OnInitDialog(UINT /*uMsg*/, WPARAM /*wParam*/, LPARAM /*lParam*/, BOOL&
/*bHandled*/){

/* Fin de l'initialisation de la boîte de dialogues */

/* Obtention d'une référence sur le driver */
        hAxelle = CreateFile(
        "\\\\.\\USB#Vid_1500&Pid_1500#5&e11c678&0&1#{36fc9e60-c465-11cf-8056-444553540000}",
                GENERIC_ALL,            // Type d'accès (lecture et écriture)
                NULL,                   // Mode de partage (aucun)
                NULL,                   // Sécurité (héritage par un processus fils)
                OPEN_ALWAYS,            // Type de création (ouvrir seulement s'il existe)
                NULL,                   // Attributs (pour la création d'un nvo fichier)
                NULL);                  // Référence pour les opérations asynchrones.
/* Vérification de l'obtention de la référence */
        if(hAxelle == INVALID_HANDLE_VALUE){
/* Driver non initialisé : erreur */
        }else{
/* Driver initialisé : OK */
                SetTimer(ID_REFRESH_TIMER, 100);      // démarrage de la scrutation
        }
}
```

Code 6 : Fonction d'initialisation.

```
LRESULT CMainDlg::OnTimer(UINT uMsg, WPARAM wParam, LPARAM lParam, BOOL& bHandled){

        char pBuffer[100];              // Tampon provisoire de formattage
        DWORD len=0;                    // Nombre d'octets renvoyés par le driver
        char pOBuffer[100];             // Tampon de sortie du driver
        DeviceIoControl(
                hAxelle,                // Référence du fichier
                22,                     // Type de transfert demandé
                NULL,                   // Tampon d'entrée
                0,                      // Taille du tampon d'entrée
                pOBuffer,               // Tampon de sortie
                100,                    // Taille du tampon de sortie
                &len,                   // Nombre d'octets retournés
                NULL);                  // Référence pour les opérations asynchrones.
        pOBuffer[len] = 0;
        wsprintf(pBuffer, "%x", pOBuffer[0] & 0xff);
        SetDlgItemText(IDE_CON1, pBuffer);
/* Décodage de l'état de chaque alimentation */
        SetDlgItemText(IDE_BOG_5V, (*pOBuffer&BOG_5V?"OK":"ERREUR"));
        SetDlgItemText(IDE_BOG_REF, (*pOBuffer&BOG_REF?"OK":"ERREUR"));
        SetDlgItemText(IDE_PITOT_10V, (*pOBuffer&PITOT_10V?"OK":"ERREUR"));
        SetDlgItemText(IDE_PITOT_5V, (*pOBuffer&PITOT_5V?"OK":"ERREUR"));
        SetDlgItemText(IDE_ACCEL_COM, (*pOBuffer&ACCEL_COM?"OK":"ERREUR"));
        SetDlgItemText(IDE_JAUGES_ADC, (*pOBuffer&JAUGES_ADC?"OK":"ERREUR"));
        SetDlgItemText(IDE_JAUGES_AMPLI, (*pOBuffer&JAUGES_AMPLI?"OK":"ERREUR"));

        return 0;
}
```

Code 7 : Fonction de scrutation de l'état de la fusée.

Le principe est le suivant : à l'initialisation de la boîte de dialogue, on contacte le driver, et s'il est lancé, on scrute l'état de la fusée toutes les 100 ms. L'état est décodé et affiché dans les différentes cases prévues à cet effet.

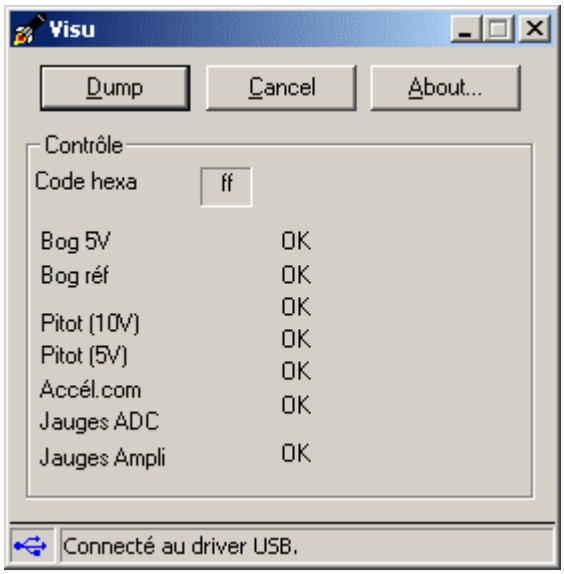


Figure 1 : L'application en action.

II. Le driver WDM[2]

A - Les outils

Les outils suivants ont été utilisés pour réaliser le driver :

- Le DDK pour Windows 2000 (il était disponible en libre téléchargement sur le site de Microsoft).
- L'outil de visualisation de messages Debug View, il permet d'intercepter les messages envoyés par les fonctions `OutputDebugString` et `DbgPrint`.

[2] Windows Driver Model

- L'explorateur d'objets WinObj, qui permet de vérifier si le driver est bien lancé et de vérifier son nom.
- L'installateur de driver qui permet de démarrer le driver sans être obligé d'écrire trop de code.
- L'outil de communication avec le driver, fiddler, permet d'envoyer des messages au driver sans écrire de code non plus (`CreateFile`, `DeviceIOControl`, `CloseHandle`).

B - Le fichier d'installation (.inf)

Pour pouvoir installer le driver, et pour que le système d'exploitation le démarrage quand on branche un périphérique, il faut l'installer dans la base de registres. Ceci est fait grâce à un utilitaire pour les générer (geninf.exe). C'est ce fichier qu'il faut utiliser lorsque le système d'exploitation demande les fichiers d'installation du périphérique. Lorsque l'on recompile, il suffit de remplacer le fichier .sys dans le répertoire c:\windows\system32\drivers, il sera utilisé au prochain démarrage ou *reset*.

C - Le fichier d'en-tête (.h)

```
extern "C" {

#include <ntddk.h>                    // Définitions classiques pour les drivers
#include <usb100.h>                   // Définitions pour le protocole USB 1.00
#include <usbdi.h>                    // Définitions d'interfaces USB
#include <usbdlib.h>          // Bibliothèque d'aide USB


NTSTATUS __stdcall DriverEntry(       // Point d'entrée dans le driver.
        IN PDRIVER_OBJECT  DriverObject,
        IN PUNICODE_STRING  RegistryPath);
}

#define CHECK(x) status = x;DbgPrint("Axelle : %s - %s", #x,
(status==STATUS_SUCCESS?"OK":"FAILED"));
                                      // Macro qui renvoie automatiquement la valeur de statut
                                      //  d'une fonction et envoi au journal.

// Variables propres à chaque instance du driver
typedef struct _DEVICE_EXTENSION{
        UNICODE_STRING wsSymbolicLinkName;    // Nom symbolique (a utiliser avec CreateFile)
        PDEVICE_OBJECT pdoTopOfStackDeviceObject;     // Pile du driver
        PUSB_DEVICE_DESCRIPTOR pUsbDeviceDescriptor;// Descripteur du périphérique
        USBD_PIPE_HANDLE hInPipe1;    // Canal de communication avec le microcontrôleur
                                      //  pour l'obtention du statut (état des alims)
        USBD_PIPE_HANDLE hInPipe2;    // Canal de communication avec le microcontrôleur
                                      //  pour le dump des données (pas terminé)
        IO_REMOVE_LOCK RemoveLock;    // Verrou pour les appels asynchrones
        unsigned long maxtransfer;    // Taille maximale des transferts
        unsigned long nNumberOfBytesTransfered;// Nombre d'octets transferrés
} * PDEVICE_EXTENSION, DEVICE_EXTENSION;

// Bloc de variables pour les fonctions asynchrones
struct _PRWCONTEXT : public _URB{
        PMDL    pMdl;
};
typedef struct _PRWCONTEXT * PRWCONTEXT, RWCONTEXT;

// Taille des transferts
#define TRANSFER_SIZE PAGE_SIZE
```

Code 8 : aXelle.h

Ce fichier *header* contient deux choses importantes. La première est l'exportation du point d'entrée du driver, et la seconde est la structure d'informations personnelles sur chaque instance du driver (initialisée lors de sa création et utilisable dans chaque fonction).

D - Le fichier source (.c)

Lors de l'initialisation du driver, il faut passer un certain nombre de pointeurs de fonctions, qui seront appelées dans le cas d'un événement correspondant. Dans chaque fonction, l'un des paramètres est un pointeur sur une structure personnalisée, qui permet entre autres de savoir de quel périphérique il s'agit (pour qui la fonction a été appelée).

```
VOID AxelleUnLoad(            // Fonction pour le déchargement du driver
                IN PDRIVER_OBJECT  DriverObject);// Pointeur sur les informations de l'instance
NTSTATUS Create(              // Fonction pour l'instanciation (branchement d'un périphérique)
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp); // Pointeur sur la pile du driver
NTSTATUS Close(               // Fonction pour le relâchement d'une référence sur le driver
                              //  (CloseHandle)
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);
NTSTATUS Read(                // Fonction pour la lecture d'informations (ReadFile)
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);
NTSTATUS Write(               // Fonction pour l'envoi d'informations (WriteFile)
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);
NTSTATUS AxelleAddDevice(     // Fonction pour l'ajout d'un périphérique
                IN PDRIVER_OBJECT  DriverObject,
                IN PDEVICE_OBJECT  PhysicalDeviceObject);
NTSTATUS AxelleDispatchDeviceControl(       // Fonction de communication (DeviceIOControl)
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);
NTSTATUS SysControlIrp(       // Fonction de communication dud système
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);
NTSTATUS AxelleDispatchPnP(   // Fonctions Plug and Play
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);
NTSTATUS PowerIrp(            // Fonctions de gestion de l'énergie
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp);

NTSTATUS FreeAll();           // Fonction de désallocation de tous les tampons de l'objet
                              //  courant

NTSTATUS IsoUsb_IrpCompletionRoutine(       // Fin d'un appel de fonction asynchrone
                IN PDEVICE_OBJECT DeviceObject,
                IN PIRP Irp,
                IN PVOID Context);
NTSTATUS IsoUsb_CallUSBD(     // Appel d'une fonction dans la pile du driver
                IN PDEVICE_OBJECT DeviceObject,
                IN PURB Urb);

NTSTATUS ReadComplete(        // Fin d'une opération de lecture asynchrone
                IN PDEVICE_OBJECT  DeviceObject,
                IN PIRP  Irp,
                IN PRWCONTEXT  Context);
```

Code 9 : Signature des différentes fonctions de CALLBACK pour le driver.

```
NTSTATUS DriverEntry(IN PDRIVER_OBJECT  DriverObject, IN PUNICODE_STRING  RegistryPath){

        DbgPrint("Axelle : DriverEntry (%s, %s)", __DATE__, __TIME__);

        DriverObject->DriverUnload = AxelleUnLoad;                          // non spécialisée
        DriverObject->DriverExtension->AddDevice = AxelleAddDevice;
        DriverObject->MajorFunction[IRP_MJ_CREATE] = Create;                // non spécialisée
        DriverObject->MajorFunction[IRP_MJ_DEVICE_CONTROL] = AxelleDispatchDeviceControl;
        DriverObject->MajorFunction[IRP_MJ_CLOSE] = Close;                  // non spécialisée
        DriverObject->MajorFunction[IRP_MJ_READ] = Read;                    // non spécialisée
        DriverObject->MajorFunction[IRP_MJ_WRITE] = Write;                  // non spécialisée
// routines for handling system PNP and power management requests
        DriverObject->MajorFunction[IRP_MJ_SYSTEM_CONTROL] = SysControlIrp;// non spécialisée
        DriverObject->MajorFunction[IRP_MJ_PNP] = AxelleDispatchPnP;
        DriverObject->MajorFunction[IRP_MJ_POWER] = PowerIrp;               // non spécialisée

        return STATUS_SUCCESS;
}
```

Code 10 : Initialisation du driver (configuration des fonctions de CALLBACK).

Les fonctions pour lesquelles est indiqué non spécialisée, correspondent à celles qui n'ont pas vraiment été implémentées (elles renvoient souvent STATUS_NOT_IMPLEMENTED, ou STATUS_SUCCESS).

1. AxelleAddDevice

Lors du branchement, et de la détection, d'un périphérique, la fonction AxelleAddDevice est appelée, c'est par elle que l'on va initialiser toutes les informations qui seront relatives à cette instance et créer le lien symbolique qui servira de nom lors de l'ouverture du driver (appel de CreateFile, cf. : Démarrage du driver)

```
NTSTATUS AxelleAddDevice(IN PDRIVER_OBJECT  DriverObject, IN PDEVICE_OBJECT
PhysicalDeviceObject){

        PDEVICE_OBJECT pDO;                                     // Informations standard sur
l'objet
        PDEVICE_EXTENSION pdeviceExtension;  // Informations personnelles sur l'objet
        NTSTATUS status = STATUS_SUCCESS;    // Valeur de retour de la onction
        ANSI_STRING sSymbolicLinkName;              // Nom du lien symbolique (à utiliser
avec CreateFile)

        DbgPrint("Axelle : AddDevice");
        CHECK(IoCreateDevice(DriverObject, sizeof(DEVICE_EXTENSION), NULL,
FILE_DEVICE_UNKNOWN, FILE_DEVICE_SECURE_OPEN , 0, &pDO));
                                                                        // Création de la
structure contenant les informations sur l'objet.
        pdeviceExtension = (PDEVICE_EXTENSION)pDO->DeviceExtension;
        CHECK(IoRegisterDeviceInterface(PhysicalDeviceObject, &GUID_CLASS_AXELLE, NULL,
&pdeviceExtension->wsSymbolicLinkName));
                                                                        // Enregistrement
de l'interface dans le système.
                                                                        //  et création du
lien symbolique
        RtlUnicodeStringToAnsiString(&sSymbolicLinkName, &pdeviceExtension-
>wsSymbolicLinkName, true);
        DbgPrint("Axelle : sSymbolicLinkName = %s", sSymbolicLinkName.Buffer);
        RtlFreeAnsiString(&sSymbolicLinkName);
        IoSetDeviceInterfaceState(&pdeviceExtension->wsSymbolicLinkName, true);
                                                                        // Signale au
système que le driver est prêt.
        pDO->Flags &= ~DO_DEVICE_INITIALIZING;                  /* ??? */
        pDO->Flags |= DO_DIRECT_IO;                                     /* ??? */
        pdeviceExtension->pdoTopOfStackDeviceObject = IoAttachDeviceToDeviceStack(pDO,
PhysicalDeviceObject);

/*
Be prepared to handle PnP IRPs for the device (such as IRP_MN_QUERY_RESOURCE_REQUIREMENTS and
IRP_MN_START_DEVICE).
*/

        return status;
}
```

Code 11 : Détection d'un nouveau périphérique.

2. AxelleDispatchPnP

```
NTSTATUS AxelleDispatchPnP(IN PDEVICE_OBJECT  DeviceObject, IN PIRP  Irp){

        PIO_STACK_LOCATION irpSp = IoGetCurrentIrpStackLocation(Irp);
        NTSTATUS status;
        NTSTATUS rStatus = STATUS_SUCCESS;
        NTSTATUS ioStatus = STATUS_SUCCESS;
        PDEVICE_EXTENSION pdeviceExtension = (PDEVICE_EXTENSION)DeviceObject->DeviceExtension;

        DbgPrint("Axelle : DispatchPnp (%x)", irpSp->MinorFunction);

        rStatus = STATUS_NOT_IMPLEMENTED;

        switch(irpSp->MinorFunction){
/*
The PnP manager sends this IRP after it has (re-)assigned hardware resources, if any,
to the device. The device may have been recently enumerated and is being started for the
first time, or the device may be restarting after being stopped for resource rebalancing.
*/
        case IRP_MN_START_DEVICE:{/* Initialisation. */}
/*
        The PnP manager uses this IRP to direct drivers to remove a device's software
        representation (device objects, and so forth). The PnP manager sends this IRP
        when a device has been removed in an orderly fashion (for example, initiated
        by a user in the Unplug or Eject Hardware program), by surprise (a user pulls
        the device from its slot without prior warning), or when the user requests to
        update driver(s).
*/
```

```
        case IRP_MN_REMOVE_DEVICE: /* Retrait du driver. */
/*
        The PnP manager uses this IRP to get a device's resource requirements list.
*/
        case IRP_MN_QUERY_RESOURCE_REQUIREMENTS:
/*
        Les autres cas non traités
*/
        default:
                rStatus = STATUS_NOT_IMPLEMENTED;
                ioStatus = STATUS_NOT_IMPLEMENTED;
        }

        Irp->IoStatus.Status = ioStatus;
        IoCompleteRequest(Irp, IO_NO_INCREMENT);

        return rStatus;
}
```

Code 12 : La fonction de traitement des appels Plug and Play.

➔ IRP_MN_START_DEVICE

La partie IRP_MN_START_DEVICE est la plus importante et la plus complexe. C'est là que l'on prend contact avec le microcontrôleur et on établit la liaison (ouverture des *Pipes* de communication).

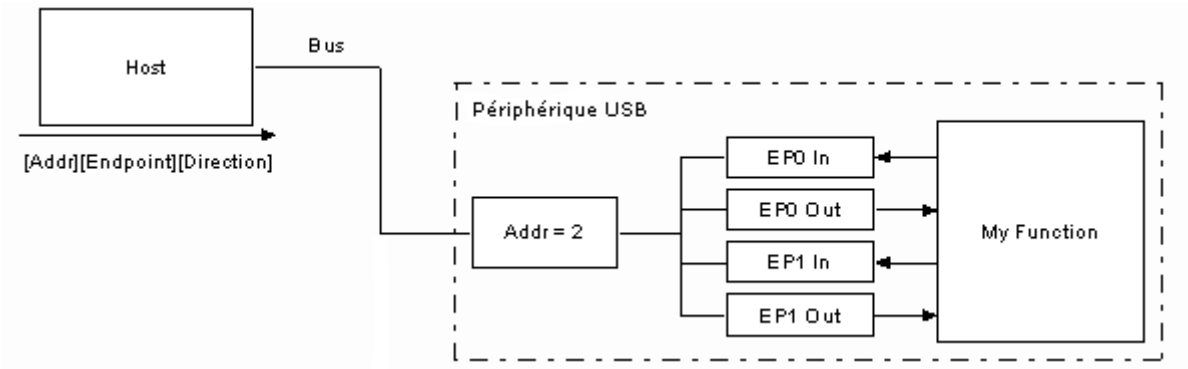


Figure 2 : Accès au composant USB.

Pour communiquer avec le périphérique, il faut établir des 'tuyaux' de communication (*communication pipes*), ils consistent en les renseignements suivants : l'adresse, l'EndPoint et la direction du transfert (ainsi que son type). Les *pipes* sont donc des liaisons logiques entre l'hôte et le périphérique. Les informations sont présentées par le périphérique de la façon suivante :

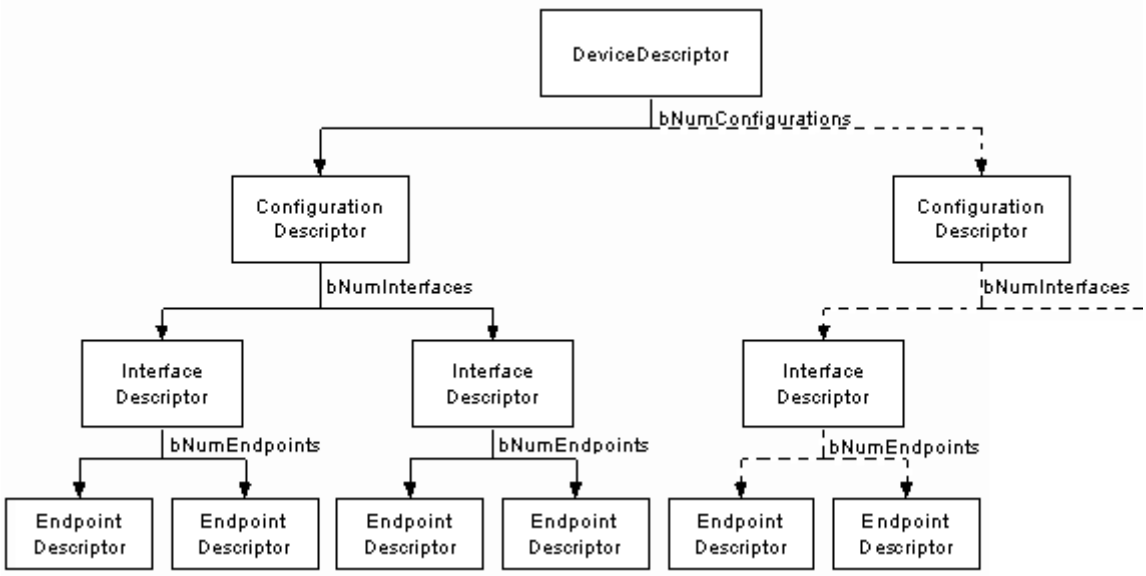


Figure 3 : Hiérarchie des descripteurs de périphérique.

Ainsi, après avoir initialisé les couches basses du driver, on demande l'ensembles des descripteurs (cf. : Figure 3 : Hiérarchie des descripteurs de périphérique.) au périphérique, et on choisit les différents éléments. Pour communiquer avec le microcontrôleur, il faut placer une requête devant les couches plus basses du driver (fournies par le système d'exploitation), grâce à une structure appelée URB[3]. Les différentes requêtes utilisent des URB spécialisés (l'URB est une union entre tous les types disponibles). Commençons donc par envoyer un premier `_URB_CONTROL_DESCRIPTOR_REQUEST`, en demandant le `USB_DEVICE_DESCRIPTOR_TYPE`, qui décrit le périphérique :

Offset	Field	Size	Value	Description
0	bLength	1	Number	Size of the Descriptor in Bytes (18 bytes)
1	bDescriptorType	1	Constant	Device Descriptor (0x01)
2	bcdUSB	2	BCD	USB Specification Number which device complies too.
4	bDeviceClass	1	Class	Class Code (Assigned by USB Org) : if(0) each interface specifies its own class code else if(0xFF) the class code is vendor specified. else field is valid Class Code.
5	bDeviceSubClass	1	SubClass	Subclass Code (Assigned by USB Org)
6	bDeviceProtocol	1	Protocol	Protocol Code (Assigned by USB Org)
7	bMaxPacketSize	1	Number	Maximum Packet Size for Zero Endpoint. Valid Sizes are 8, 16, 32, 64
8	idVendor	2	ID	Vendor ID (Assigned by USB Org)
10	idProduct	2	ID	Product ID (Assigned by Manufacturer)
12	bcdDevice	2	BCD	Device Release Number
14	iManufacturer	1	Index	Index of Manufacturer String Descriptor
15	iProduct	1	Index	Index of Product String Descriptor
16	iSerialNumber	1	Index	Index of Serial Number String Descriptor
17	bNumConfigurations	1	Integer	Number of Possible Configurations

[3] USB Request Block

Table 1 : Descripteur de périphérique (device descriptor).

Ensuite, nous envoyons une autre demande, cette fois-ci de type `USB_CONFIGURATION_DESCRIPTOR_TYPE` qui nous donnera la taille totale de tous les descripteurs. Après avoir alloué de la mémoire, il faut envoyer une deuxième fois cette requête, pour obtenir la totalité des descripteurs. Les informations sont obtenues sous la forme présentée dans la Figure 3 : Hiérarchie des descripteurs de périphérique. Il faut ensuite naviguer dans cette structure pour y trouver des informations intéressantes qui nous permettrons de configurer la cible ; cette opération de parcours est facilitée par la fonction `USBD_ParseConfigurationDescriptorEx`. On sélectionne les interfaces désirées, et on envoie une requête `USB_REQUEST_SET_CONFIGURATION`, qui peut être créée automatiquement grâce à la fonction `USBD_CreateConfigurationRequestEx`. On peut récupérer les *pipes*, juste après avoir transmis cette requête, et quitter la fonction avec un acquittement.

Voici les autres fonctions qu'il faut traiter pour que le périphérique fonctionne, leur implémentation est triviale :

- ➔ **IRP_MN_REMOVE_DEVICE**
- ➔ **IRP_MN_QUERY_RESOURCE_DEVICE**

3. AxelleDispatchDeviceControl

La dernière fonction sert réellement à dialoguer avec le programme utilisateur. Deux commandes sont traitées, 22 et 21. 22 renvoie l'état de la fusée, 21 était prévu pour *dumper* la mémoire de vol, mais je n'ai pas eu le temps de la terminer.

```
NTSTATUS AxelleDispatchDeviceControl(IN PDEVICE_OBJECT  DeviceObject, IN PIRP  Irp){

        PIO_STACK_LOCATION irpSp = IoGetCurrentIrpStackLocation(Irp);
        NTSTATUS rStatus = STATUS_SUCCESS, status;
        PDEVICE_EXTENSION pdeviceExtension = (PDEVICE_EXTENSION)DeviceObject->DeviceExtension;

        DbgPrint("Axelle : DeviceIoControl (%x)",
                irpSp->Parameters.DeviceIoControl.IoControlCode);

        switch(irpSp->Parameters.DeviceIoControl.IoControlCode){
/*
        Renvoie le statut de la fusée
*/
        case 22:{
/*
        Récupère le buffer fourni par l'utilisateur.
*/
                DWORD length = (Irp->MdlAddress?MmGetMdlByteCount(Irp->MdlAddress):0);
                if(!length){
                        rStatus = STATUS_SUCCESS;
                        break;
                }

/*
        Alloue un buffer pour le transfert.
*/
                PRWCONTEXT pRwctx=(PRWCONTEXT)ExAllocatePool(NonPagedPool, sizeof(RWCONTEXT));

                ULONG_PTR va = (ULONG_PTR) MmGetMdlVirtualAddress(Irp->MdlAddress);
                pRwctx->pMdl = IoAllocateMdl((PVOID)va, pdeviceExtension->maxtransfer,
                        false, false, NULL);
                if(!pRwctx->pMdl){
                        DbgPrint("Axelle : Peux pas allouer le MDL !");
                        ExFreePool(pRwctx);
                        rStatus = STATUS_INSUFFICIENT_RESOURCES;
                        break;
                }

                DbgPrint("Axelle : pMdl = 0x%x (IRQL = %d)", pRwctx->pMdl, KeGetCurrentIrql());
                if(!Irp->MdlAddress)
                        break;

/*
        Prépare la réception des données.
*/
                DWORD chunklen=(length<pdeviceExtension->maxtransfer?
```

```
                                        length:pdeviceExtension->maxtransfer);
                IoBuildPartialMdl(Irp->MdlAddress, pRwctx->pMdl, (PVOID)va, chunklen);
                MmMapLockedPages(pRwctx->pMdl, KernelMode);
                UsbBuildInterruptOrBulkTransferRequest(
                        pRwctx,
                        sizeof(struct _URB_BULK_OR_INTERRUPT_TRANSFER),
                        pdeviceExtension->hInPipe1,
                        NULL,
                        pRwctx->pMdl,
                        chunklen,
                        USBD_TRANSFER_DIRECTION_IN | USBD_SHORT_TRANSFER_OK,
                        NULL);

                pdeviceExtension->nNumberOfBytesTransfered = 0;

                irpSp = IoGetNextIrpStackLocation(Irp);
                irpSp->MajorFunction = IRP_MJ_INTERNAL_DEVICE_CONTROL;
                irpSp->Parameters.Others.Argument1 = pRwctx;
                irpSp->Parameters.DeviceIoControl.IoControlCode=IOCTL_INTERNAL_USB_SUBMIT_URB;

/*
        Met en place la routine de fin de transfert et lance la requête de données.
*/
                IoSetCompletionRoutine(Irp, (PIO_COMPLETION_ROUTINE)ReadComplete,
                        pRwctx, true, true, true);
                IoMarkIrpPending(Irp);// <-- ???
                IoCallDriver(pdeviceExtension->pdoTopOfStackDeviceObject, Irp);

                return  STATUS_PENDING;
                break;}

/*
        Termine la fonction.
*/
        Irp->IoStatus.Status = rStatus;
        IoCompleteRequest(Irp, IO_NO_INCREMENT);

        return STATUS_SUCCESS;
}
```

Code 13 : La fonction AxelleDispatchDeviceControl.

III. Le microcontrôleur

La troisième partie de la liaison USB est embarquée dans le microcontrôleur se trouvant dans la fusée. Le modèle utilisé est fabriqué par TI, c'est un TUSB3410 ; il dispose, en plus de son interface USB (esclave), d'une interface I²C, ce qui est très pratique, car il ne dispose que de 4 pattes d'entrée/sortie (GPIO), on peut donc, comme nous l'avons fait sur aXelle, lui adjoindre des étendeurs de ports pour en augmenter le nombre. Accessoirement, cette possibilité n'a pas été utilisée pour notre projet, il dispose d'une interface RS-232 (sorties aux niveaux TTL), il est donc très facile d'en faire une interface USB/RS-232, TI fournit même le *driver* …

L'architecture du programme est la suivante :

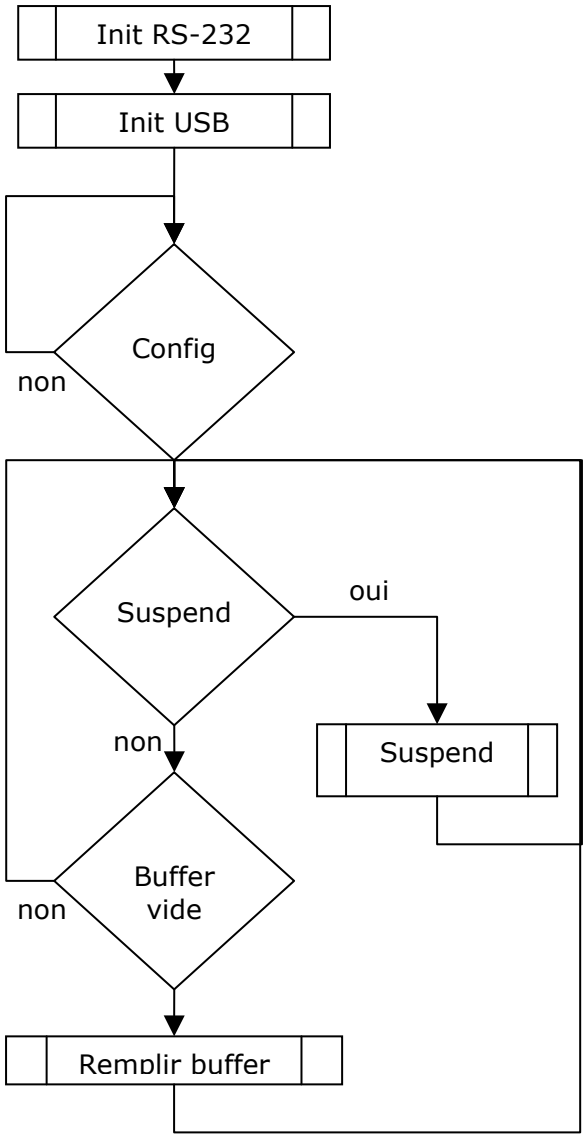


Figure 4 : fonction main.

Cette fonction est constituée d'une boucle sans fin, à la suite d'une phase d'initialisation.

On commence par initialiser une sortie série (émulée par une bibliothèque de l'environnement sur une GPIO).

On configure ensuite l'adresse USB du microcontrôleur à 0, ce qui signifie qu'il vient de démarrer et que l'hôte doit lui fournir une adresse de fonctionnement. Il faut aussi configurer les *EndPoints* qui seront utilisés pour effectuer les transferts (remarque : l'EndPoint0 est toujours défini pour les échanges de configuration du périphérique – c'est une liaison de type *Control*).

Ensuite, on attend que l'hôte initialise le périphérique (réponse à une interruption du *hardware* USB).

La boucle infinie a deux étapes la première consiste à vérifier si l'hôte nous a demandé une mise en veille (par exemple pour des raisons d'économie d'énergie), et si oui, passe dans ce mode.

Si on est actif, on vérifie si le *buffer* d'émission est vide, si c'est le cas, on le remplit (avec les informations de l'état de la fusée et on signale qu'il est plein et doit être transféré â l'hôte).

Une fois la phase d'initialisation passée, on active les interruptions, et ce sont les vecteurs d'IT qui réalisent les échanges avec l'hôte. La structure des interruptions du 8052 est la suivante :

INTERRUPT SOURCE	DESCRIPTION	START ADDRESS	COMMENTS
ES	UART interrupt	0023H	
ET1	Timer-1 interrupt	001BH	
EX1	External interrupt-1	0013H	
ET0	Timer-0 interrupt	000BH	
EX0	External interrupt-0	0003H	Used for all internal peripherals
Reset		0000H	

Table 2: Mapping des interruptions du 8052

L'interruption que nous utiliserons est la EX0, et nous connaîtrons la source plus précisément grâce au registre VECINT (FF92h). L'acquittement de l'interruption se fait en écrivant 0 dans ce même registre de vecteur d'interruption. Les interruptions traitées sont les suivantes :

- VECINT_SETUP_PACKET_RECEIVED
- VECINT_RESR_INTERRUPT
- VECINT_SUSR_INTERRUPT
- VECINT_RSTR_INTERRUPT
- VECINT_RWUP_INTERRUPT
- VECINT_INPUT_ENDPOINT0

```
void SetupPacketInterruptHandler(void){

        tDEVICE_REQUEST * pSetup = pbEP0_SETUP_ADDRESS;
        tDEVICE_DESCRIPTOR * pDesc = pbOEP0_BUFFER_ADDRESS;

        // NAK both input and output endpoints
        tEndPoint0DescriptorBlock.bIEPBCNT = EPBCT_NAK;
        tEndPoint0DescriptorBlock.bOEPBCNT = EPBCT_NAK;

usbSetupOverwrite:

        bUSBCTL |= USBCTL_SIR;

        rs232PutHexString(sizeof(tDEVICE_REQUEST), (char *)pSetup);
        rs232PutString(" - ");


        if((pSetup->bmRequestType & USB_REQ_TYPE_INPUT) == USB_REQ_TYPE_INPUT){
                bUSBCTL |= USBCTL_DIR;
                rs232PutString("DIR=1(host<-dev)\r\n");
        }else{
                bUSBCTL &= ~USBCTL_DIR;
                rs232PutString("DIR=0(host->dev)\r\n");
        }

        ProcessUSBPacket();

        if(bUSBSTA & USBSTA_STPOW){
                bUSBSTA = USBSTA_STPOW;
                rs232PutString("-- usbSetupOverwrite\r\n");
                goto usbSetupOverwrite;
        }
}
```

Code 14 : *Handler* d'interruption créant les paquets de configuration.

```
void IEP0InterruptHandler(void){
        SendNextPacketOnEIP0();
}
```

Code 15 : *Handler* appelé lorsqu'un paquet a été envoyé à l'hôte.

```
void SendNextPacketOnEIP0(void){

        tEndPoint0DescriptorBlock.bOEPBCNT = 0x00;
        if(uIEP0BufferSize == NO_MORE_DATA){
                tEndPoint0DescriptorBlock.bIEPCNFG |= EPCNF_STALL; // no more data
```

```
                rs232PutString("NO_MORE_DATA\r\n");
        }else{
                unsigned int toSend;
                unsigned int cnt;

                if(uIEP0BufferSize<=8){
                        toSend = uIEP0BufferSize;
                        uIEP0BufferSize = NO_MORE_DATA;
                }else{
                        toSend = 8;
                        uIEP0BufferSize -= 8;
                }

                rs232PutString("Send packet on IEP0");
                rs232PutString(" : ");
                if(toSend)
                        rs232PutHexString(toSend, (char *)pIEP0Buffer);
                else
                        rs232PutString("<NULL>");
                rs232PutString("\r\n");
                for(cnt=0;cnt<toSend;cnt++){
                        abIEP0Buffer[cnt] = *pIEP0Buffer++;
                }
                tEndPoint0DescriptorBlock.bIEPBCNT = toSend;
        }
}
```

Code 16 : Découpage des trames en paquets.

```
void SendOnEIP0(char * pDesc, unsigned int len){

        unsigned int cnt;

        uIEP0BufferSize = len;
        pIEP0Buffer = abDescriptor;
        for(cnt=0;cnt<len;cnt++){
                abDescriptor[cnt]=*pDesc++;
        }
        SendNextPacketOnEIP0();
        }
```

Code 17 : Fonction d'envoi d'une trame vers l'hôte.

Le deuxième aspect important est constitué par les différents descripteurs, qui permettent de reconnaître le périphérique et sa configuration.

```
tDEVICE_DESCRIPTOR code sUSBDeviceDescriptor = {
        sizeof(tDEVICE_DESCRIPTOR),   // bLength
        DESC_TYPE_DEVICE,       // bDescriptorType - Device Descriptor (0x01)
        0x1001,                 // bcdUSB - USB Specification Number which device complies to
//      0,                      // bDeviceClass - Class Code (Assigned by USB Org)
        0xff,                   //If equal to Zero, each interface specifies its own class code
                                //If equal to 0xFF, the class code is vendor specified
                                //Otherwise field is valid Class Code
        0,                      // bDeviceSubClass - Subclass Code (Assigned by USB Org)
        0,                      // bDeviceProtocol - Protocol Code (Assigned by USB Org)
        8,                      // bMaxPacketSize - Maximum Packet Size for Zero Endpoint.
                                //                          Valid Sizes are 8, 16, 32, 64
                        // Tempo, je sais pas trop ...
        0x15,                   // idVendor - Vendor ID (Assigned by USB Org)
        0x15,                   // idProduct - Product ID (Assigned by Manufacturer)
        0x0000,                 // bcdDevice - Device Release Number
        1,                      // iManufacturer - Index of Manufacturer String Descriptor
        2,                      // iProduct - Index of Product String Descriptor
        3,                      // iSerialNumber - Index of Serial Number String Descriptor
        1                       // bNumConfigurations - Number of Possible Configurations
};
```

Code 18 : Descripteur de périphérique.

```
tCONFIG_DESCRIPTOR code sUSBConfigurationDescriptor = {
        sizeof(tCONFIG_DESCRIPTOR),   // bLength
        DESC_TYPE_CONFIG,             // bDescriptorType - Configuration Descriptor (0x02)
        CHANGE_ENDIANITY(sizeof(tCONFIG_DESCRIPTOR)  // wTotalLength - Total length in bytes
        +sizeof(tINTERFACE_DESCRIPTOR) // of data returned. - ATTENTION
        +sizeof(tENDPOINT_DESCRIPTOR)), // L'USB a une endianity differente de celle du µC.
        1,                      // bNumInterfaces - Number of Interfaces
        1,// bConfigurationValue - Value to use as an argument to select this configuration
        4,      // iConfiguration - Index of String Descriptor describing this configuration
        0x80,                   // bmAttributes - D7 Reserved, set to 1. (USB 1.0 Bus Powered)
                                //                D6 Self Powered
                                //                D5 Remote Wakeup
```

```
                        //                          D4..0 Reserved, set to 0.
        100             // bMaxPower - Maximum Power Consumption in 2mA units
                    // 200 mA pour l'instant
};
```

Code 19 : Descripteur des configurations du périphérique.

```
tINTERFACE_DESCRIPTOR code sUSBInterfaceDescriptor = {
        sizeof(tINTERFACE_DESCRIPTOR),// bLength - Size of Descriptor in Bytes (9 Bytes)
        DESC_TYPE_INTERFACE,    // bDescriptorType - Interface Descriptor (0x04)
        0,                      // bInterfaceNumber - Number of Interface
        0,                      // bAlternateSetting - Value used to select alternative setting
        1,                      // bNumEndpoints - Number of Endpoints used for this interface
        0xff,                   // bInterfaceClass - Class Code (Assigned by USB Org)
        0,                      // bInterfaceSubClass - Subclass Code (Assigned by USB Org)
        0,                      // bInterfaceProtocol - Protocol Code (Assigned by USB Org)
        5               // iInterface - Index of String Descriptor Describing this interface
};
```

Code 20 : Descripteur d'interface.

```
tENDPOINT_DESCRIPTOR code sUSBEndPointDescriptor1 = {
        sizeof(tENDPOINT_DESCRIPTOR),// bLength - Size of Descriptor in Bytes (7 bytes)
        DESC_TYPE_ENDPOINT,     // bDescriptorType - Endpoint Descriptor (0x05)
        0x81,                   // bEndpointAddress - Endpoint Address
                        //      Bits 0..3b Endpoint Number.
                        //      Bits 4..6b Reserved. Set to Zero
                        //      Bits 7 Direction 0 = Out, 1 = In (Ignored for Control Endpoints)
                        //              Always from host's point of view.
        EP_DESC_ATTR_TYPE_BULK,// bmAttributes - Bits 0..1 Transfer Type
                                //      00 = Control
                                //      01 = Isochronous
                                //      10 = Bulk
                                //      11 = Interrupt
                                //      Bits 2..7 are reserved. If Isochronous endpoint,
                                //      Bits 3..2 = Synchronisation Type (Iso Mode)
                                //      00 = No Synchonisation
                                //      01 = Asynchronous
                                //      10 = Adaptive
                                //      11 = Synchronous
                                //      Bits 5..4 = Usage Type (Iso Mode)
                                //      00 = Data Endpoint
                                //      01 = Feedback Endpoint
                                //      10 = Explicit Feedback Data Endpoint
                                //      11 = Reserved
        CHANGE_ENDIANITY(0x40),         // wMaxPacketSize - Maximum Packet Size this endpoint is
capable of sending or receiving
        0// bInterval - Interval for polling endpoint data transfers. Value in frame counts.
Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range from 1 to
255 for interrupt endpoints.
};
```

Code 21 : Descripteur du point de raccordement 1.

```
tENDPOINT_DESCRIPTOR code sUSBEndPointDescriptor2 = {
        sizeof(tENDPOINT_DESCRIPTOR),// bLength - Size of Descriptor in Bytes (7 bytes)
        DESC_TYPE_ENDPOINT,     // bDescriptorType - Endpoint Descriptor (0x05)
        0x82,                   // bEndpointAddress - Endpoint Address
                        //Bits 0..3b Endpoint Number.
                        //Bits 4..6b Reserved. Set to Zero
                        //Bits 7 Direction 0 = Out, 1 = In (Ignored for Control Endpoints)
                        //      Always from host's point of view.
        EP_DESC_ATTR_TYPE_BULK,         // bmAttributes - Bits 0..1 Transfer Type
                                        //                      00 = Control
                                        //      01 = Isochronous
                                        //      10 = Bulk
                                        //      11 = Interrupt
                                        //      Bits 2..7 are reserved. If Isochronous endpoint,
                                        //      Bits 3..2 = Synchronisation Type (Iso Mode)
                                        //      00 = No Synchonisation
                                        //      01 = Asynchronous
                                        //      10 = Adaptive
                                        //      11 = Synchronous
                                        //      Bits 5..4 = Usage Type (Iso Mode)
                                        //      00 = Data Endpoint
                                        //      01 = Feedback Endpoint
                                        //      10 = Explicit Feedback Data Endpoint
                                        //                      11 = Reserved
        CHANGE_ENDIANITY(0x40),         // wMaxPacketSize - Maximum Packet Size this endpoint is
capable of sending or receiving
```

```
        0       // bInterval - Interval for polling endpoint data transfers. Value in frame
counts. Ignored for Bulk & Control Endpoints. Isochronous must equal 1 and field may range
from 1 to 255 for interrupt endpoints.
};
```

Code 22 : Descripteur du point de raccordement 2.

IV. Le matériel

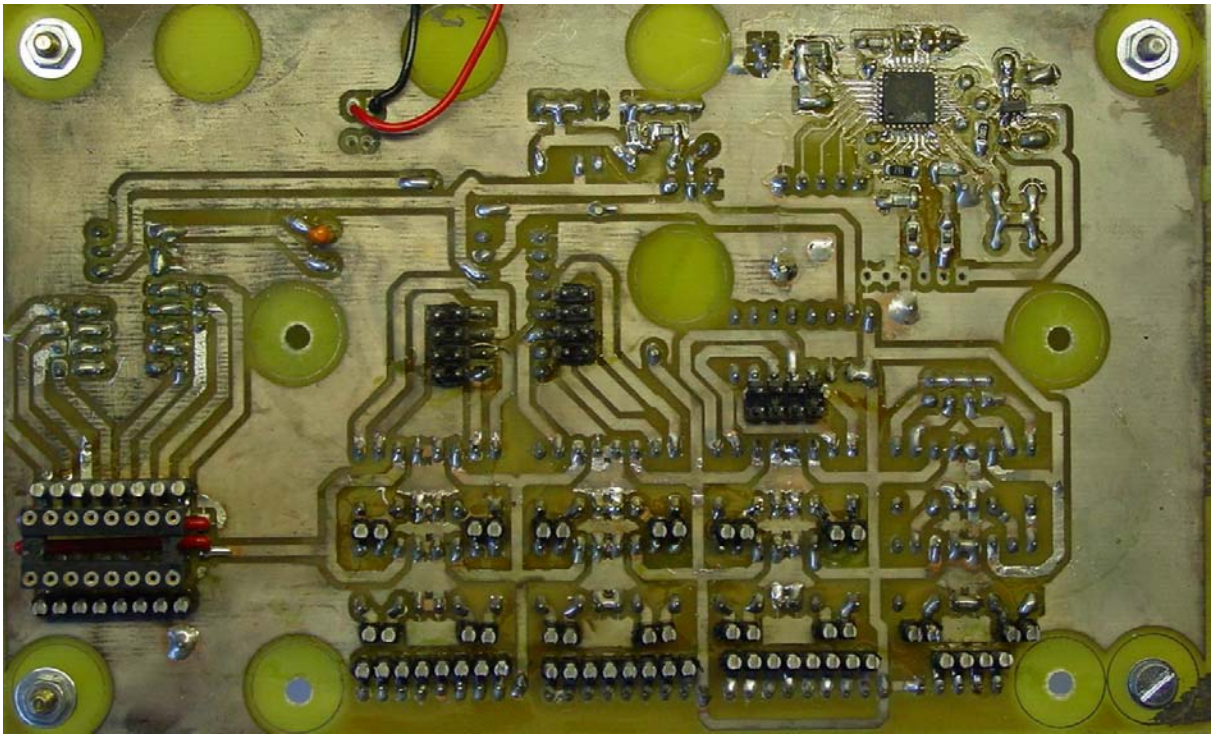


Photo 1 : Carte visu vue du dessous.

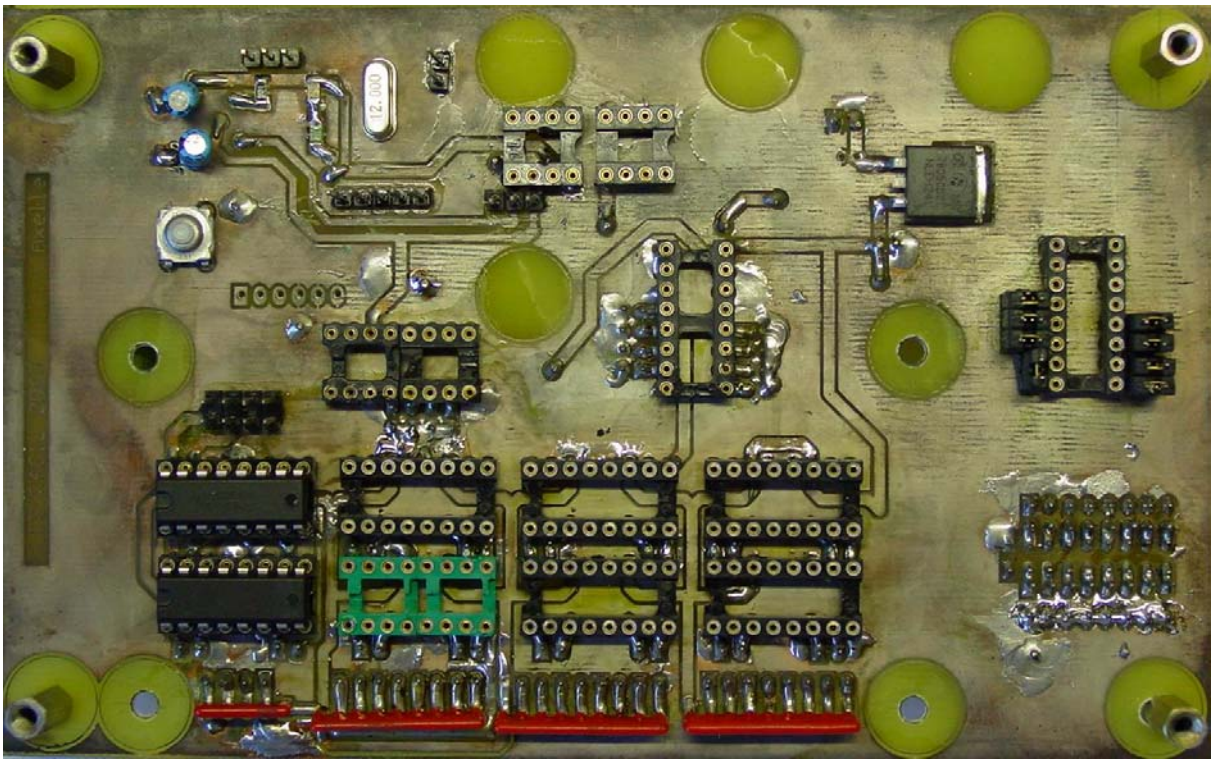


Photo 2 : Carte visu vue du dessus.

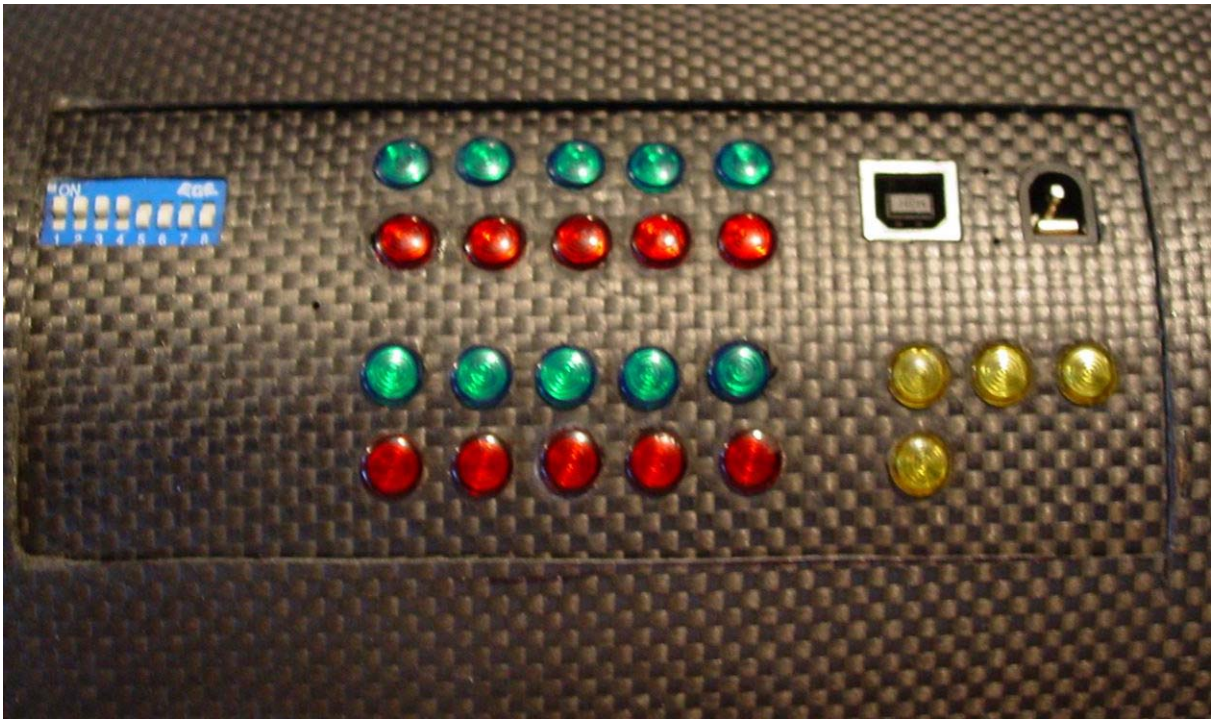


Photo 3 : Tuile (fixée sur la carte visu).

Photo 4 : Le TUSB3410, son alim et ses capas de découplage.

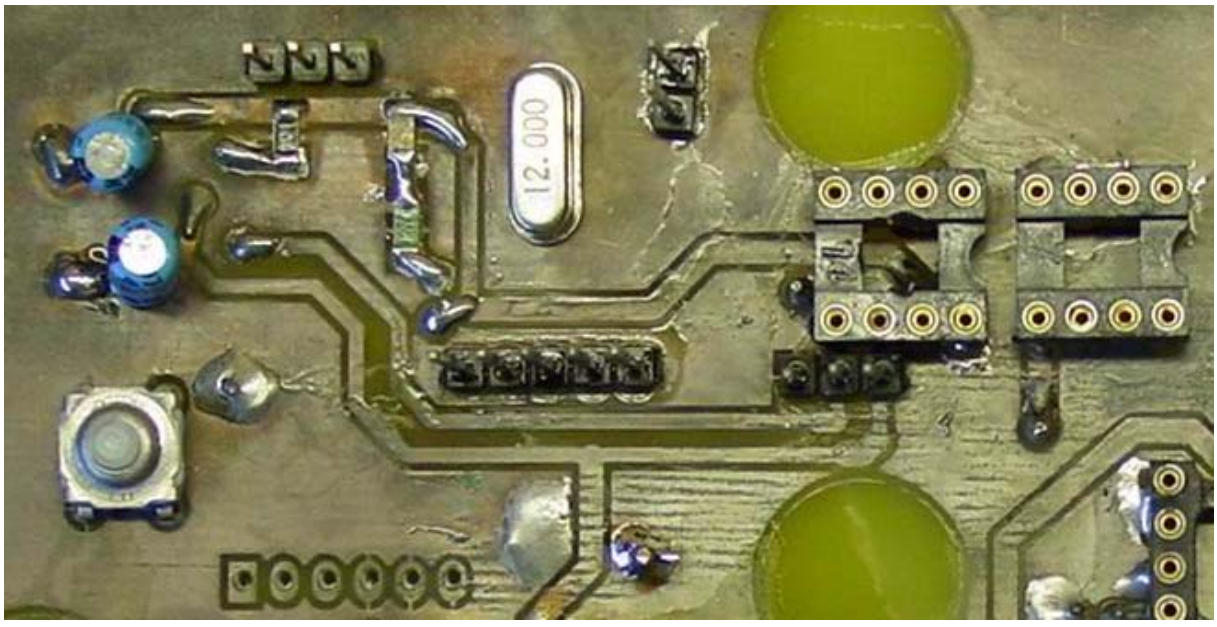


Photo 5 : Le quartz, le reset, les I/O et les supports pour les E²PROM.

V. Conclusion

Le développement d'un périphérique n'est pas aisé, d'autant plus qu'il est relativement difficile d'espionner le lien physique, car les trames qui circulent sont assez complexes (il faudrait un analyseur logique, ce qui est très onéreux dans le cas de l'USB). Il est néanmoins possible de s'en sortir grâce à des outils simples assez largement disponibles. Une technique que j'ai beaucoup utilisée a été de prendre du code existant et de l'instrumenter pour comprendre ce qu'il s'y passe - c'est cette démarche que je conseille très fortement à quiconque veut se lancer dans l'interfaçage d'un périphérique USB.

Le mot final sera que même si nous ne nous en sommes pas trop servis, cette interface a été très intéressante à réaliser, et ouvre des portes intéressantes (notamment en ce qui concerne les nouveaux portables qui ne sont plus équipés des ports de communication traditionnels). Le produit final, le jour du lancement, était un ensemble fonctionnel (c'est-à-dire que l'on avait bien les mêmes informations d'état des alimentations de la fusée que celle donnée par les DEL), même s'il manquait de robustesse (notamment au niveau du driver du côté du PC, qui était assez instable).

Il est possible que je fasse vivre ce document, et que j'y adjoigne quelques outils qui pourraient grandement faciliter la vie de ceux qui voudraient utiliser cette même interface. Tout dépendra comme toujours du temps que je peux/veux y consacrer et de l'intérêt que les autres portent à l'affaire.

http://www.insa-lyon.fr/Associations/ClesFacil/

Nicolas Chaléroux
nico@sdbdc.com

Projet : Axelle

22 Avril 2003
Version 1

13. Système d'éjection du ralentisseur

Table des mises à jour du document

Version	Date	Objet de la mise à jour	Auteur
1	22/04/03	Création du document	Nicolas Chaléroux

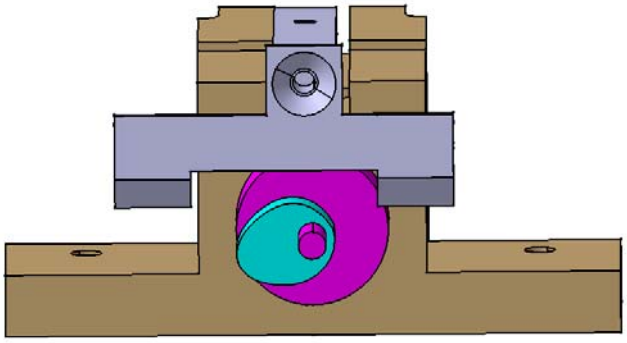
Sommaire

1. Principe

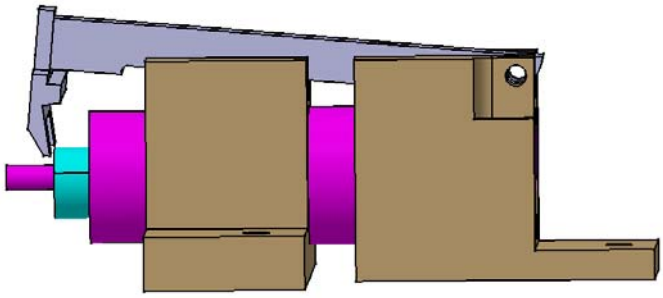
Cette année encore le CLES-FACIL reste fidèle à un système d'ouverture qui une fois activé répète le même mouvement sans arrêt. L'idée et le dimensionnement de ce système nous viennent de fusées antérieures : Pauline, PSO, S Phoenix et Phoenix. Toutes ces fusées ont effectuées un vol nominal.

L'ouverture de la case parachute et la séparation du module s'effectuent sur ce même principe d'éjection latérale. Il y a donc deux réalisations indépendantes de ces plans dans Axelle. Nous souhaitons ainsi doubler nos chances de récupérer la mémoire de la fusée (bien qu'il y ait une télémesure) : soit module s'éjecte, soit le parachute fonctionne, soit les deux, soit ☹.

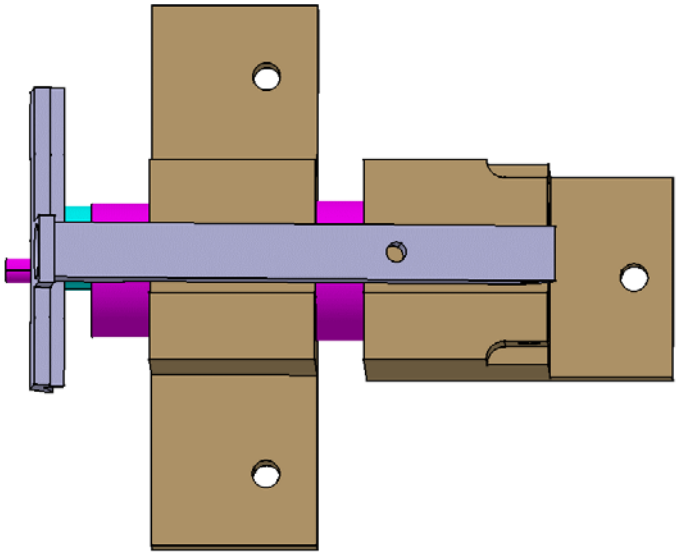
2. Vue isométrique de face



3. Vue isométrique de coté



4. Vue isométrique de dessus



CLES-FACIL

http://www.insa-lyon.fr/Associations/ClesFacil/

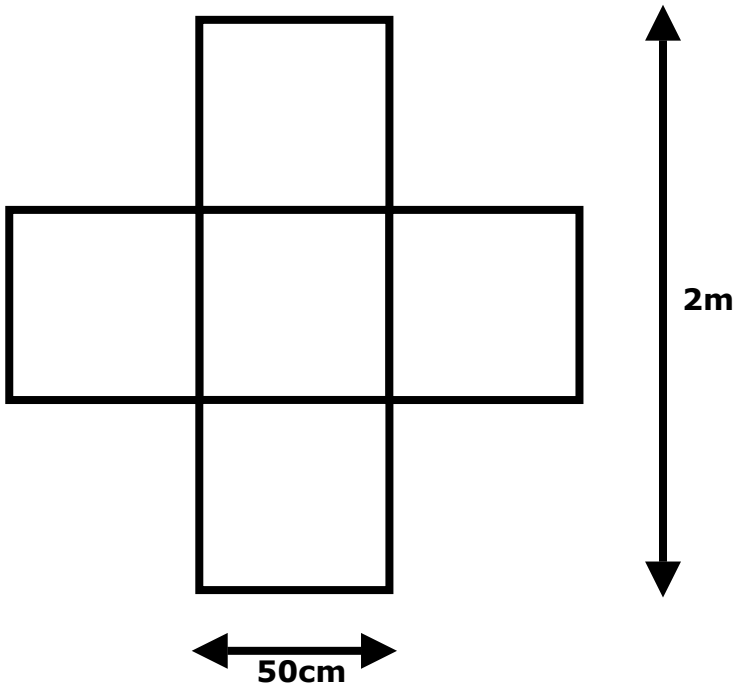
Louise Pontal & Natacha Pellet.
Louise.pontal@wanadoo.fr
pelletnata@minitel.net

Projet : Axelle

28 avril 2003
Version 1

14. Dimensionnement du système de récupération.

Le système de récupération de la fusée est un parachute dimensionné pour permettre une descente à une vitesse de 10m.s^{-1}.



Le système de récupération du module sera une banderole à poches de forme cylindrique (10cm de diamètre) avec 10 poches (15cm×15cm). Il est très difficile de calculer la vitesse exacte de descente sous « parachute » avec ce genre de système. De plus, il est important pour que notre GPS puisse se relocaliser durant la descente (mini 50sec). Ceci nous a amené à dimensionner cette banderole pour une descente à approximativement 5m.s^{-1} soit la limite autorisée !

Une demande de dérogation avait été très fortement envisagée lors de la définition de projet puis abandonnée.

CLES-FACIL

http://www.insa-lyon.fr/Associations/ClesFacil/

L'équipe méca du CLES – FACIL.
cles-facil@insa-lyon.fr

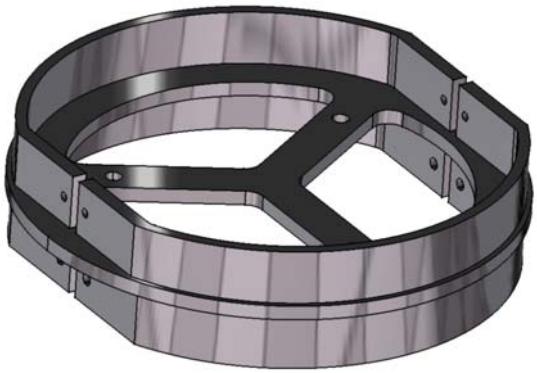
Projet : Axelle

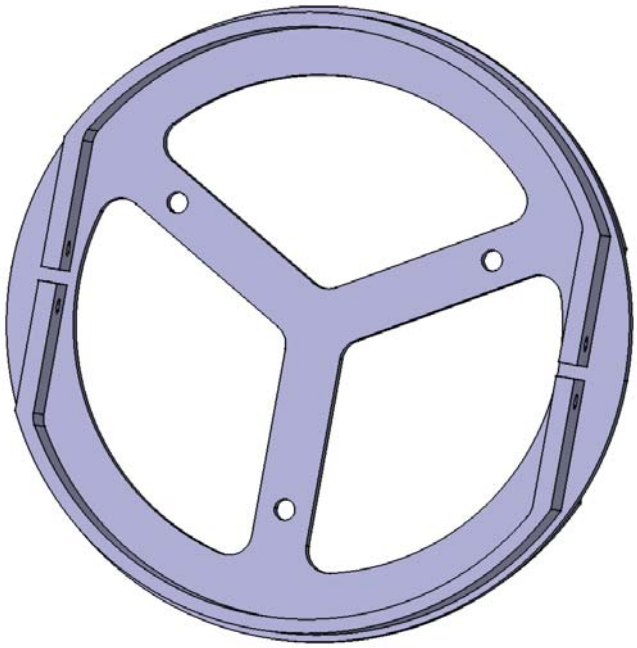
31 Août 2003
Version 1.0

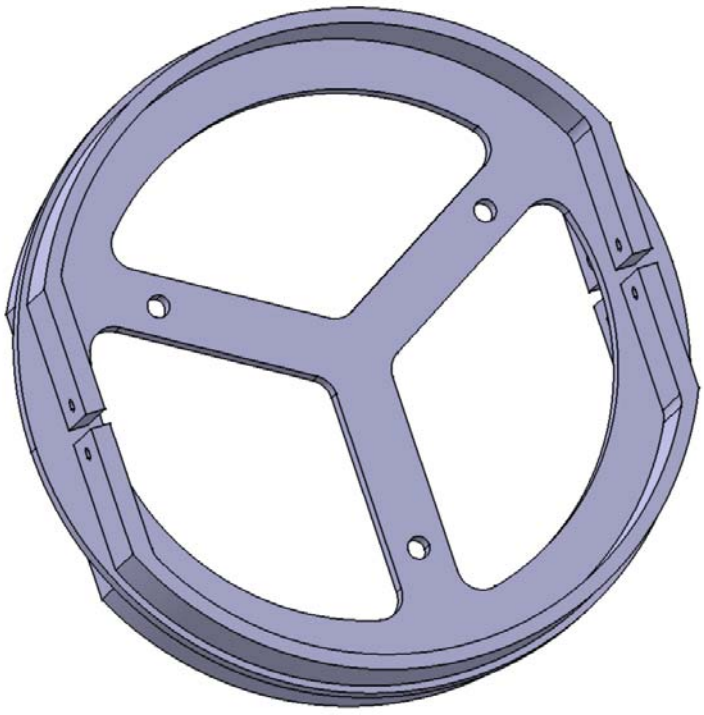
15. Dossier méca.

Les pages qui suivent regroupent un certain nombre de plans et de vues de la mécanique de la fusée : Bagues, et, particularité de ce projet, structure du module largué en vol.

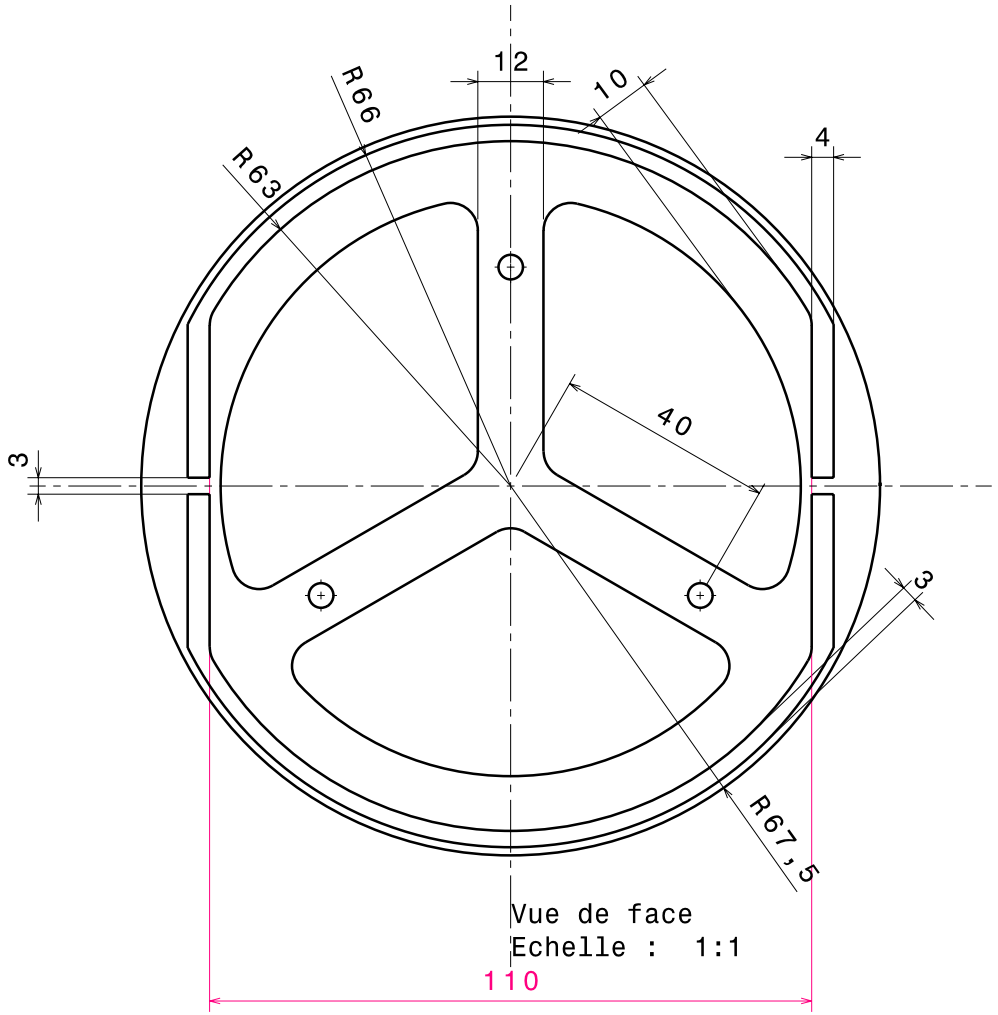
BAGUE CONE



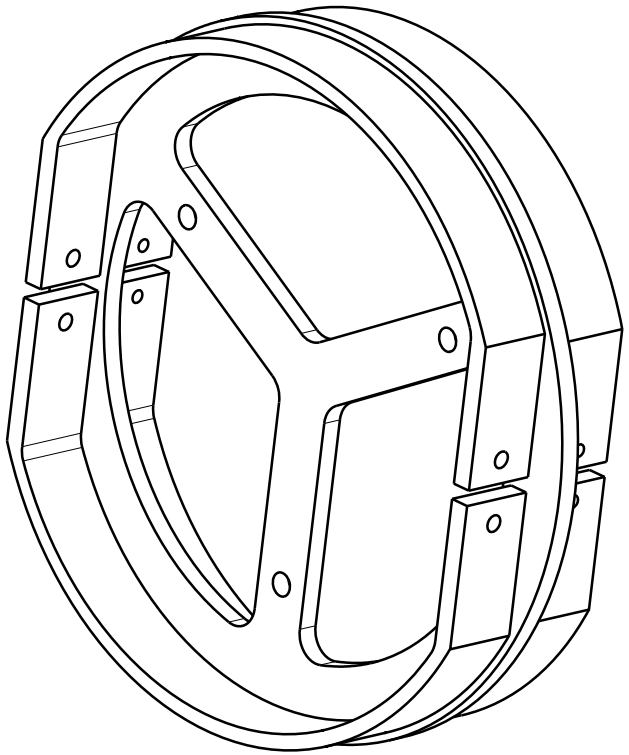




Les rayons non précisés sont de 5 mm

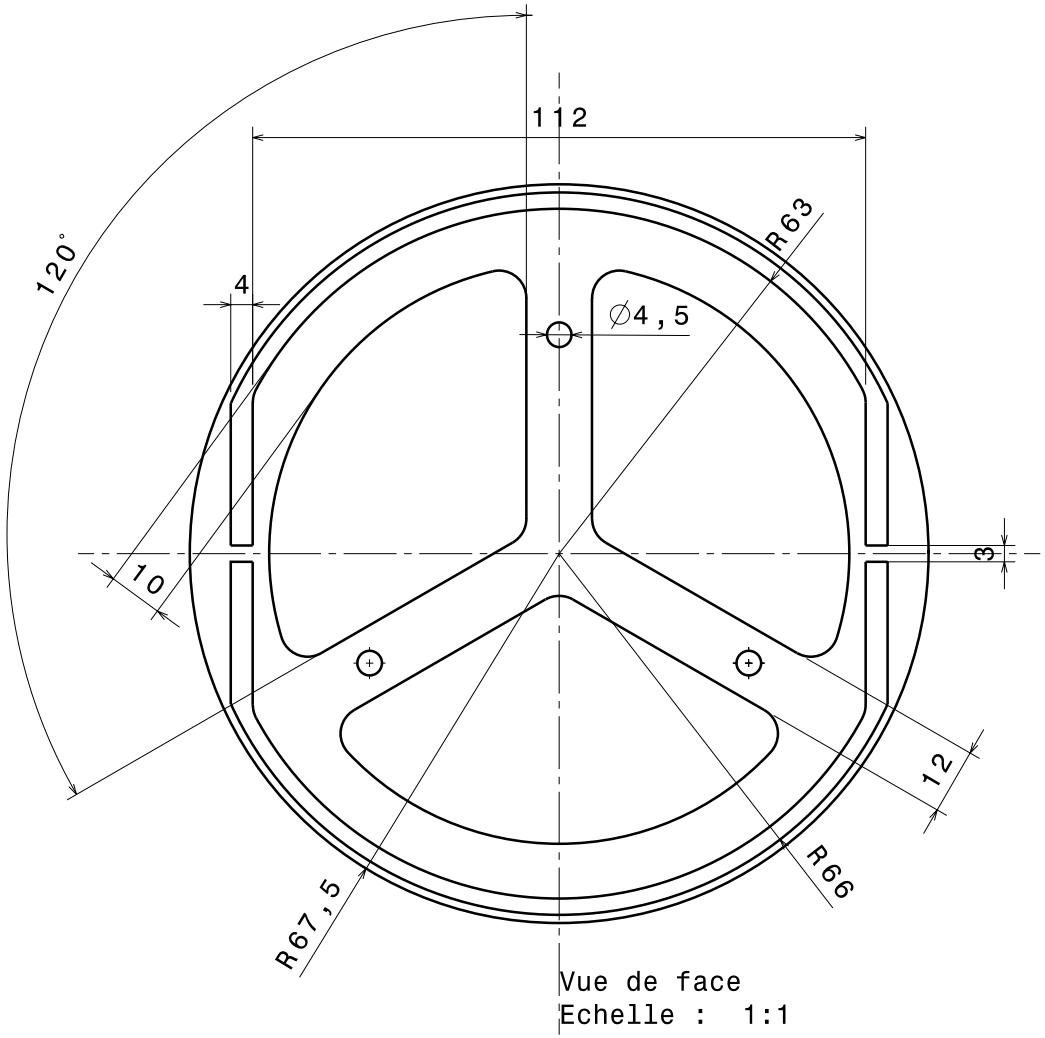


Vue de face
Echelle : 1:1

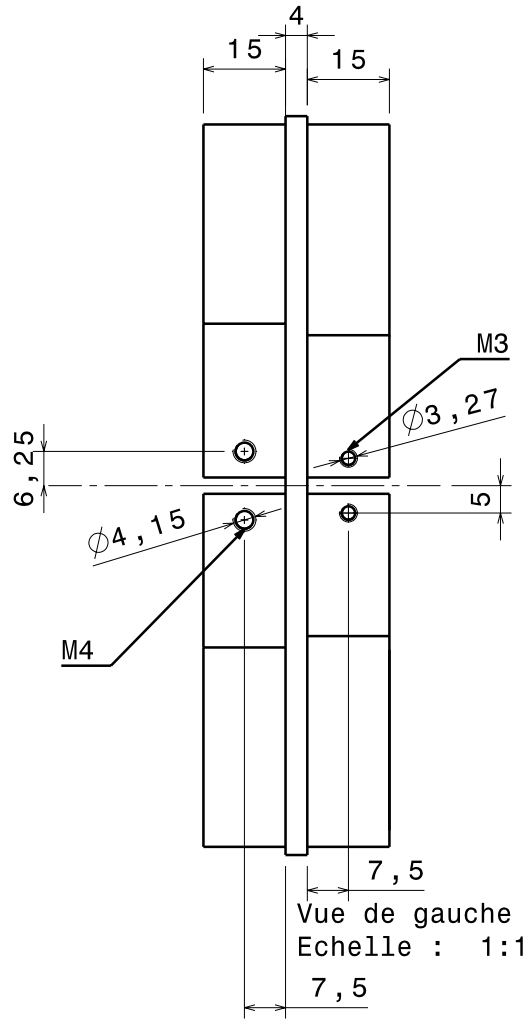


Vue isométrique
Echelle : 1:1

Quantité :1	FUSEE XXX Bague cône 2/2	CLES FACIL
Mat: AU4G Ech 1:1		
Dessiné par PEILLEX	CLES FACIL	
Le 26/11/02	e-mail :gpeillex@gmdserveur.insa-lyon.fr	



Vue de face
Echelle : 1:1

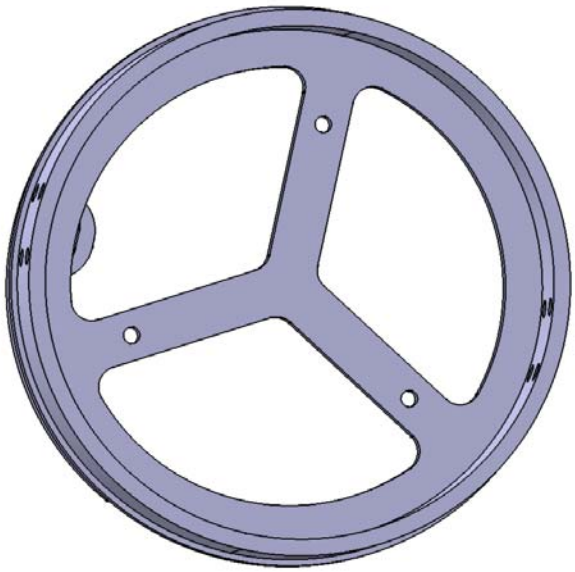


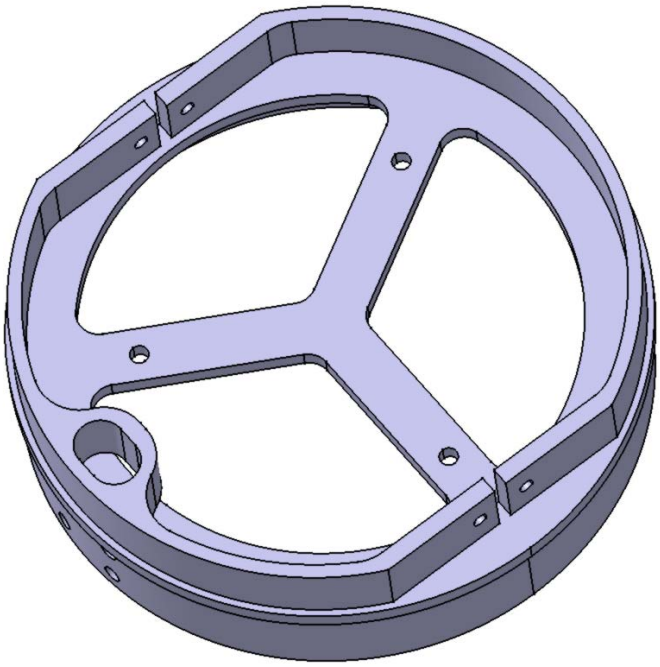
Vue de gauche
Echelle : 1:1

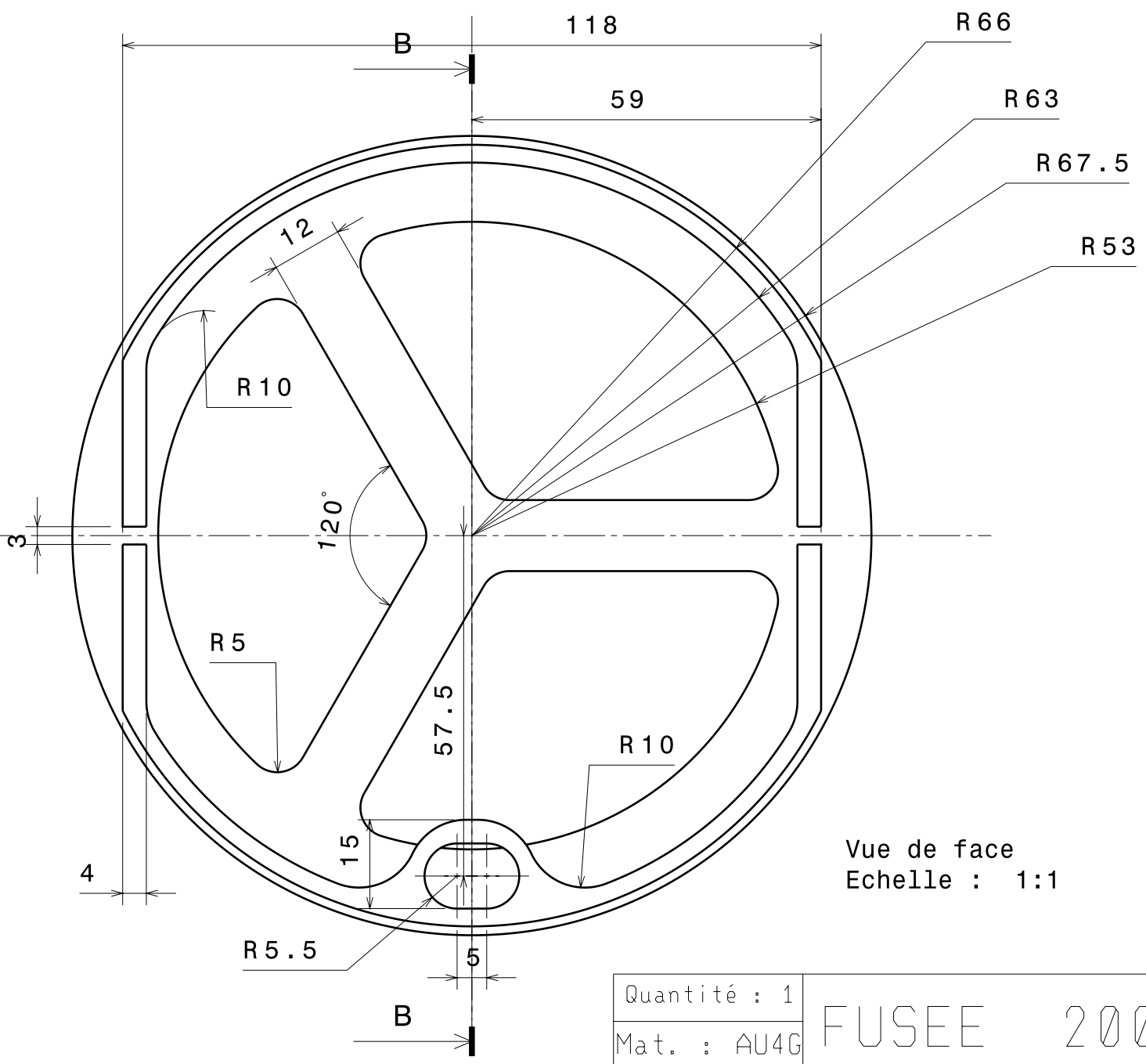
Les rayons non précisés sont de 5mm

Quantité:1	FUSEE XXX Bague cône 1/2	CLES FACIL
Mat: AU4G Ech 1:1		
Dessiné par PEILLEX	CLES FACIL	
Le 26/11/02	e-mail:gpeillex@gmdserveur.insa-lyon.fr	

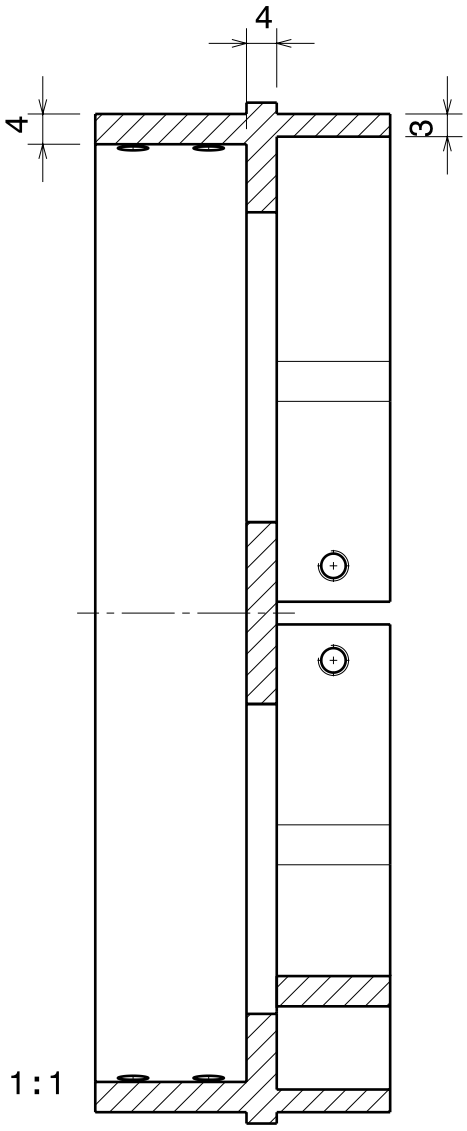
BAGUE INTERMEDIAIRE





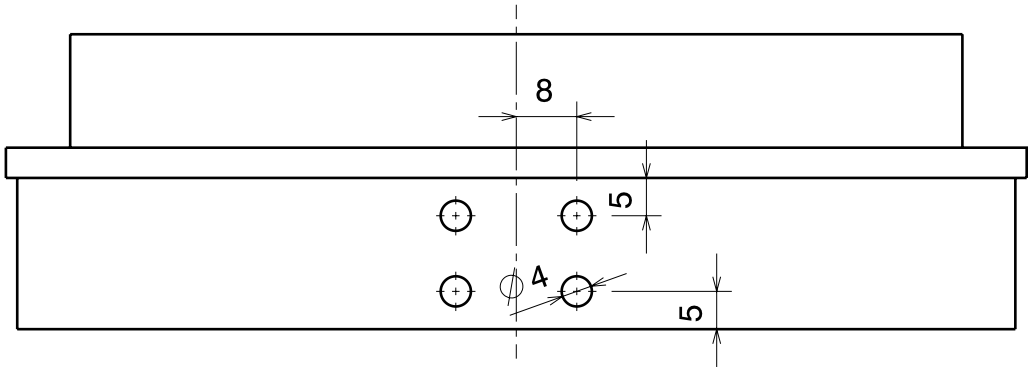


Vue de face
Echelle : 1:1

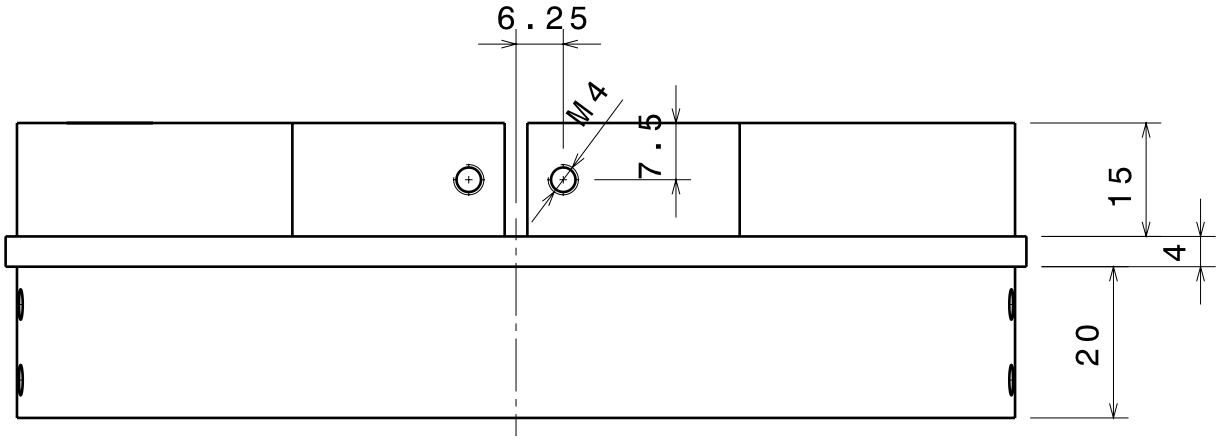


Coupe B-B
Echelle : 1:1

Quantité : 1	FUSEE 2003	CLES-FACIL
Mat. : AU4G	Bague centrale	
Ech : 1/1		
Dessiné par : Tourde	CLES-FACIL	
le 04/12/02	e-mail responsable : xtourde@free.fr	

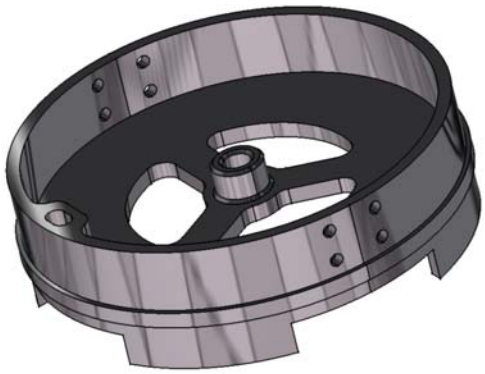


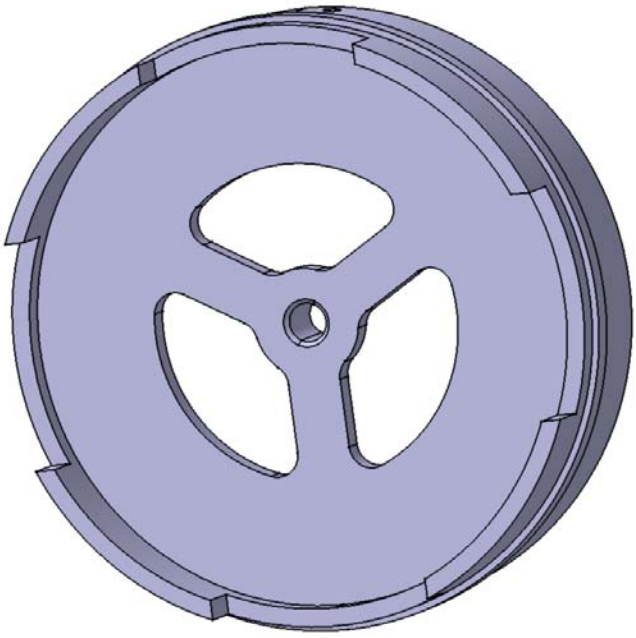
Quantité : 1	FUSEE 2003	C L E S F A C I L
Mat. : AU4G	Bague centrale	
Ech : 1/1		
Dessiné par : Tourde	CLES-FACIL	
le 04/12/02	e-mail responsable : xtourde@free.fr	

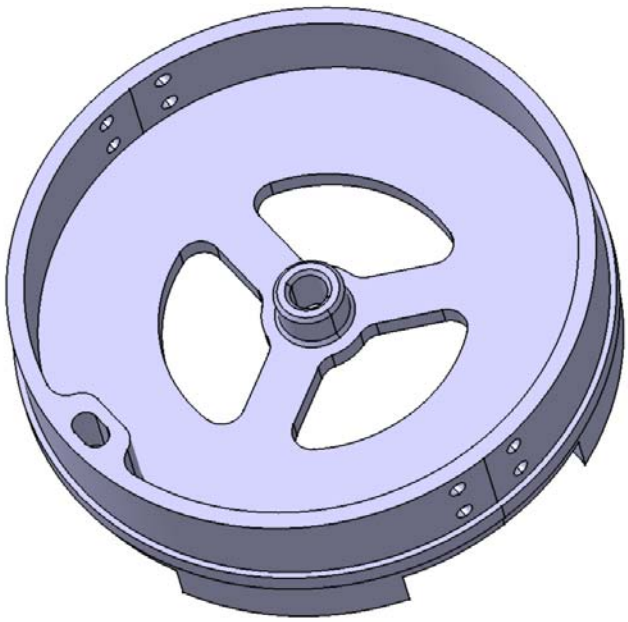


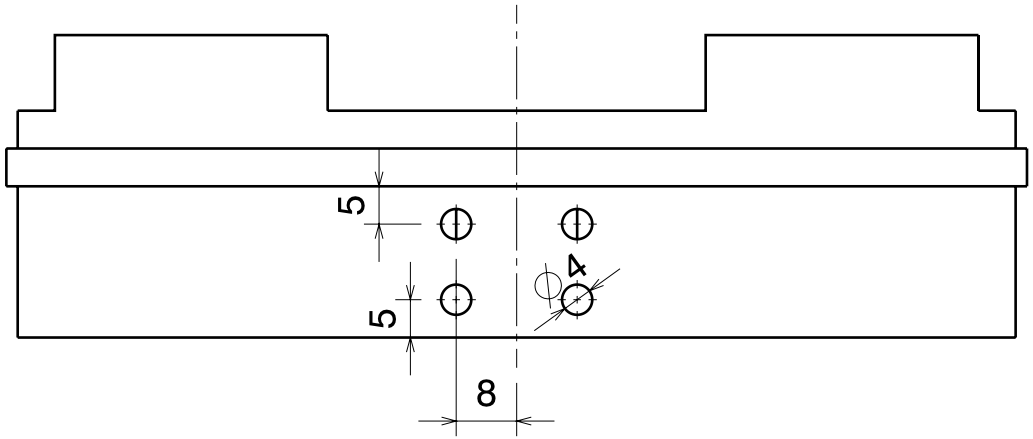
Quantité : 1	FUSEE 2003	CLES FACIL
Mat. : AU4G	Bague centrale	
Ech : 1/1		
Dessiné par : Tourde	CLES-FACIL	
le 04/12/02	e-mail responsable : xtourde@free.fr	

BAGUE POUSEE

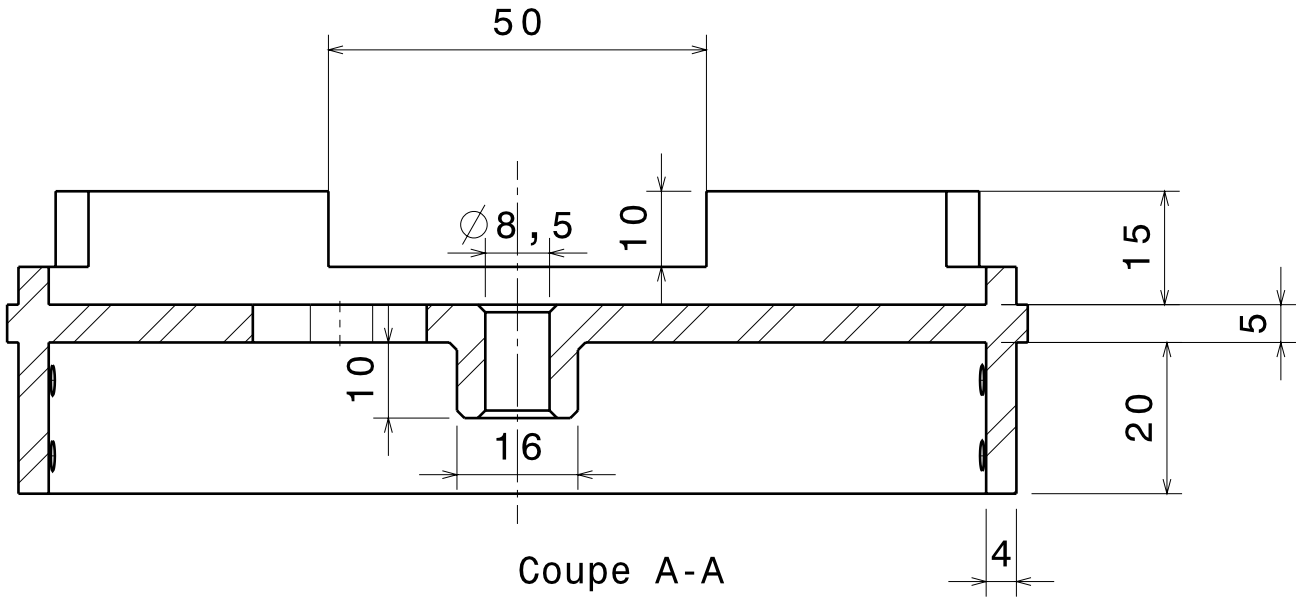






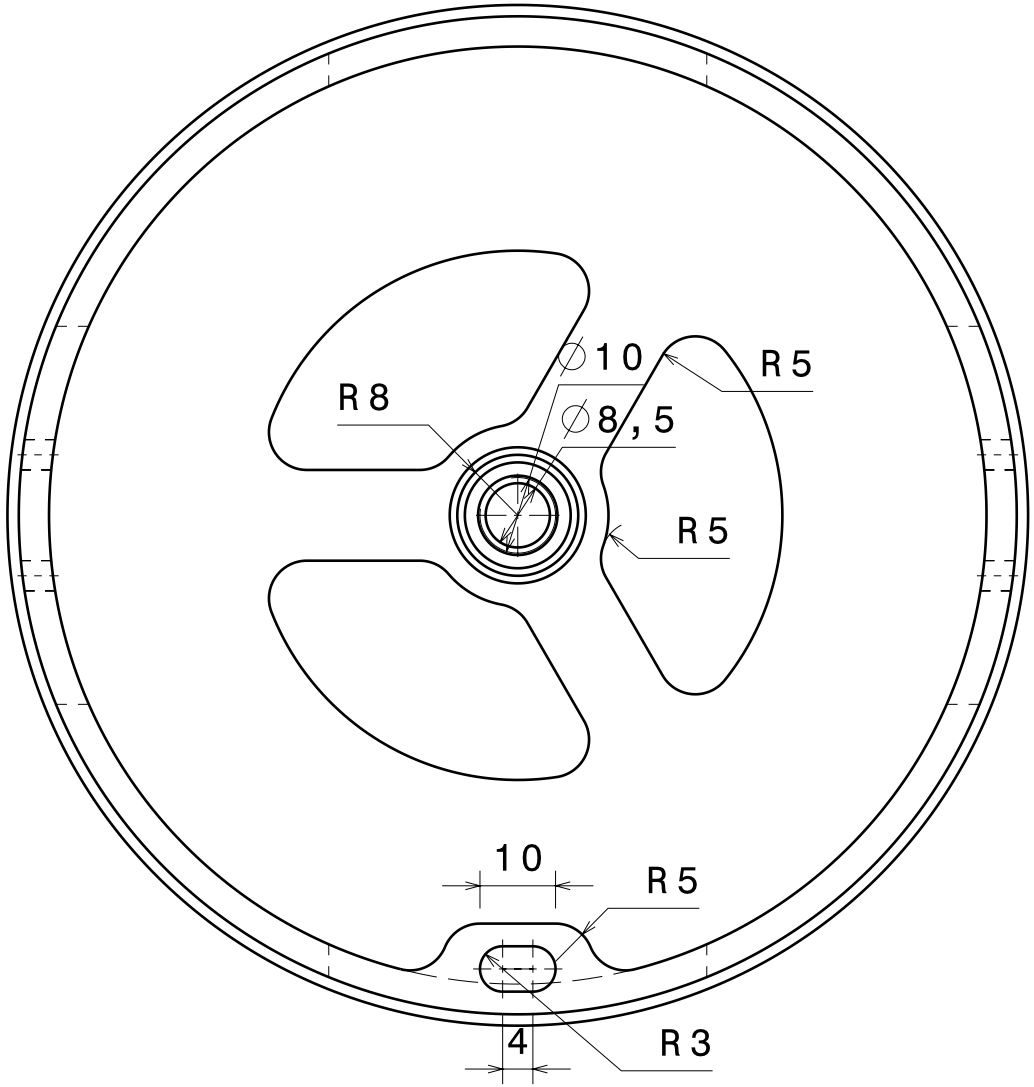


Quantité : 1	FUSEE 2003	C L E S F A C I L
Mat. : AU4G	Bague propulseur	
Ech : 1/1		
Dessiné par : Ruet	CLES-FACIL	
le 04/12/02	e-mail responsable : lruet@gmdserveur.insa-lyon.fr	

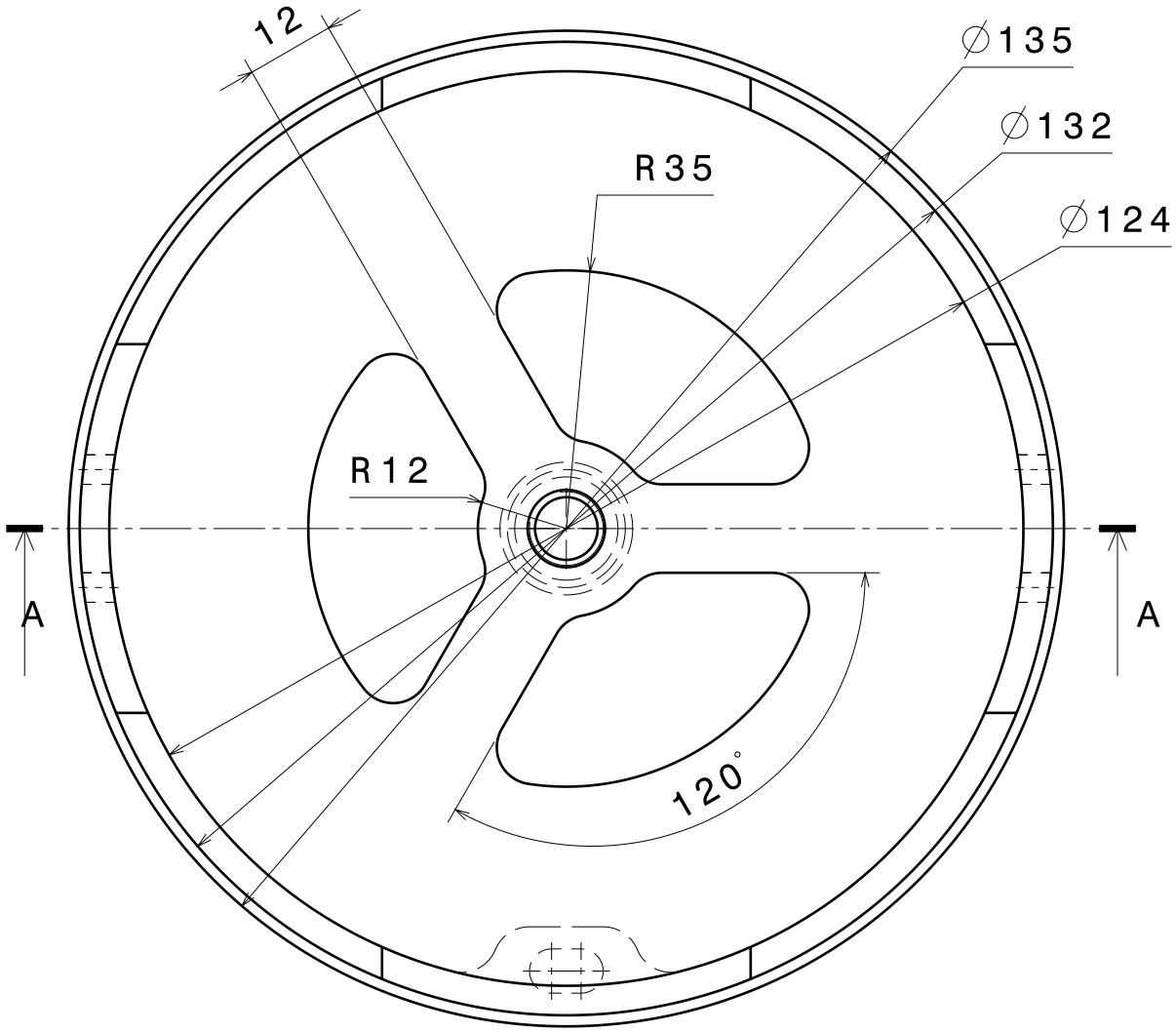


Coupe A-A
Echelle : 1:1

Quantité : 1	FUSEE 2003	C L E S F A C I L
Mat. : AU4G	Bague propulseur	
Ech : 1/1		
Dessiné par : Ruet	CLES-FACIL	
le 04/12/02	e-mail responsable : lruet@gmdserveur.insa-lyon.fr	

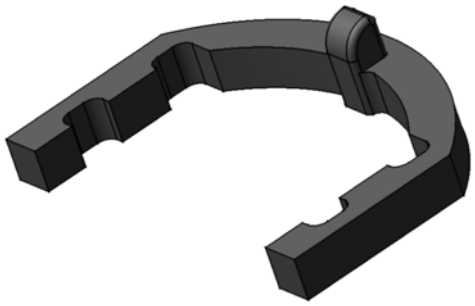


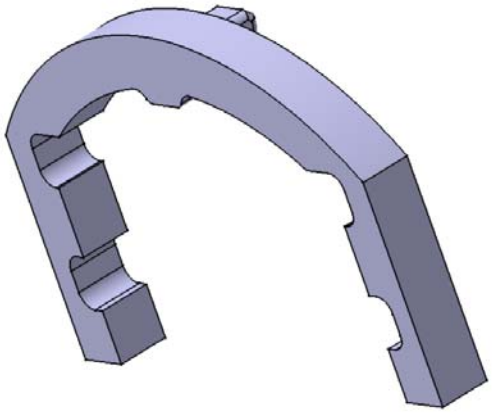
Quantité : 1	FUSEE 2003	C L E S F A C I L
Mat. : AU4G	Bague propulseur	
Ech : 1/1		
Dessiné par : Ruet	CLES-FACIL	
le 04/12/02	e-mail responsable : lruet@gmdserveur.insa-lyon.fr	

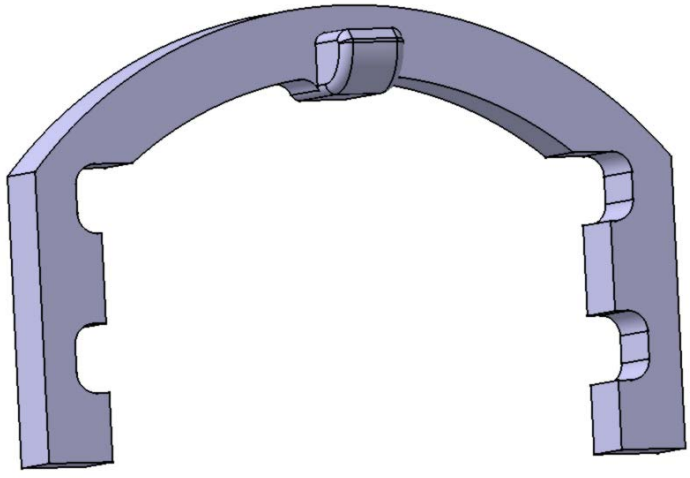


Quantité : 1	FUSEE 2003	C L E S F A C I L
Mat. : AU4G	Bague propulseur	
Ech : 1/1		
Dessiné par : Ruet	CLES-FACIL	
le 04/12/02	e-mail responsable : lruet@gmdserveur.insa-lyon.fr	

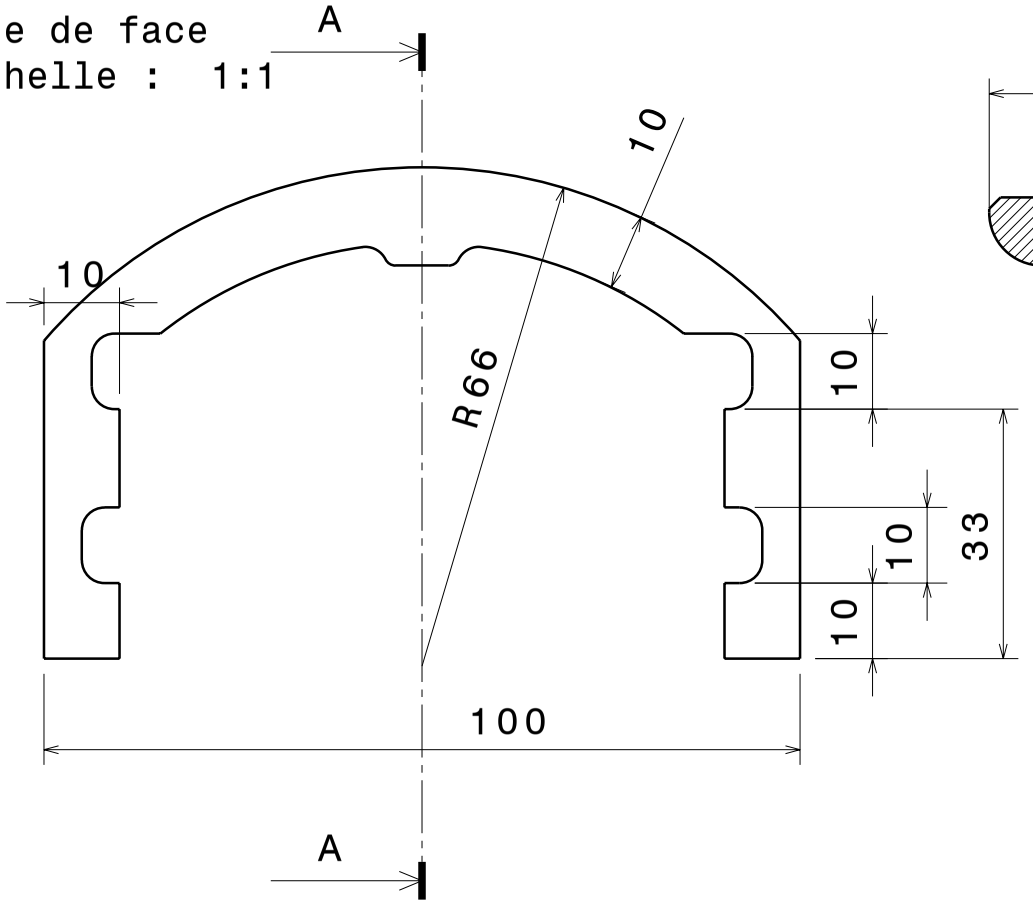
ARC MODULE



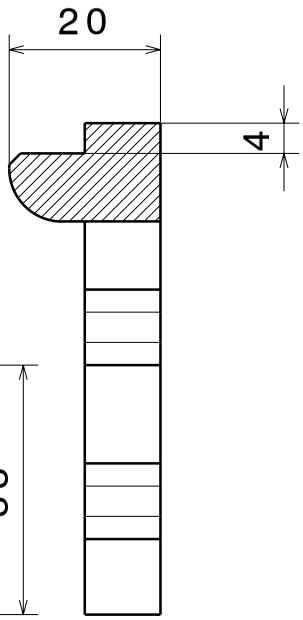




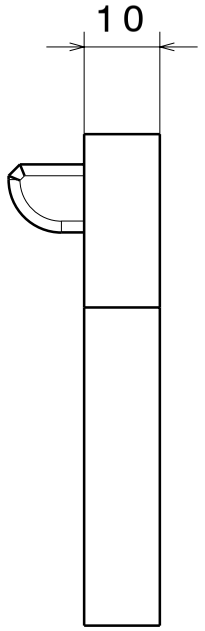
Vue de face
Echelle : 1:1



Coupe A-A
Echelle : 1:1



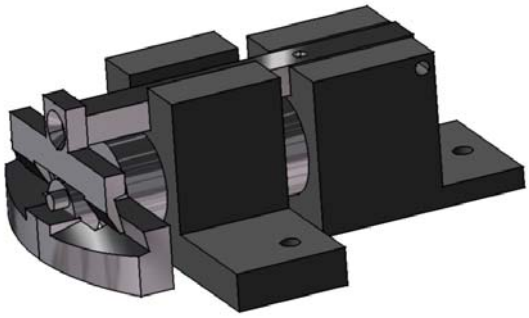
Vue de gauche
Echelle : 1:1

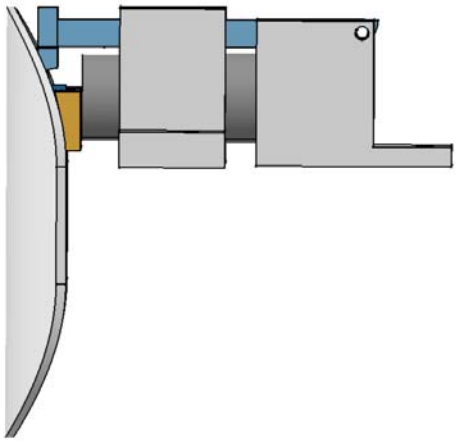


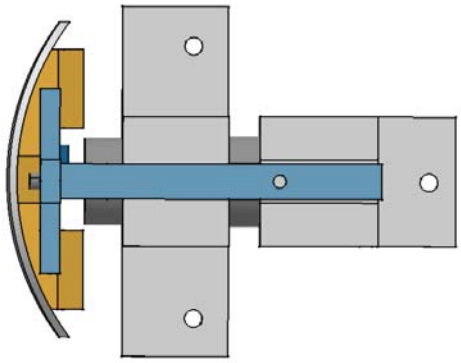
Quantité : 1	FUSEE ELLA
Mat. : AU4G	Bague minuterie - côtes sur trous
Ech : 1/1	
Dessiné par : Pontal	CLES-FACIL
le 01/12/01	e-mail responsable : lpontal@gmdserveur.insa-lyon.fr

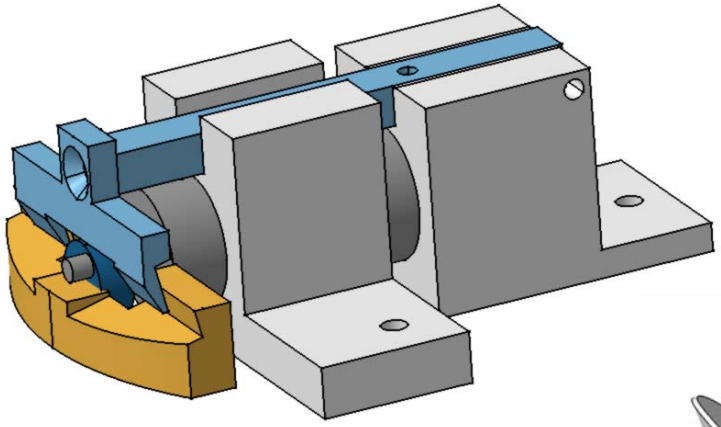
CLES FACIL

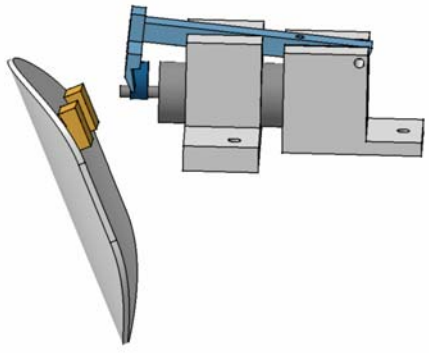
SYSTEME FERMETURE

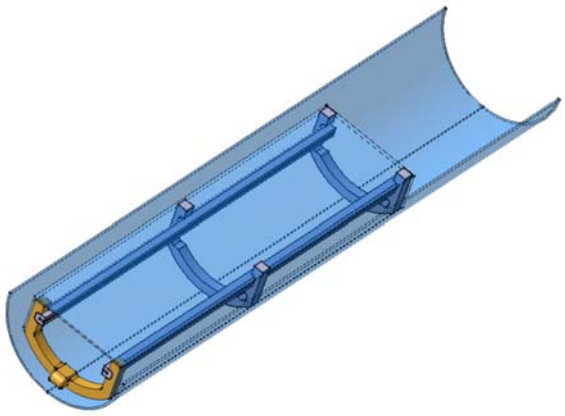










MODULE	 
CASE PARACHUTE	

16. EPILOGUE.

Axelle a effectué un vol nominal le 2 Août 2003. L'aventure ne s'arrêta pas là, loin s'en faut. Ce projet nous a permis de mettre le doigt sur quelques points à améliorer pour les années suivantes : soigner les procédures d'étalonnage, bienqu'un effort considérable ait été fait cette année sur ce point ; simplifier la carte de visualisation de façon à la rendre plus fiable, en outre il n'est pas utile d'avoir une surabondance d'information avant le décollage . Certaines expériences ont été mal spécifiées lors de leur conception, ce qui s'est inévitablement traduit par des résultats laissant à désirer. Cela est particulièrement vrai pour l'accéléromètre piézoélectrique.

Nous nous sommes également rendus compte de la nécéssité de préparer l'après-vol. Une expérience ne s'achève que lorsque toutes les données ont été analysées et les phénomènes physiques expliqués, cela reste en soi toute une aventure. Un effort est donc encore à fournir pour un prochain projet afin que l'exploitation et l'interprétation des résultats ne devienne pas un véritable calvaire.

Coordonnées du CLES-FACIL :

Pour nous écrire :

CLES-FACIL au BDR INSA
20 Avenue Albert Einstein
69621 Villeurbanne CEDEX.

Mail : cles-facil@insa-lyon.fr

Pour nous téléphoner :

04.78.94.31.73

Horaires de permanence, en période scolaire :

Le Mercredi de 20h30 à… tard,
Le Samedi de 14h00 à… tard aussi.