

2015 / 2016

NOM **LADRANGE**

PRENOM **BENOIT**

☐ Stage Découverte fin de P1

Autres cas : fin de ☐ P2 ☐ I1

☒ Stage Sc. et Techn. : ☐ S7 (juillet-déc.)

☒ I2 (juin-sept.)

☐ Stage de Fin d'études I3

Rapport de stage / emploi

>ENTREPRISE : *Institut d'Aéronomie Spatiale de Belgique (IASB)*

>DATES : *Du 06/06/2016 au 02/09/2016*

>SUJET MISSIONS : *Programmation d'une interface permettant d'envoyer des commandes à une carte électronique depuis un ordinateur.*

>TUTEUR ENTREPRISE : *(Nom, Prénom, Fonction) NEEFS Eddy, responsable du service Engineering.*



A COMPLETER UNIQUEMENT POUR LES STAGES I2 ET I3 :

>Option : ☐ SE ☐ BIO ☒ EOC ☐ NRJ ☐ LD ☐ CR ☐ SIAT

>Niv. de confidentialité: ☐ **mon rapport est confidentiel** : Niveau ☐ 1 (moy.) ☐ 2 (élevé)
Tel qu'indiqué sur convention signée. Non réponse = l'ESEO retient Niveau 0 (aucune conf.)

>Domaine de l'entrepr.: ☐ Auto. ☐ Aéron.. ☐ Banque-fi.-assur. ☐ Biomédical-santé
☐ Energie ☐ NTIC ☐ Télécoms ☒ Aéronomie

>Uniquement pour I2 : ☐ mon tuteur sera présent à ma soutenance orale (facultatif)

>Uniquement pour I3 : ☐ stage à dominante '**management**' ou ☐ à dominante '**recherche**'
☐ année I3 effectuée sous 'contrat Pro' entreprise
☐ **mon tuteur sera présent à ma soutenance orale**
☐ **mon tuteur sera présent au déjeuner le jour de ma soutenance**

Remerciements

Je tiens à remercier chaleureusement le personnel de l'institut, notamment les membres du service *Engineering*, qui m'ont accueilli comme l'un des leurs et ont fait en sorte que mon stage se déroule dans les meilleures conditions possibles. A tous, un grand merci.

Sophie BERKENBOSCH

Roland CLAIRQUIN

Eddy EQUETER

Eddy NEEFS

Francis SCOLAS

Jurgen VANHAMEL

Un remerciement particulier à l'attention de Mme Anne DE RUDDER sans laquelle je n'aurais pas trouvé ce stage.

Introduction

En 2015, l'attention des médias internationaux a été attirée par la 21ème conférence des parties (ou COP21) qui s'est déroulée à Paris et avait pour thème principal le changement climatique et les solutions envisagées pour réduire notre impact sur la planète. Ce sommet international nous a rappelé la réalité de la situation environnementale et l'urgence de prises de décisions ayant pour but de limiter le réchauffement climatique.

D'autre part, un tel accord n'aurait pu être rédigé sans le travail d'équipes de recherche à travers le monde dont les conclusions poignantes ont poussés les gouvernements à réagir.

Dans le cadre de mes études à l'ESEO (Ecole Supérieure d'Electronique de l'Ouest), j'ai souhaité effectuer mon stage technique de deuxième année de cycle ingénieur dans un institut dont les activités sont étroitement liées aux recherches en matière de climat. En effet, cet institut est porté sur l'aéronomie spatiale, soit l'étude de l'atmosphère terrestre ainsi que de l'espace extra-atmosphérique. D'une façon plus générale, l'aéronomie couvre l'analyse des couches atmosphériques s'étirant de la stratosphère jusqu'à la ionosphère.

Ce thème extrêmement vaste permet de couvrir des domaines allant de la météorologie à la physique spatiale, en passant par l'étude de la qualité de l'air ou des rayonnements UV.

Afin de mener ses différentes mesures, l'institut utilise de nombreux matériels et équipements, aussi bien sur terre que dans l'espace. La conception et la réalisation des instruments de mesure reposent sur le travail d'une équipe d'ingénieurs et de techniciens dans laquelle j'ai été affecté durant toute la durée de mon stage. Plus précisément, le contenu du stage s'inscrit dans l'activité du service d'électronique du département, me permettant de faire le lien direct avec la majeure choisie à l'ESEO, à savoir l'électronique des objets connectés.

La mission proposée m'a particulièrement attirée car elle fait partie intégrante d'un projet de mesure de différents gaz présents dans l'atmosphère, dont la plupart sont directement impliqués dans le réchauffement climatique (ozone, gaz à effet de serre). De plus, mon modeste rôle dans ce vaste projet comprend des tâches liées à l'électronique, ce qui correspond parfaitement au stage recherché.

Le présent document constitue le rapport de ce stage effectué à l'IASB (Institut d'Aéronomie Spatiale de Belgique) sur la période s'étalant du 6 juin au 2 septembre 2016.

Dans un premier temps, l'institut ainsi que le contexte de ma mission seront présentés, puis, dans un second temps, le contenu du stage sera décrit. Enfin, un rapport personnel d'étonnement sera proposé.

Fiche de synthèse

- ✧ Sujet : programmation d'une interface permettant d'envoyer des commandes à une carte électronique depuis un ordinateur.
- ✧ Entreprise : IASB (Institut d'Aéronomie Spatiale de Belgique).
- ✧ Dates de stage : du 6 juin au 2 septembre 2016.
- ✧ Principaux collaborateurs :
 - Eddy NEEFS (responsable du service ingénierie).
 - Sophie BERKENBOSCH (responsable du département électronique).
 - Jurgen VANHAMEL (ingénieur industriel).
 - Roland CLAIRQUIN (ingénieur industriel).
- ✧ Réalisations : interfaces graphiques, programmation d'un microcontrôleur et d'autres circuits intégrés.
- ✧ Contraintes et difficultés rencontrées : difficultés rencontrées lors de la programmation des interfaces graphiques.

Sommaire

Remerciements.....	2
Introduction	3
Fiche de synthèse.....	4
I - Société et contexte	6
1) Historique et organisation	6
2) Les missions de l'IASB	10
3) Services publics proposés	13
II - Contenu du stage	15
1) Présentation du projet.....	15
a. Vue d'ensemble	15
b. Particularité de la chaîne RF.....	22
2) Sujet	22
3) Objectif final	23
4) Contraintes	23
5) Moyens	25
6) Planning d'organisation	25
7) Travail réalisé	25
a. Travail effectué sur le DDS	25
b. Travail effectué sur la PLL	38
c. Autres tâches effectuées	61
III - Rapport personnel d'étonnement	66
1) Compétences acquises	66
a. Scientifique et technique.....	66
b. Organisationnel.....	66
c. Relationnel.....	67
d. Economique.....	67
2) Conclusion.....	68
ANNEXE 1 : CV	70
ANNEXE 2 : Planning du stage	72
ANNEXE 3 : Bibliographie	76
ANNEXE 4 : Glossaire des définitions et acronymes.....	77
ANNEXE 5 : Table des illustrations	80

I - Société et contexte

1) Historique et organisation

Tout d'abord, il est à noter que l'aéronomie est une science relativement nouvelle puisque le terme a été utilisé pour la première fois en 1951 par un astronome et géophysicien britannique nommé Sydney Chapman (1888-1970). Le terme fut ensuite officiellement intégré au milieu scientifique trois années plus tard, en 1954, en lui associant une définition propre : « *science des régions de la haute atmosphère où les réactions de dissociations et d'ionisations sont importantes* ».

Toutefois, des recherches dans ce domaine avaient déjà été engagées auparavant par Marcel Nicolet, un météorologue et géophysicien belge qui consacrera sa carrière à l'étude de l'atmosphère supérieure, dès l'obtention de son doctorat en 1937. C'est d'ailleurs lui qui fondera l'Institut Royal d'Aéronomie Spatiale de Belgique (IASB) en 1964, situé juste à côté de l'Institut Royal Météorologique de Belgique (IRM) et de l'Observatoire Royal de Belgique.

L'IASB est ainsi l'un des établissements scientifiques belges les plus récents, qui a pu atteindre son plein potentiel avec les avancées majeures dans le domaine des satellites artificiels. Ceux-ci évoluent dans la haute atmosphère terrestre et sont utilisés pour relever différentes mesures de paramètres propres à l'aéronomie.

Etant donné que l'IRM, l'IASB ainsi que l'Observatoire sont en étroite collaboration, ces trois structures sont regroupées au sein du même complexe situé à Uccle, l'une des 19 communes de Bruxelles.

D'un point de vue organisationnel, l'institut est placé sous l'égide de la Politique Scientifique Fédérale Belge, également connu sous l'acronyme BELSPO (*Belgian Science Policy*). Cette administration gouvernementale est chargée d'assurer le bon déroulement des programmes scientifiques et de recherche au niveau fédéral. C'est également cet organisme qui joue le rôle de relais entre le gouvernement belge et l'Union Européenne lorsqu'il est question de programmes scientifiques intereuropéens voire internationaux.

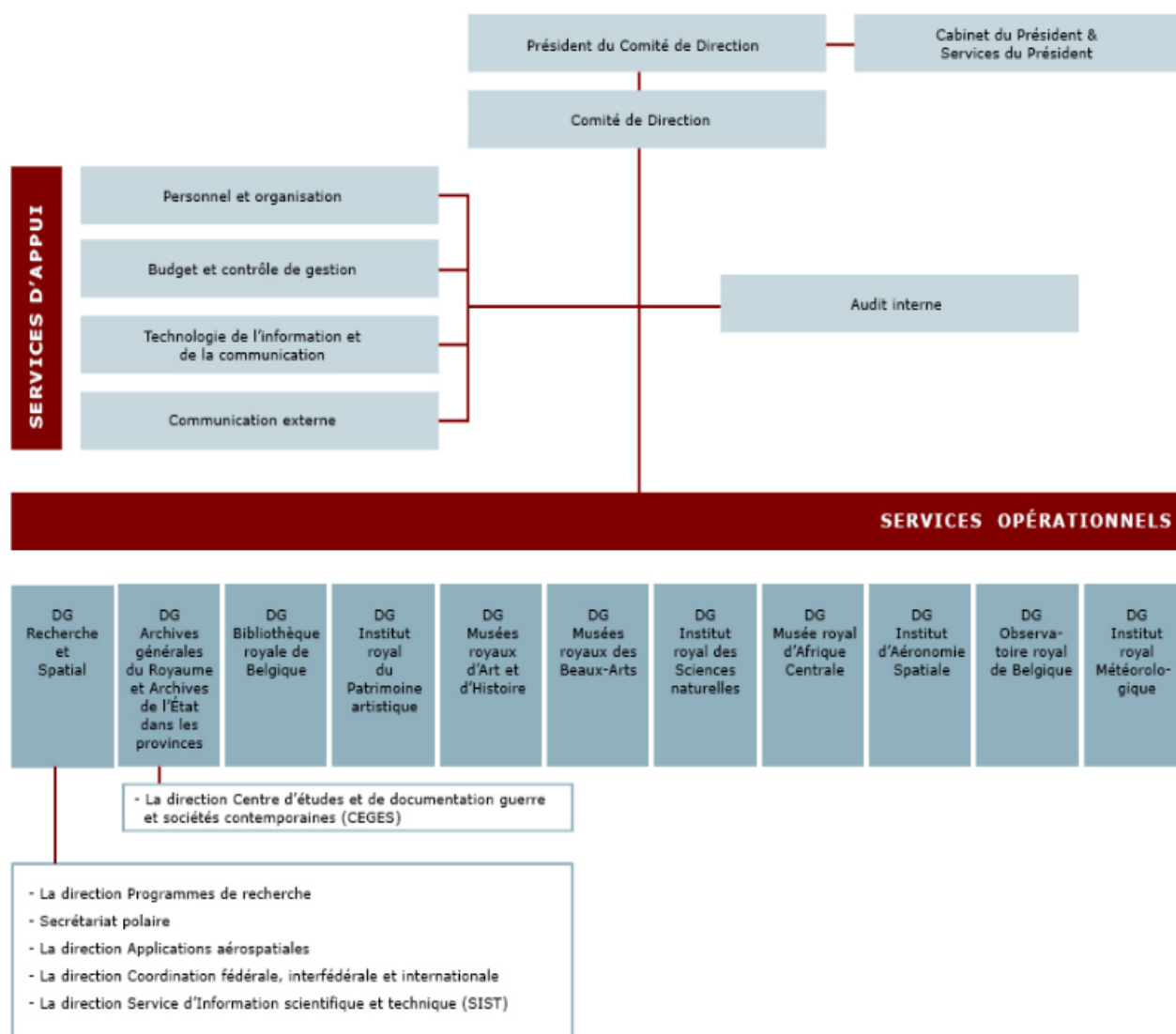


Figure 1 : organigramme du BELSPO

En plus de l'IASB, 9 autres établissements scientifiques composent le BELSPO, parmi lesquels nous retrouvons le musée royal de l'Afrique centrale ou encore les musées royaux des beaux-arts de Belgique.

En termes d'effectifs, le département de la Politique scientifique fédérale compte environ 2800 employés (dont une centaine à l'IASB) et le budget qui lui est consacré représente 30% du budget public belge.

Chaque établissement scientifique composant le BELSPO possède son organisation propre avec un directeur général à sa tête, lui permettant de coordonner efficacement les missions qui lui sont confiées.

Concernant l'IASB, ses principales missions s'articulent autour de quatre pôles présentés dans l'organigramme ci-après :

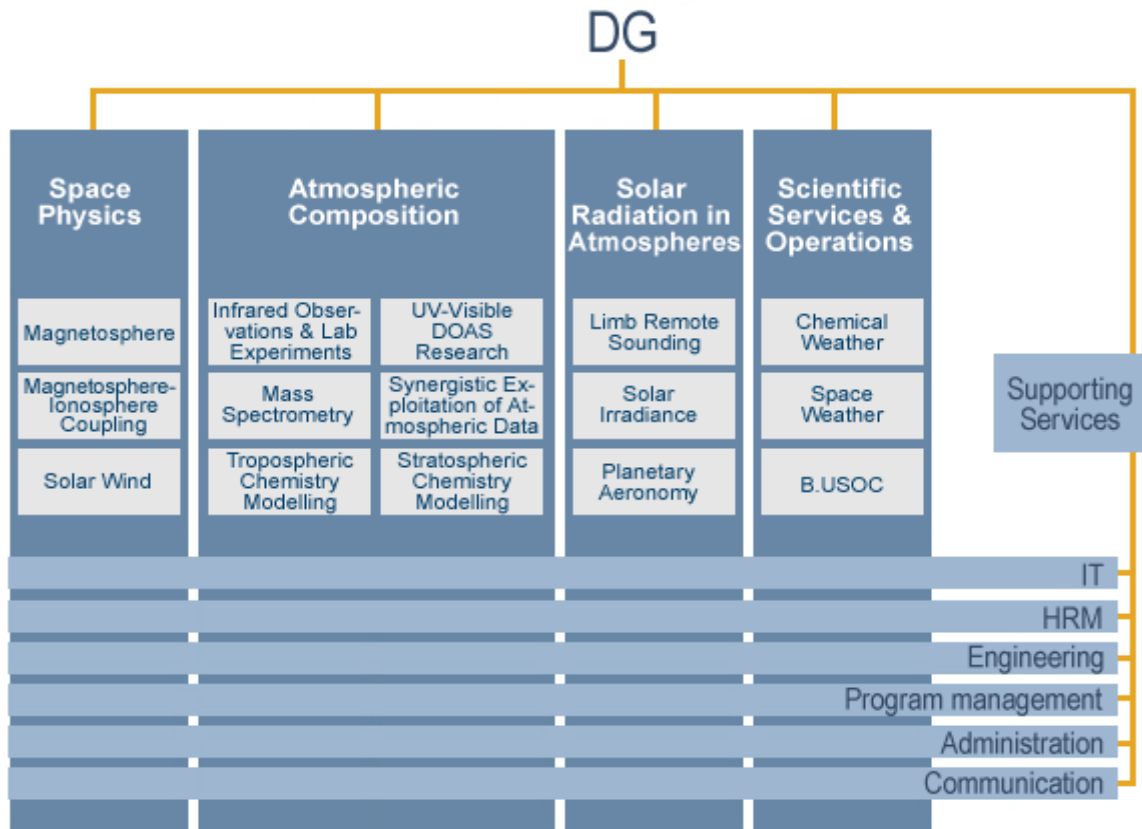


Figure 2 : organigramme de l'IASB

Le département « *Space Physics* » s'occupe notamment de l'étude physique et chimique des différentes couches de la haute atmosphère d'une planète. Ceci implique l'étude des interactions magnétiques induites par la présence des champs magnétiques générés par les corps célestes (la magnétosphère), ou encore l'étude du vent solaire.

Quant au département « *Atmospheric Composition* », son but est de recueillir un maximum de données relatives à la composition chimique de l'atmosphère puis de les traiter, en vue d'établir des statistiques et modèles précis. A titre d'exemple, c'est ce département qui relève la proportion des principaux gaz présents dans l'atmosphère impliqués dans l'effet de serre.

Les nombreux phénomènes spatiaux liés à l'activité solaire expliquent à eux seuls l'existence du département « *Solar Radiations in Atmospheres* ». C'est notamment ce service qui publie l'indice UV en plusieurs endroits du pays par le biais d'un site internet spécifique.

Enfin, le dernier département intitulé « *Scientific Services and Operations* » est impliqué dans le recueil et l'étude de données permettant d'établir des modèles de météo spatiale.

Par ailleurs, comme dans toutes entreprises, des services complémentaires existent. Ce sont les services de soutien (« *Supporting Services* » sur l'organigramme) composés des ressources humaines, du service informatique ou encore du service de communication. De plus, il existe un service d'ingénierie qui conçoit les différents appareils de mesures et capteurs en tout genre permettant aux scientifiques de recueillir les données dont ils ont besoin. Ces instruments auront soit pour vocation de rester au sol, soit d'aller dans l'atmosphère, voire même dans l'espace.

Ce service est divisé en deux équipes : une équipe mécanique et une équipe électronique. C'est dans cette dernière que j'ai effectué mon stage.

Concernant les moyens alloués aux équipes du département ingénierie, l'institut possède un laboratoire d'électronique bien équipé, avec tout le matériel nécessaire pour la conception de circuits électroniques ainsi que des équipements de test comme des oscilloscopes ou des analyseurs de spectre.



Figure 3 : laboratoire d'électronique de l'institut

Pour la partie mécanique, l'équipe possède un atelier spécifique attenant au bâtiment principal de l'IASB et possédant plusieurs machines industrielles telles que des machines à découper, à plier ou à percer le métal.



Figure 4 : atelier de mécanique de l'institut

Par ailleurs, l'institut dispose d'une salle propre avec une chambre à vide thermique. Ce lieu particulier sert à effectuer des tests dans des environnements spécifiques ou encore à assembler des composants sensibles.



Figure 5 : salle propre avec chambre à vide thermique

2) Les missions de l'IASB

La mission principale de l'institut est de concevoir des équipements destinés à effectuer des prises de différentes mesures dans l'atmosphère ou l'espace pour des projets nationaux ou internationaux.

Nous verrons ici quelques exemples de projets réalisés par l'institut qui permettront de mieux se représenter le rôle de l'IASB à travers des exemples concrets.

Projets spatiaux :

- **SOIR** : *Solar Occultation in the InfraRed*. Il s'agit d'un spectromètre à haute résolution fonctionnant dans la gamme infrarouge. L'instrument a pour but d'établir les principales caractéristiques de l'atmosphère de la planète Vénus. Cet appareil a été ajouté au satellite de la mission *Venus Express* de l'Agence Spatiale Européenne (ESA). L'appareil a permis de démontrer la présence d'une couche atmosphérique froide à une altitude de 130 km prise en étau entre deux couches chaudes, ce qui n'avait été prévu par aucun modèle.

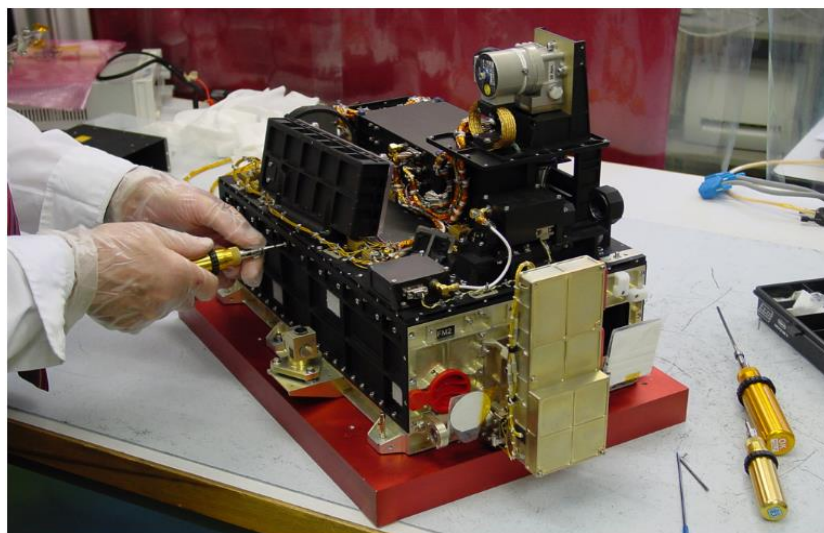


Figure 6 : photo du spectromètre SOIR

Le lancement du satellite a eu lieu le 9 septembre 2005 et la mission a pris fin en avril 2015.

- EPT : *Energetic Particle Telescope*. Il s'agit d'un instrument permettant de mesurer l'énergie provenant de flux de particules hautement énergétiques. L'appareil peut mesurer l'énergie d'une particule chargée et en déduire de quelle particule il s'agit. La particularité de cet instrument est qu'il est capable de couvrir une large gamme énergétique tout en conservant une bonne résolution.

Le télescope a permis de réaliser une cartographie des flux de particules chargées et a également contribué à l'enrichissement des connaissances scientifiques concernant les ceintures de radiation.

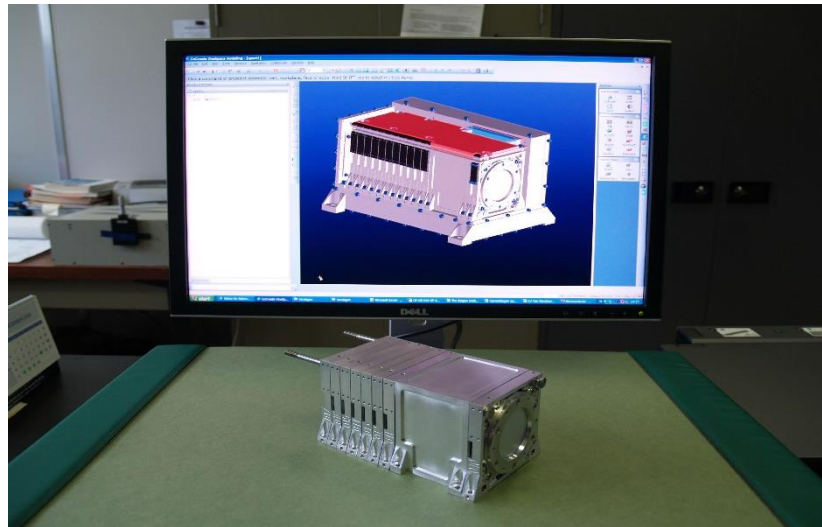


Figure 7 : photo du télescope EPT

L'EPT a été placé à bord d'un satellite belge de type PROBA-V qui a été lancé le 7 mai 2013. Les premières mesures ont pu être réalisées au mois de septembre 2013.

- NOMAD : *Nadir and Occultation for Mars Discovery*. Cet instrument de mesure est un spectromètre qui aura pour objectif d'établir la composition chimique de l'atmosphère de la planète Mars. NOMAD sera embarqué à bord de l'orbiteur *ExoMars Trace Gas Orbiter*, projet résultant de la collaboration entre l'ESA et la NASA.

Ce spectromètre a été conçu pour fonctionner sur trois canaux, deux fonctionnant en infrarouge et un autre permettant d'élargir la gamme couverte jusqu'aux rayons UV et visibles.

Les grands enjeux scientifiques de la mission sont de déterminer la géologie, le volcanisme ainsi que l'existence de traces de vie sur la planète grâce à l'analyse chimique de l'atmosphère.

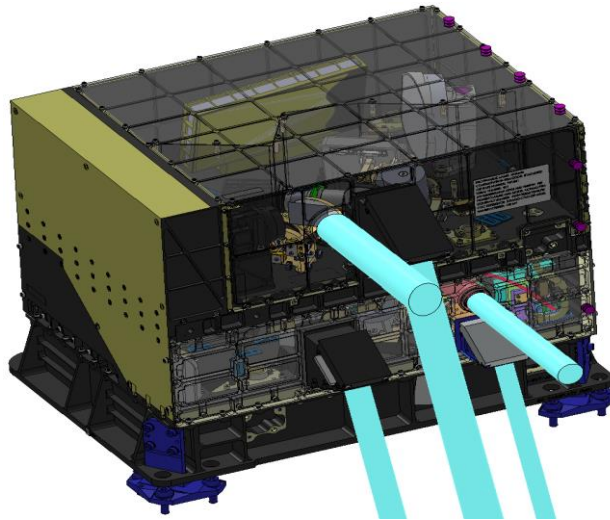


Figure 8 : schéma du spectromètre NOMAD de la mission spatiale ExoMars

Le lancement est prévu pour le 14 mars 2016.

Projets terrestres :

- IMPCVOC : *Impact of Phenology and Environmental Conditions on biogenic Volatile Organic Compounds emissions from forest ecosystems*. Il s'agit d'une étude sur l'influence des plantes sur la qualité de l'air par la mesure des flux de COVB (Composés Organiques Volatiles Biogéniques). Ces composés organiques sont majoritairement émis par les forêts et les zones de culture.

Afin de mesurer les flux de COVB, l'IASB a mis au point une sorte de réceptacle, dans lequel il est possible d'y enfermer une branche, en combinaison avec un spectromètre de masse.



Figure 9 : photo du réceptacle emprisonnant une branche d'arbre

Ce projet a été réalisé au cours de l'année 2007.

- BRAMS : *Belgian Radio Meteor Stations*. Ce réseau belge de détection radio de météores s'appuie sur l'observation selon laquelle les objets traversant notre atmosphère à grande vitesse laissent derrière eux une trainée d'électrons. L'hypothèse que cette trainée d'électrons pourrait réfléchir des ondes radios de la bande VHF a ensuite été avancée. C'est comme cela que le projet BRAMS a vu le

jour car il consiste en un réseau de plusieurs antennes réceptrices ainsi que d'un émetteur radio. Sachant que le délai de réception de l'onde réfléchie peut aller d'une fraction de seconde jusqu'à plusieurs minutes pour les objets les plus gros, il est possible de retracer la progression de l'objet dans l'atmosphère à l'aide de plusieurs récepteurs disséminés sur une large zone. Le système de mesure est ensuite capable de déterminer la masse ou encore la vitesse du météore.



Figure 10 : photo d'une des antennes de réception du projet BRAMS

Ce projet a été mis en place en 2010.

Les données recueillies seront ensuite traitées afin d'établir des modèles ou des statistiques directement exploitables par d'autres scientifiques ou même des particuliers.

3) Services publics proposés

L'institut, par le biais des différents projets réalisés, récolte et traite de nombreuses données pouvant être directement mises à disposition des particuliers. En effet, certains sites internet gérés par l'IASB présentent les résultats provenant de plusieurs instruments de mesure.

La proposition de tels services ouverts à tous est un bon moyen pour les scientifiques et ingénieurs de valoriser leur travail et de partager leurs connaissances avec le plus grand nombre.

Parmi les services proposés, on retrouve un site internet présentant des informations relatives à la météorologie spatiale (SPENVIS). Il s'agit d'un outil proposé par l'ESA mais qui est régi par l'IASB. Cette page web permet notamment à des scientifiques et ingénieurs travaillant dans le domaine spatial de consulter les risques liés à la météo spatiale avant de prévoir une mission dans l'espace, par exemple.

Toujours dans le milieu de la météorologie spatiale, les équipes de l'IASB proposent un autre outil nommé *COMESSEP Alert System*, qui agit comme un véritable système d'alerte lorsque des événements spatiaux surviennent, notamment des phénomènes solaires. A titre d'exemple, cet outil pourra indiquer la présence d'éruptions solaires ainsi que la date de leurs occurrences.

D'autre part, l'IASB participe également au projet multinational MACC (*Monitoring Atmospheric Composition and Climate*) qui a pour but de présenter la composition de l'atmosphère terrestre. Le service belge apportant sa pierre à l'édifice s'intitule BASCOE pour *Belgian Assimilation System for Chemical Observations* qui propose des observations de la

composition chimique de la stratosphère, la teneur en ozone par exemple.

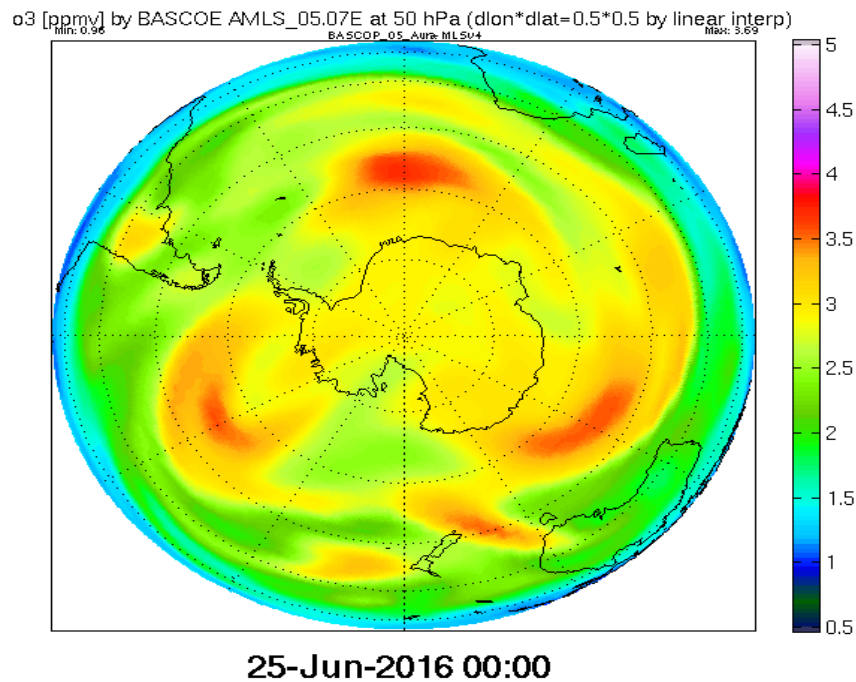


Figure 11 : capture d'écran d'un relevé de l'outil BASCOE indiquant la répartition et la concentration en ozone de la stratosphère

De plus, comme évoqué précédemment, l'institut offre la possibilité de consulter l'indice UV du pays grâce au traitement de données fournies par sept stations réparties sur le territoire. L'accès à ce type d'informations permet notamment d'adapter son comportement en fonction du rayonnement UV présent en surface.

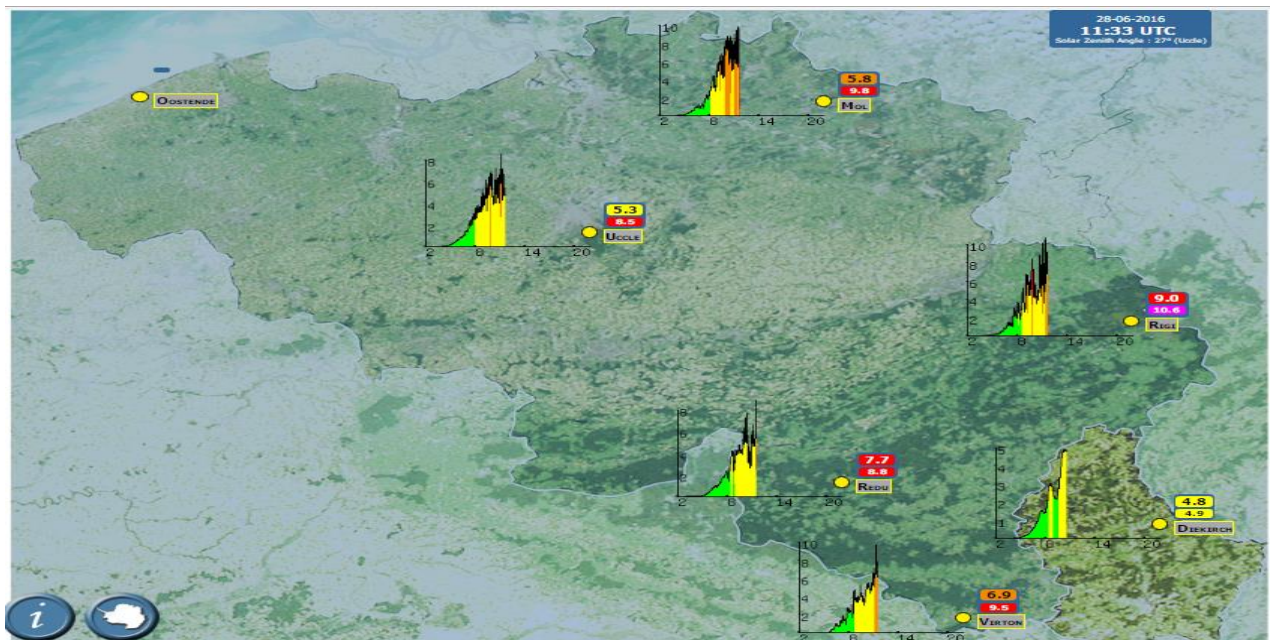


Figure 12 : capture d'écran du site internet renseignant sur l'indice UV en Belgique

Nous terminerons avec l'outil de support à l'aviation SACS (*Support to Aviation Control Service*) qui permet de visualiser sur une carte les différents gaz émis par une éruption volcanique. En effet, nous gardons encore en tête le nuage de cendres relâché par le volcan

islandais Eyjafjallajökull qui avait affecté les vols aux départs de plusieurs pays à travers le monde. Ceci explique l'importance d'un outil permettant de suivre le déplacement des nuages de poussières afin de limiter les risques pouvant impacter l'aviation.

En plus de ces services virtuels, des publications scientifiques sont régulièrement relâchées par le service de communication de l'institut et concernent les travaux de certains scientifiques ou ingénieurs y travaillant.

De surcroît, le BELSPO propose un magazine gratuit paraissant cinq fois par an et dans lequel sont regroupés de nombreuses observations, hypothèses et définition de projets concernant une certaine problématique.

II - Contenu du stage

1) Présentation du projet

a. Vue d'ensemble

La mission prévue par mon stage s'inscrit au sein d'un projet ayant pour vocation l'observation du limbe atmosphérique de la Terre. Le limbe atmosphérique correspond à l'enveloppe extérieure de l'atmosphère visible depuis l'espace, comme nous pouvons le voir sur la figure suivante :



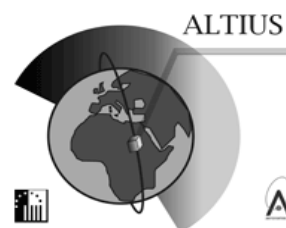
Figure 13 : photo montrant le limbe atmosphérique terrestre

Il s'agit, plus simplement, de la partie de l'atmosphère qui est éclairée par le soleil, ce qui lui donne cet aspect de halo laiteux.

Le projet porte le nom ALTIUS correspondant à *Atmospheric Limb Tracker for the Investigation of the Upcoming Stratosphere*, autrement dit un analyseur de limbe atmosphérique. Dans les grandes lignes, ce projet vise à prendre une photo du limbe, un peu de la même façon que la figure précédente, puis, avec les appareils de mesure adaptés, d'analyser la composition gazeuse de la stratosphère.

Il s'agira d'un projet particulièrement intéressant pour l'IASB car il représente la toute première analyse belge d'une des couches de la haute atmosphère terrestre.

Afin d'évaluer la concentration en gaz de l'atmosphère, différentes techniques



existent comme, par exemple, la prise d'échantillons à l'aide de ballons sondes, ou encore la télédétection qui présente l'avantage de pouvoir effectuer des mesures à distances, sans contact direct avec la cible. C'est cette-dernière technique qui servira de base au projet ALTIUS.

La télédétection repose sur la loi du rayonnement de Kirchhoff, selon laquelle un gaz situé sur le chemin d'un rayon lumineux provenant d'une source de lumière continue, en absorbe certaines couleurs.

Ensuite, après avoir défini la technique à utiliser, les personnes à l'origine du projet se sont penchées sur la question de comment prendre, véritablement, la mesure désirée. L'utilisation d'un laser à partir d'une station au sol est envisageable (LiDAR : *Light Detection and Ranging*), ou encore l'utilisation d'un ballon stratosphérique. Mais leur choix s'est plutôt porté sur l'utilisation d'un microsatellite de type PROBA construit par une entreprise belge, *QinetiQ Space*, à bord duquel sera embarqué l'instrument de mesure.

Il est à noter que deux techniques majeures de prise de mesures en télédétection spatiale existent, l'une consistant à établir la distribution horizontale du volume total d'un certain gaz. D'une manière plus imagée et simplifiée, cela consiste à ce que le satellite prenne une « photo » de la stratosphère en regardant vers le sol afin de se rendre compte de la répartition du gaz à la surface de la terre (donc selon les latitudes et longitudes, comme c'est le cas pour la figure 11 p.14).

La deuxième méthode consiste à déterminer la concentration verticale de gaz de façon à avoir une sorte de vue en coupe de la stratosphère, dans laquelle il sera possible de voir quels gaz sont présents, dans quelle proportion et à quelle altitude. Pour ce faire, une « photo » de l'atmosphère avec un appareil parallèle au sol sera prise, permettant d'obtenir un résultat similaire à la figure 13 p. 15.

Ainsi, le but scientifique du projet consisterait notamment à connaître la proportion de certains gaz, qui régissent le climat de la Terre, à un niveau supérieur de notre atmosphère. Plus précisément, les gaz tels que l'ozone, les molécules responsable de la destruction de l'ozone (dioxyde d'azote, nitrate ...), les gaz à effet de serre (vapeur d'eau, méthane ...) ainsi que les aérosols seront ciblés par la mission. Il sera également possible de déterminer les zones nuageuses de la stratosphère.

Le sondage vertical de la stratosphère peut être réalisé de deux façons majeures. Dans les deux cas, l'appareil de mesure captera la lumière provenant d'un corps céleste traversant la zone à analyser (ici, le limbe). La méthode dite d'occultation solaire consiste à positionner le satellite face au soleil et sur le chemin direct des rayons solaires, comme le montre la figure suivante :

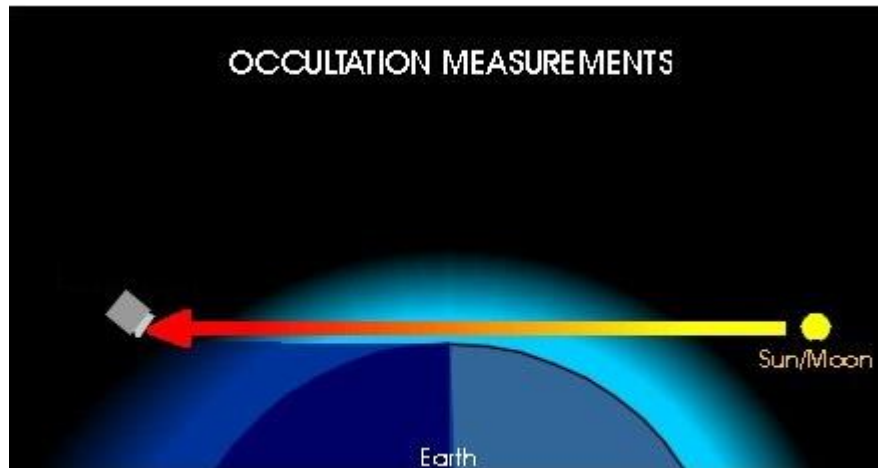


Figure 14 : explication schématique de la technique d'occultation solaire

Avec cette technique, le satellite n'observe que le lever ou le coucher du soleil à travers le limbe atmosphérique. L'avantage de ce type de mesure est que l'instrument recevra les rayons du soleil les plus intenses ce qui permettra une acquisition d'images de plus haute résolution. Mais le nombre de mesures est limité par le nombre de lever et de coucher de l'astre solaire que verra le satellite. Toutefois, la méthode peut être améliorée en considérant d'autres sources lumineuses telles que les étoiles, la lune ou d'autres planètes. En revanche, ces sources émettent des rayons lumineux relativement faibles par rapport à ceux du soleil et peuvent posséder une couleur prédominante, ce qui pourrait fausser les résultats de mesure.

L'autre méthode est appelée « technique de diffusion au limbe » et consiste, comme la méthode d'occultation, à capter les rayonnements solaires traversant le limbe. La différence est que le satellite n'a pas besoin de se placer sur le chemin direct du soleil car il est en mesure de capter les rayons dispersés par le limbe (voir figure suivante).

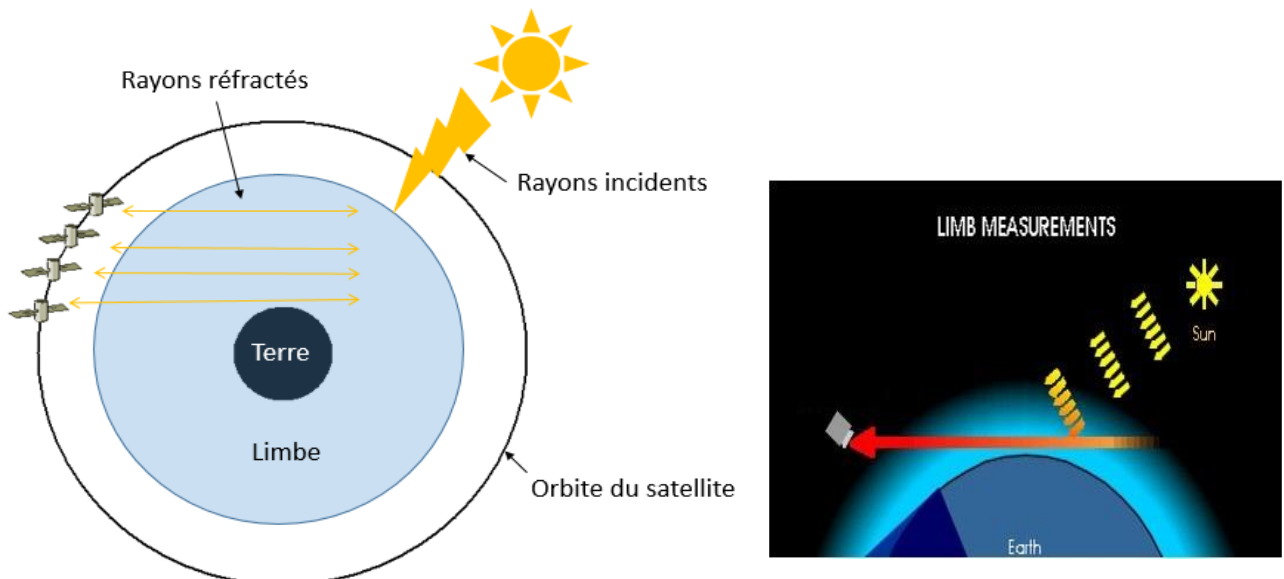


Figure 15 : explication schématique de la technique de diffusion au limbe

Grâce à cet aspect, la prise de mesure peut être effectuée quasiment en continue. Le bémol est que, comme le satellite n'est pas orienté dans la direction d'une source de lumière dont les coordonnées sont connues, il est difficile de déterminer d'où provient le rayon réfracté capté par le satellite. Effectivement, étant donné que le satellite scanne le limbe tranche par tranche (voir

figure 15 p.17), il n'y a aucun point de repère permettant de donner une idée sur l'origine du rayon dispersé par le limbe. Ceci pose un problème pour déterminer à quelle altitude a été capté le rayon.

ALTIUS sera capable d'effectuer les deux types de prise de mesure, malgré l'inconvénient inhérent à la méthode de dispersion au limbe. Ce souci a même pu être surmonté grâce à une idée tout à fait inédite. En effet, plutôt que de scanner le limbe petit bout par petit bout, l'instrument embarqué à bord du satellite prendra une photo de tout le limbe, en s'assurant que l'image comprenne également un morceau de la Terre ainsi que la partie basse de la mésosphère (zone de transition entre la Terre et l'espace). Le fait d'intégrer dans la photo une petite partie de la Terre permet d'avoir un point de référence correspondant à l'altitude 0 m. De plus, l'altitude maximale est délimitée par la limite entre la stratosphère et la partie noire de l'atmosphère (voir figure 16).

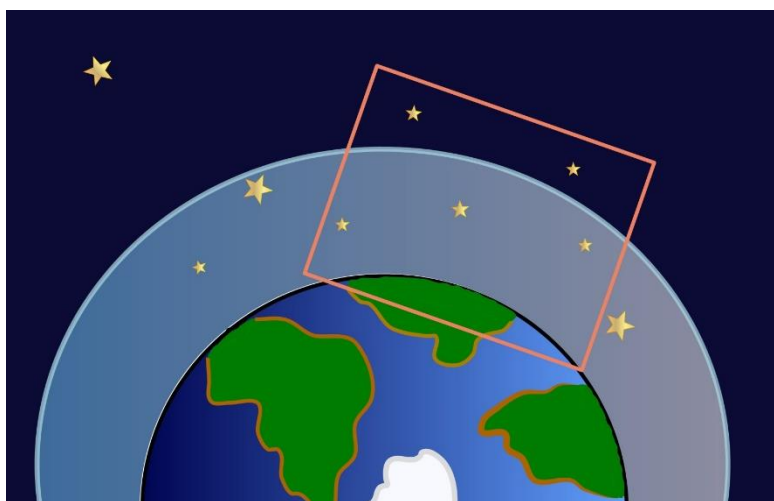


Figure 16 : schéma explicatif du principe de fonctionnement d'ALTIUS

Le limbe ainsi délimité, sa hauteur peut être connue de façon certaine et il sera possible d'attribuer à chaque pixel du détecteur de la caméra, une altitude précise. Le profil vertical de la stratosphère pourra ainsi être constitué.

La pièce maîtresse du projet sera la caméra spectrale dont la fonction est de mesurer l'intensité de la lumière incidente en fonction de la longueur d'onde.

Il est possible de découper le spectre électromagnétique en plusieurs bandes, selon les longueurs d'ondes, et d'en sélectionner une en particulier afin d'en réaliser une analyse détaillée. Une caméra spectrale permet ainsi d'enregistrer les rayonnements de ces bandes, y compris celles situées en dehors du champ visuel humain. Dans notre cas, la caméra spectrale prendra une photo suivant trois bandes : UV, proche infrarouge (NIR : *Near Infrared*) et visible (VIS). Les longueurs d'ondes ciblées par le projet sont : entre 250 et 450 nm pour la bande UV, 900 – 1800 nm pour le proche infrarouge et 440 – 800 nm pour la lumière visible.

Le spectre récupéré par la caméra correspondra à un spectre de raies d'absorption car les molécules présentes dans l'atmosphère absorberont certaines longueurs d'ondes. Par exemple, si le détecteur perçoit une réduction de l'intensité de la couleur rouge, l'explication pourra provenir de la présence d'ozone qui absorbe majoritairement cette couleur (aux alentours d'une longueur d'onde de 600 nm). De la même façon, la présence ainsi que la concentration d'autres gaz pourront être déterminées. La figure suivante est un exemple de spectre de raies d'absorption :

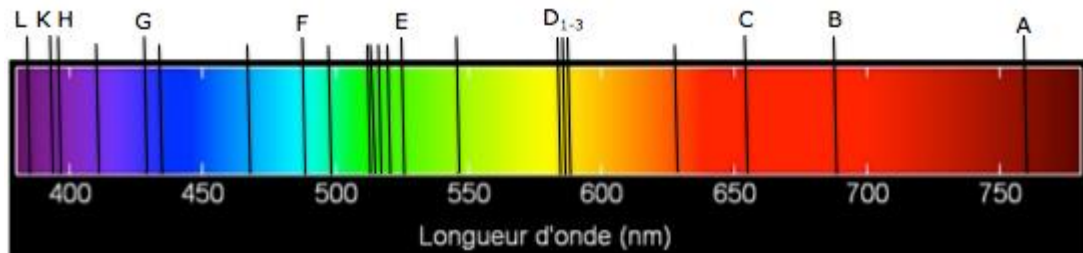


Figure 17 : exemple de spectre de raies d'absorption

Pour couvrir les trois bandes du spectre électromagnétique, l'instrument de mesure sera en réalité divisé en trois chaînes d'acquisition, chacune prévue pour recevoir les longueurs d'ondes spécifiques d'une certaine bande. Les trois sous-systèmes seront des caméras spectrales composées d'un filtre acousto-optique, d'un détecteur 2D, d'un jeu de lentilles optiques ainsi que de l'ensemble de l'électronique nécessaire.

L'utilisation d'un filtre acousto-optique (AOTF : *Acousto-Optic Tunable Filter*) est tout indiqué pour le projet car il agit un peu à la manière d'un prisme très précis et contrôlable à distance. Il permet donc de décomposer un rayon de lumière.

Un tel composant est constitué d'un cristal biréfringent et d'un transducteur piézoélectrique. En réponse à un signal radiofréquence oscillant, le transducteur entrera en vibration et émettra une onde acoustique à haute fréquence qui traversera le cristal. En se propageant à travers le cristal, cette onde acoustique aura pour effet de modifier les propriétés physiques du matériau, et plus précisément son indice de réfraction. En effet, l'onde acoustique provoque une compression/décompression locale du matériau ce qui résulte en une modulation périodique de l'indice de réfraction.

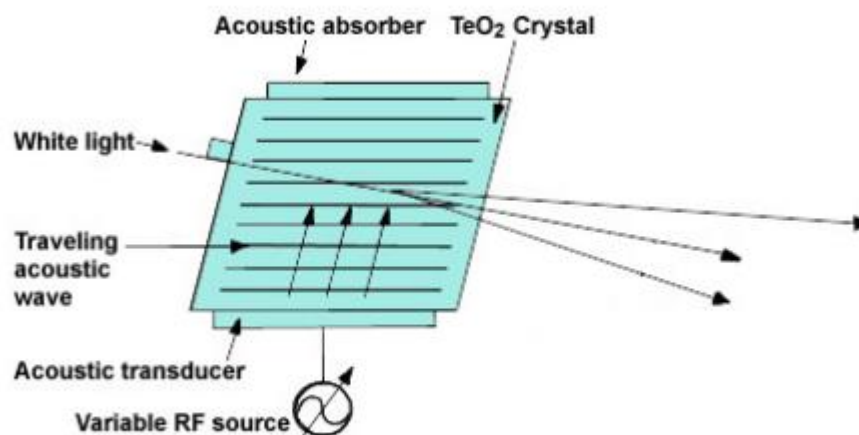


Figure 18 : schéma du principe de fonctionnement d'un AOTF

Les caractéristiques de l'onde acoustique traversant le cristal dépendent directement de la fréquence du signal RF appliqué au transducteur. Ainsi, par extrapolation, le choix méthodique de la fréquence du signal RF d'entrée permet de contrôler les propriétés de réfraction du cristal.

Ce type de filtre fonctionne à la façon d'un filtre passe-bande optique car seule une petite fraction du rayon lumineux incident respectera les conditions de diffraction du cristal. La fréquence du signal RF permet de régler la fréquence centrale de la bande de fonctionnement

du filtre afin de sélectionner des longueurs d'ondes précises, d'après l'équation : $\lambda = \Delta n * \frac{v}{f}$, avec Δn la biréfringence du cristal, v la vitesse de l'onde acoustique et f la fréquence.

Dans notre cas, afin de sélectionner nos trois canaux de fréquences, le signal RF appliqué au transducteur devra satisfaire les gammes de fréquences suivantes :

- VIS : 72 – 156 MHz.
- NIR : 46 – 96 MHz.
- UV : 130 – 260 MHz.

L'optique présente au sein du système est un jeu complexe de lentilles et miroirs permettant de guider les rayons de lumière à travers l'instrument jusqu'au détecteur (voir figure 19 p.20).

Au final, les faisceaux diffractés par l'AOTF arrivent jusqu'à un capteur 2D, similaire aux détecteurs présents dans les appareils photos numériques. Ce détecteur permet de convertir un faisceau lumineux en signal électrique interprétable électroniquement.

La figure ci-après représente l'intégration de l'ensemble des éléments cités précédemment, permettant de réaliser la caméra spectrale fonctionnant dans la bande de lumière visible.

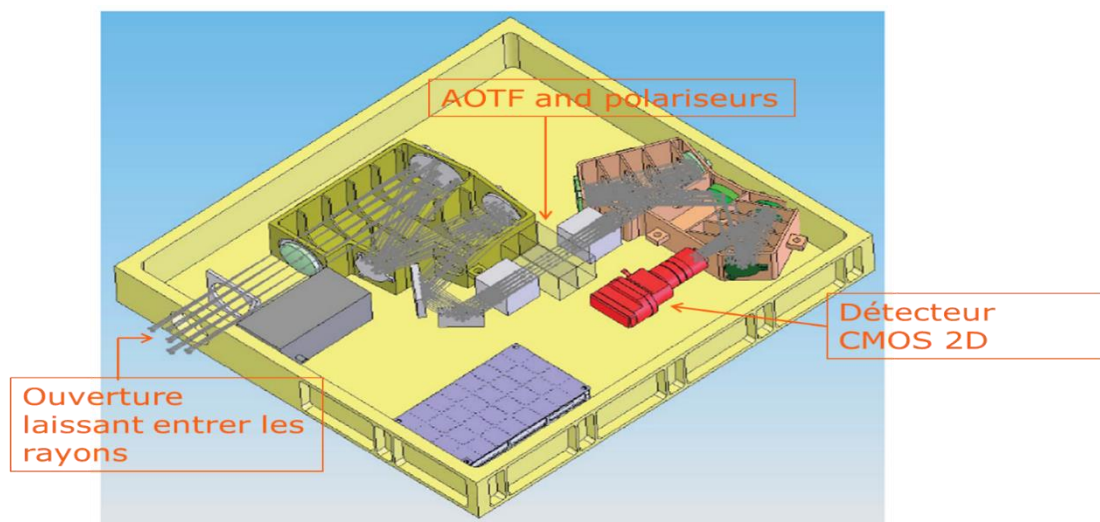


Figure 19 : caméra spectrale pour le domaine de la lumière visible

Comme indiqué sur la figure, le détecteur 2D est de type CMOS ce qui signifie qu'il est composé de photodiodes connectées à des transistors CMOS. Un tel composant permet ainsi de convertir des photons lumineux en tension.

La mise en commun des trois caméras spectrales pour former l'instrument de mesure final d'ALTIUS est représentée en figure 20 :

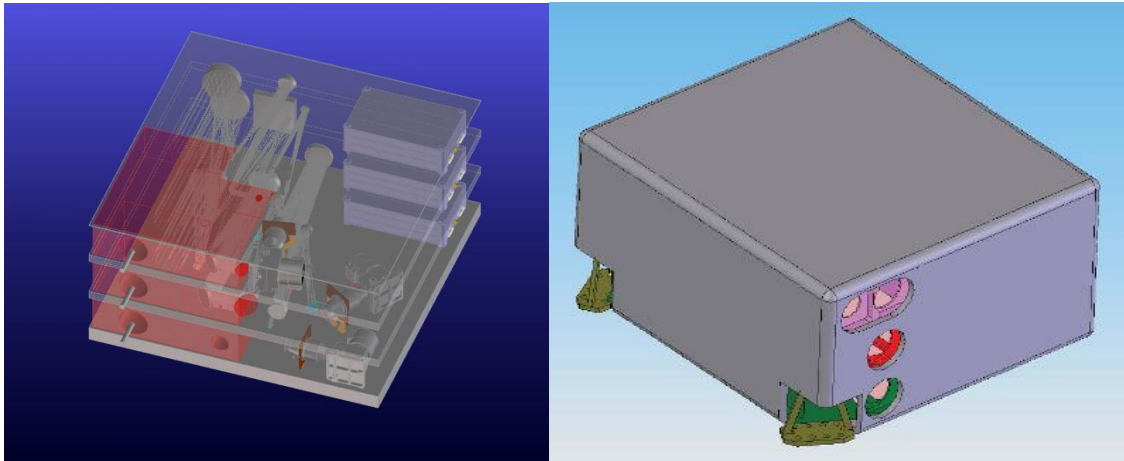


Figure 20 : vue schématique de la mise en commun des trois caméras spectrales

Enfin, ce boîtier sera intégré au microsatellite mentionné auparavant qui comportera d'autres équipements nécessaires au bon fonctionnement et à l'autonomie du satellite. Ces équipements regroupent des panneaux solaires pour assurer l'alimentation électrique des systèmes électroniques, un système de géolocalisation utilisant les étoiles, une batterie ou encore des antennes. Ci-après, la figure présente une vue schématique du satellite dans son ensemble (schéma provenant de la société produisant ces satellites : *QinetiQ Space*) :

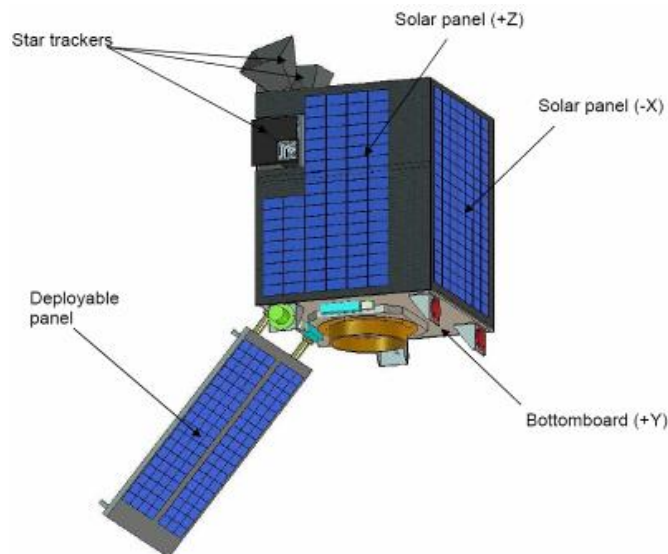


Figure 21 : vue extérieure du satellite utilisé pour la mission ALTIUS

Le satellite sera placé à une orbite de 668,5 km de hauteur par rapport au sol qui sera entièrement parcourue par l'engin spatial en 100 minutes. Par conséquent, un balayage complet de la surface de la terre pourra être achevé en 3 jours.

Après avoir donné une vue d'ensemble de la mission spatiale, nous pouvons désormais nous intéresser à la tâche qui m'a été confiée au sein de ce projet. Pour cela, il faut revenir au filtre acousto-optique dont nous avons discuté précédemment.

Tout l'intérêt du projet ALTIUS repose sur l'acquisition de spectres de lumière précis dans les trois bandes du spectre électromagnétique désirées. C'est pourquoi il faudra parvenir à isoler toutes les longueurs d'ondes intéressantes au sein de chacune des bandes. Cela pourra être envisageable en construisant une chaîne RF qui se chargera de générer plusieurs signaux RF à des fréquences précises pour piloter l'AOTF et sélectionner les longueurs d'ondes

souhaitées.

b. Particularité de la chaîne RF

Plusieurs solutions ont été envisagées par les instigateurs du projet concernant le système permettant de générer les fréquences désirées. Les solutions sont notamment tournées vers l'utilisation de FPGA (*Field Programmable Gate Arrays*), de DDS (*Direct Digital Synthesizers*), voire même une combinaison des deux.

Une vue schématique très simplifiée de la chaîne RF est proposée ci-après :

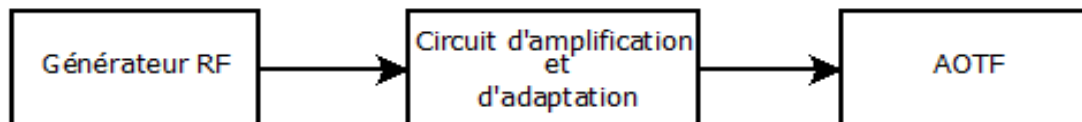


Figure 22 : vue schématique simplifiée de la chaîne RF

Pour la phase de test, le rôle de la chaîne RF consistera à générer de hautes fréquences (de l'ordre du Gigahertz) puis de les abaisser grâce à des diviseurs de fréquence jusqu'à obtenir les fréquences finales désirées (bandes UV, NIR et VIS). En effet, il sera plus commode de déterminer une unique solution pour le générateur RF qui pourra servir pour les trois canaux de fréquences. Il n'y aura alors qu'à choisir des diviseurs de fréquence différents pour obtenir les trois gammes de fréquences voulues.

Ainsi, plusieurs hypothèses ont été proposées mais seulement quelques-unes ont pu être validées théoriquement ou en simulation. Il incombe ensuite aux ingénieurs de l'institut de tester les solutions retenues afin de pouvoir sélectionner celle qui répond le mieux au cahier des charges.

Tout au long du processus de test, il faudra également garder en mémoire un paramètre important qui limite drastiquement les possibilités : il s'agit de la nécessité d'une qualification spatiale du système utilisé. En effet, les appareils électroniques ayant pour vocation d'aller dans l'espace, ils doivent être peu sensibles aux radiations et être en mesure de résister aux vibrations extrêmes causées lors du décollage de l'appareil de mise en orbite. Il faudra également que le système soit peu bruyant (du point de vue du rayonnement électromagnétique) vis-à-vis d'autres appareils embarqués à bord du satellite. En considérant l'ampleur des paramètres à contrôler en vue de qualifier un composant électronique, le coût des tests revient relativement cher aux entreprises d'électronique qui sont peu nombreuses à délivrer l'accréditation. En ce qui concerne les FPGA, seules deux entreprises délivrent des produits porteurs de la qualification.

Une autre caractéristique limitant les choix est le besoin de générer de hautes fréquences de façon à couvrir les trois bandes de mesures, comme énoncé précédemment. Or, plus la fréquence augmente, plus il devient difficile de trouver un système répondant à la fois aux besoins en terme de résolution, de puissance, de rapidité ou encore de stabilité.

2) Sujet

Le rôle de l'équipe d'ingénieurs de l'IASB est de concevoir, produire et tester l'ensemble de la chaîne RF pour les canaux UV, VIS et NIR du projet ALTIUS.

Le contenu de mon stage prévoit que je travaille sur une petite partie de l'électronique de

la chaîne RF qui permettra de tester des solutions envisagées par l'équipe d'ingénieurs concernant le générateur RF.

Actuellement, quelques solutions ont été validées théoriquement et en simulation. Il restera à les tester expérimentalement pour déterminer laquelle respectera au mieux le cahier des charges du projet.

Pour ma part, j'aurai l'occasion de travailler sur deux solutions. La première solution consiste en l'utilisation d'un DDS comme synthétiseur de fréquence alors que la seconde solution est plutôt tournée vers l'utilisation d'un montage constituant une PLL (*Phase Locked Loop*) complète.

Mes tâches seront alors d'implémenter une interface graphique qui permettra à un utilisateur de commander le synthétiseur de fréquence à partir d'un ordinateur. Il sera également à ma charge de programmer l'interface faisant le lien entre l'ordinateur et le composant électronique à commander.

3) Objectif final

L'objectif final est de parvenir à piloter l'AOTF avec les bonnes fréquences, afin de pouvoir isoler les longueurs d'ondes voulues.

Il est donc nécessaire de déterminer la meilleure solution concernant la chaîne RF au niveau expérimental. C'est-à-dire qu'il faudra acquérir le matériel *hardware* « basique » disponible dans le commerce afin de construire une chaîne RF expérimentale qui sera en mesure d'interagir avec l'AOTF.

Ainsi, les deux solutions concernant le générateur RF citées précédemment seront combinées à des circuits d'adaptation et d'amplification et le signal de sortie sera présenté à l'entrée de l'AOTF, avant d'évaluer le comportement de ce-dernier.

L'étape intermédiaire consiste à parvenir à programmer convenablement le synthétiseur de fréquence afin de commander l'AOTF.

A terme, une fois la ou les meilleures solutions validées expérimentalement, il faudra dénicher le matériel *hardware* répondant aux normes spatiales pour établir une chaîne RF qui se rapproche le plus possible de celle pouvant être embarquée directement à bord du satellite. Il est probable que la chaîne RF finale soit intégrée dans un FPGA qui sera en charge de contrôler l'ensemble de l'électronique présente dans la caméra spectrale.

4) Contraintes

- Contraintes *hardware* : à mon arrivée à l'institut du matériel hardware avait déjà été acquis ou commandé. En effet, une carte d'évaluation de l'entreprise *Analog Devices* mettant en jeu un DDS venait tout juste d'être acquise et quelques tests avaient déjà pu être conduits dessus. Cette carte correspond à l'AD9910 *evaluation board*.



Figure 23 : photo de l'AD9910 evaluation board

De plus, le schéma électrique ainsi que le PCB *design* d'une carte électronique constituant une PLL complète avaient déjà été réalisés par l'IASB. Par ailleurs, ce même PCB était en cours de production par une entreprise spécialisée qui prévoyait de livrer la carte à la mi-juillet. Le composant principal de cette carte est une PLL programmable d'Analog Devices : l'ADF4108.

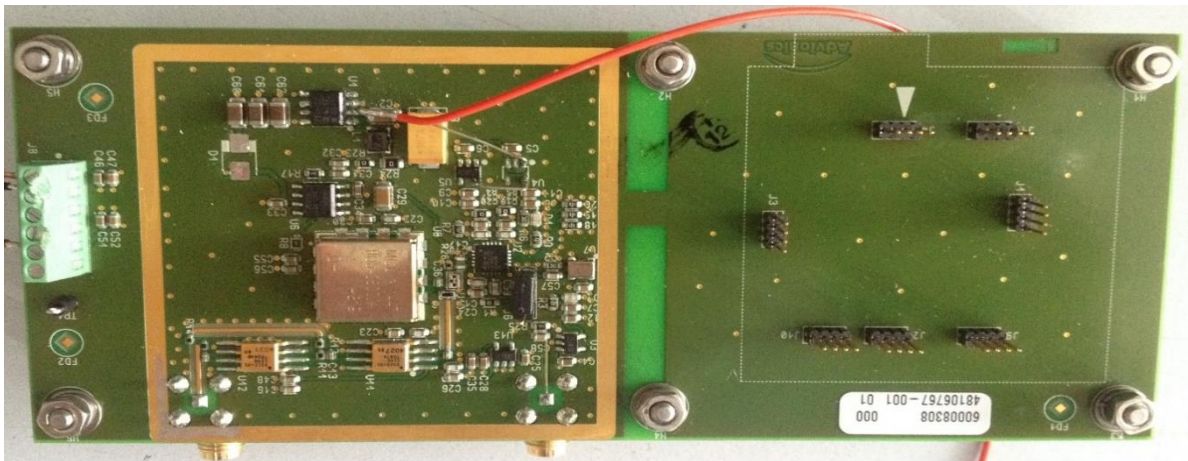


Figure 24 : photo de la carte électronique de l'IASB constituant une PLL complète

En outre, les tests ayant été conduits sur le DDS ont été réalisés à l'aide d'un module USB faisant le lien entre un ordinateur et la carte électronique : c'est le PIC18F46J50 *FS USB PIM DEMO BOARD* produit par Microchip. Il a également été prévu que ce module soit utilisé pour la carte électronique à base de PLL.

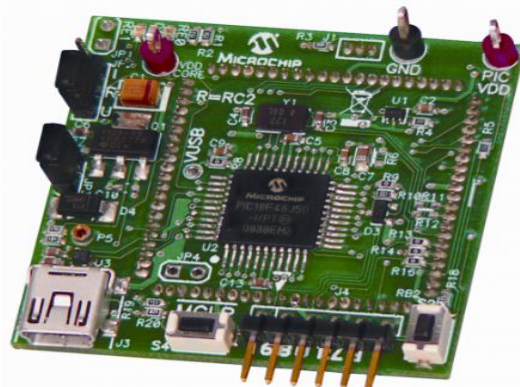


Figure 25 : photo du PIC18F46J50 FS USB PIM DEMO BOARD

- Contraintes *software* : le module USB ayant un microcontrôleur de type PIC, il devra être programmé en utilisant le langage C embarqué par le biais de l'une des interfaces de développement de *Microchip* qui sont MPLABX IDE et MPLAB IDE. Or, pour utiliser les archives de *Microchip*, contenant des projets complets (notamment pour des applications USB) les plus récentes, il convenait d'utiliser MPLABX IDE.
- Contraintes temporelles : à la fin de mon stage (début septembre), il est attendu que l'interface graphique ainsi que le module USB fonctionnent en corrélation et permettent de programmer efficacement le composant électronique cible (DDS et PLL).

5) Moyens

Les moyens mis à ma disposition lors de mon stage rejoignent ce qui a été énoncé dans la sous-partie précédente. En effet, l'ensemble du matériel *hardware* (cartes électroniques) m'a été octroyé ainsi qu'un ordinateur portable sur lequel était installé MPLABX IDE.

De plus, l'ingénieur ayant déjà travaillé sur le DDS m'a fait partager ses travaux et a apporté beaucoup de réponses à mes interrogations, ce qui m'a permis de me mettre rapidement au travail.

6) Planning d'organisation

- Construction et test des différents sous-systèmes de la chaîne RF avec des composants porteur de l'accréditation spatiale : 2017.
- Intégration de la chaîne RF dans l'instrument de mesure final. Test de l'ensemble du système : fin 2017.
- Lancement du satellite : peut-être en 2018 ?

7) Travail réalisé

a. Travail effectué sur le DDS

➤ Généralités sur les DDS

La synthèse numérique directe de fréquence est une méthode consistant à produire un signal analogique à partir d'un signal numérique variant dans le temps. Ce signal numérique sera ensuite présenté à l'entrée d'un convertisseur numérique analogique (CNA) afin de générer le signal analogique de sortie.

La figure suivante montre un schéma blocs d'un DDS typique :

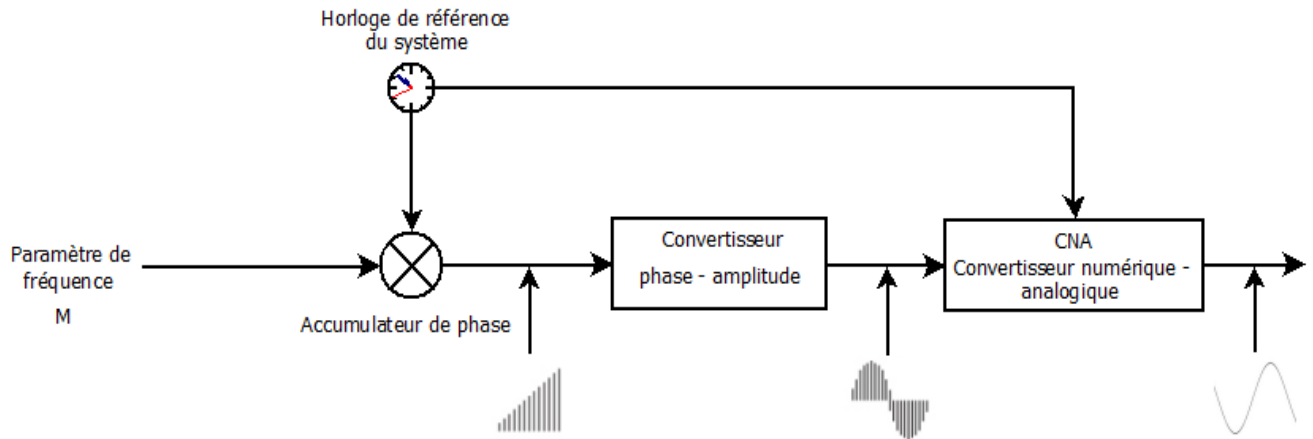


Figure 26 : schéma blocs du circuit interne d'un DDS

L'accumulateur de phase est l'élément essentiel de ce type de montage et est généralement implémenté avec un registre de phase. L'accumulateur permet de créer une sorte de tableau dans lequel sont stockées plusieurs valeurs d'angles. En effet, ce composant peut être vu comme un compteur dont le pas correspond à la valeur du paramètre de fréquence M . Ainsi, à chaque front montant de l'horloge de référence, l'accumulateur incrémente sa valeur de phase courante par M et stocke le résultat dans le registre. La figure suivante permet de mieux se rendre compte du fonctionnement de l'accumulateur :

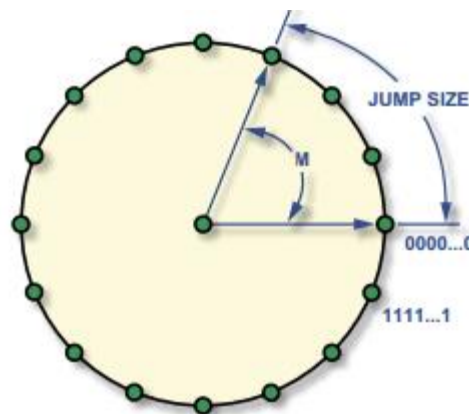


Figure 27 : explication du fonctionnement de l'accumulateur de phase

Cette figure nous indique que le paramètre de fréquence (*tuning word*, M) est en réalité une valeur binaire correspondant à la quantité de phase à ajouter au contenu de l'accumulateur à chaque front montant de l'horloge de référence. Plus simplement, cette valeur M indique à l'accumulateur de phase combien de points il doit sauter en parcourant le cercle trigonométrique avant de relever la valeur de la phase.

L'équation suivante permet de comprendre le rôle du paramètre de fréquence M dans la génération de la fréquence de sortie : $f_{out} = \frac{M}{2^n} * f_{ref}$ n étant la taille de l'accumulateur de phase en bits.

La valeur de phase est ensuite transmise au convertisseur phase – amplitude. Ce convertisseur correspond à une « table de sinus » (*sine look-up table*) contenant une valeur d'amplitude d'une sinusoïde pour chaque valeur de phase. A partir de cette table sera calculée une valeur d'amplitude de sinusoïde pour chaque angle de phase fourni par l'accumulateur. La figure suivante permet de se représenter visuellement ces propos :

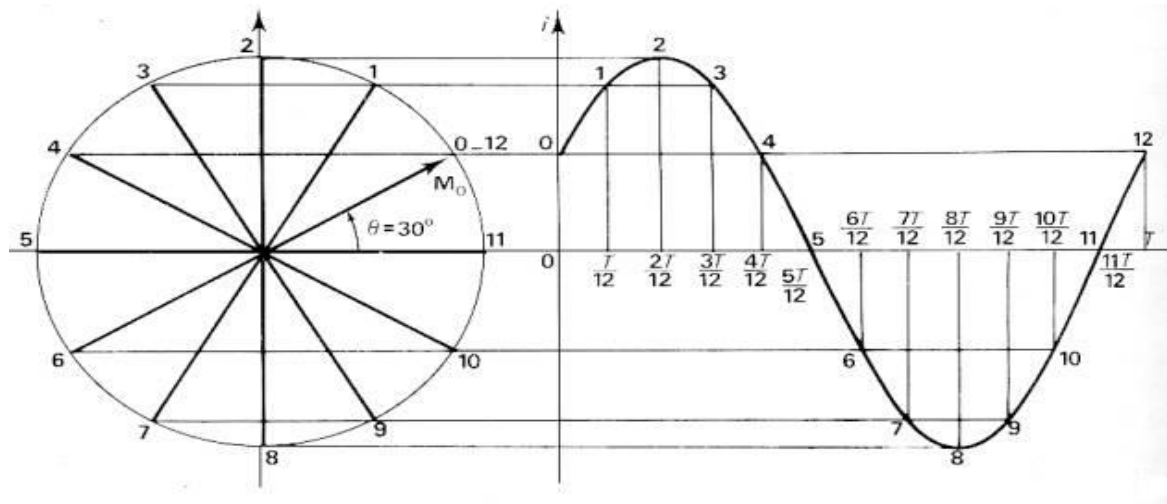


Figure 28 : explication du fonctionnement du convertisseur phase – amplitude

A la sortie du convertisseur de phase en amplitude, nous obtenons un signal sinusoïdal échantillonné, qui correspond toujours à un signal numérique semblable au graphique de droite sur la figure ci-après :

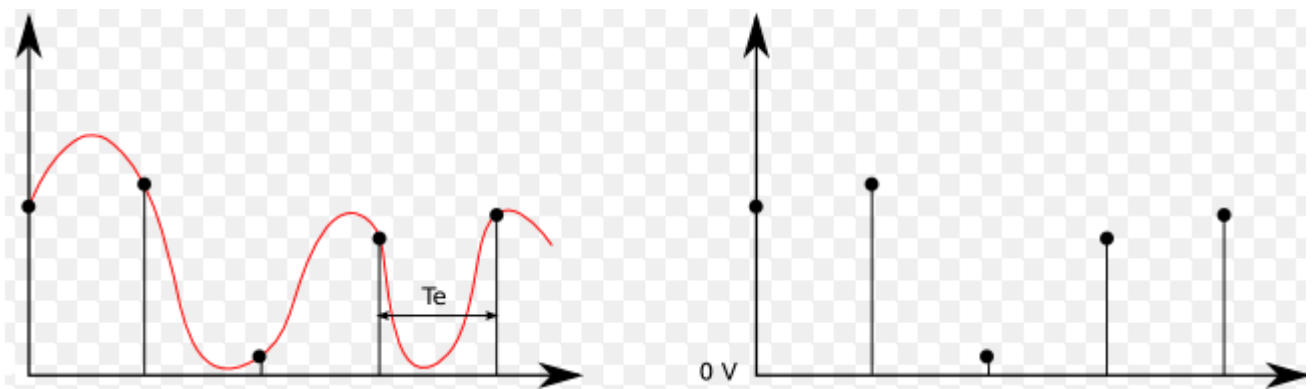


Figure 29 : représentation graphique d'un signal sinusoïdal échantillonné (à droite)

Il est donc nécessaire de présenter ce signal à l'entrée d'un convertisseur numérique – analogique afin d'obtenir une véritable sinusoïde en sortie. C'est la dernière étape du principe de fonctionnement d'un DDS qui consiste à convertir la valeur de l'amplitude fournie par le convertisseur phase – amplitude en valeur analogique de tension ou de courant.

La génération d'un signal ayant une fréquence fixe est donc rendue possible grâce à la valeur du paramètre de fréquence qui est en réalité l'incrément de phase. Plus cet incrément sera élevé, plus le bloc accumulateur de phase/convertisseur phase – amplitude parcourra rapidement la table de sinus. Il en résultera un signal ayant une fréquence élevée en sortie du DDS.

En revanche, si l'incrément de phase est faible, la table de sinus sera parcourue lentement ce qui correspondra à une fréquence faible du signal de sortie.

➤ Tâches effectuées

Normalement, le contenu de mon stage ne prévoyait pas de travailler avec un DDS, mais uniquement avec une PLL. Mais comme les deux travaux se ressemblent beaucoup, il m'a été autorisé et même encouragé de me faire d'abord la main avec le DDS avant de m'intéresser à la mission initialement prévue.

Cette partie des tâches réalisées au cours de mon stage représente donc une sorte de travaux pratiques préliminaires avant de m'attaquer au sujet principal.

Afin de valider, ou non, la solution à base de DDS, un test a dû être mené sur un produit de l'entreprise *Analog Devices* : l'AD9910 *evaluation board*. Ce produit est un circuit électronique complet implémenté autour de l'AD9910 (un DDS conçu par cette même entreprise) qui permet de configurer ce composant à partir d'un ordinateur. Cette carte électronique possède donc une interface USB pour la liaison avec un ordinateur, ainsi que d'autres circuits nécessaires au fonctionnement du DDS, comme la génération du signal d'horloge de référence par exemple.

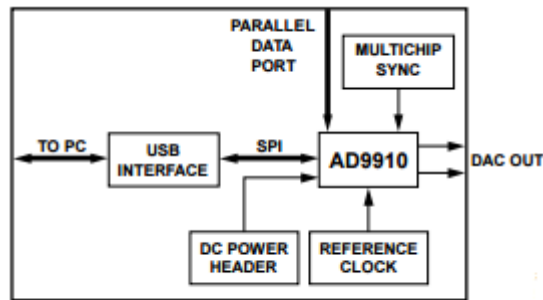


Figure 30 : schéma blocs de l'AD9910 *evaluation board*

Toutefois, il a été décidé que l'interface USB de la carte ne sera pas utilisée car cela nous contraindrait à utiliser uniquement le logiciel fourni avec le produit ce qui s'avérerait être peu modulable pour notre application.

Il est donc nécessaire de créer notre propre logiciel permettant de contrôler le composant à partir d'un ordinateur. Pour ce faire, le module USB de la société *Microchip* a été sélectionné : le PIC18F46J50 USB PIM DEMO BOARD. Ce module est connecté à l'ordinateur par liaison USB et le bus SPI du microcontrôleur est utilisé pour relayer les commandes jusqu'au DDS. Ce module permet ainsi une conversion USB-SPI afin de configurer le DDS dont les registres internes peuvent être adressés par le biais d'un protocole de communication série.

Deux programmes doivent donc être écrits : l'un permettant de concevoir l'interface graphique de contrôle sur l'ordinateur et l'autre permettant de programmer le microcontrôleur du module USB.

Mais avant toute chose, la première étape de mon travail consiste à parcourir les différentes documentations techniques concernant la carte électronique utilisée. Ces documentations regroupent la *datasheet* de l'AD9910 et le manuel d'utilisation de l'AD9910 *evaluation board*.

En outre, j'ai dû mener des recherches concernant le principe de fonctionnement des DDS étant donné que je n'ai pas encore eu l'occasion de traiter ce sujet au cours de mes études. Ces recherches ont été accompagnées de nombreuses prises de notes et questions posées aux ingénieurs présents dans le service.

❖ Programmation du microcontrôleur du module USB :

Une fois ce travail documentaire réalisé, je me suis attaqué à la programmation du microcontrôleur sur MPLAB X IDE. Pour ce faire, je me suis rendu sur le site web de la société afin de télécharger une archive comprenant plusieurs codes sources pouvant être directement utilisés dans un projet. Nous retrouvons ainsi des codes de démonstration de l'utilisation d'une pile TCP/IP, par exemple.

Cette archive comprend également plusieurs projets utilisant la pile USB, dont un projet permettant de faire tourner cette pile directement sur le PIC18F46J50 *USB DEMO BOARD*.

Afin de comprendre comment le programme fonctionne en vue de le modifier par la suite, j'ai parcouru le code source en localisant les fonctions potentiellement intéressantes pour mon application. J'ai ensuite programmé le module USB directement avec le code source, sans modifications, afin de tester la communication USB avec un ordinateur.

L'utilisation du logiciel *Docklight* m'a permis d'effectuer des tests basiques me permettant de savoir si la pile USB tournait convenablement sur mon module.

La lecture du code source m'avait permis de localiser une fonction permettant de renvoyer un message sur le port USB après un appui sur un bouton poussoir présent sur module USB. Une autre fonction permet, après l'envoi d'un caractère depuis l'ordinateur, de renvoyer le caractère directement supérieur (d'après le codage ASCII). J'ai ainsi pu tester ces deux fonctionnalités et voir apparaître les résultats attendues sur *Docklight* :

```
Button pressed.<CR><LF>
<NUL>
08/06/2016 09:18:50.455 [TX] - a
08/06/2016 09:18:50.583 [RX] - b
```

Figure 31 : tests de la liaison USB avec le logiciel *Docklight*

Les messages affichés en rouge correspondent à une réception (liaison allant du module USB vers l'ordinateur) et les messages en bleus correspondent à une transmission (commande envoyée depuis le logiciel vers le module USB).

Ensuite, j'ai commencé à modifier le code source afin d'implémenter un petit test me permettant de me familiariser avec le microcontrôleur et l'environnement de développement. Ce test consistait en l'allumage d'une LED sur appui d'un bouton poussoir.

Une fois ce petit test simple validé, j'ai entrepris d'initialiser la communication SPI qui permettra, par la suite, d'acheminer les commandes de l'ordinateur vers le DDS.

Afin d'initialiser la communication SPI, il convient de suivre pas à pas les recommandations écrites dans la *datasheet* du microcontrôleur dans la partie consacrée aux protocoles de communication série.

Plusieurs étapes étaient à suivre :

- Initialiser la direction des ports de chaque signal du protocole SPI (entrée ou sortie). Par exemple : `TRISBbits.TRISB4 = 0` ; pour indiquer au microcontrôleur que le port B4 (correspondant à SCLK, le signal d'horloge de référence de la communication SPI) sera une sortie.
- Déterminer quel mode SPI devra être utilisé. Pour cela, il était nécessaire de se référer aux chronogrammes de la liaison SPI présents dans la *datasheet* de l'AD9910.

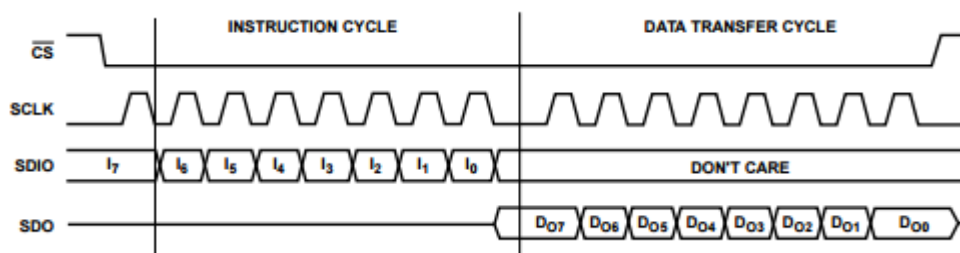


Figure 32 : chronogrammes des signaux impliqués dans la communication SPI de l'AD9910

Dans notre application, le signal intitulé SDO (*Serial Data Out*) ne sera pas utilisé car il s'agit d'une communication unilatérale allant de l'ordinateur vers le module USB. Ces chronogrammes nous indiquent qu'un bit de donnée est transmis sur front descendant de l'horloge (comme l'indique la première barre verticale en partant de la gauche). De plus, nous pouvons constater que l'état « actif » de l'horloge correspond à un état logique haut car l'état initial inactif correspond à un état bas.

Ces deux observations nous permettent de déterminer la valeur des bits CKP (*Clock Polarity*, qui définit l'état « actif » de l'horloge) et CKE (*Clock Edge select*, définit le moment de transmission des données par rapport à l'horloge) à placer dans les registres du microcontrôleur appropriés. Ces deux bits définissent le mode SPI à utiliser. Dans notre cas, il s'agira du mode 0,0 correspondant à CKE = 1 (transmission de donnée sur front descendant de l'horloge) et CKP = 0 (état « inactif » de l'horloge à l'état bas).

Une fois le mode déterminé, nous pouvons faire appel à une fonction qui écrira les valeurs souhaitées dans les registres du microcontrôleur concernés. Cette fonction est déjà écrite dans une librairie fournie dans l'archive de *Microchip* et prend en paramètre le mode SPI voulu ainsi que le canal de transfert. Le canal par défaut sera utilisé (canal 1).

- S'assurer que le signal \overline{CS} (*Chip Select*, permet au maître SPI de sélectionner l'esclave avec lequel il souhaite communiquer) soit bien à l'état inactif par défaut (selon le chronogramme) : LATBbits.LATB0 = 1 ; étant donné qu'il s'agit d'un signal inversé, l'état inactif correspond en réalité à un « 1 » logique.
- Un autre signal spécifique au DDS, non présent dans les chronogrammes ci-avant, est utilisé pour la communication SPI. Il s'agit du signal I/O_Update. La trame binaire envoyée par le bus SPI sera stockée dans un *buffer* et son contenu ne pourra être transféré vers les registres actifs du DDS uniquement lorsqu'un front montant de I/O_Update survient. Par conséquent, il convient de s'assurer que ce signal soit bien à l'état bas par défaut : LATDbits.LATD7 = 0 ;

Une fois la communication SPI initialisée, il est possible d'effectuer un envoi de données à travers le bus de communication.

Il convient alors de respecter les chronogrammes de la figure 32 p.30, à savoir :

- Mettre le signal \overline{CS} à l'état actif (correspondant à un « 0 » logique).
- Faire appel à une fonction, déjà disponible dans les librairies, permettant d'envoyer un octet à travers le bus, en fonction de l'initialisation effectuée au préalable. Cette

fonction prendra en paramètre le canal de transfert, un tableau d'octets (contenant les données à envoyer) ainsi que le nombre d'octets à envoyer.

- Mettre le signal \overline{CS} à l'état inactif (« 1 » logique).
- Mettre le signal I/O_Update à l'état haut afin de générer un front montant indiquant au DDS de charger ses registres en fonction du contenu du *buffer* d'entrée.

Afin de vérifier le bon fonctionnement de la communication SPI, j'ai placé dans le tableau 2 octets facilement identifiables à l'oscilloscope, à savoir : 0x55 et 0x5A. Ces valeurs sont facilement identifiables car constituent une série de « 0 » et de « 1 » logiques alternativement. Nous pouvons ensuite visualiser le résultat sur l'écran d'un oscilloscope :

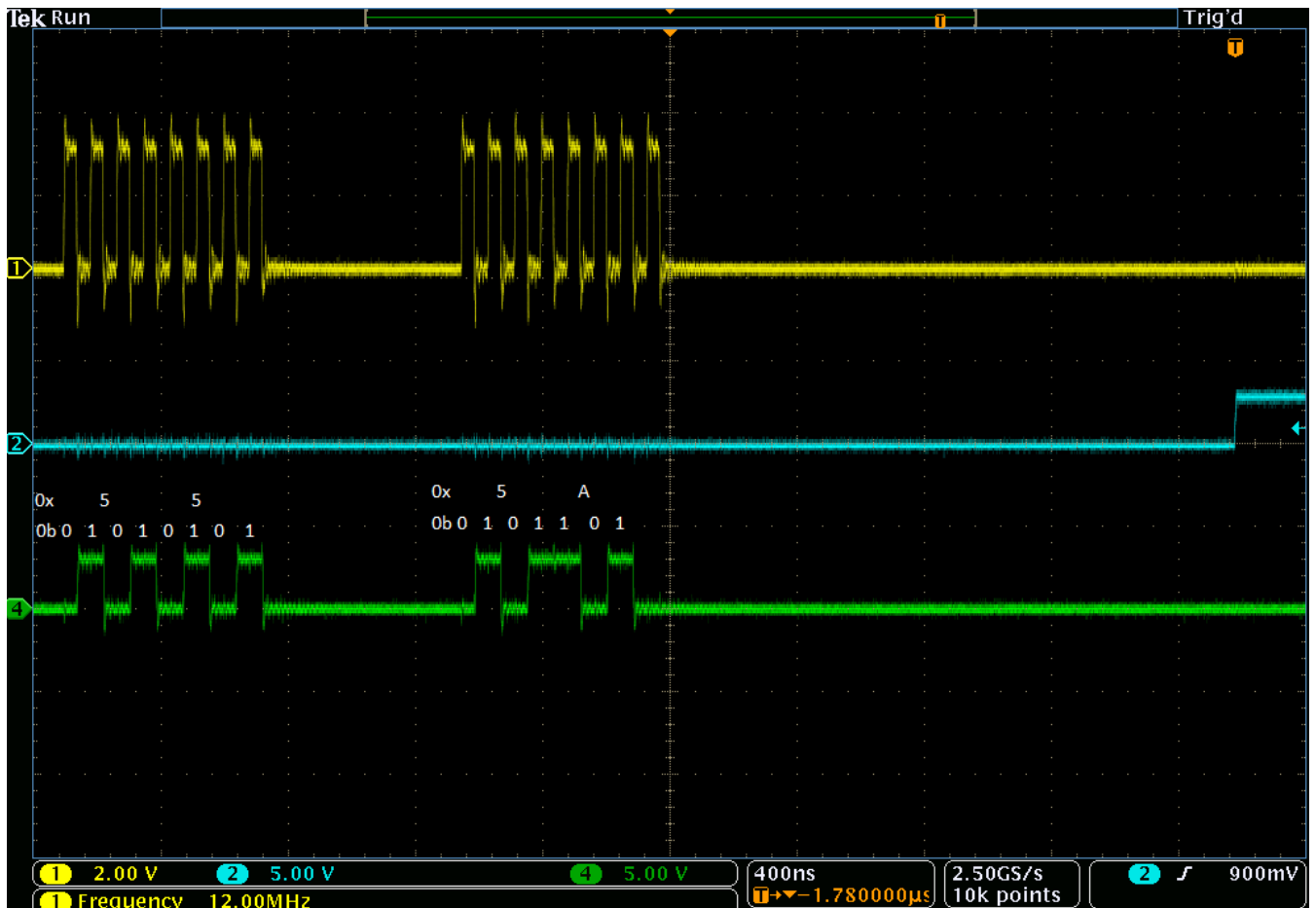


Figure 33 : test basique de la communication SPI à l'oscilloscope

Le signal en jaune représente l'horloge SCLK et nous pouvons constater qu'elle présente bien 8 fronts correspondants à chaque bit de l'octet envoyé. Le signal bleu est le I/O_Update qui se déclenche bien une fois les octets envoyés sur le bus SPI.

Enfin, le signal vert représente les 2 octets envoyés et nous observons le bon enchainement des bits. Nous pouvons également vérifier que cette acquisition est cohérente avec les chronogrammes de la figure 32 p.30 en observant bien que les données sont transmises sur front descendant de l'horloge et que l'état logique de l'horloge par défaut est bien à l'état bas.

Après s'être assuré du bon fonctionnement de la communication SPI pour l'envoi de données, il reste à déterminer de quelle façon les informations provenant de l'ordinateur via USB seront récupérées avant d'être envoyée par le bus SPI.

Pour cela, le code source fourni par *Microchip* contient une fonction permettant de placer les données reçues sur le port USB dans un tableau d'octets : `getsUSBUSART()`. Ce tableau pourra ensuite être envoyé sur le bus SPI en utilisant la même fonction que celle utilisée pour le test précédent (celle permettant d'envoyer un ou plusieurs octets préalablement stockés dans un tableau : `DRV_SPI_PutBuffer()`) :

```
/* Check to see if there is a transmission in progress, if there isn't, then
 * we can see about performing an echo response to data received.
 */
if( USBUSARTIsTxTrfReady() == true)
{
    uint8_t i;
    uint8_t numBytesRead;

    /* We retrieve the bytes sent on the USB bus until the buffer is free.
     * readBuffer : buffer containing the data read on the USBUSART bus.
     * numBytesRead : number of bytes read.
     */
    numBytesRead = getsUSBUSART(readBuffer, sizeof(readBuffer));

    /* As soon as we receive something ... */
    if(numBytesRead > 0){
        enable_CS(); //CS=0 -> DDS selected
        DRV_SPI_PutBuffer (1, readBuffer, numBytesRead); //Send data buffer on SPI bus
        disable_CS(); //CS=1 -> DDS unselected

        disable_IO_Update();

        /* I/O Update is active on a rising edge and its pulse width must be greater than one
         * SYNC_CLK period. I/O Update initiates the transfer of written data from the
         * I/O port buffer to active registers.
         */
        enable_IO_Update();

        disable_IO_Update();
    }
}
```

Figure 34 : réception de données sur le port USB et envoi sur le bus SPI

Une fois l'interface graphique implémentée, il sera possible de visionner les trames binaires à l'oscilloscope. En effet, dès que l'utilisateur changera un réglage sur l'interface graphique, nous pourrons voir les trames binaires apparaître à l'oscilloscope, comme celles de la figure 33 p.31.

❖ Interface graphique :

La communication SPI étant correctement paramétrée et fonctionnant convenablement, j'ai pu commencer à m'intéresser à l'interface graphique.

L'ingénieur de l'institut ayant déjà travaillé sur le DDS avait codé son interface graphique avec Python. J'ai décidé de continuer sur cette lancée car cela me permettait d'approfondir mes connaissances dans ce langage informatique avec lequel je n'étais pas encore très familier.

Plus particulièrement, le module PyQt de Python a été utilisé, permettant de faire le lien entre le langage Python classique et la bibliothèque Qt qui est spécifique à l'interfaçage graphique. Cette bibliothèque regroupe de nombreux objets pouvant être utilisés dans la conception d'interfaces graphiques appelés « *widgets* ». PyQt permet de définir et de disposer ces éléments comme nous le souhaitons au sein d'une fenêtre. Ceci permet de créer une petite

application facile d'utilisation pour un utilisateur quelconque.

L'interface graphique permettra à l'utilisateur de rentrer les valeurs de fréquences et d'amplitude qu'il désire pour le signal de sortie du DDS. Le programme se chargera ensuite, de façon transparente, de transférer les données entrées vers le DDS.

En premier lieu, il fallait réfléchir à une façon claire et efficace pour permettre à l'utilisateur de régler la fréquence et l'amplitude. En parcourant la liste des éléments à disposition dans la bibliothèque Qt, j'ai choisi de placer deux cadrans avec des boutons de réglages. Ces deux cadrans auront également chacun une « *spin box* » (zone de sélection numérique avec incrément/décroissement) associée afin d'afficher la valeur courante du cadran ou de permettre à l'utilisateur de changer la valeur directement via ce champ.

Le résultat obtenu est le suivant :

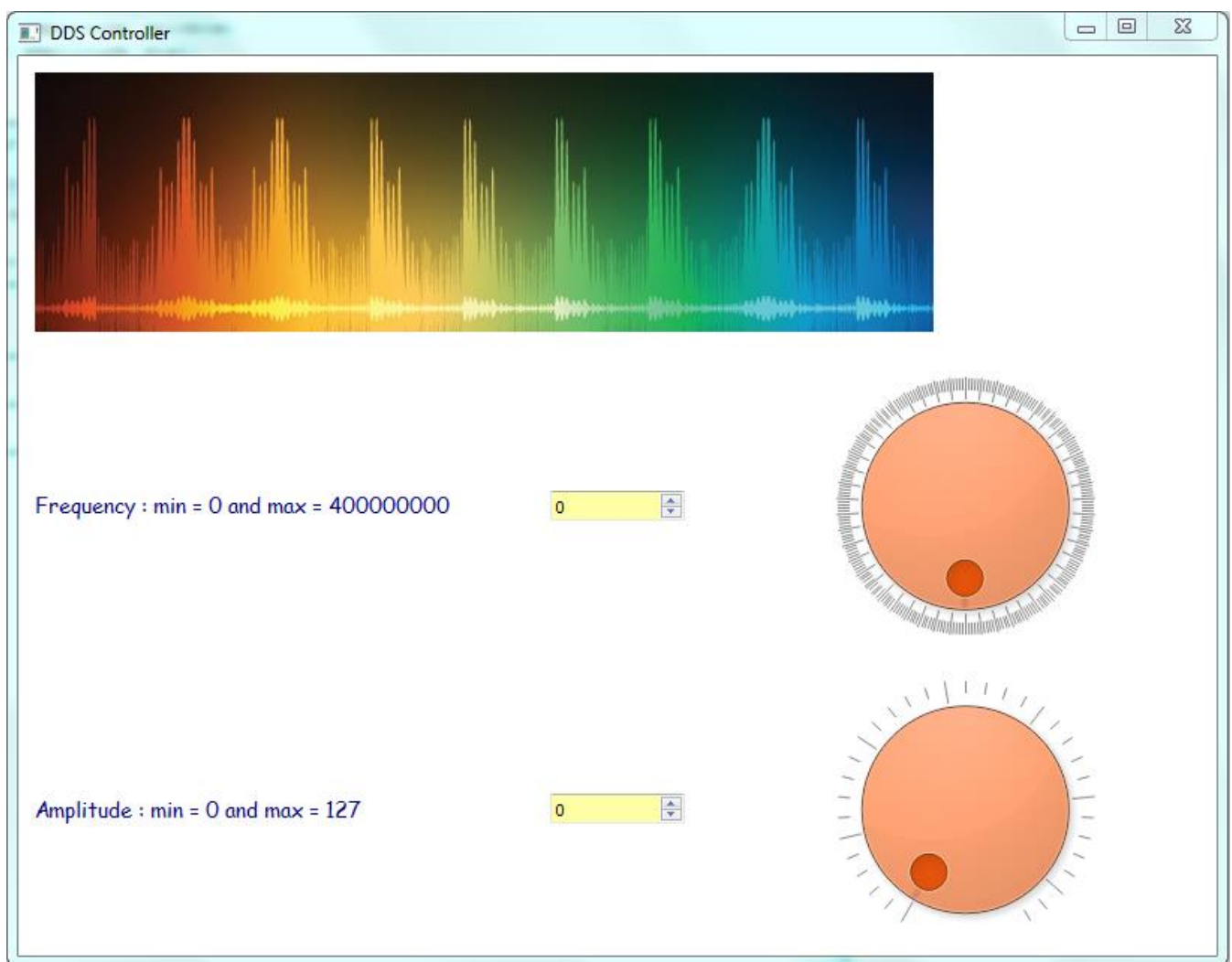


Figure 35 : interface graphique permettant de contrôler le DDS

Les boutons de réglage ainsi que les « *spin box* » sont liées dynamiquement ce qui veut dire que le changement de valeur de l'un implique le changement instantané de la valeur de l'autre. Il s'agit d'une des particularités de PyQt, à savoir que chaque élément graphique de l'interface génère des signaux qu'il est possible de connecter à d'autres éléments de l'interface ou à des fonctions spécifiques. Voici comment ont été connectés les « widgets » entre eux :

```
# Connexion des signaux émis par Les différents objets.
# On veut que Les valeurs des SpinBox et Dial soient reliées
# pour que Le changement de valeur de l'un se répercute sur l'autre
self.connect(self.frequency_dial, SIGNAL("valueChanged(int)"), self.frequency_spinbox.setValue)
self.connect(self.frequency_spinbox, SIGNAL("valueChanged(int)"), self.frequency_dial.setValue)
self.connect(self.amplitude_dial, SIGNAL("valueChanged(int)"), self.amplitude_spinbox.setValue)
self.connect(self.amplitude_spinbox, SIGNAL("valueChanged(int)"), self.amplitude_dial.setValue)
```

Figure 36 : connexion des signaux émis par les "widgets" avec Python

On remarque que le signal émis lors d'un changement de valeur d'un objet se retrouve connecté à un emplacement particulier. Cet emplacement est en réalité la fonction permettant de fixer la valeur d'un autre objet (*setValue*). C'est comme cela que les boutons de réglage et « *spin box* » se retrouvent reliées dynamiquement.

Dès que l'utilisateur change l'un des paramètres de la fenêtre, le programme fait appel à une fonction de traitement. Cette fonction de traitement a été « connectée » à un changement de valeur provenant des « *spin box* », de la même façon que celle montrée dans la figure 36 p.34.

Cette fonction de traitement permet de construire les trames binaires à envoyer à la PLL. C'est la fonction la plus importante du programme :

Fonction envoyant les commandes au DDS

```
In [1]: def send_instructions(freq, amp):

    '''L'instruction "with ... as ..." permet d'ouvrir le port série, de l'utiliser,
    puis de le fermer proprement'''
    with serial.Serial(SERIAL_PORT) as ser :

        '''Création d'une liste contenant des chaînes de caractères
        représentant les valeurs hexadécimales à placer dans les
        registres du DDS.'''
        instructions = ["000000002",
                        "0100400820",
                        "021D3F4150",
                        "03000000%02x" %amp,      # La variable "amp" correspondra à la valeur de l'amplitude entrée par
                                                # l'utilisateur et sera intégrée au registre correspondant.
                        "0E3FFF0000%08x" %freq,  # La variable "freq" correspondra à la valeur de la fréquence entrée par
                                                # l'utilisateur et sera intégrée au registre correspondant.
                        ]

        for hex_instr in instructions:
            data = hex_instr.decode('hex') # Converti Les chaînes de caractères hexa en séries
                                           # d'octets (2 valeurs alphanumériques par octet)
            ser.write(data) # Envoi des données sur le port série
```

Figure 37 : fonction permettant d'envoyer les commandes au DDS

Comme il est possible de le constater sur la figure ci-dessus, la fonction prend deux arguments en paramètres d'entrée qui correspondent aux valeurs de la fréquence et de l'amplitude contenues dans les « *spin box* ».

Il convenait ensuite de se référer à la *datasheet* de l'AD9910 afin de savoir quels registres devaient être fournis au DDS et comment les remplir en fonction de l'utilisation que souhaitions en faire.

Une fois les valeurs hexadécimales des registres déterminées, elles sont placées dans la liste intitulée « instructions ». Il est à noter que deux des registres seront complétés avec les paramètres fournis à la fonction, il fallait alors s'assurer que la valeur placée dans le registre

occupe bien le nombre exact de bits prévus dans le registre. C'est le rôle du « %08x » qui permet d'indiquer au programme que la variable sera une donnée hexadécimale avec 8 caractères alphanumériques. Si jamais cette valeur occupe moins de 8 emplacements, le programme la fera précéder de « 0 » pour conserver le bon format.

Les registres placés dans la liste sont en réalité des chaînes de caractères. Il est donc nécessaire d'extraire les véritables valeurs hexadécimales puis de les convertir en séries d'octets. Les trames envoyées doivent impérativement être des séries de bytes car la liaison SPI du module USB a été paramétrée de façon à recevoir des suites de 8 bits consécutifs uniquement. C'est ce que réalise l'instruction `hex_instr.decode('hex')`.

Lorsque cette conversion est effectuée, nous pouvons envoyer les trames sur le port série souhaité. Ce port série correspond à la constante « `SERIAL_PORT` », dans le programme, dont la valeur est une chaîne de caractères : « `COM5 :` » d'après la convention de nommage de Python.

Afin de pouvoir se rendre compte du bon fonctionnement du programme, j'ai branché la sortie de l'AD9910 *evaluation board* à l'entrée « RF » d'un oscilloscope, qui jouera le rôle d'analyseur de spectre.

Nous pouvons alors visualiser le spectre du signal de sortie et constater la présence d'un pic qui se démarque du niveau de bruit à une certaine fréquence. La fréquence à laquelle est situé ce pic correspond à la fréquence entrée sur l'interface graphique par l'utilisateur :

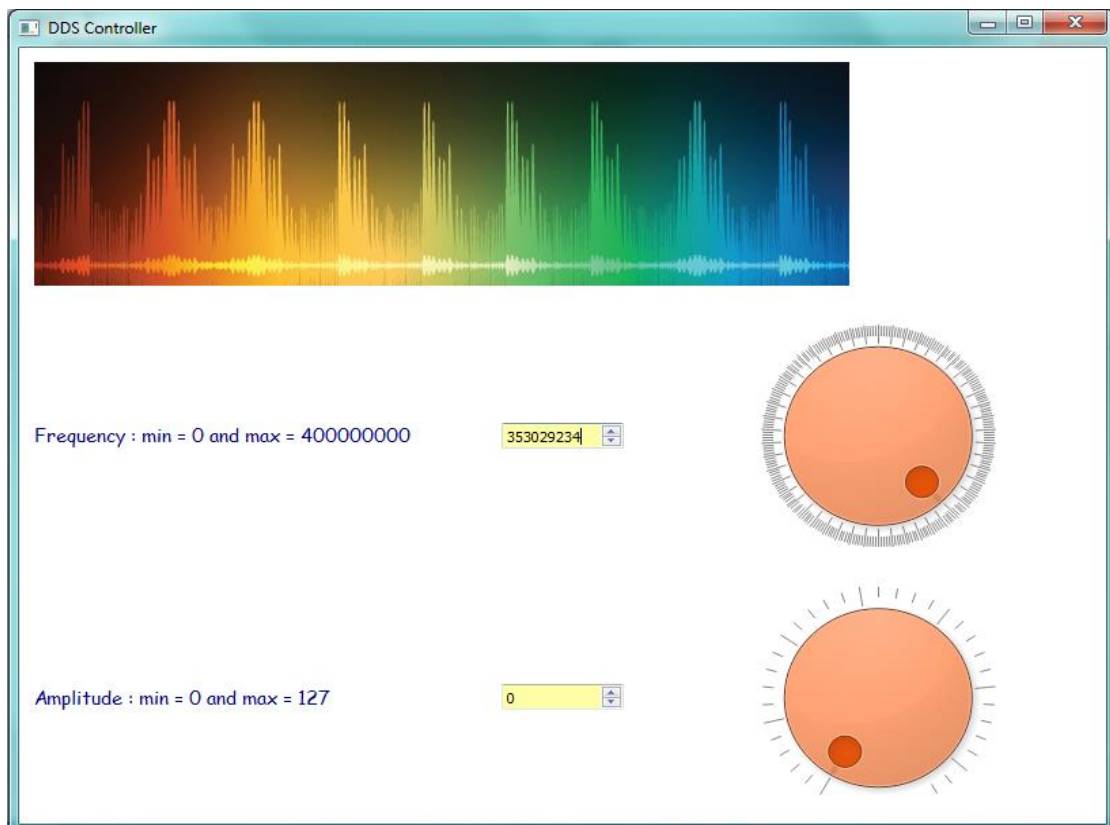


Figure 38 : vue de l'interface graphique lors de la génération d'un signal

Ainsi, à la fréquence de 353 MHz, nous visualiseront le pic de fréquence à l'analyseur de spectre :

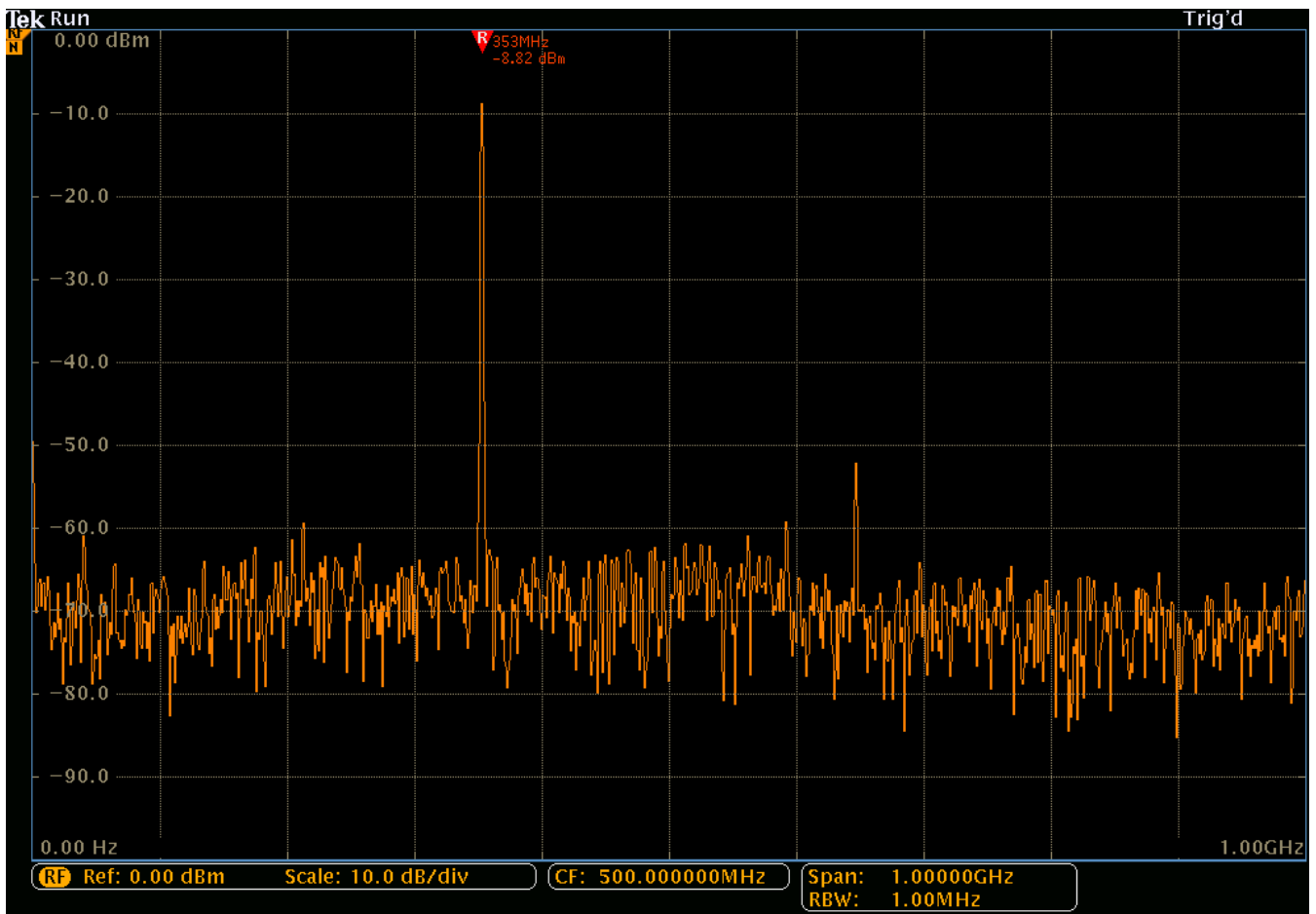


Figure 39 : spectre du signal de sortie du DDS à l'analyseur de spectre pour une amplitude minimale

Le curseur placé au-dessus du pic nous indique bien que la fréquence est de 353 MHz. De plus, ce même curseur nous indique le niveau de puissance maximal du signal qui se situe autour de -9 dBm.

Nous pouvons ensuite tenter de voir l'effet qu'aura un changement d'amplitude sur le signal de sortie. Pour cela, nous passons d'une amplitude de 0 à 127.

Remarque : il est à noter que le réglage de l'amplitude permet de modifier la valeur du courant maximal en sortie du principal convertisseur numérique/analogique de la carte AD9910 *evaluation board*. Ainsi, l'amplitude « 0 » correspond simplement à une valeur de courant minimale, ce qui n'implique pas que l'amplitude du signal soit effectivement nulle mais simplement que le convertisseur fonctionne à son niveau normal.

Nous observons le signal à l'analyseur de spectre :

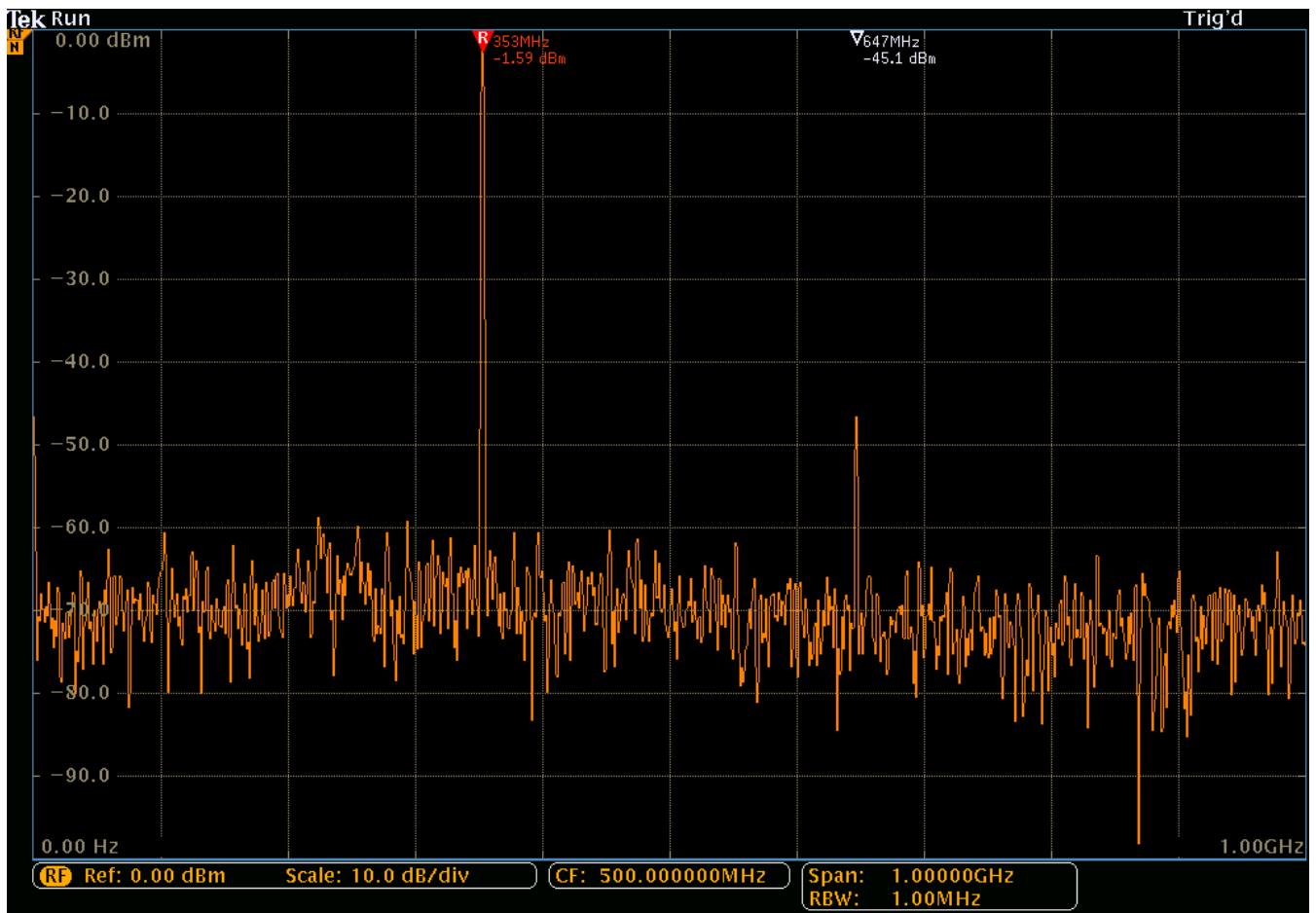


Figure 40 : spectre du signal de sortie du DDS à l'analyseur de spectre pour une amplitude maximale

Nous pouvons remarquer, grâce au marqueur, que le niveau de puissance du pic à la fréquence de 353 MHz est passé à $-1,6$ dBm ce qui correspond à une augmentation de 7 dBm.

Toutefois, que ce soit pour une amplitude maximale ou minimale, le niveau de puissance du pic est toujours nettement supérieur au niveau moyen de bruit.

Il est plus facile de se rendre compte de l'écart d'amplitude entre les amplitudes minimales et maximales lorsque l'on observe le signal de sortie sur une entrée standard de l'oscilloscope :



Figure 41 : acquisition de l'oscilloscope pour la comparaison des signaux de sortie du DDS à deux amplitudes différentes

La courbe du haut correspond au signal de sortie du DDS ayant une amplitude minimale et celle du dessous possède une amplitude maximale. Nous constatons que le signal du haut dispose d'une tension crête à crête d'environ 230 mV alors que celle du bas possède une amplitude d'environ 900 mV crête à crête. Le passage de l'amplitude minimale à maximale a donc pour effet de quasiment tripler la tension CC du signal de sortie.

Ce petit travail sur le DDS m'a permis d'acquérir les bases de l'interfaçage graphique avec Python et m'a également permis de travailler sur la programmation du module USB avec un microcontrôleur PIC18. Ces tâches préparatoires seront grandement utiles pour le travail réalisé sur la PLL.

b. Travail effectué sur la PLL

➤ Généralités sur les PLL

Une boucle à verrouillage de phase, ou boucle à phase asservie (*Phase Locked Loop*, PLL), est un circuit électronique qui, à l'instar du DDS, permet de générer un signal ayant une

fréquence multiple de la fréquence du signal d'entrée.

Le terme « asservissement » est utilisé car un tel système comporte une boucle de contre réaction permettant à la sortie du circuit d'être réinjectée à l'entrée. Cela permet de constamment comparer les phases des signaux de sortie et d'entrée entre eux à l'aide d'un comparateur de phase. Ce comparateur génère ensuite une tension, directement proportionnelle à l'écart de phase entre les deux signaux, qui sera présentée à l'entrée d'un filtre de boucle (généralement un filtre passe-bas). Puis, le signal arrive jusqu'à un oscillateur contrôlé en tension (*Voltage Controlled Oscillator*, VCO). La tension générée par le comparateur de phase permettra ainsi de régler la fréquence du signal sortant du VCO.

La figure suivante présente un schéma blocs de l'architecture interne de base d'une PLL :

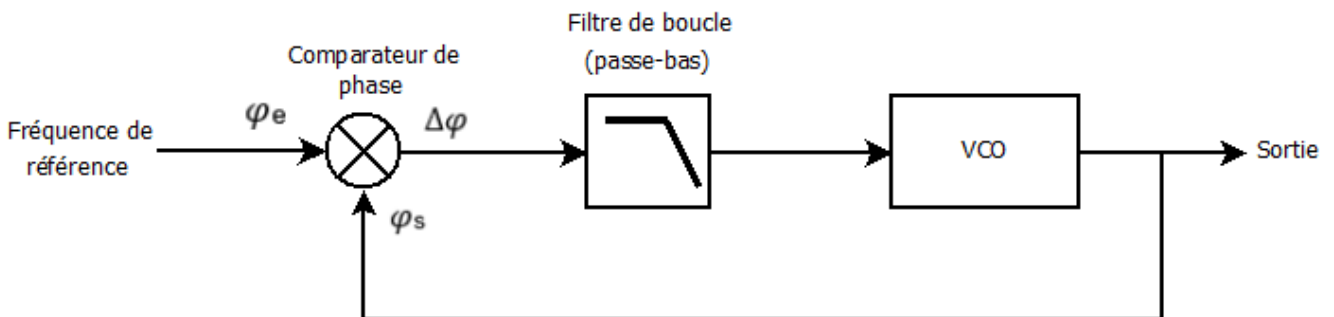


Figure 42 : schéma blocs de l'architecture interne de base d'une PLL

Dans la pratique, plusieurs variantes reposant sur cette base ont été établies. L'une de ces variantes correspond au circuit intégré qui sera utilisé pour le développement du générateur RF. La caractéristique majeure de ce circuit est qu'il comporte un diviseur de fréquence par un entier N dans la boucle de contre-réaction. Cela permet de générer, en sortie du VCO, une fréquence environ égale à la fréquence de référence multipliée par N :

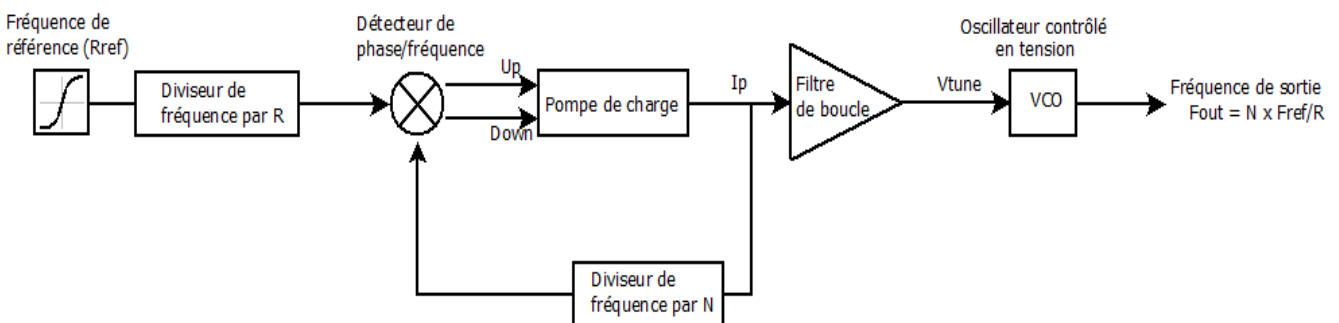


Figure 43 : schéma blocs d'une PLL avec un diviseur de fréquence par N

Nous allons expliciter plus en détail les éléments constitutifs de la PLL présentée dans la figure précédente.

- **Fréquence de référence** : fréquence à partir de laquelle sera générée la fréquence de sortie d'après l'équation $F_{out} = \frac{F_{ref}}{R} * N$. Cette référence permet également de fournir la fréquence de référence du détecteur de phase. Pour des soucis de résolution en fréquence et de stabilité, la référence est généralement un signal à haute fréquence provenant d'un bon cristal qui sera ensuite abaissé pour satisfaire aux conditions du détecteur de phase.

- **Diviseur de fréquence par R** : abaisse la fréquence de référence du système d'un facteur R afin de générer l'horloge de référence du détecteur de phase. D'un point de vue électronique, ce composant est généralement un compteur.
- **Détecteur de phase/fréquence** : ce composant permet de comparer les signaux en provenance des compteurs R et N et génère un signal proportionnel à l'écart de phase entre les deux signaux d'entrée. En réalité, le détecteur génère deux signaux distincts : « *up* » et « *down* ». En effet, lorsque le détecteur est polarisé positivement, le signal « *up* » est produit lorsqu'une différence de phase positive est détectée (lorsque le signal de référence est en avance par rapport au signal de sortie du VCO). Inversement, le signal « *down* » est généré lorsqu'un écart de phase négatif est détecté (lorsque le signal de sortie du VCO est en avance par rapport au signal de référence).

Remarque : Il est à noter qu'il existe une « zone morte » dans la fonction de transfert du détecteur. C'est une zone représentée par un plancher horizontal dans la caractéristique de transfert du composant et correspond à une région pour laquelle la sortie demeure constante, peu importe l'écart de phase. Généralement, une telle zone est problématique dans de nombreuses applications ce qui impose la constitution de circuits « anti zone morte ».

- **Pompe de charge** : ce composant est constitué de deux sources de courant opposées (I_p et $-I_p$) connectées ou déconnectées du reste du circuit grâce à deux interrupteurs. Ces interrupteurs sont reliés aux signaux « *up* » et « *down* » du détecteur de phase. Pour simplifier, il est possible de dire que le signal « *up* » force la pompe de charge à produire un courant positif en direction du filtre de boucle. En d'autres termes, l'interrupteur au niveau de la source de courant positif est abaissé dans ce cas-là. En revanche, le signal « *down* » contrôle la source de courant produisant un courant négatif ce qui a pour effet de pomper du courant hors du filtre. Dans le cas d'une pompe de charge réalisée à partir de la technologie CMOS, les interrupteurs sont des transistors dont les bases sont connectées aux signaux « *up* » et « *down* » du détecteur.

Un exemple de schéma d'une pompe de charge usuelle est proposé dans la figure suivante :

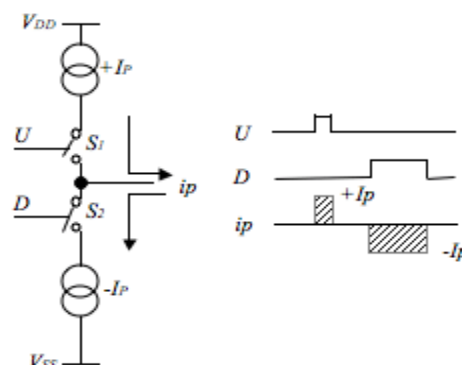


Figure 44 : schéma d'une pompe de charge usuelle

Comme le montre le chronogramme de la figure 44, la pompe de charge injecte un courant positif ou négatif durant toute la durée du front montant du signal « *up* » ou « *down* ».

Le rôle principal de la pompe de charge consiste à convertir les signaux logiques issus du détecteur de phase en signaux analogiques (courant) exploitables par le filtre de boucle et permettant, à terme, de commander le VCO.

- **Filtre de boucle** : étant donné que la pompe de charge produit un courant et que le VCO est commandé en tension, il est nécessaire d'effectuer une conversion courant – tension, ce qui est réalisé, entre autres choses, par le filtre. La tension résultant de la conversion est appelée « V_{tune} » pour « tension de réglage ».

La bande passante du filtre est choisie de façon à ce que le bruit et les harmoniques produits par le détecteur de phase ou la pompe de charge tombent en dehors de la bande.

Le filtre a également pour fonction d'améliorer le bruit de phase en assurant un bruit de faible niveau dans toute la bande passante du filtre.

D'autre part, les filtres de boucles peuvent être passifs ou actifs, tout en gardant à l'esprit que le choix d'un filtre actif implique une dégradation du bruit de phase total de la boucle. Le design rigoureux du filtre de boucle permet d'influer sur la marge de phase du système et donc sur la stabilité.

Le filtre de boucle est le composant essentiel d'une boucle à verrouillage et de lui dépendent plusieurs paramètres cruciaux comme la vitesse de verrouillage de la boucle ou encore la vitesse de changement de fréquence. Effectivement, plus la fréquence de coupure du filtre est faible, plus les changements de fréquence se feront lentement, et inversement lorsque l'on augmente la valeur de la fréquence de coupure.

Ce composant est généralement le plus difficile à concevoir dans la construction d'une PLL.

- **Oscillateur contrôlé en tension (VCO)** : la tension, proportionnelle à l'écart de phase entre les signaux d'entrée de la boucle, permet d'appliquer une tension de réglage au VCO. Cette tension de réglage commandera directement la fréquence d'oscillation.

La gamme de fréquences couverte par le VCO doit correspondre à la bande passante du filtre de boucle pour permettre un fonctionnement optimal.

De plus, un VCO doit absolument avoir une caractéristique de transfert aussi monotone que possible afin d'assurer qu'une augmentation de la tension de réglage implique une augmentation de la fréquence du signal de sortie. Un exemple de caractéristique de transfert d'un VCO est proposé en figure 45 :

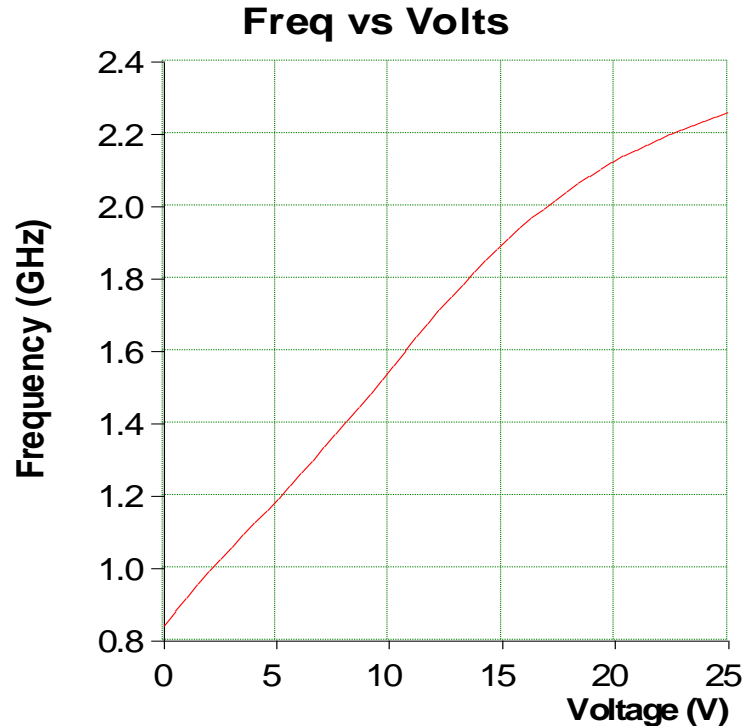


Figure 45 : exemple de caractéristique de transfert d'un VCO

Par conséquent, le principal paramètre d'un VCO est sa sensibilité de modulation, autrement dit la pente de la droite de la caractéristique de transfert. A titre d'exemple, la pente de la droite présentée dans la figure ci-dessus est de 71 MHz/V. Donc pour une variation de l'ordre d'un volt de la tension de réglage, la fréquence de sortie de l'oscillateur sera augmentée de 71 MHz.

- **Diviseur de fréquence par N** : ce compteur permet de faire le lien entre l'entrée et la sortie de la boucle à verrouillage de phase car il est placé dans la boucle de contre-réaction.

Ce circuit électronique programmable permet d'obtenir une fréquence en sortie du VCO supérieure à la fréquence de référence du système. Cela est réalisé en multipliant la fréquence du signal sortant du compteur R par la valeur programmée dans le compteur N, d'après l'équation :

$$F_{out} = \frac{F_{ref}}{R} * N$$

La technologie de ce compteur a beaucoup évoluée au cours du temps mais s'est finalement arrêtée sur une technique qui a fait ses preuves. L'évolution du bloc « diviseur de fréquence par N » a permis d'améliorer le fonctionnement des PLL en ajoutant un pré-diviseur entre le VCO et le compteur N.

Le recourt à un pré-diviseur est nécessaire lorsque l'on souhaite générer de hautes fréquences en sortie du VCO. Prenons un exemple : on souhaite générer une fréquence de 1 GHz à l'aide d'une fréquence de référence de 10 MHz et avec une horloge de référence de 1 kHz pour le détecteur de phase. Dans ce cas, la valeur du compteur R est fixée à 10 000 ($R = \frac{F_{ref}}{F_{detecteur}}$) et donc la valeur de N est 1 000 000. Ce compteur devra alors être construit de façon à pouvoir y programmer une valeur sur au moins 20

bits afin de pouvoir espérer atteindre une fréquence de sortie de 1 GHz, ce qui assez conséquent.

Cette difficulté technique peut être surmontée en introduisant le pré-diviseur qui permettra d'abaisser la fréquence à un niveau acceptable par la technologie du compteur.

Mais l'utilisation d'un pré-diviseur classique implique une perte de résolution en fréquence dans la boucle car, au lieu d'avoir $F_{out} = \frac{F_{ref}}{R} * N$, nous aurons $F_{out} = \frac{F_{ref}}{R} * N * P$, avec P la valeur programmée dans le pré-diviseur.

Cette problématique peut être résolue en utilisant un pré-diviseur double-modulo associé à deux compteurs A et B (en fixant la valeur de B plus grande que celle de A), comme montré sur la figure 46 :

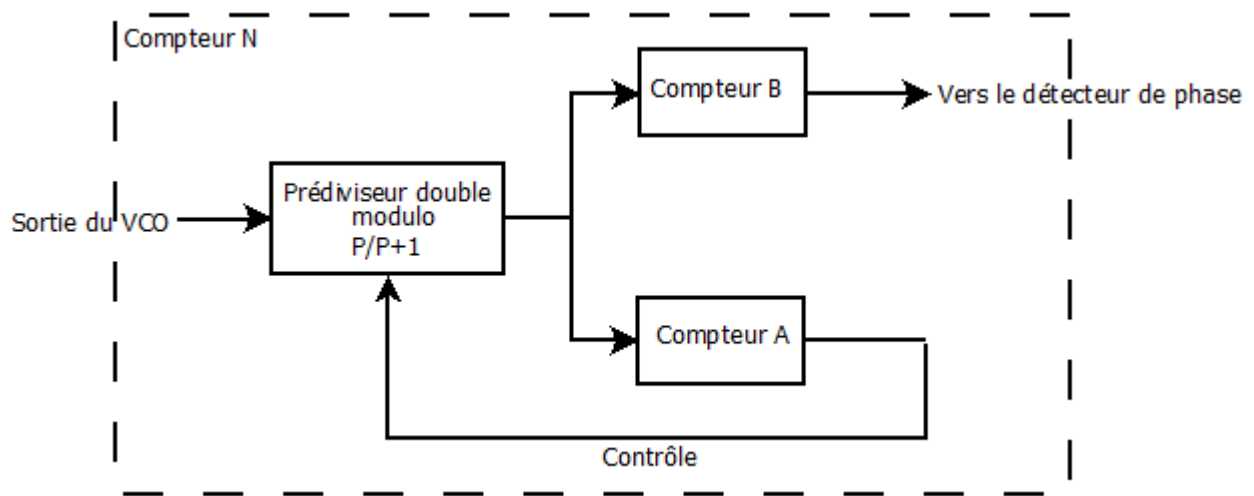


Figure 46 : schéma de l'architecture du diviseur de fréquence par N en utilisant un pré-diviseur double-modulo

Un pré-diviseur double-modulo est un compteur qui possède deux ratios de division : P et P+1, le passage de l'un à l'autre peut s'effectuer à l'aide d'un signal de contrôle extérieur. Dans notre cas, c'est le compteur A qui, lorsqu'il finit de compter, déclenche le basculement du ratio.

Une opération typique du bloc présenté dans la figure précédente se déroule de la façon suivante :

- Tant que le compteur A n'a pas fini de compter, le pré-diviseur conserve un ratio égal à P+1 (nombre impair).
Donc les deux compteurs A et B se décrémente de 1 à chaque fois que le pré-diviseur compte P+1 cycles de VCO. Ainsi, le compteur A aura fini de compter au bout de $(P + 1) * A$ cycles de VCO.
- Une fois que le compteur A atteint 0, sa sortie passe à l'état bas, forçant le pré-diviseur à passer à un ratio de division égal à P (nombre pair).
- Il restera alors $(B - A) * P$ cycles de VCO avant que le compteur B atteigne 0.
- Dès que le compteur B finit de compter, les deux compteurs sont automatiquement rechargés avec leurs valeurs initialement programmées.

Si l'on fait la somme des cycles de VCO nécessaires avant que les deux compteurs ne soient réinitialisés, on obtient :

$$N = (P + 1) * A + (B - A) * P \text{ soit } N = BP + A.$$

La fréquence en sortie du VCO devient alors :

$$F_{out} = \frac{F_{ref}}{R} * (BP + A).$$

A la différence du système utilisant un compteur N allié à un pré-diviseur P classique, le montage avec le pré-diviseur double-modulo et les deux compteurs A et B permet de d'intégrer le facteur P dans l'équation du compteur N pour obtenir $N = BP + A$. Cela est réalisé en combinant l'ensemble au sein d'un même bloc formant un compteur N à part entière.

➤ Tâches réalisées

Le contenu initial de mon stage prévoit de travailler sur un montage à base de PLL. Plus précisément, je dois programmer une interface permettant de faire le lien entre un ordinateur et la PLL. De cette façon, il sera possible de traduire des instructions provenant d'un ordinateur en des commandes compréhensibles par la PLL.

A mon arrivée à l'institut, un schéma électronique du circuit formant une PLL complète avait déjà été réalisé par les ingénieurs de l'IASB. De plus, le PCB correspondant était en cours de production par une entreprise spécialisée.

Comme évoqué en partie II.1.b, le chaîne RF consistera d'abord à générer de hautes fréquences puis à les abaisser à l'aide de diviseurs de fréquences afin d'obtenir les gammes de fréquences correspondant aux bandes UV, VIS et NIR. Ainsi, trois PCB devront être réalisés pour les trois bandes de fréquences. Ces PCB seront identiques mis à part les VCO et diviseurs de fréquence finaux. Le tableau suivant permettra d'y voir plus clair :

	Bande VIS	Bande UV	Bande NIR
Intervalle de fréquences disponible en sortie du VCO	1152 - 2496 MHz	1040 - 2080 MHz	1472 - 3072 MHz
Valeurs des diviseurs de fréquence	8 et 2	8	8 et 4
Intervalle de fréquences à obtenir en sortie de la chaîne RF	72 - 156 MHz	130 - 260 MHz	46 - 96 MHz

Figure 47 : tableau présentant les relations de fréquence dans le système utilisant une PLL

Le composant choisi pour la PLL est l'ADF4108 de la société *Analog Devices* pouvant être utilisé pour générer des fréquences allant jusqu'à 8 GHz. Ce composant comporte un détecteur de phase ainsi qu'une pompe de charge. Le filtre de boucle sélectionné est l'AD820 (filtre actif), conçu par la même entreprise, et le VCO est un ROS-2500W-319+ de *Mini-Circuits*, générant des fréquences comprises entre 1000 et 2400 MHz.

Le PCB en cours de production sera destiné à fonctionner dans la bande VIS et un schéma fonctionnel du montage est proposé ci-après :

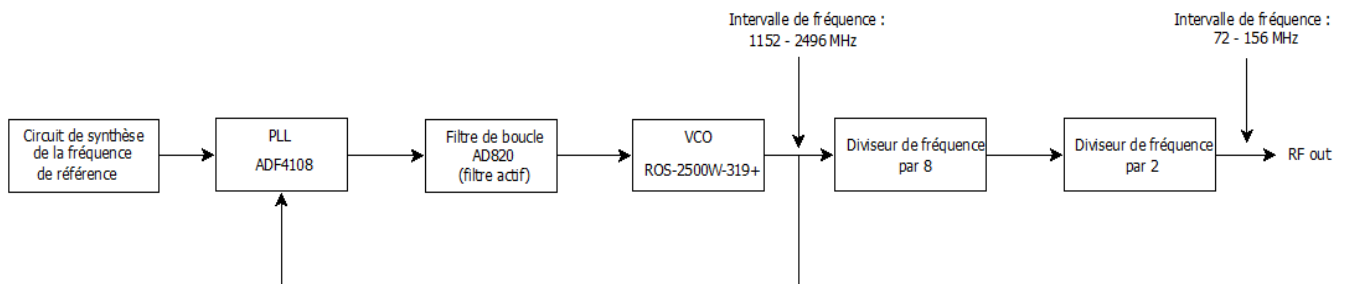


Figure 48 : schéma fonctionnel du montage à base de PLL réalisé par l'IASB

Mon travail consiste à commander la PLL dont le schéma blocs est le suivant :

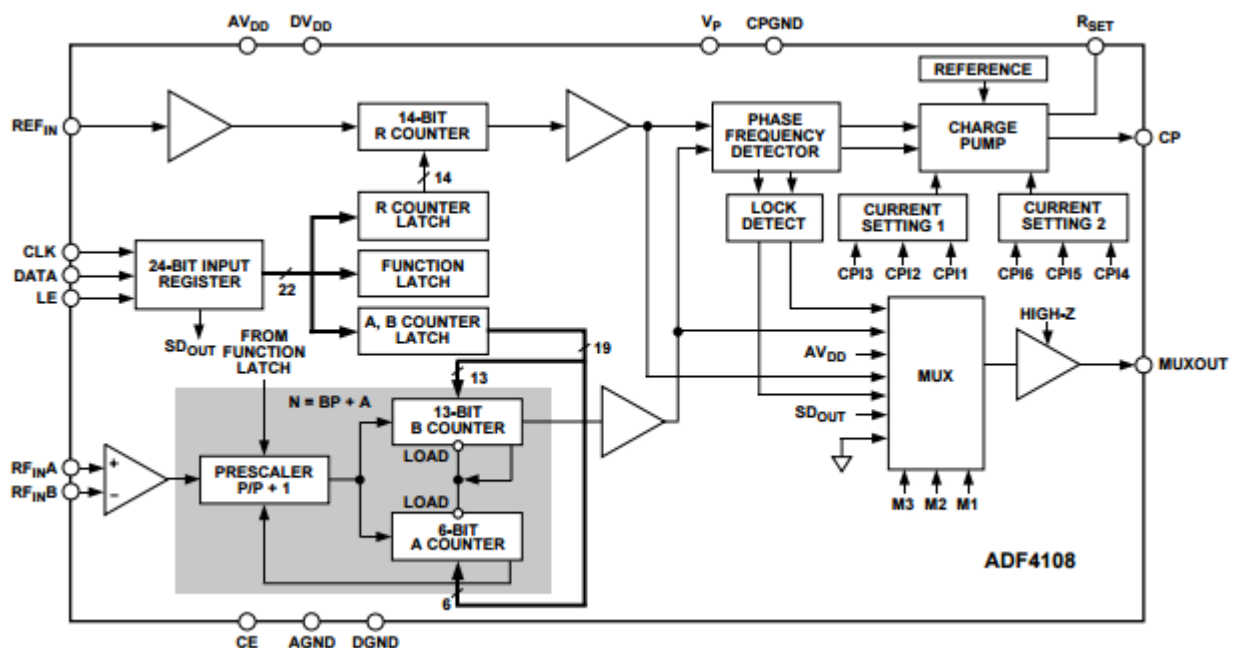


Figure 49 : schéma blocs de la PLL ADF4108 de Analog Devices

Comme pour le DDS, mon travail est de programmer une interface graphique permettant à un utilisateur de régler la PLL en ayant accès à différents paramètres programmables. Il faudra ensuite transmettre ces réglages à la PLL afin de la configurer convenablement.

Etant donné que l'ADF4108 ne possède pas d'interface USB lui permettant de se connecter directement à un ordinateur, il est nécessaire de trouver un moyen pour faire parvenir les réglages entrés sur l'interface graphique jusqu'au composant. Pour ce faire, le module USB utilisé pour le DDS sera de nouveau employé car la PLL peut être commandée par liaison SPI.

Ainsi, l'*USB PIM DEMO BOARD* jouera le rôle de « convertisseur » USB-SPI.

Deux programmes doivent donc être écrits : l'un en langage C embarqué permettant de programmer le microcontrôleur présent sur le module USB, et l'autre en Python permettant de concevoir l'interface graphique. Ces deux étapes ayant déjà été traitées dans les grandes lignes lors du travail réalisé sur le DDS, il m'a été plus facile de rentrer dans le vif du sujet.

Mais avant de saisir la première ligne de code, j'ai mené plusieurs recherches documentaires, accompagnées de prise de notes, sur le fonctionnement d'une PLL classique. Cela m'a amené à étudier toutes les différentes parties d'une boucle à verrouillage de phase, allant du détecteur de phase au VCO. Ces recherches documentaires ont été cruciales pour la

compréhension du schéma électronique élaboré par l'institut. De plus, il aurait été très difficile pour moi de parvenir à programmer la PLL sans en comprendre le fonctionnement.

Ensuite, j'ai entrepris de parcourir toute la *datasheet* de l'ADF4108 afin de déterminer de quelle façon il me sera possible de la programmer.

Une fois ce travail de recherche documentaire entrepris, j'ai pu commencer à programmer le microcontrôleur du module USB sur MPLABX IDE. Il est possible de constater sur le schéma de la figure 49 p.45 que la PLL possède trois signaux impliqués dans une liaison SPI : CLK, DATA et LE. CLK est l'horloge de référence fournie par le microcontrôleur de l'*USB PIM DEMO BOARD*, DATA correspond au flux de données à transmettre au composant et enfin LE est un signal indiquant à la PLL de transférer le contenu du registre d'entrée vers les registres actifs. J'ai donc repris le même code source que celui utilisé pour le DDS dans lequel j'ai modifié quelques paramètres liés à la communication SPI. L'initialisation de cette communication ressemble beaucoup à celle présentée dans la partie II.7.a page 29 de ce document, mis à part quelques modifications. Les différentes étapes seront présentées ci-après :

- Initialiser la direction des ports en entrée ou en sortie pour chaque signal impliqué dans la communication SPI.
- Déterminer quel mode SPI sera à utiliser en se référant aux chronogrammes présentés dans la *datasheet* de l'ADF4108 :

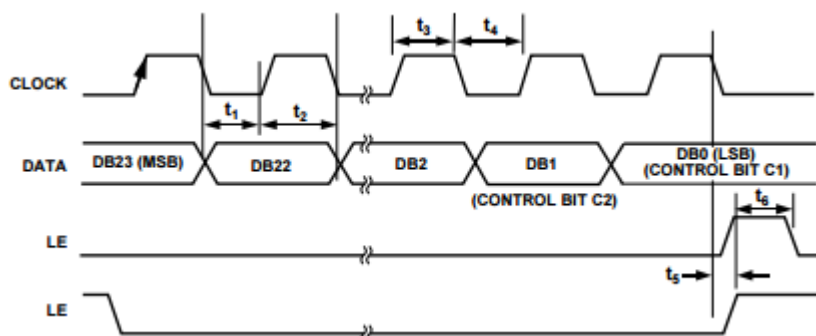


Figure 50 : chronogrammes des signaux impliqués dans la communication SPI de l'ADF4108

L'observation de ces chronogrammes nous indique que la transmission de données a lieu sur front descendant de l'horloge (comme le montre la première barre verticale en partant de la gauche). D'autre part, on constate que l'état logique initial de la « clock » est à l'état bas. Ces observations nous amènent à choisir le mode SPI à utiliser qui sera le même que celui choisi pour le DDS : le mode 0,0 avec CKP = 0 et CKE = 1.

- S'assurer que le signal LE soit à l'état bas pendant toute la durée du transfert de données. Il faudra ensuite générer un front montant une fois le bit de poids faible transmis pour indiquer à la PLL de charger le contenu du registre d'entrée vers les registres actifs.

L'initialisation de la communication SPI est réalisée au sein d'une unique fonction du programme :

```

/*****
* Function: void initialize_SPI();
*
* Overview: Initializes the SPI communication to communicate with the
* PLL (signal LE, DATA and CLK).
*
* PreCondition: None
*
* Input: None
*
* Output: None
*
*****/
void initialize_SPI(){
    TRISBbits.TRISB4=0; //Output. CLK
    TRISCbits.TRISC7=0; //Output. DATA
    TRISDbits.TRISD7=0; //Output. LE (Latch Enable)

    disable_latch(); //Default Low state for the LE signal

    DRV_SPI_INIT_DATA spiInitData = {1, 0, 1, SPI_BUS_MODE_0, 0}; //The structure that defines
                                                                    //the SPI channel's operation

    DRV_SPI_Initialize(&spiInitData); //Initializes the SPI instance specified
                                      //by the channel of the initialization structure.
}

```

Figure 51 : fonction d'initialisation de la communication SPI pour l'ADF4108

Remarque : la fonction *disable_latch()* sert uniquement à imposer un niveau logique haut pour le signal LE.

❖ Code de test sur MPLAB uniquement :

A mon arrivée à l'institut, on m'avait prévenu que la carte électronique allait sûrement arriver à la mi-juillet. Je savais alors que je n'aurai pas le temps de terminer l'interface graphique dans un délai aussi court afin de pouvoir la tester sur le PCB. J'ai donc écrit un programme entièrement sur MPLAB qui s'occupe d'initialiser la communication SPI, de construire les quatre registres à envoyer à la PLL puis de les envoyer. Ce programme aura l'avantage d'être plus rapide à écrire afin de pouvoir être testé directement sur la carte dès sa réception.

Mais pour être en mesure de modifier facilement un ou plusieurs paramètres de l'ADF4108 en vue de tester plusieurs configurations possibles, il m'a fallu trouver une idée permettant d'accéder individuellement à chaque bit de chaque registre de la PLL.

Pour mieux comprendre cette approche, la figure suivante montre comment est organisé l'un des quatre registres programmables :

INITIALIZATION LATCH

PRESCALER VALUE		POWER- DOWN 2	CURRENT SETTING 2			CURRENT SETTING 1			TIMER COUNTER CONTROL				FASTLOCK MODE	FASTLOCK ENABLE	CP THREE- STATE	PD POLARITY	MUXOUT CONTROL			POWER- DOWN 1	COUNTER RESET	CONTROL BITS	
DB23	DB22		DB20	DB19	DB18	DB17	DB16	DB15	DB14	DB13	DB12	DB11	DB10	DB9	DB8	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0
P2	P1	PD2	CPI6	CPI5	CPI4	CPI3	CPI2	CPI1	TC4	TC3	TC2	TC1	F5	F4	F3	F2	M3	M2	M1	PD1	F1	C2 (1)	C1 (1)

Figure 52 : construction interne du registre d'initialisation de l'ADF4108

On constate qu'à l'intérieur de ce registre se trouvent plusieurs paramètres qu'il faudrait pouvoir modifier individuellement, sans avoir à recalculer la valeur binaire ou hexadécimale du registre en entier à la main à chaque fois.

La solution consiste à déclarer des champs de bits (*bitfields*) car un registre peut être vu comme une structure de *bitfields*. Cette déclaration a été faite de la façon suivante dans le programme :

```
//Initialization Latch type
typedef volatile union
{
    uint32_t fullLatch; //Indicates the total value of the register
    struct{ //Bitfields of the initialization Latch
        uint8_t INITIALIZATION_LATCH_CTRL_BITS:2; //This value is 3 (0b11) for the initialization Latch
        uint8_t COUNTER_RESET:1; //Counter Reset=0 => normal operation, 1 => R, A, B counters held in reset
        uint8_t POWER_DOWN_1:1; /* CE PD2 PD1
        * 0 X X Asynchronous Power Down
        * 1 X 0 Normal Operation
        * 1 0 1 Asynchronous Power Down
        * 1 1 1 Synchronous Power Down
        */

        uint8_t MUXOUT_CONTROL:3; /* M3 M2 M1
        * 0 0 0 Three-state output
        * 0 0 1 Digital Lock detect
        * 0 1 0 N Divider output
        * 0 1 1 DVDD
        * 1 0 0 R Divider output
        * 1 0 1 N-Channel open-drain Lock detect
        * 1 1 0 Serial Data output
        * 1 1 1 DGND
        */

        uint8_t PD_POLARITY:1; //Phase Detector Polarity (0->negative, 1->positive)
        uint8_t CP_THREE_STATE:1; //Charge Pump output (0->normal, 1->three-state)
        uint8_t FAST_LOCK_ENABLE:1; /* FastLock enable FastLock mode
        * 0 X Fastlock disabled
        * 1 0 Fastlock mode 1
        * 1 1 Fastlock mode 2
        */

        uint8_t FAST_LOCK_MODE:1; //See FAST_LOCK_ENABLE for indications
        uint8_t TIMER_COUNTER_CTRL:4; //Timeout range 3 to 63 PFD cycles
        uint8_t CURRENT_SETTING_1:3; //Sets the maximum value of the charge pump output current depending on the Rset value
        uint8_t CURRENT_SETTING_2:3; //See CURRENT_SETTING_1 for indications
        uint8_t POWER_DOWN_2:1; //See POWER_DOWN_1 for indications
        uint8_t PRESCALER_VALUE: 2; /* P2 P1
        * 0 0 8/9
        * 0 1 16/17
        * 1 0 32/33
        * 1 1 64/65
        */
    }ilatchBitfields;
}I_LATCH;
```

Figure 53 : utilisation de *bitfields* pour la déclaration d'un registre

Cette solution permet de définir le type « I_LATCH » contenant une structure de *bitfields* pour lesquels on indique combien de bits ils occupent au sein du registre (en se référant à la structure présentée en figure 52 p.47). Ce type est une « union » qui permet d'associer une structure avec une variable (*fullLatch* dans le programme) correspondant à la valeur du registre total. Cette variable sera mise à jour automatiquement au fur et à mesure que les *bitfields* seront remplis.

Ensuite, il est possible d'attribuer une valeur à chaque champ de bits individuellement comme nous pouvons le voir dans sur la figure suivante :

```

/*****
* Function: void I_Register();
*
* Overview: This function configures the bitfields of the
* initialization latch of the PLL and then loads it into the PLL.
*
* PreCondition: None
*
* Input: None
*
* Output: None
*
*****/
void I_Register(){
    I_LATCH iLatch;
    uint32_t initializationLatch=0;
    iLatch.fullLatch=0;
    iLatch.iLatchBitfields.PRESCALER_VALUE=2;
    iLatch.iLatchBitfields.POWER_DOWN_2=0;
    iLatch.iLatchBitfields.CURRENT_SETTING_2=7;
    iLatch.iLatchBitfields.CURRENT_SETTING_1=7;
    iLatch.iLatchBitfields.TIMER_COUNTER_CTRL=7;
    iLatch.iLatchBitfields.FAST_LOCK_MODE=0;
    iLatch.iLatchBitfields.FAST_LOCK_ENABLE=0;
    iLatch.iLatchBitfields.CP_THREE_STATE=0;
    iLatch.iLatchBitfields.PD_POLARITY=0;
    iLatch.iLatchBitfields.MUXOUT_CONTROL=1;
    iLatch.iLatchBitfields.POWER_DOWN_1=0;
    iLatch.iLatchBitfields.COUNTER_RESET=0;
    iLatch.iLatchBitfields.INITIALIZATION_LATCH_CTRL_BITS=3;

    /* Binary operations to ensure that the concatenation of the bits is correct.
    * Selection of the 24 LSBs as the PLL expects 3 bytes.
    */
    initializationLatch = ((iLatch.fullLatch & 0x00007FFF) | ((iLatch.fullLatch & 0x00FF0000) >> 1));
    initializationLatch = initializationLatch | ((iLatch.fullLatch & 0x02000000) >> 2);

    write_PLL(initializationLatch); //Load Latch into the PLL
}

```

Figure 54 : attribution d'une valeur pour chaque *bitfields* du registre

Remarque : les deux lignes situées juste avant l'instruction `write_PLL(initialisationLatch)` sont des opérations binaires nécessaires afin d'avoir une concaténation correcte des bits pour former le registre final. En effet, le compilateur de MPLABX IDE n'effectue pas une concaténation bit à bit mais octet par octet ce qui engendre des incertitudes qu'il faut corriger grâce à des opérations binaires.

Une fois les valeurs des *bitfields* attribuées et la valeur du registre entier calculée, nous pouvons l'envoyer à la PLL via la liaison SPI. C'est le rôle de la fonction suivante :

```

/*****
* Function: void write_PLL(uint32_t Latch);
*
* Overview: This function sends a Latch through the SPI bus
* and loads it into the PLL.
*
* PreCondition: None
*
* Input: uint32_t Latch : the Latch that needs to be Loaded into the PLL.
*
* Output: None
*
*****/
void write_PLL(uint32_t latch){
    disable_latch();
    Latch[0]=((latch >> 16) & 0xFF); //Selection of the 8 MSBs of the Latch
    Latch[1]=((latch >> 8) & 0xFF); //Selection of the 8 ISBs (intermediate) of the Latch
    Latch[2]=(latch & 0xFF); //Selection of the 8 LSBs of the Latch
    disable_latch(); //LE must go down during a data transfer
    DRV_SPI_PutBuffer (1, Latch, 3); //This routine writes a buffered data to SPI bus.
    enable_latch(); //LE must go up at the end of the transfer to Load
    //data from the PLL's shift register to the correct PLL Latch
}

```

Figure 55 : fonction permettant d'adresser un des registres de la PLL via la liaison SPI

Remarque : il faut avoir en tête que chaque registre de la PLL possède 24 bits mais nous ne pouvons envoyer que 8 bits à la fois par le biais de la communication SPI. C'est pourquoi l'on décompose le registre en trois parts égales qui sont envoyées l'une après l'autre à l'ADF4108.

J'ai ensuite pu visualiser sur un oscilloscope les trames binaires envoyées à la PLL que j'ai comparé aux chronogrammes de la figure 50 p.46.

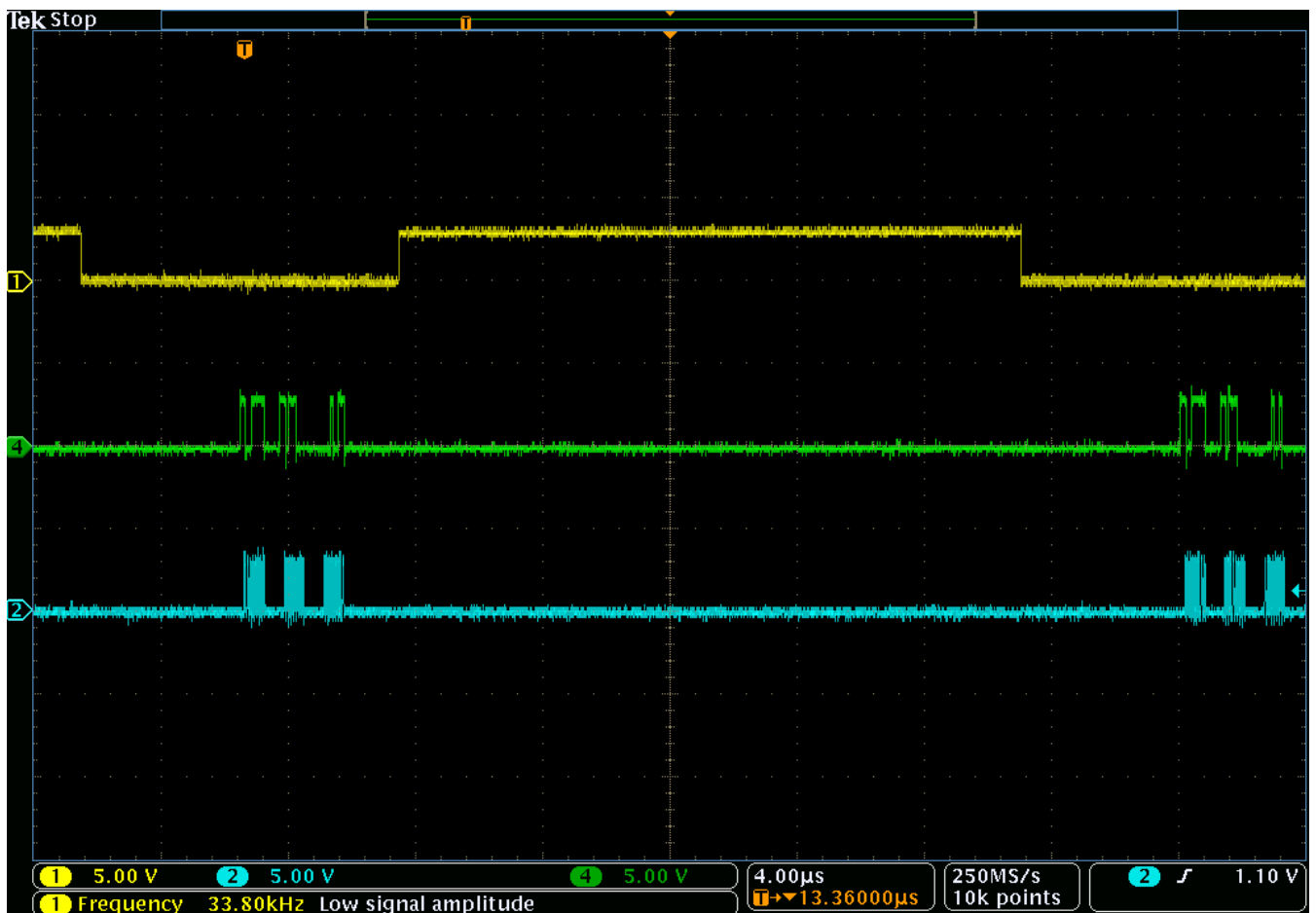


Figure 56 : visualisation des trames binaires de la liaison SPI à l'oscilloscope

Sur cette acquisition, nous observons deux trames binaires qui correspondent à l'envoi des deux premiers registres à la PLL. Cette acquisition m'a permis de constater que le programme envoyait bien les valeurs binaires à attribuer aux registres de la PLL en les découpant en trois octets au préalable (courbe en vert). L'envoi des données est cadencé par l'horloge (signal bleu) qui génère trois fois huit fronts montants pour chaque registre transmis. Nous constatons également que le signal LE (signal jaune) est bien à l'état bas pendant toute la durée d'un transfert et qu'il revient à l'état haut après l'envoi d'un registre.

Par ailleurs, en zoomant suffisamment sur chacune des quatre trames binaires (correspondant aux valeurs des quatre registres de la PLL) individuellement, j'ai pu me rendre compte que les valeurs attendues étaient bien présentes sur le bus SPI.

Après avoir présenté l'avancement du programme au responsable du projet, ce-dernier m'a précisé qu'il faudrait pouvoir parcourir une bande de fréquences précise (bande VIS, UV ou NIR) et générer une série de fréquences espacées d'un certain pas (dont la valeur sera modifiable). J'ai donc implémenté une boucle « *while* » parcourant la bande VIS et envoyant les registres à la PLL après chaque incrément de la fréquence de sortie. L'incrément a été fixé à 40 kHz et un *timer* comptant deux secondes a été utilisé afin de temporiser la génération de fréquence. En effet, après chaque envoi des trames par le bus SPI, le « *timer* » met le programme en pause pendant deux secondes avant de parcourir la boucle « *while* » de nouveau. Cela permettra d'avoir suffisamment de temps pour observer la génération de fréquence sur un analyseur de spectre lors de la phase de test.

❖ **Programmation du microcontrôleur du module USB pour le faire fonctionner avec une interface graphique :**

Finalement, la carte électronique est arrivée en retard et, lorsqu'elle est arrivée, plusieurs soucis ont été détectés ce qui rendait son utilisation limitée. J'ai donc eu le temps de programmer une interface graphique en utilisant le langage Python, à la manière de celle conçue pour le DDS.

Mais avant tout, de petites modifications sont à apporter au code permettant de programmer le microcontrôleur. En effet, il est nécessaire d'épurer le code source car toutes les définitions des registres avec les *bitfields* ne seront pas utilisées. Les fonctions relatives à l'initialisation de la liaison SPI seront conservées et nous ajouterons une fonction se chargeant de réceptionner les données reçues sur le port USB du module avant de les transmettre à la PLL via le bus SPI :

```

if( USBUSARTIsTxTrfReady() == true)
{
    uint8_t i;
    uint8_t numBytesRead;

    /* We retrieve the 3 bytes sent on the USB bus.
     * readBuffer : buffer containing the data read on the USBUSART bus.
     * numBytesRead : number of bytes read.
     */
    numBytesRead = getsUSBUSART(readBuffer, 3);

    /* PLL initialization : Initialization Latch Method
     * 1. Apply VDD.
     * 2. Program the Initialization Latch (with COUNTER_RESET=0).
     * 3. Do a Function Latch Load (with COUNTER_RESET=0).
     * 4. Do an R Latch Load.
     * 5. Do an AB Latch Load.
     */

    disable_latch();    //Make sure that the LE signal is Low before starting
                        //a transfer.

    /* As soon as we receive something ... */
    if(numBytesRead > 0){
        for(i=0; i<3; i++){
            Latch[i] = readBuffer[i];    //Store the 3 bytes read in an array
        }
        disable_latch();    //LE in Low state during data transfer
        DRV_SPI_PutBuffer (1, Latch, 3); //Send data buffer on SPI bus channel 1
        enable_latch();    //LE in high state to finalize transfer
        disable_latch();    //Make sure LE goes Low again
    }
}

```

Figure 57 : instruction "if" permettant de recevoir des données sur le port USB puis de les envoyer à la PLL

Le programme réceptionnera les données sur le port USB trois octets par trois octets (ce qui correspond à un registre entier) et stockera chaque octet dans un tableau. Ce tableau sera ensuite envoyé sur le bus SPI jusqu'à la PLL.

❖ Interface graphique :

Comme l'interface graphique écrite pour le DDS, celle permettant de contrôler la PLL a été écrite en langage Python qui offre un large panel de possibilités en ce qui concerne l'interfaçage graphique.

La problématique principale consiste à construire une application facile d'emploi et surtout facilement compréhensible. Pour construire mon interface, je me suis appuyé sur le logiciel « ADI PLL Int-N » que fournit la société *Analog Devices* avec toutes ses cartes d'évaluation à base de PLL avec diviseur de fréquence par N. Ce logiciel permet à l'utilisateur d'accéder et de modifier tous les paramètres programmables de l'ADF4108.

L'interface graphique finale, intitulée « PLL Controller », est présentée dans les deux figures suivantes :

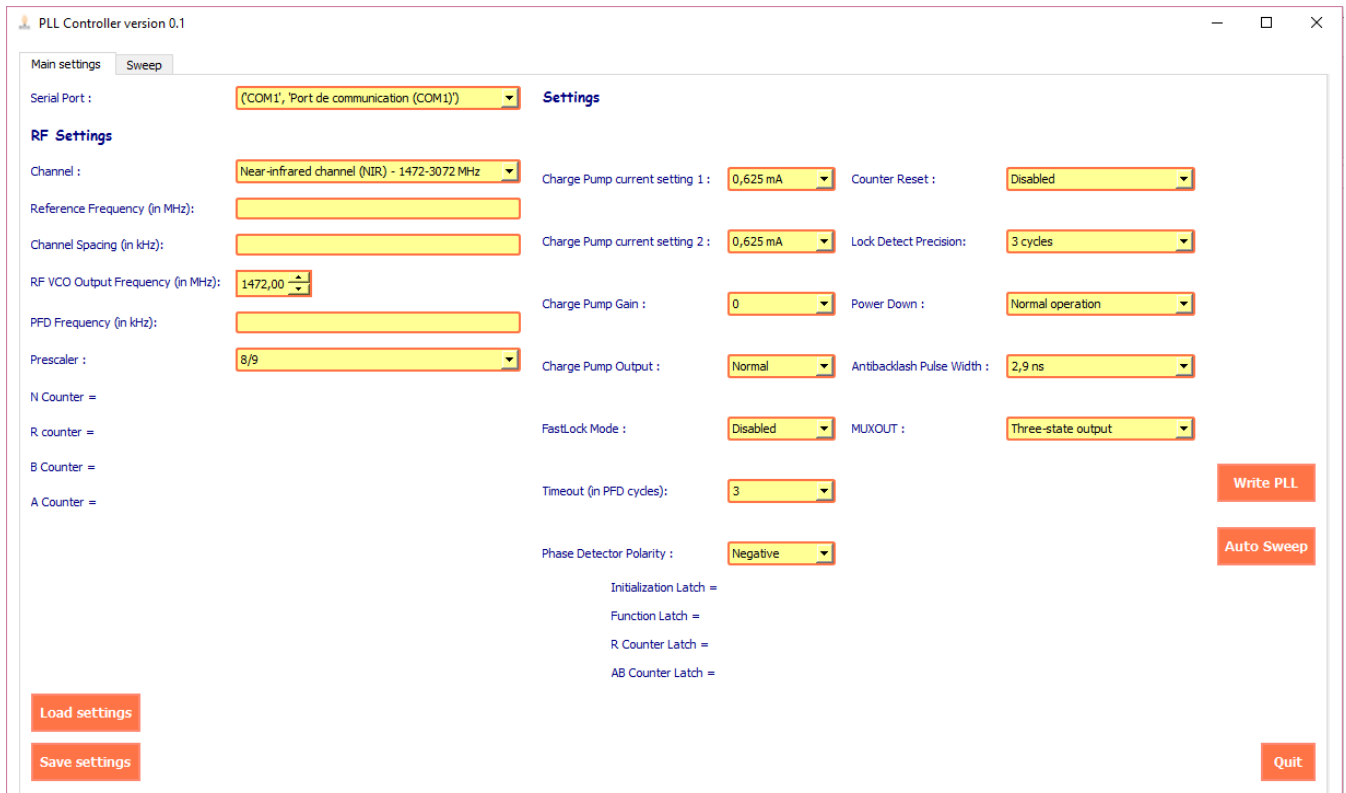


Figure 58 : interface graphique permettant de contrôler la PLL (premier onglet)

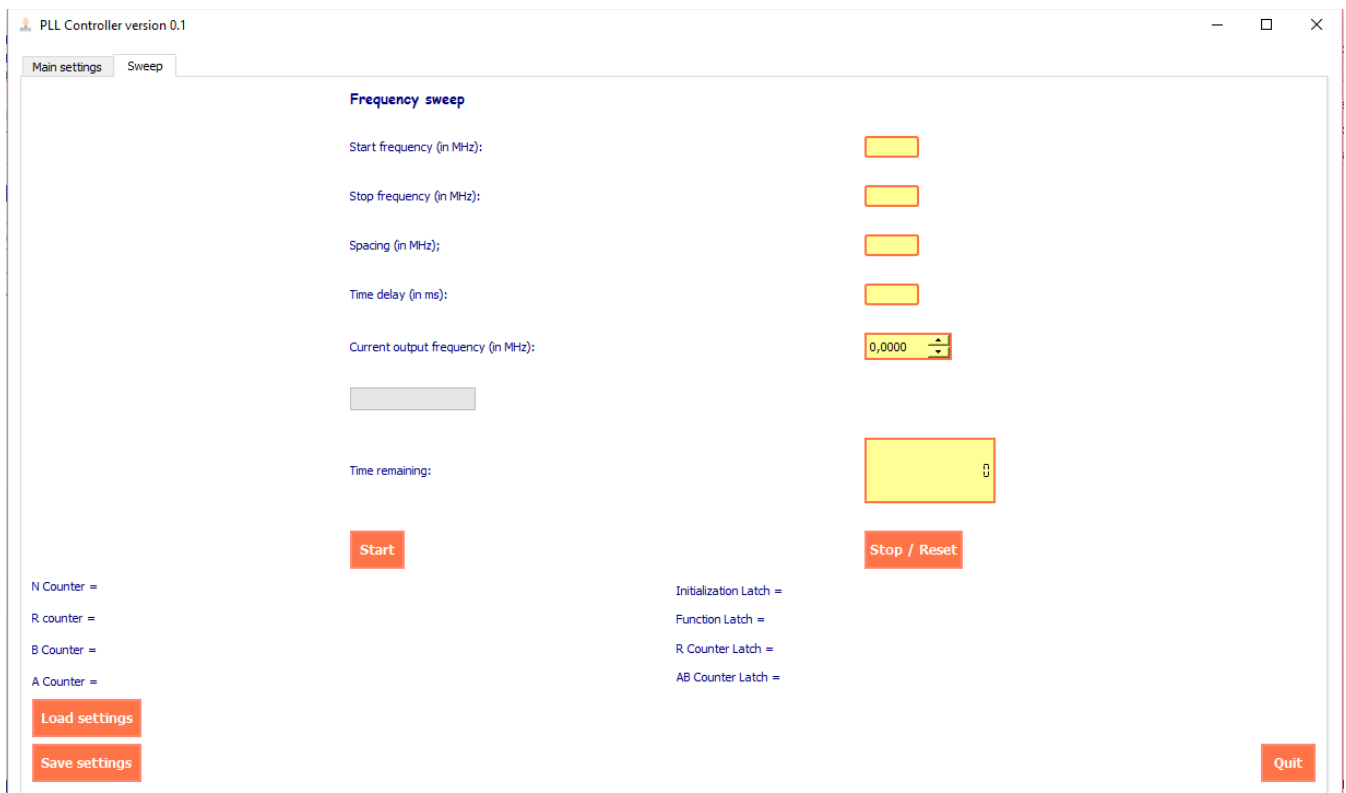


Figure 59 : interface graphique permettant de contrôler la PLL (deuxième onglet)

Le tableau suivant liste les paramètres présents dans l'onglet « *Main settings* » et présente une brève description pour chacun des éléments :

	Nom de l'élément	Type	Description	Remarque
	<i>Serial Port</i>	<i>Combo box</i>	Permet de sélectionner le port série de l'ordinateur auquel l'utilisateur a branché son module USB.	Au lancement de l'application, le programme détecte les ports séries de l'ordinateur qui sont utilisés et les affiche ici.
Section "RF Settings"	<i>Channel</i>	<i>Combo box</i>	Permet de sélectionner la bande de fréquence avec laquelle l'utilisateur souhaite travailler.	Choix parmi VIS, NIR ou UV.
	<i>Reference Frequency</i>	<i>Line edit</i>	L'utilisateur entre ici la valeur de la fréquence de référence qui sera utilisée sur le PCB de la PLL.	La valeur doit obligatoirement être un nombre entier et doit se trouver dans un intervalle précis.
	<i>Channel Spacing</i>	<i>Line edit</i>	Correspond au pas séparant deux fréquences consécutives générées en sortie du VCO.	La valeur doit obligatoirement être un nombre entier et doit se trouver dans un intervalle précis.
	<i>RF VCO Output Frequency</i>	<i>Spin box</i>	L'utilisateur entre ici la valeur de la fréquence de sortie qu'il souhaite obtenir en sortie du VCO.	Les limites max et min de l'élément sont fixées selon le choix sélectionné dans "Channel" et l'incrément/décrément correspond à la valeur entrée dans "Channel Spacing".
	<i>PFD Frequency</i>	<i>Line edit</i>	L'utilisateur doit entrer la valeur de la fréquence qui servira de fréquence de référence au détecteur de phase.	Cette valeur est la même que celle entrée dans "Channel Spacing". Ces deux éléments sont reliés dynamiquement : écrire dans l'un revient à écrire dans l'autre.
	<i>Prescaler</i>	<i>Combo box</i>	Permet de sélectionner les deux ratios de division (P et P+1) du pré-diviseur double-modulo.	Choix parmi 8/9, 16/17, 32/33 ou 64/65.
Section "Settings"	<i>Charge Pump current setting 1</i>	<i>Combo box</i>	Ces éléments correspondent à des paramètres programmables de la PLL permettant de mettre en œuvre des fonctionnalités spécifiques à l'ADF4108 qui sont détaillées dans la <i>datasheet</i> . Il serait hors sujet et fastidieux de les expliquer ici.	
	<i>Charge Pump current setting 2</i>	<i>Combo box</i>		
	<i>Charge Pump Gain</i>	<i>Combo box</i>		
	<i>Charge Pump Output</i>	<i>Combo box</i>		
	<i>Fastlock Mode</i>	<i>Combo box</i>		
	<i>Timeout</i>	<i>Combo box</i>		
	<i>Phase Detector Polarity</i>	<i>Combo box</i>		
	<i>Counter Reset</i>	<i>Combo box</i>		
	<i>Lock Detect Precision</i>	<i>Combo box</i>		
	<i>Power Down</i>	<i>Combo box</i>		
	<i>Antibacklash Pulse Width</i>	<i>Combo box</i>		
	<i>MUXOUT</i>	<i>Combo box</i>		

Figure 60 : description des paramètres présents dans l'onglet "Main settings" de l'application

Un certain nombre d'opérations peuvent être effectuées par un utilisateur à partir de cet onglet grâce aux boutons poussoirs disposés à des endroits précis de la fenêtre. Le tableau suivant résume et explique les opérations associées à chacun des boutons :

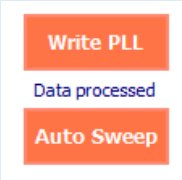
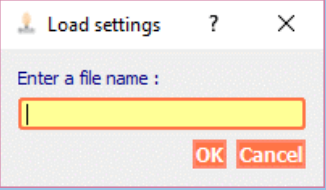
Opération	Description	Remarque	Nouveaux éléments affichés
<i>Write PLL</i>	Cette opération correspond à une opération "classique". Après appui sur le bouton éponyme, le programme récupère les valeurs de tous les éléments de l'onglet et s'en sert pour construire les registres à transférer à la PLL. La fonction permettant d'envoyer les données via le protocole USB est ensuite appelée automatiquement. La valeur de fréquence renseignée dans "RF VCO Output Frequency" pourra ensuite être retrouvée en sortie du VCO sur le PCB.	Si l'opération se déroule sans accrocs, un message s'affiche en-dessous du bouton "Write PLL" : "Data processed" (voir ci-contre).	
<i>Auto Sweep</i>	Le terme "sweep" signifie "balayer" donc cette opération permet de parcourir l'intervalle de fréquence renseigné dans "Channel" en générant une série de fréquences consécutives espacées de la valeur entrée dans "Channel Spacing". Cette opération peut être réalisée en appuyant sur le bouton "Auto Sweep" situé au-dessous du bouton "Write PLL". La valeur de la fréquence de sortie sera incrémentée dynamiquement dans le champ "RF VCO Output Frequency" toutes les deux secondes.	Lors de ce processus, deux nouveaux éléments sont affichés dans l'onglet. Le premier est un bouton "Stop / Reset" qui permet à l'utilisateur d'interrompre le processus et de réinitialiser le "RF VCO Output Frequency". Le deuxième est une barre de progression qui permet de se rendre compte de l'avancement de l'opération. Ces deux éléments seront affichés en-dessous du bouton "Auto Sweep" après un appui sur ce-dernier.	
<i>Save settings</i>	L'utilisateur, une fois avoir entré l'ensemble des paramètres, peut les sauvegarder dans un fichier texte. Une boîte de dialogue (voir dernière colonne) s'ouvre après appui sur le bouton "Save settings" situé en bas à gauche de l'onglet. Cette boîte de dialogue permet à la personne d'entrer le nom du fichier texte dans lequel seront stockés les paramètres. Il est à noter que les paramètres des deux onglets seront sauvegardés mais il n'est pas nécessaire d'avoir rempli le deuxième onglet pour sauvegarder.	Si le nom de fichier entré correspond à un fichier déjà existant dans le dossier où se trouve l'application, son contenu sera écrasé. Si l'utilisateur entre un nouveau nom, le fichier portant ce nom sera créé automatiquement. Une erreur est renvoyée si aucun nom de fichier n'a été entré par l'utilisateur.	
<i>Load settings</i>	Après avoir sauvegardé les paramètres, l'utilisateur pourra les charger ultérieurement en appuyant sur le bouton "Load settings" situé au-dessous du bouton "Save settings". Dans ce cas, une boîte de dialogue (voir dernière colonne) s'ouvre demandant à l'utilisateur d'entrer le nom du fichier texte qu'il souhaite charger.	Une erreur sera affichée si le nom entré ne correspond à aucun fichier présent dans le dossier.	
<i>Quit</i>	Un appui sur le bouton "Quit" entraîne l'arrêt de l'application et la fermeture de la fenêtre.		

Figure 61 : explication des opérations pouvant être effectuées dans l'onglet "Main settings"

Les tableaux ci-dessus n'ont pas fait mention des paramètres associés à des champs vides qui sont :

N Counter =	Initialization Latch =
R counter =	Function Latch =
B Counter =	R Counter Latch =
A Counter =	AB Counter Latch =

Figure 62 : paramètres de l'onglet "*Main settings*" associés à un champ vide

Effectivement, au lancement de l'application, ces éléments ont des valeurs nulles mais dès que l'utilisateur appui sur les boutons « *Write PLL* » ou « *Auto Sweep* », des valeurs sont affichées pour ces paramètres. Lorsque le programme récupère les valeurs des paramètres entrés par l'utilisateur, il effectue plusieurs calculs pouvant être trouvés dans la *datasheet* de l'ADF4108 qui permettent de déterminer les valeurs à programmer dans les compteurs de la PLL (compteurs présentés dans la partie II.7.b page 39). Les valeurs de chacun des compteurs, une fois calculées, sont placées à côté de leurs libellés respectifs (*N Counter*, *R Counter* ...).

De plus, le programme construit automatiquement les valeurs des registres à envoyer à la PLL à partir des paramètres entrés. Une fois construits, les valeurs hexadécimales des registres sont affichées à côté de leurs libellés (*Initialisation Latch*, *Function Latch* ...). Chacun des registres possèdera une valeur représentée sur 6 caractères alphanumériques.

Il est à noter que certaines de ces valeurs (*N Counter*, *B Counter*, *A Counter* et *AB Counter Latch*) évolueront dynamiquement dans le cas où une opération « *Auto Sweep* » a été déclenchée. En effet, ces valeurs sont directement liées à la fréquence à générer en sortie de du VCO qui est incrémentée toutes les deux secondes. Il est donc normal que ces valeurs changent également au cours du temps.

Une opération supplémentaire est proposée dans le deuxième onglet, intitulé « *Sweep* ». Ce processus ressemble beaucoup à l'opération « *Auto Sweep* » avec, toutefois, une différence majeure. Avec un « *Auto Sweep* », l'intervalle de fréquence balayé ne peut être que l'un des trois choix proposés dans « *Channel* » et le pas séparant deux fréquences de sortie consécutives est égal à la valeur entrée dans « *Channel Spacing* ». Or, avec une opération « *Sweep* », l'utilisateur peut entrer son propre intervalle de fréquence, renseigner le pas de fréquence qu'il désire et peut même définir le temps espaçant deux générations de fréquence.

Le tableau suivant liste et décrit les éléments présents dans l'onglet « *Sweep* » :

Nom de l'élément	Type	Description	Remarque
<i>Start frequency</i>	<i>Line edit</i>	L'utilisateur renseigne ici la valeur de fréquence qui sera le point de départ du balayage de fréquence.	La valeur doit obligatoirement être un nombre entier et doit se trouver dans un intervalle précis.
<i>Stop frequency</i>	<i>Line edit</i>	Cette valeur sera le point d'arrêt du balayage de fréquence.	La valeur doit obligatoirement être un nombre entier et doit se trouver dans un intervalle précis.
<i>Spacing</i>	<i>Line edit</i>	L'utilisateur peut indiquer ici quel écart de fréquence séparera deux fréquences de sortie consécutives.	La valeur doit obligatoirement être un nombre entier et doit se trouver dans un intervalle précis.
<i>Time delay</i>	<i>Line edit</i>	Un délai peut être ajouté par l'utilisateur. Cette valeur, en millisecondes, représentera la durée séparant deux générations de fréquences consécutives.	La valeur doit obligatoirement être un nombre entier et doit se trouver dans un intervalle précis.
<i>Current output frequency</i>	<i>Spin box</i>	La fréquence de sortie courante est indiquée dans ce champ. Il s'agit de la fréquence qui sera disponible en sortie du VCO.	Cette valeur sera incrémentée dynamiquement de la valeur indiquée dans le champ " <i>Spacing</i> " et cela au bout d'un temps égal au " <i>Time delay</i> ".
<i>Time remaining</i>	<i>LCD number</i>	En fonction du délai, du pas de fréquence et de l'intervalle renseigné, le programme peut calculer un temps approximatif au bout duquel l'opération de balayage sera terminée.	Le temps affiché évolue dynamiquement, étant décrémenté d'un " <i>Time delay</i> " après un temps égal à cette même valeur. Le format de la donnée affichée est : "00 jours 00:00:00".

Figure 63 : description des paramètres présents dans l'onglet "*Sweep*" de l'application

Remarque : en plus du temps restant affiché, l'utilisateur a un autre moyen pour suivre l'évolution du processus de balayage. En effet, une barre de progression située juste en dessous du libellé « *Current output frequency* » renseigne sur l'avancement de la tâche.

De façon à pouvoir lancer une opération « *Sweep* », l'utilisateur devra s'assurer d'avoir correctement complété les paramètres présents dans l'onglet « *Main settings* ». En effet, le processus « *Sweep* » implique une programmation de la PLL qui a donc besoin de certains paramètres nécessaires à son bon fonctionnement. En revanche, les champs « *Channel* » ainsi que « *RF VCO Output Frequency* » du premier onglet ne seront pas pris en compte lors d'un processus « *Sweep* ».

Une fois l'ensemble des valeurs des paramètres des deux onglets attribué, l'utilisateur peut démarrer le processus en appuyant sur le bouton « *Start* ».

Il est toujours possible pour l'utilisateur d'interrompre l'opération en appuyant sur le bouton « *Stop / Reset* ». En plus, d'interrompre le procédé, l'appui sur ce bouton aura pour effet de réinitialiser la valeur affichée dans le champ « *Current output frequency* » et réinitialisera également la barre de progression ainsi que le temps restant.

Dans l'onglet « *Sweep* », nous retrouvons les valeurs associées à des champs vides au

lancement de l'application (voir figure 62 p.56). De même que pour l'onglet « *Main settings* », les valeurs associées à ces libellés seront affichées dès l'appui sur le bouton « *Start* ». Comme lors d'une opération « *Auto Sweep* », les valeurs correspondant aux champs « *N Counter* », « *B Counter* », « *A Counter* » et « *AB Counter Latch* » changeront dynamiquement au cours du temps, s'adaptant à l'incrément de la fréquence de sortie courante à générer en sortie du VCO.

Dans l'onglet « *Sweep* » de l'application, nous retrouvons les boutons « *Save settings* » et « *Load settings* », dans le coin inférieur gauche de la fenêtre, qui ont exactement les mêmes fonctions que les boutons déjà présents dans l'onglet « *Main settings* » (voir figure 61 p.55).

De plus, le bouton « *Quit* » est également présent et permet à l'utilisateur de quitter l'application à partir de l'onglet « *Sweep* ».

Une partie du travail réalisé sur l'interface graphique consiste à gérer les exceptions qui peuvent être générées par le programme.

Prenons par exemple le champ « *Reference Frequency* » (onglet « *Main settings* ») dans lequel un utilisateur devra entrer une valeur de fréquence en MHz. Une fois le bouton « *Write PLL* » ou « *Auto Sweep* » appuyé, le programme récupère la valeur entrée et effectue des vérifications avant de la traiter.

Il y a deux vérifications à réaliser :

- La première permet de s'assurer que la donnée est bien un nombre entier, et non pas une chaîne de caractères ou des symboles. Pour ce faire, une fonction de Python intitulée « *re.match* » a été utilisée. Cette fonction prend deux arguments en paramètres : une chaîne de caractères constante et la chaîne de caractères à tester. Le deuxième argument est comparé au premier et si les deux concordent, une valeur différente de « *None* » est renvoyée. La chaîne de référence, qui correspond au premier argument de la fonction, est en réalité une expression régulière qui a la forme suivante : "[0-9]{2,3}(?![\\d.])". La partie « [0-9]{2,3} » indique que la chaîne entrée doit être un nombre de deux ou trois chiffres et le « (?![\\d.]) » implique que le nombre sur deux ou trois chiffres ne doit pas être suivi par un autre chiffre. Autrement dit, la chaîne d'entrée doit être un nombre de deux ou trois chiffres strictement. Si cette condition n'est pas satisfaite, un message d'erreur est affiché :

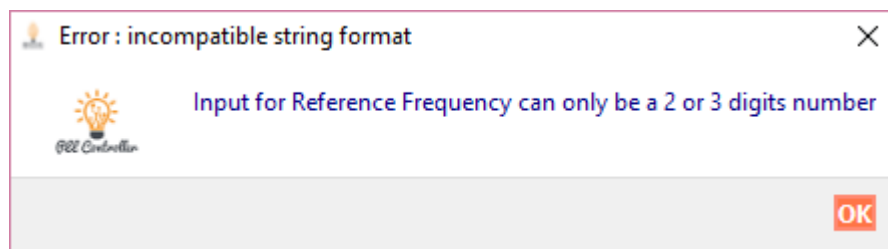


Figure 64 : message d'erreur renvoyé par l'application lorsque la valeur de la "Reference Frequency" n'est pas un nombre sur deux ou trois chiffres

- La deuxième vérification est destinée à voir si la valeur d'entrée est bien comprise dans un intervalle précis. Pour la « *Reference Frequency* », cet intervalle est 20 – 250 MHz. Une fois encore, si cette condition n'est pas satisfaite, une erreur est renvoyée :

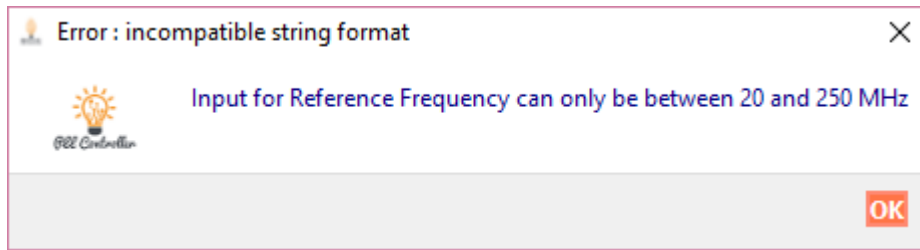


Figure 65 : message d'erreur renvoyé par l'application lorsque la valeur de la "Reference Frequency" n'est pas incluse dans l'intervalle fixé

Ces vérifications sont conduites pour chacun des champs de texte que l'utilisateur doit remplir et les messages d'erreur sont du même type que ceux présentés ci-avant, en adaptant le contenu du message à la vérification réalisée.

D'autres messages de ce type peuvent apparaître après un appui sur les boutons présents dans les onglets car le programme effectue toute une batterie de tests avant de pouvoir traiter et envoyer les informations. A titre d'exemple, des vérifications sont effectuées avant d'utiliser le port série ou avant de charger les données présentes dans un fichier texte.

De plus, des messages qui ne sont pas des messages d'erreur peuvent apparaître, par exemple pour renseigner l'utilisateur sur le succès d'une opération de sauvegarde :

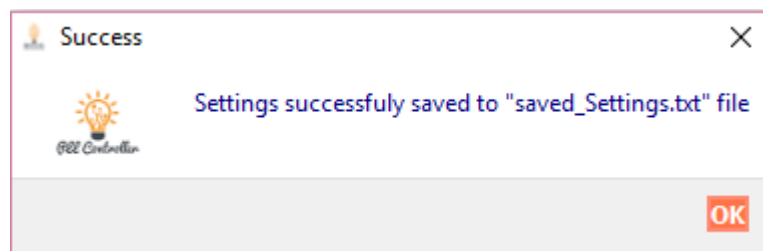


Figure 66 : message renvoyé par l'application après le succès d'une opération de sauvegarde

Une fois les derniers réglages effectués, je me suis soucié de la portabilité de l'application car, si le programme est destiné à tourner sur différentes machines et systèmes d'exploitation différents, il faudra s'assurer qu'il fonctionne dans tous les cas de figure.

Au cours de mes recherches, j'ai trouvé un logiciel de Python qui permet de transformer une application écrite dans ce langage en un unique fichier exécutable. En suivant pas à pas la marche à suivre, je suis parvenu à produire un fichier « .exe » qui a fonctionné sur mon ordinateur personnel ainsi que sur un autre ordinateur ayant une version de « Windows » plus ancienne.

J'ai pu tester l'ensemble de l'application en branchant les signaux SPI du module USB à un oscilloscope pour visualiser les trames binaires transmises. J'ai ensuite zoomé sur la trame correspondant au premier registre transmis à la PLL :

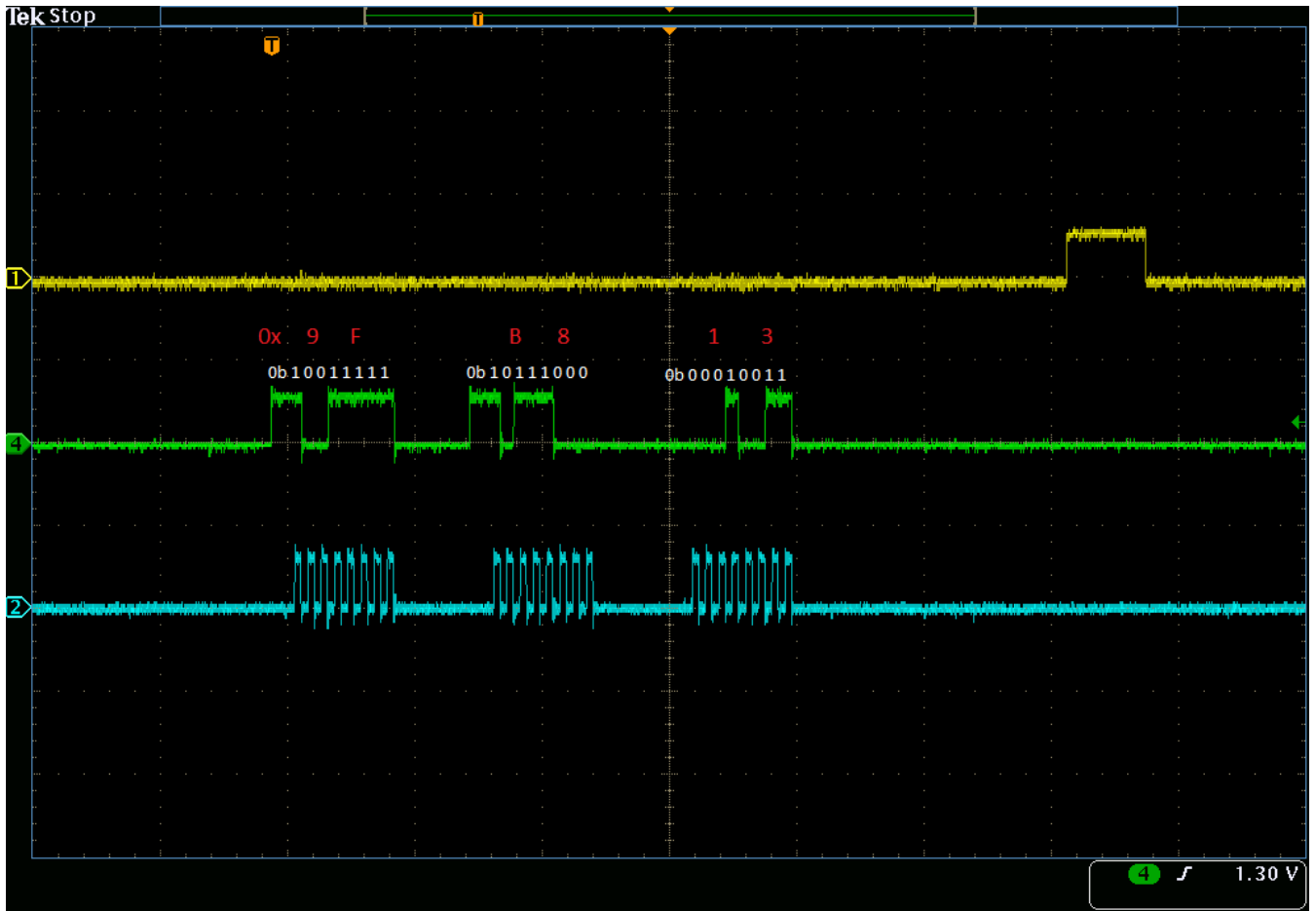


Figure 67 : visualisation d'une trame binaire à l'oscilloscope

La valeur binaire correspond bien à celle indiquée par l'application :

```
Initialization Latch = 0x9FB813
Function Latch =      0x9FB812
R Counter Latch =     0x002710
AB Counter Latch =    0x032C41
```

Figure 68 : valeurs des registres affichées dans l'application

La même opération a été répétée afin de vérifier les valeurs des trois registres restants.

Enfin, cet enchaînement a été reproduit pour chacune des opérations de l'application impliquant un envoi de données à la PLL : « *Write PLL* », « *Auto Sweep* » et « *Sweep* ».

Une fois le PCB de l'institut reçu et alimenté, j'ai eu l'occasion de tester mon application dessus. Malheureusement, la PLL ne semble pas réagir aux commandes qui lui sont envoyées car la fréquence de sortie reste désespérément bloquée à 100 MHz. Pourtant, un ingénieur de l'institut et moi-même, avons eu la certitude que le composant recevait bien les trames binaires transmises par la liaison SPI et qu'il y répondait. En effet, en modifiant l'un des paramètres de l'interface graphique et en envoyant les commandes, nous remarquons un changement notable en plaçant une sonde d'oscilloscope sur l'une des broches de sortie de l'ADF4108.

Après quelques tâtonnements et tests, plusieurs petits soucis techniques ont été décelés

et tous ont pu être corrigés en réalisant maints bricolages. Cette phase de débogage électronique a pris un certain temps car la majorité des composants utilisés pour le PCB sont extrêmement petits et nécessitaient une attention particulière lors des manipulations.

Mais finalement, j'ai eu l'opportunité de tester mon interface durant la dernière semaine de mon stage et nous avons constaté que la PLL réagit favorablement aux commandes envoyées. En effet, il est possible de constater les changements de la fréquence de sortie sur un analyseur de spectre en programmant la PLL avec des valeurs de fréquences différentes. L'objectif du stage a donc été atteint.

c. Autres tâches effectuées

➤ Manuel d'utilisation de l'application écrit avec LaTeX :

Une fois l'interface graphique codée, j'ai entrepris de rédiger un manuel d'utilisation à l'attention des personnes qui pourrait se retrouver à utiliser mon application à l'avenir. Etant donné que cette tâche ne m'a pas été expressément demandée, j'ai décidé d'utiliser un éditeur de texte que je n'avais jamais encore utilisé auparavant : LaTeX. En effet, ce logiciel est très utilisé et apprécié lorsqu'il s'agit de rédiger des documents techniques (scientifiques, informatiques ...) et correspond donc parfaitement à l'utilisation que je souhaite en faire. Par ailleurs, utiliser LaTeX me permet d'acquérir une expérience supplémentaire qui pourra surement se révéler utile à l'avenir.

Je me suis appuyé sur plusieurs exemples de manuel d'utilisation de logiciels glanés sur internet avant de mettre au point mon propre document. Ces recherches documentaires m'ont permis de structurer au mieux le contenu du manuel et de décrire le fonctionnement de l'application avec le plus de précision possible.

Finalement, le manuel couvre tout ce qui touche de près ou de loin à mon application, allant de comment ouvrir l'interface à comment la fermer, en passant par une explication détaillées de toutes les fonctionnalités offertes.

Ce document sera convertit au format PDF et joint aux fichiers que je remettrai à mon responsable à la fin du stage.

A titre d'information, la figure suivante expose le sommaire du manuel :

Contents

List of Figures	2
1 Introduction	3
1.1 Intended readership	3
1.2 Applicability	3
1.3 Purpose	3
1.4 How to use this document	3
1.5 Software characteristics	3
1.6 Related documents	3
1.7 Problem reporting	4
2 Overview	5
3 How to get the application running	6
4 How to use the PLL Controller Software	8
4.1 Prerequisites	8
4.2 "Main settings" tab's operations	9
4.2.1 Write PLL operation	9
4.2.2 Save settings option	13
4.2.3 Load settings option	15
4.2.4 Auto sweep operation	16
4.3 "Sweep" tab's operation	18
4.3.1 Prerequisite	18
4.3.2 Sweep operation	19
4.3.3 Load and save settings operations	23
5 How to quit the application	24
A Error messages and recovery procedures	27
B Glossary and abbreviations	31
C References	33

Figure 69 : sommaire du manuel d'utilisation écrit avec LaTeX

Remarque : le manuel d'utilisation a été entièrement rédigé en anglais afin de pouvoir être compris par le plus grand nombre. L'IASB comprenant néerlandophones et francophones, il convenait de pouvoir satisfaire l'un et l'autre.

➤ Conception d'une PLL complète en simulation avec ADIsimPLL :

La société *Analog Devices* propose un logiciel permettant de simuler le fonctionnement d'une boucle à verrouillage de phase construite avec ses produits. Les résultats de la simulation sont présentés sous la forme d'un rapport, accompagné de plusieurs graphiques montrant, par exemple, la réponse en fréquence de la boucle.

J'ai mené cette simulation afin d'en apprendre davantage sur le comportement des PLL et de pouvoir me rendre compte par moi-même des résultats qu'il est possible d'obtenir avec un montage spécifique.

Premièrement, j'ai suivi le tutoriel complet proposé par le logiciel lors de la première utilisation. Ensuite, j'ai créé une nouvelle simulation au cours de laquelle j'ai pu sélectionner les composants constitutifs du montage à base de PLL mis au point par l'IASB (PLL, filtre de boucle, VCO ...). Une fois ces composants sélectionnés, il est possible de renseigner les paramètres RF telle que la fréquence de référence du système ou encore la fréquence de référence du détecteur de phase.

La simulation peut ensuite être lancée et le logiciel affiche les résultats qui sont affichés

dans plusieurs onglets. L'un permet de visualiser les caractéristiques des composants sélectionnés, un autre affiche les résultats obtenus dans le domaine fréquentiel et le suivant dans le domaine temporel. Le logiciel propose également le schéma électronique final, en y intégrant les composants sélectionnés précédemment, ainsi qu'un rapport textuel.

Il est ensuite possible de jouer sur différents paramètres des composants afin d'obtenir les meilleurs résultats possibles. Par exemple, il est possible de modifier la marge de phase ou la bande passante du filtre de boucle et d'observer son influence sur le graphique montrant l'évolution de la fréquence de sortie au cours du temps. On remarque ainsi que plus l'on diminue la bande passante, plus la fréquence de sortie prend du temps avant de se stabiliser à la fréquence finale désirée. D'autre part, on constate également que le temps de verrouillage de la boucle augmente lorsque la bande passante du filtre diminue.

En outre, étant donné que les composants d'une PLL sont sources de bruit de phase, le logiciel offre la possibilité d'injecter du bruit de phase dans certains composants afin d'évaluer l'impact que cela aura sur le comportement de la boucle.

Après plusieurs réglages et tentatives ratées, je suis parvenu à m'approcher le plus possible du comportement attendu pour notre application, à savoir un temps de verrouillage suffisamment rapide, une fréquence de sortie stable après un temps relativement court et un bruit de phase total ayant un niveau faible.

La figure suivante montre quelques graphiques obtenus lors de la simulation :

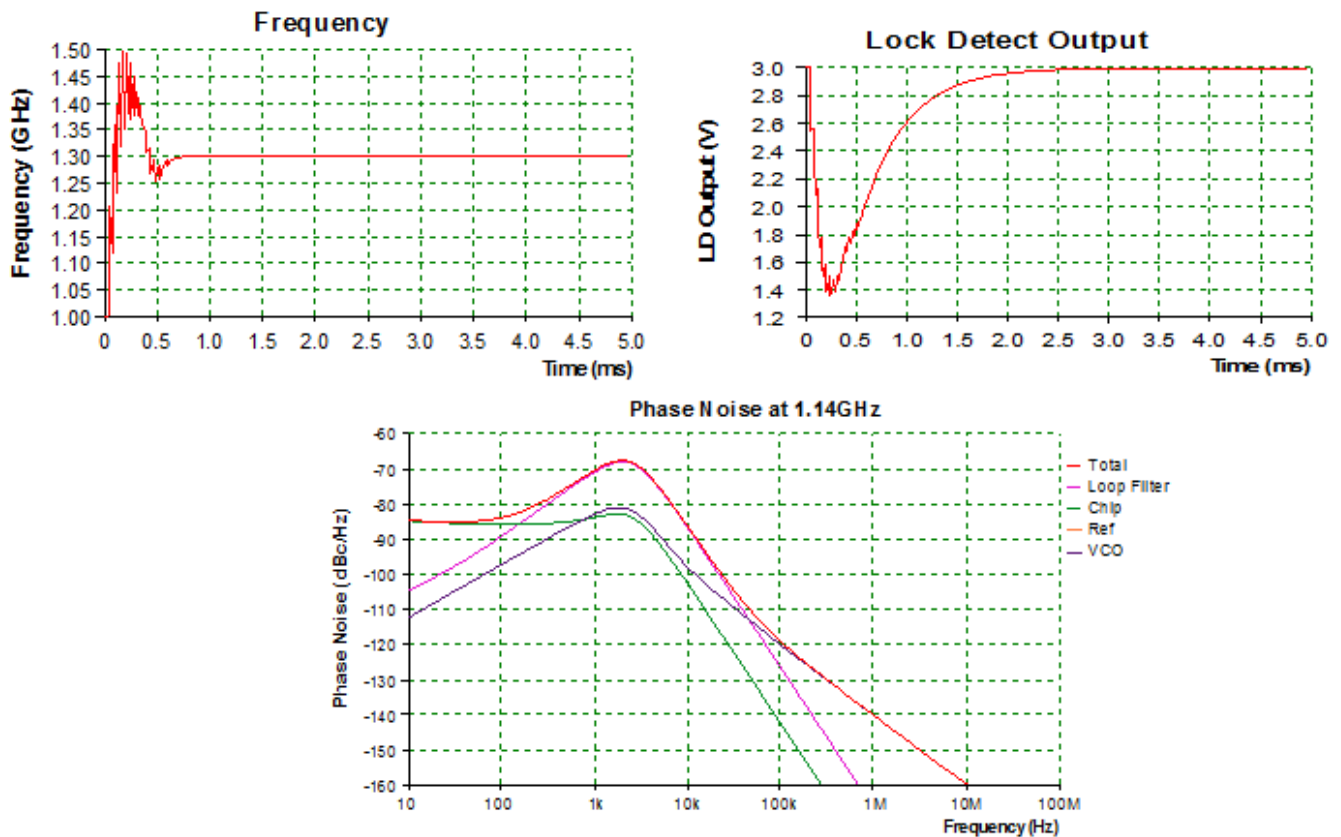


Figure 70 : graphiques obtenus après simulation de la PLL sur ADIsimPLL

J'ai tout de même trouvé cette simulation très poussée et relativement complexe pour quelqu'un qui n'a pas l'habitude de travailler avec des PLL depuis longtemps. Cela représentait

toutefois un bon exercice pour avoir une vue approximative des résultats qu'il est possible d'obtenir avec l'utilisation d'une boucle à verrouillage de phase complète.

➤ Interface graphique avec LabWindows/CVI :

L'interface graphique codée en langage Python a été terminée avant la réception du PCB donc je n'étais pas en mesure d'affirmer que l'application fonctionnera avec la PLL. Afin d'assurer quelque peu mes arrières et disposer de plusieurs cordes à mon arc, j'ai décidé de réaliser une autre interface graphique, dans un langage différent.

En me remémorant les projets réalisés au cours de l'année à l'ESEO, je me suis souvenu de LabWindows/CVI qui est un environnement de développement en langage C ANSI spécialisé dans la conception d'interface graphique et proposé par l'entreprise *National Instruments*. Cet outil est tout indiqué pour réaliser des interfaces de tests et/ou de mesures et, après quelques recherches sur internet, j'ai pu affirmer que LabWindows/CVI me permettra de mener à bien mon projet.

Je me suis donc rendu sur le site internet de *National Instruments* afin de télécharger la version d'essai de CVI ainsi qu'un tutoriel complet. La première étape de mon travail a donc consisté à suivre ce tutoriel, puis d'effectuer en autonomie les exercices supplémentaires proposés à la fin du guide.

Une fois les bases acquises, j'ai pu commencer le véritable projet. Au début d'un nouveau projet, CVI propose d'abord de construire l'aspect visuel de l'interface graphique en plaçant tous les éléments que l'on souhaite afficher dans l'application. Pour chaque élément placé, il est possible de lui associer une fonction de rappel qui sera exécutée dès qu'un certain événement aura lieu sur l'élément. Les événements auxquels un élément sera sensible peuvent être choisis parmi une vaste liste de choix dans laquelle nous retrouvons les clics gauche et droit de la souris mais aussi un appui sur la touche « entrée », par exemple.

Ensuite, il est possible de générer le code en langage C ANSI correspondant à l'interface construite. Cette génération de code comprend la génération de la fonction « *main* » ainsi que de l'ensemble des fonctions de rappels spécifiées lors de la phase précédente. Ces fonctions de rappel contiennent une structure conditionnelle de type « *switch case* » avec une condition pour chaque événement choisi :

```
int CVICALLBACK RefFreq (int panel, int control, int event,
                        void *callbackData, int eventData1, int eventData2)
{
    switch (event)
    {
        case EVENT_COMMIT:
            break;
        case EVENT_RIGHT_CLICK:
            break;
        case EVENT_VAL_CHANGED:
            break;
    }
    return 0;
}
```

Figure 71 : exemple d'une fonction de rappel générée par LabWindows/CVI

Dans l'exemple ci-dessus, l'élément associé à la fonction est un champ de texte dans lequel l'utilisateur devra entrer la valeur de la fréquence de référence du système. Cet élément sera

sensible à trois événements : `EVENT_COMMIT` (l'utilisateur réalise une action de validation, comme un appui sur la touche « entrée »), `EVENT_RIGHT_CLICK` (l'utilisateur effectue un clic droit de la souris sur l'élément) et `EVENT_VAL_CHANGED` (l'utilisateur modifie la donnée contenue dans le champ).

Ensuite, il m'a fallu compléter certaines fonctions de rappel mais aussi créer de nouvelles fonctions en vue d'obtenir le comportement souhaité pour mon application.

L'architecture générale de l'application d'un point de vue graphique est la même que celle proposée pour l'interface graphique codée en Python. Toutefois, le langage C ANSI étant un langage beaucoup moins modulable que Python, le travail sur LabWindows/CVI a été légèrement plus laborieux. Par exemple, l'utilisation des ports séries avec LabWindows nécessitait beaucoup plus de paramétrages que pour leur emploi avec Python.

L'interface graphique finale est présentée dans la figure suivante :

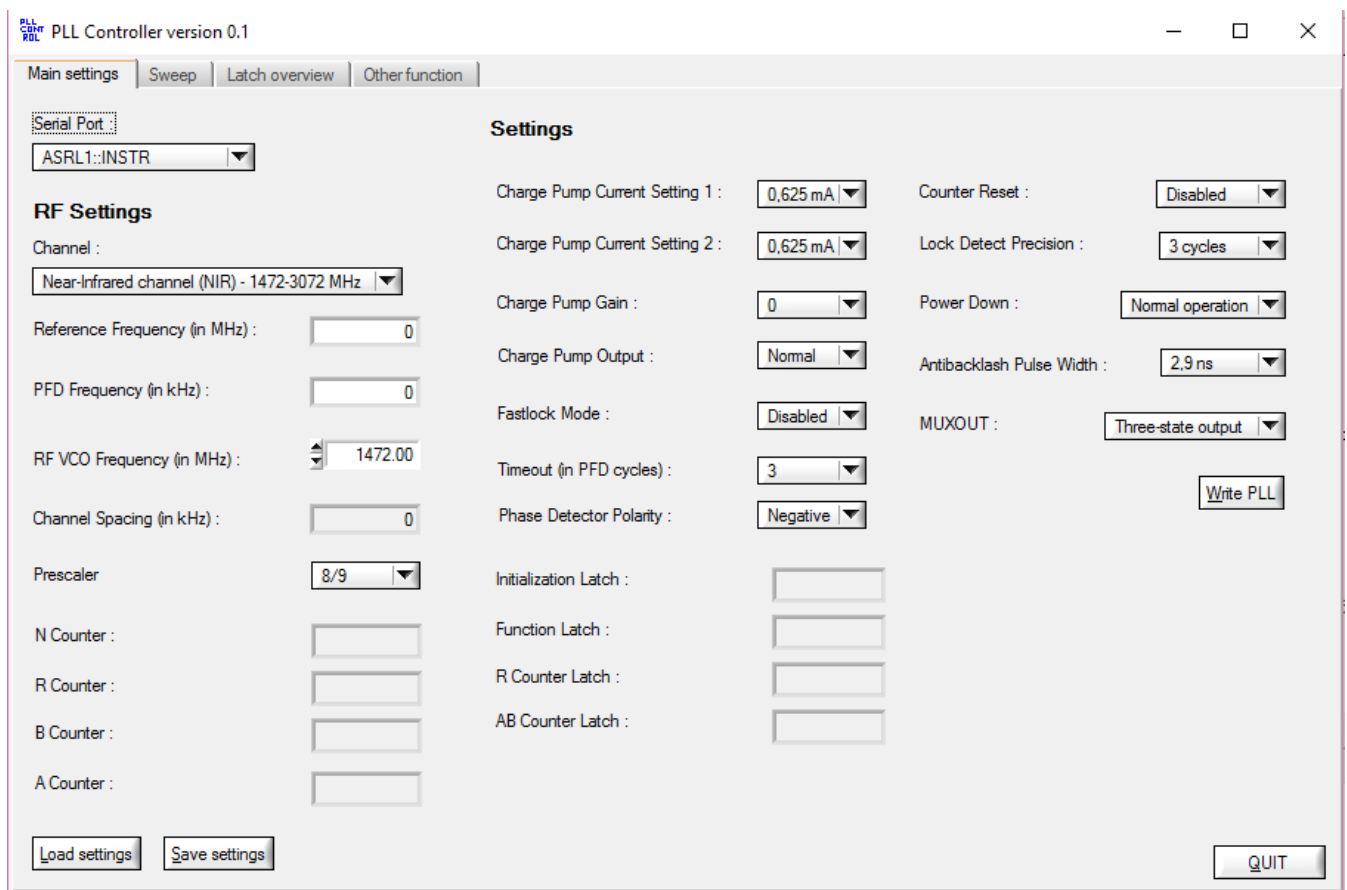


Figure 72 : interface graphique réalisée sur LabWindows/CVI

Les deux environnements, LabWindows/CVI et l'IDLE de Python, se ressemblent par certains aspects mais conservent un langage qui leur est propre. Avoir pu me frotter aux deux a grandement renforcé mes compétences dans le domaine de la programmation, et pas seulement pour la création d'interfaces graphiques seules. Bien que du temps ait été passé sur le placement des éléments de l'interface, la plus grosse partie du travail s'est effectuée en arrière-plan, pour assurer un fonctionnement optimal des opérations proposées par l'application.

III - Rapport personnel d'étonnement

1) Compétences acquises

a. Scientifique et technique

D'un point de vue scientifique et technique, je suis convaincu que ce stage m'a énormément apporté. Le fait que la carte électronique ait eu quelques soucis techniques, rendant la réalisation de tests limitée, j'ai pu occuper mon temps disponible à l'obtention de nouvelles compétences. C'est ainsi que j'ai pu m'essayer à LaTeX, à LabWindows/CVI ainsi qu'à ADIsimPLL en plus du langage Python. J'ai également eu l'occasion de programmer un microcontrôleur de type PIC18 ce qui correspondait parfaitement aux activités réalisées lors du semestre 8 en option EOC.

Le travail principal, consistant en la programmation du PIC18 et la réalisation de l'interface graphique avec Python, a été l'occasion de mener des recherches plus poussées dans ces deux domaines qui ont été abordés à l'ESEO. De même, nous avons eu l'occasion d'effectuer un travail pratique sur LabWindows/CVI en option mais il s'agissait d'un projet très court qui ne donnait qu'un aperçu des fonctionnalités proposées par l'IDE.

L'utilisation de LaTeX et d'ADIsimPLL provient d'une envie personnelle d'acquérir de nouvelles connaissances, tout en les adaptant aux activités prévues par mon stage. Le manuel d'utilisation aura l'avantage d'être clair et détaillé grâce à LaTeX, et la simulation de la boucle à verrouillage de phase m'a permis de mieux comprendre certaines notions abordées dans les documentations techniques se rapportant aux PLL.

Par ailleurs, l'emploi du protocole USB au cours du projet constitue une première pour moi et je suis persuadé que je pourrai mettre à profit cette compétence dans mes futurs projets au sein de l'option EOC.

Les interfaces graphiques réalisées au cours du stage nécessiteraient d'être améliorées, mais les fonctionnalités principales semblent fonctionner (d'après les trames binaires visibles à l'oscilloscope et le petit test réalisé sur le PCB). Toutefois, j'aurai aimé pouvoir déboguer entièrement la carte électronique afin de vérifier le fonctionnement de l'application dans son intégralité.

Une autre facette de mon stage qui a éveillé ma curiosité scientifique est le fait de travailler de près avec le domaine spatial. J'ai eu l'occasion de poser beaucoup de questions au responsable du projet ALTIUS sur le déroulement de la mission dans sa globalité ainsi que sur d'autres missions du même type.

b. Organisationnel

Il a été très intéressant pour moi d'avoir pu obtenir un stage à l'étranger car cela permet de se rendre compte des différences qu'il existe entre la France et la Belgique du point de vue management ou organisation d'une entreprise. L'institut est, toutefois, différent d'une entreprise classique car les principaux départements sont liés à l'activité propre de l'institut qui est l'aéronomie. Ainsi, au lieu de retrouver un département « Logistique » ou « Achats » présents dans l'organisation de nombreuses entreprises, nous retrouvons un service « Physique spatiale » ou « Composition atmosphérique ».

J'ai eu l'occasion de parcourir le site internet de l'IASB avant et au cours de mon stage afin de savoir quelle genre de missions sont organisées par la Belgique dans le domaine spatial.

C'est au cours de ces recherches que j'ai pu constater que le projet ALTIUS représente un enjeu important pour l'institut que je plus qu'heureux d'avoir pu apporter une petite contribution dans ce vaste projet.

Par ailleurs, j'ai appris que le responsable du projet ALTIUS réalise une thèse sur le pilotage d'un AOTF par radio fréquences dans le domaine spatial ce qui a été un moteur supplémentaire dans l'accomplissement de mes tâches.

c. Relationnel

Tout au long de mon stage, j'ai été affecté au service « *Engineering* » de l'institut qui regroupe ingénieurs et techniciens apportant leur soutien aux scientifiques dans la réalisation d'instruments de mesure notamment. L'ensemble de l'équipe composant ce service m'ont réservé un accueil des plus chaleureux et ont mis un point d'honneur à me faire sentir intégré et considéré au sein de l'institut.

De plus, mon tuteur de stage, la responsable du département électronique ainsi que le responsable du projet ALTIUS effectuaient un suivi régulier de mon avancement, en proposant des solutions ou me demandant d'ajouter des fonctions supplémentaires aux applications. J'ai également pu leur faire part de mes initiatives personnelles qui ont été plutôt bien vues et encouragées.

Lors de la phase préparatoire au cours de laquelle j'ai travaillé sur le DDS, j'ai pu poser toutes mes questions à un ingénieur du service qui m'a très bien guidé dans mon travail et m'a fait part de son expérience ce qui a été très formateur pour moi.

d. Economique

Au cours de mes entretiens avec le responsable du projet, la question du budget accordé à la partie pilotage de l'AOTF a été abordée. J'ai été très surpris de la rapidité à laquelle les prix grimpent lorsqu'il est question de faire qualifier un composant électronique spatialement. Premièrement, il est difficile de trouver une entreprise qui propose cette qualification, ce qui rend les choix de prix assez réduits. Deuxièmement, une fois l'entreprise trouvée, cette-dernière doit effectuer toute une batterie de tests coûteux qui permettront d'accorder la qualification spatiale au composant.

Ces deux aspects rendent le budget des projets spatiaux assez conséquent et il est difficile de s'y conformer pour les ingénieurs de l'institut qui doivent sélectionner l'électronique nécessaire à l'aboutissement de la mission.

Le PCB sur lequel était axé mon travail représente un coût non négligeable, notamment à cause de composants ayant une qualification spatiale dont le prix s'élève à la dizaine de milliers d'euros.

2) Conclusion

Mon stage au sein de l'IASB a été l'occasion de mettre à profit des connaissances acquises lors de mes études à l'ESEO, et notamment en option EOC, pour le développement d'un projet concret ayant pour but la surveillance du changement planétaire.

Ma mission était d'implémenter une interface permettant d'envoyer des commandes à une carte électronique depuis un ordinateur. Ce travail permettra de piloter un composant électronique, PLL ou DDS, en vue de générer une certaine gamme de fréquences. Ces fréquences permettront ensuite de contrôler un filtre acousto-optique qui isolera des longueurs d'ondes en fonction des fréquences d'entrée. L'ensemble permettra, à terme, de former une caméra spectrale pouvant être embarquée à bord d'un satellite.

Les tâches que j'ai réalisées permettront aux ingénieurs de l'institut de valider, ou non, une solution concernant le générateur de fréquences. Si les tests sont concluants, ils pourront passer à l'étape suivante qui sera d'acquérir l'ensemble des composants qualifiés spatialement pour former la carte électronique finale pouvant être placée à bord du satellite.

Avoir pu choisir le langage informatique que je souhaitais et pouvoir programmer en autonomie m'a permis de faire face à des problèmes plus ou moins retors qu'il m'a fallu résoudre seul tout au long du stage. Cela s'est révélé très instructif et a participé à améliorer mes compétences en programmation.

Par ailleurs, j'ai eu l'occasion d'étudier des documentations techniques se rapportant à des composants électroniques complexes afin de comprendre comment les programmer efficacement. Ces composants ont été abordés lors de certains cours de l'ESEO mais j'ai pu approfondir mes connaissances lors de ce stage par une étude plus poussée.

D'autre part, les tests et débogages réalisés sur la carte électronique de l'institut ont été de bons exercices pour mettre en application et compléter mes acquis dans le domaine de l'électronique.

De plus, l'utilisation de LabWindows/CVI et de LaTeX a été une réelle plus-value qui me servira indubitablement dans la suite de mes études et très certainement lors d'un futur stage ou emploi. En effet, ces deux outils sont bien connus des instances scientifiques, notamment l'informatique, et avoir été confronté à ces savoir-faire sera probablement bien vu par un futur employeur.

Un autre détail, mais qui a tout de même de l'importance, est la langue anglaise que j'ai utilisée pour dialoguer avec certains néerlandophones de l'institut. Associer l'anglais à un discours technique n'était pas des plus aisé mais s'est révélé être un exercice formateur.

Le fait d'avoir évolué dans un institut tourné vers le spatial était réellement intéressant pour satisfaire ma curiosité dans ce domaine. J'ai pu passer du temps à parcourir les sites internet de l'IASB expliquant les projets passés, présents et futurs, ainsi qu'à lire des revues scientifiques disponibles sur mon lieu de stage.

En conclusion, ce stage m'aura permis de confirmer mon choix d'option car c'est toujours l'électronique qui m'intéresse le plus parmi d'autres domaines. J'ai également pu acquérir des savoir-faire supplémentaire qui pourront m'être utiles dans le futur. Avoir été rattaché à l'institut

d'aéronomie a été une expérience unique, profondément enrichissante et il n'y a que des aspects positifs qui peuvent ressortir de ce stage.

ANNEXE 1 : CV

Benoit LADRANGE

25/03/95 20 ans

31 résidence du Puy Garnier, Angers, France

Tel : 06 03 04 77 59

Mail : benoit.ladrangle@reseau.eseo.fr



Expériences

Caissier en grande surface

Carrefour - Bruxelles - Juillet 2015

- Réaliser les opérations d'encaissement des achats.
- Assurer le bon déroulement du passage en caisse.
- Accueillir et fidéliser les clients.



Vacataire à l'état-major des armées

Etat-major des armées - Paris - Juillet 2014 et juillet 2013

- Traitement du courrier interne et externe.
- Archivage et classement de dossiers.
- Assister les secrétaires du bureau.



Employé de piscine

Association syndicale libre du domaine des Gâtines - Plaisir - Août 2012 et août 2011

- Nettoyage des bassins.
- Remise à niveau des conditions de bon fonctionnement de la piscine (niveau d'eau, sanitaires, tests de pH et de chlore).
- Propreté des abords de la piscine.

Formation

Ingénieur (ESEO – Ecole Supérieure d'Electronique de l'Ouest)

Septembre 2012 - En cours

Actuellement en deuxième année de cycle ingénieur.

Gradué en sciences de l'ingénieur (*Bachelor of Engineering*) obtenu en 2015.

Séjour d'un mois à l'université de Plymouth (Royaume-Uni) pour suivre des cours d'anglais et réaliser un projet informatique.

Séjour de deux mois à Shanghai (Chine) pour suivre des cours de culture chinoise, business en Chine, de logique combinatoire et séquentielle et réaliser un projet informatique.

BAC Scientifique

Plaisir (78) – 2012

BAC S obtenu avec mention bien.

Scolarité en école anglaise (3 ans) et en Belgique (3 ans).

Compétences

Informatique : programmation en langage C ; Word, Powerpoint, Excel ; MATLAB/Simulink ; langage assembleur (programmation de microcontrôleurs) ; VHDL ; XML ; SQL ; Java.

Electronique : manipulation de cartes à microprocesseurs (prototype de voiture télécommandée, prototype de petit système d'alarme) et de FPGA.

Réalisation de cartes électroniques (télémètres à ultrasons, fusible électronique).

Langues : anglais : courant, niveau B2 avec obtention de FCE et du TOEIC (970 points / 990) ; allemand : niveau scolaire ; mandarin : obtention du diplôme HSK niveau 2 ; français : langue maternelle, certification

Voltaire (740 points / 1000).

Centres d'intérêt

Voyages : République tchèque, Royaume-Uni, Allemagne, Grèce, Egypte, Tunisie, Espagne, Maroc, Pays-Bas, Belgique, Italie, États-Unis, Chine.

Sports : badminton, course à pied, natation.

Associatif : 2011 : rénovation du mur d'enceinte du village de Rodemack (57).

2010 : rénovation de l'intérieur de l'école du village d'Oytier-Saint-Oblas (38).

ANNEXE 2 : Planning du stage

Date	Tâches réalisées
06/06/2016	Explication du projet et de l'état d'avancement du projet utilisant le DDS. Récupération des fichiers/documents utiles. Recherche des <i>datasheets</i> et autres documentations techniques. Lecture et assimilation.
07/06/2016	Explication du projet et de l'état d'avancement du projet utilisant la PLL. Présentation des équipements déjà utilisés : l'AOTF (AOTF utilisé pour les UVs est unique au monde, construit par les russes) ainsi que la maquette taille réelle du satellite. Reprise du code de la pile USB de <i>Microchip</i> . Test de la liaison USB (<i>Docklight</i>) et initialisation de la communication SPI pour envoyer les données vers le DDS.
08/06/2016	Test BP+LED et de la communication SPI. Formation personnel sur la création d'une interface graphique avec Python. L'interface graphique permettra d'entrer la fréquence ainsi que l'amplitude du signal à synthétiser en sortie du DDS.
09/06/2016	Poursuite de l'interface graphique avec Python afin de piloter le DDS depuis un ordinateur par liaison USB.
10/06/2016	Test de l'interface graphique afin de vérifier si elle permet bien de contrôler le DDS. Recherche d'informations sur les PLL.
13/06/2016	Poursuite des recherches sur les PLL. Lecture de la <i>datasheet</i> de l'ADF4108, de l'AD820 (filtre) et du ROS-2500-319+ (VCO).
14/06/2016	Fin de lecture de la <i>datasheet</i> de la PLL et début d'écriture du code sur MPLABX IDE.
15/06/2016	Poursuite de l'écriture du code sur MPLABX. Différents calculs effectués pour programmer les registres avec les bonnes valeurs.
16/06/2016	Entretien avec un des responsables du projet, mise au point sur les différentes fonctionnalités à mettre en place pour la PLL. Mise en place d'une nouvelle façon pour modifier de façon pratique les bits des registres (utilisation d'une union/struct => <i>bitfields</i>).
17/06/2016	Poursuite de l'écriture du programme. Ecriture d'un algorithme permettant d'incrémenter la fréquence de sortie d'un certain pas afin de couvrir tout le spectre propre à chaque étude (UV, NIR et VIS). Traduction des équations déterminées auparavant sous forme de calculs en langage C. Rendre le programme facile à comprendre et à utiliser (respect de la convention de nommage, mise en place de fonctions spécifiques, commentaires ...). Vérification, également, des bonnes valeurs présentes dans les registres (en mode debugger sur MPLABX). Test avec <i>Docklight</i> => valeur hexadécimale de la fréquence de sortie renvoyée sur <i>Docklight</i> .
20/06/2016	Poursuite des tests et débogage (<i>Docklight</i> et oscilloscope). Comparaison des signaux apparaissant à l'oscilloscope avec le chronogramme de la <i>datasheet</i> . Relevé des constantes de temps. Prise de captures d'écran. Réflexion sur comment faire le code dynamique avec une GUI Python.
21/06/2016	Réflexion sur comment faire le code de l'interface graphique permettant à un utilisateur de contrôler la PLL. Début de l'écriture de l'interface en Python. Ecriture du code sur MPLABX permettant de transférer les données arrivant via USB vers la PLL via SPI.
22/06/2016	Poursuite de l'écriture de l'interface graphique, tests et débogage. Revérification de la SPI (petites retouches au code sur MPLABX). Fini de programmer la partie gauche de l'interface graphique.
23/06/2016	Fin de l'écriture du code principal de l'interface graphique, petites retouches esthétiques.

24/06/2016	Test et débogage du système complet (avec le code du PIC). Visualisation des trames sur l'oscilloscope : test concluant. Un responsable m'a demandé d'ajouter quelques fonctionnalités dans l'interface graphique. Ecriture des commentaires du code Python. Recherches sur la gestion de fichiers textes avec Python pour la nouvelle fonctionnalité.
27/06/2016	Rajout de certaines fonctionnalités à l'interface graphique : sauvegarder et charger les paramètres avec un fichier .txt + possibilité de sélectionner quel canal on souhaite (VIS, NIR ou UV).
28/06/2016	Test de l'amélioration du software. Présentation de l'avancée du projet au responsable du projet. Commentaires sur le code.
29/06/2016	Commentaires codes MPLAB, optimisation des codes + tests. Commentaires code Python du DDS.
30/06/2016	Ajout de fonctionnalités à l'interface graphique de la PLL. Fonctionnalité permettant d'incrémenter automatiquement la fréquence toutes les 2 secondes. Recherches et documentation.
01/07/2016	Demande par le responsable du projet s'il est possible d'ajouter une fonctionnalité à l'interface : pouvoir entrer une fréquence de départ, une fréquence de sortie, un espacement de canal ainsi qu'un délai puis de parcourir la bande de fréquence entrée automatiquement.
04/07/2016	Pour ce faire : création d'un nouvel onglet dans l'interface graphique. Fin de l'implémentation de la fonctionnalité demandée. Test de l'ensemble de l'interface à l'oscilloscope + débogage.
05/07/2016	Débogage du code. Rédaction des commentaires du code Python en français.
06/07/2016	Rédaction des commentaires du code Python en anglais. Tutoriels sur LaTeX en vue d'écrire un manuel d'utilisation précis pour l'application écrite en Python.
07/07/2016	Commencer à écrire le manuel d'utilisation. Mise en lumière d'une fonctionnalité manquante dans l'application : pas de possibilité de choisir le port série sur lequel est branché le PIC. Si on veut changer de port, il faut aller dans le code source pour le changer manuellement. Donc ajout de la fonctionnalité à l'application.
08/07/2016	Test de l'application avec la nouvelle fonctionnalité : tests concluants. Continuer de rédiger le manuel d'utilisation. Ajout d'une nouvelle fonctionnalité à l'application : permettre à l'utilisateur de renseigner un nom de fichier pour charger ou sauvegarder les paramètres.
11/07/2016	Poursuite de l'écriture du manuel d'utilisation + correction de détails dans le code Python (expression régulières).
12/07/2016	Ecriture du manuel d'utilisation. Petites corrections apportées au programme.
13/07/2016	Fin de l'écriture du manuel d'utilisation. Création d'un logo pour l'application. Ajout du logo dans le code Python. Création d'un unique fichier pour l'application afin d'assurer sa portabilité sur n'importe quel ordinateur (fichier exécutable). Plusieurs problèmes rencontrés lors de la création de l'exécutable.
14/07/2016	Relecture de l'ensemble des documents rédigés (code commenté, manuel d'utilisation) afin de corriger d'éventuelles erreurs. Prise en main du logiciel ADIsimPLL permettant de concevoir une PLL complète en simulation.
15/07/2016	Début du tutoriel sur ADIsimPLL.
18/07/2016	Poursuite de la formation personnelle sur le logiciel ADIsimPLL. Fin du tutoriel.
19/07/2016	Design de la PLL utilisée pour le projet sur ADIsimPLL.
20/07/2016	Arrivée et présentation du PCB comportant la PLL, préparation du matériel pour les tests.
21/07/2016	Institut fermé (fête nationale belge).
22/07/2016	Institut fermé (pont).

25/07/2016	Essayer de programmer une interface graphique avec un autre outil : LabWindows/CVI. Formation sur CVI (tutoriel). Test sur le PCB comportant la PLL. Problème au niveau du montage : un régulateur de tension générant une tension de 3V3 a brûlé.
26/07/2016	Poursuite de la formation sur LabWindows/CVI. Test de la PLL après modifications apportées suite au problème précédent. Il semble il y avoir encore un problème sur le régulateur générant le 5V.
27/07/2016	Fin du tutoriel sur LabWindows/CVI. Début de la programmation de ma propre interface graphique pour contrôler la PLL : placement des différents éléments au sein de la fenêtre.
28/07/2016	Poursuite de l'écriture de l'interface graphique sur LabWindows/CVI. Test de la carte électronique : toujours un souci avec les régulateurs de tension, utilisation d'une caméra thermique afin de visualiser quels composants dissipent le plus de chaleur afin de voir d'où provient la chute de tension ayant lieu lorsque l'on branche l'alimentation.
29/07/2016	Interface graphique sur LabWindows : recherches pour parvenir à récupérer une liste des ports séries actifs du PC.
01/08/2016	Poursuite de la programmation de l'interface graphique sur LabWindows : algorithmes pour la vérification des valeurs entrées par un utilisateur dans les champs de texte.
02/08/2016	LabWindows. Modification d'un élément dans l'interface graphique Python de la PLL.
03/08/2016	Interface graphique sur LabWindows : recherches pour la gestion de fichiers textes avec LabWindows.
04/08/2016	Interface graphique sur LabWindows : récupération des différents paramètres de la fenêtre pour calculer les valeurs des registres à envoyer.
05/08/2016	Interface graphique sur LabWindows : difficulté pour calculer les valeurs des registres. Re conduite de tests à l'oscilloscope pour l'interface graphique Python de la PLL afin d'effectuer des captures d'écrans pouvant aller dans le rapport.
08/08/2016	Interface graphique sur LabWindows : recherches internet pour résoudre le problème de construction des registres.
09/08/2016	Interface graphique sur LabWindows : mise en application des informations obtenues lors de la phase de recherche.
10/08/2016	Poursuite de la programmation de l'interface graphique sur LabWindows : test et débogage de la solution trouvée.
11/08/2016	Interface graphique sur LabWindows : écriture de la fonction permettant un envoi de données via le port série. Amélioration de l'interface graphique Python. Mise à jour du manuel d'utilisation suite aux modifications.
12/08/2016	Test d'un envoi de données => visualisation des trames à l'oscilloscope (tous les registres sont envoyés). Poursuite de la programmation en vue d'implémenter le deuxième onglet de la fenêtre : placement des différents éléments de l'onglet.
15/08/2016	Jour férié.
16/08/2016	Interface graphique sur LabWindows : écriture d'algorithmes pour vérifier les valeurs entrées par l'utilisateur dans les champs de texte du deuxième onglet.
17/08/2016	Interface graphique sur LabWindows : écriture de la fonction principale qui permettra de calculer les valeurs des registres. Réalisation d'un logo pour l'application.
18/08/2016	Tests sur la carte électronique de la PLL. Peut-être encore un souci technique détecté.
19/08/2016	Débogage de la carte électronique.
22/08/2016	Ajout d'une fonctionnalité à l'interface graphique sur LabWindows/CVI : nouvel onglet ajouté à la fenêtre permettant à l'utilisateur de renseigner directement les valeurs hexa des registres.

23/08/2016	Poursuite de l'implémentation de la nouvelle fonctionnalité.
24/08/2016	Ajustement de l'interface graphique, plusieurs détails à corriger. Ecriture de petits documents ".txt" donnant des indications sur mon travail à une personne extérieure.
25/08/2016	Fin de l'implémentation de la fonctionnalité sur LabWindows et tests.
26/08/2016	Le PCB comportant la PLL a été débogué donc réalisation de plusieurs tests.
29/08/2016	Test et débogage de la carte électronique.
30/08/2016	Préparation des éléments à rendre à la fin de mon stage. Test final du PCB => test concluant, la boucle entière fonctionne correctement avec l'interface graphique.

ANNEXE 3 : Bibliographie

Présentation de l'institut :

<http://www.aeronomie.be/fr/>
https://fr.wikipedia.org/wiki/Institut_d%27a%C3%A9ronomie_spatiale_de_Belgique
https://www.belspo.be/belspo/fsi/iasbbira_fr.stm
<https://fr.wikipedia.org/wiki/A%C3%A9ronomie>
<http://50.aeronomie.be/index.php/fr/>
https://www.belspo.be/belspo/index_fr.stm
https://fr.wikipedia.org/wiki/Politique_scientifique_f%C3%A9d%C3%A9rale_belge

DDS :

<http://www.analog.com/library/analogDialogue/archives/38-08/dds.pdf>

PLL :

http://www.eetimes.com/document.asp?doc_id=1278888
<http://iosrjournals.org/iosr-jvlsi/papers/vol2-issue4/D0242125.pdf?id=2039>
http://users.polytech.unice.fr/~aliferis/fr/teaching/courses/elec4/tp_electronique/ep_unsa_elec4_tp_electronique_03_PLL.pdf
<http://www.analog.com/media/en/training-seminars/tutorials/MT-086.pdf>

AOTF :

<http://www.olympusmicro.com/primer/techniques/confocal/aotfintro.html>
http://www.brimrose.com/pdfandwordfiles/tunable_filter.pdf

Python :

<http://www.courspython.com/composants.html>
<http://pyqt.sourceforge.net/Docs/PyQt4/qwidget.html>
<https://openclassrooms.com/forum/sujet/creation-interface-graphique-avec-pyqt>
<http://python.developpez.com/cours/TutoSwinnen/?page=Chapitre8#L8.1>

LaTeX :

<https://www.tug.org/twg/mactex/tutorials/ltxprimer-1.0.pdf>
http://www.hofler.com/pdf/User_Manual_English.pdf
https://fr.sharelatex.com/learn/Text_alignment
https://fr.wikibooks.org/wiki/LaTeX/Mise_en_forme_du_texte#Choix_du_style
<http://www.personal.ceu.hu/tex/breaking.htm>
<http://wwwis.win.tue.nl/2R690/projects/spingrid/sum.pdf>

LabWindows/CVI :

<http://www.ni.com/lwcvif/>
<https://www.ni.com/support/cvi/serlprog.htm>
<http://www.ni.com/lwcvif/technical-resources/f/>

ANNEXE 4 : Glossaire des définitions et acronymes

Terme	Définition
ALTIUS	<i>Atmospheric Limb Tracker for the Investigation of the Upcoming Stratosphere</i> . Projet belge d'analyse du limbe atmosphérique.
AOTF	<i>Acousto-Optic Tunable Filter</i> (filtre acousto-optique).
BASCOE	<i>Belgian Assimilation System for Chemical Observations</i> . Analyse belge de la teneur en gaz de la stratosphère.
BELSPO	<i>Belgian Science Policy</i> (politique scientifique fédérale belge).
Biréfringence	Différence entre les indices de réfractions minimale et maximale qu'il est possible d'obtenir dans un cristal biréfringent.
<i>Bitfields</i>	Terme utilisé en programmation informatique pour le stockage de plusieurs bits consécutifs. Cela se présente généralement sous la forme d'une structure dans laquelle sont définis plusieurs éléments en indiquant le nombre de bits qu'ils doivent occuper.
BRAMS	<i>Belgian Radio Meteor Stations</i> . Projet belge de détection radio de météores.
<i>Buffer</i>	Zone de mémoire pour le stockage temporaire de données.
C ANSI	Un des standards du langage de programmation C.
Ceinture de radiation	Regroupement de particules autour de la Terre dû au champ magnétique terrestre.
CNA	Convertisseur Numérique Analogique.
Codage ASCII	<i>American Standard Code for Information Interchange</i> . Standard informatique attribuant un code numérique à chaque caractère.
<i>Combo box</i>	Composant d'interface graphique réunissant une zone de texte et une liste déroulante.
COMESSEP Alert System	Site internet constituant un système d'alerte d'événements spatiaux.
Cristal biréfringent	Matériau dont les propriétés physiques dépendent de la direction de l'onde lumineuse incidente.
DDS	<i>Direct Digital Synthesizer</i> (synthétiseur de fréquence numérique direct).
<i>Docklight</i>	Outil de simulation logiciel pour des protocoles de communication série.
EPT	<i>Energetic Particle Telescope</i> . Projet de l'IASB visant à concevoir un instrument mesurant l'énergie de flux de particules.
ESA	<i>European Space Agency</i> (agence spatiale européenne).
FPGA	<i>Field Programmable Gate Array</i> (circuit logique programmable).
IASB	Institut d'Aéronomie Spatiale de Belgique.
IDLE	Environnement de développement intégré pour le langage informatique Python.
IMPCVOC	<i>Impact of Phenology and Environmental Conditions on biogenic Volatile Organic Compounds emissions from forest ecosystems</i> . Etude de l'IASB sur l'influence des plantes sur la qualité de l'air par la mesure des flux de COVB (Composés Organiques Volatiles Biogéniques).
Ionosphère	Couche de la haute atmosphère d'une planète.
IRM	Institut Royal Météorologique de Belgique.
LCD Number	Composant d'interface graphique. Affichage de donnée sous la forme d'un petit écran LCD.

LiDAR	<i>Light Detection and Ranging</i> . Analyse des propriétés physiques d'un faisceau traversant une surface cible.
Limbe atmosphérique	Enveloppe extérieure de l'atmosphère éclairée par le soleil et visible depuis l'espace.
Line edit	Composant d'interface graphique. Zone de texte.
MACC	<i>Monitoring Atmospheric Composition and Climate</i> . Projet multinational ayant pour but de présenter la composition de l'atmosphère terrestre.
Marge de phase	Ecart entre -180° et la valeur de la phase pour une pulsation correspondant à un gain de 0 dB. La marge de phase est utilisée pour l'étude de la stabilité d'un système asservi.
NASA	<i>National Aeronautics and Space Administration</i> (administration nationale de l'aéronautique et de l'espace).
NIR	<i>Near-infrared</i> (proche infrarouge, 800 – 2500 nm). L'infrarouge est un rayonnement électromagnétique invisible à l'œil nu car il correspond à des longueurs d'ondes supérieures à celles composant le spectre de la lumière visible. Le proche infrarouge est une des subdivisions de l'infrarouge se situant près du spectre de lumière visible.
NOMAD	<i>Nadir and Occultation for Mars Discovery</i> . Spectromètre destiné à analyser l'atmosphère de la planète Mars.
PCB	<i>Printed Circuit Board</i> (circuit imprimé).
Piézoélectricité	Capacité de certains corps à se polariser électriquement en réponse à une contrainte mécanique ou inversement.
PLL	<i>Phase Locked Loop</i> (boucle à verrouillage de phase).
PLL Controller	Nom de l'application permettant de contrôler la PLL à partir d'un ordinateur.
Port série	Port présent sur un ordinateur permettant d'établir une communication série avec un autre appareil. L'évolution du port série est le port USB.
PROBA	<i>Project for On-Board Autonomy</i> . Série de microsatellites de conception belge à faible coût utilisés par l'ESA.
PyQt	Module logiciel faisant le lien entre le langage Python et la bibliothèque Qt.
Python	Langage de programmation multiplateformes orienté objet.
Qt	Bibliothèque regroupant des fonctions utilisées dans la conception d'interfaces graphiques.
RF	Radiofréquence.
SACS	<i>Support to Aviation Control Service</i> . Outil de support à l'aviation indiquant les rejets de poussières dans l'atmosphère.
SOIR	<i>Solar Occultation in the Infrared</i> . Projet de l'IASB mettant en jeu un spectromètre à haute résolution fonctionnant dans la gamme infrarouge.
SPENVIS	Site internet géré par l'IASB présentant des informations relatives à la météo spatiale.
SPI	<i>Serial Peripheral Interface</i> : bus de communication série synchrone fonctionnant sur le principe d'une hiérarchie « maître – esclave ».
Spin box	Composant d'interface graphique. Zone de sélection numérique avec incrément/décroissement.
Stratosphère	Couche de l'atmosphère siège des principaux phénomènes météorologiques.
USB	<i>Universal Serial Bus</i> . Protocole informatique de communication série.

UV	Ultraviolet. Rayonnement électromagnétique invisible à l'œil nu car correspond à des longueurs d'ondes situées en-dessous du spectre de lumière visible (10 – 390 nm).
VCO	<i>Voltage Controlled Oscillator</i> (oscillateur contrôlé en tension).
VHF	<i>Very High Frequency</i> (très hautes fréquences). Bande de fréquences comprise entre 30 et 300 MHz.
VIS	Lumière visible. Rayonnement électromagnétique visible à l'œil nu humain (390 – 750 nm).
Widget	Composant d'interface graphique.

ANNEXE 5 : Table des illustrations

Figure 1 : organigramme du BELSPO	7
Figure 2 : organigramme de l'IASB	8
Figure 3 : laboratoire d'électronique de l'institut	9
Figure 4 : atelier de mécanique de l'institut	9
Figure 5 : salle propre avec chambre à vide thermique	10
Figure 6 : photo du spectromètre SOIR.....	10
Figure 7 : photo du télescope EPT.....	11
Figure 8 : schéma du spectromètre NOMAD de la mission spatiale ExoMars	12
Figure 9 : photo du réceptacle emprisonnant une branche d'arbre	12
Figure 10 : photo d'une des antennes de réception du projet BRAMS	13
Figure 11 : capture d'écran d'un relevé de l'outil BASCOE indiquant	14
Figure 12 : capture d'écran du site internet renseignant sur l'indice UV en Belgique.....	14
Figure 13 : photo montrant le limbe atmosphérique terrestre	15
Figure 14 : explication schématisée de la technique d'occultation solaire	17
Figure 15 : explication schématisée de la technique de diffusion au limbe.....	17
Figure 16 : schéma explicatif du principe de fonctionnement d'ALTIUS	18
Figure 17 : exemple de spectre de raies d'absorption	19
Figure 18 : schéma du principe de fonctionnement d'un AOTF	19
Figure 19 : caméra spectrale pour le domaine de la lumière visible	20
Figure 20 : vue schématisée de la mise en commun des trois caméras spectrales.....	21
Figure 21 : vue extérieure du satellite utilisé pour la mission ALTIUS	21
Figure 22 : vue schématisée simplifiée de la chaîne RF	22
Figure 23 : photo de l'AD9910 evaluation board.....	24
Figure 24 : photo de la carte électronique de l'IASB constituant une PLL complète	24
Figure 25 : photo du PIC18F46J50 FS USB PIM DEMO BOARD	24
Figure 26 : schéma blocs du circuit interne d'un DDS	26
Figure 27 : explication du fonctionnement de l'accumulateur de phase.....	26
Figure 28 : explication du fonctionnement du convertisseur phase – amplitude	27
Figure 29 : représentation graphique d'un signal sinusoïdal échantillonné (à droite)	27
Figure 30 : schéma blocs de l'AD9910 evaluation board	28
Figure 31 : tests de la liaison USB avec le logiciel Docklight.....	29
Figure 32 : chronogrammes des signaux impliqués dans la communication SPI de l'AD9910.....	30
Figure 33 : test basique de la communication SPI à l'oscilloscope	31
Figure 34 : réception de données sur le port USB et envoi sur le bus SPI	32
Figure 35 : interface graphique permettant de contrôler le DDS.....	33
Figure 36 : connexion des signaux émis par les "widgets" avec Python	34
Figure 37 : fonction permettant d'envoyer les commandes au DDS	34
Figure 38 : vue de l'interface graphique lors de la génération d'un signal.....	35
Figure 39 : spectre du signal de sortie du DDS pour une amplitude minimale	36
Figure 40 : spectre du signal de sortie du DDS pour une amplitude maximale	37
Figure 41 : acquisition de l'oscilloscope pour la comparaison des signaux de sortie du DDS à deux amplitudes différentes	38
Figure 42 : schéma blocs de l'architecture interne basique d'une PLL	39
Figure 43 : schéma blocs d'une PLL avec un diviseur de fréquence par N.....	39
Figure 44 : schéma d'une pompe de charge usuelle	40
Figure 45 : exemple de caractéristique de transfert d'un VCO	42
Figure 46 : schéma de l'architecture du diviseur de fréquence par N en utilisant un pré-diviseur double-modulo.....	43
Figure 47 : tableau présentant les relations de fréquence dans le système utilisant une PLL	44
Figure 48 : schéma fonctionnel du montage à base de PLL réalisé par l'IASB	45
Figure 49 : schéma blocs de la PLL ADF4108 de Analog Devices	45
Figure 50 : chronogrammes des signaux impliqués dans la communication SPI de l'ADF4108	46

Figure 51 : fonction d'initialisation de la communication SPI pour l'ADF4108	47
Figure 52 : construction interne du registre d'initialisation de l'ADF4108	47
Figure 53 : utilisation de bitfields pour la déclaration d'un registre.....	48
Figure 54 : attribution d'une valeur pour chaque bitfields du registre.....	49
Figure 55 : fonction permettant d'adresser un des registres de la PLL via la liaison SPI	50
Figure 56 : visualisation des trames binaires de la liaison SPI à l'oscilloscope.....	50
Figure 57 : instruction "if" permettant de recevoir des données puis de les envoyer à la PLL.....	52
Figure 58 : interface graphique permettant de contrôler la PLL (premier onglet)	53
Figure 59 : interface graphique permettant de contrôler la PLL (deuxième onglet).....	53
Figure 60 : description des paramètres présents dans l'onglet "Main settings" de l'application	54
Figure 61 : explication des opérations pouvant être effectuées dans l'onglet "Main settings"	55
Figure 62 : paramètres de l'onglet "Main settings" associés à un champ vide	56
Figure 63 : description des paramètres présents dans l'onglet "Sweep" de l'application	57
Figure 64 : message d'erreur renvoyé par l'application lorsque la valeur de la "Reference Frequency" n'est pas un nombre sur deux ou trois chiffres	58
Figure 65 : message d'erreur renvoyé par l'application lorsque la valeur de la "Reference Frequency" n'est pas incluse dans l'intervalle fixé	59
Figure 66 : message renvoyé par l'application après le succès d'une opération de sauvegarde	59
Figure 67 : visualisation d'une trame binaire à l'oscilloscope.....	60
Figure 68 : valeurs des registres affichées dans l'application	60
Figure 69 : sommaire du manuel d'utilisation écrit avec LaTeX.....	62
Figure 70 : graphiques obtenus après simulation de la PLL sur ADIsimPLL.....	63
Figure 71 : exemple d'une fonction de rappel générée par LabWindows/CVI.....	64
Figure 72 : interface graphique réalisée sur LabWindows/CVI	65