RAPPORT DE PROJET ATISE

Aurora Thermosphere Ionosphere Spectrometer Experiment

Adrien ARTAUD

Myriam LOMBARD

Killian PAREILLEUX

Alexandre SALMON

2020/2021

TABLE DES MATIÈRES :

REMERCIEMENTS

Nous tenons à remercier Monsieur Frédéric Martin, notre responsable de projet du CSUG, qui nous a accompagnés dans la réalisation et a répondu à nos questions avec efficacité. Nous remercions aussi les enseignants de Polytech Grenoble pour nous avoir proposé un tel projet et nous avoir donné les compétences pour le réaliser. Nous tenons également à remercier le FabMSTIC pour leurs prêts de matériel.

INTRODUCTION

Le projet Aurora Thermosphere Ionosphere Spectrometer Experiment, ou ATISE, est un projet de nanosatellite lancé par le Centre Spatial Universitaire de Grenoble (CSUG) en 2015. Son objectif est d'étudier les aurores boréales depuis l'espace afin d'obtenir une meilleure compréhension de la magnétosphère et de l'activité solaire. Ce satellite, de très petite taille (format 12U : seulement 30cm×20cm×20cm) est en très grande partie réalisé par des étudiants de plusieurs pays, en moyenne 120 par an. Par exemple, le CSUT, Centre Universitaire de Toulouse, est chargé de réaliser la plateforme de ce nanosatellite tandis que le CSUG a pour mission de développer la partie instrumentale (appelée Payload sur le schéma ci-contre).

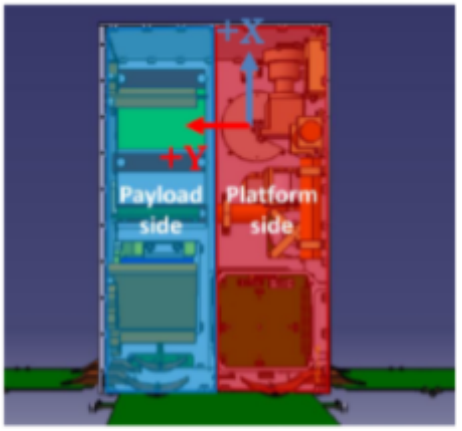


Figure 1 : Architecture du nanosatellite ATISE - Source

ATISE se sépare en deux parties distinctes : la partie Plateforme et la partie Payload. La première contient l'ordinateur principal du nanosatellite et un émetteur hautes fréquences. La seconde comporte plusieurs capteurs qui récupéreront des données sur les aurores boréales puis les enverront ensuite sur Terre afin qu'elles soient exploitées par des chercheurs. Parmi ces capteurs, on retrouve un appareil photo Onyx et 3 spectrographes HDPyx. Ces deux instruments permettent respectivement d'obtenir des photographies d'excellente qualité et des spectres des aurores boréales depuis l'espace. La miniaturisation de ce matériel scientifique de pointe a permis de construire de petits satellites comme celui-ci. Le poids d'un nanosatellite se situe en règle générale entre 1 et 30 kg, ce qui le rend moins cher, moins gourmand en énergie et bien plus abordable qu'un satellite "classique". Les coûts de production et de lancement d'un Cubesat sont moins élevés que ceux de satellites "classiques", ce qui permet à des étudiants d'utiliser cette technologie et d'ouvrir l'espace à un public plus large. ATISE a connu de nombreuses avancées, cependant le transfert de données entre la partie Instrumentation et la partie Plateforme du satellite n'a pas encore été implémenté.

OBJECTIFS ET CONTRAINTES

L'objectif de notre projet est de gérer la communication entre la partie capteur et la partie communication du satellite. Pour ce faire, nous disposons d'une carte Mars ZX3 qui est similaire à celle employée sur le nanosatellite. Nous devons réussir à faire passer un fichier (image, tableau de bits...) à travers l'UART de la carte, en utilisant le Cubesat Space Protocol (voir "Technologies employées").

Ce projet doit de plus être réalisé sous FreeRTOS, un système d'exploitation temps réel, qui assure que chaque instruction est exécutée dans un temps maximum fixé. Plus d'informations à son sujet sont présentées dans la partie Technologies employées. Enfin, notre projet doit implémenter le Cubesat Space Protocol, un protocole de livraison de couches de réseau conçu pour les nanosatellites tels que ATISE.

Jalons

Étant donné que le cadre de ce projet est assez clairement défini, nous n'avons pas réalisé de cahier des charges à proprement parler. Cependant, nous avons tout de même défini des jalons pour notre avancement :

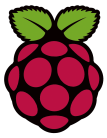
- Adapter le code Legacy C en FreeRTOS (**Sprint 1**)
- Échanger des données avec la carte Mars XZ3 via l'UART (**Sprint 2**)
    - Transfert d'un seul bit
    - Transfert d'une structure de données (tableau de bits, etc.)
- Utiliser le CSP avec l'UART (**Sprint 3**)

S'il y a le temps :

- Optimiser le débit
- Intégrer le traitement d'images

TECHNOLOGIES EMPLOYÉES

FreeRTOS est un système d'exploitation en temps réel (Real Time Operating System) pour micro-contrôleurs. Il est robuste, de taille faible et Open Source, c'est-à-dire que son code source est accessible à tout public. Ces qualités ont mené à ce que ce système d'exploitation soit principalement utilisé sur les systèmes embarqués car ils possèdent des contraintes d'espace et de code. FreeRTOS tourne sur la carte Mars ZX3, nous avons donc adapté notre code pour qu'il fonctionne sur ce système d'exploitation.



L'**UART,** Universal Asynchronous Receiver Transmitter, est un circuit physique dans un micro-contrôleur dont la fonctionnalité est de recevoir et de transmettre des données en série. Pour nous familiariser avec l'UART, nous avons commencé par effectuer des tests de communication entre deux RaspBerry PI. Nous avons tout d'abord transmis quelques bits, puis les avons placés dans un fichier dès leur réception.

Git est un outil de gestion de versions que nous utilisons pour gérer notre code et nous le partager. Nous travaillons sur la plateforme GitHub qui utilise Git. Cette plateforme nous offre également la possibilité de créer des issues et de les visualiser dans un Kanban. Dans notre projet, nous avons créé trois répertoires différents : Docs, Sandbox et Sources. Dans le répertoires Docs se trouvent toutes les documentations relatives aux technologies utilisées ainsi qu'au contexte du projet. Le répertoire Sandbox contient quant à lui nos essais et nos progrès sur la programmation de la carte Mars ZX3. Le code final, celui qui fonctionne et qui correspond aux attentes du projet, se situera dans le répertoire Sources.

Vivado Design Suite est une suite logicielle principalement composée d'un IDE permettant de concevoir des systèmes logiques électroniques. Vivado permet d'injecter directement du code C dans des appareils contenant la technologie de son producteur, Xilinx. Cette suite contient également un Software Development Kit (SDK) qui est l'environnement de conception intégré pour la création d'applications embarquées sur un microprocesseur. Le SDK inclut des pilot
par l'utilisateur ainsi que des biblioth
réseau et de fichiers.

Eclipse IDE est un environnement de développement open-source supportant une grande variété de langages, similaire à Visual Studio Code. Il s'agit d'un standard pour développer programmes et logiciels. Nous l'utilisons avec le langage C.

Le **CubeSat Space Protocol** est un protocole écrit en langage C. Son objectif est de simplifier la communication entre les systèmes embarqués au sein d'un petit réseau, comme un CubeSat. La conception suit un modèle TCP/IP et inclut un protocole de transport, un protocole de routage et plusieurs interfaces de couche MAC. L'idée est de donner aux développeurs de Cubesat les mêmes fonctionnalités qu'une pile TCP/IP, mais sans ajouter l'énorme coût du header IP. Dans le projet ATISE, le CSP est utilisé dans le but de transférer des données entre la partie Payload et la partie Plateforme du nanosatellite.

ARCHITECTURES TECHNIQUES

Nous allons à présent vous présenter l'architecture interne de la partie Payload du satellite ATISE. Dans le schéma suivant, le Frame Grabber représente les outils qui collectent des informations comme l'appareil photo et le spectromètre. Le VDMA, soit Video Direct Access Memory, donne la possibilité à un périphérique d'entrées/sorties d'envoyer ou recevoir directement des données vers ou depuis la mémoire principale pour accélérer les opérations de mémoire.

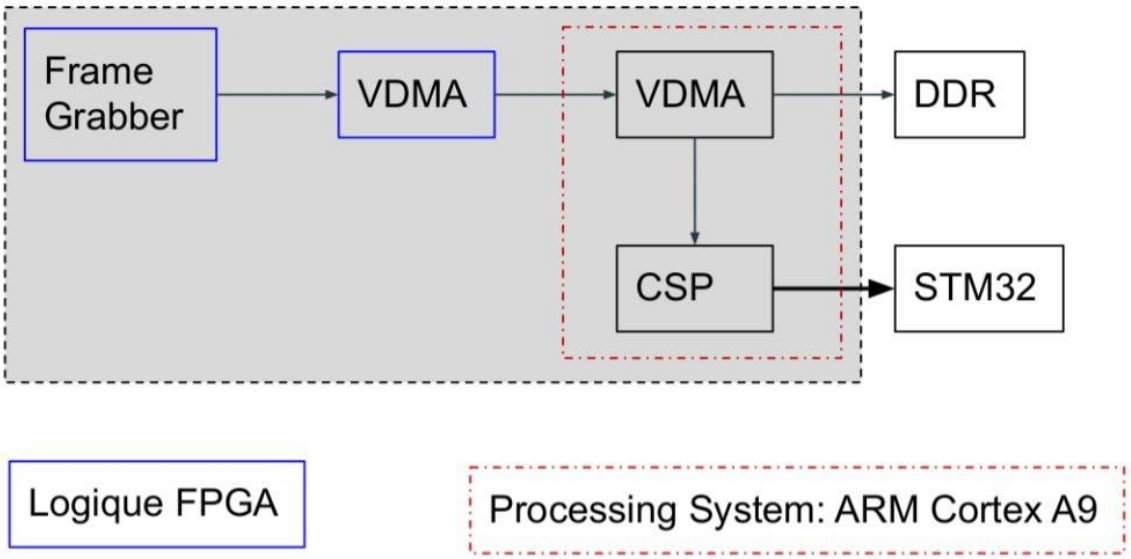


Figure 2 : Architecture interne de la partie Payload du nanosatellite ATISE - Schéma réalisé par nos soins

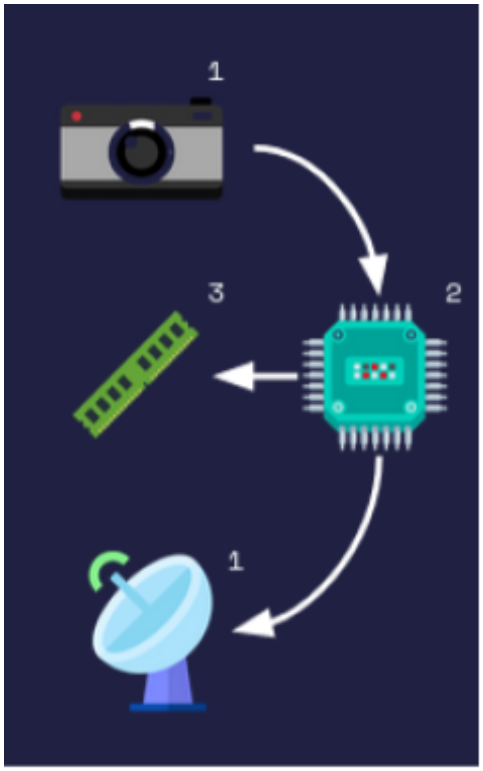


Figure 3 : Composants principaux du nanosatellite ATISE - Schéma réalisé par nos soins

1) HDPyx

Lorsque le nanosatellite sera dans l'espace, il inclura un spectromètre afin de mesurer les aurores boréales et la lumière du ciel nocturne dans la gamme spectrale. Actuellement, nous avons un composant HDPyx qui simule un spectromètre afin de nous donner des exemples de vraies données à transmettre.

2) Micro-contrôleur

Le microcontrôleur est un ARM Cortex A9. Son rôle est de récupérer les données du HDPyx, de les copier dans la mémoire interne et de les envoyer à la partie plateforme du nanosatellite. Celle-ci se chargera de les envoyer à la Terre.

3) Partie Plateforme

La partie plateforme contient le système du satellite : le guidage et la communication. La partie payload va communiquer avec elle via l'UART grâce au CSP.

RÉALISATIONS TECHNIQUES

Nous avons mené 2 tâches principales en parallèle :

- Gérer la communication avec la carte MARS ZX3

D'une part, nous avons reçu au début un code Legacy C ainsi qu'une carte MARS ZX3, nous avons donc dû nous familiariser avec les différents outils mis à notre disposition. Nous avons commencé par comprendre et déchiffrer ce que faisait chaque ligne du code Legacy C. Une fois cette tâche réalisée, nous nous sommes documentés sur FreeRTOS afin d'adapter ce code Legacy à ce système d'exploitation. Nous avons réalisé ces tests sur la carte MARS ZX3 à l'aide du SDK fourni par le logiciel Vivado. Notre porteur de projet avait déjà réalisé l'architecture hardware de la carte, nous n'avions plus qu'à faire fonctionner la partie software correspondante.

Après avoir échangé des chaînes de caractères avec la carte MARS ZX3, nous nous sommes attaqués aux fichiers format PGM. Un fichier .pgm est soit un fichier texte (ASCII), soit un fichier binaire avec une définition d'une image bitmap 8 bits en niveaux de gris. Le moyen de distinction entre ces deux formes est le premier caractère du header du fichier .pgm : P2 pour un fichier ASCII et P5 pour un fichier binaire. Un fichier au format PGM contient la structure suivante :

  - Nombre magique : **Signature** indiquant la forme du fichier (P2 ou P5)
  - Nombre de **colonnes** puis de **lignes** (séparées par un espace)
  - **Intervalle** utilisé pour coder chacune des nuances de gris (appelé N)
  - Chaque pixel est ensuite **codé** à l'aide de deux entier (entre 0 et N)

- Comprendre comment faire fonctionner l'UART et le CSP ensemble, puis le tester en conditions réelles

D'autre part, nous devions nous familiariser avec l'UART et le CSP. Notre professeur nous a conseillé de tester l'UART entre deux Raspberrys Pis que nous avons été récupérer au FabLab. Les Raspberrys Pis étant sous Linux, il était plus pratique de commencer à tester le fonctionnement en UART avec eux. Nous avons ensuite essayé d'échanger plusieurs types de données entre nos Raspberry Pis : des chaînes de caractères, des bits, et enfin des structures de données. Tout d'abord, nous affichions seulement ces informations sur un terminal, puis nous les avons stockées directement dans un fichier.

La librairie libcsp implémente le CSP, nous nous sommes donc documentés sur celle-ci. Nous avons lancé des tests, par exemple dans l'un d'eux, le CSP permet à 2 threads, client et serveur, de communiquer entre eux (le client envoie une requête au serveur). Côté serveur, il faut créer une socket, la lier à un port, créer une file d'attente, et enfin se mettre en attente des nouvelles connexions. Côté client, la procédure de communication est plus complexe :

1. Connexion à un hôte sur une adresse donnée en paramètre
2. Récupérer le tampon du paquet pour le message
3. Copier les données à transmettre dans le paquet
4. Définir la longueur du paquet (ne pas oublier de compter le symbole de terminaison)
5. Envoi du paquet
6. Fermeture de connexion

AXES D'AMÉLIORATION

Concernant la gestion de l'UART de la carte électronique, il serait appréciable de corriger l'instabilité du transfert d'images par ce biais. En effet, sur un groupe de 50 octets, nous en transmettons actuellement 49. Cette perte peut paraître minime, mais elle représente environ 2% des éléments de l'image qui manquent à l'appel.

Rajouter le protocole CSP au code de la carte électronique est une tâche qui devrait aussi être faite dans le futur du projet ATISE. Nous avons testé le protocole séparément, mais au vu de nos résultats de transfert d'images non complètes, nous n'avons pas encore implémenté le CSP sur la carte. Par ailleurs, la librairie libcsp n'est pas beaucoup utilisée et nous avons mis du temps à trouver les paquets nécessaires pour utiliser cette librairie, ces derniers n'étant pas précisés dans la documentation de la librairie CSP.

Concernant le débit de transfert d'images, il serait judicieux de vérifier et réguler ce débit afin d'arriver à envoyer plusieurs Mo en moins de 2 secondes. Le but du projet ATISE étant d'envoyer à la Terre toutes les 2 secondes une image d'1 Mo et un graphique un peu plus léger, il serait bien d'arriver à obtenir un débit supérieur ou équivalent à 1 Mo/s.

Nous n'avons pas pu faire le lien avec le traitement de l'image car nous n'avons pas reçu cette composante de l'équipe de Montpellier. Nous avons cependant préparé le code pour accepter un tel ajout.

D'autre part, la carte électronique que l'on utilise actuellement est bridée à 115200 Hz, ce qui est une fréquence assez faible. Une possibilité serait de modifier l'UART qui est utilisée pour le projet afin d'améliorer cette fréquence. Cela rendrait les tests plus courts dans le temps.

GESTION DE PROJET

Nous avons réparti les rôles de notre groupe de la manière suivante : Killian était le Scrum Master, Alexandre le Product Owner, et Myriam et Adrien étaient des développeurs simples. Il n'y avait pas besoin d'un responsable Git car chaque membre était également responsable de bien gérer le Github et de le mettre à jour régulièrement.

Pour ce projet, nous avons essayé d'appliquer la méthode Agile, tout en essayant de l'adapter à nos conditions de travail quelque peu mises à mal par le travail en distanciel. Nous avons donc au cours du projet fait des "Quasi Daily Meetings", c'est-à-dire des meetings le plus souvent possible selon les disponibilités de chacun. Adrien étudiant à l'IAE, il ne pouvait par exemple pas participer les vendredi. Nous avons décidé de plus de nous rencontrer deux fois par semaine en présentiel à Polytech, en général le mercredi et le jeudi, afin de pouvoir travailler plus efficacement à quatre sur les programmes en cours, ne pouvant être testés que sur la Mars ZX3.

Notre organisation s'est basée sur les outils proposés par Github : nous avons établi des issues sur les différents aspects du projet, que nous avons ensuite placé dans un Kanban en attribuant chaque issue à un développeur pour que chacun aie quelque chose à faire. Nous avons aussi séparé notre travail Git en trois repositories, un repo "Doc" avec la documentation et quelques aides à l'installation de certains outils, un repo "Sources" avec notre code testé, et un repo "Sandbox" avec notre code expérimental.

Planning prévisionnel

1. Adapter le code en FreeRTOS (Sprint 1 - du 08/02 au 21/02)
2. Échanger des données avec la carte via l'UART (Sprint 2 - du 22/02 au 07/03)
    a. Transfert d'un seul bit
    b. Transfert d'un tableau de bits (structure de données)
3. Utiliser le CSP avec l'UART (Sprint 3 - du 08/03 au 19/03)
4. Optimiser le débit (Si il y a le temps)
5. Prendre en compte le traitement d'images (Si il y a le temps)

Planning effectif

Sprint 1
Semaine du 25/01

1. 28/02: Début du projet
2. 29/02: rendez-vous avec Frédéric MARTIN
   Présentation du projet et de son contexte
   Présentation des outils
   Prêt de la carte Mars ZX3
3. 29/02: Emprunt des cartes STM32 au FabLab

Semaine du 01/02

- Mise en place des outils
  - Serveur Discord
  - Projet Github
  - Installation et prise en main de Vivado

Semaine du 08/02

- 10/02: Rendez-vous en présentiel à Polytech
- Découpage du projet en sprints et en tâches
- Répartition des tâches
- Début de l'étude des technologies UART, CSP et FreeRTOS
- Début de l'étude du code C fourni

Semaine du 15/02
Interruption pédagogique

- Étude des technologies UART, CSP et FreeRTOS
- Étude du code C fourni
- Production de compte-rendus

Sprint 2
Semaine du 22/02

- 24/02: rendez-vous en présentiel à Polytech
- 25/02: rendez-vous en présentiel à Polytech
  - Préparation de la soutenance de mi-parcours
- 26/02: Soutenance de mi-parcours
- Préparation du poster

Semaine du 01/03

- Portage vers FreeRTOS
- Emprunt de deux Raspberry Pis au FabLab
- 03/03: Rendez-vous avec Frédéric MARTIN
  - Explications pour le debug UART sur la carte MARS
- 04/03: Correction du poster
- Mise en place de la communication UART entre Raspberry Pis
  - Par simple utilisation du terminal
  - En C
  - Blocage par un bug empêchant le transfert de plus de 3.7ko de données (non résolu)

Sprint 3
Semaine du 08/03

- 09/02: Rendez-vous en présentiel à Polytech
- 11/02: Rendez-vous en présentiel à Polytech
- Ajout UART
    - Problème corruption données et perte paquets
    - Problème pour les transfert de plus de 6 Mo
- Test CSP sous linux
- Début de la rédaction du compte rendu

Semaine du 15/03

- 16/03: Rendez-vous avec Frédéric Martin
    - Ajout contrainte d'utilisation de la carte SD
- 17/03: Rendez-vous en présentiel à Polytech
- 19/03: Soutenance finale

Gestion des risques

La principale difficulté du projet résidait dans le manque de matériel : en effet, nous n'avions qu'une seule carte électronique. Nous avons donc dû organiser notre travail autour du fait que notre carte soit disponible en un unique exemplaire, et cela pour que ce soit un facteur le moins limitant possible. Nous avons donc mis en place deux solutions pour gérer ce problème. La première consiste à effectuer d'autres tâches utiles au projet telles que des recherches de documentations requises pour les étapes suivantes. Le but étant de rendre l'avancement de la personne chargée de la carte le plus fluide possible, tout en permettant à chacun de travailler sur le projet en parallèle. La seconde solution a été de mettre en place du pair programming pour s'occuper de la carte MARS ZX3. Ainsi, deux personnes pouvaient travailler en même temps sur la carte, l'un codant et l'autre relisant, conseillant et questionnant les choix effectués par son partenaire. La seconde personne est tout aussi active que celle qui code, son rôle est de corriger le code si nécessaire et de d'offrir des propositions d'amélioration le cas échéant. Nous avons ainsi pu mettre à profit les séances de travail où le critère de la carte unique était trop limitant pour impliquer un maximum de personnes dans le projet.

Le second risque lié à ce projet est le manque de communication entre les différents membres du groupe. En effet, nous sommes principalement à distance, et il est parfois compliqué d'échanger correctement à travers nos écrans. Nous avons beau nous partager le code via notre GitHub, cela reste tout de même plus agréable lorsque quelqu'un nous explique ce qu'il a compris ou la meilleure manière d'installer tel ou tel outil de vive voix que via une fiche de documentation.

OUTILS

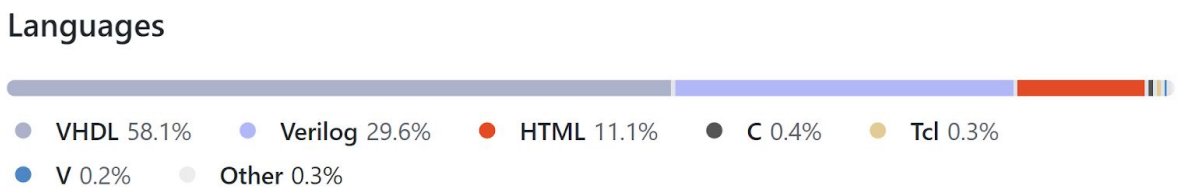
Afin de pouvoir collaborer efficacement au projet en distanciel, nous avons utilisé plusieurs outils collaboratifs à notre disposition. Nos réunions, discussions et séances de travail en ligne se faisaient sur un serveur Discord dédié, où nous partagions aussi quelques ressources en cas de besoin immédiat.

Pour travailler sur des fichiers en même temps et les stocker de manière simple, nous avons aussi utilisé Google Drive et Google Docs pour réaliser nos rapports, présentations ainsi que certaines documentations.

Enfin, comme expliqué précédemment, nous utilisions Github comme gestionnaire de versions et de projets d'équipe.

MÉTRIQUES LOGICIELS

Langages



Notre réalisation est uniquement en langage C, ce qui est un pourcentage très faible du code total, seulement 0.4%. Cela s'explique par le fait que le code contient beaucoup d'autres composantes, notamment la logique complète de la carte.

Performances

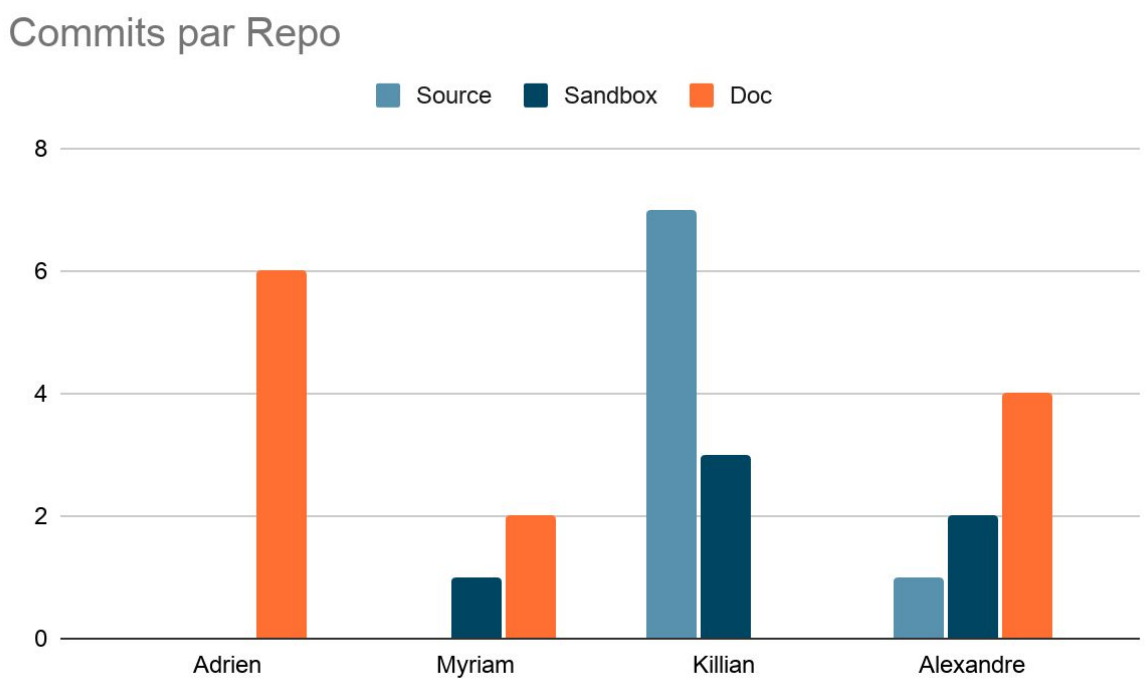
Actuellement, notre implémentation permet un débit de transmission de 14,4 kO/s. Cette valeur peut encore être améliorée et nous avons vu qu'une amélioration était possible en ajoutant une nouvelle communication UART sur la carte physique qui ne soit pas reliée au composant jtag limitant. Nous avons décidé de ne pas modifier la carte car nous avions d'autres parties du projet à réaliser, mais nous avons adapté le code pour que cette modification soit facilement réalisable par une autre équipe ou par monsieur Martin.

Temps ingénieur

Durant ce projet, nous avons travaillé en moyenne 4 jours par semaine, 7 heures par jour. Ce qui fait donc sur 8 semaines, 224 heures de travail par personne. Il s'agit d'une moyenne car nous travaillions parfois plus de 4 jours et parfois moins, selon les cours, examens et disponibilités de chacun.

Nous avons de plus, à l'aide d'une feuille de calcul proposée par le cours de MPI, évalué le budget prévisionnel du projet (voir annexes), qui s'élève à environ 12000€.

Répartition code et commits



Comme le montre ce graphique, chacun a contribué au projet d'une manière unique : Adrien s'est occupé de la documentation de l'UART, de la rédaction de rapports, de posters, Myriam a aidé les autres membres du projet dans différents secteurs et a étudié CSP, Killian qui avait la carte a commité la majeure partie du code et a effectué des tests sur sandbox. Alexandre a travaillé sur les Raspberry Pi et a donc fait des tests et de la documentation en conséquence.

CONCLUSION

Ce projet fut réalisé dans des conditions très particulières, mais fut pour nous très formateur. Nous avons dû pallier de nombreuses difficultés, entre les impossibilités de se voir, de travailler à plusieurs, ou encore les bugs et erreurs inexplicables. Nous sommes tout de même satisfaits de comment nous avons géré le projet, en essayant de maximiser les rencontres en présentiel et en essayant de toujours donner une chose à faire à chacun.

Nous sommes de plus très heureux d'avoir pu être le maillon d'une chaîne de plus grande envergure, et de savoir que notre contribution aidera de futures équipes à lancer le nanosatellite ATISE dans l'espace pour de bon. Le fait de faire partie d'un projet déjà existant nous a aussi donné un travail d'apprentissage de technologies déjà implémentées complexes, ce qui nous sera très utile pour notre de carrière d'ingénieur, où nous serons souvent amenés à complémenter des systèmes existants.

Il reste encore du travail pour améliorer la communication interne d'ATISE, mais nous avons déjà réussi à traduire le code C existant selon les contraintes demandées. Dans un contexte plus favorable au travail de groupe, nous aurions sans doute pu finaliser la stabilité des messages et rajouter CSP à notre implémentation. Nous sommes tout de même contents de ce que ce projet nous a apporté et de ce que nous avons réalisé. Notre code pourra par la suite servir de repère à d'autres équipes, comme nous avons fourni avec lui des informations et de la documentation servant à le comprendre et l'améliorer facilement.

GLOSSAIRE

ASCII : Norme informatique de codage de caractères. En effet, les machines ne comprennent que les nombres, le code ASCII est donc la représentation numérique de chaque caractère (tel que “&” ou “z”).

CSP : Le CubeSat Space Protocol est un protocole réseau permettant de faciliter la communication entre systèmes embarqués et distribués au sein d’un petit réseau, comme celui d’un CubeSat.

CSUG : Le Centre Spatial Universitaire de Grenoble a été fondé le 11 Septembre 2015 avec pour mission de développer de l’instrumentation spatiale miniaturisée. Pour cela, le CSUG implique des industriels, des chercheurs ainsi que des étudiants dans la construction de nanosatellites. À ce jour, pas moins de 6 projets de nanosatellites ont vu le jour ou sont en cours, dont le projet ATISE.

CSUT : Centre Spatial Universitaire de Toulouse, responsable de partie plateforme du projet ATISE.

Cubesat : Petits satellites de forme et poids standardisés à un cube d’un décimètre de côté.

DDR : Double Data Rate (Débit de Données Double)

FPGA : Un circuit logique programmable (Field Programmable Gate Array) est un circuit intégré logique que l’on peut reprogrammer une fois acheté.

PGM (Portable Gray Map) : Ce format appartient à la famille des formats Portable Pixmap. Le format PGM permet de coder des images en niveaux de gris. Chaque nuance de gris est codée avec un nombre entier compris entre 0 et N (sachant que la couleur noire est codée avec le nombre 0, tandis que la couleur blanche est codée par un nombre entier N, souvent 255).

SDK : Le kit de développement logiciel, connu sous le nom de System Development Kit, fournit un environnement pour la création de plateformes logicielles et d’applications destinées aux processeurs embarqués.

STM32 : Un microcontrôleur STM32 est un circuit intégré contenant un microprocesseur, plusieurs types de mémoire et des périphériques entrées/sorties de communication

VDMA : Le VDMA, soit Video Direct Access Memory, donne la possibilité à un périphérique d'entrées/sorties d’envoyer ou recevoir directement des données vers ou depuis la mémoire principale pour accélérer les opérations de mémoire.

BIBLIOGRAPHIE

CubeSat Space Protocol :

- libcsp/libcsp: Cubesat Space Protocol - A small network-layer delivery protocol designed for Cubesats

Mars ZX3 :

- https://www.enclustra.com/en/products/system-on-chip-modules/mars-zx3/

Logiciel VIVADO :

- Téléchargement : https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/archive.html

FreeRTOS :

- free RTOS source code for the Xilinx Zynq-7000 SoC

UART :

- https://www.circuitbasics.com/basics-uart-communication/

ANNEXES

BUDGET PRÉVISIONNEL			
Libellé	Quantité	Coût unitaire	Total (€)
MATIERES et COMPOSANTS :			
Total 1	0,00	0,00	0,00
FRAIS GENERAUX :			
- Déplacements	1	5	5
- Impression poster au format A0	1	40	40
- Communications	1	5	5
Total 2	3,00	16,67	50,00
MAIN D'ŒUVRE :			
- Ressources Humaines	4	3 020,20 €	12 080,80 €
-			
Total 3	4,00	3020,20	12080,80
INVESTISSEMENTS (AMORTISSEMENTS) :			
- MARS ZX3	1	12,8	12,8
- Raspberry Pi	2	0,6	1,2
- STM32 Evaluation Board	2	6,4	12,8
- Cartes SD	3	0,14	0,42
Total 4	8,00	3,40	27,22
TOTAL GENERAL			12158,02