Haute Ecole de la Province de Liège
Catégorie Technique

Travail de fin d'études

Implémentation de la gestion des télécommandes et des télémétries au sein de l'ordinateur de bord du nanosatellite OUFTI-1

Sebastien De Dijcker

Travail de fin d'études présenté en vue de l'obtention du grade de Bachelier en informatique et systèmes finalité informatique industrielle
Année académique 2010–2011

Change Log

Num. rév.	Date	Changements majeurs	Auteur
1.1	26/08/2011	$-pg.43$: Valeur de la variable d'envoi en cours $-pg.68$: Sous-Type de la TC DOPPLER_CORRECTION	De Dijcker Sebastien

Tableau 1 – Change Log

Remerciements

En préambule à ce travail de fin d'études, je tiens à remercier toutes les personnes qui ont contribué, de près ou de loin, à son élaboration.

J'adresse tout particulièrement mes remerciements à mon professeur superviseur, Monsieur Valery Broun, qui m'a offert la possibilité de travailler sur ce projet très intéressant.

Merci aussi à mon maître de stage, Nicolas Crosset, membre de l'équipe système du projet OUFTI-1, qui m'a suivi et encouragé tout au long de ma période de stage.

Je remercie ensuite le reste de l'équipe système du projet OUFTI-1, à savoir Amandine Denis (Project Manager), Jonathan Pisane, Jérôme Wertz, Nicolas Marchal et Laurent Chiarello pour leur accueil et leur soutien au sein de l'équipe ainsi que les professeurs Jacques Verly et Gaëtan Kerschen pour leur encadrement de l'équipe OUFTI-1.

Je tiens également à remercier Monsieur Paul Parisis de la société Spacebel, pour sa participation et ses remarques pertinentes lors de la Design Review.

Un tout grand merci à tous les étudiants qui ont travaillé sur le projet OUFTI-1 cette année, plus particulièrement à mon collègue Thomas Langohr avec qui j'ai beaucoup collaboré afin de développer un ordinateur de bord complet et cohérent.

Enfin, ce travail n'aurait pu aboutir sans l'aide de mes correcteurs et le soutien de ma famille et de mes amis que je remercie de tout cœur.

Liste des acronymes

ACK	Acknowledgement.
APID	Application Process Identifier.
AX.25	Amateur X.25.
BCN	Beacon.
Cal Poly	California Polytechnic State University.
CCSDS	Consultative Committee for Space Data Systems.
COM	Télécommunication.
D-STAR	Digital Smart Technologies for Amateur Radio.
EEPROM	Electrically Erasable Programmable Read-Only Memory.
EPS	Electrical Power Supply.
ESA	European Space Agency.
FCS	Frame-Check Sequence.
FIFO	First In, First Out.
GND	Ground Station.
GPL	Gnu Public License.
I^2C	Inter Integrated Circuit.
MECH	Mechanic.
MID	Measurement Identifier.
OBC	On-Board Computer.

OBSW	On-Board Software.
OUFTI-1	Orbital Utility For Telecommunication Innovations - 1.
P-POD	Poly Picosatellite Orbital Deployer.
PCB	Printed Circuit Board.
PID	Protocol Identifier.
PUS	Packet Utilization Standard.
RAM	Random Access Memory.
SCLK	Serial Clock.
SDATA	Serial Data.
SLE	Serial Load Enable.
SPP	Space Packet Protocol.
SREAD	Serial Readback.
SSID	Secondary Station Identifier.
STRU	Structure mécanique.
SWD	Sync Word Detect.
THER	Thermique.
TxRxCLK	Transmission-Reception Clock.
TxRxDATA	Transmission-Reception Data.
UI	Unnumbered Information.
ULg	Université de Liège.
VIB	Vibrations.
xEPS	eXperimental Electrical Power Supply.

Table des matières

Introduction

OUFTI-1 est un nanosatellite de type CubeSat, c'est-à-dire un cube de 10 cm de côté avec une masse maximale d'1 kg. Il est développé à l'Université de Liège par des générations successives d'étudiants depuis l'année académique 2007-2008. La charge utile principale de ce satellite est un relais D-STAR. Le D-STAR est un protocole de communication digital utilisé par les radioamateurs et qui offre la possibilité de transporter simultanément voix et données.

Afin de permettre le fonctionnement de la charge utile principale, différents sous-systèmes, dont l'ordinateur de bord, sont nécessaires. L'ordinateur de bord d'OUFTI-1 est le poste de contrôle de la totalité du satellite. Il permet la gestion des différents sous-systèmes et a la possibilité d'être commandé depuis le sol à l'aide de télécommandes. Une télécommande est une trame envoyée sur les ondes radio qui peut être réceptionnée, décodée et interprétée par le satellite.

L'ordinateur de bord d'OUFTI-1 a également la possibilité d'émettre des télémétries sur les ondes radio. Ainsi une communication bidirectionnelle sol-satellite est possible. Le protocole de communication utilisé pour les télécommandes et les télémétries est l'AX.25.

L'ordinateur de bord est principalement composé d'un microcontrôleur et d'une mémoire. Afin de gérer les différentes tâches au sein de ce microcontrôleur, le système d'exploitation temps réel FreeRTOS est utilisé.

Ce présent ouvrage est principalement consacré à l'implémentation de la communication sol-satellite, ainsi qu'à la gestion des télécommandes et télémétries au sein de l'ordinateur de bord d'OUFTI-1.

Cahier des charges

Le premier objectif à atteindre selon le cahier des charges est la maîtrise du système d'exploitation FreeRTOS et de la programmation du microcontrôleur MSP430F1612 en langage C afin de pouvoir travailler efficacement sur le reste du projet.

Les autres objectifs du cahier des charges ont été répartis en fonction des tâches exécutées par FreeRTOS. Néanmoins, il ne s'agit pas uniquement de tâches mais aussi d'espaces réservés en mémoire ou de routines d'interruptions, c'est pourquoi nous parlerons plutôt de modules. Chaque module a plusieurs fonctions à remplir.

Premièrement, le module COM Rx est dédié à la réception des informations depuis la station sol. Ces informations, ou télécommandes, parviennent à l'ordinateur de bord codées selon le protocole AX.25. Une fois décodées et interprétées, elles doivent être stockées dans un espace mémoire en fonction du Timestamp qui les accompagne. Ce Timestamp détermine le moment où doivent être exécutées ces commandes. Il faut donc implémenter un système efficace d'extraction de données des télécommandes AX.25 reçues ainsi qu'une gestion du système de mémoire où doivent être stockées ces commandes.

Ensuite, le module Sequencer est responsable de l'exécution des commandes présentes dans l'espace mémoire précédemment rempli par le module COM Rx. Il doit évidemment pouvoir traiter les différents types de commandes qu'il reçoit. Parmi ces commandes se trouvent également des corrections Doppler à envoyer au sous-système de communication afin qu'il les applique au D-STAR. Il faut donc implémenter un moyen de recevoir et d'exécuter ces corrections Doppler.

Enfin, le module COM Tx se charge du rapatriement des informations vers la station sol. Les autres modules lui donnent des données à transmettre, il se charge alors de les mettre en forme, des les encoder en AX.25 et de les envoyer.

Afin de pouvoir distinguer les différents types d'ordres que peut recevoir le satellite, ainsi que les différentes réponses qu'il peut émettre, il faut également définir précisément le format de ces télécommandes et télémétries.

Le projet OUFTI-1 ainsi que les différents sous-systèmes du nanosatellite sont présentés dans le chapitre 1. Le chapitre 2 reprend les différents composants utilisés afin de faire fonctionner l'ordinateur de bord et explique

leur fonctionnement. Les principes de fonctionnement de FreeRTOS, des différents modules ainsi que les méthodes de test sont exposés dans le chapitre 3. Le format des télécommandes et télémétries utilisées est ensuite présenté dans le chapitre 4.

Etat des lieux

Comme cela a déjà été précisé, les sous-systèmes du satellite OUFTI-1 sont développés par des générations successives d'étudiants. Ce travail se base donc sur des développements antérieurs. Ceux-ci sont présentés ici, afin de pouvoir clairement établir l'apport du présent travail.

Du point de vue hardware, une carte a été acquise auprès d'un fournisseur de composants pour CubeSat, une seconde carte permettant la redondance du système a été développée par Damien Teney [1] durant l'année académique 2008-2009. Aucun changement ne doit donc être réalisé à ce niveau.

Du point de vue du software, le module Sequencer existe déjà. En revanche, le système de gestion des commandes en mémoire n'est pas optimisé et l'exécution des différentes commandes n'est pas implémenté.

Les modules COM Rx et COM Tx ne sont pas implémentés. L'ajout d'une commande dans l'espace mémoire se faisait en interne au démarrage de l'application afin de pouvoir tester le module Sequencer.

Les routines d'encodage et de décodage AX.25 sont implémentées mais doivent être intégrées au software de l'ordinateur de bord.

Plusieurs pilotes existent déjà comme ceux permettant d'envoyer des bytes via le port UART de la carte de développement, ou encore la gestion du bus I^2C.

Le format général des télécommandes et télémétries a été défini par Laurent Chiarello [2] en 2008-2009. Néanmoins tous les services que doit pouvoir exécuter le satellite ne sont pas définis. C'est pourquoi il faut ajouter des télécommandes et des télémétries, ainsi que préciser le format exact de ces dernières.

1. cf. [Ten09].
2. cf. [Chi09].

Le software de l'ordinateur de bord n'en est qu'au début de sa phase de développement, aucune précaution particulière n'a été prise afin d'économiser de la mémoire au sein du microcontrôleur de l'ordinateur de bord.

Les principaux apports du présent travail consisteront donc en l'amélioration du module Sequencer après avoir défini précisément le format des télécommandes et des télémétries. Les modules COM Rx et COM Tx sont quant à eux à implémenter complètement et à tester.

Chapitre 1

OUFTI-1

1.1 Le standard CubeSat

Tout d'abord, il faut savoir qu'en raison du nombre croissant de satellites de petite taille en développement, un classement en fonction de la masse existe :

- Minisatellite : Satellite de masse comprise entre 100 et 500 kg. Il utilise souvent le même équipement que les gros satellites, et peut se permettre d'être équipé de propulseurs pour le contrôle d'attitude ;

- Microsatellite : Satellite de masse comprise entre 10 et 99,99 kg. La miniaturisation se fait déjà ressentir, néanmoins des propulseurs sont encore utilisés ;

- Nanosatellite : Satellite de masse comprise entre 1 et 9,99 kg. Tous les composants se doivent d'être réduits en terme de masse et de volume, les propulseurs sont donc souvent une option non envisageable. Ce type de satellite peut être lancé en tant que passager secondaire d'un lanceur destiné à un satellite commercial plus important ;

- Picosatellite : Satellite de masse comprise entre 0,1 et 0,99 kg. Le processus de miniaturisation est au maximum, par conséquent, des composants nouvelle technologie doivent souvent être utilisés. Les picosatellites sont également lancés à bord d'un lanceur destiné à un satellite commercial plus important.

Défini par la California Polytechnic State University (Cal Poly) et l'université de Stanford, le standard CubeSat [1] est un exemple de nanosatellite utilisant typiquement des composants commerciaux, et pas nécessairement qualifiés pour le spatial. Pour répondre au standard CubeSat, certains critères doivent être respectés, dont voici les plus importants :

- Chaque CubeSat ne peut excéder une masse d'un kg ;
- Les dimensions sont de 100 mm ± 0,1 selon les axes X et Y et de 113,5 mm ± 0,1 selon l'axe Z (cf. figure 1.1) ;
- Le centre de gravité doit se trouver à maximum 2 cm du centre géométrique du CubeSat ;
- Le CubeSat ne peut pas être un danger pour les CubeSats avoisinants, pour le lanceur ou pour le satellite principal ;
- Aucune partie du CubeSat ne peut se détacher de celui-ci, ni pendant le lancement, ni pendant l'éjection, ni dans sa phase orbitale ;
- Aucun système pyrotechnique ne peut se trouver à bord du CubeSat.

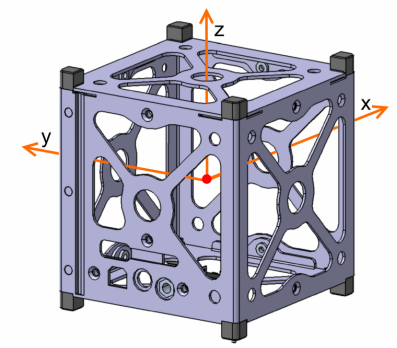

FIGURE 1.1 – Structure et axes d'un CubeSat. Source : [DP09]

Pour permettre aux CubeSats d'être passagers secondaires lors du lancement d'un satellite plus important, l'université de Cal Poly a également développé un standard de déployeur : le Poly Picosatellite Orbital Deployer

1. cf. [Pro08].

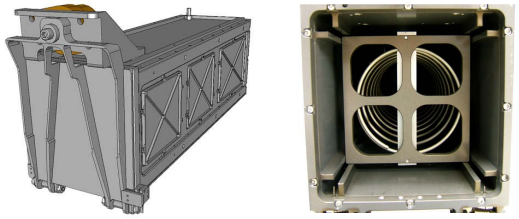

FIGURE 1.2 – Le déployeur P-POD. Source : [Pro08]

(P-POD) (cf. figure 1.2). Ce dernier a la possibilité de contenir trois CubeSats d'une unité (1U) et permet d'assurer la sécurité du satellite principal. Du point de vue du fonctionnement, un ressort est placé dans le fond du P-POD, les trois CubeSats sont ensuite insérés, et la porte du P-POD refermée. Lorsque le lanceur a atteint l'altitude et l'emplacement désirés, la porte du P-POD s'ouvre, et les trois CubeSats sont éjectés par la poussée du ressort. Etant donné qu'il possède assez de place pour contenir trois CubeSats, cela permet de développer des CubeSats de deux ou trois unités (2U ou 3U). Ainsi un CubeSat peut avoir une masse allant jusqu'à 2 kg pour le 2U ou 3 kg pour le 3U. La section du CubeSat doit évidemment rester la même, tandis que la hauteur est alors multipliée par deux ou par trois.

1.2 Le projet OUFTI-1

Orbital Utility For Telecommunication Innovations - 1 (OUFTI-1) est le premier projet étudiant de CubeSat belge. OUFTI-1 est développé à l'Université de Liège (ULg) dans l'optique d'un projet éducatif à long terme : l'initiative Léodium. Le but principal de cette initiative est de fournir une expérience pratique dans le domaine de la conception de satellites, de la phase de design au contrôle en orbite.

FIGURE 1.3 – Logo OUFTI-1. Source : [dL11]

OUFTI-1 est le premier satellite de cette initiative et servira donc de base à la conception des futurs satellites expérimentaux de l'ULg. À l'heure actuelle, un instrument pour une potentielle mission OUFTI-2 est d'ailleurs en phase d'analyse de faisabilité.

1.3 Les sous-systèmes d'OUFTI-1

Comme tout satellite, la plateforme d'OUFTI-1 est divisée en différents sous-systèmes, selon les fonctionnalités nécessaires au bon fonctionnement de la charge utile et du satellite dans son ensemble. Cette section est dédiée à l'explication des différents rôles de chaque sous-système.

1.3.1 STRU

La Structure mécanique (STRU) décrit la façon dont sont agencés les différents composants du satellite (PCBs, connectique, éléments mécaniques, etc.) et assure leur maintien sur l'ossature du CubeSat.

1.3.2 MECH

Le sous-système Mechanic (MECH) comprend le mécanisme de rétention et de déploiement des antennes. En effet, la communication entre le satellite et la station sol nécessite deux antennes de respectivement 50 et 17cm. Ces antennes ne peuvent pas se déployer avant les trente minutes qui suivent l'éjection du CubeSat afin d'éviter d'endommager les autres satellites présents dans le module P-POD. La figure 1.4 représente le support de déploiement des antennes qui sera collé sur une des six faces du CubeSat.

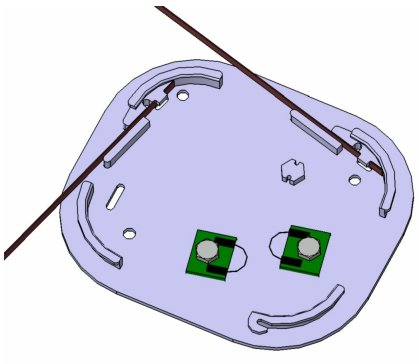

FIGURE 1.4 – Support de déploiement des antennes. Source : Amandine Denis

1.3.3 THER

Le sous-système de Thermique (THER) doit s'assurer que tous les composants du satellite restent dans une plage de température acceptable durant la mission, les capteurs de température doivent également être positionnés correctement afin de permettre une précision de mesure optimale. Finalement, un système d'asservissement comportant un capteur de température et une chaufferette permet de réguler la température des batteries afin de les protéger du froid lorsque le satellite est en éclipse.

1.3.4 VIB

Le sous-système de Vibrations (VIB) est l'étude de la résistance d'OUFTI-1 aux chocs et vibrations durant tout son cycle de vie : de l'assemblage du modèle de vol à sa vie orbitale, en passant par la phase de lancement, qui est la phase la plus contraignante.

1.3.5 EPS

Le sous-système Electrical Power Supply (EPS) inclut les panneaux solaires, les batteries et le Printed Circuit Board (PCB) de l'EPS. Les différentes fonctions de ce sous-système sont :

- la distribution de trois tensions électriques différentes (3.3V, 5V, 7.2V) à la totalité du satellite via trois bus d'alimentation distincts ;
- la génération de puissance électrique grâce aux panneaux solaires ;
- le stockage de surplus d'énergie dans les batteries afin de pouvoir restituer cette énergie ultérieurement, notamment lorsque le satellite est en éclipse.

1.3.6 COM

Le sous-système de Télécommunication (COM) est composé d'un récepteur qui permet de recevoir des télécommandes et d'un transmetteur afin de transmettre des télémétries. Ce canal de communication avec le sol utilise le protocole Amateur X.25 (AX.25). Le sous-système COM gère également le répéteur Digital Smart Technologies for Amateur Radio (D-STAR) en mode voix pour les télécommunications radioamateur. Il est à noter que l'encodage et le décodage des trames AX.25 est la responsabilité du sous-système On-Board Computer (OBC).

1.3.7 BCN

Le sous-système Beacon (BCN) est la balise morse de secours du satellite. Il transmet en continu douze mesures caractéristiques du satellite et cela indépendamment des autres sous-systèmes. En effet il n'a pas besoin du sous-système COM pour transmettre son message et peut s'affranchir du sous-système OBC car il dispose de sa propre chaîne de mesure.

1.3.8 OBC

L'On-Board Computer (OBC) est chargé du contrôle de la totalité du satellite. Il interprète les ordres venant du sol, les traite, et rapatrie les résultats. Il surveille et entretient le bon fonctionnement du système. Voici les différents rôles qui lui sont attribués :

- déploiement des antennes et activation des autres sous-systèmes ;
- acquisition, enregistrement et rapatriement de mesures ;
- transmission d'une référence temporelle à la totalité du système ;
- activation / désactivation des autres sous-systèmes en fonction de certaines conditions ;
- redémarrage d'un sous-système en cas de court-circuit ;
- gestion et configuration des payloads (xEPS et D-STAR) ;
- configuration du récepteur et de l'émetteur ;
- décodage des télécommandes ;
- ordonnancement des télécommandes afin de les exécuter ;
- encodage des télémétries.

1.3.9 GND

Le sous-système Ground Station (GND) est le seul à ne pas faire partie du satellite car il se trouve dans le segment sol et non dans le segment espace. Son rôle est de permettre la communication avec le satellite OUFTI-1 une fois qu'il sera en orbite et donc de le commander en envoyant des télécommandes et en réceptionnant les télémétries. Il doit ensuite être capable d'écouter et de décoder le signal morse provenant de la balise à bord du satellite et être en mesure d'effectuer des communications D-STAR afin de pouvoir tester la payload principale d'OUFTI-1. La figure 1.5 schématise l'implémentation du sous-système GND.

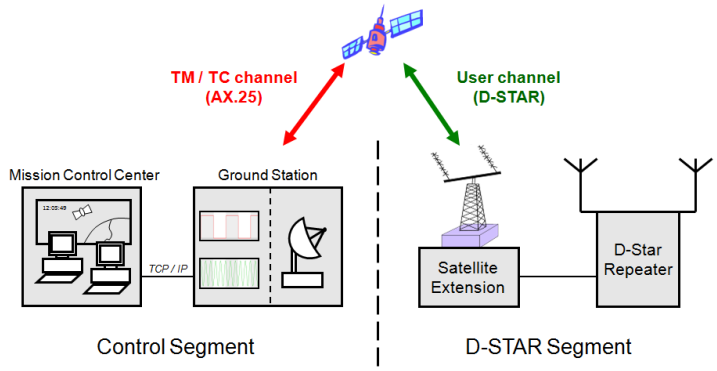

FIGURE 1.5 – Le sous-système GND. Source : [DP09]

1.4 Les payloads d'OUFTI-1

1.4.1 D-STAR

Le D-STAR est un protocole de communication radioamateur numérique permettant de transporter simultanément voix et données. Il est utilisable sur les bandes radioamateur classiques (VHF, UHF et L) et propose également une méthode de connexion aux réseaux informatiques (dont internet) via TCP/IP. OUFTI-1 sera le premier satellite équipé d'un relais D-STAR permettant de relier deux opérateurs au sol avec une couverture bien plus large qu'un relais terrestre traditionnel.

Il faudra cependant tenir compte de l'effet Doppler induit par la vitesse de révolution du satellite par rapport aux utilisateurs au sol. Les émetteurs/récepteurs D-STAR actuels ne permettant pas de corriger l'effet Doppler de façon suffisamment fine, les compensations doivent donc être effectuées à bord du satellite. Afin de simplifier l'utilisation du relais à bord, les données permettant de calibrer la compensation Doppler pour deux zones distinctes seront envoyées à OUFTI-1 par le canal de télécommunication classique (AX.25) avant chaque établissement d'une communication D-STAR.

1.4.2 xEPS

L'eXperimental Electrical Power Supply (xEPS) constitue une expérience technologique à bord d'OUFTI-1. La caractéristique principale de cette alimentation électrique expérimentale est son convertisseur de tension numérique chargé de réguler du 3,3 V. L'intérêt de l'expérimentation est que ce

type de convertisseur n'a jamais été testé dans l'espace, contrairement aux alimentations à convertisseurs analogiques. La figure 1.6 représente la carte xEPS.

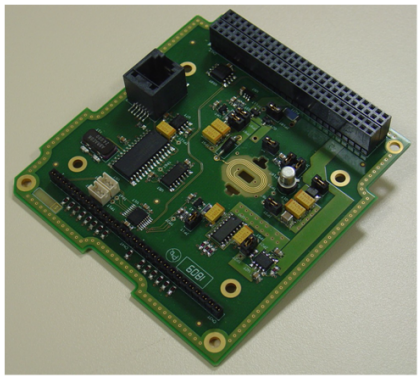

FIGURE 1.6 – La carte xEPS. Source : [Led09]

L'xEPS peut au choix alimenter une charge résistive ou le bus 3,3 V. Lors de la mise en marche de l'xEPS (par télécommande), l'OBC doit en premier lieu utiliser la charge résistive afin de vérifier le bon fonctionnement de l'xEPS avant de passer sur le bus d'alimentation 3,3 V du satellite.

1.4.3 Cellules solaires

Pour pouvoir produire sa propre énergie électrique, OUFTI-1 utilise des cellules solaires à haut rendement fournies par la société AZUR SPACE Solar Power. Ces cellules solaires à triple jonction sont capables d'atteindre les 30% de rendement. AZUR SPACE fournit gracieusement ces cellules solaires en contrepartie des données recueillies sur celles-ci à bord d'OUFTI-1. La figure 1.7 représente une des dix cellules présentes sur OUFTI-1.

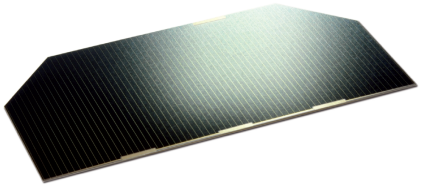

FIGURE 1.7 – Cellule solaire d'OUFTI-1. Source : [AZU09]

1.5 Les objectifs du projet

La mission d'OUFTI-1 est définie de façon graduelle et comporte donc différents objectifs [2] à remplir pour la mener à bien. Chaque objectif comporte un ou plusieurs critères de succès qui permettent de déterminer si l'objectif est atteint ou non. Voici cette liste par ordre d'importance :

1. **Amusement et éducation**. Le premier objectif concerne les différentes équipes d'étudiants travaillant sur le projet. Chaque étudiant doit prendre du plaisir à travailler sur son sous-système. L'objectif principal est en effet de leur apporter de l'expérience dans le domaine spatial et donc de mener à leur développement personnel. Les trois critères de réussite de cet objectif sont les suivants :
   (a) Les étudiants devraient obtenir de bonnes notes pour leur travail sur le projet ;
   (b) Un nombre significatif d'étudiants devrait avoir envie de travailler sur le projet OUFTI-1 chaque année ;
   (c) Le succès d'une année de travail peut être mesuré par la quantité d'étudiants diplômés qui continuent à participer activement ou non au projet.

2. **Design du système OUFTI-1**. Après l'amusement et l'éducation, le but principal et de construire un satellite, ce qui implique le design et l'implémentation d'un système spatial et d'un système sol. Le critère de succès de cet objectif est d'avoir un satellite fonctionnel et son système sol correspondant fonctionnant correctement.

3. **Lancement d'OUFTI-1**. Une fois le satellite construit, la prochaine étape est son lancement. Le critère de succès est un lancement effectué correctement.

4. **OUFTI-1 fonctionnant dans l'espace**. Après le lancement, la vie du satellite démarre. Le critère de succès est de recevoir un signal du satellite, indiquant qu'il est en vie.

5. **Commander le satellite**. Le critère de succès est de pouvoir envoyer une télémétrie et de recevoir une télécommande.

2. cf. [DP09].

6. **Avoir un système D-STAR fonctionnel**. La payload D-STAR sera la première à être activée car il s'agit de la payload principale. Le but est de prouver que le protocole D-STAR fonctionne dans l'espace. Un unique contact D-STAR via le satellite est suffisant pour considérer la payload D-STAR comme un succès.

7. **Faire fonctionner les payloads secondaires**. Les deux payloads secondaires, les cellules solaires et l'xEPS, sont finalement prises en compte. Le critère de succès est de recevoir des télémétries de ces deux payloads.

Chapitre 2

Hardware

Ce chapitre est consacré à l'explication des différents composants matériels utilisés afin de pouvoir faire fonctionner la partie software de l'ordinateur de bord d'OUFTI-1.

2.1 Le bus PC/104

Toutes les cartes composant le satellite OUFTI-1 sont munies d'un connecteur PC/104 mâle et/ou femelle. Ainsi elles s'enfichent les unes dans les autres et l'information ou l'alimentation peut facilement passer d'une carte à l'autre. Ce bus est composé, comme son nom l'indique, de 104 pins. Le gros connecteur noir visible sur le dessus de la figure 2.1 de la page 16 est un connecteur PC/104 femelle.

2.2 OBC

2.2.1 Redondance des OBCs

L'On-Board Computer (OBC) étant le cerveau du satellite, s'il venait à tomber en panne, la communication avec OUFTI-1 deviendrait impossible et le satellite serait alors perdu. Pour réduire les risques de panne à ce niveau, il a été décidé de placer non pas un, mais deux OBCs à bord d'OUFTI-1. Ainsi, si un OBC tombe en panne, le second OBC doit être capable de prendre la main afin de gérer la totalité du satellite. On parle alors respectivement de **default OBC** et de **backup OBC**. Le système de gestion de la redondance est expliqué en détails dans le travail de fin d'études de Thomas Langohr [1].

1. cf. [Lan11].

Le backup OBC

Le backup OBC doit être fiable et l'on doit être sûr qu'il fonctionne dans l'environnement spatial. C'est pourquoi le backup OBC est une carte FM430 de la société Pumpkin. Cette carte est vendue avec la structure du satellite qui répond au standard CubeSat. Le principal avantage de cette carte est qu'elle possède un héritage de vol, c'est-à-dire qu'elle a déjà été utilisée et testée dans d'autres CubeSats, et c'est pourquoi il s'agit d'une valeur sûre. Elle est composée entre autres d'un microcontrôleur MSP430F1612 et d'un connecteur USB utilisé pour charger les batteries. La carte FM430 est illustrée sur la figure 2.1.

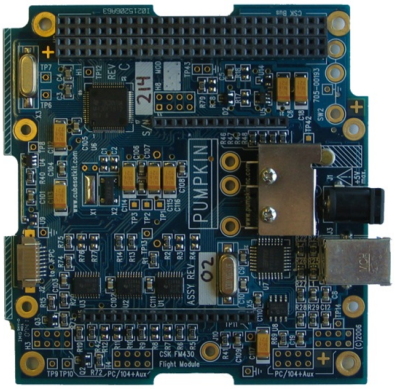

FIGURE 2.1 – Backup OBC - FM430. Source : [Ten09]

Le default OBC

Le default OBC est prioritaire par rapport au backup OBC, c'est-à-dire qu'il contrôlera la totalité du satellite tant qu'une panne ne survient pas. S'il a la possibilité de redémarrer après une panne, il reprendra le contrôle du satellite. Pour cet OBC, une carte FM430 aurait également pu être choisie, mais ne l'a pas été étant donné qu'elle comporte beaucoup de composants inutilisés comme le lecteur de carte SD et que cela consomme de la place et de la masse. Il s'agit de la raison pour laquelle une carte fabriquée sur mesure a été développée par des étudiants du projet OUFTI-1 des années précédentes. Cette carte illustrée sur la figure 2.2 est principalement composée d'un microcontrôleur MSP430F1612 (le même que sur la carte FM430) et d'une EEPROM (cf. chapitre 2.2.4). Les microcontrôleurs des deux cartes

sont reliés pin à pin via le PC/104 et le mécanisme de gestion de la redondance est implémenté via le bus I^2C.

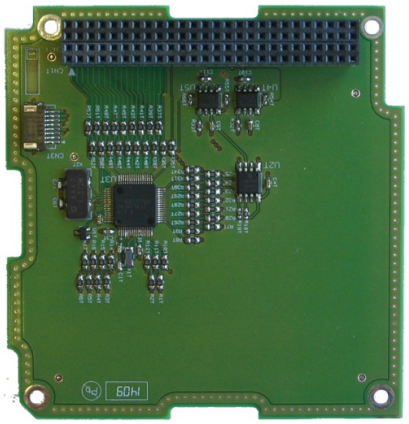

FIGURE 2.2 – Default OBC - Carte faite maison. Source : [Ten09]

2.2.2 Le microcontrôleur MSP430

Les microcontrôleurs présents sur chaque OBC sont des MSP430F1612. Fabriqué par la société Texas Instrument, le MSP430F1612 est un microcontrôleur RISC 16 bits qui a comme particularité de consommer très peu de courant. Il est cadencé à une fréquence de 7,3728 MHz par un quartz externe situé sur chacun des OBC.

Voici les caractéristiques et fonctionnalités principales de ce MSP430F1612 [2] :

- plage d'alimentation faible (entre 1,8 et 3,6 Volts) ;
- consommation en courant très faible (maximum 4,48668 mA à une fréquence de 7,3728 MHz avec une tension d'alimentation de 3,3 V en mode actif) ;
- plusieurs modes basse consommation afin de réduire encore plus la consommation électrique ;
- un convertisseur analogique-digital 12 bits à 8 canaux ;

2. cf. [Ins02].

- un convertisseur digital-analogique 12 bits à 2 canaux ;
- deux Timers 16-bits ;
- deux interfaces de communication série (USART0 et USART1) pouvant fonctionner en UART asynchrone, en SPI synchrone ou en Inter Integrated Circuit (I^2C)(uniquement USART0) ;
- deux ports d'entrée-sortie de 8 bits avec possibilité d'interruption ;
- quatre ports d'entrée-sortie de 8 bits ;
- 55 Kbytes + 256 bytes de mémoire Flash ;
- 5 Kbytes de Random Access Memory (RAM).

La figure 2.3 représente l'architecture interne du MSP430F1612.

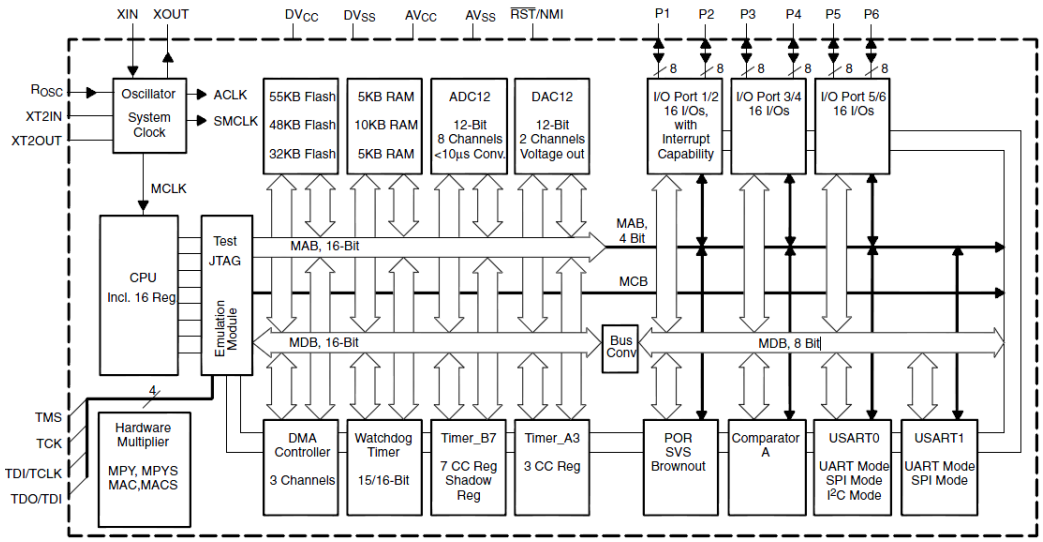

FIGURE 2.3 – Architecture interne du microcontrôleur MSP430F1612. Source : [Ins02]

2.2.3 Le bus I²C

Le bus Inter Integrated Circuit (I²C) est un bus série composé de trois fils : un pour les données, un pour la masse et un pour la clock. Utilisant le protocole de communication I²C, il permet la communication série entre différents composants électroniques. Il sera utilisé à bord d'OUFTI-1 dans les communications suivantes :

- gestion de la redondance entre les deux OBCs ;
- écriture et lecture dans l'EEPROM ;
- lecture des valeurs converties par les différents ADCs.

Etant donné que ce bus ainsi que son protocole de communication ne sont pas utilisés dans le cadre de ce travail, ils ne sont pas étudiés plus en profondeur. Pour plus d'informations sur le bus et le protocole I²C, référez-vous au travail de fin d'études de Thomas Langohr [3].

2.2.4 L'EEPROM

Située sur le default OBC, l'Electrically Erasable Programmable Read-Only Memory (EEPROM) est utilisée afin de mémoriser les mesures prises à bord du satellite et les événements majeurs se produisant durant la vie du satellite comme le déploiement des antennes ou encore les changements de mode de fonctionnement du satellite. Il s'agit d'une EEPROM 24FC1025 [4] de 1024 bytes fabriquée par Microchip. Elle a la particularité d'être adressable via le bus I²C et peut donc être utilisée par les deux OBCs lorsqu'ils en ont besoin. Sa consommation en courant est de maximum 5 mA en écriture et de 500 μA en lecture.

Etant donné que cette EEPROM n'est pas utilisée dans le cadre de ce travail, son fonctionnement n'est pas étudié plus en profondeur. Pour de plus amples informations sur l'EEPROM 24FC1025, référez-vous au travail de fin d'études de Thomas Langohr [5].

3. cf. [Lan11].
4. cf. [Mic05].
5. cf. [Lan11].

2.3 COM

La carte COM, également connectée au bus PC/104, fait partie du sous-système COM. Néanmoins, l'OBC utilise un composant de cette carte, c'est pourquoi il est nécessaire d'en expliquer le fonctionnement.

2.3.1 L'émetteur-récepteur ADF7021

Utilisé par l'OBC pour envoyer et recevoir des trames AX.25, les émetteurs-récepteurs ADF7021 [6] sont localisés sur la carte COM. Il y en trois au total. Le premier, configuré par le microcontrôleur COM, est utilisé pour recevoir des trames D-STAR. Le second, configuré par le microcontrôleur OBC, sert de récepteur AX.25. Le troisième est utilisé à la fois pour l'émission D-STAR à travers le microcontrôleur COM et pour l'émission AX.25 à travers le microcontrôleur OBC. C'est la raison pour laquelle il est géré au travers d'un multiplexeur. L'OBC a la responsabilité de donner le contrôle de l'ADF à la COM ou à lui-même car ils ne peuvent l'utiliser en même temps.

Configuration de l'ADF7021

L'OBC doit, pour recevoir et envoyer des trames AX.25, configurer les ADFs d'émission et de réception pour qu'entre autres ils émettent et reçoivent à la bonne fréquence. Pour ce faire, l'ADF7021 possède une interface série composée de plusieurs entrées-sorties destinées à configurer ses registres internes.

L'entrée **Serial Data (SDATA)** est utilisée pour envoyer les registres de configuration bit par bit du microcontrôleur OBC vers l'ADF.

L'entrée **Serial Clock (SCLK)** est destinée à recevoir le signal d'horloge de réception des données. Le microcontrôleur OBC génère donc ce signal afin de notifier à l'ADF qu'une donnée est disponible sur l'entrée SDATA. L'ADF ajoute le bit lu sur SDATA dans son registre à décalage de 32 bits lors d'un flanc montant sur SCLK.

L'entrée **Serial Load Enable (SLE)** doit être positionnée à 1 lorsque les 32 bits ont bien été envoyés à l'ADF. Ainsi il peut placer dans ses registres de configuration les bits se trouvant dans le registre à décalage. Afin de savoir à quel registre de configuration correspondent les bits présents dans le registre à décalage, les 4 bits de poids faible correspondent aux Control Bits. Ceux-ci

6. cf. [Dev07].

désignent le registre de configuration dont il s'agit.

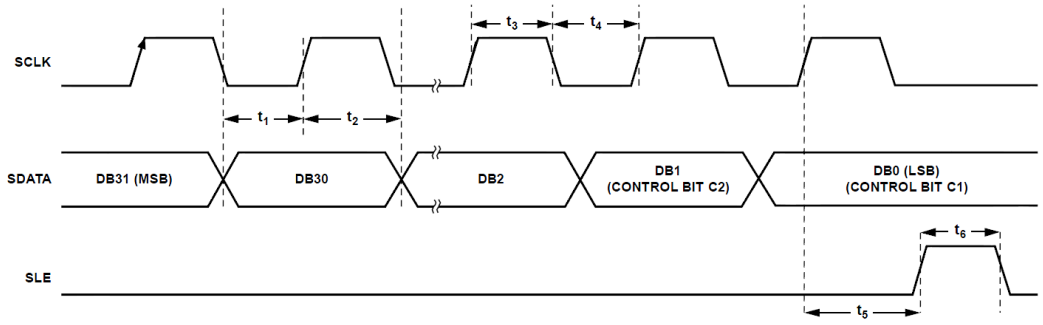

FIGURE 2.4 – Chronogramme de la configuration d'un ADF7021. Source : [Dev07]

La figure 2.4 est un chronogramme représentant les différentes entrées de l'ADF7021. On remarque donc que les données sont placées par l'OBC après un flanc descendant de SCLK, et que l'ADF échantillonne cette donnée lors du flanc montant de SCLK. Une fois les 32 bits envoyés, le SLE est bien positionné à 1 pendant au moins 20 nsec. Une demi période de SCLK doit quant à elle être de minimum 10 nsec.

La sortie **Serial Readback (SREAD)** permet quant à elle de récupérer des valeurs de registres de l'ADF vers le microcontrôleur OBC. Pour ce faire, SCLK, SDATA et SLE sont toujours utilisés. La marche à suivre pour récupérer un registre de l'ADF est la suivante : premièrement, il faut placer un mot de contrôle dans le registre de readback de la même manière qu'expliqué précédemment, ce qui permettra à l'ADF de savoir quel registre le microcontrôleur OBC aimerait recevoir. Une fois le registre écrit, il suffit de laisser SLE à 1, et à chaque flanc montant de SCLK l'ADF va placer un bit du registre désiré sur SREAD. Le microcontrôleur OBC peut alors lire les données sur chaque flanc descendant de SCLK. Une fois la réception terminée, SLE doit être remis à zéro.

Envoi et réception de données

Une fois la configuration des ADFs correctement effectuée, le microcontrôleur OBC peut alors commencer à envoyer et à recevoir des données. Pour ce faire, c'est l'ADF qui va cette fois générer le signal d'horloge sur sa sortie **Transmission-Reception Clock (TxRxCLK)**. Pour la réception, l'ADF

va placer la donnée sur sa sortie **Transmission-Reception Data (TxRxDATA)** à chaque flanc descendant de TxRxCLK. Le microcontrôleur peut donc lire un bit à chaque flanc montant de TxRxCLK. Pour l'émission, c'est le microcontrôleur OBC qui doit placer un bit à chaque flanc descendant de TxRxCLK et l'ADF échantillonnera la donnée sur le flanc montant.

La figure 2.5 représente les différentes entrées-sorties de l'ADF en fonction du temps en mode réception de données. La vitesse de transfert de données utilisée pour l'émission et la réception est de 9600 bits par seconde.

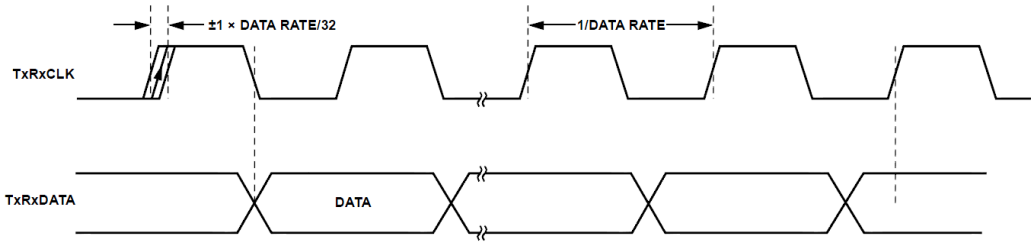

FIGURE 2.5 – Chronogramme de la réception de données de l'ADF7021. Source : [Dev07]

La particularité de la réception est que l'ADF7021 est capable de détecter un mot de synchronisation dans les données qu'il reçoit. Ainsi, lorsqu'une trame AX.25 lui parvient, et qu'elle est précédée d'un mot de synchronisation, il positionne sa sortie Sync Word Detect (SWD) à 1, ce que le microcontrôleur OBC peut facilement détecter. Cela permet à ce dernier de ne pas être interrompu dans ses tâches lorsque l'ADF écoute le vide. Si ce n'était pas le cas, le microcontrôleur OBC serait interrompu 9600 fois par seconde afin de réceptionner des bits sans importance. En utilisant ce mécanisme, il est possible d'interpréter les données uniquement lorsqu'une trame AX.25 est en cours de réception, et de ne rien interpréter le reste du temps.

Chapitre 3

Software

3.1 FreeRTOS

FreeRTOS est le système d'exploitation utilisé par l'On-Board Software (OBSW). FreeRTOS est un système d'exploitation Gnu Public License (GPL) temps réel écrit en langage C. Voici les avantages que nous tirons de ce système d'exploitation au sein de l'OBSW [1] :

- Il permet l'utilisation du multitâches ;
- Il gère plusieurs niveaux de priorité d'exécution entre les différentes tâches ;
- Chaque tâche est préemptible, une tâche de basse priorité sera donc interrompue immédiatement [2] si une tâche de plus haute priorité est prête à être exécutée ;
- Le système de gestion des tâches de même priorité est le Round-Robin, c'est-à-dire que toutes les tâches d'un même niveau de priorité vont s'exécuter en boucle (l'une après l'autre, et une fois arrivé à la dernière, on recommence à la première) ;
- L'utilisation de sémaphores avec délai d'attente pour éviter les interblocages ;

1. cf. [Bar09], [Bar10] et [Rea11].
2. C'est un Hard RTOS, les tâches sont donc interrompues immédiatement, et pas à la fin du time-slice.

- La compatibilité avec le microcontrôleur MSP430F1612 utilisé ;
- Il s'agit d'un système d'exploitation très léger (entre 4 et 9 Kbytes pour l'image binaire) ;
- Le code du système d'exploitation est Open Source, les sources peuvent donc être consultées et modifiées si nécessaire ;
- Comme son nom l'indique, il s'agit d'un système d'exploitation gratuit.

3.1.1 Les tâches

Comme cité précédemment, FreeRTOS permet l'utilisation de plusieurs tâches. Ce chapitre est consacré à l'explication détaillée de l'agencement de ces tâches au sein de la mémoire du microcontrôleur ainsi qu'à la gestion de leur exécution au cours du temps [3].

Premièrement, une tâche peut, au cours de son cycle de vie, se trouver dans un des 4 états suivants :

- Ready : La tâche est prête à être exécutée, elle attend que le scheduler du système d'exploitation lui donne la main ;
- Running : La tâche est en cours d'exécution, elle a la main sur le processeur. Plusieurs tâches ne seront jamais simultanément dans cet état ;
- Blocked : La tâche est bloquée, elle ne peut être exécutée dans l'immédiat. Elle passera dans l'état Ready si un événement survient (déblocage d'un sémaphore, fin d'un délai, etc.) ;
- Suspended : La tâche a été explicitement suspendue par une autre tâche ou par elle-même (appel à vTaskSuspend()). Elle n'est plus disponible pour le scheduler du système d'exploitation. Elle repassera en mode Ready si une autre tâche en fait explicitement la demande (appel à vTaskResume()).

3. cf. [Rea11].

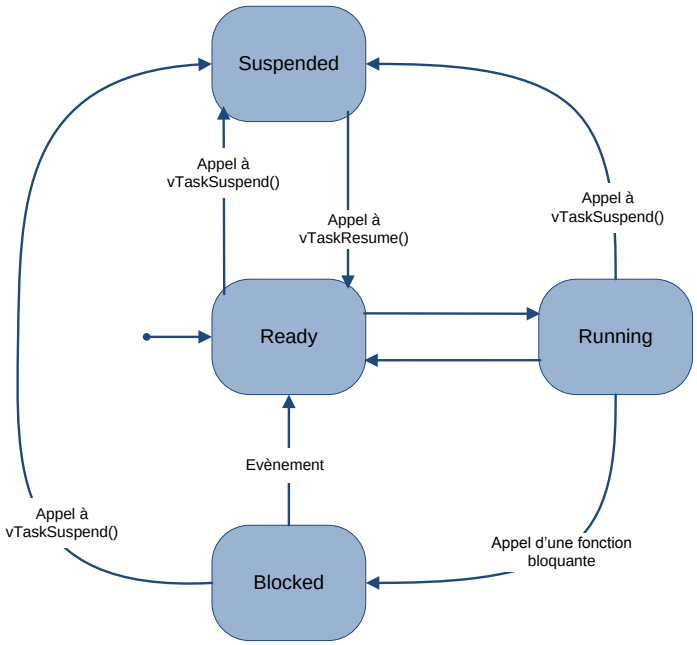

FIGURE 3.1 – Machine à état des tâches

La figure 3.1 illustre la machine à état des tâches. C'est le rôle du scheduler du système d'exploitation de gérer cette machine à état. Si plusieurs tâches sont dans l'état Ready et sont donc prêtes à être exécutées, le scheduler va passer la tâche Ready de plus haute priorité à l'état Running. Si, lorsque cette tâche est en cours d'exécution, une tâche de plus haute priorité passe à l'état Ready, le scheduler va interrompre l'exécution de la tâche courante, positionner son état à Ready, et donner la main à la tâche de plus haute priorité en positionnant son état à Running. Il faut également préciser que l'environnement volatile [4] de la tâche en cours d'exécution est sauvegardé en mémoire si la tâche doit libérer le processeur. Cet environnement volatile sera restitué lors de son prochain passage à l'état Running, et la tâche pourra donc continuer son travail là où elle s'était arrêtée.

Ensuite, au niveau de la mémoire, chaque tâche possède son propre Stack, c'est-à-dire que les variables locales sont propres à chacune. Cela peut consommer plus de mémoire en RAM car si plusieurs tâches font appel à une même fonction, les variables de cette fonction vont être mémorisées plusieurs fois, à savoir une fois par tâche. Cependant, toutes les tâches partagent le même segment de Code, de Data et de Heap. Ainsi, toutes les tâches peuvent ac-

4. L'environnement volatile d'une tâche correspond à toutes les variables et tous les registres du microcontrôleur au moment de son exécution.

céder à toutes les variables globales des autres tâches. Cela est très pratique pour échanger des informations entre les tâches. Cependant il faut être sûr que deux tâches ne vont pas accéder à la même donnée en même temps. Par exemple, si une tâche va lire une variable globale et la copie dans une variable locale afin d'effectuer un traitement sur celle-ci. Mais juste après la copie, l'exécution de la tâche est interrompue car une tâche de plus haute priorité est prête à être exécutée. Cette tâche de plus haute priorité peut aller modifier cette variable globale. Lorsque la première tâche reprend la main, elle va effectuer un traitement à partir de sa variable locale, qui ne correspondra plus à la valeur de la variable globale. Pour remédier à ce problème, l'utilisation des sémaphores est nécessaire.

3.1.2 Les sémaphores binaires

Un sémaphore agit comme un feu rouge à un carrefour, le carrefour représentant une variable globale. Il est initialisé à 1, ce qui correspond à un feu vert. Si une tâche a l'intention d'accéder à la variable globale correspondant au sémaphore, elle prend le sémaphore, ce qui va décrémenter sa valeur de 1 à 0. Ainsi, le feu passe au rouge pour les autres tâches. Tant que cette tâche n'aura pas relâché le sémaphore, aucune autre tâche ne pourra accéder à cette variable, car le feu sera au rouge. Si la tâche relâche le sémaphore, le feu repasse au vert et une autre tâche pourra accéder à ce sémaphore.

Plus concrètement, il s'agit d'un objet système comprenant un compteur. Ce compteur est initialisé à 1 et ne peut pas être négatif. Lorsqu'une tâche tente d'accéder à une variable globale, elle décrémente ce compteur. Ainsi, lorsqu'une autre tâche tente de prendre le sémaphore, elle sera bloquée car la valeur du sémaphore vaudra zéro. Quand la tâche relâche le sémaphore, elle incrémente son compteur de 1, ainsi les tâches en attente pourront à nouveau décrémenter le sémaphore. Les opérations d'incrémentation et de décrémentation de l'objet sémaphore se font de façon atomique, c'est-à-dire que ces opérations ne peuvent pas être interrompues par quoi que ce soit. Comme il s'agit d'opérations extrêmement courtes, cela ne pose pas de problème quant à la réactivité du reste du système.

Ce système permet d'assurer l'exclusion mutuelle sur des variables globales critiques ; plusieurs tâches ne peuvent accéder à cette variable en même temps.

3.2 OBSW

L'On-Board Software (OBSW) d'OUFTI-1 est la partie software de l'OBC. Codé en langage C et exécuté au sein du système d'exploitation FreeRTOS, l'OBSW est responsable de la gestion du bon fonctionnement de la totalité du satellite ainsi que de la communication avec la station sol (GND). Cela est évidemment uniquement possible grâce à l'aide de la partie matérielle du sous-système OBC et du sous-système COM (cf. chapitre 2).

Grâce au système d'exploitation utilisé, l'OBSW peut être divisé en plusieurs modules, chacun constitué entre autres d'une tâche. La figure 3.2 représente les 6 modules de l'OBSW avec leurs rôles ainsi que la manière dont sont exécutés ces rôles ; soit par interruption, soit par leur tâche respective.

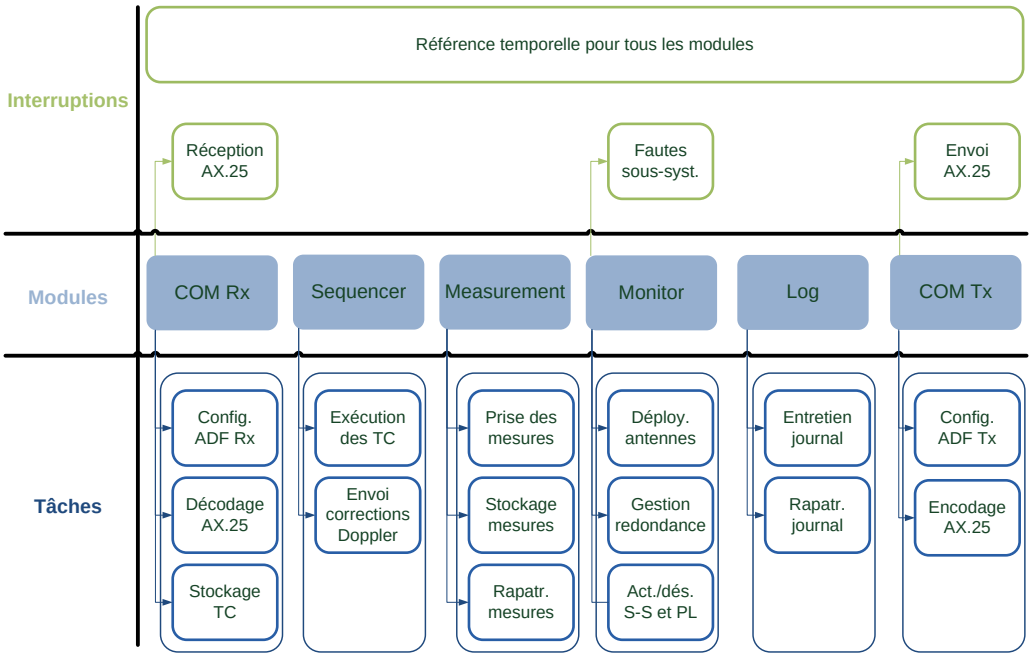

FIGURE 3.2 – Organisation de l'OBSW avec une tâche et leurs rôles par module, ainsi que leur routine d'interruption

Tout d'abord, une interruption est générée toutes les secondes grâce au Timer B du microcontrôleur MSP430 (cf. 2.2.2). Cette interruption va lancer une routine d'interruption dont le rôle est d'incrémenter un compteur. Ce

compteur va servir de **référence temporelle** à tout l'OBSW. Etant donné que ce compteur est codé sur 32 bits et qu'il est non signé, il peut atteindre une valeur de 2^{32}-1 secondes, ce qui correspond environ à 136 ans. La mission d'OUFTI-1 est programmée pour durer 1 an, donc si l'OBSW ne redémarre jamais, cette référence temporelle ne devra jamais être remise à zéro.

Le module **COM Rx** est responsable de la réception des télécommandes encodées en AX.25. Ce rôle est effectué au sein d'une routine d'interruption et est expliqué plus en détails dans le chapitre 3.3.2. Ce module est ensuite responsable de la configuration du récepteur ADF7021 et du décodage, de l'interprétation et du stockage des télécommandes reçues. Ces rôles sont gérés au sein d'une tâche COM Rx qui est expliquée dans le chapitre 3.3.3

Le module **Sequencer** consiste en une unique tâche responsable de l'exécution des télécommandes stockées par la tâche COM Rx ainsi que de l'envoi de corrections Doppler au sous-système COM. Ce module est détaillé dans le chapitre 3.4.

Le module **COM Tx**, à l'inverse du module COM Rx, est responsable de l'émission de télémétries encodées en AX.25. Egalement effectué au sein d'une routine d'interruption, ce rôle est expliqué dans le chapitre 3.5.3. La tâche liée à ce module gère le rôle d'encodage des télémétries en AX.25 ainsi que le rôle de configuration de l'ADF7021 d'émission et est expliqué dans le chapitre 3.5.2.

Le module **Measurement** est constitué d'une tâche responsable de la prise des mesures hardware et software de l'ensemble du satellite. La plupart des mesures sont prises sur des convertisseurs analogiques/digitaux 8 bits. La communication avec ces convertisseurs se fait grâce au bus I^2C. Ces mesures sont ensuite stockées dans l'EEPROM avec la référence temporelle correspondant au moment où elles ont été prises. Cette tâche est aussi responsable du rapatriement de ces mesures et de leur placement dans la partie Data d'une télémétrie afin que le module COM Tx puisse l'envoyer à la station sol. Un tableau en mémoire, contenant les fréquences d'échantillonnage de chaque mesure ainsi qu'un flag permettant de savoir s'il faut stocker cette mesure ou non, permet à la tâche d'échantillonner uniquement les mesures nécessaires et à la bonne fréquence. Ce tableau a une configuration initiale, mais peut être reconfiguré par télécommande, ce qui permet de passer d'une large vue de plusieurs mesures à basse fréquence à une vue précise d'une seule mesure à haute fréquence. Ce module est étudié en détails dans le travail de

fin d'études de Thomas Langohr [5].

Le module **Monitor** est constitué d'une tâche qui, dès le début de son exécution, va suspendre toutes les autres tâches (cf. chapitre 3.1.1), à l'exception de la tâche Log car le satellite ne doit entrer dans son fonctionnement nominal que 30 minutes après éjection du P-POD. La tâche Monitor sera donc maître à bord et attendra ces 30 minutes. Une fois ce délai écoulé, la tâche Monitor déploiera les antennes si le voltage des batteries est suffisant et réactivera les autres tâches. Ensuite, la tâche Monitor est responsable de la surveillance de certains paramètres critiques du satellite comme le voltage des batteries. Si celui-ci est trop faible, certains sous-systèmes devront être désactivés. La tâche ajoute également au Log les fautes survenues sur les autres sous-systèmes avant de les redémarrer. Celles-ci sont détectées grâce à des protections en courant qui, lors de surconsommations, font passer une entrée du microcontrôleur à zéro, ce qui va générer une interruption. Il y a donc une entrée par protection en courant [6]. Ce module est expliqué en détails dans le travail de fin d'études de Thomas Langohr [7].

Le module **Log** est responsable de l'enregistrement dans l'EEPROM des événements principaux survenant à bord du satellite comme le démarrage d'un des deux OBC, le déploiement des antennes ou encore l'activation/désactivation d'un sous-système. Ces événements sont enregistrés avec la référence temporelle correspondant au moment où ils sont survenus. Le module gère cela au sein d'une tâche qui est également responsable du rapatriement de ces événements afin qu'ils soient envoyés à la station sol par la tâche COM Tx. Ce module est expliqué en détails dans le travail de fin d'études de Thomas Langohr [8].

5. cf. [Lan11].
6. Les composants de protection de courant utilisés sont des MAX809L.
7. cf. [Lan11].
8. cf. [Lan11].

3.3 Module COM Rx

Le module COM Rx est responsable de la réception des télécommandes au travers de l'ADF7021 de réception. Celles-ci doivent être contrôlées (champs, intégrité, etc.) avant d'être ajoutées au bon endroit dans le tableau de commandes du module Sequencer. Ce chapitre explique les différents mécanismes mis en place afin que ce module puisse effectuer son rôle.

3.3.1 Tableau de télécommandes

Le tableau de télécommandes est un tableau à deux entrées stocké en RAM dont chaque entrée est un vecteur d'une longueur de 133 bytes. Ces vecteurs vont être remplis par la routine d'interruption COM Rx (cf. chapitre 3.3.2) et vont contenir chacun une trame AX.25. Une trame AX.25 a une longueur maximale de 276 bytes (cf. chapitre 4.1), mais comme la place disponible en RAM n'est que de 5 Kbytes, cette taille a dû être réduite. OUFTI-1 ne peut donc recevoir que des trames AX.25 de maximum 133 bytes.

Ces 133 bytes comprennent 20 bytes pour le préambule et la fin de la trame AX.25, 13 bytes pour le Primary Header, le Secondary Header et le Timestamp (cf. chapitre 4.2) ainsi que 100 bytes pour les paramètres. Ces 100 bytes de paramètre ne peuvent être complètement utilisés que pour l'envoi de corrections Doppler. Pour toutes les autres commandes, seuls 20 bytes de paramètres peuvent être utilisés (cf. chapitre 3.4.1).

Le tableau de télécommandes est composé de deux entrées pour la simple et bonne raison que, si deux trames AX.25 arrivent les unes à la suite des autres, elles puissent être placées dans des vecteurs différents. Néanmoins, il n'y a que deux espaces disponibles, c'est pourquoi la station sol doit envoyer ses télécommandes avec un délai d'une demi seconde entre chacune afin d'éviter une surcharge au niveau de l'OBSW, ce qui provoquerait la perte d'une ou de plusieurs télécommandes.

3.3.2 La routine d'interruption COM Rx

La routine d'interruption COM Rx a en réalité deux sources d'interruptions.

La première est la sortie SWD de l'ADF7021 de réception (cf. chapitre 2.3.1). Une fois l'interruption lancée, cela signifie qu'un mot de synchronisa-

tion a été détecté au sein de l'ADF7021 de réception, et donc que l'OBSW doit se tenir prêt à recevoir une trame AX.25. Pour ce faire, si l'interruption est lancée, il va simplement activer la seconde source d'interruptions.

La seconde source d'interruptions est la sortie TxRxCLK de l'ADF7021 de réception. Comme expliqué précédemment, cette source d'interruptions n'est activée que si le mot de synchronisation est détecté au sein de l'ADF7021 de réception. A chaque flanc montant de TxRxCLK, la routine d'interruption va être lancée et va aller lire le bit sur la sortie TxRxDATA. Chaque bit va être placé dans un des vecteurs du tableau de télécommandes et une fois la réception terminée (réception du flag de fin de l'AX.25), les interruptions provenant de TxRxCLK vont être désactivées.

Le décodage de l'AX.25 se fait également au sein de cette interruption. Chaque bit reçu va être décodé afin de reformer la même trame que celle construite à la station sol.

3.3.3 La tâche COM Rx

La figure 3.3 représente le travail effectué par la tâche COM Rx. Il s'agit d'une tâche cyclique, bouclant à l'infini. Le délai d'attente entre chaque tour de boucle est expliqué et argumenté dans le chapitre 3.7.

Tout d'abord, elle se charge de configurer l'ADF7021 de réception en AX.25 comme expliqué dans le chapitre 2.3.1. Une fois dans sa boucle, elle va effectuer une comparaison avec la référence temporelle du satellite afin de détecter si 20 minutes se sont écoulées depuis la dernière configuration de l'ADF7021 de réception. Si c'est le cas, elle va le reconfigurer car il a besoin d'être recalibré en fonction de sa température.

Ensuite, si la routine d'interruption a placé une trame AX.25 dans le tableau de télécommandes, la tâche va se charger de vérifier les différents champs de la trame AX.25 et va recalculer le CRC afin de contrôler que la trame n'a pas été altérée durant son transfert. Si un champ ou le calcul de CRC devait être incorrect, la tâche placera une télémétrie de type AX.25_FAIL (cf. chapitre 4.4.1 page 73) dans le tableau de télémétries du module COM Tx afin qu'elle soit envoyée.

Troisièmement, la tâche vérifie les champs de la trame Packet Utilization Standard (PUS) (cf. chapitre 4.2) et l'ajoute soit dans le tableau de corrections Doppler du module Sequencer s'il s'agit d'une commande de cor-

rections Doppler, soit dans le tableau de commandes du module Sequencer s'il s'agit d'une télécommande classique. Il est important de noter que seules les commandes de corrections Doppler peuvent atteindre une taille de 133 bytes car les corrections sont ajoutées dans un tableau spécifique. Les autres télécommandes qui sont ajoutées dans le tableau de commandes ne peuvent pas excéder la taille de 53 bytes en AX.25, elles ont donc seulement droit à 20 bytes pour les paramètres. Cela est dû au fait que le tableau de commandes du module Sequencer a une capacité de 20 commandes. Si elles pouvaient chacune contenir 100 bytes de paramètres, cela occuperait trop de place en RAM.

Dernièrement, si le champ ACK de la télécommande demande un acquittement pour l'acceptance de celle-ci, une télémétrie ACC_FAIL sera ajoutée au tableau de télémétries du module COM Tx en cas d'échec de l'ajout ou d'une erreur dans la trame PUS, et en cas de réussite une télémétrie ACC_SUCCESS sera ajoutée.

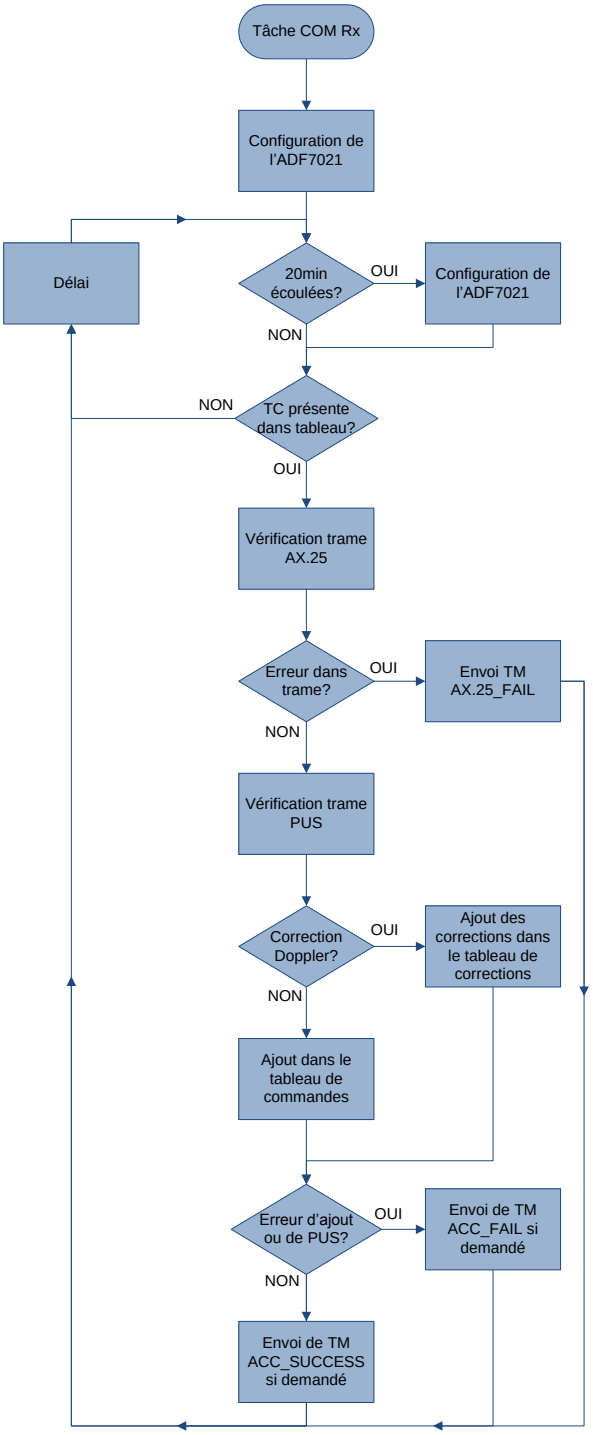

FIGURE 3.3 – Diagramme de la tâche COM Rx

3.4 Module Sequencer

Le module Sequencer est responsable de l'exécution des commandes reçues par le module COM Rx. Le moment où les différentes commandes doivent être exécutées dépend de leur Timestamp. Toute télécommande envoyée au satellite comprend un Timestamp relatif au moment de l'ajout de la commande dans le tableau de commandes. Lors de cet ajout le Timestamp reçu est donc additionné à la référence temporelle du satellite. Ainsi le module Sequencer connaît le moment absolu où doit être exécutée la commande. Par contre, si le Timestamp de la télécommande vaut zéro, il s'agit d'une commande à exécution directe, et aucune addition ne sera réalisée. Cela permet de distinguer les commandes à exécution directe (le Timestamp vaut zéro) des commandes à exécution reportée (le Timestamp est plus grand que zéro).

Un type particulier de télécommande doit également être géré par le module Sequencer : les corrections Doppler. Celles-ci sont envoyées comme toute autre commande, mais elles sont gérées différemment au sein de ce module. Les raisons de cette distinction seront expliquées au sein de ce chapitre.

Ce chapitre est donc consacré à l'explication des différents mécanismes mis en place pour que le module COM Rx puisse ajouter des commandes, pour gérer l'exécution de ces commandes ainsi que pour la gestion des corrections Doppler.

3.4.1 Tableau de commandes

Le tableau de commandes est alimenté par le module COM Rx et contient donc les commandes à exécuter à bord du satellite. Ce tableau est utilisé par le module Sequencer qui est responsable de l'exécution de ces commandes. Pour gérer le tableau de commandes en mémoire, il y a en réalité plusieurs tableaux : le tableau de commandes et le tableau d'emplacements vides.

Le tableau principal est le tableau de commandes lui-même. Alloué statiquement au démarrage de l'OBSW, ce tableau a une capacité de vingt commandes. Chaque commande est une version raccourcie de la trame PUS (cf. chapitre 4.2) originellement réceptionnée par le module COM Rx. Voici les différents champs mémorisés avec l'utilité de chacun :

- Le **Packet Id** (2 bytes) est utilisé pour les différents acquittements de commandes ;
- Le **Packet Sequence Control** (2 bytes) est mémorisé dans le même

but que le Packet Id. Ils forment à eux deux un identifiant unique pour chaque commande ;
- Le **Type** (1 byte) sert à identifier le service demandé par la commande ;
- Le **Sous-type** (1 byte) permet de connaître le sous-type de service demandé par la commande ;
- Le champ **ACK** (1 byte) est mémorisé afin de savoir quel type d'acquittement est demandé pour cette commande ;
- Le **Timestamp** (4 bytes) correspond au Timestamp de la commande additionné à la référence temporelle du satellite, il est utilisé pour savoir quand doit être exécutée la commande ;
- Le champ **PusError** (1 byte) ne fait pas partie de la trame PUS mais est utilisé lors de l'acquittement, afin de savoir si une erreur est survenue ou non ;
- Le champ **Params** (20 bytes) contient les paramètres de la trame PUS ;
- Le champ **nbParams** (1 byte) est utilisé afin de connaître le nombre effectif de bytes utilisés du champ Params ;
- Le champ **Next** (2 bytes) est utilisé afin de gérer la liste chaînée de commandes qui sera expliquée plus loin dans le chapitre.

Chaque entrée du tableau de commandes a une taille de 35 bytes, le tableau de commandes occupe donc 700 bytes en RAM.

Afin d'optimiser l'ajout dans ce tableau, un second tableau entre en jeu : le tableau d'emplacements vides. Celui-ci a le même nombre d'entrées que le tableau de commandes mais chacune ne contient qu'un pointeur vers un emplacement libre du tableau de commandes. Cela permet, lors de l'ajout, de ne pas parcourir la totalité du tableau de commandes à la recherche d'un emplacement vide. Ce tableau est en réalité une First In, First Out (FIFO) qui possède un compteur PUSH et un compteur POP. Concrètement, pour le premier ajout, on prendra le pointeur du premier élément de la FIFO, on ajoutera la commande à l'emplacement désigné par ce pointeur et on incrémentera la valeur de POP. Ainsi au prochain ajout on accédera directement à l'emplacement libre suivant grâce au compteur POP. La suppression d'une télécommande effectue l'opération inverse, c'est-à-dire qu'elle accède à l'emplacement correspondant au compteur PUSH et ajoute le pointeur pointant vers l'emplacement précédemment occupé par la commande et incrémente finalement le compteur PUSH. Evidemment, lorsque l'un des deux compteurs atteint la fin du tableau, sa valeur est réinitialisée afin de pointer à nouveau le premier élément du tableau. Une troisième variable comprenant le nombre de commandes dans le tableau est incrémentée à chaque ajout et décrémentée à chaque suppression afin d'éviter un surplus de commandes.

La figure 3.4 illustre un exemple simple de configuration possible des deux tableaux.

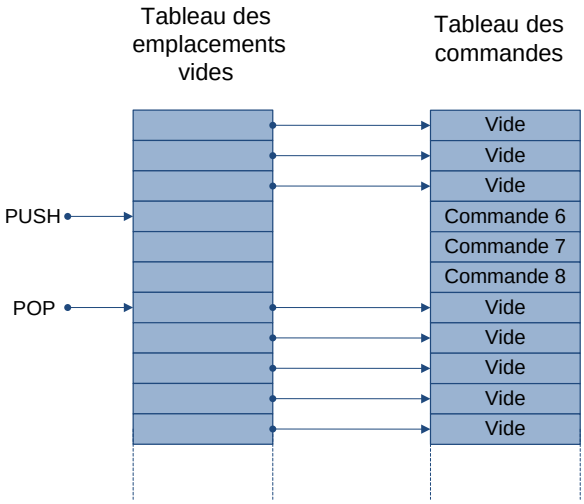

FIGURE 3.4 – Tableau des emplacements vides dans le tableau de commandes

A ce stade, l'ajout dans le tableau de commandes est optimisé. Seulement, chaque commande est accompagnée d'un Timestamp correspondant au moment où doit être exécutée la commande. Ce Timestamp vaudra 0 s'il s'agit d'une commande à exécution immédiate. S'il s'agit d'une commande à exécution reportée, le Timestamp original de la télécommande sera additionné à la référence temporelle du satellite au moment de l'ajout de la commande dans le tableau de commandes. Ainsi la tâche Sequencer est capable de savoir si une commande doit être exécutée immédiatement, ou si son exécution est reportée. Dans ce dernier cas, il est également capable, en fonction de la référence temporelle, de savoir quand doit s'exécuter cette commande. Mais pour trouver une commande dont le Timestamp est expiré, la tâche Sequencer devra dans le pire des cas parcourir la totalité du tableau de commandes. Afin d'éviter cela, un second mécanisme est mis en place : une liste chaînée triée unidirectionnelle de commandes.

Afin d'implémenter cette liste chaînée triée unidirectionnelle, un pointeur vers la première commande à exécuter dans le tableau est initialisé à NULL. Une fois une commande ajoutée dans le tableau, ce pointeur va pointer vers cette commande. Etant donné que chaque commande contient un pointeur vers la commande suivante à exécuter, il est possible de réaliser une liste de commandes chaînée et triée. A chaque ajout, la liste va être parcourue afin de

connaître l'emplacement ou doit être mise la nouvelle commande en fonction des Timestamps de chacune. Une fois l'emplacement trouvé, le pointeur de la commande précédente va être modifié pour pointer vers la nouvelle commande, et le pointeur de la nouvelle commande va pointer vers la commande suivante. Pour l'exécution, la tâche Sequencer devra uniquement vérifier si la première commande de la liste doit être exécutée ou non. Une fois la commande exécutée, la tâche ajoutera le pointeur de cette commande dans le tableau des emplacements vides et modifiera le pointeur vers la première commande afin qu'il pointe vers la commande suivante à exécuter.

La figure 3.5 représente une configuration possible du tableau de commandes en RAM, où chaque flèche correspond à un pointeur. Le tableau d'emplacements vides n'est pas représenté sur cette figure.

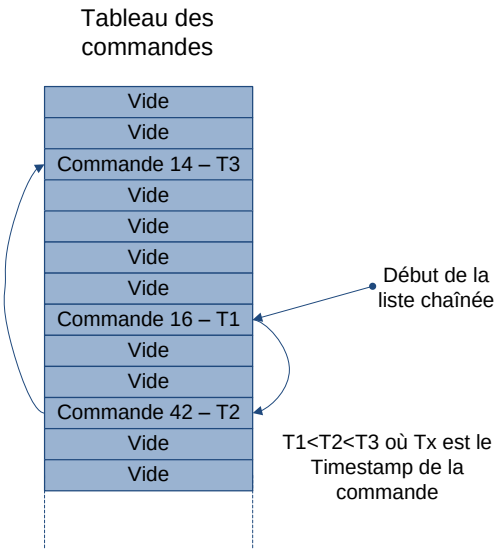

FIGURE 3.5 – Tableau des commandes avec liste chaînée triée

3.4.2 Tableau de corrections Doppler

Dans un premier temps, les corrections Doppler devaient prendre place dans le tableau de commandes, comme toute autre commande. Seulement, l'envoi d'une table de corrections Doppler de la station sol au satellite aurait nécessité environ 85 emplacements dans le tableau de commandes. Si cette solution n'avait pas été changée, le tableau de commandes aurait dû occuper 2975 bytes en RAM et le tableau d'emplacements vides 170 bytes. Le

MSP430F1612 utilisé possède une RAM de seulement 5 Kbytes, cette solution aurait donc occupé plus de 60% de l'espace disponible.

La solution consiste donc à réserver un espace plus restreint dédié uniquement aux corrections Doppler : un tableau de 85 entrées ayant chacune une taille de 2 bytes. Chaque entrée est constituée d'un Timestamp codé sur 1 byte ainsi que la correction également codée sur 1 byte. Cela permet une économie de plus de 2200 bytes en mémoire.

Du point de vue du fonctionnement, lorsqu'une correction Doppler doit être effectuée, elle est envoyée au microcontrôleur COM qui va lui se charger d'appliquer la correction. Cet échange d'informations se fait via un USART du MSP430F1612 configuré en UART (cf. chapitre 2.2.2).

Lorsque la tâche COM Rx reçoit une table de corrections Doppler par télécommande, elle va placer dans une variable que nous appellerons TDoppler la somme du Timestamp de la télécommande, de la référence temporelle du satellite et du Timestamp de la première correction Doppler. Ainsi en analysant cette variable la tâche Sequencer saura si la première correction doit être envoyée ou non. Une fois la première correction envoyée, la tâche Sequencer va ajouter à la variable TDoppler le Timestamp de la correction suivante. Chaque Timestamp de correction est donc relatif au temps d'exécution de la correction précédente, et le Timestamp de la première correction est relatif au Timestamp général de la télécommande.

Ce mécanisme de temps relatifs permet également d'économiser de la place, car si le Timestamp de chaque correction était relatif au Timestamp général de la télécommande, le Timestamp de chaque correction aurait dû être codé sur 2 bytes et non sur 1.

La tâche Sequencer gère un compteur de correction courante et un compteur de dernière correction afin de savoir quel est l'index de la prochaine correction à envoyer ainsi que l'index de la dernière correction à envoyer.

3.4.3 La tâche Sequencer

La figure 3.6 de la page 41 représente le travail effectué par la tâche Sequencer. Il s'agit d'une tâche cyclique, bouclant à l'infini. Le délai d'attente entre chaque tour de boucle est expliqué et argumenté au chapitre 3.7.

Par tour de boucle, la tâche Sequencer va soit s'occuper de l'envoi d'une

correction Doppler, soit d'une commande quelconque présente dans le tableau de commandes.

Les actions prioritaires sont les corrections Doppler, car le moment où elles sont envoyées doit être le plus précis possible par rapport au Timestamp de la correction. Pour ce faire, la tâche Sequencer va comparer la variable TDoppler (cf. chapitre 3.4.2), qui contient le temps absolu de la prochaine correction à envoyer, avec la référence temporelle du satellite. Si la variable TDoppler est plus petite ou égale à la référence temporelle du satellite, cela signifie que la prochaine correction doit être envoyée. Une fois la correction envoyée, la tâche va ajouter à la variable TDoppler le Timestamp relatif de la prochaine correction à envoyer. Cela permet de faire la comparaison sur la bonne valeur de TDoppler au prochain tour de boucle.

Si la prochaine correction Doppler ne doit pas être envoyée immédiatement, la tâche Scheduler va vérifier si une commande doit être exécutée. Plusieurs conditions entrent en jeu pour définir si une commande doit être exécutée ou non. Premièrement, si le Timestamp de la prochaine commande à exécuter vaut zéro, elle doit être exécutée. Ensuite, si le Timestamp de la prochaine commande à exécuter est plus grand que zéro, que ce Timestamp est plus petit ou égal à la référence temporelle du satellite et que l'exécution des commandes à exécution reportée est activée (cf. chapitre 4.3.4), elle doit être exécutée. Par contre, si le Timestamp de la prochaine commande à exécuter est plus grand que zéro mais que l'exécution des commandes à exécution reportée est désactivée (cf. chapitre 4.3.4), elle ne doit pas être exécutée.

Pour rappel, il suffit à la tâche Sequencer de consulter le pointeur vers la prochaine commande à exécuter ; elle n'a pas besoin de parcourir la totalité du tableau de commandes (cf. chapitre 3.4.1).

Avant l'exécution de la commande, la tâche Sequencer va vérifier si un acquittement de début d'exécution est demandé pour cette commande (cf. chapitre 4.2.2). S'il est demandé et qu'aucune erreur n'est présente dans la commande (type, sous-type et paramètres valides), la tâche enverra une télémétrie de type START_SUCCESS (cf. chapitre 4.4.1). Par contre, si une erreur est présente, elle enverra une télémétrie de type START_FAIL (cf. chapitre 4.4.1) si l'acquittement est demandé. En cas d'erreur, la commande ne sera évidemment pas exécutée.

Suivant le type de commande à exécuter, la tâche Sequencer va soit ef-

fectuer le travail elle-même, soit déléguer le travail à la tâche concernée par cette commande. Par exemple, les commandes de gestion du Sequencer, de paramétrisation des mesures ou encore de récupération du mode de fonctionnement courant d'OUFTI-1 sont des commandes exécutées directement par le Sequencer, tandis que les commandes de rapatriement du journal de bord ou de rapatriement des mesures sont déléguées respectivement au module Log et au module Measurement au travers de variables globales et de flags.

Ensuite la tâche Sequencer va acquitter la fin de l'exécution de la commande si ce type d'acquittement a été demandé (cf. chapitre 4.2.2). Si l'exécution s'est déroulée correctement et sans erreurs, elle enverra une télémétrie de type END_SUCCESS (cf. chapitre 4.4.1) tandis que si une erreur est survenue durant l'exécution de la commande, elle enverra une télémétrie de type END_FAIL (cf. chapitre 4.4.1).

Enfin, elle ajoutera le pointeur de la commande précédemment interprétée dans le tableau d'emplacements vides grâce au compteur PUSH avant de l'incrémenter et mettra à jour le pointeur vers la prochaine commande à exécuter. La nouvelle valeur de ce pointeur sera alors le champ Next de la commande précédemment exécutée. Après avoir mis à jour ces champs, elle décrémentera le compteur de commandes présentes dans le tableau de commandes.

Tous les accès aux différents tableaux sont protégés à l'aide de sémaphores binaires afin d'éviter les accès concurrents (cf. chapitre 3.1.2).

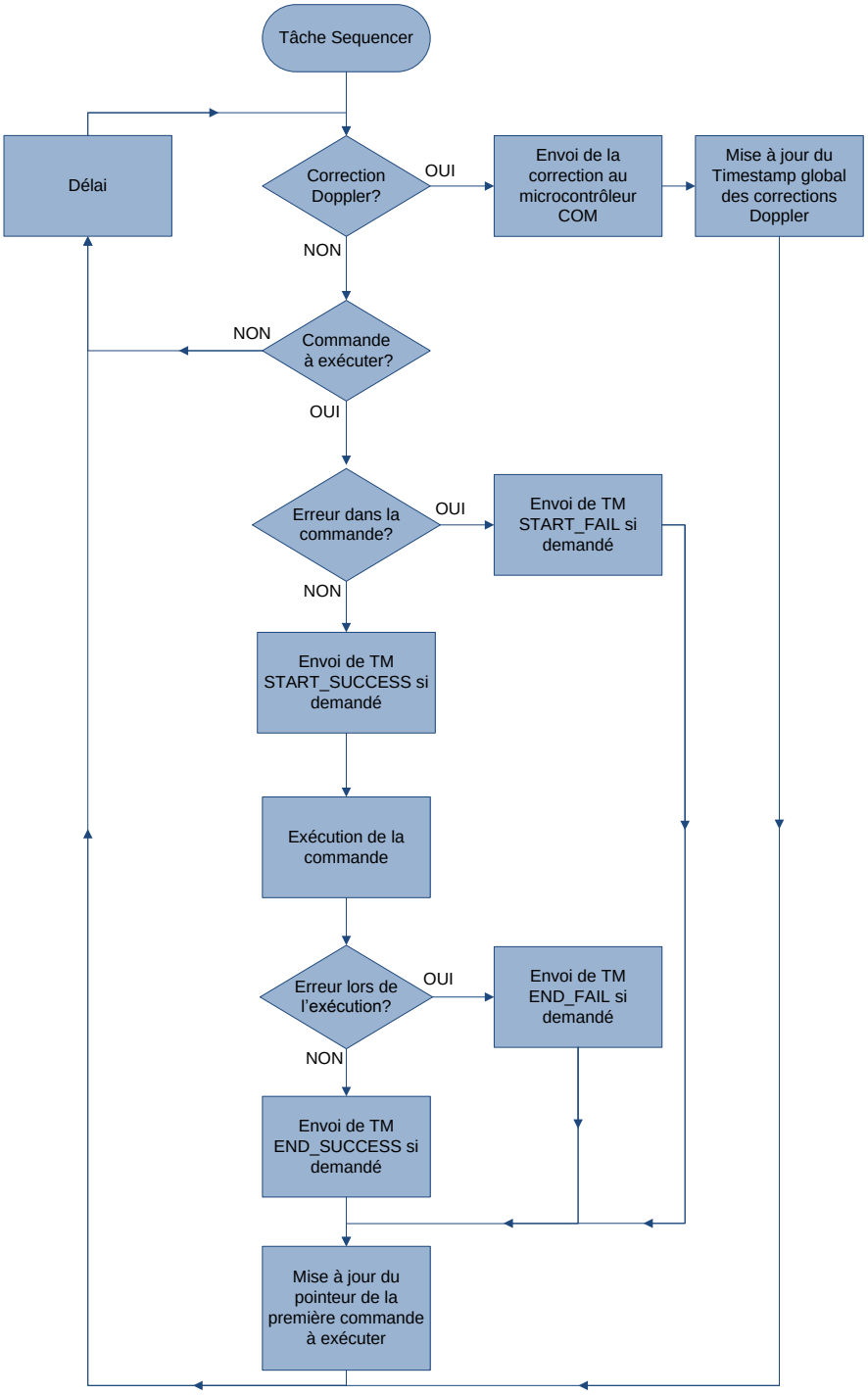

FIGURE 3.6 – Diagramme de la tâche Sequencer

3.5 Module COM Tx

Le module COM Tx est responsable de la mise en forme, de l'encodage et de l'envoi des trames AX.25 à l'ADF7021 d'émission. Il doit également gérer le fait que plusieurs modules peuvent vouloir envoyer une télémétrie en même temps. Ce chapitre est donc consacré à expliquer les mécanismes mis en place afin de gérer les différents envois de télémétries, ainsi que la configuration de l'ADF7021 d'émission.

3.5.1 Tableau de télémétries

Le module COM Tx est notamment composé d'un tableau destiné à contenir les télémétries à envoyer. Il s'agit d'un tableau à deux entrées dont chacune possède les champs suivants :

- Un champ **Reserved** (1 byte) utilisé pour déterminer si l'entrée est actuellement utilisée par un module ou non. Ce champ vaudra zéro si l'entrée n'est pas utilisée et il vaudra 1 si elle l'est ;

- Un champ **Finished** (1 byte) utilisé afin que le module COM Tx sache si la trame est prête à être envoyée ou non. Ce champ vaudra zéro si la trame n'est pas prête et donc si le module utilisant cette entrée n'a pas fini d'y insérer des données. Si le module a fini et désire envoyer la trame, ce champ vaudra 1 ;

- Un vecteur **FrameToSend** (135 bytes) contenant la trame AX.25 à envoyer. Il contient 20 bytes pour le préambule et la fin de trame AX.25, 15 bytes pour le Primary et le Secondary Header de la trame PUS ainsi que 100 bytes pour les paramètres de la télémétrie ;

- Un champ **DataLength** (1 byte) représentant le nombre de bytes placés dans la partie Data de la trame PUS contenue elle-même dans la trame AX.25. Le champ Data étant la seule partie de taille variable de la trame, il est utilisé par le module COM Tx afin de déduire la longueur de la trame à envoyer. Cela permet de ne pas envoyer la totalité du champ FrameToSend si ce n'est pas nécessaire.

Sur la base de ces champs, voyons comment se déroulent les opérations d'ajout, d'envoi et de retrait de ces trames.

Premièrement, lorsqu'un module désire envoyer une télémétrie, il doit réserver une entrée dans le tableau de télémétries. Pour ce faire, il fait appel à une méthode du module COM Tx qui parcourt le tableau de télémétries à la recherche d'une entrée dont le champ Reserved est à zéro. Si elle trouve cette entrée, elle positionne le champ Reserved à 1 et le champ Finished à zéro et enfin retourne l'index de l'entrée réservée. Ainsi le module ayant fait la demande de réservation connaît l'emplacement de l'entrée qui lui est réservée. Par contre, si toutes les entrées sont réservées, la méthode retournera -1. Le module ayant fait la demande saura donc qu'aucune entrée n'est disponible. Il décidera donc soit de réitérer sa demande, soit de ne pas envoyer la télémétrie. Tout ce qui est exécuté dans la méthode de réservation est protégé par sémaphore binaire, ainsi il est impossible que deux modules puissent réserver la même entrée en même temps.

En cas de succès de réservation, le module peut alors remplir la partie PUS Data (cf. chapitre 4.2) du champ FrameToSend. Une fois qu'il a fini de placer ses données, il place dans le champ DataLength le nombre de bytes qu'il a précédemment placé dans les paramètres, et notifie qu'il a fini son travail en positionnant le champ Finished à 1. Le travail du module désirant envoyer une télémétrie s'arrête là ; il n'a plus à se préoccuper de la télémétrie, pour lui, elle est envoyée.

La prochaine étape de l'envoi est effectuée au sein de la tâche COM Tx et est donc expliquée dans le chapitre 3.5.2.

3.5.2 La tâche COM Tx

La figure 3.7 de la page 45 représente le travail effectué par la tâche COM Tx. Il s'agit d'une tâche cyclique, bouclant à l'infini. Le délai d'attente entre chaque tour de boucle est expliqué et argumenté au chapitre 3.7.
Premièrement, la tâche COM Tx va vérifier qu'un envoi n'est pas en cours. Pour ce faire, une variable est placée à 1 si aucun envoi n'est en cours et à zéro si un envoi est en cours.

Ensuite, si aucun envoi n'est en cours, elle va parcourir le tableau de télémétries à la recherche d'une entrée dont le champ Finished est à 1. Si c'est le cas, c'est que cette télémétrie doit être envoyée. Elle construit alors le Primary et le Secondary Header de la trame PUS avant les données présentes dans le champ FrameToSend, et place autour de la trame PUS ainsi obtenue le préambule et la fin de trame AX.25. Le champ FrameToSend est donc prêt à être envoyé, et est délimité par les champs Flag de la trame AX.25 (cf.

chapitre 4.1).

Troisièmement, elle place l'index de la commande à envoyer dans une variable globale afin qu'il soit accessible depuis la routine d'interruption COM Tx et configure l'ADF7021 d'émission comme expliqué dans le chapitre 2.3.1. Contrairement à la tâche COM Rx qui reconfigure l'ADF7021 de réception toutes les 20 minutes, la tâche COM Tx reconfigure l'émetteur ADF7021 avant chaque envoi car il est aussi utilisé par le sous-système COM pour émettre en D-STAR. On s'assure ainsi que l'ADF7021 est correctement configuré.

Finalement, elle positionne la variable indiquant qu'un envoi est en cours à zéro et active les interruptions sur la sortie TxRxCLK de l'ADF7021. Ainsi la routine d'interruption COM Tx sera exécutée à 9600 Hz et elle enverra un bit à chacune de ses exécutions. Le fonctionnement de la routine d'interruption est exposé dans le chapitre 3.5.3.

3.5.3 La routine d'interruption COM Tx

La routine d'interruption est exécutée à chaque flanc descendant de la sortie TxRxCLK de l'ADF7021 d'émission (cf. chapitre 2.3.1) uniquement si la tâche COM Tx a activé cette interruption. Si une interruption est lancée, cela signifie donc obligatoirement qu'une trame est prête à être envoyée. La routine d'interruption connaît l'index de la trame à envoyer du tableau de télémétries car il a été copié dans une variable globale par la tâche COM Tx. Elle accède donc à la bonne télémétrie à envoyer.

A chaque exécution, la routine d'interruption va encoder un bit de la trame à envoyer et le placer sur l'entrée TxRxDATA de l'ADF7021 d'émission.

Les champs Flag de début et de fin de la trame AX.25 étant présents dans la trame à envoyer, la routine d'interruption peut facilement détecter la fin de la trame. Une fois détectée, elle va désactiver les interruptions générées par TxRxCLK, positionner la variable globale indiquant qu'un envoi est en cours à 1 et va finalement placer le champ Reserved de la trame précédemment envoyée à zéro.

La désactivation des interruptions de la sortie TxRxCLK de l'ADF7021 d'émission permet de ne plus être interrompu une fois la télémétrie envoyée.

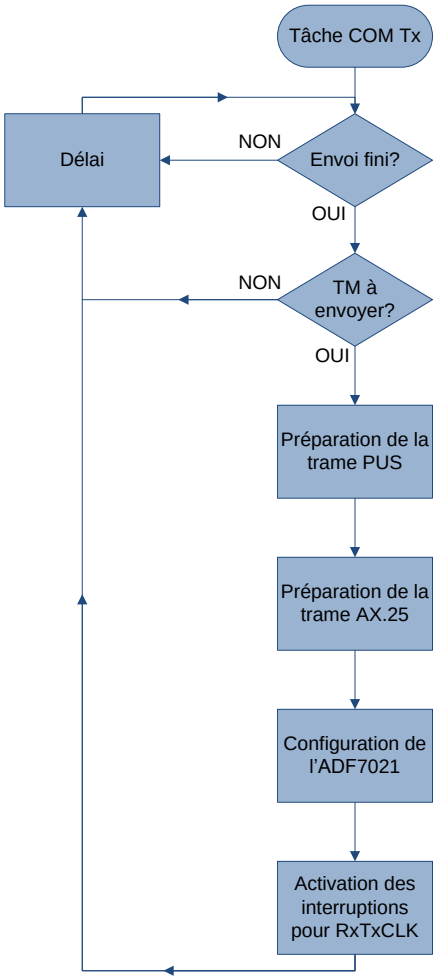

FIGURE 3.7 – Diagramme de la tâche COM Tx

La variable globale indiquant qu'un envoi est en cours est positionné à 1, ce qui signifie que l'envoi est terminé. La tâche COM Tx peut ainsi s'occuper de la télémétrie suivante s'il y en a une.

Mettre le champ Reserved à zéro libère l'emplacement précédemment occupé par la télémétrie, ce qui permet à d'autres modules de réserver cet emplacement.

3.6 Méthode de test

3.6.1 Simulation de l'ADF7021

Afin de pouvoir tester la gestion des télécommandes et des télémétries, l'idéal sera d'interconnecter la carte OBC et la carte COM. Afin de recevoir des données à l'entrée de l'ADF7021 de réception et de pouvoir émettre des données depuis l'ADF7021 d'émission, le support des antennes et les antennes devraient également être installés. Seulement, la carte COM est toujours un prototype et l'interconnexion n'est pas possible. Les moyens à mettre en œuvre pour tester l'OBC en situation réelle sont trop importants et cette solution n'a donc pas été envisagée à ce stade de l'implémentation.

La solution consiste donc à simuler l'ADF7021 en le remplaçant par une carte OBC identique à celle utilisée pour l'OBSW que nous appellerons l'OBC-ADF. Cet OBC-ADF simule à la fois l'ADF7021 d'émission et l'ADF7021 de réception.

L'OBC-ADF envoie des trames AX.25 à l'OBC via deux broches, une pour l'horloge et une pour les données, exactement comme le ferait l'ADF7021 de réception. Le signal de détection de mot de synchronisation est également implémenté à bord de l'OBC-ADF. Il génère donc une impulsion sur une troisième broche afin que le module COM Rx de l'OBC puisse activer les interruptions provenant de l'horloge de l'OBC-ADF. Les données sont envoyées de OBC-ADF vers OBC à 9600 bits par seconde. La seule chose que OBC-ADF ne simule pas est la configuration.

L'OBC-ADF simule également l'ADF7021 d'émission. Pour ce faire il génère une horloge sur une sortie du MSP430F1612 à 9600 Hz. A chaque flanc montant de l'horloge il va aller lire la donnée sur une entrée du MSP430F1612 comme le ferait l'ADF7021 d'émission. Il va ensuite attendre le champ Flag de début d'une trame AX.25 et enregistrer chaque bit de donnée jusqu'à réception du Flag de fin de la trame AX.25.

Grâce à la carte de développement fournie avec le CubeSat Kit de chez Pumpkin, qui inclut la structure du satellite et la carte FM430, il est possible de communiquer avec cet OBC-ADF via UART depuis un ordinateur et cela via un câble USB. L'ordinateur détecte alors la carte de développement comme un port série et il est possible d'envoyer des bytes au microcontrôleur MSP430F1612 de l'OBC-ADF et également d'en recevoir. Il existe plusieurs applications HyperTerminal qui permettent d'afficher les bytes reçus sur un

certain port série et offrent également la possibilité d'envoyer des bytes via ce même port série. Seulement, tout est affiché en caractères en fonction des codes ASCII. Par exemple, si l'on reçoit un byte d'une valeur de 65, l'application HyperTerminal affichera le caractère 'A' qui correspond à 65 en ASCII. Cela n'est pas très pratique quand il s'agit d'envoyer une télécommande complète. C'est pourquoi une application dédiée aux tests a été développée : OUFTI-1 HyperTerminal. Cette dernière est expliquée en détails dans le chapitre 3.6.2.

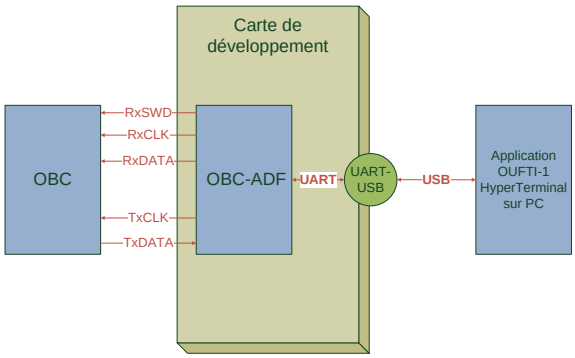

FIGURE 3.8 – Schéma de la simulation de l'ADF

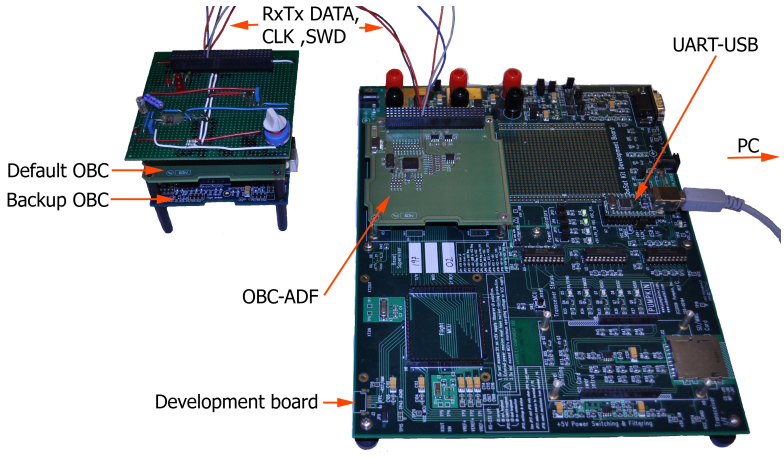

FIGURE 3.9 – Photo de la simulation de l'ADF

La figure 3.8 schématise la simulation de l'OBC-ADF et la figure 3.9 est une photo représentant les différentes cartes présentes sur la figure 3.8.

3.6.2 L'application OUFTI-1 HyperTerminal

L'application OUFTI-1 HyperTerminal a été développée dans le but de pouvoir aisément tester une fonctionnalité du satellite en envoyant une télécommande. Le retour du satellite est également géré, ce qui permet d'afficher clairement chaque télémétrie reçue.

L'application est codée en Java et utilise le package RxTx qui permet de facilement se connecter à un port série et d'envoyer des bytes à celui-ci ou d'en recevoir. La figure 3.10 représente l'interface graphique proposée par cette application. Elle est divisée en trois parties. La première est utilisée pour choisir le port auquel l'on désire se connecter ainsi que pour envoyer du texte en clair à l'OBC-ADF, ce qui était utilisé dans les premières phases de test. La seconde partie de la fenêtre permet d'envoyer des télécommandes. L'utilisateur n'a donc pas besoin de se soucier de la manière dont sera construite la commande, il lui suffit de choisir le nom de la commande à envoyer (le type et sous-type se modifient en fonction de la commande choisie), de sélectionner le type d'acquittement désiré ainsi que le Timestamp de la commande qui correspond au nombre de secondes qui doivent s'écouler entre la réception de la commande et son exécution. Les paramètres varient en fonction de la commande choisie, sur la figure 3.10 la commande choisie permet de récupérer des mesures prises à bord du satellite, les paramètres sont donc bien liés à cette commande (cf. chapitre 4.3.1). La troisième et dernière partie de la fenêtre est utilisée afin d'afficher les télémétries reçues à l'utilisateur, ainsi que du simple texte.

Du point de vue du fonctionnement, un thread se charge d'écouter en boucle sur la partie réception du port série. Dès qu'il reçoit un byte, il le traite. S'il s'agit d'un byte quelconque, il va afficher le caractère correspondant dans la boîte de texte de l'interface graphique de l'application. Cela permet à l'OBC-ADF d'envoyer des traces quelconques en texte mais cette fonctionnalité est disponible sur n'importe quelle application HyperTerminal. Par contre, si le byte reçu est un flag dont la valeur a été définie, l'application sait qu'elle doit s'attendre à recevoir une télémétrie. Elle va alors mémoriser tous les bytes reçus jusqu'à réception du flag de fin.

La trame que reçoit l'application est uniquement la partie PUS de la télémétrie AX.25 envoyée par l'OBC. En effet, à la réception d'une télémétrie, l'OBC-ADF va décoder la trame AX.25, vérifier les différents champs de cette trame, et va ensuite envoyer la partie PUS de la trame via UART. Elle décapsule donc la trame avant de l'envoyer.

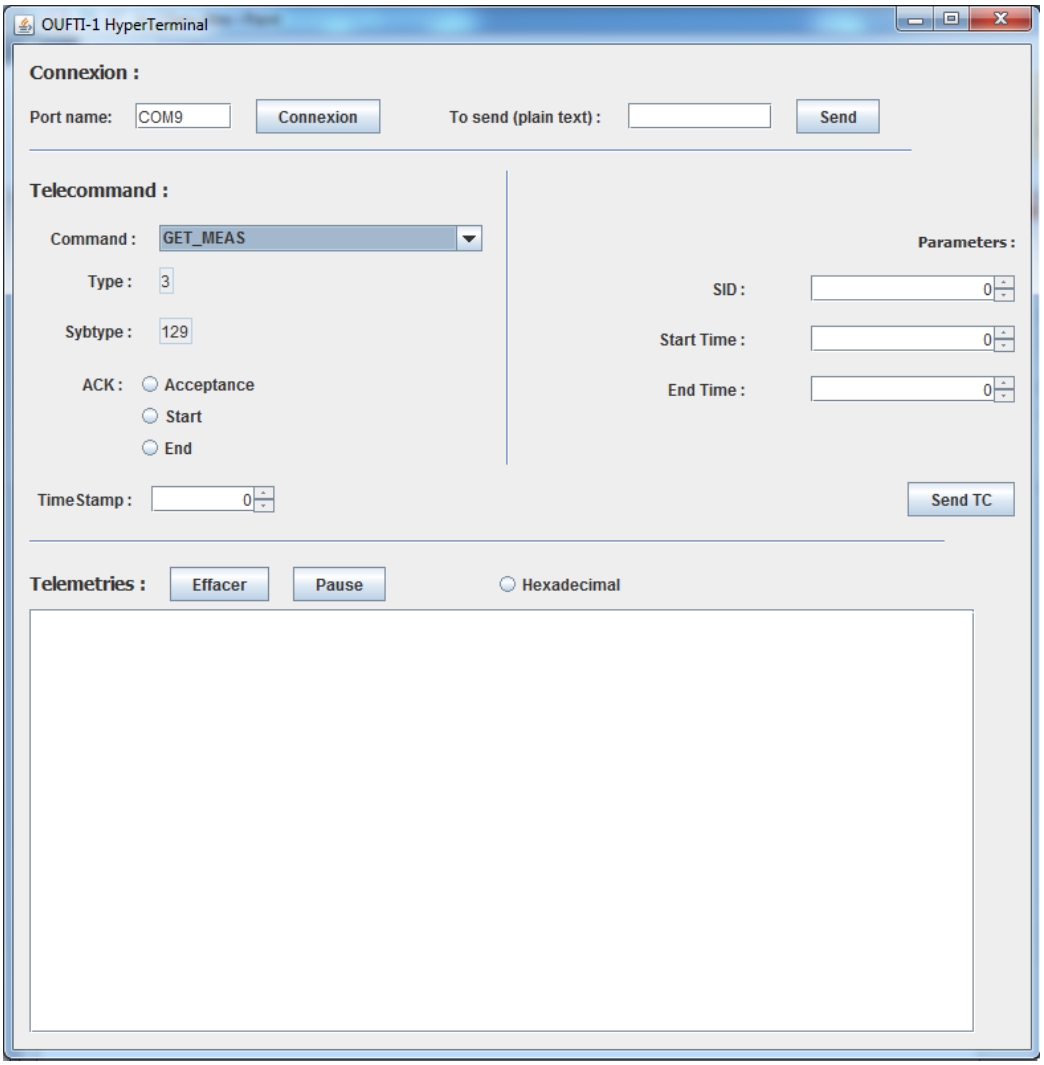

FIGURE 3.10 – OUFTI-1 HyperTerminal - L'interface graphique

Une fois cette trame PUS mémorisée dans l'application HyperTerminal, son contenu va être analysé afin de détecter de potentielles erreurs et la trame va être affichée dans la partie réception de l'interface graphique. Mais ce qui est intéressant, c'est que ce ne sont pas les valeurs brutes de la trame qui sont affichées, mais une interprétation de ces données, ce qui permet d'afficher un message clair à l'utilisateur de l'application.

La figure 3.11 est un exemple de télémétrie que l'application pourrait recevoir. Il s'agit d'une télémétrie qui contient le mode de fonctionnement

courant d'OUFTI-1. On peut donc voir qu'au temps 36 (36 secondes après le démarrage de l'OBSW), le satellite est en mode de fonctionnement par défaut.

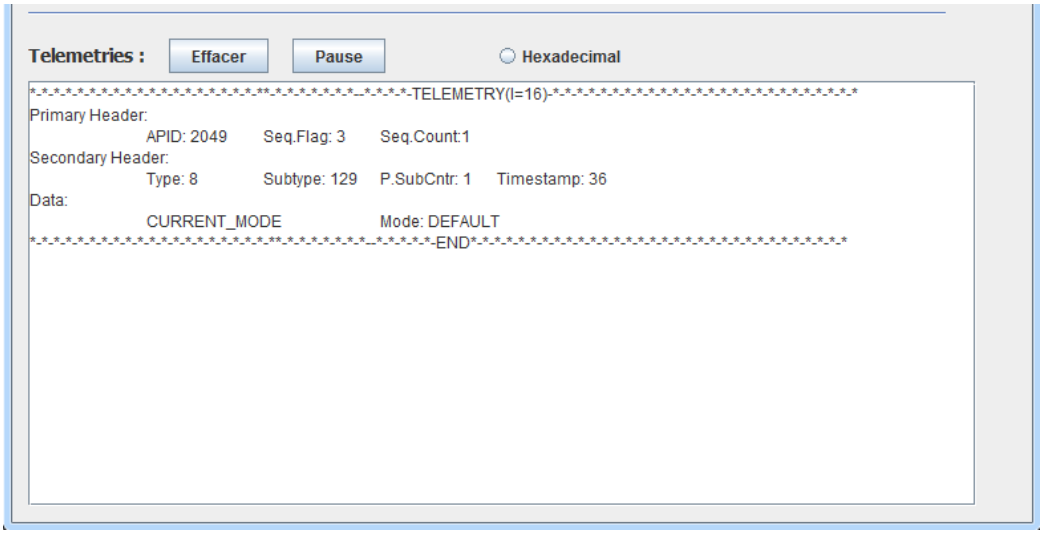

FIGURE 3.11 – OUFTI-1 HyperTerminal - Réception d'une télémétrie

Pour l'envoi de télécommandes, l'application va mettre en forme une trame PUS lors de l'appui sur le bouton 'Send TC'. Avant d'effectuer l'envoi elle va évidemment vérifier les différents champs entrés par l'utilisateur et afficher un message d'erreur si nécessaire. L'OBC-ADF va recevoir cette trame PUS, la placer au sein d'une trame AX.25 et l'envoyer à l'OBC codée.

Du point de vue de l'OBC, tous ces mécanismes sont transparents, il travaille donc exactement de la même manière que si la carte COM était présente et qu'il envoyait des télémétries à l'ADF7021 d'émission et recevait des télécommandes de l'ADF7021 de réception. Du point de vue du développeur, cela permet de tester toutes les fonctionnalités codées au sein de l'OBSW et d'avoir un retour facilement interprétable.

3.6.3 Les résultats

Grâce à ce système de test, il est possible de tester les différentes télécommandes et télémétries et de se rendre compte qu'elles sont gérées correctement au sein de l'OBSW. Les mesures et le log sont rapatriés correctement, le mode du satellite peut être récupéré, les télécommandes sont acquittées

quand il le faut, etc.

Néanmoins cela n'apporte rien quant au comportement dynamique des différentes tâches entre elles. Il n'y a pas de données précises quant au temps d'exécution nécessaire pour effectuer ces différentes tâches. C'est pourquoi le test a été poussé un peu plus loin et est expliqué en détails dans le chapitre 3.8.

3.7 Allocation mémoire

Ce chapitre est consacré à définir la taille du Stack en mémoire pour chaque tâche.

Le Stack de chaque tâche est utilisé afin de sauvegarder les différents registres lorsque la tâche fait par exemple appel à une fonction. Sa taille peut être définie lors de la création de la tâche. Afin de pouvoir consulter l'espace réellement occupé dans ce Stack par une tâche quelconque, il existe un méthode fournie par FreeRTOS (uxTaskGetHighWaterMark()) qui permet de connaître l'espace libre dans le Stack d'une tâche. La taille d'un Stack est exprimée en Words. La taille d'un Word dépend du microcontrôleur utilisé. Le MSP430F1612 étant un microcontrôleur 16 bits, un Word aura une taille de 2 bytes.

3.7.1 Méthode de test

Afin de pouvoir tester l'espace occupé par le Stack de chaque tâche, une valeur par défaut a été choisie pour la taille de Stack de chaque tâche. Ensuite, le module Measurement se charge de récupérer l'espace restant dans le Stack de chaque tâche grâce à la méthode uxTaskGetHighWaterMark(). Le module Measurement s'occupe de ces données comme d'autres mesures et les sauvegarde donc dans l'EEPROM.

Afin de pouvoir récupérer ces différentes mesures, il suffit alors d'envoyer une télécommande de récupération de mesure, et d'analyser les données récupérées par télémétrie.

3.7.2 Exploitation des résultats

Après avoir envoyé des télécommandes de demande de mesure pour chaque tâche, toutes les données nécessaires pour définir l'espace réellement occupé

par le Stack de chaque tâche sont disponibles.

Ces données correspondent à l'espace disponible dans le Stack de chaque tâche. Cela permet donc de réduire la taille de Stack de chaque tâche, en laissant une marge de 20 Words. Ainsi, l'espace utilisé en mémoire est réduit au maximum.

3.8 Comportement dynamique

Comme expliqué dans le chapitre 3.1.1, les différentes tâches formant l'OBSW peuvent avoir des priorités différentes. Ce chapitre est destiné à définir les priorités de chaque tâche, ainsi qu'à expliquer les différents temps d'exécution de chacune des tâches.

3.8.1 Temps d'exécution

Toutes les tâches sont des tâches cycliques. Elles vont donc s'exécuter en boucle avec un temps d'attente entre chaque boucle. Afin que le temps entre deux boucles soit toujours le même, FreeRTOS propose une fonction (vTaskDelayUntil()) qui permet d'attendre jusqu'à l'arrivée à un certain nombre de ticks. Un tick correspond dans le cas de la configuration de FreeRTOS pour OUFTI-1 à 1 ms. Par exemple, si le dernier tour de boucle a démarré au tick 500 et que le temps désiré d'attente est de 250 ticks, le prochain tour de boucle commencera au tick 750. Cela permet d'assurer que la tâche commencera à s'exécuter tous les 250 ticks pour autant que le temps d'exécution de la boucle ne dépasse pas ces 250 ms.

Afin de mesurer les différents temps d'exécution de ces boucles, FreeRTOS permet de récupérer le nombre de ticks courant à l'aide de la fonction xTaskGetTickCount(). Ainsi en récupérant cette valeur au début et à la fin de l'exécution d'un tour de boucle il est possible de connaître le temps nécessaire en ms afin d'exécuter cette boucle.

Les valeurs sont récupérées exactement comme pour le chapitre 3.7 ; chaque tâche réserve une télémétrie qui est envoyée via l'OBC-ADF et récupérée sur l'application OUFTI-1 HyperTerminal.

La tâche COM Rx et COM Tx effectuent leur tour de boucle entre 1 et 2 ms. La tâche Sequencer en moins d'1 ms.

Seulement, grâce à ce test de temps, un problème a pu être détecté. Il y a un bouchon au niveau de la tâche COM Tx. C'est-à-dire que si plusieurs tâches tentent de réserver un emplacement dans le tableau de télémétries simultanément, les deux premières accèderont sans soucis aux deux emplacements disponibles. Cependant les suivantes vont êtres bloquées jusqu'à ce que la tâche COM Tx envoie les télémétries et libère donc un emplacement.

C'est pourquoi il a été décidé de réduire au maximum le temps entre deux tours de boucle de la tâche COM Tx. Une télémétrie a une taille maximale de 135 bytes et est envoyée à 9600 bits par seconde. Il faut donc environ 125 ms au module COM Tx (tâche et routine d'interruption) pour envoyer une télémétrie. Afin de laisser une marge suffisante, il a été décidé que la tâche COM Tx aurait un temps d'attente maximal entre deux tours de boucle de 200 ms. Dans le cas où le tableau de commandes est vide elle s'exécutera pour rien, mais cela lui prendra moins d'une milliseconde.

La tâche Sequencer doit pouvoir être exécutée plus d'une fois par seconde, car si une correction Doppler et une commande ont le même temps d'exécution, il doit pouvoir les exécuter toutes les deux. C'est pourquoi son temps d'attente maximal entre deux tours de boucle est de 500 ms.

Une télécommande ayant une longueur maximale de 133 bytes prend maximum 110 ms à être ajoutée au tableau de télécommandes du module COM Tx. Néanmoins, afin de ne pas surcharger le système, il est demandé à la station sol d'attendre une demi seconde entre chaque envoi de télécommandes. C'est pourquoi il a été décidé de cadencer la tâche de réception COM Tx à 2 Hz, son temps d'attente maximal entre deux tours de boucle est donc de 500 ms.

3.8.2 Priorités

Il y a juste quelques précautions à prendre au niveau de l'ordre de priorité de chaque tâche.

Premièrement, le module Monitor effectue les actions les plus critiques au sein du satellite, il doit donc être capable de réagir rapidement si par exemple le niveau de batterie est trop faible. C'est pourquoi la tâche du module Monitor doit être prioritaire.

Vient ensuite la tâche COM Tx. Elle doit également envoyer les télémétries le plus rapidement possible afin de ne pas bloquer les autres tâches,

comme expliqué dans le chapitre 3.8.1. De plus, la seule tâche de plus haute priorité est la tâche Monitor qui ne nécessite pas d'envoi de télémétries.

Viennent ensuite dans l'ordre les tâches Sequencer, Measurement, Com Rx et Log triées par leur niveau critique respectif. Les tâches Sequencer et Measurement ont chacune un rôle en fonction de la référence temporelle du satellite, comme exécuter une commande au bon moment ou encore prendre des mesures à une certaine fréquence, tandis que les deux dernières ne travaillent pas en fonction du temps, mais plutôt sur des événements.

Ces tâches ont chacune un degré de priorité différent, néanmoins les tests effectués démontrent que même les tâches de plus basse priorité s'effectuent quand elles le doivent ; leur tour de boucle démarre quand il le faut, et leur temps d'exécution n'est pas plus important que celui d'une autre tâche.

Chapitre 4

Télécommandes et Télémétries

Ce chapitre a pour but de définir le protocole utilisé pour communiquer avec OUFTI-1. Il permet donc aux développeurs de la station sol de connaître le format exact des télécommandes à envoyer, ainsi que celui des télémétries à recevoir.

4.1 Le protocole AX.25

Le protocole utilisé pour la couche de liaison de données est l'AX.25. Il s'agit d'un protocole défini et utilisé par les radioamateurs et dont on se servira tant pour les télémétries que pour les télécommandes. Il définit différents types de modes de transmission mais celui choisi pour la mission d'OUFTI-1 est le mode Unnumbered Information (UI) qui est un mode non-connecté. Pour plus de détails concernant les raisons du choix de ce protocole, se référer au travail de fin d'études de Laurent Chiarello [1]. La structure d'une trame AX.25 est représentée dans le tableau 4.1

Flag	Address		Control	PID	Info	FCS	Flag
0x7E	DEST Callsign	SRC Callsign	0x03	0xF0	DATA	CRC-CCITT	0x7E
1 byte	7 bytes	7 bytes	1 byte	1 byte	N (max 256)	2 bytes	1 byte

Tableau 4.1 – Format d'une trame AX.25

1. cf. [Chi09].

Voici le détail des différents champs :

- Les champs Flags permettent de délimiter la trame, et donc de détecter le début ou la fin de celle-ci.

- Le champ Address contient les Callsigns [2] de la source et de la destination de la trame. Chaque Callsign (6 bytes) est complété par son Secondary Station Identifier (SSID) [3] (1 byte). Il est important de noter que le Callsign et le SSID ne sont pas envoyés en clair, mais chaque byte subit un décalage d'un bit vers la gauche. Ainsi par exemple le byte 0b01100101 deviendra après décalage 0b11001010. Les bits dépassant du byte après décalage sont perdus, mais cela ne pose pas de problème car les Callsigns sont constitués de caractères de 0 à 9 et de A à Z, le bit de poids fort ne vaudra donc jamais 1.

- Le champ Protocol Identifier (PID) identifie le type de protocole utilisé pour la couche 3 (Network Layer). Comme cette couche n'est pas utilisée, la valeur du champ PID est de 0xF0 [4].

- Le champ Info contient les données utiles et a une longueur maximale de 256 bytes. Dans le cas d'OUFTI-1, il contiendra la télécommande ou la télémétrie encapsulée dans le protocole PUS. Ce protocole est expliqué dans le chapitre 4.2.

- Le champ Frame-Check Sequence (FCS) est un checksum qui permet de vérifier l'intégrité des données lors de leur réception. Grâce à ce checksum, une trame altérée lors de son transfert entre la station sol et OUFTI-1 ne sera pas interprétée par l'OBSW et pourra ainsi être réémise par la station sol.

4.2 Le protocole PUS

Le format général de la trame PUS se base sur le Space Packet Protocol (SPP). Le Primary Header est donc identique (cf. chapitre 4.2.1), seule la partie Data diffère du SPP. Le protocole PUS [5] a été adapté dans ce chapitre

2. Un Callsign est un identifiant unique pour une station d'émission et de réception radioamateur, chaque radioamateur licencié possède également son Callsign. Le Callsign de la station sol de l'ULg est ON4ULG.

3. Le SSID d'un Callsign désigne le type de station émettant (caractère '0' pour une station fixe.)

4. cf. [BNT98]

5. cf. [ECS03].

afin de correspondre au mieux aux besoins de la mission d'OUFTI-1. Tout changement effectué par rapport au protocole sera bien évidemment signalé et argumenté tout au long du chapitre.

Le tableau 4.2 représente les différents champs de la trame PUS.

PUS Frame				
SPP Primary Header	PUS Data			
	PUS Secondary Header	Data	Spare	Error Control
6 bytes	Variable			

Tableau 4.2 – PUS Frame

Détaillons à présent chaque partie de le trame.

4.2.1 Primary Header

SPP Primary Header						
Packet Identification				Sequence Control		Data Length
Version Number (=0)	Packet Type	Sec. Header Flag (=1)	APID	Sequence Flag	Sequence Count	
3 bits	1 bit	1 bit	11 bits	2 bits	14 bits	16 bits
2 bytes				2 bytes		2 bytes

Tableau 4.3 – SPP Primary Header

Voici l'explication des différents champs du tableau 4.3 :

- Packet Identification :
  - Version Number : La version du protocole utilisée est la 1, la valeur de ce champs vaut alors par définition '000'.

  - Packet Type : S'il s'agit d'une télécommande, ce bit vaudra '1', s'il s'agit d'une télémétrie, il vaudra '0'.
  - Secondary Header Flag : Ce bit sert à indiquer la présence du Secondary Header, ainsi s'il est présent ce bit vaudra '1', sinon il vaudra '0'. Dans le cas d'OUFTI-1, ce bit sera toujours égal à '1', car aucune trame ne circulera sans Secondary Header.
  - Application Process Identifier (APID) : L'APID est utilisé pour désigner le sous-système visé. Dans le cas d'OUFTI-1, le seul sous-système qui va gérer les télécommandes est l'OBC, ce champ n'est donc pas d'une grande utilité mais devra tout de même valoir 0x01 pour désigner l'OBC, si cela devait ne pas être le cas, la télécommande serait rejetée.

- Sequence Control : Ce champ permet de segmenter l'information en plusieurs trames. Voici les champs intervenant dans cette segmentation :
  - Sequence Flag :
    - '00' : Si la trame est un segment de données ;
    - '01' : Si la trame est le premier segment de données ;
    - '10' : Si la trame est le dernier segment de données ;
    - '11' : Si la trame n'est pas segmentée et se suffit à elle-même.

    Cependant, une segmentation des données n'est pas implémentée à bord de l'OBSW, car l'utilité ne s'est pas fait ressentir dans le cadre de la mission. Le champ Sequence Flag sera donc toujours à '11'.
  - Sequence Count : Il s'agit d'un compteur qui sera incrémenté à chaque envoi de télécommande ou de télémétrie. Ce compteur ne doit jamais être remis à zéro. Il sera entre autres utilisé pour les acquittements ; lors d'un acquittement le Sequence Count de la télécommande correspondante sera envoyé dans la télémétrie, ainsi la station sol pourra définir quelle télécommande a été acquittée.

- Data Length : Ce champ contient la taille en bytes de la partie Data de la trame PUS (y compris le Secondary Header). D'après le protocole [6], il s'agit d'un entier non signé 'C', où 'C'='nombre de bytes dans la partie Data - 1'.

6. cf. [ECS03]

4.2.2 Secondary Header

Le Secondary Header est différent selon qu'il s'agisse d'une télécommande ou d'une télémétrie. Ce chapitre reprend les formats des différents cas.

Télécommandes

PUS Secondary Header(TC)						
CCSDS Secondary Header Flag (=0)	Version Number (=1)	ACK	Service Type	Service Subtype	Source ID (Optional)	Spare (Optional)
1 bit	3 bits	4 bits	8 bits	8 bits	n bits	n bits
1 byte			1 byte	1 byte	n bytes	

Tableau 4.4 – PUS Secondary Header (TC)

Le tableau 4.4 reprend les différents champs du Secondary Header pour les télécommandes. Voici ses différents champs détaillés :

- Consultative Committee for Space Data Systems (CCSDS) Flag : Ce champ reste à '0' pour indiquer qu'il s'agit d'un Secondary Header non défini par la norme CCSDS.

- Version Number : La version utilisée est la 1, ce champ vaudra donc '001' [7].

- Acknowledgement (ACK) : Ce champ permet de spécifier le type d'acquittement que la station sol désire recevoir du satellite. Voici les différentes valeurs possibles :
  - '0000' : La station sol ne désire pas recevoir d'acquittement ;
  - '0001' : La station sol désire recevoir un acquittement lors de l'ajout de la commande dans le tableau de commandes ;
  - '0010' : La station sol désire recevoir un acquittement lors du début de l'exécution de la commande ;
  - '1000' : La station sol désire recevoir un acquittement à la fin de l'exécution de la commande.

7. La version '000' est utilisée par le PUS European Space Agency (ESA).

Une combinaison de ces différents types d'acquittements peut facilement être demandée en positionnant plusieurs bits à 1. Par exemple si le champ ACK vaut '1010', deux acquittements seront envoyés, un au début et l'autre à la fin de l'exécution de la commande.

- Service Type : Permet de définir le type de service demandé. Les nombres compris entre 0 et 127 sont réservés pour les services standards, tandis que les nombres de 128 à 255 peuvent être utilisés pour un service spécifique à la mission.

- Service Subtype : Permet de définir un sous-type de service. Ici aussi les nombres compris entre 0 et 127 sont réservés aux services standards, tandis que les nombres de 128 à 255 peuvent être utilisés pour les services spécifiques à la mission. A noter que si le type de service est spécifique ($128 \rightarrow 255$), toute la plage de valeurs peut être utilisée pour le sous-type.

- Source ID : Permet d'identifier la source de la trame et peut être ignoré s'il n'existe qu'une seule source. Ce champ sera ignoré, car comme la trame PUS est encapsulée dans une trame AX.25, la source est déjà présente sous la forme d'un Callsign (cf. chapitre 4.1 page 55).

- Spare : Ce champ permet d'ajouter des bits afin d'obtenir un nombre entier de bytes. Néanmoins comme le champ Source ID est ignoré, le Secondary Header comptabilise un nombre entier de bytes, ce champ peut donc également être ignoré.

Télémétries

PUS Secondary Header(TM)						
Spare	Version Number (=1)	Spare	Service Type	Service Subtype	Packet Subcounter	Time
1 bit	3 bits	4 bits	8 bits	8 bits	16 bits	32 bits
1 byte			1 byte	1 byte	2 bytes	4 bytes

Tableau 4.5 – PUS Secondary Header (TM)

Le tableau 4.5 reprend les différents champs du Secondary Header pour les télémétries. Voici ces différents champs détaillés :

- Spare : Les deux premiers champs Spare sont utilisés pour obtenir un byte complet.

- Version Number : La version utilisée est la 1, ce champ vaudra donc '001' [8].

- Service Type : Tout comme le Secondary Header des télécommandes, le champ Type permet de définir le type de service.

- Service Subtype : Tout comme le Secondary Header des télécommandes, le champ Subtype permet de définir un sous-type de service.

- Packet Subcounter : Il s'agit d'un nombre incrémenté à chaque envoi d'une télémétrie d'un certain type et sous-type. Ainsi, si la station sol détecte un trou, c'est-à-dire si un compteur pour un certain type et sous-type saute une valeur, elle peut déterminer le type et le sous-type de la télémétrie manquante.

- Time : Contient la référence temporelle du satellite au moment de l'envoi de la télémétrie. Cela permettra à la station sol d'être synchrone avec l'horloge du satellite.

8. La version 0 est utilisée par le PUS ESA

4.2.3 Data

A priori, la partie Data de la trame peut contenir tous les paramètres que l'on désire. Etant donné que toute commande envoyée à OUFTI-1 va être insérée dans le tableau de commandes, il serait intéressant de faire intervenir le Timestamp dans les données. Le Timestamp est utilisé pour spécifier le temps en secondes qui doit s'écouler entre la réception de la commande et son exécution. Une commande dont le Timestamp vaut zéro correspond donc à une commande à exécution directe, et une dont le Timestamp est plus grand que zéro correspond à une commande à exécution reportée.

Un tel système permettrait de contourner une contrainte du protocole PUS qui consiste à envoyer une commande de type 11 (On-board operation sequencing) et de sous-type 4 (Insert telecommand in command schedule) afin d'insérer une commande d'un autre type ; toute commande dont l'exécution doit être reportée devrait donc être envoyée à l'intérieur même d'une autre commande de type 11 et de sous-type 4. Etant donné que toutes les télécommandes vont être insérées dans le tableau de commandes, cela serait fort lourd au niveau traitement pour l'OBSW d'OUFTI-1.

Un autre changement au niveau de la partie Data concerne le champ Error Control (cf. le tableau 4.2 page 57). Etant donné qu'un checksum est déjà présent dans la couche de liaison de données AX.25, il y a une redondance inutile au niveau du contrôle d'intégrité. Le champ Error Control de la trame PUS peut donc être ignoré, ce qui implique que le champ Spare peut également être ignoré.

La solution proposée et implémentée à bord de l'OBSW est présentée dans le tableau 4.6.

<table>
<tr><th colspan="4">PUS Frame</th></tr>
<tr><td rowspan="3">SPP Primary Header</td><td colspan="3">PUS Data</td></tr>
<tr><td rowspan="2">PUS Secondary Header</td><td colspan="2">Data</td></tr>
<tr><td>Timestamp (Only for TC)</td><td>Data</td></tr>
<tr><td>6 bytes</td><td>Variable</td><td>4 bytes</td><td>Variable</td></tr>
</table>

Tableau 4.6 – Trame PUS : Implémentation finale

Il est important de noter que le Timestamp présent dans la partie Data ne le sera que pour les télécommandes, et non pour les télémétries.

4.3 Télécommandes

Ce chapitre est consacré à la définition du format exact des trames PUS pour chaque type de service demandé à OUFTI-1. Certains services sont tirés du protocole lui-même [9], alors que d'autres sont spécifiques à la mission et ont donc été créés pour répondre à ses besoins.

Au préalable, il est important de faire remarquer à nouveau que toutes les télécommandes contiendront un paramètre commun, le Timestamp. Ainsi les 4 premiers bytes du champ Data de chaque télécommande sont consacrés au Timestamp.

4.3.1 Service de rapatriement des mesures (Type 3)

Contrôle de la sauvegarde des mesures

Activer la sauvegarde des mesures

La télécommande d'activation de la sauvegarde des mesures est envoyée afin d'activer la prise d'une ou de plusieurs mesures. Elle permet également de définir sa fréquence d'échantillonnage. Le sous-type 128 a été choisi, car le paramètre concernant la fréquence d'échantillonnage n'est pas présent dans le protocole PUS.

Nom : MEAS_ENABLE
Type : 3
Sous-type : 128
Paramètres : tableau 4.7.

NMID	MID	Frequency
1 byte	1 byte	2 bytes
← Repeated NMID times →		

Tableau 4.7 – Paramètres de MEAS_ENABLE

9. cf. [ECS03]

- NMID : Il s'agit du nombre de mesures à activer et correspond donc au nombre de paires MID-Frequency.

- MID : Le Measurement Identifier (MID) est l'identifiant de la mesure à activer. La liste des mesures et leurs MIDs correspondants est disponible dans le travail de fin d'études de Thomas Langohr [10].

- Frequency : La fréquence d'échantillonnage peut être définie pour chaque mesure indépendamment. Ce paramètre correspond donc au nombre de secondes qui séparent deux prises de mesures. Si par exemple le champ vaut 1, la mesure sera prise à 1 Hz, s'il vaut 2, à 0,5 Hz, etc. Si par contre la fréquence doit rester inchangée, le champ Frequency peut être mis à zéro.

Désactiver la sauvegarde de mesures

Cette télécommande est envoyée afin de désactiver la sauvegarde de certaines mesures. Même les mesures de diagnostic peuvent être désactivées, car elles seront toujours contrôlées par la tâche Monitor, mais pas sauvegardées [11].

Nom : MEAS_DISABLE
Type : 3
Sous-type : 8
Paramètres : tableau 4.8.

NMID	MID
1 byte	1 byte
← Repeated NMID times →	

Tableau 4.8 – Paramètres de MEAS_DISABLE

- NMID : Il s'agit du nombre de mesures à désactiver et correspond donc au nombre de MIDs présents dans la télécommande.

- MID : Le MID de la mesure à désactiver.

10. cf. [Lan11]
11. cf. [Lan11].

Rapatriement de mesures

Demande de rapatriement de mesures

La télécommande de demande de rapatriement de mesures va être envoyée afin de rapatrier un certain type de mesures au sol. Il est également possible de définir la période sur laquelle on désire récupérer les mesures. De ce fait, le sous-type choisi doit être supérieur à 128 car la notion de temps n'est pas reprise dans le protocole PUS.

Nom : GET_MEAS
Type : 3
Sous-type : 129
Paramètres : tableau 4.9.

MID	Start Time	End Time
1 byte	4 bytes	4 bytes

Tableau 4.9 – Paramètres de GET_MEAS

- MID : Le MID de la mesure à récupérer.

- Start Time : Le Start Time permet de définir combien de temps (en secondes) l'on désire revenir en arrière. Par exemple, si Start Time vaut 3600, les mesures de la dernière heure vont être prises en compte.

- End Time : Le End Time permet de fermer l'intervalle de temps dans lequel l'on désire récupérer les mesures. Par exemple si le Start Time vaut 3600 et le End Time vaut 1800, la dernière demi-heure ne sera pas prise en compte, tandis que la demi-heure précédente le sera. Si ce champ vaut zéro, seul le Start Time sera pris en compte.

4.3.2 Service de rapatriement des événements (Type 5)

Demande de rapatriement des événements

Cette télécommande permet de demander le rapatriement des événements stockés à bord en spécifiant un intervalle de temps.

Nom : GET_LOG
Type : 5
Sous-type : 128
Paramètres : tableau 4.10.

Timestamp
4 bytes

Tableau 4.10 – Paramètres de GET_LOG

- Timestamp : Ce paramètre permet de choisir les événements que l'on désire rapatrier en fonction du temps. Par exemple s'il vaut 3600, les événements de la dernière heure seront rapatriés. Si l'on désire rapatrier l'intégralité des événements, la valeur de ce paramètre doit être égale à zéro.

4.3.3 Service de gestion de fonctions (Type 8)

Le service de gestion de fonctions permet d'effectuer toutes les opérations désirées à bord du satellite. Ainsi une trame de base pour les données est fournie et peut être modifiée suivant l'opération demandée. Ce qui suit n'est donc pas spécifique au protocole PUS, mais plutôt à la mission.

Gestion du mode de fonctionnement d'OUFTI-1

OUFTI-1 possède différents modes de fonctionnement. Ce service est dédié à la gestion de ces différents modes. Les différents modes de fonctionnement sont expliqués en détail dans le travail de fin d'études de Thomas Langohr [12].

12. cf. [Lan11]

Récupération du mode courant

La télécommande de récupération du mode courant est envoyée afin de récupérer le mode de fonctionnement courant du satellite.

Nom : GET_MODE
Type : 8
Sous-type : 128
Paramètres : Aucun paramètre n'est nécessaire pour cette commande.

Demande de changement de mode

La télécommande de demande de changement de mode est envoyée afin de modifier le mode de fonctionnement d'OUFTI-1. Le changement de mode n'est pas toujours réalisable car la tâche Monitor ne l'effectuera que si certaines conditions sont remplies (vérification du voltage des batteries, etc.).

Nom : CHANGE_MODE
Type : 8
Sous-type : 130
Paramètres : tableau 4.11.

Desired Mode
1 byte

Tableau 4.11 – Paramètres de CHANGE_MODE

- Desired Mode : Il s'agit du mode dans lequel OUFTI-1 doit passer. Voici les différentes valeurs possibles :
  - 2 : DEFAULT
  - 3 : SILENCE
  - 4 : D-STAR
  - 5 : xEPS
  - 6 : FUN MODE

Rapport du sous-système COM

Demande de rapport au sous-système COM

La télécommande de demande de rapport au sous-système COM permet de récupérer le dernier rapport de la COM. Ce rapport contiendra des informations sur la dernière passe D-STAR.

Nom : GET_COM_REPORT
Type : 8
Sous-type : 131
Paramètres : Aucun paramètre n'est nécessaire pour cette commande.

Corrections Doppler

Envoi de corrections Doppler

La télécommande d'envoi de corrections Doppler est un peu particulière car elle ne va pas être insérée dans le tableau de commandes comme les autres, mais ses paramètres vont remplir un tableau dédié aux corrections Doppler.

Comme expliqué dans le chapitre 3.4.2, chaque correction est accompagnée d'un Timestamp qui est relatif au Timestamp de la correction précédente. Le Timestamp de la toute première correction est relatif au Timestamp général de la télécommande. Cela permet d'économiser de la place, et ainsi de pouvoir utiliser un seul byte par Timestamp. Par exemple, si le Timestamp de la télécommande vaut 120, et que le Timestamp de la première correction vaut 10, cette dernière sera exécutée 130 secondes après réception de la télécommande dans le satellite. Si le Timestamp de la seconde correction vaut 5, elle sera exécutée 135 secondes après réception de la télécommande et ainsi de suite.

Nom : DOPPLER_CORRECTION
Type : 8
Sous-type : 133
Paramètres : tableau 4.12.

Start Index	End index	Relative Timestamp	Correction
1 byte	1 byte	1 byte	1 byte
		← Repeated (End-start)+1 times →	

Tableau 4.12 – Paramètres de DOPPLER_CORRECTION

- Start Index : Le Start Index est utilisé pour définir l'emplacement où la première correction doit être ajoutée dans le tableau de corrections Doppler. Cela permet d'envoyer le tableau de Corrections Doppler en plusieurs télécommandes.

- End Index : Tout comme le Start Index, il s'agit cette fois de l'index de l'emplacement où doit être placée la dernière correction. Cela sert essentiellement à retrouver le nombre de corrections contenues dans la télécommande.

- Relative Timestamp : Comme expliqué ci-dessus, il s'agit d'un Timestamp relatif au Timestamp de la correction précédente, et au Timestamp général de la commande pour la première correction. A noter que la première correction est celle contenue dans la télécommande dont le Start Index vaut zéro. Le Timestamp des télécommandes dont le Start Index est plus grand que zéro ne sera pas interprété.

- Correction : La correction à envoyer au sous-système COM.

4.3.4 Service de Séquençage (Type 11)

Contrôle du Sequencer

Activer l'exécution des commandes reportées

Cette commande permet d'activer l'exécution des commandes dont l'exécution est reportée. C'est-à-dire que le Sequencer exécutera les commandes dont le Timestamp est plus grand que zéro. Par défaut, au démarrage du satellite, l'exécution des commandes reportées est activée.

Nom : SEQ_ENABLE
Type : 11
Sous-type : 1
Paramètres : Aucun paramètre n'est nécessaire pour cette commande.

Désactiver l'exécution des commandes reportées

A l'inverse de la commande précédente, celle-ci permet de désactiver l'exécution des commandes reportées. Cela signifie que le Sequencer n'exécutera plus que les commandes dont le Timestamp vaut zéro. Cela peut être utile dans le cas de l'envoi de commandes erronées, on peut alors désactiver leur exécution (si leur exécution est reportée bien sûr) afin de pouvoir les supprimer sans qu'elles ne s'exécutent. Désactiver l'exécution des commandes reportées n'empêche pas d'ajouter ce type de commandes dans le tableau de commandes, c'est pourquoi, si elle est désactivée, un emplacement sera toujours réservé aux commandes à exécution directe, afin de pouvoir avoir la possibilité de la réactiver. Si ce n'était pas le cas, le tableau de commandes

pourrait se remplir de commandes à exécution reportée, ce qui bloquerait l'exécution de toute commande.

Nom : SEQ_DISABLE
Type : 11
Sous-type : 2
Paramètres : Aucun paramètre n'est nécessaire pour cette commande.

Réinitialiser le Sequencer

La réinitialisation du Sequencer correspond à effacer la totalité du tableau de commandes. Toutes les télécommandes envoyées au préalable et non encore exécutées seront donc supprimées. L'exécution des commandes reportées sera également réactivée si elle ne l'était pas.

Nom : SEQ_RESET
Type : 11
Sous-type : 3
Paramètres : Aucun paramètre n'est nécessaire pour cette commande.

Suppression de télécommandes

Suppression d'une télécommande

Cette commande permet de supprimer une télécommande bien précise dans le tableau de commandes.

Nom : DEL_COMMAND
Type : 11
Sous-type : 5
Paramètres : tableau 4.13.

Telecommand Packet ID	Packet Sequence Control
2 bytes	2 bytes

Tableau 4.13 – Paramètres de DEL_COMMAND

- Telecommand Packet ID : Ce paramètre correspond aux deux premiers bytes du Primary Header de la télécommande à supprimer (cf. chapitre 4.2.1 page 57). La station sol peut le déterminer soit grâce à son historique, soit en demandant un résumé du contenu du tableau de

commandes au satellite.

- Packet Sequence Control : Ce paramètre contient les deux bytes du Sequence Control du Primary Header (cf. chapitre 4.2.1 page 57) de la télécommande à supprimer. Ici aussi la station sol peut le déterminer soit grâce à son historique, soit en demandant un résumé du contenu du tableau de commandes. La concaténation des champs Packet ID et Sequence Control forme un identifiant unique pour une télécommande.

Suppression de télécommandes en fonction du temps

Les télécommandes dont le temps d'exécution est compris dans un certain intervalle de temps peuvent être supprimées grâce à cette commande.

Nom : DEL_COMMANDS_TIME
Type : 11
Sous-type : 6
Paramètres : tableau 4.14.

Range	Time 1 (Optional)	Time 2 (Optional)
1 byte	4 bytes	4 bytes

Tableau 4.14 – Paramètres de DEL_COMMANDS_TIME

- Range : Il s'agit de la manière dont la commande doit être interprétée. Voici les différentes valeurs possibles avec leur signification :
  - 0 : Toutes les commandes doivent être supprimées. Time 1 et Time 2 doivent être omis.
  - 1 : Les commandes dans l'intervalle [Time 1, Time 2] doivent être supprimées.
  - 2 : Les commandes dans l'intervalle [0, Time 1] doivent être supprimées. Time 2 doit être omis.
  - 3 : Les commandes dans l'intervalle [Time 1, $+\infty$] doivent être supprimées. Time 2 doit être omis.

- Time 1 : La signification de ce champ varie en fonction du paramètre Range. S'il est utilisé, il s'agit du temps absolu d'exécution des commandes. Celui-ci peut être récupéré en demandant un résumé du contenu du tableau de commandes.

- Time 2 : A l'instar de Time 1, la signification de ce champ varie en fonction du paramètre Range.

Résumé du contenu du tableau de commandes

Demande de résumé du contenu du tableau de commandes

Cette commande a pour effet de générer le rapatriement d'un résumé du contenu du tableau de commandes. Cela permet à la station sol de vérifier entre autres si ses télécommandes sont bien présentes dans ce tableau.

Nom : GET_COMMANDS_SUMMARY
Type : 11
Sous-type : 17
Paramètres : Aucun paramètre n'est nécessaire pour cette commande.

4.4 Télémétries

4.4.1 Service de vérification des télécommandes (Type 1)

Ce service est dédié à l'acquittement des télécommandes reçues dans le satellite. En fonction du type d'acquittement demandé (cf. chapitre 4.2.2 page 59) une des télémétries suivantes sera envoyée du satellite à la station sol.

Acceptation de la commande

Acceptation de la commande - AX.25_FAIL

Cette télémétrie d'acquittement est la seule qui ne peut être demandée depuis la station sol, car si la trame AX.25 est erronée, l'OBSW ne peut pas se fier à son contenu pour interpréter quoi que ce soit. Elle se charge donc simplement de notifier à la station sol qu'une erreur est survenue lors de l'interprétation de la trame AX.25. Ce type d'erreur n'étant pas repris dans le protocole PUS mais étant plutôt spécifique à la mission doit avoir un sous-type compris entre 128 et 255.

Nom : AX.25_FAIL
Type : 1
Sous-type : 128
Paramètres : tableau 4.15.

Error Code
1 byte

Tableau 4.15 – Paramètres de AX.25_FAIL

- Error Code : Précise le type d'erreur survenue lors de l'interprétation de la trame AX.25. Voici les différentes erreurs possibles :
  - 1 : BAD_CRC
  - 2 : BAD_SRC_CALLSIGN
  - 3 : BAD_DEST_CALLSIGN
  - 4 : BAD_CTRL_FLAG
  - 5 : BAD_PID

Acceptation de la commande - Succès

Ce type de télémétrie est envoyé si la commande a bien été ajoutée dans le tableau de commandes.

Nom : ACC_SUCCESS
Type : 1
Sous-type : 1
Paramètres : tableau 4.16.

Telecommand Packet ID	Packet Sequence Control
2 bytes	2 bytes

Tableau 4.16 – Paramètres de ACC_SUCCESS

- Telecommand Packet ID : Ce paramètre correspond aux deux premiers bytes du Primary Header de la télécommande acceptée (cf. chapitre 4.2.1 page 57). Ainsi la station sol peut déterminer la télécommande qui correspond à cette télémétrie.

- Packet Sequence Control : Dans le même but que le premier paramètre, celui-ci contient les deux bytes du Sequence Control du Primary Header de la télécommande acceptée (cf. chapitre 4.2.1 page 57). La concaténation des champs Packet ID et Sequence Control forme un identifiant unique pour une télécommande.

Acceptation de la commande - Echec

En cas d'échec de l'ajout de la commande dans le tableau de commandes, une télémétrie d'erreur sera envoyée. Les causes possibles d'échec sont énumérées ci-après.

Nom : ACC_FAIL
Type : 1
Sous-type : 2
Paramètres : tableau 4.17

Telecommand Packet ID	Packet Sequence Control	Error Code
2 bytes	2 bytes	1 byte

Tableau 4.17 – Paramètres de ACC_FAIL

- Telecommand Packet ID : Idem que pour ACC_SUCCESS.
- Packet Sequence Control : Idem que pour ACC_SUCCESS.
- Error Code : Voici les erreurs qui peuvent survenir avant l'ajout dans le tableau de commandes :
  - 6 : BAD_APID
  - 7 : BAD_LENGTH
  - 9 : BAD_HEADER
  - 10 : SEQ_FULL

Début d'exécution de la commande

Ce type de télémétrie sera envoyé au début de l'exécution de la commande. Cela peut être pratique par exemple pour les commandes à exécution reportée, la station sol peut ainsi détecter si l'exécution a effectivement été lancée.

Début de l'exécution de la commande - Succès

Nom : START_SUCCESS
Type : 1
Sous-type : 3
Paramètres : Idem que pour ACC_SUCCESS (cf. tableau 4.16).

Début de l'exécution de la commande - Erreur
Nom : START_FAIL
Type : 1
Sous-type : 4
Paramètres : Idem que pour ACC_FAIL (cf. tableau 4.17), seules les valeurs de code d'erreur sont différentes :

- Error Code :
  - 11 : BAD_TYPE
  - 12 : BAD_SUBTYPE
  - 13 : BAD_DATA

Fin de l'exécution de la commande

Ce type de télémétrie sera envoyé à la fin de l'exécution de la commande.

Fin de l'exécution de la commande - Succès
Nom : END_SUCCESS
Type : 1
Sous-type : 7
Paramètres : Idem que pour ACC_SUCCESS (cf. tableau 4.16).

Fin de l'exécution de la commande - Erreur
Nom : END_FAIL
Type : 1
Sous-type : 4
Paramètres :Idem que pour ACC_FAIL (cf. tableau 4.17). Pour ce qui est des codes d'erreurs, les seules télécommandes gérées directement par le Sequencer et qui pourraient provoquer une erreur sont les commandes de suppression de commandes. Pour celles-ci il existe un unique code d'erreur :

- Error Code :
  - 20 : COMMAND_NOT_FOUND

4.4.2 Service de rapatriement des mesures (Type 3)

Rapatriement des mesures

Cette télémétrie est générée lorsqu'une demande de rapatriement de mesure a été faite au préalable. Elle contient donc les valeurs d'une certaine mesure sur un certain laps de temps.

Nom : MEAS_RETRIEVE
Type : 3
Sous-type : 130
Paramètres : tableau 4.18.

MID	NMEAS	Timestamp	Value
1 byte	1 byte	4 bytes	1 byte
		← Repeated NMEAS times →	

Tableau 4.18 – Paramètres de MEAS_RETRIEVE

- MID : Le MID est l'identifiant de la mesure rapatriée. La liste des mesures et leurs MIDs correspondants est disponible dans le travail de fin d'études de Thomas Langohr [13].

- NMEAS : Correspond au nombre de valeurs contenues dans cette télémétrie, donc le nombre de paires Timestamp-Value.

- Timestamp : Chaque valeur est accompagnée de son Timestamp absolu du satellite qui correspond au moment où la mesure a été prise.

- Value : La valeur mesurée sera placée dans ce champ. Les valeurs ne sont pas converties, il s'agit de valeurs brutes lues à la sortie des convertisseurs. La conversion doit se faire au sol en fonction du type de mesure car les valeurs converties n'ont pas le même ordre de grandeur pour une température, que pour un voltage par exemple. Leur représentation devient donc difficile à l'intérieur d'une télémétrie qui doit être générique (doit pouvoir contenir n'importe quelle mesure).

13. cf. [Lan11]

4.4.3 Service de rapatriement des événements (Type 5)

Rapatriement des événements

Cette télémétrie contient les événements survenus à bord du satellite.

Nom : LOG_RETRIEVE
Type : 5
Sous-type : 129
Paramètres : tableau 4.19

NEvent	Timestamp	Event	Parameter
1 byte	4 bytes	4 bits	4 bits
← Repeated NEvent times →			

Tableau 4.19 – Paramètres de LOG_RETRIEVE

- NEvent : Nombre d'événements rapatriés dans cette télémétrie.
- Timestamp : Il s'agit du temps absolu du satellite au moment où l'événement est survenu.
- Event : Codé sur 4 bits, c'est le type d'événement qui est survenu. Voici les différentes valeurs possibles avec leur signification :
  - 1 : EVENT_OBC_STARTED
  - 2 : EVENT_ANTENNAS_DEPLOYED
  - 3 : EVENT_XEPS_STATUS_CHANGE
  - 4 : EVENT_MECH_STATUS_CHANGE
  - 5 : EVENT_DSTAR_STATUS_CHANGE
  - 6 : EVENT_RX_STATUS_CHANGE
  - 7 : EVENT_TX_STATUS_CHANGE
  - 8 : EVENT_BCN_STATUS_CHANGE
  - 9 : EVENT_XEPS_FAULT
  - 10 : EVENT_MEAS_FAULT
  - 11 : EVENT_COM33_FAULT
  - 12 : EVENT_COM72_FAULT
  - 13 : EVENT_EEPROM_FAULT
  - 14 : EVENT_VBAT_LOW
  - 15 : EVENT_COM_OBC_FAILED

- Parameter : Les paramètres diffèrent en fonction du type d'événement. On obtient ainsi la liste suivante :
  - Pour EVENT_OBC_STARTED : zéro pour le backup OBC, 1 pour le default OBC ;
  - Pour les STATUS_CHANGE : zéro pour OFF, 1 pour ON ;
  - Pour EVENT_XEPS_STATUS_CHANGE : zéro pour OFF, 1 pour ON sur la charge résistive, 2 pour ON sur le bus 3,3V ;
  - Aucun paramètre pour les autres événements. Le champ paramètre vaudra zéro.

4.4.4 Service de gestion de fonctions (Type 8)

Gestion du mode d'OUFTI-1

Mode de fonctionnement courant d'OUFTI-1

Envoyée sur demande, cette télémétrie envoie le mode de fonctionnement courant du satellite.

Nom : CURRENT_MODE
Type : 8
Sous-type : 129
Paramètres : tableau 4.20.

Mode
1 byte

Tableau 4.20 – Paramètres de CURRENT_MODE

- Mode : L'unique paramètre retourné dans cette commande est le mode. Il vaudra 2 s'il s'agit du mode DEFAULT, et 5 s'il s'agit du mode xEPS. Les autres modes du satellite ne pourront pas être retournés pour la simple et bonne raison que l'émission en AX.25 est désactivée dans les autres modes.

Rapport du sous-système COM

Cette télémétrie est dédiée au rapport du sous-système COM qui contient des informations concernant la dernière passe D-STAR.

Nom : COM_REPORT

Type : 8
Sous-type : 132
Paramètres : Les paramètres du rapport de la COM sont à définir car le contenu de ce dernier n'a pas encore été fixé.

4.4.5 Service de Séquençage (Type 11)

Résumé du contenu du tableau de commandes

Ce type de télémétrie est envoyé au sol en réponse à la télécommande de type 11 et de sous-type 17 (GET_SEQ_SUMMARY, cf. page 72). Elle permet de récupérer les données principales de chaque télécommande non encore exécutée contenue dans le tableau de commandes de l'OBSW.

Nom : COMMANDS_SUMMARY
Type : 11
Sous-type : 13
Paramètres : tableau 4.21.

NCommands	Absolute Timestamp	Telecommand Packet ID	Packet Sequence Control
1 byte	4 bytes	2 bytes	2bytes
	← Repeated NCommands times →		

Tableau 4.21 – Paramètres de COMMANDS_SUMMARY

- NCommands : Ce champ correspond au nombre de commandes représentées dans cette télémétrie.

- Absolute Timestamp : Il s'agit de la valeur correspondant au temps absolu auquel la télécommande doit s'exécuter. Ce Timestamp ne peut donc pas être interprété sans connaître l'horloge du satellite qui est présente dans le Secondary Header de chaque télémétrie (cf. chapitre 4.2.2 page 61).

- Telecommand Packet ID : Ce paramètre correspond aux deux premiers bytes du Primary Header de la télécommande (cf. chapitre 4.2.1 page 57).

- Packet Sequence Control : Ce paramètre contient les deux bytes du Sequence Control du Primary Header de la télécommande (cf. chapitre

4.2.1 page 57).

Grâce aux paramètres fournis dans cette télémétrie, la station sol sera capable de restituer le reste des informations de la télécommande au moyen de son historique. Cette opération est possible car la concaténation des champs Packet ID et Sequence Control permet une identification unique de chaque télécommande.

Conclusion

Afin de conclure cet ouvrage, tous les points implémentés vont être résumés afin d'avoir une vue d'ensemble du travail effectué.

Premièrement, la prise en main de FreeRTOS et de la programmation du microcontrôleur MSP430F1612 en langage C a été effectuée avant le début du stage grâce au site internet de FreeRTOS [14] et à la datasheet [15] du microcontrôleur.

L'ordinateur de bord d'OUFTI-1 est dorénavant capable de recevoir des télécommandes. Une fois reçues, il vérifie leur contenu et les stocke dans un emplacement mémoire approprié. Ce travail est effectué par le module COM Rx qui communique avec un démodulateur afin de pouvoir réceptionner ces télécommandes. La configuration de ce démodulateur est également gérée.

Le module Sequencer se charge ensuite d'exécuter ces télécommandes au moment opportun en fonction du Timestamp qui les accompagne. Le module Sequencer est également capable de gérer les corrections Doppler à envoyer au sous-système COM.

Le dernier module implémenté est le module COM Tx qui se charge de dialoguer avec un modulateur afin d'envoyer des télémétries à la station sol. Le module COM Tx se charge également de la configuration de ce modulateur.

Afin de pouvoir tester cette chaîne de communication, une application de test a été implémentée et permet d'envoyer des télécommandes et de recevoir des télémétries. Elle remplace en quelque sorte la station sol, si ce n'est que la communication ne se fait pas par ondes radio, mais par USB. Son interface graphique simplifie la visualisation des télémétries reçues et permet d'envoyer

14. cf. [Rea11].
15. cf. [Ins02].

des télécommandes dont le format est géré automatiquement.

Toutefois ce travail n'aurait pas pu être réalisé sans un protocole de communication précis et définitif. C'est pourquoi le format des télécommandes et des télémétries a été précisé, et une liste des différents types de services a été réalisée afin de pouvoir terminer le module Sequencer, pour qu'il puisse effectivement connaître l'opération à exécuter en fonction des commandes reçues. Néanmoins cette liste n'est pas exhaustive et peut contenir certaines lacunes comme la possibilité d'écraser certaines parties de la mémoire afin de pouvoir corriger certaines erreurs depuis la station sol.

L'ordinateur de bord d'OUFTI-1 est fonctionnel, même si quelques concessions ont dû être faites au niveau de la taille de l'espace mémoire occupé par les télécommandes et les télémétries.

La prochaine étape à réaliser est un test avec la carte COM et ses modulateurs/démodulateurs afin de pouvoir tester leur configuration ainsi que les antennes afin d'envoyer et de recevoir des télémétries en situation réelle.

Annexe A

Activités

Cette annexe est consacrée à la présentation des différentes activités réalisées durant la période de stage en plus du travail technique.

A.1 Journée d'initiation au radioamateurisme

Le mercredi 9 mars 2011 des radioamateurs de Gembloux ont été invités à l'Université de Liège afin de nous présenter ce qu'est le radioamateurisme. Ils nous ont présenté le radioamateurisme en général (règles à respecter, déroulement d'une communication radioamateur, etc.) et expliqué en détails le fonctionnement de diverses antennes. Du point de vue pratique, nous avons installé une antenne, assisté à de nombreuses communications dont une avec un radioamateur situé en Italie, et également découvert une station radioamateur mobile dans une voiture. Cette journée fut très enrichissante et nous a permis de mieux comprendre à quoi pouvait servir un relais D-STAR dans l'espace.

A.2 CubeSat Developers' Workshop - CalPoly

Au cours de mon stage à l'Université de Liège, j'ai eu l'opportunité de participer à la huitième conférence annuelle des développeurs de CubeSats. Celle-ci se tenait à la California Polytechnic State University à San Luis Obispo aux Etats-Unis du 20 au 22 avril 2011. J'y ai présenté le design de l'ordinateur de bord devant un public de professionnels et d'étudiants. J'ai également assisté aux différentes présentations durant les trois jours de la conférence. Afin de pouvoir participer à cet événement, j'ai tout d'abord rédigé un abstract de ma présentation que j'ai envoyé aux organisateurs. Ils l'ont ensuite validé, ce qui m'a permis d'y présenter l'ordinateur de bord. J'ai

eu l'occasion de rencontrer des gens du monde entier et j'ai particulièrement sympathisé avec une équipe d'étudiants péruviens qui travaille également sur un CubeSat. Cela a été une expérience inoubliable et très enrichissante. La figure A.1 est une photo du dernier jour de la conférence avec des étudiants péruviens, anglais et turcs.

FIGURE A.1 – Photo prise durant le CubeSat Developers' Workshop à Cal-Poly en Californie

A.3 Design Review

Le week-end du 29 avril au 1er mai 2011, l'équipe d'OUFTI-1 au grand complet s'est réunie à l'Euro Space Center de Redu afin de réaliser une Design Review. Il s'agit d'un processus souvent utilisé dans les entreprises et qui sert à examiner le statut technique d'un projet. Afin de pouvoir mener à bien cette Design Review, chaque étudiant a dû écrire un Data Package en anglais expliquant en détails le fonctionnement technique du sous-système dont il est en charge. Ces Data Packages ont été lus et commentés par des experts internes ou externes au projet. Ensuite chaque étudiant a dû préparer des réponses à ces commentaires afin de pouvoir en discuter lors des réunions du week-end à Redu. Enfin, chaque étudiant a dû modifier son Data Package afin de correspondre aux remarques faites et discutées en réunion. Ainsi chaque sous-système est décrit techniquement dans un dossier.

Ce week-end a également été l'occasion de réaliser quelques activités au sein de l'Euro Space Center comme une simulation de mission dans leur simulateur de poste de contrôle (voir figure A.2).

FIGURE A.2 – Simulation de mission à l'Euro Space Center

Liste des figures

Liste des tableaux

Bibliographie

[AZU09] AZUR SPACE Solar Power : *30% Triple Junction GaAs Solar Cell - Type : TJ Solar Cell 3G30C*. Rapport technique, AZUR SPACE Solar Power GmbH, Theresienstr. 2 - 74072 Heilbronn, 2009.

[Bar09] Richard Barry : *FreeRTOS Reference Manual*. Real Time Engineers Ltd., 2009.

[Bar10] Richard Barry : *Using the FreeRTOS Real Time Kernel : A Practical Guide*. Real Time Engineers Ltd., 2010.

[BNT98] William A. Beech, Douglas E. Nielsen et Jack Taylor : *AX.25 Link Access Protocol for Amateur Packet Radio - Version 2.2*. Rapport technique, Tucson Amateur Packet Radio Corporation, Juillet 1998.

[Chi09] Laurent Chiarello : *Design and implementation of the control and monitoring system for the ground station of nanosatellite OUFTI-1 of the University of Liège*. Mémoire de Master, Université de Liège, 2009.

[Dev07] Analog Devices : *High Performance Narrow-Band Transceiver IC - ADF7021*. Rapport technique, Analog Devices Inc., Norwood, MA 02062-9106, 2007.

[dL11] Université de Liège : *OUFTI-1, Nanosatellite Project*, Mai 2011. www.leodium.ulg.ac.be.

[DP09] Amandine Denis et Jonathan Pisane : *OUFTI-1 Phase A : Mission definition, Space and ground systems description*. Rapport technique, Université de Liège, Septembre 2009.

[ECS03] ECSS : *Space engineering - Ground systems and operations – Telemetry and telecommand packet utilization - ECSS-E-70-41A*. Rapport technique, ESA-ESTEC, Requirements & Standards Division, Noordwijk, The Netherlands, Janvier 2003.

[Ins02] Texas INSTRUMENT : *MSP430x15x, MSP430x16x, MSP430x161x, Mixed Signal Microcontroler*. Rapport technique, Texas Instrument, Dallas, Texas 75265, Octobre 2002.

[Lan11] Thomas LANGOHR : *Implémentation du monitoring, de la journalisation et de l'acquisition des mesures au sein de l'ordinateur de bord du nanosatellite OUFTI-1*. Mémoire de Bachelier, HEPL, INPRES, 2011.

[Led09] Philippe LEDENT : *Design and Implementation of On-board Digitally Controlled Electrical Power Supply of Student Nanosatellite OUFTI-1 of University of Liege*. Mémoire de Master, Université de Liège, 2009.

[Mic05] MICROCHIP : *1024K I^2C CMOS Serial EEPROM*. Rapport technique, Microchip Technology Inc., Chandler, AZ 85224-6199, 2005.

[Pro08] The CubeSat PROGRAM : *CubeSat Design Specification - REV.11*. Rapport technique, California Polytechnic State University, San Luis Obispo, CA 93407, Octobre 2008.

[Rea11] REAL TIME ENGINEERS LTD. : *FreeRTOS*, Mai 2011. www.freertos.org.

[Ten09] Damien TENEY : *Design and implementation of on-board processor and software of student nanosatellite OUFTI-1*. Mémoire de Master, Université de Liège, 2009.