

# **Implémentation du protocole AX.25 à bord du nanosatellite OUFTI-1**



Mémoire présenté par  
**HARDY Johan**

pour l'obtention du grade  
**de Master en Sciences de  
l'Ingénieur Industriel**  
Section : **Industrie**

Unité de référence  
**Electronique**

Tuteur Gramme  
**VETCOUR Nathalie**  
**Chargé de cours**

Tuteur Entreprise  
**BOIGELOT Bernard**  
**Professeur**

Société  
**Institut Montefiore (B28)**  
**Université de Liège**  
**Grande Traverse, 10**  
**B-4000 Liège (Sart-Tilman)**

Défense publique le  
**29 juin 2009**



Per aspera ad astra

## Remerciements

*Ce travail de fin d'études a été réalisé à l'université de Liège - Institut Montefiore, département d'électricité, électronique et informatique. Je remercie tous les professeurs de l'université de Liège ainsi que tous les membres du projet OUFTI-1 de m'avoir accueilli et d'avoir accepté ma participation.*

*Je remercie le professeur Gaëtan Kerschen (ULg) de m'avoir encouragé à participer au projet, le professeur Bernard Boigelot (ULg) d'avoir accepté de me suivre dans ce projet et enfin le professeur Jacques Verly (ULg) pour ses encouragements et conseils lors de ma présentation à l'ESTEC. Je remercie vivement Nathalie Vetcour (Institut Gramme) pour son suivi, son intérêt, ses encouragements et son élan de motivation pour ce projet peu commun. Je remercie également la communauté radioamateur et Luc Halbach (ON6JY) initiateur du projet OUFTI-1.*

*J'ai eu le plaisir et la chance de travailler dans une équipe avec des collaborateurs de travail très agréables ; à savoir : Vincent Beukelaers (Mission Analysis - ULg), Laurent Chiarello (Control station - ULg), Nicolas Evrard (On-Board Computer - Institut Gramme), Samuel Hannay (Attitude Determination and Control - ULg), Renaud Henrard (Telecommunications - ISIL), Lionel Jacques (Thermal Control - ULg), Philippe Ledent (Electric Power System - ULg), Francois Mahy (Telecommunications - ULg), Gauthier Pierlot (Structure and Configuration - ULg), Damien Teney (On-Board Computer - ULg), Pierre Thirion (Electric Power System - ULg), Valéry Broun (Beacon - ISIL) et Jérôme Wertz (Mechanisms - Institut Gramme). Je leur fais part de ma plus grande sympathie. Damien Teney, Nicolas Evrard, Francois Mahy et Renaud Henrard ont principalement travaillé avec moi. Je les remercie pour les nombreuses discussions techniques et brainstormings que nous avons eus au cours de ce projet.*

*Je remercie également Amandine Denis et Jonathan Pisane pour le management du projet et les déplacements instructifs aux Pays-Bas (Noordwijk) et en Suisse (Lausanne). Plus particulièrement, je remercie Jonathan Pisane pour son suivi au niveau de l'AX.25 et ses éclairages techniques.*

*Concernant la phase de test, celle-ci a été effectuée à l'ISIL. Etant donné que je ne possédais pas de licence radioamateur, je tiens à remercier Renaud Henrard (ON4JRH) et François Mahy (ON4FAN) pour les tests hautes fréquences et les dispositifs qu'ils ont mis en place pour moi. Je les remercie également de leur intérêt pour mon travail.*

*A l'initiative de Gaston Bertels, nous avons pu entrer en contact avec Frank De Winne. Je le remercie également.*

*Au niveau international, je remercie Jean-Claude Cano (F6CSS), radioamateur français, pour ses conseils. Je remercie Muriel Noca (Ecole Polytechnique Fédérale de Lausanne - EPFL) pour son accueil et plus spécialement Ted Choueiri (Etudiant à l'EPFL) pour ses éclairages techniques au niveau du terminal node controller. Je remercie également Roman Merz (Université de Neuchâtel) pour son aide concernant l'implémentation du code.*

*J'exprime enfin ma gratitude à l'ensemble des personnes qui ont contribué, de près ou de loin, à l'avancement de ce mémoire.*

# Acronymes

<i>A – FSK</i>	Audio - Frequency Shift Keying
<i>ADCS</i>	Attitude and Determination Control System
<i>APRS</i>	Automatic Position Reporting System
<i>ARISS</i>	Amateur Radio on the International Space Station
<i>ASCII</i>	American Standard Code for Information Interchange
<i>BCN</i>	Beacon
<i>CCITT</i>	Consultative Committee in International Telegraph and Telephone
<i>CCSDS</i>	The Consultative Committee for Space Data Systems
<i>CNES</i>	Centre National d'Etudes Spatiales
<i>COM</i>	Communications System
<i>CRC</i>	Cyclic Redundancy Check
<i>D – STAR</i>	Digital Smart Technologies for Amateur Radio
<i>DGTRÉ</i>	Direction Générale des Technologies, de la Recherche et de l'Energie
<i>DLSAP</i>	Data-Link Service Access Point
<i>DSP</i>	Digital Signal Processor
<i>EPFL</i>	Ecole Polytechnique Fédérale de Lausanne
<i>EPS</i>	Electrical Power Supply
<i>ESA</i>	European Space Agency
<i>ESTEC</i>	European Space Research and Technology Centre
<i>FCS</i>	Frame CheckSum
<i>FM</i>	Frequency Modulation
<i>FSK</i>	Frequency Shift Keying
<i>GMSK</i>	Gaussian Minimum Shift Keying
<i>HDLC</i>	High level Data Link Control
<i>IDE</i>	Integrated Development Environment
<i>ISO</i>	International Organization for Standardization
<i>ISS</i>	International Space Station
<i>KISS</i>	Keep It Simple and Stupid
<i>MSK</i>	Minimum Shift Keying
<i>NASA</i>	National Aeronautics and Space Administration
<i>NORAD</i>	North American Aerospace Defense Command
<i>NRZ</i>	Non Return to Zero
<i>NRZI</i>	Non Return to Zero Inverted
<i>OBC</i>	On-Board Computer
<i>OSI</i>	Open Systems Interconnection
<i>PCB</i>	Printed Circuit Board
<i>PID</i>	Protocol IDentifier
<i>PLL</i>	Phase Locked Loop
<i>RTBF</i>	Radio-Télévision Belge de la communauté Française
<i>SSID</i>	Second Station IDentifier
<i>SWD</i>	Synchronization Word Detection
<i>TM/TC</i>	Télémétries/Télécommandes
<i>TNC</i>	Terminal Node Controller
<i>UDP</i>	User Datagram Protocol
<i>U/VHF</i>	Ultra/Very - High Frequency
<i>USB</i>	Universal Serial Bus

# Chapitre 1

## Introduction

Le satellite OUFTI-1 sera le premier nanosatellite belge placé en orbite. Faisant référence à une expression populaire, l'acronyme du satellite signifie : *Orbital Utility For Telecommunication Innovations*. OUFTI-1 est donc un satellite de télécommunication. L'objectif principal du projet OUFTI-1 est d'implémenter le protocole D-STAR à bord d'un CubeSat. Le D-STAR<sup>1</sup> est un protocole radioamateur novateur de télécommunication digitale qui peut transmettre simultanément de la voix et des données. L'utilisation de cette technologie dans des conditions spatiales constitue également une première mondiale.

En 2004, le projet fit ses premiers balbutiements sous le nom de code : Leodium<sup>2</sup> program. Ce programme, initié par l'université de Liège, avait pour objet le développement d'une série de satellites étudiants consacrés à des fins expérimentales. Leodium n'aurait sans doute concrètement jamais vu le jour sans l'intervention, auprès du professeur Jacques Verly, de Luc Halbach (Spacebel). Ce dernier, ingénieur civil électronicien de formation et radioamateur chevronné, eut l'idée d'implémenter le protocole D-STAR à bord d'un satellite. Le programme était lancé et le projet OUFTI-1 pouvait démarrer.

Une première équipe fut formée. Nous retrouvions bien entendu Luc Halbach, le professeur Jacques Verly mais aussi le professeur en mécanique et aérospatiale Gaëtan Kerschen ainsi que Pierre Rochus du centre spatial de Liège. L'équipe fut ensuite complétée par trois étudiants : Stefania Galli, Jonathan Pisane et Philippe Ledent. Ces derniers travaillèrent ensemble autour du même but commun : développer et fabriquer un satellite. Ensuite, les évènements s'enchaînèrent rapidement.

Durant le mois de janvier 2008, le premier relais terrestre D-STAR belge est installé à l'institut Montefiore (ULg). Les trois étudiants allèrent également défendre leur projet devant les experts du centre européen de recherches et technologies spatiales de l'ESA<sup>3</sup> dans le but d'obtenir une place dans le nouveau lanceur européen Vega. Ceux-ci affrontèrent trente autres concurrents venus des quatre coins de l'Europe. Dans le courant du mois de février 2008, le verdict tombe et OUFTI-1 prendra place à bord de Vega lors du tir inaugural à Kourou<sup>4</sup>.

S'ensuit alors le recrutement d'une série d'étudiants motivés pour concevoir les différents sous-systèmes du satellite. Ces étudiants proviennent des principales écoles d'ingénieur de la région liégeoise à savoir l'ULg, l'institut Gramme et l'ISIL.

Le satellite est donc entièrement conçu par des étudiants. Ceux-ci travaillèrent à développer des solutions innovantes, tout en tenant compte du besoin de robustesse et de fiabilité.

---

<sup>1</sup>Digital Smart Technologies for Amateur Radio.

<sup>2</sup>Liège en latin.

<sup>3</sup>European Space Agency.

<sup>4</sup>Guyane française.

Mais le projet OUFTI-1 se focalise aussi sur les objectifs suivants :

1. Offrir aux étudiants la possibilité d'acquérir une expérience pratique dans le domaine spatial.
2. Permettre aux étudiants d'acquérir des connaissances technologiques.
3. Apprendre à travailler en équipe et savoir gérer un projet.

Ces objectifs sont des points clefs maintenus tout au long du projet. Le partenariat et les compétences des différents instituts permettent également de tirer les avantages et l'expérience de chacun.

Dans le cadre de ce travail de fin d'études, l'aventure spatiale démarra fin juin 2008. Dans un premier temps, l'objectif principal fut de concevoir et d'implémenter le logiciel de l'*On-Board Computer* (OBC) d'OUFTI-1. Ensuite, dans un second temps, l'objectif s'est focalisé plus particulièrement sur l'implémentation du protocole AX.25 destiné à communiquer les télécommandes et les télémétries du satellite.

Ce mémoire décrit, en détail, la mise en oeuvre du protocole AX.25 pour le satellite OUFTI-1. Le chapitre 2 illustre de manière générale le CubeSat OUFTI-1 et le contexte du projet. Le cahier des charges du travail de fin d'études se trouve au chapitre 3. Les descriptions techniques concernant le protocole commencent à partir du chapitre 4 jusqu'au dernier chapitre. Le chapitre 11 est le dernier chapitre de ce document. Ce chapitre reprend les conclusions du travail de fin d'études. Enfin, en annexe, se trouvent une série de documents ainsi qu'une description des diverses activités ménées au cours de ce travail de fin d'études.



## Première partie

### Préambule

## Chapitre 2

# Présentation du CubeSat OUFTI-1

Développer et construire un satellite coûte très cher. De plus, si la taille de ce dernier est imposante, le coût tend vers une exponentielle. Par conséquent, des satellites *low-cost* ont été conçus pour toucher un public plus large. Parmi ces satellites *low-cost*, nous retrouvons le CubeSat. Le satellite OUFTI-1 suit ce standard.

De manière générale, ce chapitre illustre le satellite OUFTI-1 par sous-système et met l'accent sur les principaux intérêts du CubeSat. Enfin, ce chapitre tentera également de mettre en évidence le contexte d'un tel projet. En effet, l'environnement spatial est très agressif et les contraintes sont considérables. Il faut donc réfléchir et prendre des réflexes différents pour pallier à ces contraintes inhabituelles.

### 2.1 Le standard CubeSat

Le CubeSat est un satellite artificiel miniaturisé. Il a un poids très faible et une taille très petite par rapport à un satellite artificiel traditionnel pouvant parfois atteindre plus d'une tonne. La raison principale de concevoir ce type de satellite est la réduction du coût. Les gros satellites requièrent un coût en développement et fabrication conséquent. Le coût du lancement doit aussi être pris en considération. En effet, si le satellite a une masse importante, le lanceur doit aussi avoir un volume important. Les CubeSats requièrent donc des lanceurs beaucoup plus petits et beaucoup moins chers. De plus, vu leurs petites tailles, plusieurs CubeSats peuvent être mis simultanément en orbite avec le même lanceur.

En plus du coût, la seconde raison de l'utilisation de tels satellites est l'opportunité de réaliser des missions que de gros satellites ne pourraient pas mener ; par exemple :

- Constellation d'un grand nombre de satellites.
- Mise en orbite pour l'inspection de satellites plus imposants.
- Rapatriement de données à partir de plusieurs points.
- (...)

## Classification

Parmi les satellites miniaturisés, nous retrouvons plusieurs classifications :

Classifications	Masses
Minisatellite	100 à 150 kg
Microsatellite	10 à 100 kg
Nanosatellite	1 à 10 kg
Picosatellite	0,1 à 1 kg

Les CubeSats ont une masse qui oscille au voisinage du kilogramme. Cependant, la majorité des CubeSats ont une masse d'un kilogramme ; ce qui le classe dans la catégorie des nanosatellites.

## Spécifications

Les premières lignes du CubeSat ont été dessinées à CalPoly<sup>1</sup>. Ce projet avait pour but d'ouvrir la porte des étoiles aux universités. Plus de 40 universités internationales, hautes écoles et firmes privées ont participé à ce projet.

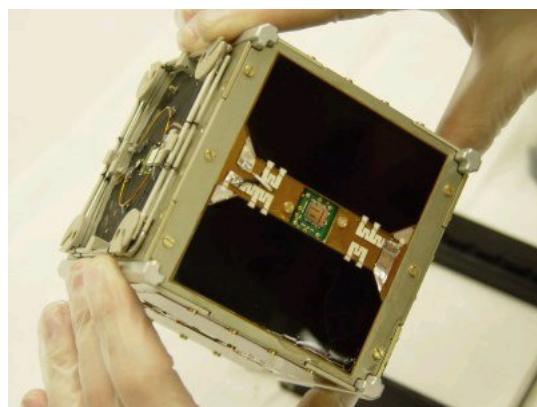


FIGURE 2.1 – Illustration d'un CubeSat

Encouragé par le professeur Bob Twiggs<sup>2</sup> (considéré actuellement comme le père du CubeSat) et après plusieurs années de développement, un standard fut adopté. Le standard CubeSat [1], souvent appelé 1U<sup>3</sup>, a un volume d'un litre (10 cm de côté) et une masse d'un kilogramme. Le CubeSat est extensible à 2U (20x10x10 cm) et 3U (30x10x10 cm).

L'intérêt grandissant des entreprises et organismes scientifiques boosta le développement de ce genre de satellite. En août 2008, on comptabilisait déjà plus de 113 CubeSats en orbite<sup>4</sup> et le chiffre ne cessera sans doute pas d'augmenter dans les années à venir.

<sup>1</sup>California Polytechnic State University.

<sup>2</sup>Department of Aeronautics and Astronautics at Stanford University.

<sup>3</sup>One Unit.

<sup>4</sup>Voir quelques exemples de CubeSat en Annexe 5

### Système de déploiement

Etant donné que le CubeSat est un standard, ses dimensions sont fixées. Le système de déploiement dans l'espace a donc été standardisé [2] aussi. Il s'agit en fait du P-POD.



FIGURE 2.2 – P-POD - Système de déploiement

Le P-POD est en fait une boîte en aluminium pouvant contenir trois CubeSats (capacité de 3x1U). A l'intérieur de cette boîte se trouve un ressort. Ce ressort, une fois comprimé, va éjecter les CubeSats lors de l'ouverture de la porte.



FIGURE 2.3 – Vue de face du P-POD

La fiabilité de ce système de déploiement a fait ses preuves, si bien qu'il est quasiment devenu la seule interface possible avec le lanceur.

### Le CubeSat Kit Pumpkin

Dans la même optique, voyant un marché grandissant, plusieurs sociétés ont commencé à commercialiser des CubeSats. C'est notamment le cas de Pumpkin. Cette société commercialise un kit<sup>5</sup> de développement : le CubeSat Kit. Ce kit comprend tous les outils nécessaires pour développer, concevoir et tester son prototype.

---

<sup>5</sup><http://www.cubesatkit.com/>

Il se compose de la structure mécanique, d'un modèle de vol, d'une carte de développement, d'un programmeur, etc.



FIGURE 2.4 – CubeSat Kit

Pour commencer le développement du logiciel de l'OBC et pour éviter de perdre du temps en développement (notamment pour la structure mécanique), ce kit a été acheté par l'université de Liège.

## 2.2 Le lanceur Vega

Déjà annoncé en introduction, c'est à bord de ce lanceur qu'OUFTI-1 sera mis en orbite. Du nom de l'étoile la plus brillante de la constellation de la Lyre, Vega est le nouveau lanceur européen de la famille Ariane<sup>6</sup>. Projet européen où la Belgique participe, c'est lors de son vol inaugural que Vega emportera également, en plus d'un satellite italien, 8 autres CubeSats :

Nations	CubeSats
Belgique	OUFTI-1
Espagne	Xatcobeo
France	Robusta
Italie	UNIcubesat
Italie	AtmoCube
Italie	e-st@r
Pologne	PW-sat
Roumanie	Goliat
Suisse	SwissCube

Malgré le fait que les satellites deviennent de plus en plus volumineux, il existe tout de même une demande importante quant à la mise en orbite basse de petits satellites inférieurs à 2000 kg. Vega répond en fait à cette demande. Grâce à Vega, le moyen d'accéder à l'espace est plus facile, plus rapide et moins cher. Le coût est en fait maintenu au minimum en réutilisant les installations de production du lanceur Ariane. Contrairement à la plupart des petits lanceurs, Vega sera en mesure de placer plusieurs charges utiles en orbite.

Les mensurations de Vega restent tout de même impressionnantes : hauteur de 30 m, diamètre de 3 m et masse au décollage égale à 137 tonnes.

<sup>6</sup>Et plus précisément Ariane Space.

## 2.3 Mission

Le développement d'un satellite se déroule en plusieurs phases. Ces phases correspondent aux états successifs de la mission. On distingue généralement six grandes phases [3] :

1. Phase 0 : relative à l'analyse de mission.
2. Phase A : relative à la faisabilité.
3. Phase B : relative à la définition préliminaire.
4. Phase C/D : relative à la définition détaillée, à la réalisation et à la qualification au sol et en vol.
5. Phase E : relative à l'utilisation.
6. Phase F : relative au retrait de service.

Plus généralement, les phases 0, A et B décrivent la mission du satellite. Ces phases sont très importantes car elles constituent une véritable ligne directrice du projet. Nous y retrouvons principalement les objectifs de la mission généralement associés aux *payloads*<sup>7</sup>.

### 2.3.1 Objectifs

Le projet OUFTI-1 est par-dessus tout un projet étudiant. Les objectifs sont avant tout éducationnels. A priori, le premier objectif est de construire un CubeSat fonctionnel. Mais l'objectif principal de la mission est de tester le protocole D-STAR à bord d'un CubeSat et a fortiori dans des conditions spatiales. Cette expérience, n'ayant jamais été tentée, constituerait une première mondiale.

### 2.3.2 Payloads

Le satellite OUFTI-1 aura plusieurs *payloads* à bord. La principale *payload* est le protocole D-STAR. La mise en oeuvre du D-STAR dans OUFTI-1 se matérialisera par un répéteur. Associé à cette première *payload*, nous retrouverons également deux autres *payloads* : une EPS<sup>8</sup> expérimentale et des cellules solaires à haut rendement.

L'EPS expérimentale est développée en partenariat avec *Thales Alenia Space ETCA* tandis que les cellules solaires sont développées par *AzureSpace*.



FIGURE 2.5 – Cellules solaires

Ces deux *payloads* sont « offertes » par les entreprises et, en contrepartie, nous devrons mettre à leur disposition plusieurs informations ; comme des courbes  $I - V$ , mesures spécifiques, etc.

<sup>7</sup>Charges utiles du satellite.

<sup>8</sup>Electrical Power Supply.

## 2.4 Le CubeSat OUFTI-1

Comme nous l'avons dit déjà plusieurs fois, OUFTI-1 est un CubeSat. Au niveau de son architecture, OUFTI-1 suit les spécifications du PC/104<sup>9</sup>. Le satellite est composé de cinq PCB<sup>10</sup>, de deux antennes, de panneaux solaires, d'une structure mécanique, etc. Le « système OUFTI-1 » est subdivisé en sous-systèmes. Nous retrouvons par exemple : le contrôle thermique, le système de télécommunication, l'analyse de mission, l'OBC, etc. En voici les détails :

### Structure et configuration

Au niveau de sa structure et de sa configuration, il faut absolument qu'OUFTI-1 « survive » aux conditions de lancement. Les accélérations et les vibrations régnant pendant le lancement sont assez impressionnantes. La structure pourrait s'effondrer sur elle-même. Il faut donc assurer l'intégrité structurelle du satellite au lancement et en orbite.

Dans la même optique, la configuration des PCB et des systèmes embarqués doit être viable et doit respecter le standard CubeSat. OUFTI-1 a la configuration suivante :

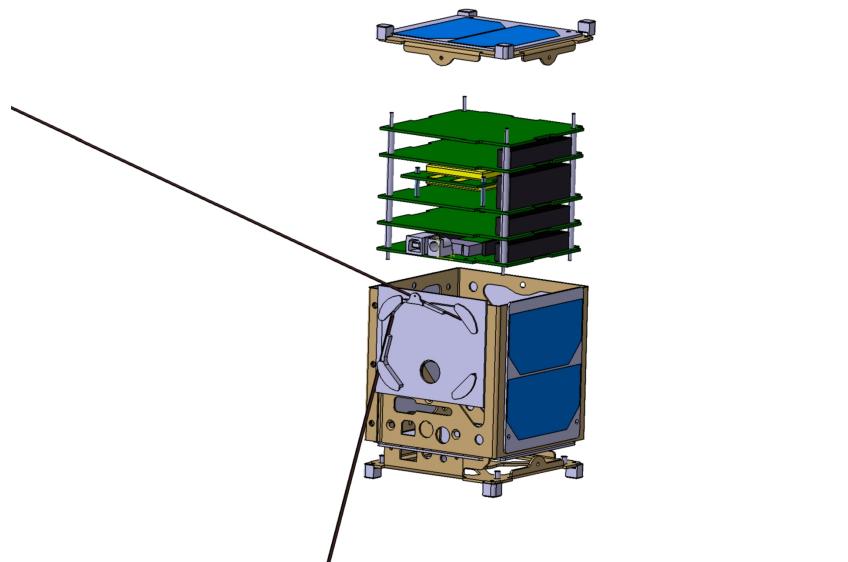


FIGURE 2.6 – Configuration d'OUFTI-1

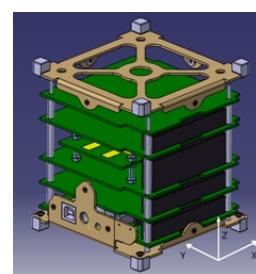


FIGURE 2.7 – Agencement des PCB

<sup>9</sup>PC/104 est une norme de PC embarqué créée par le PC/104 Consortium, qui définit une forme et une architecture (détails sur : [http://www.pc104.org/pc104\\_specs.php](http://www.pc104.org/pc104_specs.php)).

<sup>10</sup>Printed Circuit Board.

Nous retrouvons donc la structure principale fournie par Pumpkin, les cinq PCB, les cellules solaires ainsi que le système de déploiement des antennes.

Comme l'illustre la Figure 2.7, nous avons suivant le plan de base du satellite et dans l'ordre : l'OBC1, l'OBC2, l'EPS1, l'EPS2 et enfin la COM/BCN<sup>11</sup>. Les cartes sont en fait empilées et interconnectées suivant un bus. Pour fixer la structure, des plots métalliques sont vissés aux coins de chaque carte. Cette technique permet donc d'augmenter la rigidité de la structure.

Pour plus de détails concernant la structure et la configuration du satellite, il faut se référer à [4].

### Analyse de mission

L'analyse de la mission d'OUFTI-1 [5] permet de répondre à beaucoup de questions. Elle permet notamment de connaître :

- Les temps d'accès à la station sol.
- Les périodes d'éclairement et d'éclipse.
- Les caractéristiques de l'environnement spatial.
- (...)

En plus de cette liste non-exhaustive, l'analyse de mission permet également d'identifier les couples perturbateurs, d'estimer l'impact des radiations et faire aussi une estimation de la durée de vie du satellite.

### Système de déploiement des antennes

Le système de déploiement des antennes [6] est vraiment une partie critique dans sa conception. En effet, si les antennes ne se déploient pas, il se peut que le satellite ne puisse pas communiquer avec le sol. Des tests sous vide et à différentes températures ont été menés pour vérifier le *design*. Le principe consiste à enruler autour d'un support les antennes et de les déployer en mettant en fusion (comme un fusible) un conducteur qui les retient.

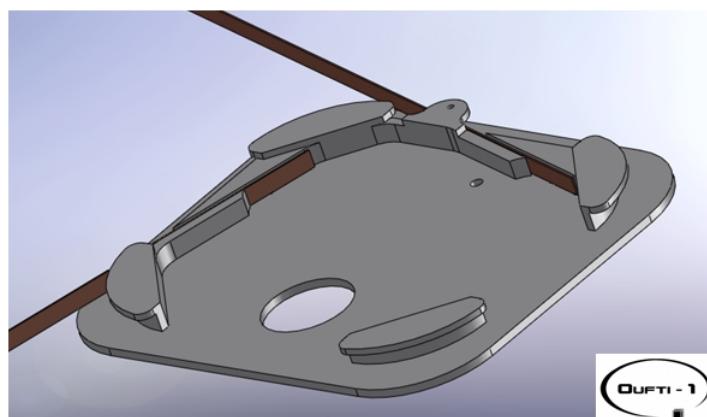


FIGURE 2.8 – Système de déploiement des antennes

---

<sup>11</sup>Communications system/Beacon.

### Contrôle thermique

L'étude thermique d'un satellite est très importante. Elle permet d'assurer la viabilité du satellite, de maintenir les composants électroniques dans leur plage de températures optimales, de réchauffer éventuellement les batteries (très sensibles au changement de température), etc. Pour y parvenir, des calculs par éléments finis et plusieurs modèles thermiques ont été réalisés [7].

### Attitude and determination control system

L'ADCS [8] est destiné à ralentir les vitesses de rotation du satellite et à l'orienter de manière optimale. Le contrôle d'attitude d'OUFTI-1 sera un contrôle d'attitude passif. L'utilisation d'aimants permanents et de matériaux hystériétiques constitue la base du système. Ce système permet de ralentir globalement les vitesses de rotation du satellite et de l'orienter approximativement suivant le champ magnétique terrestre.

### Ordinateur embarqué

L'OBC [9] est en quelque sorte le cœur du satellite. Par sécurité, il y aura deux OBC. En cas de défaillance, un système de diagnostic permettra d'éteindre l'OBC défectueux et de démarrer le second. La principale fonction de l'OBC est de faire le *monitoring* des autres sous-systèmes du satellite grâce notamment aux prises de mesures [10]. Ce *monitoring* permettra de prendre des décisions en vue de maintenir le satellite dans l'état le plus optimal possible. L'OBC devra également gérer les télécommandes reçues et rapatrier les télémétries, fournir une référence de temps et assurer les opérations initiales en orbite (actionner le déploiement des antennes, activer les sous-systèmes, etc.).

L'OBC est basé sur la technologie MSP430 de *Texas Instruments*. Un système d'exploitation temps réel sera également utilisé ; il s'agit de FreeRTOS<sup>12</sup>. Le CubeSat kit fournit déjà le modèle de vol. Il s'agit de l'FM430.



FIGURE 2.9 – FM430 du CubeSat kit

### Alimentation électrique

L'alimentation du satellite est également un sous-système important. OUFTI-1 sera composé d'une EPS robuste [11] et d'une EPS expérimentale [12].

La première devra fournir la puissance électrique grâce aux panneaux solaires. Elle devra également stocker l'énergie en surplus, provenant des panneaux solaires, dans les batteries pour la refournir ultérieurement (en période d'éclipse ou lorsque la demande en puissance est

<sup>12</sup><http://www.freertos.org/>

importante). L'EPS est donc un système très critique. En tout, trois tensions seront utilisées dans le système OUFTI-1 : 3,3 V, 5 V et 7,2 V.

L'EPS expérimentale est une *payload* du CubeSat. Elle intégrera un convertisseur *Flyback* contrôlé de manière digitale. Ce contrôle sera l'intérêt principal de cette EPS. Il sera chargé de réguler la tension de 3,3V. Cette tension alimentera soit le bus 3,3 V du satellite OUFTI-1, soit une charge résistive définie pour faire des mesures. Ce sera à l'OBC d'allumer ou d'éteindre l'EPS expérimentale mais aussi de choisir ce qu'elle alimentera en sortie.

### Système de télécommunication

Le système de télécommunication [13] [14] du satellite met en oeuvre le protocole D-STAR mais aussi l'AX.25 et une balise morse (BCN - *Beacon*). Le *design* de cette carte est basé sur un composant particulier : l'ADF7021. Au niveau du D-STAR, les principales fonctions de la COM sont de :

- Démoduler et décoder les trames D-Star venant du sol.
- Encoder et moduler les trames D-STAR à destination du sol.

Ces deux fonctions sont les deux fonctions fondamentales d'un répéteur. Pour l'AX.25, le même composant sera utilisé. Cependant, l'AX.25 sera utilisé pour transporter les télécommandes et les télémétries du satellite.

Enfin, une balise émettant un message morse fera partie du système de télécommunication. La balise sera le moyen de télécommunication primitif du satellite.

## 2.5 L'environnement spatial

L'environnement dans lequel sont plongés les satellites est complètement différent de celui que nous connaissons sur Terre. Il est important de connaître les phénomènes de haute atmosphère [3] car ils influencent fortement le choix des matériaux, des composants électroniques, des techniques utilisés, etc. L'agressivité de l'environnement spatial dépend également de deux grands paramètres : l'orbite et le type de mission.

### Le freinage atmosphérique

Le satellite OUFTI-1 sera placé sur une orbite basse. La première contrainte qui vient à l'esprit est la présence d'atmosphère résiduelle freinant le satellite. Les satellites se déplacent avec une vitesse proche des 20000 km/h. Par conséquent, OUFTI-1 subira un freinage important et finira par tomber très vite. Sa mission sera donc très courte.

Autre conséquence de cette atmosphère résiduelle est l'effet d'érosion dû à l'oxygène atomique. L'oxygène sur Terre se trouve sous la forme  $O_2$ . En haute altitude,  $O_2$  est dissocié par le rayonnement UV<sup>13</sup> en oxygène atomique  $O$  particulièrement réactif. Cet oxygène atomique provoque des dégâts considérables sur les matériaux tel que le Kapton par exemple.

### Le vent solaire

Phénomène résultant d'éruptions solaires, le vent solaire est une éjection de plasma pouvant atteindre des vitesses très rapides. Les conséquences du vent solaire sont non-négligeables. Il participe notamment aux dégradations des semi-conducteurs et plus particulièrement aux dégradations des cellules photovoltaïques directement exposées.

<sup>13</sup>Ultraviolet.

### Le rayonnement corpusculaire

D'origine solaire, le rayonnement corpusculaire correspond à l'émission de particules chargées de haute énergie (électron, proton et ion lourd). Ces particules sont en grande partie piégées dans les ceintures de Van Hallen. Cependant, les ions lourds, particulièrement pénétrants peuvent détruire complètement les semi-conducteurs. Les circuits à haute intégration sont particulièrement exposés à ce phénomène. Les énergies peuvent atteindre plusieurs MeV et plus d'un GeV pour les ions lourds.

### Le rayonnement électromagnétique

Parmi ces rayonnements, nous retrouvons les grandes longueurs d'onde, l'infrarouge, le visible, les ultraviolets et les rayonnements  $x$  et  $\gamma$ . L'infrarouge, le visible et les UV entrent pour une part importante dans le bilan thermique. Les UV ont également la particularité de rompre les liaisons polymériques de certains matériaux polymères qui les rendent bruns voir noirs. Ce phénomène amplifie donc davantage l'absorption thermique.

Concernant les rayonnements  $x$  et  $\gamma$ , ceux-ci peuvent altérer les semi-conducteurs mais de manière plus localisée.

### Les débris spatiaux et météorites

De nombreux cailloux, grains ou poussières sont en orbite autour de la Terre. Egalement placé en orbite se trouve un nombre extraordinaire d'objets artificiels : écrous, visse, gants, cellules solaires, anciens satellites, déchets biologiques, carlingues de fusée, débris d'engins spatiaux, etc. Ces objets sont en permanence suivis par les radars de la NORAD<sup>14</sup> car ils peuvent être très destructeurs pour les engins spatiaux. En effet, avec leur vitesse, ce sont véritablement des vraies balles de fusil.

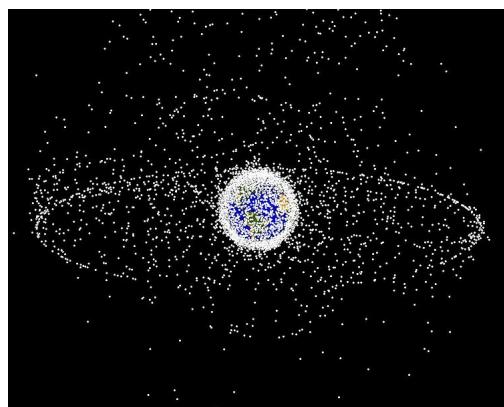


FIGURE 2.10 – Débris spatiaux autour de la Terre

Pour donner un exemple, l'un des hublots de la navette *Challenger* fut frappé par un éclat de peinture. L'impact laissa un cratère de 5 mm de diamètre.

### L'effet du vide

L'effet du vide est également non-négligeable. En effet, au niveau thermique, l'absence de molécule gazeuse se traduit par une absence de convection. L'équilibre thermique d'OUFTI-1 devra donc se faire par conduction avec la structure mécanique ou par rayonnement.

<sup>14</sup>North American Aerospace Defense Command.

Au niveau des matériaux, tous les matériaux dont la pression de vapeur est supérieure à la pression ambiante dégagent. Le choix des matériaux est donc prédominant étant donné que la pression en haute atmosphère est très faible (inférieure à 1 hPa).

Mis à part le phénomène de dégazage, il y également le phénomène de microsoudure. Si deux pièces du même matériau ou de deux matériaux solubles sont mises en contact, il en résulte une soudure directe empêchant tout mouvement relatif.

### Choix des composants semi-conducteurs

L'électronique est toujours une partie très sensible et a fortiori si celle-ci se trouve dans un environnement spatial. En effet, les composants devront faire face aux radiations, aux changements de température et au dégazage. Il existe bien entendu des composants électroniques capables de résister à de telles agressions : ce sont les composants qualifiés spatiaux. Le prix de ces composants est extrêmement élevé. Les composants semi-conducteurs des CubeSats sont donc généralement des composants *off-the-shelf*<sup>15</sup>.

Au niveau des températures de fonctionnement, nous utiliserons des composants qualifiés pour les températures industrielles ; c'est à dire : entre -40 et 85 °C. Cependant, ces composants ne sont pas prévus pour résister à l'environnement spatial si bien que plusieurs phénomènes peuvent apparaître :

- Dégradation progressive du composant jusqu'à sa destruction.
- *Bit error* ou *bit flipping* causé par les radiations dans les mémoires.
- *Latchup* entraînant un court-circuit.

Pour compléter les critères concernant le choix des composants semi-conducteurs ; la taille, la disponibilité et la consommation du composant sont également des aspects à prendre en compte. En effet, dans le cas d'OUFTI-1, la surface utile des PCB est limitée et la puissance disponible est de l'ordre du watt.

---

<sup>15</sup>Désigne un composant fabriqué en grande série et non pour un projet en particulier.

# Chapitre 3

## Cahier des charges

Présenté sous forme de cahier des charges, ce chapitre vise à fixer le contexte et les objectifs du travail de fin d'études.

### 3.1 Les télécommandes et télémétries

Lorsqu'un satellite est mis en orbite, il est évident qu'il faut pouvoir le contrôler à distance. Ce contrôle se fait par l'intermédiaire de télécommandes. La définition de ces télécommandes et de sa mise en oeuvre est vraiment délicate. En effet, si le satellite ne reçoit pas les télécommandes ou ne sait pas les interpréter, la mission est un véritable échec. Les télécommandes ont donc pour but de garder le contrôle du satellite et d'actionner les dispositifs présents à bord.

Dans le même sens, il est également important de connaître les états des systèmes à bord, de rapatrier sur Terre les résultats des expériences, etc. C'est le rôle des télémétries. Les télémétries sont aussi très importantes car elles fournissent des informations précieuses pour diagnostiquer d'éventuels problèmes à bord.

Pour communiquer avec le satellite, les TM/TC<sup>1</sup> sont envoyés via un protocole sur les bandes de fréquences radios UHF<sup>2</sup> et VHF<sup>3</sup>. Dans le cas du satellite OUFTI-1, l'*uplink* utilisera les bandes UHF et le *downlink* les bandes VHF. L'*uplink* correspond donc aux TC et le *downlink* aux TM.

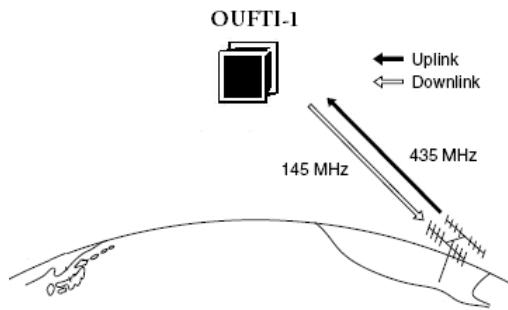


FIGURE 3.1 – Liaison satellite

La définition des TM/TC passe d'abord par la définition d'un format. Le format des TM/TC définit l'agencement et la forme que vont prendre ces dernières dans les trames.

<sup>1</sup>Télémétries/Télécommandes.

<sup>2</sup>Ultra High Frequency.

<sup>3</sup>Very High Frequency.

## 3.2 Les TM/TC et l'AX.25

L'AX.25 a été défini comme étant le protocole destiné à communiquer avec le satellite. Cette décision a été prise pour une raison principale : l'effet Doppler.

### 3.2.1 L'effet Doppler

Les satellites se déplacent à des vitesses assez impressionnantes. A cause de sa vitesse (relative), les fréquences d'*uplink* et *downlink* du satellite devient légèrement. Ce phénomène est, par définition, l'effet Doppler<sup>45</sup>. Le phénomène peut être décrit selon :

$$f = f_0 \sqrt{\frac{c - r}{c + r}} \quad (3.1)$$

où  $f$  est la fréquence déviée en réception,  $f_0$  est la fréquence de transmission,  $c$  la vitesse de la lumière et  $r$  la vitesse relative de l'émetteur par rapport au récepteur.

Cette déviation de fréquence doit être compensée pour que l'utilisation du D-STAR soit optimale. En effet, les équipements<sup>6</sup> D-STAR au sol ne possèdent aucun moyen pour compenser l'effet Doppler. La correction de fréquence devra donc se faire à bord d'OUFTI-1 et devra être complètement transparente pour l'utilisateur D-STAR.

### 3.2.2 Solution

Pour pallier à ce problème d'effet Doppler, une solution a été proposée. Chaque radioamateur aura la possibilité de réserver via un site web une passe de l'orbite du satellite. Lorsque l'utilisateur sera enregistré et en fonction de sa position sur le globe, le satellite compensera l'effet Doppler pour sa station dans une zone bien définie. Il en sera de même pour son interlocuteur.

En résumé, le principe consiste à définir deux zones compensées dans le *footprint*<sup>7</sup> du satellite.

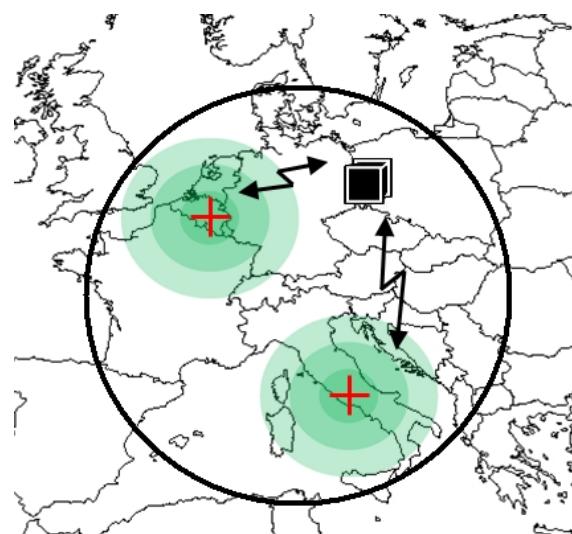


FIGURE 3.2 – Zones compensées Doppler dans le *footprint*

<sup>45</sup>Phénomène également connu avec les ondes sonores et lumineuses (Effet Doppler-Fizeau).

<sup>5</sup>Se référer à [5] pour plus de détails.

<sup>6</sup>Actuels.

<sup>7</sup>Zone de visibilité du satellite.

Ces deux zones seront compensées instantanément grâce à des tables de correction de fréquences. En effet, connaissant la vitesse relative du satellite en tout point du globe (via simulation) et en fonction de la localisation des deux stations, des tables de correction Doppler seront calculées et envoyées au satellite. Grâce à ces tables, les fréquences seront corrigées à chaque instant pour les deux stations. Par conséquent, une TC contenant ces tables devra être envoyée à OUFTI-1 depuis le sol (Liège). Cependant, au moment de l'envoi de cette TC, la fréquence ne sera pas encore corrigée à bord. Par conséquent, lors de l'envoi de cette TC, la déviation de fréquence devra donc être obligatoirement compensée sur le *transceiver* au sol. Heureusement, il existe des moyens prévus à cet effet.

Enfin, pour résumer, étant donné qu'il existe aucun matériel pour compenser l'effet Doppler avec le protocole D-STAR, la compensation devra s'effectuer via des tables de correction envoyées grâce à un autre protocole et du matériel existant prévu à cet effet. Le choix s'est donc porté sur l'AX.25. L'AX.25 servira donc à envoyer les tables de correction d'effet Doppler et plus généralement recevoir/envoyer les TM/TC.

### 3.3 Objectifs du travail de fin d'études

Mis à part la compensation de l'effet Doppler qui ne fait pas l'objet de ce travail de fin d'études, différents objectifs ont été définis. Ils peuvent être classés dans deux grandes catégories :

#### Objectifs principaux

Les principaux objectifs du travail de fin d'études sont les suivants :

1. Etude du protocole AX.25 et de sa mise en oeuvre pour le nanosatellite OUFTI-1.
2. Définition du format des TM/TC.
3. Ecriture des routines AX.25 pour le nanosatellite OUFTI-1.
4. Tests des routines AX.25 avec du matériel au sol prévu à cet effet.
5. Valider le composant ADF7021 en AX.25.

#### Objectifs secondaires

Pour compléter les objectifs, deux objectifs secondaires sont ajoutés :

1. Implémenter les routines AX.25 dans l'OBC.
2. Tester les routines implémentées avec du matériel au sol.

## **Deuxième partie**

### **Définition et description des concepts**

# Chapitre 4

## Description du protocole AX.25

Ce chapitre décrit en détail le format de la trame AX.25 ainsi que sa mise en oeuvre pour le satellite OUFTI-1. Pour y parvenir, [15] a été la principale source de ce chapitre. Les lignes qui suivent sont donc les prérequis pour une bonne compréhension des chapitres suivants.

### 4.1 Généralités

Le protocole AX.25 est un protocole dérivé du protocole X.25. L'AX.25 a été conçu par les radioamateurs pour la communauté radioamateur. En effet, la communauté radioamateur exprimait la volonté de définir un mécanisme de transport fiable des informations entre deux terminaux. Le protocole AX.25 v2.2 fournit ce service. Souvent appelé *Packet Radio*, l'AX.25 a été vite oublié par l'arrivée d'internet. Une station *Packet Radio* classique est constituée d'un ordinateur, d'un modem et d'un émetteur avec une antenne. Généralement, l'ordinateur est connecté à un *terminal node controller* qui encode/décode le format.

Le protocole est conforme à l'ISO<sup>1</sup> et HDLC<sup>2</sup>. Celui-ci suit également les recommandations du CCITT<sup>3</sup>. Ce protocole est à la base du réseau de *packet* et d'APRS<sup>4</sup>. Ce type de protocole est typiquement utilisé dans les bandes de fréquence UHF et VHF pouvant atteindre des débits de 300 à 19200 baud. Le signal, quant à lui, subit un modulation FSK<sup>5</sup>.

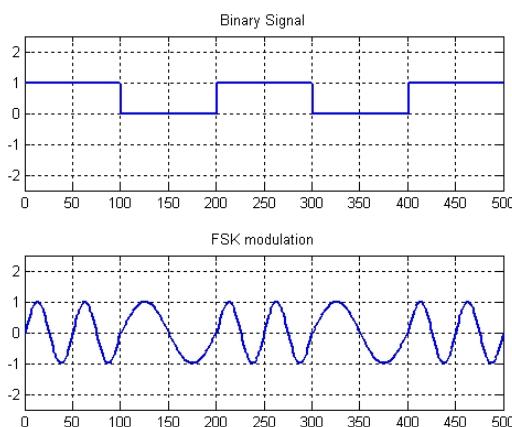


FIGURE 4.1 – Illustration de la modulation FSK

<sup>1</sup>International Organization for Standardization.

<sup>2</sup>High Level Data Link Control.

<sup>3</sup>Consultative Committee in International Telegraph and Telephone.

<sup>4</sup>Automatic Position Reporting System.

<sup>5</sup>Frequency Shift Keying.

Ce protocole a été conçu également pour fonctionner avec des connexions directes entre deux stations amateurs individuelles ou une station individuelle et un contrôleur multi-voies.

L'AX.25 fonctionne aussi bien en half qu'en full-duplex. Au niveau du modèle OSI<sup>6</sup>, l'AX.25 occupe les deux premières couches à savoir : la couche physique et la couche de liaison de données. Cependant, différents services peuvent être offerts à la couche 3 via un DLSAP<sup>7</sup>.

### 4.1.1 Modèles AX.25

Enoncé plus haut, le protocole AX.25 occupe les deux premières couches du modèle OSI. Ces deux couches peuvent être encore subdivisées en plusieurs sous-couches.

Comme le montre la Figure 4.2, différentes entités existent dans chaque couche : Segmenter, Data Link, etc. Suivant les informations venant des couches supérieures, le segmenter sert, par exemple, à former les trames. Autre exemple, l'entité Data Link sert à fournir toute la logique nécessaire pour établir une connexion entre deux stations et échanger des informations en mode connecté ou non-connecté.

Layer	Function(s)	
Data Link (2)	Segmenter	Management
	Data Link	Data Link
Link Multiplexer		
Physical (1)	Physical	
	Silicon/Radio	

FIGURE 4.2 – Modèle AX.25 - *Single Link*

Remarquons également que l'AX.25 offre la possibilité de multiplexer plusieurs liaisons de données partageant le même support physique. Le modèle se résume alors à ceci :

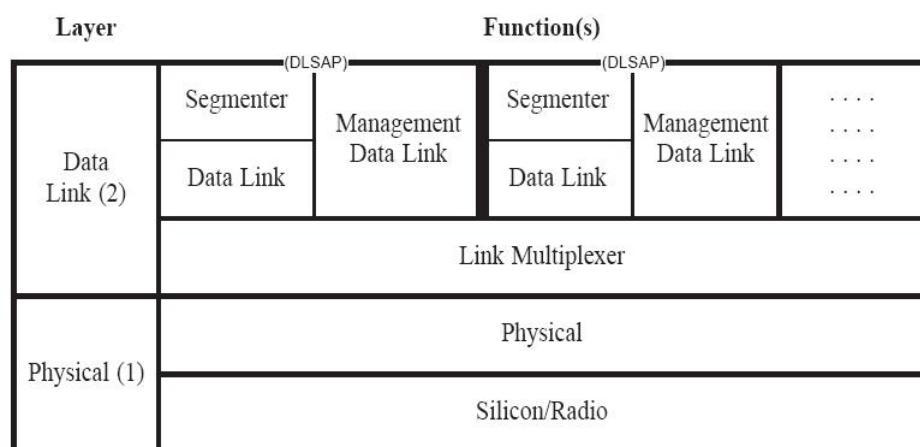


FIGURE 4.3 – Modèle AX.25 - *Multiple Stream*

<sup>6</sup>Open Systems Interconnection.

<sup>7</sup>Data-Link Service Access Point.

Les primitives qui sont échangées entre les différentes couches sont de quatre types :

- REQUEST : Cette primitive est utilisée par une couche supérieure réclamant un service d'une couche inférieure.
- INDICATION : Cette primitive est utilisée par une couche indiquant à la couche supérieure suivante d'une activité quelconque à un service.
- RESPONSE : Cette primitive sert d'acquittement d'une primitive de type INDICATION d'une couche inférieure (L'AX.25 n'utilise pas ce type de primitive).
- CONFIRM : Cette primitive fournit une confirmation que le service demandé a été accompli.

## 4.2 Structure de la trame AX.25

Le protocole AX.25 envoie ses informations sous forme de trames (*frames* en anglais). Ces blocs de données sont également appelés *Packet Radio*. Le protocole AX.25 définit trois types de trame :

1. *Information frame* (I)
2. *Supervisory frame* (S)
3. *Unnumbered frame* (U)

Chaque trame est composée de petits blocs individuels appelés champs. Les trames *S* et *U* ont la même construction. En voici les détails :

Flag	Address	Control	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	N*8 Bits	16 Bits	01111110

FIGURE 4.4 – Structure des trames U et S

La trame *I* diffère légèrement des trames *S* et *U*. Celle-ci se compose en fait d'un champ supplémentaire : PID<sup>8</sup>.

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

FIGURE 4.5 – Structure de la trame I

Les différents champs sont codés sur des bytes (ou octets). Lors de la transmission, chaque champ est envoyé suivant le bit de poids faible. Exception à la règle, le FCS<sup>9</sup> est envoyé suivant le bit de poids fort. Concernant le champ Info, celui-ci a une taille par défaut de 256 bytes (*Packet Radio* classique). Ce champ peut bien sûr être inférieur à 256 bytes suivant l'utilisation. C'est ce champ qui comportera les informations utiles destinées au satellite.

Les points qui suivent décrivent l'objet de ces champs.

<sup>8</sup>Protocol IDentifier.

<sup>9</sup>Frame CheckSum, voir Chapitre 5.

### 4.2.1 Flag

Le champ flag occupe 1 byte. Celui-ci sert, bien entendu, à délimiter les trames. Deux trames consécutives peuvent partager le même flag. La valeur de ce flag est 01111110 (0x7E). Cette suite consécutive de six bits à 1 ne peut se reproduire dans la trame étant donné que la trame subit un *bit stuffing*.

#### Bit stuffing

Le *bit stuffing* (zéro bit insertion en français) est l'insertion d'un 0 à la suite de cinq bits consécutifs à 1. Ce mécanisme est utilisé pour que la *frame sync sequence* (FSS : 0x7E) ou flag ne soit rencontré accidentellement dans la trame.

Par exemple :

0110111111001111011111111100000 (trame brute)

01101111101100111110011111011111000000 (trame « bit stuffée »)

### 4.2.2 Address

Le champ address permet d'identifier la source et la destination de la trame. Concrètement, il reprend les indicatifs radioamateurs (en ASCII<sup>10</sup> majuscule *shifté*) de la source et de la destination. Remarquons également que le champ address est variable. En effet, le protocole AX.25 permet d'encoder également de(s) indicatif(s) supplémentaire(s) dans le cas d'une trame passant à travers d'un/plusieurs répéteur(s).

#### Non-repeater mode

Dans le cas du mode *non-repeater*, le champ Address aura une longueur totale de 112 bits (14 bytes). Voici le format du champ Address :

Indicatif de la destination							Indicatif de la source						
A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14

FIGURE 4.6 – Format du champ Address en mode non-repeater

Comme le montre la Figure 4.6, seuls les indicatifs de la source et de la destination sont nécessaires dans ce mode. Chaque indicatif est codé sur 7 bytes en ASCII majuscule. Les bytes 7 et 14 sont des bytes particuliers. Ils correspondent aux SSID<sup>11</sup> des indicatifs correspondants. Le byte SSID a le format suivant : CRRSSIDH.

- Le bit C correspond au *Command/Response bit* d'HDLC. Ce bit permet en fait de maintenir la compatibilité avec les versions précédentes de l'AX.25.
- Les bits R sont des bits réservés toujours mis à 1.
- Le SSID est le numéro de la seconde station de l'indicatif.

<sup>10</sup>American Standard Code for Information Interchange.

<sup>11</sup>Second Station IDentifier.

- Le dernier bit est le bit H. Ce bit correspond à l'*extension bit* d'HDLC ([H]as-been-repeated bit). Lorsque ce bit est à 1, cela signifie qu'il s'agit du dernier byte du champ Address. Ce dernier bit implique aux autres bytes d'avoir leur bits zéros toujours à 0.

Prenons un exemple concret : source ON4ULG-0, destination OUFTI1-0.

Byte	Caractère	Bin	Shifted	Hex
A1	O	01001111	10011110	0x9E
A2	U	01010101	10101010	0xAA
A3	F	01000110	10001100	0x8C
A4	T	01010100	10101000	0xA8
A5	I	01001001	10010010	0x92
A6	1	00110001	01100010	0x62
A7	SSID	01100000	-	0x60
A8	O	01001111	10011110	0x9E
A9	N	01001110	10011100	0x9C
A10	4	00110100	01101000	0x68
A11	U	01010101	10101010	0xAA
A12	L	01001100	10011000	0x98
A13	G	01000111	10001110	0x8E
A14	SSID	01100001	-	0x61

### Repeater mode

Dans le cas du mode *repeater*, le champ Address aura une longueur maximum de 224 bits (28 bytes). Voici le format du champ Address :

Indicatif de la destination							Indicatif de la source							Indicatifs des répéteurs						
A1	A2	A3	A4	A5	A6	A7	A8	A9	A10	A11	A12	A13	A14	A15	A16	A17	...	A28		

FIGURE 4.7 – Format du champ Address en mode repeater

Dans ce mode, il s'agit du même principe que le mode précédent. Le champ est beaucoup plus étendu et le bit H doit être mis à 1 dans le byte SSID du dernier répéteur.

### 4.2.3 Control

Le champ Control identifie le type de trame et contient également des bits de contrôle pour maintenir l'intégrité de la liaison (gestion des flux via des compteurs numérotant les trames). Ce champ peut occuper 8 ou 16 bits selon le nombre de trames émises (modulo 8 ou 128). Par exemple en modulo 128, il est possible d'envoyer 127 trames en une connection.

#### 4.2.4 PID

Le champ PID est, comme son nom l'indique, l'identification du protocole de la couche 3. Dans le cas où nous n'utilisons aucun protocole en couche 3, il suffit simplement d'entrer 11110000 (0xF0) dans ce champ.

HEX	M S B	L S B	Translation
**		yy01yyyy	AX.25 layer 3 implemented.
**		yy10yyyy	AX.25 layer 3 implemented.
0x01		00000001	ISO 8208/CCITT X.25 PLP
0x06		00000110	Compressed TCP/IP packet. Van Jacobson (RFC 1144)
0x07		00000111	Uncompressed TCP/IP packet. Van Jacobson (RFC 1144)
0x08		00001000	Segmentation fragment
0xC3		11000011	TEXNET datagram protocol
0xC4		11000100	Link Quality Protocol
0xCA		11001010	Appletalk
0xCB		11001011	Appletalk ARP
0xCC		11001100	ARPA Internet Protocol
0xCD		11001101	ARPA Address resolution
0xCE		11001110	FlexNet
0xCF		11001111	NET/ROM
0xF0		11110000	No layer 3 protocol implemented.
0xFF		11111111	Escape character. Next octet contains more Level 3 protocol information.
Escape character. Next octet contains more Level 3 protocol information.		00001000	

FIGURE 4.8 – Définition des PID

#### 4.2.5 Info

Le champ Info est la partie communément appelée Data. Enoncé plus haut, ce champ a une longueur de 256 bytes par défaut. Le protocole définit également 5 types de champ Info différents :

1. *I frame - Information Frame*
2. *UI frame - Unnumbered Information Frame*
3. *XID frame - Exchange ID Frame*
4. *TEST frame*

5. *FRMR frame - Frame Reject*

#### 4.2.6 FCS

Un checksum sur 16 bits est calculé<sup>12</sup> sur la trame pour détecter une erreur éventuelle. Si une erreur est détectée, la trame est rejetée.

### 4.3 Taille d'une trame

Dans l'hypothèse d'une trame  $I$  où  $N = 256$  et que le champ Address est en mode *non-repeater*, la taille de la trame entière vaut 276 bytes :

$$L = 1 + 14 + 1 + 1 + 256 + 2 + 1 = 276 \text{ bytes} \quad (4.1)$$

### 4.4 Les modes

D'après [15] deux modes possibles se dégagent pour l'AX.25. Le premier est évident, il s'agit du mode connecté (*connection oriented*) avec la gestion des flux (trames numérotées). Le second a été moins évident à découvrir dans le document de référence ; il s'agit du mode non-connecté (*connectionless*). Pas d'acquittement lors d'une trame reçue, aucun comptage de trame, etc (analogie avec le protocole UDP<sup>13</sup>). Dans le mode non-connecté, le format de la trame AX.25 correspond uniquement aux trames *UI*.

#### 4.4.1 Mode connecté et mode non-connecté

Suite à la lecture du document de référence, nous nous apercevons vite que le mode connecté est assez lourd à implémenter pour un CubeSat et surtout pour uniquement envoyer des TM/TC.

En effet, limiter d'une part en espace mémoire et d'autre part en temps d'accès au satellite (la durée des passes peut être parfois assez courte), il serait plus judicieux d'utiliser le mode non-connecté. Le dialogue entre le satellite et la station au sol serait alors réduit à envoyer directement la TC et à recevoir la TM correspondante (gain en temps et calcul). Par contre, nous perdons tout l'avantage du contrôle de flux du mode connecté.

### 4.5 Mise en oeuvre du protocole AX.25 pour OUFTI-1

Le protocole AX.25 est un protocole radioamateur prévu pour des liaisons terrestres. Le protocole prévoit donc à cet effet, un contrôle de flux sophistiqué pour créer et gérer plusieurs connexions simultanément.

OUFTI-1 est une application spatiale. L'électronique et la puissance de calcul mise en jeu sont assez légères. Dans ce type d'application, la trame *UI* est un bon compromis. En effet, la trame *UI* a l'avantage de passer outre le contrôle de flux, d'être simple, personnalisable et tout à fait conforme au protocole AX.25. La trame *UI* est, par conséquent, souvent [27] utilisée pour les applications plus légères (comme les CubeSats) n'utilisant pas l'arsenal prévu par l'AX.25.

---

<sup>12</sup>Voir point 5.3

<sup>13</sup>User Datagram Protocol.

La mise en oeuvre du protocole AX.25 se limitera donc à l'implémentation de la trame *UI*. En voici sa mise en application :

#### 4.5.1 Trame UI

La structure de la trame *UI* est identique à celle de la trame *I*. Le remplissage des champs est par contre différent.

Flag	Address	Control	PID	Info	FCS	Flag
01111110	112/224 Bits	8/16 Bits	8 Bits	N*8 Bits	16 Bits	01111110

FIGURE 4.9 – Structure de la trame UI

La trame contient également un PID et un champ Info. Le champ PID<sup>14</sup> prendra la valeur 0xF0 (pas de protocole en couche 3). Le champ Info contiendra quant à lui les TM/TC. Le champ Adress se limitera au mode *non-repeater* et son remplissage est comparable à l'exemple du point 4.2.2.

Control Field Type	Type	Control-Field Bits						
		7	6	5	4	3	2	1
Set Async Balanced Mode	SABME	Cmd	0	1	1	P	1	1
Set Async Balanced Mode	SABM	Cmd	0	0	1	P	1	1
Disconnect	DISC	Cmd	0	1	0	P	0	0
Disconnect Mode	DM	Res	0	0	0	F	1	1
Unnumbered Acknowledge	UA	Res	0	1	1	F	0	0
Frame Reject	FRMR	Res	1	0	0	F	0	1
Unnumbered Information	UI	Either	0	0	0	P/F	0	0
Exchange Identification	XID	Either	1	0	1	P/F	1	1
Test	TEST	Either	1	1	1	P/F	0	0

FIGURE 4.10 – Spécifications du champ Control pour les trames U

Enfin, concernant le champ Control, celui-ci prendra la valeur 0x03 (le bit P/F<sup>15</sup> sera mis à 0).

#### Exemple de trame UI

Voici un exemple concret (en négligeant le *bit stuffing*) : trame *UI* émise d'OUFTI1 à destination d'ON4ULG avec 0, 1 et 2 comme informations.

0x7E, 0x9E, 0x9C, 0x68, 0xAA, 0x98, 0x8E, 0x60, 0x9E, 0xAA, 0x8C, 0xA8, 0x92, 0x62, 0x61, 0x03, 0xF0, 0x00, 0x01, 0x02, 0x93, 0x8F, 0x7E

<sup>14</sup>Voir point 4.2.4

<sup>15</sup>P/F pour *Poll/Final bit*. Lorsque ce dernier est mis à 1, une réponse immédiate doit être envoyée à la suite de cette trame.

### 4.5.2 Le CCSDS

L'AX.25 sera donc le protocole qui « transportera » les TM/TC. Cependant, il faut définir le format de ces TM/TC. Au départ, n'ayant aucune idée sur le sujet, nous<sup>16</sup> sommes partis sur une solution tout à fait arbitraire (voir Annexe 1 et 2).

Suite à plusieurs discussions et après avoir rencontré les suisses à Lausanne, il nous est apparu clair que le format CCSDS<sup>17</sup> était une solution plus qu'envisageable. En effet, le CCSDS définit un format clair et offre plusieurs services très intéressants<sup>18</sup> : référence de temps (date, heure et année) via un *time code*, le *space packet*, etc.

En plus d'être très pratique, le CCSDS est fortement recommandé par la plupart des agences spatiales (ESA, NASA<sup>19</sup>, CNES<sup>20</sup>, etc) et jouit d'une très bonne réputation. Cependant son implémentation se fait à la fois au sol et dans le satellite. Pour mettre en place une telle structure, un travail de longue halène est nécessaire (plusieurs mois). Par conséquent, l'implémentation de ce format ne fera pas l'objet de ce travail de fin d'études.

## 4.6 Avantages et inconvénients de l'AX.25

Déjà dit plus haut, la mise en oeuvre du protocole AX.25, pour le satellite OUFTI-1, se limitera à l'implémentation de la trame *UI*. Cette mise en oeuvre est à la fois simple, légère et flexible pour accueillir le format CCSDS. Le protocole AX.25 a également l'avantage d'être compatible avec n'importe quelle station AX.25 à travers le monde. L'AX.25 permet également d'éviter les problèmes d'effet Doppler dont nous avons déjà parlés.

Comparativement au D-STAR qui possède tout un artifice pour corriger les erreurs (*interleaving*, algorithme de Viterbi, convolution), l'AX.25 ne possède aucun moyen de corriger les erreurs. En effet, la trame ne possède qu'un FCS capable, uniquement, de détecter des erreurs. La perte de la gestion des flux est également un problème. Cependant, le CCSDS implémente un système haut niveau d'acquittement des TM/TC.

Pour conclure, la trame *UI* du protocole AX.25 est tout de même un bon compromis technologique pour un satellite amateur.

<sup>16</sup>En collaboration avec Laurent Chiarello.

<sup>17</sup>The Consultative Committee for Space Data Systems : <http://public.ccsds.org/>.

<sup>18</sup>Se référer à [16] pour plus de détails concernant le CCSDS et ses services.

<sup>19</sup>National Aeronautics and Space Administration.

<sup>20</sup>Centre National d'Etudes Spatiales.

# Chapitre 5

## Frame checksum AX.25

Le protocole AX.25 peut détecter des erreurs inhérentes dans ses trames. Cette détection d'erreur se fait par l'intermédiaire d'un contrôle de redondance cyclique<sup>1</sup>. Ce CRC est calculé uniquement sur les bits qui se trouvent entre les flags. Lorsqu'il s'agit d'un CRC calculé sur une trame, le CRC est aussi appelé *frame checksum* (FCS). L'implémentation du calcul de ce CRC a notamment été réalisée grâce à [17] et [18].

### 5.1 Principe

Soit un message de  $N$  bits à transmettre. Considérons maintenant ce message binaire sous la forme d'un polynôme de degré  $N-1$  :  $P(x)$ . Par définition, le CRC est le reste  $R(x)$  de la division booléenne de  $P(x)$  par un polynôme dit générateur ;  $G(x)$ . Ce CRC est ensuite concaténé au message envoyé.

A la réception du message, le CRC est recalculé et comparé à celui envoyé. Si les CRC sont différents, il y a par conséquent une erreur dans le message. Cependant, une autre méthode existe. Si nous considérons  $D$  comme étant le dividende,  $d$  le diviseur et  $R$  le reste de la division ; l'opération suivante donne un reste nul :

$$\frac{D - R}{d} \tag{5.1}$$

En arithmétique binaire, l'addition binaire est équivalente à la soustraction binaire (à une opération près). Cette dernière considération implique que lors de la réception du message et du CRC, il suffit uniquement d'effectuer :

$$\frac{P(x) + R(x)}{G(x)} \tag{5.2}$$

Si le reste de cette opération est nul, aucune erreur n'est constatée.

---

<sup>1</sup>Cyclic Redundancy Check (CRC).

## 5.2 Caractéristiques du FCS AX.25

Le FCS de l'AX.25 est un CRC calculé sur 16 bits suivant les recommandations d'HDLC (ISO 3309). En parcourant la littérature et notamment le RFC 1662 [19], nous apprenons que le CRC est un CRC de type CCITT de polynôme générateur :

$$G(x) = x^{16} + x^{12} + x^5 + 1 \quad (5.3)$$

La valeur binaire correspondant à  $G(x)$  vaut 000100000100001 ou 0x1021 en hexadécimal. La valeur initiale du CRC doit être égale à 0xFFFF. Le protocole AX.25 précise également que le bit de poids fort du champ FCS doit être envoyé en premier.

## 5.3 Implémentation

L'implémentation de ce CRC se résume aux opérations XOR et Shift. Cette implémentation est réalisée grâce à un registre à décalage et une opération XOR faisant office de division binaire. La figure qui suit montre le schéma d'implémentation pour l'AX.25.

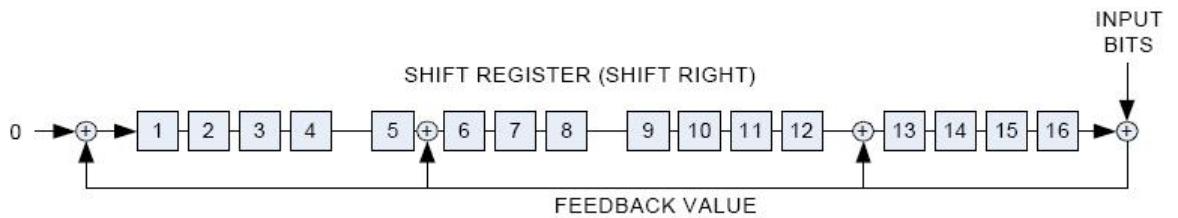


FIGURE 5.1 – Principe du CRC-16-CCITT

Le registre à décalage est initialisé à 0xFFFF et décale vers la droite. Lorsque le registre décale, un 0 se prépare à rentrer. L'opération XOR s'effectue de la manière suivante : à la suite du décalage, le 0 rentrant à gauche subit une opération XOR avec la valeur feedback ; le résultat étant stocké dans le bit 1 (idem pour les autres XOR). La valeur feedback est l'opération XOR entre les bits du message en entrée et le bit 16 décalé<sup>2</sup>.

Enfin, en décalant et en bouclant au fur et à mesure sur tous les bits du message, nous obtenons le CRC final.

## 5.4 Code source

Rappelons que le protocole AX.25 spécifie que le premier bit à transmettre doit être le bit de poids faible. Pour des raisons de facilité étant donné que le bit de poids fort du FCS doit être envoyé le premier, nous utiliserons l'ordre suivant : *reflected bit order*.

Le bit zéro d'un byte, communément appelé bit de poids faible, sera remplacé par le bit de poids fort. Le byte obtenu est donc l'image miroir du byte initial. Exemple :

00000101 devient 10100000

Dans ces conditions, lors de la transmission de la trame AX.25, le bit de poids fort du FCS sera envoyé en premier. Pour arriver à ce résultat, nous prendrons l'image miroir du polynôme générateur. Initialement, celui-ci valait 0x1021. Une fois *reflected*, celui-ci vaut 0x8408.

<sup>2</sup>Appelé parfois *carry*.

Enfin, voici l'implémentation du CRC en C :

```
/*
 * AX.25 FCS calculation (bitwise method).
 * CRC-16-CCITT G(x) = x16 + x12 + x5 + 1.
 * Polynom = 0x1021.
 *
 * PARAMETERS:
 * * buffer      pointer of the frame buffer.
 * size_frame   length of the frame (in bytes).
 *
 * RETURN:
 * the CRC with a final XORed operation.
 */
unsigned short AX25_computeCRC(char *buffer, unsigned short size_frame){
    unsigned int i, j;
    unsigned short shiftRegister, outBit;
    char byte;

    // The last flag and the 2 bytes for FCS are removed.
    size_frame = size_frame - 3;

    // Initialization of the Shift Register to 0xFFFF
    shiftRegister = 0xFFFF;

    for(i=1 ; i<size_frame; i++) { // The first flag is not calculated so i=1.
        byte = buffer[i];

        for(j=0; j<8; j++) {
            outBit = shiftRegister & 0x0001;
            shiftRegister >>= 0x01; // Shift the register to the right.

            if(outBit != (byte & 0x01)) {
                shiftRegister ^= 0x8408; // Mirrored polynom.
                byte >>= 0x01;
                continue;
            }
            byte >>= 0x01;
        }
    }
    return shiftRegister ^ 0xFFFF; // Final XOR.
}
```

Il existe une autre méthode pour calculer ce CRC. Cette méthode consiste à utiliser un tableau de registres pré-calculés<sup>3</sup>. Cette solution permet d'optimiser l'algorithme au niveau de sa vitesse d'exécution. Cependant, la table pré-calculée occupe un espace mémoire assez conséquent : 256 éléments de 16 bits. Un compromis doit être fait entre espace mémoire et utilisation du processeur. Dans le cas de l'OBC, nous préconisons d'occuper le moins d'espace mémoire possible.

---

<sup>3</sup>Voir Annexe 3

## 5.5 Vérification

Grâce à [17], nous pouvons vérifier que notre calcul de CRC fonctionne correctement. Reprenons la fonction MATLAB de [17] et notre fonction avec un exemple concret. Soit le message suivant à transmettre :

0x9E 0x9C 0x60 0xAA 0xEC 0x8E 0x60 0x9E 0xAA 0x8C 0xA8 0x92 0x62 0x61 0x03 0xF0  
0xFF 0xAA

Sous forme binaire :

```
0 1 1 1 1 0 0 1 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 0 0 1 0 1 0 1 0 1 0 1 0 0 1 1 0 1 1 1 0 1 1 1 1 0 0 0 1 0  
0 0 0 0 1 1 0 0 1 1 1 1 0 0 1 0 1 0 1 0 1 0 1 0 1 0 0 1 1 0 0 0 1 0 0 0 1 0 1 0 1 0 1 0 1 0 0 1 0 1  
0 0 0 1 1 0 1 0 0 0 0 1 1 0 1 1 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 1 0 1 0 1 0 1
```

Après calcul, MATLAB nous donne le résultat suivant :

FCS =

1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1

>>

Concernant notre fonction, celle-ci nous donne le résultat suivant :

0x8601

Le bit de poids fort du CRC est donc le bit de poids faible de 0x8601. En prenant la précaution d'envoyer ce dernier en premier par bytes successifs, le CRC vaut :

0x01 0x86

L'opération est effectuée par le code qui suit :

```
/*-----*
 * AX.25 FCS positioning.
 * Put the FCS in the right place in the frame. The FCS is sent MSB first
 * so we prepare the 15th bit of the CRC to be sent first.
 *
 * PARAMETERS:
 *   *frame      pointer of the frame buffer.
 *   size_frame  length of the frame (in bytes).
 *-----*/
void AX25_putCRC(char *frame, unsigned short size_frame){
    unsigned short crc;

    // FCS calculation.
    crc = AX25_computeCRC(frame, size_frame);

    // Put the FCS in the right place with the 15th bit to be sent first.
    frame[size_frame - 3] = (crc & 0xff);
    frame[size_frame - 2] = ((crc >> 8) & 0xff);
}
```

Enfin, comme le bit de poids fort du FCS est correctement placé, la transmission de la trame entière peut s'effectuer selon le bit de poids faible ; ce qui donne pour le FCS :

1000000001100001

Le résultat est équivalent à celui calculé par notre document de référence [17].

# Chapitre 6

## Le terminal node controller

Le *terminal node controller* (TNC) est un dispositif qui fait l'interface entre le *transceiver* radio et un terminal. Ce terminal est souvent un ordinateur. Dans notre cas, le TNC sera interfacé via USB<sup>1</sup> avec l'ordinateur.

Ce chapitre met en évidence l'utilisation du TNC pour le protocole AX.25 mais aussi sa mise en oeuvre.

### 6.1 Principe général de fonctionnement

Le TNC est un dispositif qui est totalement transparent vis-à-vis de l'utilisateur. Il fait en fait le lien entre l'utilisateur AX.25 et le *transceiver* radio. Par exemple, lors de la transmission d'une trame AX.25, l'utilisateur doit simplement lui envoyer les informations utiles à transmettre. Ensuite, le TNC se charge du reste. Le TNC rassemble en fait en une seule unité les fonctions de modem et de gestion du protocole. Suivant les baud rates et l'utilisation que l'on en fait, le TNC peut être très utile et facile à manipuler pour un novice.

La plupart du temps, le TNC se compose d'un microprocesseur et d'un modem. Le principe général est le suivant : lors de la réception d'un signal UHF ou VHF, le *transceiver* radio démodule le signal et émet un signal audio vers le TNC. Ce signal audio est ensuite décodé par le TNC et les informations utiles sont renvoyées au terminal via un port série. Le raisonnement inverse est également valable pour la transmission.

Cette explication simpliste sera détaillée au fil de ce chapitre.

### 6.2 Les baud rates

Les baud rates de l'AX.25 sont compris entre 300 et 19200 baud. Suivant le baud rate utilisé, la modulation peut différer. En effet, entre 300 et 1200 baud compris, la modulation est de type A-FSK<sup>2</sup>. Au-delà de 1200 baud, la modulation est de type FSK classique.

Après plusieurs recherches sur le sujet, les principaux baud rates utilisés pour les CubeSats sont soit le 1200 baud ou le 9600 baud. Le 1200 baud étant le baud rate historique de l'AX.25, celui-ci est le plus répandu. Ceci s'explique également par le fait qu'il est plus aisément à implémenter par rapport au 9600 baud. En effet, le 9600 baud se module en FSK mais passe d'abord par le modem G3RUH<sup>3</sup>.

---

<sup>1</sup>Universal Serial Bus.

<sup>2</sup>Audio - Frequency Shift Keying.

<sup>3</sup>Voir point 6.3.2

Voici quelques exemples de CubeSat :

CubeSats	Nations	Modulations et baud rates
Cute-I	Japon	AFSK 1200 baud
QuakeSat	USA	FSK 9600 baud
CUTE 1.7	Japon	AFSK 1200 baud
GENESAT-1	USA	AFSK 1200 baud
CAPE-1	USA	FSK 9600 baud
SwissCube	Suisse	AFSK 1200 baud
COMPASS-1	Allemagne	AFSK 1200 baud
ROUSTA	France	AFSK 1200 baud

## 6.3 Les modulations

Comme nous l'avons dit plus haut, le 1200 baud se module en A-FSK et le 9600 baud en FSK. Remarquons tout de même que nous pourrions tout à fait choisir une autre modulation. Cependant, nous devrions adapter notre matériel à cette modulation. Ces adaptations demanderaient du temps en recherche et développement supplémentaire et une non-standardisation du protocole à travers le monde.

### 6.3.1 1200 baud A-FSK

En 1200 baud A-FSK, la transmission des données est basée sur le modem Bell 202 [20]. La sortie du modem est directement appliquée sur l'oscillateur de l'émetteur (varactor) du *transceiver* radio. Pour transmettre les informations, le modem utilise deux fréquences discrètes (appelées aussi tonalités). La tonalité *mark* est conventionnellement fixée à 2200 Hz, et la tonalité *space* à 1200 Hz.

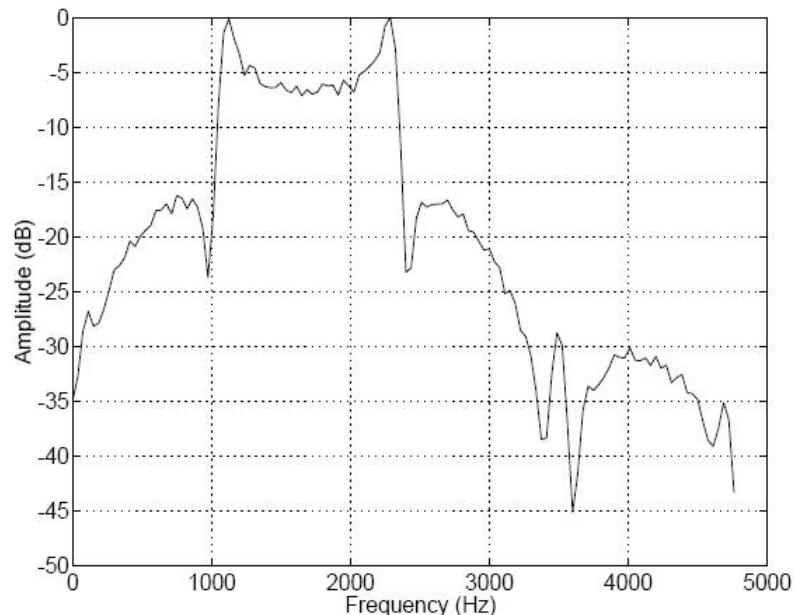


FIGURE 6.1 – Spectre du modem Bell 202

Il s'agit donc d'une transmission en bande de base. En respectant le critère de Nyquist-Shannon, ce signal est tout à fait audible. Le son est proche du son qu'émet un vieux modem

56k lors de sa connexion. Cependant, ce type de modulation réduit fortement le baud rate si bien que 1200 baud est un maximum conseillé. Par curiosité, un essai a été réalisé. L'essai consistait à visualiser la forme du signal sous MATLAB. Pour ce faire, il suffit simplement de fabriquer un câble audio mini-DIN 6 pins/Audio Jack.

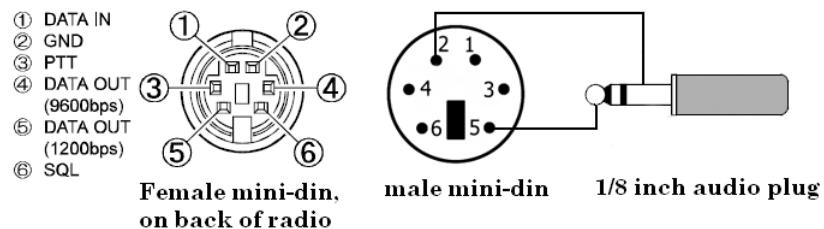


FIGURE 6.2 – Schéma du câble audio 1200 baud

Une fois le câble soudé, il ne reste plus qu'à le connecter à la sortie d'un TNC et à une carte son qui échantillonne le signal.



FIGURE 6.3 – Schéma du câble audio 1200 baud (bis)

Lorsque le fichier \*.wav est enregistré, nous pouvons le traiter dans MATLAB. Et voici le résultat :

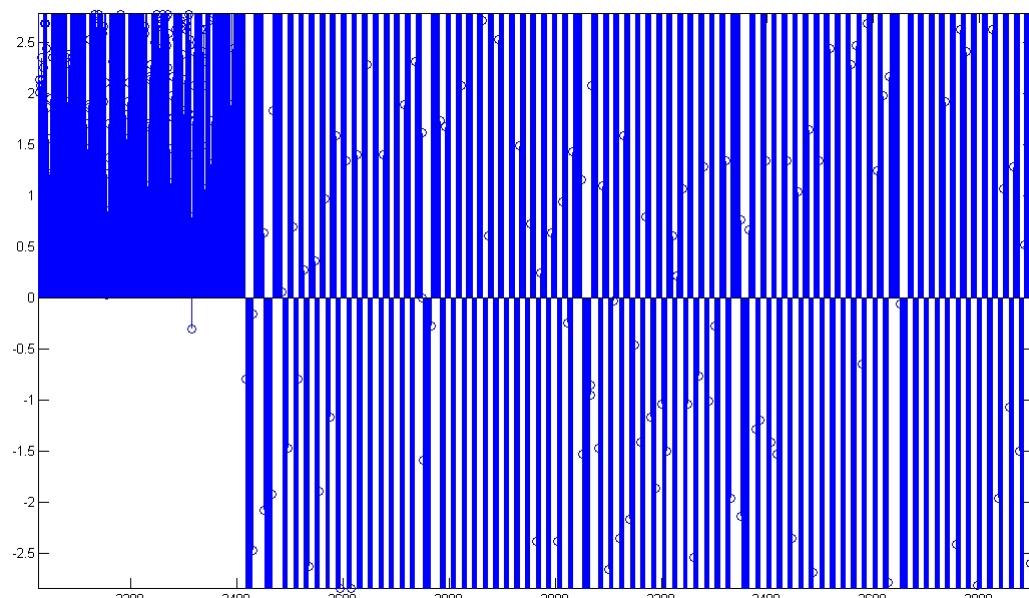


FIGURE 6.4 – Signal de sortie du TNC en 1200 baud

Le signal observé est le début d'une trame AX.25 rempli de flags (suite de 01111110, 0x7E). Il semble que le signal soit de type carré. Nous pouvons également observer la différence de tonalité (fréquence) lors d'un passage à 0. L'AX.25 1200 baud est donc très facile à implémenter.

### 6.3.2 9600 baud FSK

En 9600 baud FSK, le principe est un peu différent. En FSK, les états logiques binaires sont aussi associés à des fréquences. En revanche, ces fréquences correspondent à des déviations (positives ou négatives) de fréquences autour d'une fréquence principale appelée porteuse. En réception, un filtre permet de restituer le message binaire d'origine.

Il existe en fait deux grands types de modulation FSK : la FSK cohérente et la FSK non-cohérente.

#### FSK non-cohérente

En FSK non-cohérente, la phase n'est pas continue lors du changement de fréquence :

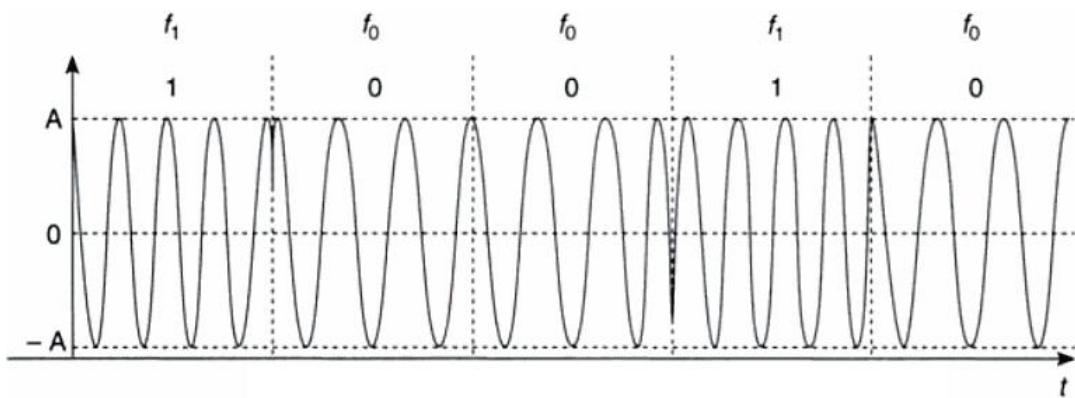


FIGURE 6.5 – Illustration de la FSK non-cohérente

La FSK non-cohérente peut se modéliser par deux oscillateurs de fréquences ( $f_0$  et  $f_1$ ) non-synchrones qui commutent suivant l'état logique 1 ou 0. Le spectre<sup>4</sup> prend donc l'allure suivante :

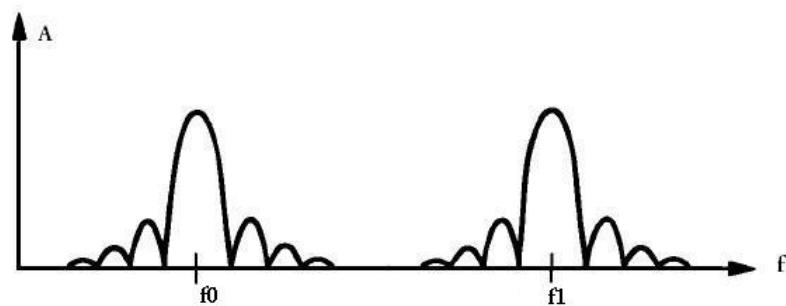


FIGURE 6.6 – Spectre de la modulation FSK non-cohérente

<sup>4</sup>Pour un signal carré.

### FSK cohérente

En FSK cohérente<sup>5</sup>, la phase est par contre continue lors du changement de fréquence :

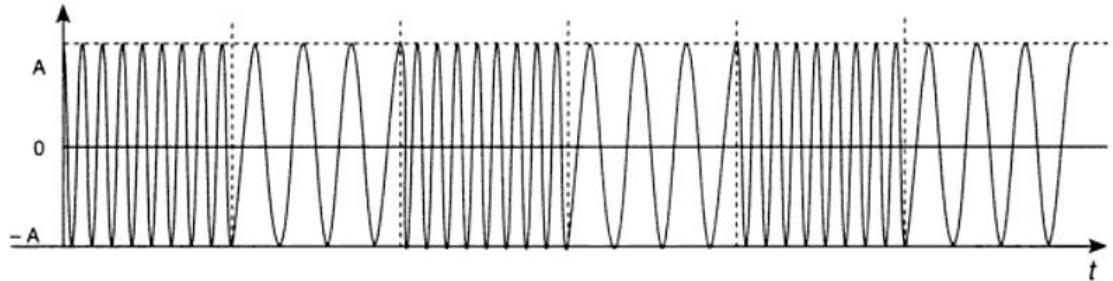


FIGURE 6.7 – Illustration de la FSK cohérente

De manière générale, les données sont envoyées suivant un débit binaire noté  $R$  (en bits/sec). Ces bits sont ensuite groupés successivement dans des blocs de  $K$  bits<sup>6</sup>. Chaque bloc représente en fait un symbole significatif envoyé toutes les  $T_s$  secondes. Les blocs ou symboles se forment donc à une fréquence (ou taux) de  $f_s = \frac{R}{K}$ , aussi appelé baud rate (avec  $K \neq 0$ ). Chaque symbole est associé à une fréquence parmi  $M = 2^K$  fréquences discrètes ; comme le montre l'expression qui suit :

$$f_p + \frac{\Delta f}{2} I(n) \quad (6.1)$$

avec  $f_p$  comme étant la fréquence de la porteuse,  $\Delta f$  étant l'espacement entre deux fréquences (par exemple dans le cas de la Figure 6.6,  $\Delta f = |f_0 - f_1|$ ) et

$$I(n) = \{\pm 1, \pm 3, \dots, \pm(M-1)\}$$

Le principe consiste donc à associer un symbole à une sinusoïde de fréquence correspondante.

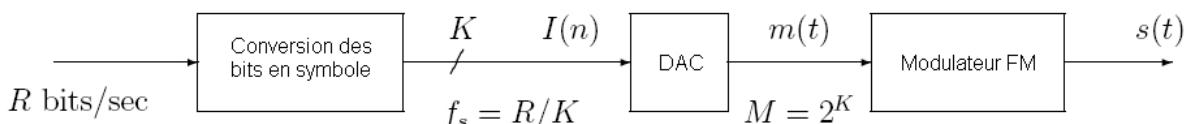


FIGURE 6.8 – Chaîne de la modulation FSK

Suivant la Figure 6.8, le DAC<sup>7</sup> génère un signal de type « escalier »

$$m(t) = \frac{\Delta f}{2} \sum_{n=0}^{\infty} I(n)p(t - nT_s) \quad (6.2)$$

où  $p(t)$  est un signal impulsionnel carré valant 1 si  $0 \leq t < T_s$  et 0 autrepart. Le message  $m(t)$ , appliqué au modulateur FM<sup>8</sup>, génère le signal de sortie

$$s(t) = A_p \cos(2\pi f_p t + \phi_m(t) + \phi_0) \quad (6.3)$$

où  $A_p$  est l'amplitude du signal et  $\phi_0$  étant la phase initiale de la porteuse comprise entre  $[\pi, -\pi]$ .

<sup>5</sup>Largement documenté grâce à [21].

<sup>6</sup>Parfois appelé Valence du signal.

<sup>7</sup>Digital to Analog Converter.

<sup>8</sup>Frequency Modulation.

Soit  $\phi_m(t)$  la phase portant l'information

$$\begin{aligned}\phi_m(t) &= 2\pi \int_0^t m(\tau) d\tau \\ &= 2\pi \int_0^t \frac{\Delta f}{2} \sum_{n=0}^{\infty} I(n) p(\tau - nT_s) d\tau \\ &= \pi \Delta f \sum_{n=0}^{\infty} I(n) \int_0^t p(\tau - nT_s) d\tau\end{aligned}$$

si on considère maintenant  $\phi_m(t)$  dans  $kT_s \leq t < (k+1)T_s$ , on effectue

$$\begin{aligned}\phi_m(t) &= \pi \Delta f \sum_{n=0}^{k-1} I(n) T_s + \pi \Delta f I(k) \int_{kT_s}^t d\tau \\ &= \pi \Delta f T_s \sum_{n=0}^{k-1} I(n) + \pi \Delta f I(k)(t - kT_s) \\ &= \pi \frac{\Delta f}{f_s} \sum_{n=0}^{k-1} I(n) + \pi \frac{\Delta f}{f_s} I(k) \frac{(t - kT_s)}{T_s}\end{aligned}$$

La phase initiale étant égale à

$$\phi_m(kT_s) = \pi \frac{\Delta f}{f_s} \sum_{n=0}^{k-1} I(n) \quad (6.4)$$

En définissant l'index de modulation<sup>9</sup> (noté  $h$ ) d'un signal FSK

$$h = \frac{\Delta f}{f_s} \quad (6.5)$$

on a enfin,

$$\phi_m(t) = \pi h \sum_{n=0}^{k-1} I(n) + \pi h I(k) \frac{(t - kT_s)}{T_s} \quad (6.6)$$

le signal de sortie vaut donc finalement pour  $kT_s \leq t < (k+1)T_s$

$$s(t) = A_p \cos [\omega_p t + \phi_m(kT_s) + \pi h I(k) \frac{(t - kT_s)}{T_s} + \phi_0] \quad (6.7)$$

Au niveau du spectre des fréquences, et suivant plusieurs indices de modulation, nous avons les allures suivantes :

---

<sup>9</sup>Lorsque  $h = 0,5$  on parle de MSK (Minimum Shift Keying).

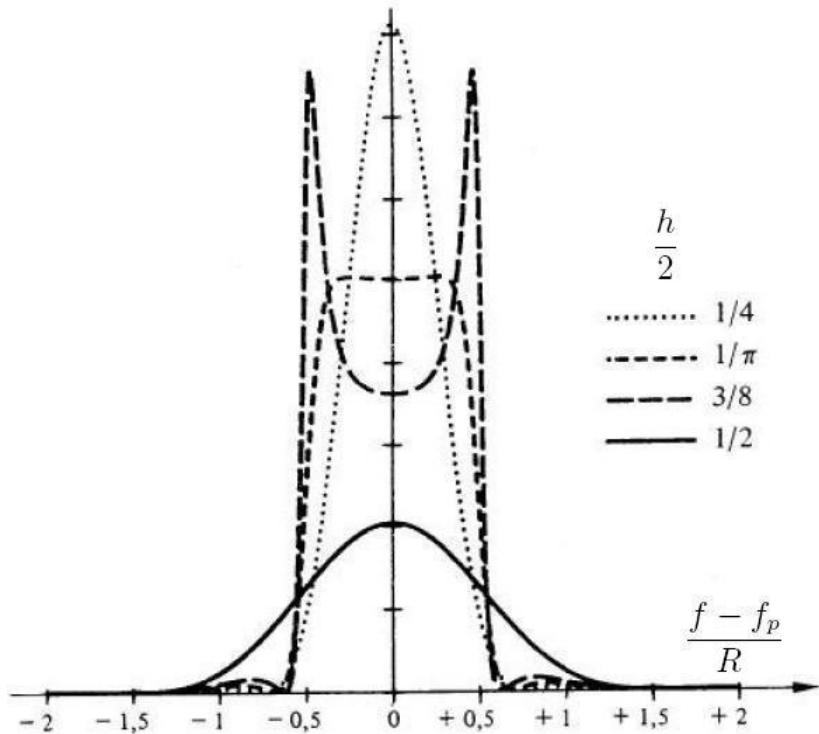


FIGURE 6.9 – Spectre de la modulation FSK cohérente (cas purement binaire,  $K = 1$ )

### Cas de l'AX.25 en 9600 baud FSK

En AX.25 9600 baud, nous n'avons que deux symboles significatifs : l'état logique 1 et l'état logique 0. Par conséquent, la logique binaire impose donc 1 seul bit pour 2 symboles significatifs ; ce qui implique que  $K = 1$ ,  $R = 9600$  bits/sec et  $M = 2$ . La fréquence instantanée  $f(t)$  du signal  $s(t)$  s'obtient en dérivant la phase instantanée de  $s(t)$  et en divisant par  $2\pi$  :

$$f(t) = f_p + \frac{1}{2\pi} \frac{d\phi_m(t)}{dt} \quad (6.8)$$

avec l'équation 6.6 et en posant  $a_i$  comme étant la valeur  $-1$  ou  $+1$  (vu que  $M = 2$  pour  $I(k)$ ), on a

$$f_i = f_p + \frac{1}{2\pi} \pi h a_i \frac{1}{T_s} \quad (6.9)$$

En simplifiant et avec l'équation 6.5, on a

$$f_i = f_p + \frac{\Delta f}{2} a_i \quad (6.10)$$

L'équation 6.10 illustre la déviation de fréquence autour de la porteuse. En effet, nous retrouvons nos deux fréquences  $f_1$  (avec  $a_i = 1$ ) et  $f_0$  (avec  $a_i = -1$ ) illustrant la modulation FSK binaire dans un cas cohérent. Concernant la largeur du spectre, nous pouvons estimer son encombrement avec la règle de Carson. En effet, la règle de Carson permet d'évaluer la largeur de bande d'un signal modulé en fréquence suivant l'indice de modulation (pour  $h = 1$  et dans un cas purement binaire) :

$$B \simeq (h + 1).R = 19,2 kHz \quad (6.11)$$

### Modem G3RUH

Le *Packet Radio* 9600 baud se base sur le modem G3RUH. Ce modem particulier au 9600 baud sera décrit plus en détails dans le chapitre 7.

## 6.4 Types de TNC

Il existe en fait deux grands types de TNC : le TNC software et le TNC hardware. En voici les particularités.

### 6.4.1 Le TNC software

Ce type de TNC est par définition un logiciel. La mise en oeuvre de ce TNC est vraiment très simple. Il ne requiert qu'un câble audio<sup>10</sup> et une carte son. Le schéma de principe se résume à la figure qui suit.

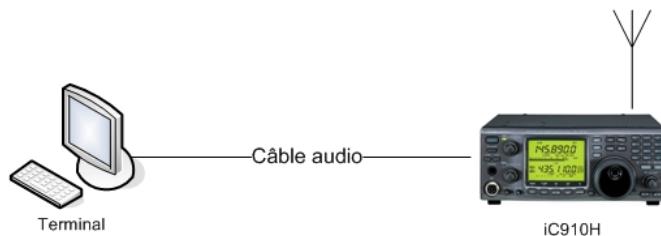


FIGURE 6.10 – Câblage du TNC software

Parmi la liste des TNC software, citons MixW<sup>11</sup>. Ce logiciel libre est un très bon TNC. MixW dispose non seulement d'un TNC AX.25 mais possède également d'autres fonctionnalités comme le code morse et bien d'autres protocoles radioamateurs.

Son principe est assez simple, lorsqu'un signal audio est émis du *transceiver* radio, celui-ci l'échantillonne directement depuis la carte son de l'ordinateur. Ensuite, le software décode la trame reçue. Si celle-ci est valide, MixW affiche les informations utiles. Dans le cas contraire, la trame est ignorée. MixW fonctionne aussi bien en 1200 baud qu'en 9600 baud.

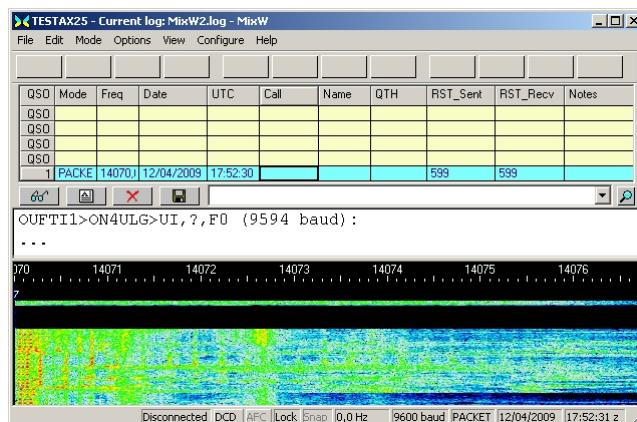


FIGURE 6.11 – Aperçu de MixW

<sup>10</sup>Voir Figure 6.3

<sup>11</sup><http://www.mixw.net/>

Le TNC software a l'avantage d'être gratuit et très facile à utiliser. Cependant, l'utilisateur doit se contraindre au logiciel et les degrés de liberté sont très faibles. Pour une utilisation plus pointue, un TNC hardware est conseillé.

### 6.4.2 Le TNC hardware

Cette unité se trouve généralement entre le terminal et le *transceiver* radio. En se référant au modèle OSI, le terminal se trouve au niveau de la couche application, le TNC se trouve en couche 2 et le *transceiver* radio se trouve au niveau de la couche physique. Le schéma de principe est le suivant :

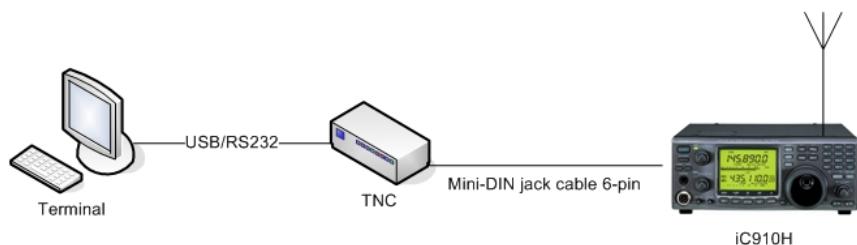


FIGURE 6.12 – Câblage du TNC hardware

Le TNC hardware est un dispositif hardware qui effectue toutes les opérations logiques de décodage de trame. Cet appareil comprend en fait un microcontrôleur et un modem. Le TNC hardware possède les mêmes fonctionnalités que le TNC software. La différence réside dans le fait que tout le traitement de l'information se fait de manière hardware. Le dialogue entre le TNC et le terminal se fait via une interface USB ou RS232 et un protocole particulier (mode KISS<sup>12</sup>). Dans le cadre du projet OUFTI-1, il y avait une volonté d'utiliser un TNC hardware. Le choix s'est porté sur le SCS Tracker/DSP TNC.

## 6.5 Le SCS Tracker/DSP TNC



FIGURE 6.13 – SCS Tracker/DSP TNC

Ce TNC nous a en fait été conseillé par Jean-Claude Cano (radioamateur français - F6CSS) et par Ted Choueiri (Etudiant EPFL) lors de notre séjour en Suisse<sup>13</sup>. De conception allemande, ce TNC utilise un DSP<sup>14</sup> et fonctionne particulièrement bien.

<sup>12</sup>Keep It Simple and Stupid.

<sup>13</sup>Voir Annexe 4 page X

<sup>14</sup>Digital Signal Processor.

Il possède tous les modes possibles :

- 300 baud AFSK (old HF-Packet-Radio standard).
- 200/600 baud Robust-Packet-Radio.
- 1200 baud AFSK.
- 9600 baud FSK (G3RUH).
- 19200 baud FSK (G3RUH).

Le TNC est entièrement configurable via USB. Il suffit simplement d'envoyer quelques caractères via USB pour basculer d'un mode à un autre.

## 6.6 Le mode KISS

Le mode KISS ou *Keep It Simple and Stupid* est en fait un format permettant d'envoyer/recevoir les trames AX.25 depuis son ordinateur avec le TNC. Le format de la trame KISS est le suivant :



FIGURE 6.14 – Format de la trame KISS

La trame est délimitée par deux flags : 0xC0. Le champ *command* est codé sur 1 byte. Les quatre premiers bits désignent la commande. Dans notre cas, nous mettrons ces quatre premiers bits à 0 (trame KISS normale comportant des données AX25). Les quatre bits restants servent à définir le numéro du port radio sur lequel les données vont être envoyées. Nous les mettrons également à 0. Le champ data AX.25 comporte les champs suivants de la trame UI :

Address	Control	PID	Info
112/224 Bits	8/16 Bits	8 Bits	N*8 Bits

FIGURE 6.15 – Format du champ data de la trame KISS

Le TNC s'occupe donc de calculer/vérifier le FCS et ajouter/enlever les flags (0x7E) de la trame AX.25. Cependant, la trame KISS subit un *byte stuffing*<sup>15</sup> pour éviter de retrouver le caractère 0xC0 dans ses données. Pour y parvenir, il suffit de boucler sur les bytes qui se trouvent entre 0xC0 et de remplacer systématiquement un éventuel 0xC0 par 0xDB 0xDC. 0xDB 0xDC sont en fait les 2 *stuffed bytes*. Subtilité, il faut également remplacer systématiquement le caractère 0xDB par 0xDB 0xDD. En effet, lorsque le TNC va recevoir 0xDB, il pourrait l'assimiler à un *stuffed byte*. C'est pourquoi nous le faisons suivre de 0xDD pour montrer qu'il s'agit bien d'une data et non d'un *stuffed byte*.

Pour information, pour quitter le mode KISS<sup>16</sup>, il suffit d'envoyer la séquence dite « magique » : 0xC0 0xFF 0xC0.

<sup>15</sup>Analogie au *bit stuffing*.

<sup>16</sup>Pour d'amples informations sur le dialogue KISS, il suffit de parcourir la toile.

# Chapitre 7

## Le modem G3RUH

Le *Packet Radio* 9600 baud se base sur le modem G3RUH. G3RUH est en fait l'indicatif radioamateur de James Miller. Dans les années 80, ce radioamateur britannique a conçu ce modem dans le but d'étendre la vitesse de transmission avec une radio classique. Le terme modem n'est pas exact. En effet, comme nous le verrons dans ce chapitre, il ne s'agit pas d'un modem mais plutôt d'un codage de l'information. Cependant, nous utiliserons tout de même le terme modem couramment employé dans le jargon.

La documentation relative à ce modem<sup>1</sup> n'a pas été facile à trouver. Bien sûr, il y a l'article de James Miller [25] mais d'autres documents ont été consultés pour mieux comprendre son fonctionnement ; à savoir : [26] [27] [28].

### 7.1 Généralités

Comme le dit James Miller dans son abstract [25] :

« *The theoretical minimum audio bandwidth required to send 9600 baud binary data is 4800 Hz. Since a typical NBFM radio has an unfiltered response from zero to some 8 kHz, transmission of 9600 baud binary data is perfectly possible through it. (...) »*

En effet, selon le théorème de Nyquist-Shannon,

$$R_{max} \leq 2BK \quad (7.1)$$

où  $R_{max}$  (en bits/sec) est le débit binaire maximum,  $B$  (en Hz) la largeur de la bande de base (ou bande passante en bande de base ou encore bande audio) et  $K$  étant la valence du signal (déjà évoquée au point 6.3.2). En cherchant l'expression du baud rate, nous avons :

$$\frac{R_{max}}{K} \leq 2B \quad (7.2)$$

Etant donnée que  $K = 1$  et que  $f_s = \frac{R}{K} = 9600\text{baud}$ , la bande passante théorique minimum vaut :

$$B_{min} \geq \frac{9600}{2} \quad (7.3)$$

$$B_{min} \geq 4800\text{Hz} \quad (7.4)$$

---

<sup>1</sup>Et a fortiori pour ce chapitre.

Il est donc techniquement possible de transmettre en 9600 baud des informations via une radio FM classique. James Miller (G3RUH) y est parvenu en concevant ce modem. Les premiers modems G3RUH étaient totalement *hardware*<sup>2</sup>. Grâce aux nouvelles techniques de programmation et à l'apparition des microcontrôleurs, il y a désormais moyen d'effectuer toutes les opérations du modem de manière *software*.

Comme nous le verrons dans ce chapitre, le modem G3RUH repose, en fait, sur le codage NRZI<sup>3</sup> et un *scrambler*<sup>4</sup>.

## 7.2 Le codage NRZI

Dans une liaison série classique, nous disposons de deux lignes de transmission : une pour les données et une pour l'horloge. Les données sont échantillonnées suivant l'horloge transmise. Cependant, dans notre cas, une seule voie de communication n'est possible. L'horloge devra donc être générée à partir des signaux de données. L'idée générale est de resynchroniser l'horloge de réception sur chaque transition des données reçues (grâce notamment à une PLL<sup>5</sup>). Pour y parvenir, le modem G3RUH se base sur le codage<sup>6</sup> NRZI (pour information, ce dernier est utilisé par la norme USB).

Premièrement, il s'agit d'un codage NRZ<sup>7</sup>. Si nous prenons une analogie électrique, le signal prend soit la valeur  $+V$  ou  $-V$ . « *Il ne retourne donc jamais à 0 V* ». Au niveau de la logique binaire, chaque bit à 0 est défini par une transition et chaque bit à 1 par une absence de transition.

Donnée	Etat à transmettre
0	Complément de l'état antérieur
1	Etat égal à l'état antérieur

Pour donner un exemple concret, la figure qui suit illustre le codage NRZI transmettant 1110001101 :

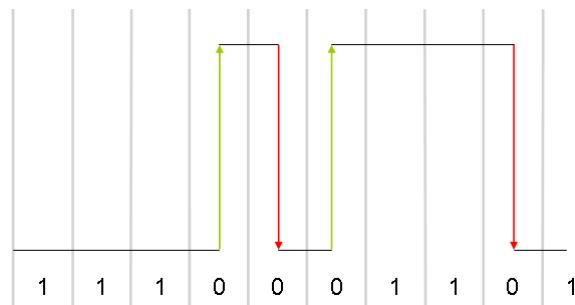


FIGURE 7.1 – Illustration du codage NRZI

<sup>2</sup>Voir schéma électrique [25].

<sup>3</sup>Non Return to Zero Inverted.

<sup>4</sup>Brouilleur en français.

<sup>5</sup>Phase Locked Loop.

<sup>6</sup>Il existe aussi d'autres codes : code biphasé, biphasé différentiel, code Manchester, ...

<sup>7</sup>Non Return to Zero.

Concernant la programmation, elle est immédiate (en pseudo-code C) :

```
if(input == 0) output = ~output; // ~ is the operator NOT
else // Nothing to do
```

Ce codage de l'information est donc très facile à mettre en oeuvre. Cependant, lors d'une longue série de 1, il n'y a plus de transition et l'horloge de réception peut malheureusement se désynchroniser. Pour rendre cette horloge parfaitement synchrone, il faut ajouter suffisamment de transitions. Pour ce faire, il faut faire appel au *scrambler*.

## 7.3 Brouillage

Parfois utilisé dans les transmissions de communications chiffrées, le *scrambler* permet notamment de brouiller les signaux. Cette propriété n'est pas celle qui nous intéresse dans notre cas. La propriété qui nous intéresse est la suivante : réduire la composante DC du signal en augmentant le nombre de transitions par unité de temps. Souvent appelé *randomizer*, ce *scrambler* empêche donc qu'une longue série de 1 ou de 0 ne se produise. L'opération inverse est effectuée par le *descrambler*.

### 7.3.1 Le scrambler

Il existe en fait deux grands types de *scrambler* :

- le *scrambler* à addition.
- le *scrambler* à multiplication.

Le modem G3RUH implémente le deuxième type. Le *scrambler* à multiplication est souvent défini par un polynôme et est récursif. James Miller a implémenté le *scrambler* de polynôme :

$$S(x) = x^{17} + x^{12} + 1 \quad (7.5)$$

### Principe du scrambler G3RUH

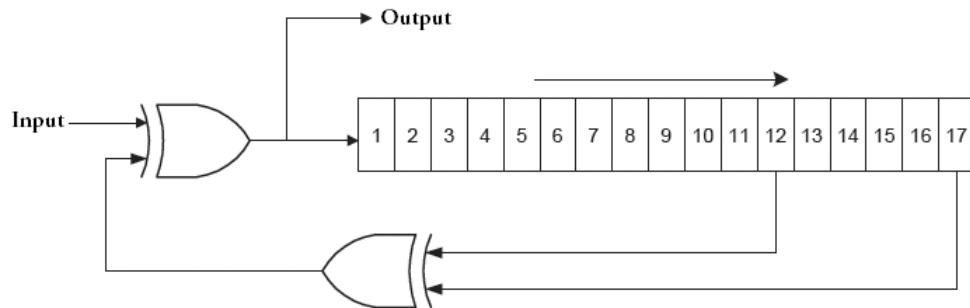


FIGURE 7.2 – Schéma d'implémentation du *scrambler*

Le *scrambler* G3RUH se limite aux opérations XOR et Shift. Contrairement au *scrambler* à addition, le registre à décalage n'a pas besoin d'être initialisé. Il peut prendre une valeur tout à fait aléatoire.

Le principe du *scrambler* est relativement simple. Une première opération XOR est réalisée entre les bits 12 et 17 du registre à décalage. La seconde opération XOR s'effectue entre le bit qui se trouve en entrée et le résultat de la première opération XOR. Enfin, le registre décale (vers la droite dans ce cas-ci) et le bit de sortie est stocké dans le registre (en position 1). La sortie *output* est l'information destinée à être modulée.

Enfin, le message transmis (noté  $X$ ) peut être décrit selon :

$$X = \text{message} \oplus \text{bit12} \oplus \text{bit17} \quad (7.6)$$

### Code

De manière générale, voici le code C du *scrambler*. Notons que, contrairement à l'exemple, le décalage se fait vers la gauche.

```
firstXorScrambler = ((shiftRegister >> 11) & 1) ^ ((shiftRegister >> 16) & 1);
output = input ^ firstXorScrambler;
shiftRegister <= 1;
if(output) shiftRegister |= 1;
```

### 7.3.2 Le descrambler

Le *descrambler* est l'opération inverse permettant de débrouiller le message transmis. En effet, en réception, nous avons donc :

$$Y = X \oplus \text{bit12} \oplus \text{bit17} \quad (7.7)$$

Où  $Y$  est le message « *désramblé* ». Suivant l'équation 7.6, nous avons :

$$Y = (\text{message} \oplus \text{bit12} \oplus \text{bit17}) \oplus \text{bit12} \oplus \text{bit17} \quad (7.8)$$

Ensuite, en simplifiant, on a :

$$Y = \text{message} \quad (7.9)$$

Par conséquent et comme nous l'annonçons déjà, le polynôme caractéristique du *descrambler* est le même que celui du *scrambler*; à savoir :

$$D(x) = x^{17} + x^{12} + 1 \quad (7.10)$$

### Principe du descrambler G3RUH

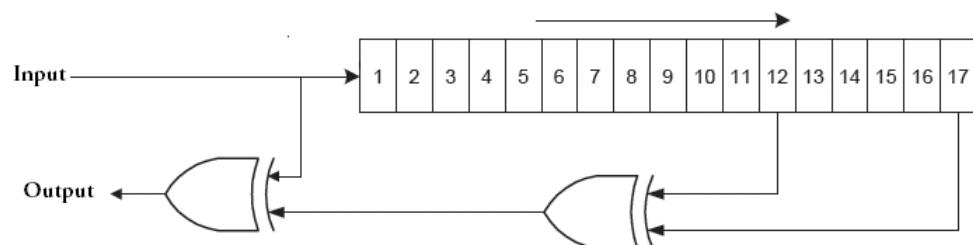


FIGURE 7.3 – Schéma d'implémentation du *descrambler*

Le *descrambler* G3RUH se limite aussi aux opérations XOR et Shift. Le registre à décalage n'a pas besoin d'être initialisé. Il peut prendre également une valeur tout à fait aléatoire. Comme le *scrambler*, une première opération XOR est réalisée entre les bits 12 et 17 du registre à décalage. La seconde opération XOR s'effectue entre le bit qui se trouve en entrée et le résultat de la première opération XOR. Enfin, le registre décale et le bit en entrée est stocké dans le registre (en position 1).

La sortie *output* correspond au message original (équation 7.9).

### Code

De manière générale, voici le code C du *descrambler*. Notons que, contrairement à l'exemple, le décalage se fait vers la gauche.

```
firstXorDescrambler = ((shiftRegister >> 11) & 1) ^ ((shiftRegister >> 16) & 1);
output = input ^ firstXorDescrambler;
shiftRegister <= 1;
if(input) shiftRegister |= 1;
```

# Troisième partie

## Aspects logiciels

# Chapitre 8

## Implémentation de la trame UI du protocole AX.25

Ce chapitre est consacré à l'implémentation logicielle de la trame *UI* du protocole AX.25. L'implémentation a été réalisée en langage C et en utilisant CROSSWORKS STUDIO comme IDE<sup>1</sup>. Le microcontrôleur utilisé est l'MSP430 de *Texas Instruments*.

### 8.1 Environnement hardware

L'environnement hardware dans lequel l'implémentation a été réalisée est, en fait, le système COM d'OUFTI-1. La COM d'OUFTI-1 peut être séparée en deux grandes parties. La première est l'émission (Tx), la seconde étant la réception (Rx). La partie Rx correspond à l'*uplink* tandis que la partie Tx correspond au *downlink*. Grâce aux travaux de [13] et [14], un prototype pour chaque partie a été réalisé sur PCB. L'écriture du code et son *debugging* ont donc été réalisés sur ces deux prototypes. Les prototypes, sous forme de cartes électroniques, se composent, dans l'absolu, d'un microcontrôleur (Ti MSP430) et de l'ADF7021.

Pour résumer, le code sera écrit dans le microcontrôleur et le *chip* ADF7021 servira de *transceiver* radio.

#### 8.1.1 Le microcontrôleur MSP430

Présentant une architecture RISC 16-bit Von-Neumann, l'MSP430 de Ti est un microcontrôleur très complet. Typiquement utilisé dans les applications sans fil, ce microcontrôleur a une consommation vraiment très faible : en pleine action celui-ci consomme 0,330 mA à 1 MHz 2,2 V. Dans le cas qui nous occupe, la tension nominale sera de 3,3 V et l'horloge sera fixée à 8 MHz. Décliné en plusieurs familles, nous nous attarderons sur la famille MSP430F16xx.

Au niveau des périphériques, l'MSP430 est assez bien garni. Nous retrouvons notamment :

- Un *watchdog timer*,
- Deux *timers* classiques,
- Deux UART (compatibles  $I^2C$  et SPI),
- Deux ADC 12 bits,
- Un comparateur hardware,

---

<sup>1</sup>Integrated Development Environment.

- Un contrôleur DMA,
- 48 I/O (dont 16 *interrupt capable*),
- (...)

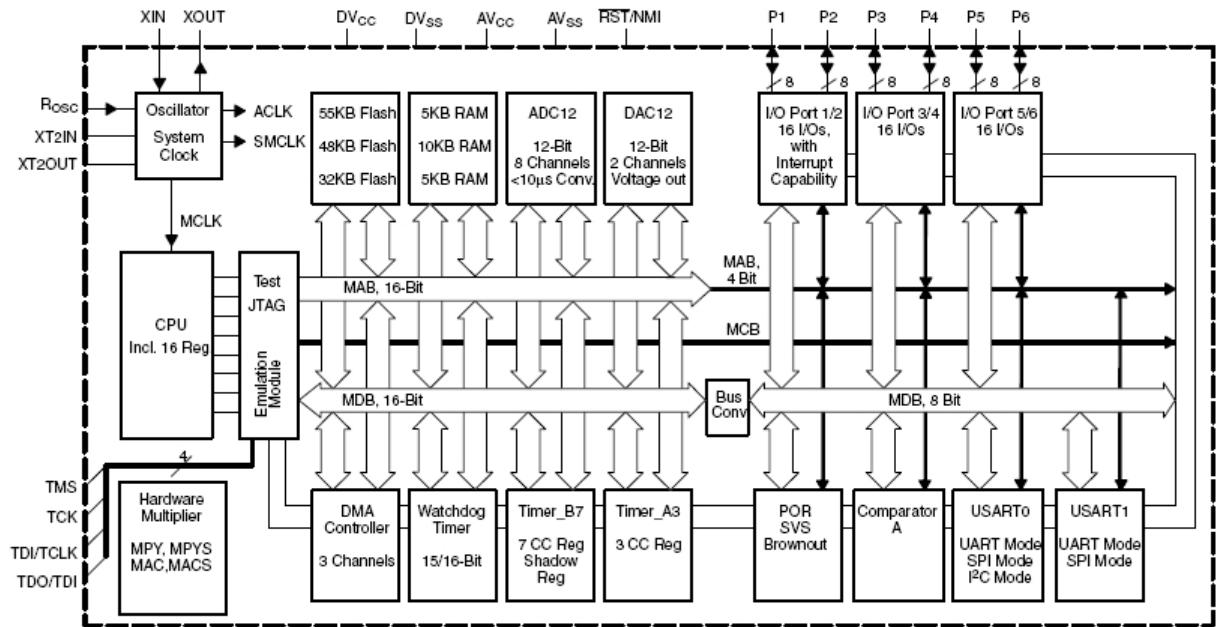


FIGURE 8.1 – Bloc-diagramme de la famille MSP430F16xx

Le microcontrôleur que nous utiliserons sera l’MSP430F1612<sup>2</sup>, présentant 5kB de RAM et 55kB de mémoire flash (ROM). Il s’agit en fait du même microcontrôleur que celui utilisé dans le modèle de vol FM430<sup>3</sup>.

### 8.1.2 L’ADF7021

Comme annoncé, le *transceiver* utilisé sera l’ADF7021. Celui-ci est également prévu pour fonctionner avec des applications sans fil car il ne consomme que très peu de courant. Concernant les modulations, il offre notamment la possibilité de moduler les signaux en 2-FSK, 3-FSK, 4-FSK et MSK.

Ce composant a d’abord été choisi pour le D-STAR car il a la possibilité de faire de la GMSK<sup>4</sup> (modulation MSK couplé à un filtre gaussien). En effet, le D-STAR se module en GMSK.

Pour communiquer avec le microcontrôleur, l’ADF7021 communique via une liaison série classique : une broche pour l’horloge (TxRxCLK) et une broche pour les données (TxRxDATA). En transmission, les bits de la trame AX.25 seront donc envoyés à fréquence de TxRxCLK sur la broche TxRxDATA pour être ensuite modulé. En réception et après démodulation, les bits seront lus sur TxRxDATA à fréquence de TxRxCLK.

<sup>2</sup>L’MSP430F1611 est également équivalent mais offre 10kB en RAM. Ce qui peut être intéressant pour les applications gourmandes.

<sup>3</sup>Voir Figure 2.9

<sup>4</sup>Gaussian Minimum Shift Keying.

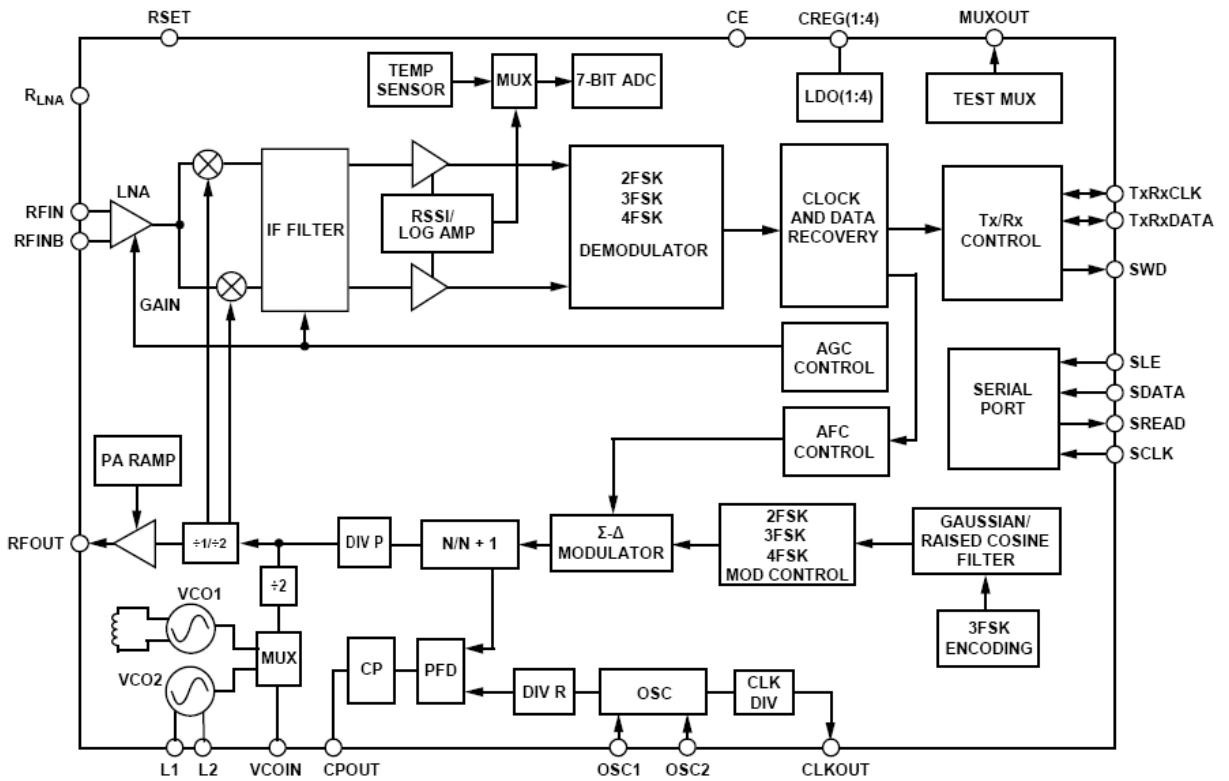


FIGURE 8.2 – Bloc-diagramme de l'ADF7021

Soulignons également que l'ADF7021 possède une broche spéciale : SWD<sup>5</sup>. Cette broche peut être très utile. L'ADF7021 possède en fait un registre interne de synchronisation où un mot de synchronisation peut être stocké (mot de 12, 16, 20 ou 24 bits). En réception et une fois que le mot est inscrit dans le registre, l'ADF7021 va rechercher automatiquement ce mot parmi les bits reçus (par comparaison). Tant que le mot n'est pas détecté, la broche SWD reste au niveau<sup>6</sup> bas. Lorsque par contre le mot est détecté, la broche SWD passe au niveau haut et la broche TxRxCLK envoie l'horloge associée aux données<sup>7</sup> sur TxRxDATA.

## 8.2 Choix et description de l'implémentation

Plusieurs choix déterminants ont été faits pour réaliser cette implémentation. Ils seront évoqués et feront office d'introduction aux points qui suivent.

### 8.2.1 Choix d'implémentation

Face au cahier des charges et aux défis à relever, plusieurs choix s'offrent à nous.

#### Baud rate et modulation

Le baud rate est le principal choix à faire. Au niveau du baud rate, nous avons deux options : le 1200 baud ou le 9600 baud. Ce choix est déterminant car il fixera aussi la modulation : A-FSK pour le 1200 baud et FSK pour le 9600 baud. Le choix du baud rate est en fait

<sup>5</sup>Synchronization Word Detection.

<sup>6</sup>Le niveau de cette broche peut être configuré.

<sup>7</sup>Les bits qui suivent le mot de synchronisation.

implicitement déjà fait. En effet, l'ADF7021 ne supporte pas la modulation A-FSK. Etant contraint à utiliser l'ADF7021, la seule modulation possible pour l'AX.25 est la modulation FSK et par conséquent le 9600 baud. Plus précisément, la modulation du signal se fera en 2-FSK<sup>89</sup>.

## Utilisation du SWD

L'utilisation du SWD concerne principalement la partie réception. En Rx, deux choix s'offrent à nous :

1. La première méthode consiste à utiliser le SWD en identifiant un mot de synchronisation dans la trame AX.25. Généralement, ce mot de synchronisation correspond aux premiers bits de la trame. Dans ces conditions, lorsqu'une trame « arrive », le mot est détecté et la réception peut débuter.
2. La deuxième méthode consiste à écouter en permanence tous les bits reçus jusqu'à trouver une trame valide (c'est-à-dire avec une longueur minimum et un FCS correct).

La première méthode est élégante. En effet, la détection se fait de manière automatique par l'intermédiaire du système de détection de l'ADF7021. Si nous prenons l'hypothèse que la lecture des bits reçus se fait sur interruption des fronts d'horloge (TxRxCLK), la méthode du SWD réduirait considérablement le nombre d'interruptions. Cette solution permettrait effectivement de désactiver les interruptions sur TxRxCLK tant que la broche SWD n'est pas au niveau haut.

Cependant, cette solution n'est pas envisageable ! Rappelons qu'en 9600 baud, les bits de la trame passent par le modem G3RUH et plus précisément par un *scrambler*<sup>10</sup>. Rappelons que le *scrambler* du modem G3RUH est un *scrambler* à multiplication. Contrairement au D-STAR qui utilise un *scrambler* à addition, le *scrambler* à multiplication implique que les bits soient envoyés de manière tout à fait aléatoire<sup>11</sup>. Par conséquent le début de la trame AX.25 sera toujours différent et donc l'utilisation du SWD est inutile.

La seconde méthode est de facto celle qui sera implémentée.

### 8.2.2 Description de l'implémentation

L'implémentation du protocole se limitera donc à la trame *UI* comme nous l'annoncions déjà dans le chapitre 4. L'implémentation de cette trame se fera dans le microcontrôleur MSP430F1612 qui correspond également au microcontrôleur de l'OBC.

Le dialogue entre le microcontrôleur et le *transceiver* (ADF7021) se déroulera suivant une liaison série classique. Etant donné que l'ADF7021 ne supporte pas la modulation A-FSK, le baud rate sera fixé à 9600 baud. Par conséquent, il faudra également implémenter le modem G3RUH de manière software.

Enfin, plus généralement, l'implémentation peut être subdivisée en deux grandes parties : la partie Tx (transmission) et la partie Rx (réception).

---

<sup>8</sup>FSK cohérente.

<sup>9</sup>Le « 2 » correspondant au niveau de modulation ; évoqué au point 6.3.2 ( $M = 2$ ).

<sup>10</sup>Voir point 7.3.1

<sup>11</sup>D'où son surnom : *randomizer*.

## 8.3 Implémentation logicielle de la partie Tx

L'implémentation de l'AX.25 est tout de même de bas niveau. En transmission, OUFTI-1 devra envoyer ses télémétries stockées quelque part dans la mémoire<sup>12</sup> de l'OBC. Ces télémétries, une fois formatées (suivant le format CCSDS par exemple) seront assemblées dans la trame *UI* et, plus précisément, se trouveront dans le champ Info de la trame. Le formatage des données est une opération de haut niveau qui ne fait pas l'objet de cette implémentation. L'implémentation consiste donc à mettre en place les outils (sous forme de fonctions en langage C) permettant d'envoyer les trames *UI* contenant les télémétries formatées.

### 8.3.1 Mise en forme du header AX.25

Si nous considérons le *header* AX.25 comme étant les champs<sup>13</sup> Address, Control et PID ; le *header* de l'AX.25 peut prendre la forme d'un tableau de caractères (le type `char` est codé sur 8 bits). Ce *header* étant fixe, nous pouvons le déclarer de manière `static const`. En effet, le champ Control prend la valeur de 0x03 et le champ PID 0xF0 (Voir point 4.5.1). Au niveau du champ Address, il faut y stocker les caractères ASCII « shiftés » des indicatifs de la source et de la destination. Ne connaissant pas encore l'indicatif du satellite OUFTI-1, nous partirons sur l'hypothèse suivante : OUFTI1-0 sera l'indicatif du satellite et ON4ULG-0 sera l'indicatif de la station au sol de l'ULg.

En langage C et grâce notamment à l'opérateur SHIFT, la programmation est immédiate :

```
static const unsigned char AX25_TxHeader[16] = {
    '0' << 1, 'N' << 1, '4' << 1, 'U' << 1, 'L' << 1, 'G' << 1, 0x60,
    '0' << 1, 'U' << 1, 'F' << 1, 'T' << 1, 'I' << 1, '1' << 1, 0x61,
    0x03, 0xF0
};
```

En transmission et conformément au protocole, le premier indicatif radioamateur est celui de la destination (ON4ULG-0) et le second est la source (OUFTI1-0). Les caractères ASCII sont « shiftés » d'un pas vers la gauche (revoir l'exemple du point 4.2.2) comme le prévoit aussi la spécification.

### 8.3.2 Chaîne d'implémentation

La chaîne d'implémentation à suivre pour l'AX.25 9600 baud doit tenir compte du *bit stuffing*, du codage NRZI et du *scrambler*. Dans un premier temps, les TM formatées (constituant le champ Info) seront assemblées au *header*, au FCS et aux flags. La première opération consiste donc à préparer la trame à envoyer et de la stocker dans un buffer du microcontrôleur. La taille du buffer dépend de la longueur du champ Info. Par défaut, le protocole la fixe à 256 bytes. Par conséquent, nous pouvons déclarer deux constantes ; à savoir :

```
#define INFO_MAX_SIZE      256
#define AX25_FRAME_MAX_SIZE 276
```

Le buffer pourra prendre la forme d'un tableau de `AX25_FRAME_MAX_SIZE`<sup>14</sup> caractères. Ensuite, dans un second temps et une fois que la trame est assemblée dans le buffer, les bits

<sup>12</sup>Ou dans une mémoire externe.

<sup>13</sup>Voir Figure 4.9

<sup>14</sup>Voir détails du calcul au point 4.3

de la trame sont envoyés à la volée suivant le bit de poids faible. Bien entendu, en tenant compte du *bit stuffing*, chaque bit subira un codage NRZI et passera à travers le *scrambler*.

Enfin, les bits seront envoyés un par un via la liaison série de l'ADF7021 pour être modulé. Pour résumer, la chaîne correspond à la figure suivante :

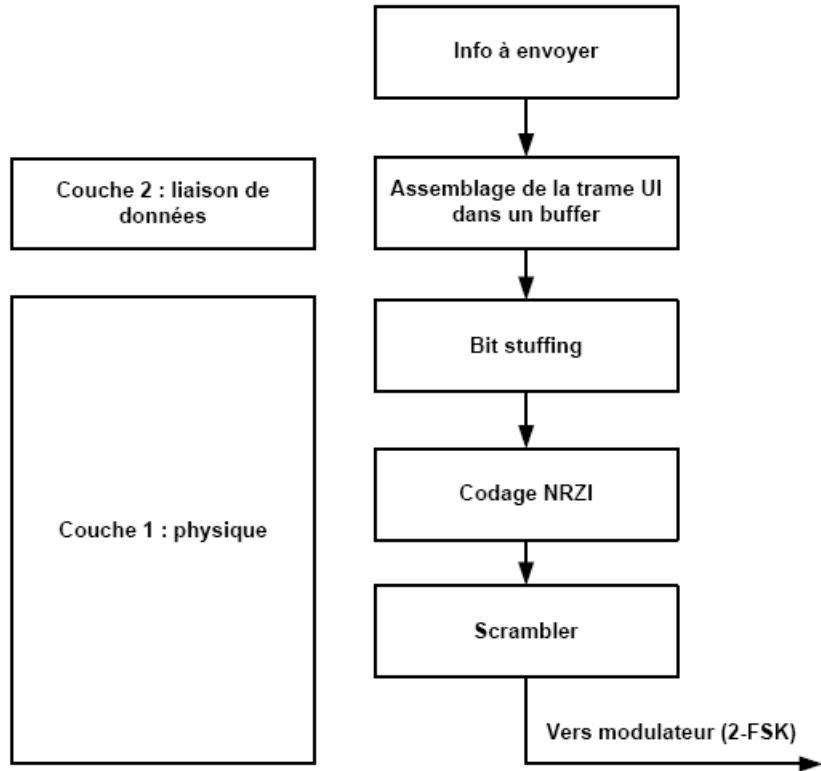


FIGURE 8.3 – Chaîne d'implémentation en transmission

### 8.3.3 Schéma haut niveau

Le déroulement de la transmission d'une trame se présente en trois grandes étapes : initialiser les variables globales (compteurs, etc), assembler/préparer la trame *UI* dans un buffer et enfin démarrer la séquence d'envoi.

La séquence d'envoi se déroule par contre en 3 grandes phases : l'envoi des premiers flags, l'envoi des données AX.25 et enfin l'envoi des flags de fin. L'envoi des premiers flags est assez important. En effet, en théorie les trames sont délimitées par uniquement 1 flag. Cependant, en pratique, il faut absolument en envoyer plus d'un. La raison vient du système de réception au sol. L'horloge de réception du TNC a besoin de plusieurs transitions pour se synchroniser. Par conséquent, quelques flags sont envoyés devant la trame pour la synchroniser. Le nombre de flag précédant la trame est caractérisé par le paramètre : *TX\_DELAY*. En fin de trame, quelques flags doivent être également envoyés ; il s'agit du *TX\_TAIL*. Ces deux paramètres sont généralement calculés par rapport au débit binaire et prennent comme unité la milliseconde. Par exemple, *TX\_DELAY* = 250 ms :

$$TX\_DELAY = \frac{0,25.9600}{8} = 300 \text{ flags.} \quad (8.1)$$

Concrètement, au niveau de la transmission, il faut envoyer la trame suivante :

Séquence d'envoi des flags (TX_DELAY_FLAG)	Séquence d'envoi des données (TX_DATA_MODE)					Séquence d'envoi des flags de fin (TX_TAIL_MODE)
TX_DELAY Flags 01111110	Address	Control	PID	Info	FCS	TX_TAIL Flags 01111110

FIGURE 8.4 – Trame AX.25 réelle

Enfin, pour résumer, voici le schéma haut niveau de l'implémentation :

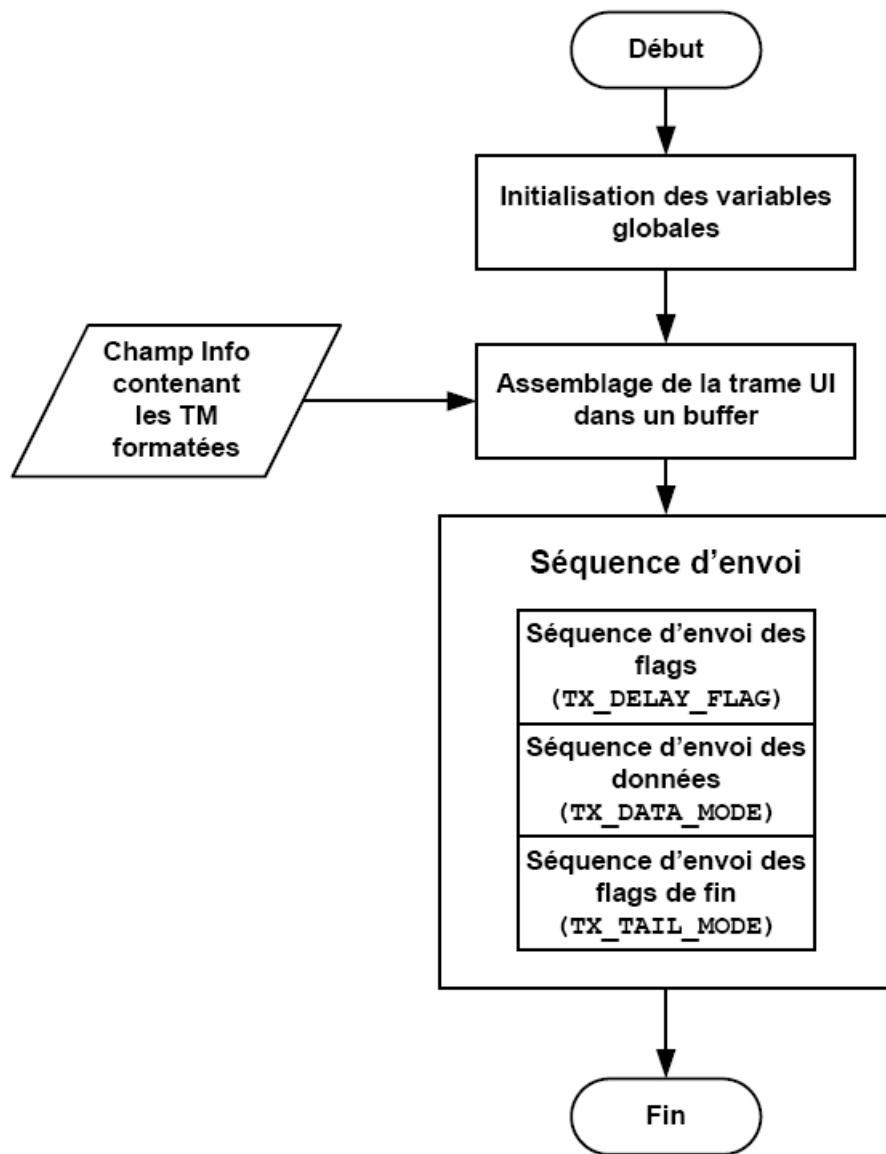


FIGURE 8.5 – Schéma haut niveau en Tx

Remarque importante : le codage NRZI et le *scrambler* sont appliqués sur toute la trame tandis que le *bit stuffing* n'est appliqué que sur les données.

### 8.3.4 Les variables globales

Plusieurs variables globales sont déclarées pour suivre la transmission des bits. Elles permettent également de tenir compte du *bit stuffing*, du codage NRZI et du *scrambler*. Elles sont en fait au nombre de 9.

Nous avons tout d'abord `unsigned char bitToSend`. Le bit de poids faible de `bitToSend` correspond au bit à écrire sur la broche TxRxDATA de l'ADF7021 et donc au bit qui va être modulé. Ce bit est donc la résultante de toutes les opérations (NRZI, *scrambler* et *bit stuffing*).

Nous avons ensuite `unsigned char lastBit` qui permet de garder en mémoire le dernier bit envoyé. Cette variable sera utilisée pour le codage NRZI, ce qui permettra de connaître l'état antérieur. `unsigned char txMode` servira à identifier le mode de transmission en cours de réalisation via des modes prédéfinis :

```
// Tx Modes
#define TX_OFF      0x00 // Off mode (Tx off).
#define TX_DELAY_FLAG 0x01 // Synchronization mode (sending flags).
#define TX_DATA_MODE 0x02 // Data transmission mode.
#define TX_TAIL_MODE 0x03 // Tail mode (send flags after the frame).
```

Concernant la longueur de la trame, nous la stockerons dans `unsigned int lengthFrame`. `unsigned long shiftRegister` sera le registre à décalage du *scrambler*. Enfin, nous avons différents compteurs :

- `unsigned char bitSetCounter` permettra de compter les bits à 1 pour le *bit stuffing*. Une fois que `bitSetCounter` sera égale à 5, il faudra insérer un zéro.
- `unsigned int nbFlagToSend` contiendra principalement le nombre de flags à envoyer avant (TX\_DELAY) et après (TX\_TAIL) la trame.
- `unsigned int byteCounter` comptera les bytes dans la trame ; ce qui permettra de pointer le bon byte du buffer à envoyer.
- `unsigned char bitCounter` comptera les bits envoyés dans le byte considéré ; une fois que `bitCounter` est égale à 8, un byte est envoyé et il faut passer au byte suivant.

Enfin, `void AX25_txInitCfg(void)` est le prototype de la fonction qui initialisera ces variables en début de transmission.

### 8.3.5 Assemblage de la trame

L'assemblage de la trame dans le buffer consiste à concaténer tous les champs de la trame *UI* : les flags, `*AX25_TxHeader`, le champ Info et le FCS. Cette opération est effectuée par :

```
void AX25_prepareUIFrame(char *buffer, char *info, unsigned int lengthInfoField)
```

En paramètre de cette fonction<sup>15</sup>, nous retrouvons le pointer du buffer, le pointer du champ Info ainsi que sa longueur. La concaténation est effectuée grâce à plusieurs boucles `for` classiques. C'est à l'appel de cette fonction que sera fixée notamment la longueur total de la trame.

Enfin, le FCS de la trame est calculé par l'intermédiaire de :

```
AX25_putCRC(buffer, lengthFrame)16
```

<sup>15</sup>Fonction réalisée grâce à Roman Merz (Université de Neuchâtel) ; voir code source [29].

<sup>16</sup>Voir chapitre 5

### 8.3.6 Séquence d'envoi

La séquence d'envoi se déroule en 3 grandes phases : l'envoi des premiers flags, l'envoi des données AX.25 et enfin l'envoi des flags de fin. La gestion de la transmission et de l'envoi des bits est réalisée grâce à une seule fonction :

```
char AX25_prepareNextBitToSend(char *buffer)
```

Etant donné que la longueur de la trame est connue (via la variable globale `lengthFrame`), le seul paramètre est le pointeur du buffer. La fonction retourne 1 tant que la transmission est en cours, et 0 lorsque celle-ci est terminée. La transmission se réduit en fait à deux cas :

1. Envoyer des flags.
2. Envoyer des données AX.25.

Un premier test est donc réalisé sur `txMode`. Dans le cas où `txMode` est différent de `TX_DATA_MODE`, nous devons envoyer des flags. Dans ce cas précis, il n'y a pas de *bit stuffing*. Il suffit uniquement d'appliquer le codage NRZI et de faire passer les bits dans le *scrambler*. Le but de l'opération consiste donc à aller chercher les bits des flags un par un dans le buffer, et de les faire passer dans la fonction

```
void AX25_txBit(char bit)
```

qui contient les opérations de *scrambling* et du codage NRZI. Le paramètre `char bit` correspond au bit à encoder. Le résultat de cet encodage est enfin stocké dans la variable `bitToSend`, prête à être envoyée. Lorsqu'un flag entier (c'est-à-dire 8 bits ; 01111110) est envoyé, la variable globale `nbFlagToSend` est décrémentée. Lorsque `nbFlagToSend` est nul et si nous sommes dans le cas `TX_DELAY_FLAG`, la transmission peut passer en mode `TX_DATA_MODE` destiné à envoyer les données AX.25.

Dans ce cas-ci, toutes les opérations doivent être réalisées : *bit stuffing*, codage NRZI et *scrambler*. Le codage NRZI et le *scrambler* sont réalisés par la fonction que nous connaissons déjà ; à savoir : `void AX25_txBit(char bit)`.

En ce qui concerne le *bit stuffing*, une fonction a été prévue. Il s'agit de

```
char AX25_checkBitStuffing(void)
```

Cette fonction n'a besoin d'aucun paramètre. En effet, cette fonction ne vérifie que l'état de `bitSetCounter`. Si ce dernier est supérieur ou égale à 5, il faut insérer un 0. Dans le cas où il faut insérer un 0, aucun compteur (`byteCounter` et `bitCounter`) n'est incrémenté. En effet, le *stuffed bit* est un bit supplémentaire qui ne fait pas partie de la trame AX.25. En revanche, ce bit doit subir le codage NRZI et doit passer dans le *scrambler* (le résultat étant stocké dans `bitToSend`). Pour information, la fonction `AX25_checkBitStuffing()` retourne 1 lorsqu'il y a présence d'un *stuffed bit* et 0 sinon.

Enfin, lorsque toutes les données sont envoyées, c'est-à-dire lorsque la condition suivante est remplie :

```
if(byteCounter == lengthFrame-1)
```

il ne reste plus qu'à envoyer les flags de fin. Nous passons donc dans le dernier mode de transmission : `TX_TAIL_MODE`. Ce mode se terminera lorsque tous les flags (constituant la « queue » de la trame) seront envoyés. C'est-à-dire quand `nbFlagToSend` sera nul. `txMode` passera en mode `TX_OFF` et la fonction `AX25_prepareNextBitToSend` retournera 0, synonyme de fin de transmission.

### 8.3.7 Communication avec l'ADF7021

Le bit à envoyer se trouve dans la variable globale : `bitToSend`. Cette variable est tout à fait accessible via le type `extern`. Ce bit doit être écrit sur la broche `TxRxDATA` de l'ADF7021 conformément au chronogramme suivant :

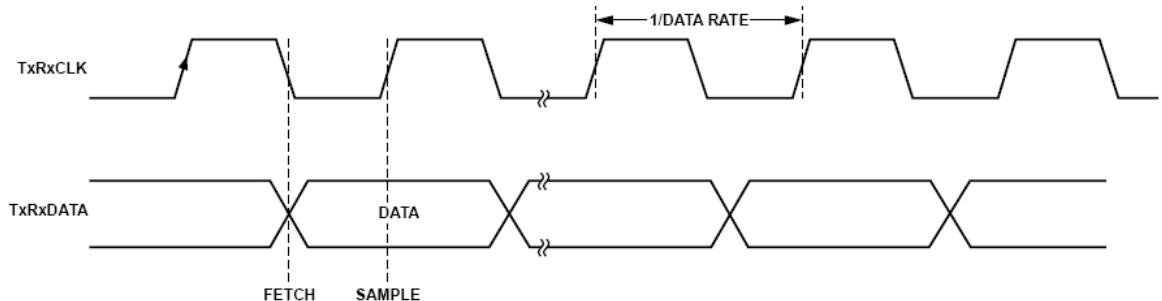


FIGURE 8.6 – Chronogramme ADF7021 en transmission

A la vue de ce chronogramme, le bit doit donc être écrit sur le front descendant de l'horloge. Voici un exemple de transmission en pseudo-code C :

```
AX25_txInitCfg();
AX25_prepareUIFrame(buffer, info, sizeof(info));
do {
    transmission = AX25_prepareNextBitToSend(buffer);
    while(TestBit(PORT, TXRXCLK) == 1); // Waiting for the falling TxRxClk edge.
    if(bitToSend) SetBit(PORT, TXRXDATA);
    else ClrBit(PORT, TXRXDATA);
    while(TestBit(PORT, TXRXCLK) == 0); // Waiting for the rising TxRxClk edge.
}
while(transmission);
```

Pour terminer cette partie Tx, il est bien entendu évident que la transmission peut se faire sur interruption de l'horloge (front descendant). Lorsque l'interruption a lieu, il suffit d'appeler `AX25_prepareNextBitToSend(buffer)` pour ensuite écrire `bitToSend` sur `TxRxDATA`.

## 8.4 Implémentation logicielle de la partie Rx

En réception, l'implémentation est du même acabit. Pour rappel, la réception consiste à récupérer les TC envoyées depuis le sol. Une trame *UI* sera donc envoyée depuis la station de contrôle vers le satellite (*uplink*). L'implémentation consiste donc à mettre en place les outils (sous forme de fonctions en langage C) permettant de recevoir les trames *UI* contenant les télécommandes du satellite.

### 8.4.1 Chaîne d'implémentation

La chaîne d'implémentation est en fait l'inverse de celle de la partie Tx ; c'est-à-dire : faire passer les bits reçus à travers le *descrambler*, faire le décodage NRZI, supprimer le *bit stuffing* et vérifier le FCS.

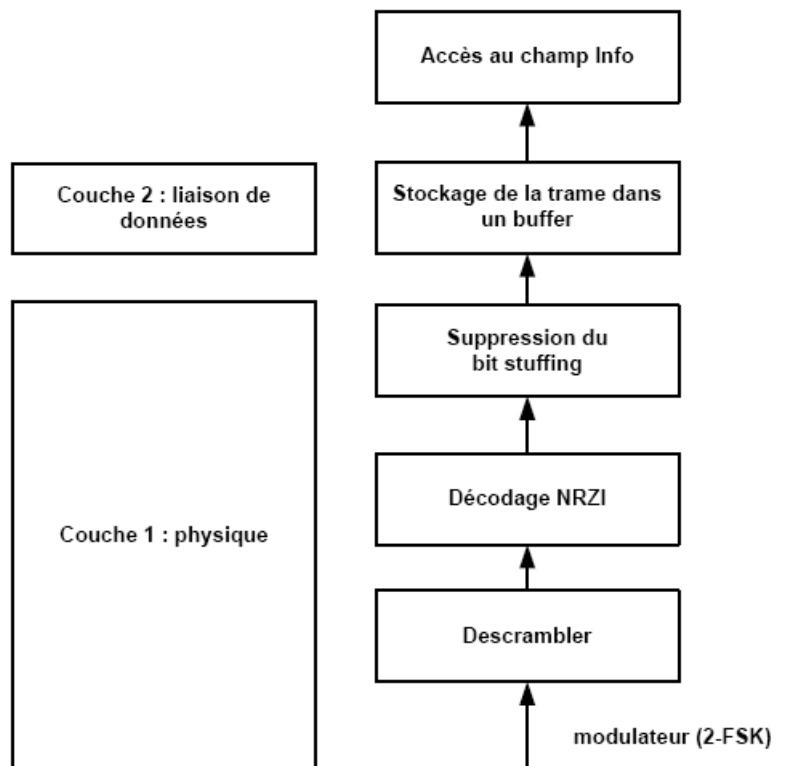


FIGURE 8.7 – Chaîne d'implémentation en réception

### 8.4.2 Schéma haut niveau

Sur base des choix du point 8.2.1, nous partons sur une solution qui consiste à écouter en permanence tous les bits reçus (même le bruit) jusqu'à trouver une trame valide.

Le déroulement de la réception se présente également en 3 étapes : initialiser les variables globales (compteurs, etc), démarrer la séquence de réception (en écoutant les bits reçus) et enfin vérifier le FCS de la trame.

La séquence de réception s'organise en 3 grandes phases. Etant donné que le TNC envoie également quelques flags (TX\_DELAY) en début de trame, la première phase consiste à en détecter un. Ensuite, une fois un flag détecté, il convient d'ignorer les flags supplémentaires qui suivent. Au moment où un byte différent de celui du flag est reçu, la séquence de réception des données AX.25 débute. Cette dernière s'estompera lorsqu'un flag de fin sera détecté.

Enfin, pour prendre la décision quant à la validité des octets reçus, le FCS sera vérifié. Si celui-ci est correct, la trame sera considérée comme valide.

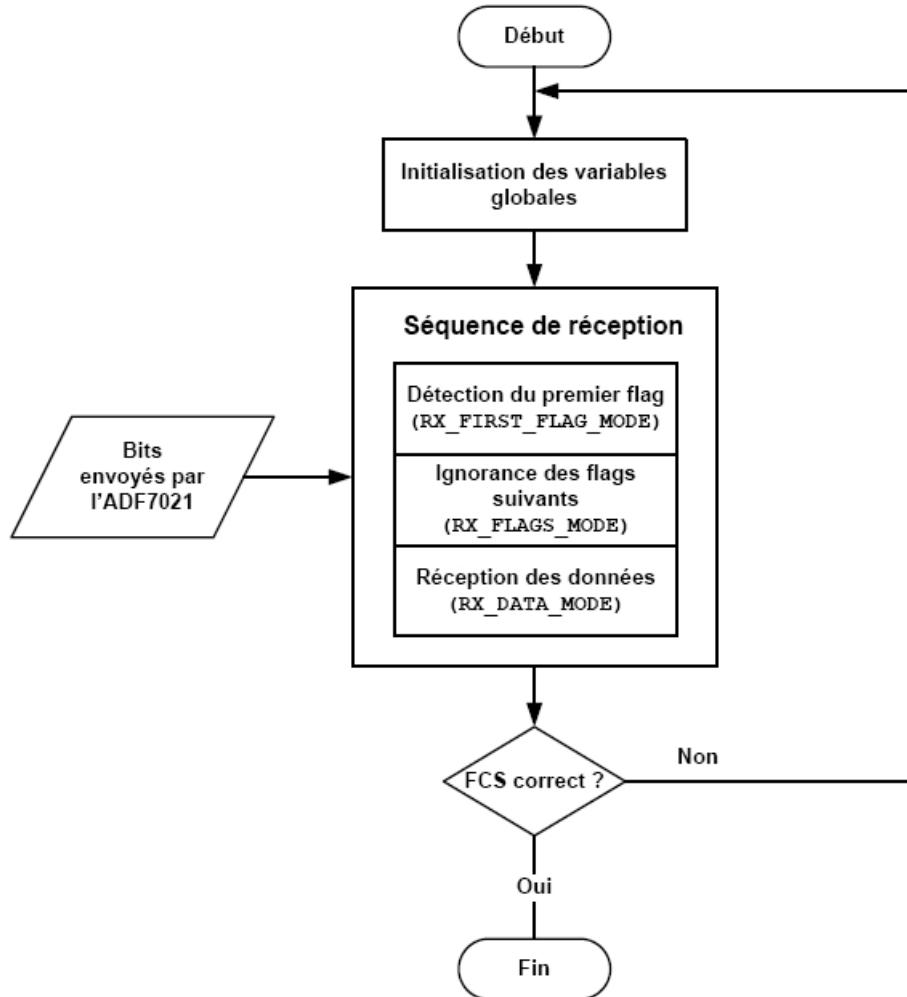


FIGURE 8.8 – Schéma haut niveau en Rx

### 8.4.3 Les variables globales

Plusieurs variables globales sont déclarées pour suivre la réception des bits. Elles permettent également de tenir compte du *bit stuffing*, du codage NRZI et du *descrambler*. Elles sont en fait au nombre de 7.

La partie Rx a été réalisée en analogie avec la partie Tx. Nous retrouvons les mêmes variables globales ; à savoir :

```

unsigned char rxMode;
unsigned char bitSetCounter;
unsigned char lastBit;
unsigned long shiftRegister;
unsigned int byteCounter;
unsigned char bitCounter;
unsigned char checkBitStuff;

```

Tout d'abord, `unsigned char rxMode` servira à identifier le mode de réception en cours de réalisation via des modes prédéfinis :

```
// Rx Modes
#define RX_OFF           0x00 // Receiver is off.
#define RX_FIRST_FLAG_MODE 0x01 // First flag detection mode.
#define RX_FLAGS_MODE     0x02 // Ignoring sync flags.
#define RX_DATA_MODE      0x03 // Data decoder mode.
```

Ensuite, dans l'absolu, `bitSetCounter` compte le nombre de 1 reçu en vue de supprimer d'éventuel *stuffed bit*. Lorsque `bitSetCounter` sera égale à 5, `checkBitStuff` prendra la valeur 1 pour signaler de vérifier le prochain bit. En effet, si le prochain bit est un 0, nous avons affaire à un *stuffed bit*; en revanche, si le prochain bit est 1, nous avons affaire à un flag.

La variable `lastBit` servira à mémoriser l'état antérieur du dernier bit reçu en vue du décodage NRZI. Le registre à décalage du descrambler se caractérisera par la variable `shiftRegister`.

Enfin, `byteCounter` comptera le nombre de bytes reçus dans le buffer et `bitCounter` comptera le nombre de bits reçus dans le byte considéré. Une fois que `bitCounter` est égale à 8, un byte entier est reçu et `byteCounter` peut être incrémenté.

Toutes ces variables sont initialisées grâce à la fonction de prototype :

```
void AX25_rxInitCfg(void)
```

#### 8.4.4 Séquence de réception

Annoncée plus haut, la séquence de réception s'organise en 3 grandes phases. La gestion en réception est principalement centralisée dans la fonction suivante :

```
char AX25_analyzeNextBit(char *buffer, char rxBit)
```

En paramètre, nous retrouvons le pointer du buffer qui va accueillir la trame et `rxBit` correspondant au bit lu sur la broche TxRxDATA de l'ADF7021 (c'est-à-dire le résultat de la démodulation 2-FSK). Cette fonction teste en fait la variable `rxMode` et exécute le code correspondant au mode de réception en cours.

Lorsque le mode de réception est `RX_FIRST_FLAG_MODE`, OUFTI-1 est en attente d'un flag. C'est-à-dire que le satellite écoute en permanence sa fréquence *uplink*. Même si aucun signal n'est envoyé depuis le sol, les bits provenant du bruit ambiant passeront tout de même à travers le *descrambler* et seront décodés (NRZI) comme si il s'agissait du bon signal<sup>17</sup>. Si nous prenons l'hypothèse d'un environnement parfait (aucun bruit), OUFTI-1 est en pleine attente d'un flag. Une fois que le sol envoie son signal et après démodulation, le premier bit de la trame se présente sur la broche TxRxDATA. Suivant l'horloge TxRxCLK, le bit est lu et est stocké dans `rxBit`. L'information stockée dans `rxBit` sera « descramblée » et décodée (NRZI) via la fonction suivante :

```
char AX25_rxBit(char bit)
```

`AX25_rxBit(char bit)` effectue donc les opérations inverses du *scrambler* et du codage NRZI. Lorsque les deux opérations sont réalisées, la fonction retourne le bit résultant des opérations de *descrambling* et du décodage NRZI. Ensuite, en traitant tous les bits reçus sur TxRxDATA via `AX25_rxBit(char bit)`, un flag sera trouvé. Le flag sera stocké dans

---

<sup>17</sup>Cette technique implique un problème majeur : un « faux » flag peut être détecté à travers le bruit ambiant. En revanche, ce problème peut être résolu plus tard lorsque le FCS sera vérifié.

le buffer à la position de `byteCounter`<sup>18</sup>. Une fois le flag stocké, `rxMode` pourra alors passer en `RX_FLAGS_MODE` destiné à ignorer les flags qui suivent. `rxMode` quittera ce mode pour `RX_DATA_MODE` dès que la condition<sup>19</sup> suivante sera satisfaite :

```
if(buffer[byteCounter] != 0x7E)
```

En `RX_DATA_MODE`, les trois opérations (*bit stuffing*, *descrambler* et décodage NRZI) doivent être réalisées. Le *descrambler* et le décodage NRZI seront effectués via la fonction dont nous avons déjà parlé : `AX25_rxBit(char bit)`.

Au fur et à mesure que `char AX25_analyzeNextBit(char *buffer, char rxBit)` reçoit ses bits, `bitSetCounter` compte le nombre de 1 reçu. Lorsque `bitSetCounter` sera supérieur ou égale à 5, `checkBitStuff` prendra la valeur 1 pour signaler de vérifier le prochain bit. Si `checkBitStuff` vaut 1, l'algorithme fait appel à la fonction qui vérifie le prochain bit reçu (`bit`) :

```
char AX25_checkBitStuffing(char bit) {
    if(AX25_rxBit(bit) == 0x00) { // Check if the rxBit is a 0.
        checkBitStuff = 0; // Clear the bit stuffing indicator.
        bitSetCounter = 0; // Clear the bit set counter.
        return 1; // If so, it is a stuffed bit.
    }
    else return 0; // If not, we have a flag.
}
```

Si le bit reçu est un 0, nous avons affaire à un *stuffed bit* qui sera supprimé. Dans le cas contraire, il s'agit d'un flag de fin, synonyme de fin de réception. Dans le cas d'un flag de fin, le FCS sera recalculé et comparé au FCS reçu. Finalement, si ces derniers sont identiques, la trame est considérée comme valide.

#### 8.4.5 Communication avec l'ADF7021

La lecture de la broche TxRxDATA de l'ADF7021 doit être réalisée selon le chronogramme suivant :

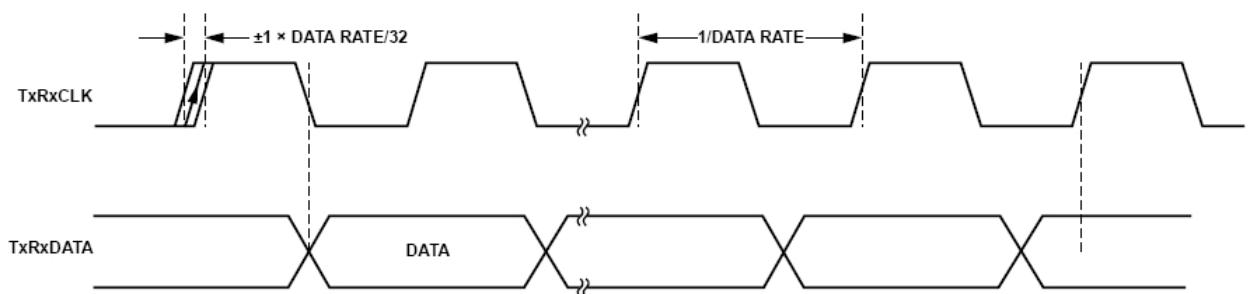


FIGURE 8.9 – Chronogramme ADF7021 en réception

A la vue de ce chronogramme, le bit doit donc être lu sur le front montant de l'horloge.

<sup>18</sup>`void AX25_rxInitCfg(void)` initialise cette variable à 0. Par conséquent, le flag (0x7E) se trouvera en première position dans le buffer ; conformément à la Figure 4.9

<sup>19</sup>Signalons que `byteCounter` a été incrémenté une fois entre temps.

Voici un exemple de code en réception (pseudo-code C) :

```
do {
    AX25_rxInitCfg(); // Initialization of the global variables.
    do {
        while(TestBit(PORT, TXRXCLK) == 0); // Waiting for the rising RxClk edge.
        rxBit = TestBit(PORT, TXRXDATA); // Fetch the bit from the ADF.
        reception = AX25_analyzeNextBit(buffer, rxBit); // Decode the bit.
        while(TestBit(PORT, RXCLK) == 1); // Waiting for the falling RxClk edge.
    }
    while(reception); // If AX25_analyzeNextBit returns 0 -> we have a frame

    byteCounter++; // Length of the frame is byteCounter + 1.
    crc = AX25_computeCRC(buffer, byteCounter); // Calculation of the CRC.

    if(buffer[byteCounter - 3] == (crc & 0xff)) {
        if(buffer[byteCounter - 2] == ((crc >> 8) & 0xff)) {
            break; // If CRC are equals, break ! The frame is good.
        }
    }
    else continue; // If not, continue.
}
while(1);
```

# Chapitre 9

## Interface graphique

Pour la phase de tests, une application a été développée pour communiquer avec le TNC. Cette application est une interface graphique développée en C#. Elle permet notamment de configurer le TNC via USB mais aussi d'envoyer ou de recevoir des trames AX.25. Cette application a permis de faciliter le déroulement de la phase de test.

### 9.1 Principe de fonctionnement

Le principe de fonctionnement de cette application est très simple. Dans un premier temps, cette application permet de se connecter via USB au TNC (via un bouton classique). Une fois connecté au TNC, un second bouton permet de configurer le TNC en 9600 baud G3RUH. Ce dernier bouton est accompagné d'un bouton *reset* au cas où le TNC aurait besoin d'une remise à zéro. Derrière tous ces boutons se cache en fait une *class* et des méthodes particulières au TNC. Ces méthodes sont exécutées lors d'événements CLICK.

Ensuite, dans un second temps, lorsque le TNC est bien connecté et bien configuré, deux modes de test sont possibles : le mode transmission et réception. Le mode transmission envoie une trame test AX.25 via un bouton à destination de la carte Rx (prototype). Le contenu de la trame est envoyé suivant le format KISS<sup>1</sup> sur l'USB. En mode réception, c'est l'inverse : l'application se met en « attente » d'une trame KISS en provenance du TNC et plus globalement de la carte Tx (prototype).

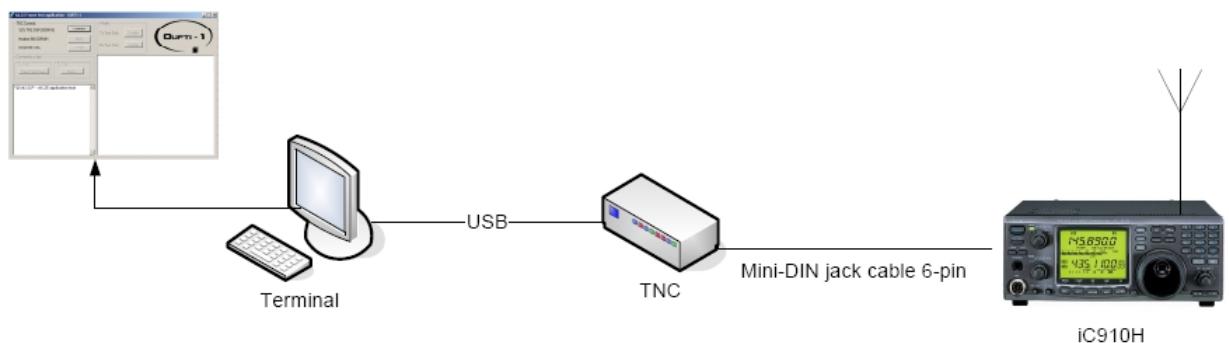


FIGURE 9.1 – Schéma de principe de l'interface graphique du TNC

<sup>1</sup>Voir Figure 6.14

## 9.2 Configuration du TNC

Pour configurer le TNC, il faut se référer à son manuel d'utilisation [30]. En parcourant le manuel, il est expliqué comment envoyer des commandes. Il suffit en fait de lui envoyer quelques caractères :

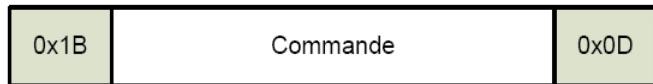


FIGURE 9.2 – Format des commandes de configuration du TNC

Les valeurs hexadécimales 0x1B et 0x0D représentent respectivement les caractères ESCAPE et ENTER. Concernant les commandes<sup>2</sup>, elles sont très faciles à manipuler. Par exemple, pour passer en mode 9600 baud G3RUH, il suffit d'envoyer : %B9600. Pour passer en 1200 baud : %B1200. Pour passer en full duplex : il suffit d'envoyer @D1.

Pour faciliter la configuration, une méthode C# a été écrite. Cette méthode écrit sur le port USB les caractères de la commande :

```
private void SendToTNC(string data)
{
    byte[] toSend = new byte[data.Length + 2];
    toSend[0] = 0x1b;
    char[] c = data.ToCharArray();
    for (int i = 0; i < c.Length; i++)
    {
        toSend[i + 1] = (byte)c[i];
    }
    toSend[toSend.Length - 1] = 0xd;
    USBPort.Write(toSend, 0, toSend.Length);
    Thread.Sleep(data.Length * 8);
}
```

Concernant le port USB, il suffit de se connecter sur le port COM qui lui correspond. Par défaut, le TNC est sur le port COM18. Pour ce qui est de sa configuration, le port doit être en 38400 baud, bits de données à 8, aucune parité, un stop bit, pas de gestion de flux et les *timeouts* mis à 500 ms.

Ensuite, lorsque le TNC est bien connecté, la configuration peut s'effectuer via la méthode suivante :

```
public void ConfigureInPacketRadio()
{
    // Configuration
    SendToTNC("@D0");      // Half duplex
    SendToTNC("@F0");      // Don't send flags during pause
    SendToTNC("%B9600");   // 9600 baud FSK
    SendToTNC("X1");       // PTT normal operations
    SendToTNC("%X400");    // Set TNC Output level to 400mV p-p for IC910H

    // Misc
    SendToTNC("%T25");    // Tx Delay (x10ms)
    SendToTNC("P128");    // Persistence
```

<sup>2</sup>Pour information, les différentes commandes se trouvent dans [30].

```

SendToTNC("W20");    // Slot Time (ms)
SendToTNC("02");    // Max Frames
SendToTNC("R0");    // Digipeating OFF

// Desable Call check
SendToTNC("@VO");

// KISS Mode ON
SendToTNC("@K");
}

```

La dernière commande est très importante, il s'agit du mode KISS<sup>3</sup>. C'est par ce biais que nous pourrons envoyer/recevoir les trames AX.25.

### 9.3 Méthodes et format KISS

Les trames AX.25 prendront la forme du format KISS. Plusieurs méthodes ont été écrites pour encoder/décoder ce format. En transmission, une seule méthode est nécessaire. Il s'agit de

```
public bool SendKISSData(byte[] data)
```

En paramètre, se trouve `byte[] data` prenant la forme de la Figure 6.15. Cette méthode assemble donc la trame KISS, réalise le *byte stuffing*<sup>4</sup> et écrit les bytes sur le port USB.

En réception, plusieurs artifices sont nécessaires. Si nous ne considérons uniquement que les méthodes principales, l'implémentation se limite à une seule méthode. Il s'agit de

```
private void port_DataReceived(object sender, SerialDataReceivedEventArgs e)
```

Cette méthode récupère les bytes sur le port USB pour reformer la trame KISS. Elle s'appuie également sur une méthode qui supprime le *byte stuffing* ; à savoir

```
private byte[] removeEscapes(byte[] buf)
```

Enfin, pour résumer, lorsque le bouton SEND sera enfoncé par la souris, nous ferons appel à `SendKISSData(...)` et lorsque le TNC enverra une trame AX.25 au format KISS, un évènement fera appel à `port_DataReceived(...)`.

### 9.4 Class TNC

Une *class C#* a été écrite<sup>5</sup> pour regrouper les méthodes utiles au TNC. Cette *class* permet d'envoyer/recevoir des trames KISS et de configurer le TNC.

La *class* s'appelle TNC et crée donc des objets de type TNC. Le prototype du constructeur de la *class* est le suivant : `public TNC(string com)`. Seul le numéro du port COM est nécessaire. Une fois le port COM connu, le constructeur ouvre un port série (`new SerialPort()`) et le configure suivant les paramètres cités plus haut.

Le constructeur est également accompagné de plusieurs méthodes qui gèrent notamment les évènements du port série, la gestion du *byte stuffing*, la transmission de trame KISS, etc. A la page suivante se trouve un aperçu de la *class*.

<sup>3</sup>Voir point 6.6

<sup>4</sup>Voir point 6.6

<sup>5</sup>Grâce notamment à Ted Choueiri (EPFL).

```

internal delegate void PacketReceivedEventHandler(object sender,
                                                PacketReceivedEventArgs e);
internal class PacketReceivedEventArgs : EventArgs {...}

class TNC
{
    private static int BUFFER_SIZE = 512;
    internal event PacketReceivedEventHandler PacketReceived;
    private byte[] buffer = new byte[BUFFER_SIZE];
    private SerialPort USBport;
    private int pointer = 0;

    public TNC(string com) {...}

    // Method isConnected return true if connected
    public bool IsConnected() {...}

    // Handle the event SerialDataReceived
    private void port_DataReceived(object sender, SerialDataReceivedEventArgs e)
    {...}

    // Analyze the Kiss Frame and remove byte stuffing
    public bool analyzeCommand(byte[] buf, out byte[] toRet) {...}

    // Method which removes stuffed bytes
    private byte[] removeEscapes(byte[] buf) {...}

    // Method to send cfg cmds of the TNC
    private void SendToTNC(string data) {...}

    // Reset the TNC
    public void Reset() {...}

    // Disconnect the TNC
    public void Disconnect() {...}

    // Configuration of the TNC in PK9k6 (G3RUH)
    public void ConfigureInPacketRadio() {...}

    // Method to send KISS frame
    public bool SendKISSData(byte[] data) {...}
}

```

## 9.5 Aperçu de l'interface graphique

L'interface graphique implémente la *class* TNC. Pour rappel, ce petit programme sert simplement à se connecter via USB au TNC, de le configurer en *Packet Radio* 9600 baud G3RUH, faire éventuellement un *reset* et surtout faire des tests (Tx et Rx).

L'application se compose d'un *log* sur la gauche permettant de suivre les évolutions du programme, d'une *TEXTBOX* sur la droite permettant d'afficher la trame envoyée/reçue et enfin de quelques boutons.

Enfin, un système de fichiers textes en arrière plan permet de sauver les trames envoyées/-reçues pour garder une trace du dernier test réalisé.

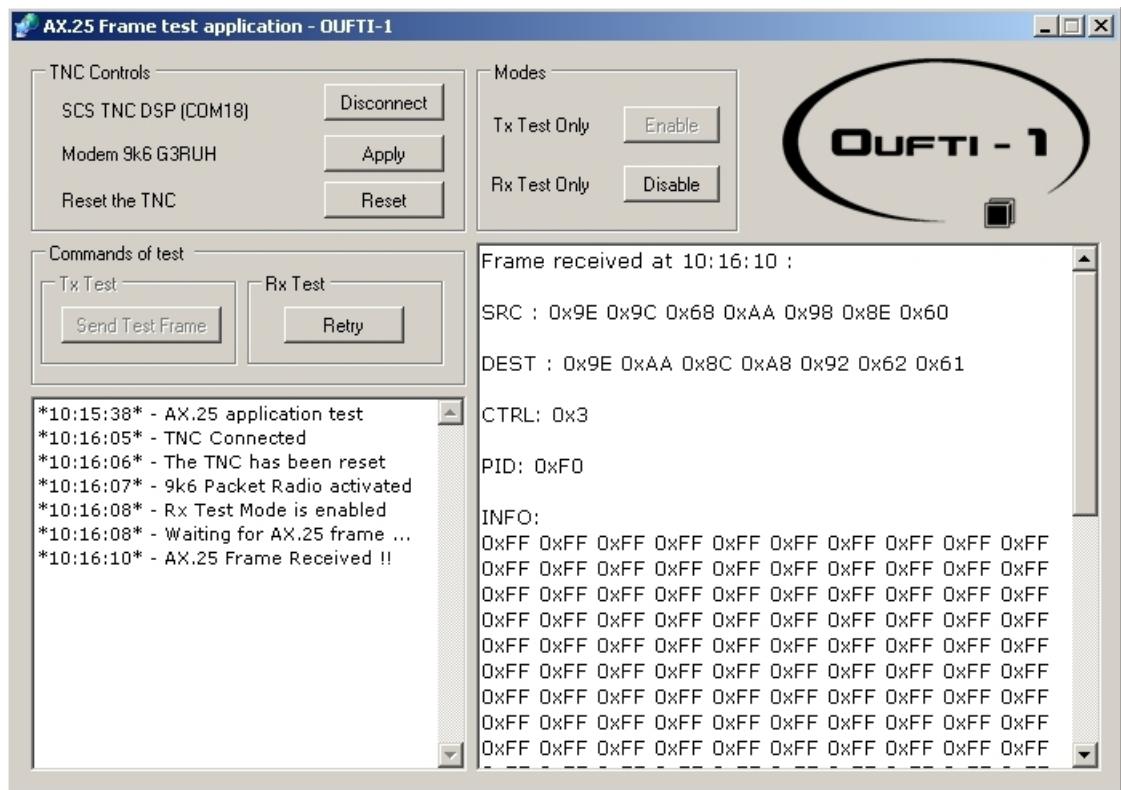


FIGURE 9.3 – Interface graphique du TNC

# Chapitre 10

## Phase de tests

Ce chapitre est consacré à la phase de tests de l'implémentation. Il s'agit donc de mettre en oeuvre le protocole AX.25 et sa trame *UI* avec les prototypes existants et du matériel radioamateur. Mise à part les simulations logicielles, trois grands essais ont été réalisés : un essai en transmission (*downlink*), un essai en transmission/réception entre les deux prototypes et enfin, un essai en réception (*uplink*). Ces essais ont été réalisés avec du matériel radioamateur et le TNC.

### 10.1 Matériels utilisés

Concernant le matériel, il fallait donc un *transceiver*, des antennes, les deux prototypes (Tx et Rx), un ordinateur, un programmeur MSP430 et des instruments de mesures (oscilloscope, analyseur de spectre, multimètre, etc). Etant donné que l'ISIL possédait le matériel nécessaire, c'est donc à l'ISIL que les essais ont été menés.

Au niveau du *transceiver*, le choix s'est porté sur l'Ic-910h. L'Ic-910h a en fait tous les modes possibles et possède les outils<sup>1</sup> permettant de compenser l'effet Doppler. C'est donc ce matériel qui fera office de « radio ».



FIGURE 10.1 – Icom 910h

Au niveau du TNC, nous utiliserons le SCS Tracker/DSP TNC évoqué dans les chapitres précédents. Le TNC sera d'une part connecté au *transceiver* via la liaison *data* et d'autre part à l'ordinateur via USB. Bien entendu, pour mener à bien ces essais, nous utiliserons l'interface graphique du chapitre 9. En fait, l'ensemble de ces dispositifs constitue une station de contrôle primitive (ou station de contrôle de bas niveau).

Concernant maintenant les prototypes, la carte Tx permettra d'émettre des trames *UI* vers le *transceiver* et la carte Rx permettra de réceptionner les trames *UI* émises du *transceiver*. Chaque carte possède un MSP430F1612 et un ADF7021. Les cartes possèdent également un connecteur pour y brancher le programmeur et donc réaliser les essais.

---

<sup>1</sup>Via le CT-17 et orbitron.

Pour résumer, voici le bloc-diagramme du banc d'essai AX.25 :

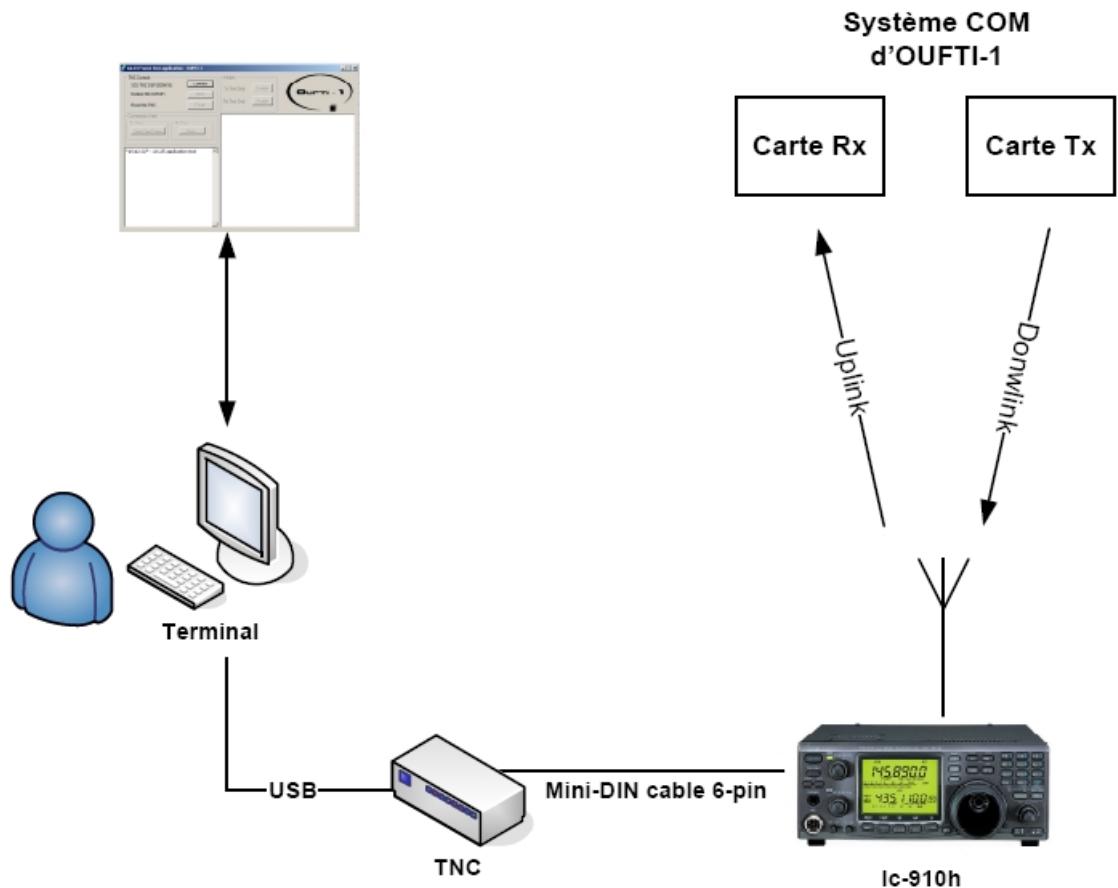


FIGURE 10.2 – Banc d'essai AX.25

### 10.1.1 Schéma de câblage du banc d'essai

Le câblage du banc d'essai n'est pas très compliqué. Pour brancher le TNC à l'ordinateur, un câble USB classique sera suffisant (les *drivers* du TNC se trouvent sur le CD-ROM fourni). La connexion entre le *transceiver* et le TNC se fera via un câble mini-din 6 pins. Enfin, les deux appareils sont alimentés en 13,8 V. Au niveau du TNC, la consommation en courant est vraiment très faible (moins d'un ampère). Par contre, le *transceiver* peut atteindre des intensités de 10 A (lorsque la puissance du signal est grande). Par conséquent, il faut être prudent pendant la manipulation.

Le schéma de câblage se trouve à la figure qui suit.

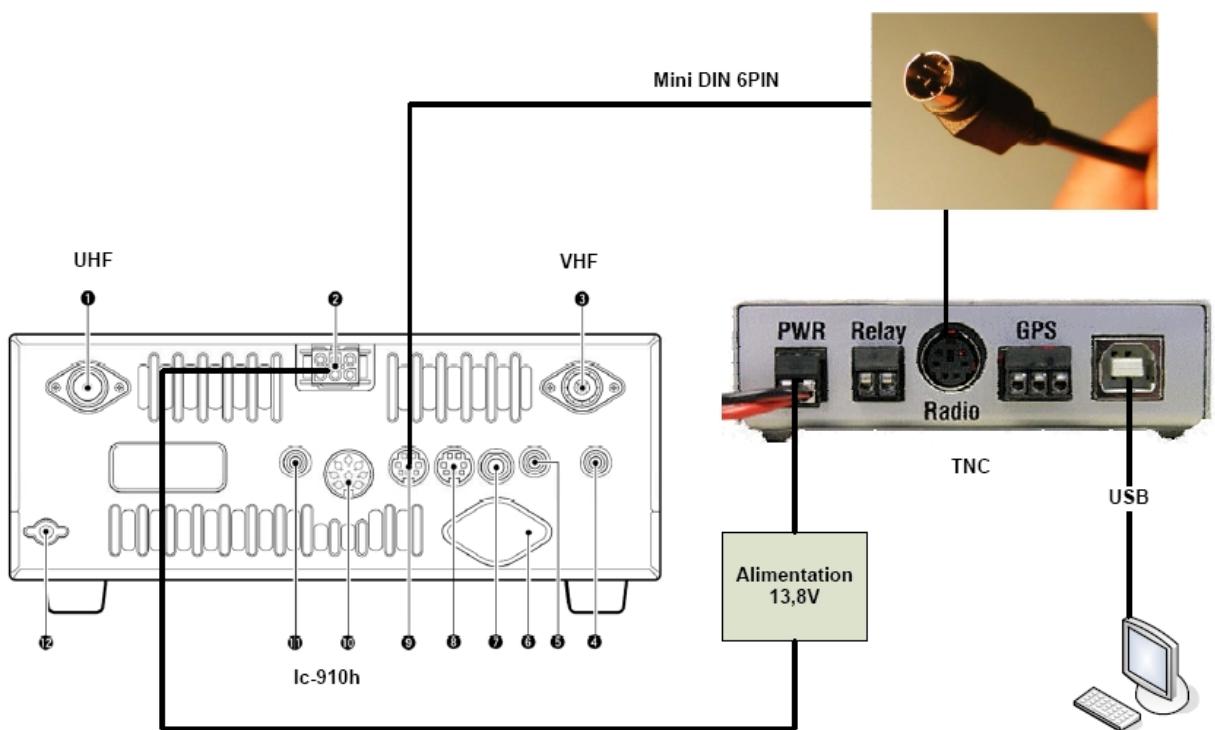


FIGURE 10.3 – Câblage du banc d'essai AX.25

### 10.1.2 Configuration de l'Ic-910h pour le 9600 baud

L'Ic-910h supporte le mode 9600 baud. Pour passer en mode 9600, il suffit d'appuyer 3 secondes sur la touche **SET**. Lorsque le menu apparaît, il faut ensuite trouver le mode 9600 en déroulant le menu grâce aux touches **UP** et **DOWN**. Enfin, une fois le mode 9600 trouvé (Figure 10.4), il suffit de tourner la roulette principale pour passer en mode **ON**.

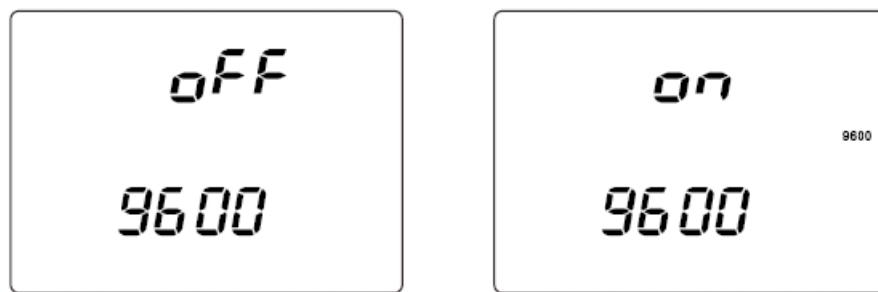


FIGURE 10.4 – Mode 9600

Pour confirmer, il suffit d'appuyer une fois sur **SET**. Au niveau de la bande de fréquence, il faut configurer l'Ic-910h en mode FM et se mettre sur la bonne fréquence. A ce stade, si tout le matériel est correctement branché et configuré, le banc d'essai est opérationnel.

## 10.2 Essais en transmission

L'essai en transmission est la mise en oeuvre du *downlink* d'OUFTI-1. Seule la carte Tx a été utilisée<sup>2</sup>.

### 10.2.1 Déroulement de l'essai

L'essai consiste à émettre une trame *UI* toutes les 4 secondes à partir de la carte Tx. Le but est bien sûr de réceptionner une trame avec l'Ic-910h/TNC et d'afficher son contenu (en hexadécimal) au moyen de l'interface graphique.

Pour cet essai, le champ Info de la trame *UI* contiendra 256 fois 0xFF. Le choix d'envoyer 256 fois 0xFF permettra de valider l'algorithme du *bit stuffing*. En effet,  $(FF)_h$  vaut  $(11111111)_b$  donc un *stuffed bit* devra être inséré tous les 5 bits à 1.

#### Critère de réussite

Le principal critère de réussite est d'**afficher le contenu de la trame émise**. En fait, l'affichage de la trame prouvera que la trame émise est valide. En effet, le TNC vérifie le FCS de la trame émise et ne renvoie que les données AX.25 au format KISS via USB. Si une trame KISS est envoyée du TNC à l'ordinateur via USB, il en résulte que le FCS, le *bit stuffing*, le codage NRZI et le *scrambler* sont réalisés correctement dans la carte Tx.

### 10.2.2 Configuration de l'ADF7021

La configuration de l'ADF7021 a été très pénible. Cependant, les bons paramètres ont été trouvés. En se référant à sa datasheet, plusieurs registres doivent être configurés. Les principaux paramètres et le calcul des registres de configuration se trouvent dans un fichier Excel (voir CD-ROM) et fait l'objet des travaux [13] et [14]. Une fois que ces registres sont déterminés, ils sont envoyés en série sur la broche SDATA à fréquence de SCLK de l'ADF7021<sup>3</sup>.

Les principaux paramètres sont les suivants :

- Mode Tx.
- Fréquence de la porteuse : 434,000 MHz (UHF) (et plus exactement 433,98667 MHz).
- Déviation de fréquence : 4800 Hz.
- *Data rate* : 9600 bps.
- Modulation : 2-FSK.

En entrant ces paramètres<sup>4</sup> dans la feuille de calculs réalisée par [13] et [14], nous obtenons les bons registres ; à savoir :

```
char ADFR0[4] = { 0b11000000 , 0b11011111 , 0b01110011 , 0b00000010 };
char ADFR1[4] = { 0b10010001 , 0b01010000 , 0b01000111 , 0b00000000 };
char ADFR2[4] = { 0b10000010 , 0b00110000 , 0b11000000 , 0b00000001 };
char ADFR3[4] = { 0b10010011 , 0b01001000 , 0b10111100 , 0b00101001 };
char ADFR8[4] = { 0b00011000 , 0b00000001 , 0b00000000 , 0b00000000 };
```

<sup>2</sup>Accompagnée, bien entendu, du reste du banc d'essai.

<sup>3</sup>Voir Figure 8.2

<sup>4</sup>Pour plus de détails voir feuille Excel sur CD-ROM.

### 10.2.3 Résultats de l'essai

Après compilation du programme Tx dans l'IDE et programmation de l'MSP430, la carte Tx commençait à émettre. En précisant à l'Ic-910h la bonne fréquence (434,000 MHz), on pouvait entendre au haut parleur du *transceiver* la trame passer. Distantes de 4 secondes, les trames sont tout à fait nettes à l'écoute. Au passage d'une trame, la led DCD (Digital Carrier Detection) du TNC s'alluma ; synonyme de bonne synchronisation au niveau des bits. Enfin, au niveau de l'interface graphique, nous obtenons ceci :

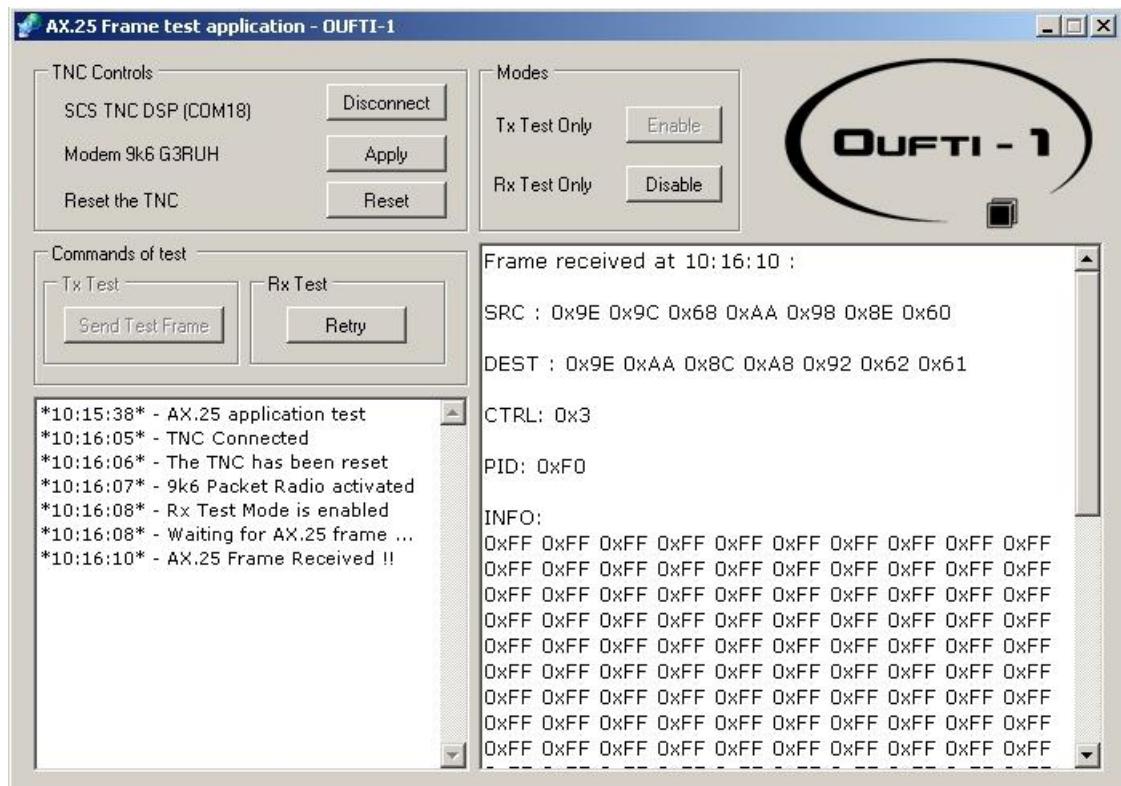


FIGURE 10.5 – *Screenshot* de l'interface graphique au moment de l'essai Tx

### Interprétations des résultats

A la vue de ce *screenshot*<sup>5</sup>, on peut dire que l'essai est couronné de succès. En effet, le *header*<sup>6</sup> ainsi que le champ Info, rempli de 0xFF, correspondent bien à nos attentes.

La trame émise de la carte Tx a donc bien été reçue. Cette considération implique donc que le critère a été rempli et que le FCS, le *bit stuffing*, le codage NRZI et le *scrambler* sont réalisés correctement dans la carte Tx.

### 10.2.4 Optimisation possible

L'essai a été réalisé en UHF (434,000 MHz). Cependant, le *downlink* d'OUFTI-1 utilisera les bandes VHF. Donc il faudrait changer la fréquence et essayer en 145,000 MHz par exemple. En revanche, au niveau technique, cela ne change rien. Le passage de UHF à VHF se résume

<sup>5</sup>Une erreur d'affichage s'est tout de même glissée mais ne remet pas en cause l'essai. DEST, doit prendre la place de SRC. En effet, 0x9E 0x9C 0x68 0xAA 0x98 0x8E correspond à l'indicatif de la destination (DEST) et 0x9E 0xAA 0x8C 0xA8 0x92 0x62 correspond à l'indicatif de la source (SRC).

<sup>6</sup>Voir la mise en forme du header AX.25 ; point 8.3.1 et l'exemple du champ Address ; point 4.2.2

à changer un paramètre dans l'ADF7021. Mise à part cette optimisation, le code fonctionne parfaitement.

## 10.3 Essais en transmission/réception des prototypes

A ce stade de la phase de tests, nous sommes à présent sûrs que la carte Tx émet des trames AX.25 valides. Par conséquent, un test entre les deux prototypes a été réalisé. Cet essai prépare en fait le test Rx final. En effet, les caractéristiques HF de l'Ic-910h étant inconnues, il vaut mieux vérifier d'abord son code avec du matériel que l'on maîtrise (c'est-à-dire les deux cartes prototypes).

### 10.3.1 Déroulement de l'essai

L'essai se limite donc aux cartes Tx et Rx<sup>7</sup>. Cet essai consiste donc à faire fonctionner la carte Tx en permanence et de tenter de récupérer la trame émise, non pas avec l'Ic-910h et le TNC mais avec la carte Rx<sup>8</sup>.

Pour y parvenir, il faut configurer l'ADF7021 de la carte Rx correctement. Pour ce faire, nous ferons appel au diagramme de l'oeil.

### 10.3.2 Configuration de l'ADF7021

Par analogie à l'essai précédent, il existe également une feuille de calculs Excel<sup>9</sup> permettant de calculer les registres de configuration de l'ADF7021 en réception.

Au niveau des paramètres, nous connaissons déjà les paramètres de la carte Tx<sup>10</sup>. Par conséquent, il ne reste plus qu'à affiner les paramètres pour trouver un oeil satisfaisant.

En reprenant les paramètres du point 10.2.2, le diagramme de l'oeil est très mauvais :

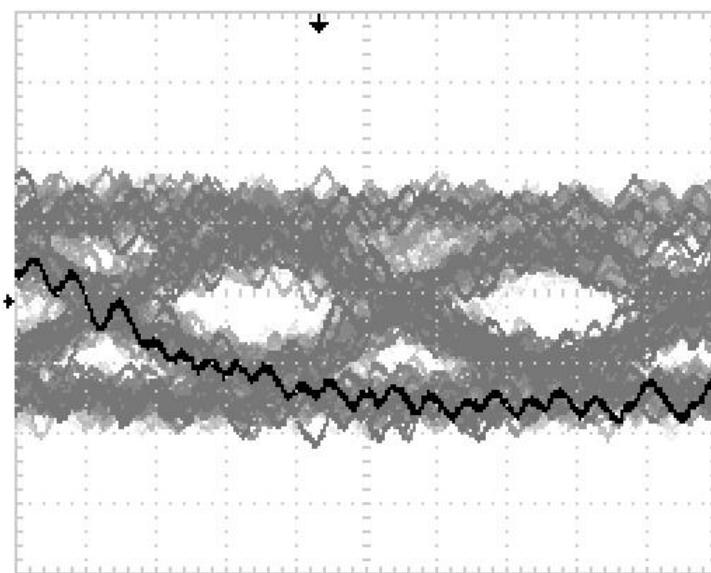


FIGURE 10.6 – Diagramme de l'oeil - Entre cartes Rx et Tx (sans affinage des paramètres HF)

<sup>7</sup>C'est-à-dire sans le TNC et l'Ic-910h.

<sup>8</sup>Il s'agit donc d'un essai intermédiaire avant le test final en Rx (c'est-à-dire avec l'Ic-910h et le TNC).

<sup>9</sup>Grâce à [13] et [14].

<sup>10</sup>Voir point 10.2.2

Les bons paramètres HF ont finalement été trouvés en itérant. Voici les paramètres de l'ADF7021 et le diagramme de l'oeil beaucoup plus net :

- Mode Rx.
- Fréquence de la porteuse : 434,000 MHz (UHF) (et plus exactement 433,89 MHz).
- Déviation de fréquence : 4800 Hz.
- *Data rate* : 9600 bps.
- Modulation : 2-FSK.
- IF filter : 18 kHz.

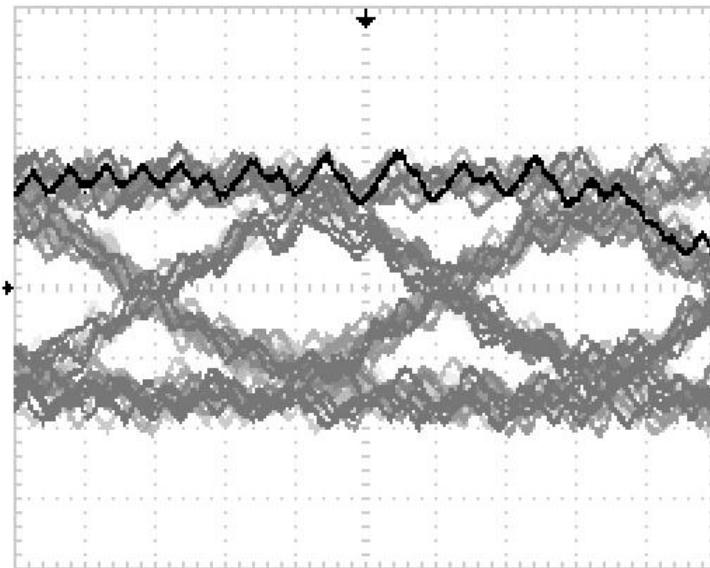


FIGURE 10.7 – Diagramme de l'oeil - Entre cartes Rx et Tx (avec affinage des paramètres HF)

L'oeil observé est à présent satisfaisant pour effectuer l'essai. En effet, grâce à l'oscilloscope, nous pouvons vérifier l'intégrité de la liaison. Une acquisition de signal a été réalisée sur la sortie de l'**MSP430** de la carte **Tx** (c'est-à-dire sur la broche TxRxDATA correspondant aux bits destinés à être modulé) :

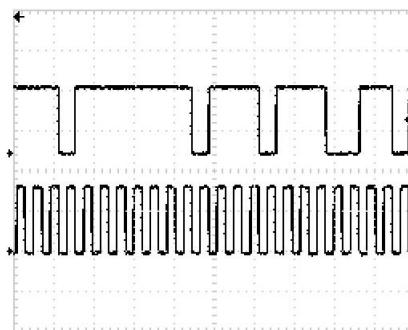


FIGURE 10.8 – Signal de sortie de l'MSP430 de la carte Tx

La même acquisition a été réalisée, cette fois-ci sur la broche de sortie de l'**ADF7021** de la carte **Rx** (c'est à dire sur la broche TxRxDATA correspondant aux bits démodulés) :

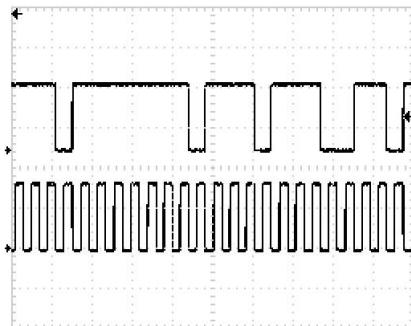


FIGURE 10.9 – Signal de sortie de l'ADF7021 de la carte Rx

Si nous comparons maintenant les deux figures précédentes, elles sont identiques. Par conséquent, la liaison HF est plus que satisfaisante et l'essai peut être mené.

### 10.3.3 Résultats de l'essai

Après compilation du programme Rx dans l'IDE et programmation de l'MSP430, la carte Rx est désormais prête à recevoir une trame. A la mise sous tension de la carte Tx et après 4 secondes, la carte Rx a atteint le *breakpoint* attendu. En vérifiant le buffer de la carte Rx, la trame attendue était bien présente<sup>11</sup>.

Cet essai confirme donc que le code Rx est fonctionnel et que le test final peut démarrer.

## 10.4 Essais en réception

Fort de l'expérience réalisée entre les deux cartes, le test Rx final a de grandes chances de réussir. L'essai en réception est donc la mise en oeuvre de l'*uplink* d'OUFTI-1. Seule la carte Rx sera utilisée<sup>12</sup>.

### 10.4.1 Déroulement de l'essai

L'essai consiste à émettre une trame *UI* à partir de l'Ic-910h/TNC et la décoder grâce à la carte Rx. Avec l'interface graphique, il est très facile d'envoyer une trame, il suffit d'appuyer sur le bouton « *send test frame* » avec la souris. Une fois la trame émise, si tout se déroule correctement, la trame sera stockée dans le buffer de l'MSP430 de la carte Rx.

#### Critère de réussite

Le principal critère de réussite est de retrouver les mêmes bytes émis, à partir de l'**interface graphique**, dans le **buffer de l'MSP430 de la carte Rx**. Si tel est le cas, nous pourrons conclure que la vérification FCS, la suppression du *bit stuffing*, le décodage NRZI et le *descrambler* sont réalisés correctement dans la carte Rx dans des conditions réelles (c'est-à-dire avec l'Ic-910h, le TNC et l'interface graphique).

<sup>11</sup>Malheureusement, aucun *screenshot* n'a été réalisé car seul l'essai Rx final importe.

<sup>12</sup>Accompagnée, bien entendu, du reste du banc d'essai.

### 10.4.2 Configuration de l'ADF7021

Grâce à l'essai précédent, nous connaissons déjà les paramètres HF en réception avec la carte Tx. Cependant, il faut maintenant les affiner avec l'Ic-910h dont voici le spectre :



FIGURE 10.10 – Spectre de l'Ic-910h

Au niveau des paramètres HF, ceux-ci ont également été trouvés en itérant. Voici les paramètres de l'ADF7021 et le diagramme de l'oeil entre l'Ic-910h et la carte Rx :

- Mode Rx.
- Fréquence de la porteuse : 434,000 MHz (UHF) (et plus exactement 433,8912 MHz).
- Déviation de fréquence : 4800 Hz.
- *Data rate* : 9600 bps.
- Modulation : 2-FSK.
- IF filter : 18 kHz.

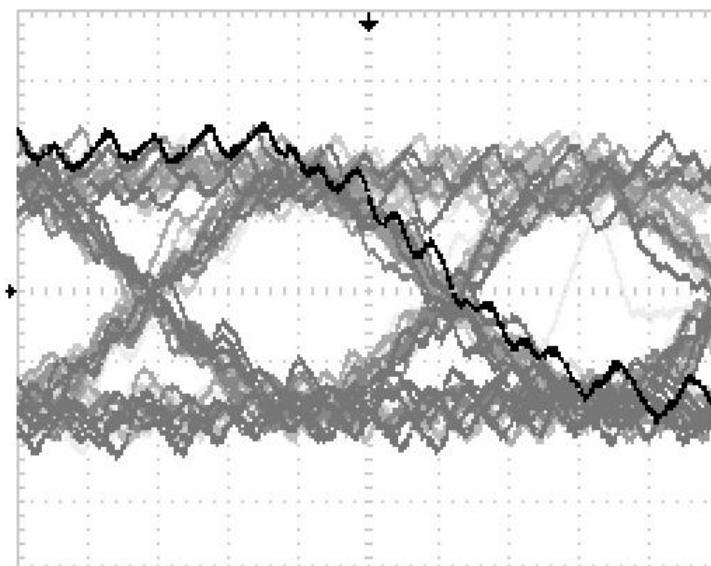


FIGURE 10.11 – Diagramme de l'oeil - Entre la carte Rx et Ic-910h

En entrant les paramètres<sup>13</sup> HF dans la feuille de calculs de [13] et [14], nous obtenons les bons registres ; à savoir :

```
char ADFR0[4] = { 0b01110000 , 0b10111100 , 0b01110011 , 0b00001010 };
char ADFR1[4] = { 0b00010001 , 0b01010000 , 0b01000111 , 0b00000000 };
char ADFR3[4] = { 0b11010011 , 0b00110000 , 0b10111000 , 0b00101001 };
char ADFR4[4] = { 0b10010100 , 0b00001010 , 0b11010011 , 0b10000000 };
char ADFR5[4] = { 0b10110101 , 0b00011011 , 0b00000000 , 0b00000000 };
char ADFR6[4] = { 0b000000110 , 0b000000000 , 0b000000000 , 0b000000000 };
char ADFR8[4] = { 0b11011000 , 0b00000111 , 0b000000000 , 0b000000000 };
```

### 10.4.3 Résultats de l'essai

Après compilation du programme Rx dans l'IDE et programmation de l'MSP430, la carte Rx est en pleine attente d'une trame *UI*. En précisant à l'Ic-910h la bonne fréquence (434,000 MHz), il ne reste plus qu'à émettre grâce à l'interface graphique. En enfonçant le bouton « *send test frame* », l'Ic-910h passe en émission et le TNC émet son signal. Enfin, au niveau de l'interface graphique, nous observons :

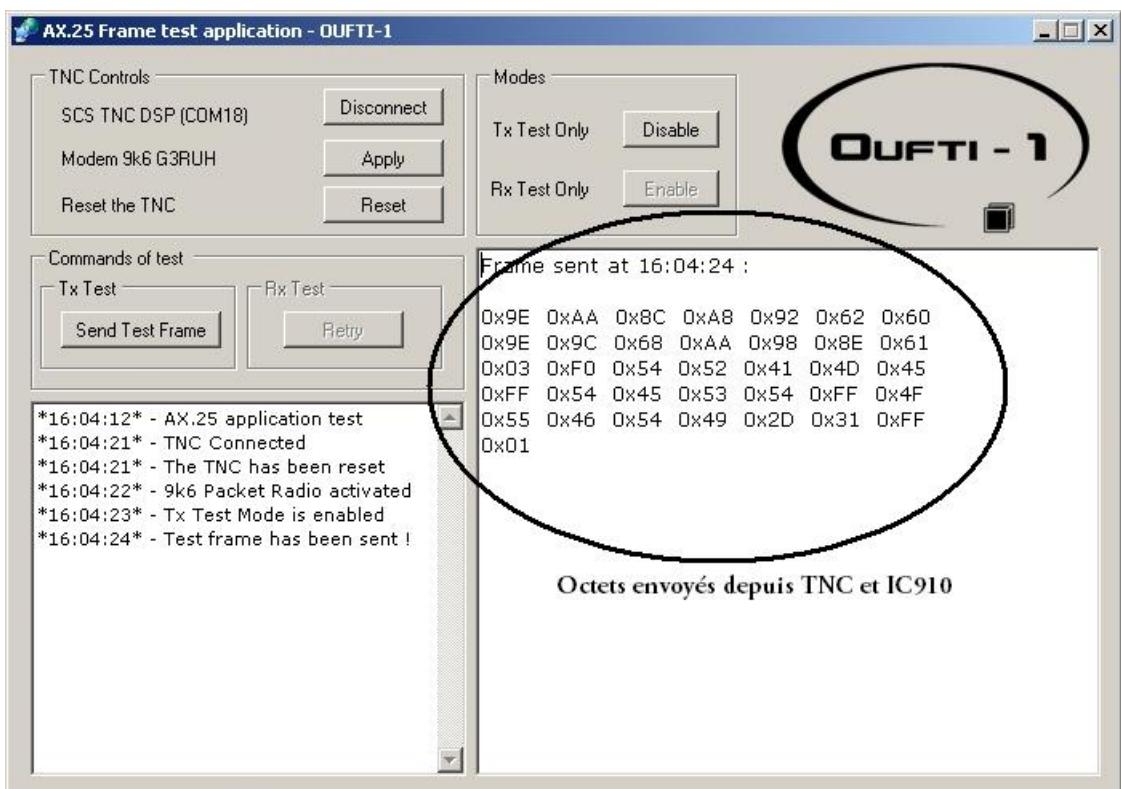


FIGURE 10.12 – Screenshot de l'interface graphique au moment de l'essai Rx

<sup>13</sup>Pour plus de détails voir feuille Excel sur CD-ROM.

et au niveau de l'IDE (CROSSWORKS STUDIO), nous observons :

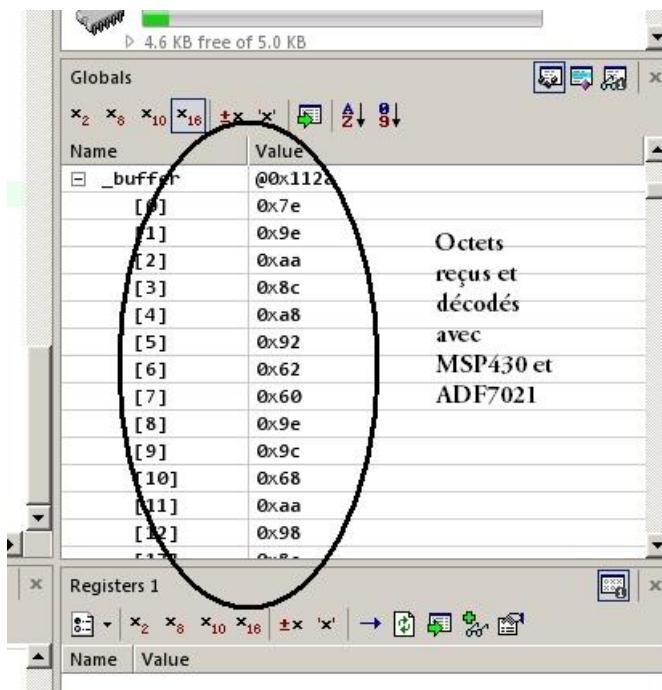


FIGURE 10.13 – *Screenshot* de l'IDE au moment de l'essai Rx

### Interprétations des résultats

A la vue de ces deux figures, nous pouvons constater que les bytes émis se retrouvent dans le buffer<sup>14</sup> de l'MSP430 de la carte Rx. L'essai est donc couronné de succès.

#### 10.4.4 Optimisation possible

Au niveau de l'optimisation, il faut bien sûr améliorer le diagramme de l'oeil qui reste tout de même fort bruité. A priori, le bruit constaté provient du circuit ou de l'environnement régnant dans le laboratoire au moment des essais. Peut-être que dans des conditions d'essai plus adéquates et avec un PCB d'*engineering model*, le diagramme de l'oeil sera encore meilleur.

<sup>14</sup>0x7E étant le premier flag.

## Quatrième partie

### Conclusions et annexes

# Chapitre 11

## Conclusions

Après plusieurs mois d'intérêt<sup>1</sup> à ce projet, ce dernier chapitre fait en quelque sorte le bilan de ce travail de fin d'études. Concernant les objectifs de ce dernier, ils furent fixés assez tardivement. Il faut savoir que le projet OUFTI-1 est un projet ambitieux, tout à fait novateur et qui n'a encore jamais été tenté. Il est évident que dans un tel projet, les choix de *design* sont souvent remis en question ce qui amène beaucoup de revirements de situation. Le projet a également une dimension spatiale qui ajoute aux problèmes initiaux des contraintes supplémentaires. Le facteur humain est aussi très important. En effet, le fait de travailler en équipe a permis de mettre en évidence les qualités et les défauts de chacun. Cependant, au niveau du projet OUFTI-1, l'esprit d'équipe fut présent et la motivation quant à la réalisation possible de ce satellite était grande.

Remarquons également que le projet OUFTI-1 est un projet pluridisciplinaire, il fallait donc être autodidacte. Par exemple, dans le cas de l'AX.25 et dans le cas de ce travail de fin d'études, plusieurs démarches ont dû être menées. Partant de la case départ, il fallait étudier le protocole et trouver une solution adaptée au satellite OUFTI-1. L'AX.25 est un protocole très spécifique et assez peu connu hors du contexte radioamateur. De plus, ce protocole est généralement utilisé pour des applications terrestres. Par conséquent, cette implémentation est assez originale et unique au satellite OUFTI-1. Plus globalement, ce travail de fin d'études consistait donc à établir une liaison satellite. Pour y parvenir, la documentation et les personnes ressources étaient très difficiles à trouver. Un séjour à Lausanne a notamment été entrepris pour élucider certains problèmes et zones d'ombre. Le fait de sortir de ses frontières permet également de faire des comparaisons entre ce qu'on fait et ce que font les autres (ce qui permet de se positionner sur un problème et prendre un peu d'abstraction). Dans la même optique, il a fallu s'adapter aux contraintes et aux choix de *design* (notamment l'ADF7021). Ces choix de *design*, parfois pris dans les 5 dernières minutes d'une réunion ou à la suite d'une réunion qui a duré 4 heures, sont parfois très déterminants. Lorsque les décisions sont prises, on se rend vite compte si celles-ci étaient bonnes ou pas.

Si nous en revenons aux objectifs de ce travail de fin d'études<sup>2</sup> et après lecture des pages précédentes, nous pouvons dire que les principaux objectifs ont été accomplis. Le travail accompli n'a cependant pas pu remplir les deux objectifs secondaires pour les raisons qui suivent. En effet, le premier objectif secondaire consistait à réécrire le code AX.25 dans l'environnement FreeRTOS de l'OBC. Cette réécriture pouvait encore prendre relativement beaucoup de temps en développement et n'aurait pu être vérifiée (ce qui constitue le deuxième objectif secondaire). Pour tester ce code, il aurait fallu avoir à disposition les cartes COM d'*engineering model* qui doivent être interfacées à l'OBC. N'ayant aucune carte à disposition

---

<sup>1</sup>Plus d'un an.

<sup>2</sup>Voir point 3.3

---

et n'ayant aucun moyen de vérifier le code (mis à part des simulations possibles), la réécriture du code aurait été vaine. Cependant, le code existant a été pensé pour être réutilisé à ces fins (moyennant quelques adaptations).

Concernant les aspects techniques, le composant ADF7021 est valide pour opérer en AX.25 FSK. Il faut cependant remarquer que le SWD de l'ADF7021 n'est pas envisageable pour de l'AX.25 9600 baud G3RUH. En tenant compte de cette considération, le système de réception implique une augmentation du nombre d'interruptions dans le microcontrôleur. Etant donné que les opérations du protocole AX.25 et de la gestion des TM/TC étaient initialement prévues dans l'OBC, il serait effectivement judicieux de revoir l'architecture OBC-ADF7021. En effet, devant déjà accomplir plusieurs tâches (tâches vitales pour la survie du satellite), l'OBC serait submergé par le nombre d'interruptions et ne pourrait pas garantir un fonctionnement optimal. Si la configuration le permet, ajouter un microcontrôleur supplémentaire dédiacé à la COM est une solution envisageable.

Pour conclure, il convient de dire que le projet OUFTI-1 était une occasion unique à saisir. En effet, ce projet était plus qu'un simple projet ; il s'agit véritablement d'une entreprise. Les étudiants qui y ont participé témoignent d'un réel esprit d'entreprendre. Ce projet a donc fait appel à beaucoup de compétences. Enfin, ces derniers mots sonnent la fin d'une première phase de développement du satellite OUFTI-1. Espérons qu'un jour il soit en orbite et qu'il rende fier les personnes qui ont contribué à sa réalisation ...

# Bibliographie

- [1] CALPOLY, *CubeSat Design Specification REV. 11*, San Luis Obispo, 2008.
- [2] CALPOLY, *Poly Picosatellite Orbital Deployer Mk III ICD*, San Luis Obispo, 2008.
- [3] COLLETTE J.P., *HELMO Gramme - Construction aéronautique et spatiale - Première partie*, Liège, édition Jipé, 2008.
- [4] PIERLOT G., *Oufti-1 : structural integrity, system configuration and vibration testing*, ULg, 2009.
- [5] BEUKELAERS V., *From mission analysis to space flight simulation of the OUFTI-1 nano-satellite*, ULg, 2009.
- [6] WERZ J., *Conception et réalisation du système de déploiement des antennes du nanosatellite OUFTI-1*, Institut Gramme, 2009.
- [7] JACQUES L., *Thermal Design of the OUFTI-1 nanosatellite*, ULg, 2009.
- [8] HANNAY S., *Modeling of the Attitude Determination and Control System of the nano-satellite OUFTI-1*, ULg, 2009.
- [9] TENEY D., *Design and implementation of on-board processor and software of nanosatellite OUFTI-1 of University of Liège*, ULg, 2009.
- [10] EVRARD N., *Développement de l'infrastructure de mesure du nanosatellite OUFTI-1*, Institut Gramme, 2009.
- [11] THIRION P., *Design and Implementation of On-board Electrical Power Supply of Student Nanosatellite OUFTI-1 of University of Liege*, ULg, 2009.
- [12] LEDENT P., *Design and Implementation of On-board Digitaly Controled Electrical Power Supply of Student Nanosatellite OUFTI-1 of University of Liege*, ULg, 2009.
- [13] HENRARD R., *Réalisation du système de télécommunication du satellite OUFTI-1*, ISIL, 2009.
- [14] MAHY F., *Design and Implementation of On-board Telecommunication System of Student Nanosatellite OUFTI-1 of University of Liège*, ULg, 2009.
- [15] BEECH W. A., NIELSEN D. E., et TAYLOR J., *AX.25 Link Access Protocol for Amateur Packet Radio v2.2*, 1998, sur [http://www.tapr.org/pub\\_ax25.html](http://www.tapr.org/pub_ax25.html).
- [16] CHIARELLO L., *Design and implementation of the control and monitoring system for the ground station of nanosatellite OUFTI-1 of University of Liège*, ULg, 2009.

- [17] NEWHALL B., *The Cyclic Redundancy Check (CRC) for AX.25*, Colorado, 2008, sur [http://ecee.colorado.edu/newhallw/TechDepot/AX25CRC/CRC\\_for\\_AX25.pdf](http://ecee.colorado.edu/newhallw/TechDepot/AX25CRC/CRC_for_AX25.pdf).
- [18] SERVIN C., *Réseaux et télécoms 2e édition*, Dunod, 2003, Sciences Sup.
- [19] SIMPSON W., *PPP in HDLC-like Framing*, 1994, p. 18, sur <http://www.rfc-archive.org/getrfc.php?rfc=1662>.
- [20] WIKIPEDIA, *Bell 202 modem - Wikipedia The Free Encyclopedia*, 2009, sur [http://en.wikipedia.org/w/index.php?title=Bell\\_202\\_modem&oldid=286119046](http://en.wikipedia.org/w/index.php?title=Bell_202_modem&oldid=286119046).
- [21] TRETTER S. A., *Continuous-Phase Frequency Shift Keying (FSK)*, 2004, sur <http://www.ece.umd.edu/class/enee429w.F99/fsk.pdf>
- [22] BARRY J. R., MESSERSCHMITT D. G., et LEE E. A., *Digital Communication : Third Edition*, Springer, 2003.
- [23] CARLSON A. B., et CRILLY P., *Communication Systems*, Mc Graw-Hill Publishing Co., 2001.
- [24] AZAN J.-L., *Précis d'électronique 2e année*, Editions Bréal, 2005.
- [25] MILLER J., *9600 Baud Packet Radio Modem Design*, sur <http://www.amsat.org/amsat/articles/g3ruh/109.html>.
- [26] SEABE P. S., *Evaluation Of Microcontroller Based Packet Radio Modem*, 2007, sur <http://etd.sun.ac.za/jspui/handle/10019/461>.
- [27] HALLIDAY L., *Communications Infrastructure for the MOST Microsatellite Project (excerpt)*, 2000, sur <http://www.qsl.net/ve7ldh/ax25.pdf>.
- [28] *Les modems 9600 bauds*, sur <http://pagesperso-orange.fr/f1my/g3ruh.htm>.
- [29] MERZ R., *comSC*, 2008, sur <http://comsc.sourceforge.net/>.
- [30] SCS, *Instruction Manual/Command Description for the Tracker/DSPTNC*, sur CD-ROM fourni avec le TNC.
- [31] RTBF, *Planète Première : satellite "Oufti"*, sur <http://www.rtbf.be/info/matin-premiere/planete-premiere-satellite-oufti-83980>.

# Table des matières

<b>Remerciements</b>	<b>4</b>
<b>Acronymes</b>	<b>5</b>
<b>1 Introduction</b>	<b>6</b>
<b>I Préambule</b>	<b>8</b>
<b>2 Présentation du CubeSat OUFTI-1</b>	<b>9</b>
2.1 Le standard CubeSat . . . . .	9
2.2 Le lanceur Vega . . . . .	12
2.3 Mission . . . . .	13
2.3.1 Objectifs . . . . .	13
2.3.2 Payloads . . . . .	13
2.4 Le CubeSat OUFTI-1 . . . . .	14
2.5 L'environnement spatial . . . . .	17
<b>3 Cahier des charges</b>	<b>20</b>
3.1 Les télécommandes et télémétries . . . . .	20
3.2 Les TM/TC et l'AX.25 . . . . .	21
3.2.1 L'effet Doppler . . . . .	21
3.2.2 Solution . . . . .	21
3.3 Objectifs du travail de fin d'études . . . . .	22
<b>II Définition et description des concepts</b>	<b>23</b>
<b>4 Description du protocole AX.25</b>	<b>24</b>
4.1 Généralités . . . . .	24
4.1.1 Modèles AX.25 . . . . .	25
4.2 Structure de la trame AX.25 . . . . .	26
4.2.1 Flag . . . . .	27
4.2.2 Address . . . . .	27
4.2.3 Control . . . . .	28
4.2.4 PID . . . . .	29
4.2.5 Info . . . . .	29
4.2.6 FCS . . . . .	30
4.3 Taille d'une trame . . . . .	30
4.4 Les modes . . . . .	30

4.4.1	Mode connecté et mode non-connecté . . . . .	30
4.5	Mise en oeuvre du protocole AX.25 pour OUFTI-1 . . . . .	30
4.5.1	Trame UI . . . . .	31
4.5.2	Le CCSDS . . . . .	32
4.6	Avantages et inconvénients de l'AX.25 . . . . .	32
<b>5</b>	<b>Frame checksum AX.25</b>	<b>33</b>
5.1	Principe . . . . .	33
5.2	Caractéristiques du FCS AX.25 . . . . .	34
5.3	Implémentation . . . . .	34
5.4	Code source . . . . .	34
5.5	Vérification . . . . .	36
<b>6</b>	<b>Le terminal node controller</b>	<b>38</b>
6.1	Principe général de fonctionnement . . . . .	38
6.2	Les baud rates . . . . .	38
6.3	Les modulations . . . . .	39
6.3.1	1200 baud A-FSK . . . . .	39
6.3.2	9600 baud FSK . . . . .	41
6.4	Types de TNC . . . . .	45
6.4.1	Le TNC software . . . . .	45
6.4.2	Le TNC hardware . . . . .	46
6.5	Le SCS Tracker/DSP TNC . . . . .	46
6.6	Le mode KISS . . . . .	47
<b>7</b>	<b>Le modem G3RUH</b>	<b>48</b>
7.1	Généralités . . . . .	48
7.2	Le codage NRZI . . . . .	49
7.3	Brouillage . . . . .	50
7.3.1	Le scrambler . . . . .	50
7.3.2	Le descrambler . . . . .	51
<b>III</b>	<b>Aspects logiciels</b>	<b>53</b>
<b>8</b>	<b>Implémentation de la trame UI du protocole AX.25</b>	<b>54</b>
8.1	Environnement hardware . . . . .	54
8.1.1	Le microcontrôleur MSP430 . . . . .	54
8.1.2	L'ADF7021 . . . . .	55
8.2	Choix et description de l'implémentation . . . . .	56
8.2.1	Choix d'implémentation . . . . .	56
8.2.2	Description de l'implémentation . . . . .	57
8.3	Implémentation logicielle de la partie Tx . . . . .	58
8.3.1	Mise en forme du header AX.25 . . . . .	58
8.3.2	Chaîne d'implémentation . . . . .	58
8.3.3	Schéma haut niveau . . . . .	59
8.3.4	Les variables globales . . . . .	61
8.3.5	Assemblage de la trame . . . . .	61
8.3.6	Séquence d'envoi . . . . .	62
8.3.7	Communication avec l'ADF7021 . . . . .	63

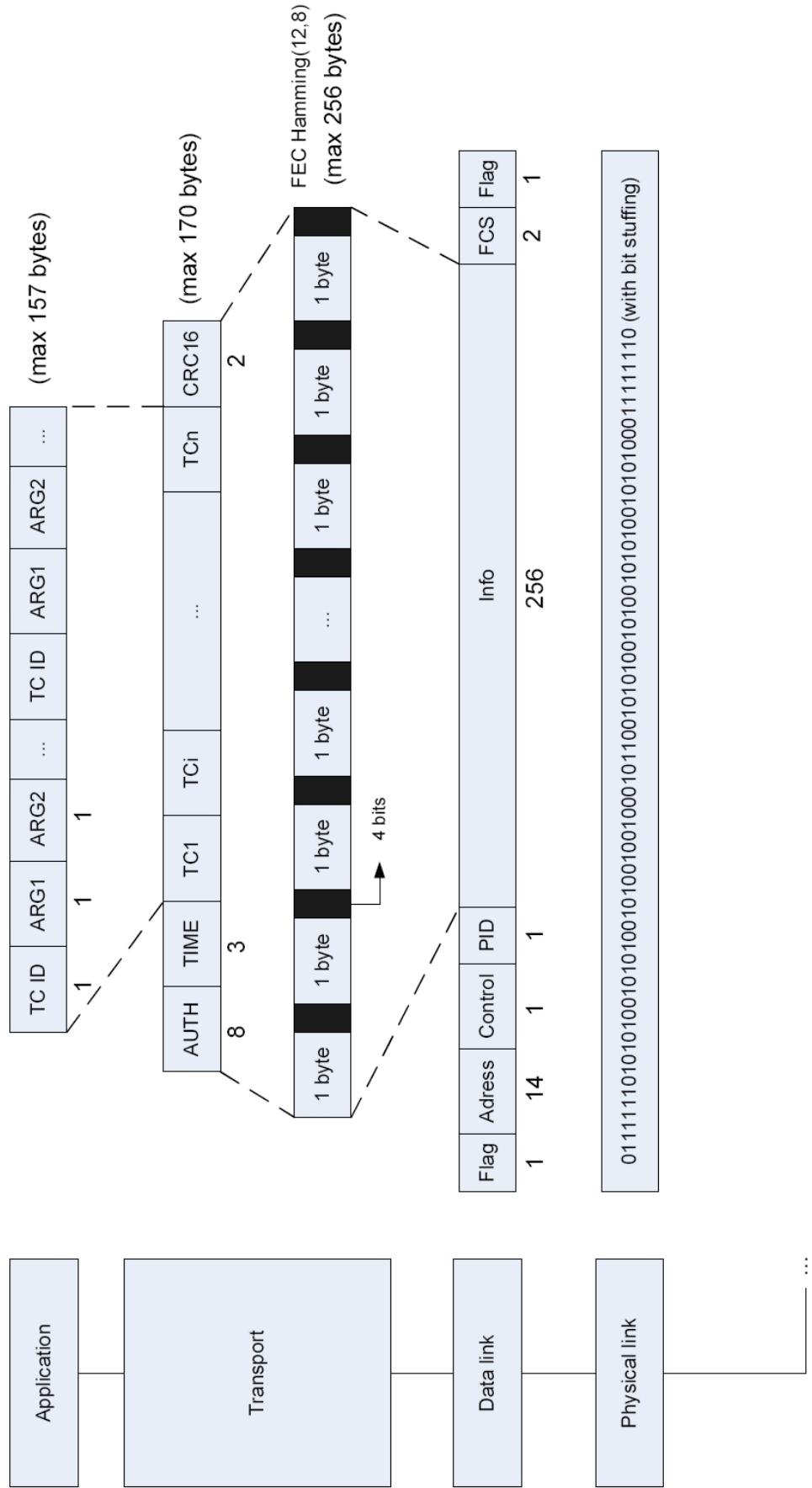
8.4	Implémentation logicielle de la partie Rx . . . . .	64
8.4.1	Chaîne d'implémentation . . . . .	64
8.4.2	Schéma haut niveau . . . . .	64
8.4.3	Les variables globales . . . . .	65
8.4.4	Séquence de réception . . . . .	66
8.4.5	Communication avec l'ADF7021 . . . . .	67
<b>9</b>	<b>Interface graphique</b>	<b>69</b>
9.1	Principe de fonctionnement . . . . .	69
9.2	Configuration du TNC . . . . .	70
9.3	Méthodes et format KISS . . . . .	71
9.4	Class TNC . . . . .	71
9.5	Aperçu de l'interface graphique . . . . .	72
<b>10</b>	<b>Phase de tests</b>	<b>74</b>
10.1	Matériels utilisés . . . . .	74
10.1.1	Schéma de câblage du banc d'essai . . . . .	75
10.1.2	Configuration de l'Ic-910h pour le 9600 baud . . . . .	76
10.2	Essais en transmission . . . . .	77
10.2.1	Déroulement de l'essai . . . . .	77
10.2.2	Configuration de l'ADF7021 . . . . .	77
10.2.3	Résultats de l'essai . . . . .	78
10.2.4	Optimisation possible . . . . .	78
10.3	Essais en transmission/réception des prototypes . . . . .	79
10.3.1	Déroulement de l'essai . . . . .	79
10.3.2	Configuration de l'ADF7021 . . . . .	79
10.3.3	Résultats de l'essai . . . . .	81
10.4	Essais en réception . . . . .	81
10.4.1	Déroulement de l'essai . . . . .	81
10.4.2	Configuration de l'ADF7021 . . . . .	82
10.4.3	Résultats de l'essai . . . . .	83
10.4.4	Optimisation possible . . . . .	84
<b>IV</b>	<b>Conclusions et annexes</b>	<b>85</b>
<b>11</b>	<b>Conclusions</b>	<b>86</b>
	<b>Bibliographie</b>	<b>88</b>

# Table des figures

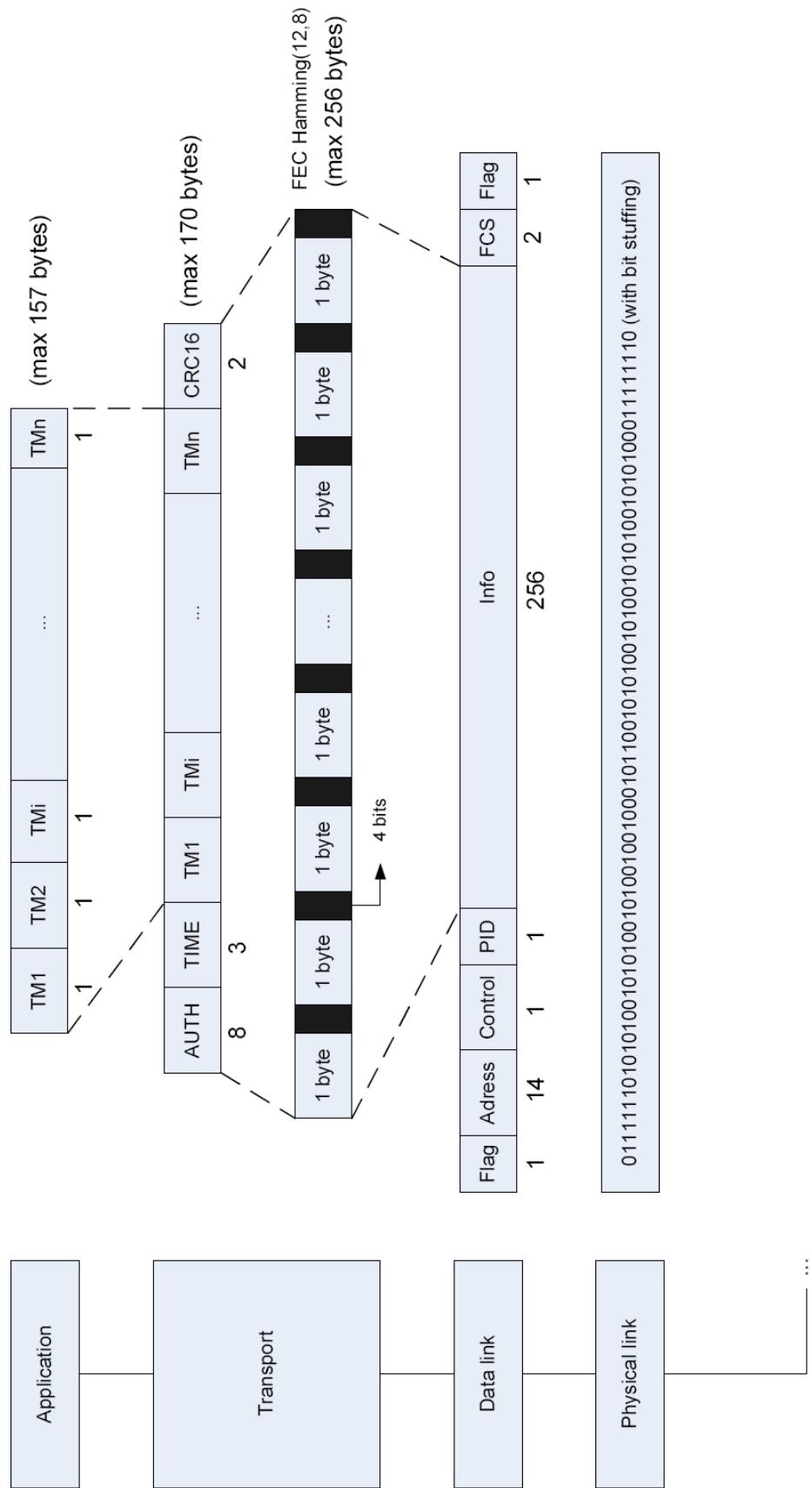
2.1	Illustration d'un CubeSat . . . . .	10
2.2	P-POD - Système de déploiement . . . . .	11
2.3	Vue de face du P-POD . . . . .	11
2.4	CubeSat Kit . . . . .	12
2.5	Cellules solaires . . . . .	13
2.6	Configuration d'OUFTI-1 . . . . .	14
2.7	Agencement des PCB . . . . .	14
2.8	Système de déploiement des antennes . . . . .	15
2.9	FM430 du CubeSat kit . . . . .	16
2.10	Débris spatiaux autour de la Terre . . . . .	18
3.1	Liaison satellite . . . . .	20
3.2	Zones compensées Doppler dans le <i>footprint</i> . . . . .	21
4.1	Illustration de la modulation FSK . . . . .	24
4.2	Modèle AX.25 - <i>Single Link</i> . . . . .	25
4.3	Modèle AX.25 - <i>Multiple Stream</i> . . . . .	25
4.4	Structure des trames U et S . . . . .	26
4.5	Structure de la trame I . . . . .	26
4.6	Format du champ Address en mode non-repeater . . . . .	27
4.7	Format du champ Address en mode repeater . . . . .	28
4.8	Définition des PID . . . . .	29
4.9	Structure de la trame UI . . . . .	31
4.10	Spécifications du champ Control pour les trames U . . . . .	31
5.1	Principe du CRC-16-CCITT . . . . .	34
6.1	Spectre du modem Bell 202 . . . . .	39
6.2	Schéma du câble audio 1200 baud . . . . .	40
6.3	Schéma du câble audio 1200 baud (bis) . . . . .	40
6.4	Signal de sortie du TNC en 1200 baud . . . . .	40
6.5	Illustration de la FSK non-cohérente . . . . .	41
6.6	Spectre de la modulation FSK non-cohérente . . . . .	41
6.7	Illustration de la FSK cohérente . . . . .	42
6.8	Chaîne de la modulation FSK . . . . .	42
6.9	Spectre de la modulation FSK cohérente (cas purement binaire, $K = 1$ ) . . . . .	44
6.10	Câblage du TNC software . . . . .	45
6.11	Aperçu de MixW . . . . .	45
6.12	Câblage du TNC hardware . . . . .	46
6.13	SCS Tracker/DSP TNC . . . . .	46

6.14 Format de la trame KISS . . . . .	47
6.15 Format du champ data de la trame KISS . . . . .	47
7.1 Illustration du codage NRZI . . . . .	49
7.2 Schéma d'implémentation du <i>scrambler</i> . . . . .	50
7.3 Schéma d'implémentation du <i>descrambler</i> . . . . .	51
8.1 Bloc-diagramme de la famille MSP430F16xx . . . . .	55
8.2 Bloc-diagramme de l'ADF7021 . . . . .	56
8.3 Chaîne d'implémentation en transmission . . . . .	59
8.4 Trame AX.25 réelle . . . . .	60
8.5 Schéma haut niveau en Tx . . . . .	60
8.6 Chronogramme ADF7021 en transmission . . . . .	63
8.7 Chaîne d'implémentation en réception . . . . .	64
8.8 Schéma haut niveau en Rx . . . . .	65
8.9 Chronogramme ADF7021 en réception . . . . .	67
9.1 Schéma de principe de l'interface graphique du TNC . . . . .	69
9.2 Format des commandes de configuration du TNC . . . . .	70
9.3 Interface graphique du TNC . . . . .	73
10.1 Icom 910h . . . . .	74
10.2 Banc d'essai AX.25 . . . . .	75
10.3 Câblage du banc d'essai AX.25 . . . . .	76
10.4 Mode 9600 . . . . .	76
10.5 <i>Screenshot</i> de l'interface graphique au moment de l'essai Tx . . . . .	78
10.6 Diagramme de l'oeil - Entre cartes Rx et Tx (sans affinage des paramètres HF) . . . . .	79
10.7 Diagramme de l'oeil - Entre cartes Rx et Tx (avec affinage des paramètres HF) . . . . .	80
10.8 Signal de sortie de l'MSP430 de la carte Tx . . . . .	80
10.9 Signal de sortie de l'ADF7021 de la carte Rx . . . . .	81
10.10 Spectre de l'Ic-910h . . . . .	82
10.11 Diagramme de l'oeil - Entre la carte Rx et Ic-910h . . . . .	82
10.12 <i>Screenshot</i> de l'interface graphique au moment de l'essai Rx . . . . .	83
10.13 <i>Screenshot</i> de l'IDE au moment de l'essai Rx . . . . .	84

## Annexe 1



## Annexe 2



## Annexe 3

```

static const unsigned short crcCcittTable[256] = {
    0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,
    0x8c48, 0x9dc1, 0xaf5a, 0xbcd3, 0xca6c, 0dbe5, 0xe97e, 0xf8f7,
    0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,
    0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,
    0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,
    0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,
    0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,
    0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,
    0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,
    0xce4c, 0xdfc5, 0xed5e, 0xfcfd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,
    0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,
    0xdecd, 0xcf44, 0xfdःdf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,
    0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,
    0xef4e, 0xfec7, 0xcc5c, 0xdd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,
    0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,
    0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,
    0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,
    0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,
    0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,
    0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,
    0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,
    0x2942, 0x38cb, 0xa50, 0xb9d9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,
    0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,
    0x39c3, 0x284a, 0x1ad1, 0xb58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,
    0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,
    0x4a44, 0xbcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,
    0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,
    0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,
    0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,
    0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,
    0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,
    0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78
};

unsigned short AX25_computeCRC(char *buffer, unsigned short length) {
    unsigned short crc = 0xffff;

    *buffer++;
    length = length-4;

    while (length--) {
        crc = (crc >> 8) ^ crcCcittTable[(crc ^ *buffer++) & 0xff];
    }
    return crc ^ 0xffff;
}

```

## Annexe 4

Durant ce travail de fin d'études, diverses activités ont été menées. En voici une courte description.

### L'espace, j'en rêve

« L'espace, j'en rêve » est un évènement qui s'est déroulé durant le mois d'octobre 2008 et initié par la DGTR<sup>3</sup>, en collaboration avec l'association Wallonie Espace. Les journées de « l'espace, j'en rêve » offraient l'opportunité aux élèves de primaire et secondaire d'entrer dans l'actualité du secteur spatial et de rencontrer des professionnels passionnés par leur métier.

Le projet OUFTI-1 était au programme. Dans les installations de l'Euro Space Center de Redu, l'équipe OUFTI-1 avait un stand avec une maquette du satellite émettant un message morse. Nous avons pu expliquer aux élèves notre projet grâce aux différents posters que nous avions confectionnés. Ces posters reprenaient tous les sous-systèmes et étaient en fait des outils de vulgarisation scientifique pour les élèves.

Enfin, l'équipe a également pu échanger quelques mots avec Marie-Dominique Simonet, ministre de la recherche, des technologies nouvelles et des relations extérieures de la région wallonne. Cette dernière était très intéressée par notre projet et nous a encouragés à aller jusqu'au bout.




---

<sup>3</sup>Direction Générale des Technologies, de la Recherche et de l'Energie.

## Space Days



Egalement, durant ce mois d'octobre 2008 se déroulait à Liège les Space Days. Cet évènement se déroulait au palais des Princes-Évêques de Liège et pendant trois jours s'enchaînaient plusieurs conférences. Le thème de 2008 était axé sur les satellites de petite taille. Par conséquent, les organisateurs des Space Days avaient convié l'équipe OUFTI-1 à présenter le projet aux industriels présents. Concernant les industriels présents, nous retrouvions : AMOS, CEGELEC, CENAERO, le CSL, DELTATEC, SABCA, SAMTECH S.A., SONACA, SPACEBEL, TECHSPACE AERO, THALES ALENIA SPACE ETCA, etc.

Etant donné que les Space Days étaient un évènement international, toutes les présentations furent en anglais. Les participants venaient de Belgique, France, Luxembourg, Espagne, Grande Bretagne, Pays-Bas, Suisse, Allemagne, etc. Personnellement, les principales conférences suivantes ont été suivies :

- Keynote de M. Thierry (Spacebel) et M. Preud'homme (Verhaert Space) : « The belgian micro-satellites approach ».
- Speech de Monique Wagner (Belgian Space Policy).
- Speech de Lionel Jacques (ULg) et François Mahy (ULg) : Présentation technique du CubeSat OUFTI-1.
- Speech de Laurent Hauser (EPFL) : Présentation technique du CubeSat SwissCube.

Les conférences furent très intéressantes. En effet, sur le plan industriel, elles permettaient de sentir les tenants et aboutissants des projets spatiaux belges. En fait, celles-ci donnaient la possibilité de sortir du format académique et d'appréhender l'aspect industriel des projets.

Enfin, cet évènement fut une bonne expérience de ce que la région wallonne était capable de faire dans le domaine spatial.

## Second European CubeSat Workshop



Du 20 au 22 janvier 2009 se déroulait le Second European CubeSat Workshop à Noordwijk (Pays-Bas) dans les installations de l'ESA-ESTEC. Ce workshop était l'occasion de rassembler en un même lieu les différentes équipes européennes. Une délégation liégeoise composée de professeurs et d'étudiants était sur place<sup>4</sup>.



Les buts de ce workshop étaient multiples. Dans un premier temps, l'ESA a présenté l'état d'avancement du lanceur Vega. Nous avons appris que la fabrication du lanceur avait pris du retard et que par conséquent notre vol était retardé de quelques mois.

Enfin, dans un second temps, plusieurs présentations techniques sur des sujets précis se sont déroulées. Ces présentations furent une mine d'informations pour les développeurs de CubeSat. Ces présentations, parfois détaillées, étaient un support technique supplémentaire. Beaucoup de sujets intéressants ont été traités. Ceux-ci furent traités par non-seulement des européens mais aussi des américains. Au niveau des nationalités, nous retrouvions : le Canada, les USA, la Suisse, l'Allemagne, l'Italie, l'Espagne, la France, le Danemark, la République Tchèque, les Pays-Bas, etc. Beaucoup de contacts ont été pris durant ce workshop. Nous avons également eu de nombreuses conversations avec plusieurs personnes de nationalité différente.

Enfin, les équipes sélectionnées pour le vol inaugural de Vega furent invitées à présenter leur état d'avancement devant les instances de l'ESA. Pour rappel, neuf CubeSats ont été sélectionnés pour être à bord de Vega. Pour OUFTI-1, François Mahy, Vincent Beukelaers et moi-même avons présenté ce *progress report* en anglais. Cette présentation d'environ trente minutes décrivait la mission, notre satellite, les sous-systèmes et le statut du projet. Ensuite, un question-réponse suivait la présentation pour que le public puisse obtenir des détails supplémentaires.

---

<sup>4</sup>Délégation liégeoise (de g. à d.) : Lionel Jacques, Nicolas Evrard, Amandine Denis, Johan Hardy, François Mahy, Jonathan Pisane et Vincent Beukelaers

La présentation a été créditée d'un bon cachet. Nous avons recueilli sur place de nombreuses réactions positives. Postérieurement au workshop, d'autres réactions ont également été recueillies. En voici une venant d'Espagne (Hector Salvador - Laboratory for Space and Microgravity Research) :

*« (...) I attended the presentation of University of Liège CubeSat. What an amazing project! Congratulations! I'm amazed by your CubeSat and the enthusiasm of the students. I'm really looking forward to talking in more details with them. Will they come to the visit? Maybe we could organize a presentation of their project for Spanish students. (...) »*

Pour conclure, ce workshop a été personnellement une expérience très enrichissante. Parler d'un sujet technique devant des chefs de projets spatiaux et qui plus est en anglais n'a pas été une tâche facile. Mais toute la présentation s'est très bien passée. Les nombreux contacts (surtout les suisses) que nous avons pris ainsi que les différentes connaissances que nous avons acquises au cours de ce workshop furent fructueuses pour tous.

## Contact radio avec Frank De Winne



L'ARISS<sup>5</sup> est un projet commandité par divers organismes et mené par les astronautes et les cosmonautes de l'ISS<sup>6</sup> ayant une licence de radioamateur. C'est notamment le cas de Frank De Winne.



Fin mai 2009, Frank De Winne a rejoint ISS. Sa mission consistait à réaliser plusieurs expériences dans l'espace pour de nombreux pays dont la Belgique. La fonction principale de Frank De Winne était le poste de commandant de bord. Ce poste implique beaucoup de responsabilités. Au yeux de l'Europe, Frank De Winne est en fait le premier européen à occuper ce poste et a fortiori le premier belge !

A l'initiative de Gaston Bertels, représentant belge de l'ARISS, l'équipe OUFTI-1 était invitée à établir un contact radio avec notre astronaute. Ce contact s'est déroulé à l'Euro Space Center de Redu à la fin du mois de février 2009. A ce moment là, Frank De Winne s'entraînait dans les installations de la NASA à Houston en vue du séjour à bord d'ISS. Ce contact radio était en fait une vraie simulation d'une conversation que nous aurions pu avoir avec lui si ce dernier était à bord d'ISS. Ce contact était par conséquent un exercice pour Franck De Winne et l'Euro Space Center de Redu.

Chaque membre du projet OUFTI-1 présent a pu poser une question. La première question posée à Frank De Winne fut la mienne : Comment appréhendez-vous votre nouvelle fonction de commandant de bord ? Frank De Winne répondit que c'était important pour l'Europe d'avoir un commandant de bord européen, et que c'était une fierté pour la Belgique. Il en attendait surtout une bonne expérience avec l'équipe qui se trouvera à bord.

Sur place, il y avait également une journaliste de la RTBF<sup>7</sup> (radio). Cette dernière a relaté notre contact sur les ondes le jour suivant [31].

<sup>5</sup>Amateur Radio on the International Space Station.

<sup>6</sup>International Space Station.

<sup>7</sup>Radio-Télévision Belge de la communauté Française.

Enfin, cette conversation très courte avec Frank De Winne a été une expérience enrichissante pour tous. Pouvoir parler avec Frank De Winne n'est pas habituel et a fortiori pendant que celui-ci s'entraîne pour sa mission. L'équipe OUFTI-1 a été très enthousiaste et chacun ressort grandi de cette expérience.

## Séjour à l'EPFL



Dans le courant de la fin du mois de mars 2009, une petite délégation liégeoise a été envoyée à Lausanne (Suisse) pour rendre visite aux membres de SwissCube. Le but de ce séjour était de répondre aux différentes questions que nous nous posions. Le but était aussi que les suisses nous répondent de vive voix. En effet, il est clair que parler avec son interlocuteur de vive voix est plus aisé que par email. Les questions étaient aussi bien techniques qu'organisationnelles.



Concernant l'EPFL, celle-ci est l'une des deux Ecoles Polytechniques fédérales en Suisse (la seconde se trouve à Zurich). L'EPFL se situe au bord du lac Léman. Elle réunit la quasi-totalité de ses membres sur un seul site, soit près de 10000 personnes. Cette école est vraiment très moderne et possède des outils de qualité.

Le satellite suisse (SwissCube) est principalement conçu à l'EPFL. Cependant, de nombreuses écoles du pays y participent aussi. Ce projet est chapeauté par Muriel Noca (EPFL). Cette dernière a travaillé plusieurs années à la NASA et est maintenant à la tête du projet satellite suisse. Les suisses développent leur CubeSat depuis 2006. En tout, 180 personnes ont travaillé sur le projet. En discutant, les suisses nous expliquent qu'ils ont eu exactement les mêmes problèmes que les nôtres : problèmes d'organisation, de management, de communication et même techniques. Cependant, une organisation solide s'est mise en place au fil du temps. Elle s'est finalement basée sur des étudiants fraîchement diplômés puis engagés par l'EPFL. Ceux-ci travaillèrent exclusivement sur le satellite. Ces étudiants sont appelés : ingénieurs systèmes. Dans la hiérarchie, Muriel Noca gère ses ingénieurs systèmes et ceux-ci recrutent des étudiants en master motivés pour faire un travail de fin d'études (très précis et détaillé) sur SwissCube.

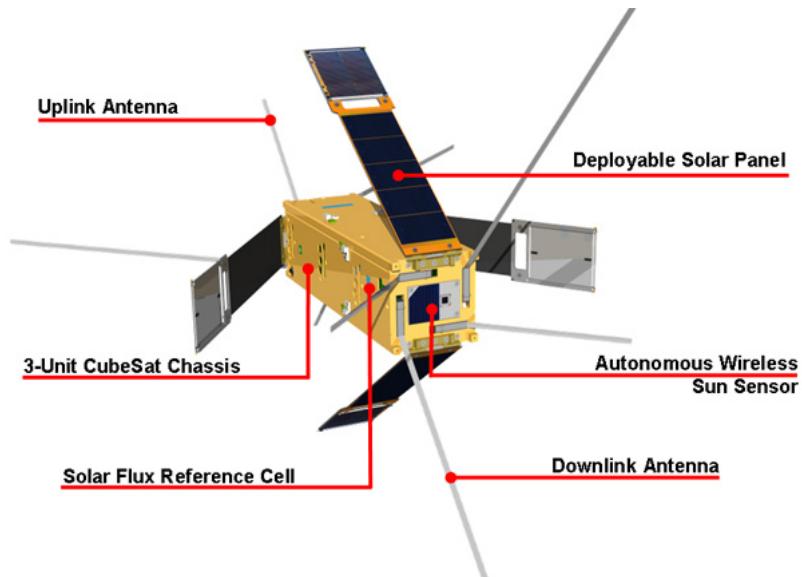
Grâce à Ted Choueiri (étudiants à l'EPFL), les questions sur l'AX.25 sans réponse ont trouvé leurs réponses. Beaucoup de problèmes ont également été résolus. Personnellement, le point clef de ce séjour était de répondre à un maximum de questions concernant le TNC (ce qui fut le cas).

Pour conclure, ce séjour a également été une expérience très instructive. Le professionnalisme et le pragmatisme des suisses sont assez impressionnantes. Enfin, je dirais simplement qu'il est important de sortir de ses frontières et voir ce qui se fait ailleurs.

## Annexe 5

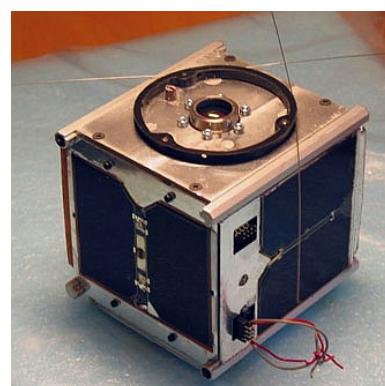
Voici quelques exemples de CubeSat ainsi qu'une courte description de leur mission.

### DELF1-C3



Delfi-C3 est le premier nanosatellite étudiant de l'université de Delft aux Pays-Bas. Delfi-C3 a la particularité d'être un CubeSat 3U. Un certain nombre de nouvelles technologies sont mises à l'essai à bord : des cellules solaires ultrafines et des capteurs solaires autonomes sans fil. Quelques caractéristiques : NORAD<sup>8</sup> # 32790, inclinaison 97,994 grad, période d'1h 37m 11s, apogée à 634 km et périgée à 615 km.

### AAU-II



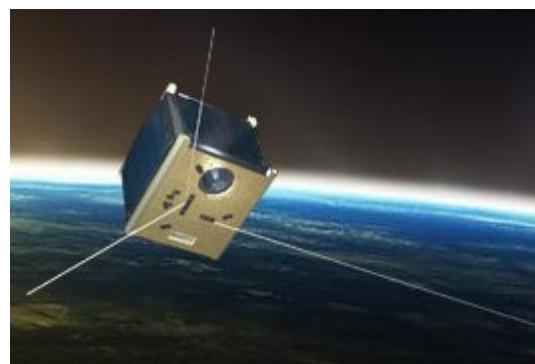
AAUSAT-II est le second CubeSat de l'université d'Aalborg au Danemark. D'une masse de 725 g, ce CubeSat expérimente un nouveau type de capteur de rayonnement au cristal. Quelques caractéristiques : NORAD # 32788, inclinaison 97,993 grad, période d'1h 37m 11s, apogée à 634 km et périgée à 615 km.

---

<sup>8</sup>North American Aerospace Defense Command.

**CP-3**

CP-3 est le troisième satellite de CalPoly. La mission de CP-3 consistait à utiliser un nouveau système d'*Attitude Determination and Control* utilisant un magnétomètre 2 axes. Quelques caractéristiques : NORAD # 31132, inclinaison 98,090 grad, période d'1h 38m 58s, apogée à 771 km et périgée à 647 km.

**SwissCube**

SwissCube est le CubeSat suisse développé à l'EPFL<sup>9</sup>. La mission de SwissCube est d'observer le phénomène d'*airglow*. Le phénomène d'*airglow* est la très faible émission de lumière visible par l'atmosphère de la Terre, empêchant ainsi une nuit d'être jamais totalement noire. Cette lumière est causée par divers phénomènes physiques se produisant dans la haute atmosphère. Pour observer l'*airglow*, une caméra sera placée à bord.

---

<sup>9</sup>Ecole Polytechnique Fédérale de Lausanne.

## Errata

