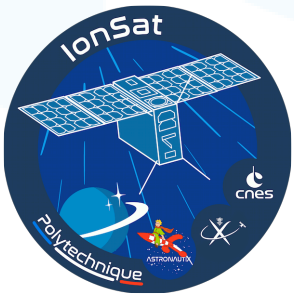
Rapport de stage

Ingénieur stagiaire : Protocole CAN et développement FPGA

Tuteur : M. Ahmed Ghouili
Tuteur pédagogique : M. Thibault Hilaire

Avril - Août 2025

Valentin Le Lièvre



Abstract

As part of my 4th-year engineering internship, I joined the Centre Spatial de l'École Polytechnique (CSEP) to contribute to the development of the IonSat nanosatellite. I was a member of the flight software and FPGA team, with responsibilities focusing on the hardware architecture of the on-board computer and the development of a CAN protocol IP.

My first task consisted in designing a CAN protocol IP, by adapting an existing core to ensure compatibility with IonSat's hardware platform. The IP was modified to support the AXI4-Lite protocol from Xilinx, and the identified issues were corrected. I then validated all functionalities through comprehensive tests, documented in a report detailing the targeted use cases. Finally, I produced a production-ready package, including user documentation, C device drivers, and integration examples.

In parallel, I contributed to the implementation of IonSat's FPGA platform, integrating various IPs covering both ground-station communication and the control of on-board equipment. I also developed and adapted automation scripts for Vivado, in order to simplify project generation and compilation across different hardware targets.

This internship allowed me to strengthen my skills in VHDL development and low-level programming, while deepening my experience with automation and software integration tools. It also confirmed my professional ambition to pursue a career in FPGA and embedded software development for the aerospace sector.

Résumé

Dans le cadre de mon stage de 4e année d'études d'ingénieur, j'ai intégré le Centre Spatial de l'École Polytechnique (CSEP) afin de contribuer au développement du nanosatellite IonSat. J'ai rejoint l'équipe en charge du logiciel de vol et du FPGA, avec pour mission de travailler sur l'architecture matérielle de l'ordinateur de bord et sur le développement d'une IP pour le protocole CAN.

Dans un premier temps, mon travail a porté sur la réalisation d'une IP pour le protocole CAN, en adaptant une IP existante afin de la rendre compatible avec la plateforme matérielle d'IonSat. L'IP a été modifiée pour assurer sa compatibilité avec le protocole AXI4-Lite de Xilinx, et les différents bugs identifiés ont été corrigés. J'ai ensuite validé l'ensemble des fonctionnalités au moyen de tests complets, documentés dans un rapport détaillant les cas d'usage ciblés. Enfin, j'ai produit un package prêt à la mise en production, comprenant la documentation d'utilisation, des device drivers en C et des exemples d'intégration.

En parallèle, j'ai participé à la mise en place de la plateforme FPGA du satellite, intégrant différentes IPs couvrant aussi bien la communication avec la station sol que le contrôle des équipements embarqués. J'ai également développé et adapté des scripts d'automatisation pour Vivado, afin de simplifier la génération et la compilation du projet sur chacune des cibles matérielles.

Ce stage m'a permis de consolider mes compétences en développement VHDL et en programmation bas niveau, tout en approfondissant l'usage des outils d'automatisation et d'intégration logicielle. Il a enfin confirmé mon projet professionnel dans le domaine du développement FPGA et logiciel embarqué appliqué à l'aérospatial.

Remerciements

Je tiens à remercier le **Centre Spatial de l'École Polytechnique (CSEP)** pour m'avoir accueilli au sein de son équipe et pour m'avoir offert l'opportunité de contribuer au projet IonSat.

J'adresse mes remerciements particuliers à mon tuteur de stage, **M. Ahmed Ghouili**, pour son encadrement et ses conseils tout au long de cette expérience.

Je souhaite également remercier l'ensemble des membres de l'équipe du CSEP pour leur accueil chaleureux et leur soutien tout au long de ce stage, ainsi que les stagiaires présents durant cette période pour les moments conviviaux partagés.

Sommaire

Abreviations

CSEP : Centre Spatial de l'École Polytechnique
CSU : Centre Spatial Universitaire
CNES : Centre National d'Études Spatiales

LEO : Low Earth Orbit
VLEO : Very Low Earth Orbit

ASIC : Application-specific integrated circuit
FPGA : Field Programmable Gate Array
IP : Intellectual Property
HDL : Hardware Description Language
VHDL : Very High Speed Integrated Circuit Hardware Description Language

CAN : Controller Area Network

1 - Introduction

Ce stage de 4 mois au Centre Spatial de l'École Polytechnique (CSEP) s'inscrit dans le cadre de ma 4e année d'école d'ingénieur à Polytech Sorbonne, où je me spécialise en systèmes embarqués et FPGA.

1.1 - Présentation de l'entreprise

1.1.1 - Le CSEP

Le CSEP (Centre Spatial de l'École Polytechnique) est une structure rattachée à l'École Polytechnique, financée par le LPP (Laboratoire de Physique des Plasmas) via la chaire `Espace - Sciences et Défis du Spatial` et avec pour rôle d'affirmer la présence de l'École Polytechnique au niveau académique et mondial dans le domaine du spatial. Ses locaux se situent sur le campus de l'École Polytechnique à Palaiseau dans le batiment Drahi-X Novation Center au coté de l'incubateur de startup, du FabLab et de la salle blanche du CSEP.

Le CSEP fait partie des vingt Centres Spatiaux Universitaires (CSU) répartis en France. Ces centres ont pour mission principale de former des étudiants aux métiers du spatial à travers des projets d'ingénierie concrets tels que le développement de satellites, de fusées expérimentales ou d'expériences embarquées à bord de la Station Spatiale Internationale (ISS).
Les CSU s'appuient généralement sur une équipe d'ingénieurs permanents qui assurent la continuité des projets, accompagnent les étudiants, supervisent les stages et prennent en charge les aspects techniques les plus complexes. Le CSEP compte actuellement cinq ingénieurs, dont un chef de projet, ainsi que des spécialistes en électronique, logiciels embarqués, télécommunications, etc.

Chaque années, en complément de cette équipe, de nombreux étudiants participent activement aux projets du CSEP, que ce soit dans le cadre de cours, de projets universitaires ou de stages, conformément à la vocation pédagogique des CSU.

Figure 1: Logo du CSEP

1.1.2 - Les activités du CSEP

Le CSEP a été créé en 2012 pour encadrer le premier projet de nanosatellite de l'École Polytechnique, X-CubeSat, lancé en 2017 apres cinq ans de développement, et qui etait alors le premier satellite étudiant français opérationnel en orbite.

Depuis, le CSEP pilote plusieurs projets et initiatives, parmi lesquels :

- Le développement du nanosatellite IonSat
- L'encadrement des Projets Scientifiques Communs (PSC) menés par les étudiants de l'École Polytechnique, qui participent à des projets en cours au CSEP ou en proposent de nouveaux ;
- La participation annuelle au programme C'Space, une campagne de lancement de fusées expérimentales étudiantes organisée en partenariat avec le CNES, à travers l'association étudiante AstronautiX.

Ces activités permettent d'accueillir chaque année plus de 80 étudiants, leur offrant une formation concrète et une porte d'entrée dans le domaine spatial.

De plus, au-delà de sa vocation pédagogique, le CSEP s'inscrit dans un écosystème spatial en pleine transformation, marqué par l'essor du New Space et par l'importance croissante des nanosatellites dans la recherche scientifique et les applications commerciales. En ce sens, ses objectifs stratégiques ne se limitent pas à la formation : il s'agit également de renforcer la visibilité scientifique de l'École Polytechnique, de développer des collaborations avec les acteurs institutionnels comme le CNES, et de préparer les étudiants à intégrer le secteur spatial, tant académique qu'industriel.

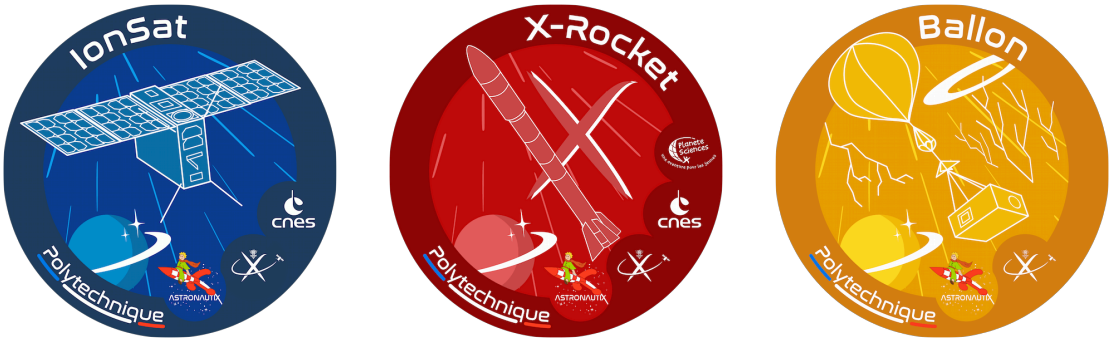


Figure 2: Stickers des activités du CSEP

1.2 - Sujet et contexte du stage

1.2.1 - Les nanosatellites - CubeSats

Un CubeSat est un petit satellite répondant à un format standardisé, basé sur un cube de 10 cm de côté pesant environ 1 kg. Plusieurs unités (ou « U ») peuvent être assemblées pour former des satellites plus grands : par exemple, un satellite 3U mesurera 30 x 10 x 10 cm. Chaque unité supplémentaire permet d'embarquer davantage de charges utiles, des composants plus volumineux et, par conséquent, d'augmenter les capacités du satellite. Toutefois, cette augmentation de taille implique également un coût de lancement plus élevé.

L'intérêt principal du format CubeSat réside dans sa capacité à démocratiser l'accès à l'espace. En effet, il permet à des universités, des laboratoires et des petites entreprises de concevoir, développer et lancer leurs propres satellites à un coût réduit, en s'appuyant sur des composants commerciaux standards disponibles sur le marché. Le coût de lancement d'un CubeSat reste généralement bien inférieur à celui des satellites conventionnels, notamment parce qu'ils sont conçus pour être lancés en groupes, mutualisant ainsi les coûts logistiques.

En général, les satellites universitaires embarquent plusieurs missions scientifiques - appelées charges utiles - qui donnent tout leur intérêt au projet. Ces missions sont souvent menées en partenariat avec d'autres universités, entreprises ou laboratoires, ce qui permet de renforcer la portée scientifique et pédagogique des CubeSats tout en offrant aux partenaires une opportunité d'envoyer leurs expériences dans l'espace de manière plus accessible et économique.

La durée de vie d'un CubeSat peut varier de quelques mois à plusieurs années, selon sa conception, son orbite et la nature de sa mission. Ces satellites sont souvent pensés pour être opérationnels sur une période limitée, au terme de laquelle ils entrent dans une phase de désorbitation contrôlée.

Les CubeSats peuvent être lancés sur différentes orbites, selon les objectifs du projet. Cependant, ils sont majoritairement déployés en orbite basse terrestre (LEO), entre 200 et 2 000 km d'altitude. Cette configuration permet non seulement de répondre à de nombreux besoins scientifiques et techniques, mais aussi de limiter la durée de vie orbitale du satellite après la fin de la mission, contribuant ainsi à la réduction des débris spatiaux.

Figure 3: X-cubesat, un satellite CubeSat 2U produit au CSEP en lancé en 2017

1.2.2 - Context du stage

1.2.2.1 - Le projet IonSat

À la suite du succès de son premier satellite, le CSEP a lancé en 2017 un nouveau projet de nanosatellite : IonSat. Il s'agit d'un CubeSat de format 6U, mesurant 30 x 20 x 10 cm, destiné à être placé en orbite terrestre très basse (VLEO), à environ 300 km d'altitude.

IonSat embarquera plusieurs charges utiles, dont la principale est un propulseur électrique. En règle générale, les CubeSats ne disposent pas de moyen de propulsion, mais uniquement de systèmes d'orientation. Toutefois, en VLEO, la traînée atmosphérique est bien plus importante qu'à plus haute altitude, ce qui entraîne une perte progressive d'altitude. Afin de prolonger la durée de vie du satellite, le propulseur permettra d'effectuer des manœuvres de correction d'orbite, évitant ainsi une désorbitation prématurée.

Parmi les autres charges utiles, on compte :
- un capteur d'oxygène atomique, nommé Resistack et fourni par l'Onera
- une caméra embarquée, la piCAM, pour prendre des images de la terre
- des gyroscopes expérimentaux, fournis par le CNES dans le but d'être testés dans l'espace
- une antenne LoRa
- une antenne radioamateur UHF / VHF
- un capteur mesurant l'effet de l'iode sur les panneaux solaires, fourni par le Von Karman Institute

Le lancement d'IonSat est actuellement prévu pour courant 2026, mais cette date reste à confirmer en fonction de l'avancement du projet, et des disponibilités des lanceurs.

1.2.2.2 - L'équipe du projet IonSat

L'équipe permanente en charge du projet IonSat est composée de cinq ingénieurs :

- Directeur du CSEP : Luca Bucciantini
- Chef de projet : Borhane Bendaci
- Ingénieur électronique & logiciel embarqué : Ahmed Ghoulli
- Ingénieur AIT (Assembly, Integration and Testing) : Nicolas Lequette
- Ingénieur télécommunications : Tony Colin

En complément, durant mon stage, quatre autres stagiaires travaillaient aux côtés des ingénieurs permanents, notamment sur le logiciel embarqué, les campagnes de tests, ainsi que sur la mise en place de la station sol. De plus, au cours de six dernières années de développement, de nombreux autres stagiaires et étudiants ont contribué au projet au travers de leurs stages et projets.

1.2.2.3 - Présentation du Nanolab academy

Dans le cadre de son nouveau projet de nanosatellite, le CSEP participe au programme Nanolab Academy piloté par le CNES. Ce programme a pour objectif d'accompagner les Centres Spatiaux Universitaires (CSU) dans la conception et la réalisation de leurs satellites. Le CNES y joue un rôle de soutien technique en fournissant à la fois des bases technologiques, des documents de référence et une plateforme de partage de connaissances destinée à faciliter le développement des projets.

Le CNES a notamment développé, dans le cadre de ce programme, le nanosatellite EyeSat, lancé en 2019 et resté opérationnel pendant quatre ans et finalise actuellement un nouveau projet, AeroSat, dont le lancement est prévu pour début 2026.

Les composants matériels et logiciels conçus pour EyeSat et AeroSat ont été mis à disposition des CSU partenaires. Ces éléments servent de base technique commune sur laquelle chaque CSU peut s'appuyer pour intégrer ses propres sous-systèmes et développer des fonctionnalités spécifiques.

1.2.3 - Sujets et objectifs du stage

Dans le cadre du projet IonSat, mon stage s'est inscrit dans le développement des systèmes électroniques embarqués du satellite. Plus précisément, en tant que stagiaire en électronique numérique spécialisé en FPGA, j'ai été chargé de deux missions principales.

La première mission portait sur la conception et l'implémentation d'un contrôleur CAN (Controller Area Network) sous forme d'IP matérielle dédiée, entièrement développée en VHDL. Ce composant a pour rôle de gérer les communications entre différents sous-systèmes du satellite via le bus CAN, un protocole robuste couramment utilisé dans les environnements embarqués pour ses performances en temps réel et sa tolérance aux erreurs.

La seconde mission consistait à intégrer plusieurs IPs sur la plateforme FPGA destinée à la mission. Ce travail comprenait la compréhension des IPs fournies par le CNES et l'adaptation de ces composants pour les adapter aux spécificités du projet IonSat. L'objectif était de garantir que toutes les IPs fonctionnent de manière cohérente et efficace, en assurant la communication entre elles et avec les autres sous-systèmes du satellite.

2 - Architecture d'IonSat

2.1 - Composants matériels

En plus des composantes des missions scientifiques (charges utiles), le satellite embarque de nombreux autres composants nécessaires à son bon fonctionnement dans l'espace et tout au long de la mission. Ces sous-systèmes incluent principalement les équipements de télécommunication, de contrôle d'attitude et d'orbite, de gestion de puissance, ainsi que l'ordinateur de bord. Ces l'ensemble des ces composants, qui sont councu au CSEP, acheté comme composants standard ou fourni par le CNES, qui doivent ensuite etre intégrés, controllé par le logiciel de bord et testé avant le lancement.

2.1.1 - Télécommunications

Pour communiquer avec la station de contrôle au sol, le satellite utilise une bande de fréquences radio située entre 2 et 4 GHz, appelée **bande S**. Cette bande est couramment employée pour les communications satellitaires, notamment grâce à un débit de données relativement élevé.

À cette fin, IonSat est équipé :
- d'une antenne en bande S pour l'émission et la réception,
- ainsi que d'un transceiver externe, fourni par le CNES, entièrement dédié à cette tâche.

Dans le vocabulaire spatial, le satellite transmet des données de **télémétrie** (TM) et reçoit des **télécommandes** (TC). Ces échanges suivent le format normalisé par le CCSDS (**Consultative Committee for Space Data Systems**), garantissant l'interopérabilité et la fiabilité des communications espace sol.

IonSat embarque aussi un transceiver en bande UHF/VHF, destiné uniquement à l'envoie de données TM et aux échanges avec les radioamateurs. Ce canal de communication est secondaire, car bien plus lent et, dans le cas d'IonSat, non sécurisé.

2.1.2 - Orientation et déplacement

L'orientation du satellite est essentielle pour plusieurs fonctions critiques : assurer une liaison stable en TM/TC avec la station sol, orienter les panneaux solaires vers le Soleil afin d'optimiser la production d'énergie, et pointer la camera vers la Terre pour l'acquisition d'images.

Pour cela, IonSat dispose de nombreux capteurs permettant de déterminer son attitude :
- un accéléromètre pour mesurer les accélérations,
- un magnétomètre pour caractériser le champ magnétique terrestre,
- un gyroscope pour suivre la rotation du satellite,
- des capteurs solaires pour identifier la direction du Soleil.

L'ensemble de ces informations est exploité par le **système de contrôle d'attitude** (ADCS), capable de calculer l'orientation du satellite et d'agir en conséquence. Plusieurs actionneurs sont intégrés : des roues de réaction pour annuler ou ajuster les vitesses de rotation, des magnétorquer pour exploiter le champ magnétique terrestre et réorienter le satellite, ainsi qu'un petit propulseur pour modifier l'orbite et effectuer des corrections de trajectoire.

2.1.3 - Gestion de puissance

Le satellite est alimenté par deux panneaux solaires déployables (repliés lors du lancement afin de réduire l'encombrement). Ces panneaux fournissent l'énergie nécessaire au fonctionnement de l'ensemble des systèmes et rechargent les batteries.

La gestion de la distribution électrique est assurée par deux cartes électroniques : une carte de gestion de puissance, responsable de l'alimentation des sous-systèmes, et une carte de passivation, permettant de déconnecter les batteries lors de la mise hors service du satellite et lors de son désorbitage en fin de vie.

2.1.4 - Ordinateur de bord

Le pilotage central de l'ensemble des sous-systèmes est confié à l'ordinateur de bord. Celui-ci est construit autour d'une puce FPGA SoC **Xilinx Zynq-7030**, intégrant deux cœurs ARM. En pratique, l'ordinateur de bord assure plusieurs fonctions critiques. Il orchestre la communication entre les différents sous-systèmes et supervise l'ensemble des séquences de mission, c'est également lui qui gère les communications de télécommande et de télémétrie. Enfin, il surveille en permanence l'état de santé du satellite en collectant des mesures issues des capteurs et en prenant, si nécessaire, des décisions correctives automatiques.

Grâce à cette architecture, l'ordinateur de bord constitue le centre d'IonSat, garantissant la cohérence et la fiabilité de l'ensemble des opérations, depuis le lancement jusqu'à la fin de vie du satellite. De plus, dû aux conditions difficile de l'espace, la carte est renforcée contre les radiations afin de garantir un fonctionnement fiable dans l'environnement spatial tout au long de la mission.

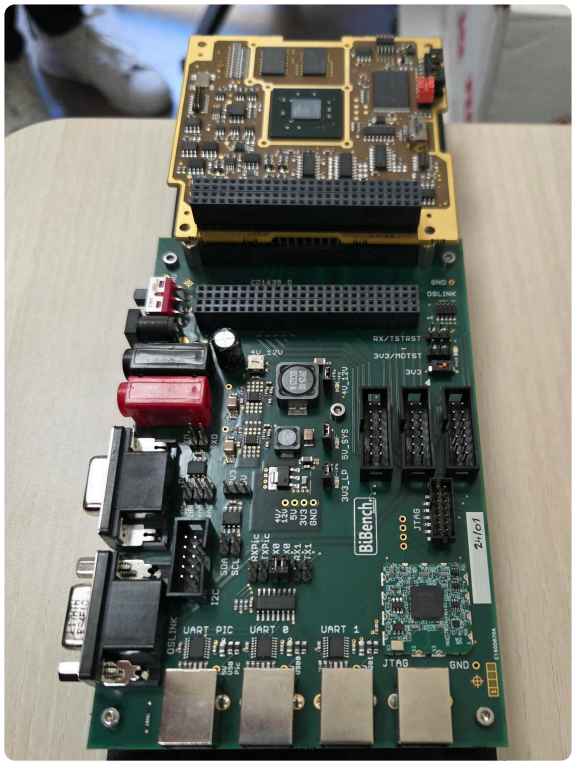


Figure 4: L'ordinateur de bord "Ninano", sur la carte d'extension

2.1.5 - Interconnexion des composants

Tous les sous-systèmes sont reliés à l'ordinateur de bord, qui assure leur contrôle et la coordination des échanges. Le choix du protocole de communication dépend de plusieurs facteurs tels que le volume de données à transmettre, la criticité des échanges, la vitesse de transfert requise ou encore la distance entre les différentes cartes électroniques.

Ainsi, les composants jugés critiques, comme le propulseur ou le contrôleur d'attitude, s'appuient sur le bus **CAN**, réputé pour sa robustesse et sa tolérance aux erreurs, ce qui en fait un standard particulièrement adapté aux environnements contraints. Les charges utiles, quant à elles, exploitent d'autres protocoles plus légers, tels que **I²C**, **SPI** ou **OneWire**, permettant une intégration plus simple mais étant moins robustes.

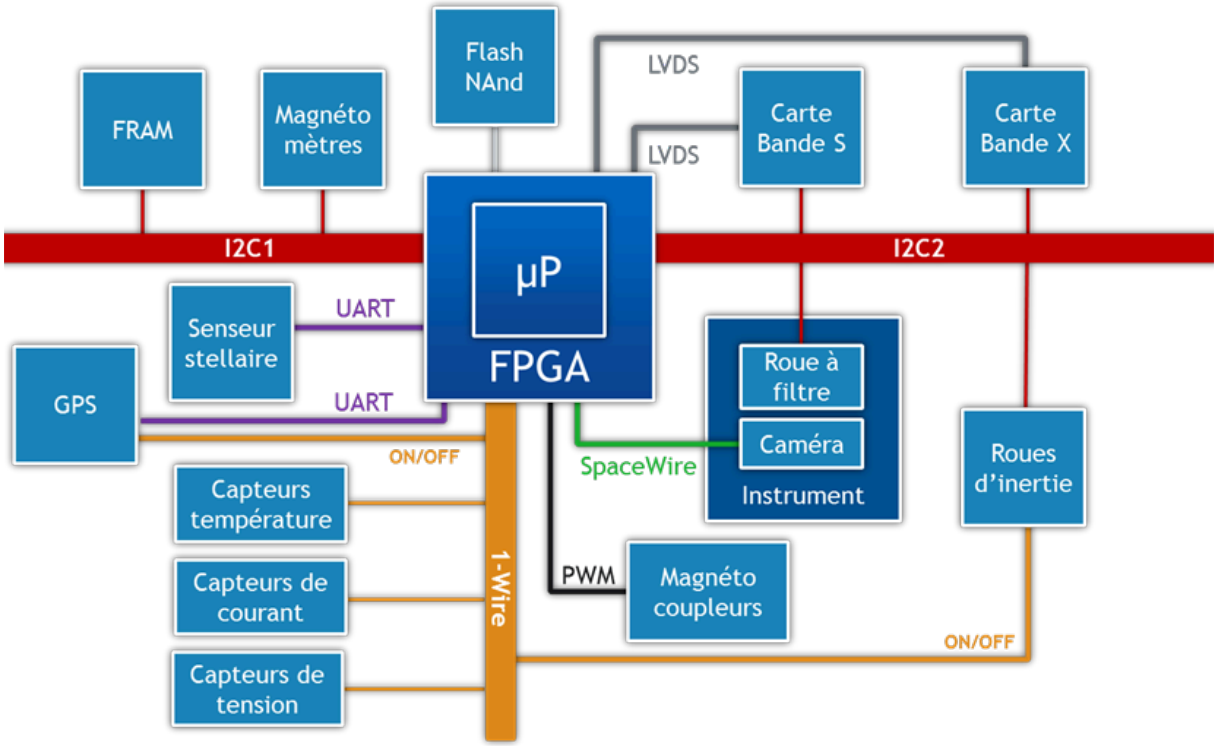


Figure 5: Composants externe et leur protocole de communication avec l'ordinateur de bord

2.2 - Logiciel embarqué

2.2.1 - FPGA

Très répandus dans le domaine spatial, les FPGA jouent un rôle essentiel en soulageant le processeur central des tâches les plus exigeantes. Leur architecture reconfigurable permet de créer de composants spécifiques à certaines tâches et de traiter efficacement des opérations qui seraient trop coûteuses en ressources CPU ou qui nécessitent une précision temporelle difficile à atteindre avec un processeur classique.

Dans le cadre d'IonSat, le FPGA est utilisé pour implémenter des contrôleurs dédiés à certains protocoles de communication non pris en charge nativement par le SoC. Il intervient également dans la gestion et le transfert des données entre sous-systèmes, ainsi que dans le codage et le décodage des trames de télémétrie et de télécommande. Ces fonctions sont critiques, car elles conditionnent à la fois la fiabilité des échanges avec la station sol et la bonne coordination interne du satellite.

Pour accomplir ces missions, le FPGA est configuré à l'aide de plusieurs blocs matériels décrits en VHDL, appelés **IPs**, qui sont intégrés dans la logique programmable. Ces IPs constituent des briques modulaires permettant d'optimiser le traitement matériel, tout en offrant une grande souplesse de reconfiguration en cas d'évolution des besoins de la mission. Cette flexibilité rend le FPGA bien plus adaptable qu'un circuit ASIC figé et plus facilement réutilisable entre plusieurs missions.

2.2.2 - Logiciel de bord

Enfin, le logiciel de bord constitue l'élément central qui coordonne l'ensemble des sous-systèmes et assure le déroulement des différentes missions du satellite. Il collecte en continu les données des capteurs, organise leur traitement puis les transmet à la station de contrôle via le système de télécommunication. Au-delà de ce rôle de supervision, il prend également en charge l'exécution des procédures automatiques essentielles, telles que les corrections d'orbite pour ajuster l'altitude, les manœuvres d'orientation vers la Terre ou vers le Soleil, ainsi que la gestion des modes de consommation d'énergie en fonction du niveau de charge des batteries.

Le logiciel embarqué est une composante à la fois cruciale et particulièrement complexe : la moindre défaillance pourrait compromettre la mission dans son ensemble. Pour limiter les risques, son architecture est conçue de manière modulaire. Chaque fonctionnalité est isolée dans une partition logicielle distincte, de sorte qu'en cas d'erreur, seule la partie concernée peut être redémarrée sans interrompre les autres services. Cette approche renforce la robustesse du système et garantit une continuité de fonctionnement, condition indispensable au succès d'une mission spatiale de longue durée.

3 - Réalisation d'une IP CAN

3.1 - Introduction

Pour IonSat, le bus CAN est une composante essentielle puisqu'il relie l'ordinateur de bord à plusieurs systèmes critiques, notamment le propulseur et le système de contrôle d'attitude. Ces sous-systèmes sont indispensables à la survie du satellite : une défaillance de communication pourrait compromettre l'ensemble de la mission. Dans un environnement spatial marqué par les perturbations électromagnétiques et les radiations, la fiabilité de la communication est donc primordiale.

L'ordinateur de bord d'IonSat intègre déjà un transceiver CAN pour générer les signaux électriques sur le bus. En revanche, il faut également un contrôleur CAN pour gérer le protocole. Celui-ci peut être implémenté de différentes manières : en utilisant une puce externe dédiée, via un contrôleur intégré au processeur, ou sous forme d'IP sur FPGA. Cette dernière approche, retenue pour IonSat, offre à la fois proximité avec le processeur, protection accrue contre les radiations et flexibilité dans l'implémentation.

La première mission de mon stage a donc consisté à développer un IP CAN pour le FPGA Xilinx de l'ordinateur de bord.

3.2 - Le CAN

Le CAN (**Controller Area Network**), développé par Bosch, combine à la fois un bus de communication et un protocole, couvrant ainsi les deux couches basses du modèle OSI : la couche physique et la couche liaison de données. La norme de référence est l'ISO 11898, définissant la version 2 du protocole.

Très répandu dans l'automobile et le spatial, le CAN est apprécié pour sa robustesse, sa capacité à fonctionner sur de longues distances (jusqu'à 1 km, bien au-delà de l'I²C ou du SPI), et sa gestion avancée des erreurs. Ces qualités en font un protocole particulièrement adapté aux environnements contraints comme le spatial. Parmi ses caractéristiques principales, le CAN propose une communication multi-maître asynchrone, une tolérance élevée aux perturbations sur la ligne de transmission avec une détection avancée des erreurs et un débit pouvant atteindre 1 Mbit/s (débit brut, incluant les bits de trame).

3.2.1 - Le bus CAN

La couche physique est définie par la norme ISO 11898-2 pour le CAN haute vitesse. Le bus est constitué d'une paire différentielle reliant l'ensemble des nœuds, chacun composé d'un contrôleur CAN et d'un transceiver. L'utilisation de deux fils (CAN H et CAN L) permet de transmettre deux niveaux logiques (dominant ou récessif) en réduisant considérablement l'impact des interférences électromagnétiques.

La paire différentielle, CAN H et CAN L, nomme deux niveaux de tension pour transmettre soit 0 soit 1 : le niveau dominant (0) et le niveau recessif (1). Au niveau recessif les tension H et L sont a 2.5V et pour le niveaux dominant, CAN H vaut 3.5V et CAN L 1.5V. Cette utilisation de la différence de potentiel permet de réduire les interférences électromagnétiques. En effet, si un bruit électromagné-

tique est capté par les deux fils, il sera capté de la même manière et donc ne changera pas la différence de potentiel. Cette propriété permet au bus CAN d'être très robuste.

Par défaut le bus est a un niveau recessif, jusqu'a ce qu'un des noeuds force un niveau dominant. Le bus etant de type "Wire AND" le niveau dominant prevau au niveau recessif et si un noeuds force le niveau dominant (quel que soit le niveau des autres noeuds) le niveau du bus sera dominan, ceci constitue la base du mécanisme d'arbitrage;

Enfin le bus est toujours terminé par une resistance de 120 Ohms de chaque coté pour evite les reflextions et s'assusrer de l'impendance du bus.

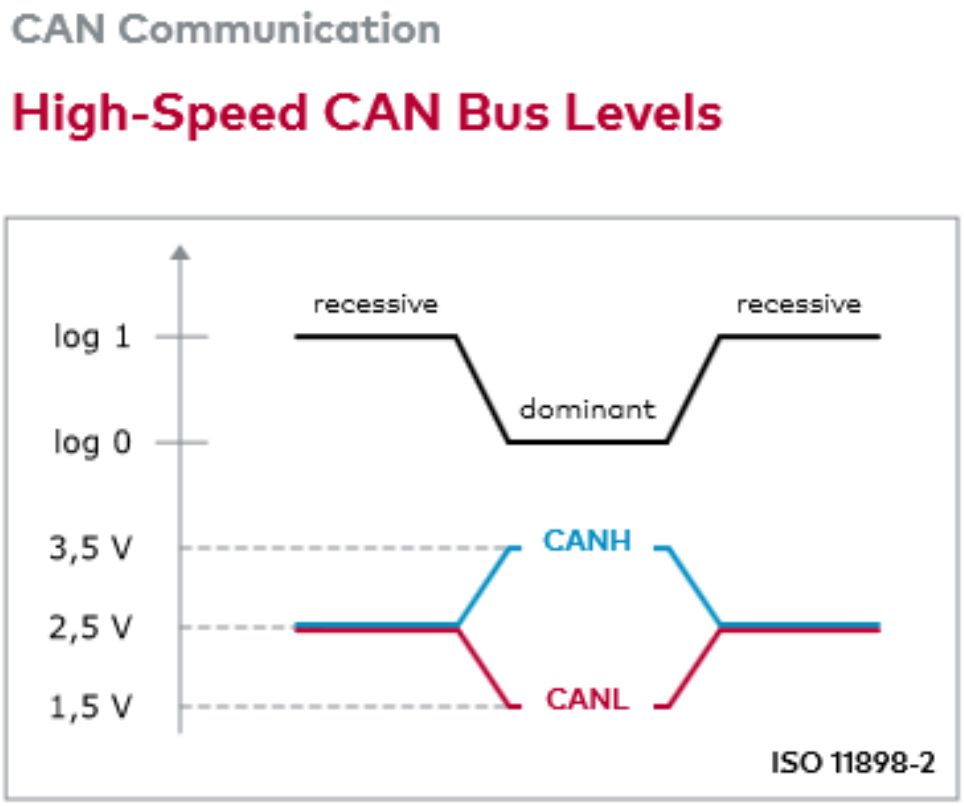


Figure 6: Composants externe et leur protocole de communication avec l'ordinateur de bord

3.2.2 - Le protocole CAN

Le protocole CAN organise les échanges sur le bus sans utiliser d'horloge commune ni de signal de contrôle. Il repose sur un mécanisme d'arbitrage permettant à plusieurs nœuds de partager le bus de manière déterministe, et sur une gestion avancée des erreurs garantissant la fiabilité des transmissions.

Trois types de trames peuvent être transmises : trames de données, trames de requête et trames d'erreur. Les trames de données, les plus courantes, transportent entre 1 et 8 octets associés à un identifiant. Cet identifiant n'est pas une adresse mais un champ de priorité : plus il est faible, plus la trame est prioritaire lors de l'arbitrage.

L'arbitrage se déroule au niveau de l'identifiant, bit par bit : si un nœud émet un niveau récessif mais lit un niveau dominant sur le bus, il interrompt sa transmission et devient récepteur. Ce mécanisme garantit qu'un seul nœud conserve le bus et évite les collisions.

De nombreux mécanismes supplémentaires renforcent la robustesse du protocole, parmi lesquels l'insertion automatique de bits de synchronisation (**bit stuffing**), le calcul d'un CRC pour vérifier l'intégrité, et l'acquittement (ACK) obligatoire par au moins un récepteur. En cas d'anomalie, une trame d'erreur est générée et des compteurs internes (TEC et REC) ajustent l'état du nœud (actif, passif ou **bus-off**).

3.2.3 - Trame de données

Les trames de données servent à transférer de l'information d'un nœud vers un ou plusieurs autres nœuds. Chaque trame contient entre 1 et 8 octets de données, associés à un identifiant. Contrairement à une adresse classique, cet identifiant n'est pas lié à un destinataire unique : il permet de définir la nature ou la priorité du message, et donc de diffuser la même information à plusieurs récepteurs en parallèle.

Une trame de données est structurée de la manière suivante :

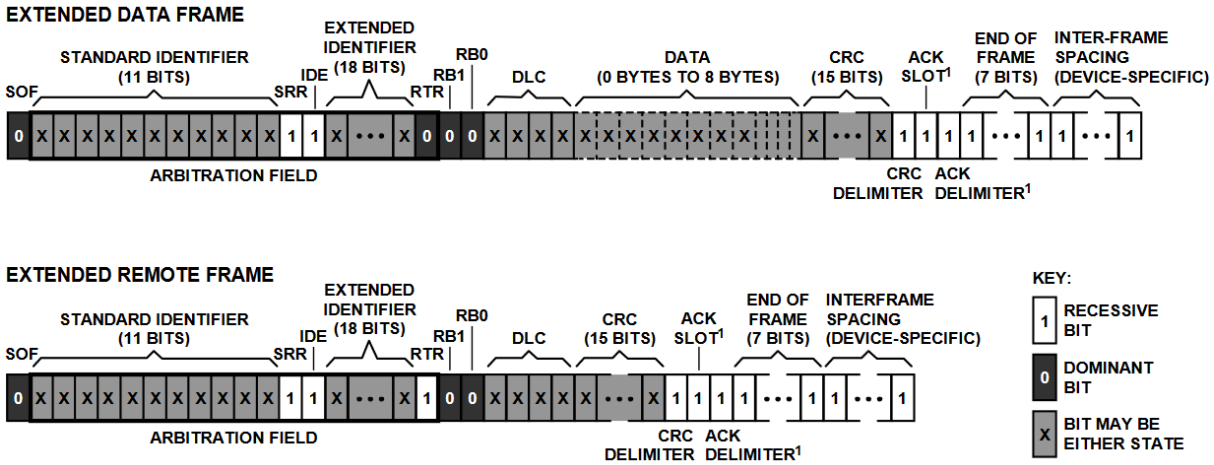


Figure 7: Structure des trames CAN

Détail des champs :

- **SOF (Start of Frame)** : premier bit dominant signalant le début de la trame.
- **Identifiant** : 11 bits définissant la donnée ou la priorité du message.
- **SRR (Substitute Remote Request)** : champ réservé, à niveau récessif (uniquement dans les trames étendues).
- **IDE (Identifier Extension)** : indique si l'identifiant est en format standard (11 bits) ou étendu (29 bits).
- **Identifiant étendu** : 18 bits supplémentaires pour atteindre 29 bits en mode étendu.
- **RTR (Remote Transmission Request)** : précise s'il s'agit d'une trame de données (0) ou d'une trame de requête (1).
- **R1, R0** : bits réservés pour de futures versions du protocole (fixés à 1).
- **DLC (Data Length Code)** : indique le nombre d'octets transportés (0 à 8).

- **Data** : les données utiles, de 0 à 8 octets.
- **CRC (Cyclic Redundancy Check)** : code de redondance pour vérifier l'intégrité de la trame.
- **ACK (Acknowledgment)** : champ d'acquittement confirmant la bonne réception par au moins un récepteur.
- **EOF (End of Frame)** : séquence de bits récessifs marquant la fin de la trame.

Deux formats existent : le mode standard (identifiant de 11 bits) et le mode étendu (identifiant de 29 bits). L'utilisation du format est signalée par le bit IDE. Le format étendu permet d'identifier un plus grand nombre de messages, mais au prix d'une trame légèrement plus longue et donc un débit utile réduit. C'est ce format qui sera utilisé pour IonSat.

3.2.4 - Trame de requête

Une trame de requête est similaire à une trame de données, à ceci près que le champ **Data** reste vide et que le bit RTR est positionné à 1. Le champ DLC peut néanmoins être utilisé pour indiquer la taille des données attendues en réponse. Cette trame peut être envoyée par n'importe quel nœud pour demander une donnée. Cette trame n'implique que la réponse à cette requête sera prioritaire, le prochain message sur le bus pourra être n'importe quel message.

3.2.5 - Arbitrage

Le bus CAN étant multi-maître, plusieurs nœuds peuvent tenter d'émettre simultanément. Pour éviter les collisions, le protocole intègre un mécanisme d'arbitrage.

Lorsque le bus est libre (au niveau récessif), un nœud peut initier une transmission en envoyant un SOF (bit dominant). Tous les autres nœuds passent alors en réception. Si plusieurs nœuds déclenchent un SOF au même instant, l'arbitrage s'effectue sur l'identifiant. Le principe repose sur la lecture simultanée du bus par chaque émetteur : après avoir transmis un bit, un nœud doit vérifier que le bus reflète bien ce qu'il a envoyé. Si un nœud émet un bit récessif mais observe un bit dominant (qui prévaut toujours), il abandonne immédiatement la transmission et devient récepteur.

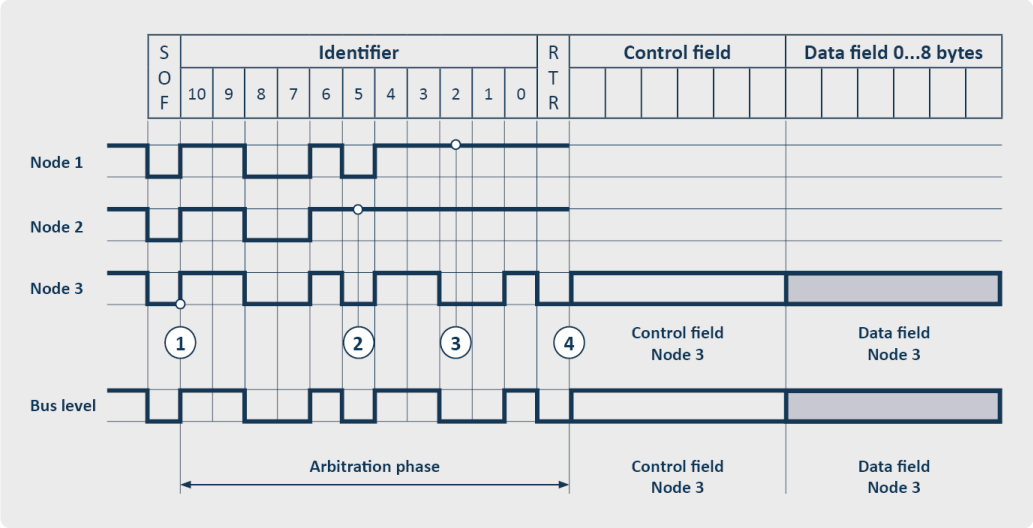


Figure 8: Arbitrage sur le bus CAN

L'arbitrage se poursuit tant que l'identifiant est en cours de transmission. Ainsi, l'identifiant le plus faible (le plus proche de 0) est prioritaire. À la fin de cette phase, le nœud qui reste en émission est assuré d'être le seul maître du bus et peut poursuivre la transmission de sa trame sans risque de collision.

3.2.6 - Synchronisation des horloges

Le CAN utilise une communication asynchrone, sans horloge commune. Chaque nœud dispose de sa propre horloge, qui peut légèrement différer des autres. Pour garantir une réception correcte des données, une synchronisation est nécessaire. Le CAN intègre un mécanisme de synchronisation basé sur la détection des fronts de bits. Chaque nœud divise le temps en segments, définis par son propre oscillateur. Lors de la réception, les nœuds ajustent leur timing en fonction des transitions détectées sur le bus. Cela permet de compenser les dérives d'horloge et d'assurer une lecture fiable des bits.

La vitesse de communication sur le bus est de 1 Mbit/s soit un bit toutes les 1 us. Cette durée est divisee en plusieurs segments pour permettre la synchronisation. Le premier segment est le segment de synchronisation (Sync Seg) qui dure 1 unité de temps. C'est dans ce segment que doivent se trouver les fronts. Ensuite viennent les segments de propagation (Prop Seg) et de phase (Phase Seg1 et Phase Seg2) qui permettent d'ajuster la synchronisation en fonction des retards de propagation et des variations d'horloge. La somme de ces segments doit être égale à la durée d'un bit (1 µs).

Lorsqu'un front est detecté en dehors du segment de synchronisation, le nœud ajuste son horloge en ajoutant ou en supprimant une unité de temps dans les segments de phase. Cela permet de réaligner la lecture des bits avec les transitions réelles sur le bus. Cette operation de synchronisation est effectuee en permanance pour compenser les dérives d'horloge et assurer la bonne lecture des données du bus.

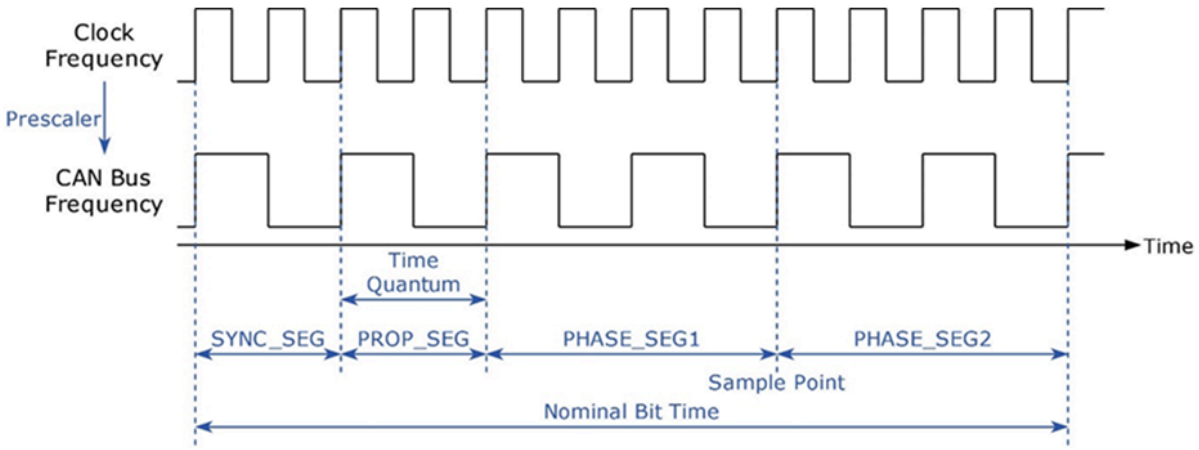


Figure 9: Synchronisation des horloges sur le bus CAN

3.2.7 - Gestion des erreurs

Le CAN intègre plusieurs mécanismes pour détecter et gérer les erreurs, garantissant ainsi la fiabilité des communications. Chaque nœud surveille en permanence le bus et utilise différents tests pour identifier les anomalies. Parmi les mécanismes de détection d'erreurs, on trouve :

- **Erreur de bit** : si un nœud émet un bit mais lit un niveau différent sur le bus, une erreur de bit est détectée.
- **Erreur de CRC** : le champ CRC permet de vérifier l'intégrité de la trame. Si le calcul du CRC ne correspond pas, une erreur de CRC est signalée.
- **Erreur de format** : si la structure de la trame ne respecte pas les spécifications (par exemple, un champ réservé mal positionné), une erreur de format est générée.
- **Erreur d'acquittement** : si aucun récepteur ne confirme la réception correcte de la trame (champ ACK), une erreur d'acquittement est détectée.
- **Erreur de bit stuffing** : si plus de cinq bits consécutifs identiques sont détectés, une erreur de bit stuffing est signalée.

Lorsqu'une erreur est détectée par un noeud celui-ci commence la transmission d'une trame d'erreur. Cette trame sera recu par tout les autres noeuds qui la detecteront comme une erreur de bit et genereront a leur tour une trame d'erreur. La trame d'erreur est composée de deux champs : le champ d'erreur (6 bits dominants) et le champ de délimitation (8 bits récessifs). Le champ d'erreur signale la présence d'une erreur, tandis que le champ de délimitation marque la fin de la trame d'erreur.

Chaque nœud maintient deux compteurs d'erreurs : le TEC (Transmit Error Counter) et le REC (Receive Error Counter). Ces compteurs sont incrémentés ou décrémentés en fonction des erreurs détectées ou des transmissions réussies. En fonction de la valeur de ces compteurs, un nœud peut se trouver dans l'un des trois états suivants :

- **État actif** : le nœud peut émettre et recevoir des trames normalement. (TEC < 128 et REC < 128)
- **État passif** : le nœud peut recevoir des trames mais ne peut plus émettre de trames d'erreur. (TEC >= 128 ou REC >= 128)
- **État bus-off** : le nœud est déconnecté du bus et ne peut plus émettre ni recevoir de trames. (TEC >= 256)

Lorsqu'un nœud entre en état bus-off, il doit attendre un certain temps (128 occurrences de 11 bits récessifs) avant de pouvoir tenter de se reconnecter au bus. Cette période permet de stabiliser le bus et d'éviter une surcharge due à des nœuds défaillants.

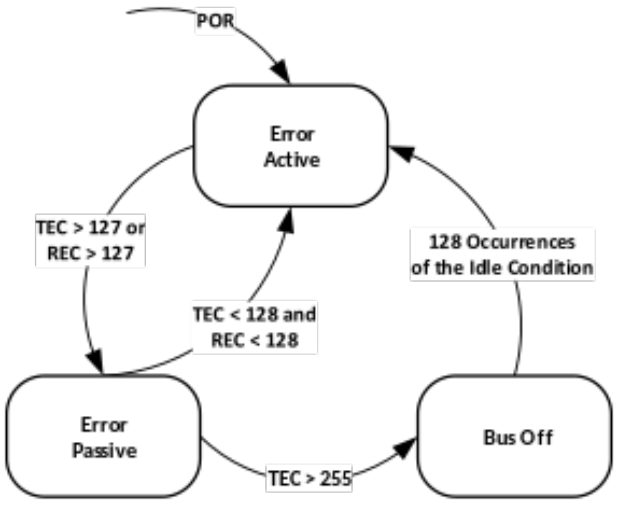


Figure 10: Modes d'erreur du CAN

3.3 - Canakri : une version open-source

3.3.1 - Description général

Afin de réduire le temps de développement et de validation, il a été décidé de partir d'une IP CAN open source existante et de l'adapter aux besoins de la plateforme IonSat. L'IP choisie, issue d'un projet universitaire, est complète et fonctionnelle. Elle implémente les modes de trames standard et étendues conformément à la norme ISO 11898-1.

Les sources, disponibles en VHDL et en Verilog, étaient initialement prévues pour les FPGA Altera (Intel) et utilisaient le bus Avalon. L'IP ne disposait cependant d'aucun tests. Malgré cela, elle présentait l'avantage d'être autonome, sans nécessiter de composants externes, et donc facilement intégrable dans un FPGA. Cette IP a donc servi de base de développement pour créer une version adaptée aux FPGA Xilinx, avec une interface AXI.

3.3.2 - Fonctionnement

L'IP **Canakari** est structurée en plusieurs blocs correspondant aux grandes fonctions décrites dans le protocole CAN : le **MAC** (Media Access Control), le **LLC** (Logical Link Control), le **Fault Confinement**, etc. Une fois instancié il est possible pour le processeur de communiquer avec l'IP via une interface bus Avalon et de configurer les registres internes pour initialiser le contrôleur, envoyer des trames ou lire les trames reçues. Cette IP gère également les interruptions pour certains événements et fournit des signaux de débogage.

3.3.2.1 - Interface

L'IP utilise le bus Avalon pour communiquer avec le processeur, ainsi que plusieurs signaux dédiés aux interruptions et au débogage.

Horloge : La fréquence d'horloge dépend du débit souhaité et de la valeur du **prescaler**. Par exemple, pour atteindre le débit maximal de 1 Mbit/s, l'IP nécessite une horloge d'entrée de 40 MHz avec un prescaler réglé à 4.

Interruptions : Trois interruptions sont disponibles. Elles sont signalées par des lignes top-level actives à l'état haut pendant un cycle d'horloge. Un registre permet de lire et de configurer les **flags** associés. Les interruptions sont désactivées par défaut.

Bus CAN : L'IP est reliée au transceiver CAN via les signaux numériques RX et TX.

Débogage : L'IP propose également des signaux de débogage, dont un signal contenant le status de la machine à états, un signal indiquant les ticks du prescaler.

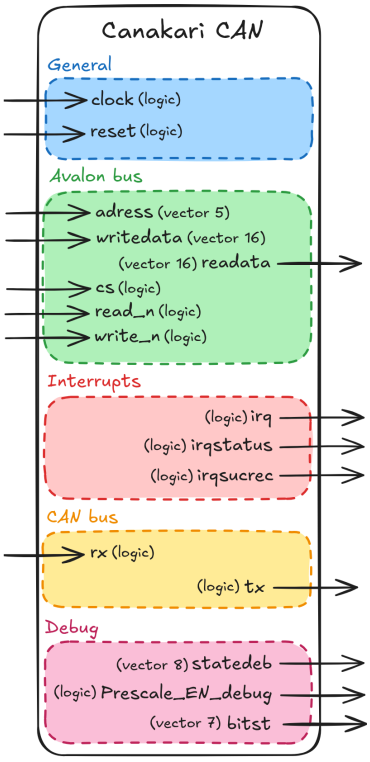


Figure 11: Interface de l'IP Canakari

3.3.2.2 - Configuration

La configuration de l'IP via le CPU se fait en lisant et écrivant les registres internes via le bus Avalon. Chaque registre est associé à une adresse spécifique, permettant ainsi au processeur de configurer le contrôleur, d'envoyer des trames ou de lire les trames reçues. En fonction de l'état des registres, l'IP effectue les opérations demandées et met à jour les **flags** d'état et d'interruption. L'IP dispose en tout de 21 registres de 16 bits, accessibles en lecture et/ou écriture.

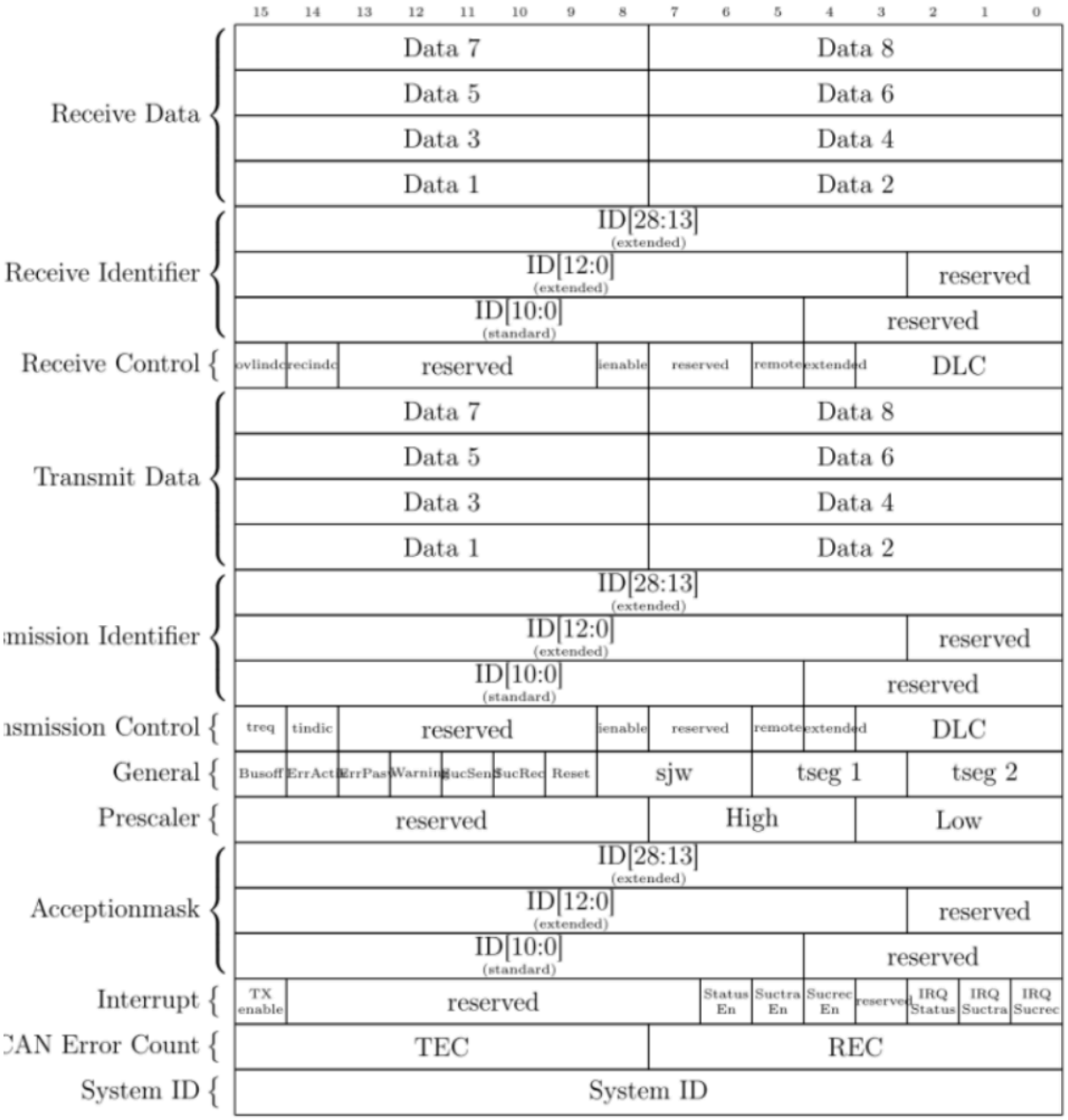


Figure 12: Banque de registres de l'IP Canakari

3.3.3 - Utilisation

Selon la documentation, l'initialisation suit la séquence suivante :

1. Effectuer un **hard reset**, attendre 4 cycles d'horloge, puis patienter 2 cycles supplémentaires.
2. Réaliser un **soft reset** en écrivant dans le registre prévu, puis attendre 4 cycles.
3. Configurer les registres.
4. Vérifier que le contrôleur est en état **idle** (`statedeb = 0x9`).
5. Pour la réception, attendre une interruption.
6. Pour la transmission, configurer les registres d'envoi.

Apres l'initialisation, l'IP peut être utilisée pour envoyer et recevoir des trames CAN. La réception est asynchrone et déclenche une interruption lorsque une trame est reçue. La transmission nécessite de configurer les registres d'envoi et de lancer l'opération. L'IP gère automatiquement l'arbitrage et la retransmission en cas d'erreur.

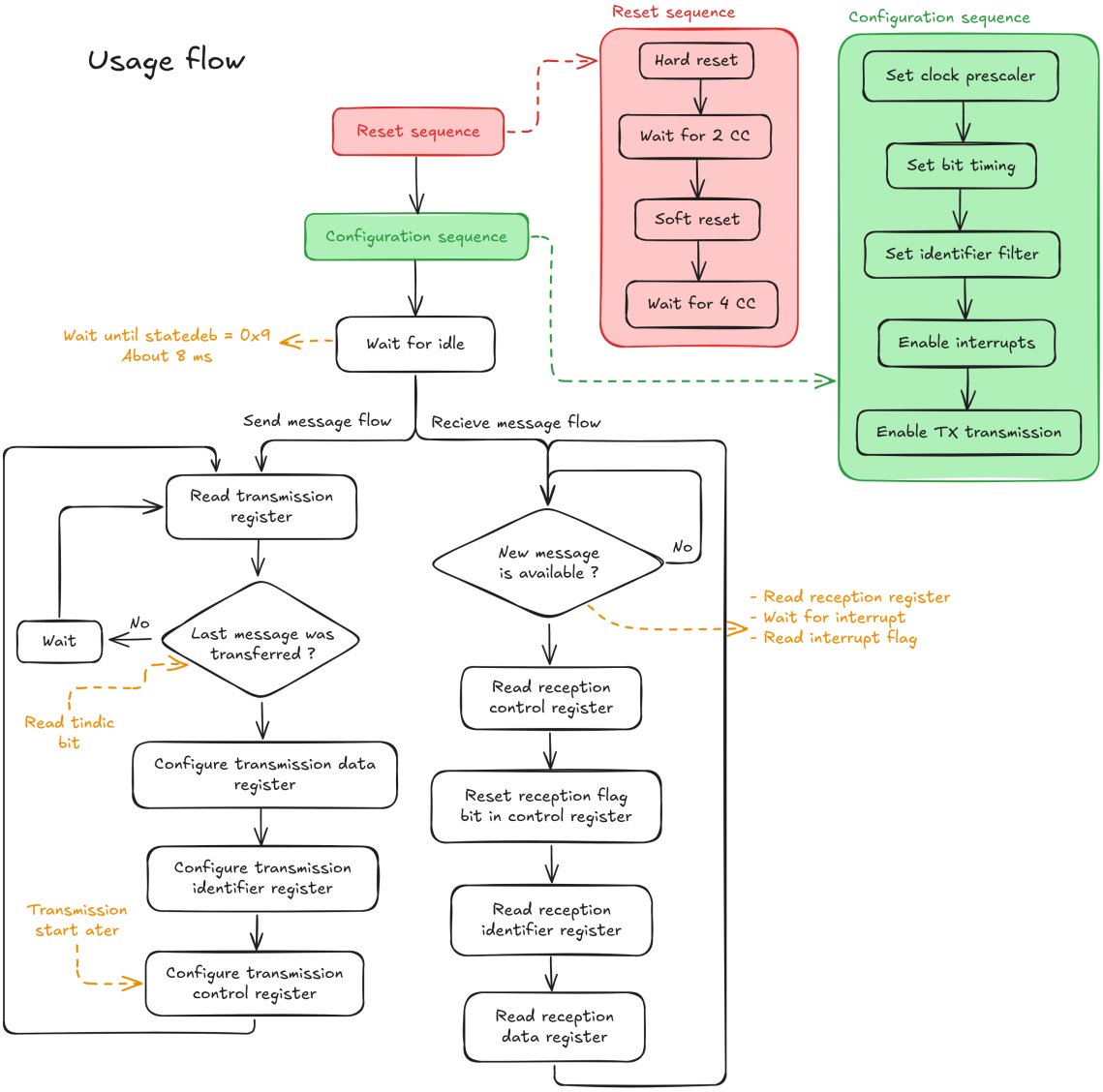


Figure 13: Flux d'utilisation de l'IP Canakari

3.4 - Adaptation et amélioration

3.4.1 - Interface AXI

L'ordinateur de bord d'IonSat étant basé sur un FPGA Xilinx, il est nécessaire d'utiliser le protocole interne AXI (issu de l'AMBA d'ARM), différent du bus Avalon. L'IP a donc été adaptée pour remplacer son interface Avalon par une interface AXI4-Lite. La structure modulaire du code a facilité cette modification : seuls les blocs de gestion des signaux Avalon ont été remplacés par des blocs AXI, tandis que les multiplexeurs de lecture/écriture des registres ont été conservés.

Une particularité d'AXI est la gestion des écritures partielles : bien que le bus soit de 32 bits, il permet par exemple de modifier seulement 8 bits d'un registre. Chaque registre a donc dû être adapté pour gérer correctement cette fonctionnalité.

Chaque module modifié a ensuite été testé de manière unitaire (registres, contrôleur AXI) afin de garantir la validité des changements.

3.4.2 - Corrections de bugs

Les premiers tests ont mis en évidence une erreur dans la gestion des registres de réception. Le registre associé contenait un bit permettant d'indiquer si le message reçu utilisait le format standard ou étendu. Dans la version d'origine, ce bit pouvait uniquement être écrit par l'utilisateur, et non par le contrôleur lui-même.

Ce comportement incohérent a été corrigé en reliant directement le signal interne du MAC (indiquant le format reçu) au registre. Cette modification était également nécessaire pour permettre le filtrage correct des identifiants reçus.

3.4.3 - Améliorations de la base de code

Le projet étant issu d'une université allemande et ayant connu plusieurs contributeurs étudiants, la base de code souffrait d'un manque de cohérence : fichiers mal nommés, commentaires partiellement en allemand, conventions de nommage hétérogènes.

Un important travail de remise en forme a donc été effectué :

- renommage des fichiers pour une organisation plus claire,
- traduction des commentaires en anglais,
- formatage homogène du code,
- uniformisation des noms de signaux internes selon une convention définie préalablement.

3.5 - Validation et tests

Chaque modification a été validée par des tests unitaires, puis l'IP complète a été soumise à une campagne de test plus large. Étant donné le caractère critique du contrôleur CAN pour IonSat, il était indispensable de couvrir un maximum de scénarios, y compris des cas d'erreur volontairement introduits.

Les tests se sont déroulés en trois étapes :
1. simulation des composants internes modifiés (registres, interface AXI),
2. simulation complète de l'IP,
3. validation sur carte dans un réseau CAN comprenant plusieurs nœuds équipés de contrôleurs commerciaux.

Un banc de test a été mis en place pour standardiser les validations. Il comprend l'IP CAN implémentée sur FPGA, un contrôleur CAN externe commercial de référence, une carte Arduino servant de maître de communication, ainsi qu'un analyseur logique pour observer les échanges sur le bus.

3.5.1 - Différents tests réalisés

Chaque fonctionnalité de l'`IP` a été testée dans différents environnements et avec plusieurs configurations. Les tests se divisent en deux grandes catégories : tests positifs (conditions normales) et tests négatifs (conditions d'erreur).

3.5.1.1 - Tests positifs

Ces tests valident le fonctionnement de l'`IP` dans des conditions d'utilisation normales, sans provoquer volontairement d'`erreurs` ou de comportements inattendus. Ils permettent de vérifier le bon fonctionnement général de l'`IP` :

- Vérification des registres de configuration (prescaler, bit timing, interruptions, filtres d'identifiants).
- Transmission de trames de données et de requêtes en mode standard et étendu.
- Réception correcte de trames de données et de requêtes.
- Gestion correcte de l'arbitrage lors de la présence de plusieurs nœuds.

3.5.1.2 - Tests négatifs

Ces tests ont pour objectif de vérifier le comportement de l'`IP` dans des situations anormales ou imprévues, afin de s'assurer qu'elle réagit de manière appropriée et qu'elle ne génère pas d'`erreurs critiques` :

- Envoi de trames invalides (longueur incorrecte, absence de signal TX).
- Détection d'erreurs de bit, de CRC, de format et d'acquittement.
- Validation du passage automatique entre les modes d'erreur (actif, passif, bus-off) et du retour au mode normal après réinitialisation.

3.6 - Empaquetage de l'IP

Une fois l'IP développée et validée, il restait à la rendre facilement réutilisable par d'autres ingénieurs. Pour cela, elle a été empaquetée sous la forme d'un module complet, accompagné de toute la documentation et des outils nécessaires à son intégration dans de futurs projets. L'objectif n'était plus uniquement de disposer d'un code fonctionnel, mais de livrer un produit final, exploitable et maintenable.

L'empaquetage inclut plusieurs éléments complémentaires :
- une documentation utilisateur,

- des pilotes logiciels (**device drivers**),
- un exemple d'utilisation,
- des tests,
- ainsi que des scripts Vivado pour automatiser la génération de l'IP et la création d'un projet de démonstration.

3.6.1 - Documentation

La documentation initiale fournie avec le projet open source décrivait essentiellement le fonctionnement interne des blocs matériels. Bien que pertinente pour comprendre l'architecture, elle n'était pas adaptée à un utilisateur souhaitant simplement intégrer l'IP. Une nouvelle documentation a donc été rédigée, en français et en anglais, en se concentrant sur l'usage pratique : description des registres accessibles, procédure d'initialisation, et guide pas à pas pour l'intégration dans un projet Vivado. Cette approche permet de considérer l'IP comme un composant fini, prêt à l'emploi.

3.6.2 - Device drivers

Afin d'exploiter l'IP depuis la partie logicielle, des pilotes bas niveau (**device drivers**) ont été développés en langage C. Ces pilotes fournissent les adresses des registres ainsi que des fonctions pour effectuer les opérations essentielles : configuration du contrôleur, écriture et lecture des trames, gestion des interruptions. Ils constituent la couche d'abstraction minimale nécessaire pour interfacer proprement le matériel avec le logiciel embarqué.

3.6.3 - Exemple

Pour faciliter la prise en main, un exemple complet d'émission et de réception de trames CAN a été fourni. Celui-ci illustre une configuration générique de l'IP et montre comment utiliser les pilotes pour initialiser le contrôleur, transmettre une trame et traiter les messages reçus. Cet exemple joue le rôle de démonstrateur et permet de valider rapidement l'intégration.

3.6.3.1 - Script Vivado

Enfin, deux scripts Vivado ont été ajoutés pour automatiser le flux d'utilisation. Le premier permet de générer automatiquement le bloc IP à partir du code source VHDL, tandis que le second crée un projet d'exemple intégrant l'IP, ses pilotes et le code de démonstration. L'objectif est de réduire au maximum la manipulation manuelle et de proposer une intégration reproductible et rapide.

3.7 - Conclusion

Le développement de l'IP CAN à partir de l'IP open source **Canakari** a constitué un projet complet, couvrant toutes les étapes depuis l'adaptation d'une base open source jusqu'à la livraison d'un composant prêt à l'emploi. Le travail a inclus la modification de l'interface pour la rendre compatible avec les FPGA Xilinx, la correction de bugs, l'amélioration de la qualité du code, ainsi qu'une campagne de tests rigoureuse pour valider chaque fonctionnalité. Ce projet m'a aussi permis d'apprendre en profondeur le fonctionnement du protocole CAN, ainsi que la méthodologie pour l'empaquetage final

avec documentation, pilotes et exemples visant à faciliter la réutilisation de l'IP dans de futurs projets, contribuant ainsi à la pérennité du travail accompli.

4 - Architecture hardware de IonSat

Ma seconde mission a consisté à élaborer une première version de l'architecture matérielle FPGA, se rapprochant autant que possible de la configuration finale. L'objectif était de fournir aux équipes logicielles une plateforme de test réaliste leur permettant de poursuivre le développement et de valider l'intégration avec les différentes IPs FPGA.

Pour cela, je me suis appuyé dans un premier temps sur l'architecture développée pour le satellite **EyeSat** du CNES. Ce projet, reposant sur une plateforme similaire, a servi de base de travail : nous avons repris son hardware afin de l'adapter progressivement aux besoins spécifiques de IonSat.

4.1 - Hardware EyeSat

Le satellite EyeSat intègre un ensemble d'IPs, dont certaines ont été développées par des étudiants de la **Nanolab Academy**, tandis que d'autres proviennent de projets antérieurs du CNES. Grâce aux sources et aux scripts fournis par le CNES, il a été possible de recréer l'environnement matériel dans Vivado. L'architecture embarque notamment des IPs dédiées à la génération des signaux TM et au décodage des TC en bandes S et X, à la gestion des mémoires Flash, ainsi qu'à divers protocoles de communication tels que **SpaceWire** et **I²C**. Elle comprend également des IPs spécialisées pour le contrôle du GPS et du tracker solaire, ainsi que pour la gestion des erreurs et le suivi des versions.

4.2 - Besoins d'IonSat

Toutes les IPs présentes dans l'architecture d'EyeSat n'étaient pas nécessaires pour la mission IonSat. La plateforme a donc été adaptée afin de ne conserver que les blocs pertinents, à savoir :

- l'IP de gestion des TM/TC en bande S,
- une IP I²C dédiée au contrôle des composants externes,
- une IP I²C pour la gestion des composants internes de l'ordinateur de bord.
- une IP de gestion des erreurs mémoire.
- une IP de gestion des versions.

4.3 - Automatisation du projet

Pour faciliter l'évolution de l'architecture FPGA, l'outil Vivado offre la possibilité d'utiliser des scripts afin d'automatiser les opérations répétitives, telles que l'importation des IPs, leur interconnexion et la compilation. Ces scripts permettent notamment de simplifier les mises à jour, qu'il s'agisse de corriger le code d'une IP ou de modifier un paramètre interne, en évitant une manipulation manuelle fastidieuse via l'interface graphique. Ils constituent également un moyen efficace de partager le projet, puisqu'il suffit de diffuser les sources et les scripts de génération, sans avoir à transmettre l'ensemble des fichiers d'état produits par Vivado.

L'adaptation de la plateforme matérielle pour IonSat a nécessité plusieurs ajustements aux scripts d'origine, principalement pour retirer les composants non utilisés dans la mission et pour optimiser l'allocation des ressources logiques.

4.4 - Portage vers la plateforme de développement

La carte **Ninano**, destinée aux tests en salle blanche, est trop coûteuse pour être utilisée en plusieurs exemplaires. Afin de disposer d'un environnement de travail quotidien, les développements et validations initiales ont donc été réalisés sur une carte **Xilinx Zybo**, plus accessible. Bien que moins performante, cette carte repose sur le même processeur que la Ninano dans une version réduite. Elle constitue ainsi une plateforme de test pratique pour développer les partitions logicielles et vérifier leur interaction avec les composants matériels, avant leur déploiement sur la véritable carte de vol en salle blanche.

4.4.1 - Contraintes d'optimisation

La Zybo, en raison de ses ressources logiques limitées, ne permet pas d'intégrer l'ensemble des IPs prévues pour IonSat. Afin de réaliser malgré tout les tests logiciels, plusieurs variantes du hardware ont été générées, chacune contenant un sous-ensemble spécifique d'IPs. Cette étape a nécessité un important travail d'optimisation et de validation afin d'exploiter au maximum les capacités de la Zybo tout en conservant un environnement représentatif pour le développement logiciel.

4.5 - Conclusion

Le portage de l'architecture matérielle FPGA d'EyeSat vers IonSat a constitué une étape importante pour le développement du satellite. L'adaptation et l'optimisation de la plateforme existante ont permis de fournir à l'équipe logicielle une première version fonctionnelle, servant de support pour valider les interactions entre matériel et logiciel et préparer l'intégration finale sur la carte de vol Ninano. L'automatisation du projet par des scripts a par ailleurs amélioré l'efficacité du flux de développement et renforcé la traçabilité des évolutions de l'architecture matérielle.

Cette deuxième mission m'a permis d'acquérir une réelle maîtrise des scripts d'automatisation Vivado, qu'il s'agisse de la génération de projets, de l'intégration d'IPs ou de la compilation. Elle m'a également offert l'occasion d'approfondir mes connaissances des IPs utilisées dans le domaine spatial, notamment pour la communication, la gestion des mémoires et les protocoles embarqués.

5 - Conclusion

TODO

Bibliographie

TODO