Abstract

TODO

Résumé

TODO

Remerciements

TODO

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

breve Introduction

2 - Présentation de l'entreprise

2.1 - 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.

2.2 - 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 (cf. section TODO)
- 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.

3 - Sujet et contexte du stage

3.1 - 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.

3.2 - Context du stage

3.2.1 - 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 moteur à ions. 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 moteur à ions 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.

3.2.2 - 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.

3.2.3 - Phases de développement du projet

Le développement d'un satellite suit un processus normé, structuré en plusieurs phases successives, comme présenté dans le tableau ci-dessous. Le projet IonSat se trouve actuellement en phase D, la plus longue, mais aussi la dernière étape avant le lancement. Cette phase concentre la majeure partie du travail de développement d'intégration électronique et logicielle pour relier tous les sous-systèmes du satellite.

Phase	Description
Phase 0	Analyse de la mission – Identification des besoins
Phase A	Étude de faisabilité
Phase B	Définition préliminaire
Phase C	Définition détaillée
Phase D	Production / Intégration / Qualification au sol
Phase E	Opérations en orbite
Phase F	Fin de vie / Retrait de service

3.2.4 - 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.

3.3 - Sujets et objectifs du stage

Dans le cadre du projet IonSat, mon stage d'ingénieur 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.

4 - Architecture d'IonSat

4.1 - Composants matériels

En plus des éléments dédiés aux 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.

4.1.1 - 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 sa bonne capacité de pénétration de l'atmosphère terrestre.

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.

4.1.2 - 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 les capteurs 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, ainsi qu'un petit propulseur pour modifier l'orbite et effectuer des corrections de trajectoire.

4.1.3 - 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 en service du satellite ou lors de son désorbitage en fin de vie.

4.1.4 - 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.

4.1.5 - 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 disposition matérielle des 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**, mieux adaptés à des échanges de moindre volume et permettant une intégration plus simple.

4.2 - Logiciel embarqué

4.2.1 - 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 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.

4.2.2 - 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.

5 - Réalisation d'une IP CAN

5.1 - 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.

5.2 - 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 (version 2016), définissant la version 2 du protocole.

Très répandu dans l'automobile, le CAN est apprécié pour sa robustesse, sa capacité à fonctionner sur de longues distances (jusqu'à 1 km, bien au-delà de l'I^2C 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 erreurs et un débit pouvant atteindre 1 Mbit/s (débit brut, incluant les bits de trame).

5.2.1 - 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 électromagnétique 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.

TODO image, potentiollement de ce site : https://se1.isc.heia-fr.ch/lecture/mcu/can/#couche-physique

5.2.2 - 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**).

5.2.3 - 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 :

TODO image trame de données, celle la est bien https://se1.isc.heia-fr.ch/lecture/mcu/can/#trame-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.

5.2.4 - Trame de requette

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.

5.2.5 - 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.

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.

TODO image

5.3 - Canakri : une version open-source

5.3.1 - 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.

5.3.2 - 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. Le fichier top-level `can.vhdl` assure l'interconnexion entre ces blocs et doit être instancié dans le projet FPGA pour intégrer l'IP.

5.3.2.1 - Interface

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

TODO image

5.3.2.2 - Signaux généraux

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.

5.3.2.3 - 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.

5.3.2.4 - Bus CAN

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

5.3.2.5 - 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.

5.3.3 - Utilisation

L'accès aux registres via l'interface Avalon permet de configurer l'IP, de lancer des transmissions ou de lire les trames reçues.

Banque de registres :

TODO image registres

5.3.3.1 - Initialisation

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.

TODO image usage flow

5.4 - Adaptation et amélioration

5.4.1 - 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.

5.4.2 - 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.

5.4.3 - 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.

5.5 - Validation et tests

Chaque modification a été validée par des tests unitaires, puis l'IP complète a été soumise à des campagnes de test plus larges. 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.

5.5.1 - Banc de tests

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.

TODO image banc de test

5.5.2 - 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).

5.5.2.1 - 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.

5.5.2.2 - 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.

5.6 - 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.

5.6.1 - 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.

5.6.2 - 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.

5.6.3 - 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.

5.6.3.1 - 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.

6 - Conclusion

TODO

Bibliographie

TODO

Annexe 1 : auto-evaluation

TODO

Annexes

TODO