Abstract

TODO

Résumé

TODO

Remerciements

TODO

Sommaire

1 - Introduction 1
2 - Présentation de l'entreprise 2
2.1 - Le CSEP 2
2.2 - Les activités du CSEP 2
3 - Sujet et contexte du stage 3
3.1 - Les nanosatellites - CubeSats 3
3.2 - Context du stage 3
3.2.1 - Le projet IonSat 3
3.2.2 - L'équipe du projet IonSat 4
3.2.3 - Phases de développement du projet 4
3.2.4 - Présentation du Nanolab academy 5
3.3 - Sujets et objectifs du stage 5
4 - Architecture d'IonSat 7
4.1 - Composants matériels 7
4.1.1 - Télécommunications 7
4.1.2 - Orientation et déplacement 7
4.1.3 - Gestion de puissance 8
4.1.4 - Ordinateur de bord 8
4.1.5 - Interconnexion des composants 8
4.2 - Logiciel embarqué 8
4.2.1 - FPGA 8
4.2.2 - Logiciel de bord 9
5 - Réalisation d'une IP CAN 10
5.1 - Introduction 10
5.2 - Le CAN 10
5.2.1 - Le bus CAN 10
5.2.2 - Le protocole CAN 11
5.2.3 - Trame de données 11
5.2.4 - Trame de requette 12
5.2.5 - Arbitrage 12
5.2.6 - Synchronisation 12
5.2.7 - CRC (Cyclic Redundancy Check) 15
5.2.8 - ACK 15
5.2.9 - Bit stuffing 15
5.2.10 - Intertrame 15
5.2.11 - Bit monitoring 15
5.2.12 - Gestion des erreurs 16
5.3 - Canakri : une version open-source 17
5.3.1 - Fonctionnement général de ' 17
5.4 - Adaptation et amélioration de ' 17
5.4.1 - Interface AXI 17
5.4.2 - Corrections de bugs 17
5.4.3 - Améliorations de la base de code 17
5.5 - Validation et tests 17
5.5.1 - Banc de tests 18
5.5.2 - Différents tests réalisés 18

5.6 - Empaquetage de l'IP 18
5.7 - Conclusion 18
6 - Architecture hardware de IonSat 19
7 - Conclusion 20
Bibliographie 21
Annexe 1 : auto-evaluation 22
Annexes 23

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 CAN est une composante essentiel pour la communication de plusieurs system critique. Il relie l'ordinateur de bord, le propulseur, et le system de control d'attitude. C'est system sont cruciaux pour la survie du satellite et les communication doivent être gérer correctement pour prevenir les erreurs dû au perturbation electronique ou aux radiations spatiales.

Le PCB étant déjà existant, l'ordinateur de bord dispose d'un transceiver CAN pour la generation des signaux electroique sur le bus mais il faut aussi disposer d'un controller CAN pour le protocole. Ce controlleur peut etre implementer es plusieurs facon, soit par un puce dédié externe, soit si le CPU dispose d'n controlleur integré ou sous forme d'IP pour FPGA. L'avantage des deux derniers solutions est que le controlleur est ainsi au plus proche du CPU et protogé contre les radiations. L'utilisation d'un FPGA apporte plus de flexibilité dans l'implentation et les fonctionnalitées.

La premiere partie du stage à ainsi consisté en la realisation d'une IP CAN pour FPGA Xilinx.

5.2 - Le CAN

Tout d'abord, il convient d'expliquer brievement comment fonctionne le CAN. Le CAN, developé par Bosh, correspond a deux choses : un bus de communication et un protocole, il correspond donc au deux couches basse du model de communication OSI : la couche physique et la couche liaison de données. Le CAN est défini dans plusieurs norme ISO, dont la norme ISO 11898 de 2016 qui definie la version 2 du protocol de communication et qui est celle avec laquelle j'ai travaillé.

Très largement utilisé dans l'automobile, le CAN est particulièrement apprécié pour sa capacité à fonctionner sur de longues distances, supérieures à celles du SPI ou de l'I2C (jusqu'à 1 kilomètre de câble). Il se distingue également par sa robustesse face aux perturbations et sa gestion avancée des erreurs

Parmi ses principaux avantages, le protocole offre :
- une communication multi-maître asynchrone,
- une tolérance élevée aux erreurs
- un débit pouvant atteindre 1 Mbit/s (ceci inclut les bits de trame ; le débit utile est donc inférieur).

5.2.1 - Le bus CAN

Le couche physique du bus CAN est définie par la norme ISO 11898-2 pour le CAN haute vitesse. Le bus de donnée se compose d'une simple paire différentiel auquel sont connecté tous les neouds sur le bus. Un noeuds CAN est composé d'un controlleur CAN pour la gestion du protocol et d'un transceiver pour la lecture et generation des niveaux de tensions sur le bus.

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 dominant.

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 protocol CAN est assez complexe a cause de la simplicite du bus (pas de clock, ni de signal de control) mais aussi car il permet un tres bonne gestion des erreurs. Le bus etant aussi multi-maitre, le protocol inclue aussi la gestion d'abitration du bus et de priorité.

Un controller peut transmettre trois type de trame sur le bus :
- une trame de données
- une trame de requette
- une trame d'erreur

5.2.3 - Trame de données

Les trame de données, servent a transferer de l'information a un ou plusieurs autre noeuds, cette données d'entre 1 et 8 octets est associé a un identifiant. L'utilisation d'un identifiant de données permet de transmettre une meme information a plusieurs noeuds, ce n'est pas l'adresse du destinataire.

Une trame de données est formatté comme suit :

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

Detail des champs :
- SOF : Start of Frame, premier bit dominant pour indiquer le debut d'un transfert
- Identifier : 11 bit d'identifiant pour la données
- SSR : champ réservé, à niveau récessif (présent uniquement dans les trames étendues).
- IDE : identifier extension, indiquant si l'identifiant est standard (11 bits) ou étendu (29 bits)
- Identifier 2 : 18 bit supplémentaire d'identifiant pour la données sur utilisation du mode etendu
- RTR : remote transmission request, indiquant une trame de données 1 ou de requette 0
- R1 : reservé pour d'autre version du protocol, forcé a 1
- R0 : reservé pour d'autre version du protocol, forcé a 1
- DLC : Data Length Code, indiquant la taille des données (0 à 8 octets)
- Data : les données a transmettre, de 0 a 8 octets
- CRC : Cyclic Redundancy Check, utilisé pour détecter les erreurs dans la trame
- ACK : Acknowledgment, utilisé pour confirmer la réception d'une trame par les autre noeuds
- EOF : End of Frame, dernier bit dominant pour indiquer la fin d'un transfert

Les trames de noeuds peuvent utiliserr des modes : le mode standar ou etendu. En mode etendu, la taile de l'identifiant passe a 29 bit au total (11 + 18). L'utilisation de ce mode est indiqué par le bit IDE.

5.2.4 - Trame de requette

Une trame de requette est composé comme une trame de données a ceci pres que le champ de données reste vide et que le bit RTR est à 1. Toutefois le champ DLC peut etre non nul pour indiquer la taille des données attendues.

5.2.5 - Arbitrage

Le bus etant multi-maitre, chaque noeuds pour vouloir parler sur le bus a tous moment. Pour eviter les conflit et ne pas erroner des trames ou perdre des données, le protocol implemente une phase d'arbitrate pour données le main sur le bus a un seul noeuds.

Dans le cas ou le bus est a niveau recessif (et qu'un transmission n'est pas en cours), un noeuds pour commencer une transmission avec le SOF (bit dominant) pour prendre le bus. Tout les autres noeuds vont alors se mettre en mode reception et attendre que le bus soit libre pour envoyer des données. Si toutefois deux noeuds ou plus venait a transmettre un SOF en même temps aucun de ces noeuds pourrai s'assurer qu'il a bien la main sur le bus, c'est l'identifian de 11 qui va permettre d'effectuer l'arbitrage.

Le protocol indique que chaque noeuds doit parallement pouvoir etre et lire sur le bus. Ainsi chaque noeuds est tenu de lire le niveau du bus apres avoir transmis un bit. Le niveau dominant etant prevalant, si un noeuds transmet un niveau recessif mais que le bus reste dominant, alors un autre noeuds est lui aussi en train de transmettre. Dans ce cas, e noeuds transmettant un bit recessif arrette sa transmission est devient un recepteur. La phase d'abitration dure pendant toute la transmission de l'identifiant, l'identifiant le plus proche de 0 est ainsi le plus prioritaire.

A la fin de la transmission de l'identifiant, comme deux noeuds ne peuvent pas transmettre deux identifiants identique (mais peuvent transmettre plusieurs identifiant différent), le noeuds restant en mode transmission est sur d'être seul sur le bus a ce moment la est peut commencer a transmettre la suite de la trame.

TODO image

5.2.6 - Synchronisation

Le bus ne disposant pas de signal d'horloge, les noeuds recepteur doivent se synchroniser sur l'emmetteur pour lire les données. Le protocol définie des mode de synchronisation basé sur un découpage temporel des transmission. Ainsi protocole CAN repose sur une synchronisation bit à bit, qui s'appuie sur les transitions du signal, des fronts récessif dominant. Ces fronts servent de repères pour corriger les décalages d'horloge.

5.2.6.1 - Bit timing

Le `bit timing` (ou `nominal bit time`) correspond à la durée de transmission d'un bit sur le bus. Il fixe donc le débit de transmission : plus cette durée est courte, plus le débit dans le bus est élevé.

Le `bit time` est divisé en plusieurs segments pour permettre l'échantillonnage et la correction de phase.

5.2.6.2 - Time Quantum (TQ)

Le `Time Quantum (TQ)` est l'unité élémentaire de temps utilisée pour diviser le `bit time`. Un `bit time` est donc composé d'un nombre entier de TQ : `bit time = N x TQ`

Le bit time est découpé en 4 segments :

- SYNC_SEG : toujours 1 TQ — utilisé pour détecter les transitions.
- PROP_SEG : compense les délais de propagation (entre 1 et 8 TQ).
- PHASE_SEG1 (aussi appelé BUFFER_SEG1) : avant l'échantillonnage (entre 1 et 8 TQ), permet une correction positive de phase.
- PHASE_SEG2 (BUFFER_SEG2) : après l'échantillonnage (entre 2 et 8 TQ), permet une correction négative de phase.

TODO image

Tous les nœuds du bus doivent partager la même configuration du bit timing, notamment les longueurs de TQ et des différents segments.

5.2.6.3 - SYNC SEG

Le segment SYNC (1 TQ) marque le début de chaque bit et doit contenir un front descendant (récessif ⟶ dominant). Si ce front n'est pas détecté, une re-synchronisation peut être déclenchée pour réaligner le timing.

5.2.6.4 - PROP SEG

Cette phase permet de compenser les retards dus à la conversion des signaux (ADC/DAC) et à la propagation dans les fils. Sa durée est fixée lors de la configuration et est généralement calculée en fonction de la longueur du bus.

5.2.6.5 - BUFFER SEG 1

Segment avant l'échantillonnage du bit. Sa durée initiale est fixe, mais elle peut être temporairement augmentée si une re-synchronisation le nécessite.

Une plus grande valeur (en nombre de TQ) de `BS1` offre une meilleure tolérance aux décalages entre les horloges des nœuds.

Le sample point, où la valeur du bit est lue, se trouve à la fin du `BS1`.

5.2.6.6 - BUFFER SEG 2

Segment après l'échantillonnage du bit. Il est également utilisé pour ajuster le timing en cas de décalage, mais à l'inverse du `BS1`, il peut être réduit lors d'une re-synchronisation.

Sa durée est comprise entre 2 et 8 TQ. Selon la norme ISO 11898, certaines contraintes s'appliquent :

- `BS2` ne doit pas être inférieur à `2 TQ`.
- `BS2` ne peut pas être ajusté lors d'une synchronisation "hard" (changement de bit).
- `BS1` doit être supérieur ou égal à `BS2` dans certaines implémentations matérielles.

5.2.6.7 - Modes de synchronisation

Il existe deux types de synchronisation : la synchronisation forcée (**hard synchronization**) et la re-synchronisation (**re-synchronization**).

Hard synchronisation

La synchronisation forcée intervient dans deux cas :
- Lors de la détection d'un Start of Frame (SOF).
- Lorsqu'un front récessif ⟶ dominant est détecté pendant le `SYNC_SEG`.

Dans ces cas, le `bit time` en cours est immédiatement abandonné, et un nouveau comptage commence à la fin du `SYNC_SEG`, juste après le front détecté. Cela permet de réaligner parfaitement les horloges sur un front fiable.

Re-synchronisation

La re-synchronisation a lieu lorsqu'un front récessif ⟶ dominant est détecté en dehors du `SYNC_SEG`. Dans ce cas, on ajuste dynamiquement la durée des segments `BS1` et `BS2` pour corriger le décalage temporel.

L'erreur de phase

L'erreur de phase `e` est calculée comme la différence entre la position réelle du front et la fin théorique du `SYNC_SEG` :
- Si le front est en avance, alors `e > 0`.
- Si le front est en retard, alors `e < 0`.

Le front est considéré en retard s'il intervient avec le point d'échantillonnage et en avance s'il intervient après celui-ci mais avant le `SYNC_SEG`.

`e` s'exprime en nombre de TQ.

Max Jump Width (RJW)

Le `RJW` est une valeur définie pour limiter l'amplitude des corrections de phase. Elle est calculée ainsi : `RJW = min(4, BS1)`

Application du décalage

L'ajustement des segments se fait comme suit :
- Si `e > 0`, on allonge `BS1` de `min(e, RJW)`.
- Si `e < 0`, on raccourcit `BS2` de `min(|e|, RJW)`.

Ces modifications assurent un réalignement progressif des horloges sans perturber l'échantillonnage.

5.2.7 - CRC (Cyclic Redundancy Check)

Le CRC est utilisé pour vérifier l'intégrité des données transmises. L'émetteur calcule un CRC à partir du contenu de la trame, et l'insère dans le champ `CRC`.

Les récepteurs effectuent le même calcul à réception de la trame. Si la valeur reçue diffère de la valeur calculée, une erreur CRC est détectée (cf. Gestion des erreurs).

5.2.8 - ACK

L'acquittement utilise les deux bits du champ `ACK` :

- Le premier bit est laissé récessif par l'émetteur. Tous les récepteurs qui ont bien reçu la trame et qui n'ont détecté aucune erreur forcent ce bit à l'état dominant.
- Le deuxième bit, appelé ACK delimiter, est toujours récessif.

Si aucun récepteur ne force le bit dominant, l'émetteur considère que l'envoi a échoué (ACK error).

5.2.9 - Bit stuffing

Pour garantir la présence régulière de fronts sur le bus (utiles à la synchronisation), un bit de stuffing est inséré automatiquement après 5 bits consécutifs de même valeur.

Exemple : `11111` devient `111110` (un bit de valeur opposée est inséré).

TODO image

5.2.10 - Intertrame

Une période de repos appelée intertrame est imposée entre deux trames (données ou requêtes).

Elle dure au minimum 3 bits récessifs consécutifs. Ce délai permet aux récepteurs de traiter la trame précédente et au bus de se stabiliser.

Si aucune nouvelle trame n'est envoyée, le bus reste au niveau récessif (idle).

Note : Dans le cas d'un nœud en mode d'erreur passive venant de transmettre un message, un champ de suspension de transmission doit être ajouté juste après l'intermission. Ce champ dure 7 bits récessifs en plus des 3 bits d'intertrame. TODO check 7

5.2.11 - Bit monitoring

Le bit monitoring consiste à lire les bits envoyés pour vérifier leur cohérence avec ce qui est transmis.

Cela permet de :

- Détecter une perte d'arbitrage (par exemple si un bit récessif est lu dominant).
- Vérifier que le contrôleur CAN transmet correctement, via son module de réception indépendant.

5.2.12 - Gestion des erreurs

Il existe 5 types d'erreurs possibles sur le bus CAN :

1. Bit Error : L'émetteur lit un bit différent de celui qu'il a envoyé.

   Exceptions :
   - Pendant l'arbitrage.
   - Si personne ne répond à l'ACK.
2. Stuffing Error : Plus de 5 bits consécutifs identiques détectés (hors champs exclus).
3. CRC Error : Le CRC reçu ne correspond pas à celui calculé localement.
4. Field Delimiter Error : Un champ de délimitation (`ACK delimiter`, `CRC delimiter`) n'est pas récessif.
5. ACK Slot Error : L'émetteur ne détecte pas de bit dominant à l'emplacement de l'ACK, indiquant que personne n'a acquitté la trame.

5.2.12.1 - Trame d'erreur

En cas d'une erreur quelconque détectée par l'un des terminaux, celui-ci doit immédiatement transmettre une trame d'erreur.

Une trame d'erreur est composée de 2 champs :
- `flag error` (sur 6 bits) : peut être un flag `passif` ou `actif`
- `error frame delimiter` (8 bits) : forcé à un niveau `récessif`

5.2.12.2 - Mode erreur actif et mode passif

Chaque terminal possède deux compteurs :
- `TEC` (Transmit Error Counter)
- `REC` (Receive Error Counter)

Ils déterminent l'état du terminal :

TODO table format

| État des compteurs | Mode d'erreur | |—————————-|—————| | `TEC` ≤ 127 et `REC` ≤ 127 | Actif | | `TEC` > 127 ou `REC` > 127 | Passif | | `TEC` > 255 | Bus Off |

- En mode actif, le terminal émet une trame d'erreur avec 6 bits dominants.
- En mode passif, le terminal émet une trame d'erreur avec 6 bits récessifs.
- En mode Bus Off, le terminal se déconnecte du bus (ne participe plus aux échanges). Il ne peut revenir qu'après un reset logiciel ou matériel.

TODO image

5.2.12.3 - Valeur des flags d'erreur

- En mode passif, le terminal envoie un Passive Error Flag : 6 bits de niveau récessif.
- En mode actif, il envoie un Active Error Flag : 6 bits de niveau dominant.

5.2.12.4 - Transmission d'une trame d'erreur

Dès qu'une erreur est détectée par un terminal, celui-ci doit, à partir du bit time suivant, émettre une trame d'erreur composée :
- d'un champ Error Flag (actif ou passif selon l'état du terminal),
- suivi d'un Error Delimiter de 8 bits récessifs.

Note : À chaque détection d'une erreur CRC, la transmission d'un drapeau d'erreur commence au bit suivant le délimiteur ACK, sauf si un drapeau d'erreur pour une autre condition a déjà été commencé.

5.2.12.5 - Reset du bus

Le bus peut être réinitialisé si 128 séquences de 11 bits récessifs sont lues sur le bus. Tous les compteurs d'un nœud en mode bus off sont remis à 0.

5.3 - Canakri : une version open-source

5.3.1 - Fonctionnement général de ‘

Présentation de l'IP open-source choisie comme base de travail, ses caractéristiques principales et ses limites dans le contexte du projet.

5.4 - Adaptation et amélioration de ‘

5.4.1 - Interface AXI

Intégration d'une interface AXI4-Lite pour rendre l'IP compatible avec l'architecture FPGA/SoC d'IonSat.

5.4.2 - Corrections de bugs

Identification et correction des dysfonctionnements rencontrés dans le code source.

5.4.3 - Améliorations de la base de code

Refactorisation, ajout de commentaires, traduction et mise au propre de la base de code pour la rendre plus compréhensible et maintenable.

5.5 - Validation et tests

Présentation de la méthodologie de test adoptée et de son importance pour garantir la fiabilité de l'IP dans un environnement critique.

5.5.1 - Banc de tests

Description du banc de test utilisé (simulation, Modelsim, etc.), et de son organisation.

5.5.2 - Différents tests réalisés

Détails des scénarios de test : émission, réception, arbitrage, gestion d'erreurs, performance, etc.

5.6 - Empaquetage de l'IP

Présentation du processus de mise à disposition de l'IP :

- exemples de code d'utilisation,
- drivers logiciels,
- documentation technique.

5.7 - Conclusion

Résumé des apports de cette mission et importance de cette IP CAN dans l'architecture d'IonSat.

6 - Architecture hardware de IonSat

TODO

7 - Conclusion

TODO

Bibliographie

TODO

Annexe 1 : auto-evaluation

TODO

Annexes

TODO