

DesIGN FPGA pour   
Robot Autonome de Surveillance



Projet De Fin D’Etudes

21 Mars 2022 – 16 Septembre 2022

**VAN LEEUWEN Isabelle**

*INP ENSEEIHT filière 3EA - INSYS*

Année 2021- 2022

# Remerciements

Tout d’abord, je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et notamment François Couturier, mon tuteur technique durant ces 6 mois pour sa disponibilité et ses conseils.

Je remercie également Laurent Carpentier, responsable d’agence chez Elsys Design, pour m’avoir fait confiance et ainsi permis d’intégrer ELSYS Design pour y effectuer mon stage dans mon domaine de prédilection, sur un sujet aussi intéressant qu’instructif et formateur.

Je tiens aussi à remercier toute l’équipe d’Elsys Design pour leur accueil et leur esprit d’équipe, plus particulièrement Lenny Laffargue et Tristan Cornière, les deux autres stagiaires avec qui j’ai travaillé pendant l’intégralité de mon stage. Nos échanges ont permis de développer notre projet et j’ai beaucoup appris d’eux.

Enfin, je remercie le département 3EA ainsi que l’équipe pédagogique de l’ENSEEIHT qui m’ont suivie durant ces trois années au cours desquelles j’ai beaucoup appris, notamment sur l’aspect technique mais également méthodologique et relationnel d’un ingénieur. Je remercie en particulier Olivier Bernal et Blaise Mulliez pour la confiance et les compétences qu’ils m’ont partagés à multiples occasions et qui ont été un facteur de réussite pour ce stage.

Table des matières

[Remerciements 1](#_Toc112769852)

[Introduction 4](#_Toc112769853)

[I. Contexte 5](#_Toc112769854)

[1. L’entreprise : ELSYS Design 5](#_Toc112769855)

[a. Présentation de l’entreprise 5](#_Toc112769856)

[b. Domaine d’activité et savoir-faire 5](#_Toc112769857)

[2. Le projet 6](#_Toc112769858)

[II. Spécifications et architecture 7](#_Toc112769859)

[1. Présentation de l’existant 7](#_Toc112769860)

[2. Spécifications 7](#_Toc112769861)

[3. Choix de la carte 8](#_Toc112769862)

[4. Architecture 8](#_Toc112769863)

[III. Gestion de projet 11](#_Toc112769864)

[1. Diagramme de GanTT 11](#_Toc112769865)

[2. Gestion du budget 12](#_Toc112769866)

[IV. Travail réalisé 13](#_Toc112769867)

[1. Le générateur d’impulsions 13](#_Toc112769868)

[2. Gestion de l’IMU (Inertial Measurement Unit) 13](#_Toc112769869)

[a. Mise en place de la communication I2C 13](#_Toc112769870)

[b. Développement du gestionnaire d’IMU 14](#_Toc112769871)

[3. Algorithme de Dijkstra 16](#_Toc112769872)

[a. Le plan 16](#_Toc112769873)

[b. L’algorithme 16](#_Toc112769874)

[4. Gestion des ultrasons 17](#_Toc112769875)

[5. Gestion du RFID 20](#_Toc112769876)

[a. Mise en place de la communication 20](#_Toc112769877)

[[1] Communication avec le tag RFID click 20](#_Toc112769878)

[[2] ISO 14443 21](#_Toc112769879)

[b. Développement de la partie FPGA 23](#_Toc112769880)

[[1] Le protocole de communication UART 23](#_Toc112769881)

[[2] Développement du gestionnaire RFID 24](#_Toc112769882)

[6. Gestionnaire du minuteur de 2h 25](#_Toc112769883)

[7. Balises 25](#_Toc112769884)

[a. Mise en place du réseau Wifi MeSH 25](#_Toc112769885)

[b. Mise en place d’une communication UART 26](#_Toc112769886)

[V. Résultats 27](#_Toc112769887)

[Conclusion 28](#_Toc112769888)

[Bibliographie 29](#_Toc112769889)

Table des Figures

Figure 1 : Implantations d’ELSYS Design 5

Figure 2: La base Holonome 7

Figure 3: Spécifications de la Zybo Z7-20 8

Figure 4 : Architecture de notre système 9

Figure 5 : Block design de la partie FPGA 10

Figure 6 : GanTT en début de projet 11

Figure 7 : GanTT en fin de projet 11

Figure 8 : Gestion du budget 12

Figure 9 : Schéma du bloc générateur d'impulsions 13

Figure 10: Diagramme des timings I2C 13

Figure 11 : Schéma bloc du gestionnaire d'IMU 15

Figure 12 : Plan d'agence pour l'algorithme de Dijkstra 16

Figure 13: Schéma block pour l'algorithme de Dijkstra 17

Figure 14 : Principe de fonctionnement des ultrasons 17

Figure 15 : Cycle de fonctionnement des ultrasons 18

Figure 16 : Emplacement des ultrasons sur le robot 18

Figure 17 : Diagramme d'état du contrôleur d'ultrasons 19

Figure 18 : Schéma block du gestionnaire des ultrasons 20

Figure 19 : Liste des commandes du lecteur RFID 21

Figure 20 : Protocole d'envoi et de réception des commandes pour le lecteur RFID 22

Figure 21 : Trame de message UART type 23

Figure 22 : Schéma block du contrôleur RFID 24

Figure 23 : Schéma du réseau Wifi MeSH 26

Figure 24 : Image du robot final 27

# Introduction

On trouve de nos jours des robots dans tous les domaines : industrie, santé, transport et même dans la sécurité, un secteur dont se préoccupent de façon croissante les entreprises et les particuliers. Leur développement fait appel à plusieurs métiers de l’électronique et constitue donc un cadre idéal pour l’évaluation d’étudiants issus de filiales complémentaires en vue d’un recrutement.

C’est dans ce contexte qu’est proposé ce stage de fin d’études : Elsys Design a décidé de mobiliser une équipe pluridisciplinaire autour d’un projet dont le but est de développer un robot autonome de surveillance.

Ce projet a démarré en 2019 avec le développement d’une base holonome par un étudiant en alternance. Par la suite, les stages des années suivantes, y-compris celui de cette année, se sont construits autour de cette base holonome, avec des équipes de trois stagiaires, responsables chacun d’un métier : FPGA, Hardware, Software.

Ce stage étant reconduit pour la troisième année consécutive, il nous a été demandé de partir uniquement de la base holonome et de construire notre robot selon notre cahier des charges, sans considérer les projets précédents. Divers choix techniques ont dû être faits, dont le choix d’ordinateur de bord mais également des différents périphériques nécessaires au bon fonctionnement du robot.

Ce document résume le travail réalisé durant ces 6 mois, notamment toute la partie FPGA qui me concernait plus particulièrement.

Dans un premier temps, ce document présentera en détail le contexte de ce stage, notamment l’entreprise où j’ai évolué et le cadre du projet. Par la suite, seront détaillées les différentes spécifications de notre robot et son architecture ainsi que la gestion du projet qui en a résulté, comme le partage des tâches entre les 3 stagiaires, avant de passer au travail effectué durant le stage. Finalement, ce document dressera un bilan de ces 6 mois de stage.

# Contexte

## L’entreprise : ELSYS Design

### Présentation de l’entreprise

ELSYS Design est une société d’ingénierie spécialisée en conception de systèmes électroniques (métiers du hardware, du logiciel et des systèmes embarqués). Elle rassemble près de 800 ingénieurs au sein de 11 implantations en France, en Serbie (filiale ELSYS Eastern Europe) et dans la Silicon Valley (filiale ELSYS America).

ELSYS Design est une branche d’ADVANS Group, qui réunit plusieurs autres sociétés expertes dans les domaines du logiciel et de la mécanique.

Figure  : Implantations d’ELSYS Design

ELSYS Design imagine et conçoit des systèmes électroniques et logiciels de pointe. Elle accompagne les projets à forte valeur ajoutée de grandes sociétés internationales, de PME et de start-up.

Ses ingénieurs interviennent sur l’ensemble du cycle de développement de systèmes complexes, au sein de nombreuses industries dont l’aérospatial, l’automobile, la défense, l’énergie, le ferroviaire, l’IoT, le médical, le multimédia, les semi-conducteurs, les télécoms…

L’une des particularités d’ELSYS Design est d’avoir été fondée par des ingénieurs ; tous ses responsables d’affaires ont également suivi des cursus techniques.

### Domaine d’activité et savoir-faire

La mission d’ELSYS Design est d’imaginer et de concevoir les solutions technologiques de demain, au côté des acteurs majeurs des domaines de l’électronique et du logiciel.

Son activité se décompose en trois parties :

* L’activité Forfait (développements dans les bureaux d’étude d’ELSYS Design pour des clients extérieurs)
* L’activité Assistance Technique (les ingénieurs ELSYS Design renforcent les équipes des clients dans le cadre de missions régie)
* L’activité R&D et innovation

Du composant au système, ELSYS Design possède un large spectre de compétences dans tous les métiers du logiciel et de l’électronique.

* Logiciel embarqué
* Drivers, BSP, firmware
* Temps-réel, traitement du signal
* Logiciel industriel
* Microélectronique
* Numérique, analogique
* FPGA, SoC
* Circuit intégré, ASIC
* Carte électronique
* Numérique, analogique
* Puissance, radiofréquence
* Hyperfréquence

L’expertise d’ELSYS Design couvre toutes les étapes de la conception matérielle et logicielle :

* Architecture, études système
* Bancs de test
* Intégration, validation
* Test, industrialisation

## Le projet

La sécurité est une préoccupation croissante pour les entreprises, en effet de nombreuses données et informations confidentielles sont stockées sur les machines en local. Dans le but de surveiller ses locaux, Elsys Design a embauché trois stagiaires, chacun s’occupant d’une partie différente du projet :

* Un concepteur Hardware : responsable du développement d’une carte interface, permettant de connecter les différents composants, et du développement d’une station de charge.
* Un développeur Software : responsable de la programmation logicielle du robot et de l’IHM
* Une conceptrice FPGA : responsable de la programmation de la partie PL (Programmable Logic) du SoC. Il définit l’architecture firmware (architecture, conception, code, test, vérification et routage).

Durant ce stage, ELSYS DESIGN s’est placé dans le rôle du client souhaitant acquérir un robot de surveillance autonome qui serait capable d’effectuer des rondes à intervalle régulier, de détecter les anomalies comme les intrusions ou un départ de feu. Dans ce cadre, il nous a fallu nous occuper de la gestion de projet en plus de la partie technique. Il nous a été demandé de gérer l’intégralité du projet en cycle en V, de rédiger des spécifications et un planning, de gérer un budget ainsi que de mettre en place des réunions régulièrement pour les encadrants.

Le cahier de charges n’était pas prédéfini, mais quelques contraintes étaient imposées : il a fallu développer un robot à partir d’une base holonome déjà existante et le but final était d’avoir un robot totalement autonome : il devait pouvoir se localiser mais aussi naviguer de manière autonome tout en évitant les obstacles qui pourraient se trouver sur son chemin. De plus, le développeur software n'avait pas le droit d’utiliser un ROS (Robot Operating System) pour contrôler le robot.

# Spécifications et architecture

## Présentation de l’existant

Afin de développer ce robot, une base holonome déjà existante a été utilisée. Celle-ci a été réalisée lors d’une alternance en 2019. Elle se compose d’un châssis en métal, de quatre moteurs et de quatre roues permettant un déplacement latéral et horizontal et un STM32F767 responsable de l’asservissement des moteurs, et grâce auquel un simple envoi de consignes à cette carte permet le déplacement du robot.

Figure : La base Holonome

Par ailleurs, même si cette année le point de départ de ce stage était uniquement la base holonome, il était possible de réutiliser le matériel des années précédentes, ce qui nous donnait un accès à :

* Une imprimante 3D : pouvant servir à imprimer la nacelle de notre robot
* Du matériel électronique : des fers à souder, un oscilloscope, un analyseur logique …
* Quelques périphériques : un Lidar, un module GSM, ...

## Spécifications

Dès le début du projet, il était important de réfléchir à ce qu’on voulait faire et de rédiger un dossier de spécifications complet de ce que le robot pourra faire, et par quels moyens procéder pour atteindre ce but. Nous avons aussi dû réfléchir à déterminer quelles fonctions étaient essentielles et prioritaires pour le robot, et ainsi pouvoir définir quelles fonctions étaient secondaires et non obligatoires pour que notre produit soit viable. Un poids a été fixé pour chaque fonctionnalité du robot pour pouvoir établir une liste de priorités.

Dans ce but, nous avions estimé que le minimum viable pour notre robot serait qu’il puisse se déplacer en autonomie ou non, se localiser, qu’il puisse éviter les obstacles et qu’il soit accessible à distance par une Interface Homme-Machine (IHM).

Par ailleurs, nous avons décidé des moyens que nous allions utiliser pour les différentes fonctions du robot, notamment pour la localisation et l’évitement d’obstacles. Nous avons décidé d’utiliser l’IMU et l’odométrie en combinaison avec des tags RFID et l’odométrie pour la localisation. En effet, l’IMU permet de se localiser grâce aux déplacements en continu mais avec une erreur qui grandit au cours des déplacements, tout comme l’odométrie. Les tags RFID répartis partout dans l’agence permettent de se repositionner, et ainsi diminuer les erreurs de localisation. Il a aussi été décidé de déterminer le trajet à effectuer à l’aide d’un algorithme calculant le plus court chemin : l’algorithme de Dijkstra. D’autre part, l’utilisation d’ultrasons pour la détection d’obstacles paraissait être le choix le plus cohérent.

Ensuite, il a fallu réfléchir au partage de l’architecture entre la partie software et FPGA et à la répartition des tâches entre les deux. Dans un premier temps, l’idée était de partir sur de l’acquisition et du traitement d’images en FPGA. Cependant, ce choix d’architecture présentait le risque d’être très chronophage sur FPGA, notamment du fait de l’interface MIPI et du DMA qu’il aurait fallu faire pour envoyer l’image au Software. Le développement de ce bloc vidéo aurait constitué la majeure partie du stage et n’aurait pas laissé beaucoup de temps pour développer d’autres fonctionnalités. De plus, comme cette année le développeur Software ne pouvait pas utiliser de ROS (Robot Operating System), il a eu besoin de plus de temps pour effectuer certaines tâches par rapport aux années précédentes. Par conséquent, il a été décidé de ne pas faire d’acquisition vidéo sur FPGA et d’implémenter plutôt des blocs d’interface et de calcul destinés à soulager le software pour lui permettre de gérer la navigation.

De ce fait, le FPGA a été utilisé pour la gestion de divers périphériques comme l’IMU qui communique en I2C, du lecteur RFID qui communique en UART. Il a aussi été utilisé pour la gestion des ultrasons, de l’algorithme de Dijkstra, des balises et d’un minuteur.

## Choix de la carte

Les années précédentes, les stagiaires avaient utilisé un Ultrascale 96. Cependant, du fait de ruptures de stocks, il y avait 52 semaines de temps de livraison et nous avons donc dû changer de carte. Il y avait diverses contraintes à ce niveau-là, il fallait que le FPGA soit de Xilinx pour être compatible avec Vivado, majoritairement utilisé par l’entreprise. Il y avait aussi des limitations de budget : la carte ne devait pas dépasser les 500 euros de budget afin de pouvoir acheter des composants supplémentaires et faire fabriquer notre carte d’interface.

Notre choix s’est finalement porté sur une carte Zybo Z7-20 qui présente une connectique assez complète (entrées-sorties audio, MIPI, ports HDMI, USB, 6 PMODS sur lesquels peut se connecter la carte interface) et offrait une puissance suffisante pour le projet. La puce intègre un processeur dual-core ARM Cortex-A9 qui constitue la partie software (PS) et un FPGA Artix-7 pour la partie logique (PL). Les différentes spécifications de la Zybo sont détaillées dans le tableau ci-dessous.

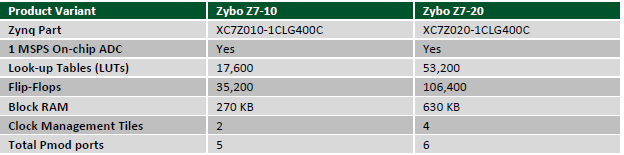


Figure : Spécifications de la Zybo Z7-20

## Architecture

Une fois toutes les spécifications fixées, nous avons réalisé l’architecture fonctionnelle de notre système sous forme de schéma blocs en représentant le partage entre Software et FPGA.

Sur le schéma ci-dessous, se trouvent différents protocoles de communication, les différents périphériques connectés aux PMODs de la Zybo ainsi que les différents programmes à mettre en place… Il y a certains périphériques pour lesquels nous avions laissé la place. Comme par exemple le gestionnaire de chargement mais qui n’a pas pu être implémenté par manque de temps. Ces modules n’étaient pas critiques pour le bon fonctionnement de notre robot.

Pour la communication entre le FPGA et le Processeur, le choix le plus logique était d’utiliser l’outil de création d’IP AXI de Vivado. Cet outil permet de mettre la mise en place d’un lien AXI 4 entre les 2 parties. C’est un protocole de bus de communication développé par ARM. Dans le cadre de ce projet, seulement le protocole AXI 4 Lite (Advanced eXtensible Interface 4 Lite) a été utilisé. L’avantage de cette interface Lite par rapport à l’interface normale est qu’elle est plus simple à mettre en œuvre. L’outil Vivado permet d’instancier facilement dans le FPGA divers registres accessibles depuis l’AXI et donc par le processeur pour échanger des données.

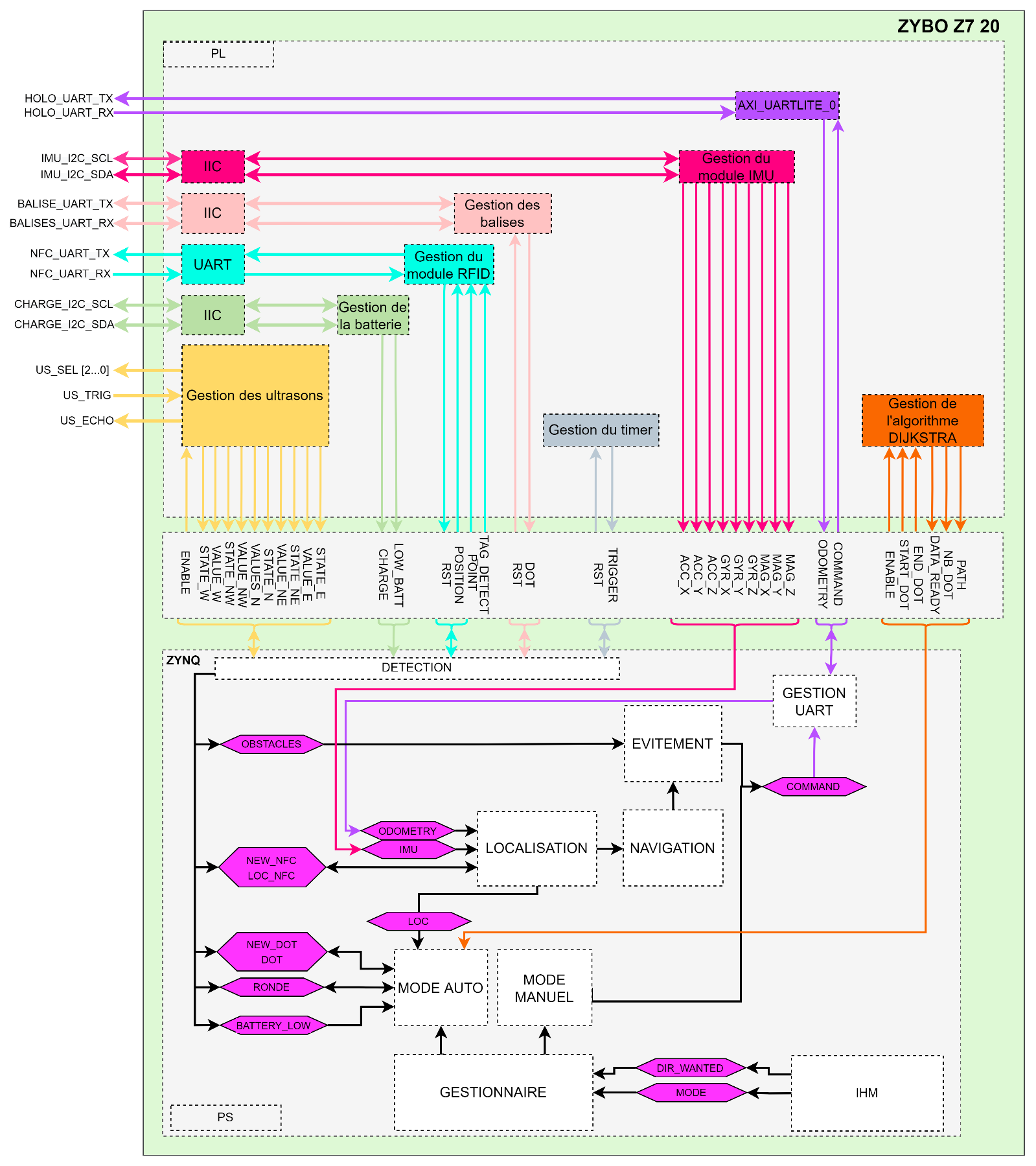


Figure : Architecture de notre système

Voici le block design complet de la partie FPGA :

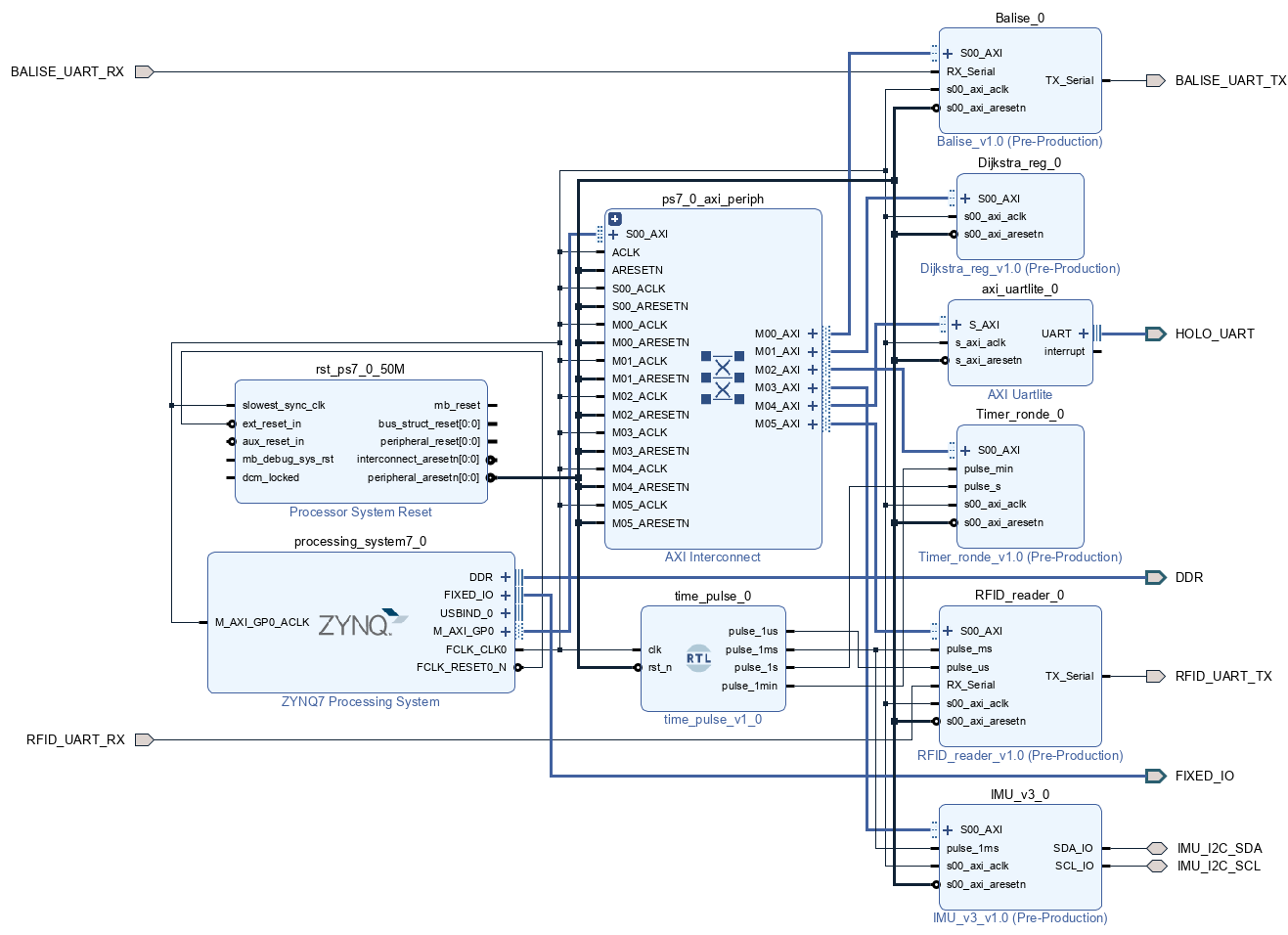


Figure : Block design de la partie FPGA

On peut remarquer qu’il n’y a pas de bloc ultrason sur ce schéma malgré son importance pour pouvoir éviter les obstacles. Les ultrasons n’ont pas pu être implémentés au niveau de la partie Software à cause d’un manque de temps. D’une part, le délai de livraison de la carte interface a été plus long que prévu, or cette dernière était nécessaire à la validation de la version définitive des ultrasons. D’autre part, du retard a été pris sur la partie software et par conséquent, le stagiaire qui s’occupait de cette partie-là n’a pas eu le temps d’implémenter toutes les fonctions nécessaires.

# Gestion de projet

## Diagramme de GanTT

Dès le début de ce projet, un diagramme de GanTT a été réalisé afin d’établir le temps nécessaire pour chaque tâche et ainsi évaluer le chemin critique du projet, c’est-à-dire la suite la plus longue d’activités qui pourrait affecter le temps nécessaire pour le projet. Mais en début de stage, il était assez difficile de déterminer le temps nécessaire pour chaque tâche ce qui nous a amené à modifier le GanTT tout au long du projet.

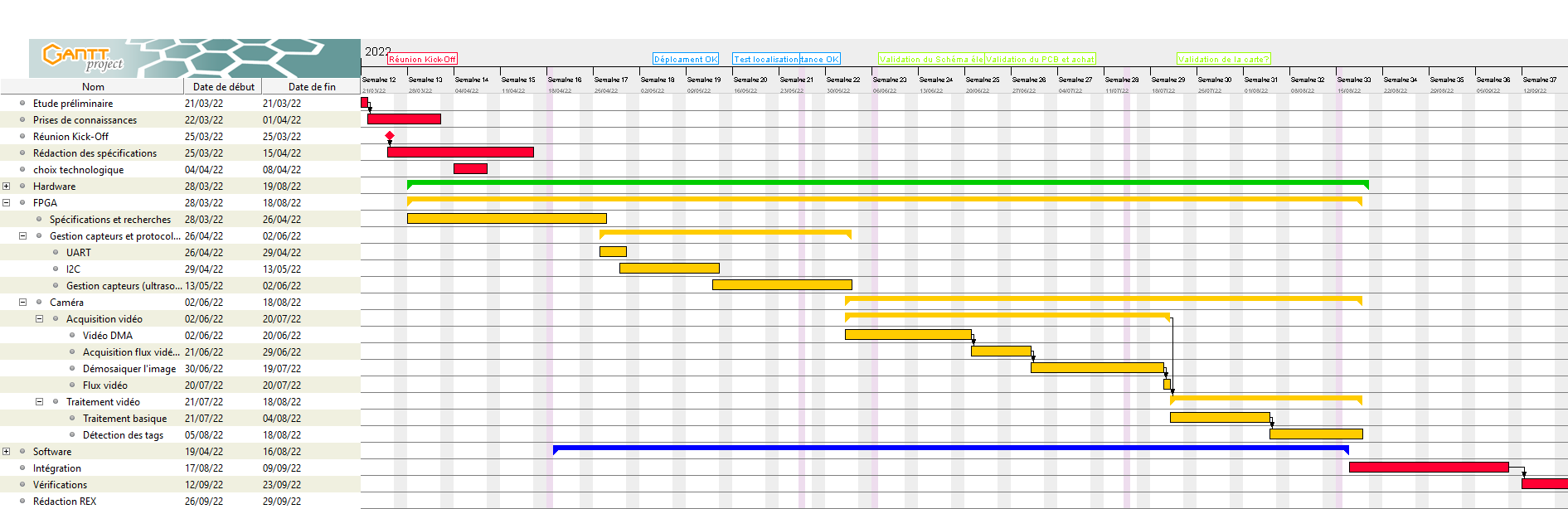
De plus, en début de projet, il était prévu que la partie FPGA fasse de l’acquisition et du traitement d’image. Comme cela a changé par la suite, le planning a dû être refait complètement au bout d’un mois de projet avec les nouvelles fonctions du Software délocalisées sur le FPGA. Par ailleurs, les quelques soucis de livraisons qui ont eu lieu durant ces 6 mois de stage ont occasionné des retards sur certaines parties du projet.

Figure : GanTT en début de projet

Nous avions ce diagramme de GanTT en début de projet :

Et celui-là en fin de projet :

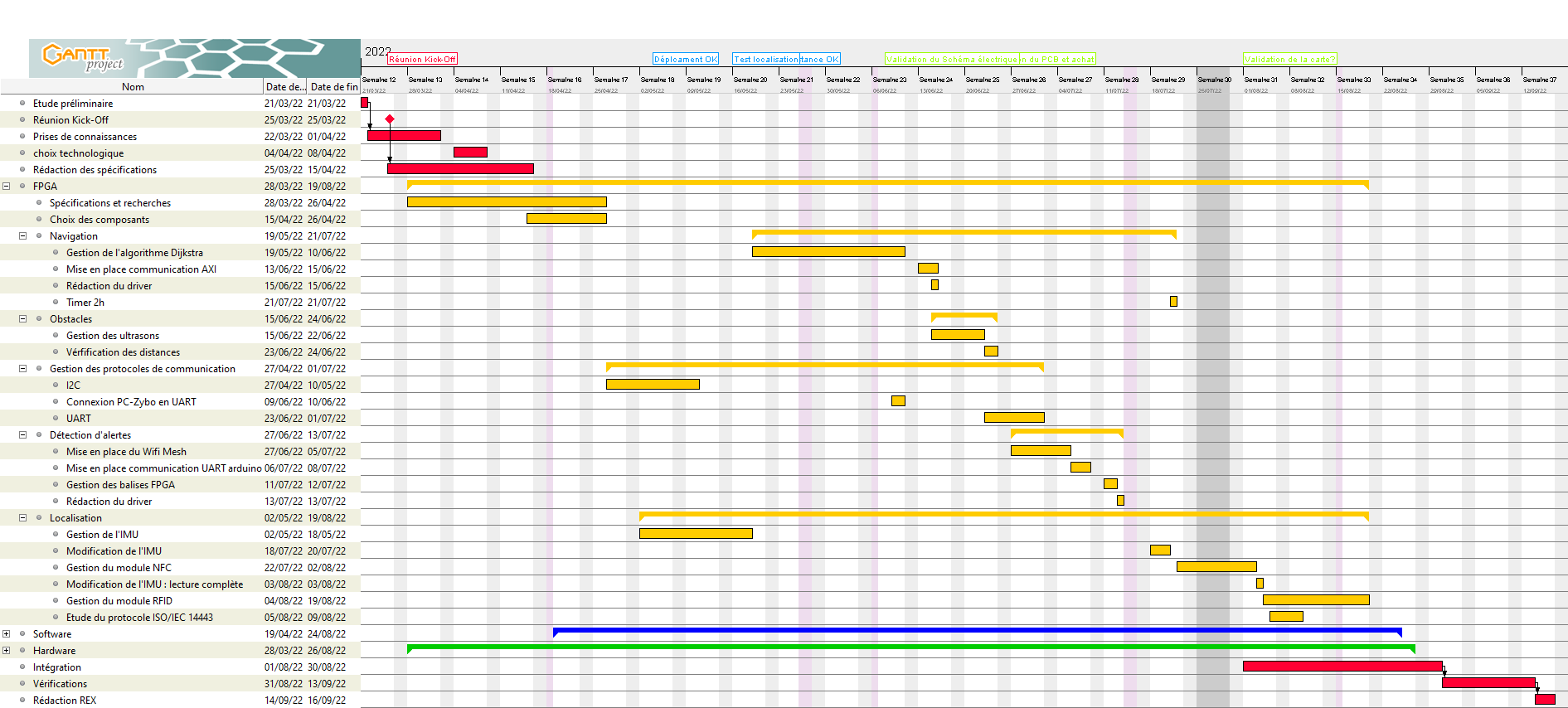


Figure : GanTT en fin de projet

On peut observer que la partie spécifications a pris beaucoup plus de temps au début, en effet, on nous a demandé de bien développer les spécifications afin d’être sûr des choix avant de commander la carte et les divers périphériques. C’est essentiellement le choix de la méthode de localisation qui a été complexe. Nous voulions une méthode assez précise mais qui ne nécessiterait pas le Lidar car il aurait été compliqué à mettre en place sans le ROS.

D’autre part, pour la partie FPGA, beaucoup de modules ont été effectués dans un ordre différent de celui qui avait été initialement prévu car des difficultés ont été rencontrées en ce qui concerne les commandes. Par conséquent, le travail sur la partie NFC n’a pu commencer que très tard à cause des problèmes de livraison du périphérique. Sans compter que, assez rapidement après avoir entamé le travail sur ce module, il paraissait clair que le module NFC était passif et qu’il ne pourrait donc pas lire de tags passifs. C’est pourquoi il a été nécessaire de commander un autre module, un module RFID compatible avec les tags.

Finalement, la partie intégration avec le software a été faite au fur et à mesure durant les mois de Juillet-Août. En effet cela permettait dans un premier temps de tester les modules et la bonne communication entre le FPGA et le software. Dans un second temps, cela permettait au développeur Software d’avoir des modules autour desquels construire son programme et faire ses tests, notamment au niveau des valeurs de l’IMU par exemple où il a été décidé après intégration de rajouter plusieurs lectures.

## Gestion du budget

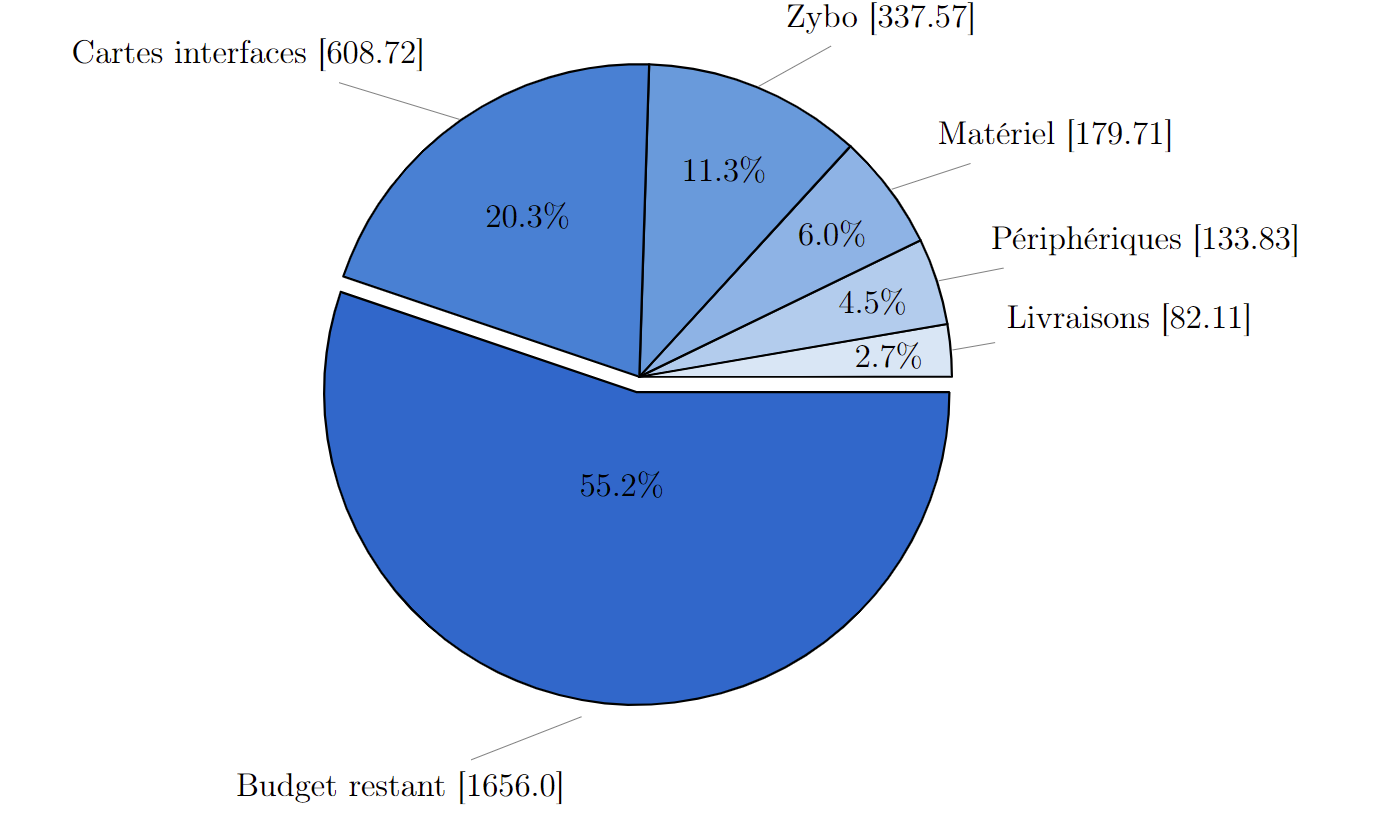
Le budget alloué pour ce stage était de 1000 euros par stagiaire, ce qui faisait donc un budget global de 3000 euros pour le projet. Ce budget a été utilisé de la manière suivante :

Figure : Gestion du budget

A la fin du stage, on peut voir qu’il restait environ la moitié du budget. On peut observer que la moitié des dépenses est liée à la fabrication de la carte interface même si le budget utilisé pour celle-ci est inférieur à ce qui était prévu en début de stage (1000 euros). Nous avons pris soin de garder une certaine marge pour éventuellement racheter des périphériques. Il était également prévu de faire fabriquer une carte pour permettre la recharge du robot par induction. Celle-ci a été conçue et simulée mais, malheureusement, le manque de temps a fait qu’il n’a pas été possible de la faire imprimer avant la fin du stage.

# Travail réalisé

## Le générateur d’impulsions

Pour ce projet, notamment pour le gestionnaire de l’IMU mais aussi les ultrasons ainsi que pour le lecteur RFID, il était nécessaire de mesurer diverses durées de manière plus ou moins précise. De ce fait, pour éviter d’avoir à mettre en place plusieurs diviseurs d’horloges avec un grand nombre de compteurs dans chaque module, la meilleure solution trouvée était de faire un bloc simple qui sort des impulsions de manière régulière pour des durées définies entre deux impulsions.

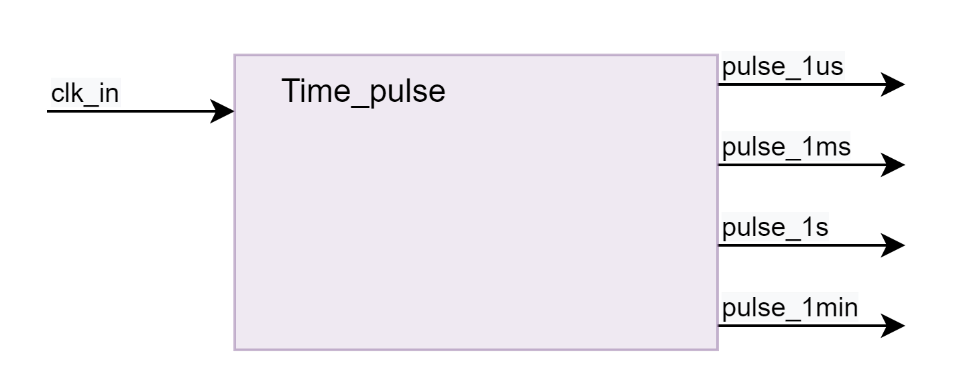
Plusieurs compteurs ont donc été réalisés. Ceux-ci permettent de générer les différentes pulses à partir de l’horloge en entrée. Le compteur de millisecondes dépend du compteur microsecondes, il ne s’incrémente que toutes les 1 microsecondes afin d’optimiser le nombre et la taille des compteurs. Les compteurs des secondes et des minutes s’appuient sur ce même principe.

Figure : Schéma du bloc générateur d'impulsions

## Gestion de l’IMU (Inertial Measurement Unit)

Le premier module réalisé a été le gestionnaire de l’IMU (Inertial Measurement Unit). Ce périphérique permet de mesurer notamment la vitesse angulaire, l’accélération, et l’orientation du capteur à l’aide d’une combinaison d’accéléromètre, gyroscope et de magnétomètre. C’est un élément très important pour le robot sachant que ce périphérique devra nous servir pour la localisation en complément avec le module RFID et l’odométrie. Ce module communique en I2C et comme il est géré par le FPGA directement, il était impossible d’utiliser l’IP I2C de Vivado directement. Il a donc fallu utiliser le code d’un I2C réalisé par un stagiaire l’année dernière. Ce module n’était pas totalement fonctionnel et a dû être repris avant de pouvoir commencer le code du Dijkstra.

### Mise en place de la communication I2C

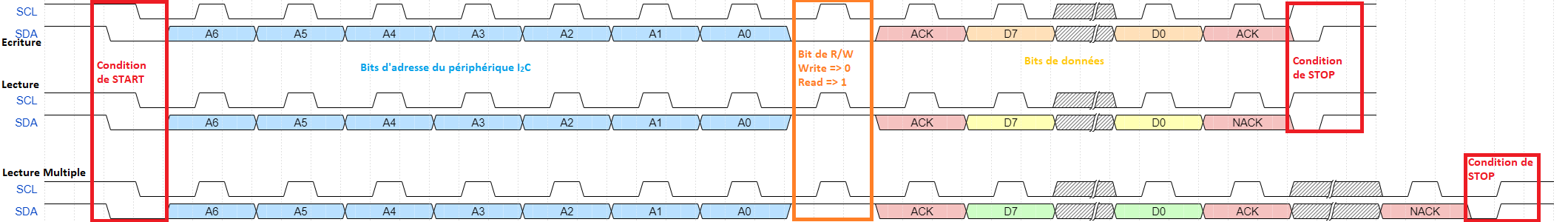
Le protocole I2C est un protocole très utilisé [1], il s’appuie sur deux lignes de communication : le SCL, qui sert de ligne d’horloge bidirectionnelle et le SDA qui sert de ligne de données bidirectionnelle. La communication se fait toujours entre un maître et un ou plusieurs esclaves et est toujours initiée par le maître à l’esclave. Les deux lignes sont tirées au niveau de tension VDD à travers des résistances de pull-up : l’état de repos des deux lignes est donc l’état haut.

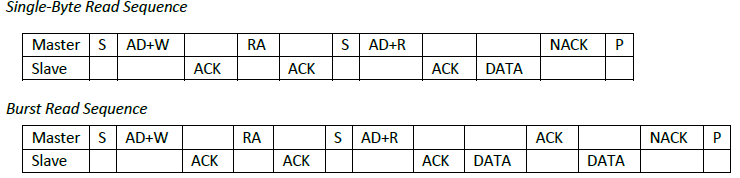
Figure : Diagramme des timings I2C

Le message s’envoie suivant des timings précis, en partant d’une condition de Start puis émission des sept bits d’adresse par le maître, plus le bit de Read/Write à ‘1’ ou à ‘0’. Ensuite arrive la réponse de l’esclave par un bit d’acquittement (ACK) en cas de bonne réception ou de non-acquittement (NACK) si mauvaise réception. Après acquittement il y aura, selon le bit de R/W, soit un envoi de données par le maître soit par l’esclave, avec à la fin un bit d’acquittement de l’esclave en cas d’écriture, soit un bit acquittement ou non par le maître en cas de lecture. Le maître envoie un non acquittement pour mettre fin à la lecture. Le principe est résumé dans l’image précédente.

La condition de Start apparait lorsque la ligne SDA est ramenée à un niveau bas alors que la ligne SCL est à niveau haut. Dans le même principe, la condition de stop apparait lorsque la ligne SDA est ramenée à niveau haut lorsque la ligne SCL est à niveau haut. Un bit vaut ‘1’ lorsque la ligne SDA est à un niveau haut pendant un coup d’horloge de SCL et inversement, un bit vaut ’0’ lorsque la lige SDA est à un niveau bas pendant un coup d’horloge de SCL.

Dans tous les cas, pour effectuer une lecture d’un registre du périphérique choisi, il faut procéder en deux étapes :

* Il faut tout d’abord envoyer l’adresse du périphérique avec un bit d’écriture puis envoyer l’adresse du registre dans lequel on veut écrire depuis le maître. Cette adresse fait souvent 8 bits.
* Une fois tous les bits d’adresse envoyés, on met fin à l’écriture et on passe en lecture. On renvoie l’adresse du périphérique avec un bit de lecture puis on lit le registre. Il est aussi possible de lire plusieurs registres à la suite.

Le principe est résumé ci-dessous :

*S : Start*

*AD : Adresse périphérique*

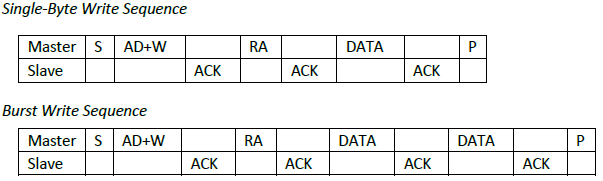
*RA : Adresse du registre*

*W : bit d’écriture*

*R : Bit de lecture*

*P : Stop*

De même, il peut être souvent nécessaire d’envoyer l’adresse du périphérique avec un bit d’écriture puis l’adresse du registre avant de procéder à l’écriture du registre choisi. Il est également possible d’écrire plusieurs registres à la suite.

*S : Start*

*AD : Adresse périphérique*

*RA : Adresse du registre*

*W : bit d’écriture*

*P : Stop*

### Développement du gestionnaire d’IMU

Une fois l’I2C fonctionnel, il a été possible de commencer le développement du gestionnaire de l’IMU. C’est l’IMU de l’année précédente qui a été réutilisé pour le projet cette année. Cette IMU fonctionnait à l’aide de 2 composants : un pour mesurer accélération et la vitesse angulaire, l’autre pour mesurer le champ magnétique. Il a donc fallu prendre en compte le fait qu’il y avait deux adresses I2C distinctes pour les deux périphériques. Par conséquent, il a été nécessaire de faire l’initialisation du composant en deux temps avant de passer à la lecture des registres qui s’effectue également en deux temps. Comme le périphérique de l’année précédente a été réutilisé, il a été possible de récupérer la valeur des registres à écrire et les registres à lire pour chaque composant. Cela a beaucoup accéléré la prise en main de ce périphérique même s’il a tout de même fallu étudier la datasheet pour comprendre sous quel format le périphérique envoie ses données.

L’initialisation ne se fait qu’au démarrage du robot, le bloc entre ensuite dans une routine de lecture périodique des registres. Pour le premier composant, il fallait effectuer six lectures dont quatre à la suite (l’accélération en x et en y, lecture de deux registres par axe) puis deux autres à la suite (la vitesse angulaire en z, lecture de deux registres par axe). Pour le second, il fallait sept lectures dont six à la suite (trois axes du gyroscope, lecture de deux registres par axe) et un pour mettre fin à la lecture. Entre deux lectures, le choix a été de laisser passer 10 ms car un des deux composants ne se réactualisait que toutes les 10 ms. Sans compter que, du fait de la vitesse assez faible du robot, il n’était pas utile de lire les différentes valeurs en continu. Pour mesurer le délai, le générateur d’impulsions présenté précédemment a été utilisé en complément avec un compteur d’impulsions.

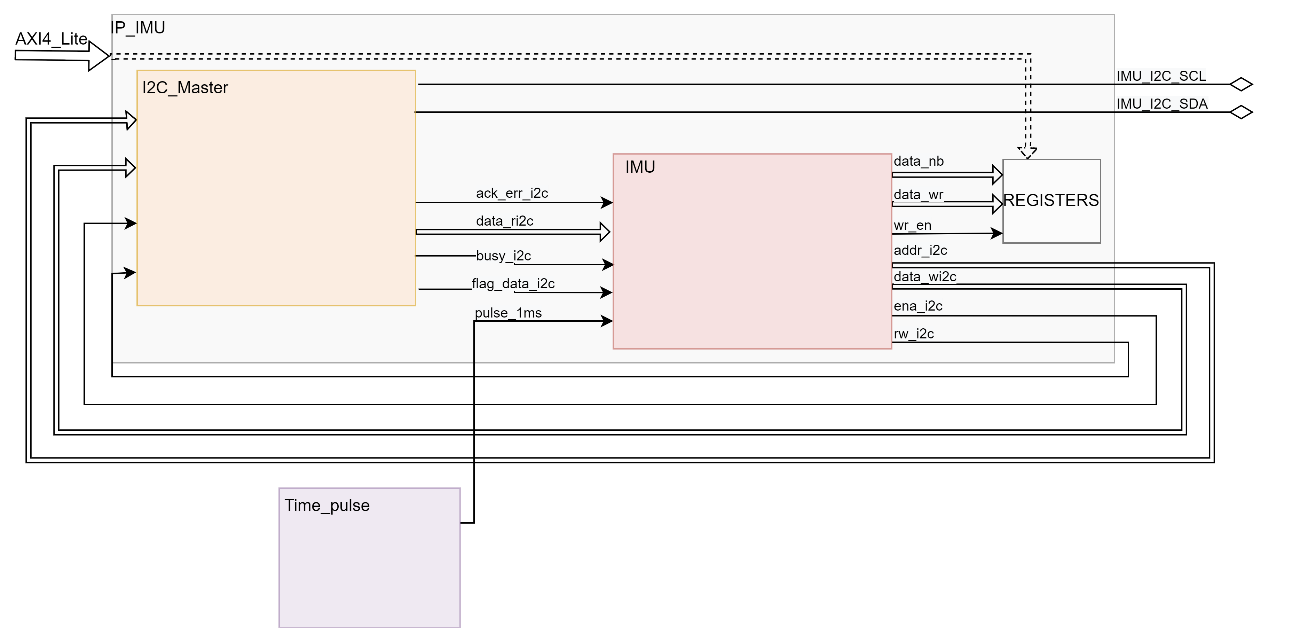
Une fois le code rédigé, simulé puis testé, il a fallu rendre le bloc accessible depuis le software : en effet, lors de ces lectures, le FPGA récupère des données que le software devra lire ensuite. De ce fait, l’utilisation du protocole de communication AXI 4 lite était nécessaire. Ce protocole permet d’écrire directement dans des registres accessibles par le processeur. Le schéma bloc ci-dessous a été obtenu.

Figure : Schéma bloc du gestionnaire d'IMU

La communication avec le processeur a été testée dans un premier temps avec Vitis. Un driver python a ensuite été rédigé afin que le développeur software puisse directement utiliser l’IMU depuis le PYNQ qu’il a mis en place. PYNQ est un cadre de développement, qui permet aux programmeurs d’exploiter de manière simplifiée les plateformes Xilinx. Il permet d’exploiter les avantages de la logique programmable et des microprocesseurs et nécessite l’utilisation du langage Python et diverses librairies.

Nous nous sommes assez vite aperçus qu’un des deux composants ne mettait pas à jour les valeurs correctement et nous renvoyait donc constamment les mêmes valeurs. Nous n’avons pas trouvé la raison de ce problème mais nous nous sommes aperçus qu’en repassant dans l’initialisation de ce composant avant la prochaine lecture, les valeurs de l’IMU se mettaient à jour. Le code a donc été modifié pour que l’initialisation de l’ICM20600 se fasse de manière régulière. Cette modification a résolu le problème.

D’autre part, nous avons décidé ultérieurement qu’il serait intéressant d’avoir toutes les valeurs d’accéléromètre, de vitesse angulaire et de gyroscope. De ce fait, le code a une nouvelle fois été modifié pour répondre à ce besoin : il y a donc au total douze lectures dans les registres du premier composant (six pour l’accéléromètre et six pour la vitesse angulaire avec toujours deux registres à lire par axe) et sept pour le second.

## Algorithme de Dijkstra

L’algorithme de Dijkstra [2][3][4] permet de résoudre le problème du plus court chemin en théorie des graphes. Dans le cadre de ce projet, il permet de déterminer le plus court chemin pour aller d’un bout à l’autre de l’agence.

Cet algorithme prend en entrée un graphe pondéré avec le poids de chaque branche proportionnelle à la distance. Il s’agit de construire un sous-graphe dans lequel sont classés les différents sommets en fonction de leur distance avec le point de départ. Au départ, on considère que la distance au sommet de départ est nulle et que les autres distances depuis le sommet de départ sont infinies. A chaque itération, nous allons nous appuyer sur le graphe pondéré, mettre à jour le sous-graphe pondéré avec les sommets accessibles depuis notre sommet actuel et déterminer quel sommet est le plus proche du sommet de départ sans repasser par des sommets déjà visités. On continue jusqu’à avoir visité tous les sommets ou jusqu’à être arrivé au sommet visé.

Cet algorithme a nécessité quelques réflexions au niveau de la représentation du plan en FPGA. Outre le fait qu’il faille l’implémenter en binaire, il était important de prendre en compte la mémoire qu’allait utiliser cet algorithme : l’exécution de cet algorithme sur un FPGA est plus rapide que lorsqu’elle est effectuée par le processeur mais cela utilise beaucoup de mémoire, ressource souvent limitée dans un FPGA.

### Le plan

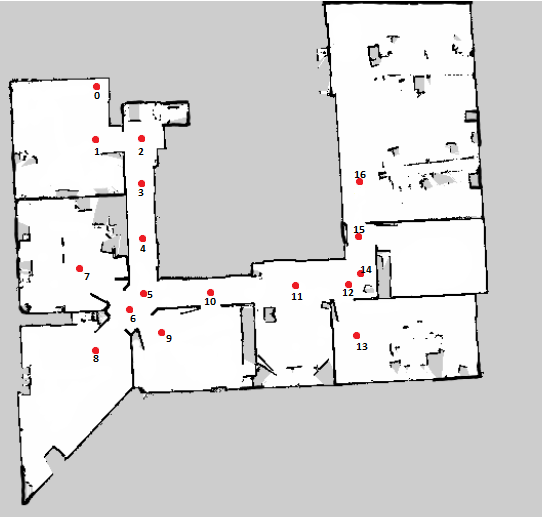
Dans un premier temps, un plan d’agence a été repris (celui-ci a été réalisé par le développeur software l’année précédente) et un certain nombre de points ont été placés dessus. Ces points ont servi comme sommets dans l’algorithme de Dijkstra. Ensuite, pour représenter les connexions entre chaque sommet, autant d’adresses ont été associées à chaque sommet que de sommets étaient accessibles par celui-ci. Entre chaque sommet, la distance a été représentée par un poids de 1 (un seul chemin est possible pour aller d’un point à un autre), à côté de cette distance a été stocké le sommet de destination. Beaucoup de sommets n’ont que deux connexions mais certains, comme le sommet 6 en ont plus (4). Pour stocker les adresses et le nombre de sommets accessibles par ce sommet, un package VHDL a été utilisé. Chaque connexion est donc représentée dans 2 sens. Le tout est stocké dans une ROM.

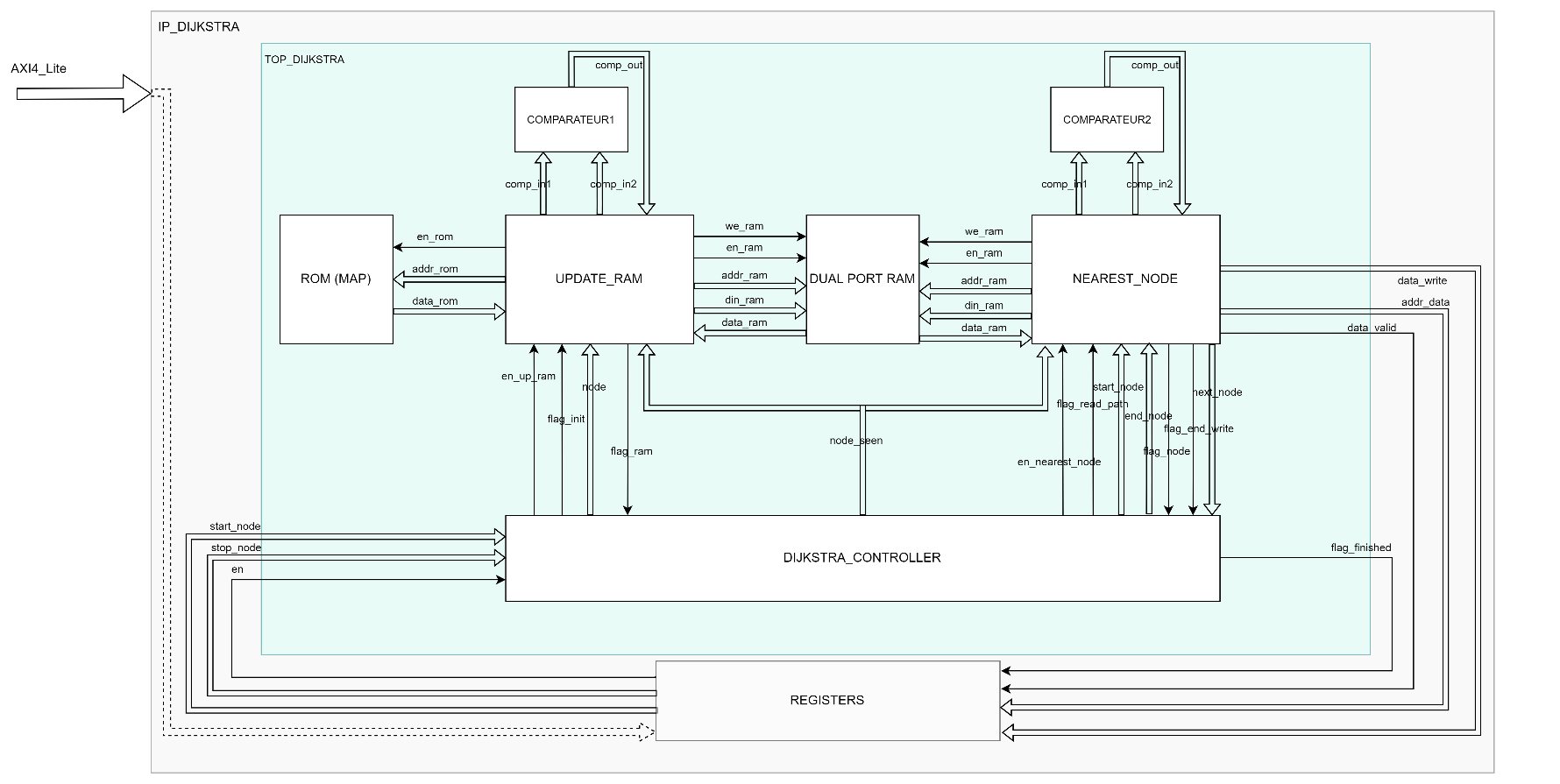
Figure : Plan d'agence pour l'algorithme de Dijkstra

### L’algorithme

Dans le but de développer l’algorithme décrit plus haut en FPGA, la structure du Dijkstra s’est faite autour de 3 blocs principaux : un bloc qui permettait de mettre à jour une RAM qui sert de stockage temporaire du sous-graphe *(UPDATE\_RAM)*, un bloc qui déterminait quel point est le plus proche parmi ceux qui n’ont pas été visités et qui lisait le chemin final *(NEAREST\_NODE)* ainsi qu’un bloc contrôleur qui permettait de gérer les différentes transitions entre les 2 blocs précédents *(DIJKSTRA\_CONTROLLER)*. Deux blocs comparateurs ont été développés. Le comparateur 1 est nécessaire pour déterminer quel est le sommet le plus proche du sommet actuel parmi ceux envoyés par le block *UPDATE\_RAM*. Le comparateur 2 permet de déterminer le chemin le plus court parmi ceux envoyés par le bloc *NEAREST\_NODE*. Une fois que l’algorithme arrivait au sommet final, le bloc *DIJKSTRA\_CONTROLLER* envoyait une commande de lecture au block *NEAREST\_NODE* qui allait ensuite lire et transmettre sommet par sommet le chemin à effectuer en partant du point d’arrivé.

Tout comme pour l’IMU, il a fallu rendre accessible les données générées par le software en utilisant des registres, un AXI 4 lite et mettre en place un driver Python.

Figure : Schéma block pour l'algorithme de Dijkstra



## Gestion des ultrasons

Nous avions décidé que pour la détection d’obstacles nous allions utiliser des ultrasons. Notre choix s’est porté sur le composant HC-SR04. Le principe de fonctionnement consiste en l’envoi d’un signal trigger (signal à niveau haut) pendant au moins 10 µs. L’ultrason va ensuite émettre des impulsions ultrasoniques pour évaluer les distances avec on environnement via les échos reçus. générer une sortie sous forme d’impulsion électrique dont la largeur est proportionnelle à la distance aller/retour. La distance vaut donc :

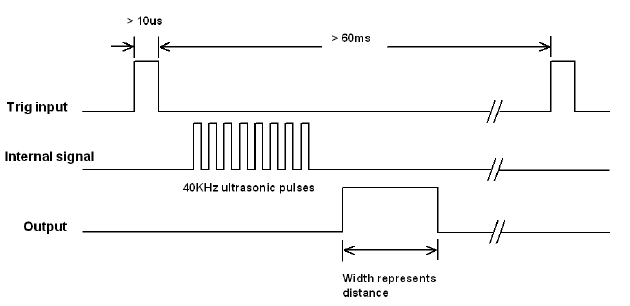


Figure : Principe de fonctionnement des ultrasons

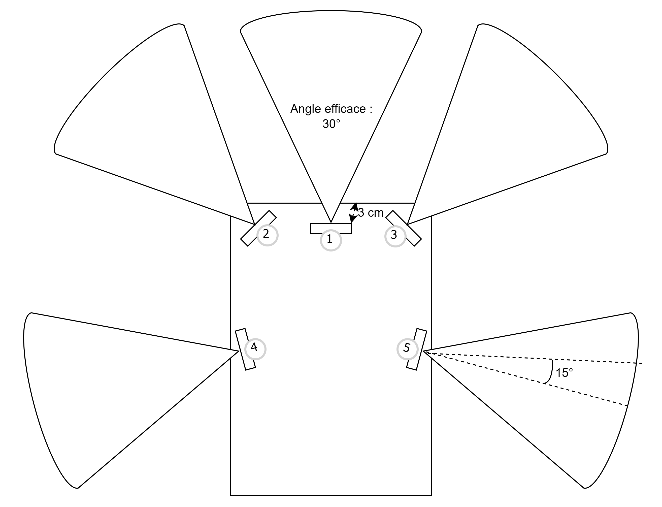
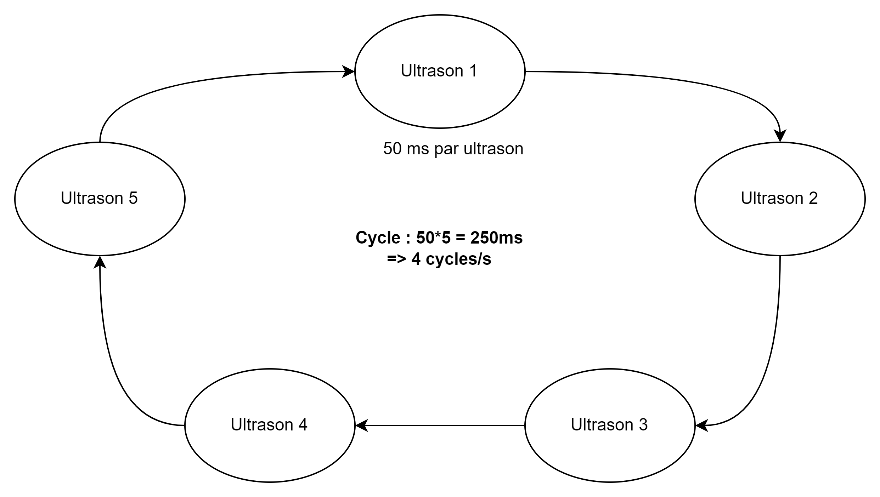
Nous avons décidé d’utiliser cinq ultrasons afin de pouvoir gérer la détection sur plusieurs faces du robot représenté sur le schéma ci-dessous. Il n’y en a pas sur la face arrière du robot car nous ne ferons que très peu de marche arrière en principe. En somme, il faut prendre en compte le fait que les ultrasons peuvent interférer entre eux. Ainsi, un ultrason peut détecter les ondes d’un autre, c’est pourquoi il a été décidé de faire fonctionner les ultrasons les uns après les autres suivant un cycle prédéfini. Le passage d’un ultrason à un autre est géré par un multiplexeur et un démultiplexeur sur la carte interface.

Figure : Cycle de fonctionnement des ultrasons

Figure : Emplacement des ultrasons sur le robot

Dans ce but, divers compteurs ont été mis en place. Ils permettent de compter le nombre d’impulsions générées par le bloc générateur d’impulsions dédié. Il fallait un compteur pour la durée du trigger, un pour déterminer la durée de l’impulsion reçue et un troisième pour déterminer quand passer à l’ultrason suivant. Un bloc compteur dédié a donc été réalisé et dupliqué trois fois, une par utilisation. Ces blocs ont ensuite été intégrés dans le bloc ultrason.

Avec le développeur software, nous avons décidé qu’un objet était détecté si la distance par rapport au robot est inférieure à 100 cm. De plus, nous avons décidé que trois données allaient être transmises pour chaque ultrason :

* La présence d’un objet ou non à moins de 100 cm
* Une donnée de zone : pour chaque ultrason, trois zones ont été définies. Se situant entre trois intervalles de distance, les zones ne sont pas forcément les mêmes pour les différents ultrasons.
* La distance de l’objet

Le diagramme d’état suivant a été obtenu. Il s’agit d’un diagramme de Mealy avec les changements des sorties sur les transitions. Cette machine d’état a été codée en 1 process et l’état buffer a été rajouté pour que la sortie reste constante pendant un coup d’horloge supplémentaire.

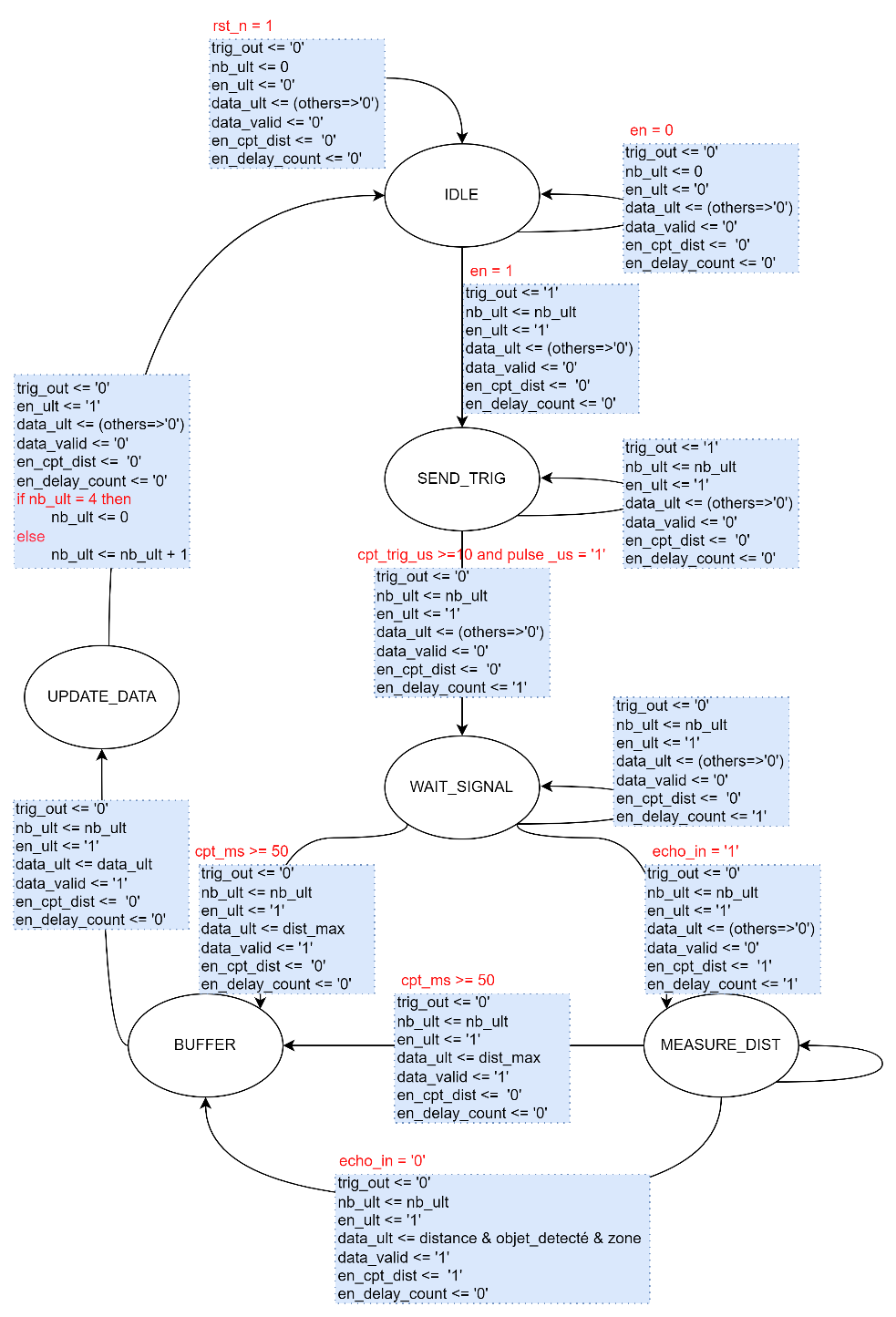


Figure : Diagramme d'état du contrôleur d'ultrasons

Comme la carte interface n’est arrivée que très tardivement et qu’elle était requise pour pouvoir effectuer les tests avec les cinq ultrasons, il n’a été possible de tester le code qu’avec un seul ultrason. Pour vérifier les distances mesurées par rapport aux distances réelles, une communication UART a été mise en place directement depuis le FPGA vers le PC et qui envoie les distances mesurées à intervalle régulier : les distances reçues étaient cohérentes avec la réalité. Le principe du protocole UART est détaillé dans une autre partie.

Malgré tout, une communication AXI 4 Lite a été mise en place pour me permettre de tester le code et mettre en place rapidement le module pour une utilisation par le software.

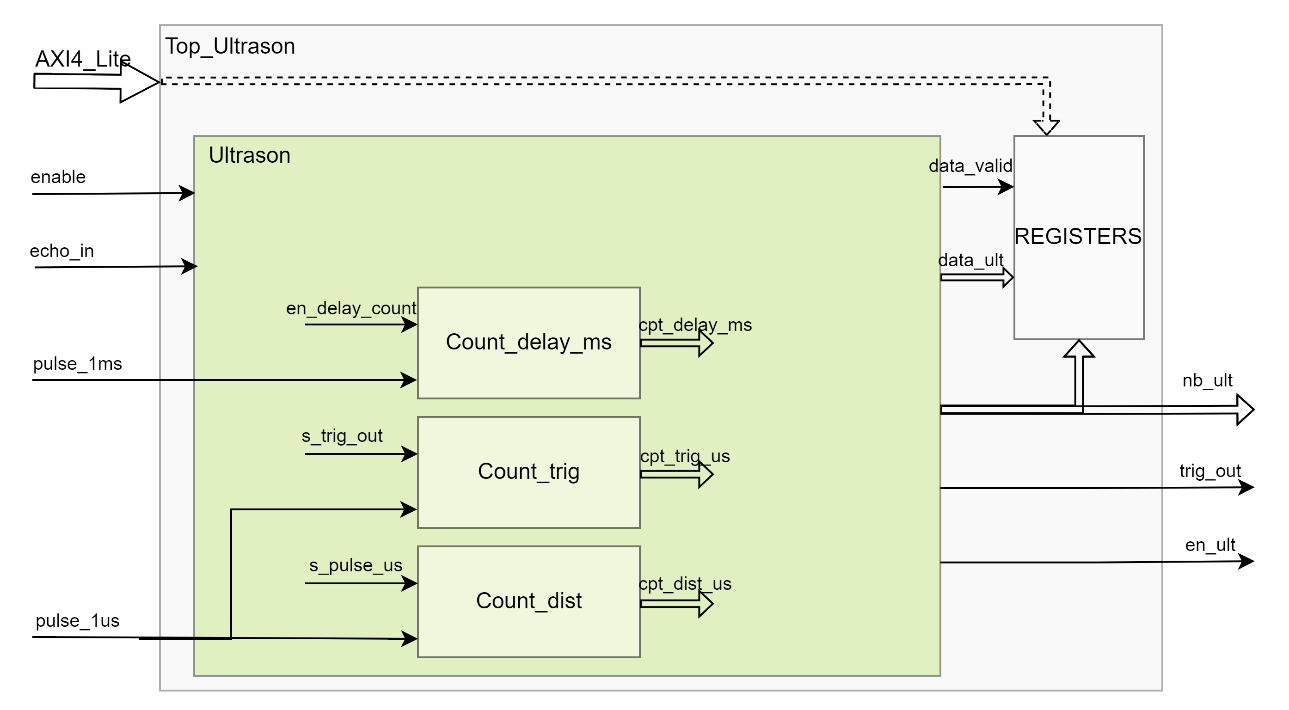


Figure : Schéma block du gestionnaire des ultrasons

## Gestion du RFID

Le module RFID est un module important pour le bon fonctionnement du robot car il va, de manière complémentaire avec l’IMU et l’odométrie, permettre de localiser le robot.

Avant de pouvoir développer un code sur Vivado, il était important de savoir ce qu’il fallait envoyer au module pour communiquer avec mais surtout il était important de comprendre comment établir une communication avec le tag. Dans ce but, il était intéressant de travailler dans un premier temps avec Arduino car cela me permettait de tester des commandes de manière efficace.

### Mise en place de la communication

#### Communication avec le tag RFID click

Pour communiquer avec le tag RFID click, la méthode est relativement simple : il faut envoyer un numéro de commande, la longueur des données et les données qui vont avec. Les différentes commandes sont résumées dans le tableau. Dans mon cas, seules 2 commandes m’étaient utiles :

* *ProtocolSelect*: nécessaire pour initialiser le composant et choisir le protocole de communication avec les tags : ISO 14443 pour nos tags.
* *SendRecv* : Permet d’envoyer des commandes aux tags par le protocole précédemment sélectionné puis de renvoyer la réponse du tag.

Les autres commandes comme *IDN* et *Echo* m’ont été utiles notamment pour tester la bonne communication avec le module RFID afin de vérifier que la réponse était cohérente.

##### 

Figure : Liste des commandes du lecteur RFID

#### ISO 14443

ISO 14443 est une norme internationale qui définit les cartes de proximité utilisées pour l’identification et définit un protocole de transmission pour communiquer avec ces cartes. Il en existe deux types : type A et Type B. Ces deux types de carte communiquent à 13.56 MHz mais il existe des différences au niveau des procédures d’initialisation et des méthodes de modulation. Dans notre cas, on n’utilise que des cartes de type A. La portée du signal est relativement faible, de l’ordre de 4 cm au maximum.

Pour ce protocole, la communication se fait en diverses étapes :

* Dans un premier temps, on envoie une requête de type REQA (Request type A car les cartes sont de type A), et on attend une réponse de type ATQA (Answer to request type A). Cela permet d’identifier la variante et de débuter la communication.
* Ensuite, l’émetteur doit identifier le récepteur et activer la carte. Dans ce but, il faut procéder à une boucle d’anticollision. Celle-ci permet d’identifier l’ID du récepteur et de lier l’émetteur à un unique récepteur. En effet, dans le cas où il y a plusieurs récepteurs dans le champ magnétique du récepteur, il pourrait y avoir des interférences.
* Finalement, une fois le récepteur identifié et activé, on peut lire et écrire dans différents registres du récepteur. Dans mon cas, je ne vais effectuer qu’une lecture d’un seul registre dans lequel on aura stocké les données de localisation.

Tout ce protocole est détaillé dans l’image suivante, avec en orange les envois à effectuer depuis le système connecté au lecteur RFID et en bleu ce qu’on reçoit. Il y a aussi la sélection du protocole lecteur avant de pouvoir débuter quelconque communication avec des tags.

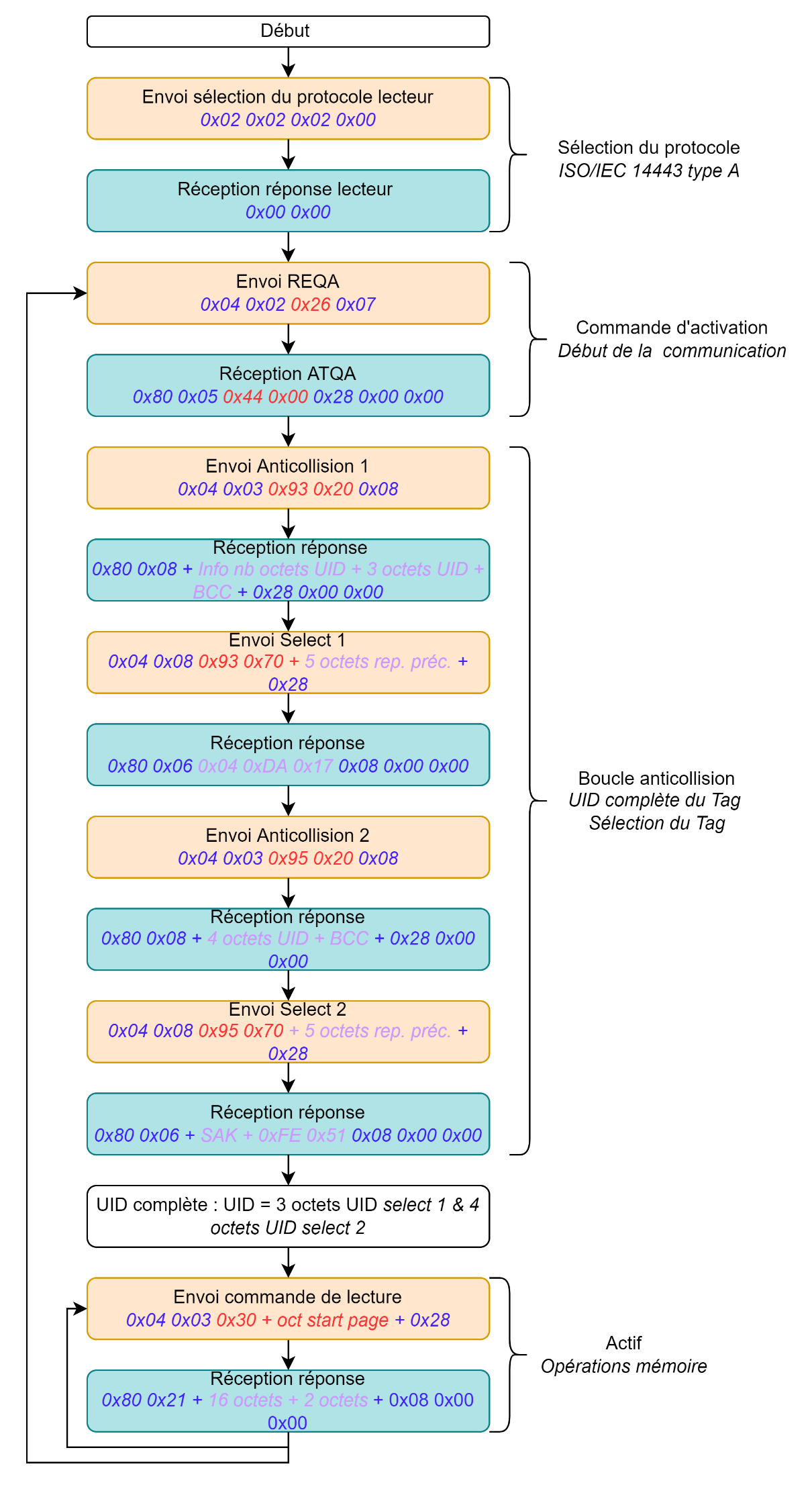


Figure : Protocole d'envoi et de réception des commandes pour le lecteur RFID

Chaque message se compose de 4 parties :

**Pour l'envoi** : N° commande pour le lecteur + longueur du message + data + flags de transmission

**Pour la réception :** Code résultat + longueur données restantes + données + ACK/ NACK sur 3 octets

Avec, au niveau des couleurs :

* En bleu : Commandes + data spécifiques du lecteur RFID
* En rouge : Commandes du protocole ISO 14443 type A
* En violet : Data Tag

### Développement de la partie FPGA

Une fois le protocole de communication défini, il était nécessaire de réfléchir à comment mettre cela en place en FPGA. Dans un premier temps, il a fallu se pencher sur le protocole de communication UART puis dans un second temps, le gestionnaire RFID a pu être développé.

#### Le protocole de communication UART

UART [8][9] signifie *Universal Asynchronous Receiver/Transmitter*. Son but principal est de transmettre et de recevoir des données série. Ce protocole n’utilise que deux fils pour transmettre les données entre les appareils. Il permet d’émettre et de recevoir dans les deux sens : les données circulent du TX de l’émetteur à la broche RX du récepteur de manière asynchrone. En effet, il n’y a pas de signal d’horloge pour synchroniser la sortie des bits de l’UART avec l’échantillonnage des bits par le récepteur : l’émetteur envoie les bits à une vitesse prédéfinie appelée *Baud*, cette vitesse permet d’avoir la même durée de bits. Les taux de *Baud* les plus courants sont : 4 800, 9 600, 19 200, 57 600 et 115 200. De plus, il y a des bits de début et de fin pour synchroniser les paquets de données. Il est aussi important d’avoir les mêmes structures et paramètres de trame pour l’émetteur et le récepteur.

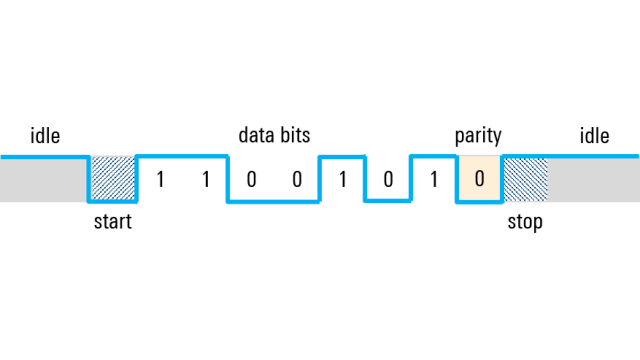
Un format type d’une trame d’UART est visible ci-dessous.

Figure : Trame de message UART type

Elle est constituée de la manière suivante :

* Un bit de start toujours à ‘0’, ce qui permet au récepteur de détecter le début du message.
* Des bits de données : il y a souvent 7 ou 8 bits de données, mais dans certains cas le nombre peut aller de 5 à 9 bits. Ils sont souvent envoyés du LSB au MSB, mais peuvent être envoyés dans l’autre sens.
* Il peut y avoir un bit de parité, mais il est optionnel. Ce bit peut être utilisé pour la détection d’erreur, il est inséré entre la fin des bits de données et le bit de stop. En cas de parité impaire, ce bit est réglé pour que le nombre total de ‘1’ soit impair. En cas de parité paire, ce bit est réglé pour que le nombre total de ‘1’ dans la trame soit pair.
* Un bit de stop toujours à ‘1’, mais sa durée peut être de 1 bit, 1,5 bits ou 2 bits.

Comme pour l’I2C, le niveau logique de repos est le niveau haut.

Pour le gestionnaire RFID, un code déjà réalisé a été réutilisé. Celui-ci a été légèrement modifié afin que le bit de stop dure 2 bits, comme précisé par la datasheet du composant et le Baud rate a été fixé à 57200.

#### Développement du gestionnaire RFID

Une fois les modules UART RX et TX mis en place, le gestionnaire RFID a pu être développé. Une machine d’état a pu être réalisée. Celle-ci permettait d’envoyer et de recevoir les séquences présentées plus haut sur la Figure 20. Afin d’envoyer ces séquences, divers tableaux contenant les différentes séquences ont été créés : un tableau init, un tableau Data avec toute la séquence d’envoi, de la séquence du REQA à la séquence de la lecture du registre désiré, un tableau offset pour se retrouver dans le tableau Data au fur et à mesure des envois de séquences et un tableau contenant les longueurs des différentes séquences.

Dans un premier temps, l’initialisation a été mise en place et va permettre d’envoyer le *ProtocolSelect*. On reboucle sur cette initialisation tant que la commande n’est pas correctement reçue par le lecteur RFID. Ensuite, l’envoi des différentes séquences et lectures correspondantes a été développé. Si lors d’une lecture le RFID renvoie une erreur de lecture, le processus d’envoi redémarre à l’envoi du REQA.

Pour certains envois depuis le FPGA, plus précisément les 2 selects de la boucle d’anticollision, il est nécessaire de récupérer une partie de la réponse précédemment reçue pour pouvoir l’envoyer à la séquence suivante. Pour cela, le tableau Data a été utilisé pour stocker ces données lors des lectures.

D’autre part, en cas de non réception d’une réponse ou en cas d’une réponse moins longue que prévue du fait d’une erreur de lecture, deux problèmes qui pourraient occasionner que la machine d’état reste bloquée dans un état et pour cela, un timeout a été mis en place : au bout de 50 ms il permet de revenir dans l’état d’initialisation et recommencer tout le processus.

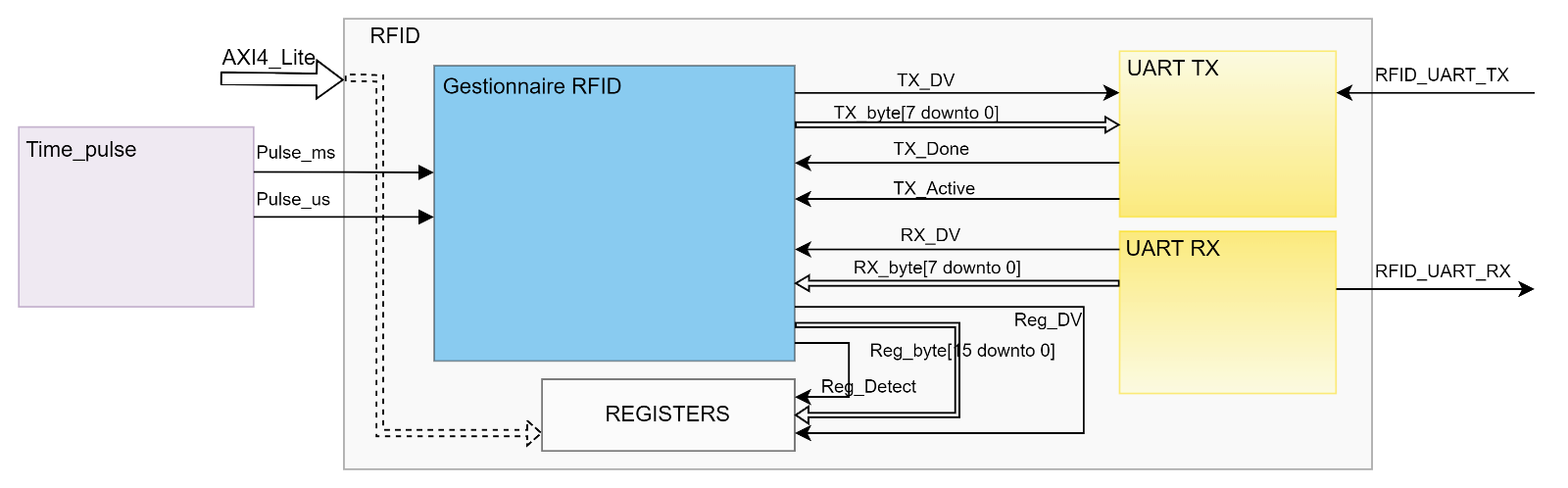
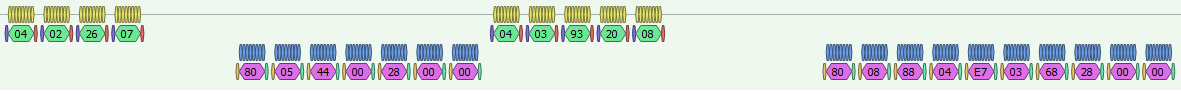
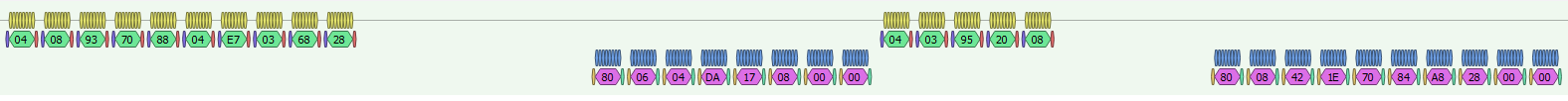
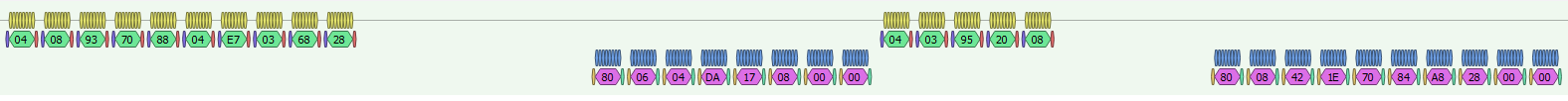
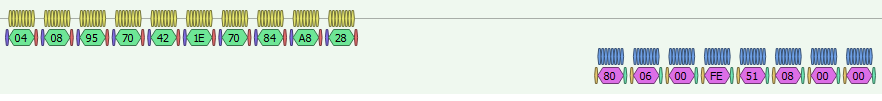
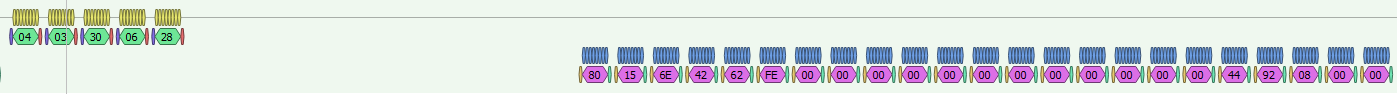
Une fois le code rédigé et testé en simulation et sur carte, tout comme pour les autres blocs, la communication avec le Software via l’AXI a été mise en place. Cette communication a été testée sur Vitis puis avec le PYNQ. Le schéma bloc visible ci-dessous a été obtenu.

Figure : Schéma block du contrôleur RFID

En parallèle, la séquence envoyée par le FPGA a été vérifiée à l’aide de l’analyseur logique. Cela permettait de vérifier les séquences d’envois mais aussi les différentes durées entre les demandes et les réponses. Le FPGA envoyait la séquence visible ci-dessous avec :

* Envoi du REQA et réception de l’ATQA puis envoi de l’anticollision 1 et réception de la réponse correspondante
* Envoi du Select 1 et réception de la réponse associée
* Envoi de l’anticollision 2 et réception de la réponse du tag
* Envoi du Select 2 et réception de ma réponse correspondante
* Envoi de la demande de lecture puis lecture du registre 0x03 du tag sélectionné

On peut observer que la séquence est cohérente avec la séquence présentée dans la Figure 20.

## Gestionnaire du minuteur de 2h

Nous voulions mettre en place une ronde qui se lance de manière automatique toutes les 2h. Dans ce but, un minuteur de 2h a été implémenté. Ce minuteur utilise le générateur d’impulsions détaillé plus haut. Pour qu’il soit plus simple à tester et plus malléable, une double entrée a été mis en place : un pour les impulsions toutes les secondes et l’autre pour les impulsions toutes les minutes. Dans ce même but, le Software peut aussi modifier le nombre de secondes ou de minutes comptées avant de lancer l’alerte « ronde ».

Le principe est relativement simple, il y a un compteur qui va aller jusqu’au nombre de minutes ou secondes voulues, et ce compteur va en fonction compter le nombre d’impulsions de la ligne sélectionnée. Une fois le nombre voulu atteint, un flag visible par le software va passer à ‘1’ qui alertera donc qu’il est temps de lancer une ronde. La communication avec le software se fait de la même façon que pour tous les autres blocs : elle passe par un AXI 4 Lite.

## Balises

Des balises sont utilisées pour détecter la présence d’une personne dans l’agence à l’aide d’un détecteur de mouvement HC-SR501 PIR et d’un STM32. Elles ont été placées à différents emplacements dans l’agence et pourront transmettre une alerte au robot. Nous avons réutilisé les balises de l’année précédente car elles étaient encore fonctionnelles. Le STM32 est utilisé pour pouvoir envoyer l’alerte reçue par le détecteur de mouvement à travers un réseau Wifi MeSH. Ce dernier permet de connecter toutes les balises entre elles. De plus, nous avons placé un STM32 sur le robot qui va permettre d’envoyer par le biais d’une connexion UART, le numéro de la balise qui a envoyé une alerte. Mon travail a été de programmer les balises et le STM32 du robot afin de mettre en place le système d’alertes. Cette programmation se fait en Arduino. Dans un second temps, le FPGA a été programmé pour lui permettre de signaler une alerte et d’envoyer les messages reçus au processeur.

### Mise en place du réseau Wifi MeSH

Dans une architecture Wifi traditionnelle, tous les nœuds sont connectés à un unique nœud central et tous communiquent à travers ce point d’accès. L’inconvénient est que le point d’accès central a une portée limitée et tous les nœuds doivent donc être à sa portée pour pouvoir communiquer.

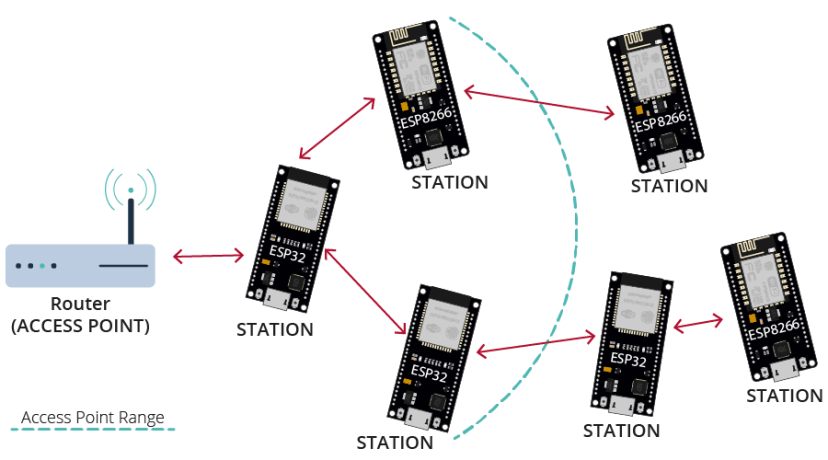
Dans le cadre d’un réseau Wifi MeSH, tous les nœuds n’ont pas besoin d’être connectés à un nœud central. Ils sont chargés de relayer les transmissions d’un nœud à un autre, ce qui permet notamment de couvrir une plus grande surface physique avec un même réseau Wifi MeSH.

Figure : Schéma du réseau Wifi MeSH

Pour mettre en place cette méthode de communication pour les balises, une librairie Arduino dédiée à la mise en place de ce réseau a été utilisée : « *PainlessMesh*».

Ensuite, les balises ont été programmées de manière à ce qu’elles envoient un message uniquement lorsqu’un mouvement est détecté par le détecteur de mouvement. D’autre part, il y a un ESP32 sur lequel il n’y a pas de détecteur de mouvement car il a été utilisé pour transférer les alertes sur le robot pour le biais d’une liaison UART.

### Mise en place d’une communication UART

Une fois le réseau Wifi-MeSH mis en place, un choix de méthode de communication entre la balise principale et le FPGA a été nécessaire. Il y avait diverses options possibles et notamment l’I2C ou l’UART. Mais le choix s’est finalement orienté vers la seconde solution car cela permet d’envoyer le numéro de la balise d’alerte automatiquement, sans attendre la demande de lecture du Master (le FPGA) en amont.

Pour mettre en place la communication UART, un protocole de communication UART déjà présent sur les STM32 a été utilisé. Ensuite, un module UART RX a été réutilisé pour la réception au niveau du FPGA. Ce qui permet de récupérer les données envoyées par les balises et les transmettre au Software. Un module UART TX a aussi été mis en place au niveau du FPGA pour qu’ultérieurement la mise en place d’une communication dans l’autre sens soit simplifiée, si nécessaire.

Comme pour les autres modules, une communication AXI 4 lite avec le software a été mise en place et l’ensemble a été testé sur Vitis puis avec un driver Python sur le PYNQ mis en place par le développeur Software.

# Résultats

En ce qui concerne la partie FPGA, la plupart des blocs que je devais développer ont été réalisés et sont fonctionnels. Il me manque notamment les tests finaux sur les ultrasons mais il manque aussi le gestionnaire de batterie : le GSM. Ces deux derniers n’étant pas critiques pour le bon fonctionnement du robot, leur absence n’altère en rien son fonctionnement.

Pour la partie Hardware, la carte interface qui a été réalisée a été testée et est bien fonctionnelle. Les divers périphériques connectés dessus communiquent correctement avec la Zybo. D’autre part, le chargeur à induction a été développé mais n’a pas été imprimé par manque de temps et n’a donc pas pu être testé.

Finalement, la partie Software a été plus longue que prévu car il était compliqué de mettre une image Pynq sur la Zybo puisqu’il n’y en avait pas de préexistante. Toutefois, le Software communique bien avec la base holonome et la communication avec le FPGA est fonctionnelle. Seuls les ultrasons n’ont pas eu le temps d’être implémentés. Le développeur Software a pu implémenter divers algorithmes de navigation ainsi qu’un gestionnaire d’alerte. Seule la connexion Wifi depuis la Zybo manque car nous n’avons pas réussi à activer le port USB sur lequel est connecté le dongle Wifi.

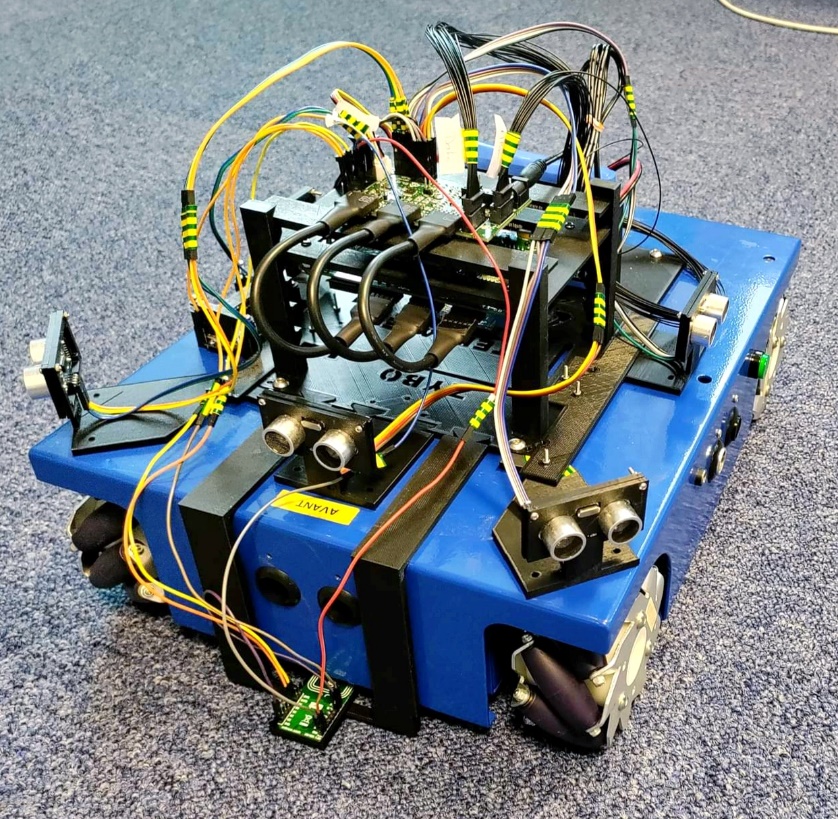
De plus, nous avons imprimé une nacelle pour le robot et des supports pour les différents périphériques grâce à l’imprimante 3D. Ceci nous permettait de réaliser plus facilement nos tests et de déplacer le robot plus sereinement. Nous obtenons le robot final visible sur l’image ci-dessous.

Figure : Image du robot final

# Conclusion

Je tire un bilan positif de ce stage et ce fut une expérience très enrichissante. J’ai rempli la plupart des objectifs fixés : les gestionnaires de l’IMU, des balises, du RFID sont fonctionnels, le Dijkstra et le minuteur. Le robot est capable de naviguer ainsi que de se localiser. Le travail autour a bien avancé malgré les retards que nous avons eu notamment au niveau des livraisons. Ce projet m’a permis de toucher à diverses nouvelles choses comme l’impression 3D, l’Arduino pour les balises ou pour tester les séquences du RFID. J’ai pu travailler sur la mise en place de la communication avec le Software, que ce soit les AXI ou les drivers en Python, cela a rendu mon stage d’autant plus complet et intéressant.

Finalement, le projet a bien avancé durant ce stage : de nombreuses fonctionnalités ont été mises en place, notamment au niveau de la localisation, la navigation, le lancement de la ronde… Mais il reste diverses fonctionnalités à développer, ce qui peut être intéressant pour les stagiaires de l’année prochaine. Par exemple, au niveau du FPGA, ils pourraient finir de mettre en place les ultrasons et pourraient également rajouter une caméra en utilisant le MIPI disponible sur la Zybo.

Ces six mois de stage m’ont permis de mettre en application de nombreuses compétences acquises lors de mes années à l’Enseeiht et ont aussi été très formateurs sur de nombreux aspects, notamment technique. Ce stage m’a permis de prendre en main toute la gestion de projet, mais également les parties conception FPGA et communication avec le Software.

Cette expérience m’a donné une idée plus claire de la réalité du métier d’ingénieur en conception FPGA et m’a permis de confirmer mon intérêt pour la conception FPGA tout comme mon désir de continuer dans ce secteur.

# Bibliographie

1. **Basics of the i2c communication protocol**, <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/>, Scott Campbell, 2016
2. **Hardware Architecture for Finding Shortest Paths**, *IEEE*, K. Sridharan, T.K. Priya, P. Rajesh Kumar, 2009
3. **Evaluation of an FPGA-Based Shortest-Path-Search Accelerator**, *Int'l Conf. Par. and Dist. Proc. Tech. and Appl.,* Yasuhiro Takei, Masanori Hariyama, Michitaka Kameyama
4. **Dijkstra’s Shortest Path Routing Algorithm in Reconfigurable Hardware**, *Signal Processing Laboratory*, Matti Tommiska, Jorma Skytä, 2001
5. **NFC Forum Type 4A & 4B Tag Platform Operations with the TRF7970A**, *NFC/RFID Training Module*, Texas Instruments, 2014
6. **Le NFC et la norme ISO/IEC 14443**, <https://redfroggy.fr/le-nfc-et-la-norme-isoiec-14443/>, Florent Perinel, 26 janvier 2016
7. **Getting Started With ESP-MESH and ESP-8266**, <https://www.instructables.com/Getting-Started-With-ESP-MESH-and-ESP-8266/>, Ruchir Sharma
8. **UART**, <https://fr.wikipedia.org/wiki/UART>
9. **Compréhension du UART**, <https://www.rohde-schwarz.com/fr/produits/test-et-mesure/oscilloscopes/educational-content/comprehension-uart_254524.html>