Stage technicien bac +2

Conception et réalisation de cartes filles pour les réseaux de capteurs

Du 21 juin au 27 août

Plan :

Tuteur de stage : Christophe Braillon

Tuteur enseignant : Bernard Glossi

1) Le cadre de travail :

Mon stage s'est déroulé à l'INRIA (Institut national de recherche en information et automatique) Grenoble, situé plus précisément à Montbonnot qui est l'un des 8 centres de recherche, les autres sont situés à Rocquencourt, Rennes, Sophia Antipolis, Nancy, Bordeaux, Lille et Saclay. Une grande partie des chercheurs sont aussi enseignants, leurs étudiants préparent d'ailleurs leur thèse au sein de l'institut (ce qui représente environ 1 000 personnes). L'INRIA est composé de 4 100 personnes dont 3 150 scientifiques. L'INRIA favorise le transfert de technologie en aidant la création d'entreprise. Plus de 98 entreprises ont été créées grâce à ce soutien. C'est sa filiale INRIA-Transfert qui s'en occupe.

L'INRIA a un domaine d'activité très vaste, qui va de la médecine numérique en passant par le domaine automobile et le monde du sport. La technologie peut en effet être développée dans tous les domaines de la vie quotidienne.

J'ai été accueilli dans le service SED : Service d'Expérimentation et de Développement. Ce service s'occupe avant tout de fournir un support à la recherche. Il met en place des plateformes pour les chercheurs. Ce service est présent dans chaque centre INRIA. Leurs missions sont identiques et se déclinent selon 3 axes importants :

- Maintenir un réseau d'expertises pour diffuser les bonnes pratiques de développement logiciel et l'utilisation d'outils communautaires au sein des équipes de projets.
- Mettre en place, développer et maintenir les plates-formes expérimentales avec les équipes de projets. Ce qui est la plus grosse part de travail.
- Ils participent aux développements logiciels ainsi qu'aux ADT (Actions de Développement Technologique) au sein des équipes de projets.

2) Présentation du cahier des charges :

Le but de mon stage était de réaliser une carte fille pour un robot. Cela s'inscrit dans un programme de communication entre une dizaine de robots identiques. Le but n'étant pas d'étudier les robots mais la communication entre eux : voir comment ils interagissent avec leur environnement, comment ils peuvent collaborer pour effectuer une tache précise comme par exemple découvrir leur environnement ou trouver un objet particulier ou une source de chaleur. Ils doivent aussi communiquer avec un réseau de capteurs. Ce sont des boitiers identiques répartis dans une pièce de façon non nécessairement uniforme. Ils communiquent entre eux ou avec un PC. Ils sont utilisés pour récolter des informations sur l'environnement où ils se trouvent.

Dans la salle ou je travaillais nous disposions de 256 capteurs disposés au plafond. Ils forment des nœuds *(routage multi-saut)*. Un autre stagiaire s'occupe de programmer les capteurs et de faire différentes études sur la propagation de signal de points en points.

Quelques boitiers formant le réseau de capteurs

Chaque nœud intègre :

- Une unité de captage chargée de capter des grandeurs physiques (chaleur, humidité, vibrations, rayonnement...) et de les transformer en grandeurs numériques
- Une unité de traitement informatique
- Une unité de stockage de données
- Un module de transmission sans fil

Les réseaux de capteurs peuvent avoir un large champ d'action, en effet, ils peuvent être utilisés pour :

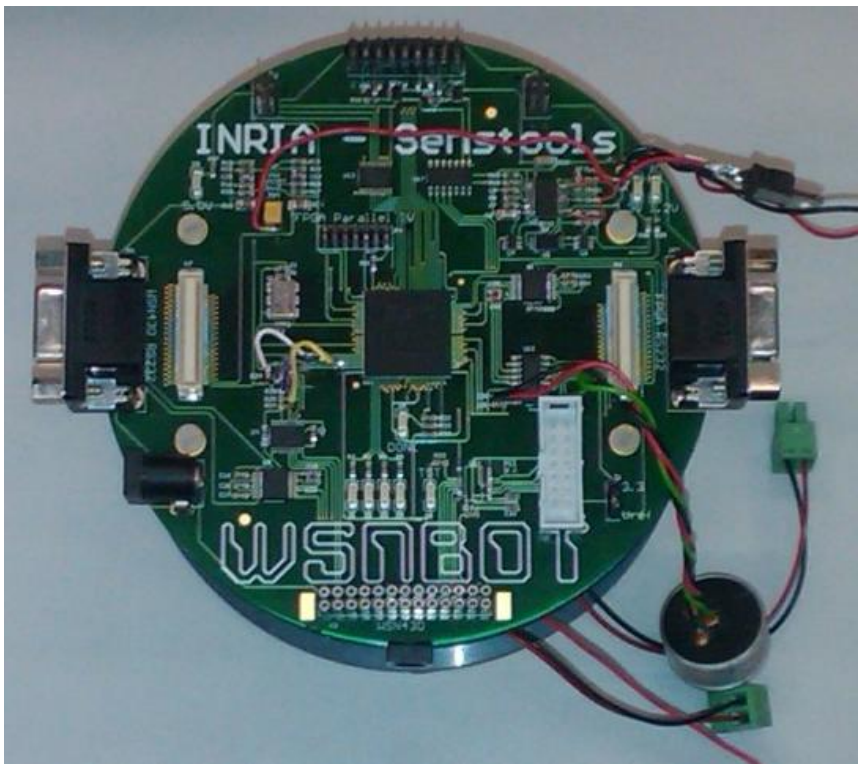
- La prévention des feux de forêt
- La mesure d'hydrométrie pour les agriculteurs (économie d'eau)

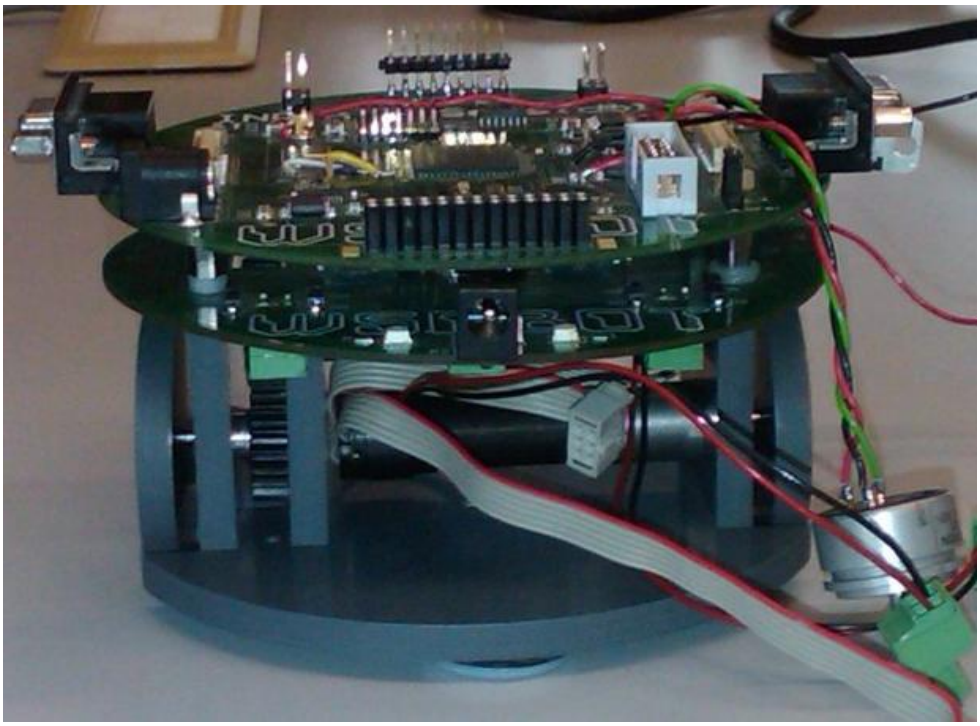
La capture de mouvement, de données physiologiques

L'INRIA va donc réaliser une flottille d'une dizaine de robots qui seront les actionneurs du réseau de capteur. A mon arrivée un prototype de robot était déjà en cours de construction. Celui-ci comportait une carte de puissance alimentant les moteurs à courant continu et une carte mère embarquant un FPGA comme composant central. Un asservissement en vitesse à été réalisé avec le FPGA pour piloter les moteurs mais le robot ne disposait d'aucun capteur. La carte que j'ai dû réaliser sera fixée sur la carte mère. Elle sera dupliquée, une pour chaque robot. Elle embarquera une centrale inertielle c'est-à-dire des composants tels qu'un accéléromètre, un gyromètre et un magnétomètre. Ceux-ci communiqueront avec le FPGA grâce à une liaison I2C ou SPI. Ils servent à connaitre la vitesse du robot, sa tendance à tourner et la direction qu'ils prennent. On peut aussi connaitre l'inclinaison du robot sur les 3 axes. Le magnétomètre est un compas électronique, il permet de connaitre l'orientation du robot par rapport au nord magnétique. Ces informations permettent d'avoir une idée très fidèle de ses déplacements. Grâce à ces capteurs très précis nous pouvons enregistrer le parcourt du robot en 3 dimensions. Cette carte doit pouvoir aussi recevoir des

capteurs de contact (bumper) pour savoir si le robot heurte un objet. Elle doit aussi pouvoir recevoir des télémètres infrarouges pour connaitre la distance séparant le robot de son environnement proche. Le projet n'étant pas fixé définitivement, cette carte doit pouvoir être évolutive. Mon travail consistait à créer cette carte (qui n'existe pas) et de récupérer les informations de position avec le FPGA. Une fois récupérées, les données sont transmises à un autre bloc (fait par une autre personne. Cette carte équipera donc chaque robot, je suis très content d'avoir travaillé sur un projet concret comme celui-ci. Il y aura une réelle application de mon travail de stagiaire. Ce stage comporte plusieurs tâches différentes, tout d'abord il faut créer cette carte, faire attention aux aspects mécaniques (dimensions, bien placer les composants, mettre les connecteurs précisément pour être sur que la carte s'adapte bien sur la carte mère). La seconde tâche consiste à créer le programme permettant de communiquer avec les différents composants.

Voici des photos du prototype réalisé :





3) Travail réalisé :

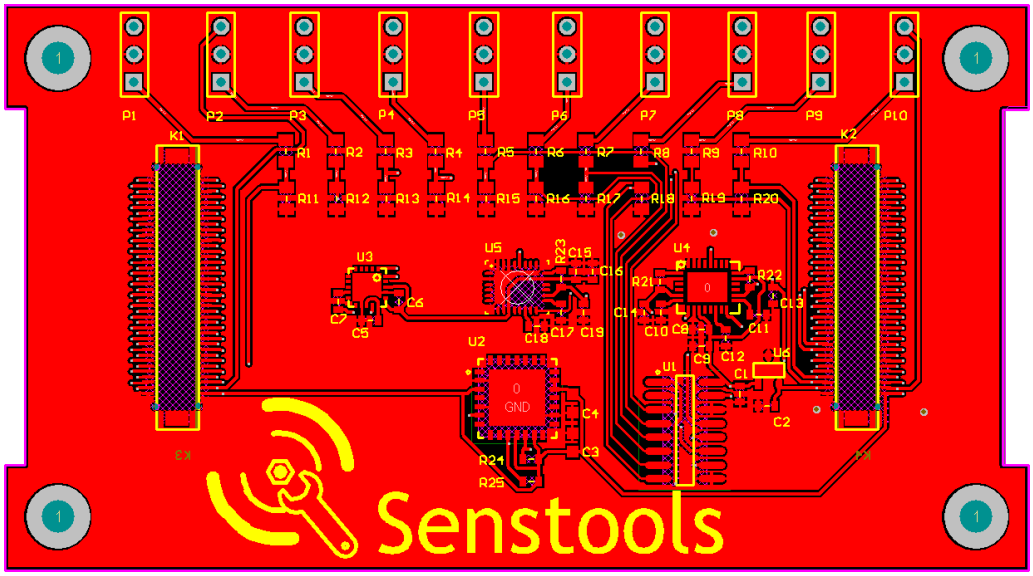
Dans un premier temps il a fallut que je me renseigne sur les composants à utiliser, connaitre quels sont leur mode de communication. La plupart des circuits ayant l'I2C et le SPI. Le choix entre les deux technologies étant imposé par le projet. Il a fallut aussi que je vérifie les tensions d'alimentation des composants. Une fois la liste définitive des circuits à utiliser rédigée j'ai saisis les schémas correspondants sur le logiciel utilisé par l'entreprise. . Il s'agit d'Altium Designer Winter 2010 (ex Protel).

Altium Designer

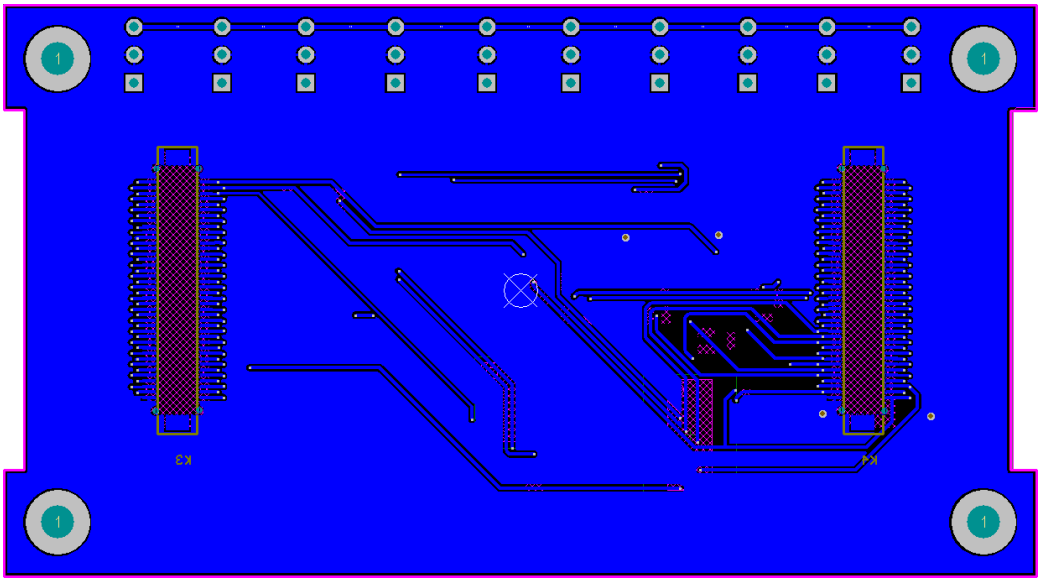
Ce logiciel permet de saisir des schémas et de créer le routage correspondant. J'avais déjà personnellement utilisé d'autres logiciels de CAO (Conception Assistée par Ordinateur) mais je n'avais jamais utilisé Altium. Celui-ci est très performant, il est très utilisé dans l'industrie de l'électronique. Il met en œuvre des fonctions très puissantes facilitant la conception de circuits. Il permet notamment d'avoir une vue 3D de son circuit ou connaitre l'impédance d'une piste.

Il m'a fallut un petit temps d'adaptation afin de maitriser les fonctions principales d'Altium. Cela me permet d'avoir une expérience professionnelle supplémentaire et de connaitre déjà un logiciel très utilisé dans l'industrie. Cela me servira certainement plus tard.

Voici la carte que j'ai réalisée durant mon stage :



Couche du dessus



Couche du dessous

Nous pouvons voir en haut une rangée de connecteurs destinée à recevoir soit des télémètres infrarouges soit des interrupteurs de contact.

Les deux barres verticales foncées sont les connecteurs qui seront enfichés dans la carte mère. Les autres composants constituent la centrale inertielle.

4) Introduction au VHDL :

Après mon travail sur la carte fille, mon tuteur de stage m'a montré les rudiments du langage VHDL. Pour cela j'ai utilisé la carte mère du robot. Le premier objectif que j'ai eu à accomplir était de réaliser une description matérielle VHDL, pour le FPGA du robot, en utilisant les leds de la carte, afin de me familiariser avec ce langage totalement différent du C. D'une part les syntaxes sont différentes et la manière de penser et de concevoir est totalement opposée. En effet le langage VHDL est descriptif alors que le langage C est impératif (séquentiel). En C, chaque instruction est effectuée dans un ordre particulier, on peut ainsi avoir une bonne vision globale du projet. A l'opposé, la description (le code) VHDL est transcrite en un circuit électrique qui lui est toujours valable. En effet, par exemple, deux portes logiques sont toujours active et en même temps. Pour retrouver le mode de fonctionnement de type C, il existe une technique très employée, qui est de mettre en place une machine à état finis. Elle démarre dans un état particulier et passe dans un autre à certaines conditions. C'est ce procédé que j'ai utilisé pour faire un chenillard avec les leds de la carte mère. J'ai donc pu mettre en œuvre ce que j'avais appris sur les machines à état lors de l'enseignement EE240. Le code VHDL à été synthétisé (compilé) avec la suite de logiciel de Xilinx. Il est ensuite intéressant de connaitre l'implémentation physique de notre programme que le compilateur Xilinx à choisis. Ici le

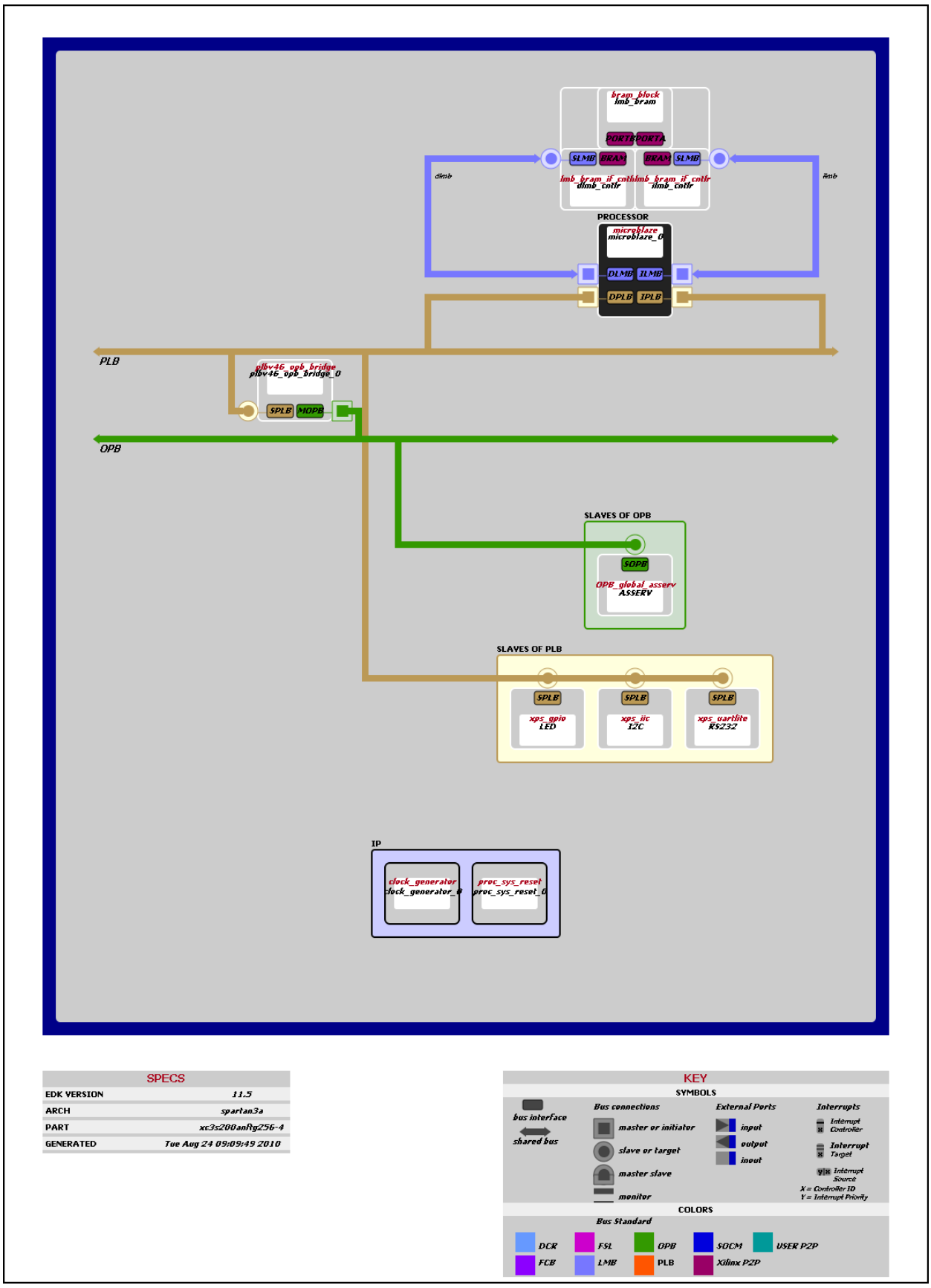
chenillard à été réalisé grâce à un registre à décalages ce qui est plutôt logique pour un chenillard qui allume les leds les unes après les autres.

Ensuite mon tuteur de stage m'a montré comment implémenter un cœur de microcontrôleur dans le FPGA. C'est-à-dire que dans le FPGA, les blocs et les portes logiques sont agencés de façon à former les organes d'un microcontrôleur : le processeur avec son ALU, ses différents bus de donnée, mais aussi de la mémoire RAM, la mémoire de programme et les autres constituants essentiels. Ici par exemple nous n'avions pas besoin d'unité de calcul en virgule flottante (FPU), très gourmande en place, nous ne l'avons donc pas implémentée. En effet, grâce à cette méthode (implémenter un microcontrôleur dans un FPGA), nous pouvons avoir un microcontrôleur sur-mesure (dans les limites du FPGA naturellement). Cela permet d'avoir une très grande souplesse, une grande rapidité d'adaptation. En effet il n'y a nul besoin de changer de composant pour changer le design physique. (Ce qui d'ailleurs reviendrait à changer la carte entière vus que les composants sont des CMS soudés au four à refusions). Le microcontrôleur de base est celui proposé par Xilinix : le microblaze (microcontrôleur 32 bits). En général les blocs le constituants sont écris en VHDL avec possibilité ou non d'avoir les sources pour savoir comment cela marche ou les modifier.

Après avoir choisi notre processeur, nous pouvons rajouter différents blocs plus ou moins standard écrits généralement en VHDL. Ce sont des « IP » (Intellectual Property), c'est-à-dire des blocs proposés par Xilinx dans une bibliothèque ou même des blocs créés par l'utilisateur (comme l'asservissement du robot). Une fois l'IP choisie il faut la configurer, et bien sûr la relier à un bus du processeur pour que celui-ci puisse s'en servir.

Le microcontrôleur est standard, il se programme en C. Xilinx intègre un compilateur (un dérivé de gcc de linux) il est donc facile de le prendre en main et de debugger ses programmes. Le projet intègre par défaut un debugger physique (implémenté dans le FPGA) permettant d'analyser le comportement du processeur et notamment de faire exécuter manuellement chaque instructions par le processeur.

Voici une vue globale du projet :



Nous pouvons voir en noire le processeur (non détaillé) ainsi que tous les bus standard *(PLB, OPB)* ainsi que les bus de donnée d'accès mémoire tel que DLMB et ILMB. Spartan3a étant la référence du FPGA utilisé.

Le bloc au dessus du processeur est l'IP gérant la mémoire de donnée. Nous pouvons définir la taille voulue, la taille de la pile, de la mémoire de programme dans les limites des ressources du FPGA. Le bloc vert est l'IP gérant l'asservissement des moteurs, il a été créé avant mon arrivée. Le bloc en bas à gauche du processeur est celui gérant les roues codeuses des moteurs. Ce bloc sert à connaitre la vitesse réelle de rotation des roues du robot afin de réaliser la boucle d'asservissement.

Le premier bloc jaune du bas est celui gérant les leds. Il est donc connecté physiquement aux leds. Pour les piloter, il suffit d'écrire dans un registre qui leur est dédié. On voit ici l'avantage d'une architecture modulaire car on peut rajouter des registres au processeur comme on le désire. Le second bloc jaune est celui gérant l'I2C, une description plus précise de celui-ci sera faite par la suite. Enfin le dernier bloc est l'UART du microcontrôleur.

Par défaut le projet ne comporte pas d'UART (de bloc gérant la liaison série RS 232) car il n'est pas forcement nécessaire. Il m'a donc été confié de le rajouter.

Il n'y a pas de difficulté particulière à en ajouter un mais j'ai cependant rencontré quelques obstacles lors de la configuration. Il faut donc rajouter l'IP, configurer la liaison sérielle (vitesse, contrôle de flux, parité). J'ai dus choisir un bus sur lequel relier l'IP (le PLB ici). La dernière étape a été de relier les broches Tx et Rx de cet IP à des broches physiques du FPGA (le mappage) afin de communiquer avec un PC. Après plusieurs tests la liaison est restée muette. J'ai d'abord mis en cause mon mappage ou la fonction xil_printf fournie par Xilinx. Nous avons finis par découvrir qu'en faite le programme marchait mais que la broche Tx n'était par défaut pas reliée vers l'extérieur mais vers le debugger. N'utilisant pas celui-ci nous l'avons supprimé, c'est pourquoi il n'apparait pas dans le schéma. Une fois cette opération effectuée la liaison série RS232 a marchée comme il fallait. Le pack Xilinx est très performant mais parfois très compliqué à prendre en main ou à comprendre d'où vient le problème. Cette erreur à mis du temps à être identifiée.

Ce projet de robot à l'INRIA étant commencé depuis quelques temps et durant encore quelques mois, n'avons pas pu produire ma carte pendant la durée de mon stage. Elle sera réalisée courant octobre. Pour mes différents tests à venir j'ai utilisé une carte générique pour la remplacer. Il m'a donc été difficile de faire des tests définitifs.

Pour pouvoir communiquer avec les différents composants il a fallut que je mette en œuvre la liaison I2C. Pour cela Xilinx propose une IP déjà faite. Le protocole I2C étant essentiellement réalisée en logiciel ici Xilinx rajoute une couche matérielle. L'IP contient des registres de configuration ainsi qu'une pile de réception et une d'émission. C'est-à-dire que nous n'avons pas à gérer l'I2C de façon « bas niveau » comme sur certains autres microcontrôleurs, en acquittant ce que l'on reçoit, en faisant attention à libérer la ligne de donnée... Cette méthode plus « terre à terre » à au moins l'avantage d'être compréhensive, il est plus facile de savoir où est un problème vus que tout est géré manuellement. Ici c'est la couche matérielle qui s'en charge. L'adresse du microcontrôleur est inscrite dans un registre. Quand un octet nous est destiné cela est détecté

automatiquement par la couche matérielle (grâce à notre adresse qui est comparée a celle de destination du message) générant aussitôt une interruption. Il nous suffit donc de dépiler la pile de réception pour collecter les données reçues. Pour émettre c'est le contraire il faut empiler l'adresse et les données dans la pile d'émission. La couche matérielle de l'IP gérant le reste c'est-à-dire détecter un conflit sur le bus (si deux maitres émettent en même temps), ralentir le maitre si l'échange est trop rapide, ré-envoyer un message en cas de problème lors de la transmission.

Cela parait simple vus que pratiquement tout est géré automatiquement cependant l'IP I2C de Xilinx s'est révélée très dure à configurer et à mettre en œuvre. Il a été difficile de savoir ce qui se passait réellement vus que ce n'est pas notre programme qui gère directement l'échange.

5) Erreurs et problèmes rencontrés :

Une erreur d'interprétation de la datasheet m'a fait perdre un temps considérable.

La voici :

Pour configurer l'I2C il faut renseigner des registres correctement.

Voici un registre prit comme exemple extrait de la datasheet :

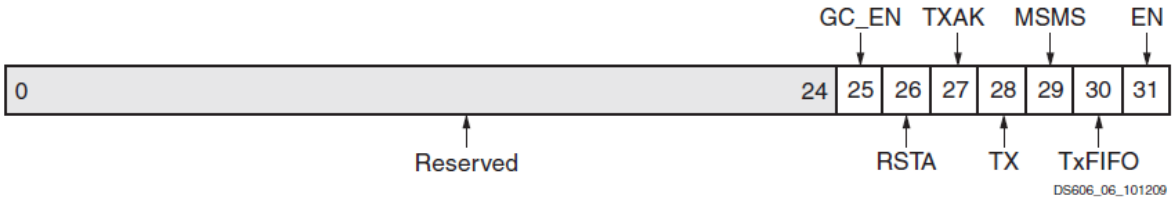


Figure 6: **Control (CR) Register**

Voyant des chiffres de 0 à 31 j'ai pensé que le bit 0 correspond à celui de poids faible, soit 2 puissance 0, de même pour le bit 31 un poids de 2 puissance 31.

Donc pour *-par exemple-* mettre à '1' le bit TX, j'inscris dans le registre 2 puissance 28 en décimal (ce qui fait en binaire un '1' et vingt-huit '0') vus que TX est le 28éme bit.

Non, en réalité le 0 du dessin représente le bit de poids fort soit 2 puissance 31 et le bit 31 a comme poids 2 puissance 0... Cependant la datasheet ne le précise pas. Pour une compréhension aisée – *personnellement*- j'aurais opté pour numéroter les bits dans l'autre sens c'est-à-dire 0 pour le poids faible et 31 pour le poids fort comme cela est fait d'habitude.

C'est mon responsable qui s'est rendu compte de cette erreur, au hasard d'un exemple donné sur une autre documentation dans les répertoires de Xilinx. Nous avons perdus beaucoup de temps à revoir notre protocole de communication i2c alors que nous ne renseignions pas les registres dans le bon sens…

Nous nous sommes demandés si cela correspondait à une représentation little-endian (petit-boutiste) ou big-endian (gros-boutiste) mais cela n'explique toujours pas cette représentation. En big-endian le mot de 32 bits est découpé en octets, effectivement l'octet de poids fort est stocké en premier (à l'adresse la plus basse) mais octet par octet (donc par blocs de 8 bits) mais pas bit par bit comme ici.

Je n'ai pas pu terminer la fin de mon projet qui consistait à communiquer avec les capteurs par manque de temps.

6) Bilan :

Ce stage durant tout l'été m'a permis d'avoir une réelle expérience professionnelle. En effet j'ai pu apprendre comment utiliser le logiciel Altium, découvrir comment marche le VHDL et voir comment fonctionne une entreprise. Durant ces 9 semaines j'ai pu me confronter aux exigences d'une entreprise c'est-à-dire travailler en autonomie, respecter des délais et communiquer avec une équipe. La réalisation de ce projet m'a permis d'approfondir mes connaissances. En effet je connaissais déjà les FPGA et le principe général du VHDL mais je n'avais jamais travaillé sur un projet aussi conséquent. Je regrette de ne pas avoir pu terminer l'intégralité de mon projet par manque de temps.

7) Remerciements :

Je tiens à remercier l'INRIA Grenoble pour m'avoir accueilli. Merci au service SED, à Roger Pissard-Gibollet, à Christophe Braillon mon tuteur de stage et à Jean-Francois Cuniberto qui m'a fait visiter le centre et m'a aider pour les questions logistiques. J'ai d'ailleurs beaucoup aimé l'ambiance qui régnait au centre.

Un merci particulier à Guillaume Roche pour m'avoir suivi et aiguillé durant ce stage.