Hes·SO VALAIS WALLIS
Haute Ecole Spécialisée
de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences
Western Switzerland

Domaine Sciences de l'ingénieur
Rte du Rawyl 47
CH- 1950 Sion 2
Tél. +41 27 606 85 11
Fax +41 27 606 85 75
info@hevs.ch

www.hevs.ch

Filière Systèmes industriels

Orientation Infotronics

Diplôme 2012

Michele Korell

Noeud Ethercat pour moteur électrique

Professeur François Corthay

Expert Claude Magliocco

Sion, le 13 juillet 2012

HES-SO Valais

SI	TV
X	X

Données du travail de diplôme
Daten der Diplomarbeit

FO 1.2.02.07.AB
pof/31/01/2009

☒ FSI ☐ FTV	Année académique / *Studienjahr* **2011/2012**	No TD / *Nr. DA* **it/2012/20**
Mandant / *Auftraggeber* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*	Etudiant / *Student* **Michele Korell** Professeur / *Dozent* **François Corthay**	Lieu d'exécution / *Ausführungsort* ☒ HES—SO Valais ☐ Industrie ☐ Etablissement partenaire *Partnerinstitution*
Travail confidentiel / *vertrauliche Arbeit* ☐ oui / ja [1] ☒ non / nein	Expert / *Experte (données complètes)* **Claude Magliocco**	

Titre / *Titel*

Noeud Ethercat pour moteur électrique

Description et Objectifs / *Beschreibung und Ziele*

Le but de ce projet est de réaliser un noeud Ethercat pour moteur électrique en FPGA.

Ethercat est un bus de terrain temps-réel basé sur une couche physique Ethernet. Un maître prépare un message qui passe de noeud en noeud et chacun en tire les commandes qui lui sont attribuées et y place les réponses qui lui sont demandées. Après avoir fait le tour complet, le message est renvoyé au maître qui en tire les informations qui lui sont nécessaires.

Le noeud à réaliser reçoit une commande de couple à imprimer au moteur et fournit en retour la vitesse de celui-ci. L'exemple d'application se fera sur une brouette avec une roue à moteur électrique intégré disponible à l'école.

Les objectifs du projet sont de :

— réaliser un circuit numérique capable de piloter la roue en fonction d'une commande d'amplitude et de retourner la vitesse moyenne de la roue

— réaliser un circuit pour transmettre ces informations par Ethernet à un PC de commande.

Délais / *Termine*

Attribution du thème / *Ausgabe des Auftrags:* 14.05.2012	Exposition publique / *Ausstellung Diplomarbeiten:* 31.08.2012
Remise du rapport / *Abgabe des Schlussberichts:* 13.07.2012 | 12h00	Défense orale / *Mündliche Verteidigung:* Semaine / *Woche 36*

Signature ou visa / *Unterschrift oder Visum*

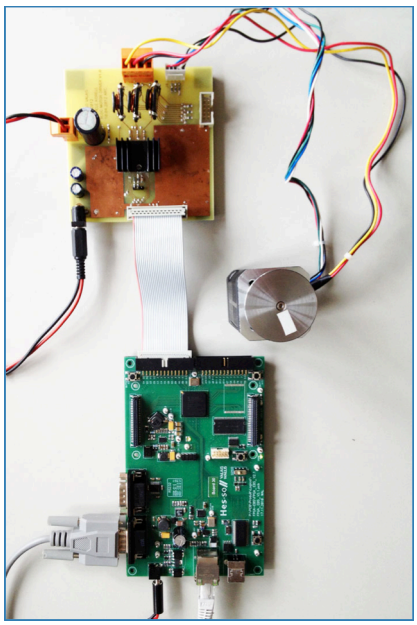
Responsable de l'orientation
Leiter der Vertiefungsrichtung: ..... [1] Etudiant/*Student*: .....

[1] *Par sa signature, l'étudiant-e s'engage à respecter strictement la directive et le caractère confidentiel du travail de diplôme qui lui est confié et des informations mises à sa disposition.*
Durch seine Unterschrift verpflichtet sich der Student, die Richtlinie einzuhalten sowie die Vertraulichkeit der Diplomarbeit und der dafür zur Verfügung gestellten Informationen zu wahren.

Rapport reçu le / *Schlussbericht erhalten am* ..... Visa du secrétariat / *Visum des Sekretariats* .....

Hes·so VALAIS WALLIS
Haute Ecole Spécialisée de Suisse occidentale
Fachhochschule Westschweiz
University of Applied Sciences Western Switzerland

Nœud EtherCAT pour moteur électrique



Diplômant/e Michele Korell

Objectif du projet

Ce projet consiste à développer un nœud de commande pour moteur électrique de type Brushless. Via le protocole EtherCAT, le nœud reçoit une commande de couple à imprimer au moteur et fournit en retour la vitesse de celui-ci.

Méthodes | Expériences | Résultats

Une application de contrôle envoie des commandes de type EtherCAT sur un réseau Ethernet à une carte de développement FPGA. Le circuit de commande traite les données reçues et génère un signal de commande pour un moteur triphasé de type Brushless à courant continu. À l'aide de capteurs à effet Hall on peut mesurer la vitesse moyenne de rotation et renvoyer ces informations à l'application de contrôle.

Le développement du système a commencé par la génération de la commande du moteur, ensuite avec l'ajout de la régulation numérique du système et pour finir avec le développement de l'interface de communication. Le contrôleur a été testé dans un réseau local Ethernet pour vérifier le fonctionnement du système.

Le premier prototype de nœud EtherCAT pour moteur électrique est fonctionnel. Il permet de contrôler la vitesse de rotation, le couple du moteur et les paramètres de fonctionnement en utilisant des commandes de type EtherCAT.

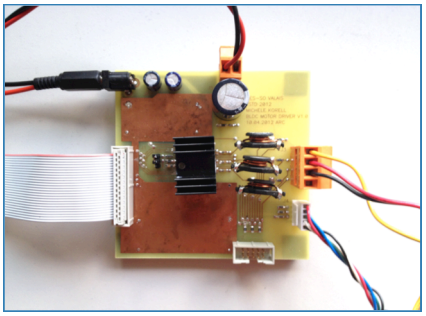
Travail de diplôme
| édition 2012 |

Filière
Systèmes industriels

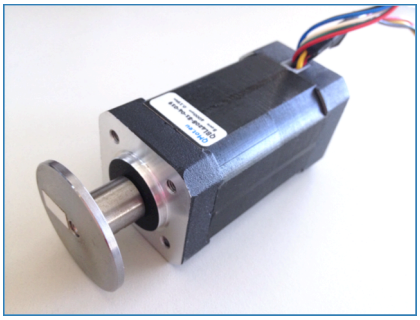
Domaine d'application
Infotronique

Professeur responsable
François Corthay
francois.corthay@hevs.ch

Partenaire
HES-SO Valais/Wallis



Circuit électronique de puissance pour la génération des signaux sinusoïdaux d'alimentation du moteur.



Moteur Brushless à courant continue (BLDC).

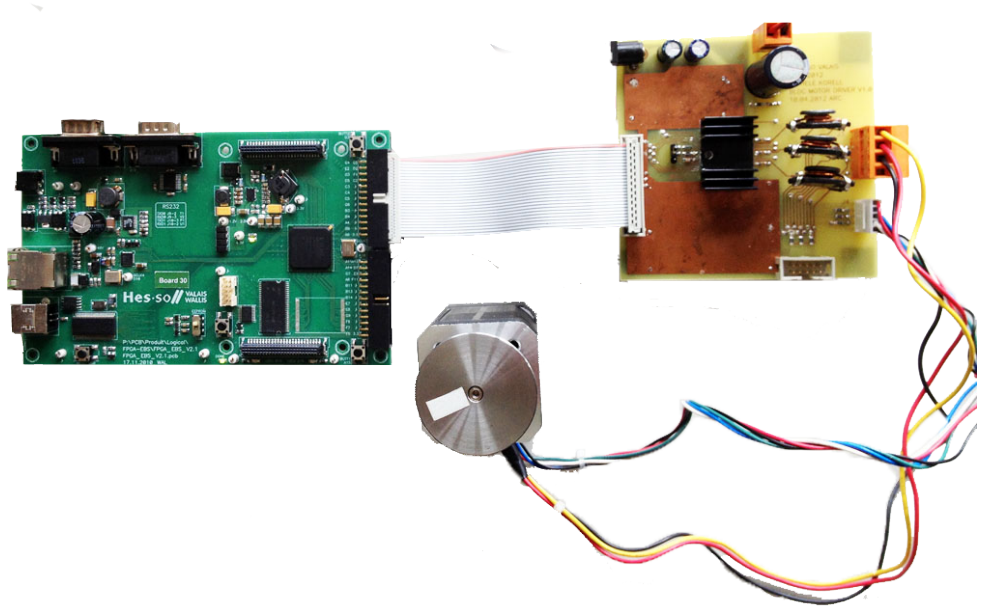
HES-SO Valais
Route du Rawyl 47
1950 Sion

Tél. 027 606 85 11
URL www.hevs.ch

MICHELE KORELL

NŒUD ETHERCAT POUR MOTEUR ELECTRIQUE

JUILLET 2012



SOMMAIRE

1. INTRODUCTION 6

2. CAHIER DES CHARGES 7

2.1 CONTEXTE 7
2.2 OBJECTIFS 7
2.3 CARACTERISTIQUES 7
2.4 SCHEMA DU SYSTEME 7

3. MOTEUR ELECTRIQUE BRUSHLESS 8

3.1 INTRODUCTION 8
3.2 MOTEUR BRUSHLESS DC 8
3.3 MOTEUR BRUSHLESS AC 8
3.4 RESUME ET APPLICATIONS 8
3.5 MESURE DE LA TENSION INDUITE 9

4. ETHERCAT 10

4.1 INTRODUCTION 10
4.2 STRUCTURE D'UN RESEAU 10
4.3 COMPOSITION D'UNE TRAME 11

5. CIRCUIT DE PUISSANCE 12

5.1 SCHEMA BLOC 12
5.2 COMMANDE 12

6. CIRCUIT DE COMMANDE NUMERIQUE 13

6.1 INTRODUCTION 13
6.2 SCHEMA BLOC 13
6.3 GENERATION DE LA COMMANDE PWM 13
6.3.1 Mode manuel 13
6.3.2 Mode automatique 13
6.3.3 Dimensionnement 14
6.3.4 Modulation PWM 14
6.3.5 Compteur de phase 15
6.3.6 Filtre passe-bas 16
6.3.7 Détecteur de phase 16
6.3.8 Boucle de réglage 17
6.4 REGISTRES DE CONFIGURATION 18
6.4.1 Registres de Base 18
6.4.2 Registres de Configuration 19
6.4.3 Registres de Fonctionnement 20
6.4.4 Registres de Contrôle 21
6.5 INTERFACES 22
6.5.1 Interface sérielle (RS232) 23
6.5.2 Commande sérielle 23
6.5.3 Interface ethernet (UDP) 24
6.5.4 Décodeur trame EtherCAT 24

7. APPLICATION DE CONTROLE 26

7.1 INTRODUCTION 26
7.2 APPLICATION PC 26
7.2.1 Interface utilisateur 26
7.2.2 Librairie EtherCAT 27
7.3 APPLICATION IPHONE 30

8. TEST DU SYSTEME 31

8.1 COMMANDE PWM 31
8.2 BOUCLE DE REGLAGE 32
8.3 COMMUNICATION ETHERNET 34
8.4 DEFAUT DU SYSTEME DE REGLAGE 35

9. AMELIORATIONS 36

9.1 BOUCLE DE REGLAGE 36
9.2 MACHINE D'ETAT ETHERCAT 36
9.3 COUCHE DE LIAISON 37

10. CONCLUSION 38

11. REMERCIEMENTS 39

12. ANNEXES ET REFERENCES 39

12.1 ANNEXES 39
12.2 REFERENCES 39

TABLE DES FIGURES

FIGURE 1: SCHEMA DU SYSTEME 7
FIGURE 2: MESURE DE LA TENSION INDUITE (BOBINAGE ETOILE) 9
FIGURE 3: TOPOLOGIE RESEAU ETHERCAT [REFERENCE 3] 10
FIGURE 4: COMPOSITION D'UNE TRAME ETHERCAT [REFERENCE 3] 11
FIGURE 5: BLDC POWER STAGE 12
FIGURE 6: CIRCUIT DE COMMANDE NUMERIQUE 13
FIGURE 7: SCHEMA FILTRE PASSE-BAS 16
FIGURE 8: SCHEMA DE PRINCIPE DE LA PLL 17
FIGURE 9: BOUCLE DE REGLAGE 17
FIGURE 10: STRUCTURE DU BUS DE DONNEES 18
FIGURE 11: MACHINE D'ETAT 23
FIGURE 12: DATAGRAM ETHERCAT [REFERENCE 3] 24
FIGURE 13: MACHINE D'ETAT DECODEUR ETHERCAT 25
FIGURE 14: INTERFACE PRINCIPALE APPLICATION PC 26
FIGURE 15: INTERFACE REGLAGES APPLICATION PC 27
FIGURE 16: TRAME ETHERCAT ENVOYE 27
FIGURE 17: TRAME ETHERCAT REÇUE 28
FIGURE 18: MESURE DU TEMPS DE TRANSMISSION D'UNE TRAME ETHERCAT 29
FIGURE 19: INTERFACE APPLICATION IPHONE 30
FIGURE 20: MESURE 1 DU SIGNAL PWM GENERE 31
FIGURE 21: MESURE 2 DU SIGNAL PWM GENERE 31
FIGURE 22: MESURE DE LA TENSION SINUSOÏDALE GENEREE 32
FIGURE 23: PARAMETRES DU TEST DE LA BOUCLE DE REGLAGE 33
FIGURE 24: MESURE DU COMPORTEMENT DE LA BOUCLE DE REGLAGE 33
FIGURE 25: TEST D'ENVOIE TRAME ETHERCAT MIXTE 34
FIGURE 26: BOUCLE DE REGLAGE AVEC FILTRE INTEGRATEUR 36

ABBREVIATIONS

BLDC : Brushless DC
PMSM : Permanent Magnet Synchronous Motor
PWM : Pulse Width Modulation
MAC : Media Access Control address
IP : Internet Protocol
UDP : User Datagram Protocol
ARP : Address Resolution Protocol
DHCP : Dynamic Host Configuration Protocol
DNS : Domain Name System
RAM : Random Access Memory

1. INTRODUCTION

Le but de ce travail de diplôme est de créer un contrôleur pour moteurs électriques de type Brushless. Ce système sert à contrôler le fonctionnement d'un moteur intégré à une roue. Il doit être capable d'échanger des données via un réseau Ethernet avec un ordinateur de commande. L'échange de données se fait en utilisant le protocole EtherCAT.
Le contrôleur du moteur reçoit une consigne de couple à imprimer au moteur et en retour donne la vitesse à laquelle le moteur est en train de tourner. La régulation effectuée par ce contrôleur n'est pas de type standard par rapport aux applications classiques des moteurs Brushless.
La création du contrôleur se fait sous forme de circuit numérique implémenté dans une carte de développement FPGA.
Par exemple, le nœud EtherCAT pourrait être utilisée dans des véhicules électriques possédant une ou plusieurs roues de traction. Dans ce cas, un maître pourrait commander chacune de ces roues de façon individuelle. À chaque roue correspondrait un nœud de commande. L'unité centrale pourrait envoyer la même commande de couple à chaque roue du véhicule, et le nœud serait capable d'effectuer la régulation de façon autonome.

2. CAHIER DES CHARGES

2.1 CONTEXTE

Le but du travail de diplôme, est de créer un nœud EtherCAT qui génère les signaux triphasés de commande pour un moteur électrique de type Brushless à courant continu.

2.2 OBJECTIFS

- Réaliser un circuit numérique capable de piloter la roue en fonction d'une commande d'amplitude et de retourner la vitesse moyenne de la roue.
- Réaliser un circuit pour transmettre ces informations par Ethernet à un PC de commande.

2.3 CARACTERISTIQUES

Le nœud de commande du moteur électrique doit être capable de générer un signal sinusoïdal triphasé, modulable en amplitude et fréquence.
Les paramètres du signal de commande du moteur doivent être mémorisés dans des registres de configurations. L'accès à ces paramètres s'effectue via une liaison Ethernet avec le protocole EtherCAT. Il doit être aussi possible d'effectuer une lecture de vitesse moyenne du moteur.
Il faut aussi créer un logiciel capable de modifier les valeurs des registres de configuration pour tester le fonctionnement correct du système de commande.

2.4 SCHEMA DU SYSTEME

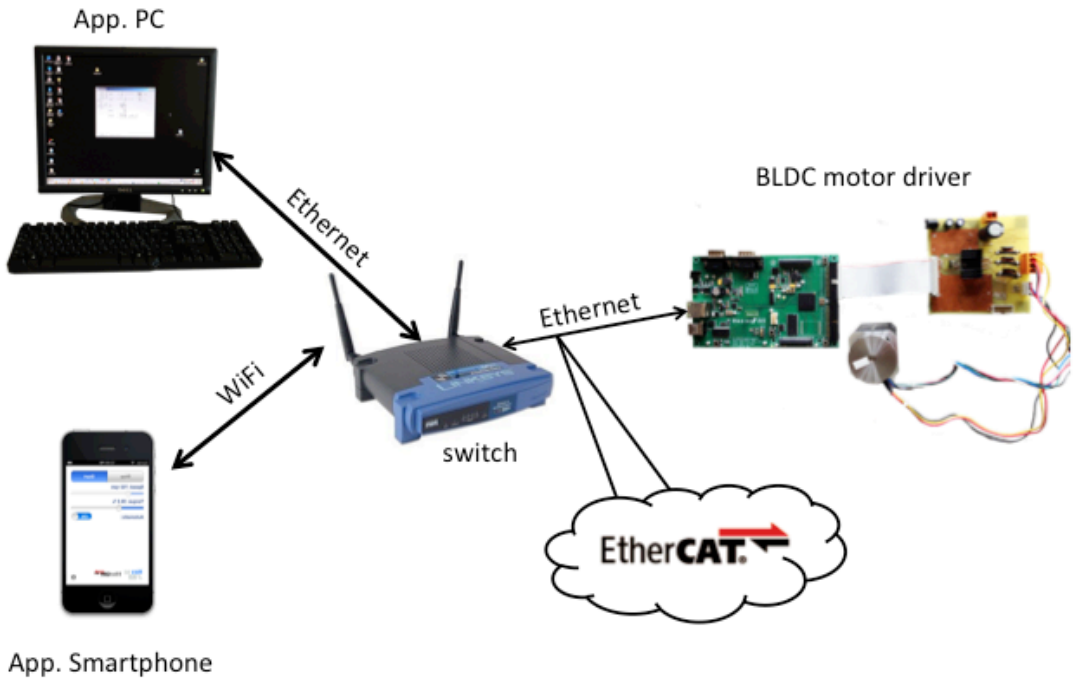


Figure 1: Schéma du système

3. MOTEUR ELECTRIQUE BRUSHLESS

3.1 INTRODUCTION

Le moteur Brushless est un moteur électrique de type synchrone à aimant permanent. Ce moteur est directement dérivé des moteurs à courant continu, mais construit de façon inverse. Il possède un rotor constitué par des aimants permanents et un stator formé par le bobinage fixe, qui sert à la création du champ magnétique.
Le contrôle du moteur se fait par le biais d'un système électronique qui s'occupe de la commutation de courant dans le bobinage. Habituellement dans les moteurs Brushless sont présents trois capteurs à effet de Hall, ils sont positionnés à un angle mécanique de 60° qui correspond à un angle électrique de 120°. Chaque capteur est synchronisé avec une phase du moteur et reste actif sur une demi-période de rotation de la phase correspondante.

Ils existent deux différents types de moteur :
- Brushless DC (BLDC) ou trapézoïdale.
- Brushless AC (PMSM) ou sinusoïdale.

3.2 MOTEUR BRUSHLESS DC

La principale caractéristique d'un moteur Brushless DC est que la magnétisation du rotor, une fois mise en rotation, génère une tension induite sur les enroulements de type trapézoïdale. Le circuit de commande électronique utilise les signaux provenant des capteurs à effet de Hall pour détecter la position magnétique du rotor. Ceci permet de savoir dans quelle phase il faut activer le courant continu pour entraîner une rotation.

3.3 MOTEUR BRUSHLESS AC

Le moteur de type Brushless AC est une évolution de la version DC. La différence principale est qu'une fois mis en rotation, il génère une tension induite de type sinusoïdal sur les enroulements. Pour obtenir un couple constant sur le rotor, il faut alimenter les enroulements avec un courant de type sinusoïdal. Généralement, la commande de ce type de moteur est effectuée en mesurant en temps réel les courants d'alimentation du moteur pour déterminer la position du rotor.

3.4 RESUME ET APPLICATIONS

Les avantages:

- Elimination des balais d'alimentation (présent dans un moteur DC classique), donc moins d'usure.
- Un bon rapport entre puissance / poids et puissance / dimension.
- Bonnes prestations dynamiques.
- Meilleur refroidissement du bobinage.

Les désavantages :

- Pour fonctionner, a besoin d'un contrôleur électronique intégré.
- Coûts du système de contrôle plus enlevé.
- Technologie peu diffusée.

Une autre caractéristique des moteurs Brushless est qu'un moteur BLDC peut être aussi commandé avec tension d'alimentation sinusoïdale, et inversement pour un moteur PMSM qui peut être commandée avec une tension de type trapézoïdale.

Dans l'industrie le moteur BLDC est souvent utilisé pour des applications où la charge a une grande inertie ou si on doit atteindre des grandes vitesses de rotation. Par exemple, ce moteur est présent dans les machines à laver, dans les ventilateurs, dans les véhicules électriques ou autres.
Le moteur PMSM trouve des applications dans les systèmes industriels de positionnement et de contrôle des axes.

3.5 MESURE DE LA TENSION INDUITE

Pour déterminer le type de construction d'un moteur de type Brushless, il faut effectuer une mesure de la tension induite en mettant en rotation le rotor. Pour les moteurs ayant un bobinage de type triangle, il suffit de mesurer avec un oscilloscope la tension induite entre deux pôles du moteur. Pour les moteurs ayant un bobinage de type étoile, il faut procéder à une reconstruction du point de masse à l'aide de résistances connectées en étoile. Ensuite il faut mesurer la tension aux bornes d'une des résistances, comme démontré dans la figure suivante.

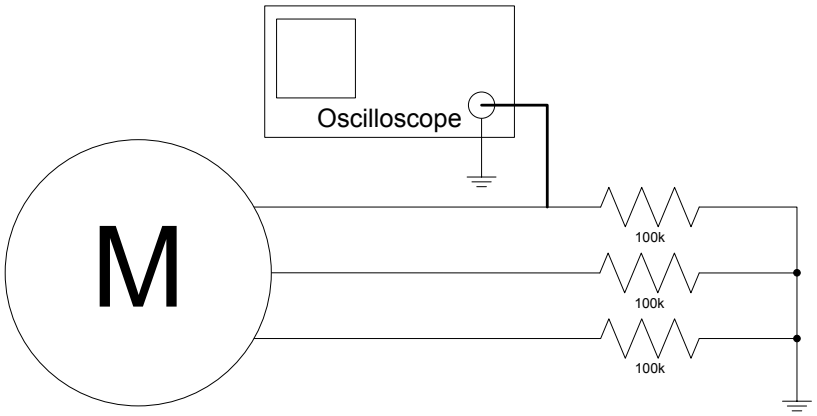


Figure 2: Mesure de la tension induite (bobinage étoile)

D'après, la mesure on peut vérifier si la tension induite sur les enroulements est de type sinusoïdal ou trapézoïdal.

Notes :

L'appellation des différents types de moteurs Brushless n'est pas normalisée, ceci pourrait entraîner des confusions.

Les caractéristiques mécaniques du moteur employé servent à définir le type de signal à générer par le contrôleur numérique. Celui-ci est décrit en détail au chapitre 6.3.

4. ETHERCAT

4.1 INTRODUCTION

EtherCAT est un bus de terrain temps-réel basé sur une couche physique Ethernet. Sur un réseau EtherCAT, un maître génère une trame qui transite entre les différents périphériques connectés. Cette trame contient différentes commandes qui sont destinées aux périphériques connectés au réseau. Quand une trame transite dans un périphérique du bus, les différentes commandes adressées au périphérique sont traitées, et les registres mis à jour. Une fois complétée le tour des périphériques, la trame retourne au maître du bus pour être analysée.
Si on veut effectuer le même travail avec le protocole UDP, on serait obligé d'envoyer une trame à chaque nœud du réseau. Par exemple, si on veut mettre à jour un registre à 16bit dans 4 différents nœuds, avec une commande de 14 Bytes en utilisant UDP, on doit envoyer 4 trames d'environ 60 Bytes qui constitue un totale de 240 Bytes. Pour effectuer le même travail avec EtherCAT, on doit envoyer une seule trame de 88 Bytes. On peut directement déduire donc que pour ce genre d'application EtherCAT est plus performant.

4.2 STRUCTURE D'UN RESEAU

Les différents périphériques sont directement connectés entre eux. Ceci permet d'éviter l'utilisation de *Switch* ou *Hub* pour créer un réseau EtherCAT. Ce réseau peut assumer deux différents types de structure en même temps, à arbre et à étoile. Dans la figure suivante on peut voir un exemple de réseau EtherCAT.

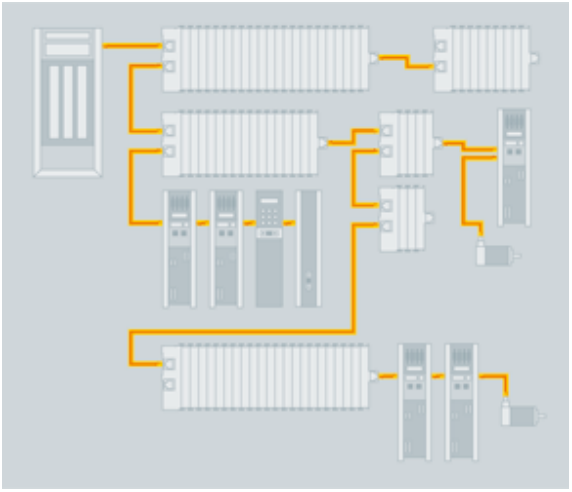


Figure 3: Topologie réseau EtherCAT [Référence 3]

Les paquets de données qui circulent dans le réseau vont passer dans tous les différents périphériques avant de retourner à l'unité principale. Un périphérique possède au moins deux ports Ethernet. Les données qui sont réceptionnées sur le premier port, après avoir été traitées, sont renvoyées sur le deuxième port. Ceci permet la circulation entre les différents périphériques. Si sur le port suivant il n'y a pas un périphérique connecté, le paquet de données est directement envoyé au port suivant. Ceci donne une robustesse majeure au réseau car, si une ligne n'est pas disponible, le paquet est directement renvoyé sur la ligne de provenance et ainsi peut retourner au maître du réseau.
Les réseaux EtherCAT supportent aussi les autres différents services basés sur Ethernet, car si on connecte un périphérique Ethernet sur un *Switch* d'un réseau EtherCAT, les trames Ethernet sont envoyées via le protocole EtherCAT. Ceci n'affecte pas les capacités temps-réel du réseau.

4.3 COMPOSITION D'UNE TRAME

La trame EtherCAT est composée par un en-tête et différentes commandes destinées aux périphériques appartenant au réseau. L'avantage de ce protocole est que le trafic de données sur une ligne est diminué de manière importante car une seule trame envoyée contient plusieurs commandes destinées à différents périphériques. Ceci nous permet de mettre à jour une grosse quantité de registres distribués dans plusieurs périphériques en très peu temps. Par exemple, pour pouvoir mettre à jour 1000 entrée/sorties digitales, il faut compter un temps de 30us.
Une trame EtherCAT peut aussi être insérée dans des paquets UDP, pour permettre la transmission sur des réseaux standard Ethernet. Dans la figure suivante est visible la composition d'une trame EtherCat.

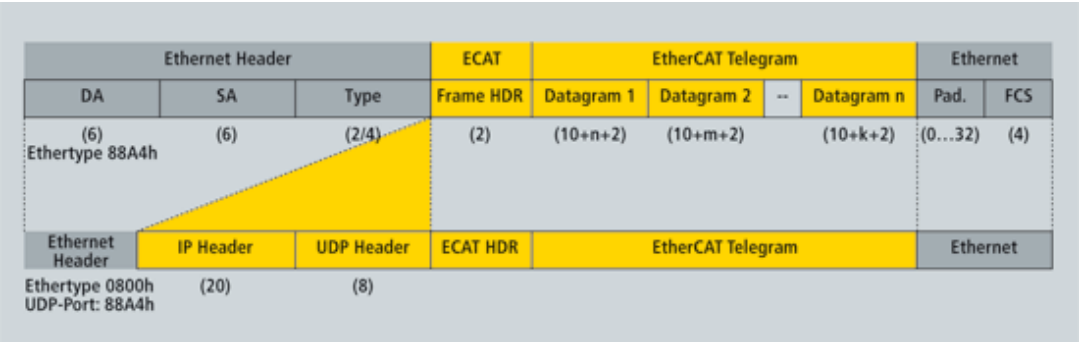


Figure 4: Composition d'une trame EtherCAT [Référence 3]

Notes :

Les détails liés à la composition des trames EtherCAT sont de caractère confidentiel, donc ne peuvent pas être mentionnés dans ce rapport. Pour des informations plus approfondies, il est possible de visiter le site internet de l'organisation EtherCAT (www.ethercat.org).

Un approfondissement lié à la technologie EtherCAT est faite dans les chapitres 6.5 et 7.2 ou le protocole est employé pour le système de commande du contrôleur.

5. CIRCUIT DE PUISSANCE

5.1 SCHEMA BLOC

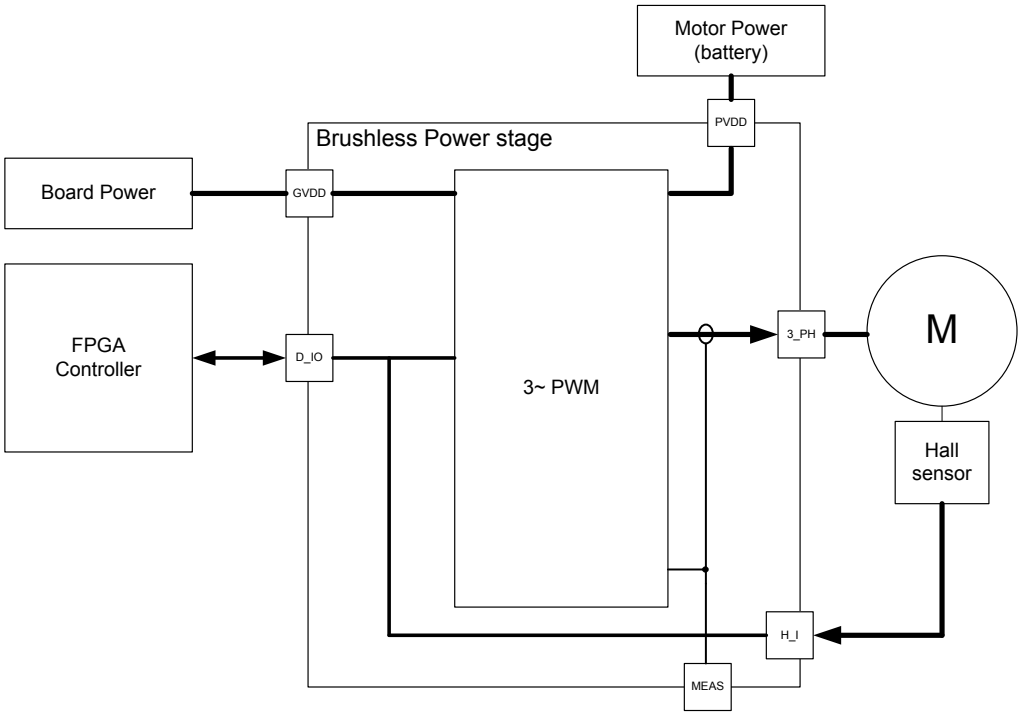


Figure 5: BLDC power stage

La fonction principale du circuit de puissance est de transformer la commande PWM en signal sinusoïdale triphasé. Il sert à piloter des moteurs de type Brushless.

Le circuit possède les interfaces suivantes :

Interface	Description
GVDD	Alimentation principale du circuit (12V DC)
PVDD	Alimentation étage puissance (max. 50V DC)
D_IO	Interface pour circuit numérique de commande (3.3V DC)
H_I	Interface pour capteur à effet de Hall (Alim. sondes 12V DC)
3_PH	Sortie de puissance 3 phase (max. 8A)
MEAS	Points de mesure de courant de phase

Les détails du circuit sont présents dans le rapport du projet de semestre dans le CD en annexe.

5.2 COMMANDE

Pour la commande des trois demi ponts en H, il faut fournir un signal de type PWM avec une fréquence maximale de 500kHz. Pour la commande d'un demi pont en H, avec un rapport cyclique supérieur à 50%, le circuit fournit un courant positif. Pour un rapport cyclique inférieur à 50%, le circuit fournit un courant négatif.
Chaque demi pont en H possède une broche de reset négative pour contrôler son état d'activation. Le contrôleur possède aussi deux broches de signalisation de défaut, une qui indique une coupure due à une surchauffe du système (OTW) et l'autre qui indique une coupure due à un court-circuit (FAULT).

6. CIRCUIT DE COMMANDE NUMERIQUE

6.1 INTRODUCTION

Le circuit de commande numérique est divisé en trois principales parties :

- Une interface de communication, qui sert à l'échange de données entre le PC de commande et le nœud EtherCAT.
- Un contrôleur qui gère la lecture et l'écriture des données dans les différents registres.
- Un générateur de commande pour le moteur qui crée une commande sinusoïdale triphasée sous forme de PWM.

6.2 SCHEMA BLOC

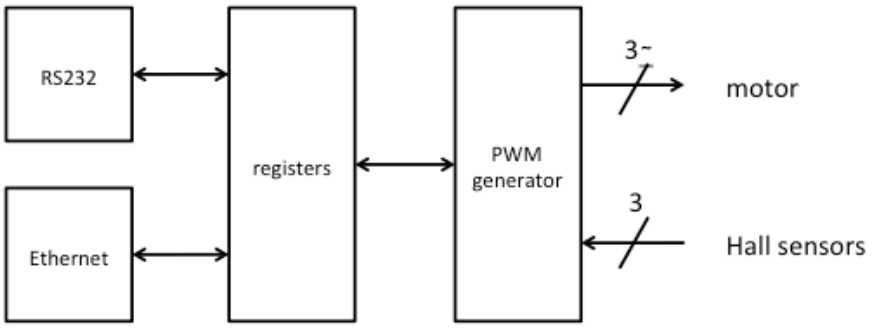


Figure 6: Circuit de commande numérique

Le schéma détaillé se trouve en annexe (Cf. annexe 1).

6.3 GENERATION DE LA COMMANDE PWM

Le générateur de commande PWM a deux modes de fonctionnements. Le premier est en mode manuel où il est possible régler la vitesse et l'amplitude du signal de sortie. Le deuxième est en mode de régulation automatique où on fixe une valeur d'amplitude et le contrôleur s'occupe de régler la vitesse du signal par rapport à celle qui est mesurée sur le moteur.

6.3.1 MODE MANUEL

Le bloc de contrôle lit la valeur de pas depuis le registre de la vitesse. Celle-ci est passée au compteur de phase qui génère la phase à la vitesse désirée. Ensuite la valeur de la phase est passée au générateur de sinus (CORDIC) qui donne en sortie une valeur de sinus avec l'amplitude configurée dans le registre de l'amplitude.

6.3.2 MODE AUTOMATIQUE

Pour le mode automatique, la génération de commande PWM est en partie identique au mode manuel. En effet, le bloc de contrôle active une boucle de réglage pour la commande de pas à donner au compteur de phase.
Un détecteur de phase mesure la différence entre celle qui est générée et celle qui est reconstruite à l'aide des sondes à effet de Hall présentes sur le moteur. La différence est ensuite additionnée à l'ancienne valeur de pas pour créer la nouvelle consigne à donner au générateur de phase.

6.3.3 DIMENSIONNEMENT

Il existe différentes possibilités pour le dimensionnement des paramètres de base du système :

- Fixer les paramètres de dimensionnement lié au moteur choisi, de manière à créer une architecture fixe.
- Fixer une partie des paramètres adaptés à une certaine gamme de moteur et à l'aide de registres de configuration, gérer les paramètres du moteur choisi.
- Créer une architecture dynamique qui permet la configuration complète du système à l'aide de registres de configuration.

Pour le dimensionnement du système j'ai choisi la solution qui consiste à fixer une partie des paramètres et gérer ceux qui sont liés au moteur choisi en les inscrivant dans les registres de configuration. En comparaison, cette solution est meilleure que celle qui fixe tous les paramètres. En effet, ce choix permet de changer de moteur sans avoir besoin de recompiler l'architecture.

Si on crée une architecture dynamique qui permet la modification de la totalité des paramètres de fonctionnement, l'utilisation des ressources de la carte FPGA est trop élevée au vu des avantages qu'on peut en tirer. Par contre, l'utilisation des registres de configuration permet de créer un système plutôt simple qui peut être adapté assez facilement à l'aide d'un logiciel, dans l'ordinateur de commande. Celui-ci calcule les valeurs de limitation à partir des paramètres du moteur. Le choix effectué me semble être un bon compromis.

6.3.4 MODULATION PWM

La spécification du circuit dit que si le cycle du signal PWM est plus grand que 50% on devrait générer un courant positif et si le cycle du signal PWM est inférieur à 50% on devrait générer un courant négatif. Donc pour la génération du signal PWM une comparaison est effectuée entre la valeur d'un compteur qui compte en boucle à la fréquence du système, et la valeur du signal sinusoïdal généré. Ceci implique que la résolution de la valeur d'amplitude du signal sinusoïdal est liée à celle du compteur. La fréquence maximale est donnée par le circuit intégré, qui contrôle la partie de puissance et est fixée à un maximum de 500kHz. Pour garder une résolution idéale de la commande d'amplitude, la fréquence PWM est fixée à une valeur de 16kHz.

Pour le calcul de résolution :

$$f = \frac{f_{clk}}{2^n}$$

Si on fixe :

$$f = 16\ kHz \quad f_{clk} = 66\ MHz$$

On trouve :

$$n = \frac{\ln \frac{66\ MHz}{16\ kHz}}{\ln 2} = 12{,}010$$

Pour une fréquence PWM de 16kHz, la résolution du compteur doit être fixée à 12 bits.

6.3.5 COMPTEUR DE PHASE

Le compteur de phase sert à générer la phase du système. Pour changer la vitesse de comptage il suffit de modifier la valeur de pas du compteur (en modalité automatique, cette valeur est gérée par la boucle de réglage). Pour déterminer la résolution de ce compteur on détermine que pour une fréquence de 100Hz, le pas de comptage doit être environ de 1'000.

Pour le calcul de résolution :

$$f = f_{clk} \cdot \frac{step}{2^n}$$

Si on fixe :

$$f = 16\ kHz \quad f_{clk} = 66\ MHz \quad step = 1'000$$

On trouve :

$$n = \frac{\ln \frac{66\ MHz \cdot 1'000}{100\ Hz}}{\ln 2} = 29.298$$

Pour le compteur de phase, la résolution est proche de 29 bits. Avec cette approximation, pour faire tourner le moteur à la fréquence désirée, il faut insérer dans l'application de contrôle les calculs qui permettent de transformer les tours par minute, en pas de comptage.

Calcul de la fréquence :

$$f = \frac{rpm \cdot pp}{60}$$

Ou :

- rpm : tours par minute
- pp : paires de pôles

Calcul du pas de comptage :

$$pas = \frac{f \cdot 2^{29}}{f_{clk}}$$

A l'aide de ces formules, dans l'application de contrôle sur l'ordinateur de commande, il est possible de configurer une interface utilisateur qui permet de régler la rotation du moteur en tours par minute.

6.3.6 FILTRE PASSE-BAS

Pour éviter des changements trop rapides dans les valeurs de configuration de la vitesse de phase et l'amplitude du signal, des filtres passe-bas de 1er ordre sont ajoutés.

Dimensionnement :

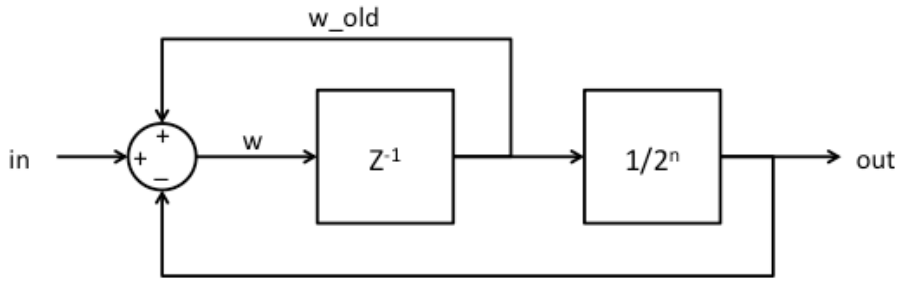


Figure 7: Schéma filtre passe-bas

Fonction de transfert :

$$\frac{out}{in} = \frac{1}{1 + s \cdot h \cdot 2^n}$$

On peut en sortir le calcul de la constante de temps :

$$T_{filtre} = h \cdot 2^n = \frac{2^n}{f_{clk}}$$

En fixant un temps de réaction de 1 seconde et avec la fréquence du système à 66MHz, on trouve que n est équivalent à 26.

6.3.7 DETECTEUR DE PHASE

Grâce aux sondes à effet Hall présentes sur le moteur, on peut déterminer à quel moment la valeur de chaque phase du moteur se trouve exactement à la moitié de la période. Dès qu'un flanc montant est détecté sur le signal d'une sonde, on calcule la différence en soustrayant la moitié de la phase de la valeur de phase actuelle.

6.3.8 BOUCLE DE REGLAGE

En analysant la boucle de réglage, on se rend compte que la structure est similaire à celle d'une boucle à verrouillage de phase (ou PLL).

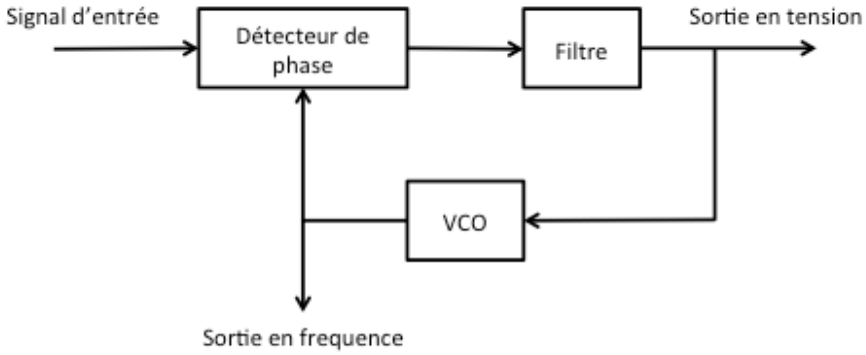


Figure 8: Schéma de principe de la PLL

En fait, le signal qui provient du capteur à effet Hall rentre dans un détecteur de phase (PD) qui, à la sortie fournit un valeur de différence de phase. Ensuite, la valeur est passée dans un filtre qui sert à stabiliser la boucle et le signal résultant est inséré dans un compteur de phase. Dans la figure suivante, les détails de la boucle de réglage sont visibles.

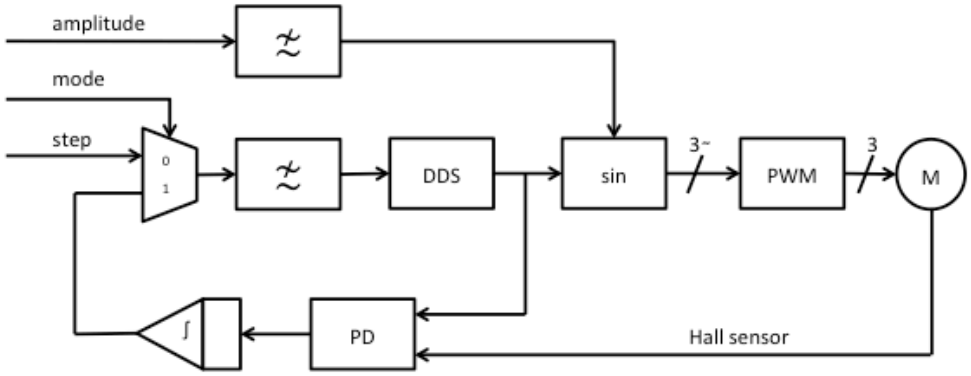


Figure 9: Boucle de réglage

Le schéma détaillé se trouve en annexe (Cf. annexe 2).

6.4 REGISTRES DE CONFIGURATION

Le système dispose de différents registres qui permettent de configurer les paramètres de fonctionnement et contrôler l'état de fonctionnement. Ces différents registres sont connectés à un bus de données contrôlé par un maître qui décide quelle interface a le droit de lire ou écrire.

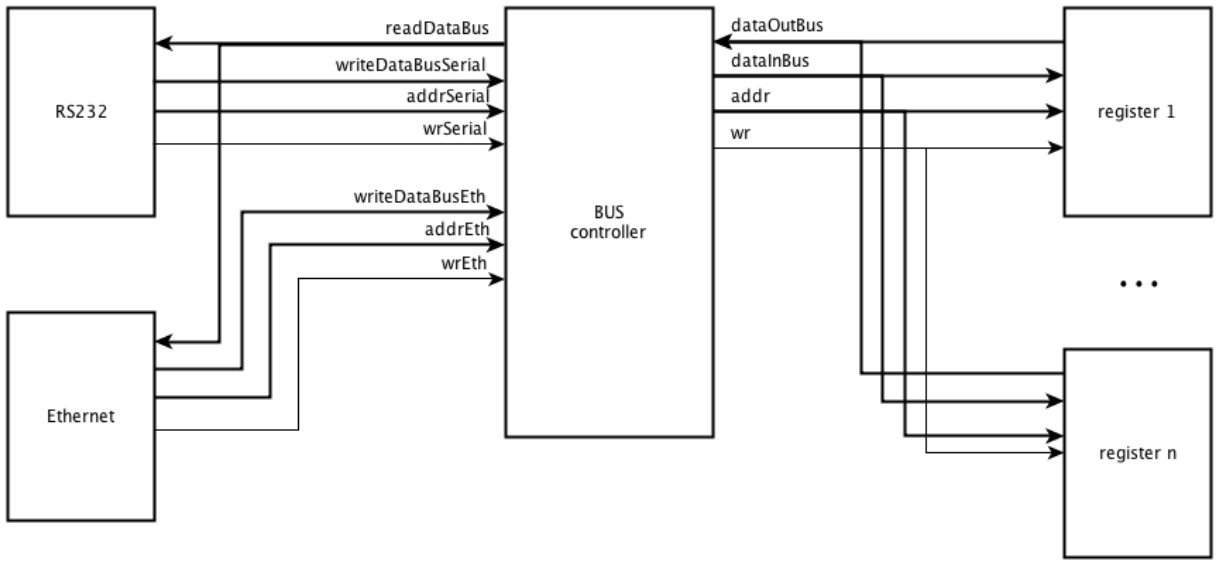


Figure 10: Structure du bus de données

Le tableau suivant présente un résumé des registres présents :

Groupe	Adresse	Access	Description
Base	0x01	R	Etat du système
	0x02	R/W	Contrôle du système
Configuration	0x03	R/W	Limite de vitesse du signal
	0x04	R/W	Limite d'amplitude du signal
Fonctionnement	0x05	R/W	Réglage de la vitesse du signal
	0x06	R/W	Réglage de l'amplitude du signal
Contrôle	0x07	R	Vitesse du signal
	0x08	R	Vitesse du moteur
	0x09	R/W	Rotation du moteur

Le schéma détaillé se trouve en annexe (Cf. annexe 3).

6.4.1 REGISTRES DE BASE

Les registres de base du système permettent de contrôler le fonctionnement des blocs de base du système.

Etat du système (0x01 – R)

Le registre d'état du système permet de contrôler les bits d'erreurs principales du système.

7							0
-	-	-	-	-	P_LOCK	OTW	FAULT

FAULT :

Si l'état de ce bit passe à 0, il y a eu une coupure du système due à une surcharge de courant sur le moteur.

OTW :

Si l'état de ce bit passe à 0, il y a eu une surchauffe du système.

P_LOCK :

Si l'état de ce bit passe à 1, la phase du moteur est alignée à la phase générée par le système.

Contrôle du système (0x02 – R/W)

Le registre du contrôle du système permet de contrôler le fonctionnement de la génération du signal et du circuit de puissance.

7				3	1	0
-	-	DIR	REG_EN	RSTN_X		MOT_EN

MOT_EN :

Mettre ce bit à 1 pour activer la génération du signal PWM triphasée.

RSTN_X [3 : 1] :

Mettre ces bits à 1 pour activer le fonctionnement des trois demi ponts en H présents sur le circuit de puissance. Si un de ces bit de trouve à 0, le demi pont en H correspondant se trouve en état de haute impédance.

REG_EN :

Mettre ce bit à 1 pour activer la régulation de couple automatique du système.

DIR :

Ce bit contrôle la direction de rotation du moteur, à l'état 1 le moteur tourne en sens horaire.

6.4.2 REGISTRES DE CONFIGURATION

Les registres de configuration permettent de configurer les limites de fonctionnement du système. Ces limites sont à configurer en fonction du moteur choisi.

Limite de vitesse du signal (0x03 – R/W)

Ce registre permet de configurer la limite du pas de comptage lié à la vitesse du signal. Cette limite est à calculer en relation avec les caractéristiques du moteur.

15				11	8
-	-	-	-	STEP_LIM	

7	0
STEP_LIM	

STEP_LIM [11 : 0] :

Cette valeur limite le pas maximal qu'on peut mettre dans le registre. Pour le calcul il faut utiliser la formule suivante :

$$step\_\lim = \frac{\frac{rpm \cdot pp}{60} \cdot 2^{29}}{f_{clk}}$$

Ou :

- rpm : nombre de tours maximal par minute du moteur
- pp : paires de pôles du moteur
- f_{clk} : fréquence de *clock* du système

Limite de amplitude du signal (0x04 – R/W)

Ce registre permet de configurer la limite de l'amplitude du signal générée. Cette valeur limite l'amplitude maximale de la tension de sortie qui est à calculer en rapport à la tension d'alimentation du moteur.

15				11	8
-	-	-	-	AMPL_LIM	

7	0
AMPL_LIM	

AMPL_LIM [11 : 0]:

Cette valeur limite l'amplitude maximale de la tension de sortie. Pour le calcul, il faut utiliser la formule suivante :

$$ampl\_lim = \frac{U}{U_{alim}} \cdot 0.6071 \cdot 2^{11}$$

Ou :

- U : Tension de sortie
- U_{alim} : Tension alimentation partie de puissance

6.4.3 REGISTRES DE FONCTIONNEMENT

Les registres de fonctionnement permettent de configurer les paramètres directement liés au fonctionnement du moteur.

Réglage de vitesse du signal (0x05 – R/W)

Ce registre permet de configurer le pas de comptage lié à la vitesse du signal.

15				11	8
-	-	-	-	STEP	

7	0
STEP	

STEP [11 : 0] :

Cette valeur règle le pas de comptage du compteur de phase quand le système fonctionne en modalité manuelle. Pour le calcul de la valeur en fonction de la vitesse utiliser la formule suivante :

$$step = \frac{\frac{rpm \cdot pp}{60} \cdot 2^{29}}{f_{clk}}$$

Ou :
- rpm : nombre de tours par minute du moteur
- pp : paires de pôles du moteur
- f_{clk} : fréquence de *clock* du système

Réglage de amplitude du signal (0x06 – R/W)

Ce registre permet de configurer la limite de l'amplitude du signal générée. Cette valeur limite l'amplitude maximale de la tension de sortie, à calculer en rapport à la tension d'alimentation du moteur.

15				11	8
-	-	-	-	AMPL	

7	0
AMPL	

AMPL [11 : 0]:
Cette valeur limite l'amplitude maximale de la tension de sortie, le calcul suivant permet de calculer la valeur à mettre selon le % choisi :

$$ampl = \frac{p}{100} \cdot 0.6071 \cdot 2^{11}$$

Ou :
- p : valeur en % de l'amplitude désirée du signal.

6.4.4 REGISTRES DE CONTROLE

Les registres de contrôle permettent de lire les différents paramètres liés au fonctionnement du moteur.

Vitesse du signal (0x07 – R)

Ce registre permet de lire la valeur de pas de comptage utilisée pour le compteur de phase.

15				11	8
-	-	-	-	ACT_STEP	

7	0
ACT_STEP	

ACT_STEP [11 : 0] :
Cette valeur indique le pas de comptage du compteur de phase, pour la transformation en tours par minute il s'agit de se référencier à la formule du registre de réglage de vitesse (0x05).

Vitesse du moteur (0x08 – R)

Ce registre permet de lire la valeur de vitesse du moteur.

15	8
M_SPEED	

7	0
M_SPEED	

M_SPEED [15 : 0] :

Cette valeur indique la quantité de coup d'horloge qui est passée entre les flancs montant des capteurs à effet de Hall. Pour le calcul de la vitesse en tour/minutes, il faut effectuer le calcul suivant :

$$rpm = \frac{m\_speed \cdot 360}{f_{clk} \cdot pp}$$

Ou :

- rpm : tours par minute du moteur
- pp : paires de pôles du moteur
- f_{clk} : fréquence de *clock* du système

Rotation du moteur (0x09 – R/W)

Ce registre permet de lire le nombre de rotations effectué par le moteur.

15							8
M_ROT							

7							0
M_ROT							

M_SPEED [15 : 0] :

La valeur est augmentée ou diminuée de un à chaque flanc montant d'un des capteurs à effet Hall. Pour le calcul des nombres de tours, il faut effectuer le calcul suivant :

$$tours = \frac{m\_rot}{3 \cdot pp}$$

Ou :

- pp : paires de pôles du moteur

6.5 INTERFACES

Le système de contrôle possède deux interfaces de communication différentes : une sérielle de type RS232 et l'autre Ethernet de type 10/100 Mbps. Ces interfaces permettent d'écrire des valeurs dans les différents registres de configurations. Les blocs de communication sérielle et Ethernet proviennent de la librairie de l'école. Le travail a consisté à créer des blocs de décodage des données en entrée.

6.5.1 INTERFACE SERIELLE (RS232)

Les données réceptionnées par la liaison sérielle sont décodées à l'aide d'une machine d'état qui détermine si le Byte de commande est valide. Si la commande est valide, l'opération d'écriture ou de lecture du registre adressé est effectuée. Dans la figure suivante est présente la machine d'état de l'interface :

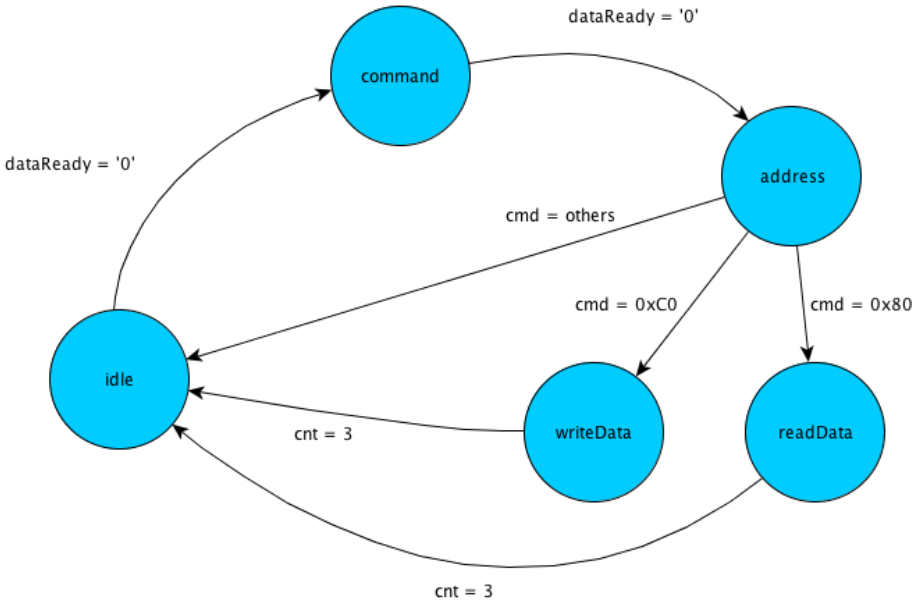


Figure 11: Machine d'état

Paramètres de communication :

- Baud rate : 9600
- Data bits : 8
- Parity bit : none
- Stop bits : 1

6.5.2 COMMANDE SERIELLE

Pour l'interface, un protocole simple de commande a été défini. Il permet de lire ou écrire dans le registre désiré. La commande est composée par un Byte de commande qui décide de l'opération à effectuer, par un autre Byte qui fixe l'adresse de destination et en cas d'écriture suivi par les 4 Bytes de données à écrire.

Commande d'écriture :

	0	1	2	3	4	5
Envoi	0xC0	Addr	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]

Commande de lecture :

	0	1
Envoi	0x80	Addr

	0	1	2	3
Réception	Data[31:24]	Data[23:16]	Data[15:8]	Data[7:0]

6.5.3 INTERFACE ETHERNET (UDP)

L'interface Ethernet est divisée en deux parties : une première qui effectue le traitement de données avec la couche physique et une autre qui s'occupe de l'extraction des données du protocole UDP.
Le premier bloc s'occupe de réceptionner les données provenant de la couche physique, de les transférer dans une mémoire RAM et d'envoyer les données à transmettre sur la couche physique.
Le deuxième bloc en réception lit les données depuis la mémoire RAM pour effectuer l'extraction des informations des différentes couches (Ethernet, IP, UDP) et isoler les données transportées par la couche UDP. Lorsqu'il exécute un envoi, il effectue exactement l'opération inverse, il insère les données dans les différentes couches, et les écrit dans la mémoire RAM d'envoi. Ce bloc permet de filtrer les données en réception par rapport aux adresses MAC, IP ainsi que le port UDP choisi.

Paramètres de communication :

- Adresse MAC : e4:af:a1:39:10:11
- Adresse IP : 192.168.1.2
- Port UDP : 34980

Note :

Le service ARP n'est pas présent, donc pour l'utilisation du système il faut insérer manuellement le routage dans la table ARP de l'ordinateur de commande.

6.5.4 DECODEUR TRAME ETHERCAT

Les données extraites de la trame UDP composent la trame EtherCAT. Pour optimiser la vitesse de traitement, ces données sont directement décodées à l'aide d'une machine d'état qui détermine l'opération à effectuer pour chaque *datagram* EtherCAT reçu. Après le traitement, les données sont directement renvoyées sur la ligne Ethernet.

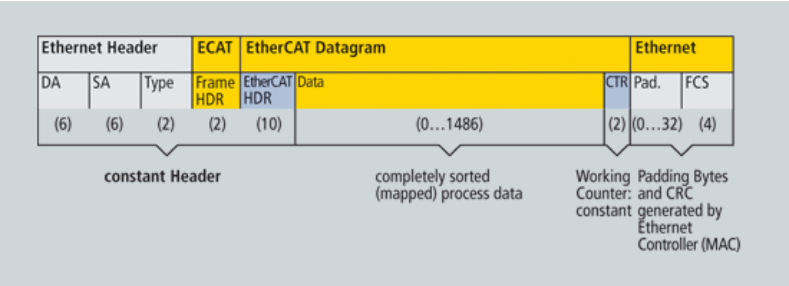


Figure 12: Datagram EtherCAT [Référence 3]

La machine d'état commence par lire la trame EtherCAT (Cf. figure 12), en décodant l'en-tête qui définit le type de données qui suivent. Si la trame reçue est une commande EtherCAT, la machine d'état va lire l'en-tête de chaque *datagram* EtherCAT suivant et traiter les données destinées au nœud. Dans la figure suivante apparaît la structure de la machine d'état.

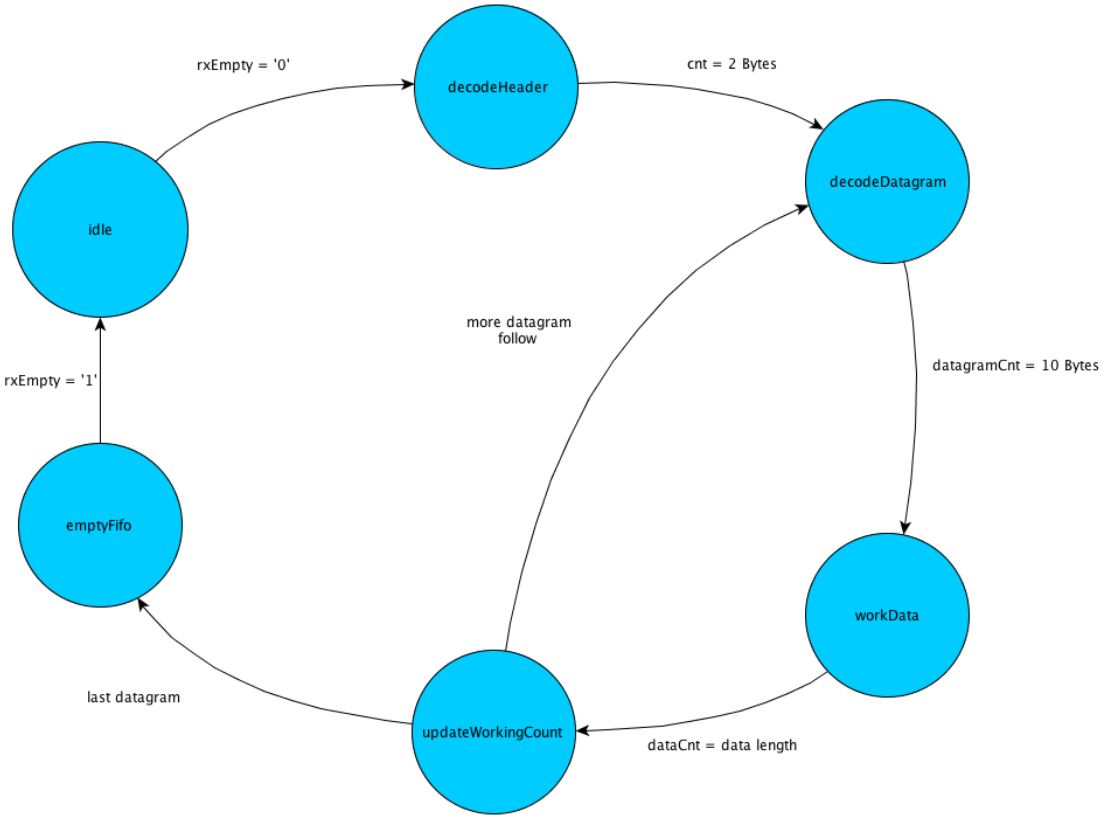


Figure 13: Machine d'état décodeur EtherCAT

Note :

Les détails liés à la composition des trames EtherCAT sont de caractère confidentiel, donc ne peuvent pas être mentionnées dans ce rapport. Pour des informations plus en détail, il est possible de visiter le site internet de l'organisation EtherCAT (www.ethercat.org).

7. APPLICATION DE CONTROLE

7.1 INTRODUCTION

Pour tester le fonctionnement du nœud de commande pour moteur électrique, il est nécessaire de développer une application qui peut fonctionner sur un ordinateur de commande qui agit comme maître du bus EtherCAT. L'application doit être capable d'envoyer des commandes via liaison sérielle ou liaison Ethernet. Les trames transmises sur réseaux Ethernet doivent être sous format EtherCAT.

7.2 APPLICATION PC

Pour la conception de l'application pour ordinateur a été choisi l'environnement de développement Qt [Référence 4]. Il permet la création d'applications multi plateformes basées sur le langage de programmation C++. Grâce à une librairie riche, il permet de développer rapidement des applications avec une simple interface utilisateur et l'emploi des interfaces de communications choisies pour ce travail.

7.2.1 INTERFACE UTILISATEUR

Tous les paramètres d'utilisation sont concentrés sur une seule fenêtre et ceux de configuration sur une autre. Son contenu permet de régler les caractéristiques du moteur, le mode de fonctionnement et la mise en marche du système. Voici l'interface utilisateur.

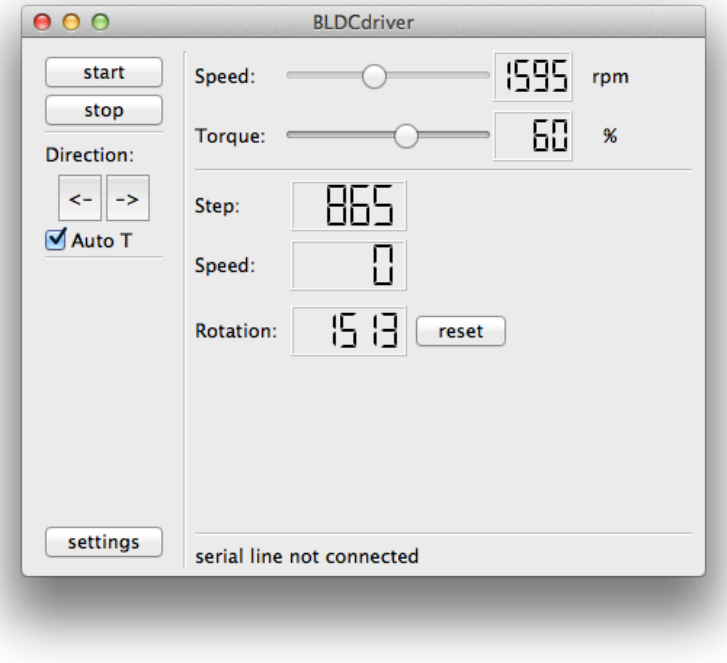


Figure 14: Interface principale application PC

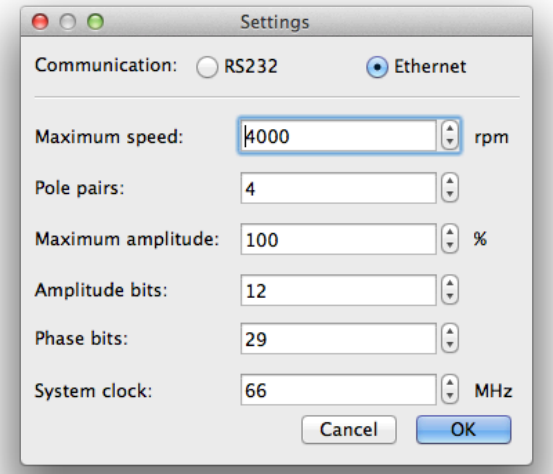


Figure 15: Interface réglages application PC

7.2.2 LIBRAIRIE ETHERCAT

Pour l'application, j'ai développé une librairie capable d'envoyer et de recevoir des trames de EtherCAT sous le protocole UDP. La librairie permet de générer des commandes d'écriture et de lecture dans les registres du nœud adressé et d'effectuer l'envoi de commandes périodiquement selon un intervalle défini.

La librairie EtherCAT que j'ai créée ne contient pas la totalité des fonctionnalités définies par la norme de ce protocole. Seules les fonctionnalités utilisées par le périphérique sont présentes dont la lecture et l'écriture de registres physiques. En annexe, est disponible l'en-tête du code de la librairie où les différentes fonctions sont décrites en détail (Cf. annexe 4).
Grâce au logiciel *wireshark* [Référence 5], j'ai effectué une capture d'une trame envoyée et la réponse correspondante.

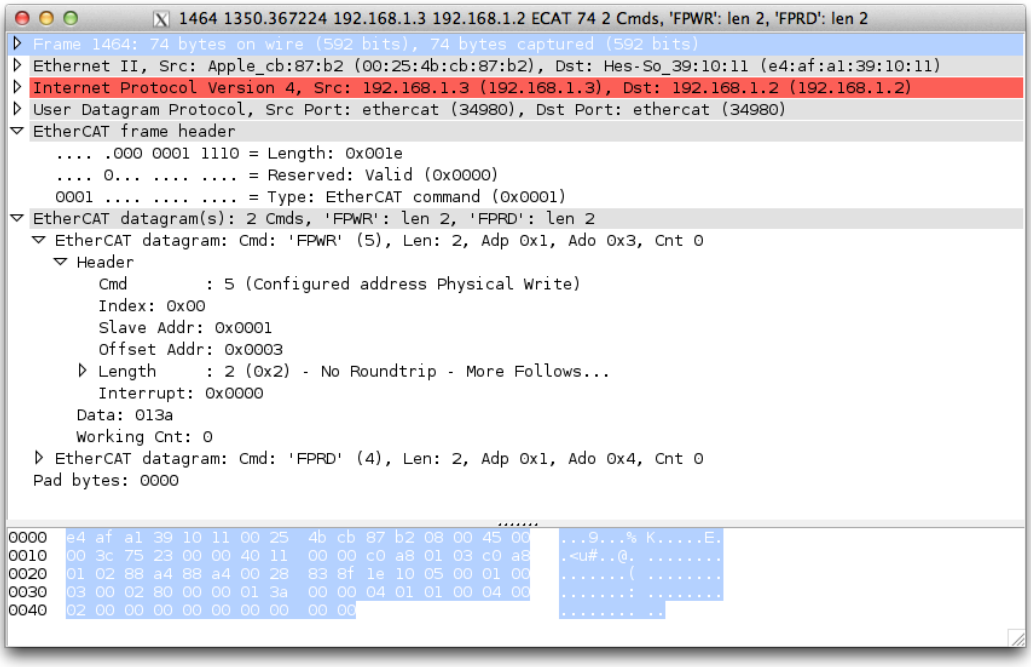


Figure 16: Trame EtherCAT envoyé

La trame envoyée contient deux commandes destinées au périphérique avec adresse 0x1, la première est une commande d'écriture de registre physique 0x3 et le deuxième est une commande de lecture du registre 0x4.

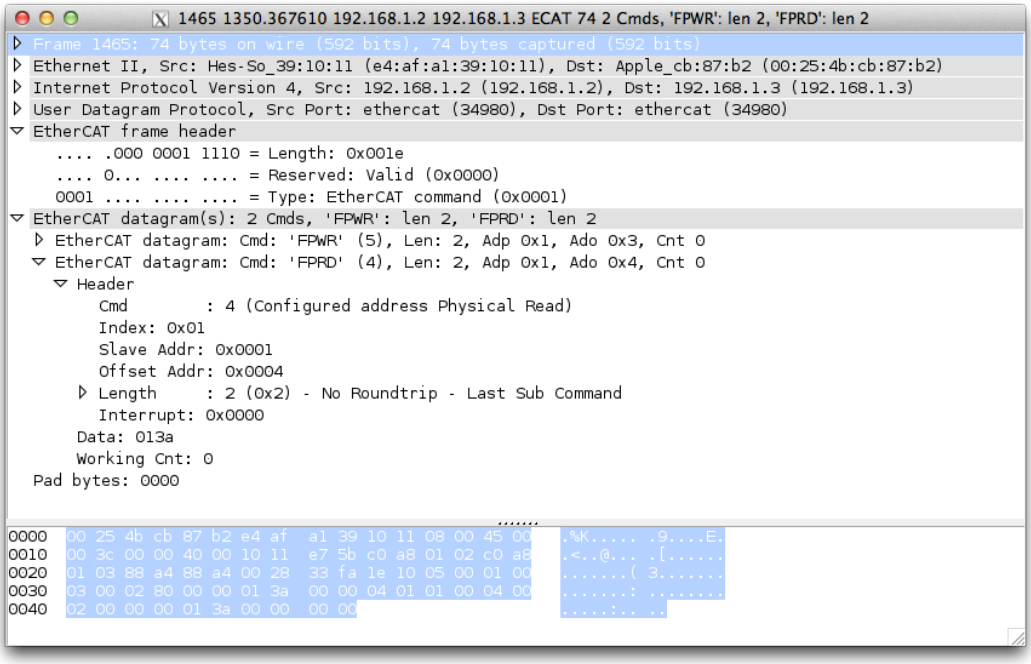


Figure 17: Trame EtherCAT reçue

En réponse, je reçois la même trame, mais avec les données de la commande de lecture modifiées.

À l'aide d'un analyseur logique (Agilent 16803A), j'ai effectué la mesure du temps de transmission et de traitement d'une trame EtherCAT. La trame envoyée contient trois commandes de lecture de registre. La taille de la trame UDP est de 100 Bytes, dont 45 Bytes de commandes EtherCAT. Le bloc Ethernet traite 4 bits à la fois à une fréquence de 25MHz. Pour la réception, on peut donc calculer le temps suivant :

$$t_{trame} = \frac{\frac{100 \cdot 8\,[bits]}{4[bits]}}{25 \cdot 10^6\,[Hz]} = 8\,[us]$$

Dans la figure suivante, on peut voir la mesure effectuée :

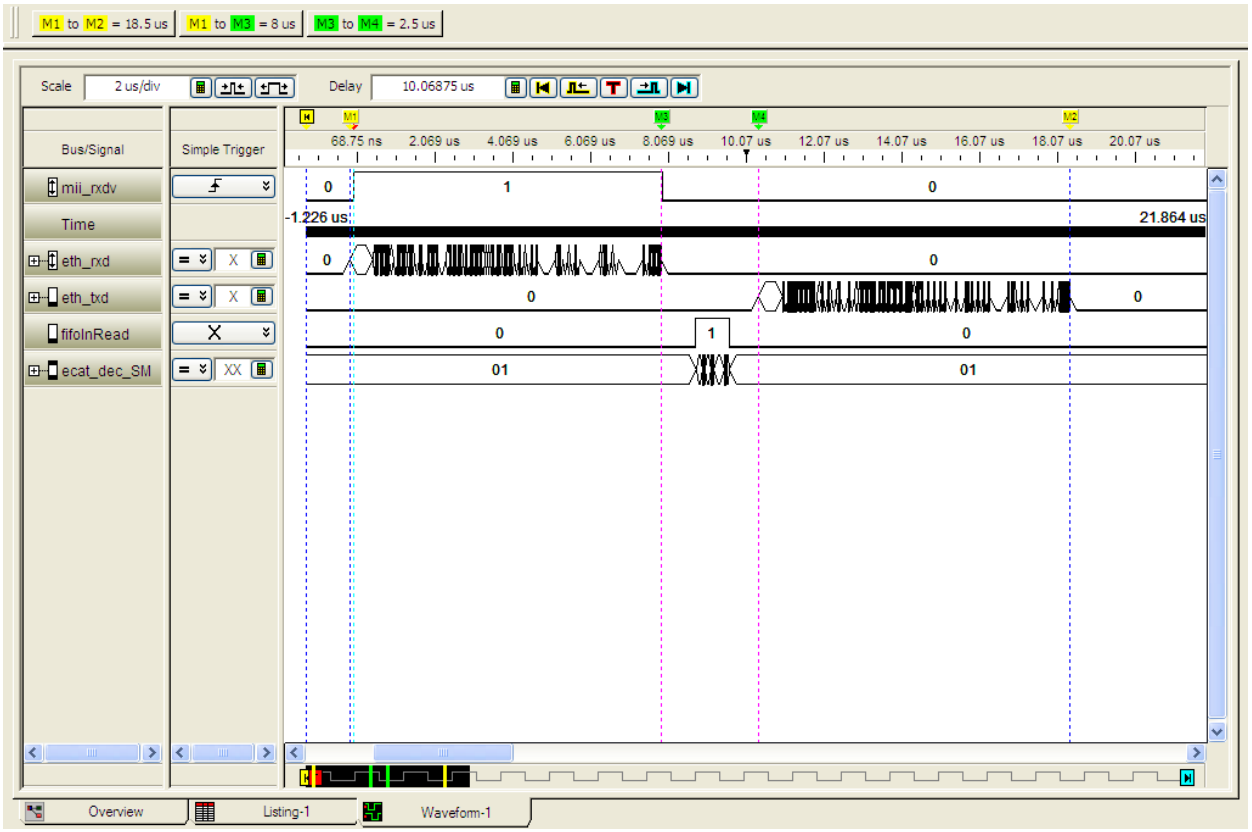


Figure 18: Mesure du temps de transmission d'une trame EtherCAT

De cette mesure on peut retirer que le temps total du traitement de la trame est de 18,5us, dont 8us de temps de réception. Ceci confirme le calcul effectué au préalable. Le temps de traitement des données entre la réception et le renvoi est de 2,5us. Par contre, l'utilisation du protocole UDP pour la transmission des trames EtherCAT augmente le temps de traitement. En utilisant directement le protocole EtherCAT, on obtient un temps de réception de 4,48us. Ce temps représente presque le double de la vitesse de traitement.

7.3 APPLICATION IPHONE

Suite à la conception de l'application pour ordinateur, j'ai opéré les modifications nécessaires pour la rendre compatible avec l'IPhone.

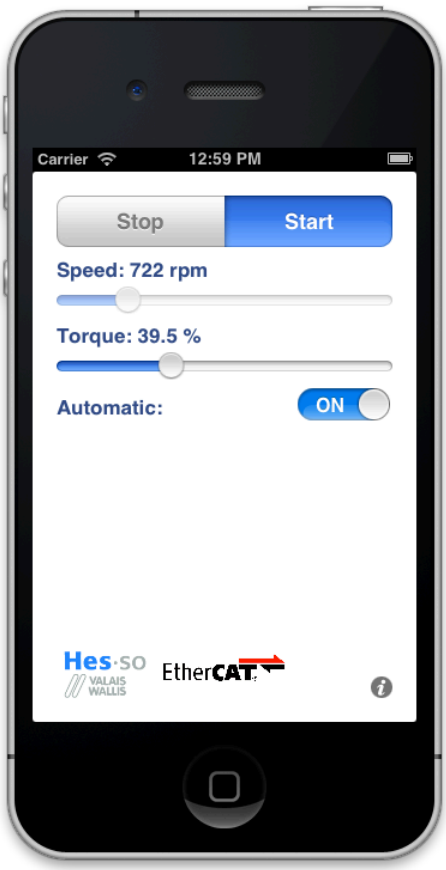


Figure 19: Interface application iPhone

Pour ce faire, j'ai dû traduire la librairie EtherCAT développée sous Qt avec le langage de programmation C++ pour le système de programmation iOS avec le langage de programmation Objective-C. J'ai aussi modifié l'interface utilisateur pour être facilement utilisable sur un écran de petite taille.

8. TEST DU SYSTEME

8.1 COMMANDE PWM

Afin de vérifier le bon fonctionnement du système de génération de commande PWM, un test a été effectué via la liaison sérielle et l'application de contrôle. Avec un oscilloscope, j'ai effectué une mesure du signal généré pour le circuit de puissance. La mesure est visible dans les figures suivantes.

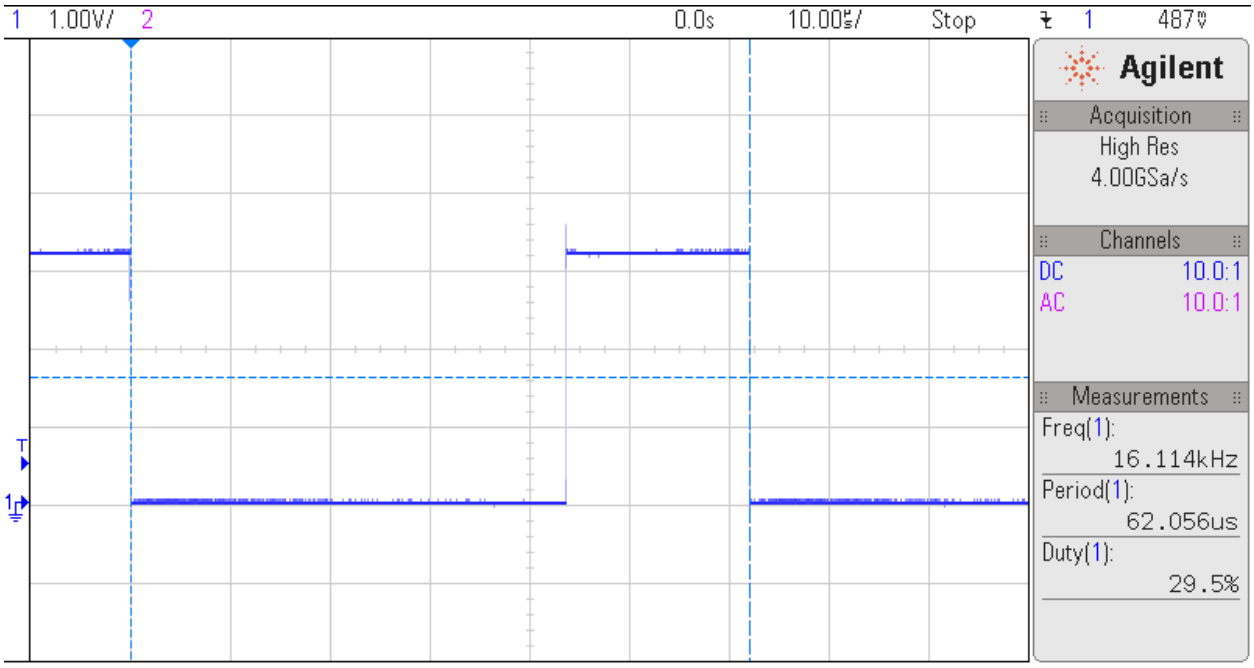


Figure 20: Mesure 1 du signal PWM généré

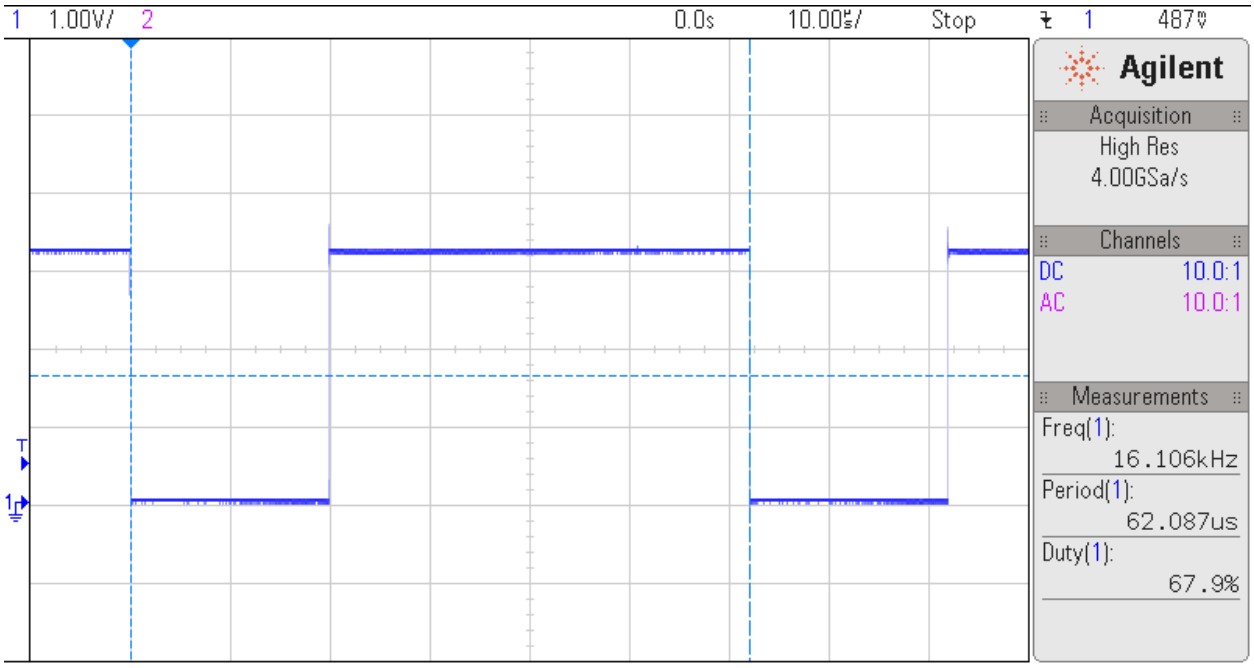


Figure 21: Mesure 2 du signal PWM généré

Dans les mesures, la fréquence du signal est environ de 16kHz. On peut aussi constater que le rapport cyclique varie. Je peux donc affirmer que le système génère une commande de type PWM.

Pour tester si la commande PWM générée correspond à un signal de type sinusoïdal, il faut connecter le circuit électronique de puissance au système. Ensuite, il faut choisir des valeurs fixes à mettre dans les registres de configuration. Pour l'expérience, j'ai choisi une amplitude du signal à 100% de la tension de l'alimentation et une vitesse de rotation de 30 tours par minute (équivalent à 2Hz). Puis j'ai alimenté le circuit de puissance à 5V. Pour mesurer la tension générée, il faut aussi connecter les trois résistances en étoile aux bornes du circuit de puissance où le monteur est connecté normalement. Ensuite, avec un oscilloscope, j'ai mesuré la tension aux bornes des résistances des deux premières phases. La mesure est visible dans la figure suivante.

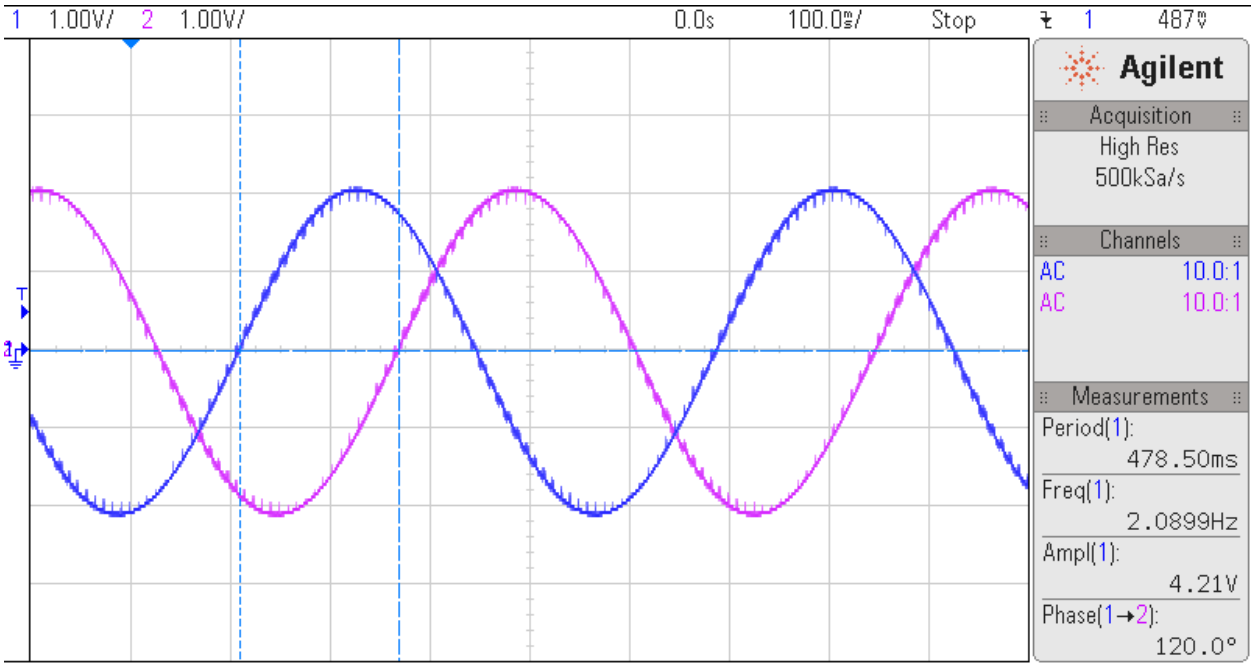


Figure 22: Mesure de la tension sinusoïdale générée

Les signaux sinusoïdaux générés possèdent une fréquence d'environ 2,09Hz, une amplitude d'environ 4.2V et ont un déphasage de 120° électrique. Je peux donc affirmer que le générateur de commande PWM sinusoïdale fonctionne correctement.

8.2 BOUCLE DE REGLAGE

Pour le test du fonctionnement de la boucle de réglage, il faut activer le mode « réglage » du système (figure 22 : « Auto T ») et brancher le moteur BLDC.
Pour le test de la boucle de réglage, j'ai alimenté le moteur avec une tension de 16V DC et j'ai fixé une amplitude à 30% de la tension. On peut le voir dans la figure suivante:

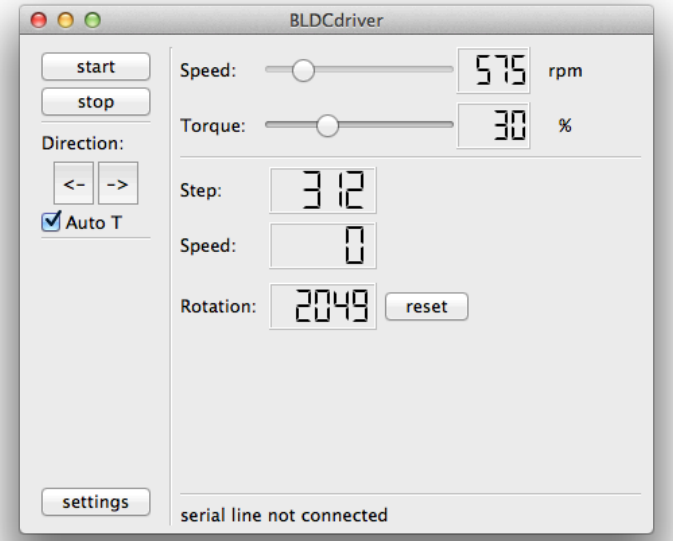


Figure 23: Paramètres du test de la boucle de réglage

Pour effectuer les diverses mesures sur mon moteur, j'ai branché une sonde de courant d'une des entrées analogiques de l'oscilloscope afin de détecter le signal de la première phase (figure 24 : signal 1). Puis avec les entrées digitales de l'oscilloscope, j'ai mesuré plusieurs signaux : le signal représentant le bit de poids fort de la première phase généré par le compteur de phase (figure 24 : signal D_0), le signal du capteur à effet Hall correspondant à la première phase (figure 24 : signal D_1) et le signal de la mise à jour de la valeur de déphasage dans le détecteur de phase (figure 24 : signal D_2).

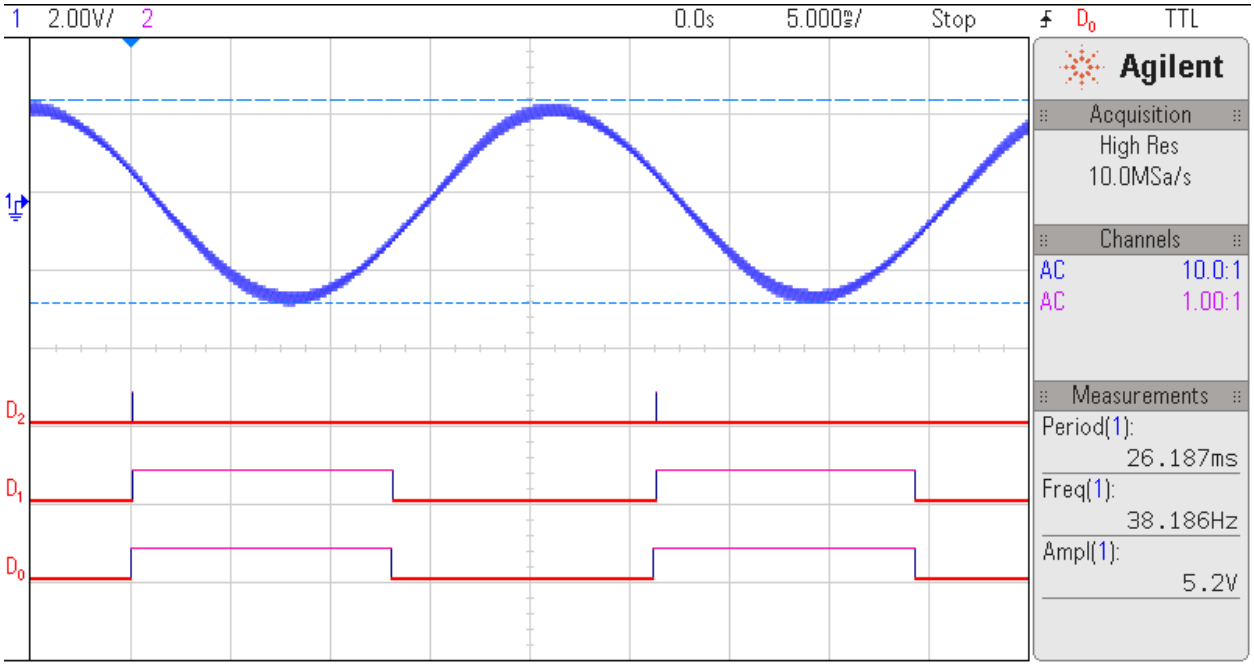


Figure 24: Mesure du comportement de la boucle de réglage

La figure 24 nous montre que le réglage fonctionne correctement, car le signal provenant du capteur à effet Hall (figure 24 : signal D_1) et le signal du bit de poids fort du compteur de phase (figure 24 : signal D_0) sont synchronisé.

Le signal de courant (figure 24 : signal 1) confirme aussi que la mesure de vitesse qu'on peut voir dans l'application (figure 23) est correct car :

$$Tours\ par\ minute = \frac{38.186\ [Hz] \cdot 60\ [s]}{4} = 572.79\ [rpm]$$

Note :

Le coefficient de 4 correspond aux paires de pôles du moteur.

8.3 COMMUNICATION ETHERNET

Pour le test de la communication, j'ai composé une trame EtherCAT ayant deux commandes de lecture de registre dont une est destinée à mon nœud de commande et l'autre à un nœud de commande fictif. À l'aide du logiciel *wireshark* [Référence 5], j'ai pu contrôler si le trafic des trames EtherCAT se déroulait de façon correcte et si le décodeur des trames EtherCAT fonctionnait aussi de manière adéquate. Dans la figure suivante on peut voir la trame de réponse aux commandes envoyées.

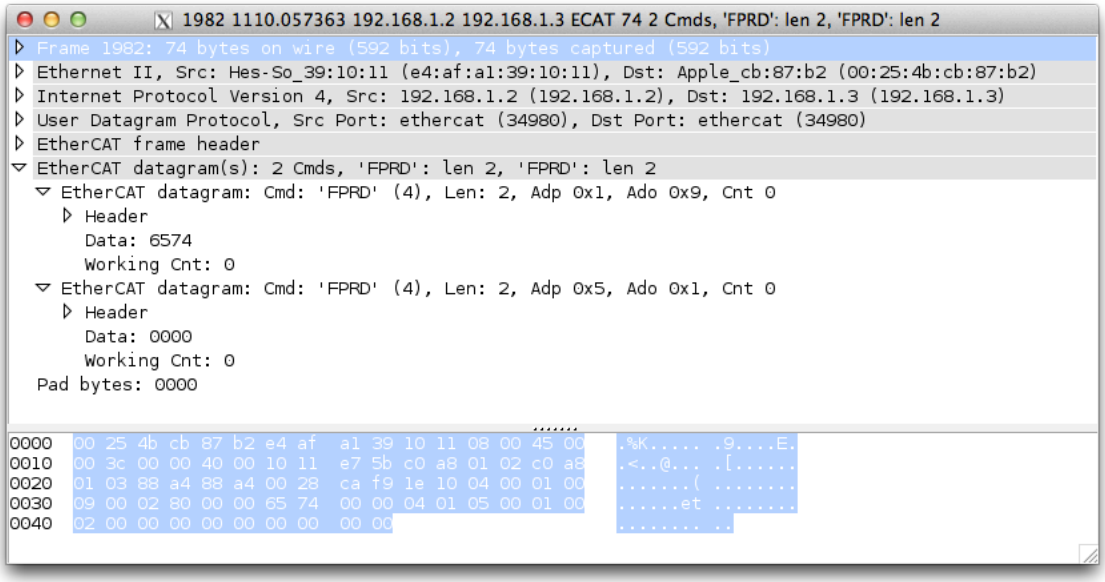


Figure 25: Test d'envoie trame EtherCAT mixte

La première commande envoyée est la lecture de registre du nombre de rotation (Figure 25 : Adp 0x1 – Ado 0x9). On peut voir ici que la valeur est mise à jour correctement. La deuxième commande est aussi une lecture de registre (Figure 25 : Adp 0x5 – Ado 0x1). Mais dans ce cas, elle est destinée à un autre nœud. On peut constater que la commande n'est pas modifiée donc le décodeur de trames EtherCAT fonctionne correctement.

Dans le récepteur Ethernet, un défaut est encore présent : après la réception d'environ 4'000 à 8'000 trames de type diffèrent, le récepteur arrête de retransmettre les trames en entrée au décodeur EtherCAT. Après une vingtaine d'essais, le même problème est toujours rencontré. Lors de mes recherches sur la cause de celui-ci, j'ai pu découvrir où se situait l'erreur. En effet, lorsque la mémoire RAM de réception est pleine, le bit qui indique que la réception est complétée n'est pas mise à jour dans l'entête du bloc mémoire alloué à la dernière trame reçue. Mais malheureusement, je n'ai pas la réponse sur les causes elles-mêmes. Les éventuelles causes restent donc toujours inconnues pour le moment.

8.4 DEFAUT DU SYSTEME DE REGLAGE

Le système présente un petit défaut. Si on se trouve en mode de réglage automatique au premier démarrage, le moteur ne démarre pas après l'augmentation de la consigne d'amplitude. Ce problème est dû au fait que le signal du pas de comptage est nul. Cela implique donc que la phase ne varie pas. Ce problème peut être résolu en écrivant une valeur dans le registre de réglage de la vitesse (Registre numéro 0x05).

9. AMELIORATIONS

9.1 BOUCLE DE REGLAGE

La boucle de réglage actuelle du système peut être améliorée en substituant le filtre passe-bas contenu dans la PLL par un filtre à intégrateur PI. Les avantages de ce filtre sont :

- Excellente mémoire de fréquence lorsque le signal d'entrée disparaît.
- Rampe de fréquence du signal d'entrée autorisée.
- Disparition de l'erreur statique après le changement de fréquence ou de phase d'entrée.

Dans la figure suivante on peut voir les modifications à effectuer dans la boucle de réglage pour insérer le filtre à intégrateur PI.

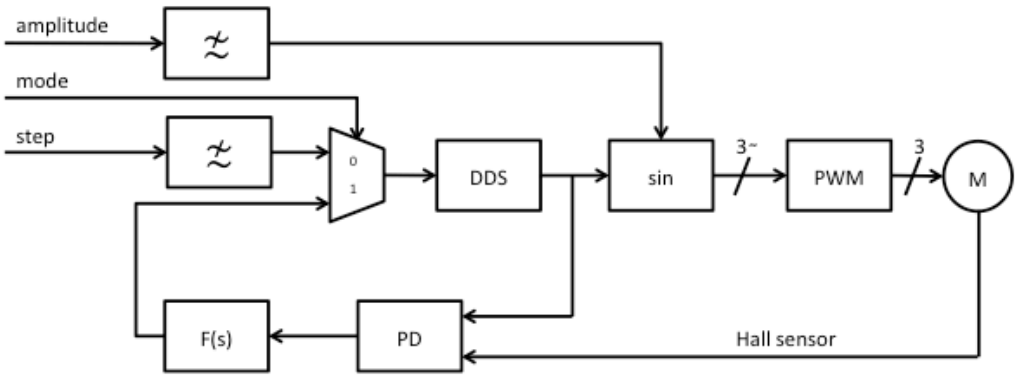


Figure 26: Boucle de réglage avec filtre intégrateur

Equation du filtre intégrateur PI :

$$F(s) = \frac{1 + s \cdot \tau_2}{s \cdot \tau_1}$$

9.2 MACHINE D'ETAT ETHERCAT

Le développement des fonctionnalités du périphérique EtherCAT n'est pas encore complet. La norme d'EtherCAT explique que dans le périphérique une machine d'état qui sert à gérer le fonctionnement de la communication sur le réseau est présente. Celle-ci est pilotée directement par le maître du réseau qui envoie des commandes de requête pour le changement d'état aux périphériques.
Une autre fonctionnalité à développer serait l'interface d'information du périphérique (ou SII). Celle-ci est représentée par une mémoire qui contient les informations générales du système. Cette fonctionnalité permettrait au maître du réseau d'identifier le type de périphérique utilisé.
D'autres fonctionnalités ne sont pas, à ce jour, implémentées. Il s'agit de :

- L'horloge distribuée
- Les interruptions

9.3 COUCHE DE LIAISON

Aucun bloc capable d'effectuer des services ARP n'est présent dans l'interface Ethernet. Ceci empêche la découverte de l'adresse MAC du périphérique par le maître du réseau. Un travail à réaliser pour éviter d'effectuer manuellement le routage de la table ARP dans l'ordinateur de commande consisterait à ajouter un bloc capable de répondre aux requêtes ARP. Étant donné la présence du module UDP, une autre possibilité serait d'ajouter un bloc capable de répondre à des requêtes de type Bonjour.

Note :

Bonjour est l'implémentation Apple du *Zero configuration networking (Zeroconf)*. Celui-ci est un groupe de technologies qui permet de créer un réseau IP sans avoir besoin de configurer des services comme le DHCP ou le DNS. Une implémentation *open source* de Bonjour est Avahi.

10. CONCLUSION

Les objectifs principaux du travail de diplôme sont atteints. Le circuit numérique créé est capable de piloter un moteur de type Brushless en fonction d'une commande d'amplitude et de renvoyer l'information concernant la vitesse moyenne de rotation du moteur. Ces informations sont transmises par Ethernet à un PC de commande sous le protocole EtherCAT. Le système est prévu pour l'utilisation avec différents types de moteurs Brushless. Ceci nous permet de pouvoir facilement l'adapter à la roue à moteur électrique intégré disponible à l'école.

Le mode de régulation développé n'est pas de type standard pour le contrôle des moteurs Brushless. La synchronisation de la phase avec les signaux provenant des capteurs à effet Hall nous permet d'éviter la mesure des courants de phase et par conséquence de simplifier le circuit électronique du système.

Pour le reste, quelques améliorations du système sont à effectuer pour obtenir un contrôle meilleur du moteur.

L'actuelle boucle de réglage fonctionne correctement mais présente un petit défaut. Au démarrage du système, si la roue ne tourne pas, le détecteur de phase n'est pas capable de calculer la différence de phase et par conséquent le pas du compteur de phase est nul.

L'interface de communication par réseau Ethernet n'est pas encore complète. Actuellement, elle supporte le protocole de communication EtherCAT/UDP, mais elle ne possède pas la totalité des fonctionnalités d'un périphérique EtherCAT. Dans le chapitre sur les améliorations encore à effectuée, il est décrit ce qui pourrait être ajouté à l'interface. (Cf. chapitre 9.2)

La structure du circuit de commande numérique permet l'utilisation séparée des blocs. Le module EtherCAT ou le module de commande du moteur pourraient être insérés dans différentes autres architectures sans trop de difficulté.

À l'heure actuelle, je peux affirmer qu'on dispose d'un premier prototype fonctionnel de contrôleur pour moteurs électriques de type Brushless manœuvrable par des commandes de type EtherCAT. Avec une adaptation du circuit électronique de puissance, ce prototype pourrait être employé sur la brouette avec moteur intégré à la roue disponible à l'école.

11. REMERCIEMENTS

Je tiens à remercier M. Samuel Chevailler, M. Dominique Roggo, M. Blaise Evéquoz pour l'aide apportée pour une meilleure compréhension du fonctionnement des moteurs Brushless, M. Joseph Moerschell pour l'aide relative à la régulation numérique, M. Silvan Zahno pour l'aide liée à la compréhension du fonctionnement d'EtherCAT et M. François Corthay pour ses nombreuses explications.

SIGNATURE

Sion le : 12 juillet 2012 *MICHELE KORELL :* _____

12. ANNEXES ET REFERENCES

12.1 ANNEXES

Annexe 1 : Structure du système numérique de commande
Annexe 2 : Structure du générateur de commande PWM
Annexe 3 : Structure des registres de configuration.
Annexe 4 : Entête librairie EtherCAT pour Qt.
Annexe 5 : Documents électroniques (CD-ROM)

12.2 REFERENCES

Référence 1 : Permanent-Magnet and Brushless DC Motors (1985, Oxford science publications, T. Kenjio – S.Nagamori).
Référence 2 : Commande d'un moteur Brushless DC pour brouette assistée électriquement (Diplôme 2011, FSI, Jérôme Vuignier).
Référence 3 : Technologie EtherCAT (http://www.ethercat.org/en/technology.html).
Référence 4 : Outil de développement Qt (http://qt.nokia.com/products/).
Référence 5 : Logiciel wireshark (http://www.wireshark.org).

BldcDriver / bldcSystem_EtherCAT / struct

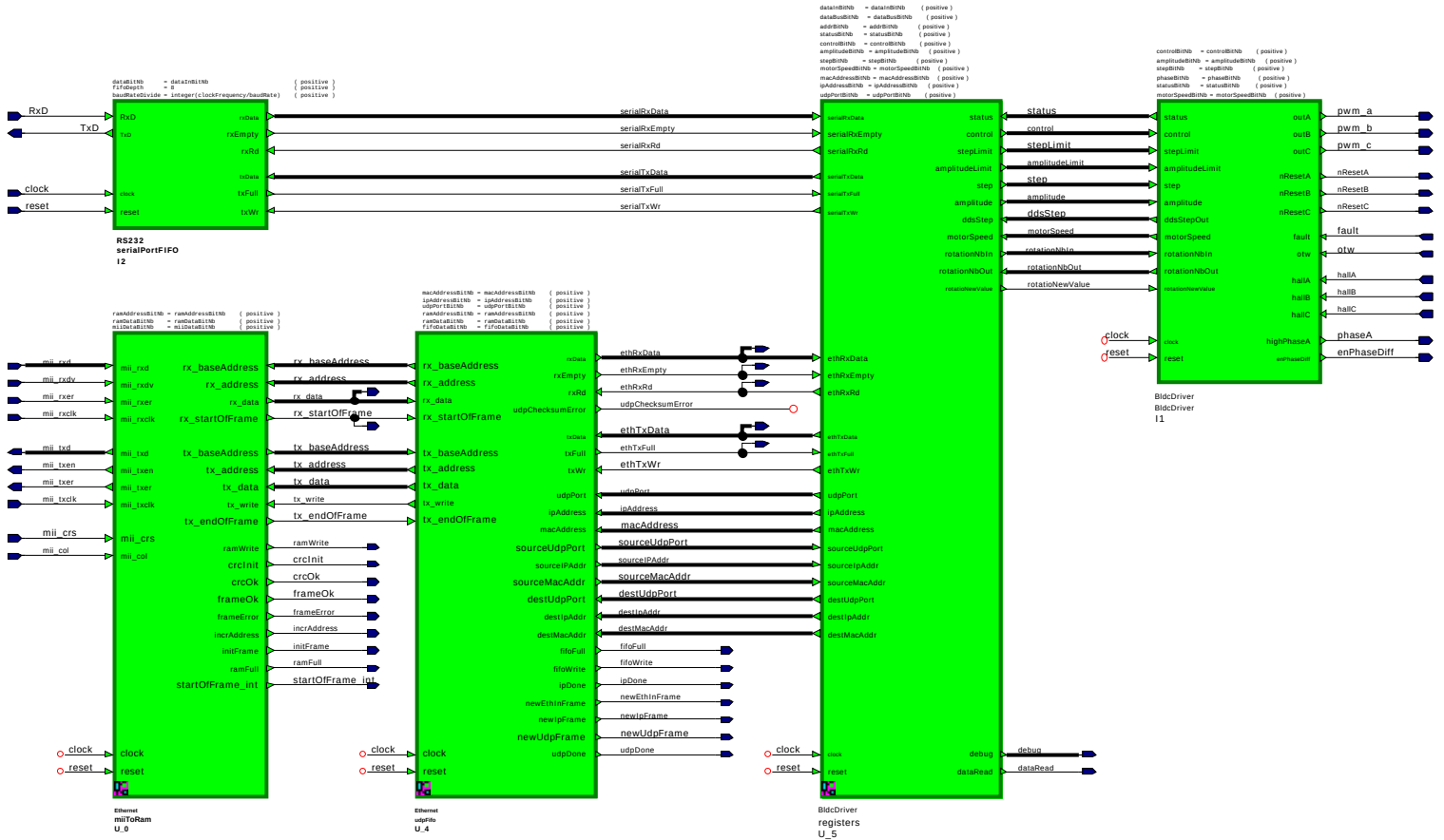
Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

Declarations
Ports:

```
RxD             : std_ulogic
clock           : std_ulogic
fault           : std_ulogic
hallA           : std_uLogic
hallB           : std_uLogic
hallC           : std_uLogic
mii_col         : std_ulogic
mii_crs         : std_ulogic
mii_rxclk       : std_ulogic
mii_rxd         : std_ulogic_vector(miiDataBitNb -1 DOWNTO 0)
mii_rxdv        : std_ulogic
mii_rxer        : std_ulogic
mii_txclk       : std_ulogic
otw             : std_ulogic
reset           : std_ulogic
TxD             : std_ulogic
crcInit         : std_ulogic
crcOk           : std_ulogic
dataRead        : std_ulogic
debug           : std_ulogic_vector(7 DOWNTO 0)
enPhaseDiff     : std_ulogic
ethRxData       : std_ulogic_vector(7 DOWNTO 0)
ethRxEmpty      : std_ulogic
ethRxRd         : std_ulogic
ethTxData       : std_ulogic_vector(7 DOWNTO 0)
ethTxFull       : std_logic
fifoFull        : std_ulogic
fifowrite       : std_ulogic
frameError      : std_ulogic
frameOk         : std_ulogic
incrAddress     : std_ulogic
initFrame       : std_ulogic
ipDone          : std_ulogic
mii_txd         : std_ulogic_vector(miiDataBitNb -1 DOWNTO 0)
mii_txen        : std_ulogic
mii_txer        : std_ulogic
nResetA         : std_ulogic
nResetB         : std_ulogic
nResetC         : std_ulogic
newEthInFrame   : std_ulogic
newIpFrame      : std_ulogic
newUdpFrame     : std_ulogic
phaseA          : std_ulogic
pwm_a           : std_logic
pwm_b           : std_logic
pwm_c           : std_logic
ramFull         : std_ulogic
ramWrite        : std_ulogic
rx_data         : std_ulogic_vector(ramDataBitNb-1 DOWNTO 0)
rx_startOfFrame : std_ulogic
startOfFrame_int : std_ulogic
udpDone         : std_ulogic
```

Diagram Signals:

```
SIGNAL amplitude        : unsigned(amplitudeBitNb-1 DOWNTO 0)
SIGNAL amplitudeLimit   : unsigned(stepBitNb-1 DOWNTO 0)
SIGNAL control          : std_ulogic_vector(controlBitNb-1 DOWNTO 0)
SIGNAL ddsStep          : unsigned(stepBitNb-1 DOWNTO 0)
SIGNAL destIpAddr       : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL destMacAddr      : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL destUdpPort      : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL ethTxWr          : std_logic
SIGNAL ipAddress        : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL macAddress       : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL motorSpeed       : unsigned(motorSpeedBitNb-1 DOWNTO 0)
SIGNAL rotationNewValue : std_logic
SIGNAL rotationNbIn     : std_ulogic_vector(rotationBitNb - 1 DOWNTO 0)
SIGNAL rotationNbOut    : std_ulogic_vector(rotationBitNb -1 DOWNTO 0)
SIGNAL rx_address       : unsigned(ramAddressBitNb -1 DOWNTO 0)
SIGNAL rx_baseAddress   : unsigned(ramAddressBitNb-1 DOWNTO 0)
SIGNAL serialRxData     : std_ulogic_vector(dataInBitNb-1 DOWNTO 0)
SIGNAL serialRxEmpty    : std_logic
SIGNAL serialRxRd       : std_logic
SIGNAL serialTxData     : std_ulogic_vector(dataInBitNb-1 DOWNTO 0)
SIGNAL serialTxFull     : std_ulogic
SIGNAL serialTxWr       : std_ulogic
SIGNAL sourceIPAddr     : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
SIGNAL sourceMacAddr    : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
SIGNAL sourceUdpPort    : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
SIGNAL status           : unsigned(statusBitNb-1 DOWNTO 0)
SIGNAL step             : unsigned(stepBitNb-1 DOWNTO 0)
SIGNAL stepLimit        : unsigned(stepBitNb-1 DOWNTO 0)
SIGNAL tx_address       : unsigned(ramAddressBitNb-1 DOWNTO 0)
SIGNAL tx_baseAddress   : unsigned(ramAddressBitNb-1 DOWNTO 0)
SIGNAL tx_data          : std_ulogic_vector(ramDataBitNb-1 DOWNTO 0)
SIGNAL tx_endOfFrame    : std_ulogic
SIGNAL tx_write         : std_ulogic
SIGNAL udpChecksumError : std_ulogic
SIGNAL udpPort          : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
```



<company name>	Project: bldc
Title: <enter diagram title here>	<enter comments here>
Path: BldcDriver/bldcSystem_EtherCAT/struct	
Edited: by korellm on 12 jul. 2012	

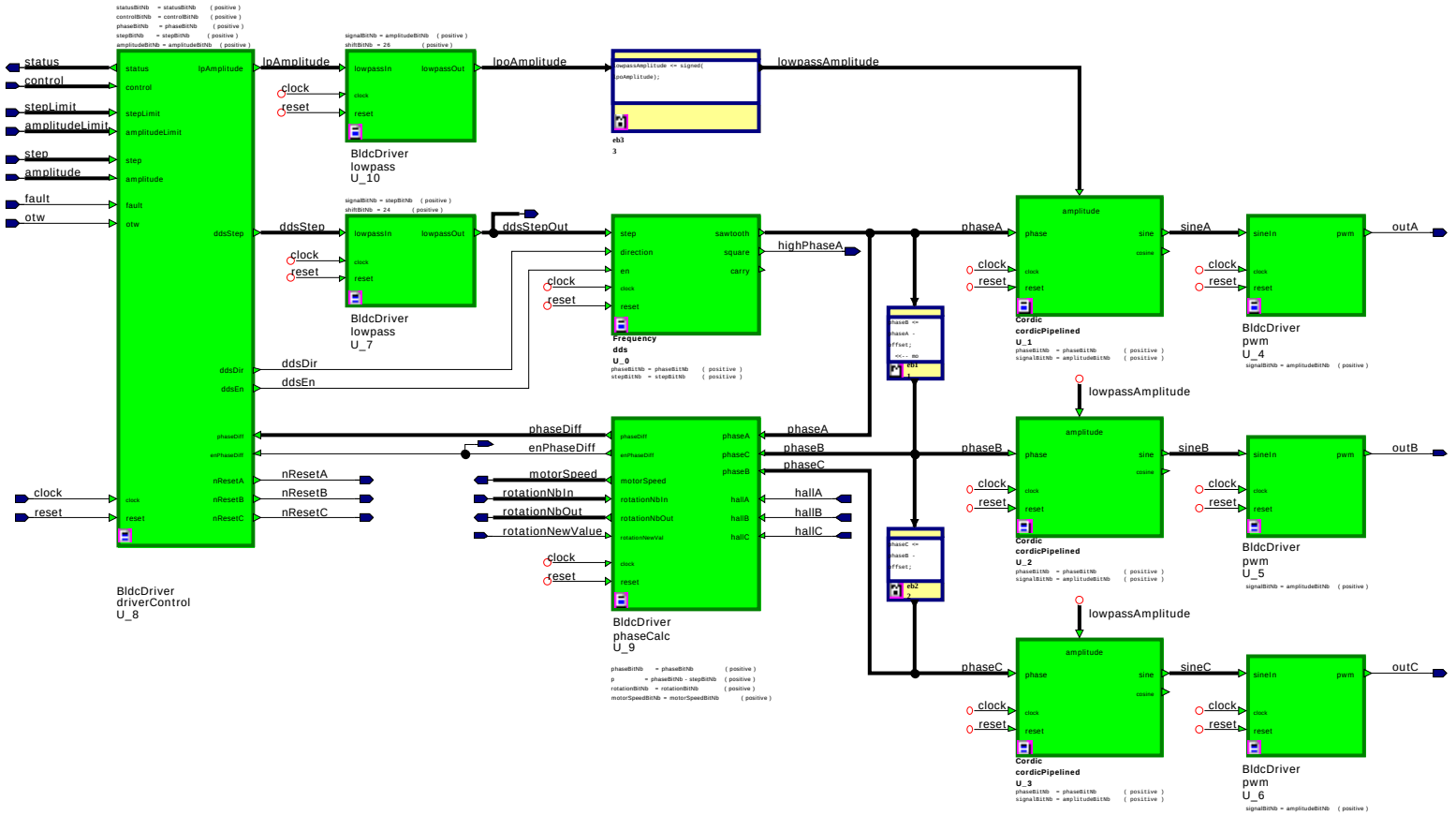
Package List
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

Declarations
Ports:

```
amplitude          : unsigned(amplitudeBitNb-1 DOWNTO 0)
amplitudeLimit     : unsigned(amplitudeBitNb-1 DOWNTO 0)
clock              : std_ulogic
control            : std_ulogic_vector(controlBitNb-1 DOWNTO 0)
fault              : std_ulogic
hallA              : std_uLogic
hallB              : std_uLogic
hallC              : std_uLogic
otw                : std_ulogic
reset              : std_ulogic
rotationNbIn       : std_ulogic_vector(rotationBitNb-1 DOWNTO 0)
rotationNewValue   : std_ulogic
step               : unsigned(stepBitNb-1 DOWNTO 0)
stepLimit          : unsigned(stepBitNb-1 DOWNTO 0)
ddsStepOut         : unsigned(stepBitNb-1 DOWNTO 0)
enPhaseDiff        : std_ulogic
highPhaseA         : std_ulogic
motorSpeed         : unsigned(motorSpeedBitNb-1 DOWNTO 0)
nResetA            : std_ulogic
nResetB            : std_ulogic
nResetC            : std_ulogic
outA               : std_logic
outB               : std_logic
outC               : std_logic
rotationNbOut      : std_ulogic_vector(rotationBitNb-1 DOWNTO 0)
status             : unsigned(statusBitNb-1 DOWNTO 0)
```

Diagram Signals:

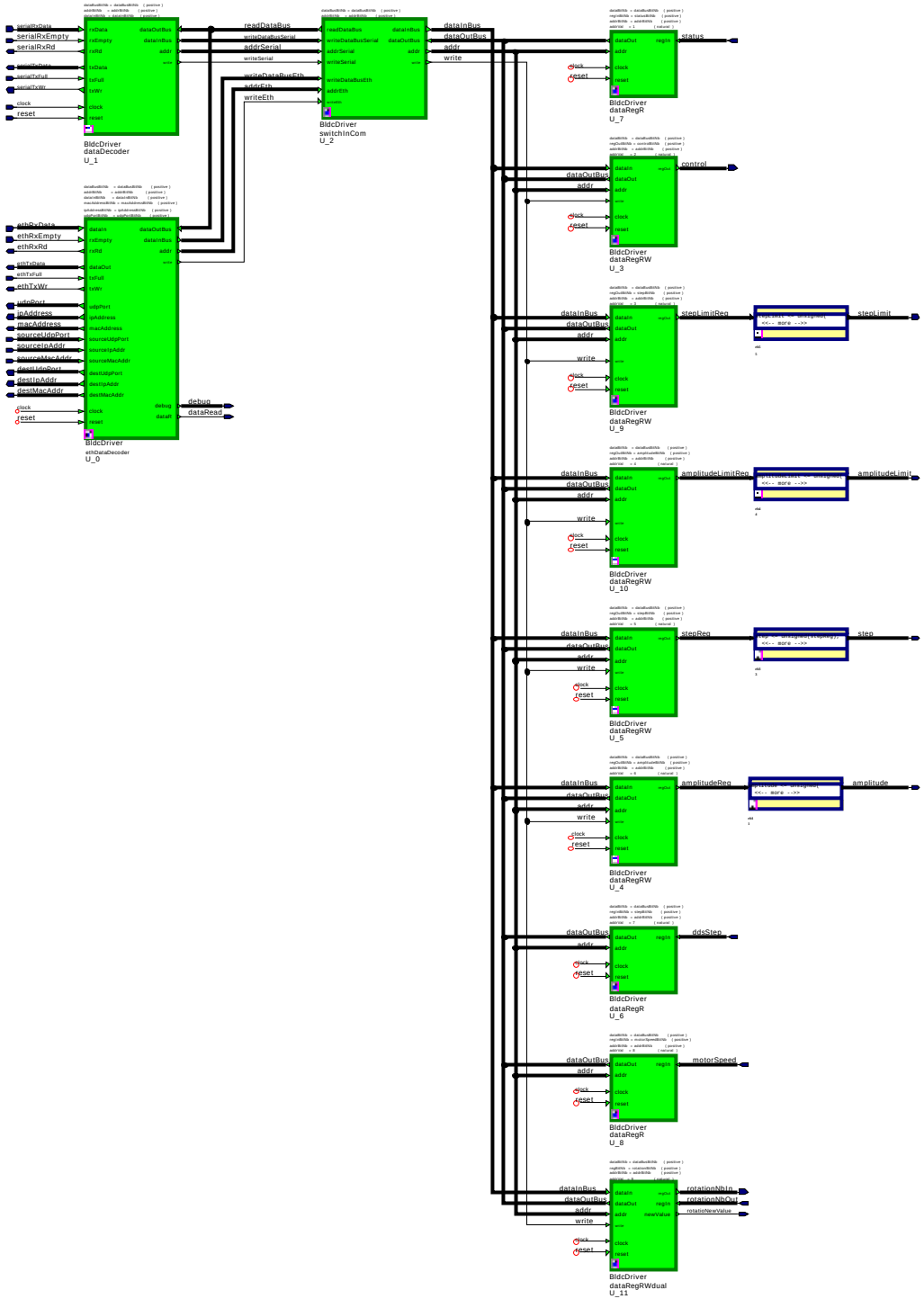
```
SIGNAL ddsDir          : std_ulogic
SIGNAL ddsEn           : std_ulogic
SIGNAL ddsStep         : unsigned(stepBitNb-1 DOWNTO 0)
SIGNAL lowpassAmplitude : signed(amplitudeBitNb-1 DOWNTO 0)
SIGNAL lpAmplitude     : unsigned(amplitudeBitNb-1 DOWNTO 0)
SIGNAL phaseA          : unsigned(phaseBitNb-1 DOWNTO 0)
SIGNAL phaseB          : unsigned(phaseBitNb-1 DOWNTO 0)
SIGNAL phaseC          : unsigned(phaseBitNb-1 DOWNTO 0)
SIGNAL phaseDiff       : signed(phaseBitNb-1 DOWNTO 0)
SIGNAL sineA           : signed(amplitudeBitNb-1 DOWNTO 0)
SIGNAL sineB           : signed(amplitudeBitNb-1 DOWNTO 0)
SIGNAL sineC           : signed(amplitudeBitNb-1 DOWNTO 0)
```



<company name>	Project: hds
Title: <enter diagram title here>	<enter comments here>
Path: BldcDriver/bldcDriver/struct	
Edited: by korellm on 12 jul. 2012	

```
------
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.NUMERIC_STD.all;

------
--
    clock               : std_ulogic
    ddsStep             : unsigned(stepBitNb-1 DOWNTO 0)
    ethRxData           : std_ulogic_vector(dataInBitNb-1 DOWNTO 0)
    ethRxEmpty          : std_ulogic
    ethTxFull           : std_logic
    motorSpeed          : unsigned(motorSpeedBitNb-1 DOWNTO 0)
    reset               : std_ulogic
    rotationNbOut       : std_ulogic_vector(rotationBitNb -1 DOWNTO 0)
    serialRxData        : std_ulogic_vector(dataInBitNb-1 DOWNTO 0)
    serialRxEmpty       : std_logic
    serialTxFull        : std_ulogic
    sourceIpAddr        : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
    sourceMacAddr       : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
    sourceUdpPort       : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
    status              : unsigned(statusBitNb-1 DOWNTO 0)
    amplitude           : unsigned(amplitudeBitNb-1 DOWNTO 0)
    amplitudeLimit      : unsigned(amplitudeBitNb-1 DOWNTO 0)
    control             : std_ulogic_vector(controlBitNb-1 DOWNTO 0)
    dataRead            : std_ulogic
    debug               : std_ulogic_vector(7 DOWNTO 0)
    destIpAddr          : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
    destMacAddr         : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
    destUdpPort         : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
    ethRxRd             : std_ulogic
    ethTxData           : std_ulogic_vector(dataInBitNb-1 DOWNTO 0)
    ethTxWr             : std_logic
    ipAddress           : std_ulogic_vector(ipAddressBitNb-1 DOWNTO 0)
    macAddress          : std_ulogic_vector(macAddressBitNb-1 DOWNTO 0)
    rotationNewValue    : std_logic
    rotationNbIn        : std_ulogic_vector(rotationBitNb - 1 DOWNTO 0)
    serialRxRd          : std_logic
    serialTxData        : std_ulogic_vector(dataInBitNb-1 DOWNTO 0)
    serialTxWr          : std_ulogic
    step                : unsigned(stepBitNb-1 DOWNTO 0)
    stepLimit           : unsigned(stepBitNb-1 DOWNTO 0)
    udpPort             : std_ulogic_vector(udpPortBitNb-1 DOWNTO 0)
------
    SIGNAL addr                 : natural
    SIGNAL addrEth              : std_ulogic_vector(addrBitNb-1 DOWNTO 0)
    SIGNAL addrSerial           : std_ulogic_vector(addrBitNb-1 DOWNTO 0)
    SIGNAL amplitudeLimitReg    : std_ulogic_vector(amplitudeBitNb-1 DOWNTO 0)
    SIGNAL amplitudeReg         : std_ulogic_vector(amplitudeBitNb-1 DOWNTO 0)
    SIGNAL dataInBus            : std_ulogic_vector(dataBusBitNb-1 DOWNTO 0)
    SIGNAL dataOutBus           : std_logic_vector(dataBusBitNb-1 DOWNTO 0)
    SIGNAL readDataBus          : std_ulogic_vector(dataBusBitNb-1 DOWNTO 0)
    SIGNAL stepLimitReg         : std_ulogic_vector(stepBitNb-1 DOWNTO 0)
    SIGNAL stepReg              : std_ulogic_vector(stepBitNb-1 DOWNTO 0)
    SIGNAL write                : std_ulogic
    SIGNAL writeDataBusEth      : std_ulogic_vector(dataBusBitNb-1 DOWNTO 0)
    SIGNAL writeDataBusSerial   : std_ulogic_vector(dataBusBitNb-1 DOWNTO 0)
    SIGNAL writeEth             : std_ulogic
    SIGNAL writeSerial          : std_ulogic
```



```
//
//  EtherCATudp.h
//  BLDCdriver
//
//  Created by Michele Korell on 12.07.2012
//  Copyright (c) 2012 Hes-so VS. All rights reserved.
//

#ifndef ETHERCATUDP_H
#define ETHERCATUDP_H

#include <QtNetwork/QUdpSocket>
#include <QList>
#include <QTimer>
#include "ethercatdefines.h"

//
//This class is a library for Qt who allow to send EtherCAT commands on udp connection
//Functions:
//  - Send EtherCAT commands
//  - Receive EtherCAT commands
//  - Setup loop send command
//
//warnig: not all EtherCAT commands are implemented
//

class EtherCATudp : public QObject
{
    Q_OBJECT

    //UDP communication socket
    QUdpSocket      socket;
```



```
public:
    EtherCATudp();
    //Setup udp connection
    void setupConnection();
    //Send datagrams on udp connection
    void sendDatagrams(int size, EcatDatagram * datagrams);
    //Return received datagrams
    EcatDatagram* getDatagrams();
    //Return the numbrer of received datagrams
    int getDatagramsNb();

    //Create EtherCAT command
    //Cofigured Address Read
    EcatDatagram commandFPRD(short address, short offset, short length, short irq, unsigned char * data);
    //Configured Address Write
    EcatDatagram commandFPWR(short address, short offset, short length, short irq, unsigned char * data);

    //Start loop command request for registers monitoring
    void startLoopCommand(int timeLoop);
    //Stop loop command request
    void stopLoopCommand();
    //add command to loop request list
    void addLoopCommand(EcatDatagram datagram);

signals:
    //Signal emitted when data are received from udp connection
    void newDatagrams();
private slots:
    //Slot to get the incoming datagrams
    int receiveDatagrams();
    //Slot to send commands when loop time is fired
    void sendLoopCommand();
private:
```

```
    //List of loop commands to send
    QList<EcatDatagram> loopCommands;
    //Incoming datagrams pointer
    EcatDatagram * dataIn;
    //Header of frame to send
    EcatHeader header;
    //Number of received datagrams
    int nbDatagrams;
    //Loop timer
    QTimer *timer;
};

#endif // ETHERCATUDP_H
```