2021-2022

Projet VAE

Rapport ER3

Introduction

Dans le cadre de notre troisième semestre, nous devons réaliser - en binôme pour notre part - un projet en Étude et Réalisations. Il s'agit de réaliser une étude de la commande et du fonctionnement d'un vélo à assistance électrique. Cette étude comprendra l'analyse du fonctionnement d'un vélo à assistance électrique et de ses constituants que nous détaillerons par la suite. Ainsi à travers tout ceci nous aurons l'occasion de mettre en œuvre les différentes connaissances acquises tout au long de ces trois semestres de DUT GEII. Mais il nous permet aussi d'apprendre de nouvelles notions et d'acquérir de nouvelles compétences par la pratique. De ce fait, ce projet tutoré réalisé en autonomie nous apprend à gérer notre temps, chercher les connaissances par nous-même mais aussi concrétiser les connaissances théoriques abordées.

Pour ce faire nous avons des créneaux de trois heures et trente minutes une fois par semaine, pour pouvoir y travailler sous la tutelle de deux professeurs.

Ce rapport commence premièrement par la présentation du projet, deuxièmement par l'étude matérielle et logicielle détaillée, suivie par le dossier de fabrication logiciel et matériel et enfin d'une conclusion avec une proposition d'amélioration.

Table des matières :

1.Dossier d'étude : vélo à assistance électrique

1.1 Eléments constitutifs d'un VAE……4

1.1.2 Types de moteurs possibles……6

1.1.3 Types de technologies de batteries possibles actuellement……12

1.2. Cahier des charges (on présente les fonctionnalités définitives attendues dans le projet même si elles ont été présentées au fur et à mesure des séances)……12

1.2.1. Liste et description de spécifications/fonctionnalités attendues (jusqu'à l'application Android et la communication CAN incluse)……15

1.3. Analyse du cahier des charges (justifier les choix technologiques faits dans le projet)

1.3.1. Schéma du découpage fonctionnel général du projet (représentation graphique): Schéma bloc fonctionnel général de l'architecture avec présentation des liaisons de communication avec l'extérieur (les liaisons avec les IHMs et les liaisons de bus de terrain)……16

1.3.2. Réaliser une fiche signalétique de chaque bloc fonctionnel (blocs matériels et blocs logiciels) qui apparaissent dans le schéma bloc fonctionnel précédent (nom du bloc, rôle, entrées sorties)……17

1.3.3. Représentation du Schéma bloc fonctionnel de la gestion uC Mbed (tâche de fond, tickers, it externes et variables échangées entre ces blocs)……18

2. Etude matérielle et logicielle détaillée

2.1. Étude du bloc autopilotage/filtrage hall/générateur PWM (PLD) (détailler les sous blocs avec les codes vhdl sources commentés, simulations commentées, procédures et résultats de validation) et conclusion partielle pour chaque bloc…… 19

2.2. Étude de chacun des blocs situés dans le microcontrôleur mBeD (détail des blocs logiciels internes, codes C associés commentés, résultats des tests, copies d'écran console et conclusion partielle sur chaque bloc)…… 25

3. Dossier de fabrication logiciel et matériel

3.1. Code C logiciel complet et commenté si possible en anglais ……32

3.2. Code VHDL et assemblage graphique complet et commenté ……39

3.3. Notice d'utilisation du menu console, plan de câblage du banc de test et procédure de calibration/réglages……. …….. 40

4. Proposition d'amélioration et conclusion

4.1. Conclusion et tableau de comparaison entre le travail réalisé et les spécifications globales attendues dans le cahier des charges : détail du travail restant à réaliser…….. 42

4.2. Propositions d'amélioration du projet…….44

4.3. Conclusion générale sur le plan technique et personnelle……..44

1. Dossier d'étude : Vélo à assistance électrique

1.1 Eléments constitutifs d'un VAE

Nous allons ici voir les différents éléments qui constituent un vélo à assistance électrique.
Nous pouvons remarquer sur la figure ci-dessous qu'un VAE est constitué de différents capteurs, l'indicateur, le contrôleur et le détecteur de pédalage, il y aussi deux autres éléments très importants qui sont le moteur et la batterie, sur lesquels nous développerons un peu plus par la suite.

Figure 1 - ÉLéments constitutifs d'un VAE

Comme vu précédemment, un des principaux éléments est le détecteur de pédalage. Le détecteur de pédalage est plus précisément un capteur de pédalage PAS "Pedal Assist Sensor", celui-ci permet de connaître la vitesse de pédalage.

La cadence de rotation du pédalier est mesurée par une roue mobile dotée de petits aimants, fixée sur l'axe tournant du pédalier. De plus, il y a un capteur de pédalage doté de 2 capteurs

magnétiques (en général à effet hall), monté fixe sur le pédalier permettant de mesurer la cadence de passage des aimants.

Figure 2 - Capteur de pédalage PAS

Le cerveau du VAE est le contrôleur, il permet la conversion d'énergie entre la batterie et le moteur d'assistance du vélo. Si c'est un moteur à courant continu il sera constitué d'un hacheur et si c'est un moteur brushless triphasé ce sera un onduleur triphasé de puissance.

De plus, c'est au niveau du contrôleur qu'il y a l'ensemble des connectiques, que ce soit pour la batterie, le moteur ou bien les différents capteurs.

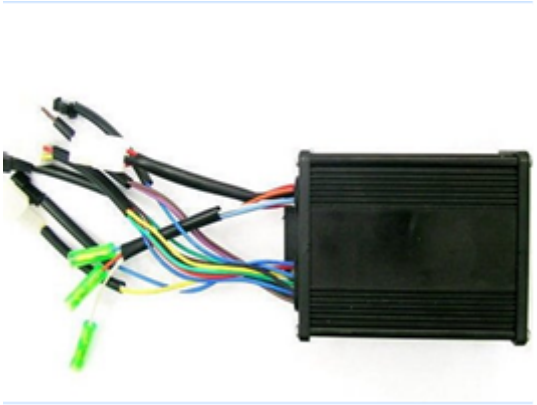


Figure 3 - Contrôleur de puissance

Il est indispensable sur un VAE comme pour tout véhicule de pouvoir freiner lorsque nous voulons. Pour cela, nous avons une poignée de frein dotée d'un contacteur électrique qui informe le calculateur d'un freinage, ce qui amène à la coupure d'alimentation en énergie du moteur.

Cette phase de freinage peut permettre la récupération d'énergie et donc recharger les batteries.

Figure 4 - Poignée de frein

Le moteur roue permet la conversion électromécanique de la chaîne de puissance. Ce moteur à courant continu ou courant alternatif remplace la roue d'origine.

Figure 5 - Moteur roue

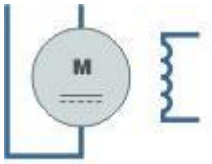
1.1.2 Types de moteurs possibles

Les moteurs électriques sont des machines électrotechniques qui convertissent l'énergie électrique en énergie mécanique. Il est donc important de bien choisir le moteur.

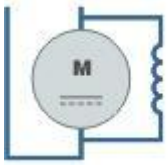
Un moteur à courant continu est constitué de deux parties électriques : une partie fixe qui génère un champ magnétique (le stator ou l'inducteur) et une partie tournante (le rotor ou induit).
Le stator peut être fait d'aimants permanents ou d'électroaimants qu'il faut alimenter.
Le rotor est constitué du bobinage induit qu'il faut alimenter pour la transformer en électroaimant, celui interagit avec le stator tourne.
Le collecteur est un ensemble de plages métalliques qui fait contact avec les balais et permet l'alimentation du rotor.

Différents types de moteur à courant continu :
Suivant l'application, les bobinages de l'inducteur et de l'induit peuvent être connectés de manière différente.
On retrouve en général :

Des moteurs à excitation indépendante.

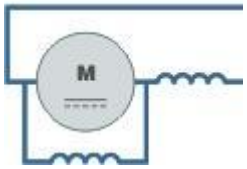


Des moteurs à excitation parallèle.



Des moteurs à excitation série.

Des moteurs à excitation composée.



Lorsqu'on alimente le moteur, il se crée une interaction magnétique qui met le moteur en mouvement. Lorsqu'on inverse le sens de la tension qui alimente le moteur, il tourne en sens inverse.

En fonction du domaine d'application, il existe différents types de moteurs électriques : moteur électrique synchrone et asynchrone, et moteur électrique monophasé ou triphasé.
Les moteurs électriques monophasés sont très présents dans les habitations. Ils fonctionnent grâce à une tension simple, c'est-à-dire entre la phase et le neutre.
Les moteurs électriques triphasés, contrairement au moteur électrique monophasé, fonctionnent à l'aide de 3 phases et permettent d'acheminer une puissance électrique beaucoup plus importante. Ces moteurs triphasés sont utilisés pour les infrastructures et équipements nécessitant une forte puissance électrique.

Le moteur synchrone se compose aussi d'un stator et d'un rotor séparés par un entrefer. La seule différence se situe au niveau de la conception du rotor. La figure ci-dessous montre un rotor à pôles saillants constitués d'aimants permanents ou d'électro-aimants alimentés en courant continu.

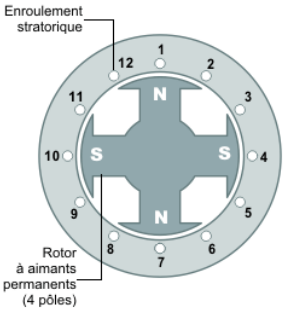


Figure 6 - Conception moteur synchrone

Après le démarrage, le moteur tourne en synchronisation avec le champ tournant.
A vide, les axes des pôles du champ tournant et du rotor sont confondus.
En charge, les axes sont légèrement décalés. La vitesse du moteur synchrone est constante quelle que soit la charge.

Le moteur asynchrone est beaucoup utilisé car il nécessite peu d'entretien, est simple à construire et robuste.

Le principe est de créer un courant électrique induit dans un conducteur placé dans un champ magnétique tournant. Ce conducteur est un des barreaux constituant le rotor du moteur. L'induction du courant se fait seulement si le conducteur est en court-circuit. De plus, il y a la création d'une force électromotrice sur le conducteur, dont le sens est donné par la règle des trois doigts de la main droite.

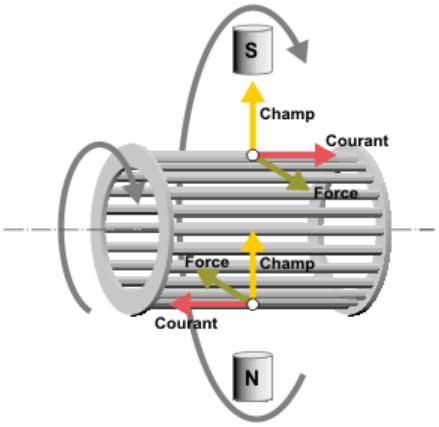


Figure 7 - Principe de fonctionnement d'un moteur asynchrone

Le stator se compose principalement de la carcasse, des paliers, des flasques de palier, du ventilateur refroidissant le moteur et du capot protégeant le ventilateur. L'intérieur du stator comprend essentiellement un noyau en fer permettant la canalisation du flux magnétique et des enroulements des trois phases logés dans les encoches du noyau.

Figure 8 - Stator

Lorsque les enroulements du stator sont parcourus par un courant triphasé, ceux-ci produisent un champ magnétique tournant à la vitesse de synchronisme qui dépend de la fréquence du réseau d'alimentation et du nombre de paires de pôles. Vu que la fréquence est fixe, la vitesse du moteur peut varier en fonction du nombre de paires de pôles.

Dans un moteur triphasé les enroulements sont au nombre minimum de trois décalés l'un de l'autre de 120° comme le montre le schéma ci-dessous.

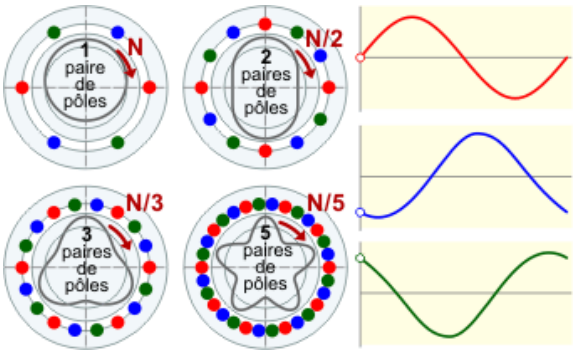


Figure 9 - Variation de la vitesse en fonction du nombre de paires de pôles

Le rotor d'un moteur synchrone se compose d'une succession de pôles Nord et Sud intercalés sous forme d'aimants permanents ou de bobines.

Figure 10 - Rotor bobiné

Figure 11 - Rotor à aimants permanents

Le rotor d'un moteur asynchrone lui est composé d'un empilage de disques minces isolés entre eux et d'une cage en aluminium.

Figure 12 - Rotor d'un moteur asynchrone

De nos jours, le moteur Brushless est le moteur le plus utilisé car il est beaucoup plus robuste qu'un moteur à courant continu. De ce fait, notre étude se limitera uniquement à ce type de moteurs associés à des onduleurs triphasés.

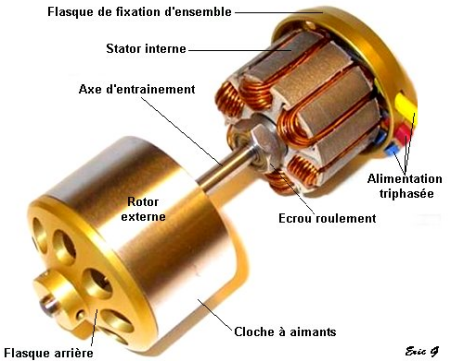


Figure 13 - Moteur Brushless

C'est un moteur triphasé alternatif doté au rotor d'un inducteur à aimants permanents, aussi appelé moteur à courant continu sans balais. Contrairement au moteur à courant continu, il n'est pas doté d'un collecteur mécanique mais d'un collecteur électronique réalisé avec des transistors.

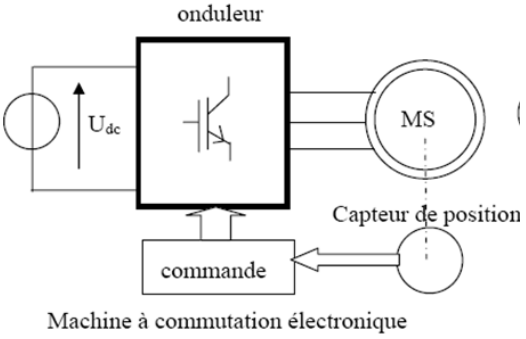


Figure 14 - Structure de convertisseurs d'énergie pour un moteur Brushless

L'autopilotage dans un moteur Brushless (MS) est tout simplement, la rotation du rotor mesurée par des capteurs de position qui permet la détermination de la phase qui sera alimentée à un instant donné.

1.1.3 Types de technologies de batteries possibles actuellement

La batterie est la source d'énergie du VAE , le choix de la batterie est donc très important. La batterie peut être soit fixée sur un support amovible, soit sur le cadre du vélo.
Pour pouvoir bien choisir sa batterie et comparer les différents types de batteries existantes actuellement, nous avons ci-dessous un tableau avec les avantages et les inconvénients de chacune d'entre elles.

	Batterie en plomb	Batterie nickel-cadmium	Batterie lithium-ion	Batterie nickel-hydrure métallique
Avantages	Coût peu élevé Bon marché Pas d'effet mémoire, Bonne durée de vie	Plus légères que le plomb Bon marché Recharge facile et grande tolérance face aux surcharges	Faible résistance interne Légère Bonne autonomie Pas d'effet mémoire Grande variété de formes	phénomène d'auto-décharge Souffrent moins de l'effet mémoire que les Ni-Ca Meilleure autonomie
Inconvénients	Poids élevé Faible autonomie Sensible au froid Difficilement transportable en	Effet mémoire (décharger complètement la batterie avant de la recharger)	Coût élevé Usure même en cas de non utilisation	Durée de vie faible Coût élevé

	raison d'acide liquide	Polluantes Autonomie moyenne		Autodécharge importante

Figure 15 - Tableau des avantages et inconvénients des différentes batteries

1.2. Cahier des charges (on présente les fonctionnalités définitives attendues dans le projet même si elles ont été présenté au fur et à mesure des séances)

Le tableau ci-dessous rassemble tous les objectifs que nous avions à faire lors de nos 14 séances d'Études et réalisations et les solutions pour répondre aux demandes.

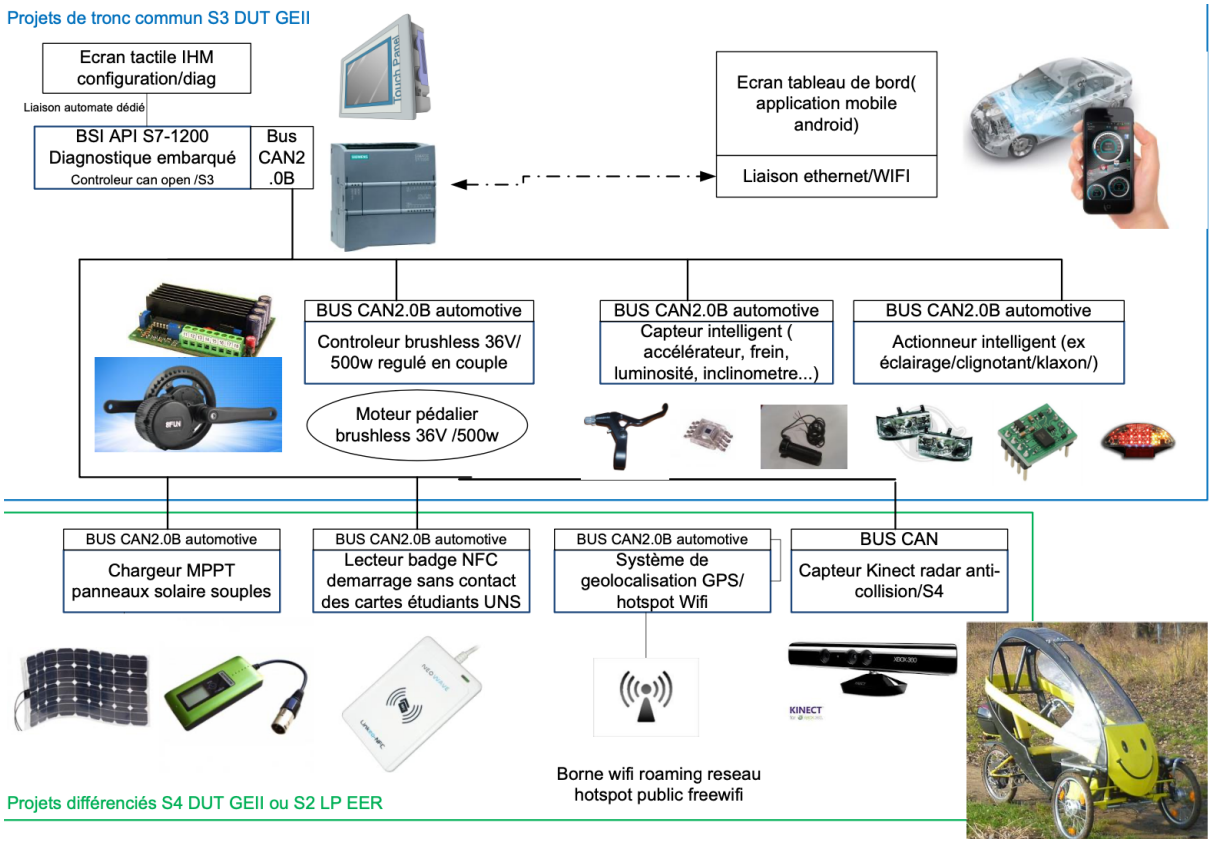
Séance	Objectif	Solution
1	analyser le sujet. faire tourner la roue en marche avant et marche arrière.	-Mettre en place un code vhdl sous quartus
2	Mise en place d'un filtre numérique des signaux	-Réaliser l'assemblage des blocs sous quartus. -Commencer à importer le code en langage c sous mbed compiler.
3	Mise en place du code en langage c sur mbed compiler.	-Réalisation de plusieurs tests pour faire tourner la roue
4	Terminer l'assemblage de 3 blocs logiques (PLD)	-Mise en place d'une IHM sur TeraTerm -Faire tourner la roue avec une ref pwm interne au PLD. -Mise en place du mbed et commande de la PWM à partir de la programmation console mbed. -Mise en place du mode auto et du mode manuel.

5	Commande en vitesse variable de la puissance de la poignée de gaz.	-Relever GAZ min et GAZ max. -Lecture du registre interne du PLD. -Gestion du frein dans la commande en vitesse variable.
6	-Affichage du registre interne PLD. -Conditionnement de la poignée de gaz avec sauvegarde dans un fichier texte persistant. -Commande de la poignée avec la gestion du frein.	-mise en place d'un switch case pour l'affiche du registre interne du PLD. -Mise en place d'un capteur de vitesse (très cher !) -Déterminer la vitesse du véhicule par la mesure de la rotation de la roue.
7	-Commande progressive avec poignée de gaz. -Gestion bride en vitesse	-Gestion de la commande progressive avec un PWM auto brut et un PWM auto final. -Modifier le programme en langage c afin d'instaurer des conditions pour réaliser la bride par rapport à la vitesse obtenue.
8	-Mesure du nombre de kilomètres parcourus.	-Compteur de distance total. -Compteur de distance partielle. -Nombre de tophall comptabilisés.
9	-Conditionnement 4 capteurs analogiques.	-Conditionner : iBat, vbat, température en degrés, % de la poignée de gaz.
10	-Mise en place de la jauge d'énergie restante (batterie).	-Prévoir option de charge initiale. -Prévoir option d'affichage de la capacité de la batterie.
11	Adapter l'affichage pour un utilisateur grand public.	-Gestion d'un affichage sur un écran LCD. -Gestion de l'affichage sur la tablette du client. (on ne fourni pas le hardware du client d'affichage)

		-Utiliser un client d'affichage natif de la plate-forme d'affichage (utiliser un navigateur web).
12	Gestion d'un capteur actionneur communicant par bus de terrain(Bus CAN).	-Neutralisation de l'utilisation des connexions cablées, donc multiplexage des infos sur un seul câble. -Décentraliser la commande de gaz.
13-14	Finitions	

1.2.1. Liste et description de spécifications/fonctionnalités attendues (jusqu'à l'application Android et la communication CAN incluse)

architecture du VAE:



La figure ci-dessus nous montre l'ensemble des fonctionnalités et du matériel que nous allons utiliser tout au long de notre projet.Nous nous arrêterons cependant à l'application android et la communication CAN pour la fin de ce semestre.

1.3. Analyse du cahier des charges (justifier les choix technologiques faits dans le projet)

1.3.1. Schéma du découpage fonctionnel général du projet (représentation graphique): Schéma bloc fonctionnel général de l'architecture avec présentation des liaisons de communication avec l'extérieur (les liaisons avec les IHMs et les liaisons de bus de terrain)

Ce schéma de découpage fonctionnel général de notre projet ci-dessous, nous permet de bien voir et analyser les différentes liaisons de communications entre les différents acteurs extérieurs.

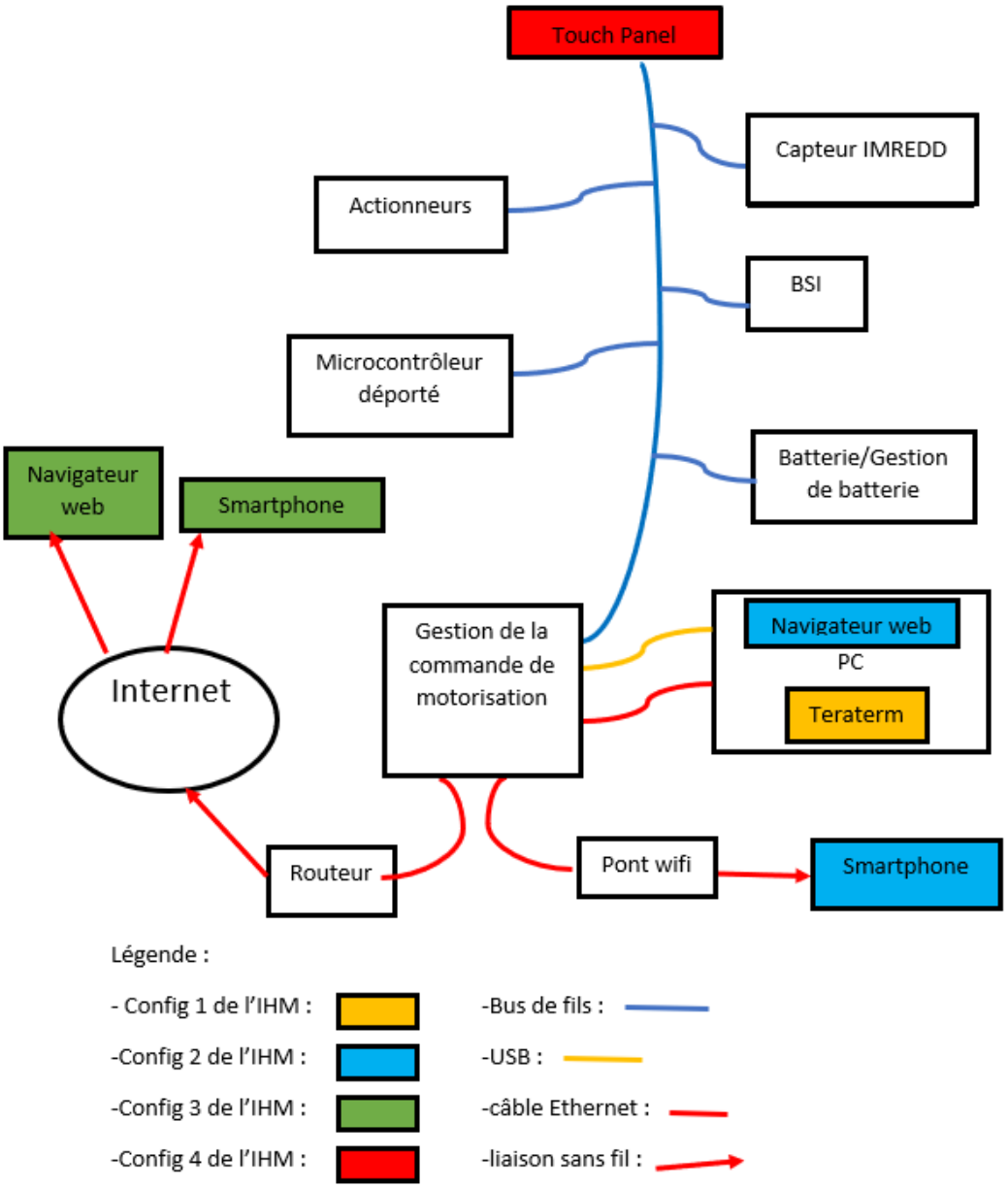


Figure 16 - Schéma à découpage fonctionnel

1.3.2. Réaliser une fiche signalétique de chaque bloc fonctionnel (blocs matériels et blocs logiciels) qui apparaissent dans le schéma bloc fonctionnel précédent (nom du bloc, rôle, entrées sorties).

La fiche signalétique ci-dessous nous permet de bien analyser le rôle, les entrées et sorties de chacun des blocs utilisés.

nom du bloc	***Actionneurs***	***Batterie***	***Micro contrôleur déporté***	***Gestion commande de motorisation***	***Routeur***	***Pont Wifi***
rôle	***l'actionneur est un dispositif matériel pour transformer une information digitale en un phénomène physique***	***Gestion de la batterie avec une visualisation sur le panel***	***Un microcontrôleur peut effectuer la plupart des instruction machine***	***détecter toute anomalie de fonctionnement des différents systèmes***	***Donne l'accès à internet***	***Donne l'accès à l'application sur le smartphone via une connection wifi***
entrées/ sorties	***Bus fils***	***Bus fils***	***Bus fils***	***Bus fils usb cable ethernet***	***cable ethernet***	***cable ethernet***

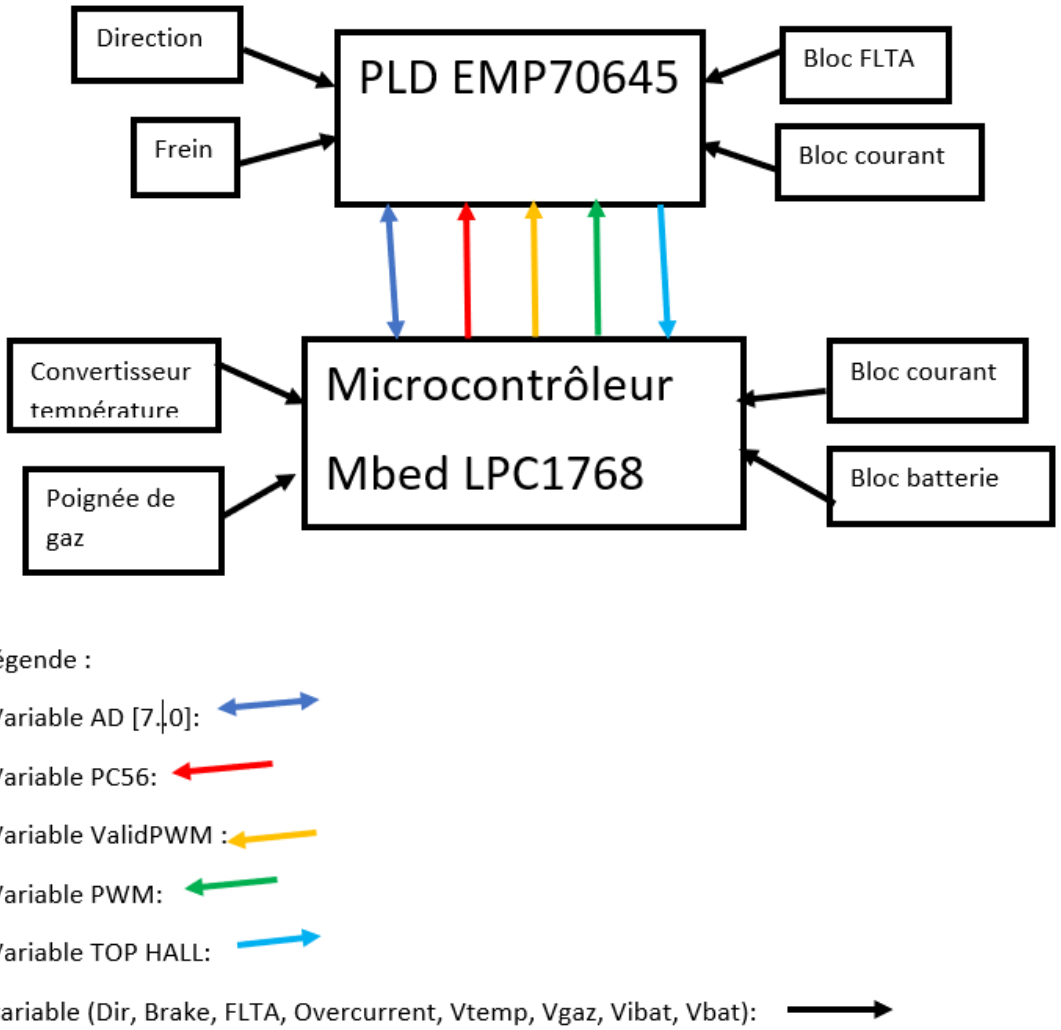
Figure 17 - Fiche signalétique des blocs matériels

nom du bloc	***Teraterm***	***Navigateur web***	***touch panel***
rôle	***afficher le menu déroulant. voir les résultats des compilations.***	***afficher toutes les données utilisateur.***	***configurer et de surveiller un ou plusieurs systèmes.***
entrées/sorties	***X***	***usb***	***Bus fils***

Figure18 - Fiche signalétique des blocs logiciels

1.3.3. Représentation du Schéma bloc fonctionnel de la gestion uC Mbed (tâche de fond, tickers, it externes et variables échangées entre ces blocs).

La schéma du bloc fonctionnel de la gestion du microcontrôleur Mbed ci-dessous nous montre bien les liaisons entre les blocs et le PLD ou le microcontrôleur.



2. Etude matérielle et logicielle détaillée

2.1. Étude du bloc autopilotage/filtrage hall/générateur PWM (PLD) (détailler les sous blocs avec les codes vhdl sources commentés, simulations commentées, procédures et résultats de validation) et conclusion partielle pour chaque bloc.

Nous allons maintenant voir séparément le but, le code VHDL, les simulations associées et ce qui constitue les différents blocs ci-dessous.

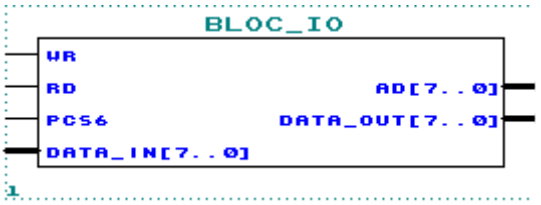


Figure 19 - Bloc IO

Ce bloc IO est fourni et permet l'échange bidirectionnel de mots de 8 bits entre le PLD Altera et le microcontrôleur MBED.

PWM_REFERENCE.vhd* | COLLECTEUR_NUM2.vhd* | matrix_scooter_pwm_mbed_etudiant_Mbed.bdf | Compilation Report - pld_mbed_etudiant | bloc_io_mbed.vhd*

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity bloc_io_mbed is
5     port (AD:inout std_logic_vector (7 downto 0); -- écrire la donnée sur le port 8 bits
6        RD_WRn,PCS6: in std_logic; DATA_OUT: out std_logic_vector (7 downto 0);
7        DATA_IN:in std_logic_vector (7 downto 0));
8  end bloc_io_mbed;
9
10 architecture bloc_ioa of bloc_io_mbed is
11 signal WR_INT,RD_INT: std_logic;
12 begin
13    RD_INT<=PCS6 or not(RD_WRn); --validation de l'accés
14    WR_INT<=PCS6 or RD_WRn;
15    AD<=DATA_IN when (RD_INT='0') else "ZZZZZZZZ"; --passer en mode écriture
16
17 P1:process (WR_INT)
18 begin
19    if (WR_INT'event and WR_INT='1' ) then --repasser en mode lecture par défaut
20       DATA_OUT<=AD;
21    end if;
22 end process P1;
23 end bloc_ioa;
```

Figure 20 - Code VHDL du bloc IO

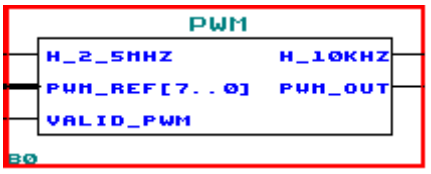


Figure 20 - Bloc de la PWM

Ce bloc PWM reçoit la consigne de PWM sur 8 bits en provenance du microcontrôleur et fournit un signal de PWM de (de fréquence de découpage de 10Khz) sortie unique qui sera plus tard distribuée par le bloc collecteur numérique.

Entrées :

- H2.5Mhz : horloge rapide 2.5Mhz externe.

- Valid_PWM : signal de validation global de la PWM (actif à 1) permettant de prévoir un arrêt d'urgence ou une limitation en courant par détection de surintensité. Si Valid PWM=0, la sortie PWM OUT=0
- PWM_REF[7..0] : consigne PWM d' entrée fournie par le microcontrôleur MBED.

Sorties :

- H_10Khz : sortie horloge 10Khz pour usage interne dans le PLD.
- PWM_OUT : sortie PWM modulée en fonction de PWM_REF[7..0]

PWM_REFERENCE.vhd* | COLLECTEUR_NUM2.vhd* | matrix_scooter_pwm_mbed_etudiant_Mbed.bdf | Compilation Report - pld_mbed_etudiant | bloc_io_mbed.vhd* | PWM.vhd*

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity PWM is
port(
   H_2_5MHz : in std_logic; -- entrée Horloge de 2.5MHz
   valid_pwm : in std_logic; -- entrée Valid PWM
   PWM_REF : in integer range 0 to 255; -- entrée de la référence PWM qui varie entre 0 et 255
   H10KHz : out std_logic; --sortie Horloge de 10KHz
   PWM_OUT : out std_logic); --sortue du PWM
end  PWM;

architecture  PWM_a of  PWM is

signal cpt : integer range 0 to 255; --signal du compteur quui varie de 0 à 255
begin
   P1 : process (H_2_5MHz)
   begin
      if(H_2_5MHz'event and H_2_5MHz ='1') then
            if(cpt=255) then
                  cpt<=0; --permet de revenir à 0 si le compteur est à 255
            else
                  cpt<=cpt+1; --incrémentation du compteur
            end if;
            if (cpt=255)then
                  H10KHz<='1'; --lorsque le compteur atteint 255 on a un front d'horloge de 10kHz (permet conversion horloge 2.5MHz à 10kHz)
            else
                  H10KHz<='0';
            end if ;
            if(valid_pwm='1') then --commencement du fonctionnement du bloc
               if(cpt < PWM_REF) then
                  PWM_OUT<='1' ; --tant que le compteur n'a pas atteint la référence PWM choisi alors la sortie fonctionne
               else
                  PWM_OUT<='0'; --lorque le compteur a atteint la référence PWM choisi alors la sortie ne fonctionne plus
               end if;
            else
                  PWM_OUT<='0'; --valid est un arrêt d'urgence et permet de mettre la sortie à 0 si valid vaut 0
            end if;
      end if;
   end process P1;
end  PWM_a;
```

Figure 21 - Code VHDL du bloc PWM

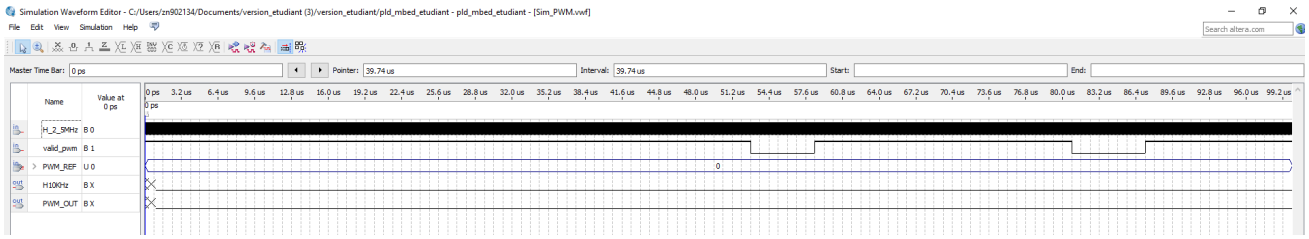


Figure 22 - Simulation du bloc PWM

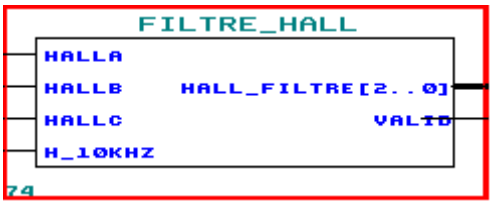


Figure 23 - Bloc Filtre Hall

Ce bloc Filtre hall permet de filtrer numériquement les signaux issus des capteurs hall et qui peuvent être bruités par les signaux de puissance au moment de leur changement d'état :

Entrées :

- HallA, HallB et HallC sont les 3 capteurs halls de position.
- H_10Khz : horloge de base interne de 10Khz.

Sorties :

- Hall_FILTRE[2..0] : Regroupe les entrées indépendantes HallC, HallB et HallA après filtrage.
- Valid : indicateur de signal stable de l'état des capteurs hall (actif à 1).

PWM_REFERENCE.vhd* | COLLECTEUR_NUM2.vhd* | matrix_scooter_pwm_mbed_etudiant_Mbed.bdf | Compilation Report - pld_mbed_etudiant

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity FILTRE_HALL is
port(
    H10KHz, HallA, HallB, HallC: in std_logic;
    valid : out std_logic;
    Hall_Filtre : out std_logic_vector(2 downto 0));
end  FILTRE_HALL;

architecture  FILTRE_HALL_a of  FILTRE_HALL is

signal hall : std_logic_vector(2 downto 0);
signal hall_1 : std_logic_vector(2 downto 0); -- mémorisation sur 1 période d'horloge
signal hall_2 : std_logic_vector(2 downto 0); -- mémorisation sur 2 périodes d'horloges

begin
hall <= HallC & HallB &HallA;

   P1 : process (H10KHz)
   begin
      if(H10KHz'event and H10KHz ='1') then --sur un front montant

            if((hall_2 = hall_1) and ( hall_1= hall)) then --état des capteurs sur la période précédente
               valid <= '1'; --le signal restituer est valide
               Hall_Filtre <= hall; --état des capteurs sur la période actuelle
            else valid <= '0';
            end if;
            hall_2 <= hall_1; -- copie de la valeur des capteur sur la sortie
            hall_1 <= hall; --le signal restituer n'est pas valide
      end if;
   end process P1;
end  FILTRE_HALL_a;
```

Figure 24 - Code VHDL du bloc Filtre hall

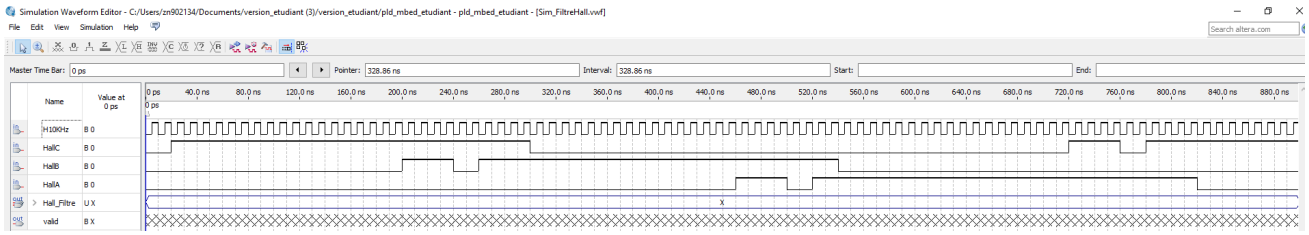


Figure 25 - Simulation du bloc Filtre hall

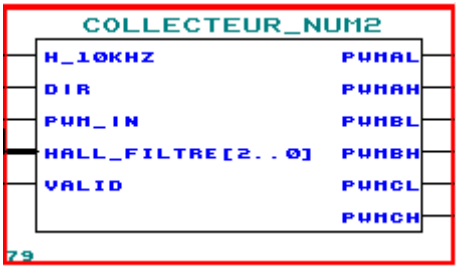


Figure 26 - Bloc collecteur numérique

Ce bloc collecteur numérique permet de commander les 6 transistors de l'onduleur triphasé.
Les transistors du Bas PWM_AL, PWM_BL,PWM_CL sont commandés en mode pleine onde et réalisent la fonction de collecteur numérique. Les sorties PWM_AH, PWM_BH, PWM_CH distribue la consigne de PWM reçue à l'entrée sur les transistors supérieurs de l'onduleur.

Entrées :

- VALID : Le signal valid indique à quel moment les capteurs de halls présentent une valeur stable. Il est actif à 1.
- DIR : permet de choisir le sens de rotation de la machine. Si DIR vaut 1 on est en marche avant AV et si DIR vaut 0 on est en marche arrière AR.
- H_10Khz :horloge de synchronisation générale 10Khz disponible en interne.
- PWM_IN : entrée de PWM modulée en largeur d'impulsion suivant la consigne de pédalage ou de gaz.

Sorties :

- PWMxL et PWMxH : commande logique des transistors de l' onduleur.

PWM_REFERENCE.vhd* | COLLECTEUR_NUM2.vhd* | matrix_scooter_pwm_mbed_etudiant_Mbed.bdf | Compilation Report - pld_mbed_etudiant | bloc_io_mbed.vhd* | PWM.vhd*

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity COLLECTEUR_NUM2 is
port(
   H10KHz,PWM_IN,DIR,valid: in std_logic; --entrées utilisées pour le collecteur
   HALL_FILTRE : in std_logic_vector(2 downto 0); --entrée HALL Filtre qui peut prendre 8 valeurs
   PWM_AH, PWM_BH, PWM_CH, PWM_AL, PWM_BL, PWM_CL: out std_logic); --sorties des transistors
end  COLLECTEUR_NUM2;

architecture  COLLECTEUR_NUM2_a of  COLLECTEUR_NUM2 is

signal SORTIE : std_logic_vector (5 downto 0); --déclaration du signal SORTIE qui varie entre 0 et 32

begin
   P1 : process (H10KHz)
   begin
      if(H10KHz'event and H10KHz ='1') then
         if(valid='1')then
            case HALL_FILTRE is
               when "001" => SORTIE<="100001"; --AH BH CH AL BL CL-- secteur 1 différentes valeur des transistors permettent de changer de secteur
               when "010" => SORTIE<="010100"; --secteur 2
               when "011" => SORTIE<="010001"; --secteur 3
               when "100" => SORTIE<="001010"; --secteur 4
               when "101" => SORTIE<="100010"; --secteur 5
               when "110" => SORTIE<="001100"; --secteur 6
               when others => SORTIE<="000000"; --sinon secteur vaut 0
            end case;
         end if;
      end if;
   end process P1;
   PWM_AL <= SORTIE(2); --affectation des sorties PWM qui distribue la consigne de PWM reçue à l'entrée sur les transistors de l'onduleur
   PWM_BL <=SORTIE(1);
   PWM_CL <= SORTIE(0);
   PWM_AH <= SORTIE(5) and PWM_IN;
   PWM_BH <= SORTIE(4) and PWM_IN;
   PWM_CH <= SORTIE(3) and PWM_IN;

end  COLLECTEUR_NUM2_a;
```

Figure 27 - Code VHDL du bloc collecteur numérique

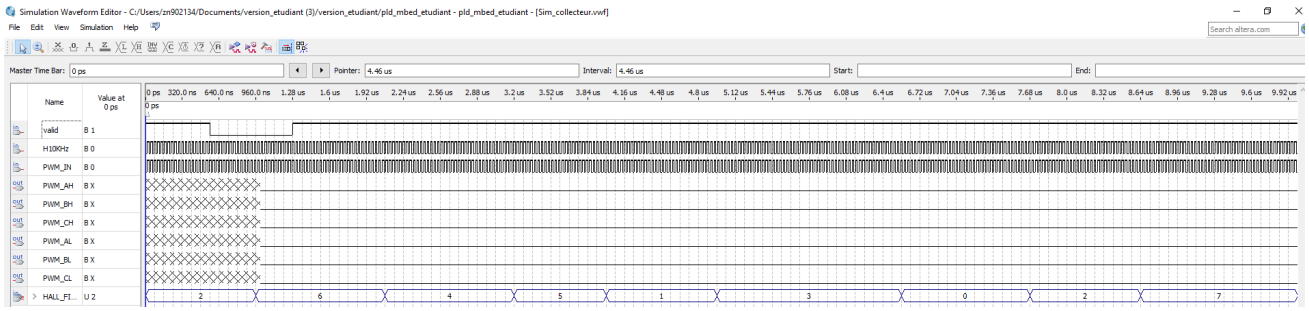


Figure 28 - Simulation du bloc collecteur numérique

2.2. Étude de chacun des blocs situés dans le microcontrôleur mBeD (détail des blocs logiciels internes, codes C associés commentés, résultats des tests, copies d'écran console et conclusion partielle sur chaque bloc).

Nous allons un par un commenter et expliquer les différentes parties de notre code C mis dans le microcontrôleur.
Ces déclarations permettent de faire une relation entre le PLD et le microcontrôleur Mbed sur les bonnes pins.

```
DigitalOut led1(LED1); //initialization of the LEDs present on the Mbed microcontroller
DigitalOut led2(LED2);
DigitalOut led3(LED3); // blink when can message is sent
DigitalOut led4(LED4); // blink when can message is received
DigitalOut valid_pwm(p21);

AnalogIn poignee(p17); //gas grip pin
InterruptIn valid (p22); //top hall pin
AnalogIn Vbat(p18); //battery voltage pin
AnalogIn Ibat(p20); //battery current pin
```

Figure 29 - Code C des pins pour le Mbed

Pour pouvoir utiliser les fonctions que nous allons détailler ultérieurement, nous devons d'abord faire le prototype de toutes ces fonctions.

```
/***************function prototypes************************/
void LecturePLD(void);
void REFPWM(void);
void Vitesse(void);
void tophall(void);
void Calibration(void);
void Bride(void);
void TensionBat(void);
void save(void);
```

Figure 30 - Code C des prototypes des fonctions

Évidemment nous avons déclaré les variables globales qui sont utilisées dans plusieurs parties de notre code C global.

```c
/****************Declaration of global variables***************/
int Dir=1, Hall,Brake, FLTA=0, OverCurrent=0;
float GazMin, GazMax;
int PWMREF;
int PWMREF_manu=0;
float vitesse=0, vitesseTr=0, cpt=0, cptpart, cpttot;
int Mode=0;
int VBride,consigne;
float DistP=0, Disttot, distpart;
float Vbatt, iBat;
```

Figure 31 - Code C des déclarations des variables globales

Nous avons ici la fonction principale avec les initialisations principales.

```c
int main()
{
    FILE *fp = fopen("/local/calib.txt","r"); //need to create and open the calib file
    if( fp== NULL) {
        pc.printf("erreur pas de fichier"); //write this if no file
    } else {
        fscanf(fp,"%f %f %d %f %f",&GazMin, &GazMax, &VBride, &cpttot, &cptpart); //write in file the different variables
    }
    fclose(fp); //close the file

    char cChoix=0; //initialization of the choice
    valid_pwm=0; //initialization of the valid_pwm, if valid_pwm = 0 PWM =0, it is an emergency stop, no external control without µC, bike stopped as long as the valid is at zero
    MyPLD.write(0);
    valid_pwm=1; //Bike can start


    Mode = 0; //automatic mode = 0
    gaz.attach(&REFPWM,0.02); //beginning of the autiomatic mode with the gaz function every 20ms

    valid.rise(&tophall); //tophall rising edge
    Cpt.attach(&Vitesse,DELTA_T); //count the number of rising edge of tophall
```

Figure 32 - Code C de la fonction principale

Le code C ci-dessous nous permet d'avoir un IHM sur teraterm.

Nous pouvons faire le choix en utilisant le clavier et ensuite nous avons la valeur de ce que nous avons demandé.

```
209     while(cChoix!='q' and cChoix!='Q') {            //Display of the proposals of the different possible choices on the console
210         pc.printf(" \r veuillez saisir un choix parmi la liste proposee: \n");
211         pc.printf(" \r a : saisie consigne pwm \n"); //manual mode
212         pc.printf(" \r b : choix mode auto \n"); //automatic mode
213         pc.printf(" \r c : Lecture du PLD \n"); //PLD parameters values
214         pc.printf(" \r d : Lecture de la vitesse \n"); //speed value
215         pc.printf(" \r e : Saisir les parametres de calibration \n"); //write the values of the calibration
216         pc.printf(" \r f : Saisir la valeur de la bride \n"); //value of the bridle
217         pc.printf(" \r g : valeur de la tension \n"); //Value of the battery voltage
218         pc.printf(" \r q : quitter \n"); //Quite the console
219
220         /************* multithreading : main thread need to sleep in order to allow web response */
221         while (pc.readable()==0) { // determine if char availabler
222             Thread::wait(10);   // wait 10 until char available on serial input
223         }
224         /************* end of main thread sleep  ****************/
225
226         pc.scanf(" %c",&cChoix);
227         switch (cChoix) {
228             case 'a':
229                 pc.printf("pwm:");
230                 pc.scanf(" %d",&PWMREF_manu); //Write the value of the PWM chosen
231                 Mode=1;
232                 break;
233             case 'b':
234                 Mode=0;
235                 break;
```

```
236             case 'c':
237                 LecturePLD();
238                 break;
239             case 'd':
240                 pc.printf("vitesse = %1.3f km/h \n\r",vitesse); //write the value of the speed in km/h
241                 pc.printf("vitesse = %1.3f tr/min \n\r",vitesseTr); //write the value of the speed in Tr/min
242                 Disttot= (((cpttot*(2*PI*R)*3.6)/(6*NBPOLES))/1000); //calculation of the total distance
243                 DistP= (((cptpart*(2*PI*R)*3.6)/(6*NBPOLES))/1000); //calculation of the partial distance
244                 pc.printf("Distance totale = %1.3f km \n\r",Disttot); //Write the value of the total distance
245                 pc.printf("Distance partielle = %1.3f km \n\r",DistP); //Write the value of the partial distance
246                 break;
247             case 'e':
248                 Calibration(); //execution of the calibration function
249                 break;
250             case 'f':
251                 Bride(); //execution of the Bride function
252                 break;
253             case 'g':
254                 TensionBat(); //execution of the TensionBat function
255                 break;
256
257             case 'q':
258                 valid_pwm.write(0); //Bike stopped
259                 gaz.detach(); //
260                 Cpt.detach();
261                 save(); ////execution of the save function
262                 break;
```

Figure 33 - Code C du choix avec l'IHM

La fonction LecturePLD nous permet de savoir si le frein est actif ou non, si nous sommes en marche avant ou en marche arrière, s'il y a une erreur de convertisseur ou encore s'il y a un court-circuit. De plus, nous avons dû utiliser la méthode read pour lire les valeurs de ces différents capteurs.

```c
void LecturePLD(void)   // Display of the brake status, the sector in which we are, if we are in C.C. and FLTA
{
    Hall=MyPLD.read(); //allows you to read the bits of the pld
    Brake=MyPLD.read()&0x20;
    Dir=MyPLD.read()&0x08;
    OverCurrent=MyPLD.read()&0x40;

    pc.printf("Secteur Hall = %d\n\r",Hall&0x07); //Sector from 0 to 6 therefore on 7bits in hexa
    if(Dir==0) {
        pc.printf("Direction = Marche arriere \n\r");
    } else {
        pc.printf("Direction = Marche avant \n\r");
    }
    if(Brake==0) {
        pc.printf("Frein = OUI\n\r");
    } else {
        pc.printf("Frein = NON \n\r");
    }
    if(OverCurrent==0) {
        pc.printf("OverCurrent = OUI\n\r");
    } else {
        pc.printf("OverCurrent = NON\n\r");
    }
    if(FLTA==0) {
        pc.printf("FLTA = Pas d'erreur convertisseur\n\r");
    } else {
        pc.printf("FLTA = Erreur convertisseur\n\r");
    }
}
```

Figure 34 - Code C de la fonction LecturePLD

Cette fonction permet le fonctionnement de la roue que ce soit en mode manuel ou en mode auto.

```c
void REFPWM(void)
{

    float a,b;
    Brake=MyPLD.read()&0x20;
    if(Mode==1) {                       //manual mode
        MyPLD.write(PWMREF_manu);
    } else if(Mode==0) {                //automatic mode
        a=255/(GazMax-GazMin);
        b=(-a)*GazMin;
        PWMREF =a*poignee.read()+b;
        if(PWMREF<0) {
            PWMREF=0;
        }
        if(PWMREF>255) {
            PWMREF=255;
        }
        if(Brake==0) {
            consigne=0;
        } else {
            if(VBride>0 && VBride<=vitesse) {
                if(vitesse>0) {
                    consigne--;
                }
            } else if(consigne<PWMREF) {
                consigne++;
            } else {
                consigne= PWMREF;
            }
        }
        if(consigne<0) {
            consigne=0;
        }
        if(consigne>255) {
            consigne=255;
        }
        MyPLD.write(consigne);
    }
}
```

Figure 35 - Code C de la fonction REFPWM

La fonction Vitesse nous permet de savoir à quelle vitesse de nous en temps réel grâce à des tickers et grâce à la fonction tophall.

```c
void Vitesse(void)
{
    vitesse=cpt*10*((2*PI*R)*3.6)/(6*NBPOLES+DELTA_T); //Retrieve number of edges in 100 ms (x10 to obtain number of edges over 1s)
                                                       //km / h speed calculation
    vitesseTr=(cpt*60)/(6*NBPOLES*DELTA_T); // speed in Tr/min
    cpttot+=cpt;
    cptpart=cptpart+cpt;
    cpt=0;
    cptpart=0;
}
```

Figure 36 - Code C de la fonction Vitesse

Cette fonction permet de compter le nombre front montant et elle est utilisée dans la fonction Vitesse.

```c
void tophall(void)
{
    cpt++; //tophall counter increment
}
```

Figure 37 - Code C de la fonction tophall

La fonction Calibration nous permet de savoir et garder en mémoire la valeur de la poignée max et de la poignée de gaz min. Seulement quand les calibrations sont faites, le vélo peut démarrer avec la poignée de gaz.

```c
void Calibration(void)
{
    char c,m;
    valid_pwm.write(0); //bike stationary
    pc.printf("\nmettre la poignee au max et appuyer sur 'f' \n\r"); //Put the gas handle to the max
    do {
        m=getchar();
    } while(m!='f'); //put f
    GazMax=poignee.read(); //read the value of the max grip gaz
    pc.printf("gazmax= %1.3f\n\r",GazMax);

    pc.printf("\nmettre la poignee au min et appuyer sur 'c' \n\r"); //Put the gas handle to the min
    do {
        c=getchar();
    } while(c!='c'); //put c
    GazMin=poignee.read();
    pc.printf("gazmin= %1.3f\n\r",GazMin); //read the value of the min grip gaz
    FILE *fp = fopen("/local/calib.txt","w");
    if( fp== NULL) {
        pc.printf("erreur pas de fichier");
    } else {
        fprintf(fp," %f %f ",GazMin, GazMax); //write the values in the calib file
    }
    fclose(fp);
    valid_pwm.write(1); //bike can start if min and max gas grip are taken
}
```

Figure 38 - Code C de la fonction Calibration

La fonction bride nous permet de rester à une vitesse constante choisie.

```c
void Bride(void)
{
    printf("Saisir une valeur de Bride en km/h");
    pc.scanf("%d",&VBride);
    /*FILE *fp= fopen("/local/calib.txt","w");
    fprintf(fp,"%f%f%d%f",GazMin,GazMax,VBride,Disttot);
    fclose(fp);*/

}
```

Figure 39 - Code C de la fonction Bride

La fonction save nous permet de sauvegarder les valeurs de la poignée maximum et minimum, ainsi que les compteurs ainsi que la bride choisie auparavant.

```c
void save(void) //allows you to save the values of the flange, the max and min handle, the partial and total counters
{
    FILE *fp= fopen("/local/calib.txt","w");
    if( fp== NULL) {
        pc.printf("erreur pas de fichier");
    } else {

    fprintf(fp,"%f %f %d %f %f",GazMin,GazMax,VBride,cpttot,cptpart);
    }
    fclose(fp);
```

Figure 40 - Code C de la fonction save

```c
void TensionBat(void)
{
    Vbatt = Vbat.read()*78.61; //x78.61 to have to real value of the voltage
    pc.printf("\n tension batterie = %f V\n",Vbatt);
    iBat = ((61*Ibat.read()*3.3)/33-2.5)/0.185F; //I=U/R So reverse voltage divider bridge
    pc.printf("\n courant batterie = %f A\n",iBat); //write the value of the battery current
}
```

Figure 42 - Code C de la fonction TensionBat

3. Dossier de fabrication logiciel et matériel

3.1. Code C logiciel complet et commenté si possible en anglais

Voici ici le code C complet que nous avons précédemment commenté et expliqué en détail bloc par bloc.

main.cpp

```
//#include "EthernetInterface.h"
#include <stdlib.h>
#include <string.h>
#include "mbed.h"
#include "rtos.h" // need for main thread sleep
#include "html.h" // need for html patch working with web server
#include "bloc_io.h"
#define RADIUS  0.2F // wheel size
#define NBPOLES 8 // magnetic pole number
#define DELTA_T 0.1F // speed measurement counting period
#define R 0.2F //Bike wheel radius
#define PI 3.14159265359F
#define SensiI 0.185 // 185mV/A : battery current sensor sensitivity
//#define VMAX 31 // Max speed in km/h

Bloc_IO MyPLD(p25,p26,p5,p6,p7,p8,p9,p10,p23,p24);// instantiate object needed to communicate with PLD
// analog input connected to mbed
// valid pmw mbed pin
Serial pc(USBTX, USBRX); // tx, rx
// Top_Hall Pin



/************ persistent file parameters section *****************/
LocalFileSystem local("local");               // Create the local filesystem under the name "local"





/********************* web server section **********************************/

var_field_t tab_balise[10];  //a tag is present in the skeleton
int giCounter=0;// acces counting



/************************ can bus section  ************/
// determine message ID used to send Gaz ref over can bus
#define _CAN_DEBUG // used to debug can bus activity
//#define USE_CAN_REF // uncomment to receive gaz ref over can_bus
CAN can_port (p30, p29); // CAN bus initialization on the pins 30 (rd) et 29(td) for lpc1768 + mbed shield
```

```
45 bool bCan_Active=false;
46
47
48
49 DigitalOut led1(LED1); //initialization of the LEDs present on the Mbed microcontroller
50 DigitalOut led2(LED2);
51 DigitalOut led3(LED3); // blink when can message is sent
52 DigitalOut led4(LED4); // blink when can message is received
53 DigitalOut valid_pwm(p21);
54
55 AnalogIn poignee(p17); //gas grip pin
56 InterruptIn valid (p22); //top hall pin
57 AnalogIn Vbat(p18); //battery voltage pin
58 AnalogIn Ibat(p20); //battery current pin
59
60
61
62 /***************function prototypes*************************/
63 void LecturePLD(void);
64 void REFPWM(void);
65 void Vitesse(void);
66 void tophall(void);
67 void Calibration(void);
68 void Bride(void);
69 void TensionBat(void);
70 void save(void);
71
72
73
74 /****************Declaration of global variables***************/
75 int Dir=1, Hall,Brake, FLTA=0, OverCurrent=0;
76 float GazMin, GazMax;
77 int PWMREF;
78 int PWMREF_manu=0;
79 float vitesse=0, vitesseTr=0, cpt=0, cptpart, cpttot;
80 int Mode=0;
81 int VBride,consigne;
82 float DistP=0, Disttot, distpart;
83 float Vbatt, iBat;
84
85
```

main.cpp

```
85
86
87
88 //************ local function prototypes ******************
89 Ticker gaz; //gas grip ticker declaration
90 Ticker Cpt; //counter ticker declaration
91 Ticker Vit; //speed ticker declaration
92 Ticker Cptpart; //partial counter ticker declaration
93
94
95
96
97
98 /**************** Read persistent data from text file located on local file system ****************/
99
100
101
102 /**************** write persitant  data to text file located on local file system ****************/
103
104
105
106
107
108
109 //************** calibation gaz function needed to record min_gaz and max_gaz value to persistent text file  *****************
110
111
112 // ************top hall counting interrupt needed for speed measurement
113
114
115 //********************** timer interrupt for speed measurement each 100ms  *************************
116
117
118
119
120
121
122 //********************* Timer Interrupt for gaz ref management each 10ms   ********************
123
124
125
126 /********* main cgi function used to patch data to the web server thread **********************************/
127 /*void CGI_Function(void) // cgi function that patch web data to empty web page
128 {
```

```
{
    char m;


    a_chaine4[20]= {}; // needed to form html response

}

*/
/********************** CAN BUS SECTION  **********************/




void CAN_REC_THREAD(void const *args)
{
    int iCount,iError;

    while (bCan_Active) {
        Thread::wait(100);// wait 100ms
        // code todo

    }

}



//************************** main function ****************************************
int main()
{
    FILE *fp = fopen("/local/calib.txt","r"); //need to create and open the calib file
    if( fp== NULL) {
        pc.printf("erreur pas de fichier"); //write this if no file
    } else {
        fscanf(fp,"%f %f %d %f %f",&GazMin, &GazMax, &VBride, &cpttot, &cptpart); //write in file the different variables
    }
    fclose(fp); //close the file

```

```
                Calibration(); //execution of the calibration function
                break;
            case 'f':
                Bride(); //execution of the Bride function
                break;
            case 'g':
                TensionBat(); //execution of the TensionBat function
                break;

            case 'q':
                valid_pwm.write(0); //Bike stopped
                gaz.detach(); //
                Cpt.detach();
                save(); ////execution of the save function
                break;
        }


    } // end while


    //************** thread deinit *********************
    //DeInit_Web_Server();
    //bCan_Active=false;
    //CanThread=false;// close can received thread
    pc.printf(" fin programme scooter mbed \n");
} // end main


void LecturePLD(void)   // Display of the brake status, the sector in which we are, if we are in C.C. and FLTA
{
    Hall=MyPLD.read(); //allows you to read the bits of the pld
    Brake=MyPLD.read()&0x20;
    Dir=MyPLD.read()&0x08;
    OverCurrent=MyPLD.read()&0x40;

    pc.printf("Secteur Hall = %d\n\r",Hall&0x07); //Sector from 0 to 6 therefore on 7bits in hexa
    if(Dir==0) {
        pc.printf("Direction = Marche arriere \n\r");
    } else {
        pc.printf("Direction = Marche avant \n\r");
    }
    if(Brake==0) {
        pc.printf("Frein = OUI\n\r");
```

```
290         pc.printf("Frein = OUI\n\r");
291     } else {
292         pc.printf("Frein = NON \n\r");
293     }
294     if(OverCurrent==0) {
295         pc.printf("OverCurrent = OUI\n\r");
296     } else {
297         pc.printf("OverCurrent = NON\n\r");
298     }
299     if(FLTA==0) {
300         pc.printf("FLTA = Pas d'erreur convertisseur\n\r");
301     } else {
302         pc.printf("FLTA = Erreur convertisseur\n\r");
303     }
304 }
305
306 void REFPWM(void)
307 {
308
309     float a,b;
310     Brake=MyPLD.read()&0x20;
311     if(Mode==1) {                   //manual mode
312         MyPLD.write(PWMREF_manu);
313     } else if(Mode==0) {            //automatic mode
314         a=255/(GazMax-GazMin);
315         b=(-a)*GazMin;
316         PWMREF =a*poignee.read()+b;
317         if(PWMREF<0) {
318             PWMREF=0;
319         }
320         if(PWMREF>255) {
321             PWMREF=255;
322         }
323         if(Brake==0) {
324             consigne=0;
325         } else {
326             if(VBride>0 && VBride<=vitesse) {
327                 if(vitesse>0) {
328                     consigne--;
329                 }
330             } else if(consigne<PWMREF) {
331                 consigne++;
332             } else {
```

```
            consigne++;
        } else {
            consigne= PWMREF;
        }
    }
    if(consigne<0) {
        consigne=0;
    }
    if(consigne>255) {
        consigne=255;
    }
    MyPLD.write(consigne);
  }
}


void Vitesse(void)
{
    vitesse=cpt*10*((2*PI*R)*3.6)/(6*NBPOLES+DELTA_T); //Retrieve the number of edges in 100 ms
                                        //(x10 to obtain the number of edges over 1s) + km / h speed calculation
    vitesseTr=(cpt*60)/(6*NBPOLES*DELTA_T); // speed in Tr/min
    cpttot+=cpt;
    cptpart=cptpart+cpt;
    cpt=0;
    cptpart=0;
}

void tophall(void)
{
    cpt++; //tophall counter increment
}

void Calibration(void)
{
    char c,m;
    valid_pwm.write(0); //bike stationary
    pc.printf("\nmettre la poignee au max et appuyer sur 'f' \n\r"); //Put the gas handle to the max
    do {
        m=getchar();
    } while(m!='f'); //put f
    GazMax=poignee.read(); //read the value of the max grip gaz
    pc.printf("gazmax= %1.3f\n\r",GazMax);

    pc.printf("\nmettre la poignee au min et appuyer sur 'c' \n\r"); //Put the gas handle to the min
```

```
    pc.printf("\nmettre la poignee au min et appuyer sur 'c' \n\r"); //Put the gas handle to the min
    do {
        c=getchar();
    } while(c!='c'); //put c
    GazMin=poignee.read();
    pc.printf("gazmin= %1.3f\n\r",GazMin); //read the value of the min grip gaz
    FILE *fp = fopen("/local/calib.txt","w");
    if( fp== NULL) {
        pc.printf("erreur pas de fichier");
    } else {
        fprintf(fp," %f %f ",GazMin, GazMax); //write the values in the calib file
    }
    fclose(fp);
    valid_pwm.write(1); //bike can start if min and max gas grip are taken
}

void Bride(void)
{
    printf("Saisir une valeur de Bride en km/h");
    pc.scanf("%d",&VBride);
    /*FILE *fp= fopen("/local/calib.txt","w");
    fprintf(fp,"%f%f%d%f",GazMin,GazMax,VBride,Disttot);
    fclose(fp);*/

}
void save(void) //allows you to save the values of the flange, the max and min handle, the partial and total counters
{
    FILE *fp= fopen("/local/calib.txt","w");
    if( fp== NULL) {
        pc.printf("erreur pas de fichier");
    } else {

    fprintf(fp,"%f %f %d %f %f",GazMin,GazMax,VBride,cpttot,cptpart);
    }
    fclose(fp);
}

void TensionBat(void)
{
    Vbatt = Vbat.read()*78.61; //x78.61 to have to real value of the voltage
    pc.printf("\n tension batterie = %f V\n",Vbatt);
    iBat = ((61*Ibat.read()*3.3)/33-2.5)/0.185F; //I=U/R So reverse voltage divider bridge
    pc.printf("\n courant batterie = %f A\n",iBat); //write the value of the battery current
```

3.2. Code VHDL et assemblage graphique complet et commenté

bdf = block diagram file complet

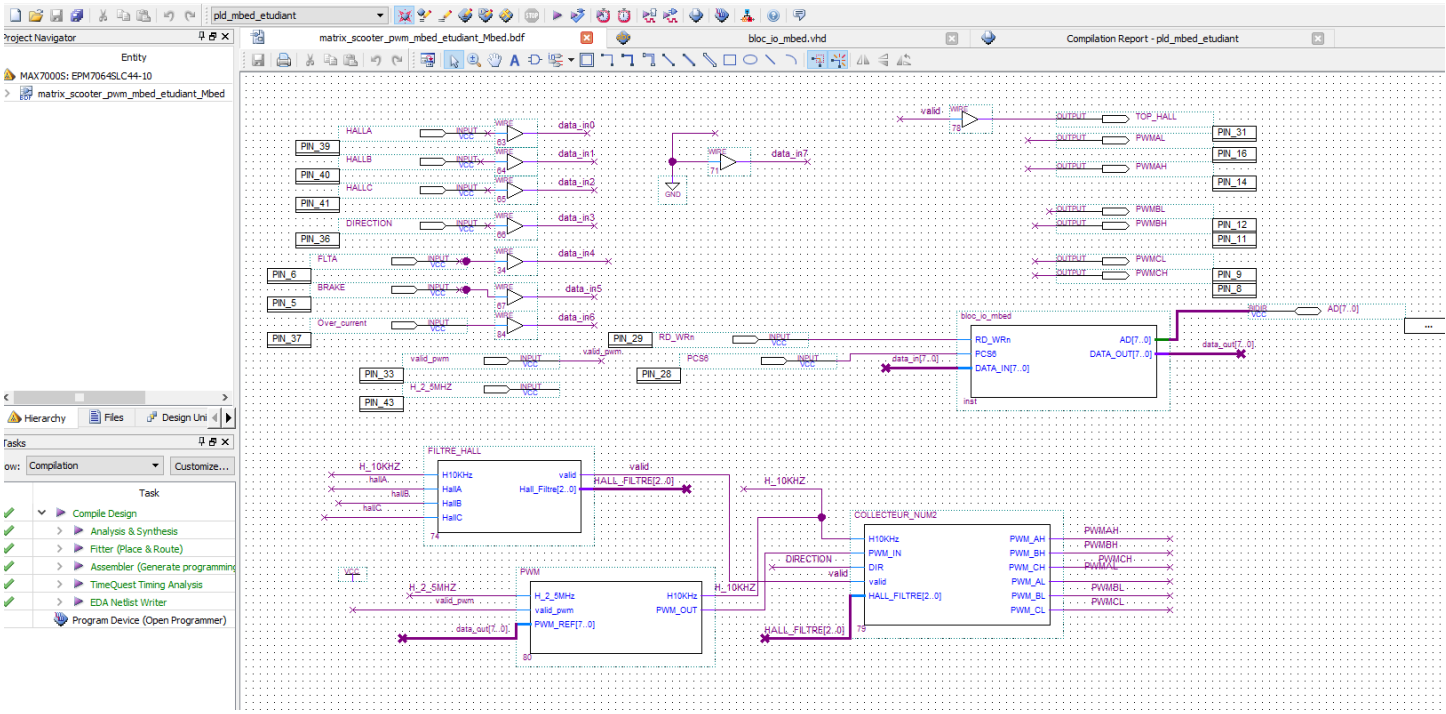


Figure 43- block diagram file complet

Nous pouvons voir ici les quatre blocs que nous avons créés et testés(séparément).

Il s'agit des quatre bloc suivants:

-Filtre_hall

-PWM

- Collecteur_num

-Bloc_io

Nous les avons assemblées afin de réaliser la structure que nous voyons ci-dessus qui est la base de notre projet. Tout ceci sera par la suite piloté par notre carte mbed à l'aide de notre code.

Une de nos premières validation a été de faire tourner notre roue de vélo à l'aide de ce bloc et uniquement de ce bloc. C'est-à-dire sans prendre en compte les données que nous allons envoyées par la suite avec notre carte mbed. Nous avions réalisé cela en séparant le bloc_io du bdf et en le remplaçant par une VCC. Puis une fois le test effectué nous avons pu remettre celui-ci dans le bdf pour que nous puissions continuer notre projet.

3.3. Notice d'utilisation du menu console, plan de câblage du banc de test et procédure de calibration/réglages

Plan de câblage du banc de test :

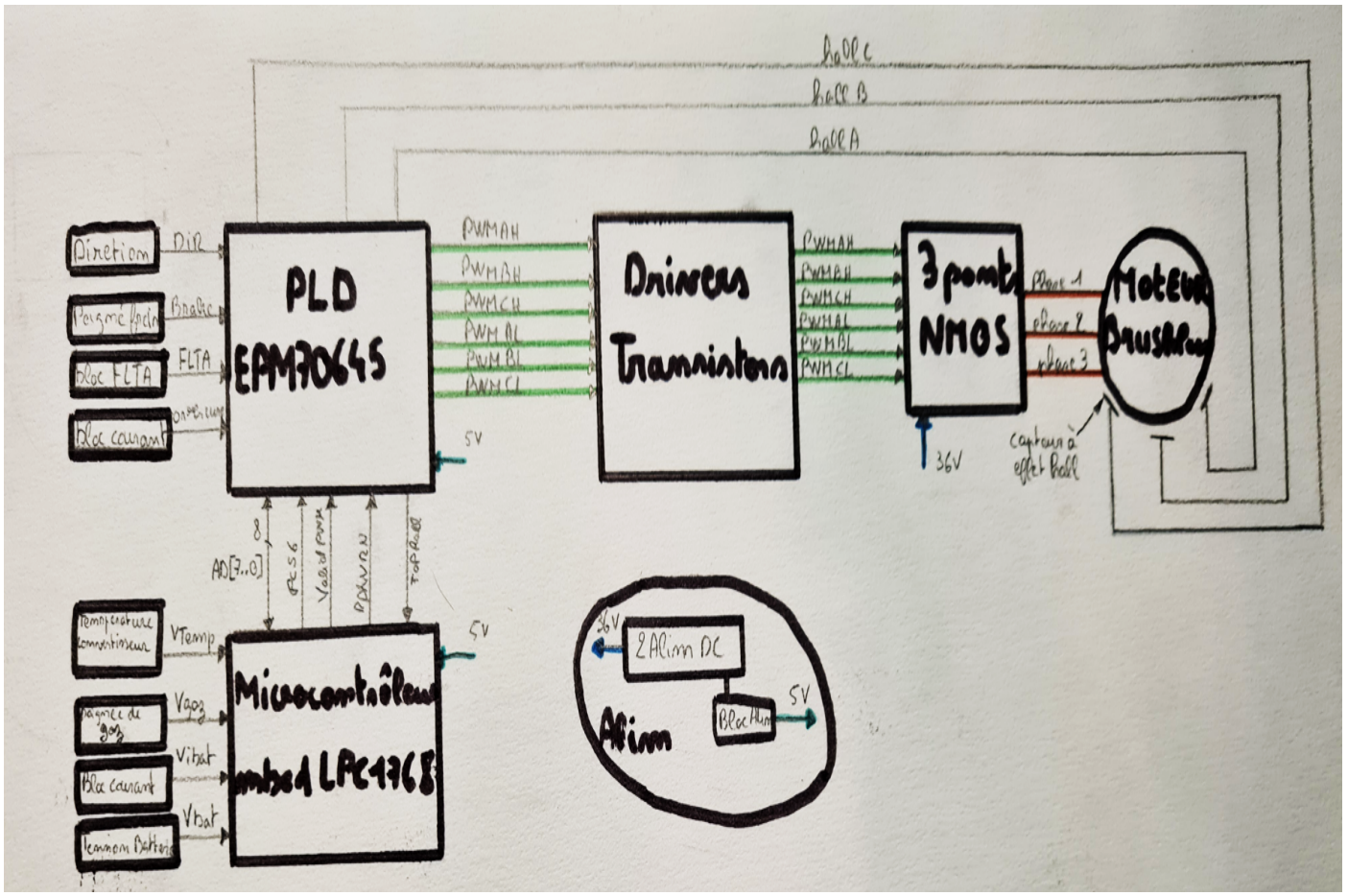


Figure 44 -Plan de câblage du banc de test

Notice utilisation du menu console:

-démarrage du programme.

-effectuer la calibration de la poignée de gaz : c'est -à -dire, calibration de Gaz_Min et de Gaz Max. Ceux-ci seront par la suite enregistrés dans un fichier et stockés.

-Mise en place, où non, d'une bride en vitesse. Qui seront également enregistrés puis stockés dans le même fichier que les valeurs de calibration de la poignée de gaz.

-Une fois toutes ces calibration effectuer le menu déroulant nous propose plusieurs options: Mesure de la vitesse, mesure de la batterie, mesure du nombre de kilomètres parcourus.

4. Proposition d'amélioration et conclusion

4.1. Conclusion et tableau de comparaison entre le travail réalisé et les spécifications globales attendues dans le cahier des charges : détail du travail restant à réaliser

Nous n'avons malheureusement pas pu finir tous les objectifs demandés.
Voici ci-dessous ce que nous avons réussi à faire avec le temps qui nous était donné.

Séance	Objectif	Solution	Accomplissement en %
1	Analyse du sujet. faire tourner la roue en marche avant et marche arrière.	mettre en place un code vhdl sous quartus	100%
2	Mise en place d'un filtre numérique des signaux	-réaliser l'assemblage des blocs sous quartus. -commencer à importer le code en langage c sous Mbed compiler.	100%
3	Mise en place du code en langage c sur mbed compiler.	-réalisation de plusieurs test pour faire tourner la roue	100%
4	Terminer l'assemblage de 3 blocs logiques (PLD)	-Mise en place d'une IHM sur TeraTerm -Faire tourner la roue avec une ref pwm interne au PLD. -Mise en place du mbed et commande de la PWM à partir de la programmation console mbed. -Mise en place du mode auto et du mode manuel.	100%

5	Commande en vitesse variable de la puissance de la poignée de gaz.	-Relever GAZ min et GAZ max. -Lecture du registre interne du PLD. -Gestion du frein dans la commande en vitesse variable.	100%
6	-Affichage du registre interne PLD. -Conditionnement de la poignée de gaz avec sauvegarde dans un fichier texte persistant. -Commande de la poignée avec la gestion du frein.	-mise en place d'un switch case pour l'affiche du registre interne du PLD. -Mise en place d'un capteur de vitesse (très cher !) -Déterminer la vitesse du véhicule par la mesure de la rotation de la roue.	100%
7	-Commande progressive avec poignée de gaz. -Gestion bride en vitesse	-Gestion de la commande progressive avec un PWM auto brut et un PWM auto final. -Modifier le programme en langage c afin d'instaurer des conditions pour réaliser la bride par rapport à la vitesse obtenue.	100% -Nous avons réalisé et validé la commande progressive. -Nous avons réussi à faire fonctionner la bride mais pas avec le bon codage donc nous ne l'avons pas validé.
8	-Mesure du nombre de kilomètres parcourus.	-Compteur de distance total. -Compteur de distance partielle. -Nombre de tophall comptabilisés.	100%
9	-Conditionnement 4 capteurs analogiques.	-Conditionner : iBat, vbat, température en degrés, % de la	50%

		poignée de gaz.	
10	-Mise en place de la jauge d'énergie restante (batterie).	-Prévoir option de charge initiale. -Prévoir option d'affichage de la capacité de la batterie.	0%
11	Adapter l'affichage pour un utilisateur grand public.	-Gestion d'un affichage sur un écran LCD. -Gestion de l'affichage sur la tablette du client. (on ne fourni pas le hardware du client d'affichage) -Utiliser un client d'affichage natif de la plate-forme d'affichage (utiliser un navigateur web).	0%
12	Gestion d'un capteur actionneur communicant par bus de terrain(Bus CAN).	-Neutralisation de l'utilisation des connexions cablées, donc multiplexage des infos sur un seul câble. -Décentraliser la commande de gaz.	0%
13	Finitions	X	X
14	Finitions	X	X

4.2. Propositions d'amélioration du projet.

Nous vous proposons plusieurs choses pour améliorer notre projet. Premièrement nous proposons de faire des fiches déjà préparer à l'avance pour les différentes étapes de validation pour que nous nous consacrions plus de temps sur l'avancement de notre projet.
Deuxièmement, nous pourrions nous même créer notre propre logiciel android dans lequel nous pourrions mettre notre menu console.
De plus, nous pourrions en apprendre davantage sur les différentes méthodes utilisées par des entreprises déjà existantes.
Nous pensons aussi que de passer tout notre projet dans un vrai vélo ou une vraie moto électrique et pouvoir le tester en temps réel serait un plus et beaucoup plus intéressant.

4.3. Conclusion générale sur le plan technique et personnelle

Nous voyons à l'aide de ce tableau que nous n'avons malheureusement pas pu terminer tout notre projet avec le temps qui nous a été donné. Néanmoins nous avons pris le temps de bien comprendre notre sujet qui était très intéressant et très instructif. Ce projet nous a également permis d'approfondir de nombreuses connaissances dans différents modules que nous étudions en GEII. Notamment l'anglais, l'informatique, le système électronique et parfois les mathématiques.
De plus, pour ce projet nous avons travaillé en équipe et cela à été un excellent entraînement pour notre avenir. Car dans notre domaine la majorité des projets réalisés dans les entreprises se font en équipe. Cela permet de limiter la marge d'erreur ainsi que d'être plus réactif en cas d'obstacle et donc d'avancer plus rapidement. Pour finir nous avons pris énormément de plaisir à réaliser ce projet mais avec le regret de ne pas avoir pu le terminer.