CHAN HAOR KING
ALLARD DE GRANDMAISON

2022-2023

Rapport de Projet

SAE ROUE

NICE CÔTE D'AZUR
UNIVERSITÉ CÔTE D'AZUR

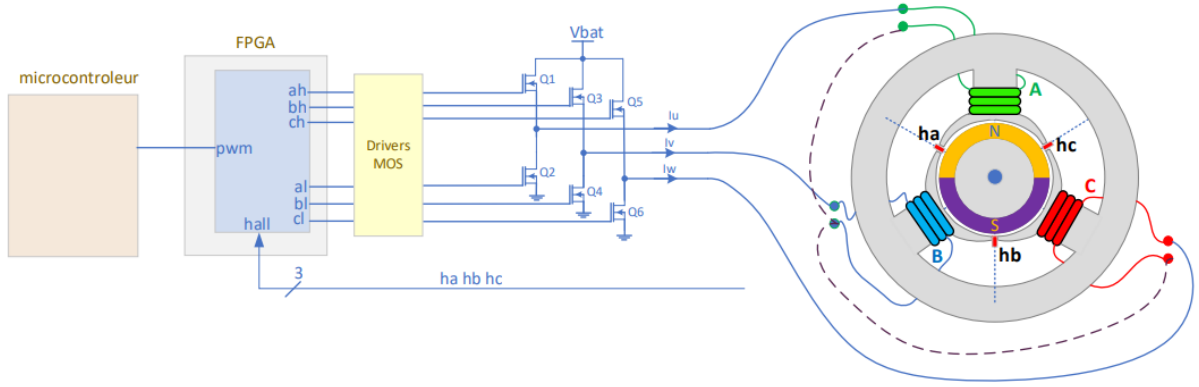
Introduction

Lors du BUT3, en Situation d'Apprentissage Évaluée, nous avons comme projet de contrôler un moteur Brushless relié à une roue de vélo.

Pour cela nous allons mettre en œuvre un banc didactique VAE et le programmer. Il sera composé d'un mbed et d'une carte fpga.

Ce rapport sera divisé en trois parties. Nous verrons le fonctionnement du moteur brushless, ensuite l'utilité des différents éléments mis à disposition et enfin la partie code.

1.Le moteur Brushless

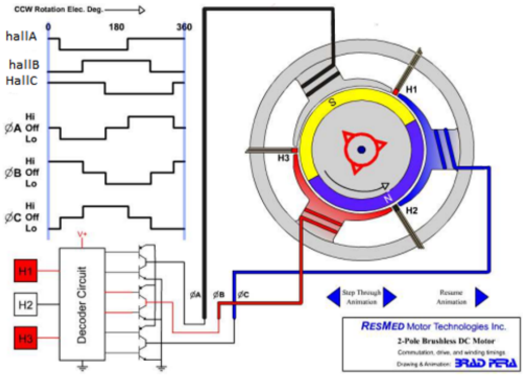


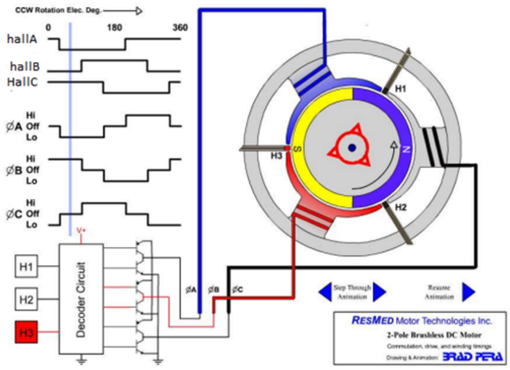
1.1 Principe du moteur

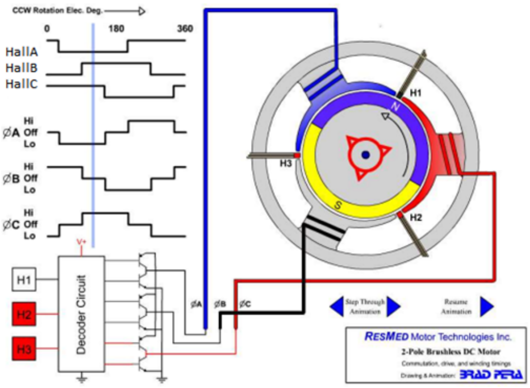
Un moteur brushless signifie que c'est un moteur qui ne possède pas de balais, donc pas de charbon ni de frottement. Ce sont des moteurs avec des rendements bien plus intéressants, plus petits, plus efficaces mais qui coûtent nécessairement plus cher.
Un moteur brushless se compose d'un rotor magnétique (avec 2 pôles comme sur le schéma ci-dessus) et d'un certain nombre de pôles. Plus il y a de pôles, moins on aura d'à-coup car on pourra de façon précise envoyer les commandes correspondantes. Cependant, plus de pôles veut également dire qu'il y aura plus de demi-ponts. Dans ce projet le stator se compose de 3 demi-ponts, nous avons donc 3 pôles.

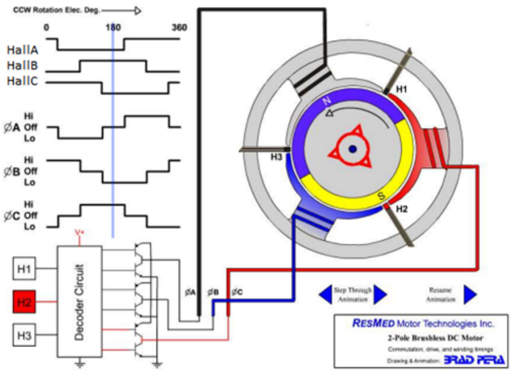
En fonction des commandes envoyées, on va exciter 2 pôles dans un certain sens, entraînant 2 champs magnétiques qui forment une résultante qui appliquera sa force au rotor.
Afin de savoir quels pôles activer, il faut connaître la position angulaire du moteur pour savoir quelle phase est excitée. Pour cela nous utilisons des capteurs Halls pour détecter le pôle nord du rotor et ainsi savoir précisément sa position puisqu'il n'y a que 6 possibilités :

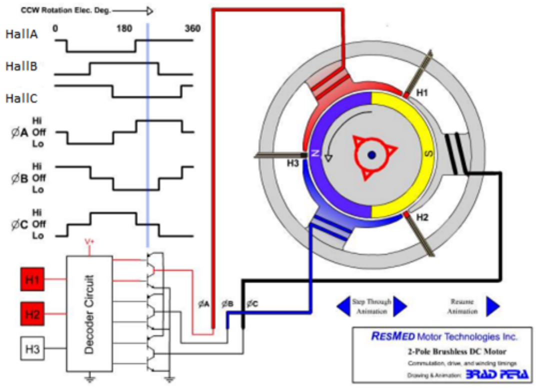
Position	Combinaison de capteur
n°1	ha
n°2	ha & hb
n°3	hb
n°4	hb & hc
n°5	hc
n°6	hc & ha

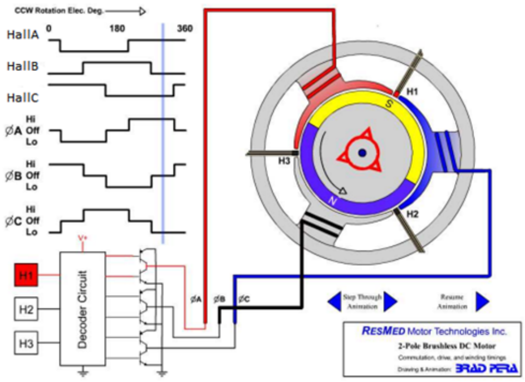






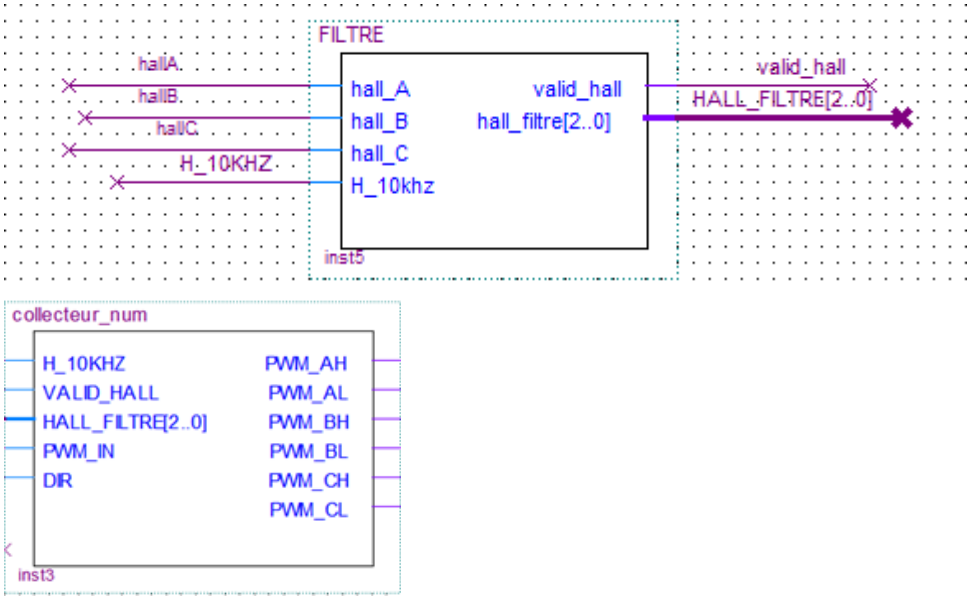






1.2 Commande du moteur

Voici les blocs que nous avons créés pour la commande du moteur. Une explication plus explicite sera fournie plus loin.



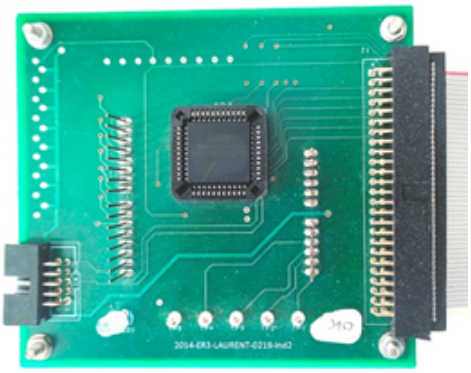
2. Étude matérielle et logicielle

2.1 Etude matérielle

Il faut en premier lieu comprendre les contraintes matérielles : nous avons une roue avec des capteurs qui nous renvoient la position du rotor, et des commandes à fournir pour faire tourner la roue dans le sens voulu. Nous avions accès à deux cartes. Un fpga ALTERA MAX7000S et une mbed LPC1768.

Qu'est-ce qu'un fpga ?
Le FPGA (Field Programmable Gate Arrays) est un PLD (Programmable logic device). Il contient des semi-conducteurs basés sur des blocs logiques configurables et programmables. Celui qu'on utilise est le ALTERA MAX7000S. Le logiciel utilisé pour le programmer est Quartus II 17.0, en VDHL ou VLOG.

Ensuite nous avons le mbed qui est un microcontrôleur. Il servira à recevoir les informations de la poignée au travers du fpga par exemple.

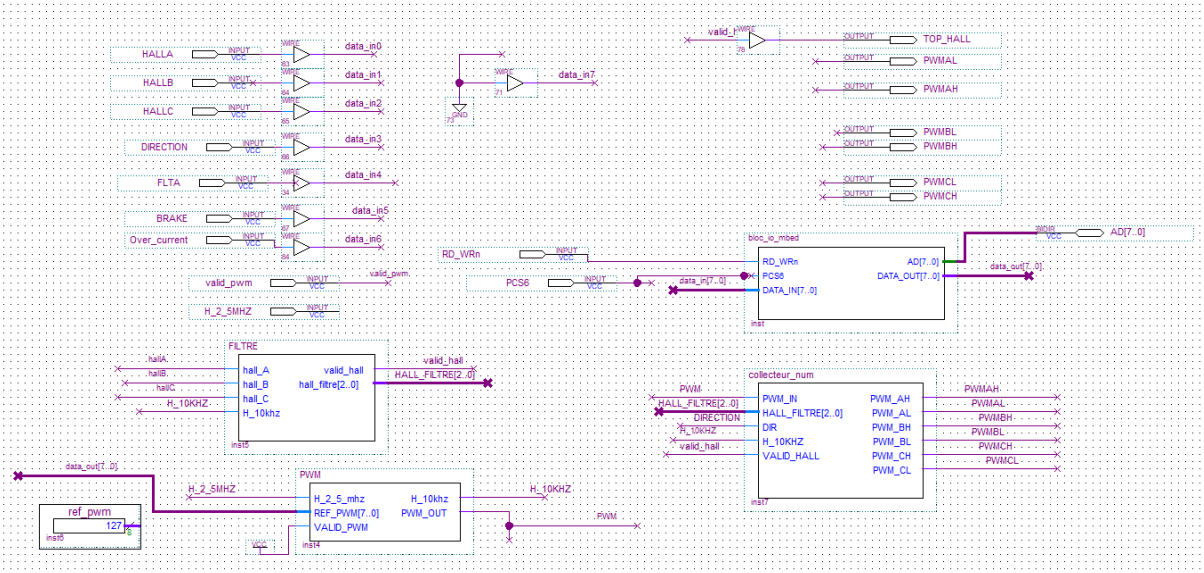


ALTERA MAX7000S

Mbed LPC1768.

2.2 Etude logicielle

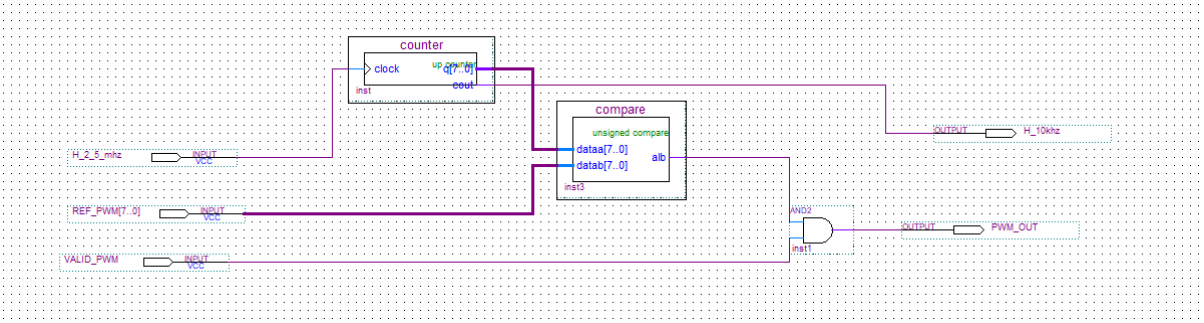
Quartus II 17.0



Dans la carte FPGA

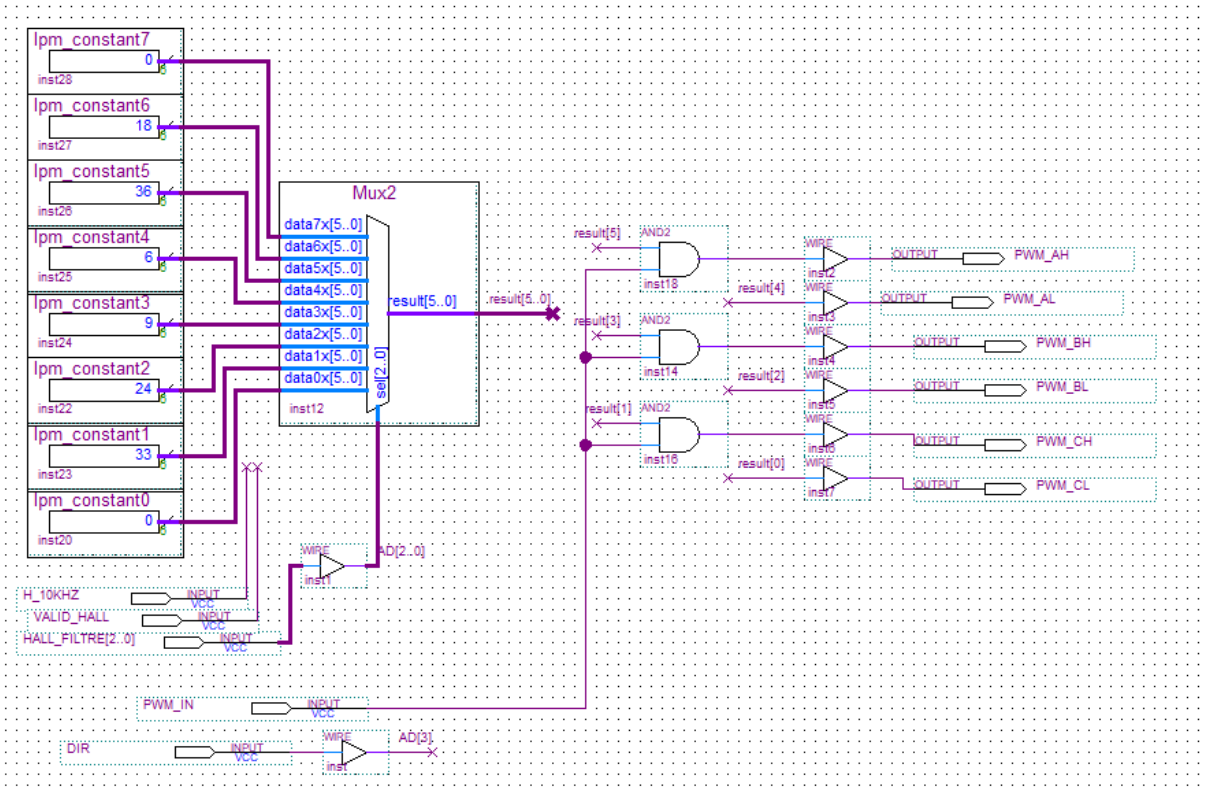
Cette image est celle du programme que nous avons téléversé dans le FPGA.
Nous voyons en haut à gauche les différentes entrées de notre système. Ces blocs *filtre, collecteur_num* et *PWM* ont été programmés par nous même.

PWM



Cette fonction réalise à partir d'une fréquence de 2.5Mhz une fréquence de 10kHz.

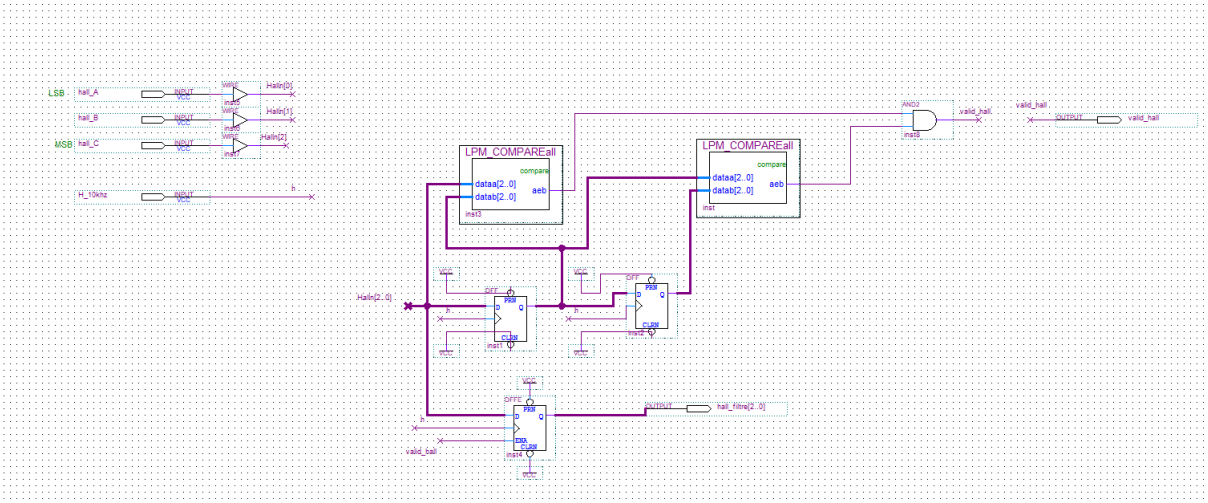
“Collecteur Num”



Multiplexeur: permet la sélection d’une entrée à partir de plusieurs.

Ici, le “HALL_FILTRE” sert à sélectionner à travers du MUX2 une des huit constantes, afin de renvoyer vers la sortie un mot binaire sur 6 bits qui servira à actionner les sorties correspondantes. Cela enverra une tension au bon endroit afin de faire tourner la roue.

Filtre



Ce filtre permet d'éliminer les perturbations électromagnétiques qui pourraient "fausser" les signaux des capteurs.

Sur ce schéma il y a :

- quatres entrées dont trois capteurs hall et un signal à 10 kHz.
- deux LPM compare
- trois PRN DFF
- un AND

Les LPM compare sont des comparateur et permettent de comparer deux entrées.
Les PRN DFF sont des types de bascule qui permettent de stocker des valeurs binaires.

Ce filtre reçoit en entrée les signaux des capteurs Hall et un signal à 10 kHz, fréquence qui permet d'éviter les perturbations électromagnétiques tout en traitant les signaux des aimants plus rapidement.

Ces signaux vont être comparés une première fois puis une dans un autre intervalle va être comparé une seconde fois. Si les deux comparateurs donnent le même signal à la porte AND, alors la sortie sera valide (valid_hall).

Ce valid_hall activera le Enable du dernier DFF et sortira la même valeur qu'il aura reçu en entrée, venant d'un capteur hall. Cette sortie (OUTPUT) sera la sortie hall_filtre destinée à aller dans le collecteur numérique.

3. Code mbed

Nous avons donc dû programmer le LPC1768 afin de lire les valeurs de poignée, température, tension et batterie.

```
#include "EthernetInterface.h"
#include <cstdio>
#include <iostream>
#include <string>
#include <fstream>
#include <stdlib.h>
#include <string.h>
#include "mbed.h"
//#include "mbed2/268/InterruptIn.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 PMIN
#define PMAX
```

Premièrement, nous avons dû inclure des bibliothèques.

```
Bloc_IO MyPLD(p25,p26,p5,p6,p7,p8,p9,p10,p23,p24);// instantiate object needed to communicate with PLD
AnalogIn poi(p17);
AnalogIn Vtemp(p19);
AnalogIn Vbat(p18);
AnalogIn Vcourant(p20);
// valid pmw mbed pin
Serial pc(USBTX, USBRX); // tx, rx
// Top_Hall Pin
```

Ensuite nous avons défini les entrées.

3.1 La calibration

Comme dispositif de commande qui est utilisé pour contrôler la vitesse de rotation du moteur, nous avons la poignée de gaz. Elle est reliée à un capteur qui mesure la position de la poignée. Ce capteur envoie ensuite une sortie analogique représentant la vitesse souhaitée.

Le microcontrôleur lit cette valeur analogique et la convertit en un signal électrique alors utilisé pour contrôler la tension d'alimentation du moteur donc la vitesse de rotation du moteur.

Mais pour effectuer cela nous devons écrire un code.

```
//************** calibration gaz function needed to record min_gaz and max_gaz value to persistent text file (min gaz=0.15, max gaz=0.71) *****
void calibration(void)
{
float min,max;
LocalFileSystem local("local");

printf("Mettre la poignee au repos pendant 5 secondes\n\r");
wait(3);
min=poi.read();
wait(2);
printf("Mettre la poignee au maximum pendant 5 secondes\n\r");
wait(3);
max=poi.read();
wait(2);
printf("La poignee a ete calibrée !\n\r");
```

Dans cet extrait de notre code nous mettons en place la calibration de la poignée de gaz. En effet chaque poignée donne des valeurs analogiques différentes, la calibration nous permet donc de nous adapter.

```
FILE *f;

f = fopen("/local/calib.txt","w");// adds content to the file
    if(f!=NULL)//si le fichier n'est pas encore crée, il en crée un automatiquement
    {
        fprintf(f,"%.2f\n%.2f",min,max);//
        printf("write min and max in file\n\r");
        fclose(f);
    }
```

Pour enregistrer ces valeurs nous devons les mettre dans un fichier texte. Ce code nous permet d'écrire les valeurs dans le fichier calib.txt. Si le fichier n'existe pas, il en crée un automatiquement.

3.2 La vitesse

Une des fonctionnalités logiques du micro-contrôleur vis-à-vis de la roue est de déterminer sa vitesse. Pour ce faire, nous détectons des balises disposées sur la roue grâce à nos capteurs. Nous avons compté 60 balises environ (chaque essais visant à déterminer le nombre n'étant pas le même).
Nous avons obtenu le calcul suivant dans le code :

```
    vit =(cpt*2*3.14)/60;//60 nbr de compteur en 1 tour
```

3.3 La température

Nous utilisons pour la température un capteur LM235A avec une pente de 10mV/C°. Voici un extrait de la datasheet :

Temperature Accuracy (Note 1)

LM135/LM235, LM135A/LM235A

Parameter	Conditions	LM135A/LM235A			LM135/LM235			Units
		Min	Typ	Max	Min	Typ	Max	
Operating Output Voltage	T_C = 25°C, I_R = 1 mA	2.97	2.98	2.99	2.95	2.98	3.01	V
Uncalibrated Temperature Error	T_C = 25°C, I_R = 1 mA		0.5	1		1	3	°C
Uncalibrated Temperature Error	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		1.3	2.7		2	5	°C
Temperature Error with 25°C Calibration	$T_{MIN} \le T_C \le T_{MAX}$, I_R = 1 mA		0.3	1		0.5	1.5	°C
Calibrated Error at Extended Temperatures	$T_C = T_{MAX}$ (Intermittent)		2			2		°C
Non-Linearity	I_R = 1 mA		0.3	0.5		0.3	1	°C

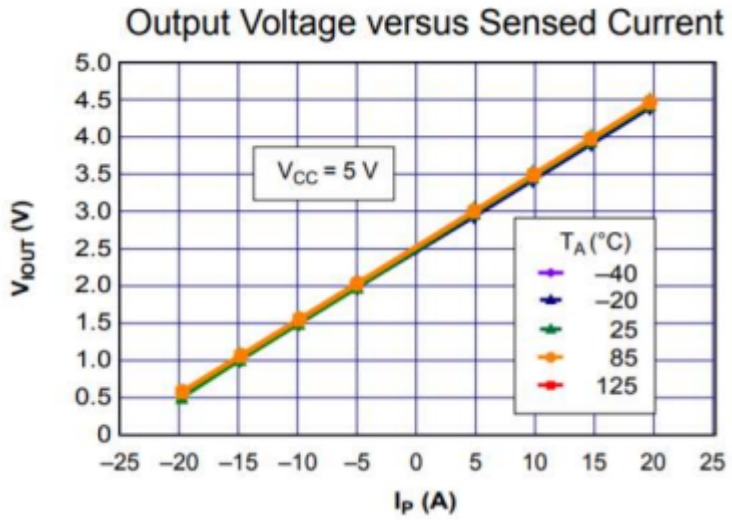
On en a déduit l'équation que nous avons inclu dans le code :

```
248         temperature=100*(Vtemp.read()*3.3-2.73);
```

Nous pouvons ainsi savoir si le système s'échauffe pour par exemple mettre en place une sécurité à l'avenir pour empêcher une surchauffe et donc une possible dégradation du matériel.

3.4 Le courant

Pour mesurer le courant, nous avons le capteur ACS712-20 qui est un capteur à effet Hall. Grâce à l'extrait de la datasheet :



On obtient l'équation I=10*(Vout)-2.5 pour mesurer le courant à partir de l'entrée analogique du Mbed.
Ainsi dans le code on aura :

```
250         courant=10*((Vcourant.read()-2.5);
```

Après sélection de la bonne fonction dans le menu sur TeraTerm, nous pouvons afficher toutes ces valeurs en continu.

3.5 Le menu sur TeraTerm

Nous avons fait en sorte que l'on puisse choisir la fonction voulue depuis TeraTerm sans devoir téléverser un nouveau code, nous avons donc utilisé la fonction " switch .. case " pour celà. Voici les lignes de codes correspondantes au menu :

```
252         switch(choix_fonction)
253         {
254             case 1 :
255             fonction_principale(var,gaz);
256             fonction_de_mesure_vitesse(1);
257             break;
258 
259             case 2 :
260             calibration();
261             goto DEBUT;
262             break;
263 
264             case 3 :
265             test_fichier();
266             wait(1);
267             goto SORTIE;
268             break;
269 
270             case 4 :
271             var=((gaz-0.158)*1.8116)*255;
272             MyPLD.write(var);
273             //controle vitesse
274             //Raz.attach(&raz,0.1);
275             wait(0.1);
276             vit = (cpt*2*3.14);
277             printf("Vitesse : %d\n\r",vit);//vitesse mesure
278             printf("Compteur : %d\n\r",cpt);//comptage impulsion
279             cpt=0;
280             break;
281 
```

Case 1: Effectue la demande principale qui est de modifier la vitesse à l'aide de la poignée.

Case 2 :Permet de calibrer la poignée

Case 3 : Effectue un test pour voir si le fichier servant à la calibration de la poignée en case 2 fonctionne

Case 4 : Il s'agit d'une fonction intermédiaire qui nous affichait la vitesse et le compteur.

```
282                 case 5 :
283                 fonction_principale(var,gaz);
284                 fonction_de_mesure_vitesse(0);
285                 if(petit_compteur==0)
286                 {
287                 printf("Affichage des temperatures batterie, de la tension, et du courant\n\r");
288                 wait(2);
289                 petit_compteur=1;
290                 }
291                 printf("Vitesse roue : %d\tTemperature :%f 'C\tTension batterie :%f V\tCourant batterie :%f A
292                 break;
293 
294             }//end switch
295         }//end while
296     SORTIE :
297     printf("FIN");
298 
299     } // end main
```

Case 5 : Fonction qui affiche tous les paramètres tels la température, la tension de la batterie ou le courant de la batterie.

4. Conclusion

Pour conclure, après avoir compris le fonctionnement de chaque élément, nous avons réussi à atteindre les objectifs principaux. La poignée est fonctionnelle, le fpga est correctement programmé en schéma bloc VHDL.

Après l'étude des différents capteurs nous avons pu programmer le microcontrôleur Mbed afin d'afficher les informations relatives au système (vitesse, tension etc…)

Ce projet nous a permis de voir les aspects électroniques, physiques et informatiques d'un tel dispositif.