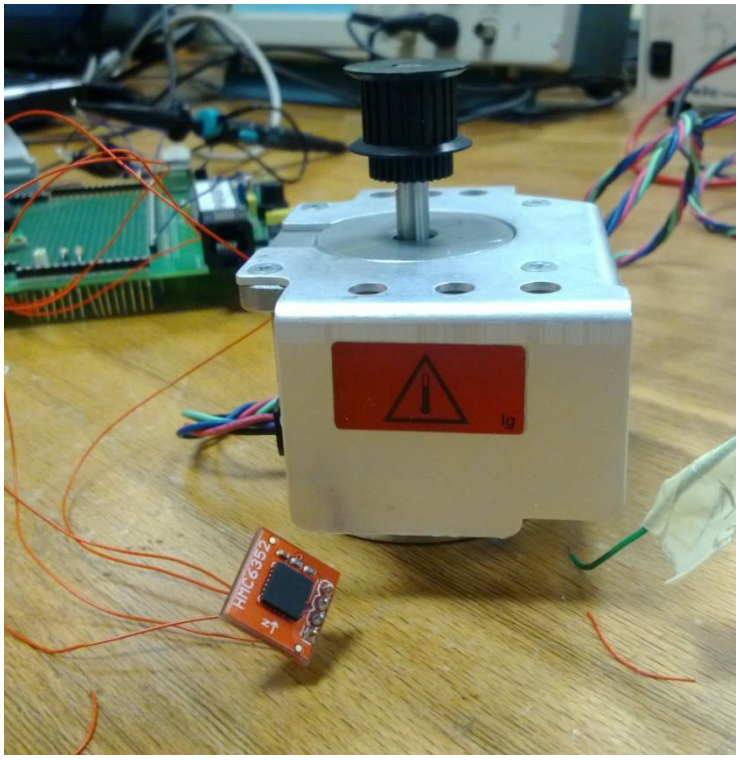
COMMANDE D'UN MOTEUR PAS A PAS AVEC UN HACHEUR



25/06/2013

Projet de Synthèse

Il s'agit ici d'orienter l'axe du moteur pas à pas dans une direction voulue par rapport au Nord magnétique.

POLYTECH CLERMONT-FERRAND
Relevons les défis de demain

NICOLAS MENDES
BACHIR MEGHNINE

LUDOVIC REYNOUARD
YOANN DUTEL

REMERCIEMENTS

Nous remercions Mr James et Mr Laffont pour leur aide et les conseils qu'ils nous ont donné tout au long du projet durant les horaires encadrées et en dehors. Nous tenons également à remercier Mr Pasquier pour ses divers conseils qu'il nous a donnés durant la réalisation et la réflexion du hacheur ainsi que Mr Landrault pour ses conseils sur les transistors et les optocoupleurs.

Enfin nous remercions le technicien Mr Sanchez pour nous avoir mis à disposition de manière très régulière la salle de projet ainsi que pour le matériel prêté pour réaliser nos différents tests.

INTRODUCTION

Dans le cadre du projet de synthèse nous avons eu comme sujet :

« Commande d'un moteur pas à pas avec hacheur ». Un hacheur est un dispositif de l'électronique de puissance permettant de convertir une tension continu en une autre tension continu. Il peut y avoir intervention de un ou plusieurs interrupteurs commandés. Dans notre cas, on aura affaire à un hacheur dévolteur, c'est-à-dire qu'on diminuera la tension d'entrée à l'aide d'un circuit de puissance adapté qui sera développé plus tard dans ce rapport. Le moteur fournit, par Mr James, était vraisemblablement un ancien moteur d'imprimante pris à l'IUT voisin. Nous avons réalisé l'ensemble de nos programmes sur MPLAB, un logiciel fournit par le fabricant du microcontrôleur.

Comme indiqué par nos professeurs le but du TP de synthèse est de synthétiser l'ensemble des enseignements de la 1ère année du cycle Ingénieur de Génie Electrique. De plus cet exercice est une première confrontation au travail d'un ingénieur en entreprise.

SOMMAIRE

1. ETUDE PRELIMINAIRE

1.1. Cahier des charges

Le système à concevoir comporte une boussole numérique montée sur un moteur pas à pas.

A l'aide d'un clavier matricé et d'un afficheur, un opérateur choisira un angle par rapport au Nord magnétique et le moteur pas à pas positionnera automatiquement le capteur boussole numérique dans la bonne direction.

Pour réaliser ce système, nous devrons concevoir une interface de puissance isolée galvaniquement permettant de piloter les deux bobines du moteur pas à pas.

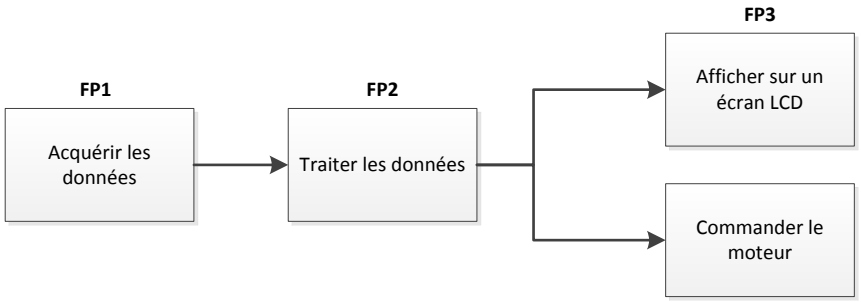
Le moteur pas à pas fourni par Mr James réalise 200 pas par tour et supporte une intensité maximale de 1 ampère. Chaque pas réalise un angle de 1.8°.

Des plaquettes sont mises à notre disposition afin de réaliser nos essais ainsi que des générateurs de tous types.

De plus une plaquette contenant entre autre le microcontrôleur nous a été donnée au début du projet. Mr Laffont a également mis à notre disposition divers logiciels et tutoriels permettant le fonctionnement du microcontrôleur sur nos machines et ce grâce à la forge (un site internet où nous devions notamment écrire l'avancement de notre projet).

1.2. Etude fonctionnelle

1.2.1. Schéma fonctionnel

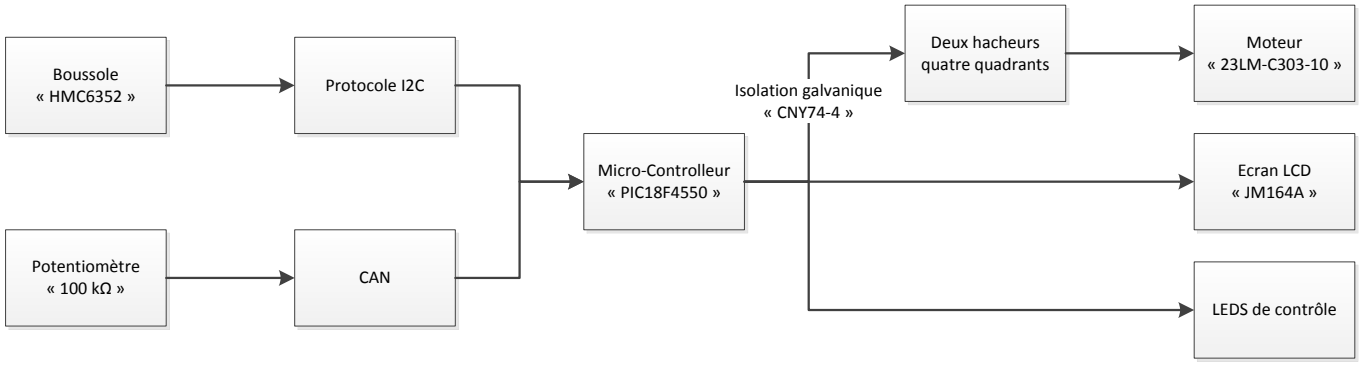


1.2.2. Découpage fonctionnel

- FP1.1 : Acquisition des données permettra au système de récupérer les données à corriger.
- FP1.2 : Consigne entrée par l'utilisateur offrira la possibilité à l'utilisateur d'entrer lui-même une valeur d'angle qui sera ensuite corrigée en fonction du nord.
- FP2 : Traitement des données permettra de traiter la donnée c'est-à-dire de définir la correction à appliquer aux données.
- FP3.1 : Affichage sur LCD permettra à l'utilisateur de connaitre la correction effectuée.
- FP3.2 : Commande du moteur agira directement sur le moteur et l'orientera vers le nord en fonction de la valeur de la donnée corrigée.

1.3. Etude structurelle

1.3.1. Schéma structurel



1.3.2. Choix des solutions structurelles

Pour la boussole, on utilisera la boussole HMC6352 fourni par Mr James. Une recherche de la documentation de cette boussole a été nécessaire. Cette dernière a été trouvée sur le site du fabricant Honeywell.

Pour le microcontrôleur comme donné dans le cahier des charges, on utilisera le microcontrôleur PIC18F4550 de MICROCHIP.

Le Hacheur sera composé de deux hacheurs 4 quadrants chacun (les raisons de ce choix seront expliquées ultérieurement).

Pour l'affichage on utilisera un afficheur LCD, fourni par le magasin de Polytech (que Francisco Sanchez nous a donné). Cet afficheur en question est un afficheur du fabricant SHENZEN JINGUA DISPLAY, et nous avons eu le modèle JM164A. Ici encore une recherche de la documentation a été nécessaire. Le fonctionnement et tous renseignements relatifs à cet afficheur seront développés plus tard.

Pour le clavier, on n'utilisera pas un clavier matricé comme défini dans le cahier des charges, en effet Mr James nous ayant informé de l'indisponibilité de ce dernier, nous avons donc remplacé le clavier par un potentiomètre et un convertisseur analogique numérique (CAN) .Le potentiomètre sera un potentiomètre classique disponible dans le magasin et pour le CAN on utilisera le CAN intégré au microcontrôleur.

2. HACHEUR

2.1. Réalisation du hacheur

La première partie de la réalisation du hacheur consistait à comprendre et découvrir le fonctionnement d'un moteur pas à pas, pour ce faire nous avons testé le moteur en question en alimentant chaque bobine de ce dernier.

En effet, ce moteur possède 4 fils visibles, nous avons donc dû au préalable déterminer quels fils étaient liés ensemble. Pour cela nous avons utilisé un multimètre sur position mesure d'inductance. On a alors déduit que les fils rouge et bleu étaient reliés à la même bobine d'inductance 5.21 mH, et que les fils noir et vert étaient reliés à la bobine de 5.26 mH.

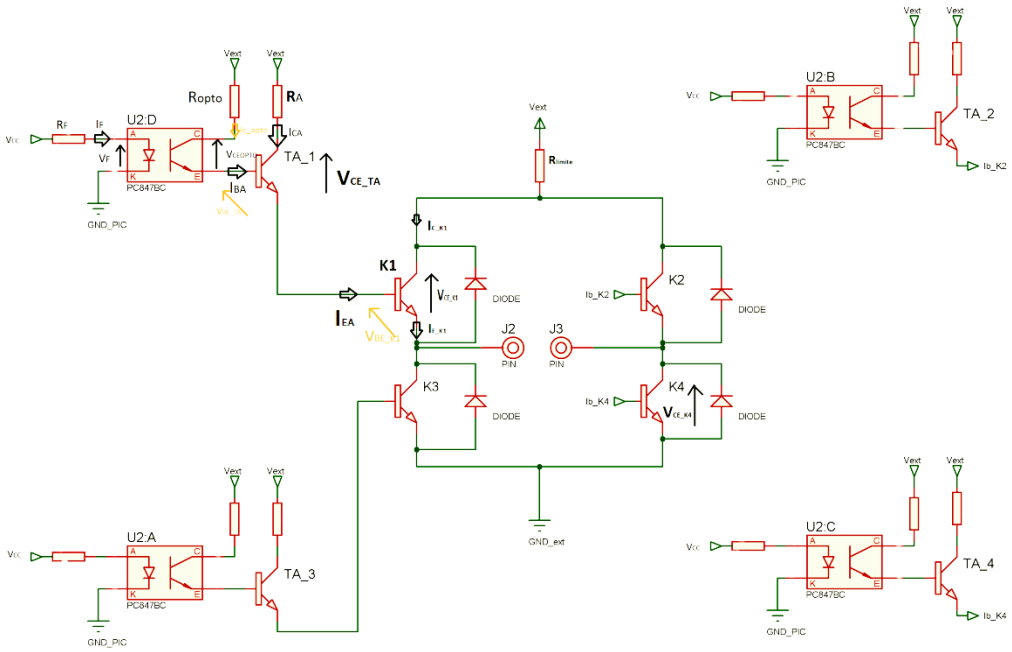
On a ensuite cherché à faire tourner le moteur d'un pas. Pour cela, nous avons utilisé une alimentation en source de courant, c'est-à-dire que nous avons fixé la tension à une valeur très basse (inférieur à 1 volt) et avons ensuite augmenté l'intensité progressivement jusqu'à voir le moteur effectuer un pas. Nous avons conclu qu'à partir de 0.8 ampères le moteur effectuait un pas.

A partir de plusieurs essais d'alimentation des bobines, on a mis en évidence le fait que la rotation du moteur s'effectue en alimentant les bobines l'une après l'autre. De plus, pour faire des pas dans le même sens, il faut veiller à inverser les polarités des bobines l'une après l'autre comme suit :

Ordre	Polarités	
	+	-
1	Vert	Noir
2	Bleu	Rouge
3	Noir	Vert
4	Rouge	Bleu

On en a donc déduit qu'un hacheur 4 cadrans était nécessaire.

On a d'abord réalisé le hacheur sans prendre en compte la partie commande.



NB: Les caractéristiques de chaque composant seront développées dans la partie 2, de plus les différents courants, résistances, tensions seront visibles sur l'illustration 1 représentant un hacheur 4 quadrants.

On a établi le calcul suivant pour trouver la valeur de R_{limite} :

$$V_{ext} = (R_{limite} \times I_{C\_K1}) + V_{CE(sat)}^{K1} + V_{CE}^{K4}$$

$$I_{E\_K1} = I_{EA} + I_{C\_K1}$$

$$I_{C\_K1} = I_{E\_K1} - I_{EA} = 0.8 - 0.1 = 0.7\ A$$

$$R_{limite} = \frac{V_{ext} - V_{CE(sat)}^{K1} - V_{CE(sat)}^{K4}}{I_{C\_K1}}$$

$$V_{CE}^{Kn} = 0.6\ V$$

$$\mathbf{AN:}\ \ \mathbf{R_{limite}} = \frac{\mathbf{9-0.6-0.6}}{\mathbf{0.7}} = \mathbf{11.14\Omega}$$

A partir de cette formule, on se retrouve avec deux inconnues : R_{limite} et V_{ext}.

On a uniquement les deux inconnues qui sont des variables dépendantes l'une de l'autre, quand R_{limite} augmente, V_{ext} augmente également. On a donc décidé de fixer arbitrairement V_{ext} pour obtenir une valeur de résistance qui soit la plus petite possible. Ainsi on obtient une valeur de V_{ext} petite et on perd le moins d'énergie possible.

On a donc pour V_{ext} = 9 Volts, une valeur de R_{limite} égale à 11,14 ohms.

On passe à présent au courant nécessaire pour la commande de nos transistors. Ayant choisi un optocoupleur qui délivre au maximum un courant de 50 mA dans l'émetteur, on s'est donc confronté au problème de courant faible dans la base du transistor de commande (K_n sur l'illustration 1). Pour remédier à ce problème on a décidé d'amplifier ce courant à l'aide d'un transistor (T_A sur l'illustration 1).

On arrive donc au calcul de R_A pour fixer le courant du collecteur du transistor d'amplification à 50 mA pour ainsi avoir 100 ma dans le courant de l'émetteur du même transistor.

Calcul de R_A :

$$V_{ext} = (R_A \times I_C^{TA}) + V_{CE}^{TA} + V_{BE(sat)}^{K1} + V_{CE(sat)}^{K4}$$

$$I_{EA} = I_{CA} + I_{BA}$$

$$I_{CA} = I_{EA} - I_{BA}$$

Pour saturer K1 il faut 0.1A dans I_{EA}

$$I_{CA} = 0.1 - 0.05 = 0.05\ A$$

$$R_A = \frac{V_{ext} - V_{CE(sat)}^{TA} - V_{BE(sat)}^{K1} - V_{CE(sat)}^{K4}}{I_{CA}}$$

$$V_{CE(sat)}^{TA} = 1.2\ V$$

$$V_{BE(sat)}^{TA} = 1.3\ V$$

$$\mathbf{A.N:}\ \ \mathbf{R_A} = \frac{\mathbf{9-1.2-1.3-0.6}}{\mathbf{0.05}} = \mathbf{118\ \Omega}$$

On arrive donc au réglage du courant dans l'optocoupleur, pour cela on utilise une résistance que l'on calcule comme suit :

Calcul de R_{opto}

$$V_{ext} = R_{opto} \times I_{c\_opto} + V_{CE(sat)}^{opto} + V_{BE}^{TA} + V_{BE(sat)}^{K1} + V_{CE(sat)}^{K4}$$

$$R_{opto} = \frac{V_{ext} - V_{CE(sat)}^{opto} - V_{BE(sat)}^{TA} - V_{BE}^{K1} - V_{CE(sat)}^{K4}}{I_{opto}}$$

$$\mathbf{I_{c\_opto\_max} = 50mA}$$

$$\text{On prend donc } I_{c\_opto} = 50\text{mA}$$

$$V_{CE(sat)}^{opto} = 0.3\text{V}$$

$$V_{BE(sat)}^{TA} = 1.2\text{V}$$

$$V_{BE(sat)}^{K4} = 0.6\text{V}$$

$$V_{CE(sat)}^{K1} = 1.3\text{V}$$

$$\mathbf{A.N:\ R_{opto} = \frac{9-0.3-1.2-1.3-0.6}{0.05} = 112\ \Omega}$$

Maintenant que la partie courant fort de l'optocoupleur a été définie, on va s'intéresser à la partie microcontrôleur. Au niveau de la diode de l'optocoupleur, il faut une tension de 1.6 volts, pour régler cette tension on utilise une résistance R_F que l'on a calculé comme ci-dessous :

Calcul de R_F :

$$V_{cc} = R_F \times I_F + V_F$$

$$I_{Fmax} = 50\text{ mA}$$

$$\text{On prend : } I_F = 20\text{ mA}$$

$$R_F = \frac{V_{cc} - V_F}{I_F}$$

$$\mathbf{AN: R_F = \frac{5-1.6}{0.02} = 170\ \Omega}$$

Comme on peut le voir il y a des diodes présentes au niveau des transistors de commandes (K_n sur l'illustration 1), celles-ci permettent de dissiper l'énergie présente dans l'une des bobines lorsqu'aucun des interrupteurs (K_n) n'est passant et que l'on passe à un autre hacheur.

A partir de nos calculs, nous sommes allés voir ce qu'il y avait à disposition dans le magasin afin de choisir nos composants.

Les hacheurs 4 quadrants réalisés sous le logiciel de CAO Proteus sont visibles sur l'annexe 4.

2.2. Choix des composants

Pour les transistors K_n, nous avons cherché des transistors qui avaient un courant de saturation proche de 1 Ampère. On a donc choisi les transistors SGS Thomson BD237. A saturation ce transistor a les caractéristiques suivantes :

$$I_{csat} = 1\ A\ ///\ I_{bsat} = 0.1\ A\ ///\ V_{cesat} = 0.6\ V\ ///\ V_{besat} = 1.3\ V$$

Le choix de R_{limite} a été défini par la puissance dissipée qui vaut :

- $P = R * I_2 = 11.14 * 0.72 = 5.46\ W$

On a donc dû prendre une résistance de puissance 10 Watts de 12 ohms.

Concernant le choix du transistor d'amplification, il nous fallait en sortie de l'émetteur un courant de 100 mA, nous avons donc choisi le transistor de PHILIPS semi-conductors 2N2219A avec les caractéristiques suivantes :

$I_{esat} = 150$ mA et $I_{bsat} = 15$ mA pour $V_{cesat} = 0.3$ V et $V_{besat} = 1.2$ V

On a choisi l'optocoupleur afin de faire une isolation galvanique, qui consiste à séparer la partie commande de la partie puissance, par séparation on entend aucune connexions électrique entre les deux parties. Ici on a choisi l'optocoupleur CNY74-4 du fabricant TEMIC qui présente les caractéristiques suivantes :

- Une tension aux bornes de la diode de 1.6 Volts
- Un courant dans le collecteur du phototransistor de 50 mA.

Concernant la diode, on a choisi une diode laissant passer un courant de 1 Ampère. Dans le magasin on a trouvé des diodes laissant passer un courant maximum de 1 Ampère. Cependant étant donné que nous étions proche de notre courant circulant dans la bobine, nous avons opté pour une diode laissant passer un courant maximum de 3 ampères afin d'avoir la certitude que tout le courant circulant dans la bobine puisse circuler dans la diode.

Pour les autres résistances, on a une puissance dissipée de :

- $P = R_a * I_{ca2} = 118 * 0.052 = 0.295\ W$ pour R_a
- $P = R_{opto} * I_{c\_opto2} = 0.28\ W$ pour R_{opto}

Ces puissances étant proches de 0.25 Watts, en accord avec les professeurs, nous avons décidé de prendre des résistances de 0.25 Watts (ces résistances chaufferont durant l'utilisation du moteur sans aucun risque de perte du composant).

Concernant la résistance R_F, on a une puissance à ses bornes de :

- $P = R_F * I_{F2} = 170 * 0.022 = 0.068\ W$

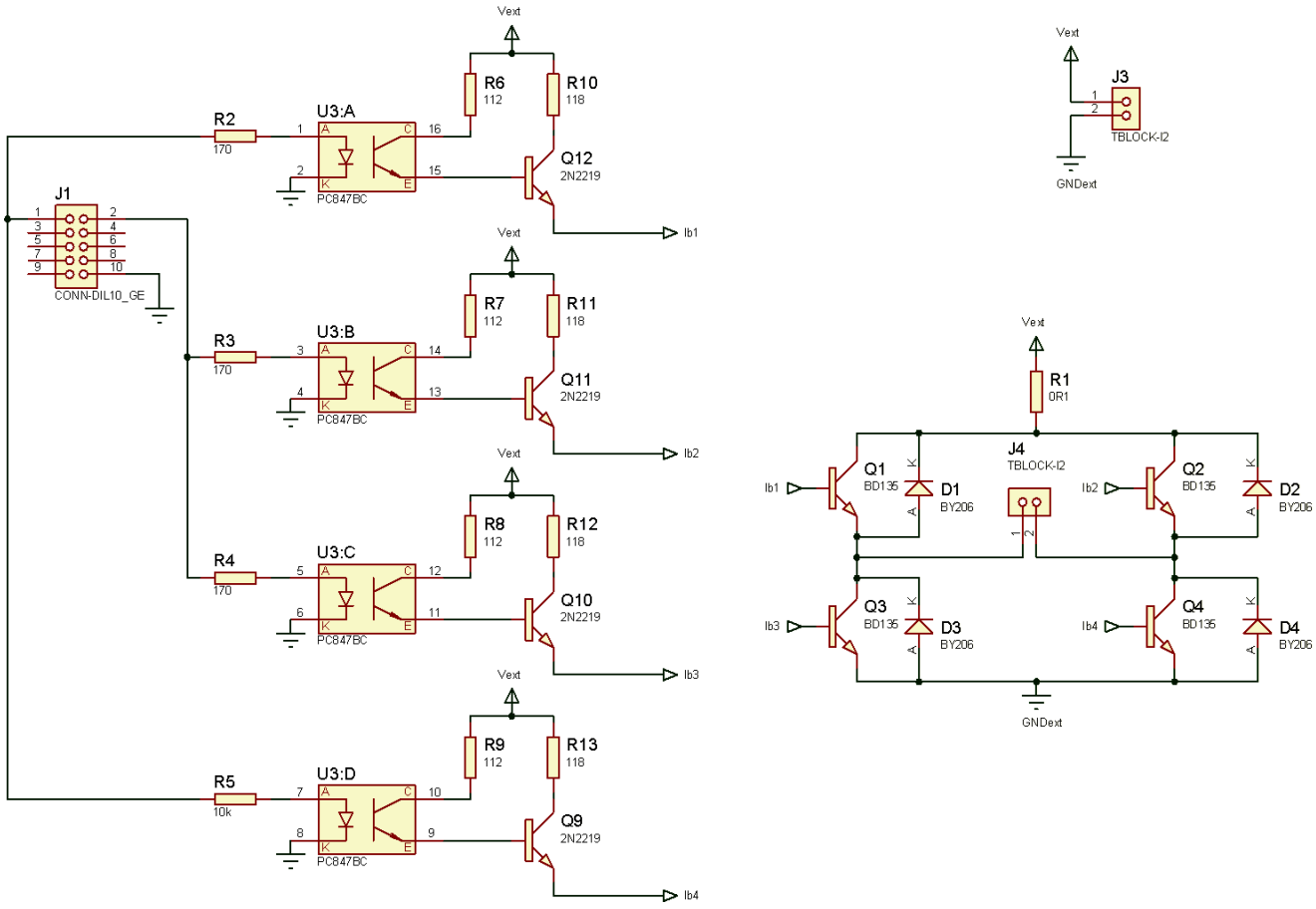
Donc pour cette dernière une résistance de 0.25 Watts est largement suffisante.

On ajoute que la tension V_{ext} continue de 9V sera délivrée par une alimentation continue fourni par Mr Sanchez.

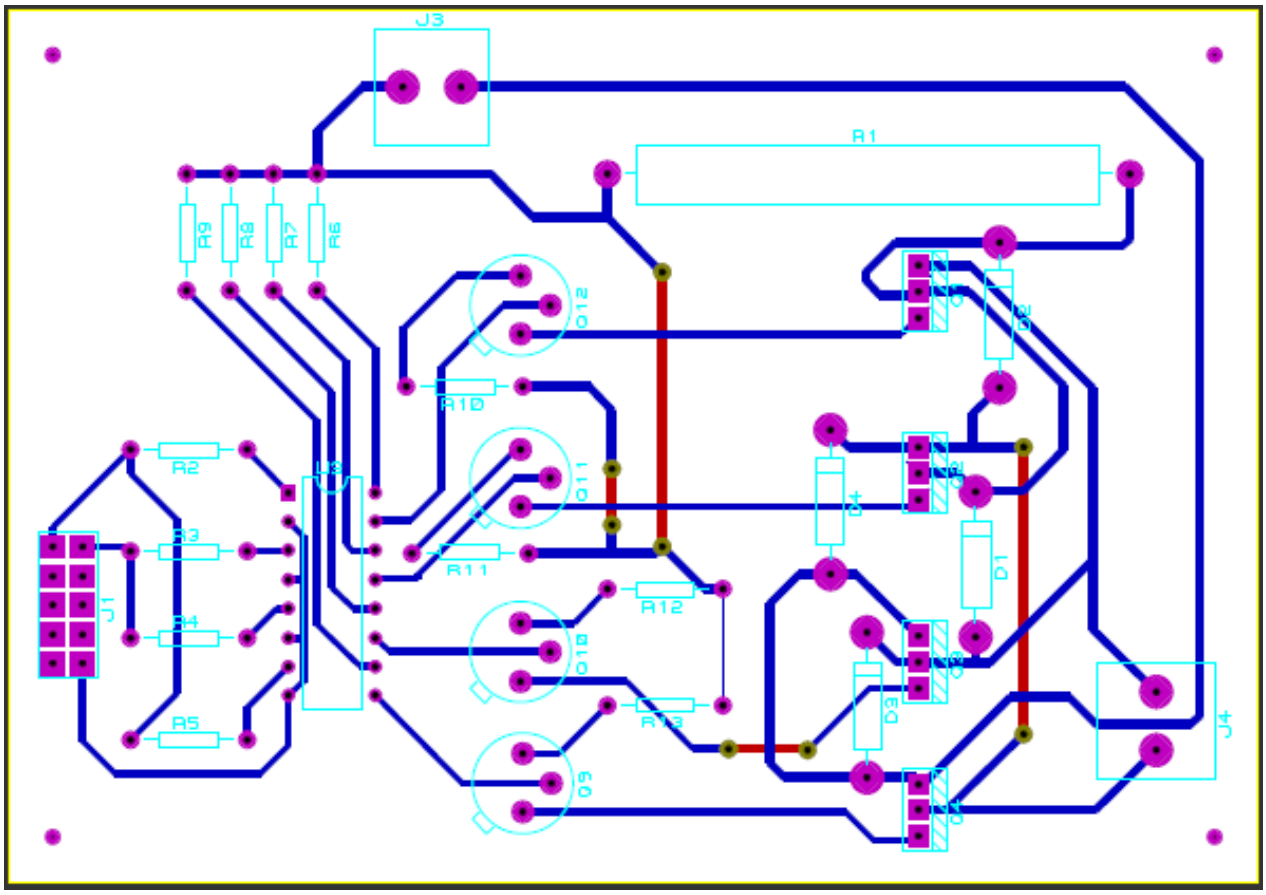
2.3. Routage

Afin d'apporter un complément à ce projet nous avons décidé de réaliser le routage de la carte de commande du moteur, celle-ci contenant l'isolation galvanique, l'étage d'amplification et le hacheur. Pour cela nous avons décidé de réaliser deux cartes différentes, une pour chaque bobine du moteur. Il aurait donc suffi de réaliser le routage une fois, puis on aurait imprimé deux fois la même carte.

Pour cela nous avons repris le schéma de commande d'une des bobines réalisé durant les TP de CAO. Il a tout de même fallu faire attention au choix des composants à mettre sur le schéma afin que leurs empreintes soient en accord avec nos composants, c'est-à-dire qu'ils aient la même taille mais aussi que les trous soit percés et placés correctement. Voici donc le schéma réalisé sous le logiciel Proteus :



Ensuite on a conçu le routage de la carte sous Ares toujours à l'aide de Proteus. On a pu remarquer que les pistes avaient une largeur par défaut de 10th, ce qui est normalement suffisant pour notre courant qui est au maximum de 1A. En effet après le calcul on a trouvé une largeur de 2.6th pour ce courant. Mais après plusieurs conseils nous avons choisi une largeur de piste de 30th pour la partie puissance et de 20th pour la partie commande. Voici donc le routage réalisé :



On peut remarquer que nous avons dû créer le connecteur J1 avec dix broches, il en existait déjà un dans la librairie de Proteus mais cela ne nous convenait pas car les broches n'était pas placé au bon endroit. Nous avons aussi dû augmenter le diamètre des pastilles pour les diodes car celle de Proteus n'était pas assez grosses, donc nous avons créé un nouveau package en plus de celui qui existait déjà dans la librairie Proteus.

Pour des raisons de manque de temps, mais aussi le fait de ne pas avoir de composants de secours en cas de problèmes. Nous avons décidé de ne pas réaliser ces cartes.

3. PROGRAMMATION

3.1. Les composants utilisés

3.1.1. La boussole « HMC6352 »

3.1.1.1. FONCTIONNEMENT GENERAL

La boussole possède deux adresses d'esclave :

- L'adresse 0x42 pour écrire une commande dans la RAM ou dans l'EEPROM
- L'adresse 0x43 pour lire des données de la RAM ou de l'EEPROM

La boussole possède deux modes de fonctionnement. On peut :

- lui dire de calculer l'angle à une certaine fréquence et lors de la réception des données on obtiendra la dernière valeur calculée et stockée dans la RAM
- ou on peut lui commander de calculer l'angle à un instant t et d'acquérir cette valeur calculée (beaucoup plus long car il faut attendre que le calcul soit effectué avant d'acquérir).

Voici les différentes adresses d'écritures et de lectures de la boussole :

Command Byte ASCII (hex)	Argument 1 Byte (Binary)	Argument 2 Byte (Binary)	Response 1 Byte (Binary)	Response 2 Byte (Binary)	Description
w (77)	EEPROM Address	Data			Write to EEPROM
r (72)	EEPROM Address		Data		Read from EEPROM
G (47)	RAM Address	Data			Write to RAM Register
g (67)	RAM Address		Data		Read from RAM Register
S (53)					Enter Sleep Mode (Sleep)
W (57)					Exit Sleep Mode (Wakeup)
O (4F)					Update Bridge Offsets (S/R Now)
C (43)					Enter User Calibration Mode
E (45)					Exit User Calibration Mode
L (4C)					Save Op Mode to EEPROM
A (41)			MSB Data	LSB Data	Get Data. Compensate and Calculate New Heading

3.1.1.2. LIAISON I2C

Pour la liaison I2C, on utilisera une émulation de cette dernière et non le périphérique intégré du microcontrôleur. Pour ce faire nous avons donc dû écrire plusieurs sous-programmes permettant de réaliser les opérations qu'aurait fait le périphérique. Ceci a notamment été commencé en TD.

Tout d'abord entre chaque envoi et réception, il faut envoyer des conditions de start et de stop pour signifier une communication sur le bus I2C.

Pour envoyer une commande, on configure la broche SDA en sortie et on envoie bit à bit sur un signal d'horloge les données. Puis on configure la broche SDA en entrée pour réceptionner le 9ème bit qui est le bit confirmant la bonne réception de la commande (ou de la donnée). Cet « acknowledge » est à 0 pour une bonne commande et à 1 pour une mauvaise.

Pour réceptionner une donnée, on fonctionne en deux parties car la donnée reçue est sur 16 bits. Tout d'abord, on envoie l'adresse de l'esclave signifiant la lecture puis passe en mode réception. En réception on configure le port SDA en entrée et on lit sa valeur à chaque coup d'horloge de SCL. Pour la réception de la première partie (MSB) il faut que le maître envoie un « acknowledge » et pour la deuxième partie (LSB) il faut que le maître envoie un « non-acknowledge ».

Voir les algorithmes pour les différentes fonctions d'initialisations et de transferts au paragraphe 3.2.3.

3.1.2. L'écran LCD « JM164A »

Cet écran LCD possède le même fonctionnement que celui étudié lors des TP de SP. Ainsi, nous avons choisi de communiquer avec lui sur 4 bits pour économiser les ports du microcontrôleur.

Nous avons donc connecté :

- RS à la broche RD0
- E à la broche RD1
- R/W à la masse car nous sommes toujours en écriture donc au potentielle 0
- Le bus de données sur les broches RD5…RD2

Pour envoyer une commande, il faut mettre la broche RS à 0. Puis (comme nous sommes en fonctionnement 4 bits) écrire la partie haute de la donnée sur le port de donnée et mettre E à 1 pour valider l'envoi et faire la même chose pour la partie basse.

Pour envoyer un caractère, l'envoi s'exécute de la même façon (en deux parties) mais cette fois il faut mettre la broche RS à 1.

Pour la question d'initialisation de cet afficheur (choix du mode de fonctionnement, présence du curseur,…) se référer aux algorithmes présents au paragraphe 3.2.4.

3.1.3. Le clavier

Comme dit précédemment, aucun clavier matricé n'étant disponible, il fut remplacé par un CAN (Convertisseur Analogique Numérique) et un potentiomètre. L'idée étant que lorsque l'on augmente ou diminue le potentiomètre, la boussole se décale du nord en fonction du sens tourné du potentiomètre (donc en fonction de l'augmentation ou diminution de la tension).

Le potentiomètre est un potentiomètre classique pris dans le magasin. Le CAN est celui intégré au microcontrôleur que nous avons programmé à l'aide de la documentation (dont les registres configurés ont été mis en annexe paragraphe 2.3).

Il nous a fallu configurer trois registres pour le CAN:ADCON0, ADCON1 et ADCON2.

Pour ADCON0, on a mis le 1er bit à 0 afin d'activer le CAN, les 5 bits suivants sont à 0 afin de sélectionner le canal AN0 du microcontrôleur (qui est sur le port RA0) et qu'aucune conversion n'est en cours (bit 2).Les bits 7 et 8 sont à 0 comme indiqué dans la documentation.

Pour ADCON1, la combinaison 1110 sur les bits de poids faible permet de mettra AN0 en analogique est les autres AN en numériques (donc de les laisser par défaut) .On met les 2 bits suivants à 0 afin d'avoir comme tension de référence VCC (bit 5) et la masse (bit 6).Les deux derniers bits étant à 0 comme indiqué dans la documentation (unimplement).

Pour ADCON2, on divise par 32 la fréquence du microcontrôleur, car sinon d'après nos test, lorsque le pré-diviseur diminue et que l'on tourne le potentiomètre, les changements sont trop rapide. Lorsqu'on augmente le pré diviseur et que l'on augmente le potentiomètre les changements sont trop lents. Le pré-diviseur 32 est dans notre cas le juste milieu. On met donc la combinaison 010 sur les bits de poids faible du registre. On a besoin de T_{ad} (>0.7 us d'après le constructeur) pour convertir un bit, on aura donc besoin de 8 T_{ad} pour convertir 8 bits (on veut faire une conversion de 8 bits) donc on aura la combinaison 100 sur les bits 4 à 6.Le bit numéro 7 est à 0 car indiqué dans la documentation et le dernier bit (le 8) est à 0 car on justifie à gauche, c'est à dire que le résultat sur 8 bits sera contenu sur la partie haute de adres (le registre contenant la conversion).

Les algorithmes relatifs au fonctionnement de ce potentiomètre se trouvent au paragraphe 3.2.5.

3.2. Les algorithmes

3.2.1. Attribution des ports (setup.h)

LED0	correspond à	LATEbits.LATE0
LED1	correspond à	LATEbits.LATE1
LED2	correspond à	LATEbits.LATE2
T1	correspond à	LATBbits.LATB1
T2	correspond à	LATBbits.LATB2
T3	correspond à	LATBbits.LATB3
T4	correspond à	LATBbits.LATB4
RS	correspond à	LATDbits.LATD0
E	correspond à	LATDbits.LATD1
DATA	correspond à	LATD
SCL	correspond à	LATCbits.LATC0
SDA	correspond à	LATCbits.LATC1
SDA_return	correspond à	PORTCbits.RC1
SDA_IO	correspond à	TRISCbits.TRISC1

Variables Globales
    constante vitesse_moteur <- 40

3.2.2. Fichier main.c

Variables Globales
    tableau de caracteres char_boussole[] <- {"-DIREC BOUSSOLE-"}
    tableau de caracteres char_choix[] <- {"DIRECTION VOULUE"}
    tableau de caracteres char_calibrage[] <- {"CALIBRAGE EN COURS"}

3.2.2.1. INIT(ENTIER MODE)

Initialisation de tous les modules du programme.

```
Début
    Initialiser les ports
    Initialiser la boussole
    Initialiser l'écran
    Initialiser le CAN
    Initialiser le moteur pas à pas

    Si (mode)
        Calibrer la boussole
        Afficher "char_calibrage" sur l'écran
    Fin si

    Afficher "char_boussole" sur l'écran
Afficher "char_choix" sur l'écran

    Allumer la LED0
Fin
```

3.2.2.2. MAIN()

Fonction principale du programme.

```
Début
    Initialisation de tout les composants avec init(0)
    Tant que(1)
        Obtenir l'angle de la boussole
        Obtenir l'angle du potentiomètre
        Orienter le moteur sur l'angle voulu du potentiomètre
    Fin tant que
Fin
```

3.2.3. Fichier setup.c

3.2.3.1. TEMP_MS(ENTIER DUREE)

Cette fonction a pour but de créer une temporisation variable de l'ordre du ms. Elle est utilisée dans toutes les parties du programme. Toutefois on ne peut pas dépasser 348 ms.

```
Variables
    Entier buffer
Début
    duree=0XFFFF-(188*duree)
    buffer <- duree
    Décalage 8 fois à droite de buffer
    Configurer le registre T0CON //16 bits, pré diviseur à 64
    Mettre buffer dans le registre TMR0H
    Mettre duree dans le registre TMR0L
    Mettre à 0 le flag du timer
    Démarrer le timer
    Attendre débordement
    Stopper le timer
Fin
```

3.2.3.2. TEMP_US(ENTIER DUREE)

Cette fonction a pour but de créer une temporisation variable de l'ordre du µs. Elle est utilisée dans toutes les parties du programme. Toutefois on ne peut dépasser 50 µs.

```
Début
    duree=0xFF-5*duree
    Configurer le registre T0CON // pré diviseur à 64, 8bits
    Mettre duree dans TMR0L
    Initialiser à 0 le flag du timer
    Démarrer le timer
    Attendre le débordement
    Arrêter le timer
Fin
```

3.2.3.3. INIT_PORT()

Cette fonction sert à initialiser tous les ports du microcontrôleur. Si ils sont en sorties, en entrées, ainsi que leurs valeurs initiales.

```
Début
        Mettre le port D en sortie              // afficheur LCD en sortie
        Initialiser à 0 le port D
        Mettre le port B en sortie              // hacheur en sortie
        Initialiser à 0 le port B
        Mettre RC0 en sortie                    // SCL en sortie
        Mettre RC1 en sortie                    // SDA en sortie
        Mettre le port E en sortie              // LED en sortie
        Initialiser à 0 le port E
        Mettre RA0 en entrée                    // potentiomètre en entrée
Fin
```

3.2.3.4. AFFICHER_DONNEES(ENTIER POSITION_CURSEUR, ENTIER ANGLE)

Cette fonction sert à transformer un nombre décimal de trois chiffres (angle) en caractères ascii dans un tableau. Puis afficher ce nombre ascii sur l'afficheur.

```
Variables
        Tableau de caractère st[] <- {"    deg NORD"}
        Entier buffer

Début
        buffer <- angle/100;                    // exemple 320° : on récupère le 3
        st[0] <- buffer+'0';                    // on le transforme en '3'
        angle <- angle - buffer*100             // on soustrait 300
        buffer <- angle/10                      // on récupère le 2
        st[1] <- buffer+'0'                     // on le transforme en '2'

        angle <- angle - buffer*10              // on soustrait 20 et on récupère le 0
        st[2] <- angle+'0'                      // on transforme en '0'
        Afficher st sur l'écran
Fin
```

3.2.4. Fichier boussole.c

3.2.4.1. TPO_I2C()

Cette fonction sert à définir la fréquence de fonctionnement de l'i2c. Ici on est régler à .....

```
Début
        Mettre les bits du registre T2CON
        Mettre le flag du registre PIR1 à 0
        Mettre la valeur 250 dans le registre PR2    //temporisation de 5us
        Mettre la valeur 0 dans le registre TMR2     //on compte de PR2 à TMR2
        Démarrer le timer
        Attendre débordement
        Arrêter le timer
Fin
```

3.2.4.2. START()

Fonction réalisant la condition de start de l'i2c.

```
Debut
        SDA <- 1
        SCL <- 1
        Attendre 5us
        SDA <- 0
        Attendre 5us
        SCL<- 0
        Attendre 5us
Fin
```

3.2.4.3. STOP()

Fonction réalisant la condition de stop de l'i2c.

```
Debut:
        SDA <- 0
        SCL <- 0
        Attendre 5us
        SDA <- 1
        Attendre 5us
        SCL <- 1
        Attendre 5us
Fin
```

3.2.4.4. ENTIER SEND_I2C(ENTIER MOT)

Cette fonction sert à envoyer un mot de 8 bits à la boussole qui nous renvoie un ACK ou un NACK.

```
Variables
        Entier i

Début
        Pour i de 0 à 7                 //on change la valeur de chaque bit
                SDA <- bit de rang i
                Attendre 5us
                SCL <- 1
                Attendre 5us
                SCL <- 0
        Fin pour
        Configurer SDA en entrée
        Attendre 5us
        SCL <- 1                        // on génère le 9ème signal d'horloge
        Lire et stocker SDA
        Attendre 5us
        SCL <- 0
        Configurer SDA en sortie
        Attendre 5us
        Retourner la valeur stockée de SDA
Fin
```

3.2.4.5. ENTIER RECEPTION_I2C(ENTIER ACK)

Cette fonction sert à réceptionner un mot de 8 bits venant de la boussole et de lui répondre par un ACK ou un NACK (suivant le paramètre d'entrée).

```
Variables
        Entiers i, a=0x00
Début
        Configurer SDA en entrée
        Pour i de 0 à 8                     //on change la valeur de chaque bit
                Attendre 5us                //génération impulsion horloge
                SCL <- 1
                Attendre 5us
                a = a|SDA_return;
                Décalage à gauche de a      //on décale a car on reçoit du MSB vers le LSB
                SCL <- 0;
        Fin pour
        Attendre 5us
        Configurer SDA en sortie
        Si ack = 1
                SDA <- 0                    // test ACK envoyé par le maître
        Sinon
                SDA <- 1                    // test NACK envoyé par le maitre
        Fin si
        SCL <- 1                            //génération 9eme impulsion horloge
        Attendre 5us
        SCL <- 0
        SDA <- 0
        Attendre 5us
        Retourner a
Fin
```

3.2.4.6. CALIBRAGE()

Cette fonction sert à lancer un étalonnage de la boussole (ne sert pas à chaque démarrage cf init()).

```
Début
        Envoyer la commande de calibrage
        Attendre 30 s
        Envoyer la commande de fin de calibrage
        Attendre 15 ms
Fin
```

3.2.4.7. INIT_BOUSSOLE()

Cette fonction initialise la boussole. On la réveille puis on lui dit de calculer l'angle à une fréquence de 20Hz pour que l'on réceptionne la valeur la plus récente de l'angle lors de notre acquisition.

```
Début
        Envoyer la commande de sortie du mode veille
        Attendre 150 us
        Envoyer la commande d'écriture dans la RAM puis le mode de fonctionnement
        Attendre 150 us
        Envoyer la commande d'écriture dans la RAM puis la façon dont la boussole doit nous
renvoyer la donnée
        Attendre 150 us
Fin
```

3.2.4.8. ENTIER RETOUR_ANGLE()

Cette fonction réceptionne les données de la boussole et renvoi l'angle par rapport au Nord en degré.

```
Variables
        entiers MSB, LSB, angle
Début
        Réceptionner la partie haute (MSB) et la partie basse (LSB) de l'angle
        Décalage à droite de 8 de MSB
        angle <- (MSB|LSB)/20
        Afficher l'angle sur l'écran
        Retourner  angle
Fin
```

3.2.5. Fichier lcd_4bits.c

3.2.5.1. LCD_CAR(ENTIER CARACTERE)

Cette fonction a pour but d'envoyer un caractère de 8 bits sur l'écran LCD réceptionnant sur 4 bits.

```
Variable
        Entier buffer
Début
        buffer <- caractere
        buffer <<= 2
        caractere >>= 2
        Attendre 100 us
        RS <- 1                                                 // mode caractère
        E <- 0
        DATA <- (caractere&0b00111100)|(DATA&0b00000011)        // envoi MSB de DATA
        Attendre 5 us
        E <- 1                                                  // démarre l'envoi
        Attendre 5 us
        E <- 0                                                  // arrête l'envoi
        DATA <- (buffer&0b00111100)|(DATA&0b00000011)           // envoi LSB de DATA
        Attendre 5 us
        E <- 1
        Attendre 5 us
        E <- 0
        Attendre 5 ms
Fin
```

3.2.5.2. LCD_COM(ENTIER COMMANDE)

Cette fonction a pour but d'envoyer une commande de 8 bits sur l'écran LCD réceptionnant sur 4 bits.

```
Variables
        Entier buffer
Début
        buffer <- commande
        buffer <<= 2
        commande >>= 2
        Attendre 100 us
        RS <- 0                                                 // mode commande
        E <- 0
        DATA <- (commande&0b00111100)|(DATA&0b00000011)         // envoi MSB de DATA
        Attendre 5 us
        E <- 1                                                  // démarre l'envoi
        Attendre 5 us
        E <- 0                                                  // arrête l'envoi
        DATA <- (buffer&0b00111100)|(DATA&0b00000011)           // envoi LSB de DATA
        Attendre 5 us
        E <- 1
        Attendre 5 us
        E <- 0
        Attendre 5 ms
Fin
```

3.2.5.3. LCD_COM_INIT(ENTIER COMMANDE)

Cette fonction a pour but d'envoyer une commande de 4 bits sur l'écran LCD réceptionnant sur 4 bits.

```
Début
        commande >>= 2
        RS <- 0                                                 // mode commande
        E <- 0
        DATA <- (commande&0b00111100) | (DATA&0b00000011)
        Attendre 5 us
        E <- 1;                                                 // démarre l'envoi
        Attendre 5 us
        E <- 0                                                  // arrête l'envoi
Fin
```

3.2.5.4. INIT_LCD()

Cette fonction sert à initialiser l'afficheur. Par exemple le choix du mode de fonctionnement en 4 bits et non 8 bits, afficher le curseur, incrémenter ou décrémenter le registre après l'écriture d'un caractère,… La méthode est décrite sur la documentation du constructeur.

```
Début
        Attendre 30 ms
        lcd_com_init(0x30)      // 0x38 pour dire qu'on envoi sur 4bits
        Attendre 5 ms
        lcd_com_init(0x30)
        Attendre 100 us
        lcd_com_init(0x30)
        lcd_com_init(0x20)
        lcd_com(0x28)
        lcd_com(0x0C)           // 0x0C pour dire qu'on éteint l'écran et qu'on affiche pas le curseur
        lcd_com(0x06)           // pour décaler le curseur d'un cran vers la droite
        lcd_com(0x01)           // pour effacer l'écran
        Attendre 5 ms
Fin
```

3.2.5.5. LCD_STR(TABLEAU DE CARACTERE STR)

Fonction qui écrit une chaîne de caractère.

```
Variables
        Entier i <- 0
Début
        Tant que str[i] != '\0'                 // tant qu'on n'est pas à la fin de la chaine
                lcd_car(str[i])
                i++
        Fin tant que
Fin
```

3.2.6. Fichier can.c

3.2.6.1. INIT_CAN()

Cette fonction sert à initialiser le convertisseur analogique numérique.

```
Début
        ADCON0 <- 0b00000001
        ADCON1 <- 0b00001110
        ADCON2 <- 0b00100010
Fin
```

3.2.6.2. ENTIER CONVERTIR()

Cette fonction sert à covertir les valeur en sortie du potentiometre qui sont analogique en valeurs numérique de 0 à 255.

```
Début
        attente 5µs
        lancement conversion
        attente fin de conversion
        retourne ADRESH
Fin
```

3.2.6.3. ENTIER CONVERTIR_VALEUR_CAN_VERS_ANGLE()

Cette fonction sert à convertir la valeur de la conversion analogique numérique en degré.

```
Variables
        Entier angle, valeur_can
Début
        valeur_can <- convertir()
        angle <- valeur_can
        angle <- angle * 360 / 255
        afficher_donnees(0xD2,angle)
        retourne angle
Fin
```

3.2.7. Fichier commande_hacheur.c

Variables Globales
Entier static etat <- 0

3.2.7.1. INIT_MOTEUR()

Cette fonction sert à initialiser les bobines du moteur pour que l'on puisse démarrer dans des conditions optimales. Avec celle-ci, on est sûr que le prochain transistor à activer est le 1 (état 0).

```
Début
        T4 <- 1
        Attendre 40 ms
        T4 <- 0
        Attendre 40 ms
        T2 <- 1
        Attendre 40 ms
        T2 <- 0
        Attendre 40 ms
        T3 <- 1
        Attendre 40 ms
         T3 <- 0
        Attendre 40 ms
        T1 <- 1
        Attendre 40 ms
        T1 <- 0
        Attendre 40 ms
Fin
```

3.2.7.2. SENS_HORAIRE(ENTIER PAS)

Cette fonction sert à faire tourner le moteur dans le sens horaire.

```
Variable
        Entier i
Début
        LED1 <- 1;
        Pour i de 1 à pas
                Selon (etat)
                        cas 0 :         T4 <- 1
                                        Attendre 40 ms
                                        T4 <- 0
                                        Attendre 40 ms
                                        etat <- 1
                        cas 1 :         T2 <- 1
                                        Attendre 40 ms
                                        T2 <- 0
                                        Attendre 40 ms
                                        etat <- 2
                        cas 2 :         T3 <- 1
                                        Attendre 40 ms
                                        T3 <- 0
                                         Attendre 40 ms
                                        etat <- 3
                        cas 3 :         T1 <- 1
                                        Attendre 40 ms
                                        T1 <- 0
                                        Attendre 40 ms
                                        etat <- 0
```

```
            Fin selon
        Fin pour
        LED1 <- 0
Fin
```

3.2.7.3. SENS_ANTI_HORAIRE(ENTIER PAS)

Cette fonction sert à faire tourner le moteur dans le sens antihoraire.

```
Variable
        Entier i
Début
        LED2 <- 1
        Pour i de 1 à pas
                Selon (etat)
                        cas 0 :         T3 <- 1
                                        Attendre 40 ms
                                        T3 <- 0
                                        Attendre 40 ms
                                        etat <- 1
                         cas 1 :        T2 <- 1
                                        Attendre 40 ms
                                        T2 <- 0
                                        Attendre 40 ms
                                        etat <- 2
                        cas 2  :        T4 <- 1
                                        Attendre 40 ms
                                        T4 <- 0
                                        Attendre 40 ms
                                        etat <- 3
                        cas 3  :        T1 <- 1
                                        Attendre 40 ms
                                        T1 <- 0
                                        Attendre 40 ms
                                        etat <- 0
                Fin selon
        Fin pour
        LED2 <- 0
Fin
```

3.2.7.4. TOURNER_VERS_NORD(ENTIER DEGRE)

Cette fonction sert à diriger l'axe du moteur vers le Nord.

```
Variables
        Entier pas
Début
        Si degre > 180
                pas <- (360 - degre) * 200 / 360
                Si pas > 2
                        Tourner dans le sens horaire de pas
                Fin si
        Sinon
                pas <- (degre * 200) / 360
                Si pas > 2
                        Tourner dans le sens antihoraire de pas
                Fin si
        Fin si
Fin
```

3.2.7.5. TOURNER_AVEC_POT(ENTIER DEGRE_BOUS, ENTIER DEGRE_POT)

Cette fonction sert à diriger l'axe du moteur vers la valeur que nous retourne le potentiomètre (entre 0° et 360°).

```
Variables
        Entier pas
Début
        Si degre_bous < degre_pot
                pas <- (degre_pot - degre_bous) * 200 / 360
                Si pas > 2
                        Tourner dans le sens horaire de pas
                Fin si
        Sinon
                pas <- (degre_bous - degre_pot) * 200 / 360
                Si pas > 2
                        Tourner dans le sens antihoraire de pas
                Fin si
        Fin si
Fin
```

4. TESTS

4.1. Test hacheur

Nos premiers tests ont porté sur les transistors K_n, pour ce faire nous avons remplacé la bobine par un ampèremètre afin de relever la valeur et le sens du courant. On a donc commandé les transistors K1 et K4 on a observé un courant de 0.8 A de plus lorsqu'on a commandé les transistors K2 et K3 on a observé un courant de -0.8 A ce qui correspondait bien à la valeur souhaitée dans les calculs théorique. La commande des transistors a nécessité l'utilisation d'une alimentation en source de courant que nous avons réglé à la valeur de 0.1 A. Lors de notre premier test, nous avions oublié de prendre en compte le fait qu'une résistance de quart de watt n'était pas suffisante au niveau de la résistance R_{limite}, on a donc eu une surchauffe de cette résistance.

Ensuite ayant constaté que l'optocoupleur ne délivrait pas assez de courant pour saturer le transistor K_n, nous avons dû ajouter un étage d'amplification, pour cela nous avons ajouté le transistor Ta qui a été commandé en simulant le courant de l'optocoupleur à l'aide de l'alimentation. En faisant cet essai nous avons eu les mêmes résultats que précédemment.

Une fois ces essais effectués, nous avons ajouté l'optocoupleur puis fait un nouvel essai avec les sorties du microcontrôleur.

Grâce a cet essai on a remarqué que certains optocoupleurs présents dans le boitier en contenant quatre ne fonctionnaient pas, on a donc résolu ce problème en les remplaçants. Au préalable nous avons mesuré la tension Vcc et le courant maximal délivré par le microcontrôleur, nous avons mesuré V_{cc}=5V et un courant environ égale à 36 mA en accord avec nos calculs théorique. Par la suite notre hacheur était opérationnel.

Pour finaliser l'étude du fonctionnement du hacheur, on a branché chaque bobine du moteur à leurs hacheurs respectifs. On a observé le moteur effectuait des pas (à l'aide d'un programme développé dans la partie programmation). Puis pour vérifier que les diodes effectuaient bien leur rôle (dissipation de l'énergie emmagasinée dans la bobine) on a utilisé un oscilloscope afin de s'assurer que le potentiel a la sortie de la bobine était inferieur au potentiel situé a la cathode de la diode supposé conductrice, cette vérification s'est avéré correcte.

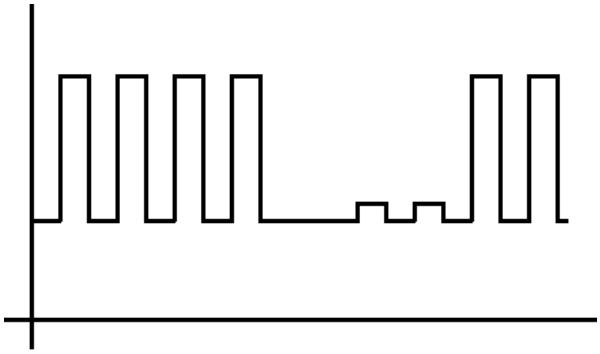
4.2. Test programmation

4.2.1. I2C et boussole

De manière générale, nous avons eu un gros souci de compréhension de la documentation de la boussole. Au-delà du fait que cette dernière est en anglais, elle est très mal expliqué et également mal présentée. De ce fait, nous avons perdu beaucoup de temps sur la compréhension de cette documentation, les problèmes ci-dessous sont notamment dus à la documentation mais pas uniquement.

Nous avons cherché à comprendre ce que renvoyait la boussole mais aussi si nous pouvions lui envoyer des commandes. Lors des premiers tests, cette dernière ne réagissait pas. On a voulu voir si les acknowledge étaient bien mis à 1 (voir algorithme send_i2c). Nous avons alors constatés que ce n'était pas le cas. Nous avons étudié de nouveau la documentation et c'est à ce moment que l'on a vu un mode appelé « exit sleep mode » (commande 'W'). Nous avons alors testé le réveil de la boussole, les acknowledges sont à ce moment passés à 1 (pour le réveil dans un premier temps). Ce problème a été long à debugger.

A partir de là, nous avons alors décidé d'envoyer de nouvelles commandes pour tester le bon fonctionnement de la boussole (écrire dans la RAM, lire la RAM). Là encore, on pouvait voir des acknowledge à 0 toutefois le réveil avait son acknowledge à 1. Nous avons cherché longtemps ou était le problème. On a alors décidé d'observer les signaux à l'oscilloscope sur conseil de nos professeurs. On a alors observé le signal suivant pour SCL :



On a alors cherché à supprimer les petits créneaux, nos tests nous ont menés à diminuer la fréquence de l'horloge d'envoi de données (SCL). Ainsi nous avons pu supprimer les petits créneaux qui ne devaient pas être là. Ici encore, ce débogage a été long.

Les deux problèmes précédents résolus, toutes les conditions nécessaires afin d'envoyer des commandes étaient réunis, du moins nous le pensions. Nous avions écrit le programme send_i2c afin de réceptionner des données. Ce programme ne renvoyait pas ce qu'il était censé renvoyer. Nous nous y sommes intéressés de plus près, c'est là que nous avons vu sur le programme même que SDA et SCL n'étaient pas remis à 0 au bon moment (dans notre programme nous le faisions pas du tout). Le résultat était donc logique. Seulement, il n'y avait pas qu'un unique problème, en effet après avoir relu la documentation, nous avons pu constater que l'envoi d'un acknowledge(ACK) n'était pas suffisant, il fallait aussi envoyer un NACK (non acknowledge). Ici encore le résultat ne pouvait être correct. Après ces deux corrections, le programme réception réalisait la fonction voulut.

Ensuite, nous pouvions à priori réaliser tous les envois de commande que nous voulions. Nous avons réalisé plusieurs essais, en observant à chaque envoi les acknowledge. Là nous avons constaté que ces acknowledges étaient pour certains à 1 d'autres à 0. Nous étions surpris de voir cela car en théorie les programmes fonctionnaient tous. Là encore, une étude de la documentation a été nécessaire, et nous n'avions pas remarqué le tableau suivant, et surtout la colonne « delays ».

Table 3 – Interface Command Delays

Command Byte ASCII (hex)	Description	Time Delay (µsec)
w (77)	Write to EEPROM	70
r (72)	Read from EEPROM	70
G (47)	Write to RAM Register	70
g (67)	Read from RAM Register	70
S (53)	Enter Sleep Mode (Sleep)	10
W (57)	Exit Sleep Mode (Wakeup)	100
O (4F)	Update Bridge Offsets (S/R Now)	6000
C (43)	Enter User Calibration Mode	10
E (45)	Exit User Calibration Mode	14000
L (4C)	Save Op Mode to EEPROM	125
A (41)	Get Data. Compensate and Calculate New Heading	6000

Comme on peut le voir des temporisations à durée précise (que nous avons considérés comme minimale en raison des éventuelles imprécisions des temporisations). Nous avions au départ mis des temporisations de la durée égale à la temporisation utilisé dans le programme I2C, en précautions nous en avions mis plusieurs, cependant ce n'était pas suffisant. Nous avons donc corrigé cette erreur en rajoutant des temporisations qui respectaient au moins les délais voulus. Si cette temporisation était trop longue, cela ne posait en aucun cas de problème, il y aurait juste eu un délai de réponse entre chaque instruction.

Une fois la réception des données de la boussole et l'envoi de commande réglé, nous pouvions enfin tester les données que renvoyait la boussole. Nous avions trouvé un registre dans la documentation renvoyant les coordonnées X et Y par rapport au nord. Nous pensions à tort que nous devions consulter ce registre selon un ordre précis et récupérer au fur et à mesure les données X et Y par rapport au nord. Nous nous sommes alors lancés dans l'écriture de programme rendant un angle en fonction des coordonnées. Cependant la consultation de ces coordonnées nécessitait un accès au registre de la RAM. Nous cherchions par tous les moyens à récupérer les coordonnées X et Y, que nous avons réussi en consultant la documentation :

Bit 2	Bit 1	Bit 0	Description
0	0	0	Heading Mode
0	0	1	Raw Magnetometer X Mode
0	1	0	Raw Magnetometer Y Mode
0	1	1	Magnetometer X Mode
1	0	0	Magnetometer Y Mode

The total bit format for the Output Mode Byte is shown below:

Bit 7 (MSB)	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0 (LSB)
0	0	0	0	0	Mode	Mode	Mode

Nous avons alors remarqué un mode supplémentaire : « heading mode ». Nous ne réussissions pas à comprendre à quoi correspondait ce mode, on a alors testé ce mode en envoyant la commande en question (0x00). A ce moment on a remarque que l'on recevait l'angle lui-même. Notre programme de conversion n'était donc pas nécessaire, et la documentation n'avait pas précisé que l'angle pouvait être directement récupérer (du moins nous ne l'avions pas compris).

Pour finir, nous précisons que les valeurs d'angles ne sont pas exactes en raison des perturbations magnétiques que peuvent créer divers appareils électronique autour mais aussi à cause du fait que la boussole est désaxée.

4.2.2. Ecran

Tout d'abord, le fonctionnement de l'écran a été long à mettre en œuvre car nous n'avions pas relié la broche V5 de l'écran à la masse. L'écran ne pouvait donc pas fonctionner. Ce bug a été résolu avec l'aide de nos collègues, ces derniers ont eu le même problème.

Lors de l'insertion du CAN à notre solution, nous avons dû changer les ports de l'écran car ce dernier se trouvait sur l'une des broches nécessaires au fonctionnement du CAN. Nous avons alors mis l'écran sur le port C par accident (les broches du port C et D étant mélangés sur la carte), ce port étant utilisés par l'USB, nous ne pouvions plus connecter notre PIC à notre ordinateur. Nous avons à l'aide de l'un de nos professeurs remarqué cette erreur et remplacé les broches utilisées par le port C par uniquement des broches du port D pour l'écran. L'écran était après opérationnel.

4.2.3. Clavier

Le principal problème constaté résidait dans la fréquence, en augmentant le pré-diviseur et en tournant le potentiomètre, les variations étaient « lentes à arriver » et à l'inverse lorsque l'on diminue le pré diviseur et qu'on augmente le potentiomètre, les variations sont plus « rapides à arriver ».Nous avons donc choisit le juste milieu et pris le pré-diviseur 32.

De manière générale, nous avions un problème avec nos temporisations variables, nous avions envoyé la partie basse avant la haute, ce qui était une erreur. En envoyant la partie basse avant la haute, l'envoi de la partie haute écrasait la partie basse. Nous avons donc inversé ces envois, la temporisation fonctionnait ensuite comme voulu. Nous n'avions pas fait attention à cela mais cette donnée était une donnée constructeur.

CONCLUSION

Pour conclure, nous proposons une solution répondant au cahier des charges, les petites différences présentes sont dues à la boussole elle-même qui peut être perturbée par des appareils électroniques. En l'absence de clavier, ce point du cahier des charges n'est pas rempli et a donc été remplacé par un CAN et un potentiomètre.

Cependant, nous aurions pu avec plus d'heures de TP de synthèse, améliorer nos programmes en ajoutant notamment des interruptions ou simplement en les optimisant. De plus, pour pleinement remplir le cahier des charges, avec plus de temps nous aurions pu commander un clavier et l'implanter dans notre système. Nous aurions pu également créer une carte qui aurait comporté les différents éléments du hacheur, ceci aurait fait suite au routage fait sur Proteus. Enfin, nous aurions pu nous passer des transistors d'amplification en passant une commande de transistor nécessitant un courant dans la base plus petite.