Réalisation d'un testeur à sonde mobile pour cartes électroniques

Georges Sarkis

Conservatoire National des Arts et Métiers

CENTRE REGIONAL ASSOCIE DU LIBAN

CENTRE D'ENSEIGNEMENT DE BEYROUTH

MEMOIRE

Présenté en vue d'obtenir

LE DIPLOME D'INGENIEUR CNAM

En

ELECTRONIQUE

Par

SARKIS Georges

Réalisation d'un testeur à sonde mobile pour cartes électroniques

---

Soutenu le: Novembre - 2014

JURY

Président: Mr Michel TERRÉ

Membres: Mr Khaled ITANI

Mr Dany MERHEJ

Mr

Mr Sami DIAB

Mr

Sommaire

Tables des figures

Glossaire

API: Application Programming Interface

CNC: Computer Numerical Control

GUI: Graphical User Interface

PCB: Printed Circuit Board

SMD: Surface Mount Device

PLL: Phase Lock loop

USB: Universal Serial Bus

CAN: Convertisseur Analogique Numérique

CAD: Computer-Aided Design

ICSP: In Circuit Serial Programming

NRZI: Non-Return-to-Zero Inverted

DLL: Dynamic Link Library

TPI: Test Point Information

CSV: Comma Seperated Values

Remerciements

Au début, Je voudrais exprimer ma gratitude à monsieur Fadi DAOU, le CEO de notre entreprise MultiLane qui m'a confier de concevoir et étudier ce projet au sein de son entreprise. J'adresse également mes sincères remerciements à mes collègues dans l'entreprise, monsieur Sami DIAB et monsieur Jean HINDI pour leur temps précieux qu'ils m'ont consacré durant la préparation de ce mémoire.

Je remercie bien évidemment monsieur Dany MERHEJ de bien avoir tuteuré ce projet et dirigé ce mémoire. Je le remercie pour le temps qu'il m'a procuré pour discuter et voir les résultats semaine après semaine. Je remercie également monsieur Khalid ITANI, que j'appelle le père de tous les étudiants, pour son bureau toujours ouvert pour entendre toutes nos complaintes.

Merci aussi à monsieur Elie HADDAD, propriétaire d'un tour qui m'a consacré deux jour complets pour m'aider à l'exécution du système mécanique.

Et bien évidemment, je ne voudrais pas terminer sans une pensée toute particulière à ma fiancée pour le support quotidien durant les trois dernières années pour arriver à terminer ce diplôme.

Merci à tous et à toutes.

Résumé

Ce projet décrit la conception et la réalisation d'un système de « test à sonde mobile » pour cartes électroniques, à travers une interface graphique.

Le concept de base pour ce projet est d'installer une sonde sur un bâti mécanique pouvant la déplacer selon les 3 axes X, Y et Z, à l'aide de moteurs pas-à-pas.

Le projet comporte trois parties : la première concerne la conception et la réalisation électronique (Firmware et hardware), la deuxième la conception et la réalisation du système mécanique, et la troisième le logiciel de contrôle sur PC et son interface graphique.

En ce qui concerne la réalisation électronique, c'est à travers un microcontrôleur qui fait contrôler tout le système avec un protocole USB, en utilisant des drivers pour les différents moteurs pour varier la fréquence à travers les impulsions venue du micro vers les drivers. Le microcontrôleur est alors programmé à travers un programme qu'on appelle firmware à travers la langage C.

Le système mécanique doit comporter trois axes variables, ces axes sont contrôlés par des moteurs pour amener le déplacement tridimensionnel. Bien sur c'est avec des guides et des paliers linéaires pour faire glisser les axes dans toutes les directions.

L'interface graphique, c'est la partie logicielle qui doit être devant l'utilisateur. Elle est codé avec le langage c# et c++ (langage de haut niveau). Son rôle principal est de communiquer avec la partie firmware à travers une interface intermédiaire pour émettre et recevoir des paquets USB pour contrôler les drivers des moteurs et afficher les résultats de tensions pour les différents points de la carte électronique venant du sonde.

Des tests spécifiques ont été réalisés pour valider les performances du système obtenu qui peut en outre être utilisé pour d'autres types d'applications nécessitant un mouvement tridimensionnel.

Introduction

En 1947 le control numérique des machines a pris naissance. Tout a commencé lorsque John Parsons un fabriquant de pales de rotor d'hélicoptère, ne pouvait pas faire ses modèles assez vite.

1949 a été l'année de la **nécessité urgente**, la commande de matériel aérien américaine s'est rendu compte que les pièces détachées pour ses avions et missiles ont été de plus en plus complexe. En outre, comme les designs ont été constamment améliorés, des changements dans le design ont souvent été faits. Ainsi dans leur recherche pour des méthodes de production plus rapide. Un contrat d'étude de la force aérienne a été attribué à la **Parsons corporation.** Le laboratoire des mécanismes d'asservissement de l'Institut de Technologie de Massachusetts(MIT) a été le sous-traitant. En 1951, le MIT a repris le travail complet, alors en 1952, le prototype d'aujourd'hui de la machine NC (Numerical Control), une machine modifiée a été démontrée. L'origine du terme de la commande numérique a été au MIT. [1]

De nos jours, des machines automatisées sont innombrables. On peut trouver plusieurs raisons pour lesquels on s'oriente à l'innovation et la construction de ces machines automatiques. On cite entre autres faciliter le travail des hommes, minimiser les erreurs (à cause de la fatigue, des tâches répétitives,..) améliorer le rendement temporel, etc.…,

Lorsqu'on parle d'une machine industrielle, il faut tout d'abord prendre en considération qu'il y a un système mécanique qui doit être conçus avec précision en plus du système électrique ou électronique qui doit contrôler cette machine. Puisqu'on parle d'un système mécanique, c'est-à-dire qu'on a un mouvement qui doit être assuré par des composantes mécaniques, on parle alors d'actionneurs qui sont la combinaison du système mécanique et électrique. Alors ce type de système se trouve dans la catégorie de l'électromécanique. Et puisqu'on a besoin de quelque chose pour contrôler l'actionneur, on parle d'un contrôleur (ex. microcontrôleur, PLC, …) pour le contrôle et la communication entre les différentes parties du système.

La problématique du projet est la suivante:

Dans tous les appareils et instruments électroniques on trouve des cartes électroniques. Ces cartes doivent être testées pour vérifier le fonctionnement avant la mise en service. La société dans laquelle je travaille « MultiLane » fait des designs dans le domaine de l'électronique de

télécommunication comme des équipements de tests pour des signaux à haut débit, pour mesurer et faire des simulations sur les lignes de transmission. Ces équipements comportent des cartes électroniques, qui nécessitent de test avant de les fournir à nos clients. Le test est divisé en deux parties, la première est le test du hardware et la deuxième celle du software. Dans la partie hardware il faut tester tous les « points de test » sur la carte qui possèdent des tensions variées l'un par rapport à l'autre.

Ce test consiste à mesurer à l'aide d'un voltmètre la tension à ces points en se basant sur les schémas électroniques (Design) et les circuits imprimés (Layout). Ce test peut prendre parfois quelques heures. L'idée de ce mémoire est alors de réaliser un système qui permet de minimiser le temps de ce test, et réduire éventuellement les erreurs de manipulation, et permet de me donner une expérience dans le domaine de la robotique.

Ce rapport de mémoire se divise en quatre chapitres qui permettent d'expliquer la réalisation de cette machine ainsi que son fonctionnement électronique et mécanique.

Dans le premier chapitre on va présenter le concept de ce projet et la sélection des composantes électroniques et mécaniques en se basant sur un schéma bloc.

Dans le second chapitre on va présenter le fonctionnement de la carte microcontrôleur responsable du contrôle des actionneurs mécaniques qui sont dans notre cas des moteurs pas à pas. On détaillera les algorithmes de fonctionnement de ces moteurs à travers des drivers ainsi que la communication à travers un port USB avec le logiciel de contrôle implémenté sur PC.

Le troisième chapitre présente l'interface graphique pour l'utilisateur, en détaillant les différents algorithmes qui permettent la connexion avec la partie hardware (chapitre 2) à travers une couche logicielle intermédiaire qu'on appelle API.

Le quatrième et dernier chapitre présenter les tests de fonctionnement global du système réalisé, en détaillant l'utilisation de l'interface graphique et la gestion des fichiers d'entrée contenant les coordonnées 3D (fichiers « Gerber ») et de sortie (résultats des tests effectués).

Une conclusion générale avec perspectives termine ce rapport en citant les applications qu'on peut les faire à l'aide du système réalisé et quelles sont les améliorations qui peuvent être mises en œuvre.

Chapitre 1

Etude du projet et sélection des composantes

1.1- Introduction

Dans ce chapitre on va présenter l'étude de ce projet, en donnant quelque exemple comme les machines CNC, en expliquant l'idée de ce projet à travers un schéma bloc qui permet d'illustrer le fonctionnement générale de tout le système. Après on va discuter le choix des matériels électroniques et mécaniques qui permettent tout ensemble de concevoir la machine, en présentant l'idée de base de contrôle du système à commander.

1.2- Principe de base des machines CNC

Computer Numerical Control » (CNC) est une machine dans laquelle les fonctions de mouvements sont commandées par l'intermédiaire d'un programme préparé contenant des données codées. Le CNC peut contrôler les mouvements de la pièce ou de l'outil, les paramètres d'entrée tels que, la profondeur de la coupe, la vitesse et les fonctions comme la rotation d'une broche, etc. [2]

Le système effectue de nombreuses fonctions et mouvements qui étaient traditionnellement effectuées par des mécaniciens qualifiés.

L'usinage avec le CNC commence avec un programme, qui est une des séquences séquentielles ou commandes codées qui dirigent les fonctions spécifiques de la machine.

Ces commandes codées sont convertis en signaux de sortie, qui déterminent les opérations de la machine comme la vitesse de la broche, le choix et le mouvement de l'outil, etc.

Le programme peut être généré manuellement ou par des systèmes de programmation assistée par ordinateur.

En intégrant un ordinateur, le CNC permet aux programmes d'usinage d'être modifiés et stockés dans la mémoire de l'ordinateur, ainsi que les fonctions de diagnostic et de contrôle de la qualité encours d'usinage.

Toutes les machines commandées par ordinateur peuvent contrôler avec répétition et précision le mouvement dans les différentes directions. Chacune de ces directions de mouvement est appelé « un axe ». Selon le type de la machine, elles possèdent deux à cinq axes. Ces axes peuvent êtres soit linéaires, dans lesquelles le mouvement se produit selon une ligne droite, ou rotatifs dans lesquelles le mouvement se produit suivant une trajectoire circulaire.

On trouve différents types de machines CNC qui sont utilisées dans l'industrie:

- CNC tournant (tours) - tours CNC produisent des pièces tournantes en faisant tourner les matériaux en forme de tige et l'alimentation d'un outil de coupe dans le matériau de braquage.
- Fraiseuse à commande numérique : Utilisation d'un outil de coupe cylindrique rotatif, elle utilise un procédé d'usinage similaire à la fois forage et coupage. L'outil de coupe dans une machine à fraiser possède la capacité de se déplacer le long de multiples axes et peut créer une variété de formes, de fentes et de trous
- Perceuse : Utilisé aussi avec la fraiseuse pour perforer des pièces métallique ou autre matière avec une grande précision.
- Machine de flamme et de découpage au Laser : peuvent être utilisés pour la coupe de différents matériaux durs comme les matériaux composites, l'aluminium, le fer et l'acier qui ont une grande épaisseur.
- Pick and place machine : utilisé pour fixer les composantes sur la carte électronique
- Etc…

1.3- Proposition du projet et schéma bloc

Le test des cartes électroniques dans notre entreprise « Multilane »se fait de façon manuelle à l'aide de multimètres, en se basant sur les schémas électroniques (design) et les circuits imprimés (layout). Pour cela j'ai proposé au CEO la réalisation d'une machine pour automatiser cette procédure, améliorer sa documentation, et la rendre plus rapide et plus fiable.

Le CEO a apprécié l'idée, mais il m'a averti de la difficulté de la partie mécanique vue qu'elle exige une précision de l'ordre de 10^{-2}mm.

Après quelques recherches et une étude de faisabilité j'ai construit un schéma bloc du projet, présenté dans la figure 1, et qui permet de présenter les principales parties du système à réaliser.

Trois moteurs pas à pas assurent les mouvements selon les 3 axes X, Y et Z munis chacun d'un interrupteur de fin de course.

Ces trois moteurs sont gérés par un microcontrôleur qui communique avec un PC contenant un logiciel de contrôle, à travers une liaison USB ou par l'Ethernet. Le microcontrôleur fournit aux drivers des moteurs les impulsions nécessaires pour effectuer des mouvements individuels ou simultanés. Il reçoit aussi les informations des interrupteurs de fin de course et convertit la tension mesurée par la sonde en valeur numérique pour l'envoyer au PC de contrôle.

Dans l'industrie électronique, la réalisation des circuits imprimés PCB (Printed Circuit Board) utilisent des logiciels CAD (Computer Aided Design). Ces logiciels génèrent des fichiers excellon ou Gerber pour effectuer la gravure et le perçage des PCBs. On peut extraire de ces fichiers la position des points de tests sur la carte PCB et utiliser cette information est utilisée dans notre système. Cette tâche sera celle de contrôle sur PC qui communiquera les points à tester au microcontrôleur et recevra les niveaux de tension correspondants.

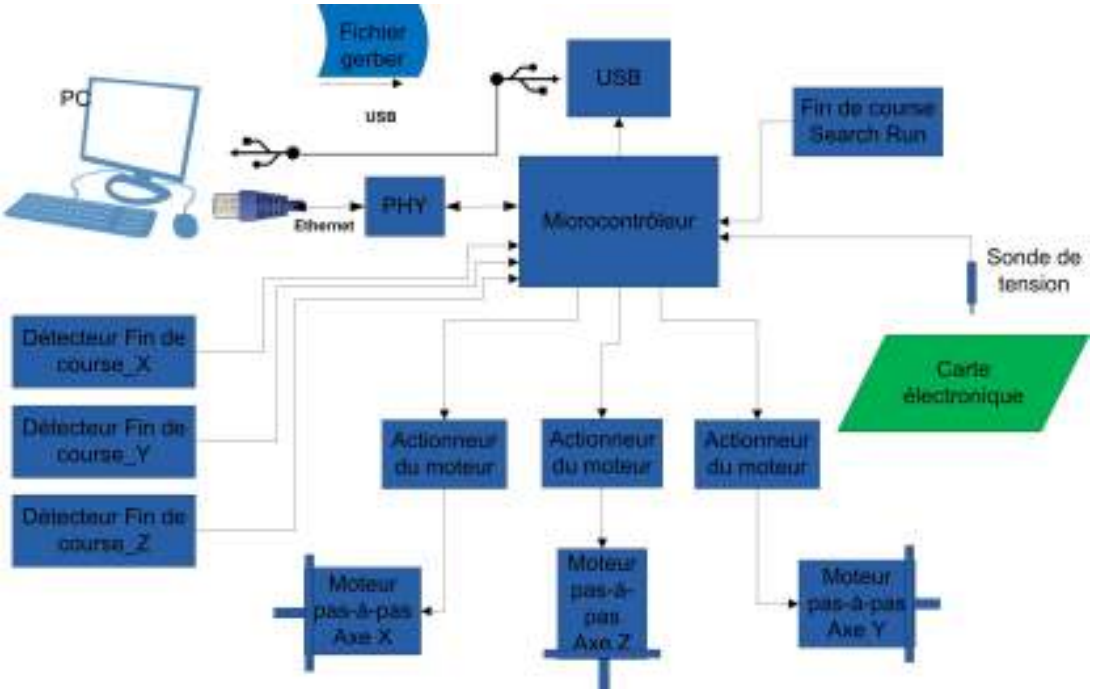


Figure 1- Schéma bloc du projet

En se basant sur ce schéma bloc le plan de travail est le suivant :

La première phase est de réaliser la partie mécanique « Construction du châssis » qui permet de porter les différents composants mécaniques comme les moteurs et les capteurs nécessaires. Cette phase contient trois étapes, le dimensionnement de la machine selon les dimensions des cartes électroniques à tester, sa conception (dessin) et sa réalisation dans un tour.

La deuxième phase est la réalisation de la carte à microcontrôleur qui permet de contrôler la partie mécanique. Dans cette phase on a aussi quatre étapes,

- la première est la sélection des composantes électroniques répondant aux exigences demandées et la conception du schéma électronique.
- La deuxième est la conception matérielle de la carte (placement et routage), dans laquelle il faut bien dimensionner la carte pour faciliter son brochage et son positionnement sur le bâti.
- La troisième étape est la fabrication du circuit imprimé. Pour cela j'ai choisi un fabriquant en Chine (www.myropcb.com), ce qui m'a permis d'utiliser des composants SMD et de minimiser les dimensions de la carte.
- Enfin, une fois la carte reçue, il faut souder les composantes et faire un teste global de la carte.

La troisième phase est la « phase de programmation » qui consiste à décrire toutes les couches nécessaires pour la communication entre le logiciel de contrôle sur PC comportant l'interface graphique de l'utilisateur (GUI) et la gestion des fichiers (en entrée, e.g. Gerber, et en sorties, e.g. résultats des tests) et la carte à microcontrôleur (firmware) responsable de la gestion des capteurs (interrupteurs, phototransistors, sonde de tension) et actionneurs (moteur pas à pas) du système.

Enfin il faut procéder à un teste globale du système réalisé pour vérifier son fonctionnement, son respect des exigences demandées (précision) et sa fiabilité.

1.4- Sélection du matériel électronique

1.4.1- Microcontrôleurs

Par rapport à des systèmes électroniques à base de microprocesseurs et d'autres composants individuels, les microcontrôleurs permettent de diminuer la taille, la consommation électrique et le coût des produits.

Les microcontrôleurs sont fréquemment utilisés dans les systèmes embarqués, comme les contrôleurs des moteurs automobiles, les télécommandes, les appareils de bureau, l'électroménager, les jouets, la téléphonie mobile, etc.

Pour sélectionner le type du microcontrôleur à utiliser, on doit savoir le type de communication avec ce microcontrôleur, e.g. série (RS232 ou RS485), USB ou Ethernet, etc.. Aussi il faut savoir le nombre de broches d'entrée/sortie nécessaires à être raccordées avec les autres composantes.

En vue des exigences demandées pour mon système j'ai choisi le microcontrôleur **PIC18F87J50** qui possède les caractéristiques suivantes [3] (voir aussi annexe A):

- Microcontrôleur 1Mbits compatible avec USB V2.0
- Permet de supporter 32 terminaux (16 bidirectionnels)
- 3.9 KB RAM double accès pour la liaison USB
- Transceiver USB intégré.
- Interfaçage mémoire externe à 8 ou 16 bits.
- Modes d'adressage à12-bits, 16-bits ou 20-bits.
- Programmation ICSP à travers 2 broches.
- Convertisseur CAN10-bits.
- 9 ports d'E/S.
- Supporte les protocoles I2C et SPI.

Selon la documentation technique, pour que la liaison USB fonctionne correctement avec ce microcontrôleur il faut utiliser un Crystal de fréquence **multiple de 4 MHz**. Pour cela j'ai utilisé un Crystal de 12MHz (**ABM3-12.000MHZ-B2-T**).

1.4.2- Le contrôle des moteurs

Le moteur d'un pas-à-pas comporte trois parties :

- L'interface de puissance
- Le contrôleur
- la logique de commande

L'interface de puissance permet de fournir les courants nécessaires aux bobines du moteur. C'est le rôle des drivers à transistors qui ont une puissance suffisante pour effectuer ce travail.

Cependant les bobines du moteur pas à pas doivent êtres alimentées selon une séquence bien définie. C'est la tâche du contrôleur d'alimenter ces bobines dans le bon ordre. On va avoir plus loin les différentes séquences possibles.

Enfin, c'est la partie logique de commande, qui est chargée de faire fonctionner le moteur d'une manière cohérente à long terme. Dans cette partie on peut par exemple déterminer la vitesse à laquelle le contrôleur doit répéter son cycle (séquences d'impulsions), choisir le sens de rotation, activer ou inhiber l'alimentation du moteur, etc.

Les drivers des moteurs doivent être compatibles avec les niveaux de sorties (tension et courant) du microcontrôleur et les exigences des moteurs utilisés, pour cela j'ai choisi le **L298N** (Dual Full- bridge driver) figure 2, qui a les principales caractéristiques [4] ci-dessous :

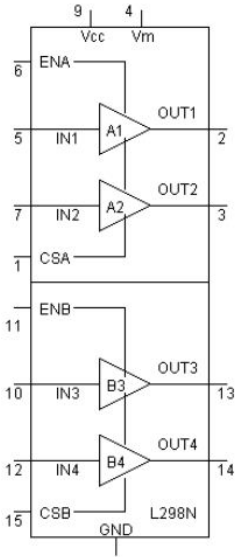


Figure 2- Schéma bloc du L298N

- Alimentation de fonctionnement $VDD = 5V$
- Tension d'alimentation des moteurs : 7.5V à 46V
- Accepte des commandes allant de 2.3 à VDD
- Courant DC des moteurs jusqu'à 4A (2A/enroulement)

Le schéma ci-suivant présente le contrôle d'un moteur pas-à-pas unipolaire ou bipolaire.

- In1, In2, In3, In4, EnA et EnB sont les entrées qui sont raccordés au microcontrôleur.
- OUT1, OUT2, OUT3 et OUT4 sont les sorties qui doivent être reliés avec les moteurs.

In1, In2, Out1 et Out2 sont les seules broches à utiliser si on a un moteur unipolaire.

1.5- Sélection du matériel mécanique

1.5-1. Actionneurs

Le système mécanique doit réaliser un mouvement avec précision selon 3 axes. La gestion de ce mouvement concerne la vitesse demandée, la distance à parcourir, la charge à supporter (et les frottements), l'inertie du système, et la combinaison de tous ces facteurs.

Il existe plusieurs façons de réaliser le mouvement, e.g. moteur pas-à-pas, moteur pas-à-pas linéaire, le servomoteur, moteur à courant continu avec ou sans balais, etc.

Dans ce projet j'ai choisi le moteur pas-à-pas bidirectionnel, pour les raisons suivantes :

(a) (b) (c)

Figure 3- (a) Symbole du moteur bipolaire (b) Moteur 1 : PL57H56-2.6-4 (c) : Moteur 2 pour l'axe Z

Il est peu onéreux, facile à contrôler et permet un fonctionnement en boucle ouverte.

Un modèle 3D pour le système mécanique a été conçu avant l'assemblage de toutes les composantes mécaniques. Le design a été étudié pour différents paramètres. Ces paramètres inclus la légèreté de poids, le faible frottement et le budget de réalisation.

Pour cela j'ai choisi deux types de moteurs pas-à-pas bipolaire (fig. 3.a), le premier type, un PL57H56-2.6-4(fig. 3.b), pour les axes X et Y, et le second type (fig. 3.c) qui est plus petit pour l'axe Z. Le premier est PL57H56-2.6-4 qui présenté par la figure ci-dessous :

La table 1 suivante donne les caractéristiques de ces moteurs.

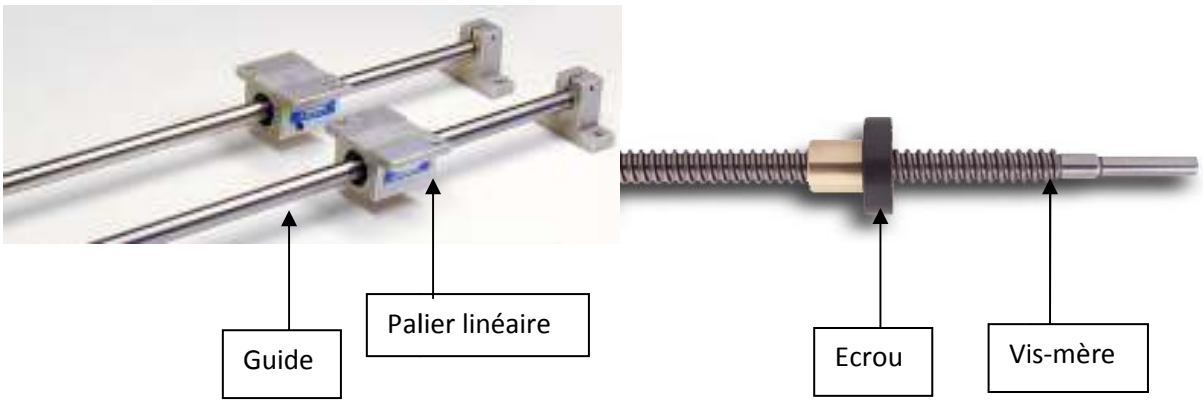
Caractéristiques	Moteur 1	Moteur 2
Courant (A)	2.6	0.33A
Couple (Nm)	1.26	0.23
Voltage(V)	24	12
Angle	1.8° ± 5%	1.8° ± 5%
Phases	2	2

Table 1- Caractéristiques des moteurs

1.5-2. Axes et paliers linéaires

En considérant tous les paramètres, le système mécanique a été construit avec les composants matériels suivants :

- Vis-mère de 14 mm (3 vis-mère pour les 3 axes X, Y et Z)
- Ecrou de 14mmde longueur 4cm
- Palier Linéaire pour le glissement (Linear Bearing)
- Guide pour faciliter le chemin du moteur



(a) (b)

Figure 4- (a): Palier linéaire avec guide (b): Vis-mère et écrou

Le système obtenu est le suivant :

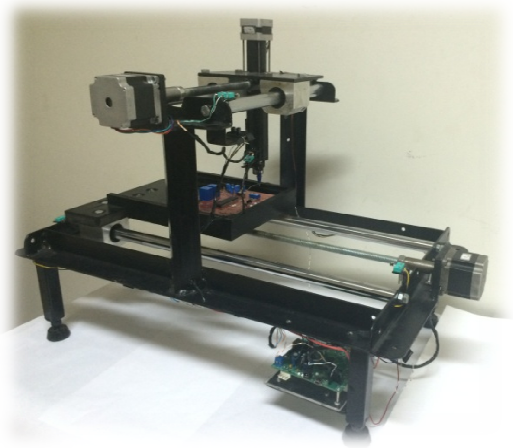


Figure 5- Système mécanique

1.6- Fonctionnement d'un moteur pas-à-pas

Le moteur est un équipement électromécanique qui permet de convertir l'énergie électrique en un mouvement de rotation. Les impulsions électriques permettent d'actionner (drive) le rotor et l'arbre du moteur. Ils sont connectés à un driver pour moteur pas-à-pas.

Ce driver est généralement connecté à un microcontrôleur et chaque impulsion permet de déplacer l'arbre du moteur d'un certain angle de rotation. Cet angle est appelé en anglais step

angle et il est fixe pour chaque moteur. La vitesse et la direction de rotation dépend de la séquence d'impulsions et de leur fréquence.

Dans notre cas le « step angle =1.8˚», alors il faut 200 impulsions [5] pour faire tourner l'arbre du moteur d'un tour complet. Plus le step angle est petit plus la précision augmente.

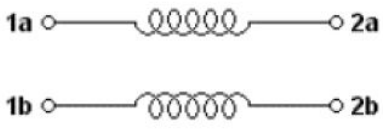
Cependant il existe une limite pour le nombre d'impulsions par seconde que le moteur peut accepter. Les moteurs de bonne qualité peuvent accepter au maximum 200 à300 impulsions par seconde, alors la vitesse angulaire maximale pour un« step angle =1.8˚» est de un à trois tr/s (rpm = 60 à 180).

Le champ électromagnétique produit par les bobines (enroulements du moteur pas à pas) est proportionnel directement au courant consommé. Plus le champ électromagnétique est grand plus le moteur fournit un couple plus grand.

Il faut noter que le moteur ne peut pas fonctionner directement à la vitesse maximale. Pour atteindre la vitesse maximale du moteur il faut accélérer graduellement. La vitesse peut commencer par 1/3Vmax pour quelques millisecondes, 2/3Vmax pour les 50 ou 75 millisecondes suivantes, après on peut atteindre la vitesse maximale.

1.6.1- Moteur pas-à-pas bipolaire

Tout d'abord, le moteur pas-à-pas unipolaire est facile à contrôler, il consomme moins de courant que celle du bipolaire. Or le moteur bipolaire est un peu différent. Ce type de moteur requiert un circuit de contrôle plus complexe. Les moteurs bipolaires sont connus par le rapport dimension/couple et il permet de fournir un couple plus grand que celle du moteur unipolaire de même dimension. Les moteurs bipolaires sont fabriqués avec des bobines séparées, qui ont besoin d'être alimentées sur dans les deux directions (la polarité doit pouvoir être inversée durant l'opération) pour achever les pas par ordre. La table 2 ci-dessous explique comment un moteur à 2-phases fonctionne.



Sens horaire

Indice	1a	1b	2a	2b
1	+	-	-	-
2	-	+	-	-
3	-	-	+	
4	-	-	-	+

Table 2- Séquence d'un moteur à deux phases

Le contrôleur du moteur bipolaire doit être capable d'inverser la polarité de tension sur chacune des bobines, pour cela le courant doit pouvoir circuler dans les deux directions. Le mécanisme qui inverse la tension sur une bobine est appelé en anglais « H-bridge », car il ressemble à la lettre « H ».

La table 3 ci-dessous décrit les trois séquences possibles pour les moteurs pas-à-pas bipolaires, la polarité des terminaisons est indiqués par les symboles +/-. Eventuellement après le dernier pas, la séquence se répète. (Phase = bobine).

Nom	**Séquence**	**Polarité 2b 2a 1b 1a**	**Description**
Une seule phase	0001 0010 0100 1000	---+ --+- -+-- +---	Consomme un courant faible, une seule phase est alimentée à un instant donnée. Assure une précision de positionnement quel que soit le déséquilibre des enroulements du moteur.
Couple élevé 2 phases	0011 0110 1100 1001	--++ -++- ++-- +--+	Cette séquence alimente deux phases adjacentes, elle permet d'offrir une amélioration du couple-vitesse et un bon couple de maintien.
Demi-pas	0001 0011 0010 0110 0100 1100 1000 1001	---+ --++ --+- -++- -+-- ++-- +--- +--+	Il permet de doubler la résolution du moteur, mais le couple n'est pas uniforme pour chaque pas. Il s'inverse à chaque pas. Cette séquence diminue la résonnance du moteur,

Table 3- Séquences d'impulsion_Moteur pas-à-pas

1.6.2- Caractéristiques des enroulements et influence de fréquence

La résistance et l'inductance sont deux propriétés physiques importantes pour l'enroulement. Ce sont les facteurs de base qui limitent la performance du moteur.

La résistance de l'enroulement est responsable de la perte de puissance et de l'échauffement du moteur. La perte de puissance est donnée par :

$$Pertes\ (w) = R.Im^2 \qquad \text{Eq. 1}$$

Avec, $R\ (en\ \Omega)$ et $Im\ (en\ \text{A})$ la résistance et le courant d'enroulement respectivement.

L'inductance permet à l'enroulement du moteur d'opposer le changement du courant, et elle permet de limiter sa vitesse. La figure 6 ci-dessous présente le comportement du circuit R-L lors des commutations.

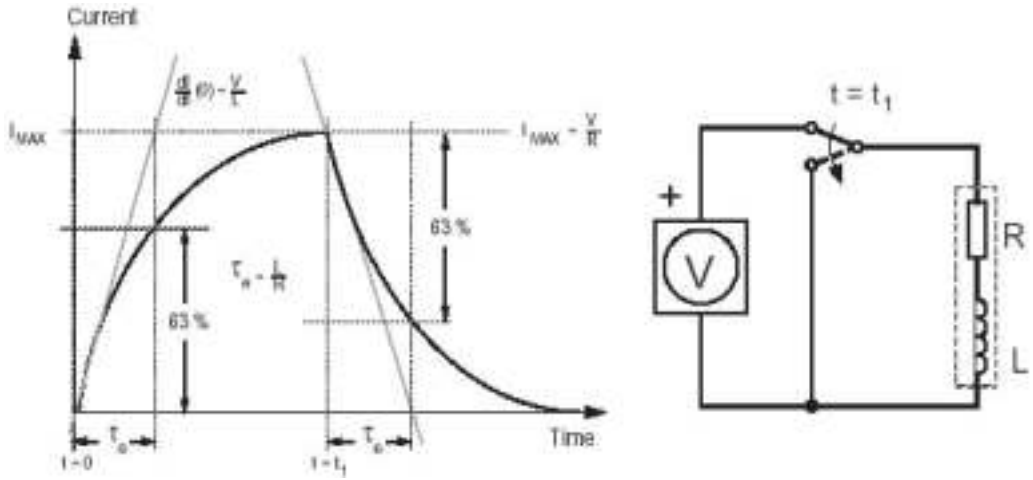


Figure 6- Courant dans un enroulement

Lorsqu'une tension est appliquée sur l'enroulement, le courant augmente suivant l'équation :

$$I\ (t) = \frac{V}{R}\left(1 - e^{-Rt/L}\right) \qquad \text{Eq. 2}$$

Initialement la dérivée du courant est: dI/dt(0)=V/L

L'augmentation du courant arrive à la valeur maximale : $Imax = V/R$

La constante de temps du circuit est définit par : $\tau = L/R$.Elle correspond au temps au bout duquel $I(t)$ atteint 63% de $Imax$.

Quand le circuit inductive-résistive n'est plus alimenté et il est court-circuité à l'instant$t = t_1$, le courant commence à décroitre suivant l'équation :

$$I\ (t) = \frac{V}{R} e^{-(t-t_1)R/L} \qquad \text{Eq. 3}$$

Etat initial : $I(t = t_1) = V/R$

Quand on applique un signal carré sur l'enroulement le courant est lissé, comme le présente la figure 7.a. En augmentant la fréquence, fig._7.b et 7.c, le courant ne peut plus atteindre sa valeur maximale. Comme le couple du moteur est approximativement proportionnel au courant, le couple maximal va diminuer quand la fréquence des impulsions augmente.

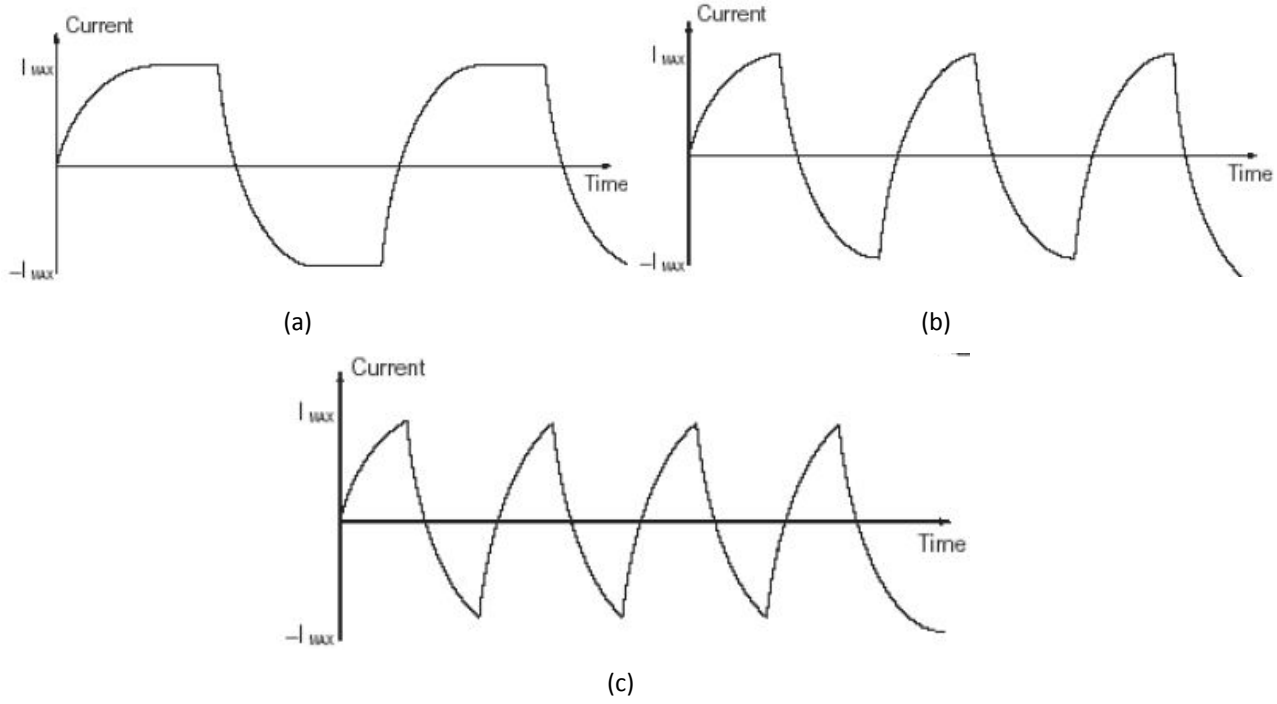


Figure 7- Effet de l'inductance pour un débit d'impulsion élevé

Chapitre 2

La partie matérielle et le contrôle de la carte

2.1- Introduction

Dans ce chapitre on va présenter le fonctionnement de toutes les parties matérielles, électronique et mécanique, mentionnées dans le chapitre 1, en expliquant brièvement la communication à travers le protocole USB.

2.2- Ingénierie du design

Tout d'abord, et quelque soit le projet électronique, il faut commencer le design électronique par le contrôleur principal, tel que le microcontrôleur puis les autres circuits intégrés qui sont nécessaires pour l'application comme les drivers des moteurs dans notre cas, etc. Concernant la partie d'alimentation pour toutes les composantes sur la carte du projet, c'est la dernière chose à faire après avoir conçu les différentes parties électronique et calculé leurs demandes énergétiques. Il faut étudier chaque composant selon sa **documentation technique (datasheet)** pour savoir la tension de fonctionnement, la demande en puissance, la puissance calorifique dissipé et la température de fonctionnement. Il faut aussi étudier la compatibilité au niveau des tensions logiques avec les autres composants du circuit et, le cas échéant, utiliser des circuits intégrés spéciales pour la traduction du niveau (Level translator).

Pour les circuits sensibles aux fluctuations de l'alimentation lors des commutations, il faut penser à ajouter des capacités de découplages pour le bon fonctionnement.

Plus on respecte les recommandations des datasheet des composants, plus les erreurs de notre design diminuent, et plus les problèmes qui nous confrontent durant la réalisation pratique seront minimes.

2.3- Le schéma électronique du microcontrôleur

La figure 8 présente le circuit de connexions du microcontrôleur.

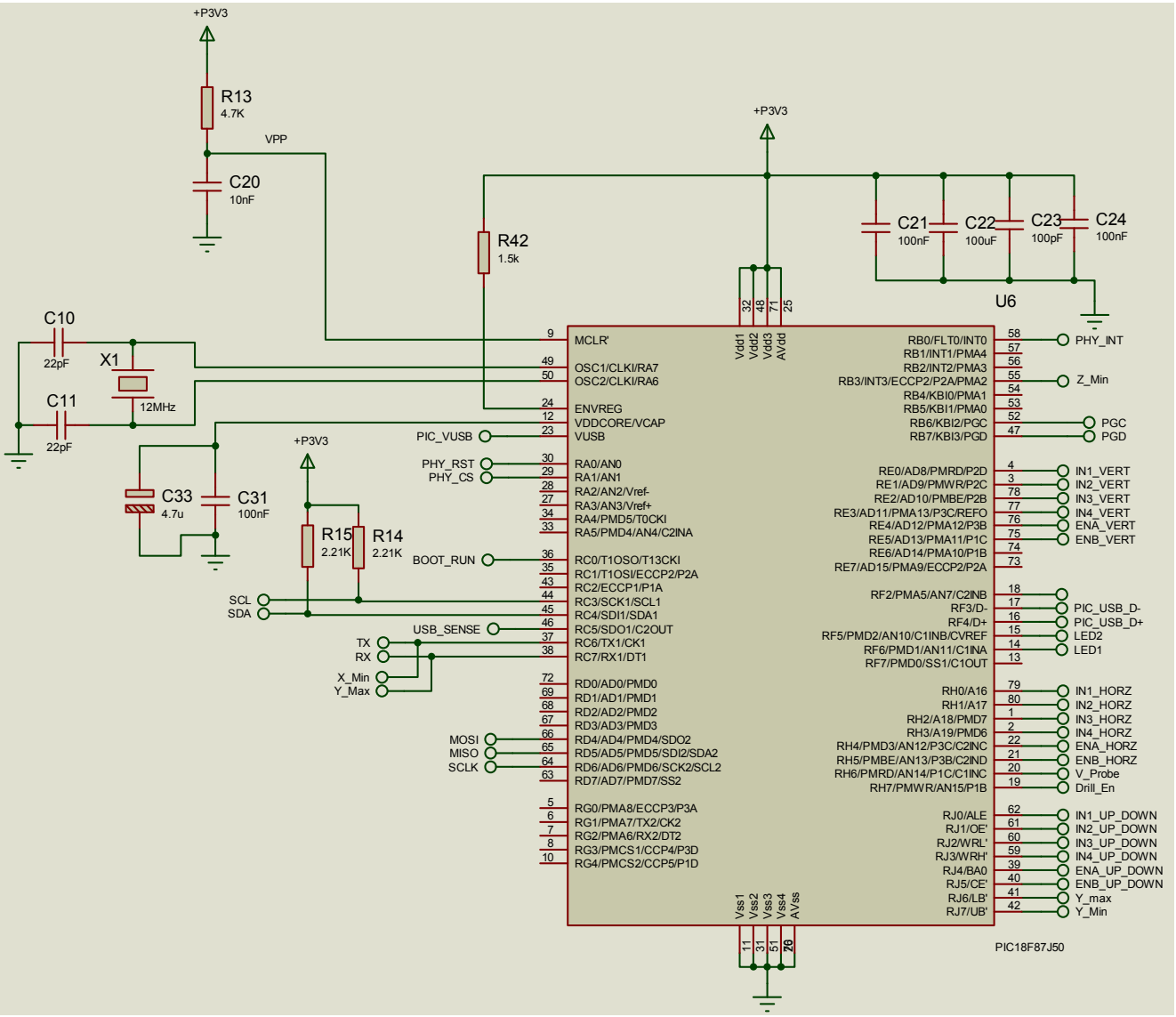


Figure 8- Circuit du microcontrôleur

L'oscillateur X1 utilisé est de fréquence égale à 12MHz. L'utilisation de l'interface USB dans le projet nous oblige d'utiliser cette fréquence (multiple de 4MHz). Pour plus d'information concernant les paramètres d'oscillateurs pour USB, et le choix des capacités C10 et C11, voir les Table 2-5 et 2-3 respectivement de la documentation du PIC18F87J50 (Annexe A).

La broche ENVREG (Enable for **On-Chip Voltage regulator**) mis à 1 à travers une résistance pull-up, permet d'alimenter les blocs intérieurs du micro (nécessitant 2.5V) par VDD à 3.3V. Quand ce régulateur est activé, une capacité de filtrage doit être connectée sur la broche VDDCORE/VCAP. C'est pour la stabilité du régulateur. La valeur de la capacité sur cette broche est recommandée dans la table 28-4 (Annexe A) de la documentation (de 4.7 à 10µF).

Les broches PGC et PGD (fig 9) sont utilisés durant la programmation du micro, en chargeant le fichier hexadécimale à travers la technique ISP (In-System Programming).

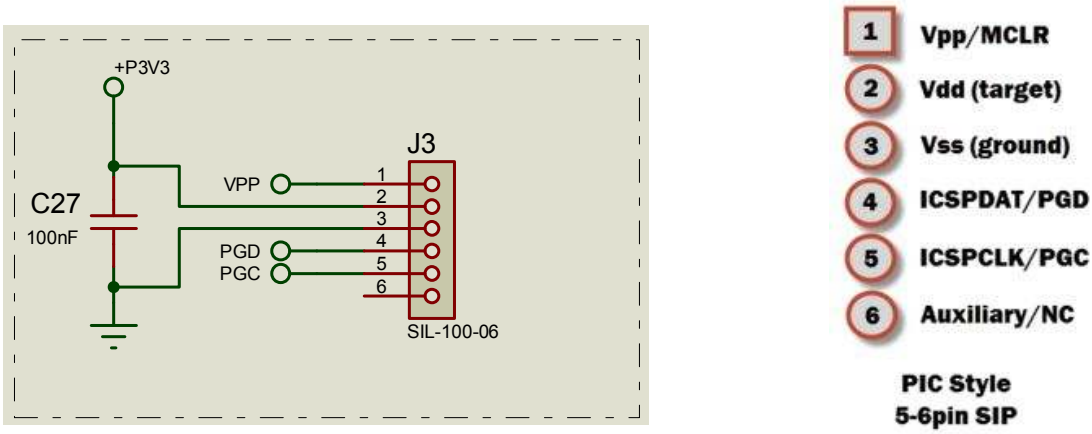


Figure 9- Broches de programmation du pic

Broche #1 du connecteur de programmation VPP : Programming mode voltage, il doit être connecté sur la broche MCLR du micro.

L'ICSP utilisé dans ce projet est le PicKit2 (Programmateur Microchip) (Annexe C), la figure 10 présente le câblage entre l'ordinateur et le microcontrôleur à programmer.

Figure 10- Méthode de programmation de Pic

2.4- Boot Loader

C'est une méthode qui permet de charger les microcontrôleurs par le fichier hexadécimale à travers un câble USB (figure 11) au lieu d'utilisation de la programmation série à travers l'ICSP. Mais cette méthode est seulement utilisée après le chargement du fichier principal et qui contient le code Bootloader qui est expliqué dans le chapitre suivant. Le circuit de sélection du bootloader est indiqué par la figure 12.

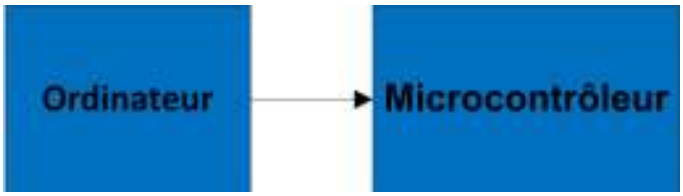


Figure 11- Méthode de programmation_Bootloader

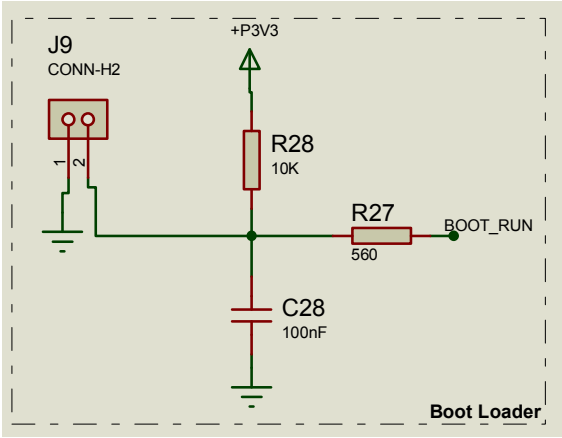


Figure 12- Circuit de sélection pour le bootloader

Le cavalier J9 de la figure 13 a pour but d'inverser l'état de la broche BOOT_Run du micro du niveau haut au niveau bas. C'est un indicateur qui permet de programmer le micro par un nouveau fichier hexadécimale, sans modifier le premier.

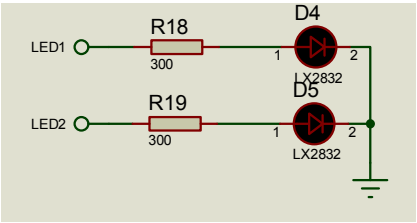


Figure 13- Indicateurs pour bootloader

Le clignotement des deux diodes LEDs (fig 14) indique si le microcontrôleur est en mode boot ou non.

2.5 Les drivers des moteurs

Il existe plusieurs types de drivers pour les moteurs pas-à-pas, comme L293, L297, L298 (voir annexe B). , etc. La sélection du driver utilisé, un L298N a été déjà expliquée dans le premier chapitre.

On a trois circuits comme celle indiqué dans la figure ci-dessous (fig 14). C'est pour les axes X, Y et Z. Les entrées de ce Dual full-bridge IC In1_VERT, In2_VERT, In3_VERT, In4_VERT, ENA_VERT et ENB_VERT sont connectés au microcontrôleur pour suivre des séquences binaires pour le fonctionnement soit vers la gauche soit vers le droit (Les séquences seront expliquées plus tard dans le chapitre suivant).

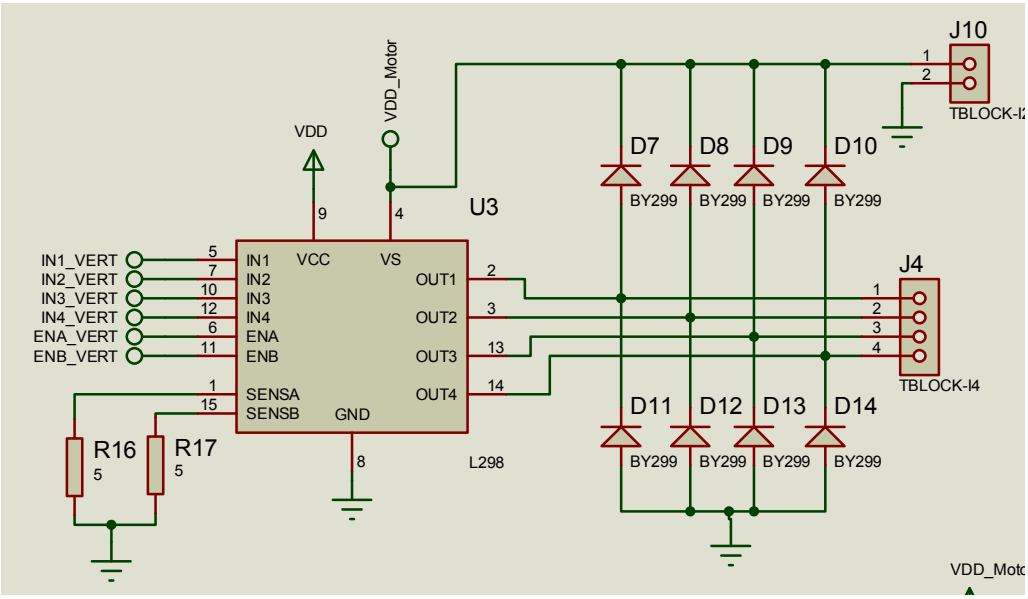


Figure 14- Circuit du Driver pour moteur

Les sorties doivent être connectés aux bobines des moteurs pas-à-pas, en utilisant les diodes de recirculation (roue libre) pour permettre à des trajets de courants vers les enroulements du moteur. Ces diodes doivent être « Fast Recovery Diodes » avec un temps de récupération inversé plus petit que 200ns pour minimiser le bruit de commutation durant les changements de pas (step).

Les résistances R16 et R17 sont utilisés comme résistances de sensibilité, pour contrôler le courant de la charge. C'est pour protéger contre une surintensité en commutant au niveau bas l'entrée EN.

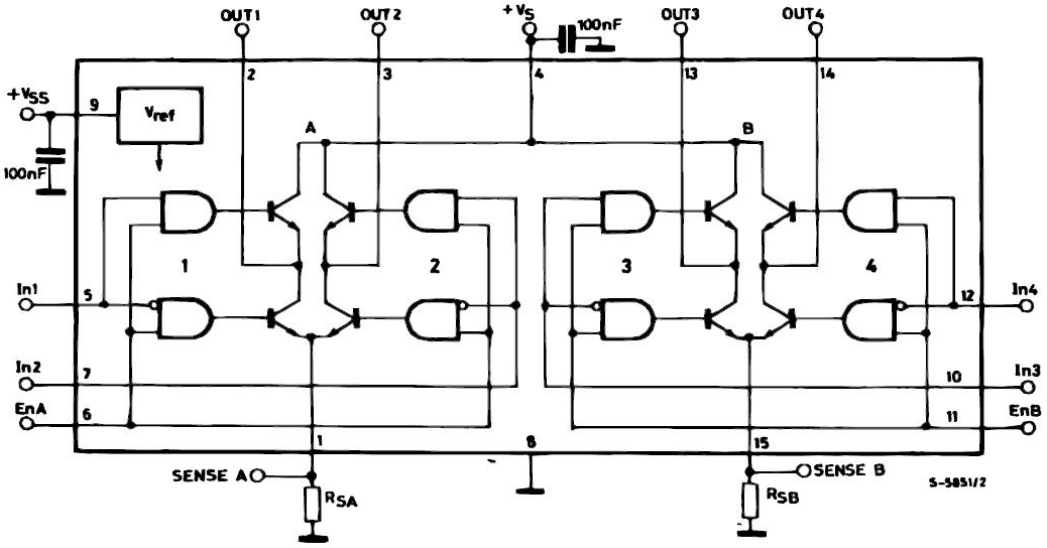


Figure 15- Circuit détaillé pour L298N

La figure 15 est le schéma électronique du driver L298N, qui est composée de portes logiques et de transistors d'amplification.

2.6- Différents modes de fonctionnement du driver L298N

Le mode de fonctionnement du driver dépend des impulsions venant du microcontrôleur, on a trois modes de fonctionnement qui sont les suivantes :

1- Half step mode : La résolution dans ce cas sera doublée, alors au lieu de tourner le moteur pas-à-pas avec une résolution de 1.8 degré, on peut le tourner avec une résolution de 0.9 degré. Le chronogramme (fig 16) montre la séquence venant du microcontrôleur pour faire fonctionner le moteur en demi-pas.

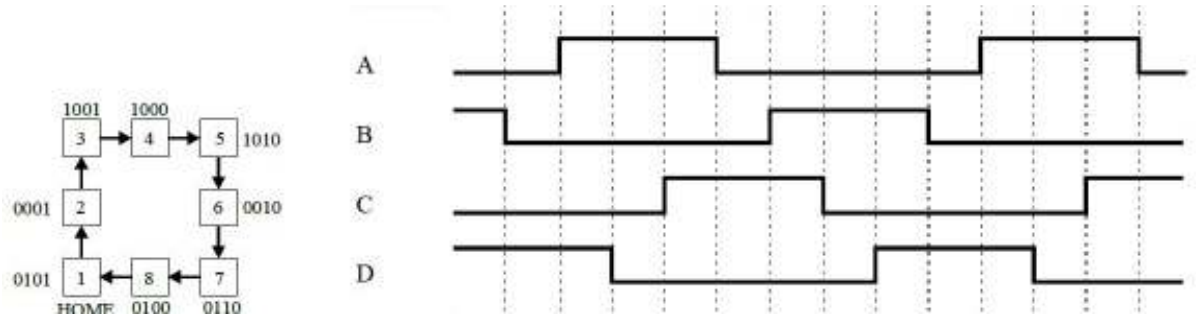


Figure 16- Séquence de données pour le driver en demi-pas

2- « Une phase en Full step » on a seulement quatre séquences qui seront répétés au lieu des huit utilisés dans le premier mode. (Voir fig 17).

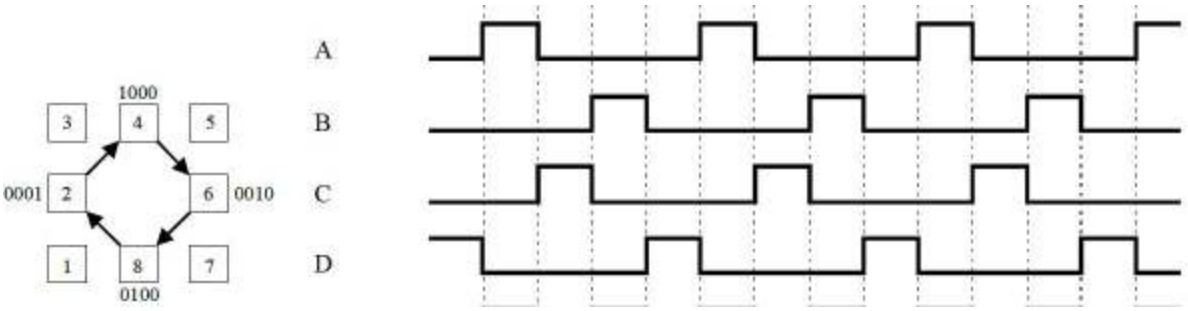


Figure 17- Séquence de données pour le driver-une seule phase

3- « Deux phases en Full step »: comme celui utilisé dans ce projet on applique la séquence suivante (fig 18):

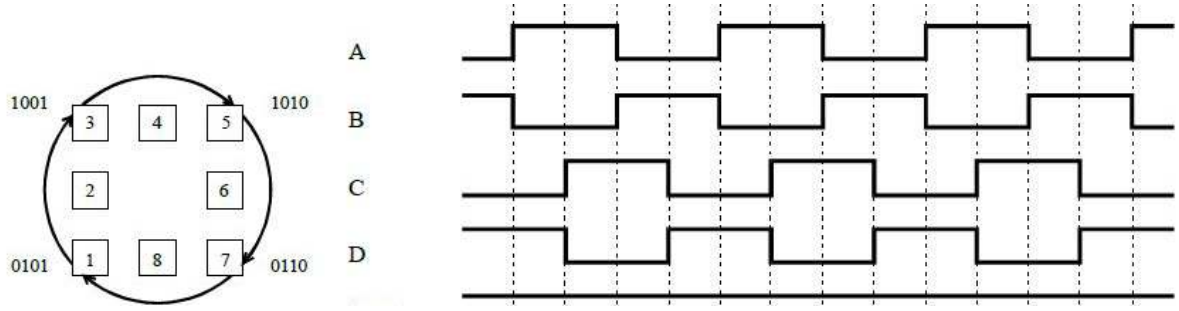


Figure 18- Séquence de données pour le driver- deux phases

On note finalement que ces séquences (fig. 16 à 18) correspondent pour des déplacements de quatre pas (step). Eventuellement, il faut répéter ces séquences pour atteindre le nombre voulu de pas.

2.7- Capteurs de fin de course

Les capteurs de fin de course sont utilisés pour améliorer la sécurité en cas de dépassement des limites logicielles sur les axes dans les systèmes CNC, afin de protéger contre les dégâts matérielles en cas de problèmes matériels ou logiciels.

Dans notre système, on a installé quatre capteurs de fin de course, deux sur chacun des axes X et Y. Quand un mouvement selon un axe donné dépasse sa limite logicielle, le capteur de fin de course active un relais qui, à son tour, provoque la coupure de l'alimentation des moteurs comme le présente la figure 20. Une intervention manuelle du personnel de service est alors nécessaire pour sortir de cet état.

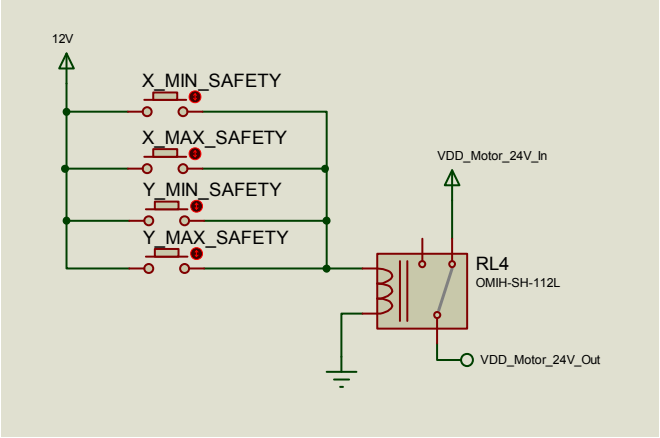


Figure 19- Interrupteurs fin de course

2.8- Search Run

À la mise sous tension de notre testeur à sonde mobile, il effectue un« Search Run » afin d'initialiser les positions des axes. La figure 20 montre les signaux X_Init et Yinit qui relient le micro aux phototransistors pour indiquer la position initiale de la machine pour les axes X et Y.

Ce type de détecteurs est préférable aux détecteurs mécaniques (interrupteurs) puisqu'il est plus précis. La figure a ci-dessous indique que le transistor reste passant tant qu'il n'y a pas un obstacle entre l'émetteur et le récepteur. Si la lumière est arrêtée à l'aide d'un obstacle non transparent, le transistor devient bloqué.

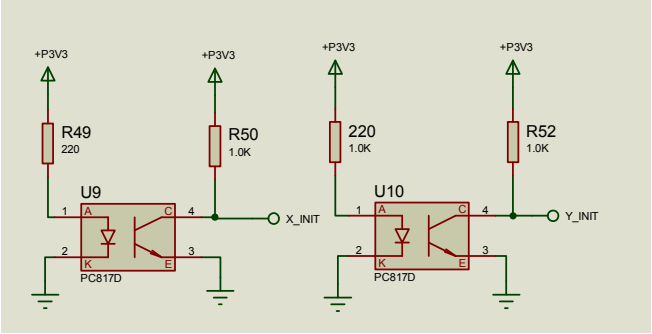


Figure 20- Interrupteurs pour Search Run

La figure 21 montre bien le fonctionnement du phototransistor. L'obstacle noir peut désactiver la sortie du transistor.

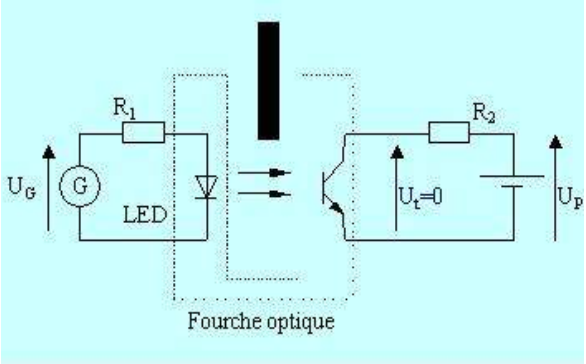


Figure 21- Circuit d'un phototransistor

2.9- Circuit de la sonde

Afin d'atteindre des tensions de mesure de 20 V, on a utilisé un diviseur de tension. En effet, le CAN du microcontrôleur utilisé (Pic18F87J50) supporte 3.5V au maximum selon sa documentation. Un circuit de diviseur de tension est utilisé dans ce cas comme indiqué dans la figure 23.

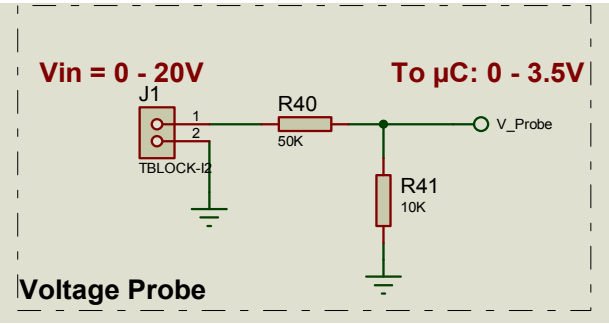


Figure 22- Circuit de la sonde

$$V_{probe}(V) = \frac{R_{41}}{R_{40} + R_{41}} V_{in} = 0.1666 V_{in} \quad \text{Eq. 4}$$

V_Probe a été connecté au microcontrôleur sur la broche AN14.

2.10- Circuits d'alimentation

Tout d'abord il faut commencer par noter quelles sont les niveaux de tension nécessaires dans ce projet, on a besoin pour deux niveaux de tensions qui sont 5V et 3.3V (3.3V pour le microcontrôleur et 5V pour les drivers des moteurs et autres ICs). Selon le datasheet du régulateur LM7805 [6] (Annexe D), l'alimentation principale a son entrée doit être entre 7.5 à 25V Le courant maximal à la sortie est 1A, la température de jonction Tj de 0℃ à 125℃. La sortie du premier régulateur est 5V (fig 23) Ce régulateur peut donner un voltage fixe à la sortie, comme dans notre cas, ou il peut être réglable à travers un diviseur de tension.

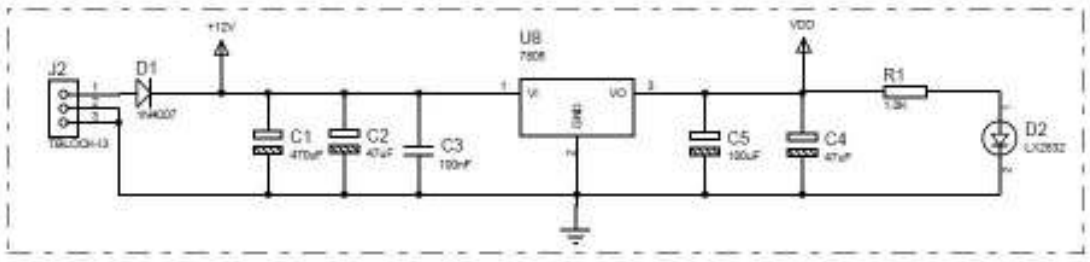


Figure 23- Circuit d'alimentation 1

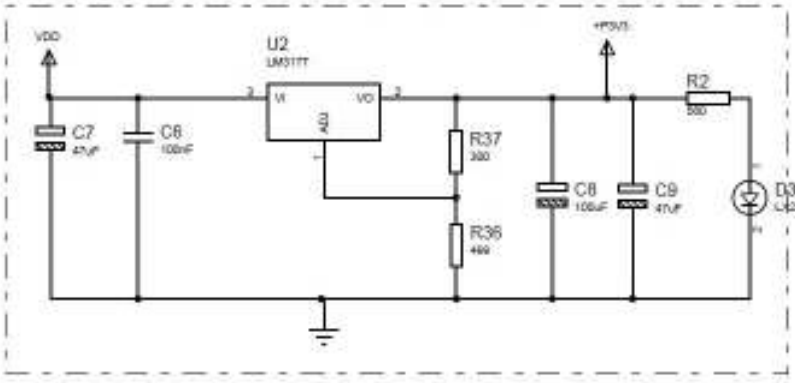


Figure 24- Circuit d'alimentation 2

Le deuxième régulateur LM317T (fig 24) est ajustable (Annexe E). Il fournit à la sortie une tension entre 1.2V et 37V avec un courant de 1.5A. L'ajustement se fait à travers des résistances avec une boucle fermée vers la broche #1 [7]. On a besoin 3.3Volt à la sortie pour alimenter le

microcontrôleur et les autres composantes. D'où la formule suivante permet de calculer la sortie désirée en fonction des valeurs de résistances R36 et R37 :

$$V_{out}(V) = 1.25(1 + \frac{R_{36}}{R_{37}}) \quad \text{Eq. 5}$$

Pour $R36 = 499\,\Omega$ et$R37 = 300\,\Omega$ on obtient : $V_{out} = 3.3V$

Le rôle des diodes LED D2 et D3 est uniquement pour indiquer que les tensions aux sorties des régulateurs sont disponibles. D1 est pour la protection contre l'inversion de la polarité à l'entrée. (La diode 1N4007 peut supporter 1A au maximum, pour être compatible avec le régulateur LM7805).

2.11- Contrôle de la carte

Comme indiqué dans le schéma bloc, que la communication utilisée avec la carte est l'USB. Ce projet doit être contrôlé à travers une interface graphique sur le PC, pour cela il faut utiliser l'un de ses ports, tels que, le port parallèle, série, Ethernet, USB, etc. On a choisi le port USB qui permet de communiquer à travers la carte électronique puisqu'elle a besoin d'un circuit simple et la dimension du câble n'est pas trop grande.

2.12- Communication avec le PC

Pour la communication bidirectionnelle avec le PC, le protocole USB est utilisé. La connexion côté Pic18F87J50 est à travers les deux broches numéros 16 et 17 (D- et D+). Avant de donner les détails du protocole USB, on va présenter l'utilisation de cette interface de communication.

2.12.1- Interface USB

C'est une interface qui permet de connecter un dispositif à un ordinateur. Avec cette connexion, l'ordinateur permet d'émettre et de recevoir des données du dispositif. C'est une interface standard qui peut être utilisée pour plusieurs applications.

Durant les années, la spécification de l'USB a subi plusieurs révisions. On a commencé par USB 1.0 en 1996. La spécification originale supportait deux débits, faible débit (LS) de 1.5Mbps et débit complet (FS) de 12Mbps. Le faible débit est moins sensible aux interférences électromagnétiques (EMI). En 1998, on a développé l'USB 1.1 qui a apporté quelques améliorations par rapport à la spécification de l'USB 1.0. En 2000, c'est l'USB 2.0 qui est capable d'atteindre un débit de 480Mbps.

Le réseau USB est composé d'un hôte, qui est typiquement un ordinateur (PC) et un multiple de dispositifs périphériques connectés à travers une topologie d'étoile (architecture maitre-esclave) [8]. L'hôte contient deux composantes, « Host Controller » et le « root hub ». Le Host Controller est un chipset matériel avec la couche du driver pour le logiciel qui est responsable des tâches suivantes :

- Détection des périphériques USB
- Gérer le flux de données entre l'hôte et les périphériques
- Assurer et gérer l'alimentation pour les périphériques connectés
- Surveiller l'activité sur le bus

Le root hub se connecte au « host controller » et agit comme la première couche d'interface USB. Dans le PC il y a plusieurs ports USB, ces ports font partie du root hub. On peut considérer le root hub comme une boîte noir qu'on appelle l'hôte.

Il existe deux types de pipes (voie de connexions entre le contrôleur hôte et une mémoire adressable« endpoint») dans le système USB, le control pipes et le data pipes.

La spécification de la communication USB définit quatre différents types de transfert de données :

Control Transfers : Utilisé pour transmettre des commandes aux dispositifs, faire des enquêtes et configurer les dispositifs. Il utilise le control pipe.

Interrupt Transfers : Pour l'envoi de petites quantités de données (burst) qui nécessite un temps garanti delatence minimum. Ce transfert utilise le data pipe.

Bulk Transfers : Pour le transfert de données qui utilisent toute la bande passante disponible de l'USB sans aucune garantie de latence ou de la vitesse de transfert. Il utilise le data pipe.

Isochronous Transfers : utilisé pour les données qui nécessite un débit garanti. C'est pour garantir le temps de livraison. Ce transfert utilise le data pipe.

Chaque dispositif a un « control pipe ». A travers cette pipe, il permet de faire la transmission et la réception des messages. Le control pipe est bidirectionnel, tandis que les data pipes sont unidirectionnelles.

Chaque endpoint est accédé par l'adresse du dispositif (attribué par l'hôte) et par le numéro de l'endpoint (attribué par le dispositif). Quand l'information est émise vers le dispositif, l'adresse de ce dernier et le numéro de l'endpoint sont identifiés dans le paquet.

Premièrement, quand un dispositif USB est connecté à l'hôte, la procédure d'énumération de l'USB est initialisée. L'énumération est la procédure d'échange de l'information entre le dispositif et l'hôte (fig 25). Elle contient aussi l'attribution de l'adresse du dispositif, reading descriptors (qui sont les structures des données pour fournir l'information concernant le dispositif), l'attribution et le chargement du driver du dispositif.

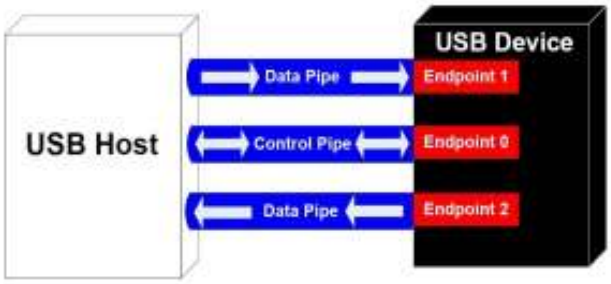


Figure 25- Flux de données entre l'hôte et le dispositif

La séquence d'énumération des évènements est décrite dans la figure 26.

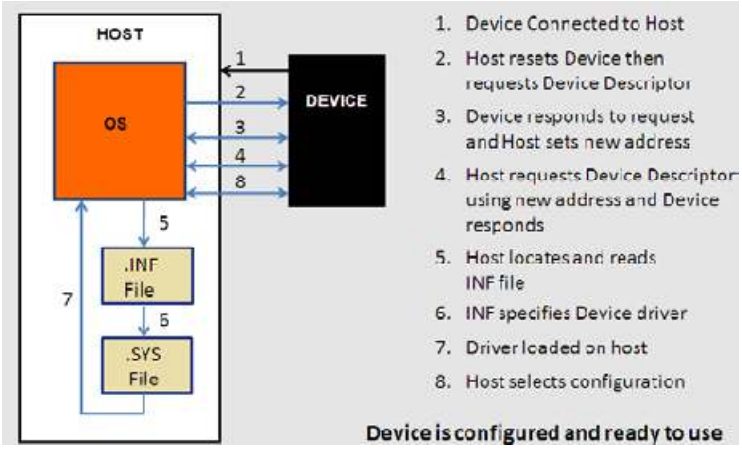


Figure 26- Séquence des évènements entre l'hôte et le dispositif

.SYS : Le driver besoin de communiquer effectivement avec le dispositif USB.

.INF : un fichier texte qui contient toutes les informations nécessaires pour installer un périphérique. Tels que, les noms des drivers et ses lieux, Windows registry information et l'information de la version du driver.

Après l'énumération du dispositif, l'hôte dirige tous les flux de trafic vers les périphériques sur le bus. Pour cette raison, aucun dispositif ne peut transférer des données sans la demande du host controller.

2.12.2- Interface physique de l'USB [9]

L'interface physique de l'USB à deux composantes, ce sont le câble et le connecteur, pour la connexion entre l'hôte et le dispositif.

A l'intérieur de la gaine extérieur du câble, il y a un multiple de fils qui sont : Deux fils d'alimentation VBUS (Rouge) 5V et masse (noir) et une paire torsadée de fils destinée au transfert de données D+(Vert) et D- (blanc), comme indiqué dans la figure 27.

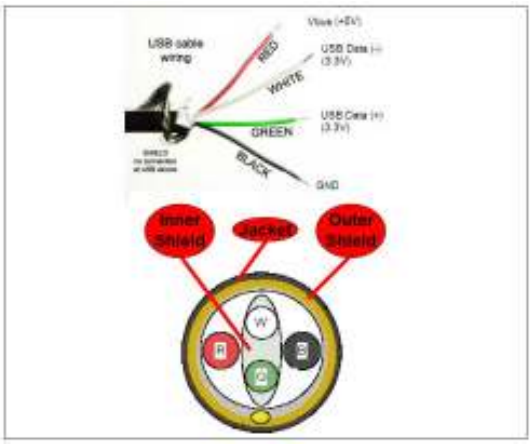


Figure 27- Le câble USB

Pour les dispositifs qui ont un débit élevé, la longueur maximale du câble doit être 5 mètres.

Les lignes de données (D- et D+) fonctionnent avec un niveau de 3.3V. L'interface USB utilisa la transmission différentielle qui est NRZI (Non-Return-to-Zero Inverted) qui est codé avec de bit de bourrage (Bit stuffing) à travers le pair torsadée.

Le codage NRZI [10] est une méthode utilisée dans la transmission pour le mappage d'un signal binaire. Le principe de base de NRZI est le changement d'état du signal pour chaque « 0 » et pas de changement d'état pour les « 1 ». Mais le niveau bas de ce code est –V au lieu de 0.

Comme indiqué dans la figure 29, les données qui doivent être transmises à travers le codage NRZI sans le bit de bourrage.

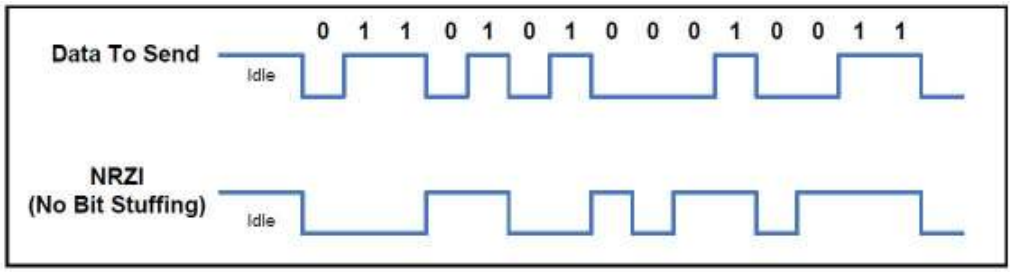


Figure 28- Codage NRZI sans bit de bourrage

Le bit de bourrage (bit stuffing) eu lieu en insérant un bit « 0 » après six « 1 » consécutifs, pour forcer une transition dans le code NRZI. Le but est pour la synchronisation du matériel de l'USB en maintenant le PLL, c'est pour éviter les pertes de données. S'il y a beaucoup de « 1 » dans la donnée, on n'aura pas beaucoup de transition dans le flux de données. Le récepteur dans le matériel de l'USB détecte automatiquement ce bit supplémentaire et le rejet pour décoder les données convenables. La figure 30 montre un exemple d'émission d'un code NRZI avec un bit de bourrage.

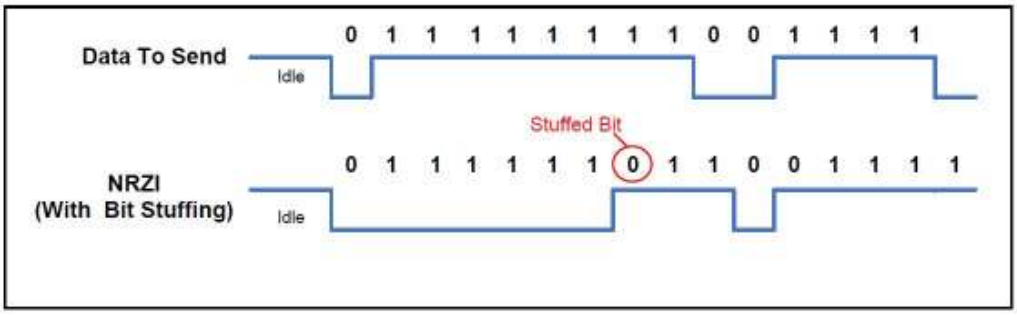


Figure 29- Codage NRZI avec bit de bourrage

Ce type de codage est uniquement utilisé pour le transport à travers le cordon USB.

Le raison pour l'utilisation deux lignes de transmissions D- et D+ est pour rejeter le mode commun du bruit. Si le bruit devient couplé dans le câble, il sera normalement présent sur tous les fils du câble. Avec l'utilisation d'un amplificateur différentiel dans le matériel intérieur de l'USB, sur le côté de l'hôte et sur l'autre côté du dispositif. Le mode commun du bruit sera rejeté dans ce cas voir figure 30.

La communication sera avec les différents états de signalisation sur D- et D+. La transmission des données sera dans quelque état, et les autres pour des conditions de signalisation spécifique.

Dans ce projet, on a utilisé l'USB de type mini-B. La dernière version est celle d'USB 3.0, avec un débit de transfert de données arrivant à 3Gbps.

Concernant l'alimentation de l'USB, On a deux catégories :

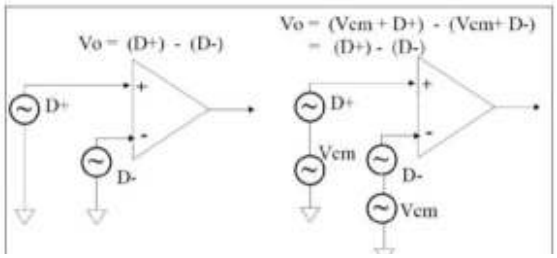


Figure 30- Amplificateur différentiel pour les données de l'USB

Alimentation par le bus : Sans aucune source DC extérieur ou intérieure, c'est de VBus de l'hôte. Avant l'utilisation de cette méthode, il faut prendre en considération la consommation du courant. Avant la configuration il ne faut pas consommer plus que 100mA. Le courant maximal qu'on peut retirer de l'hôte doit être plus petit que 500mA, si non, il faut utiliser la deuxième catégorie.

Auto alimentation : utilisation d'une source extérieure.

Aucune fonction ne peut consommer plus que 100mA sur le bus avant d'être énumérée. Après l'énumération, un dispositif peut consommer jusqu'à 500mA pour un dispositif « High power device » ou rester à 100mA pour un « Low power device ».

La figure 31 suivante présente les différents types de ports et de connecteurs USB (fig 31).

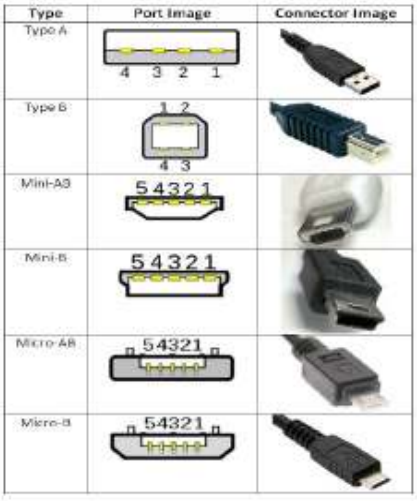


Figure 31- Types des connecteurs USB

2.13- Protocol USB

Ce protocole est comme tous les autres protocoles, c'est le protocole à encapsulation. Mais avant d'expliquer le format du paquet, on va expliquer brièvement ce protocole en se basant sur la figure 32.

Le client Driver communique les demandes de transfert des applications via le paquet d'entrée/sortie (I/O packet). Après, le USB driver traduit chaque transfert en une série de Transactions. Ensuite l'USB host Controller Driver fait regrouper les transactions en des trames, enfin l'USB host Controller traduit les transactions en paquets et enchaine les trames.

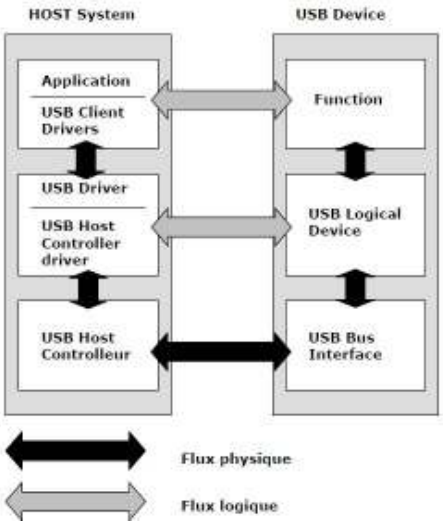


Figure 32- Protocole USB

2.13.1- Paquet USB

Contrairement à la liaison série RS232, où le format des données envoyées n'est pas défini. L'USB est composé de plusieurs couches de protocoles. La couche inférieure est importante pour les circuits intégrés, il faut la rendre invisible au regard du concepteur final.

On a quatre types de paquet dans l'USB :

- En-tête (Token)
- SOF (Start Of Frame)
- Data (Optionnel)
- Acknowledge (Hanshake)

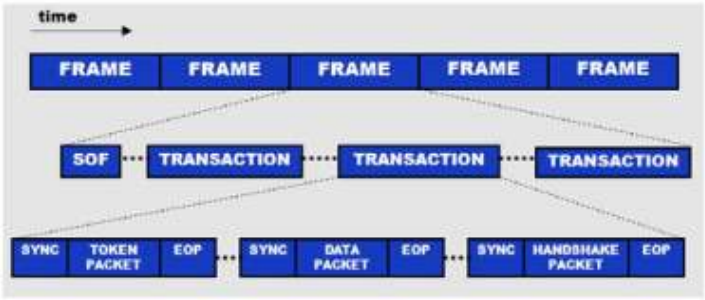


Figure 33- Communication USB à partir d'une perspective temporelle

La figure 33 montre un exemple de la communication pour une perspective temporelle

Les paquets (Token) jeton indiquent le type de la transaction qui doit suivre, dont le but est de transporter l'adresse USB et le sens du transfert.

Les données utiles se trouvent dans les paquets de données (data). Les paquets (Handshake) sont utilisés pour indiquer s'il y a des erreurs ou pour la validation des données. Les paquets de (SOF) début de la trame indiquent le début de la nouvelle trame.

L'entité de transfert USB est appelé **Transaction,** elle contienne des paquets juxtaposés. Quand une SOF aura lieu, c'est-à-dire qu'on a une succession de transactions, voire figure 34.

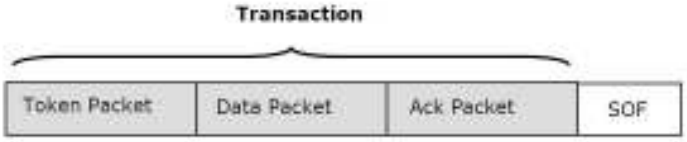


Figure 34- Transaction USB

Tous les paquets ont la structure suivante :

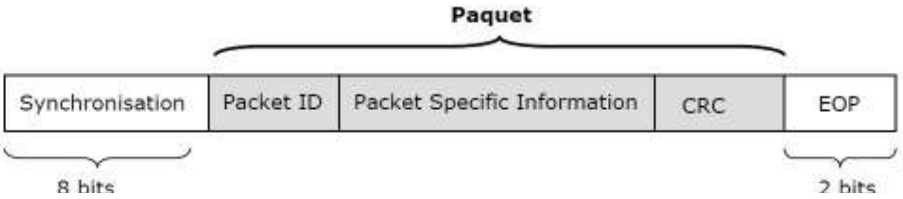


Figure 35- Structure des paquets USB

Le format se diffère d'un paquet à l'autre, alors selon la nature du paquet. Chaque trame à une durée de 1ms, un SOF est alors transmis chaque 125us, et le compteur de trame est incrémenté chaque 1ms. Avec la technique d'encapsulation les transactions sont regroupées à l'intérieur des trames sans aucun chevauchement. Une trame peut contenir une ou plusieurs transactions envoyées à un même périphérique.

On a trois types de paquets jetons (Token) (figure 36) :

- In : Informe le dispositif USB que l'hôte veut lire les informations.
- Out : Informe le dispositif USB que l'hôte veut envoyer les informations.
- Setup : Pour commencer les transferts de commande.

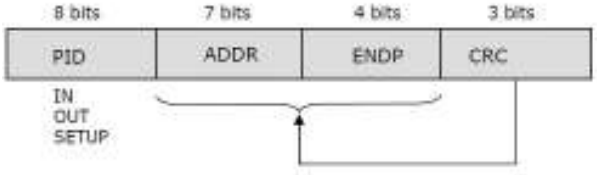


Figure 36- Format du paquet Token

La taille de données dépend de la vitesse de transfert. Pour Low speed on a 8 octets, pour Full speed 64 octets et pour High speed 1024 octets (fig 37). Les données doivent être envoyées en multiple d'octets.

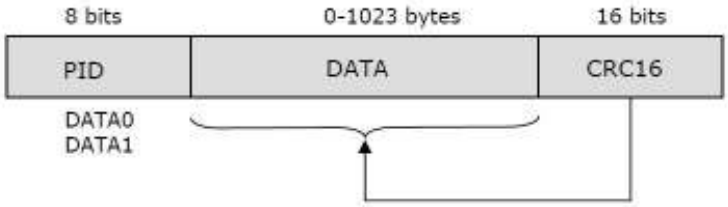


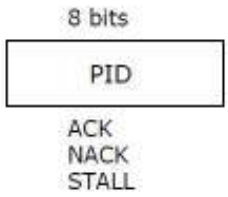
Figure 37- Format du paquet Data

Les paquets Handshake décide chaque transaction. Chaque Handshake contient 8-bit Packet ID (Annexe F), et il est envoyé par le récepteur de transaction et sont aussi divisés en trois sortes (fig xx):

Ack : Pour valider la réception correcte du paquet (LS/FS/HS)

NAK : Negative Ack Pour indiquer que le dispositif ne peut ni envoyer ou recevoir des données pour une petite période. Utilisé aussi durant les transactions d'interruptions pour l'avertissement de l'hôte qu'il n y a pas de données à envoyer.

STALL : Etat d'interruption de l'hôte exigé par le dispositif.



Enfin, on a les paquets SOF (fig 38) qui sont composés de 11 bits trame envoyé par l'hôte chaque 1ms ± 500ns sur un bus de vitesse FS ou chaque 125us ± 0.0625µs sur un bus HS.

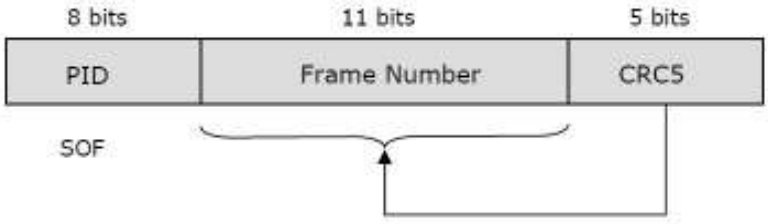


Figure 38- Format du paquet SOF

On a une grande fiabilité de transfert assurée par le standard USB. C'est pour la détection des erreurs au biveau matériel (Hardware), comme celle-ci-dessous :

Erreur de paquets (PID, Bit stuff, CRC,..), Time out (absence de réponse), Loss of activity, etc.

Chaque détection d'une erreur de paquet implique un Time Out (non réponse du périphérique).

Pour plus d'information concernant l'USB, voir annexe.

2.14- Circuit imprimé (Layout)

Après la réalisation et la vérification de fonctionnement théorique du schéma électronique, l'étape suivante est de réaliser le circuit imprimé (PCB).

Tout d'abord il faut décider si on va utiliser des circuits à double face (Top and Bottom), ce qui permet de réduire les dimensions de la carte et facilite le câblage. Bien sûr que le prix aussi sera plus cher chez le fabriquant des PCB qui se trouve en Chine.

Il y a aussi la sélection du type des broches des composantes, entre le « Throughhole » (à travers des trous) et le SMD (Surface Mount Device) qui ne nécessite pas de trous sur la carte.

J'ai utilisé des cartes PCB double faces et des boitiers SMD pour la plupart des composants (microcontrôleur, résistances, condensateurs de filtrages, etc.), ce qui m'a permis aussi de minimiser les dimensions et de donner une forme agréable à la carte.

Le « Layout »du PCB a été réalisé avec le logiciel Proteus. Si les footprints des composants ne sont pas disponibles, il faut les dessiner en se basant sur les datasheets qui contiennent toutes les informations concernant les dimensions. Ces informations se trouvent souvent à la fin de datasheet. Dans mon cas, j'ai dessiné le footprint du microcontrôleur et quelques autres composants non définis.

On note cependant qu'il faut en générale :

- Faire attention aux sections sur la carte surtout à celles de l'alimentation principale sur lesquelles le courant peut arriver à quelques ampères.
- Bien distribuer les composantes sur la carte ce qui facilite l'étape de câblage.
- Ne pas mettre le microcontrôleur proche des circuits d'alimentation (régulateurs), parce que leurs échauffement influence sur son fonctionnement.

- Mettre tous les connecteurs d'entrées et de sorties sur aux bords de la carte pour facilite le câblage.

Et plus particulièrement pour notre carte :

- Faire attention aux lignes de données différentielles de la connexion USB qui doivent avoir une distance de câblage presque égale du connecteur USB au microcontrôleur.
- Prendre en considération le positionnement des dissipateurs thermiques (heatsink) des drivers L298N.

Chapitre 3

Algorithmes et programmation du HW et du SW

Dans ce chapitre on va présenter trois parties de programmation, qui fonctionnent tous ensemble pour contrôler la machine :

- Programmation de la partie HW, c'est ce qu'on appelle Firmware, qui concerne la programmation du microcontrôleur.
- Programmation de l'API (Application Programming Interface) : qui est l'interface intermédiaire entre la partie matérielle et la partie logicielle.
- Programmation de l'interface graphique GUI (Graphical User Interface) qui permet de contrôler la machine par l'intermédiaire de l'API.

Le lien entre ces trois parties est comme le suivant : (fig 39)

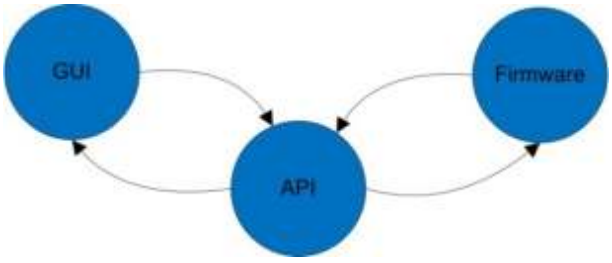


Figure 39- Lien du système logiciel complet

3.1- Firmware

3.1.1- Introduction

Depuis longtemps, le besoin de miniaturisation et de réduction des couts ont poussé les fabricants des composantes électroniques à intégrer au maximum les structures électroniques donnant naissance aux Circuits Intégrés (C.I.). Il faut reconnaitre que chaque jour de nouveaux pas sont franchis dans la miniaturisation. Un transistor est intégré sur une surface de $0.1\mu m^2$ à l'heure actuelle [11]. Lorsqu'on doit lire ou écrire un programme en langage machine, il est difficile d'utiliser la notation binaire ou hexadécimale.

On écrit les programmes à l'aide des instructions en mnémonique. Les concepteurs comme Intel, Microchip, etc., fournissent toujours une documentation avec les codes des instructions de leurs processeurs, et les symboles correspondantes.

Au début de l'invention des C.I. programmés, le programmeur était obligé de s'adapter au langage de chaque microprocesseur pour faire le contrôle désiré. Pendant les années 80 et 90, on a commencé à utiliser le langage C pour unifier le langage de programmation du microprocesseur. Ceci a eu lieu grâce à des compilateurs spécifiques qui transforment le code C en code machine.

3.1.2- Architecture du Firmware

L'architecture du code firmware est basée sur le circuit électronique du microcontrôleur présenté dans le chapitre précédant. Le but de ce code est de contrôler tous les autres circuits sur la carte st communiqué avec l'interface graphique à travers une connexion USB.

Afin de réaliser le contrôle désiré, le firmware comporte plusieurs fonctions :

- Les fonctions de contrôle des drivers des moteurs L298N,
- Les fonctions de communication à travers le bus USB,
- etc.

Ces fonctions peuvent être implémentées dans la même classe, ou dans des classes indépendantes (tout en étant dans un même projet). L'accès à ces fonctions est réalisé grâce à une fonction principale nommée main(). Elle contient la principale boucle du firmware qui a pour rôle de gérer le paquet USB. Elle extrait l'information contenue dans ce paquet, pour l'identifier et l'exécuter.

Le code firmware contient aussi des librairies qui sont essentielles pour la communication à travers le bus USB, ou pour stocker des données dans l'EEPROM extérieur. Il ya aussi un autre type de classes qui doit être utilisée ; elles correspondent au « system Service », « Hardware Profile » et fichier header « USBData » qui sont prédéfinis au préalable par l'entreprise Microchip. On a aussi les fichiers headers qui définissent l'interface du microcontrôleur par rapport à sa connexion dans la carte et les structures utilisées dans la programmation du firmware.

3.1.3- Fonctionnement du firmware

Le mode de fonctionnement du firmware correspond à une fonction principale qui exécute une boucle infinie. Dans cette boucle on a des conditions avant d'émettre et de recevoir des paquets à travers le protocole USB. La communication à base d'un fichier API (extension « .dll »), qui joue le rôle de liaisons entre interface graphique et le code firmware (sera expliqué en détail dans les paragraphes suivants).

Au début du code, on a prédéfini quelques fichiers header qui sont inclues dans le code (que j'ai écrit ou qui sont prédéfinis dans la librairie du compilateur). On cite les fichiers header suivantes :

- #**include "Prob_machine.h"** : pour définir la résolution du CAN, les FUSES de configuration du microcontrôleur, l'horloge et les registres pour chaque PORT (d'entré ou de sortie). Ce fichier contient aussi **#include "usb_bootloader.h"** pour définir la zone du code « Bootloader » (de 0x0000 à 0x0FFF), afin de ne pas y écrire. On a aussi dans ce fichier header **include <18F87J50.h>**, qui est le fichier Header standard pour le PIC18F87J50 afin de définir toutes ses broches, registres et variables de configuration. [12].

- **#include <pic18_usb.h>** : couche matériel (vitesse de transfert de données, longueur du paquet, ...) pour la librairie de l'USB qu'utilise le compilateur CCS. Ce fichier contient aussi **#include <usb_hw_layer.h>** qui contient la déclaration des fonctions utilisées dans la fonction main principale du code et dans l'API.

- **#include "usb_desc.h"**, « *Configuration Descriptor* »:Ce fichier contient la configuration, les définitions, et les endpoint. Il définit les« *Device Descriptor* » (Product ID:PID et Vendor ID:VID) par USB_DESC_STRING_TYPE,de façon que « ***Probing Machine*** »sera affiché dans l'interface du« device manager » du PC. Il contient aussi **#include <usb.h>** qui définit globalement le USB driver et les fonctions prototypes.

- **#include <usb.c>**, *Standard USB request and token handler code*: Ce code est utilisé quand une interruption est générée par la couche matérielle.

- **#include "USBDataF.h"** : Ce fichier contient l'énumération des fonctions principales (Données à l'intérieur du paquet USB), qui sont communes avec le code API. Ce fichier header est déclaré dans la fonction principale main().

Après il faut déclarer et définir les noms qui appartiennent au bit du port (d'E/S) correspondant (Pour faciliter la lecture du code pour le programmateur):

#define X_Min PIN_C6

Puis les fonctions des moteurs qui permettent de les faire fonctionner selon une direction et une vitesse donnée.

Il y a aussi les déclarations et les définitions des broches du microcontrôleur qui utilisés comme des ports d'entrées ou de sorties, comme :

PORTA = 0x00;

set_tris_a(0x18) ; //0b00011000: seulement RA3 et RA4 sont des entrées et les autres sont des sorties.

On arrive ensuite à la fonction principale ***void main()*** :

Il faut au début déclarer les variables comme

- « unsigned int8 IOBuffer[128] » pour la longueur du paquet USB
- «int16 count,Packet_length = 128»
- Etc…

Après il faut initialiser :

- les ports par InitPorts()
- le CAN
- etc.

Maintenant il faut commencer à communiquer avec le bus USB :

Au début il faut tester la connexion par **usb_init_cs()** qui permet d'appeler **usb_task()** pour tester la broche **USB_Sense**. Si cette condition est vraie, c.à.d. que les USB est connecté et le périphérique de l'USB n'est pas attaché, ce qui permet à attacher ce périphérique, alors le PC débuter le processus d'énumération. Après on a le test d'énumération, si il est vrai le bus est prêt pour émettre et recevoir des paquets, sinon : D*o not try to use the USB peripheral for sending and receiving packets until you are enumerated*.

Il nous reste un seule test qui se répète toujours à chaque action, c'est le **usb_kbhit(endpoint)**, s'il est vrai, alors on a de nouvelles données dans le buffer de réception RX.

Après la validation de toutes ces conditions, Il faut utiliser la fonction **usb_gets(1, IOBuffer, Packet_length, 100)** pour recevoir un paquet de données venant de l'hôte (le PC) et remplir le IOBuffer par les octets suivants:

CurrentUSBFunction = IOBuffer[0]; // get function

CurrentUSBCommand = IOBuffer[1]; // get command

DataLen = IOBuffer[2];

Du point de vue programmation, les paquets reçues sont stockés dans un tampon, qu'on appelle en anglais « Buffer » de longueur128 octets (128*int8). Les trois premiers octets correspondent respectivement à la fonction de l'USB, le type de commande et la longueur de données. Ces octets correspondent à des structures définies dans le fichier header **USBDataF.h** du firmware :

La première fonction a pour nom USB_function, elle permet de choisir la nature de la tâche que le microcontrôleur doit exécuter, par exemple si on va communiquer avec les drivers des moteurs, on doit tout d'abord spécifier l'interface de communication qui est dans ce cas le protocole USB, (elle peut être aussi I2C, SPI ou MDIO). Un autre exemple, est qu'on peut accéder à tous les bits des ports du micro, en sélectionnant USB_directIO.

Le deuxième octet du buffer est le type de la tâche correspondant à la lecture ou l'écriture.

Le troisième est seulement la longueur du paquet qui est défini préalablement dans le code firmware et dans l'API.

Le code suivant correspond à la définition de l'énumération USB_function dans le micro et le fichier API :

```
enum USB_Function
{
        USB_DirectIO = 0xF0,
        Move_Motor = 0xF1,
        USB_Power,
        USB_Analog,
        USB_Verification,
        USB_I2C,
        USB_Device,
} CurrentUSBFunction;
```

Comme un exemple, USB_Verification a pour rôle d'indiquer dans le code main du firmware que les moteurs ont terminés leurs tâches ou non.

L'énumération USB_directIOTarget ci-dessous est utilisée pour déterminer le niveau logique des ports du micro. Le changement sera à l'intérieur de la fonction main (PORTA = IOBuffer[4]; venue du paquet USB).

```
enum USB_DirectIOTarget {          IO_PA = 0x00,  // PORTs A ... J

                                   IO_PB,
                                   IO_PC,
                                   IO_PD,
                                   IO_PE,
                                    …
                                   IO_ADC,   // Internal ADCs
                            } CurrentUSBDirectIOTarget;
```

L'énumération ci-dessous correspond au deuxième octet du buffer qui spécifie la commande :

```
enum USB_Command {
                                   USB_Write = 0x00,
                                   USB_Read,
                                    USB_TBD
                             } CurrentUSBCommand;
```

Le quatrième octet correspond aux données envoyées ou reçues. On a aussi deux énumération pour cet octet, USB_analogCommand et USB_power Command, qui permettent respectivement d'envoyer la valeur moyenne de la tension mesurée à un point de test, et de recevoir une commande pour activer la perceuse selon l'axe z (option non implémentée qui permet d'utiliser notre système comme « Drilling machine » pour les cartes électroniques).

```
enum USB_analogCommand {
                                   Analog_VCC_Prob_sens = 0x00,
                                   Analog_Current_sens,
                                   Analog_Temp_sens
                             } CurrentUSBanalogCommand;

enum USB_powerCommand {
                                   Power_Drill = 0x00,
                                   Power_Fan,
                                   Power_PWM_CNTL
                             } CurrentUSBpowerCommand;
```

3.1.4- Organigramme du firmware

Maintenant on récapitule notre présentation précédente dans l'organigramme présenté par la figure 40.

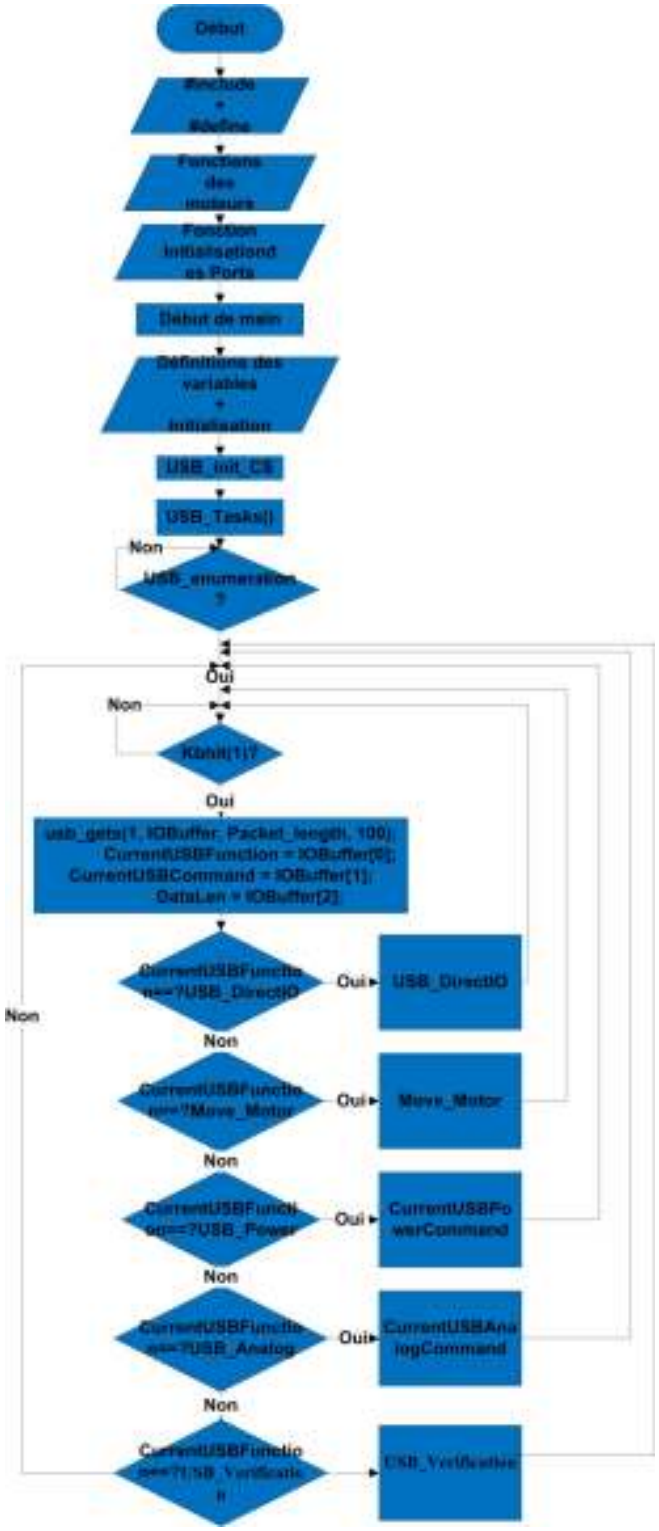


Figure 40- Algorithme pour la fonction main du code firmware

3.1.5- Fonctions des moteurs

Il y a14 fonctions pour la gestion des moteurs. Le nombre de ces fonctions peut être réduit, mais c'est plus facile pour la mise à jour ultérieure du code. Ces fonctions sont les suivantes :

Index	Nom de la fonction	Utilisation
1	void Motor_X_Reverse(int32 count)	Directe
2	void Motor_X_Reverse_Max_speed(int32 count)	Appel par 11 ou 14
3	void Motor_X_Forward(int32 count)	Directe
4	void Motor_X_Forward_Max_speed(int32 count)	Appel par 12 ou 13
5	void Motor_Y_Reverse(int32 count)	Directe
6	void Motor_Y_Reverse_Max_speed(int32 count)	Appel par 11 ou 13
7	void Motor_Y_Forward(int32 count)	Directe
8	void Motor_Y_Forward_Max_speed(int32 count)	Appel par 12 ou 14
9	void Motor_Z_Reverse(int32 count)	Directe
10	void Motor_Z_Forward(int32 count)	Directe
11	void Motor_X_Y_Reverse(int32 countX,int32 countY)	Directe
12	void Motor_X_Y_Forward(int32 countX,int32 countY)	Directe
13	void Motor_X_Forward_Y_Reverse(int32 countX,int32 countY)	Directe
14	void Motor_X_Reverse_Y_Forward(int32 countX,int32 countY)	Directe

Table 4- Fonctions des moteurs

Certaines de ces fonctions peuvent être directement appelées ou d'autres sont appelées à travers une autre fonction de gestion des moteurs.

L'utilisation des fonctions à appel indirect aura lieu dans le cas où les deux axes travaillent ensemble (comme dans le cas des fonctions # 11, 12, 13 et 14) et que la distance à atteindre diffère d'un axe à l'autre.

On va présenter ci-dessous à titre d'exemple une seule fonction moteur (dans les deux directions), les autres fonctions sont similaires.

La figure 41 présente l'algorithme de la fonction

void Motor_Y_Forward_X_Reverse(int32 countX,int32 countY)

La tâche de cette fonction est de donner les impulsions nécessaires pour les drivers des moteurs selon les axes X et Y.

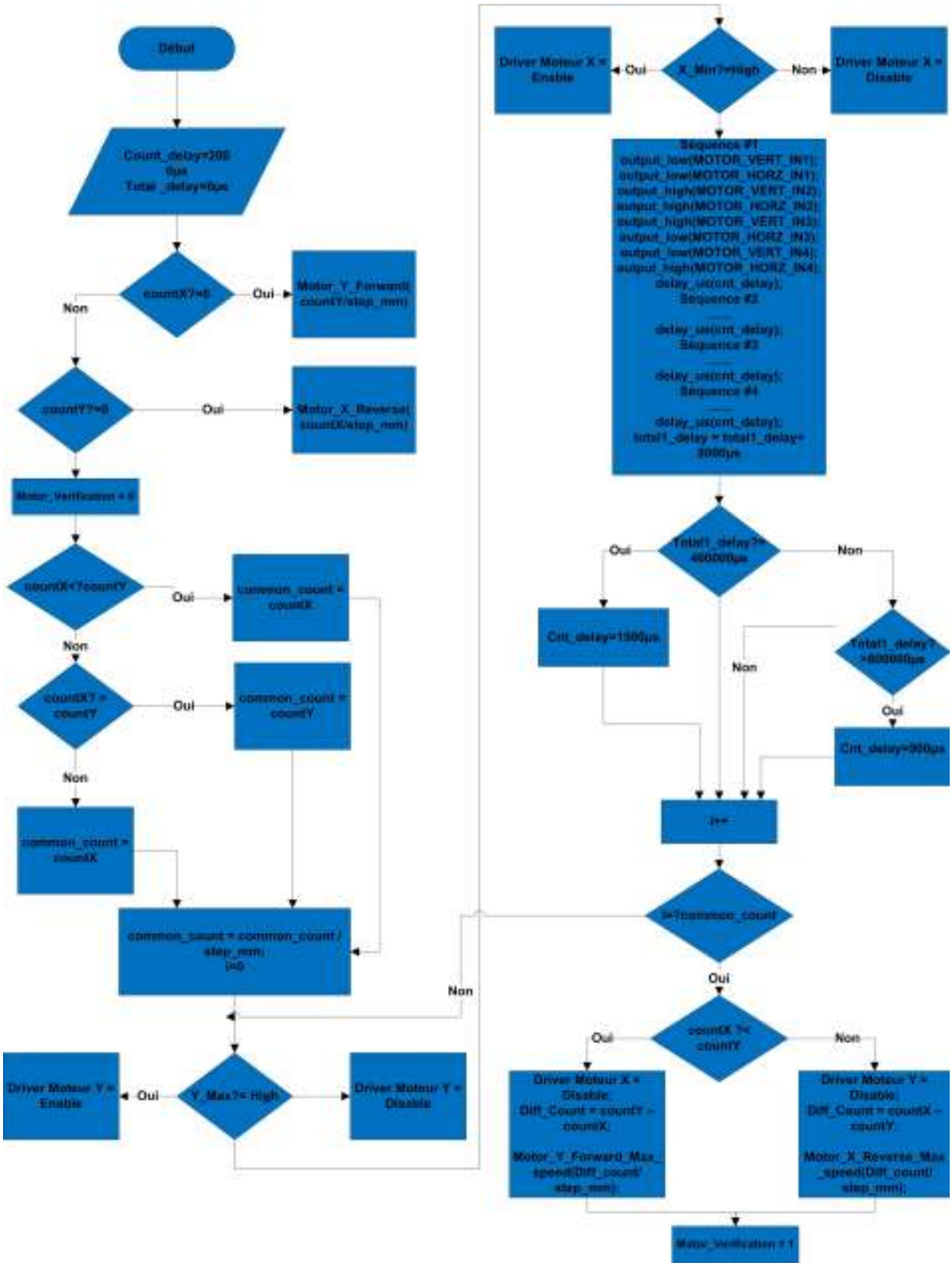


Figure 41- Algorithme void Motor_Y_Forward_X_Reverse

On commence par initialiser le delay_cnt à 2000µs, c'est le délai entre qui est le délai exécuter à aux drivers des moteurs, i.e. c'est la vitesse des moteurs pas-à-pas au démarrage. Après on teste si le nombre de pas (countX ou countY) est nul, c'est-à-dire qu'on a un seul moteur à activer. Dans ce cas on appelle une autre fonction qui est dédié à ce moteur. Après on met la variable Motor_verification à 0, ce qui permet d'indiquer si la tâche de la fonction est terminée ou non.

On cherche après le nombre de pas le plus petit selon X ou Y, pour faire l'exécution des impulsions en mode commun (common_counts). Avant l'exécution on teste si l'un des moteurs est à ses limites minimales ou maximales, si la condition est vraie, alors il on coupe l'alimentation du driver correspondant pour la protection du moteur et du système mécanique.

Maintenant on arrive à l'étape d'exécution des impulsions sur les différentes broches des drivers des deux moteurs. La commande des signaux numériques (Low and High) sera en alternance pour chaque driver. Le délai d'impulsion sera le même pour chaque moteur. On a quatre séquences qu'il faut exécuter et chaque séquence est pour commander le moteur afin d'exécuter un seul pas (step), comme il a été indiqué dans le premier chapitre.

Après d'un pas pour chaque moteur, on incrémente la variable total1_delay par 8000µs pour atteindre la valeur 400 ms, ceci permet de continuer le fonctionnement avec une vitesse plus grande que la précédente. Ceci sera répété pour atteindre la vitesse maximale, en donnant un délai de 900µs entre chaque impulsion [13] (La vitesse du moteur est proportionnelle à la fréquence des impulsions). Enfin, si le nombre commun des pas (common_counts) est épuisé, il faut arrêter le moteur qui a le plus petit nombre de pas, puis appeler une autre fonction pour continuer à exécuter le nombre restant des impulsions sur l'autre moteur. Avant de terminer cette fonction, il faut mettre à jour l'indicateur Motor_verification.

3.1.6- Fonction Move_Motor

Cette fonction a pour but de traiter les données du paquet USB, pour exécuter l'une des fonctions expliquées ci-dessus. L'algorithme de cette fonction est présenté dans la figure 42.

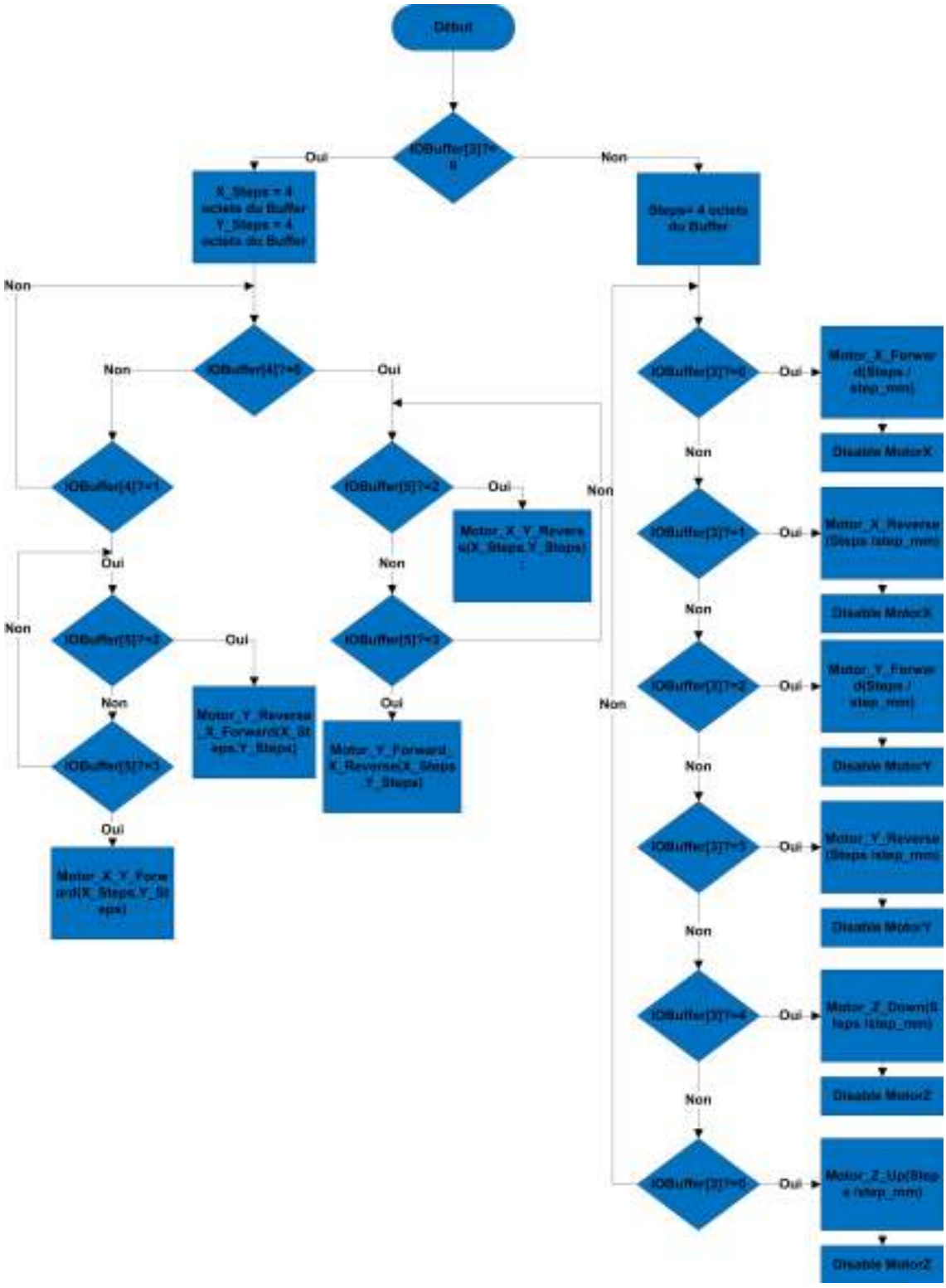


Figure 42- Algorithme de la fonction Move_Motor

IOBuffer[] : l'octet venant de paquet USB

IOBuffer[3] : Mode de fonctionnement, identifie si le fonctionnement est pour un moteur ou pour deux moteurs en même temps.

IOBuffer[4] et IOBuffer[5] : C'est la direction des moteur X et Y respectivement (Fonctionnement simultanés).

Steps : Concaténation de quatre octets, IOBuffer[4] à IOBuffer[7]. (Pour un seul moteur)

X_Steps : Concaténation de quatre octets, IOBuffer[6] à IOBuffer[9].

Y_Steps : Concaténation de quatre octets, IOBuffer[10] à IOBuffer[13].

Step_mm est défini comme constante globale égale à 0.035.C'est trajet que fait le moteur pour un pas en mm.

3.1.7- Fonction USB_Analog

La figure 43 présente l'organigramme de cette fonction. Cette fonction est appelée par le premier octet du paquet USB (CurrentUSBFunction = IOBuffer[0];)

On a plusieurs énumérations: Analog_VCC_Prob_sens, Analog_Temp_sens, etc. On utilise la fonction de **Analog_VCC_Prob_sens** (testé sous la première condition de cet algorithme) pour retourner la valeur de la tension sur chaque point mesuré de la carte électronique.

Le calcul de la tension se fait par une moyenne de huit mesures. Après cette valeur est convertis selon la résolution de l'ADC (10bits) et la valeur du pont diviseur utilisé avec la sonde. Enfin la valeur obtenue est transmise au PC dans un paquet USB.

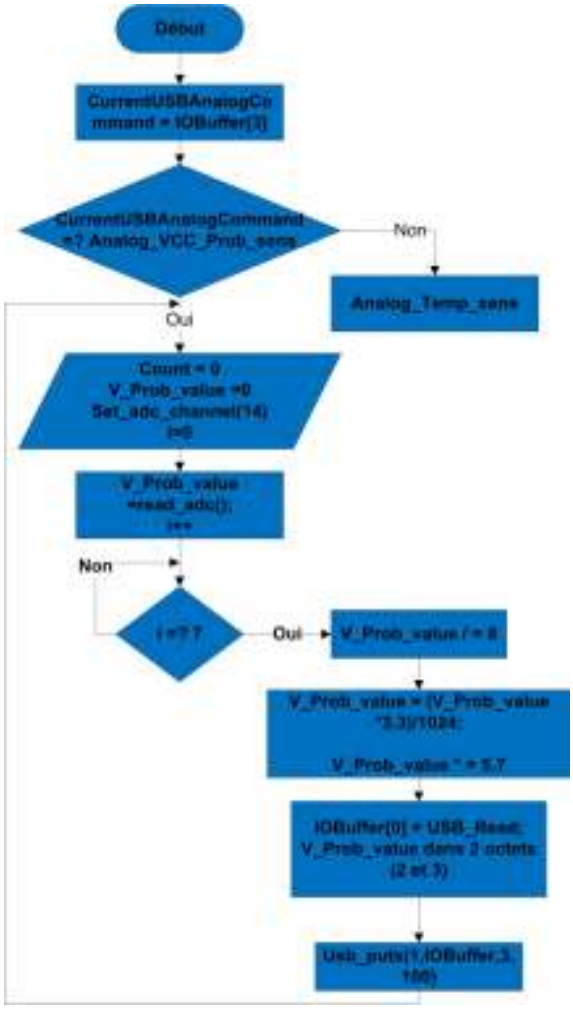


Figure 43- Algorithme USB_Analog

3.1.8- Fonction USB_Power

La figure 44 présente l'organigramme de cette fonction. Cette fonction est aussi appelée par le premier octet du paquet USB (CurrentUSBFunction = IOBuffer[0];)

On a plusieurs énumérations : Power_Drill, Power_Fan, etc. Cette fonction peut être utilisée dans le cas d'implémentation d'une « Drilling machine ».

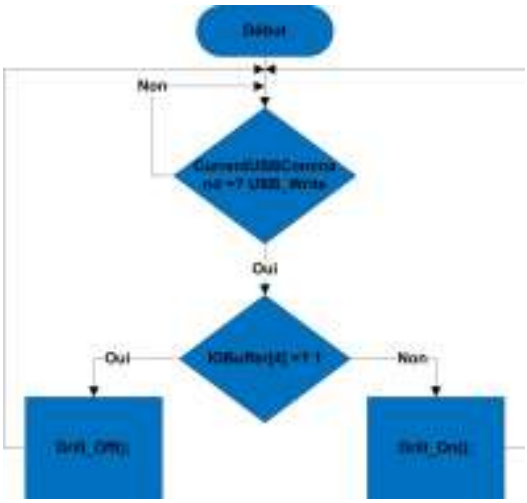


Figure 44- Alogorithme USB_Power

3.1.9- Fonction USB_Verification

Cette fonction permet d'envoyer le statut de l'indicateur Motor_Verification à travers le bus USB vers le PC. La fonction est la suivante :

```
USB_Verification:
    IOBuffer[0] = USB_Read;
    IOBuffer[1] = Motor_Verification;
    usb_puts(1,IOBuffer, 2, 100);
```

Motor_Verification est un indicateur global utilisé dans le code firmware. Il est utilisé dans chaque fonction de déplacement des moteurs. Comme il est défini dans USBDataF sous l'énumération d'USB_function, il est aussi utilisé dans le code API ce qui permet de communiquer avec l'interface graphique.

3.1.10- Code Bootloader

Le bootloader permet de programmer une nouvelle version de l'application firmware dans le microcontrôleur en utilisant la connectivité USB sans la nécessité d'un programmateur.

Le code bootloader est un petit programme qui se situe dans les premiers 4096 octets de la mémoire de programme du microcontrôleur. Le bootloader s'exécute à chaque démarrage du microcontrôleur et selon le test d'une condition, il lance le **programme application** (si la condition est fausse) ou charge une nouvelle version de ce programme d'application (si la condition est vraie).

Si la condition est vraie, le Bootloader attend les données sur la liaison USB et les écrit en mémoire flash. Après on fait Reset pour que le microcontrôleur reparte normalement avec la nouvelle version de l'application.

Le test qu'effectue le bootloader peut être :

- un niveau 0 sur une broche de micro
- la réception d'une séquence particulière
- la lecture d'un octet d'une mémoire extérieure
- etc.

Dans notre cas, on a utilisé le 1er type de test avec la broche RC0. Le test est vrai si RC0= 0 au Reset. Ceci est obtenu à l'aide d'un jumper entre cette broche et la masse, pour la forcer au niveau 0.

Le Bootloader lui-même est cependant chargé une fois dans la mémoire du microcontrôleur avec un programmateur Pickit2. Généralement la zone mémoire où se trouve ce programme est protégé en écriture car si on écrase le bootloader, on ne peut plus revenir à un fonctionnement normal sans l'utilisation d'un programmateur.

Ce procédé est utilisé dans l'industrie, pour mettre à jour le firmware par l'utilisateur final.

Microchip fournit plusieurs projets de booloader adaptables pour ses microcontrôleurs. Il faut adapter les sources fournis à la carte utilisé (Fréquence horloge, type de micro, etc. et choisir la condition d'entrée dans le mode de programmation en bootloader).

Il existe trois conditions à respecter pour écrire un programme de bootloader.

1- Le programme applicatif et le bootloader [3] doivent utiliser les mêmes valeurs des bits de configurations, comme l'exemple ci-dessous :

```
#elif defined(PIC18F87J50_FS_USB_PIM)     // Configuration bits for PIC18F87J50 FS USB Plug-In
                                          // Module board
……..
 #pragma config PLLDIV   = 3              // (12 MHz crystal used on this board=>
                                          // 12/3=4MHz    pour le USB)
```

Il y a aussi les définitions essentielles dans le fichier io_cfg.h qui sont essentielles, tels que :

```
#if defined(USE_USB_BUS_SENSE_IO)
#define usb_bus_sense      PORTCbits.RC5
#define mInitAllLEDs()     LATF = 0x00; TRISF = 0x00;
#define mLED_1             LATFbits.LATF5
#define mLED_2             LATFbits.LATF6
```

Les LED clignotent pour indiquer que l'on est dans le mode bootloader.

2- Il faut décaler l'adresse de chargement du début du programme d'application pour éviter l'écrasement du bootloader. Ce dernier occupe les adresses **0x0000 à 0x0FFF**. L'application doit démarrer à partir de **0x1000** (fig. 45).

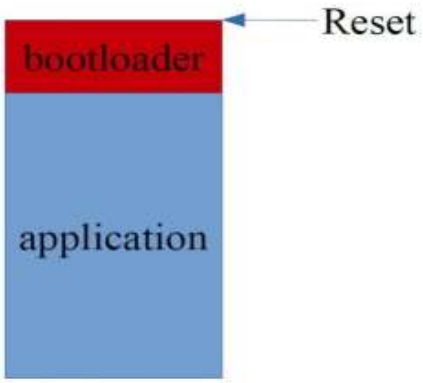


Figure 45- Lieu de bootloader

Cette condition est obtenue par un fichier ajouté au projet de type linker script (18f87j50.lkr). Ce fichier décrit l'affectation des différentes zones mémoires et leur éventuelle protection. [3]

```
…
//Modified linker script to be used with the USB HID Bootloader firmware.
CODEPAGE   NAME=vectors        START=0x0          END=0x1F           PROTECTED
CODEPAGE   NAME=BootPage       START=0x20         END=0xFFF
CODEPAGE   NAME=page           START=0x1000       END=0x1FFF7        PROTECTED
CODEPAGE   NAME=config         START=0x1FFF8      END=0x1FFFD        PROTECTED
CODEPAGE   NAME=devid          START=0x3FFFFE     END=0x3FFFFF       PROTECTED
```

Un extrait de la fonction main est le suivant :

```
#define   HWBoot()   PORTCbits.RC0 == 0
void main(void)
{
        TRISCbits.TRISC0 = 1;     //No need to explicitly do this since reset state is = 1 already.
        if ( (HWBoot()) || (SWBoot()) )   // HW or SW boot?
        {
                LoadApplicationCode();
        }
        else
        {
                _asm
                goto 0x1000     //If the user is not trying to enter the bootloader, go
                                // straight to the main application remapped "reset" vector.
                _endasm
        }
    }                           //end main
```

Alors si la condition RC0=0 est vraie on entre en mode bootloader, sinon le code d'application est lancée à partir de l'adresse 0x1000.

3- Les vecteurs d'interruptions en 0x0008 et 0x0018 doivent être décalés vers 0x1008 et 0x1018 respectivement.

```
/* V E C T O R  R E M A P P I N G */
#pragma code high_vector=0x08
void interrupt_at_high_vector(void)
{
   _asm goto 0x1008 _endasm
}
#pragma code low_vector=0x18
void interrupt_at_low_vector(void)
{ _asm goto 0x1018 _endasm}
#pragma code
```

3.2- Interface de programmation (API)

3.2.1- Introduction

L'**A**pplication **P**rogramming **I**nterface d'extension « .dll » (Dynamic Link Library) joue le rôle de liaisons entre l'interface graphique GUI et le firmware. Il est localisé dans les répertoires des fichiers du **GUI**. Son rôle principal est de prendre la commande du GUI et de la transformer en un paquet USB qui sera communiqué au firmware en utilisant des énumérations spécifiques. Ces dernières sont définies dans le firmware et dans le fichier API que nous avons nommé « Probing_machine_API.dll ».

Le langage utilisé pour écrire l'API est le C++ de Microsoft Visual Studio. C'est un projet indépendant qui utilise beaucoup des fichiers sous le format de header « .h » et « .cpp ». Ces fichiers sont utilisés pour la configuration, le driver USB, les standards des systèmes, etc.

Le fichier USBDataF.h est le même que celui du firmware et le fichier ProbingMachine.cpp définit le point d'entrée pour le fichier .dll de l'application.

Dans le fichier « ProbingMachine.h » il faut déclarer toutes les fonctions utilisés dans « ProbingMachine.cpp », et surtout la taille du Buffer qui est « unsignedchar SendBuff[128] » (équivalent au IOBuffer[128] définit dans le firmware).

Le fichier « USBAPI.h » définit la classe du système d'exploitation (OS) utilisé, « #include<windows.h> ». On a aussi « #define HeaderLength 128 » pour la longueur dupaquet USB à transmettre et les paramètres VID, PID qui sont les mêmes que dans le firmware.

Le fichier« USBAPI.cpp » contient les fonctions essentielles pour la transmission et la réception des paquets USB tels que : MPResult __stdcall, SendReceiveUSBPacket, OpenUSBPort, CloseUSBPort, etc.

Le fichier « ProbingMachine.cpp » contient des fonctions dédiées pour l'application de notre projet, tels que : SetMotor, SetMotorXY, SetDrill_ON_OFF, Get_VCCProb, etc.

3.2.2- Communication avec le firmware pour définir le fonctionnement des moteurs

La fonction « SetMotor » est définit de la manière suivante :

```
MPResult __stdcall SetMotor(int Instance, int Direction, int Steps)
```

Avec :

- Instance: Paramètre utilisé pour permettre contrôler plus qu'une carte (contrôleur).
- Direction: c'est la direction de marche du moteur en avant ou en arrière (selon X, Y ou Z).
- Steps: C'est le nombre de pas à effectuer.

Ci-dessous on donne le détail de la fonction SetMotor :

```
MPResult res;
int status_verification = 0;
        BYTE RxBuff[8];
        DWORD Rlen = 3;
        BYTE temp[4] = {0, 0, 0, 0};
        temp[0] = (BYTE)Steps;
        temp[1] = (BYTE)(Steps >> 8);
        temp[2] = (BYTE)(Steps >> 16);
        temp[3] = (BYTE)(Steps >> 24);
        SendBuff[0] = Move_Motor;
        SendBuff[3] = Direction;
        SendBuff[4] = temp[0];
        SendBuff[5] = temp[1];
        SendBuff[6] = temp[2];
        SendBuff[7] = temp[3];
        res =  SendUSBPacket(Instance, SendBuff, HeaderLength, 100);
        do
        {
                RxBuff[0] = 0;
                RxBuff[1] = 0;
```

```
                SendBuff[0] = USB_Verification;
                SendBuff[1] = USB_Read;
        SendReceiveUSBPacket(Instance, SendBuff, HeaderLength, RxBuff,    &Rlen, 100, 100);
                status_verification = (int)RxBuff[1];
            }while(status_verification == 0);
```

Les quatre octets (temp[0] à temp[3]) sont utilisés pour définir le nombre de pas à marcher, SendBuff[0] à SendBuff[7] sont des octets transmis dans le paquet USB, qui contiennent respectivement: la fonction de mouvement du moteur, sa direction et le nombre de pas à effectuer.

La boucle **do while** permet de tester l'état de l'indicateur USB_Verification qui se trouve dans l'octet SendBuff[0] de paquet USB venant de la carte à microcontrôleur vers le PC. Il est déjà défini et écrit dans le code firmware. Cet indicateur permet de savoir si la fonction de mouvement du moteur a terminé. Cette boucle se trouve aussi dans la fonction SetMotorXY.

Enfin, on a une librairie qui s'appelle « Probing_Machine.def », qui sert à définir toutes les fonctions utilisées dans ce projet et surtout celles qui sont utilisées dans le fichier « ProbingMachine.cpp ».

Après compilation, un fichier « ProbingMachine.dll » est généré automatiquement.

3.2.3- Communication des données de voltage

La fonction « Get_VCCProb » est définit de la manière suivante :

```
MPResult __stdcall Get_VCCProb(int Instance, double *Data)
{
	MPResult res;
	BYTE RxBuff[8];
	DWORD Rlen = 3;

	SendBuff[0] = USB_Analog;
	SendBuff[1] = USB_Read;
	SendBuff[3] = Analog_VCCProb_sens;

	res = SendReceiveUSBPacket(Instance, SendBuff, HeaderLength, RxBuff, &Rlen, 100,
100);
```

```
        *Data = (double)RxBuff[1];
        *Data /= 256;

        *Data += (double)RxBuff[2];

        //*Data *= 2
        return res;
}
```

Cette fonction a pour but de recevoir les données de la tension mesurée. Les données venant du firmware se trouvent dans les deux octets RxBuff[1] et RxBuff[2] (comme indiqué dans l'algorithme du firmware).

L'opération « *Data/256 » effectue un décalage à droite de huit bit pour le premier octet. Ceci est effectué avant de réaliser la concaténation des deux octets venant du firmware.

3.3- L'interface graphique GUI

3.3.1- Introduction

Pour contrôler la machine à travers un protocole de communication USB, on a besoin d'une application utilisateur.

L'application utilisateur peut être de plusieurs types, une suite de lignes de commandes ou bien un script que l'utilisateur fait passer par une application console, ou une interface graphique comme c'est le cas de notre application.

Le contrôle est à travers des boutons, des zones de texte, des curseurs, des graphes et d'autres outils est plus performant. Ce type de programmation nécessite un langage de haut niveau contenant des librairies de contrôle graphique, pour cela on a utilisé le langage C-Sharp (C#). Ce langage est conçu par Microsoft et considéré l'un des langages les plus sophistiqués avec ses librairies et ses classes écrites.

3.3.2- Hiérarchie du logicielle

La figure46 présente la hiérarchie logicielle complète de notre système. On distingue trois types de langage de programmations : le C#, le C++ et le C.

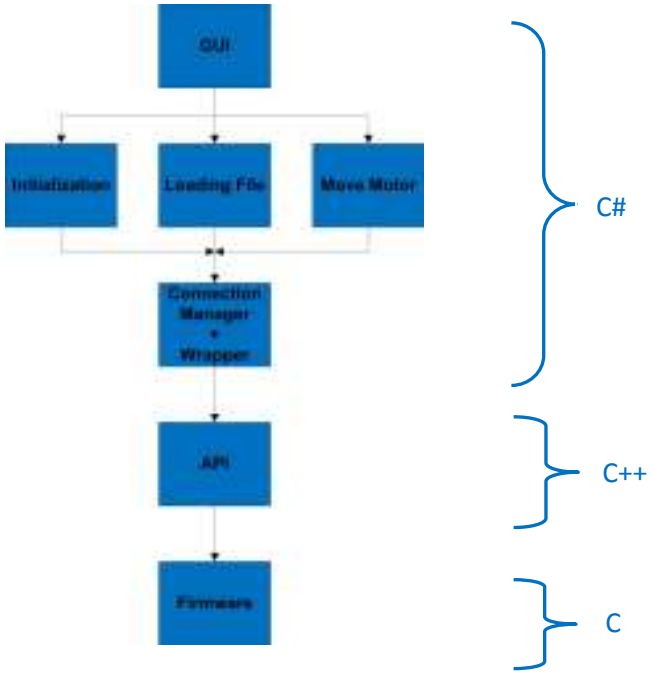


Figure 46- Hiérarchie logiciel complète de notre système

3.3.3- Classes de connexions

Les classes de connexions sont responsables de la communication avec la carte à microcontrôleur. Elles sont utilisées dans toutes les classes de contrôle du programme, tels que les classes des moteurs puisqu'ils assurent la communication entre ces classes et la carte elle-même. Il existe trois classes de ce type qui sont : classe API, classe _ProbingMachineWrapper et classe Connexion Manager.

3.3.3.1- La classe API

On a déjà parlé à propos de l'API. Elle correspond à un fichier d'extension « .dll ». L'initialisation de cette classe aura lieu en utilisant une classe prédéfinie dans le code C# et ceci

par définition du nom de la fonction qui se trouve dans le fichier dll et ses paramètres, puis en accordant à une fonction initialisée dans C# [14].

Par exemple, pour l'initialisation d'une communication, le GUI doit avoir une fonction de connexion qui doit être liée directement à une fonction de connexion dans le fichier dll, à son tour, cette fonction fait initialiser la communication avec la carte en utilisant un Socket. La liaison a la forme suivante :

```
 [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "OpenUSBPort")]

        private static extern MPResult apiOpenUSBPort(int Instance);
```

DLLimport est une fonction qui assure la liaison entre deux périphéries, elle possède deux paramètres, le premier est le répertoire du fichier dll et le deuxième la fonction dans le fichier dll. La troisième ligne a un rôle de créer une fonction dans le langage C# qui a un lien avec la fonction dans le fichier dll, avec ses paramètres qui sont les mêmes paramètres que ceux de la fonction du fichier dll.

Le but principal de ces fonctions à l'intérieur du fichier dll est de former un paquet de commande spécifique au type de communication, qui est dans notre cas un paquet USB. Il faut envoyer ensuite ce paquet au firmware qui à son tour va communiquer avec les autres CIs sur la carte pour l'exécution de ce paquet.

L'envoie du paquet au microcontrôleur se fait par l'une des deux fonctions principales de la classe API, la fonction « SendUSBPacket » pour l'écriture et la fonction « SendReceiveUSBPacket » qui envoie une commande de lecture et reçoit directement l'information qu'elle a demandée.

3.3.3.2- La classe Wrapper

C'est la première classe dans le logiciel qui communique avec la carte électronique. Elle contient un bloc de code qui a comme rôle d'initialiser les fonctions de l'API qu'on va utiliser, comme indiqué dans le paragraphe précédent « Dllimport(…) ».

Les fonctions sont de type **private,** donc elles ne peuvent pas être accessibles en dehors de cette classe. La solution est de les changer en fonctions **public** afin de donner l'accessibilité à d'autres classes. La fonction ci-dessous est un exemple qui se trouve dans la classe wrapper.

```
public    bool    SetMotorXY(MotorDirection    Direction,    MotorDirection    XDirection,
MotorDirection YDirection, int XStep, int YStep)
        {
            return    apiSetMotorXY(ClassInstance,    (int)Direction,    (int)XDirection,
(int)YDirection, XStep, YStep) == MPResult.MPUSB_SUCCES ? true : false;
        }
```

MotorDirection est une énumération qui est définie de la manière suivante :

```
public enum MotorDirection
    { X_Right = 0, X_Left = 1, Y_Right = 2, Y_Left = 3, Z_UP = 4, Z_Down = 5, XY = 6 };
```

Avec cette énumération, toutes les fonctions qui se trouvent dans le fichier API sont redéfinis dans cette classe API. Chaque fonction à son tour est reliée au firmware, ce qui assure la communication avec la GUI et la carte. (Annexe G)

3.3.3.3- La classe Connexion Manager

C'est la dernière classe de communication, elle est de niveau supérieur et est utilisée dans toutes les classes de contrôle. La différence essentielle entre cette classe et celle de la classe « Wrapper » est que tous les paramètres dans les fonctions ne sont pas nécessaires à ce niveau.

Par exemple, toutes les fonctions dans la classe « wrapper » ont un paramètre commun qui est la « `ClassInstance`» qui a pour rôle la séparation entre plusieurs cartes électroniques de même type ou de types différents. Comme c'est le cas de la connexion de deux cartes à travers un bus USB, ce paramètre fait séparer ces deux cartes en donnant à chacune son propre numéro.

La fonction SetMotorXY possède les paramètres suivants et qui permettent d'accomplir la classe Wrapper :

```
publicbool SetMotorXY(MotorDirection Direction, MotorDirection XDirection, MotorDirection
YDirection, int XStep, int YStep)
{ return wrapper.SetMotorXY(Direction, XDirection, YDirection, XStep, YStep);}
```

Les deux classes se combinent ensemble pour donner un outil de communication de haut niveau avec la carte. Pour cela il faut les connecter ensemble pour qu'on puisse utiliser les fonctions de la classe Wrapper. (Annexe H)

3.3.4- Fonction MoveToXY

Cette fonction est utilisée pour déplacer les deux moteurs des axes X et Y vers une autre position quelconque à partir d'une valeur initiale de coordonnées (0,0) (coordonnées absolues, c'est-à-dire à chaque mise hors tension, on aura une perte de position du corps mobile, il est nécessaire de procéder à un point de référence) [15]. Mais Il n'est pas nécessaire de retourner à la position initiale pour continuer à la position suivante. L'organigramme de la fonction est présenté dans la figure suivante (figure 47).

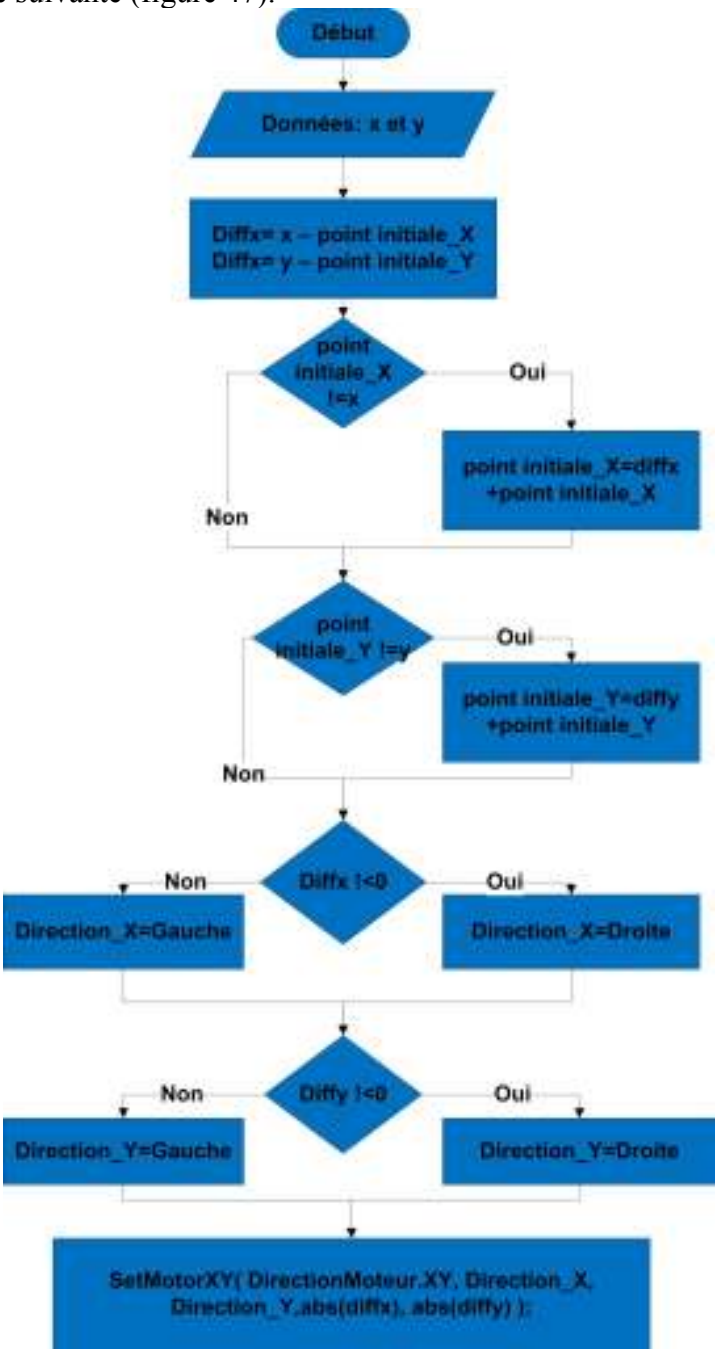


Figure 47- Algorithme de la fonction MoveToXY

Avant l'exécution de cette fonction pour la première fois, il faut initialiser les coordonnées x et y, alors point_initiale_X=0 et point_initiale_Y=0.

Après si les points initiales sont différents des données x ou y, on déplace le moteur correspondant selon la différence obtenue, et on incrémenter les points initiales (qui correspondent alors à la position actuelle) avant l'exécution du second appel de cette fonction. Puis il faut tester ces différences si elles sont positives ou négatives, pour savoir la direction du moteur correspondant.

Par exemple, si Diffx ou Diffy est négatif, il faut déplacer l'objet vers l'arrière, alors la rotation du moteur doit être selon le sens A illustré dans la figure 48.

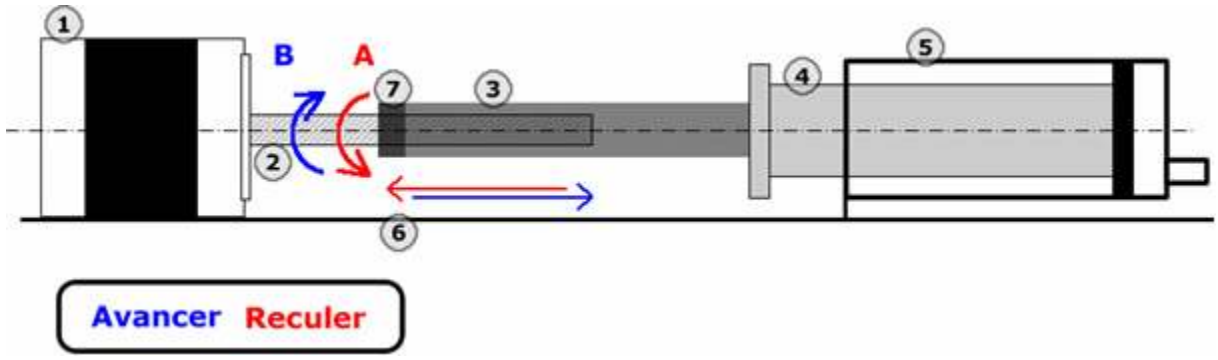


Figure 48- Mouvement de l'arbre du moteur

Enfin la fonction « SetMotor » permet de déplacer les deux moteurs selon les distances et les directions correspondantes. Cette fonction de retour booléen, se trouve dans la partie ConnectionManager et permet à son tour d'appeler la fonction du wrapper, apiSetMotorXY(ClassInstance, (int)Direction, (int)XDirection, (int)YDirection, XStep, YStep) == MPResult.MPUSB_SUCCES ? true : false;

Cette fonction communique avec les classes API pour émettre dans les premiers cinq octets du paquet USB les paramètres intérieurs du la fonction SetMotorXY. Le premier octet contient l'une des valeurs (0 à 6), selon l'énumération suivante

```
Public enum MotorDirection
    { X_Right = 0, X_Left = 1, Y_Right = 2, Y_Left = 3, Z_UP = 4, Z_Down = 5, XY = 6 };
```

Dans notre cas il faut envoyer 6. A la fin le firmware du microcontrôleur fait l'exécution de la fonction correspondante en utilisant les paramètres reçus.

3.3.5- Fonction Load File

Cette fonction (figure 49) permet de charger un fichier Gerber du programme « **Proteus** » qui contient le circuit imprimé. Ce fichier contient toutes coordonnées des points de tests avec leurs noms et la valeur de tension correspondante. Après chargement, on exécute la fonction MoveToXY(…..) pour chaque point sur la carte électronique à tester. On mesure la tension à ce point et on la sauvegarde dans un vecteur qui doit être comparé à la fin avec le fichier initial. Le but final étant d'afficher les résultats de la carte testée, et si elle fonctionne bien ou non. (Voir code dans l'annexe I)

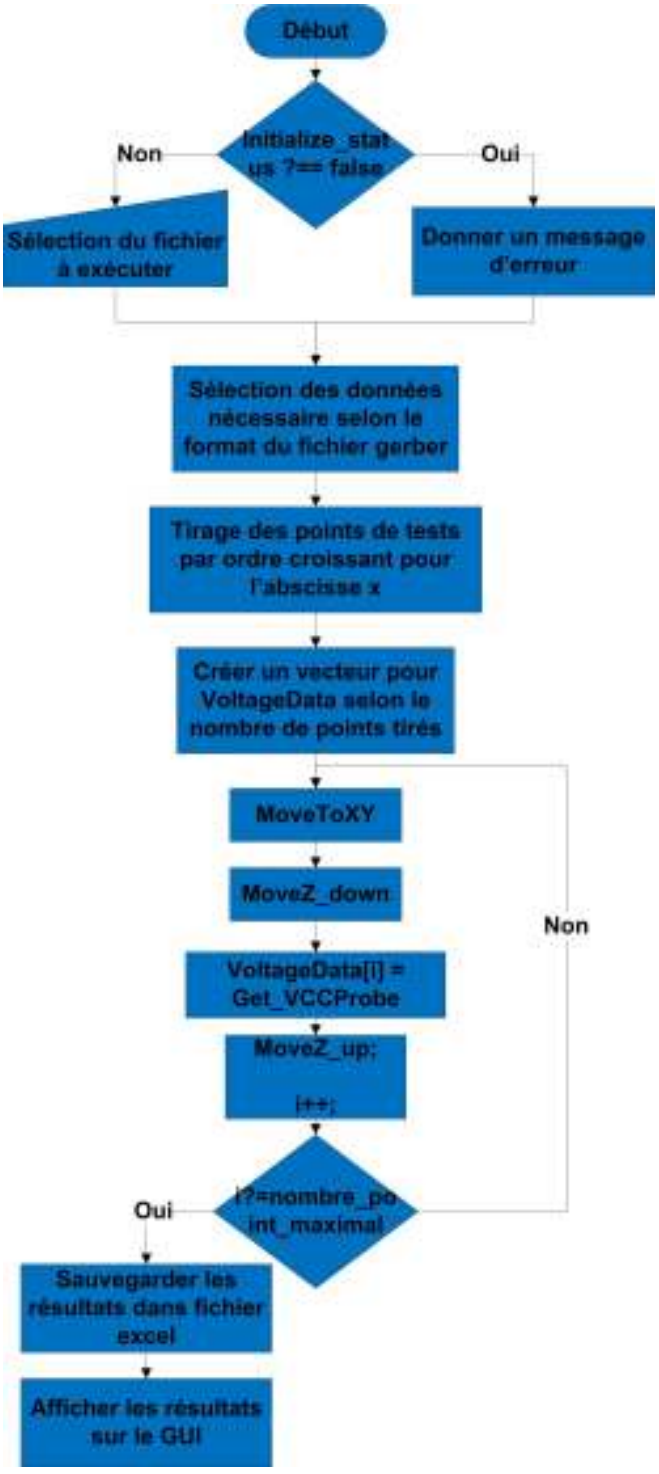


Figure 49- Algorithme LoadFile

3.3.6- Initialisation de la machine

Avant de commencer le test sur une carte électronique, il faut toujours initialiser les axes à leurs positions initiales (0, 0, 0). Pour cela un bouton sur l'interface graphique du système et qui s'appelle « Initialize » permet de donner aux trois moteurs selon les axes x, y et z les commandes nécessaires pour reculer vers leurs positions initiales. Ces positions initiales sont identifiées sur les axes par des capteurs optiques. L'algorithme de l'initialisation du système est présenté à la figure 50.

Il n'est pas nécessaire d'initialiser la machine dans le cas où les axes se trouvent déjà à leurs positions initiales. Dans ce cas on a utilisé à la fin de cet algorithme l'indicateur booléen Initialize_status, qui permet d'informer s'il est vrai que la machine ne nécessite pas d'initialisation.

On a utilisé au début la fonction Motor_Z_Up (max_value) pour être sur de ne pas casser les composantes électroniques qui se trouvent sur la carte à tester (Il ne faut effectuer des mouvements selon les axes X et Y avant de rétracter la sonde selon l'axe z). Max_value est la valeur maximale vers le haut que l'axe Z peut effectuer.

La fonction Motor_X_Reverse_Y_Forward est ensuite utilisée selon la construction mécanique de la machine afin de retourner aux positions initiales selon les deux axes X et Y. La position de la sonde est alors affichée de manière dynamique sur l'interface graphique permettant ainsi à l'utilisateur de suivre la sonde de manière instantanée.

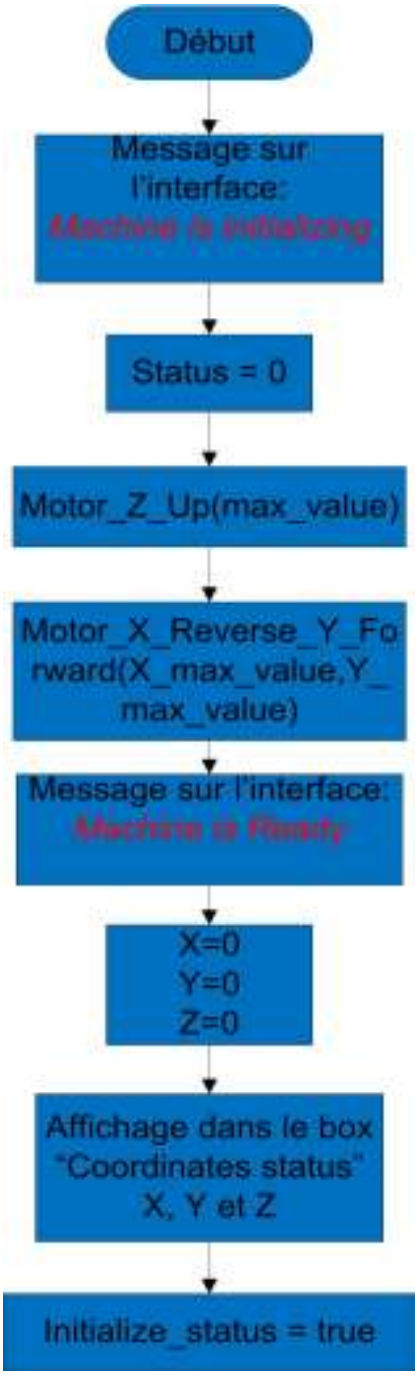


Figure 50- Algorithme d'initialisation de la machine

Chapitre 4

Essais et mesures

4.1- Introduction

Dans ce chapitre on va présenter quelques tests avec la « Probing Machine » réalisée en utilisant comme prototype des cartes électroniques qui comprennent des points de tests avec différentes tensions de mesures. Le but est de présenter les performances du système obtenu.

4.2- Câblage de la carte de contrôle

La carte de contrôle de la machine nécessite quelques câblages avec les sources d'alimentations, les moteurs pas-à-pas et les interrupteurs (SearchRun et fin de course). Les tensions d'alimentation continue pour fonctionner sont les suivants :

- 9V (400mA) pour alimenter la carte, qui sera convertie ensuite en 5 V puis en 3.3V.
- 12V (400mA) pour l'alimentation du moteur selon l'axe Z.
- 24 V (Minimum 3A) pour l'alimentation des moteurs selon les axes X et Y.

Les moteurs sont reliés avec la carte à travers des connecteurs, qui sont faciles à les ôter de la carte en cas de réparation.

Les interrupteurs de fin de course sont aussi connectés à travers des connecteurs (Pin Header).

On a utilisé une sonde spéciale pour mesurer la tension sur la carte à tester. Cette sonde contient un ressort qui est indispensable durant la mesure afin d'obtenir une bonne adhérence avec le point de test et par suite une bonne conduction. La figure 51 présente la sonde utilisée.

Le mécanisme complet de la machine est présenté par la figure 52, qui contient la carte de contrôle, les interrupteurs et la sonde. La figure 53 illustre carte de contrôle et son câblage.

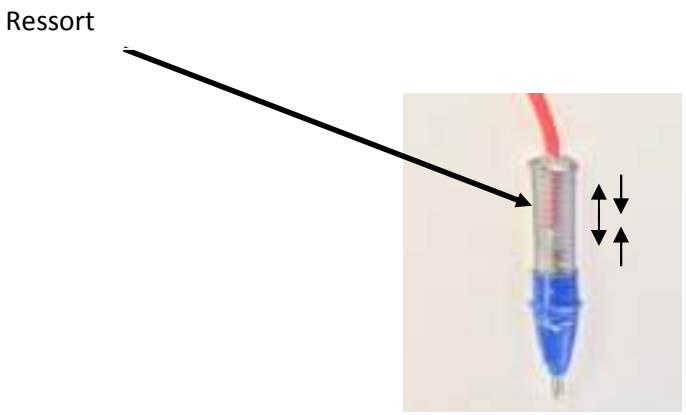


Figure 51- Mécanisme de la sonde

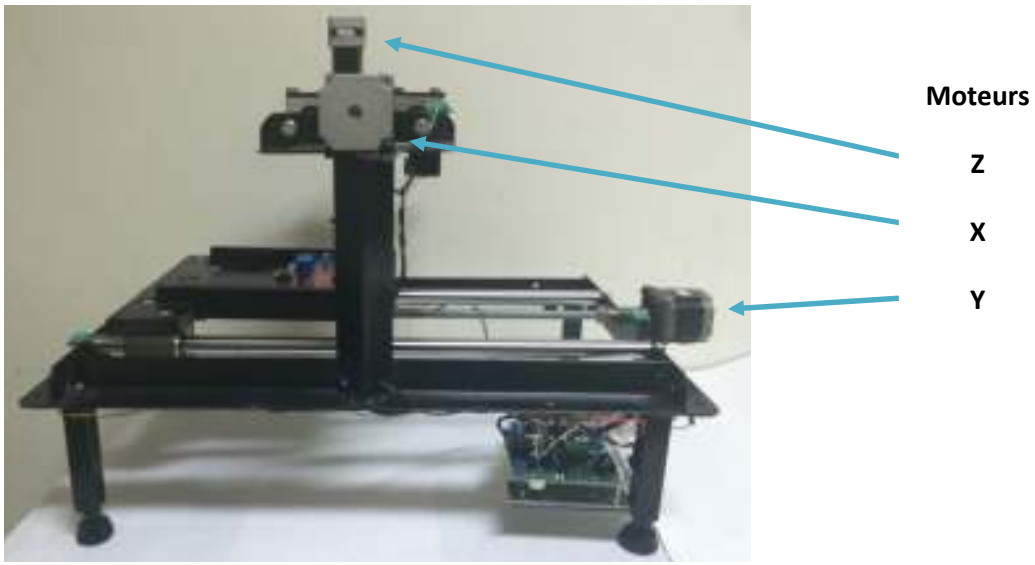


Figure 52- Mécanisme complet de la machine

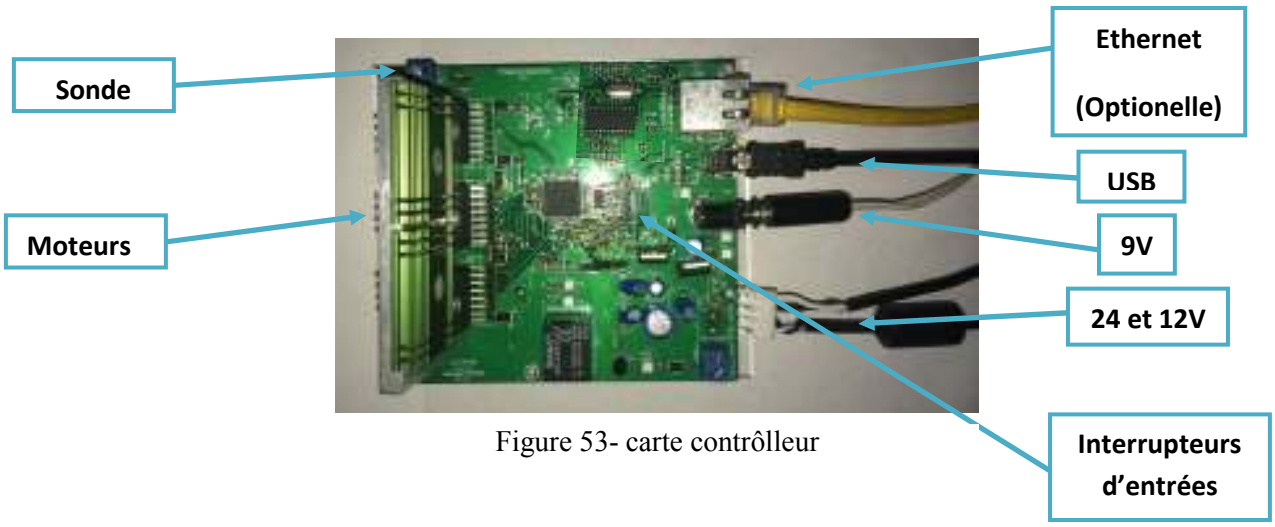


Figure 53- carte contrôlleur

4.3- Fonctionnement de l'interface graphique

Tout d'abord, on va présenter l'interface graphique, qui permet de contrôler la machine, dans plusieurs modes, en communiquant à travers le bus USB.

Figure 54- Fenêtre d'entrée de l'interface graphique

La figure 54 présente la fenêtre d'entrée de l'application. On peut y trouver plusieurs modes de fonctionnement. Cependant avant de présenter ces différents modes, il faut lancer la connexion à travers l'USB.

4.3.1- Connexion à travers l'USB

On appuie sur le bouton « **Connect** ». En cas de succès de la connexion, on obtient le message illustré dans la figure 56. Le bouton « Connect » change alors en bouton « **Disconnect** ». En cas d'échecon obtient le message d'erreur suivant (fig. 55):

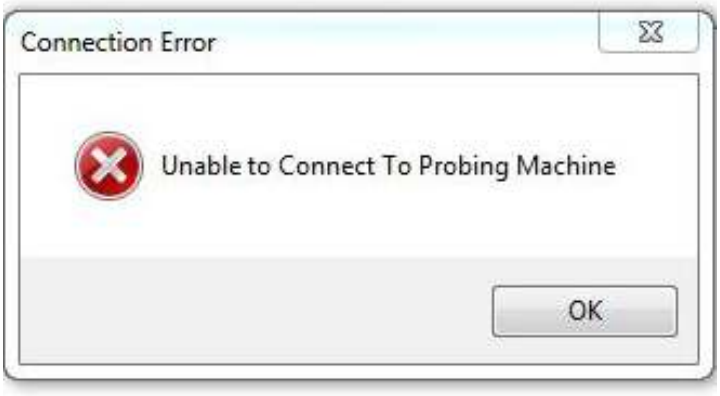


Figure 55- Message d'erreur de Connexion USB

La cause de l'échec est generaıement que:

- le câble USB n'est pas attaché au PC ou à la carte de contrôle.
- la carte de contrôle n'est pas alimentée.

Figure 56- Succès de connexion USB

Si on veut terminer le test, il faut appuyer sur le bouton « **Disconnect** », et on aura le message suivant sur l'interface : « **Machine is disconnected**». Le bouton change alors de nouveau à son état initiale « **Connect** ».

Machine is disconnected

Et le bouton sera initialisé de nouveau à sa écriture initiale **« Connect ».**

4.3.2- Initialisation de la machine

Le bouton « **Initialize** » (fig. 56) permet de renvoyer les moteurs à leurs positions initiales. Il indique à la fin que la machine est prête à démarrer avec tous les autres modes.

C'est indispensable d'initialiser la machine avant de déplacer l'un des trois moteurs, et ceci afin connaître la position selon chaque axe. Sinon, les moteurs ne fonctionnent pas et on aura le message suivant (fig. 57):

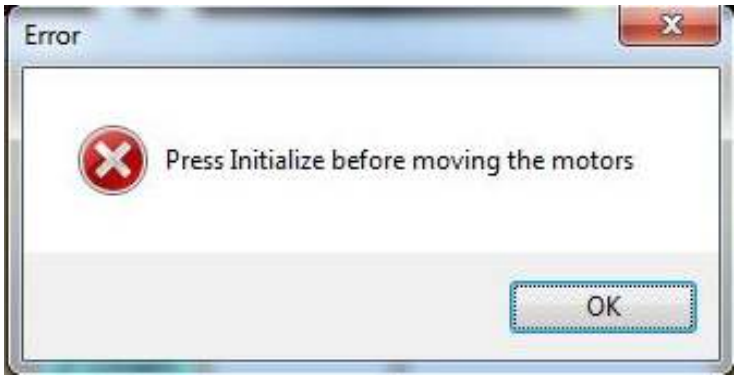


Figure 57- Message d'erreur d'initialisation

4.3.3- Mode manuel

Le premier mode de fonctionnement qu'on va présenter est le mode manuel illustré dans la figure 58. Il permet de déplacer de façon manuelle chaque axe pour la distance voulue en entrant la valeur désirée en millimètres (mm) dans la zone correspondante, puis en appuyant sur le bouton de contrôle correspondant à notre choix.

Ce mode permet aussi de tester la tension en appuyant sur le bouton « **Test Voltage »,** et le résultat sera affiché instantanément dans le cadre correspondant au-dessous de ce bouton.

Sur la droite de l'interface, on a le statut des coordonnées X, Y et Z, c'est-à-dire la position instantanée de chacun d'eux après initialisation.

Les zones blanches qui se trouvent juste à côté des boutons indiquent la distance restante pour à parcourir par les moteurs en marche.

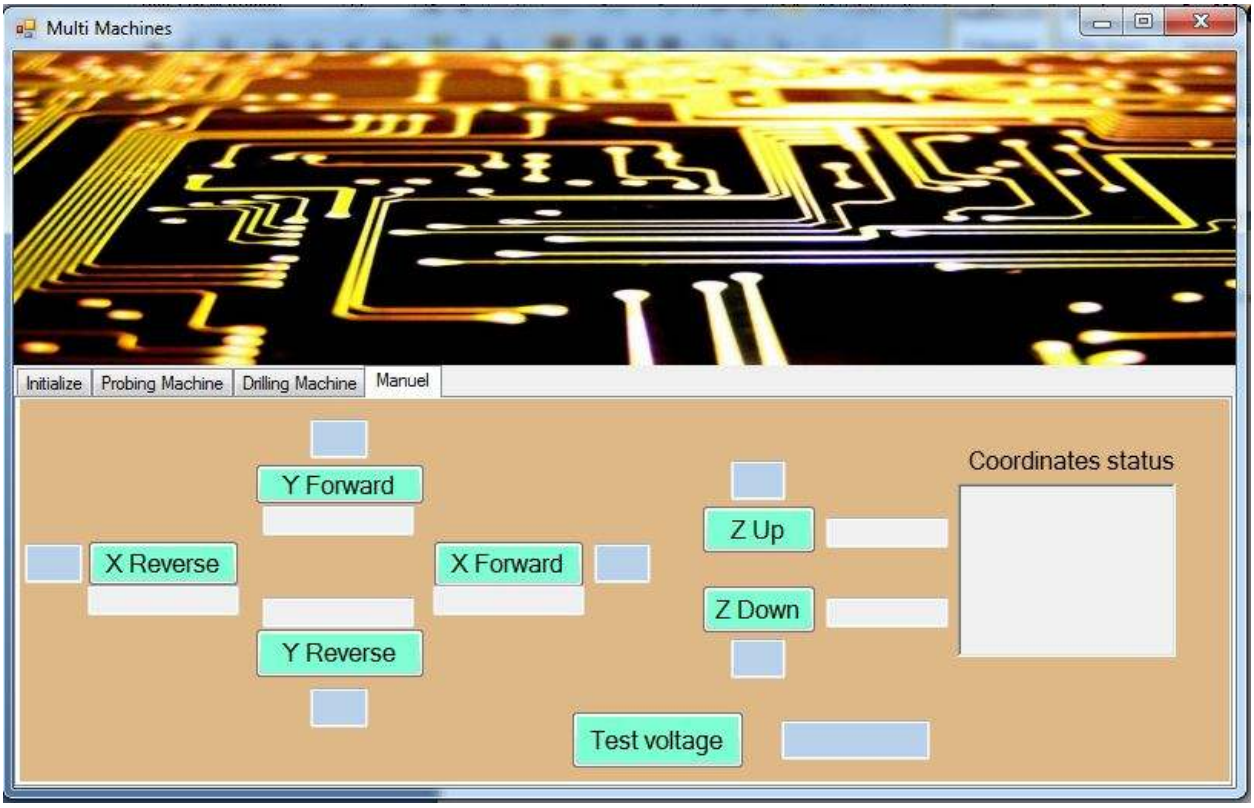


Figure 58- Mode manuel

4.3.4- Mode automatique

Le deuxième mode, est le mode automatique de « Probing Machine ». Il permet de charger la carte par un un fichier « **TPI** » (Test Point Information) . La figure 59 présente le mode automatique.

Comme dans notre cas, le programme « **Proteus** » qui contient le circuit imprimé de la carte à tester. On peut extraire tout type de fichier concernant cette carte, mais ce qui nous intéresse est le fichier Gerber ou celle de TPI (TestPoint Information), qui contient les coordonnées X et Y de tous les trous qui se trouvent sur la carte correspondante.

Le format accepté par notre interface graphique est de type TPI ou CSV.

Le bouton « **Load File** » ne fonctionne pas, que si l'initialisation de la machine a été effectuée. Après le chargement du fichier « **TPI** », il faut appuyer sur le bouton « **Start** » pour que la

machine commence à faire les tests demandés. Les résultats de test sont mises à jour instantanément dans le boitier gris, qui indiquent si la tension le point testé est correcte ou non. Ces résultats seront expliqués par la suite dans un exemple de test réel.

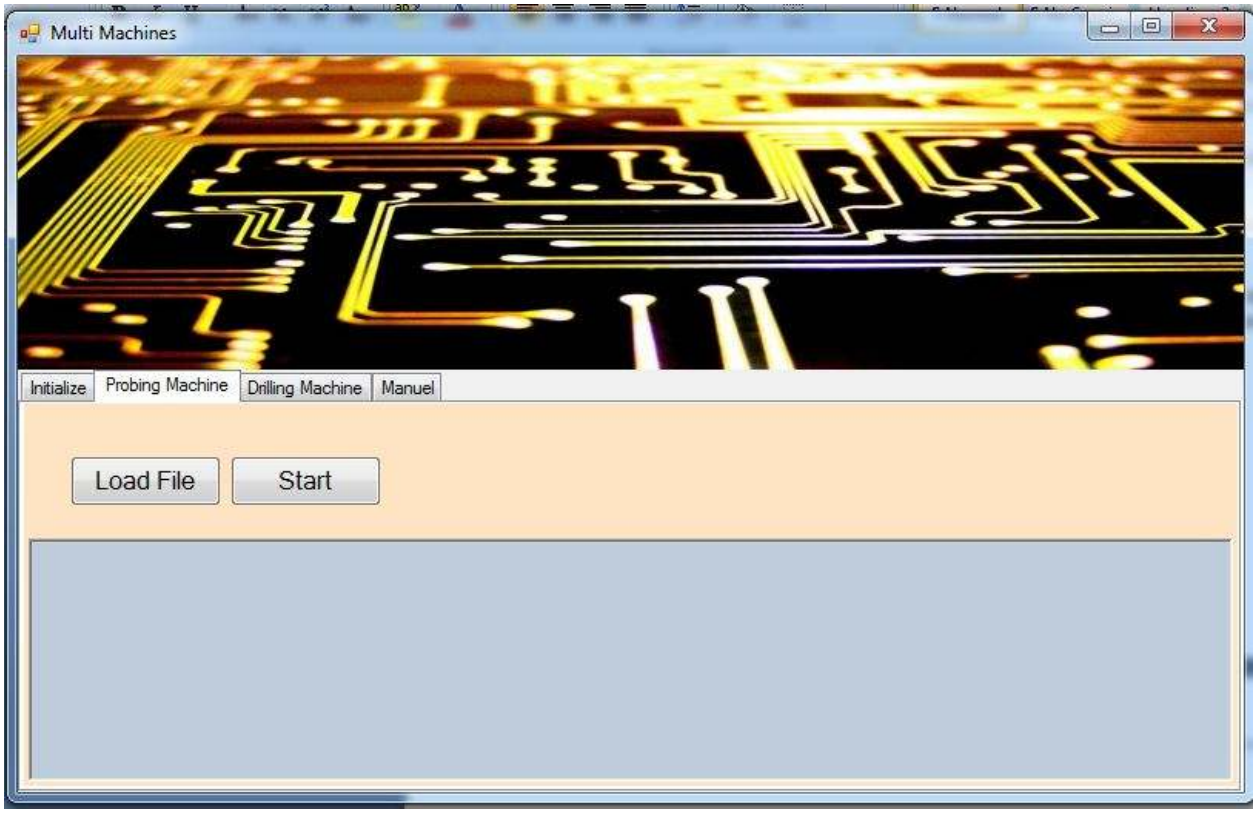


Figure 59- Mode automatique

4.3.5- Exemple de test sur une carte réelle

Maintenant on va présenter un exemple réel de test sur une carte électronique. Cette carte possède comme dimensions$15 \times 12cm$. On va y tester plus de 30 points de test, ce qui permet de vérifier le fonctionnement notre système.

La figure 60 illustre la carte en question : il faut tester tous les carreaux verts.

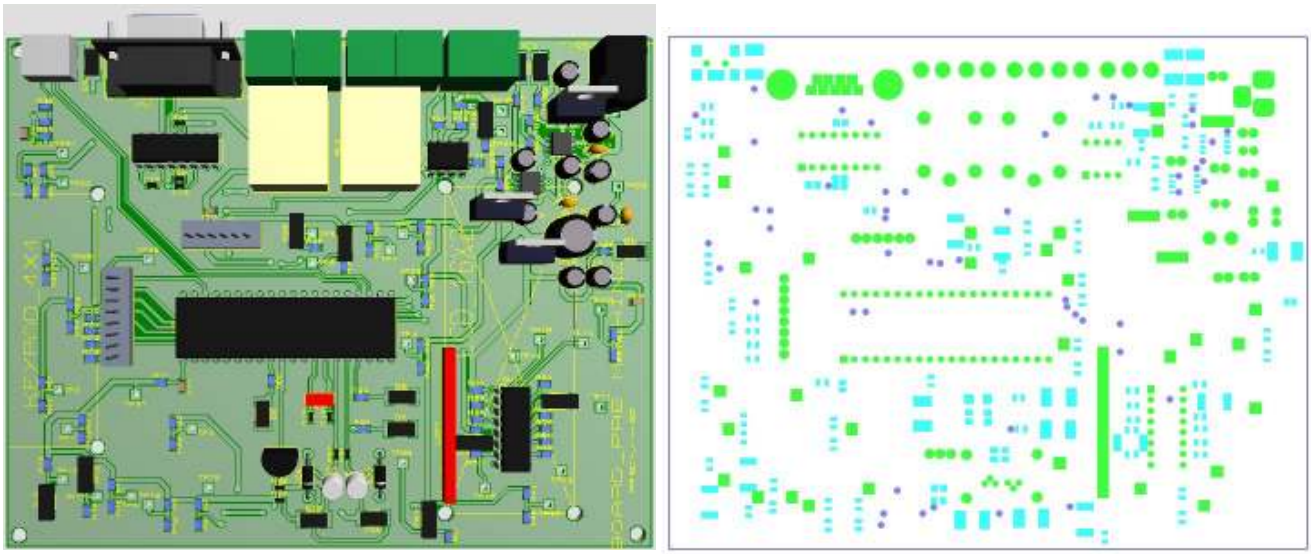


Figure 60- Exemple d'une carte en question

Tout d'abord, on récupère le fichier TPI correspondant à la carte. Ce fichier à la forme suivante (fig. 61):

Testing board22_for PCB (2).TPI - Notepad

File Edit Format View Help

```
LABCENTER PROTEUS TESTPOINT INFORMATION FILE
====================--------==================

Testpoint positions for Testing board22_for PCB.LYT

Fields: ID, Type, X, Y, Net
Units:  X, Y  - thou
Notes:  This file lists pads which are accessible from the bottom side of the board.
        The X, Y value is the centre of drill hole or pad origin relative to the outpu
        Vias are only listed if they are exposed through the solder resist.

"J1:1","THRU",5507.87,4074.8,"#00010"
"J1:2","THRU",5507.87,4330.71,"GND=POWER"
"J1:3","THRU",5311.02,4173.23,"GND=POWER"
"C1:+","THRU",5417.32,3468.5,"#00010"
"C1:-","THRU",5317.32,3468.5,"GND=POWER"
"C13:+","THRU",5326.77,2507.87,"VCC/VDD=POWER"
"C13:-","THRU",5426.77,2507.87,"GND=POWER"
"U7:6","THRU",1728.35,3519.69,"#00095"
"U7:7","THRU",1828.35,3519.69,"{NC}"
"U7:8","THRU",1928.35,3519.69,"{NC}"
"RL2:NC","THRU",3622.05,3972.44,"{NC}"
"RL2:NO","THRU",3149.61,3972.44,"#00233"
"LCD1:11","THRU",4024,1514,"D4_LCD"
"LCD1:12","THRU",4024,1614,"D5_LCD"
"LCD1:13","THRU",4024,1714,"D6_LCD"
"LCD1:14","THRU",4024,1814,"D7_LCD"
"PIN:TP1","THRU",551.181,1102.36,"P4V54"
"PIN:TP2","THRU",472.441,1456.69,"P10V41"
"PIN:TP19","THRU",511.811,629.921,"P4V64"
```

Figure 61- Format du fichier TPI

Chaque ligne de ce fichier contient respectivement le nom, le type, les coordonnées (X, Y) et la liaison de la broche. Ce qui nous intéresse sont les points de tests, par exemple TP2 est de type **« Through Hole »**, son abscisse X en mil (1/1000 inch) et son ordonnée Y aussi en mil sont (472.441,1456.69), et enfin la liaison et sa valeur de tensionP10V41→ 10.41Volt.

Il faut faire attention avant l'extraction du fichier TPI et le début du test que la carte se trouve à l'origine du repère comme indiqué par la figure 62. Il faut appuyer ainsi sur le bouton **« Intitialize »,** sinon, l'interface ne permet jamais de charger le fichier, en donnant un message d'erreur comme il a été indiqué par le paragraphe 4.3.2

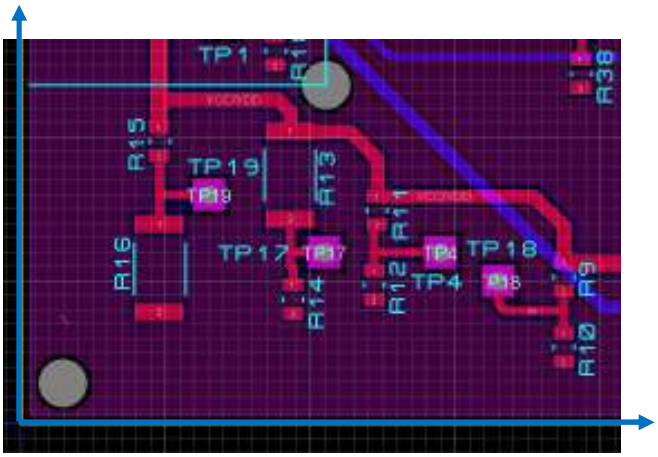


Figure 62- Repère de la carte testée

Après l'initialisation de la machine, les axes recules à leurs positions initiales, quelque soit la position de chaque axe. Puis après avoir établie la connexion USB à l'aide du bouton **« Connect »**, on obtient le message **«Machine is Ready »**.

Maintenant la machine est prête pour charger le ficher TPI en appuyant sur le bouton **« Load File »** (voir figure 63).On peut voir qu'on peut choisir deux formats de fichier, soit le « .csv » ou le « .TPI ».

Le fichier .csv est utilisé dans le cas où on va utiliser comme entrée un fichier que nous avons générer de façon manuelle, ou un fichier déjà testé et sauvegarder sous le format « .csv ».

Il faut appuyer maintenant sur le bouton « Start » pour commencer le test. Les points à tester ne sont pas parcourue selon leur apparence dans le fichier mais selon l'ordre croissant des abscisses

des points à tester selon l'axe des X. Ceci est fait de manière automatique sans aucune intervention de l'utilisateur dans le but de minimiser l'excursion selon l'axe des X et par suite accélérer le test.

Figure 63- Chargement d'un fichier TPI

4.3.5.1- Affichage des résultats sur l'interface

Le résultat individuel de chaque point de test sera affiché en temps réel avec son résultat (s'il se trouve dans la marge ou non)dans la zone gris-bleu en bas de la forme comme illustré dans la figure 64.

Le rapport de sortie contient dans sa deuxième ligne le nom du fichier chargé « .TPI ». Puis chaque ligne contient :

- le nom de la broche a testée,
- les valeurs de ces coordonnées X et Y (convertis en mm),

- la liaison électrique qui contient la valeur de la tension à mesurer « Real Voltage » (expliqué avant : P10V41),
- la tension mesurée « Real Voltage »
- et enfin le résultat du point testé, si la valeur est correct ou non (succès (Pass)si la tension mesurée est à $\pm 0.1 Volt$de la tension requise).

Une fois le test sur terminé, on peut sauvegarder le rapport sous le même nom de fichier chargé en le suffixant par « _SN » où SN est le numéro de série de la carte testée (fig. 65).

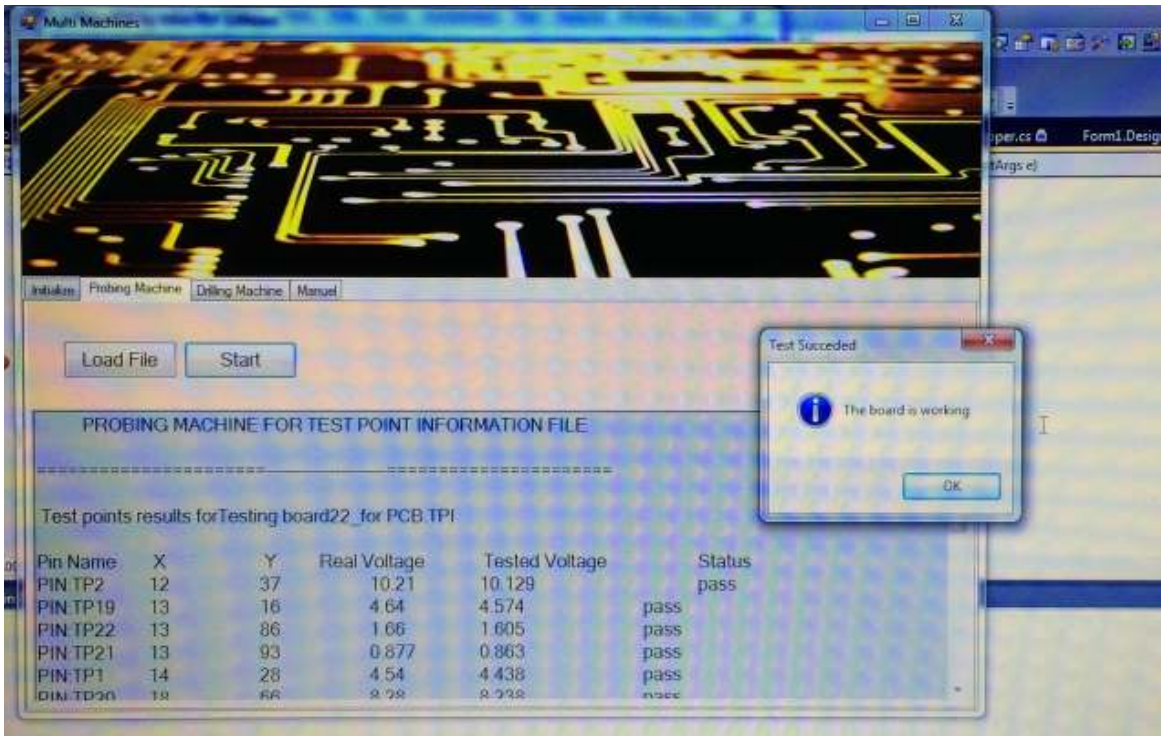


Figure 64- Résultat du test sur l'interface

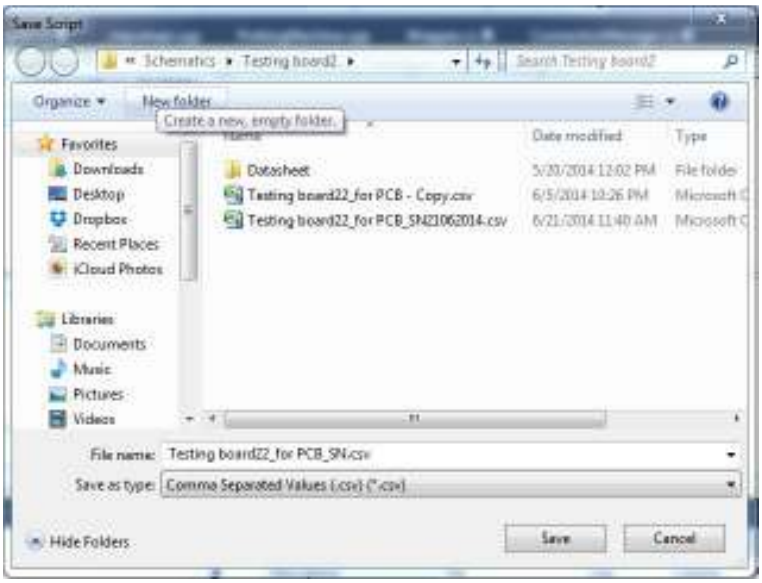


Figure 65- Fichier à sauvegarder

4.3.5.2- Résultat final de la carte testée

Après l'enregistrement du rapport, on aura l'un des deux cas :

- Si tous les points testés sont à l'intérieur de leur marge acceptable définie de la manière suivante :

$$Tested\ Voltage = Real\ Voltage\ \pm 0.1\ Volt \qquad \text{Eq. 4}$$

On obtient le message suivant (fig. 66):

Figure 66- Message de succès du test

- Siau moins un des points testés est en dehors de la marge acceptable, on obtient le message suivant (fig. 67) :

Figure 67- Message de défaillance du test

Après l'appui sur le bouton OK, on aura un autre message (fig. 68):

Figure 68- Message d'impression du rapport

On peut par la suite choisir le type de l'imprimante pour imprimer notre rapport [16] :

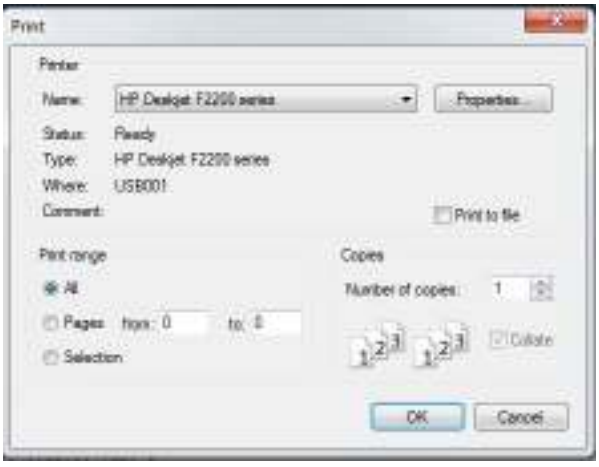


Figure 69- Choix de l'imprimante

4.6- Commentaires sur les résultats de test

Ce test effectué valide les performances du système réalisé. Le temps globale du test est 150 à 200 secondes ce qui fait une moyenne de 5 secondes par point de mesure. Ceci est largement meilleur qu'un test manuel à la main, dans lequel la carte prend au moins 15 minutes, sans l'enregistrement des résultats sur un papier ou sur un PC. Avec l'enregistrement manuel des résultats, l'opération peut prendre 20 minutes.

On a ainsi minimisé le temps de mesure des cartes électroniques, et en plus on a réduit le risque d'erreur humaine, que ça soit par exemple par un court-circuit, ou par une lecture incorrect ou un enregistrement incorrect des valeurs des tensions mesurées.

Conclusion et perspective

L'évolution de machines automatisées aide les êtres humains à être plus productif. Pour cela on aperçoit que le fonctionnement de ces machines est de plus en plus intelligent. Cette intelligence est basée sur des algorithmes : firmware et software.

Ces machines se composent de plusieurs parties, et on peut les regrouper en trois parties, tels que, la partie mécanique, le logiciel et matériel. La première partie se compose des composantes mécaniques qui construisent le mécanisme qui permet de servir l'application, tels que les moteurs, les capteurs, etc. La deuxième est le logiciel qui est toujours en liaison avec la partie matériel qui est le contrôle de la partie mécanique.

Dans de futurs projets, on peut se baser sur le système réalisé et ajouter plusieurs modes de fonctionnement.

On peut par exemple construire des machines de gravure de bois, en chargeant la machine par le fichier qui comporte toutes les coordonnées d'un dessin ou d'une écriture. Ce type de projets nécessite une précision de mouvement pour les trois axes X, Y et Z. On peut alors modifier les moteurs par des autres qui possède une résolution de 0.15˚(c.à.d. 9066 pas dans chaque tour) et changer la sonde sur l'axe Z par une perceuse. Ce type de machine permet aussi de faire des prototypes des cartes électroniques, c'est-à-dire engraver sur des plaques en cuivre le dessin des circuits imprimés et ensuite percer les pattes des composants « through hole ».

Le système réalisé est très paramétrable et permet de tester et donner des informations détaillées de tous types de circuits imprimés. Il permet aussi, par de minimes modifications, de réaliser différents projets dans le domaine de la gravure.

Bibliographie

[1] Computer pioneers. Lee, J. Jhon T. Parsons, IEEE computer society. [En ligne]. Disponible: http://computer.org/computer-pioneers/parsons.html. [Consulté en Aout 2014].

[2] Kesavadas, T., (2007). Manufacturing Automation : notes de cours : Cours MAE564/464 1ère édition, Buffalo, Newyork university

[3] Microchio Company. Forum. [En ligne]. Disponible : http://www.microchip.com/forum. [Consulté en Février 2014].

[4] - H bridge motor control circuit using L298, http://www.circuitstoday.com/ site web, Consulté en Février 2014

[5] Omega company. "Introduction to stepper motors". *Stepper motors*. [En ligne]. Disponible: http://www.omega.com/prodinfo/stepper_motors.html [Consulté en Aout 2014]

[6] Fairchild Semiconductor. "1A Positive Voltage Regulator", Fairchild Semiconductor Corporation, Etats Unis, Rapport technique, 2013. [En ligne]. Disponible : https://www.fairchildsemi.com/datasheets/LM/LM7805.pdf. [Consulté en Février 2014].

[7] Texas Instrument Incorporated. «Terminal Adjustable Regulator», Texas Instrument Incorporated, Texas, Etats Unis, Rapport technique, 2013. [En ligne]. Diponible: http://www.ti.com/product/lm317. [Consulté en Février 2014].

[8] Universal Serial Bus organization. «A Technical Introduction to USB 2.0 » [En ligne]. Disponible: http://www.usb.org/developers/usb20/backgrounder. [Consulté en Mai 2014].

[9] R. Murphy, « USB 101: An Introduction to Universal Serial Bus 2.0" Etat Unis, Rapport technique, 2012. [En ligne]. Disponible: http://www.cypress.com/?rID=39327. [Consulté en Mai 2014].

[10] F. Lusteau, « Etude, Techniques de codage sur fibre optique ou paire torsadée », Examen probatoire en Informatique, Cnam, Paris, France, 1999. [En ligne].Disponible : http://deptinfo.cnam.fr/Enseignement/Memoires/LUSTEAU.Franck/Acceuil.htm. [Consulté en Mai 2014].

[11] P. Batude, « Intégration à trois dimension séquentielle: Etude, fabrication et caractérisation », Ph.D., Institut polytechnique, Grenoble, France, 2009. [En ligne]. Disponible : http://tel.archives-ouvertes.fr/docs/00/45/54/28/PDF/manuscrit_batude.pdf.[Consulté en Mai 2014].

[12] CCS (Custom Computer Services), Forum. [En ligne]. Disponible: http://www.ccsinfo.com/forum/. [Consulté en Mars 2014].

[13] J. Quinones, « Applying acceleration and decelaration profiles to bipolar stepper motors", Texas Instrument Incorporated, Texas, Etats Unis, Rapport technique, 2012. [En ligne]. Disponible: http://www.ti.com/lit/an/slyt482/slyt482.pdf. [Consulté en Mars 2014].

[14] A. Mugali, « Calling API functions using C#", *Code project,* 2001. [En ligne]. Disponible: http://www.codeproject.com/Articles/1285/Calling-API-functions-using-C. [Consulté en Mai 2014].

[15] BEI ideacod company, « Incremental, Absolute and Analog Encoders Catalog", Shnider Electric, Strasbourg, France, Rapport technique, 2005. [En ligne]. Disponible: http://www.bmb.co.th. [Consulté en Mai 2014].

[16] Microsoft Developer Network, How to: Invoke a print dialog, 2014. [En ligne]. Disponible: http://msdn.microsoft.com/en-us/library/aa970848%28v=vs.110%29.aspx. [Consulté en Mai 2014].

Annexe A (Datasheet Pic18F87J50)

MICROCHIP

PIC18F87J50 FAMILY

64/80-Pin High-Performance, 1-Mbit Flash USB Microcontrollers with nanoWatt Technology

Universal Serial Bus Features:

- USB V2.0 Compliant SIE
- Low Speed (1.5 Mb/s) and Full Speed (12 Mb/s)
- Supports Control, Interrupt, Isochronous and Bulk Transfers
- Supports up to 32 Endpoints (16 bidirectional)
- 3.9-Kbyte Dual Access RAM for USB
- On-Chip USB Transceiver

Flexible Oscillator Structure:

- High-Precision PLL for USB
- Two External Clock modes, up to 48 MHz
- Internal 31 kHz Oscillator, Tunable Internal Oscillator, 31 kHz to 8 MHz
- Secondary Oscillator using Timer1 @ 32 kHz
- Fail-Safe Clock Monitor:
  - Allows for safe shutdown if any clock stops

Peripheral Highlights:

- High-Current Sink/Source 25 mA/25mA (PORTB and PORTC)
- Four Programmable External Interrupts
- Four Input Change Interrupts
- Two Capture/Compare/PWM (CCP) modules
- Three Enhanced Capture/Compare/PWM (ECCP) modules:
  - One, two or four PWM outputs
  - Selectable polarity
  - Programmable dead time
  - Auto-shutdown and auto-restart
- Two Master Synchronous Serial Port (MSSP) modules supporting 3-Wire SPI (all 4 modes) and I^2C™ Master and Slave modes
- 8-Bit Parallel Master Port/Enhanced Parallel Slave Port with 16 Address Lines
- Dual Analog Comparators with Input Multiplexing

Peripheral Highlights (continued):

- 10-Bit, up to 12-Channel Analog-to-Digital (A/D) Converter module:
  - Auto-acquisition capability
  - Conversion available during Sleep
- Two Enhanced USART modules:
  - Supports RS-485, RS-232 and LIN 1.2
  - Auto-wake-up on Start bit
  - Auto-Baud Detect

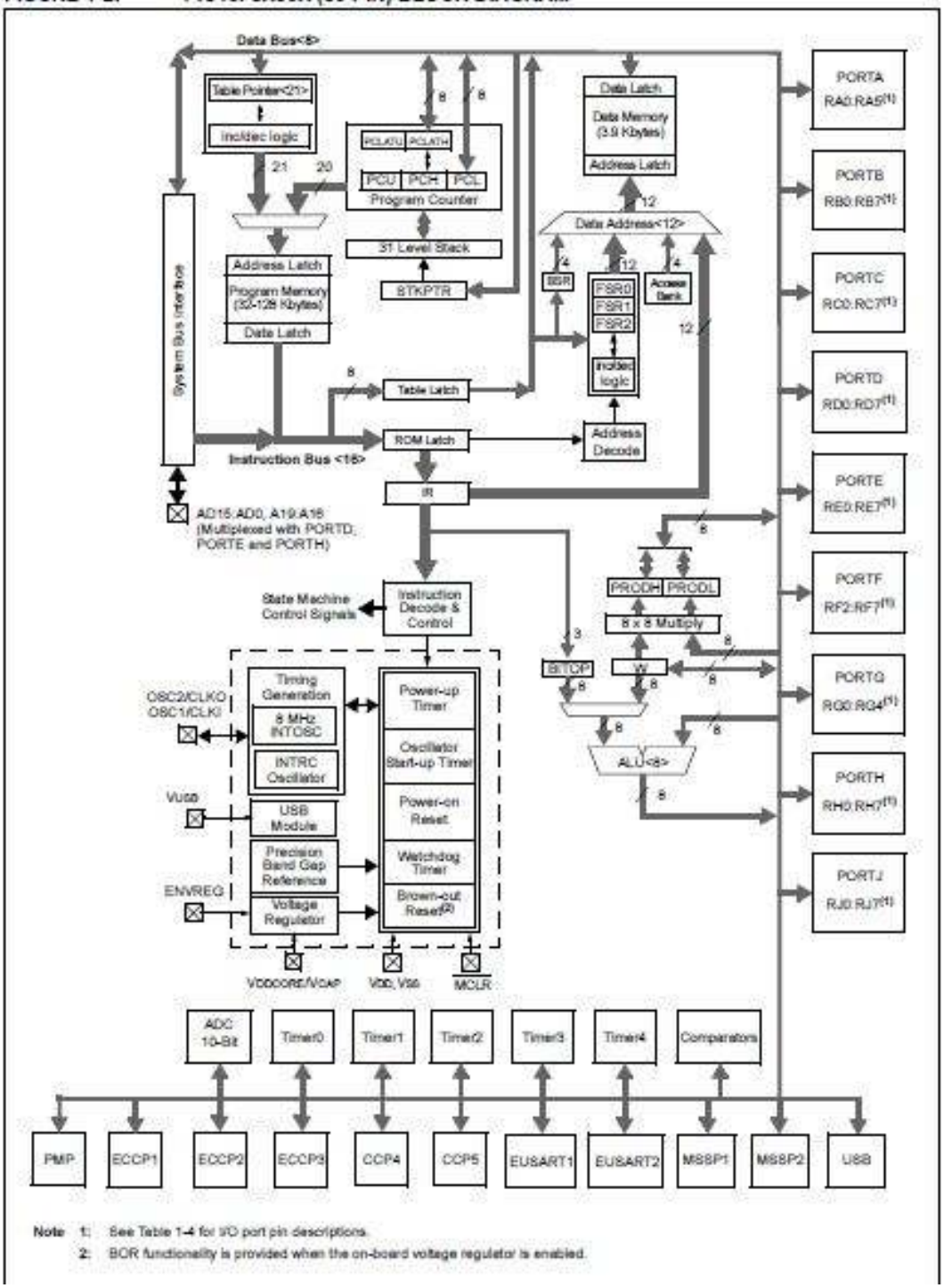
External Memory Bus (80-pin devices only):

- Address Capability of up to 2 Mbytes
- 8-Bit or 16-Bit Interface
- 12-Bit, 16-Bit and 20-Bit Addressing modes

Special Microcontroller Features:

- 5.5V Tolerant Inputs (digital-only pins)
- Low-Power, High-Speed CMOS Flash Technology
- C Compiler Optimized Architecture for Re-Entrant Code
- Power Management Features:
  - Run: CPU on, peripherals on
  - Idle: CPU off, peripherals on
  - Sleep: CPU off, peripherals off
- Priority Levels for Interrupts
- Self-Programmable under Software Control
- 8 x 8 Single-Cycle Hardware Multiplier
- Extended Watchdog Timer (WDT):
  - Programmable period from 4 ms to 131s
- Single-Supply In-Circuit Serial Programming™ (ICSP™) via Two Pins
- In-Circuit Debug (ICD) with 3 Breakpoints via Two Pins
- Operating Voltage Range of 2.0V to 3.6V
- On-Chip 2.5V Regulator
- Flash Program Memory of 10000 Erase/Write Cycles and 20-Year Data Retention

FIGURE 1-2: PIC18F8XJ5X (80-PIN) BLOCK DIAGRAM



Note 1: See Table 1-4 for I/O port pin descriptions.
2: BOR functionality is provided when the on-board voltage regulator is enabled.

TABLE 2-5: OSCILLATOR CONFIGURATION OPTIONS FOR USB OPERATION

Input Oscillator Frequency	PLL Division (PLLDIV2:PLLDIV0)	Clock Mode (FOSC2:FOSC0)	MCU Clock Division (CPDIV1:CPDIV0)	Microcontroller Clock Frequency
48 MHz	N/A	EC	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
48 MHz	÷12 (000)	ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
40 MHz	÷10 (001)	ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
24 MHz	÷6 (010)	HSPLL, ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
24 MHz	N/A[1]	EC, HS	None (11)	**24 MHz**
			÷2 (10)	12 MHz
			÷3 (01)	8 MHz
			÷6 (00)	4 MHz
20 MHz	÷5 (011)	HSPLL, ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
16 MHz	÷4 (100)	HSPLL, ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
12 MHz	÷3 (101)	HSPLL, ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
8 MHz	÷2 (110)	HSPLL, ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz
4 MHz	÷1 (111)	HSPLL, ECPLL	None (11)	48 MHz
			÷2 (10)	**24 MHz**
			÷3 (01)	16 MHz
			÷6 (00)	8 MHz

Legend: All clock frequencies, except 24 MHz, are exclusively associated with full-speed USB operation (USB clock of 48 MHz). **Bold** is used to highlight clock selections that are compatible with low-speed USB operation (system clock of 24 MHz, USB clock of 6 MHz).

TABLE 28-4: INTERNAL VOLTAGE REGULATOR SPECIFICATIONS

Operating Conditions: -40°C < TA < +85°C (unless otherwise stated)							
Param No.	**Sym**	**Characteristics**	**Min**	**Typ**	**Max**	**Units**	**Comments**
	VRGOUT	Regulator Output Voltage	2.45	2.5	—	V	VDD, ENVREG = 3.0V
	CEFC	External Filter Capacitor Value	4.7	10	—	µF	Capacitor must be low-ESR

Annexe B (Datasheet L298N)

L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)

Multiwatt15

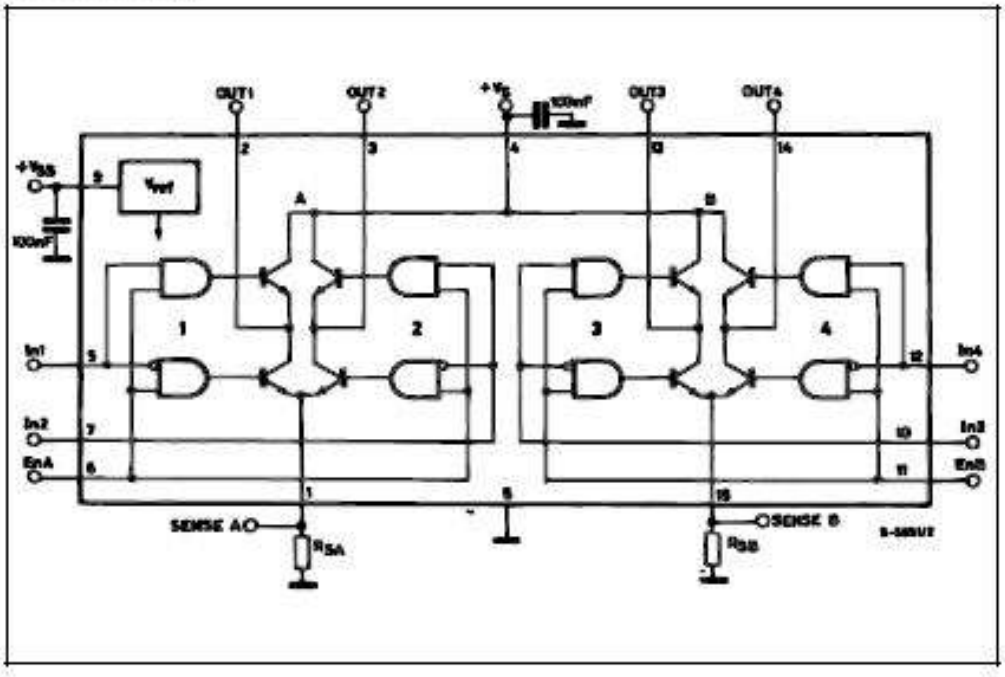
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

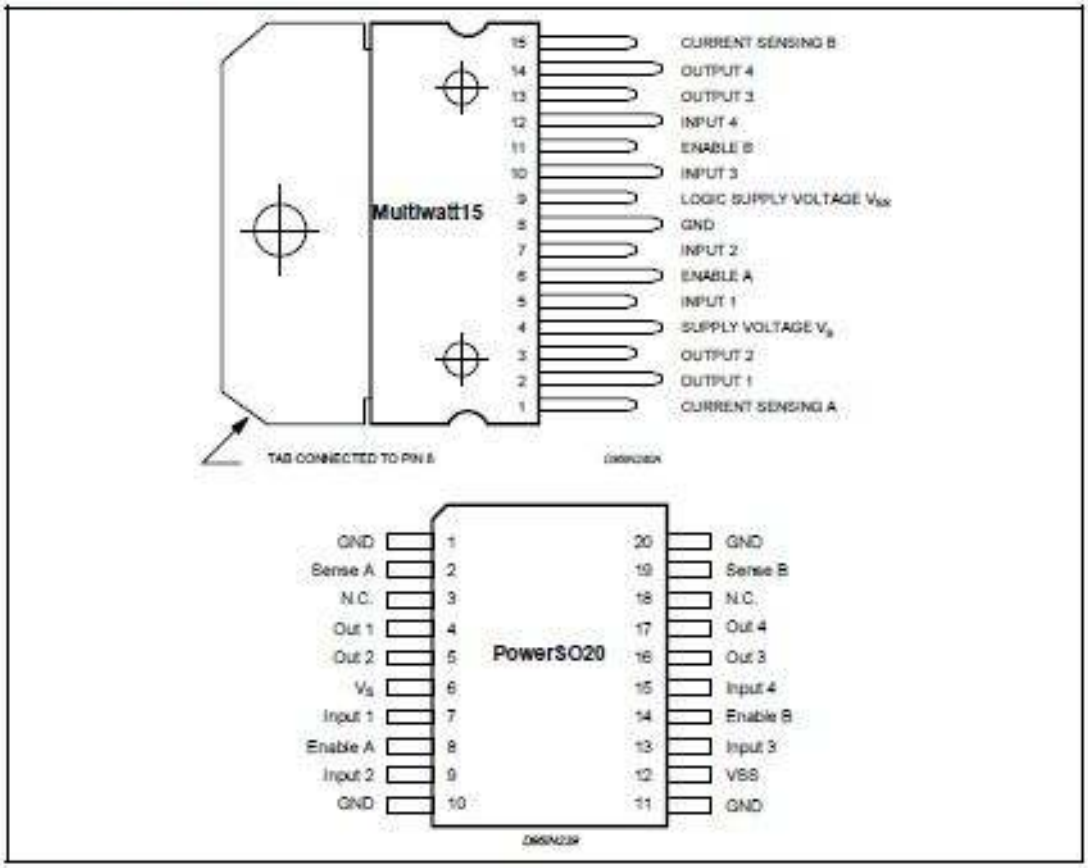
BLOCK DIAGRAM



ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_i, V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)



THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-case}}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-amb}}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

PIN FUNCTIONS (refer to the block diagram)

MW.15	PowerSO	Name	Function
1;15	2;19	Sense A; Sense B	Between this pin and ground is connected the sense resistor to control the current of the load.
2;3	4;5	Out 1; Out 2	Outputs of the Bridge A; the current that flows through the load connected between these two pins is monitored at pin 1.
4	6	V_S	Supply Voltage for the Power Output Stages. A non-inductive 100nF capacitor must be connected between this pin and ground.
5;7	7;9	Input 1; Input 2	TTL Compatible Inputs of the Bridge A.
6;11	8;14	Enable A; Enable B	TTL Compatible Enable Input: the L state disables the bridge A (enable A) and/or the bridge B (enable B).
8	1,10,11,20	GND	Ground.
9	12	VSS	Supply Voltage for the Logic Blocks. A100nF capacitor must be connected between this pin and ground.
10; 12	13;15	Input 3; Input 4	TTL Compatible Inputs of the Bridge B.
13; 14	16;17	Out 3; Out 4	Outputs of the Bridge B. The current that flows through the load connected between these two pins is monitored at pin 15.
–	3;18	N.C.	Not Connected

ELECTRICAL CHARACTERISTICS (V_S = 42V; V_{SS} = 5V, T_j = 25°C; unless otherwise specified)

Symbol	Parameter	Test Conditions		Min.	Typ.	Max.	Unit
V_S	Supply Voltage (pin 4)	Operative Condition		V_{IH} +2.5		46	V
V_{SS}	Logic Supply Voltage (pin 9)			4.5	5	7	V
I_S	Quiescent Supply Current (pin 4)	V_{en} = H; I_L = 0	V_i = L V_i = H		13 50	22 70	mA mA
		V_{en} = L	V_i = X			4	mA
I_{SS}	Quiescent Current from V_{SS} (pin 9)	V_{en} = H; I_L = 0	V_i = L V_i = H		24 7	36 12	mA mA
		V_{en} = L	V_i = X			6	mA
V_{iL}	Input Low Voltage (pins 5, 7, 10, 12)			–0.3		1.5	V
V_{iH}	Input High Voltage (pins 5, 7, 10, 12)			2.3		VSS	V
I_{iL}	Low Voltage Input Current (pins 5, 7, 10, 12)	V_i = L				–10	µA
I_{iH}	High Voltage Input Current (pins 5, 7, 10, 12)	V_i = H ≤ V_{SS} –0.6V			30	100	µA
V_{en} = L	Enable Low Voltage (pins 6, 11)			–0.3		1.5	V
V_{en} = H	Enable High Voltage (pins 6, 11)			2.3		V_{SS}	V
I_{en} = L	Low Voltage Enable Current (pins 6, 11)	V_{en} = L				–10	µA
I_{en} = H	High Voltage Enable Current (pins 6, 11)	V_{en} = H ≤ V_{SS} –0.6V			30	100	µA
$V_{CEsat\ (H)}$	Source Saturation Voltage	I_L = 1A I_L = 2A		0.95	1.35 2	1.7 2.7	V V
$V_{CEsat\ (L)}$	Sink Saturation Voltage	I_L = 1A (5) I_L = 2A (5)		0.85	1.2 1.7	1.6 2.3	V V
V_{CEsat}	Total Drop	I_L = 1A (5) I_L = 2A (5)		1.80		3.2 4.9	V V
V_{sens}	Sensing Voltage (pins 1, 15)			–1 (1)		2	V

ELECTRICAL CHARACTERISTICS (continued)

Symbol	Parameter	Test Conditions	Min.	Typ.	Max.	Unit
$T_1 (V_i)$	Source Current Turn-off Delay	0.5 V_i to 0.9 I_L (2); (4)		1.5		µs
$T_2 (V_i)$	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		0.2		µs
$T_3 (V_i)$	Source Current Turn-on Delay	0.5 V_i to 0.1 I_L (2); (4)		2		µs
$T_4 (V_i)$	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.7		µs
$T_5 (V_i)$	Sink Current Turn-off Delay	0.5 V_i to 0.9 I_L (3); (4)		0.7		µs
$T_6 (V_i)$	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.25		µs
$T_7 (V_i)$	Sink Current Turn-on Delay	0.5 V_i to 0.9 I_L (3); (4)		1.6		µs
$T_8 (V_i)$	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.2		µs
fc (V_i)	Commutation Frequency	I_L = 2A		25	40	KHz
$T_1 (V_{en})$	Source Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (2); (4)		3		µs
$T_2 (V_{en})$	Source Current Fall Time	0.9 I_L to 0.1 I_L (2); (4)		1		µs
$T_3 (V_{en})$	Source Current Turn-on Delay	0.5 V_{en} to 0.1 I_L (2); (4)		0.3		µs
$T_4 (V_{en})$	Source Current Rise Time	0.1 I_L to 0.9 I_L (2); (4)		0.4		µs
$T_5 (V_{en})$	Sink Current Turn-off Delay	0.5 V_{en} to 0.9 I_L (3); (4)		2.2		µs
$T_6 (V_{en})$	Sink Current Fall Time	0.9 I_L to 0.1 I_L (3); (4)		0.35		µs
$T_7 (V_{en})$	Sink Current Turn-on Delay	0.5 V_{en} to 0.9 I_L (3); (4)		0.25		µs
$T_8 (V_{en})$	Sink Current Rise Time	0.1 I_L to 0.9 I_L (3); (4)		0.1		µs

1) 1)Sensing voltage can be −1 V for t ≤ 50 µsec; in steady state V_{sens} min ≥ − 0.5 V.
2) See fig. 2.
3) See fig. 4.
4) The load must be a pure resistor.

Figure 1 : Typical Saturation Voltage vs. Output Current.

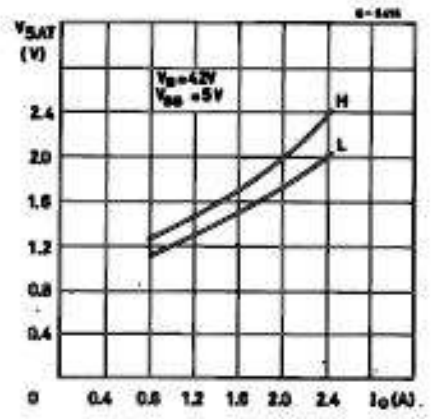
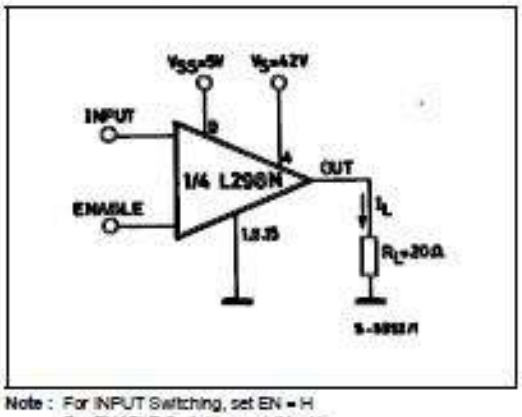


Figure 2 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = H

Figure 3 : Source Current Delay Times vs. Input or Enable Switching.

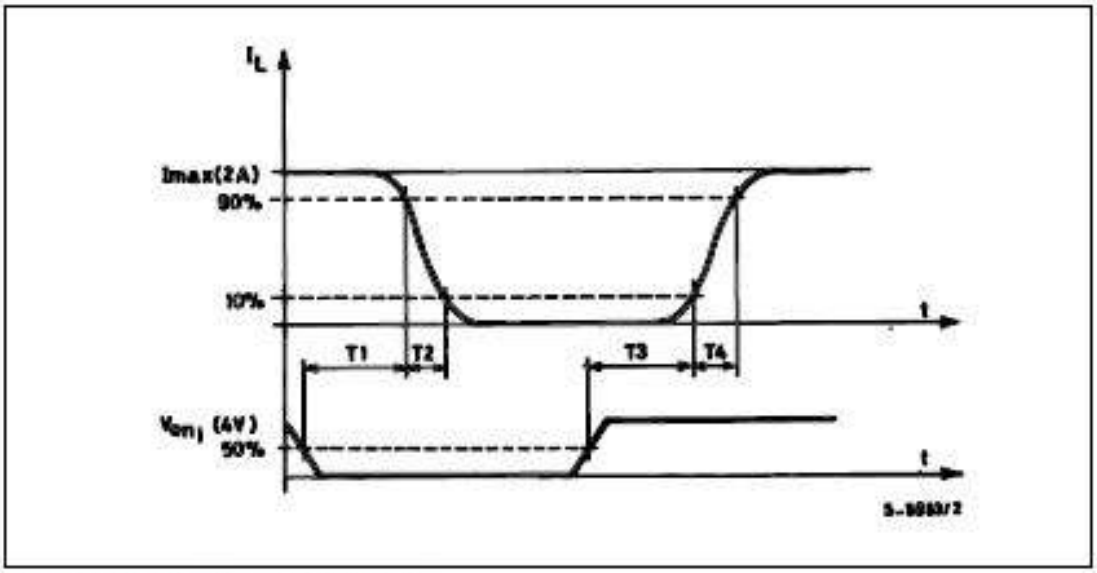
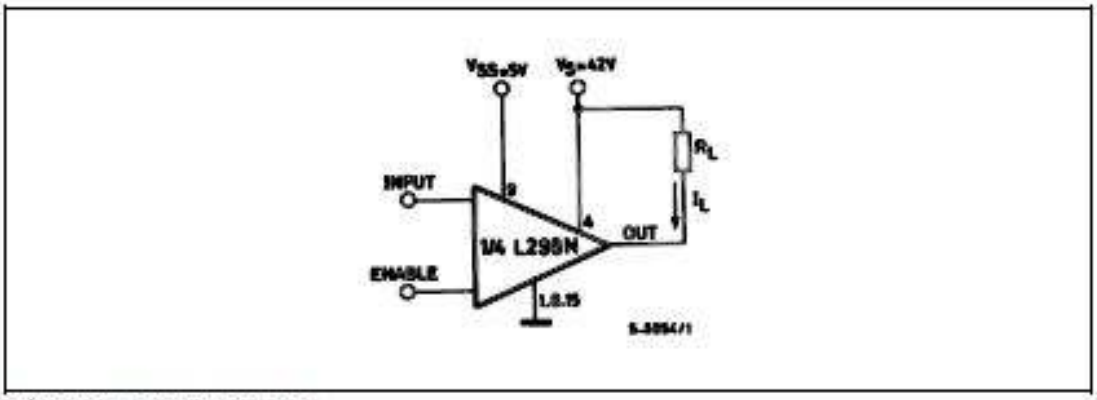


Figure 4 : Switching Times Test Circuits.



Note : For INPUT Switching, set EN = H
For ENABLE Switching, set IN = L

Figure 5 : Sink Current Delay Times vs. Input 0 V Enable Switching.

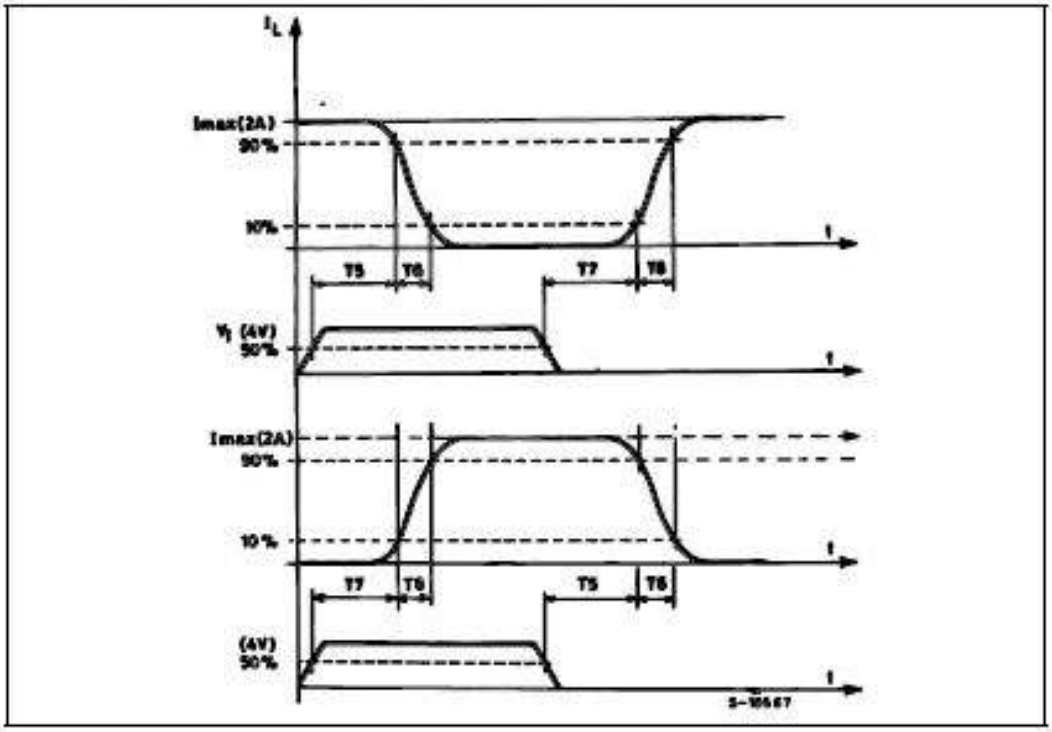
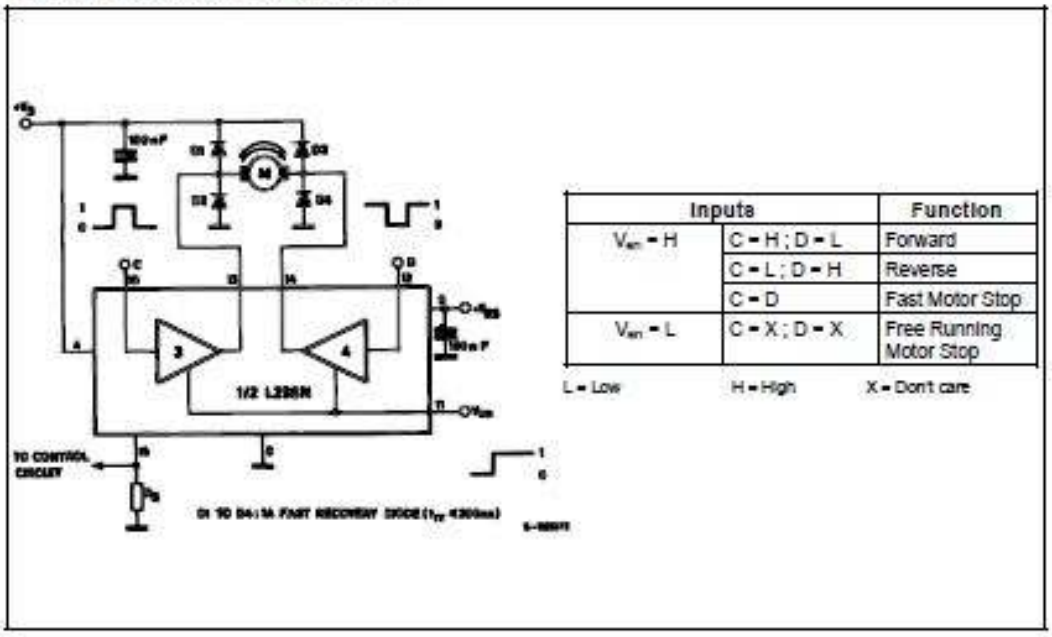


Figure 6 : Bidirectional DC Motor Control.



Inputs		Function
V_{en} = H	C = H ; D = L	Forward
	C = L ; D = H	Reverse
	C = D	Fast Motor Stop
V_{en} = L	C = X ; D = X	Free Running Motor Stop

L = Low H = High X = Don't care

Annexe C (Programmer PicKit2)

PICkit™ 2 User's Guide

The PICkit 2 unit is shown in Figure 1-1.

FIGURE 1-1: PICkit™ 2 MCU PROGRAMMER/DEBUGGER

Legend:

1 – Status LEDs	3 – Lanyard Connection	5 – Pin 1 Marker
2 – Push Button	4 – USB Port Connection	6 – Programming Connector

1.3.1 USB Port Connection

The USB port connection is a USB mini-B connector. Connect the PICkit 2 to the PC using the supplied USB cable.

1.3.2 Status LEDs

The Status LEDs indicate the status of the PICkit 2.

1. **Power** (green) – Power is applied to the PICkit 2 via the USB port.
2. **Target** (yellow) – The PICkit 2 is powering the target device.
3. **Busy** (red) – The PICkit 2 is busy with a function in progress, such as programming.

1.3.3 Push Button

The push button may be used to initiate the Write Device programming function when *Programmer>Write on PICkit Button* is checked on the PICkit 2 Programmer application menu (see item labeled 2 in Figure 1-1.)

The push button may also be used to put the PICkit 2 unit operating system firmware into Bootloader mode. For more information on this feature, see **Chapter 6. "Updating the PICkit 2 Operating System"**.

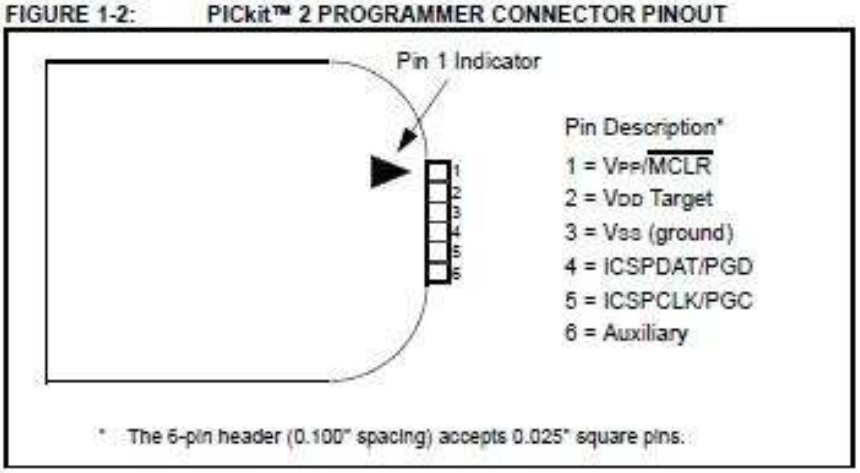
1.3.4 Programming Connector

The programming connector is a 6-pin header (0.100" spacing) that connects to the target device. See the pinout specification in Figure 1-2.

For more information on how to use the PICkit 2 with In-Circuit Serial Programming (ICSP), refer to **Chapter 3. "Using In-Circuit Serial Programming™ (ICSP™)"**.

PICkit 2 Programmer/Debugger Overview

FIGURE 1-2: PICkit™ 2 PROGRAMMER CONNECTOR PINOUT



Note: The programming connector pin functions are different for programming Serial EEPROMS and HCS devices. See the ReadMe file (*Help>Readme*) included with the PICkit 2 programming software for these pinouts.

1.3.5 Lanyard Connection

To help prevent possible loss of the PICkit 2, a convenient lanyard connection is available on the programmer.

Annexe D (Datasheet régulateur 7805)

APPLICATION INFORMATION

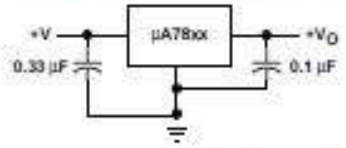


Figure 1. Fixed-Output Regulator

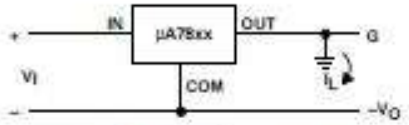
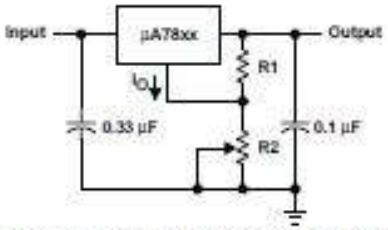


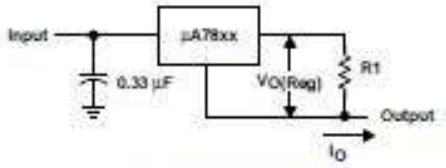
Figure 2. Positive Regulator in Negative Configuration (V_I Must Float)



NOTE A: The following formula is used when V_{XX} is the nominal output voltage (output to common) of the fixed regulator.

$$V_O = V_{xx} + \left(\frac{V_{xx}}{R1} + I_Q\right) R2$$

Figure 3. Adjustable-Output Regulator

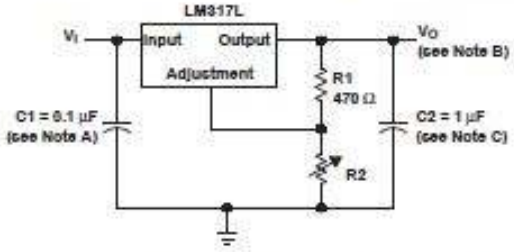


$I_O = (V_O/R1) + I_O$ Bias Current

Figure 4. Current Regulator

Annexe E (Datasheet Régulateur LM317)

APPLICATION INFORMATION



NOTES: A. Use of an input bypass capacitor is recommended if regulator is far from the filter capacitors.

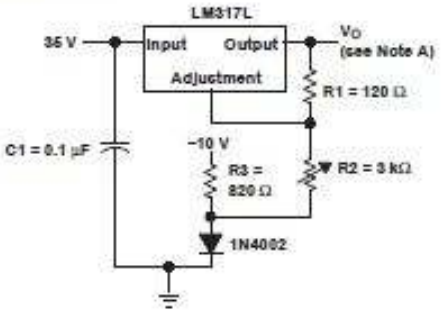
B. Output voltage is calculated from the equation:

$$V_O = V_{ref}\left(1 + \frac{R2}{R1}\right)$$

where: V_{ref} equals the difference between OUTPUT and ADJUSTMENT voltages (=1.25 V).

C. Use of an output capacitor improves transient response, but is optional.

Figure 1. Adjustable Voltage Regulator

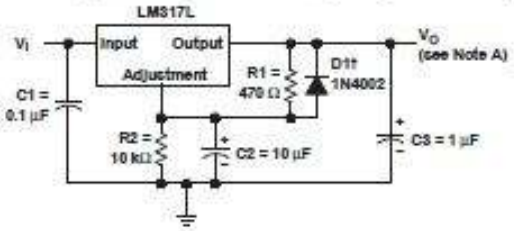


NOTE A: Output voltage is calculated from the equation:

$$V_O = V_{ref}\left(1 + \frac{R2 + R3}{R1}\right) - 10\ V$$

where: V_{ref} equals the difference between OUTPUT and ADJUSTMENT voltages (=1.25 V).

Figure 2. 0-V to 30-V Regulator Circuit



† D1 discharges C2 if output is shorted to ground.

NOTE A: Use of an output capacitor improves transient response, but is optional.

Figure 3. Regulator Circuit With Improved Ripple Rejection

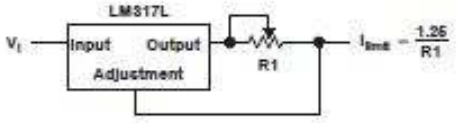


Figure 4. Precision Current-Limiter Circuit

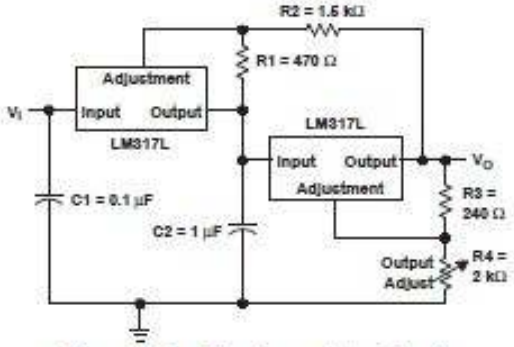


Figure 5. Tracking Preregulator Circuit

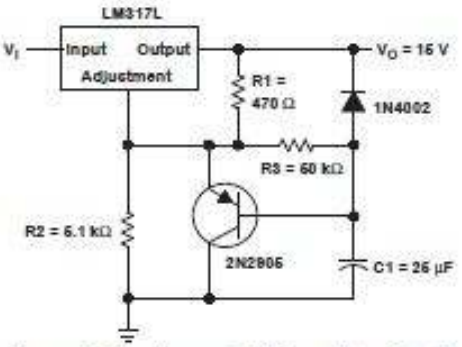


Figure 6. Slow-Turnon 15-V Regulator Circuit

Annexe F (PID/VID)

Acquiring a VID and PID

In order to sell or market a USB product you must have a VID. This means that you must purchase one from USBIF. It is important to make sure that you get a Vendor ID for your company to avoid legal consequences. Once you purchase a VID, you can use it across all USB devices that you produce. The Product ID is not purchased but chosen by the developer or company. Each product needs its own VID/PID combination.

Annexe G (Code pour classe Wrapper)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Runtime.InteropServices;
using System.Threading;

namespace ProbingMachine
{
       #region "Enumerations and Structures"

    public enum MPResult
    {
        MPUSB_FAIL = 0,
        MPUSB_SUCCES = 1,
        MPUSB_SUCCESS_LESS_DATA = 2
    }

    public struct FWInfo
    {
        public int FWDeviceID;
        public int FWRev;
        public string DateAndTime;
    }

    public enum MotorDirection
    { X_Right = 0, X_Left = 1, Y_Right = 2, Y_Left = 3, Z_UP = 4, Z_Down = 5, XY = 6 };

    public struct Point
    {
        public string Name;
        public int X;
        public int Y;
        public int Z;
        public double Voltage;
    }

    #endregion


  public class Wrapper
    {
        int ClassInstance = 0;  // class instance 0
        public bool isConnected = false;

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "OpenUSBPort")]
        private static extern MPResult apiOpenUSBPort(int Instance);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "CloseUSBPort")]
        private static extern bool apiCloseUSBPort(int Instance);
```

```
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetMotor")]
        private static extern MPResult apiSetMotor(int Instance, int Direction, int
Steps);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetMotorXY")]
        private static extern MPResult apiSetMotorXY(int Instance, int Mode, int
XDirection, int YDirection, int XStep, int YStep);

        ////////////////////////// L1 API ////////////////////

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "About")]
        private static extern void apiAbout();

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "GetDeviceCount")]
        private static extern uint apiGetDeviceCount();

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Ping")]
        private static extern MPResult apiPing(int Instance, byte[] inBuff);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Reset_MCU")]
        private static extern MPResult apiReset_MCU(int Instance);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Reset_MCU_Boot")]
        private static extern MPResult Reset_MCU_Boot(int Instance);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetDeviceID")]
        private static extern MPResult apiSetDeviceID(int Instance, byte D_ID);

        //Direct IO
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetPort")]
        private static extern MPResult apiSetPort(int Instance, byte t_Port, byte Val);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "GetPort")]
        private static extern MPResult apiGetPort(int Instance, byte t_Port, ref byte
Val);


        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "I2C_Write")]
        private static extern MPResult apiI2C_Write(int Instance, byte SlaveAddress,
byte[] DtBuff, byte Length, byte I2CStart, byte I2CStop);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "I2C_Read")]
        private static extern MPResult apiI2C_Read(int Instance, byte SlaveAddress, byte
Length, byte[] inBuff, byte I2CStart, byte I2CStop);
```

```
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "EEPROM_Write")]
        private static extern MPResult apiEEPROM_Write(int Instance, byte block, byte
Address, byte Data);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "EEPROM_Read")]
        private static extern MPResult apiEEPROM_Read(int Instance, byte block, byte
Address, ref byte Data);

        //ADC
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "GetIntADC")]
        private static extern MPResult apiGetIntADC(int Instance, byte ADCchannel, ref
byte receive_buffer);

        //power
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetPower")]
        private static extern MPResult apiSetPower(int Instance, byte rate);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetFan_ON_OFF")]
        private static extern MPResult apiSetFan_ON_OFF(int Instance, byte Enable);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Set_PWM")]
        private static extern MPResult apiSet_PWM(int Instance, byte channel, byte
value);

        //analog

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "GetCurrent_sens")]
        private static extern MPResult apiGetCurrent_sens(int Instance, ushort Address,
ref double Data);
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Get_VCC")]
        private static extern MPResult apiGet_VCC(int Instance, ref double Data);
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Get_VCCTX")]
        private static extern MPResult apiGet_VCCTX(int Instance, ref double Data);
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Get_VCCRX")]
        private static extern MPResult apiGet_VCCRX(int Instance, ref double Data);
        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Get_temperature")]
        private static extern MPResult apiGet_temperature(int Instance, ref double Data);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "Get_VCCProb")]
        private static extern MPResult apiGet_VCCProb(int Instance, ref double Data);

        [DllImport(@"C:\Users\Georges\Desktop\Mémoire_070114\GUI\George sarkis
project\Probing machine_API\debug\Probing_Machine.dll", EntryPoint = "SetDrill_ON_OFF")]
        private static extern MPResult apiSetDrill_ON_OFF(int Instance, byte Enable);

        #region "USB Connection"
```

```
public MPResult OpenUSBPort()
{
    MPResult rc;
    rc = apiOpenUSBPort(ClassInstance);
    if (rc == MPResult.MPUSB_SUCCES)
    {
        isConnected = true;
    }
    return rc;
}

public MPResult CloseUSBPort()
{
    bool oK;
    oK = apiCloseUSBPort(ClassInstance);
    isConnected = !oK;
    if (!oK)
    {
        return MPResult.MPUSB_FAIL;
    }
    else
    {
        return MPResult.MPUSB_SUCCES;
    }

}
#endregion

#region "L1"

public static UInt32 GetDeviceCount()
{
    return apiGetDeviceCount();
}

public MPResult Ping(ref FWInfo PingInfo)
{
    MPResult rc = MPResult.MPUSB_FAIL;
    try
    {
        byte[] inBuff = new byte[25];
        rc = apiPing(ClassInstance, inBuff);
        if (rc == MPResult.MPUSB_SUCCES)
        {
            PingInfo.FWDeviceID = inBuff[0];
            PingInfo.FWRev = inBuff[1];
            for (int i = 2; i < 21; i++)
            {
                PingInfo.DateAndTime += (char)inBuff[i];
            }
        }
    }

    catch
    {
        //if (QSFPuserGUI.MainWindow.autolog)
        //{
```

```
                //    QSFPuserGUI.MainWindow.sw.WriteLine("Line = " + ex.StackTrace + "
Message = " + ex.Message);
                //    QSFPuserGUI.MainWindow.sw.Flush();
                //    return MPResult.MPUSB_FAIL;
                //}
            }
            return rc;
        }

        public MPResult Reset_MCU()
        {
            return apiReset_MCU(ClassInstance);
        }

        public MPResult SetDeviceID(byte D_ID)
        {
            return apiSetDeviceID(ClassInstance, D_ID);
        }

        #endregion

        #region "Direct IO"
        public MPResult SetPort(byte t_port, byte val)
        {
            return apiSetPort(ClassInstance, t_port, val);
        }

        public MPResult GetPort(byte t_port, ref byte val)
        {
            return apiGetPort(ClassInstance, t_port, ref val);
        }

        #endregion

        #region "I2C"

        public MPResult I2C_write(byte SlaveAddress, byte Register, byte Val)//write one
byte at a time
        {
            if (!isConnected)
            {
                return MPResult.MPUSB_FAIL;
            }

            MPResult rc;
            byte[] DtBuff = new byte[2];
            DtBuff[0] = Register;
            DtBuff[1] = Val;
            rc = apiI2C_Write(ClassInstance, SlaveAddress, DtBuff, (byte)DtBuff.Length,
1, 1);// write the register address and the value to be written in that address
            Thread.Sleep(5);
            return rc;
        }

        public MPResult I2C_Read(byte SlaveAddress, byte registerAddress, ref byte[]
ReadBuff)// read 32 byte/packet
        {
            if (!isConnected)
```

```
            {
                return MPResult.MPUSB_FAIL;
            }

            MPResult rc;
            byte[] WriteBuff = new byte[32];
            WriteBuff[0] = registerAddress;
            rc = apiI2C_Write(ClassInstance, SlaveAddress, WriteBuff, 1, 1, 0);// write
the register address we want to read from
            rc = apiI2C_Read(ClassInstance, SlaveAddress, (byte)(ReadBuff.Length),
ReadBuff, 1, 1);
            Thread.Sleep(5);
            return rc;
        }

        #endregion

        public string About()
        {
            apiAbout();
            return "";
        }

        public bool SetMotor(MotorDirection Direction, int Steps)
        {
            return apiSetMotor(ClassInstance, (int)Direction, Steps) ==
MPResult.MPUSB_SUCCES ? true : false;
        }

        public bool SetMotorXY(MotorDirection Direction, MotorDirection XDirection,
MotorDirection YDirection, int XStep, int YStep)
        {
            return apiSetMotorXY(ClassInstance, (int)Direction, (int)XDirection,
(int)YDirection, XStep, YStep) == MPResult.MPUSB_SUCCES ? true : false;
        }

        public bool Get_VCCProb(ref double Data)
        {
            return apiGet_VCCProb(ClassInstance, ref Data) == MPResult.MPUSB_SUCCES ?
true : false;
        }

        public bool SetDrill_ON_OFF(byte Enable)
        {
            return apiSetDrill_ON_OFF(ClassInstance,  Enable) == MPResult.MPUSB_SUCCES ?
true : false;
        }
    }
}
```

Annexe H (Code pour classe Connection manager)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;

namespace ProbingMachine
{
    public class ConnectionManager
    {
        Wrapper wrapper;

        public ConnectionManager(Wrapper Wrapper)
        {
            this.wrapper = Wrapper;
        }

        public bool ISConnected()
        {
            return wrapper.isConnected;
        }

        public bool Connnect()
        {
            Thread MulThread = new Thread(delegate() { });

            bool threadStatus = true;

            MulThread = new Thread(delegate()
            {
                threadStatus = true;

                if (wrapper.OpenUSBPort() == MPResult.MPUSB_SUCCES)
                {
                    threadStatus = false;
                }
            });
            MulThread.Start();
            MulThread.Join(2000);

            if (threadStatus == true)
            {
                MulThread.Abort();
                return false;
            }
            else
            {
                return true;
            }
        }

        public bool Disconnect()
        {
            return wrapper.CloseUSBPort() == MPResult.MPUSB_SUCCES ? true : false;
```

```
        }

        public bool Reset_MCU()
        {
            return wrapper.Reset_MCU() == MPResult.MPUSB_SUCCES ? true : false;
        }

        public bool SetDeviceID(byte D_ID)
        {
            return wrapper.SetDeviceID(D_ID) == MPResult.MPUSB_SUCCES ? true : false;
        }

        public bool SetPort(byte t_port, byte val)
        {
            return wrapper.SetPort(t_port, val) == MPResult.MPUSB_SUCCES ? true : false;
        }

        public bool GetPort(byte t_port, ref byte val)
        {
            return wrapper.GetPort(t_port, ref val) == MPResult.MPUSB_SUCCES ? true :
false;
        }

        public bool I2C_write(byte SlaveAddress, byte Register, byte Val)//write one byte
at a time
        {
            return wrapper.I2C_write(SlaveAddress, Register, Val) ==
MPResult.MPUSB_SUCCES ? true : false;
        }

        public bool I2C_Read(byte SlaveAddress, byte registerAddress, ref byte[]
ReadBuff)// read 32 byte/packet
        {
            return wrapper.I2C_Read(SlaveAddress, registerAddress, ref ReadBuff) ==
MPResult.MPUSB_SUCCES ? true : false;
        }

        public string About()
        {
            return wrapper.About();
        }

        public bool SetMotor(MotorDirection Direction, int Steps)
        {
            return wrapper.SetMotor(Direction, Steps);
        }

        public bool SetMotorXY(MotorDirection Direction, MotorDirection XDirection,
MotorDirection YDirection, int XStep, int YStep)
        {
            return wrapper.SetMotorXY(Direction, XDirection, YDirection, XStep, YStep);
        }

        public bool Get_VCCProb(ref double Data)
        {
            return wrapper.Get_VCCProb(ref Data);
        }
```

```
        public bool SetDrill_ON_OFF(bool Enable)
        {
            byte enable = Enable == true ? (byte)1 : (byte)0;
            return wrapper.SetDrill_ON_OFF(enable);
        }
    }
}
```

Annexe I (Code pour function Load file)

```
        private void btn_StartLoadFile_Click(object sender, EventArgs e)
        {
            if (!Connection.ISConnected())
            {
                MessageBox.Show("Connect to the board first via the USB Interface",
"Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            if (fileisLoaded == false)
            {
                MessageBox.Show("Please load File before start the test", "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }

            string fileName;
            string []arraytemp = new string[50];

            fileName = loadFile.FileName;
            int counter = 0;
            arraytemp = fileName.Split(new char[]{'\\'});
            for (int i = 0; i < arraytemp.Length; i++)
            {

                if (arraytemp[i] == "")
                {
                    break;
                }
                counter++;
            }
            fileName = arraytemp[counter - 1];


            double[] VoltageData = new double[LoadScript.Count];
            string[] Result = new string[LoadScript.Count];
            rtbox_result.Clear(); //Document.Blocks.Clear();
            rtbox_result.AppendText("          PROBING MACHINE FOR TEST POINT INFORMATION
FILE\n");
            rtbox_result.AppendText("\n");
            rtbox_result.AppendText("======================----------------------
======================\n");
            rtbox_result.AppendText(" Test points results for" + fileName + "\n");
            rtbox_result.AppendText("Name\t xValue\t yValue\t real Voltage \t\t tested
Voltage\tStatus\n");
            double voltage_compare = 0;
            IntializeValue();
            Thread.Sleep(1000);

            for (int i = 0; i < LoadScript.Count(); i++)
            {
                MoveToXY(LoadScript[i].X, LoadScript[i].Y);
                //if (LoadScript[i].X > LoadScript[i].Y)
                //{
```

```
                //    Thread.Sleep(LoadScript[i].X * 200);
                //}
                //else
                //{
                //    Thread.Sleep(LoadScript[i].Y * 200);
                //}
                Thread.Sleep(1500);
                MoveZ(-10);
                Thread.Sleep(500);
                Connection.Get_VCCProb(ref VoltageData[i]);//Connection.Get_VCCProb(ref
Data);
                Thread.Sleep(500);
                MoveZ(10);
                Thread.Sleep(1000);
                //to hide and lable part of the code
                #region Printing the Location
                currentLocation.X = LoadScript[i].X;
                currentLocation.Y = LoadScript[i].Y;
                rtbox_PositionsStatus.Clear();
                rtbox_PositionsStatus.AppendText("X : " + currentLocation.X.ToString() +
"\n");
                rtbox_PositionsStatus.AppendText("Y : " + currentLocation.Y.ToString() +
"\n");
                Application.DoEvents();// do the cmd stack
                #endregion

                #region Affichage
                voltage_compare = VoltageData[i] - LoadScript[i].Voltage;
                if (VoltageData[i] < LoadScript[i].Voltage + 0.1 && VoltageData[i] >
LoadScript[i].Voltage - 0.1)
                //if (voltage_compare > -0.1 || voltage_compare < 0.1)
                {
                    Result[i] = "pass";
                }
                else
                {
                    Result[i] = "Failed";
                }
                rtbox_result.AppendText(LoadScript[i].Name + "\t" +
LoadScript[i].X.ToString() + "\t\t" + LoadScript[i].Y.ToString() + "\t\t" +
LoadScript[i].Voltage.ToString() + "\t\t\t" + VoltageData[i].ToString() + "\t" +
Result[i] + "\n");
                Application.DoEvents();// do the cmd stack
                #endregion
                //Thread.Sleep(8 * 300);
            }



            saveFileLocation = new SaveFileDialog();  // used for saving the file
location
            saveFileLocation.Title = "Save Script";
            saveFileLocation.Filter = "Comma Separated Values (.csv)|*.csv";
            string[] temp = new string[2];
            temp = loadFile.SafeFileName.Split(new char[] { '.' });
            string name = temp[0];

            saveFileLocation.FileName = name + "_SN";
```

```
            Application.DoEvents();
            if (saveFileLocation.ShowDialog() == System.Windows.Forms.DialogResult.OK)
            {
                StreamWriter saveFile = new StreamWriter(saveFileLocation.FileName);
                saveFile.WriteLine("Report test for board:  " + fileName);
                saveFile.WriteLine("-------------------------------------------------------
--------------");
                DateTime thisDay = DateTime.Today;
                saveFile.WriteLine("Date:  ");
                saveFile.WriteLine(thisDay.ToString("D"));
                saveFile.WriteLine("-------------------------------------------------------
--------------");
                saveFile.WriteLine("Name,X, Y, Real voltage, Tested Voltage");

                for (int i = 0; i < LoadScript.Count(); i++)
                {
                    saveFile.WriteLine(LoadScript[i].Name.ToString() + "," +
LoadScript[i].X.ToString() + "," + LoadScript[i].Y.ToString() + "," +
LoadScript[i].Voltage.ToString() + "," + VoltageData[i].ToString());
                }

                saveFile.Flush();   // to force the proccess to write into the file
                saveFile.Close();    // to close the proccess and to be accessible fron
anther one
                saveFile.Dispose();
            }
            else
            {
                return;
            }


            bool TestStatus = true;

            for (int i = 0; i < LoadScript.Count; i++)
            {
                if (Result[i] == "Failed")
                {
                    TestStatus = false;
                }
            }

            if (TestStatus == false)
            {
                MessageBox.Show("The baord is not working check the list of voltages",
"Test Failed", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            else
            {
                MessageBox.Show("The baord is working", "Test Succeded",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            }

            #region Saving Condition
            if (MessageBox.Show("Would you like to print this report", "Print",
MessageBoxButtons.OKCancel, MessageBoxIcon.Question) ==
System.Windows.Forms.DialogResult.OK)
            {
```

```
                content = System.IO.File.ReadAllText(saveFileLocation.FileName);
                PrintDialog printDlg = new PrintDialog();
                PrintDocument printDoc = new PrintDocument();
                printDoc.DocumentName = saveFileLocation.FileName;
                printDlg.Document = printDoc;
                printDlg.AllowSelection = true;
                printDlg.AllowSomePages = true;
                //Call ShowDialog
                if (printDlg.ShowDialog() == DialogResult.OK)
                {
                    printDoc.PrintPage += new PrintPageEventHandler(pd_PrintPage);
                    printDoc.Print();
                }
            }
            #endregion
        }

        String content = "";
        Font printFont = new Font("Arial", 12, FontStyle.Regular);









private void pd_PrintPage(object sender, PrintPageEventArgs ev)
        {
            ev.Graphics.DrawString(content, printFont, Brushes.Black,
                                ev.MarginBounds.Left, 0, new StringFormat());
        }
```