République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Abou bekr Belkaid – Tlemcen

Faculté de Technologie

Département de Génie Electrique et Electronique

Filière : Instrumentation Electronique

Projet de Fin d'Etudes

Master : Master II

Option : Instrumentation Electronique

Intitulé :

ETUDE ET REALISATION D'UNE CARTE DE COMMANDE DE TROIS MOTEURS PAS A PAS POUR MACHINE DE PROTOTYPAGE RAPIDE

Présenté par : Mr Taleb Bendiab Zakaria

Jury :

Président : Mr Benallal Mohamed (MAA) (Université de Tlemcen)

Encadreur : Mr Nemiche Ahmed (MAA) (Université de Tlemcen)

Examinateur : Mr Bechar Hassan (MAA) (Université de Tlemcen)

Année Universitaire : 2013 /2014

بسم الله الرحمن الرحيم

«و ما أتيتم من العلم الا قليلا»

صدق الله العظيم

Je dédie ce modeste travail :

A ma femme et mes trois enfants :

Pour tous les sacrifices qu'ils ont faits pour moi, pour leur soutien continu durant mon travail. Que dieu vous protège.

A mes frères et sœurs :

Que Dieu vous garde, Je vous aime et je vous souhaite une vie pleine de succès et de réussite.

A la famille de ma femme :

Qui m'ont soutenue durant toute cette période, dieu vous garde.

Ainsi qu'à mon gendre Sidi Mohamed, que dieu te protège et te guide dans le droit chemin.

Et enfin a tout ce ceux qui m'ont aidé de près ou de loin dans mon travail ainsi qu'à tous les membres de la promotion de Master 2 en instrumentation électronique 2013 /2014

Tout d'abord,

Je tiens à remercier et glorifier Dieu le tout puissant et miséricordieux d'avoir guidé mes pas avec clairvoyance pour accomplir ce travail et de le mener à terme.

Je tiens à remercier aussi mon promoteur,

Monsieur Ahmed NEMICHE

D'avoir accepté de m'encadrer pour l'exécution de ce Projet.

Je tiens à remercier aussi,

Monsieur Boumédiène BELARBI

Qui a bien voulu me guider tous le long de ce mémoire, Avec sa grande disponibilité, sa rigueur scientifique et ses précieux conseils qui m'ont permis de travailler dans de meilleures conditions.

Par ailleurs, Mes remerciements, à Monsieur Benallal Mohamed qui m'a fait l'honneur, de bien vouloir accepter de présider le jury ainsi qu'à Monsieur Bechar Hassan en tant qu'examinateur.

Et enfin, je tiens à remercier tous les membres du département de Génie Electrique et Electronique et de la faculté de Technologie de L'université Abou Bekr Belkaid qui m'ont aidé dans mon travail.

Sommaire

Table des Figures 4
Table des Tableaux 6
Résumé 7
Introduction Générale 8

Chapitre I : Les Moteurs Pas A Pas (Caractéristiques et Mode de Commande)

Introduction 10
1.1. Généralités 12
1.2. Technologie des moteurs pas à pas 13
1.2.1. Moteurs pas à pas rotatifs 13
1.2.1.1. Moteur pas à pas rotatif à reluctance variable 13
1.2.1.2. Moteur pas à pas rotatif à aimant permanent 15
1.2.1.2.1. Les moteurs unipolaires 17
1.2.1.2.2. Les moteurs bipolaires 19
1.2.1.3. Moteur pas à pas rotatif hybrides 21
1.2.2. Moteurs pas à pas linéaires 23
1.2.2.1. Moteur pas à pas linéaire à reluctance variable 24
1.2.2.2. Moteur pas à pas linéaire polarises 24
1.2.2.2.1. Les moteurs à aimant permanent 24
1.2.2.2.2. Les moteurs hybrides 24
Conclusion 25

Chapitre II : L'Interface USB (Approche Théorique et Pratique)

Introduction 26
2.1. Caractéristiques générales 28
2.2. Caractéristiques mécaniques 30

2.3. Caractéristiques électriques 32

2.4. Identification de la vitesse 33
2.5. L'alimentation USB (V_{bus}) 34
2.6. Courant de veille 36
2.7. Accès au mode veille 36
2.8. Les protocoles USB 37
2.9. Les champs de paquet USB ordinaires 37
2.9.1. Le champs SYNC 38
2.9.2. Le champs PID 38
2.9.3. Le champs ADDR 39
2.9.4. Le champs ENDP 39
2.9.5. Le champs CRC 39
2.9.6. Le champs EOP 40
2.10. Les types de paquet USB 40
2.10.1. Les paquets jetons 40
2.10.2. Les paquets de données 40
2.10.3. Les paquets " poignée de mains " 40
2.10.4. Les paquets début de trame (SOF) 40
2.11. Les fonctions USB 41
2.12. Les terminaisons (Endpoint) 43
2.13. Les canaux de communications (Pipes) 43
2.14. Les types de terminaisons (Endpoints) 43
2.14.1. Les transferts de commande 44
2.14.1.1. L'étape d'installation (Setup Stage) 44
2.14.1.2. L'étape de données (Data Stage) 45
2.14.1.3. L'étape d'état (Status Stage) 46
2.14.2. Les transferts d'interruption 49
2.14.3. Les transferts Isochrones 51
2.14.4. Les transferts En Bloc (BULK) 52

2.15. La gestion de la bande passante 54

2.16. Les descripteurs USB 54
2.16.1. Le descripteur d'appareil (Device Descriptor) 57
2.16.2. Le descripteur de configuration (Configuration Descriptor) 58
2.16.3. Le descripteur d'interface (Interface Descriptor) 60
2.16.4. Le descripteur de terminaison (Endpoint Descriptor) 61
2.16.5. Le descripteur de chaine de caractères (String Descriptor) 63
2.17. Le paquet d'installation 64
2.17.1. Les requêtes d'appareil standard 66
2.17.2. Les requêtes d'interface standard 68
2.17.3. Les requêtes de terminaison standard 69
2.18. L'énumération 70
Conclusion 72

Chapitre III : La Carte d'Interface (Théorie et Réalisation Pratique)

Introduction 73
3.1. Structure générale de la carte 74
3.2. L'unité de contrôle 76
3.3. L'unité de commande de moteur pas à pas 77
3.3.1. Présentation du circuit intégré L297 78
3.3.2. Présentation du circuit intégré L298 80
3.4. Software et programmation avancé 81
3.4.1. Programmation du PIC18F4550 82
3.4.2. Programmation VISUAL C++ 85
3.5. Réalisation du circuit imprimé 87
Conclusion 88

Conclusion générale 89

Bibliographie 90

Table Des Figures

Figure 1.1 : Imprimante 3D-EXTRU3D (à gauche) et sa Carte d'interface (à droite) 10

Figure 1.2 : Vue d'ensemble d'un moteur pas à pas 12

Figure 1.3 : Structure classique d'un moteur à reluctance variable rotatif double saillance 14

Figure 1.4 : Structure interne d'un moteur à aimant permanent 16

Figure 1.5 : Fonctionnement schématique d'un moteur pas à pas 17

Figure 1.6 : Représentation schématique d'un moteur unipolaire 18

Figure 1.7 : Séquence de rotation simple 18

Figure 1.8 : Représentation schématique d'un moteur bipolaire 19

Figure 1.9 : Séquences de commande d'un moteur bipolaire 21

Figure 1.10 : Moteur pas à pas hybride 21

Figure 1.11 : Structure interne d'un moteur hybride 22

Figure 2.12 : Topologie du bus USB 29

Figure 2.13 : Codage NRZI 30

Figure 2.14 : Différents types de connecteurs USB 31

Figure 2.15 : Câble USB 32

Figure 2.16 : Appareil pleine vitesse avec résistance de rappel état haut branché sur D+ 33

Figure 2.17 : Appareil pleine vitesse avec résistance de rappel état haut branché sur D- 34

Figure 2.18 : Structure du champ PID 39

Figure 2.19 : Structure du champ ADDR 39

Figure 2.20 : Structure du champ ENDP 39

Figure 2.21 : Structure d'un paquet jeton 40

Figure 2.22 : Structure d'un paquet de données 40

Figure 2.23 : Structure d'un paquet « poignée de mains » 41

Figure 2.24 : Structure d'un paquet début de trame 41

Figure 2.25 : Exemple de fonction USB 42
Figure 2.26 : Etape d'installation 45
Figure 2.27 : Etape de données 46
Figure 2.28 : Etape d'état (entrée) 47
Figure 2.29 : Etape d'état (sortie) 47
Figure 2.30 : Transaction d'interruption d'entée (IN) et de sortie (Out) 50
Figure 2.31 : Transaction Isochrone d'entée (IN) et de sortie (Out) 52
Figure 2.32 : Transaction en bloc d'entée (IN) et de sortie (Out) 53
Figure 2.33 : Hiérarchie des différents descripteurs USB 55
Figure 2.34 : Diagramme d'état d'un appareil USB 71
Figure 3.35 : Schéma synoptique de la carte de commande 75
Figure 3.36 : Brochage du PIC18F4550-TQFP44 76
Figure 3.37 : Brochage et structure interne du circuit L297 79
Figure 3.38 : Brochage et structure interne du circuit L298 81
Figure 3.39 : Structure des fichiers constituant le programme pour le PIC18F4550 82
Figure 3.40 : Organigramme du programme pour le PIC18F4550 84
Figure 3.41 : Simulation du Programme sur PROTEUS-ISIS 85
Figure 3.42 : Interface utilisateur pour le programme de commande 86

Table Des Tableaux

Tableau 1.1 : Ordre des différentes commandes pour moteur à reluctance variable 15

Tableau 1.2 : Ordre des différentes commandes pour moteur unipolaire 19

Tableau 1.3 : Ordre des différentes commandes pour moteur bipolaire 20

Tableau 2.4 : Brochage des différents connecteurs USB 32

Tableau 2.5 : Les différentes valeurs du PID 38

Tableau 2.6 : Format du descripteur d'appareil 57

Tableau 2.7 : Format du descripteur de configuration 59

Tableau 2.8 : Format du descripteur d'interface 60

Tableau 2.9 : Format du descripteur de terminaison 62

Tableau 2.10 : Format du descripteur de chaîne Zéro 63

Tableau 2.11 : Format du descripteur de chaîne 64

Tableau 2.12 : Format du paquet d'installation 65

Tableau 2.13 : Requêtes d'appareils standards 67

Tableau 2.14 : Requêtes d'interfaces standards 68

Tableau 2.15 : Requêtes de terminaisons standards 69

Résumé

Le prototypage rapide est une technologie extrêmement importante qui est devenu au fil du temps une technologie grand public pour la production de modèles pour évaluer l'ajustement et la forme de l'outillage pour la fabrication à faible volume. Les différents procédés utilisés sont la Stéréo lithographie ou photopolymère, le Frittage laser ou mise en forme, le dépôt par fusion (FDM), l'Impression jet d'encre et 3D, la lamination ou la stratoconception.

Toutefois, les tendances clés de l'industrie RP sont des machines de a faible cout et des applications nécessitant plus de précision vue à partir des systèmes antérieurs. Plus précisément, il s'agit, notamment, de l'amélioration de la qualité, de la réduction des délais, de l'économie de coûts, de la réduction du personnel et de la possibilité d'ajouter de nouvelles fonctionnalités. Cependant, pour illustrer l'importance du travail, l'axe de notre intervention est dirigé vers l'interface de commande qui est la partie électronique de la machine « Imprimante 3D ». II serait intéressant de chercher à réaliser de futures études de plus grande envergure, en ce sens.

Mots clés : Prototypage rapide, Protocole USB, Moteur Pas à pas, PIC 18F4550.

INTRODUCTION GENERALE :

Une machine-outil à commande numérique (MOCN, ou simplement CN) est une machine-outil dotée d'une commande numérique. Lorsque la commande numérique est assurée par un ordinateur, on parle parfois de machine CNC pour *computer numerical command*, francisé en « commande numérique par calculateur ».

Dans notre étude, nous avons essayé de rassembler les différents éléments qui constituent une machine à commande numérique.

Nous avons la partie électronique qui constitue l'interface de commande entre l'utilisateur et la machine, elle permet à l'utilisateur de transmettre des commandes simples a la machine et ceci via une interface (dans notre cas, nous avons utilisé l'interface USB).

La partie électronique, sur laquelle se base notre étude, transforme les commandes envoyées par l'utilisateur en impulsion électrique dans le but de faire tourner des moteurs (dans notre cas, des moteurs pas à pas) qui produiront à leurs tour des mouvements mécanique de translation dans différentes directions.

Notre étude va se baser essentiellement sur la partie électronique. Il y a tout d'abord l'interface Homme-Machine, celle-ci permet à l'utilisateur d'effectuer des commandes via un logiciel que nous avons mis au point à l'aide d'un langage de programmation évolué (Visual C++). Ce logiciel transmet des données à la carte de commande via l'interface USB. La carte électronique est pilotée par un microcontrôleur (Microchip PIC18F4550) qui assure la conversion des commandes envoyées par l'utilisateur en signaux électriques permettant de faire tourner les moteurs pas à pas des différents axes de la machine.

Pour la partie mécanique, nous avons essayé de la mettre au point du mieux qu'on a pu, et ceci afin de montrer le fonctionnement d'une machine à commande numérique même si elle ne rentre pas dans le domaine de notre spécialité et qu'elle relève du domaine des applications mécaniques.

Les machines à commandes numérique (CNC) sont utilisées aujourd'hui dans différents domaines d'application. Nous avons les machines-outils (fraiseuse, tours numériques etc..), les imprimantes 3D utilisés pour faire du prototypage rapide ainsi que les machines de perçage automatique.

Assignées à leur origine à l'usinage dit « 3 axes », de par les performances et possibilités des matériels et logiciels associés sans cesse croissant, les commandes numériques sont de plus en plus utilisées dans les procédés où il est requis de déplacer un ou des mobiles dont

les mouvements sont interpolés (vitesses des axes concourant au déplacement d'un point Liées entre elles) avec des contraintes de vitesse très faibles ou très élevées (de 0,001 m/min à 130 m/min) et/ou de très grande précision de trajet ou de positionnement (< 0,001 mm). (A noter que ces performances dépendent des capacités de la machine à les atteindre). De plus, certains procédés actuels requièrent de mouvoir non pas trois, quatre, cinq axes mais dix, vingt axes et même plus. Il n'est pas donc pas rare de voir une commande numérique piloter simultanément trente axes. Enfin, la puissance des microprocesseurs actuels ou circuits dédiés (FGPA, ASICS) permet d'exécuter avec célérité les algorithmes complexes de cinématique que requièrent par exemple les machines dites "hexapode".

CHAPITRE I :

Les Caractéristiques et
Les Modes de Commande
Des Moteurs Pas à Pas

INTRODUCTION :

Les moteurs pas à pas sont des transducteurs électromécaniques qui assurent la conversion des signaux électriques digitaux ou impulsionnels en mouvement de rotation ou de translation de type incrémental.

Les moteurs pas à pas, utilisés pratiquement dans tous les composants d'un système informatique, sont très mal connus, surtout en ce qui concerne leur mode de commande. Ils sont utilisés aussi bien dans les lecteurs de disquettes, les disques durs, que dans les scanners et les imprimantes 3D conçues pour le prototypage rapide[1] (**figure 1.1)**.

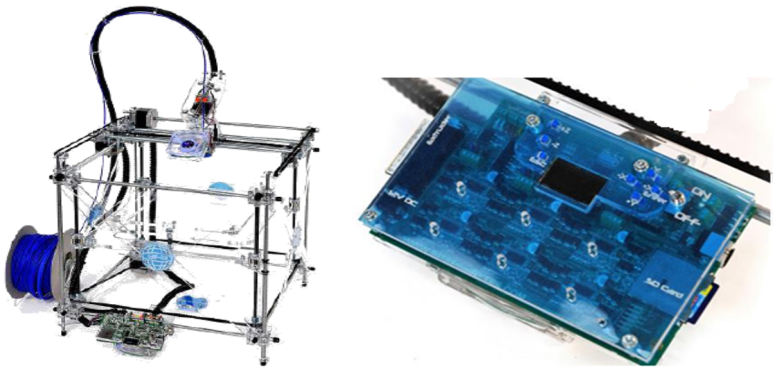


Figure 1.1. *Imprimante 3D-EXTRU3D (à gauche) et sa Carte d'interface (à droite)*

Dans le domaine de la robotique, ils sont la clef de voûte du système. Ils permettent d'obtenir une précision extraordinaire grâce à leur conception électrique et mécanique.

Le moteur pas à pas a été conçu à partir de deux démarches logiques très différentes :

- d'une part, on a cherché un moteur capable de développer un couple important à faible vitesse, voire même à l'arrêt ;
- d'autre part, on a étudié un dispositif capable de convertir des informations de caractère discret.

Le convertisseur d'énergie à basse vitesse et le transmetteur de l'information sont devenu un moteur pas à pas moderne vers les années 1970, grâce au développement conjugué de l'électronique de puissance et, surtout, grâce à l'apparition de l'électronique numérique à forte intégration.

[1] Obtention de prototype de forme complexe en un temps très court.

Le moteur pas à pas est actuellement le principal élément intermédiaire entre les dispositifs de traitement d'information et le monde électromécanique extérieur. Par ailleurs, ses capacités à contrôler la position et la vitesse, par un train d'impulsions de commande, assurent à ce convertisseur des applications comme :

- la traction des robots mobiles,
- le fonctionnement en moteur couple de grande puissance,
- l'indexage rotatif ou linéaire.

Dans sa version classique, le moteur pas à pas est alimenté à partir d'une source (de courant ou de tension) continue et le contrôle de la vitesse ou/et de la position s'effectue en boucle ouverte.

Le pilotage en boucle ouverte, qui constitue un des principaux avantages du moteur pas à pas, aussi bien du point de vue économique (coût et fiabilité d'installation) que fonctionnel (commande naturelle par "tout ou rien" à partir d'une horloge séparée ou intégrée dans un microprocesseur), présente un certain nombre d'inconvénients comme :

- la limitation du couple de démarrage,
- des instabilités de fonctionnement à certaines fréquences,
- des accélérations relativement modestes,

Maintenant, ce type de moteurs est utilisé de plus en plus dans les machines à commande numérique (CNC), leurs tailles ainsi que leur puissance ont augmenté.

Dans ce chapitre nous aborderons les différents types de moteurs pas à pas, ainsi que leur mode de commande.

1.1. GENERALITES :

Ils existent trois types de moteurs pas à pas : les moteurs à aimant permanent, les moteurs à réluctance variable ainsi que les moteurs hybrides. Nous verrons plus tard que les moteurs à aimants permanents se subdivisent en deux catégories.

Malgré les différences existant entre les moteurs, le résultat recherché est l'avance d'un seul pas, c'est-à-dire la rotation de leur axe suivant un angle déterminé à chaque impulsion que l'une ou l'autre de leurs bobines recevra. Cet angle qui varie selon la constitution interne du moteur, est en général compris entre 0.9° et 90°.

Les moteurs les plus couramment rencontrés présentent des pas de :

- 0.9° soit 400 pas par tour.
- 1.8° soit 200 pas par tour.
- 3.6° soit 100 pas par tour.
- 7.5° soit 46 pas par tour.
- 15° soit 24 pas par tour.

Il est évident que les moteurs pas à pas, de par leurs technologies, présentent une très grande précision et une durée de vie quasi illimitée, l'usure mécanique étant pratiquement inexistante (absence de frottement). **La figure 1.2** en représente l'aspect externe. Leur domaine de prédilection sera donc ceux ou la précision est de rigueur : les constituants mécaniques de l'informatique et la robotique.

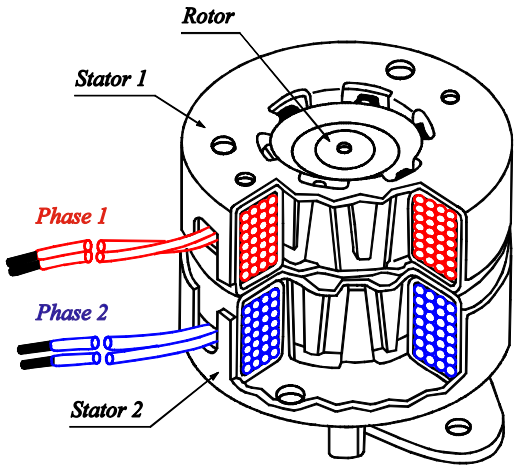


Figure 1.2. *Vue d'ensemble d'un moteur pas à pas*

Les moteurs pas à pas existent entre différentes tailles qui varient entre 1 cm et plus d'une dizaine de centimètres. Tout dépendra des applications dans lesquelles ils seront utilisés. Le plus petit moteur, par exemple, sera destiné au déplacement des têtes de lectures dans les disques durs ou un couple très faible est requis. Par contre le déplacement du bras d'un robot ou un chariot (machine CNC) demandera un couple nettement plus important, donc un moteur de diamètre élevé.

Signalons que le couple est exprimé en kilogramme par centimètre ($kg.cm^{-1}$), ce qui définit le poids en kilogrammes que pourra soulever l'axe d'un moteur pourvu d'un bras de longueur exprimée en centimètre.

La valeur de leur tension d'alimentation varie dans de grandes proportions, elle peut être comprise entre 3V et plusieurs dizaine de volts. De même, selon la résistance ohmique de leurs bobinages, le courant consommé s'étendra dans une gamme allant de quelques dizaines de milliampères a plusieurs ampères.

En simplifiant, nous pourrons dire que plus le courant sera élevé, plus le couple sera important.

1.2. TECHNOLOGIE DES MOTEURS PAS A PAS :

Un actionneur électrique peut créer deux types de mouvements : un mouvement de rotation ou un mouvement de translation.

1.2.1. Moteurs pas à pas rotatifs :

Les moteurs pas à pas peuvent être classés en fonction du phénomène physique qui est à l'origine de leur mouvement. On distingue principalement, deux catégories de moteurs pas à pas : les moteurs à reluctance variable et les moteurs polarisés.

1.2.1.1. Moteur pas à pas rotatif à reluctance variable :

La rotation d'un moteur à reluctance est engendrée par la réaction, entre un champ magnétique statorique et un rotor saillant, qui conduit à une disposition alignée de la partie saillante avec le pole créé par le champ magnétique. Dans ces conditions le flux est maximum et la réluctance du circuit magnétique est minimum.

Ce type de moteur est caractérisé par une structure dentée au niveau du rotor et du stator. Le rotor est fabriqué en acier doux non magnétique. Le nombre de bobines dans le stator et le type de connexion déterminent le nombre de phases du moteur.

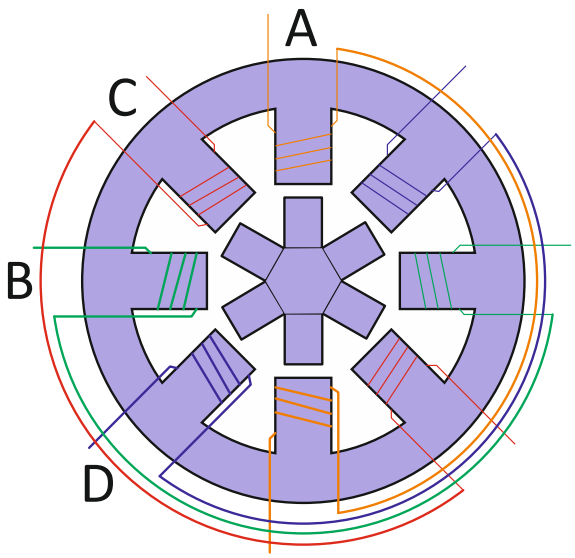


Figure 1.3. *Structure classique d'un moteur à reluctance variable rotatif à double saillance*

La **figure 1.3** présente un moteur pas à pas réluctant comportant huit plots au stator et six dents au rotor ce qui donne un moteur a quatre phases.

Chaque couple de deux bobines statoriques diamétralement opposées et connectées en série, constitue une des quatre phases de la machine considérée. L'alimentation d'une phase crée un couple permettant le déplacement du rotor vers une position d'équilibre qu'il garde tant que l'alimentation est maintenue. Cet état d'équilibre correspond à une position alignée entre les dents statoriques porteuses de la phase alimentée et les dents rotoriques. Chaque nouvelle séquence d'alimentation établit un nouvel équilibre. Le moteur se déplace donc avec un pas angulaire de 15°.

Le mode de commande peut dans ce cas, de la même façon que pour les autres moteurs, être monophasé, biphasé ou demi-pas. Les séquences de commandes sont présentées dans le **tableau 1.1.**

Mode monophasé	Mode biphasé	Mode demi-pas
A	AC	A
B	CB	AC
C	BD	C
D	DA	BC
		B
		BD
		D
		DA

Tableau 1.1. *Ordre des différentes commandes pour moteur à reluctance variable*

Pour augmenter la résolution angulaire de ces moteurs, des constructions a circuits magnétiques multiples, dites multistack, peuvent être envisagées. Ces moteurs se présentent principalement sous deux formes :

- Un empilage de plusieurs machines monophasées identiques assemblées mécaniquement sur le même arbre.
- Un stator unique, possédant plusieurs rotors magnétiquement indépendants, mais couplés mécaniquement.

Dans toutes ces machines, le rotor et les stators présentent le même nombre de dents, le mouvement étant obtenu par le décalage mécanique des rotors ou des stators, les uns par rapport aux autres.

1.2.1.2. Moteur pas à pas rotatif à aimant permanent :

Les moteurs à aimant permanent sont constitués d'un stator supportant les bobinages et d'un rotor magnétique (aimant bipolaire). Cette catégorie de moteurs se subdivise en deux types : le moteur unipolaire et le moteur bipolaire.

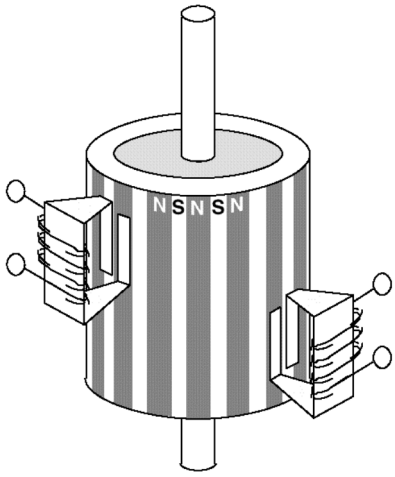
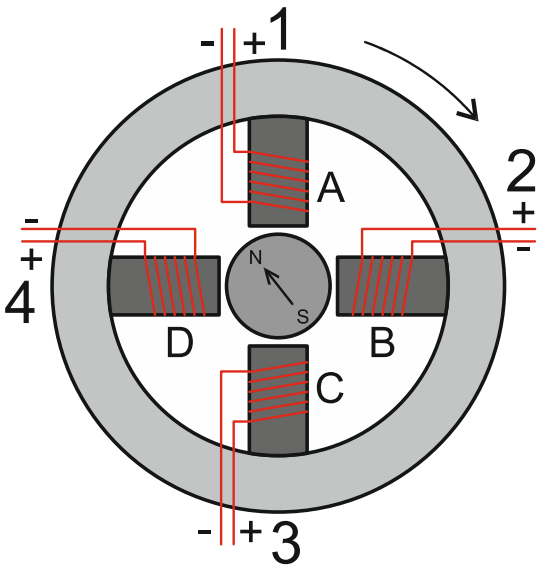


Figure 1.4. *Structure interne d'un moteur à aimant permanent*

La **figure 1.5** représente le schéma simplifié d'un moteur à aimant. Le premier dessin de la figure le représente dans sa position de repos, lorsque les bobinages ne sont traversés par aucun courant. Chacun des pôles du rotor se place devant une paire des plots du stator.



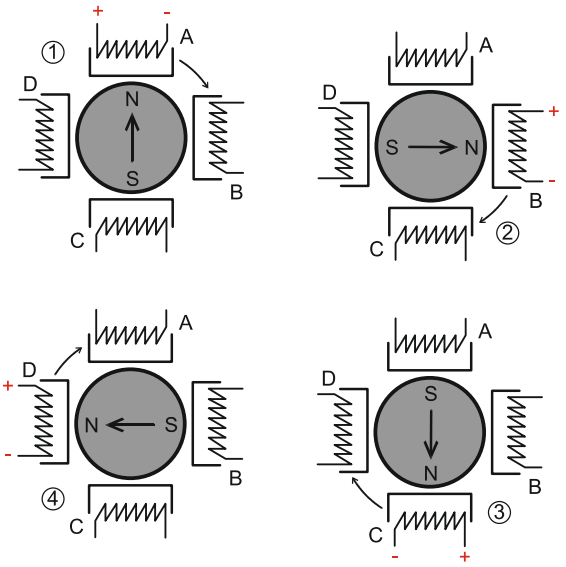


Figure 1.5. *Fonctionnement schématique d'un moteur pas à pas*

Les quatre dessins du bas de la figure illustrent ce qui se passe lorsque les bobinages sont alimentés à tour de rôle : d'abord A, puis B, puis C et enfin D. Le pôle nord du rotor sera attiré par le pôle sud du stator, pole crée par la circulation d'un courant dans le bobinage.

Cet exemple permet de comprendre la progression pas par pas du moteur. Ici, il effectuera quatre pas par tours.

1.2.1.2.1. Les moteurs unipolaires :

Une représentation schématisée d'un moteur d'un moteur unipolaire est donnée en **figure 1.6**. Afin d'inverser le sens du courant, les enroulements sont réalisés au moyen de deux fils dont l'une des extrémités est reliée au + ou au – de l'alimentation. La commande de ce type moteur est la plus simple de tous les moteurs pas à pas puisqu'il suffira d'alimenter les bobinages à tour de rôle pour faire tourner l'axe d'un pas. Le schéma de la **figure 1.7** résume la séquence la plus simple.

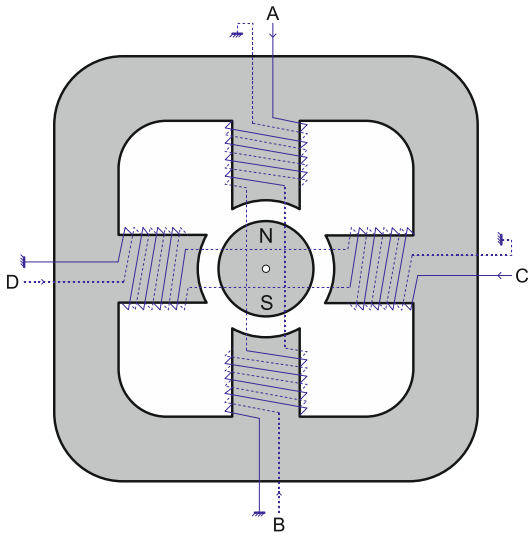


Figure 1.6. *Représentation schématique d'un moteur unipolaire*

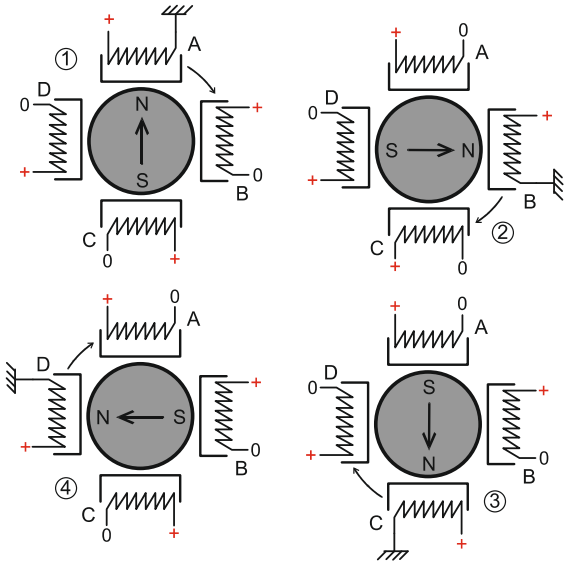


Figure 1.7. *Séquence de rotation simple*

Comme pour chaque type de moteur, le modèle unipolaire peut être commandé en mode monophasé, biphasé ou demi-pas. Le **tableau 1.2** donne l'ordre des différentes commandes.

Mode monophasé	Mode biphasé	Mode demi-pas
0001	0101	0001
0100	0110	0101
0010	1010	0100
1000	1001	0110
		0010
		1010
		1000
		1001

Tableau 1.2. *Ordre des différentes commandes pour moteur unipolaire*

Signalons que le moteur unipolaire présentera, à volume égal, un couple moins important que le moteur bipolaire.

1.2.1.2.2. Les moteurs bipolaires :

La **figure 1.8** représente la constitution interne d'un moteur de type bipolaire. Ce type de moteur nécessite une commande plus complexe que celle du moteur unipolaire, le courant devant changer de sens dans les enroulements à chaque pas effectué.

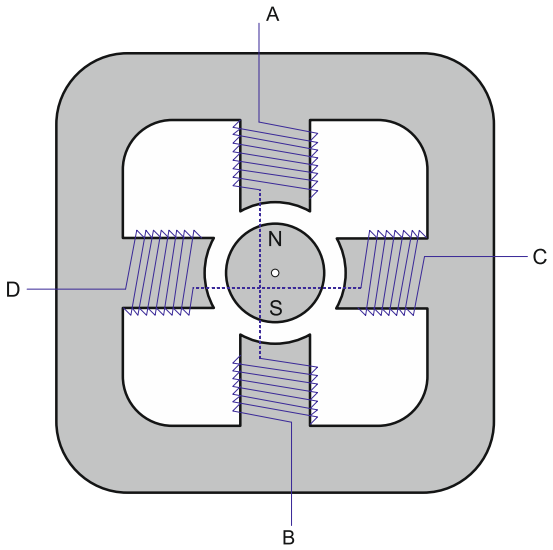


Figure 1.8. *Représentation schématique d'un moteur bipolaire*

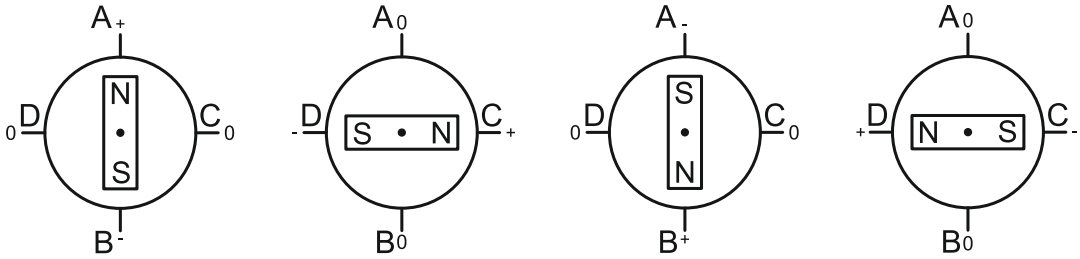
Comme pour le modèle précèdent, ce moteur peut être alimenté sous trois séquences différentes (**tableau 1.3**), représentées par ailleurs sur la **figure 1.9**.

Mode monophasé	Mode biphasé	Mode demi-pas
AB	AB-CD	AB
CD	BA-CD	AB-CD
BA	BA-DC	CD
DC	AB-DC	BA-CD
AB	AB-CD	BA
etc..	etc..	BA-DC
		DC
		AB-DC
		AB
		etc..

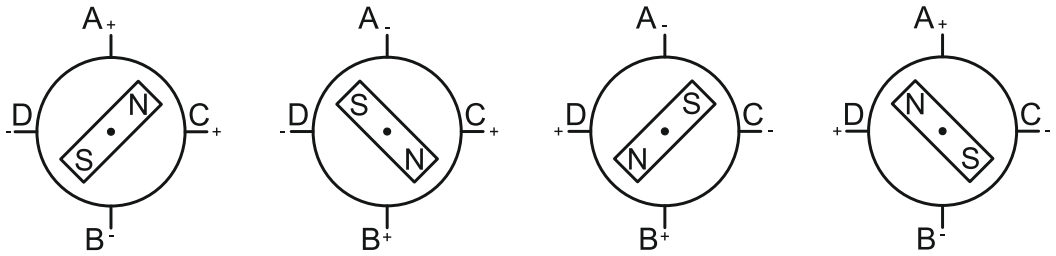
Tableau 1.3. *Ordre des différentes commandes pour moteur bipolaire*

Dans le mode monophasé, le couple n'est pas très important puisqu'un seul enroulement est alimenté pour effectuer un pas.

Mode monophasé



Mode biphasé



Mode demi-pas

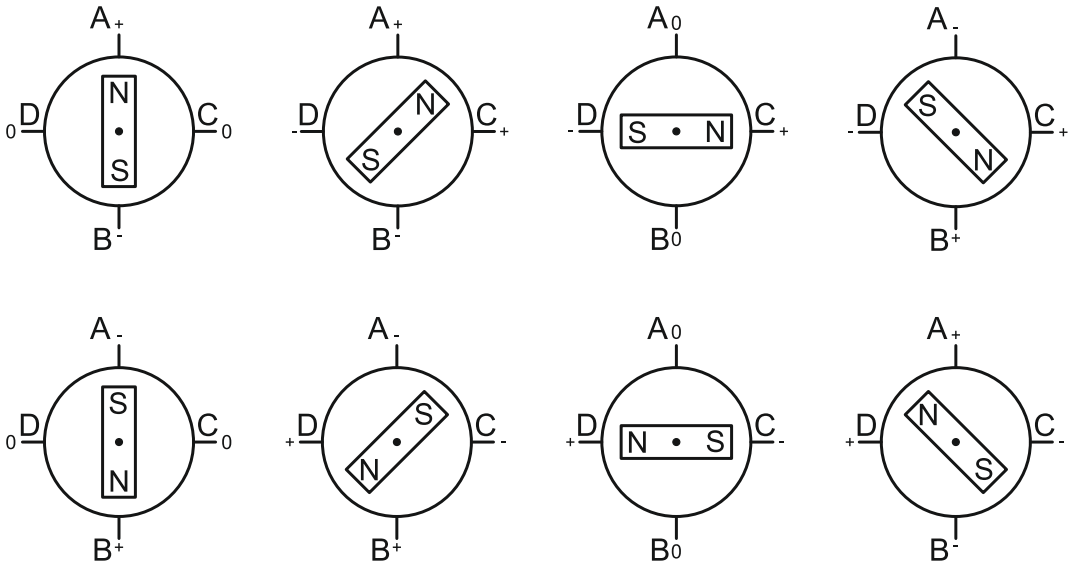


Figure 1.9. *Séquences de commande d'un moteur bipolaire*

C'est dans le mode biphasé que le moteur développera la plus grande puissance (couple élevé) car les deux phases seront alimentées en même temps.

Le mode demi-pas permet de doubler le nombre de pas qu'un moteur peut effectuer par tour, mais cette précision supplémentaire engendre un couple irrégulier. En effet, dans ce mode, la commande du moteur est un mélange de mode biphasé et monophasé. Dans ce cas, si la charge est importante, des pas risquent de « sauter », ce qui n'est pas le but recherché.

1.2.1.3. Moteur pas à pas rotatif hybrides :

Les moteurs pas à pas hybrides sont généralement constitués d'un rotor denté muni d'aimants permanents. Le rotor possède généralement deux disques polaires décalés d'un angle électrique π.

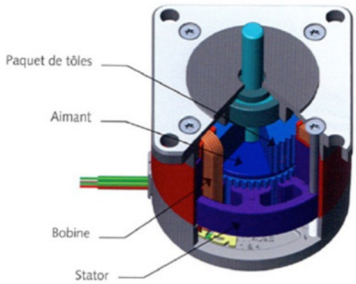


Figure 1.10. *Moteur pas à pas hybride*

Ce type de moteur présente à la fois les avantages du moteur a aimant permanent qui possède un couple élevé et ceux du moteur a réluctance variable qui permet d'obtenir un nombre importants de pas par tour. Toutefois l'inertie d'un tel rotor ainsi que les pertes fer sont relativement importantes et pénalisent donc cette structure.

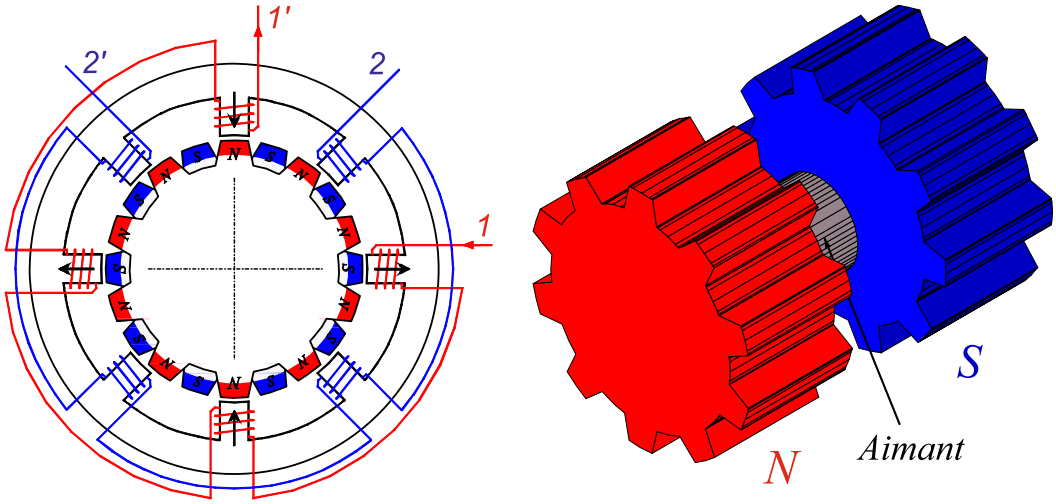


Figure 1.11. *Structure interne d'un moteur hybride*

Les moteurs pas à pas hybrides réunissent, au moins en partie, les avantages des moteurs pas à pas à reluctance variable et a aimants permanents, à savoir :

- Un grand nombre de pas par tour.
- Une fréquence propre mécanique importante.
- Un couple massique élevé.
- Un amortissement interne important.
- Une mémoire de position.

Dans sa configuration de base le moteur pas à pas hybride comporte un stator en fer feuilleté à plots saillants et deux couronnes rotoriques dentées en matériau ferromagnétique, géométriquement identiques et réunies par un aimant permanent cylindrique magnétisé axialement. Les lignes de champs de l'aimant se ferment à travers les dents du rotor. Vu du stator, le rotor présente autant de pôles magnétiques actifs qu'il possède de dents. Les dents sur une des couronnes sont décalées par rapport aux dents de l'autre d'un demi-pas dentaire $1/2\tau_{dr}$.

Le nombre de pôles vu au stator est lié au nombre de dents d'une couronne rotorique par la relation

$$p = N_{dr} \qquad 1.1$$

Le nombre de pas par tour prend donc la forme

$$N_{pt} = 2m \,.\, N_{dr} = 2mp \qquad 1.2$$

Avec

p : Nombre de pôles magnétiques vu par le stator.
N_{dr} : Nombre de dents sur une couronne rotorique.
m : Nombre de phases (doit être pair).

L'augmentation du nombre de plots statoriques alimentés simultanément permet d'augmenter le nombre de dents du rotor, et donc de diminuer le pas angulaire du rotor. Le même résultat s'obtient par la subdivision des plots en plusieurs dents.

Les moteurs pas à pas hybrides comptent parmi les moteurs pas à pas les plus fabriqués. Ils existent aussi bien en structure à circuits simples (single stack), qu'en structure multiple (multi stack) Dans presque tous les cas, les plots dentés du stator présentent le même pas que les dents aimantées du rotor.

Le couple est constitué par la variation des perméances mutuelles bobinages –dents aimantées du rotor (couple électromagnétique) et par un couple réluctant crée principalement par la variation de la perméance propre vu par les aimants (couple de détente) Les dents aimantées de chaque couronne rotorique sont perçues par le stator comme autant d'aimants permanents, l'interaction de ces aimants avec les courants statoriques engendre un couple électromagnétique identique à celui du moteur pas à pas à aimants permanents.

1.2.2. Moteurs pas à pas linéaires :

Tout comme pour les versions rotatives, les moteurs pas à pas linéaires se composent d'un stator portant les bobinages et d'une partie mobile se déplaçant cette fois-ci linéairement.

Cette partie mobile peut être réalisée à partir d'une pièce ferromagnétique dentée, dans ce cas la structure est dite a reluctance ou passive ; ou alors la partie mobile est munie d'aimants permanents et dans ce cas la structure est dite polarisée ou active.

1.2.2.1. Moteur pas à pas linéaire à reluctance variable :

Ce type de moteurs est caractérisé par une structure dentée au niveau du stator et de la partie mobile. Le circuit magnétique est généralement assemblé a partie de tôles magnétiques de forte perméabilité. Toutefois, les machines destinées au positionnement ou à une translation lente, peuvent être réalisées avec des pièces massives. Les enroulements du circuit électrique sont généralement concentrés autour des plots du stator et donc faciles à réaliser.

1.2.2.2. Moteur pas à pas linéaire polarises :

1.2.2.2.1. Les moteurs à aimant permanent :

Ce sont des moteurs dont la partie mobile est généralement lisse et formée d'une succession d'aimants permanents montés en surface.

1.2.2.2.2. Les moteurs hybrides:

Le mouvement des moteurs hybrides résulte de la superposition de la force développé par l'effet réluctant des dents et de la force créée par l'aimant. La contribution des amplitudes et des périodes géométrique de ces forces permet de réaliser des caractéristiques statiques très diversifiées. En effet, l'aimant, placé dans la structure hybride, assure une certaine distribution des lignes de champ. L'alimentation des bobines produit un phénomène d'aiguillage des lignes de champs plus au moins important suivant l'intensité du courant d'alimentation. En jouant sur l'orientation des lignes de champs, il est possible de maitriser la variation de la force résultante.

CONCLUSION :

Dans ce chapitre, nous avons donné un aperçu général sur les différents types de moteurs pas à pas, ainsi que leur mode de fonctionnement.

Voici leurs avantages :

- Rotation constante pour chaque commande (précision meilleure que 5% d'un pas).
- Existence de couple a l'arrêt.
- Contrôle de la position, de la vitesse et synchronisation de plusieurs moteurs (pas besoin de contre-réaction).
- Moteur sans balai.

Et les inconvénients :

- Plus difficile à faire fonctionner qu'un moteur à courant continu.
- Vitesse et couple relativement faible.
- Couple décroissant rapidement lorsque la vitesse augmente.
- Résonance mécanique.

En ce qui concerne notre projet, nous avons utilisés des moteurs pas à pas de type bipolaires a aimant permanent, leurs résolution est de 1,8° par pas (200 pas), ainsi qu'un couple de maintien de 4kg/cm avec une alimentation électrique de 12v.

CHAPITRE II :

Approche Théorique et Pratique Sur L'interface USB

INTRODUCTION :

Le ***Universal Serial Bus*** (**USB**, en français *Bus universel en série*, dont le sigle, inusité, est *BUS*) est une norme relative à un bus informatique en transmission série qui sert à connecter des périphériques informatiques à un ordinateur. Le bus USB permet de connecter des périphériques *à chaud* (quand l'ordinateur est en marche) et en bénéficiant du *Plug and Play* (le système reconnaît automatiquement le périphérique). Il peut alimenter les périphériques peu gourmands en énergie (disques SSD en particulier).

Apparu en 1996, ce connecteur s'est généralisé dans les années 2000 pour connecter souris, clavier d'ordinateur, imprimantes, clés USB et autres périphériques bon marché sur les ordinateurs personnels.

L'USB s'est depuis déclinée en version 2 (0,48 Gb/s) et version 3 (5 Gb/s) puis en août 2013 est annoncé le standard 3.1 à 10 Gb/s.

L'USB a été conçu au milieu des années 1990 afin de remplacer les nombreux ports externes d'ordinateurs lents et incompatibles. Différentes versions de la norme ont été développées au fur et à mesure des avancées technologiques.

- En 1996, la première version de la norme, l'**USB 1.0**, est spécifiée par sept partenaires industriels (Compaq, DEC, IBM, Intel, Microsoft, NEC et Northern Telecom).
- En 1998, la version **USB 1.1** apporte des corrections et deux vitesses de communications : 1,5 Mbit/s (*faible vitesse*, ou Low Speed), et 12 Mbit/s (soit 1,5 Mo/s) (*pleine vitesse* ou Full Speed). En août 1998, Apple est le premier constructeur à uniquement proposer les ports USB (en remplacement des ports d'ancienne génération), avec la sortie de l'iMac G3, ce qui a fait décoller le marché des périphériques USB.
- En 2000, la version **USB 2.0** ajoute des communications à 480 Mbit/s (*haute vitesse* ou High Speed) (soit 60 Mo/s).
- En 2005, le **Wireless USB**, une version sans-fil de l'USB, est spécifiée par le *Wireless USB Promoter Group*. Elle promet 480 Mbit/s à une distance de 3 m et 110 Mbit/s à 10 m.
- En 2008, l'**USB 3.0** transmet à 5 Gbit/s (soit 600 Mo/s) (*vitesse supérieure* ou SuperSpeed) et 4,5 Watts. Les nouveaux périphériques disposent de connexions à 6 contacts au lieu de 4, mais la compatibilité ascendante des prises et câbles avec les versions précédentes est assurée. La compatibilité descendante est impossible,

les câbles USB 3.0 de type B n'étant pas compatibles avec les prises USB 1.1/2.0, mais il existe des adaptateurs.

- Début 2010 introduction de l'USB 3 dans des produits grand public. Les prises femelles correspondantes sont signalées par une couleur bleue. Apparition aussi des prises femelles USB *rouges*, signalant une puissance électrique disponible supérieure, et appropriée au chargement rapide de petits appareils y compris (à condition de le paramétrer dans le BIOS ou l'EFI) lorsque l'ordinateur est éteint.

- Août 2013, l'**USB 3.1** est officialisé et promet des débits doubles de ceux de l'USB 3.0, soit 10 Gbits/s (1,2 Go/s). Le nouveau standard (câbles, interface) sera rétro compatible avec l'USB 3.0 et l'USB 2.0. Toutefois la connectique change, elle sera plus fine et n'imposera pas de sens de branchement. Pour tout de même permettre la connexion vers des connecteurs USB 2.0 et 3.0, le standard devra prendre en compte la possibilité d'avoir des adaptateurs passifs (à l'inverse des adaptateurs Lightning, le connecteur réversible qu'Apple a lancé avec l'iPhone 5 en 2012), pour garder une taille réduite et un coût de fabrication mesuré. Cette nouvelle connectique se nommera *Type C*.

2.1. CARACTERISTIQUES GENERALES :

L'*Universal Serial Bus* est une connexion à haute vitesse qui permet de connecter des périphériques externes à un ordinateur (hôte dans la terminologie USB). Il permet le branchement simultané de 127 périphériques par contrôleur (hôte). Le bus autorise les branchements et débranchements à chaud (« *Hot-Plug* », sans avoir besoin de redémarrer l'ordinateur) et fournit l'alimentation électrique des périphériques sous 5 V, dans la limite de 0,5 A, soit 2,5 W.

D'un point de vue logiciel, le bus possède une topologie arborescente (dite également en étoile) : les feuilles de cet arbre sont les périphériques ; les nœuds internes sont des *hubs* qui permettent de greffer des sous-arborescences dans l'arborescence principale. On trouve dans le commerce ces *hubs* sous forme de petits boîtiers alimentés soit sur le bus, soit sur le secteur, et qui s'utilisent comme des multiprises. Certains périphériques intègrent également un *hub* (moniteurs, claviers…). Cependant, tout bus USB possède au moins un *hub* situé sur le contrôleur : le *hub racine*, qui peut gérer les prises USB de l'ordinateur. Le nombre de *hubs* connectés en cascade est limité : *hub racine* compris, il ne doit pas exister plus de sept couches dans l'arborescence (**figure 2.12**).

À plus bas niveau, il s'agit d'un anneau à jeton (ou *Token Ring*) : chaque nœud dispose successivement du bus. Il n'y a pas de collision de paquets comme en Ethernet, mais le nombre maximal de nœuds est prédéfini. Pour cette raison, l'USB n'est pas adapté aux communications réseau : l'apparition des "modems" ADSL USB était un moyen de diffuser l'ADSL à une époque où la plupart des PC bas de gamme disposaient du port USB mais pas d'Ethernet.

La version 1.x du bus peut communiquer dans deux modes : lent (1,5 Mbit/s) ou rapide (12 Mbit/s, soit 1,5 Mo/s) :

- le *mode lent* (« *Low Speed* ») permet de connecter des périphériques qui ont besoin de transférer peu de données, comme les claviers et souris ;
- le *mode rapide* (« *Full Speed* ») est utilisé pour connecter des imprimantes, scanners, disques durs, graveurs de CD et autres périphériques ayant besoin de plus de rapidité. Néanmoins il est insuffisant pour beaucoup de périphériques de stockage de masse (ce mode permet la vitesse « 10 X » des CD).

USB 2.0 introduit un troisième mode permettant de communiquer à 480 Mbit/s. Ce mode est appelé « *High Speed* ». Il est utilisé par les périphériques rapides : disques durs, graveurs… Mais en 2009, la plupart des périphériques ont une vitesse inférieure à ce que permet l'USB 2.0.

La dernière version, l'USB 3.0, comporte un quatrième mode (« *Super Speed* ») permettant de communiquer à 5 Gbit/s. Ce nouveau mode utilisant un codage des données de type 8b/10b, la vitesse de transfert effective des données est de seulement 4 Gbit/s (500 Mo/s).

Lorsque l'on parle d'un équipement USB, il est nécessaire de préciser la version de la norme (1.1, 2.0 ou 3.0) mais également la vitesse (Low, Full ou High Speed). Une clef USB spécifiée en USB 2.0 n'est pas forcément High Speed si cela n'est pas précisé par un logo « High Speed ».

Le bus USB reste plus lent que des interfaces internes comme PCI ou AGP ou SATA /e-Sata (dans sa version 1.x et 2.0).

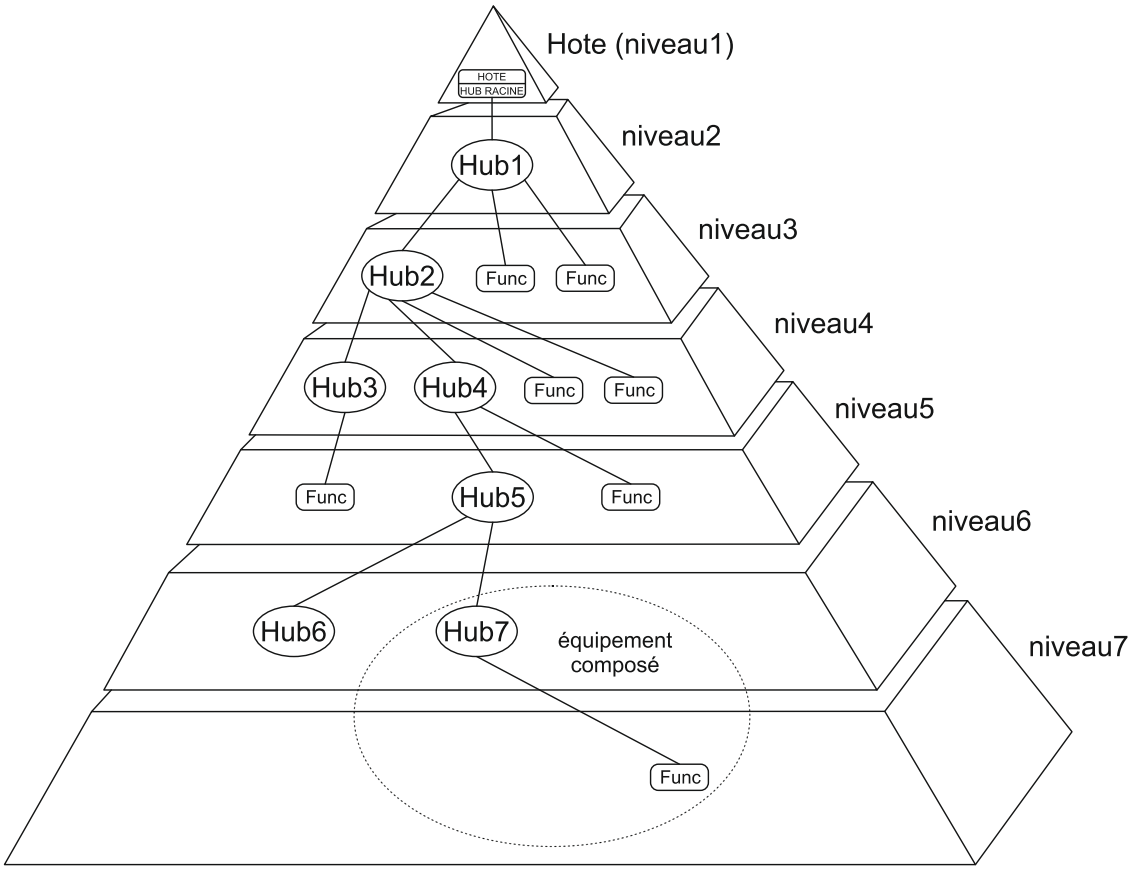


Figure 2.12. *Topologie du bus USB*

L'USB, comme son nom l'indique est un Bus Série. Il utilise quatre fils isolés dont deux sont l'alimentation (+5V et GND). Les deux restants forment une paire torsadée qui véhiculent les signaux de données différentiels. Il utilise un schéma d'encodage NRZI (Pas de retour à Zéro inversé – **figure 2.13**) pour envoyer des données avec un champ *sync* de manière à synchroniser les horloges de l'Hôte et du récepteur.

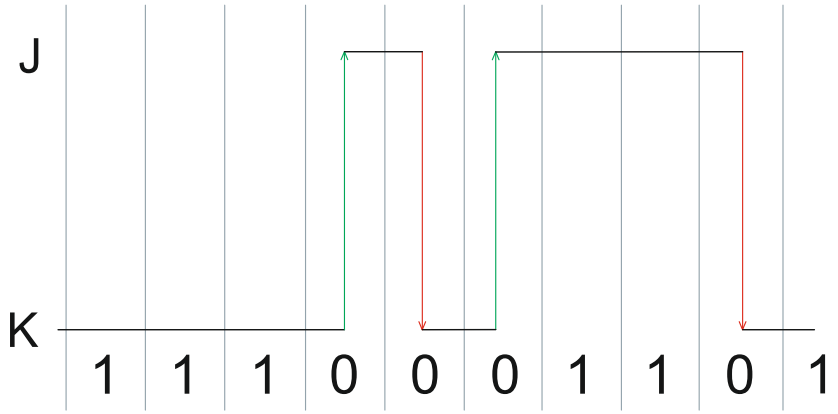


Figure 2.13. *Codage NRZI*

L'USB supporte le système " plug'n play " branchement à chaud avec des drivers qui sont directement chargeable et déchargeable. L'utilisateur branche simplement l'appareil sur le Bus. L'Hôte détectera cet ajout, interrogera l'appareil nouvellement inséré et chargera le driver approprié pendant le temps qu'il faut au sablier pour clignoter sur votre écran assurant qu'un driver est installé pour votre appareil. L'utilisateur final n'a pas besoin de se soucier des terminaisons, des termes tels que IRQs et adresses de ports, ou de la réinitialisation de l'ordinateur. Une fois que l'utilisateur a terminé, on peut simplement retirer le câble, l'Hôte détectera cette absence et déchargera automatiquement le driver.

Le chargement du driver approprié sera réalisé en utilisant une combinaison PID / VID (Interface Produit Machine / Vendeur Machine). On peut se procurer le VID au forum des fournisseurs USB en payant, ce qui est considéré comme un autre point de blocage par USB. On peut trouver le catalogue des tarifs réactualisés sur le site web des fournisseurs USB.

Une autre caractéristique intéressante de l'USB réside dans ces modes de transferts. L'USB soutient des transferts de contrôles, d'interruptions, en Bloc et Isochrone. Lorsque nous examinerons les autres modes de transferts ultérieurement, nous nous rendrons compte que l'Isochrone permet à un appareil de réserver une approximation définie de la bande passante avec un temps d'attente garanti. Ce système se révèle idéal dans les applications Audio et Vidéo ou l'encombrement peut susciter une perte de données ou une chute de trames. Chaque mode de transfert fourni au concepteur des compromis dans les domaines de la détection d'erreur et de la reprise, du temps d'attente garanti et de la bande passante.

2.2. CARACTERISTIQUES MECANIQUES :

Tous les appareils ont une connexion amont vers l'hôte et tous les hôtes ont une connexion aval vers l'appareil. Les connecteurs amont et aval ne sont pas interchangeables

mécaniquement, éliminant ainsi les connexions de rebouclage interdite aux Hubs comme pour un port aval connecté à un port aval. Il y a généralement 2 types de connecteurs, appelé type A et type B présenté ci-dessous.

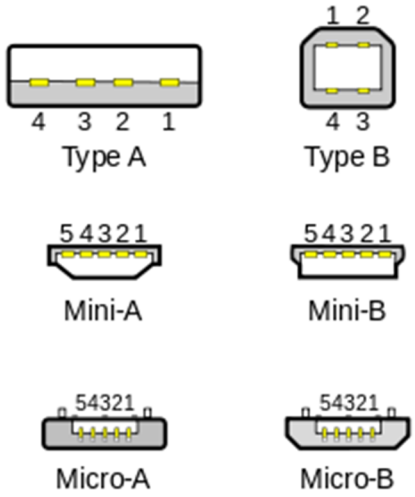


Figure 2.14. *Différents types de connecteurs USB*

Les prises mâles de type A sont toujours tournés vers l'Amont. Les prises femelles de type A se trouveront généralement sur les hôtes et les Hubs. Par exemple, les prises femelles de type A sont courantes sur les cartes mères des ordinateurs et les Hubs. Les prises mâles de type B sont toujours connecté vers l'aval et par conséquent les prises femelles de type B se trouvent sur les appareils.

Il vient de paraître une spécificité On-the-Go qui ajoute la fonctionnalité paire à paire (peer to peer) à l'USB. D'où l'introduction des hôtes USB dans les téléphones mobiles et les agendas électroniques, de même qu'une particularité pour les prises mâles mini A, les prises femelles mini A et les prises femelles mini A-B. Tout porte à croire que nous devrions être inondés de câbles mini USB et d'une gamme de câbles convertisseurs de la taille mini à standard.

Fonction	Couleur	Numéro de broche pour les types A et B	Numéro de broche pour le type mini B
Alimentation +5 V (VBUS)	Rouge	1	1
Données (D-)	Blanc	2	2
Données (D+)	Vert	3	3
Masse (GND)	Noir	4	5

Tableau 2.4. *Brochage des différents connecteurs USB*

On utilise des couleurs standards pour les fils intérieurs des câbles USB de façon à faciliter l'identification des fils d'un constructeur à un autre (**tableau 2.4**). La normalisation précise les différents paramètres électriques pour les câbles.

2.3. CARACTERISTIQUES ELECTRIQUES :

Comme nous en avons déjà discuté, l'USB utilise une paire de transmission différentielle pour les données. Celle-ci étant codé en utilisant le NRZI et est garni de bits pour assurer les transitions adéquates dans le flot de données. Sur les appareils à vitesse basse et pleine un '1' différentiel est transmis en mettant D+ au-dessus de 2,8V grâce à une résistance de 15k ohms relié à la masse et D- en dessous de 0,3V avec une résistance de 1,5k ohms relié à 3,6V. D'autre part un différentiel '0' correspond à D- plus grand que 2,8V et D+ inférieur à 0,3V avec les mêmes résistances de rappel état haut/bas adéquates.

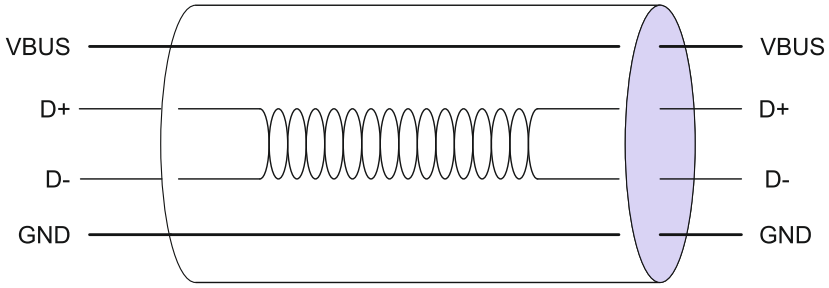


Figure 2.15. *Câble USB*

Le récepteur défini un différentiel '1' avec D+ plus grand de 200 mV que D- et un différentiel '0' avec D+ plus petit de 200 mV que D-. La polarité du signal est inversée en fonction de la vitesse du BUS. En conséquence les états référencés par les termes 'J' et 'K'

sont utilisés pour signifier les niveaux logiques. En vitesse basse, un état 'J' est un différentiel '0'. En vitesse haute, un état 'J' est un différentiel '1'.

Les émetteurs / récepteurs USB comprendront à la fois des sorties différentiels et uniques (non complémentaires). Certains états de BUS USB sont indiqués par des signaux à **sorties uniques** (*single ended zero*) sur D+, D- ou les deux. Par exemple un zéro à sorties uniques ou **SE0** peut être utilisé pour signifier la réinitialisation d'un appareil s'il est maintenu plus de 10 ms. On génère un **SE0** en maintenant D+ et D- en position basse (inférieur à 0,3V). Les sorties uniques et différentielles sont importantes d'être notées si vous utilisez un émetteurs / récepteurs et un FPGA comme appareil USB. Vous ne pouvez pas vous contenter simplement d'échantillonner la sortie différentielle.

Le BUS basse et pleine vitesse a une impédance caractéristique de 90 Ohms +/-15%. Il est donc important d'observer la documentation technique lorsque vous sélectionnez les résistances des caractéristiques électriques séries pour D+ et D- afin d'équilibrer l'impédance. Toute documentation technique devrait spécifier ces valeurs et tolérances.

2.4. IDENTIFICATION DE LA VITESSE :

Un appareil USB doit indiquer sa vitesse en mettant soit D+ ou D- à 3,3V. Un appareil pleine vitesse, représenté plus bas utilisera une résistance de rappel rattaché à D+ pour se signaler comme tel. Ces résistances de rappel à l'extrémité de l'appareil seraient aussi utilisées par l'hôte ou Hub pour détecter la présence d'un appareil connecté à son port. Sans résistance de rappel, l'USB suppose qu'il n'y a rien de connecté au BUS. Certains appareils possèdent cette résistance intégrée sur le silicium, pouvant être connecté ou non sous commande micro programmée, d'autres exigent une résistance externe.

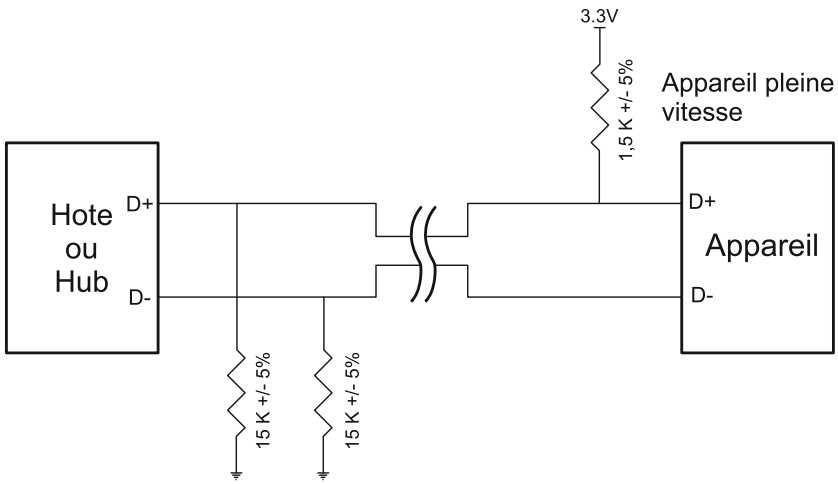


Figure 2.16. *Appareil pleine vitesse avec résistance de rappel état haut branché sur D+*

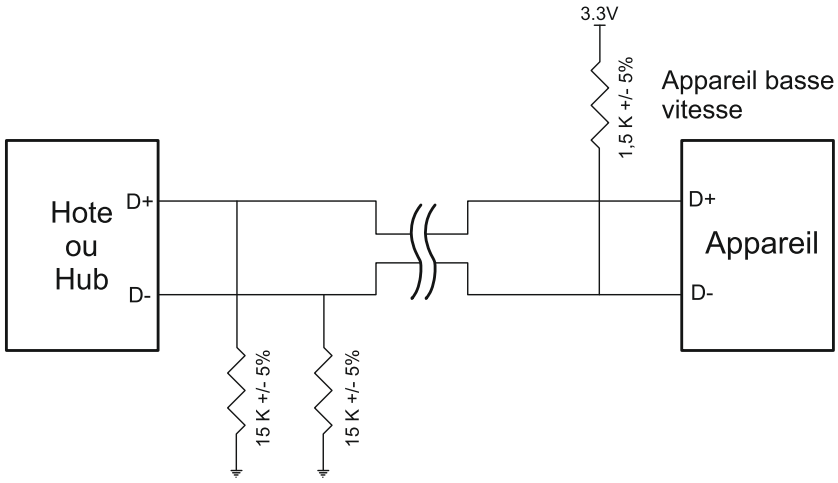


Figure 2.17. *Appareil pleine vitesse avec résistance de rappel état haut branché sur D-*

Vous noterez que nous n'avons pas inclus d'identification de vitesse pour le mode haute vitesse Les appareils haute vitesse démarreront dès qu'ils seront connectés en tant qu'appareils pleine vitesse (1,5 kOhms à 3,3V). Une fois qu'il sera attaché il émettra un **Chirp** à haute vitesse pendant la réinitialisation et établira une connexion à grande vitesse si le Hub le supporte. Si l'appareil fonctionne en mode haute vitesse, alors la résistance de rappel est retirée pour équilibrer la ligne.

Toutefois un appareil haute vitesse ne peut pas supporter le mode basse vitesse. Il devrait seulement supporter le mode pleine vitesse nécessaire en début de connexion, ensuite le mode haute vitesse s'il réussit par être négocié. Un appareil face aval à détection d'anomalies pour l'USB 2.0 (Hub ou Hôte) doit supporter les 3 modes, Haute, pleine et basse vitesse.

2.5. L'ALIMENTATION USB (V_{BUS}) :

Un des avantages de l'USB réside dans ces appareils alimentés par le Bus. Ceux-ci obtiennent leur alimentation à partir du Bus et ne demande aucune prise externe et câble additionnel. Cependant beaucoup de gens se focalisent sur cette option sans prendre en compte au préalable tous les critères nécessaires.

Un appareil USB précise sa consommation électrique exprimée en unité de 2mA dans le descripteur de la configuration que nous examinerons plus tard. Un appareil ne peut pas augmenter sa consommation électrique plus qu'il n'est précisé pendant l'énumération, même s'il perd de la puissance externe. Il existe 3 classes de fonctions USB :

- Les fonctions alimentées par le Bus à basse puissance (Low-Power)
- Les fonctions alimentées par le Bus à haute puissance (High-Power)
- Les fonctions auto alimentées (Self-powered)

Les fonctions alimentées par le Bus à basse puissance tirent toute leur puissance de V_{Bus} et ne peuvent en tirer que la charge d'une unité. La spécification USB défini une charge d'unité à 100mA. Les fonctions alimentées par le Bus à basse puissance peuvent aussi être conçues pour travailler à une tension de V_{BUS} tombant à 4,4V et montant à un maximum de 5,25V mesuré à la prise amont de l'appareil. Pour beaucoup d'appareil 3,3V, des régulateurs LDO sont obligatoires.

Les fonctions alimentées par le Bus à haute puissance tireront toute leur puissance du Bus et ne pourront tirer plus d'une unité de charge jusqu'à ce qu'elles aient été configurées, après quoi elles pourront tirer cinq unités de charge (500mA max) pourvu que cela soit demandé dans son descripteur. Les fonctions du Bus à Haute puissance doivent être capables d'être détectées et énumérées à un minimum de 4,40V. Lorsqu'elles fonctionnent à pleine charge, un V_{BUS} minimum de 4,75V est précisé avec un maximum de 5,25V. Une fois de plus, ces mesures sont prises à la prise mâle amont.

Les fonctions auto alimentées peuvent tirer jusqu'à une unité et faire dériver le reste de leur alimentation d'une source extérieure. Si cette source extérieure venait à manquer, il doit y avoir des réserves en place de manière à ne pas tirer plus d'une unité de charge du Bus. Les fonctions auto alimentées sont plus faciles à concevoir au niveau de la spécification car il ne peut guère y avoir de problèmes en ce qui concerne l'alimentation électrique. La charge alimentée par le Bus à une seule unité permet la détection de l'énumération d'appareil sans avoir besoin d'une alimentation principale/secondaire du secteur.

Aucun appareil USB, qu'il soit alimenté par le Bus ou bien auto alimentée ne peut piloter V_{Bus} sur son port face amont. Si V_{Bus} est perdu, l'appareil a une durée de 10 secondes pour retirer l'alimentation des résistances de rappel de D+/D- utilisées pour l'identification de la vitesse.

Les autres considérations de VBUS sont l'appel de courant qui doit être limité. Ceci est souligné dans le paragraphe 7.2.4.1 des spécifications USB et est souvent laissé de côté. L'appel de courant est soutenu au niveau de la capacité de tête de votre appareil entre V_{BUS} et la masse. Les spécifications précisent par conséquent que la capacité de découplage maximum que vous pouvez avoir sur votre appareil est de 10µF. Quand vous déconnectez l'appareil après que le courant soit passé par le câble USB inductif, une grande tension de retour peut se produire sur l'extrémité ouverte du câble. Pour éviter ceci, on recommande une capacité de découplage V_{BUS} minimum de 1µF.

Pour l'appareil habituel alimenté sur un Bus, celui-ci ne peut drainer plus de 500 mA, ce qui est fort raisonnable.

2.6. COURANT DE VEILLE :

Le mode Veille est obligatoire sur tous les appareils. Pendant son temps d'action, d'autres contraintes surviennent. Le courant maximum de veille est proportionnel à l'unité de charge. Pour un appareil d'une unité de charge (par défaut) le courant de veille maximum est de 500µA. Ceci comprend le courant dû aux résistances de rappel sur le Bus. Au niveau du Hub, D- et D+ possèdent des résistances de rappel niveau bas de 15 kOhms. Pour des raisons de consommation électrique, la résistance de rappel niveau bas de l'appareil est montée en série avec la résistance de rappel niveau haut de 1,5 kOhm, totalisant ainsi une charge de 16,5 kOhms sur VTERM habituel de 3,3V. Par conséquent cette résistance draine un courant de 200 µA avant même que l'on commence.

Beaucoup de développeur demande au forum des fournisseurs USB, quelles sont les complications si l'on dépasse cette limite? On comprend bien, que la plupart des Hôtes et des Hubs n'ont pas la possibilité de détecter de surcharge de cette importance et ainsi si vous drainez peut-être 5 mA ou même 10 mA cela devrait aller, tout en gardant à l'esprit qu'à la fin de la journée, votre appareil a enfreint la spécification USB. Toutefois en fonctionnement normal, si vous essayez de dépasser les 100 mA ou la charge permise qui vous est indiqué, alors attendez-vous à ce que le Hub ou Hôte le détecte et déconnecte votre appareil, dans l'intérêt de l'intégrité du Bus.

Bien sûr, ces problèmes de conception peuvent être évités si vous choisissez de concevoir un appareil auto alimenté. Les courants de veille peuvent ne pas être d'un grand intérêt pour les ordinateurs de bureau mais avec l'introduction de la spécification On-The-Go, nous allons commencer à voir des Hôtes USB construit dans les téléphones mobiles et agendas électroniques. La consommation électrique utilisée à partir de ces appareils affectera défavorablement la durée de vie de la batterie.

2.7. ACCES AU MODE VEILLE :

Un appareil USB entrera en veille lorsqu'il n'y a aucune activité sur le Bus pendant plus de 3ms. Il dispose ensuite de 7ms de plus pour éteindre l'appareil et ne prendre que le courant de veille désigné, ne prenant ainsi que le courant de veille nominal à partir du Bus 10ms après que l'activité du Bus ce soit arrêté. Afin de le maintenir connecté à un Hub ou à un Hôte mis en veille, l'appareil doit encore fournir de l'alimentation à ces résistances de rappel de sélection de vitesse pendant le mode veille.

L'USB possède un démarrage de trames de bits (frame packet) ou bien d'entretien qui sont envoyés périodiquement sur le Bus. Ceci empêche un Bus inutilisé d'entrer dans le mode veille en l'absence de données.

- Un Bus haute vitesse enverra une trame toutes les 125.0 µs ±62.5 ns.
- Un Bus pleine vitesse enverra une trame toutes les 1.000 ms ±500 ns.
- Un Bus basse vitesse aura un dispositif d'entretien qui est un EOP (End Of Packet ou Fin De Paquet) toutes les 1ms simplement en l'absence de données basse vitesse.

Le terme veille Global (Global Suspend) est utilisé lorsque le Bus USB entier entre collectivement dans le mode veille. Cependant les appareils sélectionnés peuvent être mis en veille en ordonnant au Hub sur lequel l'appareil est aussi connecté. On fait référence à cette opération comme mode "veille sélective".

L'appareil reprendra son fonctionnement quand il recevra tout signal qui n'est pas en attente. Si un appareil possède une mise en service de réveil retardé, alors il devra signaler à l'Hôte de reprendre à partir du mode veille.

2.8. LES PROTOCOLES USB :

Contrairement à la RS232 et des interfaces sérielles similaires où le format des données envoyées n'est pas défini, l'USB est composé de plusieurs couches de protocoles. En fait la plupart des Circuits Intégrés contrôleur d'USB s'occuperont de la couche inférieure, la rendant ainsi presque invisible au regard du concepteur final.

Chaque transaction USB consiste d'un :

- Paquet Jeton (Token) (en tête définissant ce qu'il attend par la suite)
- Paquet DATA optionnel (contenant la " charge utile " (payload))
- Paquet d'Etat (utilisé pour valider les transactions et pour fournir des moyens de
- corrections d'erreurs).

Comme nous en avons déjà discuté, l'USB est un Bus géré par l'hôte. L'hôte initie toutes les transactions. Le premier paquet, aussi appelé Jeton est produit par l'hôte pour décrire ce qui va suivre et si la transaction de données sera en lecture ou écriture et ce que sera l'adresse de l'appareil et la terminaison désignée. Le paquet suivant est généralement un paquet de données transportant la " charge utile " et est suivi par un paquet " poignée de mains " (handShaking), signalant si les données ou le jeton ont été reçus correctement ou si la terminaison est bloquée, ou n'est pas disponible pour accepter de données.

2.9. LES CHAMPS DE PAQUET USB ORDINAIRES :

Les données sur le BUS USB sont transmises avec le bit LSB en premier. Les paquets USB se composent des champs suivants :

2.9.1. Le champs SYNC :

Tous les paquets doivent commencer avec un champ Sync. Le champ Sync fait de 8 bits de long pour la basse et pleine vitesse ou 32 bits pour la haute vitesse est utilisé pour synchroniser l'horloge du récepteur avec celle de l'émetteur. Les 2 derniers bits indiquent l'endroit où le champ PID commence.

2.9.2. Le champs PID :

PID signifie Paquet ID (identificateur de paquet). Ce champ est utilisé pour identifier le type de paquet qui est envoyé. Le **tableau 2.5** montre les valeurs possibles.

Groupe	Valeur PID	Identificateur Paquet
Token Jeton	0001	OUT Token
	1001	IN Token
	0101	SOF Token
	1101	SETUP Token
Data Données	0011	DATA0
	1011	DATA1
	0111	DATA2
	1111	MDATA
Handshake Poignée De mains	0010	ACK Handshake
	1010	NAK Handshake
	1110	STALL Handshake
	0110	NYET(No Response Yet)
Special	1100	PREamble
	1100	ERR
	1000	Split
	0100	Ping

Tableau 2.5. *Les différentes valeurs du PID*

SOF = Start of Frame; Début de Trame
SETUP = Installation, configuration
ACK = Acknowledge; Validation
NAK = No Acknowledge; Pas de validation
STALL = Bloqué
PREamble = Synchroniseur initial

Split = Partager, Fractionner
Ping = S'assurer d'une bonne connexion

Il y a 4 bits pour le PID, toutefois pour s'assurer qu'il a été reçu correctement, les 4 bits sont complémentés et répétés faisant un PID de 8 bits au total. Le format résultant figure ci-dessous.

PID_0	PID_1	PID_2	PID_3	$nPID_0$	$nPID_1$	$nPID_2$	$nPID_3$

Figure 2.18. *Structure du champ PID*

2.9.3. Le champs ADDR :

Le champ adresse détermine à quel appareil le paquet est destiné. Sa longueur de 7 bits, lui permet de supporter 127 appareils. L'adresse 0 n'est pas valide, tant qu'un appareil qui n'a pas encore d'adresse attribuée, doit répondre aux paquets envoyés d'adresse 0.

$Addr_0$	$Addr_1$	$Addr_2$	$Addr_3$	$Addr_4$	$Addr_5$	$Addr_6$

Figure 2.19. *Structure du champ ADDR*

2.9.4. Le champs ENDP :

Le champ de terminaison est composé de 4 bits, autorisant 16 terminaisons possibles. Les appareils basse vitesse, toutefois peuvent seulement avoir 2 terminaisons additionnelles au-dessus du canal de communication par défaut (4 terminaisons maximales).

$Endp_0$	$Endp_1$	$Endp_2$	$Endp_3$

Figure 2.20. *Structure du champ ENDP*

2.9.5. Le champs CRC :

Les Contrôles à Redondance Cyclique sont exécutés sur les données à l'intérieur du paquet de charge utile. Tous les paquets jetons ont un CRC de 5 bits tandis que les paquets de données ont un CRC de 16 bits.

2.9.6. Le champs EOP :

Fin de Paquet. Signalé par une sortie unique zéro (SE0) pendant une durée approximative de 2 bits suivie par un " J " d'une durée de 1 bit.

2.10. LES TYPES DE PAQUET USB :

L'USB a quatre types différents de paquet. Les paquets jetons indiquent le type de la transaction qui va suivre, les paquets de données contiennent la charge utile, les paquets " poignée de mains " sont utilisés pour valider les données ou rapporter les erreurs et les paquets début de trame (SOF) indiquent le commencement d'une nouvelle trame.

2.10.1. Les paquets jetons :

Il y a 3 sortes de paquets Jetons :

- **In :** Informe l'appareil USB que l'hôte veut lire des informations.
- **Out** : Informe l'appareil USB que l'hôte veut envoyer des informations.
- **Setup :** Utilisé pour commencer les transferts de commande.

Les paquets jetons doivent se conformer au format suivant :

Sync	PID	ADDR	ENDP	CRC5	EOP

Figure 2.21. *Structure d'un paquet jeton*

2.10.2. Les paquets de données :

Il y a 2 sortes de paquets de données, chacun étant capable de transmettre plus de 1024 octets de données :

- Data0
- Data1

Le mode haute vitesse définit 2 autres PIDs de données, DATA2 et MDATA. Les paquets de données ont le format suivant :

Sync	PID	DATA	CRC16	EOP

Figure 2.22. *Structure d'un paquet de données*

La taille maximale de données " charge utile " pour les appareils basse vitesse est de 8 octets. La taille maximale de données " charge utile " pour les appareils pleine vitesse est de 1023 octets. La taille maximale de données " charge utile " pour les appareils haute vitesse est de 1024 octets. Les données doivent être envoyées en multiple d'octets.

2.10.3. Les paquets " poignée de mains " :

Il y a 3 sortes de paquets " poignée de mains " qui font simplement partie du PID :

- **ACK** : validant que le paquet a été reçu correctement.
- **NAK :** rapporte que temporairement l'appareil ne peut ni envoyer ou recevoir des données. Aussi utilisé pendant les transactions d'interruptions pour avertir l'hôte qu'il n'a pas de données à envoyer.
- **STALL** (Bloqué) : L'appareil se retrouve dans un état qui va exiger l'intervention de l'hôte.

Les paquets " poignée de mains " ont le format suivant :

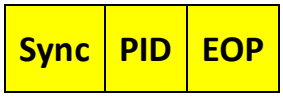


Figure 2.23. *Structure d'un paquet « poignée de mains »*

2.10.4. Les paquets début de trame (SOF) :

Le paquet SOF composé d'une trame de 11 bits est envoyé par l'hôte toutes les 1ms ± 500ns sur un bus pleine vitesse ou bien toutes les 125µs ± 0,0625µs sur un bus haute vitesse.

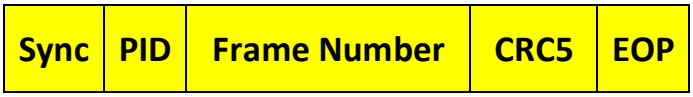


Figure 2.24. *Structure d'un paquet début de trame*

2.11. LES FONCTIONS USB :

Quand nous pensons à un appareil USB, nous pensons à un périphérique USB, mais un appareil USB peut signifier un appareil émetteur / récepteur USB sur l'hôte ou périphérique, un HUB USB ou un circuit intégré contrôleur d'hôte ou un appareil périphérique USB. Par conséquent le standard fait références aux fonctions USB qui peuvent être considérées comme appareil USB qui fournissent une possibilité ou une

fonction comme une imprimante, un lecteur Zip, un scanner, un modem ou un autre périphérique.

La plupart des fonctions USB manipulent les protocoles USB bas niveau jusqu'à la couche transaction dans le silicium.

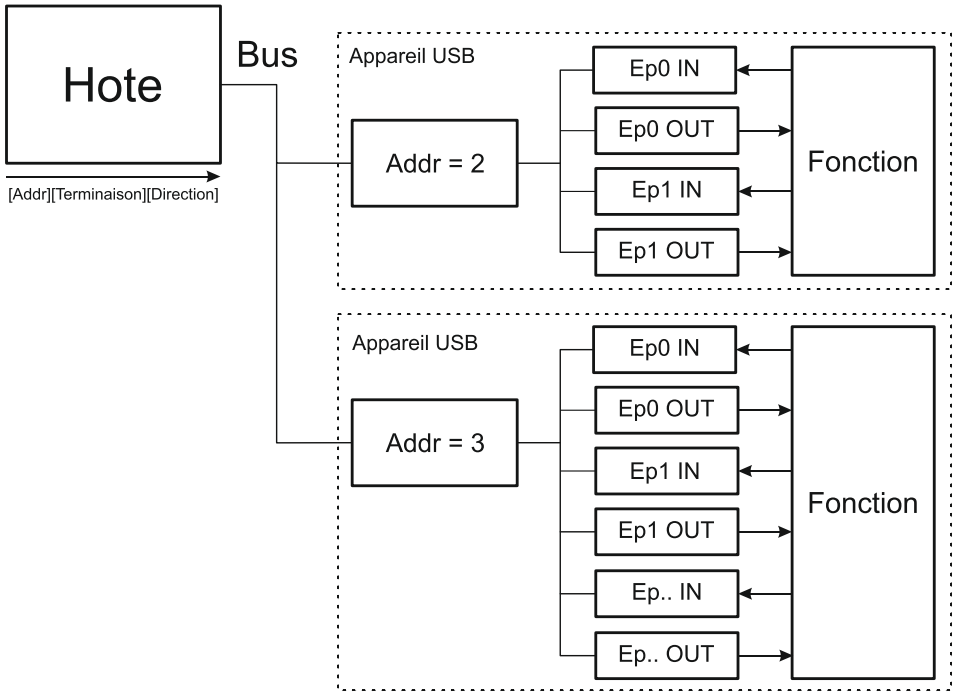


Figure 2.25. *Exemple de fonction USB*

La plupart des fonctions auront des séries de tampons (buffers), généralement de 8 octets de long. Chaque tampon appartiendra à une terminaison EP0 IN, EP0 OUT etc... Supposons par exemple, que l'hôte envoie une demande de descripteur d'appareil. La fonction matérielle lira le paquet d'installation et déterminera à partir du champ adresse si le paquet est pour lui-même, et si c'est le cas, il copiera la " charge utile " du paquet de données suivant au tampon de la terminaison appropriée, dictée par la valeur dans le champ de la terminaison du jeton d'installation. Il enverra ensuite un paquet " poignée de mains " pour valider la réception de l'octet et génèrera une interruption interne à l'intérieur du semi-conducteur / microcontrôleur pour la terminaison appropriée signalant qu'il a reçu un paquet. C'est un principe déjà intégré dans la matière (silicium).

Le logiciel a maintenant une interruption, et doit lire le contenu du tampon de terminaison et analyser la demande de descripteur d'appareil.

2.12. LES TERMINAISONS (ENDPOINT) :

Les terminaisons peuvent être décrites comme émetteurs ou récepteurs de données. Du fait que le bus est régi par l'hôte, les terminaisons se présentent à la fin de la chaîne de communications sur la fonction USB. Au niveau de la couche logicielle, le pilote (driver) logiciel de votre appareil va envoyer, par exemple, un paquet à vos appareils EP1. A la sortie de l'hôte, la donnée aboutira au tampon EP1 OUT. Votre microprogramme pourra alors lire à loisir cette donnée. S'il veut retourner la donnée, la fonction ne peut pas simplement écrire sur le BUS comme celui-ci est contrôlé par l'hôte. Par conséquent il écrit la donnée dans EP1 IN qui s'installe dans le tampon jusqu'à ce que l'hôte envoie un paquet IN à cette terminaison demandant la donnée. Les terminaisons peuvent être aussi considérées comme l'interface entre le matériel de l'appareil de fonction et le microprogramme s'exécutant sur ce même appareil.

Tous les appareils doivent prendre en charge la terminaison zéro. C'est la terminaison qui reçoit la totalité de la commande des appareils et des demandes d'états pendant l'énumération et tant que l'appareil est opérationnel sur le bus.

2.13. LES CANAUX DE COMMUNICATIONS (PIPES) :

Tandis que l'appareil envoie et reçoit des données sur une succession de terminaisons, le logiciel client transfère des données à travers des canaux de communications. Un canal de communication (Pipe) est une connexion logique entre l'hôte et les terminaisons. Les canaux de communications auront aussi un ensemble de paramètres qui leur seront associés tels que : combien de bande passante leur est allouée, quel type de transfert (Commande, Bloc, Iso ou Interruption) ils utilisent, la direction du flux de données et les tailles maximales du paquet / tampons. Par exemple le canal de communication par défaut est un canal bidirectionnel composé d'une terminaison zéro IN et d'une terminaison zéro OUT avec un type de transfert de commande.

L'USB définit 2 types de canaux de communications :

- **Les flux de données (Stream Pipes)** n'ont pas de format USB défini. Cela veut dire que vous pouvez envoyer n'importe quelle sorte de données par un flux de données et que vous pouvez rapporter les données de sorties à l'autre extrémité. Les données circulent séquentiellement et ont une direction prédéfinie, soit en entrée soit en sortie. Les flux de données supporteront les types de transferts en Bloc, Isochrone et d'Interruptions. Les flux de données peuvent soit être contrôlés par l'hôte ou l'appareil.

- **Les canaux de messages (Message Pipes)** ont un format USB défini. Ils sont contrôlés par l'hôte et sont initiés par une demande émanant de l'hôte. Les données sont ensuite transférées dans la direction voulue, dictées par la demande. Par conséquent les canaux de messages permettent aux données de circuler dans les deux directions mais ne prendront seulement en charge que les transferts de commande.

2.14. LES TYPES DE TERMINAISONS (ENDPOINTS) :

La spécification du Bus Série Universel définit 4 types de transferts ou de terminaisons :

- Transferts de commande
- Transferts d'interruption
- Transferts Isochrone
- Transferts en Bloc (BULK)

2.14.1. Les transferts de commande :

Les Transferts de commande sont régulièrement utilisés pour les opérations de commande et d'état. Ils sont essentiels pour installer un appareil USB avec toutes les fonctions d'énumération qui seront exécutés en utilisant les Transferts de commande. Ils surviennent généralement en paquets directs et par rafales qui sont initiés par l'hôte et utilisent le meilleur rendement de livraison. La longueur du paquet du Transfert de commande pour appareil basse vitesse doit être de 8 octets, les appareils pleine vitesse autorise une taille de paquet de 8, 16, 32 ou 64 octets et les appareils haute vitesse doivent avoir une taille de paquet de 64 octets.

Un Transfert de commande peut avoir plus de 3 étapes.

2.14.1.1. L'étape d'installation (Setup Stage) :

Elle se passe lorsqu'une demande est envoyée. Elle est composée de 3 paquets. Le jeton (token) d'installation envoyé le premier est celui qui contient l'adresse et le numéro de la terminaison. Le paquet de données est envoyé après et a toujours un type PID de Data0 et inclut un paquet d'installation qui détaille le type de la demande. Nous détaillerons le paquet d'installation plus tard. Le dernier paquet est une poignée de mains utilisé pour valider la bonne réception ou pour indiquer une erreur. Si la fonction reçoit correctement la donnée d'installation (CRC et PID etc...Ok) elle répond avec ACK, sinon elle ignore la donnée et n'envoie pas un paquet de poignée de mains. Les fonctions ne peuvent pas émettre un paquet STALL ou NAK en réponse à un paquet d'installation.

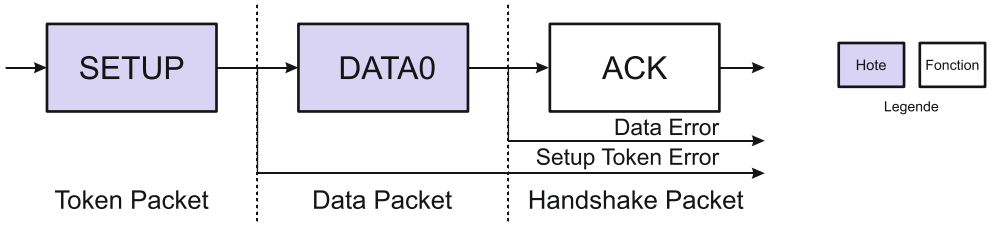


Figure 2.26. *Etape d'installation*

2.14.1.2. L'étape de données (Data Stage) :

Elle est facultative consiste en un ou plusieurs transferts IN (Entrée) ou OUT (sorties). La demande d'installation indique la quantité de données qui doit être envoyée dans cette étape. Si elle dépasse la taille maximale du paquet, les données seront envoyées en plusieurs transferts, chacune ayant la longueur maximale du paquet à l'exception du dernier paquet.

L'étape de données comporte 2 scénarios différents selon la direction du transfert de données :

- **IN** (Entrée): Quand l'hôte est prêt à recevoir les données de commande, il émet un jeton (token) IN. Si la fonction reçoit le jeton IN avec une erreur, c'est-à-dire que le PID ne correspond pas avec les bits inversés du PID, il ignore donc le paquet. Si le jeton est reçu correctement, l'appareil peut soit répondre avec un paquet de données contenant les données de commande à envoyer, soit un paquet "d'arrêt" signalant que la terminaison a eu une erreur soit un paquet NAK signalant à l'hôte que la terminaison fonctionne, mais provisoirement n'a pas de données à envoyer.
- **OUT** (Sortie): Quand l'hôte a besoin d'envoyer à l'appareil un paquet de données de commande, il émet un jeton OUT suivi par un paquet de données contenant les données de commande comme "charge utile" (payload). Si une partie du jeton OUT ou du paquet de données est altéré alors la fonction ignore le paquet. Si le buffer de terminaison de la fonction était vide et qu'il a cadencé les données dans le buffer de terminaison, il produit un ACK avisant l'hôte qu'il a bien reçu les données. Si le buffer de terminaison n'est pas vide à cause du traitement du paquet précédent, alors la fonction retourne un NAK. Toutefois si la terminaison comporte une erreur et que son bit "halt" ai été positionné, elle retourne un STALL (Bloqué).

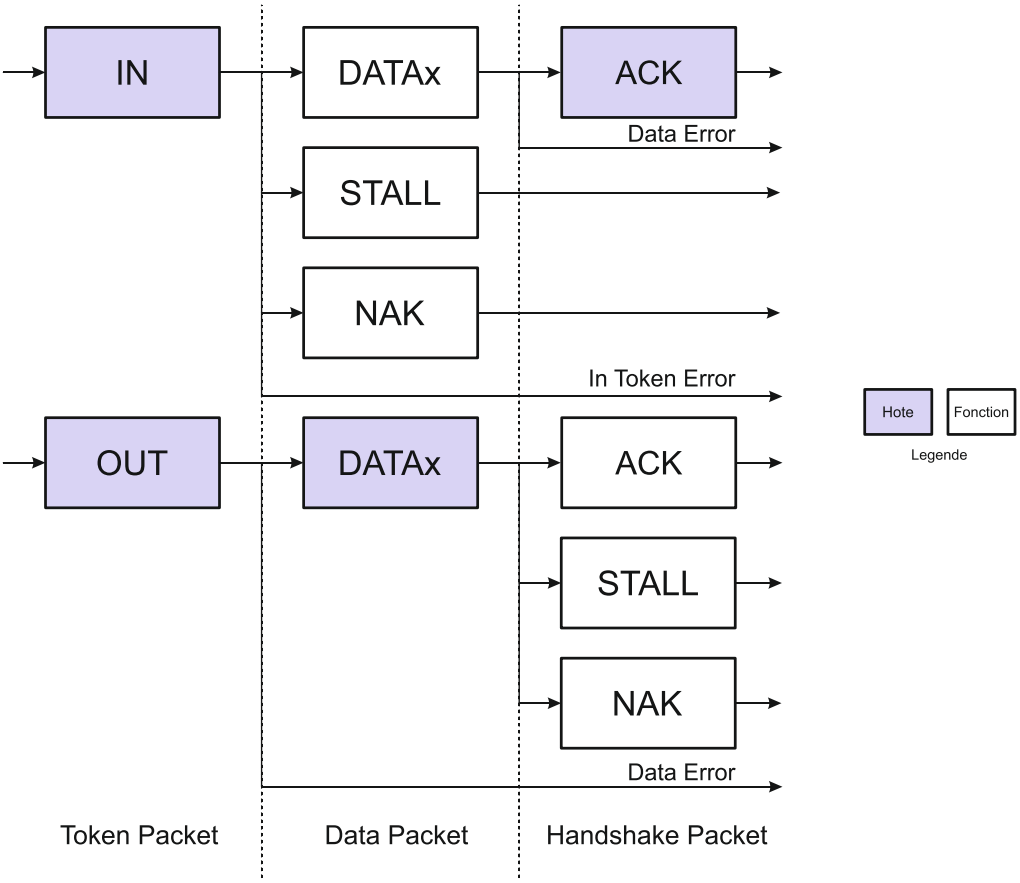


Figure 2.27. *Etape de données*

2.14.1.3. L'étape d'état (Status Stage) :

Elle rend compte des états de l'ensemble des demandes et cette fois encore selon la direction du transfert. Le rapport d'état est toujours réalisé par la fonction.

- **IN** (Entrée): Si l'hôte envoie un (ou des) jeton(s) IN pendant l'étape de données pour recevoir des données, alors l'hôte doit valider la bonne réception de ces données. Ceci est réalisé par l'hôte qui envoie un jeton OUT suivi par un paquet de données de longueur nul. La fonction peut maintenant rendre compte de son état dans l'étape poignée de mains. Un ACK indique que la fonction a achevé la commande et qu'elle est maintenant prête à accepter une autre commande. Si une erreur s'est produite pendant l'exécution de cette commande, alors la fonction émettra un STALL (Bloqué). Toutefois si la fonction continue l'exécution, elle retourne un NAK indiquant à l'hôte la nécessité de répéter l'étape d'état ultérieurement.

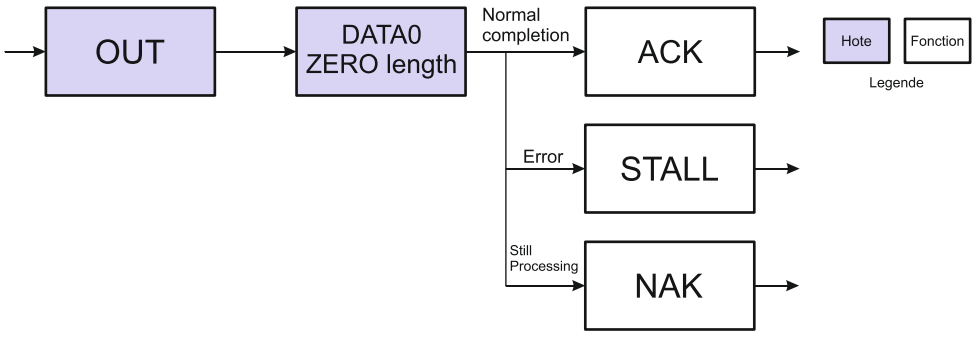


Figure 2.28. *Etape d'état (entrée)*

- **OUT** (Sortie):Si l'hôte envoie un (ou des) jeton(s) OUT pendant l'étape de données pour transmettre des données, la fonction validera la bonne réception des données en envoyant un paquet de longueur nul en réponse à un jeton IN. Toutefois si une erreur intervenait, cela déboucherait sur un STALL ou si la fonction était encore occupée à traiter les données, cela déboucherait sur un NAK demandant à l'hôte de retenter l'étape d'état ultérieurement.

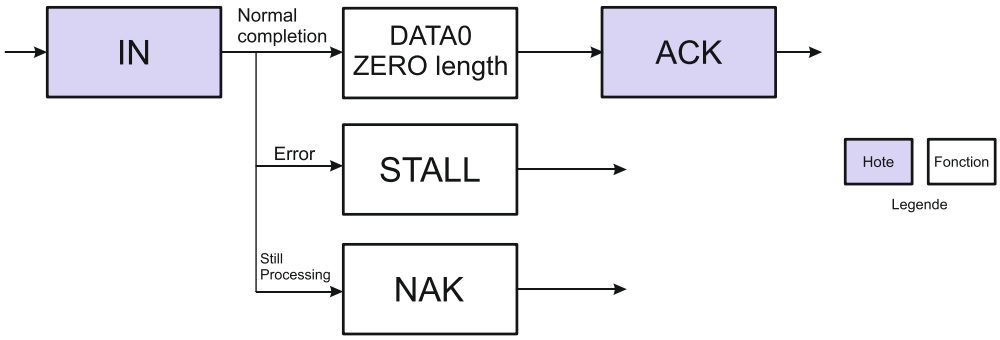


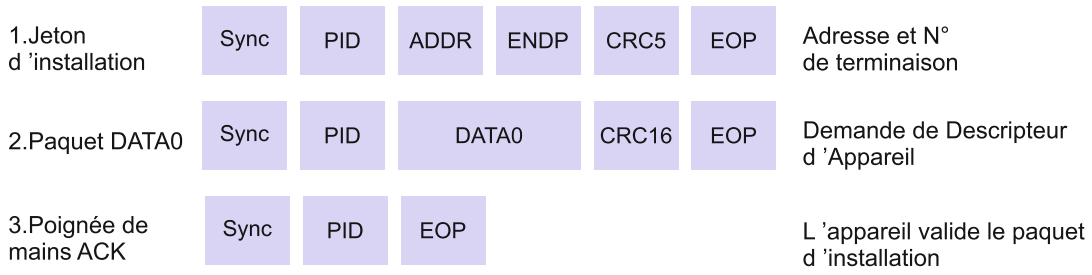
Figure 2.29. *Etape d'état (sortie)*

Pour illustrer ce que nous venons de voir concernant les différentes étapes qui composent le transfert de commande, nous allons expliquer ceci par un exemple.

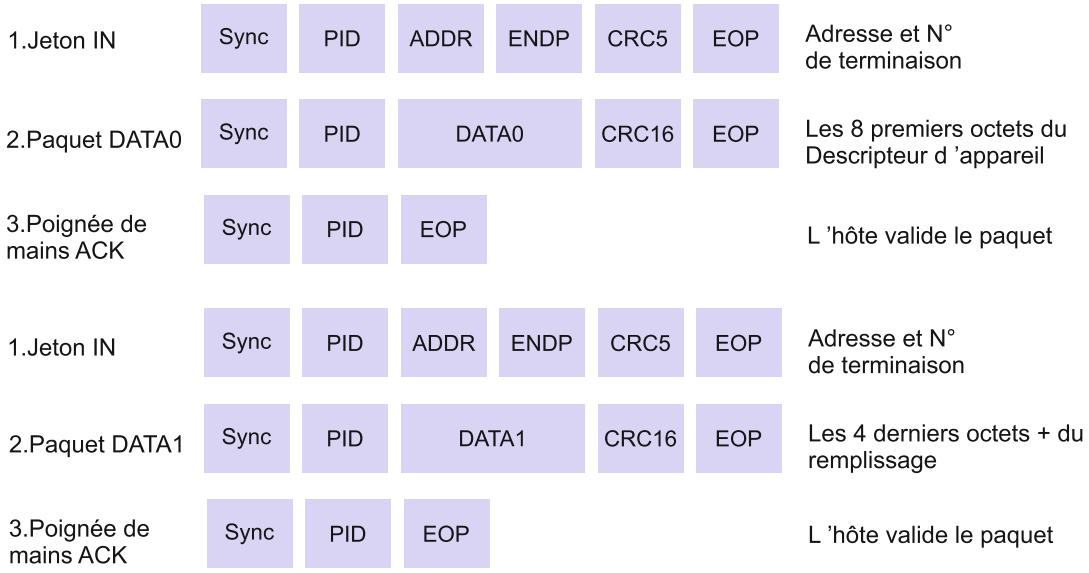
Supposons que l'hôte veuille demander un descripteur d'appareil pendant l'énumération. Les paquets qui sont envoyés sont les suivants:

L'hôte enverra un jeton d'installation disant à la fonction que le paquet suivant est un paquet d'installation. Le champ adresse fixera l'adresse de l'appareil à qui l'hôte a demandé le descripteur. Le numéro de la terminaison devrait être un zéro, indiquant une ligne défectueuse. L'hôte enverra alors un paquet DATA0. Celui-ci aura 8 octets de "charge utile" (payload) correspondant à la Demande de Descripteur d'Appareil comme souligné au chapitre 9 de la spécification USB. La fonction USB annoncera alors que le paquet d'installation a été lu correctement et sans aucune erreur. Si le paquet était reçu altéré,

l'appareil ignorerait tout simplement ce paquet. L'hôte renverra alors le paquet après un court délai.

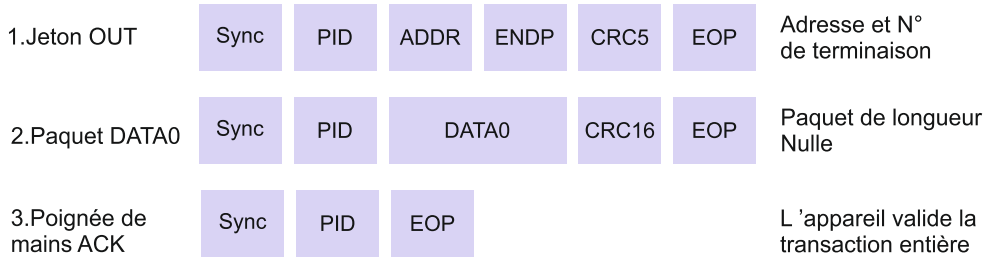


Les 3 paquets ci-dessus représentent la première transaction USB. L'appareil USB décodera maintenant les 8 octets reçus, et déterminera que c'était une demande de Descripteur d'Appareil. L'Appareil tentera d'envoyer le Descripteur d'Appareil, qui sera la transaction USB suivante :



Dans ce cas nous assumons que la taille maximale de "charge utile" (payload) est de 8 octets. L'hôte envoie le jeton IN, disant à l'Appareil qu'il peut maintenant envoyer des données pour cette terminaison. Comme la taille maximale du paquet est de 8 octets, nous devons scinder les 12 octets du Descripteur d'Appareil en morceaux avant de pouvoir les envoyer. Chaque morceau doit être de 8 octets excepté la dernière transaction. L'hôte validera chaque paquet de données que nous lui enverrons.

Une fois que le Descripteur d'Appareil est envoyé, il s'ensuit une transaction d'état. Pour le cas où les transactions seraient réussies, l'hôte enverra un paquet de longueur Nul indiquant que l'ensemble de transactions était correct. La fonction répond alors à ce paquet de longueur Nul indiquant son état.



2.14.2. Les transferts d'interruption :

Celui qui eut une expérience de demandes d'interruption sur microcontrôleurs saura que les interruptions sont générées par l'appareil. Toutefois sous USB, si un appareil demande l'attention de l'hôte, il doit attendre que l'hôte l'interroge avant de signaler qu'il a besoin d'une attention urgente.

Les transferts d'interruption sont caractérisés par :

- Temps de retard (ou Latence) garanti
- Ligne de flux - Unidirectionnel
- Détection d'erreur et nouvel essai sur période suivante

Les transferts d'interruptions sont communément non périodiques, les petits appareils qui ont initié la communication ont besoin de temps de retard limité.

- La taille maximale de "charge utile" (payload) pour des appareils basse vitesse est de 8 octets
- La taille maximale de "charge utile" (payload) pour des appareils pleine vitesse est de 64 octets
- La taille maximale de "charge utile" (payload) pour des appareils haute vitesse est de 1024 octets

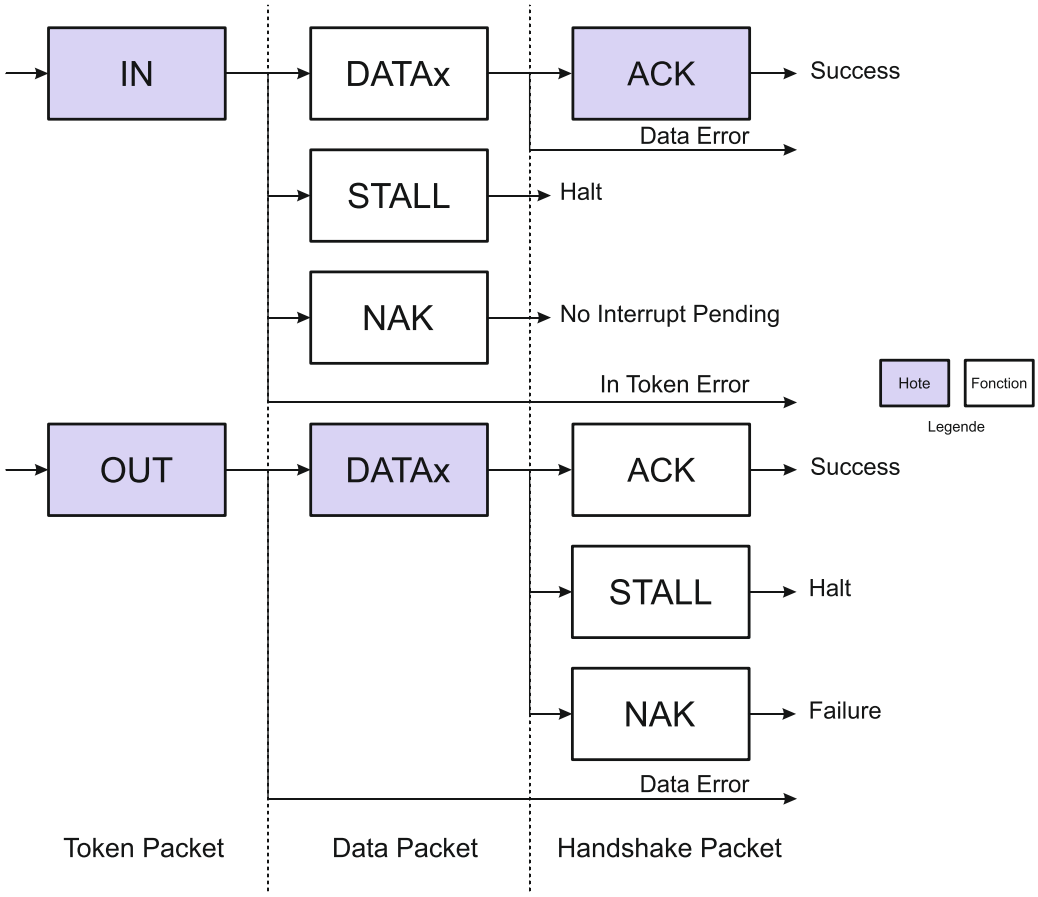


Figure 2.30. *Transaction d'interruption d'entée (IN) et de sortie (Out)*

- **IN:** L'hôte interrogera périodiquement la terminaison d'interruption. Le taux d'interrogation est spécifié dans le descripteur de terminaison qui sera examiné plus tard. Chaque interrogation obligera l'hôte à envoyer un jeton IN. Si le jeton IN est altéré, la fonction ignore le paquet et continue la surveillance du Bus pour de nouveaux jetons.
  Si une interruption a été mise en attente par l'appareil, la fonction enverra un paquet Data contenant des données ayant rapport à l'interruption quand il recevra le jeton IN. Sur des réceptions au niveau de l'hôte, celui-ci retournera un ACK. Toutefois si les données sont altérées, l'hôte ne mentionnera aucun état. Si, d'autre part, une condition d'interruption n'était pas présente quand l'hôte interrogeait la terminaison d'interruption avec un jeton IN, alors la fonction signale cet état en envoyant un NAK. Si une erreur se produisait sur cette terminaison, un STALL (Bloqué) serait envoyé en réponse à un jeton IN.

- **OUT:** Quand l'hôte veut envoyer à l'appareil les données d'interruptions, il émet un jeton OUT suivi par un paquet Data contenant les données d'interruption. Si une partie du jeton OUT ou du paquet Data est altéré alors la fonction ignore le paquet. Si le tampon de terminaison de la fonction était vide et qu'il ait cadencé les données

dans le tampon de terminaison il émettrait un ACK prévenant l'hôte qu'il a reçu correctement les données. Si le tampon de terminaison n'est pas vide à cause du traitement d'un paquet précédent, alors la fonction retourne un NAK.
Toutefois si une erreur se produisait à cause de la terminaison et que son bit d'arrêt (Halt) ait été positionné, elle renverrait un STALL (Bloqué).

2.14.3. Les transferts Isochrones :

Les Transferts Isochrones se produisent continuellement et périodiquement. Ils contiennent généralement des informations à durée de vie critique, tel des trains de données audio ou vidéo. S'il y avait un retard ou une reprise de données dans un flot de données audio, alors on pourrait s'attendre à de l'audio par intermittence contenant des signaux transitoires.

Les transferts isochrones sont caractérisés par :

- Un accès garanti à la bande passante USB.
- Un temps d'attente limité.
- Des flux de données - Unidirectionnel.
- La détection d'erreur via le CRC, mais sans reprise ou garantie de livraison.
- Seulement les modes pleines et haute vitesse.
- Pas de données de basculement (basculage, cachées, de commutation).

La taille maximale de données en " charge utile " (payload) est spécifiée dans le descripteur de terminaison d'une terminaison Isochrone. Cela peut être un maximum de 1023 octets pour un appareil pleine vitesse et 1024 octets pour un appareil haute vitesse. Comme la taille maximale de données en " charge utile " va effectuer des exigences de bande passante du Bus, il est prudent de spécifier une taille de " charge utile " modérée. Si vous utilisez de grandes " charges utiles " il peut être aussi plus intéressant de spécifier une série d'interfaces alternatives avec des tailles de charges utiles Isochrones variables. Si pendant l'énumération l'hôte ne peut pas valider votre terminaison Isochrone nommée à cause des restrictions de bande passante, il reste une solution de repli préférable à un échec complet. Les données qui ont été envoyées sur une terminaison Isochrone peuvent être inférieures à la taille pré négociée et peuvent varier en longueur de transaction en transaction.

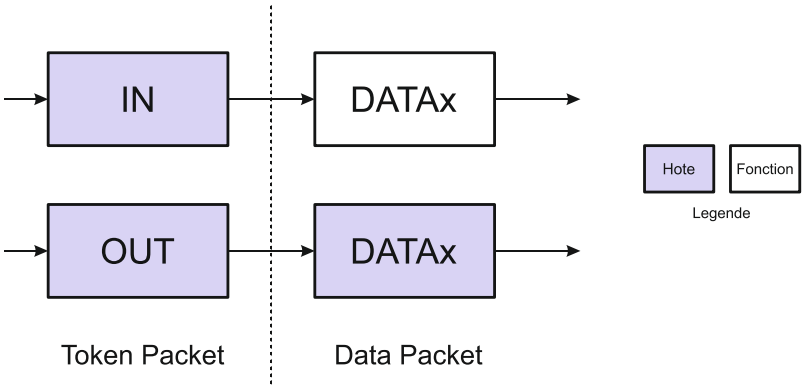


Figure 2.31. *Transaction Isochrone d'entée (IN) et de sortie (Out)*

Les transactions Isochrones n'ont pas d'étape de poignée de mains et ne peuvent pas rendre compte des erreurs ou des conditions STALL / HALT ((Bloqué) / Arrêt).

2.14.4. Les transferts En Bloc (BULK) :

Les Transferts en Bloc peuvent être utilisés pour de grandes quantités de données sporadiques. De tels exemples pourraient inclure un travail d'impression envoyé à une imprimante ou une image provenant d'un scanner. Les Transferts en Bloc se prémunissent de correction d'erreurs sous la forme d'un champ CRC16 sur les données " charge utile " et sur les mécanismes de détection et de retransmission d'erreurs qui assure la transmission et la réception de données de manière infaillible.

Les Transferts en Bloc utiliseront une bande passante de réserve non attribuée sur le Bus après que toutes les autres transactions aient été allouées. Si le Bus est occupé avec de l'Isochrone et/ou de l'interruption, les données en bloc peuvent alors s'écouler doucement sur le Bus. En conséquence, les transferts en bloc devraient seulement être utilisés pour des communications insensibles au temps du fait de la non garantie du temps d'attente.

Les transferts en bloc sont caractérisés par :

- Utilisés pour de grandes quantités de données sporadiques.
- Détection d'erreurs via le CRC, avec la garantie de livraison.
- Pas de garantie de bande passante ou du temps d'attente minimum.
- Des flux de données - Unidirectionnel.
- Seulement les modes pleines et haute vitesse.

Les Transferts en Bloc sont seulement supportés par des appareils pleine et haute vitesse. Pour des terminaisons pleine vitesse, la longueur maximale du paquet en Bloc est soit 8, 16, 32 ou 64 octets. Pour des terminaisons haute vitesse, la longueur maximale du

paquet peut aller jusqu'à 512 octets. Si la charge utile des données est inférieure à la taille maximale du paquet, elle n'a pas besoin d'être remplie avec des zéros. Un transfert en Bloc est considéré comme complet quand il a transféré la quantité exacte de données demandées, ou un paquet inférieur à la taille maximale de la terminaison, ou un paquet de longueur zéro.

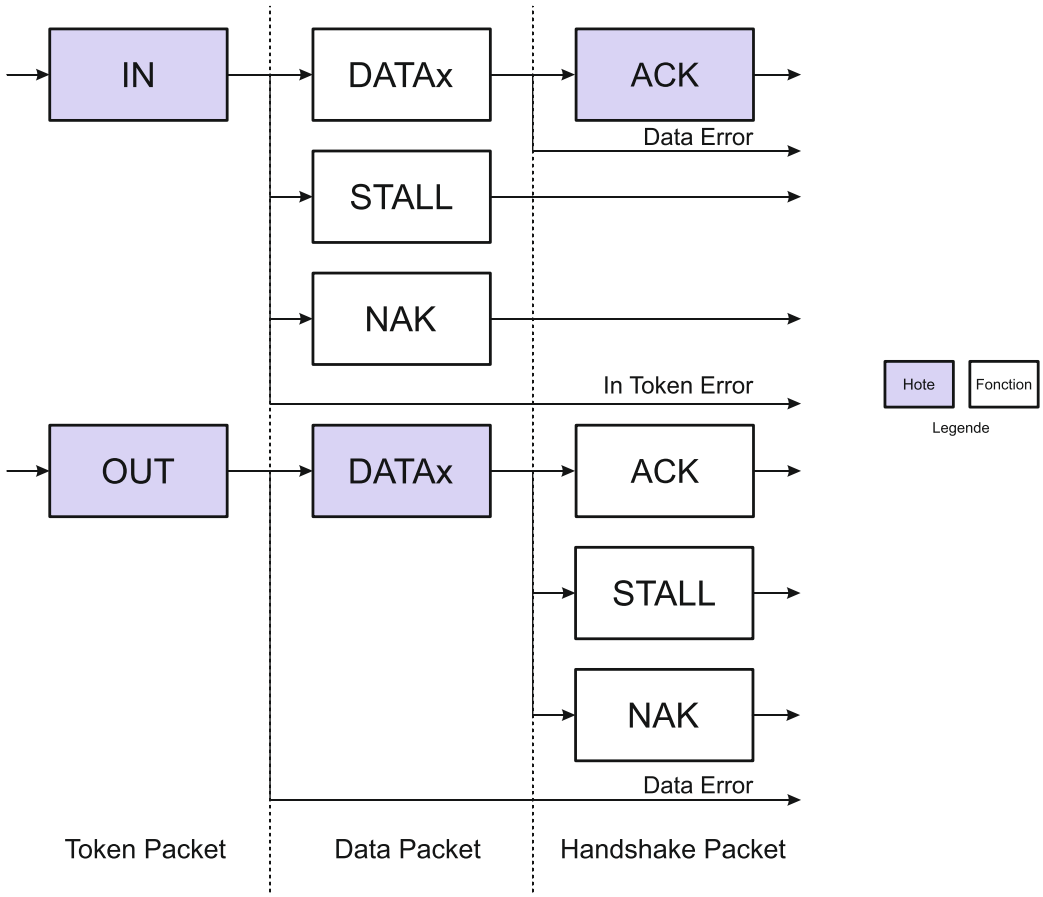


Figure 2.32. *Transaction en bloc d'entée (IN) et de sortie (Out)*

- **IN:** Quand l'hôte est prêt à recevoir des données en Bloc, il émet un jeton IN. Si la fonction reçoit le jeton IN avec une erreur, il ignore le paquet. Si le jeton est reçu correctement, la fonction peut soit répondre avec un paquet DATA contenant les données en Bloc à envoyer ou bien un paquet Stall signalant que la terminaison a eu une erreur ou un paquet NACK signalant à l'hôte que la terminaison travaille, mais provisoirement n'a pas de données à envoyer.
- **OUT:** Quand l'hôte veut envoyer à la fonction un paquet de données en Bloc, il émet un jeton OUT suivi par un paquet DATA contenant les données en Bloc. Si une partie du jeton OUT ou du paquet DATA est altérée, alors la fonction ignore le paquet. Si le tampon de terminaison de la fonction est vide et qu'il a cadencé les données dans le tampon de terminaison, il émet un ACK prévenant l'hôte qu'il a reçu correctement les données. Si le tampon de terminaison n'est pas vide à cause du traitement d'un

paquet précédent, alors la fonction retourne un NAK. Toutefois si la terminaison a eu une erreur et que son bit d'arrêt a été positionné, elle retourne un Stall (Bloqué).

2.15. LA GESTION DE LA BANDE PASSANTE :

L'hôte est responsable de la bande passante du Bus. Elle est faîte par l'énumération (dénombrement) lors de la configuration des terminaisons Isochrones et d'interruptions et durant le fonctionnement du Bus. La spécification place des limites sur le Bus, l'autorisant à allouer plus de 90% pour des transferts périodiques (Interruption et Isochrone) sur un Bus pleine vitesse. Sur des Bus hautes vitesses, cette limitation se réduit à moins de 80% d'une micro-trame qui peut être allouée pour des transferts périodiques.

Aussi vous pouvez assez rapidement voir que si vous avez un Bus hautement saturé avec des transferts périodiques, les 10% restants sont laissés pour les transferts de contrôle et une fois qu'ils ont été attribués, les transferts en Bloc prendront ce qui reste.

2.16. LES DESCRIPTEURS USB :

Tous les appareils USB ont une hiérarchie de descripteurs qui détaillent pour le compte de l'hôte des informations l'instruisant sur la nature de l'appareil, qui l'a réalisé, quelle version USB il supporte, de combien de manières il peut être configuré, le nombre de terminaisons et leurs types etc…

Les descripteurs les plus courants sont :

- Les Descripteurs d'Appareils.
- Les Descripteurs de Configurations.
- Les Descripteurs d'Interfaces.
- Les Descripteurs de Terminaisons.
- Les Descripteurs de Chaînes.

Les appareils USB ne peuvent avoir qu'un seul descripteur d'appareil. Le descripteur d'appareil inclut des informations qui précisent la révision USB à laquelle l'appareil se soumet, les Identificateurs d'Appareils du produit et du constructeur utilisés pour charger les pilotes logiciels appropriés et le nombre possible de configurations que l'appareil peut avoir. Le nombre de configurations indique combien de ramifications de descripteurs de configurations sont appelées à suivre.

Le descripteur de configuration précise des valeurs comme la quantité de puissance qu'utilise cette configuration particulière, si l'appareil est auto-alimenté ou alimenté par le bus et le nombre d'interfaces qu'il possède. Quand un appareil est énuméré, l'hôte lit les

descripteurs d'appareils et peut décider de la configuration à valider. Il peut seulement valider une configuration à la fois.

Par exemple, il est possible d'avoir une configuration d'alimentation de bus de grande puissance et une configuration auto-alimentée. Si l'appareil est branché à un hôte possédant une alimentation électrique secteur, le pilote logiciel de l'appareil choisira peut-être de permettre la configuration d'alimentation du bus de grande puissance tolérant ainsi que l'appareil soit alimentée sans être relié au secteur, cependant s'il est connecté à un laptop (ordinateur portatif) ou à un organiseur personnel, il pourra valider la seconde configuration (auto-alimentée) exigeant de l'utilisateur de brancher son appareil sur un point d'alimentation (secteur).

Les paramètres de configurations ne sont pas limités aux différences d'alimentations. Chaque configuration pourrait être alimentée de la même façon et drainer le même courant, et avoir cependant des combinaisons de terminaisons et d'interfaces différentes. Toutefois il faut tenir compte que la modification de la configuration exige l'arrêt de toute activité sur chaque terminaison. Tandis que l'USB propose cette flexibilité, très peu d'appareils possèdent plus d'une configuration.

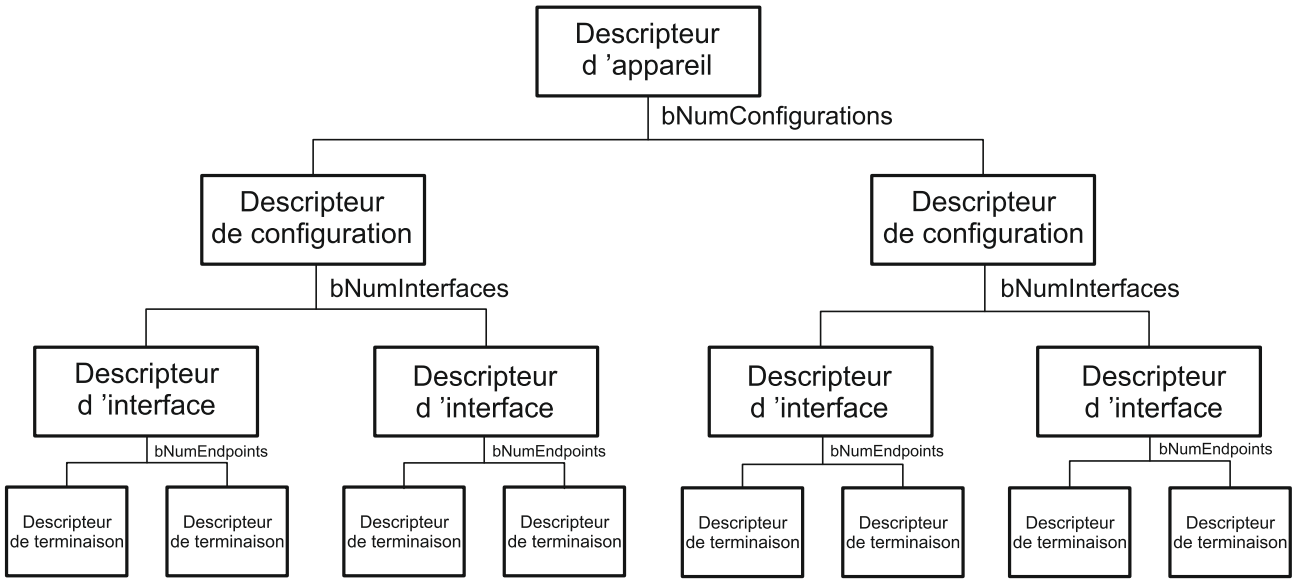


Figure 2.33. *Hiérarchie des différents descripteurs USB*

Le descripteur d'interface peut être vu comme un "en tête" ou un regroupement de terminaison à l'intérieur d'un groupe fonctionnel accomplissant une seule fonctionnalité de l'appareil. Par exemple, vous pouvez avoir un appareil multifonctions : fax / scanner / imprimante. Le descripteur d'interface 1 pourra décrire les terminaisons de la fonction fax,

le descripteur d'interface 2, la fonction scanner et le descripteur d'interface 3 la fonction imprimante. Contrairement au descripteur de configuration, il n'y a pas de limitations à avoir une seule interface validée à la fois. Un appareil peut avoir un ou plusieurs descripteurs d'interfaces validées en même temps.

Les descripteurs d'interfaces ont un champ **bInterfaceNumber** précisant le numéro de l'interface et un **bAlternateSetting** qui autorise l'interface à modifier ses paramètres au vol. Par exemple on peut avoir un appareil avec 2 interfaces, interface 1 et interface 2. Interface 1 à **bInterfaceNumber** mis à 0 indiquant qu'il est le premier descripteur d'interface et un **bAlternativeSetting** de 0.

L'interface 2 aura un **bInterfaceNumber** mis à 1 indiquant qu'il est la seconde interface et un **bAlternativeSetting** de 0 (par défaut). On pourra donc y faire entrer un autre descripteur, comprenant lui aussi un **bInterfaceNumber** mis à 1 indiquant qu'il est la seconde interface, mais cette fois le **bAlternativeSetting** mis à 1, indiquant que ce descripteur d'interface peut représenter un paramètre alternatif à celui de l'autre descripteur d'interface 2.

Quand cette configuration est validée, les 2 premiers descripteurs d'interfaces avec **bAlternativeSettings** égal à 0 sont utilisés. Toutefois pendant le fonctionnement, l'hôte peut envoyer une demande imposée SetInterface (Sélection d'Interface) à l'interface 1 avec un alternative setting (paramètre alternatif) à 1 pour valider l'autre descripteur d'interface.

Cela est plus avantageux que d'avoir 2 configurations, dans le sens où l'on peut transmettre des données via l'interface 0 tandis que l'on change les paramètres de terminaisons associés à l'interface 1 sans affecter l'interface 0.

Chaque descripteur de terminaison est utilisé pour spécifier le type de transfert, la direction, l'intervalle d'interrogation et la taille maximale de paquet pour chaque terminaison. La terminaison 0, la terminaison de commandes par défaut est toujours supposée être une terminaison de commandes et en tant que tel ne possède jamais de descripteur.

Tous les descripteurs relèvent d'un schéma commun. Le premier octet précise la longueur du descripteur, tandis que le second octet indique le type de descripteur. Si la longueur du descripteur est plus petite que ce que définit la spécification, alors l'hôte doit l'ignorer. Toutefois si la taille est plus grande que prévue, l'hôte ignorera les octets supplémentaires et ne commencera à rechercher le prochain descripteur qu'à la fin de celui-ci.

2.16.1. Le descripteur d'appareil (Device Descriptor) :

Le descripteur d'appareil d'un appareil USB représente l'appareil en entier. En conséquence un appareil USB ne peut avoir qu'un seul descripteur d'appareil. Il donne des informations élémentaires et pourtant fondamentales sur l'appareil telles la version USB supporté, la taille maximale de paquet, les identificateurs du constructeur et produits et le nombre de configurations possibles que peut avoir l'appareil.

Décalage	Champ	Taille	Valeur	Description
0	bLength	1	Nombre	Taille du descripteur en octets (12 octets)
1	bDescriptorType	1	Constante	Descripteur d'appareil (0x01)
2	bcdUSB	2	BCD	Numéro de spécification USB auquel l'appareil doit aussi se soumettre.
4	bDeviceClass	1	Classe	Code classe (class) (assigné par USB org) Si égal à 0, chaque interface précise son propre code classe Si égal à 0xFF, le code classe est précisé par le constructeur. Autrement le champ est un code classe valable.
5	bDeviceSubClass	1	Sous Classe	Code sous classe (assigné par USB org)
6	bDeviceProtocol	1	Protocole	Code protocole (assigné par USB org)
7	bMaxPacketSize	1	Nombre	Taille maximale de paquet pour la terminaison Zéro. Les tailles conformes sont 8, 16, 32, 64
8	idVendor	2	ID	IDentification du fournisseur (assigné par USB org)
10	idProduct	2	ID	IDentification du produit (assigné par USB org)
12	bcdDevice	2	BCD	Numéro de version de l'appareil
14	iManufacturer	1	Index	Index de descripteur de chaîne du fabricant
15	iProduct	1	Index	Index de descripteur de chaîne du produit
16	iSerialNumber	1	Index	Index de descripteur de chaîne du numéro de série

17	bNumConfigurations	1	Entier	Nombre de configurations possible

Tableau 2.6. *Format du descripteur d'appareil*

Le champ **bcdUSB** rapporte la version USB la plus haute que peut supporter l'appareil. La valeur est en binaire codé décimal avec un format de 0xJJMM ou JJ est le numéro de version de poids fort, M le numéro de version de poids faible et N correspond au numéro de sous-version c'est à dire USB 2.0 est inscrit comme 0x0200, USB 1.1 comme 0x0110 et USB 1.0 comme 0x0100.

Les champs **bDeviceClass, bDeviceSubClass** et **bDeviceProtocol** sont utilisés par le système d'exploitation pour trouver un pilote logiciel de classe pour votre appareil. Habituellement seul le bdeviceclass est positionné au niveau de l'appareil. La plupart des spécifications de classe choisissent de s'identifier au niveau de l'interface et en conséquence positionnent le bdeviceclass à 0x00. Cela permet à un appareil de supporter plusieurs classes.

Le champ **bMaxPacketSize** rapporte la taille maximale pour la terminaison zéro. Tous les appareils doivent supporter la terminaison zéro.

Le champ **idVendor** et **idProduct** sont utilisés par le système d'exploitation pour trouver un pilote logiciel pour votre appareil. L'identification constructeur (vendor ID) est assignée par USB-IF.

Le champ **bcdDevice** a le même format que **bcdUSB** et est utilisé pour fournir un numéro de version d'appareil. Cette valeur est assignée par le développeur.

Trois descripteurs de chaines de caractères existent pour fournir des détails du fabriquant, du produit et du numéro de série. Il n'y a pas d'obligation à avoir des descripteurs de chaines de caractères. Si aucun descripteur de chaines de caractères n'est présent, il faudrait utiliser un index de zéro.

Le champ **bNumConfigurations** définit le nombre de configuration que l'appareil peut supporter et sa vitesse normale d'exécution.

2.16.2. Le descripteur de configuration (Configuration Descriptor) :

Un appareil USB peut avoir plusieurs configurations différentes alors que la majorité des appareils sont simples et n'en ont qu'une. Le Descripteur de Configuration précise la façon dont l'appareil est alimenté, quelle est sa consommation électrique maximale, le nombre d'interfaces qu'il possède. Il est donc possible d'avoir 2 configurations, quand il est

alimenté par le bus et une autre quand il est alimenté par le secteur. Comme ceci est un "en tête " de descripteur d'interface, il est aussi possible d'avoir une configuration utilisant un mode de transfert différent de celui d'une autre configuration.

Une fois que toutes les configurations ont été examinées par l'hôte, celui-ci enverra une instruction **SetConfiguration** avec une valeur différente de Zéro qui correspondra à la valeur **bConfiguration** de l'une des configurations. Elle est utilisée pour sélectionner la configuration voulue.

Décalage	Champ	Taille	Valeur	Description
0	bLength	1	Nombre	Taille du descripteur en octets (9 octets)
1	bDescriptorType	1	Constante	Descripteur de configuration (0x02)
2	wTotalLength	2	Nombre	Longueur totale en octets de données renvoyées.
4	bNumInterfaces	1	Nombre	Nombre d'interfaces
5	bConfigurationValue	1	Nombre	Valeur à utiliser comme argument pour sélectionner cette configuration
6	iConfiguration	1	Index	Index de descripteur de chaines de caractères décrivant cette configuration.
7	bmAttributes	1	Bitmap	D7 Réservé, mis à 1 (USB 1.0 alimenté par le bus) D6 Auto alimenté D5 Activation d'une station a distance D4..0 Réservé. Mis à 0
8	bMaxPower	1	mA	Consommation électrique maximale en unités de 2 mA.

Tableau 2.7. *Format du descripteur de configuration*

Lors de la lecture du descripteur de configuration, il renvoie la hiérarchie ou l'arborescence complète de configuration qui inclut toute interface apparentée et les descripteurs de terminaisons. Le champ **wTotalLength** indique le nombre d'octets dans la hiérarchie.

Le champ **bNumInterfaces** indique le nombre d'interface présent pour cette configuration.

Le champ **bConfigurationValue** est utilisé par la demande SetConfiguration pour sélectionner cette configuration.

Le champ **iConfiguration** est un index de descripteur de chaine de caractère décrivant la configuration dans un format lisible par l'homme.

Le champ **bmAttributes** précise les paramètres d'alimentation pour la configuration. Si un appareil est auto-alimenté, il positionne D6. Le bit D7 était autrefois utilisé par l'USB 1.0 pour indiquer un appareil alimenté par le bus, ceci est maintenant réalisé par le champ **bMaxPower**. Si un appareil utilise l'énergie du bus, que ce soit un appareil alimenté par le bus ou un appareil auto-alimenté, il doit rapporter sa consommation électrique dans le champ **bMaxPower**. Les appareils peuvent aussi prendre en charge l'activation d'une station à distance qui permettra à l'appareil de réveiller l'hôte quand celui-ci est en mode veille.

Le champ **bMaxPower** définit la consommation électrique maximale que l'appareil peut prendre du bus. Elle est donnée en unités de 2 mA, jusqu'au chiffre maximum de 500 mA environ. La spécification permet à un appareil alimenté par un bus de forte puissance de ponctionner jusqu'à 500 mA à partir de Vbus. Dans le cas où un appareil perd son alimentation externe, il ne doit pas ponctionner plus que ce qui est indiqué dans **bMaxPower**. Il devrait échouer sur toutes opérations nécessitant l'alimentation externe.

2.16.3. Le descripteur d'interface (Interface Descriptor) :

Le Descripteur d'Interface peut être vu comme un " en tête " ou un regroupement de Terminaisons dans un groupe fonctionnel exécutant une simple fonction pour l'appareil.

Décalage	Champ	Taille	Valeur	Description
0	bLength	1	Nombre	Taille du descripteur en octets (9 octets)
1	bDescriptorType	1	Constante	Descripteur de configuration (0x04)
2	bInterfaceNumber	1	Nombre	Numéro d'interface
3	bAlternateSetting	1	Nombre	Valeur utilisée pour sélectionner une configuration de remplacement
4	bNumEndpoints	1	Nombre	Nombre de terminaisons utilisées pour cette interface.
5	bInterfaceClass	1	Class	Code Classe (assigné par USB org)

6	bInterfaceSubClass	1	SubClass	Code Sous Classe (assigné par USB org)
7	bInterfaceProtocol	1	Protocol	Code du Protocole (assigné par USB org)
8	iInterface	1	Index	Index du descripteur de chaine décrivant cette interface.

Tableau 2.8. *Format du descripteur d'interface*

Le champ **bInterfaceNumber** indique l'index du descripteur d'interface. Il devait être pointé à Zéro, et incrémenté une fois pour chaque nouveau descripteur d'interface.

Le champ **bAlternativeSetting** peut être utilisé pour préciser les interfaces de remplacement. Ces interfaces alternatives peuvent être sélectionnées par la demande SetInterface.

Le champ **bNumEndpoints** indique le nombre de terminaisons utilisé par l'interface. Cette valeur devrait exclure la terminaison Zéro et est utilisée pour indiquer le nombre de Descripteurs de terminaisons à suivre.

Les champs **bInterfaceClass, bInterfaceSubClass** et **bInterfaceProtocol** peuvent être utilisés pour préciser les classes prises en compte (par exemple : HID, Communications, mémoire de masse etc…). Ceci permet à plusieurs appareils d'utiliser des " drivers " (pilote logiciel) de classe évitant le besoin d'écrire des " drivers " spécifiques pour votre appareil.

Le champ **iInterface** permet d'avoir une description textuelle de l'interface.

2.16.4. Le descripteur de terminaison (Endpoint Descriptor) :

Les Descripteurs de Terminaison sont utilisés pour décrire les terminaisons autres que la terminaison zéro. La terminaison zéro est toujours censée être une terminaison de commande et est configuré avant que n'importe quel autre descripteur ne soit sollicité. L'Hôte utilisera l'information renvoyée par ces descripteurs pour déterminer les besoins de bande passante du bus.

Décalage	Champ	Taille	Valeur	Description
0	bLength	1	Nombre	Taille du descripteur en octets (7 octets)
1	bDescriptorType	1	Constante	Descripteur de terminaison (0x05)
2	bEndpointAdress	1	Terminaison	Adresse de terminaison Bits 0..3b Numéro de terminaison

				Bits 4..6b Réservés. Mis à 0. Bit 7 Direction ; 0 = Sortie, 1 = Entrée (ignoré pour les terminaisons de commande).
3	bmAttributes	1	Bitmap	Bits 0..1 Type de transfert 00 = Commande 01 = Isochrone 10 = par Bloc 11 = Interruption Bits 2..7 sont réservés Si terminaison Isochrone : Bits 3..2 = Type de synchronisation (mode Iso) 00 = Pas de synchronisation 01 = Asynchrone 10 = Adaptif 11 = Synchrone Bits 5..6 = Type d'utilisation (mode iso) 00 = Terminaison de données 01 = Terminaison de retour (Feedback) 10 = Renvoi explicite de terminaison de données 11 = Réservé
4	wMaxPacketSize	2	Nombre	Taille maximale du paquet que cette terminaison est capable d'envoyer ou de recevoir
6	bIntervale	1	Nombre	Intervalle de temps pour interroger les transferts de données de la terminaison. Valeur en nombre de trames. Ignoré pour les terminaisons par Bloc et de commande. Pour le mode Isochrone il doit être égal à 1 et le champ peut valoir de 1 à 255 pour des terminaisons d'interruptions.

Tableau 2.9. *Format du descripteur de terminaison*

Le champ **bEndpointAddress** indique quelle terminaison ce descripteur décrit.

Le champ **bmAttributes** précise le type de transfert. Cela peut être soit des transferts de type Commande, Interruption, Isochrone ou par Blocs. Si une terminaison Isochrone est précisée, des attributs supplémentaires peuvent être sélectionnés tel que la synchronisation et les types d'utilisations.

Le champ **wMaxPacketSize** indique la taille maximale de charge utile pour cette terminaison.

Le champ **bInterval** est utilisé pour préciser cet intervalle d'interrogation de certains transferts. L'unité est exprimé en trames équivalent ainsi à 1 ms pour des appareils basse / pleine vitesse et 125 µs pour des appareils haute vitesse.

2.16.5. Le descripteur de chaine de caractères (String Descriptor) :

Les Descripteurs de chaînes fournissent une information lisible pour l'homme et sont optionnels. S'ils ne sont pas utilisés, tout champ d'index de descripteurs de chaînes doit être mis à zéro indiquant qu'il n'y a pas de descripteur de chaîne disponible.

Les chaînes de caractères sont codées au format Unicode et les produits peuvent être prévus pour comprendre les diverses langues. L'index de chaîne zéro devra retourner une liste de langues acceptée. On peut trouver une liste USB d'identification de langue dans Universal Serial Bus Language Identifiers (LANGIDs) version 1.0 (Identificateurs de langue sur BUS Série Universel version 1.0).

Décalage	Champ	Taille	Valeur	Description
0	bLength	1	Nombre	Taille du descripteur en octets
1	bDescriptorType	1	Constante	Descripteur de chaîne (0x03)
2	wLANGID[0]	2	Nombre	Langue acceptée code zéro (par exemple 0x0409 Anglais, U.S.)
4	wLANGID[1]	2	Nombre	Langue acceptée code Un (par exemple 0x0C09 Anglais, Australien)
n	wLANGID[n]	2	Nombre	Langue acceptée code x (par exemple 0x0407 Allemand - Standard)

Tableau 2.10. *Format du descripteur de chaîne Zéro*

Le descripteur de chaînes ci-dessus montre le format du descripteur de chaîne zéro. L'Hôte devra lire ce descripteur pour déterminer quelles langues sont disponibles. Si une langue est acceptée, elle peut être référencée en envoyant l'Identification de la langue dans le champ **wIndex** à la demande de Get Descriptor(String).

Toutes les chaînes de caractères à venir tiennent dans le format ci-dessous.

Décalage	Champ	Taille	Valeur	Description
0	bLength	1	Nombre	Taille du descripteur en octets
1	bDescriptorType	1	Constante	Descripteur de chaîne (0x03)
2	bString	n	Unicode	Textes codés unicode

Tableau 2.11. *Format du descripteur de chaîne*

2.17. LE PAQUET D'INSTALLATION :

Tout appareil USB doit répondre aux paquets d'installation sur le canal de communication par défaut. Les paquets d'installation sont utilisés pour la détection et configuration de l'appareil et véhiculent des fonctions courantes telles que la mise en place de l'adresse de l'appareil USB, la demande d'un descripteur d'appareil ou la vérification de l'état d'une terminaison.

Un hôte USB conforme s'attend à ce que toutes les requêtes soient traitées dans une période maximale de 5 secondes. Il précise aussi des temps plus stricts pour des requêtes particulières.

Les requêtes standards d'appareils sans étage de données doivent être accomplies en 50 ms.

Les requêtes standards d'appareils avec un étage de données doivent commencer à renvoyer les données 500 ms après la requête.

- Chaque paquet de données doit être envoyé dans les 500 ms de la transmission réussie du paquet précédent.
- L'étage d'état doit être accompli dans les 50 ms après la transmission du dernier paquet de données.

L'instruction **SetAdress** (qui contient une phase de données) doit traiter l'instruction et retourner l'état dans ces 50 ms. L'appareil a donc 2 ms pour changer d'adresse avant que la prochaine requête ne soit envoyée.

Chaque requête commence avec un paquet d'installation de 8 octets qui a le format suivant :

Décalage	Champ	Taille	Valeur	Description
0	bmRequestType	1	Bitmap	D7 Direction de transfert de phase de données : 0 = Hôte vers l'appareil 1 = Appareil vers hôte D6..5 Type : 00 = Standard 01 = Classe 10 = Constructeur 11 = Réservé D4..0 Destinataire 0 = Appareil 1 = Interface 2 = Terminaison 3 = Autre 4 ..31 Réservés
1	bRequest	1	Valeur	Requête
2	wValue	2	Valeur	Valeur
4	wIndex	2	Index ou Décalage	Index
6	wLength	2	Compteur	Nombres d'octets à transférer s'il y a une phase de données

Tableau 2.12. *Format du paquet d'installation*

Le champ **bmRequestType** déterminera le sens de la requête, le type de la requête et le destinataire désigné. Le champ **bRequest** indique la requête formulée. Le champ **bmRequestType** est généralement analysé et l'exécution est raccordée à un numéro d'identificateurs comme un gestionnaire de requête d'un appareil standard, un gestionnaire de requête d'une interface standard, un gestionnaire de requête de terminaison standard, un gestionnaire de requête d'un appareil de classe, etc…. La façon d'exécuter le paquet d'installation vous appartient. D'autres pourraient choisir d'exécuter en premier **bRequest** puis de déterminer le type et le destinataire basé sur chaque requête.

Les requêtes standards sont communes à tous les appareils USB. Les requêtes de classe sont communes aux classes de drivers (pilote logiciel). Par exemple, tout appareil étant conforme à la classe HID aura un ensemble commun de requêtes spécifiques de classe. Ils

diffèreront d'un appareil conforme à la classe communication et diffèreront encore d'un appareil conforme à la mémoire de masse (de grande capacité de stockage).

Et pour finir, les requêtes définies par le constructeur. Ce sont des requêtes que vous pouvez attribuer en tant que concepteur d'appareil USB. Elles sont normalement différentes d'un appareil à un autre, mais dépendent de votre réalisation et de votre imagination.

Une requête commune peut être dirigé vers des destinataires différents et basée sur les fonctions différentes exécutée par le destinataire. Par exemple une requête standard **GetStatus** peut être dirigée sur l'appareil, l'interface ou la terminaison. Quand elle est dirigée sur un appareil, cette dernière retourne des drapeaux (flags) indiquant l'état de la station d'activation et si l'appareil est auto-alimenté.

Toutefois si la même requête est dirigée sur l'interface il renvoie toujours Zéro, ou bien si elle était dirigée sur une terminaison, l'appareil retournerait le drapeau Halt pour la terminaison.

Les champs **wValue** et **wIndex** permettent le passage de paramètres avec la requête. Le champ **wLength** est utilisé pour préciser le nombre d'octets à transférer au cas où il existerait une phase de donnée.

La section 9.4 de la spécification USB détaille les requêtes " d'appareils standard " exigées qui doivent être appliquées pour chaque appareil USB. Le standard fournit un tableau unique regroupant des articles par requête. Considérant que la plupart des microprogrammes analyseront le paquet d'installation par destinataire, nous choisirons de séparer les requêtes en fonction du destinataire pour une étude et une application plus simple.

2.17.1. Les requêtes d'appareil standard :

Il y a actuellement 8 requêtes d'appareil standard, chacune étant détaillée dans le **tableau 2.13.**

bmRequestType	bRequest	wValue	wIndex	wLength	Données
1000 0000b	GET_STATUS (0x00)	Zéro	Zéro	Deux	Etat de l'appareil
0000 0000b	CLEAR_FEATURE (0x01)	Sélecteur de fonction	Zéro	Zéro	Aucune
0000 0000b	SET_FEATURE (0x03)	Sélecteur de fonction	Zéro	Zéro	Aucune

0000 0000b	SET_ADDRESS (0x05)	Adresse de l'appareil	Zéro	Zéro	Aucune
1000 0000b	GET_DESCRIPTOR (0x06)	Type de Descripteur & Index	Zéro ou ID de Langues	Longueur du Descripteur	Descripteur
0000 0000b	SET_DESCRIPTOR (0x07)	Type de Descripteur & Index	Zéro ou ID de Langues	Longueur du Descripteur	Descripteur
1000 0000b	GET_CONFIGURATION (0x08)	Zéro	Zéro	1	Valeur de Configuration
0000 0000b	SET_CONFIGURATION (0x09)	Valeur de Configuration	Zéro	Zéro	Aucune

Tableau 2.13. *Requêtes d'appareils standards*

La requête **GetStatus** dirigée vers l'appareil retournera 2 octets pendant l'étage de données suivant le format :

Si D0 est à 1, ceci indique que l'appareil est auto alimenté. S'il est à 0, l'appareil est alimenté par le bus. Si D1 est à 1, l'activation à distance de l'appareil est validée et ce dernier peut réveiller l'hôte pendant le mode veille. Le bit d'activation à distance peut être positionné par les requêtes **SetFeature** et **ClearFeature.**

Set Address est utilisé pendant l'énumération pour attribuer une adresse unique à l'appareil USB. L'adresse est précisée dans **wValue** et peut valoir au maximum 127. Cette requête est unique dans le sens ou l'appareil ne positionnera pas son adresse tant que la phase d'état ne sera pas achevée. Toutes les autres requêtes doivent être terminées avant la phase d'état.

Set Descriptor/Get Descriptor est utilisé pour renvoyer le descripteur indiqué dans **wValue**. Une requête pour le descripteur de configuration retournera le descripteur d'appareil et, tous les descripteurs d'interfaces et de terminaisons dans la même requête.

- Les descripteurs de terminaisons ne peuvent pas être accessibles directement par une requête de **GetDescriptor / SetDescriptor**.
- Les descripteurs d'interfaces ne peuvent pas être accessibles directement par une requête de **GetDescriptor / SetDescriptor**.

- Les descripteurs de chaînes incluent une Identification de langues dans **wIndex** pour autoriser le support de plusieurs langues.

GetConfiguration/SetConfiguration est utilisé pour demander ou positionner la configuration de l'appareil actuel. Dans le cas d'une requête **GetConfiguration**, un octet sera renvoyé pendant la phase de donnée indiquant l'état de l'appareil. Une valeur zéro signifie que l'appareil n'est pas configuré et une valeur différente de zéro indique que l'appareil est configuré. **SetConfiguration** est utilisé pour valider un appareil. Il doit contenir la valeur de **bConfigurationValue** du descripteur de configuration voulu dans l'octet de poids faible de **wValue** pour sélectionner quelle configuration valider.

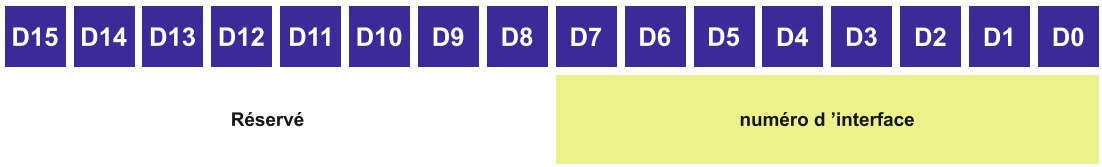
2.17.2. Les requêtes d'interface standard :

La spécification actuelle définit 5 requêtes d'interface standard qui sont détaillées dans **tableau 2.14.** Il est intéressant de noter que, seules 2 requêtes donnent quelque chose de compréhensible.

bmRequestType	bRequest	wValue	wIndex	wLength	Données
1000 0000b	GET_STATUS (0x00)	Zéro	Interface	Deux	Etat de l'appareil
0000 0001b	CLEAR_FEATURE (0x01)	Sélecteur de fonction	Interface	Zéro	Aucune
0000 0001b	SET_FEATURE (0x03)	Sélecteur de fonction	Interface	Zéro	Aucune
1000 0001b	GET_INTERFACE (0x0A)	Zéro	Interface	Un	Interface de remplacement
0000 0001b	SET_INTERFACE (0x11)	Positionnement alternatif	Interface	Zéro	Aucune

Tableau 2.14. *Requêtes d'interfaces standards*

Le champ **wIndex** est normalement utilisé pour préciser l'interface de référence pour des requêtes liées à l'interface. Voir son format dans le schéma ci-dessous.



GetStatus est utilisé pour retourner l'état de l'interface. Une telle requête à l'interface doit renvoyer 2 octets de valeur 0x00, 0x00. (Les 2 octets sont réservés pour une utilisation future).

Les requêtes **ClearFeature** et **SetFeature** peuvent être utilisées pour positionner des fonctions booléennes. Quand le destinataire désigné est l'interface, la spécification USB actuelle révision 2 précise qu'il n'y a pas de fonctions d'interface.

GetInterface et **SetInterface** règle le positionnement de l'interface de remplacement.

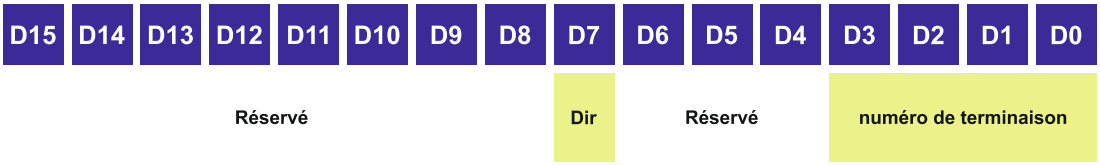
2.17.3. Les requêtes de terminaison standard :

Les requêtes de terminaisons standard sont au nombre de 4, listées dans le **tableau 2.15**.

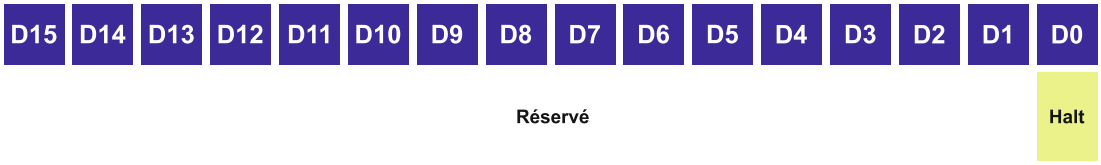
bmRequestType	bRequest	wValue	wIndex	wLength	Données
1000 0010b	GET_STATUS (0x00)	Zéro	Terminaison	Deux	Etat de l'appareil
0000 0010b	CLEAR_FEATURE (0x01)	Sélecteur de fonction	Terminaison	Zéro	Aucune
0000 0010b	SET_FEATURE (0x03)	Sélecteur de fonction	Terminaison	Zéro	Aucune
1000 0010b	SYNCH_FRAME (0x12)	Zéro	Terminaison	Deux	Numéro de Trame

Tableau 2.15. *Requêtes de terminaisons standards*

Le champ **wIndex** est normalement utilisé pour préciser la terminaison de référence et la direction pour les requêtes liées à la terminaison. Voir son format dans le schéma ci-dessous.



GetStatus renvoie 2 octets indiquant l'état (Arrêté/Bloqué) d'une terminaison. Le format des 2 octets renvoyés est illustré ci-dessous.



ClearFeature et **SetFeature** sont utilisés pour positionner les fonctions de la terminaison. Le standard défini actuellement un sélecteur de fonction de terminaison : ENDPOINT_HALT (0x00) qui permet à l'hôte de bloquer et d'effacer une terminaison.

Seules les terminaisons autres que la terminaison par défaut sont conseillées pour avoir cette fonctionnalité.

Une requête **SynchFrame** est utilisée pour rapporter une trame de synchronisation de terminaison.

2.18. L'ENUMERATION :

L'énumération est la manière de déterminer l'appareil qui vient juste d'être branché au bus et les paramètres dont il a besoin, comme la consommation électrique, le nombre et le type de terminaison, la classe du produit, etc.… L'hôte attribuera donc à l'appareil une adresse et validera une configuration lui permettant de transférer des données sur le bus. Toutefois lorsque l'on écrit un microprogramme USB pour la première fois, il est plus pratique de connaître la manière dont l'hôte répond pendant l'énumération, plutôt que le processus d'énumération général détaillé dans la spécification.

Une énumération sous Windows ordinaire implique les étapes suivantes :

- L'hôte ou Hub détecte la connexion d'un nouvel appareil via les résistances de rappel de l'appareil reliées sur les 2 fils de données. L'hôte attend au moins 100ms, le temps que la prise soit insérée complètement et que l'alimentation de l'appareil soit stabilisée.
- L'hôte émet un " reset " mettant l'appareil dans l'état par défaut. L'appareil peut maintenant répondre à l'adresse zéro par défaut.
- L'hôte (sous MS Windows) demande les 64 premiers octets du descripteur d'appareil.
- Après avoir reçu les 8 premiers octets du descripteur d'appareil, l'hôte émet immédiatement un autre reset sur le bus.
- L'hôte émet maintenant une commande **SetAdress**, mettant l'appareil dans l'état adressable.
- L'hôte demande la totalité des 18 octets du descripteur d'appareil.
- Puis il demande les 9 octets du descripteur de configuration pour déterminer la taille totale.
- L'hôte demande les 255 octets du descripteur de configuration.
- L'hôte demande l'un des descripteurs de chaînes s'ils étaient indiqués.

A la fin de l'étape 9, " Windows " demandera un driver (pilote logiciel) pour votre appareil. Il est alors courant de le voir redemander tous les descripteurs avant d'émettre une requête **SetConfiguration.**

L'étape 4 embarrasse souvent les gens qui écrivent des microprogrammes pour la première fois. L'hôte demande les 64 premiers octets du descripteur d'appareil, aussi lorsque l'hôte met à zéro votre appareil après avoir reçu les 8 premiers octets, il est tout à fait naturel de penser qu'il y a un problème soit au niveau du descripteur d'appareil soit dans la façon dont votre microprogramme manipule la requête. Cependant, comme vous le diront beaucoup de gens, si vous insistez sur la mise en œuvre de la commande **SetAdress**, elle sera récompensée par la demande suivante de 18 octets pleins du descripteur d'appareil.

Généralement quand il y a un problème avec le descripteur ou sur la façon dont il est envoyé, l'hôte tentera de le lire 3 fois avec de longues pauses entre les requêtes. Après la troisième tentative, l'hôte abandonne signalant une erreur au niveau de l'appareil.

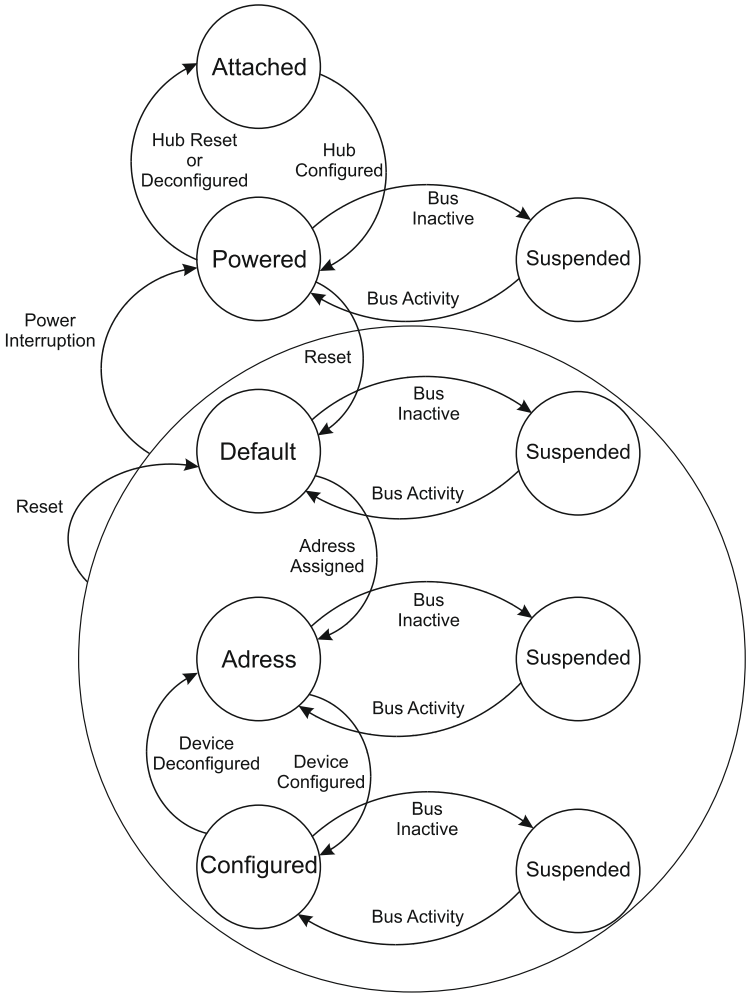


Figure 2.34. *Diagramme d'état d'un appareil USB*

CONCLUSION :

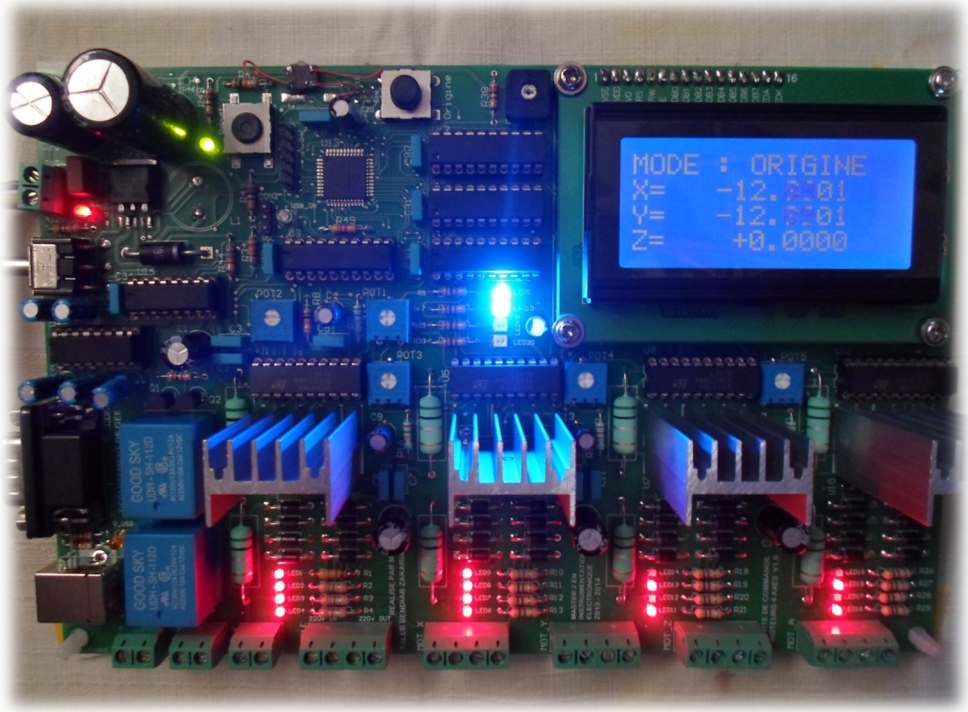
Dans ce chapitre, nous avons essayé de donner un aperçu général sur les spécifications du bus USB. Tout ce qui a été décrit dans ce chapitre a été pris de la spécification USB 2.0 qui est un document officiel décrivant toutes les caractéristiques du bus USB.

Nous avons utilisées ces informations dans notre projet pour concevoir la liaison USB entre la carte de commande et l'ordinateur, ce qui qui nous a permis d'établir une ligne de communication rapide et sure.

En définitif, le bus USB reste le moyen de communication le plus utilisé dans le monde et en particulier dans l'industrie ou on trouve des liaisons USB dans la plupart des machines et équipements industriels.

CHAPITRE III :

Théorie et Réalisation Pratique De La Carte D'interface



INTRODUCTION :

Dans ce dernier chapitre, nous allons essayer de mettre en évidence l'aspect pratique de ce que nous avons vu dans les chapitres précédents.

L'étude va être concentrée autour de la carte de commande. Celle-ci a pour but de recevoir des informations depuis un ordinateur via le bus USB, puis le processeur incorporé dans la carte doit analyser ces informations et les transformer en impulsions électriques afin de faire tourner des moteurs pas à pas.

Tous les composants utilisés dans cette carte ont une fonction bien précise dans son fonctionnement, on étudiera tout ceci le long de ce chapitre.

Enfin tout cet ensemble va fonctionner grâce à deux programmes informatiques, le premier est implanté dans le microcontrôleur qui gère la carte de commande, ce programme est écrit en langage machine. Le deuxième est un programme fonctionnant sur un ordinateur, il est écrit en langage évolué (dans notre cas c'est le VISUAL C++). Ce programme permet d'envoyer les différentes commande a la cartes ainsi que de visualiser l'avancement des opérations.

3.1. STRUCTURE GENERALE DE LA CARTE :

La structure générale de la carte de commande est représentée dans le schéma synoptique de la **figure 3.35.** On remarquera que le noyau de cette carte est le microcontrôleur de la famille microchip **PIC18F4550**.

Autour de ce microcontrôleur se trouvent les différents éléments qui lui permettent d'acquérir des données ou d'envoyer des commandes.

Nous avons l'unité d'entrée qui est composé d'un multiplexeur 8 lignes vers 1 ligne SN74HC151. Cette unité sert à identifier les différentes entrées qui sont les contacts limites pour les axes X, Y et Z ainsi que le bouton de la fonction ORIGINE.

L'unité d'affichage est composée d'un afficheur LCD de quatre lignes et 16 caractères par ligne. Elle sert à visualiser les différentes étapes d'exécution.

Nous avons quatre unités d'interfaces, chacune d'elles est associée à un moteur. Elles sont basées sur un SN74HC574 qui est un buffer trois états composé de 8 bascules D. le microcontrôleur utilise ces interfaces pour multiplexer les données envoyées aux différents moteurs.

Les unités de commandes sont aux nombre de quatre, elles sont basées sur le tandem L297 et L298 qui sont des circuits conçus pour la commande des différents types de moteurs pas à pas.

Et enfin nous avons les unités de communications, elles sont aux nombre de deux. Elles permettent au microcontrôleur de communiquer avec un ordinateur. On a l'interface RS232 qu'on n'utilisera pas dans notre projet pour le moment, en revanche on a opté pour l'interface USB qui est plus intéressante à mettre en œuvre.

Nous détaillerons tous ces points dans ce qui suit.

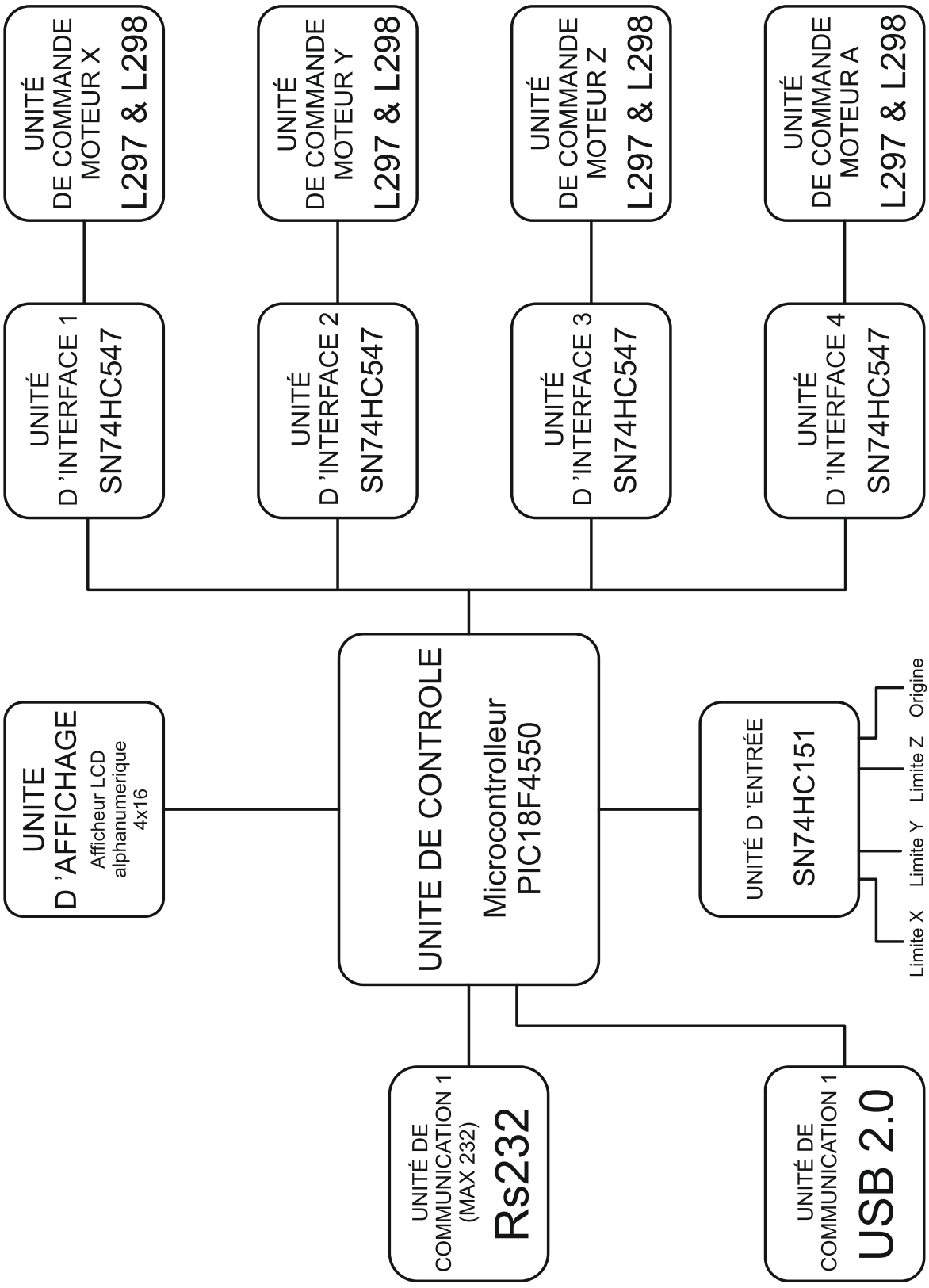


Figure 3.35. *Schéma synoptique de la carte de commande*

3.2. L'UNITE DE CONTROLE :

L'unité de contrôle est basée sur le microcontrôleur MICROCHIP PIC18F4550 qui un contrôleur huit bits dont l'avantage est qu'il est équipé d'une interface USB compatible avec le standard USB 2.0 (Annexe C).

La **figure 3.36** illustre le type de composant utilisé dans notre projet.

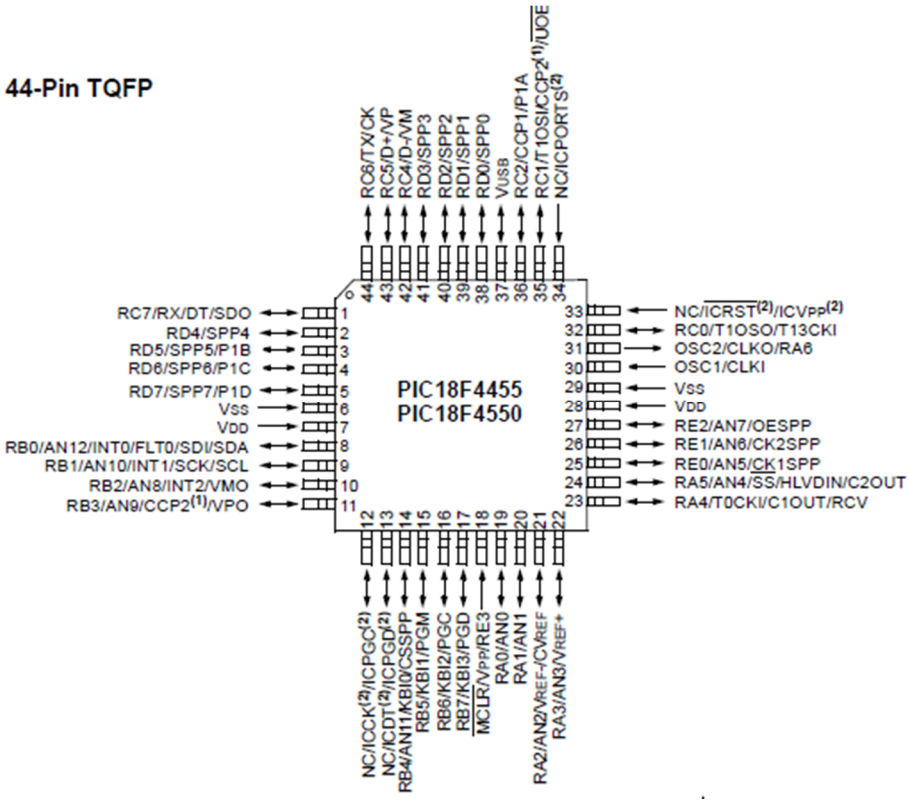


Figure 3.36. *Brochage du PIC18F4550-TQFP44*

Les pins de port **RD0..RD5** sont reliés aux circuits d'interface SN74HC574, ils permettent au microcontrôleur de commander les circuits de commande de moteur pas à pas L297 (Annexe C). La sélection des différents moteurs se fait grâce aux pins de port **RD6**, **RD7**, **RE0** et **RE1** qui sont reliés respectivement aux entrées horloge des circuits **U3**, **U6**, **U9** et **U12.**

La pin de port **RE2** est relié à la sortie sélection du circuit **U14** (SN74HC151), elle permet au microcontrôleur de lire l'état des différentes entrées (limite X, Limite Y, Limite Z, Origine). Pour ce faire le microcontrôleur utilise les pins de port **RA3**, **RA4** et **RA5** pour commander la sélection d'entrée du circuit **U14** (8 entrées possibles).

Les pins de port **RB0..RB7** représentent le bus de données reliant le microcontrôleur à l'afficheur LCD. Les pins de port **RA0**, **RA1** et **RA2** représentent respectivement les commandes **RS**, **R/W** et **E** relatives à l'afficheur LCD.

La pin de port **RC0** est reliée à la **led22**, elle représente l'état du port USB. Si le port USB est configuré alors la led22 s'allume et si le port USB est détaché alors la **led22** s'éteint (voir **figure 2.34**).

Les pins de port **RC4**, **RC5** sont reliées respectivement aux broches **D-** et **D+** du connecteur USB. La pin de port **RC1** est reliée à la broche **VCC** du connecteur USB, elle permet au microcontrôleur de savoir si le connecteur USB est relié à l'ordinateur afin d'enclencher le processus d'énumération (voir chapitre 2).

Les pins de port **RC6** et **RC7** sont reliées au circuit **U15** (MAX232), t cela afin d'utiliser l'interface RS232 du microcontrôleur.

La pin de port **RC1** est utilisée par le microcontrôleur pour commander deus relais. On utilisera les sorties des deux relais pour commander une mini perceuse ou autre mécanisme.

Les pins **OSC1** et **OSC2** sont reliées à un quartz pour former l'horloge du microcontrôleur. Dans notre cas on utilise un quartz de 20 Mhz, le microcontrôleur est configuré pour fonctionner a 48 Mhz et ceci afin d'obtenir la vitesse l'horloge permettant d'utiliser le bus USB en mode pleine vitesse (96 Mhz).

Enfin nous avons les pins de port ICPGC, ICPGD et ICRST qui sont utilisées pour programmer le microcontrôleur par la méthode ICSP (In Circuit Serial Programming). Cette méthode permet de programmer le microcontrôleur sans avoir à le retirer de la plaque de circuit imprimé.

Toutes ces informations sont illustrées dans le schéma électrique de la carte de commande en Annexe C.

3.3. L'UNITE DE COMMADE DE MOTEUR PAS A PAS :

Les circuits intégrés L297 ET L298 sont les composants les plus utilisées lorsqu'il s'agit de piloter des moteurs pas à pas. Ils permettent en effet de commander n'importe quel moteur, qu'il soit unipolaire ou bipolaire. Ils peuvent alimenter des organes consommant un courant compris entre quelques dizaines de milliampères et plusieurs ampères. Ce sont donc des composants standards par excellence car ils ont été conçus afin d'être utilisées ensemble, mais ils peuvent également être inclus dans un quelconque montage, et ce, séparément.

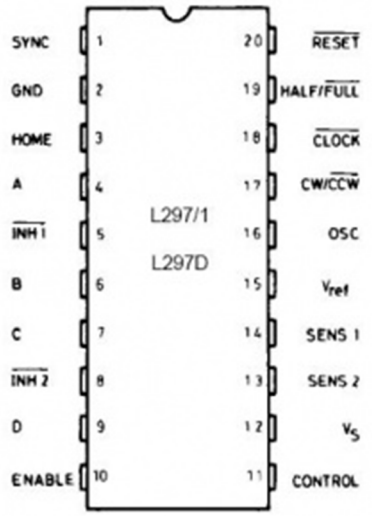
3.3.1. Présentation du circuit intégré L297 :

Le contrôleur de moteur pas à pas a été conçu, comme nous venons de le dire, afin d'être utilisé avec le circuit L298.

Ses principales caractéristiques sont :

- Commande en mode monophasé ou biphasé.
- Mode demi-pas ou pas entier.
- Sens de rotation horaire ou anti-horaire.
- Courant de charge programmable.
- Peu de composants externes nécessaires.
- Entrée de remise à zéro (RAZ).
- Sortie « HOME » (indique la position de départ).
- Entrée de validation du circuit.

La figure 3.37 donne le brochage et la structure interne du circuit L297. Le L297 est conçue pour fonctionner avec un « driver » en double pont, un réseau de quatre Darlington de puissance ou des composants discrets. Il lui suffit de recevoir des signaux d'horloge (avance des pas), de direction et de mode afin de piloter le moteur pas à pas.il génère ensuite, seul, la séquence de commande de l'étage de puissance.



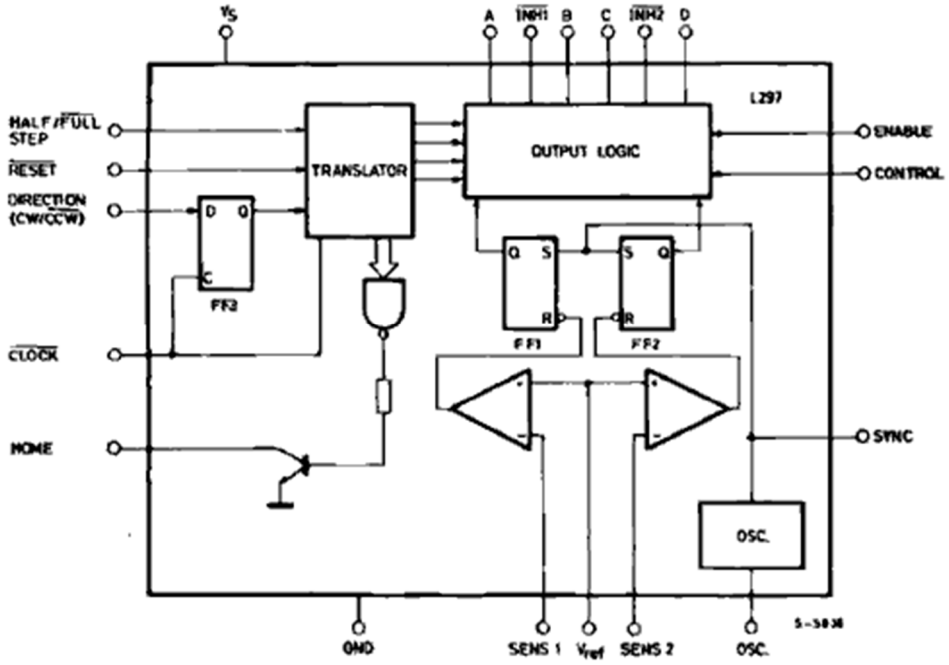


Figure 3.37. *Brochage et structure interne du circuit L297*

Comme on peut l'apercevoir sur la **figure 3.37** représentant sa constitution interne, il possède deux étages principaux : un translateur qui générales différentes séquences de commande et un doubleur hacheur PWM (Pulse Width Modulation) qui régule le courant traversant les bobinages du moteur.

Le translateur génère trois séquences différentes. Elles sont déterminées par le niveau logique appliqué sur l'entrée HALF/$\overline{\text{FULL}}$:

- La commande en mode monophasé : une seule phase alimentée.
- La commande en mode biphasé : deux phases alimentées.
- La commande en mode demi-pas : alternativement une phase puis deux phases alimentées.

Deux signaux d'inhibition (INH1 et INH2) sont également générés par le L297 dans les modes monophasés et demi-pas. Ces signaux qui sont directement appliqués aux entreés de validation du L298, permettent d'accélérer la décroissance du courant dans les bobinages du moteur lorsque ceux-ci ne sont plus alimentés. Lorsque le L297 est utilisé pour la commande d'un moteur unipolaire, les hacheurs agissent sur ces lignes.

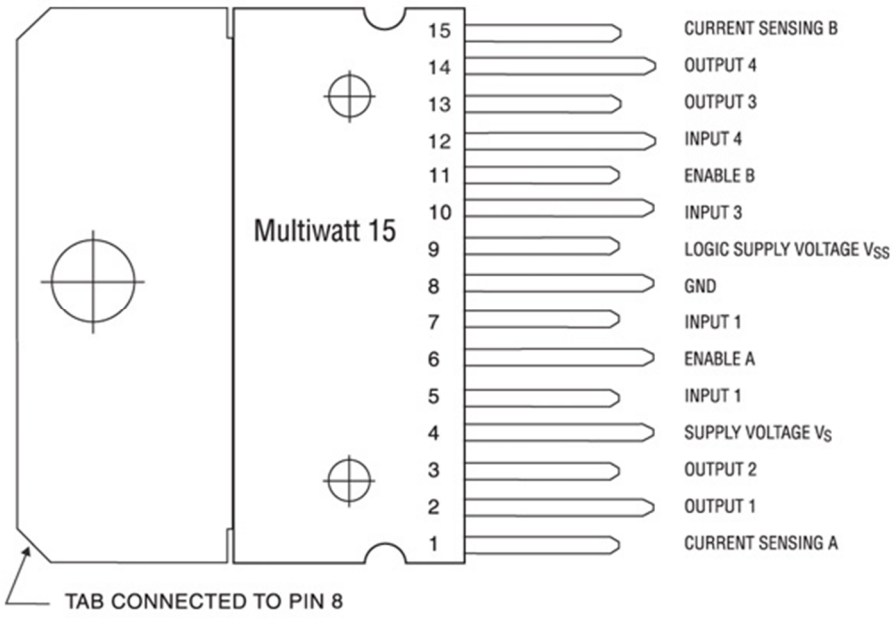
Une entrée nommée CONTROL détermine le moment où le hacheur devra agir sur les sorties A, B, C et D ou les entrées INH1 et INH2.

Un oscillateur interne commande les hacheurs. Lorsque le courant traversant l'un des bobinages du moteur atteint la tension programmée (valeur fixée par les résistances palpeuses connectées aux entrées $SENS_1$ et $SENS_2$), le comparateur correspondant interrompt l'alimentation du moteur jusqu'à la prochaine impulsion de l'oscillateur. Une tension de référence fixant la valeur du courant est dispensée aux deux comparateurs par un diviseur de tension connecté sur la broche **15** (V_{ref}) du L297. Ce diviseur peut être constitué d'une résistance fixe et d'une résistance ajustable afin de pouvoir régler la valeur du courant.

3.3.2. Présentation du circuit intégré L298 :

Le circuit intégré L298 est le circuit complémentaire du L297. Il permet, ainsi que nous l'avons déjà signalé, la simplification extrême de la construction d'une platine de commande d'un moteur pas à pas. Ce circuit n'est ni plus ni moins qu'un double pont de commande de puissance possédant des sorties de mesure de courant consommé par le moteur ainsi que des entrées de validation. Il ne nécessite donc que très peu de composants externes et sa mise en œuvre est très simple.

Il se présente sous la forme d'un boitier a 15 broches de type multiwatt dont le dessin est donné en **figure 3.38.**



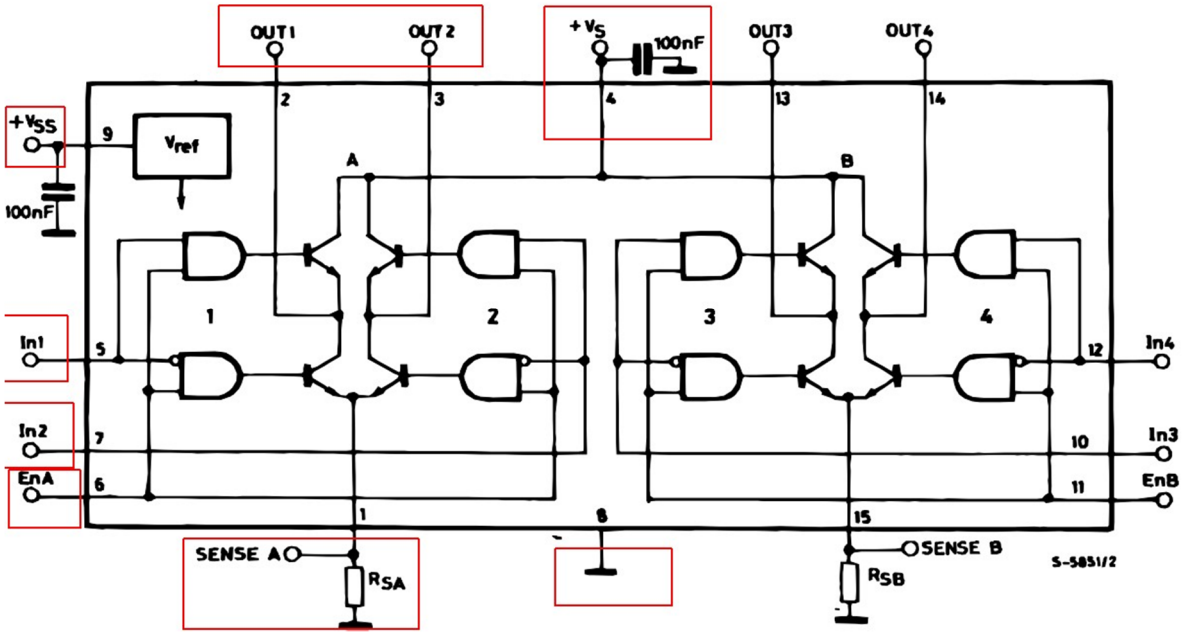


Figure 3.38. *Brochage et structure interne du circuit L298*

Le L298 permet l'utilisation d'une tension d'alimentation du moteur relativement élevée. Le courant qu'il peut débiter permet l'usage de moteurs pas à pas possédant des bobinages de valeur ohmique peu élevé. Il peut être utilisé pour la commande de moteurs pas à pas, de relais électromécaniques, de solénoïdes ou de moteurs a courant continu, et ce, en mode commun ou différentiel.

Le schéma interne du L298 est représenté en **figure 3.38.** On y aperçoit les deux ponts de quatre transistors de puissance commandés par les portes logiques, ainsi que la connexion des résistances de mesure du courant consommé par la charge reliées aux émetteurs de chaque paire de transistors.

La configuration finale utilisée pour les circuits L297 et L298 est représenté dans le schéma électrique illustré en Annexe C.

3.4. SOFTWARE ET PROGRAMMATION AVANCÉ :

Dans cette partie de ce chapitre, nous allons évoquer les différents types de programmation qu'on a utilisée dans notre étude. En premier lieu nous aborderons la partie concernant le microcontrôleur PIC18F4550 et ensuite nous verrons la partie concernant le logiciel de commande fonctionnant sur un ordinateur.

3.4.1. Programmation du PIC18F4550 :

La partie programme qui gère le PIC18F4550 a été faite en langage C. nous avons utilisé le logiciel de MICROCHIP MPLAB IDE v8.91 pour écrire notre programme ainsi que le compilateur MCC18 de MICROCHIP pour le compiler.

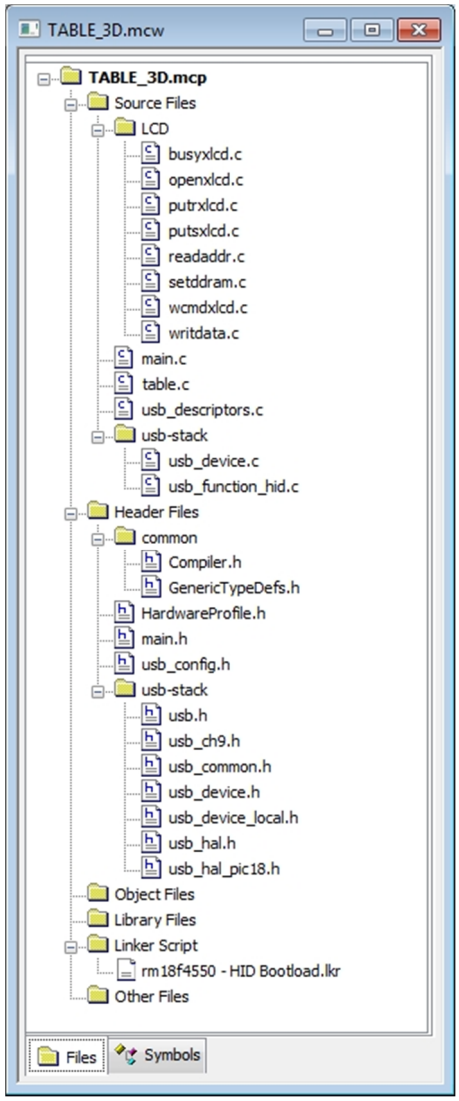
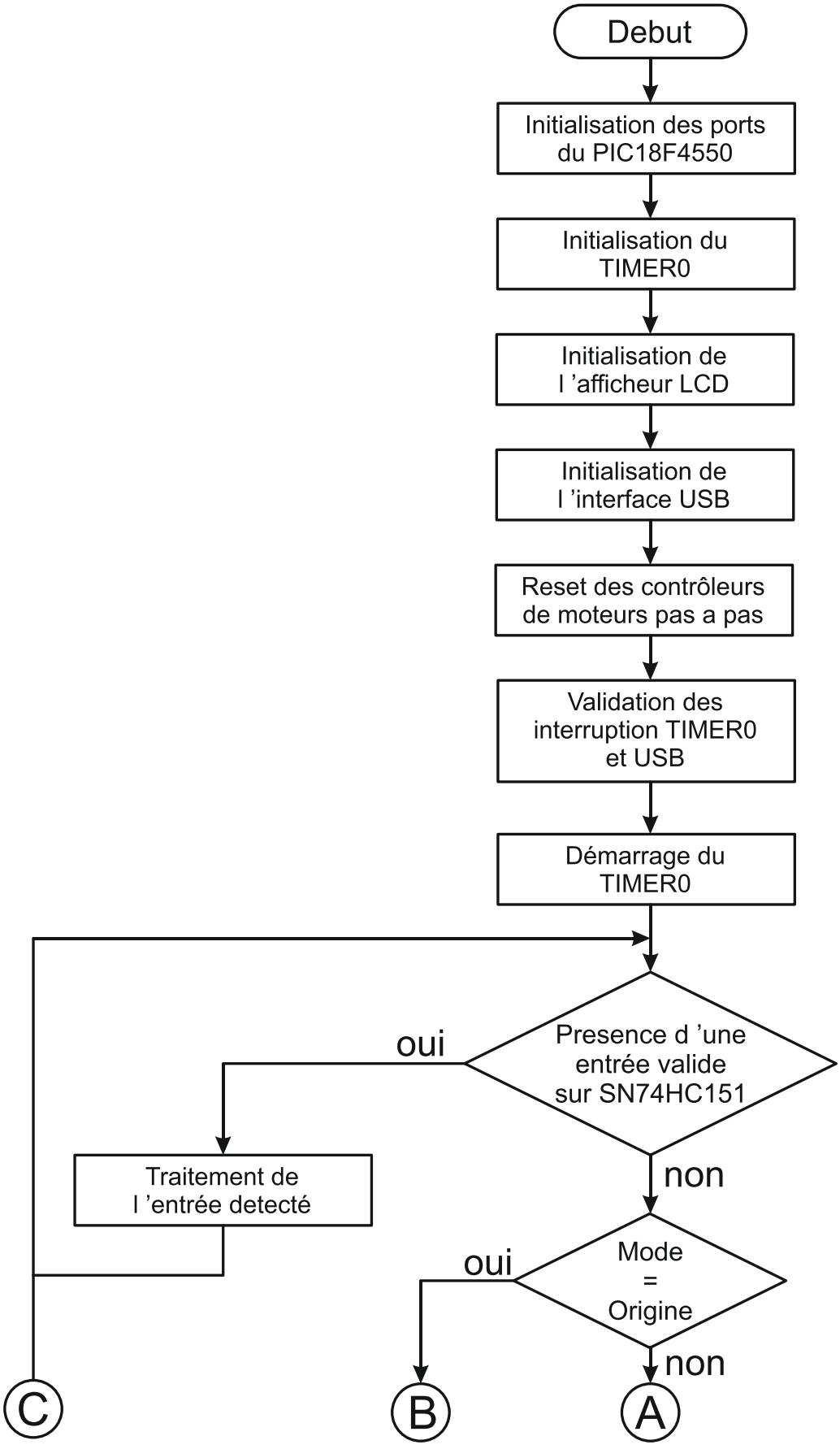


Figure 3.39. *Structure des fichiers constituant le programme pour le PIC18F4550*

Comme vous pouvez le constater sur la **figure 3.39**, nous avons utilisé une multitude de fichiers pour constituer le programme qui commande le PIC18F4550. Une partie de ces fichiers ont été ramené du composants MICROCHIP_SOLUTION_v2013-06-15 qui est un ensemble de procédures permettant d'intégrer la fonction USB dans notre programme.

Dans l'organigramme de la **Figure 3.40** nous allons essayer de mettre en évidence les différentes parties de notre programme.


Debut
Initialisation des ports du PIC18F4550
Initialisation du TIMER0
Initialisation de l 'afficheur LCD
Initialisation de l 'interface USB
Reset des contrôleurs de moteurs pas a pas
Validation des interruption TIMER0 et USB
Démarrage du TIMER0
Presence d 'une entrée valide sur SN74HC151
oui
Traitement de l 'entrée detecté
non
Mode = Origine
oui
non
C
B
A

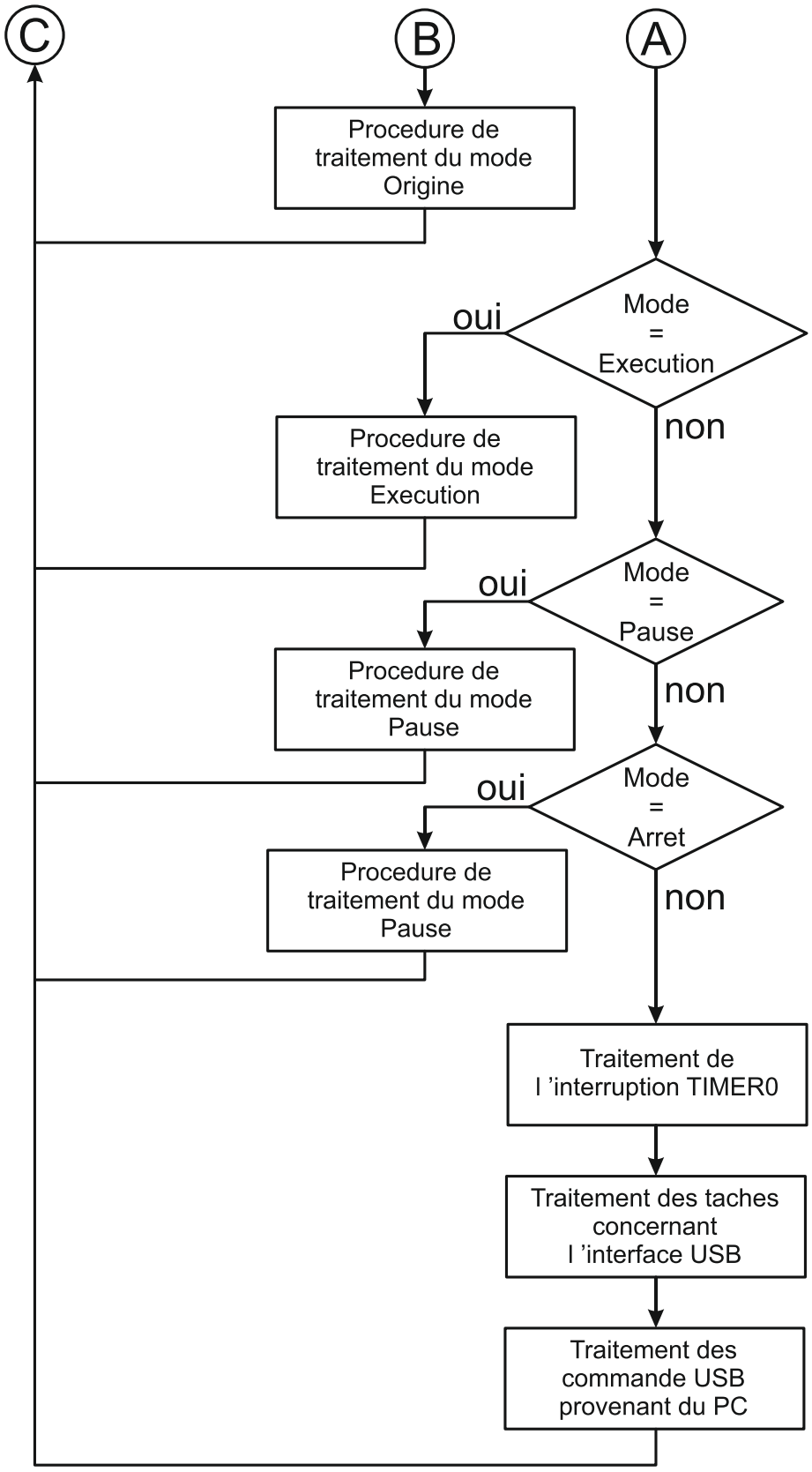


Figure 3.40. *Organigramme du programme pour le PIC18F4550*

La partie programmation du PIC18F4550 a été mise au point avec le logiciel de simulation PROTEUS-ISIS v7.10.

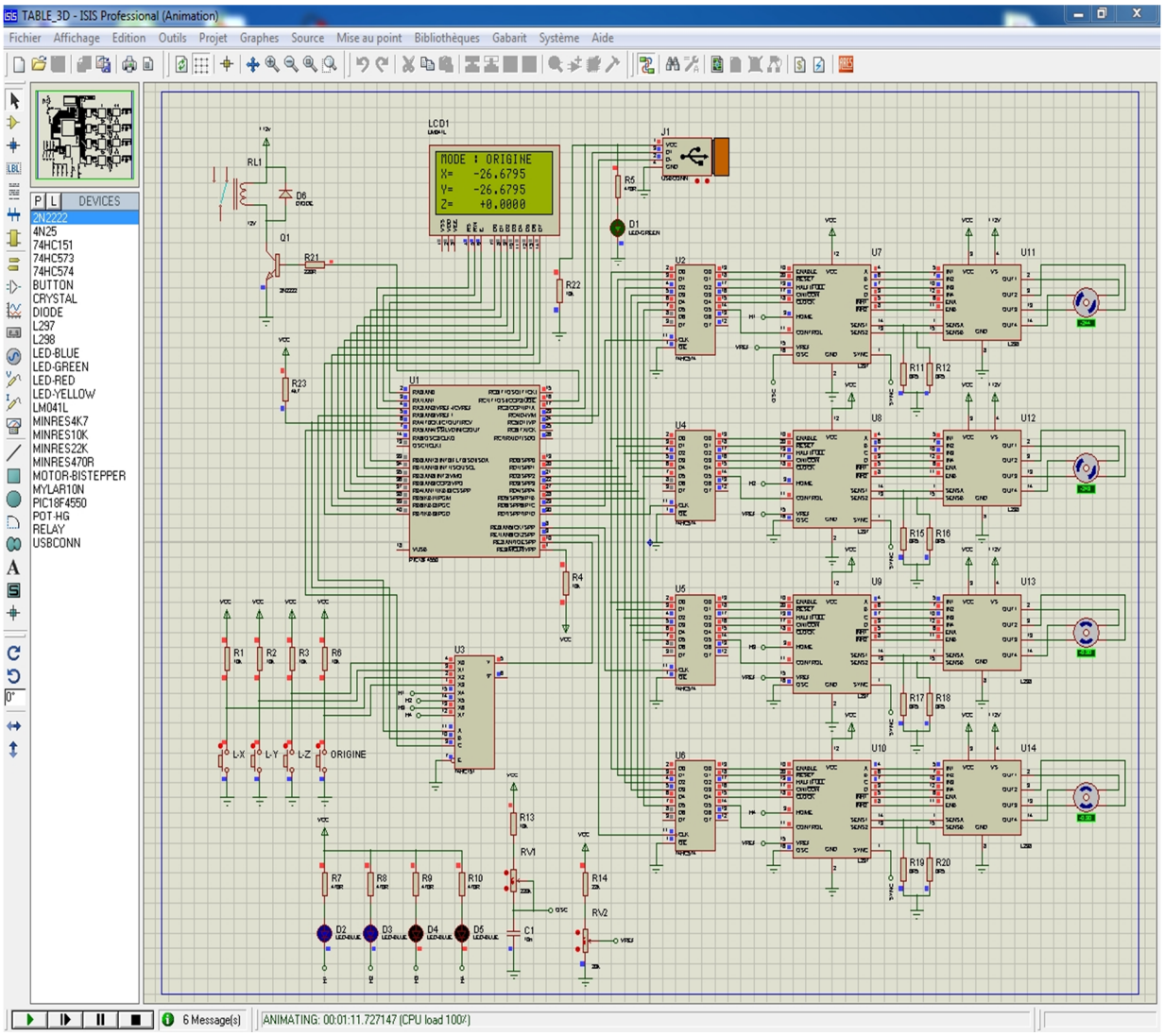


Figure 3.41. *Simulation du Programme sur PROTEUS-ISIS*

3.4.2. Programmation VISUAL C++ :

Le programme permettant d'envoyer les différentes commandes depuis l'ordinateur à la carte de commande a été réalisé grâce au logiciel VISUAL STUDIO 2008. L'interface utilisateur de ce programme est illustrée par **la figure 3.42.**

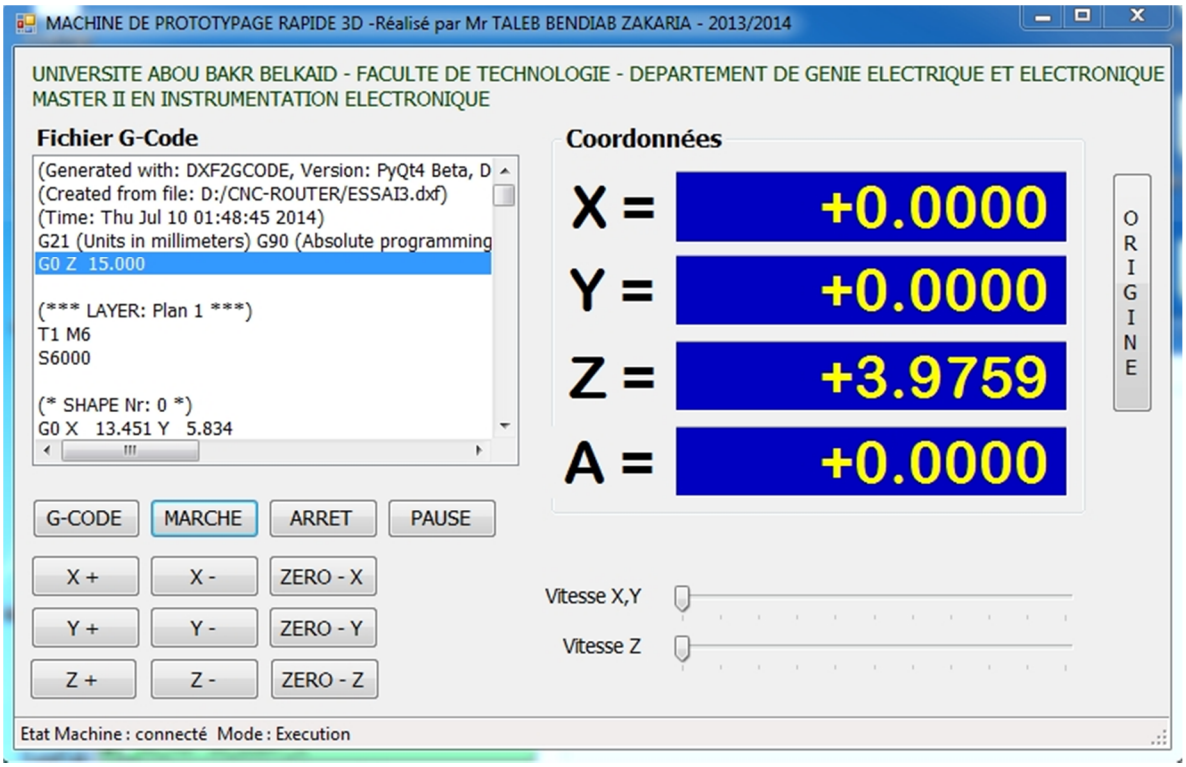


Figure 3.42. *Interface utilisateur pour le programme de commande*

L'interface utilisateur permet d'envoyer des commandes à la carte ainsi que l'affichage des différentes étapes de fonctionnement de la carte.

Nous avons d'abord les fonctions du mode Exécution qui sont les boutons G-CODE (qui permet de charger un fichier de format G-CODE et qui est un standard utilisé dans les machines à commandes numériques), MARCHE (qui sert à lancer l'exécution du fichier G-CODES), ARRET (qui sert à stopper le processus) ainsi que PAUSE (qui sert à stopper momentanément le processus).

Les boutons X+, X- , Y+, Y-, Z+ et Z+ servent à déplacer les différents moteurs dans différentes directions et ceci en étant en mode ARRET.

Les boutons ZERO-X, ZERO-Y et ZERO-Z servent à remettre à zéro les coordonnée de la carte.

Le bouton ORIGINE sert à enclencher le mode ORIGINE (l'arrêt de ce mode est obtenu soit par la touche arrêt soit par les switch limite-X et limite-Y).

Les indicateurs X, Y, Z et A affichent les coordonnées en temps réel des positions des différents moteurs.

Les boutons glissière Vitesse X, Y et Vitesse Z permettent de varier la vitesse de rotation des moteurs et ceci en mode EXECUTION.

La fenêtre intitulée « Fichier G-Code » affiche le programme en cours d'exécution.

Dans notre étude nous nous sommes limités à utiliser que quatre fonctions du format G-CODES qui en contient plusieurs et qui sont G0, G1, G2 et G3 :

- la fonction G0 permet un déplacement rapide aux coordonnées indiqué par X, Y ou Z.
- la fonction G1 permet d'exécuter un mouvement de coupe suivant une ligne droite.
- La fonction G2 permet d'exécuter un mouvement de coupe suivant une trajectoire circulaire dans le sens horaire.
- La fonction G3 permet d'exécuter un mouvement de coupe suivant une trajectoire circulaire dans le sens anti-horaire.

3.5. REALISATION DU CIRCUIT IMPRIMÉ :

Le circuit imprimé a été réalisé en double face trou métallisés grâce au logiciel ALTIUM DESIGNER V10, sont exécution a été faite à l'étranger dans l'entreprise EUROCIRCUITS S.R.L, qui est une société situé en Belgique et spécialisé dans la fabrication de circuits imprimées.

Les différents schémas du circuit imprimé sont illustrés dans l'Annexe A.

CONCLUSION :

Dans ce chapitre, nous avons abordé en détail les différentes phases d'élaboration de la carte de commande. Nous avons étudié la partie matérielle et ceci en expliquant les différentes parties qui composent la carte ainsi que leur fonctionnement respectif.

Ensuite nous avons abordé la partie logicielle qui est composée de deux parties, une partie servant à faire fonctionner la carte de commande et une autre servant à communiquer avec un ordinateur pour pouvoir envoyer des commande à la carte.

Pour le circuit imprimé et le schéma électrique, ils sont mis en évidence dans la partie Annexe de ce document.

CONCLUSION GENERALE :

Durant tout le temps qui nous a été alloué pour effectuer ce travail, nous avons essayé, du mieux qu'on pouvait, d'expliquer avec beaucoup de détail le fonctionnement d'une machine à commande numérique avec tout ce que ce nom englobe comme étapes à étudier.

Nous avons d'abord entamé la partie mouvement qui est basée sur les moteurs pas à pas que nous avons étudié en détail dans le chapitre 1. Cette partie est essentielle au fonctionnement des machines à commande numériques.

Ensuite nous avons entamé une étude sur le processus de communication entre la carte de commande et l'ordinateur. Ceci a été fait en détail dans le chapitre 2. Aucune machine à commande numérique ne peut fonctionner sans un système qui lui envoie les données à transformer en mouvements. C'est pour cela qu'on a opté pour le protocole USB, qui est un mode de communication moderne et rapide.

Enfin, dans le dernier chapitre nous avons essayé de rassembler ce qui a été étudié dans les chapitres précédent. Nous avons vu les différents type de programmation effectués au niveau du microcontrôleur ainsi qu'au niveau de l'ordinateur et ceci pour former un ensemble homogène qui permettra le bon fonctionnement de notre machine.

Pour conclure, nous souhaiterions qu'il y ' ait une continuation pour cette étude par d'autres étudiants afin de concevoir d'autre utilisations possibles pour cette carte.

Bibliographie

[1] **Ivor Horton.** (2008). Visual C++ 2008. *Wiley Publishing, Inc*.

[2] **John Paul Mueller.** (2002). Visual C++ .NET Developer's Guide. *McGraw-Hill/Osborne.*

[3] **Patrice Oguic.** (2004). Moteurs Pas à Pas et PC. *Editions Technique et Scientifiques Françaises (Dunod),* pp. 3 – 88.

[4] **Gerard Yvraut.** (1999). Les Moteurs Pas à Pas. *Séminaire Bellegarde Novembre 1999.*

[5] **Lilia El Amraoui.** (2002). Conception Electromécanique D'une Gamme D'actionneurs Linéaires Tubulaires a Reluctance Variable. *Université des Sciences et Technologies de lille*, pp. 6 – 13.

[6] **Claude Divoux.** (1999). Moteurs Pas à Pas.

[7] **G Berthome., F Mandin.** (2006). Synthèse Sur Les moteurs Pas à Pas. *Terminale STI Génie Electronique.*

[8] **Compaq Computer Corporation., Hewlett-Packard Company., Intel Corporation., Lucent Technologies Inc., Microsoft Corporation., NEC Corporation., Koninklijke Philips Electronics N.V.** (2000). Universal Serial Bus Specification Revision 2.0.

[9] **www.usb.org.** (2001). Device Class Definitionfor Human Interface Devices (HID). *USB Implementers' Forum.*

[10] **http://fr.wikipedia.org/wiki/Machineoutil_%C3%A0_commande_num%C3%A9rique.** (2014). Machine-outil à Commande Numérique.

[11] **http://fr.wikipedia.org/wiki/Universal_Serial_Bus.** (2014). Universal Serial Bus.

[12] **http://fr.wikipedia.org/wiki/Codage_NRZI.** (2014). Non Return to Zero Inverted.

[13] **MICROCHIP.** (2009). 28/40/44-Pin, High-Performance, Enhanced Flash, USB Microcontrollers with nanoWatt Technology.

[14] **SGS-THOMSON Microelectronics.** (1991). L297 Stepper Motor Controllers.

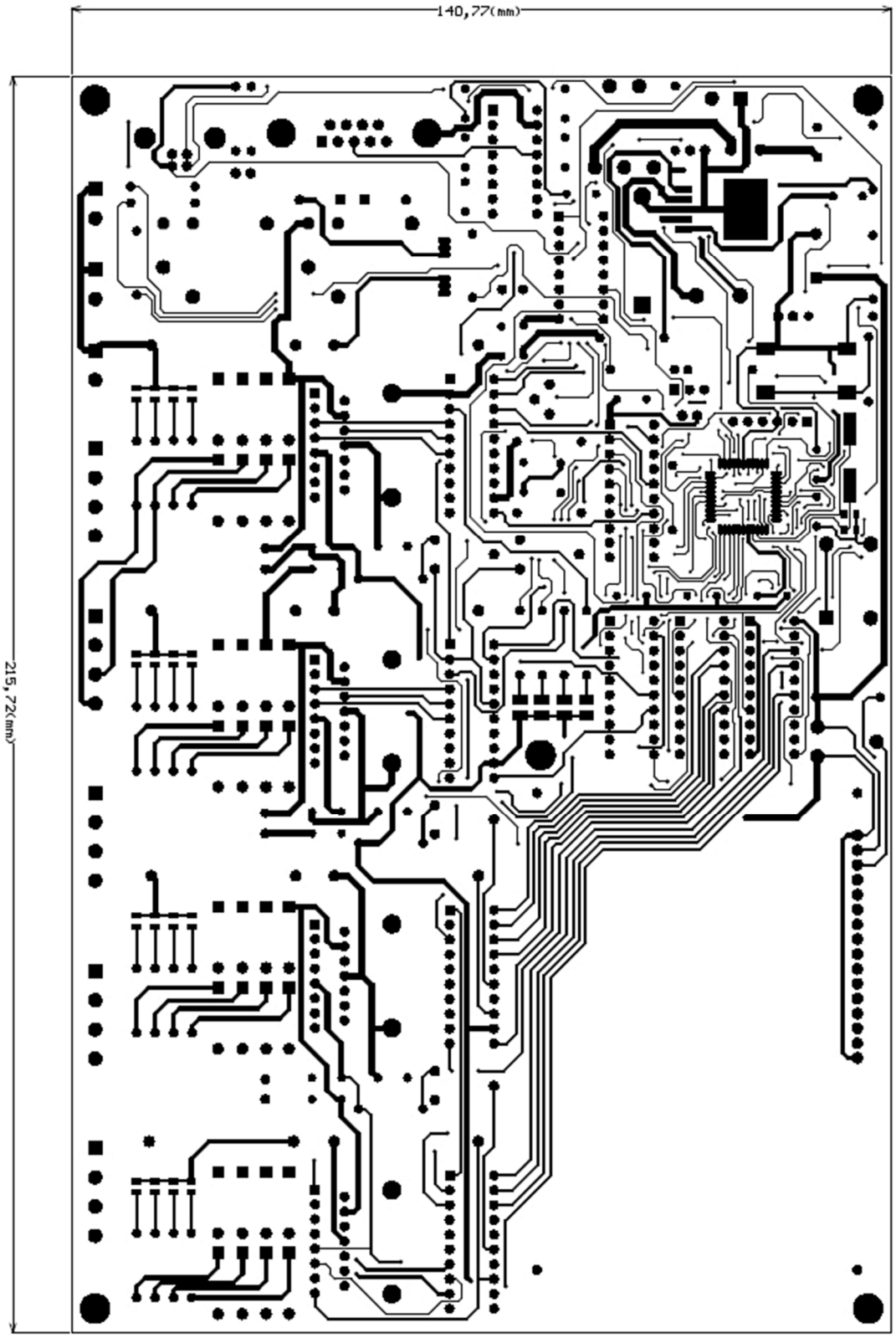
[15] **ST Micrelectronics.** (2000). L298 Dual Full-Bridge Driver.

[16] **NXP Semiconductors.** (2012). 74HC574 Octal D-type flip-flop; Positive Edge-Trigger; 3-state.

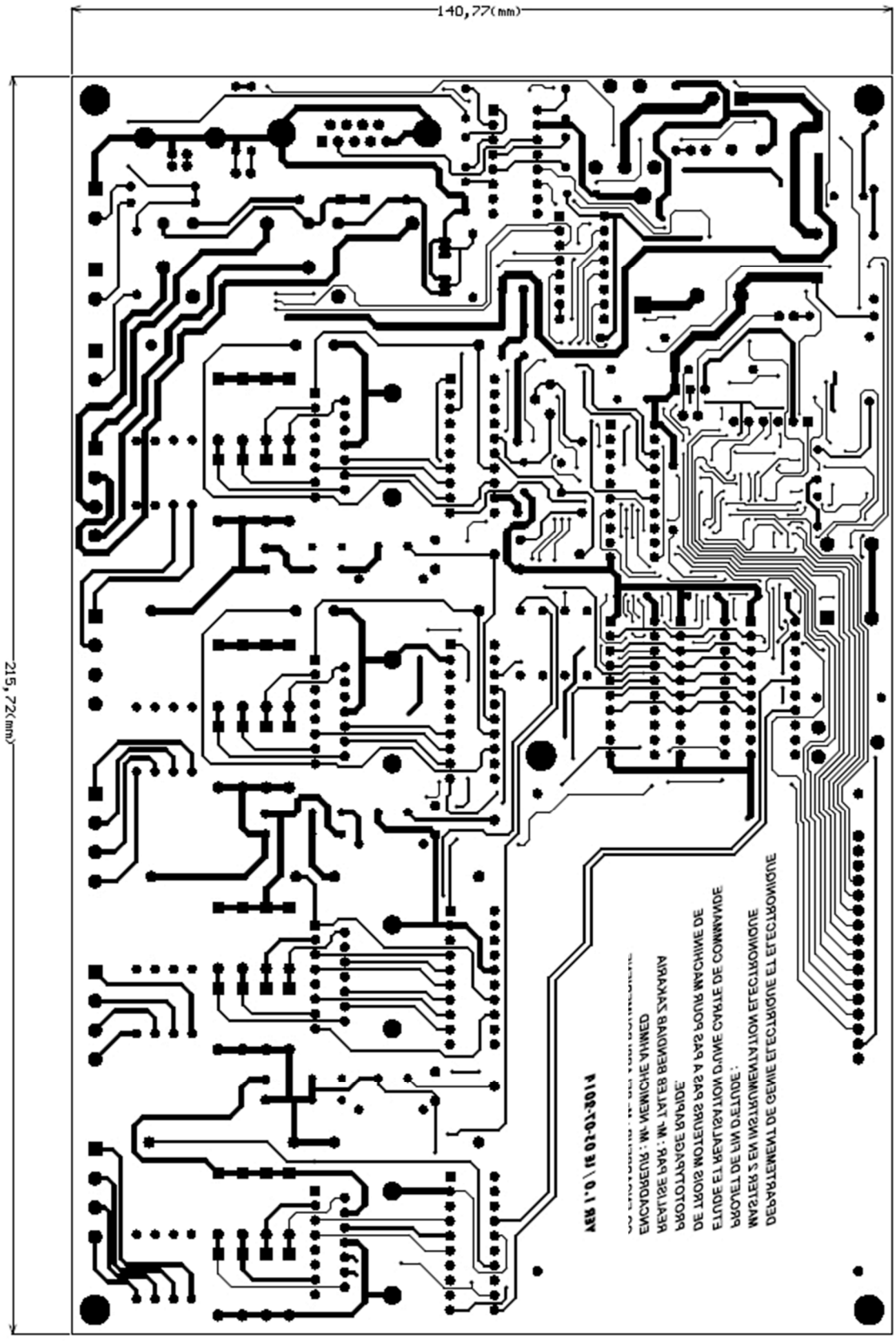
[17] **NXP Semiconductors.** (2013). 74HC151-Q100 8-Input Multiplexer.

[18] **MICROCHIP.** (2005). MPLAB C18 C Compiler User's Guide.

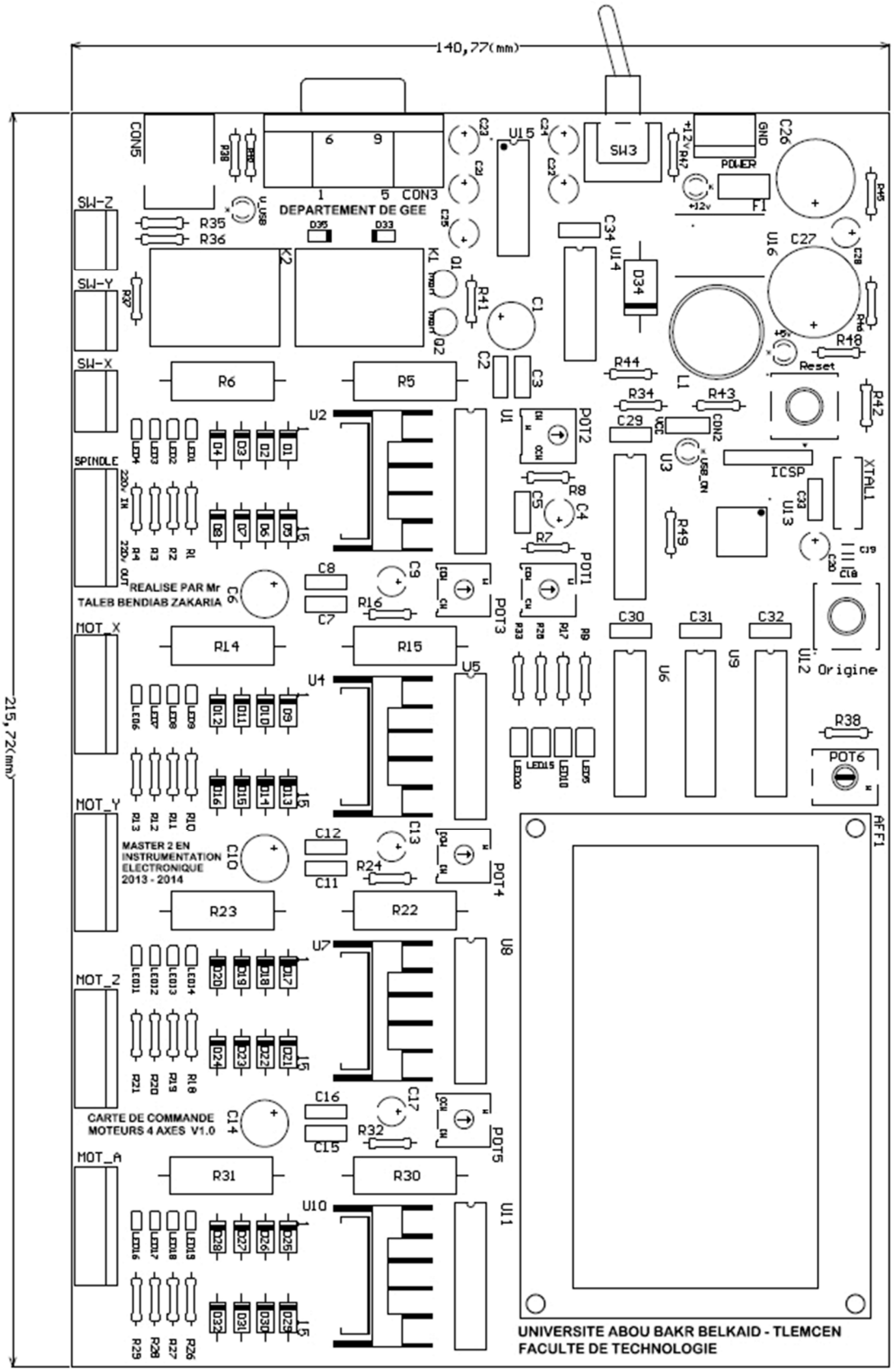
ANNEXES



Face composant du circuit imprimé



Face soudure du circuit imprimé



Face sérigraphie du circuit imprimé

Liste Des Composants

N°	Désignation	Reference	Quantité
01	U13	PIC18F4550TI/PT	1
02	U14	SN74HC151N	1
03	U3, U6, U9, U12	SN74HC574N	4
04	U2, U4, U7, U10	L298N	4
05	U1, U5, U8, U11	L297	4
06	U15	MAX232CPE	1
07	U16	MC33167D2T	1
08	XTAL1	Quartz 20Mhz	1
09	SW1, SW2	SW-DPST	2
10	SW-X, SW-Y, SW-Z, POWER+12V	Bornier 2 Plots	4
11	AFF1	LCD 4x16	1
12	MOT_X, MOT_Y, MOT_Z, MOT_A, SPINDLE	Bornier 4 Plots	5
13	SW3	SW-SPDT	1
14	C1, C6, C10, C14	470uF/25V	4
15	C2, C3, C7, C8, C11, C12, C15, C16, C29, C30, C31, C32, C33, C34	100nF	14
16	C4, C9, C13, C17, C21, C22, C23, C24, C25	1uF/25V	9
17	C18, C19	15pF	2
18	C26	1000uf/25V	1
19	C27	4700uF/25V	1
20	CON1	Connecteur 6 points	1
21	CON2	Connecteur 3 points	1
22	CON3	Connecteur SUB-D 9	1
23	CON5	Connecteur USB Type B	1
24	D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15, D16, D17, D18, D19, D20, D21, D22, D23, D24, D25, D26, D27, D28, D29, D30, D31, D32	1N4937	32
25	D33, D35	1N4148	2
26	D34	SB340	1
27	F1	Fusible 60V/1,5A	1
28	K1, K2	Relais - SPDT	2

29	L1	Inductance 180uH	1
30	LED1, LED2, LED3, LED4, LED6, LED7, LED8, LED9, LED11, LED12, LED13, LED14, LED16, LED17, LED18, LED19	Led Rouge SMD	16
31	LED5, LED10, LED15, LED20	Led Bleu SMD	4
32	LED21	Led 3mm Orange	1
33	LED22	Led 3mm Bleu	1
34	LED23	Led 3mm Rouge	1
35	LED24	Led 3mm Vert	1
36	POT1, POT3, POT4, POT5	Potentiomètre 20K	4
37	POT2	Potentiomètre 220K	1
38	POT6	Potentiomètre 10K	1
39	Q1, Q2	Transistor 2N2222	2
40	R1, R2, R3, R4, R10, R11, R12, R13, R18, R19, R20, R21, R26, R27, R28, R29, R47	Résistance 470	17
41	R5, R6, R14, R15, R22, R23, R30, R31	Résistance 0R5	8
42	R7, R16, R24, R32	Résistance 22K	4
43	R9, R17, R25, R33	Résistance 100	4
44	R8, R34, R35, R36, R37, R39, R42	Résistance 10K	7
45	R38, R40, R41, R43, R48	Résistance 220	5
46	R49	Résistance 4K7	1
47	R45	Résistance 68K	1
48	R44	Résistance 6,8K	1
49	C20	0,47uF/25V	1
50	C5	10nF	1
51	C28	0,1uF/25V	1