Soutenance ProjetTech

Réalisation d'un pilote Linux pour un NI 6003

BELLANGER Maxence
LARDY Timothée
THOMAS Maelwen

I. Contexte du projet

I.1 Description du NI 6003

I.1.1 Courte introduction

NI 6003[1]

Le NI 6003 est un CAN (Convertisseur Analogique Numérique), qui permet donc de récupérer le signal provenant d'un capteur, par exemple, afin de pouvoir le traiter sur un ordinateur pour l'analyser. C'est donc un objet nécessaire pour tout un tas d'expériences qui nécessite le traitement d'un signal.

I.1.2 Les limites du NI 6003 sur Linux

Il existe des logiciels pour communiquer avec le NI 6003, notamment LabVIEW ou encore un module Python appelé NI-DAQmx. En revanche, pour cet appareil, ces logiciels ne sont compatibles que sur Windows. En l'état, il n'y a aucun moyen d'utiliser le NI 6003 sur Linux, quelle que soit la distribution.

I.2 Objectifs du projet

I.2.1 Objectif principal

Notre tuteur utilisant Linux au quotidien et ayant besoin d'utiliser le NI 6003 dans ses expériences, il souhaiterait pouvoir l'utiliser sur Linux également. L'objectif de notre projet est donc de réaliser un pilote pour le NI 6003.

[1] Page officielle

I.2.2 Résultats attendus

Le résultat attendu est un pilote sous Python utilisant la bibliothèque PyUSB. Afin de pouvoir utiliser le NI 6003 avec les bibliothèques habituelles numpy, matplotlib, etc.

II. Bilan technique

II.1 Architecture d'un périphérique USB [2]

Afin de mener à bien ce projet, il était important de comprendre précisément l'architecture d'un périphérique USB. Un périphérique USB dispose de plusieurs entités: des configurations, des interfaces et des endpoints. Leurs rôles seront détaillés dans la suite.

II.1.1 Qu'est-ce qu'une configuration ?

Une configuration d'un périphérique USB est un ensemble d'interfaces et d'endpoints combiné avec des informations d'utilisation du périphérique, comme l'exigence en alimentation. Un périphérique USB peut avoir plusieurs configurations qui servent à différentes utilisations du périphérique. On peut par exemple imaginer qu'un périphérique ait un mode haute performance qui nécessite plus d'apport en alimentation ou d'appeler d'autres interfaces. Généralement, une configuration est accompagnée d'un descripteur qui est sous la forme :

Champ	Description
bLength	Longueur du descripteur
bDescriptorType	Type de descripteur (0x02 pour une configuration)
wTotalLength	Longueur totale des descripteurs de cette configuration
bNumInterfaces	Nombre d'interfaces dans la configuration
bmAttributes	Type d'alimentation (ex: autoalimenté)
bMaxPower	Consommation maximale en puissance

II.1.2 Qu'est-ce qu'une interface ?

Une interface est une subdivision d'un périphérique selon la fonction qui lui est dédiée. Par exemple, un casque USB peut avoir une interface pour gérer les haut-parleurs et une autre pour gérer le microphone. Pour faire cela, elle dispose d'un ensemble d'endpoints. Comme les configurations, elle dispose d'un descripteur qui garde en mémoire toutes les informations de l'interface :

Champ	Description
bInterfaceNumber	Numéro de l'interface
bAlternateSetting	Indique une version alternative des endpoints

[2] Source

BNumEndpoints	Nombre d'endpoints dans l'interface
bInterfaceClass	Classe (Type) de l'interface
bInterfaceSubClass	Sous-classe de l'interface
bInterfaceProtocol	Protocole utilisé
iInterface	Index d'une chaîne descriptive (pour décrire l'interface avec un texte)

II.1.3 Qu'est-ce qu'un endpoint ?

Un endpoint est l'entité utilisée pour communiquer avec d'autres objets connectés par USB. C'est un canal unidirectionnel de transfert de données entre un périphérique et l'hôte. Ils sont tous configurés selon un usage spécifique qui est, lui aussi, spécifié dans un descripteur :

Champ	Description
bEndpointAddress	Adresse unique de l'endpoint (elle inclut la direction (IN ou OUT))
bmAttributes	Type de Transfert
wMaxPacketSize	Taille maximale des paquets transmis ou reçus
bInterval	Intervalle de temps entre les transferts

II.2 Fonctionnement de la communication avec des périphériques USB

II.2.1 Qu'est-ce qu'un paquet ?[3]

Un paquet de données est un regroupement de données envoyé ou reçu par un périphérique USB. Les informations sont entre un périphérique et un hôte (généralement l'ordinateur). Chaque paquet a un nombre d'informations disposées au préalable pour le décrire, comme le type de transfert, l'adresse du périphérique, un code de vérification d'intégrité, sa taille et aussi les données à transmettre.

Dans une transaction (communication entre 2 entités) , il y a généralement 3 types de paquets :

II.2.1.1 Paquet de type Token

Ce paquet sert de demande au périphérique ou à l'hôte. Par exemple, il peut servir à demander à un périphérique de lire ce qu'il y a écrit sur un endpoint ou alors à dire à un périphérique de se mettre en « mode setup », car on s'apprête à lui envoyer des données à recevoir. Il est très important, car sans lui les périphériques ne savent pas comment interpréter un paquet qui arrive.

[3] Source

II.2.1.2 Paquet de type Data

Ce paquet est celui qui contient toute l'information et qui est envoyé. Un des moyens de faire cela est de lui demander de regarder l'information stockée sur un endpoint (qu'on a prévu au préalable avec un paquet token) et de l'envoyer à l'entité qui l'a demandée. C'est le paquet qui intéresse vraiment la machine qui l'a demandé, les autres types sont juste là pour assurer que tout se passe bien.

II.2.1.3 Paquet de type Handshake

Le paquet de type Handshake n'est que là pour attester du bon déroulement de la transaction. Par exemple, pour une transaction IN/OUT (de transfert de données), il atteste avoir reçu ou envoyé des données et, pour une attestation SETUP, le paquet accuse de réception et se prépare à recevoir un transfert spécifié par le paquet token.

II.2.2 Qu'est-ce qu'un protocole USB ?[4]

Les différents protocoles USB définissent toutes les règles d'un échange, c'est-à-dire la manière dont les paquets sont échangés et les types de transferts possibles. Ils ont été introduits pour avoir une très grande compatibilité entre les périphériques simples comme complexes. La phase qu'il est intéressant d'analyser est l'énumération, car c'est dans cette phase que les périphériques s'introduisent entre eux et c'est ici qu'il faut voir comment s'introduit le NI-USB 6003 pour pouvoir communiquer avec lui. De nos jours, nous en sommes au protocole USB 4.0, même si le plus utilisé reste l'USB 3.1 dû au chargeur USB-C qui est aujourd'hui un des plus utilisés. Ce protocole permet 4 principaux types de transfert.

II.2.3 Quels sont les différents types de transferts ?[5]

Ces types de transferts ont des caractéristiques spécifiques en fonction de la nature des données et de l'usage :

II.2.3.1 Transfert de contrôle

Il s'agit du type de transfert où l'on configure un périphérique, où l'on reçoit les commandes de contrôle ou où l'on effectue des échanges de configuration entre l'hôte et le périphérique. Lors de l'énumération (la phase où l'hôte et le périphérique s'introduisent), c'est ce genre de transfert que l'on voit, car ils s'envoient leur descripteur. Il y a souvent très peu de données échangées.

II.2.3.2 Transfert de masse

Comme son nom l'indique, ce transfert est fait pour envoyer des quantités volumineuses de données. Cela peut avoir lieu, par exemple, entre une clé USB et un ordinateur ou avec un disque dur externe, mais il peut parfois aussi avoir lieu avec une carte d'acquisition comme le NI-USB 6003.

II.2.3.3 Transfert isochrone

Ce transfert est réservé pour les débits continus. Cela peut inclure l'audio et la vidéo (en streaming). Cela assure une transmission régulière de données à un rythme précis. Cela est

---

[4] Source
[5] Source

notamment le cas quand on demande à l'application de la carte d'acquisition de construire un graphique des valeurs en temps réel à un débit précis.

II.2.3.4 Transfert par interruption

Ce type de transfert est utilisé pour des périphériques qui ne nécessitent pas beaucoup de transferts, mais de petits transferts rapides. Par exemple, pour une souris, la souris envoie un paquet dès qu'elle bouge et s'arrête dès qu'elle ne bouge pas : Quel serait l'intérêt d'envoyer des informations disant que la souris ne bouge pas sans cesse? C'est aussi le cas pour un clavier, on sait qu'il y a une touche enfoncée et on n'envoie un message que quand la touche n'est plus enfoncée.

II.3 Présentation des outils utilisés

II.3.1 Qu'est-ce que PyUSB[6] ?

PyUSB est une bibliothèque Python qui permet de communiquer simplement avec des périphériques USB. La communication étant très compliquée, la bibliothèque simplifie l'accès aux fonctionnalités USB comme l'envoi de paquets automatisés. Elle dispose de fonctions génériques qui permettent de communiquer avec les périphériques USB les plus minimalistes, mais pour les plus complexes, elle dispose de fonctions permettant d'entrer nous-mêmes les informations d'un paquet qu'on envoie. C'est cette fonctionnalité qui nous intéresse ici. Cette bibliothèque est compatible avec Linux et c'est grâce à cela que nous allons communiquer avec notre périphérique USB, le NI 6003.

II.3.2 Qu'est-ce que pySerial[7] ?

pySerial est une bibliothèque Python qui permet de communiquer simplement avec des périphériques utilisant le port serial qui était utilisé pour communiquer avec des périphériques sur de vieilles machines. Mais il existe aussi des périphériques USB utilisant le protocole serial, comme les microcontrôleurs Arduino ou Raspberry Pi. Nous avons découvert dans nos recherches que le NI 6003 n'utilise pas ce protocole.

II.3.3 Qu'est-ce que WireShark ?[8]

WireShark est un logiciel qui référence les protocoles réseau et permet donc d'inspecter les paquets réseau en temps réel. Pour notre projet, nous avons dû modifier WireShark pour qu'il puisse aussi interpréter des paquets issus du protocole USB. Pour cela, nous avons installé USBPcap[9], qui est un outil qui capture les paquets USB, et nous l'avons ajouté à WireShark. Ce logiciel ne propose pas de réelle interprétation des paquets, donc, à l'issue d'une capture, il faut essayer de décoder toutes les communications qui se sont passées.

---

[6] Documentation de PyUSB
[7] Documentation de pySerial
[8] Site Officiel WireShark
[9] Site Officiel USBPcap

II.3.3 Qu'est-ce que USBTrace ?[10]

USBTrace est un USB sniffer, c'est-à-dire une application permettant de voir directement tous les paquets échangés par un périphérique. Celui-ci est plus efficace que WireShark car nous n'avons pas besoin d'un outil pour adapter le trafic USB à un trafic réseau. De plus, nous avons un exemple d'application à notre disposition ce qui nous permet d'avoir une idée plus claire sur comment récupérer les bons paquets. L'exemple sur lequel nous nous appuyons sera plus détaillé dans la partie III.4.

[10] Site Officiel USBTrace

III. Nos différentes approches

III.1 Familiarisation avec PyUSB et pySerial

La première étape de notre projet était de comprendre les librairies PyUSB et PySerial. Cette étape était importante, car nous ne pouvions pas utiliser la librairie NI-DAQmx qui est nécessaire pour interagir avec le NI 6003 avec Python. Il nous fallait alors communiquer avec le NI 6003 comme tout autre périphérique USB avec une librairie compréhensible sur Python en utilisant PyUSB et PySerial. Nous y sommes alors allés crescendo en commençant d'abord avec une souris USB, ensuite une manette et enfin le NI 6003.

III.1.1 Souris/Manette

En utilisant la souris ou la manette, on arrive à décrire entièrement les entrées de ces périphériques et comment elles interagissent avec l'ordinateur. Nous utiliserons ici des images des tests effectués avec la manette, car la manette est plus complexe que la souris.

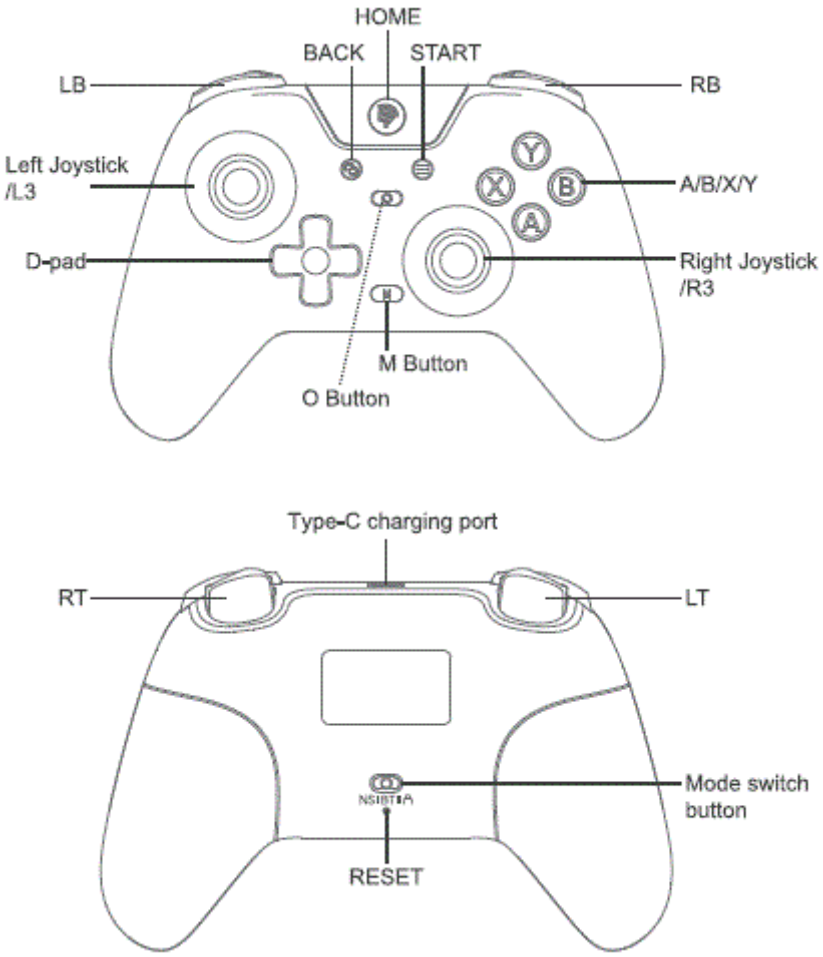


Schéma de la manette utilisée[11]

[11] Source de l'image

Une des plus grosses contraintes que nous avons pu rencontrer a été qu'on ne peut pas accéder aux informations d'un périphérique s'il est déjà utilisé par un autre processus ou un autre périphérique. Ainsi, il fallait au préalable détacher tous les processus qui étaient reliés à la manette. Cela implique que l'ordinateur ne peut plus utiliser le périphérique, mais il peut encore prendre les données renvoyées par celui-ci. Ensuite, pour des périphériques comme la manette et la souris, il suffit de trouver la configuration utilisée, l'interface où se situe l'endpoint qui nous intéresse et de directement lire sur celui-ci. Tout cela est détaillé dans le code *mouse.py*[12] sur le GitHub du projet (pour passer d'une souris à la manette, il suffit de changer le vendor_id et le product_id) .
Pour la manette, nous obtenons comme sortie un tableau de 20 valeurs comme ceci :

array('B', [0, 20, 0, 0, 0, 0, 122, 19, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Le B dans ce tableau signifie que les valeurs dans le tableau vont être des nombres de 0 à 255. Chaque entrée, comme la pression d'une touche, le mouvement d'un joystick, sera représentée dans ce tableau. Cela est aussi le cas si la manette est inactive. Quand la manette est inactive, elle renvoie ce tableau :

array('B', [0, 20, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Quand on appuie sur un des boutons, ce tableau devient :

array('B', [0, 20, 0, x, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])

Où x est 16 si on appuie sur le A, 32 si on appuie sur le B, 64 si on appuie sur le X et 128 si on appuie sur le Y. Si on appuie sur plusieurs boutons en même temps, leurs valeurs s'additionnent. Chaque bouton que l'on peut pousser a un fonctionnement similaire. Le plus intéressant qu'on ne trouve pas dans la plupart des souris sont les joysticks. Ils doivent être précis et leur niveau de pression est divisé en 256 valeurs, comme indiqué par le « B ».
À chaque joystick sont attribuées 4 positions dans ce tableau qui sont utilisées uniquement pour ces joysticks et qui définissent les 4 directions principales dans lesquelles ils peuvent aller : haut, bas, gauche, droite.

III.1.2 Microphone

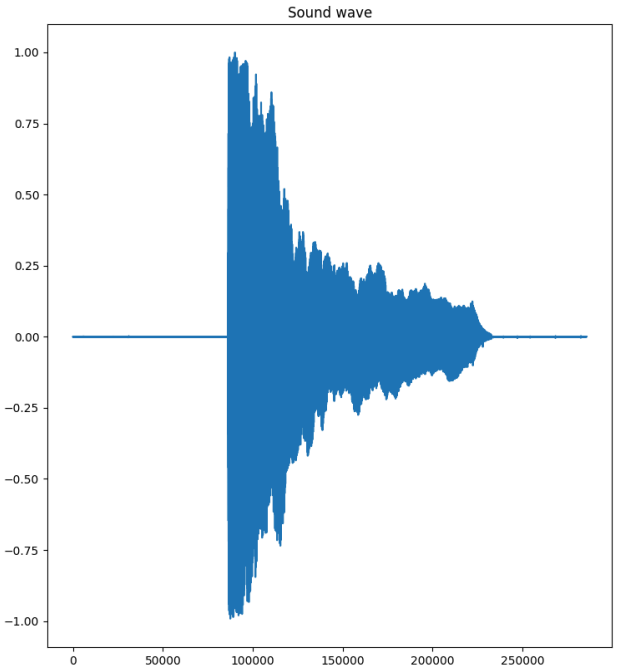
Le modèle de notre microphone est le PHILIPS SBC ME570. Ce modèle ne possède qu'une prise jack, donc nous avons utilisé un convertisseur USB/jack de la marque Logitech. Un problème récurrent s'est alors encore présenté avec ce périphérique. À chaque fois que l'on voulait accéder ou effectuer une opération sur ce périphérique, nous avons toujours obtenu une erreur : [Resource Busy], qui signifie que ce périphérique est déjà utilisé par d'autres processus. Pour le rendre accessible, il faut arrêter, directement sur le terminal Ubuntu, tous les processus reliés au microphone et le réintroduire au terminal pour qu'il soit visible pour le programme. Cela est bien plus difficile que pour la manette ou la souris. Ainsi, pour le pilote final de ce projet, nous utiliserons la manette qui est plus simple et qui utilise les mêmes processus pour communiquer

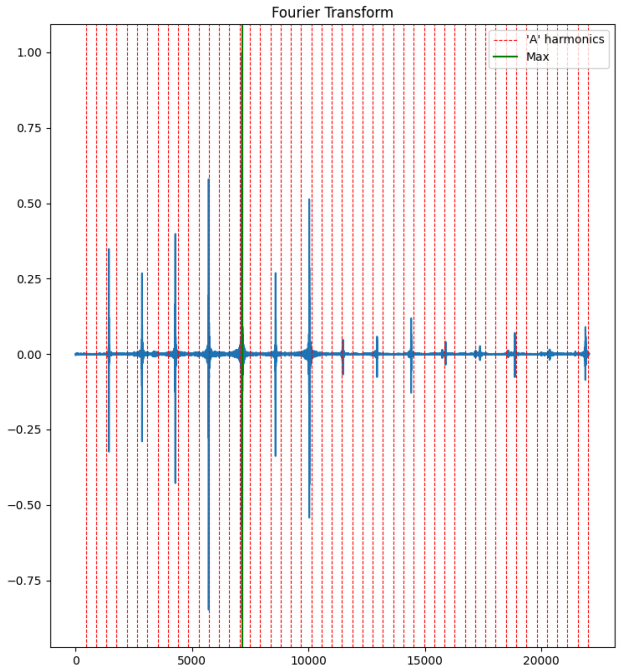
[12] Code mouse.py

avec l'ordinateur. Il est à noter que ce processus est dû à l'adaptateur qui complexifie la communication, car il ajoute une barrière supplémentaire à la collecte des données.
Une fois cette étape réalisée, on obtient des tableaux comme celui-ci :

[-2 0 1 5 5 6 4 4 4 2 4 5 5 5 5 3 1 0 -2 -1 0 0 0 -1 0 -1 -5 -5 -3 -3 -2 -2 1 1 0 1 -1 1 0 2 2 1 1 1 0 1 1 -1 -4 -3 -6 -7 -8 -8 -5 -3 -2 -1 0 6 5 4 5 3 4 1 0 2 4 2 1 0 0 2 1 -1 0 -2 -1 0 1 0 0 2 2 -1 0 1 0 0 4 4 2 1 -1 -1 -3 -5 -3 2 2 2 2 3 3 0 2 2 2 1 1 0 0 1 3 1 2 2 4 5 2 -1 -1 1 1 1 2 2 -1 -1 -2 -6 -6 -2 -1 -3 -3 -2 0 0 1 2 2 -1 0 1 -2 2 5 5 5 6 2 -2 -3 0 -2 -1 2 2 4 5 4 4 1 3 3 4 4 4 4 4 2 2 2 1 0 -2 -4 -7 -6 -10 -9 -6 -2 1 2 4 4 7 5 7 7 7 7 2 -1 -4 -6 -7 -6 -6 -4 -3 -3 -1 -2 1 0 2 3 0 3 -1 -3 -4 -4 -7 -8 -7 -6 -6 -4 -3 -5 -2 -3 -1 4 4 6 7 9 6 7 3 -2 -1 -2 -2 -2 -3 -6 -4 -2 -4 -2 0 1 1 2 2 1 1 2 0 3 4 3 2 2 1 1 1 -2 -3 -5 -2 -1 2 2 5 4 4 2 6 6 5 7 3 3 1 0 -2 -1 1 2 5 3 0 -3 0 1 1 2 -1 1 -1 -1 -1 -2 -1 -3 -6 -6 -5 -3 -3 0 -1 1 4 6 6 5 3 -1 -2 -5 -5 -6 -4 -6 -8 -8 -7 -6 -2 -2 -3 -7 -4 -4 -5 -1 -3 1 2 3 1 2 2 3 4 5 8 3 -1 -4 -8 -9 -10 -8 -8 -7 -5 -4 -2 -1 3 4 2 5 5 7 7 8 6 4 5 7 5 4 5 0 1 0 1 0 -4 -4 -1 0 0 3 3 5 5 5 5 -1 -6 -6 -6 -2 -3 -3 -2 -3 -2 -5 -6 -2 1 4 3 3 2 3 4 3 3 0 -2 -1 -2 -2 -1 1 3 4 6 8 10 10 7 5 5 3 2 -1 -2 -2 -3 -8 -7 -5 -4 -2 0 2 6 6 7 7 3 3 4 4 8 4 3 2 3 3 1 4 2 2 2 -1 -1 -2 -3 -3 -1 -2 -5 -5 -3 -2 0 1 2 5 4 3 2 3 0 2 1 2 1 -2 -3 -6 -9 -6 -3 0 0 -1 -1 -1 -1 -3 0 -1 -3 -3 -3 -1 -3 0 -1 -1 0 2 3 3 2 -1 -5 -6 -9 -8 -6 -4 -4 -4 -2 -3 -1 -1 1 3 3 4 3 3 4 1 0 -3 -3 -2 -4 -2 -1 1 3 5 4 4 2 1 0 1 2 3 1 0 2 1 4 3 3 3 0 -1 -1 1 4 6 7 8 6 6 5 6 4 2 0 0 2 -1 -1 -1 -1 -3 -3 -3 -2 -2 -3 -1 -3 -2 -2 -4 -1 -3 -2 -2 -2 -2 0 1 -1 -1 3 0 -1 -6 -5 -6 -8 -5 -8 -7 -5 -3 -1 0 -1 0 0 2 1 3 1 -1 -3 -6 -5 -5 -7 -6 -6 -4 -2 0 3 1 1 0 1 0 0 -1 0 -1 -1 7 6 4 5 3 0 -2 -6 -6 -8 -11 -8 -7 -6 -4 1 1 5 7 3 3 3 5 5 5 4 2 2 0 -4 -2 0 2 5 3 1 2 0 -3 -4 -7 -8 -8 -9 -8 -7 -7 -7 -5 -3 -2 0 2 4 3 2 2 0 0 -1 -2 -4 -2 -3 -3 -3 -6 -4 -2 -3 -1 1 3 4 5 3 2 3 4 5 4 4 4 2 2 0 0 3 4 6 3 3 6 5 3 3 1 3 4 4 6 5 2 2 -1 -2 -5 -5 -3 -1 1 2 3 4 7 7 8 8 2 1 2 3 5 7 7 3 1 1 1 1 -1 -3 -2 -1 3 6 6 7 5 4 3 5 7 3 -2 -4 -2 -4 -8 -8 -4 -4 -1 0 1 -2 -1 1 -3 -4 -3 -3 -2 0 -1 1 -1 0 -1 -2 -2 -4 -2 -1 -1 -1 -1 -1 0 2 4 7 6 8 9 6 5 4 3 3 2 1 0 -3 -3 -4 -3 -3 -3 -2 -4 -1 0 -4 -5 -5 -5 -4 -3 -3 -4 -3 -2 -5 -4 -3 -4 -1 0 -2 -1 -1 2 2 1 1 0 0 -2 -3 -4 -5 -7 -7 -7 -4 -2 -2 -1 -3 -3 -3 -1 4 2 3 -1 2 3 2 4 1 4 5 4 3 3 2 1 0 2 0 0 0 -1 -1 1 2 0 1 -2 -3 -3 -3 0 3 4 2 4 5 5 5 1 0 -1 -3 -3 -1 1 1 0 -2 0 0 1 3 0 1 1 3 1 0 -2 -3 0 1 2 3 4 8 6 4 1 2 4 3 3 1 0 1 0 0 -2 -1 0 1 1 2 0 3 4 4 7 6 3 2 4 4 6 3 5 2 1 2 1 0 0 -2 0 2 1 3 6 7 7 8 4 2 1 1 1 2 -1 4 4 6 6 4 4 1 2 0 2 3 4 4 3 2 0 -1]

Nous pouvons voir ici la grande complexité des données. Cependant, on peut directement le convertir en un fichier audio en le (en enlevant les séparateurs et en changeant les blocs en tableaux numpy) renommant le fichier en .wav. Cette simplicité des données brutes est sans doute due une nouvelle fois au convertisseur qui simplifie leur forme. Ainsi, comme pour la manette et la souris, une fois l'accès au périphérique effectué, les données sont simples et accessibles. Cela est très différent du NI 6003.





III.2 Premier test avec le NI 6003

Les premiers tests avec le NI 6003 ne sont pas aussi concluants. Nous avons bien évidemment commencé par essayer la librairie NI-DAQmx sur Python pour essayer de communiquer avec la carte d'acquisition. Cependant, les premières commandes marchent assez bien, mais quand vient le moment pour la carte d'acquisition de s'identifier, notre machine Linux ne reconnaît pas la carte. Dans la documentation NI, il est bien marqué que cet appareil n'est pas compatible avec Linux (en utilisant NI-DAQmx), donc ce n'était pas une surprise.
Ainsi, nous avons essayé d'utiliser PyUSB pour parvenir à récupérer directement les informations que nous voulions (la tension notamment) sur les endpoints. En effectuant cela, nous avons réussi à avoir accès à une information qui est:

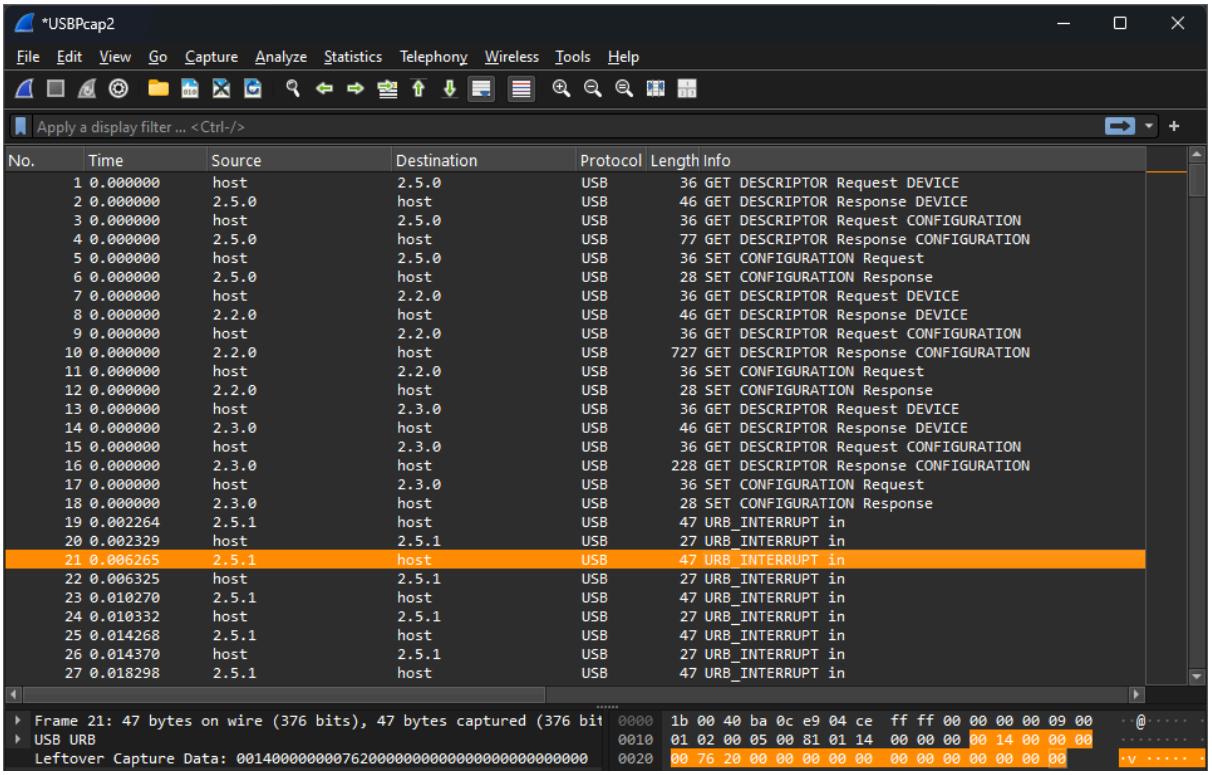
array['B']

Cela veut donc dire que nous sommes autorisés à lire sur cet endpoint, mais que l'information n'y figure pas. Cela s'est répété sur tous les autres endpoints (quand nous avons pu y accéder). Il s'agit du plus grand obstacle de ce projet. Il faut donc apprendre, comme dire au NI 6003 que l'on veut lire la tension sur un de ses endpoints. C'est pour cela qu'ensuite, nous avons essayé de lire et d'interpréter directement les paquets échangés avec l'ordinateur avec WireShark.

III.3 Reverse-engineering avec WireShark / USB Trace

III.3.1 Exemple sur une manette

Afin de mieux comprendre le principe de cette approche, prenons l'exemple de la manette dont nous connaissons le fonctionnement.



Nous pouvons voir qu'il y a d'abord la phase où le périphérique s'introduit à l'hôte et où l'hôte s'introduit dans le périphérique (entre les paquets 1 et 18). Ensuite, la manette envoie en permanence l'état dans lequel il se trouve. Il est à noter que le paquet que nous voyons (surligné en orange en bas) correspond au tableau que nous avions lors de nos tests avec PyUSB. En effet, si nous convertissons l'hexadécimal en base dix, nous trouvons:

[0, 20, 0, 0, 0, 0, 118, 32, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

C'est donc comme ça que nous souhaitons obtenir les informations du NI 6003.

III.3.2 Comment communiquer avec le NI 6003

À ce stade, nous savions qu'il fallait envoyer certains paquets au périphérique pour pouvoir espérer obtenir une réponse plus satisfaisante. Il fallait maintenant déterminer quels paquets envoyer. On a donc eu l'idée d'analyser le trafic USB entre le NI 6003 et les logiciels LabVIEW et NI-DAQmx sur un ordinateur sous Windows afin de comprendre quels paquets étaient échangés. L'objectif étant de reproduire ce comportement sous Windows.

Pour comprendre comment fonctionne le NI 6003, on a réalisé différents tests avec le NI pour pouvoir déterminer quelles informations se trouvent dans les paquets envoyés au NI.

On a utilisé l'outil de NI MAX pour récupérer un signal sur le NI 6003 et on a essayé les différents paramètres disponibles pour pouvoir identifier les paquets.

Test	Voie	Mode	Configuration d'entrée	Fréquence	Échantillons
1	A0	Fini	Différentielle	100	10
2	A0	Fini	Différentielle	1000	10
3	A0	Fini	Différentielle	10	10
4	A1	Fini	Différentielle	100	10
5	A2	Fini	Différentielle	100	10
6	A3	Fini	Différentielle	100	10
7	A0	Sur demande	Différentielle	100	10
8	A0	Continu	Différentielle	100	10
9	A0	Fini	Asymétrique	100	10
10	A0	Fini	Différentielle	100	5
11	A0	Fini	Différentielle	100	20

Différents tests réalisés[13]

Seq	Type	Time	Request	I/O	EP	DO	IRP	Status	Data Len
#0	START	0.000000	START OF LOG	-	-	-	-	-	-
#1	URB	1618.668317	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	12
#2	URB	1618.668515	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#3	URB	1618.668580	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	16
#4	URB	1618.668751	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#5	URB	1618.668967	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	24
#6	URB	1618.669207	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#7	URB	1618.669508	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	16
#8	URB	1618.669714	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#9	URB	1618.669763	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#10	URB	1618.669845	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#11	URB	1618.669870	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	12
#12	URB	1618.670018	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#13	URB	1618.670078	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	16
#14	URB	1618.670249	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#15	URB	1618.670308	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	12
#16	URB	1618.670463	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#17	URB	1618.670515	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#18	URB	1618.670647	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#19	URB	1618.670692	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#20	URB	1618.670745	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x6295CA60	STATUS_SUCCESS	0
#21	URB	1618.670767	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#22	URB	1618.771737	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x5865F2B0	STATUS_SUCCESS	0
#23	URB	1619.115019	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	12
#24	URB	1619.115261	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#25	URB	1619.115351	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	12
#26	URB	1619.115528	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0
#27	URB	1619.115767	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x42E8DA50	STATUS_SUCCESS	12
#28	URB	1619.115947	BULK_OR_INTERRUPT_TRANSFER	OUT	-	\Device\USBPDO-6	0x4D4A5A50	STATUS_SUCCESS	0

[13] Log des différents tests

Paquets échangés pendant le premier test

Request # 1 [OUT]

URB_FUNCTION_BULK_OR_INTERRUPT_TRANSFER

Urb Field	Value
Length	0x80
USBD Status	USBD_STATUS_SUCCESS (0x0)
PipeHandle	0xFFFFAB874D1C4380
TransferFlags	0x0 (USBD_TRANSFER_DIRECTION_OUT)
TransferBufferLength	0xC
TransferBuffer	0xFFFFD781410C1940
TransferBufferMDL	0x0
UrbLink	0x0

Parameter	Value
Data	00 01 00 0C 00 08 01 09 0D 30 00 00

Premier paquet reçu (log de USB Trace)

III.3.3 Analyse des paquets du NI 6003

Nous nous sommes inspirés d'un pilote déjà réalisé avec PyUSB pour le NI 6501[14] afin d'utiliser une partie du code déjà écrite pour essayer de lire des signaux sur le NI 6003. Et s'inspirer du format des paquets envoyés.

En analysant les requêtes échangées avec USB Trace, nous avons identifié des codes à utiliser pour communiquer avec le NI 6003.

Une requête USB destinée à la carte d'acquisition NI-6003 suit une structure précise :

\00\01\00\LengthTotal\00\LengthTotal-4\01\Command + Data

La majorité des requêtes envoyées ne contiennent pas de données et ne seront pas analysées.

III.3.3.1. Configuration initiale

La première requête configure le mode de fonctionnement et les paramètres initiaux de la carte :

Commande 09 : Mode différentiel

- Configuration "CONTINU" et "FINI":
  \0d\30\00\00

- Configuration "SUR DEMANDE" (ignorée dans ce contexte) :
  \0d\31\00\00

Commande 0C : Mode asymétrique

- Configuration "FINI":
  \0d\31\00\00

[14] Pilote du NI 6501

III.3.3.2. Définition du port d'écoute

La deuxième requête détermine le port d'écoute utilisé pour les mesures :

Commande 08

Structure :

\0d\30\00\00\00\03\8+Port\00

Exemples :

- Pour le port A0:

  0D 30 00 00 00 03 08 00

- Pour le port A1:

  0D 30 00 00 00 03 09 00

III.3.3.3. Paramètres de fréquence et nombre de mesures

La troisième requête configure :

1. La fréquence d'échantillonnage (en cycles de processeur à 80 MHz).

2. Le nombre de mesures à effectuer en mode **FINI**.

Fréquence

La fréquence est codée sur 4 octets. Par exemple :

- 100 Hz → 800 000 cycles :

  00 0C 35 00

- 1 000 Hz → 80 000 cycles :

  00 01 38 80

Nombre de mesures

Le nombre de mesures est aussi codé sur 4 octets. Par exemple :

- 10 mesures:

  00 00 00 0A

- 100 mesures :

  00 00 00 64

Commande 09 : Structure de la requête

Structure :

\0d\60\00\00\Freq\Freq\Freq\Freq\Sample\Sample\Sample\Sample\FF\00\Mode\00

- Mode :

  - 01 pour FINI

  - 02 pour CONTINU

Exemples :

- FINI, 100 Hz, 10 mesures :

  0D 60 00 00 00 0C 35 00 00 00 00 0A FF 00 01 00

- CONTINU, 100 Hz :

  0D 60 00 00 00 0C 35 00 00 00 00 00 FF 00 02 00

III.3.3.4. Stockage du nombre de mesures

La quatrième requête enregistre à nouveau le nombre de mesures. Cependant :

1. La valeur transmise est multipliée par 2.

2. La valeur est codée sur 4 octets.

3. En mode CONTINU, la valeur est fixée à :

   7F FF FF FF

Commande 08 : Structure de la requête

Structure :

\00\02\00\02\Sample\Sample\Sample\Sample

Exemples :

- CONTINU:
  00 02 00 02 7F FF FF FF

- 10 mesures :
  00 02 00 02 00 00 00 14

- 5 mesures:
  00 02 00 02 00 00 00 0A

III.3.3.5. Requêtes de récupération des données et de fin de communication

III.3.3.5.1 Requêtes de récupération des données

Ces requêtes permettent de récupérer les données acquises par la carte NI-6003. Leur structure varie en fonction du mode de fonctionnement de la carte mère. Cependant, elles ne semblent pas contenir d'informations significatives ou exploitables en elles-mêmes et sont donc peu intéressantes à analyser.

III.3.3.5.2 Requêtes de fin de communication

Ces requêtes mettent un terme aux communications avec la carte. Leur structure est identique dans tous les cas et, tout comme les requêtes de récupération des données, elles ne présentent pas d'informations pertinentes à examiner.

IV. Conclusions et apprentissages

IV.1. Qu’avons-nous appris avec ce projet ?

IV.1.1. Connaissances spécifiques

Nous avons appris comment fonctionne le protocole USB et à utiliser la bibliothèque PyUSB, pour interagir avec des périphériques utilisant ce protocole. Que ce soit des objets simples comme une souris ou une manette qui n’ont que peu d’entrées, ou un microphone avec des entrées plus compliquées à interpréter. Nous avons aussi appris à décortiquer la structure de périphériques USB afin de mieux communiquer avec eux. Notamment à quelle interface s’adresser quand on voulait effectuer une certaine requête.

Nous avons appris à utiliser des logiciels d’analyse de paquets comme Wireshark ou USBTrace. Ces logiciels nécessitent des connaissances poussées de l’architecture des protocoles USB, donc nous nous sommes concentrés sur les parties qui nous concernaient pour ce projet, mais ces logiciels permettent bien plus de choses.

IV.1.2. Gestion de projet

Nous avons appris à travailler sur un projet dont la solution n’est pas triviale et nous avons dû explorer de nombreuses pistes souvent infructueuses. Notamment réaliser un travail de spécification du sujet, pour pouvoir se placer dans des cas plus simples afin d’avancer pas à pas.

Nous avons dû expérimenter la démarche d’ingénieur et, bien que ce projet n'ait pas pu être mené à son terme, nous avons beaucoup appris sur la gestion de projet et l’organisation d’un projet technique difficile.