INGENIEUR POLYTECH LILLE IMA

Projets IMA4 SC 2018/2019

Manettes pour travaux pratiques

BOËNS QUENTIN

Rapport du projet

POLYTECH Lille
Secrétariat IMA - Muriel Hoogstoel
Boulevard Paul Langevin - Cité Scientifique
59655 VILLENEUVE D'ASCQ CEDEX
03-28-76-73-60
03-28-76-73-61

Encadrants école :
Xavier Redon
Alexandre Boé
Thomas Vantroys

Table des matières

I. Objectifs

Le but de ce projet est de concevoir et de réaliser des manettes à base de microcontrôleurs pour des travaux pratiques en GIS3 et IMA4. Il s'agit aussi d'écrire le code de ces manettes. Il faut à la fois programmer les microcontrôleurs des manettes et écrire les programmes PC pour gérer les manettes.

Les manettes sont constituées de :

- Manette 328p
    - Le microcontrôleur est une atmega328p
    - Un FT232BL, utilisé pour la communication USB
    - 10 LEDs
    - 5 boutons
    - 1 boutons reset
    - 2 vibreurs
- Manette 16u2
    - Le microcontrôleur est une atmega16u2
    - 10 LEDs
    - 5 boutons
    - 1 boutons reset
    - 2 vibreurs

Les fonctionnalités attendues sont décrites ci-dessous.

Chaque manette est composée de dix LEDS, affichant l'état de la manette. Leur comportement est dépendant de l'état de la manette. A moi de le définir.

Les deux manettes présentent 5 boutons, dont les états doivent être enregistrés afin de les envoyer au PC.

Les deux manettes possèdent deux vibreurs, qui sont activés suivant l'activité de la manette. J'ai le choix pour définir leur comportement.

Régulièrement, chaque manette envoie au PC une trame de données indiquant l'état des boutons.

Pour la manette 328p, un programme sur le PC doit afficher les messages reçus, à la demande de l'utilisateur.

Le PC peut envoyer un message demandant une nouvelle trame de données à la manette.

Ce projet est la continuation de deux projets réalisés par deux autres élèves l'année dernière. J'ai donc repris les travaux qu'ils avaient effectués, à savoir la conception du PCB des manettes.

II. Réalisation de la manette 328p

Cette manette comporte un microcontrôleur atmega328p.

1. PCB

Résultats de mon prédécesseur

Lorsque M. Redon m'a donné les résultats du projet de mon prédécesseur, il m'a expliqué que la manette réalisée n'était pas détectée par le PC.

Plusieurs modifications avaient été apportées au PCB afin de corriger le problème.

Après plusieurs essais, ils ont réussi à faire détecter la manette par le PC, mais un message d'erreur s'affichait. Ce message indiquait que la manette n'était pas reconnue par le PC.

Du fait de ces résultats, M. Redon a estimé que le problème devait se situer au niveau du FT232BL.

C'est donc avec ces informations que j'ai commencé à travailler sur le PCB.

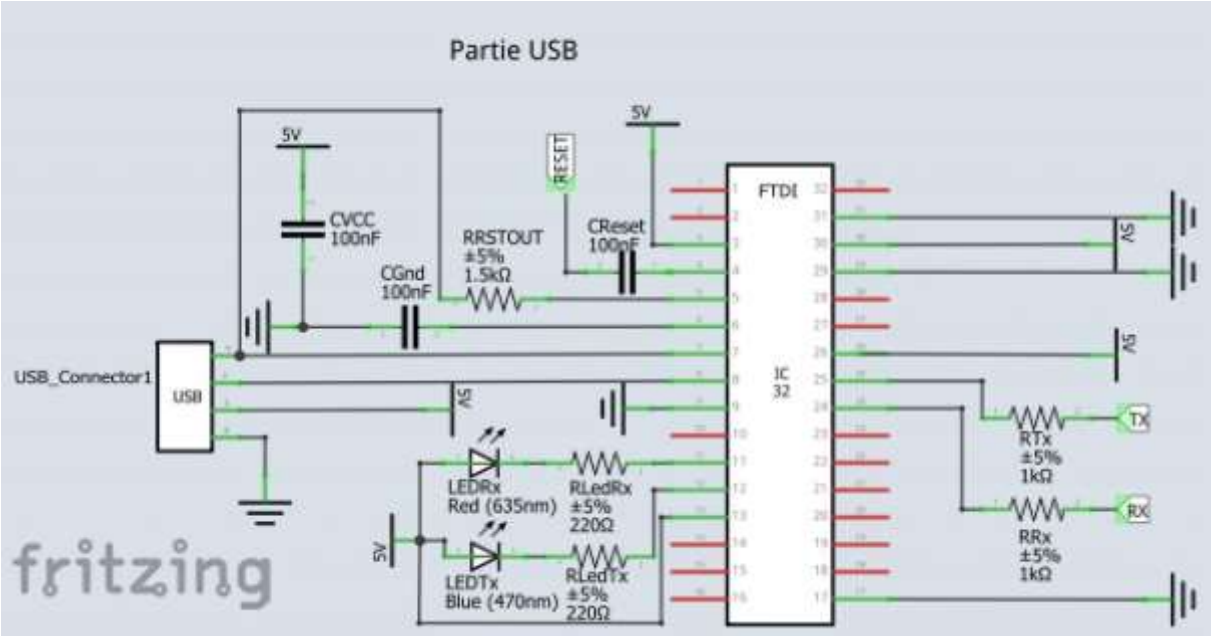
Corrections sur la carte

Mon premier objectif a été d'analyser le schéma électrique réalisé par mon prédécesseur.

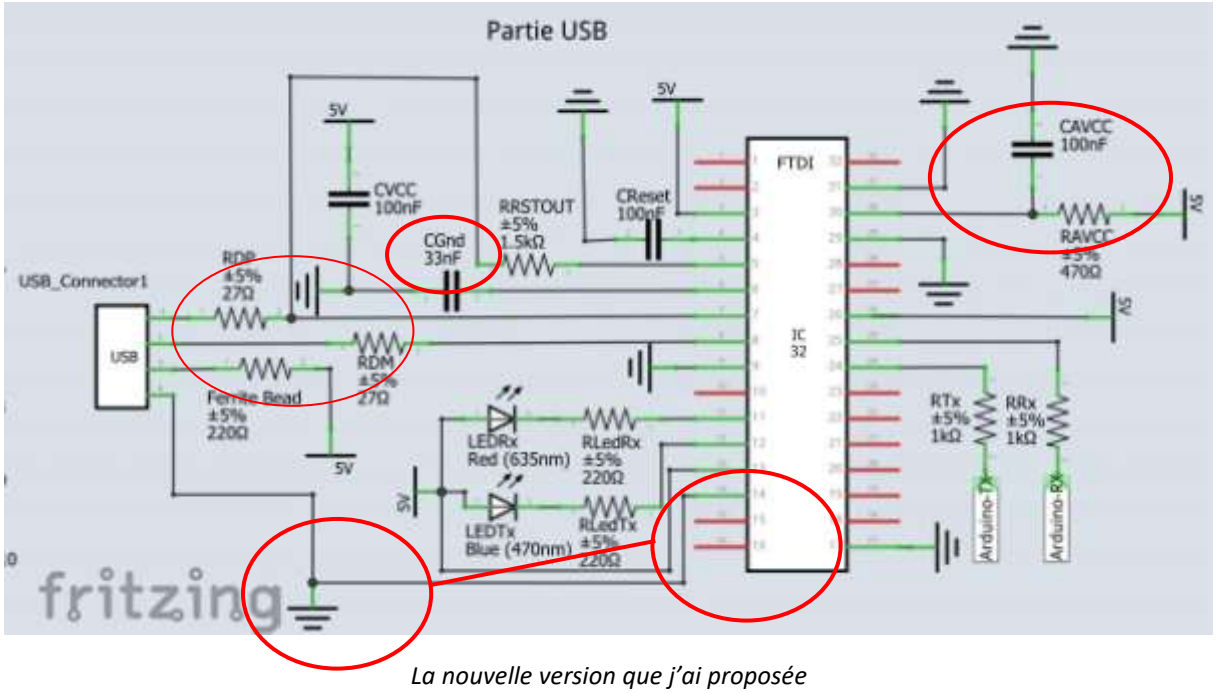
Je me suis concentré sur le FT232BL, comme me l'avait conseillé M. Redon.

En comparant le schéma électrique et la datasheet du composant, j'ai constaté l'absence de composants et de connexions sur certains pins.

Notamment au niveau des pins USB DM et USB DP. En effet, ces pins sont reliés à la prise USB, de façon à recevoir et transmettre des messages via le port USB. Il manquait notamment plusieurs résistances et certaines capacités.

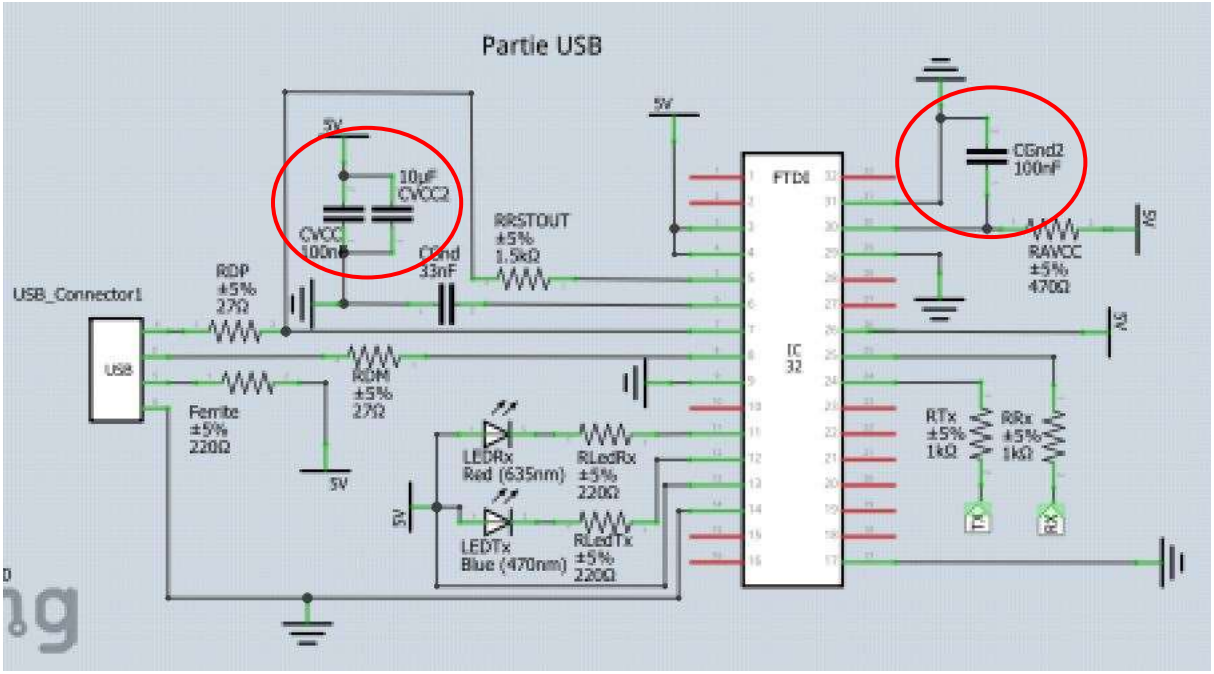


L'ancienne version



La nouvelle version que j'ai proposée

Les cercles rouges représentes les différences entre le schéma que j'avais conçu et celui de M.Redon.



Version imprimée par M.Redon

Lorsque le FT232BL reçoit un message, il le transmet ensuite au microcontrôleur. Cette transmission est assurée via les broches Tx et Rx présentes sur les deux composants. En examinant le schéma électrique, j'ai constaté que les branches Rx étaient reliées entre elles. De même pour les branches Tx.

J'ai donc reconnecté correctement les branches, pour résoudre ce problème.

Après différents échanges avec M. Redon, je suis arrivé à une version finale de la manette.

Soudage plaque N°1

Après avoir reçu ma manette j'ai réalisé le soudage, grâce aux conseils de M. Flamen.

Une fois soudée je suis passé aux contrôles de la carte.

- J'ai d'abord vérifié que chaque pin de chaque composant était relié aux bons composants.
- Puis j'ai vérifié les connexions pour la masse et le VCC.
- Les composants critiques, comme le FT232BL et le microcontrôleur, ont subis plusieurs vérifications sur leur « environnement ». Leurs capacités, résistances, quartz… ont été contrôlés pour s'assurer que les valeurs étaient les bonnes.

Tous ces contrôles ont été réalisés à l'aide d'un multimètre.

Une fois toutes ces vérifications effectuées, nous avons connecté la manette à mon PC. Celui-ci n'a pas reconnu la manette. Elle n'était pas présente dans la liste des périphériques, sur Windows comme sur Linux.

Soudage plaque N°2

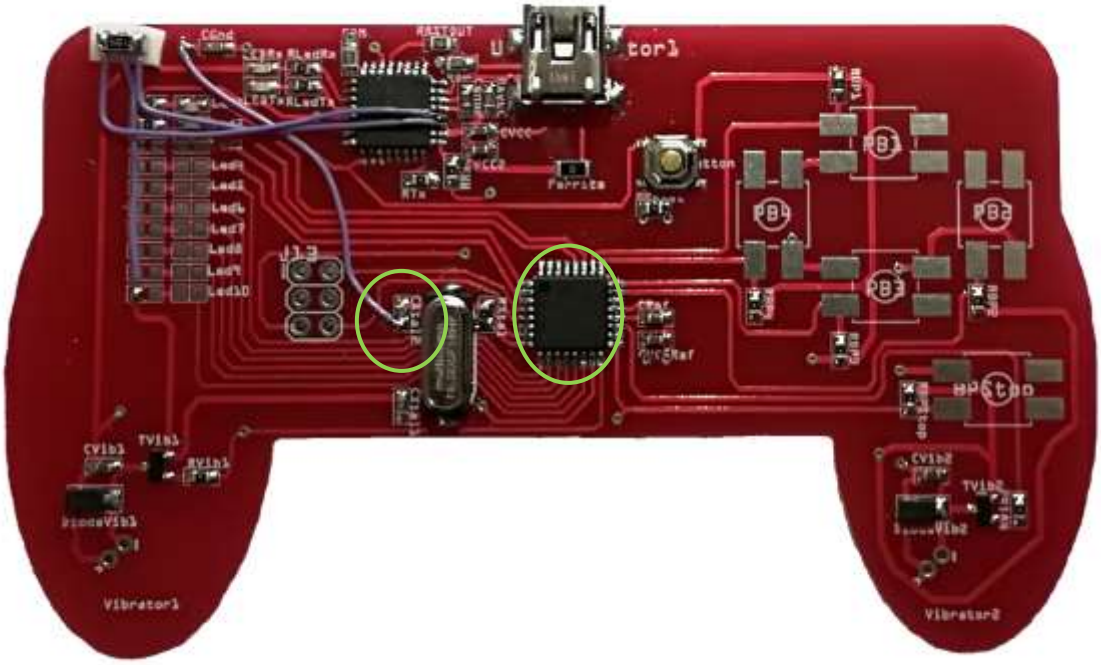
Après avoir rencontré plusieurs échecs sur le PCB, M.Flamen m'a recommandé d'en souder un nouveau. Il m'a expliqué qu'avoir branché la manette, alors que les connexions étaient mauvaises, pouvait avoir endommagé les composants.

M.Redon m'a fourni une plaque neuve et les composants nécessaires.

J'ai donc soudé une nouvelle plaque, avec ces composants.

J'ai effectué les mêmes tests que sur la première plaque.

Un des contrôles a révélé une erreur qui n'avait pas été détectée sur la première plaque.



La capacité, entourée en vert n'était pas reliée à la masse comme le voulait la datasheet de l'atmega.

En ajoutant un fil volant, ce problème a été corrigé.

Une des pins de l'atmega ne possédait pas de connexion avec le VCC. Nous avons églament rajouté un fil volant, qui n'est pas visible sur cette photo.

Puis j'ai branché ma manette à mon PC.

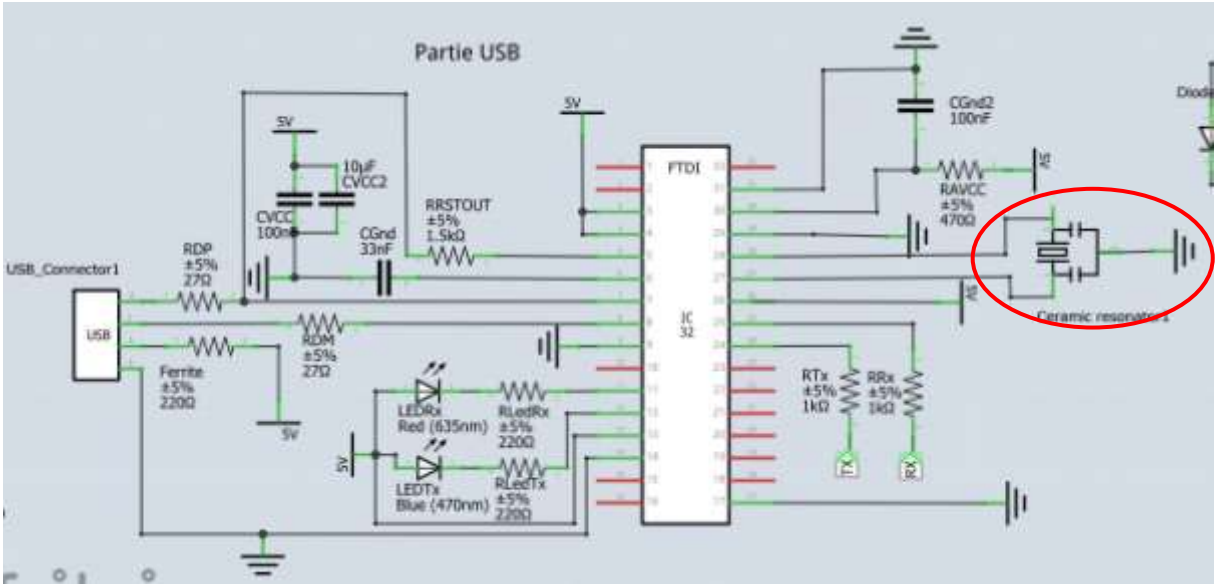
Le PC n'a pas détecté la manette, que ce soit sur Windows ou sur Linux.

Ajout de composant plaque N°2

M. Redon m'a informé qu'il serait nécessaire d'ajouter sur le FTDI un ceramic resonator. Je n'avais pas installé ce composant : selon la documentation, il n'est pas obligatoire, sauf dans certaines situations. Voir ANNEXE 3 (extrait de la doc technique du FTDI).

M. Redon a obtenu de bons résultats, avec une carte soudée partiellement : juste le FTDI, pas de microcontrôleur.

Le schéma suivant présente l'implantation de ce composant.



FT232BL ajout d'un ceramic resonator

Avec M. Flamen, nous avons ajouté ce dernier composant. Malgré plusieurs vérifications, et essais, nous n'avons pas réussi à faire fonctionner la manette. Il se pourrait que le fait de ne pas avoir utilisé de ceramic resonator ait grillé le FTDI ou le microcontrôleur. Mais c'est invérifiable.

2. Ecriture du programme manette

Chaque manette est composée de : 10 LEDs, 5 boutons et 2 vibreurs.

Pour pouvoir détecter les changements d'état états des boutons, j'ai placé les pins de l'atmega328p en entrée. Si un bouton est activé le pin devient actif. Un simple test sur PINn (n le nom du port) permet de repérer cette activation.

Pour les LEDs et les vibreurs, il faut placer les pins des ports reliés aux composants en sortie. Puis en utilisant PORTn & 0xXX on peut contrôler l'activation/désactivation des pins.

Si les deux manettes disposent d'éléments identiques (LEDs, vibreurs et boutons), elles n'ont pas le même protocole de communication.

La manette 328p doit communiquer via le port série, en utilisant le format de message des trames IP-UDP.

Le format du message est : *entête IP + entête UDP + données*

Format de l'en-tête IP

0 1 2 3	4 5 6 7	8 9 10 11 12 13 14 15	16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Version d'IP	Longueur de l'en-tête	Type de service	Longueur totale
Identification			Indicateur / Fragment offset
Durée de vie		Protocole	Somme de contrôle de l'en-tête
Adresse source			
Adresse destination			
Option(s) + remplissage			

Format de l'entête UDP + Données

0	32 bits
Port source	Port destinataire
Longueur	Checksum
Données	

Format de l'en-tête UDP

Dans mon programme, j'ai créé un tableau de type unsigned_char, de taille 32. Il permet de stocker toutes les données qui doivent être envoyées au PC. Il sera appelé pour envoyer un message. Voir annexe 2.

Les premières entrées correspondant à l'en-tête IP, les suivantes à l'UDP et les dernières mes données. La section données est remplie avant l'envoi d'un message, à partir de la lecture des ports du microcontrôleur.

Chaque en-tête contient différentes informations : @IP source, @IP destination, numéro de port …

J'ai rempli ces informations, qui ne peuvent pas être nulles. Comme elles n'ont pas d'influence n'importe quelle valeur convient (sauf 0).

Ces informations me permettent de tester le calcul des checksums IP et UDP, via les fonctions calcul_checksum_ip et calcul_checksum_udp.

Le calcul du checksum IP se fait sur l'en-tête IP, il prend en compte toutes les informations allant de la version d'IP jusqu'au protocole.

Dans le cas de l'UDP, le calcul prend en compte @IP source, @IP de destination, le protocole IP et l'en-tête UDP jusqu'à la longueur de la trame UDP. Les données font également partie du calcul.

Pour les deux calculs, on additionne les datas des différents éléments qui sont pris en compte. Si le résultat est supérieur à 0xFF, on additionne la partie supérieure 0xFF00 comme s'il s'agissait de 0xDD.

Quand ces additions sont effectuées, on complémente le tout.

Une fois ces calculs réalisés, la trame de données est prête à être envoyée.

A intervalles réguliers, la manette va envoyer au PC un message, avec les données récoltées auparavant.

La manette attendra toujours de recevoir un accusé de réception du PC avant de continuer ses autres fonctions. Si l'accusé indique une mauvaise réception, la manette renverra la trame.

Lorsque la manette reçoit un message du PC, il s'agit toujours d'une commande qui doit être appliquée. Ce message prédéfini est stocké dans un tableau, pour faciliter son analyse.

Il y a deux types de messages :

- le changement d'état d'une LED,
- une demande d'envoi d'un nouveau message.

3. Ecriture du programme PC

Le programme PC a deux objectifs principaux : il doit afficher les données reçues par la manette et permettre à un utilisateur d'interagir avec cette manette.

Lorsqu'il est lancé, le programme va essayer d'accéder au port /ttyACM0 grâce à la commande :

```
fd = open(''/dev/ttyACM0'',O_RDWR | 0_NOCTTY )
```

Grâce à l'option O_RDWR, l'accès au port se fait en lecture et en écriture.

L'option O_NOCTTY permet de séparer le processus d'ouverture du terminal.

Si l'accès est réussi, le programme continue normalement, sinon le programme s'arrête et affiche un message d'erreur demandant à l'utilisateur de vérifier la connexion PC-manette.

Dans le cas où la connexion est correcte, la communication avec la manette s'effectuera via les fonctions read() et write() de la bibliothèque unistd.h.

Dans ces fonctions, il faut indiquer fd qui cible une adresse de lecture/écriture, un message à transmettre et la taille de la trame à lire/écrire, par exemple, read(fd,lu,sizeof(lu)) :

- fd -> lecture sur le port /dev/ttyACM0
- lu -> zone de stockage des données
- sizeof(lu) -> taille des données à lire

On utilise aussi l'instruction write(fd,mess_emi,TAILLE_EMI) avec :

- fd -> écriture sur le port /dev/ttyACM0
- mess_emi -> zone de stockage des données
- TAILLE_EMI -> taille des données à écrire

mess_emi et lu sont des tableaux que je crée dans mon programme pour stocker les datas qui m'intéresse. Ils sont déclarés généralement, pour que toutes mes fonctions aient accès aux données.

Pour le format des données envoyées, le programme est similaire à celui de la manette. J'ai donc renseigné des @IP de source et de destination, des numéros de ports… qui correspondent à ceux que j'avais écrits pour le programme manette.

On retrouve également un calcul des checksums.

Afin de limiter les risques d'erreurs, j'ai mis en place un envoi d'accusé de réception. Lorsqu'un message est reçu, le programme calcul les checksums et compare ses résultats avec ceux indiqués par la trame de données. S'ils correspondent, le message est considéré comme valide, un message est envoyé à la manette pour confirmer la bonne réception. S'ils diffèrent, une demande d'envoi d'un nouveau message est envoyée.

Le programme dispose de plusieurs fonctions pour analyser les messages reçus :

- affichage_etat_LEDs() : affichage des états des LEDs
- affichage_etat_vibreurs() : affichage des états des vibreurs
- affichage_etat_boutons() : affichage des états des boutons

Toutes ces fonctions récupèrent la partie « données » du message reçu.

Puis elles réalisent des comparaisons pour savoir si un composant est actif ou non.

Cette comparaison est de la forme mess_recu[val] & 0xDD. DD est une puissance de 2, qui me permet de tester un bit précis dans mess_recu[val].

Les appels de ces fonctions dépendent des choix de l'utilisateur.

Lorsque le programme est activé, il va proposer à l'utilisateur différents choix :

1- Demander et afficher les états des LEDs
2- Demander et afficher les états boutons
3- Demander et afficher les états des vibreurs
4- Demander et afficher les états de tous les composants
5- Changer l'état d'une LED
6- Arrêter le programme

Les demandes 1 à 4 sont des messages auxquels la manette doit répondre par un message donnant les états des composants choisis.

Une fois le message de la manette reçu, le programme l'affichera sur l'écran de l'utilisateur.

La demande 5 envoie un message qui changera l'état d'une LED.

4. Programmes

Vous trouverez les programmes dans mon Wiki :

- Programme manette. Voir fichier « Prog_manette_328p.zip »
- Programme PC. Voir fichier « Prog_PC_328p.zip »

5. Tests

Afin de tester les programmes, M.Redon m'a prêté une carte Arduino.

Grâce à celle-ci, j'ai pu vérifier la détection de la manette par le programme PC : fonction OPEN.

Cette partie du programme fonctionne, la manette est correctement détectée par le PC.

Le reste des programmes ne pouvait pas être testé par cette Arduino.

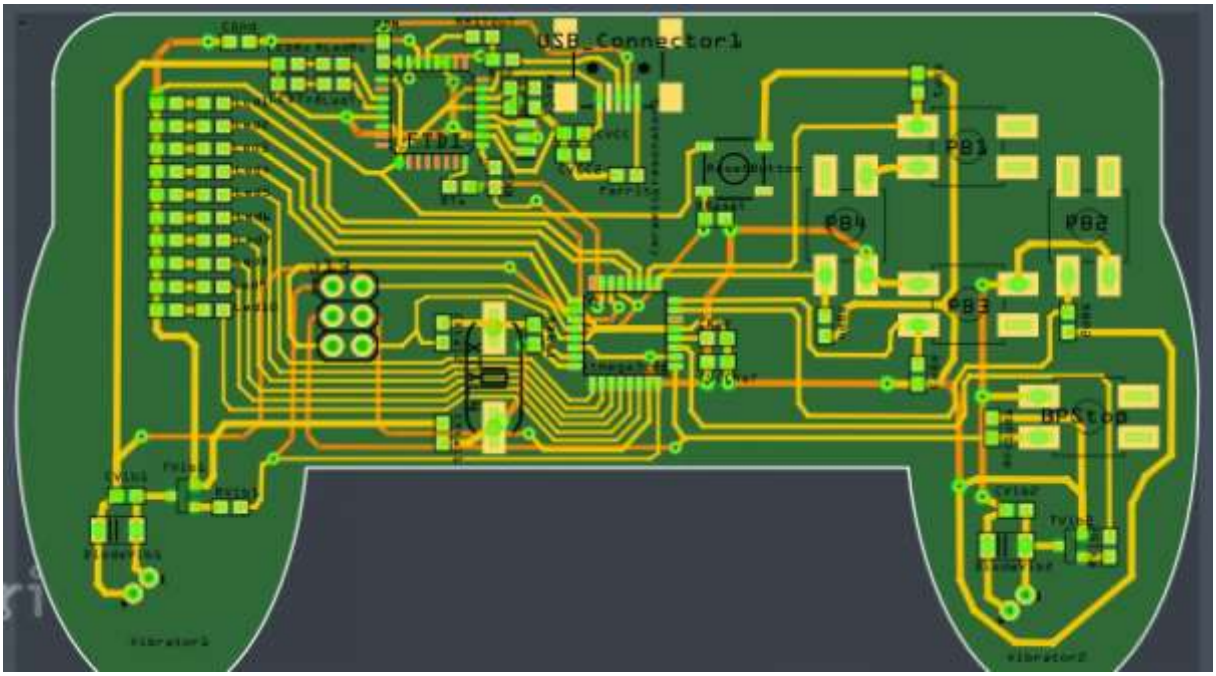
6. Conclusion pour la manette 328p

La manette 328p n'a jamais été reconnue par le PC. Pour la première fabrication, nous avons pensé à des composants défaillants. Mais la deuxième plaque donne le même résultat, avec et sans le ceramic resonator.

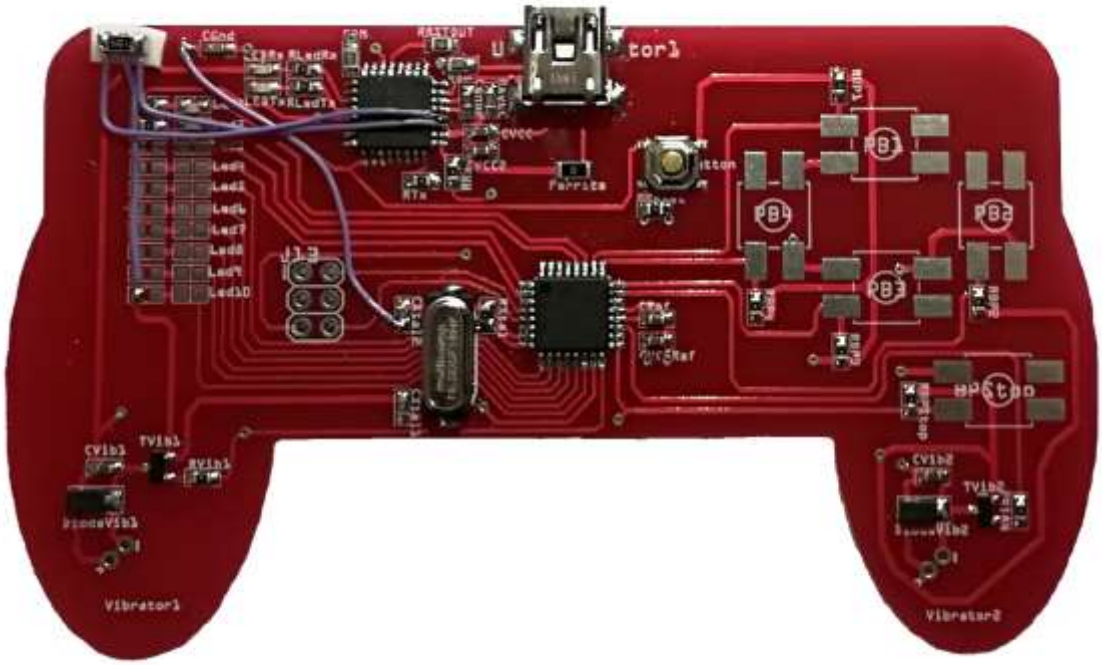
Les programmes manette et PC sont rédigés. Le seul test possible et réussi a été la détection d'une carte Arduino par mon programme PC.

J'ai perdu beaucoup de temps en vain dans la recherche de la cause du dysfonctionnement de la manette.

Par manque de temps pour reprendre l'analyse du PCB, je ne peux pas finir cette partie du projet.



La dernière version de la manette (FRITZING)



La manette 328p finale (ceramic resonator en haut à gauche)

III. Réalisation de la manette 16u2

Cette manette comporte un microcontrôleur atmega16u2

1. PCB

Résultats de mon prédécesseur

Comme pour la première manette, je disposais du PCB réalisé par un collègue.

Lorsque M. Redon m'a remis le PCB, il m'a expliqué que la manette n'avait jamais été soudée. Par conséquent, je ne savais pas si elle fonctionnait.

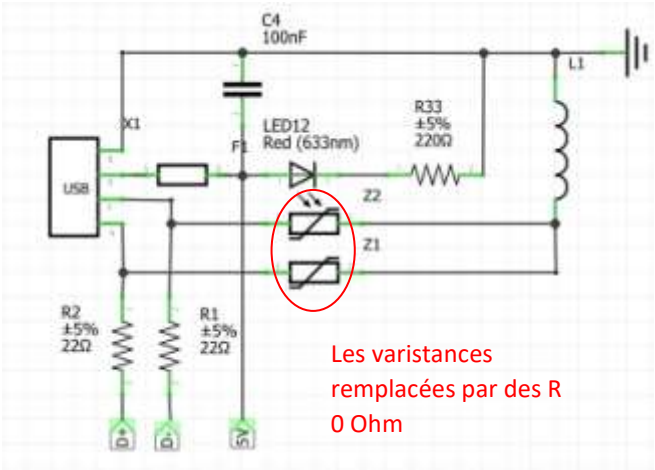
Soudage et modifications

Mon objectif avec cette manette était de la souder afin de déterminer si elle fonctionnait ou non.

Dès que j'ai reçu les composants, j'ai réalisé le soudage sur le PCB.

Pour vérifier qu'elle fonctionnait correctement je l'ai connectée à mon ordinateur. Elle n'a pas été reconnue.

En examinant le schéma électrique et en prenant en compte les composants soudés, M. Flamen et moi avons découvert un court-circuit. En effet afin de remplacer certains composants (des varistances) M. Redon m'avait fourni des résistances 0 Ohms. Or leur position dans le circuit provoquait le court-circuit. (Voir figure ci-dessous)



L'ancien schéma électrique

Finalement, j'ai dû supprimer ces composants de mon PCB pour qu'il fonctionne normalement.

2. Ecriture du programme manette

Pour cette manette, la programmation doit se faire à l'aide de la bibliothèque LUFA.

La bibliothèque LUFA

Cette bibliothèque permet à un utilisateur de programmer un périphérique USB, pour que son comportement corresponde à l'usage souhaité.

Pour mon projet, la manette devait se comporter comme un périphérique de type HID.

Compréhension

Avant de pouvoir programmer, j'ai effectué des recherches sur la bibliothèque, et prendre le temps d'analyser son fonctionnement.

Si j'ai réussi à trouver quel exemple était le plus proche de mon projet, je n'étais pas sûr des points que je devais modifier pour l'adapter à mon cas. Ma principale inquiétude était que si je modifiais trop de points (taille, contenu des données envoyées…), le programme ne fonctionnerait plus et je me retrouverais en difficulté.

De plus, beaucoup des sites et forums que j'avais consultés, partaient du principe que vous compreniez LUFA parfaitement. Même si certains forums s'adressent à des débutants, il leur manque certaines précisions.

Voici néanmoins ce que j'ai compris.

Les rapports

Un message en LUFA est appelé un rapport. Un rapport est émis chaque fois que la fonction qui le créé indique qu'il y a une différence, entre le rapport créé à l'instant t et celui à l'instant t-1. Ainsi on évite de saturer le système connecté à notre HID.

La taille d'un rapport peut être adaptée à la quantité d'informations à envoyer. La taille du rapport exemple, convenait à mes besoins. Je n'ai eu qu'à l'exploiter telle quelle.

Puisque j'avais seulement besoin de modifier la création des rapports, c'est sur la fonction CALLBACK_HID_Device_CreateHIDReport que je me suis concentré.

Dans le rapport exemple, cette fonction remplit le rapport en indiquant les états des LEDs du système sur lequel le programme est implanté.

Dans mon projet, j'ai remplacé le test des LEDs par un test sur les pins du microcontrôleur associés aux boutons.

Ex : pour un test sur le bouton haut, le code est `etat_LED[0] = ((PINC & 0x10) != 0) ?0x01 :0x00` .

Cependant, en réalisant quelques tests, je me suis rendu compte que cette méthode ne fonctionnait pas : la manette entrait dans une boucle infinie et ne réagissait plus aux actions sur les boutons. J'ai donc créé un tableau qui stocke les résultats des tests.

Les données du tableau sont ensuite affectées dans la trame. Cela a permis à ma manette de retrouver un fonctionnement normal.

Ma programmation

Une fois la gestion des messages en place, j'ai programmé le contrôle des LEDs et des vibreurs.

Pour contrôler un élément avec une atmega 16u2, il faut agir sur le port auquel cet élément est connecté.

D'abord, il faut définir sur le port quels pins sont des entrées et les lesquels sont des sorties.

Ex : DDRB & 0x01 définit le pin 0 du port B en sortie.

J'ai donc écrit une fonction output_input_init qui initie tous les ports selon mes besoins.

Ensuite j'ai travaillé sur les fonctions de contrôle des LEDs et des vibreurs. allume_eteindre_led(num_LED,action) et activite_vibreur(num_vib,act). Le premier paramètre désigne un élément précis, le second l'action à effectuer (1 : activer ; 2 : désactiver).

Les deux fonctions sont très similaires. Pour pouvoir activer un élément il faut activer le pin qui lui correspond. La commande est PORTn ^ code_element avec :

- n le port associé au pin,
- code_element le code qui correspond au pin du port.

Ex : PORTB ^ 0x01 change l'état de la pin 0 du portB.

Cependant, cette commande a pour limite de ne pas prendre en compte l'état dans lequel se trouve le port. Pour éviter d'obtenir l'effet inverse de celui souhaité, j'ai mis en place un tableau qui répertorie dans quel état se trouve les LEDs. Un autre également pour les vibreurs.

Ainsi, avant que la commande ne soit appelée, un test sur les valeurs dans le tableau est exécuté.

Quand un vibreur, ou une LED change d'état, la valeur dans le tableau associé est mise à jour.

Je n'ai pas commencé par cette partie tout de suite car je devais écrire de nouvelles fonctions et j'avais peur qu'elles entrent en conflit avec celles de LUFA.

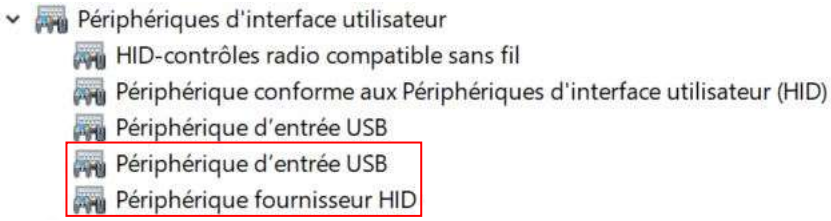
Finalement il n'y a pas eu de problème, et j'ai pu faire fonctionner mes fonctions sans provoquer d'erreur.

3. Tests

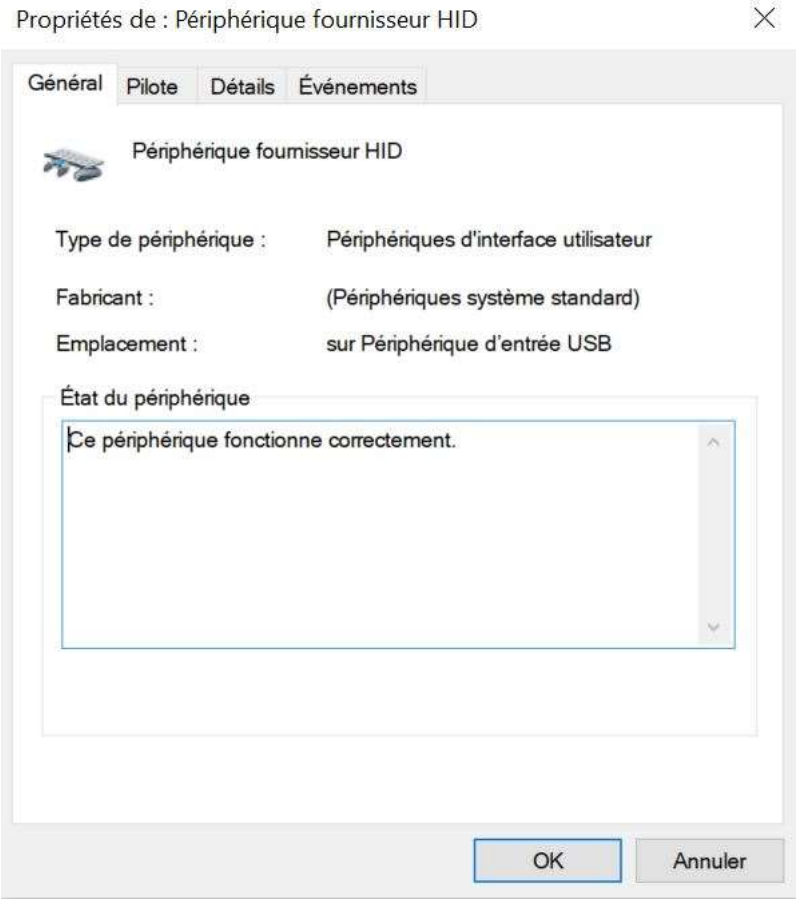
Un élément important, à contrôler sur la manette est qu'elle soit détectée correctement par le PC.

Elle doit être reconnue comme un périphérique HID.

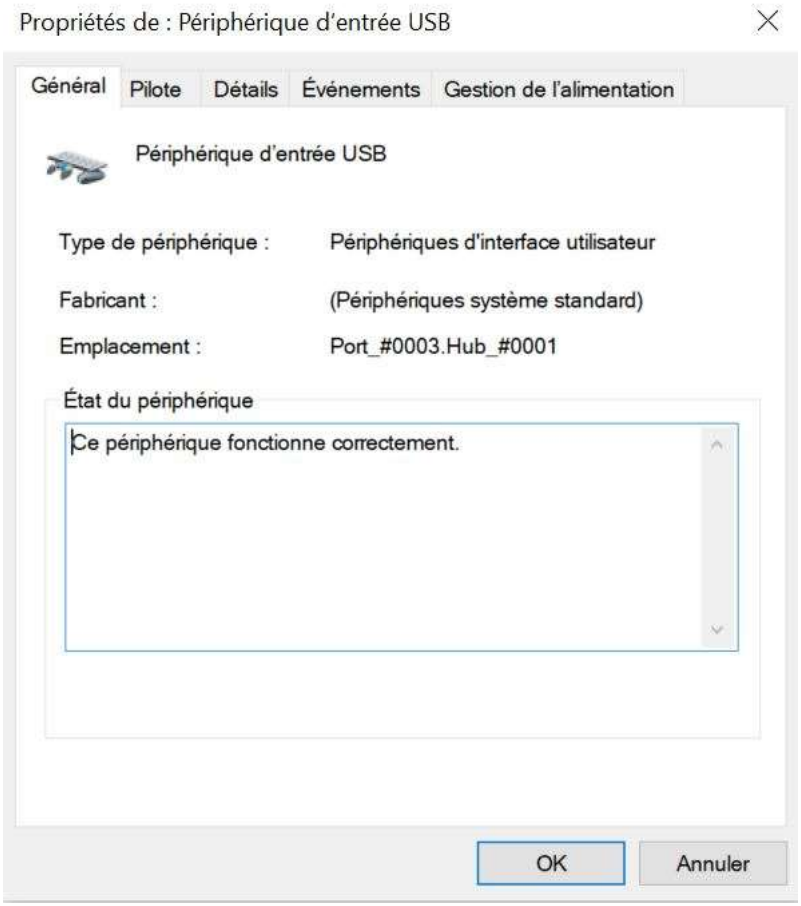
Sur Windows le gestionnaire de périphériques indique que la manette correspond aux emplacements suivants :



En étudiant leurs propriétés on obtient les résultats suivants :



Les propriétés sur HID



Les propriétés sur USB

Afin de vérifier toutes les propriétés du composant, j'ai recherché sur Linux les commandes qui afficheraient les informations détaillées de la manette. Je cherchais également à afficher les messages transmis par la manette.

Une première information importante est le vendor et le product ID. Ces identifiants uniques sont utiles pour reconnaitre mon périphérique parmi d'autres.

La commande lsusb permet d'obtenir ce résultat :

```
boens@boens-Lenovo-ideapad-510S-14ISK:~$ lsusb
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 004: ID 8087:0a2a Intel Corp.
Bus 001 Device 003: ID 3938:1031
Bus 001 Device 002: ID 03eb:204f Atmel Corp. LUFA Generic HID Demo Application
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

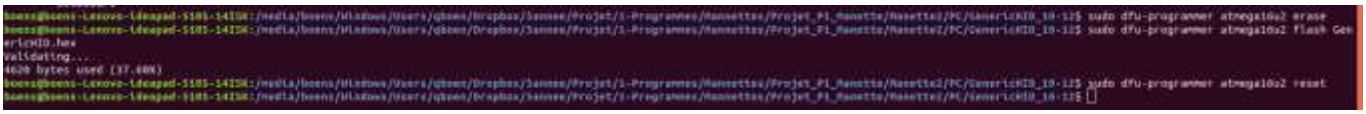
Ma manette est donc reconnue par les ID 03eb :204f (vendor :product)

Pour récupérer les propriétés de la manette, j'ai de nouveau utilisé lsusb. Cette fois, la commande affiche le détail des informations sur le périphérique cible, grâce à l'option -D. Ici, ma manette est identifiable par la cible /dev/bus/usb/001/006

```
boens@boens-Lenovo-ideapad-510S-14ISK:~$ lsusb -D/dev/bus/usb/001/006
Device: ID 03eb:204f Atmel Corp. LUFA Generic HID Demo Application
Couldn't open device, some information will be missing
Device Descriptor:
  bLength                18
  bDescriptorType         1
  bcdUSB               1.10
  bDeviceClass            0 (Defined at Interface level)
  bDeviceSubClass         0
  bDeviceProtocol         0
  bMaxPacketSize0         8
  idVendor           0x03eb Atmel Corp.
  idProduct          0x204f LUFA Generic HID Demo Application
  bcdDevice            0.01
  iManufacturer           1
  iProduct                2
  iSerial                 0
  bNumConfigurations      1
  Configuration Descriptor:
    bLength                 9
    bDescriptorType         2
    wTotalLength           34
    bNumInterfaces          1
    bConfigurationValue     1
    iConfiguration          0
    bmAttributes         0xc0
      Self Powered
    MaxPower              100mA
    Interface Descriptor:
      bLength                 9
      bDescriptorType         4
      bInterfaceNumber        0
      bAlternateSetting       0
      bNumEndpoints           1
      bInterfaceClass         3 Human Interface Device
      bInterfaceSubClass      0 No Subclass
      bInterfaceProtocol      0 None
      iInterface              0
        HID Device Descriptor:
          bLength                 9
          bDescriptorType        33
          bcdHID               1.11
          bCountryCode            0 Not supported
          bNumDescriptors         1
          bDescriptorType        34 Report
          wDescriptorLength      32
         Report Descriptors:
           ** UNAVAILABLE **
      Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress     0x81  EP 1 IN
        bmAttributes            3
          Transfer Type            Interrupt
          Synch Type               None
          Usage Type               Data
        wMaxPacketSize     0x0008  1x 8 bytes
        bInterval               5
```

Toutes ces informations sont conformes à mon périphérique.

Le chargement du programme, dans la manette, se fait avec dfu-programmer



Enfin, pour récupérer les données transmises par manette j'ai téléchargé la commande usbhid, via les librairies usbutils et hidrd.

Pour voir les messages, j'écris la commande usbhid-dump --entity=stream

J'obtiens ainsi l'affichage du descriptor de ma manette, et des messages reçus.

Ils correspondent biens à ceux associés aux boutons.

		001:009:000:DESCRIPTOR 1544352474.633172 06 00 FF 09 01 A1 01 09 02 15 00 25 FF 75 08 95 08 81 02 09 03 15 00 25 FF 75 08 95 08 91 02 C0 Starting dumping interrupt transfer stream with 1 minute timeout.
BOUTON HAUT	Appui	001:009:000:STREAM 1544352479.954023 48 00 00 00 00 00 00 00
	Relâchement	001:009:000:STREAM 1544352480.174096 00 00 00 00 00 00 00 00
BOUTON GAUCHE	Appui	001:009:000:STREAM 1544352482.534135 00 47 00 00 00 00 00 00
	Relâchement	001:009:000:STREAM 1544352482.726134 00 00 00 00 00 00 00 00
BOUTON DROIT	Appui	001:009:000:STREAM 1544352484.414058 00 00 44 00 00 00 00 00
	Relâchement	001:009:000:STREAM 1544352484.618127 00 00 00 00 00 00 00 00
BOUTON BAS	Appui	001:009:000:STREAM 1544352486.110188 00 00 00 42 00 00 00 00
	Relâchement	001:009:000:STREAM 1544352486.274188 00 00 00 00 00 00 00 00
BOUTON STOP	Appui	001:009:000:STREAM 1544352488.334227 00 00 00 00 50 00 00 00
	Relâchement	001:009:000:STREAM 1544352488.518220 00 00 00 00 00 00 00 00

Les messages reçus suivant les actions

4. Ecriture du programme PC

Durant mes semaines de recherche, j'ai consacré du temps pour la préparation d'un programme PC, équivalent à celui de la manette 1.

Cependant M. Redon m'a précisé que cette partie était hors sujet. Je l'ai donc supprimée de mon projet.

Ce programme aurait géré l'affichage des états des LEDs, des vibreurs et des boutons. Il aurait également permis à l'utilisateur de donner des ordres à la manette.

5. Programme

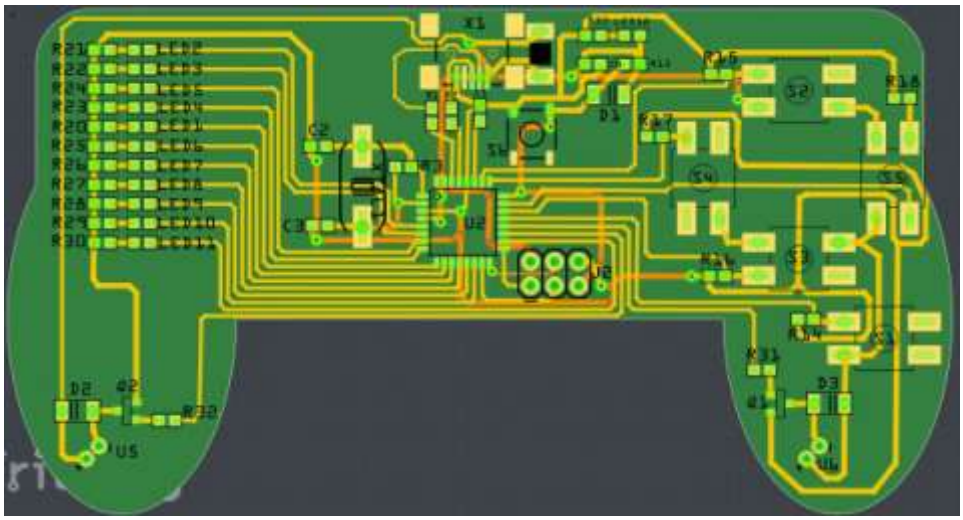
Vous trouverez le programmes dans mon Wiki :

- Programme manette. Voir fichier « Prog_manette_16u2.zip »

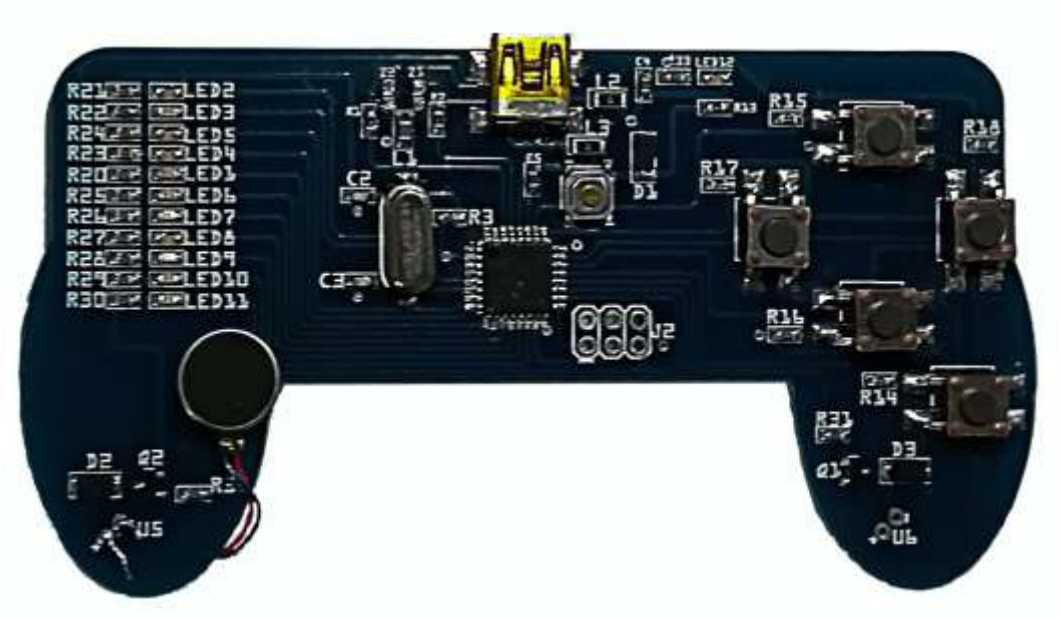
6. Conclusion pour la manette 16u2

La manette 16u2 est terminée, à l'exception d'un vibreur. Sur les 4 que j'avais reçus, 3 ont disparu.

La manette 16u2 fonctionne conformément au cahier des charges. Les Leds s'allument sur commande. Les boutons envoient les signaux attendus par le PC et le vibreur réagit aux commandes des boutons.



Version finale de la manette (FRITZING)



La manette finale

IV. Annexes

Annexe 1 : Récapitulatif des activités du projet semaine par semaine

Semaine	Manette 328p	Manette 16u2
1 : 17/09 au 21/09	Analyse et compréhension du sujet	Analyse et compréhension du sujet
2	Correction erreurs PCB	Aucune activité
3	Routage et conception PCB	Divergence entre fichier fritzing et PCB Soudage partiel
4	Programmation manette Correction du PCB par M. Redon	Aucune activité
5	Programmation manette Commande de la fabrication du PCB	Arrivée de composants
6	Réception du PCB Programmation PC	Soudage Problème sur la manette
7 (Vacances) Du 29/10 au 02/11	Programmation PC	Aucune activité
8	M. Flamen indisponible, pas de soudage	Acquisition de connaissances LUFA
9	1ère soudage : échec	Soudage manette Manette fonctionnelle
10	Contrôle du PCB Réception de nouveaux composants	Acquisition de connaissances LUFA
11	2ème soudage : échec	Programmation de la manette Programmation du PC (finalement déclaré hors sujet)
12	Aucune activité Priorité à la manette 16u2	Programmation manette Tests : réussis
13 : du 10/12 au 14/12	Dernier soudage avec un nouveau composant : échec	Soudage vibreur Test vibreur : réussi

Annexe 2 : Tableau des données envoyées par la manette 328p

```
// Partie IP

////VERSION IP + LONGUEUR EN TETE

ip_udp[0]=0x45;      //Version de IP : IPV4

                     //Sans options l'entete vaut 5 : nombres de lignes de mots de 32 bits

//// SERVICE

ip_udp[1]=0x00;    //Type de service

                   // 0001 1100 : Low Delay, Débit élevé, Très Fiable

//// LONGUEUR TOTALE TRAME

            //Notre trame à une longueur de 31 octets

ip_udp[2]=0x00;    //Longueur totale partie 1

ip_udp[3]=0x1F;   //Longueur totale partie 2

//// IDENTIFICATION

ip_udp[4]=0x00;   //Identification partie 1

ip_udp[5]=0x03;   //Identification partie 2

//// FLAGS + FRAGMENT

ip_udp[6]=0x40;    //Ox40 = 0b01000000  Flags en position 010 : on n'autorise pas la fragmenation

                   //Les derniers bits sont les premiers de la fragmentation

ip_udp[7]=0x00;

//// TTL

ip_udp[8]=0x40;    //ttl = durée

//// PROTOCOLE

ip_udp[9]=0x11;    //protocole udp donc 0x11

//// CHECKSUM IP

ip_udp[10]=0x00;    //checksum partie 1

ip_udp[11]=0x00;    //checksum partie 2

//// @IP SOURCE MANETTE 172.26.145.207

ip_udp[12]=0xAC;

ip_udp[13]=0x1A;

ip_udp[14]=0x91;
```

```
ip_udp[15]=0xCF;
//// @IP DESTINATION PC 172.26.145.206
ip_udp[16]=0xAC;
ip_udp[17]=0x1A;
ip_udp[18]=0x91;
ip_udp[19]=0xCE;
// Partie UDP
//// PORT SOURCE
ip_udp[20]=0x00;
ip_udp[21]=0xA0;    //Ecoute sur 2121
//// PORT DESTINATION
ip_udp[22]=0x0F;
ip_udp[23]=0xA0;   //Envoi sur 4000
//// LONGUEUR UDP (en tete + data)
ip_udp[24]=0x00;
ip_udp[25]=0x0B;    //Longueur = data (3 octets) + param udp (8)
//// CHECKSUM UDP
ip_udp[26]=0x00;  // checksum partie 1
ip_udp[27]=0x00;  // checksum partie 2
//// DONNEES
/*
  Etat :   - boutons x5
                                - vibreurs x2
                                - LEDs x10
  Le tout sur 3 octets
  */
ip_udp[28]=0x00;  //Boutons 1 à 4 + Bouton Stop + Vibreurs 1 et 2 + LED 1
ip_udp[29]=0x00;  //LEDs 2 à 9
ip_udp[30]=0x00;  //LED 10
                //Toutes les valeurs sont initialisées à 0
```

Annexe 3 : Extrait de la datasheet du FT232BL

27	XTIN	Input	Input to 6MHz Crystal Oscillator Cell. This pin can also be driven by an external 6MHz clock if required. Note: Switching threshold of this pin is VCC/2, so if driving from an external source, the source must be driving at 5V CMOS level or AC-coupled to centre around VCC/2.
28	XTOUT	Output	Output from 6MHz Crystal Oscillator Cell. XTOUT stops oscillating during USB suspend, so take care if using this signal to clock external logic.

Datasheet concernant l'ajout d'un ceramic resonator

Annexe 4 : Références bibliographiques

Téléchargement biblio LUFA

http://www.fourwalledcubicle.com/LUFA.php

Le manuel LUFA

http://fourwalledcubicle.com/files/LUFA/Doc/151115/html/group___group___u_s_b_class_h_i_d.html

Explication fonctionnement bibliothèque LUFA

http://www.avrbeginners.net/new/wp-content/uploads/2011/10/avrbeginners_40_USB_Control_Transfers_with_LUFA_1.0.pdf

Exemple d'une utilisation de la bibliothèque LUFA

https://eleccelerator.com/tutorial-about-usb-hid-report-descriptors/

Récupération des messages HID

http://www.slashdev.ca/2010/05/08/get-usb-report-descriptor-with-linux/

Tutorial usbhid-dump

https://www.systutorials.com/docs/linux/man/8-usbhid-dump/

Forum sur la modification des tailles des rapports

https://www.avrfreaks.net/forum/lufa-hid-atmega32u4

Logiciel FLIP

http://www.pic-control.com/loading-arduino-bootloader-to-brand-new-atmel-microcontroller/

Utilisation de FLIP

https://www.d-r.nl/forum/index.php?topic=13.0

Manuel d'installation de FLIP (dans mon wiki onglets document rendus : IMA4_P1_Connexion_manette_PC.pdf)

https://projets-ima.plil.fr/mediawiki/index.php/IMA4_2018/2019_P1