Haute Ecole Economique et Technique

Avenue du Ciseau, 15

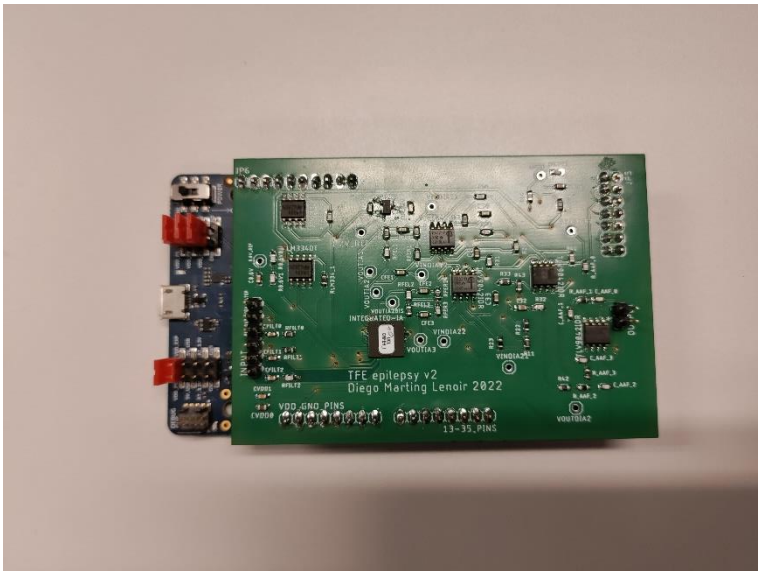
1348 Louvain-La-Neuve

Travail de fin d'études présenté en vue de l'obtention du diplôme de bachelier en Informatique et Système : finalité Technologie de l'informatique

---

Amélioration d'un système de détection de crises d'épilepsie : modification du régulateur de tension et mise en place du module de transmission et de visualisation des données

Diego MARTING LENOIR 3TL1



Rapporteuse : Stéphanie Guérit

Année Académique 2021-2022

Remerciement

Je tiens à remercier toutes les personnes qui m’ont aidé lors de mon tfe

Tout d'abord, j'adresse mes remerciements à ma rapporteuse Stéphanie Guérit qui m’a aidé à trouver le sujet de tfe via son réseau à l’UCL et qui m’a guidé durant la préparation de celui-ci.

Je tiens à remercier vivement David Bol et Rémi Dékimpe qui m’ont accepté et fait confiance pour prendre en main le projet. Plus particulièrement à Rémi Dékimpe qui m’a épaulé et répondu à toutes mes questions par rapport au projet.

Pour finir je tiens à remercier toutes les personnes qui m’ont aidé lors de la rédaction de ce tfe.

Table des matières

Introduction :

L'épilepsie est une famille de maladie neurologique dont le point commun est une prédisposition cérébrale qui engendre des crises spontanées qui touche plus de 50 millions de personnes. Cette maladie se manifeste souvent chez les jeunes ou les plus de 65 ans. Deux personnes sur trois peuvent prendre des médicaments pour se soigner. Une autre solution est de faire une chirurgie mais peu de personnes peuvent la faire. Pour finir, l'une des dernières solutions est d'appliquer des stimulations au cerveau.

Il existe deux solutions :

- The deep brain stimulation(DBS)

Cette option est très invasive des électrodes sont directement insérées profondément dans le cerveau et présentent un risque non négligeable.

- The vagus nerve stimulation (VNS)

Cette méthode stimule le nerf vagus grâce à des électrodes. Ce nerf est directement connecté au cerveau. En utilisant cette façon de faire on est moins invasive que le DBS et donc présentant moins de risques.

Dans ce travail on utilise la méthode VNS. On peut utiliser celle-ci de deux façons. La première est de tout le temps stimuler le nerf mais celle-ci est consommatrice d'énergie et n'est pas spécialement viable sur le long terme. La deuxième manière, est de stimuler le nerf seulement quand on détecte la possibilité qu'une crise se déclenche. Dans ce cas, il faut récupérer des signaux via des électrodes pour détecter la crise. Le problème c'est que le signal est d'environ 7.1 µVrms et qu'il y a des interférences. Il fallait donc implémenter un système qui récupére les données, les amplifie, enlève le bruit le tout en étant à basse consommation.

Ce travail est composé de différentes parties :

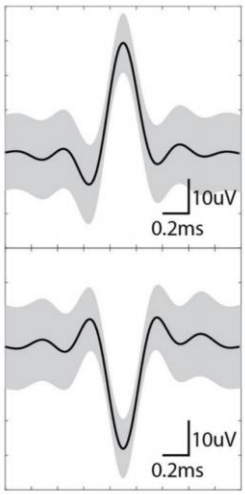
- VENG
- Méthodologie
- Le fonctionnement
- Travail précédent
- Objectif
- PCB
- Transfert des données du PCB vers Apollo3 Blue
- Transfert apollo3 Blue vers Pc
- Acquisition et affichages de données sur Pc
- Conclusion

Chacune de ces parties explique comment fonctionne chaque système mis en place lors de ce travail. Et pour finir, nous clôturerons par une brève conclusion qui résumera ce projet et proposera une vision future de celui-ci.

VENG

Les informations utilisées pour expliquer le signal VENG proviennent du mémoire de Jaminon-De Roeck, Chen-Terry.

Le nerf vague est porteur d'information parasympathique et innerve plusieurs organes. On peut par exemple y détecter des changements respiratoires et cardiaques lors d'une crise. Ce nerf est directement relié au cerveau où les crises sont déclenchées. Pour les détecter, il faut identifier les biomarqueurs associés. D'après des expériences, ceux-ci prennent une forme triphasique comme constatée sur l'image suivante.



Dans le VENG, mes formes tryphasique sont des rafales de basse et haute amplitude synchrone à la respiration et au rythme cardiaque. L'amplitude moyenne et de crête dépendent de l'électrode utilisé et de la distance entre les électrodes. En moyenne, le temps d'une pointe est inférieur à 1.5ms. Son amplitude est de 7.1+/-2.3 µV quand on utilise des électrodes brassards tripolaires avec 2mm d'espacement entre chaque électrode. L'amplitude de crête est différente entre les pics positifs et négatifs.

Positif : 20.7 +/-6.6 µV
Néfatif : 24.1 +/- 7.7 µV

Projet

- Méthodologie

Pour la réalisation de ce travail j'ai été amené à travailler avec l'UCL, plus particulièrement avec Rémi Dékimpe. En effet, ma rapporteuse Stéphanie Guérit m'a proposé un projet en collaboration avec l'UCL a pour but d'améliorer un mémoire réalisé en 2021 par Jaminon-De Roeck, Chen-Terry. Ce travail avait pour objectif de créer un printed circuit board (PCB) qui reçoit un signal en entrée puis amplifie le signal pour le ressortir à une pin. Apparemment il avait déjà fait un code permettant de traiter le signal et le transmettre mais ce code n'a pas été fourni.

Ce projet a été un défi pour moi. En effet, je me suis retrouvé avec des informations pour lesquelles je ne suis pas familier. J'ai donc dû procéder à une certaine documentation afin de comprendre les différents termes utilisés dans le mémoire. Avant de vraiment me plonger dans la réalisation du travail.

J'ai dû aussi comprendre comment fonctionnent les différents systèmes utilisés lors du travail. Car n'étant pas habitué à utiliser l'environnement de l'Apollo j'ai passé beaucoup de temps à saisir comment il fonctionne.

- Fonctionnement

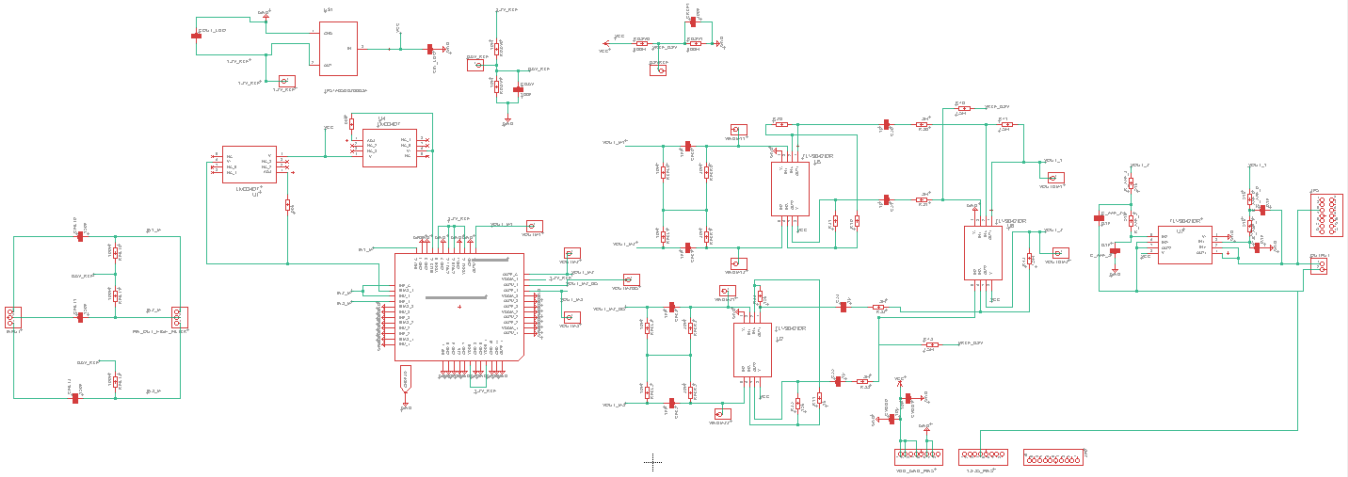
Le projet a pour but d'acquérir un signal l'amplifier, enlever le bruit et les transmettre sur un ordinateur pour visualiser les données. Donc dans un premier temps on vient récupérer les données grâce à des électrodes pour ensuite les faire passer sur le pcb qui va les amplifier pour ensuite transmettre à l'Apollo les données tout en enlevant le bruit. Pour finir l'Apollo va envoyer les données vers un ordinateur grâce à un port USB. Un programme python viendra lire les données sur un port COM pour ensuite les afficher.

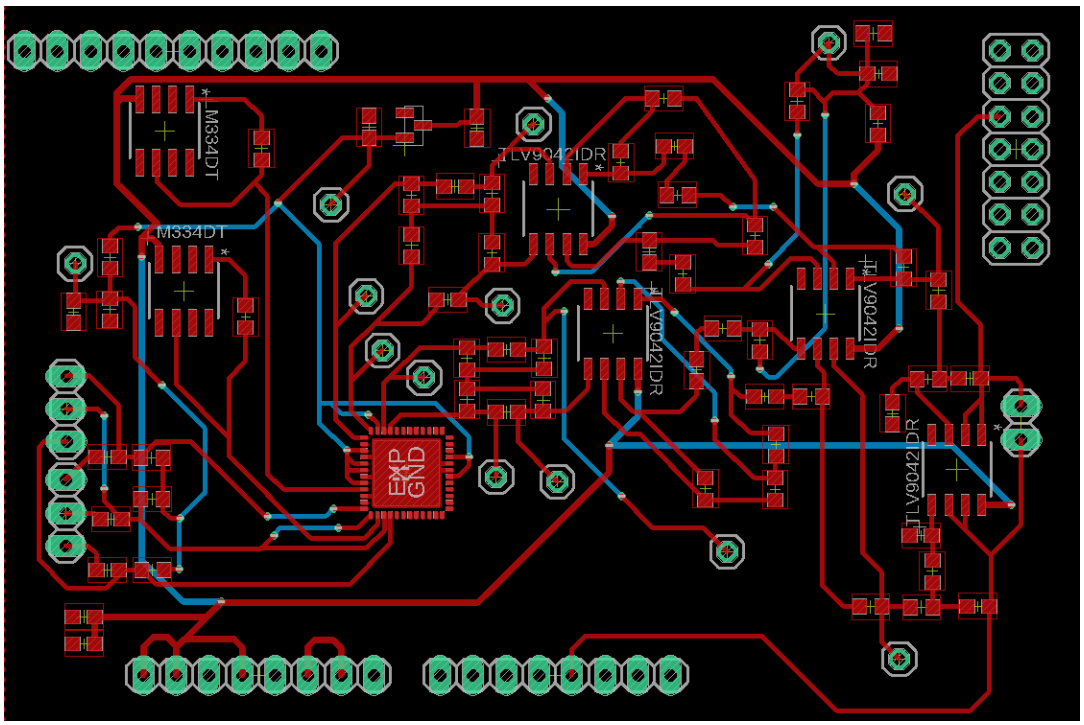
- Objectif

L'objectif est d'améliorer l'ancien travail. Pour cela, il fallait réduire la consommation du PCB en modifiant le Low-Dropout regulator (LDO). Pour ensuite transférer les signaux du PCB vers l'apollo3 Blue en les échantillonnant pour enfin transfert les données sur un ordinateur via un câble USB pour les afficher.

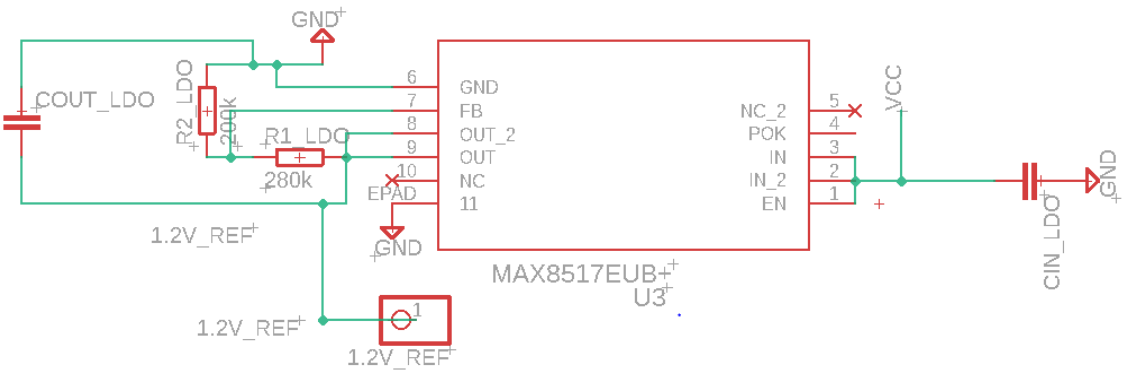
- PCB

En me basant sur le travail précédent, celui-ci mettait en place un LDO qui valait 71.3% de la consommation totale. Et donc il proposait de changer le LDO par le TPS7A0512PDBZR. Qui ne faisait que 10.8 µW à la place de 504 pour le LDO actuel. Mais malheureusement celui-ci n'était plus disponible. Il a fallu opter pour un autre LDO qui est le MCP1703AT-1202E/CB. Celui-ci permet toujours de réguler la tension à 1.8V.

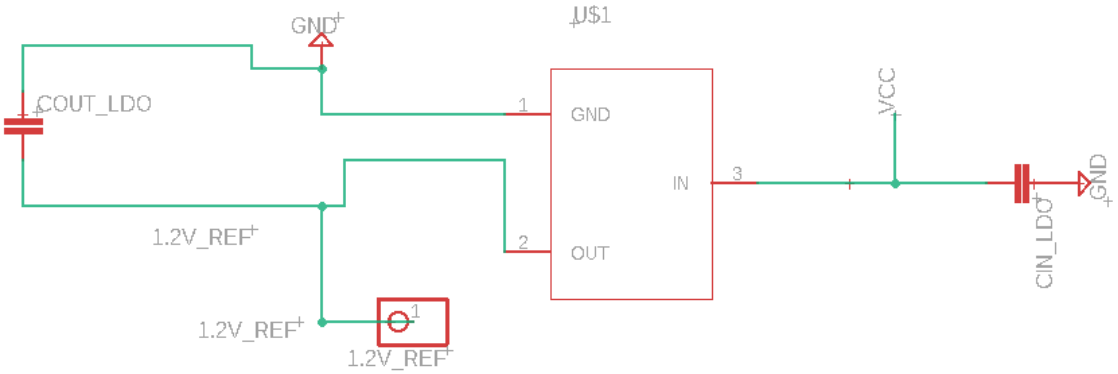




Donc pour changer le LDO il fallait revoir le schéma du PCB. Donc sur le schéma cette partie-ci a été changé :



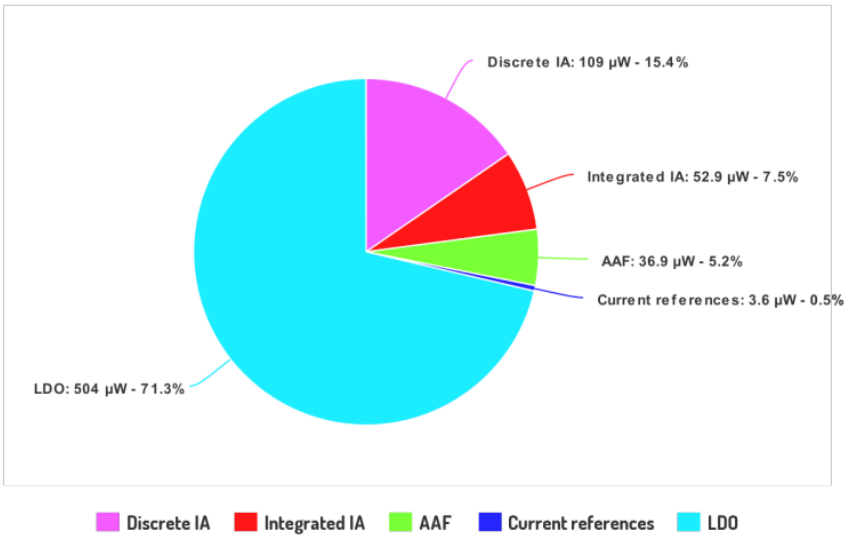
Par celle-ci :



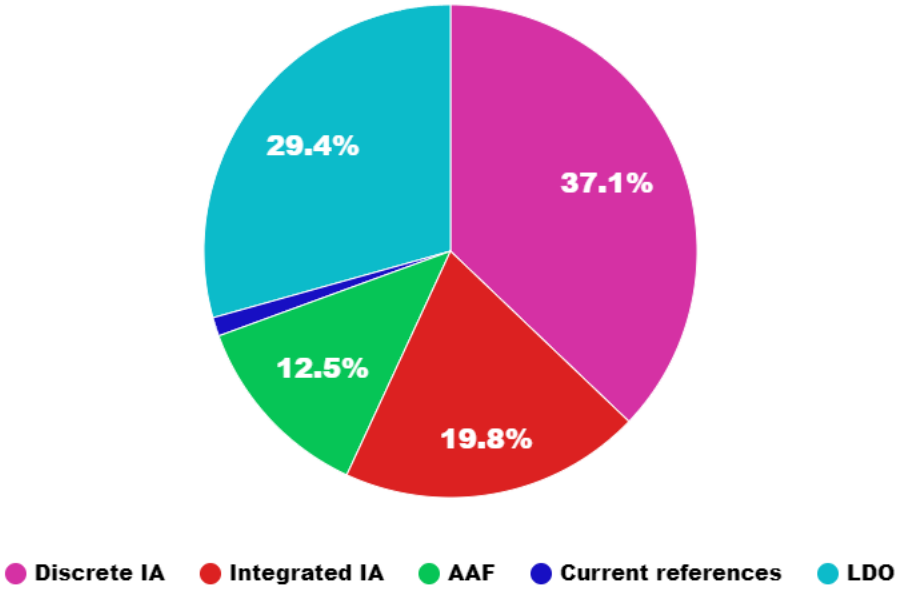
Donc pour ce faire, il a fallu créer une libraire sur Eagle grâce à la datasheet du LDO. Et ensuite enlever les résistances de sortie comme indiqué dans la documentation. Une fois la librairie validée et le schéma changé, la demande d'impression de la carte a été effectuée. Dès que la carte a été reçue, elle été assembler avec les différents composants.

Dès que toutes ces étapes ont été finies. Avec l'UCL on a testé le PCB pour vérifier qu'il fonctionnait bien. Et donc, la consommation finale du PCB à lui tout seul par rapport à l'ancien travail à diminuer il est passé d'un total de 700 µW a 288 µW. Le nouveau LDO ne consomme plus que 86 µW a la place de 504 µW. À la page suivante vous retrouvez le détail de la consommation avec un graphique de l'ancien projet comparé au nouveau.

Graphique de la consommation de l’ancien projet :



Graphique de la consommation avec le nouveau LDO :



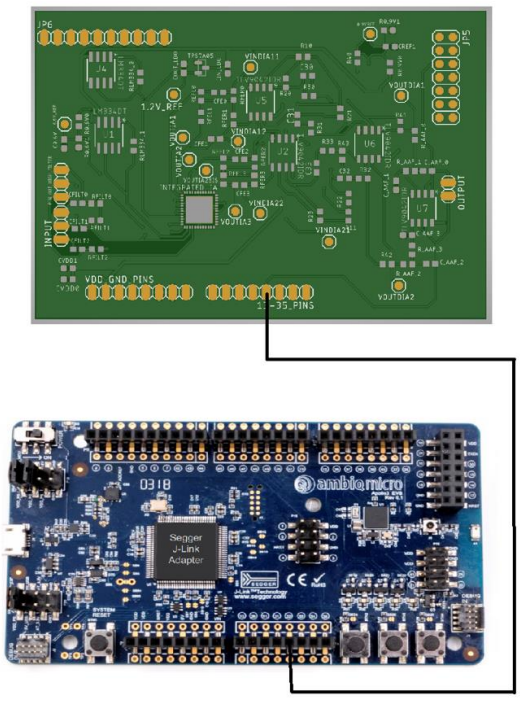
On peut voir sur le nouveau graphique que le LDO n'est plus la source principale de la consommation du système. Malheureusement le TPS7A0512PDBZR est sensé donnée 10 µW alors qu'ici avec le nouveau LDO on atteint les 86 µW. Il faudrait peut-être à l'avenir si cela en vaut la peine de changer le LDO actuel. Mais actuellement on a un gain de 412 µW ce qui n'est pas du tout négligeable. Après ce qui pourrait être aussi intéressant par la suite c'est de refaire une structuration du Board pour essayer d'avoir quelque chose de plus propre et optimisé. Mais cela ne nuise en rien à l'utilisation de la carte.

- Transfert des données du PCB vers Apollo3 Blue

Tout d'abord avant de transférer un signal il faut expliquer ce qu'est l'Apollo 3 Blue. C'est une carte qu'on va venir emboiter dans le PCB. Cette carte munie d'un microprocesseur va permettre d'y injecter du code et de traiter les informations reçues analogiquement pour les numériser. Une fois cette opération effectuée on a la possibilité de faire un peu près ce que l'on veut avec.

Donc dans un premier temps notre préoccupation est de recevoir les données du PCB. On va alors utiliser l'ADC (Analog-to-Digital Converter) qui permet de convertir l'analogique en numérique.

Pour commencer, il faut connecter l'Apollo3 et le PCB ensemble. Sur le PCB on fait ressortir le signal à une pin et celle-ci est connecté à la pin 32 de l'Apollo.



Au niveau du code, on va configurer l'Apollo pour dire que la pin32 utilise l'ADC en utilise le canal 0. Pour ce faire, on va débloquer la configuration GPIO et on sélectionne l'ADC au pad 32. Ensuite on l'active en lui donnant sa configuration qui est celle-ci (le code source est repris d'un repository mais modifié pour notre utilisation) :

```
u32Cfg =      _VAL2FLD(ADC_CFG_CKMODE,ADC_CFG_CKMODE_LLCKMODE)   //Low Latency Clock Mode
            | _VAL2FLD(ADC_CFG_CLKSEL,ADC_CFG_CLKSEL_HFRC)      //HFRC clock
            | _VAL2FLD(ADC_CFG_TRIGSEL,ADC_CFG_TRIGSEL_SWT)     //Software trigger
            | _VAL2FLD(ADC_CFG_REFSEL,ADC_CFG_REFSEL_INT2P0)    //internal 2V reference
            | _VAL2FLD(ADC_CFG_LPMODE,ADC_CFG_LPMODE_MODE1)     //Low Power Mode 1
            | _VAL2FLD(ADC_CFG_RPTEN,ADC_CFG_RPTEN_SINGLE_SCAN) //Single scan
            | _VAL2FLD(ADC_CFG_ADCEN,1);                        //Enable ADC

//
// Using slot 0
//
ADC->SL0CFG = 0;
ADC->SL0CFG_b.CHSEL0 = ADC_SL0CFG_CHSEL0_SE4; //use ADCSE4 as input;
ADC->SL0CFG_b.ADSEL0 = ADC_SL0CFG_ADSEL0_AVG_1_MSRMT; //Avaerage over 1 measurements
ADC->SL0CFG_b.PRMODE0 = ADC_SL0CFG_PRMODE0_P14B; //use 14-bit 1.2MS/s
ADC->SL0CFG_b.WCEN0 = 0; //window comparator mode disabled
ADC->SL0CFG_b.SLEN0 = 1; //slot enabled

ADC->WULIM_b.ULIM = 0; //Window comparator upper limit (not used)
ADC->WLLIM_b.LLIM = 0; //Window comparator lower limit (not used)

ADC->INTEN_b.WCINC = 1;             //enable window comparator voltage incursion interrupt
ADC->INTEN_b.WCEXC = 1;             //enable window comparator voltage excursion interrupt
ADC->INTEN_b.FIFOOVR1 = 1;          //enable FIFO 100% full interrupt
ADC->INTEN_b.FIFOOVR2 = 1;          //enable FIFO 75% full interrupt
ADC->INTEN_b.SCNCMP = 1;            //enable ADC scan complete interrupt
ADC->INTEN_b.CNVCMP = 1;            //enable ADC conversion complete interrupt

ADC->CFG = u32Cfg;

ADC->INTCLR = 0xFFFFFFFF;           //clear interrupts
ADC->SWT = 0x37;                    //trigger ADC
```

Plusieurs tests ont été effectué avec un signal qui variait de 0V à 1.8V sur différentes fréquences avec un nombre différent de mesure prise :

- 14 mesures :
    - 1KHZ

    Le signal de sortie était coupé on ne recevait que la tension entre 0.80V et 1V.
    - 500HZ

    Le signal de sortie était presque correct mais avec parfois un manque d'informations.
- 1 mesure :
    - 1KHZ

    Le signal de sortie était le signal attendu.
    - 500HZ

    Le signal de sortie était le signal attendu.

Pour conclure cette partie, le transfert fonctionne bien du PCB vers l'Apollo cependant il faudrait tester quel est le bon compromis entre le nombre de mesures faite et l'envoie d'une donnée. Comme ça on pourrait être plus précis vis-à-vis du signal de sortie. Il faudrait voir aussi s'il est possible d'optimiser le code en passant par la mémoire DMA.

- Transfert apollo3 Blue vers Pc

Une fois l'Apollo ayant numérisé le signal grâce à l'ADC, on peut commencer à transférer ces données sur le PC. Pour cela, il existe plusieurs solutions :

- BLE :

  Le BLE (Bluetooth low energy), est une technique de transmission qui complète le bluetooth. Comparé à celui-ci, il permet un débit de 1Mbit/s pour une consommation d'énergie 10 fois moindre. Ce qui lui permet d'être mis en application dans des montres connectées etc.

- UART :

  L'UART dit universal asynchronous receiver / transmitter est un protocole qui est dédié à l'échange de données entre deux appareils. Il suffit de deux câbles entre l'émetteur et le récepteur pour recevoir et émettre dans les deux sens.

Dans notre cas les deux modes de transmission ce valle. Sachant que le BLE est plus intéressant à long terme pour ce projet. Malheureusement, on va utiliser l'UART cela est dû à un manque de temps et de compréhension du module BLE. Mais cela ne va en rien entraver l'objectif.

Donc on va partir sur l'UART, la configuration se fait de cette manière (comme pour l'ADC le code vient du même repository) :

```
    //

    PWRCTRL->DEVPWREN |= (1 << PWRCTRL_DEVPWREN_PWRUART0_Pos);

    while(PWRCTRL->DEVPWRSTATUS_b.HCPA == 0) __NOP();

    //
    // Enable clock / select clock...
    //
    pstcUart->CR = 0;
    pstcUart->CR_b.CLKEN = 1;                //enable clock
    pstcUart->CR_b.CLKSEL = 1;               //use 24MHz clock

    //
    // Disable UART before config...
    //
    pstcUart->CR_b.UARTEN = 0;               //disable UART
    pstcUart->CR_b.RXE = 0;                  //disable receiver
    pstcUart->CR_b.TXE = 0;                  //disable transmitter


    //
    // Starting UART config...
    //

    // initialize baudrate before all other settings, otherwise UART will not be initialized
    SystemCoreClockUpdate();
    ConfigureBaudrate(pstcUart,u32Baudrate,24000000UL);

    // initialize line coding...
    pstcUart->LCRH = 0;
    pstcUart->LCRH_b.WLEN = 3;               //3 = 8 data bits (2..0 = 7..5 data bits)
    pstcUart->LCRH_b.STP2 = 0;               //1 stop bit
    pstcUart->LCRH_b.PEN = 0;                //no parity

    //
    // Enable UART after config...
    //
    pstcUart->CR_b.UARTEN = 1;               //enable UART
    pstcUart->CR_b.TXE = 1;                  //enable transmitter
}
```

Les seules choses importantes à savoir sur le code c'est qu'on va transmettre des données sur 8 bits et il n'y a que la transmission d'active. On n'active pas la réception car cela n'est pas nécessaire. L'Apollo3 n'a pour but que de transmettre les données. A noter que l'Apollo est configuré à 230400 Baud Rate cette valeur permet de définir la vitesse de transmission sur le câble, ça correspond au nombre de bits transmis par seconde.

Une fois la configuration terminée, on va aller dans la boucle principale du programme qui effectue le transfert de données de l'ADC à l'UART.

```
while(1)
{
        if (ADC->INTSTAT_b.CNVCMP == 1)
    {
        ADC->INTCLR_b.CNVCMP = 1;


        if ((_FLD2VAL(ADC_FIFO_COUNT,ADC->FIFO) > 0) && (_FLD2VAL(ADC_FIFO_SLOTNUM,ADC->FIFO) == 0))
        {
            u32Data = _FLD2VAL(ADC_FIFO_DATA,ADC->FIFO) >> 6;
            sprintf(data, "%d-",u32Data);
            PutStringUart(UART0,data);
            ADC->FIFO = 0; // pop FIFO

        }
    }
        if (ADC->INTSTAT_b.SCNCMP == 1)
    {
        ADC->INTCLR_b.SCNCMP = 1;
        ADC->SWT = 0x37;
    }

}
```

Dans cette boucle, on va vérifier que l'ADC contient quelque chose et que son canal est bien le 0. Une fois la vérification effectuée on vient récupérer les données dans l'ADC et les convertir en une chaine de caractères pour pouvoir rajouter un symbole qui permettra plus tard de distinguer les différentes données reçues. Une fois la conversion faite on envoie la chaine de caractères dans l'UART. À la fin on vérifie bien qu'on enlève la valeur de l'ADC pour pouvoir faire une nouvelle mesure.

Pour conclure, l'UART fonctionne très bien. La chose à voir pour le futur du projet est de regarder pour passer les données avec le BLE. Ce qui pourrait être intéressant pour pouvoir utiliser l'ensemble du système sans être attacher à un ordinateur.

- Acquisition et affichages des données sur Pc

Maintenant que l'Apollo peut transmettre des données via l'UART, il faut créer un script qui permet de recevoir ces données et les afficher. Ce script est fait en python car ce langage de programmation permet grâce à de simples librairies de mettre en application et d'exécuter un script très facilement.

Donc le but est d'afficher sur un graphique le signal qu'on reçoit via l'UART au moment où on lance le script. Premièrement on va importer les différents modules à utiliser pour faire fonctionner le script :

```
1  import serial
2  import matplotlib.pyplot as plt
3  import numpy as np
4  import sys
```

Le module serial permet de recevoir les données de l'apollo3.
Matplotlib est le module qui va afficher un graphique de nos données.
Numpy va permettre d'effectuer des opérations mathématiques avancé.
Le dernier module sys va lui arrêter le code en renvoyant un message d'erreur.

Ensuite on va découper le code en plusieurs fonctions :

- read()

  Cette fonction est celle qui lit les données envoyées sur le port COM, qui les traite et les stocks dans un tableau. Elle va lire les 1000 premières données reçues. Elle va retourner deux variables end et datas.

- graph(datas)

  Cette fonction va afficher les données récupérer par la fonction read. Elle prend en paramètre les données enregistrées.

Dans la fonction read, il faut commencer par dire au programme sur quel port COM écouter et quel est la vitesse de transmission des données. On définit donc dans le code le Baud Rate à 230400. Et le port COM est demandé quand on lance le script.

```
7   def read():
8       s = serial.Serial('COM'+sys.argv[1])
9       s.baudrate = 230400
```

Une fois la configuration de la communication établie on va définir différentes variables :

```
10      datas = []
11      data = ''
12      countExit = 0
```

datas est un tableau qui permet de stocker toutes les données reçues.

data est une chaine de caractère qui va recevoir les données et jusqu'au caractère d'échappement. Pour ensuite ajouter la valeur à datas et ensuite se vider.

countExit teste combien de fois on passe dans la boucle. Elle permet de tester si on est sur le bon port COM.

Une fois les variables établies, on peut passer à la boucle principale :

```
13      while True:
14          countExit = countExit + 1
15          if countExit == 150000:
16              return False,False
17
18          temp = s.read_all().decode("UTF-8")
19          if temp != '':
20              for i in temp:
21                  if i == '-':
22                      count += 1
23                      datas.append(round(int(data)/16384 * 1.8,2))
24                      data = ''
25                  else:
26                      data += i
27                  if len(datas)==1000:
28                      return False,datas
```

La première étape dans cette boucle est de vérifier grâce à un compteur qu'on n'est pas sur au mauvais port COM. En effet si le compteur est devenu trop élevé et qu'on n'a toujours pas eu les données affichées, c'est qu'on n'est pas sur le bon port. Dans ce cas-là on retourne deux valeurs à False, ce qui permettra de dire que le code doit se terminer.

Ensuite, on vient mettre dans la variable temp les données lues sur le port. On reçoit normalement les données en bits donc on va venir décoder celle-ci avec le format UTF-8. Comme cela on n'aura plus à traiter le format bits mais une chaine de caractères.

Une fois les données reçues on regarde si la valeur reçue n'est pas ‘’. Comme ça on évite d'ajouter les données qui ne sont pas nécessaires au script.

Par après on boucle sur la variable temp testé dans un premier temps pour voir si on retrouve le caractère qui permet de donner la fin d'une valeur ‘-‘. Sinon on rajoute le caractère dans la variable data. Si on trouve le caractère de fin alors on ajoute data dans notre tableau en utilisant la formule suivante : valeur/2^nbrbits(ici 14 via l'ADC) * tension de référence(1.8 V). Cette formule permet de convertir les valeurs de l'adc qui sont des entiers vers les vraies valeurs de tension.

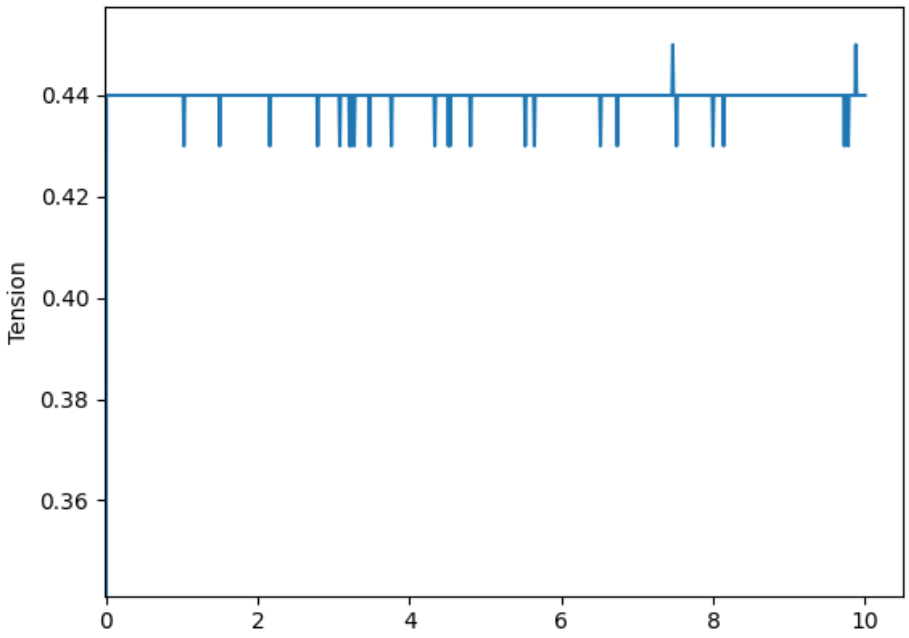
Pour finir à la fin de cette boucle il y a un compteur qui permet de vérifier si on a bien atteint les 1000 données souhaiter. Donc si on les a bien reçu on retourne False et datas ce qui respectivement va mettre un terme à la boucle de test qui se situe plus loin et va renvoyer les données.

Dans la fonction graph, on va vouloir afficher un graph avec les 1000 données reçues.

```
29    def graph(datas):
30        x=np.linspace(0,10,1000)
31        plt.plot(x,datas)
32        plt.ylabel('Tension')
33        plt.xlabel("")
34        plt.show()
```

Donc on va dans un premier temps définir l'axe des abscisses en donnant un intervalle 0 à 10. En ayant que 1000 valeurs souhaité dans celui-ci. Ensuite on va envoyer les données sur le graphique.

On voit sur le graphique ci-dessous une tension de 0.44 V.



Pour finir, le programme est exécuté dans une boucle qui permet d'éviter des erreurs.

```
37 if __name__ == "__main__":
38     stop = True
39     count = 0
40     while stop:
41         try:
42             stop,datas = read()
43         except:
44             count = count+1
45             if count == 20:
46                 print("Port COM incorrect")
47                 sys.exit()
48     if datas == False:
49         print("Port COM incorrect")
50         sys.exit()
51     graph(datas)
52
```

Dans un premier temps on initialise stop et count qui permettent de tester le nombre d'erreurs du programme et d'arrêter celui-ci quand on dépasse la limite.

Donc dans la boucle tant que stop est égale à True alors on va exécuter le programme. On va utiliser try et except qui vont nous permettent de continuer à lancer read jusqu'à 20 fois. Car parfois lors de la transmission de données il y a des erreurs de réception qui stop le programme direct. Donc pour contourner cette erreur, on va boucler 20 fois jusqu'au moment où on a plus l'erreur et que le script démarre. Sinon on arrête le programme car on ne sera pas sur le bon port COM car le bouclage de 20 permet à chaque fois de lancer la lecture sur le bon port.

Ensuite, on va récupérer stop et datas. Stop va arrêter la boucle et datas ce sont les données reçues par la lecture qui vont être envoyé dans la fonction graph. Si datas vaut False alors il y a eu une erreur de port COM on quitte donc le programme.

Pour exécuter le programme il faut lancer la console et donner en paramètre le numéro du port utilisé.

```
PS C:\Users\dmart\OneDrive\Bureau\dev\tfe\code> python .\main.py 6
Port COM incorrect
```

Pour conclure on pourrait améliorer le code pour essayer de recevoir les données en temps réel et ne pas à chaque fois exécuter le script pour recevoir les 1000 premières données. Et pourquoi ne pas essayer si cela est pertinent de stocker les données sous un fichier texte.

Historique du projet

20/01 réunion projet + ressources
21/04 assemblage pcb sans LDO
27/04 LDO
06/05 LDO
01/06 test du projet

Lors de ce travail, j'ai effectué plusieurs réunions avec Rémi Dékimpe pour avoir les ressources du projet assemblé la carte avec les différents composants et à la fin tester si le tout fonctionne bien.

De base le planning s'agençait de cette manière :

- Mars : board du PCB finie
- Avril : Transfert du signal du PCB vers l'Apollo
- Mai : Transfert et affichage des données sur ordinateur

Mais malheureusement ce planning n'a pas peut-être respecté. Le board a été fini début avril. Et le reste du code a été finie mi-mai fin mai. Ce retard est dû à plusieurs facteurs. Le premier c'est le temps de travail réduit à cause du stage qui est effectué en même temps. Et ensuite le fait de pas pouvoir tout tester de chez moi car le matériel nécessaire aux tests se trouvait à l'UCL.

Donc à cause de ce retard le module BLE n'a pas été implémenté et l'UART a été préféré pour être sûr de finir dans les temps.

Conclusion

L'épilepsie est une famille de maladie neurologique qui touche plus de 50 millions de personnes. Cette maladie se manifeste souvent chez les jeunes ou les plus de 65 ans.

En utilisant la méthode VNS, on stimule le nerf seulement quand on détecte la possibilité qu'une crise se déclenche. On récupère donc des signaux via des électrodes pour détecter la crise. Le signal étant d'environ 7.1 µVrms ayant des interférences. Il fallait donc implémenter un système qui récupéré les données, les amplifie, enlève le bruit le tout en étant à basse consommation.

Le signal qui nous intéresse est le VENG, ce signal est émis par le nerf vagus et permet de détecter quand une crise d'épilepsie se déclenche grâce à des électrodes.

Ce projet est effectué en collaboration avec l'UCL en se basant sur le travail de Jaminon-De Roeck, Chen-Terry. Il a pour but d'améliorer le travail précédent en changeant le LDO, et faisant le transfert du signal vers l'Apollo qui va les envoyer sur un ordinateur pour les afficher.

Pour commencer, l'objectif d'améliorer l'ancien mémoire en changeant le LDO, faisant le transfert de données du PCB vers l'Apollo vers le PC pour finalement les afficher est bien fonctionnel.

Ensuite, malgré les différentes difficultés rencontrées lors du projet. Qui sont plus des problèmes de compréhension de l'environnement de travail. Qui au fur et à mesure du travail est devenue plus clair.

Pour la partie du LDO sur le pcb, ce qui pourrait être intéressant ce serait encore plus de réduire la consommation avec un autre LDO. Car celui mis actuellement baisse quand même de 4x la consommation mais il est surement encore possible de la baisser avec le LDO qui était prévu à la base pour ce projet.

Les données converties par l'ADC arrivent bien sur le script python dans un format qu'on va ensuite via une formule convertir pour avoir les bonnes valeurs de tension. À ce niveau-ci, il n'y a plus grand-chose à faire, peut être vérifié s'il est possible d'optimiser le code en passant par la mémoire DMA et baisser la consommation d'énergie sur l'Apollo. Il faudrait aussi tester quel est le bon compromis entre le nombre de mesures faite et l'envoie d'une donnée.

Le script python reçoit et affiche bien les données. Mais il n'affiche que les 1000 premières données. Il est surement possible d'afficher les données en temps réel sans à chaque fois relancer le script. Cela pourrait permettre d'avoir une meilleure idée du signal reçue.

Pour conclure, l'objectif est atteint mais il reste pas mal de petits points qui pourraient être améliorés et permettre d'avoir un projet encore plus fonctionnel et optimisé. Peut-être qu'une future personne travaillant sur le projet pourrait mettre en oeuvre les différentes améliorations.

Bibliographie

[1] Jaminon-De Roeck, Chen-Terry. Implementation and optimization of an ultra-low power vagus nerve sensing system for epileptic seizures detection. Ecole polytechnique de Louvain, Université catholique de Louvain, 2021. Prom. : Bol, David. http://hdl.handle.net/2078.1/thesis:30551

[2] Lars Stumpp, Hugo Smets, Simone Vespa, Joaquin Cury, Pascal Doguet, Jean Delbeke, Emmanuel Hermans, Christian Sevcencu, Thomas N. Nielsen, Antoine Nonclercq, and Riem El Tahry. Recording of spontaneous vagus nerve activity during pentylenetetrazol-induced seizures in rats. Journal of Neuroscience Methods, 343:108832, 2020.

[3] Ambiq, Apollo 3 Blue, https://ambiq.com/ apollo3-blue/

[4] Ambiq, Apollo 3 Blue datasheet, https://ambiq. com/wp-content/uploads/2020/10/Apollo3-Blue-MCU-Datasheet.pdf

[5] Ambiq, Apollo 2 Blue, https://ambiq.com/ apollo2-blue/

[6] Ambiq, Apollo Blue, https://ambiq.com/apollo/

[7] Ambiq, Apollo 4 Blue, Ambiq, Apollo 3 Blue, accessed January 26 2021, https://ambiq.com/apollo4-blue/

[8] Mouser, "MAX8518EUB+T", https://www.mouser.be/datasheet/2/256/MAX8516-1514578.pdf

[9] Mouser, "MCP1709AT-1202 E/CB", https://www.mouser.be/ProductDetail/Microchip-Technology/MCP1703AT-1202E-CB?qs=PZkJWYMe7ld%252BTcdUZg0YJQ%3D%3D

[10] https://github.com/schreinerman/Apollo3Blue_SoftwareExamples

[11] Texas Instrument, "TPS7A05 ", https://www.ti.com/lit/ds/symlink/tps7a05.pdf?HQS= dis-mous-null-mousermode-dsf-pf-null-wwe&ts=1623233199595&ref_url=https%253A%252F%252Fwww.mouser.co.il%252F

[12] https://matplotlib.org/stable/api/index

[13] https://pyserial.readthedocs.io/en/latest/shortintro.html