

SAE 3.ESE.01

Réalisation d'un oscilloscope numérique

DJERRAH Lounes & AGA Amel

Table des matières

Sujet.....	2
1. Études préliminaires	4
2. Schémas sur KiCad	5
3. Application physique	9
4. Partie numérique sur carte	12
4.1. La carte STM32	12
4.2. STM32CubeIDE	13
4.3. La programmation	14
4.3.1. Mode AC/DC	16
4.3.2. L'atténuation	21
4.3.3. Le Gain.....	23
4.3.4. Le code final.....	25
5. La programmation Python	26
5.1. Écrire sur le port avec STM32CubeIDE.....	27
5.2. Afficher le signal avec Python.....	28
5.3. Interface graphique	30
Conclusion.....	33

Sujet

Dans le cadre d'un projet pratique 'SAE' du cycle de notre deuxième année de génie électrique et informatique industrielle au sein de l'I.U.T de Créteil (UPEC), nous avons été amenés à travailler en groupe de deux pour réaliser un projet guidé par les professeurs Monsieur DJOUANI et Monsieur FRYZIEL.

Ce document constitue donc le rapport de réalisation de notre groupe (AGA Amel et DJERRAH Lounes) durant le déroulement de cette S.A.E.

L'objectif de ce sujet de SAE est de réaliser un oscilloscope numérique opérationnel à l'aide d'un cahier des charges fournis.

Nous devons réaliser cet oscilloscope en utilisant les composants mis à notre disposition et en les comparants à l'aide des datasheets fournis afin de choisir ceux qui correspondent le mieux au cahier des charges fournis.

L'objectif final sera d'afficher les résultats de l'oscilloscope sur un support visuel tel un écran en réalisant la programmation de la partie 'backend' sur la carte STM32 qui nous permettra de simuler les actions / réactions d'un oscilloscope. Puis nous devons également réaliser la programmation de la partie 'frontend' à l'aide du langage de programmation Python, la partie Python devra alors nous permettre d'afficher la visualisation de nos signaux en fonctions de la valeur du signal d'entrée et des modifications qu'on lui apport (gain, atténuation etc...).

La partie STM32 (backend) et la partie Python (frontend) sont donc intimement liés, une connexion entre les deux est donc nécessaire, pour cela les informations devront communiquer par liaison USB donc les spécificités seront programmées dans la partie STM32 puis récupérer dans la partie Python.

Le fonctionnement global de notre projet est expliqué par le schéma suivant.

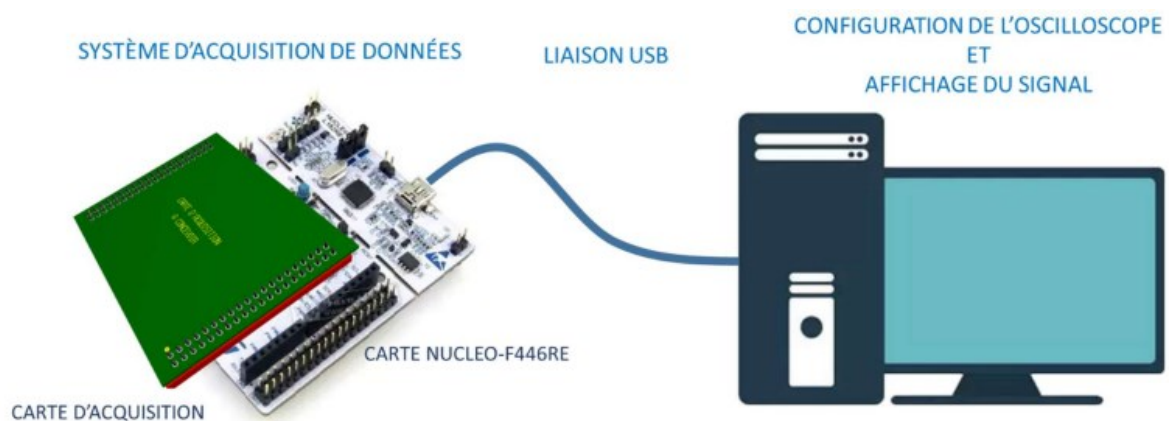


Schéma du fonctionnement de l'oscilloscope numérique

Cahier des charges :

Le cahier des charges constitue les informations spécifiques des composants mis à notre disposition et nécessaires à la réalisation de ce projet.

Ces informations on put être trouvé dans la documentations disponible sur Éprel.

1) Caractéristiques techniques de l'oscilloscope

- Nombre de voie : 1
- Impédance d'entrée : 1 M Ω
- Tension maximale d'entrée : 40 V crête à crête
- Sensibilité : 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5 V/div
- Fréquence d'échantillonnage maximale : 1 MHz
- Base de temps : 10, 20, 50, 100, 200, 500 μ s/div ; 1, 20, 50, 100, 200, 500 ms/div ; 1, 2, 5, 10 s/div
- Déclenchement (Trigger) : niveau et pente sélectionnable
- Affichage sur 8 divisions en vertical et 10 divisions en horizontal
- La carte d'acquisition sera alimentée en ± 5 V.
- Une alimentation interne de +3V3 sera à concevoir

2) Principales caractéristiques de la carte NUCLEO-F446RE

- Processeur ARM CORTEX-M4 cadencé à 180 MHz
- 512 Ko de mémoire flash
- 128 ko de SRAM
- Alimentation 5V via le port USB (externe possible)
- Régulateur 3V3 présent sur la carte
- Tension maximale sur les entrées sorties : 3V3
- Tension minimale sur les entrées sorties : 0V

1. Études préliminaires

Pour réaliser le projet de façon optimal, nous allons commencer à étudier la carte d'acquisitions en fonction des données fournis dans le cahier des charges afin de visualiser les paramètres sur lequel nous travaillerons.

Caractéristiques de l'oscilloscope :

Sensibilité (V/div)	Tension Vcc MAX	Amplification ou atténuation
0,01	0,08	41,25
0,02	0,16	20,63
0,05	0,4	8,25
0,1	0,8	4,13
0,2	1,6	2,06
0,5	4	0,83
1	8	0,41
2	16	0,21
5	40	0,08

- Après analyse, pour la réalisation de cet oscilloscope numérique nous utiliserons un amplificateur inverseur car celui présente de nombreux avantages pour notre projet :
1. **Réglage du Gain :** Sa capacité à offrir une flexibilité lors du réglage du gain. En ajustant simplement la valeur d'une résistance de rétroaction, il est possible d'obtenir des gains négatifs de manière plus simple et précise.
 2. **Inversion de Phase :** Sa capacité à générer une sortie inversée par rapport à l'entrée. Cette particularité s'avère bénéfique dans des applications où l'inversion de phase du signal est nécessaire.
 3. **Réduction du Bruit :** Avantage en termes de réduction du bruit par rapport à d'autres configurations. Cette caractéristique revêt une importance particulière dans des applications nécessitant une mesure précise, telles que celles propres à un oscilloscope.
 4. **Stabilité :** Les amplificateurs inverseurs se distinguent par leur stabilité accrue, notamment pour des gains élevés. Ce critère devient particulièrement significatif dans le cas où une précision élevée est requise sur une large plage de gains.

2. Schémas sur KiCad

Nous souhaitons maintenant commencer la réalisation de notre schéma électrique sur le logiciel de conception KiCad.

Pour cela nous devons d'abord déterminer les composants que nous utiliserons en les comparant avec les datasheets fournis afin de déterminer les plus adaptés.

Transistors :

Pour le transistor, après analyse nous avons choisis le **PN2222A** car c'était un *Thyristor* et pas un *Mofset*, en effet les Thyristor sont plus adaptés car ils évitent le phénomène d'arc électrique lors de son fonctionnement.

Relais :

Pour les relais, après analyse nous avons choisis le **HE721C0500** et le **HE3621510**.

Le HE721C0500 car il permet d'alimenter continuellement et de diriger le courant vers ou on veut ce qui permettra de créer l'atténuation de 1 ou l'atténuation de 1/10.

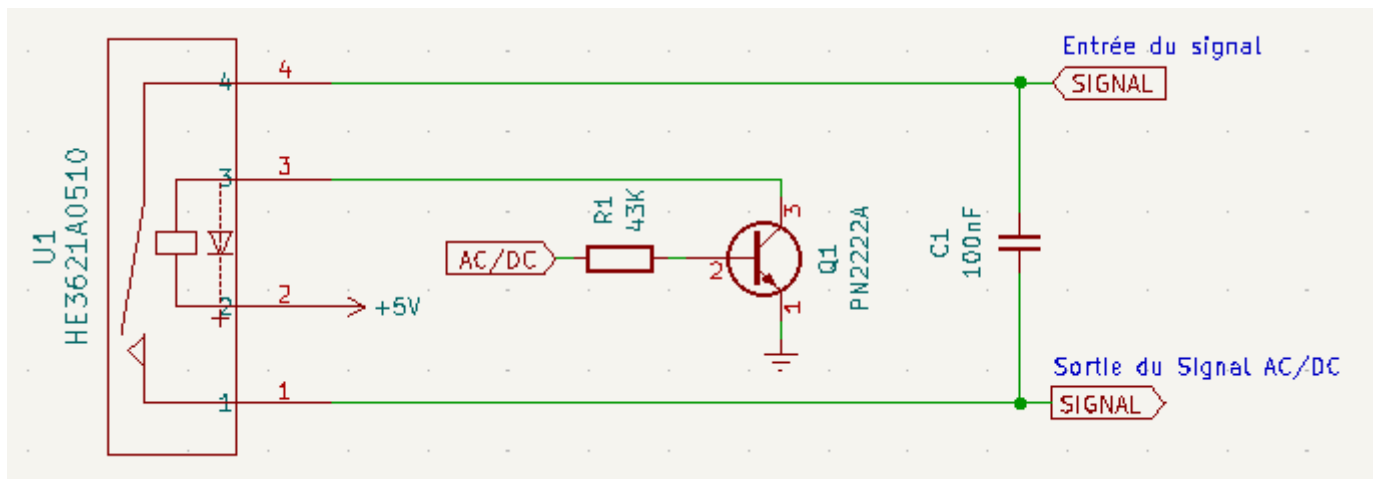
Le HE3621510 car il permet de choisir soit d'ouvrir le circuit ou soit de le fermé ce qui nous permettra de créer l'option *mode AC/DC*.

AOP :

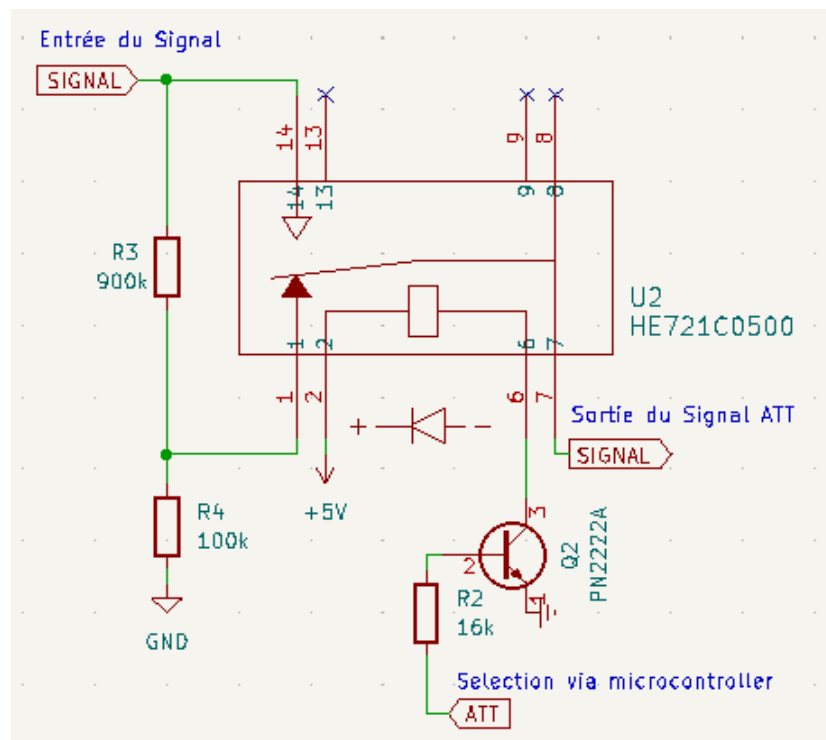
Pour l'AOP, après analyse nous avons choisis le **TLV272CD**.

Application : Maintenant, après avoir trouvé les empreintes nécessaires sur internet et après les avoir configurés sur notre session, nous allons créer les schémas sur le logiciel Kicad correspondant.

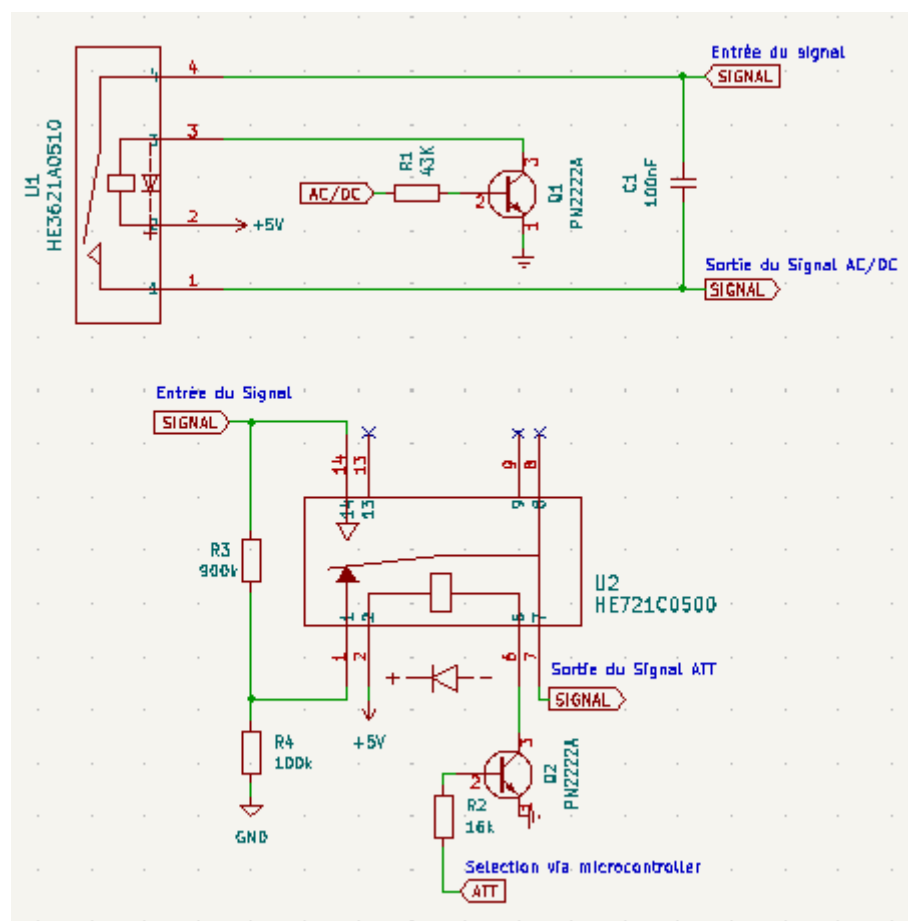
- Schéma pour les modes AC/DC :



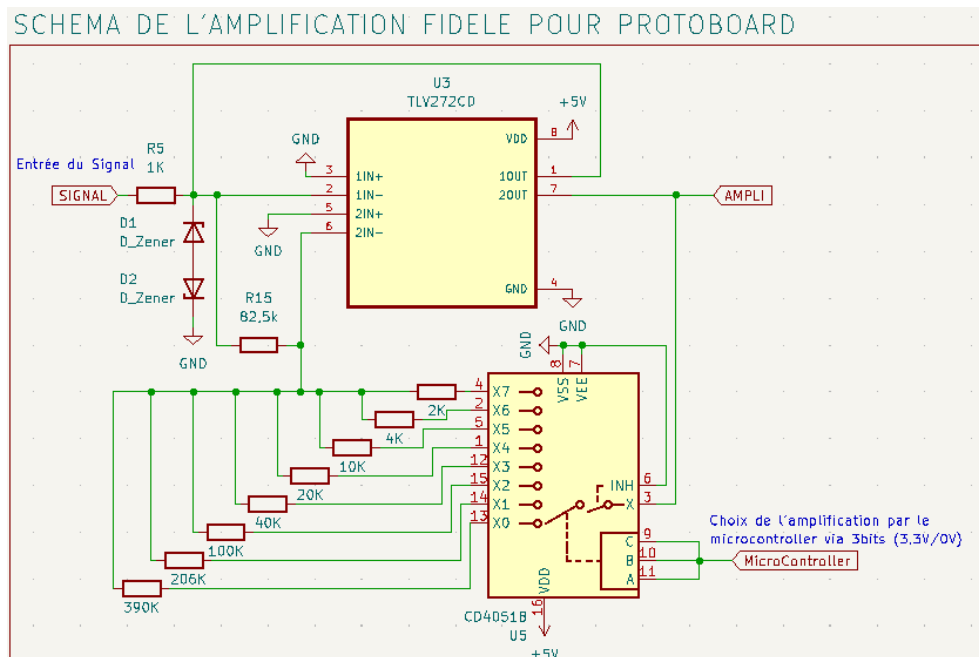
- Schéma pour l'atténuation :



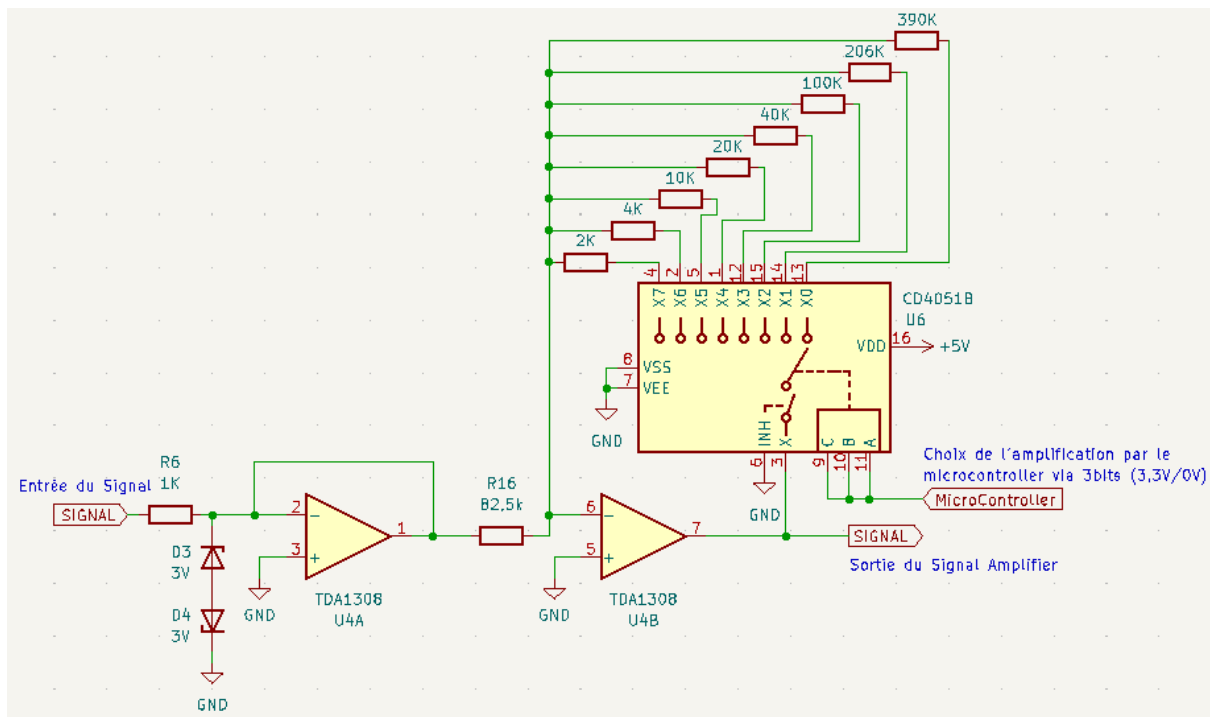
Nous mettons les 2 schémas sur un même fichier KiCad et les relient par des liaisons *label* :



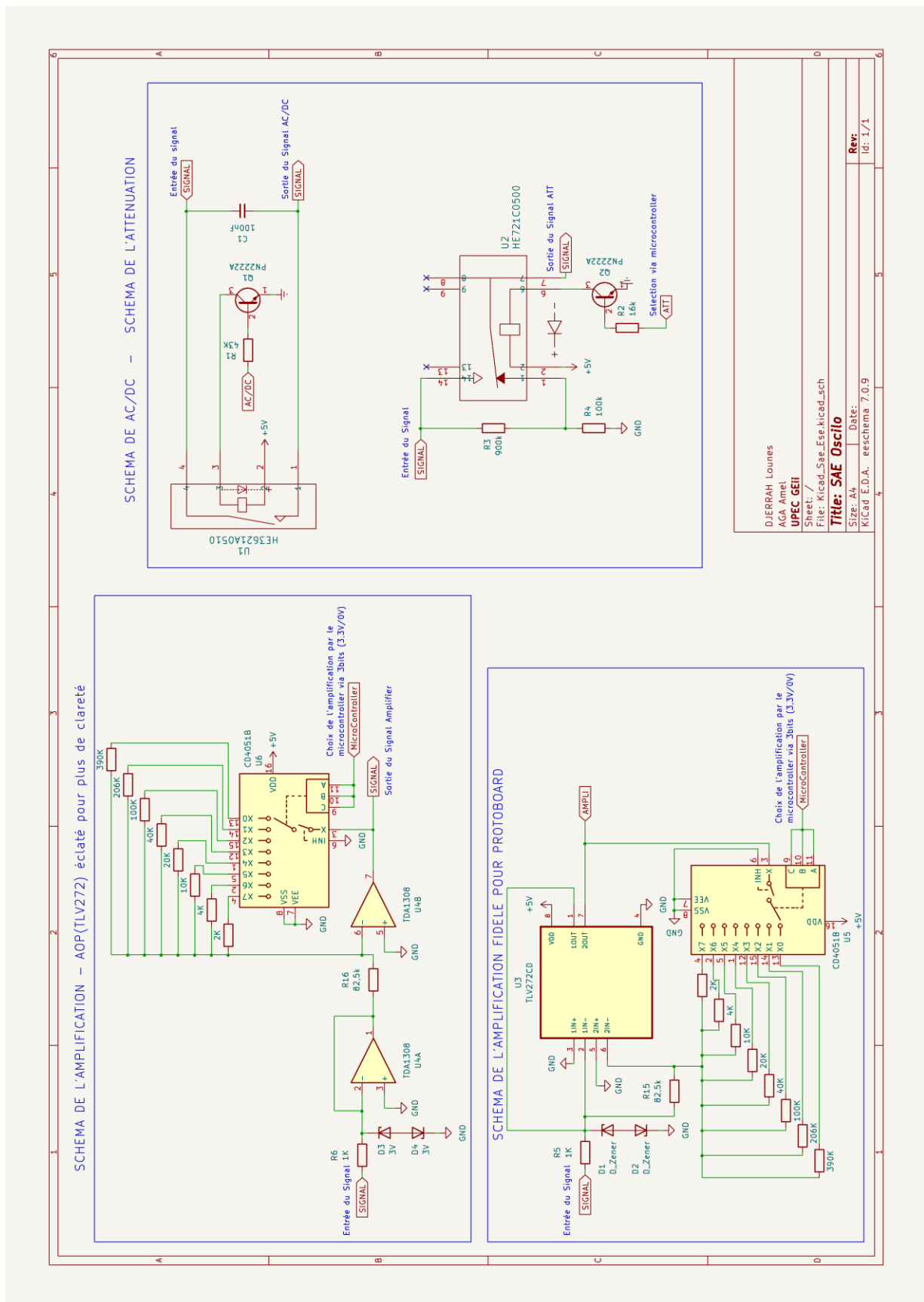
Nous configurons maintenant l'amplificateur opérationnel ainsi que le multiplexeur qui vont nous servir à régler le gain de notre oscilloscope numérique. Voici le schéma avec les liaisons des broches fidèles à la réalité afin de pouvoir nous appuyer de celui-ci pour notre montage protoboard.



Ici le montage avec le TLV272 éclaté, grâce à ce schéma on peut distinguer plus clairement :
L'adaptation d'impédance → La protection de surtension grâce aux diodes Zener 3V →
Montage suiveur → Montage Amplificateur inverseur avec son multiplexeur.



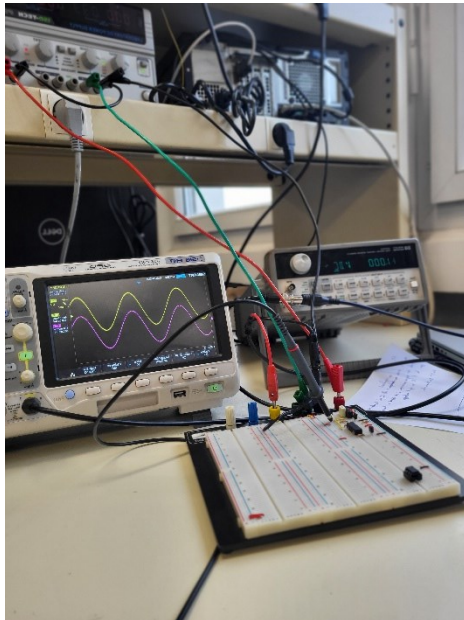
Voici donc ci-dessous notre schéma KiCad complet.



3. Application physique

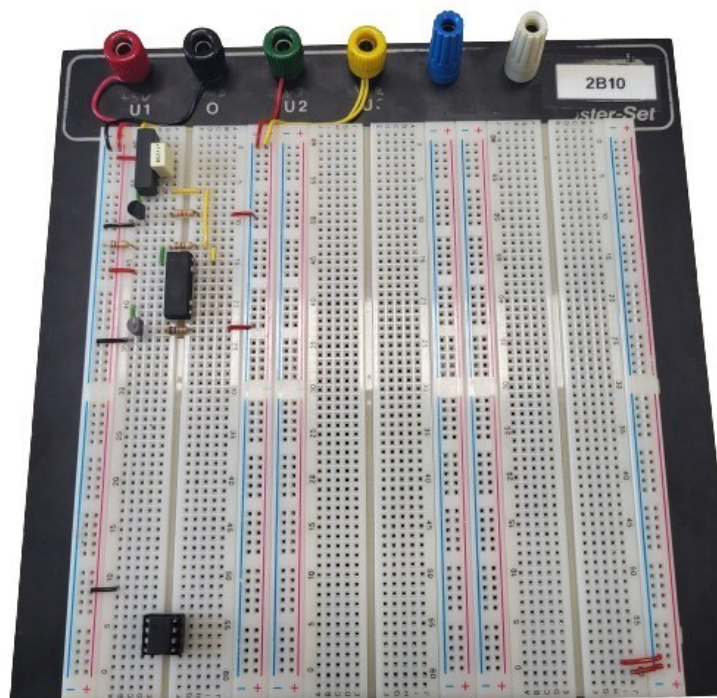
Nous allons à présent reproduire le schéma KiCad sur une plaque protoboard et tester les différents modes.

Pour pouvoir tester ces modes il est nécessaire d'utiliser une sonde que l'on connecte à la broche de sortie du composant nécessaire ainsi que sur un oscilloscope de l'I.U.T pour pouvoir visualiser les signaux de sortie.

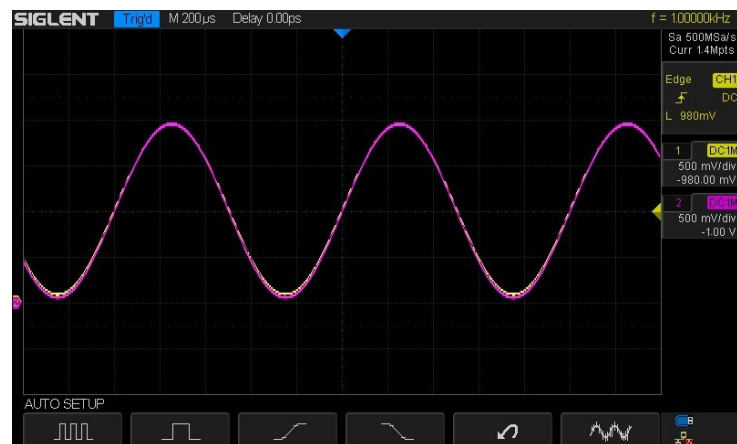
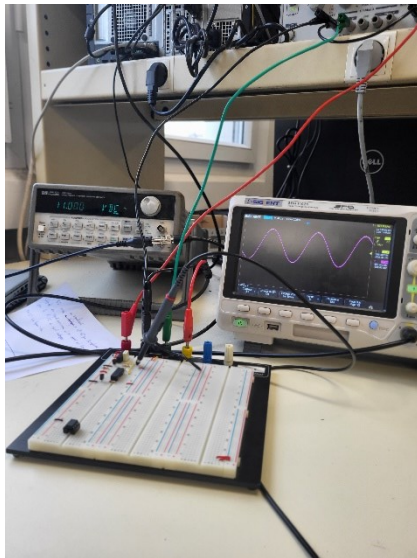


Photos représentatives de l'affichage des signaux

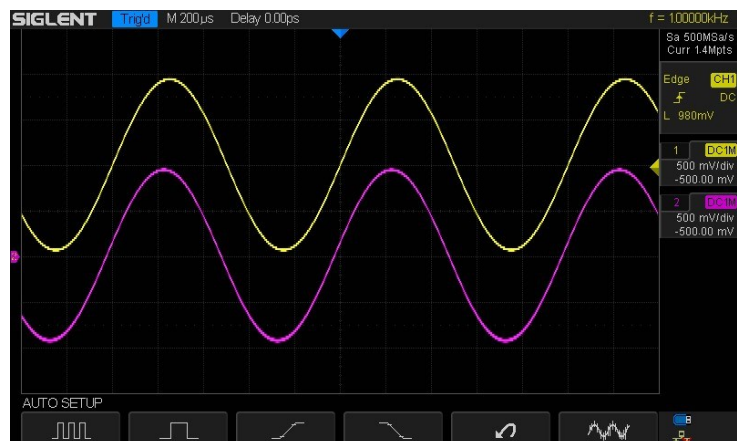
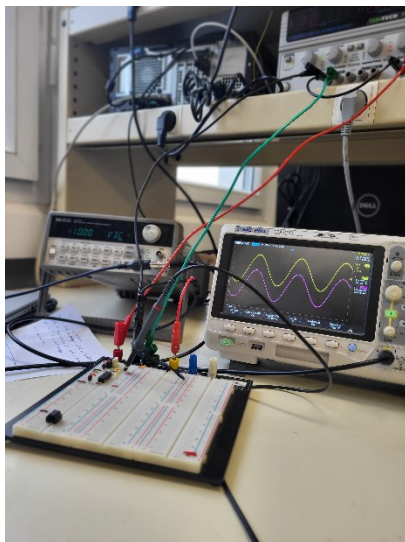
- Montage :



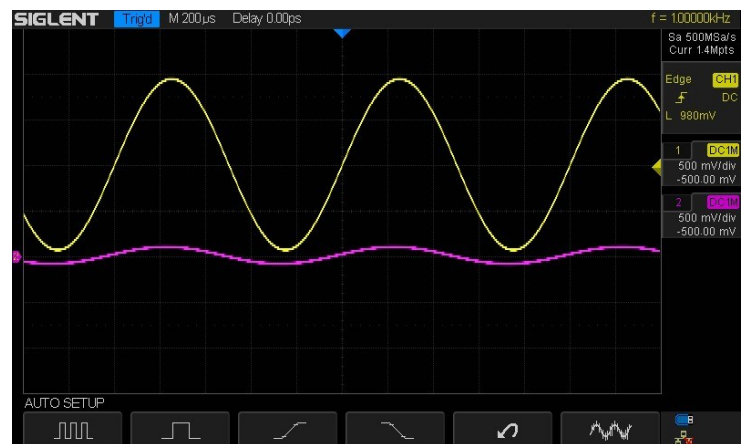
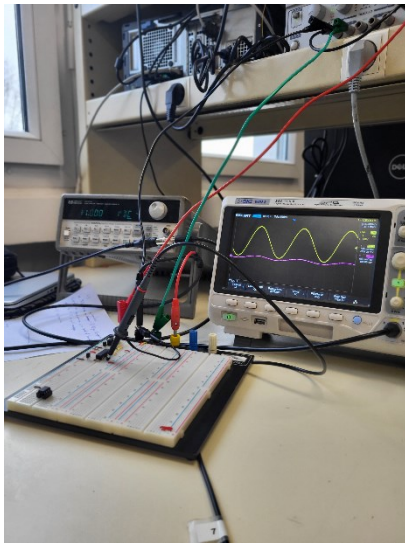
Lorsque les signaux sont confondus (mode DC)



Lorsque les signaux ne sont pas confondus (mode AC)



Lorsqu'il y a l'atténuation



4. Partie numérique sur carte

La partie d'application physique et électrique étant terminée avec des résultats concluants, il est maintenant tant de commencer la programmation numérique de notre oscilloscope.

L'objectif va être d'obtenir des résultats semblables aux signaux que l'on a pu observer grâce aux montages électriques sur notre protoboard.

Pour cela nous allons utiliser les cartes STM32 ainsi que le logiciel STM32CubeIDE mis à notre disposition.

4.1. La carte STM32

La carte que nous utiliserons est plus précisément la **STM32L476RGTxLQFP64**.

C'est une carte développée par la société STMicroelectronics basée en Suisse et en France spécialisée dans la fabrication de semi-conducteurs et fabrique mondiale de microcontrôleurs.

Sa gamme STM32 est particulièrement populaire dans le domaine des systèmes embarqués, de l'IoT, de l'automatisation industrielle et d'autres domaines où une faible consommation d'énergie et une haute performance sont essentielles.

Le nom attribué à notre carte nous donne directement beaucoup d'informations importantes sur celle-ci.

- **STM32** : C'est le préfixe qui identifie la famille de microcontrôleurs de STMicroelectronics. Dans ce cas, "STM32" indique qu'il s'agit d'un microcontrôleur de la série STM32.
- **L476** : Ce code désigne le modèle spécifique du microcontrôleur STM32 utilisé sur la carte. Dans ce cas, "L476" indique qu'il s'agit du modèle STM32L476RG.
 - **L** : Indique la famille du microcontrôleur, ici la famille "L" qui se concentre sur la basse consommation d'énergie.
 - **476** : Désigne la série spécifique du microcontrôleur. Dans ce cas, il s'agit de la série STM32L4.
- **LQFP64** : Cela décrit le type d'encapsulation du microcontrôleur. Dans ce cas, "LQFP64" indique que le microcontrôleur est encapsulé dans un boîtier LQFP (Low-profile Quad Flat Package) comportant 64 broches.

Ce microcontrôleur appartient donc à la famille STM32L4, qui est connue pour sa faible consommation d'énergie et ses performances élevées. Grâce à la documentation constructeurs disponible en bas de page on peut déduire d'autres informations intéressantes sur la carte.

- La carte possède un cœur ARM Cortex-M4 cadencé jusqu'à 80 MHz.
- La carte possède 1 Mo de mémoire flash et 128 Ko de SRAM.
- La carte possède différentes interfaces de communication intégrées, y compris **UART**, SPI, I2C, **CAN**, USB, etc.
- La carte possède de multiples timers et compteurs pour les applications de chronométrage et de contrôle.
- La carte possède un convertisseur analogique-numérique (CAN) et numérique-analogique (CNA) intégrés.

Datasheets : [Documentation constructeur STM32L476xx](#)

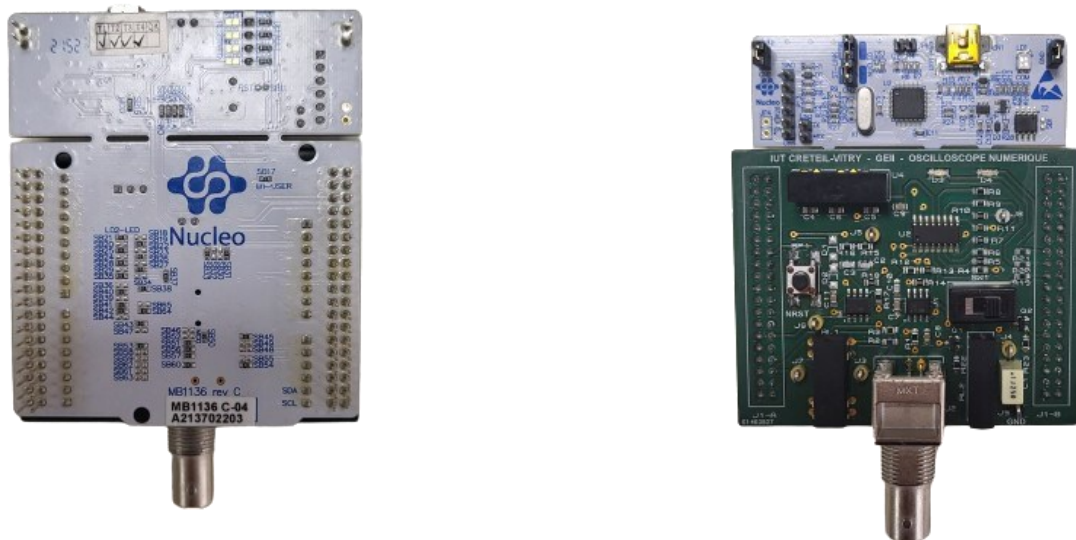


Photo recto/verso de la carte que nous utilisons en projet

On remarque grâce à ces photos qu'une autre carte a été rajouté sur carte STM32, cette nouvelle carte a été rajouté par les professeurs et n'est là que pour faciliter le processus en ajoutant un port connecteur BNC qui va nous permettre de lire les signaux de sortie de notre carte directement sur un oscilloscope de l'I.U.T.

4.2. STM32CubeIDE

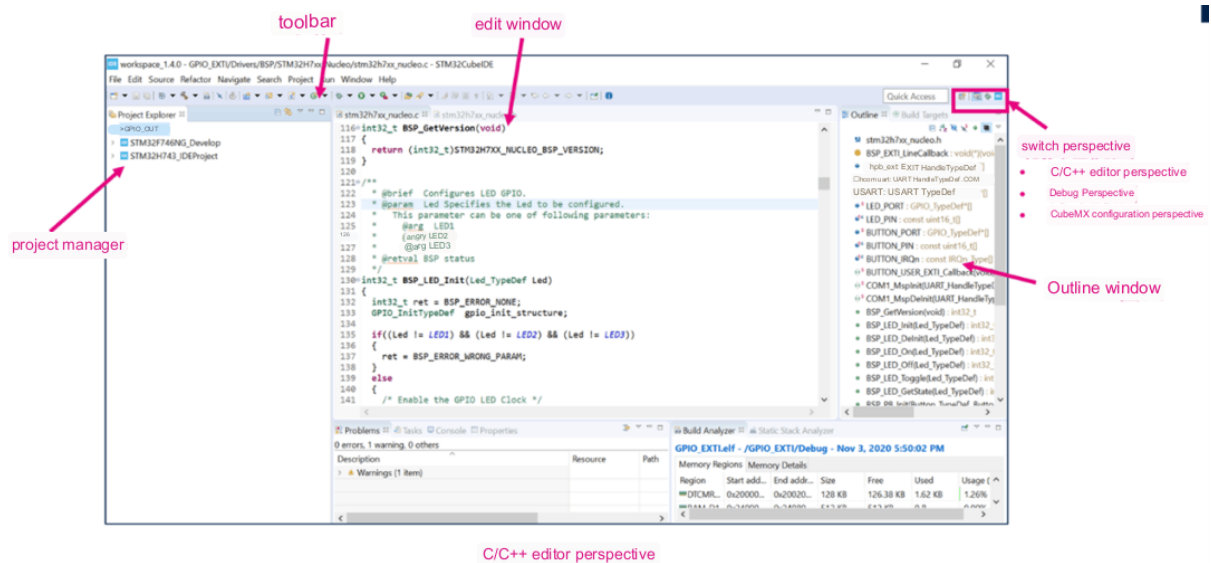


Image d'illustration de l'IDE

STM32CubeIDE est un environnement de développement intégré (IDE) développé par STMicroelectronics pour la programmation des microcontrôleurs de la série STM32.

Il est basé sur Eclipse ce qui lui confère une interface de développement intuitive et qui ressemble à celles d'autres IDE que l'on a pu utiliser tel que *Vivaldo*.

L'outil STM32CubeMX est déjà intégré ce qui permet de facilement générer du code en fonction des broches que l'on a configuré (IOC).

C'est donc avec cette IDE que nous allons pouvoir configurer tous les éléments nécessaires afin de produire un oscilloscope numérique.

Il est à noter que nous avons pu bénéficier de travaux dirigés et de travaux pratiques qui nous ont aidés à comprendre et maîtriser l'IDE ainsi que la carte afin de pouvoir réaliser cette SAE dans les meilleures conditions.

4.3. La programmation

Nous allons commencer à créer notre projet d'oscilloscope numérique en configurant d'abord les différents paramètres de création de projet (nom du projet, Workspace etc...).

Cette partie d'initialisation terminée nous arrivons maintenant à l'IOC.

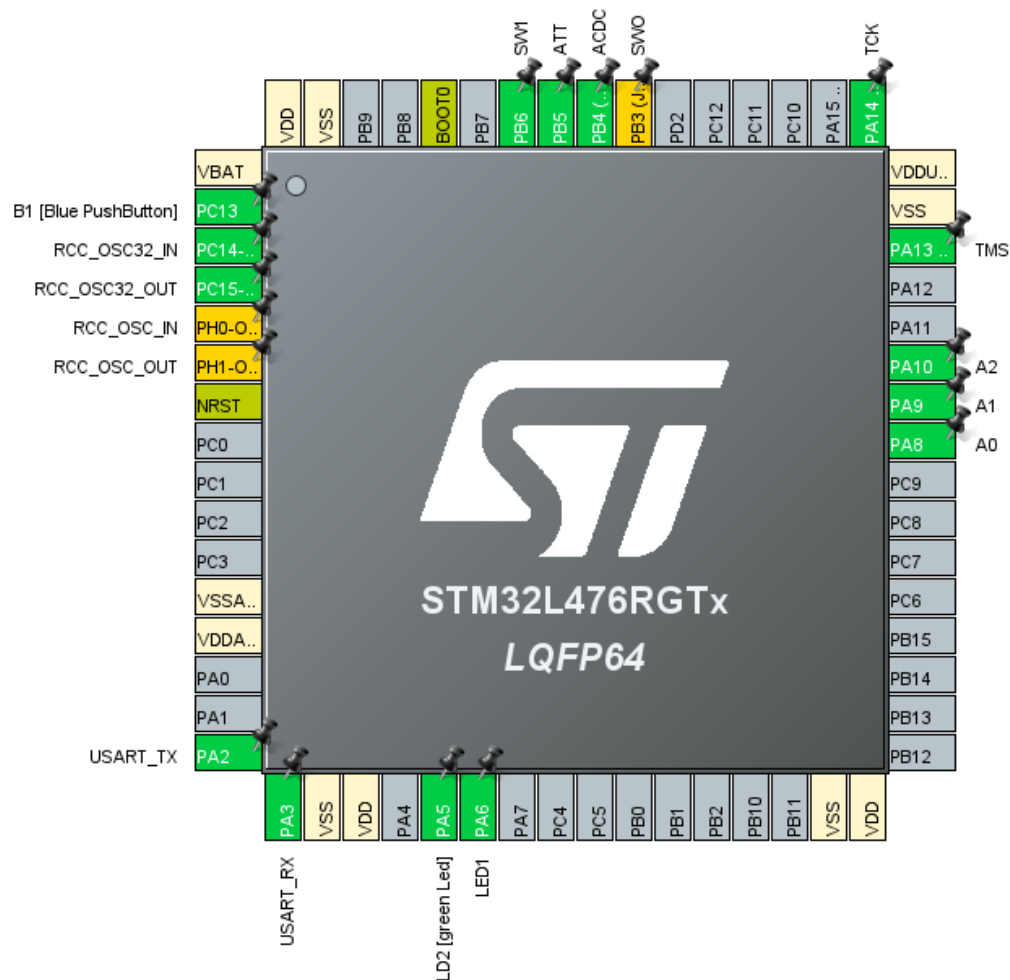
L'IOC est un fichier qui sera créé dans le répertoire de notre fichier (donc un .ioc), ce fichier sera le fichier de configurations des différentes broches de notre carte.

En effet dans cette partie il est nécessaire de désigner quelles broches nous utilisons, de préciser si c'est des entrées (in), des sorties (out) ou même des entrées-sorties (inout).

Pour cela, une interface graphique permet de se repérer plus facilement et affiche directement tous les broches de notre carte avec leurs configurations.

Une fois la configuration terminée nous pouvons générer notre code qui sera créé en fonction des agencements que l'on a fait sur l'interface graphique IOC.

Il est également à noter que pour générer le code il faut être connecté à son compte STM32 sur l'IDE.



Configuration de notre .ioc

Nous retrouvons ci-dessus la configuration de notre fichier IOC et pouvons clairement constater toutes les broches que nous avons configuré ainsi que leurs labels respectifs permettant de reconnaître leur utilité.

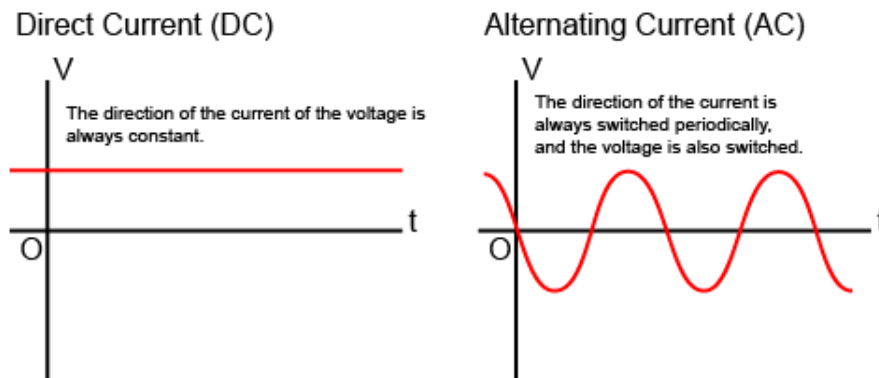
Il faut aussi noter que cette configuration peut être modifié à tout moment et n'est pas définitif.

4.3.1. Mode AC/DC

L'objectif de ce code est de pouvoir choisir entre le mode AC et le mode DC, c'est-à-dire plus concrètement lorsqu'une action d'entrée est réalisée (par exemple appuyé sur un bouton) ça change le mode de notre oscilloscope numérique soit en mode Alternative Curent (courant alternative) ou bien en Direct Curent (courant continu).

Pour rappel, voici les définitions des mode AC et DC :

- **Courant Alternative (AC)** : En mode AC, l'oscilloscope affiche uniquement les variations rapides du signal, en ignorant les parties constantes comme les niveaux de décalage.
- **Courant Continu (DC)** : En mode DC, l'oscilloscope affiche tout le signal, y compris les parties constantes et les variations rapides, sans aucun filtrage.



Exemples représentatifs des mode AC et DC

Dans notre code sur STM32CubeIDE nous allons donc initialiser une LED et une SWITCH, la SWITCH pourra grâce à son état (1 ou 0) activé ou non le mode AC et la LED nous donner l'information de si le mode AC est allumé ou non (lumière = mode AC).

```
/* USER CODE BEGIN PV */
int ACDC = 0;
int A0 = 0;
int A1 = 0;
int A2 = 0;
int LED1 = 0;
int SW1 = 0;
char str[7];
/* USER CODE END PV */

while (1)
{
    SW1 = HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin);
    ACDC = HAL_GPIO_ReadPin(ACDC_GPIO_Port, ACDC_Pin);
    LED1=HAL_GPIO_ReadPin(LED1_GPIO_Port, LED1_Pin);
    sprintf(str,"%d \n\t",ACDC);
    HAL_UART_Transmit(&huart2, (uint8_t *)str, 7, 100);

    if (SW1 ==0)
    {
        HAL_GPIO_WritePin(ACDC_GPIO_Port, ACDC_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
    }
    else
    {
        HAL_GPIO_WritePin(ACDC_GPIO_Port, ACDC_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
    }
}
/* USER CODE END WHILE */

/* USER CODE BEGIN 3 */
}
/* USER CODE END 3 */
}
```

Code que nous avons utilisé

Grâce à ce programme nous avons pu réaliser l'objectif de cette partie, en effet grâce aux fonctions de conditions (*if* et *else*) dans la boucle '*while*' on peut activer ou non le mode AC et DC.

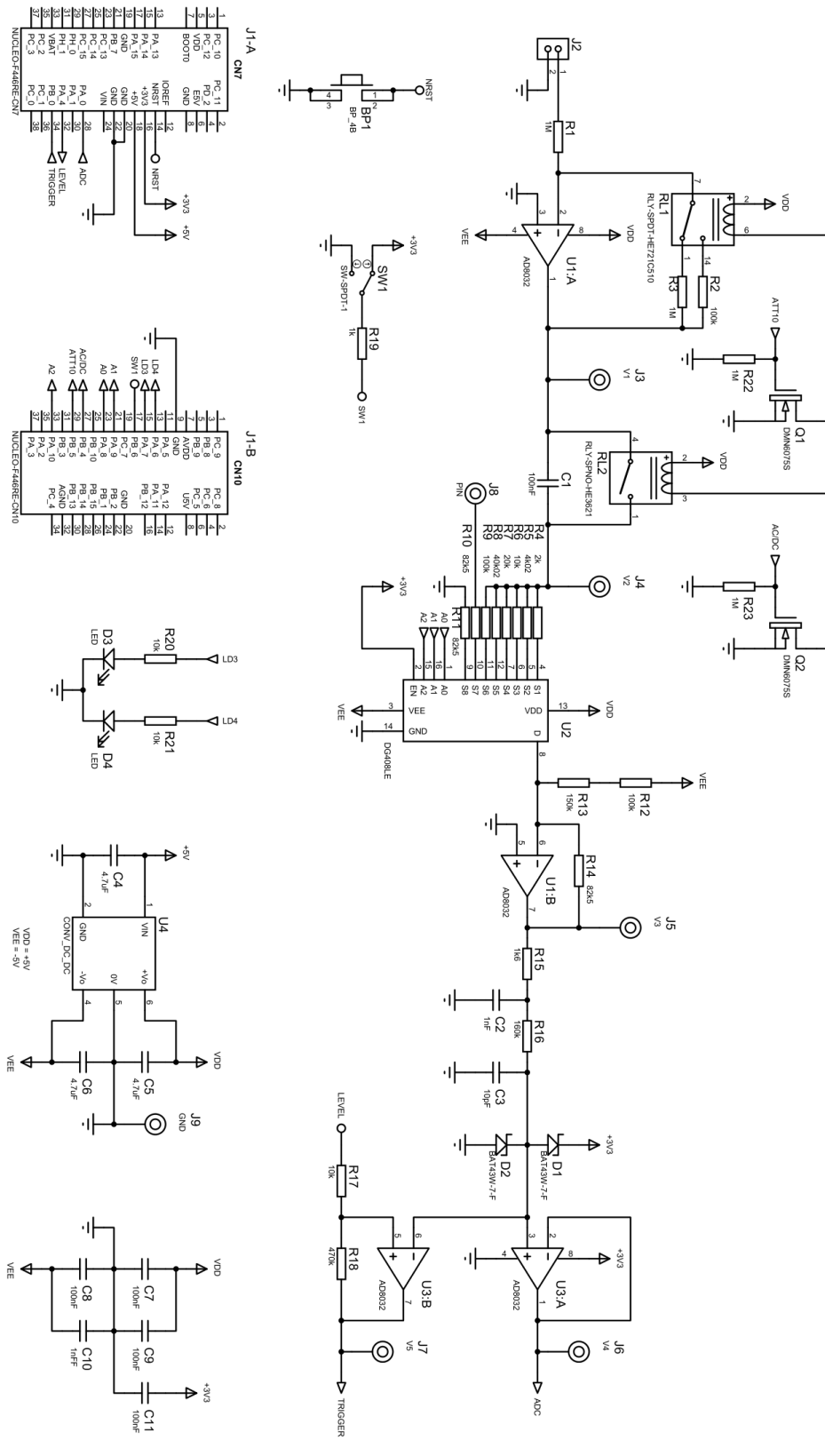
Pour pouvoir tester que le programme fonctionne correctement il va falloir que l'on teste directement sur notre carte les signaux de sortie présent sur des broches spécifiques à l'aide d'une sonde et les visualiser sur un oscilloscope de l'I.U.T.



Représentation de la broche de sortie du mode AC/DC

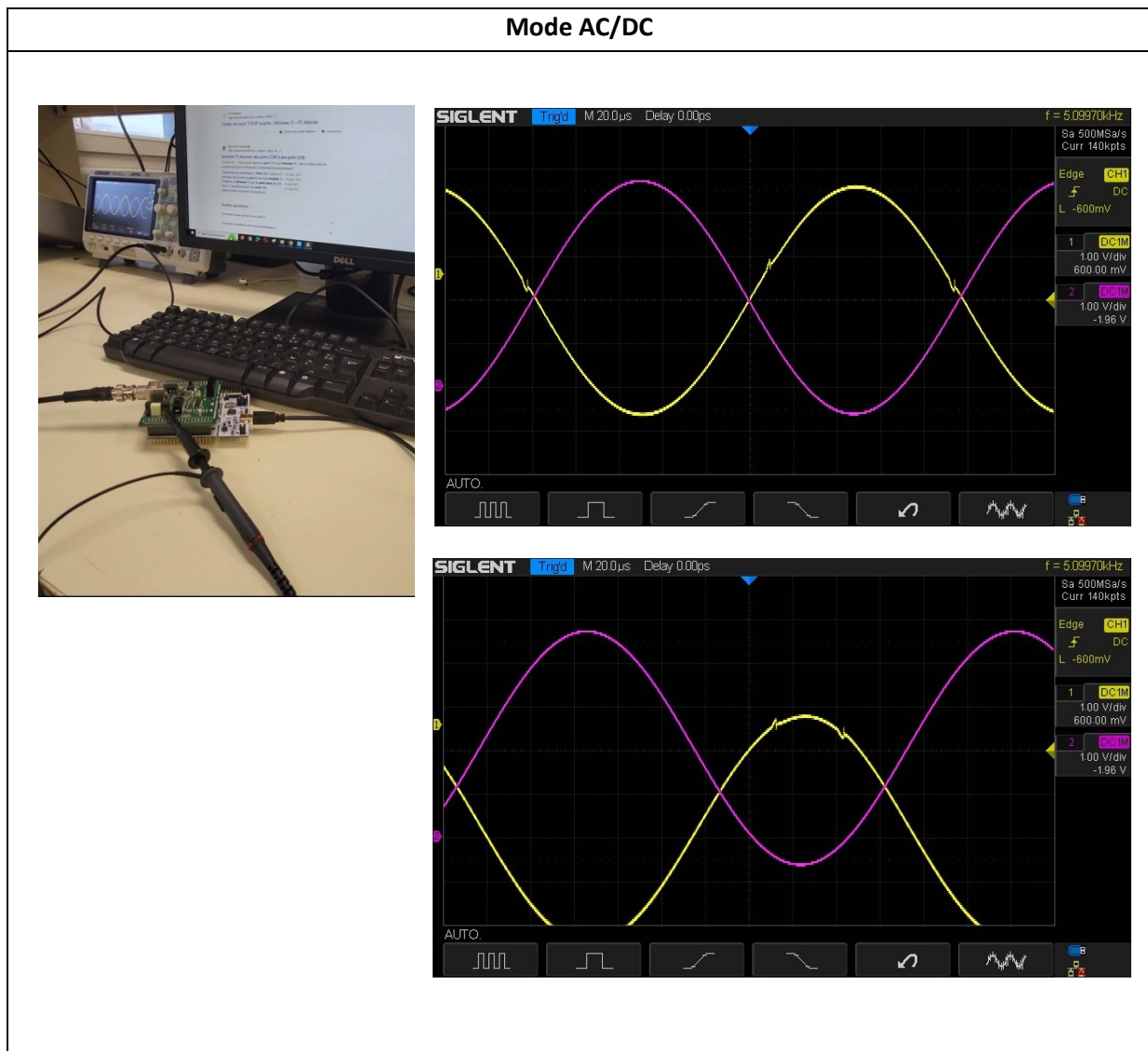
Comme on peut le constater sur l'image ci-dessus la broche à laquelle nous allons devoir connecter notre sonde est la broche de sortie '**J4**', cette broche correspond à la sortie du mode AC/DC, donc pour pouvoir visualiser les modifications de notre signal lorsque que l'on change le mode AC ou DC il faut connecter notre sonde sur cette broche et sur l'oscilloscope.

Nous avons pu déterminer que c'était la broche J4 qui correspondait aux modes AC/DC grâce à la documentation disponible sur Éprel suivante :



OSCILLOSCOPE CMS

Nous branchons alors notre sonde sur cette broche et sur l'oscilloscope et constatons les résultats suivants :



Nous remarquons une différence entre les 2 graphiques ce qui marque le passage de mode.

4.3.2. L'atténuation

Le code correspondant à l'atténuation est le suivant.

```
int ACDC = 0;
int ATT = 0;
int A0 = 0;
int A1 = 0;
int A2 = 0;
int LED1 = 0;
int SW1 = 0;

while (1)
{
    SW1 = HAL_GPIO_ReadPin(SW1_GPIO_Port, SW1_Pin);
    //ACDC = HAL_GPIO_ReadPin(ACDC_GPIO_Port, ACDC_Pin);
    LED1 = HAL_GPIO_ReadPin(LED1_GPIO_Port, LED1_Pin);
    ATT = HAL_GPIO_ReadPin(ATT_GPIO_Port, ATT_Pin);

    if (SW1 ==0)
    {
        // HAL_GPIO_WritePin(ACDC_GPIO_Port, ACDC_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_SET);
        HAL_GPIO_WritePin(ATT_GPIO_Port, ATT_Pin, GPIO_PIN_SET);
    }
    else
    {
        // HAL_GPIO_WritePin(ACDC_GPIO_Port, ACDC_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(LED1_GPIO_Port, LED1_Pin, GPIO_PIN_RESET);
        HAL_GPIO_WritePin(ATT_GPIO_Port, ATT_Pin, GPIO_PIN_RESET);
    }
}
```

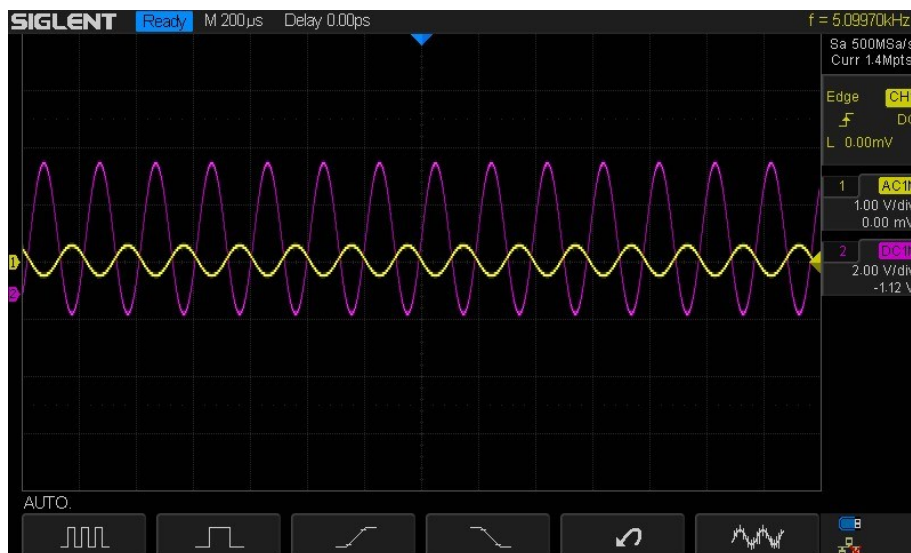
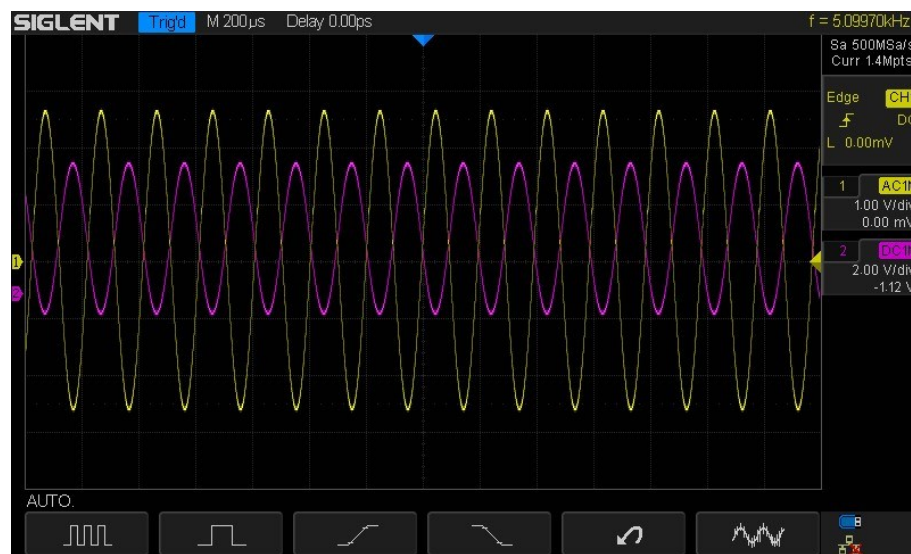
A noter qu'il se trouve dans la boucle *while* comme le mode AC/DC.

La broche de sortie correspondante au phénomène d'atténuation est la broche '**J3**', nous nous positionnons donc sur cette broche afin de visualiser nos signaux.



Représentation de la broche de sortie du mode ATT

Mode ATT



4.3.3. Le Gain

Pour réaliser la représentation de la sélection du Gain sur notre carte STM32 nous allons devoir

Devoir déclarer différentes variables correspondantes aux différentes valeurs du gain que nous avons pu voir au début de ce rapport.

Pour pouvoir visualiser la valeur de sortie et observer les variations avec les changements du gain nous devons brancher notre sonde sur la broche de sortie 'J5'.



Voici, ci-dessous, le code que nous utiliserons pour observer le gain sur l'oscilloscope.


```
switch(A)
{
    case 1 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, SET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, RESET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, RESET);
        break;

    case 2 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, RESET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, SET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, RESET);
        break;

    case 3 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, SET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, SET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, RESET);
        break;

    case 4 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, RESET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, RESET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, SET);
        break;

    case 5 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, SET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, RESET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, SET);
        break;

    case 6 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, RESET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, SET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, SET);
        break;

    case 7 :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, SET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, SET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, SET);
        break;

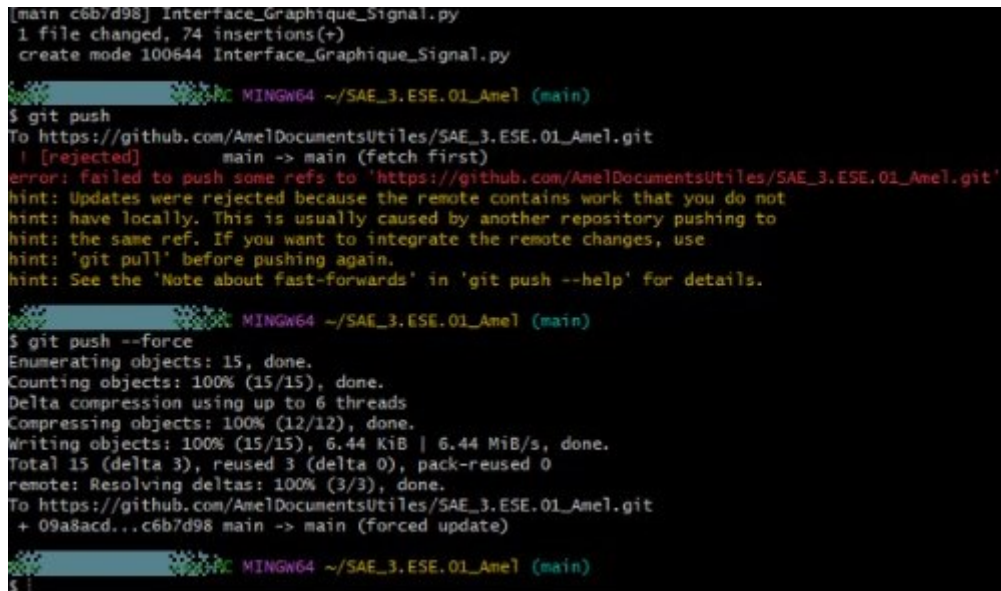
    default :
        HAL_GPIO_WritePin(A0_GPIO_Port, A0_Pin, RESET);
        HAL_GPIO_WritePin(A1_GPIO_Port, A1_Pin, RESET);
        HAL_GPIO_WritePin(A2_GPIO_Port, A2_Pin, RESET);
}
```

Malheureusement nous n'avons pas pu réaliser des essais directement avec la carte et un oscilloscope spécialement pour ce cas de figure.

4.3.4. Le code final

Le code final sur STM32CubeIDE faisant 379 lignes il ne serait pas pratique de l'afficher dans le rapport par la méthode de captures d'écrans, c'est pourquoi je vous invite à consulter le répertoire GitHub suivant que nous avons spécialement créer pour ce projet.

Vous pouvez donc retrouver le code en entier ici : [Code STM32 GitHub](#)



```
[main c6b7d98] Interface_Graphique_Signal.py
1 file changed, 74 insertions(+)
create mode 100644 Interface_Graphique_Signal.py

MINGW64 ~/SAE_3.ESE.01_Amel (main)
$ git push
To https://github.com/AmelDocumentsUtile/SAE_3.ESE.01_Amel.git
 ! [rejected]        main -> main (fetch first)
error: failed to push some refs to 'https://github.com/AmelDocumentsUtile/SAE_3.ESE.01_Amel.git'
hint: Updates were rejected because the remote contains work that you do not
hint: have locally. This is usually caused by another repository pushing to
hint: the same ref. If you want to integrate the remote changes, use
hint: 'git pull' before pushing again.
hint: See the 'Note about fast-forwards' in 'git push --help' for details.

MINGW64 ~/SAE_3.ESE.01_Amel (main)
$ git push --force
Enumerating objects: 15, done.
Counting objects: 100% (15/15), done.
Delta compression using up to 6 threads
Compressing objects: 100% (12/12), done.
Writing objects: 100% (15/15), 6.44 KiB | 6.44 MiB/s, done.
Total 15 (delta 3), reused 3 (delta 0), pack-reused 0
remote: Resolving deltas: 100% (3/3), done.
To https://github.com/AmelDocumentsUtile/SAE_3.ESE.01_Amel.git
+ 09a8acd...c6b7d98 main -> main (forced update)

MINGW64 ~/SAE_3.ESE.01_Amel (main)
$
```

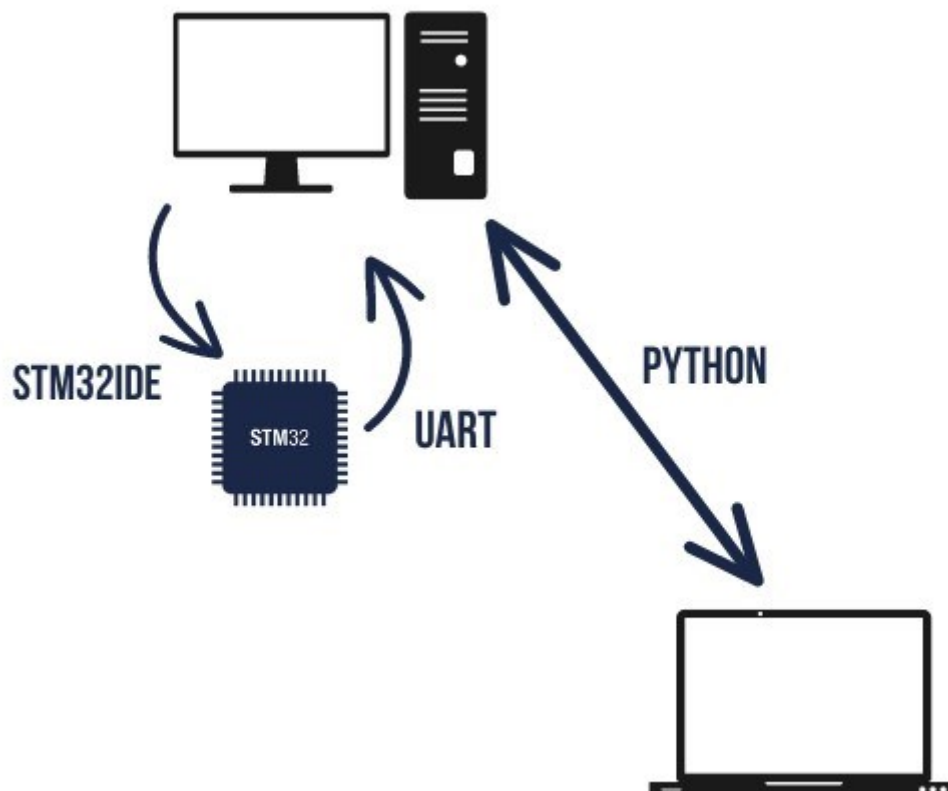
Illustration de publication de fichier dans GitHub

5. La programmation Python

La programmation Python a pour objectif d'afficher en direct le signal de sortie de notre carte STM32 sur un support visuelle tel qu'un écran d'ordinateur.

La particularité de cette programmation va être que le programme Python va devoir communiquer en direct avec notre carte et pour se faire il nous est nécessaire de commencer à rajouter quelques lignes sur notre code STM32 qui nous permettront de pouvoir communiquer les valeurs de sortie de la carte.

L'organisation peut être définie comme ci-dessous.



Description du fonctionnement

5.1. Écrire sur le port avec STM32CubeIDE

Pour se faire, nous utiliserons la méthode UART, nous allons communiquer les valeurs de la carte sur un port de l'ordinateur ou le code STM est simulé puis récupérer les valeurs disponibles dans ce port grâce à un câble USB branché sur un autre PC qui lui fait tourner le code en Python.

Pour commencer, nous allons devoir convertir nos valeurs de sortie, en effet, tel qu'elles sont maintenant ses valeurs ne sont pas exploitables convenablement car elles sont analogiques, nous allons donc utiliser un CAN (convertisseur analogique – numérique) déjà disponible sur la carte STM32 et donc simple d'utilisation.

Ensuite nous utiliserons la communication UART pour écrire sur les ports du PC (cette communication est déjà également disponible sur STM32CubeIDE).

Voici donc le code partie STM qui nous permet de faire cela :

```
while (1)
{
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
    HAL_ADC_Start(&hadc1);
    HAL_ADC_PollForConversion(&hadc1, HAL_MAX_DELAY);
    uint32_t adcValue = HAL_ADC_GetValue(&hadc1);
    float voltageData = adcValue * (3.3 / 4095); // Convert ADC value to voltage
    char uartData[50];
    sprintf(uartData, "ADC Value: %lu\r\nVoltage: %.2f V\r\n", adcValue,
voltageData);
    HAL_UART_Transmit(&huart2, (uint8_t*)uartData, strlen(uartData),
HAL_MAX_DELAY);
    HAL_Delay(1000); // Delay for 1 second
}
```

Nous pouvons observer si cela fonctionne correctement grâce au programme d'émulation de terminal disponible *TeraTerm*.

Il faut configurer sur TeraTerm sur quel port nous voulons lire (on trouve le port en allant dans les paramètres de l'ordinateur et en regardant quel port est branché par USB), par exemple le port COM3.

Il faut également modifier la vitesse de transmission dans la liaison série correspondante au nombre de bit transmis par seconde (baudrate), nous avons déterminé que notre valeur de baudrate est de 115200 bits/seconde c'est donc la valeur que nous mettons dans les paramètres de TeraTerm.

Enfin nous pouvons maintenant observer en direct la valeur de notre signal.

Vous pouvez constater sur la vidéo suivante que l'affichage avec la conversion analogique numérique fonctionne correctement.

Vidéo YouTube : [TeraTerm CAN](#)

5.2. Afficher le signal avec Python

A présent nous allons devoir écrire un code Python (nous utiliserons l'IDE de Microsoft Visual Studio Code) afin d'afficher le signal en fonction des valeurs de sortie émis par la carte STM32 que nous avons pu afficher sur TeraTerm (d'où l'importance d'avoir convertie ces valeurs analogiques en valeur numérique maintenant exploitable et explicite).

La première chose à faire est de choisir avec quelles bibliothèques nous allons travailler afin de mener ce projet à la réussite.

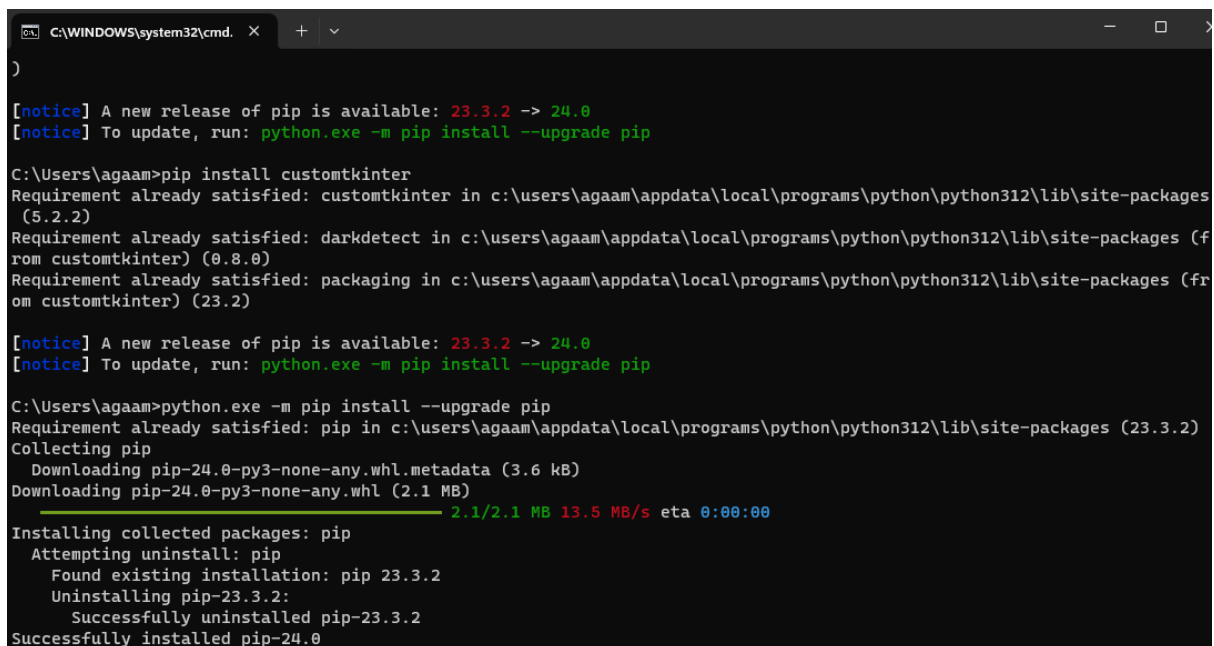
Il nous faut donc pour commencer une bibliothèque qui nous permettrait de lire les valeurs sur un port série pour pouvoir les implémenter en direct sur notre code Python, ces valeurs sont bien sur les valeurs de sorties de notre carte STM32.

Après plusieurs tentatives et plusieurs échecs, la meilleure bibliothèque que nous avons trouvée pour faire cela est la bibliothèque *PySerial* (à noter que pour l'inclure dans le code il faut écrire *include Serial* et non *include PySerial*).

Ensuite pour afficher le signal de façon graphique comme sur un vrai oscilloscope il nous faut une bibliothèque qui puisse convertir les valeurs de sorties récupérées en un graphique.

Pour cela, encore une fois après l'essai de nombreuses bibliothèques, la bibliothèque la plus adaptée que nous avons trouvé est la bibliothèque *Matplotlib*.

A noter que pour utiliser des bibliothèques Python il faut d'abord les installer via le Shell, comme montré ci-dessous.



```
C:\WINDOWS\system32\cmd. X + v
)

[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\agaam>pip install customtkinter
Requirement already satisfied: customtkinter in c:\users\agaam\appdata\local\programs\python\python312\lib\site-packages (5.2.2)
Requirement already satisfied: darkdetect in c:\users\agaam\appdata\local\programs\python\python312\lib\site-packages (from customtkinter) (0.8.0)
Requirement already satisfied: packaging in c:\users\agaam\appdata\local\programs\python\python312\lib\site-packages (from customtkinter) (23.2)

[notice] A new release of pip is available: 23.3.2 -> 24.0
[notice] To update, run: python.exe -m pip install --upgrade pip

C:\Users\agaam>python.exe -m pip install --upgrade pip
Requirement already satisfied: pip in c:\users\agaam\appdata\local\programs\python\python312\lib\site-packages (23.3.2)
Collecting pip
  Downloading pip-24.0-py3-none-any.whl.metadata (3.6 kB)
  Downloading pip-24.0-py3-none-any.whl (2.1 MB)
    2.1/2.1 MB 13.5 MB/s eta 0:00:00
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 23.3.2
    Uninstalling pip-23.3.2:
      Successfully uninstalled pip-23.3.2
  Successfully installed pip-24.0
```

Les bibliothèques étant importées nous réalisons le code suivant disponible sur le GitHub.

Vous pouvez retrouver le code en entier ici : [Code STM32 GitHub](#)

Vous pouvez également constater grâce à la vidéo YouTube suivante que le code fonctionne correctement.

Vidéo YouTube : [Affichage du signal avec Python](#)

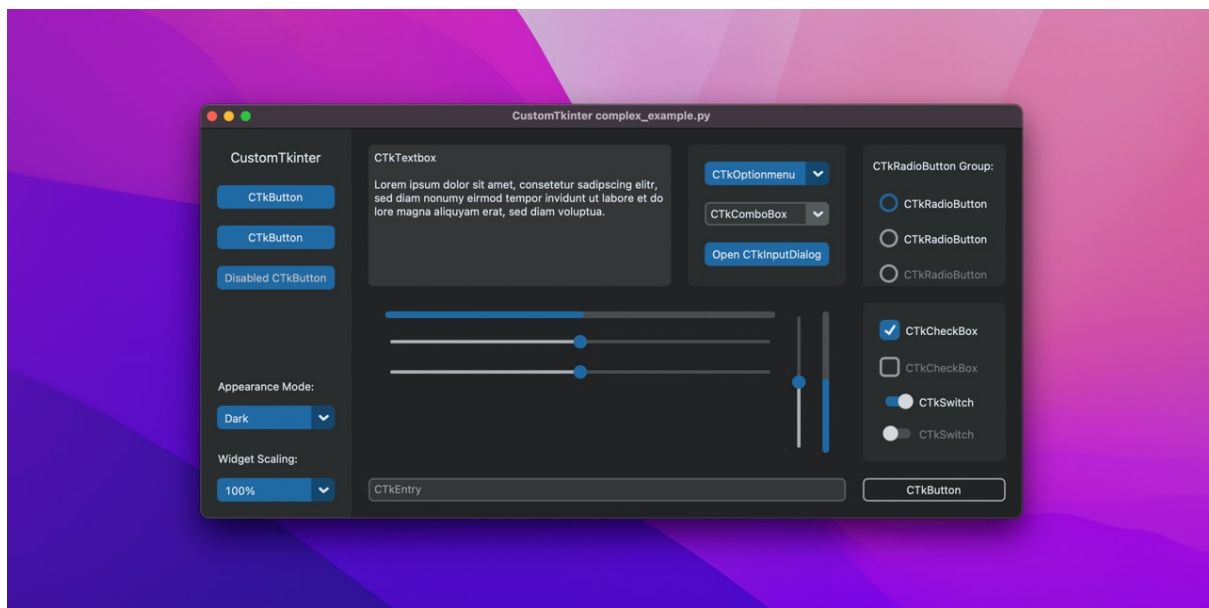
5.3. Interface graphique

L'objectif de cette partie est d'afficher le signal en direct dans une interface graphique représentant un oscilloscope.

Pour cela nous devons utiliser une bibliothèque permettant de créer des interfaces graphiques.

La bibliothèque la plus connue pour ce genre de réalisation est évidemment la bibliothèque *Tkinter* mais nous trouvons que les interfaces graphiques que l'on peut créer avec cette bibliothèque ne sont pas assez modernes.

C'est pour cela que nous utiliserons la bibliothèque *CustomTkinter* qui elle permet de créer des interfaces épurées et modernes.

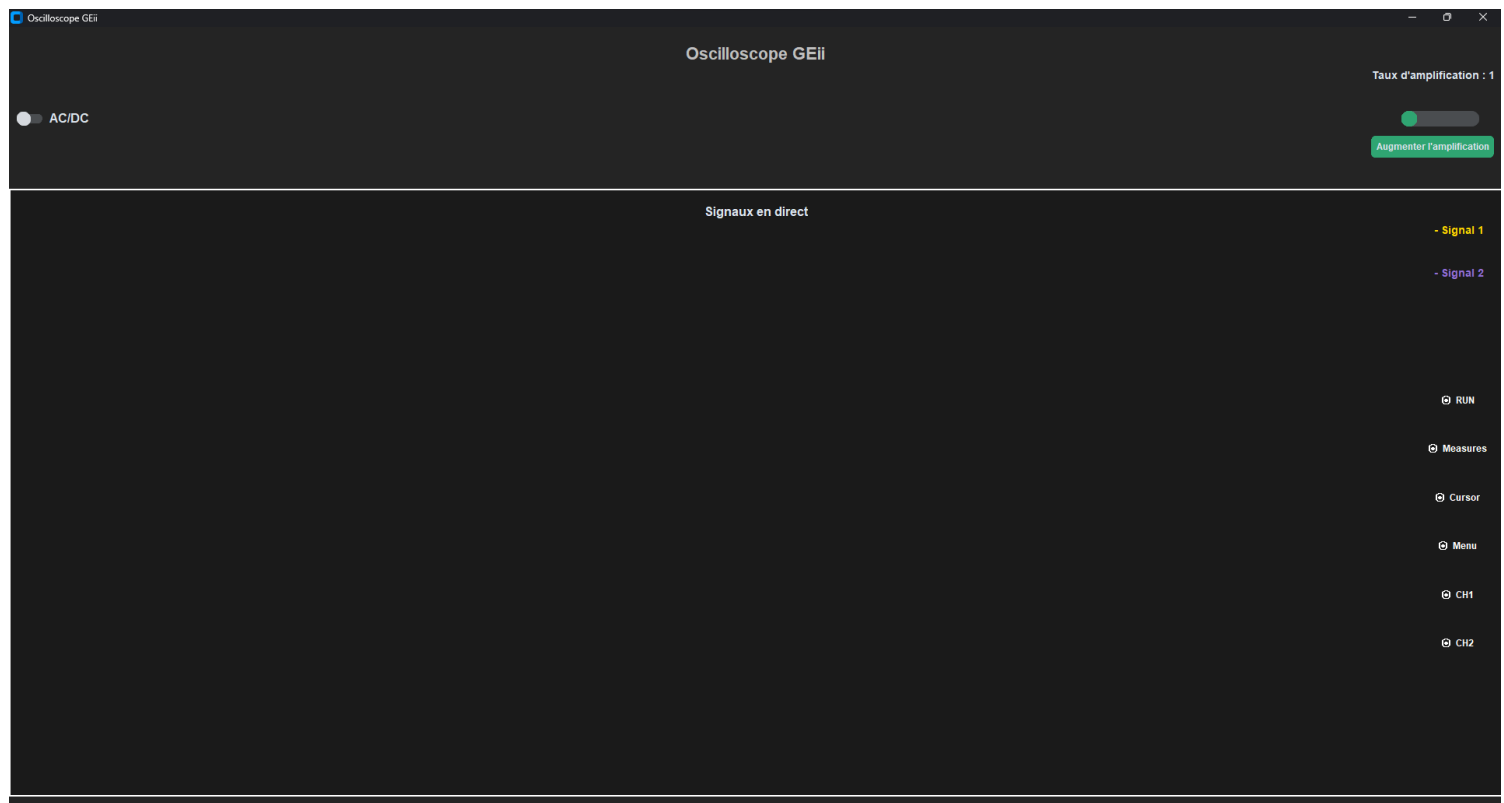


Exemple d'interface avec la bibliothèque CustomTkInter

Nous réalisons donc le code suivant disponible sur le GitHub et obtenons le résultat ci-dessous.

Le code Python : [Code STM32 GitHub](#)

Grâce à ce code nous avons pu créer l'interface graphique suivante.



Interface graphique avec Python

On remarque dans cette interface une zone réservée à l'affichage des signaux, cette zone devra être attribuée au signal que l'on a pu obtenir grâce au premier code Python.

On remarque également des boutons pour les modes AC/DC et le taux d'amplification, c'est bouton sont seulement illustrative car nous n'avons pas pu réaliser le programme qui permettrait de les assigner aux fonctions correspondantes (manque de temps).

Il y a également des boutons sur le côté droit avec différents modes mais ces boutons ne sont que illustratives.

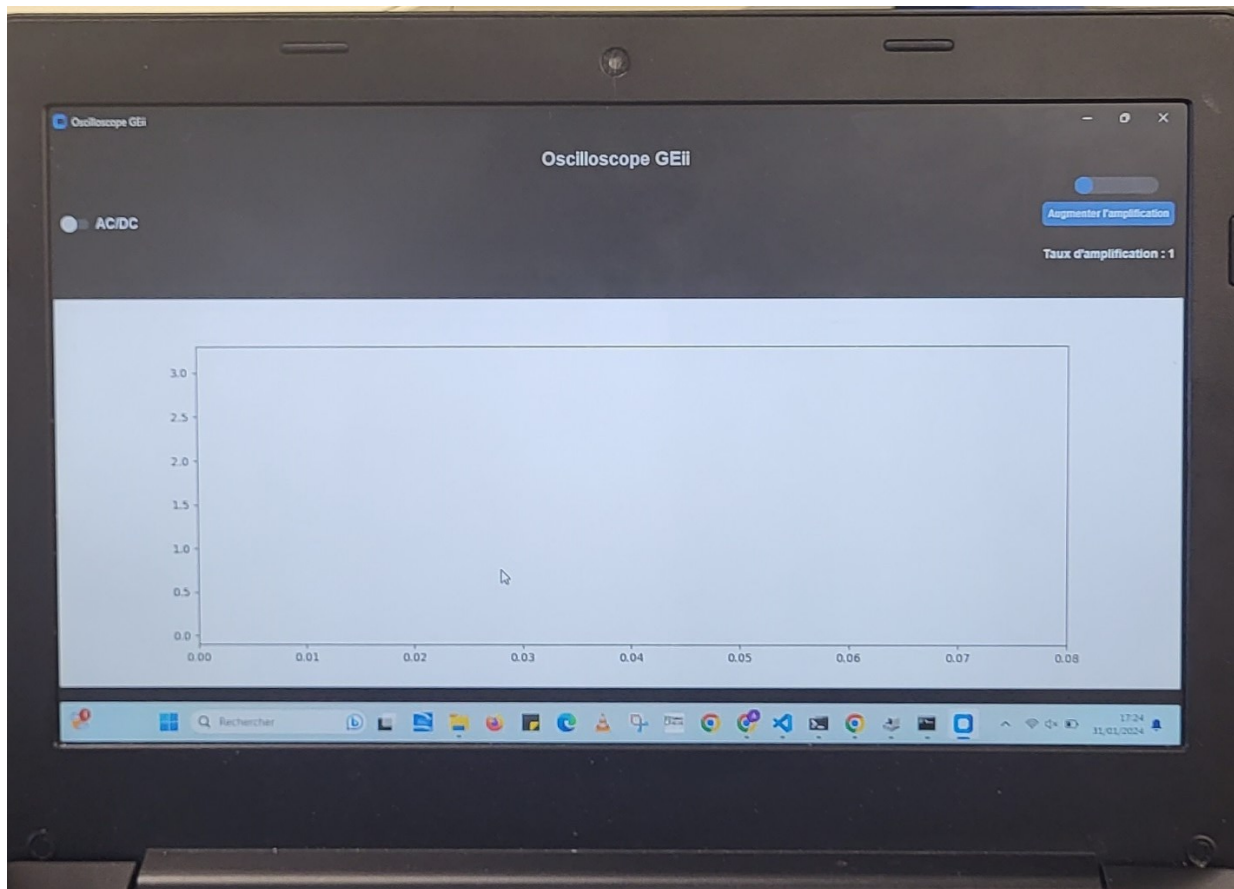
Il faut aussi noter que le système de placement des objets est un système de grille, c'est-à-dire que dans notre zone attribuée ou l'on peut placer nos objets il y a comme une grille que l'on ne peut pas voir donc lorsque l'on place un objet on définit la ligne et la colonne de la grille pour le placer, un peu comme sur Excel.

Enfin nous devons maintenant intégrer le signal de la carte directement sur notre interface graphique, cela veut dire que nous allons devoir combiner les deux code Python en un seul code.

Nous réalisons alors le code suivant disponible sur le GitHub.

Le code Python : [Code STM32 GitHub](#)

Nous compilons et obtenons le résultat suivant.



Résultat de la combinaison des deux programmes Python

On remarque que l'on arrive bien à afficher les deux interfaces graphiques avec les bons paramètres de l'axe des ordonnées par contre on remarque également que le signal sortant de la carte STM32 ne s'affiche pas.

Malheureusement, par manque de temps, nous n'avons pas pu régler ce problème mais après analyse nous arrivons à la conclusion que cela peut être dû à seulement deux problématiques, les voici.

- **Problème de communication série** : Le port que nous utilisons peut changer entre les différentes phases de test donc il faut le mettre à jour sur le code Python, si nous lisons le mauvais port série forcément rien ne s'affiche ce qui est cohérent avec l'erreur.
- **Erreur dans la configuration de l'animation** : Dans la fonction « *ani* » qui permet l'animation du signal nous pensons qu'il y a peut-être une erreur dans la configuration des paramètres de la fonction la rendant pas adaptée.

Comme nous n'avons pas tout le temps la carte STM32 à notre disposition nous ne pouvons pas vérifier cela.

Conclusion

L'objectif de cette SAE été de pouvoir visualiser les signaux sortants de notre carte STM32L476RGTxLQFP64 et de visualiser certains phénomènes comme le passage de mode AC/DC ou le taux d'amplification.

Nous avons d'abord commencé à réaliser un schéma KiCad qui montre ce qu'électriquement parlant un oscilloscope fait, c'est-à-dire comment se fait le passage du mode AC/DC, comme se fait la modification du Gain etc...

Puis nous reproduit ce schéma sur une protoboard et l'avons testé en mettant la sortie sur un oscilloscope de l'I.U.T pour observer les phénomènes attendus

Ensuite nous avons dû programmer notre carte STM32 de telle manière que le signal analogique sortant de la carte soit convertie en un signal numérique pour que l'on puisse l'exploiter.

Une fois ce signal converti nous le transférons sur un port de notre PC grâce à la méthode UART.

Puis à l'aide d'un programme Python nous lisons les valeurs communiquées sur ce port et les implémentons en direct dans notre programme afin d'afficher le signal représentant la sortie de la carte STM32 en valeur numérique sur un graphique.

Enfin nous avons créé une interface graphique permettant d'afficher le signal sur un modèle graphique moderne, malheureusement nous n'avons pas réussi à afficher le signal en direct dans notre interface.

Ce projet nous a permis de nous améliorer dans de nombreux domaines techniques.

- Nous avons appris le fonctionnement d'un oscilloscope et maîtrisons maintenant mieux son utilisation.
- Nous avons appris à soigner nos câblages sur protoboard pour éviter tous problèmes de court-circuit et pour rendre la transmission de courant plus efficace.
- Nous sommes améliorés dans la programmation de microcontrôleur sur STM32CubeIDE.
- Nous nous sommes améliorés dans la programmation sur Python plus spécifiquement dans la programmation orienté objet et nous avons appris comment trouver les bibliothèques nécessaires pour la réalisation d'un projet.
- Nous avons appris à mettre en corrélation et à communiquer entre plusieurs codes et objets.
- Nous avons appris comment fonctionner la communication par port série.

Et bien d'autres encore...

Nous remercions les enseignants qui nous ont permis la réalisation ce projet et nous ont aidés tout au long de celui-ci.