TP Systèmes à microcontrôleurs

L'objectif de ce TP en 4 séances est de concevoir, assembler et tester une carte électronique contenant un microcontrôleur STM32. Le TP a été conçu avec la version 6.0.9 de KiCAD, mais devrait être réalisable avec d'autres versions.

Cette carte sera constituée des composants suivants :
— Microcontrôleur STM32L021K4T6,
— DAC MCP4801-E/SN,
— Régulateur linéaire BU33SD5WG-TR,
— 2 LED au format 0603,
— Connecteur de programmation FTSH-107-01-F-DV,
— Plusieurs composants passifs (résistances, condensateurs...) en 0603.

La carte est donc munie d'un ADC (intégré au MCU) et d'un DAC (composant externe sur la carte). Pour démontrer le bon fonctionnement de la carte, vous programmerez un filtre numérique simple.
L'ordre des séances est le suivant :

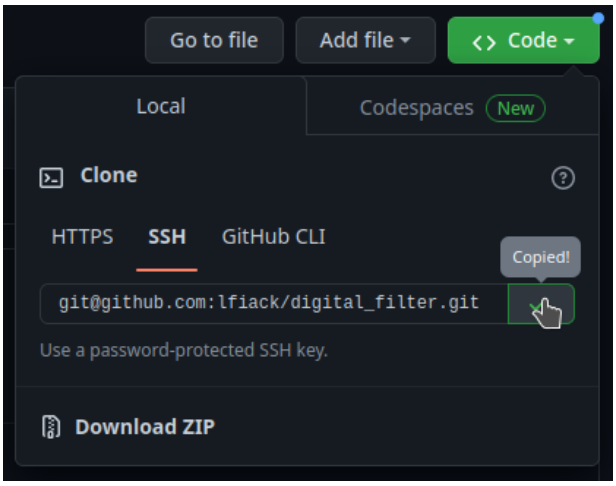
1. Schéma, association empreintes
2. Routage du circuit
3. Soudure, test...
4. Écriture du firmware

0 Utilisation de `git`

Les outils de gestion de version permettent de travailler facilement à plusieurs sur du code, ou autre. L'outil `git` est probablement le plus courant. C'est celui que nous allons utiliser.
Sous Linux, l'outil s'utilise simplement en ligne de commande. Sous Windows, il existe des outils. Je suis un Linuxien, débrouillez-vous.
`Git` est un outil puissant, complet et complexe, mais nous n'aurons besoin de connaître que 5 commandes :
— `git clone` : Cloner un projet, c'est la première étape pour récupérer son projet, ou n'importe quel projet publique.
— `git pull` : Récupérer les dernières modifications
— `git add <fichier>` : Enregistrer des fichiers auprès de git
— `git commit -m "Message"` : Réaliser un commit, c'est à dire enregistrer les modifications locales. On peut l'utiliser comme un verbe. Ex : "*Oups, j'ai oublié de commiter mon code*".
— `git push` : Pousser du code sur le serveur. Le commit se faisant localement, cette commande permet de pousser le commit sur le serveur.

> Je vous recommande **fortement** d'utiliser cet outil pour tout vos TP, que ce soit pour le code ou pour les compte-rendu.

1. Si ce n'est pas déjà fait, créez un compte (par étudiant) sur le site suivant : `https://github.com/`
2. Créez un nouveau *repository*. Laissez-le en public. Vous pouvez ajouter une description et un fichier README.
3. Récupérez l'adresse du projet. Cliquez sur Code, choisissez SSH, puis copiez l'adresse.



4. Clonez le projet. Dans un terminal, tapez la commande suivante : `git clone <adresse_du_depot>`.
5. Modifiez le fichier README.
6. Réalisez votre premier commit. Tapez les deux commandes suivantes :
```
git add .
git commit -m "First!"
```
7. Poussez les modifications sur le repository distant :
```
git push
```
8. Vérifiez que les modifications ont bien été prises en compte sur `https://github.com/`

Quelques bonnes pratiques :
- Faire un `git pull` au début de chaque session de travail (on sait jamais, peut-être que mon binôme a travaillé ?)
- Commiter régulièrement :
  - Dès que quelque chose fonctionne
  - À la fin de la session de travail
  - Éviter de commiter du code qui plante (à l'école, ça peut entrer en contradiction avec le point précédent...)
  - Mettez un message utile !
- Utiliser le fichier README comme document de travail.

1 Saisie du schéma

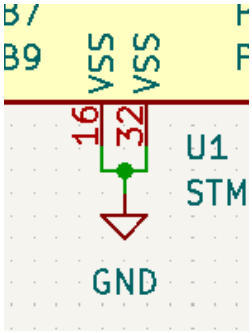
1.1 Création de la structure du projet

Le projet contiendra du hardware et du software. Il est impératif de bien structurer le dossier de travail.

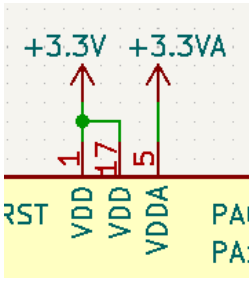
1. Dans le dossier `git`, créez un dossier `hardware`, un dossier `firmware` et un dossier `datasheets`.
2. Téléchargez les *datasheets* de tous les composants, et copiez les dans le dossier `datasheets`. Attention, la documentation du STM32 est en plusieurs parties. Téléchargez également la *datasheet* de la sonde de programmation STLINK-V3MINI. Pensez à commiter.
3. Lancez STM32CubeMX. C'est un outil pratique lors de la conception d'une carte, notamment pour gérer le *pinout*.
4. Créez un projet pour le `STM32L021K4`.
5. Dans SYS, activez `Debug Serial Wire`.
6. Activez l'USART2 en mode `Asynchrone`. Gardez les paramètres par défaut, et notez le `baud rate`.
7. Dans l'onglet `Clock Configuration`, configurez l'horloge système à sa valeur maximale.
8. Dans `Project Manager > Code Generator`, cochez `Generate peripheral initialization as a pair of '.c/.h' files per peripheral`
9. Sauvegardez dans le dossier `firmware`. L'initialisation n'est pas terminée, vous y reviendrez en temps voulu.
10. Pensez à commiter.

1.2 Le microcontrôleur sous KiCAD

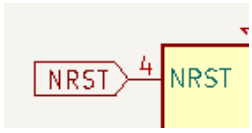
1. Lancez KiCAD. Créez un projet dans le dossier `hardware`.
2. Cliquez sur `Place > Add symbol` (Vous pouvez commencer à apprendre les raccourcis.)
3. Chechez et placez le composant STM32L021K4Tx.
4. Cliquez sur `Place > Add Power` (Aussi présent dans le menu à droite.)
5. Placer GND proche des *pins VSS* du STM32, et câblez-les comme dans la capture ci-dessous.



6. Ajouter deux autres symboles +3.3V et +3.3VA selon la capture ci-dessous.



7. Cliquez sur `Place > Add Label` Placez le label sur l'entrée NRST et nommez le NRST.

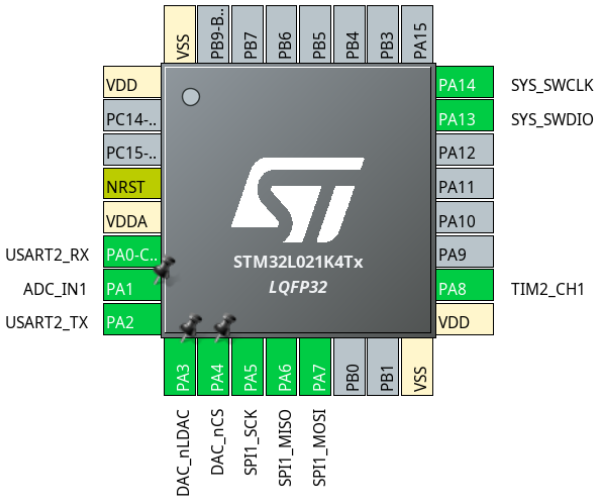


   Il sera plus tard relié au port de programmation. Deux signaux connectés à un même label sont reliés électriquement. Ça permet de simplifier la lecture du schéma.

8. Ajoutez les labels correspondant aux signaux définis dans STM32CubeMX (USART2_RX, USART2_TX, SWDIO, SWDCLK).

   D'ailleurs, c'est peut-être le bon moment de finaliser le *pinout*

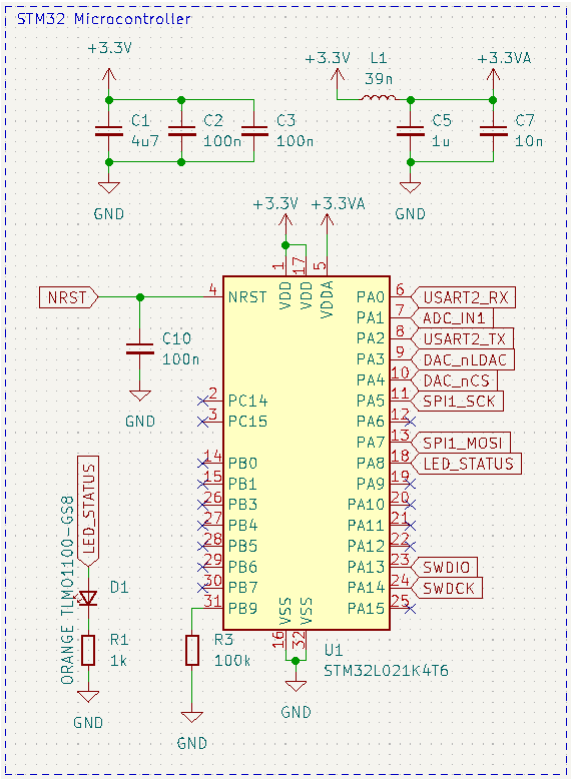
9. Activez l'entrée IN1 de l'ADC. Ce sera l'entrée analogique pour le filtre.
10. Dans TIM2, mettez *Clock Source* à *Internal Clock* et *Channel 1* à *PWM Generation CH1*. On s'en servira pour faire varier la luminosité de la LED.
11. Enfin, configurez les signaux nécessaires pour l'utilisation du DAC. Le résultat final pourrait ressembler à ça :



   Sauvegardez.

12. Il est temps d'ajouter le reste des composants. Ajoutez les composants suivants :
    — 6x C_Small
    — 1x L_Small
    — 2x R_Small
    — 1x LED_Small

    Le schéma correspondant à la partie microprocesseur pourrait ressembler à ça :



    Notez qu'il n'est pas nécessaire d'annoter les composants à ce stade. Ce sera fait automatiquement plus tard. Les composants peuvent donc être nommés C ?, L ?, U ?...

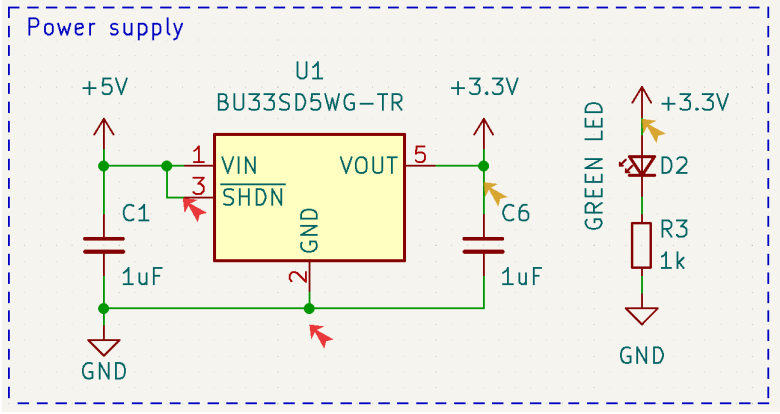
    Par contre, il est important de bien noter les valeurs des composants (4u7, 39n, 1k...)

13. Pourquoi PB9 est relié à la masse ? (Répondez à cette question, et aux suivantes, dans le README du repo git.)
14. Quel est le rôle de L1, C5 et C7 ?

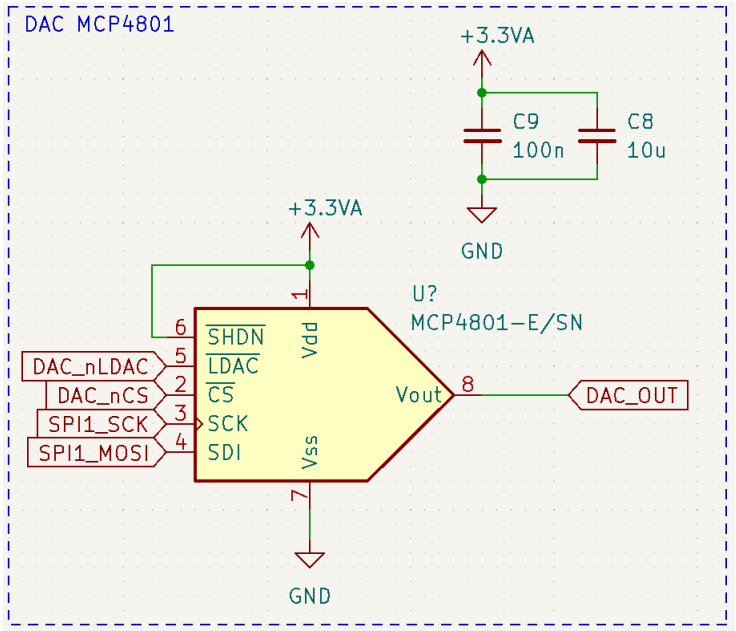
1.3 Le reste du schéma

Pour générer la tension de 3.3V à partir du 5V, nous utiliserons le régulateur linéaire LDO `BU33SD5WG-TR`. Il n'est pas présent dans la librairie de KiCAD. Pour éviter d'avoir à dessiner un nouveau symbole, nous pouvons simplement utiliser un autre composant qui dispose du même boitier, comme le `MCP1802x-xx02xOT`.

1. Ajouter le composant `MCP1802x-xx02xOT` et renommez-le en `BU33SD5WG-TR`
2. Câblez le régulateur de tension en vous basant sur la figure suivante :

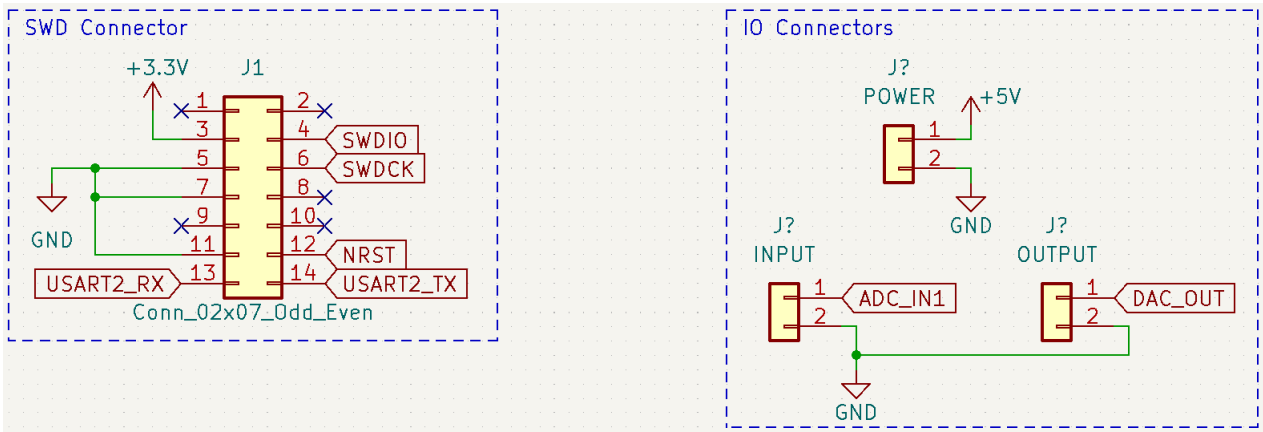


3. Quelle page de la *datasheet* indique les valeurs des condensateurs ?
4. Tracez le schéma du DAC en vous inspirant de la figure suivante :



5. Quelle page de la *datasheet* nous indique les valeurs de condensateurs ?
6. Quel est le rôle de la broche $\overline{\text{CS}}$?
7. Quel est le rôle de la broche $\overline{\text{LDAC}}$?
8. Pourquoi le signal MISO n'est pas utilisé ?

9. Câblez les connecteurs comme dans la figure suivante :



10. Où trouve-t-on les indications du pinout du connecteur SWD ?
11. Numérotez les composants en cliquant sur `Tools > Annotate Schematic...`
12. Vérifiez les ERC (Electrical Rule Check) en cliquant sur `Inspect > Electrical Rules Checker`.

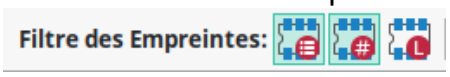
    Les erreurs suivantes peuvent être ignorées. S'il y en a plus, c'est qu'il y a peut-être des erreurs dans le schéma, corrigez les.

1.4 Affectation des empreintes

Vous avez fini le tracé du schéma. Il faut maintenant, pour chaque composant, associer un empreinte.

1. Cliquez sur `Tools > Assign Footprints...`
2. Faire les associations selon la figure suivante. Attention, en fonction du placement des composants dans le schéma, les composants n'ont peut-être pas les mêmes noms.

```
 1      C1 -              1uF : Capacitor_SMD:C_0805_2012Metric
 2      C2 -              4u7 : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
 3      C3 -             100n : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
 4      C4 -             100n : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
 5      C5 -             100n : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
 6      C6 -              1uF : Capacitor_SMD:C_0805_2012Metric
 7      C7 -               1u : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
 8      C8 -              10n : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
 9      C9 -             100n : Capacitor_SMD:C_0805_2012Metric_Pad1.18x1.45mm_HandSolder
10     C10 -              10u : Capacitor_SMD:C_0805_2012Metric
11      D1 - ORANGE TLMO1100-GS8 : LED_SMD:LED_0603_1608Metric_Pad1.05x0.95mm_HandSolder
12      D2 -        GREEN LED : LED_SMD:LED_0603_1608Metric_Pad1.05x0.95mm_HandSolder
13      J1 - Conn_02x07_Odd_Even : Connector_PinHeader_1.27mm:PinHeader_2x07_P1.27mm_Vertical_SMD
14      J2 -            INPUT : Connector_JST:JST_XH_B2B-XH-A_1x02_P2.50mm_Vertical
15      J3 -            POWER : Connector_JST:JST_XH_B2B-XH-A_1x02_P2.50mm_Vertical
16      J4 -           OUTPUT : Connector_JST:JST_XH_B2B-XH-A_1x02_P2.50mm_Vertical
17      L1 -              39n : Inductor_SMD:L_0805_2012Metric_Pad1.05x1.20mm_HandSolder
18      R1 -               1k : Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder
19      R2 -             100k : Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder
20      R3 -               1k : Resistor_SMD:R_0805_2012Metric_Pad1.20x1.40mm_HandSolder
21      U1 -     BU33SD5WG-TR : Package_TO_SOT_SMD:SOT-23-5
22      U2 -    STM32L021K4T6 : Package_QFP:LQFP-32_7x7mm_P0.8mm
23      U3 -     MCP4801-E/SN : Package_SO:SOIC-8_3.9x4.9mm_P1.27mm
```

   Pour trouver plus facilement les empreintes, il peut être utile de bien configurer les filtres 

3. Que signifie 0805 ? 0603 ? 1206 ?
4. Que signifie LQFP ? SOT-223 ? SOIC ? Ne vous contentez pas de donner le sigle, j'attends une petite description (vous pouvez copier-coller depuis wikipedia, mais lisez avant quand même))

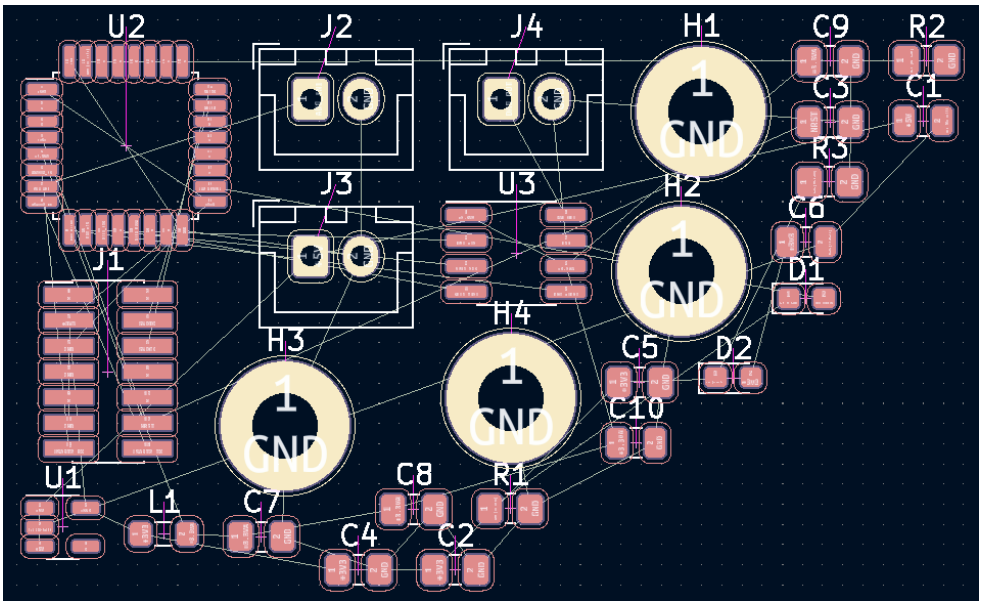
Félicitations ! Vous avez tracé le schéma complet de votre circuit et avez associé une empreinte à chaque symbole dans le schéma. Dans l'étape suivante, ces informations seront importées dans le logiciel de routage.

Faites vérifier le schéma à l'enseignant avant de passer à la suite.

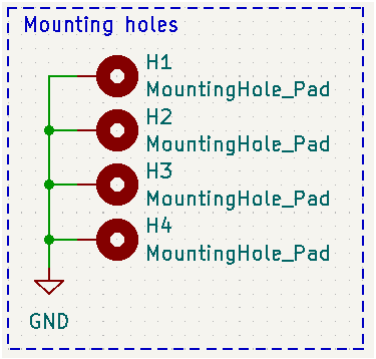
2 Routage du circuit

2.1 Placement des composants

1. Ouvrez l'*Éditeur de PCB*.
2. Cliquez sur `Tools > Update PCB from Schematic...`
   Les composants sont mis en vrac, les connexions à router sont représentées par des traits fins. Cet ensemble de traits est appelé *chevelu* (ou *ratsnest* en Anglais).



   Dans la figure ci-dessus, il y a 4 grandes pastilles que vous n'avez pas chez vous. Elles serviront de trous de fixation. Il y a plusieurs méthodes pour les ajouter. Vous allez les rajouter dans le schéma.
3. Dans l'*Éditeur de Schématique*, ajouter les composants suivants :

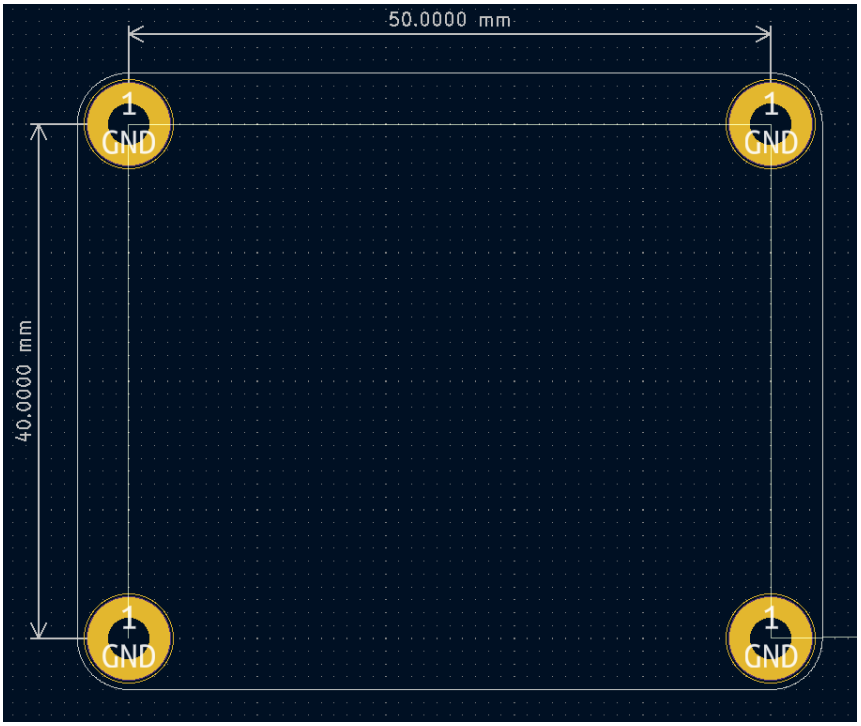


4. N'oubliez pas de faire le lien avec les empreintes :

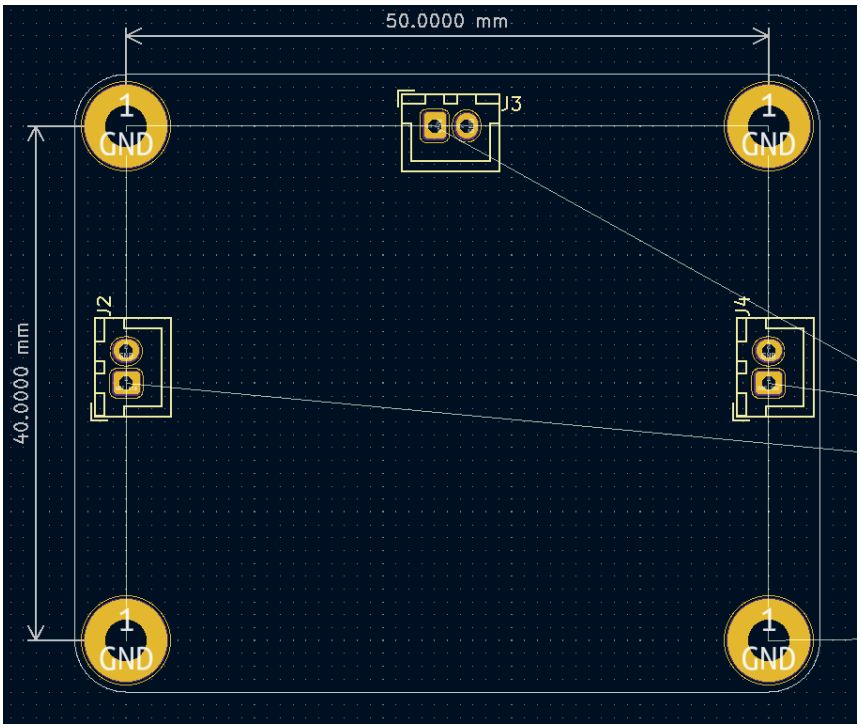
```
12      H1 - MountingHole_Pad : MountingHole:MountingHole_3.2mm_M3_Pad
13      H2 - MountingHole_Pad : MountingHole:MountingHole_3.2mm_M3_Pad
14      H3 - MountingHole_Pad : MountingHole:MountingHole_3.2mm_M3_Pad
15      H4 - MountingHole_Pad : MountingHole:MountingHole_3.2mm_M3_Pad
```

5. Réglez la grille à 1mm.

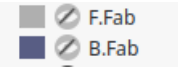
6. Commencez par placer les trous de fixation à 50mm de distance sur l'axe X et à 40mm de distance sur l'axe Y.
7. Tracez les contours du circuit. Cliquez sur `Edge.Cuts` dans le panneau de droite. Utilisez les lignes droites et les arcs de cercle pour obtenir quelque chose comme dans la figure suivante :



8. Placez ensuite le connecteur d'alimentation au milieu en haut du circuit, à 4mm du bord. Placez le connecteur pour l'entrée analogique à gauche et le connecteur pour la sortie à droite. Voir la figure suivante :



9. Cliquez sur *Placer* > *Origine des Coord de Perçage/Placement*. Positionnez l'origine sur l'un des trous de fixation. Cette étape est nécessaire pour la fabrication.
10. Pour améliorer la lisibilité, vous pouvez masquer les couches `F.Fab` et `B.Fab`



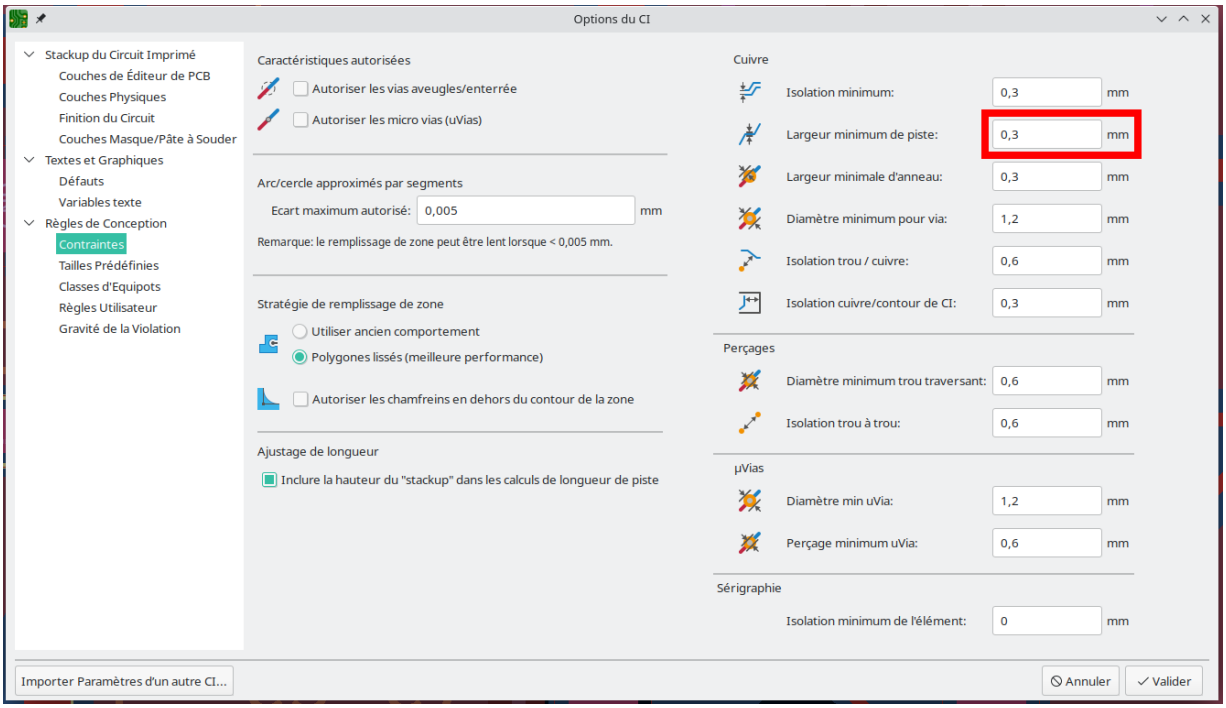
11. Placez les composants. Pour faciliter le routage, il y a quelques règles à respecter.
    — Placer le composant le plus complexe (le STM32) au centre.
    — Placer les condensateurs de découplage proche des broches d'alimentation à découpler.
    — Avoir le schéma ouvert dans une autre fenêtre, sur un autre écran de préférence.
    — Essayer de réduire au maximum les croisements dans le chevelu.
    — Prendre son temps. Ne pas hésiter à retourner à cette étape si le routage est compliqué.

    Une fois les composants placés, faites vérifier par l'enseignant avant de passer à la suite.

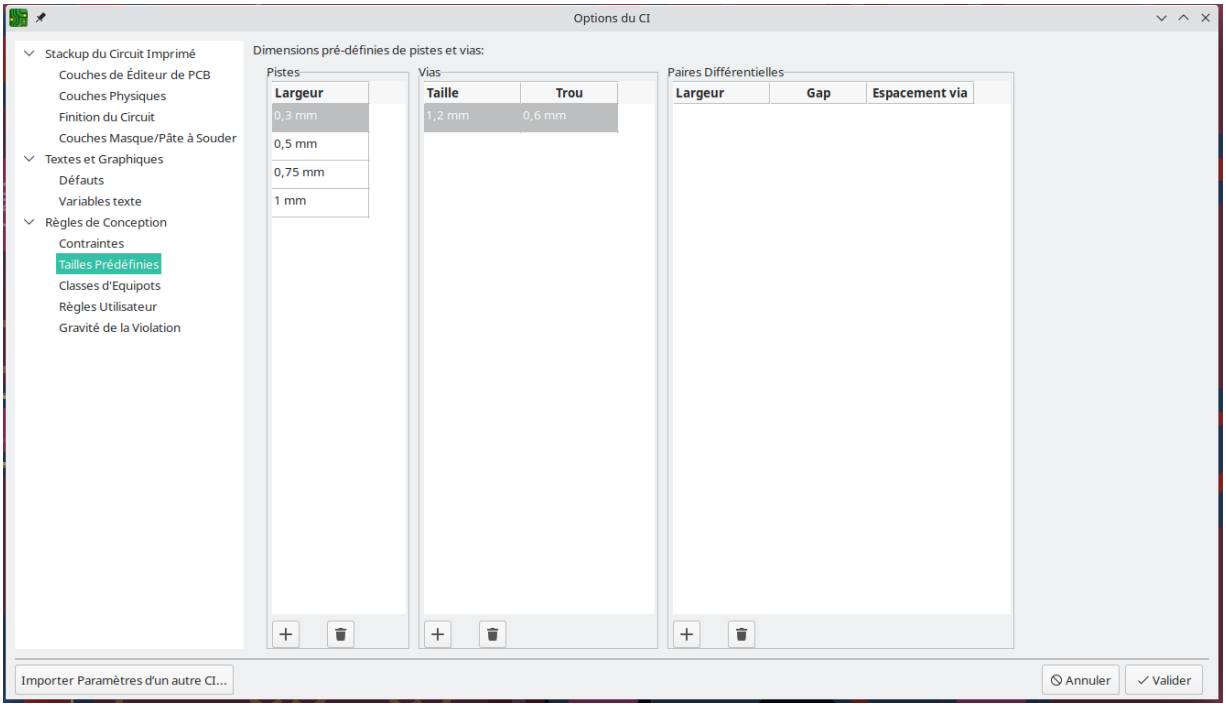
2.2 Routage

Avant de commencer à router, il faut configurer les règles de routage pour s'assurer que le circuit soit fabricable.

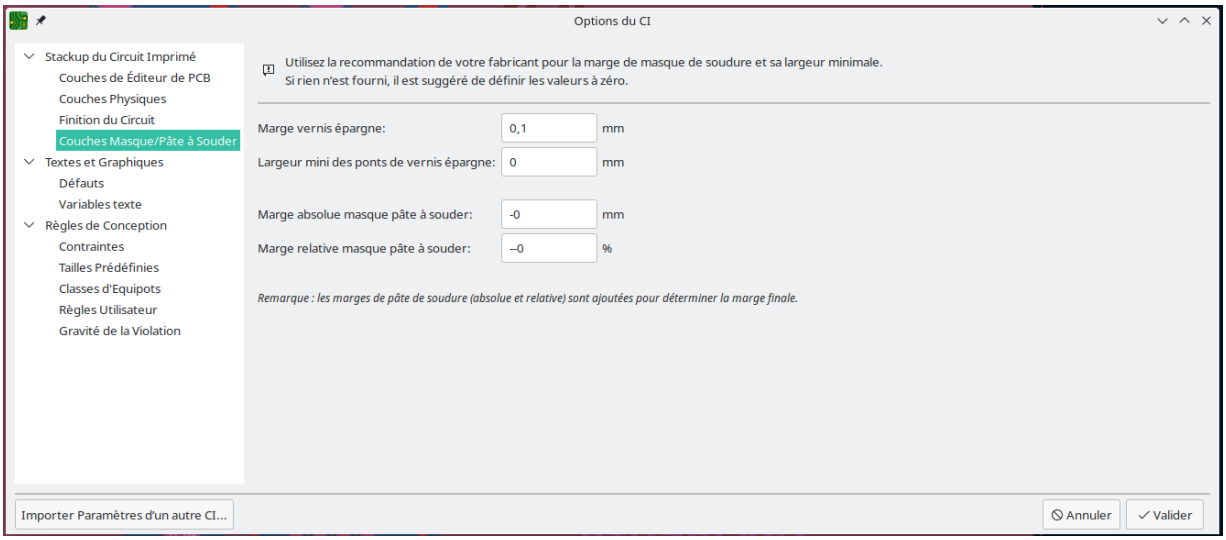
1. Cliquez sur `File > Board Setup...`
2. Dans *Design Rules* > *Constraints*, configurez selon la figure suivante :



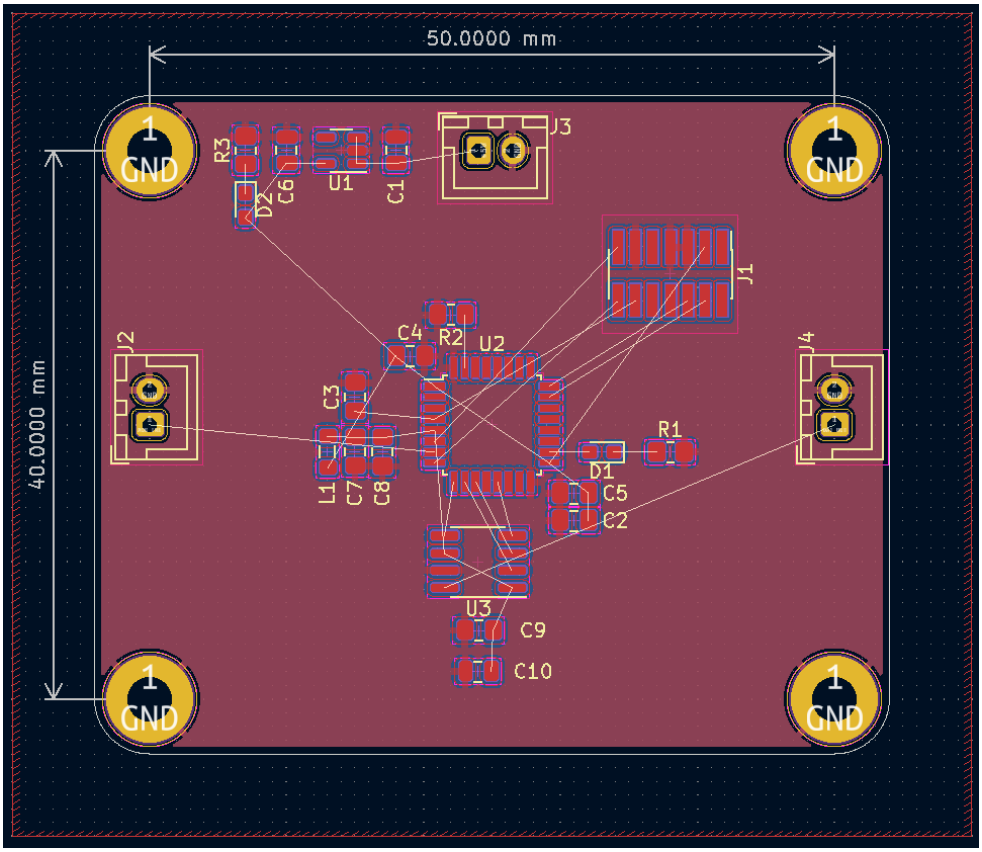
3. Dans *Design Rules > Pre-defined Sizes*, configurez selon la figure suivante :



4. Dans *Board Stackup > Solder Mask/Paste*, configurez selon la figure suivante :



5. Dans le panneau de droite, cliquez sur F.Cu. Ensuite, cliquez sur `Place > Add Filled Zone...`. Choisissez le signal GND. Tracer un rectangle englobant tout le circuit.

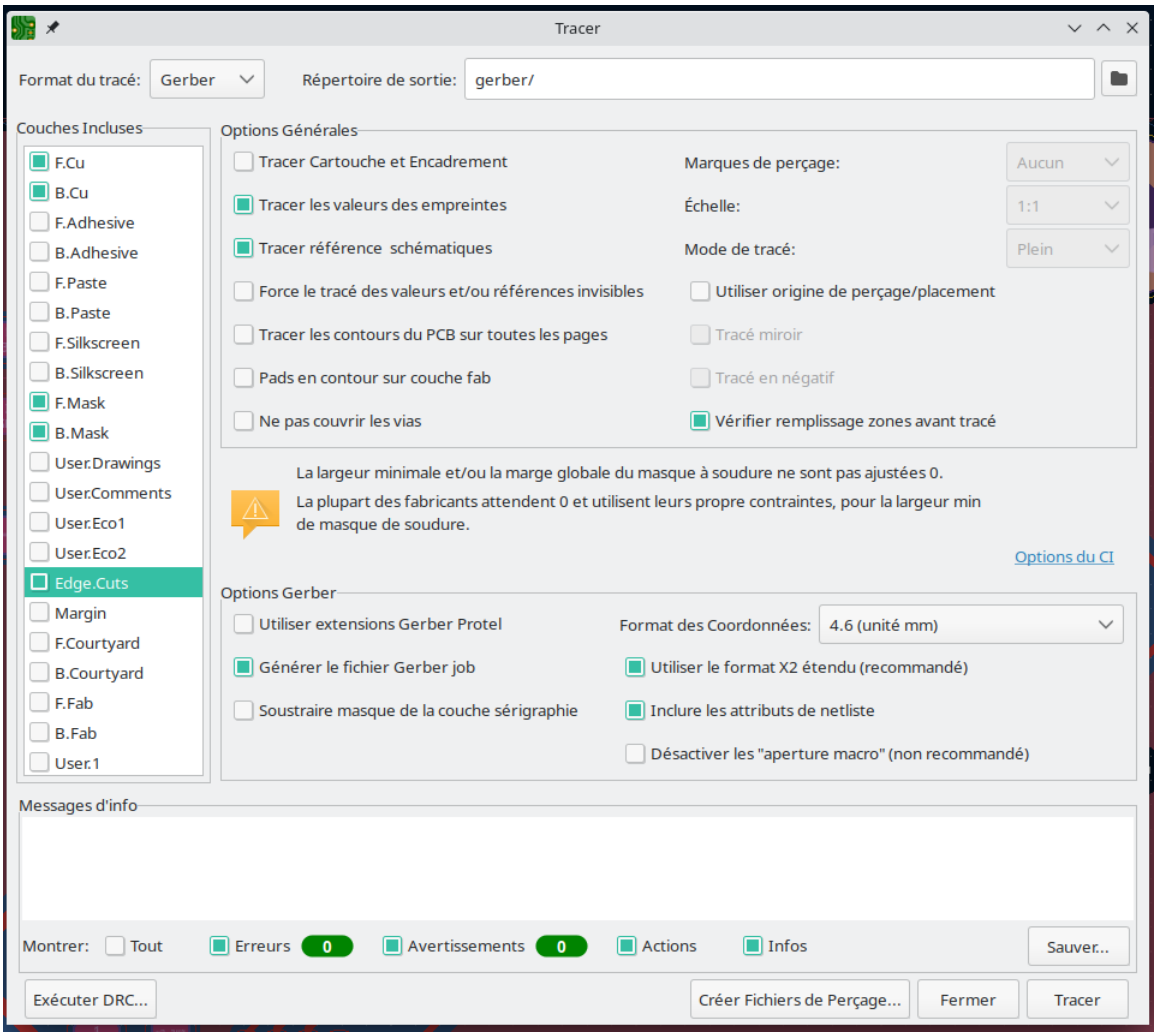


6. Faites de même avec la couche B.Cu.
7. Pour pouvoir router, cliquez sur `View > Drawing Mode > Draw Zone Outlines`. (Aussi présent dans le menu de gauche.)
8. Routez la carte. Les pistes de 0.3mm sont reservées aux signaux. Les pistes de 0.5mm, 0.75mm et 1mm sont à utiliser pour les alimentations. C'est encore une étape assez longue. N'hésitez pas à supprimer des pistes déjà routées pour trouver de meilleures solutions. Pensez également à replacer certains composants pour faciliter le routage.

Félicitations vous avez routé la carte ! Faites valider le routage par l'enseignant.

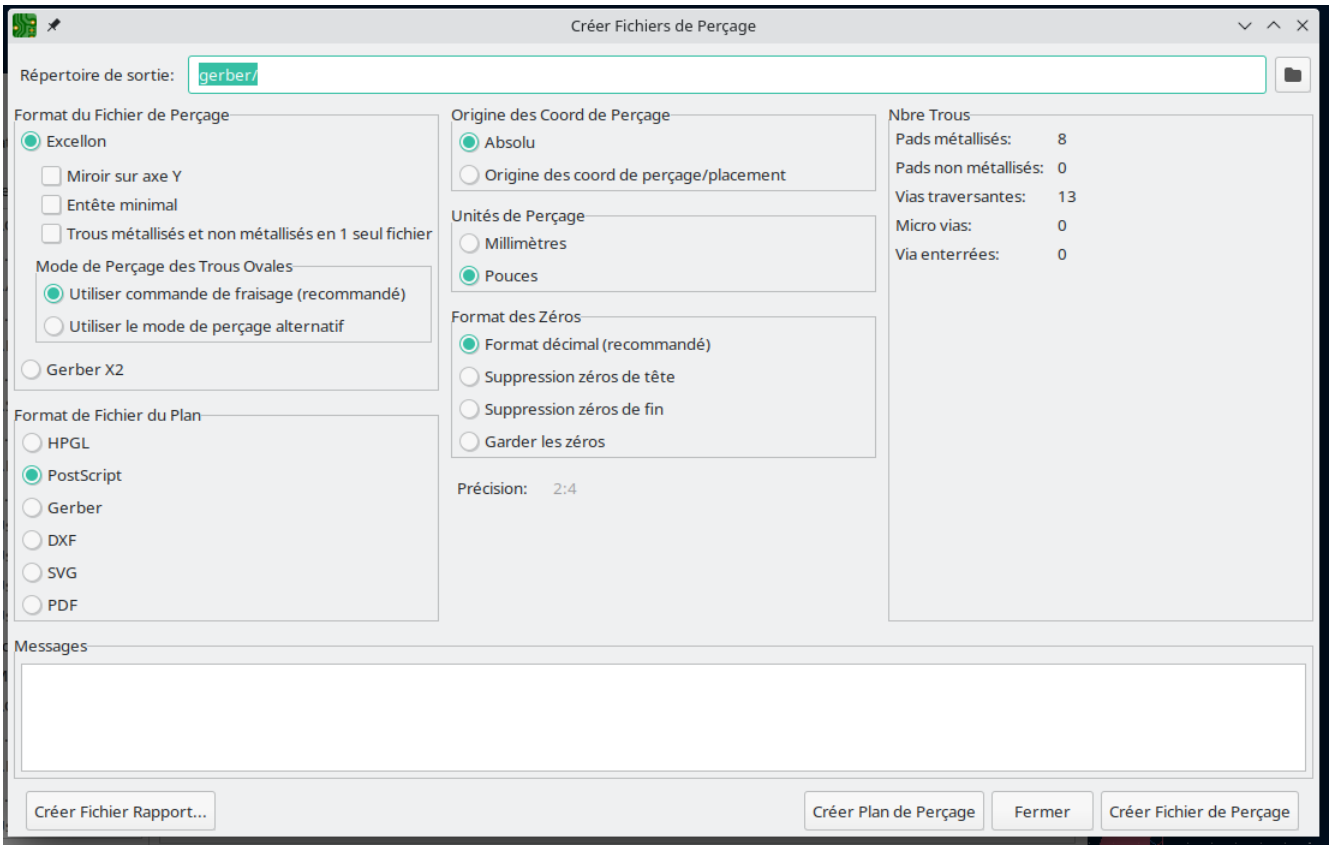
2.3 Génération des fichiers de fabrication

1. Cliquez sur *File > Fabrication Outputs > Gerbers (.gbr)*.
2. Configurer selon la figure suivante :



3. Cliquez sur *Créer Fichiers de Perçage...*

4. Configurer selon la figure suivante :



5. Cliquez sur *Créer fichier de perçage*
6. Cliquez sur *Tracer*
7. Faites valider. N’oubliez pas de commiter et de pousser sur github.

3 Écriture du firmware

La section *Écriture du firmware* est présente avant la section *Assemblage* parce que, arrivés à cette partie du projet, vous n'avez probablement pas encore votre carte. C'est quelque chose d'assez courant dans une phase de développement.

Néanmoins, il est quand même possible d'avancer :

— On peut tester que le projet compile sur la cible.
— On peut valider l'application sur PC ou sur une autre cible (F746 discovery par exemple).
— On peut tester les drivers sur une autre cible (F746 ou Nucleo).

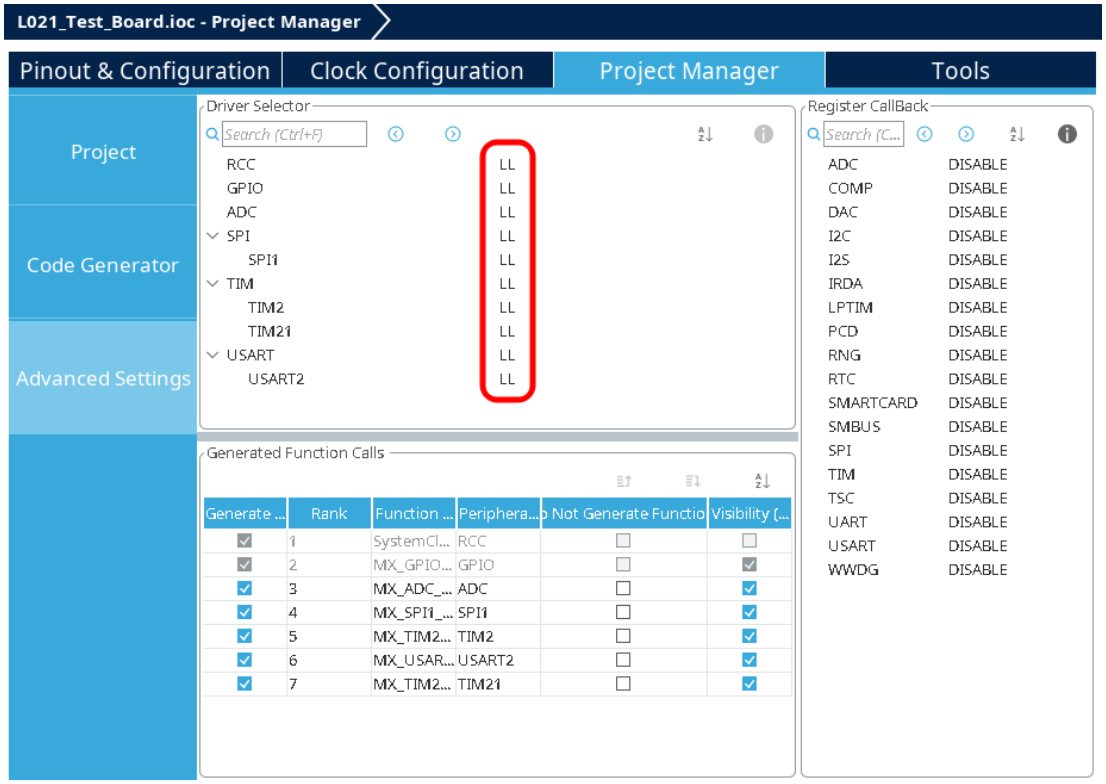
Vous constaterez rapidement que le STM32L021K4 est assez contraint en mémoire RAM et FLASH. Ainsi, vous allez souvent chatouiller les limites du microcontrôleur.

Ce point vous imposera quelques contraintes :

— Il est hors de question d'utiliser FreeRTOS.
— On ne pourra pas utiliser de nombres flottants.
— On ne pourra pas écrire de structure de drivers trop complexe (on essaiera quand même d'écrire du code propre).
— On ne pourra pas utiliser les fonctions de haut niveau de la HAL.
    — On pourra quand même utiliser les Low Layer (LL) drivers.
— On devrait pouvoir utiliser le shell.

3.1 Activation des LL drivers

1. Dans le fichier `.ioc`, dans *Project Manager* > *Advanced Settings*, choisissez LL pour tous les périphériques.



2. Regardez les fichiers générés. Quelles sont les différences ?
3. Les fonctions de la LL sont des accès directs aux registres. Il faudra chercher de la documentation. Vous aurez besoin du *Reference Manual* du STM32L021K4 et de la documentation de la HAL STM32L0 (ou lire le code à l'aide de la touche F3 sur CubeIDE).
4. Que signifie `__STATIC_INLINE` ?
5. Et pourquoi y a-t-il du code dans un .h alors que Môssieur Fiack vous a expressément demandé de pas le faire ? (Les deux questions sont peut-être liés, va savoir)

3.2 LED simple

1. Configurer le timer de la LED pour obtenir une fréquence de découpage d'1kHz et une résolution de 8bits. Ça veut dire que Counter period vaut 255.
2. Quelle valeur donner au prescaler ?
3. Créez une paire de fichiers `Led.c` / `Led.h` dans `Core/Src` et `Core/Inc` respectivement
4. Écrivez les trois fonctions suivantes :

```c
// Démarre le timer
void LedStart(void);
// Configure le rapport cyclique de la PWM entre 0 et 255
void LedSetValue(uint8_t val);
// À chaque appel, cette fonction incrémente la luminosité de la LED
// Arrivé à la valeur maximale, chaque appel décrémente la LED
void LedPulse(void);
```

5. La fonction LEDPulse demande de conserver la luminosité entre deux appels, ainsi qu'une variable permettant de savoir si on incrémente ou décrémente.
6. Si vous n'avez pas peur, vous pouvez stocker ses informations, ainsi que celles relative aux timer, dans une structure. Si c'est pas clair, n'hésitez pas à demander, votre enseignant est payé pour ça.
7. Pour un premier test, la fonction LEDPulse pourra être appelée dans une boucle infinie, accompagnée d'un délai. Ça pourrait ressembler à ça :

```c
/* USER CODE BEGIN WHILE */
LedStart();
while (1) {
    // Dans cet exemple, LedPulse fait appel à LedSetValue
    LedPulse();
    LL_mDelay(1);
    /* USER CODE END WHILE */

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

8. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

3.3 LED avec timer

1. Activer le timer qui n'est pas utilisé par la PWM de la LED.
2. Configurez le pour qu'il déclenche une interruption toutes les milli-secondes.
3. Quelles sont les valeurs de PSC et de ARR ?
4. Créez une paire de fichiers `TimeBase.c` / `TimeBase.h`
5. Écrivez une fonction :

   ```
   void TimeBaseStartIT(void);
   ```

6. Où se situe la routine de service d'interruption ?
7. Que manque-t-il par rapport au code généré par la HAL ?
8. En quoi est-ce catastrophique ?
9. Que faire ?
10. Déplacer l'appel à la fonction `LedPulse()` dans la routine de service du timer.
11. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

3.4 UART, un simple echo

1. La configuration de l'UART est probablement correcte par défaut. Vérifiez quand même, ça ne coûte pas grand chose.
2. Créez une paire de fichiers `Serial.c` / `Serial.h`
3. Écrivez les fonctions suivantes :

   ```
   // Cette fonction pourra être utilisée par le Shell v0.4
   uint8_t SerialTransmit(char * pData, uint16_t Size);
   // Dans cet exemple, on fait du polling, et c'est pas très grave
   char SerialReceiveChar(void);
   ```

4. Écrivez un test permettant de réaliser un echo. Ça pourrait ressembler à ça :

   ```
   /* USER CODE BEGIN WHILE */
   while (1) {
       char ch = SerialReceiveChar();
       SerialTransmit(ch, 1);
       /* USER CODE END WHILE */

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

5. Une erreur s'est glissée dans le code précédent, saurez-vous la retrouver ?
6. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

3.5 UART et Shell

1. Ajoutez le Shell que vous avez utilisé en TP de FreeRTOS, ou utilisez la toute dernière version disponible à cette adresse : `https://github.com/lfiack/mcuLib/tree/main/Middleware/Shell`
2. Le code pourrait ressembler à quelque chose comme ça :

```
    [...]
/* USER CODE BEGIN PV */
hShell_t hShell;
/* USER CODE END PV */
    [...]
int main(void) {
        [...]
    /* USER CODE BEGIN 2 */
    ShellInit(&hShell, &SerialTransmit);
    /* USER CODE END 2 */
        [...]
    /* USER CODE BEGIN WHILE */
    while (1) {
        char c = SerialReceiveByte();
        ShellProcess(&hShell, c);
        /* USER CODE END WHILE */

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

3. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

3.6 DAC, génération de signaux

1. Configurer le SPI dans le fichier `.ioc`
2. Créez une paire de fichiers `AnalogOut.c` / `AnalogOut.h`
3. Écrivez les fonctions suivantes :

```
void AnalogOutInit(void);
void AnalogOutConvert(uint16_t value);
void AnalogOutPulse(uint16_t increment);
```

   Comme `LedPulse()`, à chaque appel, cette fonction incrémente la valeur à envoyer au DAC jusqu'à une valeur maximale. Arrivé à la valeur maximale, chaque appel décrémente cette valeur. Ça devrait générer un signal triangle.
4. Testez avec le timer `TimeBase`. Augmenter la fréquence d'interruption du timer, par exemple à 32kHz.
5. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

3.7 ADC, un bypass analogique

1. Vous pouvez garde la configuration par défaut de l'ADC.
2. Créez une paire de fichiers `AnalogIn.c` / `AnalogIn.h`
3. Écrivez les fonctions suivantes :

```
void AnalogInInit(void);
void AnalogInStartConversion(void);
// Cette fonction fait du polling, même si c'est pas bien
uint16_t AnalogInGetValue(void);
```

4. Dans la routine de service d'interruption du timer, lire la valeur analogique, et la renvoyer telle quelle sur le DAC. Ça pourrait ressembler à ça :

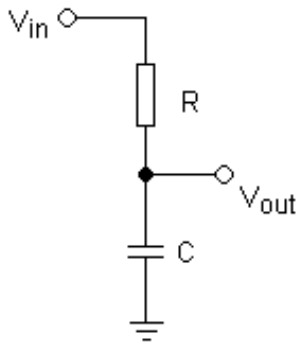
```
void TIM21_IRQHandler(void) {
    [...]

    AnalogInStartConversion();
    uint16_t ADCValue = AnalogInGetValue();
    uint16_t DACValue = ADCValue;
    AnalogOutConvert(DACValue);
}
```

5. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

3.8 Filtre RC

1. Quelle est la fréquence d'échantillonnage maximale atteignable par l'ADC ? Par le DAC ?



2. Écrivez l'équation différentielle régissant le filtre RC ci-contre. Mettre l'équation sous la forme suivante :

$$V_{in} = X \cdot \frac{dV_{out}(t)}{dt} + Y \cdot V_{out}$$

3. Écrivez l'équation de récurrence correspondante. On peut remplacer $\frac{dV(t)}{dt}$ par $\frac{V[n]-V[n-1]}{T}$. Pour faciliter le codage, on utilisera plutôt l'expression suivante :

$$f_s \cdot (V[n] - V[n-1])$$

Avec f_s la fréquence d'échantillonnage.

Mettre sous la forme suivante :

$$V_{out}[n] = \frac{A \cdot V_{in}[n] + B \cdot V_{out}[n-1]}{D}$$

4. Donnez les expressions de A, B et D. Remplacez RC par $\frac{1}{2 \cdot \pi \cdot f_c}$.
5. On travaillera avec une fréquence d'échantillonnage de 32kHz. Combien de cycles processeurs disposons-nous pour traiter chaque échantillon ?
6. Créez une paire de fichiers `RCFilter.c` / `RCFilter.h`
7. Créez la structure suivante dans `RCFilter.h` :

```c
typedef struct {
    uint32_t coeffA;
    uint32_t coeffB;
    uint32_t coeffD;
    uint16_t out_prev;
} hRCFilter_t;
```

8. Écrivez les fonctions suivantes :

```c
// Calcule les coefficients A, B et D
// Et les stocke dans la structure
void RCFilterInit(hRCFilter_t * hRCFilter,
        uint16_t cutoffFrequency, uint16_t samplingFrequency);
// Implémente l'équation de récurrence
// Faites attention au type des différentes variables
uint16_t RCFilterUpdate(hRCFilter_t * hRCFilter, uint16_t input);
```

9. Ajoutez une fonction au Shell pour modifier la fréquence de coupure.
10. Vous ne pouvez pas tester le code, n'hésitez pas à le faire valider par votre enseignant.

4 Assemblage

1. Avant de souder, faites un test de continuité. Vérifiez également qu'il n'y ait pas de courts circuits.
2. Soudez les composants :
   — Commencez par les circuits intégrés (STM32, DAC, LDO)
   — Continuez avec les petits composants (Condensateurs, résistances, LED)
   — Soudez ensuite le connecteur SWD
   — Finissez avec le connecteur 4 pin au bottom de la carte.
3. Alimentez la carte. Régler une limitation de courant de 100mA au niveau de l'alimentation.
4. Mesurer les différentes tensions d'alimentation (VDD et VDDA).
5. Branchez la STLINKV3-Mini et testez la communication, d'abord avec l'outil *stlink-gui* puis en programmant un firmware simple.
6. Testez ensuite les différents firmwares :
   (a) LED simple
   (b) LED + timer
   (c) UART
   (d) UART + Shell
   (e) DAC avec timer
   (f) Bypass ADC -> DAC. **Attention niveaux de tension !** Ne pas descendre en dessous de 0V, ne pas monter au dessus de 3,3V. Vérifier à l'oscilloscope avant d'appliquer le signal.
   (g) Filtre RC