Étude et réalisation – Semestre 4

Accordeur de Guitare

IUT De Nantes

BIZON Alexis; METAYER Simon

PEI

2017

Étude et réalisation – Semestre 4

Accordeur de Guitare

Table des matières :

[Table des matières : 1](#_Toc478561949)

[Table des Illustrations : 1](#_Toc478561950)

[0. Introduction 2](#_Toc478561951)

[0.1 Objectif 2](#_Toc478561952)

[0.2 Cahier des charges 2](#_Toc478561953)

[0.3 Travail à réaliser 2](#_Toc478561954)

[1. Accordeur de constante 3](#_Toc478561955)

[1.1 Problématique 3](#_Toc478561956)

[1.2 Solutions 3](#_Toc478561957)

[1.3 Code 5](#_Toc478561958)

[1.4 Tests 6](#_Toc478561959)

[1.5 Conclusion 6](#_Toc478561960)

[2. Accordeur de signal carré 7](#_Toc478561961)

[2.1 Problématiques 7](#_Toc478561962)

[2.2 Solutions 7](#_Toc478561963)

[2.3 Code 7](#_Toc478561964)

[2.4 Tests 8](#_Toc478561965)

[2.5 Conclusion 8](#_Toc478561966)

[3. Réalisation carte acquisition 9](#_Toc478561967)

[3.1 Problématiques 9](#_Toc478561968)

[3.2 Solutions 9](#_Toc478561969)

[3.3 Carte 11](#_Toc478561970)

[3.4 Tests 12](#_Toc478561971)

[3.5 Conclusion 13](#_Toc478561972)

[4. Accordeur de signal réel 14](#_Toc478561973)

[4.1 Problématiques 14](#_Toc478561974)

[4.2 Solutions + Code 14](#_Toc478561975)

[4.3 Code 16](#_Toc478561976)

[4.4 Conclusion 16](#_Toc478561977)

[5. Conclusion 17](#_Toc478561978)

# Table des Illustrations :

[Figure 1: Tableau des 8 premiers octaves 3](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561979)

[Figure 2: TESTS n°1 6](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561980)

[Figure 3 : Schéma micro + jack 9](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561981)

[Figure 4 : schéma filtrage 9](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561982)

[Figure 5: Schéma alimentation 10](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561983)

[Figure 6 : Schema RAM externe 10](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561984)

[Figure 7 : Schema global 11](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561985)

[Figure 8 : platine prototypage 11](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561986)

[Figure 9 : vue 3D 12](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561987)

[Figure 10 : PCB final 12](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561988)

[Figure 11 : Batterie de tests Butterworth 13](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc478561989)

# Introduction

Les 6 cordes d’une guitare sont généralement accordées comme suit : Mi La Ré Sol Si Mi. Le respect des écarts de notes entre les cordes (en demi-tons : 5 / 5 / 5 / 4 / 5) est impératif pour jouer seul. Le respect des notes absolues est impératif pour jouer à plusieurs.

## Objectif

L’objectif de ce projet est de réaliser un accordeur pour guitare électrique à base de microcontrôleur. Ce projet est proposé et encadré par P.Graziotin.

## Cahier des charges

• Affichage de la note jouée et de sa fréquence (afficheur / PC)

• Indications de la consigne de réglage de la corde (LEDs : corde trop grave / accordée / trop aigüe)

## 0.3 Travail à réaliser

• Etude de la gamme de fréquence à considérer

• Bloc diagramme fonctionnel

• Choix du microcontrôleur (pas d’Arduino à priori)

• Réalisation sur platine de prototypage (Guitare simulée par un générateur BF)

• Analyse du signal généré par les micros de la guitare

• Réalisation finale de l’accordeur sur platine de prototypage

• Eventuellement, réalisation du circuit imprimé

• Rédaction de la documentation associée au développement du projet

• Présentation du projet

# Accordeur de constante

## 1.1 Problématique

La première problématique est de réaliser un accordeur de signal non physique sur microcontrôleur, capable :

- De traiter une fréquence constante donnée lors de la compilation

- D’associer toutes les fréquences possibles à la note la plus proche et son octave associée

- D’indiquer à l’utilisateur la note la plus proche de la fréquence donnée et son octave

- D’indiquer à l’utilisateur à l’aide de 3 leds si celui-ci doit augmenter ou réduire la fréquence pour tomber pile sur la note.

## 1.2 Solutions

### 1.2.1 Calculs de notes …

Posons le décor :

Tout son musical possède une fréquence fondamentale et des harmoniques, l’emplacement du fondamental définit la note indépendamment de son amplitude. La plage de fréquence audible théorique pour un être humain adulte est de 20Hz à 20 000Hz, mais en réalité les tests d'audiométrie classiques négligent ces extrémités de la bande de fréquence, se limitant à une auscultation de la bande comprise entre 125 et 8 000 Hz. Deux notes dont les fréquences fondamentales ont un rapport qui est une puissance de deux donnent deux sons très similaires et portent donc le même nom. Cette observation permet de regrouper toutes les notes qui ont cette propriété dans la même catégorie de hauteur (do, ré, mi …).

Dans la musique occidentale, les catégories de hauteurs sont au nombre de douze. Sept d'entre elles sont considérées comme les principales et ont pour noms : do, ré, mi, fa, sol, la et si. L'intervalle compris entre deux hauteurs dont la fréquence de l'une vaut le double (ou la moitié) de l'autre s'appelle une octave. Pour distinguer deux notes de même nom dans deux octaves différentes, on numérote les octaves et donne ce numéro aux notes correspondantes : par exemple, le LA3 ; le 3 indique l’octave et le LA, la hauteur, celui-ci correspond à 440Hz. Cette fréquence de référence est donnée par un diapason. Nous utiliserons régulièrement cette notation dans ce rapport.

Dans la gamme tempérée (comprenez les 8 premiers octaves), la formule permettant de mesurer la fréquence d'une note par rapport à une note de départ est :  Avec  le nombre de demi-tons au-dessus de la note de départ .

Exemple avec et on obtient ce qui correspond bien à deux cases après le LA3 () soit un SI3.

Ce calcul nous permet donc de compléter le tableau ci-dessous :

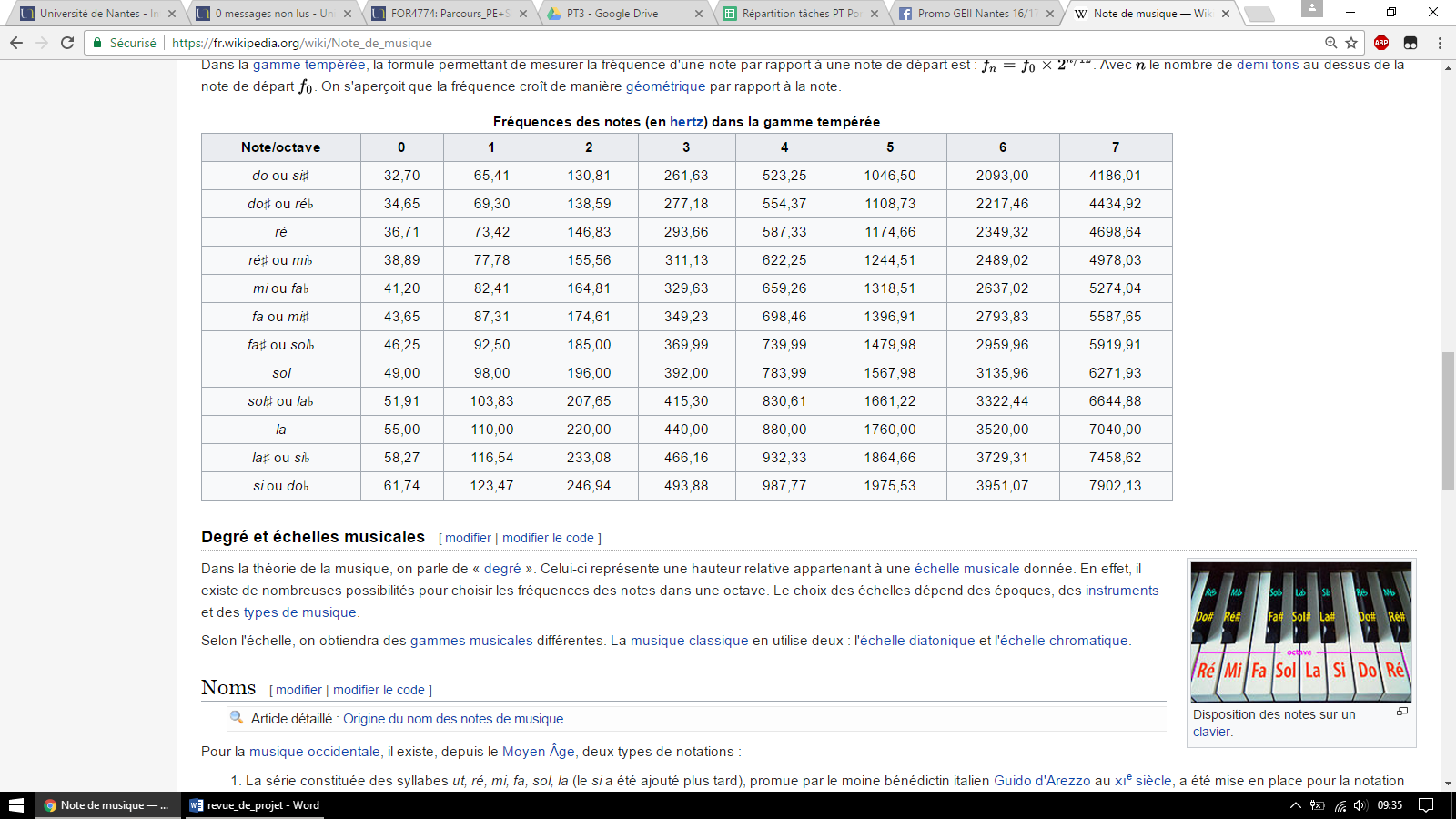


Figure 1: Tableau des 8 premiers octaves

NB : Le calcul fonctionne avec un négatif pour remonter les colonnes et peut également sauter les colonnes Exemple avec et on obtient On obtient bien la note 12 cases avant soit un LA2.

Ce calcul nous permet donc de trouver toutes les notes à partir du seul LA3 440Hz, ce qui évite la multiplicité des constantes dans notre code.

De plus, ces équations nous permettent de récupérer la note la plus proche associée à une fréquence :

On part de :  Et on exprime en fonction de et soit :

Exemple : On cherche à accorder une fréquence de 99Hz soit presque un SOL1 qui normalement vaut 98Hz, on applique donc la formule :

Il suffit ensuite d’arrondir à l’unité la valeur trouvée pour déduire le rang relatif au LA3. On obtient donc -26, en regardant le tableau de notes calculées précédemment on compte -26 cases avant le LA3 et on obtient bien un SOL1 98Hz.

Ce calcul nous permet donc de déduire quelle est la note la plus proche de la fréquence envoyée avec un rang relatif au LA3 fondamental.

### 1.2.2 Choix du microcontrôleur

Nous choisissons d’utiliser un SAMD21J18A monté sur un carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) pour plusieurs raisons :

- La puissance de calcul ; en effet celui-ci tourne à 48MHz contre les 16MHz d’une [Arduino](https://www.arduino.cc/en/Main/arduinoBoardMega2560) classique ce qui risque d’être nécessaire pour les calculs de FFT ou DFT que nous étudierons ultérieurement.

- La présence d’un [module OLED](http://www.atmel.com/tools/ATOLED1-XPRO.aspx) à l’IUT permettant un affichage sur écran avec 4 lignes, et également la présence de 3 Leds, que nous utiliserons pour l’affichage trop bas/bien/trop haut, des afficheurs classiques.

- Notre connaissance de l’appareil, car nous utilisons celui-ci depuis plusieurs projets déjà ce qui permet une réutilisation des bibliothèques de bases.

### 1.2.3 Portabilité du projet

Nous avons choisi, pour intégrer un nouvel outil professionnel et gérer plus efficacement notre code d’utiliser [GitHub](https://github.com/). [GitHub](https://github.com/) est un service web d'hébergement et de gestion de développement de logiciels, utilisant le logiciel de gestion de versions Git. Le site assure également un contrôle d'accès et des fonctionnalités destinées à la collaboration comme le suivi des bugs, les demandes de fonctionnalités, la gestion de tâches et un wiki pour chaque projet.

Tout le projet sera donc par la suite référencé directement par des liens comme celui-ci : [PROJET](https://github.com/albother/ER4_Accordeur) menant directement sur GitHub à l’endroit où est hébergé notre code. La plupart des références mènerons directement à la lige de code concernée.

## 1.3 Code

Nous commençons par déclarer [ICI](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/include/define_notes.h#L23) toutes les notes avec la méthode de calcul décrite ci-dessus basé sur le LA3 à 440Hz. La définition est très calculatoire mais permet de ne disposer que d’une SEULE constante dans tout le système de note.

Nous ajoutons également des définitions conditionnelles [ICI](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/include/define_notes.h#L43) qui renvoient la fréquence associée à une note et à son degré, donc si on écrit SOL(5) quelque part dans le projet celui-ci nous renvoie la fréquence calculée soit précisément : 1567.98Hz

Maintenant que les définitions basiques sont posées, nous allons gérer l’affichage. Les fonctions associées seront codées dans un fichier [display.c](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/display.c). Ce fichier dispose de 4 fonctions d’affichage brut sur le LCD, displayTitle, displayNote, displayFrequence et displayDegre, chaque paramètre est affiché sur sa propre ligne. Il dispose également d’une fonction diplayLedIndicator, qui allume les leds du module OLED en fonction de la correspondance de la note ciblée avec la fréquence envoyée

Nous réalisons par la suite la fonction [noteSolver](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/note.c#L29) qui contient simplement le calcul expliqué précédemment, et permet de trouver la note la plus proche à la fréquence choisie parmi DO, RE,…

S’en suit la réalisation de la fonction [degreSolver](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/note.c#L61) qui, à partir du même calcul renvois le degré/octave associé à la fréquence choisie.

Ces deux fonction Solver nous permettent de « résoudre » la fréquence et de calculer la note et le dergé/octave

Exemple : avec la fréquence arbitraire 49Hz, on obtient un SOL en utilisant noteSolver, et un 0 en utilisant degreSolver, ce qui nous permet par la suite d’afficher SOL0. NB : On aurait obtenu la même chose avec 48Hz car notre Solver « comprend » quel est la note la plus proche d’après le tableau en : Figure 1.

Pour finir nous réalisons la fonction [accorder](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/note.c#L150) qui déduit à partir de la fréquence envoyée, et des fréquences connues du tableau (Figure 1), si la fréquence est trop haute, trop basse, ou correcte. Nous introduisons aussi une erreur relative en %. En effet, pour l’instant nous imposons la fréquence au système en tant que constante (ici 49Hz), mais par la suite, celle-ci sera mesurée et forcément imprécise, or notre système ne définira jamais la note comme accordée si celui-ci vise 49Hz pile. (On pourrait se retrouver avec des situations ou l’instrument joue 49,0001Hz et l’accordeur considère celle-ci fausse). On peut donc par exemple considérer que de 48,9 à 49,1 la note est correctement accorée, et aucune oreille humaine ne saurait faire la différence.

Pour cette version d’accordeur de « constante » la fréquence est définie dans le main avant la compilation, le projet ne dispose pas de système d’acquisition de fréquence pour le moment.

Le main qui utilise les fonctions décrites précédemment avec ici 49Hz de fréquence :

/\*

\* Accordeur de Guitare

\* Authors : METAYER Simon & BIZON Alexis

\* Created Date : 23/01/17

\* Version : 1.0

\*/

#include "define.h" // inclusion des fichiers contenant toute les fonctions

int main**(**void**)**//fonction principale du programme

**{**

float freq **=** 49**;**//contient la fréquence mesurée de la note

char note **=** 0**;**//contient la note calculée à partir de la fréquence, voir les différentes valeurs dans define\_notes.h

char degre **=** 0**;**//contient le degré, ou l'octave de la note calculée à partir de la fréquence

float relativeError **=** 1.0**;**//spécifie l'erreur relative autorisée en % entre la fréquence mesurée et la fréquence réelle correspondant à la note

setup**();**//Fonction d'initialisation

displayTitle**();**//affiche le titre du projet en haut de l'écran OLED

**while(**1**)**

**{**

note **=** noteSolver**(**freq**);**//permet de déterminer la note correspondant à la fréquence mesurée

degre **=** degreSolver**(**freq**);**//permet de déterminer le degré ou l'octave correspondant à la fréquence mesurée

char acc **=** accorder**(**note**,** degre**,** freq**,** calculAbsError**(**relativeError**,** freq**));**//Permet de déterminer si la note est accordée, trop basse ou trop haute

diplayLedIndicator**(**acc**);** // Allume les leds en fonciton de l'indicateur : trop bas / ok / trop haut

displayFrequence**(**freq**);**//On affiche la fréquence sur l'afficheur OLED

displayNote**(**note**);**//On affiche la note sur l'afficheur OLED

displayDegre**(**degre**);**//On affiche le degré sur l'afficheur OLED

**}**

**return** 0**;**//Le programme renvoi 0x0000 si l'exécution c'est correctement terminée

**}**

## 1.4 Tests

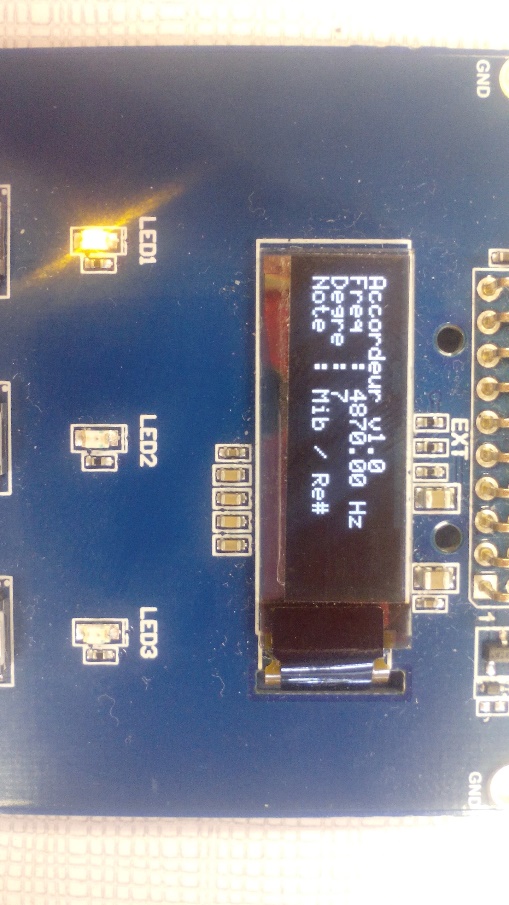
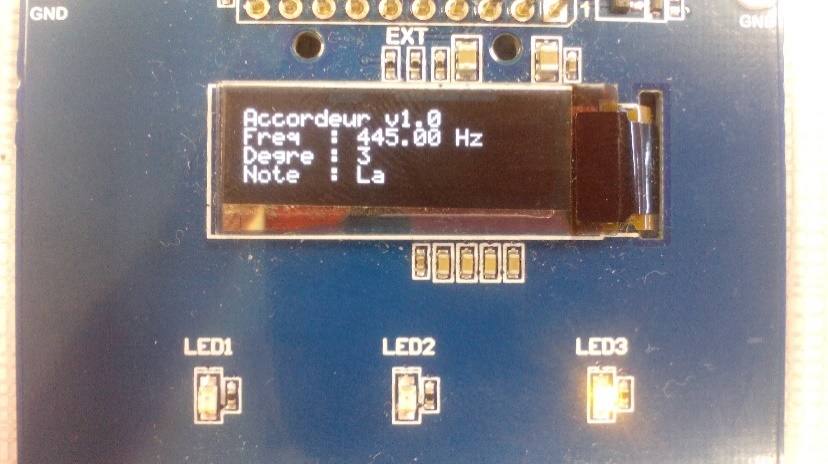


Figure 2: TESTS n°1

Premier test : Avec la fréquence 49Hz

Le calcul théorique donne : et correspond bien à un SOL0. (Voir Figure 1).

La note est donc correctement accordée c’est la LED2 du milieu qui s’allume

Second test : Avec la fréquence à 445Hz

Le calcul théorique donne : et correspond bien à un LA3. (Voir Figure 1).

Seulement la fréquence ici est 5Hz trop haut, c’est donc la LED3 de droite qui s’allume pour indiquer que la note doit être descendue. La LED3 s’éteindra vers 443Hz.

Troisième test : Avec la fréquence à 4870Hz

Le calcul théorique donne : et correspond bien à un MIb7. (Voir Figure 1).

Seulement la fréquence est 8,3Hz trop basse, c’est donc la LED1 de gauche qui s’allume pour indiquer de remonter la fréquence.

## 1.5 Conclusion

Nous avons donc réalisé un accordeur de base dans sa partie fonctionnelle et IHM. De plus nous avons réalisé tous les calculs mathématiques permettant une interprétation « musicale » (octaves, notes etc…) des grandeurs et une traduction suivant les normes de la musique (Ex : LA3, SI2 …) . Toute la partie Accordeur à proprement parlé est donc finalisée (sous réserve d’améliorations). Nous allons pouvoir passer à la partie acquisition d’un signal et traitement de celui-ci en procédant par étapes.

# 2. Accordeur de signal carré

Pour procéder par étapes dans l’acquisition et le traitement du signal nous allons d’abord faire l’acquisition d’un signal carré et tenter de l’accorder correctement.

## 2.1 Problématiques

* Acquisition du signal carré.
* Traitement du signal carré pour extraire la fréquence
* Accorder le signal carré.

## 2.2 Solutions

### 2.2.1 Acquisition

Utiliser un signal carré présente un avantage majeur. Celui-ci n’a pas forcément besoin d’être traité en ADC (Analogic to Digital Converter). Il suffit simplement de le passer dans une entrée du microcontrôleur contenant un traitement EIC (External Interrupt Controler) et ainsi chaque front montant (ou descendant ; au choix) génèrera une interruption système.

Nous précisons ici que le signal carré sera évidemment absolu, et sera donc compris entre 0v et +3.3v pour le bien de notre microcontrôleur samd21, qui n’accepte pas les tensions négatives dans cette configuration. (Nous résoudrons cette anomalie par la suite pour les signaux positifs ET négatifs)

### 2.2.2 Fréquencemètre

Pour Mesurer la fréquence, on lance un timer lors de l’interruption générée par le 1er front, qui comptera (en ticks) le temps qui passe, puis on arrête le timer au front suivant. Ainsi on connait le temps passé (en ticks) entre deux fronts montants.

Pour connaitre le « vrai temps » passé entre deux interruptions il suffit de savoir quel est le « vrai temps » en secondes passé entre deux ticks (soit la période du timer) et de le multiplier au nombre de ticks :

De ce fait nous pourrons par un simple obtenir la fréquence du signal carré.

Et on recommence l’opération à chaque front.

## 2.3 Code

Nous commençons par réaliser la fonction [initFreq](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/freq.c#L22) qui initialise dans l’ordre :

* La pin PB15 en entrée car celle-ci dispose d’un EIC, nous allons donc l’utiliser pour récupérer notre signal physique.
* Nous configurons donc cette pin en EIC, en mode « RISING » (détection de fronts montants) et nous définissons celle-ci comme étant une source d’interruption. Donc, par la suite, à chaque front montant de notre signal carré, c’est la fonction [EIC\_Handler](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/freq.c#L47) qui sera appelée et exécutée.
* Nous définissons également ici un Timer (TC3) en incrémentation sur une horloge de 1MHz avec un diviseur par 8 soit donc une période (temps en secondes entre deux ticks) de

Dans la fonction [EIC\_Handler](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/freq.c#L47) nous effectuons les étapes décrites précédemment avec un système d’if else pour inverser l’état a chaque passage d’un front ;

FRONT 1 => lance le timer

FRONT 2 => arrête le timer

FRONT 3 => lance le timer etc …

Soit un code qui deviens :

FRONT X => si le timer tourne on l’arrête, s’il ne tourne pas on le lance.

Et pour finir calcul la fréquence [ICI](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/freq.c#L60), soit a chaque arrêt du timer, avec le calcul décrit précédemment :

Ou ici la période du timer TC3 vaut 8us.

Exemple : On détecte un front, le timer se lance, il se passe 100ticks et hop, second front, on arrête le compteur, on calcule la fréquence : = 12500Hz soit une fréquence du signal carré de 12.5KHz

La dernière fonction sert juste pour transférer la fréquence vers l’utilisateur. En effet si le signal à une fréquence élevée nous allons faire tous ces calculs très rapidement et l’utilisateur n’en a pas forcément besoin. Nous plaçons donc la fonction très simple [getFreq](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/freq.c#L73) qui lorsqu’elle est appelée vient lire la dernière fréquence calculée.

On vient donc se servir de cette fonction dans le main pour récupérer la fréquence.

Celui-ci devient donc (un seul changement) ;

/\*

\* Accordeur de Guitare

\* Authors : METAYER Simon & BIZON Alexis

\* Created Date : 23/01/17

\* Version : 1.0

\*/

#include "define.h"

int main**(**void**)**//fonction principale du programme

**{**

float freq **=** 0**;**//contient la fréquence mesurée de la note

char note **=** 0**;**//contient la note calculée à partir de la fréquence, voir les différentes valeurs dans define\_notes.h

char degre **=** 0**;**//contient le degré, ou l'octave de la note calculée à partir de la fréquence

float relativeError **=** 1.0**;**//spécifie l'erreur relative autorisée en % entre la fréquence mesurée et la fréquence réelle correspondant à la note

setup**();**//Fonction d'initialisation

displayTitle**();**//affiche le titre du projet en haut de l'écran OLED

**while(**1**)**

**{**

==========🡺 freq**=**getFreq**();**//Récupère la fréquence mesurée par le fréquencemètre

note **=** noteSolver**(**freq**);**//permet de déterminer la note correspondant à la fréquence mesurée

degre **=** degreSolver**(**freq**);**//permet de déterminer le degré ou l'octave correspondant à la fréquence mesurée

char acc **=** accorder**(**note**,** degre**,** freq**,** calculAbsError**(**relativeError**,** freq**,**0**));**//Permet de déterminer si la note est accordée, trop basse ou trop haute

diplayLedIndicator**(**acc**);** // Allume les leds en fonciton de l'indicateur : trop bas / ok / trop haut

displayFrequence**(**freq**);**//On affiche la fréquence sur l'afficheur OLED

displayNote**(**note**);**//On affiche la note sur l'afficheur OLED

displayDegre**(**degre**);**//On affiche le degré sur l'afficheur OLED

**}**

**return** 0**;**//Le programme renvoi 0x0000 si l'exécution c'est correctement terminée

**}**

A partir de là on revient sur le même fonctionnement que précédemment et tout reste inchangé ; on vient lire la fréquence, analyser les notes et faire un retour à l’utilisateur sur l’accordage etc …

Les tests sont réalisés avec un GBF et sont très concluants.

## 2.5 Tests / réflexion / conclusion

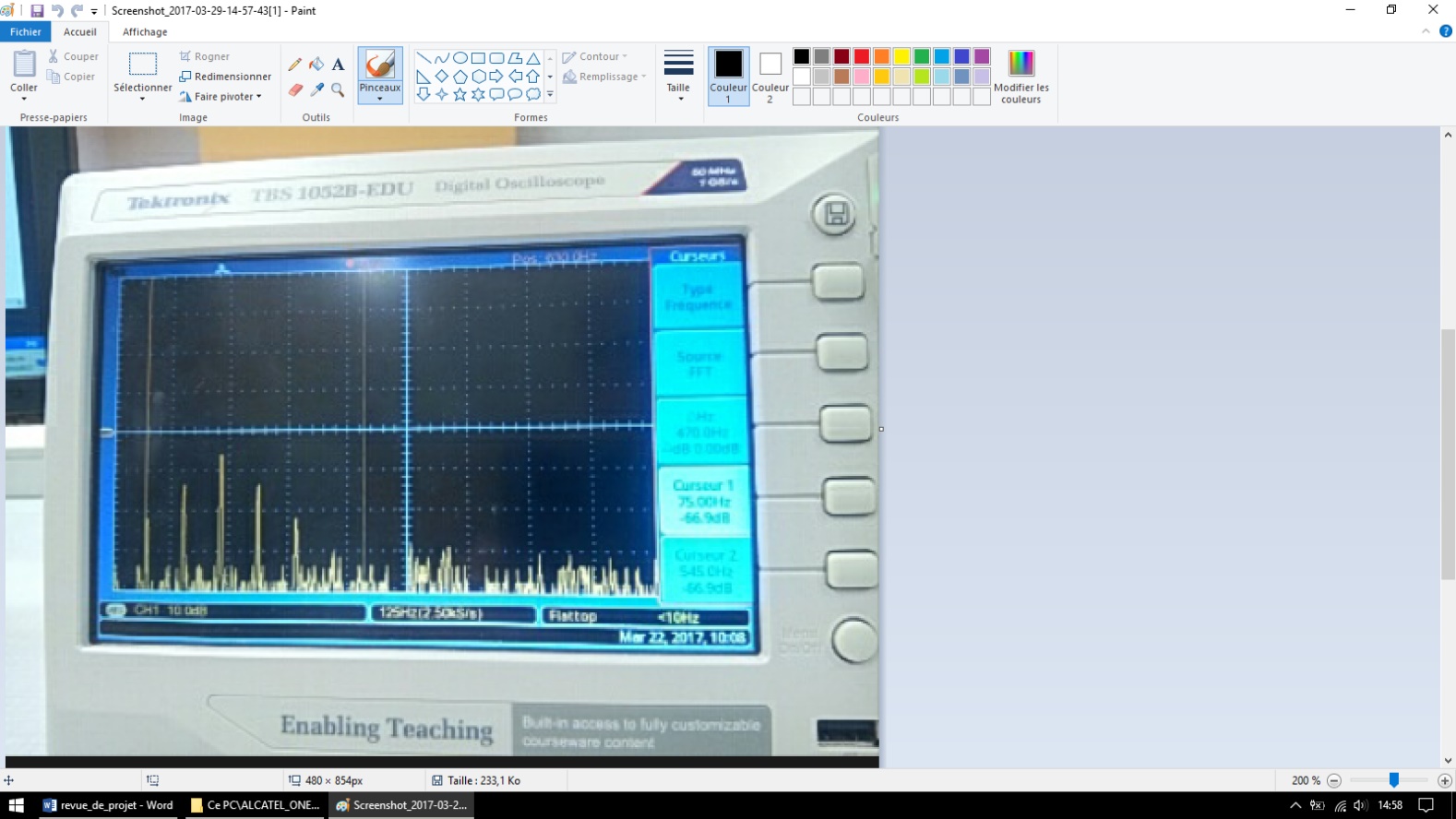


Figure 3 Harmoniques mi grave

Nous sommes donc désormais capables d’accorder un signal carré et avons certifier le fonctionnement de notre algorithme d’accordage d’instrument. A cet instant deux solutions s’offrent à nous.

- Option 1 : nous poursuivons de la sorte et tentons de traiter le signal de guitare pour qu’il ressemble à un signal carré. En effet pourquoi pas, il suffit de passer le signal audio dans un filtre passe bas pour retirer le bruit HF et ensuite de passer le signal dans un ampli AOP qui multiplie par 10000 et le tour est joué. Le « sinus » sera transformé en signal carré et il suffit d’utiliser le code précédent. Cette solution parait simple et efficace.

Mais en réalité ne fonctionne pas pour toute les notes. En effet nous avons fait quelques recherches et il apparait que pour beaucoup d’instruments, la décomposition harmonique du son présente de très forts harmoniques voir même plus fort que le fondamental. Nous avons-nous même fait le test ci-contre avec un mi grave. Donc, la période du signal temporel peut être déformée par ces harmoniques et donc l’option 1 n’est plus viable. Il faut donc envisager une seconde option.

* Option 2 : récupérer le signal temporel tel quel, faire une FFT et détecter le fondamental.

Cette solution est beaucoup plus complexe à mettre en œuvre mais parait à nos yeux la seule qui a de chances de fonctionner correctement. C’est donc dans cette voie que nous allons poursuivre.

# 3. Réalisation carte acquisition

Pour la suite de notre projet, nous allons réaliser une carte d’extension pour la carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) afin de réaliser toutes les fonctions manquantes pour notre système d’accordeur.

## 3.1 Problématiques

- Acquisition du signal sonore par microphone (pour guitares sèches et autres instruments)

- Acquisition du signal sonore par jack (pour guitares électriques)

- Filtrage analogique du signal sonore

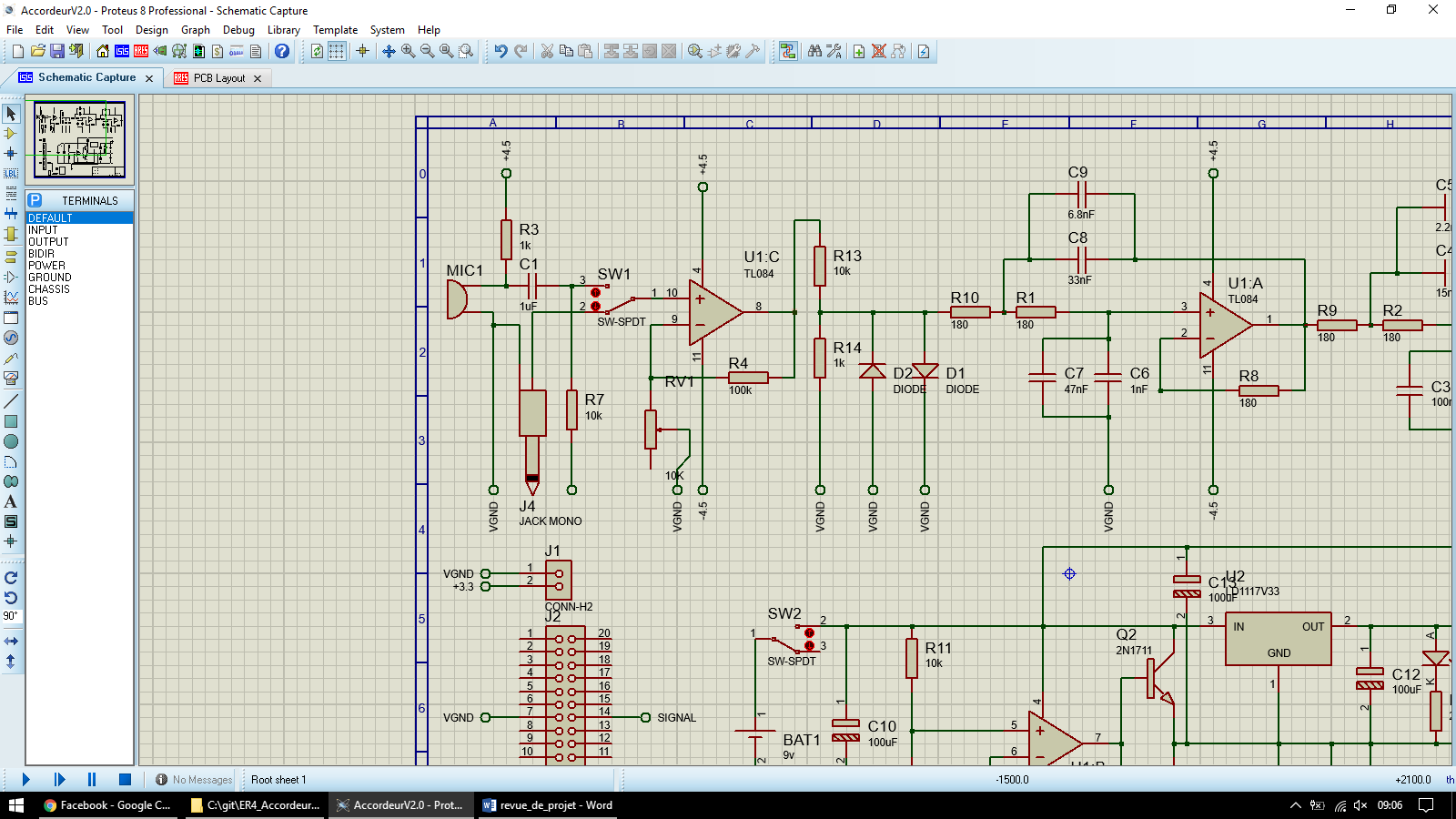
- Alimentation portable de tout le système et création des tensions nécessaires ( -4.5, +4.5, +3.3v)

- Ajout de RAM externe pour contenir de plus gros tableaux de FFT/DFT.

## 3.2 Solutions

### 3.2.1 Bloc Acquisition

Figure 4 : Schéma micro + jack



Pour l’acquisition du signal sonore nous choisissons deux voies, une simple pour accorder une guitare électrique, soit une jack femelle 2.5 (représenté ci-contre par une jack mâle) et une seconde pour étendre la gamme d’instruments que notre accordeur est capable de gérer nous choisissons d’intégrer un microphone à notre carte. Nous avons donc récupéré un [Electret](https://en.wikipedia.org/wiki/Electret_microphone), et branché comme indiqué dans la documentation. Nous avons apporté quelques modifications personnelles eu montage. Nous avons passé la taille du condensateur de 47µF à 1µF pour être sûr d’éliminer complètement la composante continue du microphone. Nous avons également ajouté une résistance de pull-down R7 pour charger le condensateur plus efficacement. Et pour finir un Switch de sélection (jumper) permettant le choix entre l’acquisition par micro et celle par jack. Schéma ci-contre.

### 3.2.2 Bloc Filtrage

A la suite du switch et avant d’envoyer le signal dans la carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) nous choisissons de le traiter analogiquement. Ici nous répondons à plusieurs problématiques à la suite, de gauche à droite sur le schéma ci-dessous :

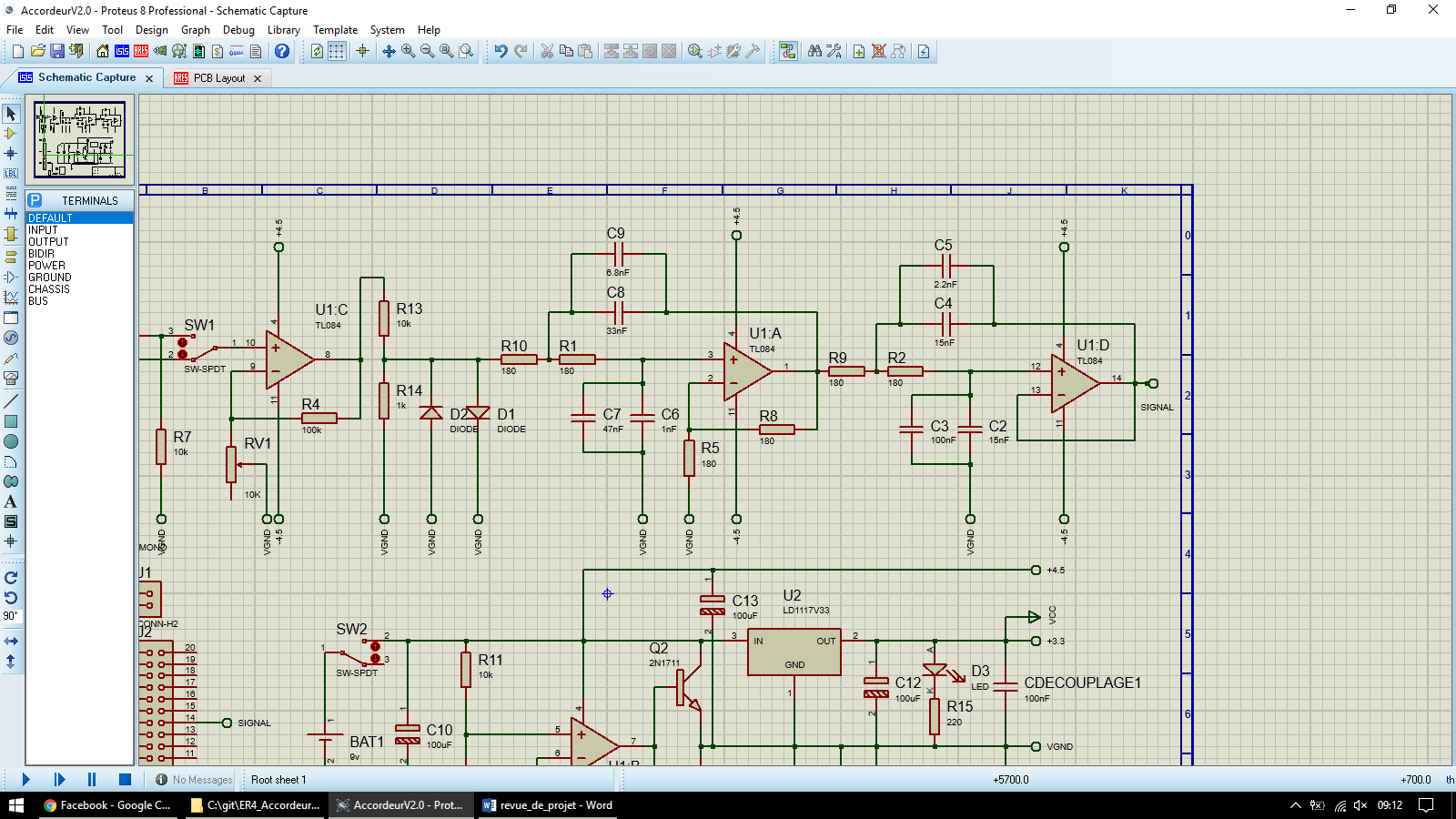


Figure 5 : schéma filtrage

Tout d’abord nous amplifions le signal par 100 avec un montage AOP simple (U1C), et nous le divisions ensuite par 10 avec un pont diviseur de tension. Ces deux fonctions de suites permettent simplement de multiplier par mais en écrêtant le signal aux tensions d’alimentations de l’AOP divisés par 10 soit . En effet l’ADC présent sur la carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) n’accepte que des tensions de . Nous avons tout de même ajouté à la suite un écrêteur à diodes qui lui, découpe aux tensions de seuils de en sécurité supplémentaire. La suite du montage avec U1A et U1D est simplement un filtre passe bas Butterworth d’ordre 4 de fréquence de coupure 20kHz calculé avec une méthode de matrice de coefficients ;

On calcule d’abord la valeur d’un pseudo condensateur référence avec la formule :

Donc

A partir de là on calcule les vraies valeurs des condensateurs en multipliant C0 par des coefficients :

Ces calculs sont issus de ce [site](http://www.electronique-3d.fr/Le_Filtre_passe_bas.html).

-C9+C8=C0\*1.0824=40nF  
-C7+C6=C0\*0,9239=50nF  
-C5+C4=C0\*2,6130=17nF  
-C2+C3= C0\*0,3827=110nF

On obtient les valeurs précédentes approximées.

### 3.2.3 Bloc Alimentation

Ici nous répondons aux problématiques d’alimentations.

Nous avons besoin pour notre montagne de 4 AOP, nous choisissons donc d’utiliser un [TL084](https://www.egr.msu.edu/eceshop/Parts_Inventory/datasheets/tl084cn.pdf). Nous avons vu précédemment que nous aurons besoin d’une alimentation symétrique en +4.5 et -4.5. La carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) quant à elle fonctionne en +3.3. Voici le schéma que nous proposons répondant à cette problématique :

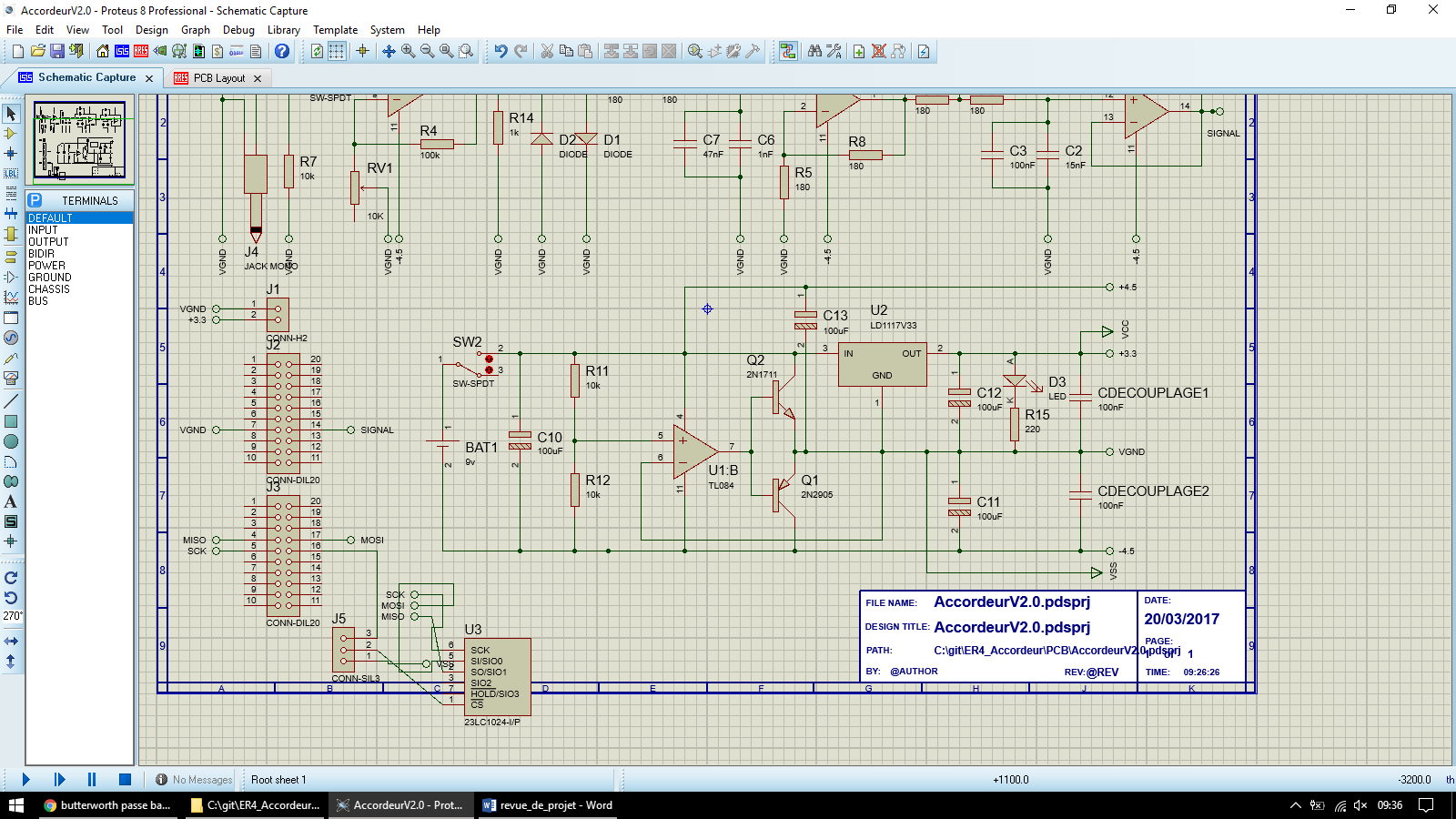


Figure 6: Schéma alimentation

Nous choisissons d’utiliser une pile 9v, car après symétrisation nous obtiendrons . L’AOP U1B sert à créer une masse virtuelle (VGND) et à symétriser l’alimentation en nous ajoutons également un régulateur [LD1117v33](http://fr.farnell.com/stmicroelectronics/ld1117v33/regulateur-ldo-3-3v-1117-to-220/dp/9755837) qui est un régulateur 3.3v que nous branchons sur le +4.5v pour limiter la chute de tension dans le régulateur. Nous disposons maintenant de toutes les tensions nécessaires, le +3.3v en sortie du régulateur, le +4.5v entre VGND et +4.5 et le -4.5v entre le VGND et le -4.5. Pour ce qui est des gestions de courants nous ajoutons un étage d’amplification entre l’AOP et le régulateur avec deux transistors PNP et NPN : Q2 ([2n1711](http://fr.farnell.com/multicomp/2n1711/transistor/dp/1611558)) et Q1 ([2n2905](https://www.elektronik-kompendium.de/public/schaerer/FILES/2n2905.pdf)). Nous ajoutons 4 condensateurs de découplages polarisés (C10,11,12,13) et 2 condensateurs de découplages de 100nF, qui sont en fait les découplages du [TL084](https://www.egr.msu.edu/eceshop/Parts_Inventory/datasheets/tl084cn.pdf). Pour finir nous ajoutons une LED témoin d’alimentation avec sa résistance de limitation de courant de 220Ω.

### 3.2.4 Bloc RAM

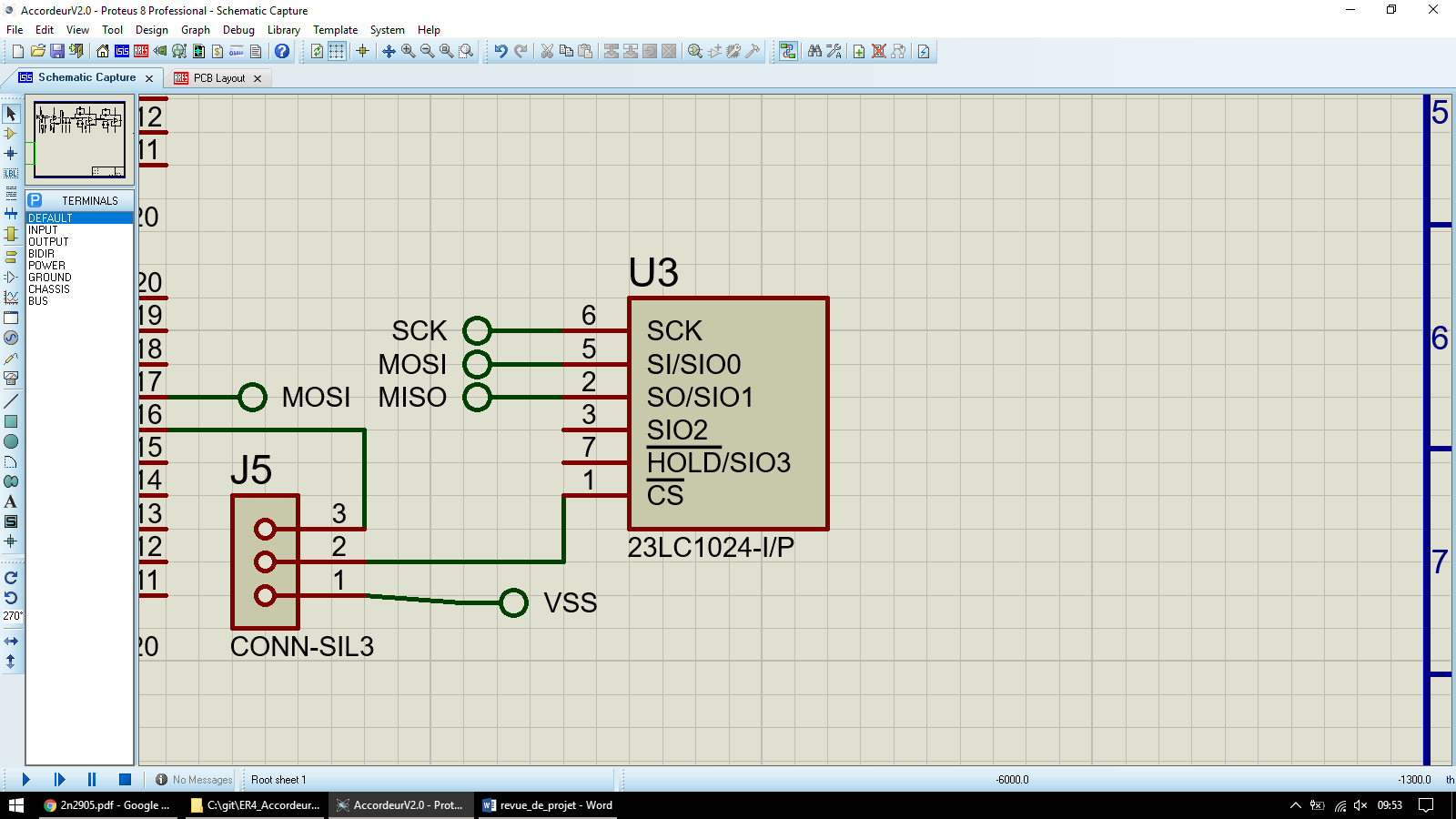


Figure 7 : Schema RAM externe

Dans l’optique d’effectuer des calculs de FFT/DFT performants nous avons choisi d’ajouter une [RAM](http://ww1.microchip.com/downloads/en/DeviceDoc/20005142C.pdf) externe à notre PCB

Le module [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) contient un [ATSAMD21J18](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx?tab=devices) qui est un ARM Cortex-M0+ avec 32KBytes de RAM, qui tourne à 48 MHz, avec 64 pins. La RAM interne contient donc 32 000\*8=256 000 cases disponibles pour stocker le tableau du signal acquis par l’ADC. Seulement notre tableau est en int16\_t, donc il n’y a finalement que Valeurs stockables pour l’ADC. Et encore cette valeur est plus petite car il faut soustraire toutes les petites variables nécessaires au fonctionnement du programme. Pour faire fonctionner notre programme correctement et atteindre la précision nécessaire de 1Hz nous devrions stocker 44 000 valeurs 16 bits environ.

Nous avons donc choisi d’ajouter un module RAM réf : [23LC1024](http://www.microchip.com/wwwproducts/en/23LC1024) qui ajoute 1024Kbits de RAM à notre Accordeur, soit un espace de valeurs 16 bits stockables. Ce qui devrait être largement suffisant pour stocker les 44 000 valeurs 16bits + toutes les petites variables annexes.

## 3.3 Carte

### 3.3.1 Schéma global

Nous ajoutons les connecteurs vers la carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) et nous obtenons donc le schéma global suivant :

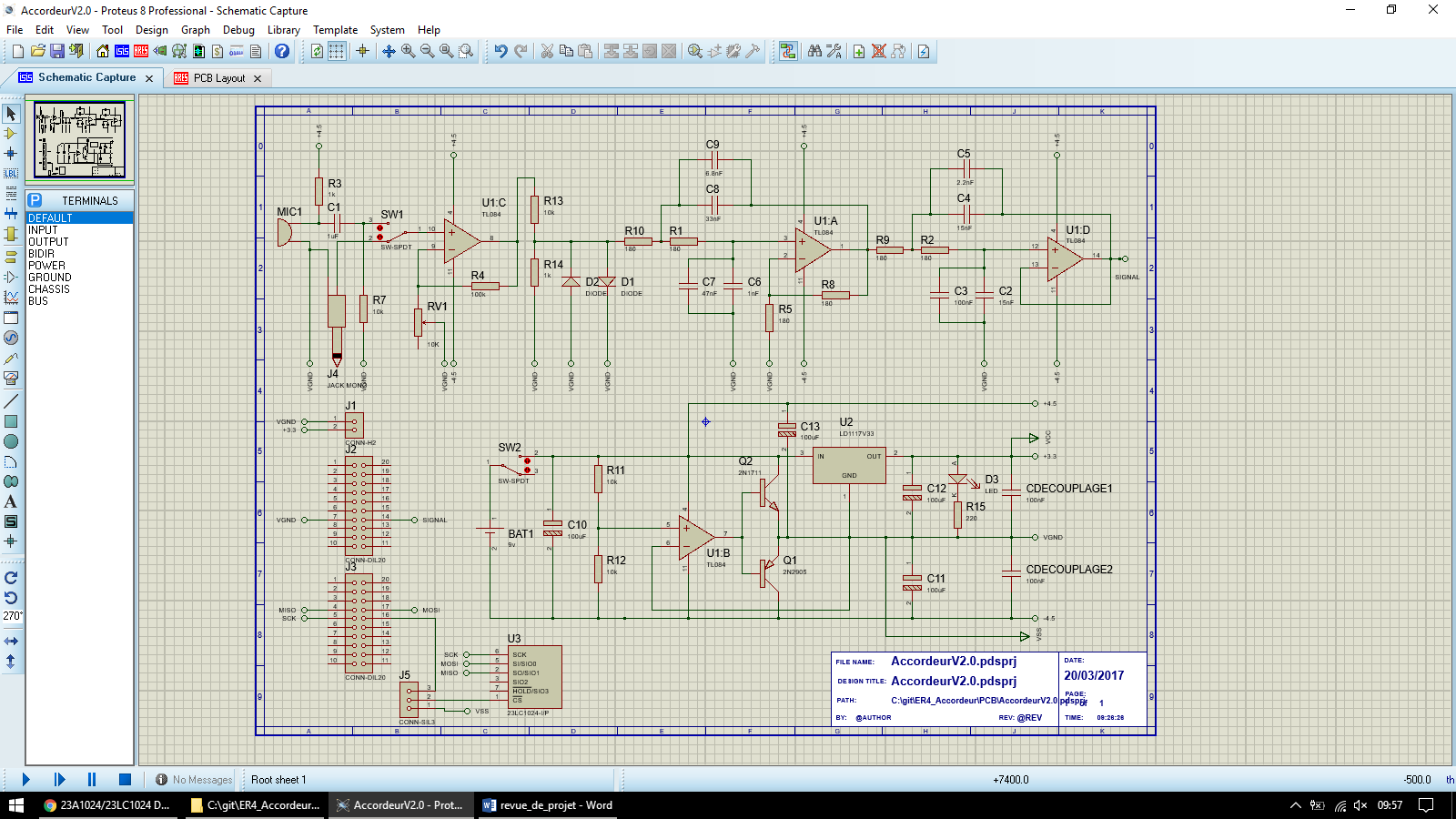


Figure 8 : Schema global

### 3.3.2 Tests sur platine de prototypage

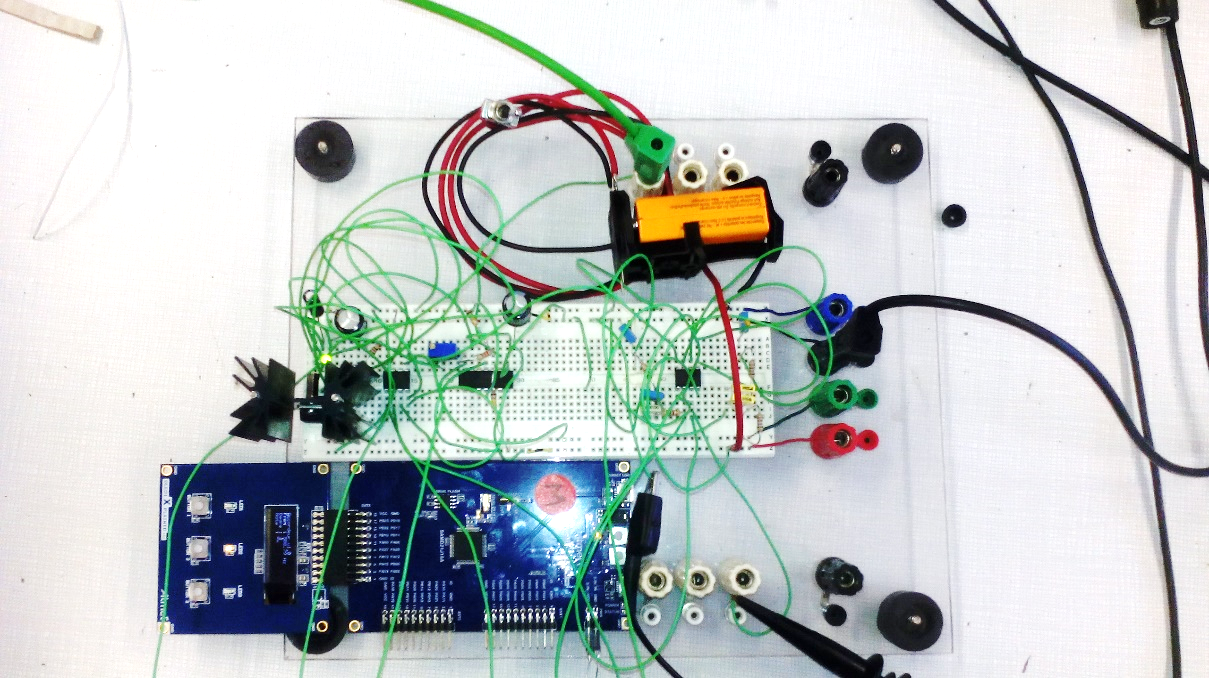


Figure 9 : platine prototypage

Sur cette image on peut voir la phase de tests du schéma ci-dessus sur platine de prototypage. Sur la droite on peut observer la carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) qui est allumée, car alimentée par la pile et donc par le biais de l’alimentation différentielle et par la suite du régulateur. Cette image atteste donc du bon fonctionnement du module d’auto alimentation de notre accordeur de guitare.

Sur cette plaque lab nous avons également réalisé de nombreux tests d’acquisition par microphone afin d’avoir le bon écrêtage du signal sonore.

Il est évident de constater qu’il est grand temps de passer à la réalisation d’un PCB !

### 3.3.3 Réalisation de la carte

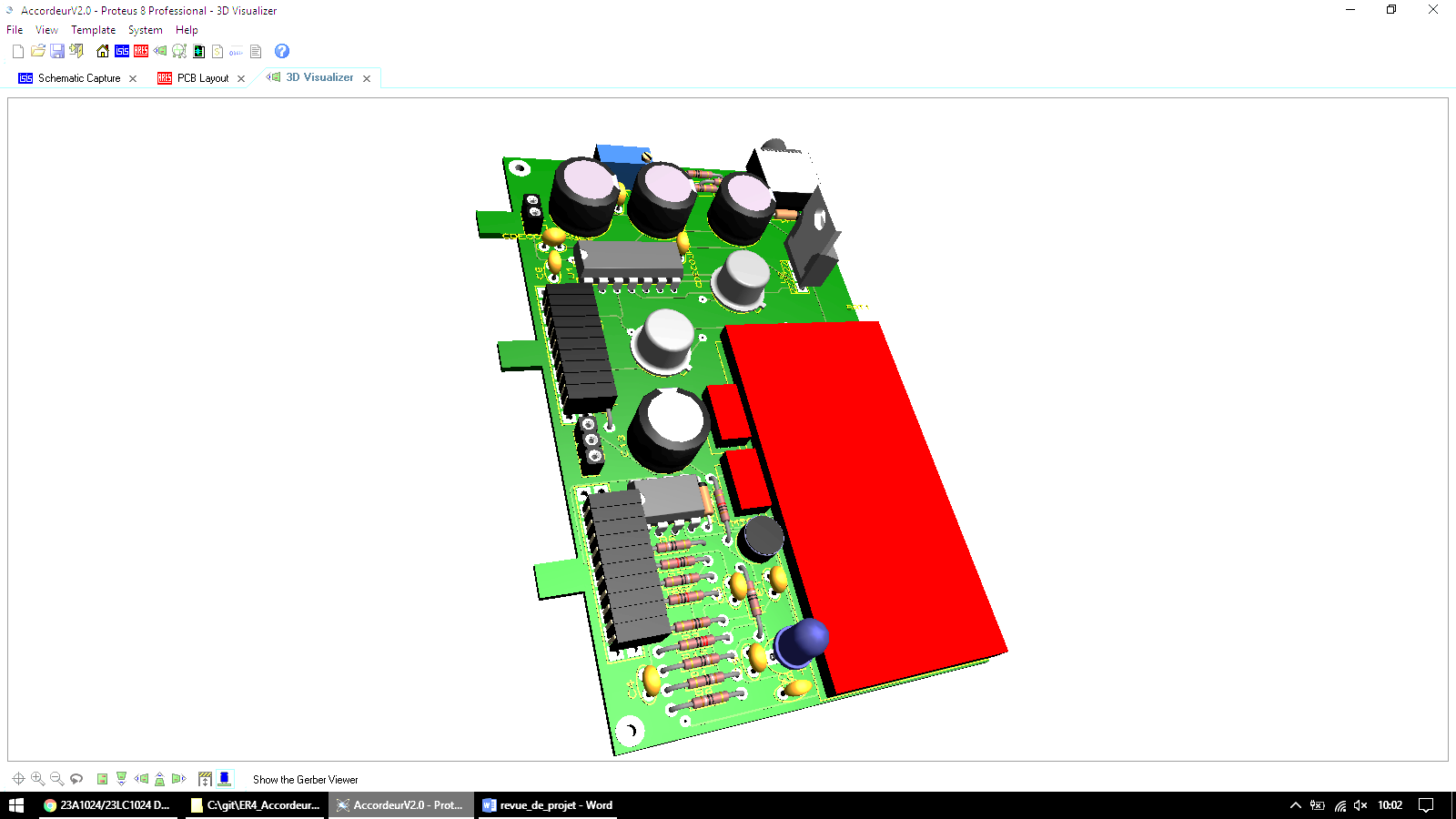
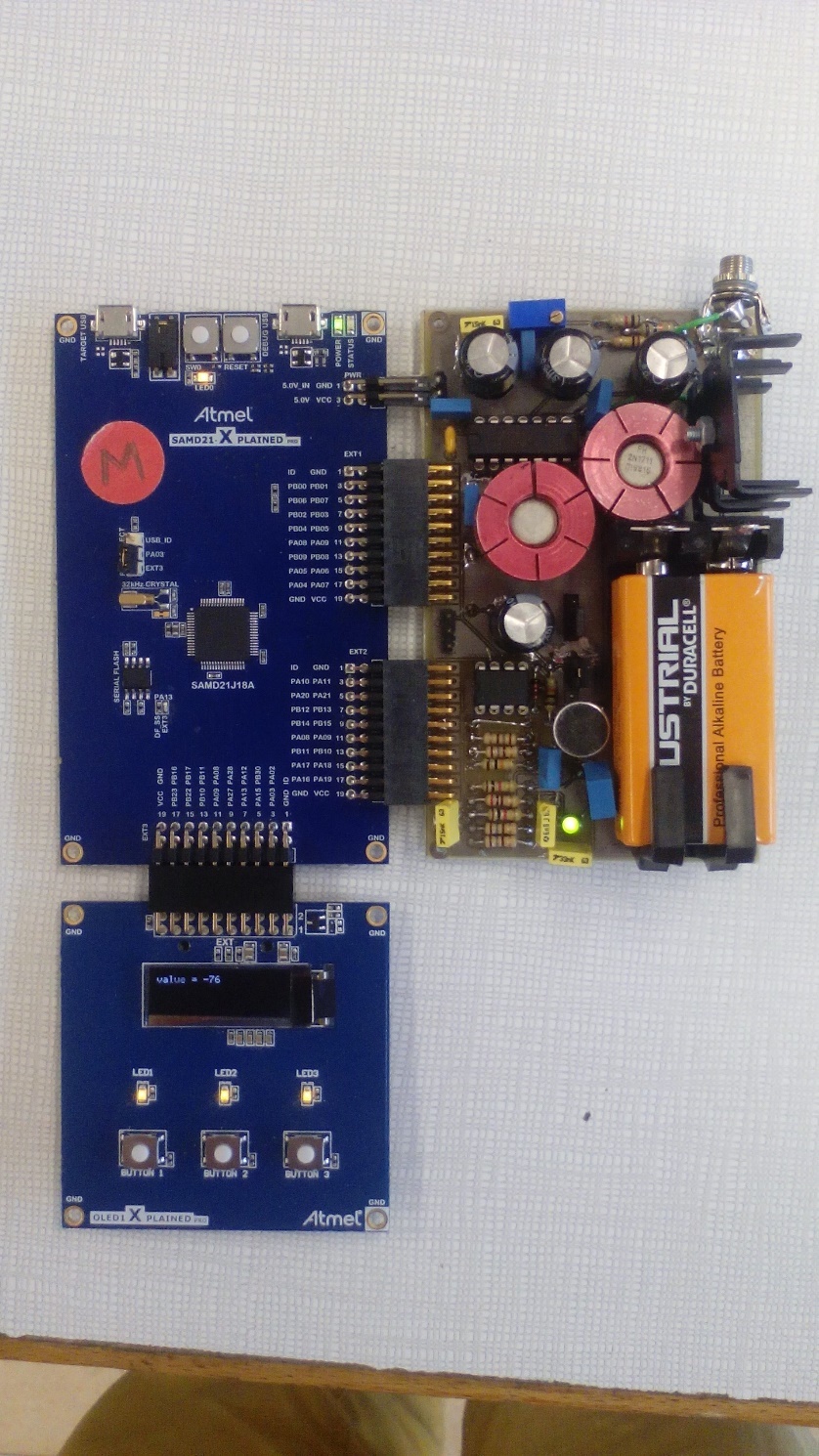


Figure 10 : vue 3D

Nous passons donc maintenant à la réalisation de la carte. Nous choisissons de réaliser un PCB de la même taille que la carte [Xplained-pro](http://www.atmel.com/tools/ATSAMD21-XPRO.aspx) (sans l’OLED) pour des raisons esthétiques. Nous obtenons la vue 3D ci-contre après routage et placement des composants avec la suite [Proteus](https://www.labcenter.com/) (qui n’est pas gratuite mais que nous utilisons quand même par question d’habitude)

Le routage nous a posé beaucoup de petits problèmes du fait de nombreux composants inconnus des bases de données classiques. Nous avons dû réaliser des empruntes sur mesure pour de nombreux composants, tels que la pile 9v ou encore le jack femelle. Nous avons dû également anticiper la présence de radiateurs sur le PCB au niveau des deux transistors et du régulateur.

Pour finir nous obtenons ce rendu :



Pile 9v

Entrée microphone

LED témoin fonctionnement

Affichage trop haut, trop bas etc …

Affichage OLED des notes

SAMD21ATJ18A

« M » de Mickaël BRIDAY qui nous a gentiment prêter son SAMD21

Port USB debug

Entrée jack

Jumper micro/jack

Jumper principal allumage

Figure 11 : PCB final

## 3.4 Tests

### 3.4.1 Tests de l’alimentation

Nous avons effectué plusieurs tests pour l’alimentation et celle-ci apparait très stable. Nous avons bien 3.3v pour alimenter le SAMD21, la masse virtuelle fonctionne parfaitement et notre érecteur (je rappelle réalisé par les tensions d’alimentations des AOP) fonctionne également, notre signal sonore sature à 400mv pour éviter les tensions max de l’ADC différentiel à .

### 3.4.2 Tests Ram

Nous n’avons malheureusement pas eu le temps de tester le module RAM, le [driver](https://github.com/albother/ER4_Accordeur/tree/master/projet/TP-info2/userMemory/src) est réalisé et incorporé au projet mais pas testé par manque de temps et donc pour l’instant simplement présent dans le code en tant que commentaire.

### 3.4.3 Tests Acquisition + Filtrage

Ici, on réalise plusieurs tests concernant le filtre Butterworth de fréquence de coupure 20kHz.

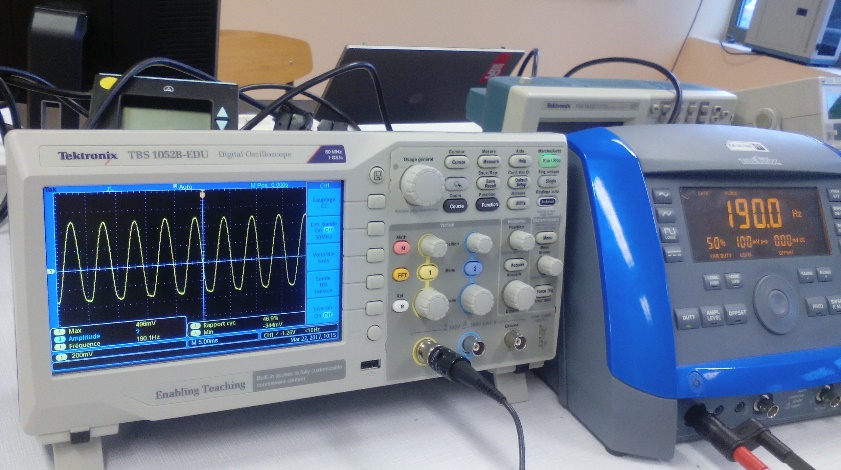
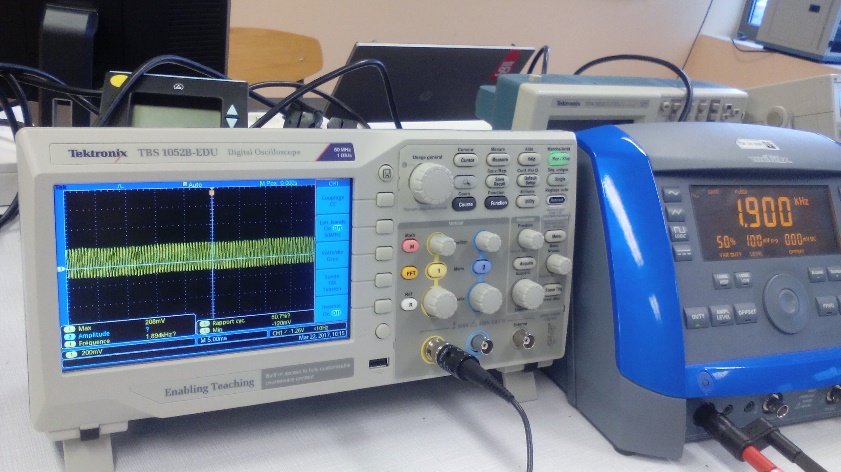


Figure 12 : Batterie de tests Butterworth

On réalise une batterie de 3 tests avec les fréquences, 190Hz, 1.9Khz et 19Khz.

On se rend compte qu’a 1.9KHz notre signal est fortement atténué et qu’a 20Khz il a complètement disparu, il semblerait donc que notre fréquence de coupure soit plus aux alentours de 10kHz que 20Khz. Cependant par manque de temps notre filtre restera en l’état. Nous avons dû faire une erreur de calcul au niveau des condensateurs ou au moment de placer les composants.

En revanche, il apparait que pour une application d’accordeur de guitare, et seulement de guitare, une fréquence de coupure à 10Khz n’est pas gênante, puisque notre guitare n’ira pas beaucoup plus haut que 1KHz environ.

Nous avons également effectué plusieurs tests avec le microphone et celui-ci fonctionne après l’ajout d’une résistance de pull down (présente sur le schéma précédent), en effet sans la résistance de 10k, l’AOP à suivre ne tirait pas assez de courant pour polariser le microphone.

## 3.5 Conclusion

En conclusion notre PCB fonctionne bien pour l’application actuelle, il faudrait revoir le filtre légèrement, et supprimer certains radiateurs, en effet, nous n’avons pas eu le temps de réaliser les calculs thermiques mais il semble que le régulateur ne chauffe pas du tout. Cela n’est pas très étonnant car celui-ci est branché sur le 4.5V il n’a donc que de chute de tension à ces bornes. Nous n’avons également pas eu le temps de tester la RAM, malgré que le driver soit réalisé. Nous avons cependant trouvé un « bidouillage » dans le soft pour se limiter à la RAM disponible en interne du SAMD21 (à voir par la suite) et fonctionner avec un tableau d’ADC de seulement 4096 au lieu de 44000.

# 4. Accordeur de signal réel

Nous allons maintenant utiliser notre PCB complet et tenter de coder une solution d’accordage complète.

Tout le code principal référencé par la suite est compris dans le même fichier et est présent dans [fft.c](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c).

## 4.1 Problématiques

C’est ici que nous avons eu le plus de problème, et où nous avons dû apporter le plus de solutions. Le traitement du signal à suivre étant complexe nous allons tout de même tenter de l’expliquer par étapes ;

* Acquisition ADC du signal différentiel
* FFT ou DFT ?
* Mise sous interruptions (TC4 & TC3)
* Traitement du signal temporel
* Traitement du signal fréquentiel
* Détection de la fréquence fondamentale
* Problème de RAM ?

## 4.2 Solutions + Code

### 4.2.1 Acquisition ADC différentielle

Pour simplifier l’acquisition du signal nous choisissons de créer un ADC différentiel, donc capable de récupérer des signaux positifs et négatifs.

EXPLICATION

On initialise donc l’ADC [ici](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L27) avec son timer associé dont nous détaillerons le calcul par la suite. Et pour nous en servir il nous suffira d’utiliser une syntaxe du type ;

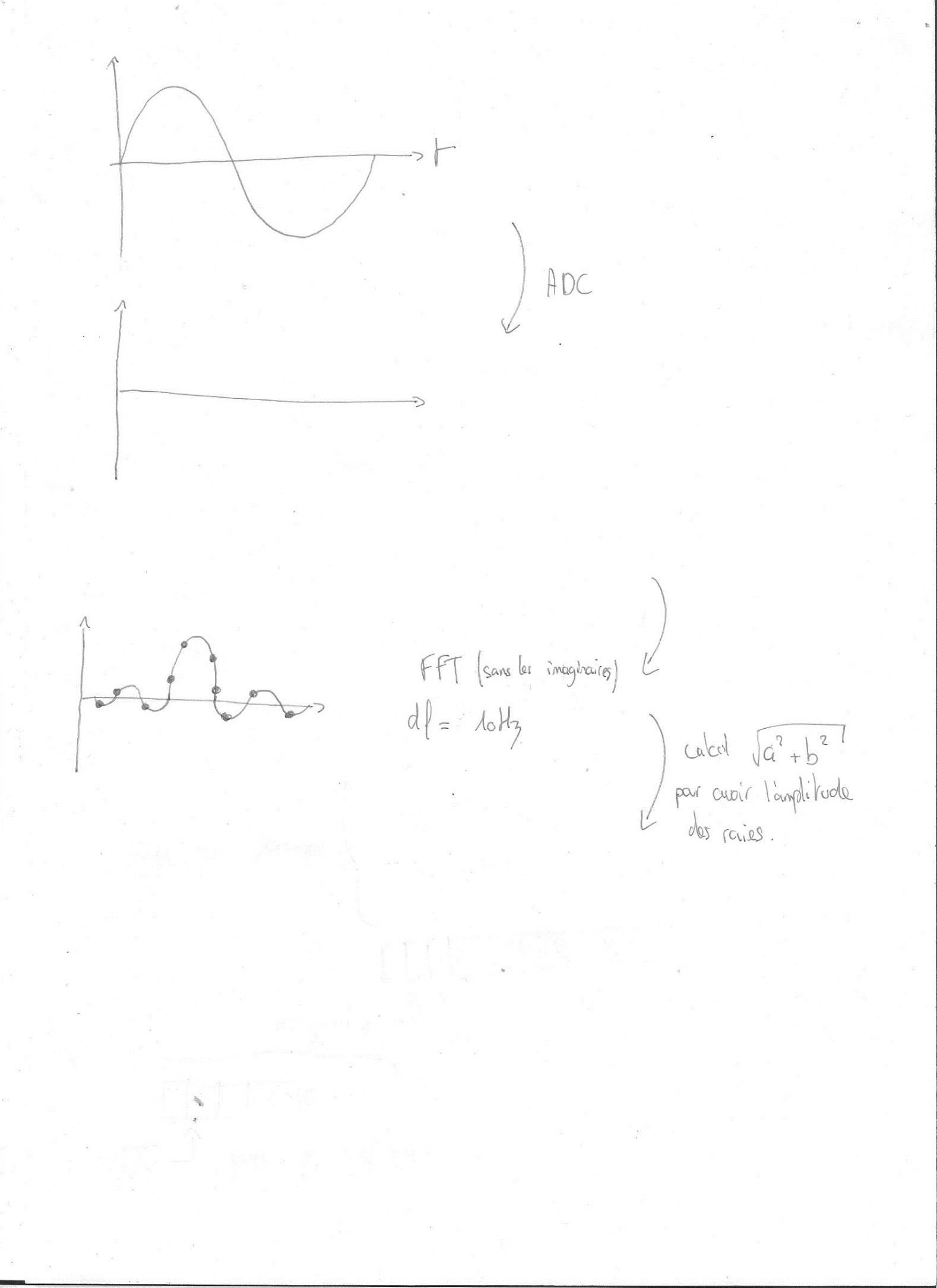
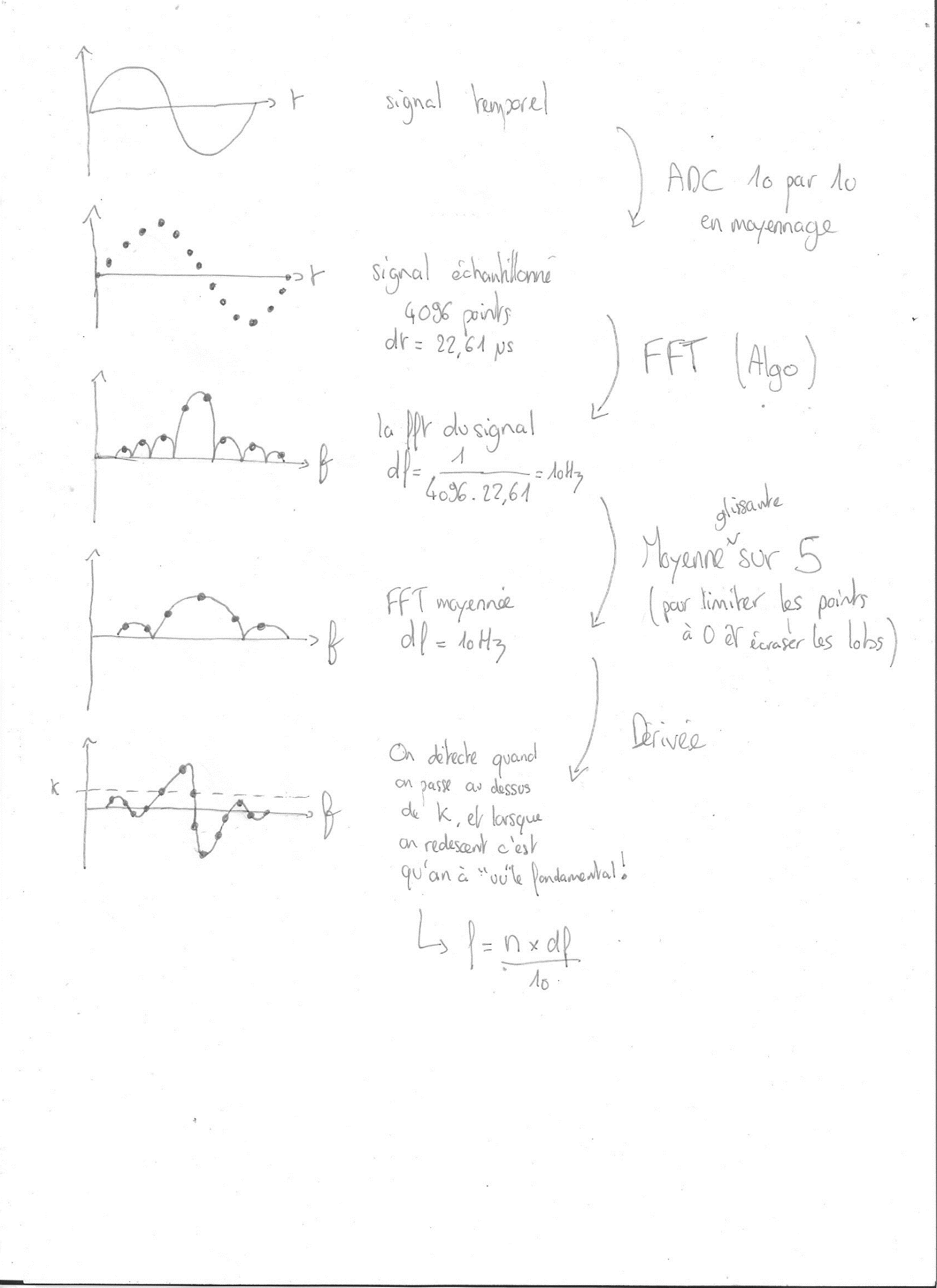
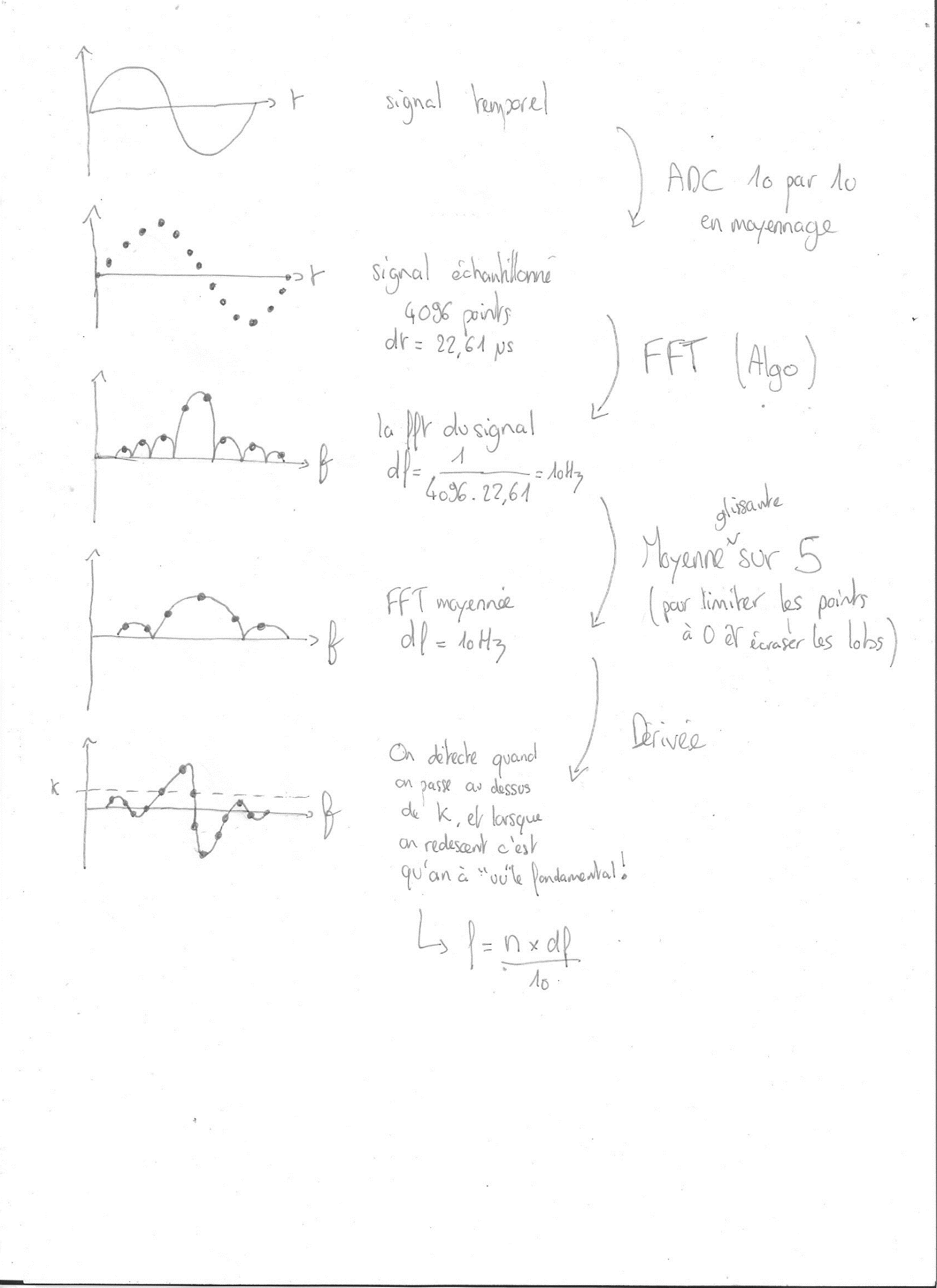
uint32\_t tab[3] = {0,SAMD21\_ADC\_MUX\_AIN2,SAMD21\_ADC\_MUX\_AIN3};

ADC\_TAB[i]=read\_adc(tab) ;

En incrémentant i autant de fois que la taille de ADC\_TAB.

### 4.2.2 La mise sur timers et idée générale

### 4.2.3 Le traitement ; FFT + Détecteur de fondamental :



Signal temporel réel

Ex : guitare , voix, GBF …

Signal échantillonné

4096 points

(image des 40960)

dt = 21,61us

Après FFT (partie réelle)

=

Vue classique de l’amplitude des raies de la FFT

df = 10Hz

FFT « écrasée »

df = 10Hz

FFT (Algorithme linéaire réalisant la transformée de Fourier)

Calcul pour avoir l’amplitude des raies

Moyenne glissante sur 5 échantillons (pour limiter les points à 0 et écraser les lobs )

Raies après dérivation, on détecte quand on passe au-dessus de k, et lorsque l’on redescend c’est qu’on a « vu » le fondamental.

ADC à 22.61 us (TC3). On récupère les valeurs de l'ADC, et toute les 10 valeurs, on en fait la moyenne et on met la valeur moyenne dans le vrai tableau de réels. Ainsi on a un tableau de 4096 valeurs qui contient en fait l’image de 40960 valeurs en « décimation ».

Dérivée et détection du fondamental

Au moment où on détecte le fondamental il suffit de repérer à quelle case (n) du tableau on est rendu et de de multiplier à l’échantillonnage df pour obtenir la fréquence du fondamental ! Seulement nous avons « bidouillé » au moment de l’acquisition ADC et chaque point à présent représentait 10 points avant, il faut donc diviser par 10 notre fréquence pour obtenir la fréquence réelle du fondamental. Soit :

Et on recommence toute les secondes tout le calcul (TC4)

## 4.3 Code

On commence par transposer tout le code du main dans une boucle timer [(TC4\_Handler)](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L234) appelée toute les 1 secondes.

Ainsi le main devient donc :

/\*

\* Accordeur de Guitare

\* Authors : METAYER Simon & BIZON Alexis

\* Created Date : 23/01/17

\* Version : 2.0

\*/

#include "define.h"

#include "samd21Adc.h"

#include "samd21Gpio.h"

int main**(**void**)**//fonction principale du programme

**{**

setup**();** // Initialisation générale

FFTInit**();** // Initialisation de la FFT

displayTitle**();** //affiche le titre du projet en haut de l'écran OLED

**while(**1**)**

**{**

// Vide

**}**

**return** 0**;** //Le programme renvoi 0x0000 si l'exécution c'est correctement terminée

**}**

Celui-ci se contente d’effectuer des initialisations puis de passer dans une boucle vide. Toutes les actions étant effectués sur timers.

Nous revenons donc dans notre fichier [fft.c](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c) ou est désormais présent tout le code principal. L’ancien code du main contenant les fonctions d’accordage sont maintenant dans [TC4\_Handler](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L234) une fonction appelée toute les secondes par l’intermédiaire de l’overflow du timer TC4 initialisé [ICI](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L49).

Nous ajoutons ensuite un second timer permettant d’effectuer l’acquisition périodique du signal temporel avec l’ADC différentiel. Ce sera donc TC3, que l’on initialise [ICI](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L34) pour appeler périodiquement la fonction [TC3\_Handler](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L200) toute les 22.61us. Cette fonction se chargera simplement d’effectuer l’acquisition et le moyennage par 10 pour « décimer » les valeurs et utiliser moins de RAM. Pour finir nous ajoutons la fonction de traitement du signal [FFT\_Detecteur\_Fondamental](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L70) avec le main dans la fonction périodique [TC4\_Handler](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L234). Ainsi le traitement du signal + l’accordage + l’affichage s’effectueront toute les secondes.

La fonction [FFT\_Detecteur\_Fondamental](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L70) effectue les actions, précédemment décrites dans le schéma de traitement des données, de manière séquentielle :

- [Algorithme FFT](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L75)

- [Calcul des raies en absolu](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L152)

- [Calcul moyennage glissant sur 5 échantillons](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L159)

- [Calcul dérivée et détection du fondamental](https://github.com/albother/ER4_Accordeur/blob/master/projet/TP-info2/userLibraries/src/fft.c#L173)

Puis la valeur en fréquence du fondamental est récupérée et envoyée dans le traitement de la note et l’accordage réalisé au début du projet.

## 4.4 Conclusion

# 5. Conclusion