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 Illustrations : 1](#_Toc476125969)

[0. Introduction 2](#_Toc476125970)

[0.1 Objectif 2](#_Toc476125971)

[0.2 Cahier des charges 2](#_Toc476125972)

[0.4 Travail à réaliser 2](#_Toc476125973)

[1. Accordeur de constante 3](#_Toc476125974)

[1.1 Problématique 3](#_Toc476125975)

[1.2 Solutions 3](#_Toc476125976)

[1.3 Code 5](#_Toc476125977)

[1.4 Tests 6](#_Toc476125978)

[1.5 Conclusion 6](#_Toc476125979)

[2. Accordeur de signal carré 7](#_Toc476125980)

[2.1 Problématiques 7](#_Toc476125981)

[2.2 Solutions 7](#_Toc476125982)

[2.3 Code 7](#_Toc476125983)

[2.4 Tests 8](#_Toc476125984)

[2.4 Conclusion 8](#_Toc476125985)

[3. Accordeur de signal réel en DFT modifiée ; détecteur de fondamental 9](#_Toc476125986)

[3.1 Problématique 9](#_Toc476125987)

[3.2 Solutions 9](#_Toc476125988)

[3.3 Code 9](#_Toc476125989)

[3.4 Conclusion 9](#_Toc476125990)

[4. Accordeur de signal réel en FFT (Avec supplément RAM) 10](#_Toc476125991)

[4.1 Problématique 10](#_Toc476125992)

[4.2 Solutions 10](#_Toc476125993)

[4.3 Code 10](#_Toc476125994)

[4.4 Conclusion 10](#_Toc476125995)

[5. Conclusion 11](#_Toc476125996)

# Table des Illustrations :

[Figure 1: Tableau des 8 premiers octaves 3](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc476126038)

[Figure 2: TESTS n°1 6](file:///C:\git\ER4_Accordeur\revueProjet\revue_de_projet.docx#_Toc476126039)

# 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)

## 0.4 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 :

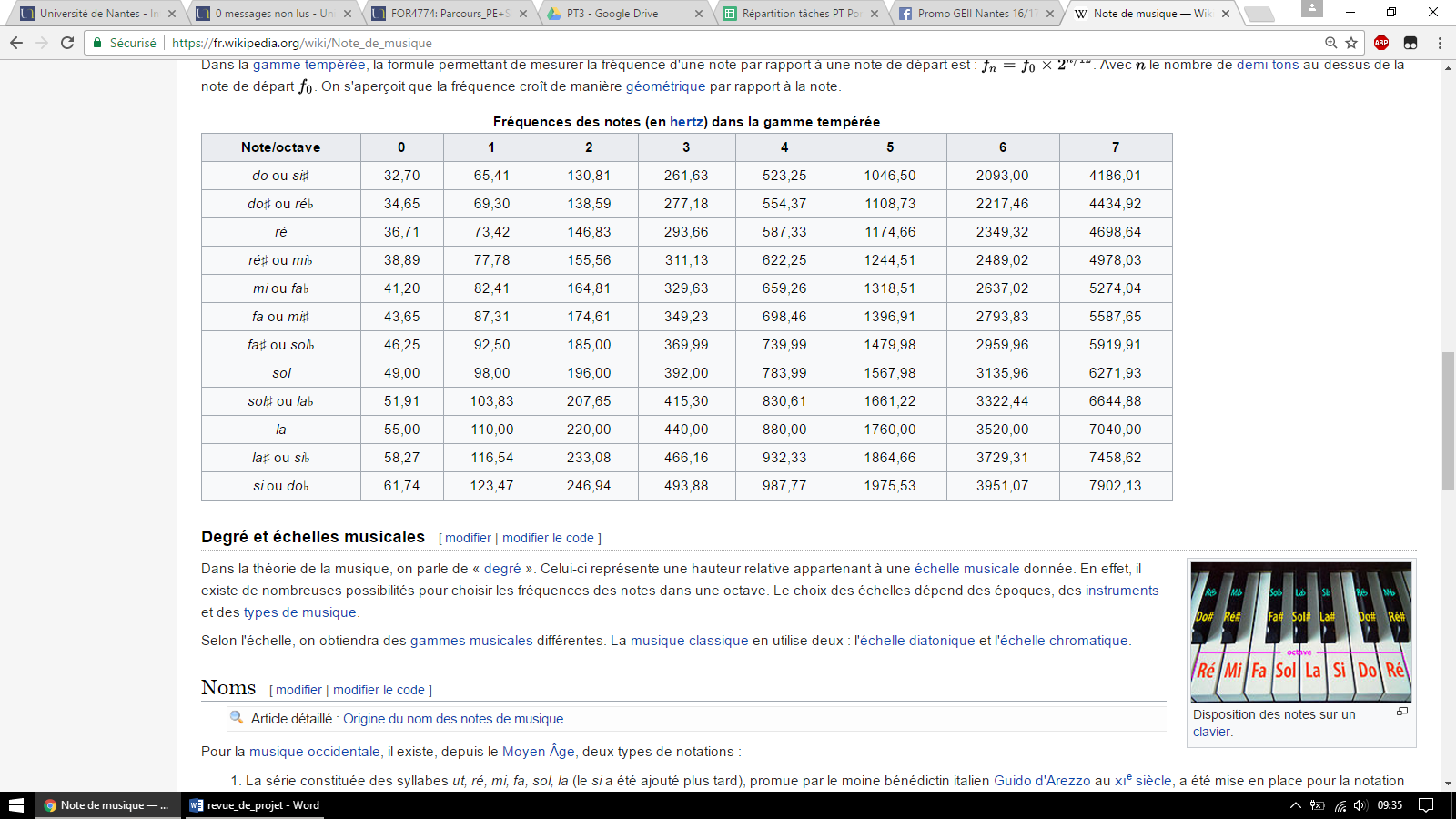


Figure 1: Tableau des 8 premiers octaves

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

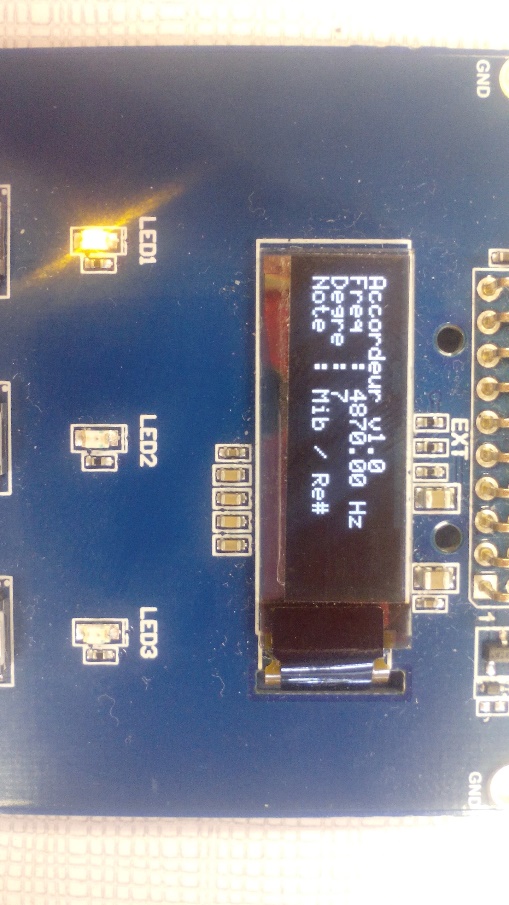
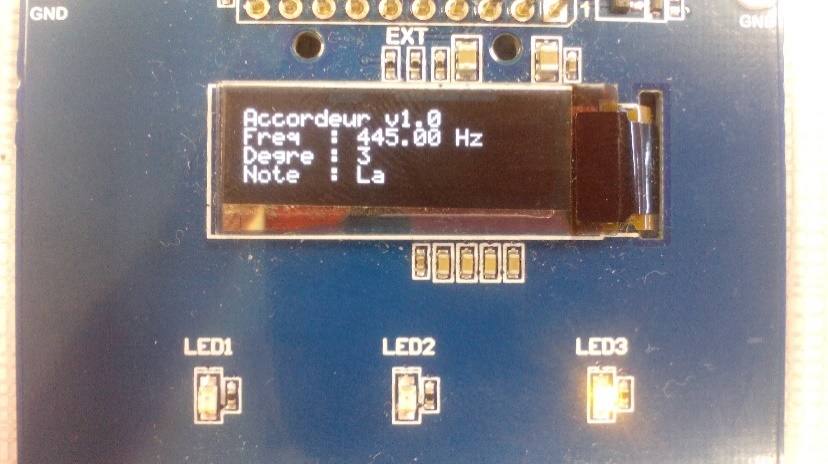
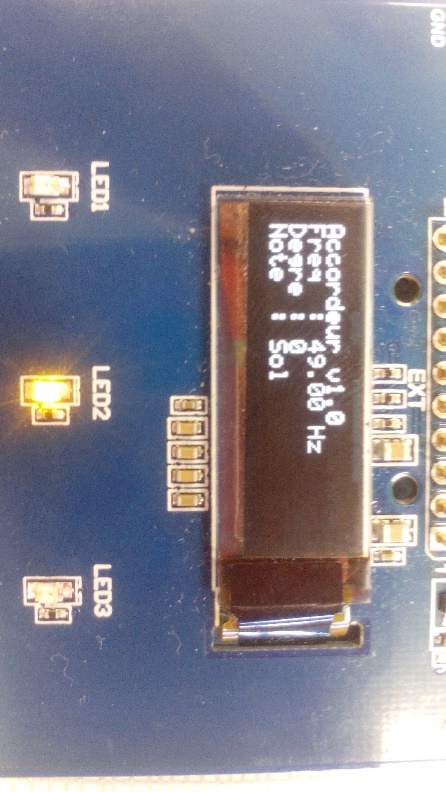


Figure 2: TESTS n°1

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 …

## 2.4 Tests

## 2.4 Conclusion

# 3. Accordeur de signal réel en DFT modifiée ; détecteur de fondamental

## 3.1 Problématique

## 3.2 Solutions

## 3.3 Code

## 3.4 Conclusion

# 4. Accordeur de signal réel en FFT (Avec supplément RAM)

## 4.1 Problématique

## 4.2 Solutions

## 4.3 Code

## 4.4 Conclusion

# 5. Conclusion