

CY IUT – GEII Neuville

Dossier de fabrication

Projet Guitare electro-acoustique

[Auteur/Redacteur]

Version : x.x – [Date de publica

Table des matières

1 Introduction – Présentation du système	
2 Schéma électronique et fonction technique	2
3 Notice de calcul des composants.....	3
3.1 FSx :	3
3.2 FSx :	3
4 Documents de contrôle de fabrication.....	3
4.1 Typon – Visuel Gerber.....	3
4.2 Plan de montage (visuel 3D).....	4
5 Nomenclature.....	4

[Date de publication] Version : x.x 1

1 Introduction – Présentation du système

Ce dossier de fabrication décrit les étapes nécessaires à la réalisation technique du projet de guitare électro-acoustique. Cette guitare est conçue pour fonctionner en mode manuel et automatique, offrant une expérience utilisateur innovante et optimisée grâce à l'intégration de plusieurs technologies.

Le système repose sur trois grandes parties : la **programmation**, la **mécanique** et l'**alimentation**.

- **Programmation** : Une carte Arduino Uno R4 Minima est utilisée pour contrôler les différentes fonctionnalités de la guitare. Le code permet de gérer l'accordage automatique, l'affichage des informations sur un écran OLED et le traitement des signaux audio pour le fréquencemètre. L'environnement de développement utilisé est PlatformIO sous Visual Studio Code, offrant des outils avancés pour le débogage et l'optimisation des performances.
- **Mécanique** : Le corps de la guitare est un élément clé du projet. Il inclut un guide de corde imprimé en 3D pour garantir un alignement précis, un système de maintien pour stabiliser la tension de la corde et un compartiment électronique intégré à la caisse de résonance. Le mécanisme de tension des cordes peut être ajusté manuellement et est conçu pour être automatisé à l'avenir via un moteur électrique.
- **Alimentation** : La guitare dispose d'une alimentation hybride capable de fonctionner sur secteur (24V AC) ou sur batterie. Un système de basculement automatique entre ces deux sources est assuré par des diodes Schottky. L'alimentation est conçue pour fournir différentes tensions adaptées aux composants : 12V pour les moteurs pas à pas, 6V pour la carte Arduino et 5V pour les actionneurs électromagnétiques.

Ce projet allie ainsi des concepts avancés d'électronique, de mécanique et d'informatique embarquée pour créer une guitare innovante et performant

1. Architecture du Système

Le système se compose de plusieurs sous-ensembles :

- **Mécanique :**
 - Guide de corde imprimé en 3D pour un alignement précis.
 - Système de maintien pour stabiliser la tension de la corde.
 - Compartiment électronique intégré à la caisse de résonance.
 - Système de tension manuel avec possibilité d'automatisation.
- **Électronique :**
 - Carte Arduino Uno R4 Minima pour le pilotage des composants.
 - Interface utilisateur avec écran OLED 128x128 pixels.
 - Fréquencemètre pour analyser les notes jouées.
 - Circuit d'alimentation hybride (secteur et batterie).
 - Commande des moteurs pas à pas pour l'accordage automatique.

2 Schéma électronique et fonction techniques

1 Conception et Fabrication des Pièces Mécaniques

1.1.1 Caisse de Résonance et Structure Générale

Le corps de la guitare, fourni par le professeur, sert de base au projet. Il est conçu pour amplifier le son produit par la vibration de la corde. Une zone spécifique sera aménagée pour loger l'électronique et l'alimentation du futur système de motorisation.

1.1.2 Système de Tension de la Corde

La corde est fixée à la caisse de résonance via une vis à la base. L'autre extrémité de la corde passe par un guide conçu pour diriger la tension vers une roue.

Le serrage manuel de la corde est assuré par un mécanisme utilisant une roue sur laquelle la corde s'enroule. En tournant cette roue, la tension de la corde est ajustée jusqu'à obtenir la note Mi. Ce principe sera ultérieurement motorisé afin d'automatiser l'accordage.

1.1.3 Pièces Conçues en CAO

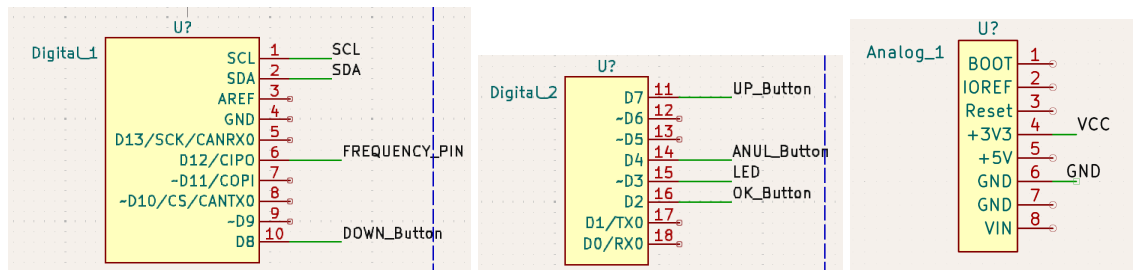
Plusieurs pièces ont été modélisées en CAO afin d'optimiser la tension et le maintien de la corde :

- **Le guide de corde**, qui assure l'alignement et la bonne tension.
- **Le support de tension**, conçu pour faciliter l'installation du futur moteur.
- **L'espace pour l'électronique et l'alimentation**, prévu dans la caisse.

Actuellement, ces pièces sont issues du recyclage d'un projet antérieur, mais elles seront remplacées par des impressions 3D adaptées aux spécifications du projet.

2.1 Fonctions Techniques

Cette partie se concentrera sur toutes les fonctions techniques du prototype A avec les câblages, les calculs nécessaires et la programmation de chaque fonctionnalité pour ce projet. Ci-dessous on peut voir toutes les broches utilisées dans ce projet:



Des schémas de câblages seront montrés dans cette section du dossier et tous les noms des broches seront reliés directement aux broches déclarées ci-dessus.

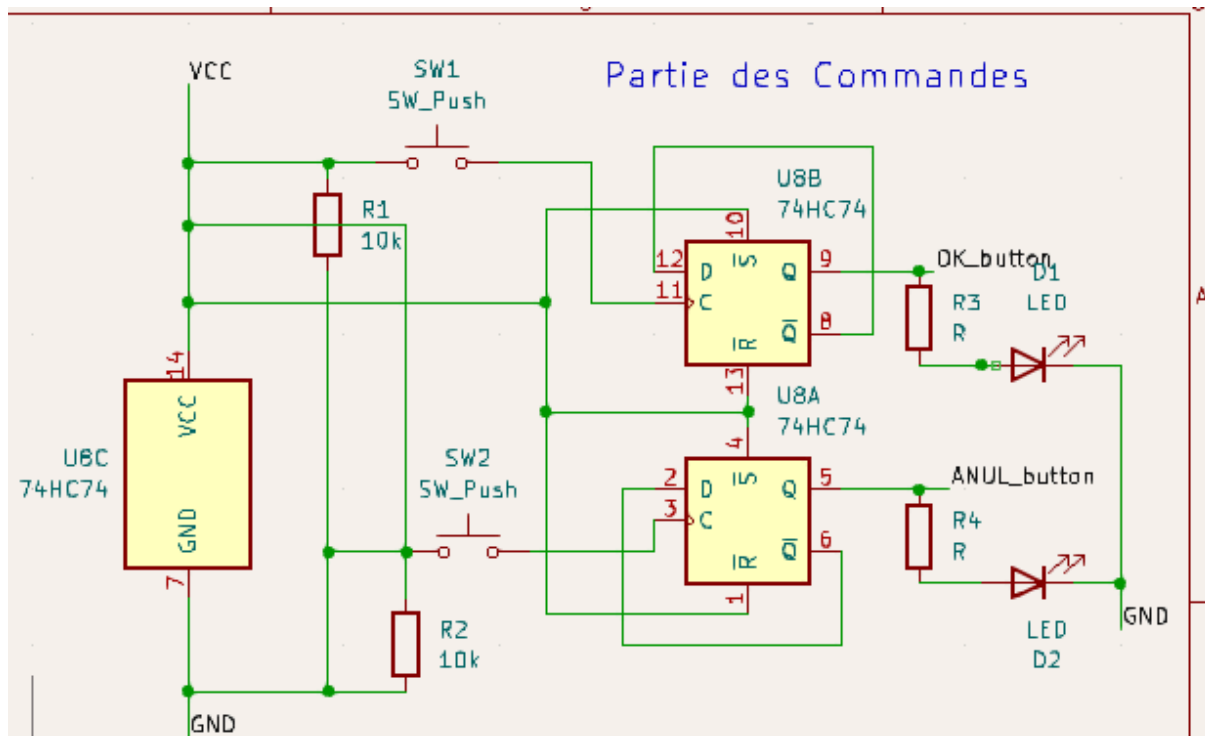
2.1.1 Partie Commandes

Cette partie consiste de quatres boutons poussoir, une bascule D SN74HC74 alimenté en 3.3 V, quatres DELs, deux résistances de 10K Ohm et quatres résistances au niveau des diodes qui sont de valeur $R = 670 \text{ Ohm}$.

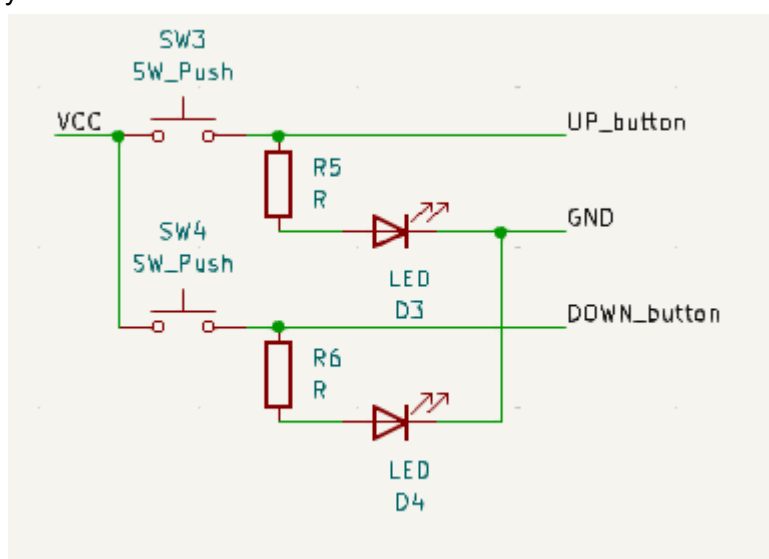
		SN54HC74			SN74HC74		
		MIN	NOM	MAX	MIN	NOM	MAX
V C C	Supply voltage	2	5	6	2	5	6
	VCC = 2 V	1.5			1.5		
VI	Input voltage	0		V C C	0		V C C
VO	Output voltage	0		V C C	0		V C C

Lorsque que l'utilisateur va appuyer sur le bouton SW1, une impulsion sera envoyée sur la bascule D qui va ensuite retenir l'état qu'il a reçu, c'est à dire lorsque la bascule est à zéro et reçoit une impulsion, elle prend la valeur opposé à celle en entrée soit "1" pour une impulsion et une sortie à "0" initialement, et le cas inverse avec "0" en sortir pour "1" initialement. Ce résultat va sur la broche D2 de l'Arduino.

Ces cas d'état haut et bas s'appliquent aussi pour le bouton SW2 représentant le bouton d'annulation ou de lancement de l'arrêt du programme. Le résultat pour le bouton SW2 ira sur la broche D4



Ces deux autres boutons servent à naviguer sur le menu et donc n'ont pas besoin de retenir leur état. Les résistances R ici ont les mêmes valeurs que celles précédentes et la même chose pour les DELs. Le résultat du bouton SW3 ira sur la broche D7 et pour le bouton SW4 il y aura la broche D8.



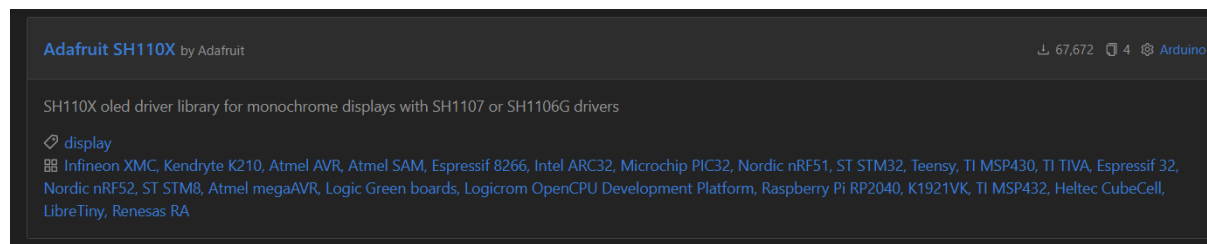
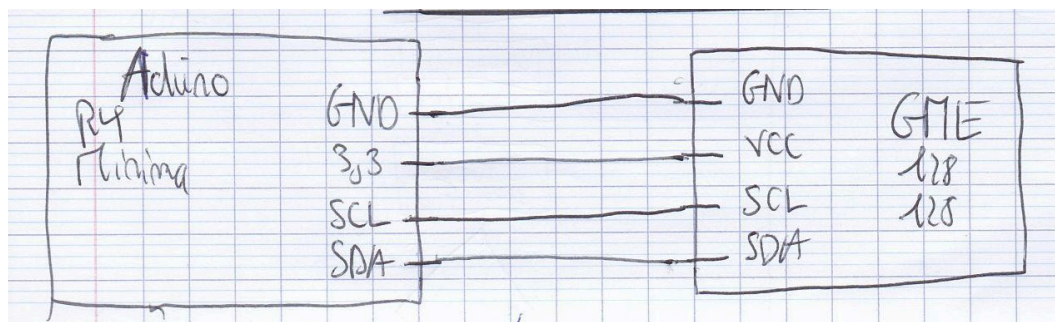
Datasheet du SN74HC74

<https://www.alldatasheet.com/html-pdf/354858/TI/SN74HC74/25/1/SN74HC74.html>

2.1.2 Affichage

Cette partie possède un schéma simple montrant le câblage de l'écran oled GME128128 fonctionnant en communication i2c et alimenté en 3.3 V. Les broches de SCL et de SDA sont reliées directement aux broches équivalentes de la carte Arduino. Ce composant possède aussi une librairie qui peut être incluse dans le projet PlatformIO nommé Arduino SH110X.

ITEM	MIN	MAX	UNIT	Condition	Remark
Supply Voltage (V _{Cl})	-0.3	4	V	Ta = 25°C	IC maximum rating
Supply Voltage (V _{CC})	8	19	V	Ta = 25°C	IC maximum rating



Datasheet du GME128128:

<https://www.scribd.com/document/492713958/GME128128-01>

2.1.3 Programmation

2.1.3.1 Bibliothèques, Définitions et fichier d'initialisation

Le projet étant codé en Arduino, il nous faut inclure les bibliothèques nécessaires pour pouvoir réaliser les fonctions pour chaque fonctionnalité. Les variables et définitions pour l'écran oled GME128128 peuvent être récupérées sur le site github et aussi la méthode d'initialisation de l'écran. Ainsi, les bibliothèques utilisées sont Arduino, Wire et SPI pour les méthodes de communication, Adafruit_GFX et SH110X pour l'affichage sur l'oled. Toutes les broches sont mobilisées pour les fonctionnalités, les informations pour l'écran et aussi les fonctions créées pour soi la gestion du programme entier, soit pour chacune des fonctionnalités.

```
#include <Arduino.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SH110X.h>

#define OK_BUTTON D2
#define ANL_BUTTON D4
#define UP_BUTTON D7
#define DOWN_BUTTON D8
#define FREQUENCE_PIN D12
#define LED D3

#define SCREEN_WIDTH 128 // display display width, in pixels
#define SCREEN_HEIGHT 128 // display display height, in pixels
#define SCREEN_RESET -1 // can set an display reset pin if desired
```



```

#define SCREEN_ADRESS 0x3C
#define BATTERY_PIN A0
Adafruit_SH1107 display = Adafruit_SH1107(SCREEN_WIDTH, SCREEN_HEIGHT,
&Wire, SCREEN_RESET, 1000000, 100000);
void GME128128_begin(uint8_t adress);
void GME128128_menu();
void Frequence();
void Auto();
void Exit();
void Battery();
void Manuel();
void RappelNotes();
void pwm_gestion();

```

2.1.3.2 Setup() et Loop()

Ici sont les deux fonctions essentielles en Arduino. La fonction setup() est la toute première fonction lancée une seule fois seulement durant tout le programme. c'est donc dans cette fonction qu'il faut déclarer toutes les variables globales, c'est-à-dire associer une valeur ou comme dans notre cas, déclarer les entrées et sorties. Seule la broche de la batterie n'a pas été définie puisque nous n'avons pas encore les détails de sortie de la batterie, notamment pour mesurer le niveau de batterie par rapport à la tension et ses caractéristiques. Enfin, on envoie l'adresse de l'écran oled et la fonction de démarrage GME128128_begin().

La fonction loop() ici sera utilisée juste pour faire une boucle du menu.

```

void setup()
{
    Serial.begin(115200);

    pinMode(DOWN_BUTTON, INPUT);
    pinMode(UP_BUTTON, INPUT);
    pinMode(ANL_BUTTON, INPUT);
    pinMode(OK_BUTTON, INPUT);
    pinMode(LED, OUTPUT);
    pinMode(FREQUENCE_PIN, INPUT);
    GME128128_begin(SCREEN_ADRESS);
}

void loop()
{
    GME128128_menu();
}

```

2.1.3.3 Signaux PWM

Les signaux PWM sont utiles pour commander des moteurs et régler leur vitesse en fonction de leur période. Nous, dans notre situation, les moteurs n'étant pas encore fait, le signal est envoyé sur une DEL. Plus précisément, ce qu'on fait est que le signal aura une période de plus en plus grande jusqu'à devenir un état logique haut (la variable *i* allant de 1 jusqu'à 256), et lorsqu'on sort de la première boucle *for*, on suit la même méthode mais de manière inverse (*i* allant de 256 à 1).

```
void pwm_gestion()
{
    for(int i = 0; i < 256; i++)
    {
        analogWrite(LED, i);
        delay(10);
    }
    for(int i = 256; i > 0; i--)
    {
        analogWrite(LED, i);
        delay(10);
    }
}
```

2.1.3.4 Initialisation GME128128

Cette fonction permet d'initialiser l'écran oled avec son adresse envoyée en argument. Le premier bloc est une suite de lignes venant directement du github de la librairie, initialisant l'écran oled. La deuxième partie est plutôt une introduction du projet avec les responsables des axes principaux du projet.

```
void GME128128_begin(uint8_t adress)
{
    display.begin(adress, true);
    display.display();
    delay(2000);
    display.clearDisplay();
    display.drawPixel(10, 10, SH110X_WHITE);
    display.display();
    delay(2000);
    display.clearDisplay();

    display.setTextCursor(SH110X_WHITE);
    display.setTextSize(1);
    display.setCursor(1, 1); display.print("BUT GE2I - PROJET SAE\n");
    display.setCursor(1, 8); display.print("3E ANNEE\n\n");
}
```

```

        display.print("Lathro-Seri Nathan pour la programmation\n\nIdmond
Benjy          pour          la          conception\n\nRamirez          Victor
pour\n\n'alimentation\n\nGuitare Electro-\nAcoustique");
    display.display();
    delay(3000);
    display.clearDisplay();
}

```

2.1.3.5 Menu

La fonction principale du programme permet de lancer les fonctionnalités choisies par l'utilisateur à travers deux boutons de navigation, un bouton de validation et un bouton d'annulation. La première possibilité est que les deux boutons OK_BUTTON et ANL_BUTTON soit à l'état haut: ainsi le programme affiche un message d'erreur puisqu'on ne peut pas sélectionner et désélectionner en même temps. La deuxième possibilité est si on choisit le bouton sur une des fonctionnalités présentées: ainsi, la fonction appartenant à la fonctionnalité est lancée et l'action attendue est exécutée. La troisième possibilité est si le bouton d'annulation est activé, ainsi la procédure d'arrêt du programme est lancée.

Toutes ces actions sont prises en compte avec un switch case dépendant d'un entier nommé menu_direction qui s'incrémente jusqu'à 6 avec les états haut de la broche UP_BUTTON et se décrémente jusqu'à 0 avec les états haut de la broche DOWN_BUTTON.

```

void GME128128_menu()
{
    static int menu_direction = 0;

    if(digitalRead(OK_BUTTON) && digitalRead(ANL_BUTTON))
    {
        display.clearDisplay();
        display.setTextColor(SH110X_WHITE);
        display.setTextSize(1);
        display.setCursor(1, 1);
        display.print("ERROR: both buttons have high state");
        display.display();
    }
    else
    {
        display.clearDisplay();
        display.setTextColor(SH110X_WHITE);
        display.setTextSize(1);
        display.setCursor(55, 1);
        display.print("-->MENU<--\n\n");
        display.setCursor(1, 2);

        if(!digitalRead(DOWN_BUTTON) && digitalRead(UP_BUTTON))
            menu_direction++;
        else
        {

```

```

        if(digitalRead(DOWN_BUTTON) && !digitalRead(UP_BUTTON))
            menu_direction--;
    }
    if(menu_direction >= 6)
        menu_direction = 0;
    else
    {
        if(menu_direction <= -1)
            menu_direction = 5;
    }

    switch (menu_direction)
    {
        case 0:
            display.print("\n\n-> Frequencemetre\n\nMode
Manuel\n\nMode Automatique\n\nBatterie\n\nRappels Notes\n\nExit");
            if(digitalRead(OK_BUTTON) &&
!digitalRead(ANL_BUTTON))
                Frequence();
            break;
        case 1:
            display.print("\n\nFrequencemetre\n\n-> Mode
Manuel\n\nMode Automatique\n\nBatterie\n\nRappels Notes\n\nExit");
            if(digitalRead(OK_BUTTON) &&
!digitalRead(ANL_BUTTON))
                Manuel();
            break;
        case 2:
            display.print("\n\nFrequencemetre\n\nMode Manuel\n\n->
Mode Automatique\n\nBatterie\n\nRappels Notes\n\nExit");
            if(digitalRead(OK_BUTTON) &&
!digitalRead(ANL_BUTTON))
                Auto();
            break;
        case 3:
            display.print("\n\nFrequencemetre\n\nMode Manuel\n\nMode
Automatique\n\n-> Batterie\n\nRappels Notes\n\nExit");
            if(digitalRead(OK_BUTTON) &&
!digitalRead(ANL_BUTTON))
                Battery();
            break;
        case 4:
            display.print("\n\nFrequencemetre\n\nMode Manuel\n\nMode
Automatique\n\nBatterie\n\n-> Rappels Notes\n\nExit");

```

```

if(digitalRead(OK_BUTTON) == HIGH)
{
    if(digitalRead(ANL_BUTTON))
    {
        RappelNotes();
        break;
    }
    default:
    {
        display.print("\n\nFrequencemetre\n\nMode Manuel\n\nMode
Automatique\n\nBatterie\n\nRappels Notes\n\n-> Exit");
        if(!digitalRead(OK_BUTTON) == HIGH)
        {
            if(digitalRead(ANL_BUTTON))
            {
                Exit();
                break;
            }
        }
        display.display();
    }
}
}

```

2.1.3.6 Fréquencemètre

La fonction du fréquencemètre permet de récupérer la fréquence d'un signal carré. Après que notre signal d'origine audio (les vibrations des cordes) soit converti en signal carré, cette fonction s'occupe de calculer la fréquence moyenne à travers l'échantillonnage du signal avec la fonction pulseIn() en Arduino, permettant de récupérer la période d'un état binaire, c'est à dire la durée de l'impulsion (état bas ou état haut). Puisque notre ouïe est capable de capter des sons de 20 à 20k Hz, le minimum dans notre cas est de faire 5 échantillons pour les état haut, 5 pour les état bas, puis faire une moyenne pour obtenir la période moyenne du signal et enfin inverser la valeur de cette période pour pouvoir récupérer la fréquence moyenne. En plus d'afficher la fréquence, si elle est égale aux fréquences des cordes, le programme affichera le nom de la note de la corde directement, sur un intervalle de +/- 5% autour de la valeur précise de la note.

```

void Frequence()
{
    const unsigned char music_note [] PROGMEM = {
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x18, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0xf8, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x0f, 0xf8, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x7f, 0xf8, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x07, 0xff, 0xf8, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x3f, 0xff, 0xf8, 0x00,
    };
}

```

[illegible]

```

        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,
0x00, 0x00, 0x00, 0x00, 0x00, 0x00
    };

    const int      frequency_notes[] = {82, 110.00, 147, 196.00, 247,
330};

    const char      frequency_names[] = {'E2', 'A2', 'D3', 'G3', 'B3',
'E4'};

    float           freq              = 0;
    unsigned long    period            = 0;
    unsigned long    sumhigh           = 0;
    unsigned long    sumlow            = 0;
    int              f_ch_indice       = 0;
    unsigned long    a                 = 0;

    for(int i = 0; i < 5; i++)
    {
        a = pulseIn(FREQUENCY_PIN, HIGH);
        sumhigh+=a;
    }
    for(int i = 0; i < 5; i++)
    {
        a = pulseIn(FREQUENCY_PIN, LOW);
        sumlow+=a;
    }

    period = (sumhigh + sumlow)/5;
    freq = 1000000.0/period;
    Serial.print("Frequency: "); Serial.println(freq);

    display.clearDisplay();
    display.setTextColor(SH110X_WHITE);
    display.drawBitmap(1, 1, music_note, 64, 64, SH110X_WHITE);
    display.setTextSize(2);
    display.setCursor(1, 70);
    display.println("FREQUENCY");
    display.setTextSize(1);
    display.setCursor(1, 90);
    display.print("Frequency = "); display.print(freq); display.print("
Hz");

    if(freq == 4294967295)

```

```

    freq = 0;

    int horschamp = 0;

    for(int i = 0; i < sizeof(frequency_notes); i++)
    {
        if((freq/frequency_notes[i]) > 0.05)
        {
            f_ch_indice = i;
            display.setCursor(1, 100);
            display.setTextSize(1); display.print("Note = ");
display.print(frequency_notes[i]);
        }
        else
            horschamp++;
    }
    if(horschamp == sizeof(frequency_notes))
    {
        display.setCursor(1, 100);
        display.setTextSize(1); display.print("Note = Hors Champ\n");
    }

    display.display();
}

```

2.1.3.7 Modes Manuel/Automatique

Le mode auto est un mode qui va lancer la fonction de gestion des signaux PWM décrite précédemment. Dans le cas où l'utilisateur veut arrêter cette fonctionnalité, il peut mettre à l'état bas la broche OK_BUTTON en appuyant à nouveau sur le bouton dédié.

Le mode manuel est prévu de faire des économies d'énergie mais pour le prototype A, c'est une simple fonction d'affichage qui sert d'exemple.

```

void Auto()
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE);
    display.setTextSize(1);
    display.setCursor(1, 1);
    display.println("Signal PWM...");
    display.display();
    pwm_gestion();

    display.clearDisplay();
    display.setTextColor(SH110X_WHITE);
    display.setTextSize(1);

```



```

        display.setCursor(1, 1);
        display.print("Desactivate the\ngreen button");
        display.setCursor(1, 2);
    }

void Manuel()
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE);
    display.setTextSize(1);
    display.setCursor(1, 1);
    display.println("Show them your talent !");
    display.display();

    //fonction de mise en veille du systeme
}

```

2.1.3.8 Batterie

N'ayant pas tous les détails sur les batteries à choisir pour notre projet, cela ne nous empêche pas de nous questionner. Dans une logique où la carte arduino doit recevoir 6 V sur la broche d'alimentation VIN, une des sorties de la batterie sera à 6 V et les bits de résolution de la broche sont de 10 bits.

```

void Battery()
{
    int battery_v = analogRead(BATTERY_PIN);
    float battery_lvl = (6/1023)*battery_v;

    display.clearDisplay();
    display.setTextColor(SH110X_WHITE);
    display.setTextSize(1);
    display.setCursor(1, 100);
        display.print("Niveau de la Batterie = ");
display.print(battery_prc); display.print(" %");
    display.setCursor(1, 1);
    display.println("Battery pourcentage");
    display.display();
}

```

2.1.3.9 Fonction d'Arrêt

Dans le cas où l'utilisateur veut arrêter le programme, il lui faudra que seul le bouton d'annulation doit être activé. Ensuite, une vérification est faite pour s'assurer que l'utilisateur veut bien arrêter le programme. Ainsi, l'écran oled affichera une suite de messages pour signifier l'arrêt. Un dernier message invite l'utilisateur a reset la carte pour relancer le programme.

```

void Exit()
{
    while(digitalRead(ANL_BUTTON) == 0)
    {
        display.clearDisplay();
        display.setTextColor(SH110X_WHITE);
        display.setTextSize(1);
        display.setCursor(1, 1);
        display.print("Do you really want to exit the program
?\\nGreen=Yes\\nRed=No");
        display.display();

        if(digitalRead(OK_BUTTON))
        {
            display.clearDisplay();
            display.setTextColor(SH110X_WHITE);
            display.setTextSize(1);
            display.setCursor(1, 1);
            display.print("Stoping...");
            display.display();
            delay(3000);
            display.clearDisplay();
            display.setTextColor(SH110X_WHITE);
            display.setTextSize(1);
            display.setCursor(1, 1);
            display.print("Reset the Arduino for launching back the
code");
            display.display();
            delay(500);

            //fonction de mise en veille du systeme

            exit(0);
        }
    }
}

```

2.1.3.10 Rappel Notes

Cette fonction n'est qu'à titre informatif si l'utilisateur utilise le fréquencesmètre mais qu'il ne connaît pas les fréquences des cordes.

```

void RappelNotes()
{
    display.clearDisplay();
    display.setTextColor(SH110X_WHITE);
    display.setTextSize(1);

```

```

display.setCursor(1, 1);
display.println("Rappel:\n");
display.print("Corde-Nom-Note-Frequence\n");
display.print("1-Mi aigu-E4-329.63Hz\n");
display.print("2-Si-B3-246.94 Hz\n");
display.print("3- Sol-G3-196.00 Hz\n");
display.print("4-Ré-D3-146.83 Hz\n");
display.print("5-La-A2-110.00 Hz\n");
display.print("6-Mi grave-E2-82.41\n");
}

```

2.1.3.11 Ressources utilisées

Ici sera tous les liens vers les ressources utilisées durant la programmation de ce projet:

Fréquence des cordes	https://www.jegratte.com/accorder-sa-guitare
Fonction d'initialisation de l'écran	https://github.com/adafruit/Adafruit_SH110x/blob/master/examples/SH1107_128x128/SH1107_128x128.ino
Calcul de la fréquence	https://www.arduino.cc/reference/fr/language/functions/advanced-io/pulsein/

2.1.4 Alimentation

2.1.4.1 Objectifs et contraintes

L'alimentation du système doit assurer une distribution stable et efficace pour répondre aux besoins des différents composants. Elle doit fonctionner en mode secteur et en mode batterie, avec un basculement automatique entre les deux sources.

Le circuit devra fournir trois tensions principales : 12V pour les moteurs pas à pas, 6V pour l'Arduino UNO R4 Minima et 5V pour les électroaimants. La consommation totale du système est estimée à environ 80W en charge maximale, ce qui nécessite une alimentation robuste et optimisée.

Pour garantir un fonctionnement sécurisé et durable, l'alimentation doit intégrer des protections contre les surtensions, les surintensités et les courts-circuits. Un interrupteur principal permettra de couper totalement le circuit lorsque la guitare n'est pas utilisée.

2.1.4.2 Architecture de l'alimentation

L'alimentation repose sur un transformateur 24V AC, qui est redressé et filtré pour obtenir une tension continue exploitable. Cette tension est ensuite abaissée par des convertisseurs Buck afin de générer les différentes tensions nécessaires.

Le passage entre le mode secteur et le mode batterie est assuré par des diodes Schottky, qui empêchent tout retour de courant indésirable et garantissent un basculement fluide.

Un interrupteur DPST (Double Pôle) est placé avant la distribution des tensions pour permettre une coupure totale du circuit.

2.1.4.3 Composants utilisés

L'alimentation est constituée des éléments suivants :

- Transformateur 24V AC : Permet d'alimenter le système à partir du secteur.
- Pont de diodes : Assure le redressement du courant alternatif en courant continu.
- Condensateurs de filtrage : Stabilisent la tension après redressement.
- Convertisseurs Buck : Régulent la tension pour obtenir 12V, 6V et 5V avec un bon rendement.
- Diodes Schottky : Gèrent le basculement entre secteur et batterie.
- Fusibles : Protègent le circuit contre les surintensités.
- Interrupteur DPST : Permet de couper complètement l'alimentation.

2.1.4.4 Procédure de fabrication

L'assemblage de l'alimentation se fait en plusieurs étapes. Tout d'abord, le circuit de redressement est monté avec un pont de diodes et un condensateur de filtrage pour obtenir une tension continue stabilisée autour de 24V.

Ensuite, les trois convertisseurs Buck sont installés et réglés pour fournir les tensions souhaitées. La fréquence de découpage des convertisseurs est ajustée afin d'optimiser leur rendement.

Les diodes Schottky sont positionnées de manière à assurer un basculement automatique entre secteur et batterie sans nécessiter d'éléments de contrôle actifs.

Enfin, les fusibles et l'interrupteur DPST sont ajoutés pour sécuriser le système et permettre une coupure totale de l'alimentation lorsque nécessaire.

2.1.4.5 Calcul des composants

Choix du condensateur de filtrage :

Le filtrage du redressement est assuré par un condensateur électrolytique dont la valeur doit être dimensionnée en fonction du courant consommé.

La valeur du condensateur peut être estimée par la formule :

où :

$$C = I / f * \Delta V$$

- I est le courant consommé ($\approx 3.5A$ max)
- f est la fréquence du réseau (50 Hz)
- ΔV est l'ondulation de tension acceptable ($\approx 2V$)

En appliquant ces valeurs :

Un condensateur de $4700\mu F$ à $10\ 000\mu F/35V$ sera utilisé pour limiter l'encombrement tout en assurant une bonne stabilisation.

Dimensionnement des diodes Schottky

Les diodes Schottky utilisées pour le basculement doivent supporter un courant de $3.5A$ avec une tension directe la plus faible possible pour minimiser les pertes.

Justification des convertisseurs Buck

L'utilisation de régulateurs linéaires (ex: 7805, 7812) n'est pas envisageable à cause des pertes thermiques importantes. Un régulateur linéaire $12V$ à partir de $24V$ dissiperait :

$$P = (24V - 12V) * 3.5A = 42W \text{ ce qui est énorme.}$$

Les convertisseurs Buck, en revanche, offrent un rendement de 85 à 90%, réduisant ainsi la dissipation thermique et augmentant l'efficacité globale.

2.1.4.6 Tests et validation

Après l'assemblage du circuit d'alimentation, plusieurs tests sont réalisés afin de vérifier son bon fonctionnement, sa stabilité et sa sécurité. Ces tests incluent la mesure des tensions de sortie, la simulation du basculement entre secteur et batterie, l'évaluation du rendement énergétique, et les tests de protection.

2.1.4.6.1 Vérification des tensions de sortie

Objectif : S'assurer que chaque convertisseur Buck délivre bien la tension prévue sous différentes conditions de charge.

Procédure :

1. À vide (sans charge)
 - Connecter un multimètre aux sorties $12V$, $6V$ et $5V$.
 - Vérifier que les valeurs mesurées sont dans les tolérances attendues ($12V \pm 0.5V$, $6V \pm 0.3V$, $5V \pm 0.2V$).
2. Sous charge nominale
 - Brancher une résistance de charge correspondant à la consommation moyenne de chaque tension.
 - Mesurer la tension aux bornes de la charge et vérifier qu'elle reste stable.
3. Sous charge maximale
 - Simuler une consommation équivalente aux moteurs, électroaimants et Arduino fonctionnant simultanément.

- Vérifier que la tension reste stable et qu'il n'y a pas de chute excessive.

Critères de validation :

- Tension stable dans les tolérances sous toutes les charges.
- Pas d'échauffement excessif des composants.

2.1.4.6.2 Test du basculement secteur/batterie

Objectif : Vérifier que la transition entre alimentation secteur et batterie se fait sans coupure et que la batterie ne se décharge pas lorsqu'elle n'est pas utilisée.

Procédure :

1. Alimentation en mode secteur
 - Alimenter le système en 230V AC et mesurer la tension de sortie après le transformateur, le redressement et les convertisseurs Buck.
 - Débrancher la batterie et vérifier qu'aucun courant ne circule vers elle.
2. Passage en mode batterie
 - Débrancher l'alimentation secteur brutalement et observer la réponse du circuit.
 - Vérifier qu'il n'y a aucune coupure d'alimentation aux bornes des convertisseurs Buck.
 - Mesurer la tension de la batterie et s'assurer qu'elle ne chute pas brutalement.
3. Retour au mode secteur
 - Rebrancher l'alimentation secteur et vérifier que le système repasse automatiquement sur cette source.
 - Vérifier que la batterie commence à se recharger si nécessaire.

Critères de validation :

- Aucune interruption des tensions en sortie lors du basculement.
- Pas de surtension ou d'oscillation anormale mesurée à l'oscilloscope.
- La diode Schottky empêche bien tout retour de courant vers le secteur.

2.1.4.6.3 Tests de protection et de sécurité

Objectif : Vérifier que les protections contre les surtensions et les surintensités fonctionnent correctement.

Procédure :

1. Test du fusible
 - Injecter une charge dépassant le courant maximal autorisé et vérifier que le fusible saute pour protéger le circuit.
2. Simulation de court-circuit
 - Réaliser un court-circuit sur la sortie 12V (via une résistance faible) et observer si l'alimentation se met en sécurité sans endommager les composants.
3. Test de surtension secteur
 - Augmenter la tension secteur progressivement (jusqu'à 250V AC) et vérifier que l'alimentation reste stable sans générer de surtension en sortie.
4. Test de décharge batterie
 - Décharger la batterie jusqu'à 10V et vérifier que l'alimentation se coupe automatiquement pour éviter une décharge profonde.

Critères de validation :

- Le fusible protège bien le circuit contre les surintensités.
- Les diodes TVS absorbent bien les surtensions sans endommager le circuit.
- La protection contre la décharge profonde fonctionne correctement.

2.2 Intégration Électronique

Le projet inclura à terme une motorisation du système de tension afin d'automatiser l'accordage. Pour cela, un espace dédié à l'électronique et à l'alimentation est intégré dans la conception. Ce compartiment permettra d'accueillir :

- **Un moteur électrique**, qui remplacera l'action manuelle de la roue.
- **Un microcontrôleur**, pour piloter l'accordage.
- **Un capteur de fréquence**, afin d'ajuster la tension automatiquement.
- **Une alimentation adaptée**, intégrée dans la caisse.

2.3 Nomenclature des Pièces

Référence	Description	Matériau	Mode de Fabrication
Caisse de résonance	Structure principale	Bois	Fournie par le professeur
Guide de corde	Maintien et alignement de la corde	Plastique	Impression 3D
Support de tension	Maintien de la roue	Plastique	Impression 3D
Roue d'accordage	Tension de la corde	Métal/Plastique	Impression 3D ou récupération
Compartiment		Bois/Plastique	Découpe et

électronique	Emplacement du système électronique		impression 3D
--------------	-------------------------------------	--	---------------

[Date de publication] Version : x.x 2



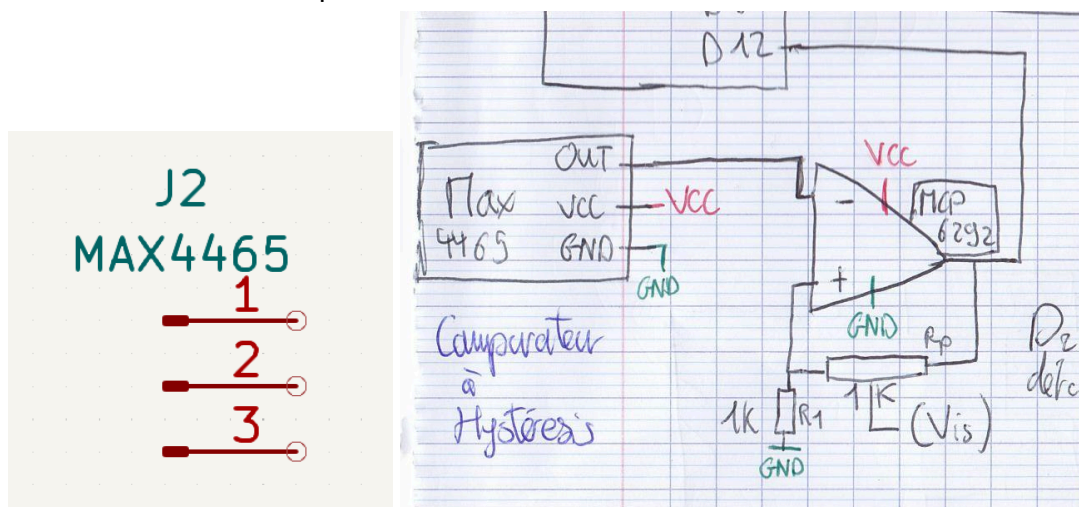
BUT 1 – SAE S1

Dossier de fabrication – [Système]

3 Notice de calcul des composants

3.1 FS1: Résistances du comparateur à hystérésis

Nous cherchons ici à faire une comparaison entre un seuil de $V_{CC}/2$ et un signal audio converti en signal électrique qui oscille entre 0 et V_{CC} et est centré sur $V_{CC}/2$. On utilise ainsi un MCP6292 avec un montage en hystérésis pour deux raisons: nous sommes en très basse tension et donc il n'est pas nécessaire de prendre un TL081/82 et aussi le signal de sortie aura beaucoup de perturbation sans montage en hystérésis. Le but ici est de faire un seuil de $V_{CC}/2$ avec un pont diviseur.



En sachant que le seuil de la tension de la broche positive, V_+ , doit être de $V_{CC}/2$, il nous faut un pont diviseur qui divise par deux.

$V(\text{sortie})$ correspond à la tension de sortie du MCP6292;

A correspond au coefficient qui doit être égale à ½.

V(+) correspond à la broche positive.

$$V(sortie) = A * V(+) \Leftrightarrow A = \frac{V(sortie)}{V(+)} = \frac{1}{2}$$

Ainsi, en choisissant une résistance R de 1k Ohm et un potentiomètre Rp de 1k Ohm maximum pour régler le seuil de basculement.

$$A = \frac{R}{R + R_p} = \frac{1}{2}, \text{ si } R_p = 1k \text{ Ohm}$$

Datasheet du MAX4465:

<https://cdn-shop.adafruit.com/datasheets/MAX4465-MAX4469.pdf>

Datasheet du MCP6292

<https://www.ti.com/lit/ds/symlink/mcp6292.pdf>

3.2 FSx :

4 Documents de contrôle de fabrication

4.1 Typon – Visuel Gerber

Le typon fourni pour fabrication est représenté en Figure 1. En cas de différence, contacter l'équipe de conception pour vérification.

Figure 2 - Typon du système

[Date de publication] Version : x.x 3

BUT 1 – SAE S1

Dossier de fabrication – [Système]



4.2 Plan de montage (visuel 3D)

En guise de plan de montage, les Figure 3 et Figure 4 donnent la vue 3D de la carte conçue à l'aide du logiciel de CAO.

Figure 3 - Vue du dessus du système Figure 4 - Vue de dessous du système

5 Nomenclature

L'ensemble des composants est résumé dans le tableau #ref. Ce tableau donne la référence du composant sur le schéma, la valeur de sa propriété physique, le cas échéant, ainsi qu'une référence fournisseur et un prix associé pour évaluer le coût de fabrication.

Tableau 1 - Nomenclature de fabrication et estimation de coût

Référence schéma	Propriété	Référence fournisseur	Coût unitaire
	Guide de guitare		0
	Systeme de maintien		0
	compartiment électronique		0
			0.10€
...	
		Total	00,0000€

D'après les références fournisseurs (toutes les informations ont été prises chez **#refFournisseur#**), le coût total composants serait de 1,50€.

[Date de publication] Version : x.x 4