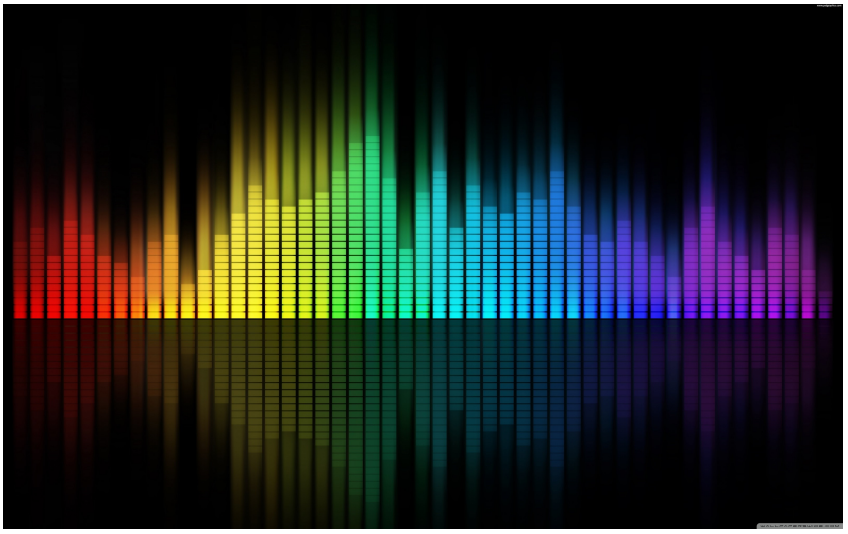
ENSEA - Année 2020-2021

Séquenceur Eurorack



Professeurs référents :
Laurent Fiack
Nicolas Papazoglou

Réalisé par :
Célia Jaffré
Oliver Michalowski

9 mai 2021

Table des matières

En musique, il pourrait souvent sembler que nous avons fait le tour des mélodies que les compositeurs ont à nous proposer et que les mélodies d'aujourd'hui ne sont que des variations de celles d'hier. En effet l'être humain s'inspire de ce qui l'entoure c'est pourquoi il est parfois difficile de trouver de nouvelles idées sans être tenté de simplement moduler les mélodies existant déjà.
En recherchant la nouveauté, la musique s'est orientée vers de nouvelles sonorités grâce à la musique électronique, qui a également permis de trouver de nouvelles inspirations en termes de mélodie. Dans l'univers musical, la synthèse aléatoire est donc une chance pour apporter un air frais à la musique que nous écoutons.
Dans cet esprit, nous avons travaillé sur un séquenceur à génération aléatoire de mélodie. Le travail de composition consiste à retenir les mélodies les plus satisfaisantes. Ce module est conçu pour fonctionner avec un système de synthétiseur modulaire Eurorack.

1 Présentation du séquenceur

1.1 Le fonctionnement d'un séquenceur

L'Eurorack est un format de synthétiseur modulaire. Ce sont des synthétiseurs où le son et les sources de modulation sonores n'ont pas d'architecture figée puisque les connexions entre les blocs se font par cordons audio par opposition aux synthétiseurs semi-modulaires et les synthétiseurs monocircuits dont l'essentiel du routage entre les blocs est déjà fait.
Quelques exemples de modules sont les sources sonores (VCO, générateurs de bruit blanc/rose), filtres (VCF ou "voltage controlled filter"), sources de modulation (LFO ou "low frequency oscillator", et générateurs d'enveloppe), fonctions spécifiques (conversion MIDI $\rightarrow$ CV, sources d'horloges, distributeurs et mélangeurs de signaux), sources de commande musicale (claviers type piano/orgue, rubans, écrans tactiles, joysticks). En particulier notre séquenceur pourrait remplacer un clavier si on voulait jouer une mélodie particulière.
Créé dans les années 1990, Eurorack est de nos jours, le standard de synthétiseur modulaire le plus populaire. En outre il donne les dimensions que doit respecter notre module : il faut que notre face avant soit environ 3U soit 133,35 mm en hauteur et un nombre entier de HP, 1 HP équivalant à 5,08 mm. Cette taille standardisée permet d'installer plus facilement le module sur les rails prévus à cet effet.

Un séquenceur, comme son nom l'indique joue une séquence de notes et la répète. Ces notes représentent essentiellement deux types de messages : un message Volt/Octave (l'image de la note que l'on jouerait sur un clavier, idéal pour commander un oscillateur, pour jouer une mélodie), et un message logique Trigger (l'image

d'un déclenchement de note, notamment adapté pour les générateurs d'enveloppe). Les séquenceurs se distinguent principalement par leur façon de jouer et générer les séquences de notes. Dans notre cas la séquence de notes est générée aléatoirement, dans le cadre des paramètres choisis par l'utilisateur.

1.2 Notre séquenceur

1.2.1 Les technologies utilisées

L'intérieur du séquenceur est construit autour d'une carte STM32L432KC. Les entrées et sorties électroniques sont adaptées à l'aide d'amplificateurs (type TL084) et pont diviseurs résistifs.
Pour l'interface, potentiomètres, boutons poussoirs, encodeurs et prises jack 3.5mm sont nécessaires. L'affichage de la séquence est réalisée à l'aide d'une matrice de LEDs 8x8.
Les périphériques d'intérêts de STM32 pour ce projet sont :

— RNG pour la génération aléatoire essentielle à la création de notes
— ADC/DAC pour la gestion des CV (lesquels sont analogiques)
— TIM pour la gestion du temps
— I2C pour piloter la Matrice LED tricolore ADAFRUIT 8x8

L'alimentation est fournie avec le rack, les niveaux d'entrée-sortie et les dimensions sont spécifiées par le standard Eurorack.

1.2.2 Les fonctionnalités implémentées

Notre séquenceur comprend 8 pas (1 séquence), à chaque pas nous avons 1 note à choisir parmi 8 notes possibles. Ces 8 notes dépendent des choix de l'utilisateur : il choisit la tonique, soit la note fondamentale de la gamme, puis il choisit le type de gamme à travers le mode : majeur, mineur mélodique, mineur harmonique, blues. Il peut choisir d'étaler ces notes sur plus d'octaves si nécessaire à l'aide d'un potentiomètre ou d'une entrée jack 3.5mm, cette commande contrôle l'étalement. La séquence peut aller dans plusieurs sens différents (gauche, droite, gauche et droite ou bien suivant un pas aléatoire). Cette commande se nomme la Direction.
Ce qui est particulier dans notre projet est la décision de générer des séquences aléatoires et non programmées "en note par note" par l'utilisateur. L'utilisateur peut sauvegarder les séquences qui retiennent son intérêt et effacer les séquences qui ne le satisfont pas.

2 Réalisation

2.1 Le routage STM32

2.1.1 les ports GPIO finaux

Le centre du séquenceur est la carte STM32 NUCLEO32 basée sur la puce STM32L432KC cadencée à 80MHz et 64kio de RAM et 256kio mémoire flash. La carte Nucleo a le même format que les puces Arduino Nano. Il a été décidé d'utiliser au maximum les entrées et sorties de la carte Nucleo. Sur la réalisation finale, nous disposons de 15 ports GPIO disponibles :

- 2 ports I2C alloués pour la Matrice de LEDs
- 1 port DAC lié au CV_OUT [1V/OCTAVE]
- 3 ports ADC respectivement liés aux $CV\_MODE, CV\_TONIC\_1, CV\_TONIC\_2, MODE\_DEFIL$
- 2 Sorties logiques pour les sorties $TRIG\_OUT$, $LAST\_STEP$
- 3 Entrée logiques pour les entrées CLOCK, $NEXT\_SEQ$, $PREV\_SEQ$
- 2 Entrées encodeur $WHEEL\_A$, $WHEEL\_B$
- 1 Entrée bouton encodeur $WHEEL\_BTN$

2.1.2 Réglage des ponts de soudure

On croit avoir 20 GPIO disponibles a priori si on retire les NRST, GND, VIN (et autres alimentations). Cependant il y a des ponts de soudure qu'il faut configurer et certains ports Arduino (nommés Ax et Dx) n'ont pas été libérés. Voici les ports problématiques :

- A0 : L'entrée analogique SPREAD n'a pas été utilisée car cette entrée est en conflit avec le port MCO
- A4 : $ADC12\_10$ est non-opérationnel car il est court-circuité par D4 correspondant à la sortie I2C1 SB18 ON
- A5 : $ADC12\_11$ est également non-opérationnel car il est court-circuité par la sortie D5 : I2C1 SB16 ON

Les ports concernant wheel green et wheel red sont remplacés par les ports liés à l'I2C.

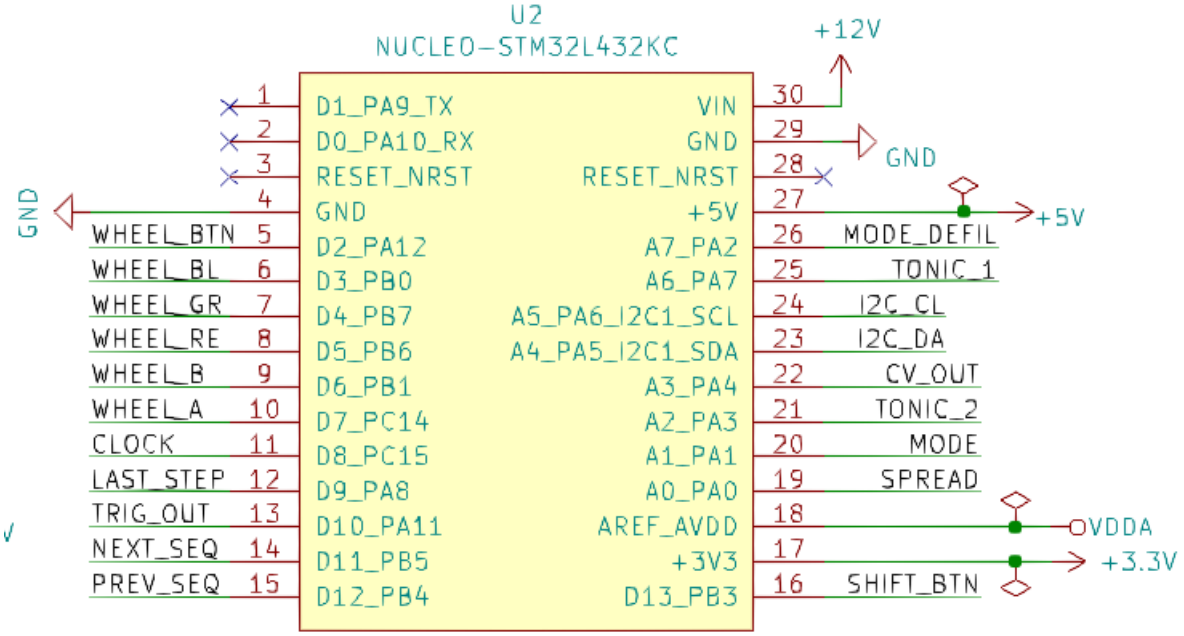
- D0/D1 : VCP_TX/RX Le débogueur nécessite d'avoir ces deux ports réservés si on veut interagir avec un terminal, par exemple pour calibrer de façon manuelle et non de manière "hard coding". On ne peut donc pas utiliser ces deux ports pour des entrées/sorties

— D7/D8 : $OSC32\_IN/OUT$ PC14 PC15. Pour avoir accès à D7 et D8 en entrées/sorties il faut configurer les pont de soudures SB5 à SB8 de la manière suivante :
SB5/SB7 doivent passer de l'état ON à l'état OFF (déconnexion STM32 de X1)
SB6/SB8 doivent passer de l'état OFF à l'état ON (connexion STM32 avec D7 et D8)
(par défaut à LSE ON avec le résonateur X1)

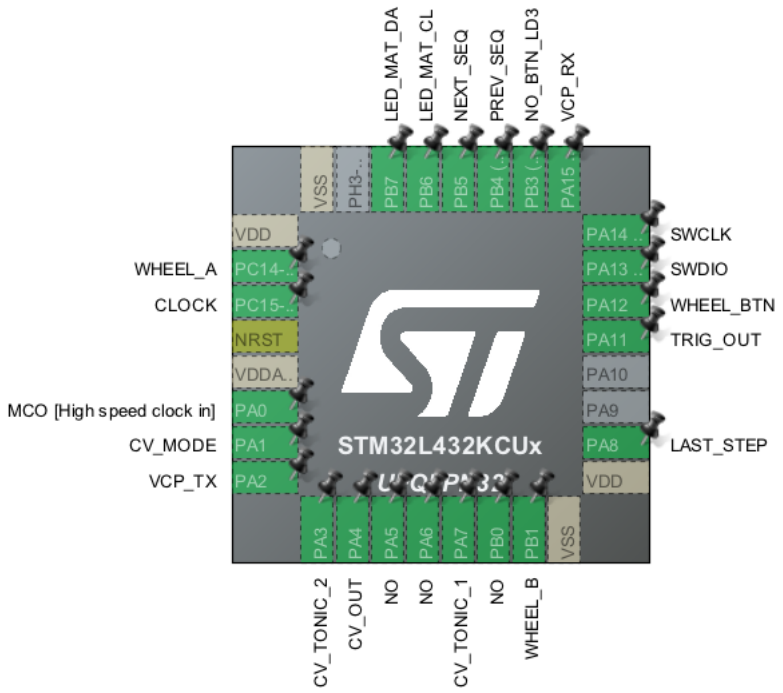
Les ports concernant Clock et Wheel A sont inutilisables si l'on effectue pas un changement de configuration des ponts de soudure.

— D13 : LD3 est connecté au monde extérieur à cause de l'état ON de SB15

Le bouton poussoir est donc inutilisable.



Voici les entrées et sorties de la carte STM32L432KC observées pour le routage du circuit imprimé sur le logiciel Kicad.



Ci-dessus la configuration de chaque entrée et sortie au sein du logiciel STM32CubeIDE.

2.1.3 Le calibrage

En entrée de l'ADC on passe un n_{ADC} qui dépend de la note MIDI (entre 0 et 127), Les formules pour le calibrage sont donc les suivantes :

$$n_{ADC} = gain_{correction} * (note_{MIDI} + offset_{correction})$$

$$gain_{correction} = gain_{ideal} * (n1_{cal} - n2_{cal})$$

$$offset_{correction} = \frac{(n1_{cal} - n2_{cal}) * offset_{ideal} + n2_{cal} * v1_{precorrection} - n1_{cal} * v2_{precorrection}}{v1_{precorrection} - v2_{precorrection}}$$

Un exemple de l'implémentation de la sortie du DAC avec *conversion_table* et les tensions mesurées pré-correction figure en annexe, avant-dernière page.

2.2 La programmation de la carte STM32L432KC

2.2.1 Le formalisme choisi

La gestion de la mémoire, des séquences et des paramètres a été implémentée à l'aide de structures. Pour pouvoir traiter leurs données il suffit alors de passer leur adresse

en paramètres de fonctions. par exemple : quand on veut réécrire la mémoire flash : $seq\_write\_mem(\&structure\_memoire)$;
ou alors pour afficher la séquence en prenant en compte un paramètre :
$seq\_disp\_step(\&structure\_sequence, \&structure\_parametres)$ Ces structures apparaissent dans l'annexe, en avant-dernière page).

2.2.2 L'affichage

Dans ce projet, l'affichage est très important. En effet , il permet de visualiser les notes qui vont être jouées mais également de pouvoir naviguer entre les séquences et afficher certaines informations.
Pour cet affichage nous avons choisi d'utiliser la matrice carrée de 64 LED BLM12X883XX d'Adafruit. Il s'agit d'une matrice de LEDs tricolores ce qui nous permet d'apporter également des informatins par la nuance dans les couleurs.

Tout d'abord nous devons procéder à quelques initialisations, nous débutons par la fonction MX_I2C1_Init présentée en annexe (dernière page). Cette fonction permet d'initialiser la communication en I2C. Ensuite nous initialisons les paramètres de la matrice de LEDs qui nous intéressent à l'aide d'un buffer, ici I2CMasterBuffer.

La commande suivante permet d'allumer l'oscillateur interne :
I2CMasterBuffer[0] = 0x21 ;
$HAL\_I2C\_Master\_Transmit$ (&hi2c1,$HT16K33\_I2C\_ADDRESS$, I2CMasterBuffer, 1, HAL_MAX_DELAY) ;

Voici maintenant la commande pour supprimer le scintillement ainsi qu'allumer l'affichage :
I2CMasterBuffer[0] = 0x81 ;
$HAL\_I2C\_Master\_Transmit$ (&hi2c1,$HT16K33\_I2C\_ADDRESS$, I2CMasterBuffer, 1, HAL_MAX_DELAY) ;

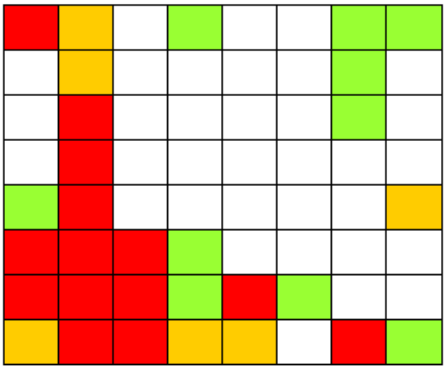
Et cette commande permet de régler l'éclairage des LEDs à une intensité élevée :
I2CMasterBuffer[0] = 0xE5 ;
$HAL\_I2C\_Master\_Transmit$ (& hi2c1,$HT16K33\_I2C\_ADDRESS$, I2CMasterBuffer, 1, HAL_MAX_DELAY) ;

Cette matrice fonctionne par le biais d'un buffer. Ce buffer contient 17 octets. L'élément [0] étant la commande '0x00' les 4 premiers bits 0000 signifient que l'on s'apprête à écrire dans la mémoire des LEDs et les 4 derniers bits 0000 signifient que la prochaine information correspondra à la première colonne.à partir de la première

colonne. Les 16 octets suivants correspondent tour à tour à la composante verte puis rouge de chaque colonne de LED pixel. Chaque composante du buffer est codée sur 8 bits correspondant à l'état allumé ou éteint de chaque LED d'une même colonne, le bit de poids fort correspondant à la LED de la partie supérieure de la matrice. Voici un exemple ci-dessous.

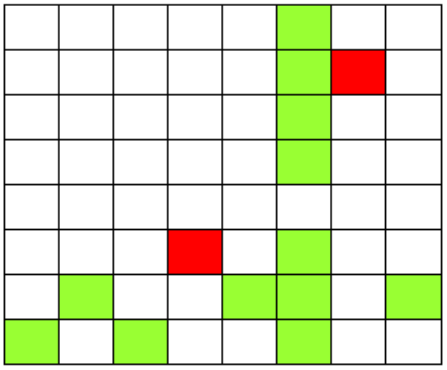
0	1	1	1	0	0	1	0	0	0	0	0	1	0	1	0
0	0	1	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0
0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	1
0	1	0	1	0	1	1	0	0	0	0	0	0	0	0	0
0	1	0	1	0	1	1	0	0	1	1	0	0	0	0	0
1	1	0	1	0	1	1	1	1	1	0	0	0	1	1	0
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Ici la première composante du buffer s'écrira 00001001 ou 09 son équivalent en hexadécimal ce qui correspond à la première colonne du tableau.
Lorsque ce buffer est transmis à la LED par la fonction $HAL\_I2C\_Master\_Transmit$ nous obtenons l'affichage suivant sur la matrice :



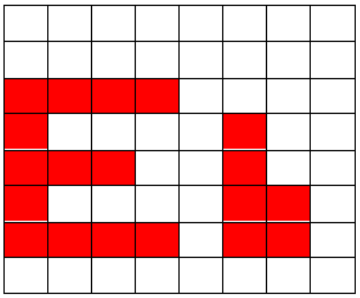
Lorsque la composante rouge et la composante verte se superposent elles forment une composante orange.
La première fonction d'affichage implémentée permet d'afficher les notes générées ainsi que le suivi en temps réel de la note jouée. Cette fonction considère la gamme

prise en compte afin d'associer une couleur verte aux tons, rouge aux demi-tons et orange pour les ton-et-demis. Cette fonction considère également l'entrée d'horloge car l'affichage doit se faire en simultané avec la musique. Et elle reçoit également un tableau de notes générées au préalable de manière aléatoire sous forme de liste de longueur 8 dont les éléments sont des entiers entre 1 et 8.
Voici un résultat ci-dessous pour une gamme majeure.



La bande verte correspond à l'instant qui est en train d'être joué et le pixel qui n'est pas éclairé correspond à la note jouée.

La seconde fonction d'affichage implémentée permet d'afficher la note qui sera la tonique de la gamme, L'utilisateur peut faire ce choix à l'aide d'un potentiomètre ou bien à l'aide d'une entrée jack 3.5 mm. Nous utilisons directement cette tension transmise par l'un des deux biais pour déterminer la note choisie. Nous définissons donc des marches auxquelles une note est associée, ceci modulo 12 car il existe 12 demi-tons dans une octave. Nous affichons donc la note sélecionnée afin que l'utilisateur visualise son choix.
Voici un mi bémol en prototype :



Voilà le rendu sur la matrice de LED d'un Si bémol.
Nous avons choisi l'écriture en lettre pour les notes car il s'agit d'une norme universelle.

Les fonctions réalisées se trouvent en annexe.

3 Conclusion

Lors des premières séances nous avons cerné le sujet, commandé le matériel dont nous avions besoin et nous avons beaucoup échangé pour nous assurer d'être en accord sur le fonctionnement fondamental de notre projet. Les tâches ont définies au début mais nous les avons adaptées afin d'être plus efficaces. Nous avons également réfléchi sur certains détails de fonctionnement comme la gestion de la mémoire qui s'adapte mieux au matériel choisi. Nous nous sommes d'abord focalisé sur les fonctions les plus importantes de notre séquenceur, à savoir fournir les informations nécessaires au générateur d'enveloppe et à l'oscillateur pour l'écoute de notes créées. Puis nous avons réalisé les fonctions secondaires à savoir les paramètres que peut choisir l'utilisateur : la tonique, le mode.. Nous n'avons pas été jusqu'à l'étape finale qui consiste en la réalisation de la face avant, deux essais ont été effectués sans grand succès car nos avons fait face à un problème de concordance des parties percées avec les composants volumineux de la carte. Le sujet de la musique nous passionne et nous avons partagé ce dynamisme à travers la réalisation de notre projet.

Références

`https://fr.wikipedia.org/wiki/I2C`
Concernant la documentation relative à l'I2C.

`https://sdiy.info/wiki/Eurorack`
Concernant l'alimentation de l'Eurorack.

`http://www.doepfer.de/a100_man/a100m_e.htm`
Concernant l'alimentation Eurorack Doepfer.

`http://www.doepfer.de/a100_man/a100t_e.htm`
Concernant les tensions relatives à l'Eurorack Doepfer

HT16K33 Datasheet
Concernant l'interface I2C de la matrice de LED

UM1956 User Manual & STML432KC Datasheet
Concernant la carte STM32L432KC

EC12PLRGBSDVBF-D-25K-24-24C-61/08-6H Datasheet
Concernant l'encodeur RGB

Annexe

Annexe Fonction d'affichage des séquences de notes

```
void SEQ_DispStep(seq_TypeDef* sequence, seq_parametres_TypeDef* p){
    for (int i = 0 ; i < 17 ; i++) {
        sequence->leds[i] = 0;
      }
    if (strcmp(p->mode,"ma")==0){ //majeur
        for (int j = 1; j < 17; j+=2){

            if ((sequence->notes[j/2+1]==2)||(sequence->notes[j/2+1]==6)){
                sequence->leds[j] = 0;
                if((j==sequence->step)||(j+1 == sequence->step)){
                    for (int k = 0; k < 16; k++){
                        if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                            sequence->leds[j] += 1<<k;
                        }
                    }
                }
                else{
                    sequence->leds[j] = 1<<(sequence->notes[j/2+1]);
                }
            }
            else{
                sequence->leds[j+1] = 0;
                if((j==sequence->step)||(j+1 == sequence->step)){
                    for (int k = 0; k < 16; k++){
                        if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                            sequence->leds[j+1] += 1<<k;
                        }
                    }
                }
                else{
                    sequence->leds[j+1] = 1<<(sequence->notes[j/2+1]);
                }
            }
        }
        HAL_I2C_Master_Transmit (&hi2c1, HT16K33_I2C_ADDRESS, sequence->leds, 17, HAL_MAX_DELAY);
    }
```

```
else if (strcmp(p->mode,"mm")==0){ //mineur mélodique
    for (int j = 1; j < 17; j+=2) {

        if ((sequence->notes[j/2+1]==1)||(sequence->notes[j/2+1]==6)){
            sequence->leds[j] = 0;
            if((j==sequence->step)||(j+1 == sequence->step)){
                for (int k = 0; k < 16; k++){
                    if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                        sequence->leds[j] += 1<<k;
                    }
                }
            }
            else{
                sequence->leds[j] = 1<<(sequence->notes[j/2+1]);
            }
        }
        else{
            sequence->leds[j+1] = 0;
            if((j==sequence->step)||(j+1 == sequence->step)){
                for (int k = 0; k < 16; k++){
                    if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                        sequence->leds[j+1] += 1<<k;
                    }
                }
            }
            else{
                sequence->leds[j+1] = 1<<(sequence->notes[j/2+1]);
            }
        }
    }
    HAL_I2C_Master_Transmit (&hi2c1, HT16K33_I2C_ADDRESS, sequence->leds, 17, HAL_MAX_DELAY);
}
```

```
else if (strcmp(p->mode,"mh")==0){ //mineur harmonique
    for (int j = 1; j < 17; j+=2) {

        if ((sequence->notes[j/2+1]==1)||(sequence->notes[j/2+1]==4)||(sequence->notes[j/2+1]==6)){
            sequence->leds[j] = 0;
            if((j==sequence->step)||(j+1 == sequence->step)){
                for (int k = 0; k < 16; k++){
                    if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                        sequence->leds[j] += 1<<k;
                    }
                }
            }
            else{
                sequence->leds[j] = 1<<(sequence->notes[j/2+1]);
            }
        }
        else if ((sequence->notes[j/2+1]==1)||(sequence->notes[j/2+1]==4)||(sequence->notes[j/2+1]==6)){
            sequence->leds[j] = 0;
            sequence->leds[j+1] = 0;
            if((j==sequence->step)||(j+1 == sequence->step)){
                for (int k = 0; k < 16; k++){
                    if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                        sequence->leds[j] += 1<<k;
                        sequence->leds[j+1] += 1<<k;
                    }
                }
            }
            else{
                sequence->leds[j] = 1<<(sequence->notes[j/2+1]);
                sequence->leds[j+1] = 1<<(sequence->notes[j/2+1]);
            }
        }
```

```
        else{
            sequence->leds[j+1] = 0;
            if((j==sequence->step)||(j+1 == sequence->step)){
                for (int k = 0; k < 16; k++){
                    if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                        sequence->leds[j+1] += 1<<k;
                    }
                }
            }
            else{
                sequence->leds[j+1] = 1<<(sequence->notes[j/2+1]);
            }
        }

    }
    HAL_I2C_Master_Transmit (&hi2c1, HT16K33_I2C_ADDRESS, sequence->leds, 17, HAL_MAX_DELAY);
}
else if (strcmp(p->mode,"bl")==0){ //blues
    for (int j = 1; j < 17; j+=2) {

        if ((sequence->notes[j/2+1]==1)||(sequence->notes[j/2+1]==5)||(sequence->notes[j/2+1]==7)){
            sequence->leds[j] = 0;
            if((j==sequence->step)||(j+1 == sequence->step)){
                for (int k = 0; k < 16; k++){
                    if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                        sequence->leds[j] += 1<<k;
                    }
                }
            }
            else{
                sequence->leds[j] = 1<<(sequence->notes[j/2+1]);
            }
        }
```

```
            else if ((sequence->notes[j/2+1]==0)||(sequence->notes[j/2+1]==4)||(sequence->notes[j/2+1]==6)){
                sequence->leds[j] = 0;
                sequence->leds[j+1] = 0;
                if((j==sequence->step)||(j+1 == sequence->step)){
                    for (int k = 0; k < 16; k++){
                        if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                            sequence->leds[j] += 1<<k;
                            sequence->leds[j+1] += 1<<k;
                        }
                    }
                }
                else{
                    sequence->leds[j] = 1<<(sequence->notes[j/2+1]);
                    sequence->leds[j+1] = 1<<(sequence->notes[j/2+1]);
                }
            }
            else{
                sequence->leds[j+1] = 0;
                if((j==sequence->step)||(j+1 == sequence->step)){
                    for (int k = 0; k < 16; k++){
                        if ((k<sequence->notes[j/2+1])||(k>sequence->notes[j/2+1])){
                            sequence->leds[j+1] += 1<<k;
                        }
                    }
                }
                else{
                    sequence->leds[j+1] = 1<<(sequence->notes[j/2+1]);
                }
            }

        }
        HAL_I2C_Master_Transmit (&hi2c1, HT16K33_I2C_ADDRESS, sequence->leds, 17, HAL_MAX_DELAY);
    }

}
```

Annexe Fonction d'affichage de la note tonique choisie ci-après

```
void SEQ_DispNote(seq_TypeDef* sequence, seq_parametres_TypeDef* p){
    for (int i = 0 ; i < 17 ; i++) {
        sequence->leds[i] = 0;
    }

    if ((p->tonique-57)%12==0) {//A REFAIRE PAREIL
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4);
        sequence->leds[4] = (1 << 2) + (1 << 5);
        sequence->leds[6] = (1 << 2) + (1 << 5);
        sequence->leds[8] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4);
    }
    else if (((p->tonique-58)%12 == 0)||((p->tonique-59)%12 == 0)) {//B
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[4] = (1 << 1) + (1 << 3) + (1 << 5);
        sequence->leds[6] = (1 << 1) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[8] = (1 << 1) + (1 << 2) + (1 << 3);
    }
    else if (((p->tonique-60)%12==0)||((p->tonique-61)%12==0)) {//C
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[4] = (1 << 1) + (1 << 5);
        sequence->leds[6] = (1 << 1) + (1 << 5);
        sequence->leds[8] = (1 << 1) + (1 << 5);
    }
    else if ((p->tonique-62)%12==0) {//D
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[4] = (1 << 1) + (1 << 5);
        sequence->leds[6] = (1 << 1) + (1 << 5);
        sequence->leds[8] = (1 << 2) + (1 << 3) + (1 << 4);
    }
    else if (((p->tonique-63)%12==0)||((p->tonique-64)%12==0)) {//E
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[4] = (1 << 1) + (1 << 3) + (1 << 5);
        sequence->leds[6] = (1 << 1) + (1 << 3) + (1 << 5);
        sequence->leds[8] = (1 << 1) + (1 << 5);
    }
```

```
    else if (((p->tonique_65)%12==0)||((p->tonique-66)%12==0)) {//F
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[4] = (1 << 3) + (1 << 5);
        sequence->leds[6] = (1 << 3) + (1 << 5);
        sequence->leds[8] = (1 << 5);
    }
    else if (((p->tonique-55)%12==0)||((p->tonique-56)%12==0)) {//G
        sequence->leds[2] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[4] = (1 << 1) + (1 << 5);
        sequence->leds[6] = (1 << 1) + (1 << 3) + (1 << 5);
        sequence->leds[8] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 5);
    }
    else {
        sequence->leds[2] = 0;
        sequence->leds[4] = 0;
        sequence->leds[6] = 0;
        sequence->leds[8] = 0;
    }

    if (((p->tonique-56)%12==0)||((p->tonique-61)%12==0)||((p->tonique-66)%12==0)) {//Sharp
        sequence->leds[12] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
        sequence->leds[14] = (1 << 2) + (1 << 4);
        sequence->leds[16] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4) + (1 << 5);
    }
    else if (((p->tonique-58)%12==0)||((p->tonique-63)%12==0)) {//Flat
        sequence->leds[12] = (1 << 1) + (1 << 2) + (1 << 3) + (1 << 4);
        sequence->leds[14] = (1 << 1) + (1 << 2);
        sequence->leds[16] = 0;
    }
    else {
        sequence->leds[12] = 0;
        sequence->leds[14] = 0;
        sequence->leds[16] = 0;
    }
    HAL_I2C_Master_Transmit(&hi2c1, HT16K33_I2C_ADDRESS, sequence->leds, 17, HAL_MAX_DELAY);
    HAL_Delay(100);
}
```

```
  const int32_t n1_cal = 1927;
  const int32_t n2_cal = 2167;
  const float v1_pre_correction = 1.0078; //V pré-correction
  const float v2_pre_correction = 2.2395; //V pré-correction
  const float gain_ideal = 1./12.;
  const float offset_ideal = -gain_ideal * 60;

//const float offset_correction = 0;
//const float gain_correction = 1;

  const float gain_correction = gain_ideal
          * (n1_cal - n2_cal)
          /(v1_pre_correction - v2_pre_correction);
  const float offset_correction =
          (
                  (n1_cal - n2_cal) * offset_ideal
                  + n2_cal * v1_pre_correction
                  - n1_cal * v2_pre_correction)
          /(v1_pre_correction - v2_pre_correction);

  int32_t temp;
  uint32_t i;

  for(i=0; i<0x1000; ++i)
  {
      temp = gain_correction * i + offset_correction;
      if(temp>0xFFF) temp = 0xFFF;
      else if(temp<0) temp = 0;
      conversion_table[i] = (uint16_t) (temp);
  }
```

Annexe Figure 2.1.3 - Calibrage

```
45 /**
46  * Le type de structure de prise en charge d'une sequence jouee.
47  */
48 typedef struct seq_TypeDef
49 {
50     uint8_t notes[8];
51     seq_leds leds;
52     uint8_t step;
53 } seq_TypeDef;
54
55 typedef struct seq_memoire_TypeDef
56 {
57     seq_TypeDef memoire[8];
58     uint8_t  def[8]; /* 0 ou 1 ou 2 : est-ce que la séquence est réservée en mémoire, ou aléatoire */
59     uint8_t position; /* position en memoire */
60 } seq_memoire_TypeDef;
```

Annexe Figure 2.2.1 - Formalisme

```
static void MX_I2C1_Init(void)
{

  hi2c1.Instance = I2C1;
  hi2c1.Init.Timing = 0x00000E14;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }
  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }

}
```

Annexe Figure 2.2.2 - Affichage