Projet 441 - Expandeur MIDI

Saša Radosavljevic

Janvier 2021

Table des matières

1 Introduction

Au cours de notre formation en informatique embarquée, nous avons à réaliser un mini-projet appliquant les connaissances théoriques et pratiques du semestre. Pour ce mini-projet, le sujet que j'ai choisi exploite la liaison série UART ainsi que le convertisseur numérique-analogique (DAC) afin de réaliser un expandeur MIDI.

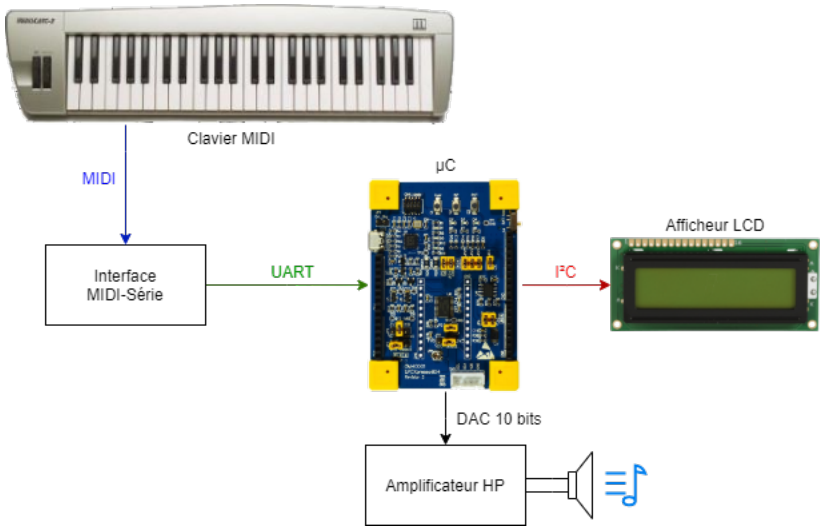


Figure 1: Synoptique

On relie ainsi un clavier numérique MIDI à notre carte micro-contrôleur et une sortie audio (Petit haut-parleur).

2 MIDI

2.1 Interface MIDI

Dans un premier temps, il a fallu réaliser une interface MIDI afin de lire les données envoyées par le clavier. On retrouve ainsi le schéma de câblage de cette interface :

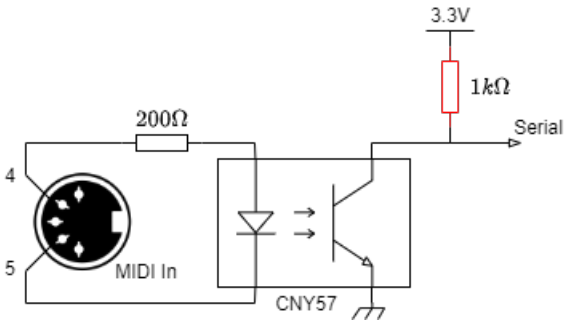


Figure 2: Interface MIDI

Différents tests de câblage on pu être réaliser avant de trouver une valeur de résistance adaptée à notre utilisation (1kΩ). En effet, pour des résistances de valeurs différentes, les données en liaison série n'étaient plus lisible, état bas pas assez long et net.

Résistance	4,7 kΩ	1 kΩ	470 Ω
0V	Oui	Oui	Non
t_r	Mauvais	Bon	X

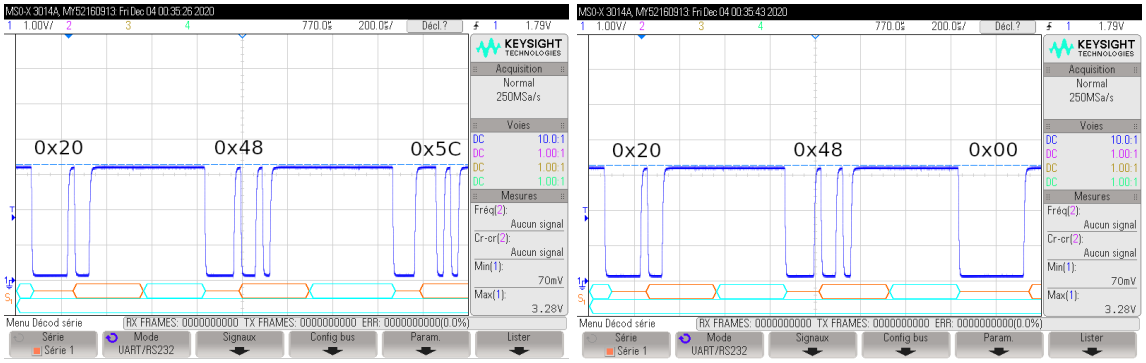
Une fois l'interface complétée, nous pouvons maintenant nous intéresser à la norme MIDI.

2.2 Norme MIDI

La norme MIDI (Musical Instrument Digital Interface) est un protocole de communication dédié à la musique et utilisé pour la communication entre instruments électroniques, contrôleurs, séquenceurs et logiciels de musique.

Notes de musique

Le protocole définit les messages note-on et note-off pour déclencher et arrêter chaque note. Une vélocité est associée à chaque note, permettant d'indiquer si la note est plus ou moins jouée fortement. Les notes de l'échelle chromatique sont représentées par un nombre entier codé sur 7 bits, permettant de coder plus de 10 octaves : du C-1 (note 0) ou C0 en notation grégorienne au G9 (note 127, resp G10) avec une résolution d'un demi-ton.
Voici par exemple l'acquisition de note-on et note-off dont la seule information a-priori est que la note est un C (Do) :



(a) C on (b) C off

Figure 3: Appui sur un C (Do) dont on ne connaît pas l'octave

Ainsi, on observe 3 octets. Le premier étant réservé à des commandes particulières, nous ne nous y intéresserons pas. Le deuxième octet donne l'information sur la touche appuyée, dans notre cas la note C est un multiple de 12, ainsi 0x48 ou plutôt $72/12 = 6$. On se trouve ainsi sur l'octave 6 (grégorienne). Le dernier octet permet, lui, d'indiquer la vélocité d'appui sur une intervalle de [0x00, 0x7F]. On comprend alors que l'appui était assez prononcé.

3 Acquisition des données

3.1 Lecture des données

Une fois notre entrée Serial (Cf. Figure 2) branchée en entrée de UART_RX (P0_20), nous devons configurer la communication. On a les paramètres suivants : 31250 bauds, 8 bits de données, pas de parité, 1 bit de stop.
On propose enfin de réceptionner les octets de données par interruption sur la liaison série, obtenant ainsi l'algorigramme suivant :

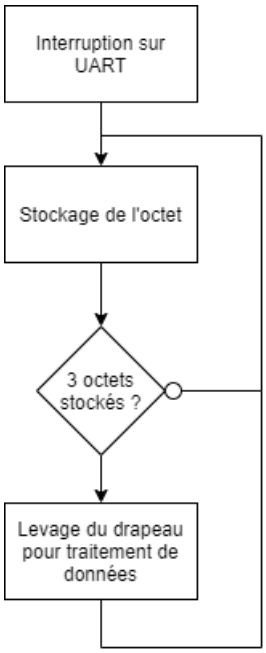


Figure 4: Algorigramme d'acquisition des données

3.2 Décodage

Maintenant que l'on récupère les données du clavier, on se concentre sur la note jouée, c'est-à-dire le deuxième octet de donnée. En se basant sur une référence de C0 (Do 32,7 Hz) équivalent à 0x00, l'information de la note est image à sa position (A4 = 0x39, B4 = 0x3B). En prenant le reste de la division de la valeur de la note par 12 (nombre de notes par octave), on obtient ainsi la note appuyée.

```
1  // Getting key's number
2  temp_char = rx_buffer[1]%12+1;
3  note[2] = ' ';
4  octave = (rx_buffer[1]-rx_buffer[1]%12)/12;
5  // First octave reference multiplied by the key's octave
6  switch (temp_char) {
7      case 20: break;
8      case 1:  Freq = (uint32_t)  32.7<<octave; note[0] = 'C'; note[1] = (unsigned char) '1'+
       octave; break;
9      case 2:  Freq = (uint32_t) 34.64<<octave; note[0] = 'C'; note[1] = '#'; note[2] = (
       unsigned char) '0'+octave; break;
10     case 3:  Freq = (uint32_t)  36.7<<octave; note[0] = 'D'; note[1] = (unsigned char) '1'+
       octave; break;
11     case 4:  Freq = (uint32_t)  38.9<<octave; note[0] = 'D'; note[1] = '#'; note[2] = (
       unsigned char) '0'+octave; break;
12     case 5:  Freq = (uint32_t)  41.2<<octave; note[0] = 'E'; note[1] = (unsigned char) '1'+
       octave; break;
13     case 6:  Freq = (uint32_t) 43.65<<octave; note[0] = 'F'; note[1] = (unsigned char) '1'+
       octave; break;
14     case 7:  Freq = (uint32_t) 46.25<<octave; note[0] = 'F'; note[1] = '#'; note[2] = (
       unsigned char) '0'+octave; break;
15     case 8:  Freq = (uint32_t)    49<<octave; note[0] = 'G'; note[1] = (unsigned char) '1'+
       octave; break;
16     case 9:  Freq = (uint32_t)  51.9<<octave; note[0] = 'G'; note[1] = '#'; note[2] = (
       unsigned char) '0'+octave; break;
17     case 10: Freq = (uint32_t)    55<<octave; note[0] = 'A'; note[1] = (unsigned char) '1'+
       octave; break;
18     case 11: Freq = (uint32_t) 58.27<<octave; note[0] = 'A'; note[1] = '#'; note[2] = (
       unsigned char) '0'+octave; break;
19     case 12: Freq = (uint32_t) 61.73<<octave; note[0] = 'B'; note[1] = (unsigned char) '1'+
       octave; break;
```

```
20      default:
21      case '0': LPC_DAC0->CTRL = 0;
22              LPC_DAC0->CR = 512;
23  }
```

decodage.c

De plus, l'information quant à la vélocité d'appui sur une touche est présente sur le 3e octet. Ainsi, une simple lecture de celui-ci donne une vélocité comprise entre 0x00 et 0x7F.
Il est alors possible de traiter toutes ces informations, que se soit l'affichage ou la génération du signal audio.

4 Génération d'un signal sonore

4.1 Sinus échantillonné

La première méthode, la plus simple à mettre en place est la génération d'un sinus sur base d'échantillons d'un sinus que l'on génère sur le DAC avec une fréquence d'échantillonnage correspondant à la fréquence de la note jouée.

Pour notre DAC, de résolution de 10 bits, on se propose d'utiliser 32 échantillons pour rendre le signal le plus sinusoïdale possible sans affecter le fonctionnement du DAC (Fréquence d'échantillonnage maximale de 1 MHz) et du programme (Traitement des données). Les échantillons sont alors les suivants :

$$s_n = 512.[1 + sin(2\pi\frac{n}{32})] \tag{1}$$

Soit, s_n = {512 612 708 796 874 938 985 1014 1023 1014 985 938 874 796 708 612 512 412 316 227 150 86 39 10 0 10 39 86 150 224 316 412}.

Une solution pour améliorer la qualité sonore est d'augmenter le nombre d'échantillons, passant ainsi dans mon cas à 64 échantillons par période.

4.2 Signal monophonique

La génération du signal se fait donc en modifiant l'amplitude de s_n (qui est en pleine échelle) et en modifiant la fréquence d'échantillonnage pour une fréquence d'horloge de 15 MHz :

```
1  // Calculate the new DAC counter reload value, and load it
2  LPC_DAC0->CNTVAL = (15000000)/(SamplesPerCycle * Freq) - 1;
```

Modification_frequence.c

```
1  // DAC interrupt service routine
2  void DAC0_IRQHandler(void) {
3      static uint32_t windex = 0;
4      LPC_DAC0->CR = ((waveform[windex++]*vel)>>8)<<6;
5      if (windex == SamplesPerCycle) {
6          windex = 0;
7      }
8  }
```

DAC_ISR.c

Avec l'algorithme suivant :

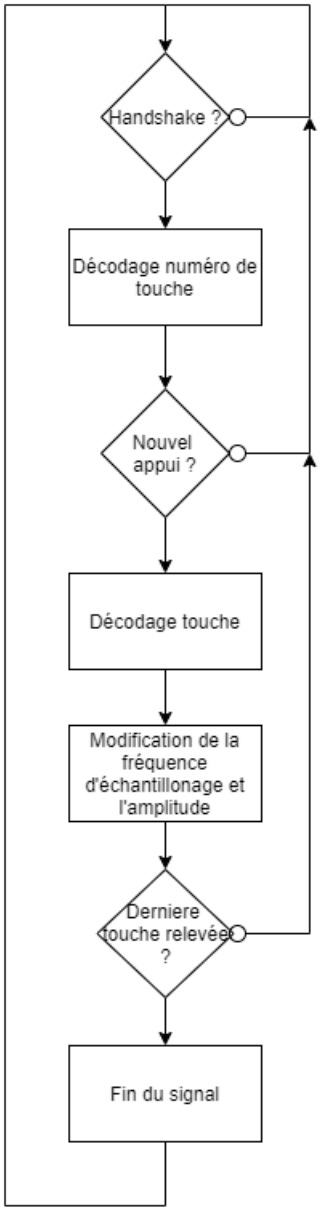


Figure 5: Génération d'un signal monophonique

Le choix de génération sur la dernière touche appuyée ne permet pas la mémoire des touches précédentes. Ainsi, si l'on appuis sur trois touches consécutivement (C4, D4, E4), on entendre au final E4 et si l'on lâche la touche E4, on arrête le son.

4.3 Signal polyphonique par approximation

Ne disposant pas de librairie "maths.h", l'addition de signaux de fréquences différentes mène à l'approximation des fonctions trigonométrique. On propose alors l'approche par décomposition en série de Taylor tel que :

$$sin(x_i) = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x_i^{2n+1} \tag{2}$$

Avec $x_i = \frac{2\pi f_{note} . i}{f_{echantillonnage}}$
En essayant d'approximer grossièrement la fonction sinus, en réduisant le calcul au maximum, avec la décomposition suivante :

$$sin(x_i) = x_i - \frac{x_i^3}{6} \tag{3}$$

On obtient une approximation qui diverge un peu avant $\frac{\pi}{2}$:

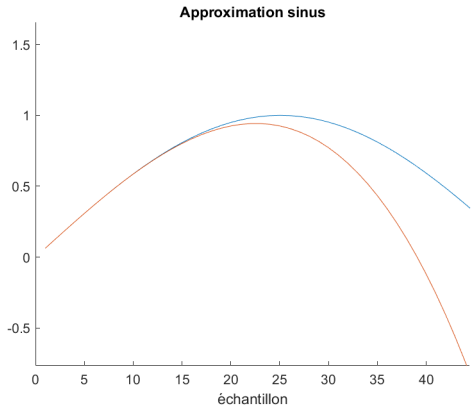


Figure 6: Approximation ordre 3

Ainsi, en effectuant une modification sur les intervalles $[\frac{\pi}{2},\pi[$, $[\pi,\frac{3\pi}{4}[$ et $[\frac{3\pi}{4},0[$ l'approximation, on obtient une première approximation :

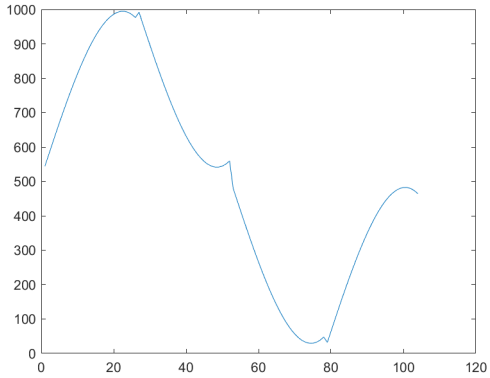


Figure 7: Approximation sinus avec $f_{signal=440Hz}$ et $f_{echantillonnage} = 44,1\ kHz$

Essai sur micro-contrôleur

L'essai ce fait avec un signal monophonique de référence tel que le La_3 (440 Hz). Malheureusement, le µC n'a pas assez de puissance de calcul pour échantillonner à hauteur d'une fréquence de $44,1\ kHz$. Ainsi, on essaye de réduire cette fréquence.
Le signal commence à devenir audible à partir d'une fréquence d'échantillonnage de $10\ kHz$ mais le temps de calcul est encore trop important et diminue ainsi la fréquence du signal en sortie. Enfin, avec une fréquence d'échantillonnage de $5\ kHz$ nous pouvons entendre une large gamme de notes mais de très mauvaise qualité

puisque la fréquence d'échantillonnage se trouve dans la bande audible et l'approximation implique certaines variations brusques qui vont se retrouver replier dans le spectre de fréquence des notes de piano.
Note : La fréquence de conversion n'était pas de 44,1 et 5 kHz mais beaucoup plus élevée à cause d'une inattention dans le programme (corrigé mais pas testé) qui ne valide plus la conclusion de l'essai sur µC.

5 Échantillonnage Piano YAMAHA

On souhaite améliorer la qualité audio en sortie du haut-parleur. Pour ce faire, l'idée est d'acquérir le son provenant d'un "vrai" piano. L'acquisition ce fait sur un piano numérique (Yamaha YDP-164) qui lui-même a un son échantillonné à partir d'un piano à queue, le Yamaha CFX.

5.1 Acquisition des données

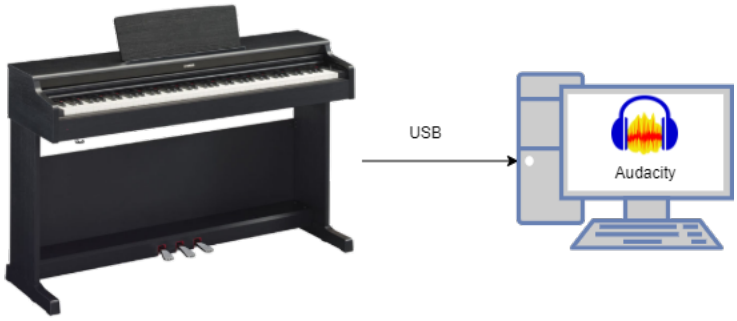


Figure 8: Acquisition des signaux d'un piano numérique

On commence par l'acquisition d'une octave à l'aide du logiciel `Audacity` en considérant le piano comme une entrée audio. On définit une fréquence d'échantillonnage qui sera également utilisée dans le µC pour la génération des signaux : 16 kHz. On observe ainsi à l'aide d'`Audacity` un exemple d'une période d'un Do4 :

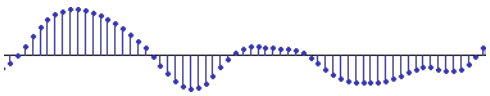


Figure 9: Échantillonnage à 16kHz d'une période d'un Do_4

`Audacity` permet également d'exporter une sélection de signal sous format texte que l'on normalise pour compenser les différences d'amplitudes lors de l'appui.

5.2 Normalisation

En utilisant les échantillons de l'octave acquis précédemment, on utilise le logiciel `Matlab` afin de traiter ces données de la manière suivante :

- Chargement des données de la note dans un vecteur
- Recherche du maximum en absolu du signal
- Multiplier le signal pour obtenir une amplitude de $\pm$ 400 (Pour la résolution de 10 bits du DAC)
- Transposer le signal en positif
- Exporter les données pour pouvoir les utiliser en C

Les données exportées sont en Annexe.

Que l'on peut vaguement représenter :

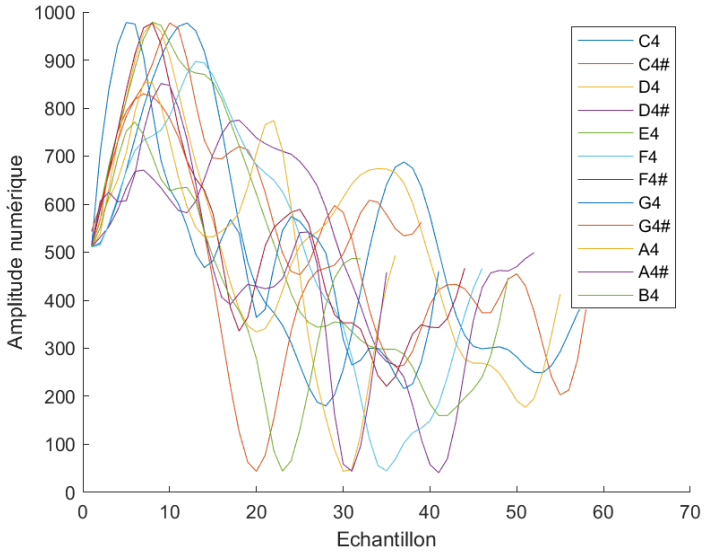


Figure 10: Normalisation des échantillons de l'octave 4

5.3 Monophonie et Polyphonie

Cette fois-ci, la génération du signal sonore n'est plus effectuée avec un échantillon de sinus pur et une fréquence de conversion du DAC variable mais avec plusieurs échantillons de longueurs différentes et une fréquence de conversion fixée à 16 kHz.
Le soucis avec cette méthode est que l'on ne peut pas avoir un nombre d'échantillons avec un coefficient de proportionnalité liant la fréquence de la note à la fréquence d'échantillonnage.

La polyphonie est générée par l'addition des échantillons entre-eux lors de l'appui avec un indice parcourant les échantillons indépendants. On ne travaille ainsi plus avec des scalaires mais avec des vecteurs que l'on va utiliser pour créer un scalaire à chaque période de conversion du DAC.

6 Conclusion

Au cours de ce mini-projet, un nouveau protocole de communication a été découvert, la norme MIDI. Elle est essentiellement utilisée pour la communication entre périphériques musicaux et appareil de gestion/transformation des données. L'objectif principal était de générer un signal sonore sur la carte UE441 possédant un LPC804 et un petit haut-parleur. Dans un premier temps, il était question d'un signal monophonique composé d'un seul signal sinusoïdale de fréquence variable. Cet objectif a été validé à l'aide d'échantillons d'un sinus que l'on sort sur le haut-parleur avec la fréquence correspondante. Dans un second temps, j'ai voulu essayer de générer un signal polyphonique et ainsi de gérer plusieurs appuis de touches simultanées. Malheureusement, cet objectif n'a pas pu être validé dû à un besoin en puissance de calcul trop important.

Bibliographie

http://doc-eea.varoqui.fr/UE441/docs/
https://fr.wikipedia.org/wiki/MIDI_Tuning_Standard
https://fr.wikipedia.org/wiki/Musical_Instrument_Digital_Interface
https://fr.wikipedia.org/wiki/Frequences_des_touches_du_piano
https://fr.wikipedia.org/wiki/Serie_de_Taylor

Annexe

Données générées avec `Matlab` après traitement :

```
NbEchantillon = {62,58,55,52,49,46,44,41,39,36,35,32}

C4 = {511,519,556,607,671,737,802,859,906,944,970,977,961,918,848,757,656,558,479,427,
395,374,348,311,265,220,188,180,203,256,332,419,506,582,641,677,688,675,636,576,503,430,
369,327,304,299,301,303,297,282,263,250,249,264,293,332,373,409,435,450,462,474}

C4d = {511,566,676,751,793,819,848,891,942,977,966,901,809,733,696,694,709,720,713,680,
625,559,499,460,453,476,521,569,597,583,528,453,381,323,282,262,265,294,343,391,422,432,
433,424,401,374,374,406,444,455,430,376,307,241,203,213,276,384}

D4 = {511,538,633,728,819,899,956,977,959,907,834,757,686,624,564,501,436,382,347,334,341,
370,420,474,513,531,542,559,585,614,642,661,671,674,673,666,644,602,545,484,424,363,309,
277,269,269,264,248,221,191,177,196,250,327,413}

D4d = {511,531,553,587,633,668,671,654,634,610,587,582,606,653,704,746,772,775,757,738,
726,717,710,704,689,667,638,592,532,478,435,388,336,292,272,265,240,181,111,58,41,70,149,
258,358,426,457,462,461,470,487,499}

E4 = {511,594,680,752,816,883,944,979,972,937,902,880,873,870,854,819,771,722,672,621,569,
518,465,414,376,354,344,346,355,352,335,317,305,299,298,298,288,261,223,184,160,160,177,
197,215,240,285,356,447}

F4 = {511,515,559,614,669,711,733,742,754,783,828,872,897,894,867,829,786,744,707,682,665,
650,626,588,534,477,431,402,381,345,281,197,113,56,45,69,103,124,134,149,184,240,306,370,
425,466}

F4d = {511,588,667,752,837,912,967,977,933,849,758,690,653,629,580,485,385,336,364,442,
513,551,568,583,589,560,493,419,370,353,353,340,295,243,221,241,286,330,349,344,343,362,
406,467}

G4 = {511,708,839,931,978,975,907,795,691,632,601,554,496,468,482,529,568,540,444,364,
381,469,545,574,564,541,528,498,418,319,265,275,300,300,279,244,216,226,272,348,460}

G4d = {543,598,663,729,783,818,830,824,807,779,740,689,623,540,440,330,221,127,63,44,77,
153,248,336,401,440,461,467,473,496,537,582,608,604,578,549,534,538,562}

A4 = {511,570,610,650,707,792,852,853,805,732,657,595,552,532,532,544,558,583,636,707,765,
774,712,594,453,323,226,155,89,44,48,119,232,343,431,493}

A4d = {511,606,625,605,607,663,752,818,851,847,801,735,644,545,465,406,391,414,433,429,
424,428,445,491,541,542,475,333,163,59,44,96,197,329,458}

B4 = {511,551,619,695,752,771,744,700,652,628,633,635,606,542,490,446,406,378,339,278,186,
87,44,66,130,214,307,390,446,476,487,486}
```

piano.txt