|  |
| --- |
| Heig-vd |
| VIRTUOSO |
| Travail de bachelor |

|  |
| --- |
| Etudiant : Quentin Salomon  Professeur : François Birling  09/04/2016 |



Table des matières

[2 Enoncé 2](#_Toc454143194)

[3 Clause de confidentialité 2](#_Toc454143195)

[4 Résumé 2](#_Toc454143196)

[5 Introduction 2](#_Toc454143197)

[6 Pré-étude 2](#_Toc454143198)

[6.1 Cahier des charges 2](#_Toc454143199)

[6.1.1 Interface utilisateurs 2](#_Toc454143200)

[6.1.2 Programme de traitement des fichiers midi 5](#_Toc454143201)

[6.2 Comparatif des plateformes 6](#_Toc454143202)

[6.3 Evaluation des risques 7](#_Toc454143203)

[6.3.1 I2C 7](#_Toc454143204)

[6.3.2 Midi 7](#_Toc454143205)

[6.3.3 Performance iot 8](#_Toc454143206)

[6.3.4 Woopsa 8](#_Toc454143207)

[6.3.5 Concept 8](#_Toc454143208)

[6.4 Choix du matériel 8](#_Toc454143209)

[6.5 UML 9](#_Toc454143210)

[7 Projet 9](#_Toc454143211)

[7.1 Protocole de communication 9](#_Toc454143212)

[7.1.1 Composition des messages 9](#_Toc454143213)

[7.1.2 Types de messages 9](#_Toc454143214)

[7.2 Serial 9](#_Toc454143215)

[7.3 I2C 10](#_Toc454143216)

[7.4 Programme de conversion 11](#_Toc454143217)

[7.4.1 Conversion midi 🡺 XML 11](#_Toc454143218)

[7.4.2 Intensité des notes 11](#_Toc454143219)

[7.4.3 Tempos de la partition 11](#_Toc454143220)

[7.5 Programme Xylophone 11](#_Toc454143221)

[7.6 Visuel interface 11](#_Toc454143222)

[7.6.1 Prototype 11](#_Toc454143223)

[7.6.2 Possibilités offertes par concept 11](#_Toc454143224)

[7.6.3 Visuel final 11](#_Toc454143225)

[7.7 Serveur WEB 11](#_Toc454143226)

[8 Conclusion 11](#_Toc454143227)

[9 Liste des références 11](#_Toc454143228)

[10 Dictionnaire de symboles et abréviations 12](#_Toc454143229)

[11 Figures 12](#_Toc454143230)

[12 Journal de travail 12](#_Toc454143231)

[13 Annexes 12](#_Toc454143232)

# Enoncé

La HEIG-VD a déjà réalisé un robot Xylophoniste anthropomorphe, appelé Virtuoso. Du fait de sa structure, il peut jouer à une vitesse proche de l'humain.

Pour pouvoir jouer la musique à une rapidité absolument impressionnante, ce nouveau travail de diplôme a pour but d'automatiser un autre xylophone de 37 touches en l'équipant d'un actionneur électromécanique en face de chaque touche, et en pilotant l'exécution musicale par un logiciel embarqué d'une part, et un logiciel PC d'autre part permettant de préparer et transmettre les partitions.

En complément, la réalisation d'une interface utilisateur permettant de contrôler le système depuis un smartphone et de sélectionner le morceau à jouer peut s'inscrire dans ce travail de bachelor.

# Clause de confidentialité

-

# Résumé

?

# Introduction

Dans le cadre du travail de bachelor, nous avons eu le choix entre plusieurs projets. Mon choix s’est orienté vers le Virtuoso proposé par mr.Birling. Le Virtuoso a pour but de jouer des partitions qu’on lui transmet. Il dispose, pour ceci, d’un actuateur sur chacune des notes. Comme le projet terminé sera pour faire la « promotion » de la heig via les portes ouvertes, une interface web interactive implémentée pour que les personnes puissent proposer des partitions et suivre l’exécution du Virtuoso à l’aide de leur smartphone.



Figure 1 : Virutoso

# Pré-étude

## Cahier des charges

Pour la réalisation du projet, il a fallu définir un cahier des charges. Celui-ci est là pour savoir qu’est-ce qui devra être implémenté et qu’est-ce qu’il devrait possible d’avoir si le temps n’est pas manquant.

### Démarche

Afin de réaliser le cahier des charges, un diagramme des cas d’utilisations a tout d’abord été fait et modifié afin de correspondre aux attentes voulues. Pour bien séparé le Virtuoso du programme pour la conversion des fichiers, deux diagrammes distincts ont été réalisé. Ceux-ci sont montrés aux Figure 2 et Figure 3. Puis à l’aide de ces diagrammes, pour chaque cas d’utilisation des charges ont été définies donnant lieu au cahier des charges.

### Interface utilisateurs



Figure 2 : Diagramme cas d'utilisation Virtuoso

#### Interface web

**Superviser l’exécution**

|  |  |
| --- | --- |
| 10.10 | L’interface web doit permettre de suivre l’exécution sur l’écran en visualisant le titre du morceau joué ainsi que sa position dans la playlist. |
| 10.20 | L’interface web devrait permettre de voir un aperçu du Virtuoso avec chaque note et montrer quelles notes sont jouées en directe. |

**Ajouter partition à la playlist**

|  |  |
| --- | --- |
| 20.10 | L’interface web devrait permettre d’ajouter une partition à la playlist. |

**Demande d’ajout de partition inexistante**

|  |  |
| --- | --- |
| 30.10 | L’interface web devrait permettre de faire une demande d’ajout d’une partition qui n’est pas dans la liste des partitions enregistrées. Cette demande pourra être vue par l’animateur et être traitée ultérieurement. |

#### Interface NUC

**Superviser l’exécution**

|  |  |
| --- | --- |
| 10.10 | L’écran tactile doit permettre de suivre l’exécution sur l’écran en visualisant le titre du morceau joué ainsi que sa position dans la playlist. |
| 10.20 | L’écran tactile devrait permettre de voir un aperçu du Virtuoso avec chaque note et montrer quelles notes sont jouées en directe. |

**Effacer playlist**

|  |  |
| --- | --- |
| 20.10 | Il doit être possible de supprimer la playlist. |

**Pause Start/Stop partition**

|  |  |
| --- | --- |
| 30.10 | L’écran tactile doit permettre de mettre sur pause la partition qui est jouée. |
| 30.20 | L’écran tactile doit permettre de la lecture d’une partition. |
| 30.30 | Le start permettra de jouer, depuis le début, la partition de la playlist qui a été stoppée. |

**Test de toutes les notes**

|  |  |
| --- | --- |
| 40.10 | L’écran tactile doit permettre de lancer un programme test qui actionne chaque note à la suite. |

**Réglage du temps de frappe des notes**

|  |  |
| --- | --- |
| 50.10 | L’interface doit permettre de régler le temps de frappe de chaque note indépendamment. |

**Changer la vitesse de lecture**

|  |  |
| --- | --- |
| 60.10 | Il devrait être possible de changer la vitesse de lecture du Virtuoso. |

**Partition en boucle**

|  |  |
| --- | --- |
| 70.10 | L’interface devrait permettre de jouer une partition en boucle. |

**Visualiser xylophone**

|  |  |
| --- | --- |
| 80.10 | Il devrait être possible de voir chaque note du xylophone en directe (voir les notes qui s’active). |

### Programme de traitement des fichiers midi



Figure 3: diagramme cas d’utilisations programme de conversion des fichiers

**Charger un fichier midi**

|  |  |
| --- | --- |
| 10.10 | Le programme doit permettre de charger un fichier au format midi et d’en extraire les notes. |

**Visualisation la partition**

|  |  |
| --- | --- |
| 20.10 | Le programme doit pouvoir visualiser les notes, avec leur hauteur et leur tic, de la partition chargée. Les notes venant de différents canaux (instruments) doivent être distinguées. |

**Arranger la partition**

|  |  |
| --- | --- |
| 30.10 | Le programme permettra de transposer toute la partition d’un nombre de ton(s) choisi(s). |
| 30.11 | Il doit pouvoir transposer chaque instrument d’un nombre de ton(s) choisi(s). |
| 30.20 | Il pourra agir sur chaque note indépendamment pour modifier sa hauteur ou son tic. |
| 30.30 | Le programme devrait permettre de régler le tempo de la partition. |

**Rogner la partition**

|  |  |
| --- | --- |
| 40.10 | Il devrait être possible de sélectionner la zone que l’on va garder sur une partition. |

**Ecouter la partition**

|  |  |
| --- | --- |
| 50.10 | Le programme devrait pouvoir écouter la partition. |

**Sauver la partition**

|  |  |
| --- | --- |
| 60.10 | Le programme doit pouvoir sauver une partition sous forme XML. |

**Envoyer la partition au Virtuoso**

|  |  |
| --- | --- |
| 50.10 | Le programme devrait permettre de transmettre une partition au Virtuoso à l’aide de Woopsa. |

## Evaluation des risques

### Communication I2C

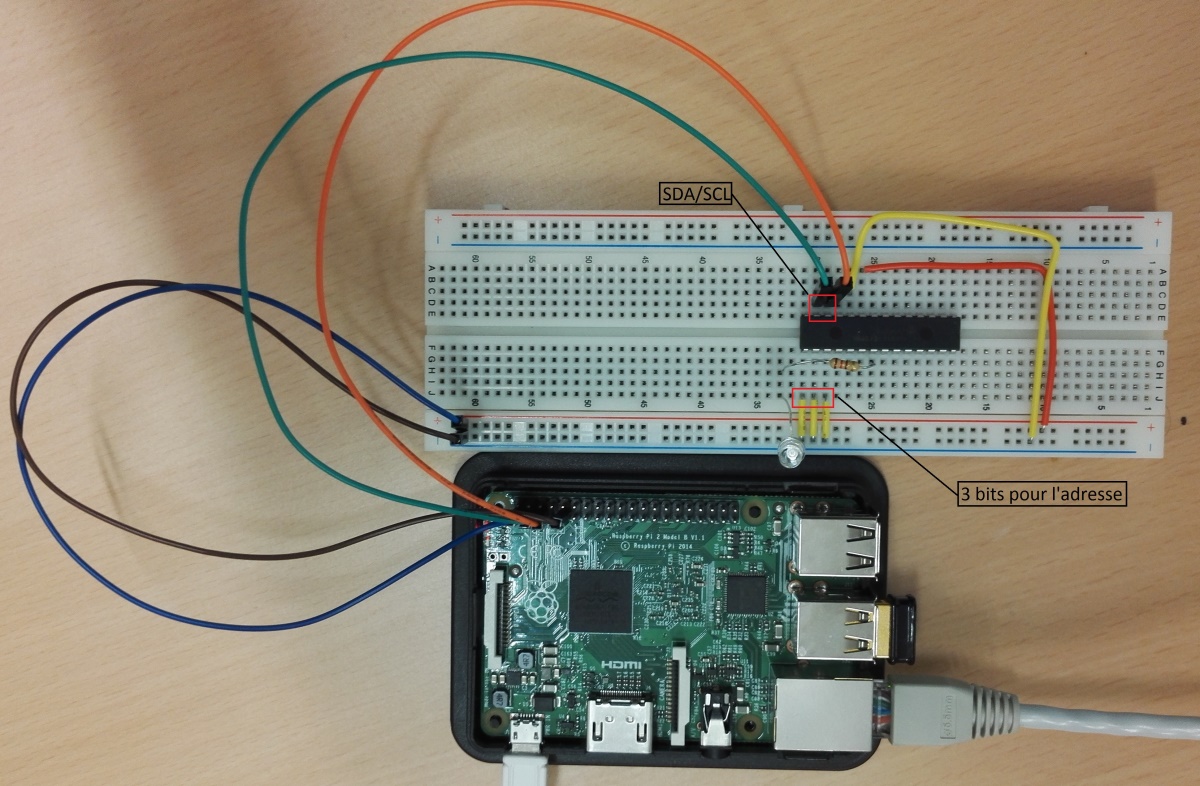
#### Problème

Comme la partie électronique du Virtuoso était implémentée avec des MCP23017 et prévu pour communiquer en I2C, le choix de la communication pour piloter les actuateurs s’est donc fait d’office.

#### Démarche

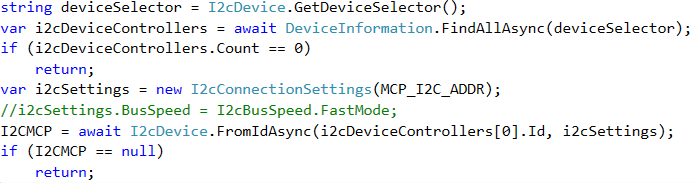
Ainsi pour tester l’I2C, un programme a été codé sur un Raspberry. Celui-ci fut créé juste pour tester l’I2C et savoir comment l’implémenté sur Windows IoT.

L’image suivante représente le câblage implémenté pour le programme de test. Avec cette configuration, nous avons l’adresse 0x20 pour le MCP décrite juste après.



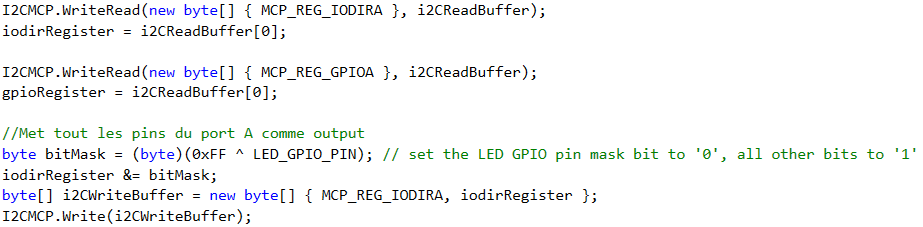
L’adressage pour le MCP23017 est sur 7 bits. Les 4 msb sont fixés et les 3 derniers peuvent être choisis. Dans notre cas, les 3 lsb valent 0, nous avons comme adresse 010 00002 = 0x20.

L’utilisation de l’I2C est plutôt simple car il suffit de trouver les bons registres à modifier. Pour le MCP23017, le registre des directions IO A est 0x00 et celui des valeurs IO A est 0x14. Ces registres nous permettent d’initialiser l’I2C puis de communiquer. Le programme initialise l’I2C :



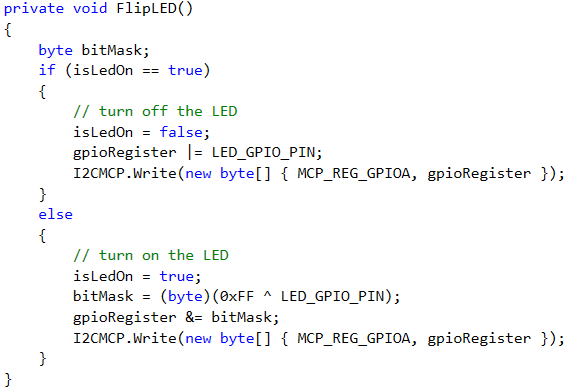
En donnant l’adresse du MCP (« MCP\_I2C\_ADDR » qui est 0x20), on obtient une variable « I2cDevice » qui nous permet d’écrire au MCP.

Puis le programme initialise les registres du MCP :



Avec les registres du MCP MCP\_REG\_IODIRA = 0x00 et MCP\_REG\_GPIOA = 0x14.

Et ensuite la communication est simple, on donne l’adresse et la valeur du registre que l’on veut écrire :



### Type de fichier convertit pour le Virtuoso : Midi

#### Problème

Pour « alimenter » le Virtuoso en musique, il faut trouver un type de fichier à convertir. Dans l’idéal, il faut un type qui aurait une grande « bibliothèque » de musique. De plus, on a besoin de fichiers qui sont sous forme de partition pour récupérer facilement les notes (et d’autre information).

#### Démarche

Après quelques recherches et propositions, le midi a été choisi pour les raisons suivantes :

Pour les fichiers son, le midi allait parfaitement étant donné qu’il permet d’obtenir chaque note jouée et de les séparées par instrument (les channels). De plus, il y a énormément de fichier midi téléchargeable sur internet, de quoi couvrir tous les goûts.

Pour traiter les fichiers midi en C#, une librairie a été trouvée. Elle se nomme miditoolkit et à l’avantage d’être libre d’utilisation. Elle dispose de toute les fonctionnalités nécessaires pour lire les fichiers midi.

Un des points faibles est qu’il n’y a que très peu d’exemples disponible pour son utilisation. Par conséquent, cela a demandé plus d’investissement en temps pour son implémentation.

Après avoir trouvé cette librairie, en voyant le peu d’exemple disponible et testant rapidement, des recherches rapides ont été faite pour regarder s’il était possible de faire une librairie maison rapidement. Il en est venu à la conclusion qu’il serait plus simple de rester sur la librairie existante.

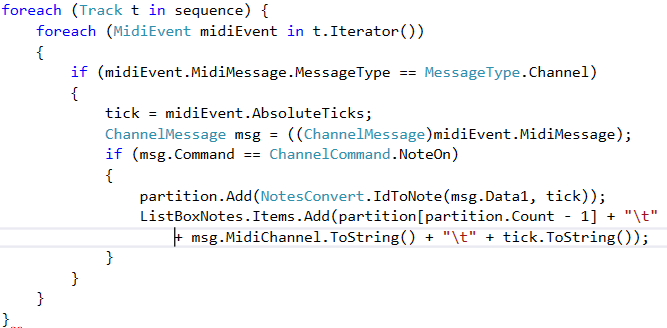
#### Test de la librairie miditoolkit

Afin de s’assurer que la librairie fonctionne bien et qu’elle offre les fonctionnalités nécessaires, un programme test a été réalisé.

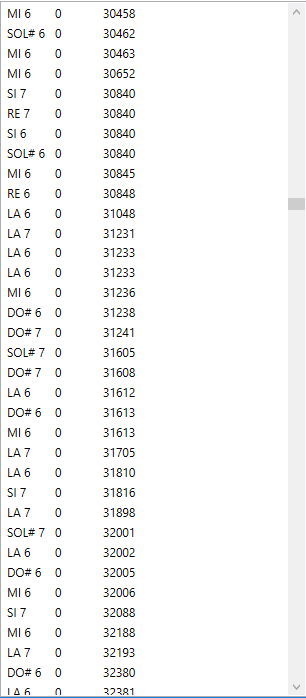
Le programme commence par charger le fichier midi avec un chemin en brut dans le code :



Puis il parcourt le fichier et charge les données dans une ListBox :



Le code ci-dessus explore chaque note de la partition en enregistrant la note dans l’instance « partition » et l’affiche à l’aide de la ListBox. Il affiche aussi le canal pour chaque note ainsi que le tick.



Pour l’affichage, le programme implémente une interface graphique très rudimentaire permettant de visualiser les résultats avec 3 colonnes (la note, le canal et le tick).

### Windows IoT

#### Qu’est-ce que c’est ?

Windows IoT est une version de Windows simplifiée pour les microcontrôleurs comme le Raspberry pi. Il a l’avantage de pouvoir développer à l’aide de Visual studio sur un pc et de compiler facilement sur le Raspberry. De plus, pour son utilisation, toutes les démarches sont décrites sur le site de Windows (il suffit de chercher un tout petit peu).

#### Performances de Windows IoT

##### Problème

Pour savoir si les performances de Windows IoT seraient concluantes pour le pilotage du xylophone, des tests ont été effectué.

##### Démarche

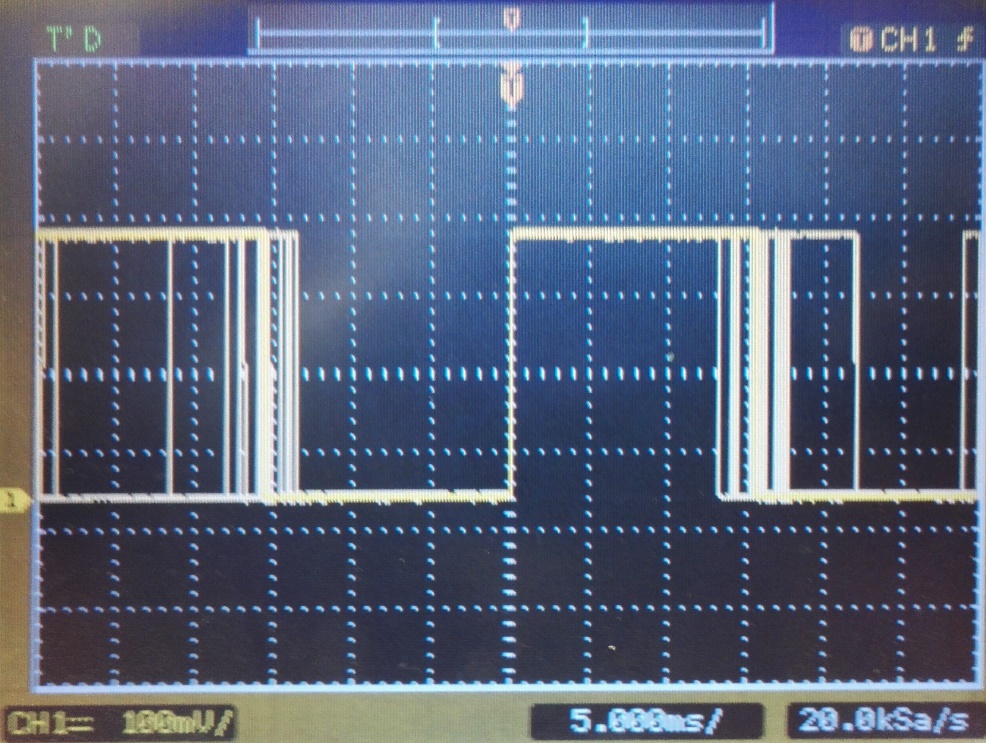
Afin de pouvoir piloter le Vituoso comme il faut, nous avons besoin d’un temps de réponse précis à la ms. Par conséquence, des tests ont été réalisé avec différentes manière.

Les tests sont des clignotement d’une sortie du Raspberry pi qui ont été mesuré à l’oscilloscope avec une rémanence infinie pour observer la gigue.

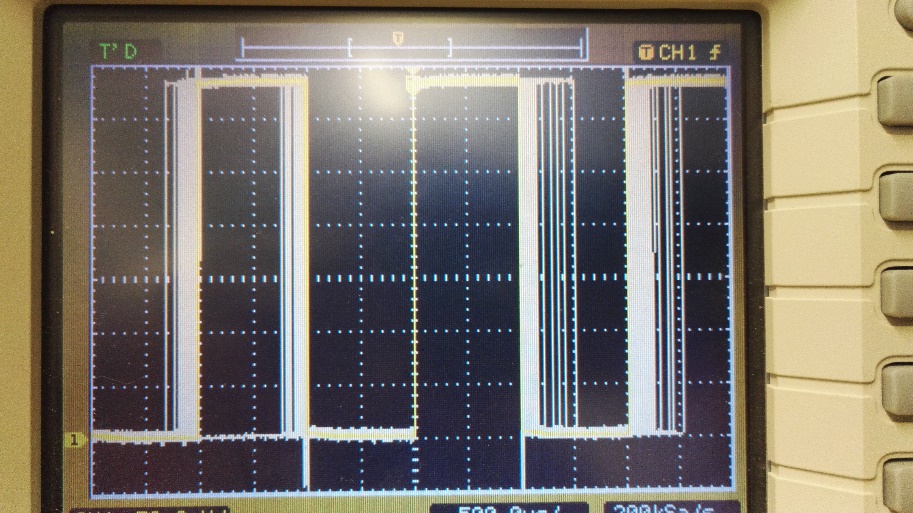
Le premier test est avec une application normal sans thread :



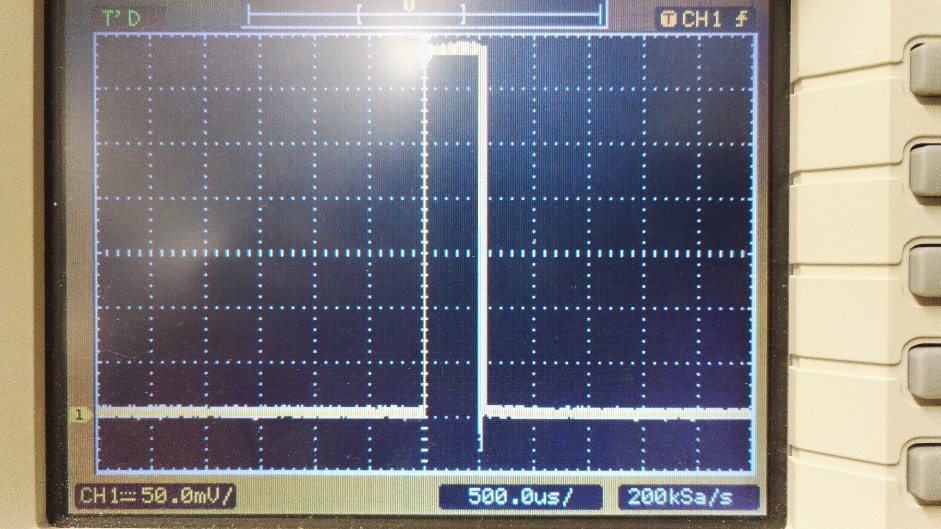
Le deuxième est avec un thread qui tourne avec des attentes passives :



Et le troisième test est avec un thread qui tourne à 100% avec un attente active :



Un autre test a été réalisé avec un application qui tourne en arrière-plan aussi avec un thread qui fonctionne à 100% :



Après ces tests, il a été constaté les faits suivants :

En attente active, Windows IoT permettrait de piloter le Virtuoso avec des temps de l’ordre de la milliseconde avec une gigue de 500 microsecondes. Malheureusement l’attente active n’est l’objectif souhaité car il fait tourner un thread à 100%.

Pour ce qui est de l’attente passive, Windows IoT ne tient pas la route car il offre une précision de l’ordre des 15ms.

### Woopsa & html

#### Qu’est-ce que c’est ?

Woopsa est un protocole de communication pour le web qui est open source. Il fonctionne sur le principe de l’orienté objet ce qui le rend pratique pour des programmes codés en objet.

#### Problème

Woopsa fait partit des exigences pour le développement du Virutoso. Car le Virtuoso doit émettre un réseau wifi et gérer la communication à l’aide de Woopsa. Afin d’utiliser Woopsa, des connaissances en html sont aussi obligatoires.

#### Démarche

Dans le but de découvrir un minimum Woopsa et l’html, car ils étaient jusqu’à maintenant inconnus, nous avons généré un programme impliquant un serveur Woopsa et implémentant un page web grâce à l’html.

Après avoir effectué un petit programme de test, il s’est avéré que Woopsa est assez simple à utiliser. Notamment grâce à son côté orienté objet. Mis à part pour les listes dynamiques. Malgré cela, le programme de test à prix un bon bout de temps car il m’a fallu apprendre toute la partie html/JavaScript.

En soit le code C# est très simple pour l’application de test car Woopsa gère quasiment tout mais par contre le JavaScript est surement un peu fouillis du faite qu’il n’est pas encore maîtrisé suffisamment pour obtenir un code propre et efficace.

### Concept HMI

#### Qu’est-ce que c’est ?

Concept HMI est une librairie pour le C# permettant de faciliter et d’accélérer le codage d’un programme grâce aux fonctionnalités qu’il implémente.

#### Problème

La question s’est posé de savoir s’il serait utile d’utiliser ou non concept pour les applications du Virtuoso et de la conversion des fichiers midi.

#### Démarche

Pour expérimenté concept HMI, un rendez-vous chez Objectis, qui est la boîte développant concept, a été pris. Objectis nous a fourni des tutoriels pour l’utilisation de concept.

Les premiers points du tutoriel ont été réalisé ainsi que ceux ciblant plus les objectifs du Virtuoso. Notamment la partie XML.

À la fin des tests, la décision fut prise d’utiliser concept surtout pour la partie XML gérant la sauvegarde et les chargements automatiquement.

Par contre, pour la suite du projet, il reste à voir comment changer les styles de concept pour correspondre aux attentes de l’application.

## Comparatif des plateformes

### Problème

Après une évaluation des risques, il s’est avéré que l’idée de base qui était d’utiliser Windows IoT ne correspondait pas aux objectifs du Virutoso. Il a donc fallu procéder à un choix d’une nouvelle plateforme.

### Démarche

Afin de résoudre le problème de la plateforme IoT, un comparatif a été accompli. Il compare chaque possibilité de pilotage (ex : Linux avec mono) avec les éléments requis pour le bon déroulement du projet. Et permet de faire un choix rapide et efficace.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Linux (mono)** | **Windows IoT** | **Windows IoT + Arduino** | **NUC + Arduino** |
| **Interface graphique** | - En html | + En C#, langage maîtrisé | + En C#, langage maîtrisé | + En C#, langage maîtrisé |
| **Lecture/écriture des fichiers** | + Fonctionne de base | - Test exemple de msdn ne fonctionne pas | - Test exemple de msdn ne fonctionne pas | + Fonctionne sur Windows |
| **Temps de réponse** | + suffisant pour le Virtuoso | - Pour obtenir un temps de réponse satisfaisant, il faut faire tourner un thread à 100% | + Gérer par l’Arduino, suffisant | + Gérer par l’Arduino, suffisant |
| **I2C** | + Implémenter directement de mono au xylophone | + Implémenter directement du Raspberry au xylophone | - Passe par l’intermédiaire de l’Arduino | - Passe par l’intermédiaire de l’Arduino |
| **Woopsa** | + Aucune modification à reprendre | - Modification à faire | - Modification à faire | + Aucune modification à reprendre |
| **Accès hardware** | - Inconnu | + Déjà tester et fonctionne parfaitement | - Inconnu sur la partie Arduino |  |
| **Debugging** |  |  |  |  |
| **Temps à investir** | - Apprendre à utiliser mono  - Revoir certain point déjà connu sur IoT | - Apporter des modifications à Woopsa | - Apporter des modifications à Woopsa  - Apprendre à utiliser Arduino | - Apprendre à utiliser Arduino  + Interface sur C#, utilisation de concept possible |

Suite au tableau ci-dessus, le choix du NUC de Intel avec un Arduino est le choix optimal. L’autre choix qui pourrait être pris, serait le Raspberry avec mono mais celui-ci n’offre pas la fonctionnalité importante du debugging. Par contre, pour la partie I2C, il n’y pas besoin de traitement particulier pour déléguer la communication avec le xylophone.

Un inconvénient du NUC est le prix qui est facilement le triple des autres.

Avec le NUC, comme il utilise Windows, il sera possible de mettre le programme pour les fichiers midi directement dessus et par conséquent, pouvoir ajouter dans partition sans un ordinateur externe. Tout sera donc regroupé dans une machine (ordinateur).

En plus de cela, il sera possible d’utiliser concept HMI pour la partie Virtuoso.

## Choix du matériel

### Problème

À la fin du comparatif des plateformes, après avoir fait le choix de partir avec un NUC combiné avec un Arduino, il a fallu faire un choix pour le matériel.

### Démarche

Le choix pour l’Arduino a été vite fait, car nous avons repris celui déjà disponible avec le Virtuoso. (Un Arduino était fourni de base avec le Virtuoso pour le travail de bachelor)

Du côté du NUC, après des recherches, le NUC5PGYH fut le meilleur choix pour les raisons suivantes :

* Un disque dur ssd 32Go implémenté de base
* Windows 10 déjà installé
* 2Go ram DDR3 mis de base
* Un prix pas trop élevé (260.-)
* Disponible et en stock sur digitec.ch



## UML

Le diagramme UML décrit ici, est le diagramme obtenu à la fin de la pré-étude. Ce diagramme m’a servi, d’une part, comme base pour le planning du projet, car il décrit dans les grosses lignes qu’est-ce que devras implémenter le projet. Et d’autre part, pour structurer le code des applications dès le début.

L’UML complet contient les UserControls ainsi que toutes les classes « prédites » pour les codes du projet. Il en découle donc plusieurs diagrammes.

### Programme Virtuoso

Pour la partie « Xylophone » : le diagramme HMI donne :



Ces différent UserControls Sont les vues offertes par l’application. Les vues « Home » et « About » n’utilise aucune autre classe.

Par contre, les vues « EditPlaylist », « Supervision » et « Settings » utilisent d’autres classes comme montrer ci-dessous.



La partie « EditPlaylist » utilise la playlist pour accéder à ces propriétés et les modifiées. Pouvant ainsi permettre à l’utilisateur de modifié la playlist du xylophone par le bief du UserControl.

La partie « Supervision » utilise le séquencer offrant à l’utilisateur les fonctions play/pause, stop, partition précédente ou suivante.

Et enfin, la partie « Settings » est là pour les réglages du Xylobot comme par exemple le temps frappe de chaque note. Permettant ainsi l’ajustement de l’intensité.

Après cela, nous avons les classes que composera le programme Virtuoso :



Je ne vais pas décrire toutes les classes, mais en bref nous avons le Xylobot, qui représente le xylophone avec ces 37 notes et actuateurs, qui contient une gestion de la communication avec le XyloCommunication. Le Xylobot offre la possibilité de test toutes les notes pour vérifier qu’il n’y ait rien de cassé.

Ensuite vient la Playlist, qui naturellement contient plusieurs Partitions (partition faîte pour le xylophone) et chaque une de ces partitions ont des Notes.

Et pour finir, le Sequencer, qui gère un peu tout derrière pour la lecture des partitions.

Le catalogue est là pour la gestion de la recherche. Il liste toutes les partitions disponibles.

Il reste le diagramme pour le serveur web :



Avec le Server qui utilisent la Playlist, car les clients se connectant dessus pourront seulement faire des propositions de partitions à jouer et voir la playlist en cours. Il inclut aussi le Sequencer, car il offrira la possibilité de voir l’avancement de la partition courante (comme décrit dans le cahier des charges).

### Programme midi🡺XML

Comme pour la partie Xylophone, nous avons le diagramme avec les UserControls :



La partie principale du programme est l’édition de partitions. Elle permet de modifier une partition midi pour ensuite l’enregistrer et l’envoyer au xylophone à l’aide du FileManagement.

On obtient les deux diagrammes suivant :



Comme le Virtuoso utilise des PartitionXylo, le FileManagement utilise aussi une PartitionXylo, car c’est lui qui les envoie au Virtuoso.

L’édition de partition se fait avec une PartitionMidi obtenue directement depuis un fichier midi.

La différence entre les PartitionXylo et PartitionMidi sont les canaux. Une PartitionMidi a plusieurs canaux représentant plusieurs instruments tandis que la PartitionXylo n’a pas de canal étant donné qu’elle ne compose qu’un instrument.



# Projet

## Protocole de communication

Afin de pouvoir communiquer entre l’Arduino et le NUC, un protocole de communication USB était nécessaire.

### Composition des messages

Message envoyé par le NUC :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte de Start | Numéro de message | Type | Taille des données | Données |
| 1 byte | 1 byte | 1 byte | 2 bytes | X bytes |

Message envoyé par l’Arduino:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Byte de Start | Numéro de message | Type | Taille disponible dans le buffer | Tick courant (Tick pour les notes) |
| 1 byte | 1 byte | 1 byte | 1 byte | 4 bytes |

### Types de messages

|  |  |  |  |
| --- | --- | --- | --- |
| **NUC** | | **Arduino** | |
| Tempo | 0000 00002 | Ok | 0000 00002 |
| Notes | 0000 00012 | Too many data | 0000 00012 |
| Start | 0000 00102 | Erreur Start byte | 0000 00102 |
| Stop | 0000 00112 | Erreur Type | 0000 00112 |
| Pause | 0000 01002 |  |  |

Le NUC envoie un paquet à l’Arduino avec un numéro de message. L’Arduino traite le message et renvoie au NUC un message avec le même numéro et ainsi de suite.

Si le NUC ne reçoit pas le message de réponse après un delay, il renvoie une fois les données et après cela, s’il n’a toujours pas de réponse, il génère une erreur.

Quand l’Arduino reçoit un message qu’il a déjà reçu, il ne le traite pas mais renvoie quand même une réponse. Si la taille des données qu’il reçoit est trop grande, l’Arduino renvoie « Too many data ».

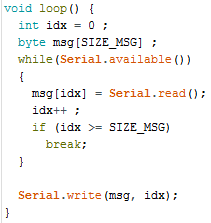
Si le NUC ne reçoit pas de réponse avant un certain temps, il génère une erreur.

La vitesse de transmission des données n’est pas modifiable. Elle est définie dans le code.

## Codage de la communication USB

Pour ce qui est de la communication serial avec le port USB, je suis partit avec le protocole décrit précédemment.

Un premier code a été fait, basé sur une faute, car nous avons testé la communication en envoyant des données à l'Arduino et en renvoyant les mêmes données depuis l'Arduino. Les données envoyées et reçues concordait bien mais plus tard en testant le code pour le protocole j'ai remarqué que l'Arduino traitait déjà des données sans les avoir toutes reçues. À cause de cela, en checkant s'il y avait des données dans le buffer du serial (fonction Arduino: Serial.available()) on les détectait mais elles n’étaient pas encore toutes arrivée. Ainsi dans l’exemple suivant, le programme recevait les bytes données et on les renvoyait un par un au lieu de paqué en un bloc d’une taille de SIZE\_MSG :



Il a fallu réadapter le code en prenant compte cet effet.

Un autre problème qui survenu et qu’il a fallu débugger est la vitesse de communication. Après quelque recherche rapide sur le site Arduino pour les vitesses disponibles pour l'Arduino méga et une vitesse a été choisie. Or il s'est avéré que cette vitesse de communication n'était enfaite pas adaptée au méga que je possède. En conséquence, quand on exécutait le code de test (envoie des données à l'Arduino et celui-ci renvoie les même) deux bytes en trop apparaissait au début des messages reçus dans l'ordinateur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| byte 0 | byte 1 | byte 2 | byte 3 | byte 4 |
| 255 | 0 | 2 | 128 | 0 |

Message envoyé :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| byte 0 | byte 1 | byte 2 | byte 3 | byte 4 |
| 171 | 171 | 255 | 0 | 2 |

Message reçu :

## I2C

La partie i2c m'a donné plus de fil à retordre que prévu. Ceci est dû à un mauvais choix de ma part car n'ayant presque jamais fait de microcontrôleurs, je suis partit sur le choix des interruptions. Et il s'est avéré qu'il n'est pas possible de communiquer en i2c à l'intérieur d'une interruption car il est bloquant et l'Arduino reste bloqué infiniment dans l'interruption.

La principale raison pour laquelle j'étais partit sur ce choix est que pour jouer les partitions, il intervient un tempo qui donne le rythme. Et je me suis laissé dire qu'avec une interruption timer cela serait idéal.

Mon choix final fut de partir sur la fonction millis(), proposé par l'Arduino, qui donne le nombre de milliseconde depuis le début du programme. A l’aide de ce choix, un premier code a vu le jour, utilisant la fonctionnalité delay(ms). Celui activait les notes les unes après les autres (en fonction du « tic » de la note dans la partition) les activaient, puis après un delay(ms), les désactivaient.

(pseudo code test I2C)

While(currentNote.Tick == currentTick)

{

PreparePush(currentNote) ;

currentNote.Next ();

}

ApplyPush() ;

Delay(9) ; //delay de 9ms

ReleasePush() ;

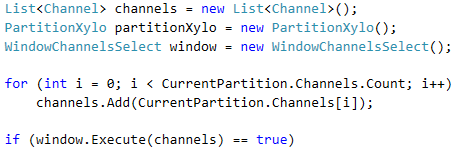
Le gros problème de cette méthode est qu’il n’est pas possible de jouer des notes avec un écart de temps plus petit que le temps de frappe de l’actuateur. Le code a donc été adapté de manière à jouer chaque note indépendamment les unes des autres. Permettant aussi de régler le temps de frappe de chaque note pour l’accordage. (Code final en annexe)

## Programme de conversion

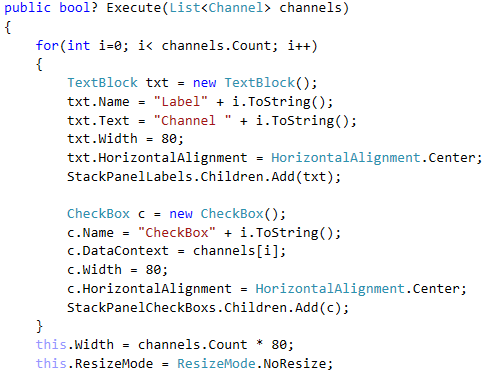
### Conversion midi 🡺 XML

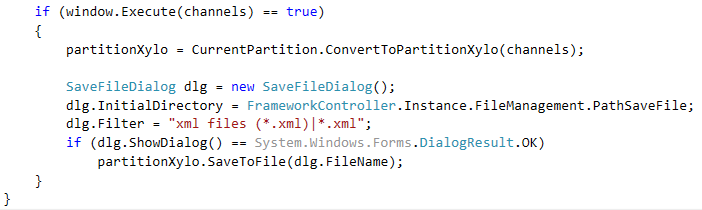
Pour la conversion, il faut que l’utilisateur choisissent les canaux qu’il veut garder. Il y a donc une fenêtre qui s’ouvre lors de la conversion, demandant à l’utilisateur de sélectionner les canaux désirés.

A l’appel de la fenêtre, on envoie une liste de tous les canaux disponibles de la partition midi.



Puis la fenêtre génère dynamiquement des check box pour la sélection des canaux. Comme l’affectation d’un content au check box ne s’affiche pas, la fenêtre crée aussi un label pour chaque canal.



Une fois les canaux validés, si l’utilisateur a confirmé son choix, on convertit la partition et on demande le nom pour la sauvegarde du fichier.

### Intensité des notes

### Tempos de la partition

## Programme Virtuoso

## Visuel des interfaces

### Prototypes

### Possibilités offertes par concept

### Visuel final

## Serveur WEB

# Conclusion

# Liste des références

# Dictionnaire de symboles et abréviations

(Facultatif)

# Figures

(Facultatif)

# Journal de travail

# Annexes