

Rapport de laboratoire

École supérieure

Électronique

DIPLOME
SLO2

Jeu Simon Inspiré

N° 2208

Réalisé par :

Ricardo Crespo

À l'attention de :

Professeur M. Bovey
Expert M. Olivier Guédat
Expert M. De Montmollin Simon

Dates :

Début du laboratoire : 8 août 2022
Fin du laboratoire : 12 septembre 2022

Table des matières

| | | |
|--------|--|----|
| 1. | Phase de design..... | 6 |
| 1.1. | Description du produit | 6 |
| 1.1. | Planning | 6 |
| 1.2. | Schéma bloc global..... | 7 |
| 1.3. | Choix effectués et dimensionnements | 8 |
| 1.3.1. | Alimentation | 8 |
| 1.3.2. | Microcontrôleur | 15 |
| 1.3.3. | Buttons avec LED RGB intégrés | 23 |
| 1.3.4. | Driver de LEDs..... | 24 |
| 1.3.5. | Interface utilisateur..... | 25 |
| 1.3.6. | Son | 27 |
| 1.4. | Conclusion phase de design | 30 |
| 2. | Hardware | 31 |
| 2.1. | PCB..... | 31 |
| 2.1.1. | Concept de design | 31 |
| 2.1.2. | Largeur des pistes..... | 32 |
| 2.1.3. | Placement des composants stratégique | 33 |
| 2.1.4. | Design PCB | 34 |
| 2.2. | Fabrication | 35 |
| 2.2.1. | Courbe de température | 35 |
| 2.2.2. | Montage du PCB..... | 37 |
| 2.2.3. | Mise en service | 37 |
| 2.3. | Boitier..... | 38 |
| 2.4. | Modifications | 42 |
| 2.4.1. | PCB | 42 |
| 2.4.2. | Boitier..... | 42 |
| 3. | Firmware | 43 |
| 3.1. | Boucle principale..... | 43 |
| 3.1.1. | Flowcharts..... | 43 |
| 3.1.2. | Initialisations..... | 44 |
| 3.2. | Boucle générale du Jeu et de paramétrage | 50 |
| 3.2.1. | Flowchart | 50 |
| 3.3. | Boucle d'exécution du Jeu du Simon..... | 51 |
| 3.3.1. | Flowchart | 51 |
| 3.4. | Détection des 7 butons et du PEC12..... | 52 |
| 3.5. | Driver de LEDs avec SPI1 | 52 |

| | | |
|--------|--|----|
| 3.5.1. | Latch | 52 |
| 3.5.2. | Initialisation | 53 |
| 3.5.3. | Commande LEDs..... | 54 |
| 3.6. | Notes de musique avec DAC12 et SPI2 | 55 |
| 3.7. | LCD avec I2C..... | 57 |
| 3.8. | Gameplay..... | 58 |
| 3.8.1. | Game Design | 58 |
| 3.8.2. | Points de Score et de Précision..... | 58 |
| 3.9. | Tableau des scores | 59 |
| 3.10. | Sauvegarde | 61 |
| 3.11. | Communication UART vers le PC via USB | 63 |
| 3.12. | Control de charge de l'accumulateur..... | 65 |
| 3.13. | Communication avec la carte microSD via SPI3 | 65 |
| 4. | Software..... | 66 |
| 4.1. | Flowcharts..... | 66 |
| 4.1.1. | Flowchart de la communication | 66 |
| 4.1.2. | Flowchart du traitement de données | 67 |
| 4.2. | Communication entre le Simon et le PC | 68 |
| 4.2.1. | Initialisation | 68 |
| 4.2.2. | Événement Timer1 | 70 |
| 4.2.3. | Événement réception de données du Firmware | 73 |
| 4.3. | Traitement des données du tableau des scores | 74 |
| 4.3.1. | Sauvegarde des données dans un fichier texte | 74 |
| 4.3.2. | Déchiffrage du tableau des scores | 76 |
| 4.4. | Graphique de progression | 79 |
| 4.4.1. | Paramétrage du graphique..... | 79 |
| 4.4.2. | Sélection des Players..... | 82 |
| 4.4.3. | Mise à jour des données du graphique..... | 83 |
| 4.4.4. | Exemples de vues | 84 |
| 5. | Test et mesures | 85 |
| 5.1. | Appareils de mesure | 85 |
| 5.2. | Alimentations..... | 85 |
| 5.3. | Timers | 85 |
| 5.4. | Debouncing butons poussoir | 86 |
| 5.5. | PEC12..... | 87 |
| 5.6. | UART2 | 88 |
| 5.7. | SPI1 Driver de LED | 90 |

| | | |
|--------|---|-----|
| 5.7.1. | Initialisation | 90 |
| 5.7.2. | Commande LEDs..... | 91 |
| 5.8. | SPI2 | 92 |
| 5.9. | I2C | 94 |
| 5.10. | Son et notes | 96 |
| 5.11. | Charge de l'accumulateur et consommation | 97 |
| 6. | État final et améliorations | 98 |
| 6.1. | Projet..... | 98 |
| 6.2. | Hardware | 98 |
| 6.3. | Firmware | 98 |
| 6.4. | Software..... | 99 |
| 6.5. | Boitier..... | 99 |
| 6.6. | Test et Mesures | 99 |
| 7. | Conclusion | 100 |
| 8. | Références..... | 102 |
| 9. | Lexic..... | 103 |
| 10. | Annexes..... | 104 |
| A. | Cahier des charges..... | 104 |
| B. | Liste des pièces et coûts..... | 104 |
| C. | Schéma électrique..... | 104 |
| D. | Listings du Firmware | 104 |
| E. | Listings du Software | 104 |
| F. | Planning du projet..... | 104 |
| G. | Journal de travail..... | 104 |
| H. | PV de toutes les séances..... | 104 |
| I. | Mode d'emploi | 104 |
| J. | Fichier de modification..... | 104 |
| K. | Mail de retard Euro Circuit | 104 |
| L. | GDD | 104 |
| M. | Toutes les mesures (uniquement format numérique)..... | 104 |

1. Phase de design

1.1. Description du produit

Le but de ce travail de diplôme est de concevoir un jeu inspiré du jeu électronique « Simon ©»; au lieu d'avoir quatre touches, comme sur la version officielle, l'interface utilisateur sera composée de 7 touches correspondant à une gamme complète de notes (Do – Ré – Mi – Fa – Sol – La – Si) ; ces touches seront accompagnées de LEDs RGB permettant de donner une couleur spécifique à chaque note.

Le but est de pouvoir reproduire des séquences de notes dans un laps de temps donné. Cela tout en respectant le tempo de la gamme actuelle qui est adapté en fonction du mode de difficulté choisi (Easy, Hard, Extreme).

Le système devra pouvoir être autonome (alimenter sous forme de batterie), soit pouvoir être connecter un bloc secteur (12V – 9V), ou via un port USB (5V). Si les batteries sont faibles, le bloc secteur ou le port USB devrait pouvoir effectuer la recharge de celles-ci.

Une interface utilisateur devra aussi réalisée en C# pour permettre d'avoir une base de données permettant de savoir le nombre de joueurs référencés, leur(s) score(s) dans les différents niveaux de difficulté. L'application permettra de récupérer les informations provenant du jeu.



Figure 1 Simon présente dans le marché grand public

1.1. Planning

| Jour du projet | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 39 | | | | |
|------------------------------|-----------|-----------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|------------|-------------|-------------|-------------|-------------|-------------|-------------|-----------|--|--|--|--|
| No semaine | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 33 | 33 | 33 | 33 | 33 | 33 | 34 | 34 | 34 | 34 | 34 | 34 | 34 | 35 | 35 | 35 | 35 | 35 | 35 | 36 | 36 | 36 | 36 | 36 | 37 | 37 | 37 | 37 | 37 | 37 | | | | | | | |
| Date | 8 aout 22 | 9 aout 22 | 10 aout 22 | 11 aout 22 | 12 aout 22 | 13 aout 22 | 14 aout 22 | 15 aout 22 | 16 aout 22 | 17 aout 22 | 18 aout 22 | 19 aout 22 | 20 aout 22 | 21 aout 22 | 22 aout 22 | 23 aout 22 | 24 aout 22 | 25 aout 22 | 26 aout 22 | 27 aout 22 | 28 aout 22 | 29 aout 22 | 30 aout 22 | 31 aout 22 | 1 sept. 22 | 2 sept. 22 | 3 sept. 22 | 4 sept. 22 | 5 sept. 22 | 6 sept. 22 | 7 sept. 22 | 8 sept. 22 | 9 sept. 22 | 10 sept. 22 | 11 sept. 22 | 12 sept. 22 | 13 sept. 22 | 14 sept. 22 | 15 sept. 22 | 7.09.2022 | | | | |
| Tâches | L | M | M | J | V | S | D | L | M | M | J | V | S | D | L | M | M | J | V | S | D | L | M | M | J | V | S | D | L | M | M | J | V | S | D | L | M | J | | | | | | |
| Phase de design | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Design Schéma | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Routage PCB | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Montage | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Software | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Firmware | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Mise en service et dépannage | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Boitier | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Test et Mesures | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Finitons + Corrections | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Rédaction du rapport et doc | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Figure 2 Planning du projet, ligne du dessus estimé, ligne du dessous réelle

1.2. Schéma bloc global

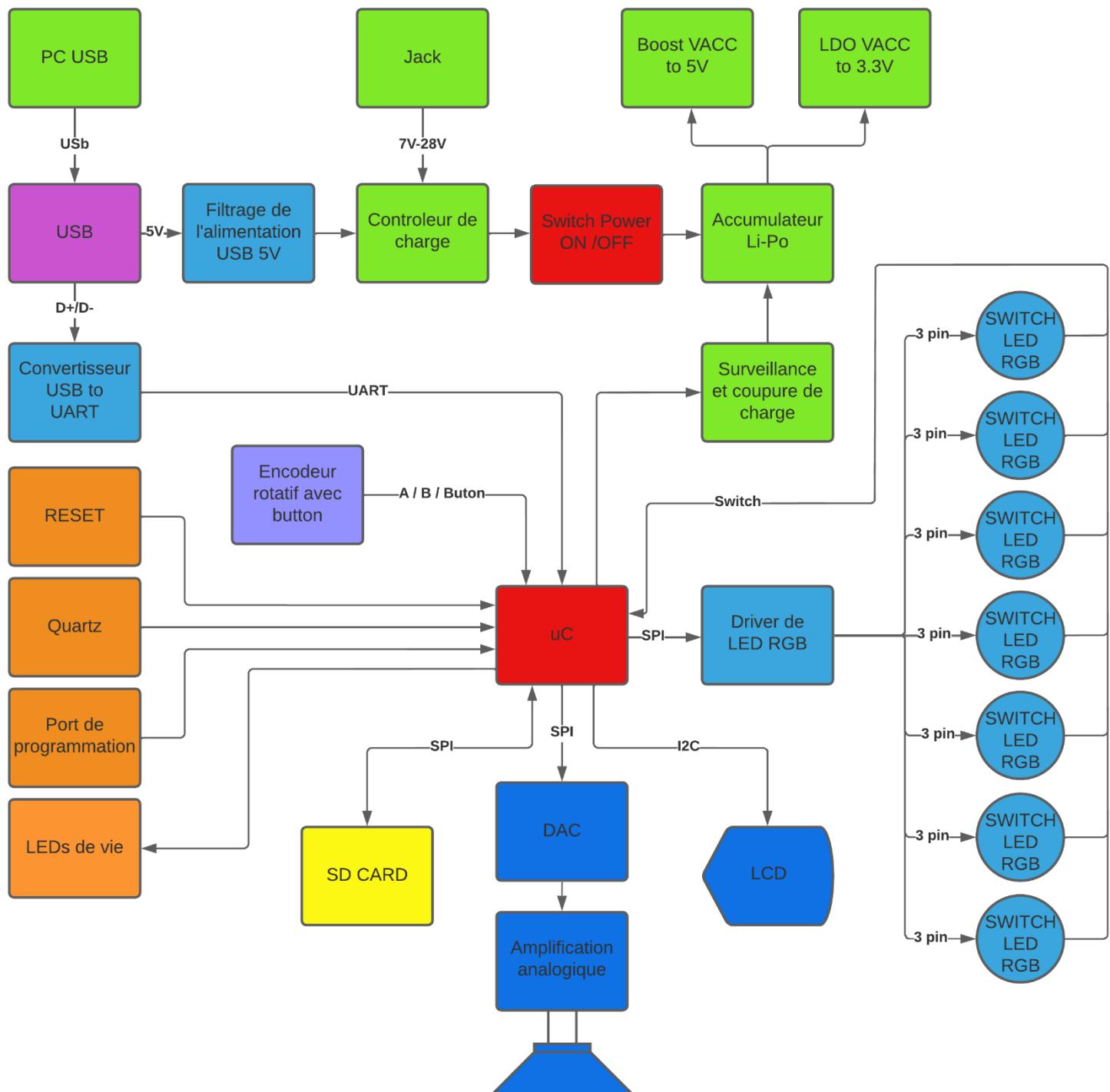


Figure 3 Flowchart du système global

1.3. Choix effectués et dimensionnements

1.3.1. Alimentation

1.3.1.1. USB

Afin de pouvoir alimenter le Simon, plusieurs possibilités ont été choisies. La première est de passer par le connecteur micro USB, afin de charger l'accumulateur et d'alimenter la carte.

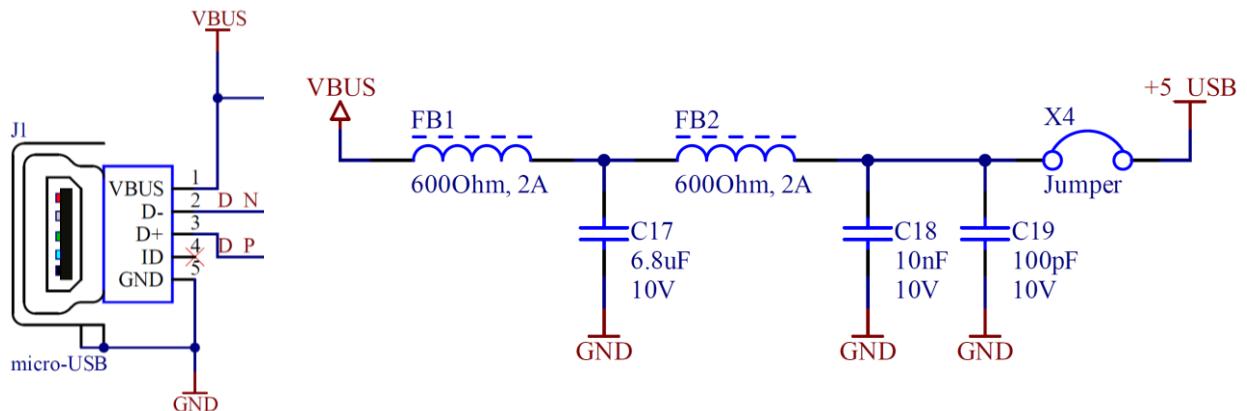


Figure 4 Connecteur micro USB

Figure 5 Filtrage du 5V USB et filtre passe-bas en T

On aura donc accès à du 5V via l'USB, et on pourra tirer jusqu'à 500mA via un port USB 2.0.

Pour filtrer les éventuelles perturbations IFTB, j'ai mis un filtre en T avec les ferrites et un condensateur juste après l'entrée par le connecteur micro USB. Ce filtre passe-bas permet de couper toutes les hautes fréquences potentielles de venir perturber l'alimentation. Le dimensionnement de ce filtre a été fait pour le précédent projet et stage effectué.

Puis des condensateurs de découplage ont été placés à la fin du filtre en T.

Un jumper a été placé à la fin de tout ça, afin que dans le cas où cette partie ne fonctionne pas correctement, on puisse connecter directement une alimentation de laboratoire à la place.

1.3.1.2. Jack

Puis la deuxième possibilité d'alimenter le circuit sera via un connecteur Jack, auquel on pourra alimenter la carte avec un block brick de 8V à 28V et de 1A. On pourra donc charger l'accumulateur plus rapidement via ce port-là.

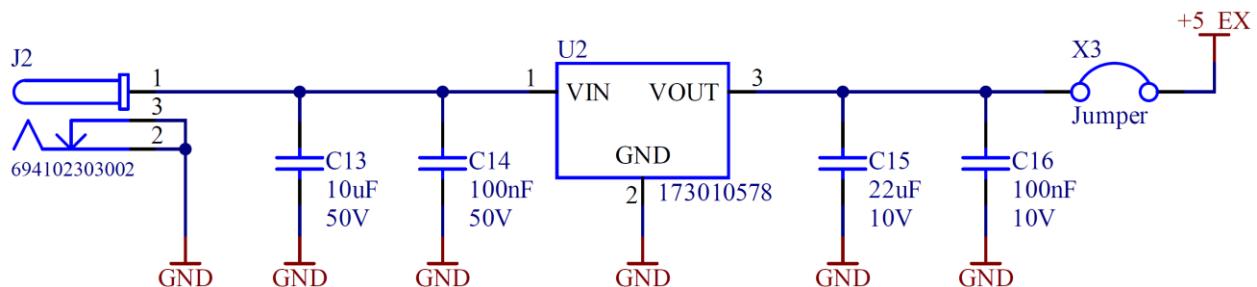


Figure 6 Entrée externe Jack pour bloc brick et convertisseur 8V-28V vers 5V

Afin de pouvoir avoir une aussi grande plage de tension en entrée en fonction du modèle de bloc brick utilisé, c'est grâce au convertisseur DC/DC qui nous assure du 5V en sortie, et qui accepte du 8V à 28V en entrée. Avec un rendement de 91% pour les 5V et 1A, on dissipera maximum 5W avec. Pour les valeurs de condensateur, elles ont été reprises des conseils du datasheet. Un jumper a également été placé pour la mise en service.

1.3.1.3. Contrôleur de charge

Afin de charger l'accumulateur, un contrôleur de charge est donc nécessaire. Le modèle a été repris d'un design déjà utilisé à l'ETML-ES.

Comme vu précédemment on a deux arrivées possibles pour assurer la charge, soit par l'USB ou soit par le Jack. Une configuration du contrôleur de charge (MCP73871) permet de définir si on est en mode USB, ou alors avec une alimentation externe plus puissante.

| | | | |
|---|-----|---|---|
| 3 | SEL | I | Input type selection (low for USB port, high for AC-DC adapter) |
|---|-----|---|---|

Figure 7 Pin 3 (SEL) contrôlant le mode choisi pour l'entrée, USB ou Jack (Datasheet MCP73871, p.17)

Un simple circuit logique a donc été fait afin de commander la pin 3 (SEL) qui y est dédiée.

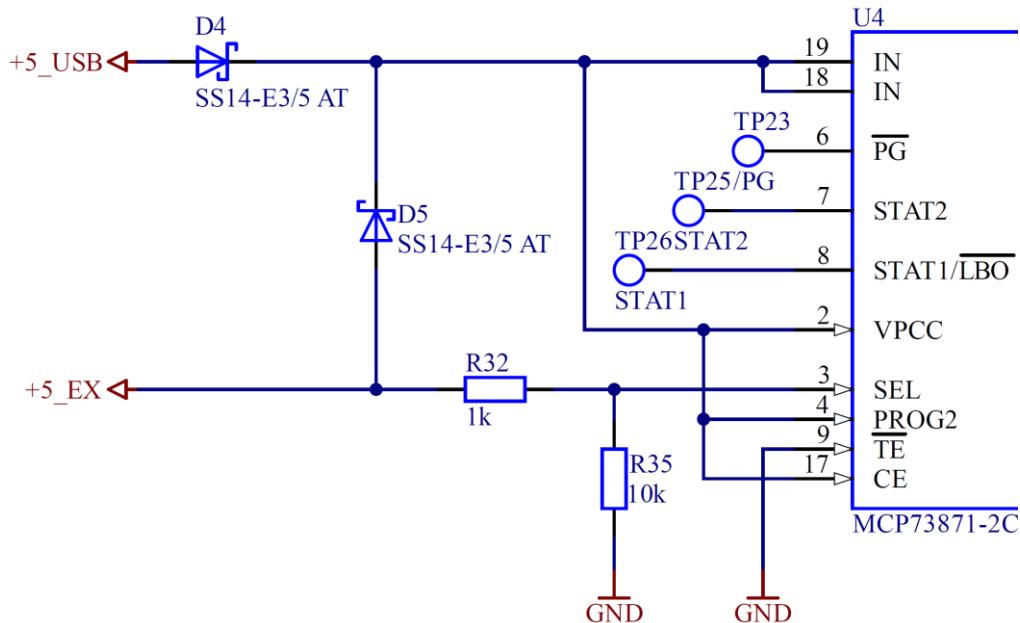


Figure 8 Schéma logique qui permet de configurer le mode en fonction de l'entrée

Ici on peut donc remarquer que dans le cas où l'on est connecté en USB, la diode D5 empêche les 5V de passer, donc la pull-down R35 met un niveau bas sur le pin 3 (SEL), qui correspond au mode USB.

En revanche, dans le cas où l'on est connecté avec le Jack, on aura un courant qui pourra circuler dans R32 et R35. Grâce à la loi d'ohm et le pont résistif effectué avec les deux résistances, on aura donc environ 4.54V en entrée de la pin 3 (SEL), qui représentera donc l'état haut avec le mode prévu pour une alimentation plus puissante.

Puis il a fallu calculer deux résistances pour fixer les courants de charge.

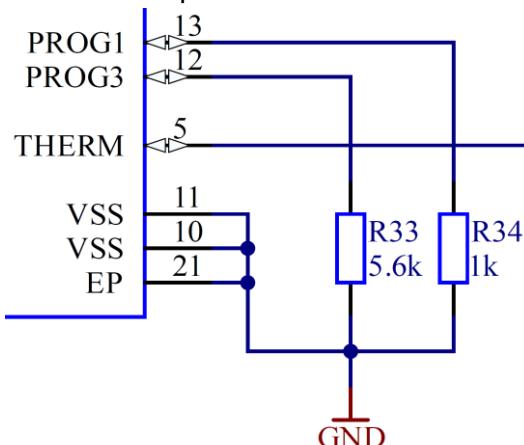


Figure 9 Résistances qui paramètrent le courant de charge des deux modes (USB et Jack)

Pour la première résistance qui viendra sur la pin 13 (Prog1), elle contrôlera le courant maximum de charge.

Afin de pouvoir la calculer, tout le nécessaire est fourni dans le datasheet.

EQUATION 4-1:

$$I_{REG} = \frac{1000V}{R_{PROG1}}$$

Where:

| | | |
|------------|---|-------------------------|
| R_{PROG} | = | kilo-ohms ($k\Omega$) |
| I_{REG} | = | milliampere (mA) |

Figure 10 Calcul de la résistance du courant max du Jack (Datasheet MCP73871, p.23)

Des informations concernant l'accumulateur vont être utilisées dans ces calculs, vous aurez plus de détails sur le modèle choisi dans le prochain point.

Afin de pouvoir calculer la résistance, il faut d'abord connaitre le courant maximum avec lequel on peut charger notre accumulateur.

| | |
|-----------------------|--------------------|
| Nominal Capacity | 1320 mAh |
| Max. Charging Current | 1.0 C CC – 1320 mA |

Figure 11 Capacité et courant de charge maximal de l'accumulateur Li-po (Datasheet ICP543759PMT, p.1)

C'est grâce à ces deux informations que l'on va pouvoir tout calculer.

Dans un premier temps il nous vaut sa capacité nominale, qui est de 1320mAh. Puis on peut calculer son courant de charge max.

$$I_{max} = 1 * C = 1 * 1.32 = 1.32A$$

Ici on retrouve la valeur que l'on nous a donné dans le datasheet, je l'ai tout simplement recalculée. Pour des raisons de simplification et de sécurité, j'ai arrondi à 1A le courant max.

Puis on peut calculer la résistance qui va ficher la charge via le Jack à 1A.

$$R_{Prog1} = \frac{1000V}{I_{max}} = \frac{1000}{1} = 1k\Omega$$

C'est donc une résistance de 1kΩ qui sera nécessaire pour la charge maximale à 1A.

Puis pour la deuxième résistance qui viendra sur la pin 12 (Prog3), elle contrôlera le courant de fin de charge.

EQUATION 4-2:

$$I_{TERMINATION} = \frac{1000V}{R_{PROG3}}$$

Where:

| | | |
|------------|---|-------------------------|
| R_{PROG} | = | kilo-ohms ($k\Omega$) |
| I_{REG} | = | milliampere (mA) |

The recommended PROG3 resistor values are between 5 kΩ and 100 kΩ.

Figure 12 Calcul de la résistance du courant de fin de charge (Datasheet MCP73871, p.23)

Cette fois-ci on ne pourra pas fournir un courant de 1A via l'USB 2.0, mais maximum 500mA.

(0.2C cut off 3.0 V at 20°C)
Figure 1 MT, p.1)

N'est au moins on se rapproche très fortement du courant de charge normal.

$$I_{morm} = 0.2 * C = 1 * 1.32 = 264mA$$

Mais on va donc calculer la résistance pour un courant de 264mA.

$$R_{Prog1} = \frac{1000V}{I_{norm}} = \frac{1000}{0.264} = 3.79k\Omega$$

C'est donc une résistance de 3.79kΩ qui serait nécessaire pour arrêter la charge à 264mA, mais on est en dessous des 5kΩ recommandés par le datasheet dans son équation 4-2. La première valeur de la série E12 alors été prise à la place, c'est-à-dire 5.6kΩ.

Ensuite on a plusieurs autres pins de commande, comme la pin 2 (VPCC) que l'on va mettre à l'état bas, car on n'en a pas besoin.

The VPCC feature functions identically for USB port or AC-DC adapter inputs. This feature can be disabled by connecting the VPCC to IN pin.

Figure 14 Description de la désactivation de VPCC (Datasheet MCP73871, p.24)

Puis il y a la pin 4 (Prog2), qui est à l'état haut pour un courant max de 500mA avec l'USB.

| | | | |
|---|-------|---|--|
| 4 | PROG2 | I | USB port input current limit selection when SEL = Low (Low = 100 mA, High = 500 mA) |
|---|-------|---|--|

Figure 15 Description de la désactivation de PROG2 (Datasheet MCP73871, p.17)

Ensuite on a la pin 9 (/TE), qui sera mise à l'état bas pour activer la sécurité.

| | | | |
|---|----|---|--|
| 9 | TE | I | Timer Enable; Enables Safety Timer when active-low |
|---|----|---|--|

Figure 16 Description de la désactivation de /TE (Datasheet MCP73871, p.17)

Enfin on a la pin 17 (CE), qui va simplement activer la charge de l'accumulateur.

| | | | |
|----|----|---|--|
| 17 | CE | I | Device Charge Enable; Enabled when CE = high |
|----|----|---|--|

Figure 17 Description de la désactivation de CE (Datasheet MCP73871, p.17)

Pour finir, la carte sera alimentée par la sortie du contrôleur de charge. C'est là que l'interrupteur général de la carte est placé.

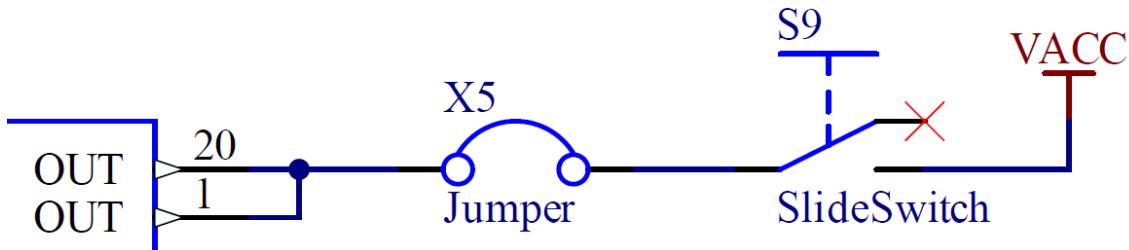


Figure 18 Sortie du contrôleur de charge avec l'interrupteur général de la carte

Placée de cette manière, la carte peut être chargée tout en étant éteinte. Lorsque l'on alimente la carte via l'USB ou le Jack, la charge de l'accumulateur se fait également en parallèle. On peut donc jouer au Simon et charger l'accumulateur au même temps.

1.3.1.4. Accumulateur LI-PO

Afin de choisir la technologie de l'accumulateur, mon choix c'est très vite restreint, car le contrôleur de charge utilisé supporte uniquement les accumulateurs Li-ion et Li-po. C'est après quelques recherches sur la sécurité de ces deux technologies, que mon choix s'est plutôt porté sur un accumulateur Li-po.

Puis il a fallu déterminer la taille de la capacité de l'accumulateur. Pour cela une estimation de la consommation a dû être faite.

| | nb | Unit [mA] | Total [mA] |
|---------------|----|--------------|---------------|
| ButtonLed | 7 | 20 | 140 |
| LCD | 1 | 5 | 5 |
| Driver de LED | 1 | 16 | 16 |
| uC + pertes | 1 | 60 | 60 |
| Haut-parleur | 1 | 200 | 200 |
| Total | | | 421 |

Figure 19 Estimation de la consommation du Simon

L'estimation est très compliquée à établir exactement, car les LEDs RGB consomment 20mA pour chaque couleur, mais les trois LEDs ne sont jamais allumées en semble au maximum. En plus de ça les 7 LEDs RGB ne sont presque jamais allumées toutes au même temps. J'ai donc défini la consommation d'une couleur tout le temps allumée au maximum sur les 7 boutons. De la même manière, le haut-parleur fonctionnera par courts instants pour les simples notes, donc une estimation très grossière a été faite.

À la fin j'obtiens 421mA de consommation, que j'ai arrondi à 500mA pour tous les autres IC non comptabilisés.

Dans le cahier des charges une autonomie de 2h a été demandée, donc je fais cette consommation fois deux pour avoir la capacité d'environ 1000mAh.

Suite aux recherches, c'est un accumulateur de 1320mAh qui a été choisi.

$$\text{Autonomie} = \frac{\text{Capacité}}{\text{Consommation}} = \frac{1320}{500} = 2.64$$

On aura donc une autonomie d'environ 2h38min.

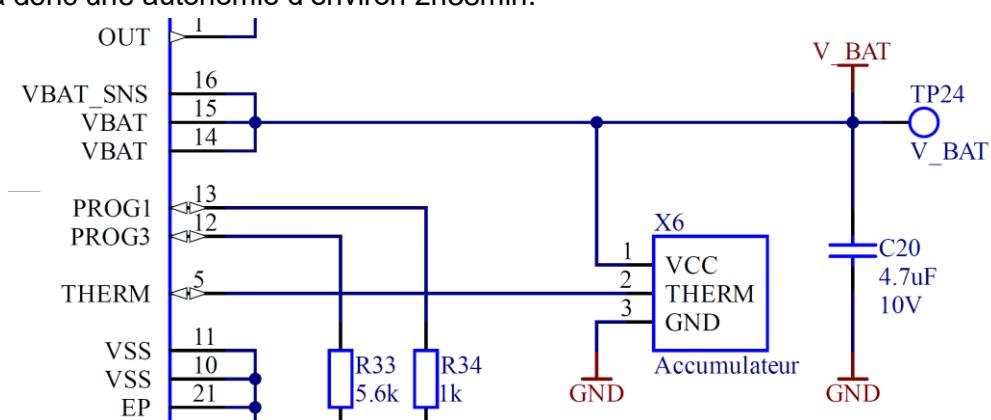


Figure 20 Branchement de l'accumulateur sur le contrôleur de charge

Il nous reste plus qu'à connecter le VCC et le GND de l'accumulateur, mais également sa thermistance sur le pin 5 (THERM) du contrôleur de charge. La valeur du condensateur de découplage et son placement ont été faits comme conseillé dans le datasheet.

1.3.1.5. Mesure de tension de l'Accumulateur

Afin de pouvoir mesurer la tension de notre accumulateur sans consommer de courant lorsque l'on ne fait pas, un petit circuit logique a été repris.

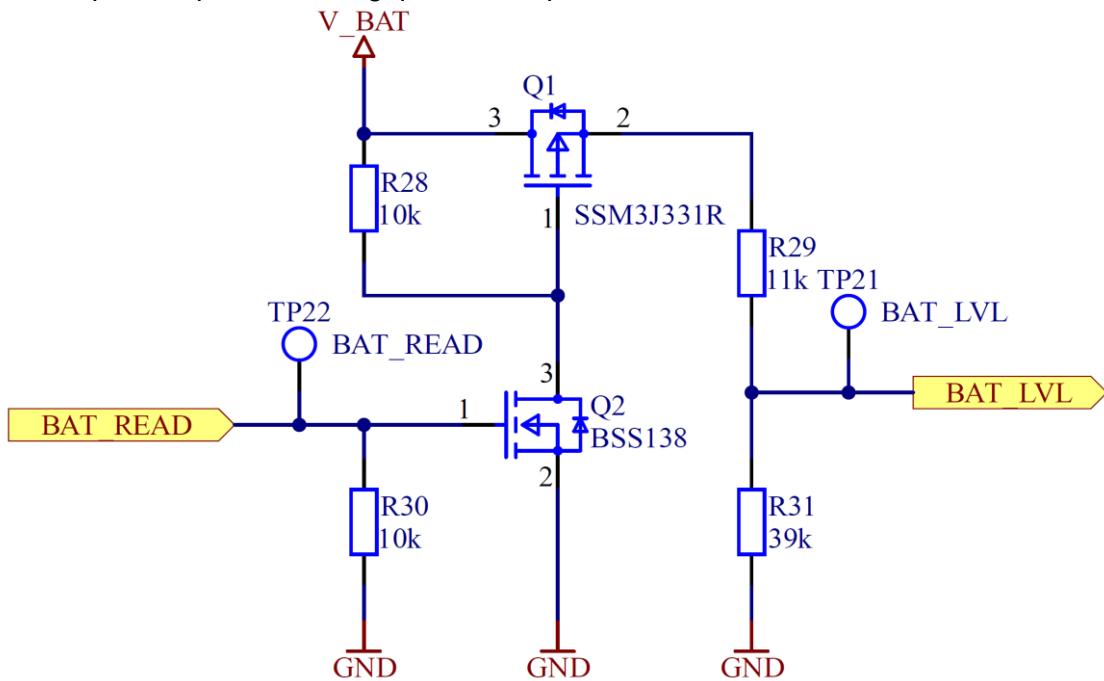


Figure 21 Montage logique pour la mesure du niveau de tension de l'accumulateur

Ici via la commande « BAT_READ » on sature Q2 avec un signal haut, qui à son tour sature Q1 en le reliant à la masse, ce qui permettra de lire la tension de l'accumulateur.

Pour cela on doit d'abord réadapter les niveaux, car l'accumulateur peut aller jusqu'à 4.2V quand il est complètement chargé, et notre microcontrôleur est alimenté en 3.3V.

Pour cela un pont diviseur est dimensionné, avec R24 qui sera fixé à 39kΩ.

$$E24 \\ R_{22} = \frac{(U_{BAT} - U_{uc})}{U_{uc}} * R_{24} = \frac{(4.2 - 3.3)}{3.3} * 39k\Omega = 10.63k\Omega \Rightarrow 11k\Omega$$

$$U_{BAT_LVL} = \frac{R_{24}}{R_{22} + R_{24}} * U_{BAT} = \frac{39 * 10^3}{11 * 10^3 + 39 * 10^3} * 4.2 = 3.276V$$

Lorsque l'accumulateur sera complètement chargé, un niveau de 3.276V sera donc mesure grâce à une pin analogique du microcontrôleur.



Figure 22 Pin analogique « AN2 » du microcontrôleur

Puis quand la mesure sera terminée, la commande « BAT_READ » sera remise à l'état bas.

Si l'accumulateur se décharge et le niveau atteint les 3.5V, un message sera affiché sur le LCD afin que l'utilisateur charge le Simon, dans cet état il sera impossible de lancer une nouvelle partie. Cela, car le LDO utilisé pour le 3.3V a une tension de Dropout de 210mV.

◆ Low 210mV Dropout at 1A

Figure 23 Dropout de 210mV du LDO 3.3V (Datasheet MAX1793, p. 1)

1.3.1.6. Adaptation des niveaux de tension

1.3.1.6.1. LDO 3.3V

Comme on est alimenté par un accumulateur qui peut varier au fur et à mesure qu'il se décharge, un étage de remise des niveaux de tension est nécessaire.

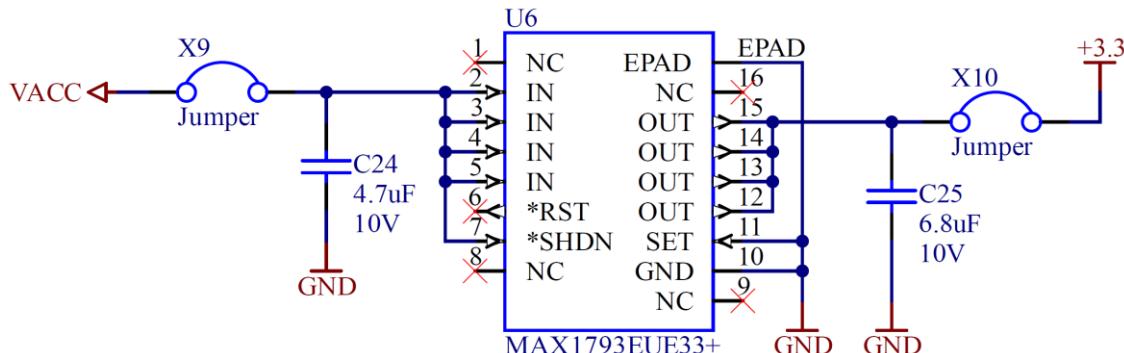


Figure 24 Branchement du LDO 3.3V MAX1793EUE33+

Ici on a donc un régulateur linéaire qui sort du 3.3V constant, avec une entrée qui peut varier entre 2.5V et 5.5V. Le montage et les valeurs ont été repris du datasheet.

Un jumper avant l'entrée, et un jumper après la sortie ont été placés pour faciliter la mise en service. En cas de défaillance ils nous permettent également de déconnecter le montage, et d'y connecter directement une alimentation de laboratoire.

1.3.1.6.2. Boost 5V

Pour les mêmes raisons on assure la stabilité du 5V, mais cette fois-ci avec un boost 5V. Le circuit a été dimensionné avec l'outil de Texas Instrument « webench power designer ».

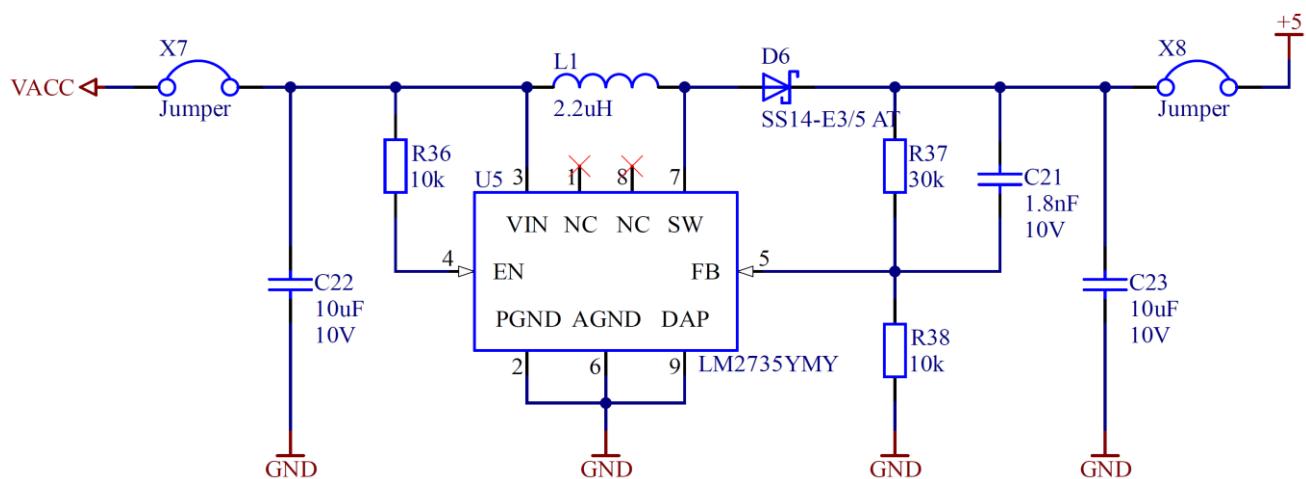


Figure 25 Branchement du boost 5V LM2735YMY

Cette fois-ci l'entrée peut varier entre 2.7V et 5.5V pour garantir une tension 5V stable en sortie. Malgré le dimensionnement par l'outil fourni par le fabricant, les valeurs ont pu être validées par des exemples de montages directement présents dans le datasheet.

Un design de layout est proposé dans le datasheet afin de pouvoir refroidir correctement le composant, tout en pouvant passer des courants aux alentours de 400mA.

De la même manière que pour le montage précédent, des jumpers avant et après le montage ont été placés, toujours pour faciliter la mise en service et le dépannage en cas de besoins.

1.3.2. Microcontrôleur

1.3.2.1. uC

Pour le choix du microcontrôleur, la famille PIC32 de Microchip a été conseillée dans le cahier des charges. Quelques critères ont été pris en compte pour le choix du modèle spécifique.

Le premier a été au niveau de la mémoire interne, car ne sachant pas encore la taille que fera le tableau des scores, et la place que prendra temporairement la lecture d'un fichier « .midi », la plus grande taille de la famille a été privilégiée. C'est donc au final les mêmes microcontrôleurs que l'on utilise au labo que je me suis redirigé.

TABLE 3: PIC32MX7XX USB, ETHERNET, AND CAN FEATURES

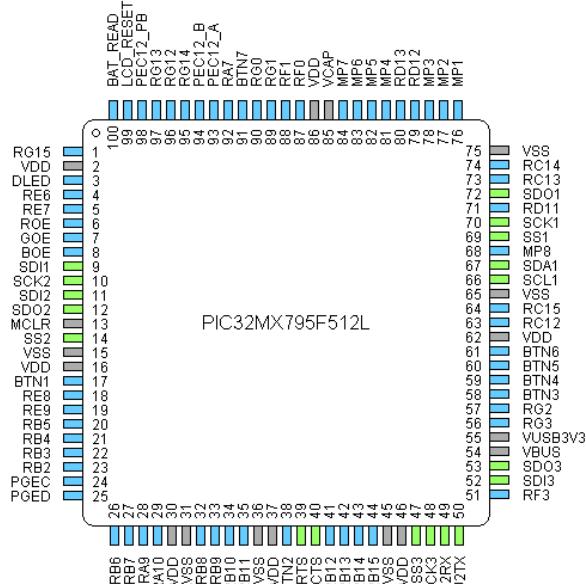
| Device | Pins | Program Memory (KB) | USB, Ethernet, and CAN | | | | | | DMA Channels (Programmable/Dedicated) | Timers/Capture/Compare | 10-bit 1 Msps ADC (Channels) | Comparators | PMP/PSSP | JTAG | Trace | Packages (4) | |
|-----------------|------|-------------------------|------------------------|-----|----------|-----|-------|-----|---------------------------------------|------------------------|------------------------------|-------------|----------|------|-------|--------------|----------------|
| | | | Data Memory (KB) | USB | Ethernet | CAN | | | | | | | | | | | |
| PIC32MX795F512H | 64 | 512 + 12 ⁽¹⁾ | 128 | 1 | 1 | 2 | 5/5/5 | 8/8 | 6 | 3 | 4 | 16 | 2 | Yes | Yes | No | PT, MR |
| PIC32MX795F512L | 100 | 512 + 12 ⁽¹⁾ | 128 | 1 | 1 | 2 | 5/5/5 | 8/8 | 6 | 4 | 5 | 16 | 2 | Yes | Yes | Yes | PT, PF, BG, TL |

Figure 26 Table de comparaison des microcontrôleurs de la famille PIC32 (Datasheet PIC32MX795F512L, p.2)

Il ne me restait plus qu'à décider entre les 64 pin ou le 100 pin. Avec tous les protocoles utilisés dans la carte (1x UART, 3xSPI et 1xI2C), je suis d'abord parti sur le 64 pin, car il possède tous ces drivers.

Mais malheureusement lorsque j'ai commencé à configurer les actions qu'auront les pins, je me suis rendu compte que lorsque l'on active les trois SPI de 64 pin, il est impossible d'avoir un I2C ou encore un UART.

J'ai donc dû partir sur celui à 100 pin, qui est le PIC32MX795F512L. Malgré cela, lorsque je configure les 1 x UART, 3 x SPI et 1 x I2C, il ne me restera plus qu'un seul I2C et un USB pur comme protocole de communication, pour de futures améliorations de la carte.



1.3.2.2. USB

Afin de pouvoir connecter le Simon au PC, on passera par un port micro USB. Il faudra donc pouvoir lire les informations USB qui arriveront sur la carte. Pour cela un convertisseur USB vers UART sera utilisé. Cela simplifie la prise en charge par le microcontrôleur de traiter une communication UART, plutôt qu'une communication USB lourde à traiter.

J'ai donc opté pour le composant « CY7C64225 » que j'ai déjà utilisé lors d'un projet de précédent de l'ETML-ES.

Pour le montage, un exemple du cas d'application alimenté en 3.3V a été suivi.

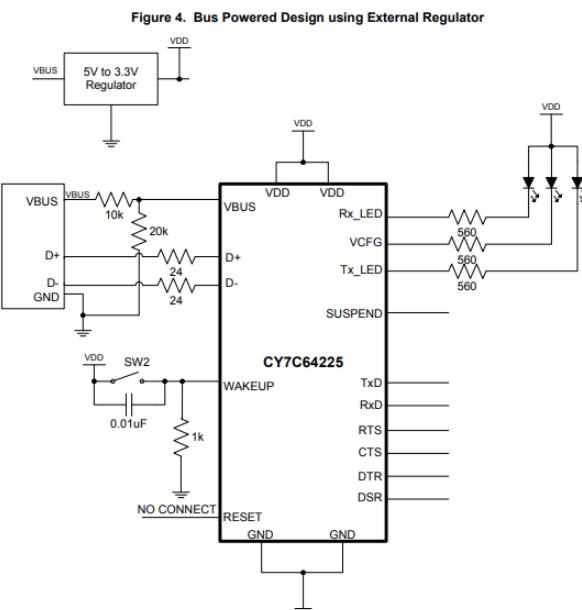


Figure 29 Schéma proposé dans le datasheet pour une alimentation de 3.3V (Datasheet CY7C64225, p.8)

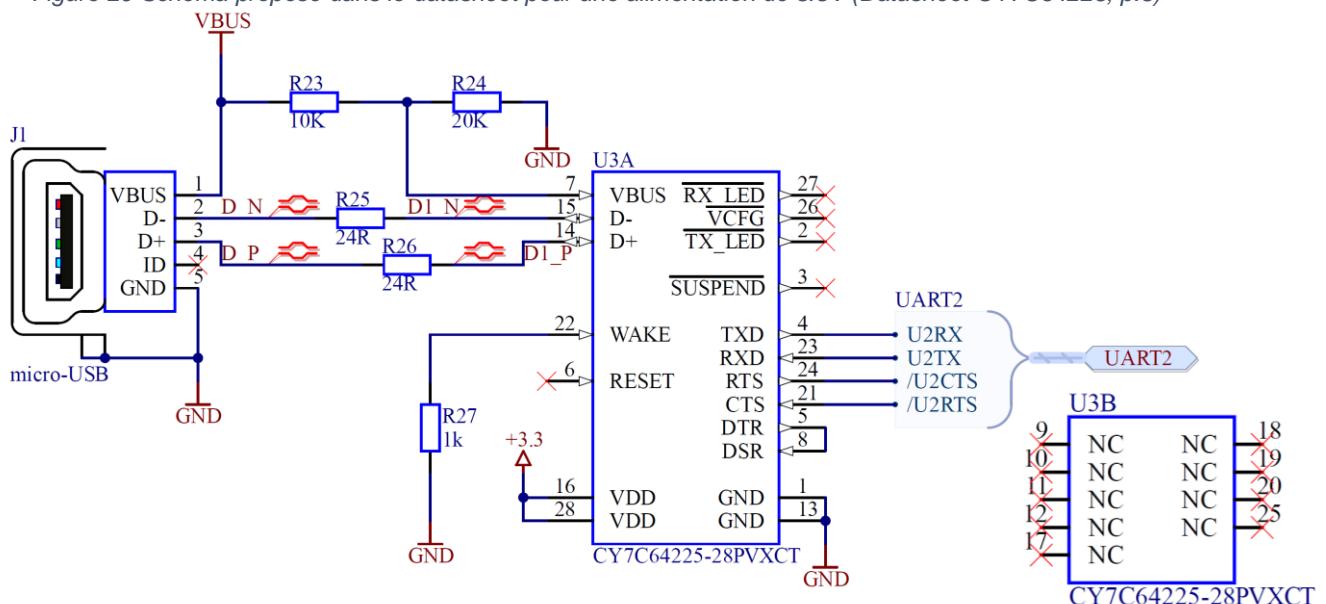


Figure 28 Branchement du convertisseur USB to UART « CY7C64225 »

Les LEDs conseillées ne sont pas utilisées, car pas besoin de ces informations, en plus on consommera donc moins. Les fonctions « RESET », « WAKE » et « SUSPEND » ne sont pas utilisées, donc elles ont été désactivées ou non connectées.

1.3.2.3. UART

Comme vu au précédent point, c'est via l'UAT que l'on va lire les informations venant de l'UASB du PC.

Suite aux configurations, c'est l'UART2 qui était le dernier UART qui puisse être utilisé.

TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

| Pin Name | Pin Number ⁽¹⁾ | | | | Pin Type | Buffer Type | Description |
|----------|---------------------------|--------------|---------------|--------------|----------|-------------|---------------------|
| | 64-Pin QFN/TQFP | 100-Pin TQFP | 121-Pin TFBGA | 124-pin VTLA | | | |
| U2CTS | 21 | 40 | K6 | A27 | I | ST | UART2 clear to send |
| U2RTS | 29 | 39 | L6 | B22 | O | — | UART2 ready to send |
| U2RX | 31 | 49 | L10 | B27 | I | ST | UART2 receive |
| U2TX | 32 | 50 | L11 | A32 | O | — | UART2 transmit |

Figure 30 Table de configuration des pin de l'UART2 (Datasheet PIC32MX795F512L, p.30)

J'ai pu confirmer et réserver le choix des pins via le configutateur Harmony pour l'UART2.

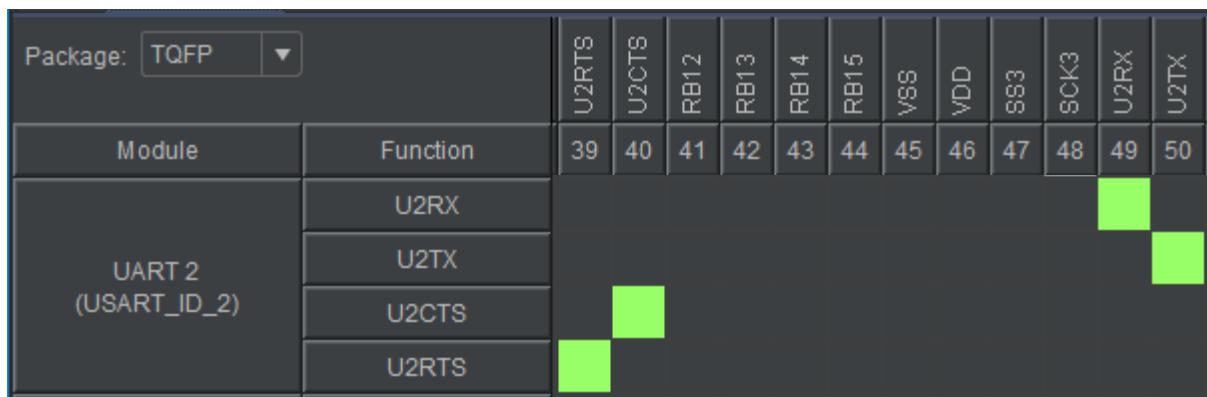


Figure 31 Configurations des pins du microcontrôleur pour l'UART2 dans Harmony

Afin de connecter le convertisseur USB to UART au microcontrôleur, il ne faut pas oublier de croiser les signaux comme l'exemple ci-dessous.

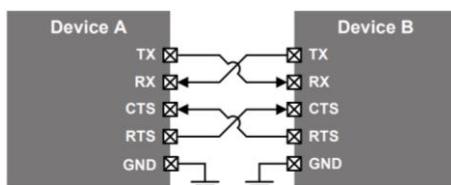


Figure 32 Connexion de l'UART Master to Slave

En effet, le PC étant le master, une fois les trames USB converties en UART, il faut les réceptionner avec le microcontrôleur. C'est pourquoi l'envoi de données « TX » du PC est connecté à la réception de données « RX » du microcontrôleur. De même dans l'autre sens de communication, vu que l'on va devoir envoyer les tableaux des scores du microcontrôleur au PC.

Il faudra faire la même chose pour les signaux « RTS » et « CTS ».

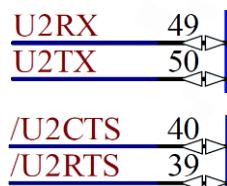


Figure 34 Communication UART venant du microcontrôleur



Figure 33 UART venant du convertisseur USB to UART

1.3.2.4. SPI

Trois communications SPI sont nécessaires, une pour le driver de LEDs RGB, un autre pour le DAC 12bit, et un dernier pour la lecture de la carte SD. J'ai donc pris les trois premiers SPI qui étaient disponibles.

TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

| Pin Name | Pin Number ⁽¹⁾ | | | | Pin Type | Buffer Type | Description |
|----------|---------------------------|--------------|---------------|--------------|----------|-------------|--|
| | 64-Pin QFN/TQFP | 100-Pin TQFP | 121-Pin TFBGA | 124-pin VTLA | | | |
| SDI1 | — | 9 | E1 | B5 | I | ST | SPI1 data in |
| SDO1 | — | 72 | D9 | B39 | O | — | SPI1 data out |
| SS1 | — | 69 | E10 | A45 | I/O | ST | SPI1 slave synchronization or frame pulse I/O |
| SCK1 | — | 70 | D11 | B38 | I/O | ST | Synchronous serial clock input/output for SPI1 |
| SCK2 | 4 | 10 | E3 | A7 | I/O | ST | Synchronous serial clock input/output for SPI2 |
| SDI2 | 5 | 11 | F4 | B6 | I | ST | SPI2 data in |
| SDO2 | 6 | 12 | F2 | A8 | O | — | SPI2 data out |
| SS2 | 8 | 14 | F3 | A9 | I/O | ST | SPI2 slave synchronization or frame pulse I/O |
| SCK3 | 49 | 48 | K9 | A31 | I/O | ST | Synchronous serial clock input/output for SPI3 |
| SDI3 | 50 | 52 | K11 | A36 | I | ST | SPI3 data in |
| SDO3 | 51 | 53 | J10 | B29 | O | — | SPI3 data out |
| SS3 | 43 | 47 | L9 | B26 | I/O | ST | SPI3 slave synchronization or frame pulse I/O |

Figure 35 Table de configuration des pin du SPI1, SPI2 et SPI3 (Datasheet PIC32MX795F512L, p.30 - 31)

J'ai pu confirmer et réserver le choix des pins via le configurateur Harmony pour le SPI1, SPI2 et le SPI3.

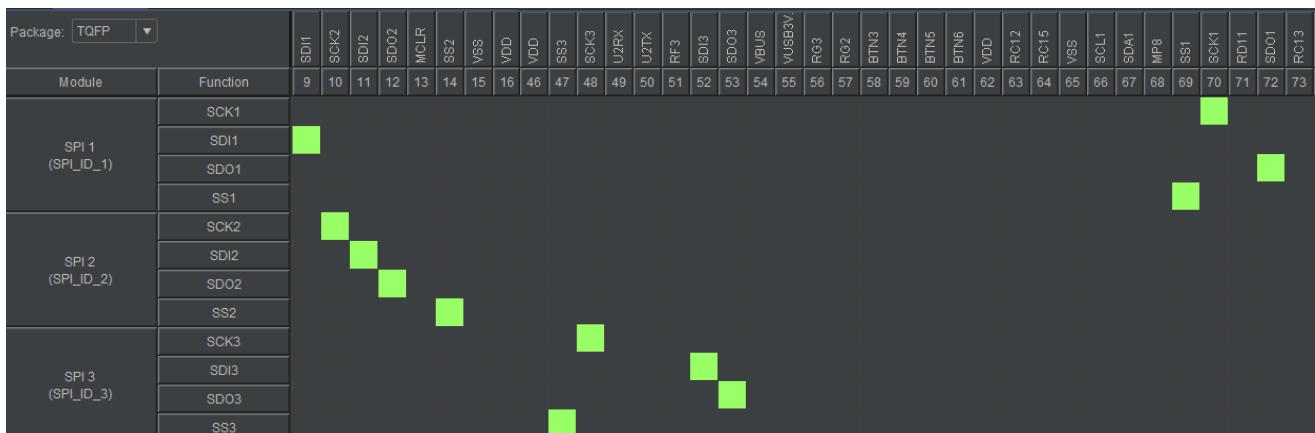


Figure 42 s des pins du microcontrôleur pour le SPI1, SPI2 et SPI3 dans Harmony



Figure 36 SPI1 sur le Driver de LED

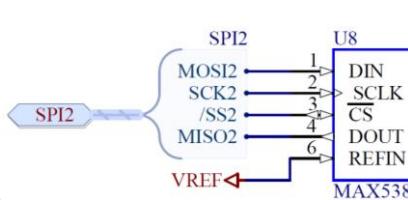


Figure 40 SPI2 sur le DAC 12bit

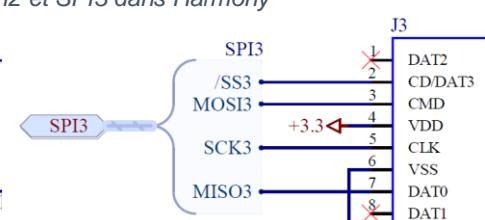


Figure 41 SPI3 sur la carte microSD



Figure 37 Pin du SPI1 sur le uC

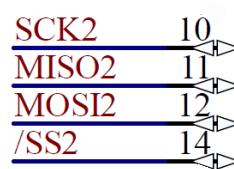


Figure 38 Pin du SPI2 sur le uC

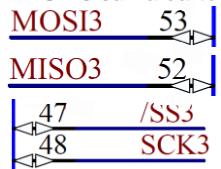


Figure 39 Pin du SPI3 sur le uC

1.3.2.5. I2C

Afin de pouvoir communiquer avec le LCD, il y avait deux possibilités, soit par SPI, soit par I2C. Malheureusement tous les SPI sont déjà dédiés à d'autres utilisations. J'ai donc dû directement utiliser l'I2C du LCD.

Je n'avais plus qu'à disposition deux I2C, j'ai donc opté pour le premier I2C1.

TABLE 1-1: PINOUT I/O DESCRIPTIONS (CONTINUED)

| Pin Name | Pin Number ⁽¹⁾ | | | | Pin Type | Buffer Type | Description |
|----------|---------------------------|--------------|---------------|--------------|----------|-------------|--|
| | 64-Pin QFN/TQFP | 100-Pin TQFP | 121-Pin TFBGA | 124-pin VTLA | | | |
| SCL1 | 44 | 66 | E11 | B36 | I/O | ST | Synchronous serial clock input/output for I2C1 |
| SDA1 | 43 | 67 | E8 | A44 | I/O | ST | Synchronous serial data input/output for I2C1 |

Figure 43 Table de configurations des pin de l'I2C1 (Datasheet PIC32MX795F512L, p.30)

J'ai pu confirmer et réserver le choix des pins via le configIBUTEUR Harmony pour l'I2C1.



Figure 44 Configurations des pins du microcontrôleur pour l'I2C dans Harmony

Pour que l'I2C fonctionne correctement, des pull-up doivent être positionnés sur les deux lignes de l'I2C. Dans le datasheet du LCD il nous le rappelle une fois de plus.

The maximum clock rate for I²C bus is 400 kHz.
Please be informed, that the pins SDA+SCK contain an internal resistance of 600 to 1000 Ohm, or even more (Important, because of the LO-level while reading data and the ACK-Bit).

Figure 45 Informations sur l'I2C du LCD (Datasheet EA DOGS164-A, p.7)

De plus on doit respecter une capacité sur le bus, comme vu en théorie.

Remarque : au niveau du kit PIC32MX, la valeur de résistance utilisée est de 2.2 kΩ, ce qui permet une capacité de bus maximum d'environ 400 pF. La tension utilisée est de 3,3 V, donc il serait possible de mettre des résistances de plus faible valeur (min 1 kΩ).

Figure 46 Conseille pour les valeurs des résistances pull-up de l'I2C du PIC32MX795F512L (Théorie MINF CH9, p.4)

C'est donc une pull-up de 2.2k qui sera mise sur les deux lignes de l'I2C.

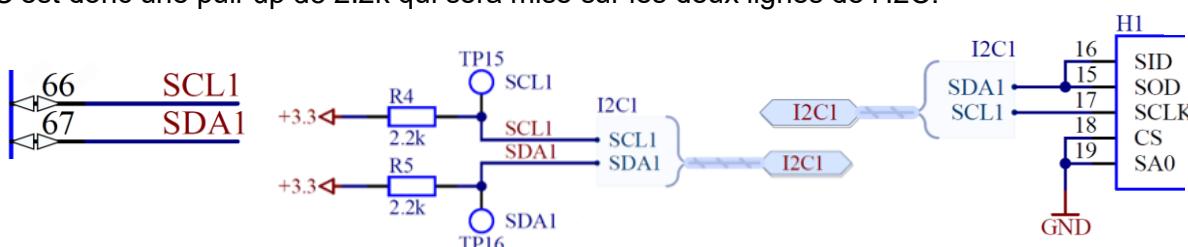


Figure 48 Communication I2C venant du microcontrôleur

Figure 47 I2C venant du LCD

1.3.2.6. Quartz

Le projet comprenant un haut-parleur, les signaux que l'on va lui envoyer devront être exactement aux bonnes fréquences. Si non, ça ne sera pas les bonnes notes qui seront jouées, et on aura l'impression que le Simon est désaccordé.

L'oscillateur ayant une variation de 0.9%, le choix s'est vite redirigé vers un quart avec 0.003% de variation. De plus il est prévu de travailler à 80MHz avec le microcontrôleur, car c'est les fréquences avec lesquels on a l'habitude de travailler en laboratoire. C'est donc un quartz de 8MHz qui sera pris, dont la fréquence sera modifiée dans microcontrôleur avec la PLL intégrée.

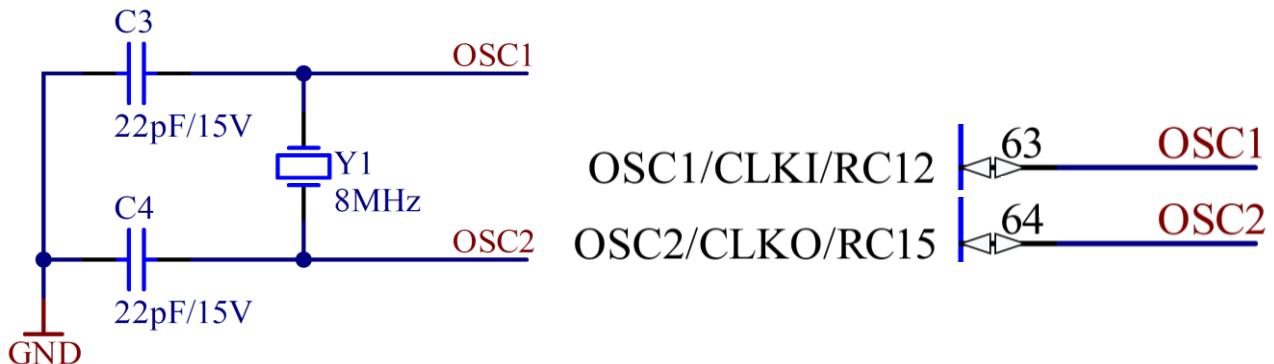


Figure 50 Branchement du quartz avec le condensateur de découplage

Figure 49 Branchement du quartz au uC

Le calcul des condensateurs a été fait grâce au $C_L = 18\text{pF}$ et le $C_0 = 7\text{pF}$ du quartz utilisé.

$$C7 = C8 = 2 * (C_L - C_0) = 2 * (18 - 7) = 22\text{pF}$$

1.3.2.7. Reset

Afin de pouvoir reset le microcontrôleur lors de la phase de développement, un bouton poussoir a été placé. Il ne sera pas accessible par l'utilisateur du Simon, car si le joueur a un problème bloquant, il lui suffira d'éteindre et de rallumer le Simon avec le bouton power ON / OFF. Un reset est également fait au démarrage avec ce montage repris d'un ancien projet de l'ETML-ES. Un reset pourra être fait également depuis le port de programmation.

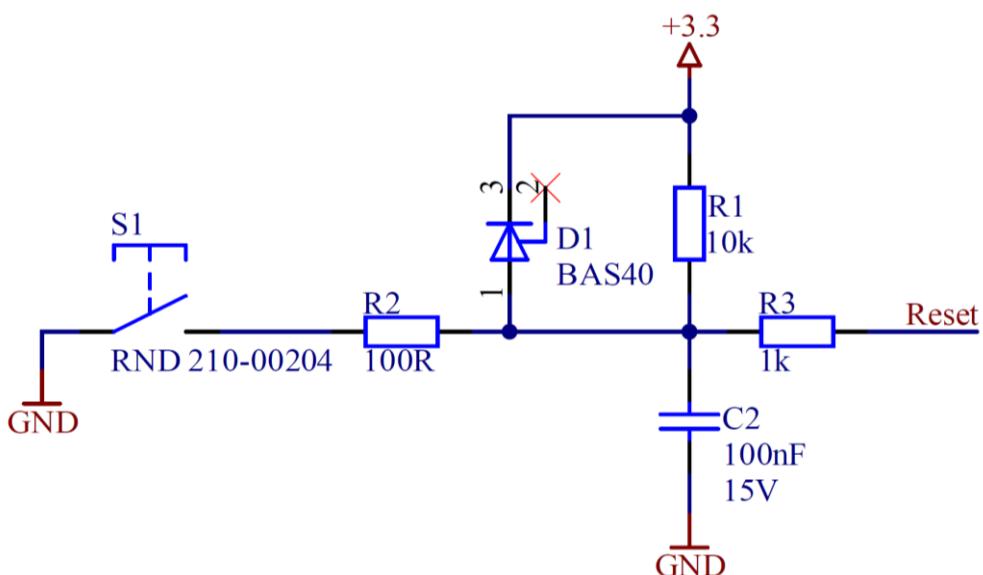


Figure 52 Branchement du circuit logique de reset du microcontrôleur



Figure 51 Reset du port de programmation

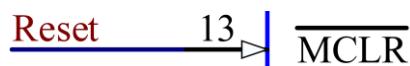


Figure 53 Pin de reset du microcontrôleur

1.3.2.8. Port de programmation

Le JTAG est là pour pouvoir programmer le microcontrôleur en chargeant le firmware à l'intérieur. Je pourrai également connecter le débugger pour débugger en temps réel notre firmware.

Le dimensionnement a été repris également du kit micro utilisé au labo. Il a été designé pour accueillir un debugger « ICD3 ».

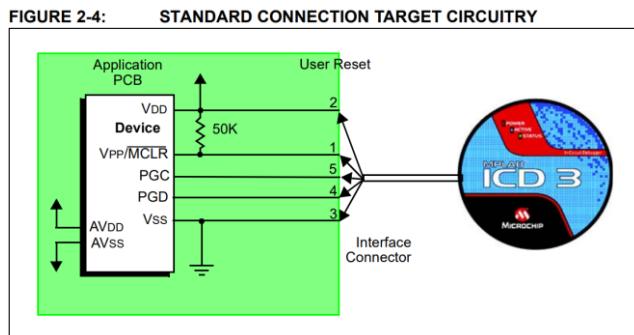


Figure 54 Branchements du « ICD3 »

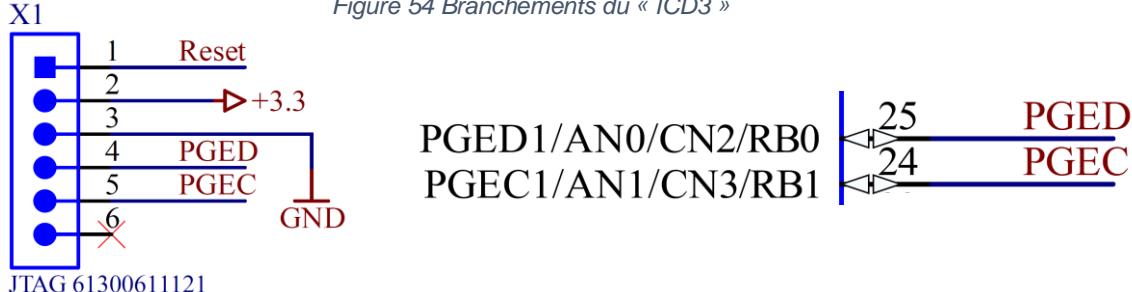


Figure 55 Branchement du port de programmation avec le microcontrôleur

1.3.2.9. Monitoring

Lors de la phase de développement, il sera utile d'avoir des pins en plus que l'on puisse commander via le microcontrôleur afin d'y mettre des signaux avec des états connus. Pour cela j'ai sorti huit pins digitales prévus que pour du debug. En plus de cela, j'ai également sorti un 3.3V et un GND sur le même port prévu pour un connecteur de câble plat 10 pin.

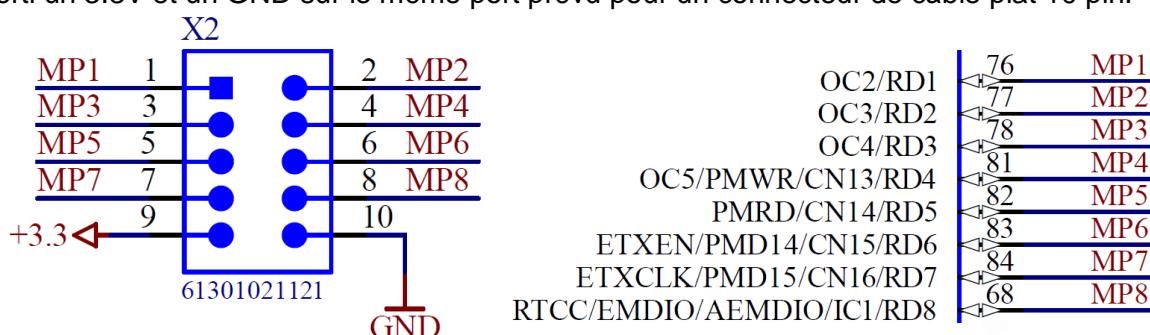


Figure 57 Port 10 pins pour le monitoring

Figure 56 Pins de monitoring du microcontrôleur

En plus de cela, j'ai sorti quatre points de test répartis dans toute la carte, afin de pouvoir venir connecter les appareils de mesure au GND le plus proche.

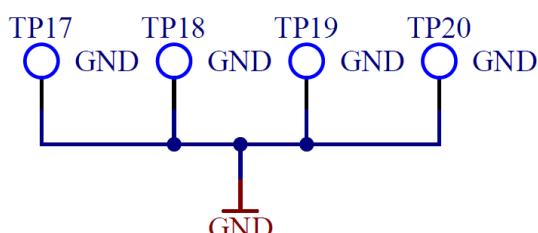


Figure 58 Pointes GND réparties sur la carte

1.3.2.10. LED de vie

Afin de simplifier le débogage pendant les premiers tests pendant le développement du firmware, une LED de vie a été connectée au microcontrôleur. Pour la version finale, la LED de vie pourra simplement clignoter pour annoncer que le microcontrôleur fonctionne, ou alors on peut simplement la laisser éteinte. On peut donc l'allumer et l'éteindre quand on le souhaite. De même une LED a été connectée au 3.3V, afin de confirmer que la carte est alimentée.

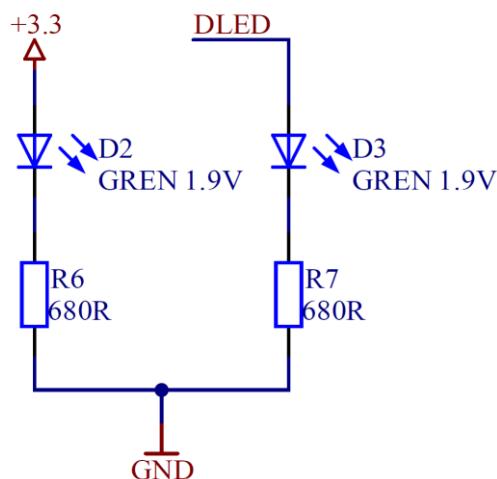


Figure 59 Led de vie connectée au microcontrôleur et au 3.3V

Pour une alimentation de 3.3V, avec sa résistance dimensionnée comme ceci :

$$R_x = \frac{(U_{3.3V} - U_{LED})}{I_{LED}} = \frac{(3.3 - 1.9)}{2 * 10^{-3}} = 700\Omega \Rightarrow E24 680\Omega$$

1.3.2.11. Découplage

Comme conseillé dans plusieurs datasheets, de nombreux condensateurs de découplage ont été placés. Rien que pour le microcontrôleur, sept condensateurs ont été nécessaires. Les placer au plus proche des composants à découpler est primordial lors de la réalisation du PCB.

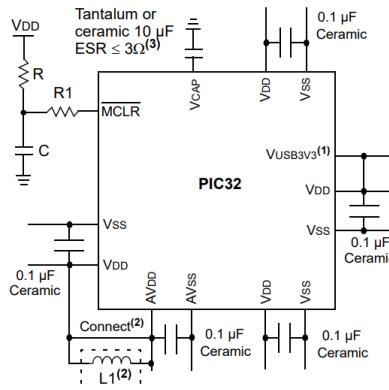


Figure 61 Découplage conseillé pour le microcontrôleur (Datasheet PIC32MX795F512L, p.38)

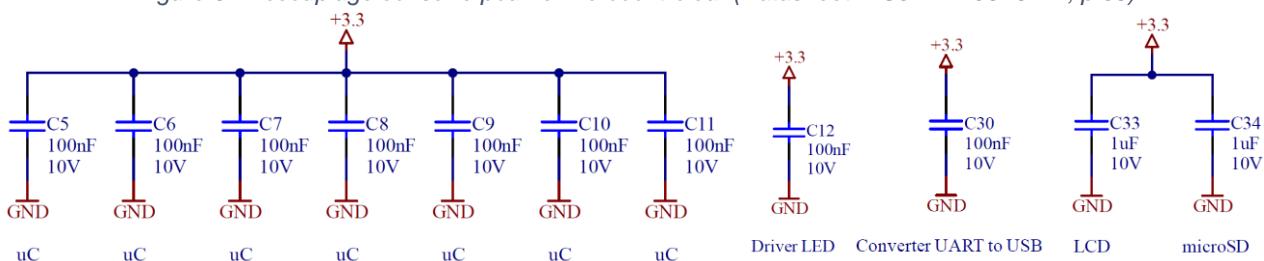


Figure 60 Tous les condensateurs de découplages des différents circuits intégrés du schéma

1.3.3. Buttons avec LED RGB intégrés

Afin de pouvoir interagir avec le Simon, des touches ont été demandées dans le cahier des charges. Mais également un retour lumineux via des LED RGB ont été demandés.

Initialement ces deux composants ont été prévus d'être séparés, mais suite à des recherches un bouton avec une LED RGB directement intégré a été trouvé. Cela rendra l'expérience utilisateur plus agréable, et le Simon sera un peu plus familier vis-à-vis de ses reproductions industrielles.

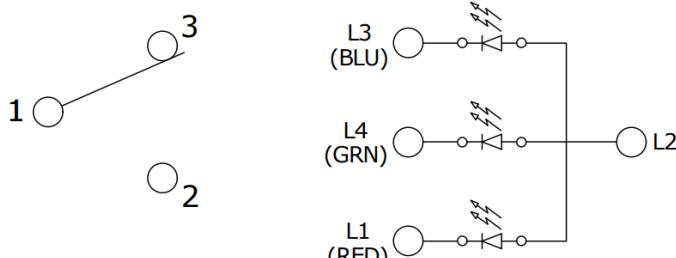


Figure 62 Schéma interne du bouton avec sa LED RGB (Datasheet ULP12OAP1RSFCL1RGB, p.1)

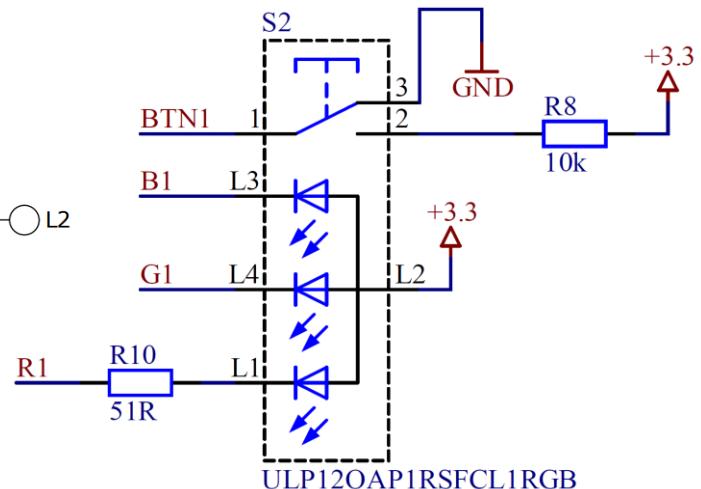


Figure 63 Branchements du bouton avec sa LED RGB

La partie avec le bouton poussoir sera connectée à une pull-up, afin de pouvoir lire un signal à l'état haut quand on appuiera sur le bouton.

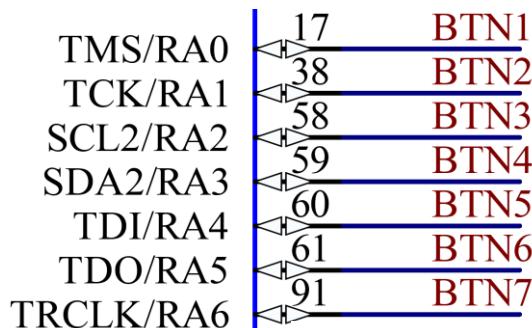


Figure 64 Connexion des bouton au uC

Afin de pouvoir lire les boutons, chaque bouton a été relié directement à une pin du uC.

- 15. LED COLOR: RED: 1 CHIP, 2.1-2.5 VDC@20mA
- GRN: 1 CHIP, 3.2-4.0 VDC@20mA
- BLU: 1 CHIP, 3.1-3.8 VDC@20mA

Figure 65 Propriétés des LEDs RGB du bouton poussoir (Datasheet ULP12OAP1RSFCL1RGB, p. 1)

Les LEDs seront connectés au 3.3V, et consommeront chacune jusqu'à maximum 20mA. En revanche pour la LED rouge une résistance a dû être ajoutée, car ne supportant pas les 3.3V. J'ai alors dimensionné une résistance qui redescende la LED à 2.3V.

E24

$$R = \frac{U_R}{I} = \frac{3.3 - 2.3}{20 \cdot 10^{-3}} = 50\Omega \Rightarrow 51\Omega$$

1.3.4. Driver de LEDs

Afin de commander toutes les LEDs RGB il va nous falloir un driver de LEDs. Il va également nous permettre de régler le courant pour toutes les LEDs avec une seule résistance.

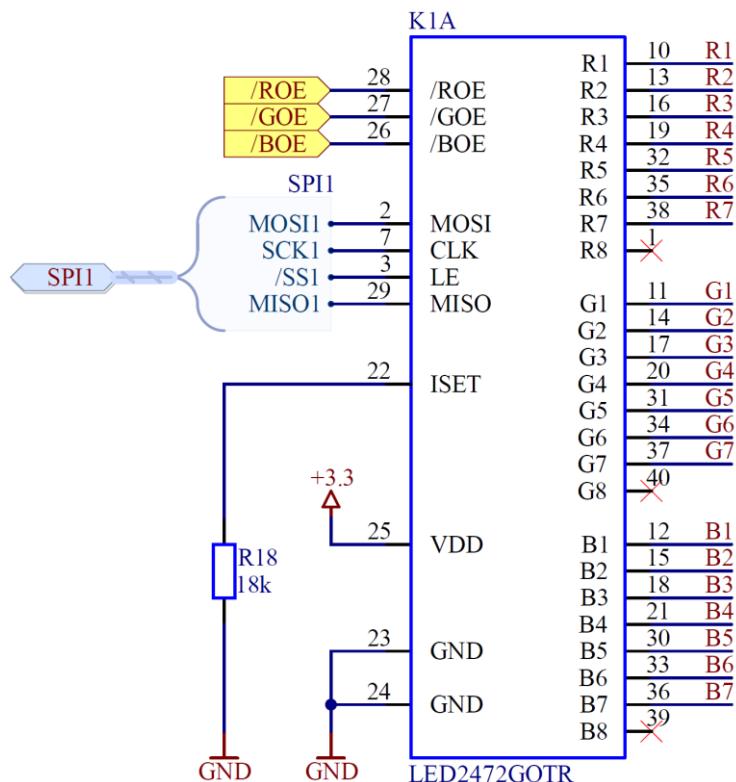


Figure 66 Branchement du driver de LEDs REG

On va pouvoir communiquer avec le driver de LEDs via un SPI. De plus il possède une pin dédiée à chaque couleur, pour éteindre toutes les LEDs d'une même couleur, ou alors les allumer (26, 27, 28).

Comme il est connecté à toutes les LEDs, il va pouvoir régler le courant pour chaque type de couleur séparément via des registres, en plus du réglage général fait par la résistance ISET.

La valeur de la résistance ISET a été reprise d'un exemple dans le datasheet, en prenant la configuration low current range et le courant de 20mA souhaité par LED.

Figure 14. I_{out} vs. gain

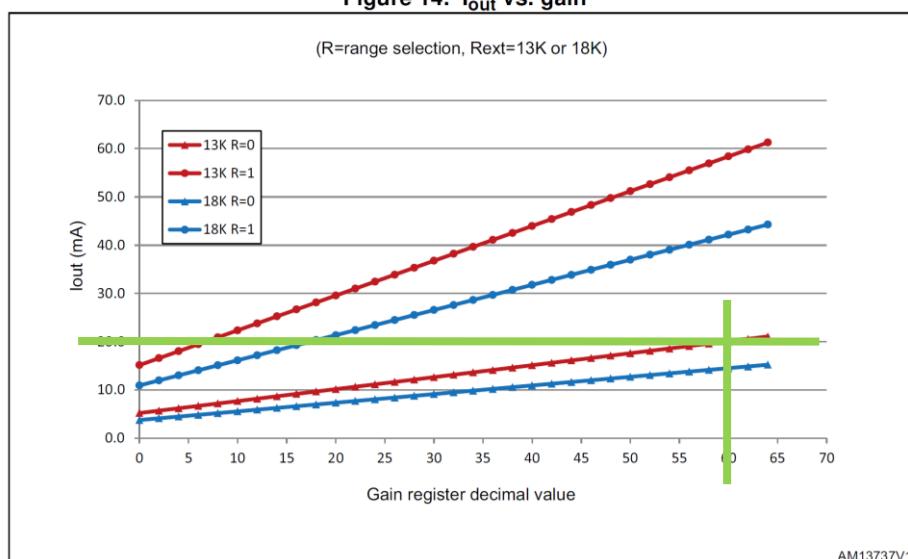


Figure 67 Dimensionnement de la résistance ISET qui fixe le courant de toutes les LEDs (Datasheet LED2472G, p.25)

1.3.5. Interface utilisateur

1.3.5.1. Encodeur rotatif PEC12

Afin de pouvoir naviguer dans les menus et valider des actions, l'utilisation d'un encodeur a été demandée. Je me suis inspiré de celui qui est utilisé dans le KIT microcontrôleur de l'ES.

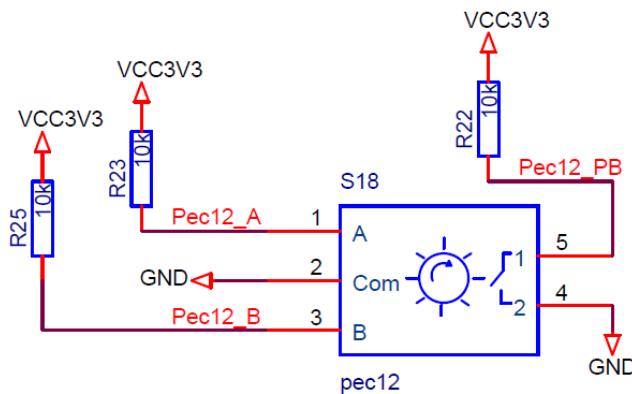


Figure 70 Connexion de l'encodeur PEC12 du KIT ES (Schéma 11020d)

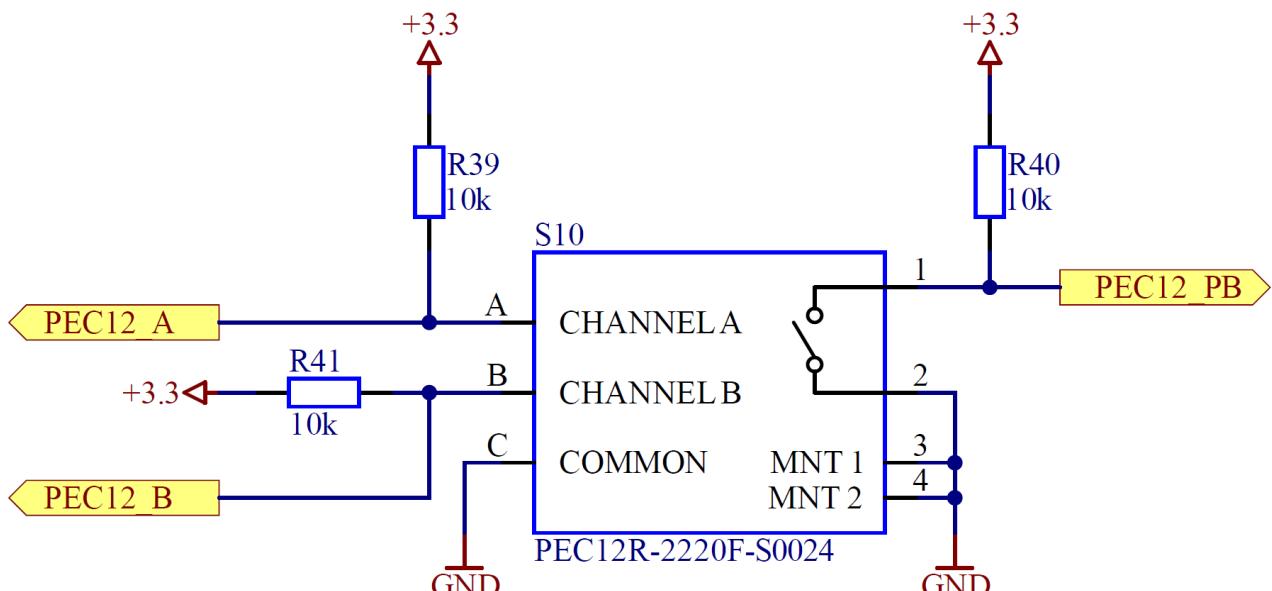


Figure 69 Branchements de l'encodeur rotatif PEC12 et son bouton

Le point commun de l'encodeur a été mis à la masse, et le channel A et B ont été tirés au 3.3V avec des pull-up de $10\text{k}\Omega$. Le bouton intégré a été traité de la même manière.

Puis le tout a été connecté sur le microcontrôleur sur des pins digital, dans un port où il y avait encore de la place.

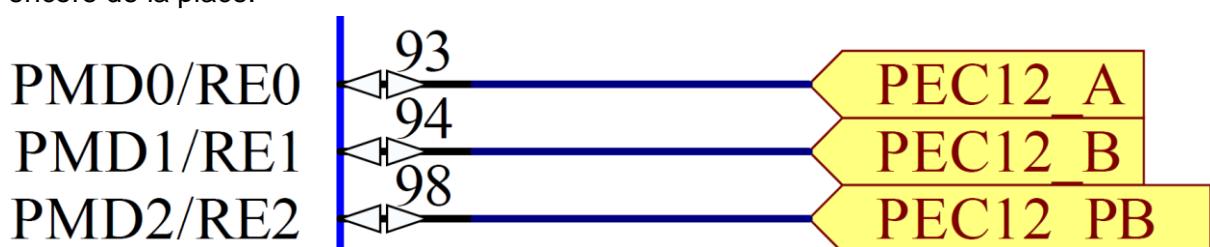


Figure 71 Pins de lecture des deux canaux et du bouton de l'encodeur PEC12

1.3.5.2. LCD

Un LCD d'au moins deux lignes était demandé dans le cahier des charges, il s'avéra à informer le joueur de plusieurs informations relatives au Simon.

Pour faire mon choix, j'ai voulu avoir quelque chose de compact, mais qui au même temps puisse afficher plusieurs informations à la fois. Mon choix s'est donc porté sur le « DOGS164-A », qui dispose de 4 lignes et de 16 caractères. C'est un LCD de type « Reflective », ce qui veut dire qu'il n'a pas de backlight, donc on consommera beaucoup moins.

Afin de pouvoir communiquer avec, c'est via l'I2C que je pourrais lui transmettre des informations. Même s'il y a également la possibilité de communiquer par SPI, mais due au grand nombre de protocoles utilisés, le choix s'est restreint pour les derniers composants.

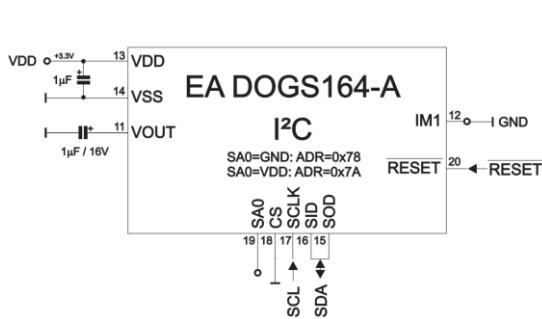


Figure 72 Schéma I2C (Datasheet DOGS164-A, p.4)



Figure 74 Connexion de la pin de reset du LCD sur le uC

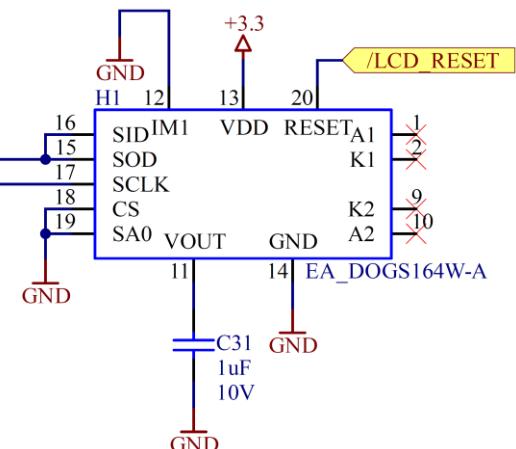


Figure 73 Schéma du branchement du LCD

Je me suis donc inspiré du schéma conseillé dans le datasheet pour l'utilisation de l'I2C. Pour l'adresse, c'est la « 0x78 » qui sera utilisée en mettant « SA0 » à l'état bas.

Pour la pin 12 « IM1 », il faut indiquer le protocole que l'on a choisi, c'est donc à la masse qu'il doit être mis pour la communication avec l'I2C.

| 12 | IM1 | H/L | Mode H: SPI / L: PC |
|----|-----|-----|---------------------|
| | | | |

Figure 75 Pin de configuration du choix du mode de communication (Datasheet DOGS164-A, p.8)

Puis pour finir, comme il n'y a pas de backligh pour ce modèle « Reflective », les pin 1, 2, 9 et 10 ne sont donc connectées à rien.

1.3.5.3. Carte micro SD

Afin d'avoir accès à une carte micro SD, un connecteur a été placé pour. Afin de communiquer avec, un SPI est utilisé pour la lire. Des exemples ont été trouvés sur internet, voir références.

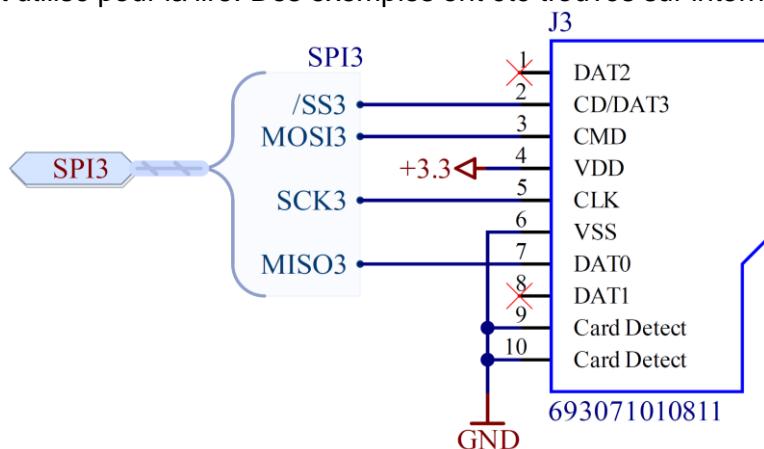


Figure 76 Branchement du connecteur de la carte micro SD avec la lecture via le SPI3

1.3.6. Son

1.3.6.1. DAC

Afin de pouvoir convertir les notes de musique au format numérique en analogique, il va nous falloir un convertisseur DAC.

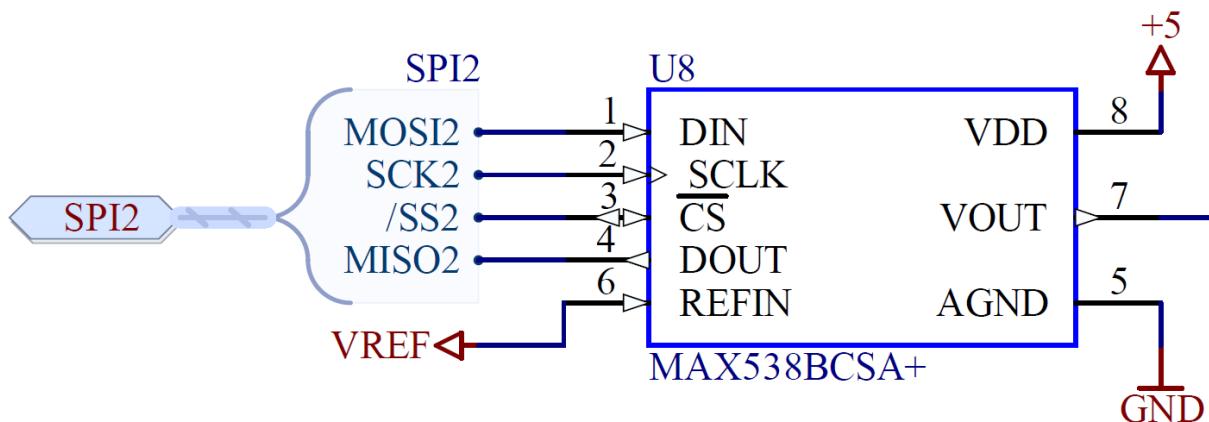


Figure 77 Branchement du DAC 12 bit

Ici le choix a été porté vers un DAC 12bit, car on doit être assez précis tout en restant avec des prix raisonnables pour un seul composant.

C'est via une communication SPI que l'on va lui transmettre les valeurs à convertir au DAC.

Puis afin de le faire fonctionner correctement, une tension de référence doit lui être transmise. Pour cela, il va nous falloir une tension de référence précise.

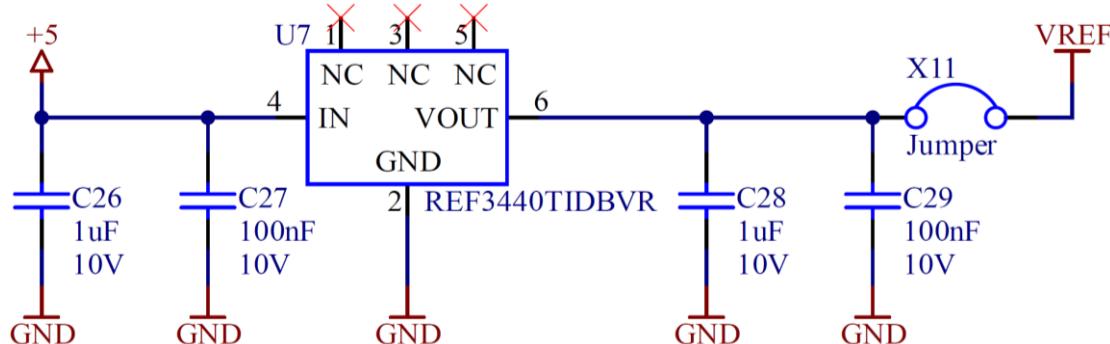


Figure 78 Générateur d'une référence stable à 4.096V pour le DAC 12bit

Généralement on utilise une référence de 4.096V avec les DAC de 12bit.

$$\text{Résolution} = \frac{V_{ref}}{2^{bit}} = \frac{4.096}{4096} = 1mV$$

Cette valeur de 4.096V n'est donc pas choisie au hasard, comme on peut le voir cela nous permet d'avoir au final une résolution de 1mV par bit.

L'utilisation d'un générateur de référence est primordiale, car sinon les conversions seront faussées. C'est pour cela que l'on a une précision de $\pm 0.05\%$ sur ce modèle.

Un jumper a également été mis à la sortie du générateur de référence, pour les mêmes raisons de mise en service et de dépannage que les autres jumpers précédemment utilisés.

1.3.6.2. Amplification

Une fois les notes de musique converties en analogique, on va pouvoir les envoyer sur le haut-parleur. Mais avant ça il faut pouvoir lui fournir suffisamment de courant pour qu'il puisse faire sortir des notes avec sa membrane dirigée par une bobine.

C'est pour cela que l'ajout d'un suiveur a été nécessaire juste après la sortie du DAC 12bit.

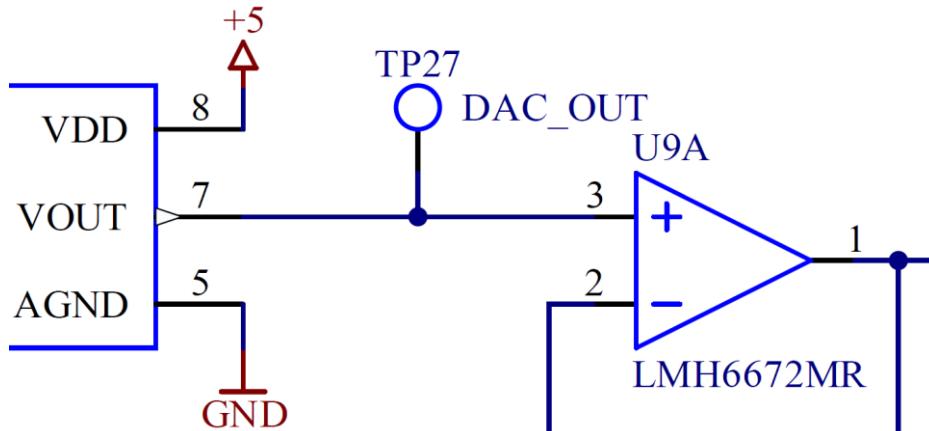


Figure 79 Montage d'amplificateur opérationnel suiveur en sortie du DAC 12bit

Pour le choix de l'amplificateur opérationnel, une attention particulière a été apportée au courant max qu'il pouvait fournir en sortie.

Car comme l'on va avoir un haut-parleur de 8Ω , et que dans le meilleur des cas on pourrait avoir une tension de 5V dessus, un courant pic de 625mA serait alors sollicité.

$$I = \frac{U}{R} = \frac{5}{8} = 625mA$$

C'est donc le LMH6672MR qui a été choisi, car en plus de respecter le premier critère, il peut être alimenté en mono avec une tension d'alimentation de 5V.

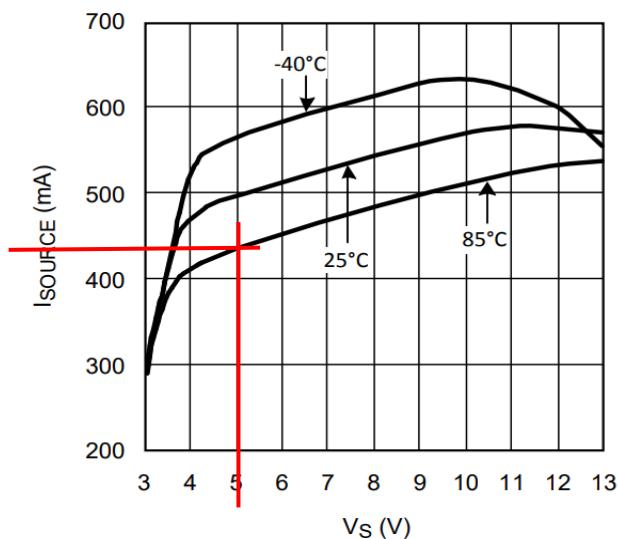


Figure 11. Sourcing Current vs. Supply Voltage

Figure 80 Graphique du courant vs la tension de sortie (Datasheet LMH6672MR, p.8)

Grâce à ce graphique, on peut donc estimer que l'on pourra tirer un maximum d'environ 430mA avec une alimentation de 5V. Comme on est en dessous de la consommation max de l'haut-parleur, le son sera tout simplement moins fort.

1.3.6.3. Haut-parleur

Pour le choix du haut-parleur, on m'a proposé d'en prendre un soit de 1W ou alors de 2W. Mon choix c'est directement porté vers le 1W, car qui dit moins de puissance, dit moins de consommations, donc plus d'autonomie en mode portable.

C'est donc un haut-parleur de 1W et 8Ω que mon choix c'est porté, car plus faible puissance dite aussi moins de consommation.

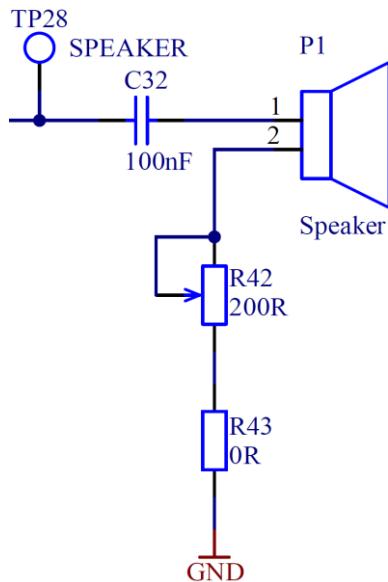


Figure 81 Branchement du haut-parleur avec le trimmer pour régler le volume

Afin de pouvoir ajuster un peu son volume, un trimer a été ajouté en série de l'haut-parleur. Pour le dimensionner, des calculs empiriques de plage ont été faits. Car plus on augmente la résistance, moins de courant passera, donc le son sera plus faible.

$$I = \frac{U}{R} = \frac{5}{208} = 24.04mA$$

$$I = \frac{U}{R} = \frac{5}{228} = 21.9mA$$

Mon choix s'est arrêté sur un trimer de 200Ω, car si l'on passe à des résistances plus grandes la diminution du courant devient de plus en plus faible.

De plus avec une résistance la plus faible possible, on obtient plus de précision du réglage sur la totalité d'un tour du trimer. Dans le cas d'une trop grande résistance, peine avoir fait un quart de tour, on atteindrait déjà de faibles valeurs de courant. Afin de pouvoir appliquer un offset, une résistance de 0Ω a été placée en série avec le haut-parleur. Sa valeur sera amenée à être changée, si lorsque l'on règle le trimer au minimum, on entend encore du son.

Un condensateur de liaison a été placé afin de couper la composante continue en sortie de notre amplificateur, qui lui est alimenté au 5V et GND.

$$C = \frac{1}{2 * \pi * R * f_c} = \frac{1}{2 * \pi * 208 * \frac{10}{\sqrt{2}}} = 109nF \Rightarrow 100nF \quad E24$$

Ici c'est le calcul pour le condensateur du filtre passe haut, avec la fréquence de coupure divisée par racine de deux, comme vu en théorie au CH5 ELAN page 15.

1.4. Conclusion phase de design

Les choix des composants ont été pris pour faire en sorte que le Simon soit portable, et puisse tenir dans la main de l'utilisateur.

C'est donc avec un accumulateur Li-po que le Simon pourra tenir un peu plus de deux heures. On pourra ensuite le recharger soit via le microUSB, ou alors plus rapidement avec le port Jack. Une surveillance du niveau de l'accumulateur sera faite par le microcontrôleur, et qui affichera les informations nécessaires à l'utilisateur sur l'écran LCD réflectif.

Le microcontrôleur utilisé sera donc le PIC32MX795F512L, qui est utilisé sur les KIT ES dans le laboratoire. Il comporte une très large palette de moyens de communication, dont notamment je vais utiliser.

Pour résumer, j'utilise 3 SPI, le premier pour le driver de LEDs, le deuxième pour le DAC 12bit et le troisième pour la carte microSD. Puis j'utilise la communication UART pour lire les données arrivant de l'USB du PC. Enfin j'utilise l'I2C pour commander l'écran LCD.

Pour les boutons, des tout en un ont été trouvés, cela comporte donc un bouton poussoir avec une LED REG intégrée. Pour commander les LEDs RGB un driver de LEDs va être utilisé, car trois les de couleur avec sept boutons, nous donne 21 LEDs à commander séparément. Il nous évite donc de devoir connecter une pin du microcontrôleur à chaque LED, mais à la place de pouvoir commander toutes les LEDs juste qu'avec un SPI. De plus le courant pourra être réglé via une seule résistance.

Afin de pouvoir sélectionner les différentes options parmi les menus du Simon, un encodeur rotatif avec un bouton sera utilisé.

Une entrée pour une carte micro SD a également été prévu pour le mode mélodie.

Pour la partie sonore, un DAC 12bit suivi d'un étage d'amplification ont été nécessaire afin de pouvoir sortir un son sur le haut-parleur 1W choisi. On d'ailleurs gérer le volume du haut-parleur via un trimeur lors de la phase de développement.

Vous trouverez le budget total de la réalisation du Simon dans la liste des pièces.

2. Hardware

2.1. PCB

2.1.1. Concept de design

Je n'ai eu aucune contrainte imposée dans le cahier des charges pour le design du PCB. J'ai donc commencé par les parties les plus importantes du Simon.

Tout d'abord la forme générale du PCB est ronde, car j'ai voulu que l'utilisateur puisse le tenir dans ces mains. Pour son diamètre je ne pourrais donc pas faire du 20cm, car il ne tiendrait plus dans une main.

Pour déterminer le diamètre, j'ai tout simplement commencé par placer les sept boutons de manière circulaire, afin de former un heptagone entre eux. J'ai fait en sorte de laisser assez d'espace entre chaque bouton, afin que l'utilisateur n'appuie sur deux boutons au même temps dans faire exprès.

Puis j'ai esquissé deux concepts de design de tout l'assemblage qui dépendrait de la taille de l'écran LCD, et de l'emplacement voulu de l'encodeur.

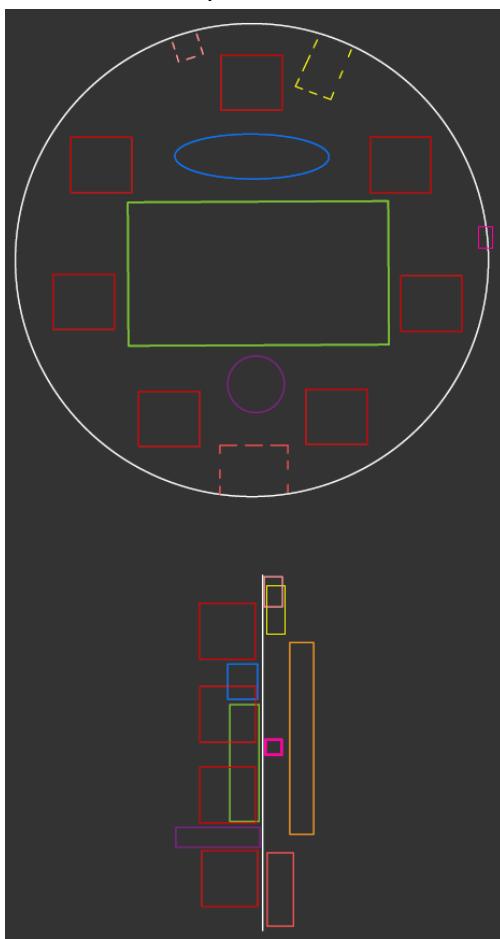


Figure 83 Concept avec encodeur sur le dessus

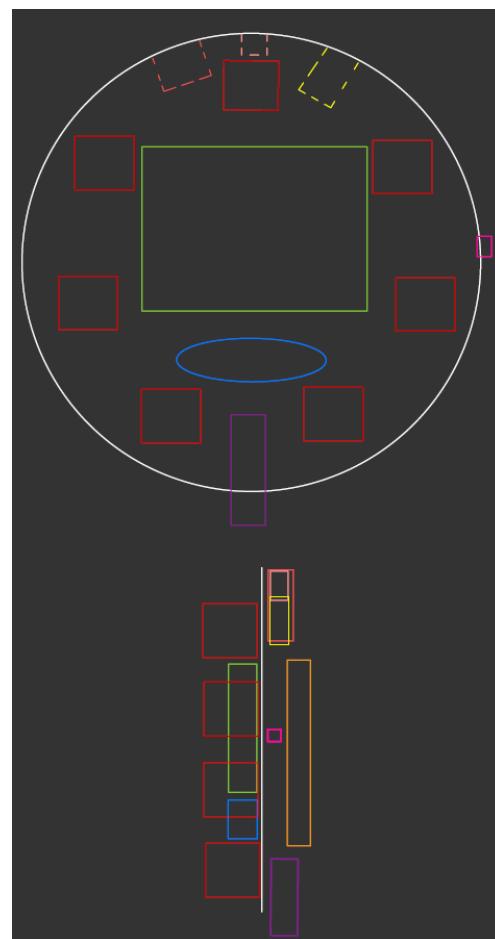


Figure 82 Concept avec encodeur en dessous

Après la disposition sommaire de tous les principaux éléments mécaniques, je suis dans un premier parti sur un disque de 12cm de diamètre. Puis après avoir placé la plupart des composants, il restait beaucoup de zones vides sans composants. J'ai donc réduit le diamètre et il est passé à 10cm.

2.1.2. Largeur des pistes

Afin de déterminer la largeur des pistes, je me suis référé au graphique dédié d'Eurocircuits.

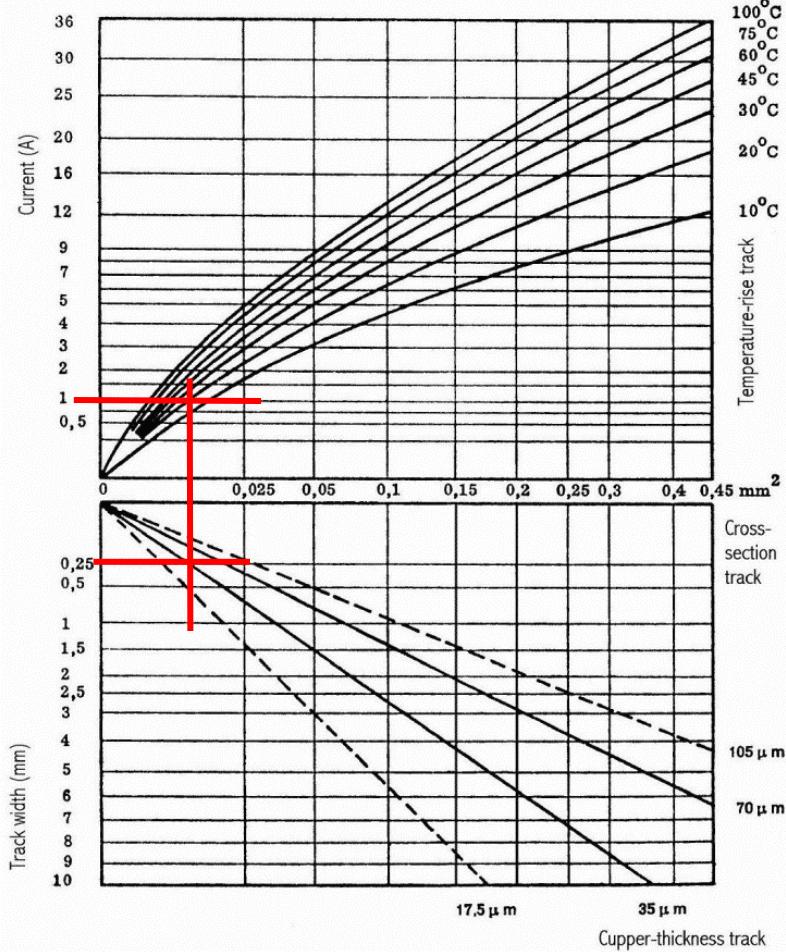


Figure 84 Graphique de la largeur des pistes d'Eurocircuits (www.eurocircuits.com)

Ici plusieurs paramètres entrent en compte, tout d'abord on définit le courant max qui passera dans notre piste sur le côté gauche du cadran du haut. Ici j'ai choisi 1A, car c'est via le Jack que l'on a défini que l'on puisse atteindre maximum ce courant.

Puis il faut déterminer la température, ici j'ai pris la courbe de 20°C, car c'est la plus proche de la température ambiante.

Ensuite il faut passer sur le cadran du bas avec les droites d'épaisseur de cuivre des pistes. La taille standard que l'on commande est de 35um, donc c'est celle-là que j'ai prise.

Puis la dernière étape est de relever la valeur de la largeur de piste que l'on doit respecter. Ici on tombe sur une largeur de 0.25mm (9.84mils).

Cette valeur est la largeur minimale à respecter, j'ai donc pris au final des valeurs un peu plus grandes afin de surdimensionner le courant qui pourra y passer.

Je vais donc utiliser des pistes de 20mils (0.508mm) de largeur pour les parties où un courant de jusqu'à 1A pourrait passer.

Puis des pistes de 15mils (0.391mm) de largeur seront utilisées pour les alimentations en général qui consommeront dans les dizaines de milli ampères.

Enfin, pour les pistes pour les signaux numériques, c'est une largeur de 9mils (0.229mm) qui sera utilisée.

2.1.3. Placement des composants stratégique

Les composants ont été à chaque fois été placés par blocs de fonctionnement, et petit à petit ils ont été rapprochés.

Les deux zones avec le principaux sont, la zone analogique (verte), et la zone numérique (bleu).

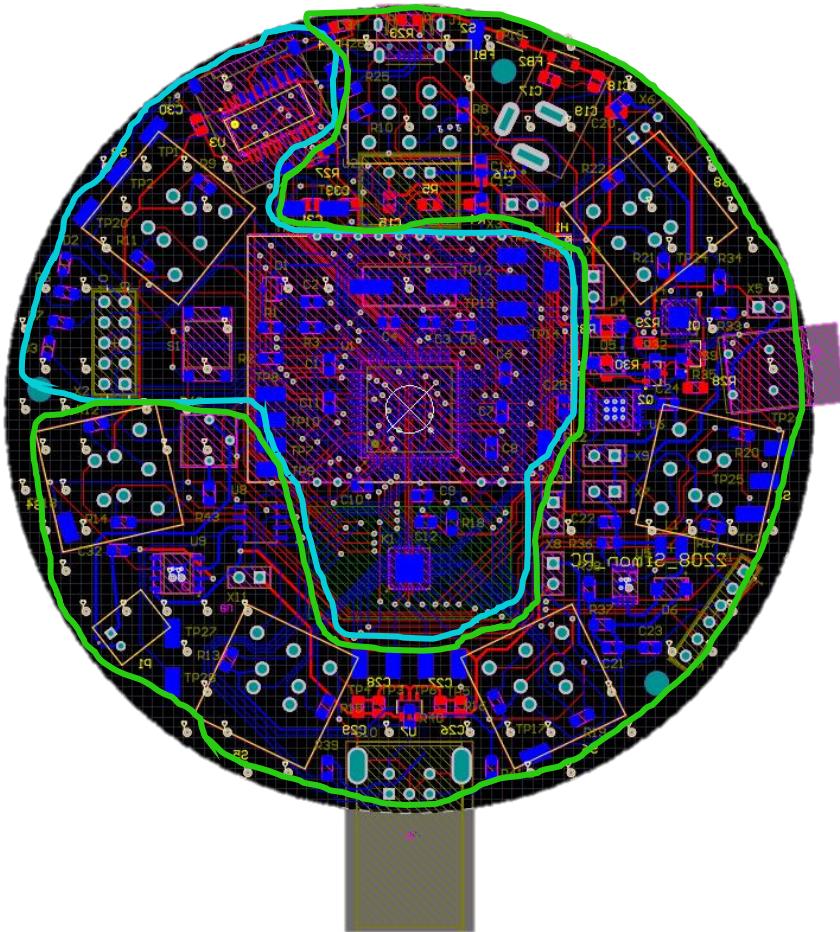


Figure 85 Les deux principales zones, analogique (vert) et numérique (bleu), vue BOT

En procédant de cette manière, les composants qui sont interconnectés entre eux sont proches les uns des autres, et cela évite de tirer des pistes qui traversent tout le PCB.

Le haut-parleur est positionné en dessous du LCD sur la partie TOP, où aucun composant n'a été placé. Des craintes de perturbations pouvant venir du haut-parleur nous ont fait le déporter, et c'est pour cela qu'il est connecté à un connecteur JST avec des fils, et non pas directement brasé sur le PCB.

De plus, afin d'essayer de quand même essayer de positionner le haut-parleur à son emplacement de prévu, des plants de masse sur les deux côtés ont été créés. Tout ça formera alors un blindage, et il sera connecté à la masse, afin de rediriger toutes les perturbations entrantes. Du stiching a également été placé afin d'uniformiser le tout, et de raccourcir le plus possible le chemin parcouru par les perturbations magnétiques, mais également ESD par la même occasion, entrantes dans la carte.

Trois trous de passage ont été créés afin de pouvoir laisser passer les trois vis M3 traversantes, qui fixeront le boîtier mécaniquement.

2.1.4. Design PCB

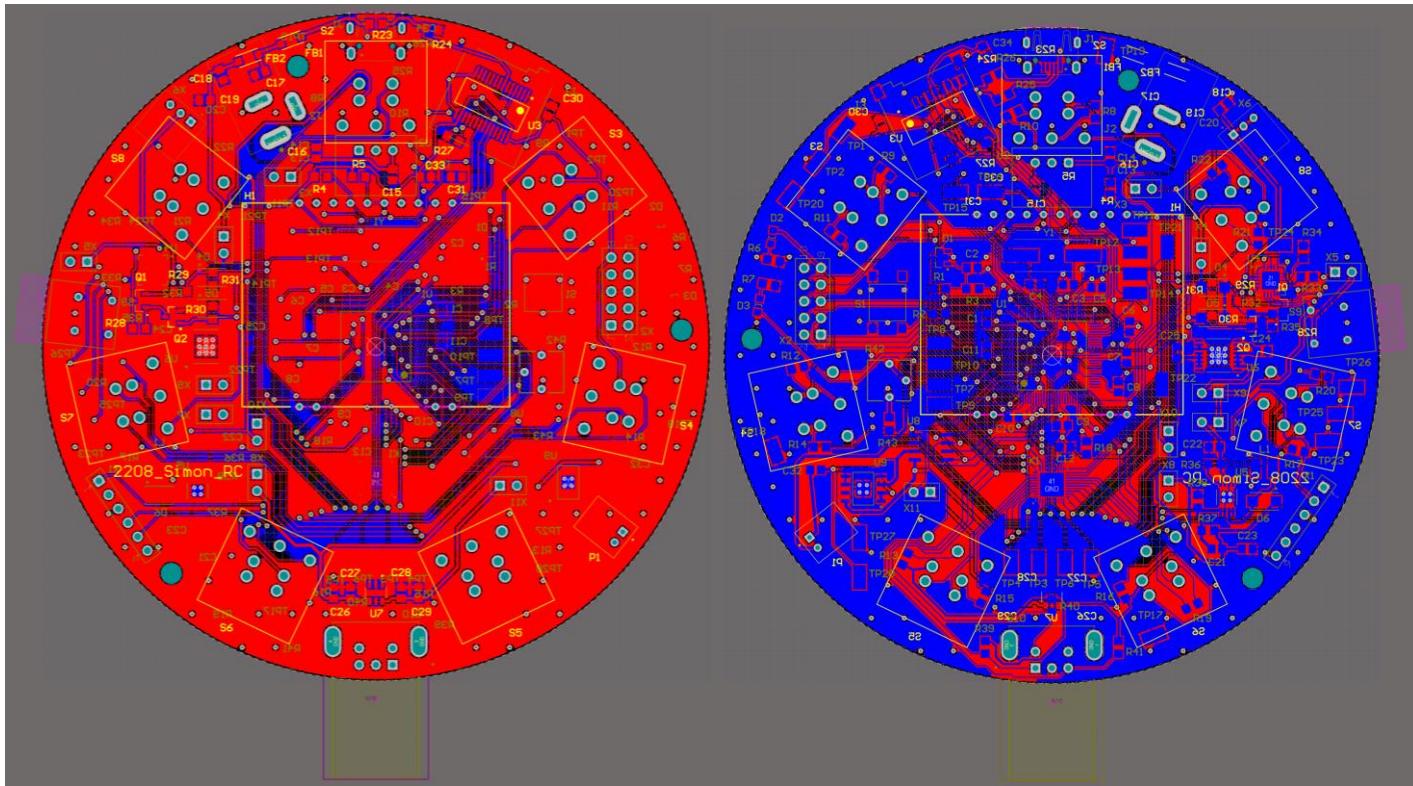


Figure 87 Vue 2D du TOP (à gauche) et du BOT (à droite) du PCB du Simon

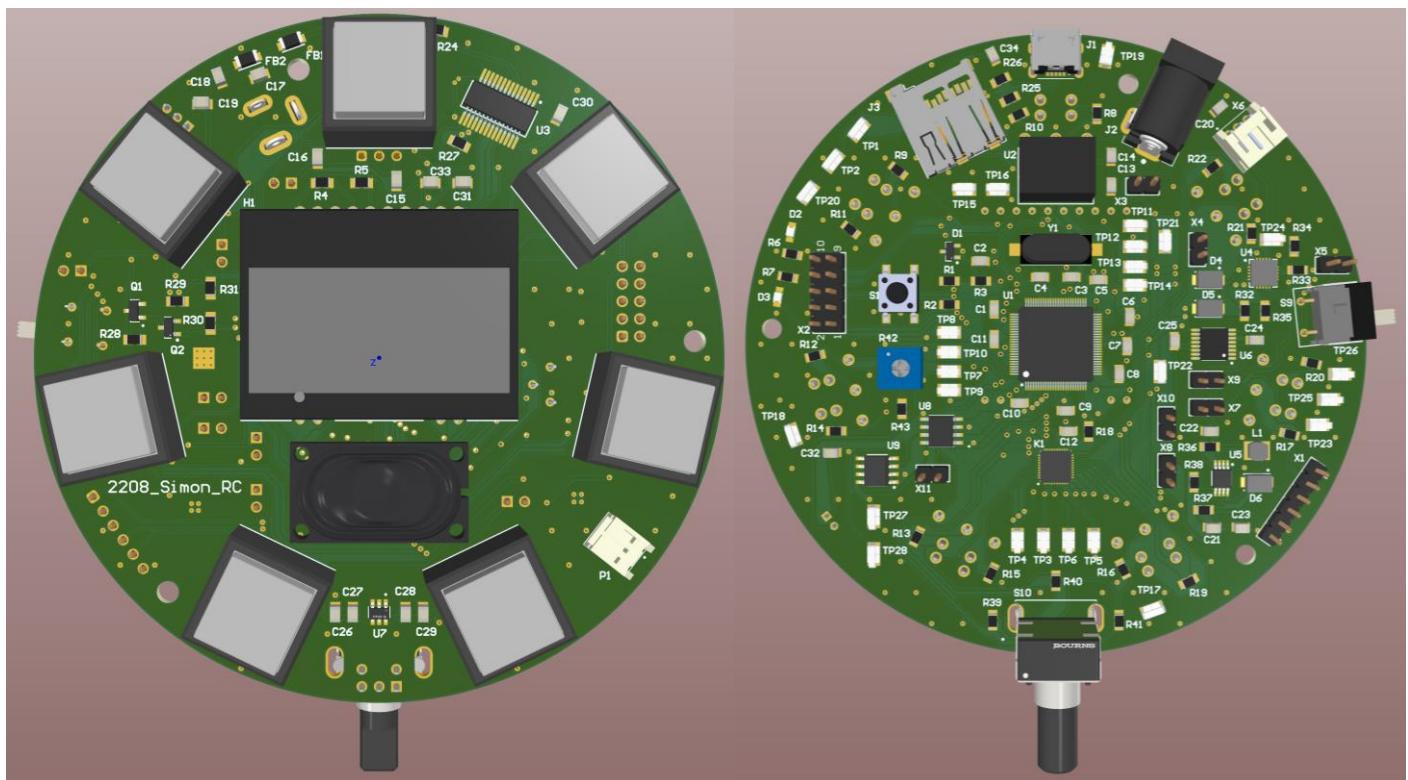


Figure 86 Vue 3D du TOP (à gauche) et du BOT (à droite) du PCB du Simon

2.2. Fabrication

2.2.1. Courbe de température

Afin de pouvoir braser les boîtiers QFN, et de simplifier la phase de production, un stencil a été commandé lors de la commande de PCB. Cela me permet de déposer de la pâte à braser sur le PCB, puis de placer le composant et finir par braser les composants au four.

Pour cela il faut déterminer une courbe de température. Malheureusement tous les composants n'ont pas la même courbe, j'ai donc dû déterminer la courbe la plus critique.

Au final uniquement le connecteur microUSB, le connecteur pour la carte micro SD et la bobine SMD avaient une courbe de température dans leurs datasheets. Pour le reste des composants qui n'ont pas, cela veut dire qu'il se trouve dans la moyenne, et qu'ils supportent une très grande plage de températures à différents timings.

Parmi les trois composants, c'est la bobine qui a les températures les plus basses, et les temps les plus courts, c'est donc sur elle que je me suis basé.

■ RECOMMENDED REFLOW PROFILE

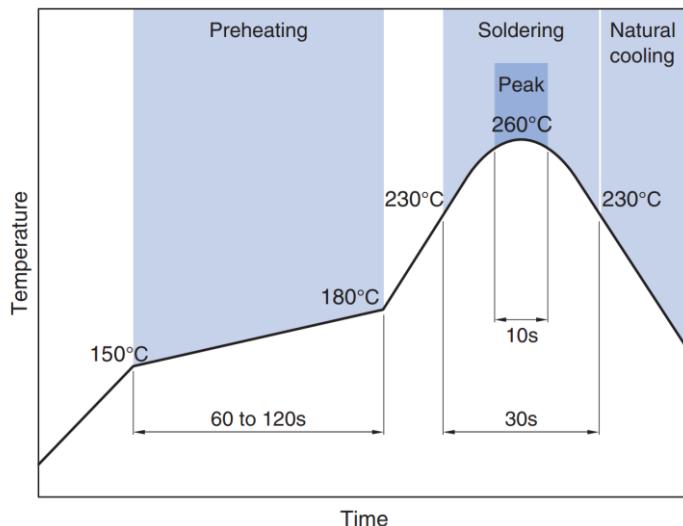


Figure 88 Courbe de température de brasage conseillée de l'inductance (Datasheet VLS3015CX, p.3)

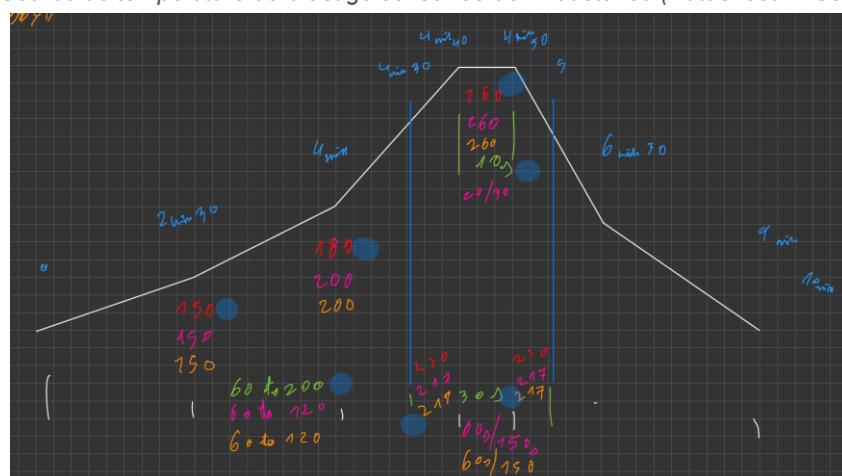


Figure 89 Courbe de température finale de tous les composants

Lors du paramétrage de la courbe de température au four, un premier test à vide a été fait, afin de pouvoir évaluer l'inertie du four. Puis un deuxième test a été fait avec les ajustements pour suivre la courbe théorique le plus possible.

Dans un premier temps j'ai configuré la courbe théorique de chauffe du four et fait un test avec le PCB sans composants.

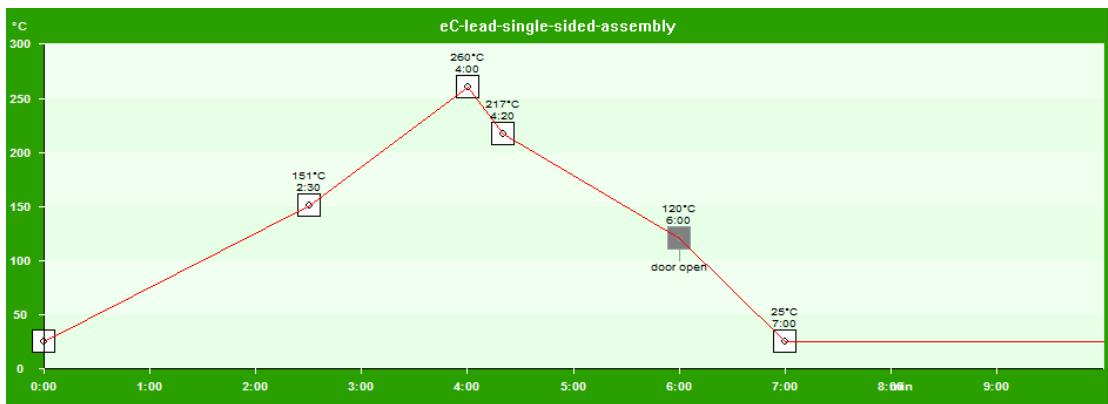


Figure 90 Courbe de chauffe du four théorique initiale

Puis suite aux ajustements dus à l'inertie du four, on obtient cette courbe finale.

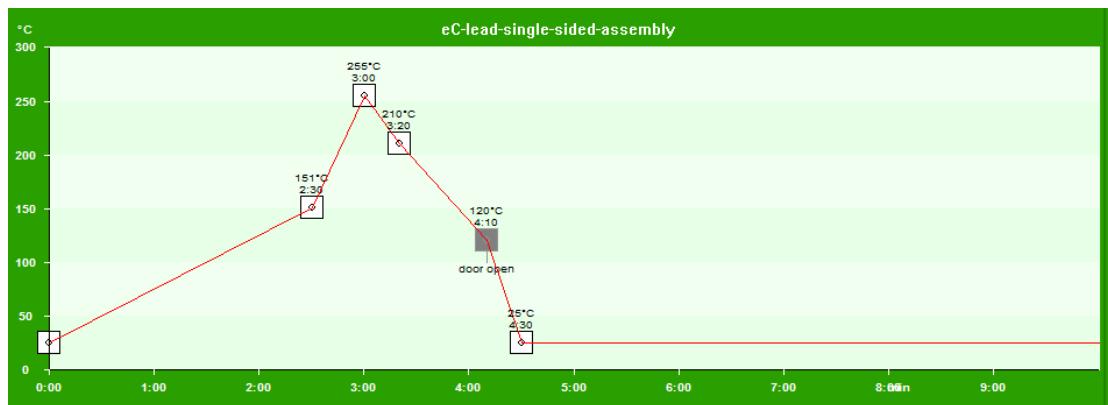


Figure 91 Courbe de chauffe du four théorique adaptée

Avec ce paramétrage, j'ai pu récupérer la courbe réelle lors du brasage des composants sur le PCB.



Figure 92 Courbe de chauffe du four réelle lors du brasage des composants

Suite à sa sortie du four, des ponts étaient présent sur certains composants, ils ont dû être enlevés à la main. Puis j'ai pu monter le reste des composants à la main, sauf le LCD, le PEC12 et les Boutons pousoirs.

2.2.2. Montage du PCB

C'est grâce au stencil qui a été commandé au même temps que le PCB, que j'ai pu déposer de la pâte à braser uniquement sur les pads de composants sur la face bot du PCB. De cette manière on peut directement placer les composants sur la pâte, et ensuite passer le PCB au four pour tous les braser.

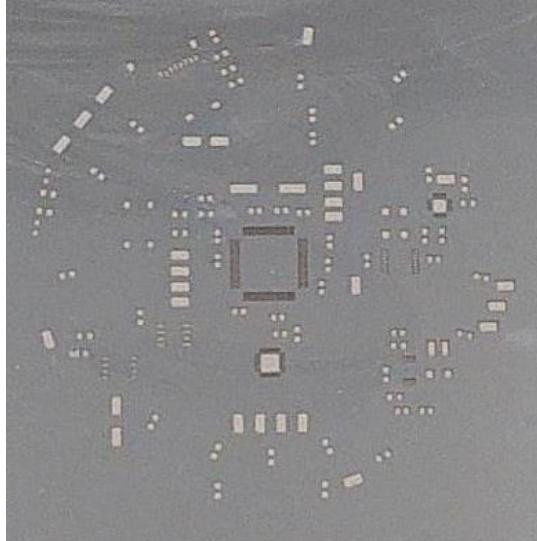
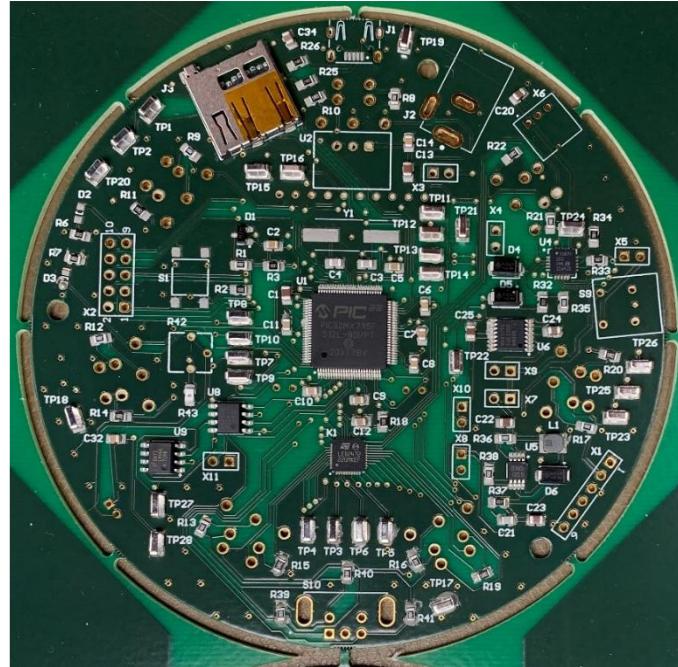


Figure 93 Stencil du panel de 2 PCB BOT



2.3. Boitier

Pour la conception du boitier, dû à la forme circulaire de la carte, le boitier sera lui aussi circulaire. Afin de simplifier sa conception, le boitier est designé en 3D et imprimé grâce à une imprimante 3D. Il a donc été pensé pour, et a été réalisé sur SolidWorks.

En plus du PCB monté, il faudra également positionner l'accumulateur à l'intérieur du boitier, et fixer le haut-parleur.

Afin d'avoir un peu de marge sur la fermeture du boitier, une sorte de glissière a été créée tout autour. Cela permet d'avoir une marge de fermeture de 4mm, tout en faisant en sorte que le boitier pince le PCB.



Figure 97 Encoches pour la fermeture du boitier (position ouverte de 2mm)

Puis afin de maintenir le PCB, ce sont les trois piliers qui feront passer les vis qui fermeront le boitier qui s'en chargeront. Ces piliers se chargeront de supporter le PCB quand les joueurs appuieront sur les boutons.

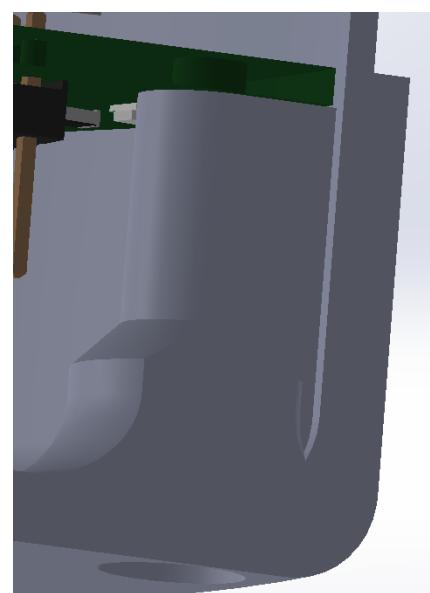
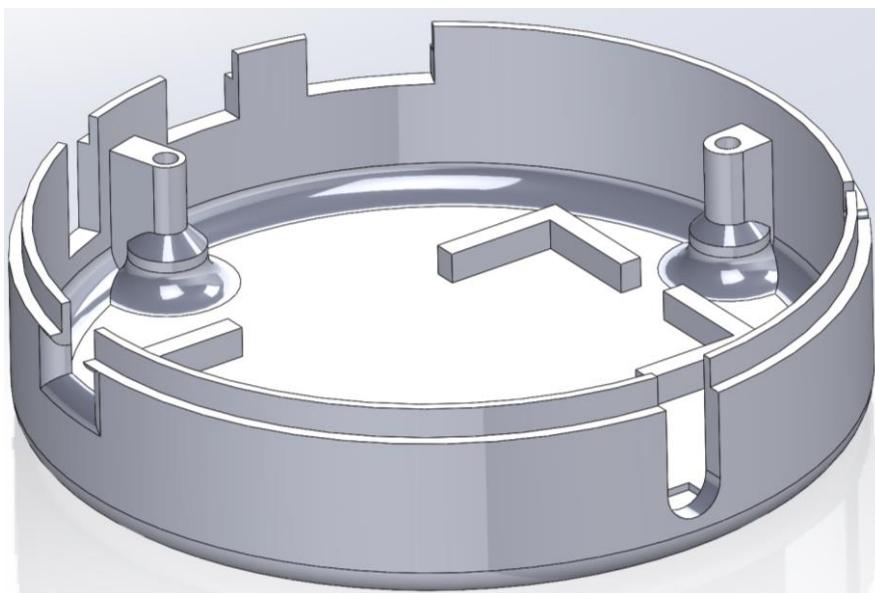


Figure 98 Vue de la partie BOT du boitier avec ses piliers pour le PCB Figure 99 PCB posé sur le pilier du BOT du boitier

En plus d'être posé, le PCB sera aussi coincé entre la partie du haut et la partie du bas du boîtier.

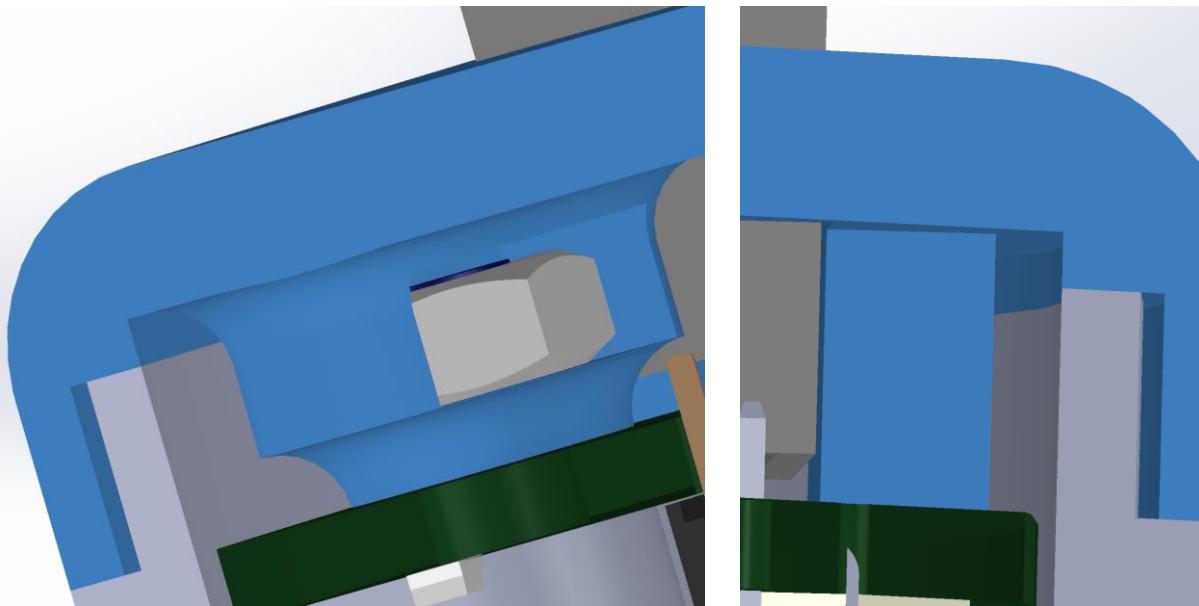


Figure 100 Pilier du TOP du boîtier qui coince le PCB

C'est à l'intérieur de ces petits piliers sur la partie du haut du boîtier que viendront se visser les vis directement dans l'écrou qui est encastré dans la partie du haut du boîtier.

Il y aura donc trois vis M3 qui traverseront tout le boîtier, pour le fermer, mais également pour tenir le PCB en place.

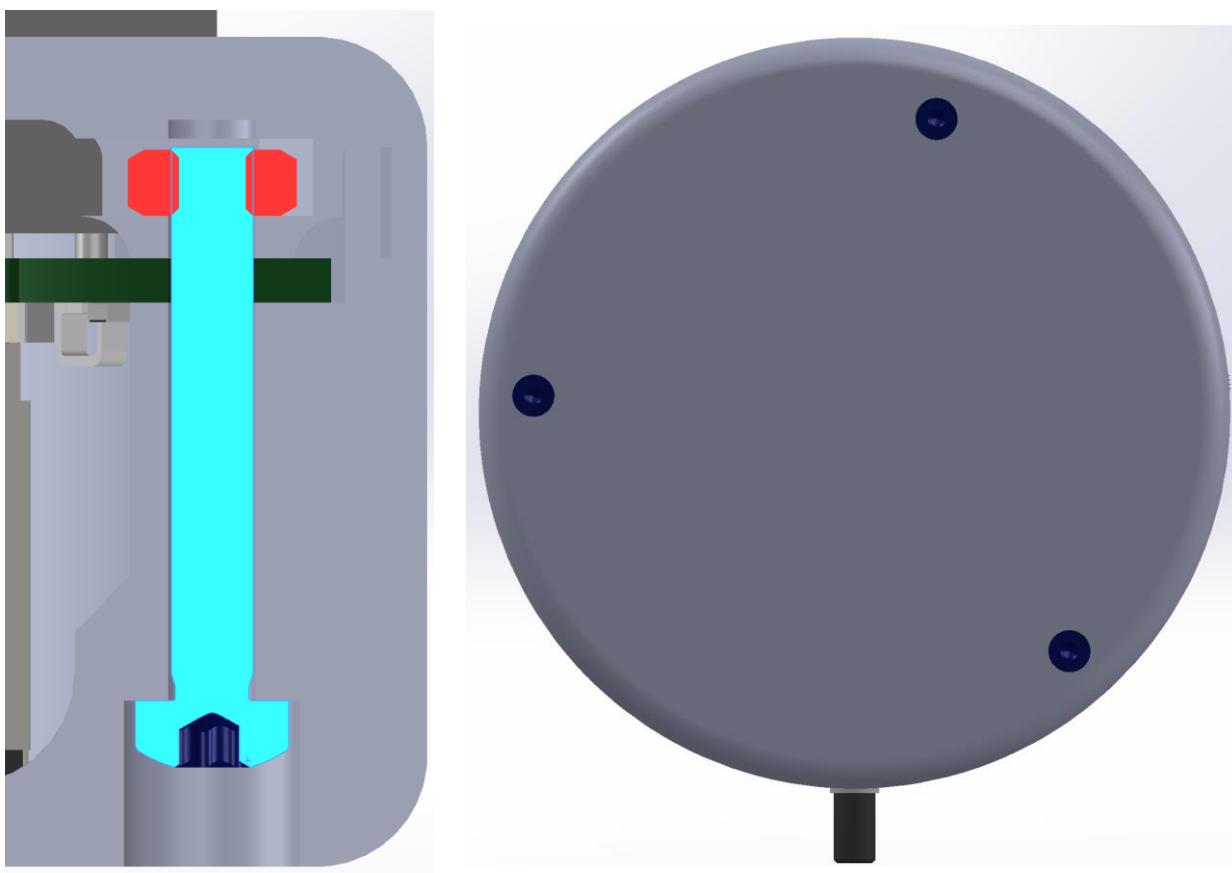


Figure 101 Passage de la Vis M3 puis dans écrous M3

Figure 102 Placement des trois vis de fixation

Encoches pour les boutons, le haut-parleur et le LCD de la partie TOP du boitier.

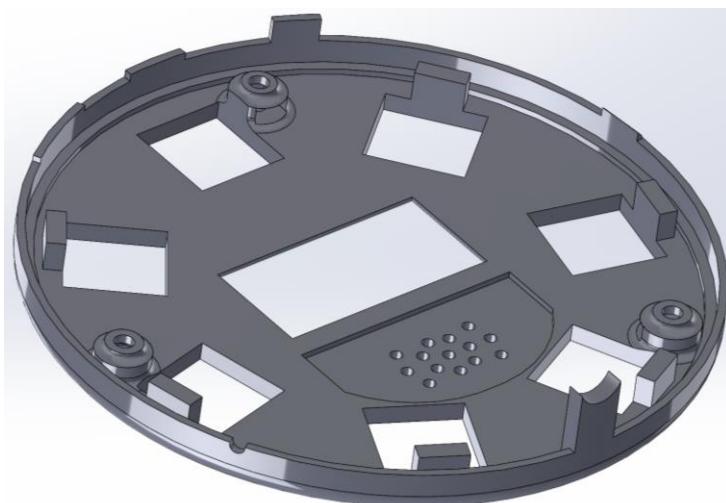


Figure 104 Vue intérieure de la partie TOP du boitier

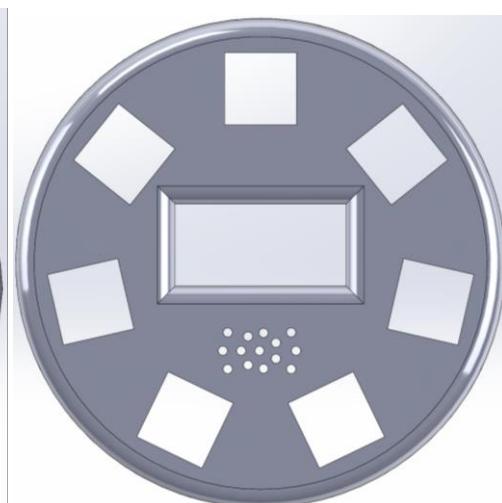


Figure 103 Vue extérieure de la partie TOP du boitier

Des encoches ont également été faites, afin que l'on aille accès aux différents ports de la carte.

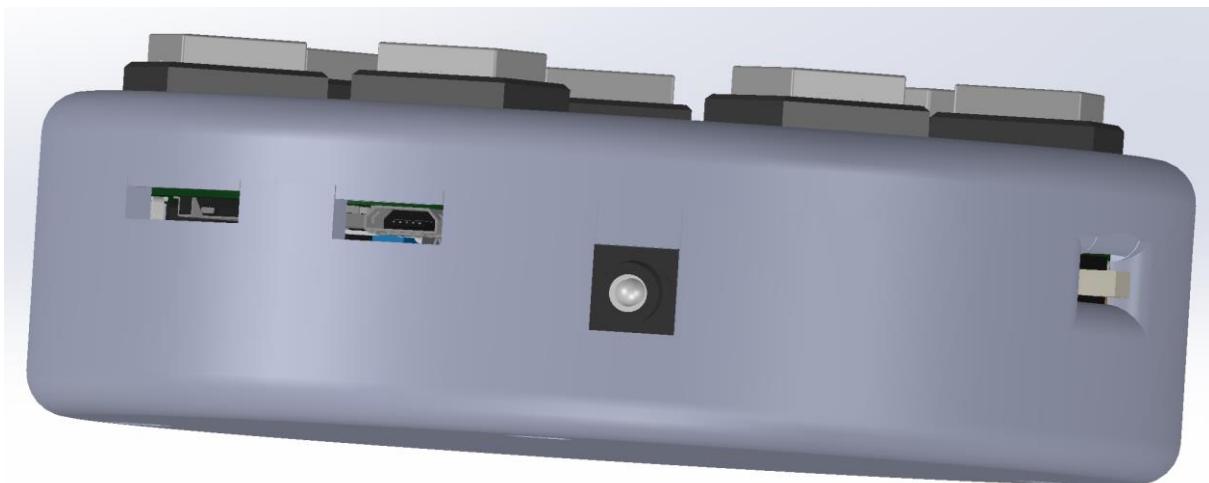


Figure 105 Encoches pour le micro USB, le Jack et la carte micro SD

Des encoches ont été faites dans le Bot et le Top du boitier, afin de maintenir l'accumulateur et le haut-parleur.

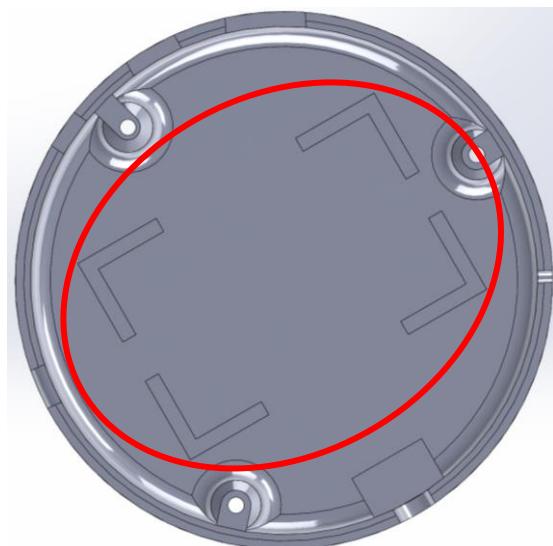


Figure 107 Encoche pour l'accumulateur sur le BOT

Ricardo Crespo

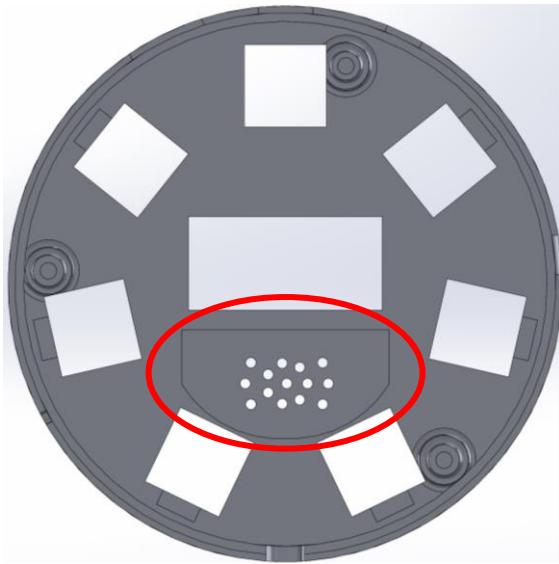


Figure 106 Encoche pour le haut-parleur sur le TOP

Des encoches afin de faire passer le son ont été faites pour le cas où le haut-parleur est placé à son emplacement. Dans le cas où on veut le déporter, une ouverture a été faite pour laisser passer les fils tout en pouvant fermer le boîtier. Une ouverture pour laisser passer l'encodeur a également été faite.

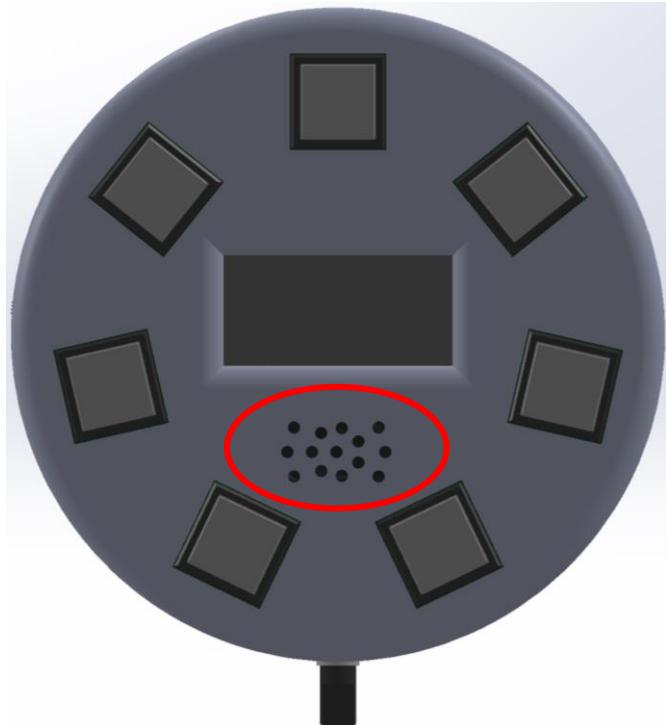


Figure 109 Ouvertures pour laisser passer le son

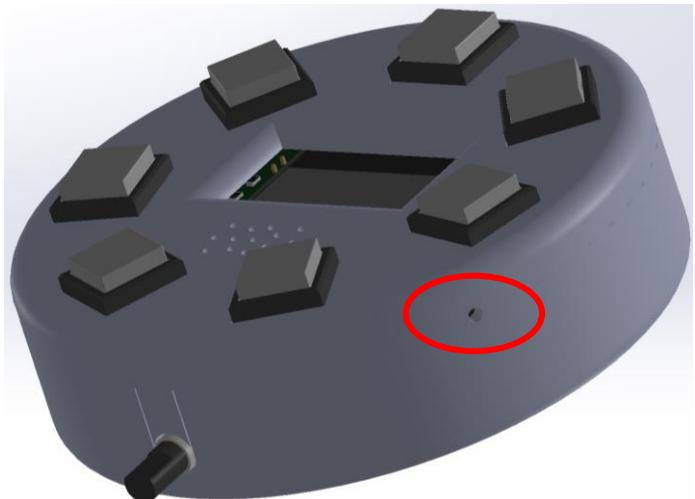


Figure 108 Ouverture pour pouvoir déporter le haut-parleur

Une intégration du boîtier a également été faite sur Altium, afin de contrôler que tout s'emboîte correctement avec tous les éléments, y compris l'accumulateur Li-po.

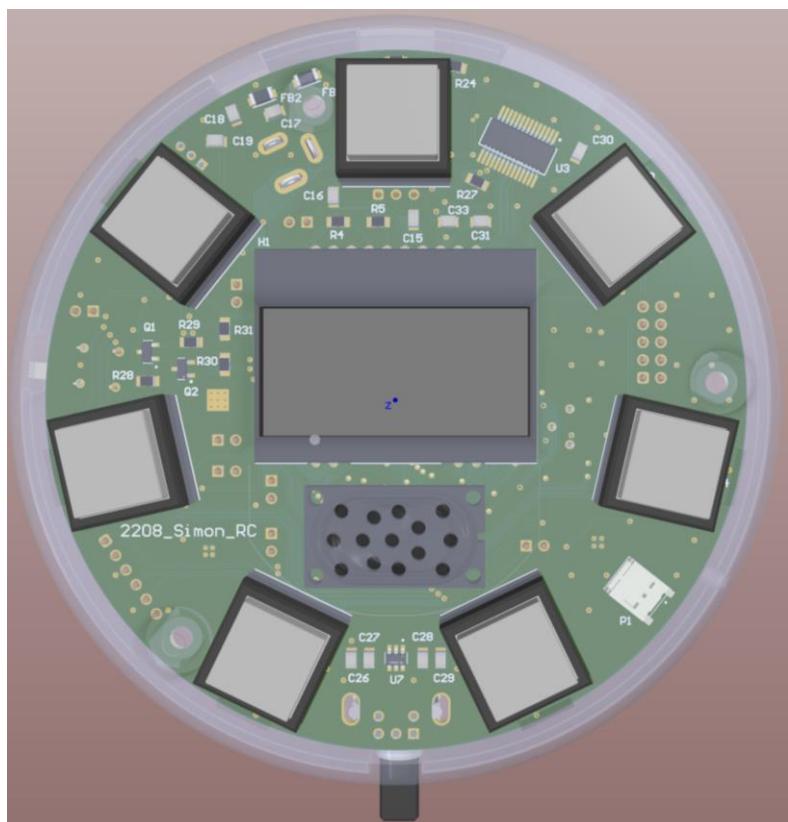


Figure 111 Vue Top de tout l'assemblage sur Altium

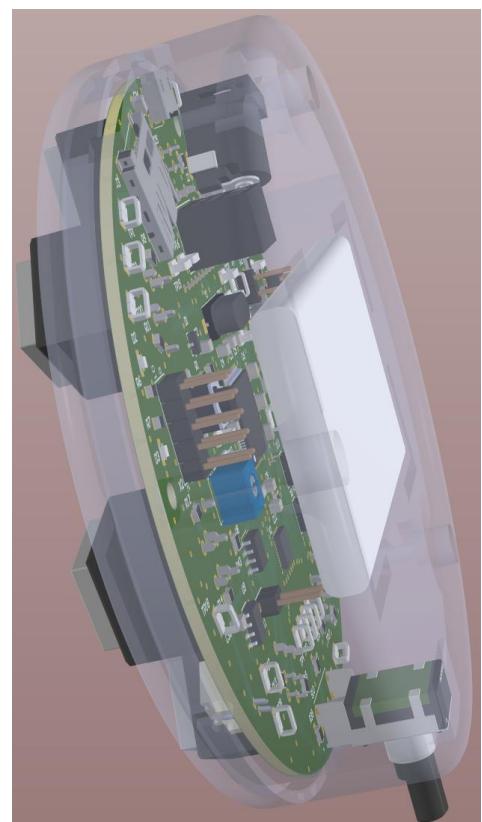


Figure 110 Vue profil de l'assemblage sur Altium

2.4. Modifications

2.4.1. PCB

Lors de la mise en service, le son de l'hautparleur était très faible. J'ai donc mesuré le courant qui le traversait. On tournait aux alentours de 1mA, ce qui n'est pas du tout suffisant pour générer un son assez fort.

J'ai d'abord réglé le trimmer R42, mais le son n'était pas du tout encore assez fort. J'ai donc décidé de brancher le haut-parleur sur le point de test TP28, avant le condensateur de liaison.

C'est là que le son était beaucoup plus fort, et audible normalement.

Le condensateur de liaison qui était là pour enlever la composante continue s'avère d'une par inutile, car avec la composante continue le son est également généré. Mais également d'autre part il nous empêchait de tirer suffisamment de courant.

Il a donc fallu retirer le condensateur C32, et le remplacer par une résistance 0Ω . La modification a été faite sur le PCB, et également dans le projet Altium version B, voir fichier de modifications en annexes.



Figure 113 Condensateur C32 avant modification



Figure 112 Résistance de 0Ω après modification

2.4.2. Boitier

Après réception du premier boitier complètement bien imprimé des deux parties, j'ai pu déceler quelques améliorations qui pouvaient être faites.

En effet il n'était pas possible de connecter le câble Jack de l'ES avec le boitier fermé. De plus certaines ouvertures étaient décalées de quelques millimètres avec les connecteurs.

Les modifications ont essentiellement été faites au niveau des ouvertures de toutes les connexions. En général elles ont toutes été agrandies de quelques millimètres.

Le petit trou pour laisser passer les câbles du haut-parleur a également été agrandi. Pour la structure tenant la batterie, elle a été resserrée.

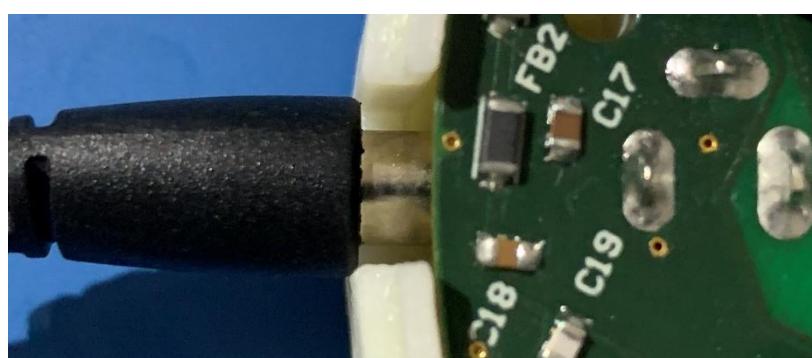


Figure 114 Jack ne peut pas être connecté correctement avec le boîtier 3D

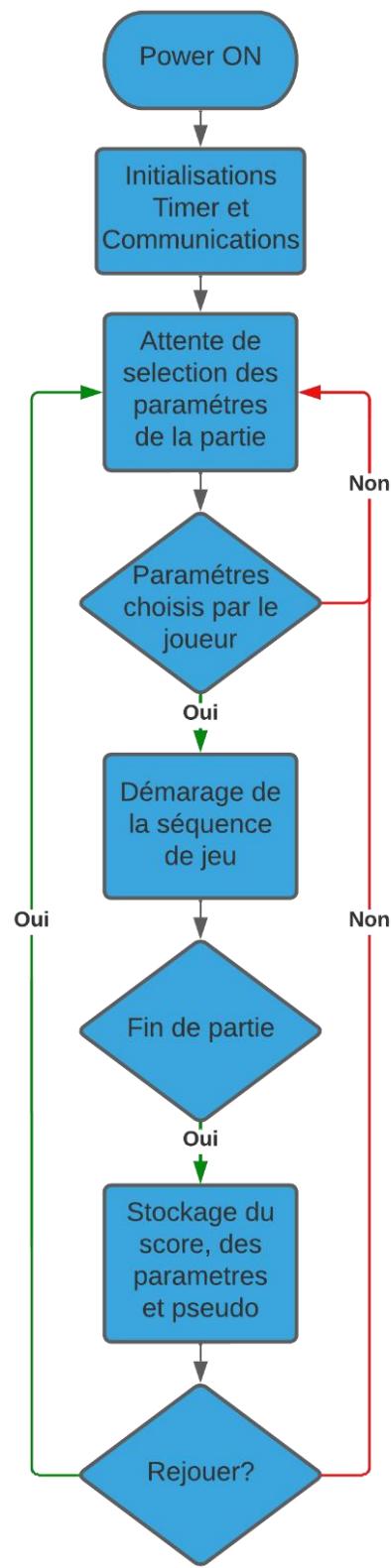
3. Firmware

Dû à la très grande complexité du Firmware réalisé, les flowcharts sont extrêmement simplifiés. Pour plus de précisions, le détail de tous les listings se trouve en annexes.

Pour le développement du firmware j'ai utilisé MPLAB X IDE V5.45, avec la version XC32 V2.50 de son compilateur, et la version d'Harmony V2.06.

3.1. Boucle principale

3.1.1. Flowcharts



3.1.2. Initialisations

3.1.2.1. Timers

3.1.2.1.1. Timer1

Afin de déterminer les valeurs de l'autoreload des Timers, on tien compte que l'on a une fréquence du microcontrôleur de 80MHz. À chaque fois le plus petit prescaler possible a été choisi, afin d'avoir la plus grande valeur de recharge, et donc le plus de précision possible.

Le Timer 1 est le timer principal, qui fait cadencer la machine d'état principale du programme, et qui fait également les mesures de l'accumulateur. Cadencé à 1ms choisie arbitrairement.

$$\text{Autoreload Timer1} = \frac{T}{\text{prescaler} * T_{CLK}} = \frac{1 * 10^{-3}}{8 * 12.5 * 10^{-9}} - 1 = 9999$$

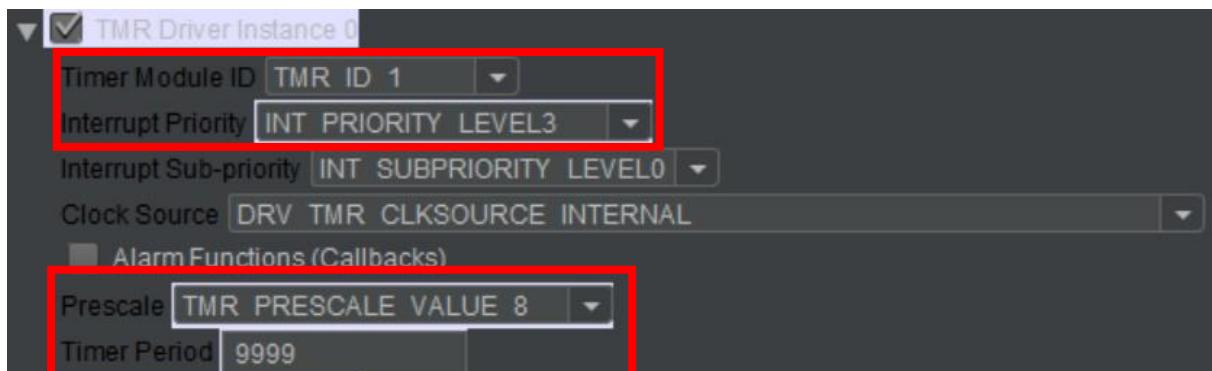


Figure 116 Configuration graphique du Timer 1 à 1kHz

3.1.2.1.2. Timer2

Le Timer 2 est utilisé pour la commande de la Lacth du driver de LEDs. Il est donc réglé à la même fréquence que celle du SPI1, qui est de 100kHz

$$\text{Autoreload Timer1} = \frac{T}{\text{prescaler} * T_{CLK}} = \frac{10 * 10^{-6}}{1 * 12.5 * 10^{-9}} - 1 = 799$$

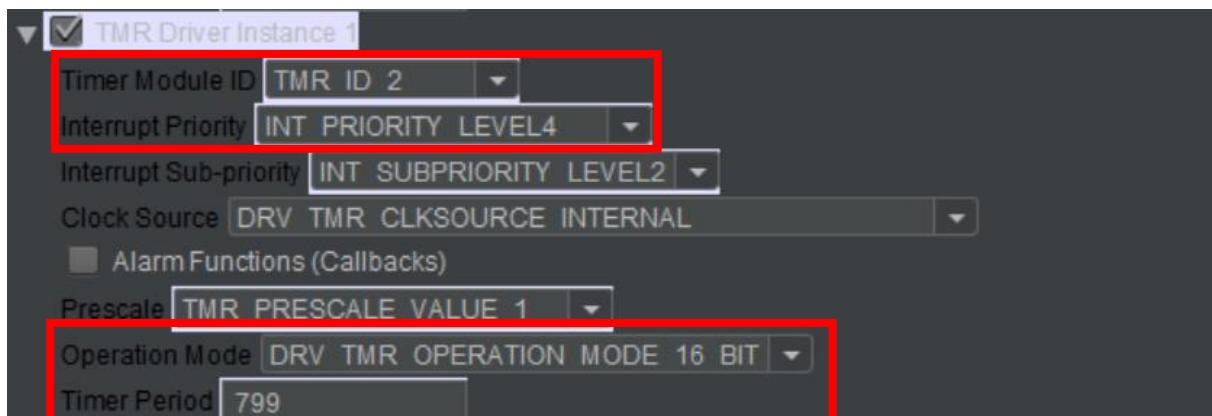


Figure 117 Configuration graphique du Timer 2 à 100kHz

3.1.2.1.3. Timer3

Le Timer 3 sert lui à la génération du signal sonore, via la communication avec le DAC 12bit. Sa valeur par défaut a été mise à 10kHz, mais il va varier entre 12'564Hz et 23'712Hz, afin de générer les 7 notes de la gamme 5 choisie.

$$\text{Autoreload Timer1} = \frac{T}{\text{prescaler} * T_{CLK}} = \frac{100 * 10^{-6}}{1 * 12.5 * 10^{-9}} - 1 = 7999$$

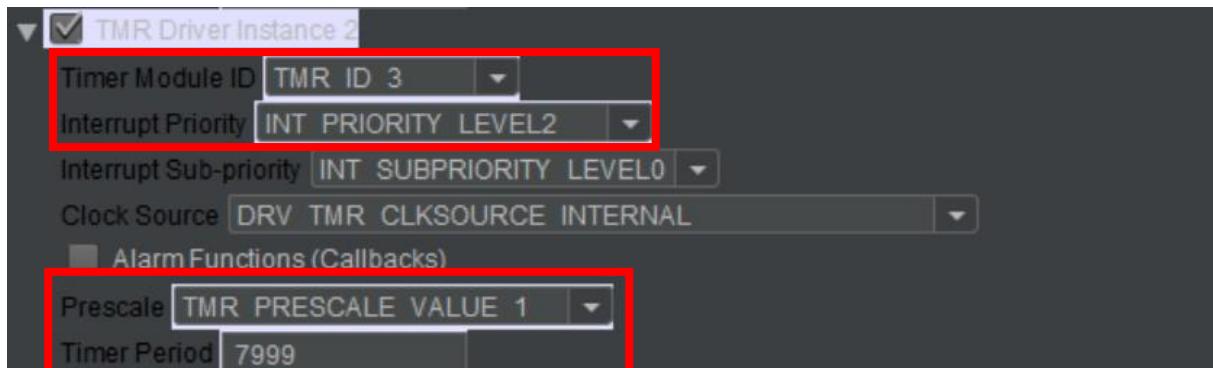


Figure 118 Configuration graphique du Timer 3 à 10kHz

3.1.2.1.4. Timer4

Le Timer 4 sera lui utilisé pour cadencer la séquence de jeu en fonction du mode choisi.

Une valeur a dû être fixée, et c'est une fréquence de 100Hz qui a été choisie.

$$\text{Autoreload Timer1} = \frac{T}{\text{prescaler} * T_{CLK}} = \frac{10 * 10^{-3}}{16 * 12.5 * 10^{-9}} - 1 = 49999$$

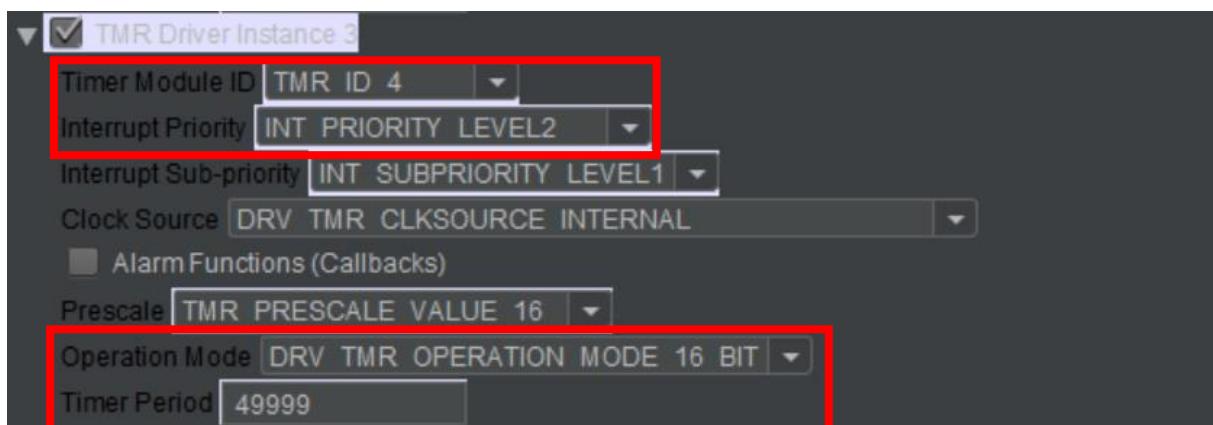


Figure 119 Configuration graphique du Timer 4 à 100Hz

3.1.2.2. Priorité des Timers

Une attention particulière a été apportée à la priorité d'interruption des Timers. Au départ c'est le Timer principal qui était le plus propriétaire, mais le Timer 2 ayant des interruptions plus rapides que le Timer1, c'est lui qui est le plus prioritaire.

Puis c'est le cadencement de la partie qui est avant-dernier prioritaire, et on finit avec la génération de signal.

3.1.2.3. UART

Pour la configuration de l'UART, j'ai choisi de communiquer à la vitesse la plus standard de 9600 bauds. Également avec la trame standard de huit bits de data, pas de parité, un bit de stop et pas d'utilisation du Handshake

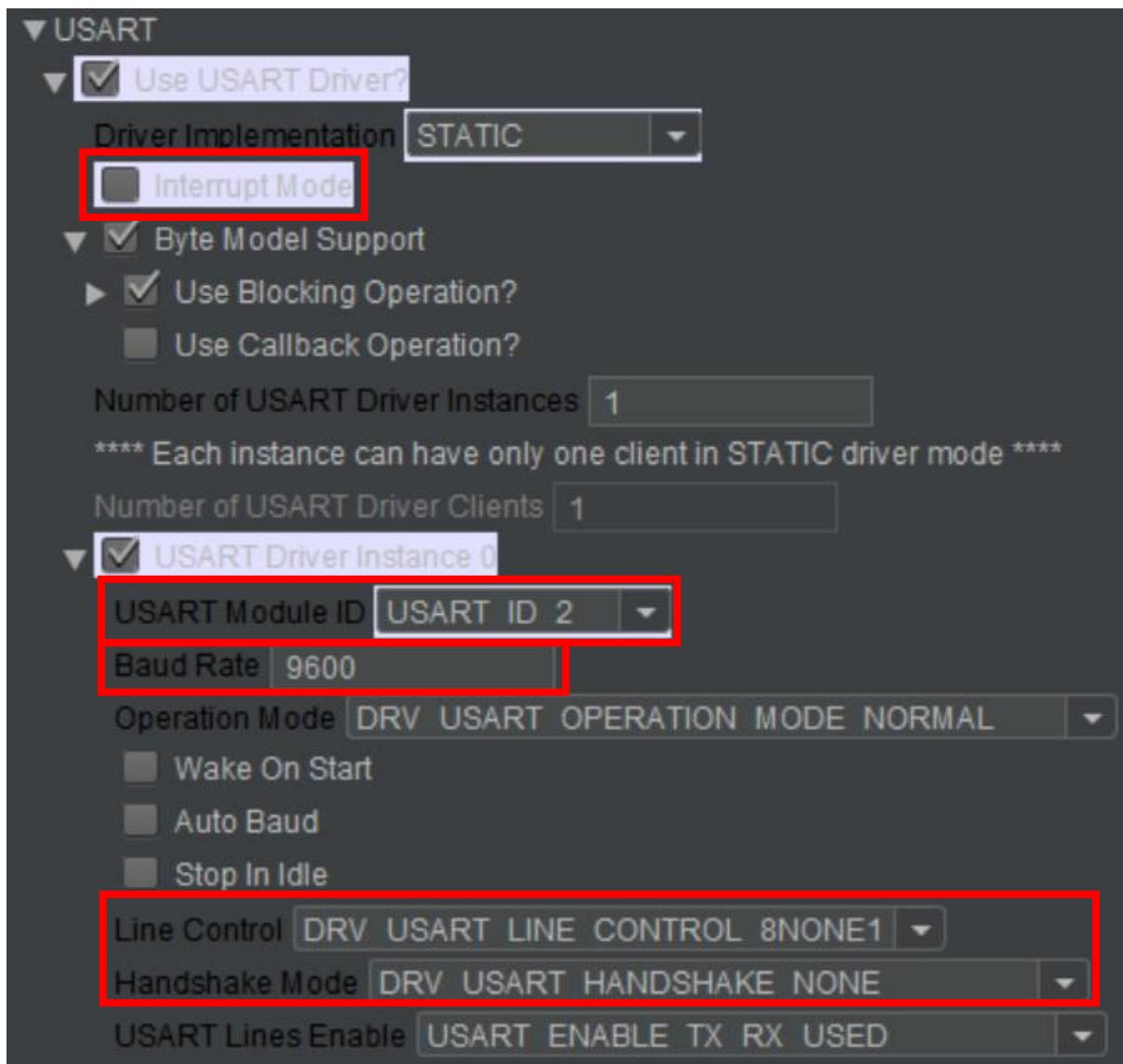


Figure 120 Configuration de l'USART 2

Il ne faut pas oublier d'indiquer que c'est l'UART numéro 2 que l'on utilise.

Vous pouvez également remarquer que la coche « Interrupt Mode » est décochée, c'est normal, car on veut fonctionner par Polling. Ce choix a été fait, car cela été la façon la plus facile à implémenter la partie UART dans le projet.

On pourrait imaginer dans une version future du projet communiquer via interruption.

3.1.2.4. SPI1

Le SPI1 est utilisé pour la commande du Dirver de LEDS. Il a été choisi de communiquer à une grande vitesse, car des changements de couleur pourrait être faits très rapidement, c'est donc 100kHz qui a été choisi.

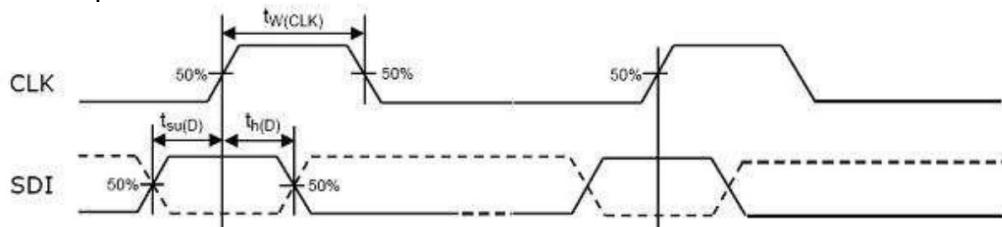


Figure 121 Communication SPI du Driver de LEDs (Datasheet LED2472G, p.15)

Ici on doit principalement remarquer que le clock est au repos à l'état bas, et que les données sont décalées au flanc montant. On devra donc configurer le SPI mode en flanc descendant pour la validation des données, et l'état bas pour le clock au repos.

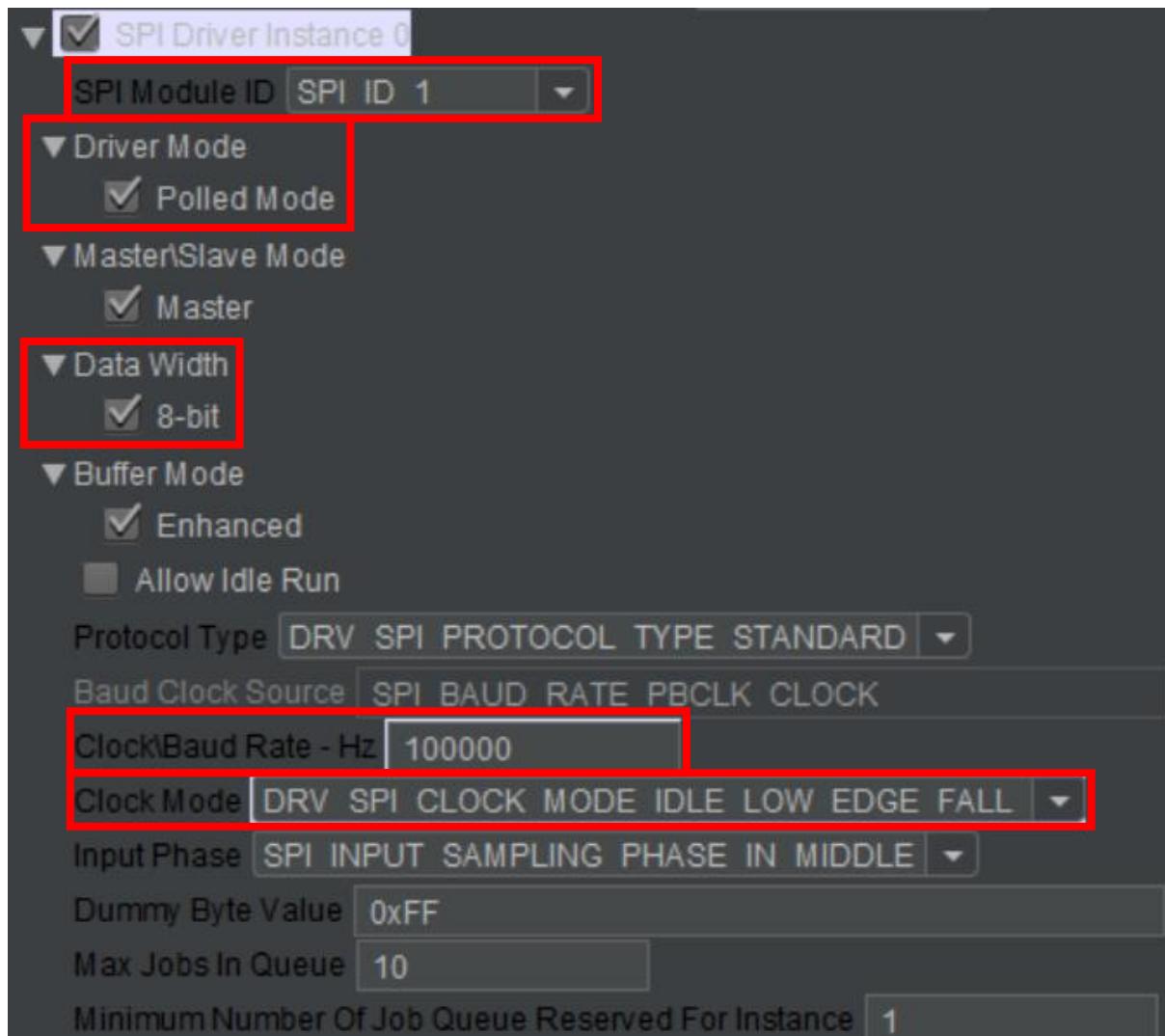


Figure 122 Configuration du SPI 1

J'ai choisi de communiquer avec des mots de 8bits, car aucune configuration de 24bit n'est possible avec le SPI de notre microcontrôleur, uniquement 8bit, 16bit et 32bit.

3.1.2.5. SPI2

Le SPI2 est utilisé pour la commande du DAC 12bit. Il est conseillé dans le datasheet de communiquer jusqu'à 14MHz, c'est donc la fréquence de 14MHz qui a été définie.

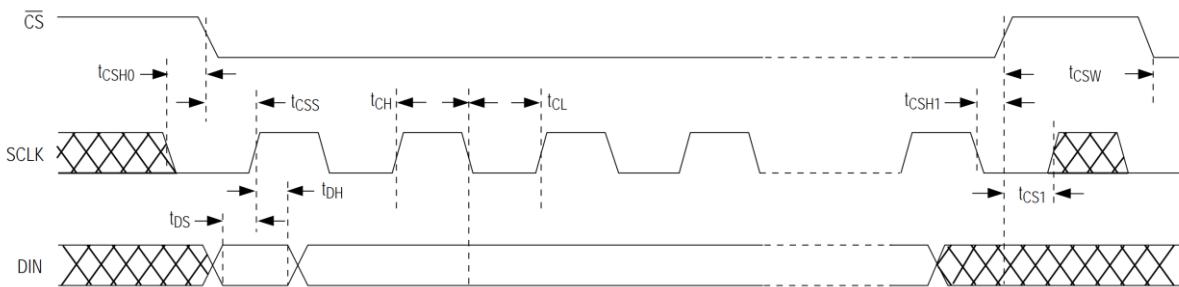


Figure 123 Communication SPI du DAC 12bit ()

Ici on doit principalement remarquer que le clock est au repos à l'état bas, et que les données sont décalées au flanc montant. On devra donc configurer le SPI mode en flanc descendant pour la validation des données, et l'état bas pour le clock au repos.

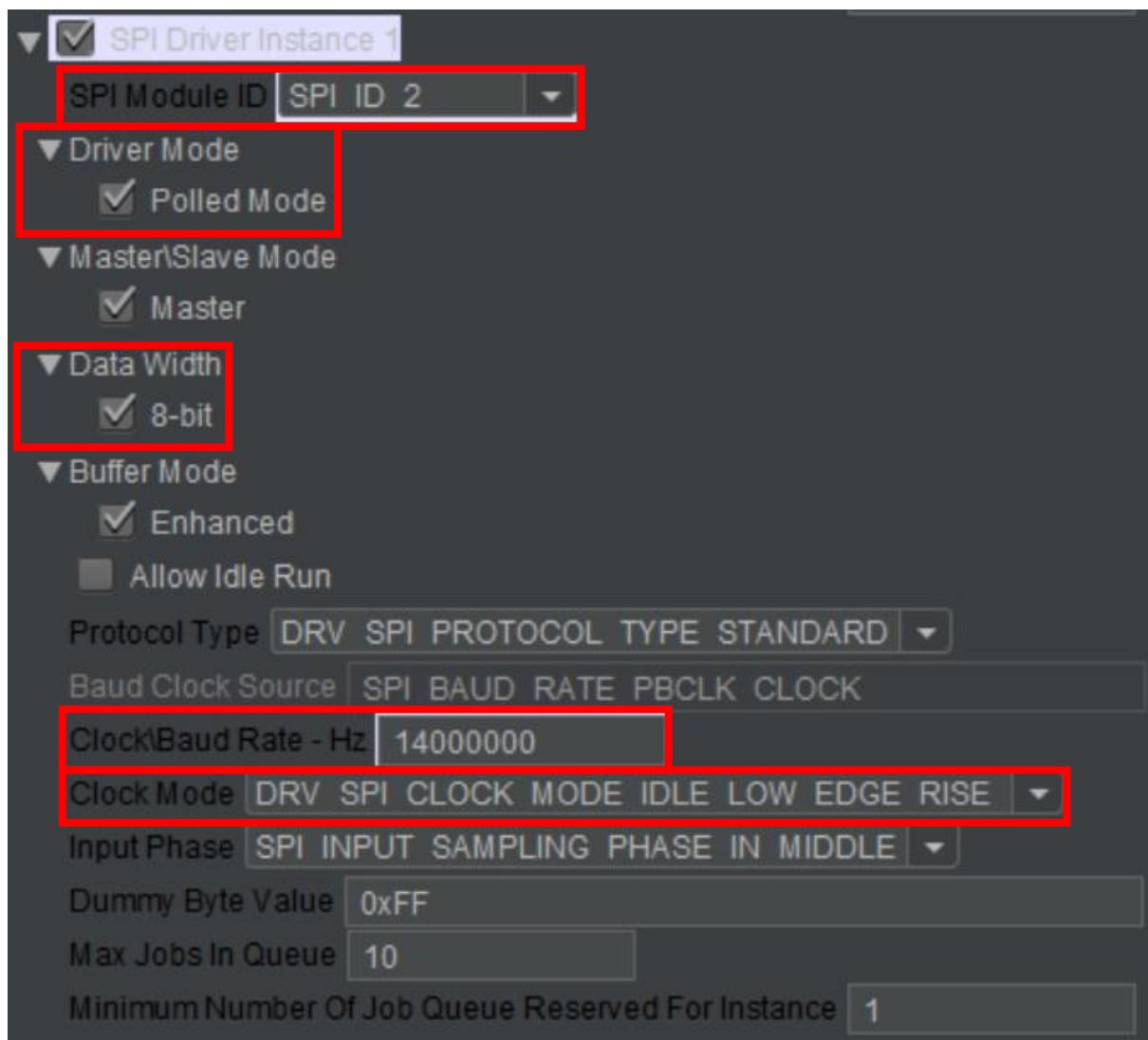


Figure 124 Configuration du SPI 2

On communique avec des mots de 8 bits, car on envoi deux bytes d'affilés pour remplir le registre de 12 bit du DAC.

3.1.2.6. SPI3

La configuration de l'SPI3 n'appas été fait, car l'option de la lecture de la carte SD a été mise en opinel dès le début du projet. Donc toute la partie carte SD n'a pas été mise en service.

3.1.2.7. I2C

La configuration de l'I2C a été faite grâce à la library utilisé au Labo, Mc32_I2cUtilCCS.c / .h. Se sont toutes les configurations par défaut, et la communication en mode fast 400kHz.

3.1.2.8. ADC

Pour l'ADC, il a fallu indiquer la bonne pin AN2 qui est connectée au microcontrôleur pour la mesure de l'accumulateur. Mais également préciser le mode « Alternate Input » pour la mesure de la valeur que sur la pins spécifiée.

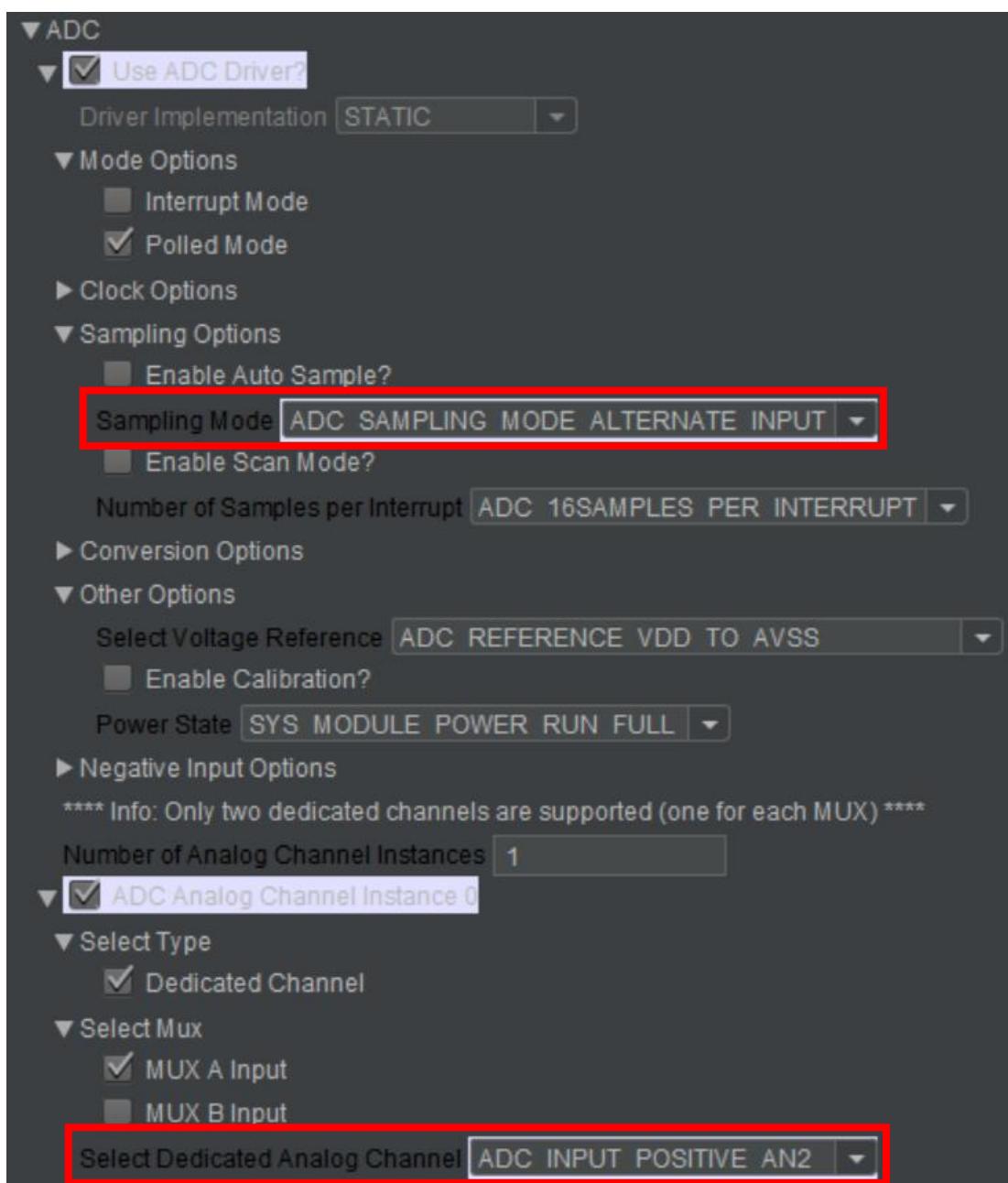


Figure 125 Configuration de l'ADC

3.2. Boucle générale du Jeu et de paramétrage

3.2.1. Flowchart



Figure 126 Flowchart de la boucle général de Jeu et de paramétrage

3.3. Boucle d'exécution du Jeu du Simon

3.3.1. Flowchart

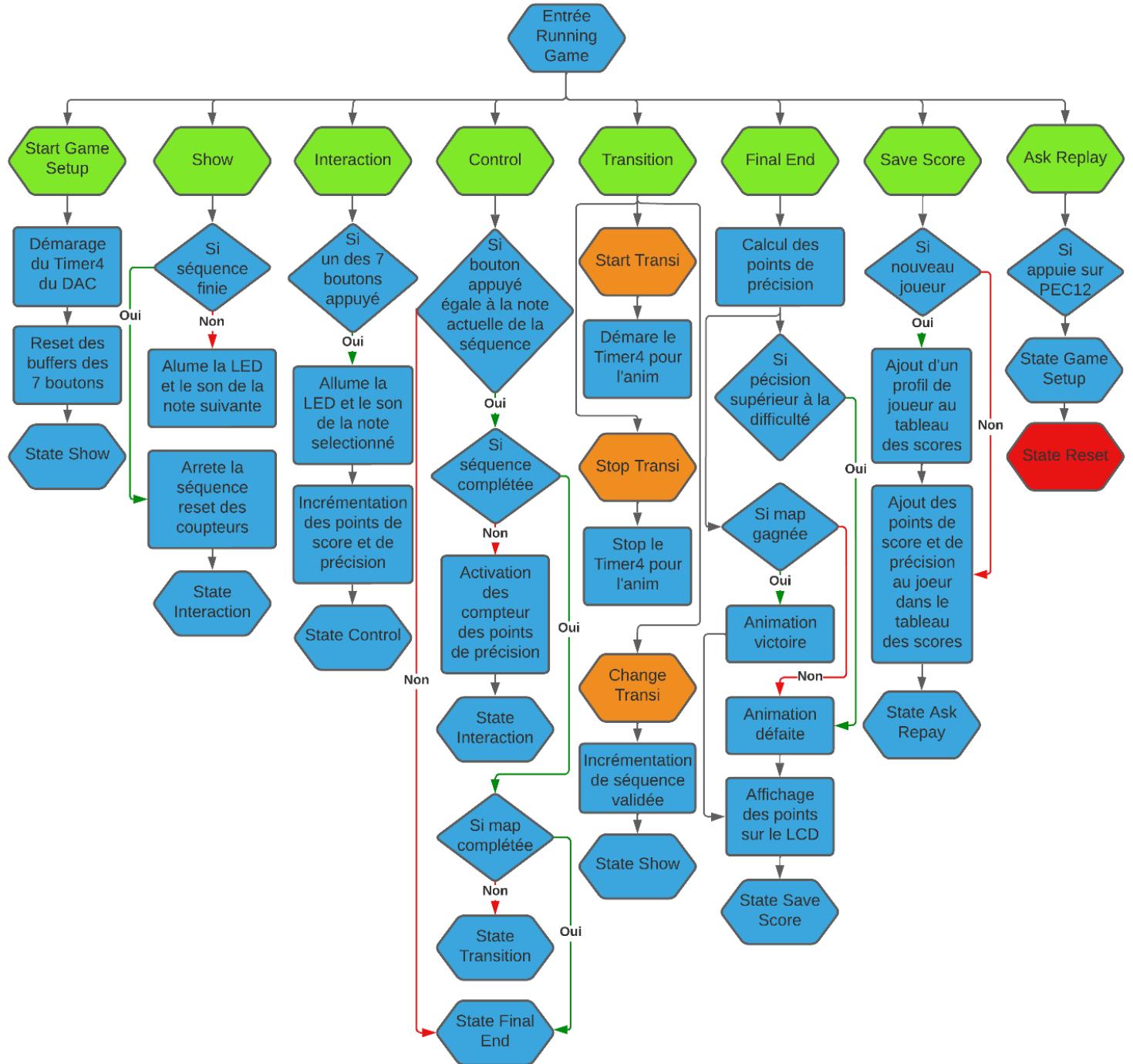


Figure 127 Flowchart de la boucle d'exécution du Jeu du Simon

3.4. Détection des 7 butons et du PEC12

Afin d'avoir une expérience utilisateur optimale, un système d'antirebond a été mis en place. Cela sur les 7 boutons poussoirs de la face avant, mais également sur les signaux du PEC12.

Plus d'informations dans MC32Dbounce.c / .h, gestPec12.c / .h et keys.c / .h

3.5. Driver de LEDs avec SPI1

3.5.1. Latch

Afin de contrôler le driver de LEDs, on doit lui transmettre une adresse et une data.

Afin de lui communiquer dans quel registre on veut écrire, on va devoir faire varier la durée de la Latch.

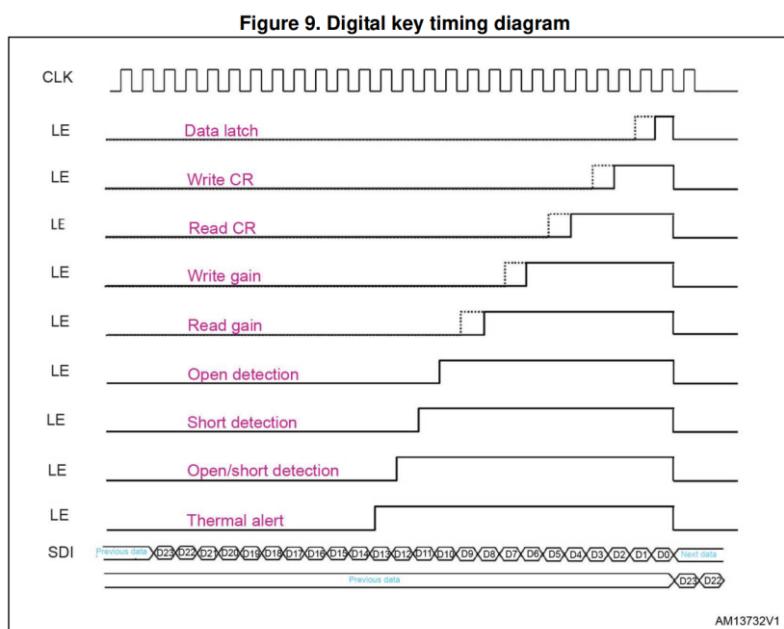


Figure 128 Accès aux différents registres en fonction de la durée de la Latch (Datasheet LED2472G, p.19)

Pour cela, on ne dispose pas de mode SPI avec durée de Latch modifiable, son control a sont été créé de toute pièce.

C'est lorsque l'on fait l'envoi de données que l'on va démarrer simultanément le Timer2. Puis à chaque fois que l'on atteint une certaine valeur de comptage, on active ou désactive la Latch. Afin de pouvoir se caler sur les derniers bits de data envoyés, une valeur d'offset a été calculée, puis ajustée via mesure. Le driver est composé de trois registres de 8bits.

$$\text{nbr Pulses Offset} = \text{nbrBytes} * 8\text{bits} = 3 * 8 = 24$$

Lors de la configuration du Timer2, on a mis donc la même fréquence de communication que la transmission SPI1. Comme le clk du SPI2 n'est pas continu, suite à des essais, le nombre de comptes en plus qu'il faut rajouter pour la période d'init, et entre les clk est de 5. On obtient donc un offset de 29 pulses pour redescendre la Latch.

Donc si l'on veut accéder à un registre précis, il nous suffit de soustraire sa valeur à l'offset pour avoir la bonne durée de la Latch.

$$\text{nbr Pulses} = \text{offsetPulses} - \text{registerConfig} = 29 - 3 = 26$$

Il nous faudra donc par exemple 26 pulses pour accéder au registre de configuration

3.5.2. Initialisation

3.5.2.1. Registre de configuration

Table 8. Configuration register

| BIT | Definition | Attribute read/write | Configuration register function description | | | | | | Default |
|-------------|-------------------------|-------------------------|--|------|-------|-------|-----|--|---------|
| CFG-0 | RED current range | R/W | "0" low current range "1" high current range | | | | | | 0 |
| CFG-1 | GREEN current range | R/W | "0" low current range "1" high current range | | | | | | 0 |
| CFG-2 | BLUE current range | R/W | "0" low current range "1" high current range | | | | | | 0 |
| CFG-3 | RED voltage det. thr. | R/W | "0" LED short-circuit detection threshold 2 V "1" LED short-circuit detection threshold 3 V | | | | | | 0 |
| CFG-4 | GREEN voltage det. thr. | R/W | "0" LED short-circuit detection threshold 2 V "1" LED short-circuit detection threshold 3 V | | | | | | 0 |
| CFG-5 | BLUE voltage det. thr. | R/W | "0" LED short-circuit detection threshold 2 V "1" LED short-circuit detection threshold 3 V | | | | | | 0 |
| CFG-6 | Auto OFF | R/W | "0" device always ON "1" auto power shutdown active (Auto OFF) | | | | | | 0 |
| CFG-7 | SDO delay | R/W | "0" SDO half clock delay disabled "1" SDO half clock delay enabled | | | | | | 0 |
| CFG-8 | Gradual output delay | R/W | "0" gradual outputs delay is applied "1" all channels switch ON and OFF simultaneously | | | | | | 0 |
| CFG-9 | Data flow | R/W | Color data flow management | CFG9 | CFG10 | CFG11 | | | 0 |
| CFG-10 | | | | 0 | 0 | 0 | RGB | | |
| CFG-11 | | | | 0 | 0 | 1 | GBR | | 0 |
| CFG 12 ÷ 23 | Don't care | | | | | | | | |

Figure 129 Tableau du registre de configuration du Driver de LEDs (Datasheet LED2472G, p.20-21)

Ici les 3 premiers bits vont être mis à 0, pour être dans la plage la plus basse de consommation de courant pour les LEDs. Puis les 3 bits suivants ils vont être mis à 1, car on veut que la protection détecter en dessous de notre seuil d'alimentation des LEDs. Puis pour les bits 6 et 7, on les mettra à 0, car on veut que le driveur soit toujours allumé et que l'on aille la durée totale du clock. Puis pour le bit 8, on le mettra à 1, car on veut que toutes les LEDs s'allument ou s'éteignent au même temps. Enfin pour les bits 9, 10 et 11 on les a tous mis à 0, car on laisse l'ordre de base qui est R / G / B.

Ce qui nous donne au final en binaire 0b0000 0001 0011 1000, et donc en hexadécimale 0x38.

3.5.2.2. Registre de gain

Afin de pouvoir régler plus précisément le courant passant pour chacune des couleurs, un registre permet d'y appliquer un gain.

Pour cette configuration, tout a été mis à 0, afin d'avoir le moins de courant dans la plage précédemment choisie.

3.5.3. Commande LEDs

Pour commander les LED, les registres se composent de cette manière après notre configuration.

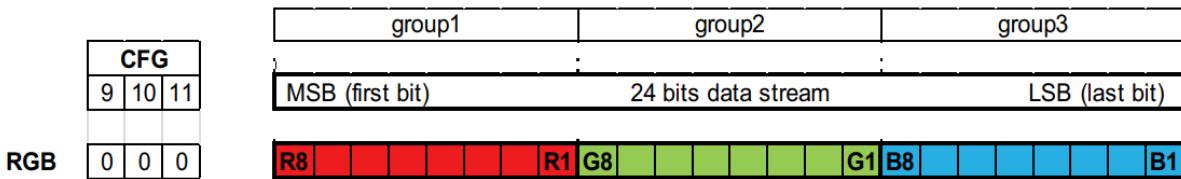


Figure 130 Disposition des registres des LEDs

L'envoi des couleurs doit donc se faire dans un ordre précis, d'abord les LEDs rouges en MSB, puis les LEDs vertes au milieu, et enfin les LEDs bleus en LSB.

```

153 // Envoi de données pour configurer les registres du Driver de LEDs
154 void ConfigRegisterLEDDriver(uint8_t _registerNbr, uint8_t _24to16, uint8_t _16to8, uint8_t _8to0)
155 {
156     // Envoi un byte pour calibrer les pulses
157     SendOneByteRaw(0x00);
158     // Récupérer la valeur du registre à modifier
159     registerNbr = _registerNbr;
160
161     // Démare le Timer 2
162     PLIB_TMR_Start(TMR_ID_2);
163
164     // Envoie du byte MSB
165     SendOneByteRaw(_24to16);
166     // Envoie du byte intermédiaire
167     SendOneByteRaw(_16to8);
168     // Envoie du byte LSB
169     SendOneByteRaw(_8to0);
170 }

```

Figure 131 Fonction de commande des LEDs avec l'ordre d'envoi des données via SPI1 au Driver de LEDs (keys.c)

Cette fois ci, le registre à accéder est le n°2, la durée de la Latch a donc été adaptée automatiquement grâce au travail fait lors de sa commande.

De plus, des couleurs ont été associées à des notes spécifiquement. Mais comme j'ai fait qu'un seul commande simple des LED, les trois couleurs ne peuvent être que soit allumées, soit éteintes.

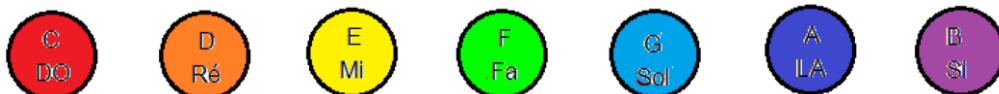


Figure 132 Couleurs en fonction des notes du Cahier des charges (CDC)

Il est possible de faire plus de mélanges de couleurs, mais pour cela il faut communiquer plus rapidement, afin de créer un PWM sur les commandes des LEDs pendant une période donnée.

Des tests ont été faits aux prémisses de la commande des LEDs, mais ils ont très vite été abandonnés, car non nécessaires uniquement pour créer l'orange. Donc la note orange a été remplacée par la couleur blanche. Mais surtout cela nous enlève une grosse charge de calcul au microcontrôleur, car c'est l'action qui est faite le plus souvent par lui (toutes les 10us).

3.6. Notes de musique avec DAC12 et SPI2

Afin de pouvoir générer un son, il nous faut d'abord créer un signal. Afin d'avoir un son le plus chaleureux possible, une sinus a été choisi pour le type de signal.

Il faut donc d'abord définir l'amplitude et l'offset de ce signal.

```
95 // Paramétrages par défaut du signal généré par le DAC 12bit
96 sSound sSoundSet = { .fréquence = 1000, .amplitude = 3100, .offset = 4700};
```

Figure 133 Paramètres par défaut du signal généré (app.c)

```
98 // Mise à jour de la forme du signal, amplitude et offset
99 void UpdateSoundSignal(sSound *pParam)
100 {
101     uint16_t i = 0;                                // Compteur du nombre d'échantillons
102     int32_t newEch = 0;                            // Buffer du nouveau échantillon
103     float radian = 0;                             // Variable de conversion en radians
104
105     // Calcul de tous les échantillons
106     for(i = 0; i < MAX_ECH; i++)
107     {
108         // Conversion d'un échantillon en radian (0 -> 0[RAD], 99 -> 2PI[RAD])
109         radian = sin(map_float(i, MIN_ECH, MAX_ECH, 0, 2 * 3.14));
110         // Adaptation avec l'amplitude et l'offset choisi
111         newEch = ((radian * (pParam->amplitude) + pParam->offset));
112         // Sauvegarde de l'échantillon calculé
113         tbEch[i] = newEch;
114     }
115 }
```

Figure 134 Fonction mettant à jour l'amplitude et l'offset du signal généré (sound.c)

Tout d'abord il y a une conversion des degrés en radians :

$$Rad_{ech} = n^{\circ}Ech * 3.6 * \frac{\pi}{180}$$

Puis une conversion en sinus :

$$Val_{sin} = \sin(Rad_{ech})$$

Finalement on adapte la valeur avec les niveaux de tension souhaités :

$$Val_{ech} = Val_{sin} * amplitude + offset$$

L'amplitude du signal n'a pas pu être de 5V comme initialement imaginé, mais de 3.1V. Cela à cause d'une part de la référence 4.096V du DAC 12bits, et du fait que l'AOP en sortie du DAC n'est pas rail-to-rail. Les valeurs exactes ont donc été définies via des tests par tâtonnement.

```
24 #define MAX_ECH 12 // Echantillon max
```

Figure 135 Nombre d'échantillons par période (sound.h)

Le nombre d'échantillons a été fixé à 12, car suite à des tests en fonction de la gamme choisie, le microcontrôleur n'arrivait plus à calculer tous les échantillons, car il n'avait plus assez de temps pour le faire, avant qui ait à recommencer.

Cette valeur pourrait être augmentée une fois le Firmware complètement fini et optimisé, donc uniquement lors d'une deuxième version du Firmware.

On aurait donc pu utiliser un DAC 8bit pour générer les signaux que l'on génère actuellement.

Pour la fréquence, il a simplement fallu reprendre le calcul de la valeur « Time Période » du Timer 3, et la recalculer dynamiquement à la vitesse en fonction du mode choisi. On va donc communiquer 12 fois plus rapidement que le signal que l'on génère réellement.

```

56 // Mise à jour de la période d'échantillonage
57 void UpdateNote(eNotes _note)
58 {
59     uint16_t valTMR3 = 0; // Buffer du calcul de la valeur du Timer3
60
61     // Stop du timer 3
62     PLIB_TMR_Stop(TMR_ID_3);
63
64     // Calcul de la valeur de comptage du timer 3
65     valTMR3 = (UC_FREQUENCY / ((uint32_t)noteFreq[_note] * MAX_ECH)) - 1;
66
67     // Établissement de la période du timer 3
68     PLIB_TMR_Period16BitSet(TMR_ID_3, valTMR3);
69
70     // Start du timer 3
71     PLIB_TMR_Start(TMR_ID_3);
72 }
```

Figure 136 Mise à jour de la fréquence de la note (sound.c)

Une gamme de notes a dû être choisie, et c'est la cinquième qui a été choisie. Cela, car suite à des tests, c'était celle avec les fréquences les plus basses, tout en restant audibles assez fort.

```

23 // Gamme 5 des notes de musique
24 const uint16_t noteFreq[7] = {1047, 1175, 1319, 1397, 1568, 1760, 1976};
```

Figure 137 Fréquence de toutes les notes de la gamme 5 (sound.c)

Pour modifier la note générée, il suffira donc de changer la fréquence du signal généré.

Pour la cadence du bpm du mode choisi, c'est le Timer 4 qui est utilisé.

```

146 // Si on a atteint un période du bpm actuel
147 if (noteDuration >= modeDiff[gameMode])
148 {
149     // Incrémentation de la note
150     mapTiming++;
151     // Reset du compteur de la période du bpm
152     noteDuration = 0;
153 }
154 // Si non on a pas atteint une période du bpm actuel
155 else
156 {
157     // Incrémentation du compteur de la période du bpm
158     noteDuration++;
159 }
```

Figure 138 Gestion de la période de bpm dans l'interruption du Timer 4 (system_interrup.c)

Les bpm, son le nombre de notes par minute, dans le CDC on nous a demandé respectivement 60bpm, 80bpm et 100bpm pour les trois modes de difficulté.

$$\text{nbr Notes} = \frac{60s}{\text{bpm}} * 100 - 1 = \frac{60}{80} * 100 - 1 = 74$$

```

36 // Valeurs de bpm des différents niveaux de difficulté
37 const uint8_t modeDiff[NMB_MODES] = {99, 74, 59};
```

Figure 139 Valeurs de comptage pour les bpm demandés dans le CDC des charges, 60bpm, 80bpm et 100bpm (game.c)

3.7. LCD avec I2C

Pour l'utilisation de LCD, une initialisation été proposées dans le datasheet.

INITIALIZATION EXAMPLE

| Initialization Example SPI and I2C | | | | | | | | | | | | |
|------------------------------------|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|------|--|
| Command | RS | R/W | DB7 | DB6 | DB5 | DB4 | DB3 | DB2 | DB1 | DB0 | Hex | Remark |
| Function Set | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 1 | 0 | \$3A | 8 bit data length extension Bit RE=1; REV=0 |
| Extended function set | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | \$09 | 4 line display |
| Entry mode set | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | \$06 | bottom view |
| Bias setting | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | \$1E | BS1=1 |
| Function Set | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | \$39 | 8 bit data length extension Bit RE=0; IS=1 |
| Internal OSC | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | \$1B | BS0=1 -> Bias=1/6 |
| Follower control | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 1 | 0 | 0 | \$6C | Devider on and set value |
| Power control | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | \$56 | Booster on and set contrast (DB1=C5, DB0=C4) |
| Contrast Set | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 0 | \$7A | Set contrast (DB3=DB0=C3-C0) |
| Function Set | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 0 | 0 | 0 | \$38 | 8 bit data length extension Bit RE=0; IS=0 |
| Display On | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | \$0F | Display on, cursor on, blink on |

Figure 140 Exemple d'initialisation du LCD (Datasheet DOGS164-A, p.4)

```

39 // Configuration du LCD
40 LCD_WriteCMD(COMMAND_8BIT_4LINES_RE1_IS0);
41 LCD_WriteCMD(COMMAND_4LINES);
42 LCD_WriteCMD(COMMAND_BOTTOM_VIEW);
43 LCD_WriteCMD(COMMAND_BS1_1);
44 LCD_WriteCMD(COMMAND_8BIT_4LINES_RE0_IS1);
45 LCD_WriteCMD(COMMAND_BS0_1);
46 LCD_WriteCMD(COMMAND_FOLLOWER_CONTROL_DOGS164);
47 LCD_WriteCMD(COMMAND_POWER_CONTROL_DOGS164);
48 LCD_WriteCMD(COMMAND_CONTRAST_DEFAULT_DOGS164);
49 LCD_WriteCMD(FUNC_SET3);
50 LCD_WriteCMD(COMMAND_8BIT_4LINES_RE0_IS0);
51 LCD_WriteCMD(COMMAND_DISPLAY | COMMAND_DISPLAY_ON
52 | COMMAND_CURSOR_ON | COMMAND_BLINK_OFF);

```

Figure 141 Ordre d'initialisation du LCD (LCD.c)

Uniquement la dernière ligne a été modifiée, afin de désactiver le clignotement du curseur.

Afin de pouvoir communiquer avec le LCD, on réutilise l'adresse que l'on avait fixée dans la partie design « 0x78 ».

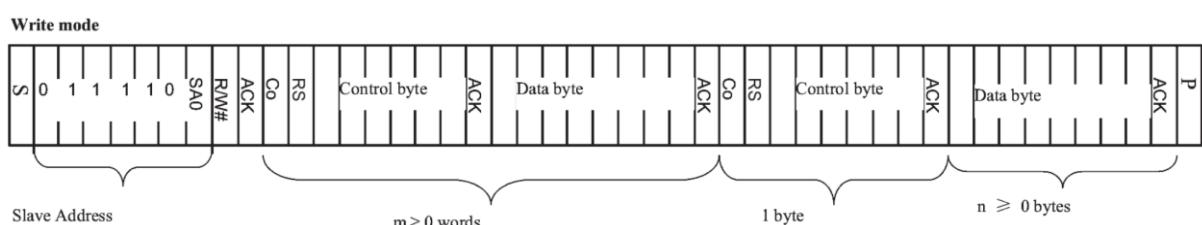


Figure 142 Exemple de trame d'écriture I2C du LCD (Datasheet DOGS164-A, p.7)

Pour effectuer l'écriture, il suffit de suivre l'exemple d'écriture qu'il y a dans son datasheet.

```

45 #define MODE_COMMAND      0x00 // Registre pour les commandes
46 #define MODE_DATA        0x40 // Registre pour les données

```

Figure 143 Deux modes d'accès aux registres de commandes et de données (LCD.h)

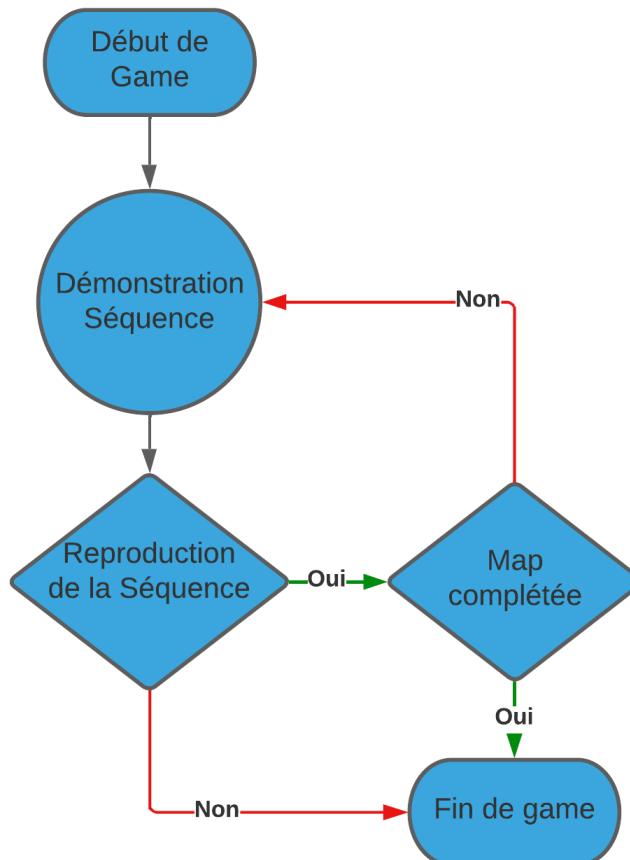
En plus de l'adresse du Driver de LEDs, il faudra préciser si l'on veut faire une modification du registre de configuration, ou alors si on veut écrire quelque chose sur le LCD.

3.8. Gameplay

3.8.1. Game Design

Afin de centraliser toutes les règles du Jeu du Simon, un « Game Design Document » (GDD) a été créé. Il explique le Gameplay, les mécaniques de jeu, et la comptabilisation des différents points.

La boucle de Gameplay principale est donc la suivante :



3.8.2. Points de Score et de Précision

Les points de scores sont comptabilisés à chaque fois que l'on appuie sur un bouton correcte de chaque séquence.

Un système de points de précision est également comptabilisé tout au long de la game. En effet, comme les séquences sont cadencées par un bpm spécifique à son mode de difficulté, les joueurs devront au plus possible reproduire la séquence à la même cadence.

Ces points de précision représentent l'erreur entre le moment où le Player doit appuyer sur le bouton, et le moment où le Player appuie réellement sur le bouton.

Les pourcentages d'erreur doivent être inférieurs à certains niveaux en fonction du mode de difficulté choisi, afin de valider la victoire de la game.

Tous les points sont affichés en fin de game, et également lors de la visualisation des courbes de progression du Player, dans l'application réalisée avec le Software.

3.9. Tableau des scores

Afin de comptabiliser les profils des Players, avec leurs points en fonction des modes choisis pour chacune de leur gamme, un tableau des scores a été créé.

Une structure de données du tableau des scores a été développée :

Cette structure de données a été possible en C, grâce à des « Linked Lists ». Elles utilisent le principe d'une structure récursive. N'ainent pas vu cela lors du programme de cour, toutes les informations ont été tirées d'internet.

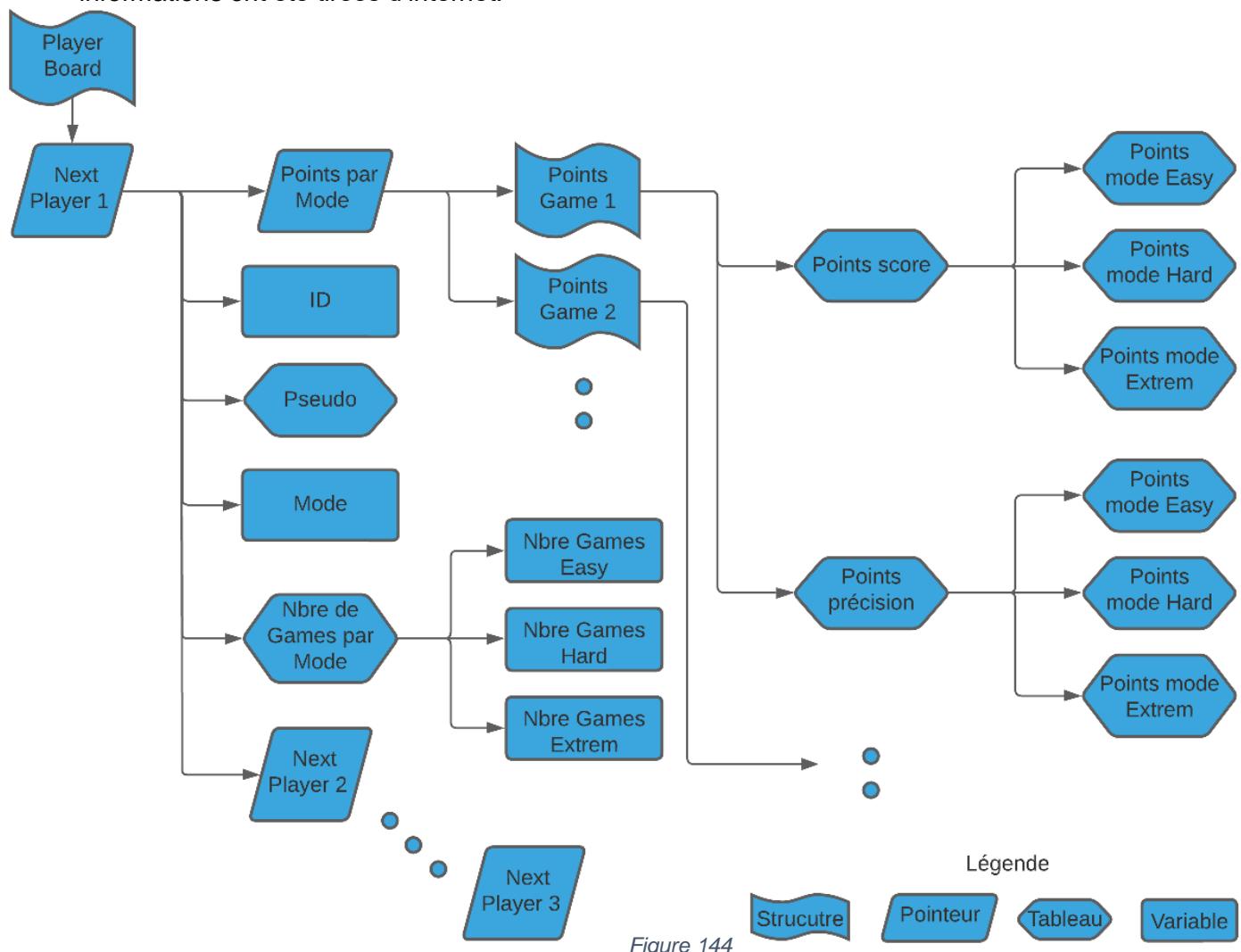


Figure 144

```

33 // Structure du Score
34 typedef struct ScorPoints{
35     uint16_t score[NMB_MODES];
36     uint16_t accurate[NMB_MODES];
37
38     struct ScorPoints *nextScore;
39 }sScorPoints;
40
41 // Structure du Player
42 typedef struct Player{
43     uint16_t playerID;
44     char pseudo[MAX_PSEUDO];
45     uint8_t gameMode;
46     uint16_t gamePerMode[NMB_MODES];
47     sScorPoints *scorePoints;
48
49     struct Player *nextPlayer;
50 }sPlayer;

```

L'utilisation des « Linked Liste » sont là principalement pour optimiser la place que prendra le tableau des scores de manière dynamique. Elle permet également d'avoir une structure lisible par le développeur, et cela facilite le développement.

De cette manière on peut par exemple se balader dans le tableau des scores de Player en Player en passant par le pointeur de structure récursive.

```

30     // Accède à l'adresse du dernier Player du tableau des scores
31     while (currentPlayer->nextPlayer != NULL)
32     {
33         // Passe à l'adresse du prochain Player
34         currentPlayer = currentPlayer->nextPlayer;
35     }
36
37     // Crée le nouveau Player en allouant de la mémoire
38     currentPlayer->nextPlayer = (sPlayer*) malloc(sizeof(sPlayer));

```

Figure 146 Ajout d'un nouveau joueur à la fin du tableau (scoreBoard.c)

Une fois l'adresse du dernier Player récupérée, on peut en ajouter un nouveau dynamiquement au tableau des scores, en allouant de la mémoire pour lui.

La même manière a été choisie pour la comptabilisation des scores.

```

89     // Si on a plus qu'une game
90     if((currentPlayer->gamePerMode[currentPlayer->gameMode] > 1))
91     {
92         // Crée un nouveau score en allouant de la mémoire
93         currentPlayer->scorePoints[(currentPlayer->gamePerMode[currentPlayer->gameMode] - 2)].nextScore =
94             (sScorPoints*) malloc(sizeof(sScorPoints));
95     }
96
97     // Ajoute les points de score dans le tableau des scores du player actuel dans la game actuel
98     currentPlayer->scorePoints[(currentPlayer->gamePerMode[currentPlayer->gameMode] - 1)].score[currentPlayer->gameMode] = _score;
99     // Ajoute les points de score dans le tableau des scores du player actuel dans la game actuel
currentPlayer->scorePoints[(currentPlayer->gamePerMode[currentPlayer->gameMode] - 1)].accurate[currentPlayer->gameMode] = _accurate;

```

Figure 147 Ajout d'un nouveau score au Player actuel dans le mode actuel (scoreBoard.c)

Ci-dessous vous avez un aperçu du tableau des scores vu lors du debugg.

| | Name | Type | Address | Value | Char | Decimal |
|-------------|--------------|------------|------------|------------|------------|------------|
| playerBoard | sPlayer* | 0x0000210 | 0xA0001A10 | 0x0000 | " | 2684361232 |
| playerID | uint16_t | 0xA0001A10 | 0x0000 | 0 | | |
| pseudo | char[3] | 0xA0001A12 | "\0\0\0" | "\0\0\0" | | |
| gameMode | uint8_t | 0xA0001A15 | NUL; 0x0 | NUL; 0x0 | 0 | |
| gamePerMode | uint16_t[3] | 0xA0001A16 | ... | ... | | |
| scorePoints | sScorPoints* | 0xA0001A1C | 0x00000000 | " | 0 | |
| nextPlayer | Player* | 0xA0001A20 | 0xA0001A60 | '\u001a' | 2684361312 | |
| playerID | uint16_t | 0xA0001A60 | 0x0001 | '\u0001' | 1 | |
| pseudo | char[3] | 0xA0001A62 | "ABC" | "ABC" | | |
| gameMode | uint8_t | 0xA0001A65 | STX; 0x2 | STX; 0x2 | 2 | |
| gamePerMode | uint16_t[3] | 0xA0001A66 | ... | ... | | |
| scorePoints | sScorPoints* | 0xA0001A6C | 0xA0001AC0 | '\u001a\0' | 2684361408 | |
| score | uint16_t[3] | 0xA0001AC0 | 0x0000 | " | 0 | |
| score[0] | uint16_t | 0xA0001AC0 | 0x0000 | " | 0 | |
| score[1] | uint16_t | 0xA0001AC2 | 0x0000 | " | 0 | |
| score[2] | uint16_t | 0xA0001AC4 | 0x0006 | '\u0006' | 6 | |
| accurate | uint16_t[3] | 0xA0001AC6 | 0x0000 | " | 0 | |
| accurate[0] | uint16_t | 0xA0001AC6 | 0x0000 | " | 0 | |
| accurate[1] | uint16_t | 0xA0001AC8 | 0x0000 | " | 0 | |
| accurate[2] | uint16_t | 0xA0001ACA | 0x0004 | '\u0004' | 4 | |
| nextScore | ScorPoints* | 0xA0001ACC | 0x00000000 | " | 0 | |
| nextPlayer | Player* | 0xA0001AD8 | 0xA0001AD8 | '\u001a\0' | 2684361432 | |

Figure 148 Tableau des scores vu depuis el debugger

Dans ce cas on a un Player avec l'ID 1, qui a fait une game en mode Extreme, et qui a récupéré 6 points de score, et qui a fait 4% d'erreur. Le détail du remplissage du tableau des scores est similaire à celui fait dans le Software, plus de détails dans scoreBoard.c / .h.

3.10. Sauvegarde

Afin de perdurer le tableau des scores, une sauvegarde est faite à chaque nouvel ajout de score dans le tableau des scores. Pour cela, dans un premier j'ai voulu directement sauvegarder la structure en entier du tableau des scores, mais il m'était impossible de sauvegarder toutes les dépendances de Player et de Score dans l'ordre.

J'ai donc opté par la réutilisation de la fonction de cryptage qui sera utilisé pour l'envoi du tableau des scores au Software.

Globalement la trame est composée de caractères spéciaux séparant les différentes informations de chaque joueur dans le tableau des scores. Vous trouverez le détail des trames dans la partie Software.

Avant de sauvegarder, il faut récupérer la taille totale du tableau des scores

```

123 // Récupération du nombre de Player et du nombre de scores total
124 GetTotalSizeOfLinkedList(&sizePlayer, &sizeScore);
125
126 // Taille total maximal du tableau des scores actuel
127 totalsize = (sizeof(sPlayer) * sizePlayer) + (sizeof(sScorPoints) * sizeScore);
128
129 // Alocation de la mémoire dynamiquement en fonction de la taille du tableau des scores max
130 bufferBoardSave = malloc(sizeof(char) * totalsize);
131
132 // Reset du tableau
133 memset(bufferBoardSave, 0, totalsize);
134
135 // Conversion de la structure du tableau des scores en trame chiffrée et récupération de la taille exacte
136 totalsize = EncryptionScoreBoard(bufferBoardSave, 2);

```

Figure 149 Récupération de la taille du tableau des scores dans la mémoire (scoreBoard.c)

Puis une fois chiffré, on y ajoute la clé magique et la taille totale dans les deux premières cases du tableau de sauvegarde, et on sauvegarde sa totalité dans la flash du uC.

```

138 // Stock la clé magique pour confirmer qu'il y a eu une sauvegarde
139 bufferBoardSave[0] = MAGIC_SAVE;
140 // Taille total en bytes de la structure
141 bufferBoardSave[1] = totalsize;
142
143 // Sauvegarde du tableau des scores dans la mémoire flash du uC
144 NVM_WriteBlock((uint32_t*)bufferBoardSave, totalsize);
145

```

Figure 150 Chiffrement du tableau des scores, ajout de la clé magique et de la taille totale exacte (scoreBoard.c)

Afin de pouvoir utiliser la fonction « malloc » qui alloue de la mémoire dynamiquement en fonction de la taille qu'on lui donne, une taille de la « Heap » de mémoire a dû être définie.

Suite à des tests, une structure de Player pèse 24 bytes, et une structure de scores pèse 12 bytes. On dispos d'une taille maximale d'environ 500kB, c'est alors que l'on a décidé lors d'un meeting de fier à 20% la taille de la Heap, donc à 100kB.

C'est lorsque l'on va dans les propriétés du projet MPLAB, que l'on a accès à l'onglet « xc32-Id ».



Figure 151 Paramétrage de la mémoire du microcontrôleur

Puis c'est dans la partie « Heap » que l'on peut ajouter la valeur que l'on a définie. Ce réglage est mis à 0 à la création du projet, il faut donc obligatoirement le régler.



Figure 152 Paramétrage de la Heap

L'estimation du nombre de joueurs peut énormément différer en fonction du nombre de games qu'ils ont fait, et dans quels modes. J'ai donc fait une estimation du nombre de joueurs max avec une seule game, et celui avec 3 games chacun.

$$nbr\ Players = \frac{taille\ de\ la\ Heap}{Taille\ 1\ Player + Taille\ 1\ score} = \frac{100 * 10^3}{24 + 12} = 2777\ Players$$

$$nbr\ Players = \frac{taille\ de\ la\ Heap}{Taille\ 1\ Player + Taille\ 3\ score} = \frac{100 * 10^3}{24 + 36} = 1666\ Players$$

Les craintes de manque de mémoire lors de la phase de design s'estompent à présent. Avec un nombre supérieur à mille Players, on dépasse largement la centaine de Player réel que le Simon aura lors de sa durée de vie.

Il n'est donc plus du tout envisageable d'éventuellement stocker le tableau des scores dans la carte SD.

Puis une fois que l'on redémarre le Simon, une lecture de la mémoire est faite.

```

155 // Lecture des deux premières cases de la mémoire
156 NVM_ReadBlock((uint32_t*)&nbrSize, sizeof(nbrSize));
157
158 // Control de la clé magic indicant un ancienne sauvegarde
159 if(nbrSize[0] == MAGIC_SAVE)
160 {
161     // Allocation de la mémoire dynamique en fonction de la taille du tableau des scores
162     bufferScoreBoardRead = malloc(sizeof(char) * (uint16_t)nbrSize[1]);
163
164     // Lecture de la totalité du tableau des scores
165     NVM_ReadBlock((uint32_t*)bufferScoreBoardRead, (uint16_t)nbrSize[1]);
166
167     // Déchiffrement du tableau des scores
168     FillNewDataOfSaveRead(bufferScoreBoardRead);
169 }

```

Figure 153 Lecture du tableau des scores précédemment sauvegardé (scoreBoard.c)

On lit dans un premier temps que les deux premières cases sauvegardées, car on vérifie que l'on retrouve la clé magic qui indique qu'il y a eu une sauvegarde. Puis si c'est le cas, on peut par la suite allouer de la mémoire dynamiquement pour la taille totale du tableau des scores. Ensuite il nous reste plus qu'à la lire en entier, et à le déchiffrer.

| bufferBoardSave | char[40] | 0xA001FF44 | "plu0012#1\$ABC@1%1,6.15;l 0 0 0 -1610606352 |
|----------------------|----------|------------|---|
| *bufferBoardSave[0] | char | 0xA0001BF0 | 'p'; 0x70 112 |
| *bufferBoardSave[1] | char | 0xA0001BF1 | DC2; 0x12 18 |
| *bufferBoardSave[2] | char | 0xA0001BF2 | '#'; 0x23 35 |
| *bufferBoardSave[3] | char | 0xA0001BF3 | '1'; 0x31 49 |
| *bufferBoardSave[4] | char | 0xA0001BF4 | '\$'; 0x24 36 |
| *bufferBoardSave[5] | char | 0xA0001BF5 | 'A'; 0x41 65 |
| *bufferBoardSave[6] | char | 0xA0001BF6 | 'B'; 0x42 66 |
| *bufferBoardSave[7] | char | 0xA0001BF7 | 'C'; 0x43 67 |
| *bufferBoardSave[8] | char | 0xA0001BF8 | '@'; 0x40 64 |
| *bufferBoardSave[9] | char | 0xA0001BF9 | '1'; 0x31 49 |
| *bufferBoardSave[10] | char | 0xA0001BFA | '%'; 0x25 37 |
| *bufferBoardSave[11] | char | 0xA0001BFB | '1'; 0x31 49 |
| *bufferBoardSave[12] | char | 0xA0001BFC | '; 0x2c 44 |
| *bufferBoardSave[13] | char | 0xA0001BFD | '6'; 0x36 54 |
| *bufferBoardSave[14] | char | 0xA0001BFE | '; 0x2e 46 |
| *bufferBoardSave[15] | char | 0xA0001BFF | '1'; 0x31 49 |
| *bufferBoardSave[16] | char | 0xA0001900 | '5'; 0x35 53 |
| *bufferBoardSave[17] | char | 0xA0001901 | '; 0x3b 59 |

Figure 154 Tableau des scores déchiffré dans un tableau de char pour la sauvegarde

Il a été remarqué qu'à partir d'un certain nombre de Player non défini et de nombre de scores dans plusieurs modes, le système ne retrouvait plus les bonnes cases mémoires. Vous trouverez un tableau récapitulatif définissant les parties totalement fonctionnelles du tableau des scores dans le point de l'état final du projet.

3.11. Communication UART vers le PC via USB

Afin de pouvoir communiquer avec le PC, un échange de clés est fait entre le PC et le microcontrôleur, détaillés ci-dessous.

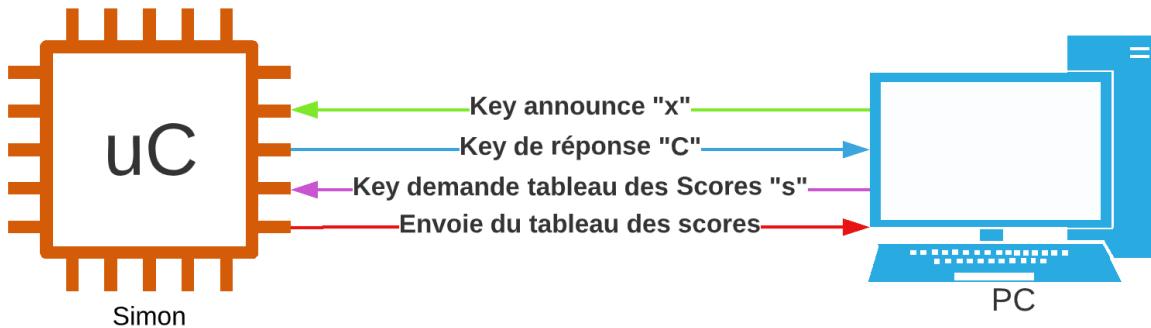


Figure 155 Schéma de l'échange de clés entre le PC et le Microcontrôleur

On a donc la partie de control des clés envoilées par le PC, soit celle d'annonce du PC, soit celle de demande du tableau des scores.

```

298 // Si on reçoit la clé de communication
299 if (character == keyCom)
300 {
301     // Récupère le caractère reçu
302     receiveCharacter = character;
303     // Peut répondre à présent
304     canReply = true;
305 }
306 // Si non si on a reçu la clé de demande du tableau des scores
307 else if (character == keyScore)
308 {
309     // Récupère le caractère reçu
310     receiveCharacter = character;
311     // Peut envoyer le tableau des scores
312     canSendScore = true;
313 }
```

Figure 156 Control des clés de réception via l'UART envoilées par le PC par l'USB (app.c)

Puis on a les deux messages que le microcontrôleur peut envoyer, soit la clé de réponse, soit le tableau des scores en entier.

```

316 // Si on a reçu la clé de communication ET que l'on peut répondre
317 if((receiveCharacter == keyCom) && canReply)
318 {
319     // Envoi la clé d'annonce
320     SendKey(key);
321     // Reset du beuffer de réception des caracteres
322     receiveCharacter = ' ';
323     // Ne peut plus répondre
324     canReply = false;
325 }
326 // Si non si on a reçu la clé d'envoi du tableau des scores ET que l'on peut l'envoyer
327 else if ((receiveCharacter == keyScore) && canSendScore)
328 {
329     // Reset l'autorisation d'envoie du tableau des scores
330     canSendScore = false;
331     // Envoi du tableau des scores
332     SendScoreBoard();
333 }
```

Figure 157 Control des autorisations d'envoi de clé ou de tableau des scores au PC via l'UART (app.c)

De la même manière que pour la sauvegarde, un chiffrage du tableau des scores est fait.

| char*_tbBufferScoreBoard[40] | 0xA001FF38 | "\0\0#1\$ABC@2%1,14.11#2\$" | \u1b30; 0x1b30 | -1610605776 |
|------------------------------|------------|-----------------------------|----------------|-------------|
| *_tbBufferScoreBoard[0] | 0xA0001B30 | NUL; 0x0 | NUL; 0x0 | 0 |
| *_tbBufferScoreBoard[1] | 0xA0001B31 | NUL; 0x0 | NUL; 0x0 | 0 |
| *_tbBufferScoreBoard[2] | 0xA0001B32 | #; 0x23 | #; 0x23 | 35 |
| *_tbBufferScoreBoard[3] | 0xA0001B33 | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[4] | 0xA0001B34 | '\$'; 0x24 | '\$'; 0x24 | 36 |
| *_tbBufferScoreBoard[5] | 0xA0001B35 | 'A'; 0x41 | 'A'; 0x41 | 65 |
| *_tbBufferScoreBoard[6] | 0xA0001B36 | 'B'; 0x42 | 'B'; 0x42 | 66 |
| *_tbBufferScoreBoard[7] | 0xA0001B37 | 'C'; 0x43 | 'C'; 0x43 | 67 |
| *_tbBufferScoreBoard[8] | 0xA0001B38 | '@'; 0x40 | '@'; 0x40 | 64 |
| *_tbBufferScoreBoard[9] | 0xA0001B39 | '2'; 0x32 | '2'; 0x32 | 50 |
| *_tbBufferScoreBoard[10] | 0xA0001B3A | '%'; 0x25 | '%'; 0x25 | 37 |
| *_tbBufferScoreBoard[11] | 0xA0001B3B | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[12] | 0xA0001B3C | '"'; 0x2c | '"'; 0x2c | 44 |
| *_tbBufferScoreBoard[13] | 0xA0001B3D | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[14] | 0xA0001B3E | '4'; 0x34 | '4'; 0x34 | 52 |
| *_tbBufferScoreBoard[15] | 0xA0001B3F | '"'; 0x2e | '"'; 0x2e | 46 |
| *_tbBufferScoreBoard[16] | 0xA0001B40 | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[17] | 0xA0001B41 | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[18] | 0xA0001B42 | '"'; 0x3b | '"'; 0x3b | 59 |
| *_tbBufferScoreBoard[19] | 0xA0001B43 | #; 0x23 | #; 0x23 | 35 |
| *_tbBufferScoreBoard[20] | 0xA0001B44 | '2'; 0x32 | '2'; 0x32 | 50 |
| *_tbBufferScoreBoard[21] | 0xA0001B45 | '\$'; 0x24 | '\$'; 0x24 | 36 |
| *_tbBufferScoreBoard[22] | 0xA0001B46 | 'A'; 0x41 | 'A'; 0x41 | 65 |
| *_tbBufferScoreBoard[23] | 0xA0001B47 | 'C'; 0x43 | 'C'; 0x43 | 67 |
| *_tbBufferScoreBoard[24] | 0xA0001B48 | 'D'; 0x44 | 'D'; 0x44 | 68 |
| *_tbBufferScoreBoard[25] | 0xA0001B49 | '@'; 0x40 | '@'; 0x40 | 64 |
| *_tbBufferScoreBoard[26] | 0xA0001B4A | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[27] | 0xA0001B4B | '%'; 0x25 | '%'; 0x25 | 37 |
| *_tbBufferScoreBoard[28] | 0xA0001B4C | '1'; 0x31 | '1'; 0x31 | 49 |
| *_tbBufferScoreBoard[29] | 0xA0001B4D | '"'; 0x2c | '"'; 0x2c | 44 |
| *_tbBufferScoreBoard[30] | 0xA0001B4E | '6'; 0x36 | '6'; 0x36 | 54 |
| *_tbBufferScoreBoard[31] | 0xA0001B4F | '"'; 0x2e | '"'; 0x2e | 46 |
| *_tbBufferScoreBoard[32] | 0xA0001B50 | '4'; 0x34 | '4'; 0x34 | 52 |
| *_tbBufferScoreBoard[33] | 0xA0001B51 | '6'; 0x36 | '6'; 0x36 | 54 |
| *_tbBufferScoreBoard[34] | 0xA0001B52 | '"'; 0x3b | '"'; 0x3b | 59 |

Figure 158 Tableau des scores déchiffré dans un tableau de char pour envoi via UART

Ici on a un exemple avec deux Player qui ont fait une game chacun dans un mode.

La demande du tableau des scores peut se faire à tout moment, mais c'est la dernière sauvegarde qui sera envoyée. C'est-à-dire que si un nouveau Player jouer actuellement, il ne sera pas pris en compte.

3.12. Control de charge de l'accumulateur

Afin de mesurer le niveau de tension de l'accumulateur, il nous suffit d'activer la pin dédiée et de lire le niveau ne notre ADC 10bit. Cette lecture est faite toutes les 60 secondes.

```

353     // Mise à l'état haut la pin de lecture de l'accumulateur
354     BAT_READ_On();
355
356     // Start du driver ADC
357     DRV_ADC_Start();
358     // Tant que la conversion n'est pas finie
359     while(!DRV_ADC_SamplesAvailable()) {}
360     // Stop de l'ADC
361     DRV_ADC_Stop();

```

Figure 159 Activation de la pin de lecture du niveau de tension de l'accumulateur mesuré avec l'ADC (app.c)

Afin de récupérer la valeur exacte en tension, il va falloir réadapter les valeurs avec les niveaux de tensions exactes. Car un pont diviseur avait été installé avant l'entrée de l'ADC afin que le niveau maximum mesuré ne dépasse pas 3.3V. En effet l'entrée de l'ADC ne fait pas des pins supportant le 5V.

$$U_{brute\ mesurée} = \frac{valeur\ de\ l'ADC}{valeur\ max\ de\ l'ADC} * tension\ max = \frac{950}{1023} * 3.3 = 3.065V$$

Une fois la valeur brute obtenue, il faut la réadapter avec la valeur max de l'accumulateur.

$$U_{réelle\ de\ l'accumulateur} = \frac{U_{brute\ mesurée}}{U_{max\ accu}} * U_{max\ accu} = \frac{3.065}{3.3} * 4.7 = 3.9V$$

On lira donc une tension sur le uC de 3.065V, quand il y aura 3.9V sur l'accumulateur.

```

363     // Récupération de la valeur de l'ADC
364     readBat = DRV_ADC_SamplesRead(0);
365
366     // Si le niveau de l'accumulateur est inférieur à son niveau min
367     if((readBat <= MIN_ACCU_LVL) && (readBat >= NOT_CONNECTID_ACCU))
368     {

```

Figure 160 Lecture et vérification du niveau de tension de l'accumulateur (app.c)

Lorsque que l'on passe en dessous de 3.5V décidé comme tension min, on affiche un message à l'utilisateur de recharger le Simon. Puis on reconstruit si on a chargé suffisamment pour rejouer, la valeur a été fixée à 3.7V arbitrairement.

```

373     }
374     // Si non si le niveau de l'accumulateur dépasse la valeur min d'arrêt de charge
375     else if (readBat >= MIN_ACCU_CHARGE_LVL)
376     {

```

Figure 161 Contrôle si la valeur est de nouveau supérieure au minimum pour rejouer

À la fin, il ne faudra pas oublier de redescendre la pin de mesure, afin de ne plus consommer de courant pour la mesure, et de la refaire dans 60 secondes.

```

387     // Mise à l'état bas la pin de lecture de l'accumulateur
388     BAT_READ_Off();

```

Figure 162 Remise à l'état bas de la pin qui permet de mesurer la tension de l'accumulateur

3.13. Communication avec la carte microSD via SPI3

La partie carte SD a été mise de côté dès le début du projet. Elle n'a donc pas été mise en service, et aucune partie Firmware la concernant n'a été faite.

4. Software

4.1. Flowcharts

4.1.1. Flowchart de la communication

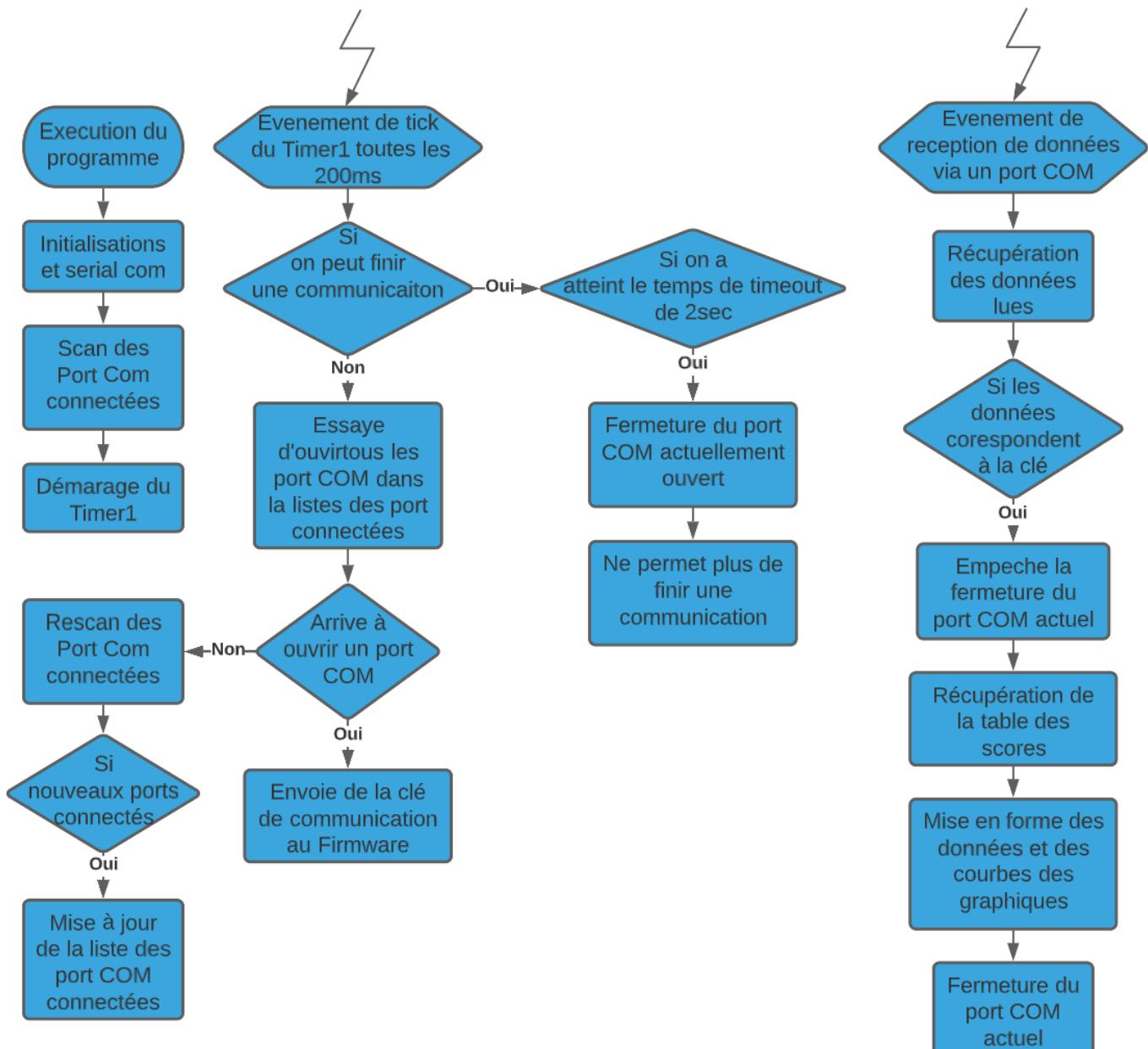


Figure 163 Flowchart Software de la communication UART

4.1.2. Flowchart du traitement de données

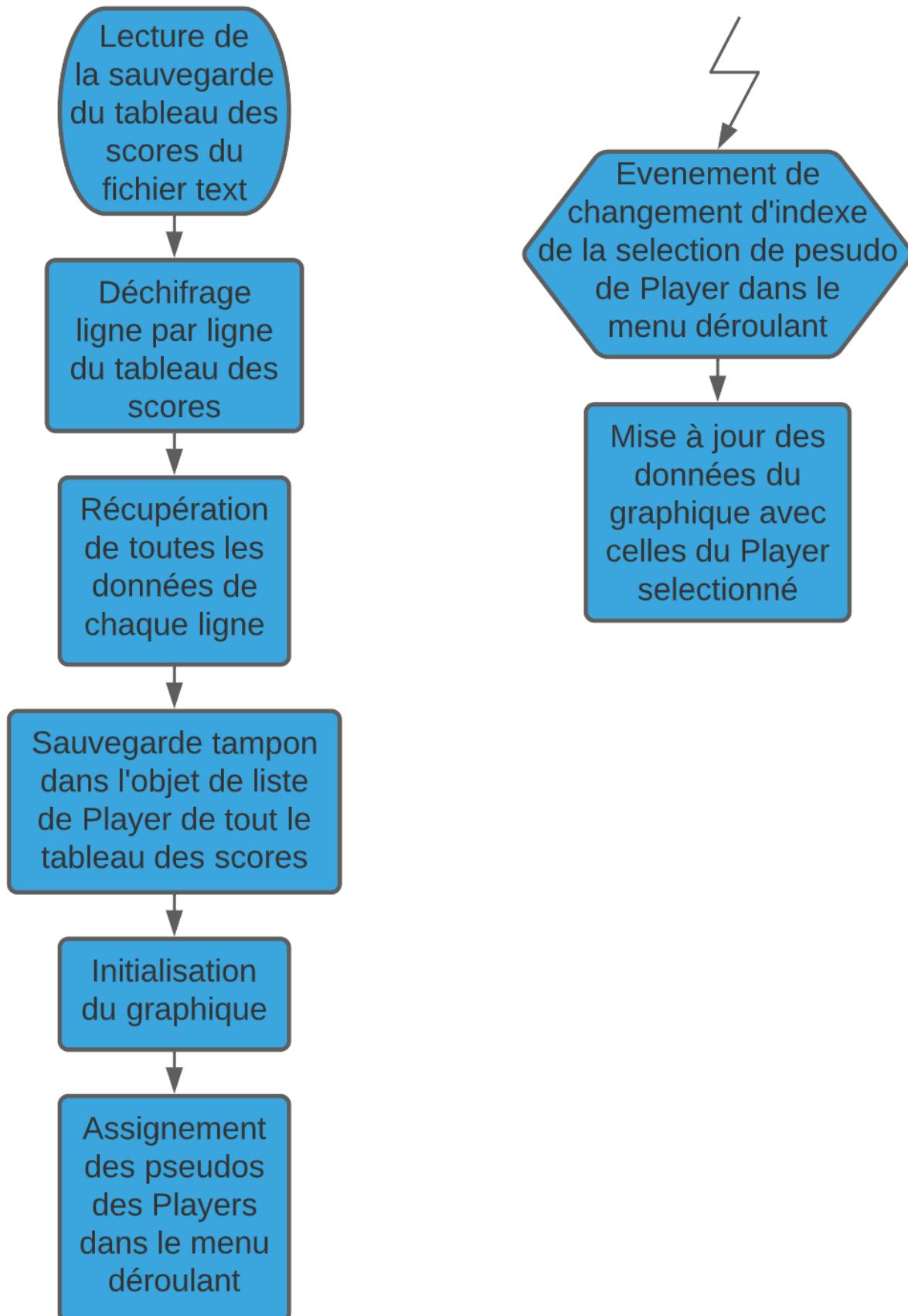


Figure 164 Flowchart Software du traitement de données du tableau des scores reçu

4.2. Communication entre le Simon et le PC

Afin de pouvoir détecter notre Simon lorsqu'on le connecte au PC, j'ai dû trouver un mécanisme pour y arriver.

J'ai opté pour un scan de tous les ports COM connectés au PC, puis à tour de rôle d'essayer d'ouvrir chaque port COM. Si l'ouverture d'un port COM est faite, une clé est envoyée. Puis pendant la durée de deux secondes, on attend une réponse d'une clé spécifique de la part du Firmware.

Si la bonne clé est récupérée, la table des scores est ensuite envoyée via ce même port COM, puis il referme la communication tout de suite après.

Le programme fonctionne si on connecte d'abord la carte puis on exécute le programme, ou alors en Hotplug si on exécute d'abord le programme puis on connecte la carte. Il supporte également le multi Hotplug, c'est-à-dire que l'on peut connecter et déconnecter autant de fois que l'on veut la carte, tout en laissant le Software lancé.

4.2.1. Initialisation

Lors du démarrage du programme, on va commencer par les initialisations

```

52         // Constructeur de la class View
53         1 référence
54     public View()
55     {
56         // Initialisation
57         InitializeComponent();
58         // Lecture des ports au démarage
59         canRetryOpen = RefreshSerialPort();
60         // Configure les paramètres de la communicaiton UART
61         InitSerialCom();
62         // Démarré le timer1 principale qui fait tourner tout le programme
63         timer1.Start();
64
65         // Création et paramétrage du graphique de progression
66         CreatCharType();
67
68         //----- DEBUG -----//
69         AssignAllPlayerToList();
    }
```

Figure 165 Initialisations au démarrage du Software (View.cs)

Lors de l'initialisation du système, on va commencer par scanner tous les ports COM qui sont connectés au démarrage, grâce à la méthode « RefreshSerialPort ».

```

98         // Récupéraiton les ports COM du system
99         2 références
100    private bool RefreshSerialPort()
101    {
102        // Récupération de la listes des port COM du system
103        portsNew = SerialPort.GetPortNames();
```

Figure 166 Méthode « RefreshSerialPort » et récupérations des noms des ports COM connectés au PC (View.cs)

Ici on utilise directement les méthodes permettant d'accéder aux paramètres de la communication série. Dans notre cas on va uniquement récupérer les noms de tous les ports COM connectés et les stocker.

Puis si la nouvelle liste est différente de l'ancienne liste de ports COM, alors on peut les sauvegarder dans notre tableau de travail des noms de tous les ports COM connectés.

```

104         // Si la nouvelle liste de ports COM est différente de la précédente
105         if (!Enumerable.SequenceEqual(portsNew, portsOld))
106         {
107             // Redimensionnement du tableau des ports COM à la taille du nouveau tableau COM
108             Array.Resize<string>(ref ports, portsNew.Length);
109             // Copie du tableau des nouveaux ports dans le tableau des ports de travail
110             portsNew.CopyTo(ports, 0);
111             // Redimensionnement du tableau des anciens ports COM à la taille du tableau de travail COM
112             Array.Resize<string>(ref portsOld, ports.Length);
113             // Copie du tableau des ports de travail dans le tableau des anciens ports COM
114             ports.CopyTo(portsOld, 0);
115             // Retourner une réponse vrai
116             return true;
117         }
118         // Si non on a la même liste de ports
119     else
120     {
121         // Retourner une réponse fausse
122     }
123 }
124 }
```

Figure 167 Sauvegarde dans le tableau de travaille uniquement si de nouveaux ports COM (View.cs)

C'est lors de ces initialisations que l'on va configurer les paramètres de communication série UART passant par le port COM. Il faudra que le Firmware et le Software aient les mêmes configurations, afin qu'ils puissent communiquer entre eux.

```

78         // Initialisaiton des paramètres UART du port COM
79         1 référence
80     private void InitSerialCom()
81     {
82         // Configuration du BaudRate
83         serialPort1.BaudRate = 9600;
84         // Pas de parité
85         serialPort1.Parity = Parity.None;
86         // Taille du message de 8 bits
87         serialPort1.DataBits = 8;
88         // Un seul bit de stop
89         serialPort1.StopBits = StopBits.One;
90         // Pas de Handshake
91         serialPort1.Handshake = Handshake.None;
92
93         // Set read timeouts
94         serialPort1.ReadTimeout = 500;
95         // Set write timeouts
96         serialPort1.WriteTimeout = 500;
}
```

Figure 168 Configuration de la communication série UART passant par le port COM (View.cs)

Puis une fois le tout configurer on peut commencer à réellement faire les actions normales du programme. Pour cela on démarre donc le timer1.

```

73         // Démarre le timer1 principalle qui fait tourner tout le programme
74         timer1.Start();
```

Figure 169 Démarrage du Timer1 (View.cs)

4.2.2. Événement Timer1

C'est grâce au Timer1 que l'on va cadencer nos essais de communication et nos timeouts.

```

347 // Événement du Timer1 toutes les 100ms
348 1 référence
349 private void timer1_Tick(object sender, EventArgs e)
350 {
351     // Si on peut commencer le délai d'attente de la réception de la clé
352     if (canEndCom)
353     {
354         // Si on a attendu 2sec
355         if (counterTimerEndCom == 10)
356         {
357             // Fini la communication avec le port COM actuel et le ferme
358             EndTryCom();
359             // Reset compteur d'attente de réponse
360             counterTimerEndCom = 0;
361         }
362         // Incrémentation du compteur d'attente de réponse
363         counterTimerEndCom++;
364     }
365     // Si non on a pas une communication en cours et donc on peut essayer d'en commencer une
366     else
367     {
368         // Commence l'essay de communication avec tous les ports COM
369         StartTryCom();
370     }

```

Figure 170 Méthode de l'événement de tick du Timer1 toutes les 200ms (View.cs)

Ici si on ne peut pas fermer une communication, c'est que l'on n'a pas encore ouvert de port, c'est pour cela que l'on va donc essayer d'ouvrir une communication d'abord.

Afin de pouvoir envoyer quelque chose sur le port COM, il va valoir l'ouvrir d'abord, c'est pourquoi en arrivant dans la méthode « StartTryCom », que la première chose que l'on fait est de tester si on a réussi à ouvrir un port.

```

126 // Commence l'essay de communication avec tous les ports COM
127 1 référence
128 private void StartTryCom()
129 {
130     // Si on arrive à ouvrir le port COM et qu'on a le droit d'en ouvrir un
131     if (TryOpenCom() && canRetryOpen)

```

Figure 171 Méthode qui essaye de communiquer avec un port COM (View.cs)

Si on n'a pas réussi à ouvrir aucun des ports, cela veut peut-être dire que notre carte n'est pas encore branchée, ou alors qu'elle est déjà branchée et que l'on a déjà communiqué avec.

```

146 // Si non on n'arrive pas à ouvrir les ports COM et on n'a pas le droit d'en ouvrir
147 else
148 {
149     // Récupéraiton des ports COM du system
150     canRetryOpen = RefreshSerialPort();
151 }
152 }

```

Figure 172 Si pas de port COM ouverts, rescanne de tous les ports COM connectées au PC (View.cs)

Dans les deux cas, on va effectuer un rescanne de tous les ports COM qui sont connectés au PC.

Pour réaliser ce test, on va utiliser la méthode « TryOpenCom », qui va simplement à tour de rôle essayer d'ouvrir les ports COM stockés dans le tableau de travail des ports COM connectés.

```

154          // Essay d'ouvrir un port COM
155          1 référence
156      private bool TryOpenCom()
157      {
158          // Testes tous les ports COM du tableau de travail des ports COM
159          for (int i = 0; i < ports.Length; i++)
160          {
161              // Essaye d'ouvrir le port COM actuelle
162              try
163              {
164                  // Récupération du nom du port COM actuel
165                  serialPort1.PortName = (string)ports[i];
166                  // Ouverture du port COM actuel
167                  serialPort1.Open();
168                  // Compteur du port COM sélectionné actuellement
169                  currentSelectedPort = i;
170                  // Retourner une réponse vrai
171                  return true;
172              }
173              // Si non s'il n'arrive pas
174              catch
175              {
176                  // Passe à la prochainne action
177                  continue;
178              }
179          }
180          // Retourner une réponse fausse
181          return false;
182      }

```

Figure 173 Méthode « TryOpenCom » qui va essayer d'ouvrir les port COM connectés (View.cs)

Si parmi la liste des ports COM on n'arrive pas à en ouvrir un, la méthode va retourner une réponse fausse. En revanche si on arrive à ouvrir un port comme, la méthode va retourner une réponse vraie.

Si on a eu l'autorisation pour réouvrir un port, dans notre cas au scan que l'on a effectué à l'initialisation du programme, alors le test d'ouverture sera un succès.

```

129          // Si on arrive à ouvrir le port COM et qu'on a le droit d'en ouvrir un
130          if (TryOpenCom() && canRetryOpen)
131          {
132              // Essaye d'envoyer la clé d'envoy pour s'annoncer au device voulu qui l'attend
133              try
134              {
135                  // Envoi le la clé d'annonce sur le port COM ouvert actuellement
136                  serialPort1.WriteLine(ANNOUNCE_KEY);
137              }
138              // Si non s'il n'arrive pas continue normalment l'exécution du programme
139              catch { }

```

Figure 174 Test si on a pu et on a le droit d'ouvrir un port COM et envoie de la clé au Firmware (View.cs)

Il ne nous reste plus qu'à envoyer la clé de communication qui fera le lien avec le Firmware du Simon.

```
23          private const string ANNOUNCE_KEY = "x";    // Clé d'annoncement
```

Figure 175 Clé pour communiquer avec le Firmware

Une fois la clé envoyée, on démarre le compteur du temps d'attente d'une réponse du Firmware.

```

141 |           // Démare le délai d'attente de réception de la clé de retour sur le port COM actuel
142 |           canEndCom = true;
143 |           // Arrete d'essayer d'ouvrir un autre port COM
144 |           canRetryOpen = false;
145 |

```

Figure 176 Démarrage du compteur du temps d'attente d'une réponse du Firmware (View.cs)

C'est à partir de ce moment que l'on va attendre deux secondes en tout, en passant dix fois dans l'événement de tick du Timer1.

```

347 |           // Évenement du Timer1 toutes les 100ms
348 |           1 référence
349 |           private void timer1_Tick(object sender, EventArgs e)
350 |           {
351 |               // Si on peut commencer le délai d'attente de la réception de la clé
352 |               if (canEndCom)
353 |               {
354 |                   // Si on a attendu 2sec
355 |                   if (counterTimerEndCom == 10)
356 |                   {
357 |                       // Fini la communicaiton avec le port COM actuel et le ferme
358 |                       EndTryCom();
359 |                       // Reset compteur d'attente de réponse
360 |                       counterTimerEndCom = 0;
361 |                   }
362 |                   // Incrémentation du compteur d'attente de réponse
363 |                   counterTimerEndCom++;

```

Figure 177 Compteur du temps d'attente de deux secondes d'une réponse du Firmware (View.cs)

Si après les deux secondes on n'a pas obtenu de réponse, c'est la méthode « EndTryCom » qui sera appelée pour terminer la communication en fermant le port COM, et en l'effaçant de la liste de travail des ports COM connectés au PC.

```

183 |           // Fini la communicaiton avec le port COM actuel et le ferme
184 |           1 référence
185 |           private void EndTryCom()
186 |           {
187 |               // Ferme le port COM actuel
188 |               serialPort1.Close();
189 |               // Efface le port COM actuel de la liste de travail des ports COM
190 |               ports[currentSelectedPort] = " ";
191 |               // Arrete la fermeture des ports et reprend le code normallement
192 |               canEndCom = false;
193 |               // Peut réessayer d'ouvrir un autre port
194 |               canRetryOpen = true;

```

Figure 178 Méthode « EndTryCom » de fermeture du port COM (View.cs)

Il n'y a qu'un seul moyen d'arrêter le compteur d'attente de réponse, c'est bien évidemment d'avoir reçu une réponse. C'est donc au point suivant que vous allez trouver les explications concernant la réponse reçue de la part du Firmware.

4.2.3. Événement réception de données du Firmware

Une fois la clé envoyée, et pendant le temps d'attente, il n'y a que l'événement de réception de données qui peut tout arrêter.

```
316 // Événement si on reçoi des données via la communication UART sur un port COM
317 1 référence
318 private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
319 {
320     // Buffeur de lecture des données reçues
321     string dataRx = serialPort1.ReadExisting();
```

Figure 179 Événement de réception de données série UART du port COM et lecture des données (View.cs)

Une fois entré dans l'événement, cela veut dire que l'on a reçu des données via le port série UART du port COM actuellement ouvert. C'est donc à ce moment-là que l'on va aller lire le buffeur de réception.

Une fois les données lues, on peut les comparer avec la clé à laquelle on s'attend de recevoir.

```
22 private const string KEY = "C"; // Clé de retour attendue X
```

Figure 180 Clé de réponse du Firmware (View.cs)

```
322 // Si les données reçues sont égales à la clé secrète
323 if ((dataRx == KEY) && !canGetScoreArray)
324 {
325     // Reste avec le port COM ouvert
326     canEndCom = false;
327     // Reset le compteur d'attente de réponse
328     counterTimerEndCom = 0;
329     // Envoi la clé d'envoi de la table des scores sur le port COM ouvert actuellement
330     serialPort1.WriteLine(ANNOUNCE_KEY);
331     //
332     canGetScoreArray = true;
333 }
```

Figure 181 Test si les données reçues correspondent à la clé de réponse du Firmware (View.cs)

Si les données reçues ne correspondent pas à la clé de réponse du Firmware, alors une fois le timeout des deux secondes écoulé, la communication s'arrêtera et le port se ferme, et le programme pourra suivre son fonctionnement normal.

Mais dans le cas où les données correspondent à la clé de réponse du Firmware, alors on peut en toute sécurité envoyer à notre tour la clé de demande d'envoi du tableau des scores.

Puis quand le Firmware va nous envoyer le tableau des scores, on passera donc dans l'autre partie de la condition qui stockera les données dans un fichier texte.

```
360 // Si non on peut réceptionner le tableau des scores
361 else if (canGetScoreArray)
362 {
363     // Ecrit les données reçues dans le fichier texte de sauvegarde
364     dataProcess.WriteDataToFile(dataRx);
365 }
366 }
```

Figure 182 Sauvegarde du tableau des scores envoyé par le Firmware dans un fichier texte (View.cs)

4.3. Traitement des données du tableau des scores

4.3.1. Sauvegarde des données dans un fichier texte

Afin de pouvoir traiter les données du tableau des scores plus aisément, et d'en avoir un historique, lors de la réception de celui-ci il est directement sauvegardé dans un fichier texte.

Ce fichier texte est créé dans le constructeur de la classe « Data », lorsqu'on l'instancie.

```
48 | Data dataProcess = new Data(); // Accès à la class Data via cet objet
```

Figure 183 Instanciation de l'objet permettant d'accéder aux méthodes et propriétés de la class « Data » (View.cs)

```
36 | // Constructeur de la class Data  
1 référence  
37 | public Data()  
38 | {  
39 |     // Récupère le chemin d'accès au fichier text de sauvegarde créé  
40 |     path = Path.GetDirectoryName(Application.ExecutablePath) + $"\\{fileName}";
```

Figure 184 Création du fichier texte dans le constructeur de la class « Data » (Data.cs)

Ici on crée non seulement le fichier à la racine où se trouve le .exe, mais on récupère également son chemin d'accès exact.

S'il avait déjà été créé lors d'une précédente exécution du Software, tout le texte présent sera effacé.

```
43 | // Supprime tout le text du fichier text de sauvegarde  
44 | File.WriteAllText(path, string.Empty);  
45 | }
```

Figure 185 Effacement de tout le texte déjà présent dans le fichier texte (Data.cs)

Puis c'est lorsque l'on récupère le tableau des scores via le Firmware, que l'on va créer l'objet qui nous permet de faire la gestion du fichier texte.

```
53 | // Stock le tableau des scores récupéré par l'UART dans un fichier text  
1 référence  
54 | public void WriteDataToFile(string receiveData)  
55 | {  
56 |     // Essaye d'écrire les données reçus dans le fichier  
57 |     try  
58 |     {  
59 |         // Instanciation de l'objet de gestion du fichier text  
60 |         swScoreBoard = new StreamWriter(path, true);
```

Figure 186 Création de l'objet permettant de faire la gestion du fichier texte (Data.cs)

Ensuite on va écrire tout le texte reçu en brut dans le fichier texte.

```
61 | // Écrit ligne par ligne dans le fichier text  
62 | swScoreBoard.WriteLine(receiveData);  
63 | // Ferme le fichier text  
64 | swScoreBoard.Close();
```

Figure 187 Ecriture des données reçues dans le fichier texte (Data.cs)

Puis cela jusqu'à recevoir la clé de fin de fichier.

```
66 | // Si on reçoit la clé de fin de fichier  
67 | if(receiveData.Contains(END_FILE))  
68 | {  
69 |     // Reformate le fichier text  
70 |     ReformatTextFile(swScoreBoard);  
71 | }
```

Figure 188 Si on reçoit la clé de fin de fichier (Data.cs)

Afin de pouvoir recevoir les données avec autant d'espaces entre les informations et autant de retour à la ligne due à l'envoi non continu du tableau des scores, une remise en forme du fichier texte est faite.

```
84 // Liste de reconstruction du tableau des scores
85 List<string> records = new List<string>();
```

Figure 189 Liste buffer de récupération des données lues (Data.cs)

Pour cela on va tout mettre dans une liste de string, afin de partitionner tous les caractères.

```
93 // Lit toutes les lignes du fichier
94 while ((currentLine = srScoreBoard.ReadLine()) != null)
95 {
96     // Si c'est la première ligne du fichier
97     if (headerline)
98     {
99         // This handles the header record
100        records.Add(currentLine);
101        headerline = false;
102    }
103    // Si non c'est le reste du fichier
104    else
105    {
106        //this is the continuation on a ongoing record, so append to the last item in the list
107        if (!string.IsNullOrWhiteSpace(currentLine))
108        {
109            //add to the current record only if the line is not empty
110            records[records.Count - 1] += currentLine;
111        }
112    }
113 }
```

Figure 190 Traitement du fichier dans la liste (Data.cs)

Puis on va réécrire la liste dans le fichier texte, mais cette fois-ci on va ajouter un retour à la ligne après chaque caractère de fin de ligne.

```
115 // Ferme le fichier text
116 srScoreBoard.Close();
117
118 // Supprime tout le text du fichier text de sauvegarde
119 File.WriteAllText(path, string.Empty);
120
121 string bufferAllFILE = records[0];
122 // Instanciation de l'objet de gestion du fichier text
123 swScoreBoard = new StreamWriter(path, true);
124 // Ajoute un retour à la ligne au caractère de fin de ligne
125 string receiveDataLined = bufferAllFILE.Replace(END_LINE, $"{END_LINE}\n");
126 // Écrit ligne par ligne dans le fichier text
127 swScoreBoard.WriteLine(receiveDataLined);
128 // Ferme le fichier text
129 swScoreBoard.Close();
```

Figure 191 Rajout d'un retour à la ligne à la fin de chaque caractère de fin de ligne (Data.cs)

On aura donc sur chaque ligne les informations de toutes les games d'un Player dans un mode de difficulté.

```
132 // Déchiffre le tableau des scores et le stock dans la liste d'objets de Player
133 DecryptData();
134
135 // Assigne tous les pseudos de tous les Player du tableau des scores dans le menu déroulant
136 viewProcess.AssignAllPlayerToList();
137 }
```

Figure 192 Fermeture du fichier texte et déchiffrement du tableau des scores reçu (Data.cs)

Enfin il ne reste plus qu'à refermer le fichier, et commencer à déchiffrer.

4.3.2. Déchiffrage du tableau des scores

Afin de déchiffrer le tableau des scores, on va devoir lire l'intégralité du fichier texte ligne par ligne, jusqu'à la fin du fichier.

```

67     // Déchiffre le tableau des scores et le stock dans la liste d'objets de Player
68     1 référence
69     public void DecryptData()
70     {
71         int nbrAllPlayers = 0;           // Nombre de Players dans le tableau des scores
72         int oldIndexScore = 0;          // Ancienne valeur de l'index du caractère séparateur de score
73         int oldIndexAccur = 0;          // Ancienne valeur de l'index du caractère séparateur de précision
74
75         string accurParse = " ";       // Dernière valeur de précision en fin de ligne
76
77         // Traite toutes les lignes du fichier text
78         foreach (string line in File.ReadLines($"{{fileName}}"))
    {

```

Figure 193 Boucle permettant de lire ligne par ligne tout le fichier texte (Data.cs)

Pour rappel, une ligne représente le nombre de points de score et de précision par mode de jeu, pour un Player à la fois.

Figure 194 Structure d'une ligne du tableau des scores

La ligne démarre par un « # » qui est suivi de l'ID du Player, dans ce cas c'est l'ID « 1 ». Puis c'est le « \$ » qui est suivi du pseudo du Player, dans ce cas « RICQUI ». Ensuite on a le « @ » qui est suivi du mode de difficulté, 1 pour « Easy », 2 pour « Hard » et 3 pour « Extrem ». Puis c'est le « % » qui est suivi du nombre de parties effectué dans le mode actuel par le Player, dans ce cas le Player a joué 3 parties dans le mode actuel « Easy ».

Après les informations pour savoir de qui et de quoi on parle, on a les points de scores et de précision qui se suivent pour le nombre de games effectuées.

C'est donc après une « , » que l'on a les points du score obtenus, puis après le « . » que l'on a les points de précision obtenus. Ensuite le paterne se répété autant de fois que de games faites par le Player, dans ce cas on a un score de 12 points et une précision de 48 pour la première partie, et le paterne se répète 3 fois.

Tous ces caractères ont été choisis lors du développement du Software, mais ils peuvent être modifiés à tout moment, si le projet est amené à évoluer.

```

22     private const string END_LINE = "\n";           // Caractère de fin de ligne du tableau des scores
23     private const string COUNT_PLAYER = "#";        // Caractère de début de ligne du tableau des scores
24     private const string PLAYER_PSEUDO = "$";        // Caractère de début de pseudo du tableau des scores
25     private const string GAME_MODE = "@";           // Caractère de début du mode du tableau des scores
26     private const string NBR_GAMES = "%";           // Caractère de début du nombre de games du mode actuel du tableau des scores
27     private const string SCORE_SEPAR = ",";          // Caractère de séparation des parties du tableau des scores
28     private const string SCORE_ACCUR = ".";          // Caractère de séparation entre le score et la précision mode du tableau des scores

```

Figure 195 Constantes des caractères spéciaux séparateurs des lignes du tableau des scores (Data.cs)

C'est donc par chercher l'ID du Player que l'on va commencer. Pour cela on va parser la première ligne à la recherche du numéro qui se trouve entre le caractère « # » et le caractère « \$ ».

```

82     // Recherche et assigne l'ID du player actuel
83     actualPlayerID = int.Parse(line.Substring(COUNT_PLAYER) + 1, line.IndexOf(PLAYER_PSEUDO) - 1);

```

Figure 196 Recherche et sauvegarde de l'ID de la première ligne (Data.cs)

En utilisant la recherche avec le caractère avant et après notre information, cela nous permet d'avoir une information qui varie en nombre de caractère, et donc d'être compatible avec de petites et des grandes valeurs.

Puis c'est à ce moment-là que l'on va vérifier si c'est un nouveau Player.

```

85     // Si c'est un nouveau Player
86     if (actualPlayerID > nbrAllPlayers)
87     {

```

Figure 197 Test si c'est un nouveau Player, ou si c'est un Player connu (Data.cs)

Si c'est un nouveau joueur qui n'a pas encore été déchiffré, on va alors mettre à jour le nombre de Player max, et on va ajouter un objet « Player » à la liste des Players.

```
88 // Met à jour le nombre de Players qu'il y a dans le tableau des scores
89 nbrAllPlayers = actualPlayerID;
90 // Instancie l'objet du player et assigne son ID
91 allPlayers.Add(new Player(actualPlayerID));
```

Figure 198 Création de l'objet Player dans la liste de Players (Data.cs)

C'est cet objet qui nous permettra d'exploiter les informations du tableau des scores pour chaque Player. La schématique de la structure complète se trouve ci-dessous, puis le détail sera expliqué au fur et à mesure du point actuel.

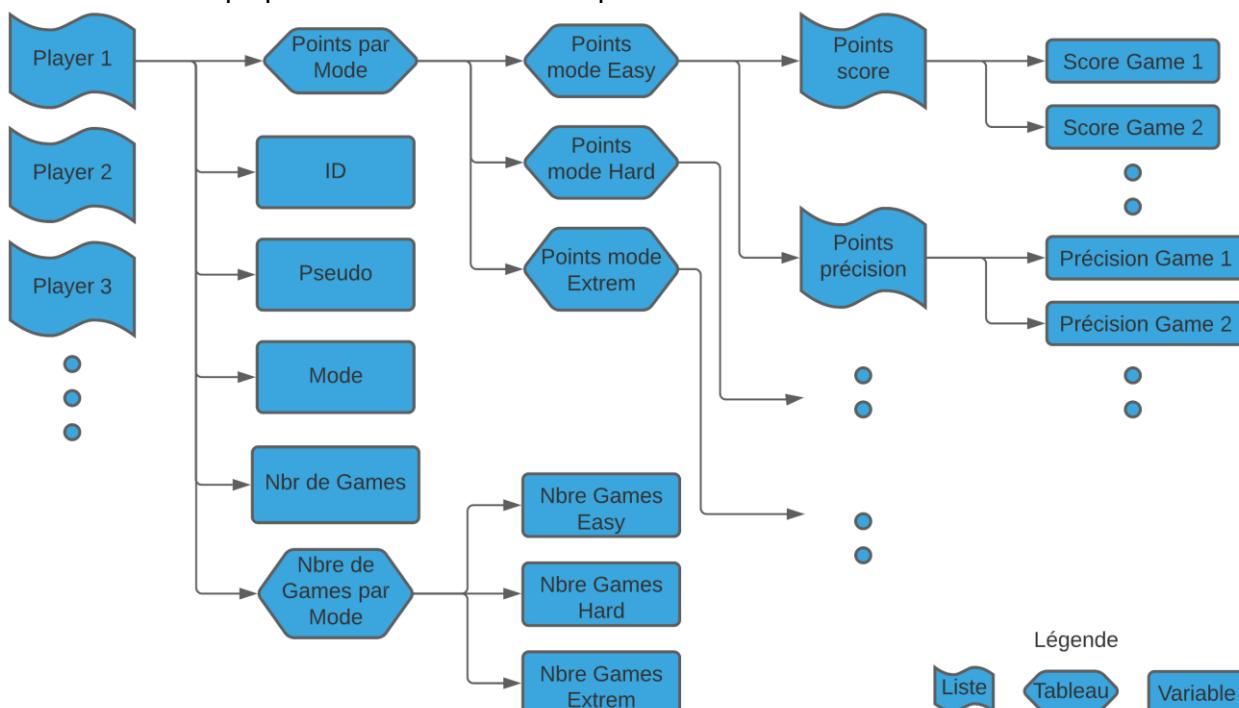


Figure 199 Structure des données récupérées par Player depuis le tableau des scores

C'est donc finalement une liste d'objet Player qui va pouvoir être exploitée dans le code, à partir des informations déchiffrées du tableau des scores.

Après IID, c'est le pseudo du Player qui va être récupéré entre le caractère « \$ » et le caractère « @ ».

```
92 // Recherche et assigne le pseudo du player actuel qui est entre un caractère de début et de fin
93 allPlayers[actualPlayerID].playerName =
94     line.Substring(line.IndexOf(PLAYER_PSEUDO) + 1, (line.IndexOf(GAME_MODE) - (line.IndexOf(PLAYER_PSEUDO) + 1)));
95 }
```

Figure 200 Recherche et sauvegarde du pseudo du Player (Data.cs)

Ensuite on va récupérer des informations que l'on va devoir stocker soit du nouveau Player, soit d'un Player que l'on a déjà stocké un mode de jeu. C'est donc par le mode actuellement traité que l'on va chercher entre le caractère « @ » et le caractère « % ».

```
97 // Recherche et assigne le game mode actuel qui est entre un caractère de début et de fin
98 allPlayers[actualPlayerID].gameMode =
99     int.Parse(line.Substring(line.IndexOf(GAME_MODE) + 1, (line.IndexOf(NBR_GAMES)) - (line.IndexOf(GAME_MODE) + 1)));
```

Figure 201 Recherche et sauvegarde du mode de la ligne actuelle (Data.cs)

Puis c'est le nombre de games pour le mode actuellement traité que l'on va récupérer entre le caractère « % » et le premier caractère « , ».

```
100 // Recherche et assigne le nombre de games du mode actuel qui est entre un caractère de début et de fin
101 allPlayers[actualPlayerID].numberGames[allPlayers[actualPlayerID].gameMode - 1] =
102     int.Parse(line.Substring(line.IndexOf(NBR_GAMES) + 1, (line.IndexOf(SCORE_SEPAR) - (line.IndexOf(NBR_GAMES) + 1))));
```

Figure 202 Recherche et sauvegarde du nombres de games du mode actuel (Data.cs)

Puis nous alors récupérer les points de scores et de précision du Player pour chaque game du mode actuel effectuée.

```
104 | // Traite les points de score et de précision pour toutes les games faites par le Player sur le mode actuel
105 | for (int i = 0; i < allPlayers[actualPlayerID].numberGames[allPlayers[actualPlayerID].gameMode - 1]; i++)
106 | {
```

Figure 203 Boucle permettant de récupérer tous les points de score et de précision du mode actuel (Data.cs)

Puis c'est entre le premier caractère « , » et le premier caractère « . » que l'on va trouver les points de score de la première game.

```
107 | // Recherche et assigne le score actuel du mode actuel qui est entre un caractère de début et de fin
108 | allPlayers[actualPlayerID].allModes[allPlayers[actualPlayerID].gameMode - 1].scoresPoints[0].Add(
109 |     int.Parse(line.Substring((line.IndexOf(SCORE_SEPAR, oldIndexScore) + 1),
110 |         line.IndexOf(SCORE_ACCUR, oldIndexAccur) - (line.IndexOf(SCORE_SEPAR, oldIndexScore) + 1)));
```

Figure 204 Récupération et sauvegarde des points de scores de la game actuellement traitée (Data.cs)

Comme le paterne pour les points de scores va se répéter, et que l'on a le même caractère « , » à chaque fois pour les séparer, il va fallois faire les recherches de la prochaine valeur à partir de la dernière que l'on a déjà lue.

C'est pour cela que l'on va sauvegarder les positions des premiers caractères « , » et « . ».

```
112 | // Sauvegarde la position du caractère séparant les données de la game actuelle
113 | oldIndexScore = line.IndexOf(SCORE_SEPAR, oldIndexScore) + 1;
114 | // Sauvegarde la position du caractère séparant les données de score et de précision de la game actuelle
115 | oldIndexAccur = line.IndexOf(SCORE_ACCUR, oldIndexAccur) + 1;
```

Figure 205 Sauvegarde de la position des caractères « , » et « . » actuels (Data.cs)

Ensuite il ne nous manque plus qu'à récupérer les points de précision, qui se trouvent entre le premier caractère « . » et le deuxième caractère « , ».

```
117 | // Essaye de récupérer les données de précision du mode actuelle
118 | try
119 | {
120 |     // Recherche le niveau de précision actuel du mode actuel qui est entre un caractère de début et de fin
121 |     accurParse = line.Substring(oldIndexAccur, line.IndexOf(SCORE_SEPAR, oldIndexAccur) - oldIndexAccur);
122 | }
```

Figure 206 Récupération des points de précision de la game actuellement traitée (Data.cs)

Malheureusement quand le paterne arrivera à la fin de la ligne, les points de scores se trouveront alors entre le caractère « . » et le caractère de fin de ligne « ; ». C'est pour cela que l'on va dans le cas de la récupération des points de précision de la dernière game de la ligne, prendre toutes les données après le dernier caractère « . ».

```
123 | // On est alors à la fin d'une ligne
124 | catch (Exception)
125 | {
126 |     // Récupère le niveau de précision actuel
127 |     accurParse = line.Substring(oldIndexAccur);
128 |     // Enlève le caractère de fin de ligne
129 |     accurParse = accurParse.Replace(END_LINE, string.Empty);
130 | }
```

Figure 207 Récupération des points de précision en fin de lignier de la game actuellement traitée (Data.cs)

Lors de cette dernière récupération, on retire le caractère de fin de ligne qui est également récupéré, afin de récupérer uniquement les points de précision.

Puis on finit par sauvegarder les points de précision dans notre objet Player actuel.

```
132 | // Assigne le niveau de précision actuel du mode actuel
133 | allPlayers[actualPlayerID].allModes[allPlayers[actualPlayerID].gameMode - 1].scoresPoints[1].Add(
134 |     int.Parse(accurParse));
135 |
136 | }
```

Figure 208 Sauvegarde des points de précision de la game actuellement traitée dans l'objet du Player actuel (Data.cs)

Il nous reste plus qu'à reset les valeurs de position, et de déchiffrer la prochaine ligne.

```
137 | // Reset l'index de l'ancienne valeur de l'index du caractère séparateur de score
138 | oldIndexScore = 0;
139 | // Reset l'index de l'ancienne valeur de l'index du caractère séparateur de précision
140 | oldIndexAccur = 0;
141 | }
```

Figure 209 Reset des positions des caractères séparateurs des points de score et de précision (Data.cs)

4.4. Graphique de progression

4.4.1. Paramétrage du graphique

Lors du démarrage du Software, les initialisations du graphique sont également faites.

```
69 // Création et paramétrage du graphique de progression
70 CreatCharType();
```

Figure 210 Initialisation des paramétrages du graphique de progression (View.cs)

Dans ce graphique de progression, beaucoup d'informations vont apparaître au même endroit. Un concept a été réalisé en amont du développement du software.

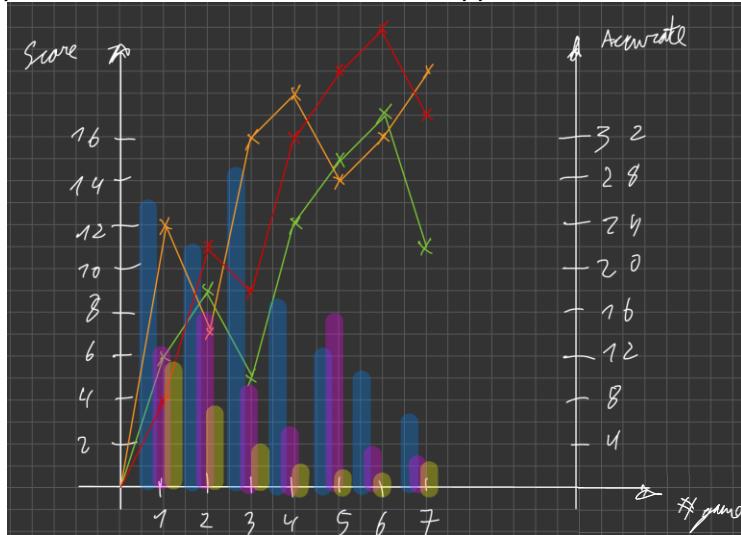


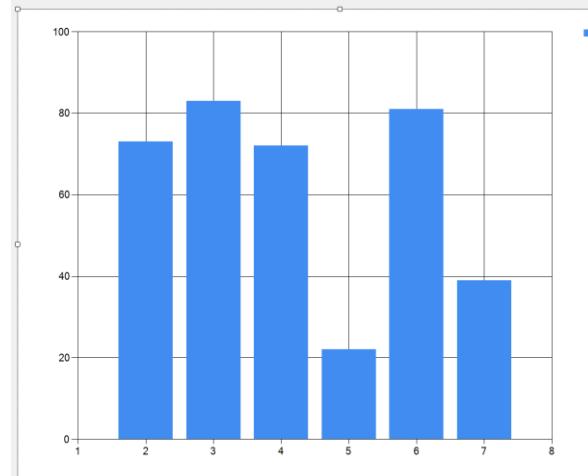
Figure 211 Concept de graphique de progression

On à donc sur l'axe horizontal, le nombre de games faite par le joueur sélectionné. Puis sur l'axe vertical de gauche, on a le nombre de points de score effectués. Ensuite, sur l'axe vertical de droite on a le nombre de points de précision effectués.

Sur cette grille on a ensuite trois courbes de type ligne, qui représentent les points de scores des trois modes, verte pour Easy, orange pour Hard, et rouge pour Extrem. Puis de la même manière, on a les trois courbes de type colonne pour les points de précision. La première à gauche pour Easy, celle du milieu pour Hard, et celle à gauche pour Extrem.

Ce graphique permet de centraliser toutes els données d'un seul joueur pour le mode de jeu aléatoire, pour les trois difficultés et pour les points des score et de précision.

Afin de pouvoir modifier un graphique, il faut tout d'bor en créer un. Ici il a été fait via l'interface graphique de Windows Form, depuis la boîte à outils il faut sélectionner « Chart ».



Avant de commencer tout paramétrage, on va d'abord supprimer la courbe qui est paramétrée par défaut.

```

200          // Crée et paramètre le graphique de progression
201      1 référence
201      public void CreatCharType()
202      {
203          // Efface toute les série déjà paramétrées
204          progressGraph.Series.Clear();

```

Figure 213 Suppression de la courbe par défaut du graphique (View.cs)

Ensuite on va commencer par ajouter nos 6 courbes au graphique, en indiquant leurs noms.

```

206          // Crée toutes les courbes dans l'ordre de derier vers l'avant en Z
207          progressGraph.Series.Add("AccurateEasy");
208          progressGraph.Series.Add("AccurateHard");
209          progressGraph.Series.Add("AccurateExtrem");
210          progressGraph.Series.Add("ScoreEasy");
211          progressGraph.Series.Add("ScoreHard");
212          progressGraph.Series.Add("ScoreExtrem");

```

Figure 214 Ajout des 6 courbes au graphique de progression (View.cs)

Puis c'est le type de la courbe que l'on va pouvoir choisir, dans notre cas on veut 3 courbes de type ligne pour les points de scores, et 3 courbes de type colonnes pour les points de précision.

```

214          // Paramétrage des trois courbes linéaires de progression du score
215          progressGraph.Series["ScoreEasy"].ChartType = SeriesChartType.Line;
216          progressGraph.Series["ScoreHard"].ChartType = SeriesChartType.Line;
217          progressGraph.Series["ScoreExtrem"].ChartType = SeriesChartType.Line;
218          // Paramétrage des trois courbes en colonnes de progression de précision
219          progressGraph.Series["AccurateEasy"].ChartType = SeriesChartType.Column;
220          progressGraph.Series["AccurateHard"].ChartType = SeriesChartType.Column;
221          progressGraph.Series["AccurateExtrem"].ChartType = SeriesChartType.Column;

```

Figure 215 Configuration du type des courbes (View.cs)

Ensuite des couleurs différentes ont été assignées à chaque courbe.

```

223          // Paramétrage des couleurs de chaque courbe
224          progressGraph.Series["ScoreEasy"].Color = Color.Green;
225          progressGraph.Series["ScoreHard"].Color = Color.Orange;
226          progressGraph.Series["ScoreExtrem"].Color = Color.Red;
227          progressGraph.Series["AccurateEasy"].Color = Color.Blue;
228          progressGraph.Series["AccurateHard"].Color = Color.Purple;
229          progressGraph.Series["AccurateExtrem"].Color = Color.Black;

```

Figure 216 Configuration des couleurs des courbes (View.cs)

Puis pour des questions de lisibilité, les courbes de type ligne ont été épaissies.

```

231          // Modification de l'épaisseur des courbes de score
232          progressGraph.Series["ScoreEasy"].BorderWidth = chartLineThickness;
233          progressGraph.Series["ScoreHard"].BorderWidth = chartLineThickness;
234          progressGraph.Series["ScoreExtrem"].BorderWidth = chartLineThickness;

```

Figure 217 Configuration de l'épaisseur des courbes de type ligne (View.cs)

Afin de pouvoir avoir un deuxième axe à la verticale, on doit l'ajouter au graphique de base.

```

236          // Activation de l'axe vertical secondaire
237          progressGraph.ChartAreas[0].AxisY2.Enabled = AxisEnabled.True;

```

Figure 218 Création du deuxième axe vertical (View.cs)

Une fois créé, on peut y assigner les courbes que l'on veut, dans notre cas c'est les courbes de type colonne qui représentent les points de précision du joueur.

```
238 // Assigneemnt des courbes en colonnes sur l'axe secondaire en vertical
239 progressGraph.Series["AccurateEasy"].YAxisType = AxisType.Secondary;
240 progressGraph.Series["AccurateHard"].YAxisType = AxisType.Secondary;
241 progressGraph.Series["AccurateExtrem"].YAxisType = AxisType.Secondary;
```

Figure 219 Ajout des courbes de type colonne sur l'axe secondaire vertical (View.cs)

Comme la pire moyenne de précision que l'on peut faire est 100%, l'axe secondaire vertical a donc été paramétré avec une limite de 100 maximum

```
242 // Paramétrage d'un valeur maximum de l'axe vertical secondaire
243 progressGraph.ChartAreas[0].AxisY2.Maximum = 100;
```

Figure 220 Fixage d'une limite de 100 sur l'axe secondaire vertical (View.cs)

Pour une meilleure distinction des différents très de la grille, les traits verticaux ont été transformés en traitillés, et le traits horizontaux en pointillés.

```
245 // Paramétrage du type de traits horizontaux de la grille du graphique
246 progressGraph.ChartAreas[0].AxisY.MajorGrid.LineDashStyle = ChartDashStyle.Dash;
247 progressGraph.ChartAreas[0].AxisY2.MajorGrid.LineDashStyle = ChartDashStyle.Dash;
248 // Paramétrage du type de traits verticaux de la grille du graphique
249 progressGraph.ChartAreas[0].AxisX.MajorGrid.LineDashStyle = ChartDashStyle.Dot;
```

Figure 221 Modification du type de traits des lignes horizontales et verticales de la grille du graphique (View.cs)
La couleur des traitillés horizontaux de la grille ont également été changés en gris et rouge.

```
251 // Paramétrage de la couleur des traits horizontaux de la grille du graphique
252 progressGraph.ChartAreas[0].AxisY.MajorGrid.LineColor = Color.Gray;
253 progressGraph.ChartAreas[0].AxisY2.MajorGrid.LineColor = Color.Red;
```

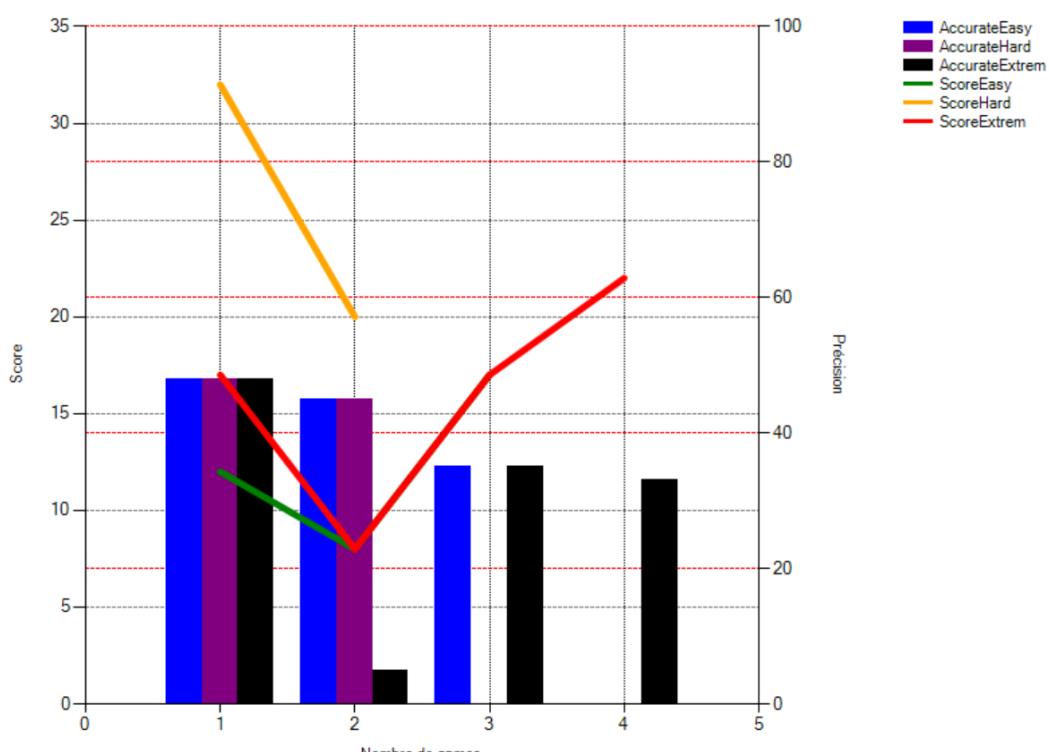
Figure 222 Modification de la couleurs des traitillés horizontaux en gris et rouge

Puis pour finir, on va nommer les axes, afin d'indiquer à l'utilisateur à quoi ils correspondent.

```
255 // Titre des axes
256 progressGraph.ChartAreas[0].AxisX.Title = "Nombre de games";
257 progressGraph.ChartAreas[0].AxisY.Title = "Score";
258 progressGraph.ChartAreas[0].AxisY2.Title = "Précision";
259 }
```

Figure 223 Ajout d'un nom au axes du graphique (View.cs)

Ce qui donne au final un graphique comme l'exemple ci-dessous.



Ricardo Crespo Figure 224 Exemple de graphique de progression (2208_JeuSimon.exe)

4.4.2. Sélection des Players

Lorsque l'on démarre le software, et que l'on connecte le Simon, il nous reste plus qu'à sélectionner le Player pour lequel on veut voir sa courbe de progression.

Pour cela on utilise une liste déroulable, qui se prénom « ListBox » dans la boîte à outils de Windows Form.



Figure 225 Liste déroulable de Windows Form pour la sélection du Player (View.Designer.cs)

C'est qu'une fois que le tableau des scores complètement déchiffré, que la liste se remplira.

```

319     // Ajoute tous les pseudos des Player dans la liste déroulante
320     public void AssignAllPlayerToList()
321     {
322         // Ajoute tous les Players qui sont dans la table des scores
323         for (int i = 1; i < (dataProcess.allPlayers.Count); i++)
324         {
325             // S'il y a besoin d'invoquer la liste box
326             if (this.InvokeRequired)
327             {
328                 // Invoque la liste box et ajouter les pseudos de tous les Players dans le menu déroulant
329                 this.Invoke((MethodInvoker)(() => lbUsernames.Items.Add(dataProcess.allPlayers[i].playerName)));
330             }
331             // Si non on a déjà accès à la liste box
332             else
333             {
334                 // Ajouter les pseudos de tous les Players dans le menu déroulant
335                 lbUsernames.Items.Add(dataProcess.allPlayers[i].playerName);
336             }
337         }
338     }

```

Figure 226 Assignment de tous les pseudos des Player du tableau des scores dans la liste déroulante (View.cs)

D'un à l'architecture de données que l'on a utilisé pour sauvegarder les données du tableau des scores, il est donc très simple de reprendre l'information que l'on a besoin.

Ici on boucle simplement la liste de Player que l'on a récupéré, et on ajoute chaque pseudo dans la liste déroulable.

Suite à des problèmes d'accès à la list box, un contrôl pour vérifier que l'on a l'accès est fait. Si on le l'a pas, on crée l'accès.

Finalement on a donc une liste avec tous les pseudos des différents joueurs qui sont présents dans le tableau des scores.

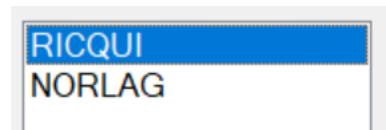


Figure 227 Liste déroulante avec deux Players sélectionnables (2208_JeuSimon.exe)

4.4.3. Mise à jour des données du graphique

Lorsque l'on sélectionne un Player dans la liste déroulant, un évènement se crée et on récupère l'index de la sélection, qui est exactement le même que l'ID du Player qui a été placé.

```
340 // Évènement d'un changement de selection de pseudo dans la liste déroulante
341 1 référence
342 private void lbUsernames_SelectedIndexChanged(object sender, EventArgs e)
343 {
344     // Met à jour les courbes du graphique de progression avec celles du Player sélectionné
345     UpdateChart(lbUsernames.SelectedIndex);
}
```

Figure 228 Evènement de changement de sélection de Player dans la liste déroulante (View.cs)

Lors de cet événement, on va envoyer l'ID du Player sélectionné à la méthode qui met à jour le graphique de progression, et on va commencer par supprimer les anciennes données.

```
261 // Met à jour les données du graphique avec le joueur actuellement sélectionné
262 1 référence
263 public void UpdateChart(int _playerID)
264 {
265     // Efface tous les points de toutes les courbes du précédent Player
266     ResetChart();
}
```

Figure 229 Reset des courbes lors de la mise à jour du graphique (View.cs)

Ici on va tout simplement supprimer tous les points des 6 courbes de notre graphique.

```
284 // Efface tous les points de toutes les courbes du précédent Player
285 1 référence
286 private void ResetChart()
287 {
288     // Essaye d'effacer les points des courbes si elles sont existantes
289     try
290     {
291         // Efface tous les points de toutes les courbes
292         progressGraph.Series["ScoreEasy"].Points.Clear();
293         progressGraph.Series["ScoreHard"].Points.Clear();
294         progressGraph.Series["ScoreExtrem"].Points.Clear();
295         progressGraph.Series["AccurateEasy"].Points.Clear();
296         progressGraph.Series["AccurateHard"].Points.Clear();
297         progressGraph.Series["AccurateExtrem"].Points.Clear();
298     }
299 }
```

Figure 230 Suppression de tous les points de toutes les courbes du graphique (View.cs)

Puis on va rechercher dans la liste de Player, lequel correspond à l'ID sélectionné.

```
267 // Récupération de l'objet du Player actuellement sélectionné depuis la liste de Player du tableau des scores
268 Player selectedPlayer = dataProcess.allPlayers.Find(x => x.playerID == _playerID + 1);
```

Figure 231 Recherche de l'ID sélectionné dans la liste de Players sauvegardée (View.cs)

Une fois trouvé, il nous reste plus qu'à retranscrire dans le graphique tous les points de toutes les courbes que l'on avait sauvegardées dans l'objet du Player sélectionné au moment du déchiffrage du tableau des scores.

```
270 // Traite les données de tous les modes de difficulté
271 for (int i = 0; i < NUMBER_OF_MODS; i++)
272 {
273     // Traite les données du mode de difficulté sélectionné pour le nombre de parties jouées par le Player
274     for (int j = 0; j < selectedPlayer.numberGames[(i < NUMBER_OF_MODS) ? i : (i - NUMBER_OF_MODS)]; j++)
275     {
276         // Ajoute les points sur les courbes de scores du graphique à partir de la liste des scores du mode sélectionné du Player sélectionné
277         progressGraph.Series[chartSerieses[i]].Points.AddXY(j + 1, selectedPlayer.allModes[i].scoresPoints[SCORE_DATA][j]);
278         // Ajoute les points sur les courbes de précision du graphique à partir de la liste de précision du mode sélectionné du Player sélectionné
279         progressGraph.Series[i + NUMBER_OF_MODS].Points.AddXY(j + 1, selectedPlayer.allModes[i].scoresPoints[ACCUR_DATA][j]);
280     }
281 }
}
```

Figure 232 Mise à jour des courbes avec les données des points de score et de précision du Player sélectionné (View.cs)

4.4.4. Exemples de vues

Ici on a un premier exemple lorsque l'on sélectionne le Player « RICQUI ».

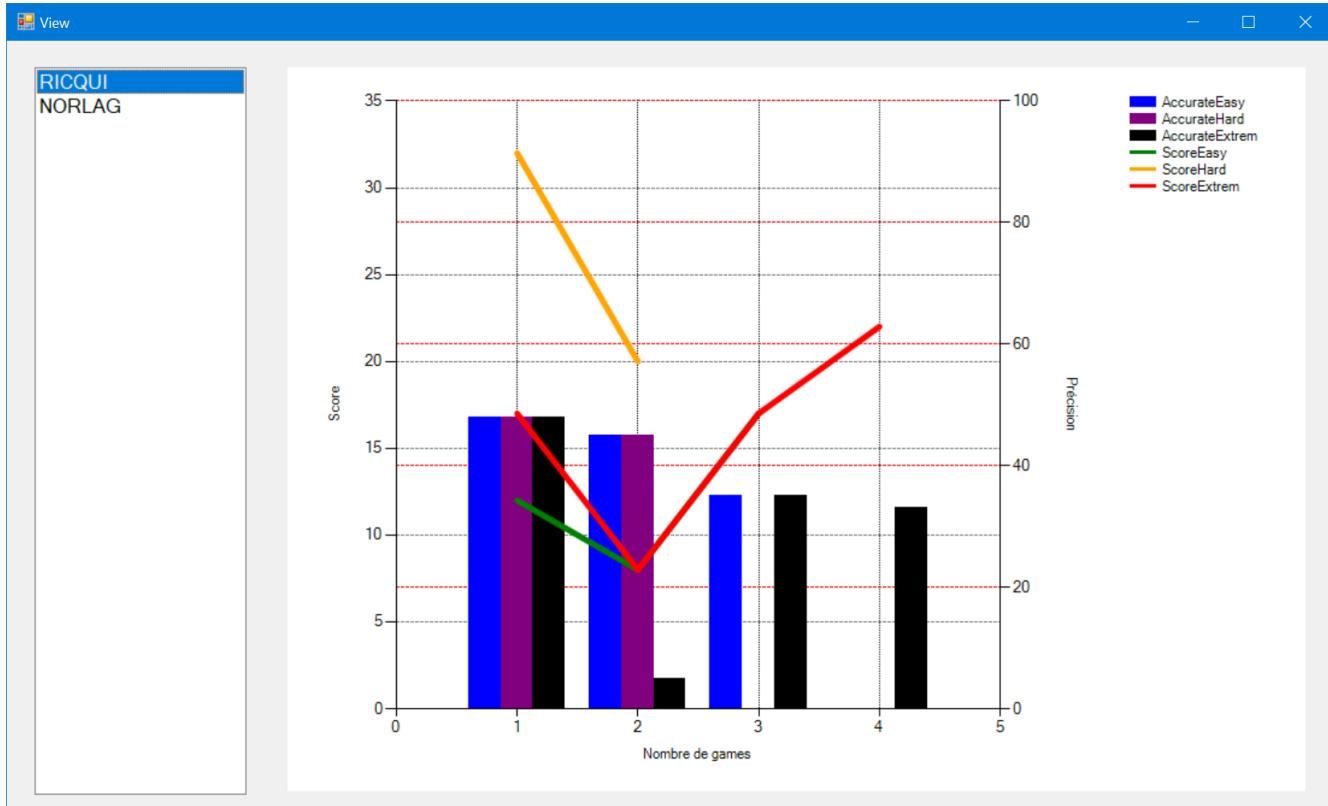


Figure 233 Exemple de graphique de progression pour la première sélection (2208_JeuSimon.exe)

Puis lorsque l'on sélectionne le Player « Norlag ».

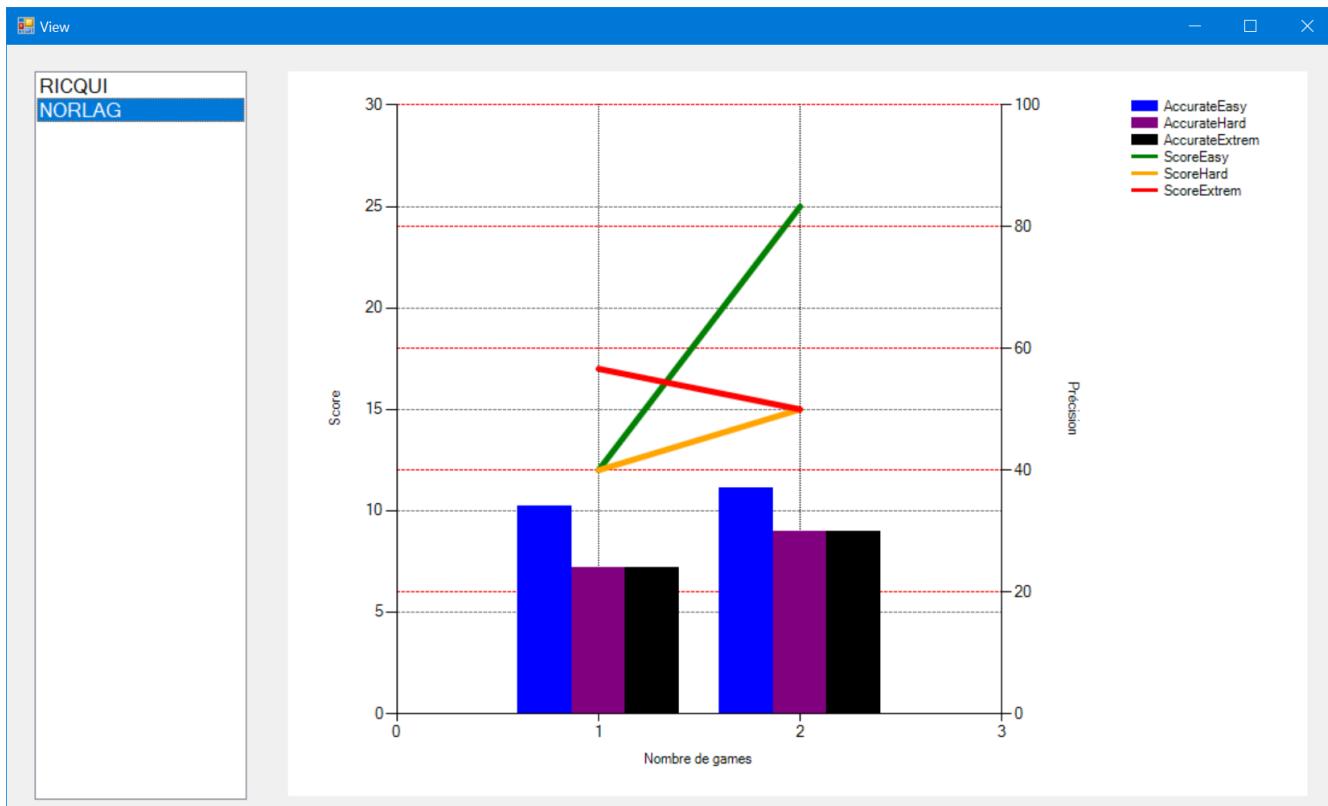


Figure 234 Exemple de courbe de progression pour la deuxième sélection (2208_JeuSimon.exe)

5. Test et mesures

5.1. Appareils de mesure

| | | | |
|------|--------------|------------------|------------------|
| • P1 | Oscilloscope | ROHDE&SCHWARZ | ES.SLO2.05.01.12 |
| • P2 | Multimètre | Gwinstek GDM-396 | ES.SLO2.00.00.77 |
| • U1 | Alimentation | Sefram 6330 | ES.SLO2.00.00.24 |

5.2. Alimentations

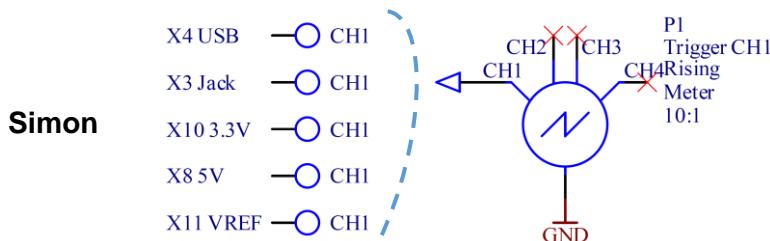


Figure 235 Schéma de mesure des alimentations

| | Input [V] | 3.3V [V] | 5V [V] | Erreur Absolue Input [mV] | Erreur Absolue 3.3V [mV] | Erreur Absolue 5V [mV] |
|---------------------|-----------|----------|--------|---------------------------|--------------------------|------------------------|
| USB | 5 | 3,28 | 5,05 | 0 | 20 | 50 |
| Jack | 5,04 | 3,29 | 5,05 | 40 | 10 | 50 |
| Accumulateur | 3,57 | 3,28 | 5,05 | - | 20 | 50 |

On peut remarquer que toutes les alimentations sont très proches des valeurs voulues. Pour les petits écarts restants, ils sont très certainement dus à la précision des composants passifs constituant ces montages, qui sont de l'ordre de 1%.

Malheureusement, ne disposant pas d'appareils de mesure suffisamment précis, la mesure de la référence du DAC12 bit a été seulement contrôlée à 4.11V.

5.3. Timers

Les mesures des Timers ont été faites grâce au toggle de pins de monitoring dans le Firmware.

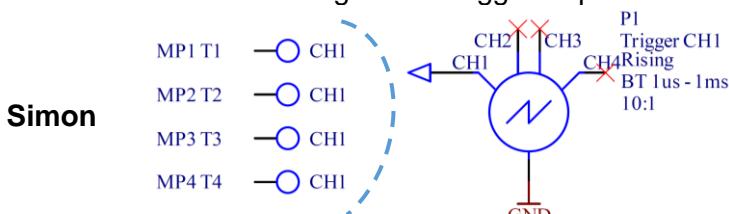


Figure 236 Schéma de mesure des Timers

| | Période [ms] | Erreur Absolue [us] |
|---------------|--------------|---------------------|
| Timer1 | 1 | 0 |
| Timer2 | 0,01 | 0 |
| Timer3 | 0,05966 | 9 |
| Timer4 | 10 | 0 |

On peut remarquer que l'on est précis presque à la perfection. La seul erreur de mesure est dû à une certaine limite de précision de l'oscilloscope, notamment au niveau du placement des curseurs.

5.4. Debouncing butons poussoir

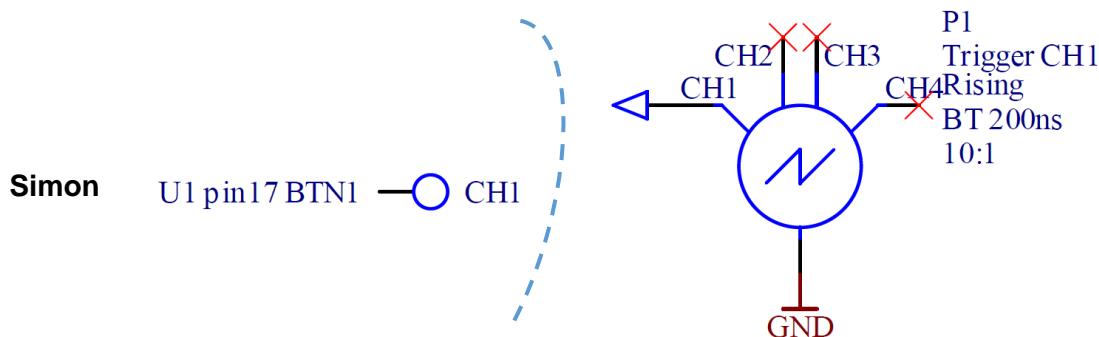


Figure 237 Schéma de mesure du debouncing du bouton poussoir BTN1

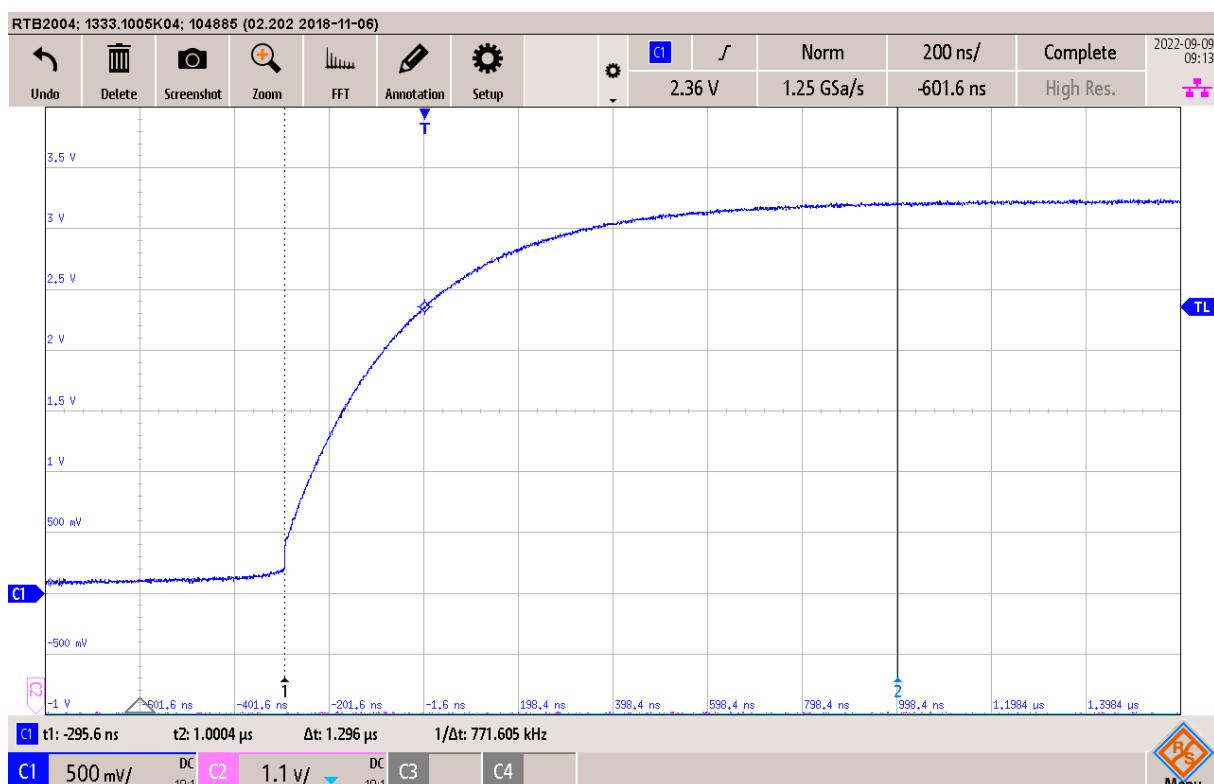


Figure 238 Debouncing d'un appui sur le bouton poussoir

Ici on peut remarquer que le bouton poussoir prend environ 1.296μs pour passer de l'état inactif, à l'état actif. Mais on peut également remarquer qu'il n'y a pas de rebonds notables,

5.5. PEC12

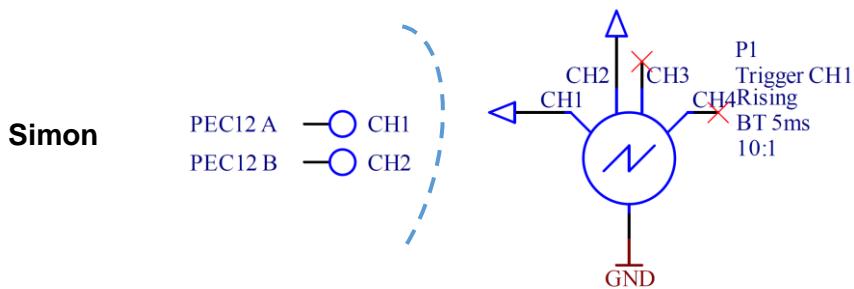


Figure 239 Schéma de mesure des signaux A et B du PEC12

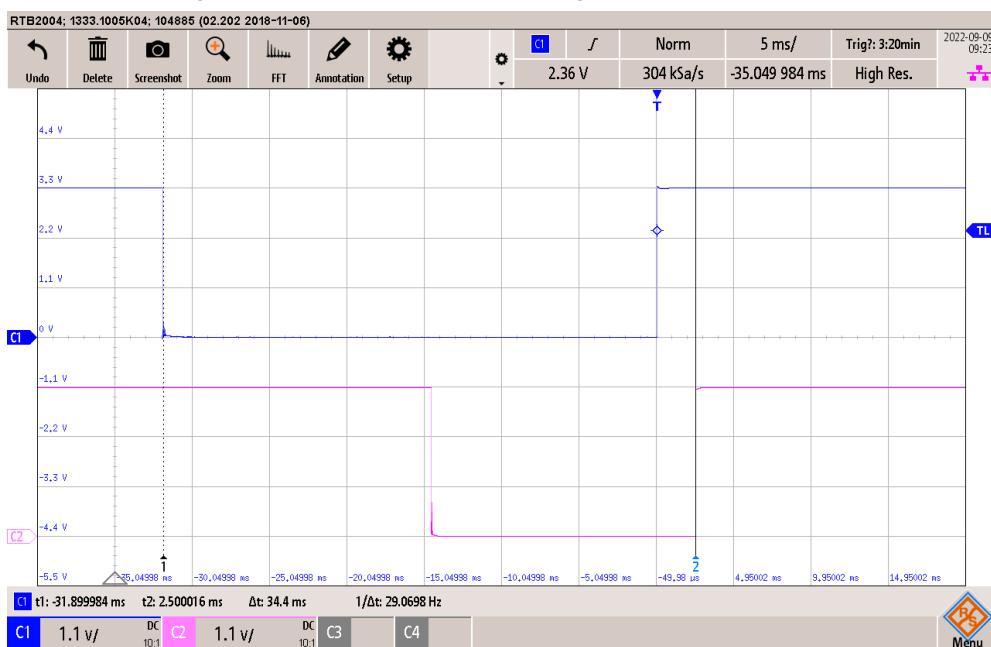


Figure 240 Sens CW avec l'état bas sur A quand B est à l'état haut

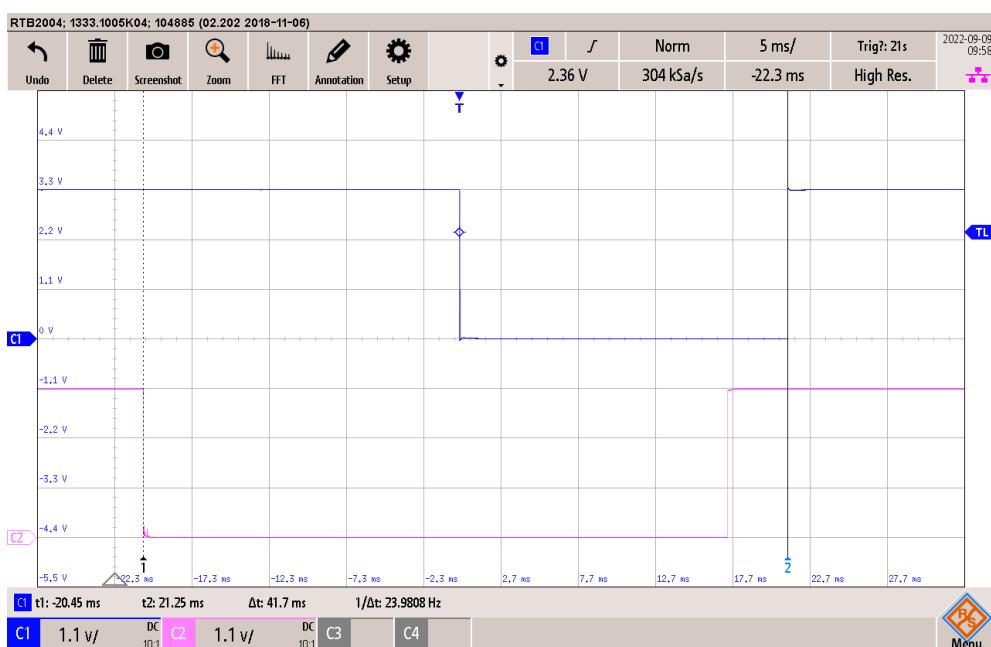


Figure 241 Sens CCW avec l'état bas sur A quand B est à l'état haut

Mesure juste à titre indicatif pour expliquer le sens en fonction des états logiques.

5.6. UART2

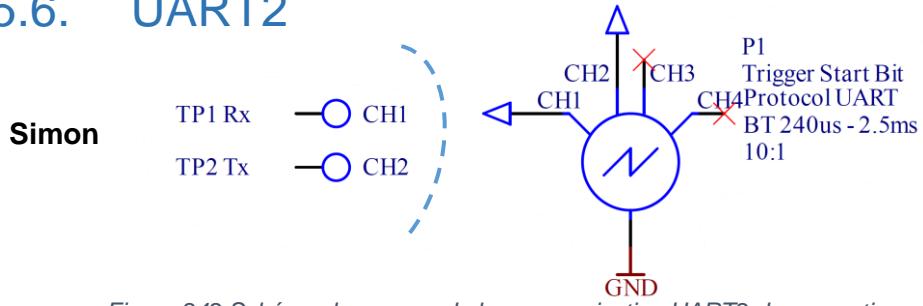


Figure 242 Schéma de mesure de la communication UART2 du convertisseur USB

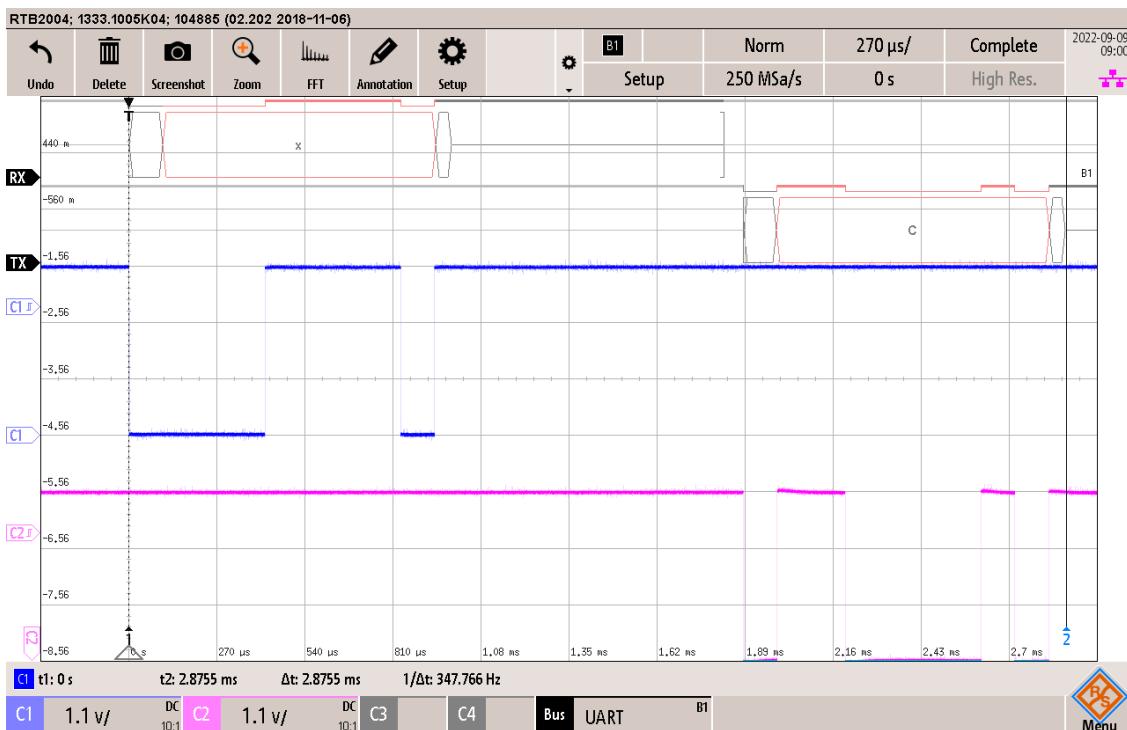


Figure 243 Clé « x » de reconnaissance (PC) et clé « C » d'annoncement (uC)

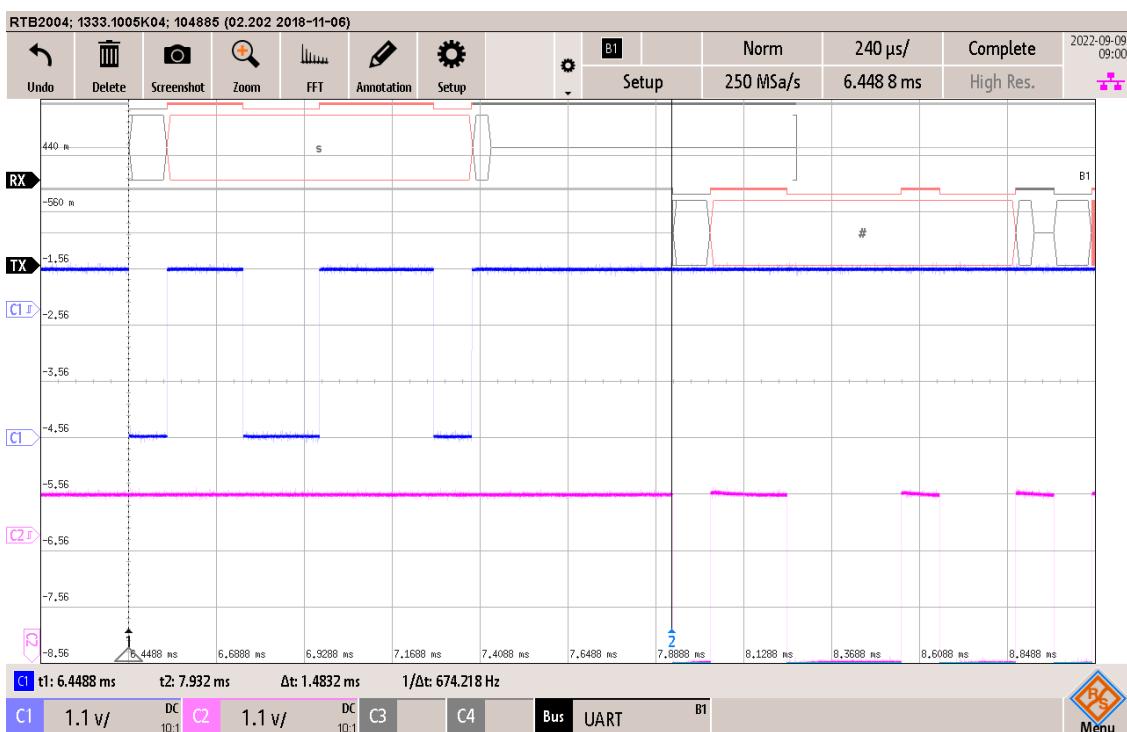


Figure 244 Clé « s » de demande d'envoi du tableau des scores (PC) et début de l'envoi du tableau des scores

Une fois les clés échangées, c'est le tableau des scores qui est envoyé au complet.

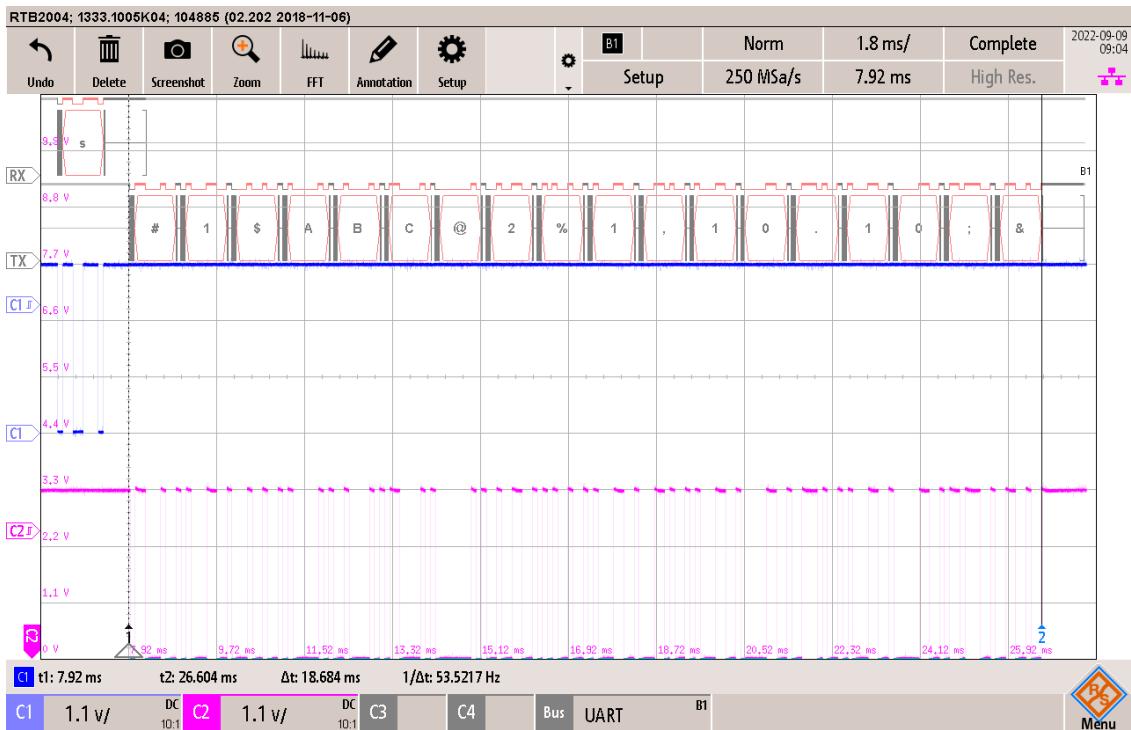


Figure 245 Envoie du tableau des scores du uC au PC

C'est donc un exemple de tableau des scores, avec un player avec l'ID n°1, avec le pseudo « ABC », avec une game en mode « Extreme » et un score et erreur de 10 chacun.

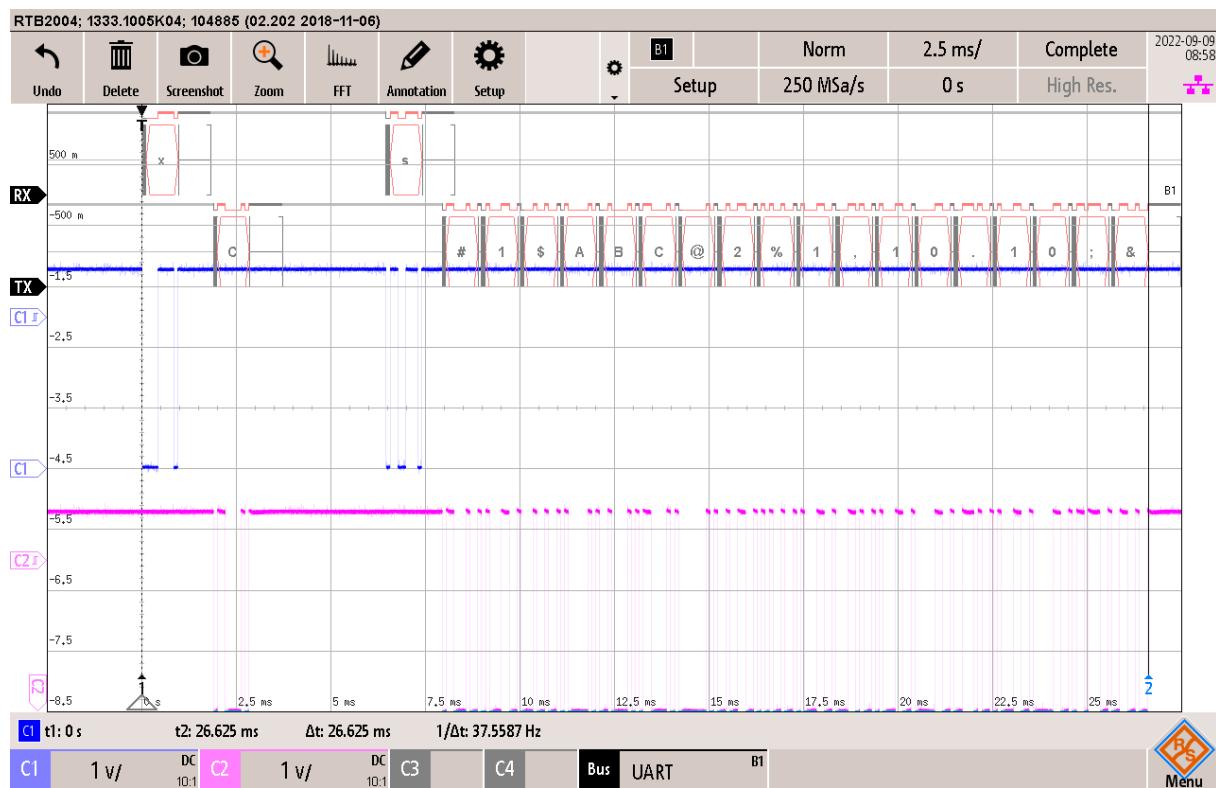


Figure 246 Temps total de la communication total du tableau des scores

Toute la communication avec les clés et le tableau des scores prend un total de 26.625ms à être envoyé.

5.7. SPI1 Driver de LED

5.7.1. Initialisation

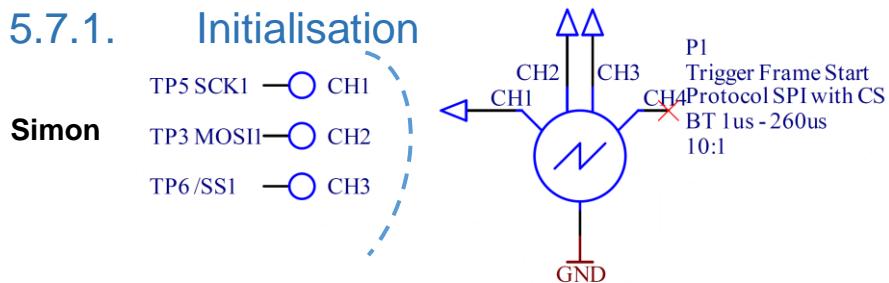


Figure 247 Schéma de mesure du SPI1 du Driver de LEDs

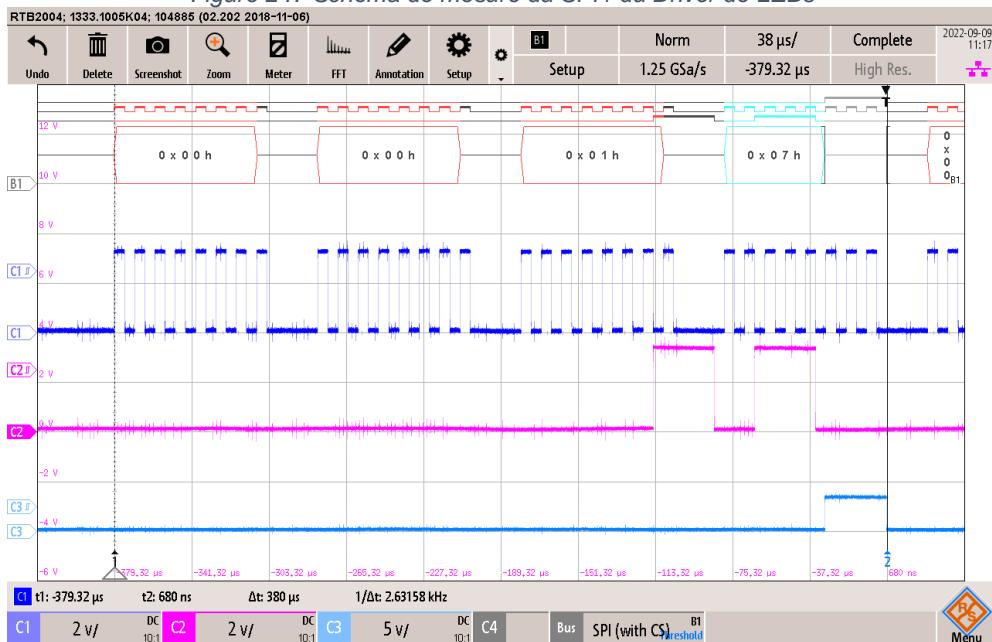


Figure 248 Initialisation du registre de configuration

La Latch a une durée de 3 pulses positives, ce qui donne accès au registre de configuration.



Figure 249 Initialisation du registre de gain

Après la configuration, un reset des LED et une configuration des registres est faite de nouveau, cela est fait car des LED restaient allumées occasionnellement au reset du uC.

5.7.2. Commande LEDs



Figure 250 Commande d'allumage de toutes les LEDs en rouge

Ici c'est la commande d'allumage de toutes les LED en rouge. Avec une Latch qui a une durée de 2 pulses positives, ce qui donne accès au registre de gain.

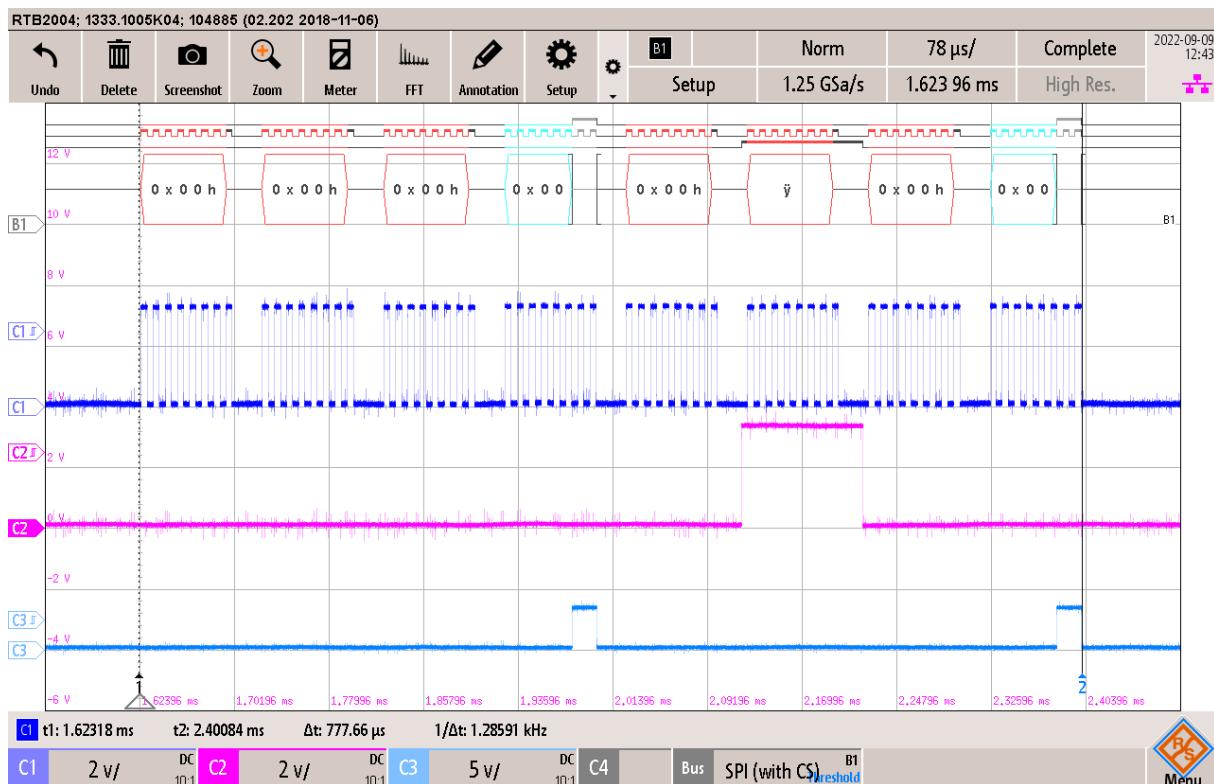


Figure 251 Reset des LED suivie de l'allumage de toutes les LEDs en rouge

Ici on voit un exemple de reset avant d'afficher une seule couleur sur toutes les LED.

5.8. SPI2

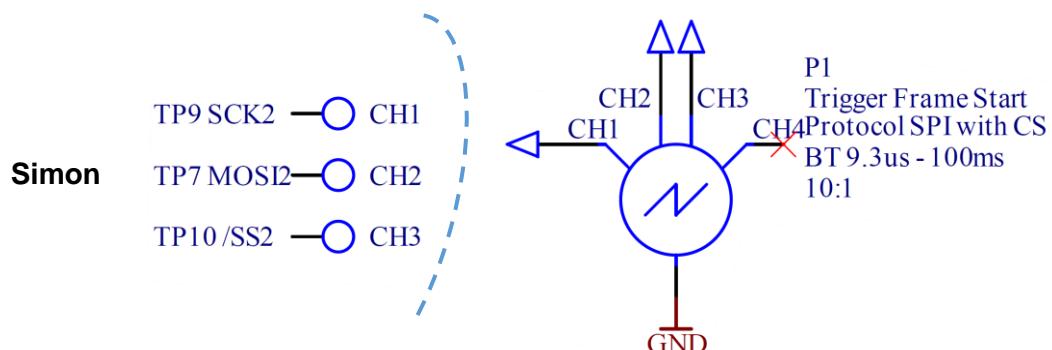


Figure 252 Schéma de mesure du SPI2 du DAC12 bit

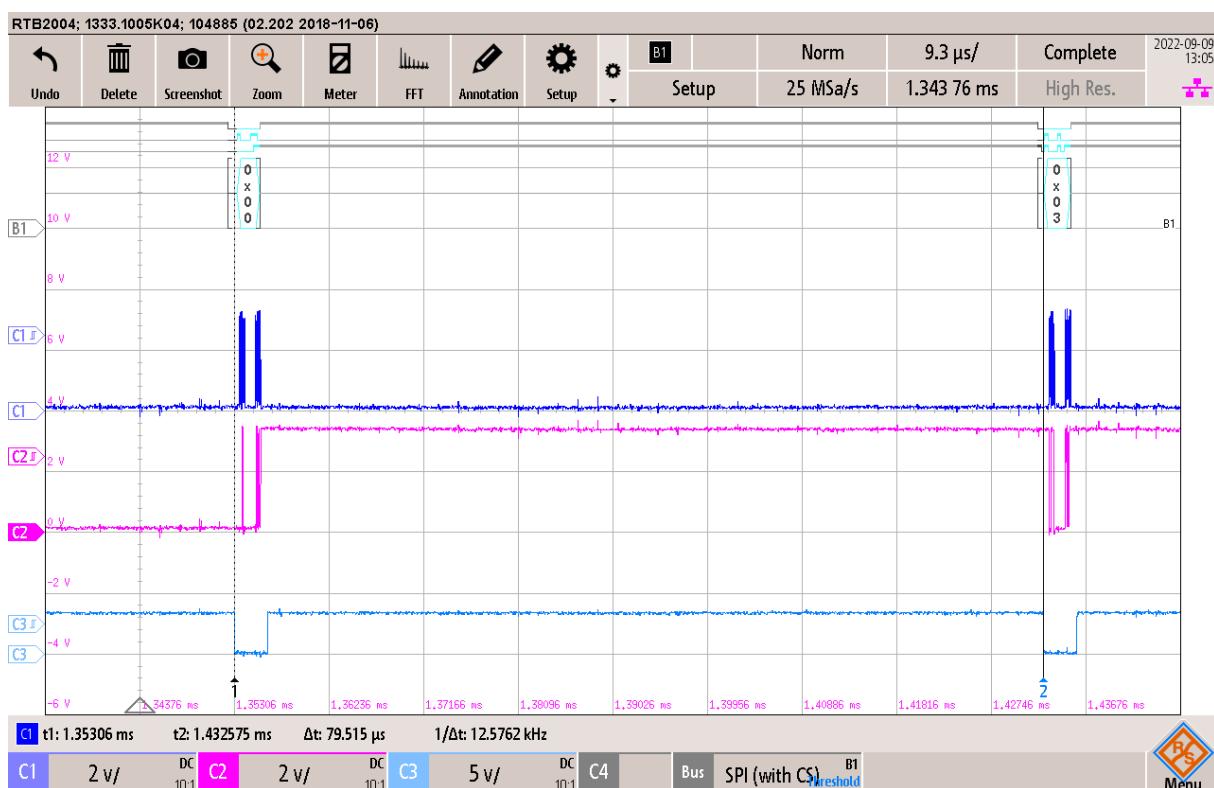


Figure 253 Temps entre deux commandes d'échantillons du DAC12 pour générer un DO

$$T_{ECH\ DO} = \frac{1}{f_{DO}} = \frac{1}{1047} = 79.59\mu s$$

Comme vu précédemment, nous avons une fréquence qui est 12 fois plus grande que celle que l'on veut générer, car on génère 12 échantillons par période.

C'est uniquement la valeur brute qui est directement envoyée au DAC 12bit. Dans ce cas on voit un échantillon à 0x00 et le prochain à 0x03.

Lors de l'animation ou de l'appui d'une touche, un son est émis, il faut donc le générer pendant le temps alloué en fonction du mode de difficulté choisi.

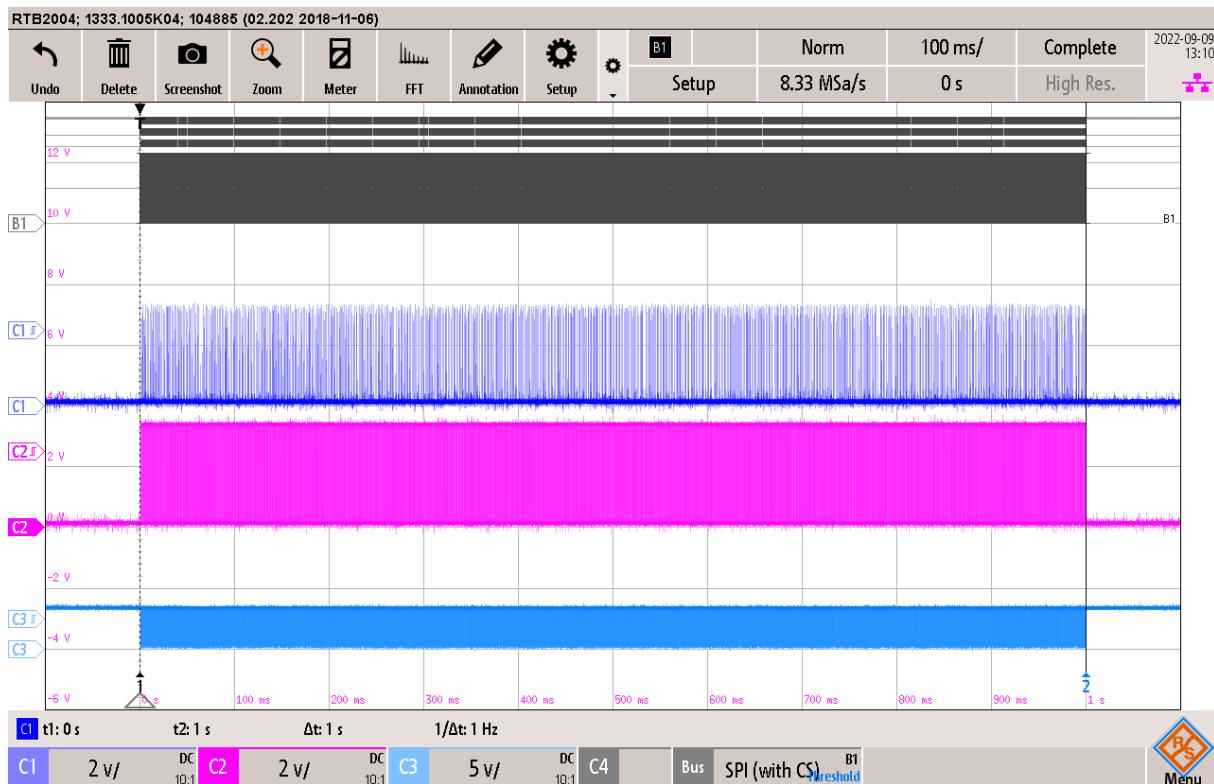


Figure 254 Temps d'interruption de 60bpm pour le mode Easy

| | Fréquence Théorique [Hz] | Période [ms] | Erreur Absolue [ms] |
|---------------|--------------------------|--------------|---------------------|
| 60bpm | 1 | 1 | 0 |
| 80bpm | 1,3 | 0,75 | 0 |
| 100bpm | 1,6 | 0,6 | 0 |

Ici on peut donc remarquer que les tempos sont respectés.

5.9. I2C

Simon

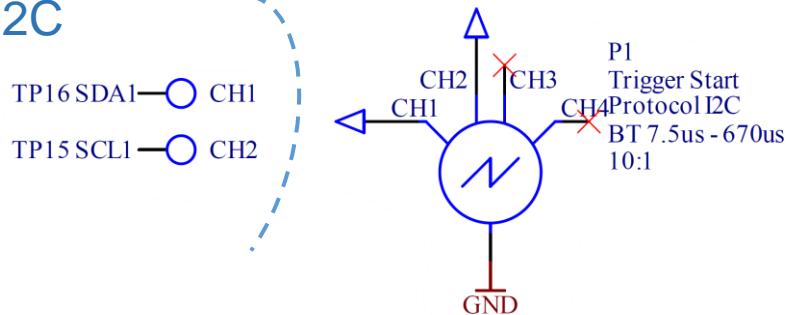


Figure 255 Schéma de mesure du I2C du LCD



Figure 256 Envoi de la première commande au LCD

Ici on retrouve la configuration du LCD, d'abord l'adresse du LCD « 0x3C », suivi de l'adresse du registre de commandes « 0x00 », puis du byte de donné « 0x3A ».

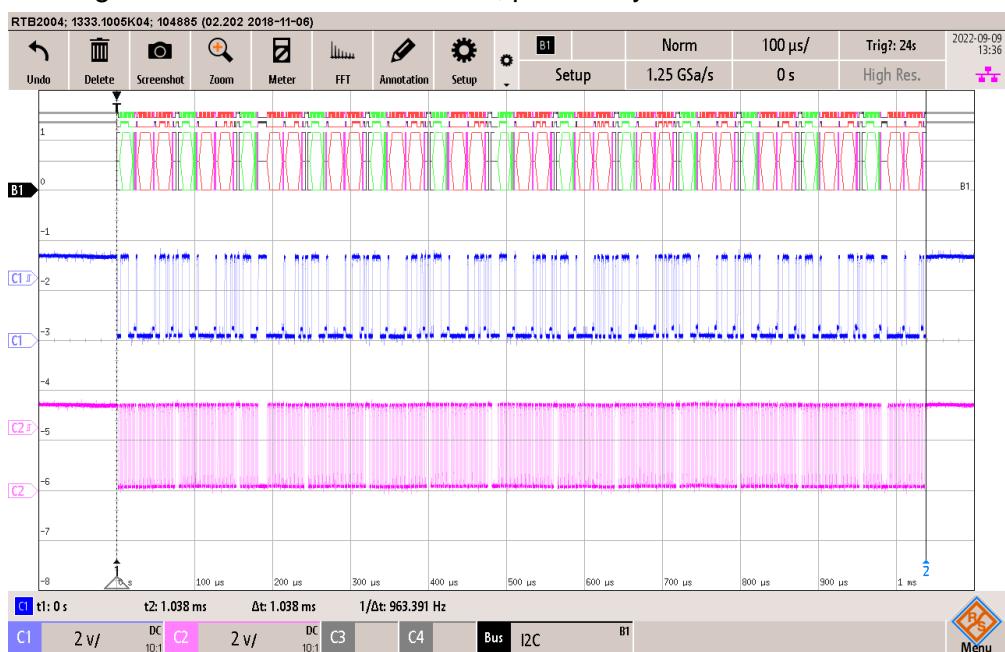


Figure 257 Temps total d'initialisation du LCD de 1.038ms

Pour réaliser toute l'initialisation du LCD, il nous faut au total 1.038ms.

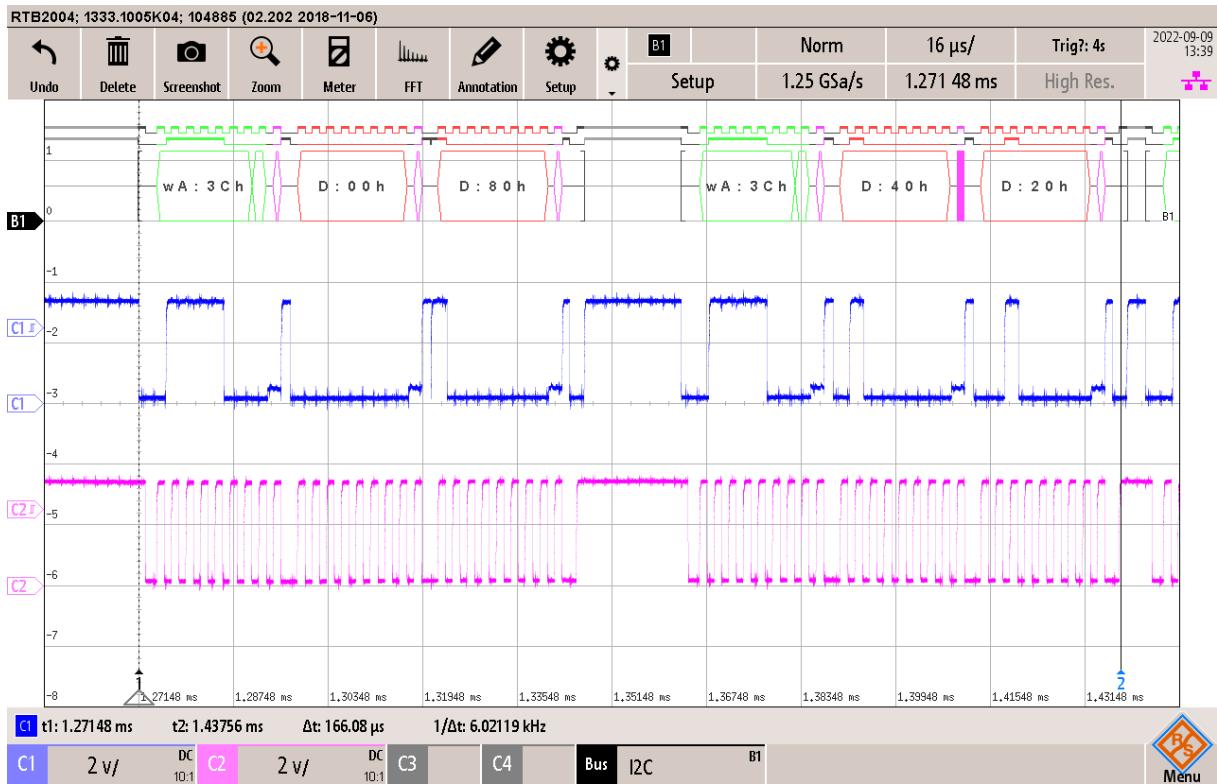


Figure 258 Envoi d'une commande, suivi de l'écriture d'un espace sur le LCD

Ici on retrouve la commande de GoTo pour déplacer le curseur d'écriture, suivi de l'écriture d'un espace.

Cette fois-ci, l'adresse d'écriture de date est « 0x40 »

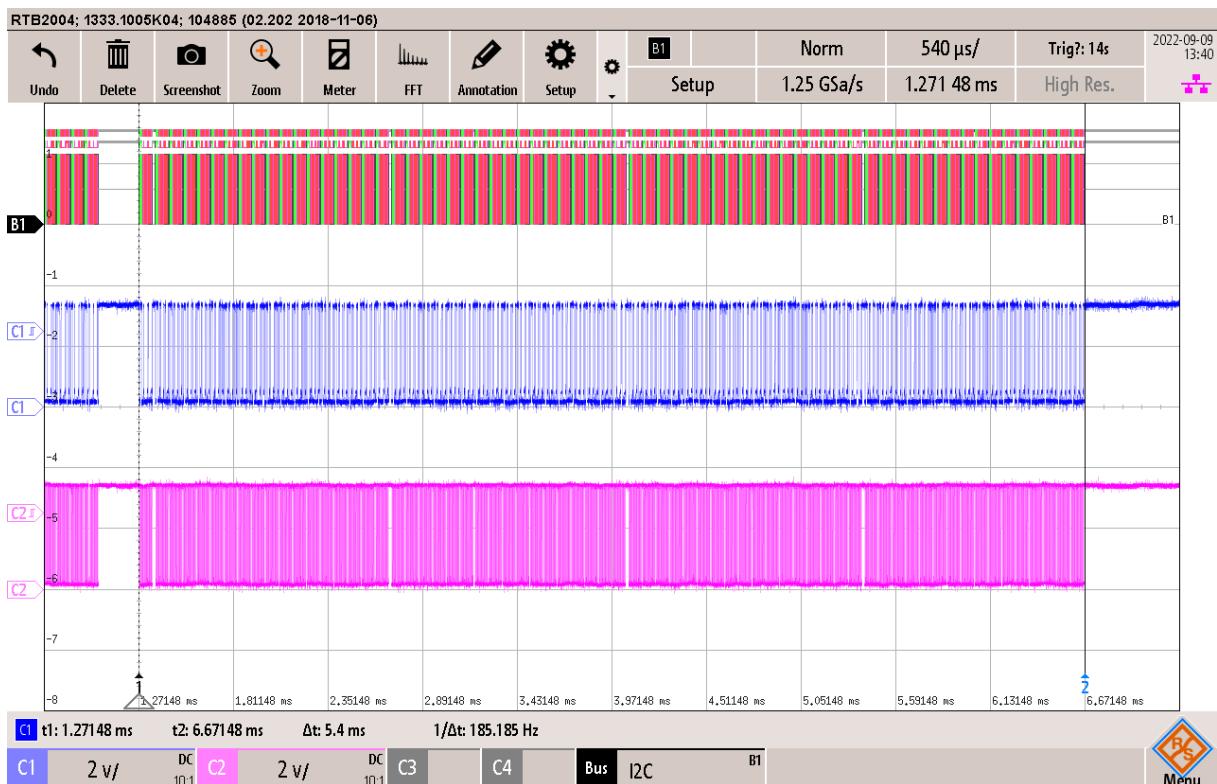


Figure 259 Temps total d'envoi du premier menu d'accueil du Simon de 5.4ms

Afin d'afficher tout le premier menu d'accueil du Simon, il nous faut 5.4ms.

5.10. Son et notes

Simon

TP28 Speaker

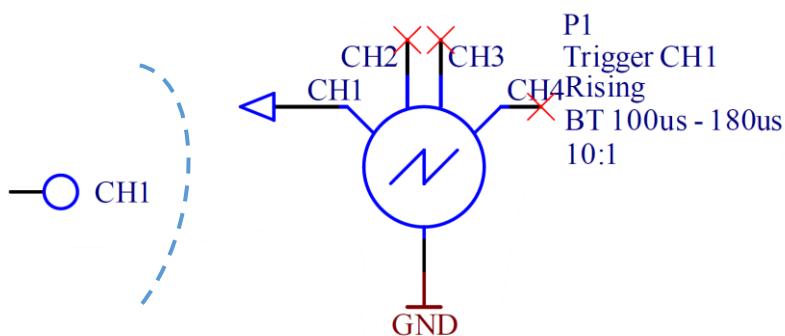


Figure 260 Schéma de mesure de la fréquence des notes de musique

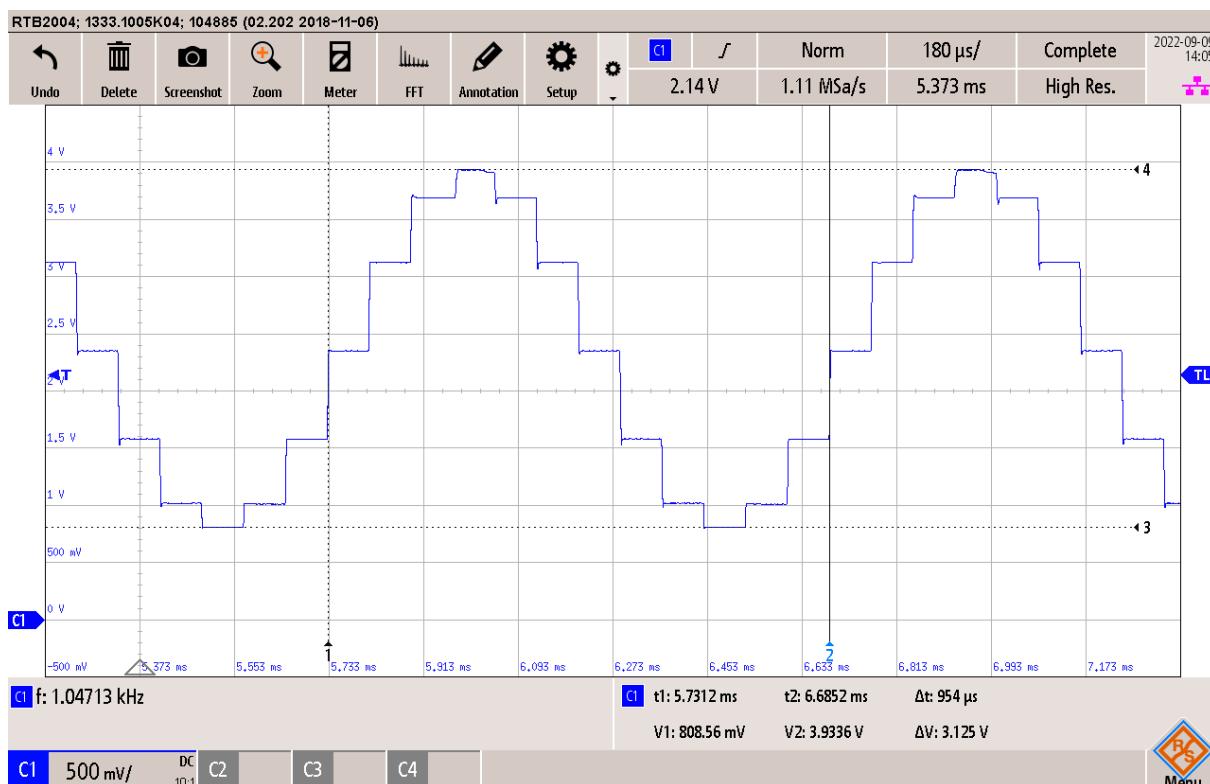


Figure 261 Période généré pour la note DO

Ici on peut recompter le nombre d'échantillons, et on retrouve nous 12 que l'on par période.

| | DO | RE | MI | FA | SOL | LA | SI |
|--------------------------|------|-------|-------|-------|-------|------|-------|
| Fréquence Théorique [Hz] | 1047 | 1175 | 1319 | 1397 | 1568 | 1760 | 1976 |
| Période mesurée [us] | 954 | 851,7 | 758,8 | 716,3 | 637,2 | 568 | 506 |
| Erreur Absolue [us] | 1,1 | 0,64 | 0,65 | 0,48 | 0,55 | 0,18 | 0,073 |

Au final, grâce au quartz nous avons uniquement un écart de quelques nanosecondes pour les fréquences de chaque note.

On peut également remarquer qu'au final, après l'abaissement dû à la référence du DAC 12bit à 4.096V, et le fait que l'AOP suiviteur n'est pas Rail-to-Rail, on se retrouve avec une amplitude du signal à 3.125V.

5.11. Charge de l'accumulateur et consommation

Simon

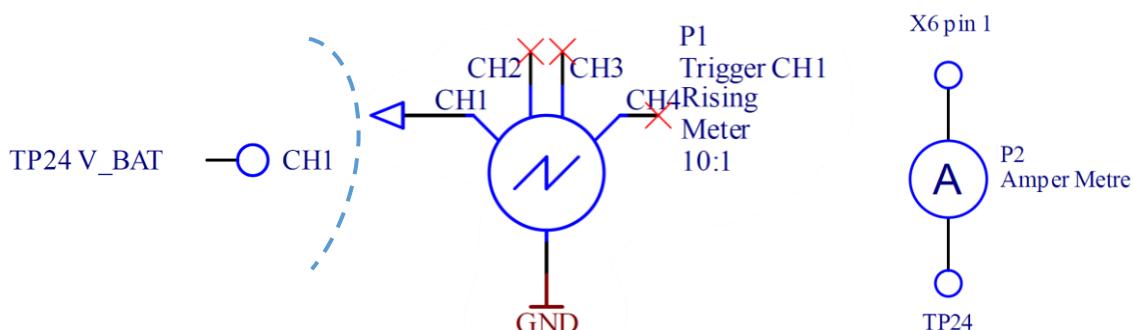


Figure 262 Mesure de la charge de l'Accumulateur et de la consommation de courant

Afin de pouvoir mesurer la charge de l'accumulateur, on utilise la pin 100 du microcontrôleur, afin de commander le circuit logique avec les deux transistors.

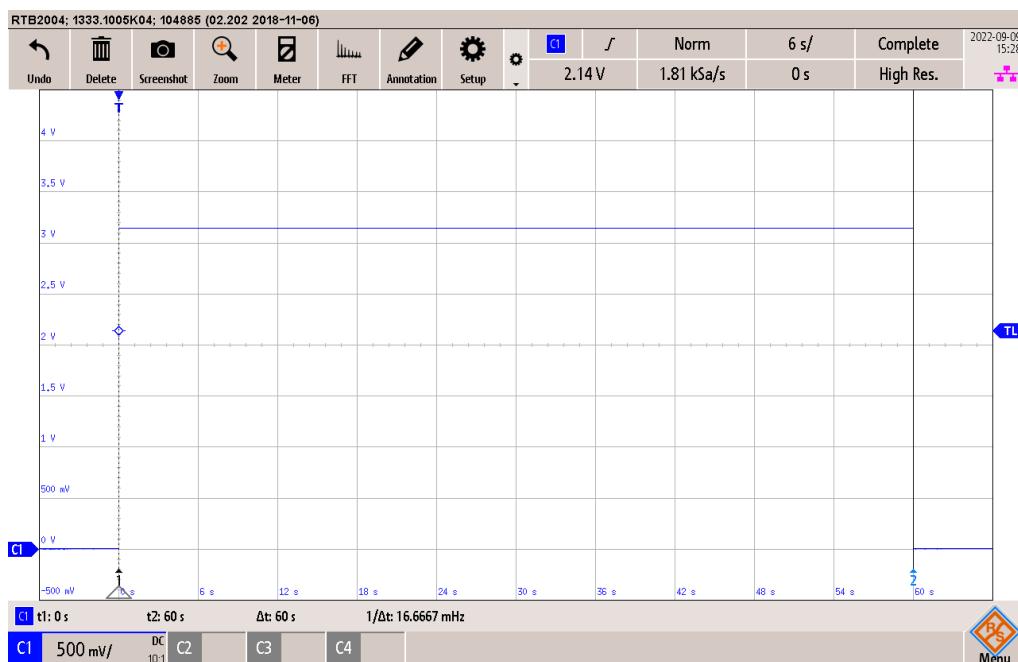


Figure 263 Commande de lecture de la charge de l'accumulateur

| | Tension Init [V] | Tension après 2min [V] | Charge Absolue [mV] |
|------|------------------|------------------------|---------------------|
| USB | 3,57 | 3,63 | 60 |
| Jack | 3,64 | 3,76 | 120 |

Via ces mesures on peut également contrôler que l'on charge à 500mA via l'USB, et que l'on charge à 1A via le Jack. Car en effet on a exactement le double de la charge avec le Jack par rapport à l'USB.

La consommation à l'allumage est de 160mA. Lors ce qu'une LED est allumée avec une note, on consomme environ 247mA. Puis lorsque toutes les LEDs sont allumées en rouge, on a une consommation de 330mA. On a donc environ 70 mA qui passent dans l'haut-parleur.

On peut donc estimer que le Simon pourrait tenir plus de 4h avec une consommation moyenne de 300mA environ.

6. État final et améliorations

6.1. Projet

Dans la globalité le projet respecte le Cahier des charges après sa modification.

La partie avec un possible deuxième mode « Mélodie » a été abandonnée au début du projet, car les technologies utilisées n'étaient pas très claires par le client. De plus suite à une étude brève sur la faisabilité de cette option, une librairie intégrale aurait dû être faite pour la lecture de fichier « .midi ».

De ce fait, la partie hardware a quand même été implémentée, mais rien d'autre touchant de près ou de loin à ce mode n'a été développé.

6.2. Hardware

Toute la partie Hardware demandée dans le cahier des charges a été effectuée. Une modification a été apportée au niveau du condensateur de liaison du haut-parleur.

La partie carte SD n'a pas été mise en service.

Des améliorations pourraient être apportées en fonction de quelle direction on veut aller avec le projet.

Si l'on fait du vrai aléatoire, augmenter le nombre d'échantillons du son, ou encore la capacité de la mémoire interne, on pourrait se diriger vers la famille PIC32MZ.

Si en revanche on veut rester avec le même microcontrôleur, et réduire les coûts de fabrication, on pourrait remplacer le DAC 12bit par un DAC 8bit.

Au niveau des craintes venant des potentielles perturbations dues au haut-parleur, les avis sont plutôt

6.3. Firmware

Le Firmware n'est pas terminé à 100%, et il ne le sera jamais. Énormément d'améliorations peuvent être apportées au Firmware. Que ce soit une méthode différente de faire certaines actions du code, à des fonctionnalités utiles à l'expérience utilisateur.

N'est au moins, la partie du tableau des scores dysfonctionne dans certains cas, qui n'ont pas été tous définis. En effet lorsque que l'on commence à mélanger le nombre de modes, de Players et de nombre parties, des dysfonctionnements ont été notés.

On pourrait par exemple empêcher le joueur de jouer tant que le Simon n'a pas atteint un certain niveau de charge de l'accumulateur.

Bien sûr qu'il faudrait également essayer de mettre en place le mode mélodie, avec le système de carte SD. Mais comme évoqué précédemment, cette partie a été mise de côté au début du projet.

Ou encore d'ajouter d'autres modes de jeu, comme celui où l'on doit reproduire le plus rapidement une séquence, cela ferait appel à la mémoire musculaire, et les tryharders apprécieraient plus ce gameplay.

6.4. Software

Toute la partie Software fonctionne correctement, il n'y a aucun bug apparent.

On pourrait tout fois imaginer des améliorations lors de futures versions du projet. Comme par exemple, la mise en valeur de certaines valeurs du tableau des scores, comme classer les joueurs de celui avec le plus de points, à celui qui en a le moins.

Ou encore, donner les points minimums et maximums de chaque joueur dans leurs courbes de progression personnel.

Dans l'extrême, on pourrait également reconstruire le tableau des scores, mais sous forme de tableau. On y verrait les pseudos de tous les Player, et on pourrait avoir le détail des valeurs de chaque game effectué par chacun.

6.5. Boitier

Le boitier a été terminé à 100%, uniquement en attente de l'impression 3D de la dernière version pour valider les derniers petits changements.

Les améliorations possibles seraient de faire en sorte que l'on puisse clipser les écrous dans leur emplacement, et qu'on aille pas besoin d'y mettre un point de colle.

6.6. Test et Mesures

Malheureusement il est impossible de contrôler toutes les différentes communications que le microcontrôleur et le PC peuvent faire lors de l'envoi du tableau des Scores. Car le tableau des scores peut avoir énormément de configuration différentes.

7. Conclusion

Lors de ce travail de diplôme, j'ai pu mettre en pratique toutes les compétences acquises lors de ma formation de Technicien ES en Génie Électrique, spécialisation électronique. Mais également toute l'expérience acquise lors des autres formation et stage que j'ai pu réaliser.

Cela tant dans la partie de design du schéma, que dans la partie de la réalisation du PCB et de sa fabrication. Mais également pour toute la partie de modélisation 3D, et la partie Software et Firmware. Sans oublier toute la partie de gestion de projet et de documentation du travail.

Lors de la phase de design, énormément de features ont été demandées dans le cahier des charges. Malgré la persistante pénurie de composants et de stock, le design a pu être complété et dimensionné.

Malgré le grand nombre de features implémentées, j'ai voulu faire en sorte que le Simon puisse être tenu en main, c'est pour cela que j'ai fait un PCB de forme ronde. La disposition des composants a été faite par petit regroupement, puis les blocs se sont regroupés entre eux, jusqu'à former le PCB. Une attention au positionnement des connecteurs a également été apportée, afin que l'on puisse y avoir accès aux bords du PCB. Un concept de design de tout l'emboîtement des différents composants et connecteur prenant de la place a été fait, afin d'optimiser le plus possible la réalisation d'un boitier adapté.

Afin de pouvoir braser les QFN plus facilement, et de faciliter la phase de fabrication, un stencil a également été commandé lors de la commande du PCB. Cela m'a permis de braser presque au complet toute la face BOT du PCB en une fois au four. Ça a également été un grand gain de temps lors de la phase de développement.

La modélisation 3D a été faite à partir du modèle 3D du PCB une fois réalisé. Cela a été beaucoup plus simple de modéliser un boitier qui s'emboitait parfaitement avec le PCB conçu. Il n'aurait pas été possible de trouver un boitier dans le commerce avec toutes les caractéristiques de la carte. Il aurait également été très difficile de réaliser tous les trous pour faire passer tous les connecteurs.

La plus grande phase du projet a été celle du Firmware, c'est ce qui donne vie à chaque partie du projet. Une première partie du Firmware a été faite afin de valider le bon fonctionnement du Hardware. Puis c'est toute la partie algorithmie qui a été développée. J'ai pu découvrir lors de ce travail le « Linked List » avec des structures récursives, ce qui n'a pas été vu lors de la formation d'ES. Le Firmware a atteint une certaine complexité, et il aurait été impossible de détailler chaque partie du code. C'est pour cela qu'un système des points les plus importants ont été décis dans le rapport. N'est au moins, l'intégralité du Firmware est commenté si vous souhaitez plus de précisions.

Pour la partie Software j'ai pu conceptualiser un graphique comprenant toutes les courbes représentant les différents points que le Player a pu faire lors de ses différentes games. J'ai également pu programmer l'application en C#, qui est un langage que j'utilise également lors du développement de jeux vidéo.

Lors de la phase de mesure, j'ai pu confirmer le bon fonctionnement de tous les protocoles de communication et les différents timings.

J'ai beaucoup apprécié pouvoir également mettre en pratique mes connaissances de game design. En effet cela fait également plusieurs années que je fais développer de jeu vidéo, et j'ai pu notamment réaliser un GDD pour le Simon.

Lors du développement du Firmware et du Software, un dépôt git a été créé. Cela m'a principalement permis de faire du versioning tout au long du développement, mais également d'avoir accès au code depuis la maison.

Malgré la planification, il y a plusieurs points sur lesquels je n'avais aucun contrôle. J'ai donc et du retard sur la livraison de PCBs et également une commande de composants bloquées. L'impression 3D du boitier a également pris plusieurs semaines, après de multiples problèmes de calibration de l'imprimante 3D. Malgré tous ces contretemps, je ne suis jamais resté bloqué et j'ai pu avancer sur d'autres activités en parallèle.

Au-delà du mode de jeu mélodie supplémentaire, il y a énormément d'explantions du projet possible. On peut imaginer une infinité de modes de jeux, mais également d'autres jeux sur le même hardware. Il y aurait par exemple le jeu du tap taup, où il faut le plus rapidement réappuie sur la touche qui s'allume avant qu'elle ne s'éteigne.

Au total, j'ai travaillé 35 jours non-stop, avec une moyenne entre 10h et 12h par jour. Cela équivaut à un travail à environ 200% par semaine. Si j'avais travaillé 8h par jour, 5 jours par semaine, pour fournir le même état d'avancement que j'ai rendu, cela m'aurait pris plus de 10 semaines.

Je tiens également à remercier toutes les personnes qui ont contribué de près ou de loin à la réalisation de mon travail de diplôme ES.



ETML-ES, le 12 septembre 2022

Ricardo Crespo

8. Références

Datasheet MCP73871

<https://ww1.microchip.com/downloads/en/DeviceDoc/MCP73871-Data-Sheet-20002090E.pdf>

Datasheet ICP543759PMT

<https://www.renata.com/fr-ch/downloads/?product=icp543759pmt&fileid=26760f2abea111a29c91240df5>

Datasheet ULP12OAP1RSFCL1RGB

http://spec_sheets.e-switch.com/specs/B800064.pdf

Datasheet DOGS164-A

<https://www.mouser.ch/datasheet/2/127/dogs164e-1532335.pdf>

Datasheet LMH6672MR

https://www.ti.com/lit/ds/symlink/lmh6672.pdf?HQS=dis-mous-null-mousermode-dsf-pf-null-www&ts=1660424399209&ref_url=https%253A%252F%252Fwww.mouser.ch%252F

Eurocircuits PCB Design Guidelines - EuroTrack Width Graphic

<https://www.eurocircuits.com/pcb-design-guidelines/track-width-graphic/>

Image connexion UART master to slave

<https://www.silabs.com/documents/public/application-notes/an0059.0-uart-flow-control.pdf>

Lecture d'une carte microSD avec du SPI

http://elm-chan.org/docs/mmc/mmc_e.html

Condensateur de liaison

https://sonelec-musique.com/electronique_theorie_condensateur_liaison.html

Gamme des notes

https://fr.wikipedia.org/wiki>Note_de_musique

Famille PIC32MZ

<https://ww1.microchip.com/downloads/aemDocuments/documents/MCU32/ProductDocuments/DataSheets/PIC32MZ-Embedded-Connectivity-with-Floating-Point-Unit-Family-Data-Sheet-DS60001320H.pdf>

ELAN Électronique Linéaire Ch5 : Les amplificateurs opérationnels

K:\ES\Maitres-Eleves\SLO\Modules\SL135_ELAN\Theorie

MINF Programmation des PIC32MX Ch 9 : Gestion du bus I2C

K:\ES\Maitres-Eleves\SLO\Modules\SL229_MINF\Theorie\Cours

9. Lexic

Séquence : Enchainement de notes à reproduire dans l'ordre

Map : Ensemble de séquences s'incrémentant d'une note à chaque nouvelle séquence

Game : Un partie avec l'ensemble des séquences d'une Map, récites ou non par le joueur

Player : Utilisateur lançant une Game

uC : Microcontrôleur

PCB : Printed Circuit Board, carte électronique

10. Annexes

- A. Cahier des charges
- B. Liste des pièces et coûts
- C. Schéma électrique
- D. Listings du Firmware
- E. Listings du Software
- F. Planning du projet
- G. Journal de travail
- H. PV de toutes les séances
- I. Mode d'emploi
- J. Fichier de modification
- K. Mail de retard Euro Circuit
- L. GDD
- M. Toutes les mesures (uniquement format numérique)