

Rapport de laboratoire

École supérieure

Électronique

PROJ
SLO2

Affichage Matriciel

N° 2126

Réalisé par :

Ricardo Crespo

À l'attention de :

Professeur M. Bovey
Professeur M. Castoldi
Professeur M. Moreno

Dates :

Début du laboratoire : 17 novembre 2021
Fin du laboratoire : 16 juin 2022

Table des matières

1.	Pré-Étude	6
1.1.	But du projet et CDC	6
1.2.	Schéma bloc général du système.....	6
1.3.	Schéma bloc hardware.....	6
1.4.	Choix technologiques	7
1.4.1.	LEDs	7
1.4.2.	Multiplexeur.....	7
1.4.3.	MCU.....	7
1.5.	Estimation des coûts	8
1.6.	Planning	8
1.7.	Conclusion et perspectives pré-étude	9
2.	Phase de design.....	10
2.1.	Description du produit	10
2.2.	Principaux choix effectués.....	10
2.3.	Choix effectués et dimensionnement.....	11
2.3.1.	USB	11
2.3.2.	Alimentation	12
2.3.3.	Microcontrôleur	13
2.3.4.	Multiplexeur.....	17
2.3.5.	Matrice à LEDs.....	18
2.3.6.	Liaison intercarte.....	19
2.4.	Concept software	20
2.5.	Concept firmware	21
2.6.	Conclusion phase de design	21
3.	Hardware.....	22
3.1.	MainBoard.....	22
3.1.1.	Spécifications	24
3.1.2.	Fabrication	24
3.2.	Matrix	25
3.2.1.	Spécifications	27
3.2.2.	Fabrication	27
3.3.	Boitier.....	28
3.4.	Modifications	29
3.4.1.	MainBoard.....	29
3.4.2.	Matrix	30
4.	Firmware	32

4.1.	Flowchart	32
4.2.	Initialisations et cycles d'interruption	33
4.2.1.	Configurations et initialisations	33
4.2.2.	Interruption et Callback	36
4.3.	Détection automatique du nombre de Matrix connectées	38
4.4.	Initialisation des MAX7221	41
4.4.1.	Mécanisme de registre à décalage.....	41
4.4.2.	Configuration des registres.....	43
4.4.3.	Intensity.....	45
4.4.4.	ShutDown	45
4.4.5.	Affichage	46
4.5.	Communication avec le Software	47
4.6.	Traitement du nom	49
4.6.1.	Contrôle du nom reçu.....	49
4.6.2.	Mise en forme du nom.....	49
4.6.3.	Affichage du nom sur les Matrix	53
4.6.4.	Animation de défilement.....	54
5.	Software	56
5.1.	Flowchart	56
5.2.	Communication avec le Firmware	57
5.2.1.	Initialisations	57
5.2.2.	Événement Timer1	59
5.2.3.	Événement réception de données du Firmware	62
6.	Test et mesures.....	64
6.1.	Fréquence d'interruption du Timer1.....	64
6.1.1.	Matériel de mesure	64
6.1.2.	Méthode de mesure	64
6.1.3.	Schéma de mesure	64
6.1.4.	Analyse de mesure	65
6.2.	Communication UART.....	66
6.2.1.	Méthode de mesure	66
6.2.2.	Schéma de mesure	66
6.2.3.	Analyse des mesures	67
6.3.	Communication SPI.....	69
6.3.1.	Méthode de mesure	69
6.3.2.	Schéma de mesure	69
6.3.3.	Analyse des mesures	70

6.4.	Consommation en courant	75
6.4.1.	Méthode de mesure	75
6.4.2.	Schéma de mesure	75
6.4.3.	Analyse des mesures	76
7.	État final et améliorations	77
7.1.	Projet.....	77
7.2.	Hardware	77
7.3.	Firmware	77
7.4.	Software.....	77
7.5.	Boitier.....	77
7.6.	Test et Mesures	77
8.	Conclusion	78
9.	Références.....	79
10.	Annexes.....	80
A.	Cahier des charges.....	80
B.	Schéma électrique de la MainBoard	80
C.	Schéma électrique de la Matrix	80
D.	Liste des pièces et coûts	80
E.	Listings du Firmware.....	80
F.	Listings du Software	80
G.	Planning du projet	80
H.	Journal de travail.....	80
I.	Mode d'emploi	80
J.	Résumé	80
K.	Affiche du projet.....	80

Ce rapport a des parties écrites à la troisième personne du singulier, cela comporte l'auteur mais également dans certain cas les code, et dans d'autres le lecteur.

Pour toutes les captures de code, le numéro des lignes ont également été affichées, cela vous permet de vous repérer dans les listings fournis en annexes.

Il est très vivement recommandé de lire ce rapport au format couleur, si ce n'est pas votre cas la version digitale est la plus adaptée, afin de préserver notre planète.

1. Pré-Étude

1.1. But du projet et CDC

Lors de la rentrée des premières années à l'ETML-ES, ils doivent écrire leur nom sur une feuille de papier pour que les enseignants puissent retenir leurs noms. Afin de simplifier cette étape et de la normaliser, un affichage matriciel à installer derrière les écrans des PCs, permettra d'afficher le nom lié au profil utilisateur de l'étudiant logué.

Vous trouverez le cahier des charges en annexes, pour plus de détails.

1.2. Schéma bloc général du système

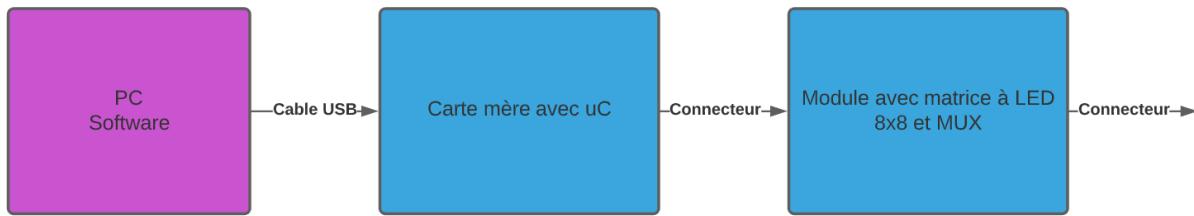


Figure 1 Schéma bloc général du système

1.3. Schéma bloc hardware

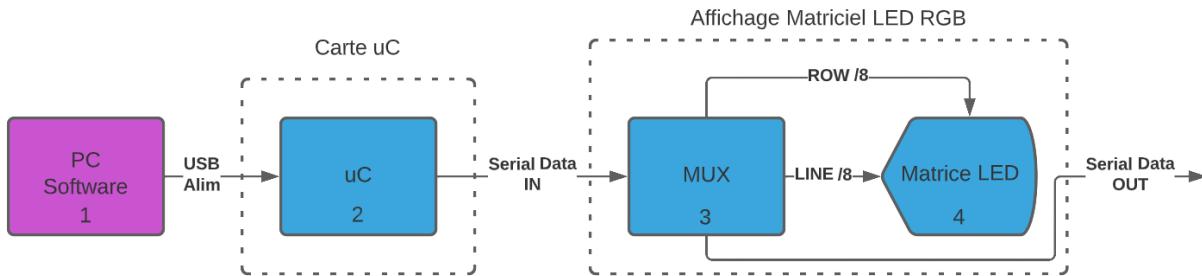


Figure 2 Schéma bloc du hardware du projet

On a principalement trois parties, le PC avec le software, une carte mère avec le microcontrôleur, et la partie des cartes avec les matrices avec les LEDs.

Dans le bloc n°1, on retrouve le PC sur lequel on va connecter à la carte avec le microcontrôleur. On va également réaliser un programme en C#, qui va récupérer le nom de l'élève logué sur la machine quand il se connectera.

Une fois connecté, dans le bloc n°2 le microcontrôleur récupère le nom à afficher, et le convertit en caractère pour le bloc suivant. On stockera également le nom de l'élève dans l'EEPROM, pour qu'à l'allumage de l'affichage le dernier nom stocké soit affiché.

Dans le bloc n°3, on retrouve des multiplexeurs 8bit d'adresse et 8bit à 24bit de data qui reçoivent les informations du microcontrôleur, et qui permettront de commander toutes les colonnes et lignes de LEDs.

Finalement dans le bloc n°4, on y trouve les 8 matrices 8x8 LEDs, 6cm de large et de long. C'est donc avec 8 cartes chainées comportant une matrice et un multiplexeur, que l'on va pouvoir créer un affichage qui fera environ 48cm de long et 6cm de haut.

Cet affichage nous permettra d'afficher 10.8 caractères 5x7 avec des espaces d'une LED compris.

L'affichage sera fixé à l'arrière des écrans à l'aide de crochets, comme sur une webcam.

1.4. Choix technologiques

1.4.1. LEDs

Le choix d'utiliser une matrice a très vite été écarté, car aucun modèle ne respecte nos contraintes de consommation.

Pour le choix des LEDs, je pars sur des LEDs basse consommation pour respecter les limites de consommation du port USB de 500mA. Je vais positionner les LEDs de manière à créer des lignes et des colonnes. De cette manière que vais pouvoir afficher toute la longueur de la ligne sur la totalité des matrices au même temps. J'aurais donc un maximum de 8 LEDs par ligne fois 8 matrices.

Il y a deux possibilités, soit une LED avec une seule couleur, soit une LED RGB.

Donc 64 LEDs avec une seule couleur allumée avec une consommation de 2mA, je serais à 128mA max juste pour la consommation des matrices à LEDs.

Donc 64 LEDs allumées avec une consommation de 5mA, je serais à 320mA max juste pour la consommation des matrices à LEDs. Donc dans ce cas l'utilisation d'une des trois couleurs respecte les limitations de consommation. Mais si on veut allumer les trois couleurs au max, on aura une consommation de 15mA par LED, donc une consommation totale de 960mA. Il faudra donc avoir une luminosité plus basse avec le mode RGB, ou alors allumer qu'une couleur à la fois.

1.4.2. Multiplexeur

Pour pouvoir commander les lignes et les colonnes des LEDs uni couleurs, je pourrais utiliser un multiplexeur avec 8 lignes d'adresse, et 8 lignes de data.

Pour pouvoir commander les lignes et les colonnes des LEDs RGB, je vais devoir utiliser trois mêmes composants précédemment utilisés pour les LEDs uni couleurs. Ainsi j'aurais 24 lignes de data, car trois couleurs par LED, et les 8 lignes de datas des trois multiplexeurs auront les mêmes valeurs, car ils seront connectés ensemble sur les matrices de LEDs RGB.

Pour pouvoir communiquer les informations à travers la matrice, une communication série est utilisée, je pourrais notamment utiliser le module SPI du microcontrôleur pour effectuer cette tâche. Je vais pouvoir chainer les différents multiplexeurs, et commander toutes les matrices depuis la carte mère avec le microcontrôleur.

Grâce à cette communication série, il nous suffira que d'environ 4 fils à connecter entre les modules à matrices, car DIN, CS, VCC et GND.

1.4.3. MCU

Je vais utiliser un micromoteur de la famille PIC32MK, car ils comportent une EEPROM intégrée. Leur utilisation m'évitera de rajouter une EEPROM externe. Pour le reste des caractéristiques, je ne suis pas très limité, je prendrais un modèle préférentiel de l'ETML-ES, notamment le PIC32MK0512MFC064.

1.5. Estimation des coûts

Ici j'ai calculé l'estimation des coûts avec les principaux composants, il faudra donc tenir en compte que ce ne sont pas des valeurs définitives, car tout n'est pas encore défini.

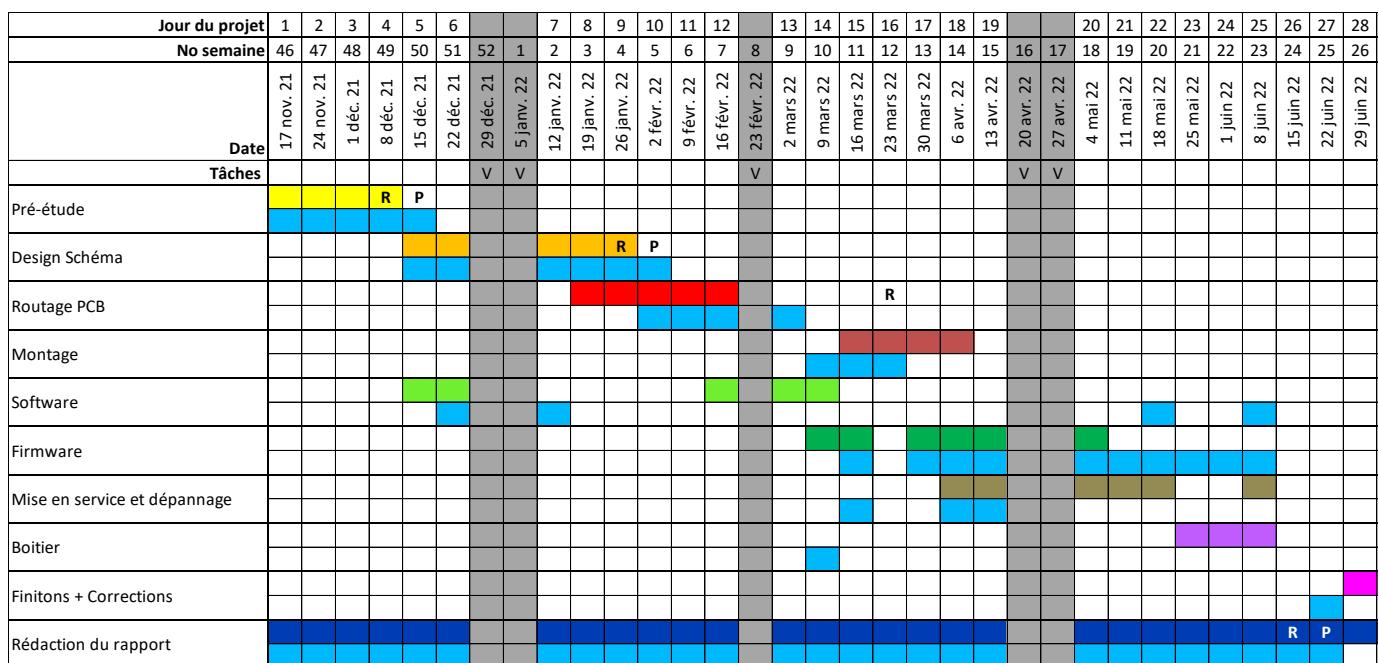
Les estimations ont été faites de sorte à pouvoir équiper 18 places avec les affichages.

Vous retrouverez le montant total et unitaire pour les deux options, couleur UNIE ou RGB.

	Quantité	Description	Supplier	Price unit	CHF	Price	CHF
UNIE	144	MAX7221 Serially Interfaced, 8-Digit LED Display Drivers	Aliexpress	0,67	CHF	96,48	CHF
	10000	LED RED DIFFUSED 5MM T/H	DigiKey	0,10323	CHF	1032,3	CHF
	18	PIC32MK0512MCF064T-I/PT	DigiKey	7,79	CHF	140,22	CHF
	144	PCB Matrix	Eurocircuit	3	CHF	432	CHF
	18	PCB MainBoard uC	Eurocircuit	4	CHF	72	CHF
					Total	1773	CHF
				Unitaire		98,5	CHF

	Quantité	Description	Supplier	Price unit	CHF	Price	CHF
RGB	432	MAX7221 Serially Interfaced, 8-Digit LED Display Drivers	Aliexpress	0,67	CHF	289,44	CHF
	10000	LED standard - CMS RGB LED OSIRE E3635	Mouser	0,172	CHF	1720	CHF
	18	PIC32MK0512MCF064T-I/PT	DigiKey	7,79	CHF	140,22	CHF
	144	PCB Matrix	Eurocircuit	3	CHF	432	CHF
	18	PCB MainBoard uC	Eurocircuit	4	CHF	72	CHF
					Total	2653,66	CHF
				Unitaire		147,43	CHF

1.6. Planning



1.7. Conclusion et perspectives pré-étude

Grâce à ce projet, l'étape de l'atelier « bricolage » à la rentrée pour fabriquer les pancartes avec les noms des étudiants n'aura plus lieu. De plus, dû à la normalisation du format de l'affichage pour tous les élèves, les enseignants n'auront plus aucune raison de ne pas savoir les noms des élèves.

De plus, cela fera un projet de plus à montrer pendant le portes ouvertes, qui est d'ailleurs très visuel, et qui attirera du nouveau monde dans notre formation d'électronicien.

Dû aux contraintes de consommation, des choix ont déjà été faits, notamment l'obligation d'utiliser des LEDs « low current ».

Je pense que la nouvelle option de faire des modules séparés avec des matrices de 8x8 chainables, permettra de réutiliser ces modules pour d'autres projets futurs.

De plus on pourra les chaînes dans tous les sens, c'est-à-dire que l'on pourra faire des lignes et des colonnes avec de multiples modules, et en faire des panneaux de plusieurs.

Suite à cette pré-étude, le client va devoir choisir quel voit il veut suivre pour la suite du projet, l'option UNIE couleur, ou l'option RGB.

Au-delà du fonction normale souhaité par le client, de récupérer uniquement les noms des élèves logués sur leurs sessions, on pourrait imaginer ajouter des options supplémentaires sur le software que je vais devoir développer. On pourrait imaginer par exemple de pouvoir y inscrire le texte que l'on veut, ou encore changer les animations ou les couleurs.

Une fois le choix fait, je pourrais commencer le développement concret du projet, et cela jusqu'à son aboutissement.

2. Phase de design

2.1. Description du produit

Lors de la rentrée des premières années à l'ETML-ES, ils doivent écrire leur nom sur une feuille de papier pour que les enseignants puissent retenir leurs noms. Afin de simplifier cette étape et de la normaliser, un affichage matriciel à installer derrière les écrans des PCs, permettra d'afficher le nom lié au profil utilisateur de l'étudiant logué.

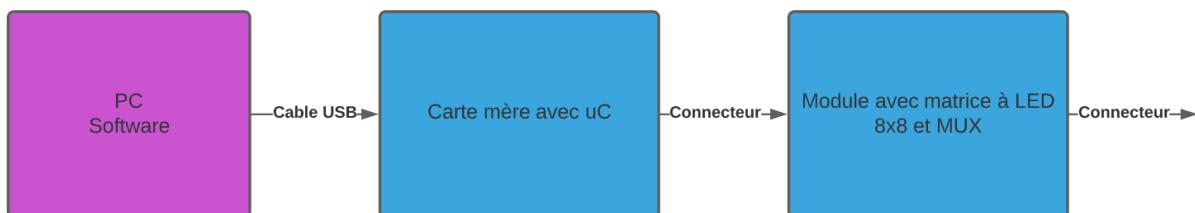


Figure 4 Schéma bloc du system complet

Vous trouverez plus de détails sur le schéma bloc dans la pré-étude et dans le CDC.

Globalement on est sur deux cartes, une avec le microcontrôleur qui récupère les informations qui viennent via l'USB du PC. Puis les deuxièmes types de cartes sont les modules avec les matrices à LEDs.

2.2. Principaux choix effectués

Suite aux retours de la pré-étude, je suis passé sur les familles MX, car on peut également dédier une partie de la mémoire comme EEPROM.

J'aurais besoin d'une communication UART, et d'une communication SPI. Je n'aurais donc pas besoin de beaucoup de ports pour faire fonctionner le tour. C'est pourquoi j'ai pris un microcontrôleur adapté à mon usage. Suite à la vérification des disponibilités en stock, j'ai opté pour le microcontrôleur « PIC32MX130F064B » de 28 pins.

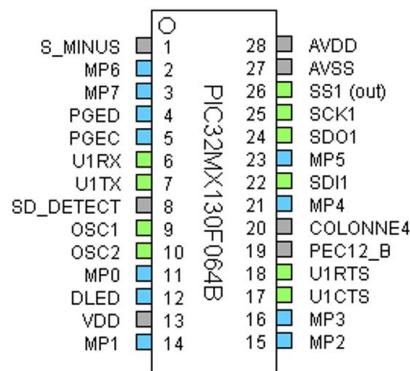


Figure 5 Microcontrôleur PIC32MX130F064B

Le reste des choix pour les autres composants seront détaillés dans leurs sections dédiées.

Prenez en compte les valeurs de tension des condensateurs, comme une valeur minimum.

2.3. Choix effectués et dimensionnement

2.3.1. USB

Pour permettre de faire la communication entre le pc et ma carte, je vais utiliser une communication USB. C'est également via ce lien que je vais pouvoir alimenter tout le dispositif, et je n'aurais donc pas à mettre une alimentation externe, et donc respecter le cahier des charges.

Le microcontrôleur pic32 utilisé ne possède pas de module pour la communication USB. C'est pourquoi pour les projets le choix de prendre un convertisseur USB to UART a été fait.

Pour cela nous avons un modèle de prédilection à l'ES, le « FT230XS », mais malheureusement il n'est plus disponible dans le monde actuellement. C'est pourquoi j'ai cherché une alternative, et prenant le circuit « CY7C64225 ».

Je me suis basé sur le schéma proposé par le fabricant, dans la configuration avec le microcontrôleur alimenté avec un régulateur externe. Dans mon cas il sera alimenté en 3.3V, comme ci-dessous, et donc les sorties UART seront également en 3.3V

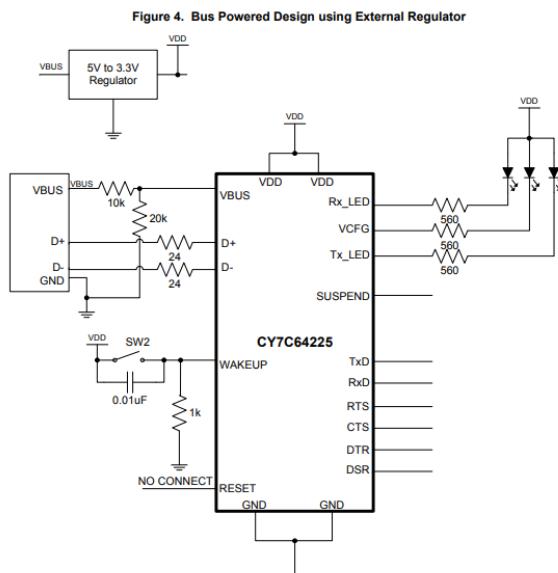


Figure 4. Bus Powered Design using External Regulator

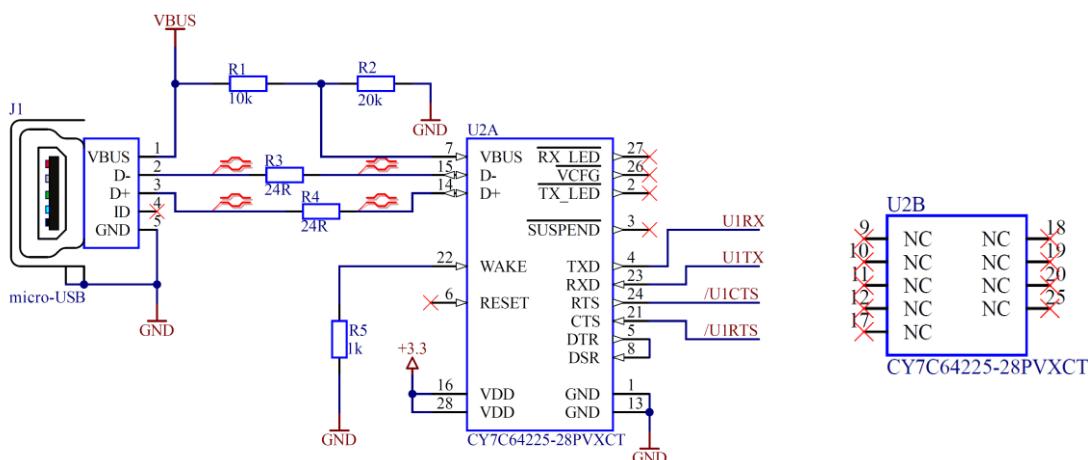


Figure 6 Schéma proposé par le fabricant pour le composant « CY7C64225 »

2.3.2. Alimentation

2.3.2.1. 5V

La liaison établie entre le pc et la carte principale étant faire par une communication USB, je dispose donc d'une alimentation en entrée de 5V et 500mA.

Pour filtrer les éventuelles perturbations IFTB, j'ai protégé le circuit avec un filtre en T avec les ferrites et un condensateur. 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.

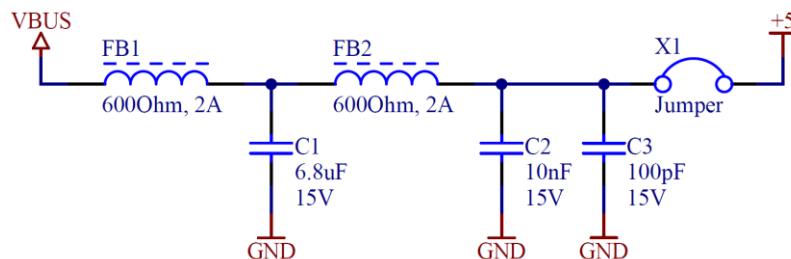


Figure 8 Filtrage de l'alimentation USB VBUS qui fournit le 5V propres à la carte

Après le filtre en T, vous remarquerez la présence d'autres deux condensateurs, ils sont là pour découpler l'alimentation sur une grande plage de fréquences.

Un jumeau a également été placé pour pouvoir alimenter la carte avec une alimentation externe lors de la mise en service, et du développement du firmware.

2.3.2.2. 3.3V

Pour pouvoir alimenter le microcontrôleur j'ai besoin d'une tension d'alimentation de 3.3V. J'ai donc pris, en consensus avec tous ces qui devaient alimenter un microcontrôleur le même composant. C'est donc le LDO « MAX1793 » qui a été choisi. Il est simple à implémenter et nous avons du stock à l'ES.

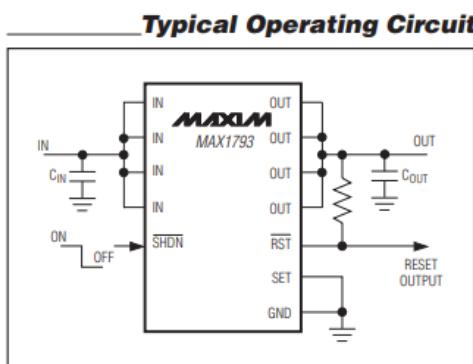


Figure 10 Recommandations du fabricant du LDO

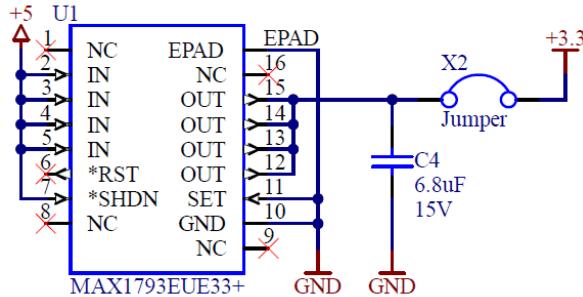


Figure 9 Schéma effectué pour le LDO

Vous pourrez remarquer que je n'ai pas mis le condensateur CIN, car j'arrive avec les 5V du USB, et on ne peut pas charger cette ligne avec plus de 10uF. En effet lors du filtrage de l'alimentation 5V j'utilise déjà un condensateur de 6.8uF

2.3.3. Microcontrôleur

2.3.3.1. Communication

2.3.3.1.1. UART

Comme précédemment dit, c'est donc une communication UART qu'il faudra décoder avec mon microcontrôleur les informations envoyées par le PC via USB. On aura notamment le nom de l'étudiant qu'il faudra afficher.

J'ai donc choisi d'utiliser l'UART numéro 1 du microcontrôleur, qui se trouve sur les pins suivantes :

U1CTS	PPS	PPS	PPS	PPS	I	ST	UART1 clear to send
U1RTS	PPS	PPS	PPS	PPS	O	—	UART1 ready to send
U1RX	PPS	PPS	PPS	PPS	I	ST	UART1 receive
U1TX	PPS	PPS	PPS	PPS	O	—	UART1 transmit

Figure 11 Tableau pour les connexions de l'UART numéro 1 du microcontrôleur

Comme vous pouvez le voir, on a la mention « PPS », qui veut dire « Peripheral Pin Select ». Cela signifie que l'on peut choisir sur que pin on veut mettre nos signaux.

J'ai donc pu choisir les pins suivantes pour l'UART1 sur MPLAB pour fixer leur position.

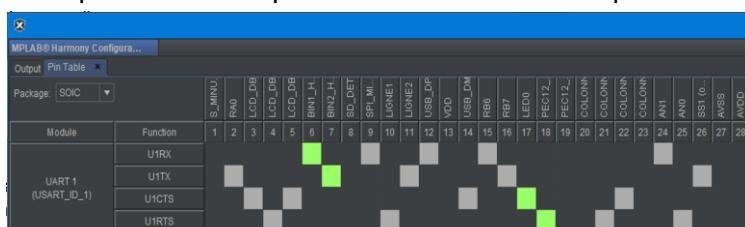


Figure 12 Configuration des pins de l'UART1 dans MPLAB

Puis pour la connexion au microcontrôleur, il ne faut pas oublier de croiser les signaux, comme ci-dessous.

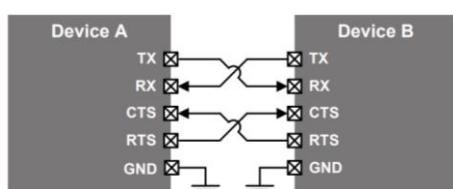


Figure 13 Connexions de l'UART Master et Slave

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, même s'il ne sera pas fondamental d'envoyer des données du microcontrôleur au PC.

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

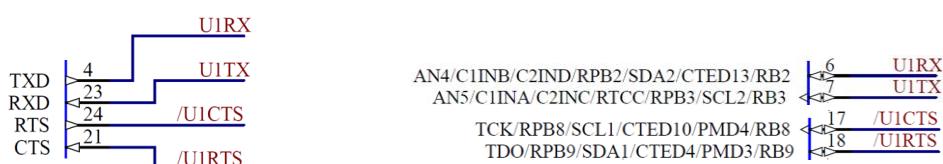


Figure 14 Pins utilisées pour la communication UART

2.3.3.1.2. SPI

Pour communiquer entre la carte principale et les modules de matrices à LED, j'utiliserais le SPI. Cela est dû au multiplexeur utilisé sur chaque module, qu'on verra plus loin dans le rapport.

J'ai donc choisi de prendre le premier SPI disponible, dans ce cas j'ai pris le SPI numéro 1, qui se trouve sur les pins suivantes :

SCK1	22	25	28	14	I/O	ST	Synchronous serial clock input/output for SPI1
SDI1	PPS	PPS	PPS	PPS	I	ST	SPI1 data in
SDO1	PPS	PPS	PPS	PPS	O	—	SPI1 data out
SS1	PPS	PPS	PPS	PPS	I/O	ST	SPI1 slave synchronization or frame pulse I/O

Figure 15 Tableau pour les connexions du SPI numéro 1 du microcontrôleur

J'ai donc pu choisir les pins suivantes pour le SPI1 sur MPLAB pour fixer leur position.

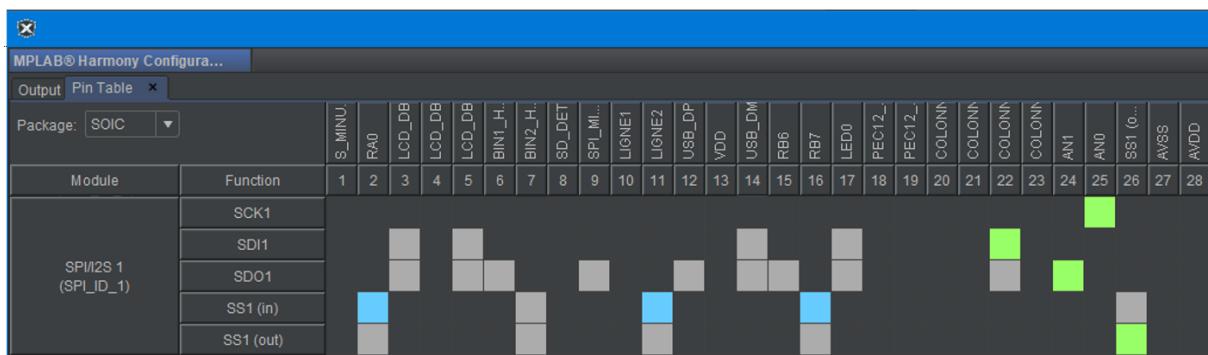


Figure 16 Configuration des pins du SPI1 dans MPLAB

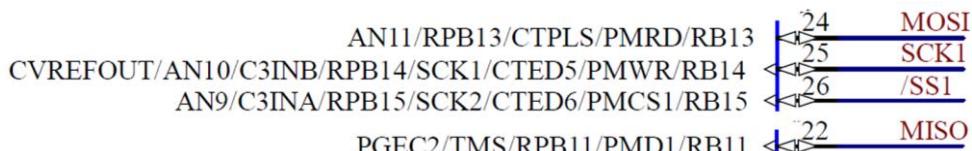


Figure 17 Pins du microcontrôleur utilisées pour la communication SPI1

2.3.3.2. Quartz

Pour le choix de la fréquence du quartz, je me suis basé sur le kit micro que l'on utilise en classe, et j'ai repris la même fréquence de 8MHz. Cette fréquence sera modifiée grâce à la PLL du micro pour monter jusqu'à 80MHz, pour travailler avec les mêmes fréquences que pour les autres travaux faits au labo. J'ai opté pour l'utilisation d'un quartz car j'utilise plusieurs protocoles de communication, et car l'affichage très rapide et successif de tous les modules connectés à ma carte principale doit se faire sans accros.

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

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

Figure 19 Dimensionnement selon le site de Microchip, voir références

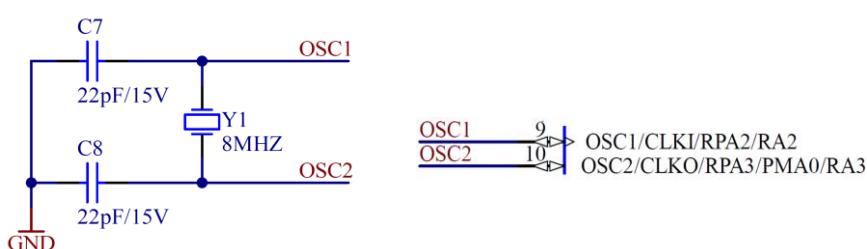


Figure 18 Pins utilisées pour le quart du microcontrôleur

2.3.3.3. Reset

L'implémentation d'un button poussoir sur le reste est là pour faciliter le développement, et de réinitialiser le tout s'il y a un changement d'hardware. Notamment le nombre de modules de matrices connectées. En plus d'un reset manuel, lors de la mise sous tension du circuit un reset est automatiquement effectué également.

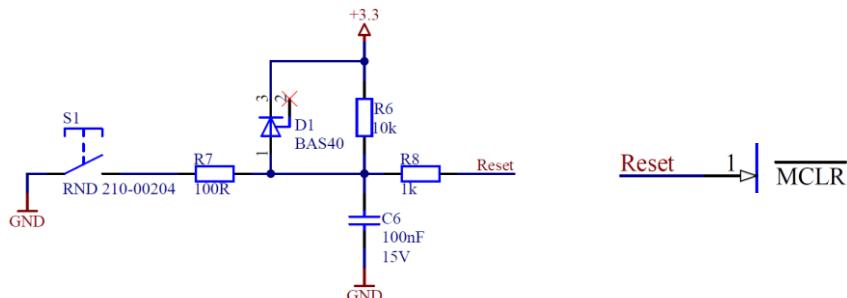


Figure 20 Schéma du montage de reset et la pin de reset

2.3.3.4. JTAG

Le JTAG est là pour pouvoir programmer le microcontrôleur en chargent 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. Est a été designé pour accueillir un debugger « ICD3 ».

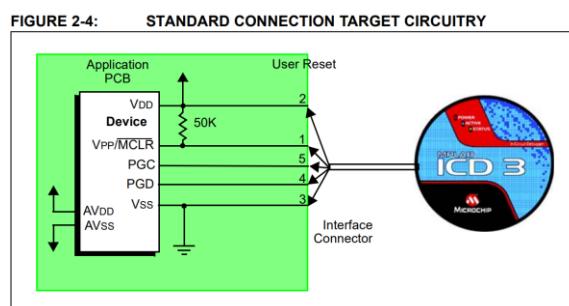


Figure 21 Connexions du « ICD3 »

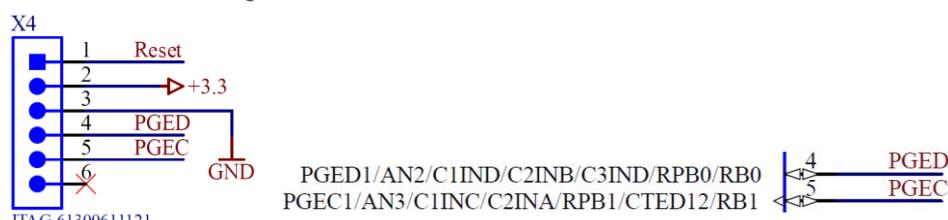


Figure 22 Connexions du port JTAG avec le microcontrôleur

2.3.3.5. Monitoring

Pour faciliter le développement du firmware, et le test de fonctionnalités, j'ai sorti toutes les pinnes restantes sur un connecteur pour y avoir un accès direct. Cela va me permettre d'y mettre le signal que je veux pour tester certaines fonctionnalités. J'y ai également mis le VCC et le GND.

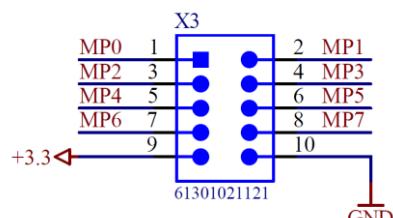


Figure 23 Port pour le monitoring

2.3.3.6. LED de vie

Pour s'assurer que les deux alimentations fonctionnent correctement et que le circuit est allumé, j'ai ajouté deux LEDs de vie.

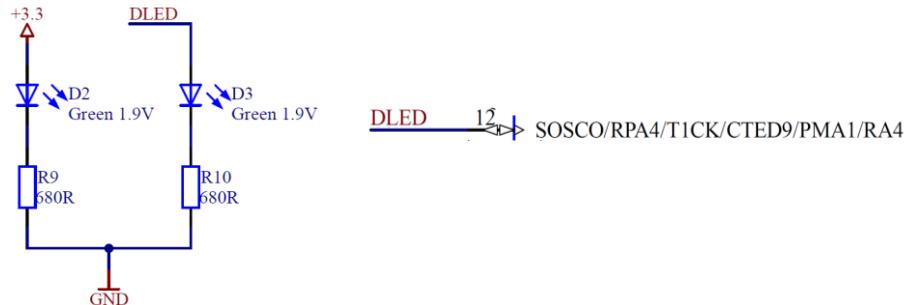


Figure 24 Connexion des LEDs de vie

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

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

2.3.3.7. Découplage

Comme indiqué et conseillé sur le datasheet du fabricant du microcontrôleur, j'ai ajouté un certain nombre de condensateurs de découplage entre le VCC et le GND des différentes pates du microcontrôleur.

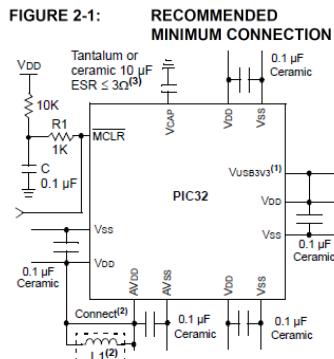


Figure 25 Recommandations de découplage du fabricant du microcontrôleur

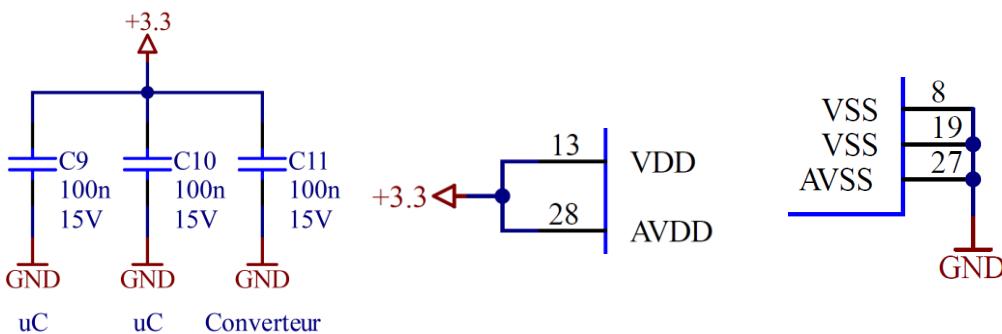


Figure 26 Découplage du microcontrôleur et du convertisseur USB to UART

En plus des deux condensateurs de découplage pour les alimentations « VDD » et « AVDD » du microcontrôleur, j'ai un troisième condensateur qui découple le convertisseur USB to UART.

2.3.4. Multiplexeur

2.3.4.1.1. Data et Adresse

Pour réduire le nombre de connexions à transmettre entre chaque module de matrice, un multiplexeur sera utilisé. Il me permettra également de directement driver les LD en courant avec une seule résistance dimensionnée par la suite.

C'est ce composant qui va décoder la trame SPI envoyée depuis le microcontrôleur.

J'ai donc 8 pins qui seront connectées sur les anodes des LEDs de la matrice, qui se trouveront connectées ensemble sur les colonnes. J'ai également 8 pins qui seront connectées sur les cathodes des LEDs de la matrice, qui se trouveront connectées ensemble sur les lignes.

Les colonnes « ROW » seront les pins « SEG » du multiplexeur, et les lignes « LINES » seront les pins « DIG », comme le propose un exemple dans le datasheet du fabricant.

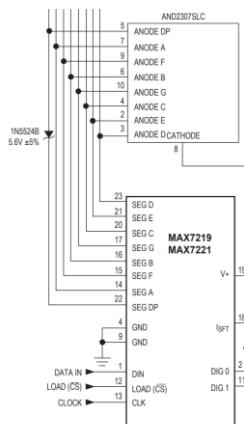


Figure 27 Recommandations du fabricant du convertisseur

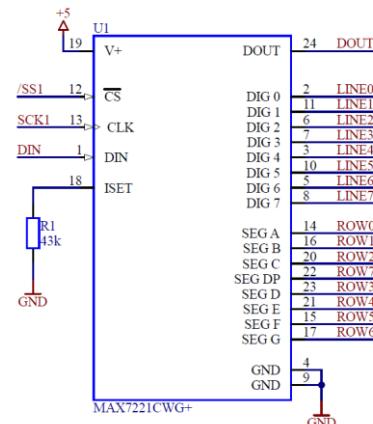


Figure 28 Connexion du convertisseur

2.3.4.1.2. Résistance ISET

Pour dimensionner la résistance qui va nous permettre de fixer le courant pour driver les LEDs, je n'ai pas eu accès à un calcul me permettant de calculer la valeur exacte. C'est pourquoi je me suis basé sur le seul tableau avec des valeurs que le fabricant propose. J'ai donc tracé une courbe de la tendance de ces valeurs, et j'y est estimé la valeur pour une consommation de 16mA. Car j'afficherais au maximum une ligne de 8 LEDs à la fois, consommant chacune 2mA.

Table 11. R_{SET} vs. Segment Current and LED Forward Voltage

I _{SEG} (mA)	V _{LED} (V)				
	1.5	2.0	2.5	3.0	3.5
40	12.2	11.8	11.0	10.6	9.69
30	17.8	17.1	15.8	15.0	14.0
20	29.8	28.0	25.9	24.5	22.6
10	66.7	63.7	59.3	55.4	51.2

Note: R_{SET} values are in Kilo Ohms (kΩ)

Figure 30 Table pour dimensionner R_{SET}

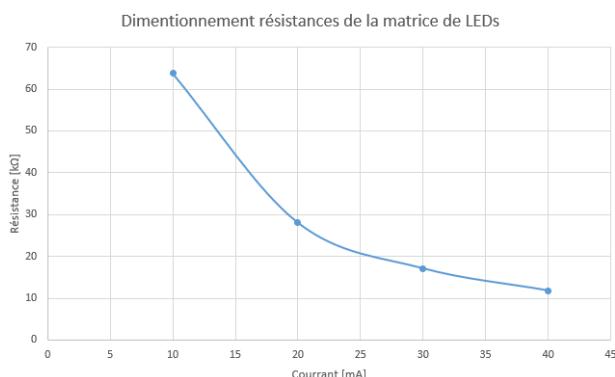


Figure 29 Courbe extraite du tableau pour dimensionner R_{SET}

En prenant du principe que la LED soit à environ 2V, et une consommation de 16mA, je tombe donc sur une valeur de 43kΩ pour la résistance.

2.3.5. Matrice à LEDs

Afin de respecter les contraintes de consommation, j'ai opté pour la réalisation des matrices à LEDs par moi-même, avec des LEDs séparées, lors de la pré-étude.

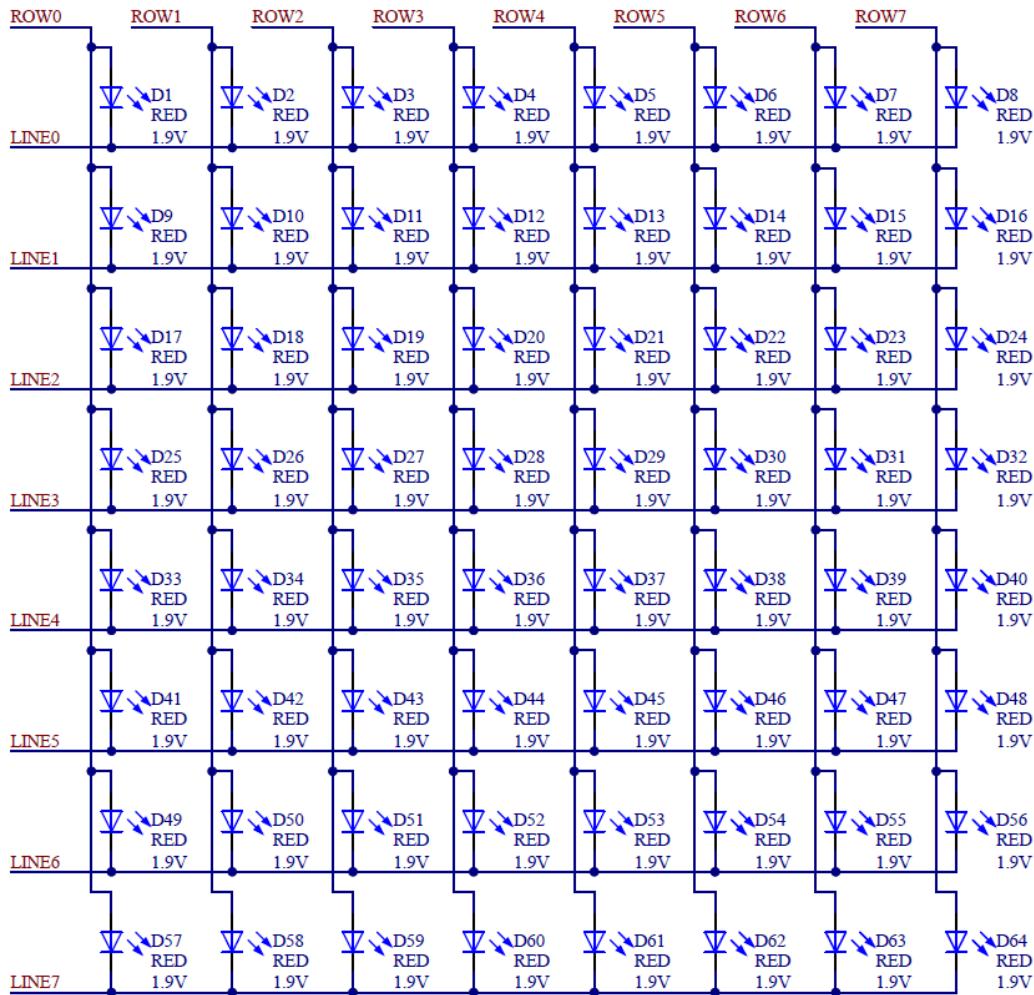


Figure 31 Connexions de la matrice de LEDs

C'est donc avec un système de ligne et de colonnes que je vais pouvoir commander la LED que je veux indépendamment des autres.

Pour cela je commanderais ligne par ligne de 8 LEDs. Par exemple si je veux allumer la deuxième LED de la première ligne, je mettrais la ligne « LINE0 » à 0, et la colonne « ROW1 » à 1. Cela fonctionne de la même manière pour toutes les autres combinaisons.

Une fois la première ligne affichée, je passerais à la deuxième ligne, et ainsi de suite jusqu'à afficher les 8 lignes.

Je ferais ensuite tourner en boucle l'affichage des 8 lignes en permanence. Une ligne représentera toutes les LEDs connectées sur cette même ligne. C'est-à-dire que si j'ai trois modules de matrices chainées, j'afficherais à chaque fois sur une ligne les 24 LEDs au même temps. Je synchroniserais le tout grâce au chip sélec, qui va me permettre de d'abord charger les informations sur tous les modules, et de déclencher l'affichage au même temps pour tous les modules.

2.3.6. Liaison intercarte

Afin de connecter tous les modules à la carte principale, et de pouvoir chainer les matrices, j'ai dû créer une disposition des signaux minimums à avoir. J'ai donc sorti le « MOSI » et le « MISO » du SPI, le clock du SPI et son chip sélec. En plus des signaux de communication, j'envoie également l'alimentation, dont le 5V et le GND, comme ci-dessous.

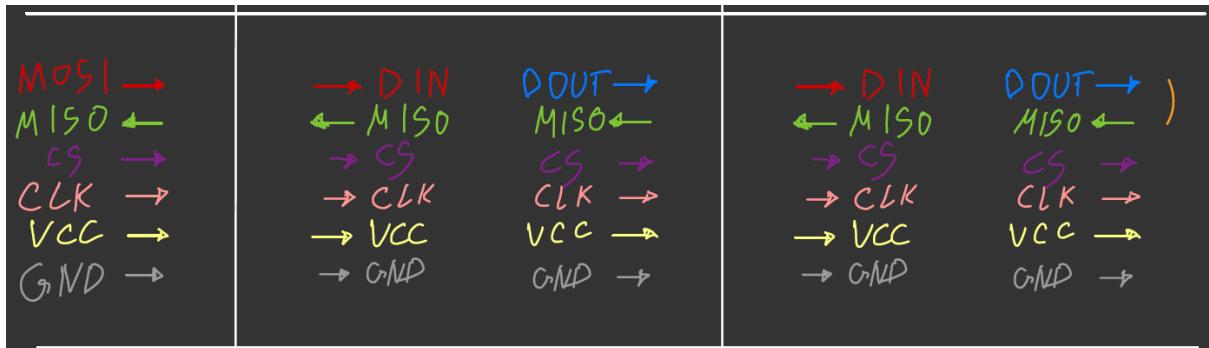


Figure 32 Pinning des connecteurs intercarte, carte principale à gauche, suivie de deux modules à matrices

Ce qui se traduit par ça sur la carte principale, et les deux connecteurs sur le module avec la matrice.

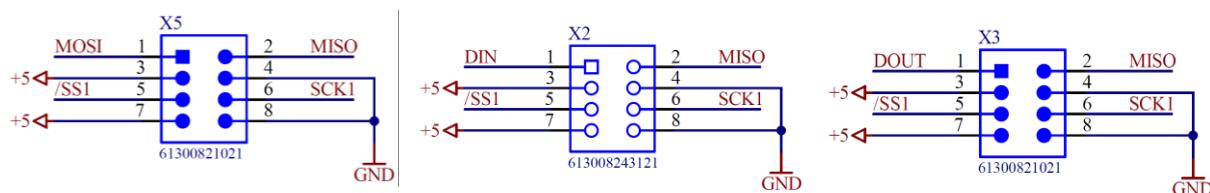


Figure 33 Connecteur de la carte principale, suivie des deux connecteurs des modules

L'utilisation du signal « MISO » pourrait être mise en doute quant à son utilité. C'est grâce à ce signal que je vais pouvoir déterminer le nombre de modules de matrices qui seront chainées à l'allumage ou au reset du dispositif.

C'est en plaçant un jumper entre le « DOUT » et le « MISO » du dernier module de matrices qu'une boucle sera créée. Ensuite ça sera grâce au firmware que je pourrais faire un test successif en envoyant des données sur les registres à décalage du driver de LEDs, et relire les informations que je reçois en retour.

Pour un exemple où je connecte deux modules, je devrais envoyer deux fois les packs utiles à leur utilisation, une fois pour les remplir, et une fois pour les revider et lire les informations reçues. Dans le cas où j'envoie que les informations nécessaires pour un seul module, je ne recevrais pas en retour toutes les informations envoyées.

Le choix d'utiliser un connecteur avec deux rangées des quatre pinnes a été fait, pour que le tout soit compatible avec les connecteurs 8 pins pour les câbles plats. Cela facilite très grandement la portabilité et l'adaptation du system.

2.4. Concept software

Afin de récupérer le nom de la cession de l'élève connecté sur sa session, un logiciel à installer sur le PC sera nécessaire. Une fois le nom récupéré, il faudra qu'il l'envoie sur le port USB qui sera connecté à la carte principale.

Afin de faciliter la démarche, j'ai sou traité la réalisation d'un programme en C# fait par un informaticien de quatrième année de CFC. Je lui ai demandé de récupérer le nom de la session et de l'afficher dans une fenêtre sur le bureau. De mon côté je devrais envoyer l'information récupérée, et l'envoyer via USB vers la carte principale de mon système.

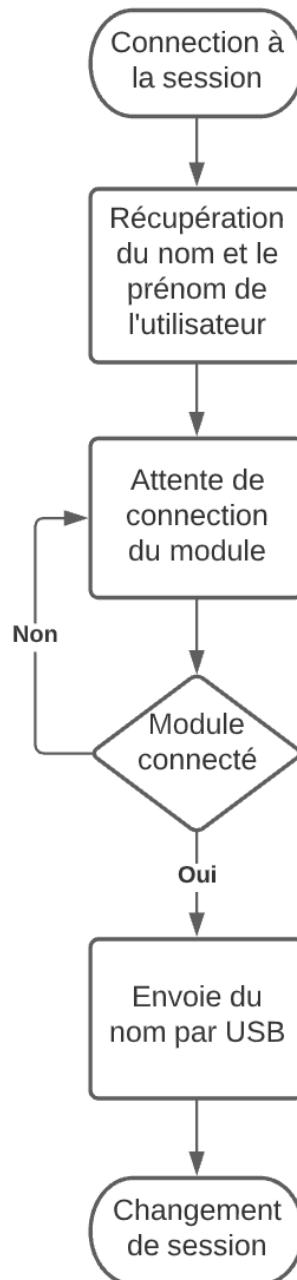


Figure 34 Flowchart du Software

2.5. Concept firmware

Pour la partie firmware, je devrais à l'allumage du système récupérer le nombre de matrices qui sont connectées à la carte principale.

Puis une fois les configurations effectuées, je devrais récupérer le nom de l'élève logué sur sa session.

Il ne me restera plus qu'à transmettre aux modules de matrices les LEDs qu'il faut allumer et éteindre pour afficher le nom correctement.

Si le nom ne rentre pas en entier sur le nombre de matrices connectées, un défilement sera effectué.

Après le premier affichage d'un nom, celui-ci est stocké dans l'EEPROM. Donc si le système est mis hors tension, puis remis sous tension, il affichera le dernier nom sauvegardé.

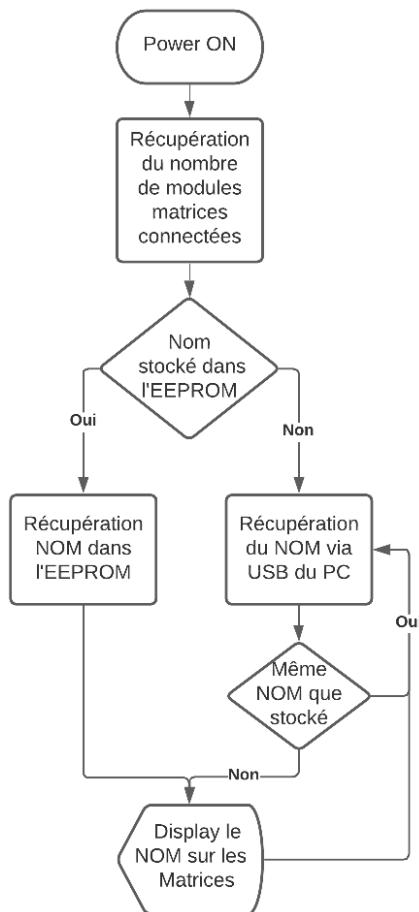


Figure 35 Flowchart du Firmware

2.6. Conclusion phase de design

Suite à cette phase de design, j'ai pu établir et définir tous les composants, J'ai également pu réaliser l'intégralité des deux schémas électriques des deux cartes.

J'ai également pu corriger les différentes erreurs faites lors de la phase de pré-étude. Mais également put me confronter à des problématiques, et devoir trancher sur plusieurs choix décisifs pour le projet.

La prochaine étape sera de réaliser le PCB des deux cartes. Puis suite à leurs commandes, les monter, et dans un deuxième temps développer le firmware.

3. Hardware

Vous trouverez le schéma électrique complet en annexes, qui a été expliqué précédemment.

Si dessous vous trouverez les différentes vues du PCB réalisé à partir des schémas.

3.1. MainBoard

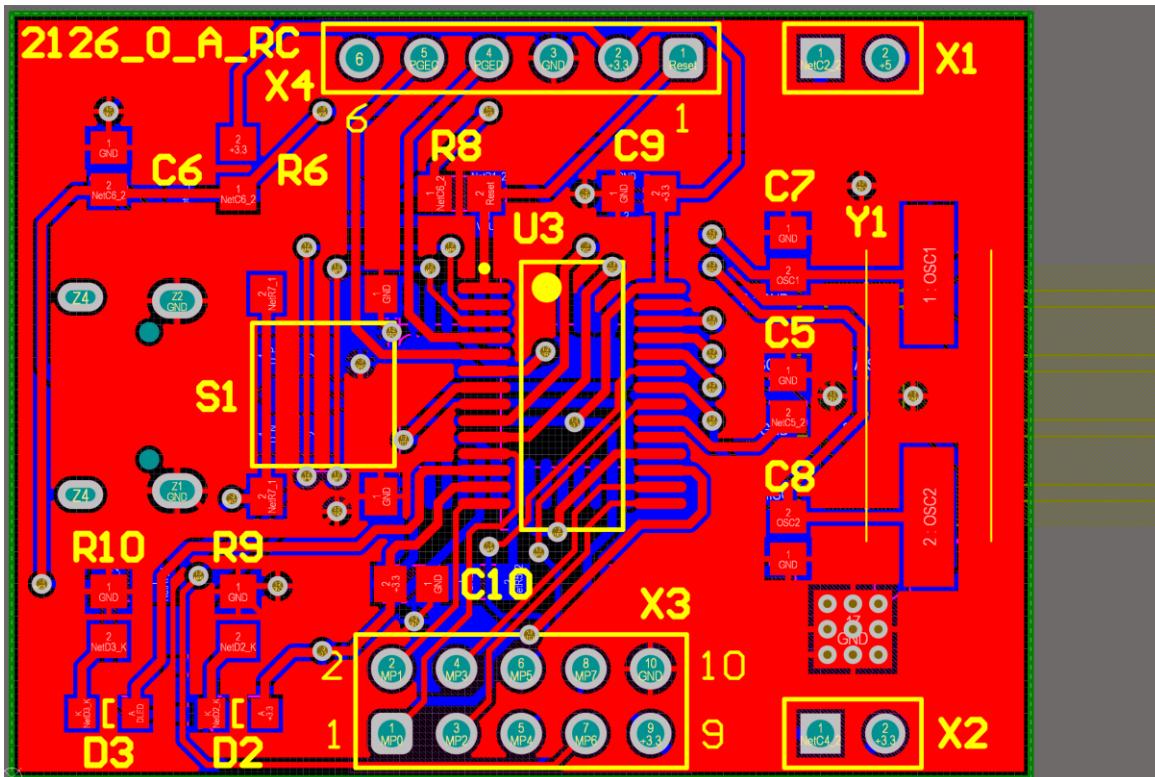


Figure 36 Vue du PCB TOP 2D de la Main Board version A

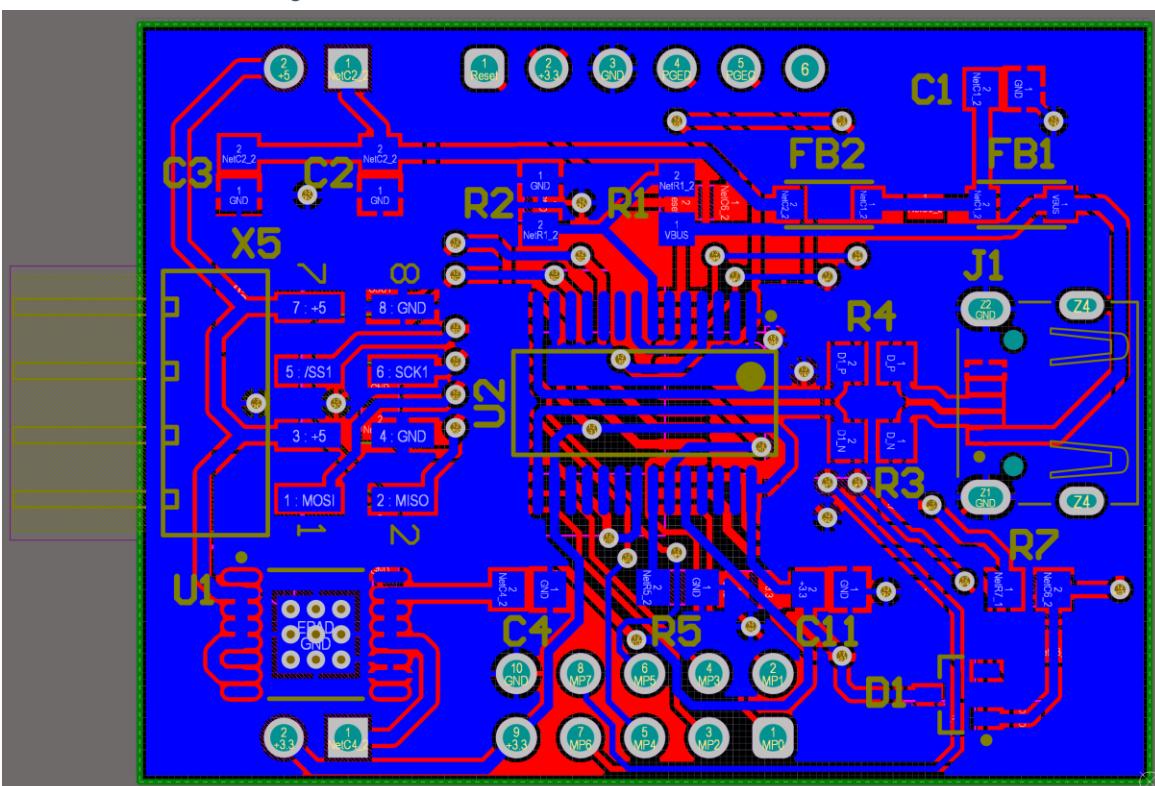


Figure 37 Vue du PCB BOT 2D de la Main Board version A

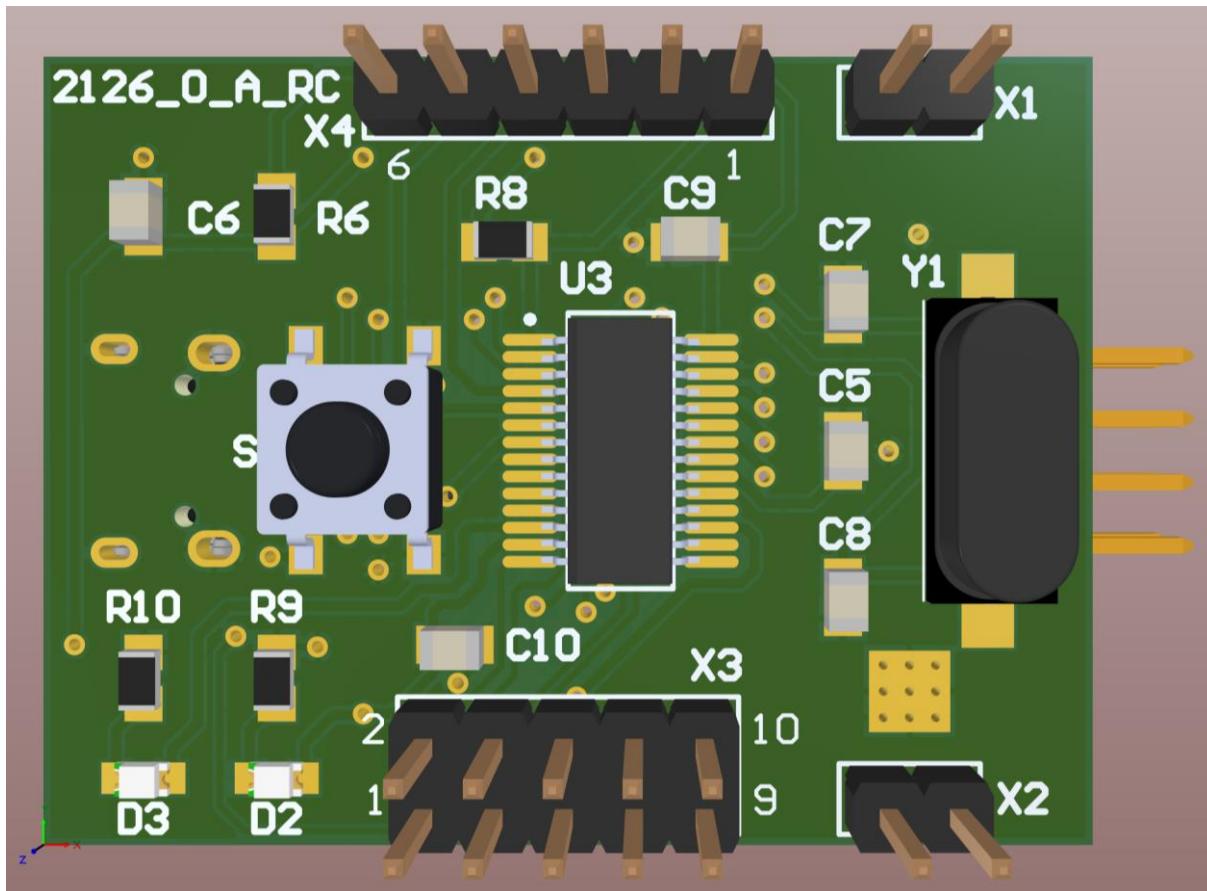


Figure 38 Vue du PCB TOP 3D de la Main Board version A

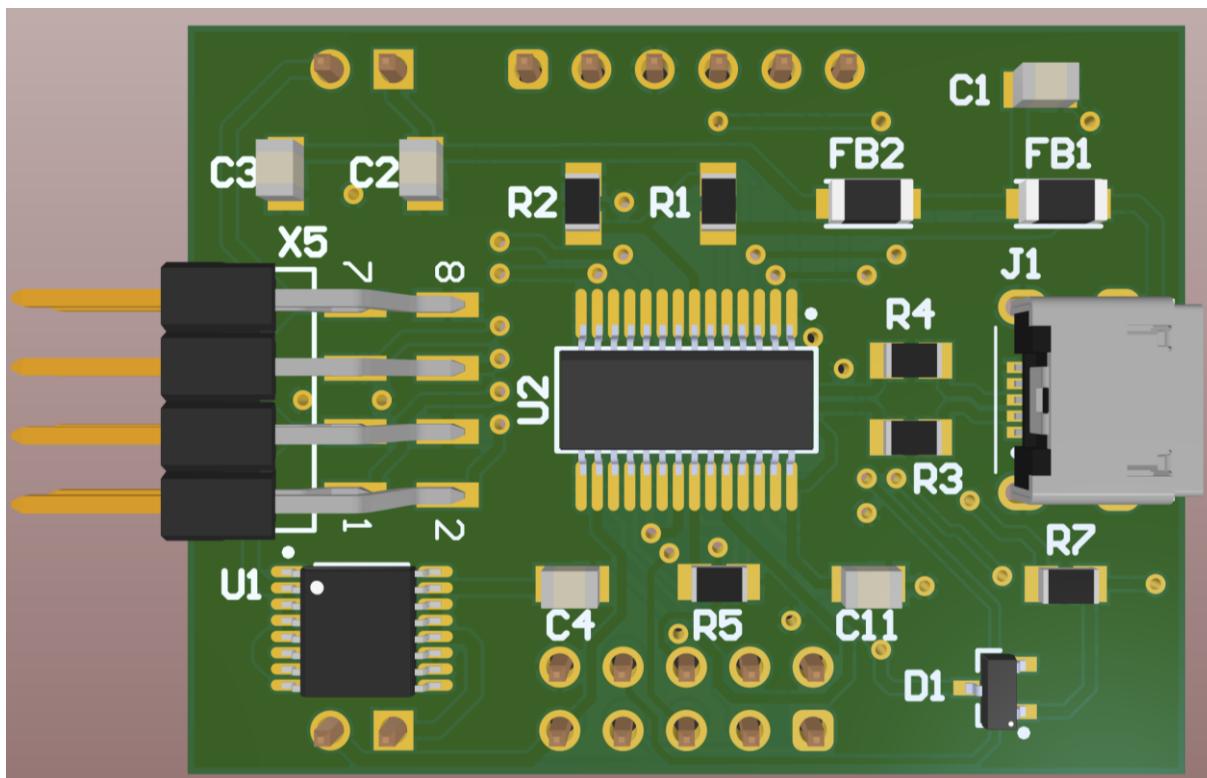


Figure 39 Vue du PCB BOT 3D de la Main Board version A

3.1.1. Spécifications

Pour la réalisation de la MainBoard je n'ai eu aucune contrainte. J'ai donc pu choisir le nombre de couches, la taille et la forme du PCB librement.

Je ne me suis pas fixé une taille de PCB non plus, j'ai plutôt fait la démarche de placer d'abord les composants qui gravitent autour des circuits intégrés proches les uns des autres, un peu comme des groupes.

Puis au fil et à mesure, j'ai rassemblé les groupes et sérés de plus en plus les composant entre eux afin de réduire la taille du PCB. Pour le choix de quel composant va sur quelle face du PCB, j'ai fait à chaque fois au plus pratique pour relier les blocs les uns des autres les plus proches.

Il a juste fallu faire attention à avoir une paire différentielle la plus courte possible entre le connecteur micro-USB, et le convertisseur USB vers UART.

La seule chose que j'ai voulu respecter, c'est que l'on puisse facilement enficher les cartes entre elles à l'horizontale.

3.1.2. Fabrication

La carte a été entièrement montée à la main avec un fer à braser.

Vu le choix judicieux d'avoir placé des jumpers aux points stratégiques des alimentations de la carte, tous les composants ont pu être montés d'un seul coup.

3.2. Matrix

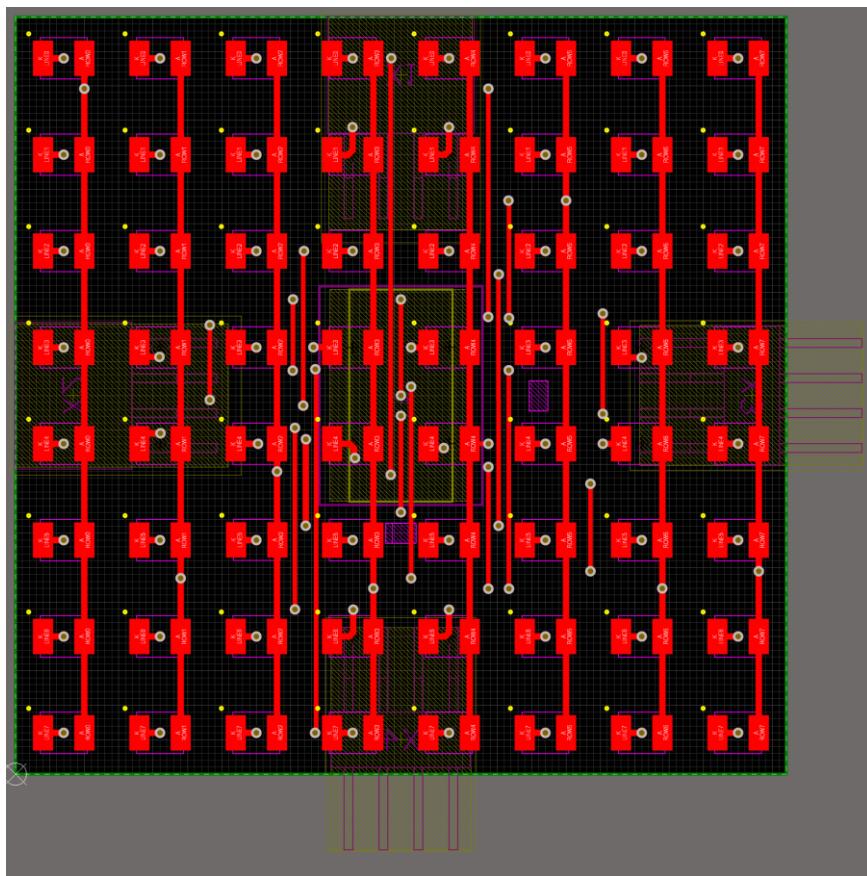


Figure 40 Vue du PCB TOP 2D de la Matrix version B

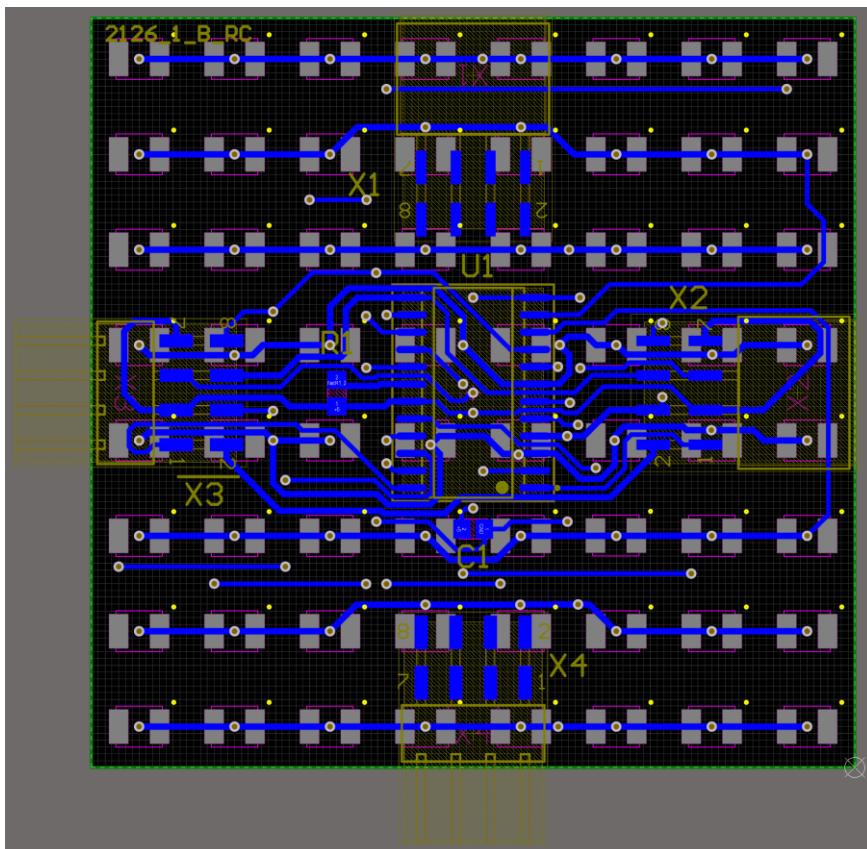


Figure 41 Vue du PCB BOT 2D de la Matrix version B

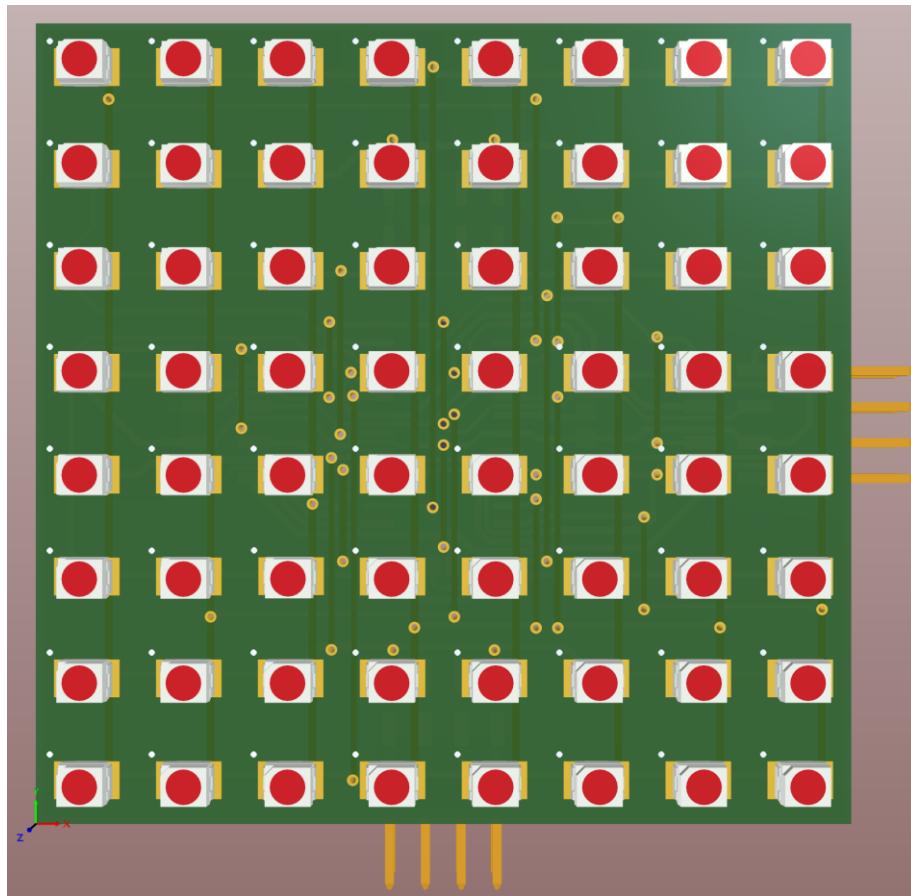


Figure 42 Vue du PCB TOP 3D de la Matrix version B

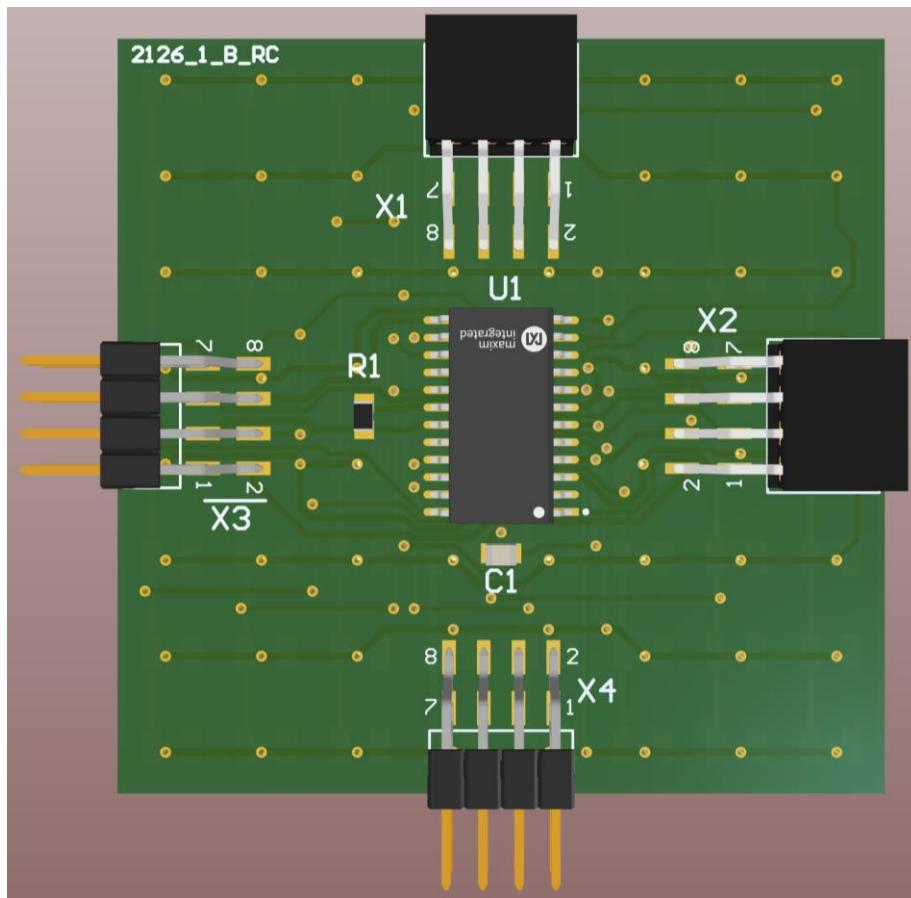


Figure 43 Vue du PCB BOT 3D de la Matrix version B

3.2.1. Spécifications

Pour la réalisation des PCB des Matrix, je n'ai pas non plus eu de très grandes contraintes. La seule chose que je devais respecter, est le fait de pouvoir le connecter à la chaîne en horizontal, mais également en verticale.

Donc on était parti sur le principe de faire des matrices de 8 x 8 LEDs, et d'environ 6cm de large et de haut.

J'ai donc fait en sorte de me rapprocher le plus de ces valeurs avec un espace régulier entre les LEDs. C'est donc un espace de 7mm qui a été choisi, avec lequel j'ai créé une grille dédiée directement sur Altium.



Figure 44 Espace de 7mm entre les LEDs des Matrix

Puis plus tard dans le projet, la comptabilité avec un boîtier m'a été proposée. J'ai donc dû me référer au datasheet du fabricant afin de respecter ses cotations.

Fits 80mm (3.149") X 55.0mm (2.165") PCB

Figure 45 Conseil du fabricant HAMMOND MANUFACTURING (datasheet 1457C80)

J'ai donc dû adapter la largeur, mais également la longueur du PCB pour toujours respecter le format carré du PCB.

3.2.2. Fabrication

Afin de pouvoir monter toutes les LEDs de manière la plus alignée possible, le choix s'est porté sur une placeuse manuelle. C'est-à-dire que grâce à un système de bras se déplacent uniquement sur l'axe x et y, j'ai pu le déplacer avec ma main. J'ai utilisé cette machine pour placer la pâte à brasser à l'aide d'une commande pneumatique, commandé avec une pédale. Puis j'ai pu de la même manière placer les LEDs avec le même système pneumatique.

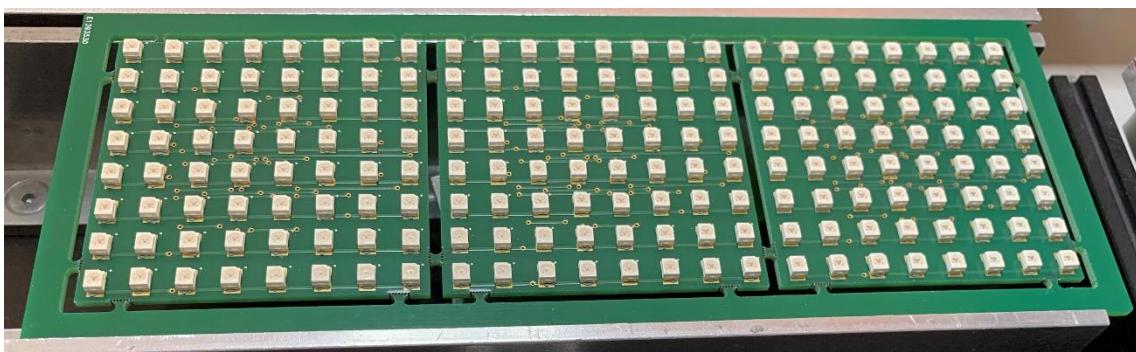


Figure 46 Panneau avec la pâte à braser et les LEDs placés sur les trois Matrix

Une fois la pâte et les LEDs placées, le panneau est passé au four avec trois zones de chaleur, comme spécifié sur le datasheet des LEDs utilisées.

Puis pour l'autre côté du PCB, le restant des composants ont été brisés à la main.

3.3. Boitier

Lors de la réalisation du cahier des charges, un boitier n'était pas prévu, mais simplement un support qui serait directement fixé sur les PCBs.

Juste avant la commande des PCB la proposition m'a été faite, et j'ai accepté de chercher une solution pour mettre en boîtier les Matrix.

Mon choix s'est porté sur un boîtier profilé de chez « HAMMOND MANUFACTURIN », le modèle 1457C801.

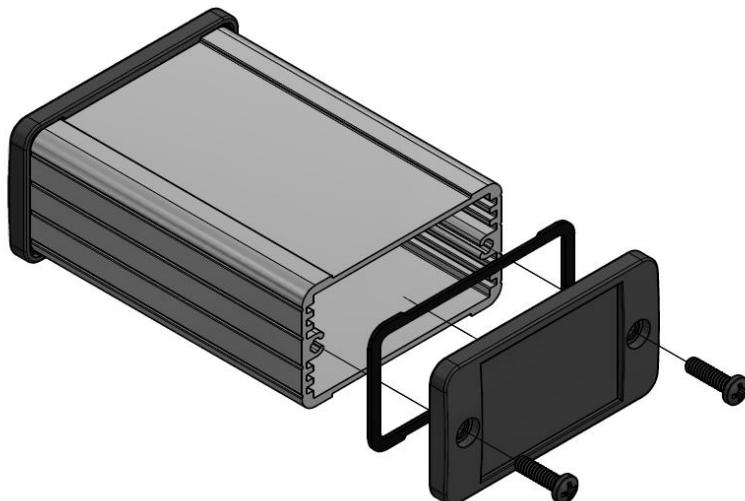


Figure 47 Vue 3D du boîtier choisi (datasheet 1457C80)

Comme vous pouvez le voir, le boîtier est complètement fermé. Afin de pouvoir l'utiliser, il faudra usiner une des faces pour pouvoir faire une ouverture.

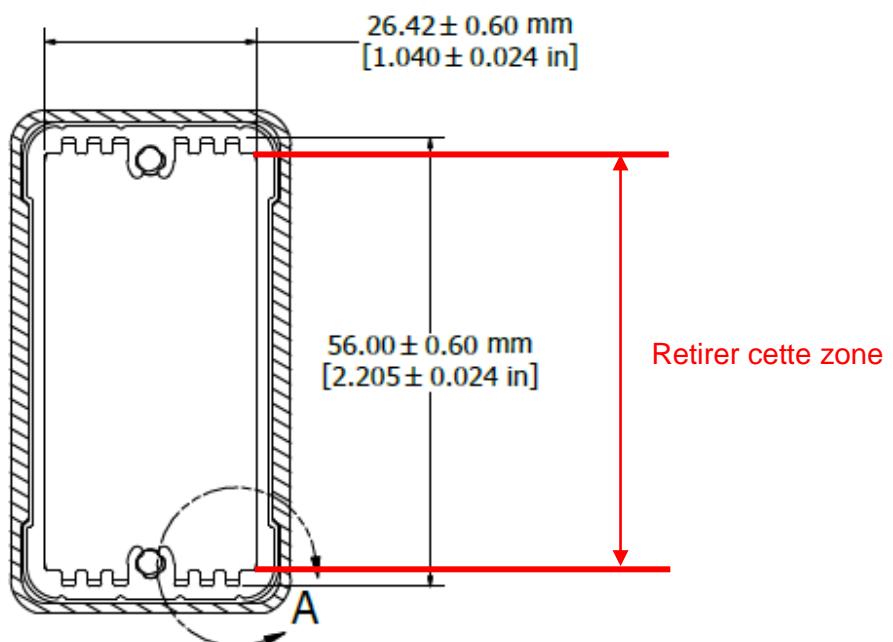


Figure 48 Exemple de découpe de l'ouverture du boîtier (datasheet 1457C80)

Puis vous pourrez laisser l'espace vide, ou alors y coller un diffuseur ou simplement du plexiglas.

3.4. Modifications

3.4.1. MainBoard

Les modifications apportées à la MainBord étaient tellement superficielles et esthétiques, que l'on est resté sur la version A.

3.4.1.1. Valeur C3

Lors de la réalisation du schéma électrique, un condensateur en entrée du MAX1793 de 4.7uF avait été placé. Mais suite à une revue de schéma, il a été retiré pour ne pas surcharger l'alimentation USB avec plus de 10uF. Car un condensateur de 6.8uF se trouve déjà sur la même ligne d'alimentation dans le filtre en T.

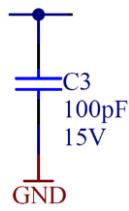


Figure 50 Condensateur C3 première itération version A

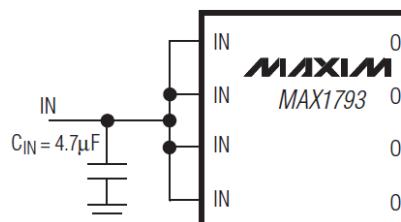


Figure 49 Conseille du fabricant (datasheet MAX1793)

Mais suite à la mise en service, il était impossible de charger un code sur le microcontrôleur. Puis suite à du dépannage grâce aux jumeurs sur les alimentations, on a pu détecter que c'était à cause du condensateur d'entrée qui n'était pas assez grand, comme conseillé sur le datasheet du fabricant du composant.

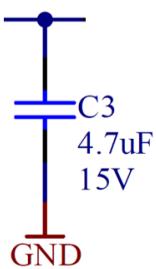


Figure 52 Condensateur C3 deuxième itération version A



Figure 51 Modification ajout condo 4.7uF C3

Suite à l'ajout de la nouvelle valeur du condensateur C3, j'ai pu programmer correctement le microcontrôleur qui était cette fois-ci correctement alimenté.

3.4.1.2. Insertion Silkscreen C5 et C8

Il se trouve que lors du montage du PCB de la MainBoard, j'ai remarqué que l'indication des condensateurs C5 et C8 étaient inversées. J'ai donc pu les placer correctement lors du montage. Puis la modification a été faite sur le design Altium de la version A.



Figure 54 Inversion du Silkscreen de C8 et C5

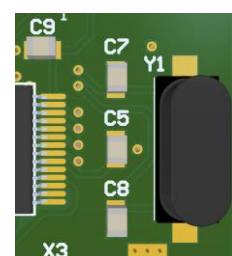


Figure 53 Correction du Silkscreen de C8 et C5 version A

3.4.2. Matrix

3.4.2.1. Pull up R1 passe à pull down

Lors de la réalisation du Firmware je n'arrivais pas à allumer une seule LED via les MAX7221. C'est un problème sur lequel je suis resté bloqué pendant plus d'une semaine. Malgré l'intervention de mon supérieur technique, aucune erreur n'a pu être trouvée, bien qu'au niveau Firmware qu'au niveau Hardware.

C'est alors que j'ai décidé de directement contacter le fabricant du MAX7221, qui est « Maxim Integrated », appartenant à « Analog Devices ». J'ai pu prendre contact via le formulaire de contact sur leur site, puis suite à ma demande j'ai pu directement discuter avec un ingénieur.

C'est après plusieurs échanges de mail que j'en suis venu à lui envoyer mon schéma. C'est là qu'il a pu remarquer une erreur au niveau du schéma.

Il m'a indiqué qu'il conseillait dans le datasheet du MAX7221 de mettre la résistance qui fixe le courant au VCC, et non au GND.

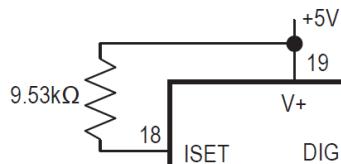


Figure 55 Connexion de R_{ISET} (datasheet MAX7221)

Suite à la mise en évidence de cette information, j'ai pu faire les changements nécessaires dans un premier temps en modifiant les trois cartes Matrix que l'on avait commandées.

Puis dans un deuxième temps j'ai directement fait la modification sur la version B du projet des Altium des Matrix.

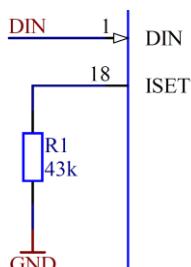


Figure 57 Schéma Matrix version A

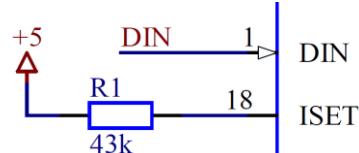


Figure 56 Schéma Matrix version B

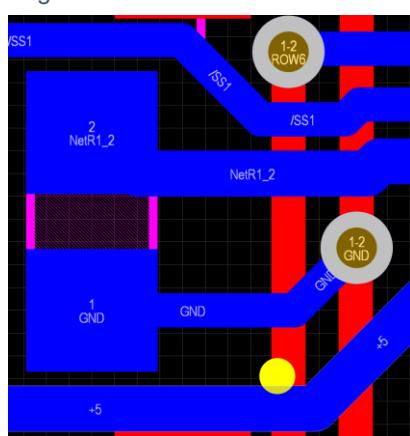


Figure 59 PCB Matrix version A

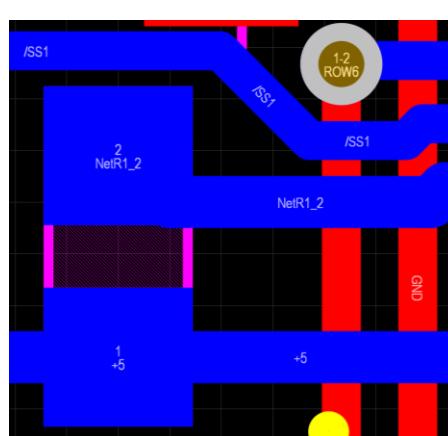


Figure 58 PCB Matrix version B

Malgré les multiples revues de schéma et de PCB, l'erreur n'a pas été trouvée. Il y a donc eu une faille dans le processus de vérification.

3.4.2.2. Modification de la taille du PCB

Lors de la réalisation du PCB, je suis parti d'un espacement fixe entre les LEDs afin de dimensionner le PCB. Puis juste avant de faire la commande des PCB, une proposition de mise en boitier m'a été faite.

Afin de respecter les dimensions conseillées du datasheet du boitier, j'ai dû enlever un millimètre de largeur du PCB.

Mais afin de respecter la possibilité de placer les Matrix côte à côte à l'horizontale et à la verticale, j'ai tenu à également retirer un millimètre sur la longueur du PCB.

En espérant que grâce à la mauvaise précision du fabricant, et surtout les restes des accroches entre les PCB sur les panneaux, que le mauvais alignement ne se voie pas trop.

Malheureusement à la réception des PCB, le mauvais alignement était visible. En effet la précision du fabricant était telle que les dimensions du PCB respectaient exactement le design. Au niveau des déchets, je n'ai même pas eu besoin de limer les PCBs après les avoir extraits du panel.

Pour la version B, le choix a été pris conjointement avec mon supérieur Technique, de faire l'impasse sur la forme carrée du PCB. J'ai donc remis le millimètre manquant sur la longueur, mais maintenant les PCB sont rectangulaires. N'est au moins maintenant l'alignement devrait être parfait dans la version B.

Si toute fois lors d'une future utilisation des Matrix pour faire par exemple plusieurs lignes et colonne, il faudrait faire une version C. Avec cette fois-ci des cartes rectangulaires, mais qui ne seraient plus compatibles avec le boitier en profilé.

4. Firmware

4.1. Flowchart

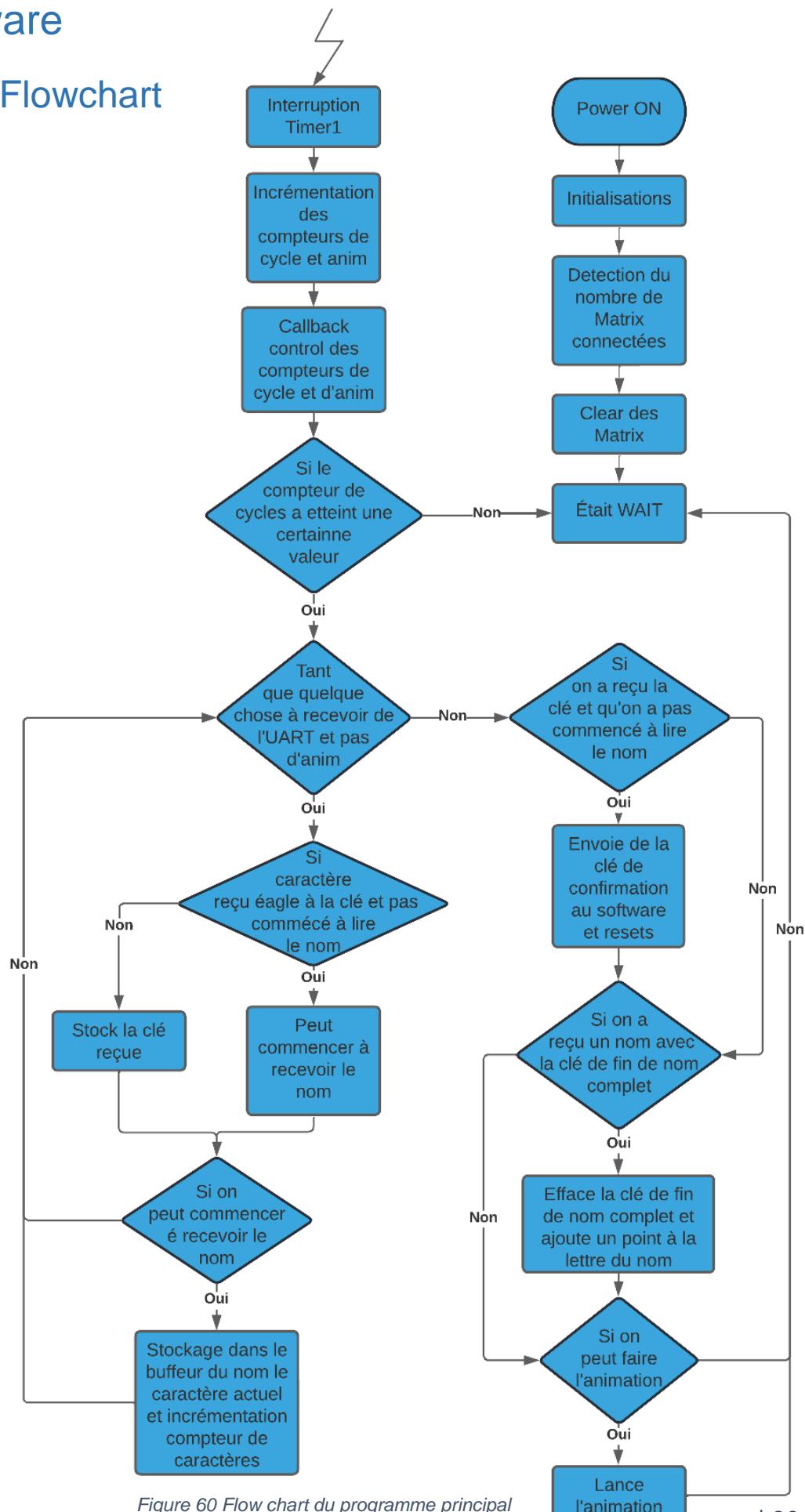


Figure 60 Flow chart du programme principal

4.2. Initialisations et cycles d'interruption

4.2.1. Configurations et initialisations

4.2.1.1. Timer1

Le programme comporte qu'un seul timer pour le rythmer toutes les 10ms. Pour ce faire on a simplement utilisé le même calcul que sur la datasheet du PIC32 pour un clock de 40MHz.

$$\text{Autoreload} = \frac{T}{\text{prescaler} * T_{CLK}} = \frac{10 * 10^{-3}}{1 * 25 * 10^{-9}} - 1 = 39999$$

On doit donc configurer le Timer1 avec un prescaler à 1 et une valeur d'autoreload à 39999.

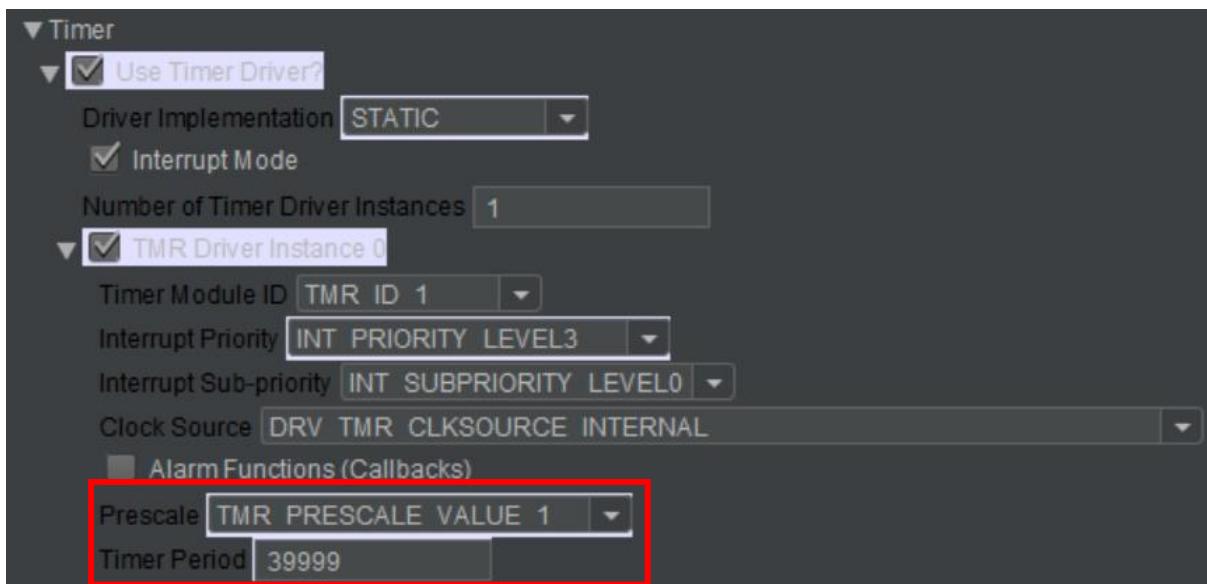


Figure 61 Configuration du Timer1 dans le « Harmony Configurator » de MPLAB

Cette configuration a été directement faite depuis le « Harmony Configurator » accessible dans Tools, puis dans Embedded, directement depuis MPLAB.

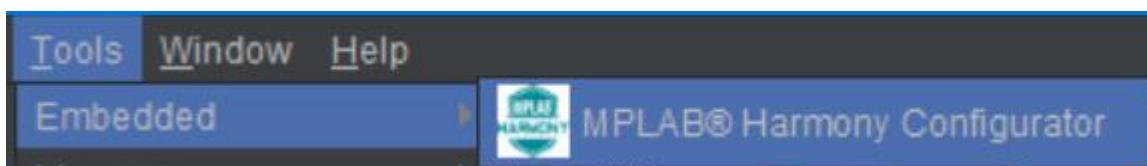


Figure 62 Ouverture du « Harmony Configurator » dans MPLAB

J'ai choisi le niveau de priorité, car c'est celui que l'on utilisait le plus souvent lors de nos travaux pratiques de MINF quand on utilisait notre PIC32.

4.2.1.2. 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 bit de data, pas de parité, un bit de stop et pas d'utilisation du Handshake.

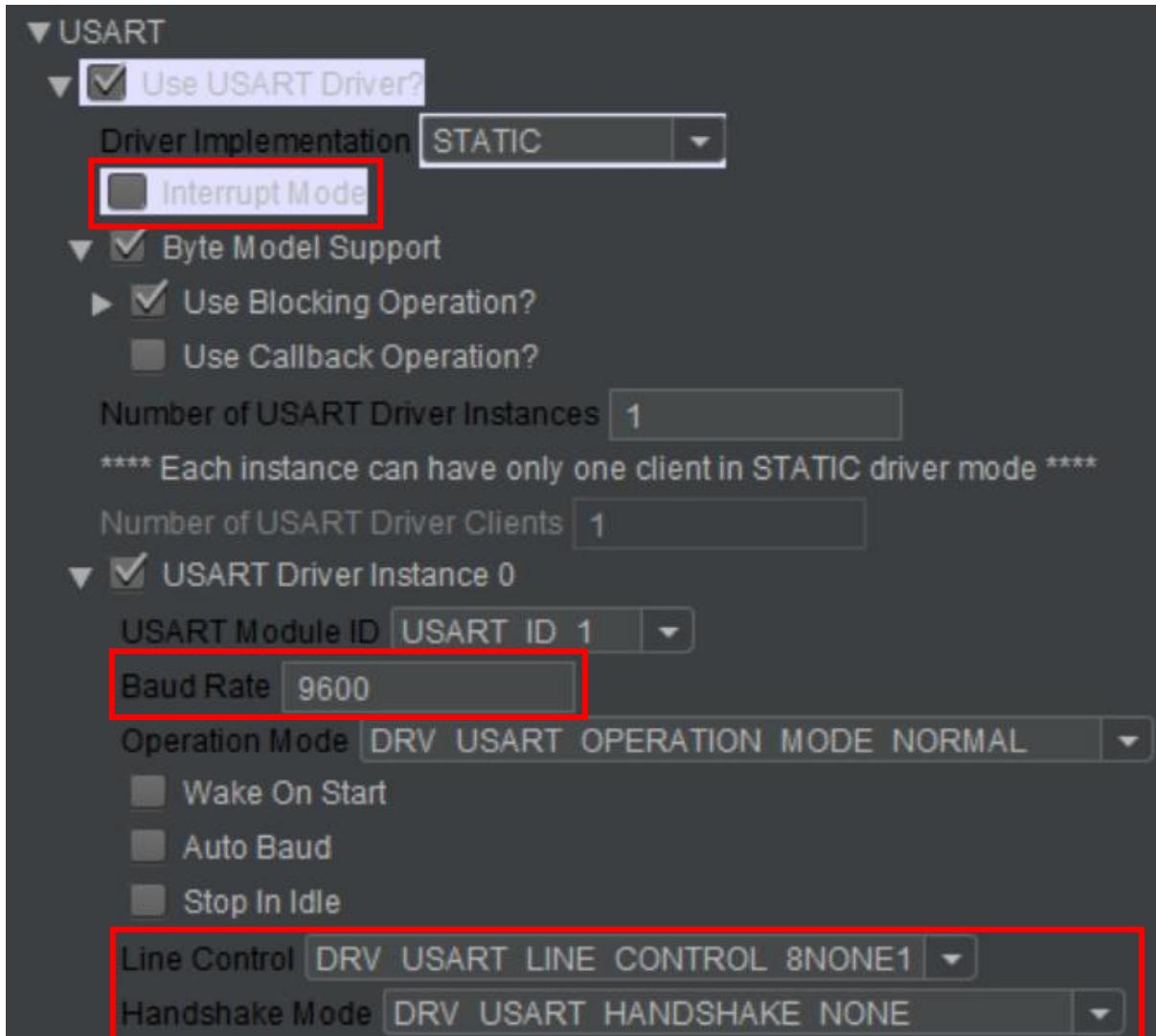


Figure 63 Configuration de l'UART1 dans le « Harmony Configurator » de MPLAB

Vous pouvez également remarquer que la coche « Interrupt Mode » est décochée, c'est normal, car on veut fonctionner par Polling.

En effet le mode par interruption n'est pas utile, car on n'a pas de problème à utiliser les fonctions bloquantes qu'engendre la méthode par Polling de l'UART. Car c'est la première action que le firmware va faire, et il ne passera pas à l'affichage avant de recevoir un nom. Puis une fois qu'il affichera le nom, on n'aura plus besoin de lire le buffeur de l'UART.

4.2.1.3. SPI

Pour la configuration du SPI, j'ai dû me référer aux exemples de trames présentes dans le datasheet du MAX7221.

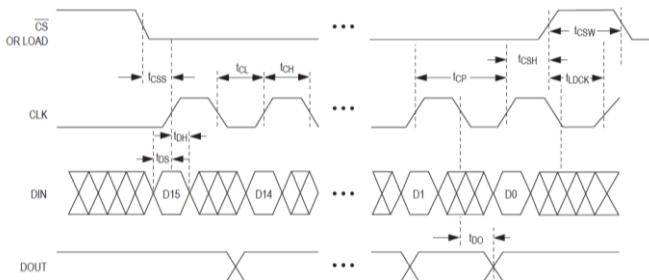


Figure 64 Forme de la trame de communication SPI du MAX7221 (datasheet MAX7221)

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.

Il ne reste plus qu'à choisir une fréquence, et cette information se trouve directement dans le datasheet du MAX7221, qui est de 10MHz conseillé et mode d'envoi d'un byte à la fois.

Puis pour le SPI on utilisera également la méthode par Polling, car on n'a pas de lecture et écriture très active dans le fonctionnement normal du system complet. En effet si on utilise de fonctions bloquantes pour juste écrire dans des registres en boucle, cela ne pose aucun problème.

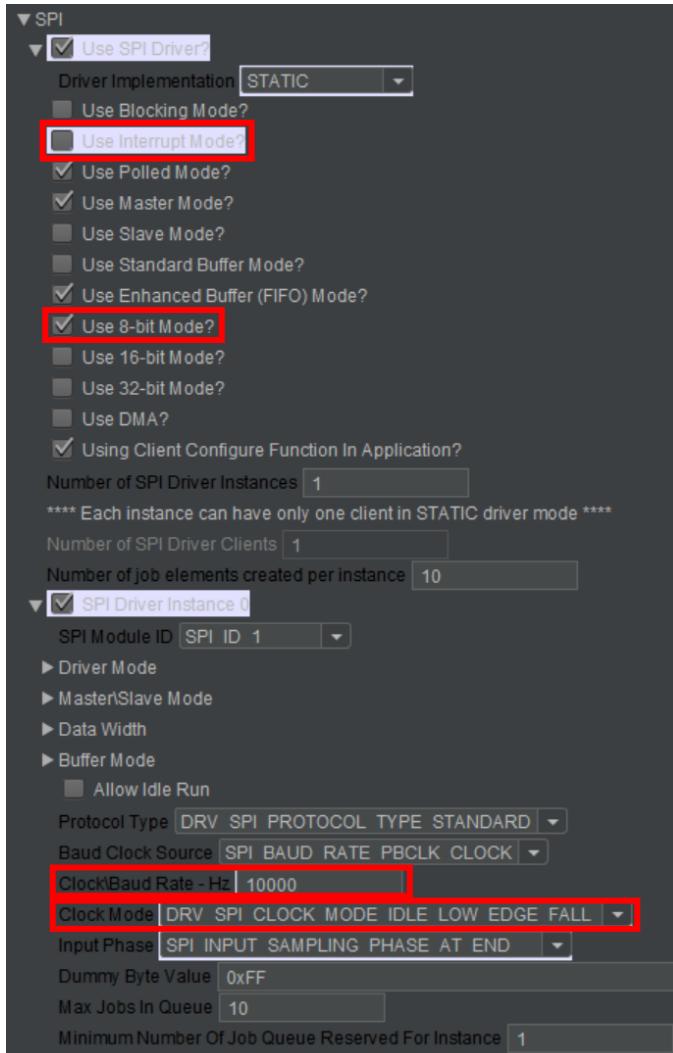


Figure 65 Configuration du SPI 1 dans le « Harmony Configurator » de MPLAB

4.2.2. Interruption et Callback

Afin de minimiser le temps d'exécution de l'interruption du Timer1, j'ai utilisé la méthodologie de fonctions Callback. C'est-à-dire que l'on va déporter la charge d'exécution pendant l'exécution de l'interruption via des flags.

La seule action en plus du clear du flag de l'interruption, est seulement l'appel de la fonction Callback.

```

80 // Main program cadence 10ms
81 void __ISR(_TIMER_1_VECTOR, ipl3AUTO) IntHandlerDrvTmrInstance0(void)
82 {
83     // Appel de la fonction de Callback de commande de la machine d'état principale
84     MainTimer_Callback();
85
86     // Clear du flag d'interruption du Timer1
87     PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
88 }
```

Figure 66 ISR du Timer1 avec l'appel de la fonction de Callback et le clear du flag de l'interruption (system_interrupt.c)

Dans cette fonction « MainTimer_Callback » on va simplement faire tourner des compteurs afin d'atteindre nos différents états de la machine d'état principal, ou certaines actions.

```

102 // Fonction Callback du Timer1 pour commander la machine d'état principale
103 void MainTimer_Callback()
104 {
105     // Compteur du temps de cycle principale
106     static uint32_t mainTimerCount = 0;
107     // Compteur du temps de défilement de l'animation
108     static uint32_t shiftTimerCount = 0;
```

Figure 67 Initialisation des compteurs de cycle et d'animation dans la fonction de Callback (app.c)

On va concrètement l'utiliser pour deux actions, la cadence de notre machine d'état principal, et pour la vitesse de défilement de l'animation sur l'affichage des Matrix.

```

113     // Incrémentation compteur du temps de cycle principale
114     mainTimerCount++;
115
116     // Si le compteur du temps de cycle principale a atteint la valeur voulue
117     if (mainTimerCount >= TIMER_MAIN)
118     {
119         // Reset le compteur du temps de cycle principale
120         mainTimerCount = 0;
121         // Active le flag qui appelle la machine d'état principale
122         appData.mainTimerHasOccurred = true;
123     }
```

Figure 68 Incrémentation et test de la valeur pour le flag de la machine d'état principale dans la fonction de Callback (app.c)

C'est donc un simple compteur que l'on va incrémenter à chaque fois que l'on va rentrer dans la fonction de Callback. Puis la valeur de comptage de ce compteur, on va pouvoir la comparer avec une valeur que l'on va définir.

```
47 #define TIMER_MAIN 1 // Valeur de butée pour la machine d'état
```

Figure 69 Valeur de butée pour le compteur de la machine d'état principale (matrix.h)

Puis si on a atteint la valeur souhaitée, on va activer le flag assigné qui sera utilisé par la suite dans la fonction principale « APP_Tasks ».

Ces flags ont été directement ajoutés à la structure principale du programme « APP_DATE », afin d'avoir un accès global dessus.

```

125     /* TODO: Define any additional data used by the application. */
126
127     // Flag qui appelle la machine d'état principale
128     bool mainTimerHasOcurred;
129     // Compteur du temps de défilement de l'animation
130     shiftTimerCount++;
131
132     // Si le compteur du temps de défilement de l'animation a atteint la valeur voulue
133     if (shiftTimerCount >= TIMER_SHIFT)
134     {
135         // Reset le compteur du temps de défilement de l'animation
136         shiftTimerCount = 0;
137         // Active le flag qui effectue la prochaine étape de l'animation
138         appData.shiftTimerHasOcurred = true;
139     }
140 }
```

Figure 71 Incrémentation et test de la valeur pour le flag de l'animation dans la fonction de Callback (app.c)
Vous pourrez remarquer la présence d'un deuxième flag, qui celui-ci est utilisé pour gérer la vitesse de défilement de l'animation.

C'est le même mécanisme de compteur avec une valeur à atteindre, puis activation du flag.

```

49 #define TIMER_SHIFT      5           // Valeur de butée pour l'animation
```

Figure 72 Valeur de butée pour le compteur de l'animation (matrix.h)

Puis une fois le flag de cycle de machin d'état actif, une fois arrivé dans « APP_Tasks » on pourra tester s'il est actif, et si c'est le cas du reset et d'exécuter le changement d'état.

```

192     // Si le flag d'appel de la machine d'état principale est actif
193     if(appData.mainTimerHasOcurred)
194     {
195         // Reset le flag de la machine d'état principale
196         appData.mainTimerHasOcurred = false;
197         // Passe dans l'état TASKS
198         APP_UpdateState(APP_STATE_SERVICE_TASKS);
199     }
```

Figure 73 Test et clear du flag de la machine d'état principale et changement d'état (app.c)

Enfin pour l'animation c'est exactement le même mécanisme de test et clear du flag, sauf que cette fois-ci on appelle la fonction d'animation.

```

322     // Si le flag qui effectue la prochaine étape de l'animation est actif
323     if(appData.shiftTimerHasOcurred && canShift)
324     {
325         // Reset du flag qui effectue la prochaine étape de l'animation
326         appData.shiftTimerHasOcurred = false;
327         // Effectue l'animation de défilement
328         ShiftAllMatrixRow(tbMatrix);
329     }
```

Figure 74 Test et clear du flag de l'animation et appel de la fonction d'animation (app.c)

4.3. Détection automatique du nombre de Matrix connectées

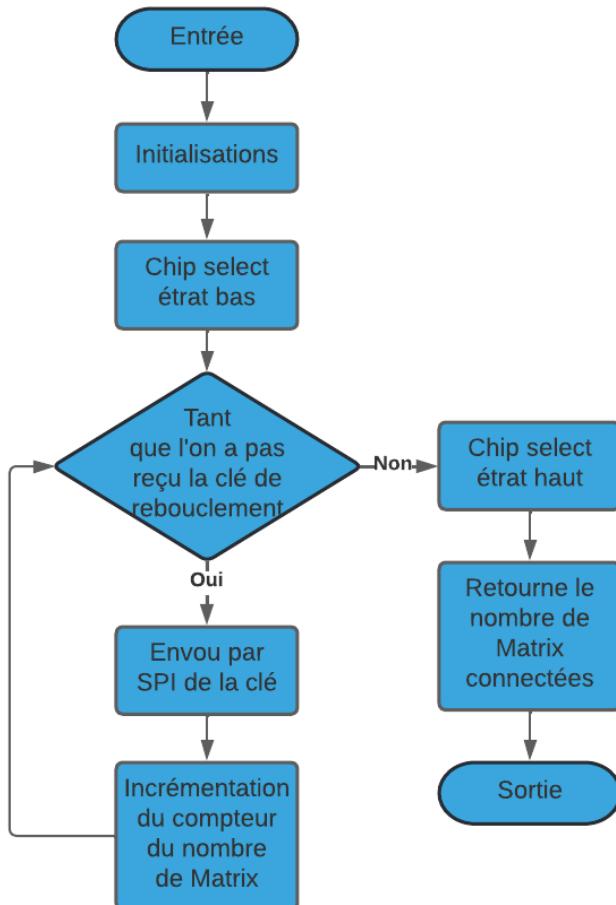


Figure 75 Flowchart de la fonction de détection du nombre de Matrix connectées

Comme expliqué au point 2.3.6, il suffit de positionner un jumper entre le DOUT et le MISO de la dernière Matrix afin de relier le MOSI et le MISO de l'SPI.

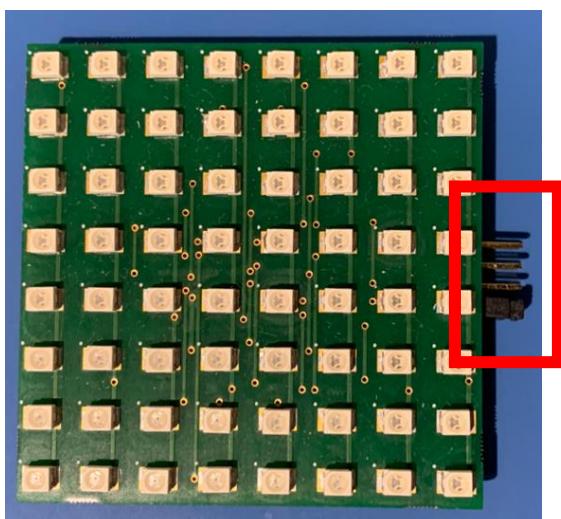


Figure 77 Vue du dessus du jumper

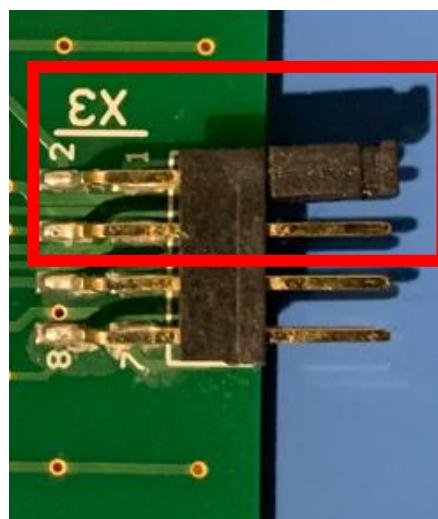


Figure 76 Vue du dessous du jumper

Grâce à cela on a pu envoyer des données et les relier une fois qu'elles ont traversé tous les registres à décalages des MAX7221 présents sur les Matrix.

Lorsque l'on appelle la fonction de détection du nombre de Matrix connectée à l'initialisation, on commence par initialiser des variables locales et on baisse le chip select du SPI.

```

109 // Fait la détection du nombre de Matrix qui sont connectées
110 void FindNumberMatrix(uint8_t* _numberMatrix)
111 {
112     uint8_t bufferReadKey;           // Buffeur de lecture de la clé
113     uint8_t keyMatrix = 123;        // Clé de reboucllement des Matrix
114     static uint8_t counterNumberMatrix = 0; // Compteur du nombre de Matrix connectées
115
116     // Set le chip select à l'état bas pour commencer la communication SPI
117     startSS();

```

Figure 78 Initialisations de la fonction qui détecte le nombre de Matrix connectées (matrix.c)

Puis on commence par remplir les registres de valeurs nuls, ce qui nous permet de partir de registres vides, afin d'éviter que l'on détecte par inadvertance notre clé potentiellement précédemment envoyée.

```

119     // Clear les registres d'un certain nombre imaginaire de Matrix
120     FillRegisterMatrix();

```

Figure 79 Appelle de la fonction de remplissage des registres des Matrix de caractères nuls (matrix.c)

Dans cette fonction on va simplement envoyer des caractères nuls, en l'occurrence 0x00 dans les registres des MAX7221 des Matrix.

```

137 // Clear les registres d'un certain nombre imaginaire de Matrix
138 void FillRegisterMatrix()
139 {
140     uint8_t i;                  // Compteur de la boucle de remplissage des registres
141
142     // Envoie pour un certain nombre imaginaire de Matrix d'un remplissage de registre
143     for(i = 0; i < NUM_MATRIX_CLEAR; i++)
144     {
145         // Envoye un remplissage de registre pour le clear sur les Matrix
146         SendOneByteRaw(MAX7221_REG_NOOP);
147     }
148 }

```

Figure 80 Fonction de remplissage des registres des Matrix de caractères nuls (matrix.c)

Vous pourriez vous demander comment j'ai choisi le nombre de registres que j'ai voulu remplir, c'est assez simple. Il y a deux registres d'un byte par Matrix, j'ai tout simplement choisi d'en remplir potentiellement 16 Matrix, j'ai donc fait 32 envois.

Pour l'envoi de ces données, comme le chip sélec est déjà à l'état bas quand on rentre dans cette fonction, on envoie uniquement le byte demandé grâce à la fonction « SendOneByteRaw ».

```

355 // Envoie uniquement un byte via le SPI
356 void SendOneByteRaw(uint8_t _data)
357 {
358     // Écrit le byte avec la fonction de lecture du SPI1 pour clear le buffer hardware
359     spi_read1(_data);
360 }

```

Figure 81 Fonction d'écriture d'uniquement un byte (matrix.c)

Dans cette fonction on retrouve la fonction « spi_read1 » qui nous vient de la library « Mc32SpiUtil.h » fournie par notre école. Vous vous demandez pourquoi je fais une lecture sur le SPI numéro 1 alors que je veux écrire, car pour lire un byte dans une communication SPI il faut envoyer un byte pour que les registres à décalage se décalent. On peut donc envoyer le caractère « 0x00 », puis une lecture est faite, ce qui a pour effet d'effacer la valeur reçue dans les registres. Cette dernière action n'est pas faite par la fonction d'écriture de base du SPI, c'est par un souci de réutilisabilité que j'ai utilisé la fonction de lecture, et non par recréer une fonction d'écriture avec vidage du buffer à la fin.

Cette action qui a permis de remplir potentiellement 16 Matrix pourrait être modifiée dans d'autres versions du projet si le but est de connecter une très grande quantité de Matrix.

Une fois les registres vidés, on peut commencer à envoyer des bytes connus et incrémenter un compteur pour savoir le nombre de byte qu'on a envoyé.

```

122     // Tant que l'on ne reçoit pas la clé de reboulement des matrix
123     do{
124         // Écriture de la clé dans les Matrix via SPI
125         bufferReadKey = spi_read1(keyMatrix);
126         // Incrémentation du compteur du nombre de Matrix
127         counterNumberMatrix++;
128     }while(bufferReadKey != keyMatrix);

```

Figure 82 Envoie des byte et compteur jusqu'à ce que l'on reçoive la clé de reboulement (matrix.c)

Puis on va faire ça jusqu'à ce que l'on reçoive pour la première fois la clé que l'on a envoyée. Cette fois-ci j'utilise la fonction « spi_read1 » pour réellement lire la valeur reçue, mais également pour écrire la clé de reboulement à chaque fois.

Le fonctionnement est assez simple, une fois que le premier registre a été rempli, le coup d'après on remplit le registre suivant, et ainsi de suite. Puis comme on a positionné le jumper sur la dernière Matrix, on peut relier les données une fois qu'elles ont traversé tous les registres.

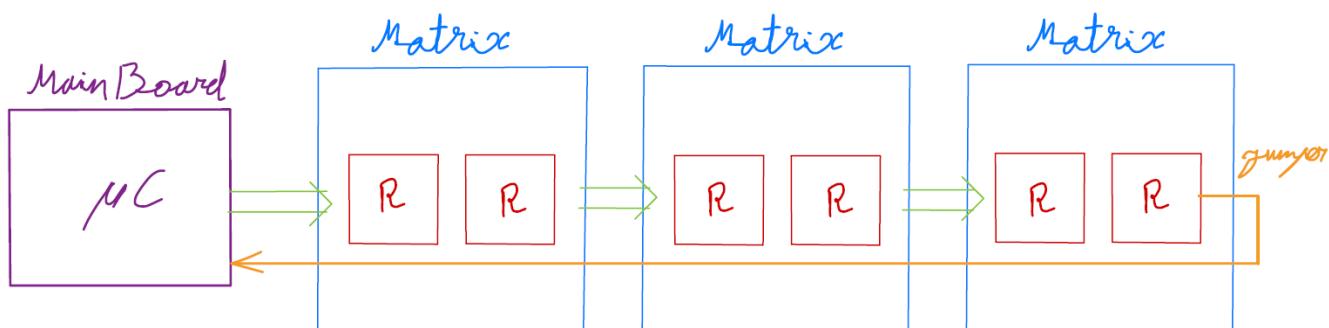


Figure 83 Passage des clés de reboulement dans le system

C'est donc une fois la clé lue pour la première fois que l'on arrête d'envoyer des données, et que l'on remonte le chip select du SPI.

```

130     // Set le chip select à l'état haut pour stoper la communication SPI
131     stopSS();
132
133     // Retourne le nombre de Matrix connectées
134     *_numberMatrix = (counterNumberMatrix - 1) / 2;
135 }

```

Figure 84 Calcul du nombre exacte de Matrix qui sont connectées (matrix.c)

C'est à ce moment-là que l'on peut faire le calcul qui me donnera le nombre de Matrix exacte connectée.

Pour recevoir la première clé lue, on a dû faire un coup de plus que le nombre de registres, c'est donc pour ça que l'on retire un à la valeur de comptage obtenu. Puis il y a deux registres par MAX7221 dont il y en a un par Matrix, il faut donc encore diviser par deux pour avoir la valeur de Matrix qui sont connectées.

Pour finir, on stock cette information dans une variable globale que l'on a passée en paramètre d'entrée, qui sera utilisé dans énormément de fonctions qui dépendent toutes du nombre de Matrix connectées.

4.4. Initialisation des MAX7221

4.4.1. Mécanisme de registre à décalage

Afin de pouvoir commander les MAX7221 comme voulu, on doit au préalable les configurer correctement. Pour cela on va utiliser notre fonction « InitMatrix » qui appliquera les configurations sur toutes les Matrix connectées.

```
216 // Initialise les matrix, donc setup les MAX7221
217     InitMatrix();
```

Figure 85 Fonction d'initialisation des MAX7221 de toutes les Matrix connectées (app.c)

La première chose que l'on va faire est de remplir les registres d'une valeur connue, en l'occurrence de « 0x00 ». On est obligé de le faire, car quand on va commencer à configurer la première Matrix, et avec les chips selects interconnectées, les valeurs qui seront sur les autres Matrix devront être connues, pour ne pas faire de configurations involontaires.

```
306 // Initialisation des MAX7221 de toutes les Matrix connectées
307 void InitMatrix ()
308 {
309     // Vide tous les registres des Matrix connectées
310     ShiftNooMatrix();
```

Figure 86 Vidage des registres de toutes les Matrix connectées (matrix.c)

Pour ce faire on va utiliser la fonction « ShiftNooMatrix », qui va nous envoyer directement que des caractères nuls dans les registres.

```
329 // Vide tous les registres des Matrix connectées
330 void ShiftNooMatrix()
331 {
332     int i = 0;           // Compteur de la boucle du nombre de Matrix connectées
333
334     // Pour toutes les Matrix qui sont connectées
335     for(i = 0; i < maxMatrix; i++)
336     {
337         // Envoie des caractères nuls sur la Matrix actuel
338         SendByte(MAX7221_REG_NOOP, 0x00);
339     }
340 }
```

Figure 87 Fonction qui envoie des caractères nuls sur les registres des toutes les Matrix connectées (matrix.c)

Le vidage est fait sur la totalité des Matrix connectées, que l'on sait maintenant grâce à notre précédente fonction de détection automatique.

C'est donc avec la fonction « SendByte » que l'on va envoyer ces informations via le SPI.

```
342 // Envoie un byte d'adresse et un byte de data avec la gestion des chip select du SPI
343 void SendByte(uint8_t _addr, uint8_t _data)
344 {
345     // Set le chip select à l'état bas pour commencer la communication SPI
346     startSS();
347     // Écrit le byte d'adresse avec la fonction de lecture du SPI1 pour clear le buffer hardware
348     spi_readl(_addr);
349     // Écrit le byte de data avec la fonction de lecture du SPI1 pour clear le buffer hardware
350     spi_readl(_data);
351     // Set le chip select à l'état haut pour stoper la communication SPI
352     stopSS();
353 }
```

Figure 88 Fonction d'envoi de deux bytes avec la gestion des chip select du SPI (matrix.c)

Ici on a simplement en plus de la gestion du chip select du SPI1, l'envoi de deux bytes.

Une fois les registres remplis d'une valeur connue « 0x00 », qui représente la valeur de « no operation », on peut commencer la configuration. Pour toutes les configurations suivantes on va utiliser la même mécanique.

C'est-à-dire que l'on a un système d'adresse et de données, qui rentrent dans l'ordre du MSB pour l'adresse au LSB pour les données. Ce registre a donc la taille de deux bytes pour pouvoir accéder à une adresse spécifique et au même temps y insérer la valeur souhaitée.

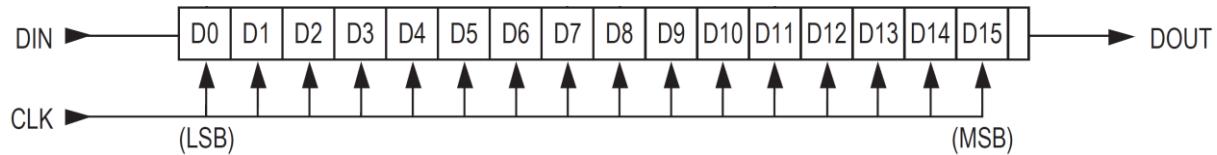


Figure 89 Extrait de la disposition des registres à décalage du MAX7221 (datasheet MAX7221)

Afin de savoir à quelle adresse se trouve quelle configuration, on a un tableau extrêmement utile dans le datasheet du MAX7221.

Table 2. Register Address Map

REGISTER	ADDRESS					HEX CODE
	D15–D12	D11	D10	D9	D8	
No-Op	X	0	0	0	0	0xX0
Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8
Decode Mode	X	1	0	0	1	0xX9
Intensity	X	1	0	1	0	0XA
Scan Limit	X	1	0	1	1	0XB
Shutdown	X	1	1	0	0	0XC
Display Test	X	1	1	1	1	0XF

Figure 90 Extrait de la table avec les adresses du MAX7221 (datasheet MAX7221)

4.4.2. Configuration des registres

4.4.2.1. Scan Limit

Maintenant la disposition connue on peut commencer avec la première configuration, qui est le « Scan Mode ». Cette configuration est à l'origine designé pour un nombre de digits de sept-segments que l'on affiche au même temps.

Dans notre cas avec une utilisation de Matrix, cela aura un effet sur le nombre de lignes que l'on veut afficher, car les sorties « Digit » sont connectées sur nos lignes.

Table 8. Scan-Limit Register Format (Address (Hex) = 0XB)

SCAN LIMIT	REGISTER DATA								HEX CODE
	D7	D6	D5	D4	D3	D2	D1	D0	
Display digit 0 only*	X	X	X	X	X	0	0	0	0xX0
Display digits 0 & 1*	X	X	X	X	X	0	0	1	0xX1
Display digits 0 1 2*	X	X	X	X	X	0	1	0	0xX2
Display digits 0 1 2 3	X	X	X	X	X	0	1	1	0xX3
Display digits 0 1 2 3 4	X	X	X	X	X	1	0	0	0xX4
Display digits 0 1 2 3 4 5	X	X	X	X	X	1	0	1	0xX5
Display digits 0 1 2 3 4 5 6	X	X	X	X	X	1	1	0	0xX6
Display digits 0 1 2 3 4 5 6 7	X	X	X	X	X	1	1	1	0xX7

Figure 91 Table de configuration du « Scan Limit » (datasheet MAX7221)

Comme on veut avoir pleinement accès aux huit lignes de notre Matrix, on va donc choisir la dernière option qui gère toutes les lignes, avec la valeur « 0x07 ».

On a donc la valeur que l'on veut mettre, il ne manque plus que l'adresse à la quel la mettre, que l'on va la trouver dans le tableau des adresses.

Scan Limit	X	1	0	1	1	0xB
------------	---	---	---	---	---	-----

Figure 92 Adresse de configuration du « Scan Limit » (datasheet MAX7221)

Il ne reste plus qu'à envoyer notre data « 0x07 » à l'adresse « 0x0B » à l'aide de nouveau avec la fonction « SendByte ».

```
312 // Scan Limit
313 SendByte(max7221_reg_scanLimit, 0x07);
```

Figure 93 Configuration du « Scan Limit » (matrix.c)

4.4.2.2. Decode Mode

Puis on peut passer à la configuration du « Decode Mode », qui va nous permettre d'enlever un décodage automatique qui serait effectué par le MAX7221 directement.

Table 4. Decode-Mode Register Examples (Address (Hex) = 0X9)

DECODE MODE	REGISTER DATA								HEX CODE
	D7	D6	D5	D4	D3	D2	D1	D0	
No decode for digits 7–0	0	0	0	0	0	0	0	0	0x00
Code B decode for digit 0 No decode for digits 7–1	0	0	0	0	0	0	0	1	0x01
Code B decode for digits 3–0 No decode for digits 7–4	0	0	0	0	1	1	1	1	0xF
Code B decode for digits 7–0	1	1	1	1	1	1	1	1	0xFF

Figure 94 Table de configuration du « Decode Mode » (datasheet MAX7221)

Dans notre cas on utilise des matrices à LEDs avec un système de lignes et de colonnes, on ne veut donc pas qu'il y ait un décodage pour le format sept-segment. On va donc choisir l'option « No decode », avec la valeur « 0x00 ».

On a donc la valeur que l'on veut mettre, il ne manque plus que l'adresse à la quel la mettre, que l'on va la trouver dans le tableau des adresses.

Decode Mode	X	1	0	0	1	0xX9
-------------	---	---	---	---	---	------

Figure 95 Adresse de configuration du « Decode Mode » (datasheet MAX7221)

Il ne reste plus qu'à envoyer notre data « 0x00 » à l'adresse « 0x09 » à l'aide de nouveau avec la fonction « SendByte ».

```
314 // Decode mode
315 SendByte(max7221_reg_decodeMode, 0x00);
```

Figure 96 Configuration du « Decode Mode » (matrix.c)

4.4.2.3. Display Test

Ensute on va configurer un registre un peut spécial, celui du « Display Test ». Il nous permet de vérifier que les Matrix fonctionnent, et que toutes les LEDs s'allument.

J'ai utilisé cette configuration pour allumer brièvement toutes les LEDs de toutes les Matrix connectées au moment du branchement au PC. Puis tout de suite derrière de reéteindre toutes les LEDs, afin de pouvoir allumer par la suite les LEDs voulues.

**Table 10. Display-Test Register Format
(Address (Hex) = 0XF)**

MODE	REGISTER DATA							
	D7	D6	D5	D4	D3	D2	D1	D0
Normal Operation	X	X	X	X	X	X	X	0
Display Test Mode	X	X	X	X	X	X	X	1

Figure 97 Table de configuration du « Display Test » (datasheet MAX7221)

C'est donc d'abord une valeur de « 0x01 » que l'on va envoyer pour tout allumer, puis une valeur de « 0x00 » pour repasser en mode normal.

On a donc la valeur que l'on veut mettre, il ne manque plus que l'adresse à la quel la mettre, que l'on va la trouver dans le tableau des adresses.

Display Test	X	1	1	1	1	1	0XF
--------------	---	---	---	---	---	---	-----

Figure 98 Adresse de configuration du « Display Test » (datasheet MAX7221)

Il ne reste plus qu'à envoyer notre data « 0x01 » et « 0x00 » à l'adresse « 0x09 » à l'aide de nouveau avec la fonction « SendByte ».

```
316 // Display test
317 SendByte(max7221_reg_displayTest, 0x01);
318 // Display test
319 SendByte(max7221_reg_displayTest, 0x00);
```

Figure 99 Configuration du « Display Test » (matrix.c)

4.4.3. Intensity

Puis on a la configuration « Intensity », qui va nous permettre de régler l'intensité d'éclairage des LEDs des Matrix.

Table 7. Intensity Register Format (Address (Hex) = 0xA)

DUTY CYCLE		D7	D6	D5	D4	D3	D2	D1	D0	HEX CODE
MAX7219	MAX7221									
1/32 (min on)	1/16 (min on)	X	X	X	X	0	0	0	0	0x00
3/32	2/16	X	X	X	X	0	0	0	1	0x01
5/32	3/16	X	X	X	X	0	0	1	0	0x02
7/32	4/16	X	X	X	X	0	0	1	1	0x03
9/32	5/16	X	X	X	X	0	1	0	0	0x04
11/32	6/16	X	X	X	X	0	1	0	1	0x05
13/32	7/16	X	X	X	X	0	1	1	0	0x06
15/32	8/16	X	X	X	X	0	1	1	1	0x07
17/32	9/16	X	X	X	X	1	0	0	0	0x08
19/32	10/16	X	X	X	X	1	0	0	1	0x09
21/32	11/16	X	X	X	X	1	0	1	0	0xA
23/32	12/16	X	X	X	X	1	0	1	1	0xB
25/32	13/16	X	X	X	X	1	1	0	0	0xC
27/32	14/16	X	X	X	X	1	1	0	1	0xD
29/32	15/16	X	X	X	X	1	1	1	0	0xE
31/32	15/16 (max on)	X	X	X	X	1	1	1	1	0xF

Figure 100 Table de configuration de « Intensity » (datasheet MAX7221)

Mon choix c'est d'abord porté sur la valeur au centre du tableau, puis après plusieurs tests lors de la mise en service, cette intensité a été validée, car très visible et réduit encore la consommation d'énergie. Même si le nom reste lisible avec l'intensité la plus basse, pour le confort de l'utilisateur on va donc choisir l'intensité « 15/32 », avec la valeur « 0x07 ».

On a donc la valeur que l'on veut mettre, il ne manque plus que l'adresse à la quel la mettre, que l'on va la trouver dans le tableau des adresses.

Intensity	X	1	0	1	0	0xA
-----------	---	---	---	---	---	-----

Figure 101 Adresse de configuration de « Intensity » (datasheet MAX7221)

Il ne reste plus qu'à envoyer notre data « 0x07 » à l'adresse « 0xA » à l'aide de nouveau avec la fonction « SendByte ».

```
320 // Intensity
321 SendByte(max7221_reg_intensity, INTENSITY);
```

Figure 102 Configuration de « Intensity » (matrix.c)

4.4.4. ShutDown

Il nos reste plus qu'à configurer le registre de « ShutDown », qui par défaut à la mise sous tension est configuré en « Shutdwon Mode ».

Dans ce mode les options de décodage intégrées au MAX7221 ne sont pas actives et l'intensité est réglée au minimum.

C'est pour cela qu'une fois toutes les autres configurations faites que l'on va pouvoir configurer ce registre en « Normal Operation ».

Table 3. Shutdown Register Format (Address (Hex) = 0XC)

MODE	ADDRESS CODE (HEX)	REGISTER DATA							
		D7	D6	D5	D4	D3	D2	D1	D0
Shutdown Mode	0XC	X	X	X	X	X	X	X	X
Normal Operation	0XC	X	X	X	X	X	X	X	1

Figure 103 Table de configuration du « ShutDown » (datasheet MAX7221)

C'est donc la valeur de « 0x01 » que l'on va devoir envoyer pour passer en mode normal.

On a donc la valeur que l'on veut mettre, il ne manque plus que l'adresse à la quel la mettre, que l'on va la trouver dans le tableau des adresses.

Shutdown	X	1	1	0	0	0XC
----------	---	---	---	---	---	-----

Figure 104 Adresse de configuration du « ShutDown » (datasheet MAX7221)

Il ne reste plus qu'à envoyer notre data « 0x07 » à l'adresse « 0x0A » à l'aide de nouveau avec la fonction « SendByte ».

```
322 // No ShutDown
323 SendByte(max7221_reg_shutdown, 0x01);
```

Figure 105 Configuration de « ShutDown » (matrix.c)

Une fois la dernière configuration effectuée, on va encore refaire un appel à la fonction qui vide les registres, afin de pousser les configurations jusqu'à la dernière Matrix connectée.

```
325 // Vide tous les registres des Matrix connectées
326 ShiftNooMatrix();
327 }
```

Figure 106 Vidage des registres des MAX7221 pour finir la configuration de toutes les Matrix connectées (matrix.c)

4.4.5. Affichage

Par la suite on voudra écrire dans les registres pour allumer des LEDs spécifiques. Pour cela on devra utiliser le tableau des adresses des « Digit ».

Digit 0	X	0	0	0	1	0xX1
Digit 1	X	0	0	1	0	0xX2
Digit 2	X	0	0	1	1	0xX3
Digit 3	X	0	1	0	0	0xX4
Digit 4	X	0	1	0	1	0xX5
Digit 5	X	0	1	1	0	0xX6
Digit 6	X	0	1	1	1	0xX7
Digit 7	X	1	0	0	0	0xX8

Figure 107 Tableau des adresses de commande pour des lignes des Matrix (datasheet MAX7221)

Il suffira d'indiquer le numéro de digit qui représente le numéro de la ligne souhaitée dans notre cas, puis d'y insérer la valeur hexadécimale des LEDs à allumer sur la ligne.

4.5. Communication avec le Software

Comme montré sur le flowchart du programme principal, après les initialisations, lorsque l'on rentre dans l'état d'exécution on va par Polling lire les datas du buffer hardware de l'UART si on en a reçu. J'ai choisi l'option du Polling plutôt que l'interruption, cas on effectue les demandes de communication avec le PC qu'avant que l'on commence à afficher les noms. Donc si on a cette partie qui est bloquante, cela ne pose aucun problème, car c'est la seule action faite pensant cette partie du processus. On s'assure également que l'on n'a pas encore commencé l'animation, ce que veut dire que l'on n'a pas encore reçu de nom.

```

248     // État d'exécution
249     case APP_STATE_SERVICE_TASKS:
250     {
251         // Tant que l'on reçoit des datas dans le RX buffer ET pas l'animation
252         while(PLIB_USART_ReceiverDataIsAvailable(USART_ID_1) && !canShift)
253     {

```

Figure 108 Tant que l'on reçoit des datas dans le buffer hardware de l'UART et que l'on n'a pas d'animation (app.c)
C'est le premier état pour détecter si on a reçu la clé que le Software envoie quand il ouvre les ports COM qui sont connectés au PC. La partie Software vous sera expliquée plus en détail plus loin dans le rapport.

Pour cela on récupère le caractère lu et on le compare à notre clé de communication avec le Software, qui est « x ». Une fois le nom récupéré on arrêtera de contrôler cela, c'est donc qu'une seule fois que l'on essaye de faire la liaison entre le PC et la Main Board. Donc si on change de session, il faudra reset la carte en la débranchant et rebranchant au PC.

Puis si on a quelque chose dans le buffer hardware de l'UART, on va donc aller le récupérer en le lisant avec les fonctions de la PLIB.

```

254     // Récupération du caractère depuis le RX buffer
255     character = PLIB_USART_ReceiverByteReceive(USART_ID_1);

```

Figure 109 Lecture du caractère reçu dans le buffer hardware de l'UART (app.c)
Au démarrage on va obligatoirement recevoir la clé de communication avec le software en premier, donc on va d'abord la stocker.

```

257         // Si on a pas reçu la clé de communication et qu'on a pas encore lu le nom
258         if((character != keyCom) && (!startReadName))
259         {
260             // Reset du compteur du nombre de caractères du nom
261             countCar = 0;
262             // Peut démarer la lecture du nom complet
263             startReadName = true;
264         }
265         // Si non on stocke la clé de communication avec le software
266         else
267         {
268             // Sauvegarde de la clé de communication avec le software
269             receiveCharacter = character;
270         }

```

Figure 110 Si on lie notre clé de communication avec le software (app.c)

Puis une fois la clé reçue, on va contrôler que c'est la bonne clé, et si on n'a pas encore commencé à décoder un nom, alors on peut renvoyer la clé de confirmation au software.

```

282         // Si on a reçu la clé du Software ET que l'on lit pas le nom
283         if((receiveCharacter == keyCom) && !startReadName)
284         {

```

Figure 111 Si on reçoit le premier caractère du prénom de l'élève (app.c)

```
183 |     char keyCom = 'x';           // Clé de communication de la part du software
```

Figure 112 Clé de réception de l'annoncement du Software (app.c)

Pour cela on l'a simplement envoyée par l'UART avec la méthode bloquante.

```
285 |     // Tant qu'on a pas fini la chaîne ET que l'on a pas plus de 8 caractères
286 |     while ((key[numberChar] != 0) && (numberChar < ZISE_KEY))
287 |     {
288 |         // Attent que le TX buffeur soit disponible
289 |         while(PLIB_USART_TransmitterBufferIsFull(USART_ID_1));
290 |         // Envoi de la clé de confirmation au software
291 |         PLIB_USART_TransmitterByteSend(USART_ID_1, key[numberChar]);
292 |         // Incrémentation du compteur de nombre de caractères
293 |         numberChar++;
294 |     }
295 |     // Reset du buffer de réception des caractères
296 |     receiveCharacter = ' ';
297 |     // Reset du compteur de nombre de caractères
298 |     numberChar = 0;
299 | }
```

Figure 113 Envoi de la clé de réponse au software et reset des buffers et compteur des caractères (app.c)

Une fois la clé envoyée on efface le buffer de réception software, et on reset le compteur de caractères.

```
182 |     char key[] = {'C', '\0'};           // Clé de confirmation pour le software
```

Figure 114 Clé de confirmation pour le Software (app.c)

Puis à ce moment-là le Software recevra la clé de confirmation, et nous renviera en échange le nom de la session sur la quel l'élève est connecté.

On aura donc de nouveaux caractères à lira dans notre buffeur hardware de l'UART.

```
257 |         // Si on a pas reçu la clé de communication et qu'on a pas encore lu le nom
258 |         if((character != keyCom) && (!startReadName))
259 |         {
260 |             // Reset du compteur du nombre de caractères du nom
261 |             countCar = 0;
262 |             // Peut démarer la lecture du nom complet
263 |             startReadName = true;
264 |         }
```

Figure 115 Peut commencer à stocker le prénom et le nom de l'élève (app.c)

Mais cette fois-ci on n'aura plus la clé dans notre buffer, mais bien des caractères du prénom et du nom de l'élève, on peut donc commencer à stocker le nom.

```
272 |         // Si on peut faire la lecture du nom complet
273 |         if(startReadName)
274 |         {
275 |             // Stockage du caractère actuel dans le buffer de stockage du nom
276 |             buffReadName[countCar] = character;
277 |             // Incrémentation du nombre de caractères du nom
278 |             countCar++;
279 |         }
280 |     }
```

Figure 116 Si on peut commencer à stocker le nom, on le met dans le buffeur d'affichage et incrémentation compteur (app.c)

C'est pourquoi maintenant on stock chaque caractère dans le buffeur de récupération du nom. Cela va être fait jusqu'à ce que le buffeur hardware de l'UART soit vide.

4.6. Traitement du nom

4.6.1. Contrôle du nom reçu

Lors de l'envoi du nom de l'élève, le software glisse une clé de fin de nom pour s'assurer que l'on a reçu le nom en entier. Dans notre cas nous avons « XDR » qui se positionne à la fin du nom. C'est donc grâce à cette clé de fin de nom que l'on peut savoir quand est-ce que l'on peut traiter le nom afin de l'afficher par la suite.

```

301 // Si on a reçu un nom d'élève et qu'on a pas encore d'animation
302 if((buffReadName[0] != 0x20)
303 && (buffReadName[0] != NULL)
304 && (buffReadName[0] != keyCom)
305 && (buffReadName[countCar - 3] == keyEndName[0])
306 && (buffReadName[countCar - 2] == keyEndName[1])
307 && (buffReadName[countCar - 1] == keyEndName[2])
308 && (!canShift))
309 {

```

Figure 117 Si on a reçu le prénom et le nom avec la clé de fin de nom complet (app.c)

On contrôle donc que les trois derniers caractères du nom sont égales à la clé de fin de nom, mais également si la première case n'est pas égale à la clé de communication entre le software.

```

184 char keyEndName[] = {'X', 'D', 'R'}; // Clé de fin de nom complet

```

Figure 118 Clé de fin de nom complet (app.c)

Puis pour effacer d'une certaine manière les caractères de la clé de fin de nom, on va soustraire sa taille moins un à la variable qui a compté le nombre de caractères reçu.

```

310 // Enlève les position de la clé de réception du nom
311 countCar -= (END_NAME_KEY_SIZE - 1);
312 // Insère un '.' après la lettre majuscule du nom de l'élève
313 buffReadName[countCar - 1] = '.';

```

Figure 119 Recalcul du nombre de caractères et ajout d'un point à la fin (app.c)

On l'a donc positionné sur le premier caractère de la clé de fin de nom, car on va le remplacer par un point, ce qui fera comme effet d'ajouter un point après la lettre majuscule du nom de l'élève.

4.6.2. Mise en forme du nom

Une fois le buffeur avec le nom mis au bon format, on va maintenant devoir le transformer et le mettre en forme afin de pouvoir l'afficher sur les LED des Matrix.

Pour cela on va utiliser la fonction « SendText », en indiquant le buffeur avec le nom, mais également le buffeur d'affichage.

```

314 // Écrit le nom réceptionné dans le buffeur d'affichage
315 SendText(buffReadName, tbMatrix);

```

Figure 120 Transformation du nom en format pour les LED des Matrix (app.c)

C'est avec un système de tableau à deux dimensions que l'on va représenter l'affichage réel sur les Matrix.

```

221 // Met une suite de caractères dans le buffeur d'affichage
222 void SendText(char *_text, uint8_t _pMatrix[])[8])
223 {
224     int i;          // Compteur de la boucle pour le nombre de caractères traités
225
226     // Pour tous les caractères du nom
227     for (i = 0; i <= countCar -1; i++)
228     {
229         // Ajoute le caractère actuel au buffeur d'affichage au format pour les Matrix
230         AddCharacter(_text[i], _pMatrix, (i + 1));
231     }
232 }
```

Figure 121 Fonction de conversion du prénom et nom de l'élève au format affichable sur les Matrix (matrix.c)

Dans cette fonction « AddCharacter » on y rentre le caractère actuel, mais également le buffeur d'affichage et le numéro du caractère dans le nom.

Puis une fois que l'on rentre dans cette fonction, on commence par initialiser quelques variables locales.

```

234 // Ajoute un caractère au bon format pour les Matrix dans le buffeur d'affichage
235 void AddCharacter(char _characteer, uint8_t _pMatrix[])[8), uint8_t _witchPlace)
236 {
237     int i;          // Compteur de la boucle pour le nombre de lignes
238
239     uint8_t numCar = _witchPlace; // Pour la position du caractère que l'on place
240     uint8_t placeMatrix = 0;      // Pour indiquer sur quel Matrix on écrit
241     uint8_t startCar = 0;        // Pour la position sur la Matrix dans le quel on écrit
```

Figure 122 Initialisation des variables locales de la fonction « AddCharacter » (matrix.c)

Puis afin d'optimiser l'affichage, c'est par ligne entières que l'on va afficher les textes par la suite, donc on écrit les caractères dans le buffeur d'affichage de la même manière.

```

243 // Pour le nombre de lignes de notre Matrix
244 for (i = 0; i < ROW_MATRIX; i++)
245 {
```

Figure 123 Effectue la conversion sur le nombre de lignes que comporte nos Matrix (matrix.c)

Dans notre cas, nos Matrix on huit lignes, mais c'est ici que l'on pourrait par exemple faire le double de ligne si on a deux rangées de Matrix connectées une sur l'autre, afin d'afficher des caractères plus grands, mais cela sera pour une autre version améliorée du projet.

Puis on doit maintenant d'avoir dans quel Matrix est-ce que l'on va devoir écrire notre caractère. On part du principe que tous les caractères se suivent, et nous alors les afficher de gauche à droite dans notre buffer d'affichage.

```

246 // Calcule sur quel Matrix est-ce que l'on doit écrire le caractère sélectionné à la suite
247 placeMatrix = (uint8_t)((numCar - 1) / 1.4);
```

Figure 124 Calcul exacte du nombre de Matrix connectées (matrix.c)

Ici on le numéro du caractère actuel moins un qui va être divisé par le nombre de caractères que l'on peut afficher par Matrix, puis casté en entier. Si on a une font de cinq de large et que l'on met un espace d'une LED de large, on obtient un ratio de 1.4.

$$\text{Numéro de Matrix (uint8_t)} = \frac{\text{NuméroCaractèreActuel} - 1}{\frac{((\text{NombreColones} - \text{LargeurFonte}) - \text{LargeurEspace})}{\text{LargeurFonte}} + 1}$$

On peut faire un exemple pour le caractère à la troisième position, qui sera un 'c'.

$$\text{Numéro de Matrix (uint_8)} = \frac{3 - 1}{\frac{(8 - 5) - 1}{5} + 1} = 1.428 \Rightarrow 1$$

Ici on obtient donc que l'on doit commencer à écrire le caractère numéro trois dans la Matrix numéro une. Sachant que l'on commence avec la Matrix numéro zéro, cela veut donc dire que c'est sur la deuxième Matrix physiquement connectée que l'on va l'afficher.

Puis une fois que l'on sait sur quel Matrix on doit commencer à écrire, on doit maintenant savoir sur quelle LED parmi les huit colonnes est-ce que l'on doit commencer à écrire notre caractère.

```
248     // Calcul sur quel LED de la Matrix actuel est ce que l'on doit commencer à écrire le caractère
249     startCar = (((numCar - 1) * 5) + (numCar - 1)) - (ROW_MATRIX * placeMatrix);
```

Figure 125 Calcul de la position de la LED dans la Matrix à la quel on doit commencer à écrire (matrix.c)

Ici on va calculer la taille que prennent tous les caractères déjà écrits, puis on soustrait la taille de toutes les LED des toutes les Matrix avant celle que l'on doit écrire.

$$\begin{aligned} \text{Position LED} = & [(NumeroCaractèreActuel - 1) * LargeurFonte] \\ & + (NumeroCaractèreActuel - 1) \\ & - (\text{NuméroLignes} * \text{NuméroDeMatrix}) \end{aligned}$$

Prenons de nouveau l'exemple du troisième caractère que l'on sait déjà que l'on doit commencer à écrire dans la Matrix numéro une.

$$\text{Position LED} = [(3 - 1) * 5] + (3 - 1) - (8 * 1) = 4$$

Ici on sait que l'on doit écrire à la LED numéro quatre, cela en tenant compte que l'on commence à partir de la LED zéro, et que l'on va de gauche à droite sur la Matrix.

Ensuite on va pouvoir convertir le caractère en font au format de la matrix, que ce soit de l'alphabet en majuscules, en minuscules, un chiffre ou un point.

```
251     // Majuscules - Si le caractère est compris entre l'alphabet en majuscules de la table ASCII
252     if(_characteer >= 0x41) && (_characteer <= 0x5A)
253     {
```

Figure 126 Si notre caractère fait partie de l'alphabet en majuscules de la table ASCII (matrix.c)

Comme l'implémentation de toute la table ASCII n'a pas été faite, des tests pour savoir dans quelle zone on se trouve ont été faits. Ici on regarde donc si notre troisième caractère est une majuscule. C'est une procédure qui pourra être unifiée si une amélioration du projet est faite. Dans notre cas on a comme caractère un 'c', on a donc affaire à une minuscule.

```
263     }
264     // Minuscules - Si le caractère est compris entre l'alphabet en minuscules de la table ASCII
265     else if (_characteer >= 0x61) && (_characteer <= 0x7A)
266     {
```

Figure 127 Si non si notre caractère fait partie de l'alphabet en minuscules de la table ASCII (matrix.c)

C'est donc dans cette partie que l'on va traiter notre caractère pour le convertir au bon format pour le stocker dans notre buffeur d'affichage.

On va donc commencer par la première ligne, dans la quel on va aller chercher dans notre stockage de la font de l'alphabet minuscule de la table ASCII.

```

51 // Font de l'alphabet en minuscules
52 uint8_t tbLowAlphabet[27][8] = {{0x00,0x00,0x70,0x08,0x78,0x88,0x78,0x00}, // a
53 {0x80,0x80,0xb0,0xc8,0x88,0x88,0xf0,0x00}, // b
54 {0x00,0x00,0x70,0x80,0x80,0x80,0x70,0x00}, // c
55 {0x00,0x00,0x70,0x88,0x88,0x88,0x78,0x00}, // d
56 {0x00,0x00,0x70,0x88,0x88,0x88,0x78,0x00}, // e
57 {0x30,0x40,0xe0,0x40,0x40,0x40,0x40,0x00}, // f
58 {0x00,0x00,0x78,0x88,0x78,0x08,0x70,0x00}, // g
59 {0x80,0x80,0xb0,0xc8,0x88,0x88,0x88,0x00}, // h
60 {0x00,0x20,0x00,0x20,0x60,0x20,0x70,0x00}, // i
61 {0x08,0x00,0x18,0x08,0x08,0x48,0x30,0x00}, // j
62 {0x80,0x80,0x90,0xa0,0xc0,0xa0,0x90,0x00}, // k
63 {0x60,0x20,0x20,0x20,0x20,0x70,0x00}, // l
64 {0x00,0x00,0xd0,0xa8,0xa8,0xa8,0xa8,0x00}, // m
65 {0x00,0x00,0xb0,0xc8,0x88,0x88,0x88,0x00}, // n
66 {0x00,0x00,0x70,0x88,0x88,0x88,0x70,0x00}, // o
67 {0x00,0x00,0xf0,0x88,0xf0,0x80,0x80,0x00}, // p
68 {0x00,0x00,0x88,0x78,0x78,0x08,0x00}, // q
69 {0x00,0x00,0xb0,0xc8,0x80,0x80,0x80,0x00}, // r
70 {0x00,0x00,0x70,0x80,0x70,0x08,0xf0,0x00}, // s
71 {0x40,0x40,0xe0,0x40,0x40,0x48,0x30,0x00}, // t
72 {0x00,0x00,0x88,0x88,0x88,0x98,0x68,0x00}, // u
73 {0x00,0x00,0x88,0x88,0x88,0x88,0x50,0x20,0x00}, // v
74 {0x00,0x00,0x88,0x88,0xa8,0xa8,0x50,0x00}, // w
75 {0x00,0x00,0x88,0x50,0x20,0x50,0x88,0x00}, // x
76 {0x00,0x00,0x88,0x88,0x78,0x08,0x70,0x00}, // y
77 {0x00,0x00,0xf8,0x10,0x20,0x40,0xf0,0x00}, // z
78 {0xf0,0xf8,0x10,0x80,0xf0,0xf0,0x00}}; // Full

```

Figure 128 Tableau avec la font de tout l'alphabet majuscule de la table ASCII (matrix.c)

Puis on va placer cette valeur dans la ligne du numéro de la Matrix, et décalé du numéro de position de notre caractère vers la droite, en sachant qu'on le positionne par défaut à gauche de l'affichage.

```

267 // Converti le caractère minuscules en forme pour les LED de la Matrix actuelle
268 _pMatrix[(placeMatrix)][i] |= (tbLowAlphabet[_characteer - 0x61][i]) >> (startCar);

```

Figure 129 Stockage des valeurs convertis au format pour la Matrix actuelle dans le buffer d'affichage (matrix.c)

Puis on va contrôler que si on commence à écrire notre caractère à la position voulue, qu'avec la taille de l'ont on plus, on ne dépasse pas la taille max d'affichage de la Matrix actuel. Pour calculer cette position max de débordement sur l'autre Matrix calculez :

$$\text{Position Max} = (\text{NombreColones} - \text{LargeurFonte}) + 1 = (8 - 5) + 1 = 4$$

Ici, si on commence à écrire à la position numéro quatre on débordera forcément.

```

269 // Si le caractère minuscules déborde sur la prochaine Matrix
270 if(startCar >= NEXT_CAR)
271 {
272     // Converti le caractère minuscules en forme pour les LED de la Matrix suivante
273     _pMatrix[(placeMatrix + 1)][i]
274     |= (tbLowAlphabet[_characteer - 0x61][i]) << (ROW_MATRIX - (startCar));
275 }
276 }

```

Figure 130 Stockage des valeurs convertis au format pour la Matrix suivante dans le buffer d'affichage (matrix.c)
Comme dans notre cas on commence justement à écrire à la position quatre, on va déborder sur la prochaine Matrix.

On va donc écrire sur la Matrix suivant les données restantes dans notre buffer d'affichage, qui correspondent à notre caractère décalé à gauche du nombre de colonnes, qui est égale au nombre de lignes, car des Matrix carrées, moins la position à la quel on a dû écrire.

$$\text{Nombre de décalage} = \text{NombreColones} - \text{Position LED} = 8 - 4 = 4$$

Dans notre cas on doit donc décaler à gauche de quatre positions notre caractère, en sachant qu'il faut cinq de large et qu'on le positionne par défaut à gauche de l'affichage.

4.6.3. Affichage du nom sur les Matrix

Une fois le prénom, le nom de l'élève et le point mis au bon format et stockés dans le buffeur d'affichage, il nous reste plus qu'à afficher le buffeur réellement dans les Matrix.

```
316 // Envoie le buffeur d'affichage sur les Matrix
317 SendAllMatrixRow(tbMatrix);
```

Figure 131 Fonction d'affichage du buffeur d'affichage sur les Matrix (app.c)

On va d'abord initialiser nos compteurs de boucle pour le nombre de lignes, et le nombre de Matrix.

```
150 // Envoie le buffeur d'affichage sur les Matrix
151 void SendAllMatrixRow(uint8_t _pMatrix[])[8])
152 {
153     int i = 0;           // Compteur de la boucle pour le nombre de lignes
154     int j = 0;           // Compteur pour le nombre de Matrix
```

Figure 132 Initialisations des compteurs pour le nombre de lignes, et le nombre de Matrix (matrix.c)

Puis on va procéder par une méthode d'affichage par lignes, c'est-à-dire que si trois Matrix sont connectées, on va d'abord envoyer la première ligne de toutes les trois Matrix.

```
156 // Pour toutes les lignes qu'il fait afficher
157 for(i = ROW_MATRIX-1; i >= 0; i--)
158 {
159     // Set le chip select à l'état bas pour commencer la communication SPI
160     startSS();
161
162     // Pour toutes les Matrix connectées
163     for(j = (maxMatrix - 1); j >= 0; j--)
164     {
165         // Envoie tout le buffeur d'affichage sur les Matrix
166         SendByteRaw(tbDecodRow[i], ROTATE_RIGHT_BYTE(_pMatrix[j][i]));
167     }
168
169     // Set le chip select à l'état haut pour stoper la communication SPI
170     stopSS();
171 }
172 }
```

Figure 133 Méthode d'affichage du buffeur d'affichage sur les Matrix connectées ligne par ligne (matrix.c)

Donc on va baisser le chip select, puis envoyer toute une ligne, puis remonter le chip select. Ensuite on refait la même chose pour toutes les autres lignes, dans notre cas on en a huit.

La colonne du « dot » se trouvant au LSB, un décalage circulaire d'une position vers la gauche a été nécessaire afin que l'affichage se fasse correctement.

```
51 #define ROTATE_RIGHT_BYTE(x) ((x >> 1) | (x << 7))
```

Figure 134 Macro du décalage circulaire d'une position vers la gauche (matrix.h)

Vous pourrez remarquer que l'on doit d'abord envoyer la fin du tableau et remonter jusqu'à son début, pour pouvoir respecter l'ordre d'affichage, étant donné que la Main Board est connectée à gauche des Matrix, et que données transitent donc de gauche vers la droite.

Cette méthode reste toutefois très bien optimisée contenu de l'architecture de registres à décalage utilisé. Cela veut dire que l'on remplit à chaque fois tous les registres de toutes les MAX7221 de Matrix en une fois avant de valider les données. Ça n'aurait pas été le cas si on affichait toutes les lignes de la première Matrix, puis qu'on passerait à la Matrix suivante. Car on remplirait les registres des Matrix suivantes et on aurait des affichages erronés, car les chip select sont tous inter connectés entre les MAX7221.

4.6.4. Animation de défilement

Une fois le prénom et le nom de l'élève affichés sur le Matrix, on va donner l'autorisation d'activer l'animation.

```

318     // Peut démarer l'animation
319     canShift = true;
320 }
```

Figure 135 Autorisation pour l'animation sur l'affichage des Matrix (app.c)

C'est donc si on arrive au moment de l'animation, régulé par la fonction de callback de l'interruption du Timer1 précédemment expliquée, que l'on va donner la cadence d'affichage.

```

322     // Si le flag qui effectue la prochaine étape de l'animation est actif
323     if(appData.shiftTimerHasOcurred && canShift)
324     {
325         // Reset du flag qui effectue la prochaine étape de l'animation
326         appData.shiftTimerHasOcurred = false;
327         // Effectue l'animation de défilement
328         ShiftAllMatrixRow(tbMatrix);
329     }
```

Figure 136 Si on peut faire l'animation de défilement et que l'on a l'autorisation de le faire (app.c)

Une fois rentré dans la condition, on va reset le flag d'autorisation de la fonction de callback, et on va donc appeler la fonction « ShiftAllMatrixRow ».

```

174     // Animation de défilement du nom sur les Matrix
175     void ShiftAllMatrixRow(uint8_t _pMatrix[])
176     {
177         uint8_t i = 0;                      // Compteur pour le MSB des lignes de la première colonne
178         uint8_t j = 0;                      // Compteur pour tous les caractères que l'on affiche
179         uint8_t k = 0;                      // Compteur pour toutes les lignes des Matrix
180         uint8_t pMatrixBuffer[8];           // Buffeur des MSB des lignes de la première colonne
181
182         // Shift le texte seulement s'il est plus grand que la surface d'affichage
183         if(countCar > maxMatrix)
184     {
```

Figure 137 Initialisation et contrôle de la taille d'affichage suffisante dans la fonction d'animation (matrix.c)

Une fois entré dans la fonction et les initialisations effectuées, on va contrôler que l'on a pu afficher le prénom et le nom avec le point au complet sur la surface de LED des Matrix disponibles. En effet, car si on n'a pas assez de place pour tout afficher, c'est pour ça que l'on va faire défiler le texte. Ce n'est pas la taille exacte, mais on par du principe qu'on a un caractère par Matrix, ce qui évitera d'avoir un nom qui prend l'intégralité des Matrix. Cela pourrait porter à confusion, si on ne sait pas s'il y a réellement un caractère de plus après le nom de l'élève.

Puis si on peut faire l'animation, on va commencer par sauvegarder la première colonne toute à gauche de la première Matrix, donc les MSB des datas de la première Matrix.

```

185     // Récupère la valeur de tous les MSB des lignes de la première colonne
186     for(i = 0; i < ROW_MATRIX; i++)
187     {
188         // Récupération du MSB de la ligne actuelle
189         pMatrixBuffer[i] = RECUP_MSB(_pMatrix[0][i]);
190     }
```

Figure 138 Récupération de tous les MSB des lignes de la première colonne de la première Matrix connectée (matrix.c)

On les garde pour pouvoir à la fin les remettre du côté opposé pour quand le texte reviendra.

Puis on va faire l'animation sur l'intégralité du texte, tout en gardant la même méthode d'affichage par lignes.

```
192     // Pour tous les character que l'on affiche
193     for(j = 0; j < countCar - 1; j++)
194     {
195         // Pour toutes les lignes des Matrix
196         for(k = 0; k < ROW_MATRIX; k++)
197         {
```

Figure 139 Animation sur tout le texte et par méthode d'affichage par lignes (matrix.c)

Puis on va faire une animation de défilement vers la gauche, donc l'utilisateur pourra lire le nom normalement comme s'il le lisait sur une feuille.

```
198     // Décale de un vers la gauche toutes les lignes du buffer d'affichage
199     _pMatrix[j][k] = _pMatrix[j][k] << 1;
```

Figure 140 Décalage vers la gauche d'une LED tout le texte (matrix.c)

Ici on a donc simplement décalé vers la gauche d'un toutes les datas dans notre buffer d'affichage. Ce qui aura comme effet de décaler tout le texte d'une LED vers la gauche.

```
201     // Si on n'est pas à la dernière Matrix du buffer d'affichage
202     if(j < (countCar - 3))
203     {
204         // Récupère le MSB de la prochainne Matrix et le met dans le LSB de la Matrix actuel
205         _pMatrix[j][k] |= RECUP_MSB(_pMatrix[j + 1][k]);
206     }
```

Figure 141 Récupération du MSB de la matrix suivante et remplacement du LSB de la Matrix actuelle (matrix.c)

Puis si on n'est pas en train de traiter le dernier caractère de notre texte, on va récupérer à chaque fois le MSB qui est sur la Matrix d'après, pour le mettre dans le LSB de la Matrix actuelle.

Enfin si on arrive au dernier caractère, on va remplacer donc le LSB de la dernière Matrix par le MSB de la toute première Matrix que l'on avait récupéré au début de la fonction.

```
201     // Si on n'est pas à la dernière Matrix du buffer d'affichage
202     if(j < (countCar - 3))
203     {
204         // Récupère le MSB de la prochainne Matrix et le met dans le LSB de la Matrix actuel
205         _pMatrix[j][k] |= RECUP_MSB(_pMatrix[j + 1][k]);
206     }
207     // Si non on est à la dernière Matrix du buffer d'affichage
208     else
209     {
210         // Remplace le LSB de la dernière Matrix du buffer d'affichage dans le MSB de la Matrix actuel
211         _pMatrix[j][k] |= pMatrixBuffer[k];
212     }
```

Figure 142 Remplacement des LSB de la Matrix actuel avec les MSB de la première Matrix précédemment stockés (matrix.c)

Puis une fois le buffer d'affichage totalement décalé d'une LED vers la gauche, il nous reste plus qu'à réafficher tout le bufeur d'affichage sur les Matrix. Pour cela on va donc réutiliser la fonction « SendAllMatrixRow », qui a précédemment déjà été utilisée et expliquée.

```
216     // Envoie le buffer d'affichage sur les Matrix
217     SendAllMatrixRow(_pMatrix);
218 }
219 }
```

Figure 143 Affichage du nouveau bufffer d'affichage décalé d'une LED vers la gauche (matrix.c)

Puis le programme va reboucler à l'infinie, et donc faire défiler le texte à l'infini. Une fois le nom affiché et l'animation le programme reste donc bloqué dans l'animation, car il ne sert plus qu'à ça à ce moment-là. Pour une future amélioration du projet et ajout d'autres animations, ce comportement devra donc être légèrement modifié.

5. Software

5.1. Flowchart

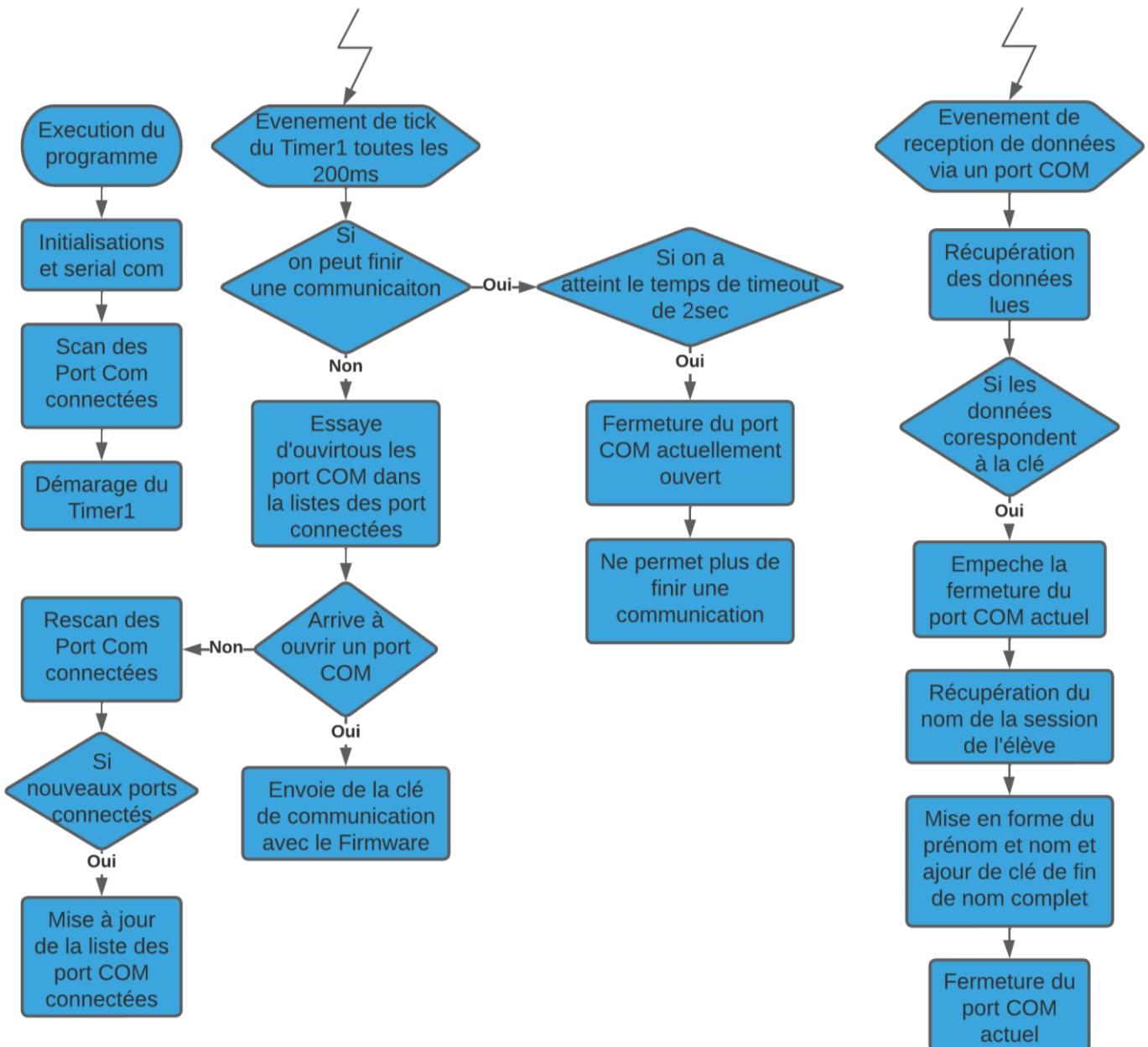


Figure 144 Flow chart du Software

5.2. Communication avec le Firmware

Afin de pouvoir détecter notre MainBoard 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, le nom de l'élève est ensuite envoyé 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.

La partie récupération du nom de la session a été déléguée au début du projet à un étudiant de quatrième année en CFC d'Informatique, M. Santiago Sugrañes Oria, en collaboration avec son supérieur technique, M. Alain Girardet.

5.2.1. Initialisations

Lors du démarrage du programme, on va commencer par initialiser plusieurs choses.

```

43     /// <summary>
44     /// Default constructor
45     /// </summary>
46     public View()
47     {
48         // Initialisation des components
49         InitializeComponent();
50         // Lecture des ports au démarage
51         canRetryOpen = RefreshSerialPort();
52         // Configure les paramètres de la communicaiton UART
53         InitSerialCom();
54         // Démarre le timer1 principalle qui fait tourner tout le programme
55         timer1.Start();
56     }

```

Figure 145 Initialisations au démarrage du Software (View.cs)

Après avoir fait les initialisations du système, on va commencer par scanner tous les ports COM qui sont connectés au démarrage, grâce à la méthode « RefreshSerialPort ».

```

103     // Récupéraiton les ports COM du system
104     private bool RefreshSerialPort()
105     {
106         // Récupération de la listes des port COM du system
107         portsNew = SerialPort.GetPortNames();

```

Figure 146 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.

```

109     // Si la nouvelle liste de ports COM est différente de la précédente
110     if (!Enumerable.SequenceEqual(portsNew, portsOld))
111     {
112         // Redimensionnement du tableau des ports COM à la taille du nouveau tableau COM
113         Array.Resize<string>(ref ports, portsNew.Length);
114         // Copie du tableau des nouveaux ports dans le tableau des ports de travail
115         portsNew.CopyTo(ports, 0);
116         // Redimensionnement du tableau des anciens ports COM à la taille du tableau de travail COM
117         Array.Resize<string>(ref portsOld, ports.Length);
118         // Copie du tableau des ports de travail dans le tableau des anciens ports COM
119         ports.CopyTo(portsOld, 0);
120         // Retourner une réponse vrai
121         return true;
122     }
123     // Si non on a la même liste de ports
124     else
125     {
126         // Retourner une réponse fausse
127         return false;
128     }
129 }
```

Figure 147 Sauvegarde dans le tableau de travaille uniquement si de nouveaux ports COM (View.cs)

Puis on devra configurer notre communication série UART passant par le port COM. Pour cela il faudra veiller à paramétrer les mêmes configurations entre le Software et le Firmware.

```

131     // Initialisaiton des paramètres UART du port COM
132     1 référence
133     private void InitSerialCom()
134     {
135         // Configuration du BaudRate
136         serialPort1.BaudRate = 9600;
137         // Pas de parité
138         serialPort1.Parity = Parity.None;
139         // Taille du message de 8 bits
140         serialPort1.DataBits = 8;
141         // Un seul bit de stop
142         serialPort1.StopBits = StopBits.One;
143         // Pas de Handshake
144         serialPort1.Handshake = Handshake.None;
145
146         // Set read timeouts
147         serialPort1.ReadTimeout = 500;
148         // Set write timeouts
149         serialPort1.WriteTimeout = 500;
    }
```

Figure 148 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.

```

54     // Démarre le timer1 principale qui fait tourner tout le programme
55     timer1.Start();
56 }
```

Figure 149 Démarrage du Timer1 (View.cs)

5.2.2. Événement Timer1

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

```

221     // Événement du Timer1 toutes les 200ms
222     1 référence
223     private void timer1_Tick(object sender, System.EventArgs e)
224     {
225         // Si on peut commencer le délai d'attente de la réception de la clé
226         if (canEndCom)
227         {
228             // Si on a attendu 2sec
229             if (counterTimerEndCom == 10)
230             {
231                 // Fini la communication avec le port COM actuel et le ferme
232                 EndTryCom();
233                 // Reset compteur d'attente de réponse
234                 counterTimerEndCom = 0;
235             }
236             // Incrémentation du compteur d'attente de réponse
237             counterTimerEndCom++;
238         }
239         // Si non on a pas une communication en cours et donc on peut essayer d'en commencer une
240         else
241         {
242             // Commence l'essay de communication avec tous les ports COM
243             StartTryCom();
244         }
245     }
246 }
```

Figure 150 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.

```

180     // Commence l'essay de communication avec tous les ports COM
181     1 référence
182     private void StartTryCom()
183     {
184         // Si on arrive à ouvrir le port COM et qu'on a le droit d'en ouvrir un
185         if (TryOpenCom() && canRetryOpen)
186         {
```

Figure 151 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.

```

200     // Si non on n'arrive pas à ouvrir les ports COM et on n'a pas le droit d'en ouvrir
201     else
202     {
203         // Récupéraiton des ports COM du system
204         canRetryOpen = RefreshSerialPort();
205     }
206 }
```

Figure 152 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.

```

151     // Essaye d'ouvrir un port COM
152     1 référence
153     private bool TryOpenCom()
154     {
155         // Teste tous les ports COM du tableau de travail des ports COM
156         for (int i = 0; i < ports.Length; i++)
157         {
158             // Essaye d'ouvrir le port COM actuelle
159             try
160             {
161                 // Récupération du nom du port COM actuel
162                 serialPort1.PortName = (string)ports[i];
163                 // Ouverture du port COM actuel
164                 serialPort1.Open();
165                 // Compteur du port COM sélectionné actuellement
166                 curentSelectedPort = i;
167                 // Retourner une réponse vrai
168                 return true;
169             }
170             // Si non s'il n'arrive pas
171             catch
172             {
173                 // Passe à la prochainne action
174                 continue;
175             }
176             // Retourner une réponse fausse
177         }
178     }

```

Figure 153 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 vrai.

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.

```

183     // Si on arrive à ouvrir le port COM et qu'on a le droit d'en ouvrir un
184     if (TryOpenCom() && canRetryOpen)
185     {
186         // Essaye d'envoyer la clé d'envoy pour s'annoncer au device voulu qui l'attend
187         try
188         {
189             // Envoi le la clé d'annonce sur le port COM ouvert actuellement
190             serialPort1.WriteLine(ANNOUNCE_KEY);
191         }
192         // Si non s'il n'arrive pas continué normallement l'exécution du programme
193         catch { }

```

Figure 154 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 de notre MainBoard.

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

Figure 155 Clé pour communiquer avec le Firmware

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

```

195     // Démare le délai d'attente de réception de la clé de retour sur le port COM actuel
196     canEndCom = true;
197     // Arrete d'essayer d'ouvrir un autre port COM
198     canRetryOpen = false;
199 }
```

Figure 156 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.

```

221     // Évenement du Timer1 toutes les 200ms
222     1 référence
223     private void timer1_Tick(object sender, System.EventArgs e)
224     {
225         // Si on peut commencer le délai d'attente de la réception de la clé
226         if (canEndCom)
227         {
228             // Si on a attendu 2sec
229             if (counterTimerEndCom == 10)
230             {
231                 // Fini la communicaiton avec le port COM actuel et le ferme
232                 EndTryCom();
233                 // Reset compteur d'attente de réponse
234                 counterTimerEndCom = 0;
235             }
236             // Incrémentation du compteur d'attente de réponse
237             counterTimerEndCom++;
238     }
```

Figure 157 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.

```

208     // Fini la communicaiton avec le port COM actuel et le ferme
209     1 référence
210     private void EndTryCom()
211     {
212         // Ferme le port COM actuel
213         serialPort1.Close();
214         // Efface le port COM actuel de la liste de travail des ports COM
215         ports[currentSelectedPort] = " ";
216         // Arrete la fermeture des ports et reprend le code normalment
217         canEndCom = false;
218         // Peut réessayer d'ouvrir un autre port
219     }
```

Figure 158 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.

5.2.3. Événement réception de données du Firmware

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

```
70 | // Évenement si on reçoi des données via la communication UART sur un port COM
  | 1 référence
71 |     private void serialPort1_DataReceived(object sender, SerialDataReceivedEventArgs e)
72 |
73 |     // Buffeur de lecture des données reçues
74 |     string dataRx = serialPort1.ReadExisting();
```

Figure 159 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.

```
27 |     private const string KEY = "C"; // Clé de retour attendue
```

Figure 161 Clé de réponse du Firmware (View.cs)

```
76 |     // Si les données reçues sont égales à la clé secrète
77 |     if (dataRx == KEY)
78 |     {
79 |         // Reste avec le port COM ouvert
80 |         canEndCom = false;
81 |         // Reset le compteur d'attente de réponse
82 |         counterTimerEndCom = 0;
83 |         // Envoie le nom de la session
84 |         SendMesageCom();
85 |     }
86 | }
```

Figure 160 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 le nom de la session à la carte.

Pour cela on va utiliser la méthode « SendMesageCom », mais avant ça on va voir rapidement comment est-ce que l'étudiant informaticien a pu récupérer le nom de la session.

Je récupérer le nom de la session dans une de ses méthodes qu'il utilise pour l'afficher sur la fenêtre qui apparaît sur le bureau.

```
62 |     internal void SetText(string userInfos)
63 |     {
64 |         // Récupération du nom de la session
65 |         userName = userInfos;
```

Figure 162 Récupération du nom de la session depuis la méthode « SetText » (View.cs)

Mais pour aller jusqu'au fond des choses, je suis allé chercher le code qui récupérait concrètement le nom de la session. C'est dans la classe « Model » que j'ai donc pu trouver la ligne qui a elle-même récupéré toute seule le nom de la session.

```
26 |     return UserPrincipal.Current.DisplayName;
```

Figure 163 Récupération du nom de la session (Model.cs)

Maintenant que l'on a le nom de la session, il ne nous reste plus qu'à le mettre en forme.

```

88     // Envoi du nom de la session par UART via le port COM
89     1 référence
90     private void SendMesageCom()
91     {
92         // Récupération du prénom et de la première lettre du nom de la session
93         string name = userName.Substring(0, userName.IndexOf(' ') + 2);

```

Figure 164 Récupération du prénom et de la première lettre du nom de la session de l'élève (View.cs)

Ici on commence par récupérer uniquement le prénom et la première lettre du nom de la session. Car les noms de notre école s'affichent « Ricardo Rodriguescrespo », donc il n'est pas possible de faire la différence entre les différents nom. C'est alors qu'en récupérant le nom de la même manière pour tout le monde, on obtient un résultat lisible « Ricardo R ».

Puis on va insérer la clé de fin de nom, pour que l'on puisse contrôler à la réception que le nom reçu est complet.

```

28     private const string END_NAM_KEY = "XDR";    // Clé de fin de nom
93     // Ajoute une clé de fin de nom pour s'assurer le l'envoy complet à la réception
94     name = name.Insert(name.Length, END_NAM_KEY);

```

Figure 166 Clé de fin de nom complet (View.cs)

Puis après de multiples tests, j'ai dû également convertir les lettres avec des accents, à son équivalent sans accent.

```

95     // Conversion de tous les potentiels accents en lettres normales
96     byte[] tempBytes = System.Text.Encoding.GetEncoding("ISO-8859-8").GetBytes(name);

```

Figure 167 Conversion du nom à son équivalent dans accents (View.cs)

Puis il ne nous reste plus qu'à envoyer le prénom et nom traité via le port série UART du port COM actuellement ouvert.

```

97     // Envoie du nom de la session correctement encodé
98     serialPort1.Write(System.Text.Encoding.UTF8.GetString(tempBytes));
99     // Arrête la communication et ferme le port COM actuel
100    EndTryCom();
101 }

```

Figure 168 Envoi du prénom et nom de l'élève via le port puis fermeture du port COM (View.cs)

Puis après l'envoi la communication est terminée et le port COM fermé.

Ensuite le programme suit son cours normalement, c'est-à-dire qui rescanne les nouveaux ports COM avec l'événement du Timer1.

Si on débranche la carte une fois un nom affiché, toute la procédure expliquée se refera, et cela à l'infini, ou en tout cas jusqu'à ce que l'on éteigne le PC.

6. Test et mesures

6.1. Fréquence d'interruption du Timer1

6.1.1. Matériel 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 |

6.1.2. Méthode de mesure

Il faudra connecter au moins une Matrix avec le jumeau à la MainBoard, puis la connecter à un PC via un câble USB.

Puis il faudra placer la sonde de l'oscilloscope comme indiqué sur le schéma de mesure.

Puis la mesure se fera en deux parties, la première il ne faudra pas lancer le Software afin que le Firmware reste bloqué dans l'attente d'un nom via l'UART.

Puis dans un deuxième temps il faudra lancer le Software afin que le nom récupéré s'affiche sur les Matrix, puis d'effectuer la mesure à ce moment-là.

6.1.3. Schéma de mesure

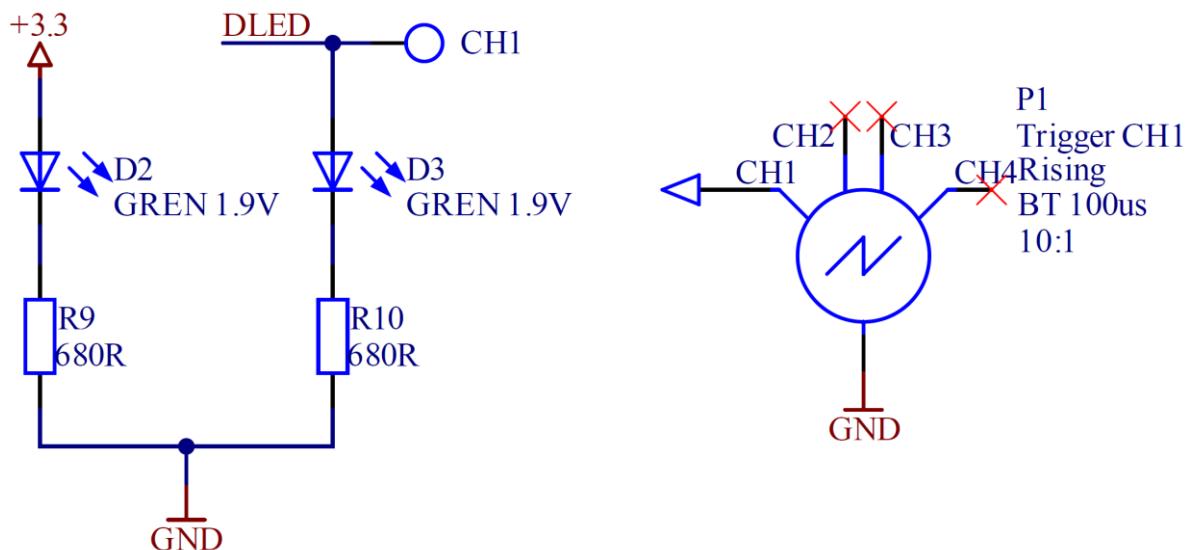


Figure 169 Schéma de mesure de la fréquence d'interruption du Timer1

6.1.4. Analyse de mesure

Afin de contrôler le bon fonctionnement de notre cycle d'interruption pensant l'exécution du programme, j'ai effectué deux mesures. La première lorsque l'on attend de recevoir un nom, puis la deuxième lorsque l'on en affiche un sur le Matrix.

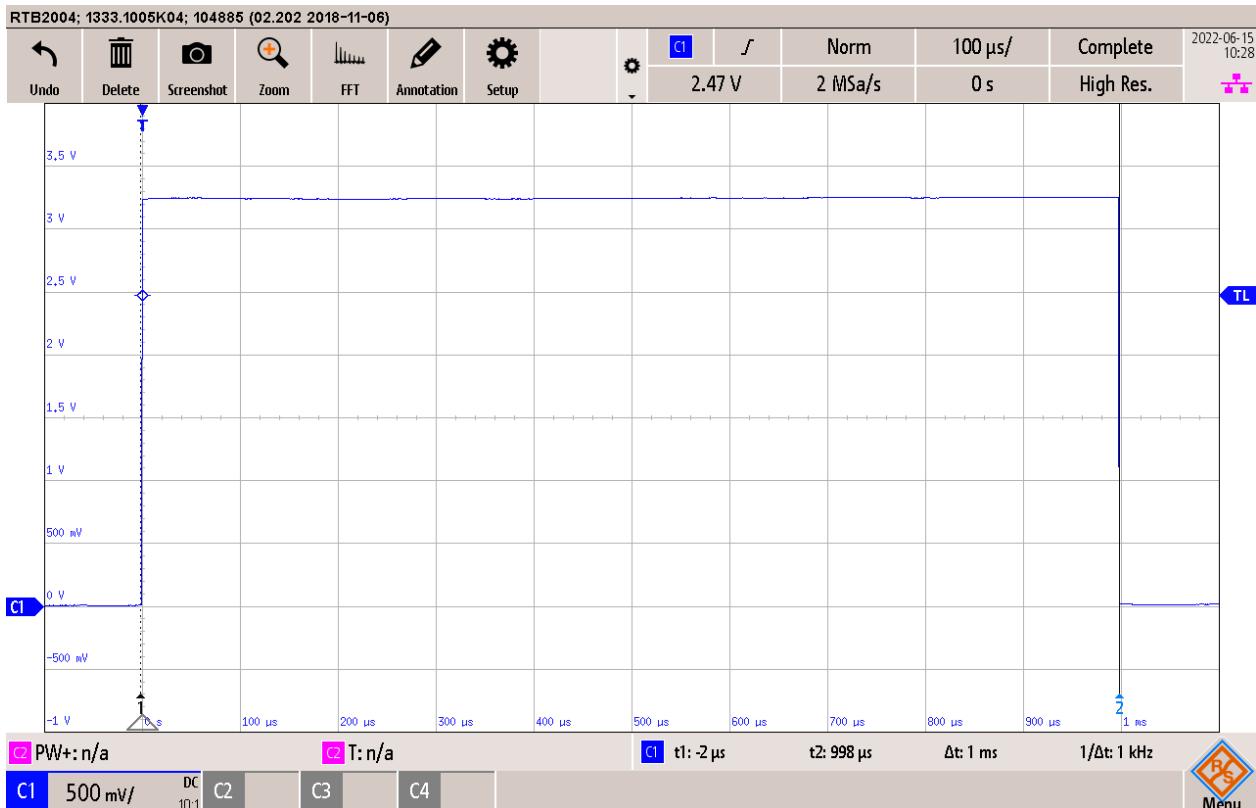


Figure 170 Période du Timer1 lors de l'affichage de l'animation du nom sur les Matrix

	Valeur Théorique [ms]	Valeur mesurée [ms]	Erreur absolue [ms]	Validation OK / NOK
T _{Timer1} pendant com UART	10	10	0	OK
T _{Timer1} pendant com SPI	10	10	0	OK

Lors de la phase d'affichage sur les Matrix, on obtient bien nos 10ms. Grâce à ces deux mesures, je peux confirmer que le système de Callback fonctionne correctement, et que le Firmware ne prend pas du retard au cours de son exécution, peu importe la phase.

Vous pourrez retrouver le détail des mesures en annexe.

6.2. Communication UART

6.2.1. Méthode de mesure

Il faudra connecter au moins une Matrix avec le jumpeur à la MainBoard, puis la connecter à un PC via un câble USB.

Puis il faudra placer la sonde de l'oscilloscope comme indiqué sur le schéma de mesure, et également configurer le décodeur UART avec les paramètres utilisés sur le Software.

Ensuite dans un premier temps pour les deux premières échanges entre le Software et le Firmware, puis vice versa, il faudra configurer le trigger du décodeur UART de l'oscilloscope sur le « Start Bit ».

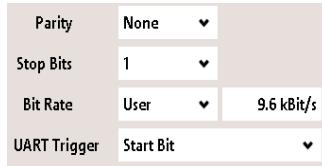


Figure 171 Paramétrage du décodeur de protocole UART sur l'oscilloscope avec le trigger « Start Bit »

Puis pour la capture du nom envoyé, il faut paramétriser le trigger sur la première lettre du prénom de la session sur laquelle la MainBoard est connectée. Dans mon cas c'est le caractère 'R', que je traduis depuis la table ASCII en hexadécimal par '0x52'.

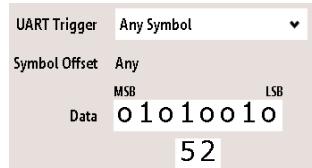


Figure 172 Paramétrage du décodeur de protocole UART sur l'oscilloscope avec le trigger « Any Symbol » sur '0x52'

6.2.2. Schéma de mesure

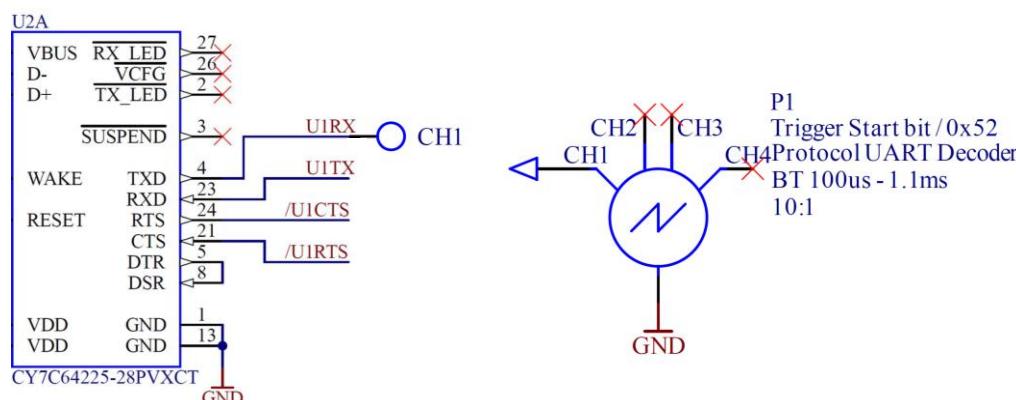


Figure 174 Schéma de mesure pour la communication UART du Software vers le Firmware

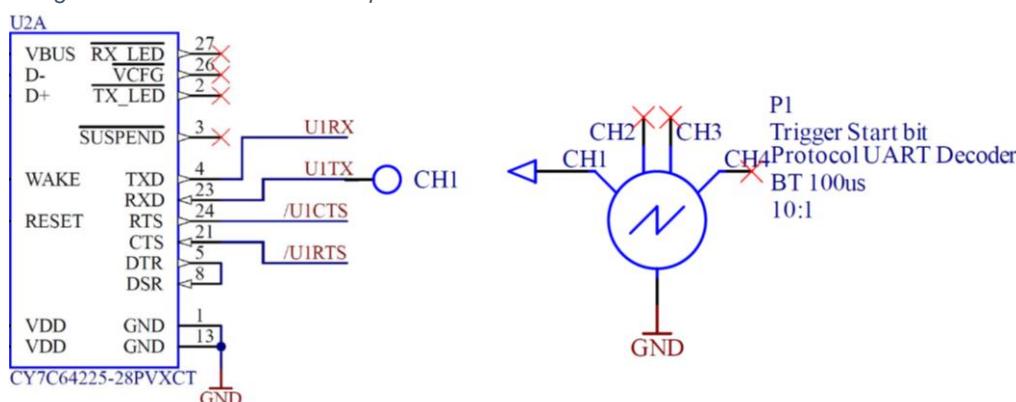


Figure 173 Schéma de mesure pour la communication UART du Firmware vers le Software

6.2.3. Analyse des mesures

6.2.3.1. Envoie de la clé du Software

Afin de pouvoir reconnaître que c'est bien notre carte que l'on a connectée au PC, le Software envoie une clé connue ('x'), à l'ouverture du port COM.

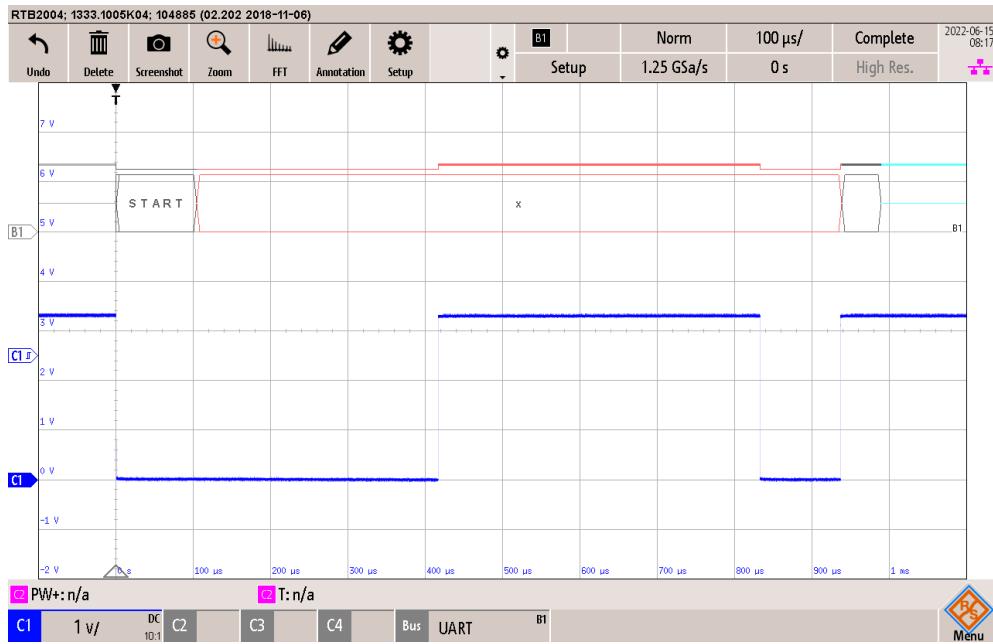


Figure 175 Mesure du caractère d'annoncement du Software vers le Firmware 'x'

Ici on peut voir que l'on reçoit correctement la clé 'x' envoyée par le Software, via le port COM en UART.

6.2.3.2. Envoi de la clé du Firmware

Puis afin de confirmer que l'on a bien reçu la clé du Software, le Firmware envoie à son tour sa clé ('C').

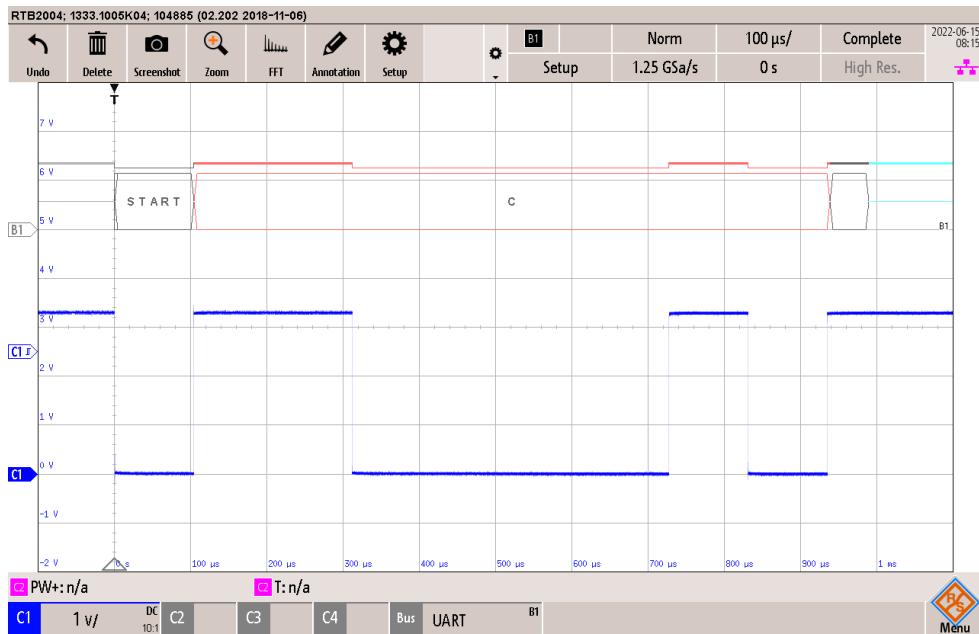


Figure 176 Envoi du caractère 'C' depuis le Firmware vers le Software

On peut bien voir que c'est la bonne clé 'C', qui est envoyée depuis notre carte.

6.2.3.3. Envoi du nom de l'élève

Puis une fois que le Software sait que c'est bien notre carte qui est connectée, il envoie alors le nom de la session sur la quel il est lancé.

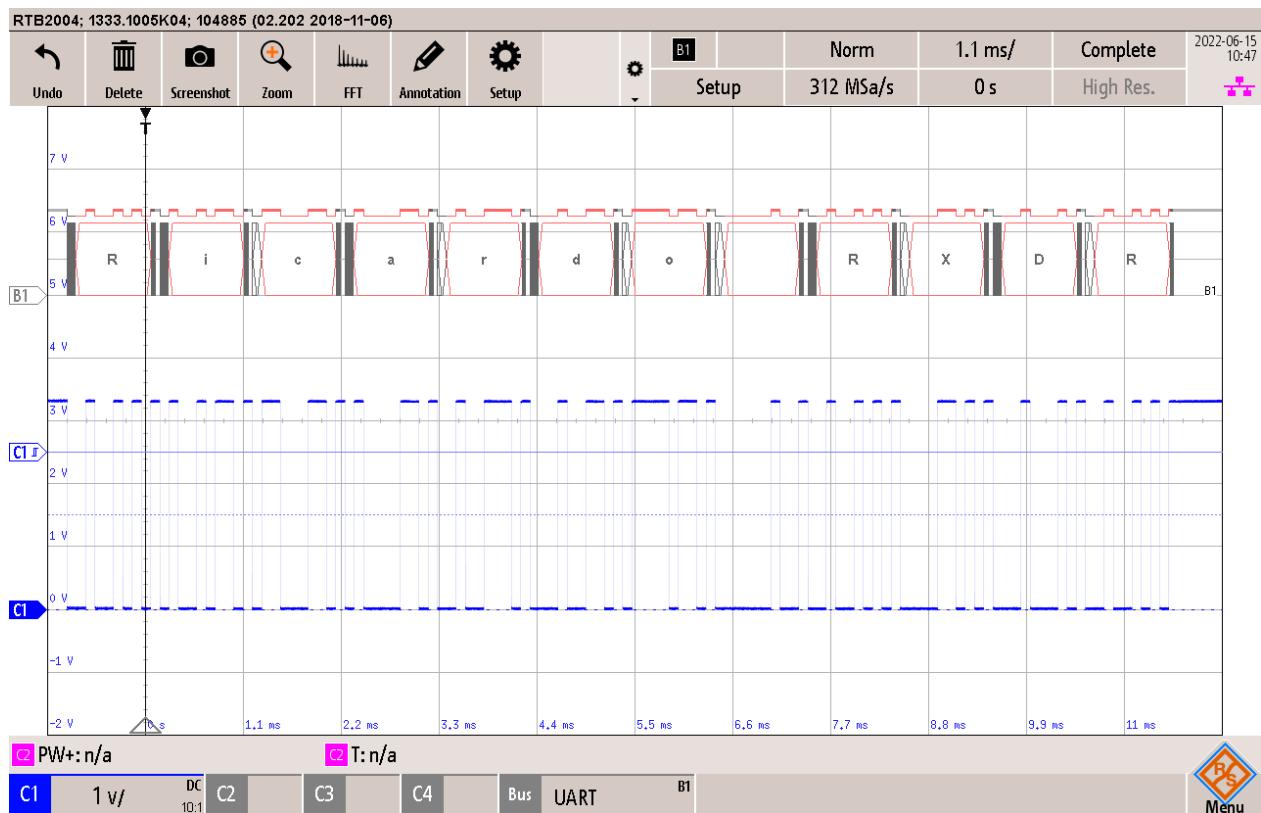


Figure 177 Réception du prénom, nom et clé de fin de nom complet 'Ricardo RXDR'

Ici on retrouve bien le prénom 'Ricardo ' suivi d'un espace, puis de la première lettre du nom, et pour finir la clé de fin de nom complet 'XDR'.

6.3. Communication SPI

6.3.1. Méthode de mesure

Afin de pouvoir reproduire exactement les mêmes mesures, il vous faudra connecter exactement trois Matrix avec le jumpeur à la MainBoard, puis la connecter à un PC via un câble USB.

Puis il faudra placer la sonde de l'oscilloscope comme indiqué sur le schéma de mesure, et également configurer le décodeur SPI avec les paramètres utilisés sur le firmware.

Pour la première mesure il vous faudra configurer le trigger du décodeur SPI sur la clé de reboulement '123', qui convertie en hexadécimal est '0x7B'.

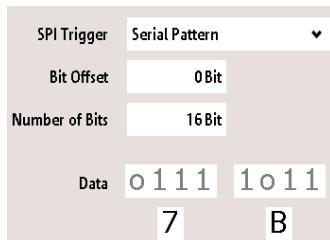


Figure 178 Configuration du trigger du décodeur SPI sur '0x7B'

Puis pour la deuxième et troisième mesure il vous faudra vous caler sur la première configuration du MAX7221 qui est '0x0B 0x07', afin de pouvoir toute la séquence d'initialisation.



Figure 179 Configuration du trigger du décodeur SPI sur '0x0B 0x07'

En suite pour la troisième mesure il vous faudra revenir sur le mode « Frame Start ».



Figure 180 Configuration du trigger du décodeur SPI sur « Frame Start »

Enfin pour la dernière mesure il vous faudra simplement suivre le schéma de mesure.

6.3.2. Schéma de mesure

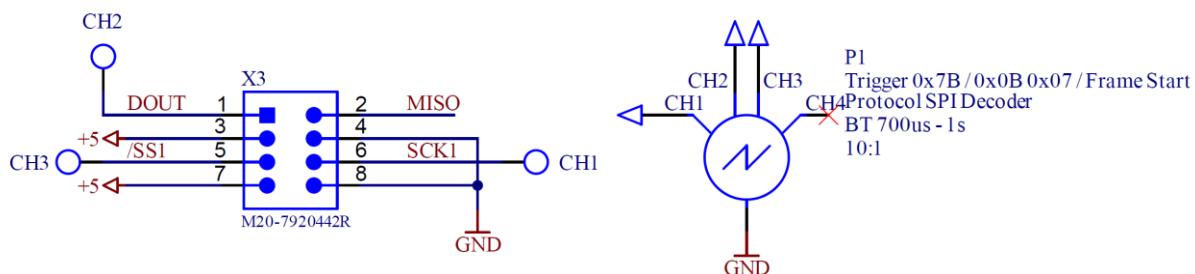


Figure 181 Schéma de mesure de la communication SPI

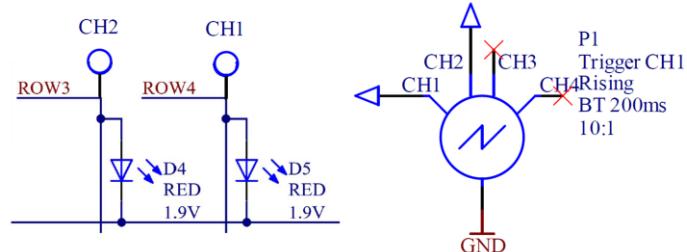


Figure 182 Schéma de mesure des timing des LEDs

6.3.3. Analyse des mesures

6.3.3.1. Détection des Matrix connectées

Lors de l'allumage de la MainBoard, la première chose qu'elle fait avec les Matrix, c'est de détecter le nombre de Matrix qui sont connectés.

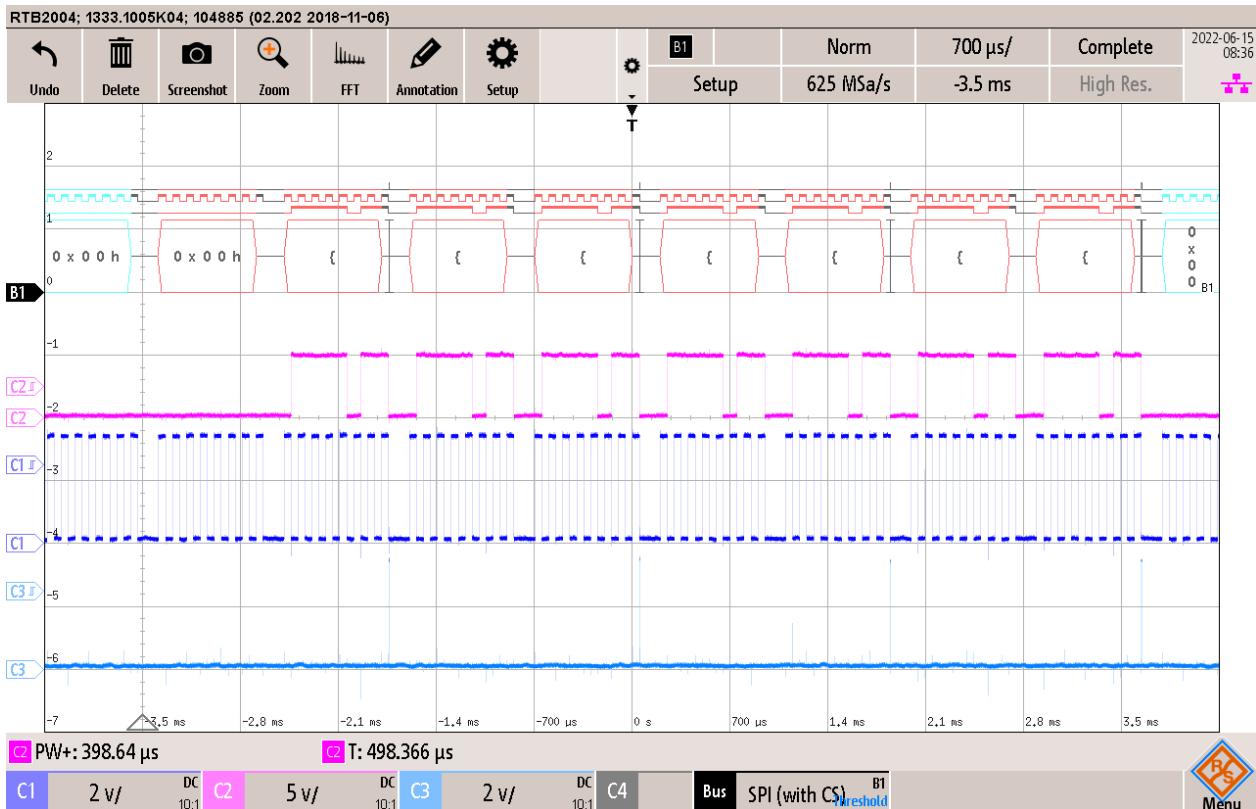


Figure 183 Trames de détection automatique du nombre de Matrix connectées à la MainBoar

Ici on retrouve bien notre vidage des registres, puis le remplissage avec les clés de reboulement '123', représenté ici par son caractère ASCII '{'. Puis une fois les clés arrivées de nouveau au microcontrôleur, l'envoi s'arrête.

On peut donc confirmer via la mesure le nombre de Matrix connectées.

$$NbrMatrixCo = \frac{NbrClé - 1}{2} = \frac{7 - 1}{2} = 3$$

Ici on retrouve donc bien les trois Matrix que j'ai connecté réellement à la MainBoard.

6.3.3.2. Initialisation

Puis une fois le nombre de Matrix trouvée, on peut les initialiser. Pour cela un certain nombre d'informations doivent être écrites sur les MAX7221.

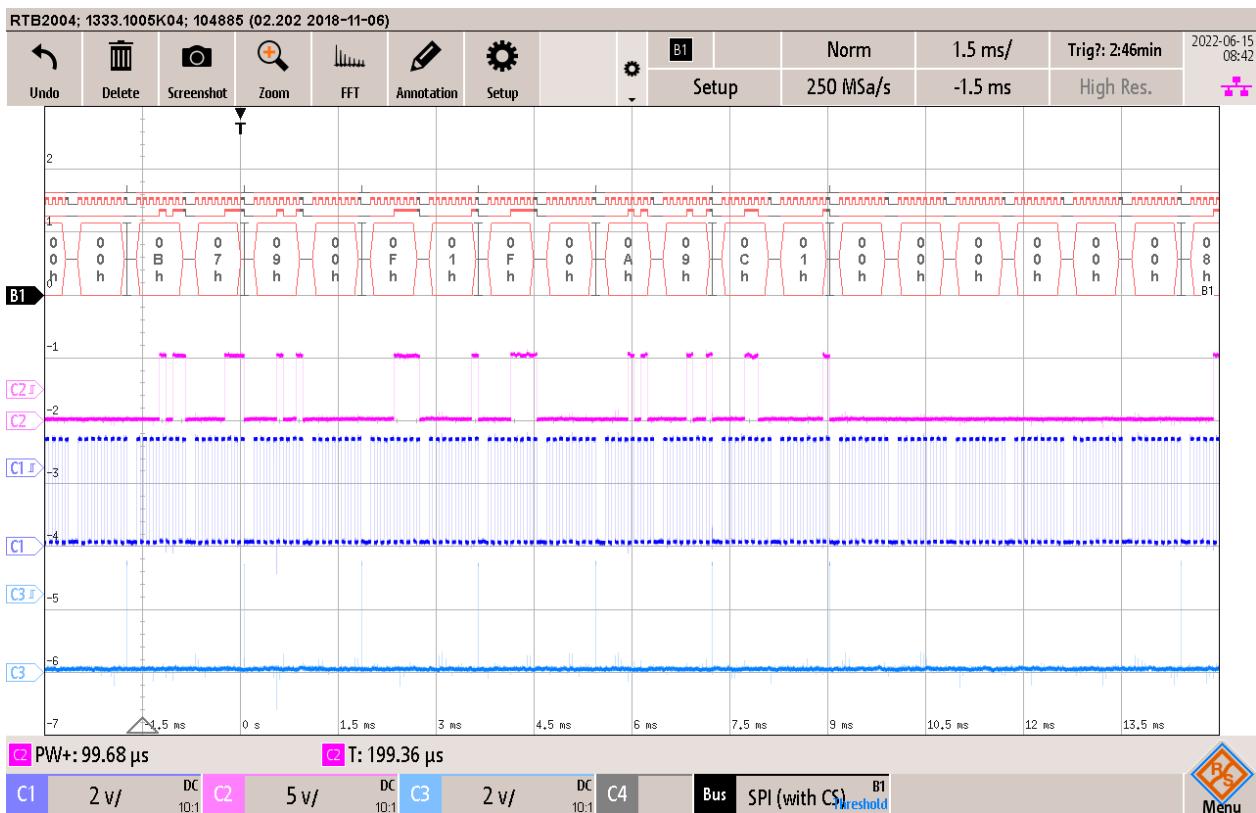


Figure 184 Trame de configuration des MAX7221 des Matrix

Ici on retrouve dans l'ordre toutes les configurations effectuées dans le Firmware, qui sont expliquées au points 4.4.

On retrouve donc bien dans l'ordre, le remplissage à vide des registres puis :

L'adresse du ScanLimit '0x0B' avec la valeur 0x07.

L'adresse du Decode Mode '0x09' avec la valeur 0x00.

L'adresse du Display test '0x0F' avec la valeur 0x01 dans un premier temps pour le test de l'allumage des LEDs, puis la valeur 0x00 pour revenir au fonctionnement normal.

L'adresse du Intesité '0x0A' avec la valeur 0x09.

L'adresse du ShutDown '0x0C' avec la valeur 0x01.

Pour finir, il y a de nouveau une salve de remplissage des registres à vide pour que la dernière Matrix reçoive bien toutes les configurations.

6.3.3.3. Affichage Nom

Une fois le tout configuré correctement, et un nom de session reçu, l'affichage statique s'effectue dans un premier temps.

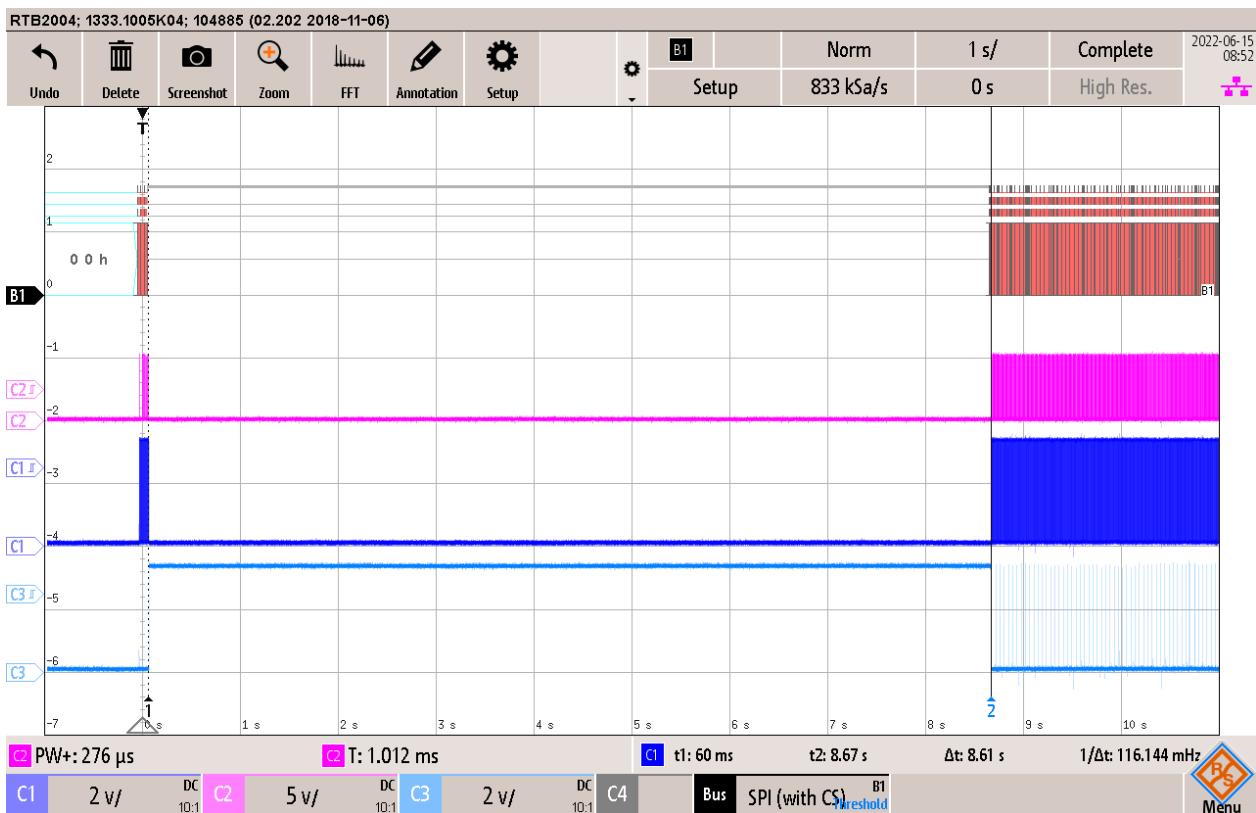


Figure 185 Trame du temps entre l'initialisation et l'affichage du nom sur les Matrix de 8.67sec

Ici on retrouve donc après une durée de 8.67sec l'affichage du nom sur le Matrix. Ce temps est dû au grand nombre de ports COMs connectés au PC de l'école.



Figure 186 Port COM connectés au PC avec la MainBoard connectée

Comme vous pouvez le voir ci-dessus, il y a trois ports IART connectés au PC de l'école. Le time out étant de deux secondes pour chaque port, on peut déjà compter 4 secondes de base.

Mais le test a été fait en Hot Plug, donc on peut partir du principe que l'on a quatre secondes pour les deux port COM déjà connectés. Puis lors de la connexion de la carte au PC, de nouveau quatre secondes pour les deux ports déjà listés. Donc on est déjà à huit secondes.

Puis les 610ms restantes comportent le temps que le software et le Firmware échangent leurs clés plus le temps d'envoi du nom, et également le temps du décodage et de la conversion du nom par le microcontrôleur pour qu'il soit affichable sur les Matrix.

Puis à ça il faut ajouter le temps que prend

6.3.3.4. Animation

Puis dans un second temps c'est l'animation du nom qui est faite, afin qu'il puisse être lu en entier.

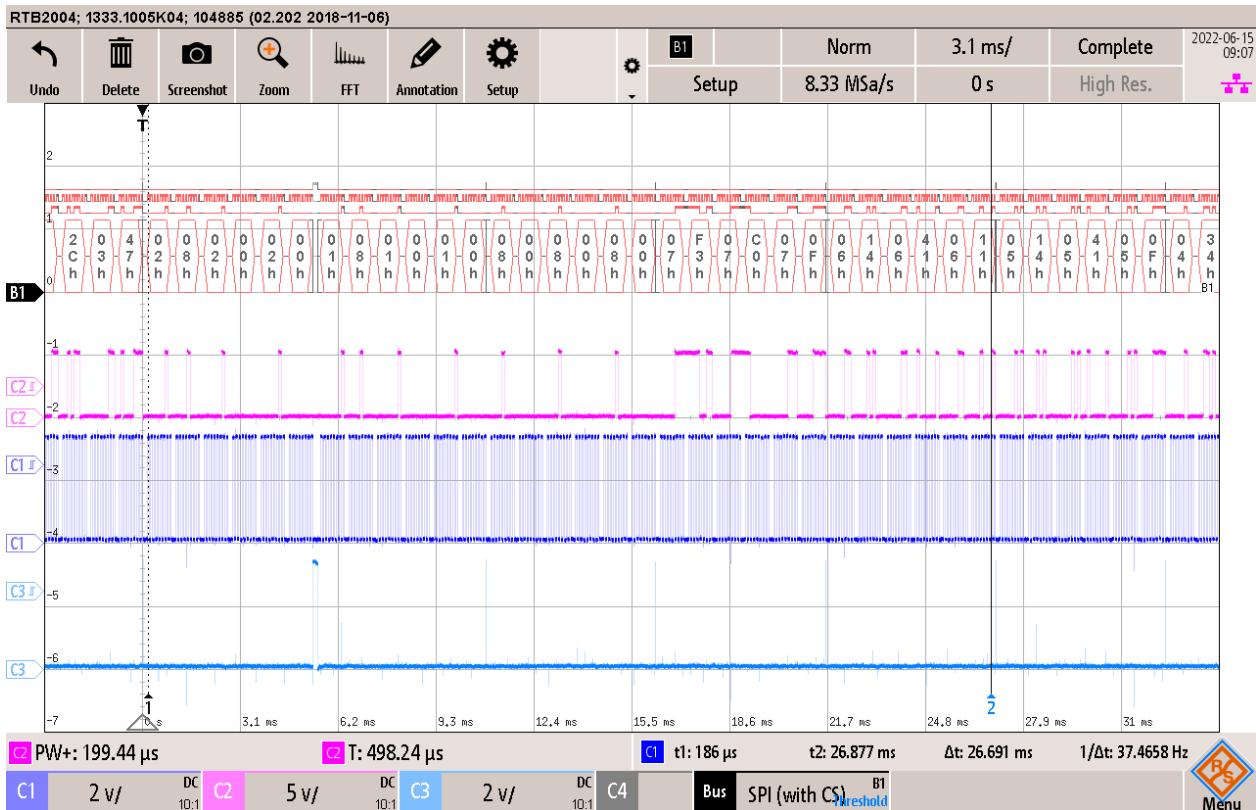


Figure 187 Trames UART de l'animation du nom sur les Matrix

Ici on peut retrouver les caractères qui sont envoyés en continu sur le Matrix.

6.3.3.5. Allumage LEDs

J'ai également voulu voir les signaux qui traversaient réellement les LEDs des Matrix.

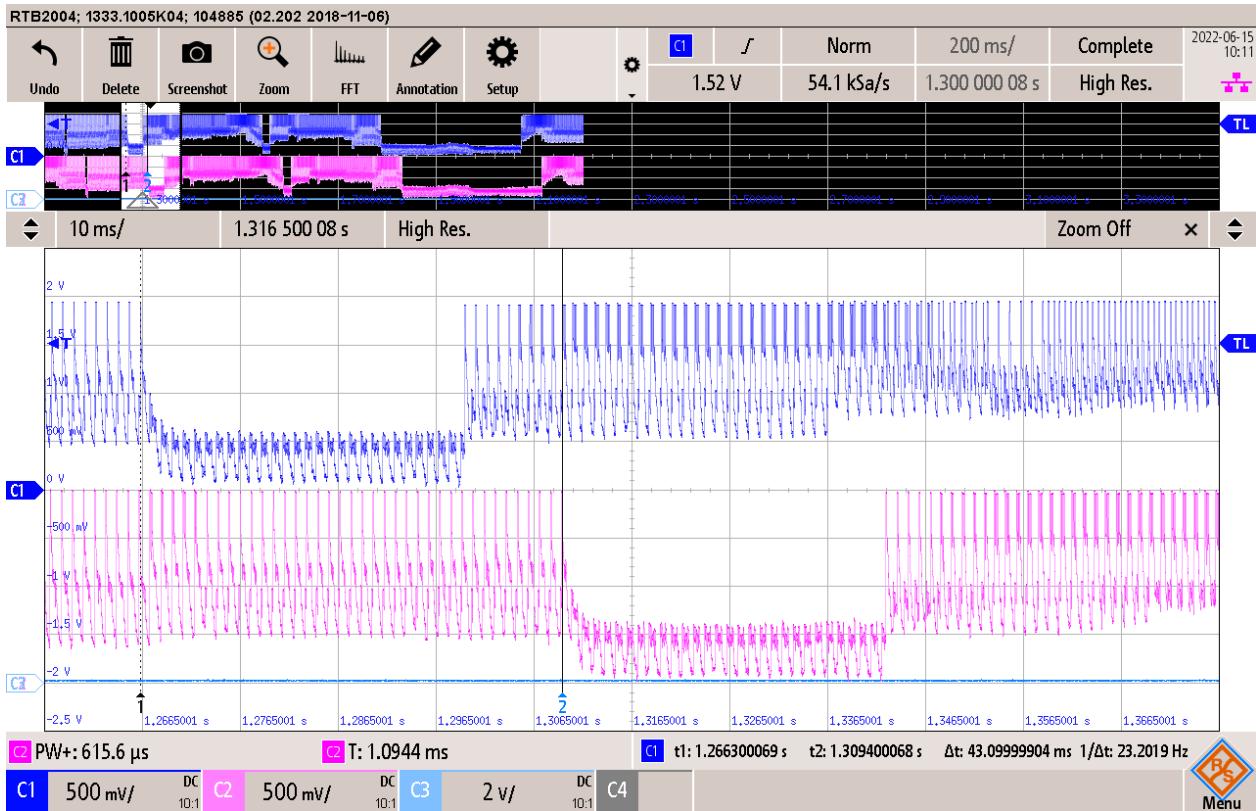


Figure 188 Niveaux électriques des deux LEDs D4 et D5

Malheureusement la mesure effectuée n'est pas exploitable, car on ne peut pas savoir quels états représentent quoi, et surtout quand.

Mais grâce à cette capture, on peut confirmer qu'il y a bien une sorte de PWM sur les LEDs, et que cela contribue à ce que tout le système consomme moins.

6.4. Consommation en courant

6.4.1. Méthode de mesure

Afin de pouvoir reproduire exactement les mêmes mesures, il vous faudra connecter exactement trois Matrix avec le jumpeur à la MainBoard. Puis il faudra cette fois-ci alimenter la carte MainBoar à l'aide d'une alimentation de laboratoire.

L'utilisation d'une version du Firmware altérée est également nécessaire. Les lignes nécessaires sont commentées dans l'initialisation de la machine d'état principale.

Puis il faudra placer l'Ampèremètre comme indiqué sur le schéma de mesure.

Puis dans un premier temps on va connecter uniquement la MainBoard.

Puis dans un second temps on va allumer toutes les LEDs des trois matrices grâce au mode « Dyplay Test » des MAX7221.

Enfin on finira par afficher un nom en dur dans le code pour le voir défiler avec l'animation, afin de pouvoir mesurer sa consommation en courant.

6.4.2. Schéma de mesure

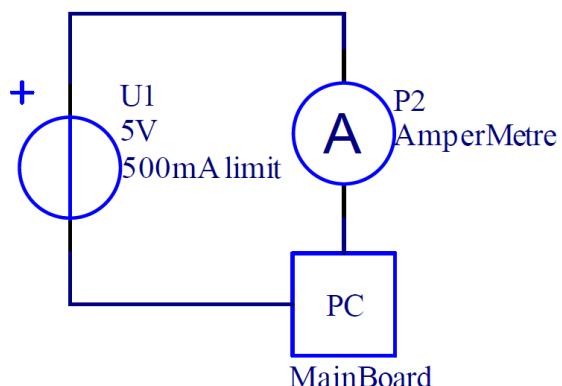


Figure 189 Schéma de mesure de la mesure de consommation de courant

6.4.3. Analyse des mesures

6.4.3.1. Sans Matrix connectées

Dans un premier temps j'ai fait la mesure de consommation en courant d'uniquement la carte MainBoar dans l'état d'affichage, mais sans Matrix de connectées.



Figure 190 Consommation de la MainBoard

Ici on a donc la consommation de tout l'étage d'alimentation, plus le convertisseur USB vers UART, et la consommation du microcontrôleur, pour un total de 36mA.

6.4.3.2. Toutes les LEDs allumées luminosité max

Puis la deuxième mesure a été de lancer une version du Firmware qui affiche uniquement le début du nom sans animation. Cela avec trois Matrix connectées, donc 192 LEDs en plus de la consommation de la MainBoard.



Figure 191 Consommation de la MainBoard plus trois Matrix avec toutes les LEDs allumées

Ici on peut donc calculer la consommation réelle de chaque LED allumée à l'intensité MAX.

$$\text{CourantLED} = \frac{\text{CourantTOT} - \text{CourantMainBoard}}{\text{nbrLED}} = \frac{292 * 10^{-3} - 36 * 10^{-3}}{192} = 1.3mA$$

On peut donc se rendre compte que les LED ne sont pas complètement à leur capacité maximale, car elles peuvent monter jusqu'à 2mA. Cela veut dire que si on recherche à faire l'affichage le plus puissant possible, il faudrait recalculer la résistance qui fixe le courant sur les Matrix, qui est connectée sur les MAX7221.

6.4.3.3. Animation du nom

Enfin la dernière mesure a été de tester la consommation en temps normal, avec un nom défilant avec l'animation de lancée, et une intensité réglée à 9.



Figure 193 Consommation min de trois Matrix avec animation Figure 192 Consommation max de trois Matrix avec animation

Si l'on soustrait la consommation de la MainBoard, on a donc une consommation entre 34mA et 45mA rien que pour que les Matrix affichent un nom qui défile. On pourra donc connecter au final plus de Matrix en une fois que prévue lors de la pré-étude.

7. État final et améliorations

7.1. Projet

Tous les points du cahier des charges ont été effectués, sauf la possibilité de sauvegarder le dernier nom affiché. Cela a été un choix de ma part, car en passant uniquement via la communication USB avec le PC lors ce qu'une session est logée nous assure que c'est bien la personne qui est devant le PC est bien celle qui est affichée. Surtout lors des phases de travail pratique où tout le monde change de place, les Matrix pourront donc afficher les noms des personnes qui sont réellement connectées aux PC. Si toute fois le souhait d'implémenter cette feature sur une version future, l'utilisation de la library « Mc32NVMUtil » est fortement conseillée, car très simple à mettre en place, et utilise de la mémoire du microcontrôleur.

7.2. Hardware

Tout l'Hardware qui a été monté fonctionne parfaitement. Toutes les modifications nécessaires sur les prochaines commandes ont déjà été faites.

Il y a uniquement les dimensions des Matrix qui pourraient éventuellement être modifiés dans une version C, si le but est d'avoir de nouveau des Matrix carrées.

7.3. Firmware

Le Firmware est complètement fonctionnel, et toutes les valeurs qui pourraient être menées à être modifiés se trouvent toutes dans le fichier de configuration « matrix.h ».

Une seule animation a été faite, il est maintenant possible d'en ajouter autant que vous le souhaitez. Il faudra simplement faire d'autres fonctions d'animation, et un peu d'imagination.

Afin de pouvoir afficher correctement tous les noms avec des accents, il faudrait convertir l'intégralité de la table ASCII en caractères affichables sur les Matrix. Puis de faire en sorte du coup que le software envoie également les noms avec les caractères spéciaux.

7.4. Software

Le Software est complètement fonctionnel, mais des ajustements de temps pourraient être faits, comme le raccourcissement des temps de time out de recherche des ports COM du PC. Pour le déploiement final, il faudrait désactiver l'aspect graphique de l'application.

7.5. Boitier

Un boitier a été commandé avec une taille suffisante pour accueillir deux Matrix. Mais aucune ouverture n'a été usinée. Toute fois un devis pour des tailles spécifiques a été demandé au fabricant, et vous le trouverez en annexes.

7.6. Test et Mesures

Il y a énormément plus de test qui peuvent être faits dans de très cas de figure différents. Par exemple le temps d'envoi d'un nom est considérablement plus rapide sur un PC portable qui ne possède pas de ports COM connectés par défaut. Ou encore une plus grande variété de nom et prénoms pourrait être testés, notamment avec des caractères spéciaux, et des machines différentes.

8. Conclusion

Lors de ce projet de semestre, j'ai pu mettre en application toutes mes connaissances en électronique emmagasinées lors de mon CFC et de ma première année de Technicien ES à l'œuvre.

En effet ce travail a touché tous les domaines d'activité d'un électronicien, tant bien le design et production de PCB, à la programmation de Firmware et Software, et jusqu'à la mesure pour les tests et validations.

Lors du choix de ce projet, je n'ai eu que deux lignes expliquant le projet, j'ai donc eu une énorme liberté pour la réalisation du cahier des charges. Cela m'a permis de faire des propositions, et d'imposer certaines contraintes.

Suite à la première phase de pré-étude j'ai pu faire un compte rendu de l'ampleur du projet, et donc faire valider par la même occasion le projet aux supérieurs techniques. C'est également lors de cette étape que j'ai pu faire la plupart de mes choix concernant les composants, les méthodes et les stratégies que j'ailais suivre pour le reste du projet.

Puis je suis passé à la phase de design, où il a fallu penser à réaliser des PCB qui puissent être montés à la main. Mais également veiller à ce que les espacements entre les LEDs soient correcte, et que l'on puisse interconnecter les Matrix et la MainBoard entre elles.

Ensuite une fois les cartes montées j'ai pu passer à la mise en service, dans la quel j'ai pu me rendre compte d'un certain nombre de modifications à apporter aux PCB, qui on toutes été mises à jour.

Puis je suis passé à la partie Firmware, qui a été la partie la plus grande du projet. En effet, pouvoir configurer et faire fonctionner tous les protocoles de communications, puis créer tous les algorithmes nécessaires au bon fonctionnement des deux cartes est ce qui a pris le plus de temps.

La dernière étape a été de faire la partie Software, qui pour le coup grâce à la collaboration avec l'élève en informatique, a pu être écourtée. En effet partant d'une base où j'avais déjà le nom de l'élève qui était récupéré. J'ai pu ensuite facilement implémenter mes algorithmes de communication entre le PC et la carte MainBoard.

Une fois toutes les étapes finalisées, j'ai pu avoir la satisfaction d'avoir un produit final fonctionnel. Maintenant il ne reste plus qu'à faire la production de masse, dont des commandes sont d'ores et déjà prévues, et d'autres déjà reçues.

L'utilisation d'un dépôt GIT afin de faire du versionning, a été utilisé pour la partie Firmware et également la partie Software.

Malgré le choix de s'être dirigé vers une version des Matrix d'une seule couleur, une version RBG sera faite par mes soins. Je prévois de la faire en quatre couches et intégrant trois MAX7221. De plus il ne faudra pas apport beaucoup de modification au Firmware, mais simplement tripler toutes les actions que l'on fait, pour les trois couleurs cette fois-ci.

Dans la finalité du projet, il est utilisable pour ce qu'il a été réfléchi et designé, donc comme tout system embarqué, il est dédié à une tâche, qui en l'occurrence est d'afficher le nom de l'élève connecté sur le PC.



9. Références

CY7C64225 - USB-to-UART Bridge Controller

https://www.infineon.com/dgdl/Infineon-CY7C64225_USB-to-UART_Bridge_Controller-Datasheet-v08_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ecc10346da

MAX1793 - Low-Dropout, Low IQ, 1A Linear Regulator

<https://datasheets.maximintegrated.com/en/ds/MAX1793.pdf>

Image connexion UART master to slave

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

Datasheet Quartz 8MHz CSM4Z-A2B3C3-60-8.0D18

<https://www.cardinalxtal.com/uploads/files/csm4-csm5.pdf>

Dimensionnement des condensateur du quartz

<https://microchipdeveloper.com/faq:937>

MPLAB® ICD 3In-Circuit DebuggerUser's GuideFor MPLAB X IDE

<https://ww1.microchip.com/downloads/en/DeviceDoc/50002081B.pdf>

Datasheet PIC32MX130F064B

<http://ww1.microchip.com/downloads/en/DeviceDoc/PIC32MX1XX2XX-28-36-44-PIN-DS60001168K.pdf>

Datasheet LED LS T67K-K1L2-1-0-2-R18-Z

https://www.osram.com/ecat/TOPLED%20LS%20T67K/com/en/class_pim_web_catalog_103489/prd_pim_device_2191024/

Datasheet MAX7221

<https://www.maximintegrated.com/en/products/power/display-power-control/MAX7221.html>

10. Annexes

- A. Cahier des charges
- B. Schéma électrique de la MainBoard
- C. Schéma électrique de la Matrix
- D. Liste des pièces et coûts
- E. Listings du Firmware
- F. Listings du Software
- G. Planning du projet
- H. Journal de travail
- I. Mode d'emploi
- J. Résumé
- K. Affiche du projet