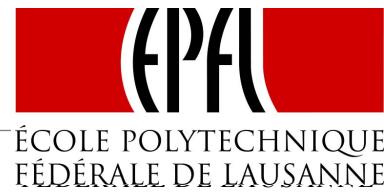
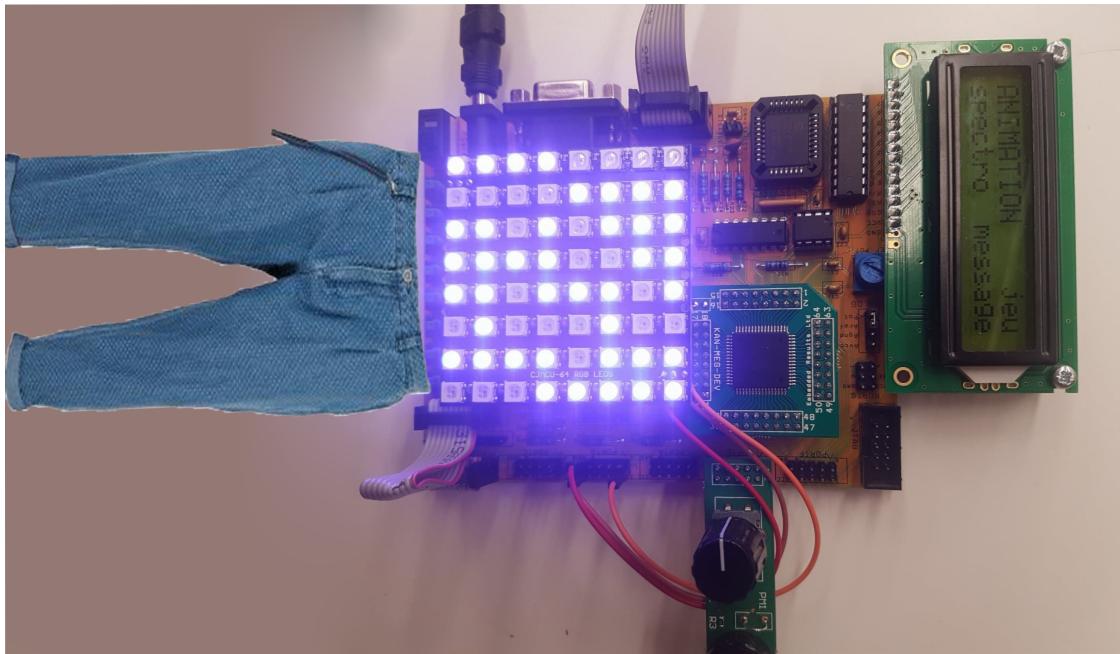


SÉCTION DE MICROMECHANIQUE



## Rapport de Projet de fin de Semestre

## Project Matrix



Auteurs	Niccolò Stefanini, Gauzelin Vidovic
Numéro de groupe	L14
Assistant responsable	Ivo Arabadzhiev
Enseignant	Alexandre Schmidt
Affiliation	Microtechnique
Semestre	BA4
Cours	Microcontrôleurs
Date	29.05.18
Signatures	

## Table de matières

<b>1. INTRODUCTION .....</b>	<b>3</b>
<b>2. SPÉCIFICATIONS &amp; MANUEL D'UTILISATION.....</b>	<b>3</b>
<b>2.1. Spécifications .....</b>	<b>3</b>
<b>2.2. Manuel d'utilisation .....</b>	<b>3</b>
2.2.1. Démarrage & Menu principal .....	3
2.2.2. Animation .....	4
2.2.3. Jeu .....	4
2.2.4. Spectromètre .....	4
2.2.5. Message .....	4
<b>3. FONCTIONNEMENT DU PROGRAMME .....</b>	<b>5</b>
<b>3.1. Fonctionnement général.....</b>	<b>5</b>
3.1.1. Driver matrice WS2818B.....	5
3.1.2. Structure inter-modulaire .....	6
<b>3.2. Fonctionnement animation .....</b>	<b>6</b>
<b>3.3. Fonctionnement jeu .....</b>	<b>6</b>
<b>3.4. Fonctionnement spectromètre .....</b>	<b>7</b>
<b>3.5. Fonctionnement message .....</b>	<b>8</b>
<b>4. CONCLUSION .....</b>	<b>8</b>
<b>4.1. Conclusion personnelle.....</b>	<b>8</b>
<b>4.2. Liste exhaustive des routines et macros implémentées .....</b>	<b>9</b>
4.2.1. Liste des routines .....	9
4.2.2. Liste des macros.....	10
<b>5. ANNEXES .....</b>	<b>10</b>

# 1. Introduction

*The Matrix* est un projet visant à imiter un téléphone portable basique. Quatre modes de fonctionnement ont été implémentés, à savoir un écran de veille (composé d'animations), un mini-jeu (reproduire une mélodie), un analyseur sonore (décomposition du spectre) et un service de messagerie (réception et affichage de texte).

L'absence d'une transmission vocale peut paraître surprenante, mais pourrait être facilement palliée. En effet, l'analyse du spectre sonore est déjà implémentée, le protocole de communication des données est déjà assuré par la messagerie et la génération d'un son à partir de différentes fréquences est réalisé par le jeu. La raison de son absence est que pour reconstituer le signal original, il faudrait pourvoir générer chacune des fréquences reçues simultanément. Dans le cas présent, la décomposition est faite selon 8 fréquences distinctes, ce qui correspondrait donc à 8 buzzers. La limitation est donc matérielle, étant donné qu'autant de périphériques ne peuvent pas être branchés sur les 6 ports de sortie de la carte. En outre, une décomposition en un nombre plus restreint de fréquences aurait reconstruit un son probablement très dénaturé.

# 2. Spécifications & Manuel d'utilisation

## 2.1. Spécifications

Le cahier des charges, visible en annexe, stipule les points clés suivants :

- menu principal affiché sur l'écran LCD et contrôlable par l'encodeur angulaire,
- 4 fonctionnalités (animation, jeu, spectromètre et messagerie) exploitant la matrice de LEDs,
- création d'un driver pour communiquer avec la matrice,
- réactivité assurée par un watchdog timer.

Les périphériques externes suivant sont utilisés :

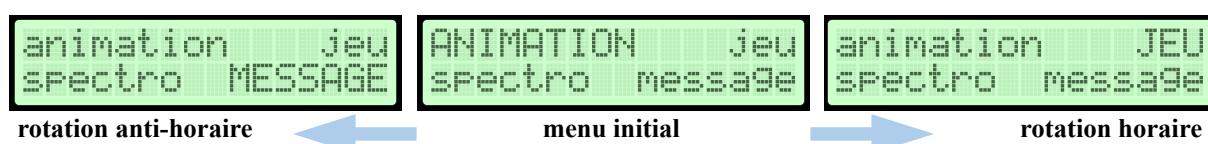
- (x1) écran LCD
- (x1) matrice LEDs RGB 8x8 (WS2818B)
- (x1) encodeur angulaire
- (x1) buzzer
- (x1) câble UART/USB

Suite à un problème de livraison, le microphone n'a pas été reçu suffisamment tôt pour permettre le développement du module spectromètre. Afin de ne pas abandonner entièrement le module, le microphone intégré à l'ordinateur lié via UART a été utilisé en substitut.

## 2.2. Manuel d'utilisation

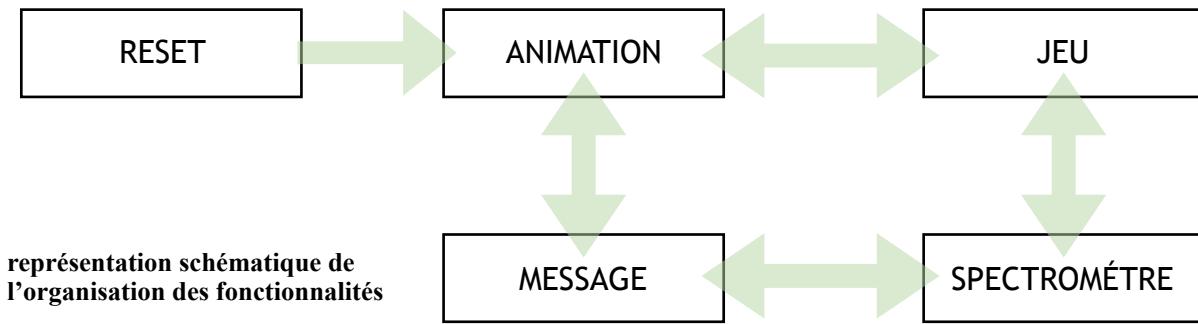
### 2.2.1. Démarrage & Menu principal

La carte s'allume en actionnant l'interrupteur principal, situé à gauche des boutons poussoirs. Une animation de démarrage s'affichera alors sur la matrice de LEDs. Par défaut, la carte démarre dans le mode animation. Le menu principal est visible sur l'écran LCD. Il se présente comme suit :



Le texte en majuscule désigne la modalité courante. Quant aux textes minuscules, ils représentent les autres possibilités. Le changement d'option s'effectue en tournant l'encodeur angulaire jusqu'à obtenir la fonction désirée. Une rotation dans le sens horaire sélectionnera la fonctionnalité directement à droite, alors que le sens anti-horaire permet de se déplacer vers la gauche. Les modes démarrent automatiquement dès qu'ils sont sélectionné par le curseur. Il est possible de réinitialiser un mode à tout moment en appuyant sur l'encodeur angulaire.

Il est important de noter que lors d'une longue période d'inactivité de la carte, elle redémarrera automatiquement et affichera les animations en guise d'écran de veille.



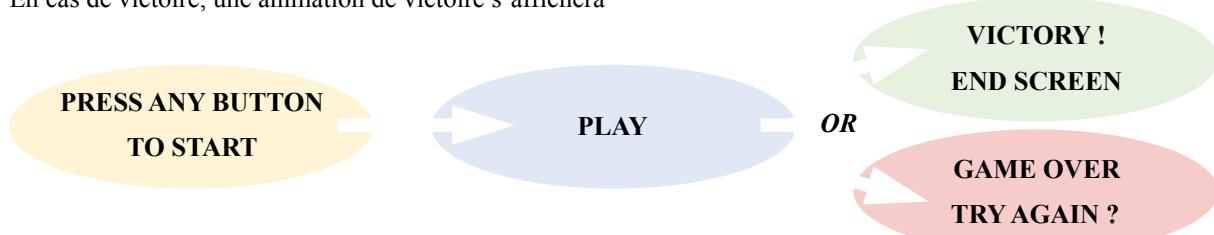
## 2.2.2. Animation

Le mode *animation* est le mode le plus simple. Ce mode ne requiert aucune interaction de l'utilisateur. Il se décompose en deux phases principales : Premièrement, le microcontrôleur affiche et lis la partition du thème principal de Tetris, puis la matrice affiche une boucle d'animation multicolore.

## 2.2.3. Jeu

Le mode *jeu* consiste en un test de rythme dont le but est de reproduire le thème principal de Tetris. Le jeu ne commence que lorsque l'utilisateur signale qu'il est prêt par l'appui de n'importe lequel des boutons pousoirs. La partition s'affiche alors sur la matrice puis se met à défiler. La note à jouer est toujours celle affichée sur la rangée de LEDs la plus basse de la matrice. À chaque note correspond un bouton. L'utilisateur doit jouer la bonne note durant tout le thème. Une tolérance de 4 erreurs a été implémentée pour faciliter le jeu. Une faute est détectée à chaque fois que l'utilisateur se trompe ou oublie une note. La couleur des LEDs indique en tout temps, par un dégradé allant du vert au rouge, le nombre de « vies » restantes au joueur. Si trop d'erreurs sont commises, la partition s'arrête de défiler pour que le joueur puisse analyser la séquence sur laquelle il s'est trompé. Lorsque l'utilisateur souhaite continuer, il lui suffit d'appuyer sur un bouton pour réinitialiser le jeu.

En cas de victoire, une animation de victoire s'affichera



## 2.2.4. Spectromètre

Le mode *spectromètre* (désigné par « *spectro* » sur le menu) effectue une analyse du spectre sonore sur différentes fréquences. Le signal à l'origine de la décomposition sera recueilli par le microphone de l'ordinateur connecté avec le câble UART/USB. Le résultat de la décomposition est quant à lui observable sur la matrice de LEDs. L'utilisateur peut donc s'amuser à parler, siffler ou jouer de la musique près du micro et observer le résultat.

## 2.2.5. Message

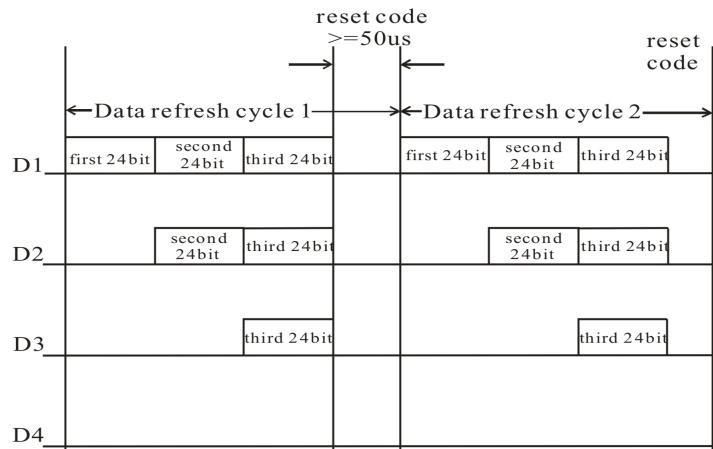
Le mode *message* est une messagerie permettant de communiquer de l'ordinateur vers le microcontrôleur (transmission unilatérale). L'utilisateur peut écrire des messages sur l'ordinateur à l'aide de RealTerm par exemple. Le texte apparaîtra alors en défilant de droite à gauche sur la matrice. À chaque nouveau message, la couleur de l'affichage sera modifiée.

### 3. Fonctionnement du programme

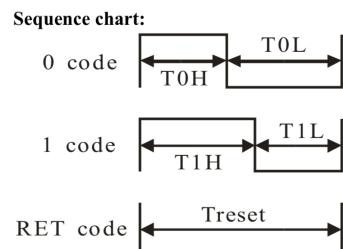
#### 3.1. Fonctionnement général

##### 3.1.1. Driver matrice WS2818B

La matrice utilisée pour réaliser l'affichage utilise le chip WS2818B. C'est un circuit qui permet de contrôler toutes les LEDs à l'aide de 3 câbles seulement : 2 lignes d'alimentation et une ligne de données. Le mode opératoire est le suivant : les 64 LEDs composant la matrice sont connectées en série. Le microcontrôleur transmet bit par bit un code RGB à la première LED, soit 24 données au total. Lorsque celle-ci a reçu les 3 bytes, elle arrête de retenir les informations envoyées et les transmet à la deuxième LED, qui va à son tour lire les 24 premiers bits lui étant adressés. Elle transmet alors le reste à la troisième et ainsi de suite. Les LEDs ayant reçu un code RGB complet continuent de transmettre les données tant qu'une pause de 50 µs ou plus n'est pas marquée.



WS2818B utilise un protocole de communication RZ à une seule ligne. La transmission étant asynchrone, le protocole prescrit l'écriture d'un 0 et d'un 1 comme suit :



**Data transfer time( TH+TL=1.25µs±150ns)**

TH	Code, high voltage time	TL	Code, low voltage time
T0H	0 code ,high voltage time	0.35us	±150ns
T1H	1 code ,high voltage time	0.9us	±150ns
T0L	0 code , low voltage time	0.9us	±150ns
T1L	1 code ,low voltage time	0.35us	±150ns
RES	low voltage time	Above 50µs	

Le facteur le plus limitant de la transmission est l'écriture de T0H ou T1L. En effet, l'envoie d'une impulsion peut être réalisée à l'aide de la macro OUTI ou des commandes sbi et cbi. Or, toutes les instructions mentionnées sont exécutées en deux cycles, soit 0,5 µs en opérant à 4MHz. Cette durée est la valeur seuil permise par le protocole, ce qui ne garantit pas une stabilité dans l'envoi des données. **La carte doit donc opérer à 8 MHz.** De plus les interruptions seront désactivées durant la communication, afin de ne pas parasiter les impulsions.

Les macros *SEND0* et *SEND1* du fichier *avrLED.asm* assurent respectivement la génération d'un 0 ou d'un 1. Elles sont appelées par *SEND\_RGB* qui traduit un code R,G ou B en série de bits. La routine *led\_rgb* regroupant l'envoi des trois bytes RGB est elle même appelée par *LINE\_OUT\_RGB*. L'intérêt de cette dernière macro vient de la structure utilisée pour mémoriser l'état des LEDs. En effet, les registres c0 à c3 et d0 à d3 sont exclusivement réservés à cet effet. Chacun des 64 bits de cet ensemble de registres agit comme un booléen qui décide si oui ou non une LED soit être allumée. Cela permet d'accéder et de modifier rapidement l'état de la matrice. Chaque registre contient 8 bits, soit l'équivalent d'une ligne de l'affichage. La macro *LINE\_OUT\_RGB* prend donc comme paramètre un registre et sort l'état de ses bits sur une des lignes de LEDs.

Finalement, *led\_display\_rgb* affiche les 8 registres à la suite sur la matrice, ce qui correspond à un affichage entier. La routine *led\_reverse\_display* est très similaire dans sa fonction. Elle inverse simplement la signification des booléens allumé/éteint de chaque LED. Cela est dû à la forme de la lookup table utilisée pour représenter les caractères ASCII. Il est intéressant de mentionner la macro *SHIFT\_DIS* qui effectue une rotation des registres c0 à d3, ce qui permet d'obtenir une impression « glissante » à l'affichage.

Pour ce qui est des nuances de couleurs, toutes les LEDs d'un affichage donné utilisent les mêmes bytes RGB afin de limiter le nombre de données à mémoriser. Ces bytes sont stockés dans les registres r2, r4 et r5.

### 3.1.2. Structure inter-modulaire

Un changement de fonctionnalité pouvant survenir n'importe quand, la pile n'est pas garantie d'être vide au démarrage d'un mode. Toutefois, de par la nature indépendante des divers modules, il n'y a pas de transfert d'information à assurer sous une quelconque forme. De plus, toutes les fonctions sont volatiles, c'est-à-dire qu'elles ne gardent rien en mémoire entre deux utilisations. Le pointeur de pile est donc réinitialisé lors de chaque transition. Cela permet d'éviter que des données fantômes restent à la souche de la pile sans jamais être purgées, ce qui conduirait à une diminution graduelle de la capacité de la pile.

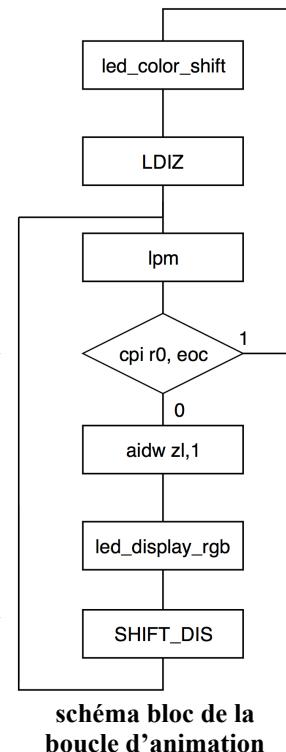
Les branchements entre les différents modules sont effectués par des interruptions liées à l'encodeur angulaire. Ces interruptions, par leur nature, rebranchent toujours au début d'un nouveau module. Elles n'ont donc pas besoin de mémoriser l'état des registres au début de leur appel, et n'ont pas besoin non plus de registre dédié. **Les registres \_u et \_w sont donc utilisés au même titre que les autres registres dans les divers fonctionnalités.** De plus, les routines d'interruption ne se terminent pas par reti, les interruptions doivent donc être réactivées manuellement au début des différents modes. Il est important de noter que tous les modules sont des boucles infinies. Il est donc impossible d'en sortir autrement qu'avec une interruption.

### 3.2. Fonctionnement animation

Le mode animation réutilise en partie la bibliothèque *sound.asm* donnée par le polycopié de cours. Elle a cependant été modifiée afin de supporter la possibilité de jouer des blancs dans la mélodie, sans que ceux-ci ne soient considérés comme étant des *end of file*. Ceci est réalisé simplement en rajoutant une comparaison entre la note chargée depuis la lookup table et la constante *no sound* (abrégé ns) suivie d'un branchement vers la routine *sound\_off* si nécessaire.

Pour ce qui est de la partition, elle est encodée dans les notes également. En effet, toutes les lookup tables sont mémorisées dans la mémoire programme, or seul le pointeur z est capable d'y accéder. Utiliser deux tables simultanément est donc peu ergonomique, puisqu'il faudrait charger alternativement z avec deux adresses différentes. La solution adoptée à la place consiste à coder un chiffre entre 0 et 7 dans les trois derniers bits de chaque note, pour correspondre à l'indice de la LED à allumer. Par exemple, la note la plus aigüe qui sera jouée dans la mélodie est un la2. Elle sera placée tout à droite de l'affichage, elle aura donc l'indice 0. Elle correspond donc à une constante de la forme 0bxxxxx000. De manière analogue, la note la plus grave, la1, a la forme 0bxxxxx111. Le masque des LEDs à afficher peut s'extraire en calculant  $2^{\text{code}}$ . Cette solution a toutefois l'inconvénient de dénaturer les notes émises par le buzzer. Afin de limiter la dissonance, les notes sont définies deux fois trop grandes, puis sont divisées par deux avant d'être jouées. Cela est possible car les notes ont toutes en commun un bit de poids fort à zéro. Grâce à cette astuce, un des trois bits de poids fort est déplacé dans le carry avant que la note soit jouée, et l'erreur sur la fréquence est toujours inférieure à 1,2 %.

À la fin de la mélodie, une seconde animation bouclant indéfiniment apparaît. Celle-ci est générée à l'aide d'une lookup table contenant une suite de masques. Ces masques sont affichés consécutivement sur la matrice. Lorsque le microcontrôleur détecte la fin de la table, il rebranche au début et recommence. Entre deux boucles, les valeurs de registres RGB sont modifiées cycliquement afin d'obtenir un rendu multicolore.



### 3.3. Fonctionnement jeu

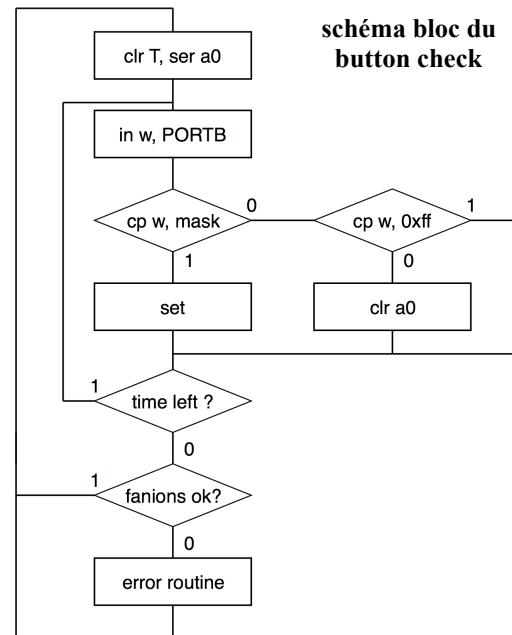
Le fonctionnement du jeu est extrêmement similaire à celui décrit dans l'animation. Les principales différences étant le bpm, la reconnaissance des boutons, et la gestion des couleurs de l'affichage. Le bpm est réglé par une constante dans le code et est donc très facile à changer. Il est réduit d'un facteur 2 pour faciliter le jeu.

La reconnaissance des boutons est une routine qui remplace la boucle d'attente nécessaire à la gestion du son. Durant cette routine, les boutons sont continuellement comparés au masque du pattern attendu. Il est intéressant de noter que le masque en question est le même que celui utilisé pour l'affichage sur la matrice. Si les boutons sont identiques au masque, le fanion T est set. Sinon, l'entrée est soumise à un second test : elle est comparée à

la position neutre, c'est-à-dire au scénario où aucun bouton n'est appuyé. Si les deux éléments ne sont pas égaux, le fanion a0 est clear. Ce test est indispensable puisqu'il serait irréaliste d'attendre du joueur de presser le bon bouton pendant la durée précise de chacune des notes et de le relâcher exactement à la fin de la note, sans déborder sur le blanc présent entre chaque notes. À la fin de la génération d'une note, les deux fanions sont vérifiés. Si au moins un des fanions est à 0, la routine d'erreur est appelée. Sinon, le jeu se poursuit normalement. Les fanions sont alors réinitialisés au début de la lecture de la note suivante. Le fanion T est clear et a0 est set.

La couleur est utilisée pour indiquer au joueur son nombre d'erreur. Pour ce faire, elle est initialisée au vert en début de partie, puis à chaque fois que l'on rentre dans la routine *jeu\_error* le byte responsable du vert décroît alors que son homologue pour le rouge croît. Cela se traduit par une modification de la couleur de la partition vers le rouge.

Finalement, si lorsque le pointeur détecte le *end of file* l'utilisateur a encore des vies, alors le programme rentre dans une routine d'animation de victoire.



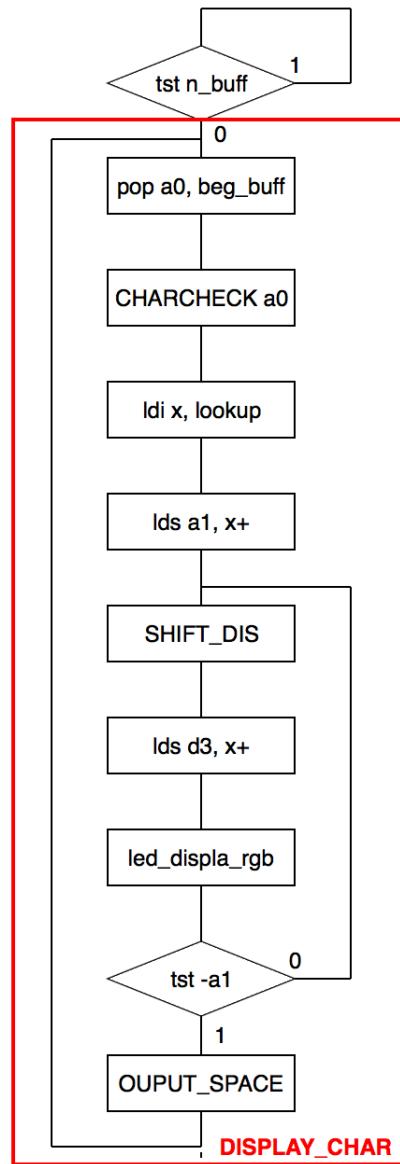
### 3.4. Fonctionnement spectromètre

Le spectromètre devait initialement utiliser un microphone connecté en périphérique externe, et calculer les transformées de Fourier sur le microcontrôleur directement. Cependant, en raison d'un problème de livraison, le microphone n'est pas arrivé dans des délais permettant le bon développement de la modalité. Afin de ne pas abandonner complètement la fonctionnalité, le microphone intégré dans l'ordinateur a été utilisé pour pallier à ce manque. La décomposition de Fourier en fréquence est donc calculée grâce à un script Python s'exécutant sur l'ordinateur, et les données finales sont transmises à la carte. Pour des raisons de sécurité, la communication UART de cette modalité est toujours initiée par l'envoi d'un caractère begin, un espace, suivi des 8 bits correspondants aux intensités des 8 fréquences affichées. L'initialisation de la communication par un caractère spécifique permet d'assurer le bon alignement des huit valeurs sur les huit colonnes, et ainsi éviter que d'éventuels restes présents dans le buffer ne nuisent à l'affichage.

Le programme python prend des échantillons de données toutes les 80 ms qui sont analysés en temps réel sur 4096 fréquences possibles. Ces valeurs sont par la suite réduites à huit intervalles, puis, grâce à une échelle logarithmique, les résultats sont réduits dans l'intervalle 0...8. La chaîne de caractères est donc composée avec un espace au début. Enfin elle est envoyée par UART avec un baud rate de 4800 Hz.

Le microcontrôleur gère les transmissions de la manière suivante : lorsqu'une donnée transmise est reçue par UART, une interruption est déclenchée. Cette interruption lit le caractère envoyé et le pousse en entrée d'un buffer circulaire (first in, first out). Pendant ce temps, le programme principal patiente en attendant de détecter un nombre non nul d'éléments dans le buffer. Lorsque cette condition est remplie, il prends alors le premier caractère du buffer et vérifie que ce soit un espace, auquel cas il poursuit son exécution. Les 8 prochaines valeurs sont alors analysées et affichées. Une fois ceci fait, le microcontrôleur retourne dans la boucle d'attente jusqu'à l'arrivée de l'échantillon suivant.

Les transmissions ont lieu environ toutes les 100 ms. Le taux de rafraî-



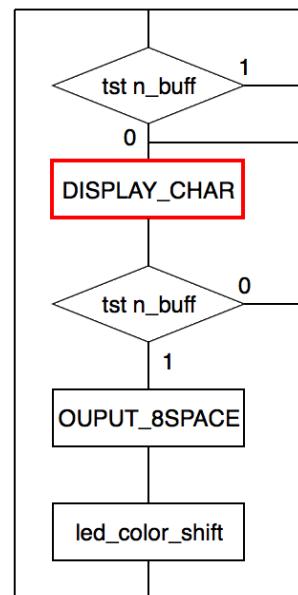
**schéma bloc de l'affichage d'un caractère**

chissement sera donc de 10 scènes par seconde ce qui est assez rapide pour un défilement continu correspondant au son ambiant en temps réel.

### 3.5. Fonctionnement message

Le principe de base de la messagerie est similaire à celui du spectromètre, étant donné qu'elle reçoit également ses données de l'ordinateur via UART. La réception des messages dans le buffer d'entrée repose sur le même principe que celui décrit dans la section précédente. Le programme repose par défaut dans une boucle infinie, en attendant de détecter des données à afficher dans le buffer. Dans ce cas, le premier caractère sera lu et comparé à divers codes ASCII pour comprendre de quel symbole il s'agit. Une fois le caractère retrouvé, sa représentation sur la matrice de LEDs est extraite d'une table stockée en mémoire. Les colonnes de LEDs sont alors copiées en sortie une à une, avec un délai entre chaque mise à jour de l'affichage permettant d'obtenir une illusion de texte « glissant ». En plus de contenir les patrons de LEDs à afficher pour chaque caractère, la lookup table contient la place qu'occupe chaque symbole en nombre de colonnes. Cela permet de savoir si l'affichage d'une donnée est terminé ou non. Une colonne de LEDs éteintes permet de démarquer les caractères consécutifs sur l'affichage.

Lorsque le microcontrôleur détecte que le buffer est vide à nouveau, c'est-à-dire à la fin de l'affichage du texte, 8 colonnes supplémentaires de LEDs éteintes sont ajoutée afin de faire sortir tout les caractères de la matrice. En outre, les couleurs de l'affichage sont modifiées pour ajouter une touche de fantaisie.



**schéma bloc de l'affichage d'un texte**

## 4. Conclusion

### 4.1. Conclusion personnelle

Ce projet réalise un système multitâche modulaire, profitant de la communication avec l'ordinateur et du display LCD. Il se caractérise par plusieurs interruptions et requiert une application quasi-totale du cours de microcontrôleurs. De plus le codage du driver pour la matrice de LEDs WS2818 nous a poussé vers des nouvelles directions, nous obligeant à changer la fréquence d'oscillation naturelle de la carte pour communiquer à une fréquence adéquate aux LEDs.

Un point important a été la création d'un script python analysant le spectre sonore et appliquant un algorithme de FFT sur les échantillons audio reçus en temps réel à travers du microphone. Cette tâche s'est avérée très complexe, étant donné que nous sommes deux étudiants sans background solide ni en python ni en traitement de signaux, mais ceci nous a permis de beaucoup apprendre.

Nous avons cherché à optimiser les structures d'accès en mémoire pour les rendre plus rapides et performantes, ceci est le cas du stockage et montre totalement la puissance de l'assembleur. Un contrôle précis à la taille du bit serait très compliqués même avec des langages de bas niveau comme le C.

La choix d'inclure une matrice RGB permet d'un coté d'avoir à disposition plus d'un milliard de combinaisons de couleurs possibles. Malheureusement nous n'avons pas pu profiter de tout le spectre: 255 est la valeur maximale théorique pour chacun des bytes RGB. Or un simple 20 est déjà très fort et un 100 devient éblouissant si l'observateur est trop proche. Nous avons conçu des fonctions pour changer la couleur de LED pendant le développements de modalités, ce qui a ajouté un petit coté artistique très appréciable au projet. Nous aurions quand même pu plus travailler sur le défilement des couleurs. FastLED, la librairie qui gère la communication avec les modules WS2818 notamment pour Arduino, implémente un deuxième type de choix de gestion pour la couleur des LEDs basé sur la tonalité, la saturation et la valeur de luminosité (HSV). Ceci permet de changer d'une couleur à l'autre en gardant la même luminosité très simplement. Nous avons cherché à s'inspirer de ce principe mais notre résultat reste quand même inférieure.

Next step: <https://youtu.be/HJ6zy2oVxqk>

## 4.2. Liste exhaustive des routines et macros implémentées

### 4.2.1. Liste des routines

ROUTINE	IN	OUT	DESCRIPTION
led_rgb	r2,r4,r5 = RGB	-	allume une LED avec le code RGB fourni
led_display_rgb	d3...d0,c3...c0 = on/off LEDs r2,r4,r5 = RGB	-	initialise tout l'affichage selon les informations contenues dans d3...c0. Un niveau 0 est interprété comme une LED allumée et inversement. Une LED allumé sera de la couleur spécifiée par les bytes RGB
led_reverse_display	d3...d0,c3...c0 = on/off LEDs r2,r4,r5 = RGB	-	initialise tout l'affichage selon les informations contenues dans d3...c0. Un niveau 1 est interprété comme une LED allumée et inversement. Une LED allumé sera de la couleur spécifiée par les bytes RGB
led_color_shift	r2,r4,r5 = RGB	r2,r4,r5 = RGB	modifie cycliquement les bytes RGB
led_clear_all	-	-	éteint toutes les LEDs
mode_anim	-	-	initialise tout le mode animation
mode_jeu	-	-	initialise tout le mode jeu
mode_spectro	-	-	initialise tout le mode spectromètre
mode_message	-	-	initialise tout le mode messagerie
reset	-	-	réinitialise la carte
int_4	-	-	routine d'interruption de changement de mode
int_5	-	-	routine d'interruption de changement de mode
int_6	-	-	routine d'interruption de réinitialisation de mode
uart_rxc	UDR0	-	routine d'interruption de la communication UART
anim_stand_by	-	-	joue une animation
jeu_error	r2,r4,r5 = RGB	r2,r4,r5 = RGB	traite une erreur du joueur durant le jeu
jeu_over	-	-	réinitialise le jeu si le joueur perd
button_analysis	PIND, a0, T	a0, T	vérifie si le joueur appuie les bons boutons

#### 4.2.2. Liste des macros

MACROS	IN	OUT	DESCRIPTION
SEND0	-	-	envoie une impulsion 0 à la matrice
SEND1	-	-	envoie une impulsion 1 à la matrice
SEND_RGB	@0	@0	envoie une impulsion équivalente au LSB de @0
LINE_OUT_RGB	@0	@0	règle 8 LEDs selon les 8 bits LSB...MSB de @0
LINE_OUT_REV	@0	@0	règle 8 LEDs selon les 8 bits MSB...LSB de @0
LED_CLEAR	-	-	éteint une LED
SHIFT_DIS	d3...d0,c3...c0	d3...d0,c3...c0	décale tous les registres mémorisant l'affichage
LCDPUTS	@0		choisit quelle version du menu @0 afficher
LDMASK	@1	@0, @1	charge @0 avec 1<<@1
JEU_INIT	-	@0	charge le masque de la prochaine note à jouer dans @0
CB_INIT	-	-	initialise le tampon circulaire
CB_PUSH	@0	-	push @0 dans le tampon circulaire
CB_POP	-	@0	pop une donnée du tampon circulaire dans @0
MES_CHRCHECK	@0	-	transforme un caractère ASCII en un offset pour la lookup table
MES_OUTPUT_SPACE	-	-	sort une colonne de LEDs blanches sur la matrice afin de marquer un espace

### 5. Annexes