



Projet de semestre

2225 Vumètre Fréquences

Auteur·ice : Maëlle Clerc
Professeur·e·s : Serge Castoldi, Juan José Moreno, Philippe Bovey
Date : 22 juin 2023

Table des matières

Introduction	5
Cahier des charges	6
But du projet	6
Spécifications du projet	6
Tâches à réaliser	7
Jalons principaux	7
Livrables	7
Convention de nommage et liens	8
Stockage du fichier	8
Pré-étude	9
Filtrage numérique	9
Schéma bloc général du système	11
Schéma bloc hardware	11
Architecture firmware	13
Choix des technologies clés du système	14
Microcontrôleur	14
Ampli de traitement	15
Alimentation	15
Évaluation des coûts	15
Planning	15
Conclusion et perspectives	16
Design	17
Changements suite à la pré-étude	17
Schéma	18
Choix des composants	19
Microcontrôleur et clock	19
Partie alimentation et découplage	20
Référence de tension	21
Connectiques d'entrées et sorties audio	22
Connectique avec la matrice à Leds	23
Reset et connecteur de programmation	24
Amplification du signal audio	25
BOM intermédiaire	25
Conclusion et perspectives	26
Montage et programmation	27
Montage	27
Connecteur 5V	27
Modification de la partie amplification	28
Programmation	29
Pin Settings	29
Configuration des timers	31
Configuration de l'ADC	32
Débogage	33
Abandon de la transformée de Fourier	34

Filtrage numérique	34
Problème de lecture de l'ADC	36
Mode d'emploi	37
État du projet	38
Conclusion	39
Webographie	40
Datasheetographie	40
Journal de travail	41
Schéma	44
Annexes	45
Code Octave GNU (pré-étude)	45
Listing de app.c	46
Listing de system_interrupt.c	49
Listing de SampleFilter.c	51
Listing de SamplFilter.h	52
Fichier MOD	52
Résumé du projet	56
Affiche du projet	58

Table des figures

1	Vidéo Youtube à l'origine du projet	5
2	Premier schéma bloc	7
3	Résultats du script Octave GNU	10
4	Schéma bloc du système	11
5	Schéma bloc hardware	12
6	Architecture firmware	13
7	Routine d'interruption	14
8	Planning	15
9	Schéma complet	18
10	Schéma du quartz	19
12	Schéma tension de référence	21
13	Schéma connectique Audio	22
14	Schéma connecteur pour la matrice à Leds	23
15	Schéma reset et connecteur de programmation	24
16	Schéma Ampli Audio	25
17	PCB monté	27
18	Signal en sortie de l'Ampli	28
19	Schéma Ampli Audio avant modification	28
20	Signal de sortie de l'Ampli après modifications	29
21	Pin Settings partie 1	30
22	Pin Settings partie 2	31
23	Configuration des timers	32
24	Configuration de l'ADC	33
25	Affichage du problème lors du debuggage	33
26	Interface de TFilter	34
27	Contrôles de l'interface de TFilter	35
28	Code généré par TFilter	35
29	Code généré par TFilter	36

Introduction

Dans le cadre de mes études à l'école supérieure de l'ETML pour devenir technicienne en électronique, il nous est demandé de réaliser un projet sur une durée d'un semestre complet.

Le projet qui m'a été attribué est la réalisation d'un vumètre du spectre fréquentiel. Cette idée vient de M. Bovey, inspiré par une vidéo trouvée sur YouTube (le lien de cette vidéo est en page 40).



FIGURE 1 – Vidéo Youtube à l'origine du projet

La version numérique du présent document comporte bon nombre de liens hypertextes pour en faciliter la lecture, ainsi tous les énoncés de la table des matières sont cliquables pour amener directement aux pages correspondantes, toutes les notions de page dans le texte le sont également, et enfin, la mention de table des matières en haut de chaque page permet de revenir à cette dernière.

Cahier des charges

Vumètre Fréquences 2225

Projet de semestre

Entreprise/Client :	ETML-ES	Département :	SLO
Demandé par (Prénom, Nom) :	Philippe Bovey	Date :	16.11.2022

Auteur (ETML-ES) :	Maëlle Clerc	Filière	SLO
		Date :	23.11.2022

But du projet

Il s'agit de designer un vumètre pour différentes gammes de fréquence, un analyseur de spectre visuel de 20 à 20kHz (filtrage pour les gammes suivantes : 20 - 50 - 100 - 200 - 500 - 1k - 2k - 10k - 20k) avec visualisation de l'amplitude. Le microcontrôleur est à choix selon ce qui conviendra le mieux. L'entrée du système viendra d'un système audio grand public, ou d'un générateur de signal. L'interface visuelle est à choix entre des leds (éventuellement reprendre le projet de matrice à leds 2126), une interface en C#, voire une communication USB.

Spécifications du projet

Le système doit afficher visuellement l'amplitude de chaque fréquence de la gamme citée précédemment venant d'un signal audio fourni en entrée. Le traitement du spectre sonore, par filtrage, sera fait numériquement par le microcontrôleur.

L'idée est de chaîner ce système entre la source et l'ampli audio, donc il faut traiter le signal en vue de le filtrer par le microcontrôleur, mais aussi laisser passer le signal à travers le système sans le modifier (true bypass).

Sont à faire, mais laissés libres, le choix du microcontrôleur, l'affichage, leds ou en C# (la matrice à leds du projet 2126 est à envisager), le design du boîtier ainsi que ses dimensions.

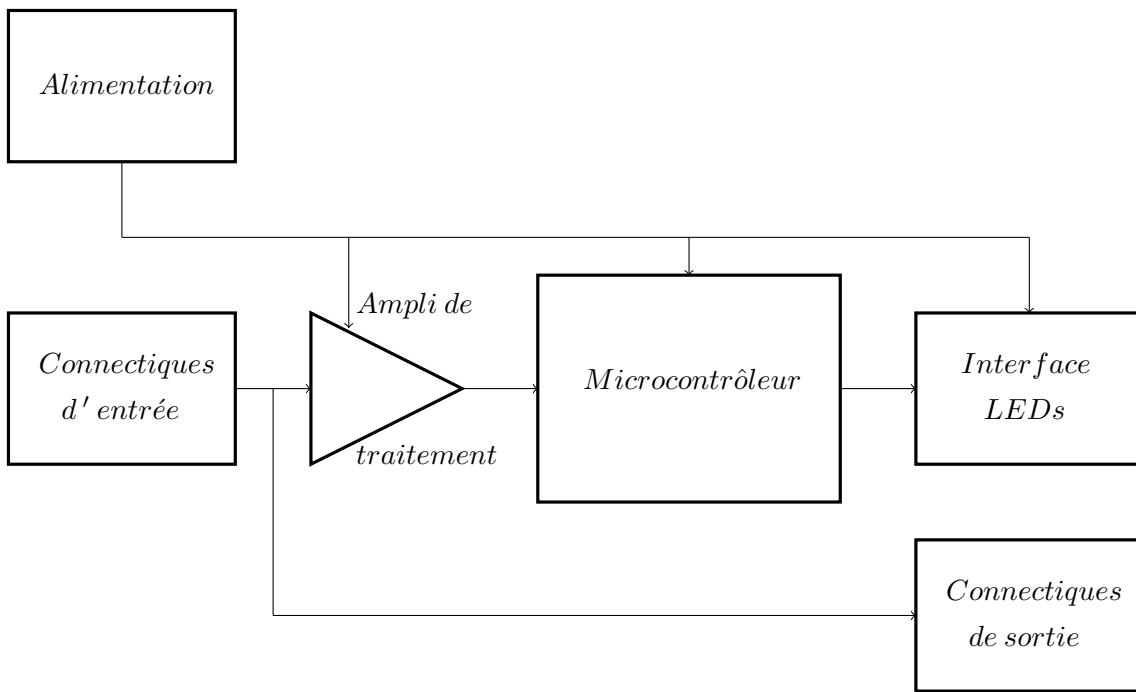


FIGURE 2 – Premier schéma bloc

Tâches à réaliser

1. Choisir le bon microcontrôleur pour ce projet, selon les entrées analogiques, la fréquence d'échantillonnage disponible, la qualité du DAC.
2. Designer l'ampli de traitement audio, peut-être qu'un simple ampli OP sera suffisant.
3. Choisir une interface Leds, voir si le projet 2126 serait bon.
4. Choisir les connectiques d'entrées et sorties.
5. Designer le circuit d'alimentation.
6. Designer le tout, en un schéma.
7. Designer et rooter le PCB, avec la BOM, et faire la commande.
8. Programmer le microcontrôleur.
9. Montage, tests et validation.

Jalons principaux

1. 07.12.2022 Rendu du rapport de pré-étude.
2. 25.01.2023 Rendu du rapport de design.
3. 22.03.2023 Rendu des fichiers de fabrication.
4. 14.06.2023 Fin du montage et de la programmation, et rendu du rapport final.

Livrables

1. Les fichiers sources de CAO électronique des PCB réalisés
2. Tout le nécessaire à fabriquer un exemplaire hardware de chaque :
3. Fichiers de fabrication (GERBER) / liste de pièces avec références pour commande / implantation (prototype) / modifications / dessins mécaniques, etc

4. Les fichiers sources de programmation microcontrôleur (.c / .h)
5. Un calcul / estimation des coûts
6. Un rapport contenant les calculs - dimensionnement de composants - structogramme, etc.

Convention de nommage et liens

Le nom de ce fichier doit être unique et doit contenir le nom du projet avec le format suivant :

aaii_ nomProjet-CDC _ Vn.docx

avec :

1. CDC : pour cahier des charges
2. aaii : numéro de projet, exemple 1708 pour le projet de 2017 numéro 8
3. Vn : ou n indique la version du document

Exemple :

1. **0910x_PICEthernet-CDC_V1.docx**

Stockage du fichier

Ce fichier sera stocké à la racine du dossier **/doc** d'un projet.

Ainsi, tous les fichiers de documentation faisant partie du projet sont centralisés dans le même répertoire.

Pré-étude

Filtrage numérique

Pour démarrer ce projet, j'ai commencé par me plonger dans le filtrage numérique. J'ai d'abord pensé faire un jeu de filtres passe bande pour chaque gamme voulue. Mais, cela complexifiait beaucoup le traitement, il aurait fallu traiter autant de tableaux que de gammes de fréquences voulues. Une transformée de Fourier nous permet de ne traiter qu'un seul tableau pour obtenir toutes les fréquences. Le calcul que nécessite un filtrage par passe bande est plus simple qu'une transformée de Fourier, mais il devrait être fait autant de fois que l'on veut de gamme de fréquence, sans les fréquences minimum et maximum, mais cela forme un total de neuf filtres à calculer par le microcontrôleur. Le calcul de la transformée de Fourier est plus compliqué, mais permet d'obtenir l'amplitude pour toutes les fréquences présentent dans le signal en une seule fois, et permettrait même d'ajouter ou de diminuer à volonté cette gamme de fréquences affichées sans ajouter de calcul. J'ai même trouvé une librairie microchip pour faire cette transformée, faite pour la famille de microcontrôleur PIC32.

Pour tester le fonctionnement de ce que j'apprenais sur le filtrage, j'ai, au fur et à mesure de mes recherches, testé différentes fonctions à l'aide de Octave GNU. Cela m'a permis de mieux apprêhender les points théoriques du filtrage numérique, mais surtout de faire une sorte de transition vers la programmation sur microcontrôleur. Le code sur Octave est bien plus simple que sur microcontrôleur, mais m'a donné une direction à prendre. Une fois la décision de partir sur une transformée de Fourier prise, j'ai pu coder un petit script me permettant de mettre en pratique cela. J'ai généré un signal formé par l'addition de plusieurs sinus de fréquence et d'amplitude différentes, puis j'en ai fait la transformée de Fourier. J'ai affiché le signal, sa transformée, ainsi qu'écris dans la console les fréquences où la transformée trouvait un pique. Le script est disponible dans les annexes.

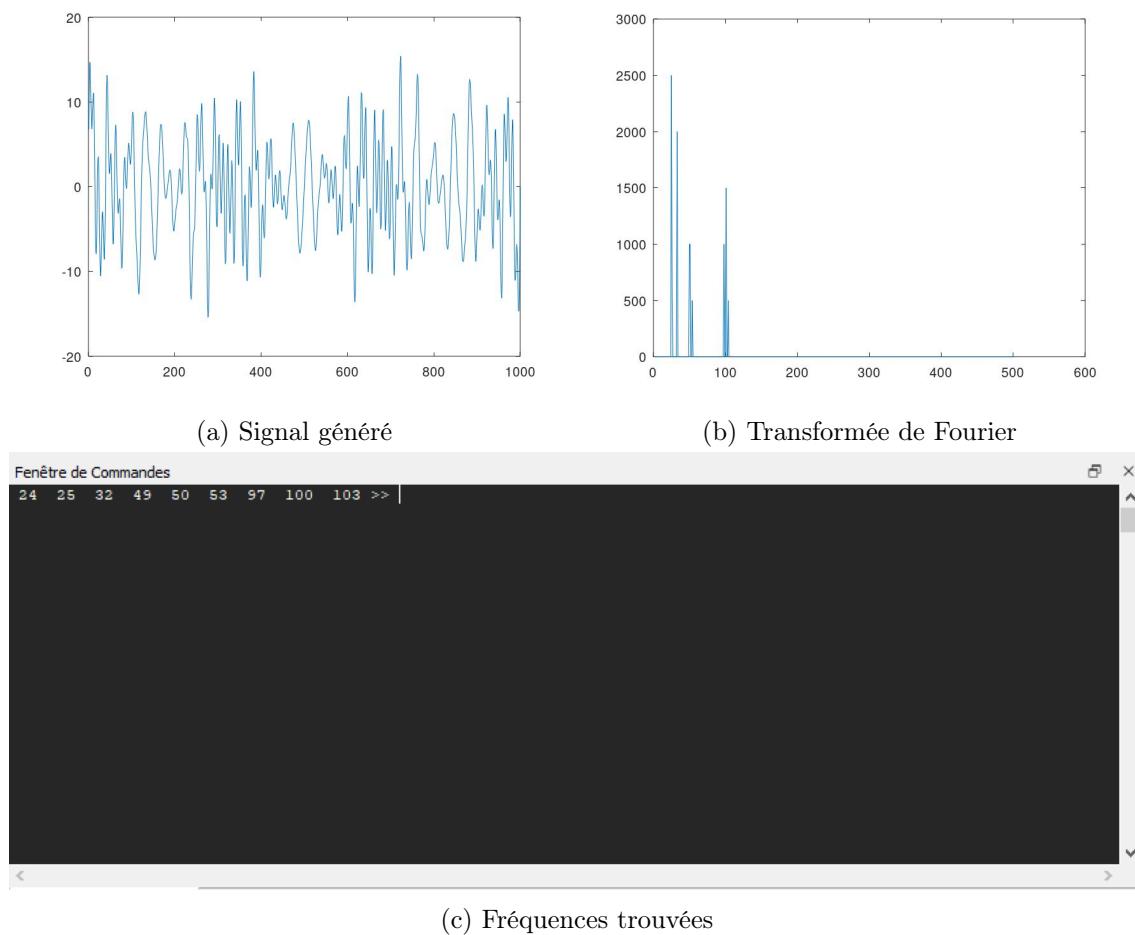


FIGURE 3 – Résultats du script Octave GNU

On voit donc ci-dessus, le signal généré, sa transformée de Fourier, et les valeurs en fréquences correspondant à chaque pique de la transformée de Fourier.

Schéma bloc général du système

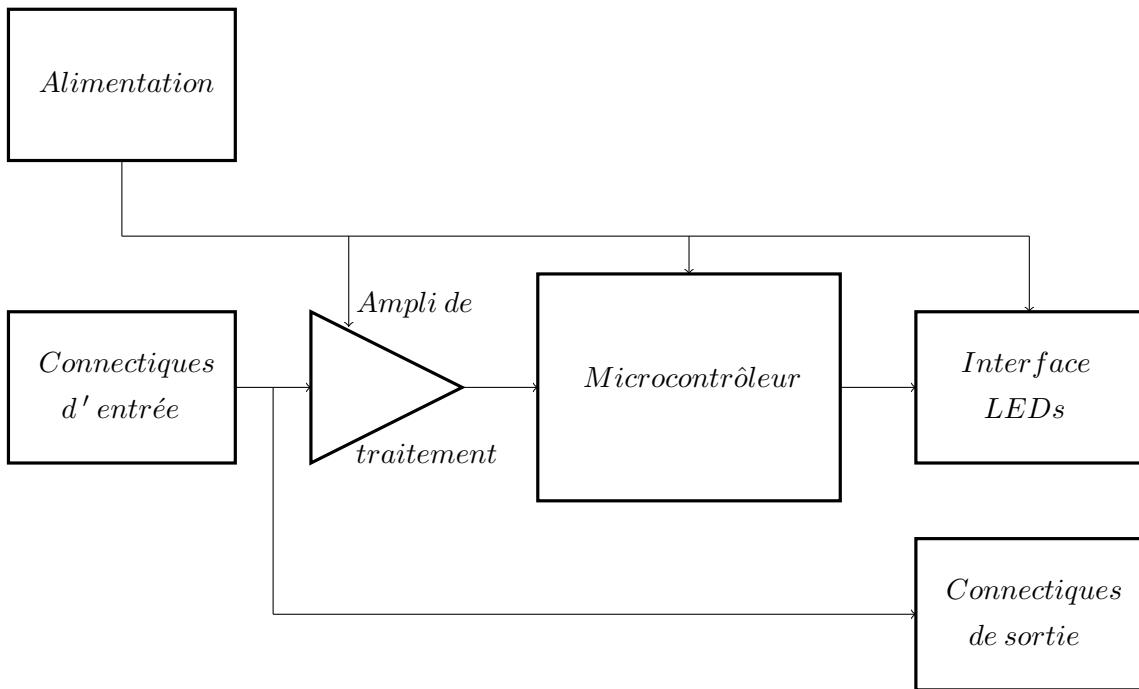


FIGURE 4 – Schéma bloc du système

Description des blocs :

- Alimentation : Ce bloc fournira l'alimentation pour l'ensemble du projet. Il y a principalement deux points à prendre en compte pour cela, alimenter le microcontrôleur en 3.3V et la partie amplification.
- Connectiques d'entrées : L'ensemble de connecteurs physique permettant de fournir un signal sonore au système. Mr. Bovey proposait les formats jack 3.5mm et bnc, et je pensais ajouter un jack 6.35mm pour utiliser le système avec un instrument de musique.
- Connectiques de sorties : Comme le système se branche en série entre une source et un ampli audio, les mêmes connecteurs qu'en entrée doivent se trouver dans ce bloc.
- Ampli de traitement : Comme le système viendra se brancher en série entre la source du signal et un ampli audio, il faut pouvoir traiter le signal sans toutefois le modifier. Ce bloc, très certainement un ampli OP en suiveur, permettra de séparer la partie traitée de la partie qui sera amplifiée, permettant de faire un true bypass.
- Microcontrôleur : Le cœur du projet, c'est le microcontrôleur qui traitera le signal sonore et fera le filtrage pour indiquer l'amplitude des différentes gammes de fréquences.
- Interface Leds : Ce bloc, qui sera éventuellement repris du projet de matrice à Leds 2126, permettra d'indiquer l'amplitude de chaque gamme de fréquences.

Schéma bloc hardware

Le schéma bloc général étant à la fois simple et suffisamment détaillé, le schéma bloc hardware en reprend les blocs, mais en les précisant au niveau composant.

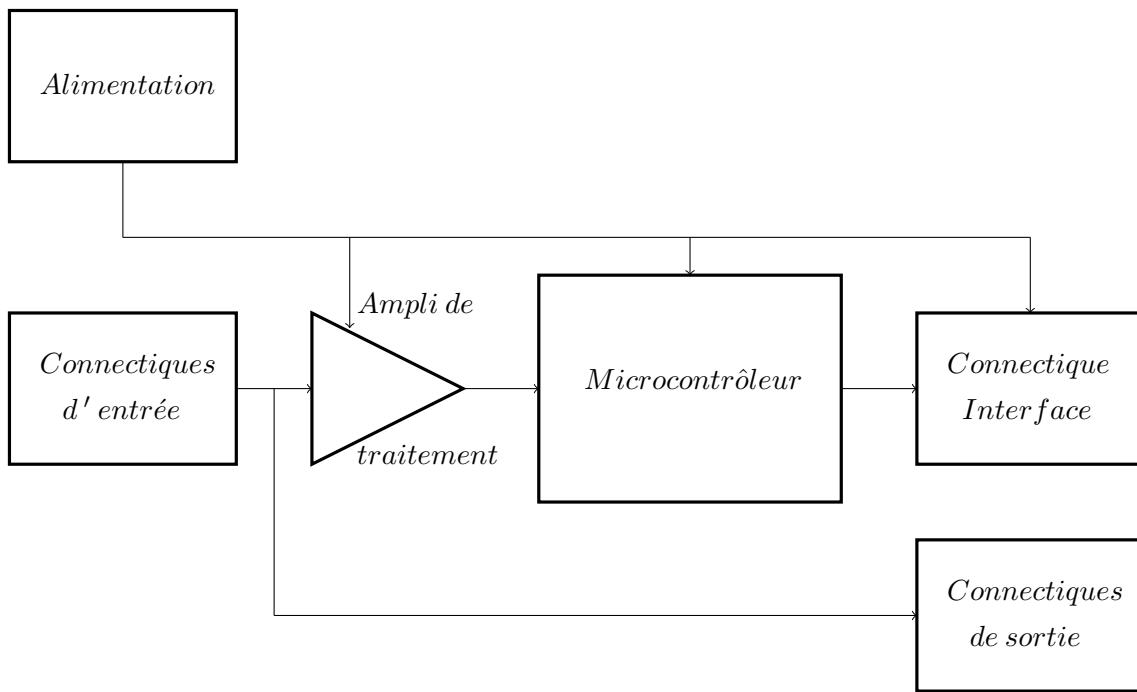


FIGURE 5 – Schéma bloc hardware

Description des blocs :

- Alimentation : Ce bloc prendra comme source un transformateur 24V extérieur, il faudra donc convertir cette tension en +3.3V, +5V et -5V, la première tension pour alimenter le microcontrôleur, et les deux suivantes pour l'ampli OP.
- Microcontrôleur : Mon choix se porte sur un modèle de PIC32MX pouvant fonctionner à 80MHz.
- Ampli de traitement : Un ampli OP simple en montage suiveur ou en amplification pour accorder le signal à la plage de l'ADC
- Connectiques, entrée et sortie : Jack 3.5mm, jack 6.35mm, BNC, ainsi qu'un montage soit pour ne sélectionner qu'un signal, soit pour les additionner.
- Connectique d'interface : Connectique pour l'interface Leds choisie.

Architecture firmware

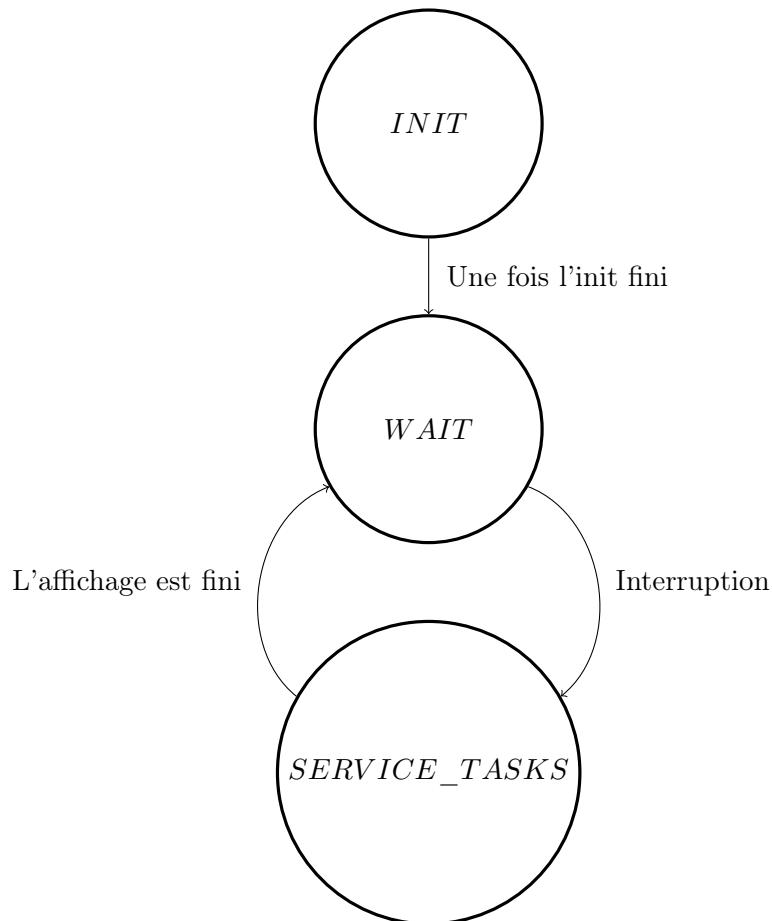


FIGURE 6 – Architecture firmware

La machine d'état du système est simple : une fois un état d'initialisation fini, le système va en état WAIT, et n'en sors qu'après une interruption. Cette interruption est détaillée plus bas. L'état SERVICE_TASKS traite les données, lance la transformée de Fourier, et affiche l'amplitude des gammes de fréquences sur les Leds.

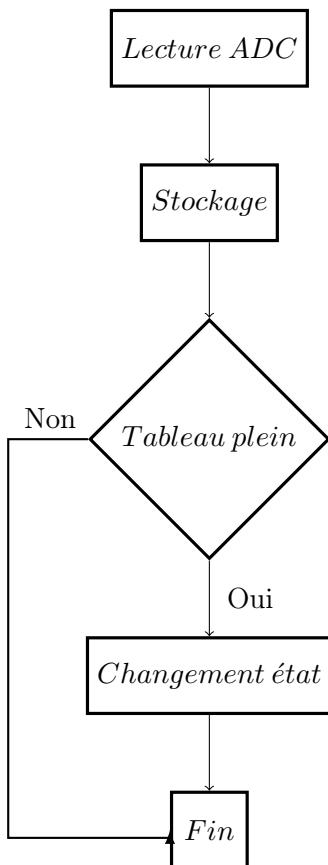


FIGURE 7 – Routine d'interruption

Voici comment se déroulera l'interruption, à chaque appel, on lit la valeur de l'ADC, on la stock dans un tableau, dont la taille est à déterminer, et si le tableau est plein, on change l'état de la machine d'état en SERVICE_TASKS.

En résumé, voilà le déroulement du programme : dans l'état WAIT, l'interruption va venir mémoriser la valeur de l'ADC à une fréquence de 40kHz, une fois le tableau plein, l'interruption se coupe et la machine d'état passe en état SERVICE_TASKS. Dans cet état, elle va mémoriser puis traiter, en faisant la transformée de Fourier ; en traitant et triant les piques trouvés par celle-ci pour les afficher sur l'interface Leds. Une fois l'affichage envoyé, la routine d'interruption est à nouveau autorisée, et tout recommence.

Choix des technologies clés du système

Microcontrôleur

Pour le choix du microcontrôleur du projet, l'important n'est pas le nombre de ports disponibles, mais la vitesse de calcul, en effet, pour pouvoir travailler sur le traitement d'un signal d'une fréquence de 20kHz, il faut au minimum pouvoir échantillonner à 40kHz. Ainsi, je pensais me tourner vers les modèles PIC32MX340FS512H ou PIC32MX795FS512H, ils peuvent être cadencés jusqu'à 80MHz, et étant des versions "H", ils ne possèdent que 64 pins, ce qui sera déjà plus que suffisant pour ce projet.

Ampli de traitement

Pour garantir un true bypass pour le système, un ampli permettra d'assurer une séparation entre le signal traversant le système, et celui qui sera traité par le microcontrôleur. Le montage de l'ampli OP pourra donc être simplement un suiveur, ou éventuellement un ampli (inverseur ou non) pour venir accorder l'amplitude du signal à la plage de l'ADC et ainsi garantir une meilleure précision de mesure.

Alimentation

Pour la partie alimentation du projet, une tension de 24V sera amenée par un transformateur extérieur. Il faudra commencer par convertir cette tension à une valeur plus facile à gérer, environ 6V, à l'aide d'un transformateur, puis la faire passer par plusieurs régulateurs, je vais sûrement me diriger vers du step-down, aux tensions utiles au projet, 3.3V, 5V et -5V.

Évaluation des coûts

Il n'y a pas beaucoup de composants clés pour ce projet, l'essentiel de la complexité se trouvant dans le traitement du signal par le microcontrôleur.

Composants	Prix minimum [CHF]	Prix maximum [CHF]
PIC32MX795F512H	11.32	14.15
Ampli OP linéaire	0.60	7.70
PCB (entre 7 à 14 jours)	41.92	50.72
Prise jack 6.35 femelle	1.36	2.62
Prise jack 3.5 femelle	1.16	2.81
Prise BNC femelle	1.24	2.54
Total	57.60	80.54

Planning

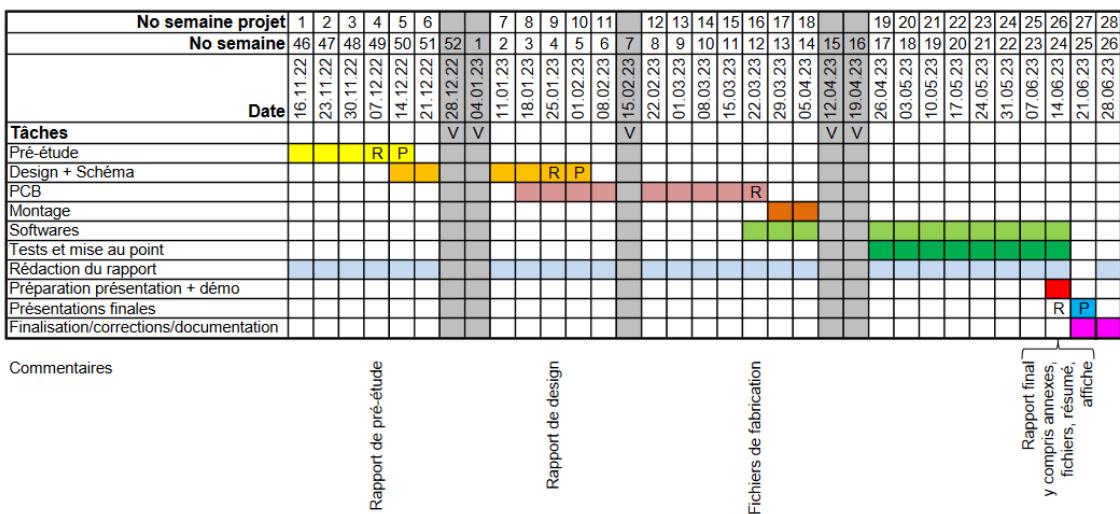


FIGURE 8 – Planning

Conclusion et perspectives

Grâce à cette pré-étude, je sais dans quelles directions orienter mes efforts pour la suite de ce projet. Le filtrage et le traitement de signal sont des sujets que j'avais pu aborder lors de mes années d'études en ingénierie à la HEIG-VD, mais je n'avais jamais appliqué cela avec un microcontrôleur. Le domaine de l'électronique audio étant proche de ma passion pour la musique, je me réjouis de mener à bien ce projet.

Cette pré-étude m'a permis de réviser mes connaissances sur le traitement de signal, particulièrement les transformées de Fourier, dont je me souvenais de l'utilité, mais plus de la manière de les calculer.

Design

Changements suite à la pré-étude

Suite à la présentation faite après la pré-étude, quelques changements de directions ont été apportés.

1. Pour limiter le nombre d'alimentations, celle de l'ampli sera asymétrique, il suffira d'ajouter un offset, cela ne changera rien au traitement du signal. (Le kit de programmation de première année possède un montage qu'il suffirait de modifier pour ce projet).
2. Pour améliorer la lecture de l'ADC, il faut ajouter une référence de tension 3.3V.

Schéma

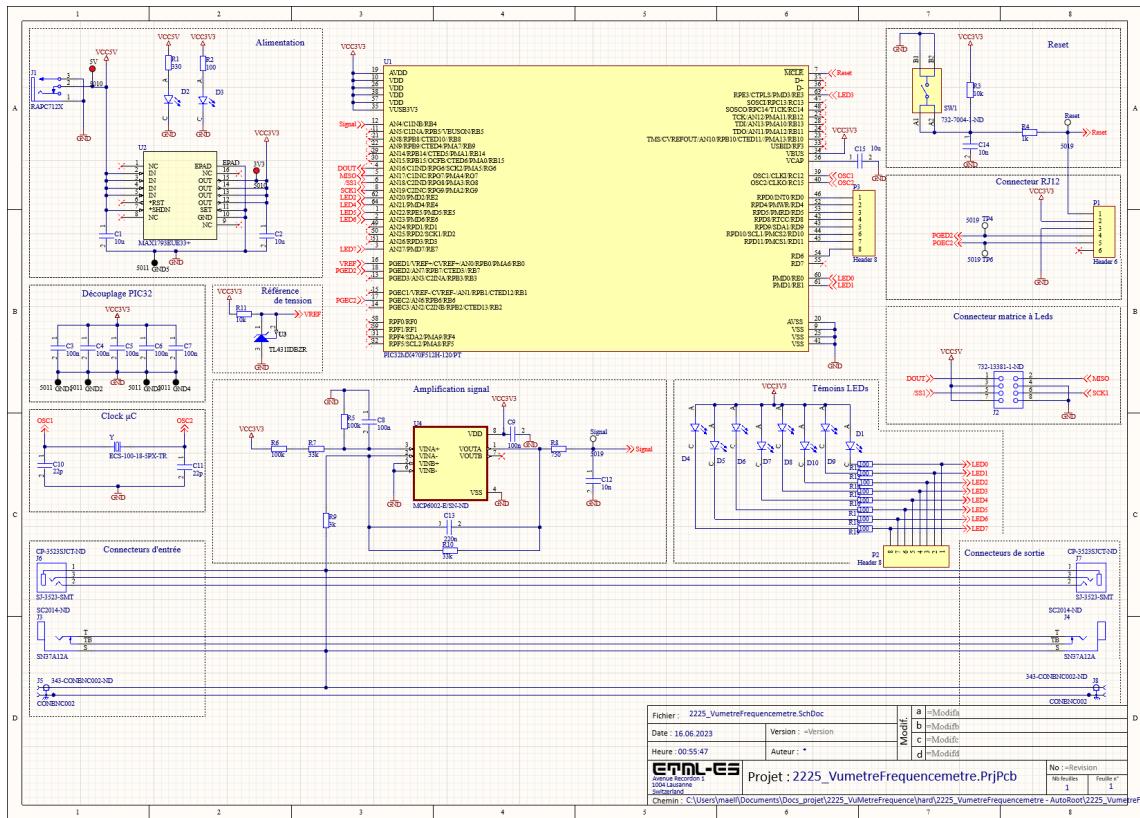


FIGURE 9 – Schéma complet

Je suis consciente que ces traits qui traversent tout le bas du schéma ne sont pas très élégants, je voulais les laisser pour bien montrer que ce soit sur le schéma ou le PCB, ces connexions sont true-bypass, et traverse l'entièreté du système sans être modifiées.

J'ai demandé à Miguel Santos de faire une review de mon schéma, cela m'a permis de corriger mes GND que j'avais laissés avec le symbole normalement associé à la terre, ainsi que les pins VBUS et VCAP que j'avais oublié de connecter (ce qui aurait empêché le microcontrôleur de s'allumer).

J'ai essayé d'ajouter un nombre suffisant de points de test pour le futur du projet. Il y a un point de test pour chaque tension d'alimentation, 5 points de GND, et un point de test pour chaque signal.

Choix des composants

Microcontrôleur et clock

Pour assurer une vitesse de calculs suffisante, je me suis tournée vers un modèle de PIC32 pouvant être cadencé à 120MHz, et n'ayant pas besoin de beaucoup d'entrées/-sorties, j'ai choisi le modèle half, soit le PIC32MX470F512H-120/PT. J'aurais bien voulu avoir un microcontrôleur avec encore moins de pins, mais celui-ci est le modèle de la gamme PIC32 avec le plus petit nombre de pattes. Au moment de la rédaction de ces lignes, ce modèle est disponible sur digikey, avec un stock de plus de 400 pièces.

Dans une optique de faire des mesures le plus précisément possible, j'ai choisi un modèle de quartz avec une précision de 10ppm. Pour que des mesures temporelles soient utilisables dans mon cas, il est nécessaire que le cadencement même de ces mesures soit précis, j'ai donc décidé de mettre un petit peu plus cher dans l'achat du quartz. Même si le prix d'un quartz 10ppm par rapport à un modèle 30ppm plus de double, cela reste en dessous de 1CHF.

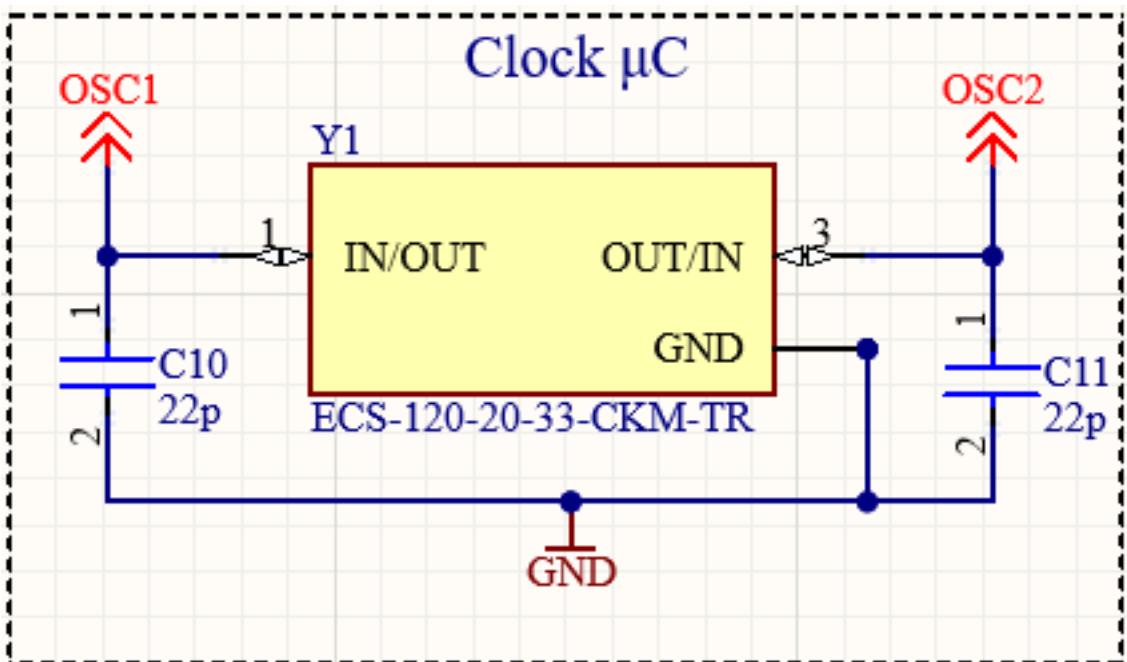


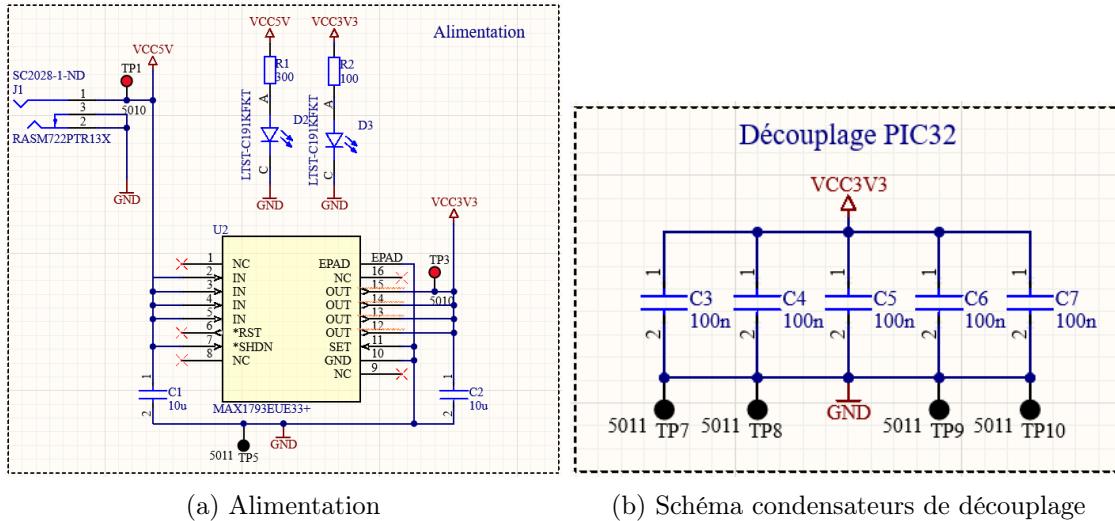
FIGURE 10 – Schéma du quartz

Partie alimentation et découplage

Pour la partie alimentation du projet, je me suis grandement inspirée du kit ES avec lequel nous travaillons pendant le cours de MINF. J'ai donc choisi un MAX1793 (modèle MAX1793EUE33+ dans mon cas) pour convertir le 5V et 3.3V. Ce composant est un peu cher (environ 4CHF/pièce), mais il a pour avantages d'être en stock à l'ES, et aussi d'être le modèle des cartes de développement, et

J'ai mis également des leds témoins pour l'alimentation, sur le 5V et le 3.3V, pour donner une indication simple pour le dépannage.

Toujours en m'inspirant du kit de développement, j'ai mis cinq condensateurs de 100nF pour le découpage du microcontrôleur.



Référence de tension

Pour ajouter à la précision de mesure de mon système, j'ai ajouté une référence de tension 3.3V. Pour faire cela, je suis partie sur la version simple à l'aide d'une TL431. Cette tension n'est connectée qu'à la pin VREF+ du microcontrôleur, et n'est pas influencée par le reste du circuit.

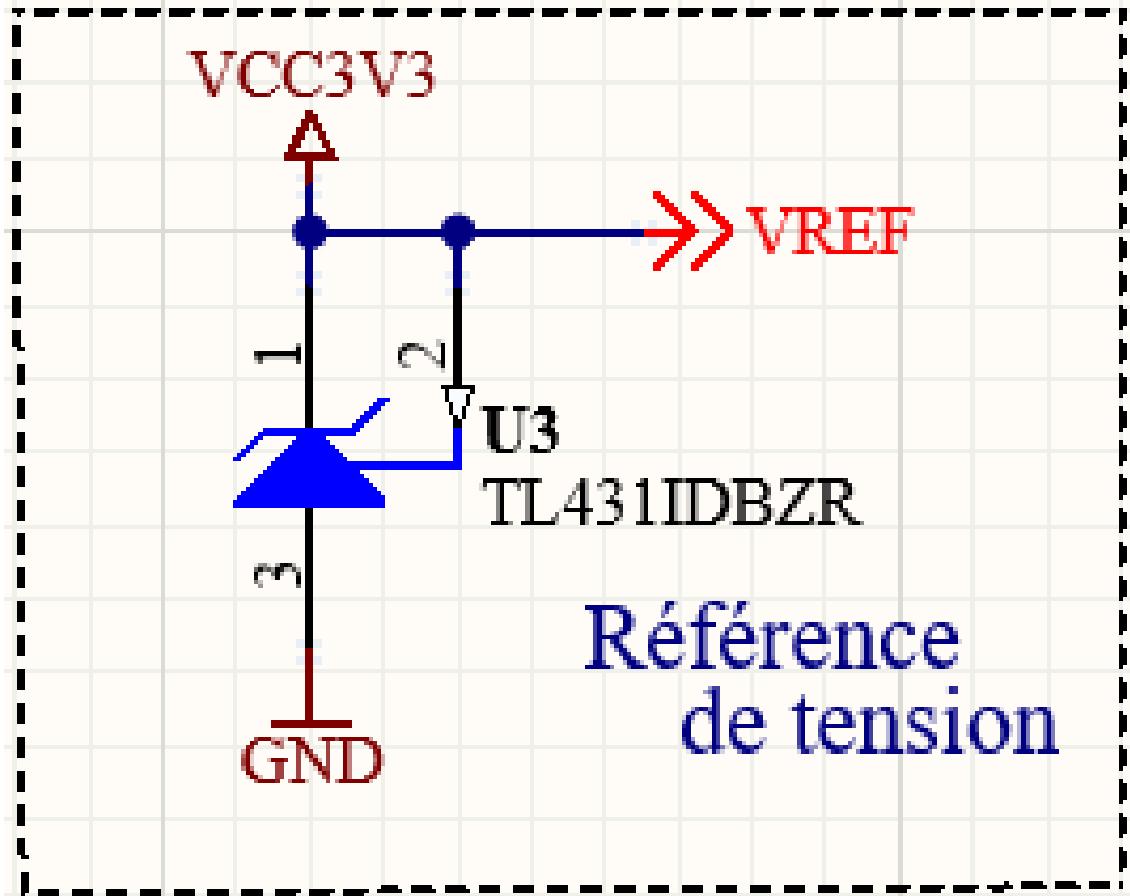


FIGURE 12 – Schéma tension de référence

Connectiques d'entrées et sorties audio

Pour la connectique audio, rien de compliqué, jack 3.5mm et 6.35mm ainsi que du BNC. Les mêmes connecteurs se retrouvent en sortie de la carte en true-bypass.

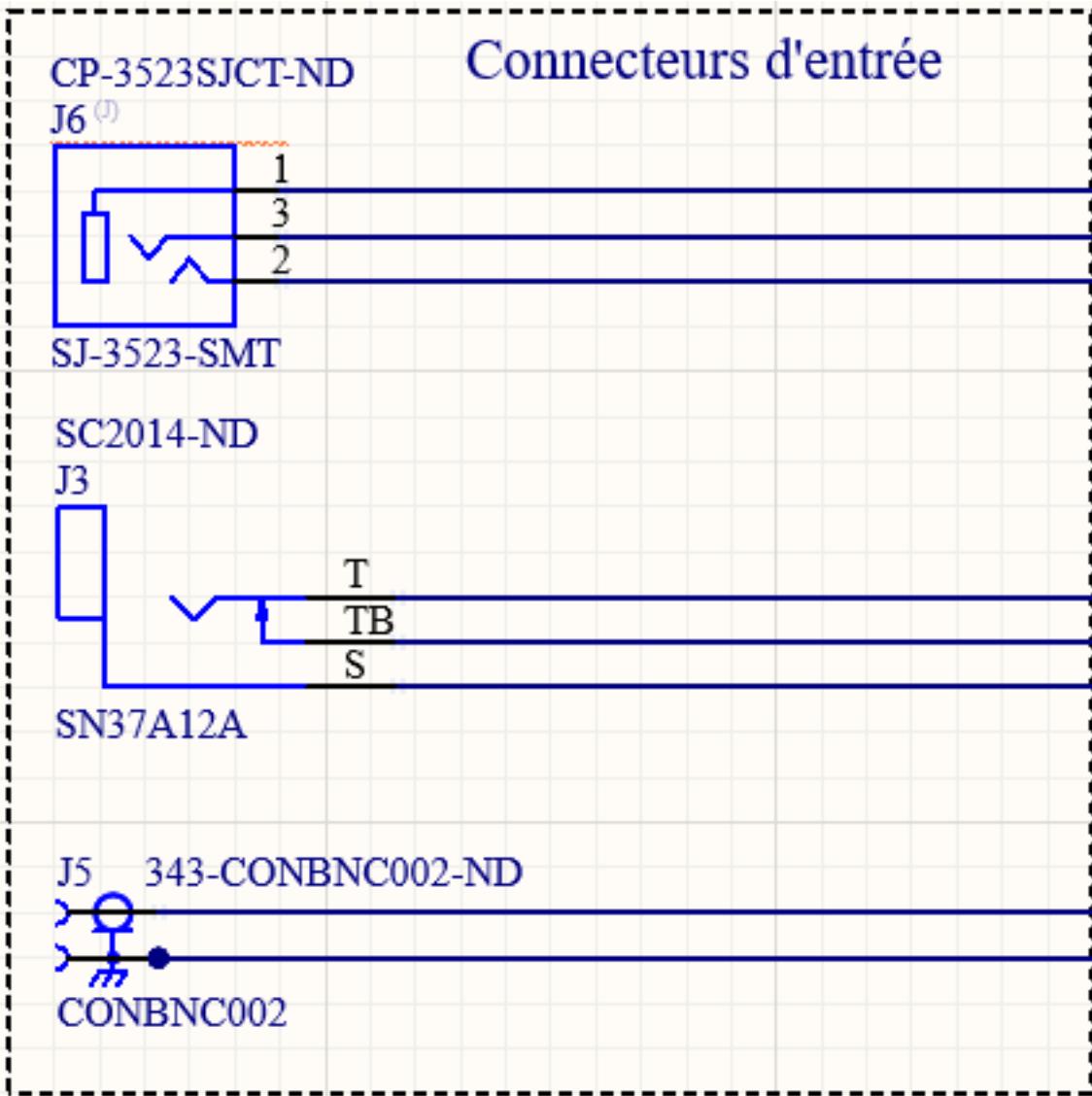


FIGURE 13 – Schéma connectique Audio

Connectique avec la matrice à Leds

Pour la partie affichage du projet, je me suis tournée vers le projet 2126 qui est une matrice à Leds. J'ai donc repris le même connecteur que ce projet et suivis la même logique de connexion.

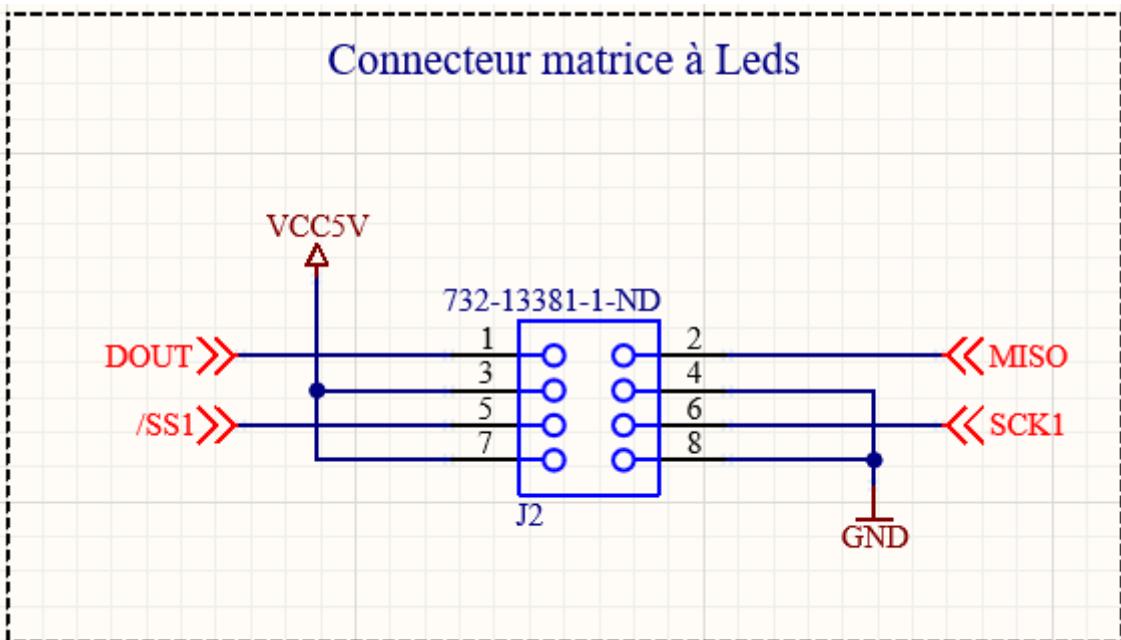


FIGURE 14 – Schéma connecteur pour la matrice à Leds

Reset et connecteur de programmation

J'ai repris cette partie du schéma du kit de développement de l'ES. Il me fallait de toute façon un connecteur de programmation, et comme il se trouvait sur le schéma d'origine avec le circuit de reset, je me suis dit qu'un switch de reset pourrait s'avérer utile au moins lors de la phase de programmation de la carte.

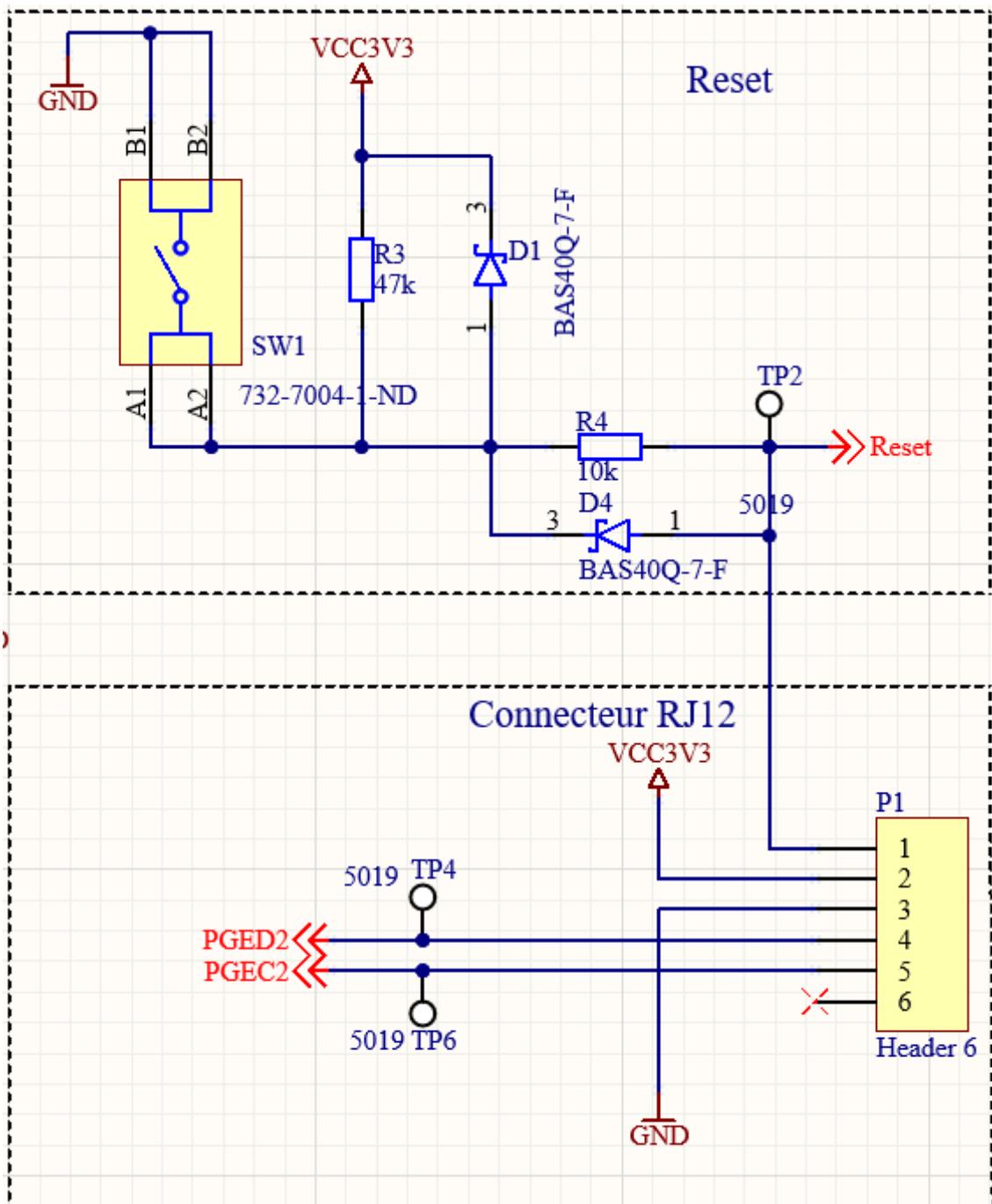


FIGURE 15 – Schéma reset et connecteur de programmation

Amplification du signal audio

Pour cette partie du schéma, je me suis inspirée du projet KitArm0 pour faire une adaptation du signal audio. La tension normalisée pour un niveau ligne grand public est de 316mV efficaces, et pour profiter au maximum de la précision de l'AD, j'ai voulu amplifier ce signal pour m'approcher d'une tension efficace de 3.3V.

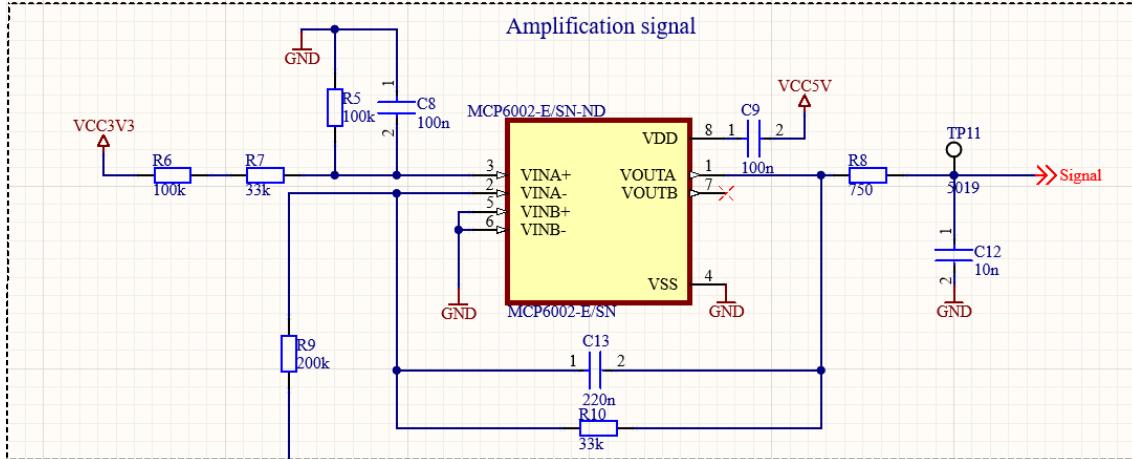


FIGURE 16 – Schéma Ampli Audio

BOM intermédiaire

Pour me donner une idée plus précise du prix lors du design, j'ai tenu une petite BOM toute simple. L'intégralité des prix viennent de digikey au moment de leur sélection, bien évidemment au moment de la commande, je regarderais sur quel site ces composants seront disponibles au meilleur prix. Les composants ont été choisis également selon leur disponibilité.

Composants	Quantité	Prix [CHF]
PIC32MX470F512H-120/PT	1	9.83
Ampli-OP	1	0.4
Connecteur Jack 3.5mm	2	0.94
Connecteur Jack 6.35mm	2	3.63
Connecteur BNC	2	2.87
Connecteur programmation	1	0.82
Connecteur alimentation	1	2.31
Quartz 12MHz	1	0.93
Diode Schottky 40V 200mA	2	0.32
Bouton poussoir	1	0.49
Regulateur 5V-3V3	1	4.06
Led témoin alimentation	2	0.22
Connecteur matrice à Leds	1	1.68
Référence de tension	1	0.6
Total	19	37.08

Il est à noter qu'il manque encore le prix des résistances et condensateurs, ainsi que les points de test.

Conclusion et perspectives

Je suis plutôt contente du design sur lequel j'ai pu aboutir, il est simple, et j'espère qu'il puisse être tout autant efficace. Il reste encore un grand nombre de pins disponible sur le microcontrôleur (que je n'ai pas encore mises en no-connect), et il se peut qu'une utilité leur sera trouvée dans le futur du projet.

J'ai essayé pour ce design de m'inspirer au maximum de projet déjà abouti à l'ES pour limiter les risques, et également faciliter le dépannage en cas de problème.

Montage et programmation

Montage

Une fois les composants et le PCB commandés et reçus, j'ai pu commencer le montage de la carte. Il n'y a pas grand-chose de spécial à dire sur cette partie, tout s'est passé sans problème en ce qui concerne le brasage des composants. Il n'y a eu aucun problème de footprints ou de pistes créant un court-circuit.

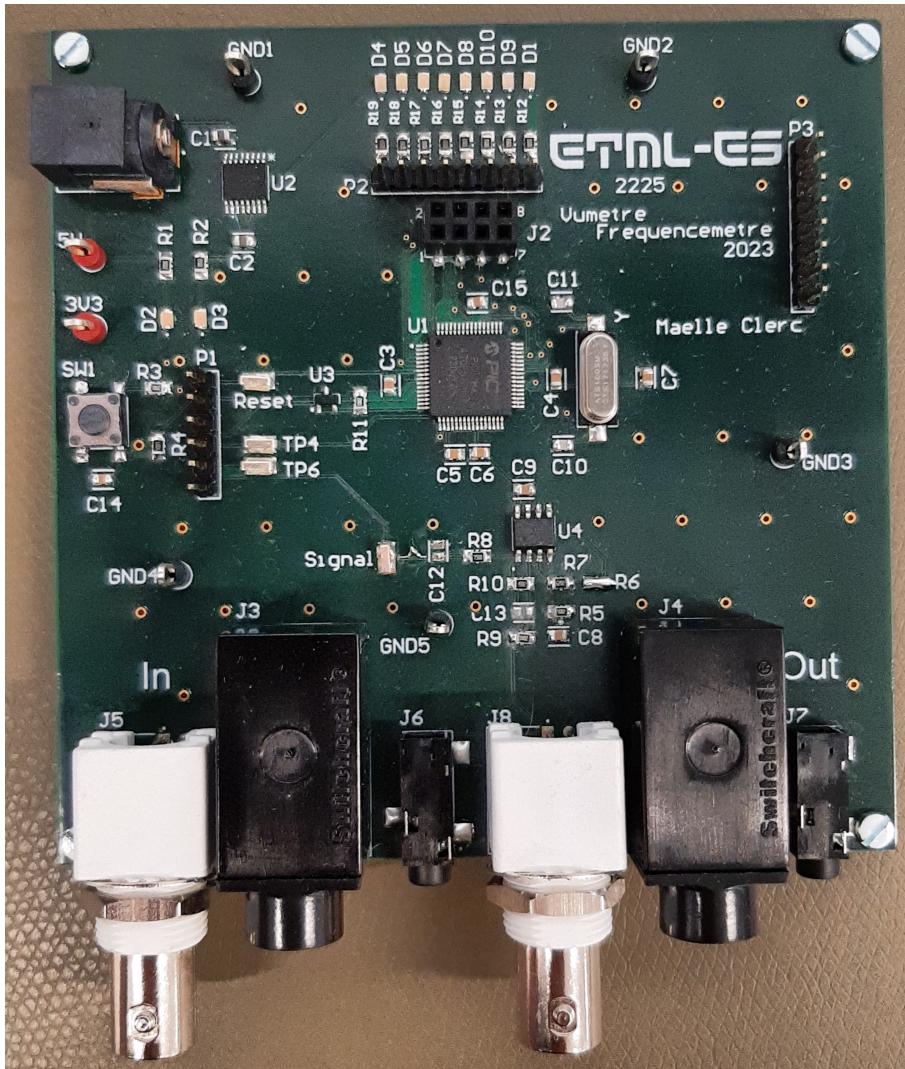


FIGURE 17 – PCB monté

Connecteur 5V

Bien que le footprint du connecteur 5V que j'ai utilisé était le bon, j'ai malheureusement inversé sa polarité. Cela ne m'a pas empêchée de continuer à travailler sur le projet, car j'ai pu utiliser un câble de bon type en inversant simplement le plus et le moins sur l'alimentation de laboratoire de ma place de travail. C'est cependant quelque chose qu'il faudra modifier dans une version future du projet (cette mention se retrouve dans le document MOD).

Modification de la partie amplification

Il s'est avéré, lors de mes premiers tests de la partie analogique de la carte, que mon dimensionnement de la partie amplification était mauvais. En effet, le signal reçu était bien trop peu amplifié pour pouvoir être utilisable :

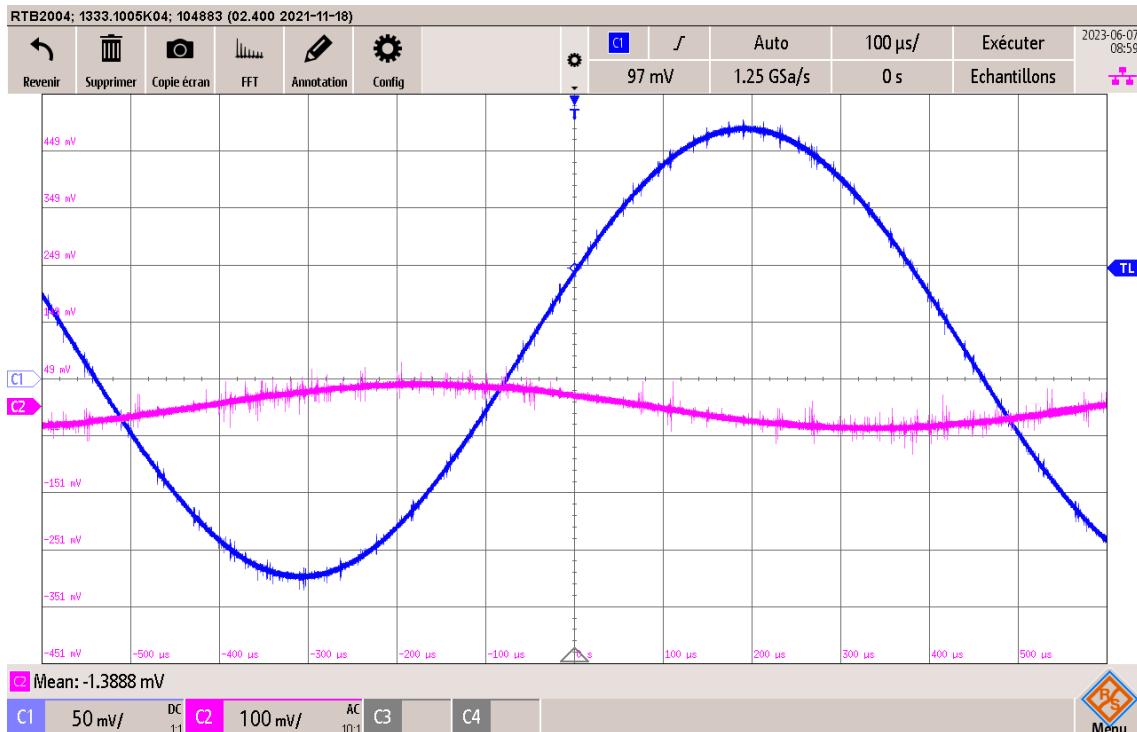


FIGURE 18 – Signal en sortie de l'Ampli

On peut voir sur cet oscilloscopogramme que le signal en sortie de l'ampli est atténué alors qu'il devrait être amplifié. Mes calculs pour les valeurs de résistances étaient faux, et grâce à l'aide de M. Castoldi, j'ai pu les reprendre :

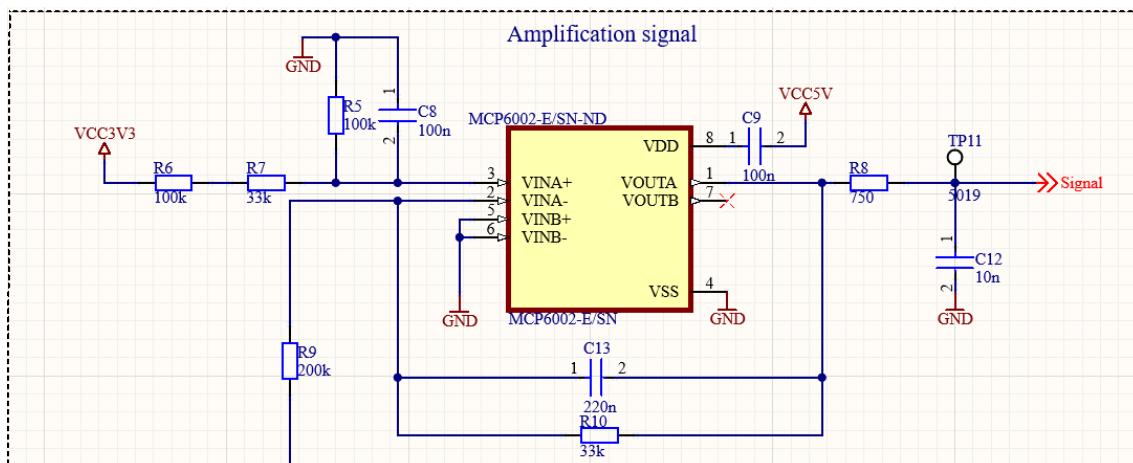


FIGURE 19 – Schéma Ampli Audio avant modification

$$U_{out} = -U_{in} * \frac{R_{10}}{R_9} + V_{DD} + \frac{R_4}{R_7 + R_5} * (1 + \frac{R_{10}}{R_9}) \quad (1)$$

Avec cette équation, nous arrivons à la conclusion qu'il faut changer les valeurs des résistances comme suit :

1. R_9 vaut $10k\Omega$
2. R_{10} vaut $33k\Omega$
3. R_7 vaut $62k\Omega$
4. R_5 vaut $10k\Omega$

Nous nous sommes également rendus compte que la valeur de C_{13} n'était plus valable non plus et devrait être remplacé par un condensateur de $220pF$.

Dans cette même suite de mesure, nous avons remarqué que la tension de référence était mauvaise, la valeur de la résistance R_{11} devrait être changée pour une 390Ω .

Pour tester la partie hardware, j'ai retiré les condensateurs C_{12} et C_{13} et remplacé les résistances citées plus haut (sauf celle de la tension de référence). Une fois ces modifications faites, le signal est bien amplifié comme il faut :

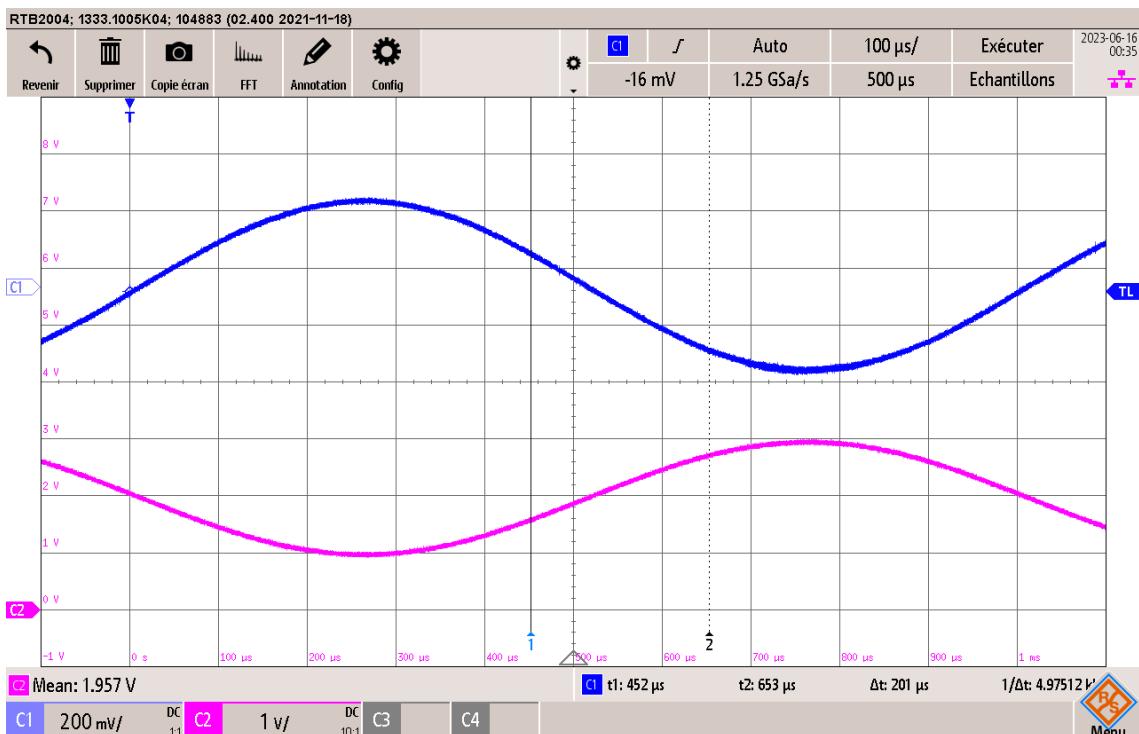


FIGURE 20 – Signal de sortie de l'Ampli après modifications

Il faudrait donc encore braser les nouveaux condensateurs ainsi que la résistance pour la tension de référence (ces modifications se trouvent dans le document MOD).

Programmation

Pin Settings

La toute première chose que j'ai faite pour la programmation du microcontrôleur, une fois que j'ai constaté que je pouvais dialoguer avec, fut de nommer toutes les pins utiles au

projet. Pour ce faire, j'ai utilisé le configuateur graphique de Harmony, et en ayant mon schéma sous les yeux, j'ai nommé chaque pin avec le même nom que je lui avais attribué sur mon schéma.

Pin Number	Pin ID	Voltage Tolerance	Name	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)
1	RE5		LED5	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
2	RE6		LED6	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
3	RE7		LED7	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
4	RG6		DOUT	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
5	RG7		MISO	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
6	RG8		SS1	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
7	MCLR	5V			In	n/a	<input type="checkbox"/>	Digital
8	RG9		SCK1	GPIO_OUT	Out	Low	<input type="checkbox"/>	Digital
9	VSS				In	n/a	<input type="checkbox"/>	Digital
10	VDD				In	n/a	<input type="checkbox"/>	Digital
11	RB5			Available	In	n/a	<input type="checkbox"/>	Analog
12	RB4		Signal	AN4	n/a	n/a	<input type="checkbox"/>	Analog
13	RB3			Available	In	n/a	<input type="checkbox"/>	Analog
14	RB2			Available	In	n/a	<input type="checkbox"/>	Analog
15	RB1			Available	In	n/a	<input type="checkbox"/>	Analog
16	RB0		VREF	AN0	n/a	n/a	<input type="checkbox"/>	Analog
17	RB6		PGE2	Available	In	n/a	<input type="checkbox"/>	Analog
18	RB7		PGED2	Available	In	n/a	<input type="checkbox"/>	Analog
19	AVDD				In	n/a	<input type="checkbox"/>	Digital
20	AVSS				In	n/a	<input type="checkbox"/>	Digital
21	RB8			Available	In	n/a	<input type="checkbox"/>	Analog
22	RB9			Available	In	n/a	<input type="checkbox"/>	Analog
23	RB10			Available	In	n/a	<input type="checkbox"/>	Analog
24	RB11			Available	In	n/a	<input type="checkbox"/>	Analog
25	VSS				In	n/a	<input type="checkbox"/>	Digital
26	VDD				In	n/a	<input type="checkbox"/>	Digital
27	RB12			Available	In	n/a	<input type="checkbox"/>	Analog

FIGURE 21 – Pin Settings partie 1

Pin Number	Pin ID	Voltage Tolerance	Name	Function	Direction (TRIS)	Latch (LAT)	Open Drain (ODC)	Mode (ANSEL)
38	VDD				In	n/a	□	Digital
39	RC12		OSC1	OSC1	n/a	n/a	□	Analog
40	RC15		OSC2	OSC2	n/a	n/a	□	Analog
41	VSS				In	n/a	□	Digital
42	RD8	5V	Supp3	GPIO_OUT	Out	Low	□	Digital
43	RD9	5V	Supp4	GPIO_OUT	Out	Low	□	Digital
44	RD10	5V	Supp5	GPIO_OUT	Out	Low	□	Digital
45	RD11	5V	Supp6	GPIO_OUT	Out	Low	□	Digital
46	RD0	5V	Supp0	GPIO_OUT	Out	Low	□	Digital
47	RC13			Available	In	n/a	□	Analog
48	RC14			Available	In	n/a	□	Analog
49	RD1			Available	In	n/a	□	Analog
50	RD2			Available	In	n/a	□	Analog
51	RD3			Available	In	n/a	□	Analog
52	RD4	5V	Supp1	GPIO_OUT	Out	Low	□	Digital
53	RD5	5V	Supp2	GPIO_OUT	Out	Low	□	Digital
54	RD6	5V	Supp7	GPIO_OUT	Out	Low	□	Digital
55	RD7	5V		Available	In	n/a	□	Digital
56	VCAP				In	n/a	□	Digital
57	VDD				In	n/a	□	Digital
58	RF0	5V		Available	In	n/a	□	Digital
59	RF1	5V		Available	In	n/a	□	Digital
60	RE0	5V	LED0	GPIO_OUT	Out	Low	□	Digital
61	RE1	5V	LED1	GPIO_OUT	Out	Low	□	Digital
62	RE2		LED2	GPIO_OUT	Out	Low	□	Digital
63	RE3	5V	LED3	GPIO_OUT	Out	Low	□	Digital
64	RE4		LED4	GPIO_OUT	Out	Low	□	Digital

FIGURE 22 – Pin Settings partie 2

Configuration des timers

Toujours à l'aide du configIBUTEUR graphique de Harmony, voilà comment j'ai réglé les deux timers que j'ai utilisé pour le projet,

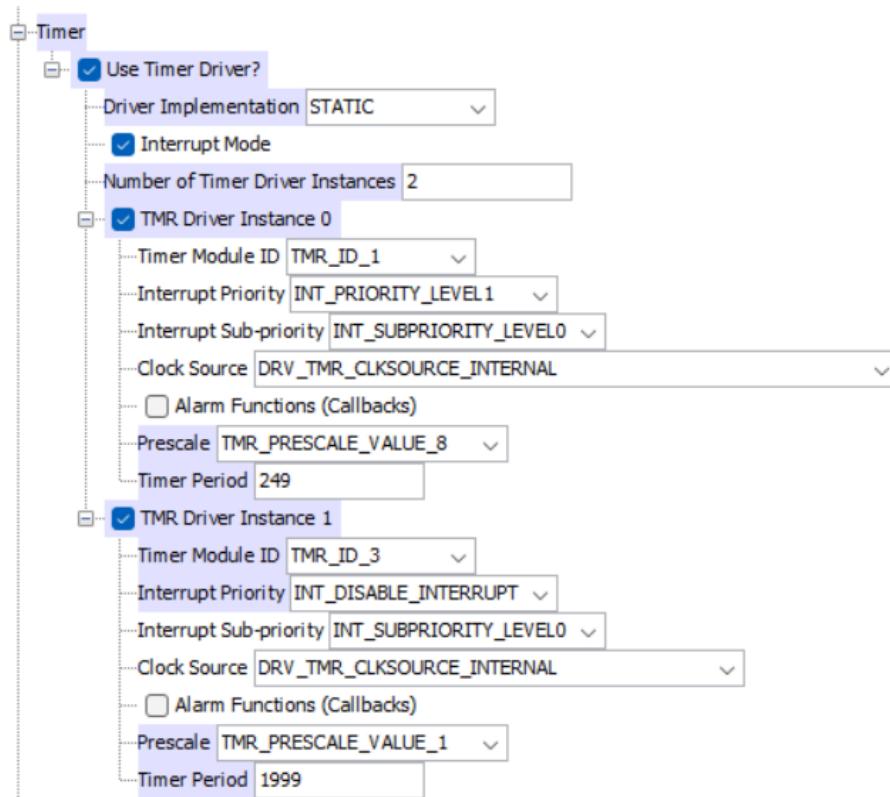


FIGURE 23 – Configuration des timers

Configuration de l'ADC

Grâce à l'aide de M. Castoldi, j'ai pu configurer l'ADC pour qu'il génère une interruption après avoir capturé 8 valeurs analogiques et qu'il les ait mémorisés dans le buffer :

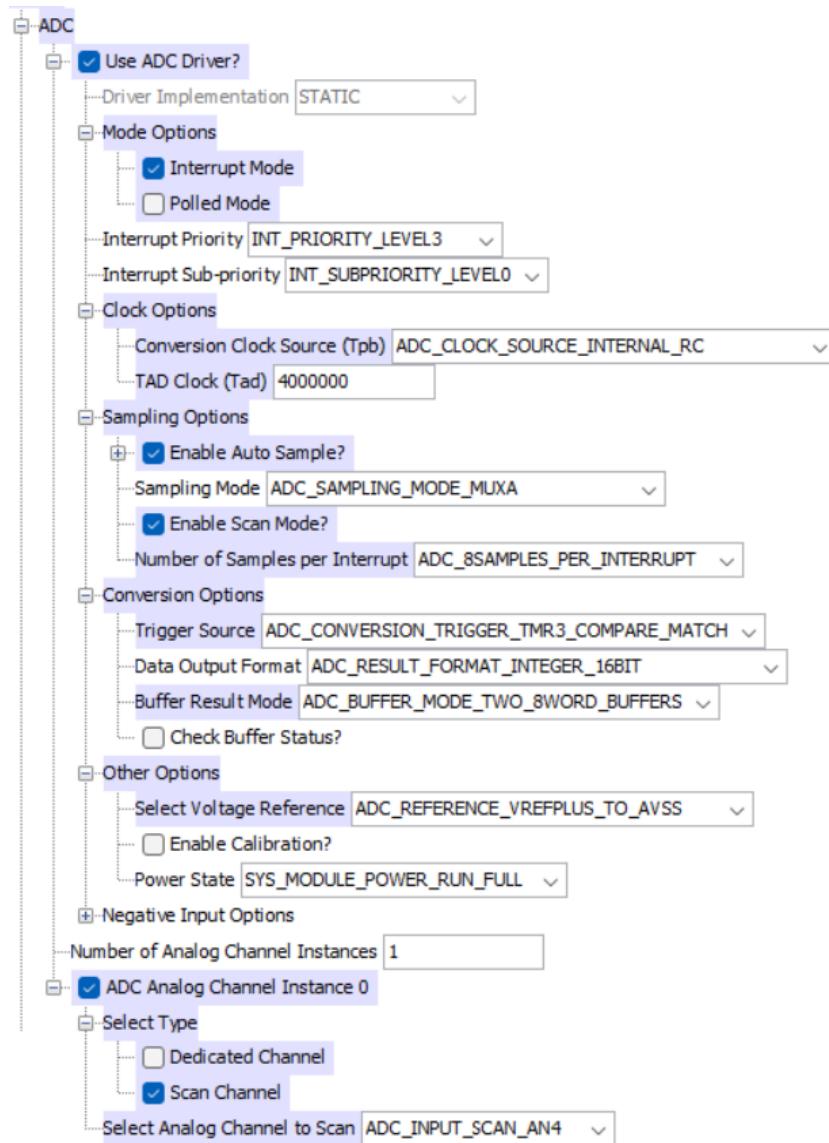


FIGURE 24 – Configuration de l'ADC

Débogage

Lorsque j'ai voulu essayer de débugger mon code pour la première fois, un problème est survenu : il est impossible de débugger à 120Mhz.

```

Project Loading Error x Configuration Loading Error x Project Loading Warning x 2225_spectrometre (Build,Load,...) x Debugger Console x ICD 3 x
.../.../.../.../...
Target device PIC32MX470F612H found.
Device Revision ID = c0000000

Device Erased...
Programming...
The following memory area(s) will be programmed:
program memory: start address = 0x1d000000, end address = 0x1d0027ff
boot config memory
configuration memory
Programming/Verify complete
The target device is not ready for debugging. Please check your configuration bit settings and program the device before proceeding. The most common causes for this failure are oscillator and/or PGC/PGD settings.

```

FIGURE 25 – Affichage du problème lors du debuggage

J'ai essayé ensuite d'utiliser le postscaler du peripheral bus clock en pensant que c'était la fréquence des périphériques qui créait ce problème, mais aucun changement. La seule

version que j'ai réussi à faire fonctionner pour le debug était de mettre le system clock à 100MHz et de diviser le peripheral bus clock par deux. Sinon, toute autre configuration supérieure à 80Mhz rencontrait le même problème. J'ai essayé également avec d'autres câbles de connexion, d'autre ICD3, même avec un ICD4, le problème persiste.

Abandon de la transformée de Fourier

Lors de la phase de pré-étude, j'étais partie sur une transformée de Fourier, car le site de Microchip présentait sa librairie dsp.h comme étant compatible avec les microcontrôleurs de la famille PIC32. (le lien de cette page est disponible en page 40)

Mais, il s'est malheureusement avéré que cette librairie n'est compatible qu'avec des DSPIC, le lien des fichiers sur cette page n'est pas à jour avec le texte présenté plus haut au même lien. L'entête ainsi que le nom des fonctions n'est pas le même dans les fichiers que sur le site. Et surtout, les fonctions de fft elles-mêmes sont codées en assembleur dans une version compatible uniquement avec des DSPIC.

Filtrage numérique

Pour pouvoir designer le filtrage numérique, je me suis aidée du site internet que m'a fourni M. Moreno TFilter (voir l'adresse en page 40).

Ce site permet de calculer les facteurs de filtrage, ainsi même que le code en C++ pour ce filtre. Voilà l'interface du site :

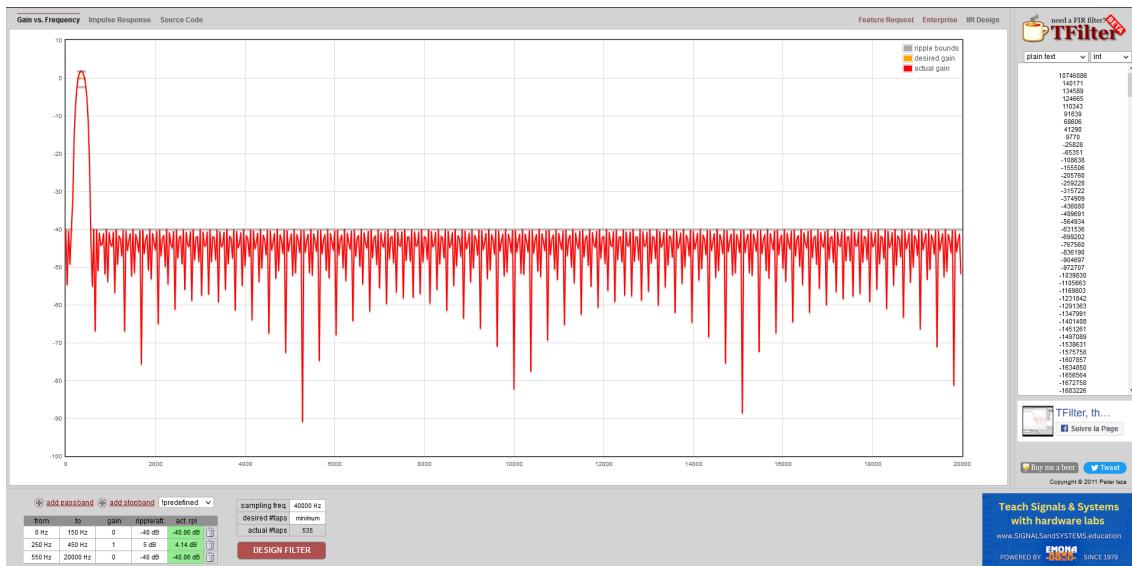


FIGURE 26 – Interface de TFilter

On peut contrôler le filtre que l'on veut créer à l'aide du rectangle en bas à gauche. Chaque ligne tout à gauche permet de contrôler les bandes de fréquences qui nous intéressent ainsi que le gain et le ripple correspondant. Il ne faut pas oublier d'indiquer en haut à droite la fréquence d'échantillonnage que l'on utilise.

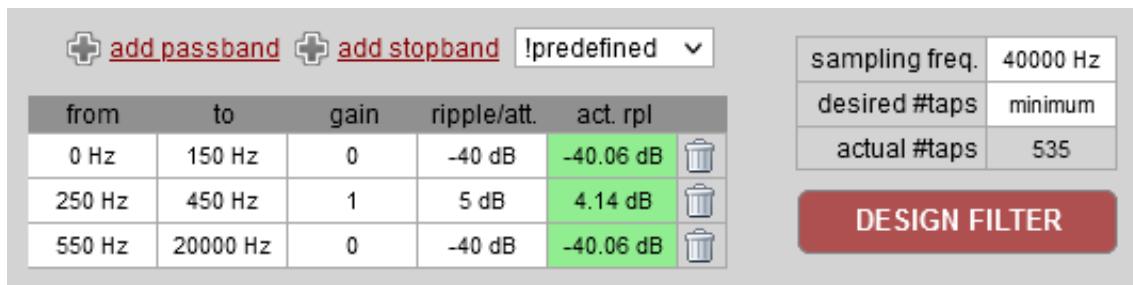


FIGURE 27 – Contrôles de l’interface de TFfilter

Et voilà, une fois les paramètres entrés, le code que TFfilter nous génère :

The screenshot shows the generated C++ code for a `SampleFilter`. On the left, configuration settings are shown:

- C code generator**
- Please note that the generated code is faster if the number of the taps is a power of 2.*
- Name of filter: Sample
- Number format: Integer
- Fixed point precision of taps: 16
- C type of coefficients: `int16_t`
- C type of accumulator: `int32_t`
- Unroll convolution loop (checkbox checked)

Sample C++ code

```

void SampleFilter_init(SampleFilter* f) {
    int i;
    for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
        f->history[i] = 0;
    f->last_index = 0;
}

void SampleFilter_put(SampleFilter* f, int16_t input) {
    f->history[f->last_index++] = input;
    if(f->last_index == SAMPLEFILTER_TAP_NUM)
        f->last_index = 0;
}

int16_t SampleFilter_get(SampleFilter* f) {
    int32_t acc = 0;
    int index = f->last_index, i;
    for(i = 0; i < SAMPLEFILTER_TAP_NUM; ++i) {
        index = index != 0 ? index-1 : SAMPLEFILTER_TAP_NUM-1;
        acc += (int32_t)f->history[index] * filter_taps[i];
    }
    return acc >> 16;
}

```

SampleFilter.h Select all

```

#ifndef SAMPLEFILTER_H_
#define SAMPLEFILTER_H_

/*
FIR filter designed with
http://t-filter.appspot.com

sampling frequency: 40000 Hz

fixed point precision: 16 bits

* 0 Hz - 150 Hz
gain = 0
desired attenuation = -40 dB
actual attenuation = n/a

* 250 Hz - 450 Hz
gain = 1
desired ripple = 5 dB
actual ripple = n/a

* 550 Hz - 20000 Hz
gain = 0
desired attenuation = -40 dB
actual attenuation = n/a

```

FIGURE 28 – Code généré par TFfilter

On peut voir que le nom des types de variable peut être modifié, ce qui nous permet de les mettre directement dans la norme utilisée à l’ES.

Problème de lecture de l'ADC

Avec l'impossibilité d'utiliser la librairie dsp.h, j'ai dû changer pour la version filtre à seulement quelques jours de la fin du projet, et je n'ai donc pas pu finir de mettre en place tout cela. C'est donc au moment de tester ce filtre que je me suis rendu compte que la lecture ADC de plusieurs série de valeurs dans le buffer ne fonctionnait pas. Seules les huit premières valeurs étaient prises, les suivantes restent nulles :

The screenshot shows the MPLAB Harmony Configurator interface. At the top, there is a code editor window displaying assembly code. The code is part of a switch statement for the case APP_STATE_SERVICE_TASKS. A specific line of code, 'appData.state = APP_STATE_WAIT;', is highlighted in green. Below the code editor is a memory dump window titled 'Variables'. It lists an array named 'history[0..99]' of type '_int16_t'. The array contains 100 entries, indexed from 0 to 99. The first 8 entries have non-zero values (0x0122, 0x0122, 0x0126, 0x0128, 0x0122, 0x0122, 0x0128, 0x0122), while the remaining 92 entries are zeroed out.

Name	Type	Address	Value
history[0..99]	_int16_t	0xA0000224	...
history[0]	_int16_t	0xA0000224	0x0122
history[1]	_int16_t	0xA0000226	0x0122
history[2]	_int16_t	0xA0000228	0x0126
history[3]	_int16_t	0xA000022A	0x0128
history[4]	_int16_t	0xA000022C	0x0122
history[5]	_int16_t	0xA000022E	0x0122
history[6]	_int16_t	0xA0000230	0x0128
history[7]	_int16_t	0xA0000232	0x0122
history[8]	_int16_t	0xA0000234	0x0000
history[9]	_int16_t	0xA0000236	0x0000

FIGURE 29 – Code généré par TFilter

Mode d'emploi

Pour utiliser ce projet, il suffit de brancher une source audio via Jack 6.35mm, Jack 3.5mm ou BNC, mais avec une seule de ces trois entrées à la fois, et le spectre du signal audio sera représenté à l'aide de la matrice à Leds, ou, si elle n'est pas branchée, des Leds présentent directement sur la carte. Un câble peut également être branché sur un des ports de sorties selon le même format que les entrées pour que le signal puisse être acheminé vers un ampli audio par exemple. Permettant au système de se brancher en série avec un système audio et d'y ajouter une information visuelle.

État du projet

(Les points rapportés ici font partie du fichier de modification présent en annexe.)

Le projet n'a malheureusement pas abouti dans la partie software, la partie hardware est maintenant bonne, même si quelques petites modifications peuvent être apportées pour la prochaine version du projet.

1. Après recalcults, les valeurs de R5, R7, R9, R10, R11 et C12 ont été changées sur le projet (voir rapport final), sur le PCB, R5, R7, R9 et R10 ont été changés, il faudrait changer également R11 et C12, ainsi que modifier les valeurs dans le fichier Altium.
2. La polarité du connecteur J1 est inversée sur le PCB actuel, il faudrait l'inverser dans le fichier Altium.

Pour la partie hardware, cela nécessitera peut-être plus de modification que cela, mais voici les premières modifications a apporté pour faire fonctionner le projet :

1. Le software actuel ne lit que huit valeurs analogiques au lieu de remplir le tableau complet, il faudrait modifier le code de l'interruption pour que le tableau soit complètement rempli.
2. Lorsque le tableau complet de valeurs analogiques sera rempli, il faudra tester si le filtre numérique implémenté fonctionne, et si c'est le cas, utiliser le site TFfilter pour implémenter les huit filtres passe-bandes selon le cahier des charges.

La partie communication avec la matrice à Leds n'a pas pu être implémentée non plus.

Il serait possible pour une nouvelle version du hardware de préférer un microcontrôleur de la famille MZ, la librairie dsp.h est disponible pour cette famille, et cela permettrait donc d'implémenter la transformée de Fourier.

Conclusion

Le projet n'a malheureusement pas abouti aussi bien que je l'espérais, la partie hardware comporte quelques soucis mineurs, mais surtout, la partie software ne fonctionne pas encore.

Malgré l'état d'avancement, je suis contente d'avoir pu travailler sur ce projet, la musique et l'audio sont des domaines qui me passionnent, et pouvoir travailler dans ces domaines avec de l'électronique m'a énormément plu.

J'ai pu apprendre énormément de choses durant ce projet, mais surtout que j'ai tendance à trop me fixer sur une seule voie, j'ai en effet perdu beaucoup de temps à vouloir utiliser la transformée de Fourier alors que cela s'est avéré impossible avec le microcontrôleur que j'avais choisi. Même si les indications de Microchip me laissait entendre que c'était possible, j'aurais dû envisager plus tôt la possibilité de changer de voie pour le projet et de faire du filtrage.

Webographie

Sites internets

- [1] Nom du site : Vidéo à l'origine du projet
<https://www.youtube.com/watch?v=GtKIkkLkrwU>
- [2] Nom du site : TFilter
<http://t-filter.engineerjs.com/>
- [3] Nom du site : DSP Library for PIC32
<https://www.microchip.com/SWLibraryWeb/product.aspx?product=DSP%20Library%20for%20PIC32>

Datasheetographie

Liens des Datasheets

- [1] PIC32MX470F512H-120/PT :
<http://ww1.microchip.com/downloads/en/devicedoc/60001185g.pdf>
- [2] MAX1793 :
<https://www.micro-semiconductor.sg/datasheet/d6-MAX1793EUE25.pdf>
- [3] MCP6002-E/SN_ND :
<https://ww1.microchip.com/downloads/en/DeviceDoc/MCP6001-1R-1U-2-4-1-MHz-Low-Power-Op-Amp-DS20001733L.pdf>
- [4] ECS-100-18-5PX-TR :
<https://ecsxtal.com/store/pdf/csm-7x.pdf>
- [5] TL431IDBZR :
https://www.ti.com/lit/ds/symlink/lm92.pdf?ts=1682410289518&ref_url=https%253A%252F%252Fwww.google.com%252F
- [6] LTST-C191KFKT :
<https://optoelectronics.liteon.com/upload/download/DS22-2000-222/LTST-C191KFKT.pdf>
- [7] RAPC712X :
https://www.switchcraft.com/assets/1/24/rapc712x_cd.pdf?6293
- [8] SJ-3523-SMT :
<https://www.cuidevices.com/product/resource/sj-352x-smt.pdf>
- [9] CONBNC002 :
<https://linxtechnologies.com/wp/wp-content/uploads/conbnc002-ds.pdf>
- [10] SN37A12A :
<https://datasheet.ciiva.com/pdfs/VipMasterIC/IC/SWIT/SWIT-S-A0001226488/SWIT-S-A0001226488-1.pdf?src=supplier=IHS+Markit>

Journal de travail

Semaine 1 (21 au 25 novembre)

Premier jet du cahier des charges, premières discussions et questions avec Mr. Bovey autour du projet. Premières recherches pour faire du filtrage numérique. Création d'un premier schéma bloc général du système.

Semaine 2 (28 novembre au 2 décembre)

Validation du cahier des charges, suite des recherches sur le filtrage numérique et premiers tests de scripts sur Octave GNU. Après plusieurs discussions avec Mr. Bovey, Moreno et Castoldi, Mr. Castoldi m'a conseillé d'envisager de faire une série de Fourier plutôt que plusieurs filtres passe-bande pour alléger la charge de travail du microcontrôleur. Création d'une première ébauche de l'architecture du firmware.

Semaine 3 (5 au 9 décembre)

Suite des recherches sur le filtrage numérique, amélioration du script avec la solution avec série de Fourier. Création du schéma bloc hardware. Rédaction de la pré-étude du rapport.

Semaine 4 (12 au 16 décembre)

Préparation de la présentation de la pré-étude, puis passage pour celle-ci. Reprise de la pré-étude suite aux remarques faites pendant la présentation.

Semaine 5 (19 au 23 décembre)

Reprise des tests de filtrage et transformées de Fourier, mais cette fois sur le microcontrôleur. Tests de la librairie microchip.

Semaine 6 (9 au 13 janvier)

Suite des tests, en parallèle, choix des composants pour le design.

Semaine 7 (16 au 20 janvier)

Suite du choix des composants, début du design lui-même. Rédaction de rapport de design.

Semaine 8 (23 au 27 janvier)

Finalisation du choix des composants et du design, fin de la rédaction du rapport de design.

Semaine 9 (30 janvier au 3 février)

Début du design du PCB, premiers placements des composants.

Semaine 10 (6 au 10 février)

Suite du placement des composants.

Semaine 11 (20 au 24 février)

Suite et fin du placement des composants. Début du rooting.

Semaine 12 (27 février au 3 mars)

Rooting des parties alimentations de la carte.

Semaine 13 (6 au 10 mars)

Rooting de la partie microcontrôleur de la carte.

Semaine 14 (13 au 17 mars)

Finalisation du rooting, commande du PCB et des composants

Semaine 15 (20 au 24 mars)

Début de conception de la partie software sans le microcontrôleur.

Semaine 16 (27 au 31 mars)

Suite de la conception software.

Semaine 17 (3 au 7 avril)

Suite de la conception software, tests sur le kit ES.

Semaine 18 (24 au 28 avril)

Réception des commandes, début du montage de la carte.

Semaine 19 (1 au 5 mai)

Suite du montage de la carte. Tests de la partie alimentation de la carte et de la communication avec le microcontrôleur.

Semaine 20 (8 au 12 mai)

Après réception des composants manquants, fin du montage et début de la conception software.

Semaine 21 (15 au 19 mai)

Suite de la conception software.

Semaine 22 (22 au 26 mai)

Suite de la conception software.

Semaine 23 (29 mai au 2 juin)

Modification de la partie hardware liée à l'amplification du signal

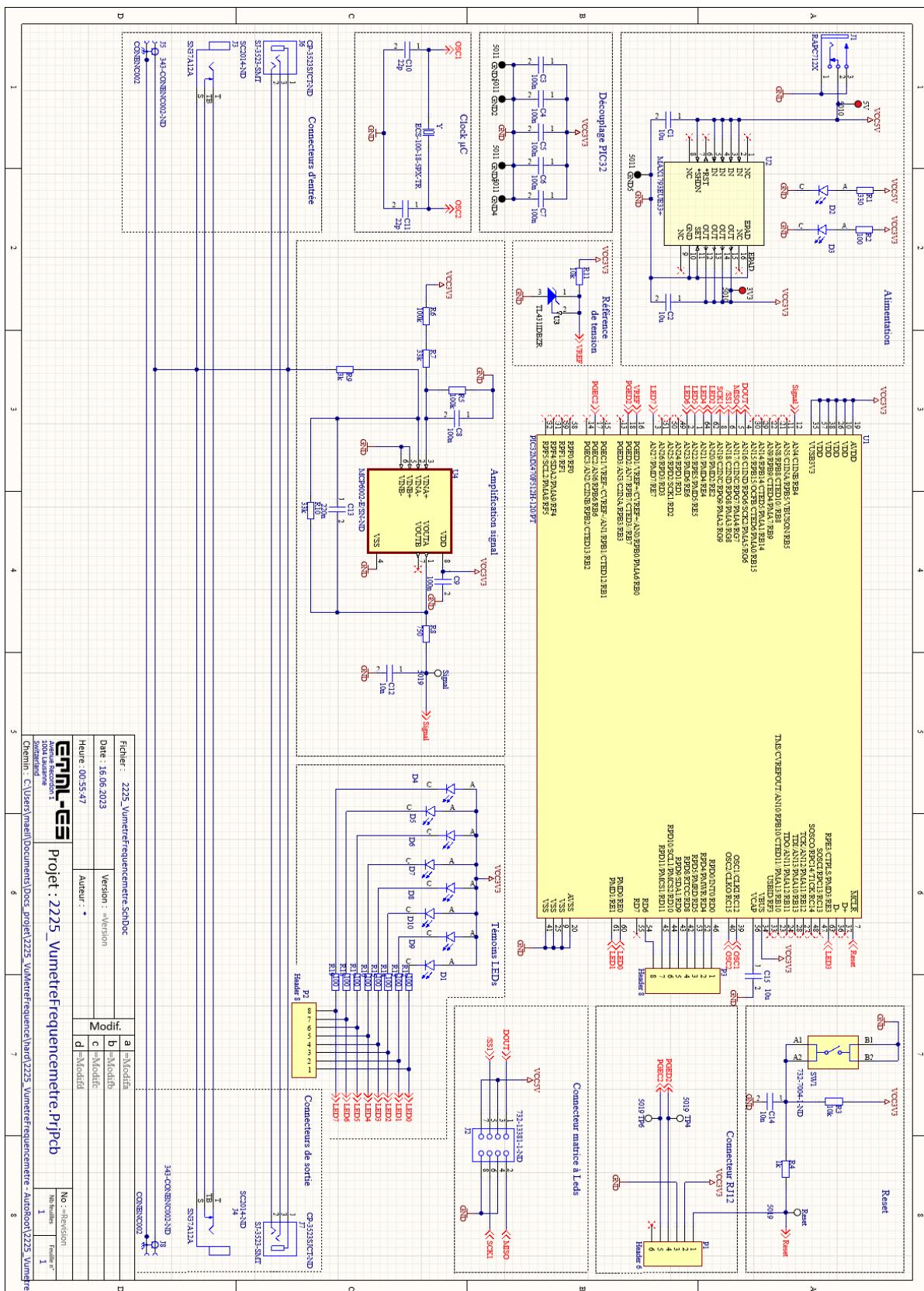
Semaine 24 (5 au 9 juin)

Suite des modification hardware.

Semaine 25 (12 au 16 juin)

Changement de direction pour le software lié à l'impossibilité d'utiliser la librairie dsp.h.
Début de conception d'un filtre basse-bande. Rédaction du rapport.

Schéma



Annexes

Code Octave GNU (pré-étude)

```
1 close all;
2 clear all;
3 clc;
4 pkg load signal;
5
6 l = 1000; % Longueur du signal
7
8 % Signal forme d'une addition de plusieurs sinus de fréquences différentes
9 signal = sinetone(25,1,1,1)+sinetone(50,1,1,2)+sinetone(100,1,1,3)+ ...
10      sinetone(24,1,1,5)+sinetone(49,1,1,2)+sinetone(103,1,1,1)+ ...
11      sinetone(32,1,1,4)+sinetone(53,1,1,1)+sinetone(97,1,1,2);
12
13 % Affichage du signal
14 plot(signal);
15
16 % Transformée de Fourier sur ce signal et affichage
17 signal_fft = fft(signal)(1:l/2+1);
18 figure;
19 plot(abs(signal_fft));
20
21 % Affichage dans la console des fréquences trouvées par la fft
22 for i = 1:length(signal_fft)
23     if(abs(signal_fft(i)) > 1)
24         printf(" %d ", (i - 1));
25     endif
26 endfor
```

Listing de app.c

```
1
2
3 #include "app.h"
4 #include "SampleFilter.h"
5
6 APP_DATA appData;
7
8
9 SampleFilter samples;
10 bool samplesFull = false;
11
12 void APP_Initialize ( void )
13 {
14     /* Place the App state machine in its initial state. */
15     appData.state = APP_STATE_INIT;
16
17     /* TODO: Initialize your application's state machine and
18      * other
19      * parameters.
20      */
21
22
23 /*****
```

24 Function:
25 void APP_Tasks (void)
26
27 Remarks:
28 See prototype in app.h.
29 */
30
31 void APP_Tasks (void)
32 {
33 // Variables locales
34 int16_t resultat_filtre;
35
36 /* Check the application's current state. */
37 switch (appData.state)
38 {
39 //
40
41 /* Application's initial state. */
42 case APP_STATE_INIT:
43 {
44 // Eteindre toutes les leds
45 LED0On();

```

45         LED1On() ;
46         LED2On() ;
47         LED3On() ;
48         LED4On() ;
49         LED5On() ;
50         LED6On() ;
51         LED7On() ;

52         // Initialisation de la structure des samples
53         SampleFilter_init(&samples) ;

55         // Active les timers
56         DRV_TMR0_Start() ;
57         DRV_TMR1_Start() ;

59         // Active l'ADC
60         DRV_ADC_Open() ;
61         DRV_ADC_Start() ;

63         appData.state = APP_STATE_WAIT;
64     }
66 //////////////////////////////////////////////////////////////////

68     case APP_STATE_SERVICE_TASKS:
69     {
70         // On fait le filtrage
71         resultat_filtre = SampleFilter_get(&samples) ;

72         appData.state = APP_STATE_WAIT;
73
74         break;
75     }
77 //////////////////////////////////////////////////////////////////

79     case APP_STATE_WAIT:
80     {
81         // On reste dans cet etat tant que le tableau d'
82         // n'est pas plein
83         if (samplesFull == true)
84         {
85             appData.state = APP_STATE_SERVICE_TASKS;
86             samplesFull = false;
87         }
88
89         break;
90     }

```

```
91
92     /* TODO: implement your application state machine.*/
93
94
95     /* The default state should never be executed. */
96     default :
97     {
98         /* TODO: Handle error in application's state machine.
99             */
100        break;
101    }
102 }
```

Listing de system_interrupt.c

```

1 #include "system/common/sys_common.h"
2 #include "app.h"
3 #include "system_definitions.h"
4 #include "stdint.h"
5
6 void __ISR(_ADC_VECTOR, ipl3AUTO) _IntHandlerDrvAdc(void)
7 {
8     uint8_t i;
9     static int8_t j = 0;
10    uint8_t offset = 0;
11    ADC_RESULT_BUF_STATUS BufStatus;
12
13    BufStatus = PLIB_ADC_ResultBufferStatusGet(ADC_ID_1);
14    if (BufStatus == ADC_FILLING_BUF_0TO7)
15    {
16        offset = 8;
17    }
18
19    for (i = 0; i < 8; i++)
20    {
21        SampleFilter_put(&samples, (int16_t)DRV_ADC_SamplesRead(i
22                                +offset));
23    }
24
25    j += 8;
26
27    if (j >= SAMPLEFILTER_TAP_NUM)
28    {
29        j = 0;
30        samplesFull = true;
31    }
32
33    /* Clear ADC Interrupt Flag */
34    PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_ADC_1);
35
36 // timer 1 pour cadencement APP
37 void __ISR(_TIMER_1_VECTOR, ipl1AUTO) IntHandlerDrvTmrInstance0(
38     void)
39 {
40     PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_1);
41 }
42
43 // timer 3 pour starter ADC. ISR non utilisee
44 void __ISR(_TIMER_3_VECTOR, ipl0AUTO) IntHandlerDrvTmrInstance1(
45     void)
46 {

```

```
45      PLIB_INT_SourceFlagClear(INT_ID_0, INT_SOURCE_TIMER_3) ;  
46  
47      //PLIB_PORTS_PinToggle(PORTS_ID_0, PORT_CHANNEL_C,  
        PORTS_BIT_POS_4) ;  
48      LED0Toggle() ;  
49  }
```

Listing de SampleFilter.c

```
1 #include "SampleFilter.h"
2 #include "app.h"
3
4 static int16_t filter_taps [SAMPLEFILTER_TAP_NUM] = {};
5
6 void SampleFilter_init(SampleFilter* f)
7 {
8     // Variables locales
9     int i;
10
11    for (i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
12    {
13        f->history[i] = 0;
14    }
15
16    f->last_index = 0;
17 }
18
19 void SampleFilter_put(SampleFilter* f, int16_t input)
20 {
21     f->history[f->last_index++] = input;
22
23     if (f->last_index == SAMPLEFILTER_TAP_NUM)
24     {
25         f->last_index = 0;
26     }
27 }
28
29 int16_t SampleFilter_get(SampleFilter* f)
30 {
31     // Variables locales
32     int32_t acc = 0;
33
34     int index = f->last_index, i;
35
36     for (i = 0; i < SAMPLEFILTER_TAP_NUM; ++i)
37     {
38         index = index != 0 ? index-1 : SAMPLEFILTER_TAP_NUM-1;
39         acc += (int32_t)f->history[index] * filter_taps[i];
40     }
41
42     return acc >> 16;
43 }
```

Listing de SamplFilter.h

```

1 #ifndef SAMPLEFILTER_H_
2 #define SAMPLEFILTER_H_
3
4 /*
5
6 FIR filter designed with
7 http://t-filter.appspot.com
8
9 sampling frequency: 40000 Hz
10
11 fixed point precision: 16 bits
12
13 * 0 Hz - 150 Hz
14     gain = 0
15     desired attenuation = -40 dB
16     actual attenuation = n/a
17
18 * 250 Hz - 450 Hz
19     gain = 1
20     desired ripple = 5 dB
21     actual ripple = n/a
22
23 * 550 Hz - 20000 Hz
24     gain = 0
25     desired attenuation = -40 dB
26     actual attenuation = n/a
27
28 */
29
30 #include <stdint.h>
31
32 #define SAMPLEFILTER_TAP_NUM 535
33
34 typedef struct {
35     int16_t history[SAMPLEFILTER_TAP_NUM]; // SAMPLEFILTER_TAP_NUM
36     unsigned int last_index;
37 } SampleFilter;
38
39 void SampleFilter_init(SampleFilter* f);
40 void SampleFilter_put(SampleFilter* f, int16_t input);
41 int16_t SampleFilter_get(SampleFilter* f);
42
43 #endif

```

Fichier MOD

Projet ETML-ES – Modification

PROJET:	2225 VuMetreFrequence		
Entreprise/Client:	ETML-ES	Département:	SLO
Demandé par (Prénom, Nom):	Philippe Bovey	Date:	16.06.2023
Objet (No ou réf, pièce, PCB...)			
Version à modifier:	01		

Auteur (ETML-ES):	Maëlle Clerc	Filière:	SLO
Nouvelle version:	02	Date:	16.06.20023

1 Description ou justification

Modification 1 :

Après recalculs, les valeurs de R5, R7, R9, R10, R11 et C12 ont été changées sur le projet (voir rapport final), sur le PCB, R5, R7, R9 et R10 ont été changé, il faudrait changer également R11 et C12, ainsi que modifier les valeurs dans le fichier Altium.

Modification 2

La polarité du connecteur J1 est inversée sur le PCB actuel, il faudrait l'inverser dans le fichier Altium.

Modification 3

Le software actuel ne lit que 8 valeurs analogique au lieu de remplir le tableau complet, il faudrait modifier le code de l'interruption pour que le tableau soit complètement rempli.

Modification 4

Lorsque le tableau complet de valeurs analogiques sera remplis, il faudra tester si le filtre numérique implémenté fonctionne, et si c'est le cas, utiliser le site TFilter pour implémenter les huit filtres passe-bandes selon le cahier des charges.

2 Référence conception

Indiquer ici le(s) dossier(s) de conception de référence et emplacement. (N/A pour entreprises)

3 Détail des modifications

Chaque rangée du tableau ci-dessous contient le détail d'une seule modification.

Exemples:

- 1 / Changer tous les boîtiers de résistances 0805 en 0603 / OK / JMO
- 2 / Remplacement U4 - TL074 par LM124 / NOK / SCA

#	Description	Fait	Approuvé
1	Dans Altium, remplacer les valeurs de R9 par 10K, R10 par 33K, R7 par 62K, R5 par 10K, R11 par 390 et C12 par 220pF	NOK	
2	Inverser la polarité de J1 et refaire le rooting pour ces changements	NOK	
3	Modifier le code de l'interruption de la lecture de l'ADC pour que les valeurs suivantes soient prises	NOK	
4	Après tests du passe-bande déjà codé, utiliser TFilter pour générer les huit bandes de filtre	NOK	
5			
6			
7			
8			

4 Remarques

Au besoin, indiquer ici des détails nécessaires à la compréhension, ainsi que les raisons d'une modification non effectuée ou reportée.

Exemple: Le point 2 (marqué NOK), est reporté pour une prochaine version pour épuiser notre stock de composants. Cette modif n'est pas critique fonctionnellement.

5 Convention de nommage et liens

Le nom de ce fichier doit être unique et doit donc contenir le numéro du projet et un numéro consécutif de modification avec le format suivant :

aaii_MOD_nn.docx

ou

NomProjet_MOD_nn.docx

avec :

- MOD : pour modification
- aaii : numéro de projet, exemple **1708** pour projet de 2017 no 08
- NomProjet : Si le projet n'est pas numéroté ou mandat de client.
- nn : numéro de modification. La première est 01

Exemples :

- **1708_MOD_01.docx** 1ere modification pour le projet 1708
- **1708_MOD_02.docx** 2e modification pour le projet 1708
- **CapteurVolets_MOD_01.docx** Cas de projet externe

Le schéma et/ou les documents de production de la pièce ou du PCB se référeront à ce document dans les cartouches.

Si un nouveau projet reprend un design d'un autre projet, créer un document de **modification numéro 00**. Ainsi, on pourra décrire les modifications initiales dans le fichier.

Exemple :

- **1803_MOD_00.docx** Modification initiale pour le nouveau projet 1803 à partir d'un autre projet (par ex. 1708)

5.1 Stockage du fichier

Ce fichier sera stocké à la racine du dossier **/doc** d'un projet.

Ainsi, tous les fichiers de modifications des pièces ou PCBs faisant partie du projet sont centralisés dans le même répertoire. La numérotation devient implicite.

Résumé du projet

RESUMÉ - Projet

Maëlle Clerc
SLO2
2022-2023

Titre:

2225 VumètreFréquence

Contexte et objectifs:

Il s'agit de concevoir un vumètre pour différentes gammes de fréquence, un analyseur de spectre visuel de 20 à 20kHz (filtrage pour les gammes suivantes: 20 - 50 - 100 - 200 - 500 - 1k - 2k - 10k - 20k) avec visualisation de l'amplitude. Le microcontrôleur est à choix selon ce qui conviendra le mieux. L'entrée du système viendra d'un système audio grand public, ou d'un générateur de signal. L'interface visuelle est à choix entre des leds (éventuellement reprendre le projet de matrice à leds 2126), une interface en C\#, voire une communication USB.

Résultats obtenus et conclusion :

La partie hardware a pu être faite et nécessite quelques courtes modifications.

La partie software n'a pas abouti, la lecture complète du signal la période d'échantillonnage ne fonctionne pas, et le filtre passe-bande n'a pas pu être testé.

Maître(s) de projet:
Entreprise mandataire:

Serge Castoldi et Juan José Moreno
ETML-ES

Affiche du projet

Proposition : Il s'agit de concevoir un vumètre pour différentes gammes de fréquence, un analyseur de spectre visuel de 20 à 20kHz (filtrage pour les gammes suivantes : 20 - 50 - 100 - 200 - 500 - 1k - 2k - 10k - 20k) avec visualisation de l'amplitude. Le microcontrôleur est à choix selon ce qui conviendra le mieux. L'entrée du système viendra d'un système audio grand public, ou d'un générateur de signal. L'interface visuelle est à choix entre des leds (éventuellement reprendre le projet de matrice à leds 2126), une interface en C#, voire une communication USB.



Voilà le type de résultat visuel qui a inspiré le projet

Chaque colonne représente une gamme de fréquence, la hauteur jusqu'à laquelle cette colonne s'allume représente l'intensité de la présence de cette gamme de fréquence dans le spectre sonore.



Voici le PCB que j'ai pu concevoir.

Les leds visibles en haut de la carte servent d'affichage condensé a cas ou la carte matrice à leds n'est pas branchée.

Les six connecteurs en bas de la carte servent pour l'entrée et la sortie du signal sonore. Trois types de connectiques sont disponibles : Jack 6.35mm, Jack 3.5mm et BNC.

La carte est « true bypass », c'est-à-dire que le signal sortant de la carte n'est absolument pas modifié durant le passage. Le traitement du signal se fait après le son.

Le connecteur en haut à droite de la carte n'est pas utilisé, il est là en cas de modifications futures du projet.

La partie hardware a pu être faite et nécessite quelques courtes modifications.

La partie software n'a pas abouti, la lecture complète du signal la période d'échantillonnage ne fonctionne pas, et le filtre passe-bande n'a pas pu être testé.