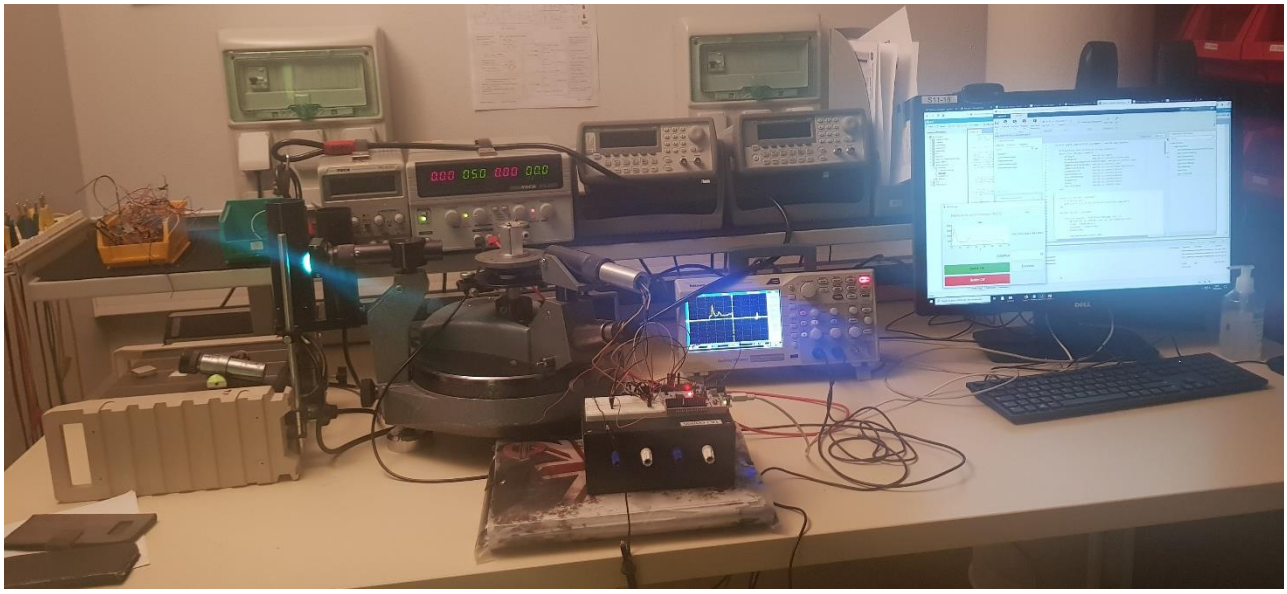
INSTITUT d'OPTIQUE GRADUATE SCHOOL ParisTech | LEnsE Laboratoire d'Enseignement Expérimental



RAPPORT TECHNIQUE

Spectromètre

Projet d'Electronique de Première Année à l'Institut d'Optique

L'objectif de ce projet est de réaliser un spectromètre tel que le spectre en sortie d'une source s'affiche sur l'écran de l'ordinateur

Eline CAVADORE, Lauriane GUGNOT, Marie-Hélène CARRON

Introduction et présentation du dispositif

L'objectif de ce projet était de réaliser un spectromètre en utilisant un goniomètre connecté à un dispositif embarqué permettant d'afficher une image du spectre de la source éclairant le goniomètre directement à l'écran. En particulier, le système conçu puis réalisé devait répondre aux exigences suivantes :

- Capter la lumière émise par la source via une barrette CCD constituée de 64 photodiodes
- Par l'intermédiaire d'une carte Nucléo, prendre 64 mesures de façon cyclique et synchronisée pour que chaque photodiode reçoive une information utile et que mises bout à bout, les informations de chacune de ces 64 diodes reconstituent le spectre de la source (ici une lampe à vapeur de mercure)

- Relier le sous-ensemble {goniomètre+barrette} à l'ordinateur
- Permettre l'acquisition du spectre via une interface graphique sur Matlab
- Afficher le spectre de la source

Si le principe de ce dispositif paraît simple, il répond à certaines contraintes, non seulement sur les plans matériel et optique, mais aussi et surtout sur le plan informatique (avec notamment deux langages utilisés à savoir le C et Matlab).
En effet, il faut dans un premier temps s'assurer de la bonne captation de la lumière de la source (donc de l'information), puis réaliser 64 mesures de façon synchronisée afin que chaque photodiode reçoive une et une seule information sur un cycle. Ensuite, la liaison avec l'ordinateur doit permettre l'acquisition des données mesurées mais aussi l'activation du cycle de mesure, l'ensemble étant géré par une interface qu'il faut adapter au dispositif. Enfin, après avoir réalisé plusieurs cycles, moyenné les résultats fournis par la barrette, il reste encore à les interpréter afin de retranscrire l'information sous forme de spectre.

Ce sont toutefois ces difficultés ainsi que l'intéressant mélange entre optique, électronique et informatique et notre goût marqué pour l'optique ondulatoire qui ont motivé ce choix de sujet.

A) Découpage fonctionnel

Voici un schéma fonctionnel explicitant les éléments principaux du système, les fonctions *Correction* et *Asservissement* étant des propositions d'optimisation du dispositif que nous n'avons pas pu réaliser :

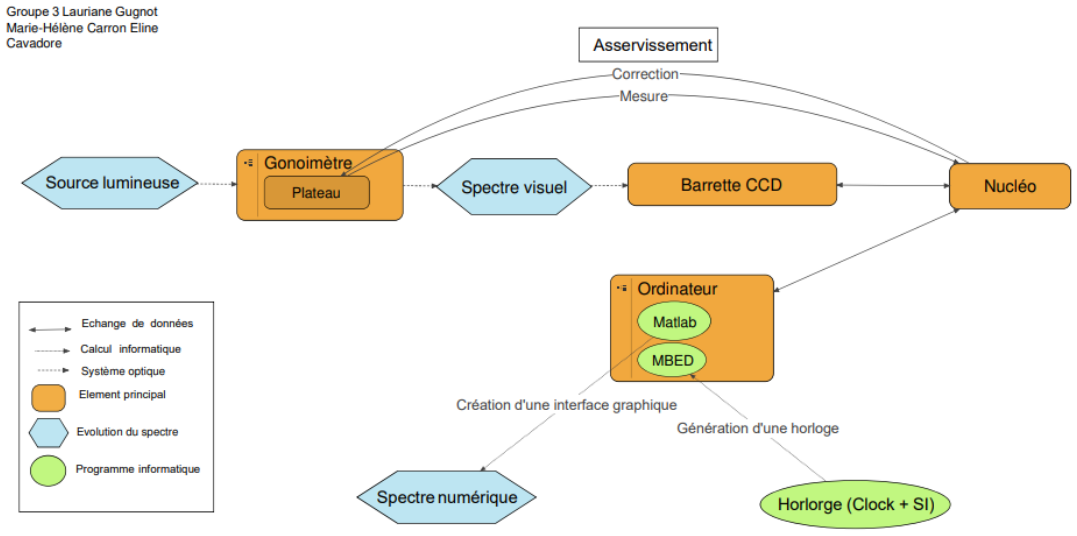
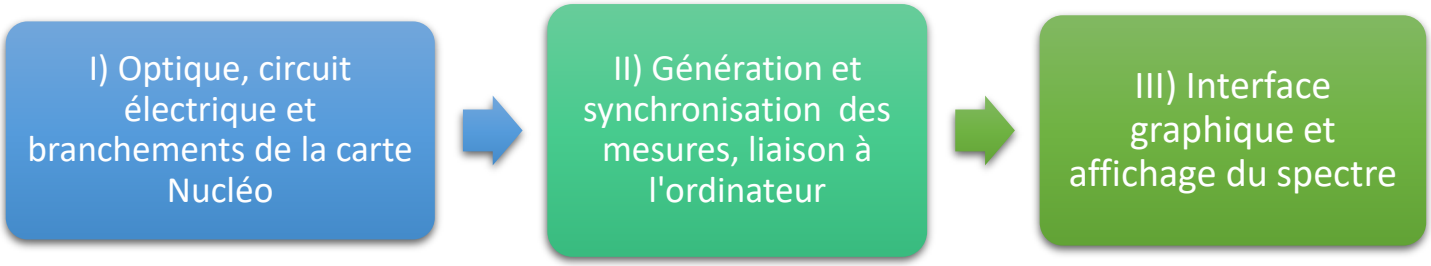
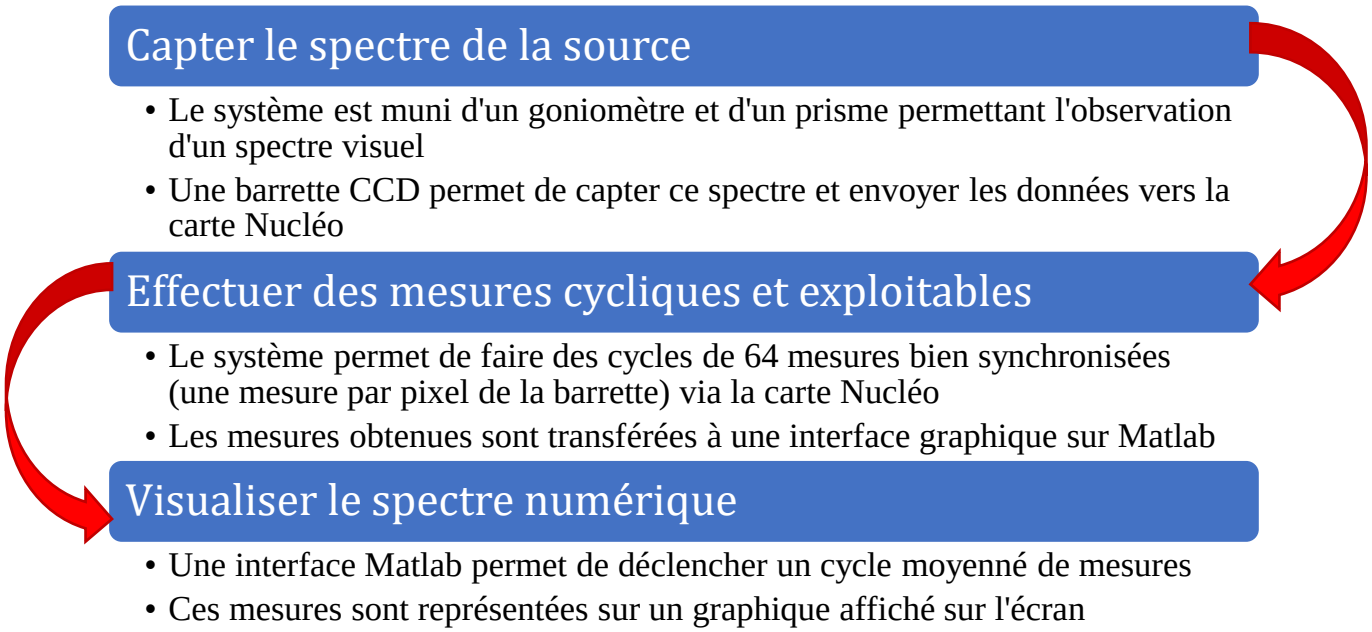


Schéma fonctionnel de notre spectromètre embarqué

Nous avons donc structuré notre projet en trois grandes parties présentées ci-dessous et détaillées dans les pages suivantes :



Plus précisément, le système permet les fonctionnalités suivantes :

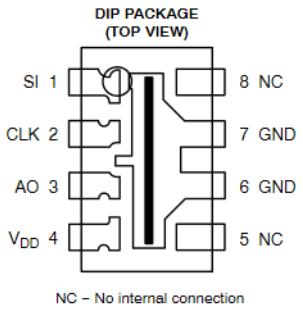


B) Réalisation, tests et validation

I) Optique, circuit électrique et branchements de la carte Nucléo

Pour commencer ce projet, nous nous sommes intéressées au fonctionnement d'une barrette CCD. En effet, ce composant permet de capter le spectre d'émission d'une source lumineuse (ici une lampe à Mercure).

La barrette optique que nous avons utilisée est une barrette CCD TSL201R-LF, dont le schéma est donné **Figure B.I.1.** C'est un dispositif à transfert de charges (en français DTC) équipé de 64 pixels, des éléments photosensibles avec lesquels les photons interagissent tels que la charge électrique soit directement proportionnelle à l'éclairement du détecteur. La géométrie linéaire de la barrette est particulièrement adaptée à l'étude d'un spectre.
Elle est alimentée avec une tension de 5V sur V_{DD}, on récupère ensuite le signal de sortie sur la broche AO.



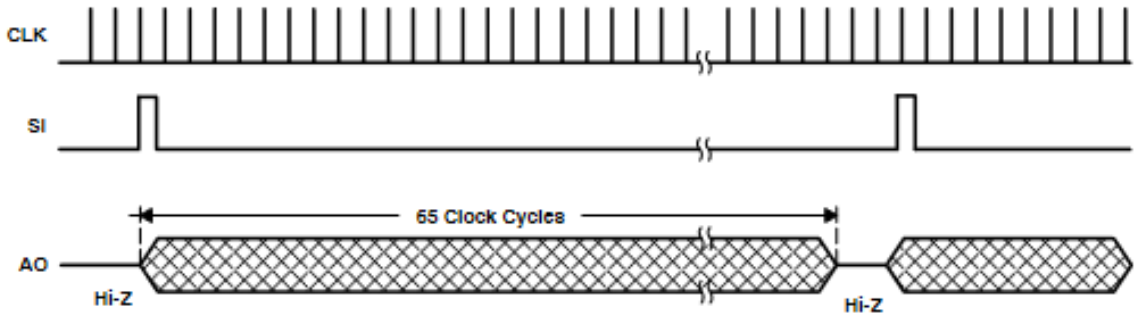
https://datasheet.octopart.com/TSL201R-LF-TAOS-datasheet-8327114.pdf

Figure B.I.1 : Schéma technique de la barrette

Le fonctionnement de cette barrette nécessite l'utilisation de deux horloges. En effet, pour assurer le transfert des charges accumulées dans la zone photosensible vers les registres de lecture, une première horloge enclenche le cycle des 64 mesures. Cette horloge est appelée **SI** (serial-input), elle correspond donc à un « top départ », à chaque impulsion démarre un nouveau cycle.

Une deuxième horloge appelée la **Clock** (**CLK** sur le schéma) assure le transfert séquentiel des charges provenant de chacun des photo-éléments sur un circuit de sortie muni d'une diode qui effectue la conversion charge-tension. Ce signal d'horloge permet donc de calibrer la durée d'une séquence pour chaque pixel et assure le passage d'un pixel à l'autre.

La synchronisation de ces deux horloges est présentée sur la **Figure B.I.2** extraite de la datasheet du composant.



https://datasheet.octopart.com/TSL201R-LF-TAOS-datasheet-8327114.pdf

Figure B.I.2 : Synchronisation des deux horloges et du signal de sortie

Ces deux horloges ont d'abord été générées à l'aide de deux GBF (**Figure B.I.3**), ce qui nous a permis de nous familiariser avec la synchronisation des signaux. Pour ce faire, nous avons alimenté la barrette avec une tension de 5V et avons utilisé un condensateur de 0.1 μF ainsi qu'une résistance de 320 Ω.

La période de la **Clock** a été prise à 100 μs et celle de la **SI** à 12.5 ms.

Résultats expérimentaux :

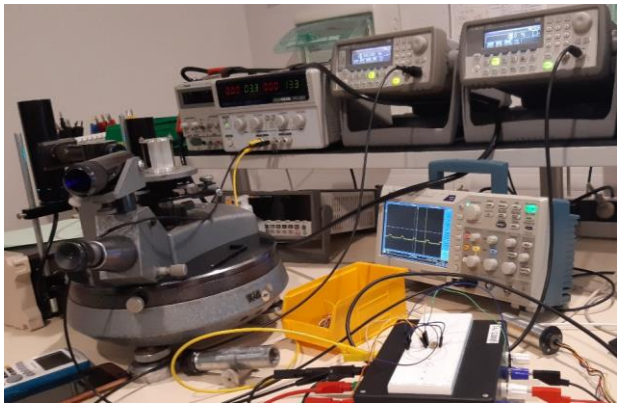


Figure B.I.3 : Génération des horloges avec deux GBF

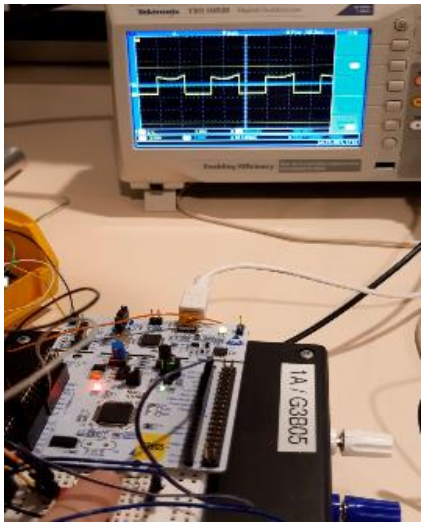


Figure B.I.4 : Validation de la synchronisation

On remarque sur la **Figure B.I.4** que la valeur de tension de sortie chute lorsque la barrette est mise dans l'obscurité (ici on a en particulier caché une partie de celle-ci en appuyant dessus).

Il s'agit ensuite de capter le spectre d'émission de notre source lumineuse, ici une lampe à vapeur de Mercure, grâce à la barrette. Nous avons utilisé un montage dispersif, i.e. un prisme placé sur un goniomètre.

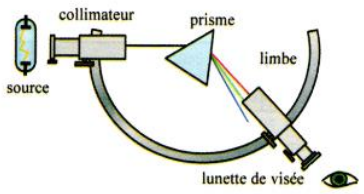


Figure B.I.5 : Schéma du montage dispersif

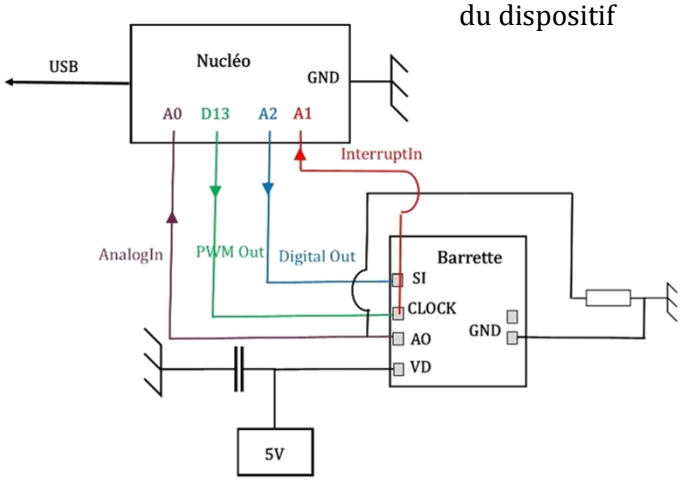
Le prisme permet d'étaler le spectre de notre lampe à vapeur de Mercure. De cette façon, chaque pixel de la barrette sera associé à la mesure d'intensité d'une longueur d'onde particulière. Contrairement au montage de la **Figure B.I.5**, une lunette de visée n'a pas été utilisée, la sortie de notre goniomètre étant constituée d'un tube dans lequel nous avons pu insérer la barrette CCD.

Le réglage du goniomètre s'est fait de manière approximative pour que la barrette capte le spectre visible **Figure B.I.6** de la manière la plus optimale possible.

Figure B.I.6 : Spectre visuel obtenu en sortie du goniomètre

Enfin, nous avons généré les horloges à l'aide d'un microcontrôleur, ici une carte NUCLEO. Le circuit électrique du dispositif est donné **Figure B.I.7**.

Figure B.I.7 : Circuit électrique du dispositif



La sortie de la barrette AO a été reliée à une entrée *AnalogIn* de la carte, tandis que pour les horloges, la **SI** concerne une sortie *DigitalOut* et la génération de la **Clock** a demandé l'utilisation d'un module PWM, qui génère des signaux créneaux à une fréquence que peut choisir l'utilisateur.

L'utilisation de la **SI** et de la **Clock** par la carte, les codes mis en place pour leur fonctionnement ainsi que l'intérêt du retour de la **Clock** vers la carte NUCLEO via une entrée *InterruptIn* de la carte seront décrits dans la partie suivante.

Résultats expérimentaux :

Les horloges ont ainsi été générées par la carte NUCLEO.

La **Figure B.I.8** présente le spectre obtenu sur l'oscilloscope via la barrette placée en sortie du goniomètre. En effet, l'oscilloscope relié à la sortie *AO* de la barrette affiche un signal sur lequel deux pics sont visibles, ils correspondent à deux raies du spectre d'émission de la lampe à vapeur de Mercure.

L'objectif suivant est alors l'obtention numérique de ce spectre via Matlab par une liaison-série avec l'ordinateur.

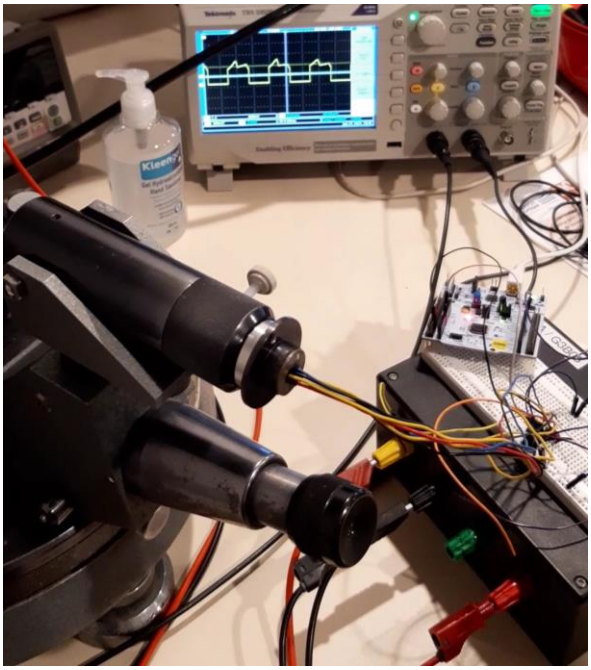


Figure B.I.7 : Validation de la génération des horloges par un microcontrôleur

II) Génération et synchronisation des mesures, liaison à l'ordinateur

Après avoir réglé le goniomètre pour que les raies d'émission arrivent bien sur la barrette CCD, il faut récupérer la valeur de l'intensité lumineuse frappant chaque pixel, donc récupérer la tension en sortie de chaque pixel, pour connaitre l'intensité lumineuse selon la fréquence comme expliqué **Partie I**.

Comme vu **Partie I**, il faut deux horloges pour récupérer la tension de sortie capteur par capteur : l'horloge CLOCK et l'horloge SI. Il faut une impulsion de la CLOCK pour mesurer la tension de sortie d'un capteur. Donc, lors d'une série de mesures, il faut 64 impulsions de la CLOCK et l'horloge SI, qui donne le départ des séries de mesures, doit se déclencher toutes les 64 impulsions de la CLOCK.

1) Génération des horloges avec des GBF

Avant de tester la synchronisation des horloges avec le programme MBED, nous avons généré les horloges avec deux GBF : un générant le signal de la CLOCK, un autre le signal de la SI. Ces tests permettent de vérifier si la valeur de fréquence donnée par le constructeur pour la CLOCK est la bonne ainsi que tester différentes valeurs de fréquence et de temps d'impulsion pour la SI.

Nous avons appliqué une CLOCK de fréquence 100µs avec un rapport cyclique de 0.5 et une SI d'impulsion de moins de 1µs. Le plus difficile était de trouver la bonne fréquence pour la SI, de telle sorte que 64 impulsions de la CLOCK se déroulent entre 2 impulsions de la SI, donc 157 Hz. Or cette valeur de fréquence est très limitée et il suffit qu'une petite désynchronisation survienne pour rater la 64ème impulsion. Il a donc fallu réduire la fréquence de la SI pour s'assurer qu'au moins 64 impulsions se déroulent entre deux impulsions de la SI. Avec une fréquence réduite, la barrette affichait bien une bonne tension de sortie.

- Test : Pour vérifier que les tensions de chaque capteur étaient bien envoyées à la suite les unes des autres, et non toutes en même temps, on cachait avec le doigt une partie des 64 capteurs de la barrette comme montré **Partie I.**

2) Génération des horloges avec MBED

Nous avons ensuite voulu générer ces horloges avec MBED. Nous avons dû choisir quel type de voie de sortie nous devions utiliser pour les horloges. Nous avons d'abord tenté de générer les 2 horloges avec 2 voies *PWMOut*.
Nous appliquons une fréquence de 100µs et un rapport cyclique de 0.5 pour la CLOCK et une fréquence de 100 Hz et un rapport cyclique de 0.0001 environ pour la SI. Mais MBED ne peut pas créer de signaux avec un rapport cyclique si petit.
Il a donc fallu augmenter le temps d'impulsion de la SI, mais nous devons maintenant faire attention aux éventuelles impulsions de CLOCK pouvant se dérouler durant l'impulsion de la SI. Nous avons augmenté l'impulsion à 60µs, la seule contrainte étant que l'impulsion ne soit pas plus grande qu'une période de la CLOCK.

- Test : Pour voir si les horloges fonctionnaient bien, nous visualisions la sortie de la barrette sur une voie de l'oscilloscope et les horloges sur la deuxième voie de l'oscilloscope grâce à une sonde.

3) Codage du programme mesurant la tension de sortie

Il a ensuite fallu coder la mesure de tension des 64 capteurs. Deux options s'offraient. Nous avons testé les deux options mais aucune ne mesurait des valeurs de tension cohérentes, ni même ne permettait d'afficher des tensions non nulles en sortie de barrette :

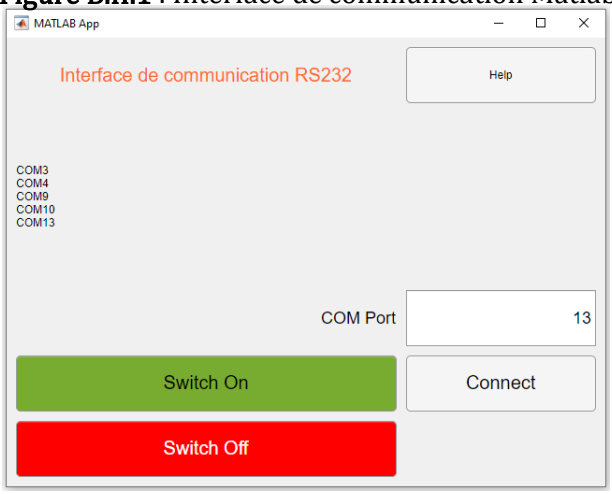
- Idée : On générait les horloges d'abord puis on mesurait les tensions après
  Problème : La première option impliquait d'utiliser deux détections de front montant pour compter les impulsions de CLOCK et celle de la SI. Mais la fonction de détection de front avait été faite à la main et non avec les voies Interrupt et la fonction nom_signal.fall(&fonction) que nous ne connaissions pas encore. Or la fonction de détection front montant/descendant s'exécutait trop lentement pour ne pas rater une impulsion de CLOCK et cela imposait aussi de reboucler les deux horloges.

- Idée : On générait les horloges et on mesurait les tensions en même temps (on génère une impulsion de CLOCK puis on mesure la tension juste après).
  Problème : La deuxième option n'est pas non plus possible, car la fonction lisant la tension prend du temps. Cette durée d'exécution varie selon la tension à lire et peut facilement dépasser les 20µs. Ainsi, on ne pouvait être assuré que la CLOCK soit périodique et de rapport cyclique 0.5.

- Solution : Nous avons donc trouvé un compromis : la CLOCK serait toujours générée en PWM et la lecture de la tension serait exécutée en parallèle sans modifier la CLOCK à chaque fois qu'un front descendant serait repéré grâce à un rebouclage du signal de sortie de la CLOCK sur une voie *InterruptIn* de la carte Nucléo et la fonction « entree_clock.fall(&detection_frontMontant) ». Cela est possible car la lecture et le stockage des valeurs de tensions dans un tableau s'exécutent en moins de 100µs. La SI serait générée à chaque fois qu'un compteur de front descendant de la CLOCK atteindrait 64 et serait envoyé avec une voie *DigitalOut*, et non *PWMOut*.
- Test : Pour vérifier que les fronts descendants étaient bien détectés et comptés, on utilisait un signal « dbg » qui changeait d'état (1 ou 0) à chaque entrée dans la fonction de détection et qu'on visualisait sur l'oscilloscope.

4) Codage du programme envoyant les données à Matlab

Figure B.II.1 : Interface de communication Matlab



L'étape suivante était de coder l'envoi des données à Matlab. Grâce à une interface Matlab représentée **Figure B.II.1**, on connectait la carte Nucléo à l'ordinateur et activait des cases, qui déclenchaient alors une action sur la Nucléo.
La case « Switch On » ou case 'a' activait la génération des horloges et autorisait l'envoi de données à Matlab, tandis que celle « Switch Off » ou 'e' arrêtait toutes les horloges.

On ne pouvait pas envoyer chaque donnée de tension en même temps que sa lecture, sinon le tout s'exécuterait en plus de 100µs et le prochain front de la CLOCK ne serait pas détecté et compté, ce qui décalerait toutes nos mesures d'un ou plusieurs capteurs.

Nous avons donc choisi d'envoyer les données de tension à Matlab une fois que toutes les séries de 64 mesures de tension sont finies. La difficulté est que les variables stockant les valeurs de tensions sur MBED ne sont pas reconnues sur Matlab. Il a dont fallu envoyer les données via 64 lignes de caractères ASCII, qui comportaient chacune le numéro du capteur et sa tension. Matlab récupérait les nombres inscrits dans cette ligne avec « *sscanf* ».

- Test : Avant de tester avec plusieurs séries de mesures, nous avions testé avec une seule série. Pour tester si Matlab recevait bien des données, nous utilisions une commande dans Matlab qui permettait de voir le nombre de bits/octets et les lignes de caractère reçus par Matlab. Nous avons eu beaucoup de difficultés sur cette partie-là car Matlab se stoppait dès la première donnée reçue et le code Matlab a même dû être revu par son créateur, Julien Villemejane. Nous avons été aussi confrontées à un problème de mémoire et nous avons dû réduire le nombre de bits affilié à chaque valeur de tension.

5) Rendu final du code (donné en **Annexe** via Code complet sur MBED)

Nous avons suivi le constructeur qui conseille une fréquence de 10kHz pour la CLOCK et un rapport cyclique de 0.5. On génère donc la CLOCK grâce à une voie *PWMOut*, après l'activation d'une case sur l'interface Matlab. Le programme crée en parallèle l'horloge SI via une voie *DigitalOut* et à l'aide de variables d'activation (qui prennent des valeurs 1 ou 0 et permettent une action si est égales à 1 ou 0).

De plus, on s'appuie sur des variables de comptage, qui comptent le nombre d'impulsions de la CLOCK (donc le nombre de capteurs dont on a pris la tension) et d'impulsions de la SI (donc le nombre de séries de 64 mesures réalisées). Pour compter le nombre d'impulsions de la CLOCK, il faut reboucler le signal de la CLOCK sur une voie *InterruptIn.*

Ainsi, on introduit deux variables de comptage :

- **j** qui compte le nombre d'impulsions de la CLOCK après le signal de la SI avec une fonction « entree_clock.fall » qui détecte les front descendants de la CLOCK du signal rebouclé « entree_clock »
- **compteur** qui compte le nombre d'impulsions de la SI lorsque que celle-ci est créée digitalement

On a aussi deux variables d'activation :

- **i** qui autorise la création du signal SI lorsque i=1
- **data_ok** qui autorise l'envoi de données à Matlab lorsque data_ok=0 et devient 1 lorsque les données ont toutes été envoyées

Le schéma en **Annexe** « Schéma résumant la synchronisation des différentes horloges pour une série » montre comment tous les signaux sont synchronisés entre eux sur une même échelle de temps et représente ce qu'on verrait en connectant les voies de la CLOCK, de la SI et celle de sortie à l'entrée d'un oscilloscope.

Les mesures de tensions sont stockées dans un tableau de 64 cases. Les mesures ne sont pas ajoutées à la suite les unes des autres dans le tableau par un « append » mais on initialise plutôt un tableau de 64 cases vides, puis on stocke la tension du j ^{ième} pixel dans la (j-1) ^{ième} case du tableau. Ce processus sert à éviter les décalages dans le tableau de mesures si une erreur de lecture ou de stockage survient pour un pixel. On moyenne la tension mesurée avec celles mesurées lors des séries précédentes en même temps que le stockage.

Après avoir réalisé toutes les séries de mesures (ici 10) et avoir moyenné toutes les tensions de chaque série, on transfère le tableau vers Matlab en envoyant à l'ordinateur 64 lignes de caractères. Chaque ligne comporte le numéro du pixel et sa valeur de tension moyennée (ici sur 10 séries). Matlab lit ensuite les lignes reçues et récupère les nombres présents sur cette ligne à l'aide d'un « *sscanf* ».

Le schéma en **Annexe** « Schéma fonctionnel du code MBED pour 10 séries de mesures » résume la chronologie, de la génération des horloges à la réception des mesures par Matlab.

III) Interface graphique et affichage du spectre, obtention du produit final

Le code MBED précédent a permis d'effectuer la mesure du spectre sur un ou plusieurs cycles et de stocker ces données dans un tableau transféré à Matlab.

Il faut ensuite que Matlab récupère ces données pour pouvoir en tracer le graphe. Pour cela, nous créons un tableau *TableauTension* de 64 cases, chacune devant stocker la mesure d'un pixel précis. A des fins de rigueur et de meilleure précision, nous ne considérons que les mesures d'un cycle moyenné (ici moyenné à partir de 10 cycles simples).

Nous remplissons le tableau avec les données acquises via la commande « *sscanf* » puis nous traçons le graphe correspondant aux entrées de ce tableau avec une abscisse comprise entre 1 et 64 (chaque unité d'abscisse correspondant à un pixel de la barrette).

L'ensemble de ces commandes est donné en **Annexe** via Code d'acquisition et de tracé du spectre (Matlab).

Compte-tenu du bon fonctionnement de la liaison avec MBED et du code MBED lui-même, la validation du code Matlab ne s'est faite que selon deux conditions :

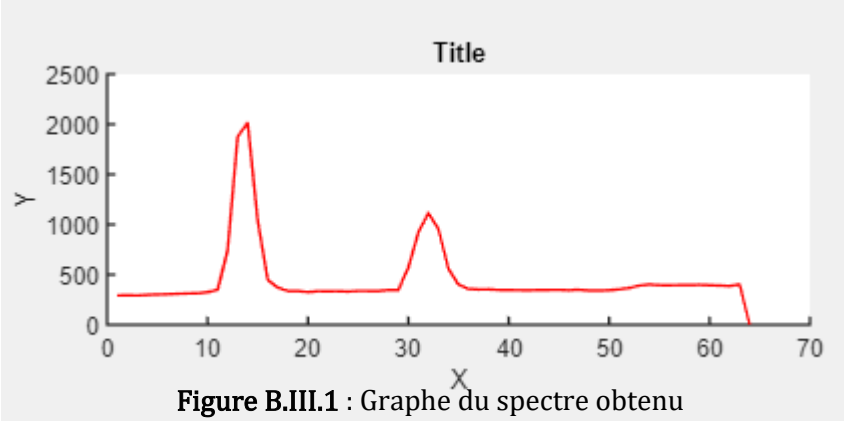


Figure B.III.1 : Graphe du spectre obtenu numériquement via le code Matlab décrit

-en premier lieu, obtenir le graphe du spectre (c'est-à-dire ne pas afficher une courbe sans point), comme ce qui est présenté en **Figure B.III.1**.

-en second lieu, obtenir une courbe en corrélation avec ce qu'affiche l'oscilloscope et dont la forme est modifiée lorsqu'on bouge le tube du goniomètre, signifiant alors bien que ce soit le spectre de la source qui est affiché.

Le résultat est donc satisfaisant et respecte le cahier des charges, lequel indique que nous devons pouvoir obtenir une image du spectre sur l'écran d'ordinateur. Toutefois, nous pourrions améliorer ce système, voici des pistes d'améliorations possibles.

Si nous avions eu un moteur à disposition, nous aurions pu asservir celui-ci afin de sélectionner de façon précise la zone du spectre visuel à mesurer, ce qui aurait permis une augmentation de la résolution du dispositif.

Une autre piste d'amélioration concernant l'image obtenue elle-même est donnée via un programme Python disponible en **Annexe** sous le nom Programme Python de tracé de spectre de raies. Celui-ci permettrait, après calibration via des sources parfaitement connues ou en connaissant exactement la déviation ou l'angle du prisme, de détecter les maxima locaux de la courbe mesurée et de leur affecter à chacun une raie. Un exemple de spectre tracé avec ce programme est donné par la **Figure B.III.2**.

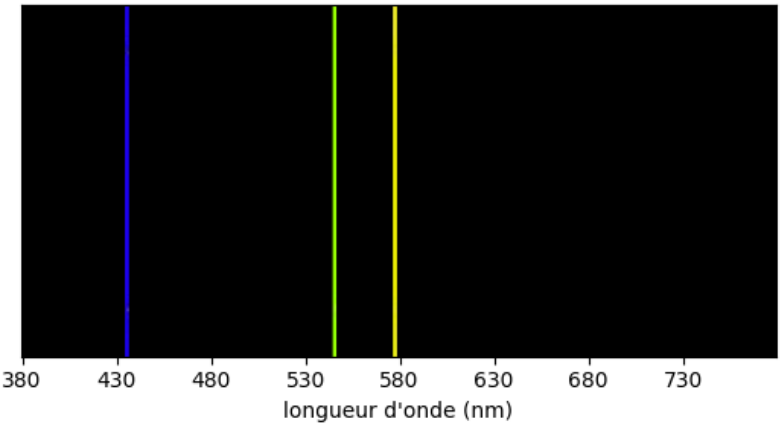


Figure B.III.2 : Spectre de raies du Mercure obtenu avec le programme Python décrit

In fine, nous aurions donc un spectre de raies tel que donné usuellement. En réalisant cette fonctionnalité sur Matlab et en créant un programme permettant de tracer cette figure immédiatement après les mesures, nous aurions obtenu un dispositif très largement optimisé.

****

Organisation du travail et analyse de la démarche projet

1) Planning détaillé

Séance 1 :

Le spectre est obtenu en sortie d'un goniomètre via un prisme et récupéré grâce à une barrette CCD composée de 64 photodiodes.

- Compréhension du fonctionnement de la barrette à l'aide des fiches constructeurs
- Génération des horloges à l'aide des GBF et synchronisation des signaux
- Réglages grossiers du goniomètre, câblage et structuration du circuit électrique
- Visualisation du signal de sortie de la barrette sur l'oscilloscope

Séance 2 : Le spectre numérique est ensuite généré grâce à une carte Nucléo L476RG.

- Fonctionnement de la barrette en générant les horloges numériquement avec la carte Nucléo via MBED
- Réglages du goniomètre pour obtenir le spectre sur une longueur de 1cm (longueur de la barrette)
- Visualisation du signal de sortie de la barrette sur l'oscilloscope

Séance 3 :

Le signal de sortie est ensuite traité pour afficher le spectre via Matlab par une liaison-série avec l'ordinateur.

- Reconstitution du montage pour retrouver le spectre en sortie du goniomètre avec une lentille cette fois
- Compréhension de l'interface graphique Matlab, réalisation d'un test sur "LED1" de la carte Nucléo

Séance 4 :

- Acquisition des 64 mesures de tension via Matlab
- Modification du code pour stocker sur Matlab les valeurs de tension acquises via MBED

Séance 5 : Le programme MBED est repensé de sorte à être fonctionnel avec l'interface Matlab.

- Remplacement du Ticker du signal SI par une commande qui déclenche une action automatiquement lorsqu'un front descendant est détecté
- Génération de la SI à l'activation d'une case de l'interface

Séance 6 :
Le signal de sortie est reçu par Matlab et le spectre en sortie du goniomètre est visualisé via l'interface graphique de Matlab.

- Vérification des actions commandées par l'interface et de la bonne acquisition des données par Matlab
- Mise en place d'une interface graphique

Séance 7 :
La résolution du signal est améliorée en réalisant une moyenne sur plusieurs acquisitions.

- Réception et analyse des spectres obtenus sur Matlab, corrélation entre les mesures de la barrette et le spectre de la source
- Moyennage sur une dizaine d'acquisition

2) Difficultés rencontrées

En premier lieu, le dispositif accueillant la barrette a rendu difficile la bonne récupération du spectre visuel en sortie du goniomètre car nous n'avions pas un accès facile à la distance focale de la lentille et donc à son plan focal image, plan qui aurait maximisé la quantité d'information reçue. Il nous a donc fallu procéder pas à pas à chaque fois de sorte à obtenir, soit à l'oscilloscope soit plus tard sur l'écran, une courbe dont les pics étaient facilement discernables.

En second lieu, nous avons dû changer de programme de génération d'horloges et de synchronisation de nombreuses fois avec des problèmes parfois issus d'un problème de version MBED, parfois d'origine inconnue et ne se résolvant pas en utilisant une méthode précise mais plutôt à tâtons, ce qui a pris beaucoup de temps. De plus, c'est aussi parce que nous avons procédé à tâtons que nous n'avons pas pu régler le problème immédiatement, chaque cas nous faisant revoir la plupart des programmes, aussi bien sur MBED que sur Matlab. Une méconnaissance de toutes les fonctionnalités de MBED (notamment la fonction de détection de fronts) ainsi qu'un problème au niveau du code Matlab n'étant pas de notre ressort ont ralenti notre progression.

L'absence de données sur la déviation du prisme associé au goniomètre a rendu difficile la corrélation entre mesures via la barrette et le spectre sous sa forme attendue (raies associées à une couleur). Nous avons cependant un code Python qui permettrait d'obtenir cette image sous forme de raies assez facilement.

3) Analyse du travail d'équipe

Au sein du groupe, nous avons eu une bonne ambiance de travail et nous avons su réfléchir ensemble lorsqu'un problème majeur se présentait. Nous avons su aussi nous entraider lorsqu'une difficulté se présentait à l'une d'entre nous. En effet, nous avons appris à rechercher l'origine d'un problème étapes par étapes car il pouvait être dû à des parties spécifiques du dispositif dont différentes personnes s'étaient occupées. Par exemple, il pouvait s'agir d'un problème optique, un problème de câblage ou un problème dû aux codes.

Cependant, nous avons eu du mal à nous répartir les tâches : au lieu de réfléchir longtemps sur un seul problème pour lequel seul le chargé de TP pouvait nous donner la réponse, nous aurions dû utiliser ce temps pour réaliser des tâches annexes ou ultérieures qui nous auraient permis d'avancer plus vite une fois le problème principal résolu. Nous avons aussi eu du mal à gérer une plateforme de partage pour mettre en commun tous nos travaux. Ainsi, il n'était pas rare que nous demandions à l'une ou l'autre un document ou fichier qu'une seule personne détenait.

****

Conclusion générale du projet

Ainsi, au terme de notre projet, nous avons réussi à obtenir numériquement le spectre d'émission de notre source lumineuse en faisant une moyenne du spectre sur dix acquisitions de mesures. Ce projet nous a permis de découvrir un nouveau composant, à savoir une barrette CCD et nous avons également acquis de nouvelles compétences dans la synchronisation de signaux ainsi que la configuration d'une liaison-série avec l'ordinateur de sorte à afficher notre spectre à l'aide d'une interface graphique sur Matlab que nous n'avions jamais utilisée auparavant.

Ce projet nous a particulièrement intéressées car il comportait à la fois une partie optique et une partie électronique. Les premières séances ont été agréables car le dispositif avançait rapidement, différents objectifs ont été atteints, à savoir la compréhension de la barrette CCD et la génération des horloges par la carte NUCLEO. Les séances qui ont suivies, et qui ont notamment concerné le débogage de codes, ont été un peu plus difficiles. Néanmoins, c'est avec une grande satisfaction que nous avons finalement obtenu notre spectre final.

Annexes

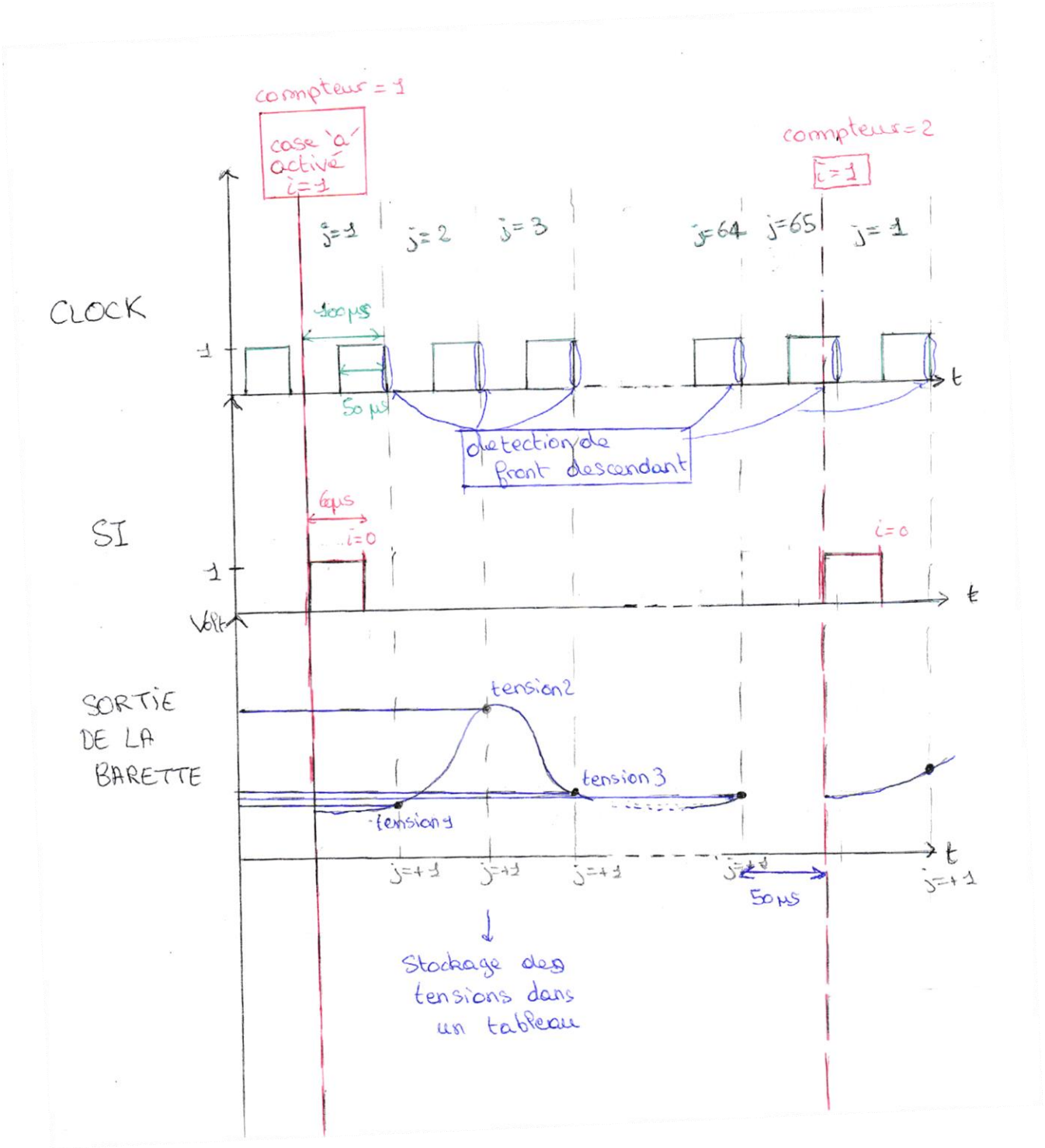


Schéma résumant la synchronisation des différentes horloges pour une série de mesures

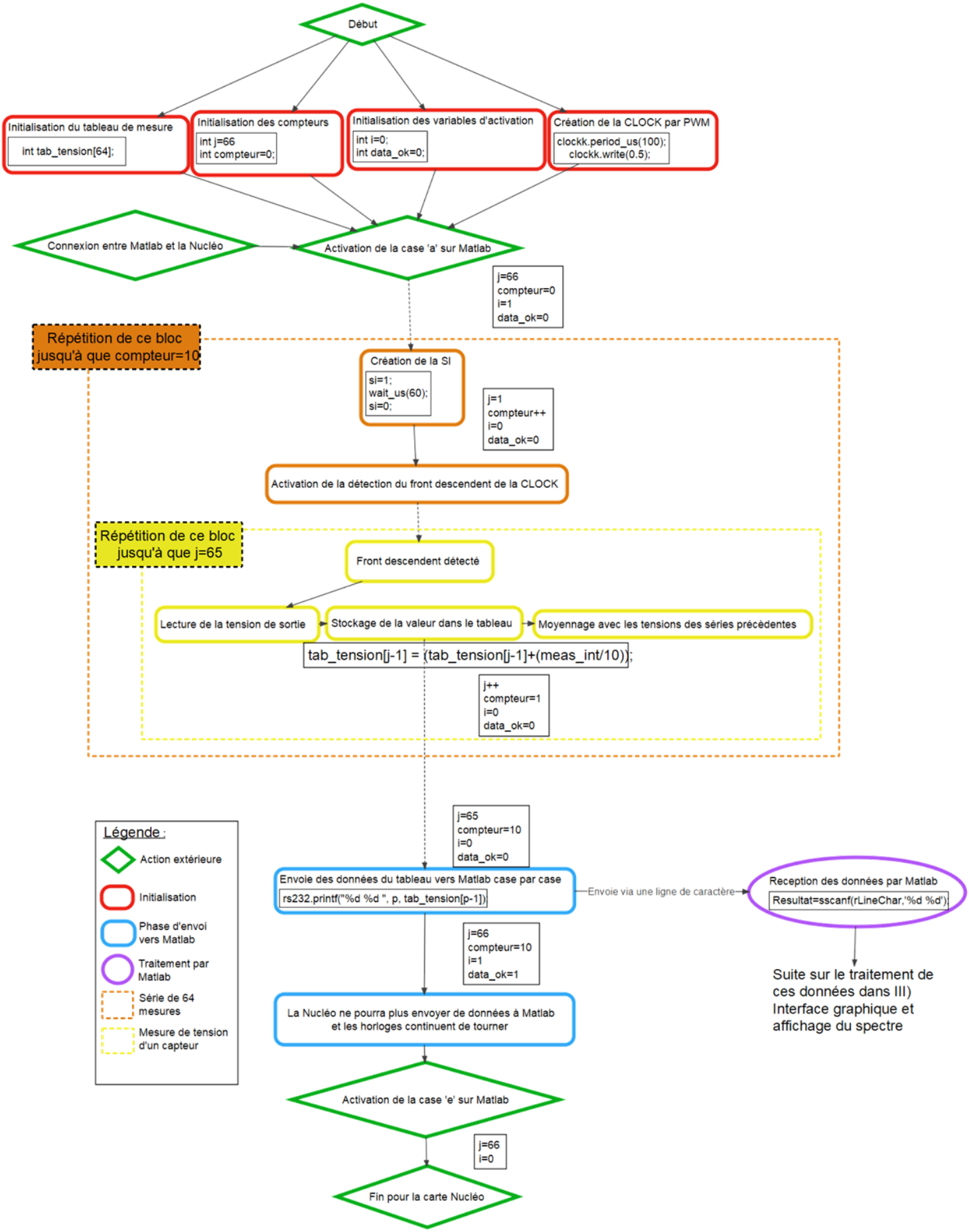


Schéma fonctionnel du code MBED pour 10 séries de mesures

Code complet sur MBED

```
/* mbed Microcontroller Library
 * Copyright (c) 2019 ARM Limited
 * SPDX-License-Identifier: Apache-2.0
 */

#include "mbed.h"

// Configurations des entrées et sorties
Serial      rs232(USBTX, USBRX);            // Déclaration de la liaison entre Matlab et la Nucléo
DigitalOut si(A2);                  // Déclaration de la voie où sera envoyé la SI
PwmOut clockk(D13);                     // Déclaration de la voie où sera envoyé la Clock
DigitalOut dbg(D10);                    // Déclaration de la voie oà sera récupéré le signal de débug
AnalogIn entree_analog(A0);             // Déclaration de la voie où sera réupéré la tension en sortie de barette
InterruptIn entree_clock(A1);               // Déclaration de la voie où sera rebouclé la clock



// Fonctions système:
void ISR_get_data(void);                // Fonction qui permet de éclencher une action quand on éxécute une commande sur Matlab
void detection_frontMontant(void);          // Fonction qui réalise une action à chaque détection de front montant


// Variables:
char data_received = 0;                 // Récupère des caractères envoyés depuis Matlab
int tab_tension[64];                    // Initialise le tableau de mesures de tension
int i=0;                          // Permet le démarrage de la SI
int j=66;                         // Compte le nombre de valeurs de tensions mesurés et de fronts montants détectés
                                // Renseigne alors à quel pixel correspond la valeur de tensions mesurés
int compteur=0;                       // Compte le nombre de série de 64 mesures réalisés (ici de 10)
int meas_int;                         // Variable intermédaire lors de la mesure de tension
int data_ok;                          // Indique lorsqu'il envoyé les mesures de tensions à Matlab



// Fonction principale
int main() {
    dbg=0;
    clockk.period_us(100);              // Crée le signal de clock PWM avec une période de 100µs
    clockk.write(0.5);                  // Crée le signal de clock PWM avec un raport cyclique de 0.5
    rs232.baud(115200);                 // Permet la connection avec l'ordinateur avec la bonne vitesse d'envoi de donnés
    rs232.attach(&ISR_get_data, Serial::RxIrq);     // Permet qu'une action sur Matlab est un impact sur la Nucléo via la
                                                    // fonction ISR_get_data
    entree_clock.fall(&detection_frontMontant);     // Fait appel à la fonction detection_frontMontant à chaque fois qu'un front
                                                    // descendant est détecté sur entrée_clock

    while (1) {

        if (i==1){                  // Se déclenche lorsque la case 'a' est activé ou qu'une série de mesures doit recommenser
                                    // cad lorsque j=65)
            si=1;                   // Crée le signal de la SI qui est une porte de durée 60 µs
            wait_us(60);
            si=0;
            i=0;                    // i est remis à 0 pour que d'autres signaux SI ne soient crées lors des 64 mesures
            j=1;                    // Permet d'activer l'action de mesurer la tension à chaque front descendant dans
                                    // detection_frontMontant
            compteur+=1;            // Indique qu'une série de relevée va être réalisé en plus


        }

        if (j==65){                         // Déclenche à nouveau un signal de SI en remettant i=1
            j=66;                           // Permet qu'aucune condition ne se réalise durant l'envoi des mesures
                                            // ou le renouvellement de la SI
            if(data_ok == 0){

               if (compteur==10){                   // Déclenche l'envoi des mesures de tensions déjà moyenné aprés toutes les séries de
                                                    // mesures réalisées
                    dbg =1;
                    for (int p = 1; p < 65; p++){           // Envoie ls valeurs de tensions moyenne correspondant à chaque
                                                            // pixel pixel par pixel
                        rs232.printf("%d %d ", p, tab_tension[p-1]);    // Les 2 information (tension et numéro du pixel) sont
                                                                        // contenau dans les %d

                     }
                    rs232.printf("\r\n");              // Indique à Matlab que toutes les données nécéssaires ont été envoyés
                    data_ok = 1;                    // Permet que plus aucune données ne soient envoyés à Matlab
                }
            }
            i=1;                            // Permet la création d'un nouveau signal SI en remplissant la condition du bloc if précédent

        }
        wait_us(50);                            // Permet d'attendre au moins 50us entre chaque série de mesures de tensions
    }
}
```

```
void ISR_get_data() {
    __disable_irq();
    data_received = rs232.getc();           // La Nucléo est prête à recevoir les données envoyées depuis Matlab
    switch(data_received){
        case 'a':                   // Si la case 'a' de l'interface Matlab est activé
            //rs232.printf("k\r\n");
            data_ok = 0;                // L'envoie des valeurs de tension par la Nucléo vers Matlab est autorisé
            i=1;                    // Le signal de la SI peut être envoyé et la synchronisation peut être effectué
            break;
        case 'e':                   // Si la case 'e' de l'interface Matlab est activé
            //rs232.printf("b\r\n");
            j=66;                   // j a une valeur de tel sorte que aucune condition soit rempli et tout se stoppe
            i=0;                    // i a une valeur de tel sorte que aucune condition soit rempli et tout se stoppe
            break;
        default:                    // Si aucune case de l'interface Matlab est activé
            //rs232.printf("d\r\n");
            __nop();                        // Rien ne se passe
    }
    __enable_irq();
}

void detection_frontMontant() {
    if((j>0)&&(j<65)){                          // Si j est entre 1 et 64 inclus et à chaque fois qu'un front descendant est
                                                // repéré sur entrée_clock

        meas_int = entree_analog.read_u16() >> 4;           // Mesure la tension du j ième pixel de la barrette sur 12 bits MSB
        tab_tension[j-1] = (tab_tension[j-1]+(meas_int/10));        // Stockage de cette mesure sur la j ième case du tableau de mesure
                                                                    // et moyenne en même temps sur 10 mesures
        j++;                                // j est agrémenté pour passer au pixel suivant

    }
}
```

Code d'acquisition et de tracé du spectre (Matlab)

```
    function results = rs232Interrupt(app, src, ~)
        %% Affiche les données reçues sur une interface texte
        rLine = readline(src);
        rLineChar = char(rLine);
        disp(rLine);

        TableauTension=zeros(1,64); % initialisation du tableau
        Resultat=sscanf(rLineChar,'%d %d');
        for k = 0:63
            TableauTension(Resultat(2*k+1))=Resultat(2*k+2);
        end;

        %results=TableauTension;

        app.PortListLabel.Text = rLine;

        %% Trace le graphe du spectre à partir du tableau précédent
        Abscisse=1:64;
        plot(app.UIAxes, Abscisse,TableauTension)

        drawnow;
        flush(app.s)
    end
end
```

Programme Python de tracé de spectre de raies

```
import numpy as np                  # Importe le module numpy
import matplotlib.pyplot as plt     # Importe le module matplotlib
from PIL import Image               # Importe le module Pillow


def spectre (tab_frequence):        # Crée une fonction spectre avec le tableau de fréquences en entrée

image =Image.open("Spectre_Lumière_blanche.jpg")      # Ouvre l'image du spectre de la lumière blanche qui fait 400 pixels de longueur
M=np.array(image)                   # Transforme l'image en matrice de hauteur 3
t=tab_frequence                     # Abrège "tab_fréquence" en "t"
n=len(t)                            # Récupère la longueur de ta_frequence
for p in range (n):                 # Boucle qui permet d'aligner la longueur 380 avec le premier pixel
t[p]=t[p]-380

if n=0:                             # Cas particulier pour à tableau à 0 fréquence
M=[:,:,:]=[0,0,0]                   # Rien est émis, le spectre est entièrement noir

if n=1:                             # Cas particulier pour à tableau à 1 fréquence
M[:,:(t[0]-1):1,:]=[0,0,0]          # Noircit toute les longueurs d'onde sauf celle
M[:,(t[0]+1)::1,:]=[0,0,0]          # émise par la lampe en laissant une colonne de pixel non noircie

if n=2:                             # Cas particulier pour à tableau à 2 fréquences
M[:,:(t[0]-1):1,:]=[0,0,0]          # Noircit toutes les longueurs d'onde jusqu'à la première
M[:,(t[0]+1):(t[1]-1):1,:]=[0,0,0]  # Noircit toutes les longueurs entre la première et la deuxième
M[:,(t[1]+1)::1,:]=[0,0,0]          # Noircit toutes les longueurs entre le dernière longueur et la fin

else:                               # Cas pour plus de fréquences
M[:,:(t[0]-1):1,:]=[0,0,0]          # Noircit toutes les longueurs d'onde jusqu'à la première
for k in range (0,n-1,1):           # Noircit toutes les longueurs entre la k ième et la k+1 ième
M[:,(t[k]+1):(t[k+1]-1):1,:]= [0,0,0]
M[:,(t[n-1]+1)::1,:]=[0,0,0]        # Noircit toutes les longueurs entre le dernière longueur et la fin

plt.imshow(M)                       # Commandes pour afficher le spectre final
axes = plt.gca()                    # Variable des axes par matplolib
axes.yaxis.set_visible(False)       # Supprime l'axe vertical
axes.set_xlabel('longeur donde')    # Donne un titre à l'axe horizontal
axes.xaxis.set_ticklabels(['0','380', '430', '480', '530', '580','630','680','730']) # Permet de graduer l'image sans la translater
plt.show()                          # Affiche le spectre final avec la bonne interface

return(M)                           # Retourne la matrice du spectre d'émission
```