Master 1 Physique fondamentale et applications

Rapport d'activités
Septembre-Décembre 2022

Génération d'impulsions pour manipulations d'initialisation de spin dans des boîtes quantiques d'InAs/GaAs

Corentin Morin

Institut des Nano-Sciences de Paris

Encadrante : Valia Voliotis

Table des matières

1 Introduction

Le but de ce stage fut la mise en place de l'électronique et de l'optique correspondant à la séquence d'impulsions lasers et TTL à envoyer pour réaliser une expérience d'initialisation de spin comme décrite dans [1].

2 Situation du problème et problématique

2.1 Le système

L'équipe PMTEQ travaille sur des boîtes quantiques "auto-organisées" d'InAs/GaAs. Celle-ci sont réalisée par des techniques de croissance [2] (voir figure 2.1.1.I/).

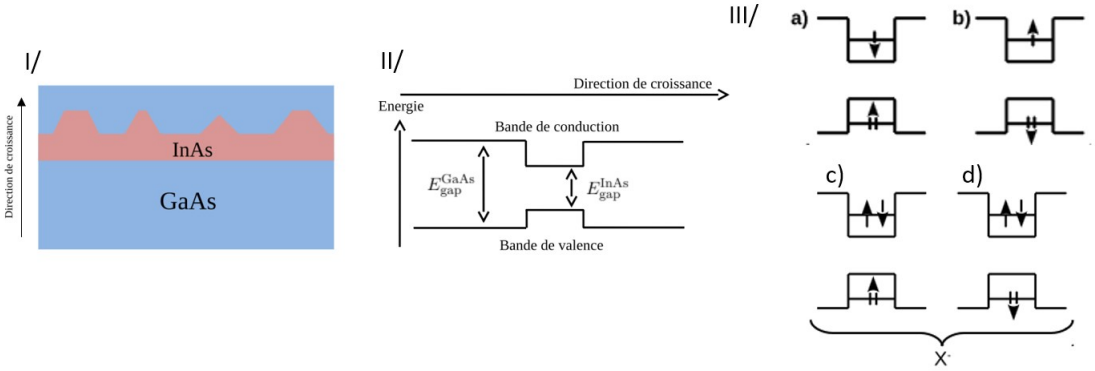

FIGURE 2.1.1 – I/ Représentation d'une tranche d'échantillon où des boîtes quantiques se sont auto-organisées, II/ Représentation schématique de la structure de bande de l'échantillon selon l'axe de croissance, III/ Description de différents états possibles dans une boîte quantique — Figures adaptées de [2]

Les matériaux, en l'occurrence InAs et GaAs ayant des énergies de gap électronique différentes(voir figure 2.1.1.II/), les zones où l'InAs s'est amalgamé forment des puits quantiques à 3 dimensions où les énergies des électrons (dans le bande de conduction) et des trous (dans la bande de valence) sont quantifiées. Il peut alors se former, après absorption d'énergie des complexes électron trou appelés excitons composés d'un électron et d'un trou piégés dans le puit quantique (voir figure 2.1.1.III/a-b)).

Afin de contrôler le niveau de charge initiale de la boîte quantique, il est possible de coupler les boîtes quantiques à une mer de Fermi en réalisant des dopages de certaines des parties de GaAs (voir 2.1.1.b)) [3]. L'ajout de contacts électriques permet alors de modifier le niveau de Fermi et donc de contrôler le taux d'électrons passant par effet tunnel dans la boite quantique.

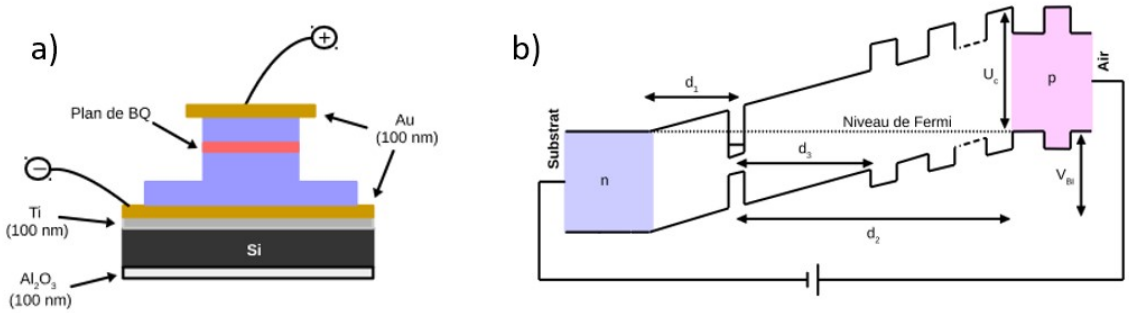

FIGURE 2.1.2 – a) Représentation des échantillons contactés, b) Représentation schématique de la structure de bande de l'échantillon contacté — Figures adaptées de [2]

Ainsi, des états chargés d'excitons sont possibles dans les boîtes quantiques. Nous ne nous intéresserons par la suite qu'aux quatres états de plus basse énergie des boîtes quantiques chargées avec un électron présentés en figure 2.1.3.

2.2 Manipulation de spin

L'application d'un champ magnétique aux boîtes quantiques permet de lever les dégénérescences des niveaux d'énergie présentés en figure 2.1.3. Les nouveaux niveaux d'énergie sont présentés en figure 2.2.1.

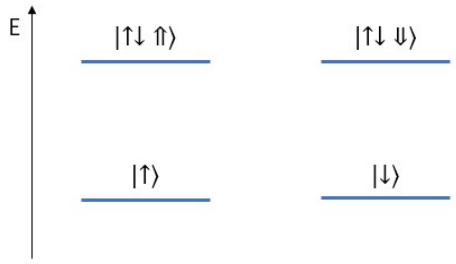

FIGURE 2.1.3 – États de plus basse énergie pour des puits quantiques chargés avec un électron

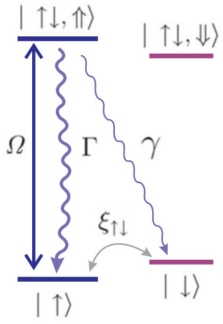

FIGURE 2.2.1 – Niveaux d'énergie d'une boîte quantique chargé d'un électron sous champ magnétique — Figure extraite de [1]

Les transitions $|\uparrow\rangle \leftrightarrow |\uparrow\downarrow\Uparrow\rangle$ et $|\downarrow\rangle \leftrightarrow |\uparrow\downarrow\Downarrow\rangle$ sont autorisées et actives optiquement. La transition $|\uparrow\downarrow\Uparrow\rangle \rightarrow |\uparrow\rangle$ a un taux d'émission spontanée Γ.La transition $|\uparrow\downarrow\Uparrow\rangle \rightarrow |\downarrow\rangle$ est normalement interdite pour des raisons de conservation de moment cinétique mais des effets de relaxation la rendent possible avec un taux γ. [1]

Avec un laser à la résonance de la transition $|\uparrow\rangle \leftrightarrow |\uparrow\downarrow\Uparrow\rangle$ et une tension spécifique, il est possible de complètement dépléter le niveau $|\uparrow\rangle$. Cela est vrai si la pulsation de Rabii de ce système, Ω est telle que $\Omega \gg \Gamma$. Suite à cela, en laissant de laser allumé, l'état $|\uparrow\downarrow\Uparrow\rangle$ se désexcitera vers l'état $|\downarrow\rangle$. Dans la littérature, cette transition s'effectue en quelques μs.

L'état $|\downarrow\rangle$ n'étant pas l'état de plus basse énergie, celui-ci va alors se désexciter vers l'état $|\uparrow\rangle$ avec un taux $\xi_{\uparrow\downarrow}$. C'est ce taux de désexcitation que nous souhaitons étudier. Typiquement, dans la littérature, ce temps de désexcitation est de l'ordre de quelques dizaines de ms [1].

Pour connaître la population d'électrons dans l'état $|\downarrow\rangle$, nous souhaitons sonder optiquement la transition $|\uparrow\downarrow\Downarrow\rangle \leftrightarrow |\downarrow\rangle$. Pour ce faire nous souhaitons avoir accès à l'intensité de la fluorescence émise lors de la désexcitation de l'état $|\uparrow\downarrow\Downarrow\rangle$ vers l'état $|\downarrow\rangle$ après un pompage optique de la transition $|\downarrow\rangle \rightarrow |\uparrow\downarrow\Downarrow\rangle$. L'intensité de la fluorescence est proportionnel à la population dans l'état $|\downarrow\rangle$. Cela nous permet donc d'étudier la dynamique temporelle de la désexcitation $|\downarrow\rangle \rightarrow |\uparrow\rangle$

Dans les boîtes quantiques étudiées par le groupe, la transition $|\uparrow\rangle \leftrightarrow |\uparrow\downarrow\Uparrow\rangle$ se situe à une longueur d'onde de l'ordre de 920 nm.

2.3 Problématique

Aux vues des différentes transitions présentées précédemment, afin d'étudier la population de l'état $|\downarrow\rangle$, il convient de réaliser plusieurs étapes.

- Pompage de l'état $|\uparrow\rangle$ vers l'état $|\downarrow\rangle$ ⇒ Illumination du QD par un laser à la résonance de la transition $|\uparrow\rangle \leftrightarrow |\uparrow\downarrow\Uparrow\rangle$.
- Envoi d'une impulsion laser à la résonance de la transition $|\downarrow\rangle \leftrightarrow |\uparrow\downarrow\Downarrow\rangle$ à un certain temps δ après la fin du pompage (afin de reconstruire une dynamique de la population de l'état $|\downarrow\rangle$)
- Étude de l'intensité de la fluorescence émise lors de la désexcitation de l'état $|\uparrow\downarrow\Downarrow\rangle$ à l'état $|\downarrow\rangle$ ⇒ Détermination temporelle de la population de l'état $|\downarrow\rangle$. Cela est donc associé à une fenêtre de mesure.

La fluorescence étant très faible, il convient de répéter ce cycle de nombreuses fois afin d'obtenir un signal moyenné exploitable. Ces cycles seront effectués à une fréquence de répétition f_{mes}

Le fait de créer une fenêtre de lecture de la fluorescence permet de ne pas solliciter les appareils de mesures alors que des signaux lumineux pouvant parasiter les données acquises sont actifs.

Une représentation schématique de l'enchainement de ces étapes est présentée en figure 2.3.1.

Ainsi, durant ce stage il a été question de mettre en place l'électronique et l'optique nécessaires à l'enchaînement de ces différentes étapes, et tout cela automatiquement.

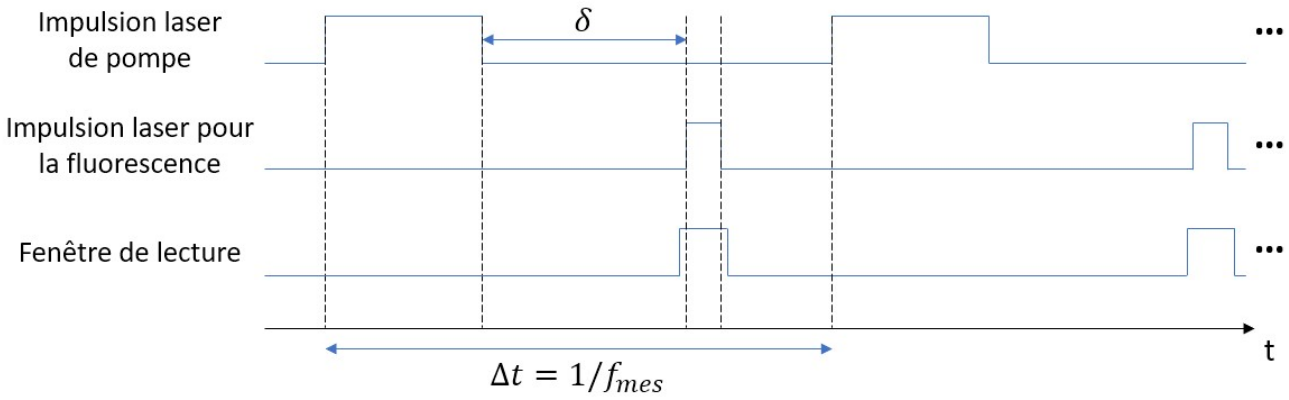

FIGURE 2.3.1 – Représentation schématique de l'enchaînement des étapes à effectuer lors d'un cycle de mesure

3 Matériel et architecture du programme informatique

3.1 Lasers

Les sources de lumière dont nous disposons pour ce projet sont deux lasers Ti :Sa accordables dans le proche infrarouge. Ils émettent de la lumière en continue, il convient donc de moduler l'intensité de cette lumière avec des éléments optiques.

3.2 Génération de l'impulsion de pompe

L'impulsion de pompe est une impulsion laser assez longue (de l'ordre de quelques μs). Nous avons donc choisi de la réaliser grâce à un AOM, puisque celui dont nous disposons possède un temps de montée de l'ordre de quelques dizaines de ns [4].

Un AOM est un dispositif composé d'un cristal relié à deux transducteurs piézoélectriques. Ceux-ci excitent le cristal à une fréquence fixe et créent des ondes de pression et donc des changement locaux d'indice optique. Cela crée alors un réseau de diffraction sinusoïdal (voir figure 3.2.1). Il est possible de démontrer que la répartition de l'intensité lumineuse dans les différents ordres de diffraction dépend de l'amplitude des ondes de pression et donc de l'amplitude de la tension électrique appliquée aux transducteur selon une fonction de Bessel. [5]. Il est alors possible de moduler l'intensité d'un laser continu grâce à un AOM.

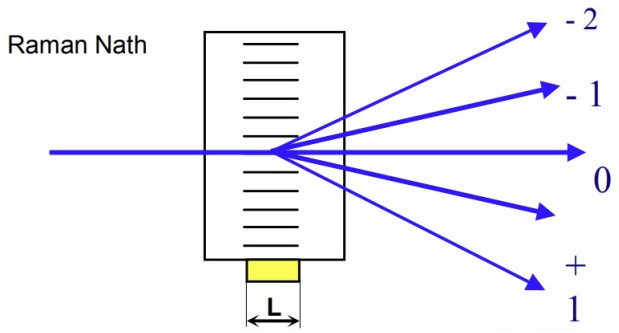

FIGURE 3.2.1 – Représentation schématique de la diffraction dans un AOM — Extrait de [4]

Les piézoélectriques de l'AOM sont excités par un driver RF modulant le signal de l'impulsion laser à envoyer par une sinusoïde à la fréquence propre du cristal. Si l'accord entre la fréquence de la sinusoïde et celle du cristal n'est pas parfait, il peu y avoir une perte de puissance optique et une mauvaise répartition de l'intensité dans les ordres de diffraction.

Dans la suite, tout ce qui concernera les impulsions créées pour modulation lumineuse par l'AOM sera en réalité une modulation appliquée au driver RF (voir figure 3.2.2). La modulation de ce driver se fait tel qu'un signal haut = LIGHT ON (1V), un signal bas = LIGHT OFF (0V)).

3.3 Génération de l'impulsion de sonde

La sonde requérant des impulsions plus courtes et plus rapide que la pompe et n'ayant pas d'autre AOM, nous avons choisi de réaliser l'impulsion de sonde grâce à un EOM.

FIGURE 3.2.2 – Schéma du bloc "AOM" avec un faisceau laser subissant une modulation temporelle en violet

Un EOM est un modulateur électro-optique. Ce dispositif se base sur l'effet électro-optic ou effet Pockels [6]. Cet effet traduit le changement d'indice d'un milieu (matériau spécifique comme le $LiNbO_3$) en fonction d'un champ électrique extérieur appliqué à celui-ci. Ainsi, on peut grâce à un beam splitter, faire passer une partie de l'intensité du laser dans un guide optique d'indice fixe et l'autre partie dans un guide optique d'indice au choix de l'utilisateur. L'interférence entre ces deux faisceaux donne alors lieu à des interférences constructives ou destructives en fonction de l'amplitude du champ électrique appliqué au milieu. Il s'agit d'une architecture de Mach-Zender (Voir figure 3.3.1).

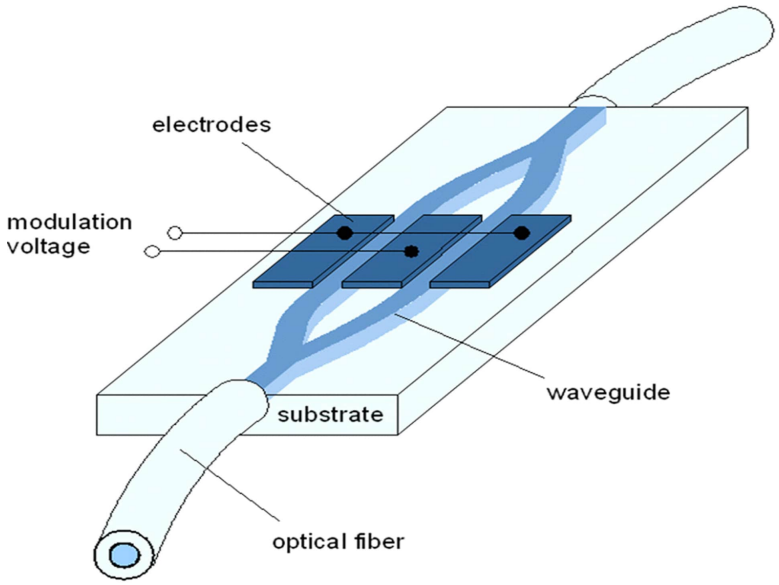

FIGURE 3.3.1 – Représentation schématique d'un dispositif EOM — Extrait de [6]

L'effet Pockels et les interférences nécessitent néanmoins un contrôle de la polarisation que l'on effectue grâce à des fibres à maintient de polarisation.

Le temps de montée d'un pulse d'EOM est de l'ordre de 200 ps, ainsi, le temps de montée effectif du système est surtout limité par le temps de montée de l'impulsion électronique de modulation.

Un contrôle du point de fonctionnement de l'EOM peut être réalisé grâce à l'application d'une tension contrôle par un contrôleur extérieur. L'intensité du faisceau passant à travers l'EOM peut être contrôlée grâce à l'entrée RF de l'EOM (voir figure 3.3.2).

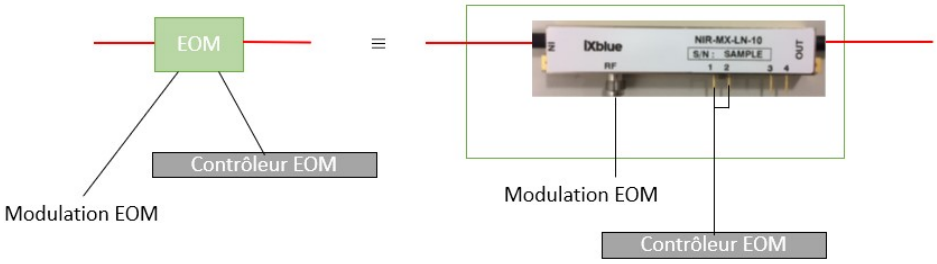

FIGURE 3.3.2 – Schéma du bloc "EOM" avec un faisceau laser subissant une modulation temporelle en rouge

3.4 Génération électronique des formes d'impulsion souhaitées

Des deux dernières parties, il ressort que nous avons besoin de fournir aux dispositifs optiques que sont l'AOM et l'EOM des impulsions électriques de la forme souhaitées. Nous disposons pour cela de deux générateurs (deux pour

pouvoir les positionner dans différentes pièces).

Le premier est le DG645 de chez Stanford Research. Il peut générer 4 signaux de délais et et longueurs différentes et possède une horloge de trig interne allant jusqu'à 10 Mhz. Les temps de montée sont de l'ordre de la nano-seconde. Le contrôle à distance de celui-ci peut se faire par connection éthernet.

Le second est le TOMBAK de chez Aérodiode. Les temps de montée sont de l'ordre de la nano-seconde. Il est possible de le synchroniser aux signaux lumineux grâce à une photo-diode intégrée. Le contrôle à distance de celui-ci se par connexion usb.

3.5 Mesure de la fluorescence

La photoluminescence émise lors de la recombinaison de l'exciton est très faible, et nous souhaitons pouvoir étudier celle-ci avec une très bonne résolution temporelle.

Nous disposons d'un module de comptage de photons de la marque PicoQuant, le PicoHarp 300. Combiné à celui-ci, nous possédons un APD (Amplified photo-detector) de la marque Excelitas. Le PicoHarp possède une résolution temporelle maximale de l'ordre de 4ps. La détection fonctionnne sur un principe de start-stop.

Ce principe repose sur la corrélation entre les signaux électriques sur les deux entrées du PicoHarp. Tout d'abord, le 'Channel 0' est un channel de synchronisation. Il s'agit d'un signal à la fréquence de l'excitation périodique envoyé au dispositif à étudier. Sur la 'Channel 1' est branché l'APD qui converti l'arrivée d'un photon en impulsion électrique. Il est conseillé de faire en sorte que seulement 1 à 5 % des cycles de mesures détectent un photon. [7]

Ainsi, afin de ne pas monopoliser l'électronique du module de comptage de photon à chaque cycle, une prise de mesure est déclenchée à l'arrivée d'un photon (voir figure 3.5.1.a)). C'est l'évènement de start, le PicoHarp commence à mesurer le temps. Lorsque le prochain signal de synchronisation arrive (qui est à la même fréquence que la sonde et dont on connaît le délai avec celle-ci par des mesures électriques), c'est l'évènement stop. Nous pouvons ainsi remonter au délai temporel entre le signal lumineux et le signal de synchronisation. En reliant ce dernier au signal de pulse, il est possible de remonter à l'écart temporel entre la réception du signal lumineux et le moment de l'excitation.

En répétant cette opération de nombreuses fois il est possible de faire une statistique sur le nombre de photons incidents sur le détecteur en fonction du délai temporel avec la pompe (voir figure 3.5.1.b).

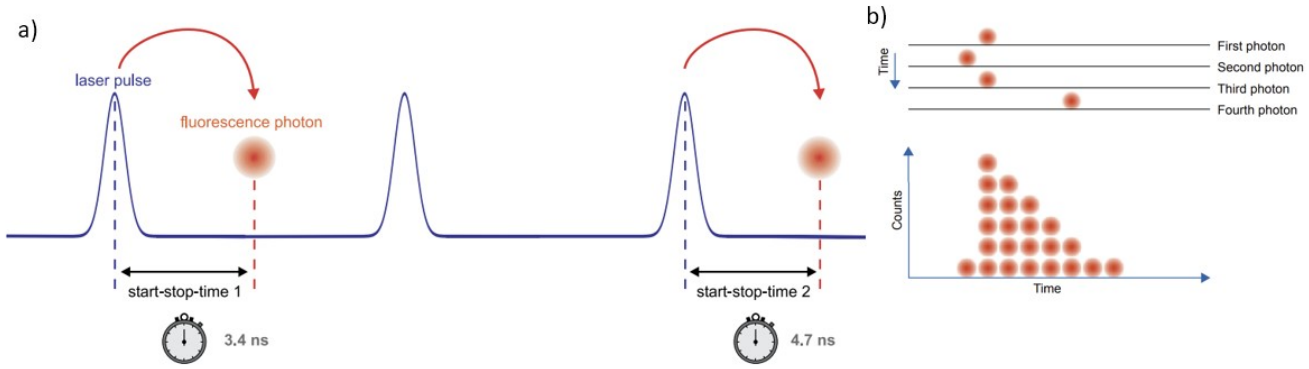

FIGURE 3.5.1 – a) Fonctionnement temporelle d'une prise de mesure start/stop avec le PicoHarp 300 b) Schématisation de la reconstruction d'une dynamique temporelle — Figures extraites de [7]

3.6 Automatisation et programme informatique

Pour chaque délai δ entre l'impulsion laser de pompe et l'impulsion laser de sonde, nous répétons les mesures $N_{samples}$ fois pour obtenir assez de points (puisque nous utilisons un détecteur de photons uniques) et reconstruire la statistique temporelle du nombre de photons dans l'état $|\downarrow\rangle$. A la fin de la mesure, nous pouvons recommencer un nouveau cycle. Ainsi, la fréquence de répétition des cycle peut être variée afin d'optimiser le temps d'acquisition des données.

Après avoir récupérer les données pour un délai δ , il convient de changer de délai (et donc la fréquence des cycles) afin d'étudier la statistique à un autre instant. On répétera là aussi les mesures $N_{samples}$ fois.

En répétant ces étapes, il nous sera alors possible de remonter à la statistique temporelle de l'occupation de l'état $|\downarrow\rangle$.

La durée des de chaque impulsion est une donnée à fournir au programme informatique.

Ainsi, une représentation schématique de l'enchaînement du programme informatique est présentée en figure 3.6.1.

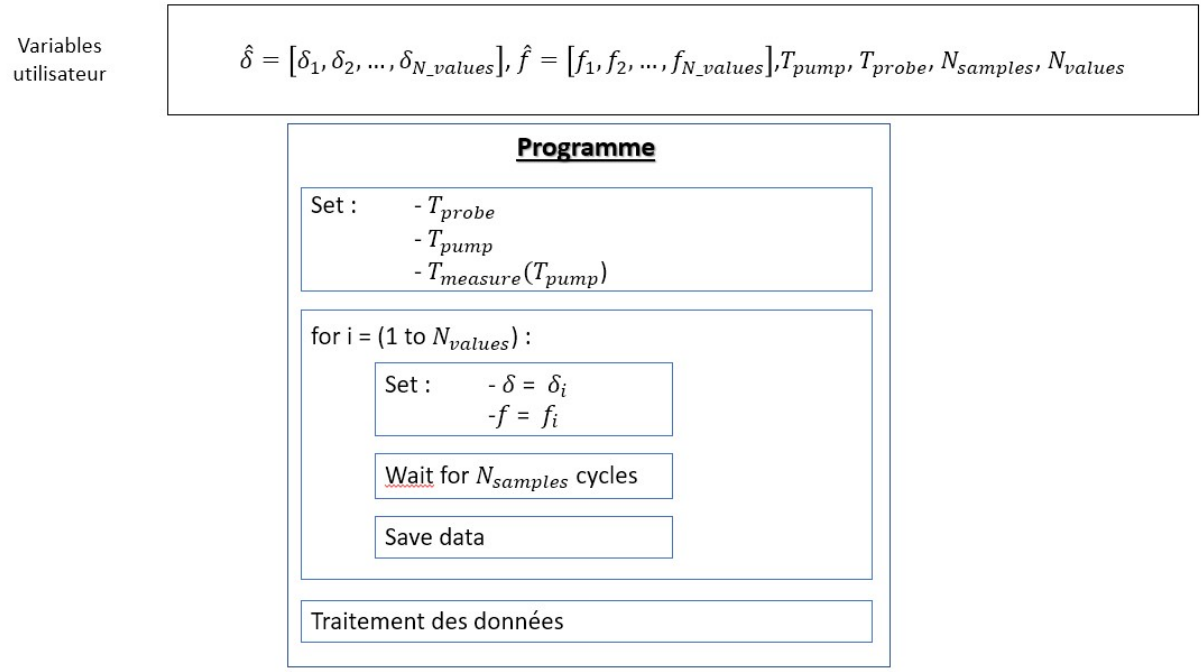

FIGURE 3.6.1 – Représentation schématique des actions à effectuer par le programme informatique

4 Prise en main, spécificité et contrôle des appareils

4.1 L'APD

Il est possible de gater l'APD, c'est à dire de n'activer la détection de photons que lorsqu'on le souhaite (voir comment le faire sur la figure 4.1.1.a)) . Cependant, il est important de noter que les 15 premières et dernières nanosecondes de la fenêtre de mesure sont extrêmement bruitées et non-utilisables (voir figure 4.1.1.b)). Il faut prendre ceci en compte dans le code informatique. Ce "gating" s'effectue par l'application d'une tension sur un connecteur BNC situé à l'arrière de l'APD. Il est important de noter que la documentation d'Excelitas explique d'appliquer un niveau TTL haut (Entre 2 et 5V) pour activer le comptage et un niveau bas (entre 0.5 et 0V) pour désactiver le comptage. Il se trouve qu'après une étude des tensions aux bornes de l'APD, un offset de -2.5V est ajouté à la tension aux bornes du connecteur BNC. Ainsi, les signaux à envoyer à l'APD pour le gater doivent être compris entre -2.5V et 2.5V (-2V à 2V fonctionnent également).

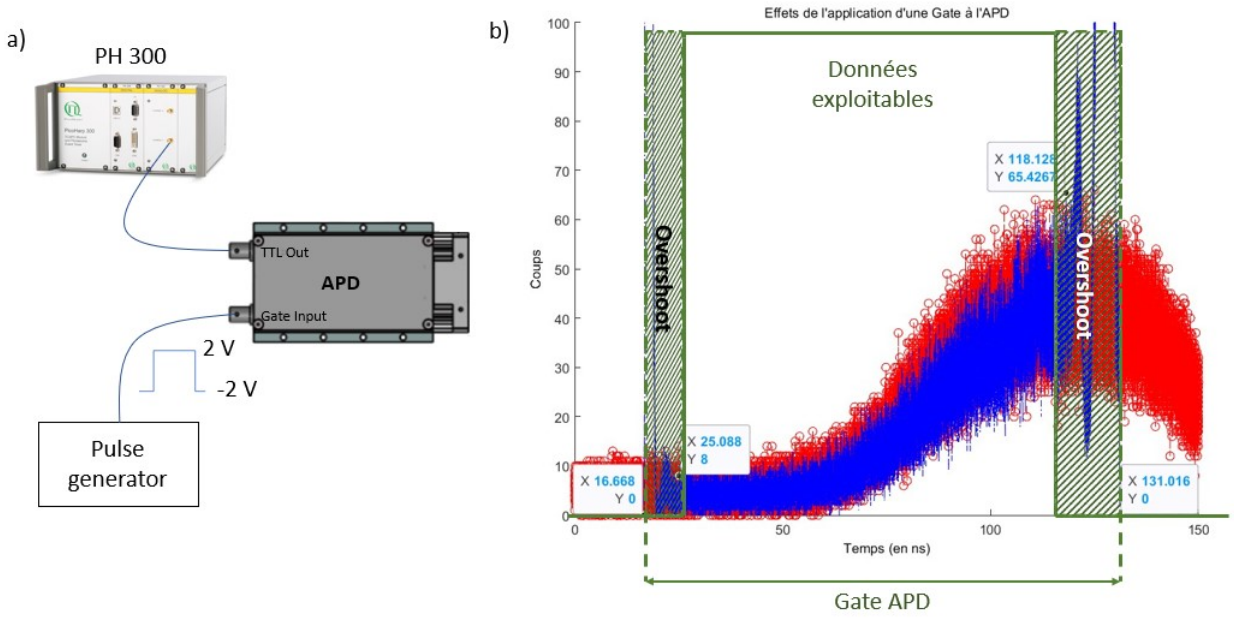

FIGURE 4.1.1 – a) Représentation du schéma électronique pour gater l'APD, b) Illustration du phénomène d'overshoot qui rajoute du bruit

4.2 Le module de comptage de photons

4.2.1 Général

L'électronique du PicoHarp est telle que la détection des signaux électriques se fait sur front descendant. De plus, les signaux appliqués à l'entrée du PicoHarp doivent être compris entre 0 et -1V. Optimalement, le niveau 'logique' haut devrait se situer autour de -600mV. De plus, la largeur temporelle de la fenêtre d'acquisition du PicoHarp dépend de la résolution temporelle choisie. Voir les correspondances en figure 4.2.1

Code range PH	Résolution temporelle (en ps)	Largeur de la fenêtre de mesure (en ns)
0	4	262.14
1	8	524.28
2	16	1048.56
3	32	2097.12
4	64	4194.24
5	128	8388.48
6	256	16776.96
7	512	33553.92

FIGURE 4.2.1 – Correspondance résolution temporelle et largeur temporelle de la fenêtre de mesure du PicoHarp 300

Afin de programmer une mesure, il faut régler le délai entre l'impulsion de pompe et l'impulsion de sonde. Puis, pour la mesure, il faut programmer l'impulsion de synchronisation du PicoHarp de telle sorte à ce que les mesures effectuée se trouvent dans la fenêtre d'acquisition, pour ce faire, on introduit un décalage entre le signal de synchronisation et le signal de pompe afin de mesurer la statistique des photons au temps souhaité.

4.2.2 Contrôle informatique

Grâce au logiciel constructeur, il est possible de trouver les valeurs des différents paramètres tels que les zeros cross optimaux pour la détection des signaux souhaités.

Le PicoHarp possède une dll (*Dynamic Link Library*). Ainsi, j'ai créé une classe python exploitant celle-ci. Elle contient plusieurs fonctions :

- open_ph() :Avec les paramètres trouvés grâce au logiciel constructeur, permet d'ouvrir et de paramétrer le PicoHarp dans python
- set_range(range) : Permet de paramétrer la résolution des mesures (voir code pour les différentes options)
- start_measure_with_plot(T_acq) : Réalise une mesure de la statistique temporelle moyennée pendant le temps T_acq. Toute les secondes une visualisation de cette statistique est mise à jour. Renvoie un vecteur contenant les valeurs temporelles et un vecteur contenant le nombre de coups associé à ces valeurs temporelles.
- start_measure(T_acq) : Réalise une mesure de la statistique temporelle moyennée pendant le temps T_acq. Il n'y a pas de visualisation de la statistique. Renvoie un vecteur contenant les valeurs temporelles et un vecteur contenant le nombre de coups associé à ces valeurs temporelles.
- close_APD : Coupe la communication entre Python et le PicoHarp

4.3 Récapitulatif de la séquence d'impulsion à envoyer pour réaliser une mesure

Afin de réaliser une mesure de luminescence il convient donc de choisir une résolution temporelle du PicoHarp telle que tout le signal de luminescence se trouve dans la fenêtre de mesure, et d'activer l'APD quelques temps avant d'activer la synchronisation du PicoHarp de façon à ce que l'overshoot ne se trouve pas dans la fenêtre de mesure du PicoHarp. De la même manière, il conviendra d'éteindre l'APD quelques temps après la fin de la fenêtre de mesure du PicoHarp. (Voir illustrations en figure 4.3.1).

4.4 Le générateur d'impulsions : le DG645

Le DG645 se contrôle par éthernet, par protocole SCPI grâce à *telnetlib* dans Python. Une classe pyhton avec différente fonctions permettant de le contrôler a été crée. Voici les fonctions et leur rôle :

- open_ip(IP) : Fonction permettant d'initier une connexion avec le DG645 possédant l'ip *"IP"*.
- trig_freq_query() : Fonction retournant la fréquence actuelle du trigger interne du DG645 (fréquence en Hz).
- set_trig_rate(trig_rate) : Permet de régler la fréquence interne du trigger à la fréquence *trig_rate* (en Hz).
- set_gate_APD_channel(channel) : Permet d'indiquer au DG645 de générer des impulsions possèdant la forme appropriée pour gater l'APD étant branché sur l'entrée *channel*. *channel* est un nombre entier prenant des valeurs entre 1 et 4 (1 étant l'entrée BNC AB, 2 étant l'entrée BNC CD, 3 étant l'entrée BNC EF, 4 étant l'entrée BNC GH).

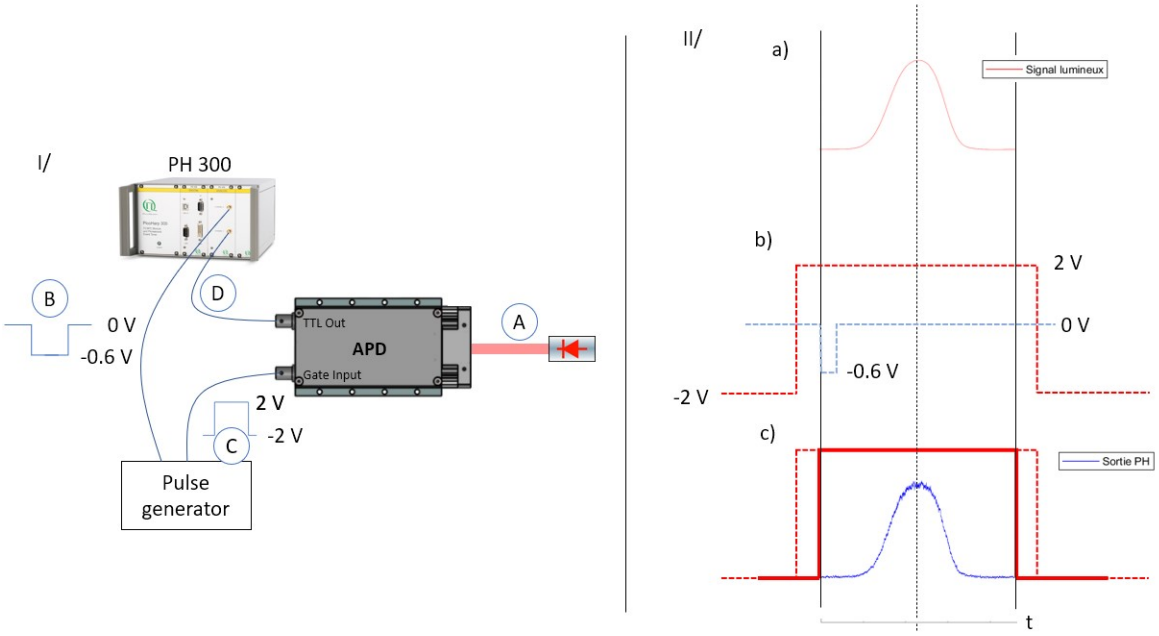

FIGURE 4.3.1 – I/Schéma des connexions entre les appareils et des caractéristiques des signaux. II/ a) Signal lumineux à capter, b) Impulsion électriques à envoyer : en bleu, l'impulsion du PH de telle sorte à ce que le signal lumineux soit centré dans la fenêtre de mesure, en rouge, impulsion pour gater l'APD, c) Représentation des fenêtres de mesure du PH (en trait rouge plein), de la fenêtre de gate de l'APD (en trait rouge pointillé), mesure réalisées au PH (en bleu)

- set_synchro_PH_channel(channel) : Permet d'indiquer au DG645 de générer des impulsions possèdant la forme appropriée pour trigger le PicoHarp étant branché sur l'entrée *channel*. *channel* est un nombre entier prenant des valeurs entre 1 et 4 (1 étant l'entrée BNC AB, 2 étant l'entrée BNC CD, 3 étant l'entrée BNC EF, 4 étant l'entrée BNC GH).
- set_modu_AOM_channel(channel) : Permet d'indiquer au DG645 de générer des impulsions possèdant la forme appropriée pour moduler le driver de l'AOM étant branché sur l'entrée *channel*. *channel* est un nombre entier prenant des valeurs entre 1 et 4 (1 étant l'entrée BNC AB, 2 étant l'entrée BNC CD, 3 étant l'entrée BNC EF, 4 étant l'entrée BNC GH).
- set_modu_EOM_channel(channel) : Permet d'indiquer au DG645 de générer des impulsions possèdant la forme appropriée pour moduler l'EOM étant branché sur l'entrée *channel*. *channel* est un nombre entier prenant des valeurs entre 1 et 4 (1 étant l'entrée BNC AB, 2 étant l'entrée BNC CD, 3 étant l'entrée BNC EF, 4 étant l'entrée BNC GH).
- set_delay_x(delay) Permet de régler le délai entre le trigger et le déclenchement de la channel x (x pouvant être AB, CD, EF, GH ou directement correspondre à un appareil) à *delay* (en s).
- set_larg_x(delay) Permet de régler la largeur de l'impulsion de la channel x (x pouvant être AB, CD, EF, GH ou directement correspondre à un appareil)) à *delay* (en s).
- close() : Permet de couper la connexion avec le DG645.

4.5 Contrôle de l'EOM

L'EOM possède un contrôleur permettant d'effectuer une contre réaction et de garder un niveau moyen de puissance constant. Le point de fonctionnement est choisi par l'utilisateur dans le logiciel du constructeur de l'EOM. Lors des tests, le point de fonctionnement le plus adapté pour réaliser une modulation ON/OFF du signal lumineux était le point de fonctionnement "MIN" dans le logiciel de contrôle. La modulation RF de l'EOM doit se faire grâce à des signaux TTL (haut = ON (3.5V), bas = OFF (0V)).

5 Expérience de test

5.1 Introduction

Une expérience de test, dont le schéma est présenté en figure 5.1.1 a été mise en place pour vérifier le fonctionnement de la démarche et tester différents délais δ. Par manque de beam-splitter fibré, nous n'avons pas réaliser les mesures par photo-diode.

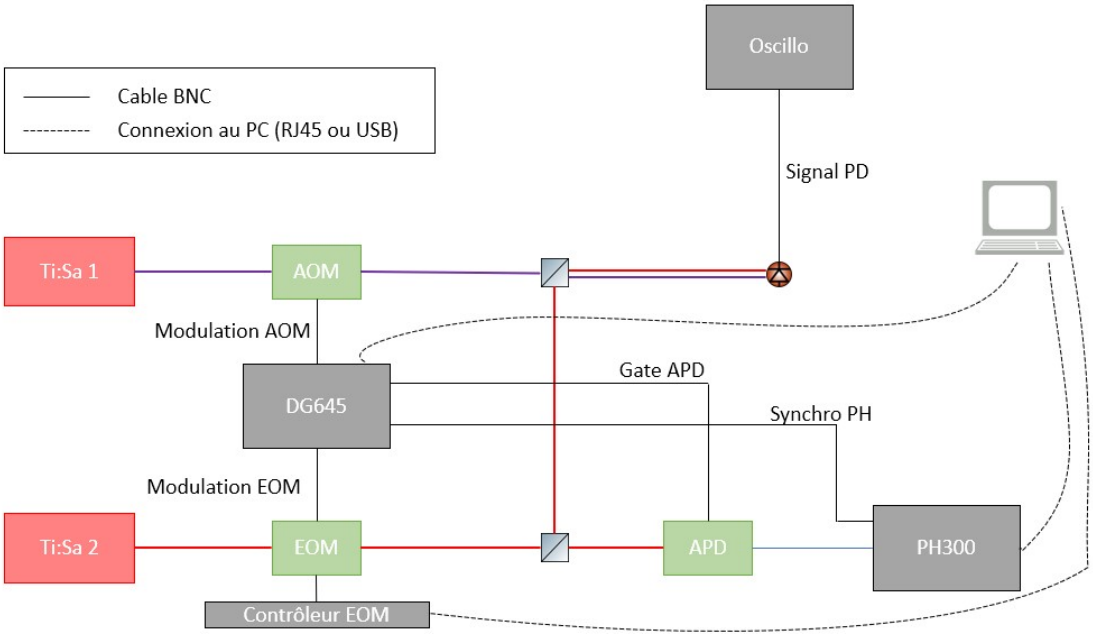

FIGURE 5.1.1 – Schéma de l'expérience permettant de tester la manipulation

5.2 Exemple de code

Un code suivant le schéma de la figure 3.6.1 a été rédigé, le voici :

```
from DG645 import dg645
from PicoHarp_Coding_class import PH
from ctypes import *
import sys
import time
import matplotlib.pyplot as plt
from scipy.io import savemat
import numpy as np

#%% Important informations for the code

HOST_DG645 = b"192.168.1.103" #DG645 IP addres

#%% Set the cycles options
wanted_number_of_cycles=100000 #Number of cycles (periods) the PH will integrate on
vect_delay = [1,10,20,30,40,50] # in us, #The different delays we want to measure between the end
↪  of the pump and the probe


#%% Opening the devices

mydg=dg645() #Instanciating the code for controlling the DG645 : the impulsions generator
myph=PH() #Instanciating the code for controlling the PicoHarp

myph.open_ph() #Opening the connexion to the PicoHarp
mydg.open_ip(HOST_DG645) #Opening the connexion to the DG645

#%% Set the resolution of the PH

#Range:
    # 0, Resolution 4ps, Time span ; 262.1 ns
    # 1, Resolution 8ps, Time span ; 524.3 ns
    # 2, Resolution 16ps, Time span ; 1.049 us
    # 3, Resolution 32ps, Time span ; 2.097 us
    # 4, Resolution 64ps, Time span ; 4.194 us
    # 5, Resolution 128ps, Time span ; 8.389 us
```

```
    # 6, Resolution 256ps, Time span ; 16.777 us
    # 7, Resolution 512ps, Time span ; 33.554 us
#/!\ Default range is 0

#Pour des sondes de 5us pourquoi pas une reso de 128ps?
my_range=5 #Range we chose
myph.set_range(my_range) #Set the resolution of the PH

mydg.set_modu_AOM_channel(1) # Indicating the channel the APD driver is connecting on
mydg.set_modu_EOM_channel(2) # Indicating the channel the EOM is connecting on
mydg.set_synchro_PH_channel(3) # Indicating the channel the PH trig channel is connecting on
mydg.set_gate_APD_channel(4) ## Indicating the channel the APD gate channel is connecting on

#%% Set the largers of the pulses we want

time_unit="us" #All the times will be in this unit
time_multi=1e-6 #Number we multiplicate seconds by to get this unity
length_pump = 50 #Length of the pump (AOM modu)
larg_sonde = 5 #Length of the probe (EOM modu)
larg_PH=t = 2**((my_range)+2) * 65535 * 10**-6 #Length of the PH window (determined by the range)
wait_gate_APD=0.1 #Time wait before the APD gating and the PH window to avoid overshoot
larg_gate_APD = larg_PH + (2*wait_gate_APD) #Larger of the gate for the APD
wait_after_end=1 # Time wait after closing APD gate
time_between_PH_probe = (larg_PH-larg_sonde)/2


mydg.set_larg_APD(larg_gate_APD*time_multi) #Setting the larger of the gate
mydg.set_larg_PH(80e-9) #Setting the larger of the PH trigger (the legth does not count, only the
↪  change of state does)
mydg.set_larg_AOM(length_pump*time_multi) #Setting the larger of the modulation of the AOM
mydg.set_larg_EOM(larg_sonde*time_multi) #Setting the larger of the modulation of the EOM

#%% Physical cable delays
#Assuming the delay for the AB cable is zero
cable_PH_trig=0 #Delay between the trigger and the actual trggering at the end of the cable
↪  running to the PH (this number should be positive)
cable_probe_trig=(0) #Delay between the trigger and the actual trggering at the end of the cable
↪  running to the EOM (this number should be positive)
cable_APD_trig=0 #Delay between the trigger and the actual trggering at the end of the cable
↪  running to the APD (this number should be positive)



#%% Boucle de mesure

for i in range(len(vect_delay)):

    ##Setting the delays of the diverse impulsions and trig rate for the DG645
    delay_pump_probe = vect_delay[i]
    beg_probe=delay_pump_probe+length_pump
    beg_PH=beg_probe - time_between_PH_probe
    beg_APD = beg_PH-wait_gate_APD

    end_probe = beg_probe + larg_sonde
    end_PH=beg_PH+larg_PH
    end_APD = beg_APD + larg_gate_APD

    period_mes=end_APD+wait_after_end
    t_acquisition = (period_mes*wanted_number_of_cycles)/1000 #in usec
    f_pump = 1/(period_mes * time_multi)

    mydg.set_trig_rate(f_pump)
    mydg.set_delay_APD((beg_APD-cable_APD_trig)*time_multi)
```

```
    mydg.set_delay_PH((beg_PH-cable_PH_trig)*time_multi)
    mydg.set_delay_AOM(0)
    mydg.set_delay_EOM((beg_probe-cable_probe_trig)*time_multi)

    #Taking a measure with the PH
    [new_counts,new_t]=myph.start_measure_with_plot((t_acquisition))

    #Arranging the datas into matrix for saving

    if (i>1):
        temp_t=np.zeros([1,np.shape((new_t))[0]])
        temp_count=np.zeros([1,np.shape((new_t))[0]])
        new_t+=(beg_PH-cable_PH_trig )
        temp_t[0,:]=new_t
        temp_count[0,:]=new_counts
        new_t=np.concatenate((old_t, temp_t), axis=0)
        new_counts=np.concatenate((old_count, temp_count), axis=0)
        old_t=new_t
        old_count=new_counts
        mdic = {"Delays": vect_delay, "Time": new_t, "Counts": new_counts}

    elif (i==0):
        new_t+=(beg_PH-cable_PH_trig)
        old_t=new_t
        old_count=new_counts
        mdic = {"Delays": vect_delay, "Time": new_t, "Counts": new_counts}

    elif (i==1):
        temp_t=np.zeros([2,np.shape((old_count))[0]])
        temp_count=np.zeros([2,np.shape((old_count))[0]])
        temp_t[0,:]=old_t
        temp_count[0,:]=old_count
        new_t+=(beg_PH-cable_PH_trig)
        temp_t[1,:]=new_t
        temp_count[1,:]=new_counts
        old_t=temp_t
        old_count=temp_count
        mdic = {"Delays": vect_delay, "Time": new_t, "Counts": new_counts}

    savemat("matlab_matrix.mat", mdic)


#%% Close all

# mytombak.off_out()
myph.close_APD()
mydg.close()
```

5.3 Résultats

Pour le test, 6 délais δ ont été appliqués. Pour chaque délai, les coups de l'APD ont été intégrés durant 100 000 cycles. Puis, le délai suivant est appliqué. Les six couleurs de signaux différentes correspondent aux résultats intégrés par le PH pour différents délais δ.

Nous observons alors que le système fonctionne comme attendu. Il faudra faire attention cependant lors du montage de l'expérience final à prendre en compte les délais introduits par la longueur des cables partant du DG645 aux autres appareils. Cela est prix en compte dans le code dans la section *Physical cable delays*.

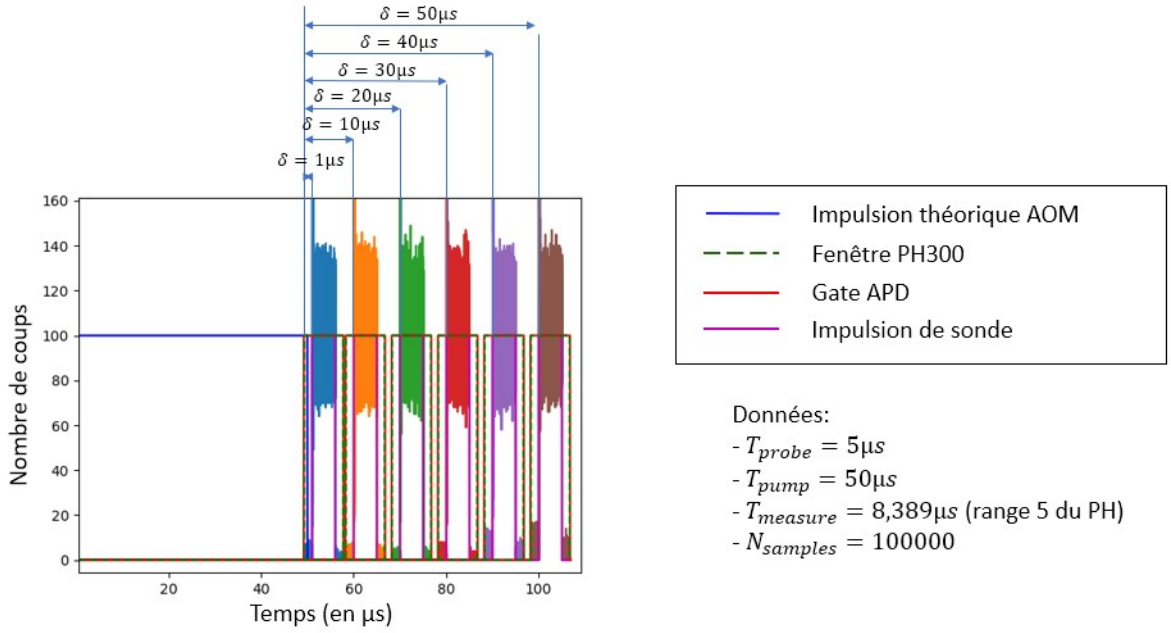

FIGURE 5.3.1 – Résultats récoltés grâce au PH et mis en commun pour l'expérience de la figure 5.1.1

6 Schéma de l'expérience finale

Voici (figure 6.0.1) un schéma de l'expérience finale telle qu'elle pourrait être utilisée (l'échantillon est en configuration transmission pour plus de lisibilité du schéma).

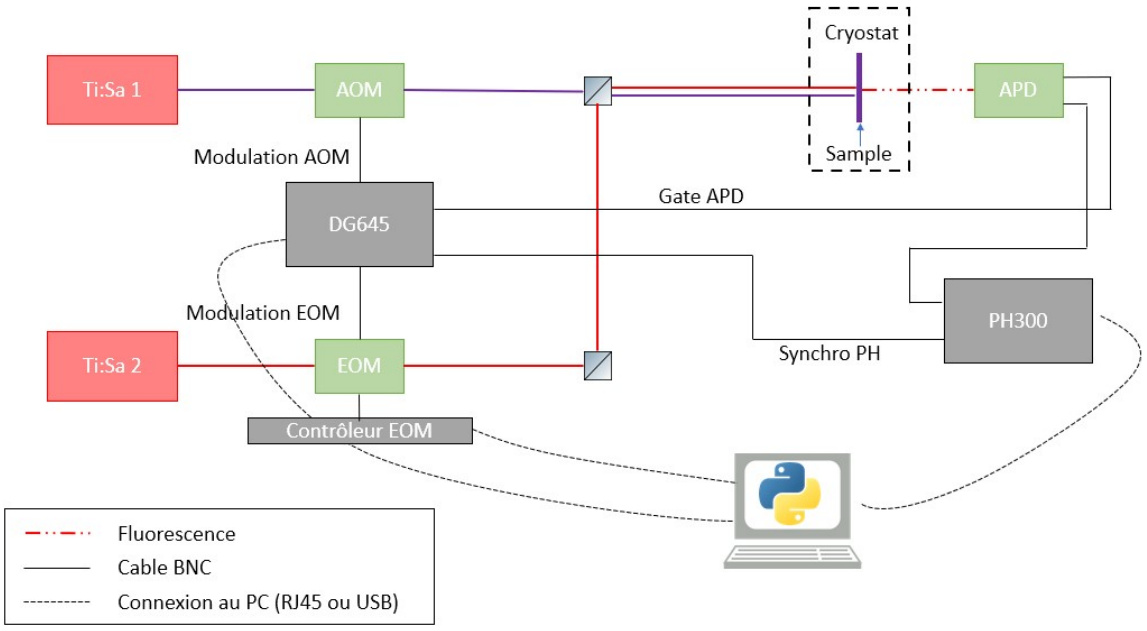

FIGURE 6.0.1 – Schéma de l'expérience finale

7 Récapitulatif des logiciels et des installations à faire

- Python 32 bit avec les bibliothèques suivantes :
  - matplotlib
  - scipy.io
  - numpy
  - telnetlib
  - pyserial
- Lociciel de contrôle du PicoHarp

— DLL du Picoharp ! Elle est en 32 bits !
— Logiciel Aerodiode Tombak
— Module python Aerodiode Tombak
— Logiciel contrôle BIAS Ixblue (pour l'EOM)

8 Update du 07/04 : Ajout de la renormalisation

Les lasers utilisés ne sont pas très stables en puissance, ainsi, il paraît intéressant de mesurer la puissance moyenne envoyer sur l'échantillon. Il s'agit donc d'additionner la puissance dans chaque pulse de sonde. Pour ce faire, nous avons utiliser le SR400 (sa documentation est disponible ici https://www.thinksrs.com/downloads/pdfs/manuals/SR400m.pdf, un compteur qui peut gater ses mesures. Nous utiliserons donc un autre APD, que nous brancherons sur l'input 1 du SR400. Un code commenter sur la communication en RS232 avec cet appareil est disponible ici : https://github.com/CorentinMorinSorbonne/spin_init/blob/main/Main/New/SR400.py.

Nous triggerons celui-ci avec une photodiode rapide sur le front montant du signal de sonde. Tous les codes sont situés ici : https://github.com/CorentinMorinSorbonne/spin_init/tree/main/Main/New. Cette photodiode sera reliée à l'entrée Trigger du SR400. Ce trigger peut être réglé entre -2V et 2V.

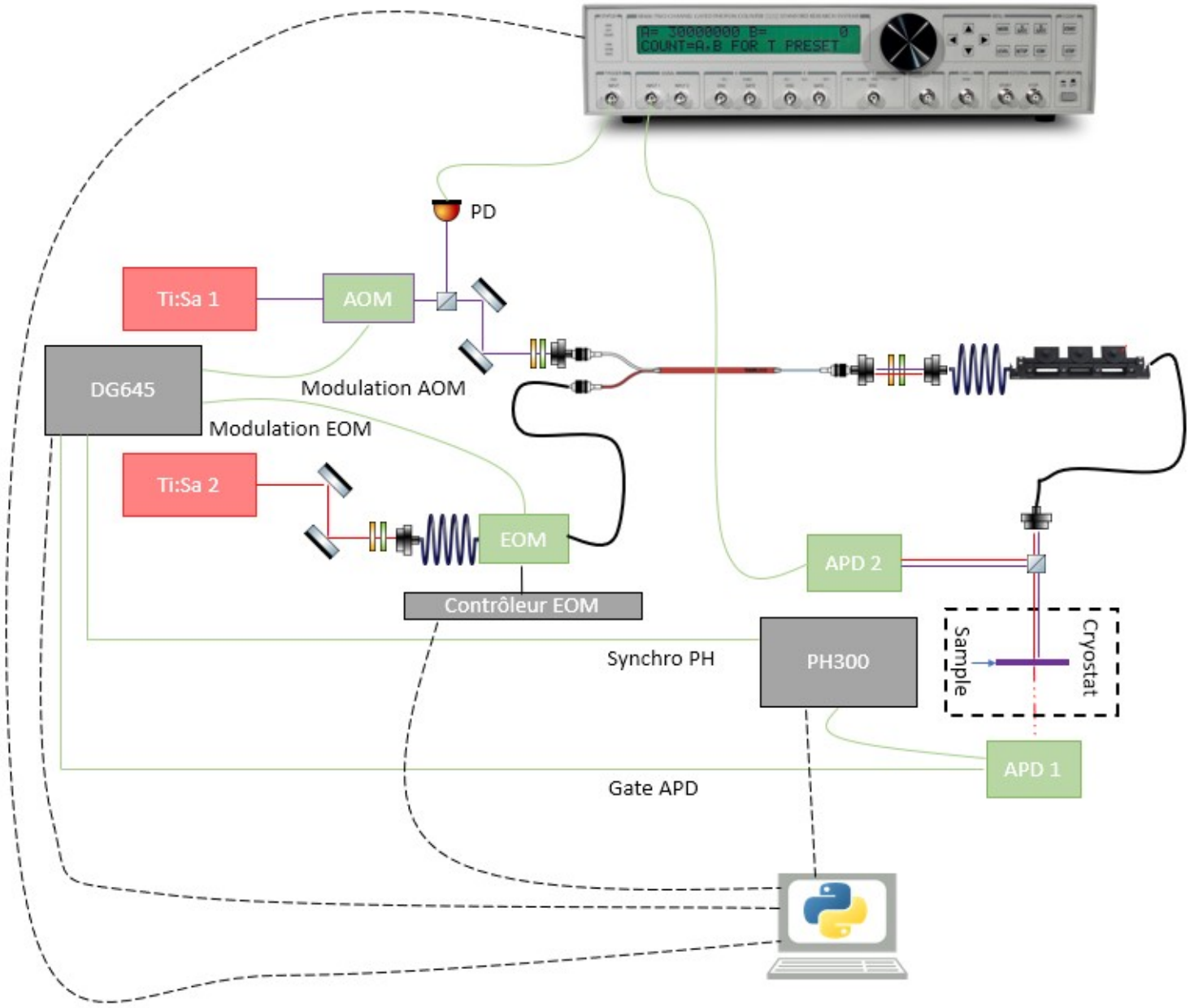

FIGURE 8.0.1 – Schéma total de l'expérience

L'idée est de compter seulement sur des gates que l'on peut choisir (voir les codes), pour ce faire on choisi la longueur de la gate et son décalage avec le . Un exemple de fonctionnement est présenté en figure 8.0.2.

Le code ensuite divise simplement le comptage du PicoHarp par le comptage du SR400 : cela crée une renormalisation.

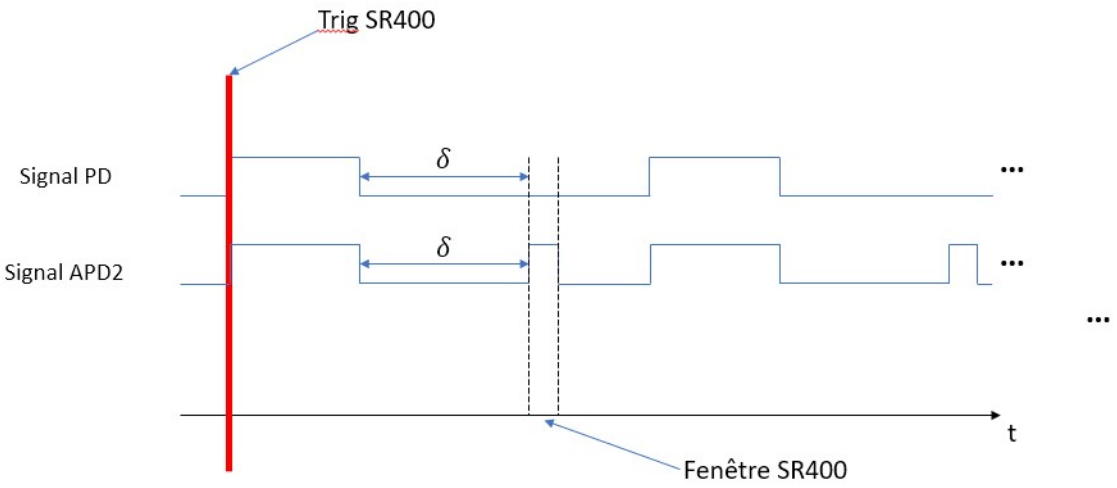

FIGURE 8.0.2 – Principe du comptage sur le SR400

8.1 Présentation des codes

8.1.1 Interfaçage du SR400

```
    import serial
import time

class mySR400():

    def open_sr(self,com_string):

        self.ser = serial.Serial(com_string, 9600, timeout=0, parity=serial.PARITY_EVEN,
        ↪  rtscts=1)
        self.ser.write(b"CR\n") #Reset Counters
        self.ser.write(b"SS \n") #Set counter A and B and Trigger
        self.ser.readlines()


    def setup_sr(self):
        self.ser.write(b"CM 0\n") #Set counter A and B and Trigger
        self.ser.write(b"CI 0,1\n") #Set counter A on input 1
        self.ser.write(b"CI 1,1\n") #Set counter B on input 1
        self.ser.write(b"CI 2,3\n") #Set T on trigger input
        self.ser.write(b"GM 0,1\n") #Set A counter gate fixed
        self.ser.write(b"GM 1,1\n") #Set B counter gate fixed
        self.ser.write(b"NP 1\n") # Number of counting after trigger value reached = 1
        self.ser.write(b"NE 0\n") #Counting will stop  after trigger value reached
        self.ser.write(b"DT 2E-3\n") #Minimum dwell time
        self.ser.write(b"TS 0\n") #Trigger on rising edge
        self.ser.write(b"TL 0.5\n") #Trigger at 0.5V
        self.ser.write(b"DS 0,1\n") #Rising slope for counter A
        self.ser.write(b"DS 1,1\n") #Rising slope for counter B
        self.ser.write(b"DM 0,0\n") #Discriminator mode fixed
        self.ser.write(b"DM 1,0\n") # //
        self.ser.write(b"DM 2,0\n") # //
        self.ser.write(b"DL 0,-0.3\n") #Discriminator levels @ 0.3V (the max)
        self.ser.write(b"DL 1,-0.3\n") # //
        self.ser.write(b"DL 2,0.3\n") # //
        self.ser.write(b"SS \n") #Set counter A and B and Trigger
        print("Error reg after setting: "+ self.ser.readline().decode())

    def is_data_ready_query(self):
        self.ser.readline()
        self.ser.write(b"SS 1\n") #Set counter A and B and Trigger
```

```
        time.sleep(1)
        string=self.ser.readline()
        biny=int(string.decode())
        return biny


    def beg_count(self, n_per,delay_pump,width_pump,delay_probe,width_probe):

        self.ser.write(b"CR \n") #Set number of trigger counts

        self.ser.write(b"CP 2,"+str(n_per).encode()+b"\n") #Set number of trigger counts
        self.ser.write(b"GD 0,"+str(delay_pump).encode()+b"\n") #Set delay for counter A, the
        ↪  pump, should be 0
        self.ser.write(b"GD 1,"+str(delay_probe).encode()+b"\n") #Set delay for counter B, the
        ↪  probe
        self.ser.write(b"GW 0,"+str(width_pump).encode()+b"\n") #Set delay for counter B, the
        ↪  probe
        self.ser.write(b"GW 1,"+str(width_probe).encode()+b"\n") #Set delay for counter B, the
        ↪  probe

        self.ser.write(b"CS\n") #Start counting

    def recover_count(self):

        self.ser.readline()


        self.ser.write(b"QA\n")
        time.sleep(0.5)
        count_A=int(self.ser.readline().decode())

        self.ser.write(b"QB\n")
        time.sleep(0.5)

        count_B=int(self.ser.readline().decode())
        return count_A, count_B

    def close_sr(self):
        self.ser.close()
```

8.1.2 Code total

```
    from DG645 import dg645
from PicoHarp_Coding_class import PH
from my_aerodiode_class import my_tombak
from SR400 import mySR400
from ctypes import *
import sys
import time
import matplotlib.pyplot as plt
from scipy.io import savemat
import numpy as np

#%% Important informations for the code

HOST_DG645 = b"192.168.1.103" #DG645 IP addres
# COM_tombak = 'COM5' #COM address for the aerodiode
#%% Set the cycles options
wanted_number_of_cycles=100000 #Number of cycles (periods) the PH will integrate on
vect_delay = [0] #  # in us, The different delays we want to measure between the end of the pump
↪  and the probe


```

```
#%% Opening the devices

mydg=dg645()  #Instanciating the code for controlling the DG645 : the impulsions generator
myph=PH() #Instanciating the code for controlling the PicoHarp
mysr=mySR400() #Instanciating the code for controlling the SR400

myph.open_ph() #Opening the connexion to the PicoHarp
mydg.open_ip(HOST_DG645) #Opening the connexion to the DG645
mysr.open_sr("COM4") #Opening the connexion to the SR400 Warning : check COM port
mysr.setup_sr() #Setup the RS400

#%% Set the resolution of the PH

#Range:
    # 0, Resolution 4ps, Time span ; 262.1 ns
    # 1, Resolution 8ps, Time span ; 524.3 ns
    # 2, Resolution 16ps, Time span ; 1.049 us
    # 3, Resolution 32ps, Time span ; 2.097 us
    # 4, Resolution 64ps, Time span ; 4.194 us
    # 5, Resolution 128ps, Time span ; 8.389 us
    # 6, Resolution 256ps, Time span ; 16.777 us
    # 7, Resolution 512ps, Time span ; 33.554 us
#/!\ Default range is 0

#Pour des sondes de 5us pourquoi pas une reso de 128ps?
my_range=5  #Range we chose
myph.set_range(my_range) #Set the resolution of the PH

mydg.set_modu_AOM_channel(1) # Indicating the channel the APD driver is connecting on
mydg.set_modu_EOM_channel(4) # Indicating the channel the EOM is connecting on
mydg.set_synchro_PH_channel(3) # Indicating the channel the PH trig channel is connecting on
mydg.set_gate_APD_channel(2) # Indicating the channel the APD gate channel is connecting on

#%% Set the largers of the pulses

time_unit="us" #All the times will be in this unit
time_multi=1e-6 #Number we multiplicate seconds by to get this unity
length_pump = 50  #Length of the pump (AOM modu)
larg_sonde = 5 #Length of the probe (EOM modu)
larg_PH=t = 2**((my_range)+2) * 65535 * 10**-6 #Length of the PH window (determined by the range)
wait_gate_APD=0.1 #Time wait before the APD gating and the PH window to avoid overshoot
larg_gate_APD = larg_PH + (2*wait_gate_APD) #Larger of the gate for the APD
wait_after_end=1 # Time wait after closing APD gate
time_between_PH_probe = (larg_PH-larg_sonde)/2 #A constant for the code


mydg.set_larg_APD(larg_gate_APD*time_multi) #Setting the larger of the gate
mydg.set_larg_PH(80e-9) #Setting the larger of the PH trigger (the legth does not count, only the
    ↪  change of state does)
mydg.set_larg_AOM(length_pump*time_multi) #Setting the larger of the modulation of the AOM
mydg.set_larg_EOM(larg_sonde*time_multi)  #Setting the larger of the modulation of the EOM

#%% Physical cable delays
#Assuming the delay for the AB cable is zero
# cable_PH_trig=0 #Delay between the trigger and the actual trggering at the end of the cable
    ↪  running to the PH (this number should be positive)
# cable_probe_trig=(0) #Delay between the trigger and the actual trggering at the end of the
    ↪  cable running to the EOM (this number should be positive)
# cable_APD_trig=0 #Delay between the trigger and the actual trggering at the end of the cable
    ↪  running to the APD (this number should be positive)


cable_PH_trig=0.133
```

```
cable_probe_trig=(0)
cable_APD_trig=0.133 #trig first


#%% Boucle de mesure
my_ind=0

for i in range(len(vect_delay)):
     ##Setting the delays of the diverse impulsions and trig rate for the DG645
    delay_pump_probe = vect_delay[i]
    beg_probe=delay_pump_probe+length_pump
    beg_PH=beg_probe - time_between_PH_probe
    beg_APD = beg_PH-wait_gate_APD

    end_probe = beg_probe + larg_sonde
    end_PH=beg_PH+larg_PH
    end_APD = beg_APD + larg_gate_APD

    period_mes=end_APD+wait_after_end
    t_acquisition = (period_mes*wanted_number_of_cycles)/1000 #in usec
    f_pump = 1/(period_mes * time_multi)

    mydg.set_trig_rate(f_pump)
    mydg.set_delay_APD((beg_APD-cable_APD_trig)*time_multi) #Setting the begining of the gate
    mydg.set_delay_PH((beg_PH-cable_PH_trig)*time_multi)
    mydg.set_delay_AOM(0)
    mydg.set_delay_EOM((beg_probe-cable_probe_trig)*time_multi)

     #Taking a measure with the PH

    [new_counts,new_t]=myph.start_measure_with_plot((t_acquisition)) #Taking a measure

    # Counting the power of the pump with the SR400, the figures are the length of the gate for
    ↪  the measure
    mysr.beg_count(wanted_number_of_cycles, 100E-9, 100E-9, length_pump*time_multi +
    ↪  delay_pump_probe*time_multi + 100E-9, larg_sonde*time_multi - 100E-9)

    dark=3000 #Dark count for the APD on the SR400
    # Calculating the number of dark counts on the SR400
    duty_cycle_SR400= (larg_sonde*time_multi - 100E-9)*f_pump
    tau_integ=wanted_number_of_cycles*duty_cycle_SR400/f_pump
    total_dark=3000*tau_integ

    if (i>1):
        temp_t=np.zeros([1,np.shape((new_t))[0]])
        temp_count=np.zeros([1,np.shape((new_t))[0]])
        temp_value_SR400=np.zeros([1,np.shape((old_value_SR400[0,:]))[0]])

        new_t+=(beg_PH-cable_PH_trig )
        temp_t[0,:]=new_t
        temp_count[0,:]=new_counts
        new_t=np.concatenate((old_t, temp_t), axis=0)
        new_counts=np.concatenate((old_count, temp_count), axis=0)
        old_t=new_t
        old_count=new_counts

        ready_quer = mysr.is_data_ready_query()
        while ready_quer==0:
            ready_quer = mysr.is_data_ready_query()
        count_SR400=mysr.recover_count() #Recovering the number of counts of the SR400
        new_renormalized_APD_counts=np.concatenate((old_renormalized_APD_counts,
        ↪  temp_count/(count_SR400[1])), axis=0)
        old_renormalized_APD_counts=new_renormalized_APD_counts
```

```
        ↪ new_renormalized_APD_counts_darkless=np.concatenate((old_renormalized_APD_counts_without_dark,
        ↪ temp_count/(count_SR400[1]-total_dark)), axis=0)
        old_renormalized_APD_counts_without_dark=new_renormalized_APD_counts_darkless

        temp_value_SR400[0,:]=count_SR400
        new_value_SR400=np.concatenate((old_value_SR400, temp_value_SR400), axis=0)
        old_value_SR400=new_value_SR400

        mdic = {"Delays": vect_delay, "Time": new_t, "Counts": new_counts, "Counts_renormalized":
        ↪ old_renormalized_APD_counts, "Counts_renormalized_corrected":
        ↪ old_renormalized_APD_counts_without_dark, "SR400_Counts": old_value_SR400}

    elif (i==0):
        new_t+=(beg_PH-cable_PH_trig)
        old_t=new_t
        old_count=new_counts

        ready_quer = mysr.is_data_ready_query()
        while ready_quer==0:
            ready_quer = mysr.is_data_ready_query()

        count_SR400=mysr.recover_count()

        old_renormalized_APD_counts=new_counts/(count_SR400[1])
        old_renormalized_APD_counts_without_dark=new_counts/(count_SR400[1]-total_dark)

        old_value_SR400=count_SR400

        mdic = {"Delays": vect_delay, "Time": new_t, "Counts": new_counts, "Counts_renormalized":
        ↪ old_renormalized_APD_counts, "Counts_renormalized_corrected":
        ↪ old_renormalized_APD_counts_without_dark, "SR400_Counts": old_value_SR400}

    elif (i==1):
        temp_t=np.zeros([2,np.shape((old_count))[0]])
        temp_count=np.zeros([2,np.shape((old_count))[0]])
        temp_renormalized_APD_counts=np.zeros([2,np.shape((old_renormalized_APD_counts))[0]])

        ↪ temp_renormalized_APD_counts_darkless=np.zeros([2,np.shape((old_renormalized_APD_counts))[0]])

        temp_value_SR400=np.zeros([2,np.shape((old_value_SR400))[0]])

        temp_t[0,:]=old_t
        temp_count[0,:]=old_count
        temp_renormalized_APD_counts[0,:]=old_renormalized_APD_counts
        temp_renormalized_APD_counts_darkless[0,:]=old_renormalized_APD_counts_without_dark
        temp_value_SR400[0,:]=old_value_SR400

        ready_quer = mysr.is_data_ready_query()
        while ready_quer==0:
            ready_quer = mysr.is_data_ready_query()

        count_SR400=mysr.recover_count()


        new_t+=(beg_PH-cable_PH_trig)
        temp_t[1,:]=new_t
        temp_count[1,:]=new_counts
        temp_renormalized_APD_counts[1,:]=new_counts/(count_SR400[1])
        temp_renormalized_APD_counts_darkless[1,:]=new_counts/(count_SR400[1]-total_dark)
        temp_value_SR400[1,:]=count_SR400

```

```
            old_t=temp_t
            old_count=temp_count
            old_renormalized_APD_counts=temp_renormalized_APD_counts
            old_renormalized_APD_counts_without_dark=temp_renormalized_APD_counts_darkless
            old_value_SR400=temp_value_SR400

            mdic = {"Delays": vect_delay, "Time": new_t, "Counts": new_counts, "Counts_renormalized":
            ↪  old_renormalized_APD_counts,"Counts_renormalized_corrected":
            ↪  old_renormalized_APD_counts_without_dark, "SR400_Counts": old_value_SR400}

        savemat("matlab_matrix.mat", mdic)


    #%% Close all

    # # mytombak.off_out()
    myph.close_APD()
    mydg.close()
    mysr.close_sr()

```

Références

[1] C.-Y. Lu, Y. Zhao, A. N. Vamivakas, C. Matthiesen, S. Fält, A. Badolato, and M. Atatüre, "Direct measurement of spin dynamics in InAs/GaAs quantum dots using time-resolved resonance fluorescence," *Physical Review B*, vol. 81, no. 3, p. 035332, Jan. 2010. [Online]. Available : https://link.aps.org/doi/10.1103/PhysRevB.81.035332

[2] A. Reigue, "Boîte quantique en interaction avec son environnement : excitation résonante pour l'étude des processus de décohérence," Ph.D. dissertation, Pierre et Marie Curie, INSP Paris, Sep. 2017.

[3] R. J. Warburton, C. Schäflein, D. Haft, F. Bickel, A. Lorke, K. Karrai, J. M. Garcia, W. Schoenfeld, and P. M. Petroff, "Optical emission from a charge-tunable quantum ring," *Nature*, vol. 405, no. 6789, pp. 926–929, Jun. 2000. [Online]. Available : http://www.nature.com/articles/35016030

[4] A. Opto-Electronic, "Acousto-optic Theory, Application Notes," 2013. [Online]. Available : http://www.aaoptoelectronic.com/wp-content/uploads/documents/AAOPTO-Theory2013-4.pdf

[5] J. W. Goodman, *Introduction To Fourier Optics*, 3rd ed. Englewood, Colo : Roberts & Company Publishers, Apr. 2016.

[6] Jenoptik, "How can you influence laser light powerfully ?" [Online]. Available : https://www.jenoptik.com/products/optoelectronic-systems/light-modulation/integrated-optical-modulators-fiber-coupled

[7] "PicoHarp 300 | PicoQuant." [Online]. Available : https://www.picoquant.com/products/category/tcspc-and-time-tagging-modules/picoharp-300-stand-alone-tcspc-module-with-usb-interface#documents