Introduction

Mon stage s'inscrit dans le cadre d'une manipulation de spectroscopie de fluorescence dans le laboratoire Lphia de l'université d'Angers.

Le Standford SR430, est un ancien appareil de mesure multi-canal, utilisé dans divers applications, et est considéré comme un compteur de photons qui compte des événements en fonction du temps.

Figure 1 : Le Stanford SR430

LE Stanford SR430 est utilisé depuis plusieurs années dans la manipulation de spectroscopie de fluorescence, et malgré de bons résultats de mesures, son mode de fonctionnement est devenu aujourd'hui très laborieux. Cet appareil, est en effet de plus en plus difficile à remettre en route après une extinction. Des problèmes avec le bus GPIB empêchent la communication avec l'ordinateur et impose l'utilisation de vieilles disquettes 3"1/2 pour récupérer les mesures. Enfin la pile interne de sauvegarde des paramètres est défectueuse et demanderait un retour de l'appareil chez le fabricant aux États-Unis. Il devient donc urgent de trouver une solution de remplacement à cet appareil de mesure.

Le comptage de photons impose de pouvoir compter des impulsions de l'ordre de quelques nanosecondes à des fréquences élevées. Les circuits logiques programmable et reconfigurable comme les FPGA sont naturellement bien adaptés à ces taches.

Durant ce travaille on s'est intéressé à un article qui décrit comment construire et exploiter une coïncidence statistique rapide, "Simple and Inexpensive FPGA-based Fast Time Resolving

Acquisition Board ", cet article décrit aussi comment faire l'acquisition pour la détection des coïncidences entre les impulsions électriques avec une carte FPGA .

On a étudié et analysé le fonctionnement de cette carte FPGA, qui est un circuit électrique reconfigurable et qui contient des millions de composants, afin de configurer cette carte, et construire la logique dont on a besoin pour faire le comptage de photons pour mesurer les temps de vie. Il fallait définir des tâches de traitement numérique par le logiciel LabVIEW, puis de les compiler sous forme de fichier DLL, qui est une bibliothèque de liens dynamiques, qui contient des informations sur la manière dont les composants doivent être reliés, afin de reprogrammer la carte FPGA et de la reconfigurer à notre manière, pour faire le comptage de photons, et pour qu'elle remplace au final l'ancien appareil de mesure le Stanford SR430, utilisé dans la manipulation de spectroscopie de fluorescence.

1. Description de la manipulation de spectroscopie de fluorescence

La manipulation de spectroscopie de fluorescence consiste à mesurer des spectres de fluorescence d'un échantillon de verre dopé à l'Europium Eu^{3+} et excité par différents lasers.
Les lasers utilisés sont un laser à solide NdYAG et un laser à Argon ionisé.
Durant notre travaille sur la manipulation de spectroscopie de fluorescence, nous avons travaillé juste avec le laser NdYAG, comme le montre la figure ci-dessous :

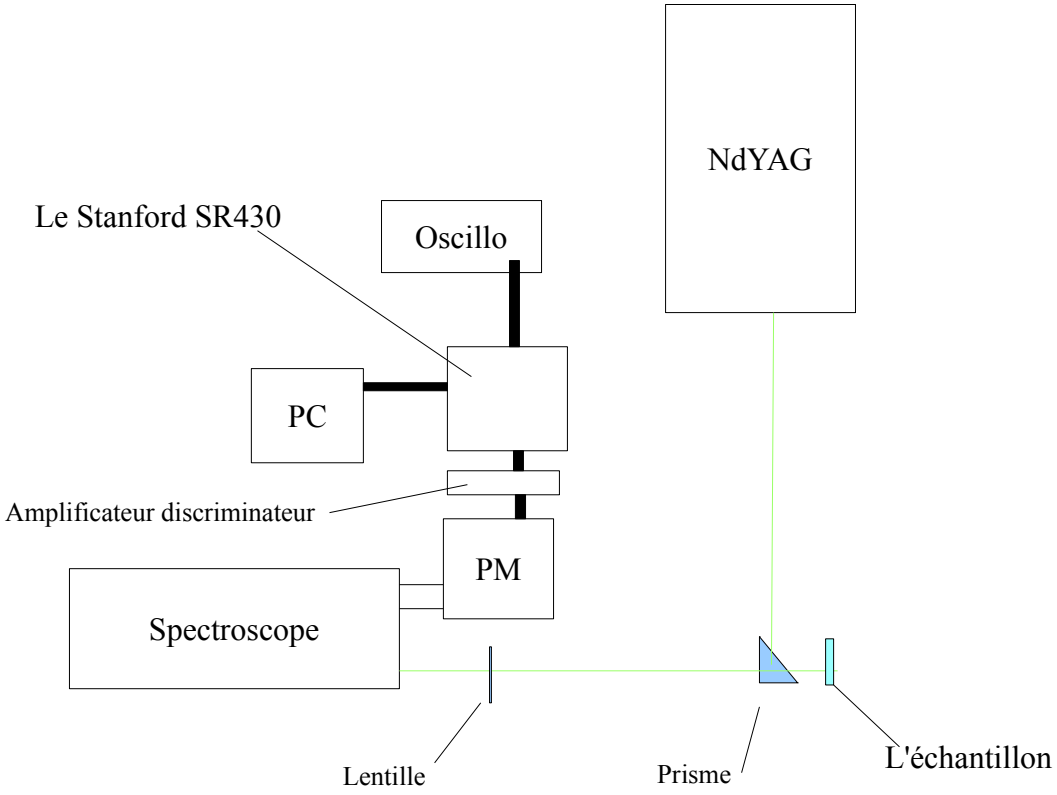


Figure 2 : schéma du dispositif expérimentale avec le Stanford SR430

Le faisceau laser est dirigé grâce à un prisme sur l'échantillon Eu^{3+} qui fluoresce, et par la suite le faisceau issu de la fluorescences est focalisé sur la fente d'entrée du spectre par une lentille.

La lumière traverse le spectroscope et est détectée par le photomultiplicateur ou le PM. Par la suite le signal recueilli par le PM est amplifié par l'amplificateur discriminateur pour ensuite être traité par le Stanford SR430 qui est relié à l'ordinateur.

On se propose d'étudié chaque élément du montage pour mieux apprécier leur fonctionnement.

1.1.Fluorescence

La fluorescence est la propriété que possèdent des molécules d'absorber la lumière et de réémettre une lumière à certaines longueurs d'onde. Une molécule à l'état de repos occupe le niveau d'énergie fondamentale E_0. En excitant cette molécule à l'aide d'une source lumineuse à la fréquence ν_1, la molécule est excitée vers un niveau d'énergie E_1 tel que $E_1-E_0 = h\nu_1$. De plus la molécule ne va pas aller occuper exactement le niveau E_1, mais un niveau E_1' qui est un sous niveau de E_1; en effet les niveaux sont multiples et il y a des pertes au sein de la molécule. C'est à partir du niveau E_1' que la molécule se désexcite sur son niveau d'énergie fondamentale en émettant un photon à la fréquence ν_1' tel que $h\nu_1' = E_1' - E_0$.

1.2.Le laser NdYAG

Le milieu actif d'un laser à solide est composé d'un barreau cylindrique de matériau non conducteur, transparent à la longueur d'onde du laser et contenant une petite quantité d'impuretés. Le laser à solide le plus employé est le laser NdYAG. Le YAG (Yttrium Aluminum Garnet) est un cristal clair et dur, il est dopé par Nd^{3+} (Neodymium triply-ionized) et est pompé optiquement par des lampes flash à spectre large (du rouge à l'infrarouge). Il est composé d'une alimentation, d'un système de refroidissement, d'une tète laser et d'un système de synchronisation qui permet de déclencher le Q-Switch, qui est un résonateur qui contrôle la facteur de qualité de la cavité. Sa fréquence est de 10 HZ, soit un taux de répétition de 100 ms. La durée des pulses est de 7ns. La longueur d'onde la plus utilisée est 1064nm, un doubleur de fréquence permet une émission à 532nm.

1.3.Le spectromètre

Le spectromètre utilisé est un spectromètre à réseau de diffraction. Il sert à sélectionner les différentes raies d'une source lumineuse. Il est constituer d'une fente d'entrée et d'une fente de sortie de largeur réglables, de deux miroirs conjuguant l'entrée et la sortie et d'un réseau de 1800 traits/mm pivotant autour d'un axe de façon à sélectionner des longueurs d'onde grâce au phénomène de diffraction. Son schéma est présenté sur la figure 3. La lumière qui entre par la fente d'entrée se réfléchit sur un premier miroir et vient frapper le réseau de diffraction. Le faisceau est alors diffracté avec un angle différent pour chaque longueur d'onde. Le faisceau diffracté est réfléchit par un deuxième miroir qui renvoie la lumière vers la fente de sortie. Seul la longueur d'onde qui coïncide avec l'axe de la fente sort du spectromètre et son intensité peut être mesurée par un détecteur placé à sa sortie.

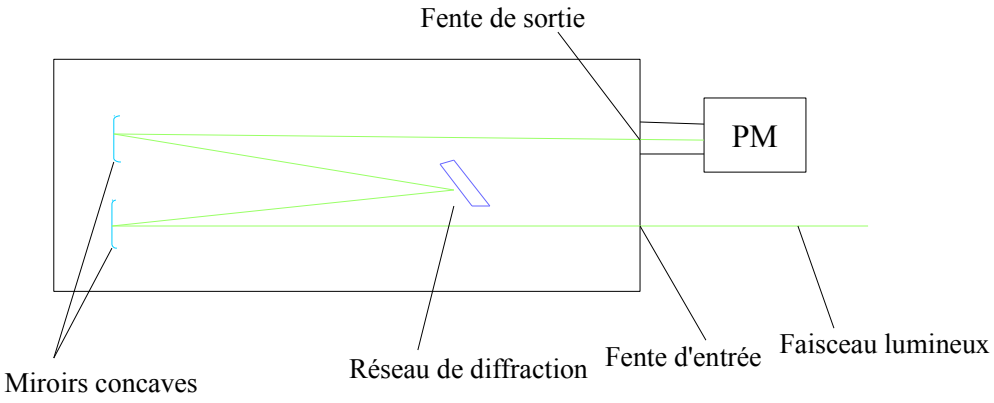


Figure 3 : Schéma de principe de spectromètre

La rotation du réseau se fait à l'aide d'un moteur à vitesse réglable muni d'un encodeur optique et la longueur d'onde sélectionnée est indiquée par un compteur. Le produit quantité de lumière-résolution ($\lambda / \Delta\lambda$) est une constante du spectromètre, une bonne résolution impose une restriction au niveau de l'intensité lumineuse qui y entre. De plus, les intensités émisent par fluorescence sont très faibles, il est nécessaire d'avoir un appareil de détection très sensible en sortie du spectromètre. Le détecteur utilisé ici est un photomultiplicateur ou PM.

1.4. Le photomultiplicateur ou PM

Le photomultiplicateur est un détecteur de lumière visible et ultraviolette. Il se situe à la fin du trajet optique et permet de détecter la quantité de lumière transmise par le spectromètre.

Le PM est composé d'une photocathode, d'un multiplieur d'électrons(composé de plusieurs dynodes) et d'une anode. Le modèle utilisé est un PM Hamamatsu modèle R943-02 (Voir l'annexe 1), il a un temps de monté de 3ns et peut supporter un courant maximale de sortie de 1µA .

Son mode de fonctionnement est simple, un photon vient frapper la photocathode du PM qui émet des électrons. Ces électrons sont multipliés à travers les dynodes et collectés par l'anode sous forme de pulse.

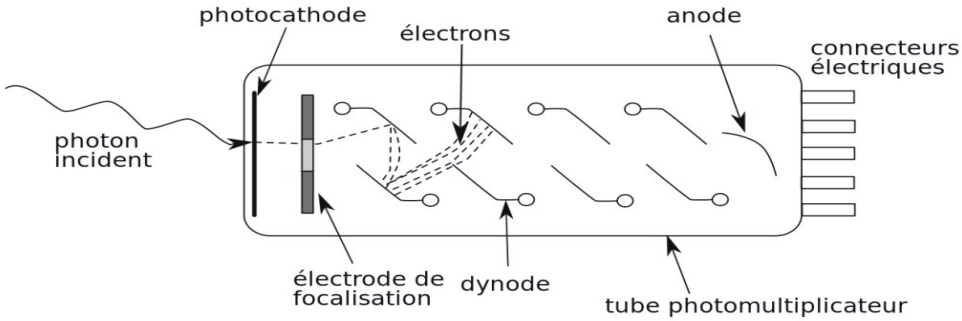


Figure 4 : Schéma de principe du PM

1.5.Amplificateur discriminateur

En sortie de PM les impulsions sont faibles en intensité et en tension, et extrêmement courtes en durée. Il est nécessaire de les amplifier et d'augmenter la largeur des impulsions pour qu'elles soient ensuite détectables par un compteur électronique. Le schéma électronique de l'amplificateur discriminateur est en annexe 2. Il est constitué de trois types de composants principaux.

1.6. LE Stanford SR430

Le stanford SR430 est le premier mesureur multi-canal qui associe des amplificateurs, des discriminateurs, des horloges, et l'analyse des données dans un instrument unique.
Avec ses nombreuses fonctionnalités et son interface piloté par menus facile à utiliser, le SR430 simplifie l'expérience de comptage de photons. Et vous trouverez sur l'annexe 3 les caractéristiques de Stanford SR430.

Le SR 430 compte les impulsions entrantes dans chaque horloges ce qui est équivalent à des petits intervalles de temps ou des bacs de temps (T_1,T_2,...,T_N) . Une impulsion de déclenchement démarre le cycle d'acquisition de données. La durée des bacs de temps est programmable de 5 ns à 10,5 ms. Le nombre d'impulsions de signal comptées pendant chaque bacs de temps est stockée dans la mémoire. Le bus d'adresse de la mémoire est alors incrémenté et le compteur est remis à zéro pour le prochain domaine d'horloge. Le résultat est un enregistrement de la somme ou moyenne de tous les acquisitions de chaque domaine d'horloge.

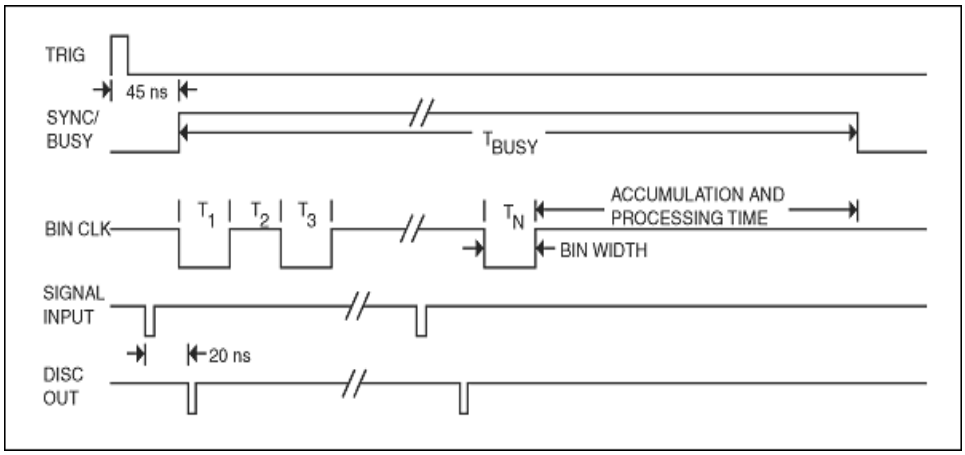


Figure 5 : Chronogramme de Stanford SR430

Voici ci-dessous un exemple de mesure de Stanford SR430,qui a été réaliser sur la manipulation de spectroscopie de fluorescence, pour mesurer des temps de vie.

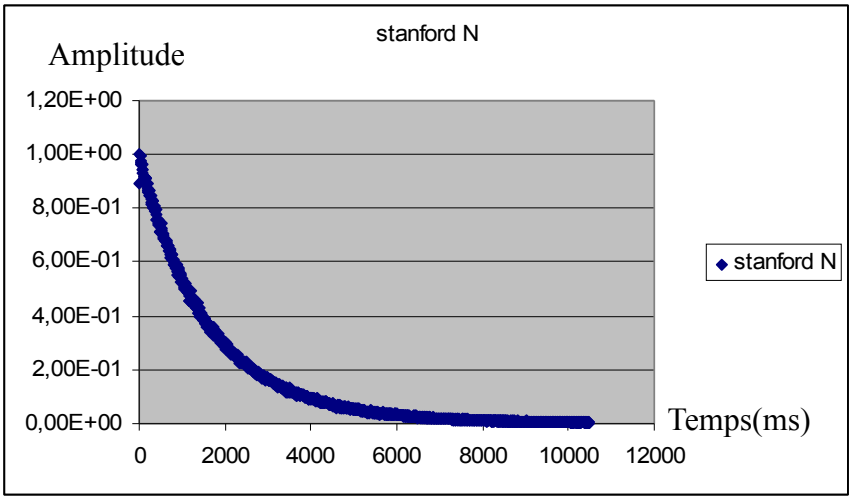


Figure 6 : Les temps de vie de photons mesurés par le Stanford SR430

2. Élément sur les statistiques de photons

Une source macroscopique de lumière produit un flux de photons lui-même macroscopiques, c'est-à-dire composé d'un très grand nombre de photons. Pour décrire cette lumière, il est généralement suffisant de se contenter d'une approche ondulatoire. Ainsi, si l'on envoie sur une lame semi-réfléchissante une impulsion de lumière classique, l'impulsion est divisée en deux impulsions identiques, l'une transmise et l'autre réfléchie, d'intensités égales à 50 % de l'intensité originale.

Si l'on place un photodétecteur sur le trajet de la lumière réfléchie et un autre sur le trajet de la lumière transmis, on observera deux signaux de photodétection coïncidant temporellement. Et donc la question que l'on se pose : que se passe-t-il si l'impulsion envoyée sur la lame semi-réfléchissante contient un seul photon ?, Ce photon sera en fait soit réfléchi, soit transmis, au hasard, avec une probabilité de 50 % pour chaque cas et on ne pourra donc jamais obtenir un signal sur chacun des photodétecteurs simultanément.

Considérant la lumière comme un flux de photons, on pourrait être tenté de penser que pour obtenir un photon unique, il suffit d'atténuer suffisamment une impulsion de lumière classique. En fait, il n'en est rien. Si l'on atténue une impulsion laser de façon à avoir un photon en moyenne dans chaque impulsion atténuée et que l'on envoie une telle impulsion sur une lame semi-réfléchissante, la probabilité d'obtenir un signal sur chacun des photodétecteurs simultanément (c'est-à-dire d'obtenir deux photons) est égale à la moitié de la probabilité d'en avoir un seul: un photon « en moyenne » est donc clairement très différent d'un photon « unique ».

Si l'impulsion est atténuée encore plus fortement, de façon à ce que le nombre moyen de photon devienne très inférieur à un, l'impulsion contiendra rarement un photon et encore plus rarement deux photons. Mais ceci n'est pas non plus une impulsion à un photon, puisque l'on a le plus souvent zéro photon au lieu d'un, et que la probabilité d'avoir deux photons n'est jamais nulle (alors qu'elle devrait toujours l'être si l'on n'avait qu'un seul photon).

On considère que l'arrivé de photons sur le détecteur, est un processus aléatoire ou stochastique, c'est-à-dire l'événement: « l'arrivée d'un photon sur le détecteur » a une probabilité p identique, de se produire. Et pendant un temps T grand, le nombre de photons qui arrivent sur le détecteur est constant et vaut N. La probabilité B(n,N,p) que n photons arrivent pendant un temps plus court Δt est la probabilité d'avoir n événements de probabilité p parmi N , soit:

(1) $B(n,N,p) = C(N,n)\, p^n\, (1- p)^{(N-n)}$; avec $C(N,n) = N! / n!(N-n)!$

C'est la loi binomiale.

Et parce que la distribution temporelle des événement est uniforme, on peut écrire que :

(2) $p = \Delta t /T$

Dans le cas de la statistique de poisson, on fait l'hypothèse que N tend vers l'infini, mais le produit Np reste constant. Et comme, $p = \Delta t /T$,cela revient à dire que le rapport N/T reste constant.

On pose alors : $\lambda = Np = N\ \Delta t /T$ (3)

Dans ce cas, on peut démontrer que la loi binomiale tend vers une nouvelle loi appelée statistique de poisson.

En remplaçant C(N,n) par sa forme générale dans l'expression (1), on aura donc :

$$B(n,N,p) = [(Np)^n/n!][(1-Np/N)^N][N!/(N-n)!N^n][(1 – Np/N)^{(-n)}]$$

En utilisant l'expression (3) :

$$B(n,N,p) = [(\lambda)^n/n!][(1- \lambda/N)^N][N!/(N-n)!N^n][(1 – \lambda/N)^{(-n)}]$$

Or, la limite de $(1+ \lambda/x)^x$ quand x tend vers l'infini est égale à $\exp(\lambda)$,donc si N tend vers l'infini on a $[(1- \lambda/N)^N]$ tend vers $\exp(- \lambda)$.

Par ailleurs, si N tend vers l'infini, on a aussi $[(1- \lambda/N)^{(-n)}]$ tend vers 1

Donc on peut écrire que si N tend vers l'infini, alors B(n,N,p) tend vers

$\longrightarrow$ $[\lambda^n/n!]\exp(- \lambda)\ [N!/(N-n)!N^n] = [\lambda^n/n!]\exp(- \lambda)\ F(n,N)$

avec $F(n,N) = N!/((N-n)!N^n)$

Il ne reste plus qu'à évaluer la limite de F(n,N) lorsque N tend vers l'infini, et pour cela, on part des premier termes de la formule de Stirling :

$$n! = 6{,}28\ [\exp(-n-1)].(n+1)^{\,n+(1/2)}\ .[1 + 1/(12(n+1)) +...]$$

Donc, on peut écrire F(n,N) sous la forme suivante :

$$F(n,N) = (1/N^{n}) . [e^{-n-1} .(N+1)^{n+(1/2)}] / [e^{-(N-n)-1} .(N-n+1)^{N-n+(1/2)}]$$

Or

$$(N-n+1)^{N-n+(1/2)} = e^{(N-n+(1/2)) \text{ Log}[N(1+(1-n)/N)]} = e^{(N-n+(1/2)) [\text{Log } N+\text{Log}(1+(1-n)/N)]}$$

Et si N est grand, alors

$\text{Log}[1+ (1- n)/N] \sim (1- n)/N$, donc :

$$(N-n+1)^{N-n+(1/2)} \sim e^{(N-n+(1/2)) [\text{Log } N + (1-n)/N)]} \sim N^{n} N^{-n+(1/2)} e^{1-n}$$

Et on a aussi de la même façon :

$$(N+1)^{n+(1/2)} \sim N^{n} N^{(1/2)} e$$

Donc, si N est grand :

$$F(n,N) \sim (1/N^{n}) . [e^{-n-1} N^{n} N^{(1/2)} e] / [e^{-(N-n)-1} N^{n} N^{-n+(1/2)} e^{1-n}] = 1$$

Donc, En conclusion, si N tend vers l'infini, alors :

$B(n,N,p) \longrightarrow [\lambda^{n}/n!]\exp(-\lambda) [N!/(N-n)!N^{n}] = [\lambda^{n}/n!]\exp(-\lambda) F(n,N) = [\lambda^{n}/n!]\exp(-\lambda)$

avec $\lambda = Np$

Donc, la loi binomiale, tend vers une statistique de poisson, donné par :

$$B(n,N,p) \longrightarrow P(n, \lambda) = \lambda^{n}/n!.\exp(-\lambda)$$

En conclusion, La nature produit en général de la lumière qui suit la statistique de Poisson conduisant aux nombres de photons par impulsion.

3. Présentation de la carte FPGA

Les FPGA (Field-Programmable Gate Array) sont des circuits intégrés reprogrammables. Ross Freeman, le co-fondateur de la société Xilinx, a inventé le premier FPGA en 1985.

Les FPGA se situent entre les réseaux logiques programmables et les circuits logiques près diffusés. Les réseaux logiques programmables sont des composants qui ne nécessitent aucune étape technologique supplémentaire pour être personnalisés, ce sont des circuits standards, programmables par l'utilisateur grâce aux différents outils de développement et qui incluent un grand nombre de solutions basées sur les variantes de l'architecture des portes ET et OU. Les prés diffusés sont des circuits intégrés basés sur l'utilisation des réseaux de cellules dont les blocs ont été préalablement diffusés.

Les FPGA combinent donc à la fois la souplesse de la programmation des réseaux logiques programmables et les performances des circuits prés diffuses. En d'autres termes le FPGA permet d'avoir une architecture conçue sur mesure à haute densité dans un circuit électronique, avec la possibilité de modifier cette architecture quand des nouvelles applications apparaissent.
Sur la figure 7, on trouve la carte FPGA utilisé durant ce travaille.

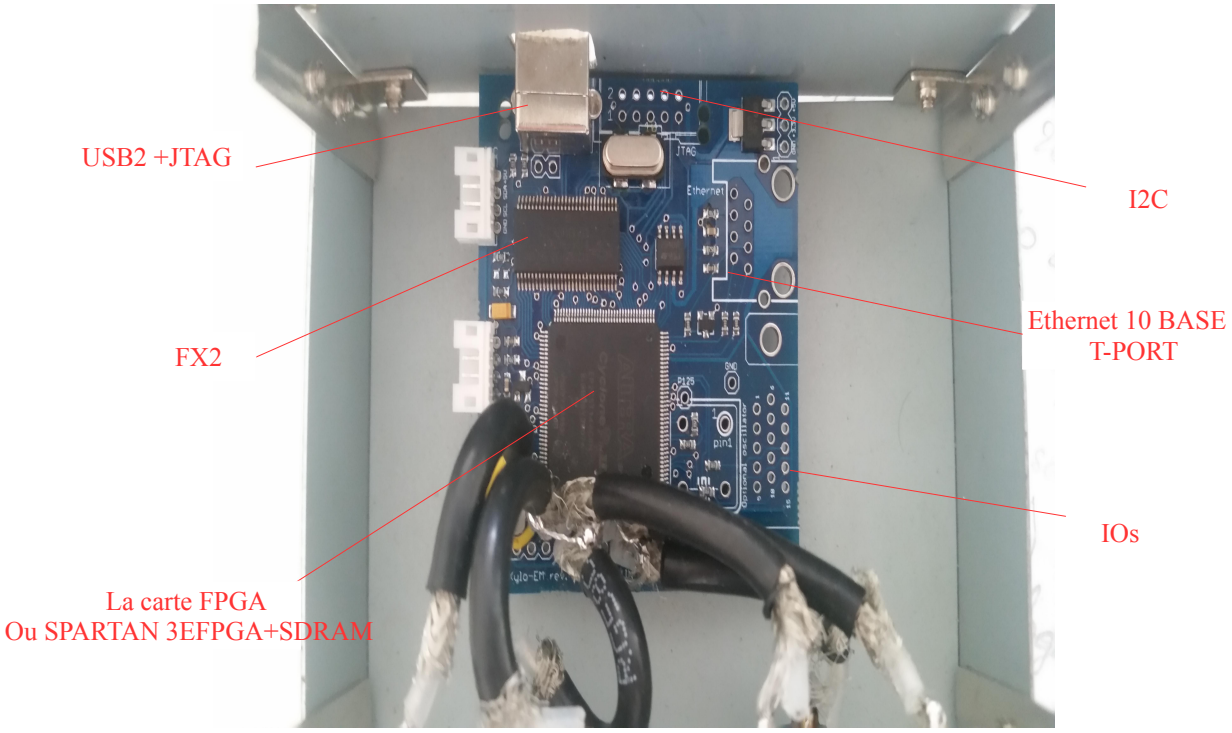


Figure 7 : Représentation de la carte graphique

Notre carte graphique comprend les éléments suivants et ce sont les plus importants :

JTAG ou JointTestAction Group : Le JTAG est conçue pour faciliter et automatiser le test des cartes électroniques numériques. Elle consiste à donner un accès auxiliaire aux broches d'entrée-sortie des composants numériques fortement intégrés. Et aussi il établie la communication entre le FPGA et le PC dans le cas d'une reconfiguration partielle du FPGA .

I2C : c'est le bus de données, il permet de relier facilement un microprocesseur ou des périphériques avec la carte FPGA, et sa fréquence de fonctionnement est de 100 ou 400 KHZ.

FX2: C'est une puce électronique, qui est utilisée pour fournir une interface USB rapide entre le FPGA et l'ordinateur. Il comprend une interface dite FIFO qui transfert directement les données entre le FX2 et l'interface externe.

Ethernet 10 BASE-T PORT : C'est une interface réseau

IOS : Ce sont des Entrées/Sorties standards on les appels aussi les broches utilisateurs ,et ces broches peuvent être programmées pour être On/entrées, In/sortie ou bi-directionnel.

SPARTAN 3EFPGA+SDRAM memory : Représente notre circuit intégré le FPGA, il est constitué d'un nombre déterminé de ressources prédéfinies comportant des interconnexions programmables qui permettent de mettre en œuvre un circuit numérique reconfigurable et des blocs d'E/S pour que le circuit puisse accéder au monde extérieur.

3.1. Architecture des FPGA

Dans ce paragraphe nous présentons les caractéristiques des FPGA, en particulier pour clarifier la façon et le mode de fonctionnement dont ils peuvent mettre en oeuvre la personnalisation du matériel via leur reconfiguration.
Il existe actuellement plusieurs fabricants de circuits FPGA, dont Xilinx et Altera qui sont les plus connus.
L'architecture de FPGA (figure 8), se présente sous la forme de deux couches distinctes, la première est la couche appelée circuit configurable et la deuxième est une couche de réseau mémoire SRAM.

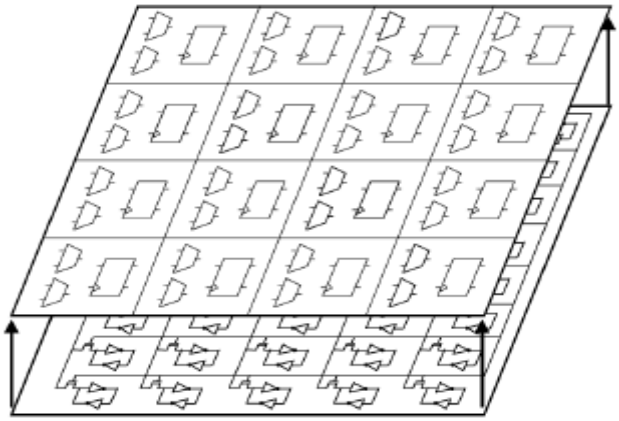


Figure 8 : Architecture de FPGA, la première est la couche appelée circuit configurable et la deuxième est une couche de réseau mémoire SRAM

La première couche (circuit configurable) est constituée d'une matrice de blocs logiques configurables (CLB – en anglais Configurable Logic Bloc). Les CLB permettent de réaliser des fonctions séquentielles et combinatoires . Autour de ces blocs logiques configurables, nous trouvons les blocs entrées/sorties (IOB – en anglais Input Output Bloc). Ils permettent de gérer les entrées-sorties pour réaliser l'interface avec les modules extérieurs.
La seconde couche est un réseau de mémoire SRAM qui permet la programmation du circuit FPGA .

La programmation est réalisée en appliquant les potentiels adéquats sur la grille de certains transistors à effet de champ pour interconnecter les éléments des CLB et des IOB afin de réaliser les fonctions souhaitées et d'assurer la propagation des signaux.
Ces potentiels sont mémorisés dans le réseau de mémoire SRAM . Un dispositif interne au FPGA

permet à chaque mise sous tension de charger les SRAM internes à partir de la ROM externe où est stockée la configuration du FPGA.

3.2. Architecture interne d'un FPGA

Les circuits FPGA utilisent deux types de cellules ou Blocs de base, les cellules d'entrées/sorties appelées IOB et les cellules logiques appelées CLB. Ces différentes cellules sont reliées entre elles par un réseau d'interconnexions configurables. Donc les trois blocs principaux sont les blocs logiques configurables, les blocs d'entrées/sorties et les ressources de communications (Figure 9).

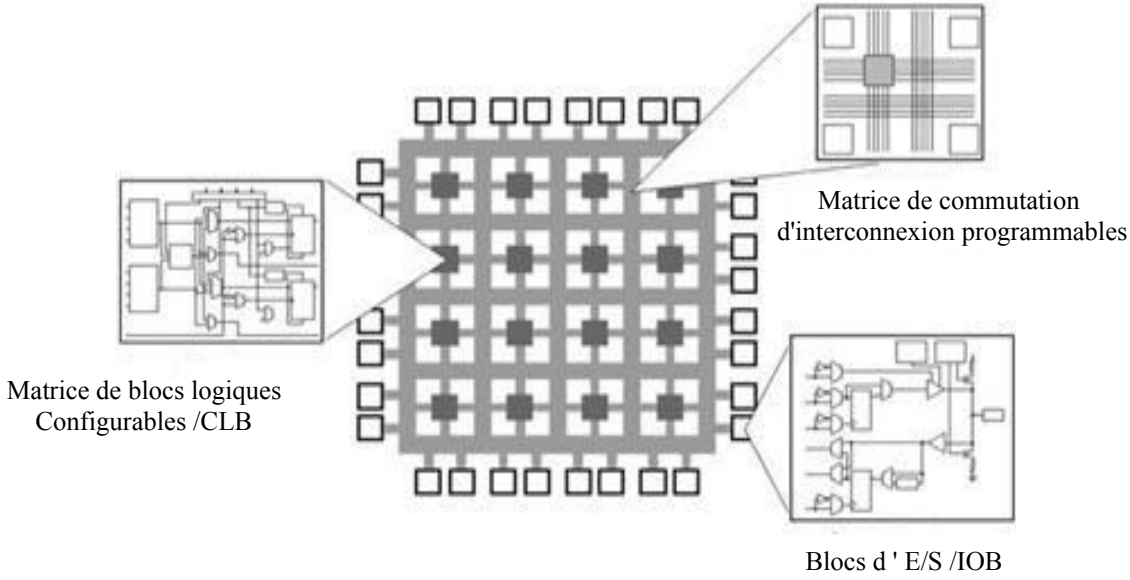


Figure 9 : Représentation des différents secteurs d'un FPGA

3.2.1. Les blocs logiques configurables (CLB)

Les blocs logiques configurables sont les principaux éléments d'un FPGA, et ils se composent de deux parties : des tables de correspondance (en anglais ,LUT), et des bascules (flip-flop).

L'ensemble de la logique combinatoire (ET, OU, NON ET,etc...) est implémentée sous forme de tables de vérités dans la mémoire LUT. Et comme on le sait une table de vérité est une liste prédéfinie d'états de sorties pour chaque combinaison d'entrée. Donc La fonction de la LUT est de stocker la table de vérité de la fonction combinatoire à implémenter dans la cellule.

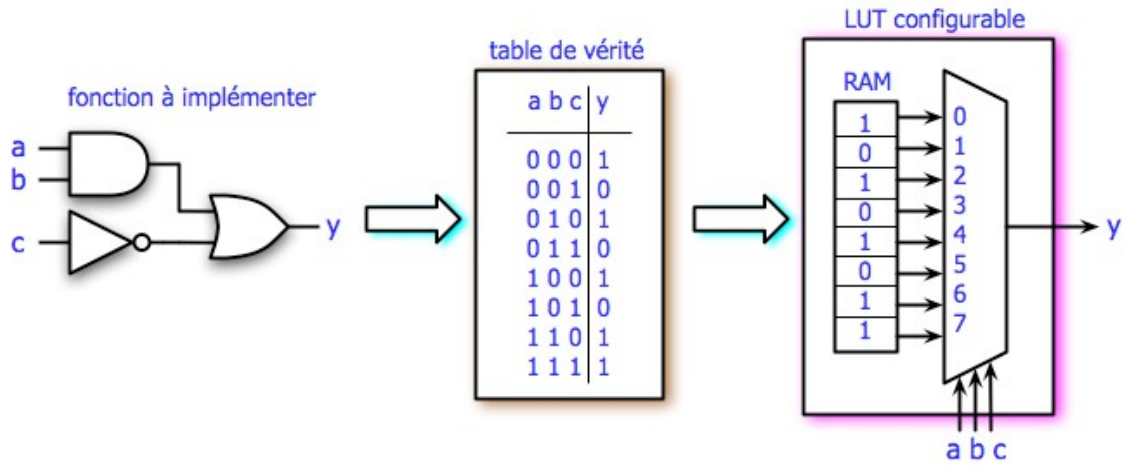


Figure 10 : Stockage de la table de vérité de la fonction à implémenté dans la cellule logique

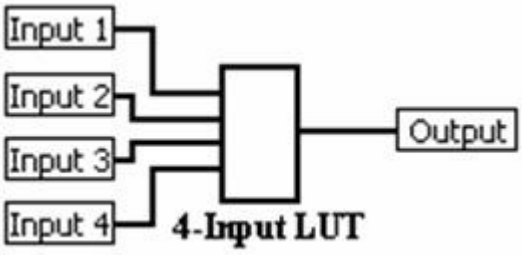


Figure 11 : Tables de correspondance à 4 entrées

Les bascules flip-flop sont des registres à décalage binaires qui servent à synchroniser la logique et à enregistrer les états logiques entre chaque cycle d'horloge au sein d'un circuit FPGA. À chaque front d'horloge, une bascule verrouille la valeur 1 ou 0 (VRAI ou FAUX) à son entrée et la maintient stable jusqu'au prochain front d'horloge .

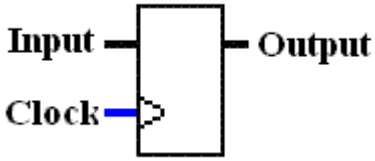


Figure 12 : Symbole de la bascule flip-flop

3.2.2. Les blocs d'entrées-sorties (IOB)

Les blocs d'entrées-sorties permettent l'interconnexion de la logique interne aux ports d'entrées et de sorties du FPGA. Les IOB ont leur propre mémoire de configuration, elles stockent les standards de tension et la direction des ports. Ces blocs sont présents sur toute la périphérie du circuit FPGA. Chaque bloc IOB contrôle une broche du composant et il peut être défini en entrée, en sortie, en signaux bidirectionnels ou être inutilisé.

3.2.3.Les ressources de communications

Les ressources de communications ou des interconnexions au sein d'un FPGA permettent la connexion arbitraire des CLB et des IOB. Les connexions internes dans les circuits FPGA sont composées de segments métallisés. Parallèlement à ces lignes, nous trouvons des matrices programmables réparties sur la totalité du circuit, horizontalement et verticalement entre les divers CLB. Elles permettent les connexions entre les diverses lignes, celles-ci sont assurées par des transistors MOS dont l'état est contrôlé par des cellules de mémoire vive ou RAM.

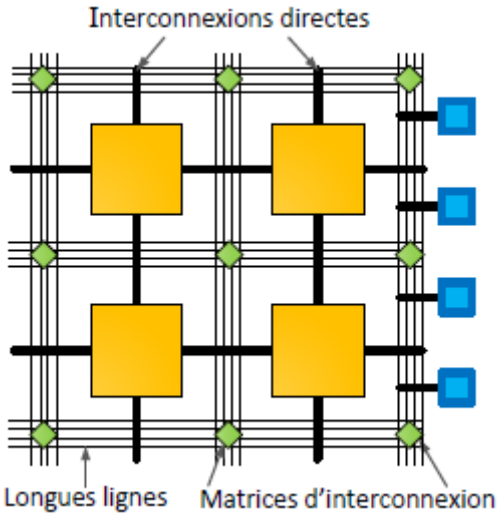


Figure 13 : Connexion interne d'un FPGA

En plus des trois blocs décrits dans le paragraphe précédent, les FPGA possèdent souvent ce que l'on appelle des ressources additionnelles.

La composition détaillée d'un circuit FPGA comme le montre la figure 14, illustre la disposition de ces ressources ou blocs additionnels sur un FPGA.

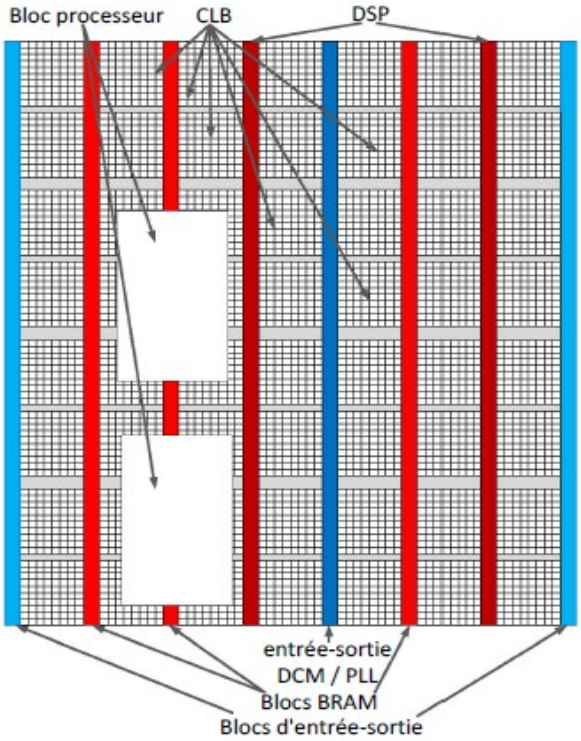


Figure 14 : Ressources d'un circuit FPGA

- Les DSP ou Les « Digital Signal Processing » sont des blocs qui permettent des conceptions plus complexes,qui peuvent consister soit en traitement numérique du signal ou seulement certains assortiments de multiplication, addition et soustraction .

Un bloc DSP permet de réaliser un multiplicateur, un accumulateur, un additionneur, et des opérations logiques (AND, OR, NOT, et NAND) sur un bit. Il est possible de combiner les blocs DSP pour effectuer des opérations plus importantes, telles que l'addition, la soustraction, la multiplication, la division, et la racine carrée. Et le nombre de blocs DSP est dépendant du dispositif.

- Les blocs processus sont l'un des ajouts les plus importants pour le FPGA. Beaucoup de conceptions nécessitent l'utilisation d'un processeur embarqué.

- DCM génèrent les fréquences d'horloge à partir du signal externe,dérivées de l'horloge interne ou principale à 48 MHZ (comme le cas de notre carte FPGA utilisé , l'horloge interne est à 48 MHZ), et ils sont basés sur des PLL(phase-locked loops) ou des DLL(digital delay-locked loop)

Les PLL sont des systèmes de commande qui génèrent la sortie de signal d'horloge ,dont la phase est en relation avec la phase du signal d'entrée, et les DLL se sont des dispositives électriques qui permettent de changer la phase du signal d'horloge .

La distribution du signal d'horloge se fait par le biais d'un arbre, minimisant ainsi les retards d'arrivée du signal aux flip-flops les plus éloignés .

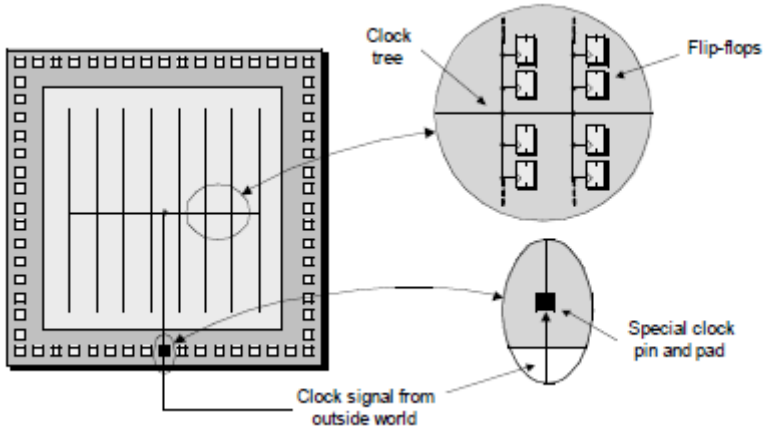


Figure 15 : Structure des flip-flops

3.2.4.Les Blocs RAM (SDRAM memory)

Les blocs RAM servent à stocker des ensembles de données. En fonction de la catégorie de FPGA, la mémoire RAM embarquée est configurable en blocs de 16 ou de 32 kilo-octets. Les mémoires RAM sont de type double ports, elles permettent une écriture et une lecture indépendante sur chaque port avec une horloge différente.

Le logiciel graphique de conception NI LabVIEW, modifie les règles de la programmation sur la carte FPGA avec de nouvelles technologies qui permettent de convertir le code graphique de la carte en circuits matériels numériques.

4. Présentation de LabVIEW

Le langage graphique LabVIEW (Laboratory Virtual Instrument Engineering Workbench), est un logiciel d'instrumentation qui permet la mesure des processus physiques de la même façon que les instruments classiques (oscilloscope,voltmètre,etc…), et qui est doté d'une partie programmation permettant le contrôle et l'acquisition de données.

Le temps nécessaire à l'assemblage d'un système de mesure ou de contrôle sous LabVIEW est généralement négligeable par rapport à celui nécessaire à la programmation dans un langage classique (Pascal,C/C++,etc…).

De plus, couplé à des entrées-sorties, il permet de gérer des flux d'informations numériques ou analogiques et de simuler des appareils de laboratoire.

Les programmes de mesures programmés sous LabVIEW peuvent se comparer aux instruments de mesure utilisés en laboratoire, de ce fait on peut parler d'Instrument Virtuel (ou VI). L'utilisateur manipule des instruments virtuels comme s'il s'agissait d'instruments réels.

4.1.Structure d'un projet LabVIEW

Un projet LabVIEW est une coquille qui contiendra tous les objets nécessaires à une application donnée (Instrument Virtuel VI, Statechart, …).
L'extension d'un projet est « .lvproj ».
Ouvrir un projet Labview permet un accès et une navigation rapide à l'ensemble de son contenu.

Un projet Labview pourra principalement contenir :

- Un Instrument Virtuel « VI » composé d'une face avant et d'un diagramme.
- Un diagramme d'états (Statechart) permettant de programmer un système séquentiel.
- Des variables partagées entre plusieurs ordinateurs ou systèmes du réseau.

4.2.Structure d'un VI

Le VI comporte une interface utilisateur, ou face-avant, avec des commandes et des indicateurs. Les commandes sont des boutons rotatifs, des boutons-poussoirs, des cadrans et autres mécanismes d'entrée. Les indicateurs sont des graphes, des LED et d'autres afficheurs de sortie.

Après avoir construit la face-avant, vous créez un diagramme. Ce diagramme contient le code de programmation contrôlant les objets de la face avant.

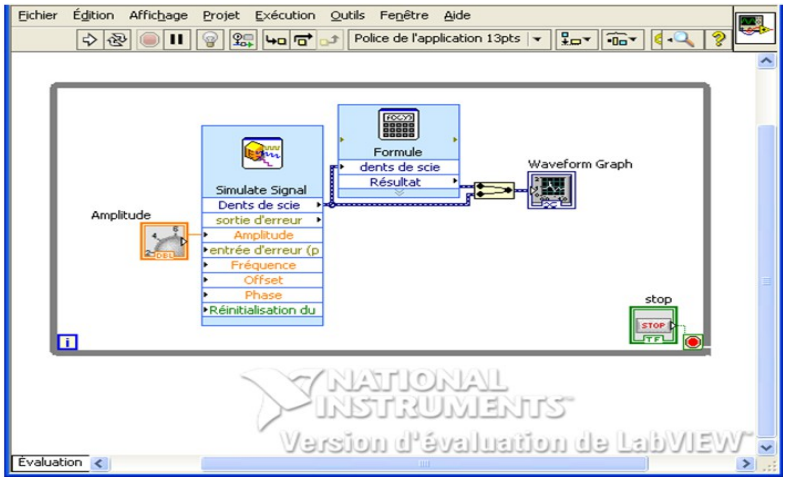


Figure 16 : Exemple de diagramme

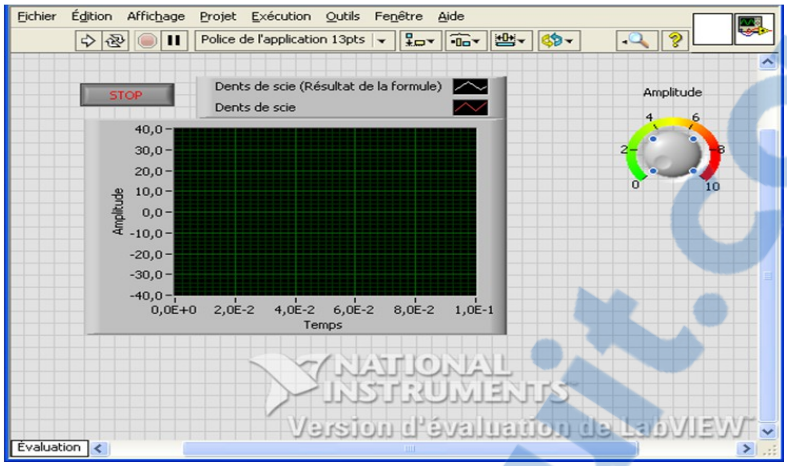


Figure 17 : Exemple de la face-avant

5. Réalisation du projet

avant de commencer nous allons créer un fichier DLL, qui est une bibliothèque de liens dynamiques, c'est-à-dire un fichier destiné pour communiquer avec la carte FPGA, et lui fournir des fonctions. Sur le diagramme de logiciel LabVIEW, et dans la palette des fonctions, il y a une fonction s'appelle « appeler une fonction d'une DLL ».

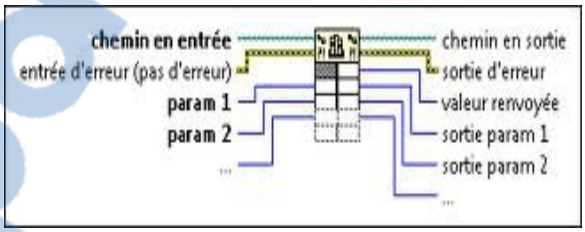


Figure 18 : Symbole de la fonction DLL

Bien qu'on puisse spécifier la bibliothèque partagée à appeler par nom ou par chemin, ces techniques utilisent des algorithmes de recherche différents pour trouver la bibliothèque partagée et ont différentes ramifications pour distribuer les bibliothèques partagées avec des applications autonomes.

- La fonction DLL se composent des éléments suivants :

chemin en entrée : Il permet d' identifie le chemin ou le nom de la bibliothèque partagée que vous voulez appeler. Et il faut sélectionner l'option "spécifier le chemin sur le diagramme" dans la boîte

de dialogue "Appeler une fonction d'une DLL ", pour que cette entrée apparaisse sur le connecteur.

entrée d'erreur : Cette entrée décrit les erreurs survenues avant l'exécution de ce nœud. Cette entrée fournit la fonctionnalité entrée d'erreur standard.

param 1..n : Ces paramètres représente des exemples de paramètres en entrée pour la fonction de bibliothèque.

chemin en sortie : Le chemin en sortie renvoie le chemin de la DLL ou de la bibliothèque partagée appelée. Il faut sélectionner l'option Spécifier le chemin sur le diagramme dans la boîte de dialogue Appeler une fonction d'une DLL pour que cette sortie apparaisse sur le connecteur.

sortie d'erreur : Il contient des informations sur l'erreur. Cette sortie fournit la fonctionnalité sortie d'erreur standard.

valeur renvoyée : C'est un exemple de valeur renvoyée pour une fonction de bibliothèque.

param 1..n en sortie : Représente des exemples de paramètres en sortie pour la fonction de bibliothèque.

5.1.Comment créer une DLL

L'utilisation d'une bibliothèque est un excellent moyen de réutiliser le code. Au lieu d'implémenter les mêmes routines dans chaque programme que nous créons .
En plaçant du code dans la DLL, on économise de l'espace dans chaque application , et on peut mettre à jour la DLL sans recompiler toutes les applications.
Dans la suite je vais vous montrer comment faire des dll, pour exporter une classe.

Lancer le logiciel de programmation Visual C++ ,et créez un nouveau projet, de type Projet Win32 comme ci-dessous :

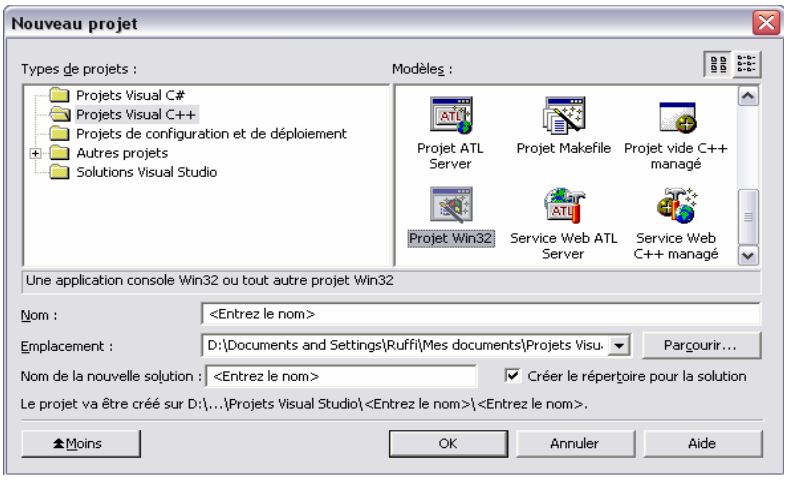


Figure 19 : Création de projet Win32

Puis sélectionnez DLL ,et Projet vide

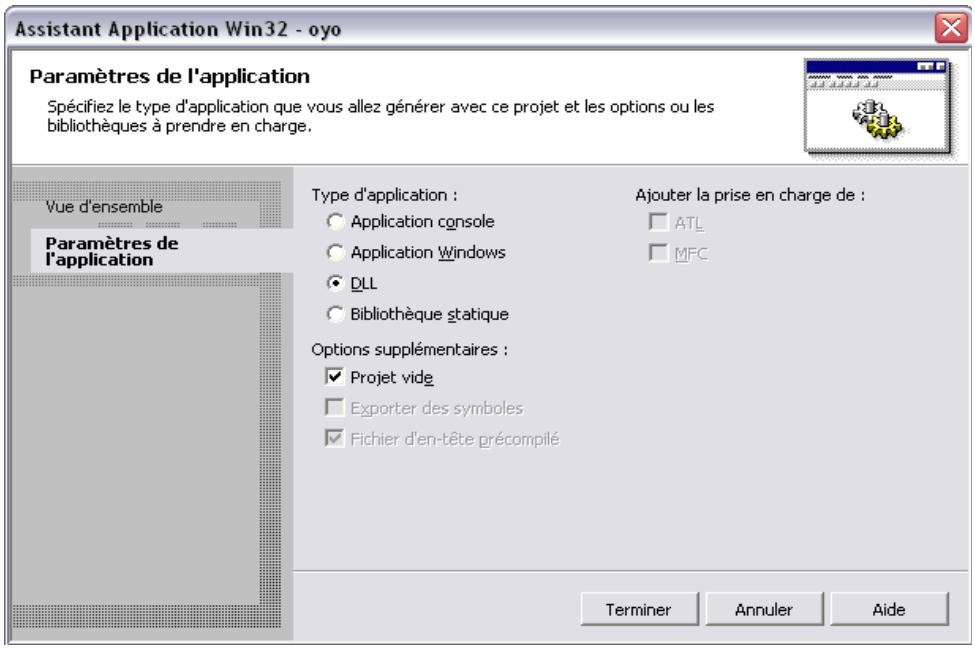


Figure 20 : Sélection de DLL et projet vide

Par la suite on aura deux fichiers, un fichier.h et un fichier.cpp.
Et voici la forme générale du fichier.h qu'on a recopié sur un exemple qu'on a fait :

```
#ifndef __MAIN_H__
#define __MAIN_H__

#include <windows.h>

/*  To use this exported function of dll, include this header
 *  in your project.
 */

#ifdef BUILD_DLL
   #define DLL_EXPORT __declspec(dllexport)
#else
   #define DLL_EXPORT __declspec(dllimport)
#endif


#ifdef __cplusplus
extern "C"
{
#endif
```

```
void DLL_EXPORT MaFunction(int taille,float c)          /* Ici on définit notre fonction
                                                           et les variables */

#ifdef __cplusplus
}
#endif

#endif // __MAIN_H__
```

Et voici le code du fichier.cpp

```
#include "main.h"

// a sample exported function
void DLL_EXPORT MaFunction(int taille,float *c);  /*Ici il faut mettre la même fonction
                                                    qu'on vient de définir avec
                                                    les mêmes variables */
{
   int i ;
   for(i=0;i<taille;i++)
   {
      c[i]=3*i;
   }



}
```

Et voici un autre exemple qui donne la somme des deux nombre , et les codes pour les deux fichiers

Code pour le fichier.h

```
#ifndef __MAIN_H__
#define __MAIN_H__

#include <windows.h>

/*  To use this exported function of dll, include this header
 *  in your project.
 */

#ifdef BUILD_DLL
   #define DLL_EXPORT __declspec(dllexport)
#else
   #define DLL_EXPORT __declspec(dllimport)
#endif
```

```
#ifdef __cplusplus
extern "C"
{
#endif

void DLL_EXPORT MaFunction(int a,int b,float *c);

#ifdef __cplusplus
}
#endif

#endif // __MAIN_H__
```

Le code pour le fichier.cpp

```
#include "main.h"

// a sample exported function
void DLL_EXPORT MaFunction(float a,float b ,float *c)
{
   *c=a+b;
}
```

Et sur le logiciel LabVIEW on ouvre un nouveau projet, on choisit VI vide, on aura les deux fenêtres de VI qui s'ouvrent et on clique droite sur la fenêtre de diagramme on choisit "appeler une fonction d'une DLL". Après en cliquant deux fois sur la même fonction on spécifie le chemin ou nom de la bibliothèque, et aussi on choisit les paramètres de nos variables.

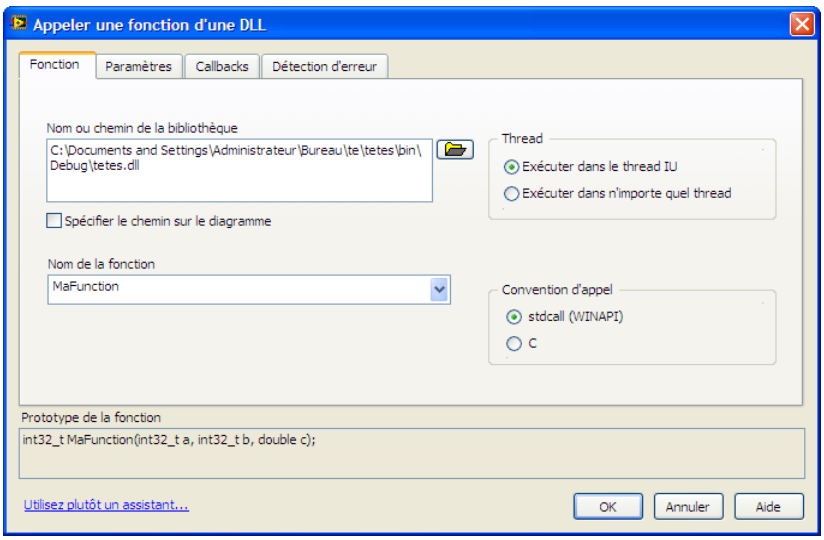


Figure 21 : Spécification du chemin ou le nom de la bibliothèque, et il faut faire attention à garder le même nom de la fonction, ici sur notre exemple c'est « MaFunction »

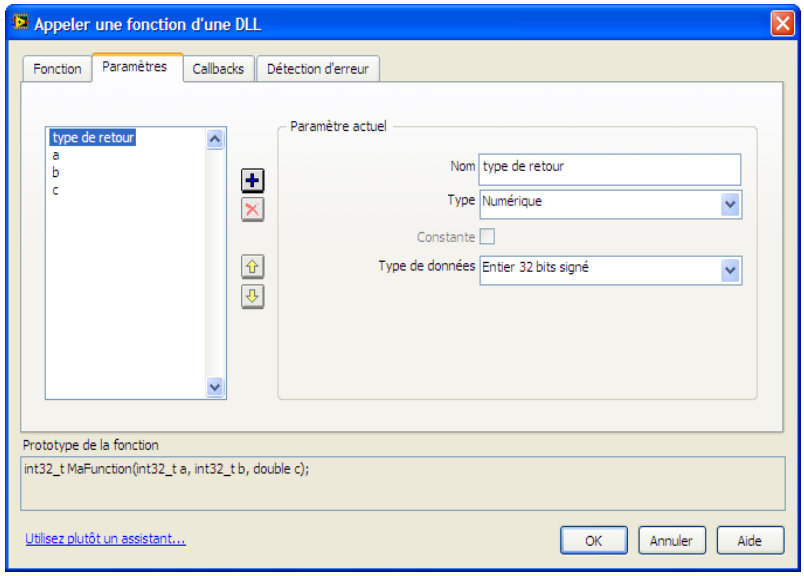


Figure 22 : Le choix des paramètres et le type des variables ou des données

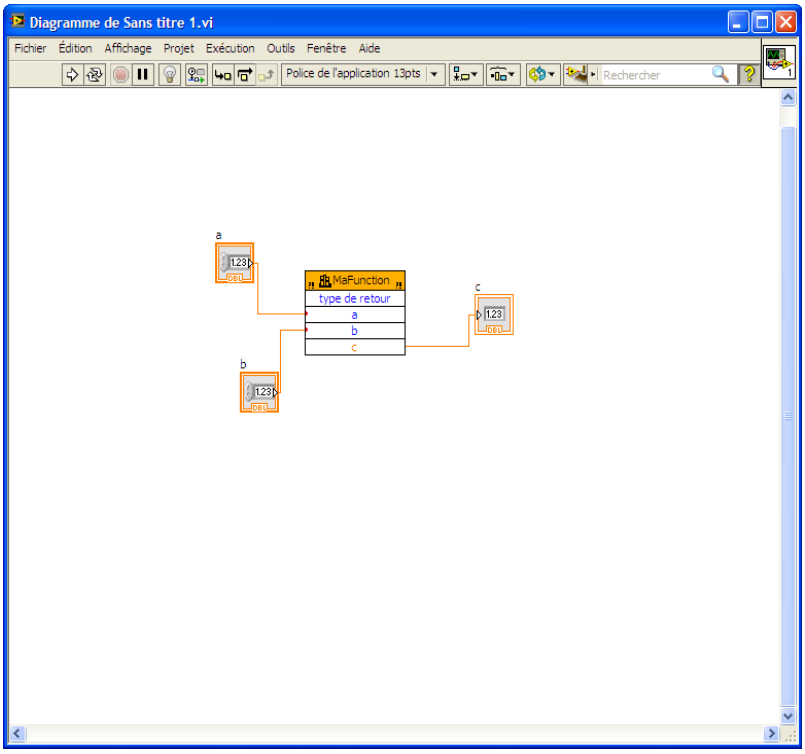


Figure 23 : Représentation du VI de notre exemple

Et vous trouverez sur l'annexe 4, le code de la DLL utilisé durant notre travaille pour faire le comptage de photons.

Et pour le faire fonctionner, il faut copier le .DLL et le .lib généré dans le projet de dll dans celui de notre application, et par la suite il faut lier la lib dans le compilateur. Et il faut le faire bien pour toutes les configurations.

5.2. Création de VI LabVIEW FPGA

Après avoir décrit la carte graphique et les éléments de base qui constituent le FPGA, et comment la carte FPGA communique avec le LabVIEW. Maintenant on montre comment implanter le VI dans le FPGA .

À l'installation de la carte graphique via le port USB2 ,Un « wizard » permet de créer une ébauche de projet FPGA. On détecte automatiquement la cible et ses accessoires.
Une fois fini, le projet contient la description complète du matériel de la cible (E/S, horloges,etc…) .

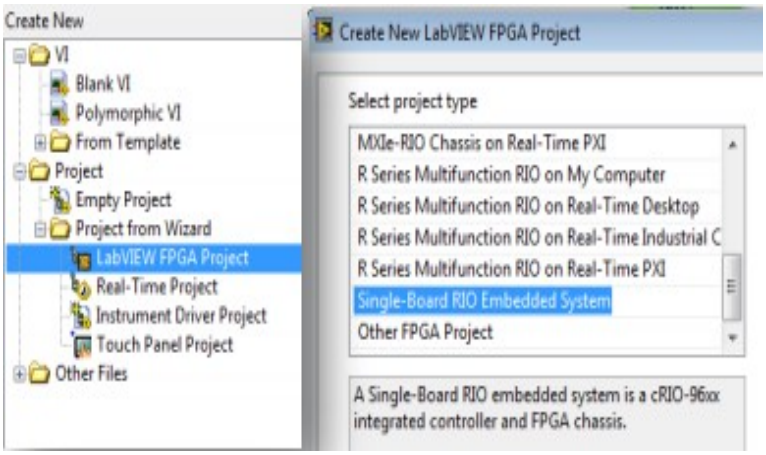


Figure 24 : Création de projet FPGA

La carte est équipée de quatre connecteurs permettant d'accéder aux 10 ports de 11 lignes. Chaque port est composé de 10 lignes nommées Portx/DIOy et d'une ligne Portx/DIOCTL. Les premières sont des lignes rapides routées ensembles et appariées en longueur, et les autres lignes sont destinées à des E/S lentes . Sous le nom Portx/Dio9 :0 sont regroupées les dix lignes d'un port. Chaque ligne peut être renommée (F2), et les lignes d'un même type regroupées dans un répertoire virtuel (Figure 25).

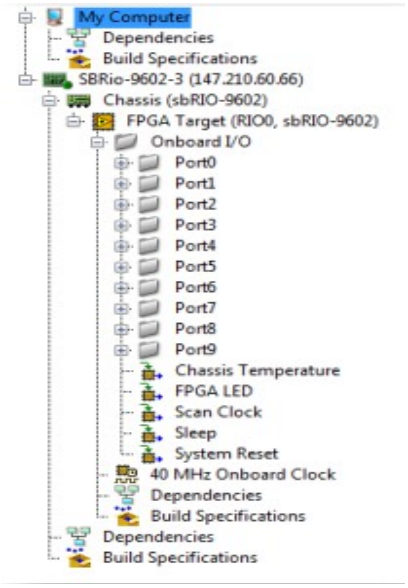


Figure 25-a : Regroupement des lignes dans un répertoire virtuel

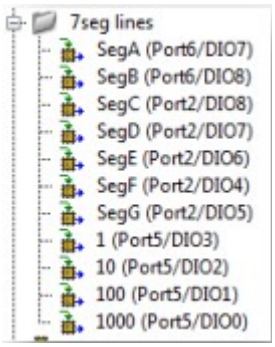


Figure 25-b : l'ensemble des E/S de la cible

Une application LabView FPGA contient en général trois parties :

a) Les VIs implémentés sur le FPGA.
b) Les VIs implémentés sur le processeur de système d'exploitation destinée à servir en temps réel les données de processus d'application .
c) Les VIs implémentés sur le PC de supervision.

Pour créer un VI destiné à être exécuté sur le FPGA il faut pointer l'icône de la cible FPGA avant de choisir « New ⟹ VI » dans le menu contextuel (Figure 26) .

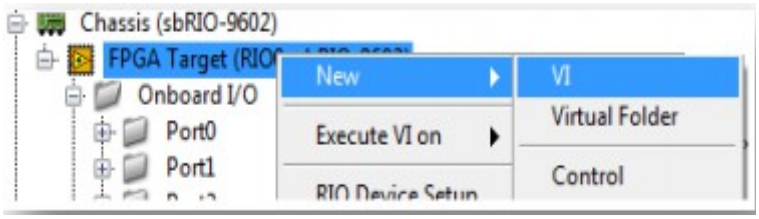


Figure 26 : Création d'un nouveau Vi FPGA

Avec un clic droit sur la face avant ou le diagramme, on a la palette des fonction de FPGA, il est plus réduite que la palette normale de VI .

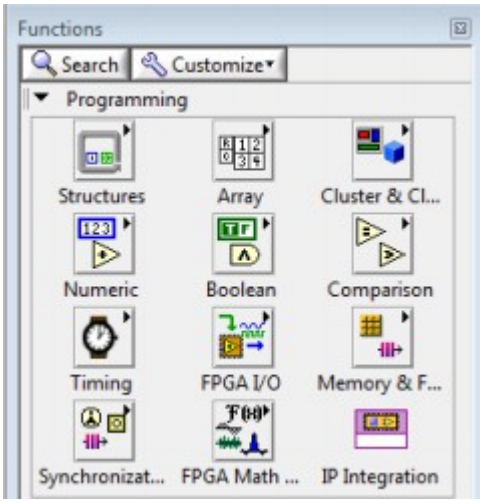


Figure 27 : Palette des fonctions de FPGA

Le VI LabVIEW-FPGA comporte 3 boucle, une boucle d'entrée(noté FPGA open), une boucle de sortie (FPGA close) et une boucle qui interface le FPGA (FPGA interface) et on peut regarder ça sur la DLL qui se trouve sur l'annexe 4, et de coup il y aura 3 sous VI.

La communication entre les variables du code FPGA et des variables de système d'exploitation peuvent se faire Par des FIFO (en français PEPS,première entrée-première sortie), comme le montre la figure ci-dessous :

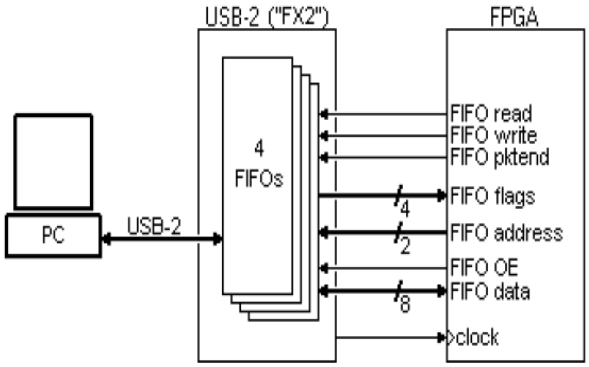


Figure 28 : Le FX2 est connecté au FPGA par un bus de données de 8 bits

FIFO address	FIFO accessed
00	FIFO2
01	FIFO3
10	FIFO4
11	FIFO5

Figure 29 : La liste d'adresses FIFO

Et sur l'annexe 5, vous trouverez la liste complète des signaux utilisés en interne par le FPGA pour accéder aux FIFO.

L'ordinateur communique avec le FPGA via le port USB2, on utilisant un pilot CyUSB signé pour interfacer notre système électronique et le rendre accessible depuis l'ordinateur.
Et à partir de notre DLL (Annexe 3), le protocole de communication USB se compose de cette structure de base :

- Une demande envoyée par notre ordinateur (envoie un octet via FIFO2, qui est utilisé pour un contrôle primitif du FPGA (Annexe 5).)
- Une réponse du FPGA (répond avec 2 octets via FIFO5, rapporte la longueur des informations collectées en octets)
- les données d'une longueur totale déclarée à l'étape 2 est alors disponible pour téléchargement via FIFO4

Un programme FPGA est en général une collection de boucles " While " tournant en parallèle pour gérer des flux d'entrées/sorties de périphériques en effectuant au passage des opérations logiques ou de calculs.

5.2.1.Sous VI FPGA-open

Cette fonction initialise le protocole de communication USB, entre l'ordinateur et le FPGA. FPGA-Open.VI (figure 30) renvoie un code erreur. Le 0 indique un fonctionnement correct, tandis qu'un -1 indique que la connexion USB a déjà été lancé. Et ce fonctionnement est géré par un gestionnaire d'erreurs qui délivre un message d'erreur décrivant la fonction "error", comme c'est montré sur la page suivantes (figure 31) .

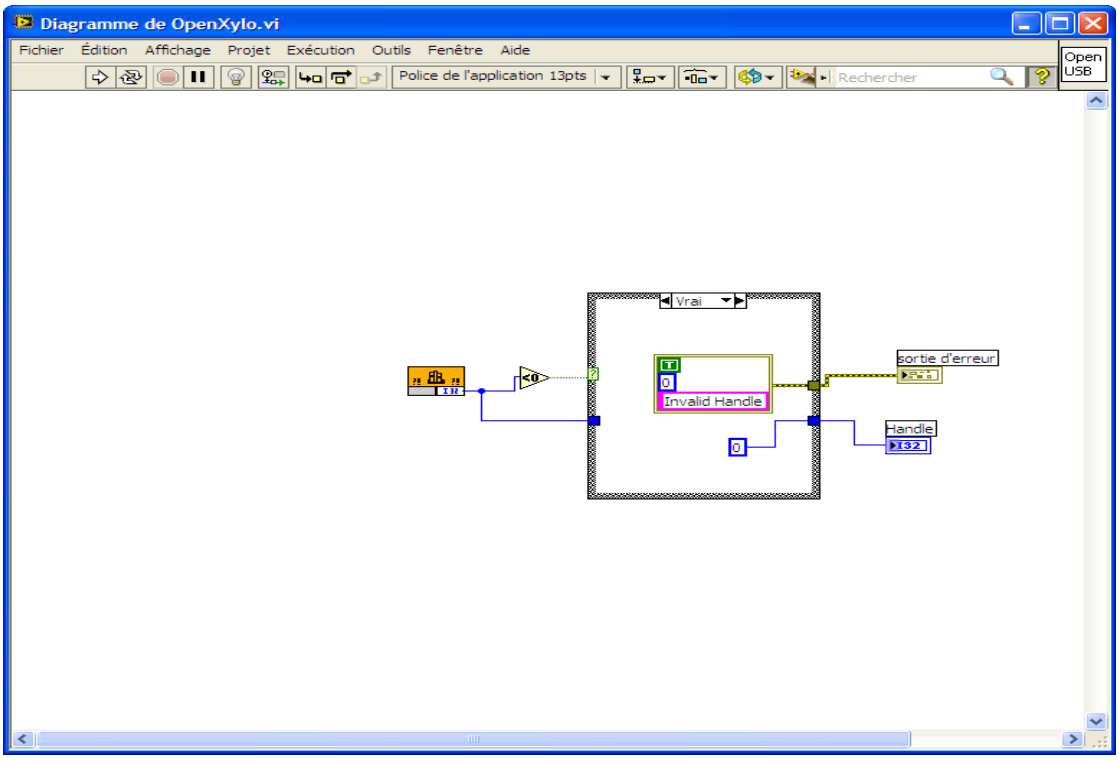


Figure 30 : Diagramme FPGA-OPEN.VI

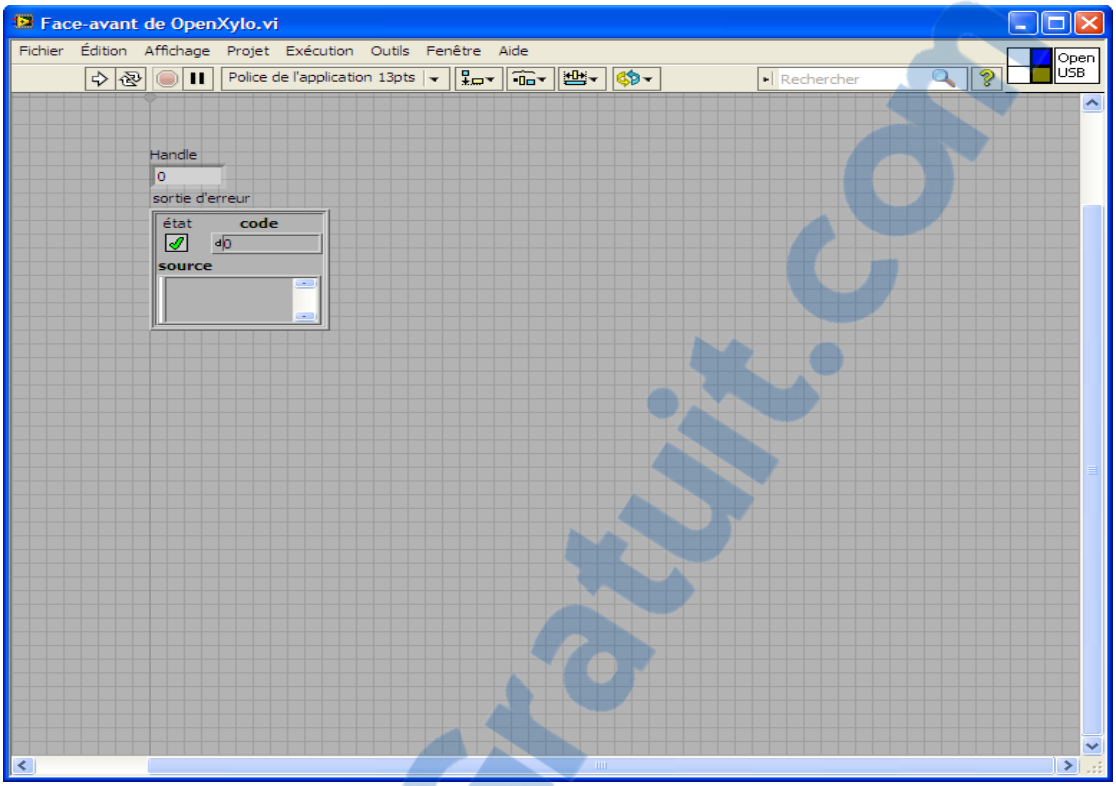


Figure 31 : Face avant FPGA-OPEN.VI

5.2.2.Sous VI FPGA-Close

Cette fonction ferme le protocole de communication USB (figure 32 et 33). Le 0 indique un fonctionnement correct, tandis qu'un -2 indique que la connexion USB n'a jamais été initié. Ceci est géré par un gestionnaire d'erreur qui délivre un message d'erreur décrivant l'erreur.

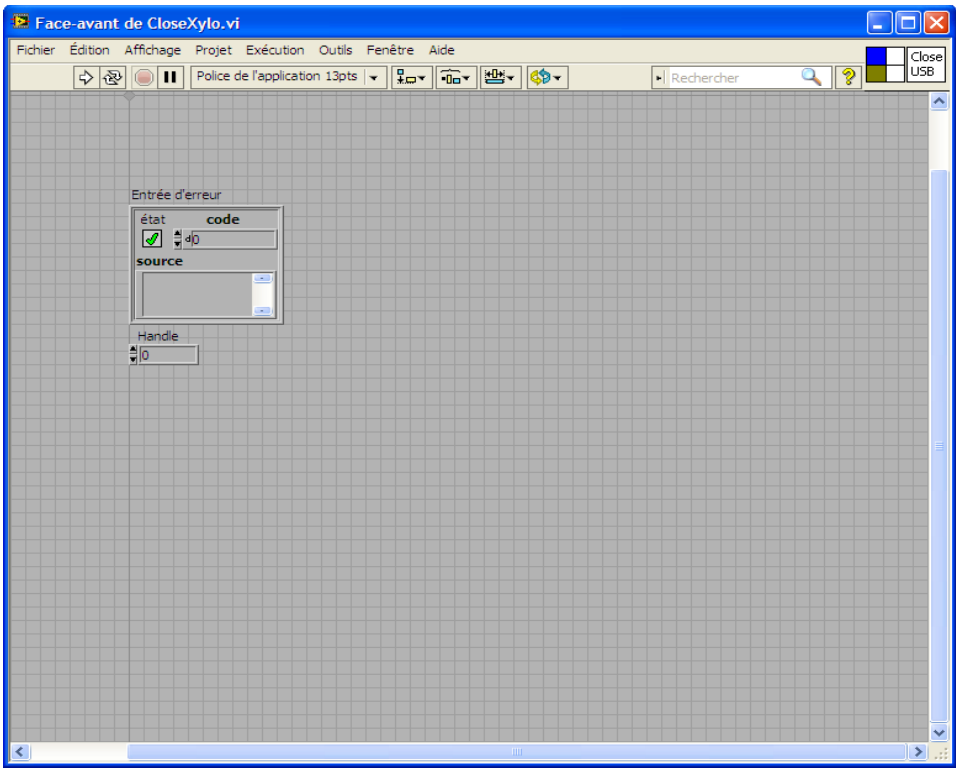


Figure 32 : Face avant FPGA-Close.VI

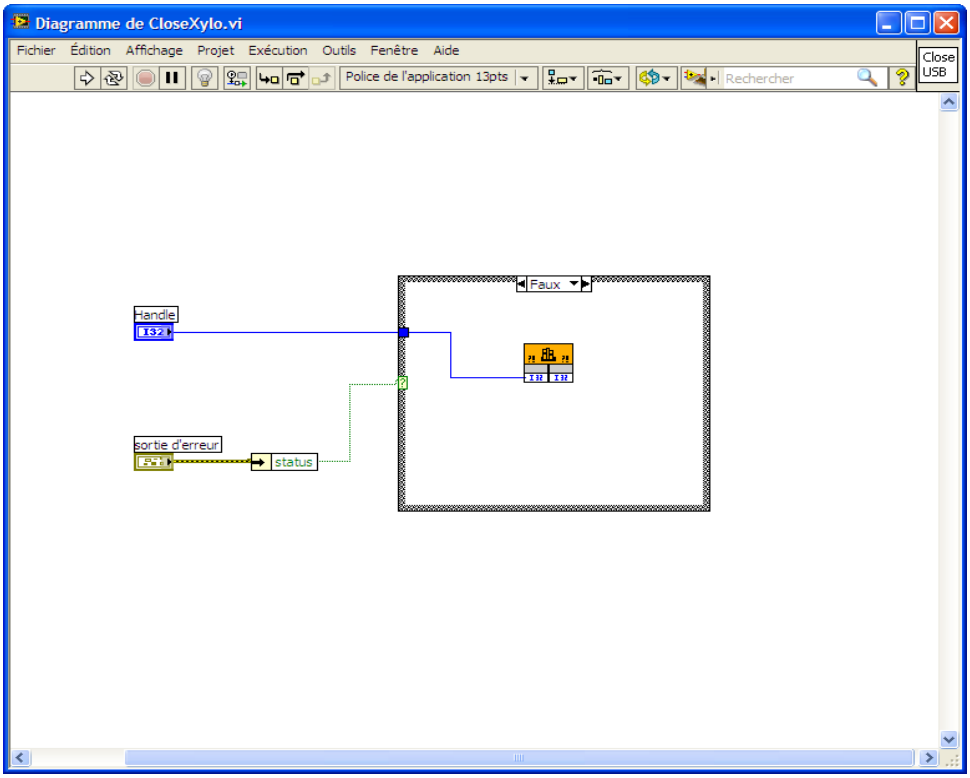


Figure 33 : Diagramme FPGA-Close.VI

5.2.3. Sous VI FPGA-Interface

Ce VI (figure 35) communique avec le FPGA en enregistrant les statistiques des événements de chaque domaine d'horloge , ainsi que le nombre de cycles d'horloge écoulés et les statistiques de coïncidence.

Un message est envoyé pour interroger la DLL(Annexe 4), et enregistrer les donnée à l'aide des tampons(Buffers) internes du FPGA, afin d'afficher les résultats de comptage pour chaque petits intervalles de temps (figure 36).

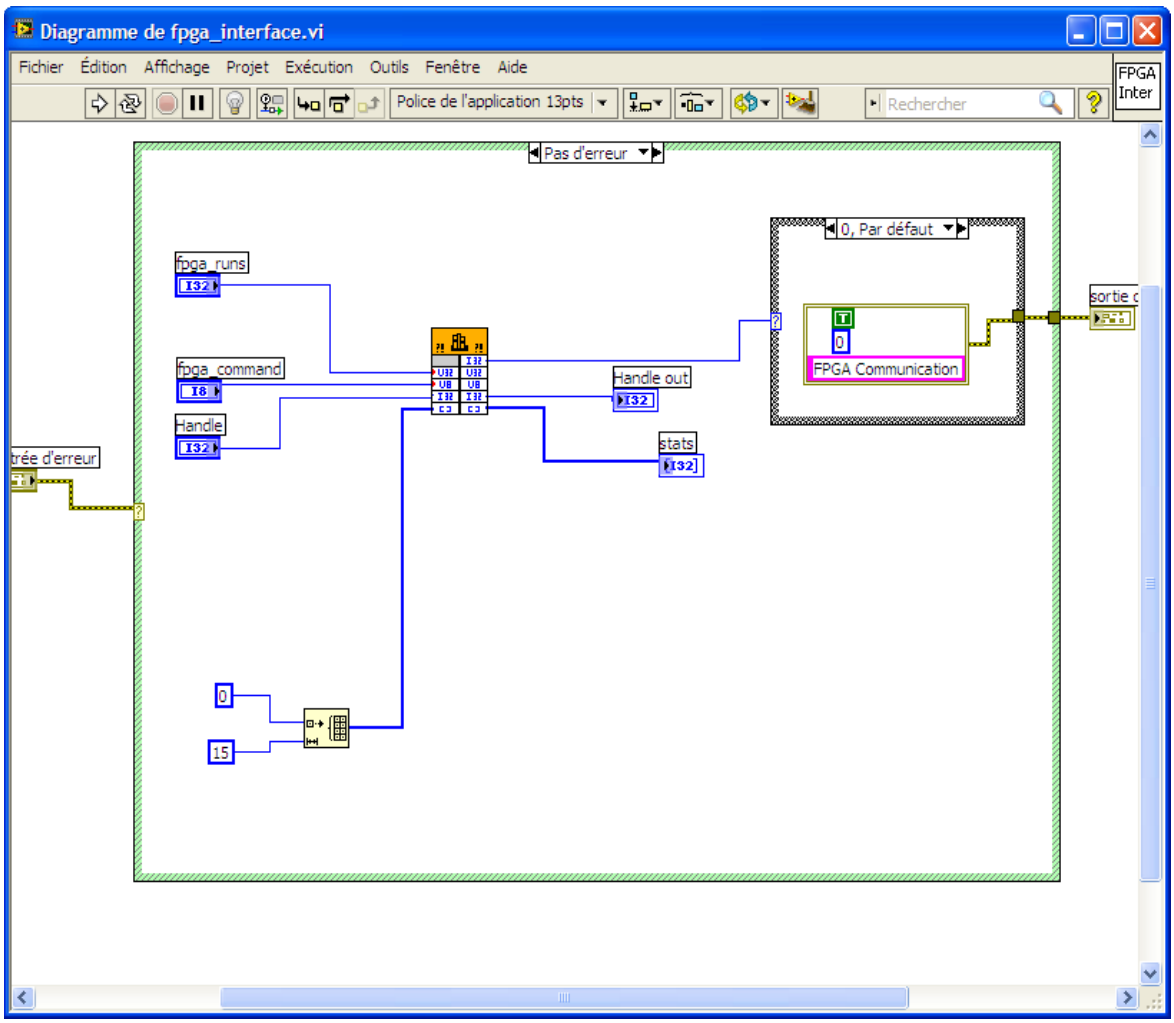


Figure 35 : Diagramme FPGA-Interface.VI

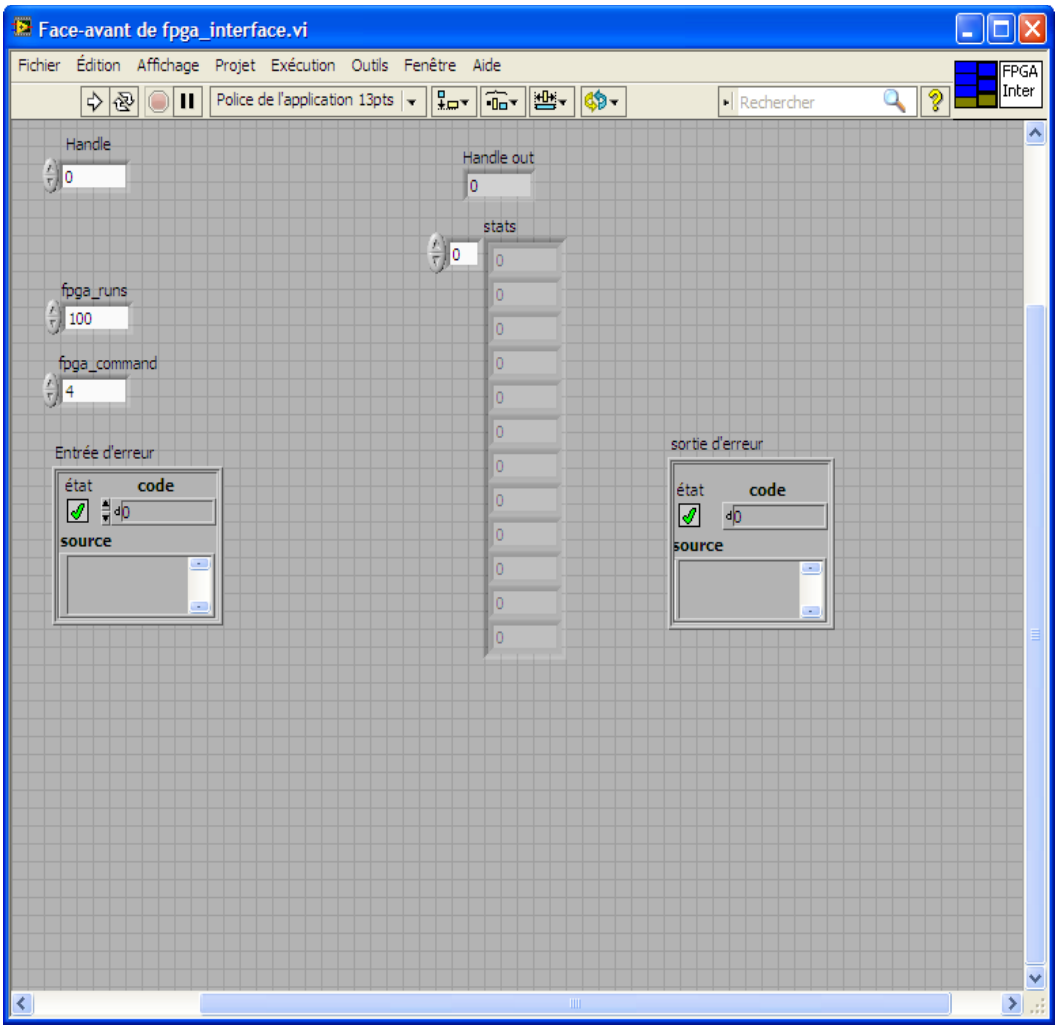


Figure 36 : Face avant FPGA-Interface.VI

5.2.4. VI LabVIEW-FPGA

Notre VI LabVIEW-FPGA ,regroupe les trois sous VI, FPGA-Open, FPGA-Close et FPGA-Interface (figure 37 et 38).

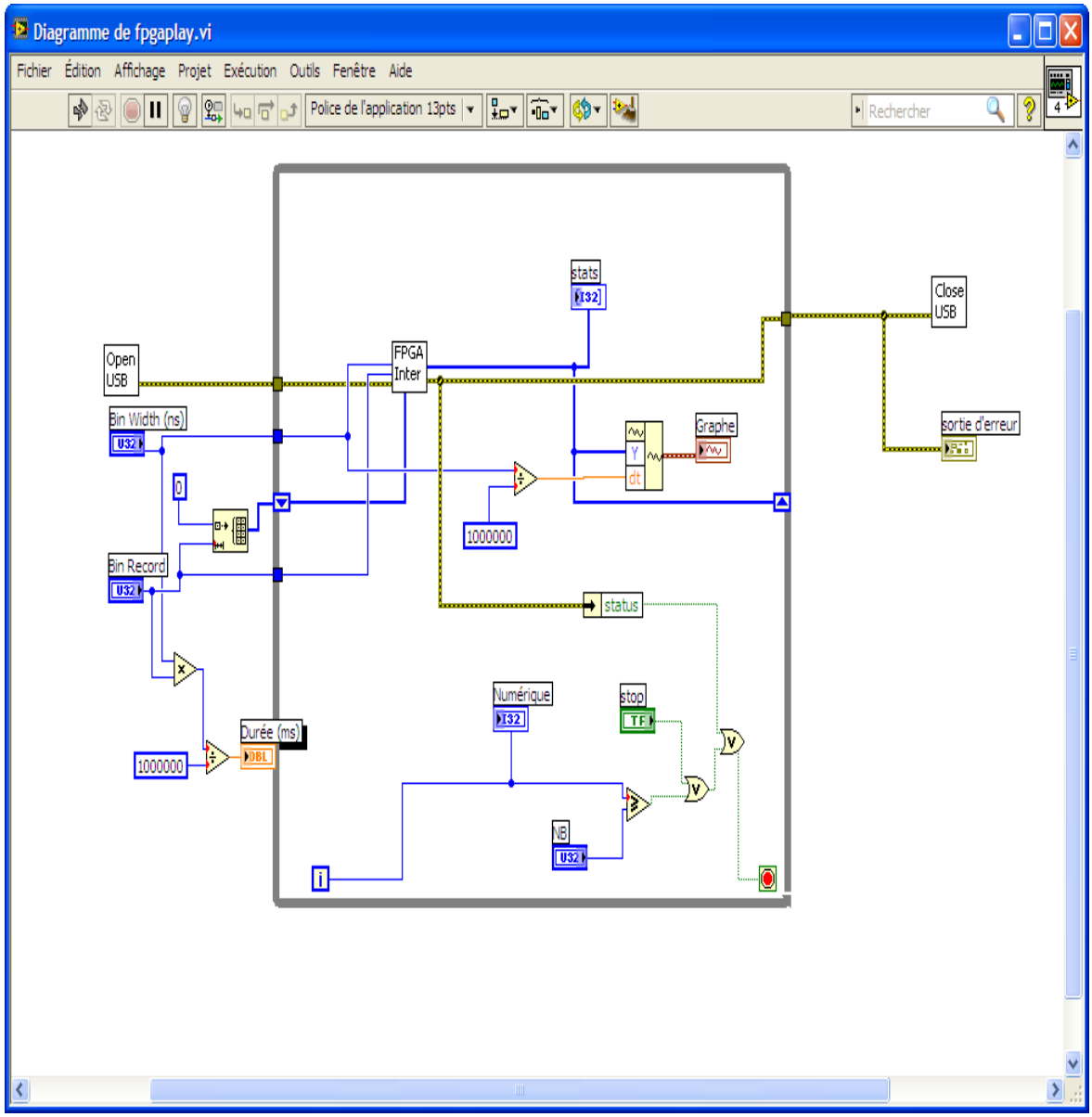


Figure 37 : Diagramme de VI LabVIEW-FPGA

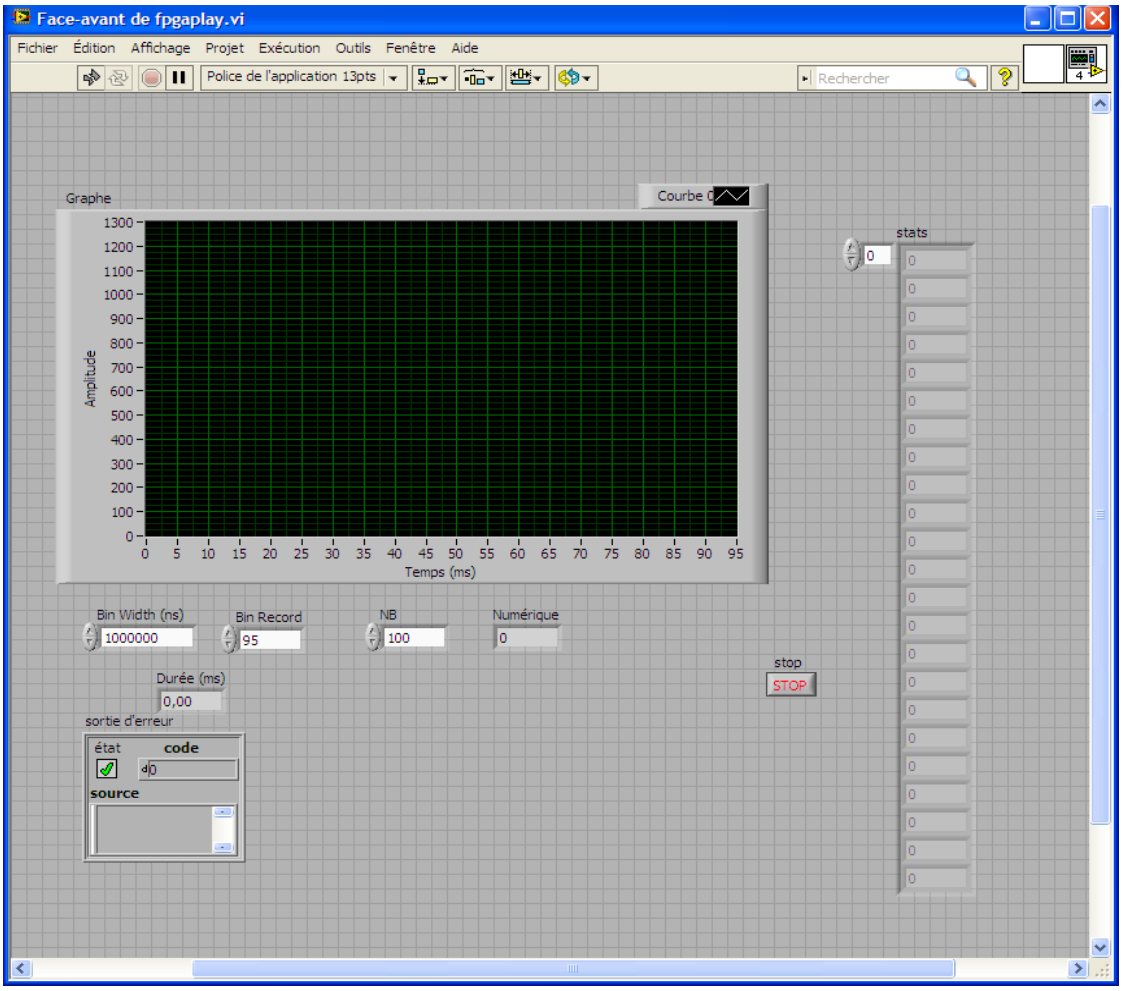


Figure 38 : Face avant de VI LabVIEW-FPGA

Par la suite , on va rajouter d'autres outils pour développer notre VI, comme par exemple les curseurs et aussi un fichier pour enregistrer et sauvegarder nos données.

6. Tests et résultats

Nous avons utilisé l'Arduino, pour générer des impulsions suivant une distribution exponentielle, et qui seront enregistré par l'oscilloscope pour être traité par la suite par notre carte FPGA, qui va faire le comptage afin de validé l'étude théorique, faite au paragraphe 3 sur les statistiques de comptage . Et après, nous avons tester la carte dans la manipulation de spectroscopie de fluorescence.

6.1. Arduino Uno

Avec le langage Arduino, nous avons réalisé un programme qui génère des impulsions suivant une distribution exponentielle. Vous trouverez le code de ce programme sur l'annexe 6.

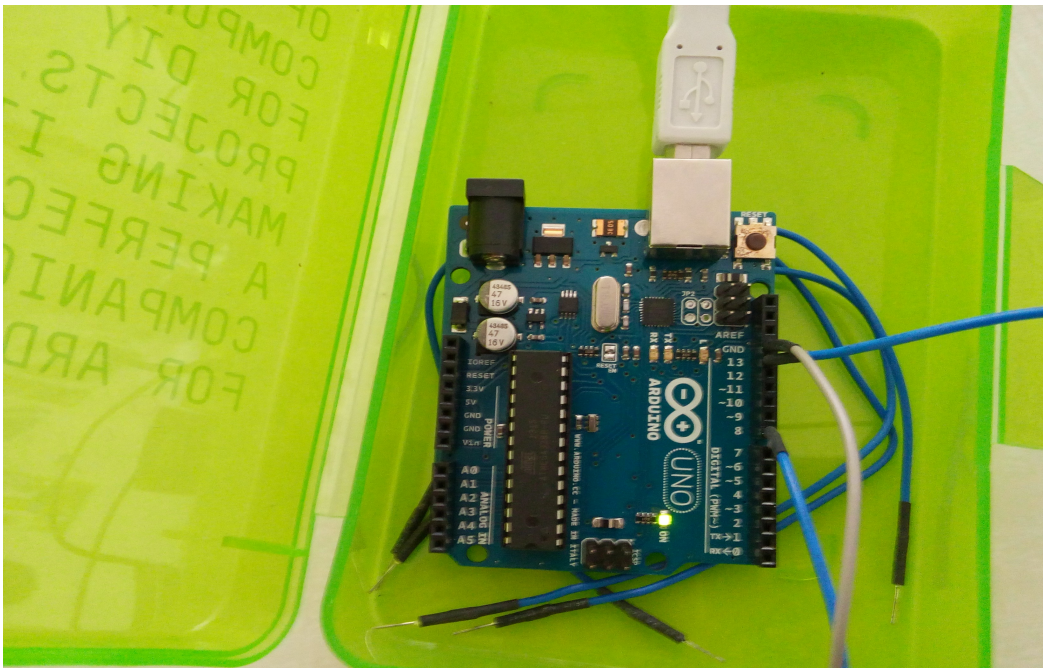


Figure 39 : La carte Arduino Uno , utilisé durant ce travaille

L'Arduino Uno est une carte microcontrôleur basée sur un micro-controleur, l' ATmega328.
Et il dispose de ces éléments suivants :

- 14 broches numériques d'entrée / sortie, dont 6 peuvent être utilisées comme sorties
- 6 entrées analogiques
- Un résonateur de 16 MHz
- Une connexion USB
- Une prise d'alimentation
- Une embase ICSP et d'un bouton de réinitialisation.

Il contient tout le nécessaire pour soutenir le microcontrôleur .
On branche la carte à l'ordinateur avec un câble USB, et on utilise juste les broches 8 et 13 et la masse GND.

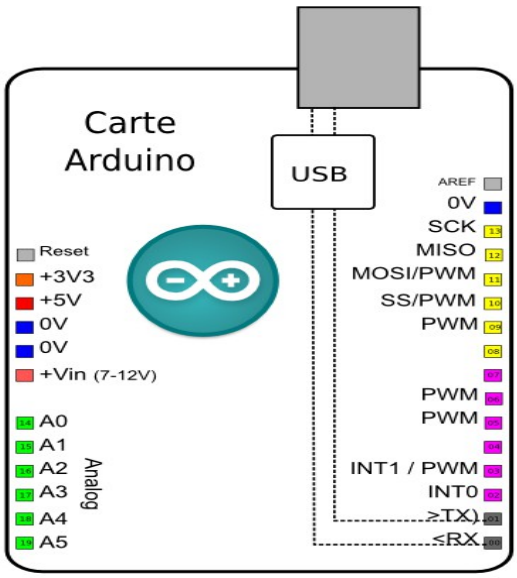


Figure 40 : Brochage de la carte Arduino Uno

Les broches d'alimentation sont les suivantes :

- VIN : La tension d'entrée positive lorsque la carte Arduino est utilisée avec une source de tension externe (à distinguer du 5V de la connexion USB ou autre source 5V régulée).
- 5V : La tension régulée utilisée pour faire fonctionner le microcontrôleur et les autres composants de la carte. Le 5V régulé fourni par cette broche peut provenir de la tension d'alimentation VIN (Tension positive en entrée) via le régulateur de la carte.
- 3V3 : Une alimentation de 3.3V fournie par le circuit intégré FTDI, qui est un circuit qui fait l'adaptation du signal entre le port USB de l'ordinateur et le port série de l'ATmega de la carte. Et l'intensité maximale disponible sur cette broche est de 50mA
- GND : Broche de masse (ou 0V).

Les entrées et sorties numériques :

Chacune des 14 broches numériques de la carte UNO (numérotées des 0 à 13) peut être utilisée soit comme une entrée numérique, soit comme une sortie numérique, en utilisant les instructions pinMode(), digitalWrite() et digitalread() du langage Arduino.

Ces broches fonctionnent en 5V. Chaque broche peut fournir ou recevoir un maximum de 40mA d'intensité .

De plus, certaines broches ont des fonctions spécialisées :

- Communication Série: Broches 0 (RX) et 1 (TX). Utilisées pour recevoir (RX) et transmettre (TX) les données séries de niveau TTL. Ces broches sont connectées aux broches correspondantes du circuit intégré ATmega8U2 programmé en convertisseur USB-vers-série de la carte, composant qui assure l'interface entre les niveaux TTL et le port USB de l'ordinateur.

- Interruptions Externes: Broches 2 et 3. Ces broches peuvent être configurées pour déclencher une interruption sur une valeur basse, sur un front montant ou descendant, ou sur un changement de valeur.
- Impulsion PWM (largeur d'impulsion modulée)**:** Broches 3, 5, 6,8, 9, 10, et 11. Fournissent une impulsion PWM 8-bits à l'aide de l'instruction analogWrite() du langage Arduino.
- SPI (Interface Série Périphérique) : Broches 10 (SS), 11 (MOSI), 12 (MISO), 13 (SCK). Ces broches supportent la communication SPI (Interface Série Périphérique) .
- I2C: Broches 4 (SDA) et 5 (SCL). Supportent les communications de protocole I2C (ou interface TWI (Two Wire Interface - Interface "2 fils").
- LED: Broche 13. Il y a une LED incluse dans la carte connectée à la broche 13. Lorsque la broche est au niveau HAUT, la LED est allumée, lorsque la broche est au niveau BAS, la LED est éteinte.

Nous avions utilisé les broches d'entrée/sorties numériques 8 et 13 ,et avec le langage Arduino nous écrivons un programme qui génère des impulsions suivant une distribution ou courbe exponentielle .Vous trouverez le code de ce programme sur l'annexe 6.

On a dit qu'on allait développer le VI LabVIEW-FPGA, on rajoutant des autres outils comme le montre la figure ci-dessous:

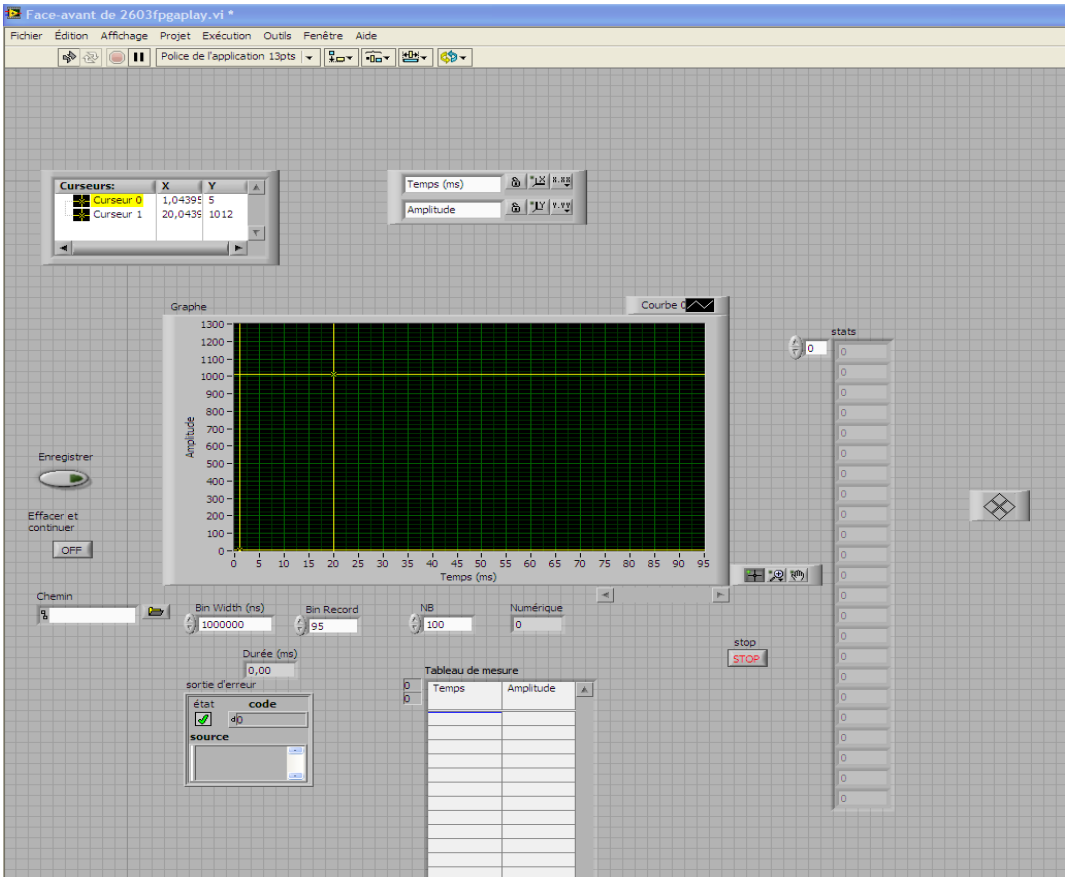


Figure 41 : La forme générale de la face avant de notre VI FPGA

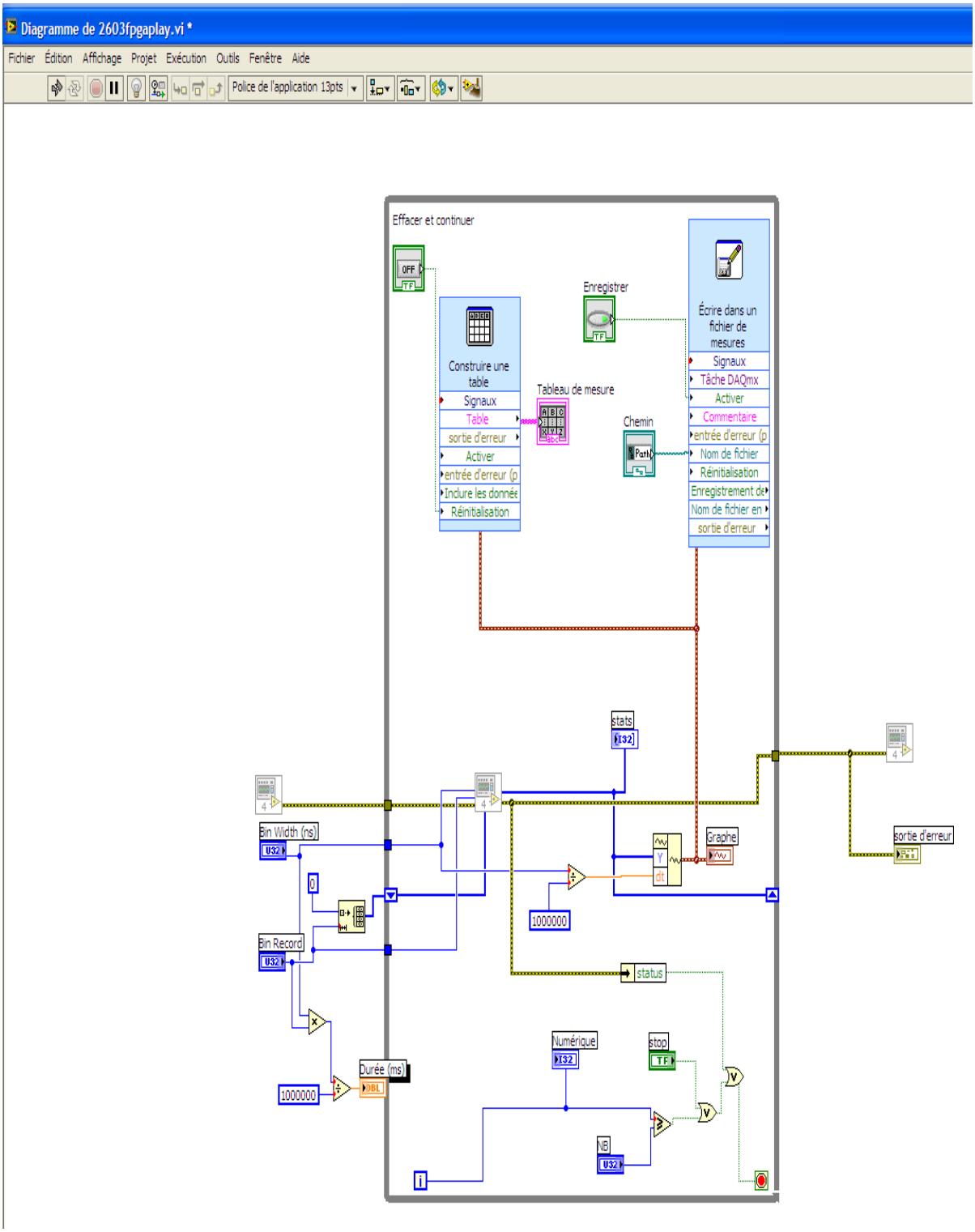


Figure 42 : La forme général de digramme de notre VI FPGA

Après l'installation de tous les logiciels utilisés, on connecte la carte Arduino et la carte FPGA avec l'ordinateur, et on lance le logiciel de configuration de la carte FPGA (qu'on a déjà installé, FPGAconf), et pour bien choisir nos paramètres on utilise BOARD, Tools et Option (figure 43).

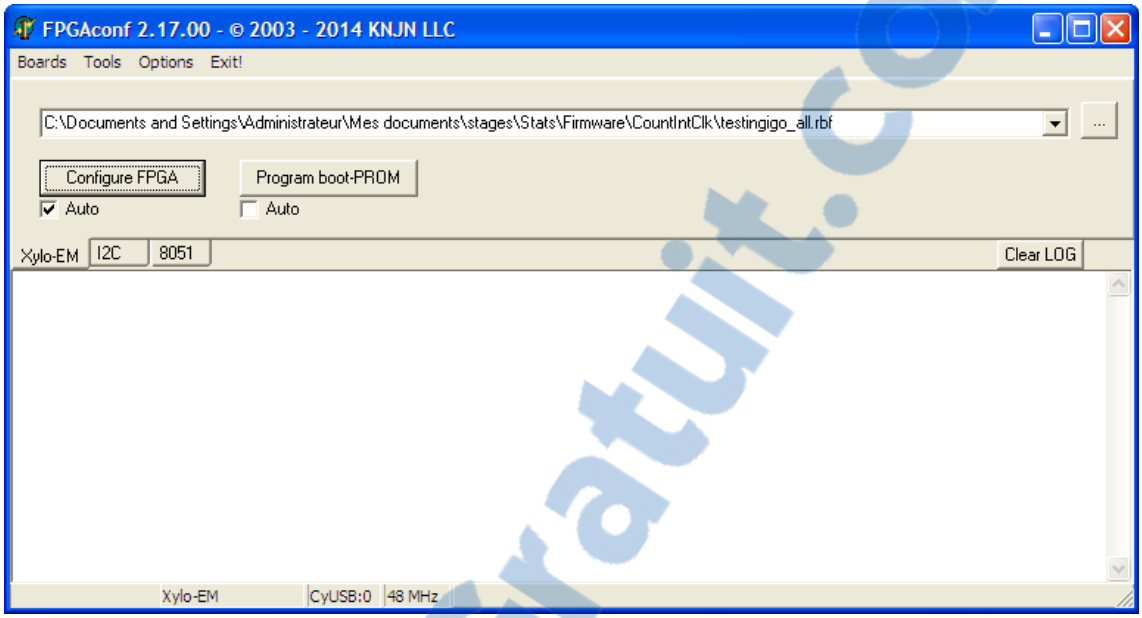


Figure 43 : Logiciel d'installation et configuration de FPGA

On connecte les pin ou broches d'entrées/sorties(8 et 13) de la carte Arduino à l'oscilloscope, on ouvre le langage Arduino, pour compiler et exécuter le code informatique (Annexe 6).
par la suite l'oscilloscope nous génère des impulsions sur des bacs de temps séquentielle ou des petits intervalles de temps $\Delta t = 20ns$ avec T= 93ms.(Figure 44)
Et on connecte la sortie de circuit via la carte FPGA .

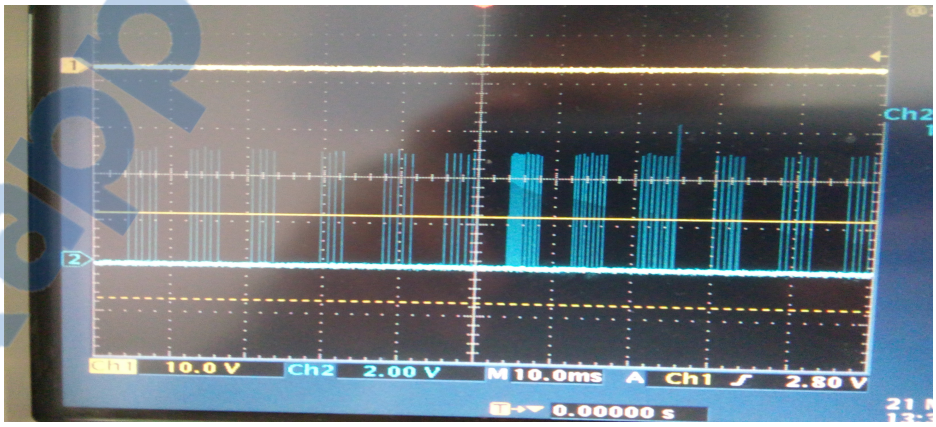


Figure 44 : La forme des impulsions générer

On remarque que la distribution des impulsion a une forme exponentielle, comme on a souhaité le voir, au début du déclenchement(le top départ du comptage) on a beaucoup d'impulsion (on les vois en gras sur oscilloscope) , et après certains temps, le nombre des impulsions diminuent .

Après la configuration de la carte FPGA , on lance le logiciel LabVIEW, et sur la face avant de VI LabVIEW-FPGA, on enregistre les résultats comme le montre la figure 45.

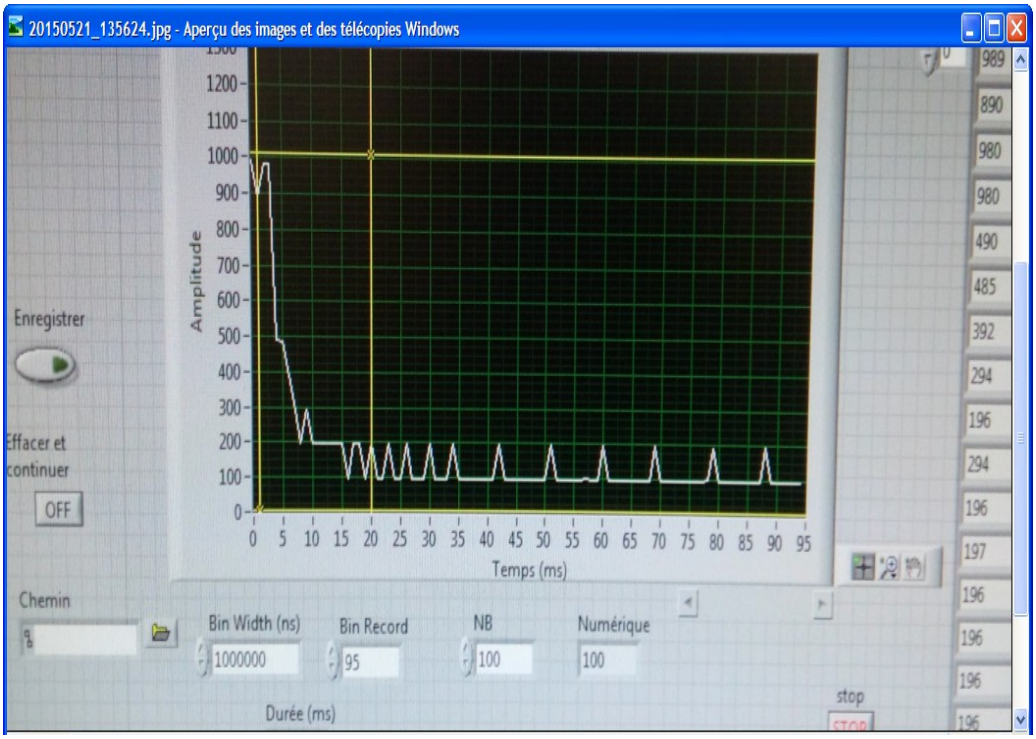


Figure 45 : Les résultats de comptage des impulsuin sur la face avant de VI

On remarque sur la figure 45 que la courbe décroit avec le temps sous forme exponentielle ce qui est compatible avec notre étude théorique faite au première partie sur les statistique de comptage . Et on vois aussi que le nombre des impulsions pour chaque petits intervalles du temps diminuent ce qui est le cas aussi pour les impulsions qu'on vois sur l'oscilloscope .

En conclusion , les résultats de VI LabvIEW-FPGA correspond à les même résultats de l'oscilloscope ,ce qui nous permet par la suite de tester ce projet sur la manipulation de spectroscopie de fluorescence afin de valider notre travaille

6.2. Résultats

Dans la manipulation de spectroscopie de fluorescence , nous avons remplacer le Stanford SR430 par la carte FPGA, comme c'est montrer ci-dessous :

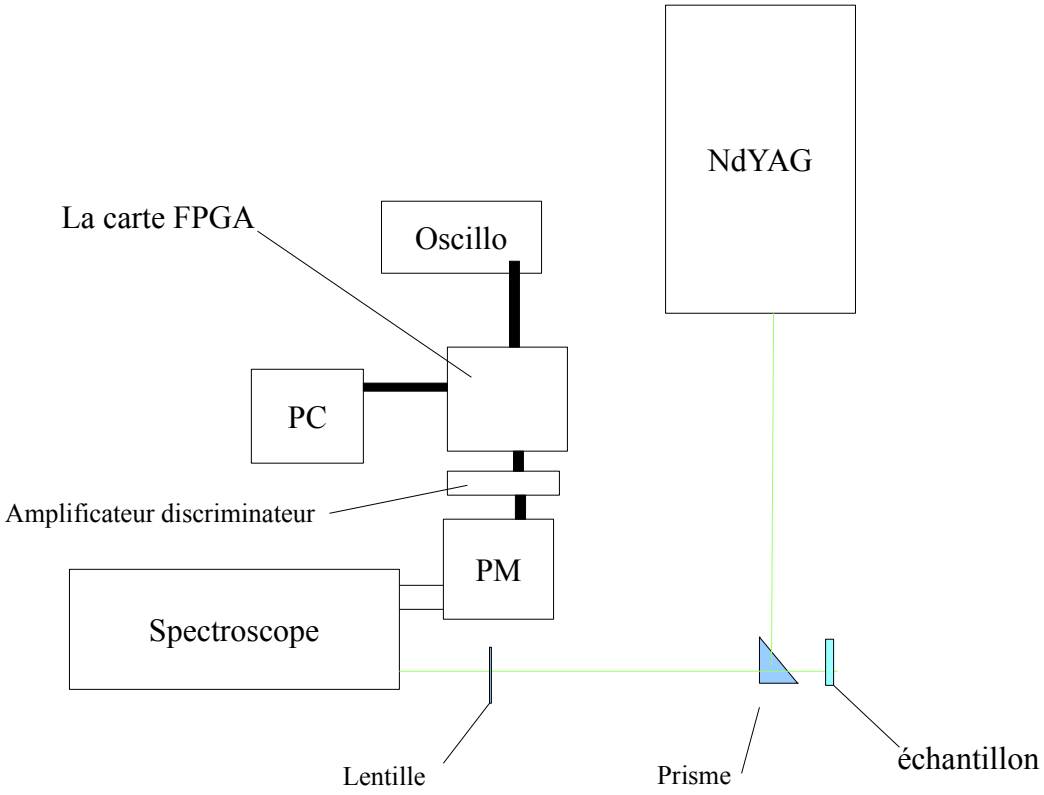


Figure 46 : Schéma du dispositif expérimentale avec la carte FPGA

Après avoir connecté tous les éléments ensembles comme c'est montrée sur la figure 46, on lance notre VI LabVIEW-FPGA pour enregistrer les événements (l'arrivée du photons) sur la face avant.

La carte FPGA fait l'acquisition des informations provenant de l'amplificateur discriminateur, et le compteur qui est à l'intérieur de la carte (sa taille est de 32 bits) s'incrémente chaque 20 ns et à chaque impulsion détectée.

Les résultats sont ensuite enregistrés dans un fichier excel, pour qu'on puisse tracer la courbe de l'amplitude de signal en fonction du temps, et la comparer par la suite avec la courbe de l'ancien appareil de mesure (Le Standford SR430).

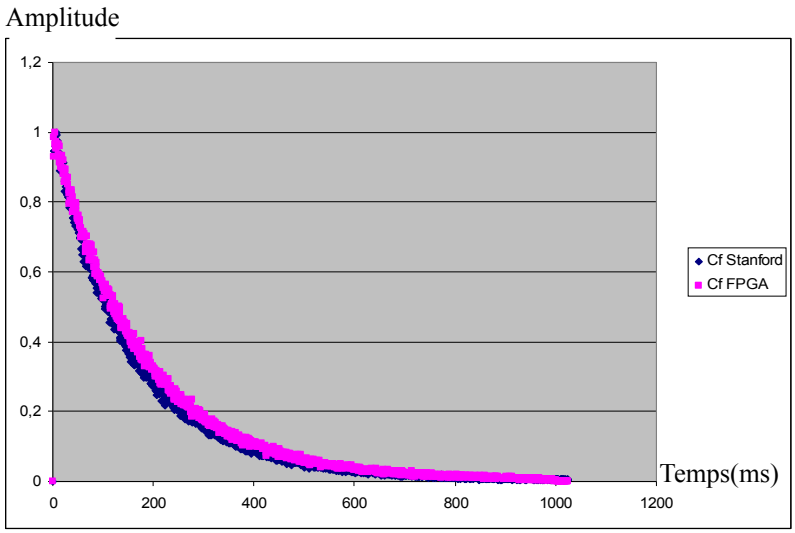


Figure 47 : Les deux courbes donnant les temps de vie

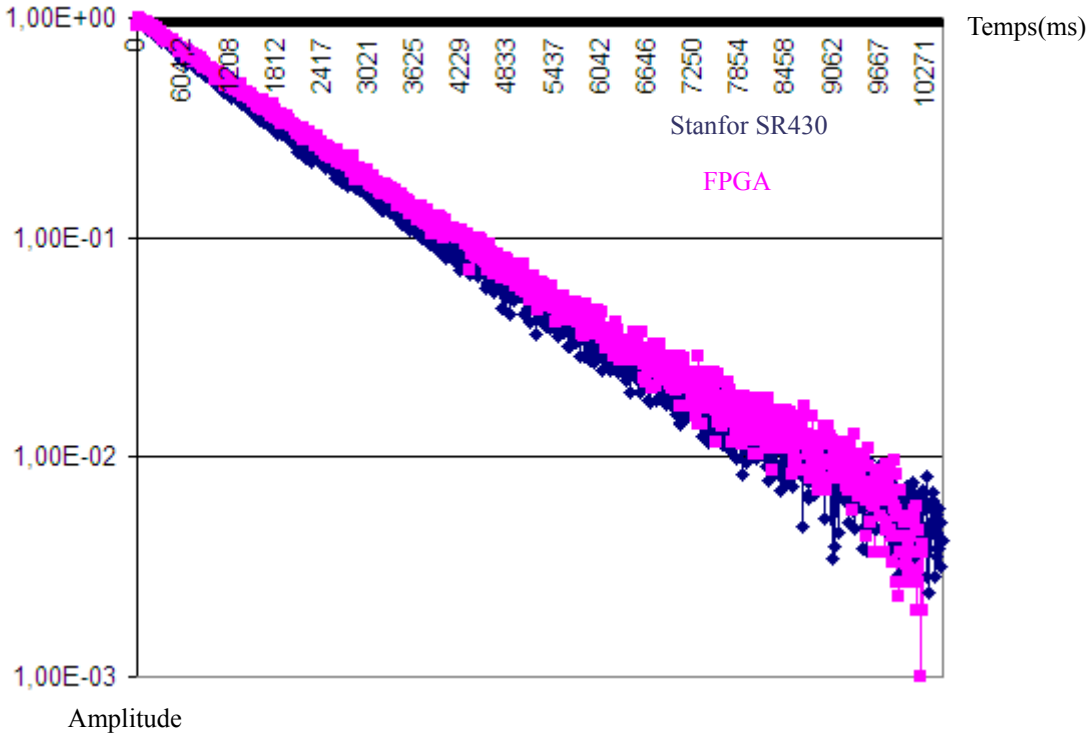


Figure 48 : Les deux courbes donnant les temps de vie de photons en échelle logarithmique

On remarque que les deux courbes sont pareilles, et il y a juste une petite différence. Au début on a beaucoup de photons qui arrivent et la petite différence qui se trouve à la fin des deux courbes est dû au bruit de photon et à l'amplificateur discriminateur externe de la carte FPGA qui était imposé et qui est moins performant par rapport à l'amplificateur utilisé dans la manipulation. Et donc les deux résultats de mesures sont compatibles et ce qui nous permet par la suite de valider notre travaille de recherche .

Les deux systèmes d'acquisition sont pareilles, il y a juste une différence par rapport à l'interface utilisateur. Avec la carte FPGA, l'interface utilisateur avec le LabVIEW est beaucoup plus facile a utilisé par rapport à celui de Stanford SR430. Et aussi, il y a des différences par rapport à la taille, la carte FPGA est petites et ne prend pas beaucoup de place dans la manipulation de spectroscopie de fluorescences .

Notre système d'acquisition avec la carte FPGA, est beaucoup plus performant ,il donne les résultats de mesures rapidement, donc il est beaucoup plus efficaces que le Stanford SR430, qui est devenu ancien et très lent à l'utilisation.

Conclusion

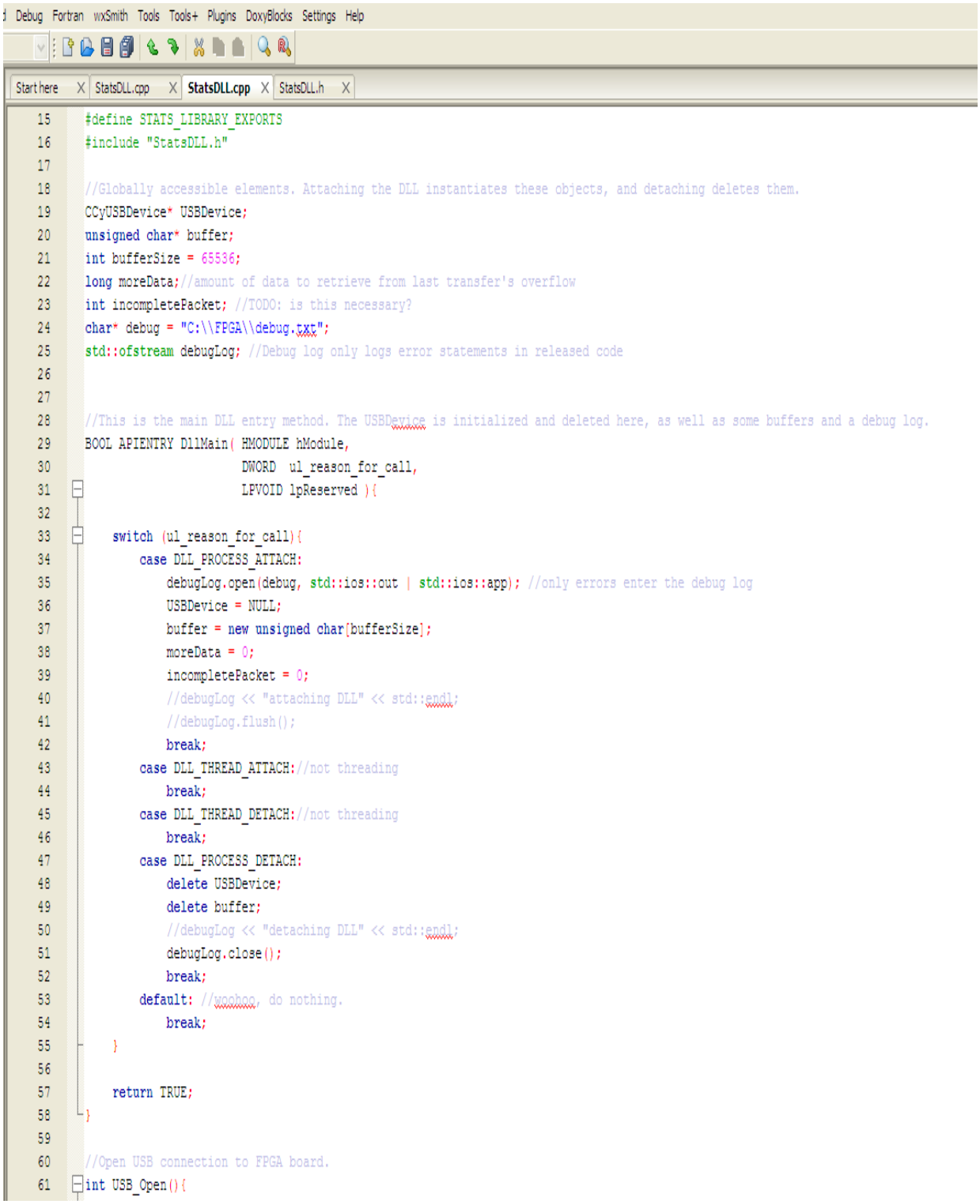
Ce travaille a permis de réaliser un compteur de photons avec la carte FPGA, et qui prendra par la suite la place de Sranford SR430 dans la manipulation de spectroscopie de fluorescences.

On peut utiliser ce travaille dans d'autres manipulations, car les compteurs de photons sont utilisés en biophotonique, en microscopie confocal, en microscopie de fluorescence, et en astronomie.

Et on peut dire que l'utilisation des circuits électriques intégrés, c'est un bon moyen pour développer nos systèmes de mesures ou des manipulations .

Annexe 4

Le code informatique de la DLL utilisé



Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help

Start here | StatsDLL.cpp | StatsDLL.cpp | StatsDLL.h

```
#define STATS_LIBRARY_EXPORTS
#include "StatsDLL.h"

//Globally accessible elements. Attaching the DLL instantiates these objects, and detaching deletes them.
CCyUSBDevice* USBDevice;
unsigned char* buffer;
int bufferSize = 65536;
long moreData;//amount of data to retrieve from last transfer's overflow
int incompletePacket; //TODO: is this necessary?
char* debug = "C:\\FPGA\\debug.txt";
std::ofstream debugLog; //Debug log only logs error statements in released code


//This is the main DLL entry method. The USBDevice is initialized and deleted here, as well as some buffers and a debug log.
BOOL APIENTRY DllMain( HMODULE hModule,
                       DWORD  ul_reason_for_call,
                       LPVOID lpReserved ){

    switch (ul_reason_for_call){
        case DLL_PROCESS_ATTACH:
            debugLog.open(debug, std::ios::out | std::ios::app); //only errors enter the debug log
            USBDevice = NULL;
            buffer = new unsigned char[bufferSize];
            moreData = 0;
            incompletePacket = 0;
            //debugLog << "attaching DLL" << std::endl;
            //debugLog.flush();
            break;
        case DLL_THREAD_ATTACH://not threading
            break;
        case DLL_THREAD_DETACH://not threading
            break;
        case DLL_PROCESS_DETACH:
            delete USBDevice;
            delete buffer;
            //debugLog << "detaching DLL" << std::endl;
            debugLog.close();
            break;
        default: //woohoo, do nothing.
            break;
    }

    return TRUE;
}

//Open USB connection to FPGA board.
int USB_Open(){
```

```
62
63      if(USBDevice == NULL){
64          //std::cout << "Creating USBDevice." << std::endl;
65          //debugLog << "Creating USBDevice." << std::endl;
66          USBDevice = new CCyUSBDevice(NULL, GUID_CYUSB_FX2,true);
67          return 0;
68      }
69      else{
70          //std::cout << "Error: USB communication already initiated." << std::endl;
71          debugLog << "Error: USB communication already initiated." << std::endl;
72          return ERROR_USB_INITIATED;
73      }
74  }
75
76  //Close USB connection to FPGA board.
77  int USB_Close() {
78      if(USBDevice == NULL){
79          //std::cout << "Error: Trying to close NULL USB connection." << std::endl;
80          debugLog << "Error: Trying to close NULL USB connection." << std::endl;
81          return ERROR_USB_UNINITIATED;
82      }
83      else{
84          debugLog << "Closing and deleting USB Device." << std::endl;
85          delete USBDevice;
86          USBDevice = NULL;
87          return 0;
88      }
89  }
90
91  // Pulls timestamped clicks from FPGA, and calculates user-defined statistics data (stats)
92  int FPGA_com(long &size, unsigned char &fpgaCommand){
93      //debugLog << "start" << std::endl;
94      //debugLog.flush(;)
95      //debugLog << "Max packet size: " << USBDevice->MaxPacketSize << std::endl;
96
97      //clicklog
98      //std::string line;
99      //static char delimiter = ' ';
100
101     std::ofstream clickLog;
102
103     //command to send to FPGA or data received from FPGA
104     unsigned char fpgaCommunicator[2];
105     long fpgaCommunicatorBytes = 1;
106     fpgaCommunicator[0] = fpgaCommand;
107
108     //variables
109     int status = 0;
110
```

```
110
111  quest number of new data points
112 status){
113  fpgaCommunicator[0] = fpgaCommand;
114  fpgaCommunicatorBytes = 1;
115  ///debugLog << "requesting size" << std::endl;
116  status = !BulkOutPipe2->XferData(fpgaCommunicator, fpgaCommunicatorBytes);
117
118 if(status){
119      debugLog << "error requesting data size" << std::endl;
120      debugLog.flush();
121      status = ERROR_FPGA_REQUEST_DATA_SIZE;
122  }
123
124 status){
125  //get data size
126  fpgaCommunicator[0] = 0;
127  fpgaCommunicator[1] = 1;
128  fpgaCommunicatorBytes = 2;
129  status = !BulkInPipe5->XferData(fpgaCommunicator, fpgaCommunicatorBytes);
130
131 if(status){//error
132      debugLog << "error getting size of data" << std::endl;
133      debugLog.flush();
134      status = ERROR_FPGA_GET_DATA_SIZE;
135  }
136
137
138 status){
139  //calculate size
140  size = (long) fpgaCommunicator[1]+ ((long) fpgaCommunicator[0])*256; //number of bytes stored in FPGA since last query
141  //debugLog << "data size: " << size << " + " << moreData << " from last round = " << size + moreData << std::endl;
142  //debugLog.flush();
143  size += moreData;
144
145 if(size < USBFrame){
146      size = USBFrame;
147  }
148  moreData = size%USBFrame;
149  size -=  moreData;
150
151  status = !BulkInPipe4->XferData((buffer + incompletePacket), size); //starting off at pointer + incompletePacket, to ensure that
152  //data point from last query is completed.
153
154 if(status){//error
155      debugLog << "error sending data transfer request for size: " << size << std::endl;
156      debugLog.flush();
```

```
157                 status = ERROR_FPGA_GET_BULK_DATA;;
158             }
159             else{
160                 size += incompletePacket; //the total data size in buffer is the size of the data retrieved + the
161                 //incomplete packet from the previous query
162                 //debugLog << "got data of size: " << size << std::endl;
163                 //debugLog.flush();
164             }
165         }
166 
167         if (!status){
168             incompletePacket = size%FPGAdataPointSize;  //incomplete packets from this query
169             if(incompletePacket){
170                 debugLog << "Incomplete packet: " << incompletePacket << std::endl;
171                 debugLog.flush();
172             }
173 
174 
175             if(incompletePacket){
176                 for(int k = 0; k < incompletePacket; k++){
177                     buffer[k] = buffer[size + k - incompletePacket]; //transfer the bits from the incomplete data packet to
178 //the beginning of the buffer
179                 }
180             }
181         }
182 
183     //debugLog << "stop" << std::endl;
184     //debugLog.close();
185 
186     return status;
187 }
188 
189 
190 _declspec(dllexport) int  FPGA_Interface (unsigned long Bin_width, unsigned long Bin_record,
191                                 HANDLE XyloDeviceHandle, int * stats)
192 {
193     long nb_bytes_received = 0;
194     unsigned char fpga_command = 4;
195 
196     //Acquisitions variables
197     unsigned char acq_run = 0;
198     long Timestart = 0;
199     unsigned long Bin = 0;
200 
201     //Timeout 100ms default
202     unsigned int runs = 100;
203 
```

```
    int error_fpga;


    //timeout check: if data acquisition is too slow this finishes the routine without running the protocol "runs" times
    clock_t Time_out = CLOCKS_PER_SEC*runs/1000 + clock();

    while (Bin<Bin_record)
    {
        if (Time_out<=clock()) return 1;

        if ((error_fpga = FPGA_com(nb_bytes_received, fpga_command)) < 1) return error_fpga;

        fpga_command = 0;

        //step 7: compute the simple statistics
        REALTIME_FUNCTION (buffer,nb_bytes_received,stats,chanel1,chanelclear,acq_run,
                           Timestart,Time_out,Bin,Bin_width,Bin_record);
    }

    fpga_command = 2;
    if ((error_fpga = FPGA_com(nb_bytes_received, fpga_command)) < 1) return error_fpga;


    return Bin; //no error
}



////////////////////////////////////////////////////
// Default real-time
// Simple correlation analysis. Computes statistics.
////////////////////////////////////////////////////
/*
stats structure: number of occurences of these events

*/
void declin ( unsigned char * data, int length, int *stats, unsigned char acq_chanel, unsigned char start_chanel,
              unsigned char &acq_run, long &Timestart, clock_t &Time_out,
              unsigned long &Bin, unsigned long Bin_width, unsigned long Bin_record)
{
    int i;
    unsigned char acquisition;
    unsigned char start_clear;
    double Timestamp;


    for (i=0; i<length; i+=4)
```

```
250     for (i=0; i<length; i+=4)
251     {
252         acquisition = (data[i+3]&acq_chanel)?true:false;         //Is photon detected
253         start_clear = (data[i+3]&start_chanel)?true:false;       //Is Laser Synchronisation detected
254
255         //Time stamp event
256         Timestamp = (data[i] + (data[i+1]<<8) + (data[i+2]<<16) + ((data[i+3]&0x07)<<24))*time_step;
257
258
259         if (acq_run==0)
260         {
261           //First step wait the Laser synchronisation to start acquisition
262           if (start_clear)
263           {
264                     acq_run++;                //set acquisition status to run
265                     Timestart = 0;//Timestamp;     //Time origin
266                     Time_out = CLOCKS_PER_SEC/*(Bin_record*Bin_width/1e9) */+ clock();    //Adjuste time out to
267                     //record time float(Bin_record*Bin_width/1e9)
268           }
269         }
270         //second step count photon
271         else
272         {
273             if (acquisition)
274            {
275                Bin = (unsigned long)(Timestamp/(Bin_width*1e-9));          //Bin calculation
276                 if (Bin<Bin_record)                                        //if current Bin low than Bin record
277                         stats[Bin] += 1;                                   //count photon
278           }
279         }
280     }
281     return;
282 }
283
```

Annexe 6

Le code informatique utilisé dans la langage de l'Arduino

sim_declin2 | Arduino 1.6.1

Fichier Édition Croquis Outils Aide

sim_declin2

```
/*

 */
#include <TimerOne.h>
//#include <TimerThree.h>

#define nbpoint 100
#define laserperiod 100000
#define dt  10


unsigned int freq[nbpoint];
unsigned int i;
unsigned int j;
unsigned int nt;

// the setup routine runs once when you press reset:
void setup() {
  // initialize serial communication at 9600 bits per second:
  //Serial.begin(115200);
  pinMode(13, OUTPUT);
  pinMode(8, OUTPUT);
  float g = dt/exp(1);
  for(i=0;i<nbpoint;i++)
  {
    freq[i] = dt-g*exp(1-(float)i/15);
  }
  i = 0;
  nt=0;
  j=0;
  Timer1.initialize(laserperiod/(nbpoint*dt));
  Timer1.attachInterrupt( timerIsr );
}

// the loop routine runs over and over again forever:
void loop() {

}

// --------------------------
/// Custom ISR Timer Routine
/// --------------------------
void timerIsr()
```

```
{
   j++;
   if (j>=freq[i]) {
     j=0;
      digitalWrite(8,HIGH);
      digitalWrite(8,LOW);
   }
   nt++;
   if (nt>=dt) {
     nt=0;
     i++;
     if (i>= nbpoint) {
       i=0;
       digitalWrite(13, HIGH);
       digitalWrite(13,LOW);
     }
   }
}
```

Annexe 5

Signal name	Width	Direction	Purpose
FIFO_CLK	1 bit	FX2 → FPGA	Clock (all the other signals are synchronous to this clock)
FIFO_RD	1 bit	FPGA → FX2	FPGA reads from FX2
FIFO_WR	1 bit	FPGA → FX2	FPGA writes to FX2
FIFO_PKTEND	1 bit	FPGA → FX2	FPGA indicates "End of packet" (i.e. FPGA is done writing)
FIFO_DATAIN_OE	1 bit	FPGA → FX2	FPGA wants the FX2 to drive the DATA bus
FIFO_DATAOUT_OE	1 bit	internal	FPGA drives the DATA bus
FIFO_DATAIN	8 bits	FX2 → FPGA	Data from FX2
FIFO_DATAOUT	8 bits	FPGA → FX2	Data to FX2
FIFO_FIFOADR	2 bits	FPGA → FX2	FPGA selects one FX2 FIFO (out of the four available)
FIFO2_empty / FIFO2_data_available	1 bit	FX2 → FPGA	FIFO2 is empty, or not
FIFO3_empty / FIFO3_data_available	1 bit	FX2 → FPGA	FIFO3 is empty, or not
FIFO4_full / FIFO4_ready_to_accept_data	1 bit	FX2 → FPGA	FIFO4 is full, or not
FIFO5_full / FIFO5_ready_to_accept_data	1 bit	FX2 → FPGA	FIFO5 is full, or not

La liste complète des signaux utilisés en interne par le FPGA pour accéder aux FIFO

Annexe 1

le photomultiplicateur PM

HAMAMATSU

PHOTON IS OUR BUSINESS

PHOTOMULTIPLIER TUBE R943-02

GaAs (Cs) Photocathode, Wide Spectral Response, 51 mm (2") Diameter Head-on Type for Photon Counting, Low Dark Counts, Excellent P.H.D.

FEATURES

- **Wide Spectral Response ..... 160 nm to 930 nm**
- **High Quantum Efficiency in Near IR ... 14 % at 632.8 nm**
- **Fast Rise Time ..... 3.0 ns at 1500 V**
- **Excellent Single Photoelectron Pulse Height Distribution ..... Peak to Valley Ratio 2.3 (at -20 °C)**
- **Low Dark Counts ..... 20 s^{-1} Typ. (at -20 °C)**

APPLICATIONS

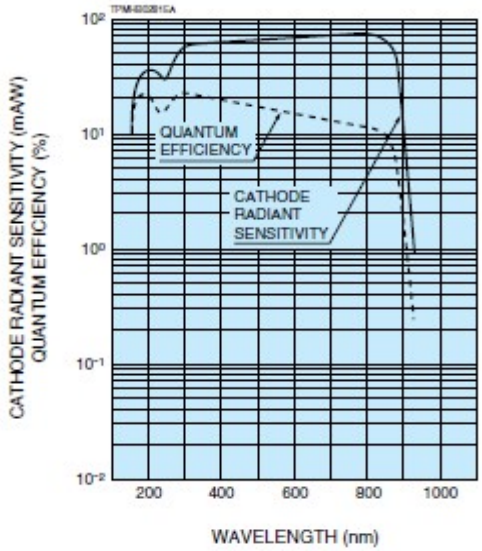
- **Raman Spectroscopy**
- **Fluorescent Spectroscopy**
- **Astrophysical Measurement**
- **Laser Detection**

Hamamatsu R943-02 is a 51 mm (2") diameter head-on type photomultiplier tube having GaAs (Cs) photocathode and synthetic silica window. The combination of the GaAs photocathode and the synthetic silica window allows high sensitivity over a wide spectral range from UV to IR (160 nm to 930 nm).
The R943-02 is selected for photon counting and features low dark counts and excellent pulse height distribution (PHD) of single photoelectrons.

GENERAL

Parameter	Description / Value	Unit
Spectral Response	160 to 930	nm
Wavelength of Maximum Response	300 to 800	nm
Photocathode		
Material	GaAs(Cs)	—
Minimum Effective Area	10 × 10	mm
Mode	Opaque	—
Window Material	Synthetic silica glass	—
Dynode		
Secondary Emitting Surface	Cu-BeO	—
Structure	Linear focused	—
Number of Stages	10	—
Direct Interelectrode Capacitances		
Anode to Last Dynode	Approx. 2.7	pF
Anode to All Other Electrodes	Approx. 5.0	pF
Base	21-pin glass base	—
Suitable Socket	E678-21C (supplied)	—
Weight	93	g
Operating Ambient Temperature	-30 to +50	°C
Storage Temperature	-80 to +50	°C

Figure 1: Typical Spectral Response



MAXIMUM RATINGS (Absolute Maximum Values)

Parameter	Value	Unit
Supply Voltage		
Between Anode and Cathode	2200	V
Between Anode and Last Dynode	250	V
Average Anode Current Ⓐ	1	μA
Average Pulse Count Rate Ⓑ	6×10^6	s^{-1}
Average Cathode Current Ⓒ	1	nA

CHARACTERISTICS (at 25 °C)

Parameter	Min.	Typ.	Max.	Unit
Cathode Sensitivity Ⓓ				
Quantum Efficiency				
at 253.7 nm (Hg-Line)	—	15	—	%
at 632.8 nm (He-Ne Laser)	—	14	—	%
Luminous Ⓔ	300	600	—	μA/lm
Radiant at 253.7 nm (Hg-Line)	—	30	—	mA/W
at 632.8 nm (He-Ne Laser)	—	70	—	mA/W
at 700 nm	—	71	—	mA/W
at 852.1 nm (Cs-Line)	—	65	—	mA/W
Red/White Ratio Ⓕ	—	0.58	—	
Anode Sensitivity Ⓖ				
Luminous Ⓔ	150	300	—	A/lm
Radiant at 253.7 nm (Hg-Line)	—	1.5×10^4	—	A/W
at 632.8 nm (He-Ne Laser)	—	3.5×10^4	—	A/W
at 700 nm	—	3.6×10^4	—	A/W
at 852.1 nm (Cs-Line)	—	3.3×10^4	—	A/W
Gain Ⓖ	—	5×10^5	—	—
Equivalent Anode Dark Current Ⓗ	—	1	10	nA
Anode Dark Counts Ⓙ	—	20	50	s^{-1}
Single Photoelectron PHD (Peak to Valley Ratio)	—	2.3	—	—
Time Response Ⓚ				
Anode Pulse Rise Time Ⓚ	—	3.0	—	ns
Electron Transit Time Ⓛ	—	23	—	ns

NOTES

Ⓐ Averaged over any interval of 30 seconds maximum.

Ⓑ Measured at single photoelectron level. The discriminator level is set at valley point.

Ⓒ In practical operation, the cathode current should be lower than 0.1 nA to prevent shortening the life of the photocathode.

Ⓓ Supply voltage is 150 volts between the cathode and all other electrodes.

Ⓔ The light source is a tungsten filament lamp operated at a distribution temperature of 2856 K.

Ⓕ The quotient of the cathode sensitivity measured with the light source is the same as Note Ⓓ passing through a red filter (Toshiba R-68) divided by the cathode luminous sensitivity without the red filter.

Ⓖ Measured with supply voltage and voltage distribution ratio in Table 1.

Ⓗ Measured with supply voltage to provide the anode luminous sensitivity of 200 (A/lm) and the voltage distribution ratio in Table 1 after 30 minute storage in the darkness.

Ⓙ Measured with supply voltage that gives 2×10^6 gain and with the voltage distribution ratio shown in Table 1 after one hour storage in the cooler set at -20 °C.
The discriminator is set at 1/3 of a single photoelectron level.

Ⓚ The rise time is the time it takes the output pulse to rise from 10 % to 90 % of the peak amplitude when the entire photocathode is illuminated by a delta function light pulse.

Ⓛ The electron transit time is the interval between the arrival of a delta function light pulse at the entrance window of the tube and the time when the output pulse reaches the peak amplitude. In measurement the entire photocathode is illuminated.

Table 1: Voltage Distribution Ratio

Electrode	K	Dy1	Dy2	Dy3	Dy4	Dy5	Dy6	Dy7	Dy8	Dy9	Dy10	P
Distribution Ratio	3	1.5	1	1	1	1	1	1	1	1	1	

Supply Voltage : 1500 V, K : Cathode, Dy : Dynode, P : Anode

Replacement Information

The R943-02 is similar to the Burle C31034 series photomultiplier tube. The base and voltage divider are different.

Warning–Personal Safety Hazards
Electrical Shock — Operating voltages applied to this device present a shock hazard.

Figure 2: Typical Single Photoelectron Pulse Height Distribution

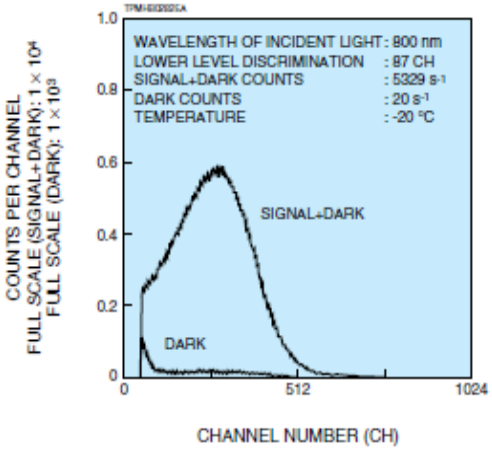


Figure 3: Typical Gain

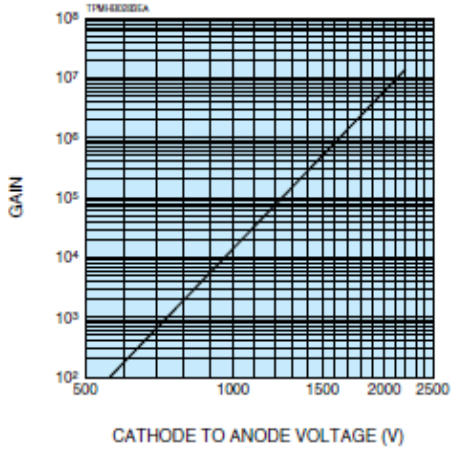


Figure 4: Typical Time Response

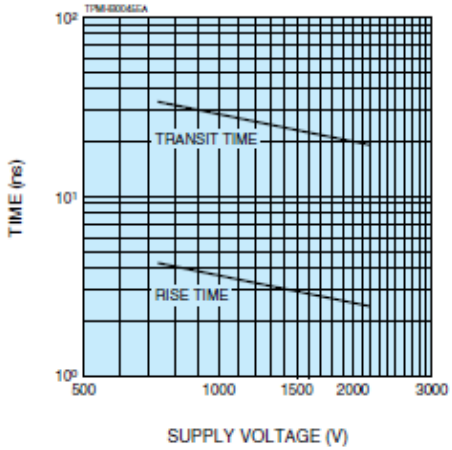


Figure 5: Typical Temperature Coefficient of Quantum Efficiency

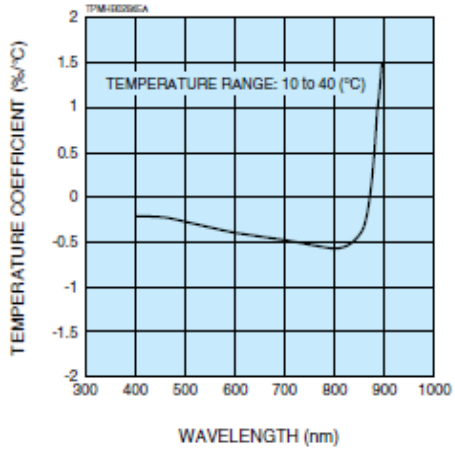
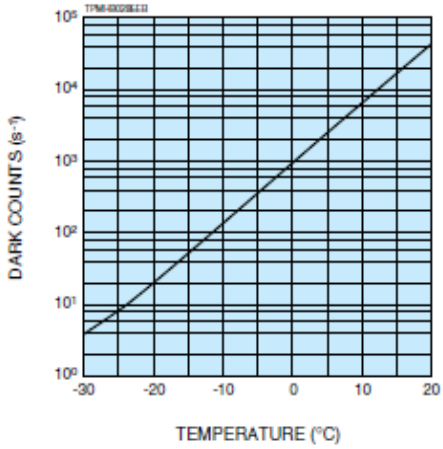


Figure 6: Typical Dark Counts vs. Temperature

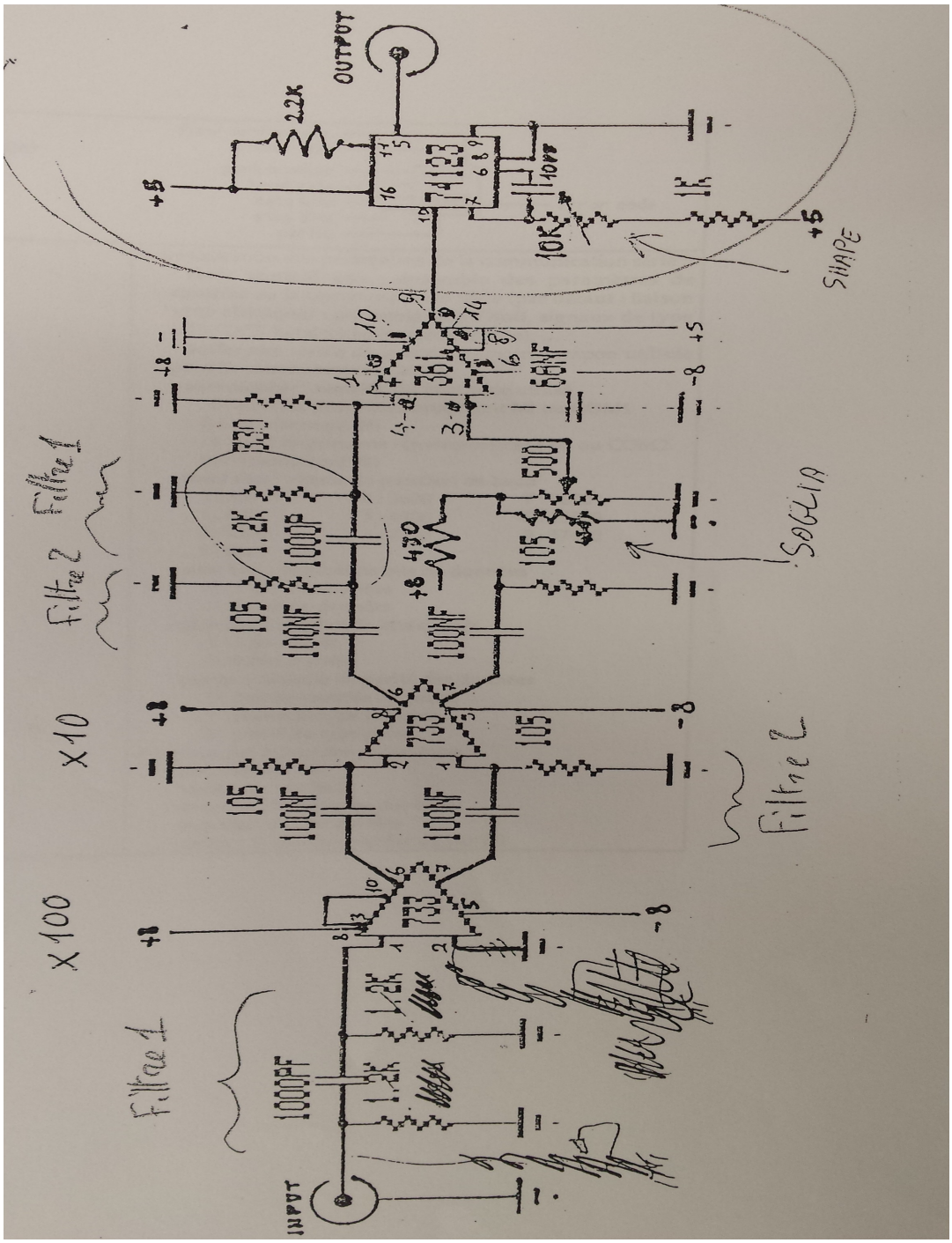


COOLING

As Figure 6 shows, the dark counts of the R943-02 decreases by cooling the tube. Therefore, when performing photon counting, it is recommended that the tube be cooled down to about -20 °C. The cooler C10372 which features temperature control from -30 °C to 0 °C is available from HAMAMATSU.

Annexe 2

l'amplificateur discriminateur



Annexe 3
les caractéristique de Stanford SR430

Entrée de signal

Bande passante	DC à 250 MHz
Impédance d'entrée	50 Ω
Gamme linéaire	± 300 mV (à l'entrée)
Protection d'entrée	± 5 VDC, 50 V pour 1 us
récupération de surcharge	5 ns pour <10 ms surcharge de durée

Discriminateur

gamme de Discriminator	-300 MV à +300 mV
Résolution	0,2 mV
Pente	Positive ou négative
Précision	2 mV + 1%
Min. amplitude d'impulsion	10 mV
Pulse résolution de paire	10 ns (typ.)
vue DISC sortie	Niveau en 50 Ω NIM. (Il existe une insertion retard de 20 ns à partir de l'entrée de signal à la sortie du discriminateur.)

Entrée Trigger

Impédance	10 kΩ
Seuil	-2.000 V à 2,000 V par pas de 1 mV
Pente	Montant ou descendant
Protection	15 VDC, 100 V pour 1 us

Temps interne Bins

Largeur Bin	5 ns, 40 ns, 80 ns ns, 160, 320, 640 ns ns, 1,28, 2,56 ps ps, ... 10,486 ms. (10 ns et 20 ns ne sont pas disponibles)
Précision	1 ns + 20 ppm de la largeur du bac
Jitter (RMS)	100 ps + 10 ppm de retard à partir de SYNC / sortie BUSY (bacs sont synchrones avec SYNC / sortie BUSY)
Indétermination	2,5 ns à l'égard de déclencher entrée
délai d'insertion	45 ns de déclenchement à la première poubelle. Bord de SYNC Rising / sortie BUSY se produit au début de la première bin. Signal impulsions arrivant 25 ns après le déclenchement sera comptée dans le premier bac.

Extérieurement Clocked Temps Bins

Entrée EXT BIN CLK	Front montant déclenche prochain bac de temps
Fréquence maximale	4 MHz (250 ns largeur bin minimum)
Délai minimum haute	100 ns
Délai minimum bas	100 ns
délai d'insertion	Front montant de SYNC / sortie BUSY se produit au premier front montant de EXT BIN CLK après déclenchement. Le début du premier bin se produit en même temps.

Compteurs / Accumulation

Bacs par enregistrement	1k à 16k incrément de 1 Ko (1024 à 32 704, y compris décalage de déclenchement)
Max. taux de comptage	100 MHz
Max. compter	32,767 par bac par déclencheur
Records / accumulation	1 à 64k (ou free run)
Max. accumulation	32,767 par bac en mode Ajouter, ± 16 383 par bac mode à bascule ou externe
Ajouter / soustraction	Les dossiers peuvent être ajoutés ou basculées (ajouter / soustraire sur les déclencheurs en alternance). Entrée de soustraction externe peut également contrôler la bascule.

Fréquence de déclenchement

Le temps de déclenchement minimum	Tp = (nombre de bacs × largeur de bin) + (# de bacs × 250 ns) + 150 ms SYNC / sortie BUSY est élevé pour Tp après chaque déclenchement. Lorsque SYNC / BUSY rendements bas, l'enregistrement suivant peut être déclenchée. Déclencheurs reçu pendant SYNC / BUSY est élevée sont ignorés.

Sorties

DISC	Niveau en 50 Ω NIM. Faible dès que l'entrée de signal dépasse le niveau discriminante avec la pente correcte.
SYNC / BUSY	Niveau TTL. Front montant est synchrone avec le premier bac de temps de chaque enregistrement. Reste élevé jusqu'à réarmée pour le prochain déclencheur.
BIN CLK SORTIE	Niveau en 50 Ω NIM. Chaque transition est une frontière bin. Actif uniquement alors qu'un enregistrement est en cours d'acquisition. Timing inclinaison par rapport au disque en est <2 ns.
BASCULE	Niveau TTL. Indique si l'enregistrement suivant sera ajouté ou soustrait de l'accumulation. (Mode bascule interne)
ESSAI	50 MHz sortie NIM dans 50 Ω
AUX1, AUX2	(General Purpose sorties analogiques)
Pleine échelle	± 10 V
Résolution	5 mV
Le courant de sortie	10 mA
Impédance de sortie	<1 Ω
Précision	0,1% + 10 mV

Entrées

SIGNAL	Entrée analogique 50 Ω
TRIGGER	10 entrées kQ
BIN CLK ENTRÉE	Entrée TTL. Front montant déclenche prochain bac de temps.
ACC. INHIBER	Entrée TTL, échantillonné chaque déclenchement. Si élevé, provoque l'enregistrement en cours pour être ignoré (pas accumulé)
SUBTRACT	Entrée TTL, échantillonné chaque déclenchement. Si élevé, provoque l'enregistrement en cours à soustraire de l'accumulation (en mode bascule externe).

Résumé

L'objectif de ce stage est de réaliser un compteur de photons avec une carte FPGA; qui est un circuit logique reconfigurable, et ce pour remplacer un ancien appareil de mesure, le Stanford SR430, qui est le premier mesureur multi-canal, utilisé dans la manipulation de spectroscopie de fluorescence du laboratoire Lphia de l'Université d'Angers, et il est considéré comme un compteur de photons, permettant la mesure des temps de déclin de fluorescence.

Summary

The objective of this internship is to realize a photons counter with a FPGA card; which is a logical reconfigurable circuit, and it for replace a old measuring device, the Stanford SR430, who is the first multi-channel measuring device, used in the fluorescence spectroscopy experience in the Lphia laboratory of the Angers university, and it is considered as a photons counter, allouing the fluorescence decay time measurment.

MASTER 2 PHOTONIQUE SIGNAL ET IMAGERIE
UFR SCIENCE DE L'UNIVERSITE D'ANGERS
2014-2015

Comptage de photons multi-canal

Présenter par : MROURTI Karim

Sous la direction de Mr GUICHAOUA Dominique

Remerciements

Je tiens tout d'abord à remercier Mr Dominique Guichaoua qui m'a proposé ce stage et m'a guidé durant ce travaille. Je voudrais aussi remercier Léo Monnereau pour le matériel informatique. Et je remercie également Mme Nathalie Gaumer, Mr Patrice Razo et Mr Alain Mahot pour leur aides.

Sommaire

Annexes

Bibliographie

Article " Simple and Inexpensive FPGA-based Fast Multichannel Acquisition board ". By Sergey Polyakov & Joffrey Peters. January 27, 2015.

Article " KNJN FX2 FPGA development board "

www.Photonique.com

www.mon-club-elec.com

www.ni.com

www.thinksrs.com