

ProTIS : Dualité onde-corpuscule pour un photon unique

Livrable final

Anouk AZAIS, Jeanne DUBOEUF, Sami HAMRITI & Noé LE LAY

Avril 2025

Nous attestons que ce travail est original, que nous citons en référence toutes les sources utilisées et qu'il ne comporte pas de plagiat.



Table des matières

1	Introduction	3
1.1	Présentation du projet	3
1.2	Objectifs	3
1.3	Cahier des charges et performances attendues	4
2	Présentation de la fonction "compter les coïncidences"	5
2.1	Objectifs	5
2.2	Protocoles et tests	5
3	Présentation de la fonction "corrélérer les signaux"	7
3.1	Objectifs	7
3.2	Protocoles et tests	7
4	Intégration des sous-fonctionnalités	7
5	Notice d'utilisation	8
6	Bilan	8
7	Annexe	9
7.1	Carte imprimée	9
7.2	Code Python	10
7.3	Code Mbed	13
7.4	Nucléo 431KB	20

1 Introduction

1.1 Présentation du projet

Ce projet ProTIS s'inscrit dans l'avancement électronique d'un projet DéPHI encadré par Benjamin VEST. Les étudiants sur ce projet nommé "Interférences à un photon unique", sont : Jeanne DUBOEUF, Anouk AZAIS, Maël LECOEUCHE, Helwan OULYADI et Noé LE LAY.

L'objectif du projet DéPHI est de réaliser une source de photons uniques annoncés à partir d'une source de photons jumeaux. A partir de cette source, il est souhaité de démontrer expérimentalement la dualité onde-corpuscule d'un photon à l'aide respectivement de l'observation de franges d'interférences de photons uniques et de mesure de $g^{(2)}$. Le but de cette expérience est d'enrichir le sujet de TP mettant en évidence l'effet Hong-Ou-Mandel proposé aux élèves de Master 2 QLMN. Ce projet présente donc des enjeux pédagogiques avec l'extension du sujet de TP HOM. De plus, des enjeux d'agencement sont considérés afin de ne pas modifier l'expérience existante. L'électronique réalisée au cours du projet servira à renouveler celle précédemment utilisée. Enfin, des considérations environnementales seront prises en compte dans le développement de ce système.

C'est dans la partie "renouvellement" du circuit électronique que nous intervenons. Le travail que nous réalisons est donc orienté autour de cette sous-partie du projet DEPHI présenté précédemment.

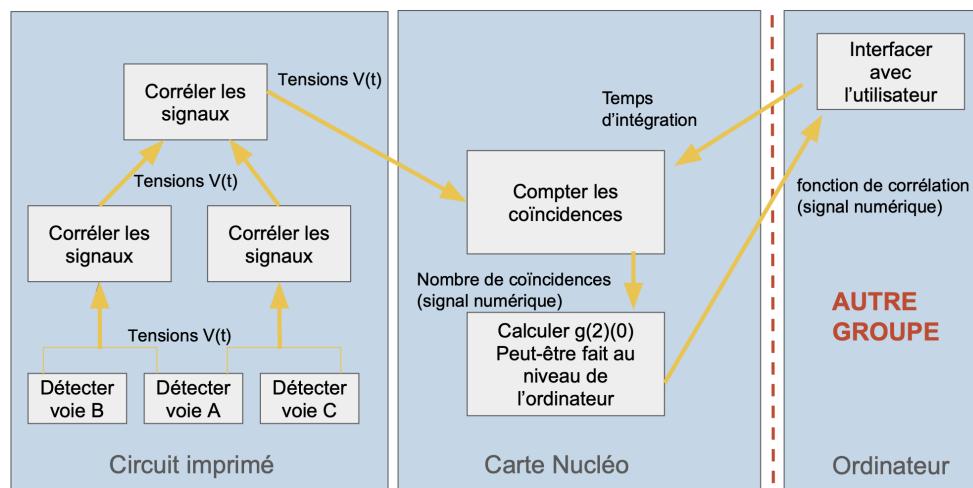


FIGURE 1 – Schéma fonctionnel du montage

1.2 Objectifs

Notre travail est essentiel pour pouvoir montrer qu'un photon se comporte comme un corpuscule. Pour montrer le caractère corpusculaire du photon, on cherche à calculer le coefficient de corrélation du second ordre $g^{(2)}$. Dans le cas d'une source de photons uniques, nous sommes sensés observer des anti-corrélations. Pour 3 détecteurs, on a :

$$g^{(2)}(0) = \frac{N_A N_{ABC}}{N_{AB} N_{AC}}$$

Avec N_A le nombre de photons sur la voie A du montage, N_{AB} le nombre de coïncidences sur les voies A et B, N_{AC} le nombre de coïncidences sur les voies A et C et N_{ABC} le nombre de coïncidences sur les voies A, B et C.

En effet un photon envoyé sur un cube séparateur sortira, s'il se comporte comme un corpuscule, par une seule des deux voies du cube, donc $g^{(2)} = 0$. Le schéma de notre montage est présenté figure 2. Pour réaliser la mesure de $g^{(2)}$, on intercepte un des deux photons jumeaux que l'on fait rentrer dans un interféromètre de Michelson tandis que le second photon joue le rôle d'annonceur (il annonce l'arrivée du 2e photon). Pour vérifier que notre source est annoncée, il faut que ces deux photons soient détectés en même temps en prenant en compte la différence de marche des chemins parcourus par les deux photons jumeaux. Pour réaliser cette expérience, nous avons réalisé le schéma figure 2 et avons réalisé une nouvelle carte électronique mesurant les coïncidences entre les différentes voies servant à calculer le $g^{(2)}(\tau)$.

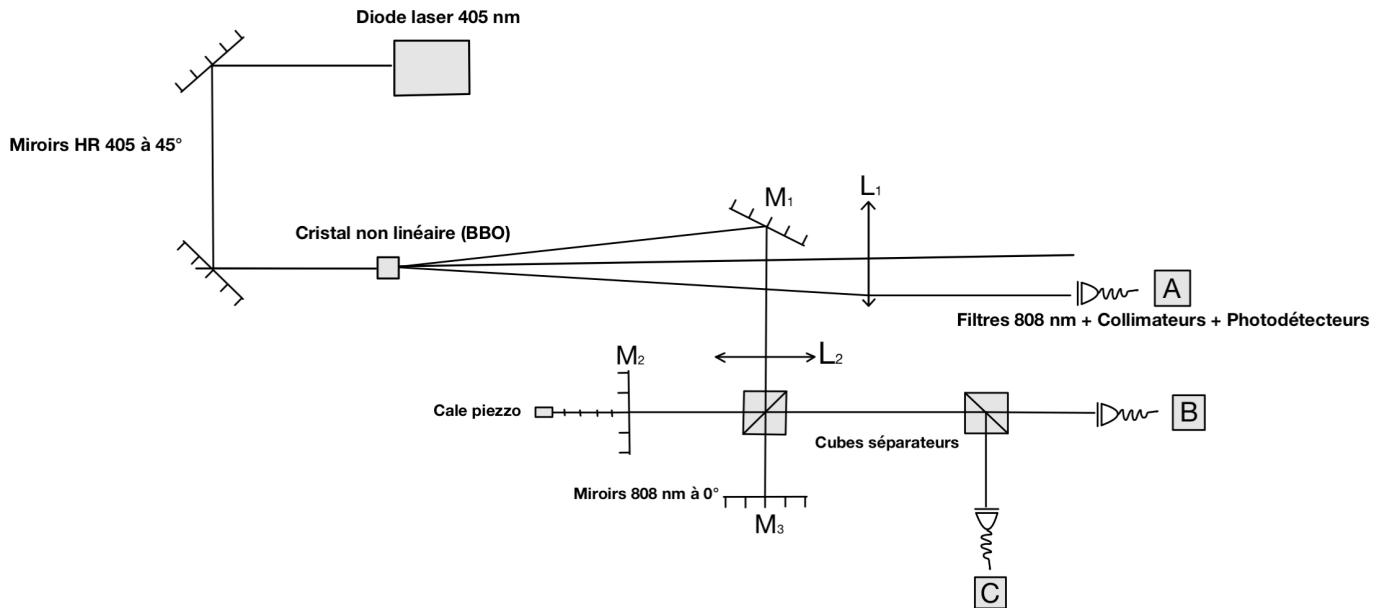


FIGURE 2 – Schéma du montage optique du projet DéPHI : Interférences à un photon unique [*réalisé par Helwan OULYADI, membre du projet DéPHI*]

Les objectifs que nous nous fixons dans le cadre du projet ProTIS sont les suivants :

- Redessiner le montage à 3 détecteurs pour l'intégrer sur une carte imprimée.
- Souder les composants nécessaires sur la carte imprimée.
- Écrire un script en langage C++ sous *KeilStudio* pour commander le comptage de fronts montants par la carte Nucléo.
- Faire le lien entre Python et la carte Nucléo (lien avec une interface, la Nucléo envoie les données mesurées à Python et Python envoie le temps d'intégration).
- Dessiner, sous SolidWorks, un boîtier qui accueillera la carte imprimée et la carte Nucléo.

La répartition du travail au sein du groupe est présentée dans le tableau figure 3.

1.3 Cahier des charges et performances attendues

- Réaliser des coïncidences entre des impulsions d'une largeur de 30 ns, d'amplitude 5 V et de fréquence 500 kHz.
- Compter le nombre des coïncidences ainsi créées sur 5 entrées de la carte Nucléo.

Date des séances	29/01	05/02	12/02	DEPHI
Sami	Schéma fonctionnel + théorie	Soudure faite à part deux composants. Il faut encore mesurer à l'oscilloscope les temps delta t	Il faut mesurer les delta t et vérifier que le montage ressort un signal en sortie + ensemble des composants du montage à 3 détecteurs soudés	
Anouk	Planning + théorie (les retards théoriques)	Soudure faite à part deux composants. Il faut encore mesurer à l'oscilloscope les temps delta t	Il faut mesurer les delta t et vérifier que le montage ressort un signal en sortie + ensemble des composants du montage à 3 détecteurs soudés	Vérification du montage à 2 détecteurs
Jeanne	Schéma fonctionnel + théorie (les retards théoriques)	Code Nucléo pour 4 entrées OK (voir T_int) - reste à vérifier le temps d'aveuglement des voies avec des signaux créneaux oscillants	Finir les objectifs de la séance précédente et donner la donnée T_int depuis une interface python	Vérification du montage à 2 détecteurs
Noé	Planning + théorie	Code Nucléo pour 4 entrées OK (voir T_int) - reste à vérifier le temps d'aveuglement des voies avec des signaux créneaux oscillants	Finir les objectifs de la séance précédente et donner la donnée T_int depuis une interface python	Montage de l'interféromètre

12/03	19/03	26/03	02/04	DEPHI
Préparation de l'AUDIT + Livrable	AUDIT (2h15) + Test du montage à 3 détecteurs	Commencer à préparer la présentation	Meeting pour la présentation / Présentation finale (2h15)	
Finalisation du montage à 3 détecteurs	AUDIT (2h15) + Test du montage à 3 détecteurs	Commencer à préparer la présentation	Meeting pour la présentation / Présentation finale (2h15)	Finalisation du projet - derniers tests
Finalisation du montage à 3 détecteurs	AUDIT (2h15) + Test du montage à 3 détecteurs	Faire le lien avec l'équipe d'Helwan (lien montage / nucléo / python / interface)	Meeting pour la présentation / Présentation finale (2h15)	Finalisation du projet - derniers tests
Préparation de l'AUDIT + Livrable	AUDIT (2h15) + Test du montage à 3 détecteurs	Faire le lien avec l'équipe d'Helwan (lien montage / nucléo / python / interface)	Meeting pour la présentation / Présentation finale (2h15)	Finalisation du projet - derniers tests

FIGURE 3 – Planning élaboré à la première séance avec en vert ce qui a été réalisé et en rouge ce qui n'a pas été fini ou qui doit être refait

2 Présentation de la fonction "compter les coïncidences"

La carte de circuit imprimé doit être refaite en raison d'un mauvais branchement décrit dans un article bibliographique. C'est pourquoi nous avons décidé de présenter le travail réalisé sur la carte Nucléo. L'objectif principal de cette fonction est d'obtenir une valeur numérique correspondant au nombre de coïncidences, ou "coups de photons". En effet, les détecteurs A, B et C sont des photodiodes à avalanche qui génèrent un pulse lorsqu'ils détectent un photon. Ces pulses sont ensuite corrélés par un circuit électronique. Dans le cadre du projet DePhi, nous cherchons à mesurer la fonction de corrélation $g(2)$ présentée précédemment, ce qui implique de mesurer le nombre de coïncidences N_A , N_{ABC} , N_{AB} et N_{AC} détectées au cours d'un temps d'intégration T choisi par l'utilisateur. Il est également nécessaire, pour le réglage de l'expérience, d'accéder aux valeurs de N_B et N_C .

La carte Nucléo est reliée au circuit imprimé et dispose d'une liaison série avec Python, qui constitue l'interface utilisateur. La communication électronique-numérique est illustrée dans le schéma en Annexe.

2.1 Objectifs

- Vérifier le comptage de fronts montant à une voie
- Vérifier le comptage de fronts montant à plusieurs voies et vérifier l'action quasi-simultanée des différentes fonctions d'interruption
- Permettre un comptage qui a bien lieu sur un temps T choisi par l'utilisateur et donc s'affranchir du temps de calcul et de transfert de données
- Ne pas calculer plusieurs fronts montants quand le pulse du circuit est oscillant
- Assurer la communication python-C++

2.2 Protocoles et tests

Dans un premier temps nous avons vérifié que la carte comptait bien les fronts montants.

On branche une entrée à l'alimentation qui produit des pulses de largeur 270ns, toutes les 50 μ s. Le temps d'intégration choisi était de 1s. Le nombre de coups attendu était $N_A=20\ 000$ et on a obtenu 18 623, soit une erreur de 7 %. En changeant la fréquence des pulses, la même erreur relative est obtenue. Mais en changeant le temps d'intégration, l'erreur relative est

modifiée. Pour un temps d'intégration de $2s$, nous avions une erreur relative de 3,5%. Cela nous a permis de comprendre que le temps de calcul et d'affichage limitait le nombre de pulses comptés.

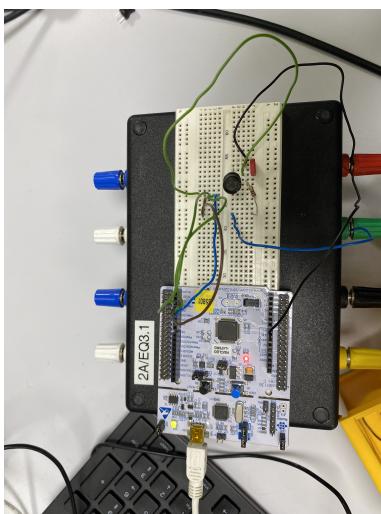
Conclusion : Les fronts montants sont bien comptés mais sur un temps $T-t_{calcul}$. Pour remédier à cela, on a introduit un deuxième ticker qui arrête le premier ticker quand on transmet les données. On relance le ticker une fois les calculs et l'affichage faits.

Dans un second temps, on introduit un signal d'entrée sur les 3 voies et on vérifie que le nombre de pulses comptés correspond bien à la fréquence des pulses multiplié par le temps d'intégration. Nous avons donc pu vérifier qu'en appliquant la solution suggérée, nous comptions effectivement le bon nombre de pulses sur toutes les voies (à 1 ou 2 pulses près). Tous les tests précédents ont été faits avec des signaux "propres". Cependant, il n'est pas garanti que la carte imprimée fournisse des signaux dits "propres". On se décide donc à générer des pulses avec des oscillations pour recréer les conditions expérimentales. On génère donc un pulse grâce à un GBF et on réalise un circuit RLC série entre l'entrée du GBF et la sortie (= entrée de l'oscilloscope). On se place dans le régime pseudo-périodique du circuit RLC. Les caractéristiques étant :

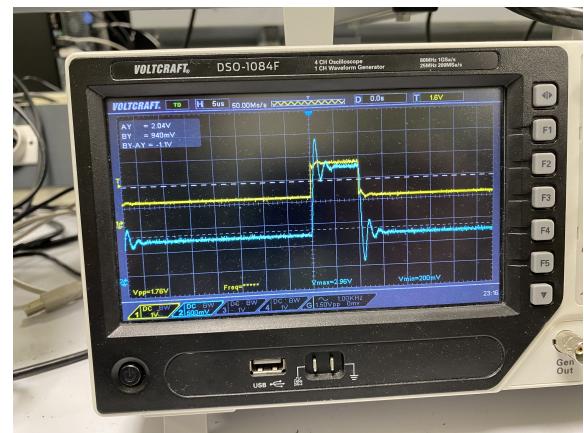
$$w_0 = \frac{1}{\sqrt{LC}} \quad \xi = \frac{R}{2} \sqrt{\frac{C}{L}}$$

La condition de pseudo-oscillation est : $\frac{1}{\sqrt{2}} < \xi < 1$. De plus, il faut choisir R pour avoir un temps de réponse faible et L et C pour avoir une bande passante assez grande. On choisit donc de prendre : $R \sim \text{quelques } \Omega$, $L = 100\mu H$ et $C = 10nF$.

Avec ces conditions expérimentales et en générant des pulses de l'ordre de $10\mu s$ (car le temps de réponse est trop grand pour des pulses plus faibles et on avait pas de résistance plus faible), nous avons pu vérifier que le nombre de pulses compté était bien le bon. Cela signifie que la carte Nucléo permet de compter les pulses dans les conditions de l'expérience.



(a) Circuit RLC



(b) Pulses générés et observés à l'oscilloscope

FIGURE 4 – Photos prises lors de la séance

Après avoir vérifié que notre programme permettait à la carte Nucléo de bien compter les pulses reçus, nous avons décidé de faire communiquer, grâce à une liaison série, Python et la carte Nucléo. Ainsi, une fois l'interface Python réalisée, un utilisateur peut indiquer le temps d'intégration qu'il veut (le temps de comptage de la Nucléo) et la carte Nucléo renverra les valeurs de N_A , N_B , N_C , N_{AB} , N_{AC} et N_{ABC} à l'interface Python.

Lors de l'écriture du programme, nous avons été confrontés à plusieurs problématiques dont la manière dont était envoyée les valeurs de T_{int} et de comptage de pulses. Nous avons considéré que l'envoi de valeurs sur 3 octets était largement suffisant pour l'utilisation que l'on en fait. De plus, en liaison série, les octets sont envoyés les uns après les autres et pas en même temps. Il a donc fallu prendre en compte cela dans notre programme. Toutes les remarques et commentaires de nos programmes sont disponibles en Annexe.

Une fois toute cette partie terminée, nous avons retesté nos programmes avec des signaux envoyés à la Nucléo à l'aide de plusieurs GBF. Nous nous sommes rendus compte que le nombre de pulses que pouvait compter la Nucléo en une seconde était

relativement faible ($\sim 400 \text{ kHz}$). L'objectif suivant est donc d'optimiser le nombre de coups comptés pour que l'on puisse utiliser la diode laser à pleine puissance et ainsi capter plus de photons. La raison pour laquelle la Nucléo ne compte pas plus de coups est à cause des fonctions d'interruption utilisées dans le programme Mbed. Celles-ci ralentissent la carte Nucléo qui ne peut pas faire plusieurs tâches en même temps. La solution que l'on a envisagée est d'utiliser 2 cartes Nucléo. Les programmes pour utiliser 2 cartes en même temps ont été réalisés pendant la semaine DéPHI. Cependant, M. VILLEMEJANE a trouvé une autre solution qui est d'utiliser une carte Nucléo plus rapide (Nucléo G431KB). Lors de la dernière séance de ProTIS (12/03/2025), nous avons pu tester cette nouvelle carte qui peut monter jusqu'à 800 000 coups par seconde, ce qui est suffisant pour notre projet DéPHI et donc ProTIS.

3 Présentation de la fonction "corréler les signaux"

Nous avons détecté une erreur dans la bibliographie et donc dans les branchements de notre première version de circuit imprimé. Une fois la correction faite, nous avons pu travailler sur notre nouvelle version de plaque. De plus, nous avons utilisé un modèle de carte Nucléo plus rapide (G431KB) pour notre comptage de coïncidences (voir Annexe 1).

3.1 Objectifs

- Souder tous les composants sur la plaque.
- S'assurer que l'amplitude des signaux de sortie ne dépasse pas 3.3 V pour ne pas griller la carte Nucléo.
- S'assurer de la réalisation des corrélations entre les signaux quelque soit leur fréquence initiale.
- S'assurer que le signal de sortie possède un front montant "franc" pour effectuer le comptage par la carte Nucléo.

3.2 Protocoles et tests

Une fois l'ensemble des composants soudés, nous avons envoyé des signaux pulsés au GBF sur notre carte. Dans un premier temps, nous avons utilisés des impulsions larges (250 ns), d'amplitude 5 V et de fréquence 50 kHz. A l'aide d'une sonde et d'un oscilloscope, nous avons vérifié si les voies de sortie émettaient un signal. En particulier, nous avons vérifié si les sorties donnant le signal d'entrée décalé dans le temps avaient la même allure que le signal initial.

Une fois satisfaits, nous avons vérifié, toujours à l'aide d'une sonde et d'un oscilloscope, l'amplitude des signaux à chaque sortie. Cette amplitude devait être supérieure à 2 V car il s'agit du seuil de détection pour le comptage des fronts montants avec le code en C++, mais inférieure à 3.3 V pour ne pas griller la carte Nucléo. Ainsi, des ponts diviseurs de tension sont placés entre nos portes logiques et la carte Nucléo, afin de ramener l'amplitude initiale de 5 V dans la gamme acceptée par la Nucléo. Nous avons remarqué que le signal sortant de la voie AB (les corrélations entre les voies A et B) avait une amplitude comprise entre 2 et 3 V, donc uniquement pour cette voie, nous avons fait le choix de ne pas utiliser de pont diviseur.

Le dernier test que nous avons mis en place consistait à réduire les largeurs des impulsions d'entrée et à augmenter leur fréquence. Par la présence de circuits RC, nous remarquons que la largeur des pulses de sortie dépend des valeurs de R et de C choisies. Notre système n'a eu aucun mal à corrélérer les signaux courts et de fréquence allant jusqu'à 500 kHz.

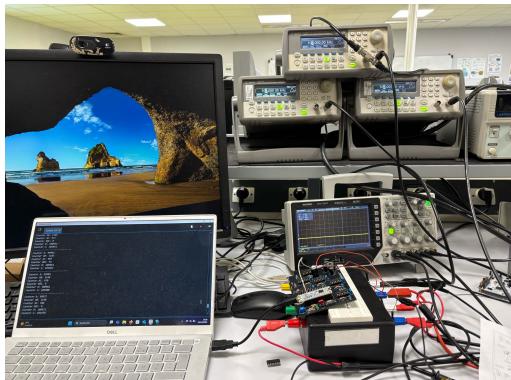
4 Intégration des sous-fonctionnalités

L'apogée de notre projet consistait à mettre en communication la partie hardware et software. Ce lien a été effectué avec succès. Afin de confirmer cette réussite, nous avons mis en place un test assez simple. Nous avons décidé d'envoyer, grâce à 3 GBF, 3 signaux d'entrée synchronisés sur les voies A, B et C (figure 5). La plus grande difficulté ici était d'assurer ladite synchronisation. Pour cela, nous avons utilisé 4 GBF, 1 en mode "maître" et 3 en mode "esclave". La synchronisation des 3 esclaves était gérée par un déclencheur extérieur annoncé par le maître. Les signaux créneaux ainsi générés avaient une largeur de 20 ns, une fréquence allant jusqu'à 200 kHz et une amplitude de 5 V. A l'aide d'une sonde et d'un oscilloscope, nous avons vérifié que l'amplitude des signaux de sortie était bien comprise entre 2 V et 3.3 V. En ajoutant la carte Nucléo au montage, nous obtenions sur chaque voie de sortie, la même valeur de fréquence. Ce résultat était totalement attendu car toutes les voies d'entrées étaient synchronisées et donc tous les signaux étaient parfaitement corrélés. Remarquons toutefois que notre système de détection est sensible aux décalages temporels de l'ordre de la nanoseconde. Le décalage issu du temps de propagation du signal électrique à travers les câbles coaxiaux était suffisamment important pour perdre nos parfaites corrélations. Ainsi, il faut veiller à utiliser des câbles coaxiaux de même longueur. Pour remarquer cette chute du nombre de corrélations, il suffit de

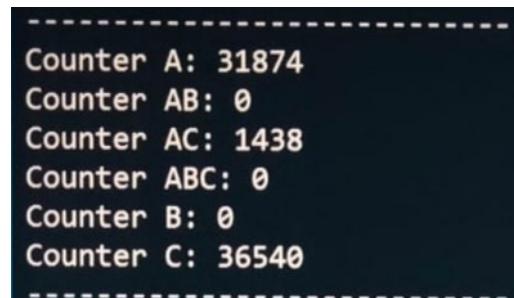
modifier la phase d'un des signaux d'entrée. Nous avons remarqué que le nombre de coïncidences chute complètement quand la phase introduite est $\geq 40^\circ$.

5 Notice d'utilisation

- Pour utiliser notre carte de comptage, il faut
- Alimenter la plaquette avec une tension continue comprise entre 7 et 12 V.
 - Brancher la carte Nucléo à un ordinateur et lui fournir le code de comptage.
 - Injecter les signaux d'entrée sur les ports indiqués sur la plaquette (A, B et C). Ces signaux peuvent provenir d'un GBF ou d'une photodiode à avalanche. Vérifier que leur amplitude est de 5 V maximum.
 - Lancer le code Python et observer les comptages réalisés.



(a) Photo du prototype final en cours de la phase de test



(b) Affichage du nombre de coups comptés sur les voies A, C et AC avec des signaux de vrais photons

FIGURE 5 – Photos prises lors de la phase de test

6 Bilan

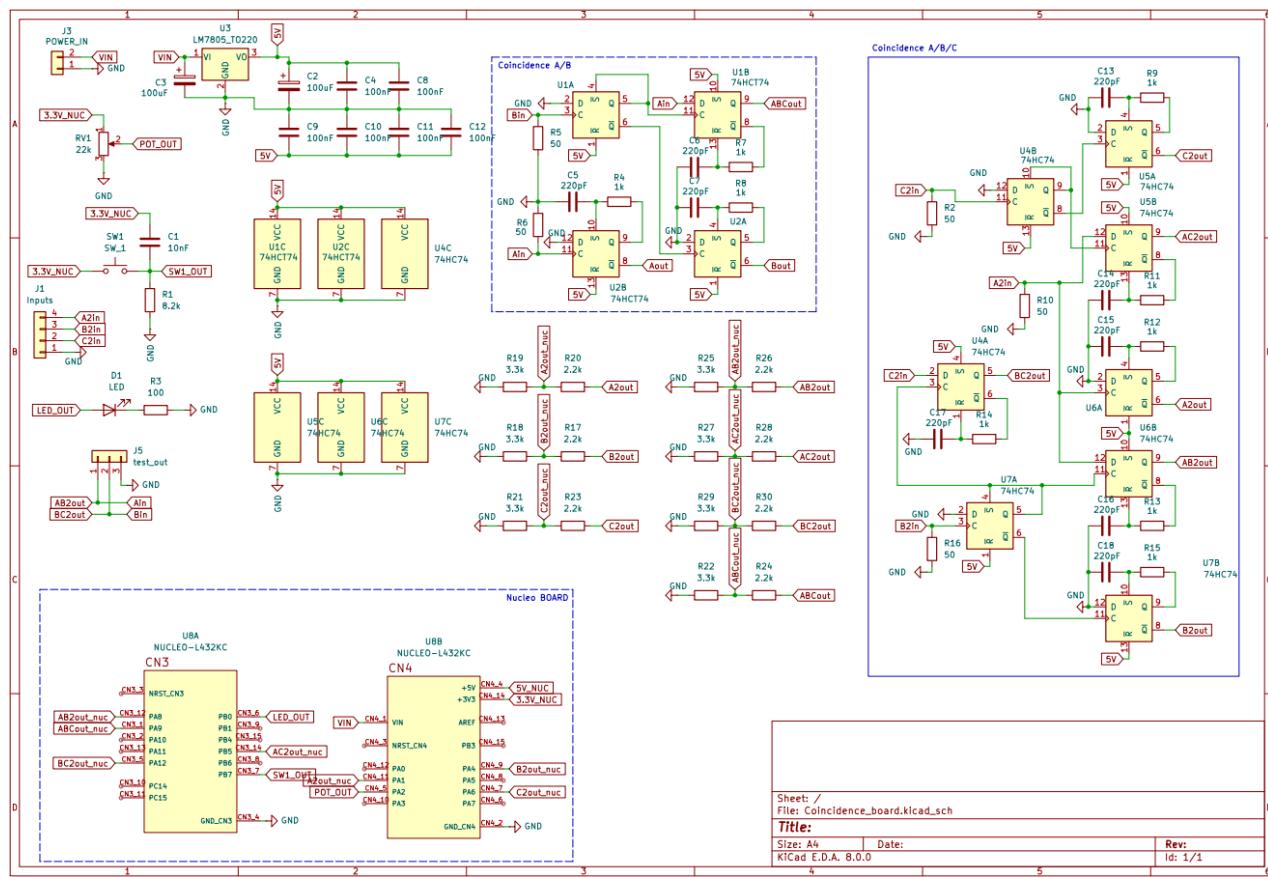
A la fin des séances de ProTIS, nous avons réussi à former des coïncidences entre trois signaux d'entrée en utilisant une plaquette imprimée. Nous avons aussi pu caractériser ces signaux de sortie. La connexion entre la partie software et hardware est réussie : la carte Nucléo compte le bon nombre de fronts montants qu'elle reçoit du système. Afin d'intégrer nos avancées dans notre projet DePHI, il serait souhaitable d'effectuer des tests avec des vrais signaux, ceux issus des photodiodes qui collectent nos photons uniques. Nous avons bon espoir quant à la réussite de cette opération.

Le travail en équipe s'est bien passé. Nous avons formé 2 équipes qui ont travaillé pur l'une d'entre elle sur la carte imprimée (Sami et Anouk) et sur le code Mbed et Python pour l'autre (Jeanne et Noé). Dans chacune des équipes, la bonne communication et les idées de chacun ont permis de faire avancer le projet dans les différentes missions. Cependant, la communication entre les deux équipes a été un peu négligée. Nous parlions de nos avancées entre nous mais nous aurions pu faire des points d'avancement en fin de séance et noter les problèmes rencontrés. Ainsi, en relisant ces notes au début des séances suivantes, nous aurions pu gagner un peu de temps.

Nous avons tout de même réussi à mener à bien notre projet malgré des obstacles majeurs autour de la carte imprimée qui ont été surmontés.

7 Annexe

7.1 Carte imprimée



7.2 Code Python

```

1  # -*- coding: utf-8 -*-
2  """
3
4  Created on Wed Feb 12 15:48:20 2025
5
6  ****
7
8  @author: Le Lay Noé et Duboeuf Jeanne
9  Written thanks to the work of Villemejane Julien :
10 → https://lense.institutoptique.fr/mine/python-pyserial-premier-script/
11 ****
12
13 Le but de ce programme est de créer un lien entre python et
14 la carte Nucléo G431KB (en langage C++) pour pouvoir transférer
15 à la carte Nucléo une valeur T_int de temps d'intégration.
16
17 ****
18
19 * Tested with G431KB / Mbed OS 6
20
21 ****
22
23 * Solec / https://solecgroup.wordpress.com/
24 * ProTIS / https://lense.institutoptique.fr/
25
26 """
27
28 # Avant de lancer le programme, il faut mettre 'pip install pyserial' dans la console et pylab et il faut
29 → ouvrir une fenêtre 'Pylab'
30
31 import serial
32 import serial.tools.list_ports
33 import struct
34 import time
35
36
37 serial_port = serial.tools.list_ports.comports() # L'objet serial_port est alors une liste contenant tous
38 → les objets de type port de communication série de votre machine

```

```

1
2     for port in serial_port:
3         # Cela affiche chaque élément de la liste serial_port, c'est à dire tous les ports de communication
4         #→ série de l'ordinateur
5         print(f"{port.name} // {port.device} // D={port.description}")
6
7
8     ser = serial.Serial("COM12", baudrate=9600) # Cela permet d'ouvrir une connexion avec le port série
9     #→ "COM70" -> Vérifier que la Nucléo se trouve bien sur le COM70
10    # "COM11" : port - obligatoire pour spécifier le nom du port auquel se connecter
11    # baudrate=9600 : baudrate (int) - la vitesse de transfert des données
12
13    """
14        Il est indispensable de libérer le port série une fois que vous avez fini
15        de l'utiliser. La commande pour le faire est la suivante : ser.close()
16    """
17
18
19
20    # Cela crée une chaîne de caractères (seulement un octet est envoyé à la fois)
21    T_int = 1000
22    byt_val = T_int.to_bytes(2, 'big')
23    print(byt_val)
24    ser.write(byt_val)
25
26
27    """
28
29    """
30    *****
31    Le programme qui suit permet de lire des sonnées envoyées par la carte Nucléo
32
33    Remarque :
34        - Les données sont envoyées sur 3 octets
35        - Il faut s'assurer qu'il n'y a pas de printf dans le programme C++ de la Nucléo
36    *****
37
38    ser.flushInput()
39
40    # Fonction pour lire les compteurs envoyés par le Nucleo
41    def read_counters():
42        # Lire 18 octets (3 octets par compteur)
43        data = ser.read(18)
44

```

```

1
2     if len(data) == 18:
3         # Convertir les 18 octets reçus en 6 compteurs (3 octets pour chaque compteur)
4         counter_A = struct.unpack('>I', b'\x00' + data[0:3])[0] # Utilise '>I' pour un entier Big Endian
5         counter_AB = struct.unpack('>I', b'\x00' + data[3:6])[0]
6         counter_AC = struct.unpack('>I', b'\x00' + data[6:9])[0]
7         counter_ABC = struct.unpack('>I', b'\x00' + data[9:12])[0]
8         counter_B = struct.unpack('>I', b'\x00' + data[12:15])[0]
9         counter_C = struct.unpack('>I', b'\x00' + data[15:18])[0]
10
11     return counter_A, counter_AB, counter_AC, counter_ABC, counter_B, counter_C
12 else:
13     print("Erreur de lecture des données")
14     return None, None, None
15
16
17
18 # Boucle principale
19 while True:
20     try:
21         counter_A, counter_AB, counter_AC, counter_ABC, counter_B, counter_C = read_counters()
22
23     if counter_A is not None:
24         # Afficher les résultats
25         print("-----")
26         print(f"Counter A: {counter_A}")
27         print(f"Counter AB: {counter_AB}")
28         print(f"Counter AC: {counter_AC}")
29         print(f"Counter ABC: {counter_ABC}")
30         print(f"Counter B: {counter_B}")
31         print(f"Counter C: {counter_C}")
32
33     # Attendre un peu avant de lire à nouveau les données
34     time.sleep(1)
35
36 except KeyboardInterrupt:
37     ser.close()
38     print("Fin du programme.")
39     break
40
41 # Ne pas oublier de mettre ser.close() dans la console pour fermer le port avant de relancer le code
42

```

7.3 Code Mbed

```

1  /*
2   * Structure of a main file for embedded project @ Solec/LEnSE
3   *
4   * This code (explain in 2 lines what is the main function of this code)
5
6
7 This code will count the number of pulse there is on each input of the Nucleo card (there is a counter for
8    ↵ each input). The goal is to have
9 the number of pulse on each input during a time T_int. When this time is over, the counters resets to 0.
10 With that, we give the datas of the counters to Python which will calculate the g2 fonction (which depends
     ↵ on every pulse
11 on every input).
12
13
14
15 ****
16 * Pinout :
17 *      - signal_A / Input for a numerical signal on PA1 port of the Nucleo
18 *      - signal_AB / Input for a numerical signal on PA8 port of the Nucleo
19 *      - signal_AC / Input for a numerical signal on PB5 port of the Nucleo
20 *      - signal_ABC / Input for numerical signal on PA9 port of the Nucleo
21 *      - signal_B / Input for numerical signal on PA4 port of the Nucleo
22 *      - signal_C / Input for numerical signal on PA6 port of the Nucleo
23 *      - T_int / It is the time a user choose to do the measurement of g2
24 ****
25 * Tested with Nucleo G431KB board / Mbed OS 6
26 ****
27 * Authors : J.DUBOEUF and N.LE LAY / Solec Group - Creation 2025/02/05
28 ****
29 * Pre-compiler json file at the end of this file
30 ****
31 * Solec / https://solecgroup.wordpress.com/
32 * ProTIS / https://lense.institutoptique.fr/
33 * Based on Mbed OS 6 example : mbed-os-example-blinky-baremetal
34 */
35 #include "mbed.h"
36

```

```

1
2 InterruptIn signal_A(A1);           // It defines a numerical signal to the PA1 port of the Nucleo.
3 InterruptIn signal_AB(D9);         // It defines a numerical signal to the PA8 port of the Nucleo.
4 InterruptIn signal_AC(D11);         // It defines a numerical signal to the PB5 port of the Nucleo.
5 InterruptIn signal_ABC(D1);        // It defines a numerical signal to the PA9 port of the Nucleo.
6 InterruptIn signal_B(A3);          // It defines a numerical signal to the PA4 port of the Nucleo.
7 InterruptIn signal_C(A5);          // It defines a numerical signal to the PA6 port of the Nucleo.
8 BufferedSerial usb_pc(USBTX, USBRX); //It connects the Nucleo to the computer in order to transmit data
→   through python
9
10
11 int     counter_A = 0;             // It sets counter to 0, counter_A is the number of time a pulse appears on
→   A output
12 int     counter_AB = 0;            // It sets counter to 0, counter_AB is the number of time a pulse appears on
→   AB output
13 int     counter_AC = 0;            // It sets counter to 0, counter_AC is the number of time a pulse appears on
→   AC output
14 int     counter_ABC = 0;           // It sets counter to 0, counter_ABC is the number of time a pulse appears
→   on ABC output
15 int     counter_B = 0;             // It sets counter to 0, counter_B is the number of time a pulse appears on
→   ABC output
16 int     counter_C = 0;             // It sets counter to 0, counter_C is the number of time a pulse appears on
→   ABC output
17
18
19 Ticker reset_ticker;              // Ticker which resets the counters
20
21
22 // Integration Time
23 int T_int = 1;
24 int compteur_octet = 0; // It sets a cpt for every bit received
25 int T_int_values[2] = {0, 0}; // It sets a liste of 2 elements in order to change T_int when we receive a
→   new value
26
27
28
29
30 // Variables to blind an input during x seconds (timeout)
31 bool blind_A = false;
32 bool blind_AB = false;
33 bool blind_AC = false;
34 bool blind_ABC = false;
35 bool blind_B = false;
36 bool blind_C = false;
37

```

```

1
2
3 // Durée du timeout (par exemple 1 seconde)
4 #define TIMEOUT 1.0 // Timeout in seconds
5
6
7 void rising_A() {
8     if (!blind_A) { // if signal_A isn't blinded
9         counter_A++; // add +1 to counter_A
10        blind_A = true; // Blinds signal_A during a TIMEOUT
11        thread_sleep_for(TIMEOUT * 1e-9 * 300); // waits timeout
12        blind_A = false; // allows the detection on signal_A again
13    }
14 }
15
16
17
18 void rising_AB() {
19     if (!blind_AB) { // if signal_AB isn't blinded
20         counter_AB++; // add +1 to counter_AB
21         blind_AB = true; // Blinds signal_AB during a TIMEOUT
22         thread_sleep_for(TIMEOUT * 1e-9 * 300); // waits timeout
23         blind_AB = false; // allow the detection on signal_AB again
24     }
25 }
26
27
28 void rising_AC() {
29     if (!blind_AC) { // if signal_AC isn't blinded
30         counter_AC++; // add +1 to counter_AC
31         blind_AC = true; // Blinds signal_AC during a TIMEOUT
32         thread_sleep_for(TIMEOUT * 1e-9 * 300); // waits timeout
33         blind_AC = false; // allow the detection on signal_AC again
34     }
35 }
36
37
38 void rising_ABC() {
39     if (!blind_ABC) { // if signal_ABC isn't blinded
40         counter_ABC++; // add +1 to counter_ABC
41         blind_ABC = true; // Blinds signal_ABC during a TIMEOUT
42         thread_sleep_for(TIMEOUT * 1e-9 * 300); // waits timeout
43         blind_ABC = false; // allow the detection on signal_ABC again
44     }
45 }

```

```

1
2 void rising_B() {
3     if (!blind_B) { // if signal_ABC isn't blinded
4         counter_B++; // add +1 to counter_ABC
5         blind_B = true; // Blinds signal_ABC during a TIMEOUT
6         thread_sleep_for(TIMEOUT * 1e-9 * 300); // waits timeout
7         blind_B = false; // allow the detection on signal_ABC again
8     }
9 }
10
11
12
13 void rising_C() {
14     if (!blind_C) { // if signal_ABC isn't blinded
15         counter_C++; // add +1 to counter_ABC
16         blind_C = true; // Blinds signal_ABC during a TIMEOUT
17         thread_sleep_for(TIMEOUT * 1e-9 * 300); // waits timeout
18         blind_C = false; // allow the detection on signal_ABC again
19     }
20 }
21
22
23 void send_counters() {
24     uint8_t buffer[18];
25
26     // Encode counter_A into 3 bytes
27     buffer[0] = (counter_A >> 16) & 0xFF;
28     buffer[1] = (counter_A >> 8) & 0xFF;
29     buffer[2] = counter_A & 0xFF;
30
31
32     // Encode counter_AB into 3 bytes
33     buffer[3] = (counter_AB >> 16) & 0xFF;
34     buffer[4] = (counter_AB >> 8) & 0xFF;
35     buffer[5] = counter_AB & 0xFF;
36
37
38     // Encode counter_AC into 3 bytes
39     buffer[6] = (counter_AC >> 16) & 0xFF;
40     buffer[7] = (counter_AC >> 8) & 0xFF;
41     buffer[8] = counter_AC & 0xFF;
42

```

```

1
2     // Encode counter_ABC into 3 bytes
3     buffer[9] = (counter_ABC >> 16) & 0xFF;
4     buffer[10] = (counter_ABC >> 8) & 0xFF;
5     buffer[11] = counter_ABC & 0xFF;
6
7
8     // Encode counter_B into 3 bytes
9     buffer[12] = (counter_B >> 16) & 0xFF;
10    buffer[13] = (counter_B >> 8) & 0xFF;
11    buffer[14] = counter_B & 0xFF;
12
13
14    // Encode counter_C into 3 bytes
15    buffer[15] = (counter_C >> 16) & 0xFF;
16    buffer[16] = (counter_C >> 8) & 0xFF;
17    buffer[17] = counter_C & 0xFF;
18
19
20
21
22    // Send the 12-byte buffer through the serial port
23    usb_pc.write(buffer, 18);
24 }
25
26
27
28
29 void reset_counters() {
30     // Arrêter et réinitialiser le Ticker
31     reset_ticker.detach(); // Arrêter le Ticker existant
32     // It prints the values of counters after T_int has passed
33     // printf("Counter A: %d\n", counter_A);
34     // printf("Counter AB: %d\n", counter_AB);
35     // printf("Counter AC: %d\n", counter_AC);
36     // printf("Counter ABC: %d\n", counter_ABC);
37     // printf("T_int : %d\n", T_int);
38

```

```

1
2
3     send_counters();
4     // It resets the counters
5     counter_A = 0;
6     counter_AB = 0;
7     counter_AC = 0;
8     counter_ABC = 0;
9     counter_B = 0;
10    counter_C = 0;
11
12    // Réattacher le Ticker avec la nouvelle valeur de T_int
13    reset_ticker.attach(&reset_counters, T_int);
14 }
15
16 void usb_pc_ISR(void) {
17     uint8_t rec_data_pc[2]; // Array to store the 2 bytes received from USB serial
18     int rec_length = 0;
19     int m = 0;
20
21     // Check if data is available on the serial port
22     if (usb_pc.readable()) {
23         // Read 2 bytes from the USB serial port into the array
24         rec_length = usb_pc.read(rec_data_pc, 2); // Read 2 bytes into the array
25         m = rec_data_pc[0]; // Get the first byte (most significant byte)
26         compteur_octet++; // Increment the byte counter
27
28         // If the first byte has been received (compteur_octet == 1)
29         if (compteur_octet == 1) {
30             // Combine the two bytes into an integer value (Big Endian)
31             T_int = m * 256; // The first byte is the most significant byte
32         }
33
34         // If the second byte has been received (compteur_octet == 2)
35         if (compteur_octet == 2) {
36             // Combine the second byte to complete the integer value (Big Endian)
37             T_int = (T_int + m) * 1e-3; // Add the second byte and apply scaling (e.g., convert to
38             // seconds)
39             compteur_octet = 0; // Reset the byte counter after both bytes have been received
40
41             // Update the list of the last two values of T_int
42             T_int_values[0] = T_int_values[1]; // The oldest value becomes the previous element
43             T_int_values[1] = T_int; // The new value becomes the latest element
44

```

```

1           // Detach the old Ticker and attach a new one with the updated T_int value
2   reset_ticker.detach(); // Detach the old Ticker
3   reset_ticker.attach(&reset_counters, T_int_values[1]); // Attach a new Ticker with the latest
   ↳ T_int value
4
5   }
6 }
7 }
8
9 int main() {
10    usb_pc.set_baud(9600);
11    usb_pc.sigio(callback(usb_pc_ISR));
12    // We detect the rising on each input signal
13    signal_A.rise(&rising_A); // Detect a rising on signal_A
14    signal_AB.rise(&rising_AB); // Detect a rising on signal_AB
15    signal_AC.rise(&rising_AC); // Detect a rising on signal_AC
16    signal_ABC.rise(&rising_ABC); // Detect a rising on signal_ABC
17    signal_B.rise(&rising_B); // Detect a rising on signal_B
18    signal_C.rise(&rising_C); // Detect a rising on signal_C
19
20
21
22
23    // We stop the detection after T_int has passed and we reset the counters
24    reset_ticker.attach(&reset_counters, T_int); // It runs the reset_counters function every T_int
25
26    while (true) {
27        wait_us(1000);
28    }
29 }
30
31 /* mbed_app.json - Precompiler option for Mbed Studio and Keil Studio.
32 {
33     "requires": ["bare-metal"],
34     "target_overrides": {
35         "*": {
36             "target.c_lib": "small",
37             "target.printf_lib": "minimal	printf",
38             "platform.minimal	printf-enable-floating-point": true,
39             "platform.stdio-baud-rate": 9600,
40             "platform.stdio-minimal-console-only": true
41         }
42     }
43 }
44 */

```

7.4 Nucléo 431KB



NUCLEO-XXXXXX

Data brief

STM32 Nucleo-32 boards



NUCLEO-G431KB example. Boards with different references show different layouts. Picture is not contractual.

Features

- Common features
 - STM32 microcontroller in 32-pin package
 - 1 user LED
 - 1 reset push-button
 - Board connectors:
 - Arduino™ Nano V3 expansion connector
 - Micro-AB USB connector for the ST-LINK
 - Flexible power-supply options: ST-LINK, USB V_{BUS} or external sources
 - On-board ST-LINK debugger/programmer with USB re-enumeration capability: mass storage, Virtual COM port and debug port
 - Comprehensive free software libraries and examples available with the STM32Cube MCU Package
 - Support of a wide choice of Integrated Development Environments (IDEs) including IAR™, Keil® and GCC-based IDEs
- Board-specific features
 - 24 MHz crystal oscillator
 - Arm® Mbed Enabled™ compliant

Description

The STM32 Nucleo-32 board provides an affordable and flexible way for users to try out new concepts and build prototypes by choosing from the various combinations of performance and power consumption features, provided by the STM32 microcontroller.

The Arduino™ Nano V3 connectivity support allows the easy expansion of the functionality of the STM32 Nucleo open development platform with a wide choice of specialized shields.

The STM32 Nucleo-32 board does not require any separate probe as it integrates the ST-LINK debugger/programmer.

The STM32 Nucleo-32 board comes with the STM32 comprehensive free software libraries and examples available with the STM32Cube MCU Package.

arm MBED



1 Ordering information

To order an STM32 Nucleo-32 board, refer to **Table 1**. For a detailed description of each board, refer to its user manual on the product web page. Additional information is available from the datasheet and reference manual of the target STM32.

Table 1. List of available products

Order code	Board reference	User manual	Target STM32	Differentiating features
NUCLEO-F031K8	MB1180	UM1956	STM32F031K8T6	<ul style="list-style-type: none"> ▪ Arm® Mbed Enabled™ ▪ ST-LINK/V2-1
NUCLEO-F042K8			STM32F042K8T6	<ul style="list-style-type: none"> ▪ Arm® Mbed Enabled™ ▪ ST-LINK/V2-1
NUCLEO-F301K8			STM32F301K8T6	<ul style="list-style-type: none"> ▪ ST-LINK/V2-1
NUCLEO-F303K8			STM32F303K8T6	<ul style="list-style-type: none"> ▪ Arm® Mbed Enabled™ ▪ ST-LINK/V2-1
NUCLEO-G031K8	MB1455	UM2591	STM32G031K8T6U	<ul style="list-style-type: none"> ▪ ST-LINK/V2-1
NUCLEO-G431KB	MB1430	UM2397	STM32G431KBT6U	<ul style="list-style-type: none"> ▪ STLINK-V3E ▪ 24 MHz crystal oscillator
NUCLEO-L011K4	MB1180	UM1956	STM32L011K4T6	<ul style="list-style-type: none"> ▪ Arm® Mbed Enabled™ ▪ ST-LINK/V2-1
NUCLEO-L031K8			STM32L031K8T6	<ul style="list-style-type: none"> ▪ Arm® Mbed Enabled™ ▪ ST-LINK/V2-1
NUCLEO-L412KB			STM32L412KBU6U	<ul style="list-style-type: none"> ▪ ST-LINK/V2-1
NUCLEO-L432KC			STM32L432KCU6U	<ul style="list-style-type: none"> ▪ Arm® Mbed Enabled™ ▪ ST-LINK/V2-1

1.1 Product marking

Evaluation tools marked as "ES" or "E" are not yet qualified and therefore not ready to be used as reference design or in production. Any consequences deriving from such usage will not be at ST charge. In no event, ST will be liable for any customer usage of these engineering sample tools as reference design or in production.

"E" or "ES" marking examples of location:

- On the targeted STM32 that is soldered on the board (for illustration of STM32 marking, refer to the STM32 datasheet "Package information" paragraph at the www.st.com website).
- Next to the evaluation tool ordering part number that is stuck or silk-screen printed on the board.

Some boards feature a specific STM32 device version, which allows the operation of any bundled commercial stack/library available. This STM32 device shows a "U" marking option at the end of the standard part number and is not available for sales.

In order to use the same commercial stack in his application, a developer may need to purchase a part number specific to this stack/library. The price of those part numbers includes the stack/library royalties.

1.2 Codification

The meaning of the codification is explained in Table 2.



NUCLEO-XXXXKX
Codification

Table 2. Codification explanation

NUCLEO-XXYYKT	Description	Example: NUCLEO-G431KB
XX	MCU series in STM32 Arm Cortex MCUs	STM32G4 Series
YY	MCU product line in the series	STM32G431
K	STM32 package pin count	32 pins
T	STM32 Flash memory size: <ul style="list-style-type: none"> • A for 16 Kbytes • B for 32 Kbytes • C for 64 Kbytes • D for 128 Kbytes • E for 256 Kbytes 	128 Kbytes

The order code is mentioned on a sticker placed on the top side of the board.



NUCLEO-XXXXXX
Development environment

2 Development environment

2.1 System requirements

- Windows® OS (7, 8 and 10), Linux® 64-bit, or macOS®
- USB Type-A to Micro-B cable

Note: *macOS® is a trademark of Apple Inc. registered in the U.S. and other countries.*

2.2 Development toolchains

- Keil® MDK-ARM⁽¹⁾
- IAR™ EWARM⁽¹⁾
- GCC-based IDEs
- Arm® Mbed™⁽²⁾ online⁽³⁾ (see mbed.org)

Note:
1. *On Windows® only.*
2. *Arm and Mbed are registered trademarks or trademarks of Arm Limited (or its subsidiaries) in the US and or elsewhere.*
3. *Refer to the www.mbed.com website and to the "Ordering information" section to determine which order codes are supported.*

2.3 Demonstration software

The demonstration software, included in the STM32Cube MCU Package corresponding to the on-board microcontroller, is preloaded in the STM32 Flash memory for easy demonstration of the device peripherals in standalone mode. The latest versions of the demonstration source code and associated documentation can be downloaded from www.st.com.



NUCLEO-XXXXXX

Revision history

Table 3. Document revision history

Date	Version	Changes
08-Sep-2015	1	Initial release.
15-Jan-2016	2	Added <i>Table 1: Device summary</i> and updated <i>Table 2: Ordering information</i> .
09-Jun-2016	3	Updated <i>Section : Description</i> and <i>Section : System requirements</i> to add NUCLEO-L432KC.
07-Jul-2017	4	Updated <i>Features</i> .
		Extended document scope to NUCLEO-L412KB.
23-Aug-2018	5	Updated <i>Table 1: Device summary</i> , <i>System requirements</i> , <i>Development toolchains</i> , and <i>Ordering information</i> . Added <i>Demonstration software</i> .
13-Nov-2018	6	Extended document scope to NUCLEO-F301K8: updated <i>Features</i> , <i>Table 1: Device summary</i> , and <i>Table 2: Ordering information</i> .
10-May-2019	7	<p>Revised the entire document to accommodate multiple feature combinations:</p> <ul style="list-style-type: none"> ▪ Reorganized <i>Features</i> ▪ Updated <i>Description</i> ▪ Updated <i>Ordering information</i> ▪ Added <i>Development environment</i> ▪ Updated <i>Table 1. List of available products</i> and <i>Table 2. Codification explanation</i> <p>Extended document scope to the NUCLEO-G431KB board.</p>
5-Jun-2019	8	Extended document scope to the NUCLEO-G031K8 board. Global product description: updated <i>Features</i> , <i>Ordering information</i> , and <i>System requirements</i> .



NUCLEO-XXXXXX

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. For additional information about ST trademarks, please refer to www.st.com/trademarks. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2019 STMicroelectronics – All rights reserved