

Rapport

Projet d'Architecture des systèmes embarqués - Quartus Carte Radar 2D

Membre : Vladislav Levovitch BALAYAN
Contribution précieuse : Daniel FERREIRA LARA

Aide rédactionnelle, assistance au débogage, support pour le code, et remerciements à ceux qui m'ont guidé et apporté leur expertise : Ayoub, Nicolas, Liza, Salma, Farah, Vladislav, Maxime, Ayman, Yulin, Victor, Quentin, ChatGPT, Benjamin, Lucien Bachelard, Monsieur ABDELAZIM Abdelrahman, Monsieur DOUZE Yann et l'incroyablissime Monsieur VIATEUR.

Remerciement

Mes remerciements vont évidemment tout d'abord à mes amis, Daniel FERREiRA LARA et Ayoub LADJiCi, qui ont su m'encourager et m'épauler tout au long de ce semestre, et ce malgré la charge de travail de plus en plus insoutenable. Même lorsque les deadlines s'accumulaient et que le stress devenait difficile à gérer, ils n'ont jamais hésité à me tendre la main, à partager leurs connaissances et à m'apporter leur soutien moral. Leur présence constante, leur patience et leur solidarité m'ont permis de traverser cette période avec plus de sérénité, et je leur en serai toujours reconnaissant.

Mes remerciements vont aussi à Monsieur ABDELAZIM et à Monsieur DOUZE qui ont toujours fait preuve d'une grande pertinence dans leurs réponses même lorsque mes questions étaient des plus basiques. Merci aussi à notre très bien aimé Thibault (alias Papa des Ei) qui a beaucoup participé à mon épanouissement personnel et qui m'a appris à mieux structurer mes codes.

Pour finir, j'aimerais remercier toutes ces personnes extraordinaires que j'ai rencontrées cette année et la précédente dans la formation de mes rêves depuis de nombreuses années, et qui, par leur simple présence, me rendaient heureux : les copains du TPE, Yulin, Daniel, Inessa, Xiaochen, Jana, Joakim, Zimeng, Nicolas, Maxime, Ayman, Victor, Quentin, Papa, Ilias, Benjamin, Nadir, et ceux du TPA, nos incroyables alternants avec une mention toute particulière pour ces perles inoubliables qui m'ont accompagné tout au long de l'année dernière dans tous nos différents travaux en traitement du signal : Liza, Jihen, Amel, Salma, Farah et Ioana, les alternants de mon groupe, Asdjad, Fatou, Enes, Hakan, Vedat, Amine, les deux Tarek, Michel, Emmanuel, et les alternants qu'on ne pouvait voir que trop rarement grâce aux restructurations mais qui ont tout de même su rayonner : Arthur, Léa, Gloire A Dieu, Wassim, Ali, Elsa, Karlitou, Sékouba, Tom et pour finir Mon Ptit Chou.

Introduction

Dans le cadre de notre parcours universitaire en 4ème année d'école, nous sommes amenés, en Électronique Numérique 4, à réaliser et simuler un cœur de processeur mono-cycle. Pour une meilleure gestion de notre projet, nous avons, sur les conseils de Thibault, codé dans un environnement de travail pertinent avec l'IDE VScode, un suivi avec [GitHub](#) et une remise en ligne via [Google Doc](#).

Ce projet ambitieux implique la conception et la simulation de différents blocs principaux tels que l'unité de traitement, l'unité de gestion des instructions et l'unité de contrôle. L'objectif est de diviser le projet en petits composants tels que des multiplexeurs, des registres, des bancs de mémoire, ainsi qu'une Unité Arithmétique et Logique (UAL). Cela nous permettra de simuler, d'assembler et de tester notre modèle sur une carte FPGA.

Table des matières

Remerciement.....	2
Introduction.....	2
1. Télémètre ultrason.....	5
1.1. Implémentation en VHDL du télémètre seul.....	5
Figure 1.1.1 : Chronogramme du télémètre à ultrason.....	6
1.2. Essai sur carte FPGA du télémètre seul.....	6
Figure 1.2.1 : Test hardware du télémètre pour d = 10 cm.....	7
Figure 1.2.2 : Test hardware du télémètre pour d = 4 cm.....	7
Figure 1.2.3 : Test hardware du télémètre pour d = 6 cm.....	7
1.3. Implémentation en VHDL du télémètre compatible avec l'interface Avalon.....	8
Figure 1.3.1 : Chronogramme du télémètre à ultrason pour l'interface Avalon.....	8
1.4. Programmation logicielle sur carte FPGA du télémètre.....	9
Figure 1.4.1 : Console Nios ii affichant les mesures post traitées du télémètre.....	10
Figure 1.4.2 : Test hardware du télémètre avec affichage pour d = 10 cm.....	10
Figure 1.4.3 : Test hardware du télémètre avec affichage pour d = 9 cm.....	11
Figure 1.4.4 : Test hardware du télémètre avec affichage pour d = 5 cm.....	11
Figure 1.4.4 : Test hardware du télémètre avec affichage pour d = 0 cm.....	12
2. Servomoteur.....	13
2.1. Implémentation en VHDL du servomoteur seul.....	13
Figure 2.1.1 : Chronogramme du servomoteur seul.....	13
2.2. Test unitaire sur carte FPGA du servomoteur seul.....	14
Figure 2.2.1.a : Test hardware du servomoteur pour un angle de 90,0°	15
Figure 2.2.1.b : Mesure d'une impulsion haute de 2 ms sur l'oscilloscope.....	15
Figure 2.2.2.a : Test hardware du servomoteur pour un angle de 45,0°	16
Figure 2.2.2.b : Mesure d'une impulsion haute de 1,5 ms sur l'oscilloscope.....	16
Figure 2.2.3.a : Test hardware du servomoteur pour un angle de 0,0°	17
Figure 2.2.3.b : Mesure d'une impulsion haute de 1 ms sur l'oscilloscope.....	17
Figure 2.2.4.a : Test hardware du servomoteur pour un angle de 22,5°	18
Figure 2.2.4.b : Mesure d'une impulsion haute de 1,25 ms sur l'oscilloscope.....	18
2.3. Implémentation en VHDL du servomoteur compatible avec l'interface Avalon.....	19
Figure 2.3.1 : Chronogramme du servomoteur compatible avec l'interface Avalon.....	19
Figure 2.3.2 : Estimation de l'impulsion haute maximale de 2,2 ms sur l'oscilloscope.....	20
Figure 2.3.3 : Chronogramme du servomoteur compatible avec l'interface Avalon avec une plage de rotation étendue.....	21
2.4. Programmation logicielle sur carte FPGA du servomoteur.....	22
Figure 2.4.1 : Test hardware du servomoteur avec toutes les switchs éteintes pour un angle de 0° et mesure d'une impulsion haute de 0,5 ms sur l'oscilloscope.....	22
Figure 2.4.2 : Test hardware du servomoteur avec les switchs en position 450 pour un angle de 90° et mesure d'une impulsion haute de 1,35 ms sur l'oscilloscope.....	23
Figure 2.4.3 : Test hardware du servomoteur avec les switchs en position 675 pour un angle de 90° et mesure d'une impulsion haute de 1,75 ms sur l'oscilloscope.....	23
Figure 2.4.4 : Console Nios ii affichant les angles commandés.....	24

3. Télémètre tournant : Mesure angulaire distance aux obstacles.....	25
Figure 3.1 : Mesure de distances pour des angles de 62° et 81°.....	25
Figure 3.2 : Affichage des angles et distances mesurés dans la console Nios ii.....	26
4. LEDs Neopixel : intégration esthétique.....	27
4.1. Implémentation en VHDL du module Neopixel seul.....	27
Figure 4.1.1 : Chronogramme du module Neopixel seul.....	27
Figure 4.1.2 : Chronogramme du module Neopixel zoomé pour la vérification des durées T1H, T1L, T0H, T0L.....	28
4.2. Test unitaire sur carte FPGA des LEDs Neopixel.....	29
Figure 4.2.1 : Mesure des impulsions T1H, T1L, T0H, T0L sur l'oscilloscope.....	29
Figure 4.2.2 : Mesure de la durée du reset à l'état bas sur l'oscilloscope.....	30
Figure 4.2.3 : Lectures des envoies pour les couleurs simples cyan, jaune, et blanc pour les trois premiers LEDs sur l'oscilloscope.....	30
Figure 4.2.4 : Test hardware avec les couleurs complexes bleu nuage, orange pêche, et rose bonbon.....	31
Figure 4.2.5 : Test hardware des LEDs neopixel tel que les switchs sont en position 7..32	32
Figure 4.2.6 : Test hardware des LEDs neopixel tel que les switchs sont en position 1023	
33	
4.3. Implémentation en VHDL du module Neopixel compatible avec l'interface Avalon.....	33
Figure 4.3.1 : Chronogramme du module neopixel compatible avec l'interface Avalon..	33
4.4. Programmation logicielle sur carte FPGA du module Neopixel.....	35
Figure 4.4.1 : Test hardware du module neopixel pour une consigne de N = 1 LED.....	35
Figure 4.4.2 : Test hardware du module neopixel pour une consigne de N = 4 LEDs.....	36
Figure 4.4.3 : Test hardware du module neopixel pour une consigne de N = 8 LEDs.....	36
Figure 4.4.4 : Test hardware du module neopixel pour une consigne de N = 12 LEDs...	37
Figure 4.4.5 : Test hardware du module neopixel pour une consigne de N = 0 LED.....	37
4.5. Intégration des LEDs Neopixel au radar.....	38
Conclusion.....	40

1. Télémètre ultrason

Le télémètre ultrason est un dispositif qui permet de mesurer une distance en utilisant la réflexion d'ondes sonores. Cet outil est largement utilisé dans des applications telles que la robotique, la cartographie et l'évitement d'obstacles. Dans notre projet, il joue un rôle crucial pour cartographier une scène en mesurant les distances entre le capteur et les obstacles dans son champ de détection.

1.1. Implémentation en VHDL du télémètre seul

Le capteur utilisé, le HC-SR04, fonctionne en envoyant une onde ultrasonore et en mesurant le temps que cette onde met à revenir après avoir été réfléchie par un obstacle. Ce temps est directement proportionnel à la distance entre le capteur et l'obstacle.

Pour déclencher une mesure, un signal Trigger de 10 µs est envoyé au capteur. Ce dernier génère une série d'ondes ultrasonores et active un signal Echo, dont la durée reflète le temps aller-retour de l'onde. La distance est ensuite calculée à l'aide de la formule :

$$d = v_{son_{340}} \times t_{demi} = \frac{340}{2} t_{in} = 170t_{in}$$

où $v_{son_{340}}$ la vitesse de propagation du son (340 m.s-1) et t_{demi} la durée mesurée, divisée par deux pour prendre en compte l'aller-retour.

Dans notre cas, avec une fréquence d'horloge de 50 MHz, chaque cycle correspond à une durée de 20 ns, et une impulsion de 1 µs représente 500 cycles. Ainsi, un centimètre de distance équivaut à environ 2 941 cycles d'horloge, soit un temps de 0,00005882s. Pour simplifier nos calculs, nous avons approximé ce facteur en arrondissant à 3 000 cycles (200 × 15 coups d'horloge) par centimètre soit un temps de $60 \times 1\mu s$, introduisant une erreur minime mais acceptable pour nos besoins.

Pour l'implémentation, nous avons adapté cette formule dans notre code VHDL afin que la mesure de la durée d'Echo soit convertie en distance. Nous avons utilisé un compteur principal pour accumuler les cycles lorsque le signal Echo est actif. Cette durée est ensuite divisée par le facteur vu précédemment de 60 pour obtenir une estimation de la distance en centimètres.

Figure 1.1.1 : Chronogramme du télémètre à ultrason.



Pendant les simulations sur Modelsim, cette approximation a été testée avec plusieurs valeurs d'impulsions d'Echo. Par exemple, une impulsion de 2 ms a donné une mesure de 3 cm, correspondant à une distance calculée de 3,39 cm, tandis qu'une impulsion de 5 ms a produit 8 cm, proche de 8,47 cm théoriques. Enfin, une impulsion plus longue de 17 ms a donné 28 cm, avec une valeur réelle de 28,81cm. Ces résultats, bien qu'ils montrent une petite erreur de l'ordre d'un centimètre, confirment la validité de notre approche.

La simulation a d'abord permis d'identifier une anomalie dans le système, où la distance affichée ne se mettait à jour qu'après la réception d'un nouvel écho. Toutefois, cette logique a été revue sur les conseils de Monsieur DOUZE, et il a été décidé de conserver la dernière valeur valide tant qu'une nouvelle mesure n'est pas confirmée. Cette approche vise à éviter que des valeurs erronées ne soient affichées en cas de prélèvements trop rapides, ce qui pourrait entraîner une estimation incorrecte de la distance, comme une valeur nulle. Une telle situation pourrait laisser penser à un obstacle, alors que l'environnement pourrait en réalité être dégagé. Nous avons ainsi modifié le code pour que la valeur affichée reste stable jusqu'à la réception d'un nouvel écho confirmé.

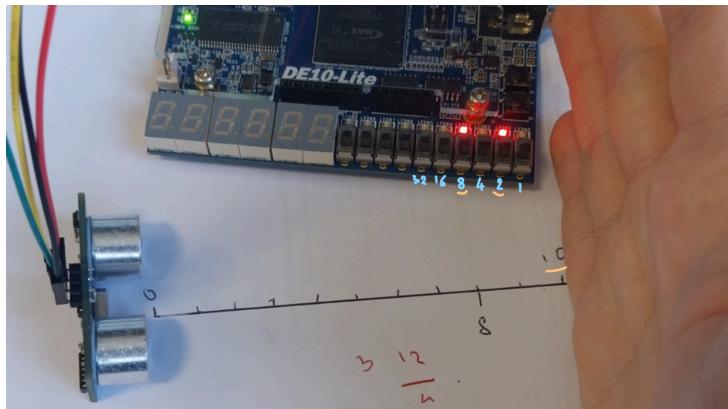
Enfin, un test du comportement du système en cas de reset asynchrone a été effectué, confirmant que le reset interrompt correctement les opérations en cours et réinitialise tous les signaux à leur état initial.

1.2. Essai sur carte FPGA du télémètre seul

Dans cette étape, nous avons réalisé des tests matériels en mode standalone pour vérifier le bon fonctionnement de l'implémentation VHDL du télémètre. L'objectif principal était de valider que le modèle proposé répond aux attentes en termes de précision et de cohérence des mesures. Ces essais ont également permis de corriger une erreur dans le coefficient utilisé pour convertir les cycles d'horloge en distance. Une erreur de facteur 10 avait initialement été introduite dans les calculs. Grâce à un protocole expérimental itératif, nous avons ajusté ce coefficient et obtenu des résultats précis et fiables.

Le protocole consistait à connecter le télémètre HC-SR04 aux broches GPIO de la carte FPGA. La sortie, correspondant à la distance mesurée, a été connectée aux LEDs rouges (LEDR[9..0]) de la carte DE10-Lite, permettant une visualisation directe des résultats.

Figure 1.2.1 : Test hardware du télémètre pour $d = 10 \text{ cm}$



Lors du premier test, nous avons placé un obstacle à une distance de 10 cm du capteur. Les LEDs indiquaient bien le bit de données 8 + 2. Ce résultat a permis de vérifier la juste proportionnalité entre la durée du signal écho et la distance réelle, et de valider les calculs de conversion pour cette plage de mesure.

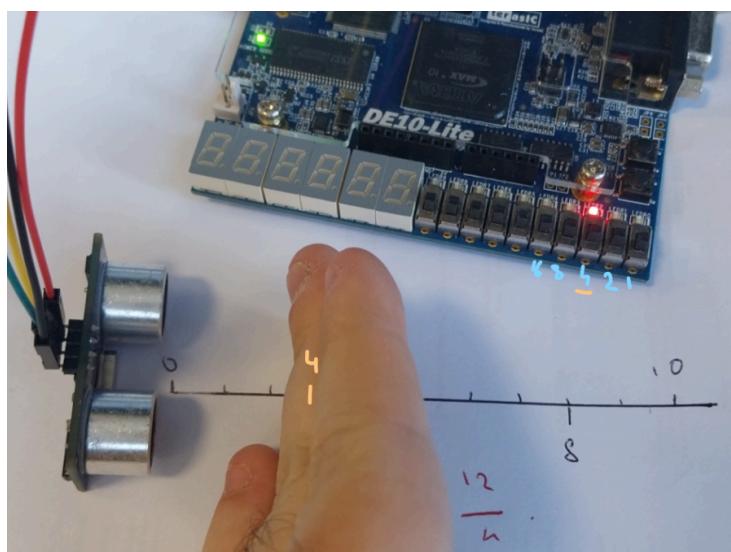
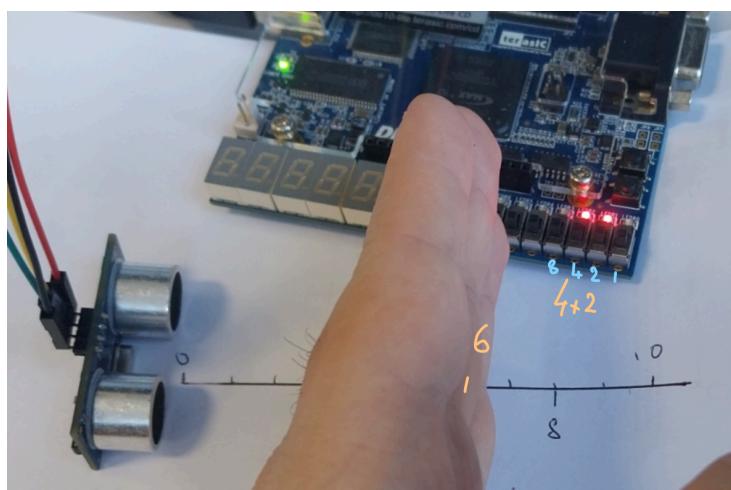


Figure 1.2.2 : Test hardware du télémètre pour $d = 4 \text{ cm}$

Un second test a été mené avec un obstacle à 4 cm. Cette distance plus courte visait à évaluer la précision du système dans une plage très rapprochée. Les résultats étaient conformes, et aucune anomalie n'a été relevée, confirmant la robustesse du modèle même pour des distances proches.

Figure 1.2.3 : Test hardware du télémètre pour $d = 6 \text{ cm}$

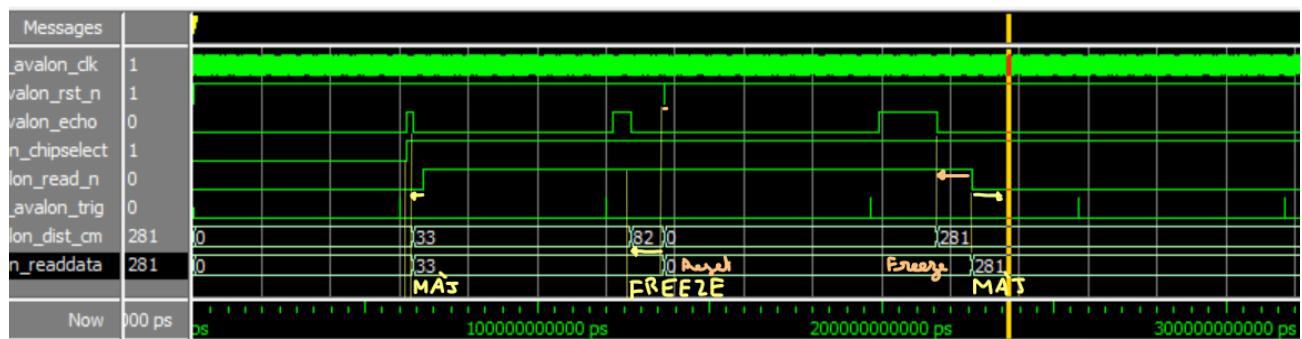


Enfin, une variation de 2 cm a été introduite, en déplaçant cette fois-ci l'obstacle à 6 cm. Ce test visait à vérifier la capacité du système à détecter des variations subtiles de distance. Le télémètre a mesuré la distance correctement, avec une précision suffisante pour distinguer clairement ce petit écart de 2cm.

1.3. Implémentation en VHDL du télémètre compatible avec l'interface Avalon

Afin d'interconnecter le module télémètre au reste du système de manière standardisée, nous avons intégré l'interface Avalon - MM. Cette intégration garantit une compatibilité avec les autres périphériques et permet une gestion centralisée des données par le bus Avalon. Pour cela, nous avons ajouté les signaux spécifiques à Avalon comme chipselect pour l'activation du module, read_n pour la lecture (actif à l'état bas) et readdata pour transmettre les données mesurées.

Figure 1.3.1 : Chronogramme du télémètre à ultrason pour l'interface Avalon



Les tests effectués sur cette nouvelle architecture ont confirmé son bon fonctionnement. Nous avons observé que, dans un état initial, avant toute réception d'un signal Echo, la valeur de readdata restait nulle. Lorsqu'un signal Echo est détecté, la distance mesurée est correctement calculée et mise à jour. Cependant, si la lecture est désactivée (avec read_n à 1), la valeur de readdata reste figée, même si une nouvelle mesure est effectuée. Ce comportement garantit la cohérence des données sur le bus Avalon.

Un test particulier a été mené pour vérifier la gestion du reset. Lorsque le signal de reset est activé, tous les compteurs internes et les données sont correctement réinitialisés à zéro, assurant ainsi une remise à l'état initial du système. En outre, après une série d'impulsions d'Echo, nous avons confirmé que la distance était calculée et que les données étaient transmises sur readdata uniquement lorsque read_n repassait à l'état bas.

Ces résultats montrent que l'implémentation Avalon fonctionne comme prévu et permet une intégration fluide du télémètre dans un système SoC.

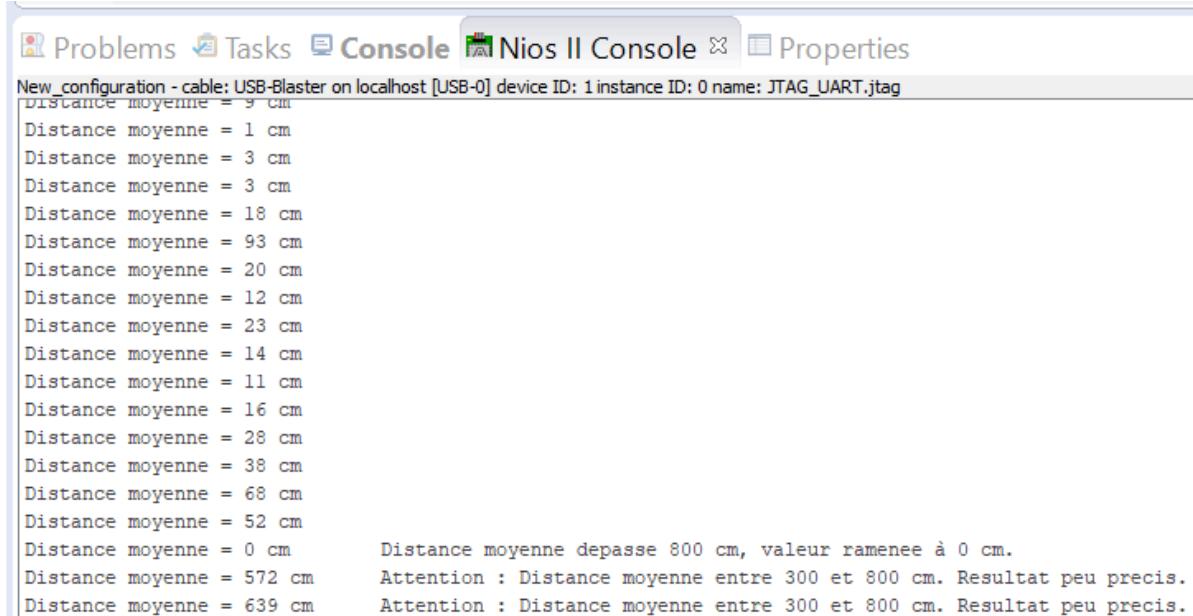
1.4. Programmation logicielle sur carte FPGA du télémètre

La mise en œuvre logicielle du télémètre sur une carte FPGA a nécessité plusieurs étapes, allant de la capture des mesures brutes à leur traitement et leur affichage sur un écran sept segments. Le développement a également inclus des mécanismes de filtrage pour traiter les anomalies et les fluctuations des mesures dues à des contraintes matérielles ou des limitations intrinsèques du télémètre.

Nous avons dans un premier temps mis en place un affichage sur six afficheurs sept segments pour visualiser directement, sur la carte FPGA, les distances mesurées par le télémètre. Ce choix visait à permettre une interprétation rapide des données sans dépendre d'un affichage externe. Cependant, il a été constaté que par moments certaines situations généraient des valeurs aberrantes, par exemple des distances de l'ordre de la dizaine de mètres lorsque le télémètre était un petit peu incliné ou instable. Ces aberrations étaient souvent causées par de légères vibrations ou des inclinaisons imprévues du module. Pour améliorer la robustesse, un mécanisme de filtrage a été ajouté : les mesures dépassant une certaine limite ont été exclues du calcul.

Par ailleurs, les distances mesurées présentaient des fluctuations importantes en raison de la nature instable du support du télémètre et de son environnement. Pour atténuer ces fluctuations, nous avons introduit un délai entre chaque mise à jour des données, pendant lequel une moyenne glissante corrigée est calculée. Cette moyenne prend en compte un nombre paramétrable de mesures, tout en rejetant les valeurs aberrantes. Enfin, un autre cas problématique a été identifié lorsque le télémètre était très proche d'un obstacle, par exemple lorsqu'il était collé à un mur. Dans ces conditions, les mesures étaient incohérentes, affichant des distances extrêmement grandes au lieu de zéro. Une correction spécifique a été intégrée pour gérer cette situation, afin de garantir que le système ne considère pas un obstacle proche comme étant loin, ce qui pourrait être critique dans un système embarqué.

Figure 1.4.1 : Console Nios ii affichant les mesures post traitées du télémètre



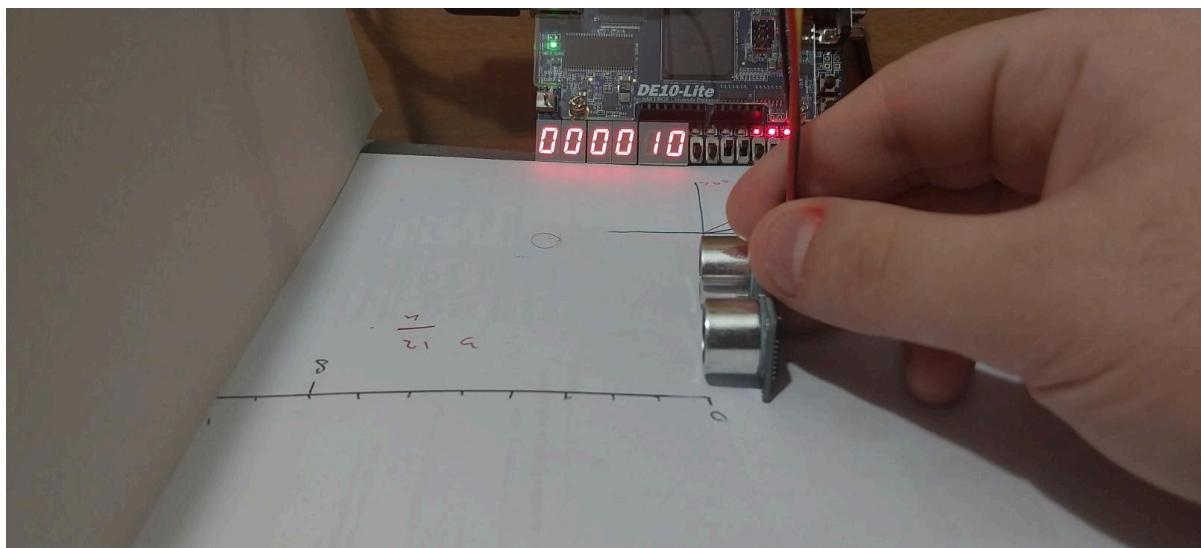
```

Problems Tasks Console Nios II Console Properties
New_configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: JTAG_UART.jtag
Distance moyenne = 9 cm
Distance moyenne = 1 cm
Distance moyenne = 3 cm
Distance moyenne = 3 cm
Distance moyenne = 18 cm
Distance moyenne = 93 cm
Distance moyenne = 20 cm
Distance moyenne = 12 cm
Distance moyenne = 23 cm
Distance moyenne = 14 cm
Distance moyenne = 11 cm
Distance moyenne = 16 cm
Distance moyenne = 28 cm
Distance moyenne = 38 cm
Distance moyenne = 68 cm
Distance moyenne = 52 cm
Distance moyenne = 0 cm
Distance moyenne = 572 cm
Distance moyenne = 639 cm
Distance moyenne depasse 800 cm, valeur ramenee à 0 cm.
Attention : Distance moyenne entre 300 et 800 cm. Resultat peu precis.
Attention : Distance moyenne entre 300 et 800 cm. Resultat peu precis.

```

Nous avons initialement testé le télémètre en interne, via des logs générés par la console Nios II. Ce test préliminaire nous a permis d'observer les données brutes de manière rapide et d'identifier les premières anomalies dans les prélèvements de mesures. Les ajustements et corrections de notre modèle ont ainsi pu être réalisés avant de passer à l'affichage matériel sur la carte FPGA. La [Figure 1.4.1](#) montre un exemple des mesures post-traitées affichées dans la console Nios II. Ces tests incluent des mesures à courte distance, allant jusqu'à un mètre, et des mesures à des distances maximales dans une salle pour vérifier les messages d'avertissement. Nous avons également simulé des cas où le télémètre était trop proche d'un mur pour observer la bonne gestion de cette incohérence.

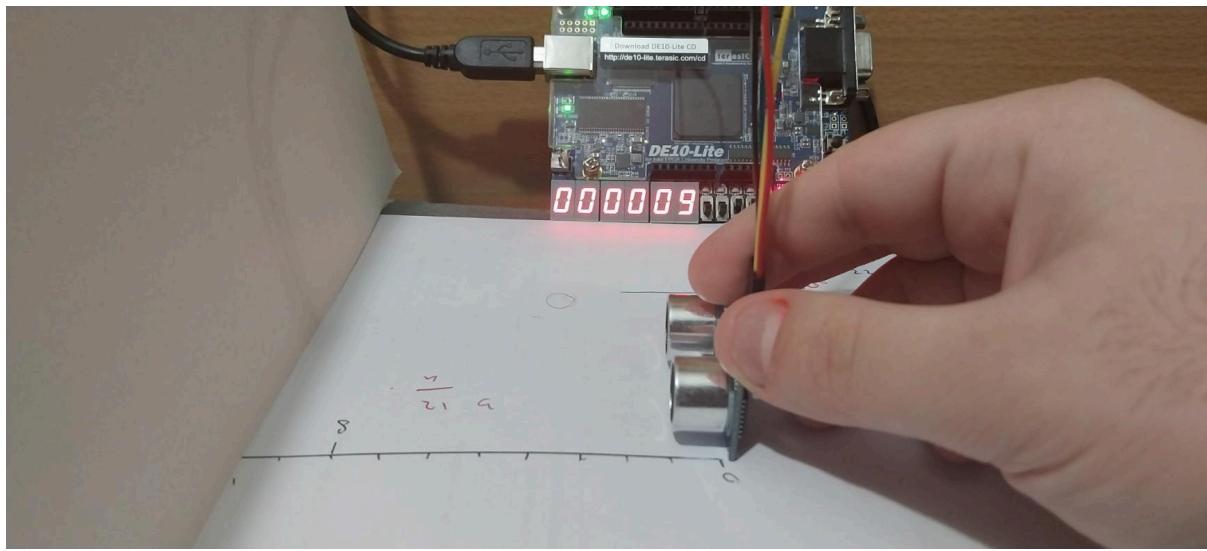
Figure 1.4.2 : Test hardware du télémètre avec affichage pour $d = 10 \text{ cm}$



Après validation des traitements dans la console, nous avons testé le système en affichant les résultats directement sur les afficheurs sept segments de la carte FPGA. Ce mode permet une interprétation visuelle immédiate, essentielle pour des applications en conditions réelles.

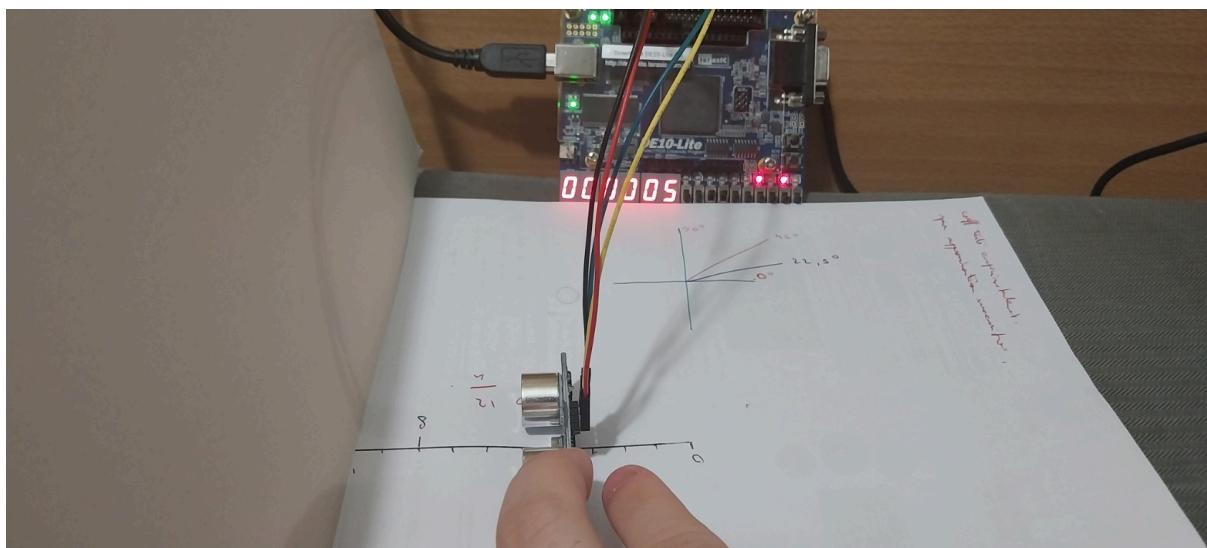
Sur ce test nous observons le bon fonctionnement de l'affichage pour une distance de 10 cm, une valeur arbitraire choisie pour s'assurer que les résultats sont correctement affichés.

Figure 1.4.3 : Test hardware du télémètre avec affichage pour $d = 9 \text{ cm}$



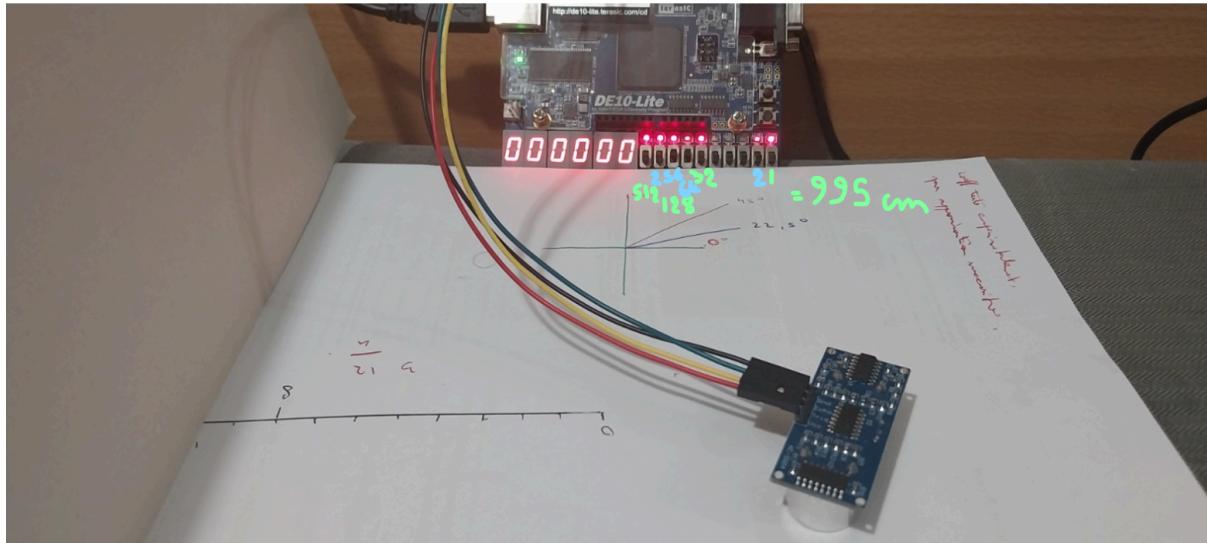
Ensuite, nous avons réalisé un test à 9 cm pour observer si le système détecte et affiche des variations faibles de distance. La précision et la stabilité ont été validées pour ces mesures proches.

Figure 1.4.4 : Test hardware du télémètre avec affichage pour $d = 5 \text{ cm}$



Ce test supplémentaire, pour une distance de 5 cm, montre que le traitement logiciel est capable de gérer avec fiabilité les courtes distances mesurées.

Figure 1.4.4 : Test hardware du télémètre avec affichage pour $d = 0 \text{ cm}$



Enfin, un test particulier a été réalisé en dirigeant le télémètre vers le sol. Bien que les LED connectées au bus de sortie 10 bits affichent une distance brute de 9,95 m (valeur erronée), le traitement intégré fonctionne correctement, ramenant l'affichage à 0 cm sur les afficheurs sept segments. Ce comportement prouve l'efficacité du filtrage et de la gestion des incohérences.

En résumé, cette partie a permis de développer une chaîne de traitement robuste pour le télémètre, intégrée sur la carte FPGA. Les différentes étapes, depuis la capture des mesures jusqu'à leur affichage, incluent des mécanismes de filtrage et de correction qui garantissent une fiabilité accrue même dans des conditions difficiles. Avec ces ajustements, le télémètre est prêt à être utilisé dans des systèmes embarqués nécessitant des mesures précises et cohérentes.

2. Servomoteur

Dans cette deuxième partie consacrée au servomoteur, nous avons choisi de convertir les positions angulaires du servomoteur en dixièmes de degré, soit sur 10 bits d'entrée, afin d'assurer une meilleure précision et d'anticiper un affichage plus détaillé des mesures sur l'écran VGA. Les valeurs supérieures à 900 dixièmes de degré (soit 90°) sont volontairement ramenées à cette limite pour respecter les contraintes physiques du servomoteur.

2.1. Implémentation en VHDL du servomoteur seul

Le fonctionnement du servomoteur repose sur un compteur incrémental dans une période de 20 ms, correspondant à 1 000 000 cycles d'horloge. Les valeurs de position transmises, exprimées en dixièmes de degré, déterminent la durée de l'impulsion haute (entre 1 ms pour 0° et 2 ms pour 90°). Par exemple, une position intermédiaire de 45° correspond à une impulsion de 1,5 ms, tandis qu'une précision au dixième de degré est assurée grâce à un incrément minimal de 55 cycles d'horloge par 0,1°. Par ailleurs, les états angulaires sont déterminés en fonction de la valeur du compteur régulant les impulsions.

Figure 2.1.1 : Chronogramme du servomoteur seul.



Afin de valider ce modèle, nous avons élaboré un banc de test en simulation. La première partie des tests concerne les positions limites et demi du servomoteur : 0°, 45° et 90°.

Nous avons observé des impulsions respectives de 1 ms, 1,5 ms et 2 ms sur le chronogramme, en parfaite cohérence avec les calculs théoriques.

Ensuite, nous avons testé la gestion d'une valeur angulaire intermédiaire, comme $32,4^\circ$, dont l'impulsion correspond à 1,356 ms. Cette valeur a été vérifiée avec succès, notamment par l'absence de levée des drapeaux d'avertissement en simulation, le tout avec une précision au microseconde.

Enfin, nous avons réalisé des tests sur des scénarios particuliers. Un test avec le signal `reset_n` a validé la remise à zéro de l'impulsion de commande lorsque le reset est activé. De même, les positions supérieures à 900 dixièmes de degré ont été correctement ramenées à une impulsion de 2 ms, confirmant la robustesse de notre implémentation.

2.2. Test unitaire sur carte FPGA du servomoteur seul

Pour valider le modèle matériel du servomoteur, des tests unitaires ont été réalisés sur la carte FPGA DE10-Lite. Ces tests avaient pour objectif de vérifier que les commandes générées respectaient les durées d'impulsion théoriques attendues pour des positions angulaires données. Afin de simuler ces positions, des interrupteurs (`switch`) ont été utilisés pour paramétriser la commande. Bien que des simulations fonctionnelles aient été effectuées préalablement sur ModelSim, une validation sur carte s'est avérée indispensable, non seulement pour vérifier la conformité du signal avec un oscilloscope, mais aussi pour évaluer la réponse réelle du matériel.

Lors des tests initiaux, le servomoteur n'a pas réagi, malgré un code vérifié par simulation. Ce comportement inattendu illustre l'importance de cette phase de vérification : si nous avions directement intégré l'IP dans un projet complet sans tester le matériel en solo, nous aurions pu attribuer l'échec à une erreur dans l'ensemble du système Quartus.

Une analyse approfondie a révélé que le servomoteur initialement utilisé était défectueux. En comparant les signaux mesurés par l'oscilloscope à la théorie, nous avons confirmé que les durées d'impulsion générées par la carte FPGA étaient correctes. Cela nous a permis d'identifier et de remplacer le servomoteur par un modèle fonctionnel, économisant ainsi un temps précieux.

Pour chaque position angulaire définie par les interrupteurs, les signaux de commande ont été observés à l'oscilloscope à l'aide d'un Analog Discovery 2. Cette étape est essentielle pour s'assurer que les périodes et les largeurs d'impulsion respectent les spécifications du servomoteur, qui repose sur une modulation en largeur d'impulsion (PWM). Par exemple, pour un angle de 90° , une impulsion haute de 2 ms doit être générée avec une période totale de 20 ms. Ces mesures ont été effectuées pour plusieurs valeurs de commande, comme le montrent les figures ci-dessous.

Figure 2.2.1.a : Test hardware du servomoteur pour un angle de 90,0°

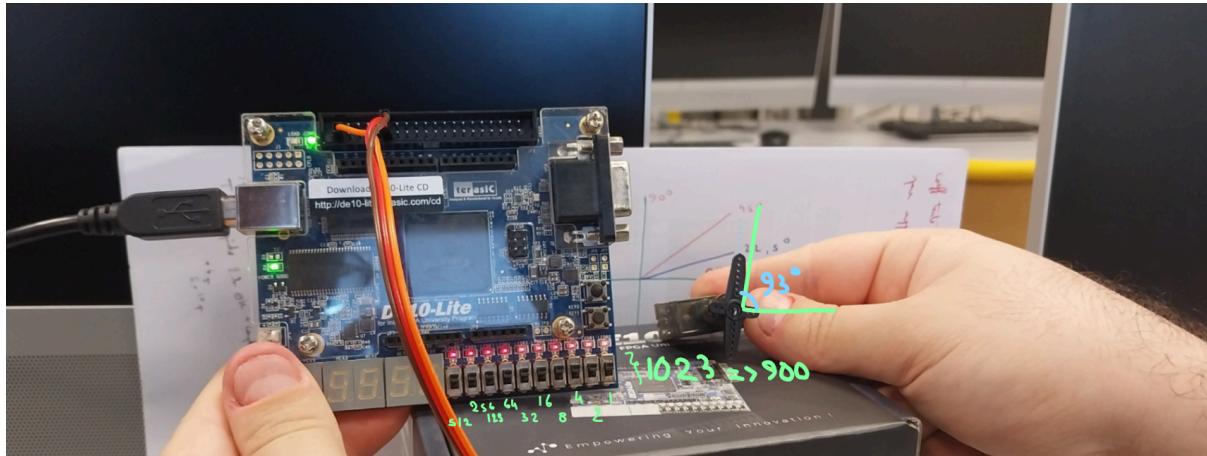
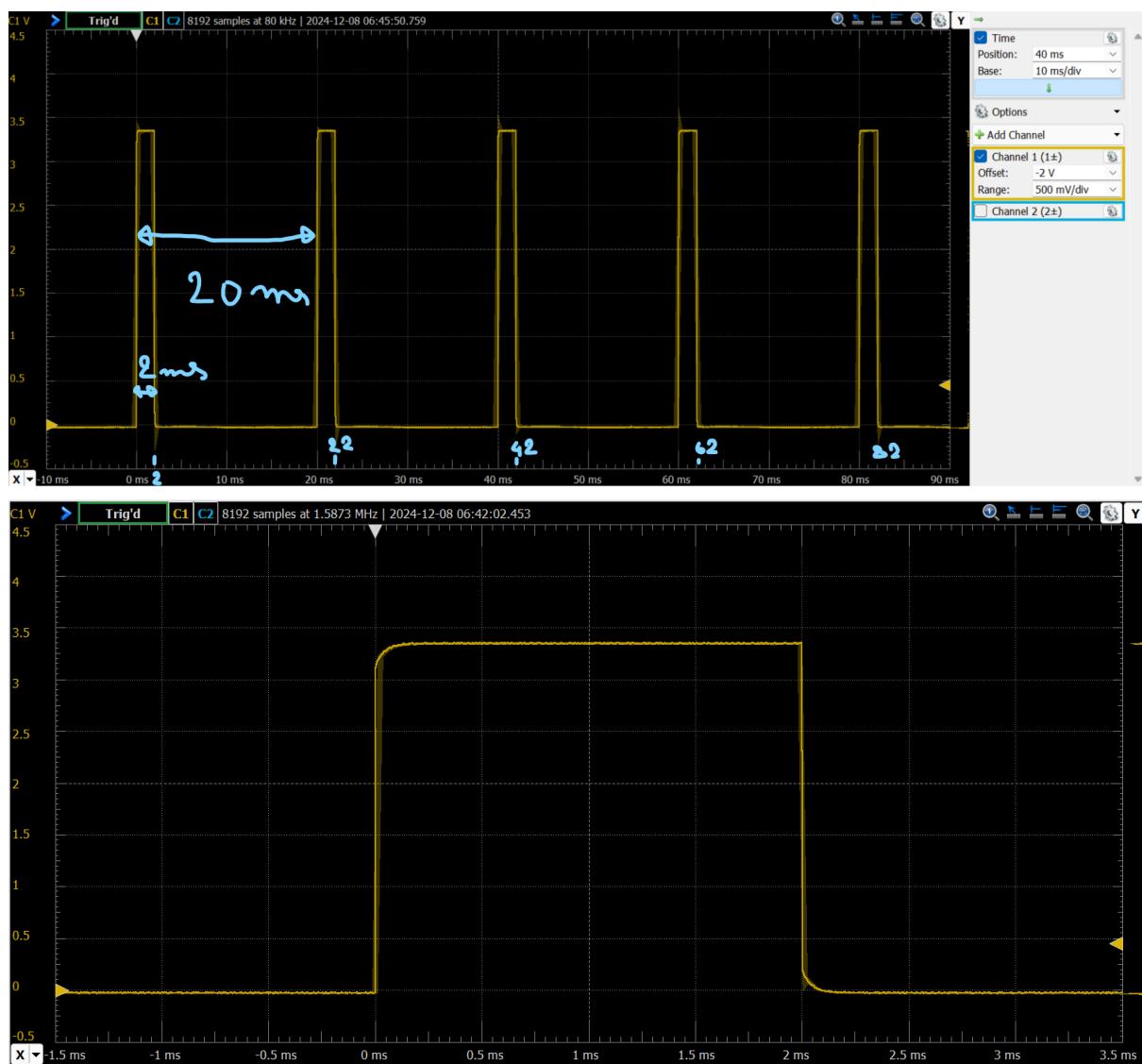


Figure 2.2.1.b : Mesure d'une impulsion haute de 2 ms sur l'oscilloscope



Lors du premier test, l'angle de 90° a été simulé en actionnant tous les interrupteurs de sorte à vérifier encore une fois que la consigne sera bien plafonnée pour transmettre une

commande correspondante à 90° . La largeur d'impulsion mesurée sur l'oscilloscope était bien de 2 ms, conformément aux attentes théoriques. Cependant, lors de l'utilisation du servomoteur, il a été observé que pour cette commande, l'angle effectif variait légèrement entre 95° et 92° en fonction des servomoteurs testés. Cette légère anomalie pourrait être corrigée par une adaptation du modèle, par exemple en réduisant la largeur d'impulsion maximale à une valeur légèrement inférieure (comme 1,95 ms), ou en ajustant globalement la relation entre les impulsions et les positions angulaires. Cependant, dans notre cas d'utilisation, ces écarts n'ont pas d'impact significatif sur les résultats attendus.

Figure 2.2.2.a : Test hardware du servomoteur pour un angle de $45,0^\circ$

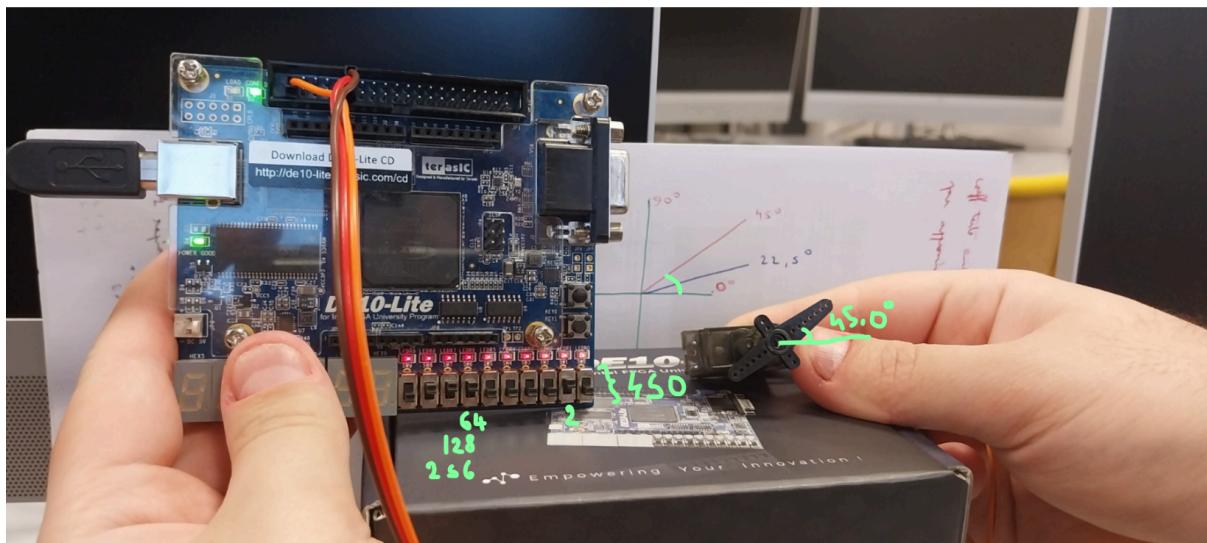
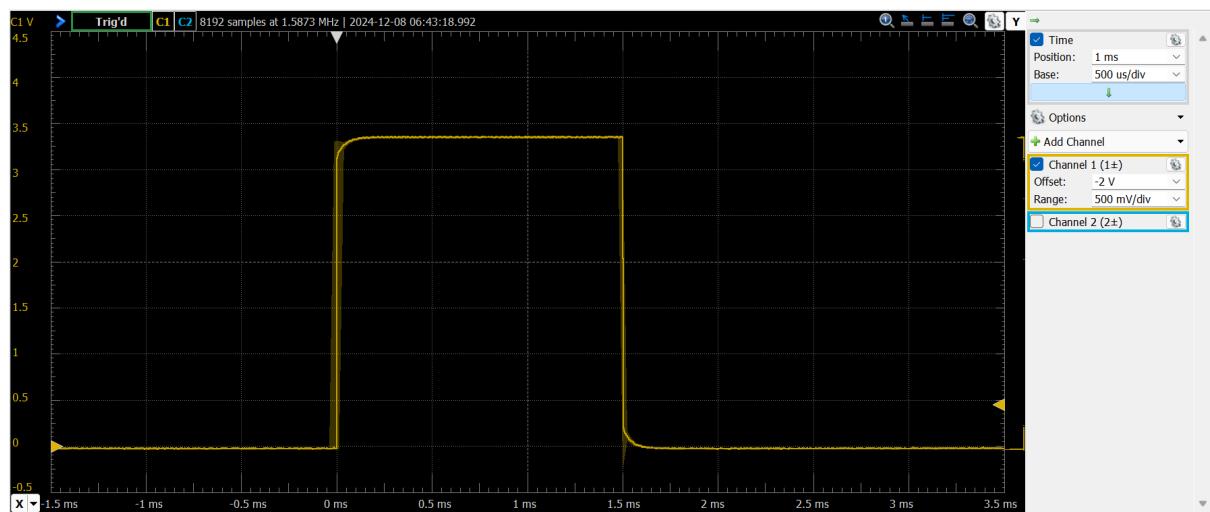


Figure 2.2.2.b : Mesure d'une impulsion haute de 1,5 ms sur l'oscilloscope



Pour un angle intermédiaire de 45° , une largeur d'impulsion de 1,5 ms a été mesurée. Ce test a permis de valider la gestion correcte des positions intermédiaires et de vérifier que le signal transmis par la carte FPGA correspondait aux spécifications.

Figure 2.2.3.a : Test hardware du servomoteur pour un angle de 0,0°

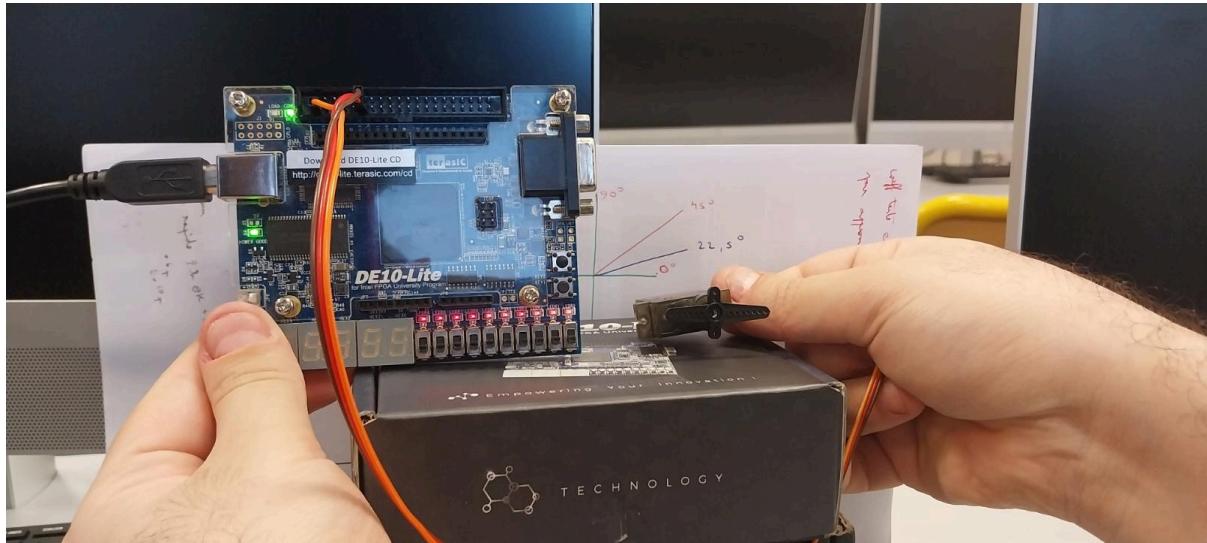
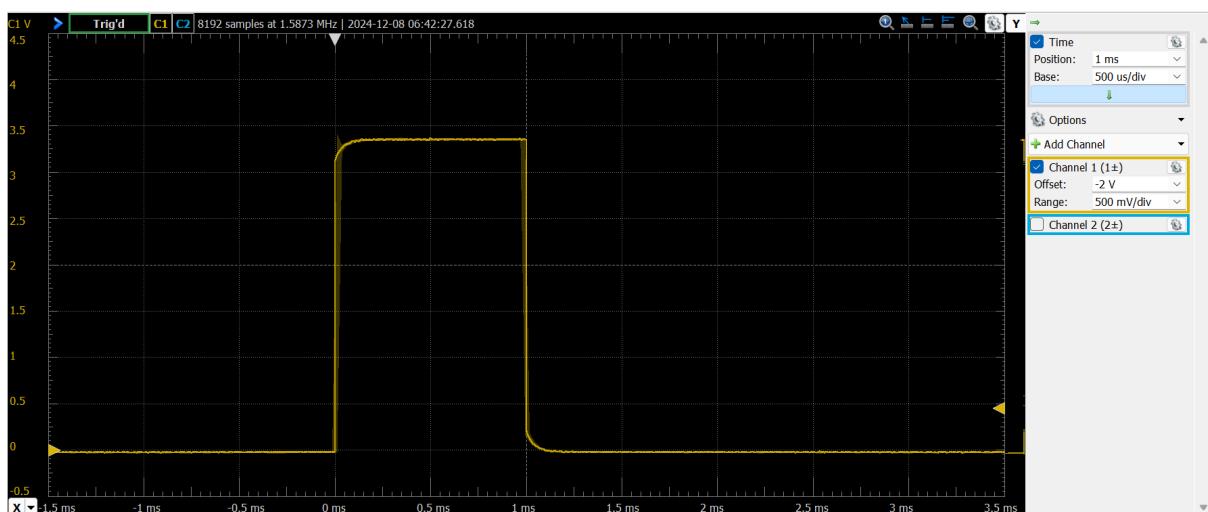


Figure 2.2.3.b : Mesure d'une impulsion haute de 1 ms sur l'oscilloscope



Un autre test a été effectué pour l'angle minimal de 0°. Comme attendu, une largeur d'impulsion de 1 ms a été relevée, confirmant le fonctionnement correct pour cette position limite.

Figure 2.2.4.a : Test hardware du servomoteur pour un angle de 22,5°

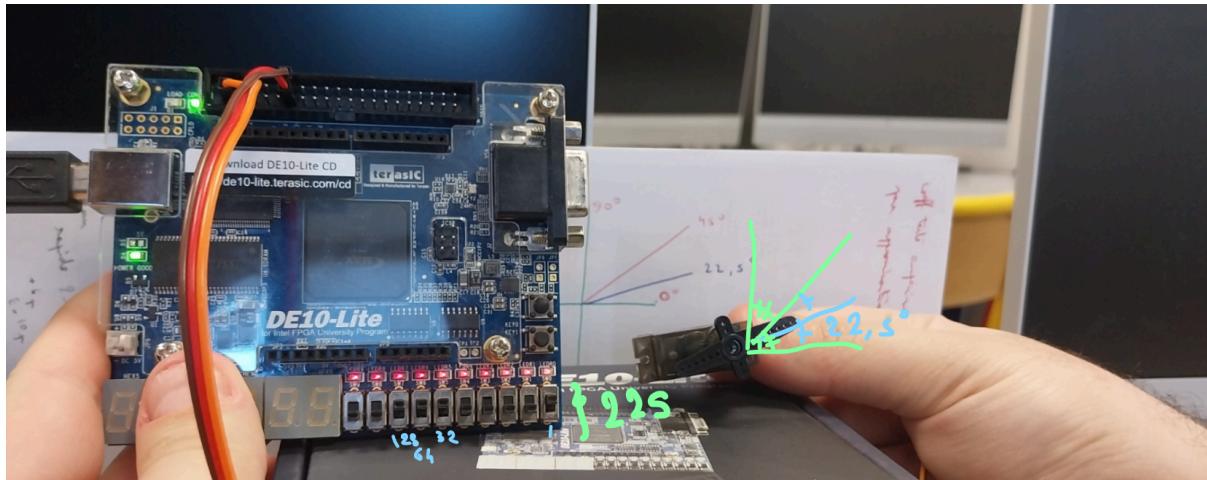
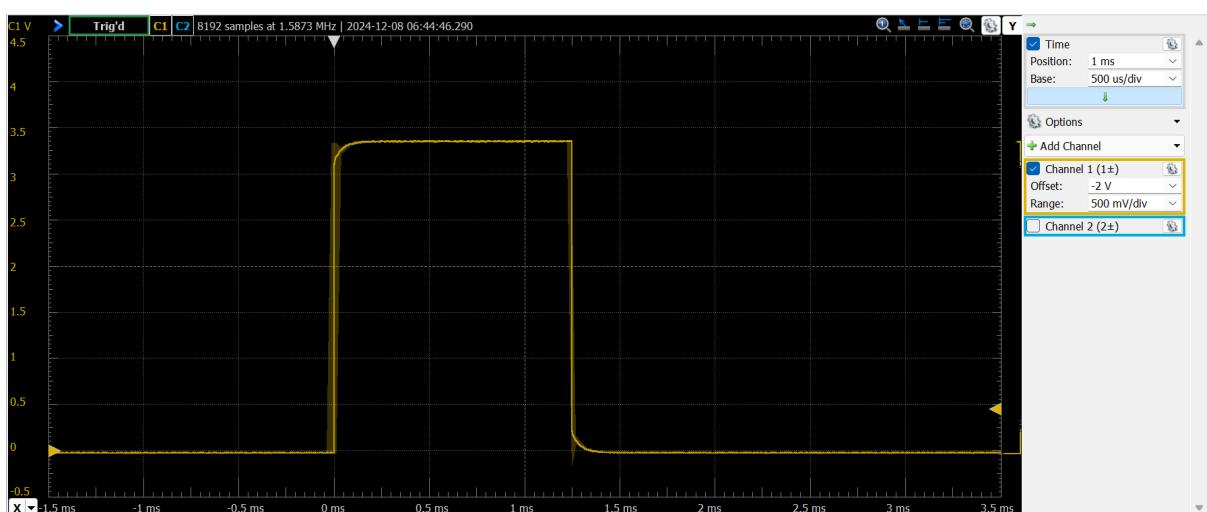


Figure 2.2.4.b : Mesure d'une impulsion haute de 1,25 ms sur l'oscilloscope



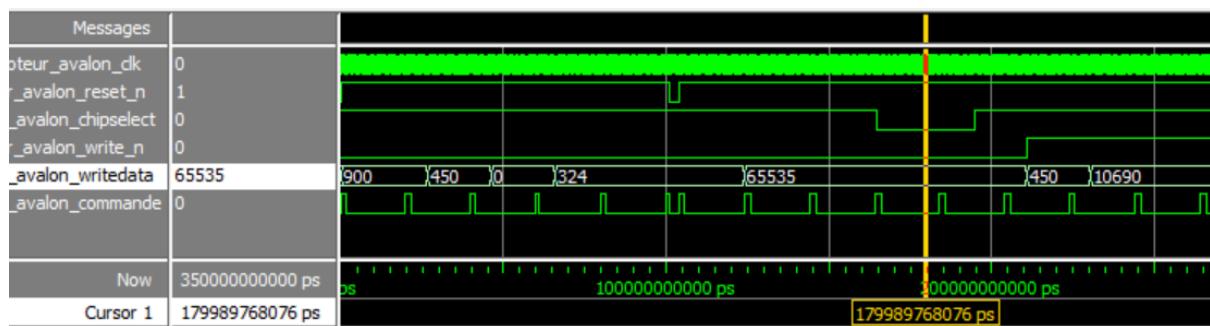
Enfin, un test de précision a été réalisé pour une position angulaire précise de 22,5°. L'oscilloscope a permis de mesurer une largeur d'impulsion de 1,25 ms, en accord avec les calculs théoriques. Ce test a démontré la capacité du système à générer des commandes précises, au dixième de degré près.

Les tests effectués ont confirmé que le système respecte les spécifications théoriques des signaux PWM pour le servomoteur. Bien que le code ait été validé par simulation, ces vérifications sur carte ont permis d'identifier un servomoteur défectueux avant son intégration dans le projet complet, illustrant l'importance de ces étapes de validation matérielle. Les durées d'impulsion mesurées pour chaque position angulaire sont conformes aux attentes, malgré quelques légers écarts dans les angles mesurés en pratique, dus aux caractéristiques propres des servomoteurs utilisés. Ces écarts pourraient être ajustés dans une version ultérieure du système, mais ils n'ont pas d'impact significatif dans notre cas d'application. Les tests valident donc la robustesse et la précision de l'IP, ouvrant la voie à son intégration avec le reste du projet.

2.3. Implémentation en VHDL du servomoteur compatible avec l'interface Avalon

Afin d'interconnecter le module servomoteur au reste du système de manière standardisée, nous avons intégré l'interface Avalon - MM. Cette intégration garantit une compatibilité avec les autres périphériques et permet une gestion centralisée des données par le bus Avalon. Pour cela, nous avons ajouté les signaux spécifiques à Avalon comme chipselect pour l'activation du module, write_n pour l'écriture (actif à l'état bas) et WriteData pour transmettre les données mesurées.

Figure 2.3.1 : Chronogramme du servomoteur compatible avec l'interface Avalon



Les premiers tests ont repris les vérifications initiales sur trois valeurs particulières, afin de confirmer la période correcte du signal de commande généré par le module. Les tests incluaient également la vérification du comportement du système après un reset, ainsi que la validation d'une valeur supérieure à 900, où la période observée est restée conforme à celle attendue de 2 ms. Tous ces tests ont donné des résultats satisfaisants.

Dans un second temps, des essais spécifiques aux nouvelles fonctionnalités liées à l'intégration Avalon ont été réalisés. Par exemple, il a été observé que lorsque le signal chipselect est désactivé (état 0), la commande continue d'être appliquée à la position précédente, et qu'à sa réactivation (état 1), la commande reprend avec la même période que précédemment. De plus, lorsque write_n passe à l'état 1, les modifications apportées à WriteData n'ont pas d'impact sur la sortie, qui reste inchangée. Ces comportements confirment le bon fonctionnement du module et la conformité de l'interface avec les spécifications Avalon.

Une erreur initiale a été identifiée dans la configuration de la fréquence d'horloge. Contrairement à la fréquence de 50 MHz utilisée sur le DE1-SoC, la fréquence actuelle était de 100 MHz, nécessitant l'introduction d'un facteur correctif dans la partie logicielle du télémètre Avalon. Ce correctif sera également appliqué au module servomoteur.

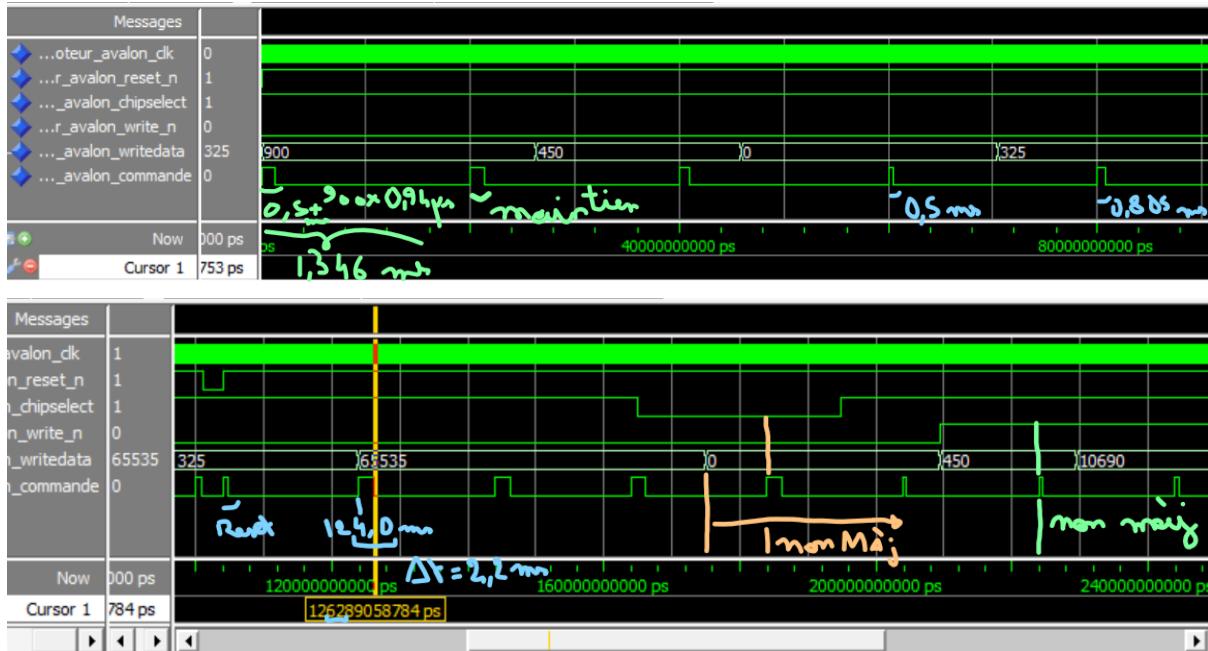
Par ailleurs, pour améliorer le champ de vision et fournir un panorama plus large des distances aux obstacles, la plage de rotation angulaire du servomoteur a été étendue. Initialement, une amplitude de 90° avait été retenue pour les tests, car cette plage bénéficie d'une harmonisation commune des durées d'impulsion entre différents modèles de servomoteurs dans l'intervalle de 1 ms à 2 ms. Cependant, il a été constaté que la plage standard associée à une amplitude de 180° (allant théoriquement de 0,5 ms à 2,5 ms) ne correspondait pas au modèle utilisé. Lors des tests d'étalonnage réalisés avec l'oscilloscope, il a été relevé que notre servomoteur acceptait une impulsion minimale de 0,5 ms, mais atteignait un plafond de 2,2 ms, et non 2,5 ms comme supposé initialement.

Figure 2.3.2 : Estimation de l'impulsion haute maximale de 2,2 ms sur l'oscilloscope



Ces nouvelles observations ont conduit à une adaptation de l'IP pour respecter les spécifications ajustées. Pour une variation angulaire de 180° , la variation temporelle correspondante est désormais de 1,7 ms (2,2 ms - 0,5 ms), soit un pas angulaire de 1° pour $9,44 \mu\text{s}$, ce qui correspond à 944 périodes d'horloge pour 100 MHz. Ainsi, pour atteindre une précision d'un dixième de degré, 94 périodes d'horloge sont nécessaires.

Figure 2.3.3 : Chronogramme du servomoteur compatible avec l'interface Avalon avec une plage de rotation étendue



Après correction, une impulsion de 1,346 ms correspond désormais à une position de 90°, au lieu de 1,5 ms précédemment. Cette valeur est maintenue tant que la consigne reste inchangée, et la période globale entre les impulsions demeure de 20 ms. De nouveaux tests ont été réalisés pour valider cette mise à jour, avec une légère adaptation du banc de test. Par exemple, une impulsion de 0,5 ms correspond désormais à une position angulaire de 0°, ce qui est cohérent avec le déplacement du point de départ pour élargir la plage angulaire. Pour une consigne fractionnaire correspondant à 0,805 ms, le système fonctionne correctement. Le reset continue de réinitialiser les sorties de façon asynchrone et redémarre l'impulsion à la valeur initiale. Les consignes dépassant la plage maximale sont plafonnées à 2,2 ms, correspondant à un angle de 180°, avec une marge de sécurité pour éviter d'endommager le servomoteur.

Enfin, les tests avec chipselect désactivé montrent que la dernière valeur reste appliquée, même en présence d'une consigne modifiée, jusqu'à la réactivation du chipselect. Le comportement est similaire avec write_n, où les modifications de l'angle sont ignorées lorsque WriteData est désactivé.

En conclusion, l'IP servomoteur a été corrigée et ajustée suite aux expérimentations, permettant de mieux comprendre les écarts constatés auparavant. Par exemple, en [Figure 2.2.1.a](#), l'écart de quelques degrés relevé était dû à une hypothèse erronée selon laquelle 1 ms complète représentait une amplitude de 90°. Avec le coefficient corrigé de 94 cycles par dixième de degré au lieu de 111, il a été démontré que la variation temporelle de 0,846 ms pour 90° explique ces écarts. En tenant compte de ces ajustements, la position angulaire maximale a été recalculée pour correspondre à une impulsion de 2,2 ms, avec une réserve de sécurité par rapport à la valeur théorique maximale estimée à 2,3 ms. Ces choix permettent de garantir la fiabilité et la robustesse du système tout en protégeant le matériel.

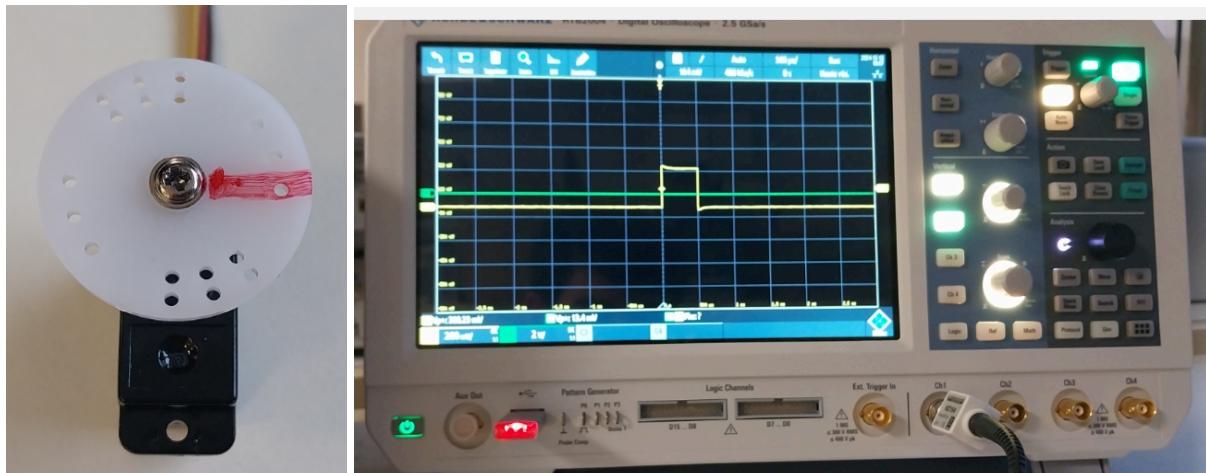
2.4. Programmation logicielle sur carte FPGA du servomoteur

Avec l'intégration de l'IP Servomoteur corrigée, l'étape suivante a consisté à programmer le processeur embarqué pour commander le servomoteur en position angulaire à partir des interrupteurs (switchs) disponibles sur la carte FPGA. Le programme logiciel développé a permis de générer les consignes nécessaires, en exploitant les valeurs des switchs comme entrée.

Une particularité de la conception mérite d'être soulignée : la carte FPGA dispose de seulement 10 switchs, permettant ainsi de coder des valeurs entières comprises entre 0 et 1023. Cependant, pour tirer pleinement parti de la plage angulaire étendue de 180° , avec une précision au dixième de degré, il aurait fallu pouvoir coder des valeurs jusqu'à 1800 soit 180,0°.

Pour pallier cette contrainte, nous avons introduit un décalage à gauche des bits des switchs dans la partie logicielle, équivalent à une multiplication par deux. Cela permet de doubler la résolution et d'obtenir une plage angulaire pratique allant jusqu'à 2047, dont seule la portion jusqu'à 1800 est utilisée dans notre système. Ce compromis, jugé acceptable pour l'expérimentation, garantit une précision fonctionnelle tout en s'accommodant des limites matérielles.

Figure 2.4.1 : Test hardware du servomoteur avec toutes les switchs éteintes pour un angle de 0° et mesure d'une impulsion haute de 0,5 ms sur l'oscilloscope



Dans ce cas, aucun switch n'est activé, correspondant à une consigne de 0°. L'oscilloscope montre une impulsion haute de 0,5 ms, confirmant la position angulaire de 0°.

Figure 2.4.2 : Test hardware du servomoteur avec les switchs en position 450 pour un angle de 90° et mesure d'une impulsion haute de 1,35 ms sur l'oscilloscope



En activant les switchs tel que “01_1100_0010” = $450 * 2 = 900$ pour commander un angle de 90,0°, une impulsion haute de 1,35 ms est observée sur l'oscilloscope. Cela correspond à la nouvelle configuration du système après correction, contrairement aux 1,5 ms initialement associées à un angle de 90° dans les tests théoriques et pratiques précédents. Cette vérification met en évidence l'adaptation réussie de l'IP à la plage angulaire élargie.

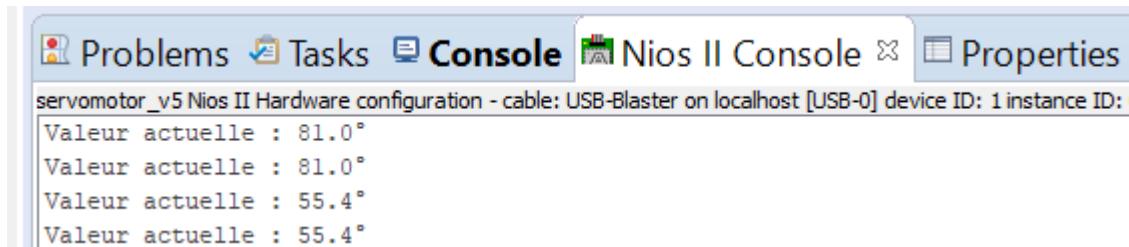
Figure 2.4.3 : Test hardware du servomoteur avec les switchs en position 675 pour un angle de 90° et mesure d'une impulsion haute de 1,75 ms sur l'oscilloscope



Pour une consigne correspondant à 135°, une impulsion de 1,75 ms est mesurée. Cela confirme l'extension effective de la plage angulaire, tout en respectant la limite supérieure fixée à 2,2 ms pour un angle maximal de 180°.

Ainsi, pour valider la nouvelle configuration, plusieurs tests ont été effectués en commandant le servomoteur à l'aide des switchs. Dans le premier cas, tous les switchs étaient éteints, correspondant à une consigne de 0° . La position angulaire mesurée s'est avérée cohérente avec l'impulsion minimale de 0,5 ms, comme attendu. Ensuite, une consigne correspondant à 90° a été testée, en comparaison avec les valeurs théoriques et expérimentales déterminées lors des étapes précédentes. Enfin, une consigne de 135° a été commandée, avec des mesures confirmant la correspondance entre les angles demandés et les signaux générés. Toutes ces vérifications montrent une cohérence totale entre les valeurs des switchs, les largeurs d'impulsion générées et les positions atteintes par le servomoteur.

Figure 2.4.4 : Console Nios ii affichant les angles commandés



```
servomotor_v5 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0
Valeur actuelle : 81.0°
Valeur actuelle : 81.0°
Valeur actuelle : 55.4°
Valeur actuelle : 55.4°
```

Le programme logiciel développé a également été configuré pour afficher en temps réel les positions angulaires commandées via la console du NIOS II SBT. Lors des tests, des valeurs successives ont été relevées pour confirmer la robustesse et la précision du système. Par exemple, une commande angulaire de $81,0^\circ$ a été reproduite avec une stabilité parfaite sur plusieurs cycles, suivie d'un autre test pour une position de $55,4^\circ$.

Les résultats affichés sur le terminal confirment que les angles commandés via les switchs correspondent aux positions angulaires effectivement atteintes par le servomoteur.

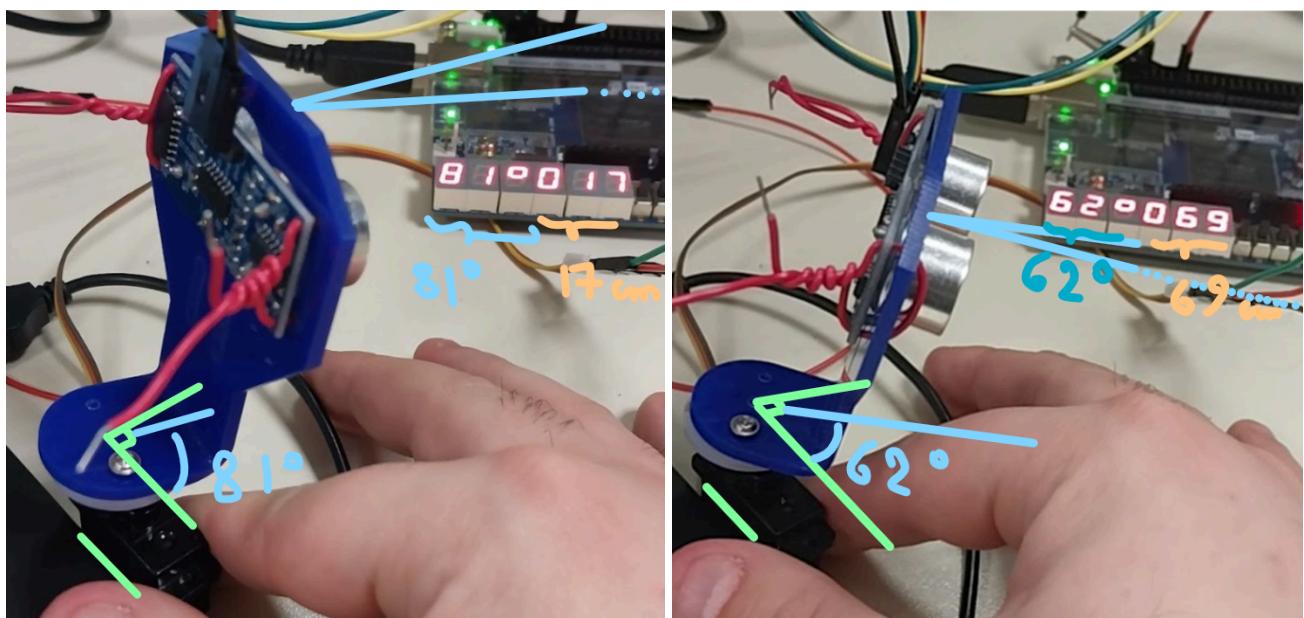
Les validations réalisées dans cette étape montrent que le servomoteur peut être commandé avec une précision au dixième de degré près sur toute la plage de 180° . Ces tests permettent de conclure que l'IP servomoteur est désormais entièrement fonctionnelle et prête pour une intégration dans le projet global.

La prochaine étape consiste à exploiter ce servomoteur pour une rotation dynamique sur 180° , associée au télémètre ultrason. Ce système combiné permettra de détecter des obstacles tout au long de cette plage angulaire, avec des mesures régulières affichées sur le terminal. Les résultats de cette nouvelle fonctionnalité, qui inclura des affichages sous forme de relevés angulaires et de distances détectées, feront l'objet d'une analyse détaillée dans la section suivante.

3. Télémètre tournant : Mesure angulaire distance aux obstacles

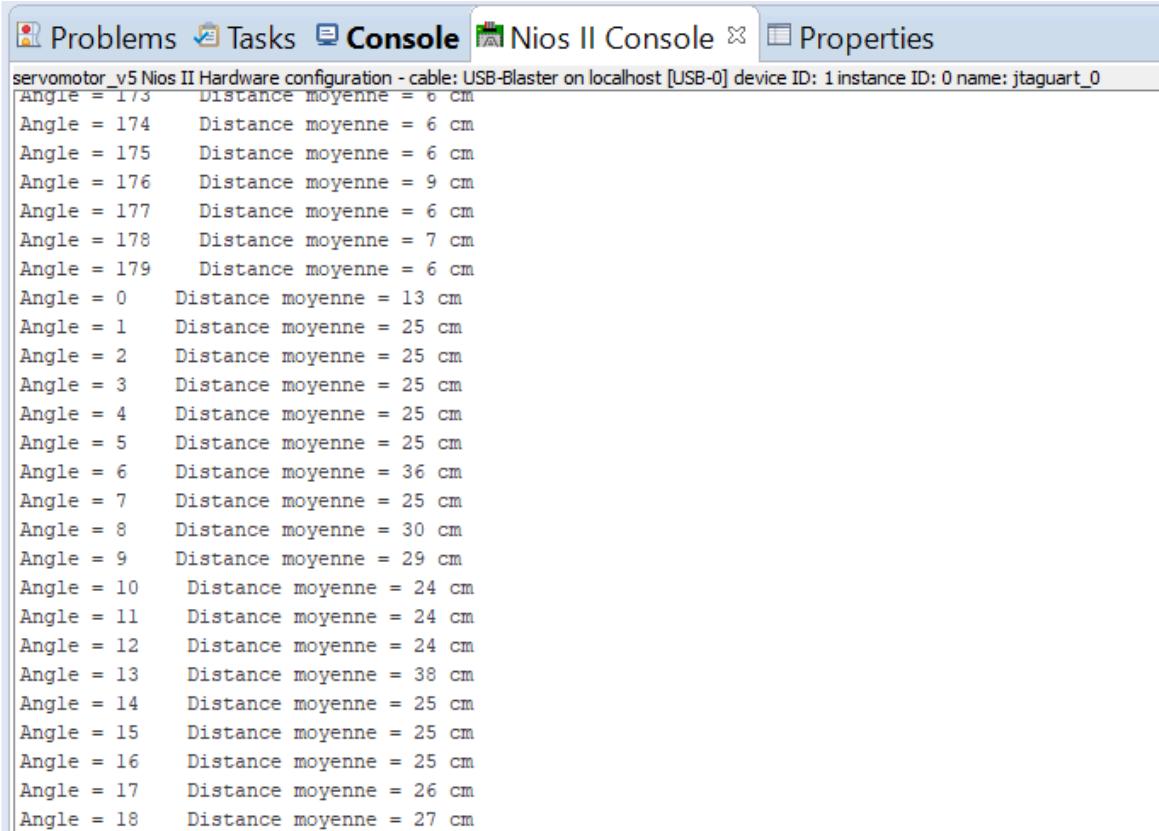
Dans cette partie, le système combine le servomoteur et le télémètre ultrason pour effectuer une rotation dynamique et mesurer des distances à différents angles. L'objectif est d'obtenir un balayage sur une plage angulaire de près de 180° (légèrement restreinte pour des raisons de sécurité) et d'afficher les résultats en temps réel sur les afficheurs 7 segments, ainsi que sur la console NIOS II.

Figure 3.1 : Mesure de distances pour des angles de 62° et 81°



Pour chaque position angulaire atteinte, le système affiche sur les 7 segments l'angle courant sur les deux premiers afficheurs, le symbole “ \circ ” sur le troisième afficheur pour une meilleure ergonomie visuelle, et la distance mesurée en centimètres sur les trois derniers afficheurs. Un détail notable est que l'affichage de l'angle ne montre que les deux derniers chiffres, ce qui signifie qu'au-delà de 99° , l'afficheur revient à 0° et incrémenté jusqu'à 79° pour représenter l'angle maximal de 179° . Par exemple, les deux images présentent des exemples concrets d'angles mesurés respectifs de 62° et 81° , avec des distances associées de 69 cm et 17 cm. Pour une meilleure ergonomie, ces valeurs sont directement visibles sur les afficheurs 7 segments, offrant une représentation claire et instantanée des données collectées.

Figure 3.2 : Affichage des angles et distances mesurés dans la console Nios ii



```

servomotor_v5 Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtaguart_0
Angle = 173 Distance moyenne = 6 cm
Angle = 174 Distance moyenne = 6 cm
Angle = 175 Distance moyenne = 6 cm
Angle = 176 Distance moyenne = 9 cm
Angle = 177 Distance moyenne = 6 cm
Angle = 178 Distance moyenne = 7 cm
Angle = 179 Distance moyenne = 6 cm
Angle = 0 Distance moyenne = 13 cm
Angle = 1 Distance moyenne = 25 cm
Angle = 2 Distance moyenne = 25 cm
Angle = 3 Distance moyenne = 25 cm
Angle = 4 Distance moyenne = 25 cm
Angle = 5 Distance moyenne = 25 cm
Angle = 6 Distance moyenne = 36 cm
Angle = 7 Distance moyenne = 25 cm
Angle = 8 Distance moyenne = 30 cm
Angle = 9 Distance moyenne = 29 cm
Angle = 10 Distance moyenne = 24 cm
Angle = 11 Distance moyenne = 24 cm
Angle = 12 Distance moyenne = 24 cm
Angle = 13 Distance moyenne = 38 cm
Angle = 14 Distance moyenne = 25 cm
Angle = 15 Distance moyenne = 25 cm
Angle = 16 Distance moyenne = 25 cm
Angle = 17 Distance moyenne = 26 cm
Angle = 18 Distance moyenne = 27 cm

```

En parallèle de cet affichage matériel, le terminal de la console Nios ii permet un suivi plus détaillé des mesures réalisées à chaque angle. Les angles sont incrémentés par pas de 1° sur la plage autorisée, et pour chaque position, une distance moyenne est calculée afin d'obtenir une valeur fiable.

Afin de garantir la fiabilité des données et d'améliorer la robustesse du système, plusieurs mécanismes logiciels ont été intégrés. Le calcul de la distance affichée repose sur la moyenne d'un certain nombre de mesures successives, dont le paramètre est ajustable. Ce processus limite l'impact des variations rapides ou bruitées des données provenant du télémètre. Par ailleurs, pour éviter que des valeurs aberrantes issues d'un obstacle trop proche ne perturbent les calculs, une limite maximale a été fixée dans le logiciel. Toute mesure dépassant ce seuil est automatiquement ramenée à zéro.

Enfin, pour des raisons de sécurité mécanique et pour protéger le système, la plage angulaire est volontairement restreinte à une amplitude légèrement inférieure à 180°. Cette précaution limite les risques liés à des mouvements mécaniques excessifs ou imprévus du servomoteur.

Ces améliorations combinées permettent d'obtenir un système stable et fiable, capable de produire un balayage angulaire précis tout en collectant des données pertinentes sur les distances mesurées. L'affichage temps réel sur les afficheurs 7 segments et la console Nios ii offrent une visualisation claire et ergonomique de ces données, ce qui facilite l'analyse des résultats.

4. LEDs Neopixel : intégration esthétique

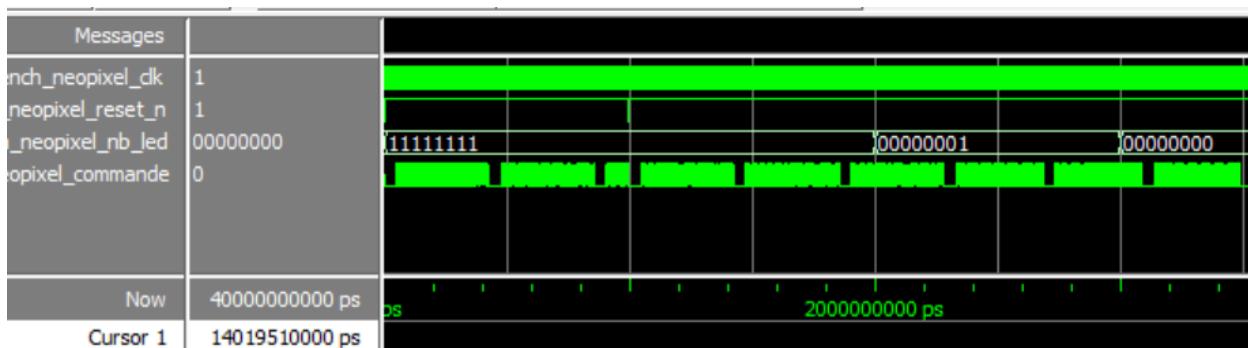
Pour rendre la progression du scan du radar sur 180° plus ludique et visuellement attrayante, nous avons intégré un anneau de 12 LEDs Neopixel à notre système. Cette technologie, largement utilisée dans les dispositifs d'éclairage dynamique, repose sur des LEDs intelligentes WS2812 qui combinent des diodes GRB et un contrôleur intégré. Les LEDs Neopixel offrent l'avantage d'un protocole de communication série basé sur une seule ligne de données, simplifiant le câblage tout en permettant un contrôle précis et indépendant de chaque LED.

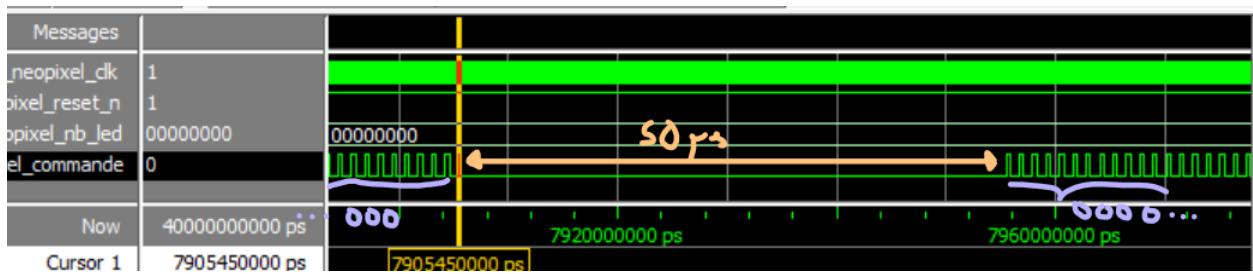
Notre objectif était de créer une barre de chargement circulaire représentant l'état d'avancement du scan, synchronisée avec l'angle courant du servomoteur. Chaque LED sera associée à une section angulaire spécifique. Le design évolue progressivement avec un choix de couleurs inspiré des feux tricolores : les cinq premières LEDs s'allument en bleu nuage pour indiquer le début de la progression ; les quatre suivantes en orange pêche pour signaler un avancement intermédiaire ; les deux suivantes en rose bonbon pour indiquer la phase terminale de la progression. Enfin, une LED blanche éclatante indique la complétion du cycle. Ce design esthétique et fonctionnel simplifiera la compréhension visuelle de la progression de scan du radar.

4.1. Implémentation en VHDL du module Neopixel seul

Pour intégrer les LEDs Neopixel, nous avons développé un module capable de piloter les LEDs WS2812 selon leur protocole. Chaque LED nécessite une trame de données de 24 bits soit 8 bits pour chaque composante : vert, rouge, et bleu. Notre module transmet en boucle une séquence correspondant aux 12 LEDs, soit un total de 288 bits d'impulsion. Les timings spécifiques du protocole ont été respectés pour garantir un fonctionnement correct.

Figure 4.1.1 : Chronogramme du module Neopixel seul

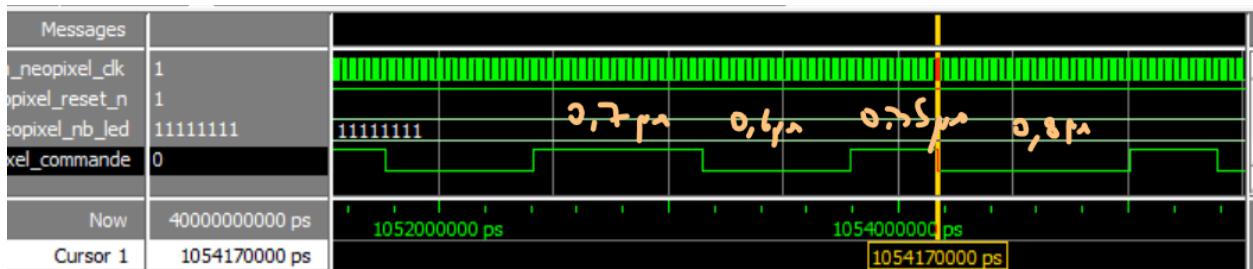




Tout d'abord, nous avons confirmé que l'ordre d'envoi des données est respecté : les 24 bits de chaque LED sont transmis successivement. Nous avons aussi vérifié que seules les données de seulement 12 LEDs sont adressées, et que le reset asynchrone fonctionne : une pause de réinitialisation interrompt la transmission avant de reprendre, garantissant que les données ne sont pas corrompues.

Nous avons également observé que lorsqu'une seule LED est allumée, seuls les 24 bits correspondants contiennent des valeurs non nulles, tandis que les autres LEDs restent éteintes. De plus, lorsque aucune LED n'est allumée, une trame de 12 séquences représentant la couleur noire est envoyée, ce qui garantit l'extinction totale de l'anneau. Nous avons vérifié que la durée de la pause reset était bien de 50 µs, conformément aux spécifications du protocole WS2812. Enfin, le système conserve bien une récurrence des impulsions (les trames sont envoyées en boucle après une pause de reset).

Figure 4.1.2 : Chronogramme du module Neopixel zoomé pour la vérification des durées T1H, T1L, T0H, T0L

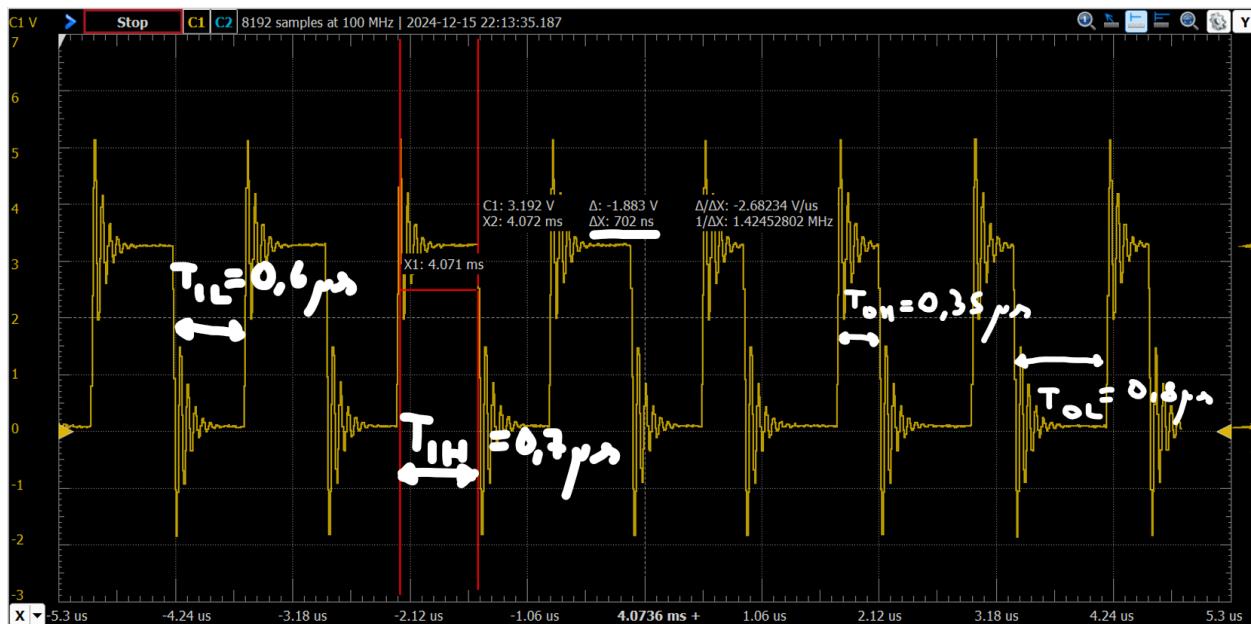


Pour finir, nous avons vérifié que les durées des impulsions respectaient les spécifications tel que $T1H = 0.7\mu s$, $T1L=0.6\mu s$, $T0H=0.35\mu s$, $T0L=0.8\mu s$. Ces résultats confirment que notre module est conforme aux spécifications du fabricant, garantissant ainsi une communication fiable avec les LEDs Neopixel.

4.2. Test unitaire sur carte FPGA des LEDs Neopixel

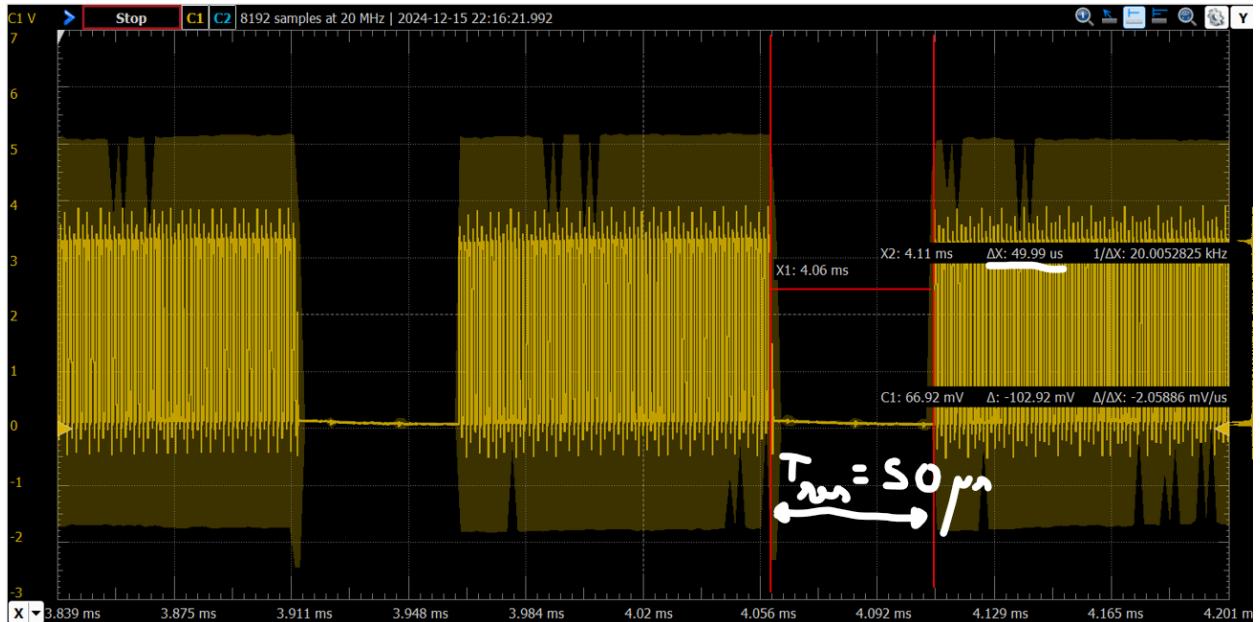
Dans cette étape, des tests unitaires ont été effectués pour vérifier la bonne gestion des signaux, des durées d'impulsion et de pause, ainsi que l'acheminement correct des couleurs vers les LEDs. Dans un premier temps, nous avons utilisé des consignes simples en mode GRB avec des états "tout ou rien" (255 ou 0). Ces tests permettent de simplifier l'analyse en envoyant uniquement des niveaux logiques équivalant à 1 ou 0 sur chaque canal, facilitant ainsi la validation des signaux.

Figure 4.2.1 : Mesure des impulsions T1H, T1L, T0H, T0L sur l'oscilloscope



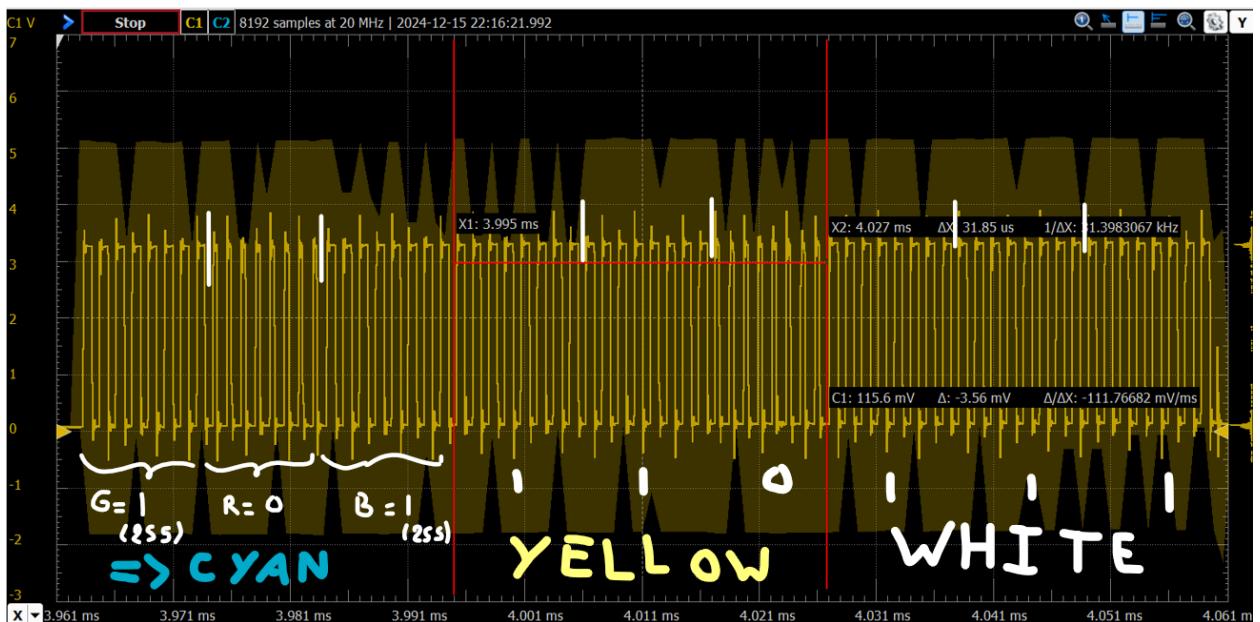
Nous avons mesuré les durées des impulsions T1H, T1L, T0H et T0L à l'aide de l'oscilloscope de l'Analog Discovery 2. Ces durées ont été vérifiées et respectent les spécifications : T1H = 0,7 μs , T1L = 0,6 μs , T0H = 0,35 μs , et T0L = 0,8 μs . Le système fonctionne donc conformément aux exigences des LEDs WS2812.

Figure 4.2.2 : Mesure de la durée du reset à l'état bas sur l'oscilloscope



Nous avons ensuite vérifié la durée de la pause reset, réalisée en abaissant le signal pendant une période définie. Ce test a été facilité en envoyant de manière forcée seulement trois données couleurs répétées en boucle, ce qui permet de bien distinguer les séquences et la pause sur le chronogramme. La durée mesurée est de 50 μ s, ce qui est conforme.

Figure 4.2.3 : Lectures des envoies pour les couleurs simples cyan, jaune, et blanc pour les trois premiers LEDs sur l'oscilloscope

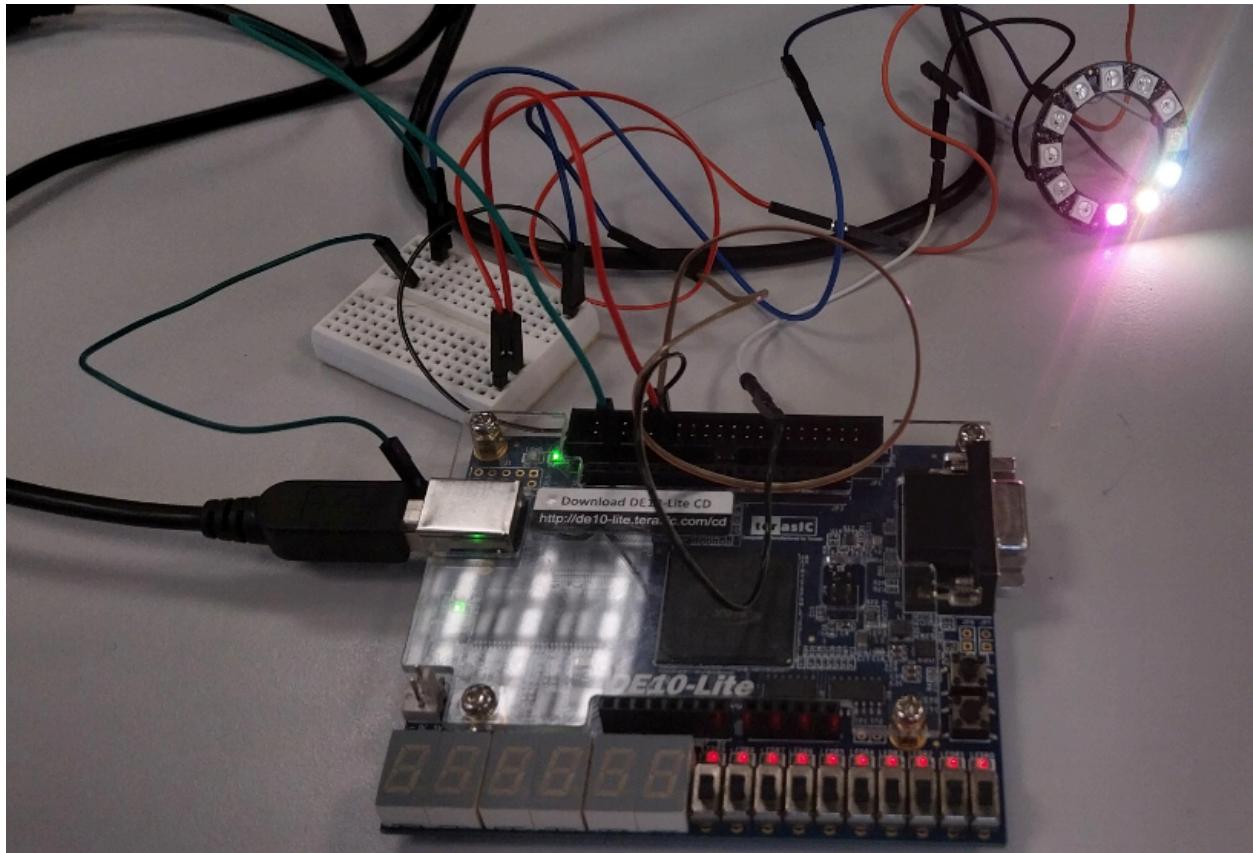


Pour vérifier l'affichage des couleurs primaires combinées en mode simple sur les trois premières LEDs, nous avons utilisé des couleurs spécifiques : le cyan, le jaune et le blanc. Le cyan a été obtenu en activant les canaux vert et bleu tout en désactivant le rouge (GRB = 1, 0, 1). Le jaune a été généré en activant à la fois le vert et le rouge, avec le bleu désactivé (GRB =

1, 1, 0). Enfin, le blanc a été produit en activant simultanément les trois canaux, à savoir le vert, le rouge et le bleu (GRB = 1, 1, 1).

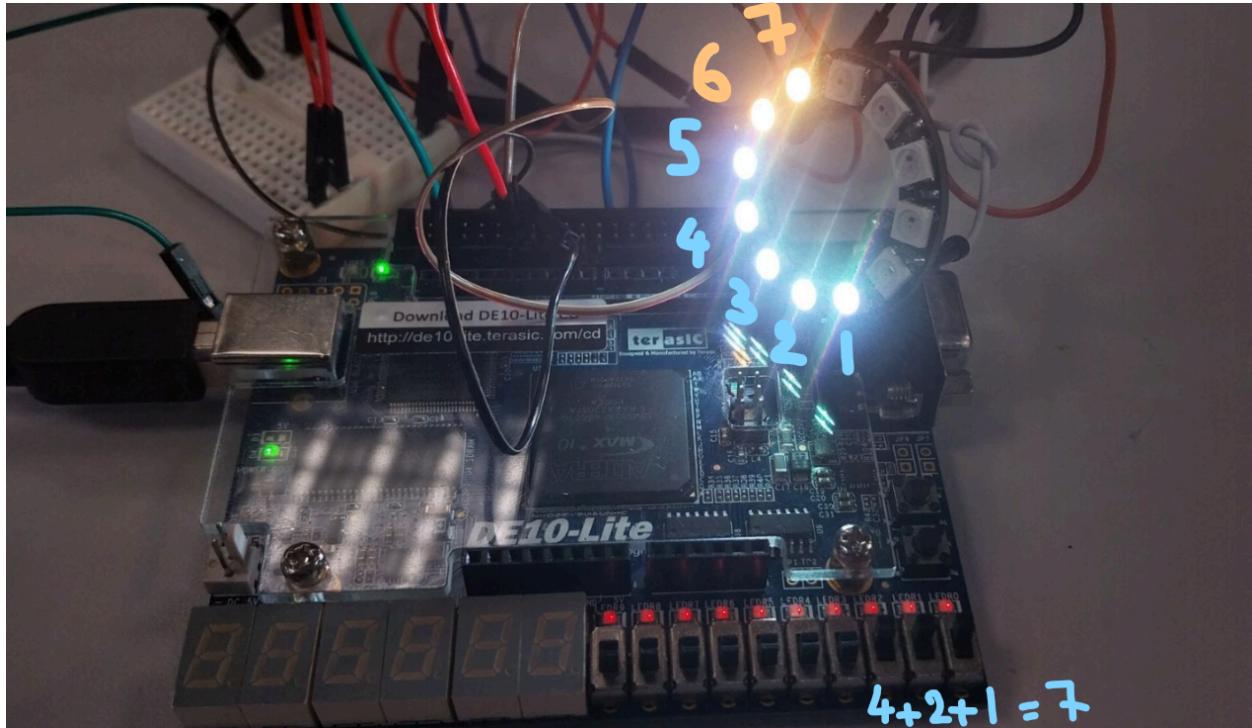
Les résultats, observés à l'oscilloscope, confirment que les couleurs ont été correctement reproduites pour chaque LED, en accord avec les valeurs attendues.

Figure 4.2.4 : Test hardware avec les couleurs complexes bleu nuage, orange pêche, et rose bonbon



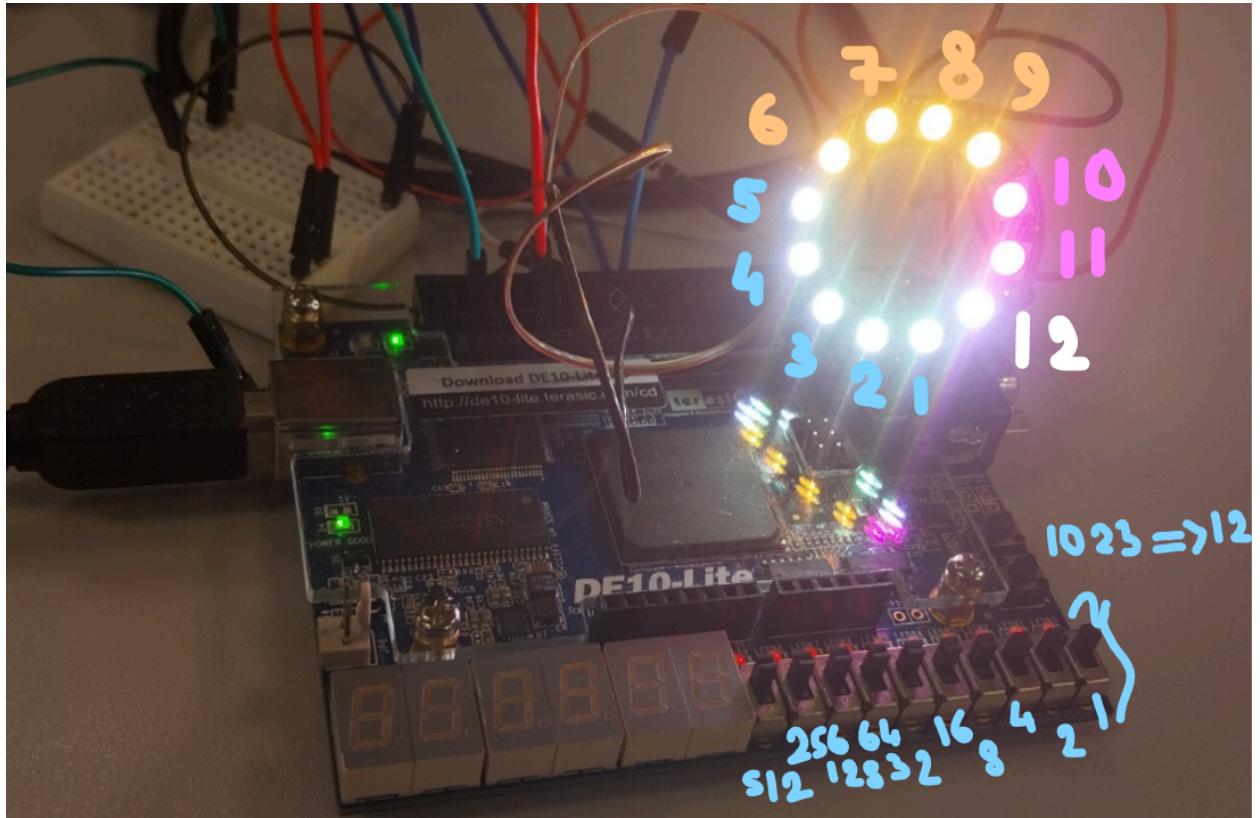
Après les couleurs simples, nous avons réglé les données commandées pour afficher les trois couleurs complexes choisies : bleu nuage, orange pêche, et rose bonbon. Ces couleurs, nécessitant des données plus spécifiques que "tout ou rien", ont été appliquées aux LEDs avec succès. Le résultat confirme que les LEDs reproduisent bien les couleurs souhaitées.

Figure 4.2.5 : Test hardware des LEDs neopixel tel que les switchs sont en position 7



Enfin, des tests hardware ont été réalisés pour valider la gestion des LEDs. Ceux-ci consistaient à utiliser des switchs pour contrôler le nombre de LEDs à allumer, conformément aux couleurs du cahier des charges. Lorsque les switchs indiquaient la valeur 7, les 7 premières LEDs étaient allumées : 5 en bleu, suivies de 2 en orange. Ce test a permis de s'assurer que le système fonctionnait correctement en conditions réelles, avec un nombre réduit de LEDs allumées, sans se soucier du plafond de LEDs autorisées.

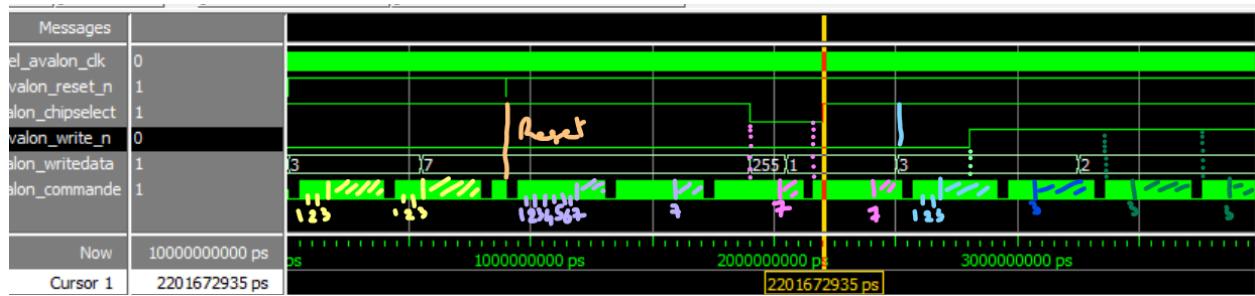
Figure 4.2.6 : Test hardware des LEDs neopixel tel que les switchs sont en position 1023



Ce second test a été effectué pour vérifier le plafond de LEDs allumées et la gestion des données envoyées. Lorsque toutes les switchs sont activées, ce qui correspond à la valeur maximale de 1023, le système doit allumer toutes les 12 LEDs disponibles. Ce test a permis de confirmer que le Neopixel gérait correctement la limite de 12 LEDs. L'affichage des couleurs s'est déroulé comme prévu, avec les cinq premières LEDs en bleu, quatre en orange, deux en rose et une dernière en blanc, ce qui a validé la capacité du système à supporter cette configuration maximale. Ce test a été crucial pour vérifier que notre implémentation ne présentait aucun dysfonctionnement lorsque la limite de LEDs était dépassée.

4.3. Implémentation en VHDL du module Neopixel compatible avec l'interface Avalon

Figure 4.3.1 : Chronogramme du module neopixel compatible avec l'interface Avalon



Dans cette phase de tests, nous avons intégré l'interface Avalon avec le module Neopixel et vérifié son bon fonctionnement avec cette interface. Au début, nous avons commencé par une consigne simple de 3, ce qui nous a permis de confirmer que seuls les trois premiers segments de 24 bits étaient activés, comme prévu. Chaque segment correspond à une LED, et nous avons observé que les données étaient correctement envoyées et affichées pour ces trois LEDs.

Ensuite, nous avons effectué un test pour vérifier que la mise à jour des données pendant une transmission ne perturbait pas l'affichage des LEDs. En particulier, nous avons vérifié que les données ne se mettaient à jour qu'après la fin de l'envoi des bits de la LED en cours. Cela permet d'éviter des transitions abruptes, comme par exemple un passage de 2 à 10 LEDs allumées d'un coup, ce qui perturberait l'affichage visuel. Nous avons donc confirmé que les mises à jour étaient bien synchronisées avec la fin de la transmission des bits d'une LED complète.

Un autre aspect critique testé a été le comportement du signal de reset. Nous avons validé que le reset fonctionnait de manière asynchrone. En d'autres termes, nous avons vérifié que la remise à zéro du système était bien effectuée indépendamment de l'état de la transmission en cours, sans interférer avec les autres signaux. Le reset s'est révélé fiable et correct, sans provoquer d'instabilité.

Nous avons ensuite testé les écritures de données. Pour cela, nous avons envoyé une consigne simple, où nous avons demandé l'allumage de 7 LEDs. Nous avons observé que les données étaient envoyées correctement pour ces 7 LEDs, mais que, lorsque le chipselect était désactivé, les données suivantes ne se mettaient pas à jour. Par exemple, nous avons demandé l'affichage de 255 LEDs, mais seules les 7 premières LEDs étaient affectées, et les autres données ne se sont pas mises à jour, comme attendu. Cela a confirmé que l'utilisation du chip select fonctionne correctement pour limiter l'envoi des données.

De plus, nous avons testé le comportement lors du changement de consigne pendant une transmission. Lorsque nous avons modifié la consigne de 7 à 1 LED, la mise à jour a été retardée jusqu'à ce que la transmission en cours soit terminée. Le changement de consigne n'a donc eu aucun impact immédiat, ce qui a permis de garantir la stabilité de l'affichage. Ensuite, nous avons modifié la consigne à 3 LEDs et nous avons observé que la mise à jour s'est effectuée comme prévu, sans aucun problème.

Enfin, nous avons testé le comportement lorsque nous avons modifié les signaux write_n et la consigne à 2 LEDs. Nous avons constaté que, lorsque write_n était activé, la consigne ne se mettait pas à jour et se maintenait à 3 LEDs. Ce test a permis de confirmer que la gestion du write_n était correcte et qu'il n'y avait pas de mise à jour prématurée des données pendant l'envoi.

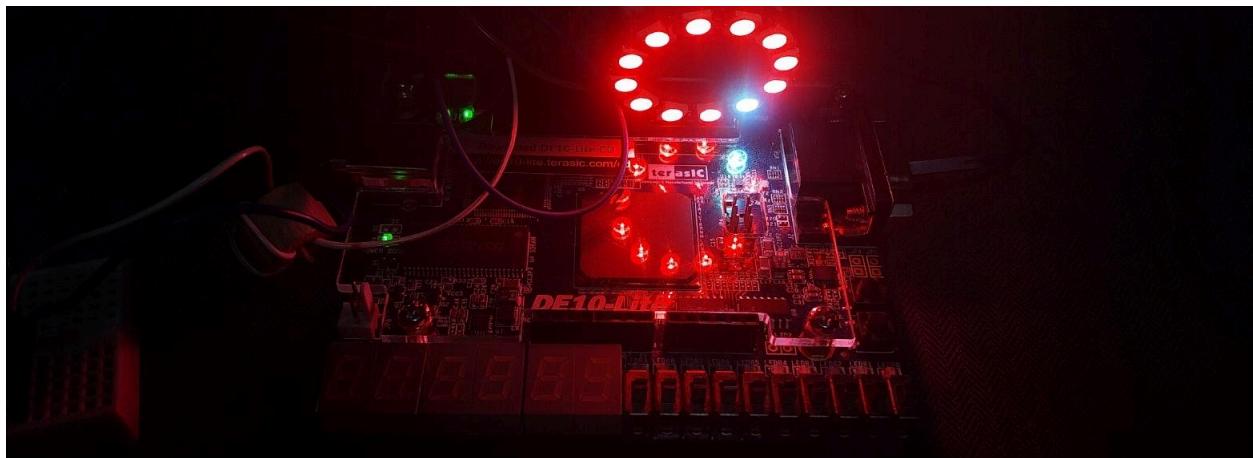
Dans l'ensemble, tous les tests ont montré que l'implémentation avec l'interface Avalon fonctionnait correctement. Les signaux de contrôle comme chipselect, write_n, et le reset ont été correctement gérés et ont permis une transmission stable des données pour l'affichage des LEDs Neopixel.

4.4. Programmation logicielle sur carte FPGA du module Neopixel

Dans cette phase de développement, nous avons programmé le module Neopixel sur la carte FPGA afin de tester et vérifier son fonctionnement via un contrôle logiciel. Pour cela, nous avons choisi de faire un test progressif en augmentant le nombre de LEDs allumées par un éventuel pourcentage artificiel. Ce test consiste à incrémenter progressivement l'avancement, en revenant à zéro après avoir allumé les 12 LEDs. Ce processus a permis de vérifier que l'envoi des données et l'allumage des LEDs se faisaient correctement, et que le contrôle logiciel était bien fonctionnel.

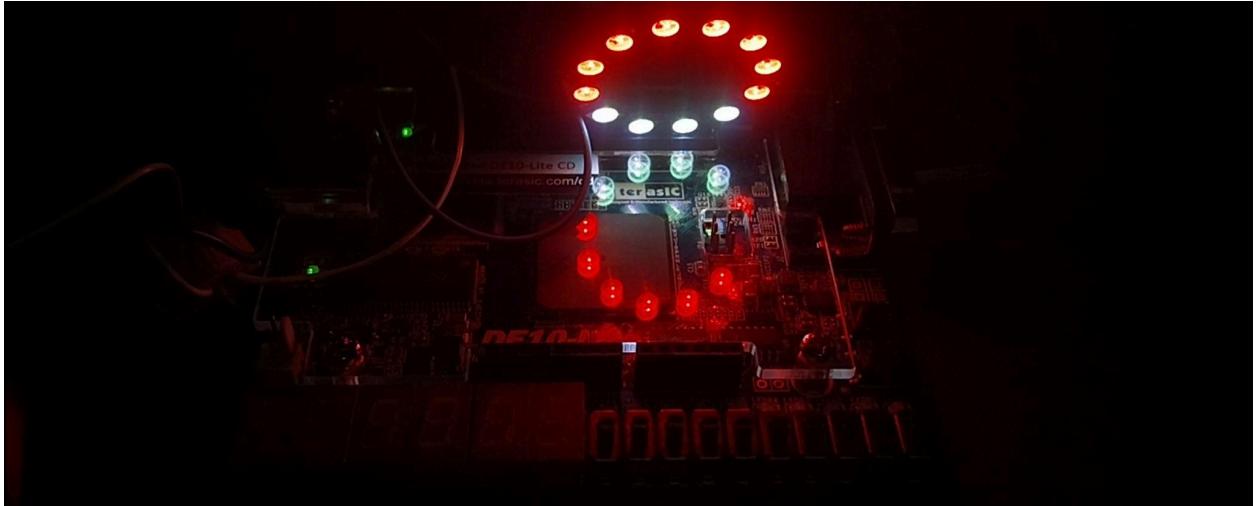
Lors des premiers tests, les LEDs complémentaires étaient éteintes, ce qui donnait un effet visuel assez simple. Pour améliorer cela et rendre l'affichage plus dynamique, nous avons décidé de modifier la couleur des LEDs complémentaires en les passant au rouge. Cette modification a immédiatement apporté un effet visuel plus prononcé et épique, rendant l'affichage plus attrayant et plus facile à distinguer.

Figure 4.4.1 : Test hardware du module neopixel pour une consigne de N = 1 LED.



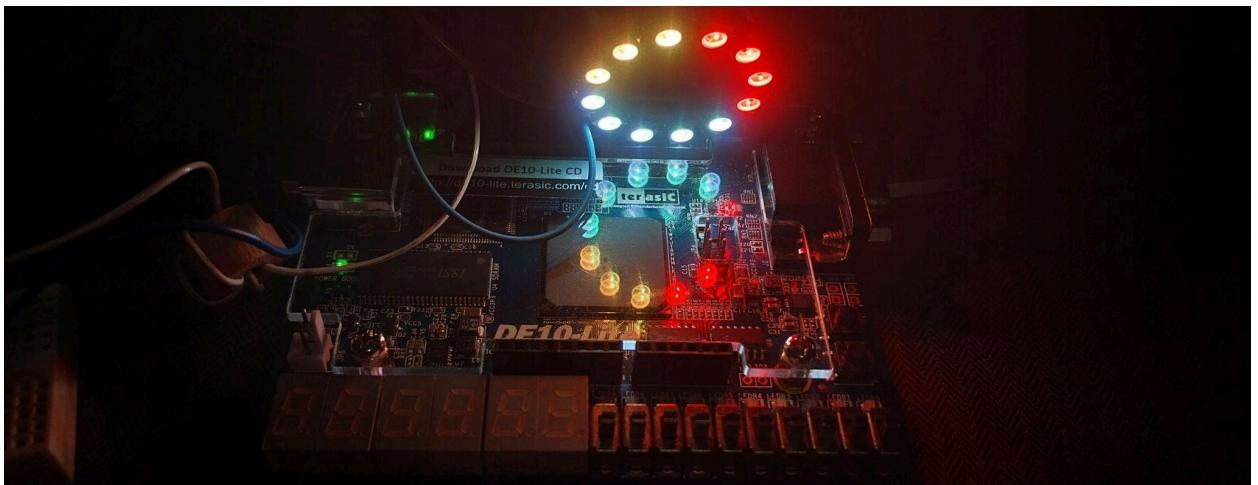
Nous avons ensuite testé le Neopixel pour différentes consignes, en commençant par l'allumage d'une seule LED. Lorsque la consigne était de $N = 1$ LED, une seule LED bleue s'allumait, tandis que les 11 autres LEDs restaient éteintes ou rouges, selon l'état du système. Ce test a confirmé que le système gérait correctement l'allumage d'une LED spécifique.

Figure 4.4.2 : Test hardware du module neopixel pour une consigne de $N = 4$ LEDs.



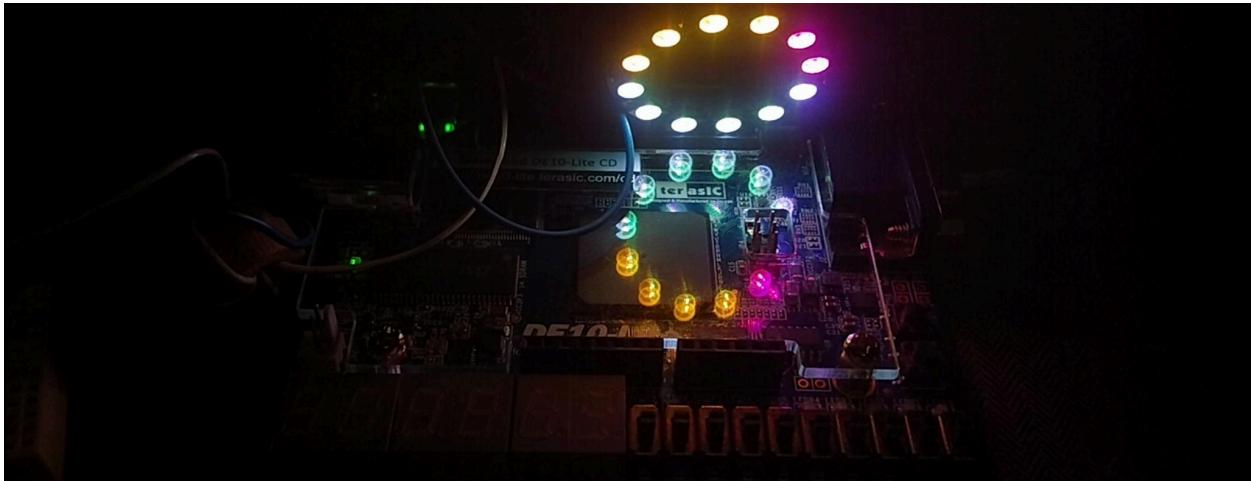
Ensuite, avec $N = 4$ LEDs, les quatre premières LEDs s'allument, tandis que les autres restent éteintes ou rouges. Dans ce cas, les LEDs allumées étaient toutes de la même couleur, ce qui simplifie la gestion visuelle. Ce test a confirmé que le module pouvait allumer plusieurs LEDs sans problème, en respectant l'ordre et la consigne.

Figure 4.4.3 : Test hardware du module neopixel pour une consigne de $N = 8$ LEDs.



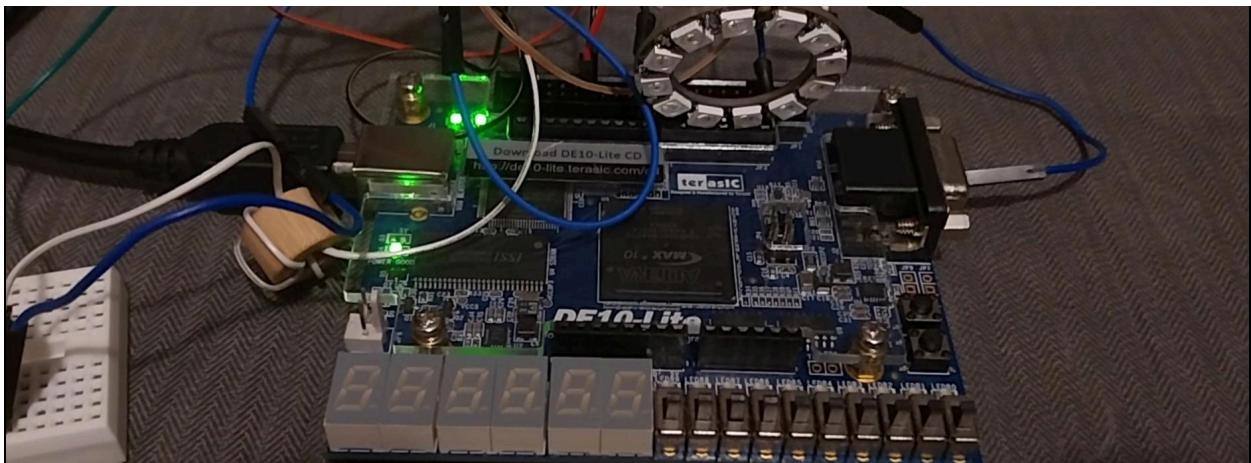
Dans le cas de $N = 8$ LEDs, les huit premières LEDs se sont allumées. Les cinq premières étaient bleues, suivies de trois LEDs orange. Ce test a confirmé que le module pouvait allumer des couleurs différentes, avec la bonne séquence et les bonnes durées d'impulsion pour chaque LED.

Figure 4.4.4 : Test hardware du module neopixel pour une consigne de $N = 12$ LEDs.



Lors de l'essai avec $N = 12$ LEDs, toutes les LEDs se sont allumées correctement. Les cinq premières étaient bleues, suivies des quatre LEDs orange, des deux LED rose et de la dernière blanche. Chaque test a permis de vérifier que le système réagissait correctement en fonction du nombre de LEDs allumées, tout en maintenant la bonne gestion des couleurs complémentaires.

Figure 4.4.5 : Test hardware du module neopixel pour une consigne de $N = 0$ LED.



Enfin, un test a été effectué pour vérifier le comportement lorsque $N = 0$ LED. Dans ce cas, toutes les LEDs étaient éteintes, ce qui a confirmé que le système pouvait éteindre proprement toutes les LEDs lorsque nécessaire. Cet état est particulièrement important pour la phase d'intégration suivante, où il est essentiel de désactiver les LEDs avant que le servomoteur ne revienne en position zéro ou attende avant de commencer le scan. Cette étape de désactivation des LEDs est cruciale pour une gestion correcte du processus et l'intégration finale au radar.

Chaque test a permis de confirmer que le module Neopixel fonctionnait de manière stable et correcte, avec un contrôle précis sur le nombre de LEDs allumées, indépendamment du nombre choisi dans la consigne. En parallèle, l'ajout de LEDs complémentaires (rouges) a apporté un rendu visuel plus riche et plus dynamique, ce qui a permis d'améliorer l'expérience visuelle et de mieux distinguer les états du système.

4.5. Intégration des LEDs Neopixel au radar

<https://f-leb.developpez.com/tutoriels/fpga/controleur-vga/>

Conclusion

Bien que le projet n'ait pas été achevé et ait connu un démarrage sur les chapeaux de roues, avec un très long blocage au niveau de l'UAL et des problèmes personnels, celui-ci m'aura beaucoup apporté, notamment une grande fierté personnelle. J'espère également avoir su prouver, par la lecture de ce rapport, que j'ai compris les principaux aspects du module et ai fini par obtenir une organisation plus claire et structurée, tant de mon environnement de développement informatique que de mon code lui-même, en grande partie grâce au soutien et aux conseils de Thibault. => Faire une conclu plus positive cette fois + ici résumer les résultats, jusqu'où j'ai avancé

J'ai finalement trouvé une forme de libération et surtout une satisfaction personnelle qui me donne encore plus de motivation pour la suite de mon parcours universitaire et humain. Ce projet m'a permis de mieux comprendre la gestion des données dans le processeur, leur déplacement via les bus de données, ainsi que la signification concrète des signaux sortant des composants. J'ai également beaucoup apprécié en savoir plus sur les composants que l'on rencontre souvent sur les schémas électroniques de nos documentations techniques et mieux les comprendre. => Et là que je suis très fier de moi car ce projet m'a demandé beaucoup d'investissement et même si cela me prends plus de temps que certain, j'ai pu aller le plus loin possible et j'ai vraiment beaucoup aimé réaliser ce projet et encore une fois ça m'aide à comprendre le sens de ma formation.

Pour finir, j'espère également que notre belle filière Ei, EiSE, Ei2I, E2i ELi, SETi, MCR, GPi continuera de prospérer comme elle l'a toujours fait, grâce à Thibault HILAIRE, Dimitri GALAYKO, Yann DOUZE, Roselyne CHOTIN, Annick ALEXANDRE, Andrea PINNA, Sylvain FERUGLIO, Cécile BRAUNSTEIN, Benoît FABRE, et enfin Michel REDON, et que toutes les futures promotions s'y sentiront aussi épanouies que j'aurais pu l'être, tout comme l'a été notre grand frère Monsieur VIATEUR.