

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 mon ami, Daniel FERREIRA LARA, qui, a su m'épauler tout au long de cette année et m'encourager.

Un très grand merci aussi à Monsieur MARTIN qui m'a beaucoup appris pendant nos séances encadrées depuis le cinquième semestre. Avec son collègue Monsieur PINNA, qui par le cours d'électronique numérique 1, m'ont fait découvrir une nouvelle passion pour ce langage étant l'amour parfait entre l'informatique et l'électronique.

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 à lui et à notre très bien aimé Thibault (alias Papa des Ei) qui ont beaucoup participé à mon épanouissement personnel et qui m'ont appris à mieux structurer mes codes.

Pour finir, j'aimerais remercier toutes ces personnes extraordinaires que j'ai rencontrées cette année 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 Pitit 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.....	4
1.1. Implémentation en VHDL du télémètre seul.....	4
Figure 1.1.1 : Chronogramme du télémètre à ultrason.....	5
1.2. Essai sur carte FPGA du télémètre seul.....	5
Figure 1.2.1 : Test hardware du télémètre pour d = 10 cm.....	6
Figure 1.2.2 : Test hardware du télémètre pour d = 4 cm.....	6
Figure 1.2.3 : Test hardware du télémètre pour d = 6 cm.....	6
1.3. Implémentation en VHDL du télémètre compatible avec l'interface Avalon.....	7
Figure 1.3.1 : Chronogramme du télémètre à ultrason pour l'interface Avalon.....	7
1.4. Programmation logicielle sur carte FPGA du télémètre.....	8
Figure 1.4.1 : Console Nios ii affichant les mesures post traitées du télémètre.....	9
Figure 1.4.2 : Test hardware du télémètre avec affichage pour d = 10 cm.....	9
Figure 1.4.3 : Test hardware du télémètre avec affichage pour d = 9 cm.....	10
Figure 1.4.4 : Test hardware du télémètre avec affichage pour d = 5 cm.....	10
Figure 1.4.4 : Test hardware du télémètre avec affichage pour d = 0 cm.....	11
2. Servomoteur.....	12
2.1. Implémentation en VHDL du servomoteur seul.....	12
Figure 2.1.1 : Chronogramme du servomoteur seul.....	12
2.2. Test unitaire sur carte FPGA du servomoteur seul.....	13
Figure 2.2.1.a : Test hardware du servomoteur pour un angle de 90,0°.....	14
Figure 2.2.1.b : Mesure d'une impulsion haute de 2 ms sur l'oscilloscope.....	14
Figure 2.2.2.a : Test hardware du servomoteur pour un angle de 45,0°.....	15
Figure 2.2.2.b : Mesure d'une impulsion haute de 1,5 ms sur l'oscilloscope.....	15
Figure 2.2.3.a : Test hardware du servomoteur pour un angle de 0,0°.....	16
Figure 2.2.3.b : Mesure d'une impulsion haute de 1 ms sur l'oscilloscope.....	16
Figure 2.2.4.a : Test hardware du servomoteur pour un angle de 22,5°.....	17
Figure 2.2.4.b : Mesure d'une impulsion haute de 1,25 ms sur l'oscilloscope.....	17
2.3. Implémentation en VHDL du servomoteur compatible avec l'interface Avalon.....	18
Figure 2.3.1 : Chronogramme du servomoteur compatible avec l'interface Avalon...	18
2.4. Programmation logicielle sur carte FPGA du servomoteur.....	19
Conclusion.....	21
Annexe.....	22

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.

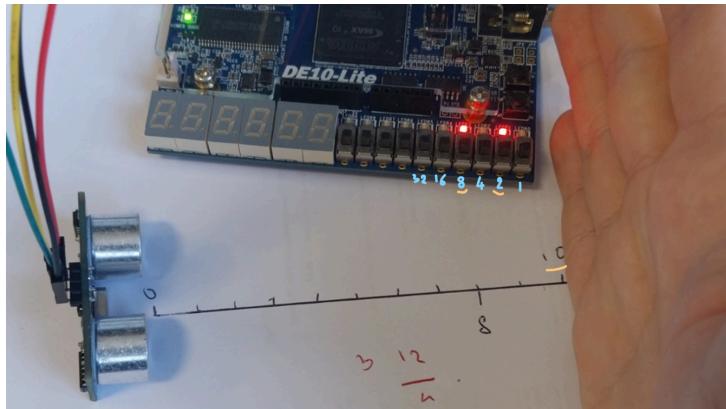
1.2. Essai sur carte FPGA du télémètre seul

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.

Bla bla pour vérifié la proportionnalité.

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 donnée 8 + 2. Ce résultat a permis de vérifier la juste proportionnalité entre la durée du signal echo et la distance réelle, et de valider les calculs de conversion pour cette plage de mesure.

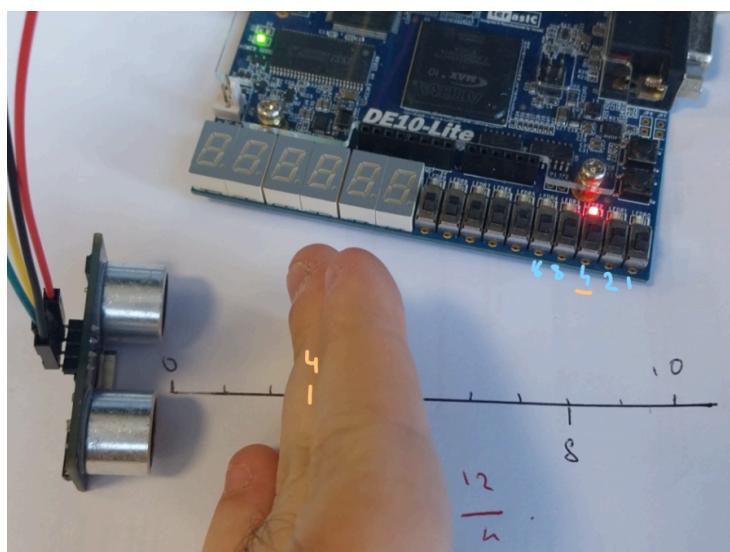
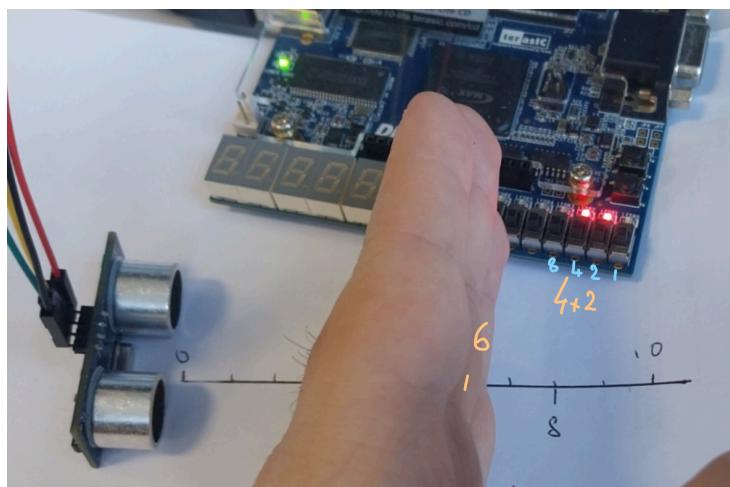


Figure 1.2.2 : Test hardware du télémètre pour $d = 4 \text{ cm}$

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 $d = 6 \text{ cm}$

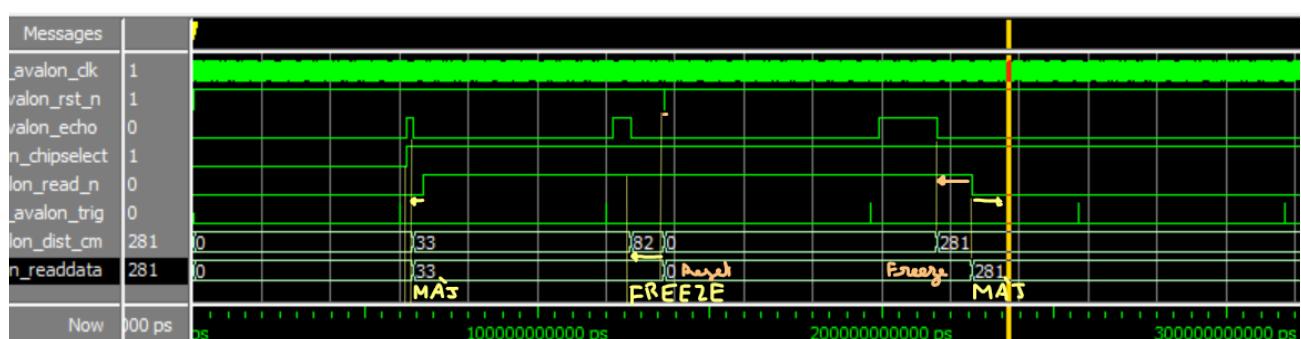


Enfin, une variation de 2 cm a été introduite, en déplaçant cette fois ci l'obstacle à 6 cm. Ce test visait à vérifier la capacité du système à détecter des variations subtiles de distance. Le télémètre a mesuré la distance correctement, avec une précision suffisante pour distinguer clairement ce petit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

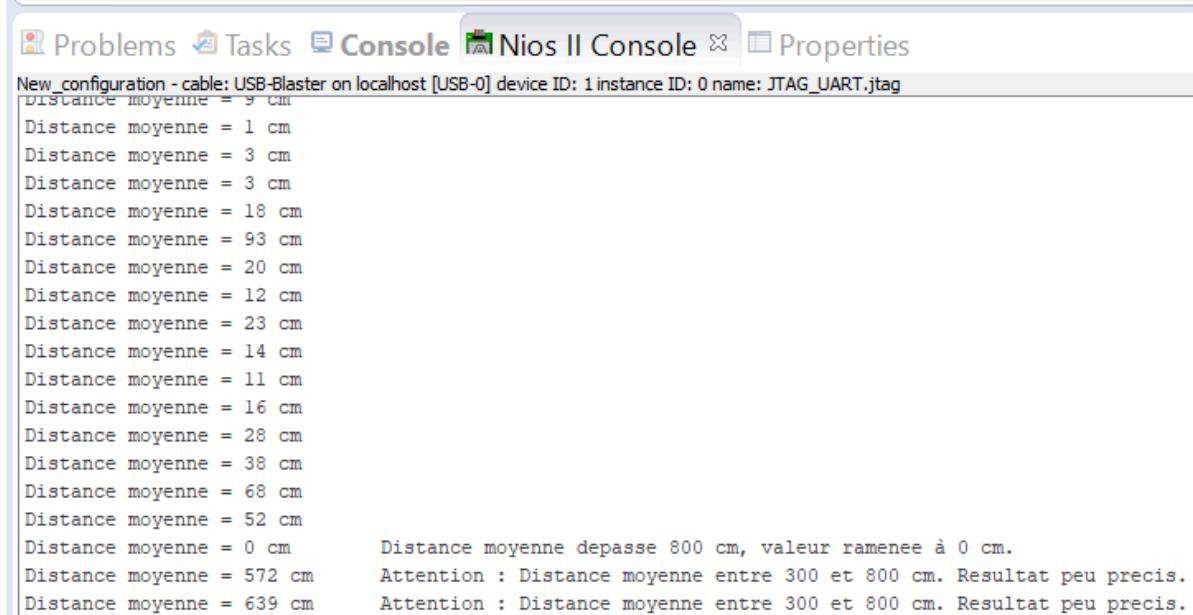
1.4. Programmation logicielle sur carte FPGA du télémètre

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



```

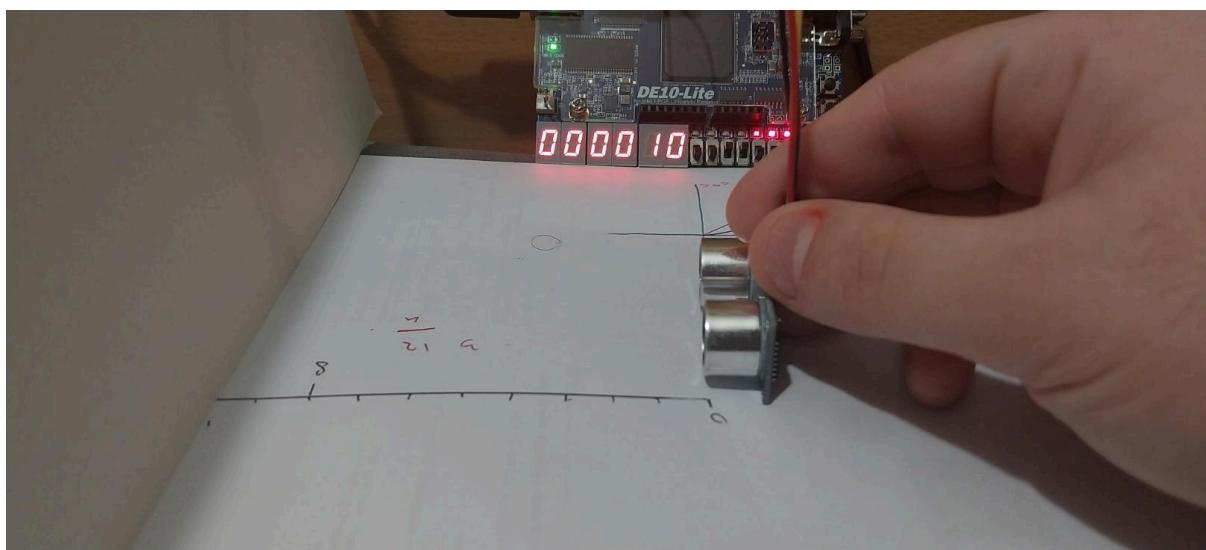
Problems Tasks Console Nios II Console Properties
New_configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: JTAG_UART.jtag
distance moyenne = 3 cm
Distance moyenne = 1 cm
Distance moyenne = 3 cm
Distance moyenne = 3 cm
Distance moyenne = 18 cm
Distance moyenne = 93 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.

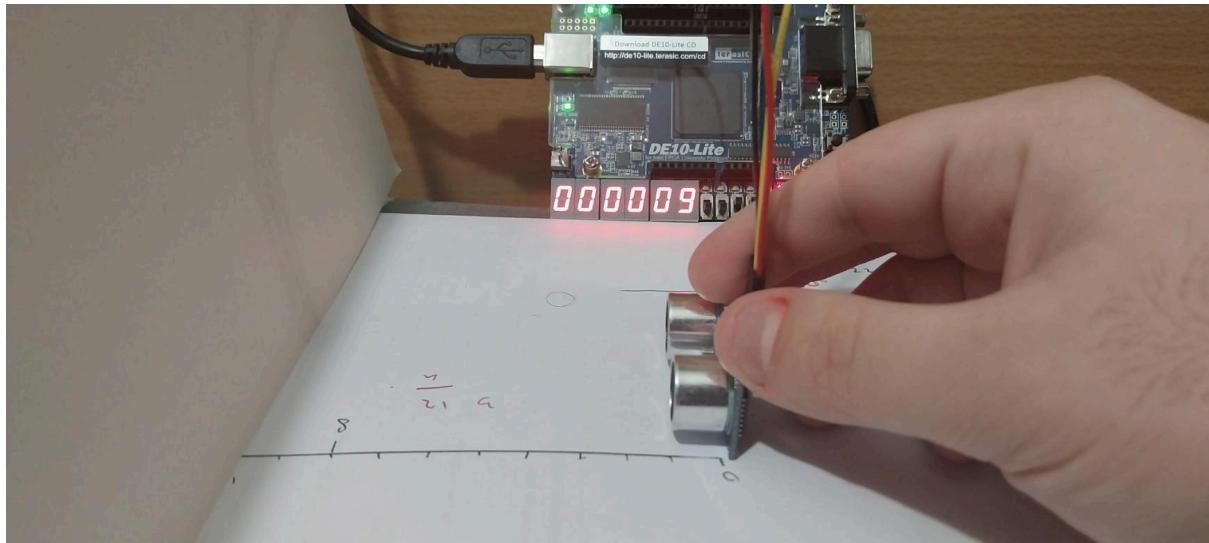
Figure 1.4.2 : Test hardware du télémètre avec affichage pour $d = 10 \text{ cm}$



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.

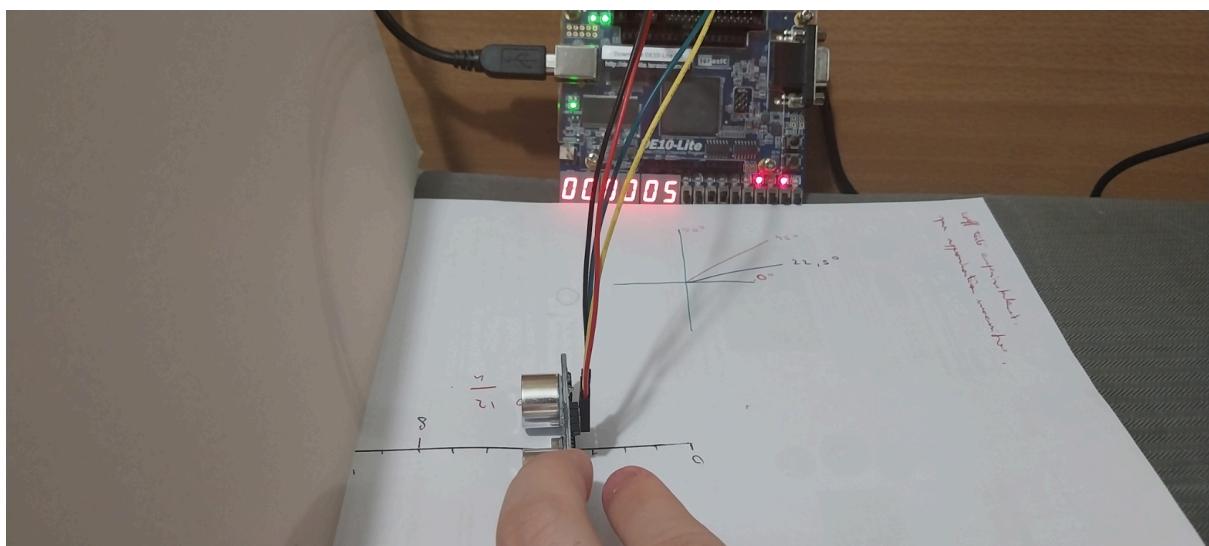
Sur ce test nous observons le bon fonctionnement de l'affichage pour une distance de 10 cm, une valeur arbitraire choisie pour s'assurer que les résultats sont correctement affichés.

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



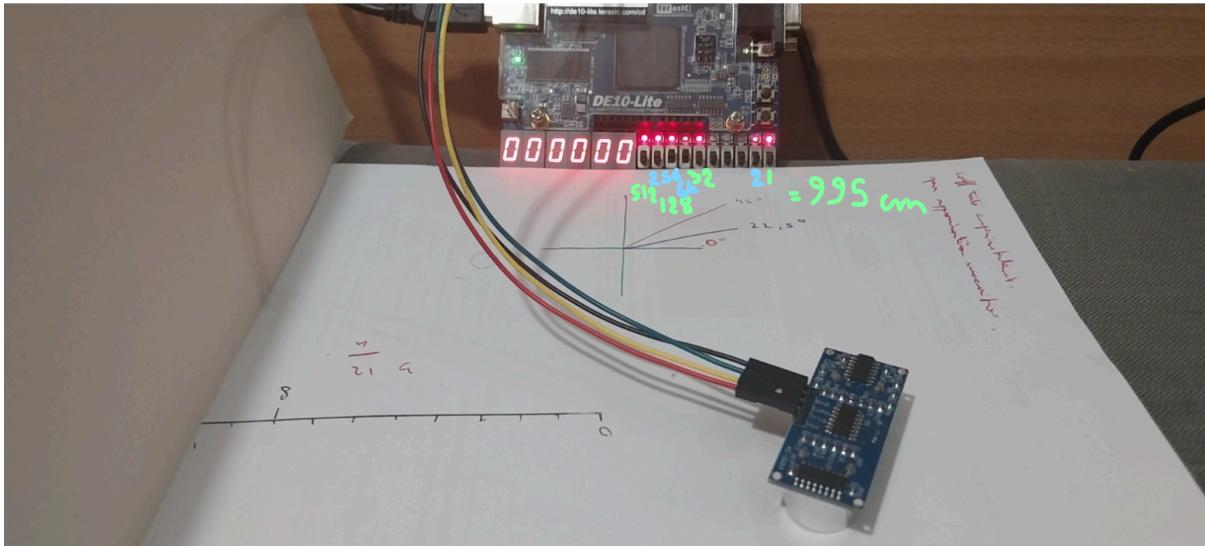
Ensuite, nous avons réalisé un test à 9 cm pour observer si le système détecte et affiche des variations faibles de distance. La précision et la stabilité ont été validées pour ces mesures proches.

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 valeurs 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°

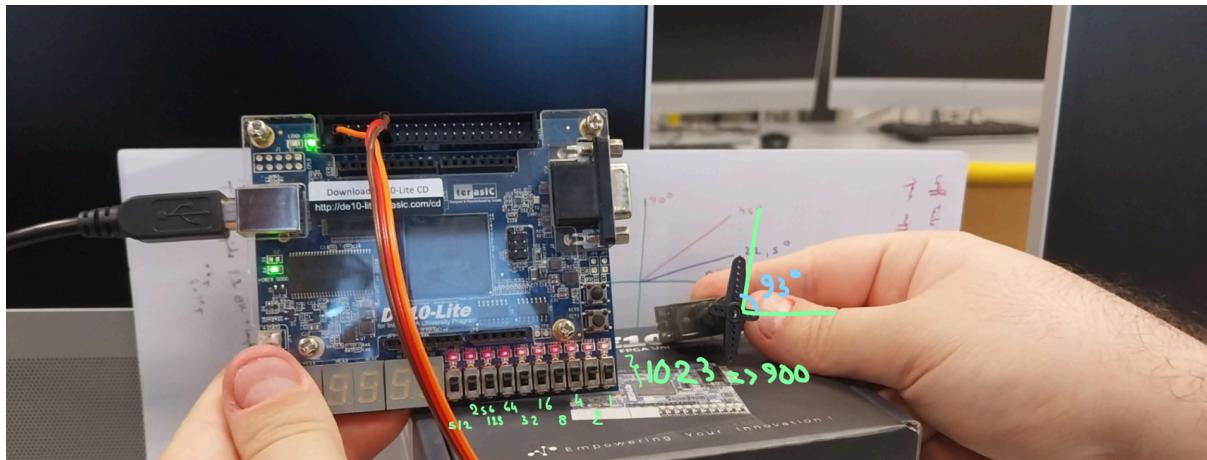
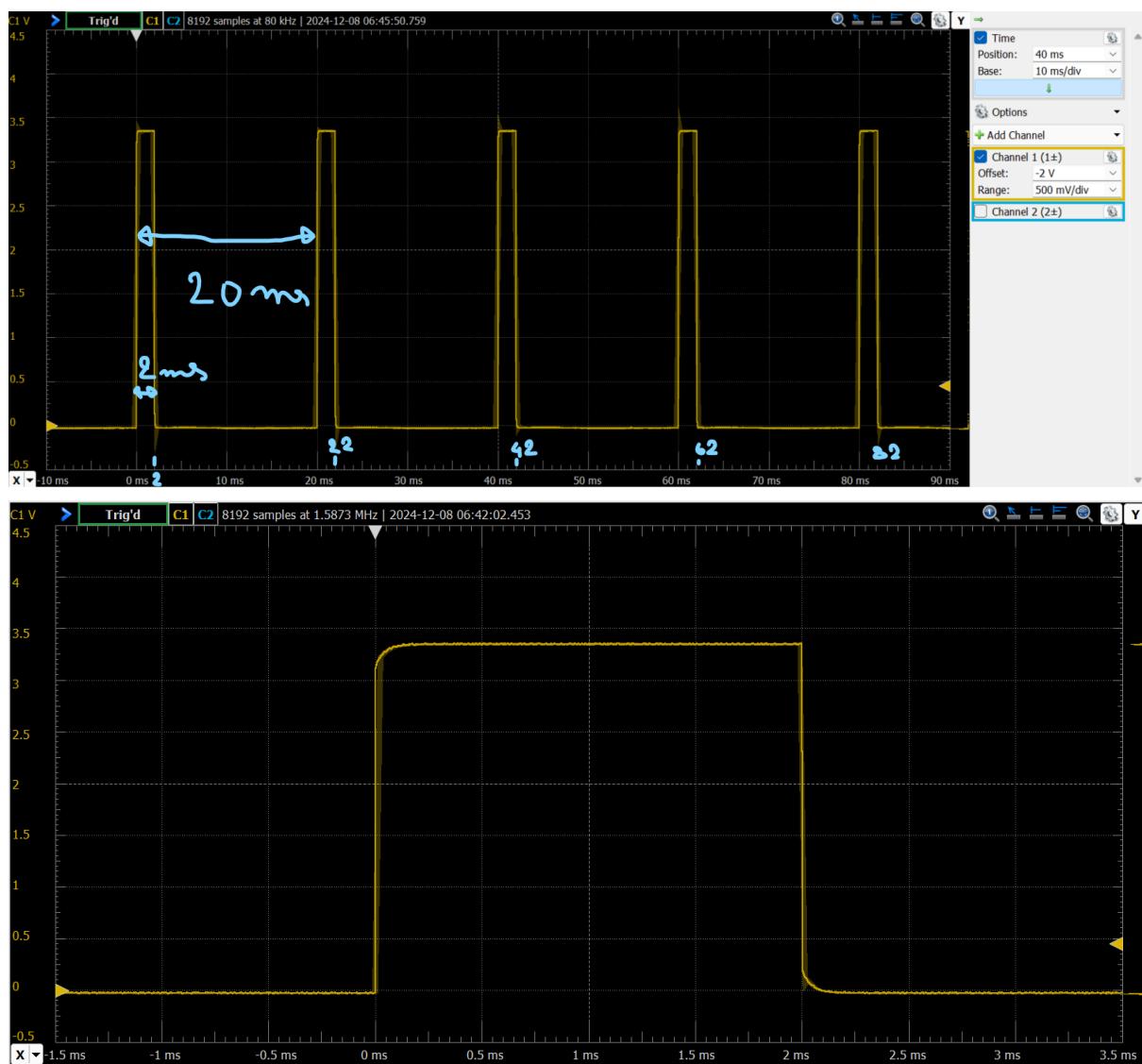


Figure 2.2.1.b : Mesure d'une impulsion haute de 2 ms sur l'oscilloscope



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 $45,0^\circ$

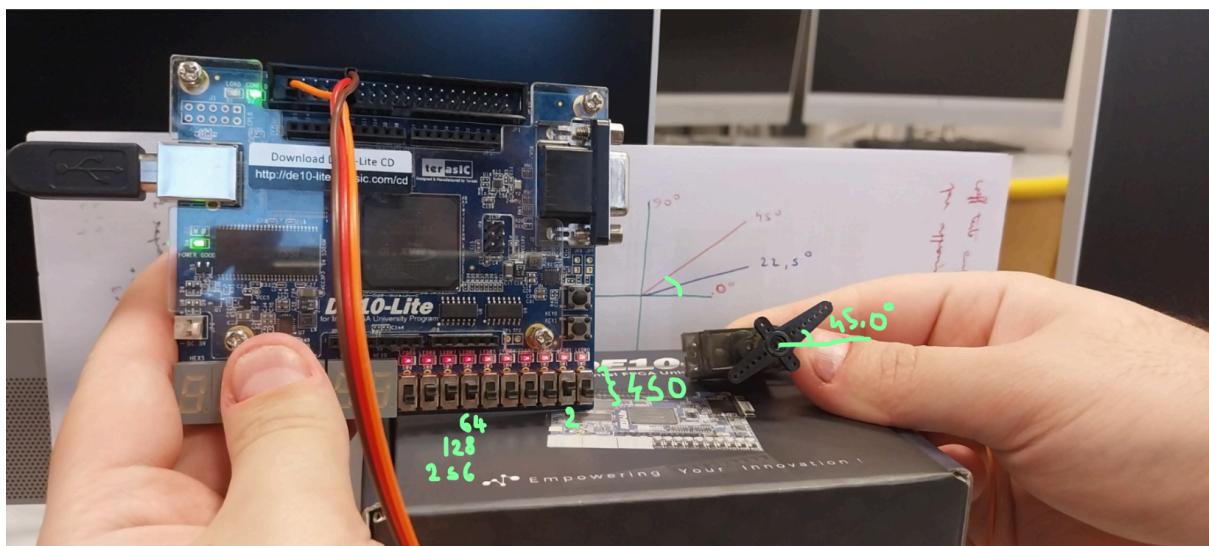
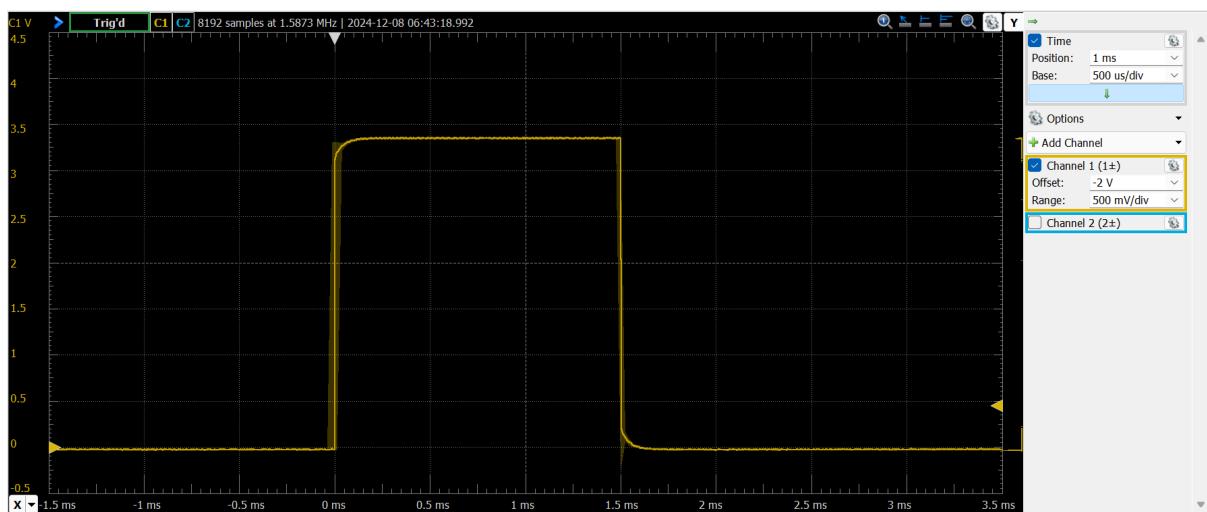


Figure 2.2.2.b : Mesure d'une impulsion haute de 1,5 ms sur l'oscilloscope



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°

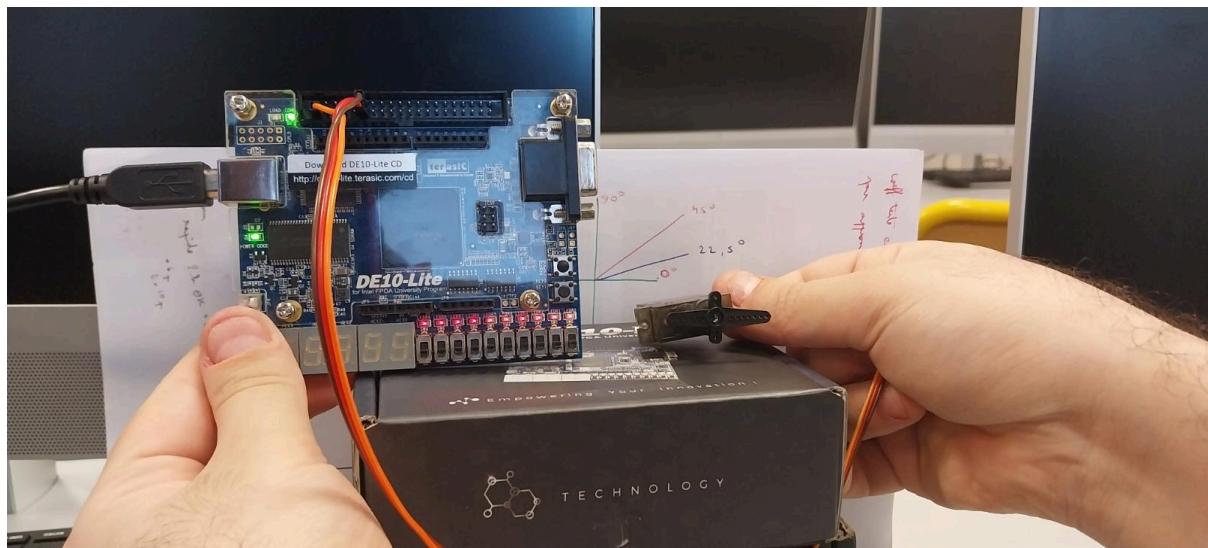
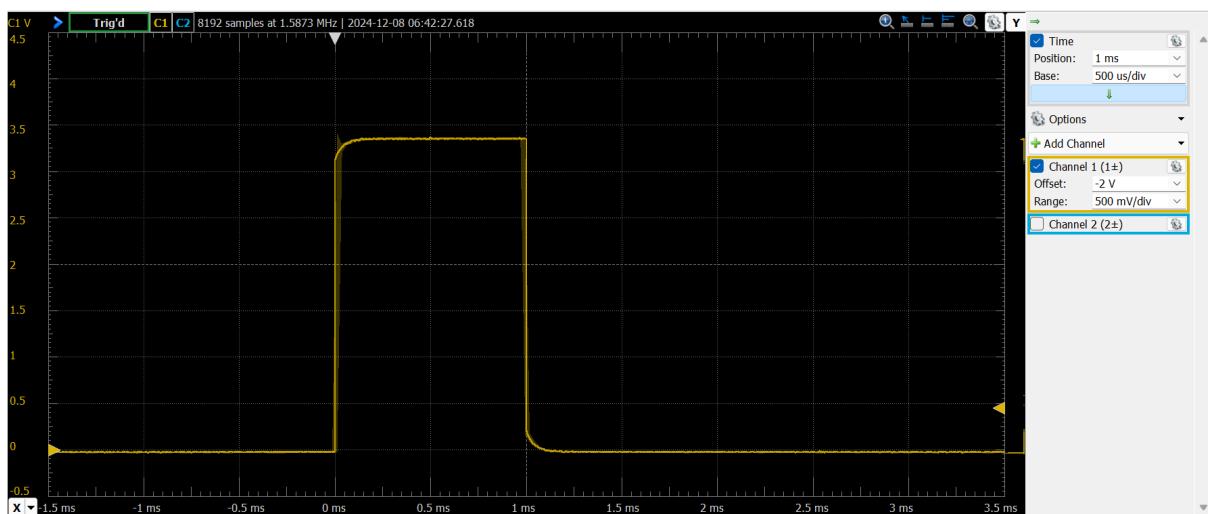


Figure 2.2.3.b : Mesure d'une impulsion haute de 1 ms sur l'oscilloscope



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°

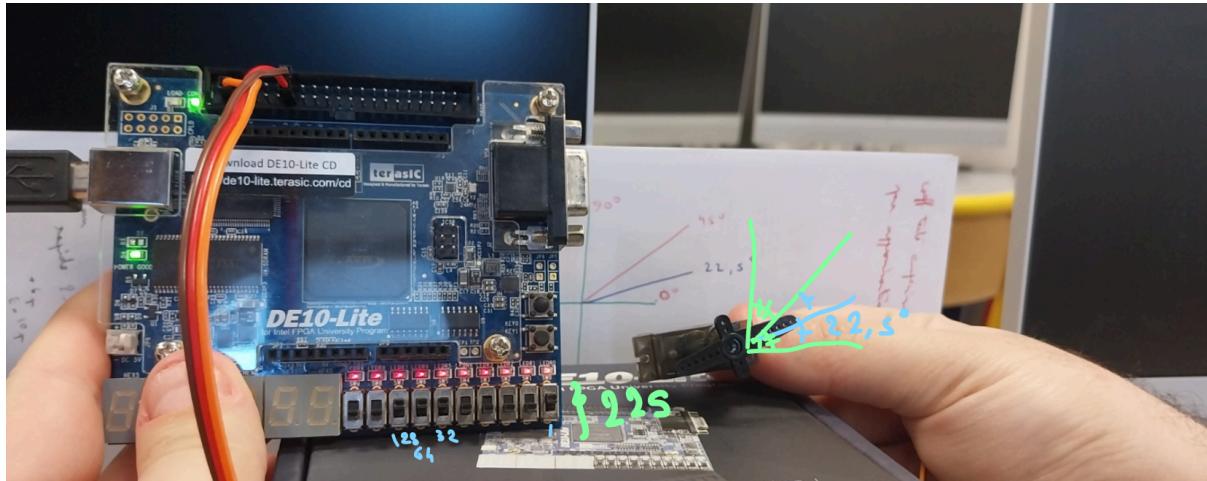
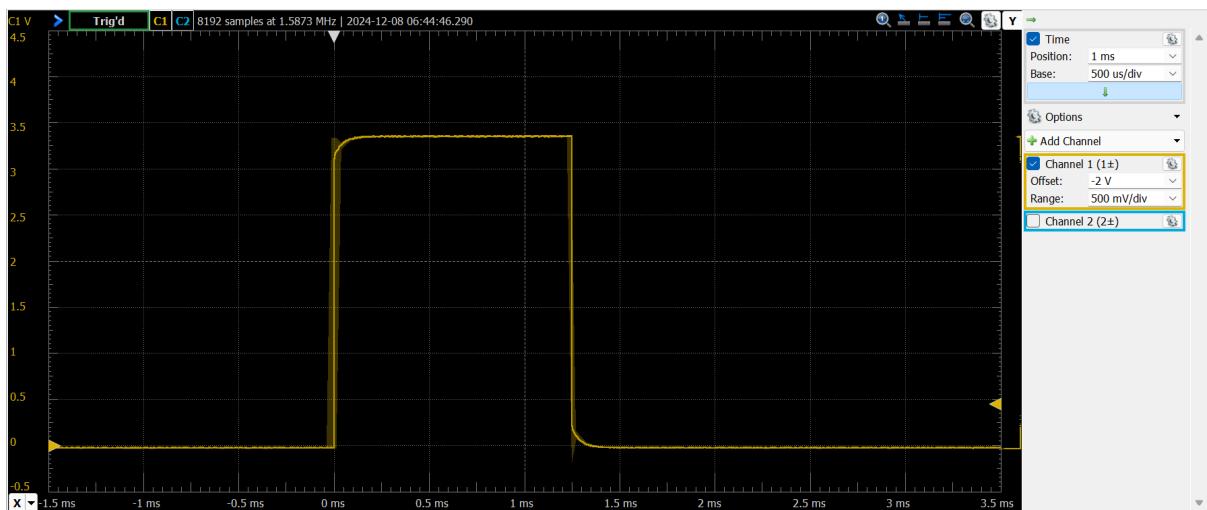


Figure 2.2.4.b : Mesure d'une impulsion haute de 1,25 ms sur l'oscilloscope



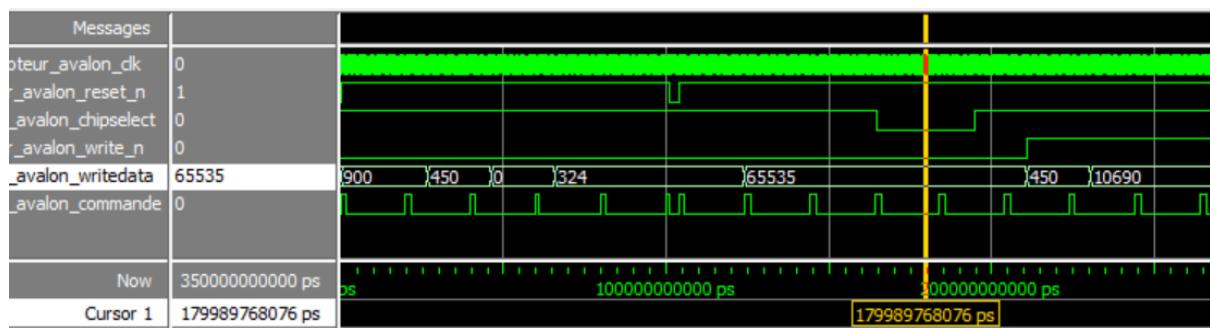
Enfin, un test de précision a été réalisé pour une position angulaire précise de 22,5°. L'oscilloscope a permis de mesurer une largeur d'impulsion de 1,25 ms, en accord avec les calculs théoriques. Ce test a démontré la capacité du système à générer des commandes précises, au dixième de degré près.

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



Nous commençons par effectuer les mêmes tests que précédemment, à savoir : le test sur trois valeurs particulières, afin de vérifier la bonne période de sortie de la commande ; le test sur le reset, qui réinitialise correctement la mesure ; et enfin, le test pour une valeur supérieure à 900, dont la période est identique à celle de 900, soit 2 ms, ce qui s'avère concluant dans tous les cas. Ensuite, dans un deuxième temps, nous testons les nouveautés liées à l'adaptation à l'interface Avalon. Ainsi, nous observons que lorsque le chipselect est à 0, la commande, continue d'être executée à la position précédente. Puis, lorsque celui-ci repasse à 1, la commande reprend toujours avec la même période qu'auparavant. Enfin, lorsque le write_n passe à 1, bien que le writedata soit modifié, la sortie reste inchangée. Ainsi notre modèle est validé.

2.4. Programmation logicielle sur carte FPGA du servomoteur

<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.

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.

Pour J'espère également que notre belle filière Ei, EiSE, Ei2I, 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.

Annexe