



POLYTECH SORBONNE UNIVERSITÉ

ARCHITECTURE DES SES MINI-PROJET À L'AIDE D'UN TÉLÉMÈTRE ULTRASON ET UN SERVOMOTEUR RAPPORT

Radar 2D avec DE10_Lite

Élève(s) :
Daniel FERREIRA LARA

Enseignant(s) :
Yann DOUZE

Table des matières

1	Introduction	2
2	Télémètre HC SR04	2
2.1	IP Standalone	2
2.2	IP Avalon	3
2.3	Demonstration	4
2.4	Programmation sur le NIOS II	6
3	Servomoteur MG90S	7
3.1	IP Standalone	8
3.2	IP Avalon	9
3.3	Demo sur le NIOS II	11
4	Montage Servo-Télémètre	13
5	Tracé radar sur écran VGA	15
6	Développement de l'IP UART	17
6.1	IP Standalone	18
6.2	IP Avalon	19
7	Développement de l'IP Neopixel 12 LEDs	20
7.1	IP Standalone	20
7.2	IP Avalon	23
7.3	Demo NIOS II	24
8	Montage Final	25
9	Conclusion	26

1 Introduction

Ce projet d'architecture de systèmes embarqués est un radar rotatif 2D permettant de reconnaître le milieu environnant. Ce type de radar est très utile dans les applications de balayage et fonctionne comme un sonar dans une implémentation simplifiée.

Tous les éléments proposés ont été mis en œuvre et le présent rapport les explique de manière suffisamment détaillée sans tomber dans l'excès. J'espère que vous prendrez plaisir à le lire et voici les liens importants :

1. **Repo GitHub** : [DE10_Lite Radar 2D](#)
2. **Démonstration vidéo** : [URL vidéo](#)

Par ailleurs, je me tiens à votre disposition pour répondre à vos questions sur ce travail dans le mail daniel.ferreira_lara@etu.sorbonne-universite.fr et vous remercie d'avance pour la lecture de ce rapport.

2 Télémètre HC SR04

Le télémètre est un capteur à ultrasons qui peut mesurer des distances en faisant rebondir l'onde sonore sur des objets. Le modèle choisi est largement utilisé dans les projets de prototypage pour détecter les obstacles pour les robots suiveurs de ligne, détecter la présence ou mesurer les distances par rapport aux objets environnants.

Avant de penser à la logique à l'intérieur de DE10_lite, il est important de comprendre comment fonctionne le capteur, afin de pouvoir calculer la distance de l'objet par le temps de réception du signal de retour à partir de la moitié du temps passé, c'est-à-dire l'aller ou le retour.

Le capteur fonctionne à 5 V et peut mesurer des distances comprises entre 4 et 400 cm avec une certaine précision. Pour commencer la mesure, nous devons envoyer le déclencheur pendant 10 s et attendre un écho. Après 60 ms, si nous n'obtenons pas d'écho, nous recommençons.

La vitesse du temps étant $v = 343 \text{ m/s}$, et en ne considérant qu'un seul itinéraire, nous pouvons calculer la distance en cm comme suit :

$$d = \frac{v_{\text{son}} \cdot t}{2} \approx 170 \cdot t [\text{m}],$$

2.1 IP Standalone

Sur la base du calcul ci-dessus, nous pouvons utiliser la fréquence de l'horloge pour définir un compteur capable de traiter les données du capteur. Pour toutes les IP réalisées en Standalone (sans Avalon), nous utilisons une fréquence de 50 MHz, la période est donc de 20 ns. Cette valeur est importante car nous devons calculer la distance sur la base du compteur.

$$t_{1cm} = 5.88 \times 10^{-5} \text{ s}, \text{ si on a } 20 \text{ ns de period, } \frac{t_{1cm}}{t_{\text{clk}}} = 2941, \text{ c'est le facteur de division.}$$

Il convient de commenter les deux bascules echo_r et echo_rr, qui sont utilisées pour que le bloc combinatoire de calcul de la distance ne pose pas de problème. Cinq états ont été définis :

1. Idle : État initial pour effacer les données
2. S1 : État pour l'envoi de 12 µs sur TRIG (compteur 600)
3. S2 : État permettant d'identifier le début de la réception par écho
4. S3 : Etat pour décompter le temps d'écho et envoyer la réponse sur le signal Dist_cm
5. S4 : Etat d'attente de 60 ms pour recommencer (compteur 3 000 000)

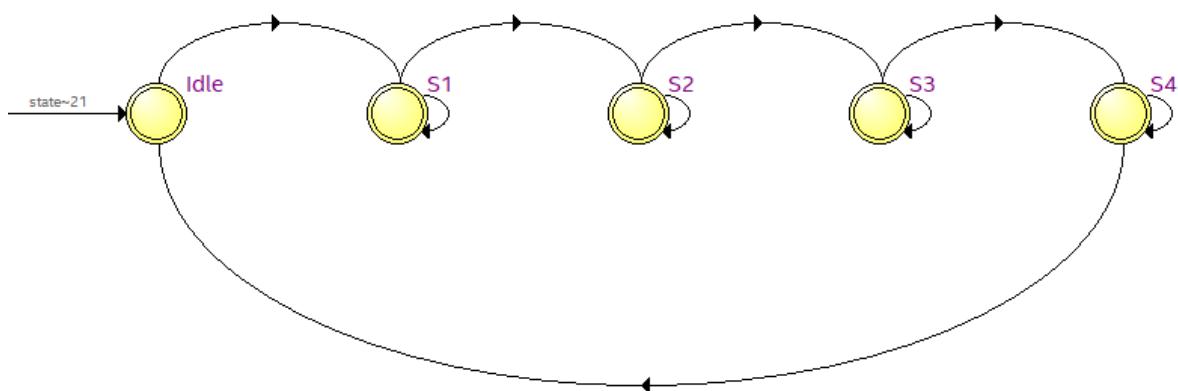


FIGURE 1 – Machine à états (FSM)

Voici la simulation en modelsim :



FIGURE 2 – Modelsim IP telemetre

Dans l'image ci-dessus, vous pouvez voir les signaux TRIG et ECHO dans une simulation de 34cm, 85cm et 170cm, respectivement 2 ms, 5 ms et 8 ms.

2.2 IP Avalon

Pour la deuxième partie, l'intégration du module au bus Avalon, il suffit d'ajouter quelques modifications, qui sont un facteur (FREQ_DIV) dans le compteur et divisant la distance par lui aussi, puisque la fréquence est maintenant de 100 MHz. Dans ce cas, nous avons deux fois plus de cycles dans le même temps, et ce facteur a été multiplié dans les conditions d'arrêt pour résoudre le problème.

$t_{1cm} = 5.88 \times 10^{-5} s$, si on a 10 ns de period, $\frac{t_{1cm}}{t_{clk}} = 2941 \times 2 = 5882$, c'est le facteur de division.

J'ai également ajouté les signaux read_n, readdata et chipselect, qui sont responsables de la connexion au bus Avalon du SoC NIOS II. Les GPIO suivants ont été choisis pour TRIG et ECHO :

Signel	GPIO	Nom du PIN
TRIG	3	PIN_W9
ECHO	1	PIN_W10

Voici la simulation en modelsim :



FIGURE 3 – Modelsim 34 cm Avalon

Dans ce cas, nous avons simplement envoyé le test à 2 ms, et il a fonctionné comme le premier, mais les signaux d'Avalon montrent maintenant qu'il fonctionne bien.

2.3 Demonstration

Pour la démonstration, j'ai pris une règle de 1 mètre et je l'ai collée au mur. Voyons les photos à différentes distances, le test a été fait sur la version Avalon, car elle montre à la fois les LEDs et les 7 segments.

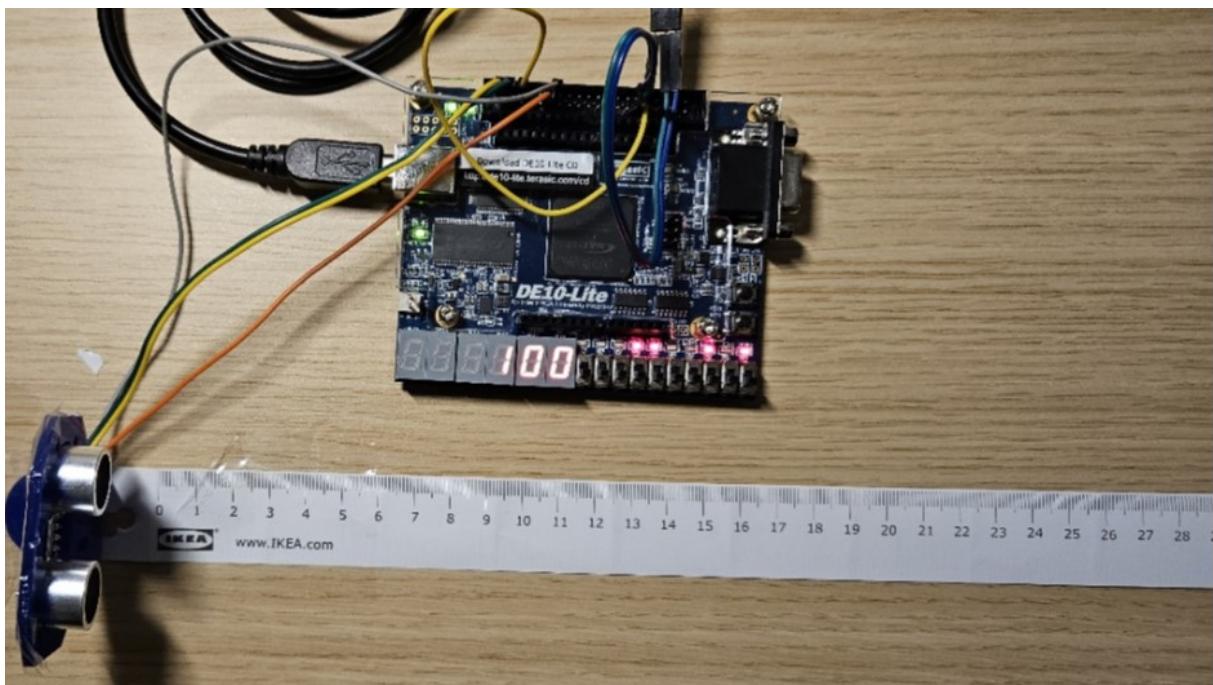


FIGURE 4 – Distance 1 m

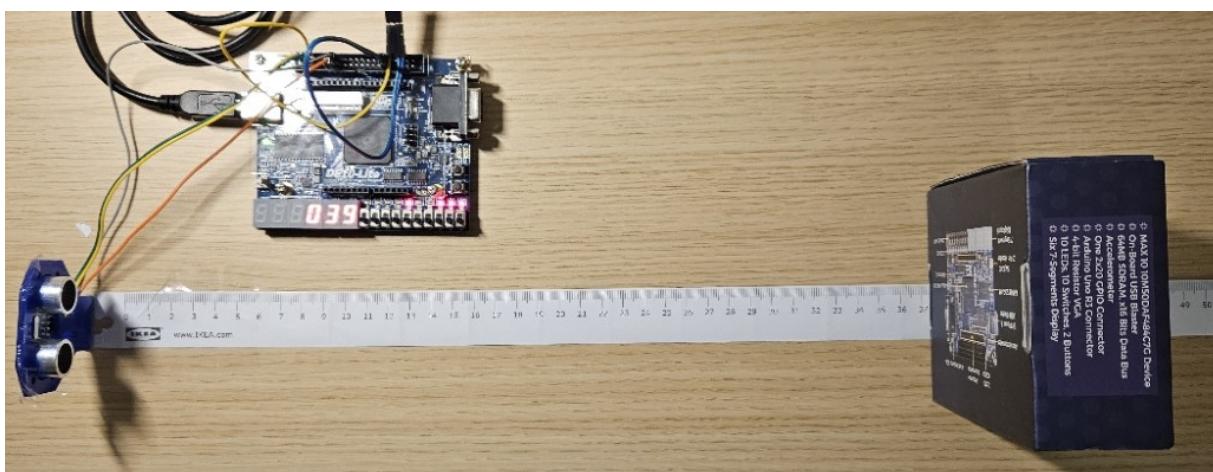


FIGURE 5 – Distance 37,5 cm

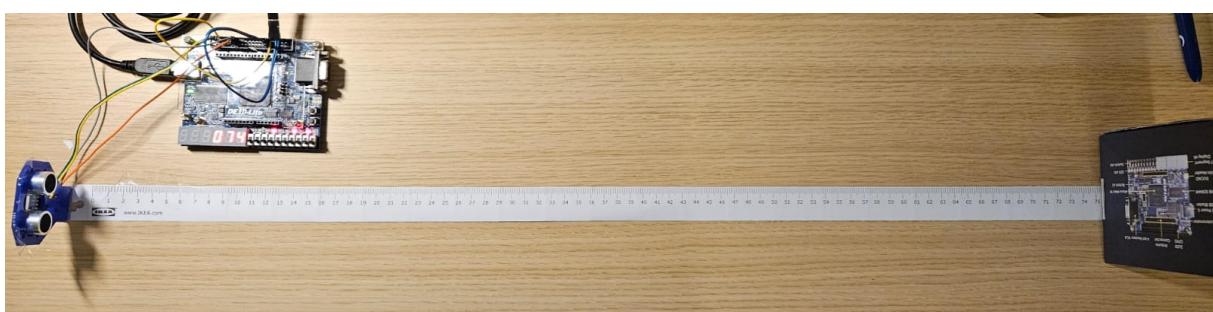


FIGURE 6 – Distance 75,5 cm

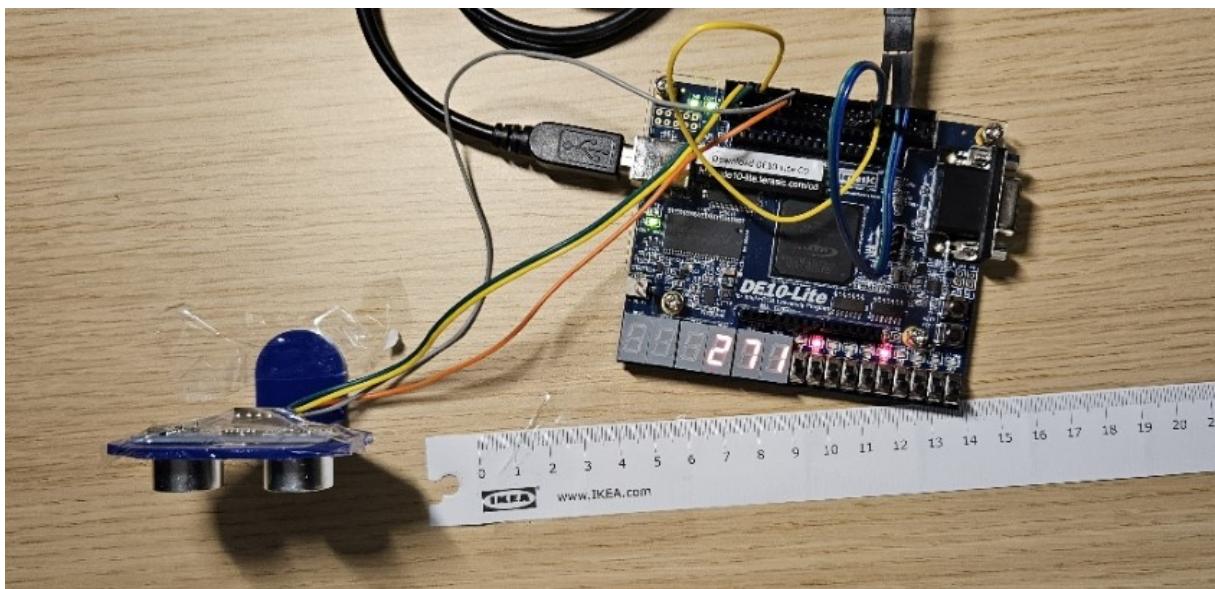


FIGURE 7 – Distance au mur du chambre : 271 cm

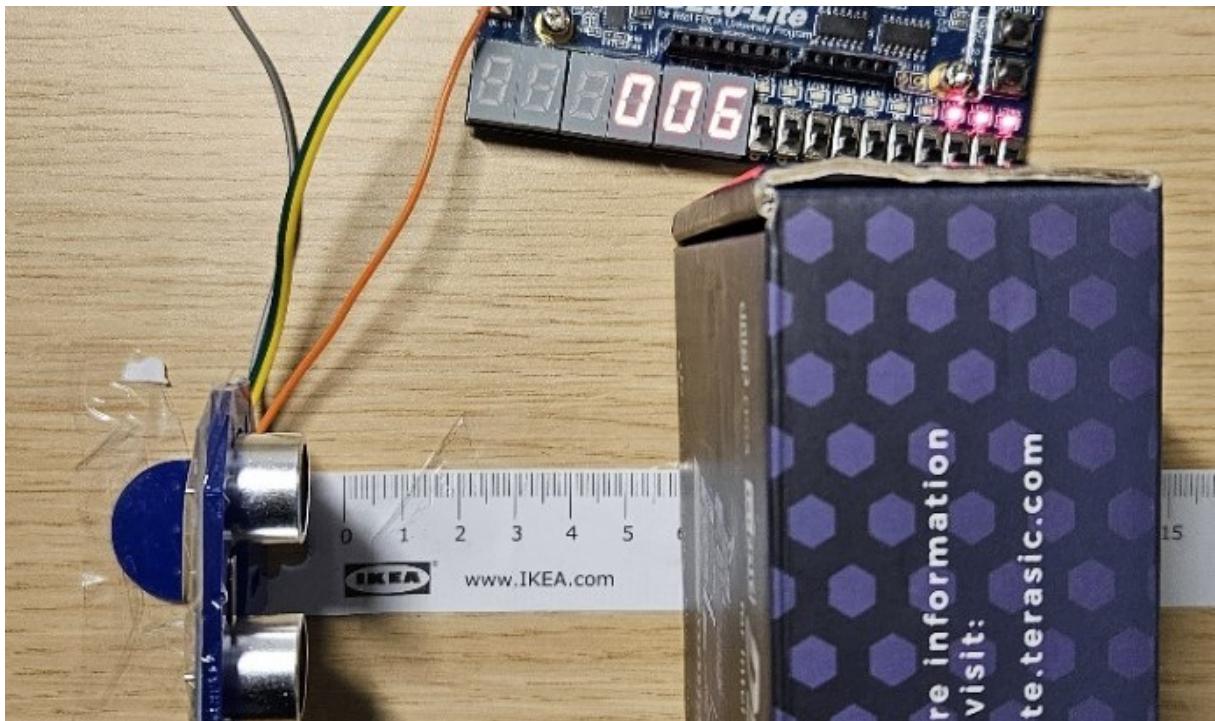


FIGURE 8 – Distance 6 cm

Constatons donc que, malgré une petite erreur de calcul de la distance, inférieure à 2 cm dans les tests présentés, il est très efficace.

2.4 Programmation sur le NIOS II

Jusqu'à présent, différents tests ont été effectués et les démonstrations précédentes étaient le résultat d'un code implémenté dans l'IDE Eclipse. Pour m'assurer que les don-

nées affichées sur l'écran à sept segments concordent, je les ai imprimées sur le terminal NIOS II.



```

Problems Tasks Console Nios II Console ✎ Properties
Telemetre_7_seg Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: JTAG_UART.jtag
Value: 119 cm
Value: 113 cm
Value: 108 cm
Value: 98 cm
Value: 94 cm
Value: 116 cm
Value: 80 cm
Value: 74 cm
Value: 70 cm
Value: 62 cm
Value: 58 cm
Value: 51 cm
Value: 41 cm
Value: 34 cm
Value: 28 cm
Value: 156 cm
Value: 46 cm
Value: 8 cm
Value: 32 cm
Value: 171 cm
Value: 196 cm

```

FIGURE 9 – Simulation sur terminal

3 Servomoteur MG90S

La deuxième propriété intellectuelle développée concernait la rotation du servomoteur et le balayage du radar. Le servomoteur est conçu pour fonctionner à 90 degrés d'amplitude, il est contrôlé par un PWM avec une période entre 1 (0°) et 2 ms (90°), cependant c'est la fonction que je voudrais, donc j'ai étudié les limites de l'appareil.

Après plusieurs essais pratiques, j'ai réussi à atteindre les limites de l'actionneur à un point où il n'était pas possible d'atteindre l'amplitude de 180° , mais j'en étais très proche. Voici ce que j'ai obtenu :

$$0^\circ = 0.8 \text{ ms}, \text{ et } 170^\circ = 2.6 \text{ ms}$$

La fonction qui régit le calcul du temps du PWM actif est la suivante :

$$T_{PWM_On} = (80.000 + \frac{(260.000 - 80.000) \cdot position}{180}) \cdot \frac{1}{Freq_Div},$$

$$T_{PWM_On} = 80.000 + 100 \cdot position \cdot \frac{1}{Freq_Div},$$

$$\text{soit } \frac{position}{Freq_Div} \leq 900$$

Le calcul du temps PWM actif augmente progressivement au fur et à mesure que la valeur de la position augmente, avec une résolution de 0.2° de mouvement.

3.1 IP Standalone

Dans cette implémentation, j'utilise une résolution de 10 bits lors du traitement des données, j'ai donc 1024 états possibles et pour les besoins des calculs approximatifs, j'ai considéré 180° d'amplitude. La mise en œuvre étant très simple, je vais passer directement aux démonstrations, voyons les résultats obtenus.

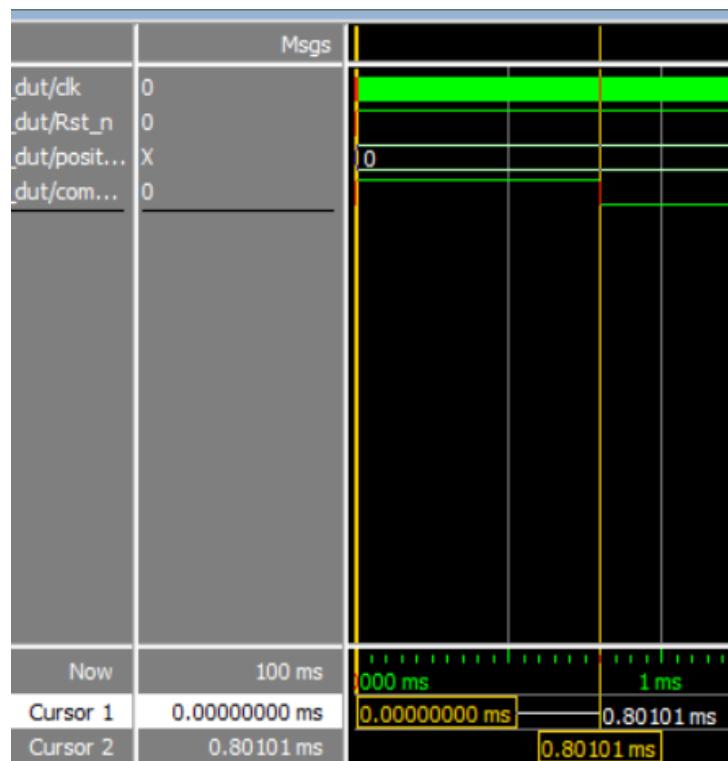


FIGURE 10 – PWM 0 degré

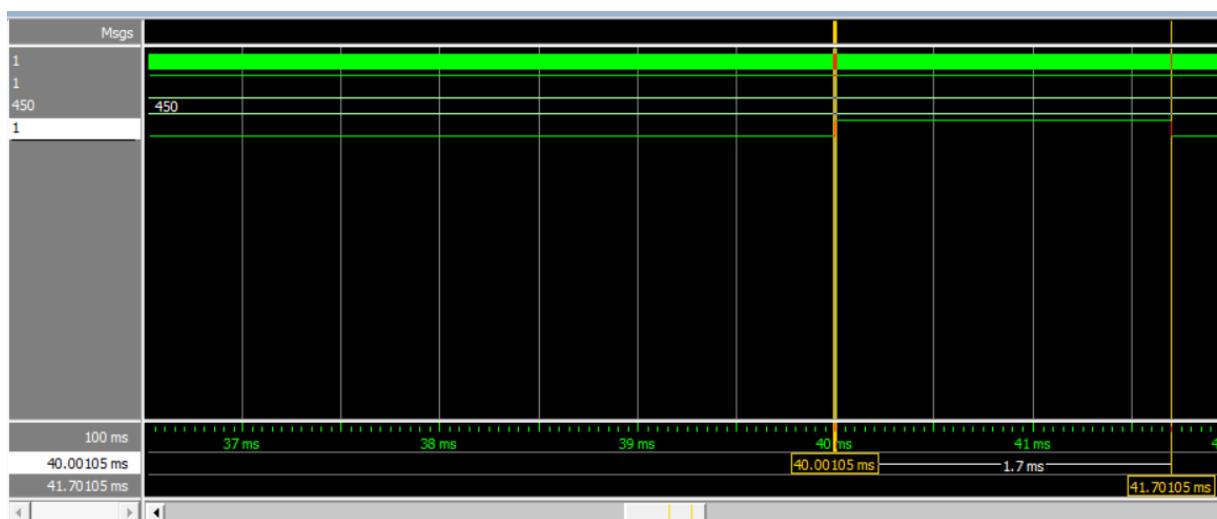


FIGURE 11 – PWM 90 degrés

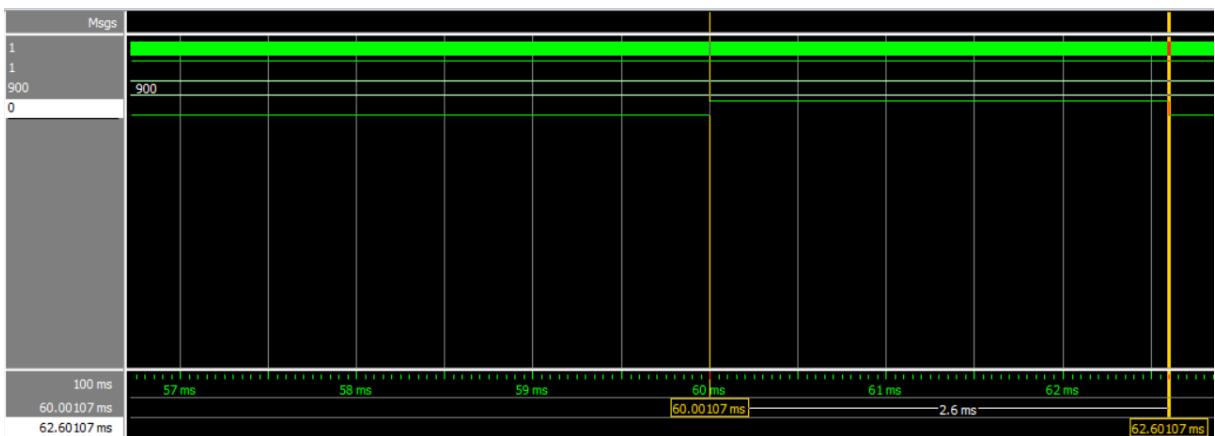


FIGURE 12 – PWM 180 degrés

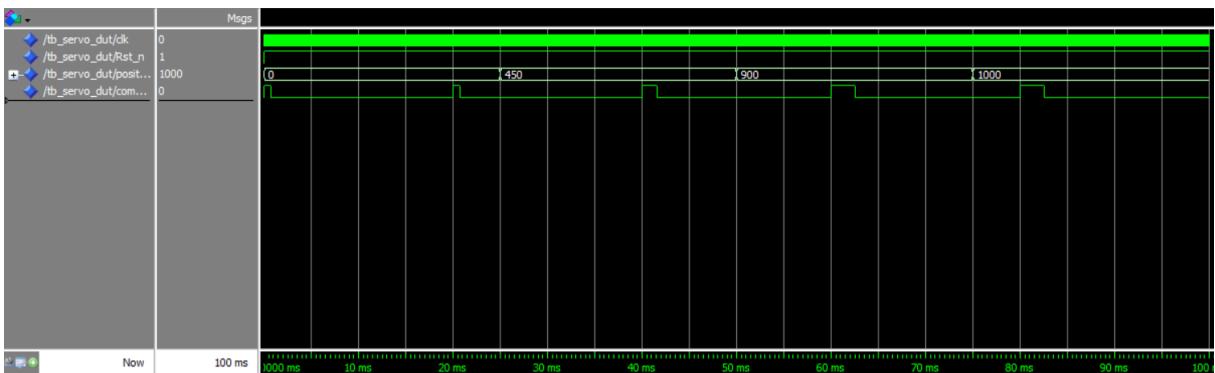


FIGURE 13 – Simu complet

On constate que la position au-delà de 900 est limitée à un maximum de 900.

3.2 IP Avalon

Maintenant, pour l'implémentation du bus Avalon, j'ai dû le modifier un peu, principalement le nombre de cycles a été doublé. En outre, comme nous disposons d'une plus grande résolution, j'ai placé la position entre 0 et 1800, pour avoir une résolution de 0.1. Les signaux Avalon ont été ajoutés chipselect, write_n, WriteData. Maintenant, le GPIO utilisé pour le servo était :

Signel	GPIO	Nom du PIN
Commande	5	PIN_W8

Voici les simulations du bus avalon sur modelsim :

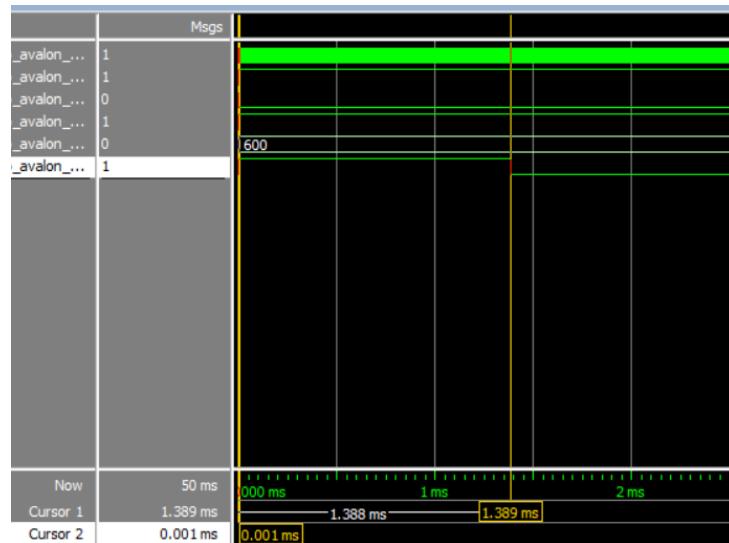


FIGURE 14 – PWM 60 degrés

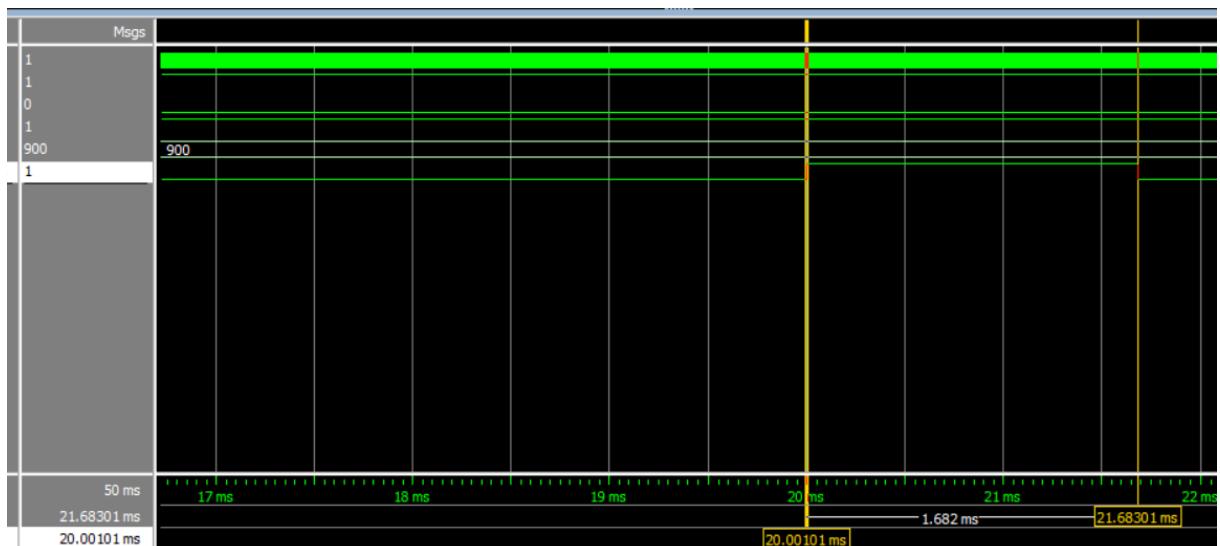


FIGURE 15 – PWM 90 degrés

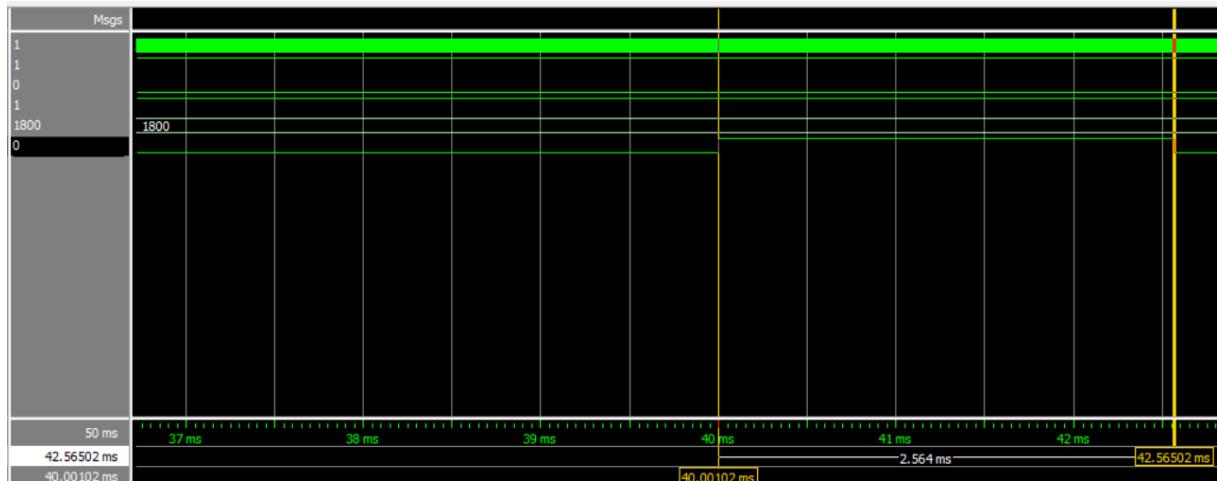


FIGURE 16 – PWM 180 degrés

3.3 Demo sur le NIOS II

En créant un code dans NIOS II, j'ai pu l'afficher sur l'écran à 7 segments et nous pouvons également le voir sur le terminal. Vous pouvez voir sur les photos les angles presque complémentaires entre le premier et le dernier.

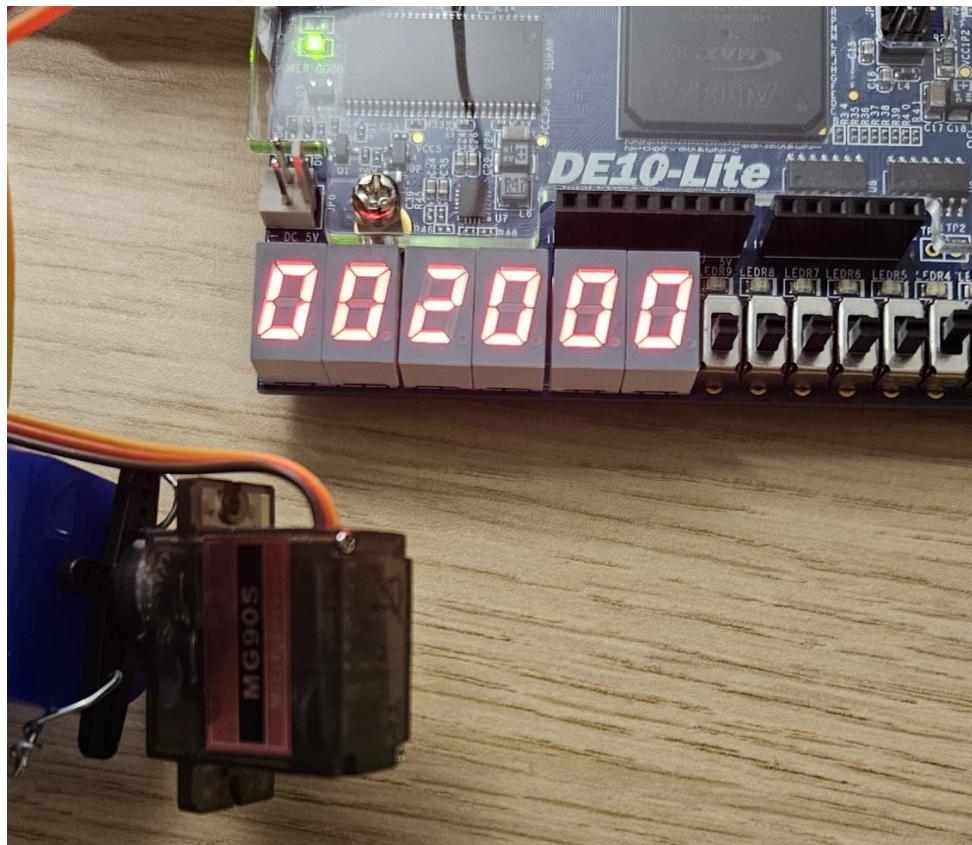


FIGURE 17 – 2 degrés

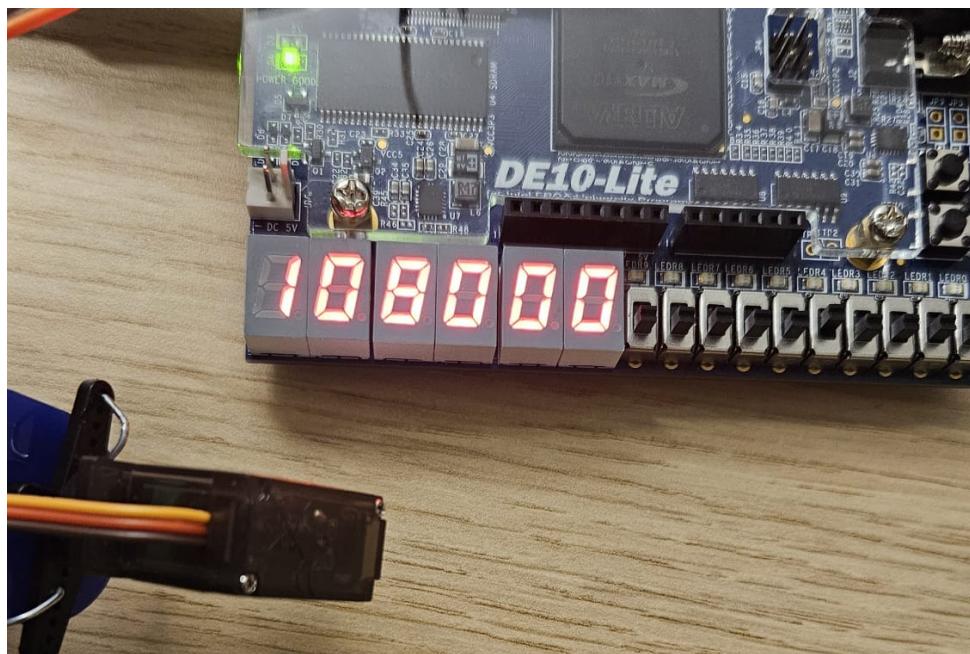


FIGURE 18 – 108 degrés

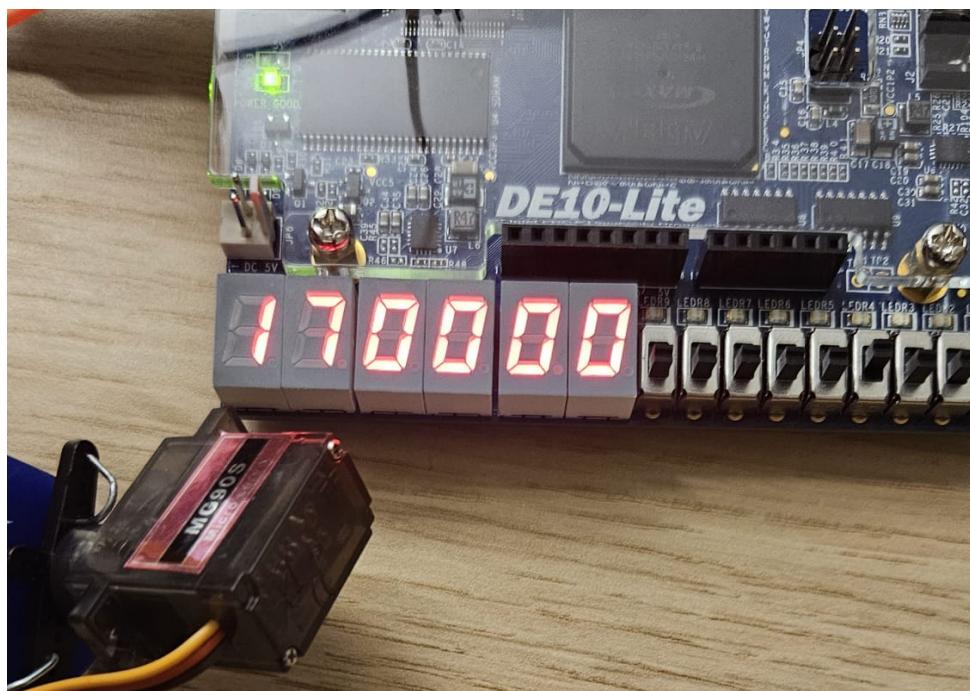
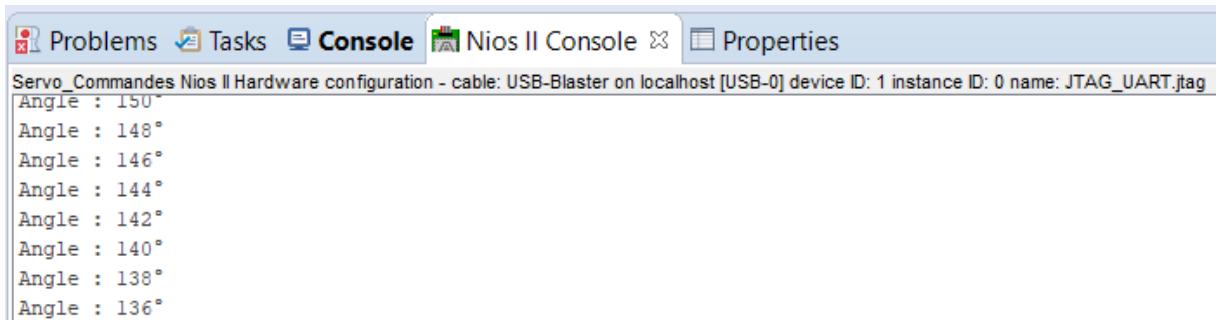


FIGURE 19 – 170 degrés



```
Problems Tasks Console Nios II Console ✎ Properties
Servo_Commandes Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: JTAG_UART.jtag
Angle : 150°
Angle : 148°
Angle : 146°
Angle : 144°
Angle : 142°
Angle : 140°
Angle : 138°
Angle : 136°
```

FIGURE 20 – Terminal

4 Montage Servo-Télémètre

Pour cette troisième étape, les IP ont déjà été dûment ajoutées au Platform Designer et configurées sur le bus Avalon du SoC NIOS II. Il ne nous reste plus qu'à traiter le code C pour relier les deux implémentations et nous aurons un radar fonctionnel. Voir les GPIO utilisés :

Signel	GPIO	Nom du PIN
ECHO	1	PIN_W10
TRIG	3	PIN_W9
Commande	5	PIN_W8

0° -> 205 cm	96° -> 22 cm
6° -> 49 cm	102° -> 23 cm
12° -> 47 cm	108° -> 23 cm
18° -> 208 cm	114° -> 23 cm
24° -> 206 cm	120° -> 25 cm
30° -> 205 cm	126° -> 107 cm
36° -> 51 cm	132° -> 17 cm
42° -> 49 cm	138° -> 14 cm
48° -> 986 cm	144° -> 14 cm
54° -> 985 cm	150° -> 13 cm
60° -> 985 cm	156° -> 12 cm
66° -> 985 cm	162° -> 13 cm
72° -> 985 cm	168° -> 13 cm
78° -> 23 cm	174° -> 13 cm
84° -> 22 cm	180° -> 14 cm
90° -> 22 cm	

FIGURE 21 – La sortie du Terminal avec pas 4 degrés

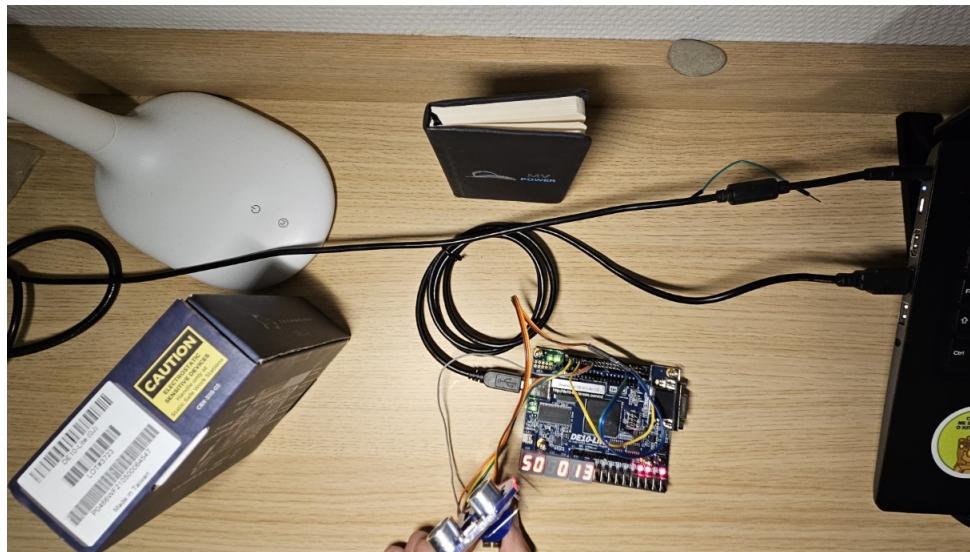


FIGURE 22 – 50 degrés, 13 cm

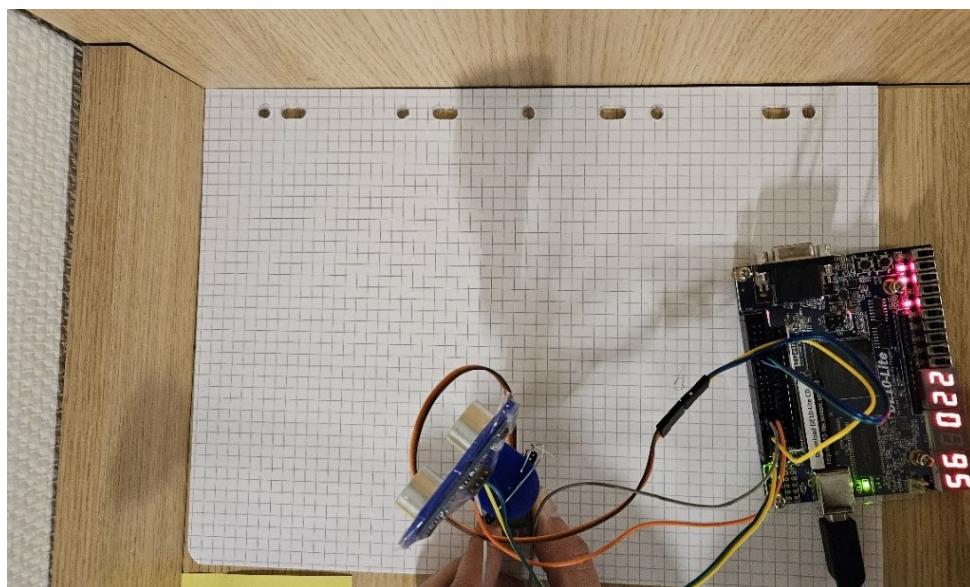


FIGURE 23 – 56 degrés, 22 cm

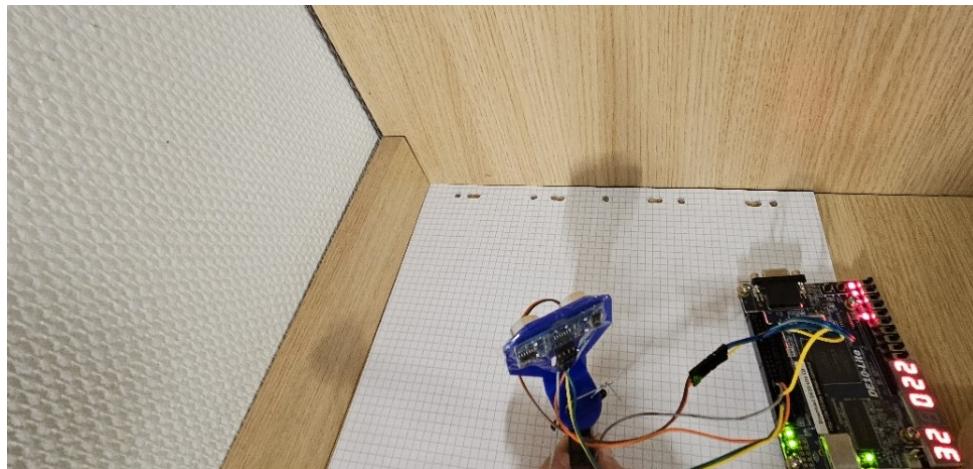


FIGURE 24 – 32 degrés, 22 cm

Il convient de souligner la facilité avec laquelle il a été possible d'intégrer les IP développées et de développer le code pour le SoC. Par conséquent, je pense que la compréhension du processus de création du système de base 2D Radar avec un télémètre a été une grande expérience d'apprentissage. Nous allons maintenant passer à la mise en œuvre des autres fonctionnalités proposées.

5 Tracé radar sur écran VGA

Sur la base du code disponible sur le [*DE10-Lite Computer System User Manual*](#) [1], j'ai pu le tester avec un simple rectangle à l'écran, comme indiqué ci-dessous :

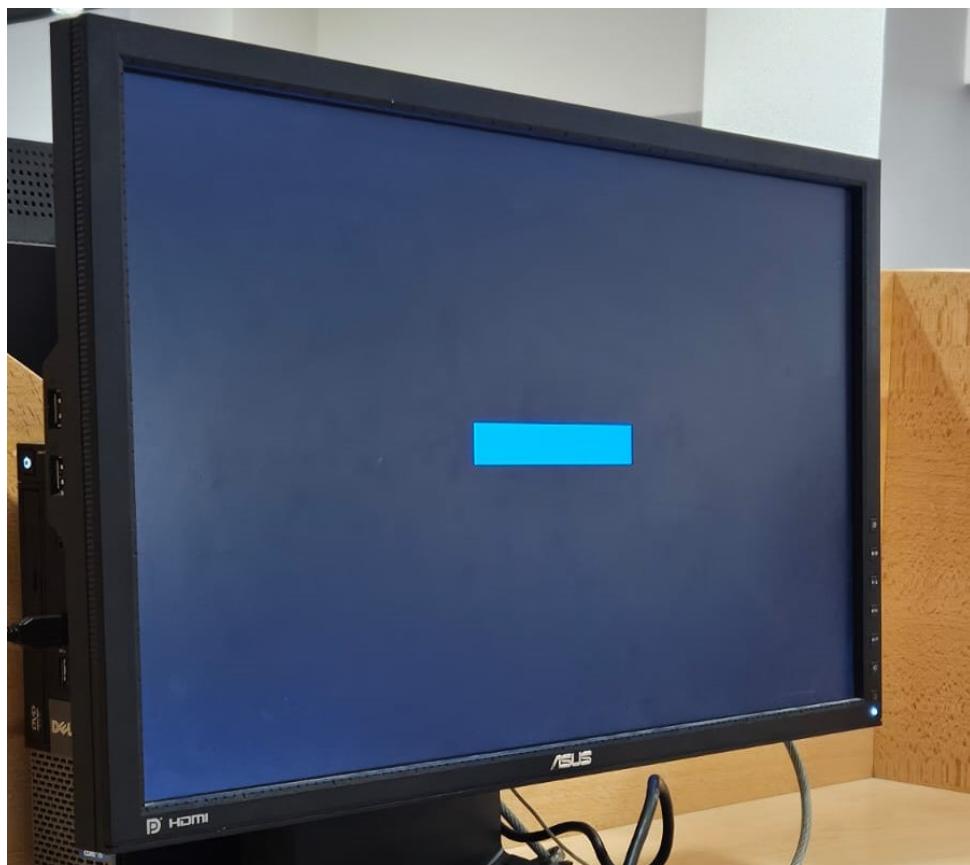


FIGURE 25 – Test initiale

Après ce premier test, j'ai créé une fonction qui trace un vecteur sous la forme du radar suivant :

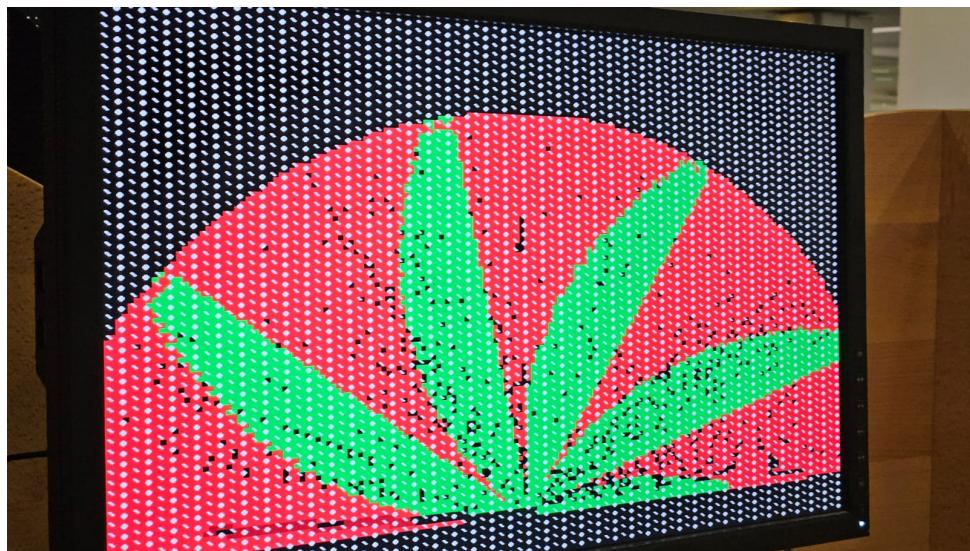


FIGURE 26 – Test du radar sans le télémètre

J'ai ensuite effectué un test avec une distance maximale de 400 cm, cette limite ne fonctionne pas bien lorsque les objets sont proches.

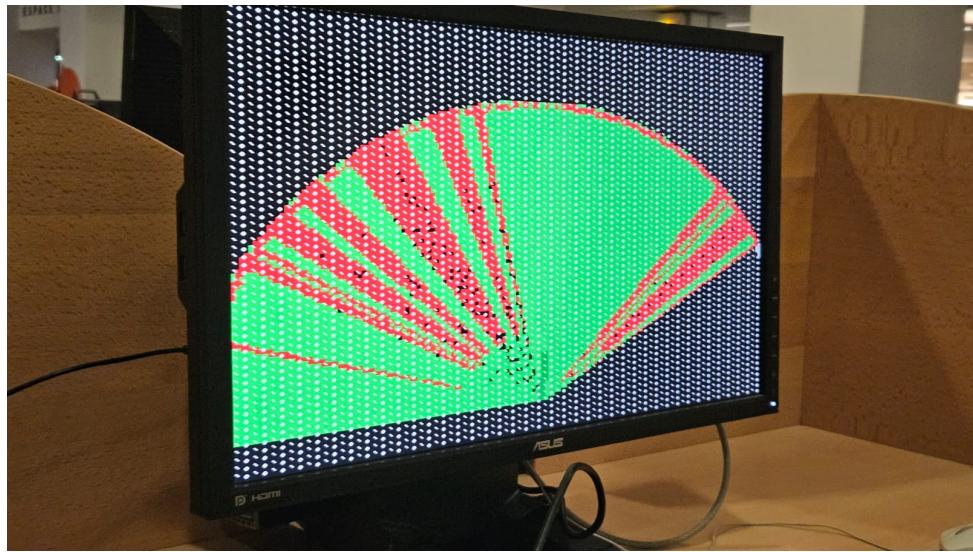


FIGURE 27 – Test du radar avec le télémètre et servo

Les points blancs dans l'image ci-dessus étaient un problème qui a été corrigé dans la version suivante :

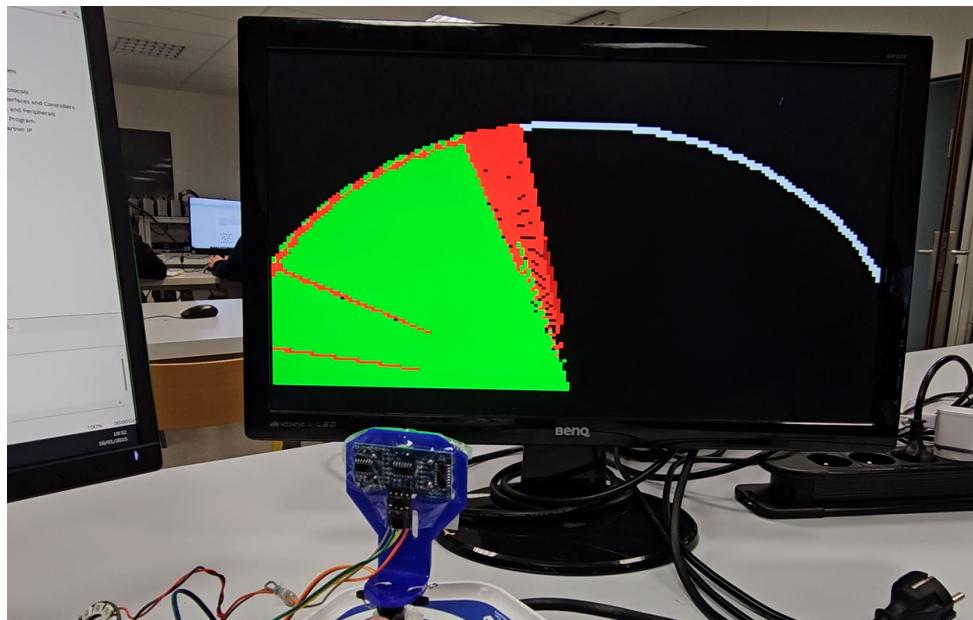


FIGURE 28 – Test du radar avec le télémètre et servo

Une vidéo a été faite démontrant le fonctionnement des servos fonctionnant avec l'affichage sur l'écran VGA, elle se trouve dans la section 8.

6 Développement de l'IP UART

Pour le module UART, j'ai essayé d'utiliser une approche purement logicielle au-dessus du SoC, mais j'ai rencontré plusieurs problèmes parce que le pont JTAG_UART est utilisé pour le terminal NIOS II et ne permet donc pas l'accès à d'autres systèmes. Lorsque j'ai

essayé d'utiliser les fonctions existantes `get_jtag` et `put_jtag`, elles ne peuvent pas gérer de nombreux caractères, logiquement, comme cela a déjà été expliqué par le partage du pont.

L'idée était donc d'utiliser un convertisseur série-USB pour créer un nouveau pont de connexion basé sur 2 autres GPIOs, Tx/Rx. J'ai dû développer l'IP nécessaire pour cela, en utilisant des sources internet comme les forums FPGA et aussi des LLMs comme [Mistral AI](#) pour comprendre les erreurs. Les GPIO choisis pour Tx et Rx sont les suivants :

Signel	GPIO	Nom du PIN
Rx	0	PIN_V10
Tx	2	PIN_V9

6.1 IP Standalone

L'IP autonome utilise SW pour sélectionner l'octet de données à envoyer et le bouton KEY1 pour quitter via GPIO.

Ci-dessous, la simulation Modelsim montrant l'envoi de données par le DE10Lite et la simulation de la réception de l'octet correspondant à la lettre L.

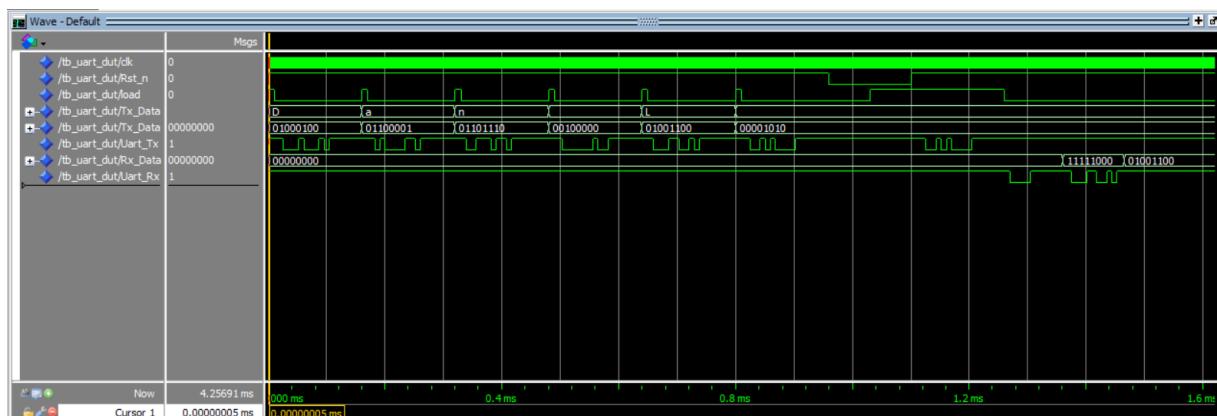


FIGURE 29 – Envoi de "Dan L /n"

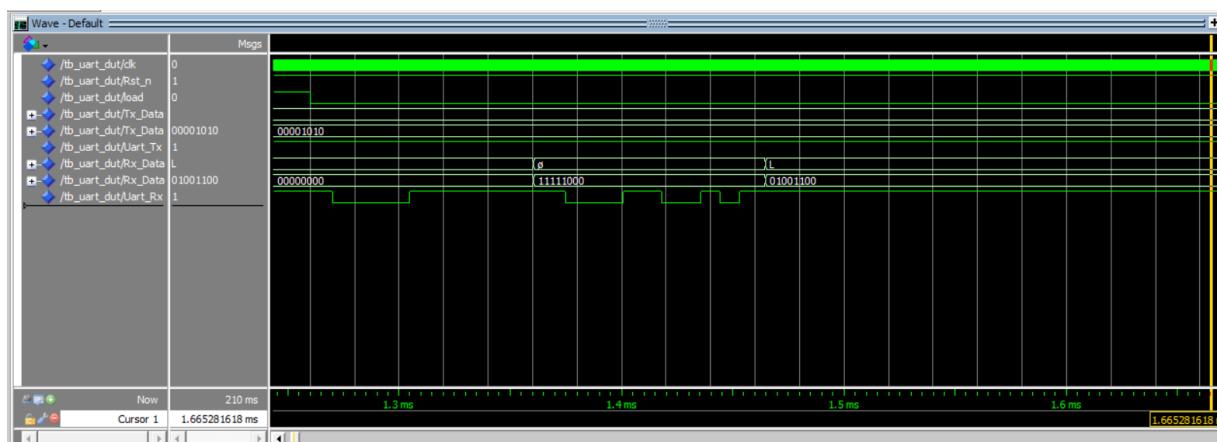


FIGURE 30 – Réception de L

Lors de l'attachement via Pin planner, j'ai pu recevoir le message sur l'ordinateur.

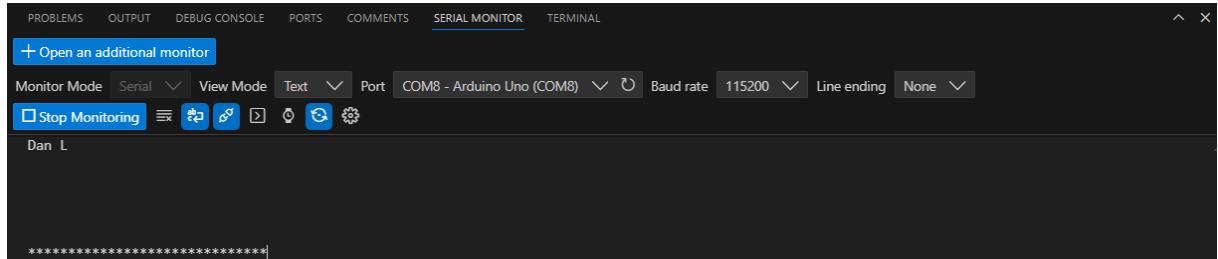


FIGURE 31 – Envoye par switch

6.2 IP Avalon

En ajoutant les informations requises au bus Avalon, j'ai également simulé l'envoi de la même information, nous pouvons voir que nous utilisons maintenant WriteData pour envoyer la même information. Regardez la deuxième image, qui se concentre sur le début de l'envoi des lettres :

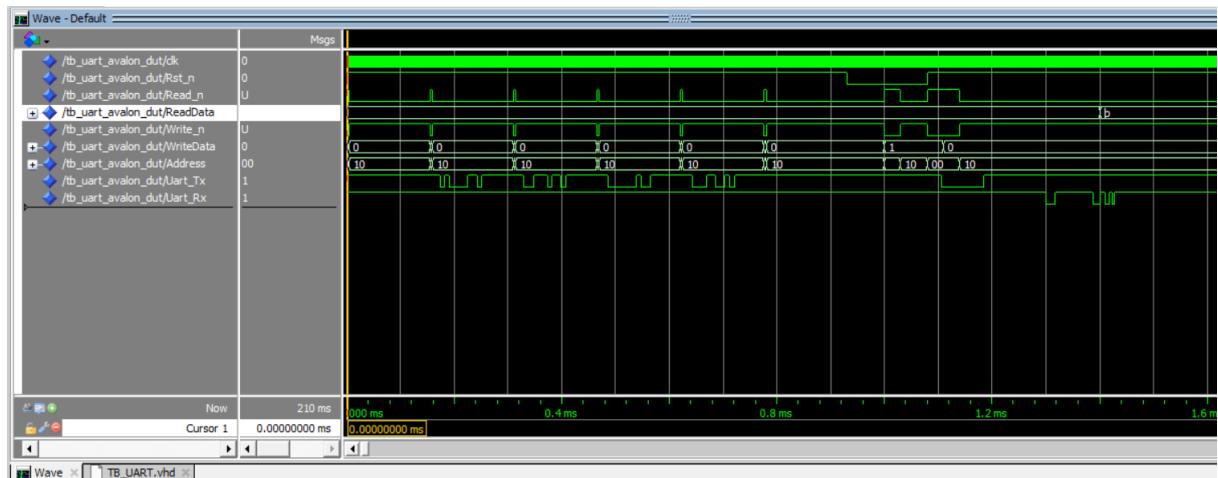


FIGURE 32 – Envoi de "Dan L \n"

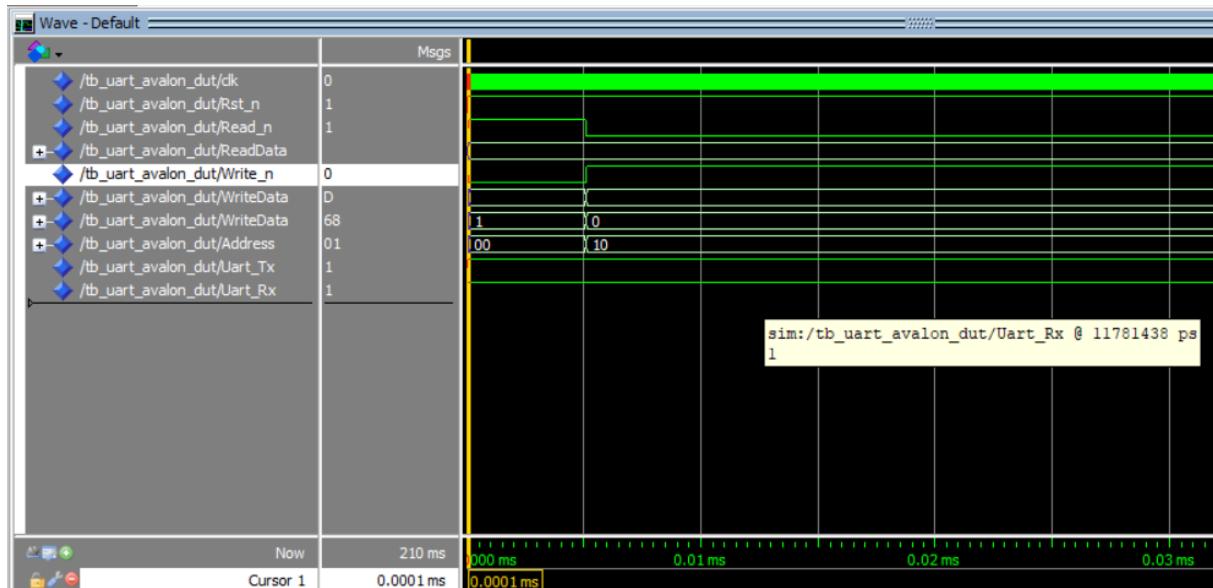


FIGURE 33 – Début

7 Développement de l'IP Neopixel 12 LEDs

La dernière propriété intellectuelle à développer est la gestion de l'anneau NeoPixel 12 LED. Pour ce module, j'ai donc utilisé les sources [WS2812 Intelligent Control LED Datasheet \[2\]](#), [Adafruit NeoPixel Uberguide \[4\]](#) et [Power Analysis WS2812 \[3\]](#), ainsi que des discussions sur le fonctionnement avec un camarade de classe, Balayan.

Parmi les possibilités de mise en œuvre, j'ai pensé à faire quelque chose comme le fait déjà le radar, en indiquant par une couleur s'il y a ou non un objet devant, mais l'idée mise en œuvre utilise un concept légèrement différent, car j'ai déjà tracé le radar deux fois, de sorte que la proposition ici est que plus l'angle augmente, plus les LED s'allument. Il sera possible de choisir la couleur de la led.

7.1 IP Standalone

Sur la base de l'idée définie ci-dessus, étant donné que dans ce cas nous n'avons que 10 bits de résolution maximale, j'ai développé la logique d'envoi des données en fonction de l'entrée des 4 interrupteurs la plus proche de 0.

Voir ci-dessous le port choisi pour gérer les données du Neopixel ring :

Signel	GPIO	Nom du PIN
commande_led	9	PIN_V5
Write_led	SW0-SW3	Voir Doc
color_led	SW4-SW9	Voir Doc

Par défaut, chaque LED a une couleur prédéfinie, celle-ci est utilisée pour bien identifier chacune d'entre elles. En plus, j'ai créé une fonctionnalité qui, avec 6 bits de résolution, nous permet de définir la même couleur pour toutes les LED (GGRRBB), la sélection étant faite par les 6 autres interrupteurs sur les 10. Exemple d'une couleur comme le

jaune : 010100(GGRRBB), dans ce cas nous avons du jaune à faible luminosité. Si nous avions 111100, nous aurions toujours du jaune et à forte luminosité.

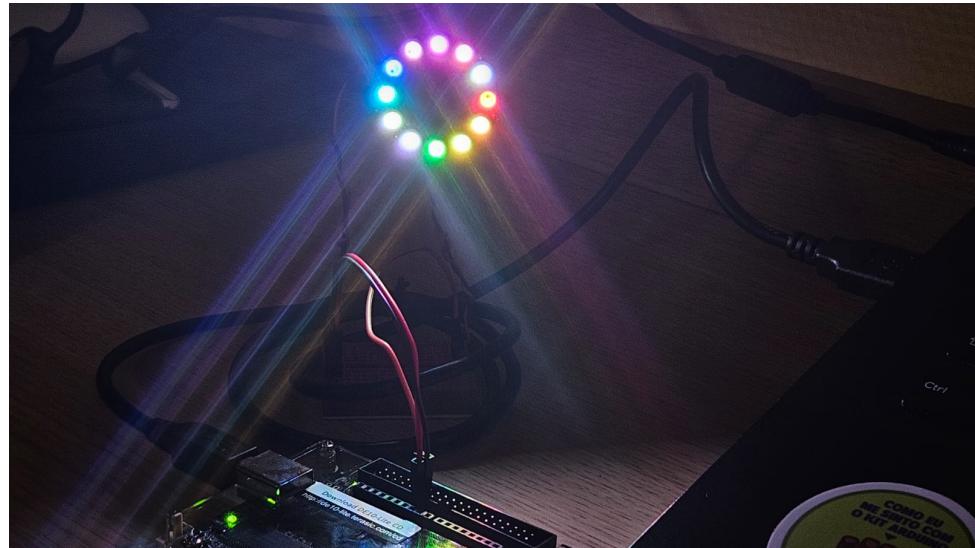


FIGURE 34 – Toutes les Leds, avec coleurs differents

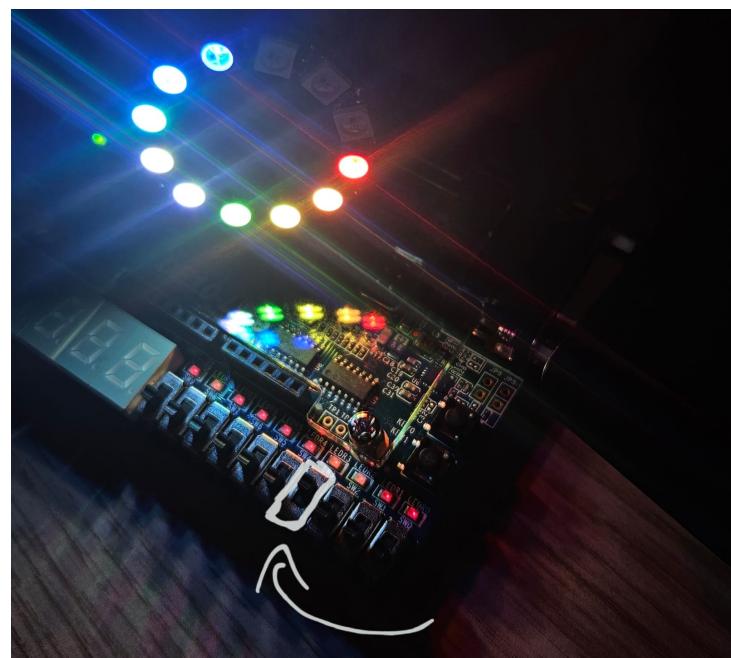


FIGURE 35 – 8 LEDS, choisi 1000

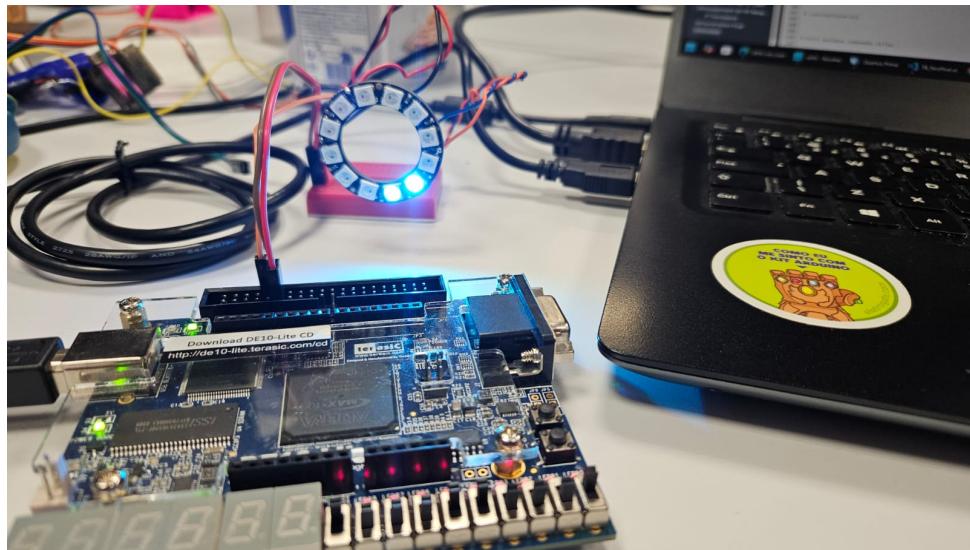


FIGURE 36 – 1 LED, choisi 100011_0001

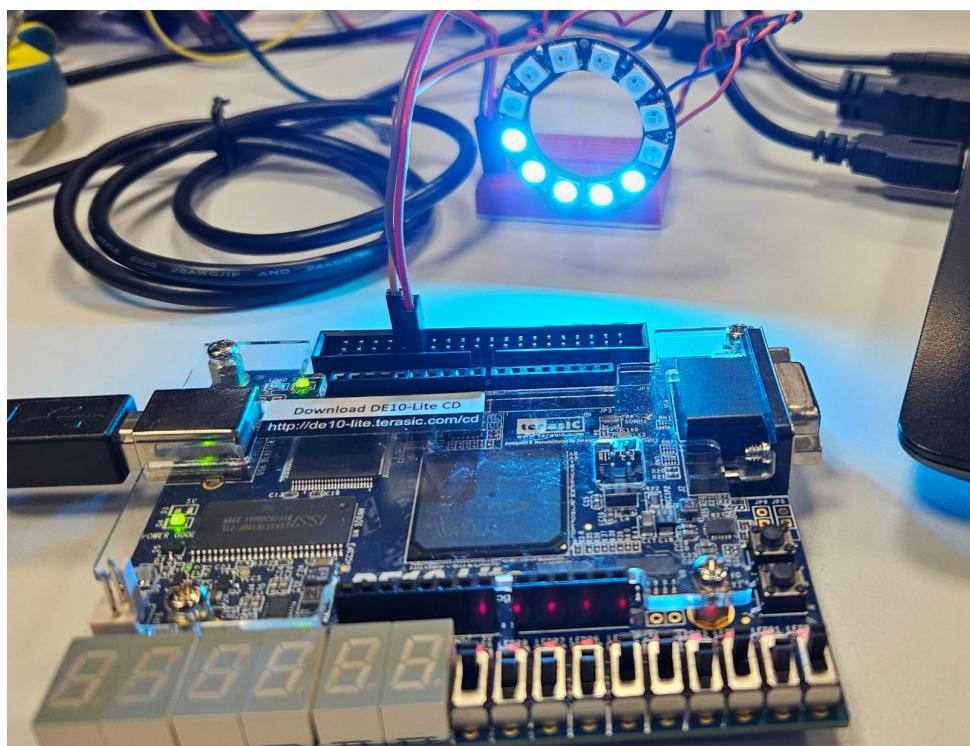


FIGURE 37 – 5 LEDs, choisi 100011_0101

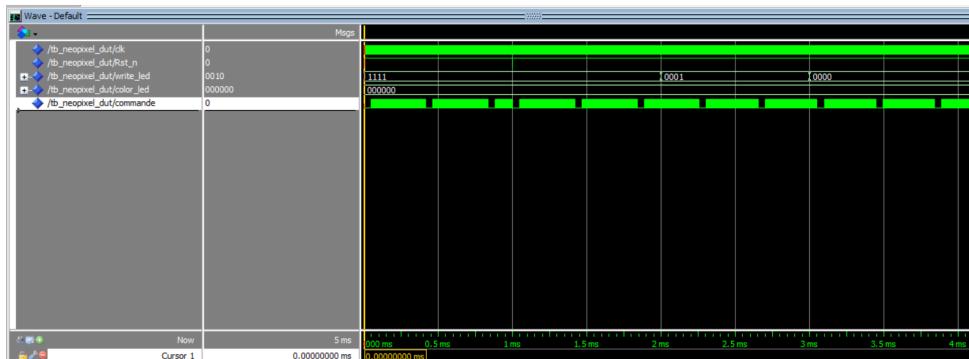


FIGURE 38 – Simulation Standalone

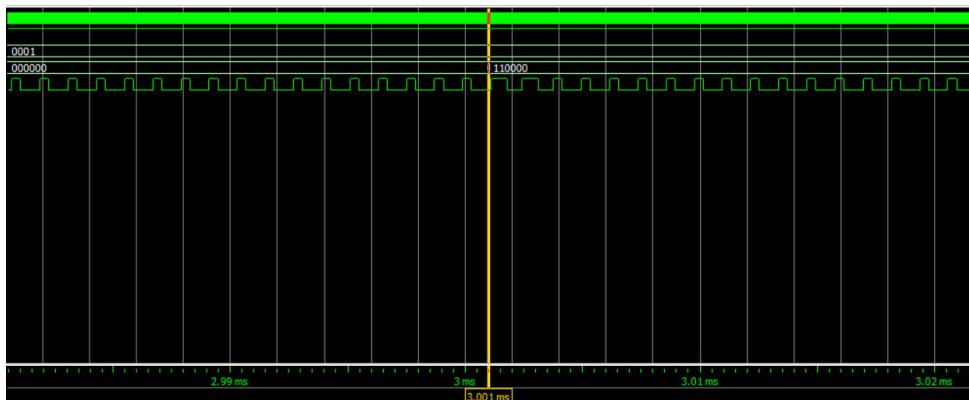


FIGURE 39 – Simulation Standalone Zoomed

7.2 IP Avalon

Pour le module Avalon, j'ai utilisé les mêmes méthodes que pour les autres IP, en changeant le diviseur de fréquence et en ajoutant les valeurs appropriées, ainsi qu'en ajoutant le codage de sélection de couleur et de valeur sur WriteData, avec 10 bits sur 32. Vous pouvez voir la simulation dans Modelsim ci-dessous :

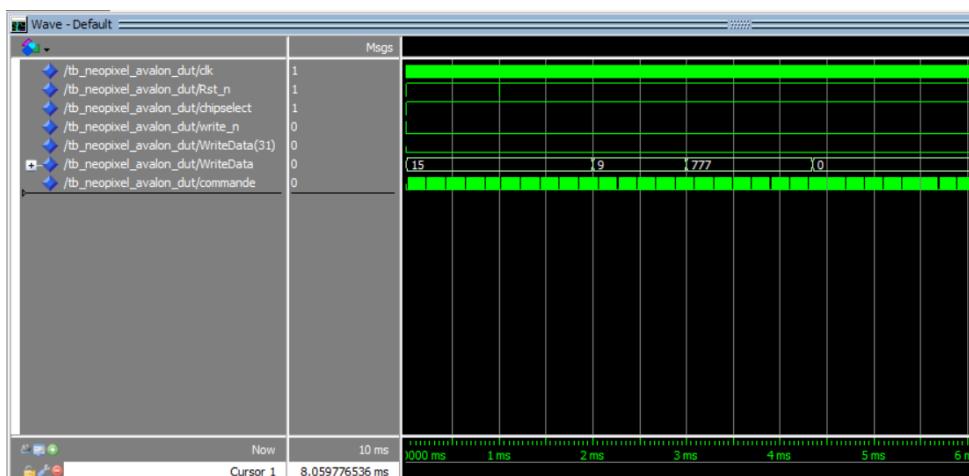


FIGURE 40 – Simulation Avalon

7.3 Demo NIOS II

Le comportement des diodes électroluminescentes était conforme aux attentes, voici la sortie du terminal :

```
NeoPixel_Testing Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1
Valeur actuelle : 9, Couleur Actuelle : 001100
Valeur actuelle : 8, Couleur Actuelle : 001100
Valeur actuelle : 7, Couleur Actuelle : 001100
Valeur actuelle : 6, Couleur Actuelle : 001100
Valeur actuelle : 5, Couleur Actuelle : 001100
Valeur actuelle : 4, Couleur Actuelle : 001100
Valeur actuelle : 3, Couleur Actuelle : 001100
Valeur actuelle : 2, Couleur Actuelle : 001100
Valeur actuelle : 1, Couleur Actuelle : 001100
Valeur actuelle : 0, Couleur Actuelle : 001100
Valeur actuelle : 1, Couleur Actuelle : 001100
Valeur actuelle : 2, Couleur Actuelle : 001100
```

FIGURE 41 – Terminal Nios2

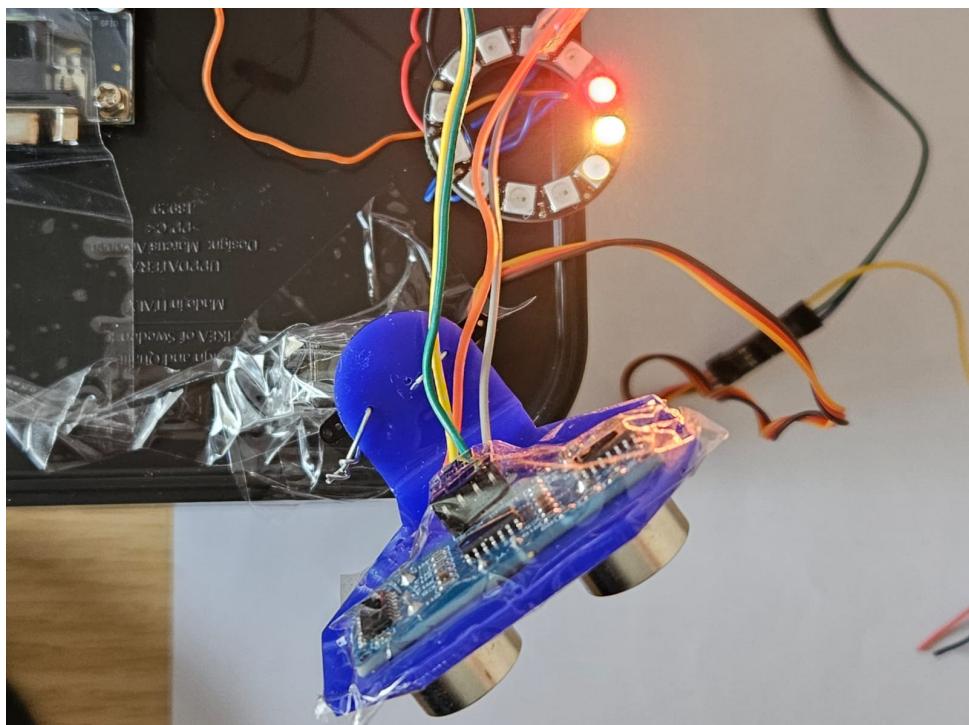


FIGURE 42 – NeoPixel et Servo



FIGURE 43 – NeoPixel et Servo

8 Montage Final

Pour l'assemblage final, un fichier Demo.c a été créé dans l'IDE Eclipse pour regrouper tous les composants dans un seul fichier. J'ai pu voir le résultat de tout ce qui fonctionne ensemble dans cet assemblage, je vous suggère donc de lire le code ainsi que la bibliothèque C que j'ai créée et qui s'appelle radar.c/.h. Le GPIOS choisi est décrit dans le schéma ci-dessous :

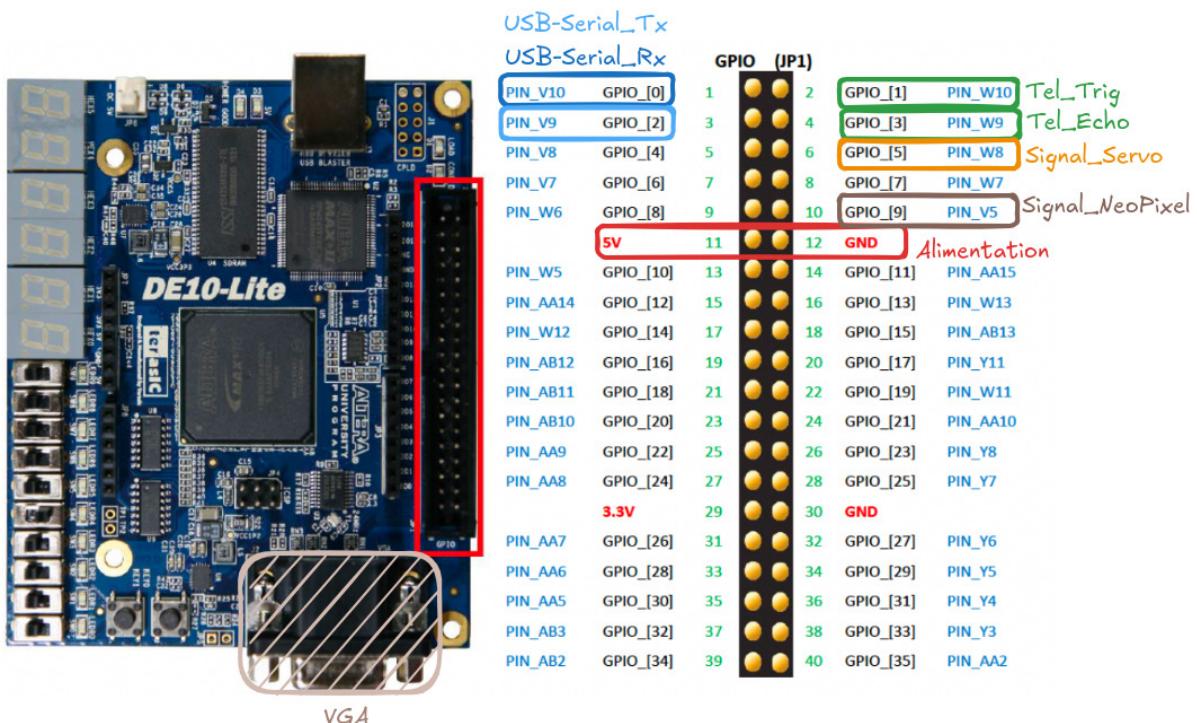


FIGURE 44 – Architecture

Enfin, vous pouvez voir la démo à partir du lien [demo.mp4](#).

9 Conclusion

Ce projet a été un grand processus d'apprentissage, nécessitant de nombreuses heures de travail pour mettre en œuvre toutes les fonctionnalités proposées. Il a été très intéressant d'exercer le processus de découverte typique de l'ingénierie, à la fois en termes de fonctionnement de composants tels que le NeoPixel, le télémètre et le servo, et également du point de vue du logiciel de modélisation de la logique en VHDL.

De nombreuses heures ont été investies dans cette mise en œuvre, la proposition est relativement vaste et le travail a été effectué au fil du temps afin d'éviter une surcharge à la fin du semestre. Ce temps a été consacré à l'étude de la documentation, du langage VHDL, que je n'avais pas encore appris, au développement et à l'implémentation de la logique IP Standalone et du bus Avalon. Et bien sûr, au cours de ce processus, la correction des bogues et l'amélioration des fonctionnalités ont été un travail constant.

Enfin, je suis très reconnaissant d'avoir eu l'occasion de développer ce projet, qui a été une expérience très constructive et très précieuse pour mon développement en tant qu'étudiant en ingénierie.

Références

- [1] Intel Corporation. *DE10-Lite Computer System User Manual*. Intel Corporation, 2018. Available in the University Program section of Intel's website.
- [2] Worldsemi Corporation. *WS2812 Intelligent Control LED Datasheet*, 2015. Technical specifications of the WS2812 LED.
- [3] CPLD CPU. *Power Analysis Probing WS2812 RGB LEDs*, December 2020. Detailed analysis of the power characteristics of WS2812 RGB LEDs.
- [4] Adafruit Industries. *Adafruit NeoPixel Uberguide*, 2021. Comprehensive guide to using NeoPixels.