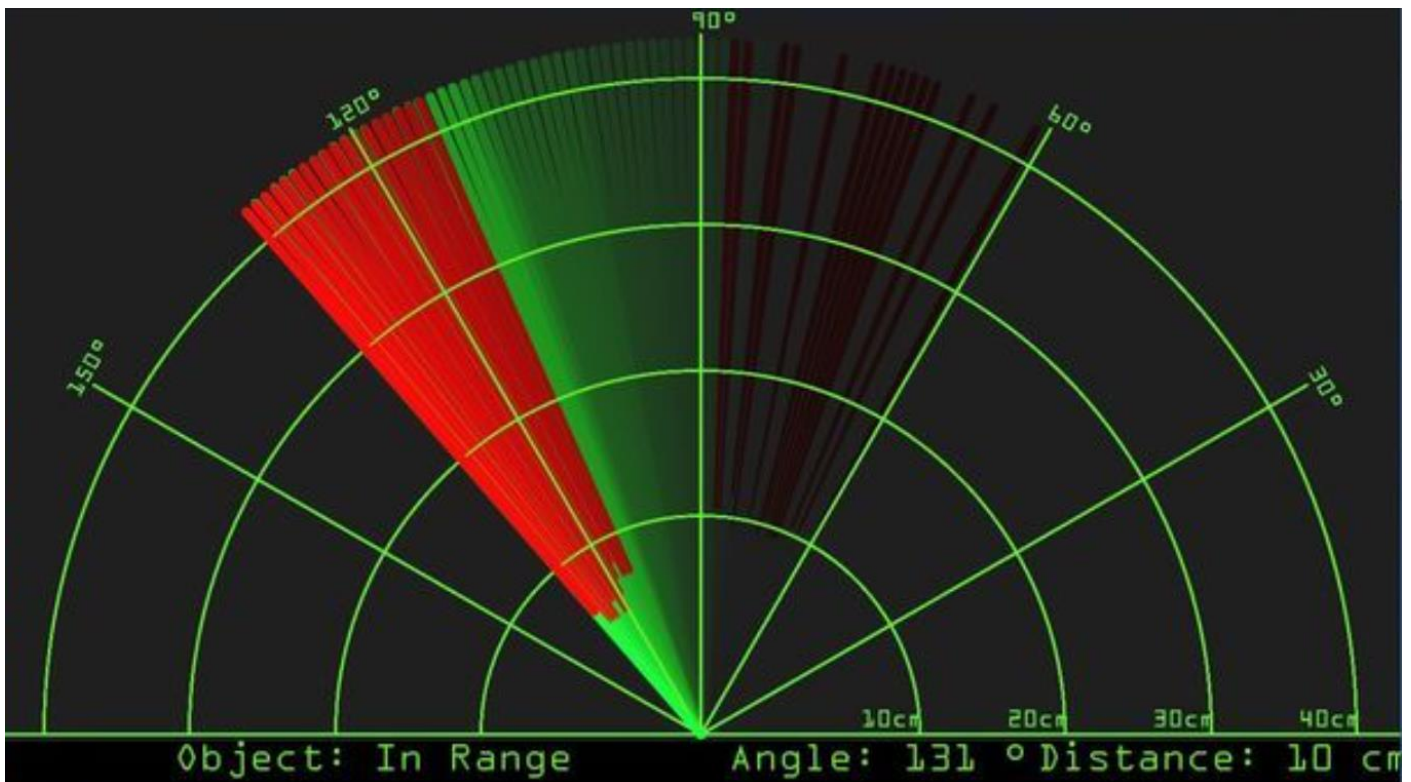


Compte-rendu du Projet

Radar 2D



Étudiant : Ayoub LADJICI

Spécialité : Électronique – Informatique

Enseignants référents : Yann DOUZE – Abdelrahman ABDELAZIM

Table des matières

Introduction.....	3
1. Implémentation de l'IP télémètre ultrason HC SR04	4
1.1 Fonctionnement du capteur HC-SR04.....	4
1.2 Simulation de l'IP Télémètre.....	4
1.3 Test de l'IP Télémètre sur carte	4
1.4 Intégration de l'IP Télémètre dans Platform Designer	6
2. Conception de l'IP Servomoteur	8
2.1 Fonctionnement du servomoteur	8
2.2 Simulation de l'IP Servomoteur.....	8
2.3 Test de l'IP Servomoteur sur carte	9

Introduction

Dans le cadre du module d'Architecture des systèmes embarqués, nous sommes amenés à réaliser un projet ayant pour objectif de concevoir un système capable de cartographier une scène en 2D et s'inscrivant dans le développement d'applications pratiques combinant matériel et logiciel.

Ce système nécessitera l'utilisation d'un télémètre ultrason HC-SR04 et d'un servomoteur avec la possibilité d'ajouter des fonctionnalités supplémentaires telles que la communication UART ou des animations lumineuses via un anneau de 12 LEDs NeoPixel WS2812.

L'archive du projet est structurée de manière à faciliter la progression et l'organisation des différentes parties :

- **Dossier src** : contient tous les fichiers en VHDL pour décrire les composants nécessaires
- **Dossier simu** : regroupe les testbenchs pour simuler les composants
- **Dossier fit** : inclut le fichier .sof pour la configuration du FPGA et la mise en œuvre du design.

Un dossier supplémentaire, DE10_Lite_Computer_YD, contient les fichiers suivants :

- **.sopcinfo et .qsys** : fichiers associés à la génération et à la description du système.
- **.sof** : fichier de configuration pour le FPGA.
- **Dossier software** : contient les projets programmés en C

1. Implémentation de l'IP télémètre ultrason HC SR04

1.1 Fonctionnement du capteur HC-SR04

Le télémètre ultrason HC-SR04 permet de mesurer une distance en émettant une onde ultrasonore via son signal *Trig*. Cette onde se réfléchit sur un obstacle et revient sous forme d'un signal réfléchi sur la sortie *Echo*. La distance peut être calculée en multipliant la vitesse du son ($v_{\text{son}} = 340 \text{ m/s}$) par la durée entre l'émission et la réception. On sait que le son fait un aller-retour donc on divisera la durée par 2. Par ailleurs, en VHDL, nous ne calculerons pas la durée mais le nombre de ticks, qui sera associé à un front montant de l'horloge. La fréquence d'horloge de la carte DE10-Lite est de 50 MHz.

$$\text{Distance} = \frac{v_{\text{son}} \times \text{duree}}{2 \times F_{\text{clk}}} = \frac{34000 \frac{\text{cm}}{\text{s}} \times \text{duree}}{2 \times 50 \times 10^6 \text{ Hz}} = \frac{\text{duree}}{2941} \sim \frac{\text{duree}}{3000}$$

On va préférer une division par 3000 pour faciliter les calculs en logique numérique.

Pour chaque composant de notre projet, nous allons écrire un testbench pour vérifier le bon fonctionnement. J'ai créé une variable booléenne *OK* qui indique si les tests ont été réussis.

1.2 Simulation de l'IP Télémètre

J'ai d'abord testé mon IP Télémètre d'abord en vérifiant si la durée de l'impulsion *Trig* à l'état haut valait bien 10 μs . Puis j'ai testé si mon télémètre affichait la bonne distance en fonction de deux durées d'impulsion *Echo* différente. Par exemple, pour un *Echo* de 540 μs , on peut voir sur la Figure 1.1 que le nombre de ticks calculés à l'état MEASURE correspond bien à la durée de l'Echo, en effet $\frac{540 \times 10^{-6}}{20 \times 10^{-9}} = 27000$ ticks, où 20 ns est la période d'un tick, et que la distance obtenue est celle attendue ($\frac{27000}{3000} = 9 \text{ cm}$). De même, pour un *Echo* de 180 μs , le nombre de ticks vaut 9000 et la distance obtenue est 3 cm.

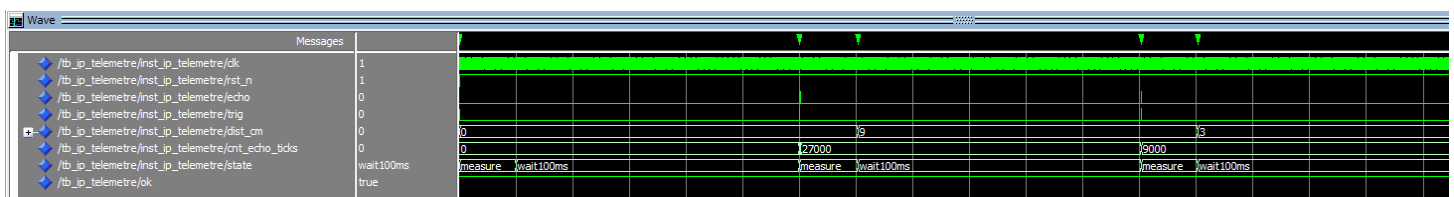


Figure 1.1 : Chronogramme illustrant la simulation de l'IP Télémètre

1.3 Test de l'IP Télémètre sur carte

Nous allons créer un projet sur le logiciel Quartus développé par Intel permettant principalement la conception, la simulation, la synthèse et la programmation de FPGA. Dans un premier temps, nous allons compiler le fichier source de notre télémètre qui sera considéré comme le Top Level. Puis on doit réaliser l'assignement des pins en respectant le tableau entrées-sorties de l'IP de la manière suivante :

Signaux de l'entité telemetre_us_HC_SR04	Signaux de la carte DE10_Lite (voir User Manual)	Pinout
Rst_n	KEY0	PIN_B8
CLK	MAX10_CLK1_50	PIN_P11
Trig	GPIO_[1]	PIN_W10
Echo	GPIO_[3]	PIN_W9
Dist_cm (9 downto 0)	LEDR[9..0]	Voir page 27 du DE10-Lite_User_Manual.pdf

A présent, on peut recompilier et téléverser le projet sur la carte.

Pour le premier test, j'ai placé une boîte, qu'on va considérer comme notre obstacle, à une distance de 9 cm du capteur (voir Figure 1.3.1). Les LEDs affichent la valeur de la distance en binaire, c'est-à-dire chaque LED représente une puissance de 2, dont celle la plus à droite représente la moins significative et celle la plus à gauche, la plus significative. On observe que la LED 0 et la LED 3 se sont allumées donc on a : $2^0 + 2^3 = 1 + 8 = 9$. Donc ce premier test est validé.

Pour le second test, j'ai placé cette fois la boîte à une distance de 3 cm du capteur (voir Figure 1.3.2). On observe que la LED 0 et la LED 1 se sont allumées donc on a : $2^0 + 2^1 = 1 + 2 = 3$. Donc le second test est validé.

Par conséquent, notre télémètre fonctionne bien sur notre carte FPGA.

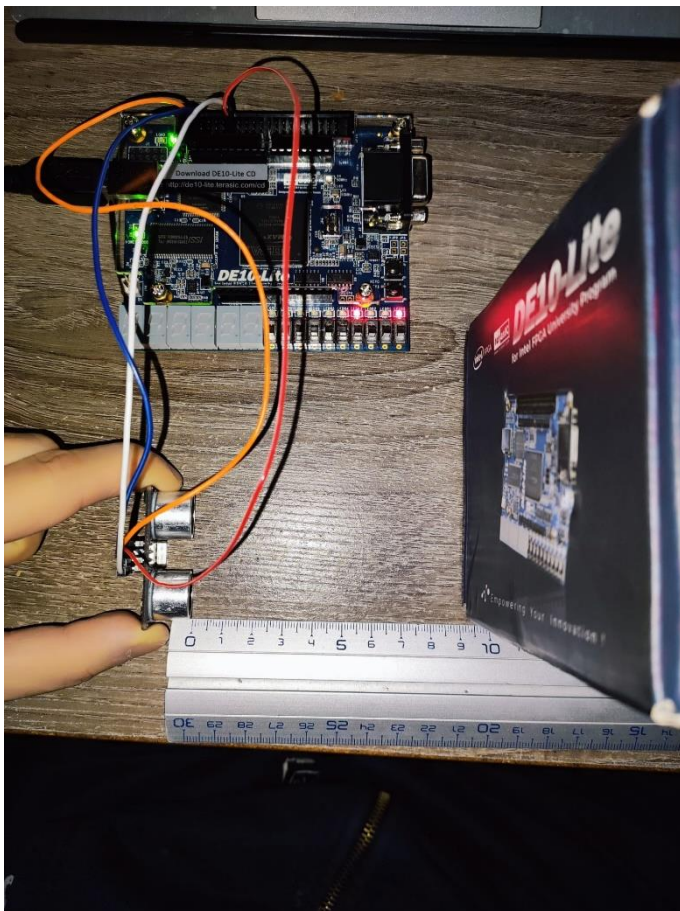


Figure 1.3.1 : Test du télémètre pour une distance de 9 cm

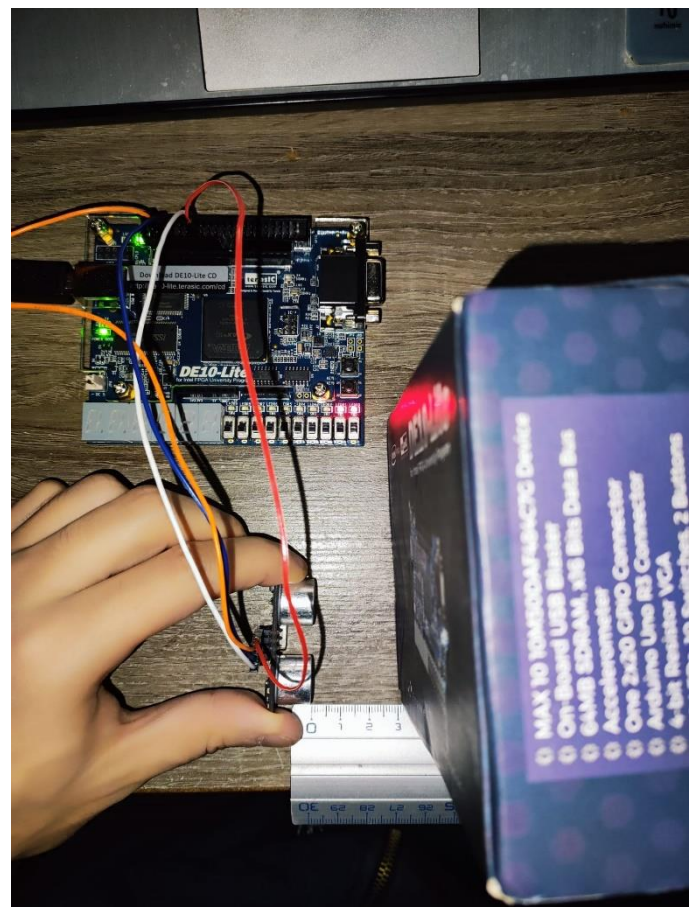


Figure 1.3.2 : Test du télémètre pour une distance de 3 cm

1.4 Intégration de l'IP Télémètre dans Platform Designer

Platform Designer est un outil permettant de construire notre système autour d'un processeur softcore, Nios2. Cet environnement nous permet d'ajouter notre composant télémètre via une interface Avalon afin de faciliter le transfert de données entre le matériel et le logiciel. Nios2 peut lire les mesures du télémètre en accédant à l'adresse mémoire des signaux standard (*Read_n*, *chipselect*, *readdata*).

Un testbench a été réalisé pour vérifier la communication entre l'IP Télémètre et le bus Avalon. Pour cela, j'ai effectué deux tests pour vérifier si les valeurs de distance mesurée ont été correctement transmises dans le signal *readdata*. Par exemple, pour un *Echo* de 540 μ s, on peut voir sur la Figure 1.4.1 qu'on mesure une distance de 9 cm et que la valeur de *readdata* se met bien à jour lorsque *chipselect* est à l'état haut et *read_n* à l'état bas pour permettre une demande de lecture sur le périphérique. De même, pour un *Echo* de 180 μ s, la valeur de *readdata* est identique à celle de la distance mesurée. Donc l'interface Avalon de notre télémètre est bien fonctionnelle.

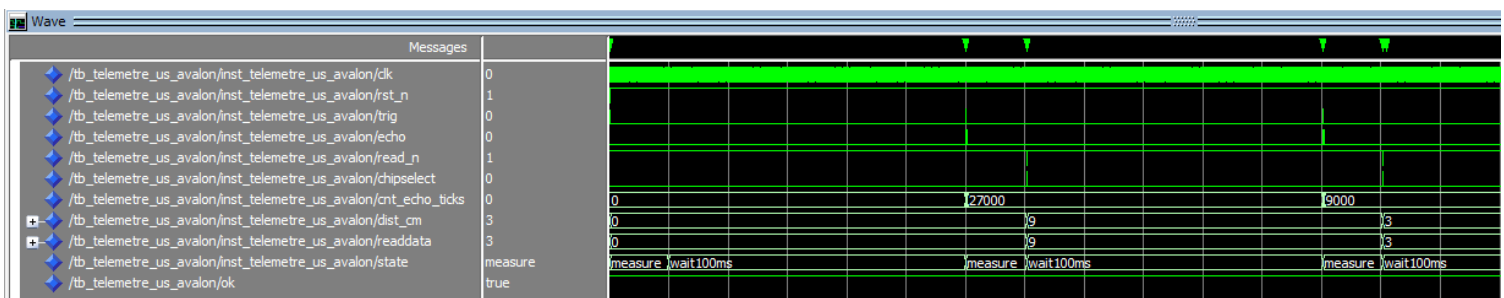


Figure 1.4.1 : Chronogramme illustrant la simulation du Télémètre Avalon

On peut désormais intégrer ce composant avec le reste des périphériques nécessaires au processeur. L'interface de notre composant a été configurée comme sur la Figure 1.4.2.

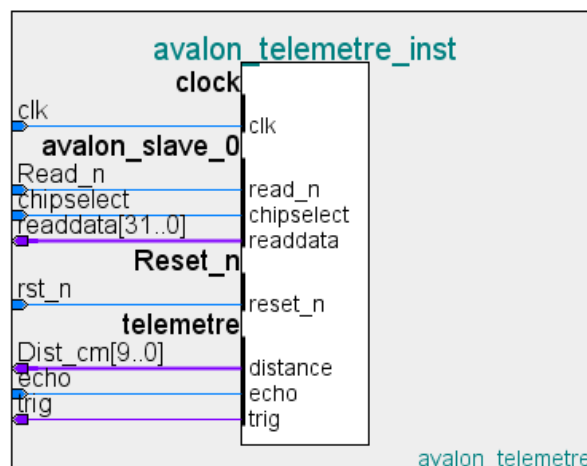
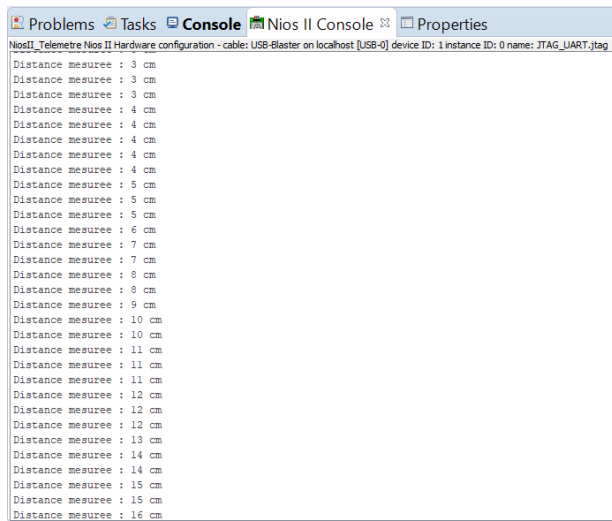


Figure 1.4.2 : Schéma illustrant l'entité Télémètre Avalon

On génère le nouveau système puis on ajoute les signaux du télémètre (*distance*, *trig*, *echo*) dans le code VHDL du top-level. Enfin, on peut compiler puis implémenter le système sur la carte FPGA.

Dans un premier temps, nous avons programmé en langage C le processeur Nios2 pour qu'il récupère et affiche les mesures de distance dans le terminal de la console du Nios2. Je l'ai testé en glissant ma boîte sur une distance allant de 3 cm jusqu'à 16 cm. Comme illustré dans la Figure 1.4.3, les valeurs mesurées et affichées en temps réel confirment le bon fonctionnement de l'interface logicielle avec l'IP Télémètre.



```

Problems Tasks Console Nios II Console Properties
NiosII_Telemetre Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: JTAG_UART.jtag
Distance mesurée : 3 cm
Distance mesurée : 3 cm
Distance mesurée : 3 cm
Distance mesurée : 4 cm
Distance mesurée : 4 cm
Distance mesurée : 4 cm
Distance mesurée : 4 cm
Distance mesurée : 5 cm
Distance mesurée : 5 cm
Distance mesurée : 5 cm
Distance mesurée : 6 cm
Distance mesurée : 7 cm
Distance mesurée : 7 cm
Distance mesurée : 8 cm
Distance mesurée : 8 cm
Distance mesurée : 9 cm
Distance mesurée : 10 cm
Distance mesurée : 10 cm
Distance mesurée : 11 cm
Distance mesurée : 11 cm
Distance mesurée : 11 cm
Distance mesurée : 12 cm
Distance mesurée : 12 cm
Distance mesurée : 12 cm
Distance mesurée : 13 cm
Distance mesurée : 14 cm
Distance mesurée : 14 cm
Distance mesurée : 15 cm
Distance mesurée : 15 cm
Distance mesurée : 16 cm
  
```

Figure 1.4.3 : Capture d'écran du terminal de la console du Nios2 affichant les mesures du télémètre

Dans un second temps, nous l'avons programmé pour qu'il puisse cette fois afficher la distance en cm sur les afficheurs 7 segments. Comme l'illustre la Figure 1.4.4 et 1.4.5, les tests effectués sur 2 distances différentes affichent la valeur correcte sur les afficheurs 7 segments donc l'affichage des distances en temps réel fonctionne parfaitement.

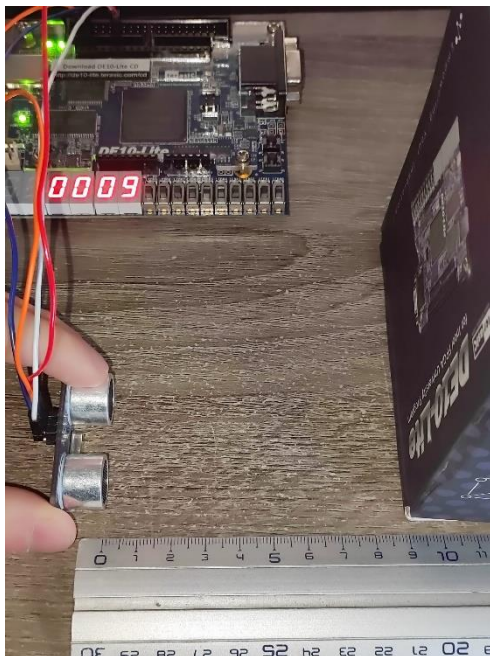


Figure 1.4.4 : Test d'affichage pour une distance de 9 cm

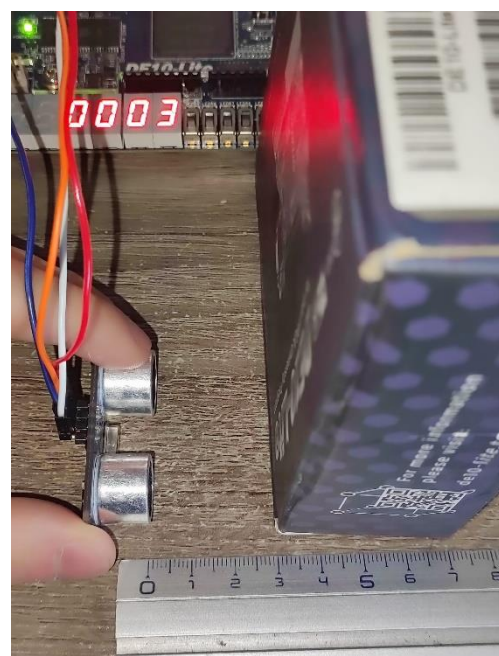


Figure 1.4.5 : Test d'affichage pour une distance de 3 cm

2. Conception de l'IP Servomoteur

2.1 Fonctionnement du servomoteur

Pour ce projet, j'ai utilisé le servomoteur SG90 qui peut normalement tourner entre 0° et 180°. Il est commandé en utilisant un signal modulé en largeur d'impulsion (PWM) de 50 Hz de fréquence, soit une impulsion toute les 20 ms. La position du servomoteur est déterminée par la durée des impulsions. Voici notre configuration PWM :

- Position = 0 => 1 ms (50 000 ticks) => 0°
- Position = 85 => 1.50 ms (75 000 ticks) => 45°
- Position = 170 => 2 ms (100 000 ticks) => 90°
- Position = 255 => 2.5 ms (125 000 ticks) => 135°

Rappel : 1 Tick = 20 ns

On utilisera la formule suivante pour calculer la largeur d'impulsion en nombre de ticks :

$$\text{Duty} = 50\,000 + \frac{\text{Position} \times 75\,000}{255} \text{ où Position varie entre 0 et 255}$$

Voici un tableau dressant le branchement du SG90 sur la carte DE10-Lite :

Servomoteur SG90	Couleur du fil	DE10-Lite
GND	Marron	GND
5V	Rouge	5V
Signal PWM	Jaune-orangé	GPIO 0

J'ai fait le choix de concevoir mon IP Servomoteur de manière comportementale. Cela se déroule en trois processus : le premier permet de calculer la durée d'impulsion PWM en fonction de la valeur des switchs sur 8 bits et ainsi piloter la position angulaire du servomoteur, le deuxième pour gérer la période du signal PWM à l'aide d'un compteur, et le troisième pour générer le signal PWM en comparant la durée calculée avec le compteur.

2.2 Simulation de l'IP Servomoteur

Des tests ont été réalisés aux angles : 0°, 45°, 90° et 135°. Chaque test nous a permis de vérifier que la durée de la largeur d'impulsion correspondait à la position attendue. Les résultats obtenus sur la Figure 2.2.1 confirment le bon fonctionnement de l'IP Servomoteur.

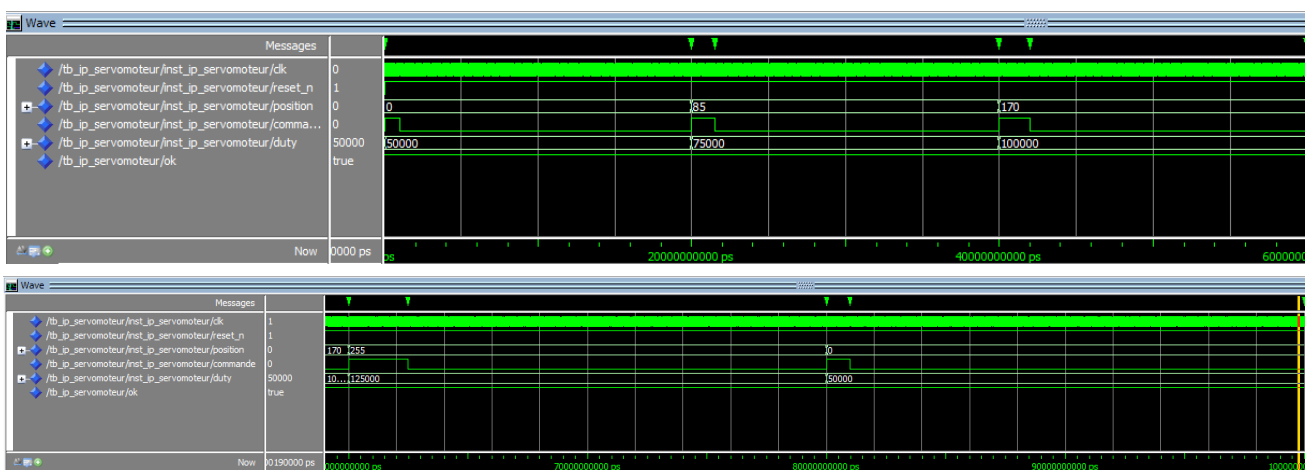


Figure 2.2.1 : Chronogramme illustrant la simulation de l'IP Servomoteur

2.3 Test de l'IP Servomoteur sur carte

Préalablement, nous avons vérifié à l'aide de notre Analog Discovery si la largeur d'impulsion de la sortie *commande* correspondait bien à la position imposée en entrée. Comme l'illustre la Figure 2.3.1, pour chaque position, nous avons la bonne durée d'impulsion, qui est répété toutes les 20ms donc notre carte FPGA a bien été configurée. Maintenant, nous pouvons le tester avec notre servomoteur.

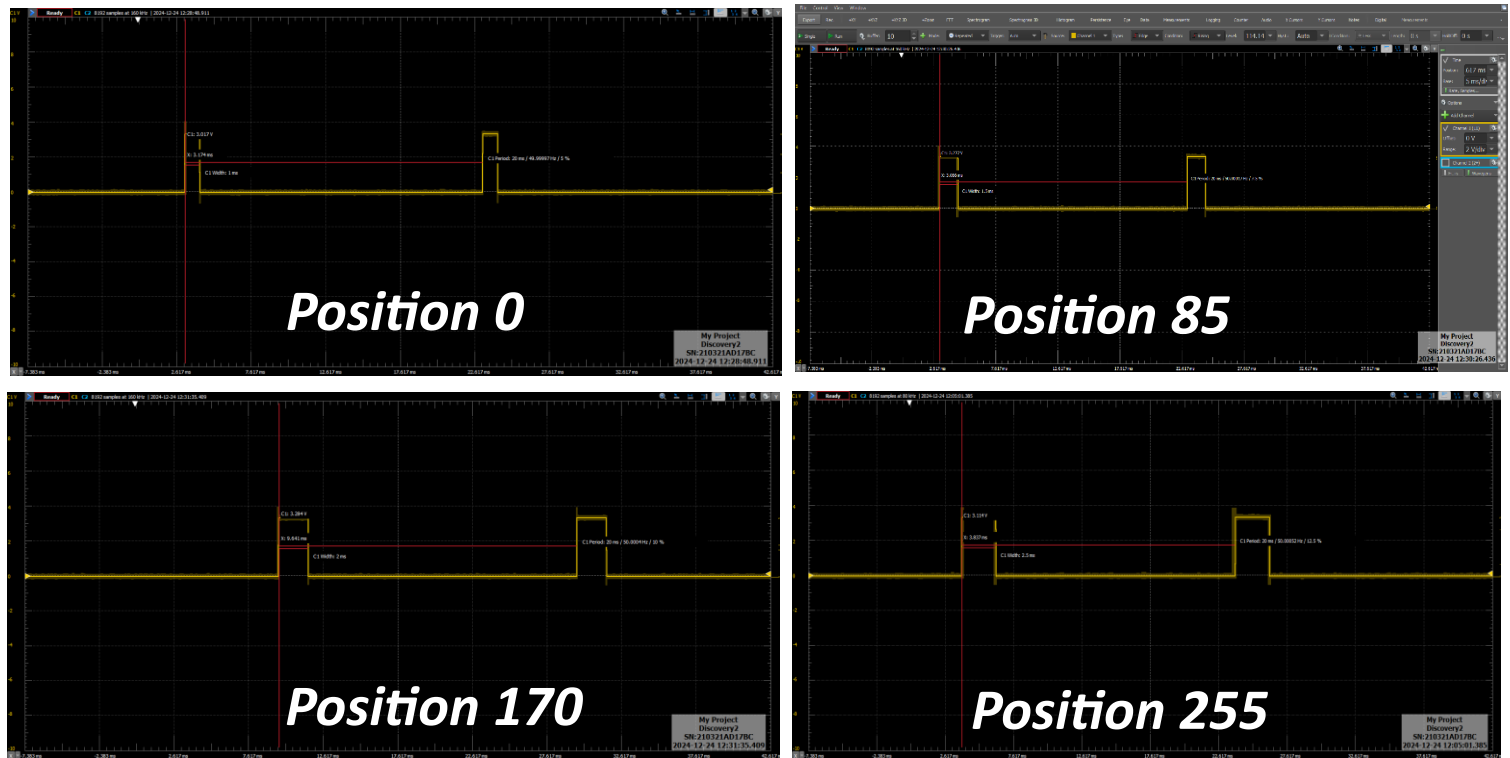


Figure 2.3.1 : Chronogramme illustrant le timing de la sortie commande en fonction de différentes valeurs de positions (0, 85, 170, 255)

Nous avons actionné pour chaque test les switches nécessaires pour avoir la position souhaitée. Et comme l'illustre la Figure 2.3.2, pour chaque test, le servomoteur est parvenu à effectuer l'angle de rotation qui a été établi dans notre cahier des charges. Tous les tests ont été validés donc notre servomoteur fonctionne bien sur notre carte FPGA.

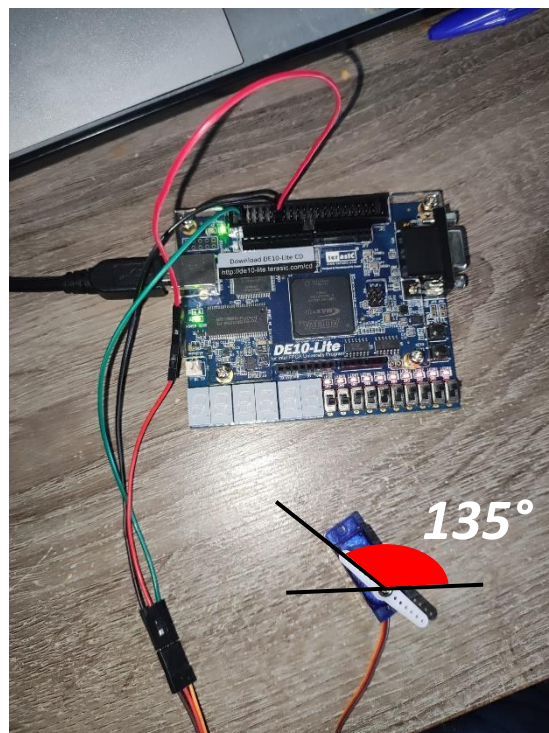
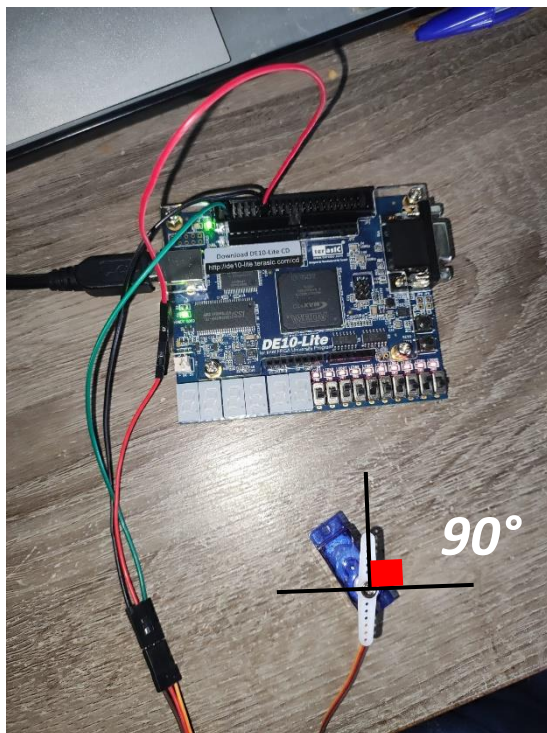
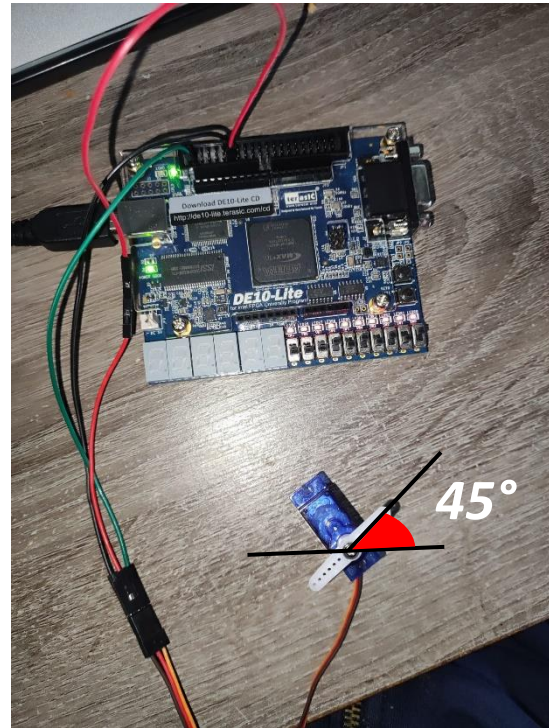
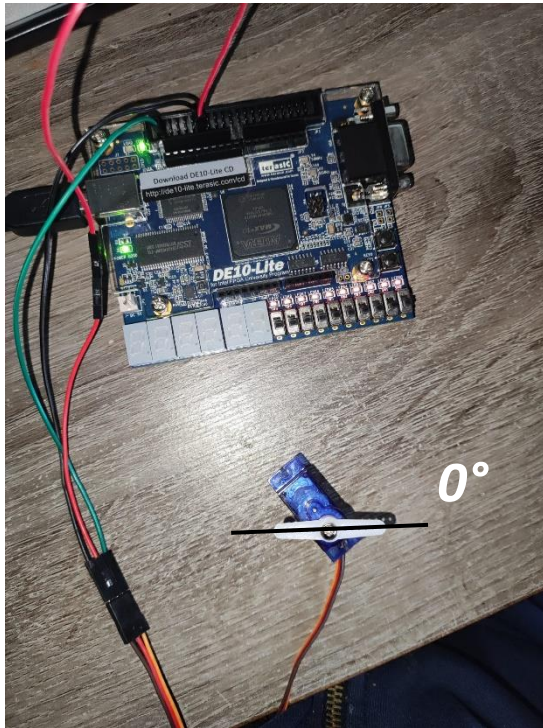


Figure 2.3.2 : Test de chaque angle de rotation du servomoteur en fonction de la position imposée par l'utilisateur

2.4 Intégration matérielle de l'IP Servomoteur

Pour vérifier la communication entre l'IP Servomoteur et le bus Avalon, j'ai effectué 4 tests à différents angles (0°, 45°, 90° et 135°) afin d'observer s'il y a bien une correspondance entre

les valeurs écrites dans le signal *writedata* et la durée des impulsions PWM envoyées au servomoteur. Les résultats obtenus sur la Figure 2.4.1 montrent que l'interface Avalon de notre servomoteur est bien fonctionnelle.

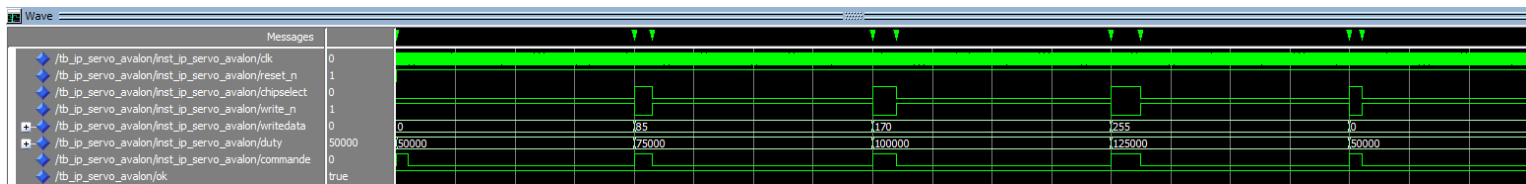


Figure 2.4.1 : Chronogramme illustrant la simulation du Servomoteur Avalon

On peut intégrer ce composant dans notre processeur comme nous l'avons fait avec le télémètre. L'interface de notre composant a été configurée comme sur la Figure 2.4.2.

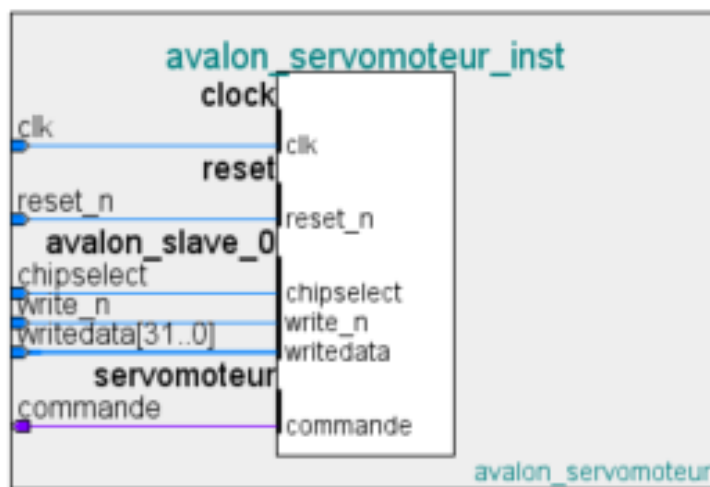


Figure 2.4.2 : Schéma illustrant l'entité Servomoteur Avalon

Ensuite, nous avons programmé le processeur pour qu'il puisse lire la position des switches et l'envoyer au servomoteur pour qu'il effectue l'angle de rotation associé.

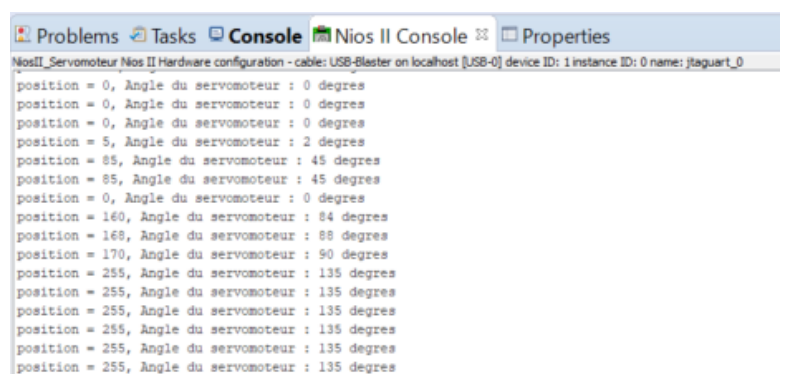


Figure 2.4.3 : Capture d'écran du terminal de la console du Nios2 affichant La position des switches et l'angle effectué par le servomoteur

3. Affichage des obstacles

Dans cette partie, l'objectif est de pouvoir faire tourner notre servomoteur sur 180° tout en faisant des mesures régulières avec le télémètre ultrason. Toutefois, notre servomoteur permet un balayage angulaire allant jusqu'à 135° . Les résultats peuvent présenter quelques imprécisions car le capteur est légèrement incliné vers le haut après l'avoir fixé sur les hélices du servomoteur. Le terminal de la console Nios2 affiche en temps réel la distance mesurée pour différentes positions du télémètre.

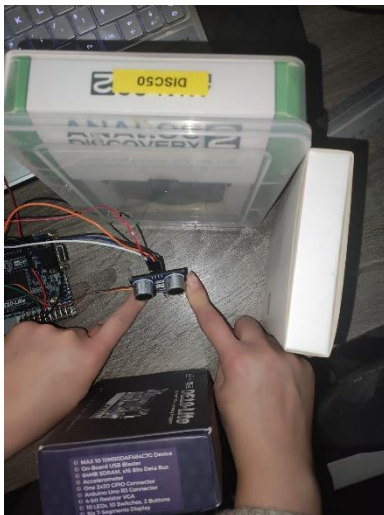


Figure 3.1 : Photos prises à 3 moments (0° , 90° , 135°) lors du balayage angulaire du servomoteur

```
Problems Tasks Console Nios II Console
NiosII_affichage_obstacle Nios II Hardware configuration - cable: USB-Blaster on localhost
0° -> 5 cm
1° -> 5 cm
1° -> 5 cm
1° -> 5 cm
2° -> 5 cm
2° -> 5 cm
2° -> 5 cm
3° -> 5 cm
3° -> 5 cm
4° -> 5 cm
4° -> 5 cm
4° -> 5 cm
5° -> 5 cm
5° -> 5 cm
6° -> 5 cm
6° -> 5 cm
7° -> 5 cm
7° -> 5 cm
8° -> 5 cm
8° -> 5 cm
```

```
Problems Tasks Console Nios II Console
NiosII_affichage_obstacle Nios II Hardware configuration - cable: USB-Blaster on localhost
87° -> 11 cm
87° -> 11 cm
88° -> 11 cm
88° -> 11 cm
88° -> 11 cm
89° -> 11 cm
89° -> 11 cm
89° -> 11 cm
90° -> 11 cm
90° -> 11 cm
90° -> 10 cm
91° -> 10 cm
91° -> 10 cm
91° -> 10 cm
92° -> 10 cm
92° -> 10 cm
92° -> 10 cm
93° -> 10 cm
93° -> 10 cm
94° -> 10 cm
94° -> 10 cm
94° -> 10 cm
95° -> 10 cm
```

```
Problems Tasks Console Nios II Console
NiosII_affichage_obstacle Nios II Hardware configuration - cable: USB-Blaster on localhost
140° -> 11 cm
139° -> 11 cm
139° -> 11 cm
139° -> 11 cm
138° -> 11 cm
138° -> 11 cm
138° -> 11 cm
137° -> 11 cm
137° -> 12 cm
137° -> 12 cm
136° -> 12 cm
136° -> 12 cm
135° -> 12 cm
135° -> 12 cm
135° -> 12 cm
134° -> 12 cm
134° -> 12 cm
133° -> 13 cm
133° -> 13 cm
133° -> 13 cm
132° -> 14 cm
```

Figure 3.2 : Captures d'écran du terminal de la console du Nios2 affichant la distance mesurée en fonction de l'angle de rotation du télémètre