Projet Électronique Numérique
User manuel
EN202

Ranchoup Théo, Msallem Maysa

10 janvier 2024

Table des matières

1 Introduction

Le but de ce projet est de faire du contrôle de servomoteur à l'aide d'un accéléromètre via un FPGA. Pour cela, nous avons dû créer une architecture qui lit les données de l'accéléromètre et qui les transmet au servomoteur. Le but est de faire en sorte que le servomoteur suive l'inclinaison de la carte. C'est-à-dire qu'un angle de 30° de la carte se caractérisera par un angle de 30° sur le servomoteur. Nous avons aussi décidé de faire des indicateurs visuels pour connaître les différents états, dans lequel est la carte. Cela nous permet aussi de connaître l'inclinaison actuelle de la carte.

Nous avons aussi fait un travail d'optimisation du projet, en mettant par exemple des tables de vérités plutôt que des process asynchrones.

2 Configuration physique de la carte

Tout le projet s'est fait avec une carte (FPGA) Nexys 4 DDR.
Il y a 3 modes de fonctionnements (détaillés dans la section d'après) :

- Initial
- Play
- Pause

Pour changer d'état, il faut utiliser les boutons BTNL et BTNR.
Enfin, pour brancher le servomoteur, il faut le brancher sur le port des PMOD :

- JB4 pour la PWM (fil orange pour les servomoteurs SG90)
- pin 6 en 3.3V pour l'alimentation (le fil rouge pour les servomoteurs SG90)
- pin 5 (le gnd) pour les masses communes (le fil marron pour les servomoteurs SG90)

Cela suit le schéma suivant avec en orange le pin de la PWM (figure 1).

FIGURE 1 – Schéma de branchement du servomoteur (schéma issu du Reference Manual)

L'accéléromètre ADXL362, étant monté en surface sur la carte, il n'y a pas besoin de faire les branchements. Le montage final ressemble alors à cela (figure 2) :

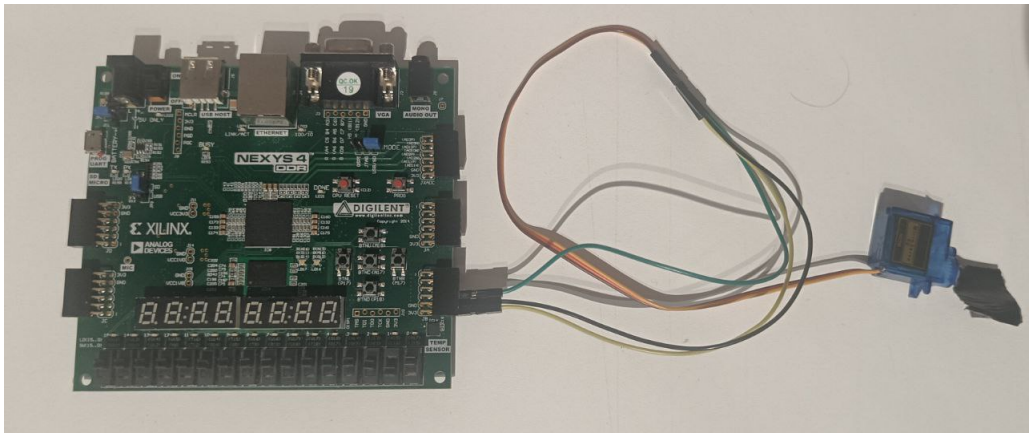


FIGURE 2 – Montage complet

3 Architecture

3.1 Architecture globale du projet

Nous avons une architecture qui se décompose en différents blocs, comme illustré dans le schéma figure 3.

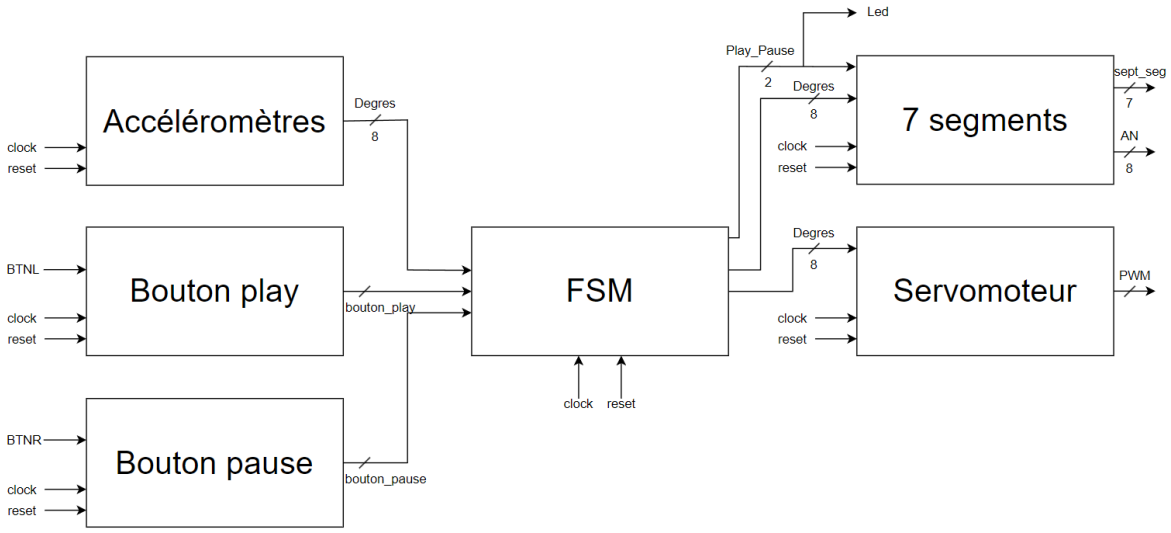


FIGURE 3 – Architecture globale du projet.

3.2 Architecture de la FSM

Les différents états de la FSM sont décrits dans le schéma figure 4.

- **État INITIAL** : Nous sommes en état d'attente, avec un état initial de 0°. Ainsi, lorsque nous passons par cet état, on passe par un angle de 0°. Sur les 7 segments, il y a des traits horizontaux pour indiquer cet état. De même, les leds K15 et H17 sont toutes les deux allumées.
- **État PLAY** : Nous sommes dans l'état dans lequel le servomoteur suit l'inclinaison de la carte. Sur les 7 segments, il y a une flèche pour indiquer cet état. De même, la led K15 est allumée et H17 est éteinte.
- **État PAUSE** : Nous sommes dans l'état dans lequel le servomoteur est en pause. Si nous décidons de reprendre le suivi, nous reprendrons depuis la dernière position du servomoteur. Sur les 7 segments, il y a deux traits verticaux pour indiquer cet état. De même, la led K15 est éteinte et H17 est allumée.

Nous avons fait le choix d'associer le bouton play (B_play) à BTNL, et le bouton pause (B_pause) à BTNR.

De plus, il y a 2 reset différents (un naturellement à 0 pour l'accéléromètre et un naturellement à 1 pour le reste de l'architecture), c'est pourquoi l'interrupteur L16 doit être orienté vers le bas et J15 vers le haut.

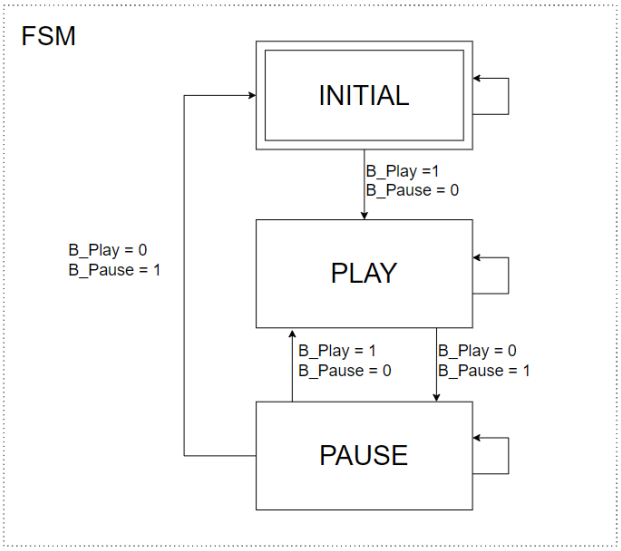


FIGURE 4 – Architecture de la FSM.

Nous détaillerons l'architecture de chaque bloc dans la partie associée au sus nommé. De plus, chaque bloc fait l'objet d'un top level intermédiaire pour faciliter l'intégration de chacun de ces blocs.

4 Servomoteur

4.1 Principe du contrôle d'un servomoteur SG90

Le servomoteur se contrôle à l'aide d'un signal PWM. Il s'agit d'un signal modulé de fréquence 50 Hz, avec une largeur comprise entre $0.5mss$ et $2.5ms$. Ainsi, nous devons avoir un signal PWM de la forme figure 5 :

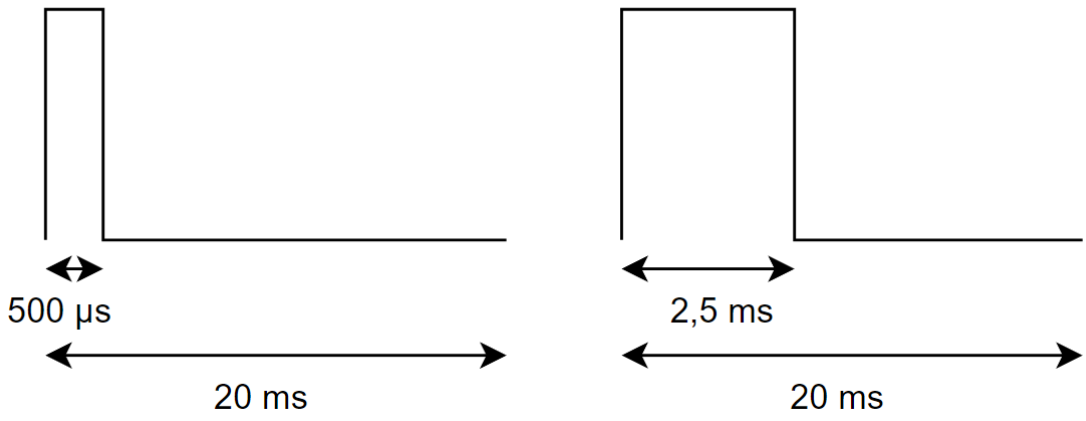


FIGURE 5 – Signal PWM à appliquer au servomoteur.

Le signal PWM avec une largeur de $0.5ms$ correspond à une consigne d'un angle de 0° pour le servomoteur. De même, un signal PWM de $2.5ms$ correspond à une consigne pour un angle de 180°.

Ces valeurs (fréquence et largeur) pour la PWM ont été obtenus après mesures sur une carte qui commande des servomoteurs.

4.2 Architecture VHDL pour la commande du servomoteur

Nous avons donc décomposé l'architecture en différents blocs : nous avons des degrés en entrée. Ces degrés sont conservés dans un registre jusqu'à ce qu'ils soient actualisés toutes les secondes. Ce délai est donné pour laisser le temps au servomoteur d'atteindre sa destination avant qu'il reçoive une autre commande, ce qui pourrait causer des problèmes mécaniques. Ils sont ensuite convertis par un process asynchrone (look up table) en nombre de coups d'horloges. Il s'agit du temps correspondant à la largeur de la PWM en nombre de coups d'horloge. L'architecture suit donc le schéma figure 6

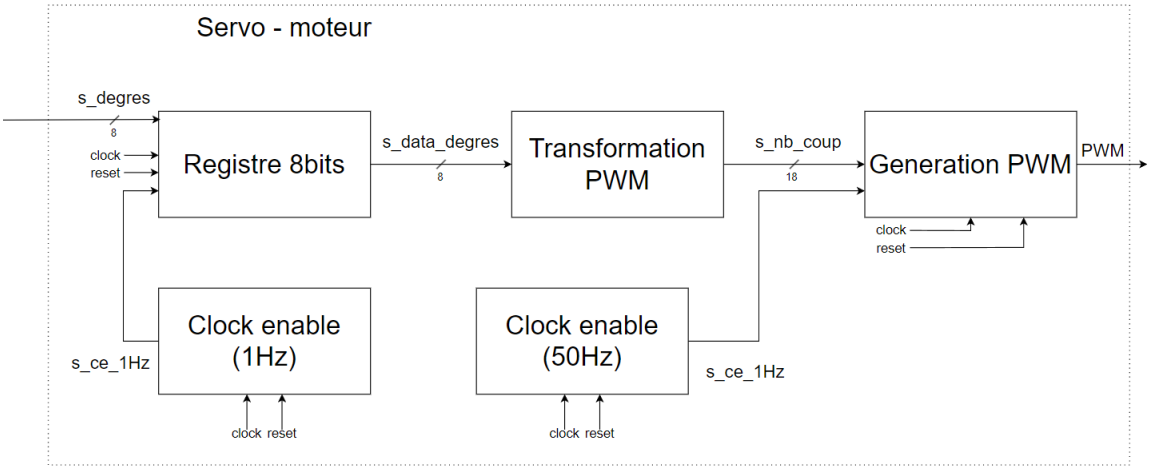


FIGURE 6 – Architecture de la commande du servomoteur

5 Acceleromètre

5.1 Description VHDL utilisée

L'accéléromètre est monté en surface sur la carte et communique en SPI avec le FPGA. Pour des raisons de facilités d'utilisations, nous allons utiliser une description envoyée par monsieur Jégo. Cette description nous permet d'avoir l'inclinaison de la carte. Après l'avoir adapté à notre système, nous obtenons de la description qui donne une valeur comprise entre 0 et 511 correspondants à l'inclinaison de la carte. Cette valeur doit alors ensuite être adaptée pour être convertie en degrés. C'est-à-dire compris entre 0° et 180°. Pour cela, nous avons utilisé dans un premier temps un calcul asynchrone. Nous avons finalement décidé de faire une look up table pour optimiser les ressources. Nous avons ainsi dû faire une légère approximation dans la conversion. De plus, il est important de noter que nous allons utiliser que l'axe X. Ce choix est arbitraire et nous aurions aussi pu choisir l'axe Y.

5.2 Architecture VHDL pour l'accéléromètre

Nous avons alors l'architecture suivante (figure 7) :

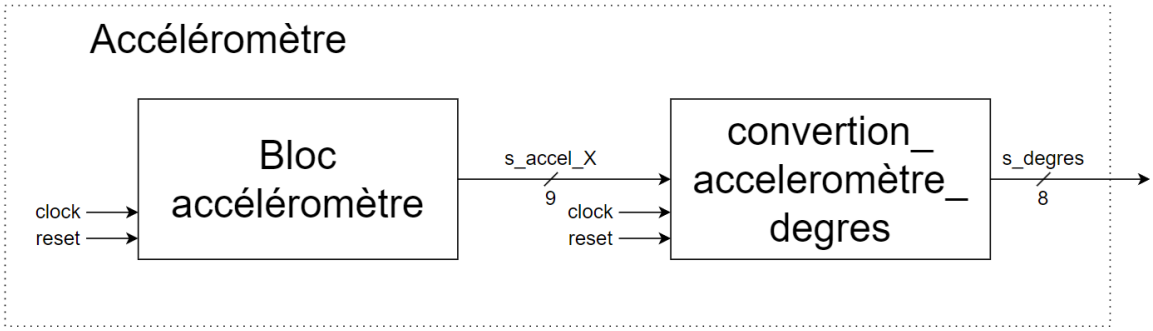


FIGURE 7 – Architecture de l'accéléromètre.

6 Affichage

6.1 Principe de fonctionnement de l'afficheur 7-segments

La carte Nexys 4 contient 2 afficheurs 7-segments à anode commune de 4 chiffres chacun. Ces afficheurs sont configurés de manière à ne former qu'un afficheur de 8 chiffres.

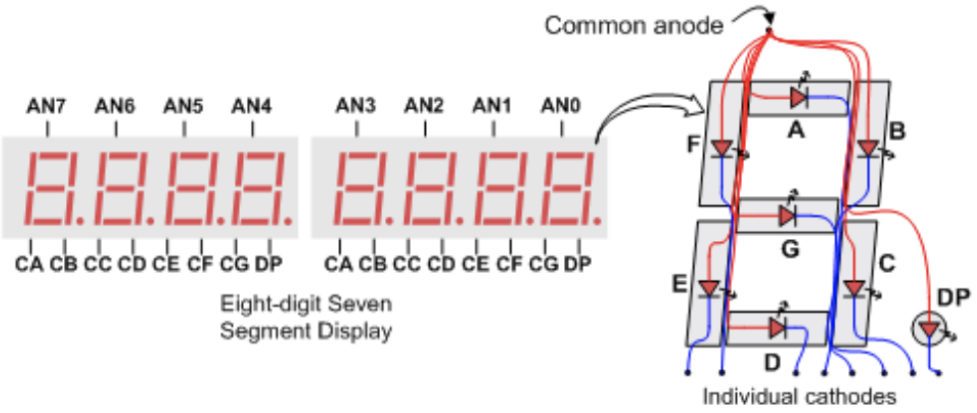


FIGURE 8 – Schéma de l'afficheur 7-segments (Schéma issu de Reference Manual)

Chaque chiffre est formé de 7 LEDs. Bien que les anodes des 7 LEDs soient connectées, elles peuvent être allumées individuellement, car leurs cathodes restent indépendantes. Cependant, les cathodes des 8 chiffres, correspondant à une LED du 7-segments, sont regroupées en un nœud. Ainsi, les signaux de cathode sont communs à tous les chiffres, mais ils peuvent uniquement illuminer les segments du chiffre dont le signal d'anode est activé. Par conséquent, pour afficher des motifs sur les 8 chiffres, on utilise un contrôleur d'affichage par balayage. Pour que les 8 motifs soient continuellement illuminés, l'anode de chaque motif doit être activée successivement et continuellement à une fréquence définie de manière à ce que l'œil humain ne puisse détecter le scintillement. Pour cela, le Reference Manuel nous indique que les 8 chiffres doivent être pilotés une fois au moins toutes les 16 ms pour une fréquence de rafraîchissement de 62 Hz. Pour illuminer un segment, les signaux des cathodes et des anodes doivent être réglés à un niveau bas.

6.2 Motifs affichés

Nous allons utiliser l'afficheur 7-segments pour connaître en temps réel l'inclinaison de la carte mais aussi pour repérer dans quel état de la machine d'état, nous sommes.
Sur l'afficheur 7-segments de gauche, nous allons afficher le degré d'inclinaison de 0° à 180°. Lorsque que le FPGA est posé à plat, le degré d'inclinaison est de 90°.
Sur l'afficheur 7-segments de droite, nous allons afficher selon les états :

- **État INITIAL** : - - - -
- **État PLAY** : - - -]
- **État PAUSE** : | |

6.3 Architecture VHDL pour l'affichage

Nous avons l'architecture suivante (figure 9) :

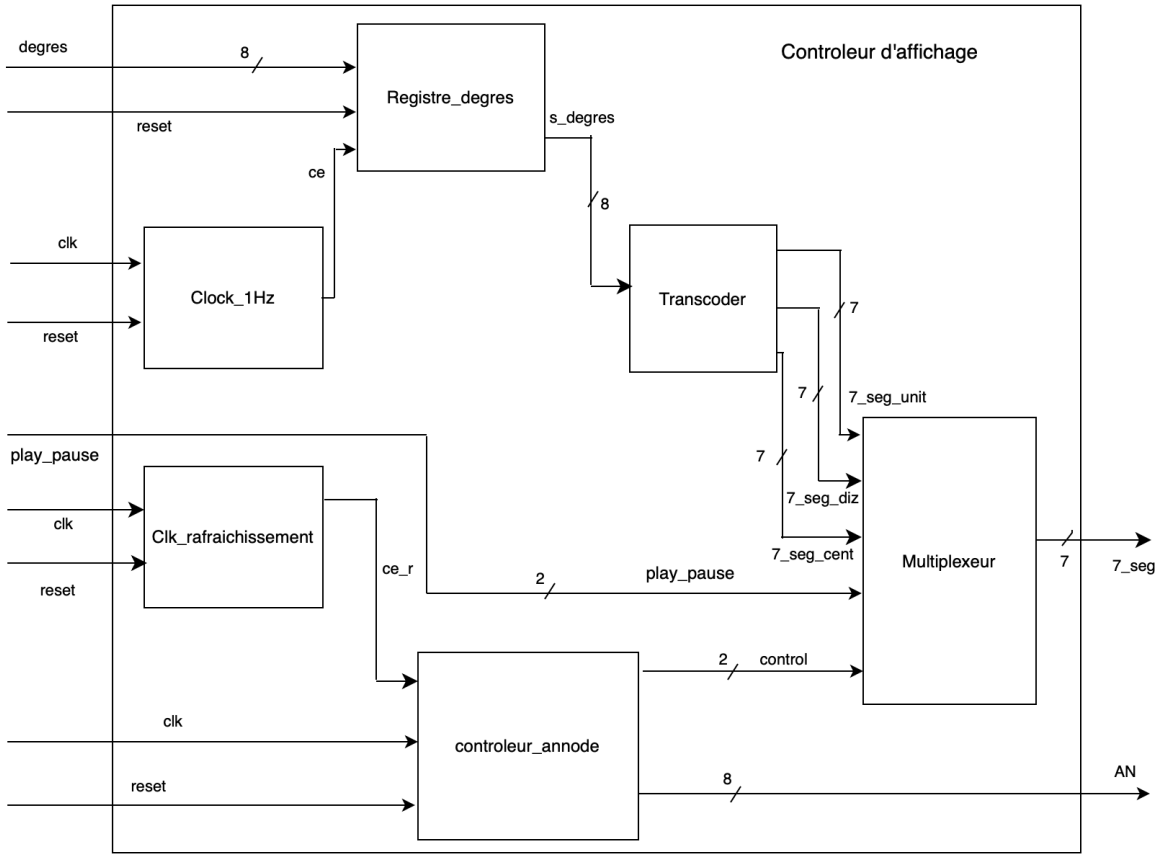


FIGURE 9 – Architecture du contrôleur d'affichage

Pour l'affichage des degrés, nous avons repris le registre et la clock à 1 Hz, utilisés dans la commande du servomoteur pour conserver la valeur des degrés jusqu'à ce qu'elle soit actualisée toutes les secondes. Cette valeur passe ensuite dans un transcodeur. Le transcodeur utilise un process asynchrone réalisé avec un script python et qui pour chaque degré donne les signaux des cathodes pour l'unité, la dizaine et la centaine de la valeur.
Comme expliqué plus haut, l'anode de chaque motif doit être activée continuellement à 1/8 de la fréquence de rafraîchissement qui est de 62 Hz. Ainsi, chaque anode doit être activée au maximum durant 2 ms. La clock associée à cette opération rythme le contrôleur d'anode. Ce contrôleur se présente

sous la forme d'un compteur modulo 8, produisant en sortie les signaux d'anode et un vecteur de contrôle qui permet de déterminer quelle anode est actuellement activée.Enfin, le dernier bloc est le multiplexeur. En fonction de l'entrée *play_pause* et du vecteur de contrôle, les signaux d'anode correspondants sont associés.

7 Gestion des boutons

7.1 Boutons utilisés

La carte Nexys 4 est équipée de cinq boutons-poussoirs disposés en forme de croix. Pour notre projet, nous utilisons seulement le bouton BTNR et le bouton BTNL.
En appuyant sur BTNL, notre système passe en mode play. En utilisant BTNR, on active le mode pause, et si le bouton est pressé une deuxième fois consécutive, le système se réinitialise.
Les boutons-poussoirs génèrent une sortie basse au repos et une sortie haute lorsqu'ils sont pressés. Cependant, lorsqu'un bouton est pressé, il reste enfoncé pendant plusieurs périodes d'horloge. Ainsi, si la sortie de ces boutons est injectée en tant qu'entrée de notre machine à états finis (FSM), notre système changerait d'état à une fréquence de 10 MHz, ce qui n'est pas réalisable. De plus, ce n'est pas ce qu'on souhaite faire. Ainsi, un bloc de gestion de bouton est intercalé entre la FSM et chaque bouton-poussoir pour résoudre ce problème.

7.2 Architecture VHDL pour les boutons-poussoirs

Lorsque BTNR ou BTNL est à l'état haut, le signal de sortie passe à l'état haut sur un seul cycle d'horloge, puis reste au niveau bas jusqu'à ce que le bouton correspondant revienne à l'état bas.
Ainsi, nous avons l'architecture suivante (:

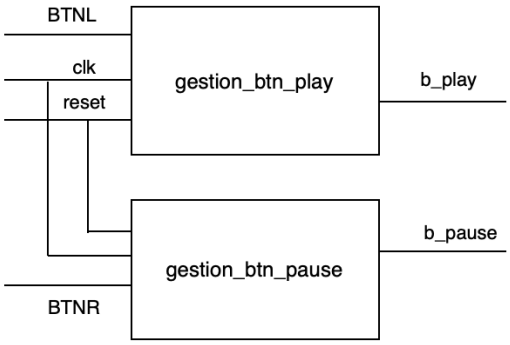


FIGURE 10 – Architecture de la gestion des boutons

8 Ressources

Nous avons fait en sorte d'utiliser et d'optimiser au mieux les ressources. L'analyse portera sur la consommation (puissance), la fréquence maximale possible, le nombre de LUT utilisés et enfin la surface du FPGA utilisée.

8.1 Puissance

Nous consommons au total **0.107 W**. Cette consommation se répartit suivant la figure 11.

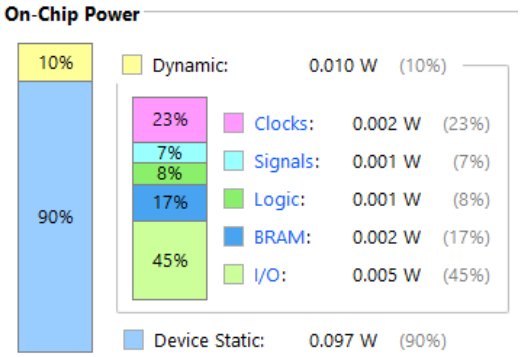


FIGURE 11 – Puissance utilisée lors de l'utilisation de la carte.

Il est important de relever que cette consommation est celle de la partie logique de la carte. Le servomoteur, qui consomme du courant, lui, n'est pas pris en compte dans cette analyse.

8.2 Temps de propagations

Ce qui est le plus intéressant et de connaître la fréquence maximale théorique de fonctionnement du FPGA. Pour cela, nous devons connaître le chemin critique, c'est-à-dire le temps de propagation le plus lent qui limitera la fréquence. Nous pouvons l'obtenir dans Vivado dans le rapport implémentations dans la catégorie Timing. Nous obtenons alors figure 12.

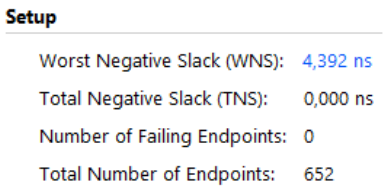


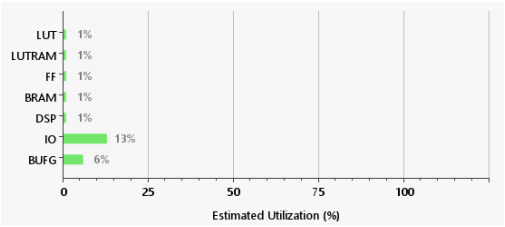
FIGURE 12 – Rapport des temps de propagation.

Nous avons un Worst Negative Slack de 4,392 ns. Cela nous contraint donc à avoir une fréquence de fonctionnement maximale théorique d'environ **227 MHz**.

Nous pouvons aussi lire sur la figure 12, qu'il y a eu 652 points de test lors de l'implémentation, et il n'y a eu aucunes erreurs, ce qui est plutôt positif.

8.3 LUT utilisés

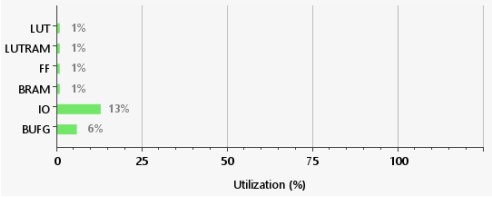
Nous pouvons voir les ressources logiques de la carte utilisée via le graphique ainsi que la table obtenue dans le résumé de projet. Cependant, les ressources utilisées post-synthèse et post-implémentation ne sont pas les mêmes. C'est ce que nous pouvons observer figure 13.



Resource	Estimation	Available	Utilization %
LUT	441	63400	0.70
LUTRAM	6	19000	0.03
FF	434	126800	0.34
BRAM	1	135	0.74
DSP	3	240	1.25
IO	27	210	12.86
BUFG	2	32	6.25

FIGURE 13 – Graphique représentant l'utilisation des LUT estimé (à gauches) ; tableau dénombrant les différentes ressources potentiellement utilisées (à droite).

Nous nous sommes rendu compte que l'implémentation arrive à optimiser les ressources et nous donne alors les données figure 14.



Resource	Utilization	Available	Utilization %
LUT	322	63400	0.51
LUTRAM	4	19000	0.02
FF	338	126800	0.27
BRAM	1	135	0.74
IO	27	210	12.86
BUFG	2	32	6.25

FIGURE 14 – Graphique représentant l'utilisation des LUT réelle (à gauche) ; tableau dénombrant les différentes ressources utilisées (à droite).

Nous pouvons donc y lire que l'implémentation permet d'économiser près de 120 LUT soit 27% des LUT initialement prévues. De plus, l'implémentation permet d'économiser des Flip Flop, 96 exactement, soit 22% des Flip Flop initialement prévues.

8.4 Surface du FPGA

Aux regards des résultats précédents, il est peu étonnant que nous utilisions peu de surface disponible sur le FPGA. Voici une image de la surface utilisée figure 15.

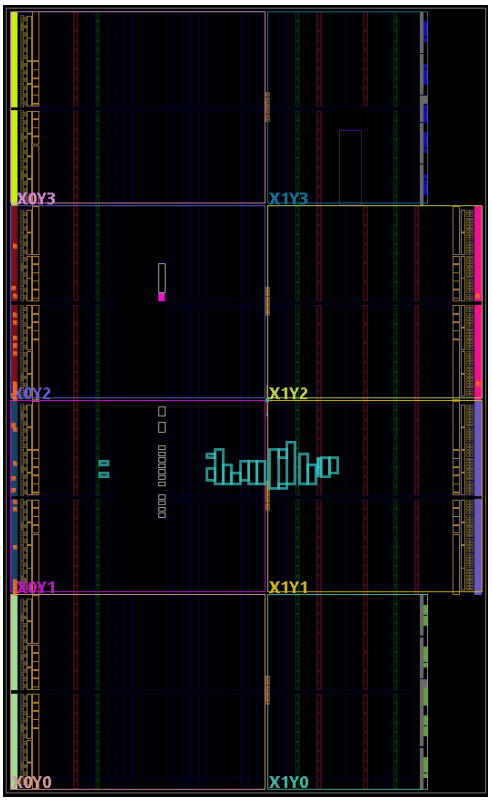


FIGURE 15 – Surface du FPGA utilisée.

Le résultat est donc cohérent vis-à-vis des mesures précédentes.

9 Conclusion

Nous avons du tout d'abords faire une architecture claire et précise afin de savoir quels blocs VHDL nous devrions mettre en place. Le projet a été pensé sous forme de blocs fonctionnels et testé individuellement, afin de faciliter le débogage. C'est pourquoi, il y a différent top-level intermédiaires, correspondants à chaque bloc. Ainsi, nous avons pu tester chaque sous-partie (simulation et test sur carte) avant de tout relier ensemble. C'est pourquoi il y a des tops levels intermédiaires dans le projet (i.e. Le test de la PWM qui s'est fait avec une rom et qui possède son top level intermédiaire pour le test.)

Pour conclure, nous réussissons à contrôler un servomoteur avec l'accéléromètre. L'objectif principal a été atteint. De plus nous avons l'affichage sur les 7 segments qui fonctionne correctement et permet de montrer l'angle de la carte et l'état dans lequel nous sommes. Nous avons également un code sur les LEDs qui permet d'afficher l'état actuel. Le seul objectif non atteint est l'indication sur le 7 segments du sens de rotation.

10 Bibliographie

Nexys A7 Reference Manual :
https ://digilent.com/reference/programmable-logic/nexys-a7/reference-manual