Haute Ecole d'Ingénierie
Hochschule für Ingenieurwissenschaften

HES-SO // Valais - Wallis

Projet : Cursor

Commande de moteur par FPGA

Simon Donnet-Monay & Rémi Heredero
23/01/2022

Table des matières

1 Introduction

Dans le cadre des cours d'électronique numérique nous devons en fin de semestre réaliser un projet. Ce projet consiste à réaliser le design d'un curseur. Celui-ci-doit se déplacer à une position pré-enregistrée avec une certaine accélération et un certaine décélération. Nous devons réaliser le design de la partie logique de notre système. Pour ce faire, nous utilisons le logiciel « HDL Designer ». Les instructions et la documentation complète se trouve sur Cyberlearn dans la section « Cursor ».

Nous avons réalisé ce projet en binôme et avons utilisé GIT afin d'optimiser notre collaboration. Nous avons réalisé ensemble le design général décrit ci-dessous, puis avons travaillé chacun sur des parties spécifiques du schéma général.

2 Design général

Pour le design général, nous avons décidé de séparer notre top level (Figure 1) en 4 blocs. Un bloc qui gère la position du curseur en fonction des entrées de l'encodeur (« PositionBlock »). Un bloc qui gère les boutons pour éviter d'avoir plusieurs appui simultané (« ButtonBlock »). Un bloc driver qui gère la puissance et la direction du moteur (« driver2 »). Et enfin le bloc principal qui gère la logique globale du curseur (« Main2 »).

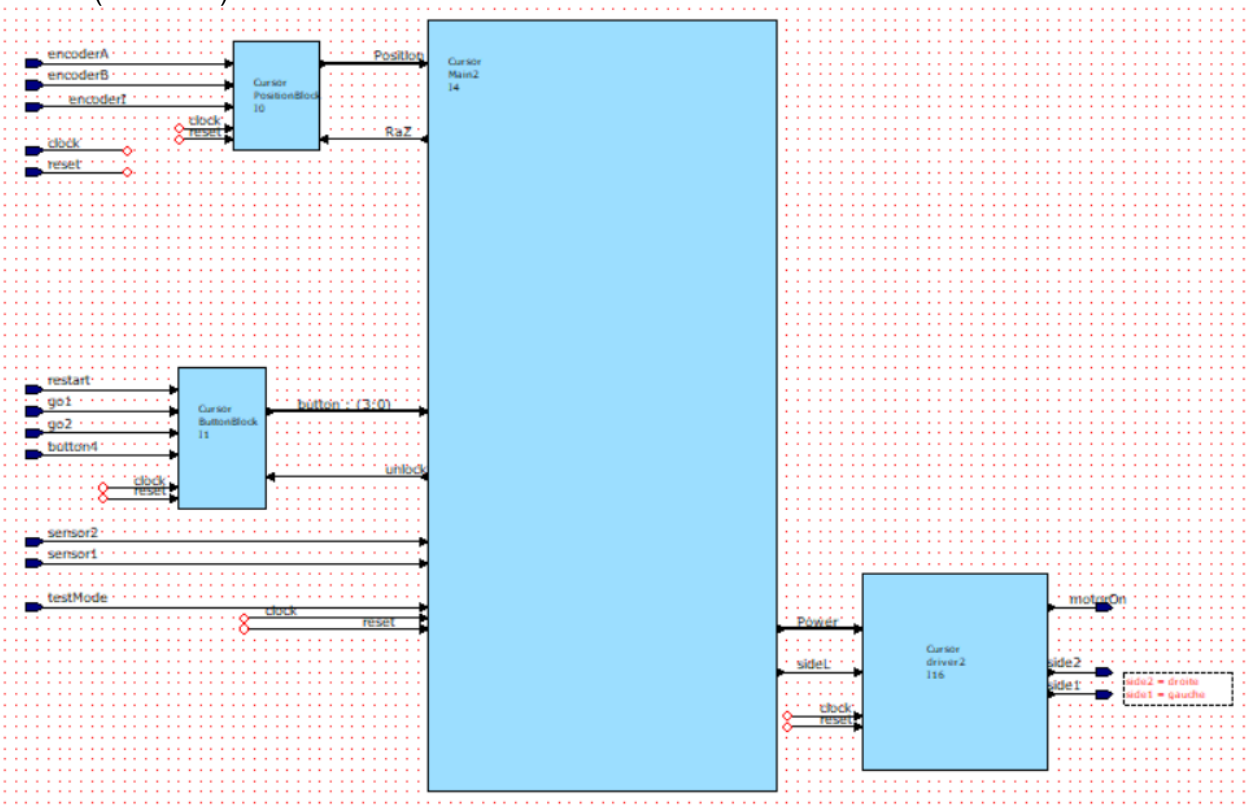

Figure 1 Top level

2.1 « PositionBlock »

Le « PositionBlock » (Figure 2) s'occupe de donner la position absolue depuis le reset. Il prend les signaux de l'encodeur de position en entrée ainsi que le signal reset. Sa seule sortie est la position codée sur 16 bits.

Le bloc est découpé en 2 grande parties. La première s'occupe de générer un signal de comptage ou de décomptage en fonction du sens de rotation de l'encodeur et donc de la direction du curseur.

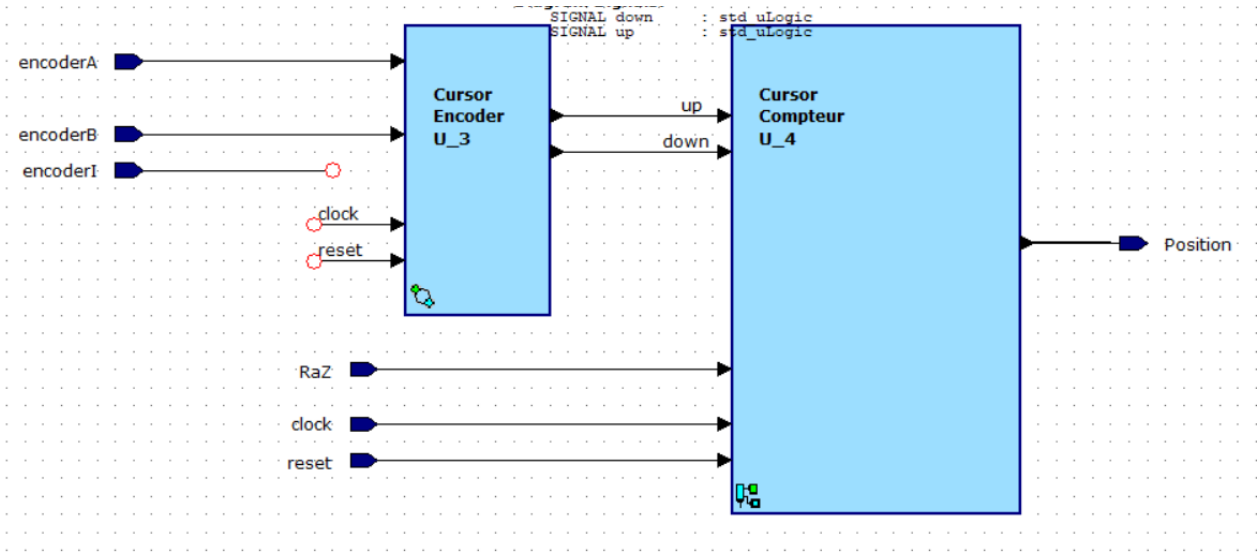

Figure 2 Architecture « PositionBlock »

Pour savoir s'il faut compter ou décompter, nous regardons dans quel sens tourne le codeur incrémental. Comme il possède 2 bits, il suffit de regarder quel est l'état de ces deux bits les uns après les autres (Figure 3[1]). Cette partie « Encoder » transmet simplement un signal s'il faut incrémenter la position ou la décrémenter (Figure 4).

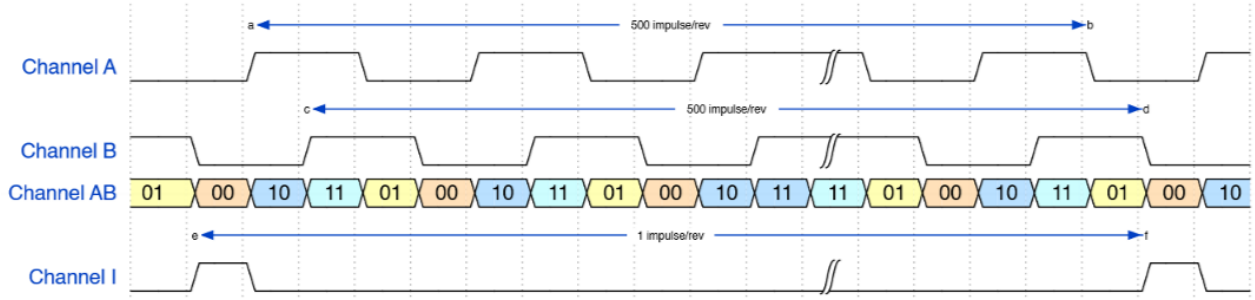

Figure 3 Signaux du codeur incrémental

[1] Graphique provenant de la donnée (Figure 10)

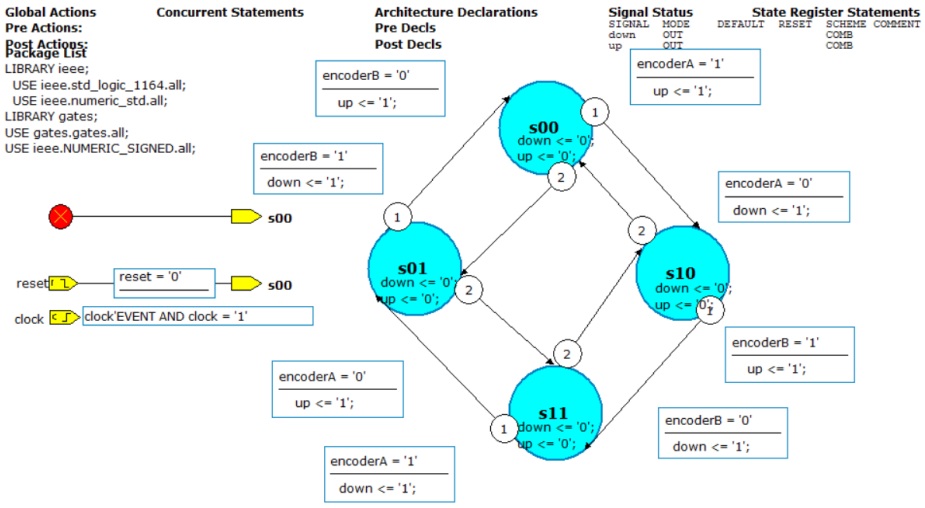

Figure 4 FSM encoder

Pour calculer la position absolue, nous devons déjà connaître la valeur maximale de notre compteur. L'encodeur aillant 500 positions par tour pour chacun des deux signaux, nous avons donc un changement de bit 2000 fois par tour. Le pas de vis étant de 1,75 [mm], Nous pouvons calculer qu'il y aura 11'430 changements pour 1 [cm] soit 171'450 pour les 15 [cm] de course totale du curseur. Nous avons donc besoin d'aller au moins jusqu'à 0010'1001'1101'1011'1010 pour faire rond 20bit.

<u>Remarque :</u> À la suite d'une erreur de calculs lors de notre toute première version, nous avons réalisé un compteur sur 24 bits. Comme nous l'avons créé sur 24 bits et que cela ne change rien au process, nous gardons cette version a 24 bit, mais sachez qu'il est tout à fait possible (si ce n'est préférable) de faire le compteur sur seulement 20 bit

Pour déterminer cette position absolue, nous avons donc besoin de compter et décompter. Nous avons pour ceci créé notre compteur principal avec plusieurs sous compteurs. Pour commencer, le bloc primordial, le compteur 1 bit qui peut compter ou décompter et possède une remis à zéro synchrone (Figure 5). Ensuite en en assemblant 4 et en connectant leurs bits de carriage à leurs entrées enable, nous avons un compteur 4 bit (Figure 6). Enfin la même chose avec 6 *(ou 5 si compteur 20bit)* de ces compteur 4 bits pour construire notre compteur total (Figure 7).

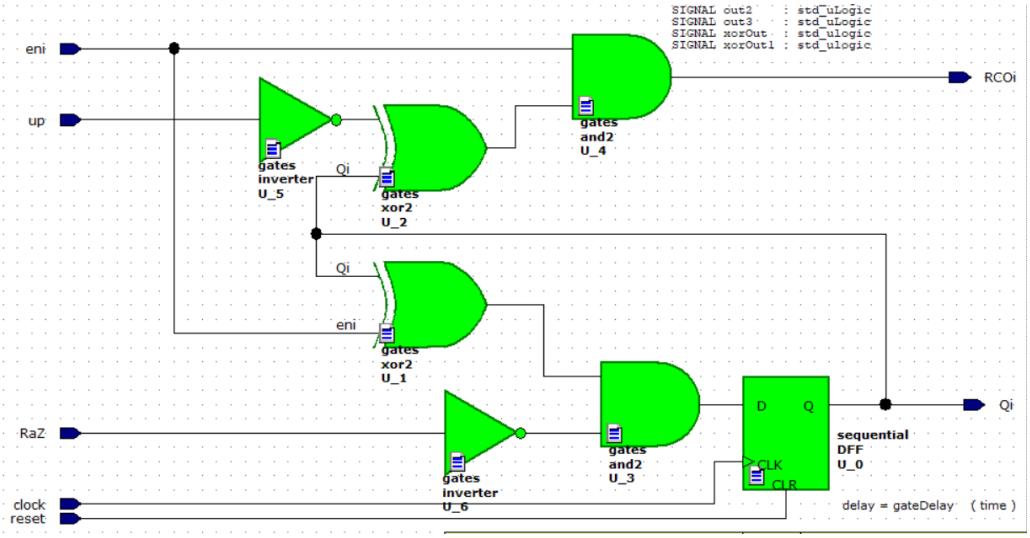

Figure 5 Counter 1 bit

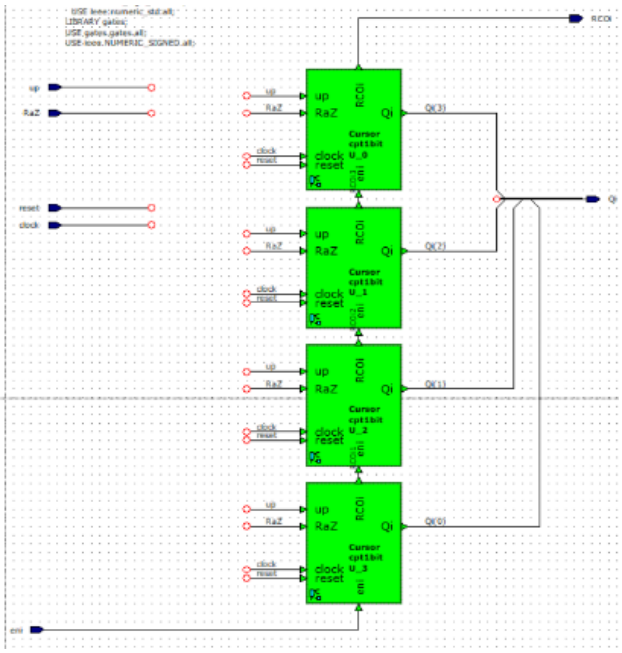

Figure 6 Counter 4 bits

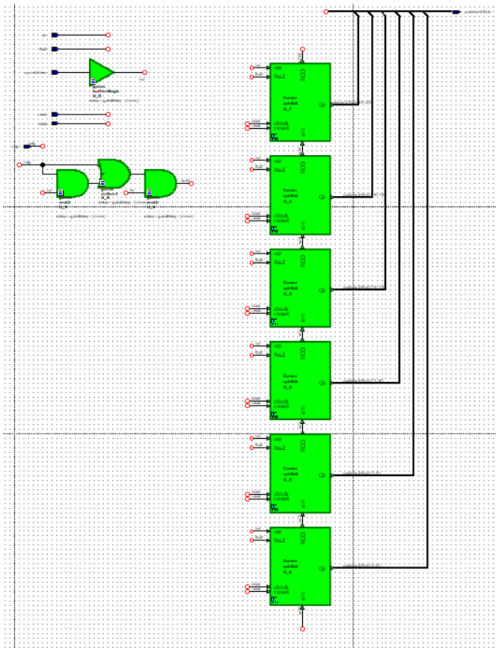

Figure 7 Counter 24 bits

Pour éviter un dépassement de bit, il faut bloquer le décomptage lorsque la valeur de ce dernier est à 0.
Pour ce faire, un signal est envoyé depuis un autre bloc qui fait la comparaison et renvoie si la valeur est à 0.
Lorsque ce signal est à 1, il bloque le décomptage. Ceci ce fait grâce à l'équation :

$$en * \overline{neg} + en * up * neg = en * (\overline{neg} + neg * up)$$

Cette équation est traduite en porte logique dans un zoom de la Figure 7 en Figure 8

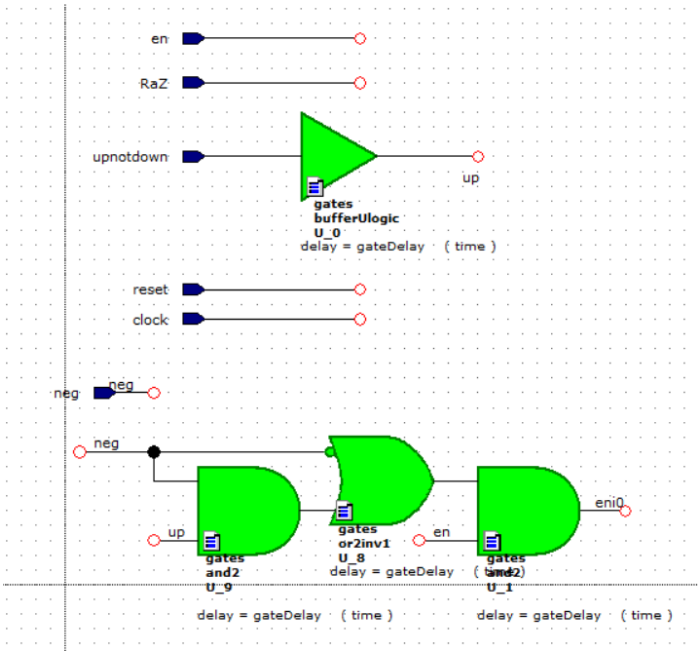

Figure 8 Blocage pour éviter le dépassement

Comme cette valeur est codé sur 24 bits (ou 20 bits), et que nous souhaitons travailler sur 16 bits pour plus de facilité et de clarté nous de prenons que les 16 bits de poids forts qui nous intéresse. Notre valeur maximale pour 15 cm codé sur 24 bits étant 0000'0010'1001'1101'1011'1010 on peut voir que les 6 bits de poids fort ne nous intéresse pas. Nous prenons donc les 16 bits de poids fort à partir de du 7e.
Nous perdons certes en précisions à éliminer les bits de poids faible, mais il s'agit là de doute façon d'une précision que nous ne pourrions pas atteindre dût à la mécanique du système. Avec cette réduction de précision nous avons donc une valeur de 2'857 par [cm]. Cette valeur correspond en binaire à 0000'1011'0010'1001.

Pour faire cette réduction, le signal est filtré par un bloc en FSM (Figure 9). Dans le cas général, nous prenons les bit 17 à 2. Si la valeur sur 24 bits était plus grande que notre course maximale, nous forçons la valeur maximale au signal de sortie de la position. Pour éviter un dépassement de bit lors du décomptage, nous envoyons un signal aux compteurs lorsque la valeur est à 0. Ce signal a pour effet de bloquer le décomptage comme vu précédemment dans la Figure 8.

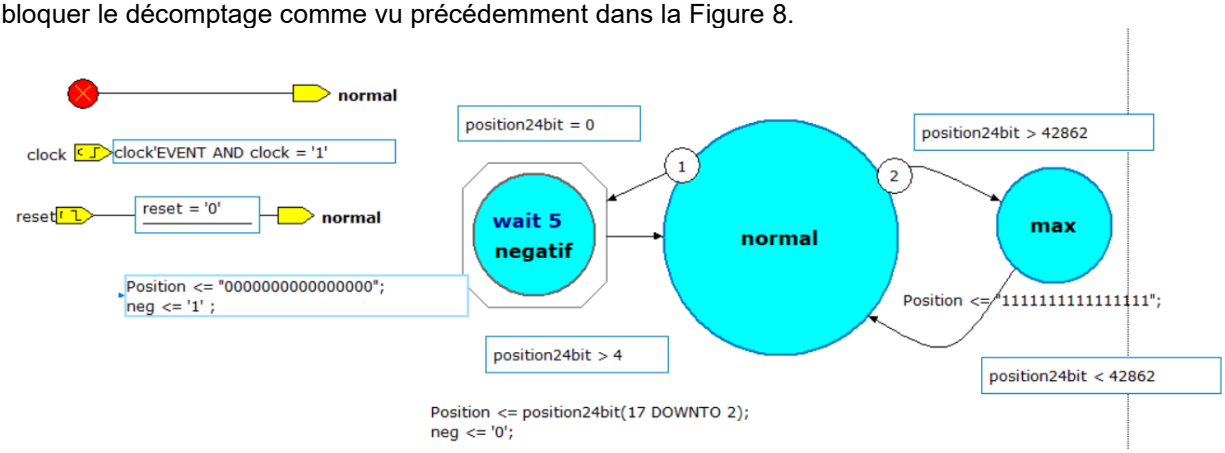

Figure 9 Filtrage de 24 bits à 16 bits

Cette façon de faire le comptage permet d'avoir à tout moment la valeur absolue de la position une fois passé le premier reset. Ainsi, il sera possible d'aller à la position 1 ou 2 depuis n'importe quelle position atteinte au moment de faire un stop par exemple.

2.2 « ButtonBlock »

Le bloc qui gère les boutons est très simple. Dès que l'un des boutons est pressé, nous entrons dans un état « d'attente » jusqu'à recevoir un signal de libération qui indique la fin du process engagé par l'actionnement du dit bouton (Figure 10). Le bouton 4 est une exception à ce système. Nous l'utilisons comme bouton d'arrêt, il est donc directement relié à la sortie (Figure 11). La sortie est un signal de 4 bit non signé avec dans l'ordre du poids faible à fort : le bouton reset, pos1, pos2, button4.

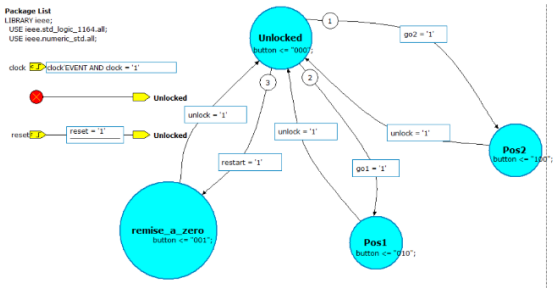

Figure 10 Stand by des boutons

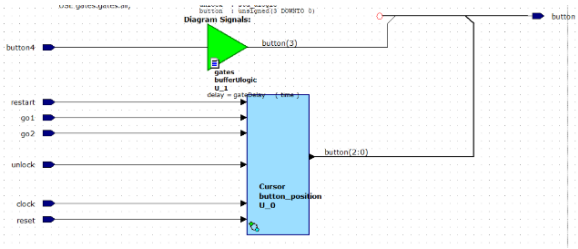

Figure 11 Bypass du button4

2.3 « Driver »

Le bloc driver sert à piloter le moteur selon un sens et une puissance. (Figure 12)
Il est composé de 5 parties. :

- Le premier bloc (« counterEnableResetSync ») est un simple compteur qui permet de créer un signal qui monte jusqu'à 255. Ce signal est reset par un autre bloc et sera utilisé pour la PWM.
- Un deuxième bloc « Counter_Controller » (Figure 13) fait compter le compteur de la PWM et du reset lorsque ce dernier a atteint sa valeur maximale. Il donne l'impulsion pour compter qu'une clock sur trois pour ne pas faire une fréquence trop élevée pour le pont H du moteur.
- Le Bloc « PWM » fait une comparaison entre la puissance voulue et la valeur du compteur afin de créer un vrai signal PWM. Lorsque la PWM est inférieur ou égal à la puissance désirée, le signal de sortie s'active, sinon il se désactive.
- Le dernier bloc « if0 » active le moteur si la puissance n'est pas nul.
- La dernière partie simplement un démultiplexeur qui met le signal de PWM sur la bonne sortie de direction du moteur en fonction de l'a direction souhaitée.

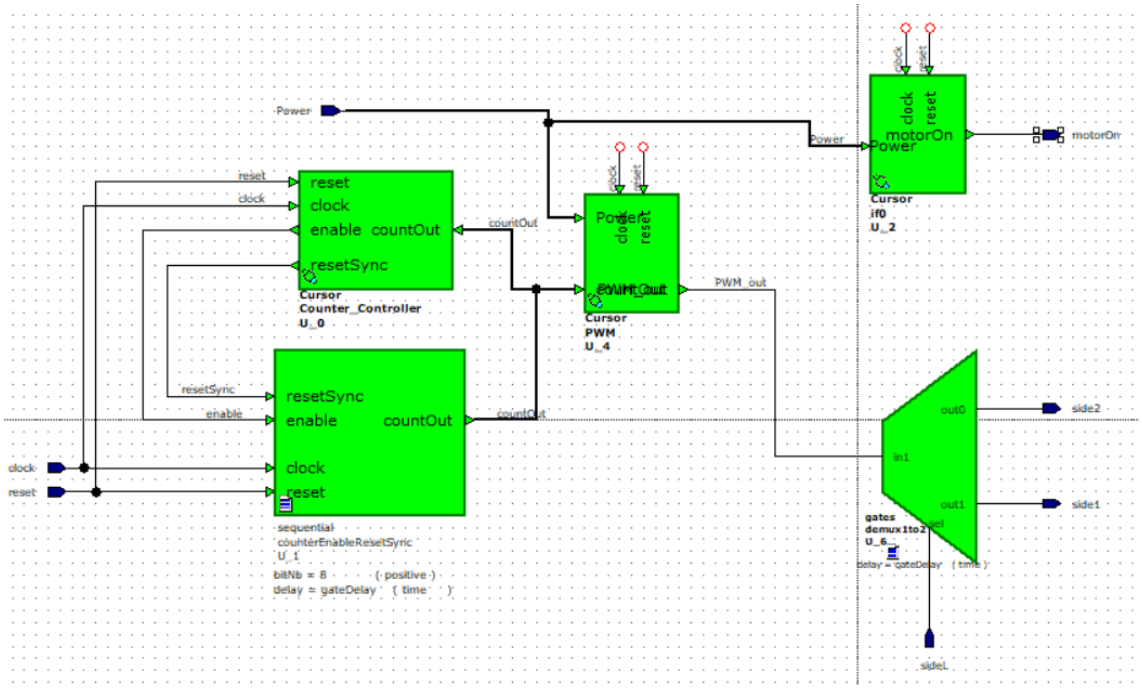

Figure 12 Architecture « Driver »

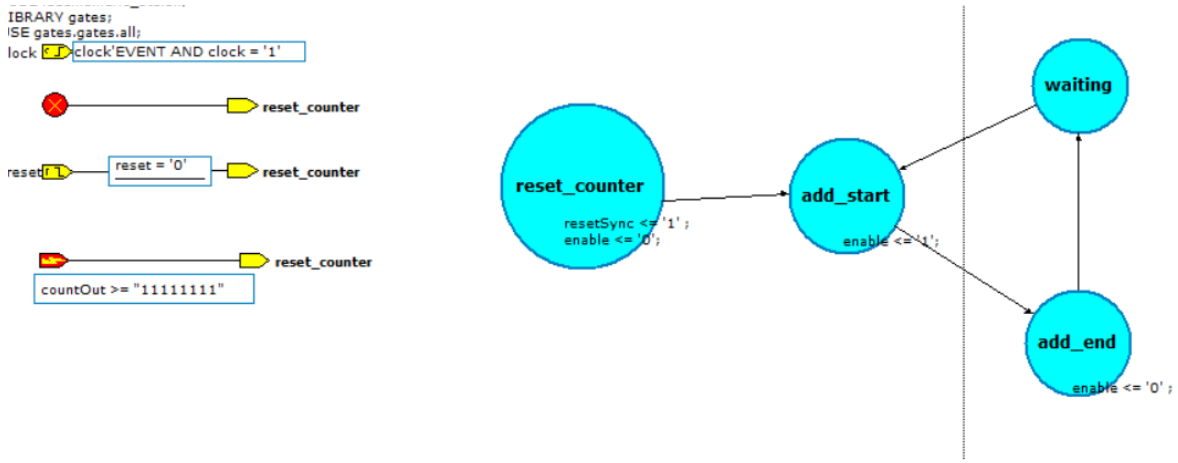

Figure 13 Bloc « Counter_Controller »

2.4 « Main »

Le bloc main est le cœur du système. C'est lui qui gère la logique d'accélération et de décélération en fonction de la position du curseur et la position choisie par les boutons.
Le diagramme ci-contre représente l'intérieur de notre bloc (Figure 14).

Lors de l'appuis du bouton Reset, si aucun mouvement n'était en cours, le signal power est fixé à fond, la direction est définie à gauche. L'arrêt est immédiat au capteur reed.

Lors de l'appui d'un des 2 boutons de positions, la position actuelle absolue est comparée avec la position désirée pour choisir la direction dans laquelle le curseur doit se déplacer. Ensuite, une phase d'accélération à lieu sur 1 cm jusqu'à la vitesse maximal. 1 cm avant position désirée, la phase de décélération commence jusqu'à l'arrêt à la position exact.

L'accélération et la décélération sont défini par 10 phases. La puissance minimum est à 25% afin de compenser le frottement. Le reste est séparé en 10 tranches. La position de déclanchement de la phase suivante est calculée sur une courbe x^2 afin d'avoir une accélération constante et non exponentielle

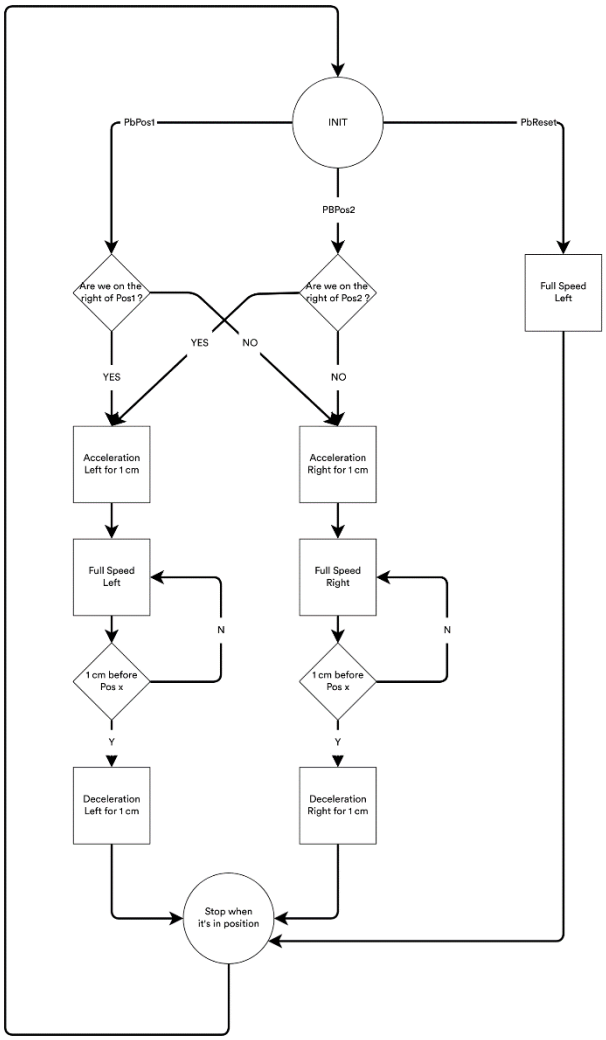

Figure 14 Architecture "Main"

3 Vérification et validation

Pour savoir si notre design est opérationnel, une simulation est un passage obligatoire. Elle sert à régler la majorité des problèmes que le circuit peut avoir. C'est aussi ici que nous pouvons voir si une erreur majeure de conception a été faite.

Pour faire les simulations, un bloc de test nous a été fournis *(*Figure 19 – *bloc bleu)*. Ce bloc contient des instructions précises qui sont envoyé en entrée de notre conception.

Les simulations que nous avons faites, ont été produite par le logiciel « ModelSim ».

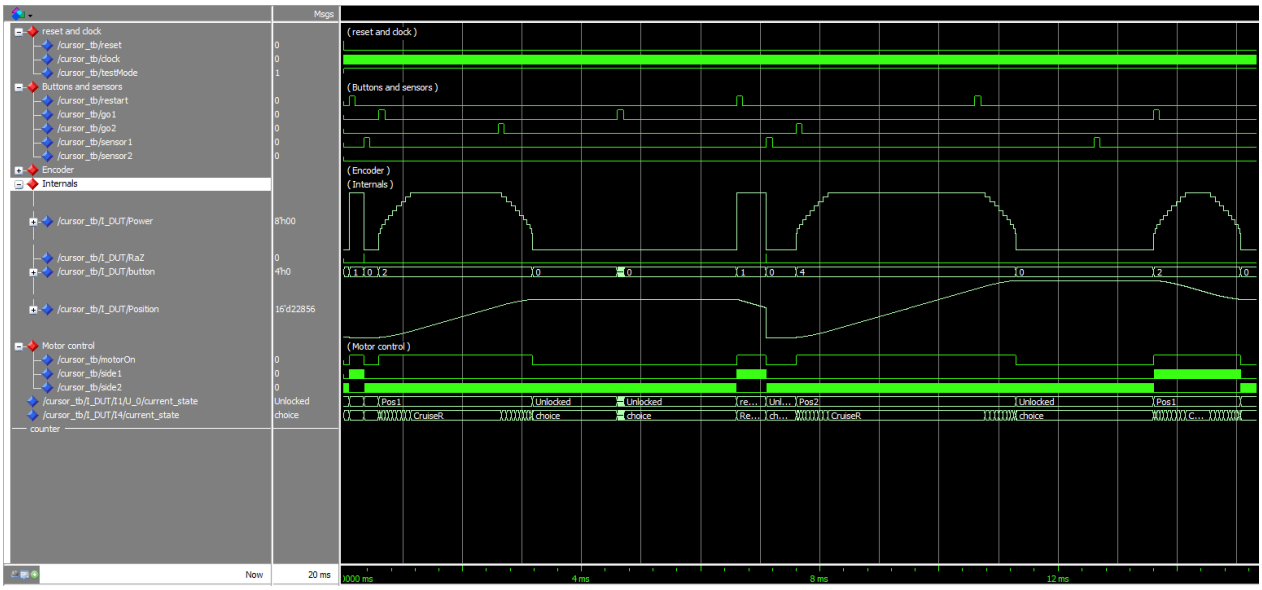

Figure 15 Simulation vue global

3.1 Simulation Initialisation

Tout d'abord, il faut que le circuit soit dans une position stable lors d'un reset.

Nous voyons (Figure 16) que lors d'un reset (premier signal) tous les signaux devienne connu et, dans notre cas, se mettent prêt à recevoir une commande de l'utilisateur. Nous pouvons donc dire que nos resets sont correctement programmé.

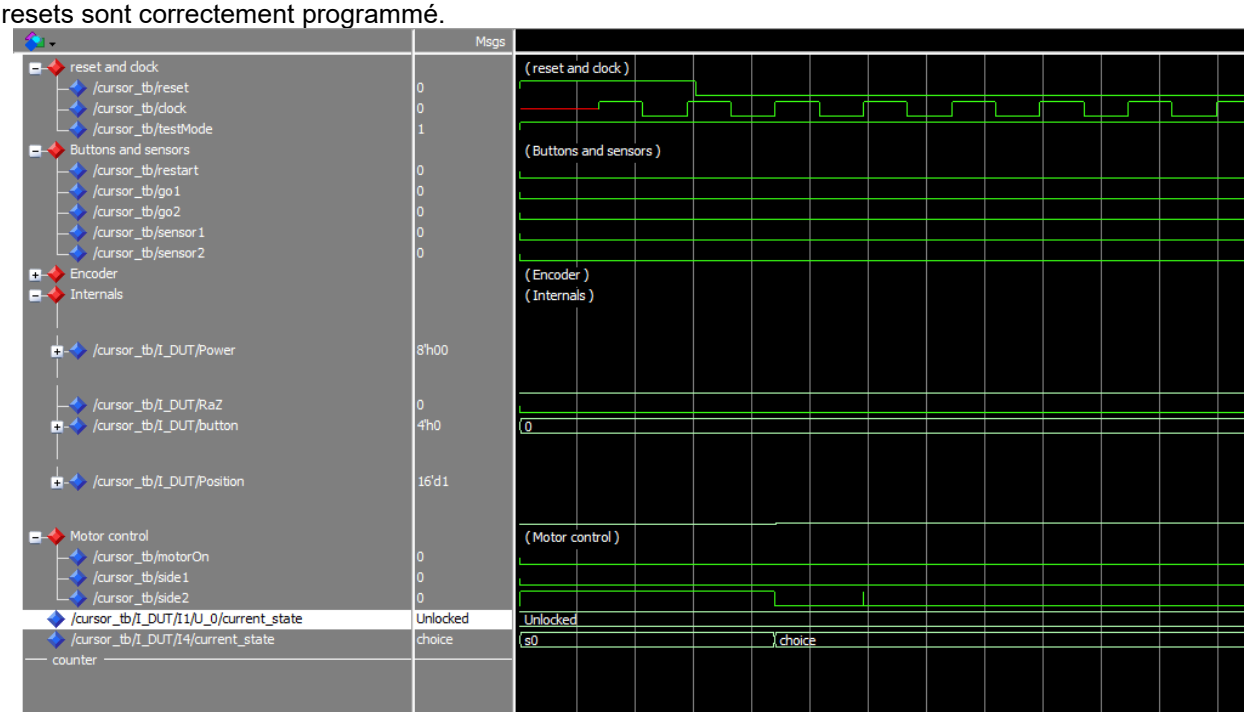

Figure 16 Simulation Reset

3.2 Simulation Position

Pour simuler la position, lorsque le signal « motorOn » la simulation fait que l'encodeur tourne dans le sens défini par « side1 » et « side2 ».

Sur Figure 15, nous constatons que la position augmente ou diminue en fonction des contrôles du moteur ainsi que peut être mis à 0 quand le signal de remise à 0 « RaZ » envoi une pulse. Le bloc qui gère la position est donc fonctionnel.

Pour rendre la durée de la simulation raisonnable nous avons pris les 16 premier bits de la position à la place de prendre des bits de poids plus fort (uniquement pour la simulation, sur le curseur nous avons remis la bonne plage de bits). Pour avoir une simulation pertinente il faut simuler chaque signal à une fréquence supérieure au clock du système, dans notre cas nous sommes obligés de réduire ce signal pour atteindre des temps de calcul de la simulation correct.

3.3 Simulation Boutons

L'appuis des boutons est simulé par le bloc de simulation et est traité par notre bloc « *ButtonBlock* ». Toujours sur la Figure 15, nous voyons que dans « Buttons and sensors » les inputs sont bien envoyé par le bloc de simulation et sur le signal « button » ils sont transmis au bloc « Main ». Ils ne sont pas tous transmis car le bloc « ButtonBlock » attend de recevoir un signal qui signifie que l'action demandé par l'utilisateur a été terminé.

Le quatrième bouton n'est pas simulé car ne contribue pas directement à la fonctionnalité du circuit.

3.4 Simulation Main

Pour vérifier si le bloc « Main » fonctionne, il faut surtout regarder les signaux (Figure 15*)* « Power », « RaZ » et « I4/current_state ». Power est la consigne de puissance que le bloc « main » envoi au bloc « driver ». RaZ est, comme écris précédemment, une pulse qui dit au bloc « Position » de remettre à 0 le compteur de la position. Enfin le signal « I4/current_state » est l'état dans lequel est la machine d'état.

En analysant tous ces signaux, nous constatons que tous se comporte comme prévu, les accélérations et les décélérations se font, la remise à 0 s'active au bon moment, et enfin que la machine d'état se met en attente lorsqu'elle a fini son action. Nous voyons aussi, indirectement, que le signal de déblocage des boutons et correctement envoyé car le signal « button » se met à 0 lorsqu'une action est terminée.

3.5 Simulation Driver

Vérifier ce bloc consiste à regarder si la PWM est généré correctement ainsi qu'à contrôler que les signaux « side1 » et « side2 » soit cohérant.

Pour la PWM, nous voyons (Figure 17*)* que lors d'une accélération un des signaux « side » passe d'un signal qui n'est presque pas actif à un signal actif quasi constamment. Nous voyons aussi que le signal « motorOn » s'active dès que nous voulons faire bouger le moteur.

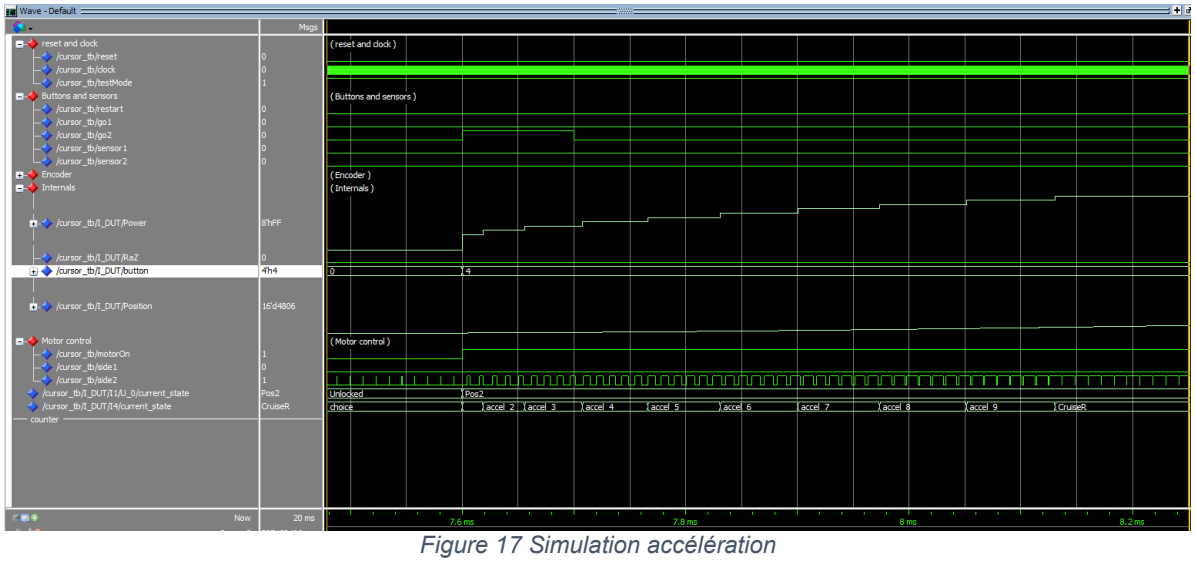

Figure 17 Simulation accélération

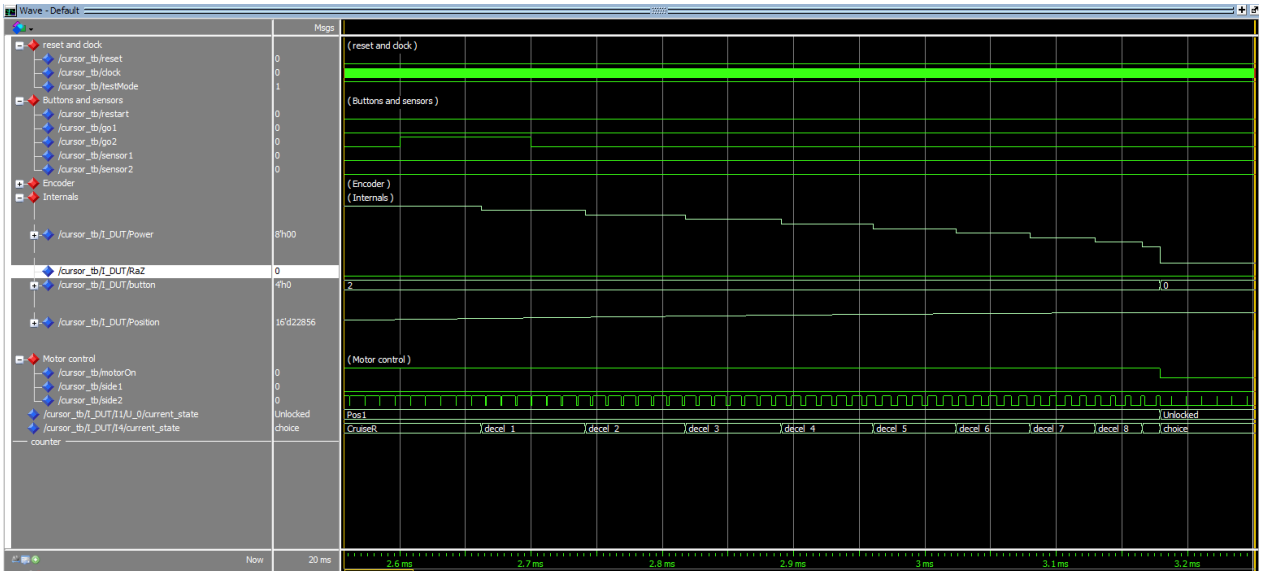

Figure 18 Simulation décélération

Sur cette capture (Figure 18), nous voyons qu'à la décélération il se passe l'inverse de ce qui se passe lors de l'accélération.

Les pics lorsque la PWM est censé être complètement activé ou complètement éteinte ne sont pas voulu mais ne posent pas de problème car ils sont si court qu'ils n'influent pas la vitesse du moteur, de plus dans le cas où le moteur ne doit pas bouger, le signal « motorOn » est bas ce qui assure que le moteur ne bougera pas.

En regardant la fréquence de la PWM nous constatons qu'elle est légèrement inférieure à 100 kHz, ce qui nous était demandé dans le cahier des charges.

3.6 Validation

Nous allons revenir une dernière fois sur la Figure 15 pour vérifier si tous les éléments ont fait les bonnes actions.

Tout d'abord le circuit ne bouge pas, ensuite il reçoit l'ordre de reset sa position. Il va donc sur la gauche à pleine vitesse et dès que le « sensor1 » envoie quelque chose, il s'arrête et envoie le signal de remise à 0 ainsi la position est à 0. Ensuite, le bouton « go1 » est appuyé, donc le système accélère pendant une certaine distance et continue d'avant à la puissance maximale jusqu'à une certaine distance avant la position 1 puis décélère jusqu'à précisément la position 1. Nous voyons aussi que les autres actions que les boutons veulent envoyer sont ignoré. Après nous voyons qu'étant donné que nous sommes déjà en position 1 lorsque le bouton « go1 » est appuyé le système de bouge pas. Ensuite, le système va recalibrer la position 0 sur la demande du bouton « restart ». Une fois ceci fait, le bouton « go2 » est pressé ce qui fait que le curseur va vers la position 2 avec une accélération et une décélération. Ensuite un fin de course s'active mais est ignoré car n'as aucun sens. Pour finir le bouton « go1 » se réactive ce qui fait que le curseur repart pour aller en position 1 et s'arrête à cette position.

Nous pouvons constater que tout a marché comme nous le souhaitions donc nous pouvons finalement essayer notre circuit sur le curseur en physique.

4 Intégration

Pour « programmer » un FPGA, il faut passer par plusieurs étapes qui vont être détaillée dans cette partie. Le terme de programmation n'est pas très correct pour parler d'un FPGA. Il ne s'agit pas de mettre du logiciel qui sera interprété mais de décrire la configuration d'un circuit électronique.

4.1 HDL-Designer

La première étape pour la réalisation du projet est de réaliser le design décrit plus haut sur HDL-Designer. Ce logiciel de Siemens permet de réaliser des circuits logiques de toutes sorte. Nous avons fait notre design de façon très graphique à l'aide de portes logique, de bascules, de machines d'états, …
Mais tout ceci n'est qu'une interface plus pratique pour de la description de matériel « VHDL ». Tous ces blocs que nous avons mis et décrit dans le chapitre « Design général » ne sont en réalité fait que de VHDL. Il est donc aussi possible de décrire directement tout notre design de cette façon.

4.2 Test Design

Le logiciel HDL-Designer permet aussi de réaliser des simulations comme expliqué au chapitre « Vérification et validation ». Pour ce faire, le logiciel doit compiler tout le design en VHDL pour l'interpréter dans la simulation comme un circuit numérique. En complément vient aussi un fichier de vérification de matériel qui permet de stimuler et valider le design. Dans la Figure 19 on peut voir le bloc vert qui est l'ensemble de notre design décrit au chapitre « Design général » et le bloc bleu qui est le fichier de vérification de matériel écrit lui aussi en VHDL. Une fois sur cette vue, il est possible de compiler pour passer au test.

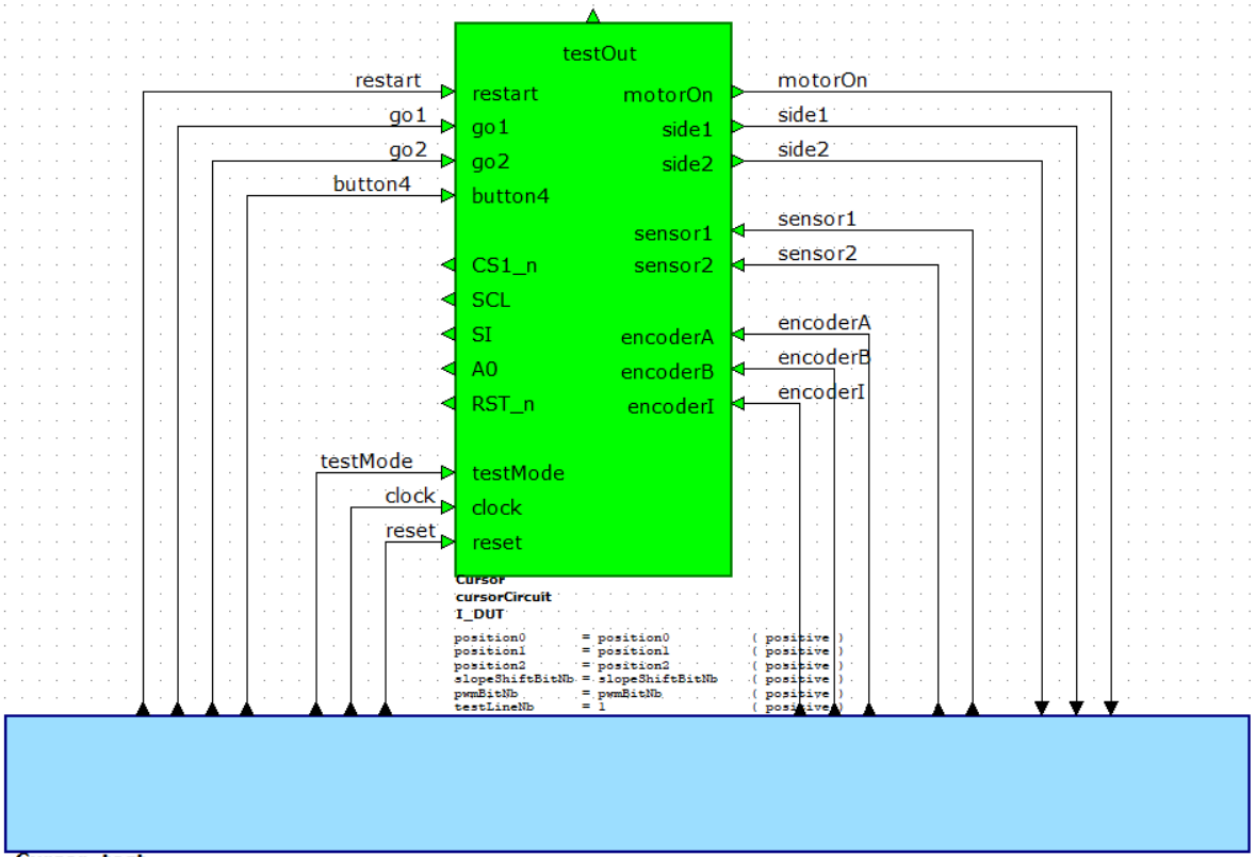

Figure 19 Root view pour la simulation

Pour préparer l'intégration dans le FPGA, il faut non plus se mettre sur la Root View, mais sur « FPGA_cursor » et lancé la « ModelSim Flow » comme dans la Figure 20.

Plusieurs étapes vont se faire. En premier, le logiciel va générer tous les fichiers en VHDL à partir du design. Puis il va compiler afin de vérifier s'il n'y a pas d'erreur, puis lancer la Simulation (ce qui n'est pas utile dans le cas du chargement dans le FPGA).

Figure 20 ModelSim Flow

4.3 *Préparation pour la synthèse*

Une fois notre design généré et compilé, il faut rester sur la même vue et faire « Prepare for Synthesis » comme dans la Figure 21.

Le logiciel va d'abord récupérer les fichiers généré précédemment, puis les assembler en un seul fichier VHDL par blocs qui contient des I/O. Ensuite, il va changer les librairies pour généraliser tous les composants. Comme nous souhaitons configurer un FPGA et non simuler un circuit logique avec des composant précis, il suffit d'indiquer qu'on utilise une porte AND général par exemple.

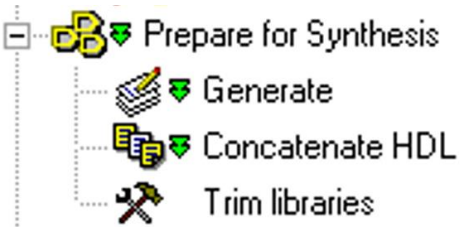

Figure 21 Prepare for Synthesis

4.4 *Synthèse*

La synthèse comporte de nombreuses étapes. La première se situe encore dans le logiciel « HDL-Designer ».
En double cliquant sur « Xilinx Project Navigator » comme sur la Figure 22 le logiciel récupère le fichier généré par l'étape précédente (« Préparation pour la synthèse ») pour l'updater aux spécifications liées au logiciel Xilinx.
Ensuite, il ouvre simplement le logiciel « ISE Project Navigator »

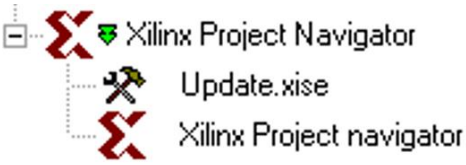

Figure 22 Xilinx Project Navigator

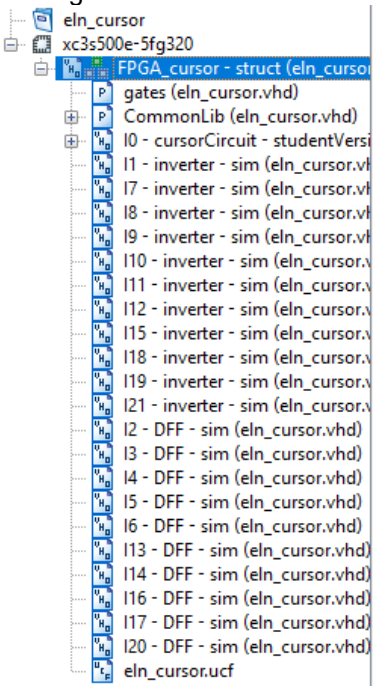

Figure 24 Hierarchy in ISE

Une fois le logiciel « ISE Project Navigator » ouvert, on peut y trouver l'architecture de notre projet (Figure 24). Tout en bas, on peut voir qu'il y a un nouveau fichier : « eln_cursor.ucf » Ce fichier UCF sert à faire la connexion entre les signaux I/O VHDL et les ports physique du FPGA. Selon la version de la board, il faut définir des ports différents en mettant en commentaire l'autre version de la board tel que sur la Figure 23.

```
 8  #----------------------------------
 9  # Buttons, V1 board
10  #
11  #NET "restart_n"     LOC = "E8" ;
12  #NET "go1_n"         LOC = "G9" ;
13  #NET "go2_n"         LOC = "F9" ;
14  #NET "button4_n"     LOC = "F7" ;
15
16  #----------------------------------
17  # Buttons, V2 board
18  #
19  NET "restart_n"     LOC = "G9" ;
20  NET "go1_n"         LOC = "F9" ;
21  NET "go2_n"         LOC = "F7" ;
22  NET "button4_n"     LOC = "F8" ;
23
24  #----------------------------------
```

Figure 23 Buttons on ucf file

Il faut le faire pour les boutons et les LEDs. Si l'écran LCD n'est pas utilisé, il faut aussi mettre toute la partie LCD en commentaire.

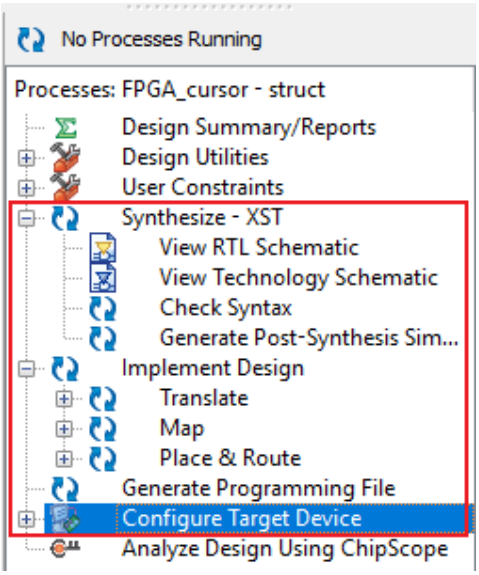

Figure 25 Processes ISE

Pour créer la configuration pour le FPGA il faut lancer « Configure Target Device » visible sur la Figure 25.

Ce process va lancer le processus « Synthesis » qui va adapter notre design pour les spécificité de notre FPGA.

Ensuite il lancera le processus d'implémentation qui va traduire le projet généralisé par l'étape « Préparation pour la synthèse » en bloc logique configurable (CLB), les mapper puis enfin choisir l'emplacement et faire la description du routage.

Enfin, le résultat de tout ceci donnera un fichier « .bit » qui est tout simplement le description de la configuration du FPGA.

4.5 *Configuration*

À ce stade de l'implémentation, il ne nous reste plus qu'à charger notre configuration dans le FPGA. Pour ceci, le logiciel iMPACT se lance automatiquement à la fin de la dernière phase. On peut le voir dans la Figure 26. Il faut cliquer sur Boundary Scan puis faire clique droite pour trouver le FPGA. On y voit à gauche la mémoire flash pour garder un programme après l'extinction de la board. À droite, se trouve la mémoire interne du FPGA pour charger une configuration. Il faut donc faire clique droite sur la mémoire de droite et ajouter un nouveau fichier de configuration. Dans le dossier « ise » se trouve le fichier « .bit » généré par l'étape précédente
Il ne reste plus qu'à faire à nouveau clique droite sur la mémoire de droite et faire « Program ».
Cette fois c'est bon, la configuration est dans le FPGA.

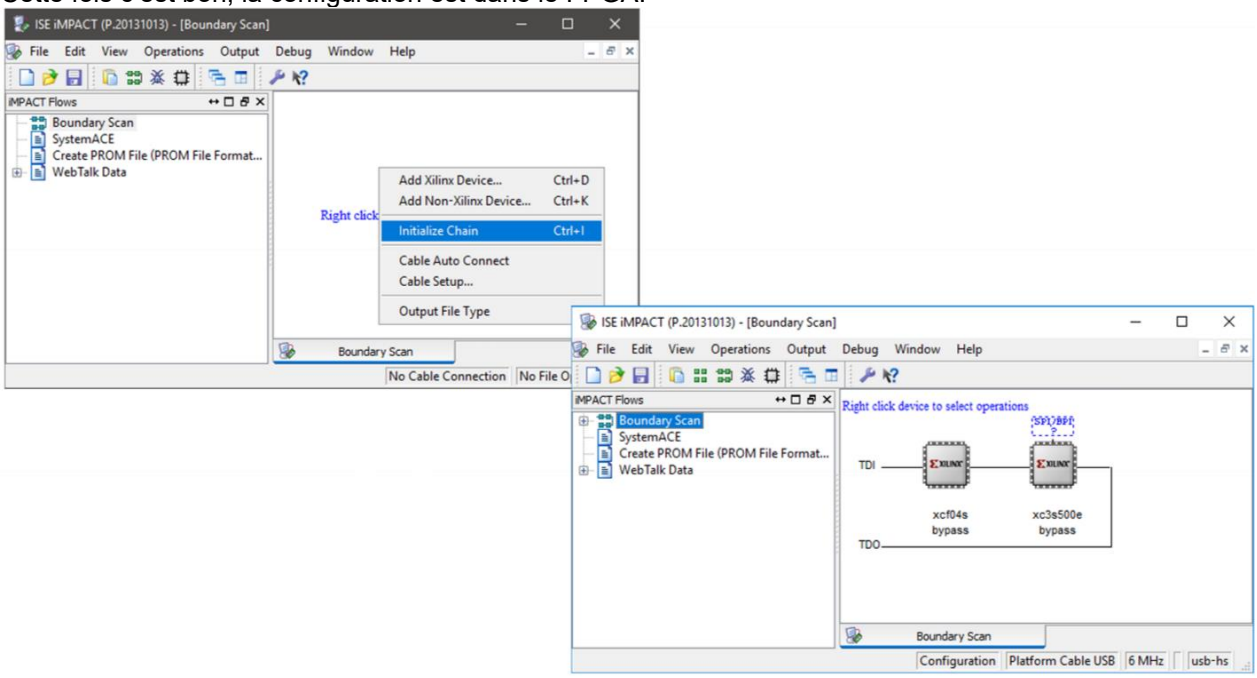

Figure 26 iMPACT

<u>Remarque :</u> Il est possible de faire en sorte que le design reste après extinction de la carte. Pour ceci, il faut générer le fichier de configuration de la ROM. Avec « Create PROM File » il est possible de transformer son « .bit » en « .mcs »,de la charger dans la mémoire de gauche puis de la programmer.

5 Conclusion

Ce design remplit le cahier des charges qui est de pouvoir aller dans 2 positions pré-enregistrée depuis une position 0 défini par un reset et de naviguer entre ces positions pré-enregistrées. En plus de ce cahier des charges, il est possible de s'arrêter à n'importe quel moment et de choisir une des 2 positions (pour autant que le reset ait été fait une fois au moins).

Avec une architecture de base relativement simple il est possible de faire des solutions efficaces qui fonctionne parfaitement avec une grande réactivité. Il faut cependant se méfier du matériel et des perturbations qui peuvent entrainer des bugs.

Les points d'améliorations sont donc évidemment une prévention des bugs lié aux interférences, mais aussi modifier le fonctionnement de certain bloc. Nous avons fait pleine confiance au logiciel pour réaliser des comparaisons, mais il pourrait ne pas réaliser cette comparaison comme nous le souhaiterions. Il serait donc préférable de réaliser par nous même un bloc de comparaison.

6 Signatures

Sion, le 23/01/2022

Simon Donnet-Monay

Rémi Heredero