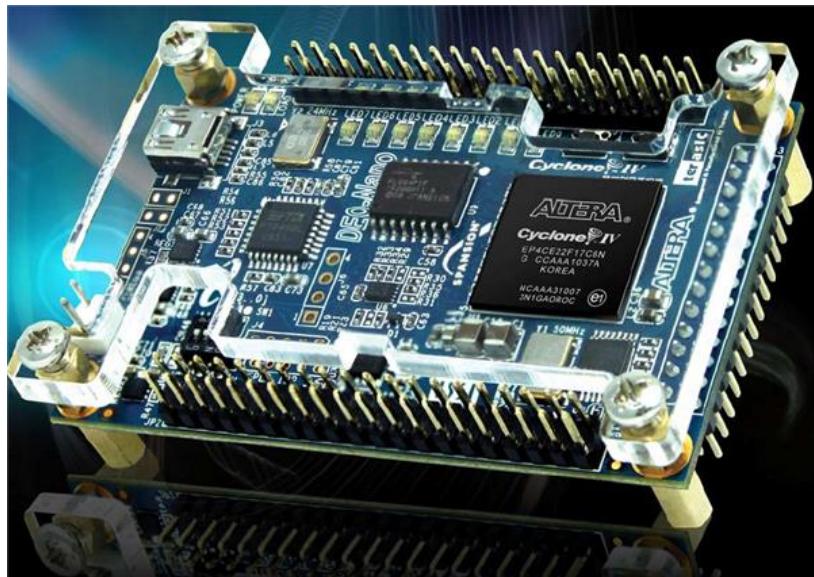


## BUREAU D'ÉTUDE : VHDL



Mohamed AIT-BRAHIM, Brendan ROBIN

Master 2 SME - 2022 - 2023

## Tables des matières

[I/Introduction](#)

[II/Fonction 1 : Mesure direction et vitesse du vent](#)

- [a\) Anémomètre](#)
- [b\) Girouette](#)
- [c\) Bus Avalon Fonction1](#)

[III/Fonction 6 : Pilotage du vérin](#)

- [a\) Présentation fonction 6](#)
- [b\) Pilotage du vérin](#)
- [c\) Bus Avalon Fonction 6](#)

[III/Fonction 7 : IHM](#)

- [a\) Pilotage des boutons](#)

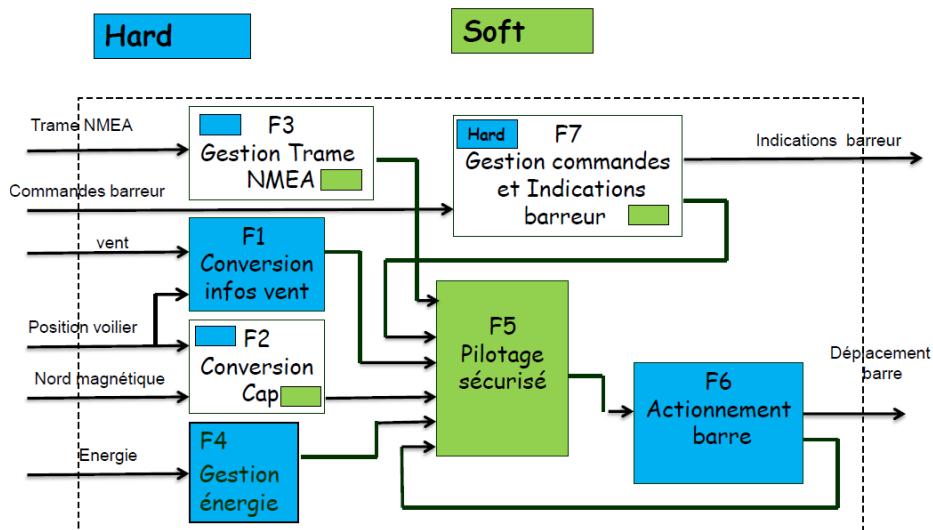
[III/Conclusion](#)

## I/Introduction

Durant ce bureau d'étude VHDL, nous avons travaillé sur la carte DE0-Nano, une carte de développement FPGA, avec une puce Altera Cyclone IV. Une maquette avec un vérin, un moteur, et une IHM était à disposition.

Pour ce qui est du logiciel, nous avons programmé sur Quartus 18.1, nous avons utilisé plusieurs fonctionnalités de ce logiciel, notamment Platform Design ainsi que Nios II Software Eclipse.

L'objectif de ce BE, va être de mettre en œuvre système de pilote de barre franche. Pour cela le système a été découpé en plusieurs fonctions distinctes.



Dans ce rapport, nous allons principalement nous intéresser à la fonction 1, la fonction 6 ainsi que la fonction 5. La fonction 5 est la communication entre le VHDL et le NIOS, via un bus avalon. Enfin nous avons aussi travaillé sur la fonction 7.

## II/Fonction 1 : Mesure direction et vitesse du vent

La première fonction que l'on va étudier, va être la fonction 1, elle va permettre de mesurer la vitesse ainsi que la direction du vent.

### a) Anémomètre

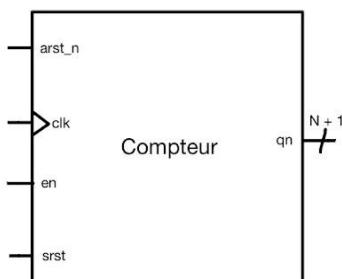
Pour mesurer la vitesse du vent nous avons à notre disposition un anémomètre qui va permettre de mesurer la vitesse du vent. Voici les caractéristiques de l'anémomètre:

#### => Vitesse du vent: Anémomètre



Type de mesure	Mesure de la vitesse du vent
Signal de sortie	logique fréquence variable 0 à 250 Hz
Alimentation	non alimenté
Capteur	type alternateur
Protection	IP65
Connexion	Connecteur étanche IP65 à 4 broches
Boîtier	Aluminium anodisé
Matière palette	Plastique
Protection électrique	Diode Transzorb
Filtres EMI	EN50081 EN50082
Résistance de charge	Min 2 KOhms
Limite de destruction	Supérieure à 75 m/s (270 km/h)
Température de fonctionnement	-30 à +70 °C
Plage	0-250 Km/h

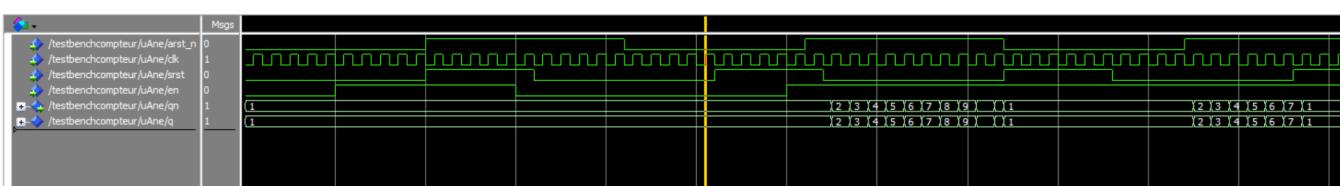
L'anémomètre en sortie délivre une fréquence variable qui varie de 0 à 250 Hz, en fonction de la vitesse du vent. Pour la mise en place de l'anémomètre, nous allons utiliser un bloc **Compteur**:



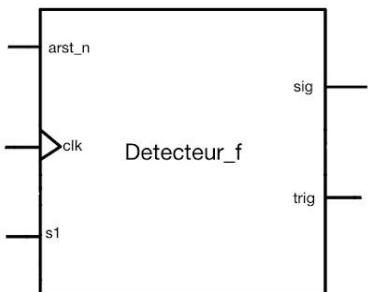
arst_n	clk	srst	en	q
0	*	*	*	0
1	↓	1	*	0
1	↓	0	1	q + 1
1	↓	0	0	q

TABLE 2 – Table de vérité d'un compteur

Simulation du compteur qui respecte la table de vérité ci-dessus:

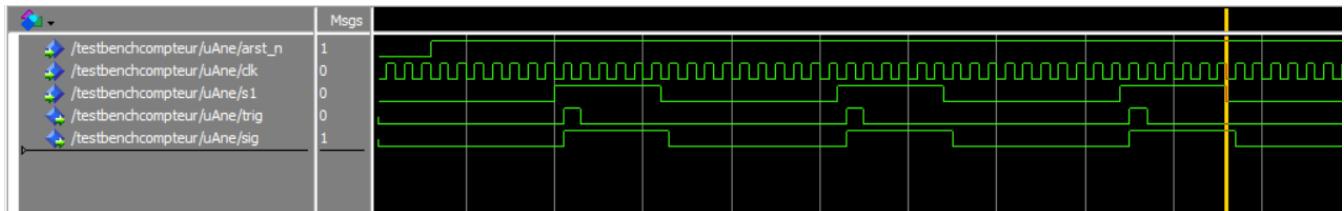


Et un bloc **Détecteur de front montant**:



Le signal  $s_1$ , correspond au signal que l'on veut détecter. Trig c'est le trigger de détection de front montant. Sig c'est le signal retardé d'un coup d'horloge de  $s_1$ .

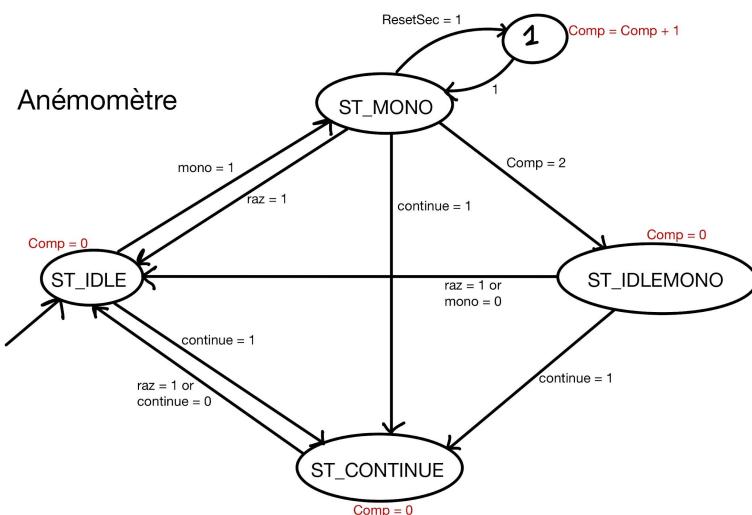
Simulation du détecteur de front montant:



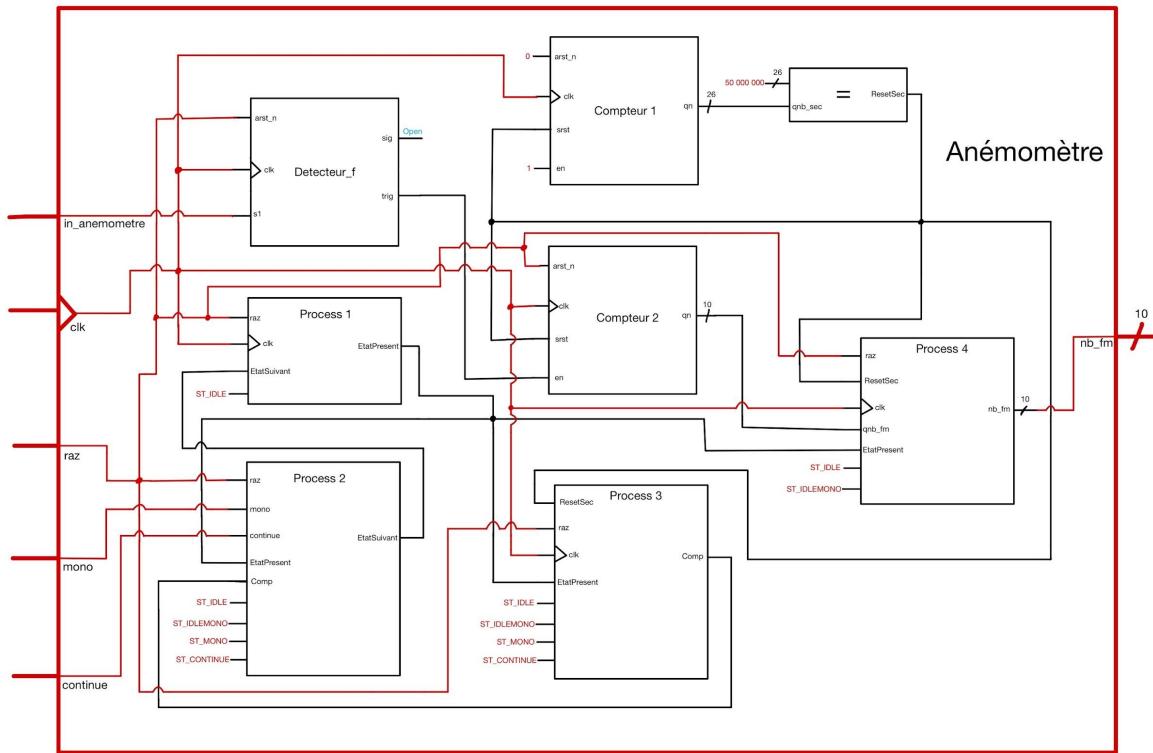
On peut voir dans cette simulation, qu'à chaque front montant, nous avons notre sortie trig, qui passe au niveau logique '1' sur un coup d'horloge.

Ensuite une machine à état à aussi été mis en place pour gérer les différents mode de fonctionnement de l'anémomètre :

- Le mode continue : Mesure en continue de la vitesse du vent
- Le mode Monocoup : Mesure unique de la vitesse du vent
- Le mode OFF : Mesure off



En utilisant nos deux blocs (compteur et détecteur de front montant) ainsi que la machine à état, voici le schéma fonctionnelle de l'anémomètre.



### Explication:

Le bloc Detecteur\_f, va détecter chaque front montant du signal à fréquence variable envoyé par l'anémomètre

A chaque détection, le Compteur 2 va s'incrémenter.

Le Compteur 1, va se charger de reset le Compteur 2 toutes les secondes.

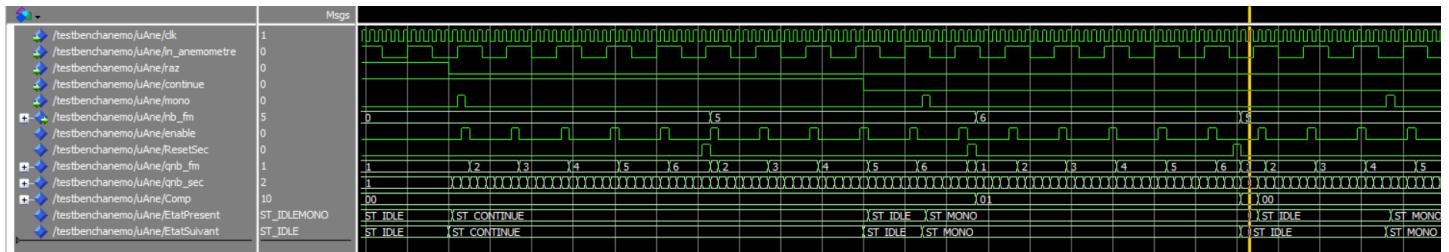
Le Compteur 2 va donc compter le nombre de front montant du signal envoyé par l'anémomètre sur une période d'une seconde. Nous allons avoir de cette manière la fréquence directement en Hz.

Le Process 4, met à jour toutes les secondes la valeur dans le vecteur nb\_fm qui correspond à la fréquence en Hz de l'anémomètre.

Les process 1, 2 et 3, correspondent à la machine à état vu précédemment.

## Synthèse et mise en œuvre des systèmes

Pour la simulation on va faire en sorte de diminuer drastiquement la période de reset du Compteur 2 générer par le Compteur 1, afin de nous éviter une trop longue simulation.



On simule un signal d'entrée dans `in_anemometre`, avec une fréquence quelconque.

On retrouve nos état mis en jours en fonction des signaux raz, continue et mono. La valeur de la fréquence est mise à jour toutes les secondes dans notre vecteur `nb_fm`. La seconde est comptée à partir d'un compteur (comme vu précédemment) on retrouve sa valeur dans le vecteur `qnb_sec`.

Le signal `enable` correspond au détecteur de front montant, qui vient incrémenter un compteur.

### b) Girouette

Pour mesurer la direction du vent nous avons à notre disposition une girouette qui va permettre de mesurer la direction du vent. Voici les caractéristiques de la girouette:

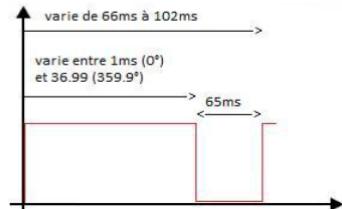
#### => Direction du vent: Girouette



Type de mesure	Mesure de la direction du vent avec une palette (girouette)
Signal de sortie	PWM
Alimentation	12 Vdc
Consommation	10 mA
Protection (position verticale)	IP65
Connection électrique	Connecteur étanche IP65 à 4 broches
Boîtier	Aluminium anodisé
Matière palette	Plastique
Protection électrique	Diode Transzorb sur toutes les sorties
Filtres EMI	EN50081 EN50082
Résistance de charge	Min 2 KOhms
Limite de destruction	Supérieure à 75 m/s (270 km/h)
Température de fonctionnement	-30 à +70 °C
Plage	0-360
Seuil de déplacement	0,25 m/s (0,9 km/h)
Linéarité	1 %
Résolution	0,4
Précision	3

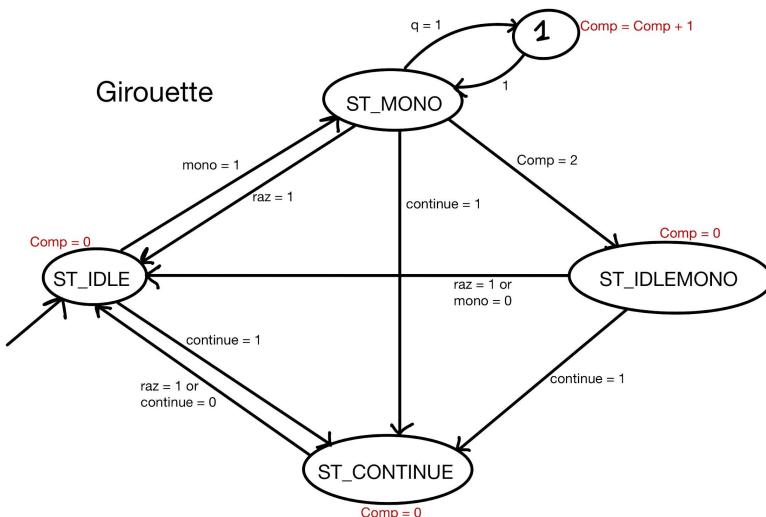
#### => Basé sur PWM: sortie numérique

- robuste
- très peu sensible aux parasites
- mise en œuvre facile

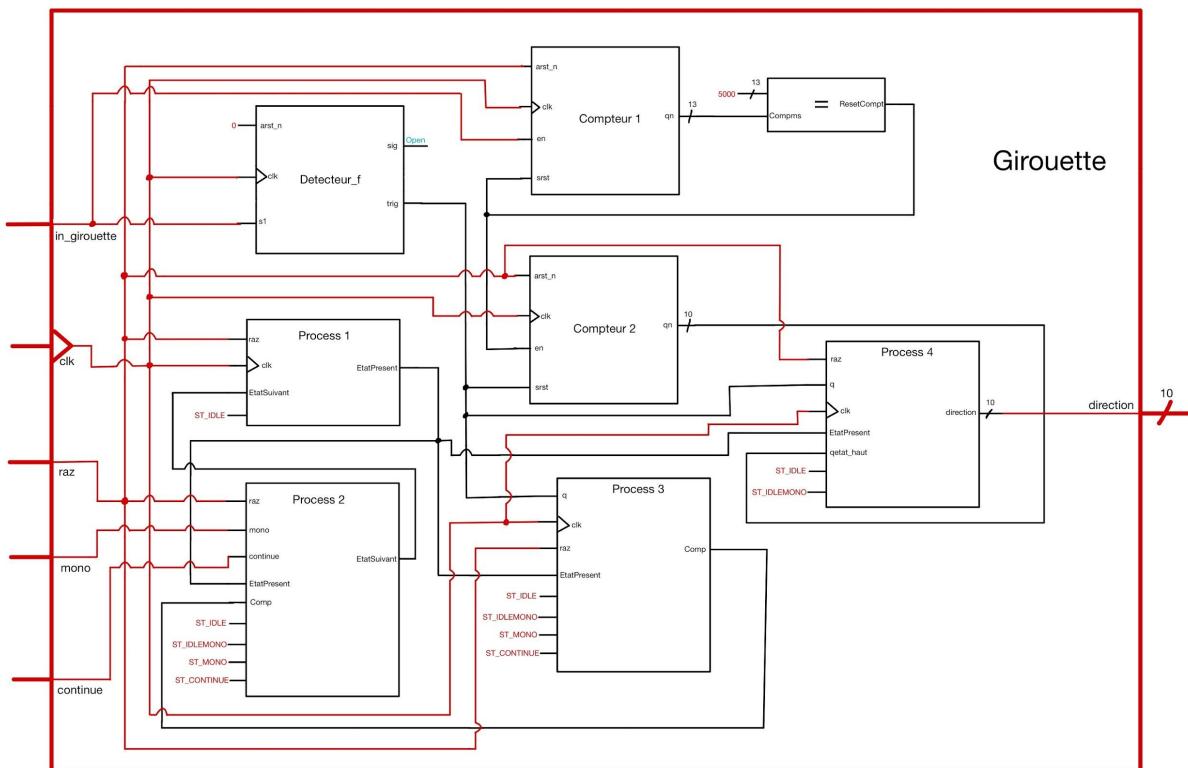


La girouette sort une PWM variable où la durée de l'état haut du signal représente la direction direction du vent, comme le voit sur le graphe temporelle ci-dessus, une durée à l'état de 1ms représente un angle de 0° et 39,99 ms représente un angle de 359,99°. Pour mettre en place la girouette, nous avons réutilisé nos blocs compteur de détecteur de front montant.

Comme précédemment avec l'anémomètre, nous avons également mis en place une machine à état afin de décrire le mode Continue, Monocoup et OFF de la girouette, la machine à état qui décrit son comportement est identique à celui de l'anémomètre.



En utilisant nos deux blocs (compteur et détecteur de front montant) ainsi que la machine à état, voici le schéma fonctionnelle de la girouette :



### Explication:

Le bloc Detecteur\_f sert à détecter le moment où le signal PWM passe à 1 pour remettre à 0 de manière synchrone le Compteur 2

Le Compteur 1 qui est un compteur qui est directement relié au signal PWM via sa broche enable, va compter tant que le signal PWM est à 1. Ce compteur réinitialise sa valeur toutes les 0.1 ms

Le Compteur 2 incrémente sa valeur toutes les 0.1 ms, comme la durée maximum de l'état haut du signal PWM est de 37 ms, la valeur maximum de ce compteur de 370.

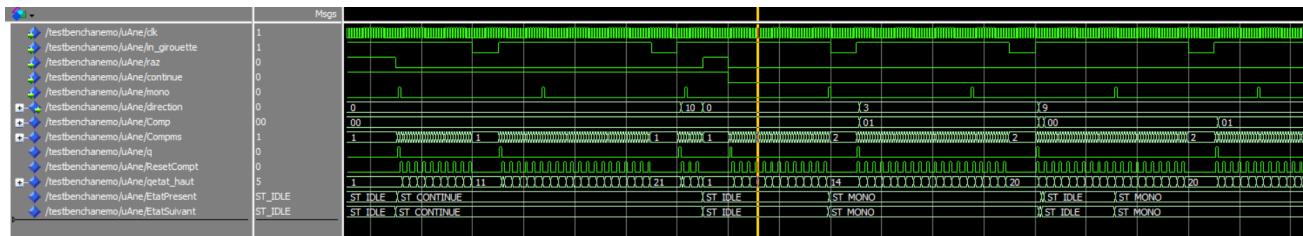
Ainsi pour avoir la valeur de l'angle en degré, il suffit juste de lire la valeur contenue dans Compteur 2, soustraire 10 à cette valeur, et on obtient directement l'angle.

Le process 4 sert à récupérer la nouvelle valeur de direction à chaque fois que le signal PWM repasse à 1, car cela indique une nouvelle acquisition à faire.

Les process 1, 2, et 3, comme avec l'anémomètre servent à implémenter la machine à état dans le code VHDL.

## Synthèse et mise en œuvre des systèmes

Pour la simulation de la girouette, on a réduit drastiquement le compteur de milliseconde, afin de pouvoir avoir une visualisation de tous les modes de fonctionnement de la girouette.



Nous retrouvons tous nos états pilote en fonction des états de raz, continue et mono, c'est le même principe que pour l'anémomètre.

La signal *q*, correspond à la détection de front montant du signal d'entrée (sortie de la girouette). En mode continue c'est à chaque front montant qu'on met à jour la valeur de la direction.

Le vecteur direction, correspond directement à l'angle de la girouette, en effet en comptant toutes les millisecondes, cela nous permet via un calcul simple de déterminer directement en logique l'angle.

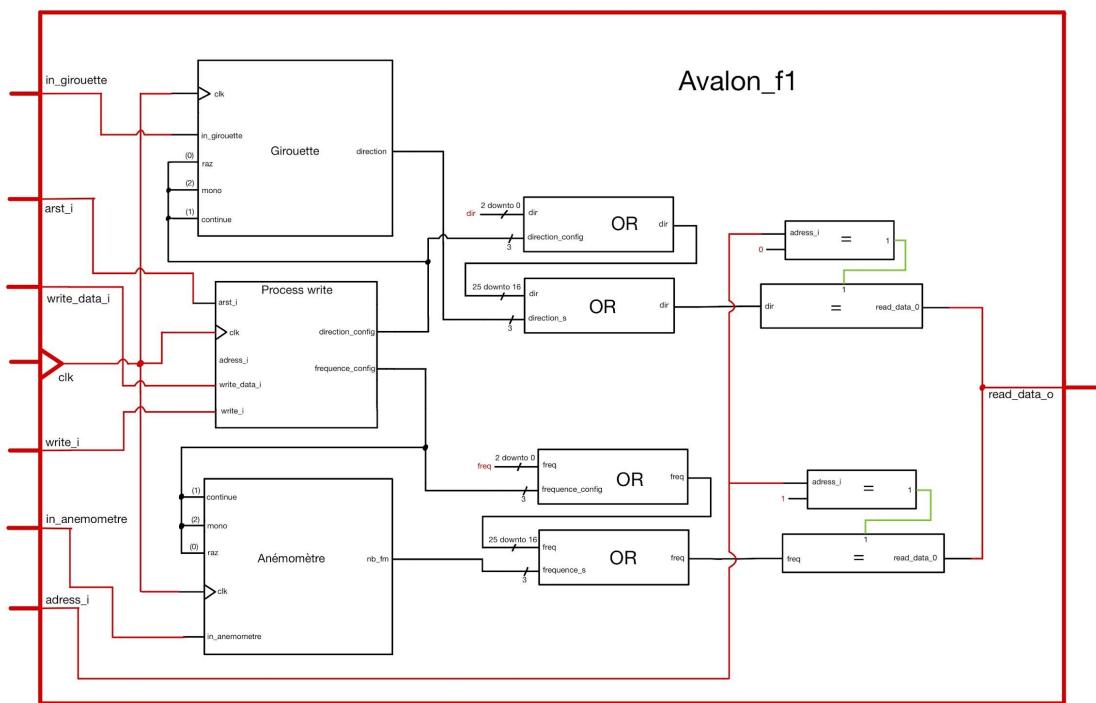
Le vecteur qu'on incrémente tous les millisecondes, afin de compter le temps de l'état haut, est *qetat\_haut*.

### c) Bus Avalon Fonction1

Une fois l'implémentation de l'anémomètre et de la Girouette terminé, il faut envoyer les données dans vers un CPU pour être traiter. Ce CPU pourra également commander le fonctionnement de l'anémomètre et de la girouette, s'ils doivent fonctionner en mode Continue, Monocoup ou OFF.

La première étape sera de créer, au sein de l'FPGA, un CPU, une mémoire intégrée (RAM ou ROM) et surtout un bus de communication entre le CPU, la girouette et l'anémomètre, ce bus s'appellera l'Avalon, et on aura un Bus Avalon différent pour chaque fonction. Pour créer le CPU et la mémoire intégrée on utilise un outil de quartus qui se nomme Platform Designer. On utilise également cet outil pour créer l'Avalon, mais à la différence du CPU et de la mémoire intégrée qui sont des composants déjà disponible dans Platform Designer, on doit créer l'Avalon de toute pièce en commençant par écrire son code VHDL puis importer ce code sur Plateform Designer pour créer et le composant.

Voici un schéma fonctionnelle de l'Avalon de notre fonction 1 :



#### Explication:

Les blocs Girouette et Anémomètre sont ceux que nous avons implémentés précédemment.

Le bloc Process write est le process qui écrit la configuration défini par le CPU sur la girouette et l'anémomètre.

Pour lire la donnée de direction venant de la girouette, ça se fera directement en combinatoire, on attendra juste que que address\_i vaut 0 pour envoyer la donnée vers le CPU.

Pour lire la donnée de direction venant de l'anémomètre, ça se fera directement en combinatoire également, on attendra juste que que address\_i vaut 1 pour envoyer la donnée vers le CPU.

## III/Fonction 6 : Pilotage du vérin

### a) Présentation fonction 6

La fonction 6 est la fonction en charge du contrôle d'un vérin. Cette fonction devra générer un signal PWM afin de contrôler la rotation d'un moteur qui à son tour fera tourner un vérin vers la gauche ou la droite, rapidement ou pas.

Pour implémenter cet fonction 6, on doit d'abord gérer la création du signal PWM qui servira à commander le moteur, puis on doit également gérer le contrôle des butées car en effet le moteur fera tourner le vérin si seulement l'angle du vérin se trouve dans la plage de valeur délimitée par les valeurs de butées. On aura également la gestion du convertisseur AN MCP 3201 avec lequel on lira l'angle en temps réel du vérin pour surveiller si notre angle se trouve toujours dans la plage de valeur voulue. Finalement il y aura l'interface Avalon qui servira de communication entre le CPU et le vérin.

Voici un schéma avec les différents blocs vhdl à développer:

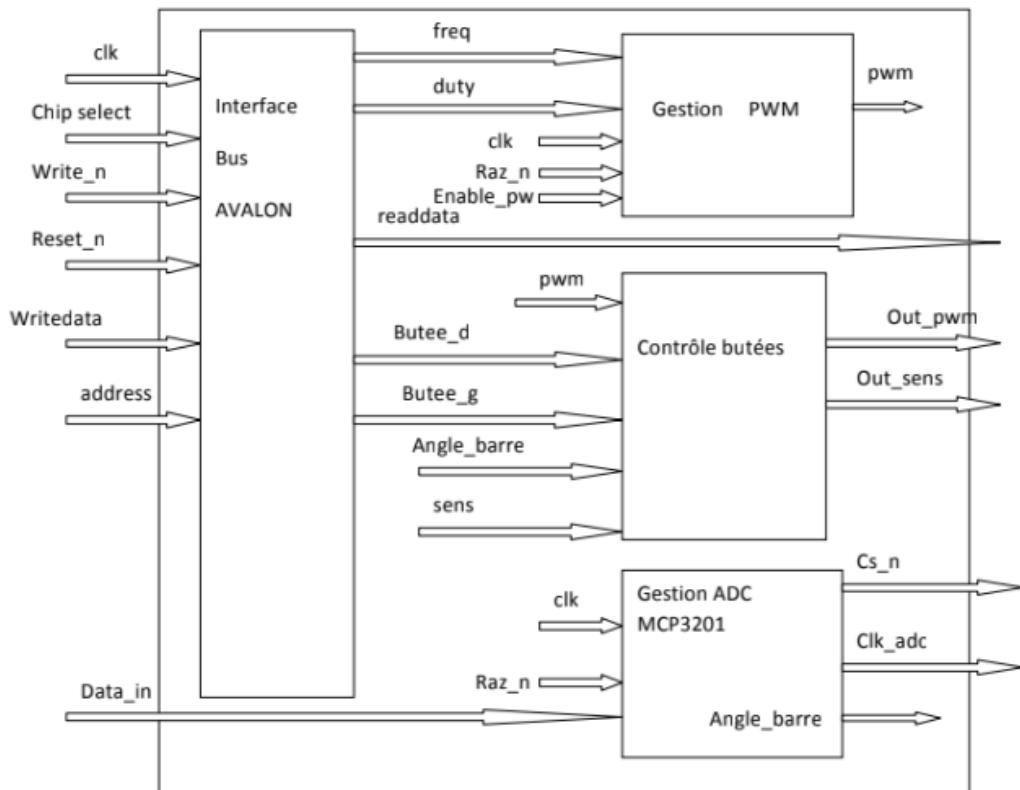
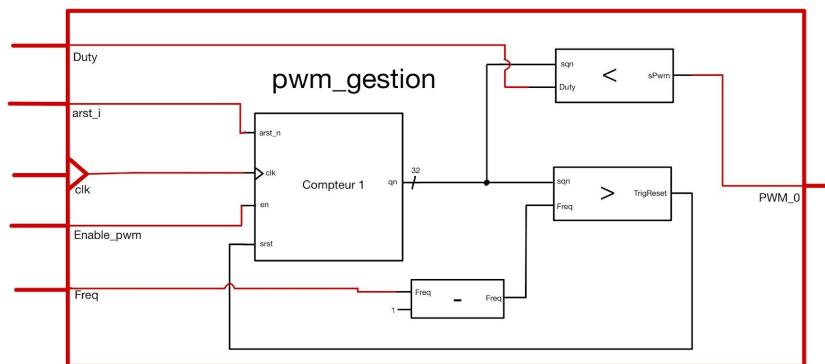


Schéma fonctionnel du circuit `gestion_verin`

### b) Pilotage du vérin

Le premier bloc que l'on va étudier va être le bloc de la gestion du signal PWM, voici le schéma fonctionnel associé à ce bloc:



#### Explication:

Compteur 1 comptera tout le temps sans jamais s'arrêter, on compare en permanence la valeur sortant du compteur avec la valeur du rapport cyclique (Duty) et de la fréquence (Freq). La valeur de Freq sera toujours supérieure à celle de Duty. Tant que la valeur du compteur est inférieure à celle de Duty, on laisse le signal PWM\_o à 1. Dès que la valeur de compteur devient supérieur Duty, on met le signal PWM\_o. La valeur du compteur continue à s'incrémenter et dès qu'il dépasse la valeur de Freq, on réinitialise le compteur et on reprend à 0.

#### Remarque :

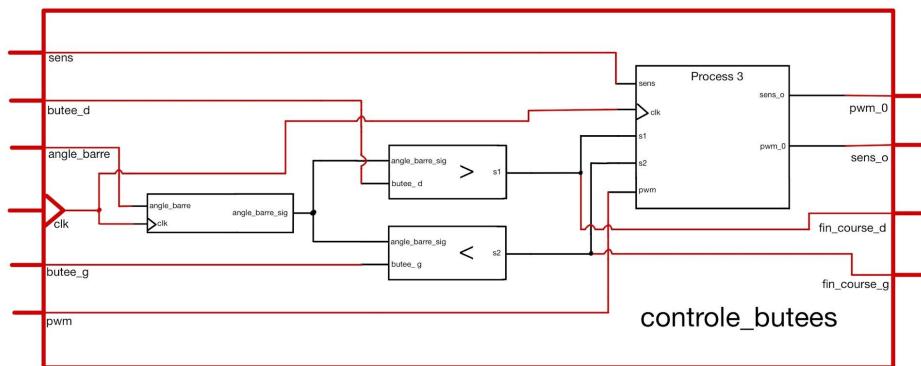
La valeur de Freq et Duty ne sont pas les valeurs réelles d'une fréquence et d'un rapport cyclique. Ce sont des valeurs obtenus via la formule "Freq = clk/Freq\_reel" , par exemple avec clk = 50 MHz, Freq\_reel = 25 kHz, on obtient Freq = 2000 et si on a un rapport cyclique de 0.5 , on obtient Duty = 1000. Ces valeurs seront calculées et fournies directement par le CPU.

Voici la simulation de ce bloc gestion PWM:



On retrouve des valeurs pour le rapport cyclique ainsi que la fréquence, on peut apercevoir que lorsque notre compteur sqn atteint la valeur 200, notre signal PWM passe au niveau logique '0'.

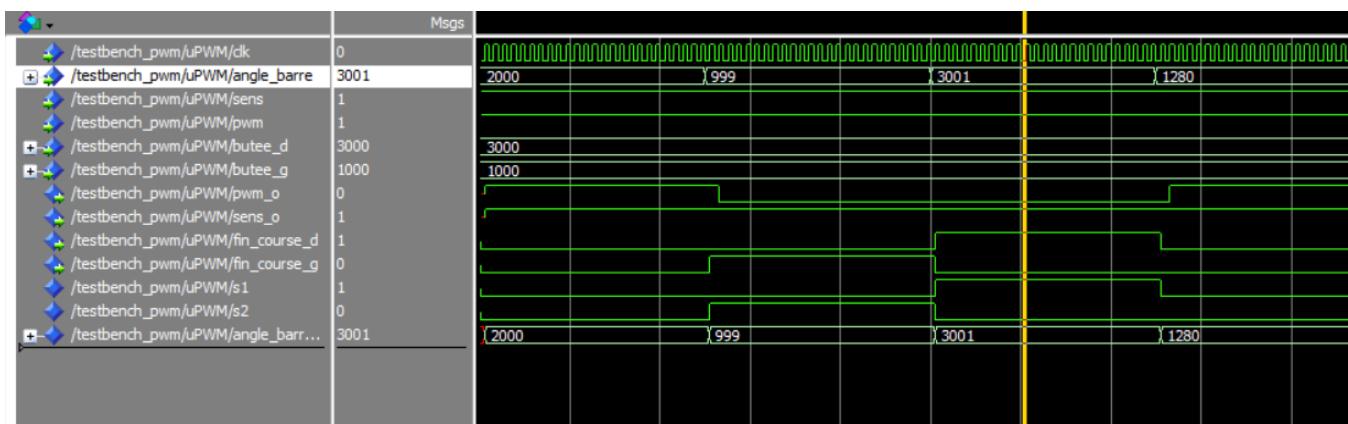
Ensuite pour le contrôle des butées nous avons le schéma fonctionnel ci dessous :



### Explication:

Avec butee\_g supérieur à butee\_d, on surveille en permanence que la valeur d'angle\_barre se trouve toujours bien entre butee\_g et butee\_d. Si ce n'est pas le cas, via le process 3, on va mettre la valeur de pwm\_o à 0 qui aura pour effet d'arrêter le moteur et donc arrêter la rotation du vérin. Si la valeur d'angle\_barre ne se trouve plus entre butee\_d et butee\_g, on mettra fin\_course\_d à 1 ou fin\_course\_g selon si c'est la valeur de la butee\_d ou butee\_g qui a été dépassé. Avec cette fonction on contrôlera aussi le sens de rotation du moteur, selon la valeur de sens\_o (0 ou 1), le moteur tournera vers la gauche ou vers la droite.

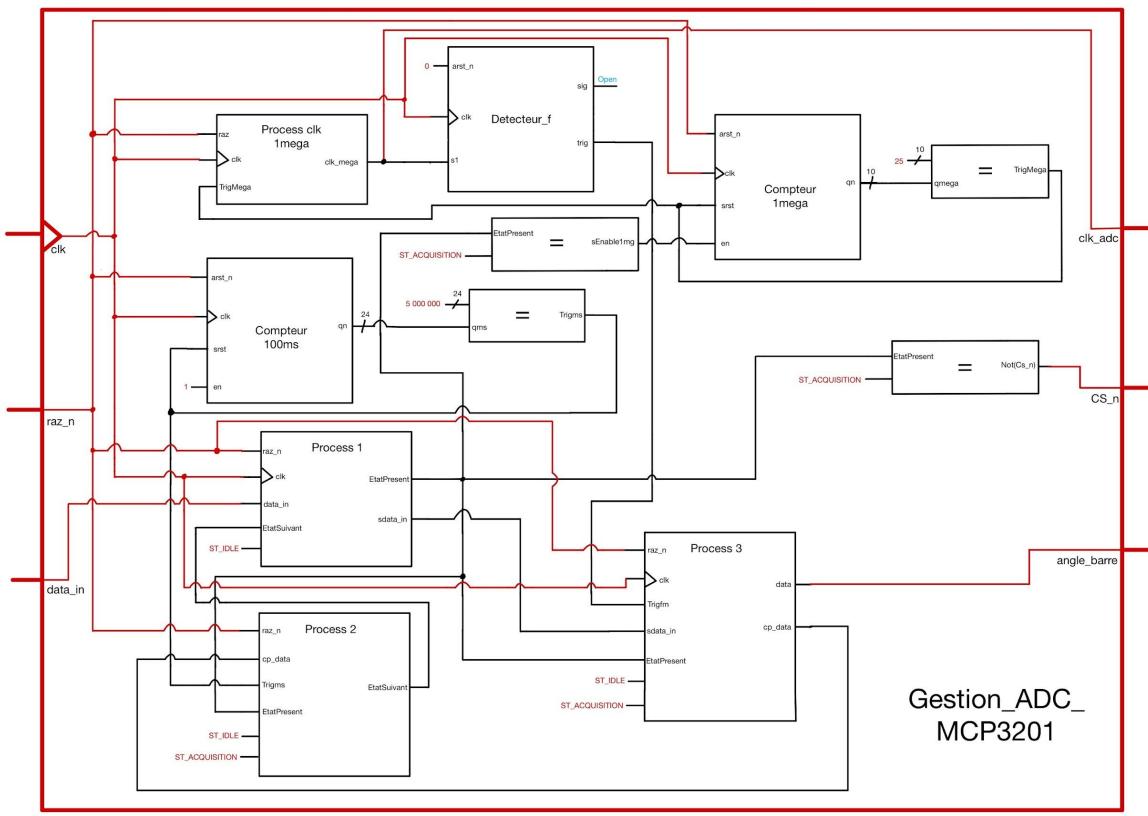
Voici la simulation associé à ce bloc de contrôles des butées:



En figeant la butée\_d et la butée\_g à 3000 et 1000, on peut voir qu'en faisant varier la valeur dans le vecteur angle\_barre, nous avons le PWM qui se met à zéro, ainsi que les fin de course droite et gauche qui se mettent respectivement au niveau logique '1'.

## Synthèse et mise en œuvre des systèmes

Enfin pour la gestion du convertisseur AN MCP 3101 voici son schéma fonctionnel :



### Explication:

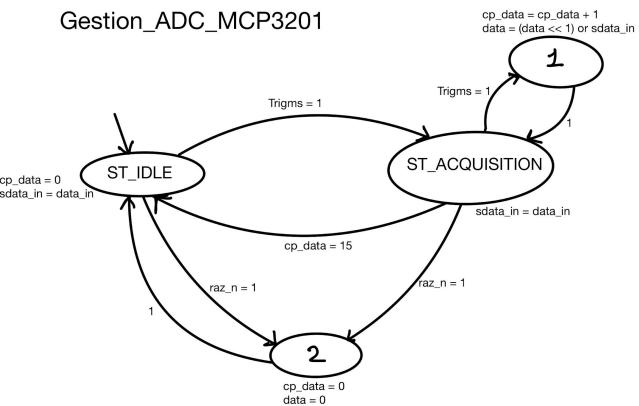
Le convertisseur ADC MCP3201 est un convertisseur qui fonctionne en SPI avec une horloge de 1MHz. Pour faire fonctionner ce convertisseur, on génère une horloge de 1 MHz avec le “Process clk 1mega”. Le “compteur 1mega” enverra un Trigger au “Process clk 1mega” toute les 0,5 us, qui servira à compléter le signal clk\_mega toute les 0,5 us, et on obtient ainsi une horloge avec une fréquence de 1 MHz. “compteur 1mega” se réinitialise toutes les 0,5 us.

Ensuite il y a un “Compteur 100ms” qui se réinitialise toutes les 100ms et qui va permettre de lancer une acquisition de l’angle de la barre toutes les 100 ms.

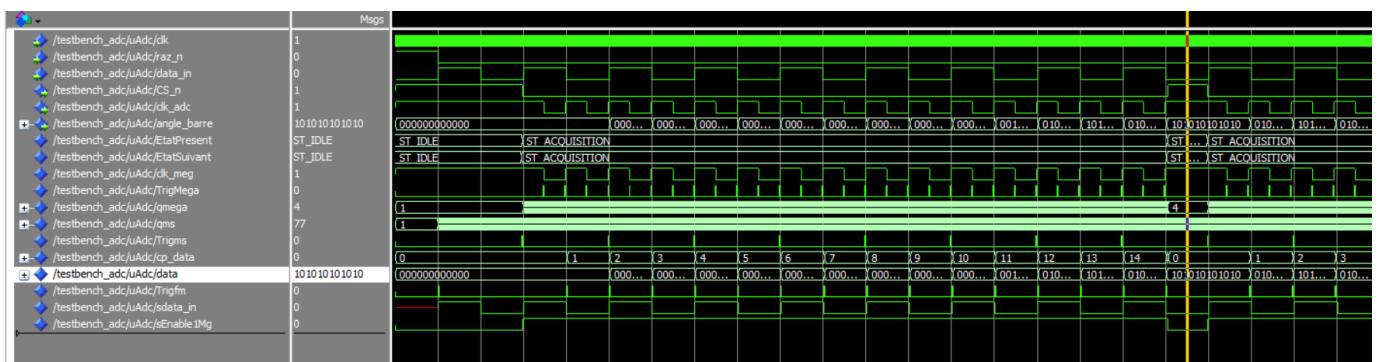
L’acquisition de la valeur d’angle barre se fait via la mise à 0 du Cs\_n et le process 3. La mise à 0 de Cs\_n sert à indiquer au convertisseur d’envoyer la donnée d’angle barre, et le process 3 sert lire la donnée d’angle barre bit par bit et d’enregistrer la valeur de l’angle dans un signal en prenant soin de faire un décalage bit à bit lors de l’enregistrement de la valeur d’angle barre lu.

Finalement les process 1 et 2 servent à gérer la machine à état de ce bloc, si on se trouve dans le mode Acquisition ou dans le mode repos. Il faut savoir que l’horloge 1 MHz se lance seulement quand on se trouve dans le mode Acquisition.

Voici la machine à état qui décrit la gestion de l'ADC MCP3201 :



Voici la simulation de ce composant qui va gérer les données envoyé par l'ADC MCP3201:



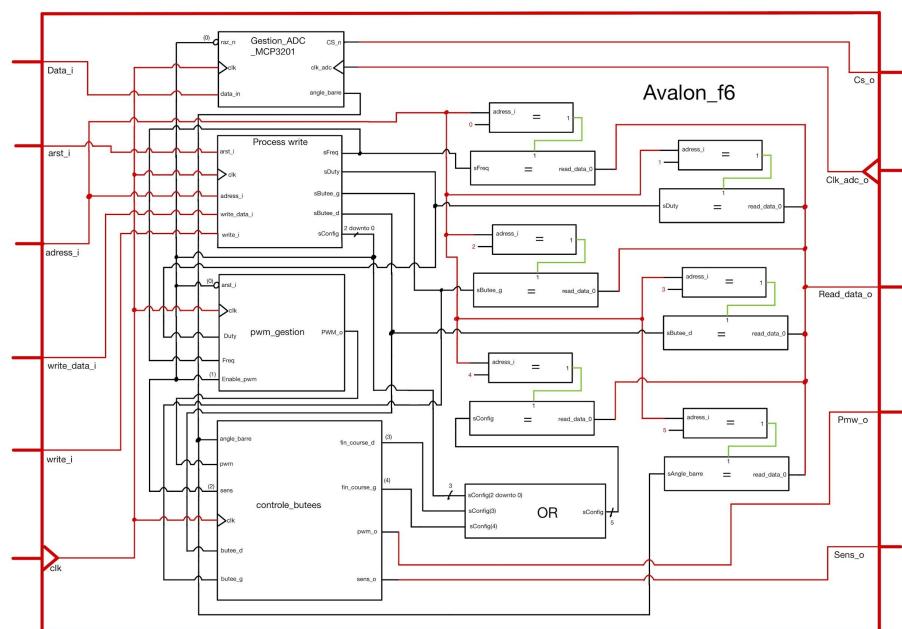
Comme l'indique notre machine à état, lorsque Trigms passe au niveau logique '1', nous entrons dans une phase d'acquisition des données. Cela induit notre Cs à 0, puis nous pouvons voir dans notre vecteur data, l'acquisition de toutes les données de l'ADC, qu'on a simuler avec sdata\_in.

Une fois l'acquisition faite, on retourne dans notre état d'attente jusqu'au prochain Trigger de Trigms.

### c) Bus Avalon Fonction 6

Finalement pour finaliser la gestion de l'ADC MCP3201, il faut comme avec l'anémomètre et la girouette permettre la communication avec le CPU, car le CPU aura besoin de lire la valeur de l'angle de la barre, ainsi que la valeur de fin\_course\_g et fin\_course\_d pour qu'il sache si une butée a été dépassée. Le CPU devra aussi imposer certaine configuration au bloc des gestions du convertisseur, comme par exemple imposer les valeurs de fréquences, rapport cyclique, butee\_g et butee\_d afin de permettre une bonne gestion de l'ADC MCP3201.

Comme expliqué précédemment avec l'Avalon de la fonction 1, pour créer ce bus de communication entre le CPU et le bloc de gestion du convertisseur, il faut écrire le code VHDL de la fonction 6, puis avec on pourra créer le composant Avalon\_f6 sur Platform Designer. Voici le schéma fonctionnel de l'Avalon de la fonction 6 :



#### Explication:

Les blocs Gestion\_ADC\_MCP3201, pwm\_gestion et controle\_butees que nous avons implémentés précédemment seront lié entre eux ici dans l'Avalon de la fonction 6.

Comme avec l'Avalon de la fonction 1, ici le bloc Process write est le process qui écrit la configuration défini par le CPU sur les 3 blocs Gestion\_ADC\_MCP3201, pwm\_gestion et controle\_butees.

Pour lire les données d'angle de la barre, fin\_course\_d et fin\_course\_g , fréquence, rapport cyclique, configuration (données de configuration du CPU + fin\_course\_d + fin\_course\_g), ça se fera directement en combinatoire, on attendra juste que l'address\_i vaut 0 et 1 pour envoyer les données du la fréquence et rapport cyclique vers le CPU, 2 et 3 pour envoyer les données de butee\_g et butee\_d vers le CPU, et puis 4 et 5 pour envoyer les données de configuration et angle\_barre vers le CPU.

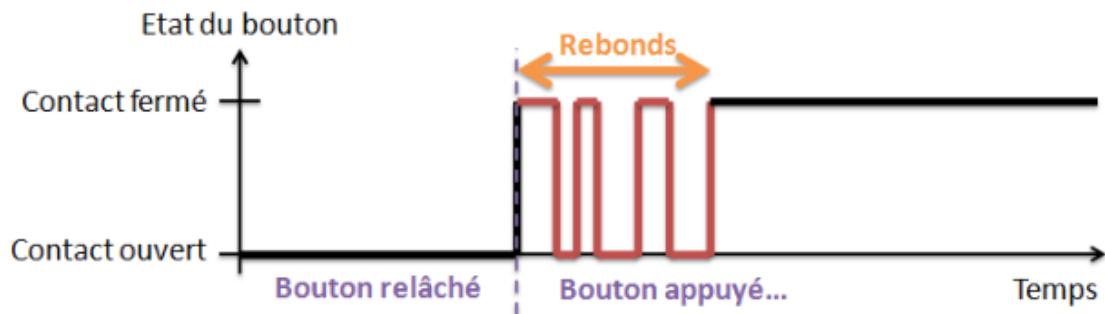
### III/Fonction 7 : IHM

La Fonction 7 met en œuvre plusieurs boutons qui vont permettre de piloter notre système de navigation, un mode veille et automatique sera mis en place, avec une possibilité de régler le cap manuellement à partir de ces boutons.

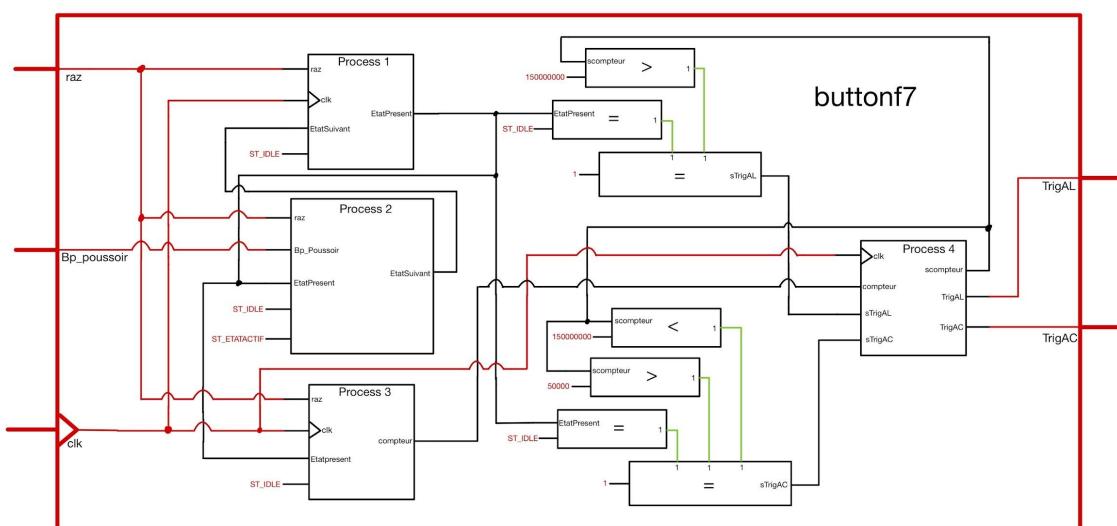
Une IHM constituée principalement de led sera aussi mise en place dans cette fonction.

#### a) Pilotage des boutons

Lors de l'appui d'un bouton physique des rebonds sont induits, le changement d'état logique ne se fait pas parfaitement, des rebonds sont induits:



Notre système prend aussi en considération un appui long ou court du bouton, pour le réglage du cap par exemple, voici le schéma fonctionnel de la gestion des rebonds lors d'un appui sur un bouton poussoir :



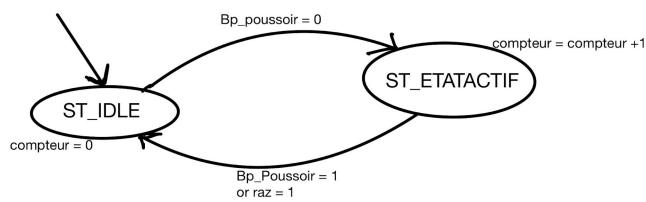
## Explication:

Dans le process 3, nous avons un compteur qui va s'incrémenter lorsque la machine à état est dans l'état ST\_ETACTIF, sinon le compteur s'initialise à zéro.

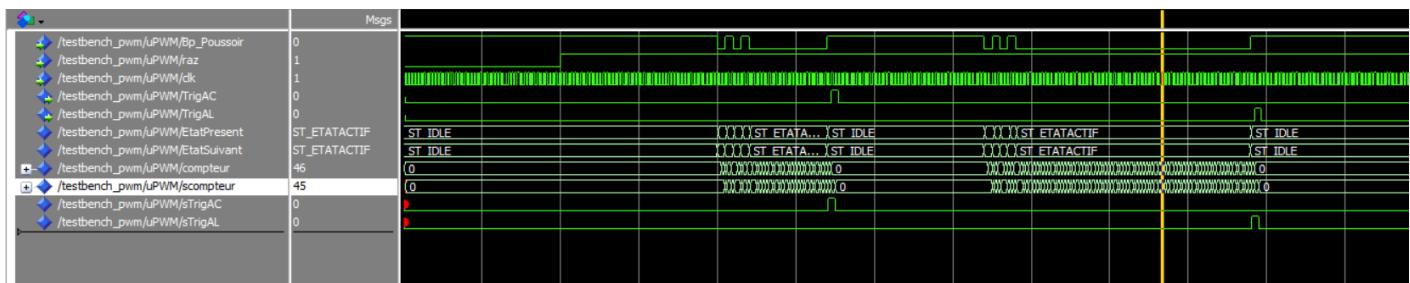
Lorsque ce compteur dépasse 50000 (soit 1ms) on considère que nous ne sommes pas dans un rebond du bouton. L'appuie du bouton sera donc pris en compte.

Afin de pouvoir déterminer si l'appuie est long ou court, et pouvoir Trigger le bon signal, nous allons une deuxième fois comparer ce compteur. Lorsque l'appuie est situé entre 1ms et 3 secondes, on considère que l'appuie est court, au delà de 3 secondes l'appuie est long.

Voici la machine à état qui décrit gestion des rebonds lors d'un appuie sur un bouton poussoir :



Voici en simulation le composant bouton:



On peut voir en entrée on a simulé un bouton poussoir avec des rebonds, en appuie court ou appuies long.

Le premier appuie est un appuie court, TrigAC passe donc au niveau logique '1', tandis que le second appuie est un appuie long, TrigAL passe donc au niveau logique '1'.

### III/Conclusion

En conclusion, dans ce BE, nous avons fait le choix dans un premier temps de partir sur la fonction 1 afin de prendre en main le VHDL ainsi que le NIOS et le bus Avalon. Nous avons au préalable, avant de partir en test, simuler tous nos bloc VHDL en essayant d'avoir un testbench qui se rapproche au mieux de tous les cas qu'on peut rencontrer en pratique.

Ensuite nous sommes parties vers la fonction 6, afin de pouvoir récupérer la direction du cap, plusieurs blocs indépendants étaient créés et simulés de manière indépendante, pour enfin rassembler et intégrer chaque blocs dans un seul projet.

Enfin, nous avons commencé par faire la fonction 7, la gestion des boutons et des leds. Nous avons validé en simulation et en test le bus avalon de cette fonction (qui ressemble beaucoup au autre) ainsi que notre composant qui sert à gérer les rebonds et la temporisation des boutons.

Nous avons ensuite continué sur l'intégration de ce composant afin de finaliser la fonction 7 en vain.

Globalement, nous trouvons qu'on a eu une évolution constante dans le projet, et cela nous a permis de nous familiariser avec le VHDL ainsi que le NIOS, et l'intégration d'un CPU en général sur une puce FPGA.