Rapport
BE voilier

2022-2023

JEANNIN Lucie
MAZARGUIL Marlon

M2 SME
Université Paul Sabatier
Synthèse et mise à en oeuvre des systèmes

Introduction

Dans le cadre de notre formation nous avons fait un projet, pour nous initier et mettre en pratique nos connaissances en VHDL. Pour ce faire nous avons utilisé plusieurs logiciels, comme Quartus pour compiler le code et ModelSim pour le tester et vérifier son fonctionnement.

Pour ce projet, nous travaillons sur les capteurs d'un voilier, comme le capteur de position par rapport au Nord magnétique, la vitesse et la direction de vent.

Nous travaillerons aussi sur l'interface Homme-machine. L'utilisateur aura plusieurs modes pour modifier le cap via des boutons.

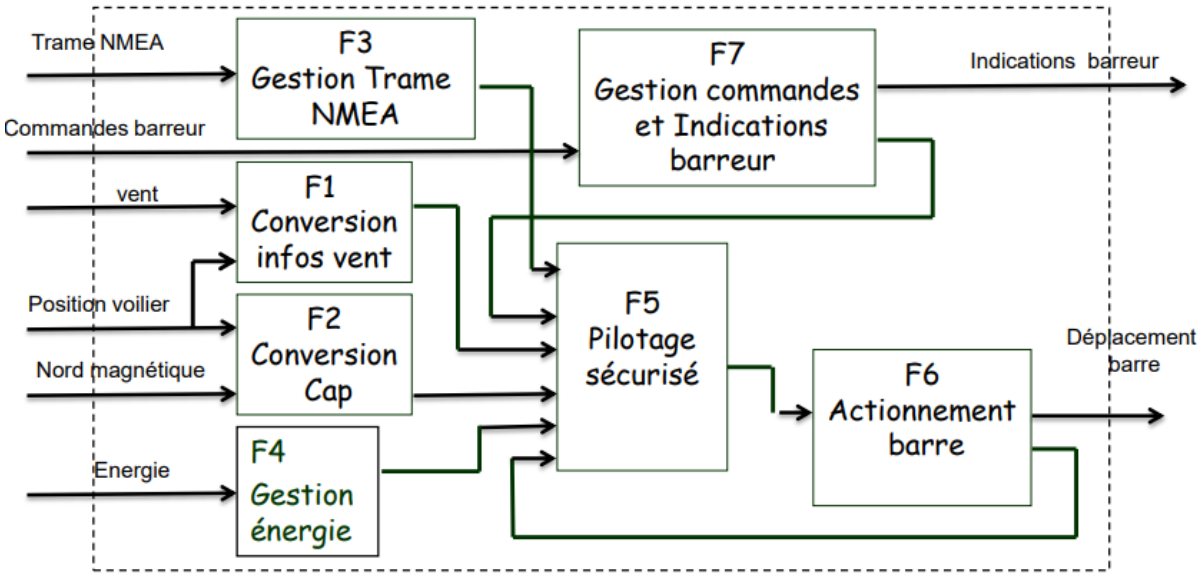
Le programme sera expliqué avec des schémas fonctionnels/blocs

Projet

Le projet “voilier” est découpé en 7 fonctions différentes. Nous nous intéresserons aux étapes :

- Conversion infos vent (F1),
- Conversion Cap (F2),
- Gestion Commandes et indications barreur (F7)

On peut voir toutes les étapes du projet sur l’image ci-dessous



Nous traiterons d’abord la fonction F2 car elle contient moins de “bloc”. De plus, nous utiliserons F2 pour faire la fonction F1. On finira par F7 qui est la fonction compliquée du projet.

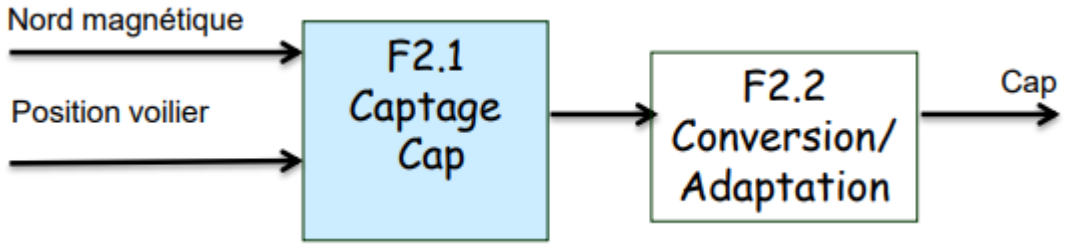
F2 Conversion Cap

La fonction Conversion Cap a pour but de convertir les données analogiques (Nord magnétique et position voilier) qu'on recevra sur une boussole en donnée numérique (le cap).

Sur le schéma ci-dessous nous pouvons voir les deux blocs principaux de la fonction F2.

Un bloc captage Cap : qui représente la boussole : Il reçoit les données analogiques (Nord magnétique et position voilier) et les convertit en numérique (PWM)

Et un bloc Conversion/Adaptation : qui recevra la PWM et devra la convertir en degrés pour indiquer l'orientation du voilier



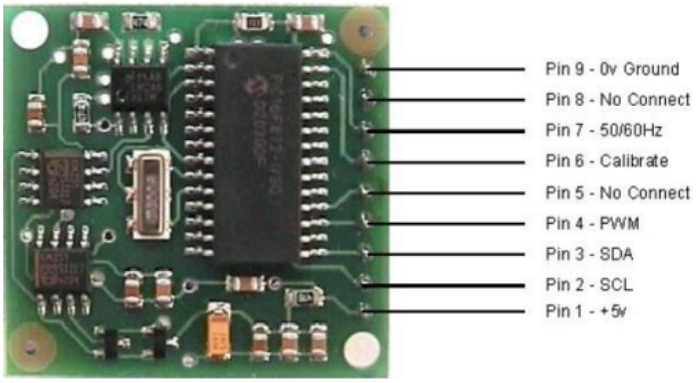
F2.1 Captage Cap

Comme on peut le voir sur le tableau ci-dessous nous avons choisi une boussole magnéto-résistive car même si on a une précision moins importante que les autres, le type de données en sortie nous convient mieux.

Type	précision	Sortie	Choix
Compas Fluxgate	<1 degré	analogique	
Boussole magnéto-résistive	3 degrés	numérique (PWM) ou I2C	X
Module Compas	<1 degré	Trame NMEA	

Cette boussole fera la conversion analogique numérique de la position du voilier et du Nord magnétique en une PWM que nous pourrons traiter. Nous utiliserons la boussole CMPS03 (lien de la datasheet dans l'annal).

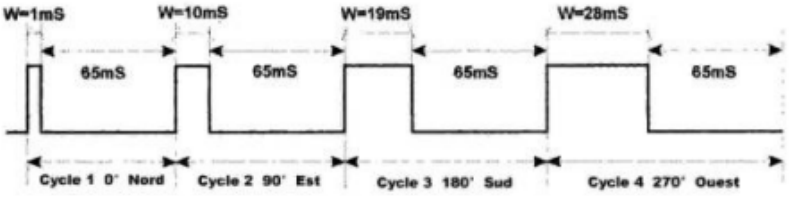
Sur les images ci-dessous (extrait de la datasheet) nous pouvons voir les branchements de la boussole.



+ 5 Volts	Alimentation
SCL	Ligne d'horloge du bus **I2C** (100 KHz standard)
SDA	Ligne de données du bus **I2C**
PWM	Sortie modulée en largeur d'impulsion
N.C	Non Connectée
CALIBRATE	Entrée pour le calibrage matériel du module
50/60 Hz	Entrée de sélection du filtre interne
N.C	Non Connectée : en fait, ligne de reset du microcontrôleur avec une résistance pull-up. Utilisée lors de la programmation du module. Peut servir pour faire un reset du module boussole
0 V	Masse de l'alimentation

Pour le traitement des données de la boussole nous nous référerons à l'image ci-dessous (extrait de la datasheet)

L'impulsion est générée sur 16 bits par le microcontrôleur donnant 1 µs de résolution soit un angle de 0,1°, mais il n'est pas recommandé en pratique de mesurer des directions avec des angles inférieurs à 3 ou 4°.



Il est important de noter que la PWM en sortie de la boussole n'est pas "classique", car la période n'est pas stable, on reste toujours le même temps à l'état bas (65 ms).

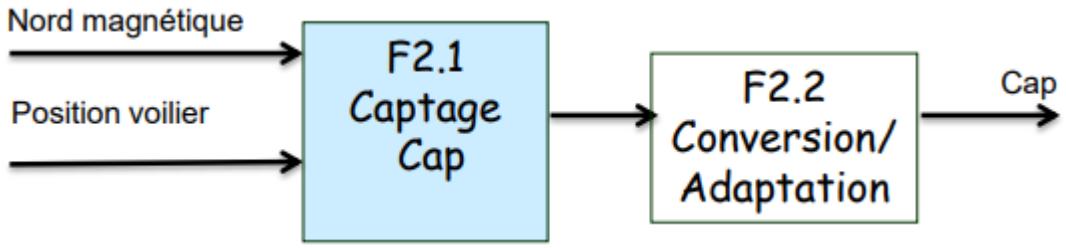
Donc pour connaître la position on doit se référer au temps à l'état haut :

1 ms → 0°
10 ms → 90°
19 ms → 180°
28 ms → 270°

On a donc 0,1 ms = 1 degrés
On peut en déduire l'équation pour connaître les degrés en fonction du temps :

$$\theta(t) = (t - 1) * 10 \quad , avec\ t\ en\ ms\ et\ \theta\ en\ degrés$$

Maintenant que nous savons quel va être le signal d'entrée nous allons voir comment le traiter.



F2.2 Conversion/Adaptation

Le but de cette partie est de faire convertir un signal PWM en degré.

Rappel information PWM : 0,1 microseconde à l'état haut de la PWM correspond à 1 degré (le zéro degré correspond à un état haut de 1 ms).
On utilisera la formule suivant pour faire le lien entre le temps à l'état haut et les degrés :

$$\theta(t) = (t - 1) * 10 \quad , avec\ t\ en\ ms\ et\ \theta\ en\ degrés$$

Nous devons répondre à plusieurs besoins pour satisfaire cette partie :

- Faire la conversion PWM → degrés
- Faire deux modes :
    - Un mode monocoup : il permet d'actualiser le cap uniquement à l'appui d'un bouton
    - Un mode continu : il permet d'actualiser le cap toutes les secondes

Pour ce faire nous avons 5 entrées :

- clk_50M : l'horloge,
- raz_n : le reset asynchrone (actif à l'état bas),
- in_pwm_compas : la PWM,
- continu : un bouton pour gérer le mode continu ou monocoup,
- start_stop : un bouton pour actualiser en mode monocoup

Et nous devons avoir 3 sorties :

- data_compas : valeur du cap en degrés
    - si mode continu (continu à 1) : actualisation toutes les secondes
    - sinon mode monocoup (continu à 0) :
        - si appui sur le bouton sur le bouton d'actualisation (start_stop = 1) : actualiser le cap
        - sinon pas d'appui (start_stop = 0) : pas d'actualisation (on reste sur le dernier cap)
- out_1s :
    - il est à 1 toutes les secondes pendant un coup d'horloge
    - sinon 0
- data_valid :
    - si mode continu (continu à 1) : data_valid à 1
    - sinon si mode monocoup (continu à 0) :
        - si appui sur le bouton sur le bouton d'actualisation (start_stop = 1) : data_valid à 1
        - sinon pas d'appui (start_stop = 0) : data_valid à 0

Nous pouvons donc distinguer plusieurs parties dans ce bloc :

- Compter toutes les microsecondes : pour conversion des données de la PWM en degrés
- Mettre à jour le cap toutes les secondes
- gestion des modes

- la validation du cap

Compter toutes les microsecondes

Pour obtenir la sortie en degrés :

On doit dans un premier temps savoir combien de microsecondes la PWM reste à l'état haut. On doit être précis au micron près car toutes les 100 microsecondes on doit incrémenter de 1 degré.

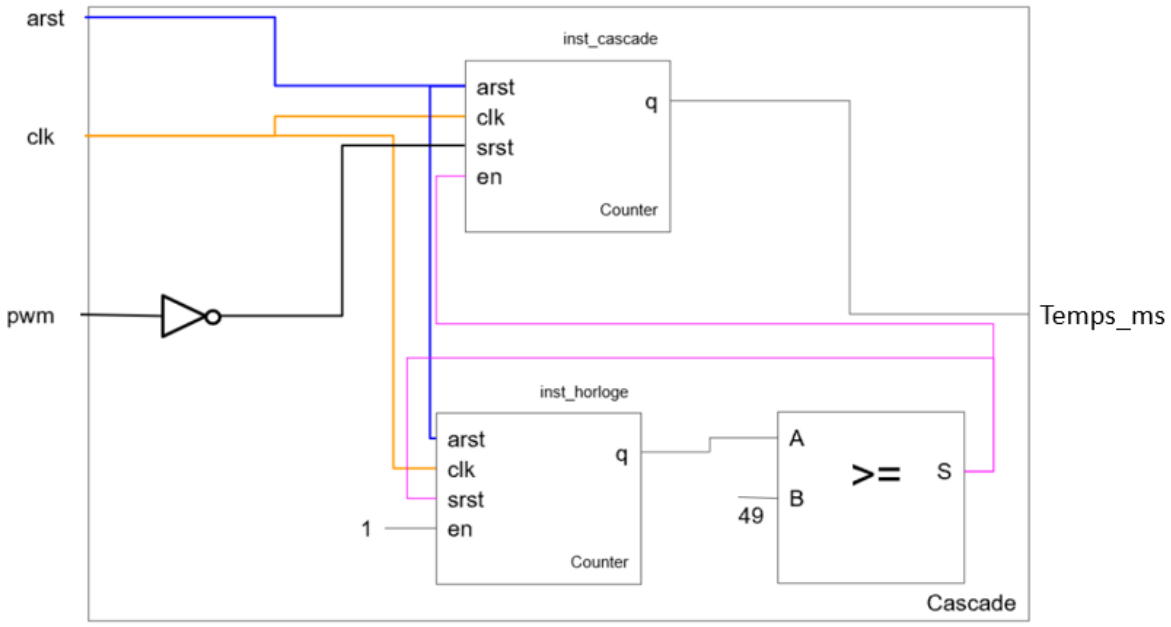
Pour ce faire, on a fait un compteur et un comparateur qui permettent d'avoir un bit de sortie à 1 (pendant un coup d'horloge) toutes les microsecondes. Ce qui va permettre à un autre compteur de compter toutes les microsecondes.

Explication : Si on a l'entrée enable à 1 on compte, et sinon s'il est à 0 on arrête de compter (attention, ça ne remet pas à zéro le compteur). Donc si on a l'enable à 1 pendant un coup d'horloge toutes les microsecondes, on aura un compteur qui compte toutes les microsecondes.

On sait que l'horloge est à 50 MHz donc on a une période de 0,02 microseconde, donc il faut 50 coups d'horloge pour avoir 1 microseconde. On devra donc compter jusqu'à 49 car on commence à compter à 0.

Le compteur se réinitialise si on a la PWM à 0 donc on peux le connecter directement au reset. Comme le reset est actif à l'état haut, on inverse le signal de la PWM pour remettre le compteur à 0 à l'état bas de celle-ci.

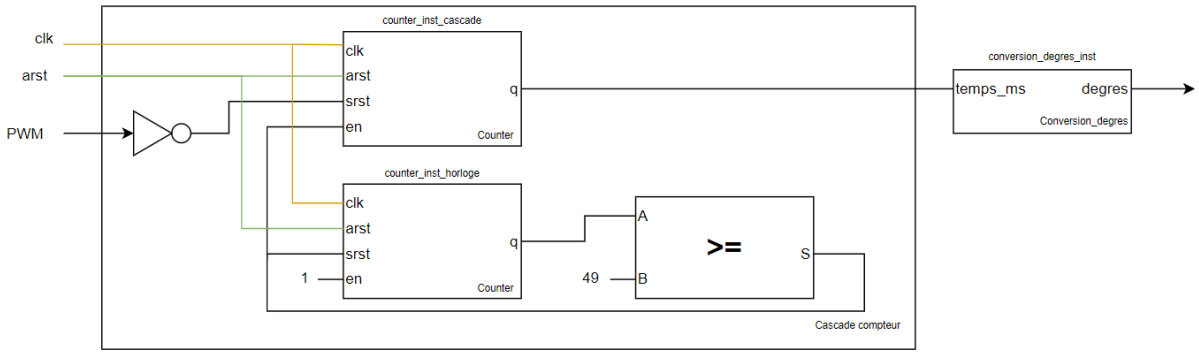
On peut voir le schéma fonctionnel de cette partie ci-dessous :



Une fois qu'on a pu vérifier que le temps était correct nous avons rajouté un convertisseur qui permet de connaître les degrés en fonction du temps grâce à la formule suivant :

$$\theta(t) = (t - 1) * 10 \quad , avec\ t\ en\ ms\ et\ \theta\ en\ degrés$$

On peut voir le schéma fonctionnel final de cette partie ci-dessous :



Mise a jours du cap toutes les secondes

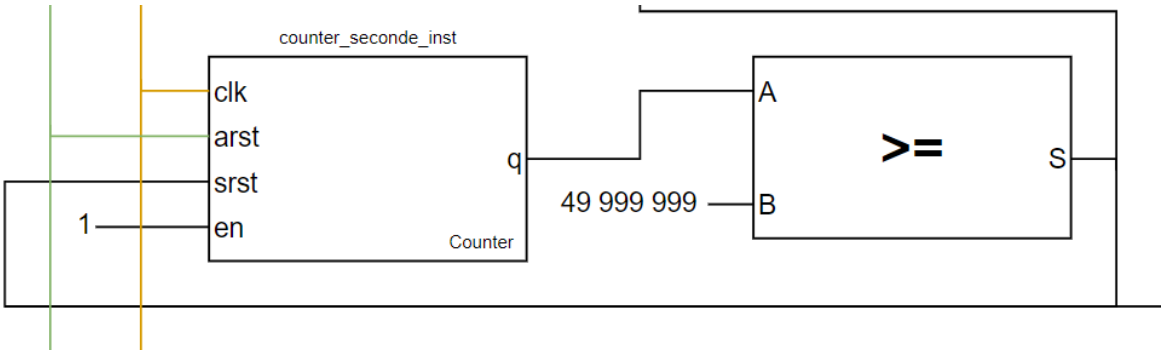
Pour faire la mise à jour toutes les secondes, nous avons utilisé le même principe que dans la partie précédente avec l'horloge et le comparateur sauf qu'on doit l'adapter pour 1 seconde.

$$\text{horloge à } 50Mz = 20\,ns = 2 * 10^{-9}s$$

$$1s = 2 * 10^{-9} * (50 * 10^{6})$$

donc il faut $50 * 10^{6}$ coups d'horloge, on commence à compter à 0 donc on compte jusqu'à 49 999 999.

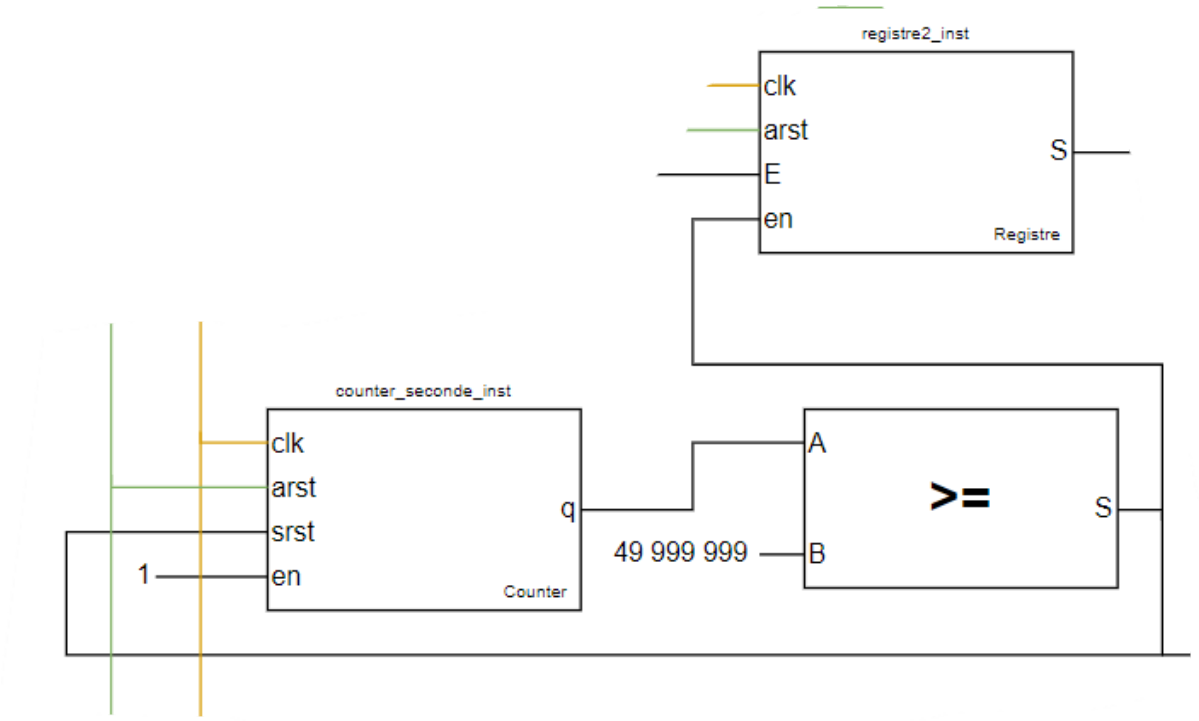
On peut voir le schéma bloc de la mise à 1 toutes les secondes ci-dessous



Maintenant il faut enregistrer la valeur, pour cela nous allons faire un registre, le principe d'un registre est simple :

- si enable à 1 : l'entrée égale à la sortie si enable est à 1,
- sinon enable à 0 : on garde la dernière valeur de l'entrée

Donc on doit mettre la sortie S du bloc précédent à l'entrée du registre, ce qui donne le schéma ci-dessous :

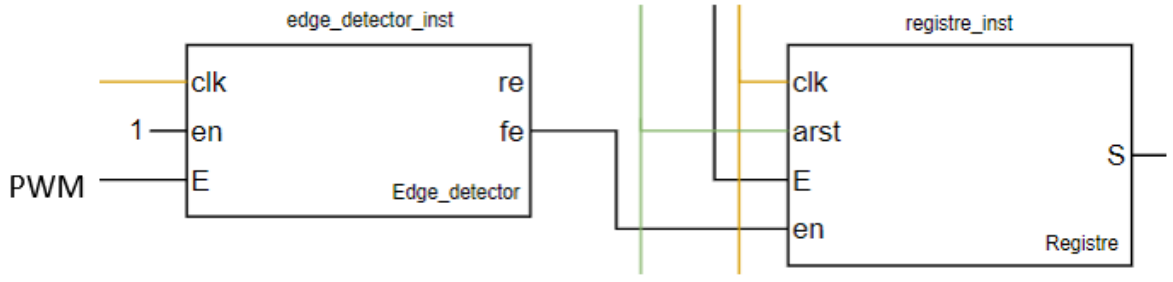


Dans la partie précédente on a fait un compteur qui compte le temps à l'état haut de la PWM et qui la convertit en degrés, donc quand on est à l'état bas il renvoie 0.

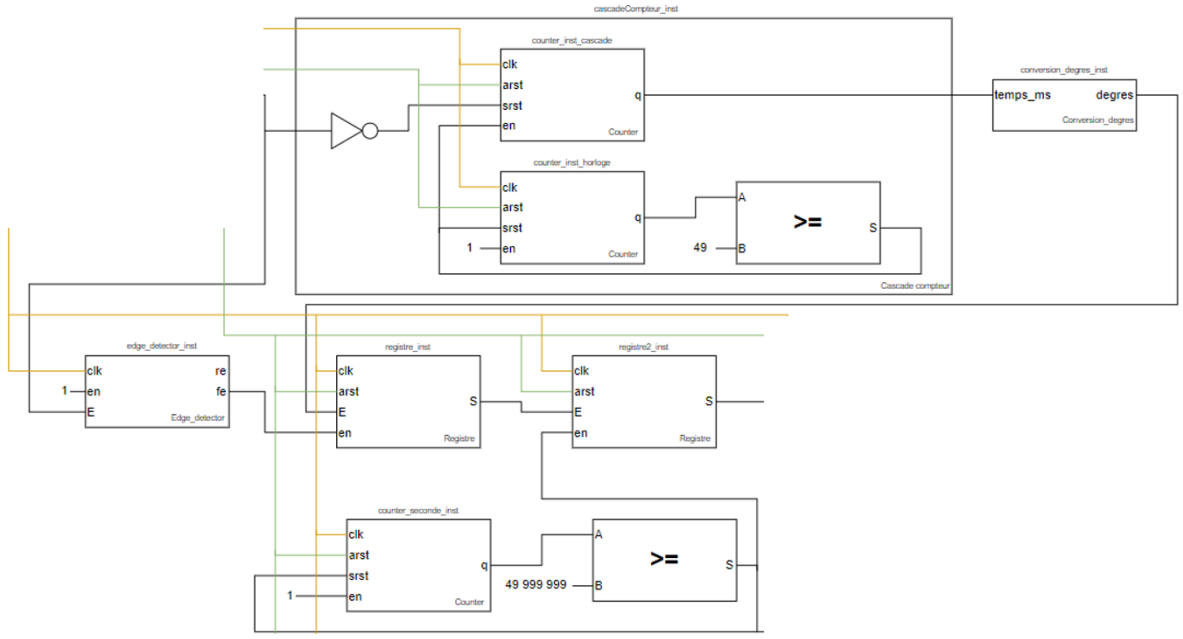
Donc nous devons enregistrer la dernière valeur avant que la PWM soit à 0, (sinon il y a une forte probabilité pour qu'on obtient une valeur fausse quand on fait la mise à jour)

Pour ce faire nous devons détecter le front descendant de la PWM. Lorsque nous détectons le front descendant, on enregistre la valeur des degrés.

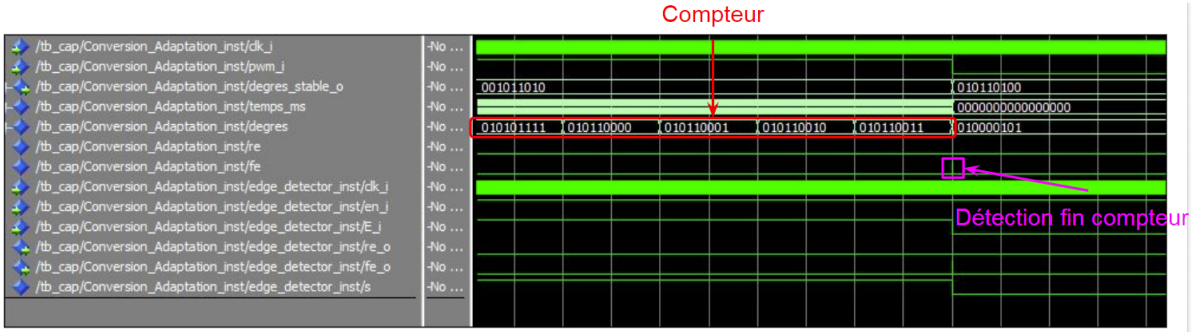
Ce qui nous donne le schéma bloc ci-dessous :



Si on met les schémas ensemble, on obtient :



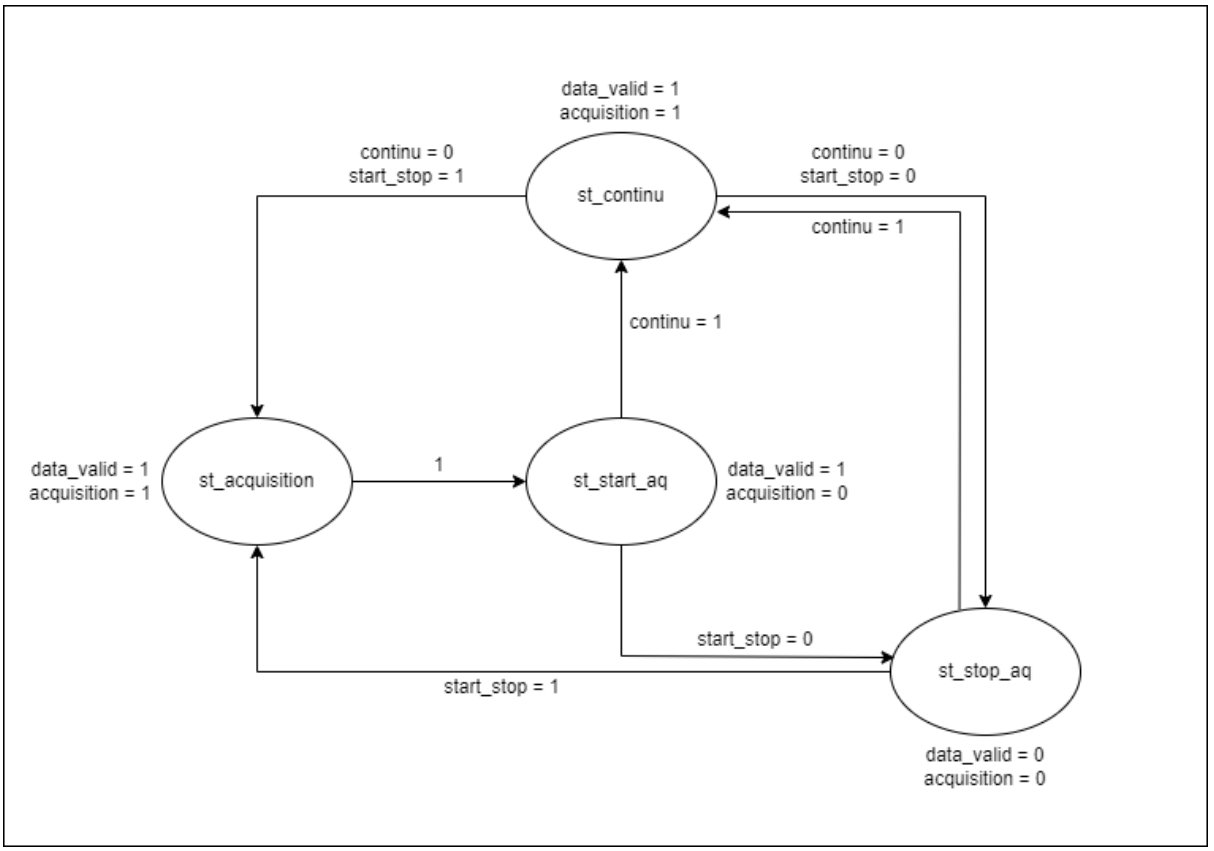
Avec ModelSim on a pu tester cette partie, on peut voir sur les deux chronogrammes ci-dessous (le deuxième est un zoom du premier), qu'on compte jusqu'à ce qu'on observe un front descendant de la PWM et qu'on enregistre la valeur jusqu'au prochain front montant.



Gestion du mode

Rappel des modes :
SI continu = 1 on fait une acquisition toute les secondes
SINON continu = 0 alors on regarde la valeur de start_stop
SI start_stop = 1 seul on fait une seul acquisition (monocoup)
SI start_stop = 0 on ne fait pas d'acquisition

On peut en déduire la machine à état suivante :



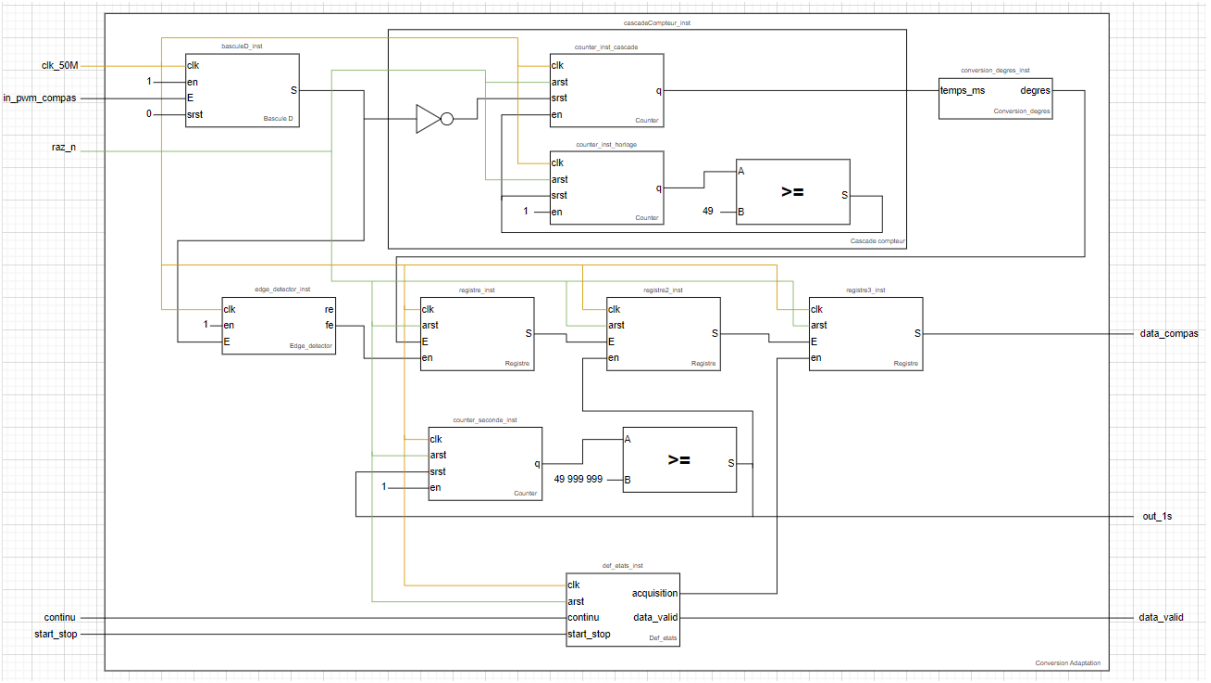
On a donc rajouté deux blocs : un bloc qui sera la machine à état et un autre bloc registre qui enregistrera ou non les degrés selon l'état dans lequel on est.

Finalisation de la fonction F2

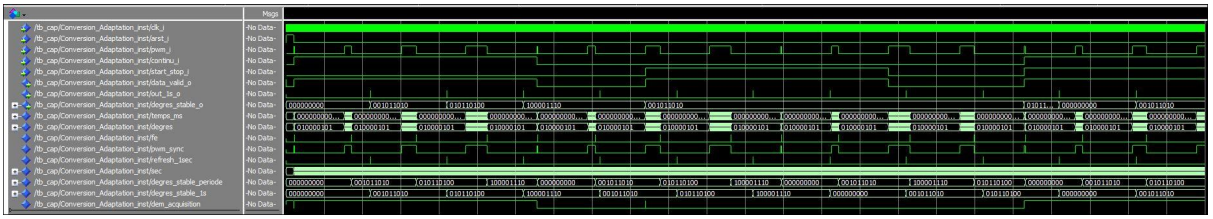
On a rajouté une bascule D en entrée, elle permet de synchroniser la PWM avec l'horloge.

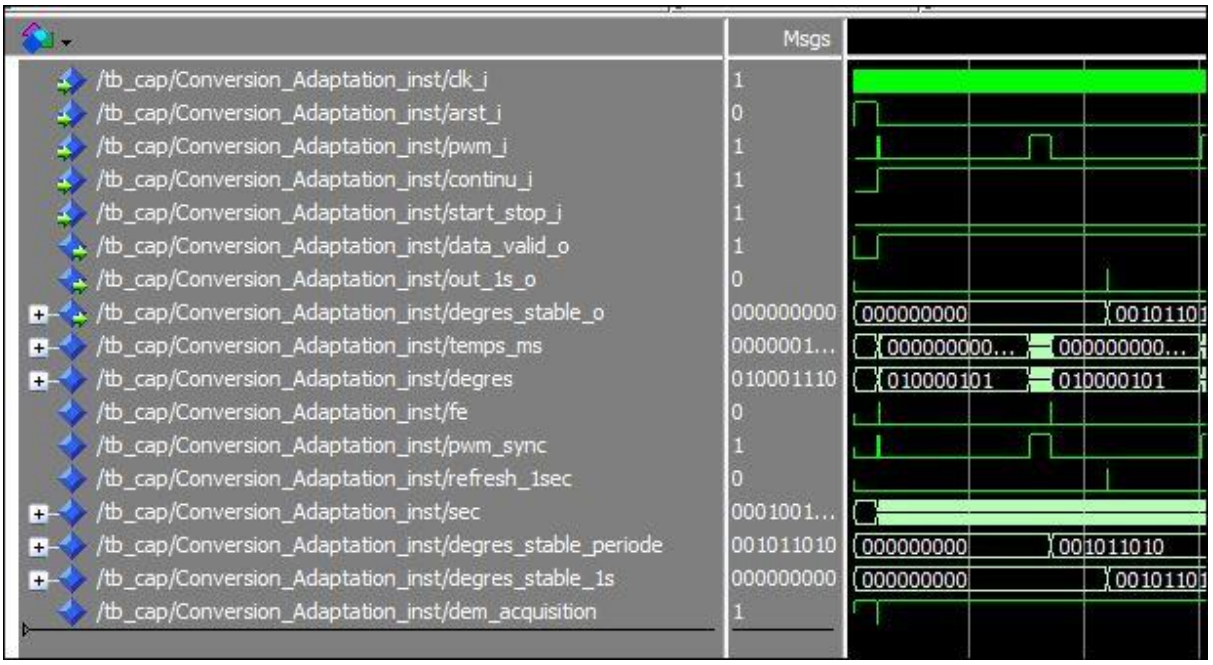
De plus on a connecté des blocs à des sorties comme out_1s qui doit être à 1 toutes les secondes et la machine à état qui nous donne la valeur de data_valid.

En associant toutes les parties on obtient finalement le schéma fonctionnel suivant :



On peut visualiser le chronogramme final qui nous permet de vérifier que tout fonctionne correctement (le deuxième est un zoom du premier)





Récapitulatif de tous les signaux

ENTRÉES :

clk : horloge, 50 MHz, utile pour la base de temps elle va permettre de mettre du temps dans les fonctions.

arst : restart asynchrone, initialiser le circuit : si il est à 1 tous les signaux de sorties sont à 0, sinon on passe à la suite.

pwm : orientation/cap du capteur.

continu : Si 1 → mode pour actualiser la valeur toutes les secondes, sinon 0 on regarde le start_stop.

start_stop : si il est à 1 on fait que une acquisition (mode monocoup) sinon à 0 on ne fait pas d'acquisition (si continu à 0).

SORTIES :

data_valid : 1 si la mesure est valide (c'est à dire si continu = 1 ou start_stop = 1).

out_1s : il est à 1 toutes les secondes.

data_compas : valeur du cap en degrées.

SIGNAUX INTERNES :

temps_ms : temps en microseconde à l'état haut de la PWM.

degres : degrés à chaque période de la PWM.

fe : (fall edge) front descendant de la PWM.

pwm_sync : pwm synchronisé avec l'horloge.

refresh_1sec : signal out_1sec.

sec : pour compter jusqu'à 1 sec.

degres_stable_periode : pour garder la dernière valeur de degrés au moment de la remise à zéro de temps_ms.

degres_stable_1sec : actualise la valeur de degrés toutes les secondes.

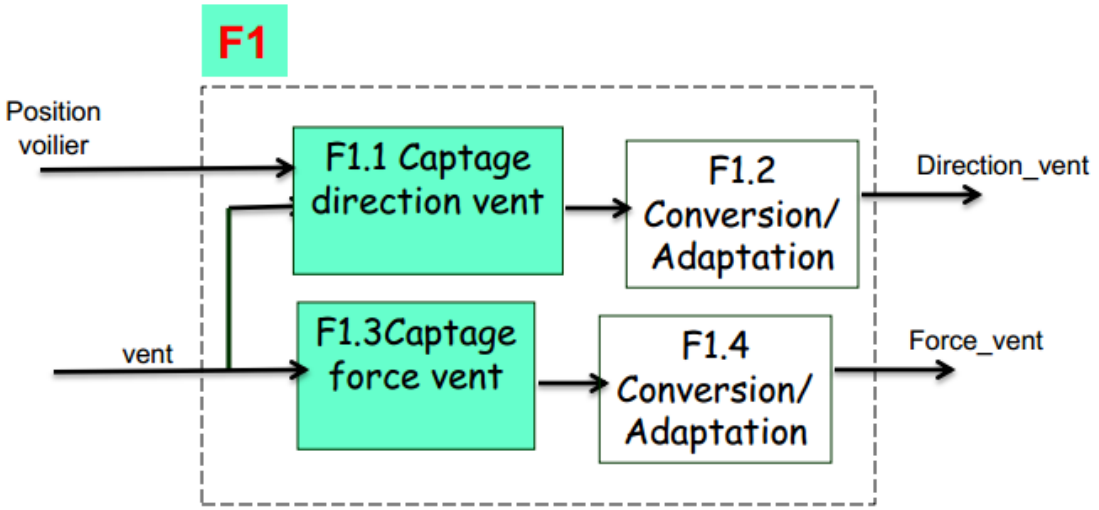
dem_acquisition : (demande d'acquisition) pour valider l'acquisition selon le mode, si continu = 1 → dem_acquisition reste à 1, si start_stop = 1 → dem_acquisition est à 1 pendant un coup d'horloge.

F1 Conversion info vent

Pour la fonction F1 Conversion info vent, elle se divise en 4 parties :

- F1.1 Captage direction vent : capteur qui nous donne la direction du vent
- F1.3 Captage force vent : capteur qui nous donne la vitesse du vent
- F1.2 et F1.4 Conversion/Adaptation : sont des fonctions qui permettent de convertir les données reçues depuis les capteurs en données traitables par le système

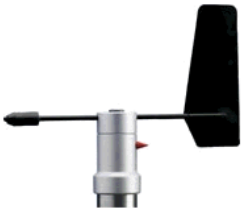
Dans cette partie nous réutilisons les mêmes blocs ou la même logique que ceux de la partie précédente (F2), nous ne les re-développerons pas ici.



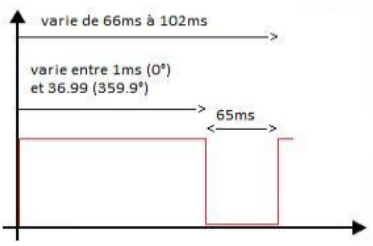
F1.1 Captage direction vent

Le capteur qui nous permettra de connaître la direction du vent est une “Girouette”. On peut voir les informations sur le capteur sur l’image ci-dessous :

=> 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
Boitier	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

L'information principale qui nous intéresse est le signal de sortie qui est une PWM, le capteur convertit la direction du vent en PWM.

Cette PWM se comporte de la même manière que celle de la partie précédente, c'est à dire que c'est l'état haut qui nous donne l'information sur la direction du vent et l'état bas est constant (65 ms).

- 1 ms → 0°
- 36,99 ms → 359,9°

On peut en déduire la même équation que la partie précédente :

$$\theta(t) = (t - 1) * 10 \quad , avec\ t\ en\ ms\ et\ \theta\ en\ degrés$$

F1.3 Captage force vent

Le capteur qui nous permettra de connaître la vitesse du vent est un "Anémomètre". On peut voir les informations sur le capteur sur l'image ci-dessous :

=> 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
Boitier	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	0-250Km/h

L'information principale qui nous intéresse est le signal de sortie de fréquence variable entre 0 à 250 Hz, le capteur convertit la vitesse du vent en une fréquence.

La plage de vitesse qu'il peut capter est de 0 à 250 km/h.

On peut donc déduire :

$$fréquence(Hz) \equiv vitesse(km/h)$$

- 0 Hz → 0 km/h
- 250 Hz → 250 km/h

F1.2 Conversion/Adaptation

L'objectif de cette fonction est de reconvertir le signal PWM valeur en degrés dont l'équation est la suivant :

$$\theta(t) = (t - 1) * 10 \quad , avec\ t\ en\ ms\ et\ \theta\ en\ degrés$$

La fonction est strictement la même que pour F2 donc nous ne l'avons pas refaite.

F1.4 Conversion/Adaptation

L'objectif de cette fonction est de convertir une fréquence en vitesse.
L'équation est la suivante :

$$fréquence(Hz) \equiv vitesse(km/h)$$

- 0 Hz → 0 km/h
- 250 Hz → 250 km/h

On a une égalité directe entre la fréquence et la vitesse.
Donc on va s'intéresser à trouver une méthode pour connaître la fréquence.

On peut utiliser la définition de la fréquence : le nombre de périodes sur une seconde. Pour connaître le nombre de périodes, il suffit de connaître le nombre de fronts montants :

$$nombre\ de\ période \equiv nombre\ de\ front\ montant$$

Pour ce faire, on a donc besoin d'un compteur qui compte tous les fronts montants et qui se remet à zéro toutes les secondes.

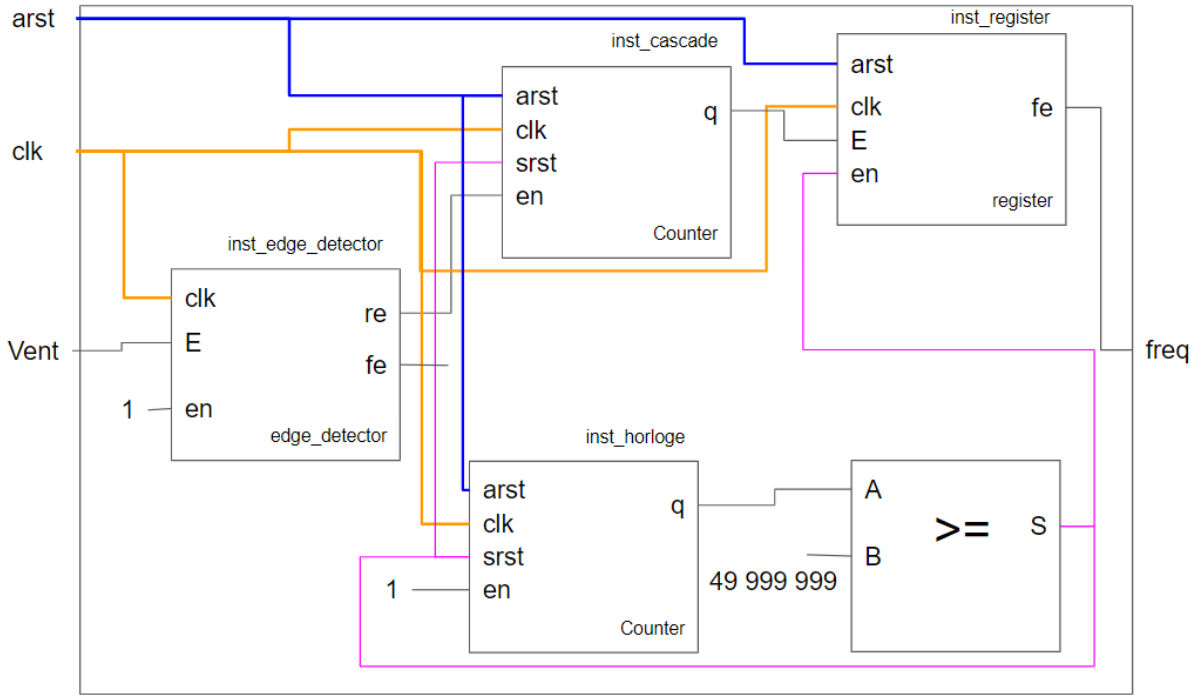
Pour ce faire nous utilisons le détecteur de front montant qui ce met à 1 pendant une période d'horloge et qui est relié à l'enable du compteur.

Cela permet d'incrémenter de 1 le compteur tous les fronts montants de l'horloge.

Ensuite on utilisera un registre qui enregistre la valeur du compteur toutes les secondes.

Pour avoir une action toutes les secondes, nous avons créé un bloc (compteur + comparateur) qui envoie 1 pendant un coup d'horloge toutes les secondes (nous avons expliqué son principe dans la partie F2). Ce bloc sera relié au reset du compteur pour la fréquence et à l'enable du registre.

On peut donc en déduire le schéma fonctionnel ci-dessous :


arst
clk
Vent
1
freq
inst_cascade
arst
clk
srst
en
q
Counter
inst_register
arst
clk
E
en
fe
register
inst_edge_detector
clk
E
en
re
fe
edge_detector
inst_horloge
arst
clk
srst
en
q
Counter
1
49 999 999
A
B
>=
S

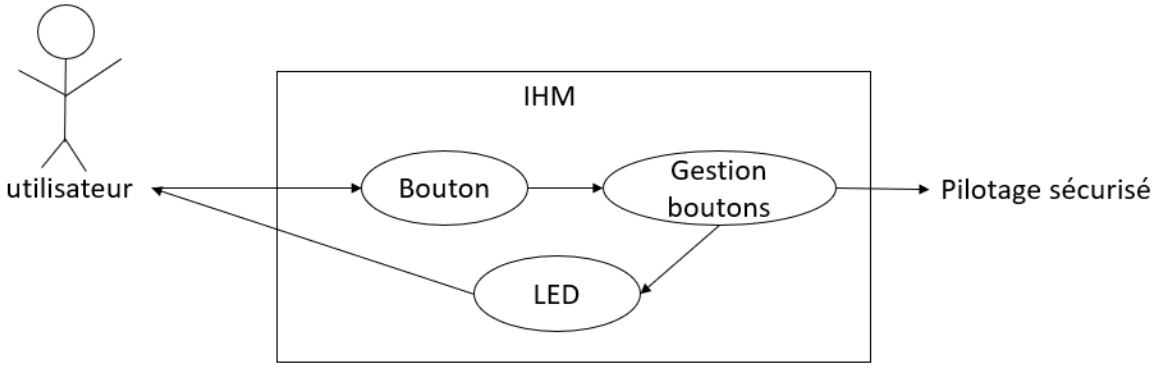
F7 Gestion commandes et indications barreur

Dans cette fonction, l'objectif est de gérer l'interaction des boutons avec le système.

Il y a trois boutons, et 2 types d'action sur les boutons :

- Appui court (moins de deux secondes)
- Appui long (plus de deux secondes)

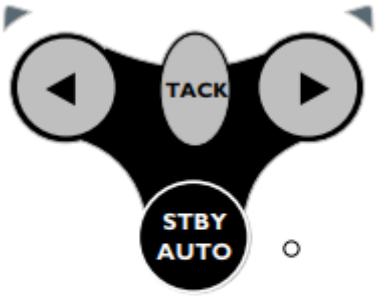
Les boutons enverrons des données à la fonction de pilotage sécurisé, de plus l'utilisateur aura un retour visuel et sonore quand il appuiera sur un bouton.



Pour visualiser l'IHM avec les boutons cf l'image ci-dessous. Nous avons :

- Un bouton **STBY/AUTO** qui permet de changer de mode (mode manuel ou mode auto)
- Un bouton bâbord, représenté par une flèche allant vers la gauche
- Un bouton tribord, représenté par une flèche allant vers la droite

Le bouton **TACK** ne sera pas géré dans ce projet.
Il y aura une LED à côté de chaque bouton



Dans le mode manuel, si on exerce un appui sur un des deux boutons (babord ou tribord), on bouge le cap dans la direction du bouton (tant qu'on est en appui sur le bouton, le cap se déplace). Pour passer dans le mode auto, on a juste à effectuer un appui court sur le bouton STBY/AUTO, la led contiguë à ce bouton clignote dans le mode manuel.

Dans le mode auto**,** le pilote verrouille le cap actuel, la LED contiguë à cette touche arrête de clignoter et reste allumée tant que l'appareil est en mode Pilote Automatique. On peut faire varier notre commande du cap, en appuyant sur les boutons babord et tribord : un appui court permet de faire varier le cap désiré de 1° dans la direction souhaitée et un appui

long fera varier le cap de 10°. Si on exerce un appui prolongé (plus de 2 secondes) sur la touche **STBY/AUTO,** le pilote émet un second “bip”, on verrouille la commande du cap.

Nous avons découpé cette partie en quatre sous-parties :

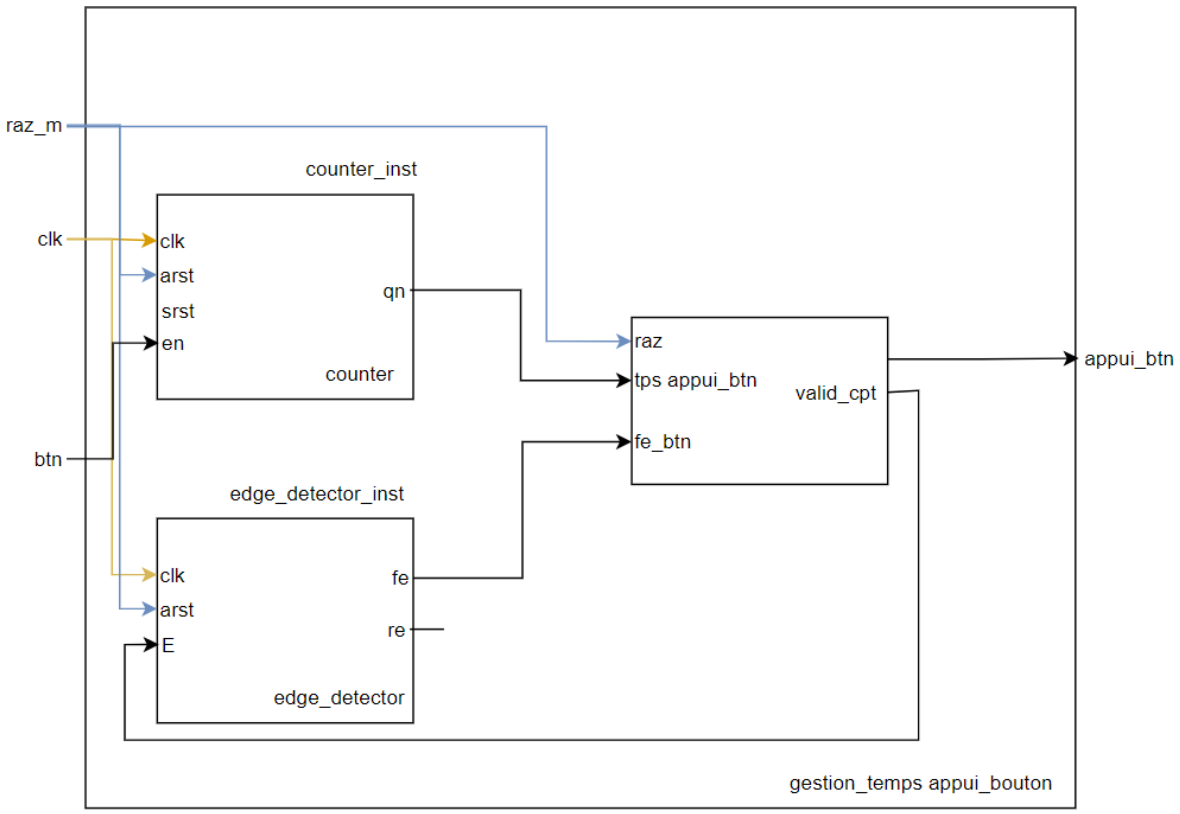
- La gestion du temps d’appui des boutons
- La gestion des différents modes
- La gestion du bip sonore
- La gestion des LED

Gestion temps appui bouton

Nous devons distinguer si l'utilisateur fait un appui court ou un appui long :

- Un appui long est un appui de plus de 2 secondes.
- Un appui court est un appui de moins de 2 secondes.

Ci-dessous le schéma fonctionnel de la gestion du temps d'appui sur un bouton



Explication du schéma fonctionnel :

L'appui sur un bouton déclenche le compteur, tant qu'on reste appuyé sur le bouton on compte le nombre de coups d'horloge. A la détection de relâchement du bouton (front descendant sur le signal qui vient du bouton), on regarde le temps afin de définir l'appui long (plus de 2 secondes) ou l'appui court (moins de 2 secondes).

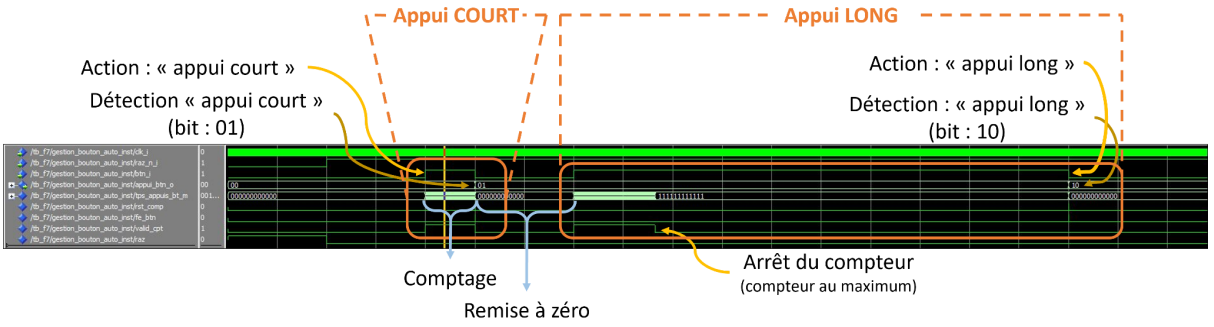
On a donc un signal de sortie sur deux bits qui nous permet de définir l'absence d'appui sur le bouton, l'appui court et l'appui long :

- "00" : absence d'appui
- "01" : appui court
- "10" : appui long

Contrainte : l'utilisateur restera un temps indéfini sur un bouton, donc il faut gérer le fait que l'utilisateur peut rester appuyer plus longtemps que la valeur maximal du compteur (si le compteur dépasse sa capacité maximale, le compte se réinitialisera).

Solution : Pour résoudre ce "problème", on met un signal intermédiaire (valid_cpt) qui permet d'arrêter le compteur si on dépasse le nombre de bit défini.

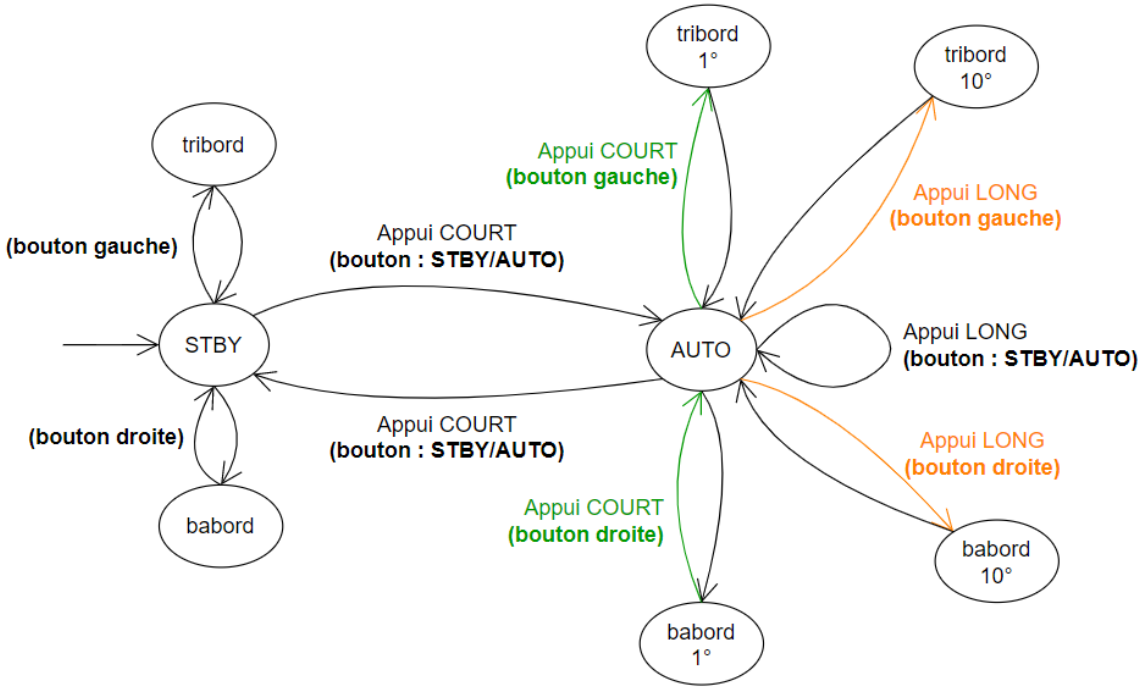
Une fois le code fait, nous l'avons testé sur modelSim. On peut voir le résultat de la simulation sur le chronogramme ci-dessous



Avec ce chronogramme nous pouvons valider la détection de l'appui long ou court et le fait que le compteur ne se réinitialise pas si on dépasse sa capacité.

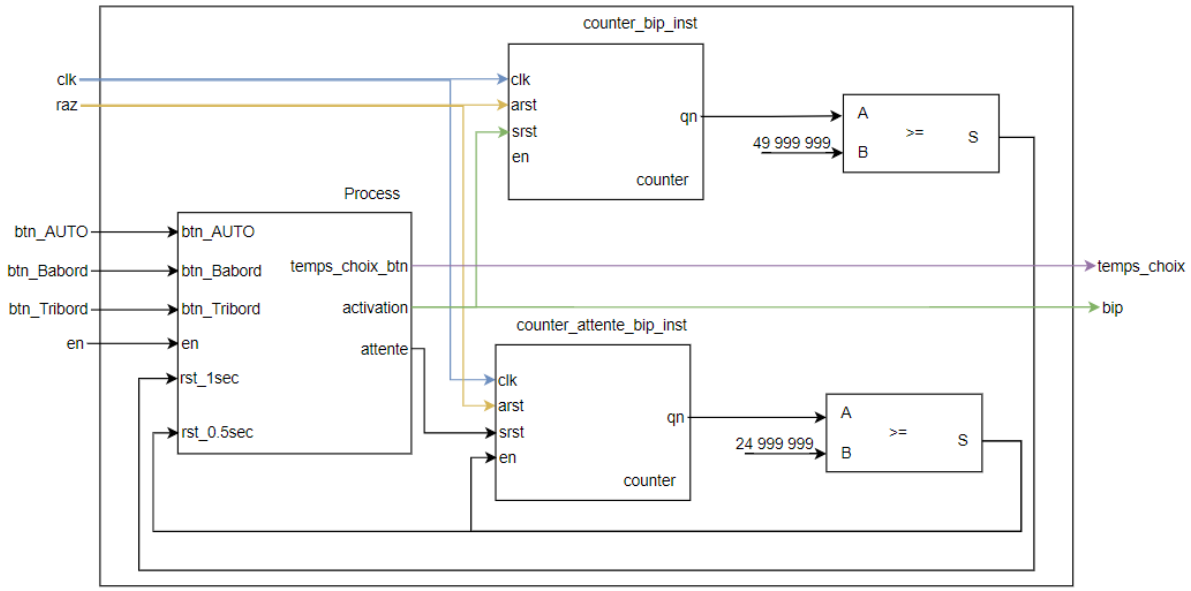
Gestion des différents modes

Pour comprendre l'influence des boutons sur le système nous pouvons nous référer à la machine à état ci-dessous.



Gestion BIP

Chaque appui sur un bouton est confirmé par un signal sonore (un appui long est confirmé par un double bip et un appui court par un bip seul). On construit donc le schéma fonctionnel suivant pour gérer ce signal sonore.

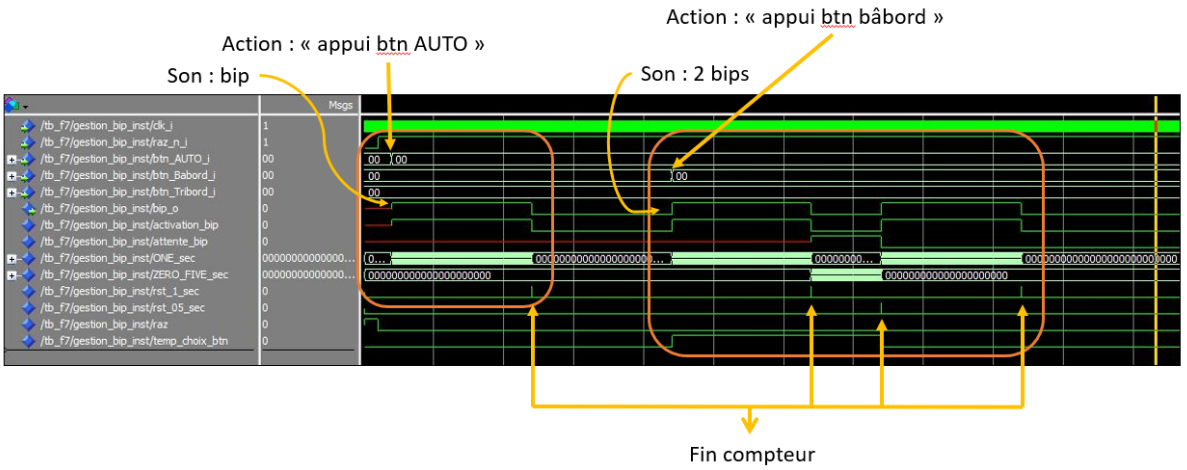


Explication :

On a donc en entrée de notre fonction, l'appui des boutons (appui long ou court). Le process permet de générer les signaux d'activation du bip et d'attente de bip. A partir de ces deux signaux et de compteurs, on peut générer un bip ou un double bip.

On a un compteur, associé à un comparateur, qui nous permet d'allumer le bip pendant une seconde. Et un autre ensemble compteur-comparateur qui nous permet d'attendre 0,5 seconde au cas où on doit générer un double bip.

La simulation de cette fonction nous permet d'obtenir le chronogramme ci-dessous :



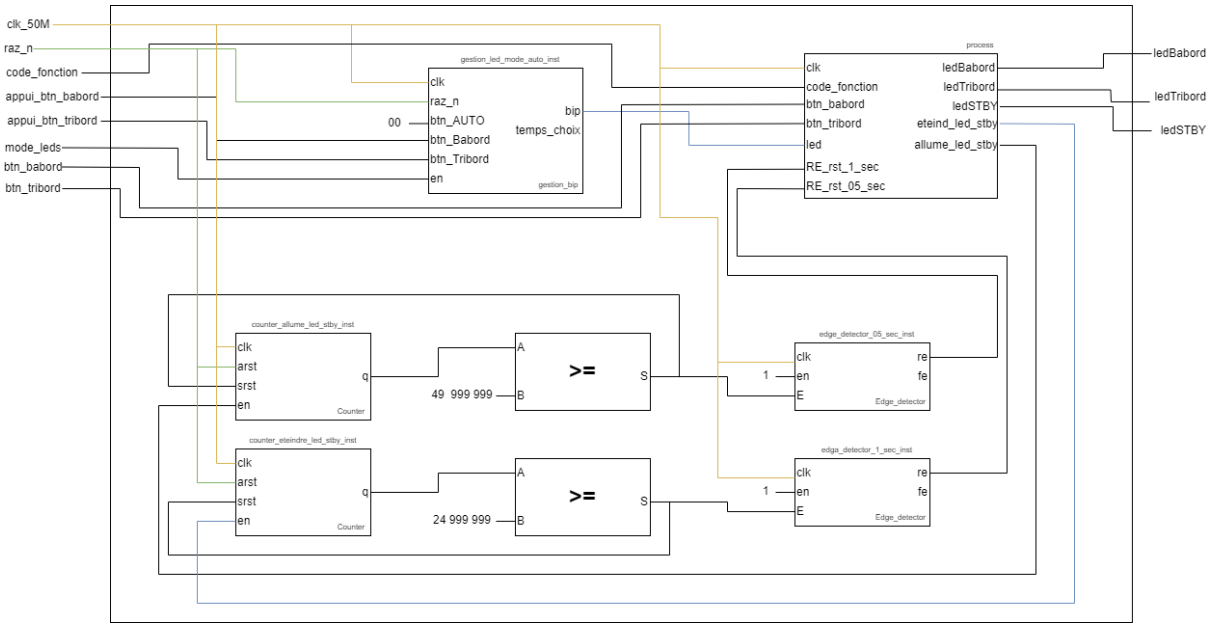
On obtient bien le signal bip désiré en fonction de l'appui de nos boutons, on peut donc valider le fonctionnement de cette fonction.

Gestion LED

La gestion des leds reprend plus ou moins le fonctionnement du bip : un clignotement lors d'un appui court et un double clignotement lors d'un appui long. Ce fonctionnement est valable pour les appui sur les boutons babord et tribord en mode auto. En mode manuel, on allume la led contiguë au bouton lors de l'appui de celui-ci.

Concernant la led à côté du bouton STBY/AUTO, elle clignote quand on est en mode manuel et elle reste allumée quand on est en mode auto.

On a donc le schéma fonctionnel suivant pour la gestion des leds :

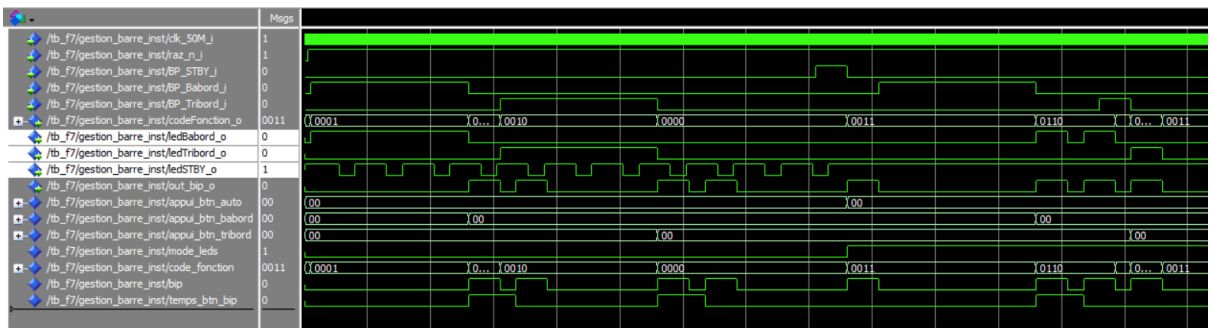


Explication :

On reprend donc le fichier de gestion du bip pour gérer le clignotement des leds babord et tribord en mode auto (mode_led = 1).

Pour gérer la led STBY, on utilise le même principe que dans le fichier gestion bip (compteur et comparateur pour allumer la led une seconde et l'éteindre 0,5 seconde). On gère le clignotement ou l'allumage fixe en fonction de mode_led.

La simulation de cette fonction nous permet d'obtenir le chronogramme ci-dessous :



On obtient les mêmes résultats que pour le test de la fonction gestion bip donc on valide le fonctionnement de gestion led.

Conclusion

Dans ce TP, nous avons pu voir plusieurs problématiques liées au niveau du langage VHDL, comme la synchronisation des signaux pour ne pas avoir de perte de données, la gestion de l'horloge et ses connexions.

Nous n'avons pas pu valider F7 en test réel par manque de temps.

Nous avons d'abord fait tous les schémas blocs et tests de simulation avec modelSim pour valider les différentes fonctions. Mais certaines erreurs ne pouvaient pas être détectées en simulation. Et lorsque nous avons fait les tests réels sur les fonctions entières, nous avons vu des erreurs. Or il est beaucoup plus compliqué de corriger une erreur quand le projet à beaucoup avancé que de tester des petits bouts de fonction. Cela nous a pris beaucoup de temps.

De plus nous pouvons tester uniquement en classe de TP, donc il aurait fallu valider un maximum de petit bout de fonction pour être efficace lorsqu'on rencontre une erreur à la fin du projet.

Annal

Datasheet boussole CMPS03 : http://www.mytopschool.net/mysti2d/activites/polynesie2/ETT/C044/32/BusI2C/files/cmps03.pdf)

Cahier des charges : http://thierryperisse.free.fr/index.php/be-vhdl/