Université Paul Sabatier Master 2 EEA
Parcour SME

EIEAS3GM
Synthèse et mise en oeuvre des systèmes
Rapport BE VHDL

Pilotage de barre franche

Élèves :
Abdoulaye NIANE

Enseignant :
Pedro CARVALHO
MENDES
Thierry PERISSE

9 janvier 2022

Table des matières

1 Introduction

1.1 Présentation du projet

Le cycle de vie d'un système part de l'analyse du besoin. C'est ainsi que le choix de technologie, méthodologie, techniques est primordial pour la conception de ce système.
Dans le cadre de notre formation Master **SME (Systèmes et Microsystèmes embarqués)**, nous avons été amenés à développer un projet dont l'objectif était l'automatisation d'une **barre franche**.

Pour répondre à cette problématique, nous avons fonctionné en bureau d'études en ayant comme objectif de développer une solution logicielle et matérielle répondant à la nécessité de gérer et de contrôler la barre franche d'un bateau.
Plusieurs composants sont nécessaires à son fonctionnement, qui seront présentes plus loin dans ce rapport.
L'objectif est de nous familiariser avec le développement sur FPGA grâce à la programmation **VHDL**.

Pour chaque sous-système, un schéma fonctionnel sera présenté dans chaque partie. Cela permet de poser le contexte et les fonctions à réaliser. Il sera suivi des simulations réalisées sur **Quartus 9**, puis d'une implémentions **C** avec un bus avalon sur **Quartus 11**. Ce qui nous permettra de faire interagir tous les sous-systèmes entre eux.

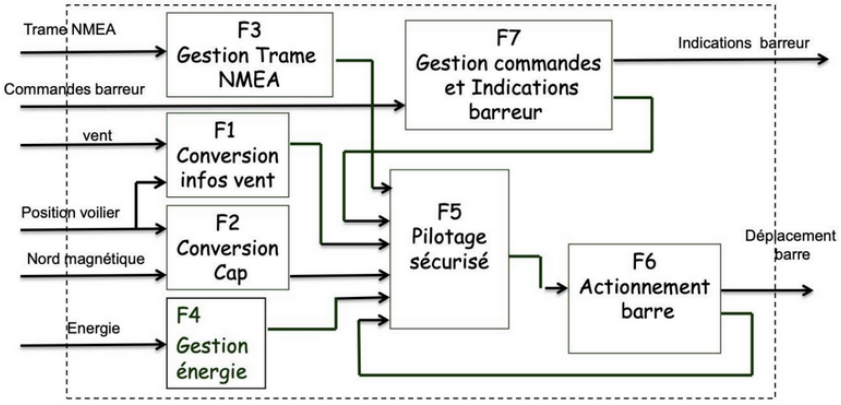

FIGURE 1 – Schéma des différentes architectures à concevoir

1.2 Contexte et objectifs

La mission principale du système est de fournir un contrôle de trajectoire en deux modes de fonctionnement possibles (automatique et manuel). Ainsi, le système doit être capable de récupérer des informations sur la trajectoire (fournis par une boussole), la position de la barre (fournit par un potentiomètre), et de définir le mode de fonctionnement adéquat.
Le système doit aussi disposer d'une interface de communication **NMEA** sur un **standard RS232** pour récupérer la trajectoire réelle d'un **GPS**. Une interface de bouton contenant des LED pour l'affichage de certaines informations et un buzzer, faisant office d'alarme seront implémentées.

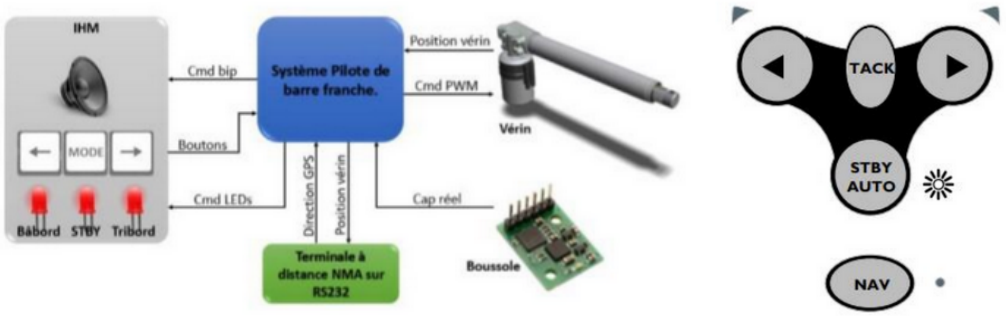

FIGURE 2 – Schéma de principe du vérin et boutons

Ce compte-rendu s'attardera sur les systèmes suivants : **PWM, Acquisition de l'anémomètre, la gestion boutons et leds.**

2 Vue général du projet

Pour réaliser ce projet on peut se baser sur une documentation, définissant les différentes fonctions à réaliser. Celles-ci devront être testées une par une durant le projet puis implémentées sur la carte DE0 Nano pour réaliser la fonction première de notre système. Voici un exemple de notre diagramme fonctionnel général :

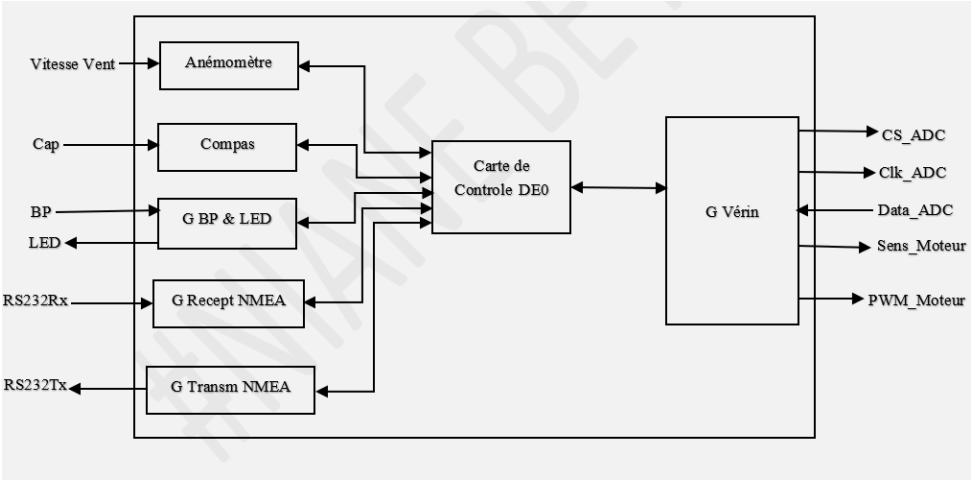

FIGURE 3 – Schéma des différentes fonctions à realiser

- **Le Compas :** donne l'angle suivant la position du bateau, comme une boussole.
- **L'anémomètre :** transmet la vitesse du vent en décimal au système de contrôle.
- **Gestion boutons poussoir et LED :** permet à l'utilisateur d'interagir avec le système en lui donnant la possibilité de choisir un mode particulier (automatique, manuel) et la possibilité de changer de cap. L'indication de l'état du système se fera suivant l'état des LEDS et du bip sonore.
- **Gestion vérin :** commande le moteur du vérin qui permet de bouger la barre suivant les ordres que lui envoi la carte de contrôle.
- **Gestion NMEA :** crée la communication NMEA, envoie et reçoit les différentes informations.

J'ai réalisé la fonction anémomètre et gestion bouton et je l'ai testé. Je vais vous présenter maintenant ces différentes fonctions en détails.

3 Matériel et logiciel

On a eu droit d'utiliser différent matériel et logiciel mis a notre disposition pour réaliser ce projet :

3.1 Partie matérielle

Nous avons eu à disposition différentes cartes électroniques, la première, la DE2 qui m'a permis de tester certaines fonctions, ensuite, j'ai testé sur une autre carte la DE0 nano qui nous était fournie. Enfin une prise en main de la maquette qui comportait une carte DE0 nano, **une interface le bouton LED, un bip sonore et un vérin.**.

3.2 Partie logielle

En premier lieu, j'ai codé toutes mes fonctions sur le logiciel Quartus, en utilisant le langage VHDL. Ensuite de tester les fonctions à l'aide de simulation puis de pouvoir transférer le code sur la carte électronique. Puis j'ai utilisé le logiciel **ALTERA Sopc builder** qui me permets de créer l'interface avalon par laquelle y a toutes les connections entre les diverses entrées et sorties. Et enfin j'ai utilisé le logiciel eclipse qui me permet de coder en langage C les interactions et les valeurs initiales qu'on donnera aux différentes fonctions.

4 Prise en main des logicielles et création de la fonction PWM

Cette interface est nécessaire pour contrôler les différents éléments du système, comme par exemple le contrôle du vérin et pour le calcul de la vitesse du vent. D'autre part, cette interface pourra permettre de valider certaines fonctions comme l'anémomètre. Le principe consiste à fournir la consigne du **duty cycle** ainsi que la fréquence.
Le schéma fonctionnel est défini fig : 4

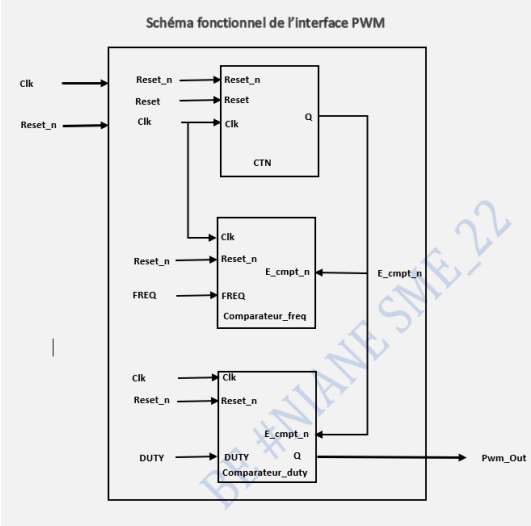

FIGURE 4 – Schéma fonctionnel de l'interface PWM

Compteur(CTN) : pour compter un certain nombre de front montant de l'horloge pour ajuster la valeur du Duty-cycle. Cette valeur est donnée en paramètre par l'entrée duty codé sur 8 bits également.
Comparateur de Freq) :
Comparateur de Duty) :

J'ai réalisé la fonction PWM, puis je l'ai implémenté sur Quartus et ensuite testé sur la carte DE2 à l'aide de l'oscilloscope pour visionner le signal PWM.

5 Réalisation de la fonction Anémomètre

La fonction anémomètre a pour principe de donner la vitesse du vent. voir fig : 5

FIGURE 5 – anémomètre

Il va nous fournir un signal **TTL** d'une certaine fréquence comprise entre **1** et **250 Hz** et nous devons avec notre fonction anémomètre définir la fréquence du signal qui sera la valeur de la vitesse du vent : **1Hz = 1 Km/h et 250Hz = 250Km/h**.
Pour répondre au cahier des charges, voici le schéma fonctionnel réaliser voir fig : 6

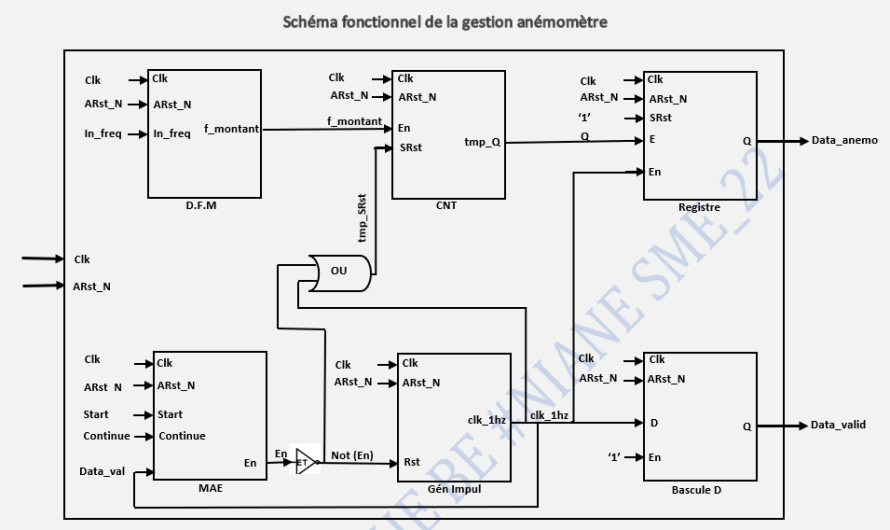

FIGURE 6 – Schéma fonctionnel de la gestion anémomètre

— **Détecteur de Front Montant(DFM)** : Qui prend en entrée une fréquence **in-freq** et en sortie **frontmontant** qui va permettre de détecter un front montant.

— **Générateur d'impulsions** : le générateur 1Hz permet au système d'avoir l'information du moment où les données doivent être actualisées. Ce signal va nous servir à échantillonner signal Infreq d'anémomètre, c'est-à-dire que pendant une période d'horloge, on additionne le nombre de front d'horloge du signal Infreq anémomètre. Ceci va nous donner la valeur de la vitesse transmise par le capteur.

Si on veut générer un signal 1hz il faut calculer la valeur à comparé, on a donc

$$ValToCompare = \frac{freqIn}{freqOut - 1}$$

— **Compteur(CTN)** : compte le nombre de fronts montants provenant de l'entrée infreq.

— **Machine à Etat(MAE)** : pour interagir avec le système, qui peut demander le début de l'acquisition en mode **continu** ou effectuer une mesure simple **(monocoup)**.

— **Bascule D** : pour permettre d'assurer un état de sortie stable

— **Registre** : pour la mémorisation des données

5.1 Principe

Le principe étant que si le signal **continu** ou **start stop** est à **1** on réalise une acquisition de la mesure toutes les secondes. Si le signal **continu** est à **0** et **Data valid** est à **1** on reste en mode repos pas d'acquisition. Le signal de sortie data l'anémomètre représente la vitesse du capteur sur **8 bits** et il est accompagné d'un bit de validité : **data valid** qui va dire si la valeur de data anémomètre est juste s'il est à **1** sinon elle n'est pas à prendre en compte.
voir fig : 7

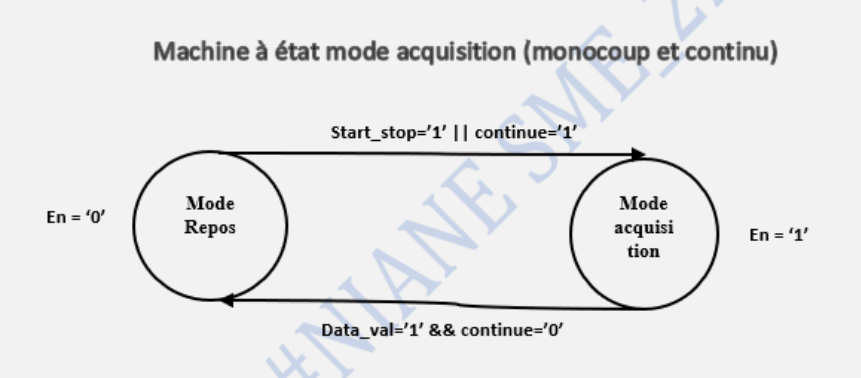

FIGURE 7 – Machine à état mode acquistion

5.2 Teste et Simulation

J'ai ensuite testé la fonction sur la carte DE2 pour apercevoir le résultat de la fonction. Celle-ci m'a donné la valeur de la fréquence que j'avais mise sur infreq anemo. J'ai ensuite validé cette fonction sur le logiciel eclipse en faisant apparaître le résultat de la valeur de data anémomètre sur la console. voir fig : 8

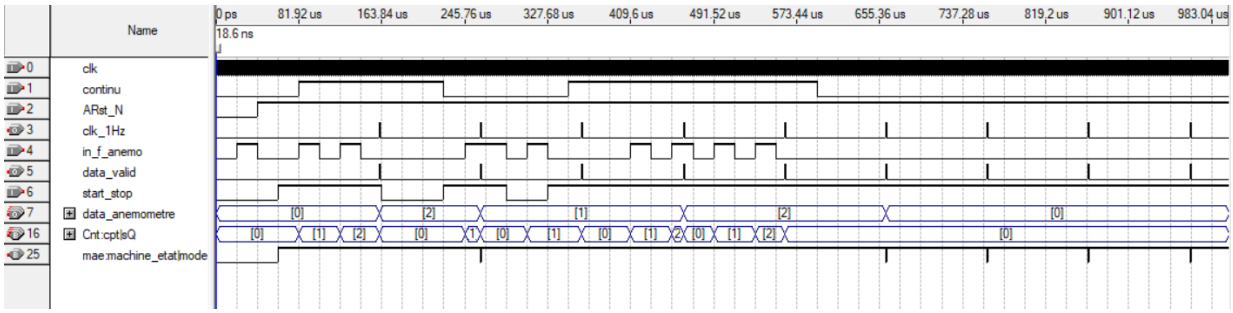

FIGURE 8 – Simulation de l'interface anémomètre

6 Réalisation de la fonction gestion des boutons

La fonction gestion des boutons a pour principe de permettre à l'utilisateur d'interagir avec la carte et de choisir son mode pour barrer, de changer de cap. Il est averti par des leds ou un bip à chaque fois qu'il appuie sur un bouton.
voir fig : 9

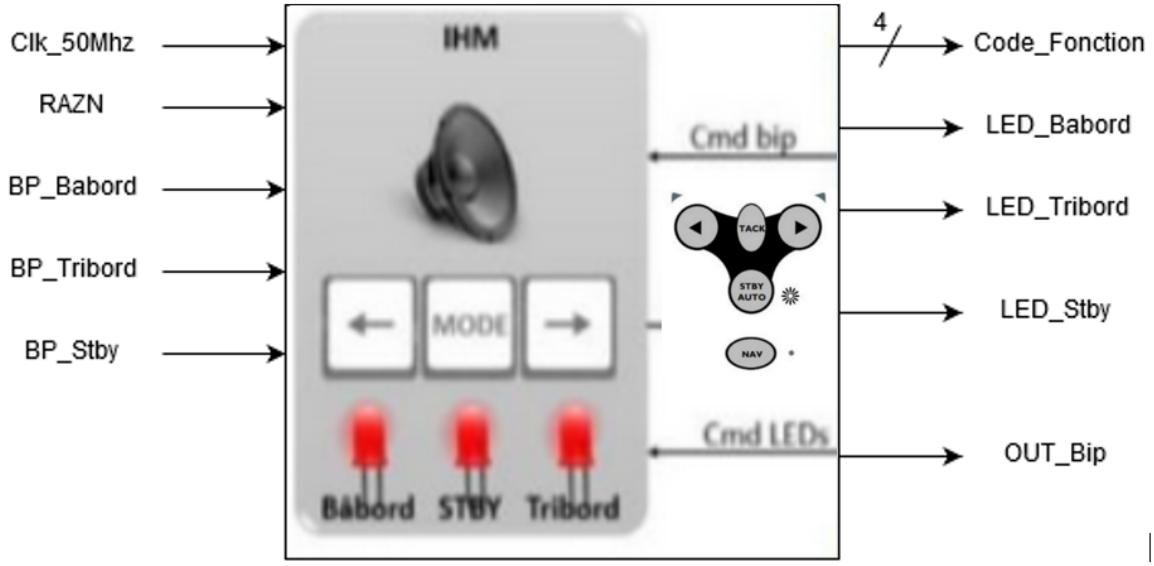

FIGURE 9 – schema gestion des boutons

6.1 Principe

À l'initialisation, le pilote de barre franche est en mode manuel, c'est-à-dire que l'utilisateur, à l'aide des flèches gauches et droites, peu gérer l'angle de la barre. S'il veut garder un cap, il doit réaliser un appui sur le bouton en bas : **STBY AUTO** qui permet de passer en auto pilote, c'est-à-dire que le pilote va garder le cap avec les informations qu'il reçoit des différents capteurs. L'utilisateur peut interagir avec le pilote pendant le mode automatique en lui changeant de quelques degrés de son cap.

On a donc en entrée une horloge **Clk 50MHz**, un reset (**RAZN**) actif à 0 et trois boutons poussoirs actifs à 0 qui sont**BPBabord, BPTribord, BPStby**.
En sortie, on a 3 LEDS : **LEDBabord, LEDBTribord, LEDStby**, la sortie du **OUT-BIP** actif à 0, et le code fonction qui définit l'action qui va être réalisée par le vérin :

code fonction=0000 : pas d'action, le pilote est en veille
code fonction=0001 : mode manuel action vérin babord
code fonction=0010 : mode manuel action vérin tribord
code fonction=0011 : mode pilote automatique/cap
code fonction=0100 : incrément de 1° consigne de cap
code fonction=0101 : incrément de 10° consigne de cap
code fonction=0111 : décrément de 1° consigne de cap
code fonction=0110 : décrément de 10° consigne de cap

J'ai réalisé pour cette fonction le schéma fonctionnel : voir fig : 10

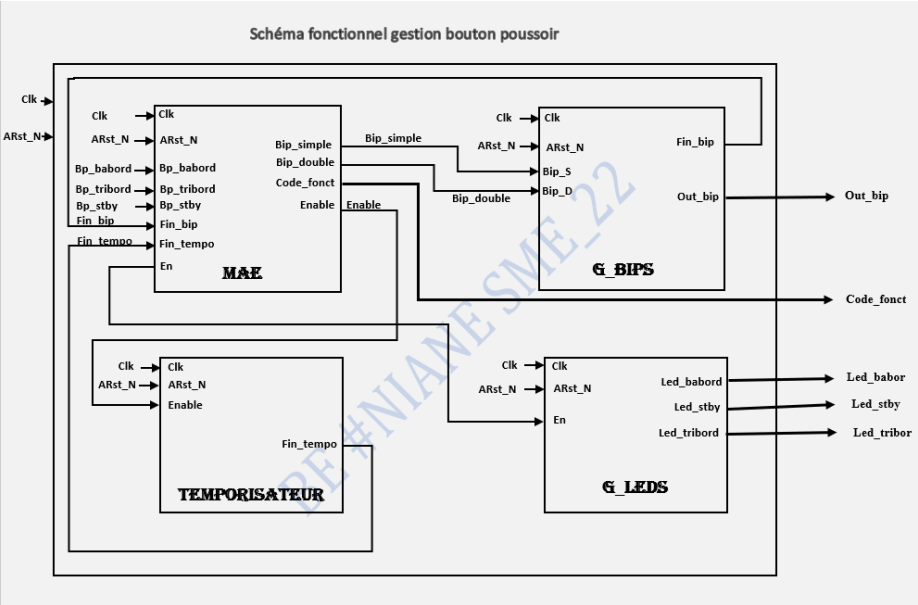

FIGURE 10 – schema fonctionnel de la gestion des boutons

Le principal aspect étant de réalisé aussi des MAE pour la gestion des LEDS et du BIP suivant l'appui des boutons. voir fig : 11
voir fig : 12

6.2 Teste et Simulation

J'ai fait une simulation sur le logiciel Quartus 11 et on voit bien que le résultat attendu est bien atteinte.

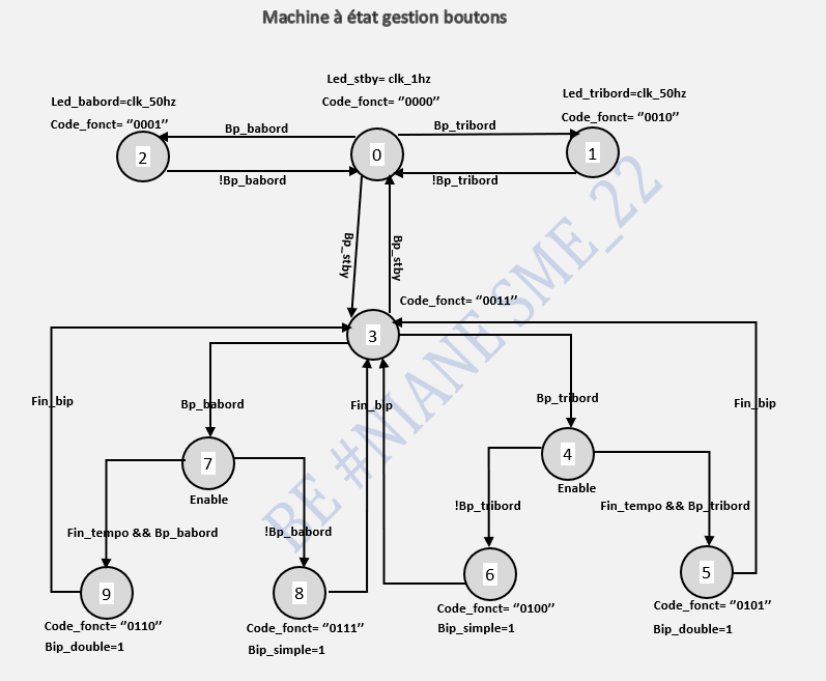

FIGURE 11 – MAE de la gestion des boutons

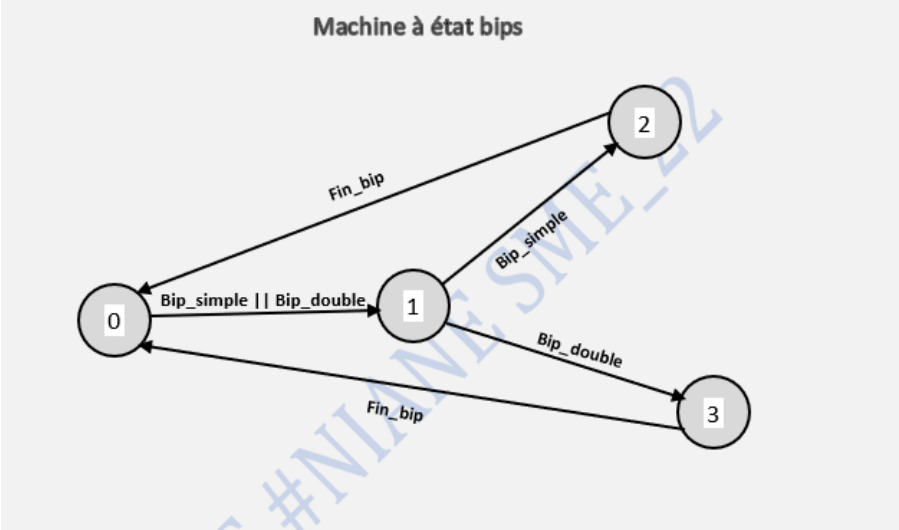

FIGURE 12 – MAE de la gestion des Bips

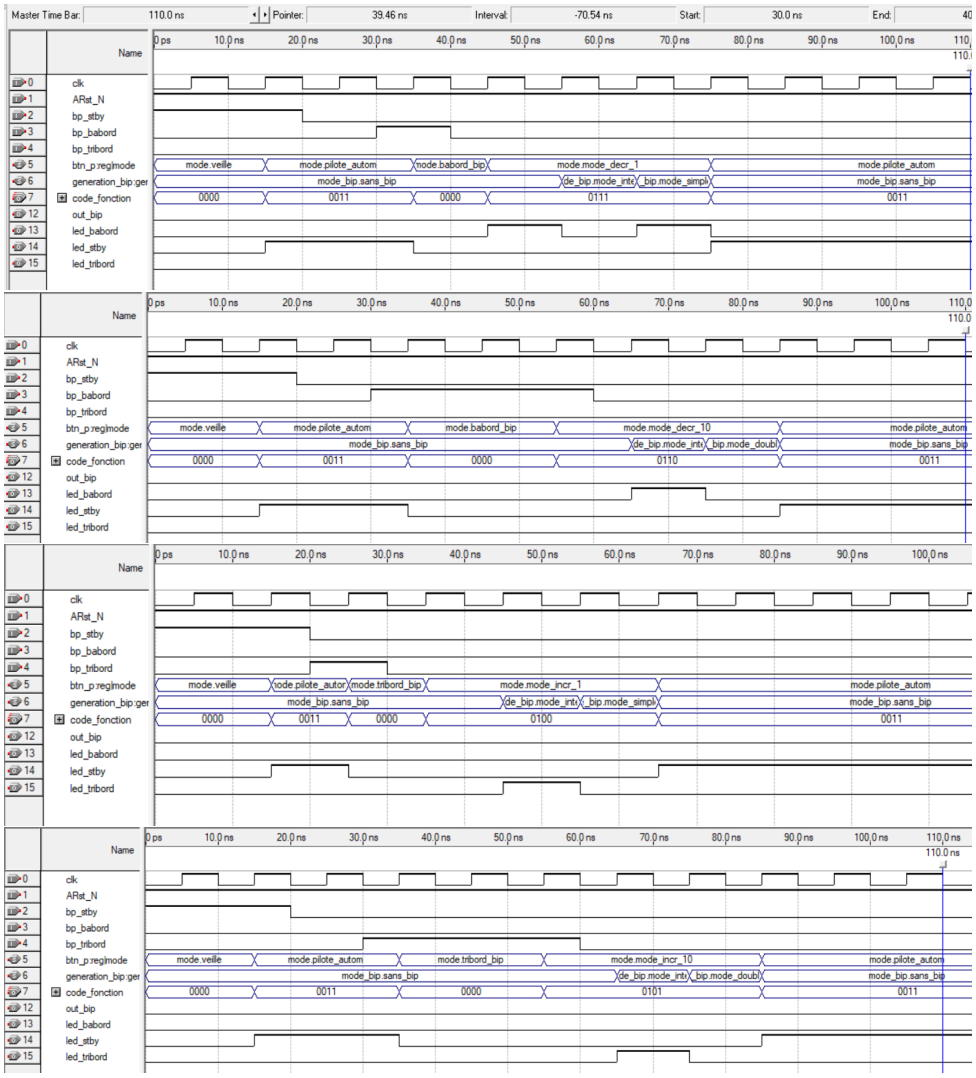

FIGURE 13 – Simulation de l'interface gestions des boutons

7 Explication Bus Avalon

Durant tout ce projet, j'ai créé toutes les fonctions pour qu'elles s'intègrent au bus avalon qui permet l'échange entre ces différentes fonctions. À l'aide du logiciel eclipse on peut interagir sur les différentes entrées et visionner sur un terminal les valeurs des différentes sorties. voir fig : 14

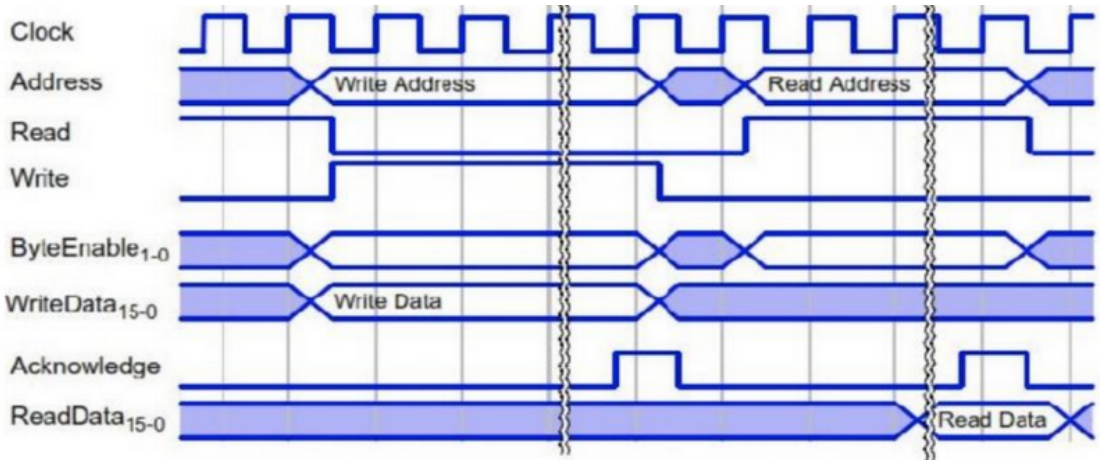

FIGURE 14 – Trame Bus Avalon

On peut définir les paramètres qui seront utilisés lors de la programmation pour l'utilisation de la bus avalon :

- **Adresse :** qui permet de sélectionner le composant et d'y effectuer une action.
- **Read :** Boolean, active ou désactive la lecture de données.
- **Write :** Boolean, active ou désactive l'écriture de données.
- **WriteData :** Bus d'écriture des données.
- **Read Data :** Bus de lecture des données.
- **Ackowledge :** Renvoie true si les données sont bien reçues.

7.1 Développement et intégration du système au SOPC

L'intégration de mes différentes fonctions a été faite en utilisant l'outil SOPC Builder (System On Programmable Chip Builder) de Quartus, qui permet de générer automatiquement des systèmes sur une puce en connectant des composants et de concevoir des microcontrôleurs spécifiques à une application précise.
J'ai intégré les circuits périphériques et un ensemble de composants. fournis par la bibliothèque de SOPC Builder sous Quartus 11.0. Les composants ajoutés sont les suivants :

- Un processeur NIOS II de 32 bits qui exécutera le logiciel de contrôle de trajectoire.
- Une mémoire SRAM de 20Ko.
- Le composant JTAG qui fournit une interface entre le SOPC et l'environnement NIOS II pour réaliser la programmation et le débogage de logiciel.
- Notre bus Avalon qui a été créé sur Quartus 11 pour la communication.
- Le programme VHDL.

Voici le SOPC défini pour l'anémomètre voir fig : 15
Voici les entrées et sorties que l'on doit obtenir à la fin du projet : voir fig : 16

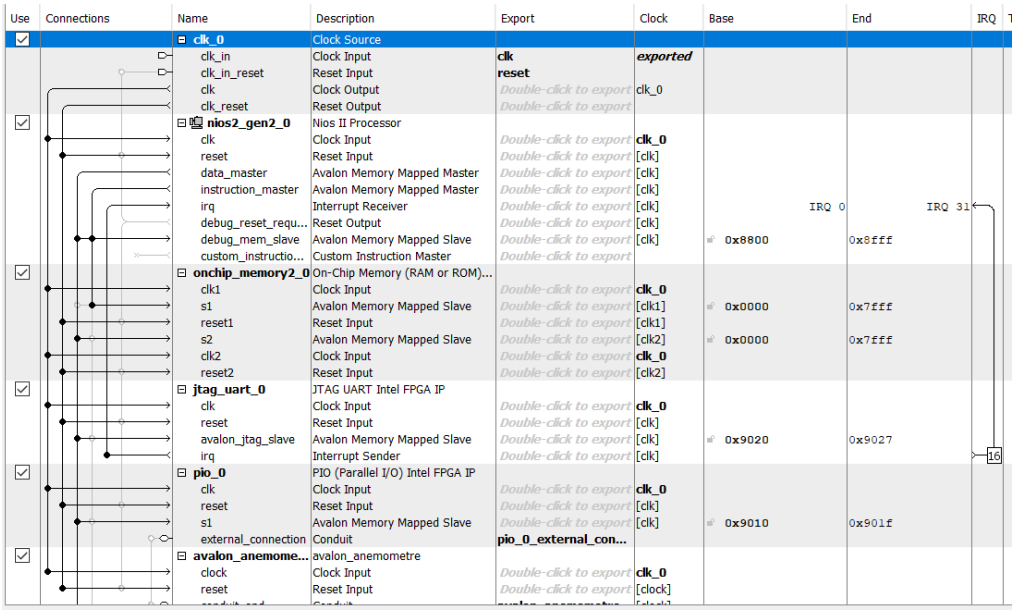

FIGURE 15 – Composants SOPC, anémomètre

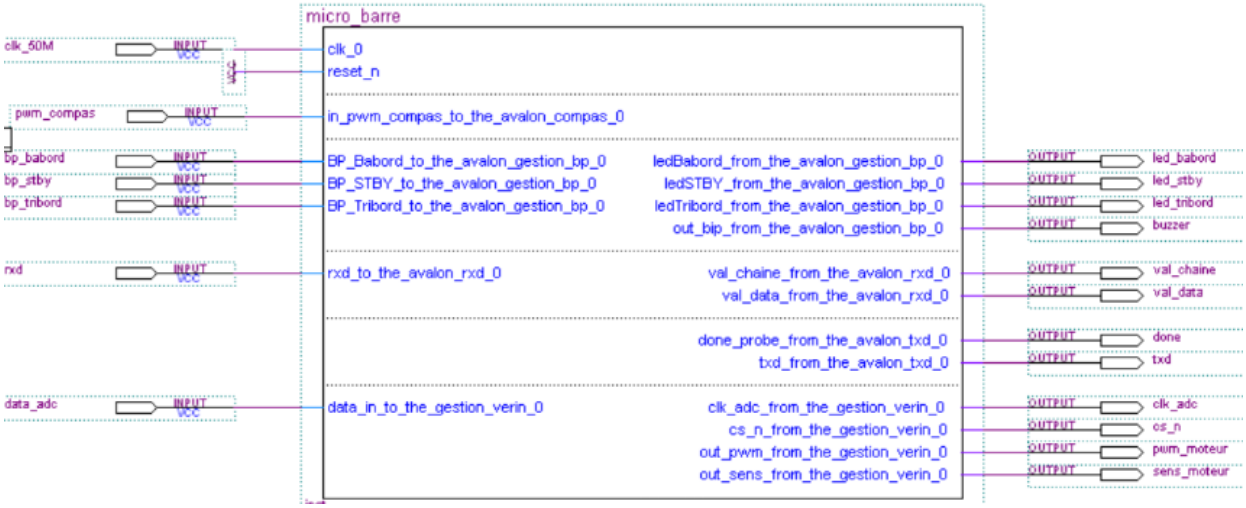

FIGURE 16 – Symbole du SOPC complet version carte DE0 Nano

8 Conclusion

Ce projet m'a apporté de multiples compétences, notamment en programmation en langage VHDL et aussi la programmation des cartes et de nouvelles interfaces. De plus, le logiciel m'a permis de connaître la taille et de l'implémentation de code fonction dans une carte afin d'optimiser mon code VHDL. Ce bureau d'étude est donc aussi une discipline transverse avec nombre de matière du master systèmes et microsystèmes embarqués.