UNIVERSITE TOULOUSE III PAUL SABATIER

Université de Toulouse

Faculté Sciences et Ingénierie

Master 2 SME

Rapport BE : Pilote Barre Franche

Réalisé par :

- MEKHATRI Mehdi
- Yala Ali Amine

Encadré par

- Pedro Carvalho
- Thierry PERISSE

Année universitaire : 2022/2023

Tables des matières

I. Introduction 3
II. Présentation du projet 4
II. Fonctions simples : 6
I.1 Fonction simple gestion anémomètre 6
I.1.1 Schéma bloc de la fonction gestion anémomètre 6
I.1.2 Implémentation du code vhdl gestion anémomètre 7
I.1.3 Validation - Simulation de la fonction gestion anémomètre sous Modelsim 7
I.1.4 Mise en œuvre de la fonction anémomètre et test sur la carte DE0-NANO 8
I.2 Fonction simple gestion compas 8
I.2.1 Schéma bloc de la fonction gestion compas 9
I.2.2 Implémentation du code vhdl gestion compas 9
I.2.3 Validation - Simulation de la fonction gestion compas 10
I.2.4 Mise en œuvre de la fonction compas et test sur la carte DE0-NANO 10
II. Fonction complexe – gestion vérin 11
II.1 Développement de la partie vhdl du vérin 12
II.2 Bloc gestion du convertisseur MCP3201 12
II.3 Développement de l'interface Avalon en vhdl 13
II.4 Implémentation du processeur Nios sur Platform designer 14
II.5 Partie programmation logicielle en C 15
II.6 Validation PWM – commande vérin partie VHDL 17
Conclusion : 18

I. Introduction

Afin de mettre en avant nos compétences acquises durant notre formation, nous devions réaliser un projet qui consiste à concevoir un système de pilotage de barre franche. Ceci est dans le cadre la validation de l'unité d'enseignement Synthèse et mise en œuvre de systèmes. L'objectif principal est de développer une solution Hardware/Software qui répond au cahier des charges.

Le projet consiste à réaliser un système de contrôle de trajectoire d'un voilier de barre franche. L'objectif est de partir d'une carte FPGA (Altera) sur laquelle nous mettrons en œuvre ce projet.

Nous passerons par une étape d'analyse afin d'étudier et de décomposer le travail, ensuite nous coderons et implémenteront ces dernières. A la dernière étape, nous devrons développer un bus Avalon afin de mettre en harmonie tout le projet. Ce dernier permet de créer des connexions entre les fonctions et le MCU du FPGA.

Pour se faire, nous devons suivre les étapes du cycle en V qui nous ont guider de l'expression du besoin jusqu'à la recette finale. Il est nécessaire de bien analyser le besoin afin d'en faire des exigences qui seront ensuite développé et coder pour nous mener aux étapes de test. Grâce à l'enseignant qui nous a préparé les étapes d'analyse en décomposant le système en sous-systèmes de fonctions simple et complexe nous avons pu gagner du temps sur les étapes en Amont du projet. Pour finir nous devons arriver à l'étape de validation de notre travail sur la maquette.

II. Présentation du projet

Introduction technique :

Dans ce projet on nous demande de travailler sur un FPGA embarqué (Altera) qui doit être utilisé pour contrôler une barre franche sur un voilier.

Ce projet comprend plusieurs dispositifs électroniques qui agissent comme des capteurs et des actionneurs du système. On peut citer : anémomètre, gyroscope, GPS, actionneurs, boutons, leds...

Description générale du système :

La réalisation de ce système doit mettre en avant les compétences acquises durant la formation Systèmes et Microsystèmes Embarqués, notamment en VHDL et en C embarqué.

Ce dernier doit pouvoir répondre aux critères de bases suivants :

- Le système devra utiliser 2 appareils de mesure :
  - Anémomètre : capteur de vitesse de vent
  - Compas : donne le cap actuel
- Le système devra avoir comme actionneur un Vérin change le cap du voilier
- Une trame NMEA devra être utilisée de communiquer les données entre la carte FPGA et l'interface
- Des boutons et des LEDs devront faire office d'IHM

Le système est décrit brièvement sur la figure qui suit :

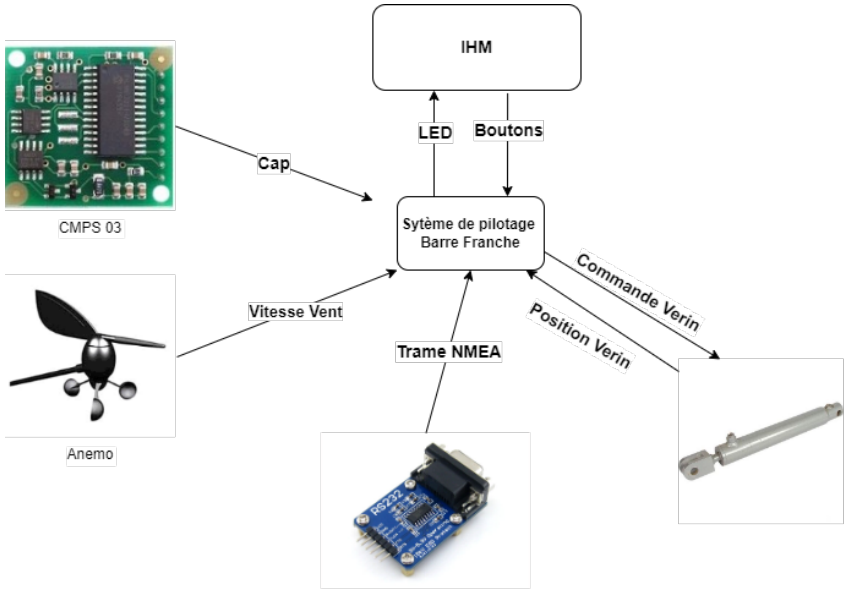

Figure 1 – Schéma bloc du projet

Description technique du système :

Afin de réaliser ce Bureau d'Etude, le système a été divisé en sous-systèmes (des fonctions) par l'enseignant. On y trouve des fonctions simples et des fonctions complexes.

Les fonctions peuvent comportées des partie Software et/ou Hardware.

La partie Hardware représente la partie du travail sur le FPGA (Altera pour notre cas).

La partie Software est la partie intégrée/développée dans le FPGA.

Ces deux dernières parties communiqueront à l'aide d'un BUS AVALON. Ce BUS a été développé par Altera pour leur SOPC.

La carte qui nous a été confiée est une carte FPGA Cyclone IV EP4CE22F17C6N d'Altera.

Cette dernière permet de développer les deux parties Software et Hardware sur la même carte.

Les différentes fonctions qu'il nous a été demandé de développer sont synthétisées sur ce schéma :

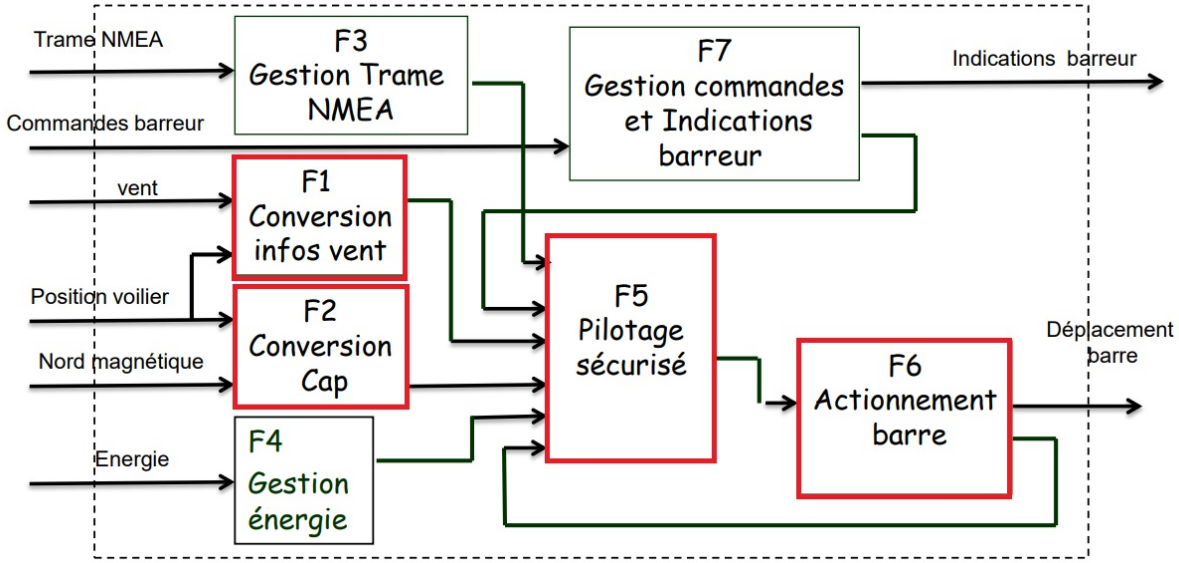

Figure 1 - Architecture du projet

Les fonctions encadrées en rouge sont les fonctions qui nous avons eu le temps de faire. La contrainte de temps nous a malheureusement empêchés d'aller jusqu'au bout du projet.

Afin de bien mettre en œuvres toutes les parties de ce projet, nous avons dû passer par la phase d'analyse comportementale de chaque bloc qui devait être exécuté, créer des composants simples d'abord, qui ont été utilisés pour réaliser les fonctions plus complexes.

La mise en œuvre de chaque composant a été développée sur le logiciel Quartus. L'objectif de cette démarche est de réaliser différents blocs qui sont mis en commun. De cette manière nous avons développé de blocs simples de composants qui sont ensuite utilisés dans l'architecture générale du projet.

II. Fonctions simples :

I.1 Fonction simple gestion anémomètre

Cette fonction sert à mesurer la vitesse du vent à l'aide d'un anémomètre, le dispositif matériel choisi mesure la vitesse du vent et délivre un signal carré en sortie de fréquence variable en fonction de la vitesse du vent, la plage de mesure du capteur est comprise entre 0 – 250 km/h, et délivre en sortie un signal carré de fréquence correspondant à la vitesse du vent, soit 0 – 250 Hz.

Le but est de réaliser un code en vhdl permettant de lire la fréquence délivrée par l'anémomètre et la convertir en un nombre binaire 8 bits représentant une fréquence entre 0 et 255.

Deux modes de fonctionnement sont exigés :

- Mode continu : La donnée est rafraichie toutes les secondes
- Mode monocoup : Une seule acquisition est effectuée lorsque start_stop = 1, une fois l'acquisition terminée, data_valid passe à 1, et se remet à 0 une fois start_stop = 0. Afin de lancer une nouvelle acquisition il faut réactiver start_stop.

I.1.1 Schéma bloc de la fonction gestion anémomètre

Ci-dessous le schéma bloc général du gestion anémomètre

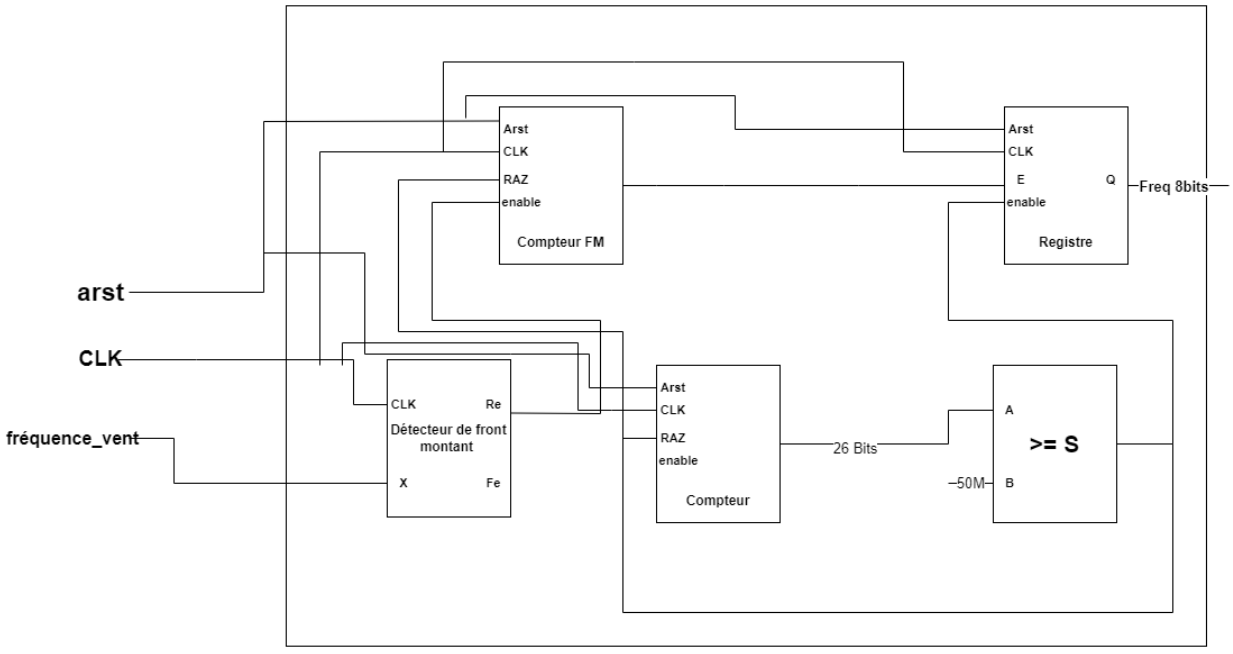

Figure 3 – Schéma bloc gestion anémomètre

Le schéma est constitué de 5 blocs principaux, un détecteur de front montant, un comparateur, un registre et deux compteurs.

Le détecteur de front montant permet de détecteur les fronts montants du signal d'entrée qui représente le vent, ce détecteur est suivi d'un compteur 8 bits permettant de compter le nombre de ces fronts pendant une durée de 1 seconde, et ça à l'aide d'un autre compteur qui s'incrémente

après chaque front montant du signal d'horloge, suivi d'un comparateur permettant d'activer le registre de sortie et remettre le compteur à zéro après chaque 1s.

I.1.2 Implémentation du code vhdl gestion anémomètre

Le code vhdl développé contient les blocs nécessaires à la mise en œuvre du gestion anémomètre, à savoir un compteur, un détecteur de fronts montants, un comparateur, un registre, et un autre fichier principal top qui permet de faire les interconnections entre tous les blocs et l'implémentation d'une machine à état qui définit les modes de fonctionnement anémomètre.

Après compilation du projet, on obtiendra le schéma bloc ci-dessous généré par Quartus

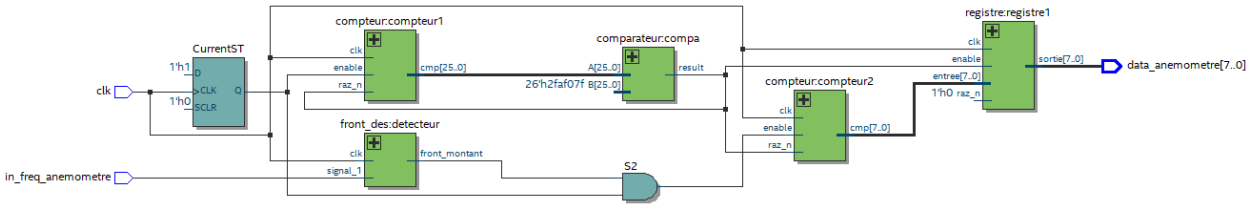

Figure 4 – Schéma bloc gestion anémomètre généré par Quartus

I.1.3 Validation - Simulation de la fonction gestion anémomètre sous Modelsim

Afin de simuler notre fonction, il va falloir créer un autre fichier testBench non synthétisable permettant de simuler le comportement de la fonction gestion anémomètre.

Vu qu'il n'est pas pratique de faire une simulation pendant 1s, nous avons réduit l'intervalle de temps de mesure à 100us, et ça en mettant la valeur d'entrée du comparateur à N = 5000, au lieu de 50M.

Nous avons simulé un vent de fréquence 10us, la valeur de sortie doit être égale à N=10. C'est ce que nous avons vérifié via la simulation comme le montre la figure ci-dessous.

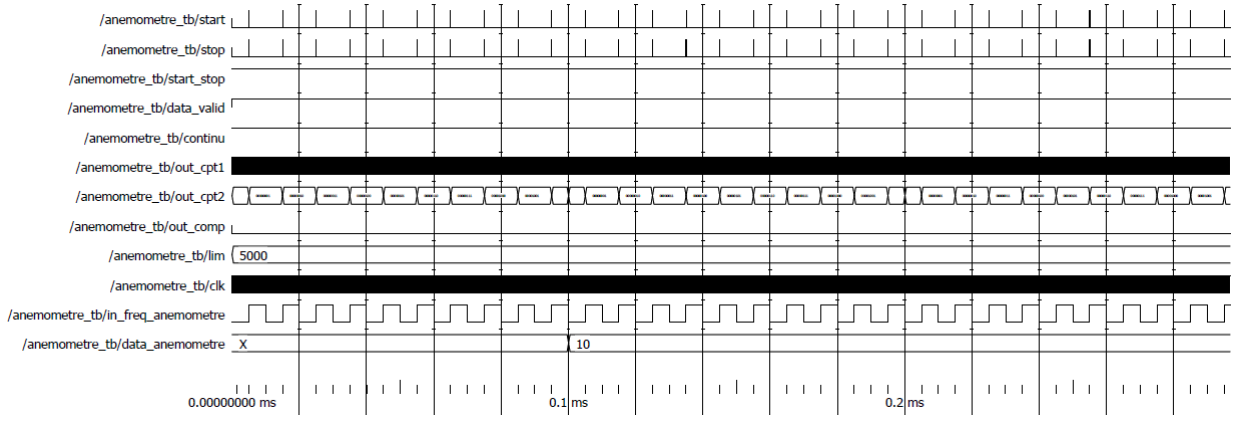

Figure 5 – Résultat de Simulation gestion anémomètre sur ModelSim

I.1.4 Mise en œuvre de la fonction anémomètre et test sur la carte DE0-NANO

Pour tester notre fonction sur la carte, on a utilisé un GBF pour simuler la sortie de l'anémomètre d'une fréquence allant de 0 à 255Hz, l'entrée du signal a été affectée à une entrée GPIO du DE0-NANO et la sortie 8 bits a été affectée aux LEDs embarquées sur la carte afin d'observer directement la sortie, les affectations sur Pin Planner sont montrées dans la figure ci-dessous.

Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
altera_reserved_tck	Input				PIN_H3	2.5 V (default)		8mA (default)			
altera_reserved_tdi	Input				PIN_H4	2.5 V (default)		8mA (default)			
altera_reserved_tdo	Output				PIN_J4	2.5 V (default)		8mA (default)	2 (default)		
altera_reserved_tms	Input				PIN_J5	2.5 V (default)		8mA (default)			
clk	Input	PIN_R8	3	B3_N0	PIN_R8	2.5 V		8mA (default)			
data_anemometre[7]	Output				PIN_D8	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[6]	Output				PIN_C8	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[5]	Output				PIN_C9	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[4]	Output				PIN_A7	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[3]	Output				PIN_E9	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[2]	Output				PIN_E8	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[1]	Output				PIN_B7	2.5 V (default)		8mA (default)	2 (default)		
data_anemometre[0]	Output				PIN_F8	2.5 V (default)		8mA (default)	2 (default)		
in_freq_anemometre	Input				PIN_E16	2.5 V (default)		8mA (default)			

Figure 6 – Affectation des pins sur Pin Planner gestion anémomètre

I.2 Fonction simple gestion compas

Le but de cette fonction est d'implémenter une structure permettant de mesurer le temps d'état haut d'un signal PWM. Ce signal est généré par un module capteur boussole permettant de mesurer la direction et donner l'angle sous forme d'un signal PWM.

Le composant utilisé est le CMPS03, le signal PWM généré par ce capteur représente un angle entre 0 et 360 degrés, l'état haut varie entre 1ms pour $\theta = 1°$ et 36.99 ms pour $\theta = 359.9°$. L'état bas est toujours fixe et égale à 65ms.

Comme pour l'anémomètre, deux modes de fonctionnement sont exigés :

- Mode continu : La donnée est rafraichie toutes les secondes
- Mode monocoup : Une seule acquisition est effectuée lorsque start_stop = 1, une fois l'acquisition terminée, data_valid passe à 1, et se remet à 0 une fois start_stop = 0. Afin de lancer une nouvelle acquisition il faut réactiver start_stop.

CMPS03 – Robot Compass Module

Ci-dessous, le module CMPS03 utilisé dans ce projet

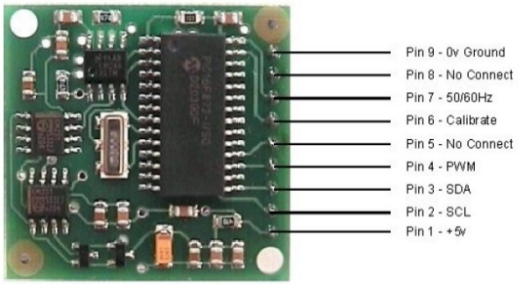

Figure 7 – Module Boussole CMPS03

I.2.1 Schéma bloc de la fonction gestion compas

Ci-dessous le schéma bloc général du gestion compas

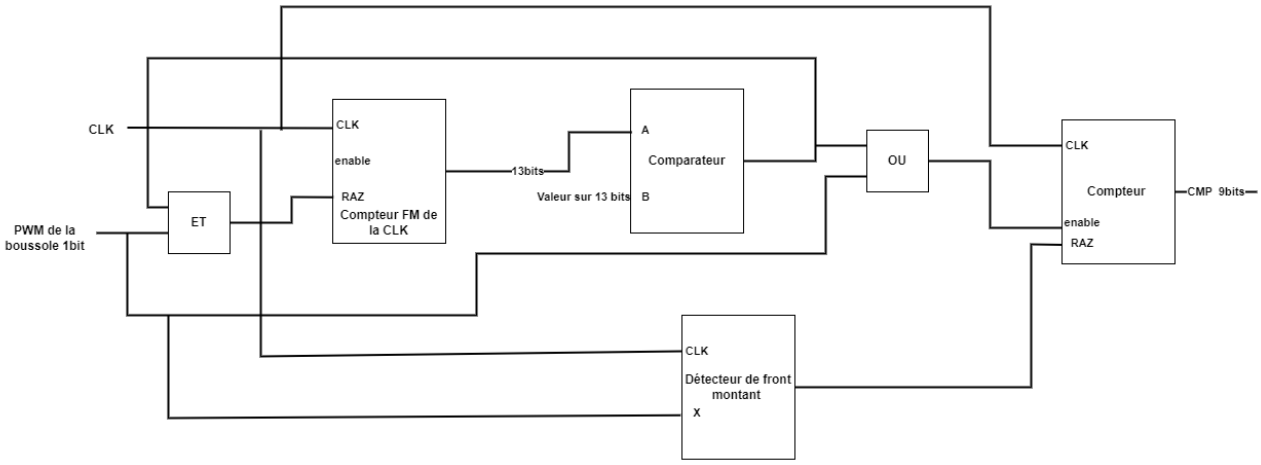

Figure 8 – Schéma bloc de la fonction gestion compas

Le schéma est constitué de 4 blocs principaux, un détecteur de front montant, un comparateur et deux compteurs.

Le premier compteur permet de détecter le nombre de fronts montants de l'horloge pendant que le signal PWM est à l'état haut, suivi d'un comparateur, les deux jouant le rôle d'un diviseur de fréquence, qui sert à incrémenter le deuxième compteur 9 bits après chaque intervalle de temps T défini par la valeur d'entrée B du comparateur. Une fois le signal PWM passe à 0, la sortie du deuxième comparateur représente directement la valeur du temps pendant lequel le signal PWM est resté à l'état haut, et donc la valeur de l'angle peut être calculé.

Le détecteur de fronts montants remet à zéro le deuxième comparateur après chaque front montant du signal d'entrée PWM, et ce afin de lancer une nouvelle mesure.

I.2.2 Implémentation du code vhdl gestion compas

Le code vhdl développé contient les blocs nécessaires à la mise en œuvre du gestion compas, à savoir un compteur, un détecteur de fronts montants, un comparateur, et un autre fichier principal top qui permet de faire les interconnections entre tous les blocs et l'implémentation d'une machine à état qui définit les modes de fonctionnement gestion compas.

Après compilation du projet, on obtiendra le schéma bloc ci-dessous généré par Quartus

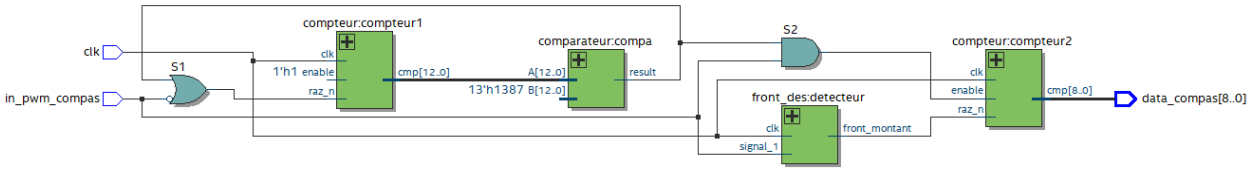

Figure 9 – Schéma bloc de la fonction gestion compas généré par Quartus

I.2.3 Validation - Simulation de la fonction gestion compas

Pour simuler notre fonction, nous avons développé un testBench permettant de :

- Générer un signal PWM d'un état haut th = 10ms et d'un état bas tb = 65ms
- Le compteur de sortie s'incrémente chaque 100us, donc on a pris une valeur de N = 5000 en entrée B du comparateur.
- La sortie dans ce cas doit être égale à N = $\frac{10\ x\ 10^{-3}}{100\ x\ 10^{-6}} - 1 = 99$.

La figure ci-dessous représente le résultat de simulation sur ModelSim

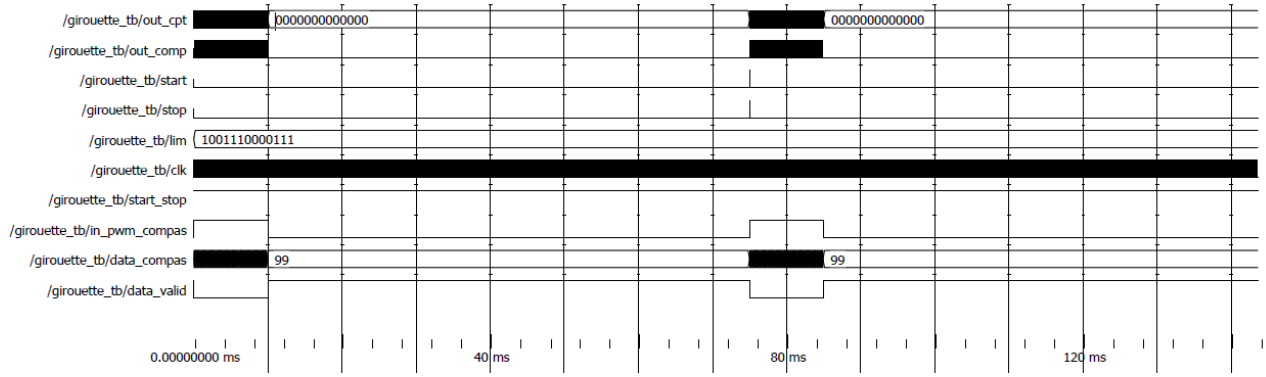

Figure 10 – Résultats de simulation gestion compas sur ModelSim

I.2.4 Mise en œuvre de la fonction compas et test sur la carte DE0-NANO

Nous avons utilisé la maquette du projet pour pouvoir tester notre fonction gestion compas, la sortie PWM du module CMPS03 est reliée au pin T14 de la carte DE0-NANO, comme l'indique la figure ci-dessous

Figure 11 – Brochage des pins maquette Projet / DE0-Nano

Comme pour l'anémomètre, afin de pouvoir observer la sortie nous avons affecter la sortie aux LEDs embarqués sur la carte DE0-NANO, nous avons utilisé également l'analyseur logique de Quartus.

La figure ci-dessous montre l'affectation des pins sur Pin Planner

Named: * Edit: Filter: Pins: all

Node Name	Direction	Location	I/O Bank	VREF Group	Fitter Location	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair	Strict Preservation
in_pwm_compas	Unknown	PIN_T14	4	B4_N0		2.5 V (default)		8mA (default)			
data_compas[0]	Unknown	PIN_A15	7	B7_N0		2.5 V (default)		8mA (default)			
data_compas[1]	Unknown	PIN_A13	7	B7_N0		2.5 V (default)		8mA (default)			
data_compas[2]	Unknown	PIN_B13	7	B7_N0		2.5 V (default)		8mA (default)			
data_compas[3]	Unknown	PIN_A11	7	B7_N0		2.5 V (default)		8mA (default)			
data_compas[4]	Unknown	PIN_D1	1	B1_N0		2.5 V (default)		8mA (default)			
data_compas[5]	Unknown	PIN_F3	1	B1_N0		2.5 V (default)		8mA (default)			
data_compas[6]	Unknown	PIN_B1	1	B1_N0		2.5 V (default)		8mA (default)			
data_compas[7]	Unknown	PIN_L3	2	B2_N0		2.5 V (default)		8mA (default)			

Figure 12 – Schéma bloc de la fonction gestion compas

II. Fonction complexe – gestion vérin

Le circuit de gestion du vérin est constitué de quatre fonctions principales :

- Gestion de la PWM moteur :
  Le bloc PWM sert à générer un signal PWM permettant de contrôler le vérin en jouant sur la fréquence et le rapport cyclique.

- Contrôle des butées de fin de course du vérin :
  Ce bloc met le signal « PWM » à 0 si « angle_barre » se situe en dehors des butées « butee_g » et « butee_d » et selon le sens de rotation du moteur.

- Gestion du convertisseur AN MCP 3201 de recopie de position de barre :
  Ce bloc permet la gestion du convertisseur MCP3201 en utilisant une machine à états afin de contrôler le sens de rotation du vérin.

- Interface avec le bus Avalon du NIOS :
  Cette partie consiste à implémenter le processus NIOS et le développement du bus Avalon permettant de faire le lien entre la partie matérielle faite en VHDL, et la partie logicielle développée en C sous Eclipse.

II.1 Développement de la partie vhdl du vérin

Nous avons développé un code vhdl permettant d'implémenter les trois blocs de la partie vérin, à savoir le bloc de génération PWM, le bloc gestion butées et le bloc de gestion du convertisseur MCP3201, comme le montre la figure ci-dessous :

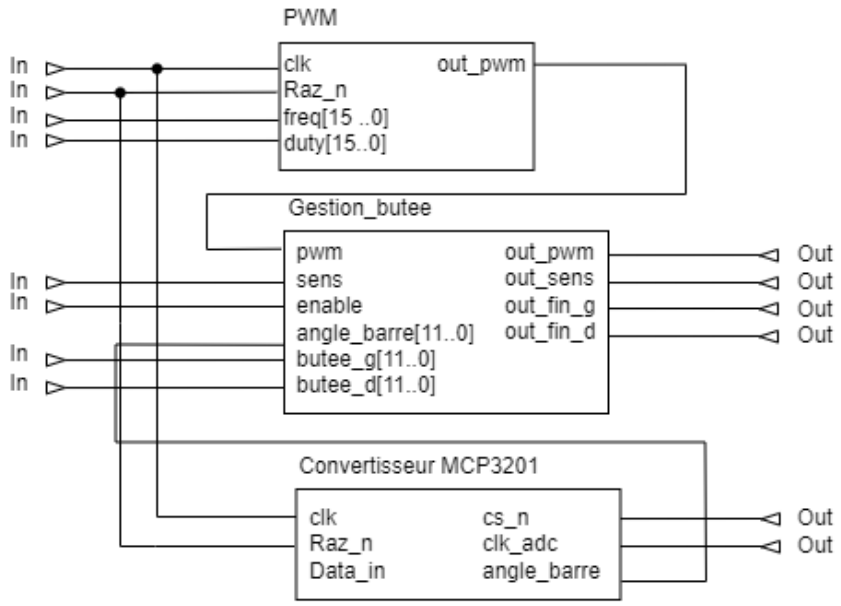

Figure 13 – Schéma bloc de la fonction gestion vérin

II.2 Bloc gestion du convertisseur MCP3201

Le programme vhdl MCP3201 est constitué de 3 sous blocs principaux qui sont :

- Un générateur de signal 1 MHZ
- Un registre à décalage
- Compteur de fronts montant pour le signal d'horloge clk_adc
- Générateur de signal start_conv chaque 100ms
- Une machine à états qui fonctionne comme montrée dans la figure ci-dessous :

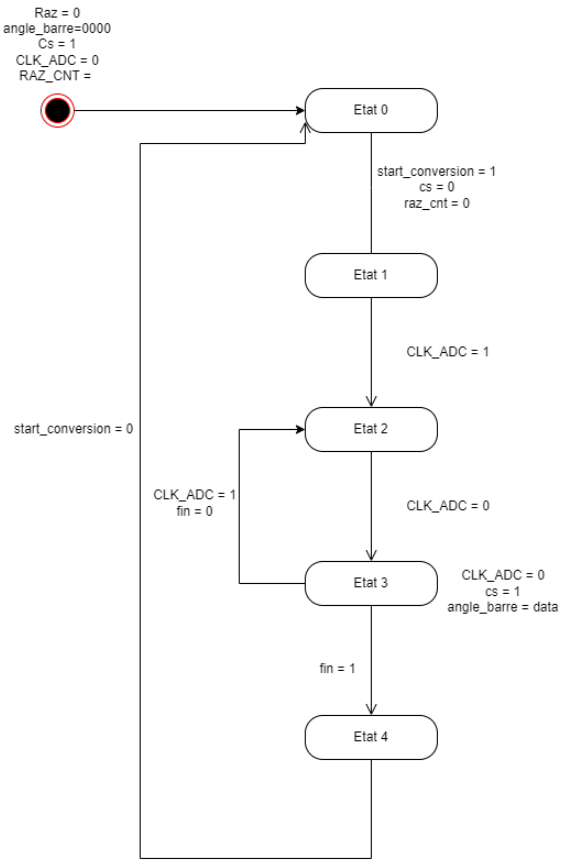

Figure 14 – Machine à états du convertisseur MCP3201

II.3 Développement de l'interface Avalon en vhdl

Dans cette partie, nous avons réalisé l'interface Avalon permettant de faire la connexion entre la partie que nous allons développer sous Platform Designer utilisant SOPC Builder, et la partie matérielle que nous avons déjà développée en vhdl et logicielle que nous allons développer dans ce qui suit.

Afin de piloter la barre franche ou récupérer les données, on devrait désormais passer par les registres du bus Avalon, soit en faisant des lectures ou écritures. En respectant les exigences données dans les documents qui nous ont été fournis pendant ce projet, nous avons rajouté au schéma bloc précédent du vérin développé en vhdl, l'interface Avalon comme le montre la figure ci-dessous :

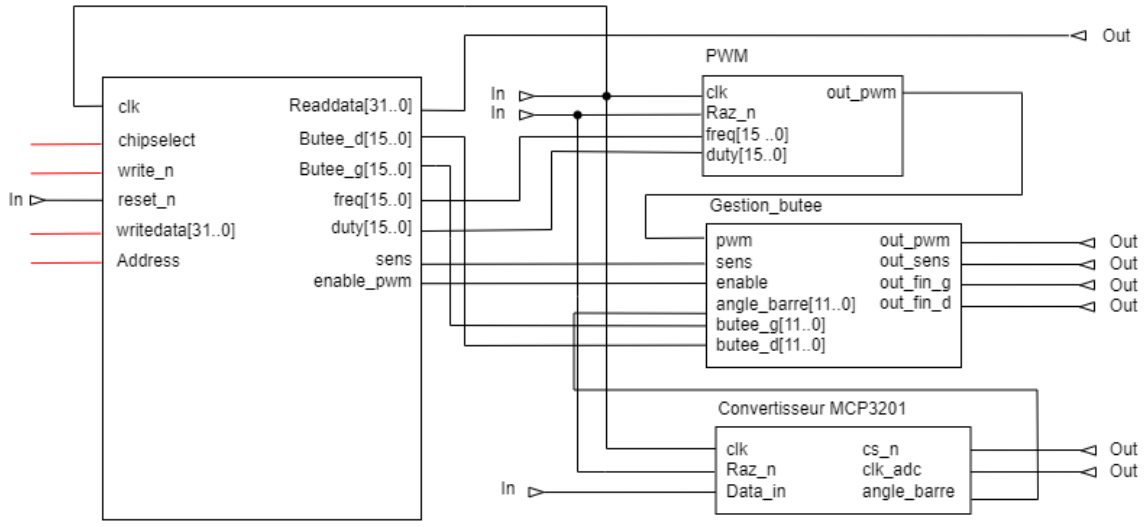

Figure 15 – Schéma bloc de l'interface Avalon / vérin

II.4 Implémentation du processeur Nios sur Platform designer

Dans cette partie, à l'aide de Platform Designer et SOPC Builder, nous avons généré le processeur NIOS II 32 bits ainsi que les différents éléments nécessaires à savoir la RAM, le JTAG UART, les PIO.

Nous avons également créé un nouveau composant que nous appelé F6_Avalon_Verin, dans lequel nous avons chargé notre fichier vhdl avalon.

La figure ci-dessous montre tous les composants ajoutés sur Platform Designer

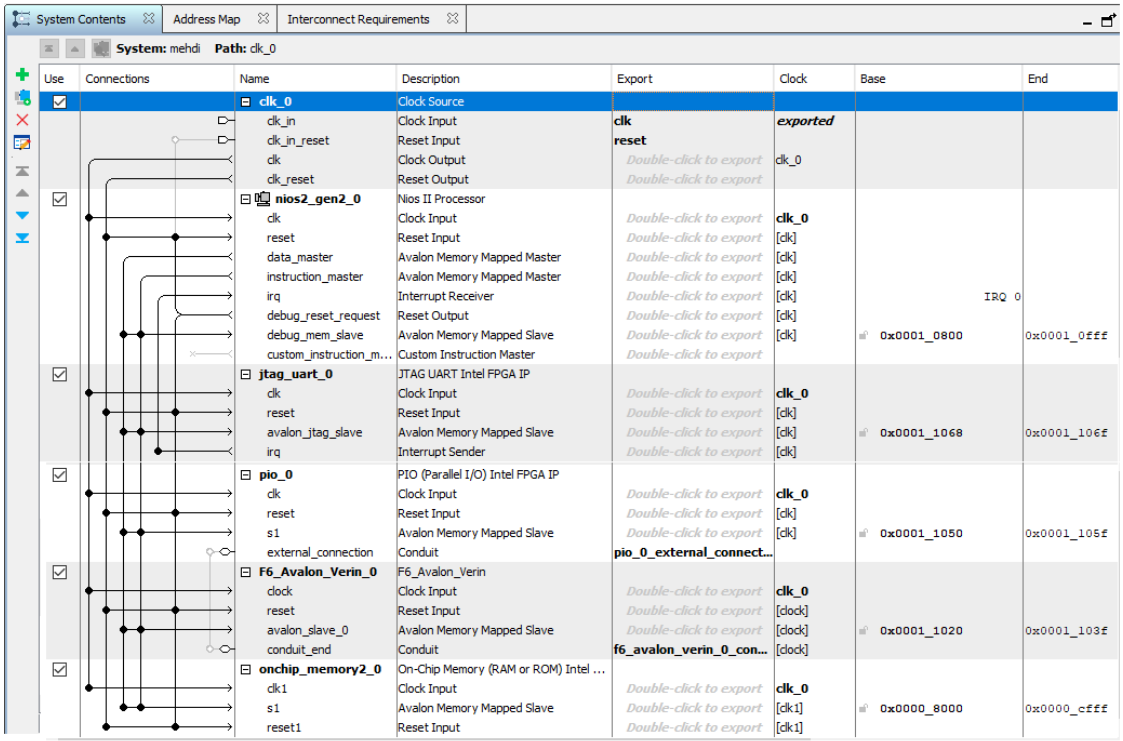

Figure 16 – Ajout du processeur NIOS et les autres composants sous Platform Designer

La figure ci-dessous montre le schéma bloc du composant créé F6_Avalon_Verin après avoir chargé notre fichier vhdl ainsi que les différents signaux mis en conduits (entrées/sorties) et signaux internes

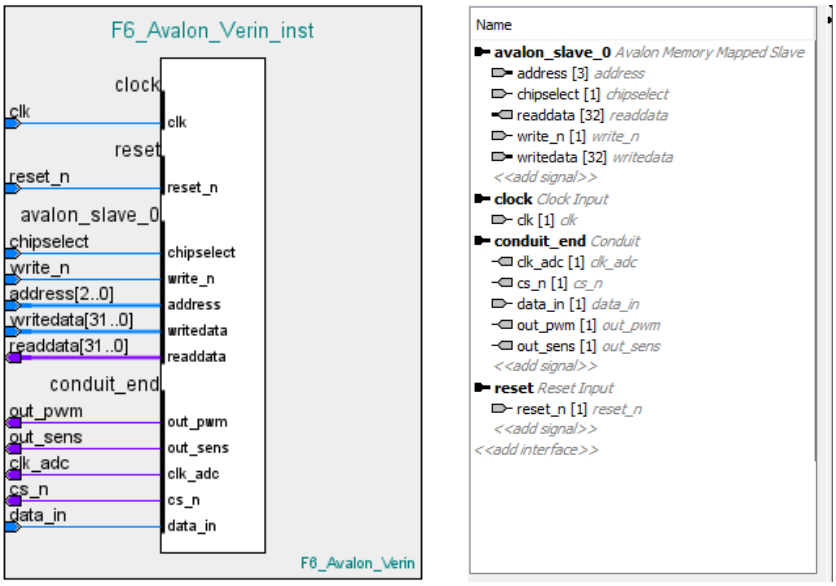

Figure 17 – Schéma bloc composant F6_Avalon_Verin

Une fois on finit notre configuration, on la sauvegarde sous extension .qsys et on génère le fichier HDL, des fichiers vont être créés par la suite qui vont nous servir par la suite à créer notre nouveau vhdl top, qui permet de faire la liaison entre tous les composants crées sous Platform designer et nos codes vhdl, pour pouvoir passer ensuite à la partie programmation en langage C sous Eclipse.

II.5 Partie programmation logicielle en C

Dans la partie C, on fait lire et écrire dans les registres que nous avons défini lors de la génération du bus Avalon, en se basant sur le tableau donné dans le document cahier des charges suivant :

Figure 17 – Schéma bloc composant F6_Avalon_Verin

Registre	adresse	type	Bits concernés
freq	0	R/W	b15..b0
duty	1 (4)	R/W	b15..b0
Butee_g	2 (8)	R/W	b15..b0
Butee_d	3 (12)	R/W	b15..b0
config	4 (16)	R/W R/W R/W R R	b0 :raz_n (à 0=reset circuit) b1 :enable_pwm (1= pwm actif) b2: sens rotation (0=gauche) b3:fin_course_d (=1 si fin course_d) b4: fin_course_g (=1 si fin course_g)
Angle_barre	5 (20)	R	b11..b0

Figure 18 – Affectation des registres du circuit

Nous avons compris sur la partie logicielle embarquée qu'il est nécessaire de définir les adresses de chaque fonction de tous les blocs.

Lors de la compilation du projet sur Eclipse, nous récupérons les adresses des composants afin de les utiliser sur le main.c comme vous pouvez le voir sur l'entête du fichier hello_world_small.c.

```
#define freq (unsigned int*)PWM_AVALON_0_BASE
#define duty (unsigned int*)(PWM_AVALON_0_BASE + 4)

#define entree_anemo_vent (unsigned int*)(AVALONANEMO_0_BASE + 4)
#define config (unsigned int*)AVALONANEMO_0_BASE

#define butee_droite (int *)(AVALONVERIN_0_BASE+12)
#define butee_gauche (int *)(AVALONVERIN_0_BASE+8)
#define freq_ver (int *)AVALONVERIN_0_BASE
#define duty_ver (int *)(AVALONVERIN_0_BASE+4)
#define config_ver (int *)(AVALONVERIN_0_BASE+16)
#define ang_bar (int *)(AVALONVERIN_0_BASE+20)

#define config_compas (unsigned int*) VHDL_COMPASS_0_BASE
#define code_compas (unsigned int*) (VHDL_COMPASS_0_BASE+4)
```

Figure 19 – Code en C définitions matérielles

II.6 Validation PWM – commande vérin partie VHDL

Lors des premières séances, nous avons travaillé sur la fonction PWM. Cette dernière a pour objectif d'être utilisée sur la partie « Gestion vérin ». Pour cela on se devait de comprendre le bon fonctionnement de cette fonction.

La PWM, ou la modulation de largeur d'impulsion en français, est une technique utilisée afin de synthétiser des signaux pseudo analogique à l'aide de circuits numériques.

Dans notre cas, on souhaite que le système génère un signal PWM qui sera interprété pas un composant qui pilotera le vérin ensuite.

On observe sur nos simulations sur ModelSim qu'en faisant varier notre entrée « DUTY », le rapport cyclique varie, comme attendu, en sortie.

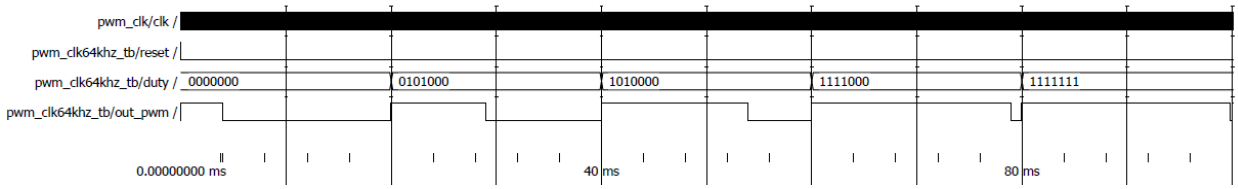

Figure 20 – Résultat simulation PWM sur ModelSim

Conclusion :

Finalement, ce travail fût très enrichissant surtout d'un point de vue organisation d'un projet, en passant par toutes les étapes de conception et de validation. En effet, ce dernier a été réalisé avec un FPGA ce qui nous a permis de découvrir de nouveaux outils qui permettent de valider le bon fonctionnement des fonctions logique en VHDL, qui ont été identifiées durant la phase d'analyse. Nous avons d'abord pu instancier notre travail sur des DE2 d'Altera qui nous a été fourni et qui est aussi la carte qui se trouve sur la maquette finale. Ensuite, nous avons travaillé sur l'interfaçage des composants à l'aide du bus du processeur (Bus Avalon). Et enfin, nous avons finit par le test et la vérification du système complet.

Ce projet nous a permis d'acquérir des compétences techniques notamment de mettant en œuvre différentes compétences acquises lors de notre formation. Nous avons aussi été confrontées à des difficultés techniques qui l'on peut rencontrer lors du cycle de vie d'un projet.

Réaliser un projet de ses premières étapes jusqu'aux étapes de test fût très enrichissant et nous procure une satisfaction particulière autant que futurs ingénieurs.