Université Paul Sabatier

Rapport VHDL

Barre franche

Auteurs :
Gautier Jobert
Alexandre regnere
Encadrant :
Thierry PERISSE

Table des matières

1 Introduction

A travers ce projet, nous allons apprendre a developper sur carte FPGA avec le langage VHDL au travers de l'asservissement d'une barre franche. Le developpement se fera coté hardware mais également en partie software avec le bus Avalon.
Le contexte de l'asservissement d'une barre franche nous permettra de traiter des données physiques (ex : force du vent) ainsi que la direction d'un cap.

Vous pourrez retrouvez notre projet avec le code et toutes les informations complémentaires au lien suivant :

`https://github.com/GautierDev31/M2SME_VHDL_GR4`

2 Présentation du projet

2.1 Cahier des charges

Le cahier des charge demandé nous avons plusieurs fonctions, avec plusieurs niveau de difficultés qui servirons a asservir un cap sur un bateau. En entrée on retrouvera plusieurs composents a prendre en compte comme la vitesse du vent, la position... mais aussi les commande choisis par l'utilisateur. En sortie nous aurons les indications prises par notre système et le déplacement de la barre en conséquence.

Le temps et les conditions de travail ne nous permettrons pas de traiter et d'intégrer toutes les fonctions. C'est pourquoi nous avons choisis deux fonctions qui vont nous permettre de mettre a profis les compétences a acquérir pour ce BE.

Parmi toutes les fonctions de cette barre franche, nous allons réaliser deux fonctions :
- Une fonction simple F1 : Conversion info vent ou Gestion anémomètre
- Une fonction complexe F7 : Gestion commandes et indications barreur

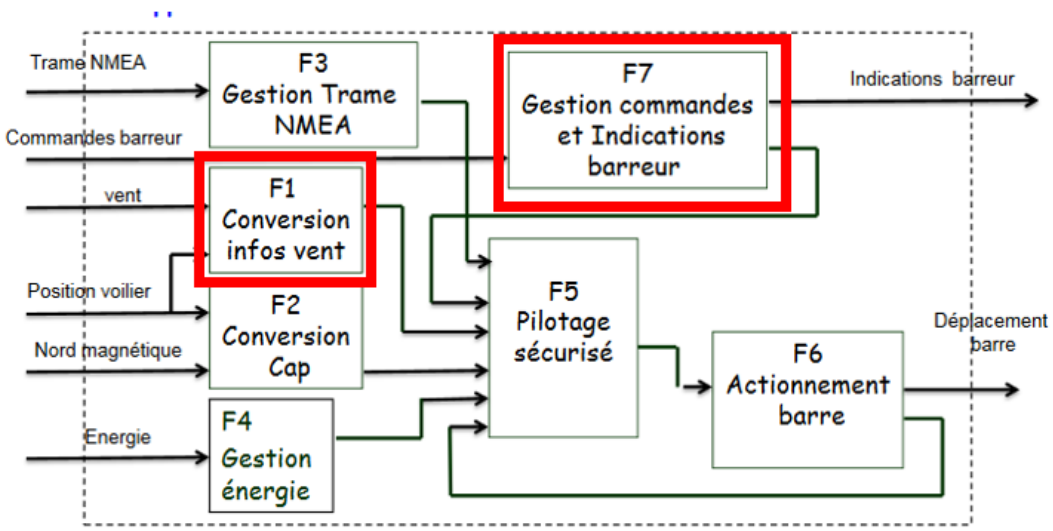
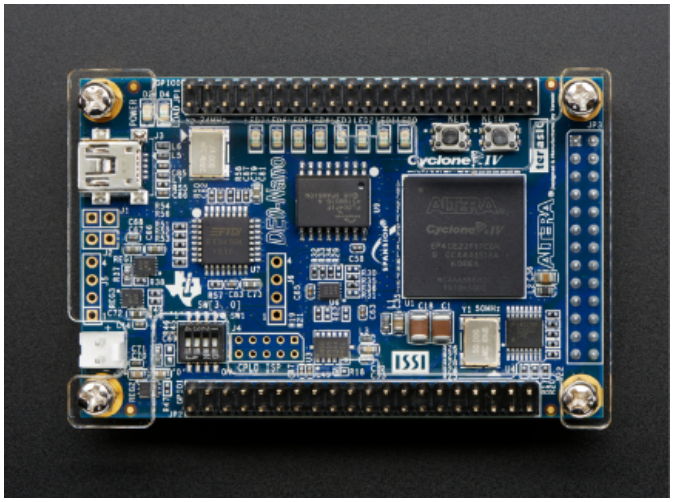


Schéma global avec les fonctions réalisées

2.2 Matériel et outils

Voici les outils que nous avons utilisés :
Carte FPGA DEO : Carte sur laquelle nous avons travaillé.
Quartus 9 et 11 : Developpement du code VHDL et assignation des pin pour le FPGA. Quastus 9 nous a permis dans un premier temps de simuler le code, nous avons ensuite utilisé Quartus 11 avec ModelSim pour les simulations.
Nios II et Eclipse : Developpement de la partie Software en language C qui utilise le bus avalon.



Carte Altera DE0

2.3 Partie software - Bus avalon

Le Bus Avalon est un bus qui a été developpé par Altera afin de permettre l'integration entre différentes parties. Il va permettre l'interconnexion entre le processeur NIOS II et les différents périphériques des composants correspondant a chaques fonctions.
Lors de ce BE, le bus Avalon va nous permettre principalement d'afficher sur la console via le bus série les informations lues par les fonctions comme la vitesse du vent pour la fonction anemometre et les condes des différentes actions pour la fonction commande barreur.

2.4 Organisation du BE

Pour travailler a 2 sur le même code, nous avons travaillé en utilisant l'outil Git en hebergeant nos fichiers sur GitHub.Cette plateforme nous permet d'heberger le code et de le rendre accessible en sauvegardant tout les changements effectués.
Pour avoir des hebergements de travail différent et ne pas introduire des bug lorsque l'on fais des test, nous avons créé différentes branches. Par exemple lorsque l'on travaillais sur le SOPC nous faisions un "Push" (envoie du code) vers la branche "SOPC" et lorsque nous travaillons sur la fonction F1 nous faisiont un Push vers la branche "Dev F1".
Lorsque le travaille est terminé et testé, nous avont fait un "merge" vers la branche principale, c'est a dire que nous avons assemblé les différentes branche entre elles. Le travail qui est accessible sur le GitHub est le travail sur la branche principale.

Pour le reste des communication, nous avons travaillé sur "Discord" entre nous et sur "Slack" pendant les heures de BE.

Pour pouvoir s'y retrouver dans les fichiers sur GitHub, Voici l'arborescence :
Barre franche : Contient les codes VHDL des composants et des fonctions
Barre franche > Components : Contient les différenst codes des components utilisés dans les fonctions
Barre franche > F1 et F7 : Contient le code des deux fonctions qui est un mappage entre les différents components
PWM : Code d'un component pour la fonction F1
SOPC : Code SOPC des fonctions F1 et F7
TP Base : Regroupe les codes pour les TB de bases
images : Images utilisés pour les descriptions
Rapport et présentation : Contient le rapport, la présentation et la video

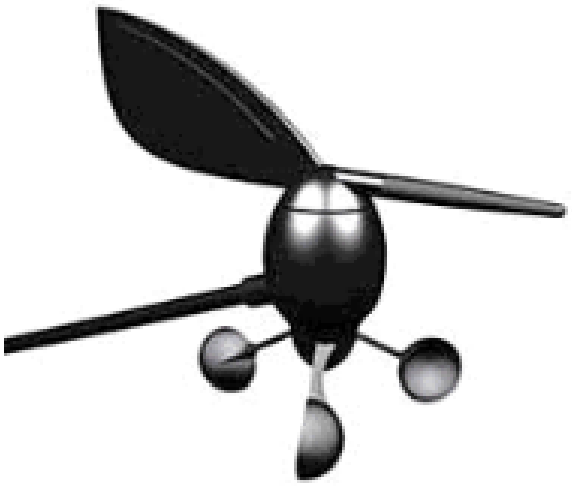
3 Réalisation

3.1 F1 - Gestion anémomètre

3.1.1 Cahier des charges

La fonction doit convertir la vitesse du vent en un signal de 8 bit ainsi qu'un signal indiquent que la valeur est disponible.
La fonction comportera 2 modes de lectures :
Mode continue :Lis la vitesse du vent en continue et l'affiche toute les secondes.
Mode Start and Stop : Commence a lire la vitesse du vent lorsque l'on appuis sur Start et affiche la vitesse du vent en moyenne en appuyant sur Stop.

```
--*********************************************
--entrées:
--clk_50M : hologe 50MHz
--raz_n: reset actif à 0 => initialise le circuit
--in_pwm_compas: signal PWM de la boussole, durée varie de 1ms à 36,9ms
--continu : si=0 mode monocoup, si=1 mode continu
-- en mode continu la donnée est rafraîchie toute les secondes
--start_stop: en monocoup si=1 démarre une acquisition, si =0
-- remet à 0 le signal data_valid
--*************************************************************
-- sorties:
-- data_valid: =1 lorsque une mesure est valide
-- est remis à 0 quand start_stop passe à 0
-- out_1s : signal de contrôle du top seconde (normalement pas utilisé)
-- data_compas : valeur du cap réel exprimé en degré codé sur 9 bits
--*********************************************************************
```



Anemometre

3.1.2 Schéma fonctionnel

Afin de décomposer la fonction en différentes parties, nous avons fait un schéma fonctionnel.

Diviseur de fréquence : L'horloge du FPGA est de 50 MHz, pour la ramener a une echelle de temps exploitable, nous avons fait deux diviseurs de fréquences, à 2Hz et 1Hz.

Compteur : Le compteur va permettre de savoir combien de temps l'entrée est a l'état haut. En sortie il indiquera ce temps sur 8 bits.

Raf anemo : Cette fonction permet de rafraichir la valeur de temps haut reçu toute les secondes.

Compteur Monocoup : Ce component permet d'activer le comptage de l'état haut lorsque le bouton start est activé.

Synchro anemo : Cette fonction permet de renvoyer l'information de la valeur de l'anemometre en prenant la bonne valeur entre le monocoup et le mode continu. Il renvois également le data valid pour savoir si la valeur est bien valide.

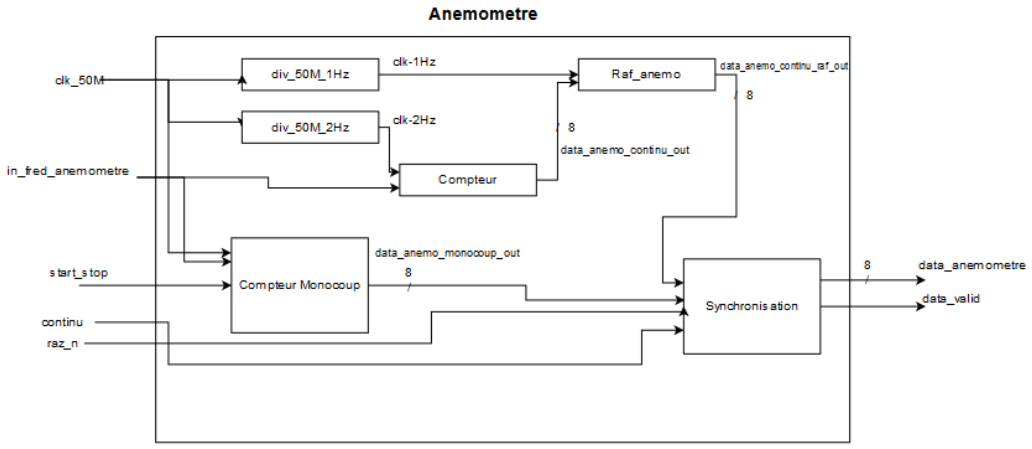
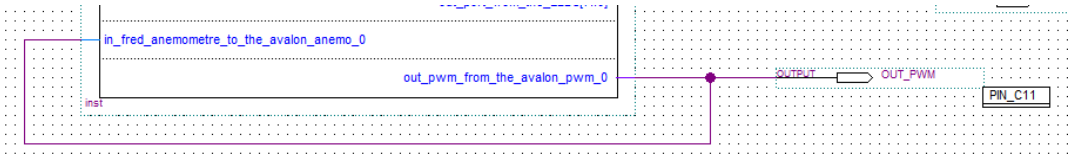


Schéma fonctionnel F1 : Gestion anémomètre

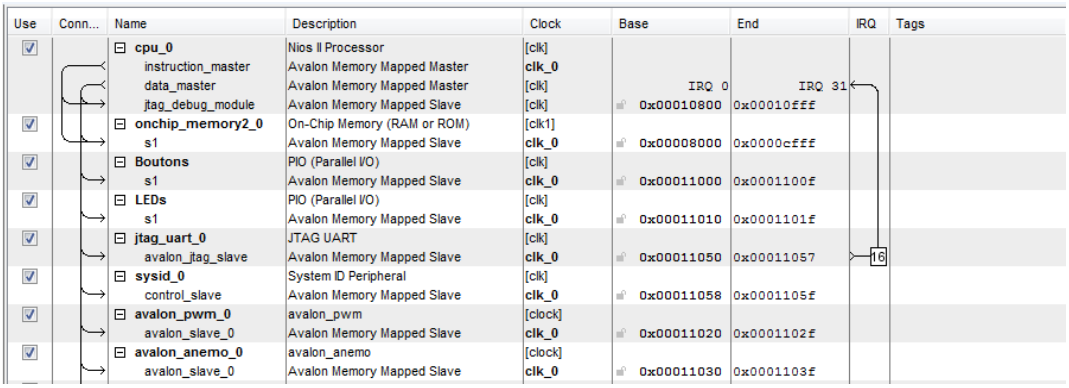
3.1.3 Partie software

Afin de pouvoir tester la partie software avec une seule platine, nous allons générer un signal PWM en sortie du FPGA que nous allons branché sur une entrée pour pouvoir la lire. Le câblage sera fait virtuellement, on peut voir sur le fichier .bdf ci-dessous la sortie reliré a l'entrée.



Anemometre BDF

Nous avons générer le système SOPC avec "SOPC Builder" qui comprend les composants de la fonctions ainsi les les composants physiques :
- Le processeur NIOS 32 Bits qui permet l'exécution des fonctions
- La mémoire RAM de 20Ko
- Le JTAG qui creer une interface entre le NIOS et le SOPC
- Le SYSIP qui attribut au système un numero d'identification



Use	Conn...	Name	Description	Clock	Base	End	IRQ	Tags
☑		cpu_0	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master	[clk]			IRQ 0	IRQ 31
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00010800	0x00010fff		
☑		onchip_memory2_0	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00008000	0x0000cfff		
☑		Boutons	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00011000	0x0001100f		
☑		LEDs	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00011010	0x0001101f		
☑		jtag_uart_0	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00011050	0x00011057	16	
☑		sysid_0	System ID Peripheral	[clk]				
		control_slave	Avalon Memory Mapped Slave	clk_0	0x00011058	0x0001105f		
☑		avalon_pwm_0	avalon_pwm	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011020	0x0001102f		
☑		avalon_anemo_0	avalon_anemo	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011030	0x0001103f		

SOPC Anemometre

Pour le code en C, nous avons définis dans la première partie la fréquence et le rapport cyclique qui allais être envoyé. Dans la seconde partie on affiche la valeur lue.

```
*freq = 0x2625A0 ; // CLK divisee par 2 500 000 => freq = 20 Hz
*duty = 0x1312D0 ; //10% de 2 500 000 => duty = 10%
*control = 0x3 ;

*config = 0x2 ;
```

Code C Anemometre : Declaration des variables

```
/********************** DATA ANEMOMETRE ****************************/
/*******************************************************************/
        *data = *data&0x3FF ;
         printf("reg = 0x%08X\n", *data) ;
         alt_printf("config = 0x%x\n", *config) ;
```

Code C Anemometre : Affichage de la valeur de l'anemometre

3.2 F7 - Gestion commandes et indications barreur

3.2.1 Cahier des charges

L'objectif de cette fonction est de prendre en entrée les commandes au travers de boutons poussoirs par l'utilisateur pour commander le barreur.
En mode Manuel : Les boutons bâbord et tribord incrémente le cap de 1° pour un appuis court et 10° pour un appuis long.
En mode automatique : Le bateau devra conserver son cap.

Note : Nous avons adapté notre cahier des charges a celui existant pour des raison de faisabilité.

Voici les entrées/sorties :

```
--**************************************************************************
-- entrées: BP_Babord,BP_Tribord, BP_STBY, clk_50M, raz_n
-- sorties: codeFonction, ledBabord, ledTribord,ledSTBY, out_bip
--**************************************************************************

--clk_50M: horloge à 50MHz
-- raz_n: actif à 0 => initialise le circuit
-- valeurs de codeFonction:
-- =0000: pas d'action, le pilote est en veille
-- =0001: mode manuel action vérin babord
-- =0010: mode manuel action vérin tribord
-- =0011: mode pilote automatique/cap
-- =0100: incrément de 1° consigne de cap
-- =0101: incrément de 10° consigne de cap
-- =0111: décrément de 1° consigne de cap
-- =0110: décrément de 10° consigne de cap
--**************************************************************************
```

Commandes barreur

3.2.2 Schéma fonctionnel

Nous avons de nouveau fait un schéma fonctionnel pour séparer les différentes parties a développe.

Diviseur de fréquence : Nous avons réutilise le diviseur de fréquence 1Hz. Le compteur comptera donc en seconde.

Compteur BP : Cette fonction permet de savoir combien de temps un bouton a été appuyé.

Synchronisation : Cette fonction est une machine a état qui fait passer d'un état a un autre en fonction du bouton en entrée et de sa longueur d'appuis.

Note : En utilisant les fonctions "Compteur BP" et "Synchronisation" nous avons rencontré quelques soucis de fonctionnement. C'est pourquoi nous avons fait une autre fonction "Fonction F7" qui est simplement une machine a état qui change d'état toute les secondes. Cette fonction marche parfaitement et c'est grâce a elle que nous avons validé la fonction. Nous avons cependant voulu laissé les différents codes (sur GitHub) et la réflexion que nous avons eu.

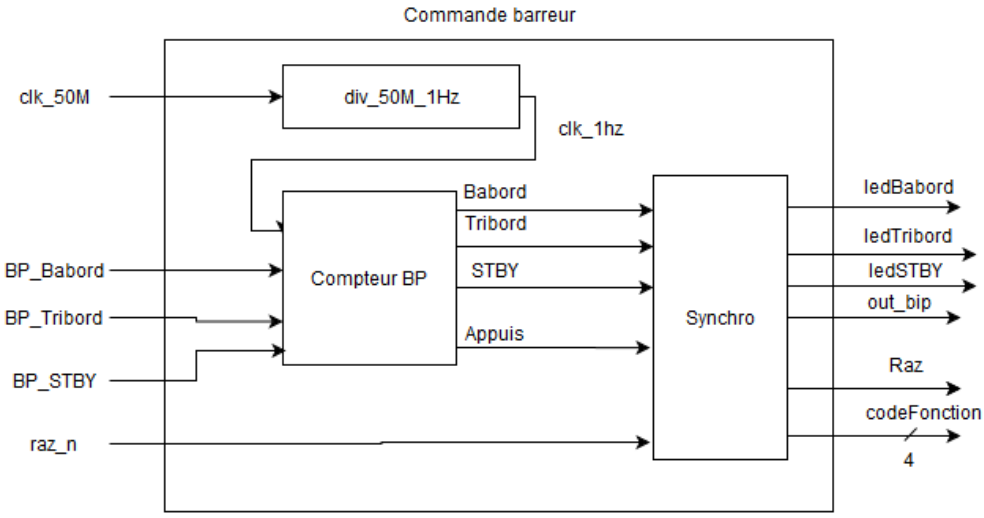


Schéma fonctionnel F7 : Gestion commandes et indications barreur

3.2.3 Platine de tests

La situation actuelle ne nous as pas permis d'avoir accès a la maquette de TP pour tester notre fonction. C'est pourquoi nous avons réalisé notre propre plaquette de test regroupant les différents entrées sorties necessaires a savoir :

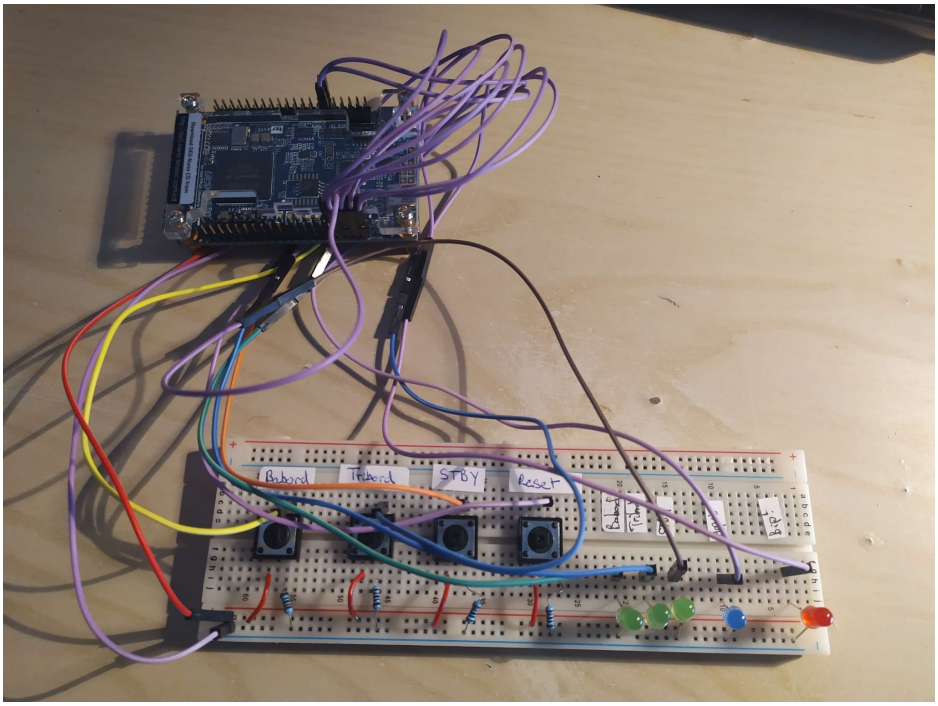
Entrées :
- Bouton poussoir Babors
- Bouton poussoir Tribord
- Bouton poussoir STBY
- Bouton poussoir reset

Sorties :
- Led Babord
- Led Tribord
- Led STBY
- Led représentant le BIP
- Led Appuis (Cette Led qui n'est pas prévus dans le cahier des charges a été ajouté dans un premier temps pour vérifier la détection d'un appuis long sur un bouton)

La plaquette a parfaitement marché et a été utilisé lors du test de la fonction lors de la validation.
Ci dessous vous trouverez les différents pin de sorties utilisés ainsi que la plaquette.

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strength	Slew Rate	Differential Pair
clk_50M	Unknown	PIN_R8	3	B3_N0	2.5 V (default)		8mA (default)		
code_fonction[3]	Unknown	PIN_A11	7	B7_N0	2.5 V (default)		8mA (default)		
BP_Barbord	Unknown	PIN_A8	8	B8_N0	2.5 V (default)		8mA (default)		
BP_STBY	Unknown	PIN_D3	8	B8_N0	2.5 V (default)		8mA (default)		
BP_Tribord	Unknown	PIN_B8	8	B8_N0	2.5 V (default)		8mA (default)		
out_bip	Unknown	PIN_C3	8	B8_N0	2.5 V (default)		8mA (default)		
code_fonction[0]	Unknown	PIN_A15	7	B7_N0	2.5 V (default)		8mA (default)		
code_fonction[1]	Unknown	PIN_A13	7	B7_N0	2.5 V (default)		8mA (default)		
code_fonction[2]	Unknown	PIN_B13	7	B7_N0	2.5 V (default)		8mA (default)		
led_Babord	Unknown	PIN_A2	8	B8_N0	2.5 V (default)		8mA (default)		
led_Tribord	Unknown	PIN_A3	8	B8_N0	2.5 V (default)		8mA (default)		
led_STBY	Unknown	PIN_B3	8	B8_N0	2.5 V (default)		8mA (default)		
led_appuis	Unknown	PIN_B4	8	B8_N0	2.5 V (default)		8mA (default)		
reset	Unknown	PIN_A4	8	B8_N0	2.5 V (default)		8mA (default)		

Repartition des PINs

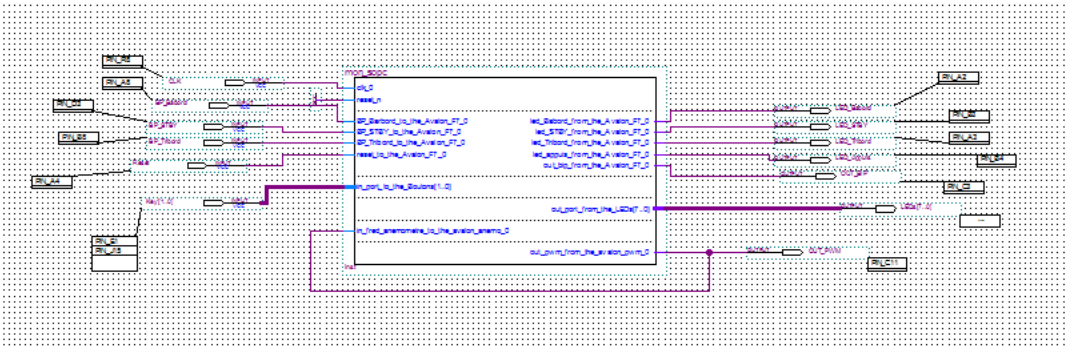


Platine de test

3.2.4 Partie software

Nous avons implémenté la partie software dans celle déja réalisé pour l'anémometre.
Pour le code VHDL, nous faisons passer dans le bus l'information du code de commande.
Nous avons rajouté un composant comprenant les fonctions du commande barreur sur le SOPC.

Vous retrouverez ci-dessous le fichier BDF global ainsi que la composition du SOPC.



Commandes barreur BDF

Use	Conn...	Name	Description	Clock	Base	End	IRQ	Tags
☑		⊟ **cpu_0**	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	**clk_0**				
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31		
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00010800	0x00010fff		
☑		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	**clk_0**	0x00008000	0x0000cfff		
☑		⊟ **Boutons**	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	**clk_0**	0x00011000	0x0001100f		
☑		⊟ **LEDs**	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	**clk_0**	0x00011010	0x0001101f		
☑		⊟ **jtag_uart_0**	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_0**	0x00011050	0x00011057	16	
☑		⊟ **sysid_0**	System ID Peripheral	[clk]				
		control_slave	Avalon Memory Mapped Slave	**clk_0**	0x00011058	0x0001105f		
☑		⊟ **avalon_pwm_0**	avalon_pwm	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	**clk_0**	0x00011020	0x0001102f		
☑		⊟ **avalon_anemo_0**	avalon_anemo	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	**clk_0**	0x00011030	0x0001103f		
☑		⊟ **Avalon_F7_0**	Avalon_F7	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	**clk_0**	0x00011040	0x0001104f		

SOPC Commandes

4 Conclusion

Pour conclure, on peut dire que grace a ce projet nous avons acquis un large spectre de compétences de bases pour le developpement en VHDL.
Le language VHDL étant un language de description machine, il ne se code pas de la même façon que les autres languages appris durant notre formation. Ce projet nous aura donc permis d'avoir de l'expérience dans le code de language de description.
Nous avons aussi développer nos compétence en conception en réfléchissants aux différents composants pour réaliser une fonction avec des schémas fonctionnels. Ces composants auront été cablé par la suite en faisant un MAP en VHDL.
Grâce au bus Avalon développé par Altera nous avons appris à faire une partie software en faisant communiquer notre PC avec le FPGA.
Enfin, une compétence importante que nous avons expérimenté avec ce projet à été de gerer notre projet avec Git, ce qui deviens un compétence indispensable aujourd'hui pour le travail en entreprise.