COMPTE-RENDU DU BE VHDL

BE Électronique numérique: VHDL

José CARTAGENA, Imane HAMIDA

Encadrant: T. Périssé

Table des matières

I. Préambule

Dans le cadre de l'UE "Synthèse et mise en œuvre des systèmes", nous avons été amenés à réaliser plusieurs séances de travaux pratiques portant sur les notions de la programmation en VHDL pour réaliser le projet: pilote de barre franche. Il existe plusieurs différentes fonctions à réaliser pour assurer le fonctionnement intégral de ce projet. Dû à la crise sanitaire du COVID-19 les séances de ce BE se sont déroulées pour la plupart à distance. Dans ce rapport nous allons donc aborder tout d'abord les programmes réalisés lors des séances introductives consacrées au "mini-projet". Ensuite nous allons diviser le contenu principal de ce compte-rendu en deux axes: premièrement les programmes qui correspondent aux différentes fonctions et deuxièmement l'implémentation de ces fonctions sous SOPC.

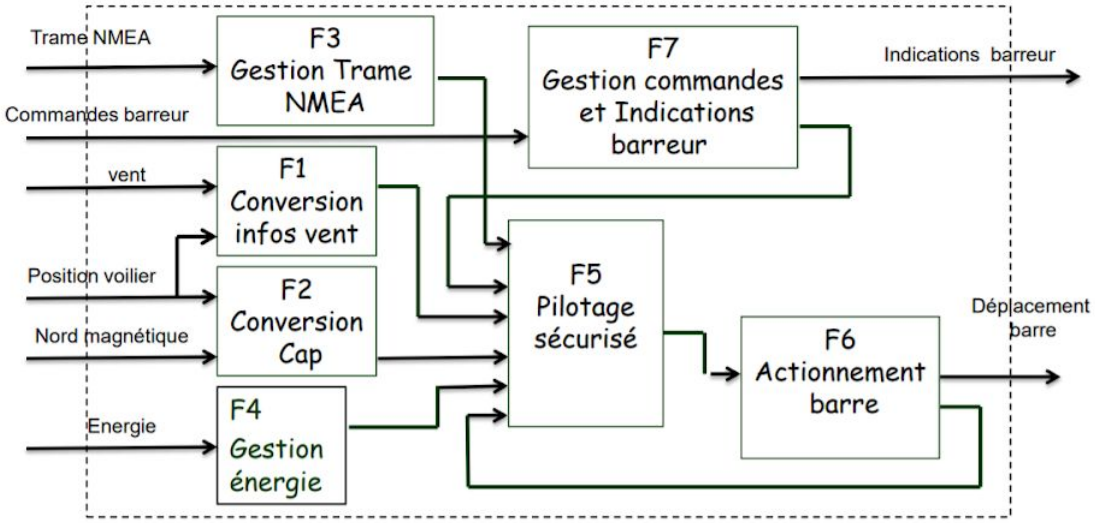
II. Fonction implémentées

Pour les premières manipulations nous nous sommes familiarisés avec les différents logiciels utilisés pendant ce bureau d'études ainsi qu'avec le langage de programmation VHDL qui sert à représenter le comportement d'un système électronique numérique.

Tout au long de ce BE nous avons travaillé sous le logiciel de conception de dispositifs logiques programmables Quartus (versions 9.1 et 11.1). Avec ce logiciel l'utilisateur est capable de programmer, simuler et tester des dispositifs logiques. Pour la prise en main de ce nouvel environnement nous avons effectué les fonctions suivantes:

- Fonction "Porte ET"
- Fonction "Compteur"
- Fonction "Diviseur de fréquence"
- Fonction "PWM"

Dans l'intérêt de la longueur de ce rapport nous allons pas insister sur les trois premières fonctions de ce "mini-projet" mais plutôt sur la quatrième qui correspond à la création d'une impulsion de fréquence PWM car elle sera utilisée pour modéliser le vent dans notre système. En effet, l'architecture du système en entier est représentée par le schéma suivant:



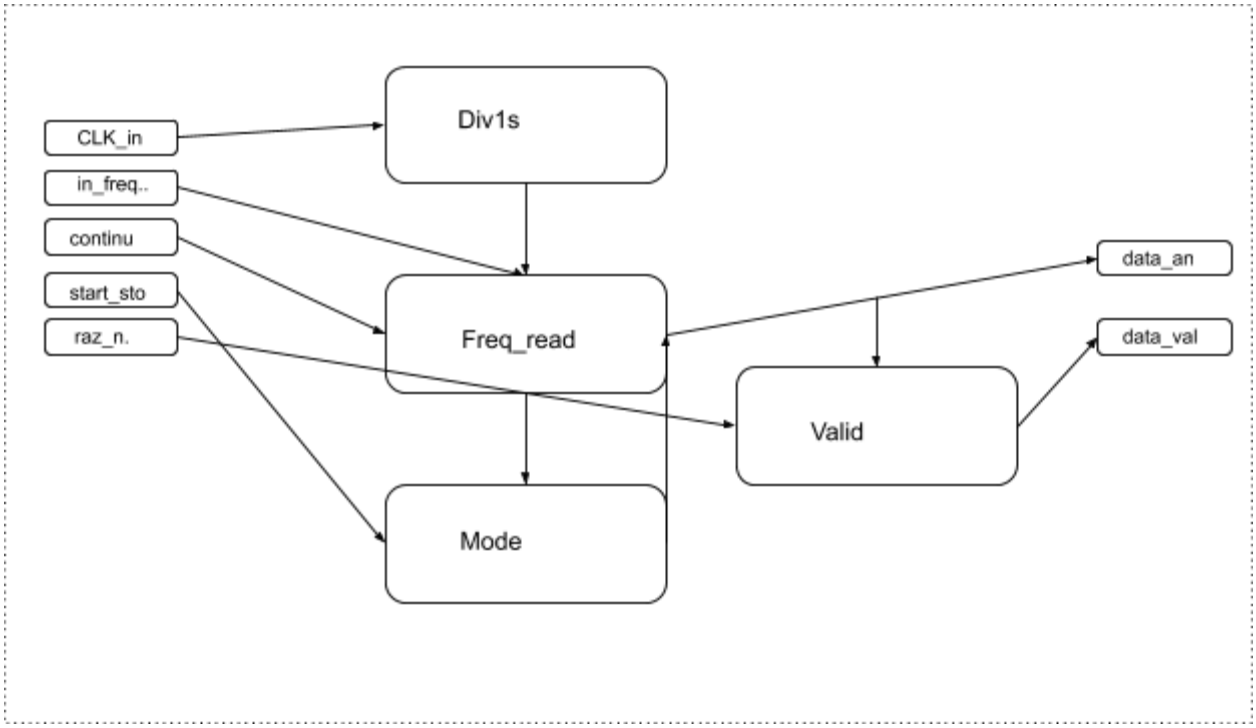
En effet, comme dit auparavant le signal PWM correspond au signal en entrée qui sera traité par notre système. L'extrait de code suivant décrit les entrées et sorties du PWM, ceci étant important parce que ce sera aussi le signal en entrée des deux fonctions simples que nous avons implémentées: F1 Conversion Infos Vent et F2 Conversion Cap.

```
ENTITY  pmw1 IS
generic(N : integer := 8
);
PORT(
clk                         :            in     std_logic;
--reset                     :            in     std_logic;
PWMOut                      :            out     std_logic;
rapport_Cyclique_SWs              :            in     std_logic_vector(3 downto 0)
);
END pmw1;

ARCHITECTURE pwm of pmw1 IS

signal counter_PWM: std_logic_vector(N-1 downto 0):=(others => '0');
signal rapport_Cyclique: std_logic_vector(N-1 downto 0):=x"00";
```

1. **F1 Conversion Infos Vent**

Dans l'arborescence de notre espace de travail Github cette fonction est appelée "Gestion Anémomètre" parce qu'elle permet de mesurer la vitesse du vent où bien la traduire en vitesse selon le signal qu'il reçoit en entrée.

D'après le cahier des charges, l'anémomètre émet un signal logique de fréquence variable entre 0 et 250 Hz. Ce dernier possède 2 modes de fonctionnement : monocoup et continu. Voici une décomposition fonctionnelle de la fonction anémomètre:



D'après le code qui correspond à cette fonction qui se trouve également dans notre répertoire de travail les signaux d'entrée et sortie sont les suivantes:

• Entrées

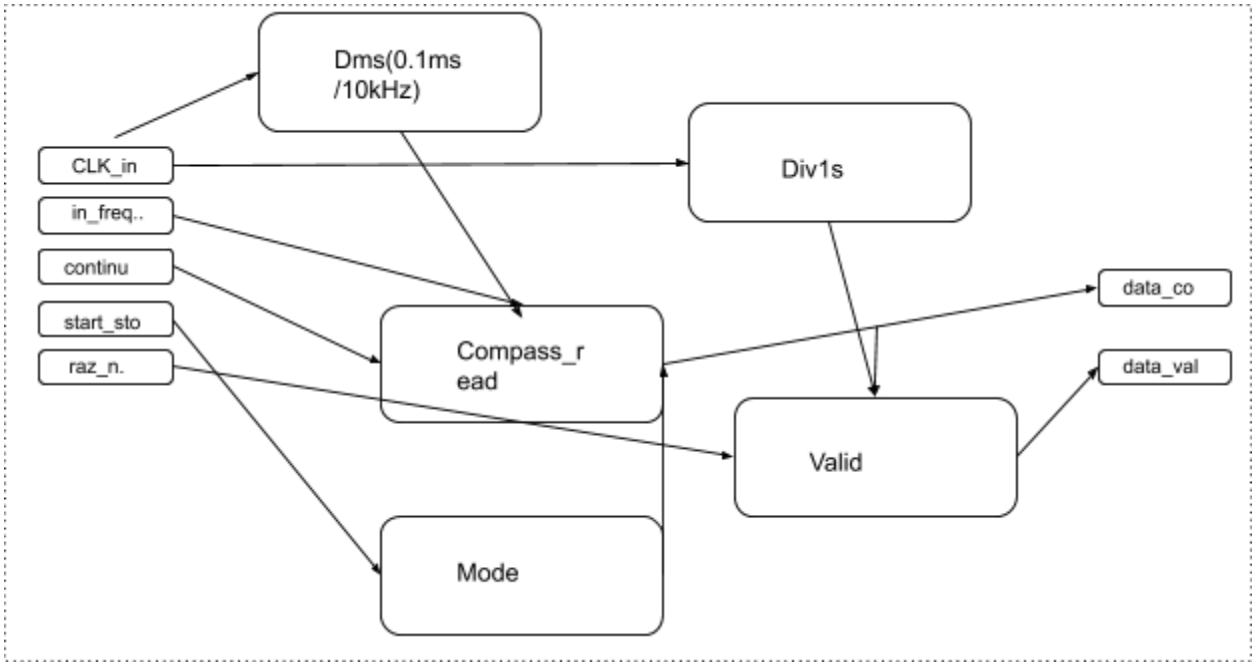
- CLK_in : Horloge 50 MHz.
- raz_n : initialise le circuit.
- in_freq anémomètre : signal envoyé par l'anémomètre à mesurer - Continu : à 1, acquisition de la vitesse toutes les secondes.
- start stop : démarre acquisition en mode monocoup.

• Sorties

- data valid : permet de vérifier la validité de la mesure.
- data anémomètre : fréquence mesurée codée sur 8 bits.

2. F2 Conversion cap

Dans l'arborescence de notre espace de travail Github cette fonction est appelée "Compas" parce qu'elle permet de simuler le fonctionnement d'une boussole et d'avoir des informations sur la position du bateau. Cette fonction traite le signal PWM que nous avons défini auparavant et renvoie un nouveau signal que nous devons ensuite transformer en angle pour avoir une donnée facile à lire pour l'utilisateur du dispositif. D'après le cahier des charges, le signal émis doit être compris entre 1 ms et 36,9 ms. Voici une décomposition fonctionnelle:



Comme pour la fonction antérieure nous allons expliciter les signaux d'entrée et sortie:

Entrées:

- CLK_in: Horloge 50MHz.
- raz_n: initialise le circuit.
- in_freq_compass: signal PWM envoyé par la boussole.
- Continu: à 1, mesure réalisée toutes les secondes.
- start_stop: démarre une acquisition en mode monocoup.

Sorties:

- data_valid: permet de vérifier la validité de la mesure.
- data_compass: valeur du cap exprimé en degré codé sur 9 bits.

3. F7 Gestions commandes et Indications barreur

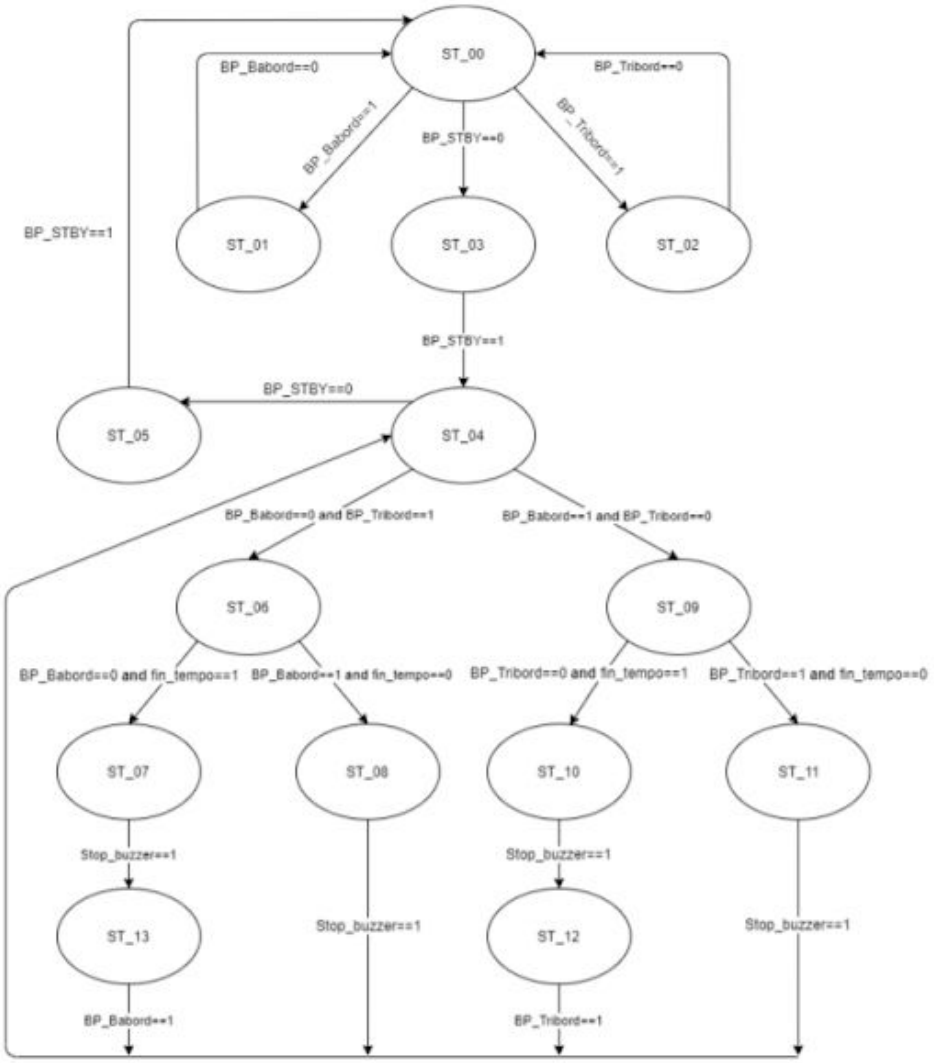
Dernièrement nous allons aborder la troisième étape de cette manipulation que nous avons choisie comme fonction complexe. La fonction à pour rôle de gérer les actions liées aux boutons et aux leds du dispositif. Cette fonction opère en utilisant trois LEDs, trois boutons poussoirs. Le fonctionnement est très intuitif: deux boutons qui sont censés gérer l'addition/soustraction à la valeur de l'angle de position, un bouton pour passer du mode Manuel au mode Automatique et vice-versa. Les LEDs servent à indiquer le succès de chaque appui du bouton, dans le vrai dispositif il existe des confirmations audio-visuels; ici nous en avons qu'une confirmation visuel. Le fonctionnement est explicité dans le manuel SIMRAD fournit par l'encadrant de ce bureau d'études lors des premières séances et décrit un cahier de charges comme celui-ci:

Mode Manuel : Le vérin peut être contrôlé manuellement à l'aide des boutons Babord et Tribord. Ce mode est signalé par le clignotement de la led STBY à une fréquence de 1 Hz. Ce mode permet aussi de régler le cap avant de passer en mode Automatique grâce à un appui sur le bouton correspondant.

- Mode Automatique : Ce mode a pour but de garder le cap réglé par l'utilisateur. Il est signalé par la led STBY qui reste allumée. Le cap reste cependant réglable en mode Automatique. Un appui court sur les boutons Babord ou Tribord modifiera le cap de 1° dans la direction choisie. Un appui long modifiera le cap de 10°.

Afin de mieux comprendre ce qui est décrit nous avons effectué une machine à états en reprenant le code de monsieur Jean-Louis Boizard:

(http://jeanlouis.boizard.free.fr/m2_sme/divers/vhdl/avalon_gestion_bp.vhd).



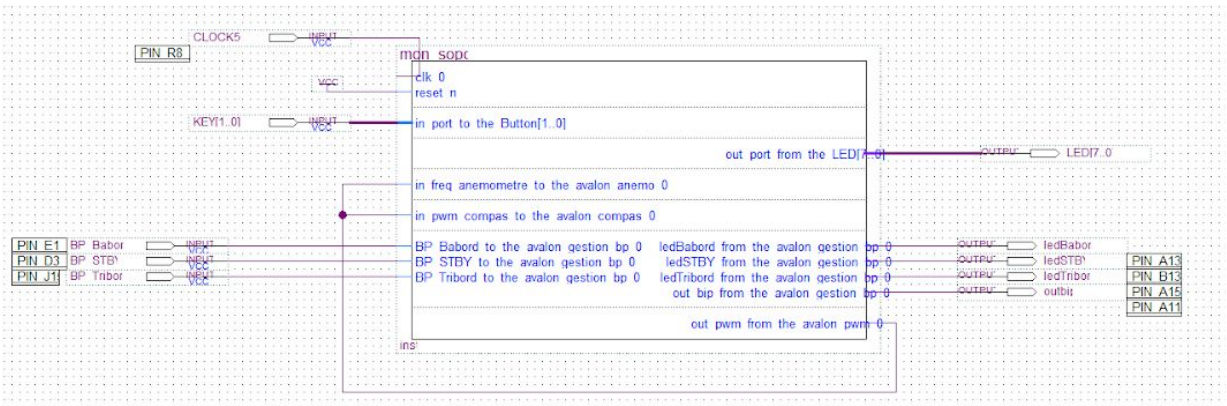
En conclusion de cette sous-partie, nous avons programmé toutes ces fonctions de façon indépendante et nous avons été capable de les tester sur la carte DE0 NANO (ou bien sur les cartes mieux équipées des salles U-305 lors des séances présentielles en bureau d'études). Ceci en faisant une affectation des pins avec le pin-planner du logiciel, pour exemplifier ceci voici le pin-planner du projet qui nous permet de tester la "gestion de boutons poussoirs".

Node Name	Direction	Location	I/O Bank	VREF Group	I/O Standard	Reserved	Current Strengtl	Differen
BP_Babord	Input	PIN_E1	1	B1_N0	2.5 V (default)		8mA (default)	
BP_STBY	Input	PIN_D3	8	B8_N0	2.5 V (default)		8mA (default)	
BP_Tribord	Input	PIN_J15	5	B5_N0	2.5 V (default)		8mA (default)	
CLOCK50	Input	PIN_R8	3	B3_N0	2.5 V (default)		8mA (default)	
KEY[1]	Input				2.5 V (default)		8mA (default)	
KEY[0]	Input				2.5 V (default)		8mA (default)	
LED[7]	Output				2.5 V (default)		8mA (default)	
LED[6]	Output				2.5 V (default)		8mA (default)	
LED[5]	Output				2.5 V (default)		8mA (default)	
LED[4]	Output				2.5 V (default)		8mA (default)	
LED[3]	Output				2.5 V (default)		8mA (default)	
LED[2]	Output				2.5 V (default)		8mA (default)	
LED[1]	Output				2.5 V (default)		8mA (default)	
LED[0]	Output				2.5 V (default)		8mA (default)	
ledBabord	Output	PIN_A13	7	B7_N0	2.5 V (default)		8mA (default)	
ledSTBY	Output	PIN_B13	7	B7_N0	2.5 V (default)		8mA (default)	
ledTribord	Output	PIN_A15	7	B7_N0	2.5 V (default)		8mA (default)	
outbip	Output	PIN_A11	7	B7_N0	2.5 V (default)		8mA (default)	

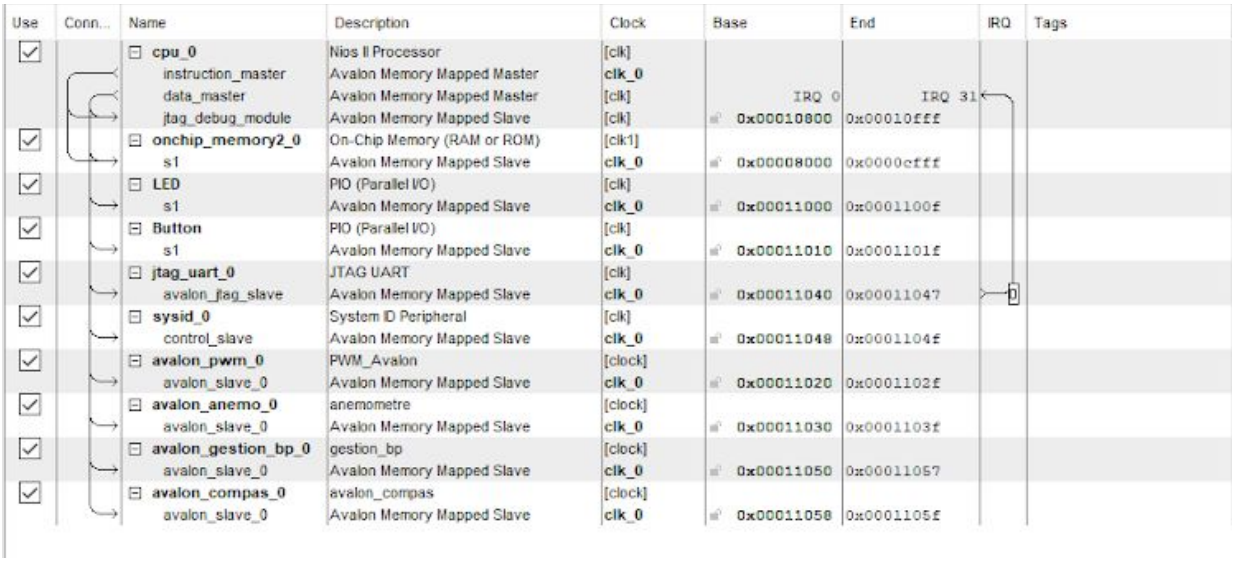
Comme nous pouvons le voir nous avons définit les LEDs en sorties, les boutons en entrées et les différents signaux comme l'horloge. Notre carte DE0 n'a que deux boutons, nous avons dû ajouter un bouton extérieur à la carte avec des composants complémentaires.

III. Construction du SOPC

Dans cette deuxième partie nous allons utiliser l'outil SOPC Builder de notre logiciel. En effet, on s'intéresse à interface toutes nos fonctions ensemble autour d'un processeur programmable en C. Cet outil est disponible dans la version 11.1 du logiciel Quartus, alors que les simulations des fonctions se font grâce à la version 9.1 de ce même logiciel. La prise en main de cet outil se fait tout d'abord en faisant un SOPC de notre signal PWM que nous avons évoqué au début de ce compte-rendu, l'interface que nous avons créée sera celle qui sera utilisé tout au long de cette sous-partie. Il s'agit donc non pas de créer des SOPC différents pour chaque fonction du projet mais de toutes les ajouter et de pouvoir permettre leur fonctionnement de façon harmonieuse. Voici à quoi ressemble le SOPC complet de toutes nos fonctions:



SOPC Builder permet donc aussi d'affecter les différentes adresses de chaque composant que nous avons importé et nommé, ceci évite ainsi les interférences et permet le bon fonctionnement du système:



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		
☑		LED	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00011000	0x0001100f		
☑		Button	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	0x00011040	0x00011047	0	
☑		sysid_0	System ID Peripheral	[clk]				
		control_slave	Avalon Memory Mapped Slave	clk_0	0x00011048	0x0001104f		
☑		avalon_pwm_0	PWM_Avalon	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011020	0x0001102f		
☑		avalon_anemo_0	anemometre	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011030	0x0001103f		
☑		avalon_gestion_bp_0	gestion_bp	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011050	0x00011057		
☑		avalon_compas_0	avalon_compas	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011058	0x0001105f		

La prochaine étape est de téléverser notre SOPC et exploiter ses composants sous l'environnement NIOS II. Cet environnement marche en langage C et utilise des librairies avec lesquelles nous pouvons définir et affecter des adresses à des différents composants pour finalement coder des procédures en C.

Finalement nous pouvons lancer le programme et observer les résultats sur la carte ainsi que sur le terminal avec des fonctions qui nous permettent de visualiser des informations ce qui a été fait pendant la séance de validation présentiel du jeudi 17/12/2020 avant de rendre notre carte DE0 NANO au laboratoire.

IV. Conclusion

Ces séances de bureau d'études ont été pour nous une façon très pédagogique de découvrir le monde des FPGA et le langage de programmation VHDL. L'opportunité d'avoir pu travailler sur un tel projet nous a permis de voir de façon concrète certaines notions que nous avons appris tout au long de notre formation universitaire. Malheureusement ce binôme n'a pas pu effectuer toutes les fonctions de ce BE à cause des conditions de la crise sanitaire du COVID-19 qui contraigne les manipulations avec une carte moins équipée et sans avoir accès à l'encadrement présentiel d'un professeur. Malgré cela nous tenons à remercier monsieur Thierry Périssé qui a mis en place des outils tels que Github et Slack pour garantir aux étudiants une façon de pouvoir travailler à distance.