Département EEA - Faculté Sciences et Ingénierie

Master 2 EEA SME, Systèmes et Microsystèmes Embarqués

RAPPORT DE BE VHDL

Barre franche

Par :
Kossi Pascal SEWODA, Kahina ACHARI
Université Toulouse III Paul Sabatier
Encadre par : Thierry PERISSE et Pedro CARVALHO-MENDES

Date : 29 novembre 2021

Année universitaire : 2021-2022

Table des matières

Introduction

Les circuits intégrés ont une place prépondérante aujourd'hui dans plusieurs applications surtout avec l'explosion des objets connectés. La fiabilité des CI est donc très importante. La conception d'un circuit intégré numérique commence par une modélisation haut niveau suivi d'une description RTL (Register Transfer Level) en utilisant des langages de description comme le VHDL ou le verilog. Ensuite il faut réaliser la simulation temporelle et fonctionnelle, réaliser la synthèse logique, placement routage (en testant sur FPGA) et enfin le lancement de la fabrication. Notre BE consiste à réaliser toutes les étapes précédentes, sans aborder la dernière partie. Il s'agit de concevoir un pilote de barre franche.
Dans un premier temps on va réaliser un des TP de base et des mini projet afin de prendre en main l'environnement de travail qui est Quartus 18.1, et ainsi le validé sur des FPGA (carte DE0 et DE2 nano) d'Intel. Ensuite on va réaliser la conception de deux fonctions de la barre franche (l'anémomètre et gestion des boutons) et enfin on va fait l'intégration dans un SOPC en utilisant l'architecture NIOS II.

I. TP de base

Dans premier projet il s'agit de prendre en main l'environnement de développement Quartus qui sera utilisé tout au long de ce TP d'électronique numérique avec du VHDL.
Le VHDL est un langage de description matériel qui permet défaire de la synthèse logique sur FPGA.
La création de la porte ET, comme n'importe quelle composant en VHDL, est décomposé en deux parties : l'entité et l'architecture.
L'entité : l'entité représente la vue boîte noire du composant ou du système avec ses ports d'entrées sorties.
L'architecture : Représente la façon dont le composant ou le système est implémenté. Un système peut avoir plusieurs architectures.

1. Porte ET

Ci-dessous, l'entité représentant la porte ET.

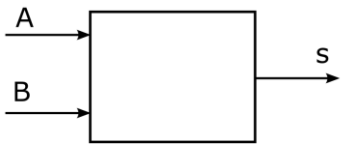

Figure 1: Entité de la porte ET

Pour réaliser cette porte ET on utilise l'outil de saisie graphique de Quartus.

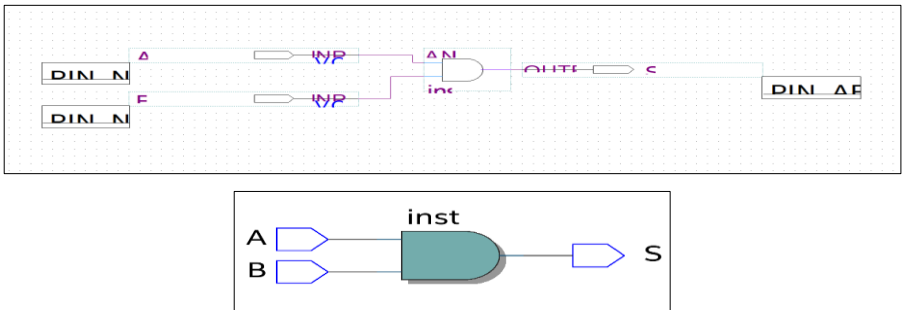

Figure 2: Saisie sur Quartus et schéma électrique résultant

Simulation fonctionnelle : Ce type de simulation permet de ne pas tenir compte des temps de propagation et de simuler seulement la fonction.

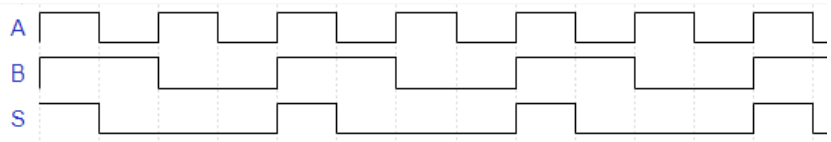

Figure 3: Résultat de simulation de la porte ET

La simulation temporelle : Elle permet de tenir compte des temps de propagation des composants matérielles.

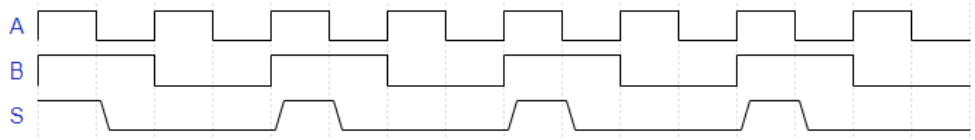

Figure 4: Simulation fonctionnelle

2. Additionneur

Il s'agit de réaliser trois additionneurs (2 mots 1 bits, 2 mots 3 bits et 2 mots 5 bits) en utilisant le langage de description VHDL et aussi l'outil de saisie graphique disponible sur Quartus.

Additionneur 2x1 bit

Figure 5 : Vue boite noire

Implémentation avec du VHDL

```
--Debut de declaration de l'entite
entity ent_add1 is
port(
       --Entrée logique
       A,B,Cin: in std_logic ;
       --Sortie logique
       Cout,S: out std_logic  );
end;
--Fin de declaration de l'entite

--Début de la desciption de l'archictecture
architecture arch_add1 of ent_add1 is
begin
-- mise en oeuvre de l'architecture de l'additionneur 2x1 bit par des équations
S <= (A xor B )xor Cin;
Cout <= (A and Cin) or (B and Cin) or (A and B);
end arch add1;

--Fin de la desciption de l'archictecture
```

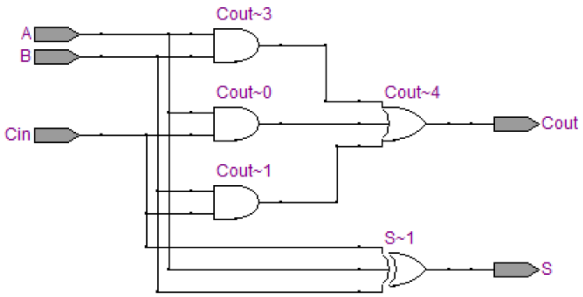

Figure 6: Circuit générer par Quartus

Additionneur 2x3 bits

Sur ce même principe on à réaliser l'additionneur 2x3bits en utilisant 3 additionneurs 2x1bit montés en cascade en utilisant l'outil graphique.

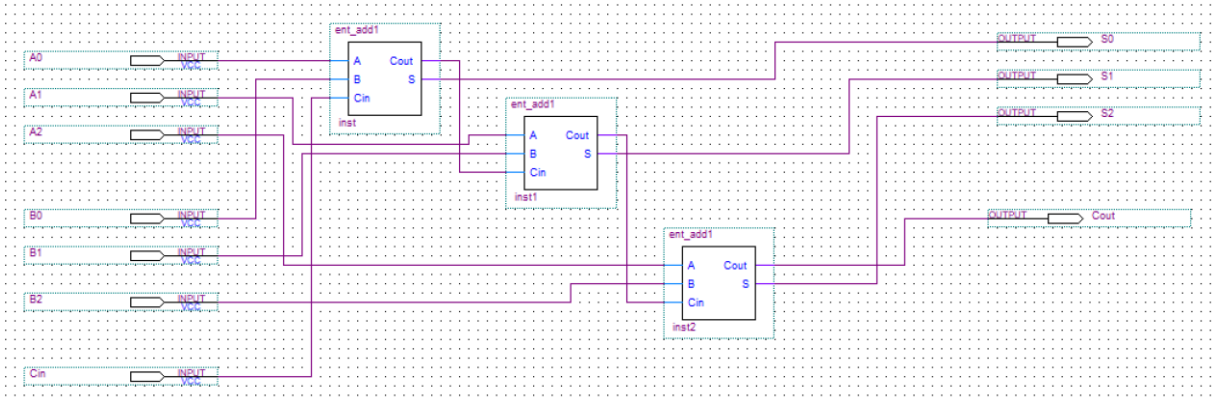

Figure 7: ADD2x3bits avec des ADD2x1

Additionneur 2x5 bits

Pour réaliser l'additionneur ADD2x5bits on utilise une autre approche qui consiste à utiliser des vecteurs sur 5 bits en VDHL, à l'aide de la bibliothèque :

```
use ieee.numeric_std.all;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.numeric_std.all;

entity add5_q9 is

port(
        -- Vector
        a, b: in std_logic_vector(4 downto 0);
        Cin: in std_logic ;

        s: out std_logic_vector(4 downto 0);
        Cout: out std_logic );
end;
```

Schéma résultant (voir figure 8)

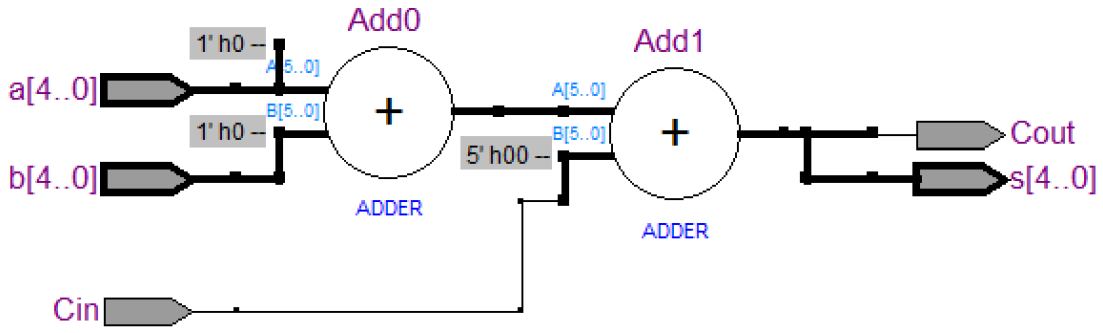

Figure 8. Schéma de connexions réalisé par Quartus

Simulation

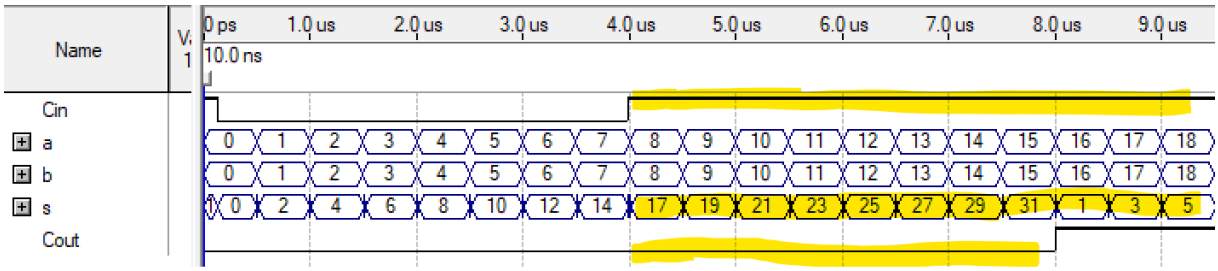

Figure 9: Simulation de ADD2x5bits

3. Mini projet

Dans ce mini projet il s'agit de réaliser un compteur BCD comptant des secondes en utilisant l'oscillateur interne du DE2 NANO et afficher le résultat de comptage sur un afficheur 7-segments.
Il s'agit concrètement de réaliser :

- Diviseur de fréquence
- Un compteur BCD
- Décodeur 7-segments

La réalisation de ce mini projet a permis de découvrir de nouveau concept du langage VHDL :

- La création et l'instanciation des composants pour réaliser une tâche
- L'utilisation d'un process avec les signaux de sensibilités
- L'utilisation des boucles (process) et de certaines instructions combinatoires (when).

La figure 9 montre le schéma fonctionnel du système.

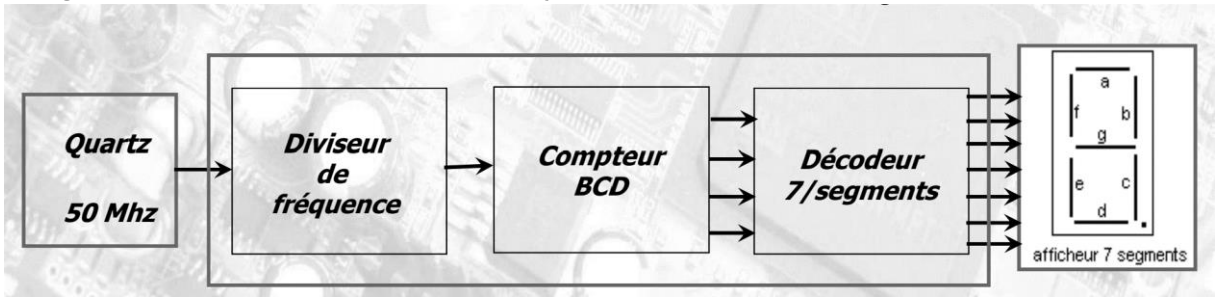

Figure 10: Schéma fonctionnel de l'ensemble

Le quartz utilisé est celui présent sur la carte DE2 en salle de TP.
Diviseur de frequence permet d'obtenir une fréquence de 1Hz correspondant à la seconde à partir de l'oscillateur interne de 50MHz. Voir le schéma fonctionnel la figure 11.

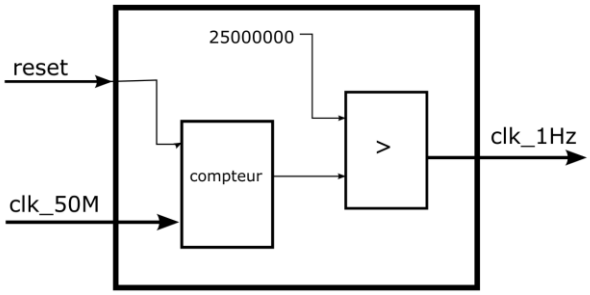

Figure 11: Schéma fonctionnel du diviseur

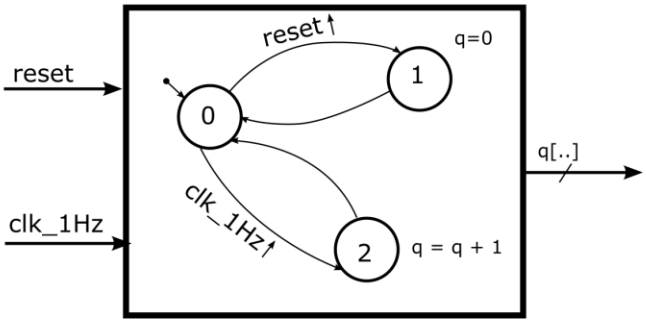

Figure 12: Principe de fonctionnement du compteur

Le décodeur est un système combinatoire qui permet d'associer à chaque valeur de q une lettre [abcdefg] avec un multiplexeur.

4. Circuit de génération de PWM

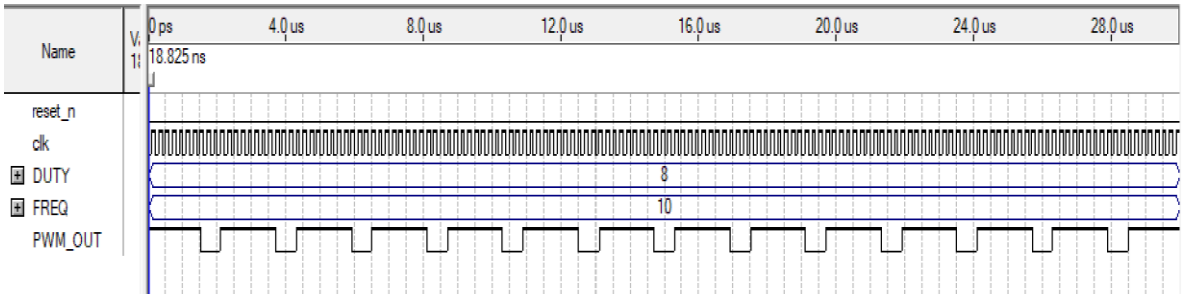

Figure 13: Simulation PWM

II. BE : Interfaces pilote de barre franche

L'interface de barre franche est constituée de hardware et de software permettant de remplir chacune une fonction particulière. Dans ce BE il s'agit de concevoir quelques fonctions faisant partie de l'interface de pilote de barre franche. Une fonction dite simple et une fonction dite complexe.
Ces fonctions seront intégrées sous forme d'IP dans un SOPC (system on programmable chip). Ce SOPC est un microcontrôleur utilisant l'architecture NIOS II et communicant avec les autres périphériques en utilisant le bus **avalon** (voir partie 3). Afin de que ces fonctions soit compatible pour être intégré dans le SOPC on ajoute à ces derniers une un registre et une interface avalon.

1. Conception d'une fonction simple (gestion_anemometre) avec son interface

1.1 Spécifications de l'anémomètre

Le système doit être capable de mesurer une fréquence entre 0 et 250Hz
Le temps de réponse du système doit être maximum de 1 seconde.
Les données doivent être dans un registre **config** et **data**
Le système doit disposer du bus communication Avalon pour les échanges de données.

1.2 Architecture générale

La figure 14 montre l'architecture générale du système à concevoir et à mettre en parallèle avec la figure 15.
La partie fonctionnelle représente ce qui doit être implémenté en hardware. L'ajout des registre config, code et l'interface avalon permet de concevoir **avalon_gestion_anemometre** qui est essentielle pour l'intégration dans le SOPC. Ceci permettra de récupérer les données du registre à travers le bus avalon et faire les traitements nécessaires avant une éventuelle utilisation.

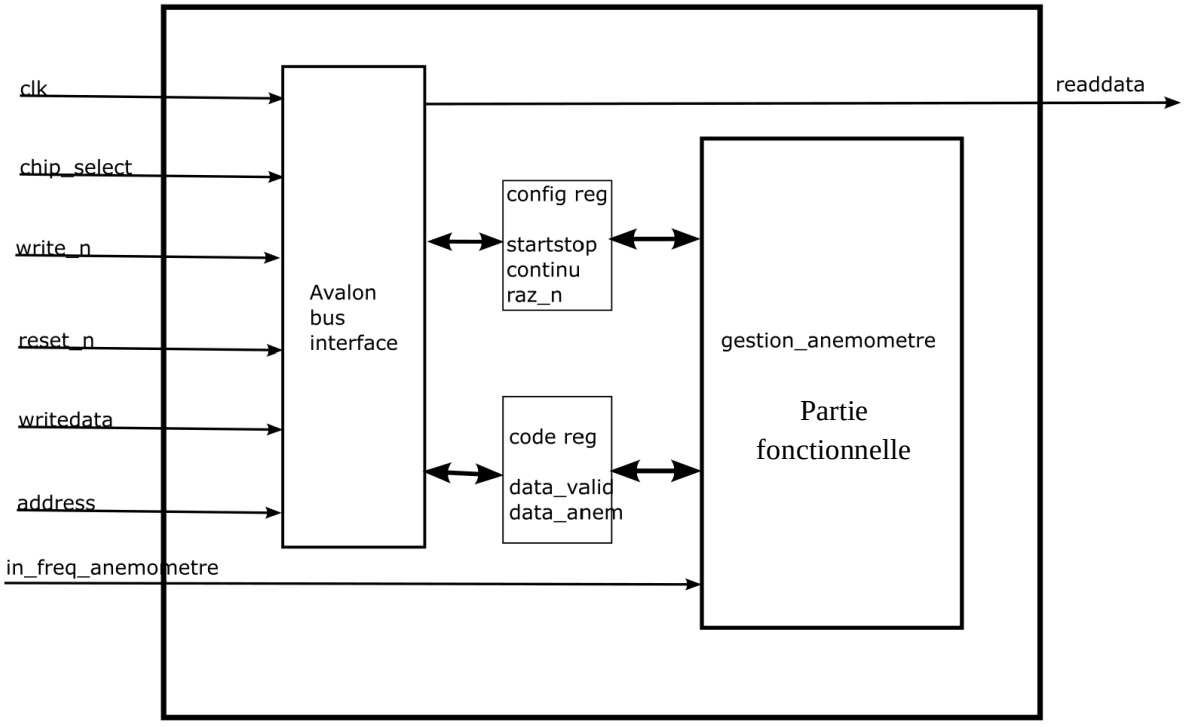

Figure 14 : Architecture générale

1.3 Description de la partie fonctionnelle

Notre choix de fonction simple s'est porté sur l'anémomètre. Cette fonction permet de mesurer la vitesse du vent entre 0 et 250km/h en mesurant la fréquence issue d'un capteur de vent. Cette fréquence varie entre 0 et 250Hz. La valeur de 0 et 250km/h correspondent respectivement à la fréquence de 0 et 250Hz. La fonction doit permettre de mesurer cette fréquence et renvoyé la valeur vers le bus Avalon pour réaliser des asservissements.

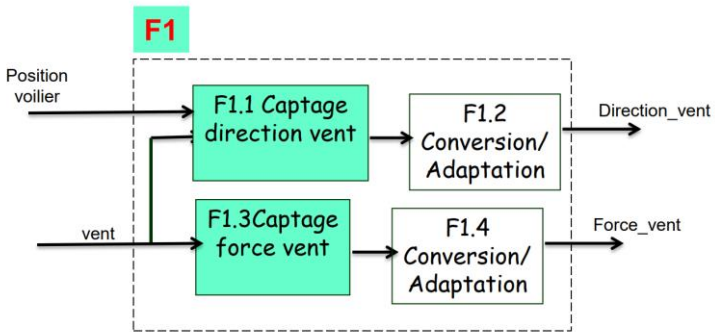

Figure 15: En vert, la partie hardware et en blanc partie software

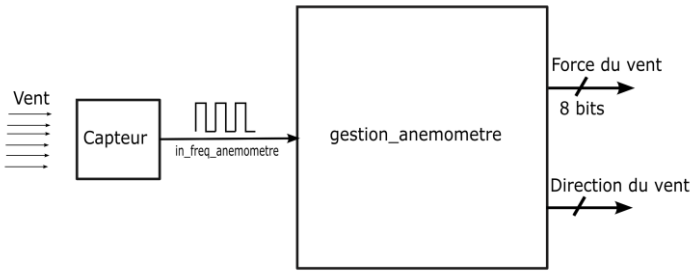

Figure 16: Schéma fonctionnel simplifié de l'anémomètre

Principe de mesure : Principe de la mesure de in_freq_anemometre basé sur le comptage des fronts montants de in_freq_anemometre pendant une période de 1s. Ce qui permet d'avoir directement la fréquence. Voir figure 15.

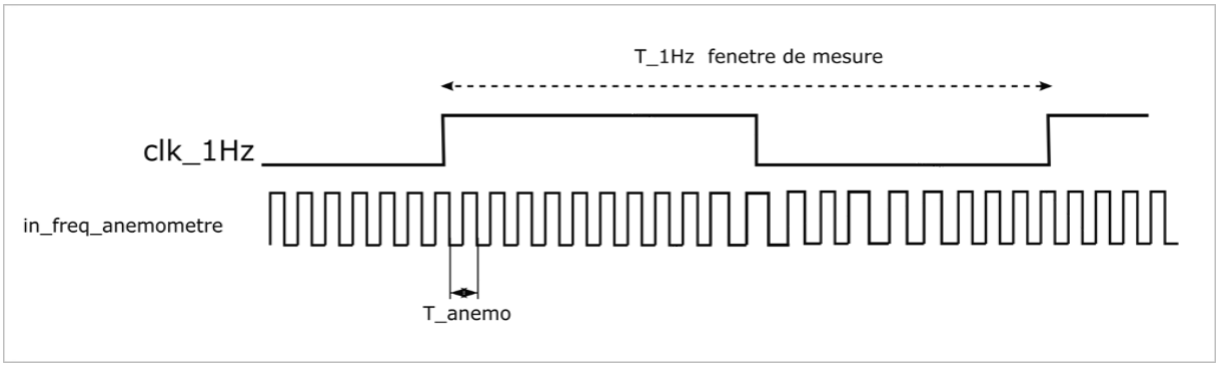

Figure 17: Principe de mesure in_freq_anemometre

Description de l'interface

Clk_50M : Horloge de 50MHz présente sur la carte DE2 NANO. Elle permet de piloter tous les éléments du système.
In_freq_anemometre : la fréquence entrante à mesurer
Raz_n : actif à l'état haut. Permet de réinitialiser le système.
Continu : Anémomètre réalise des mesures de façon continue lorsque actif.
Start_stop : Permet de lancer une mesure lorsqu'on n'est pas en mode continu. (1bit)

Data_anemometre : Contient les données de mesures sur 8bits.
Data_valid : Actif lorsqu'une donnée est disponible et mise à zéro quand les données sont lues.

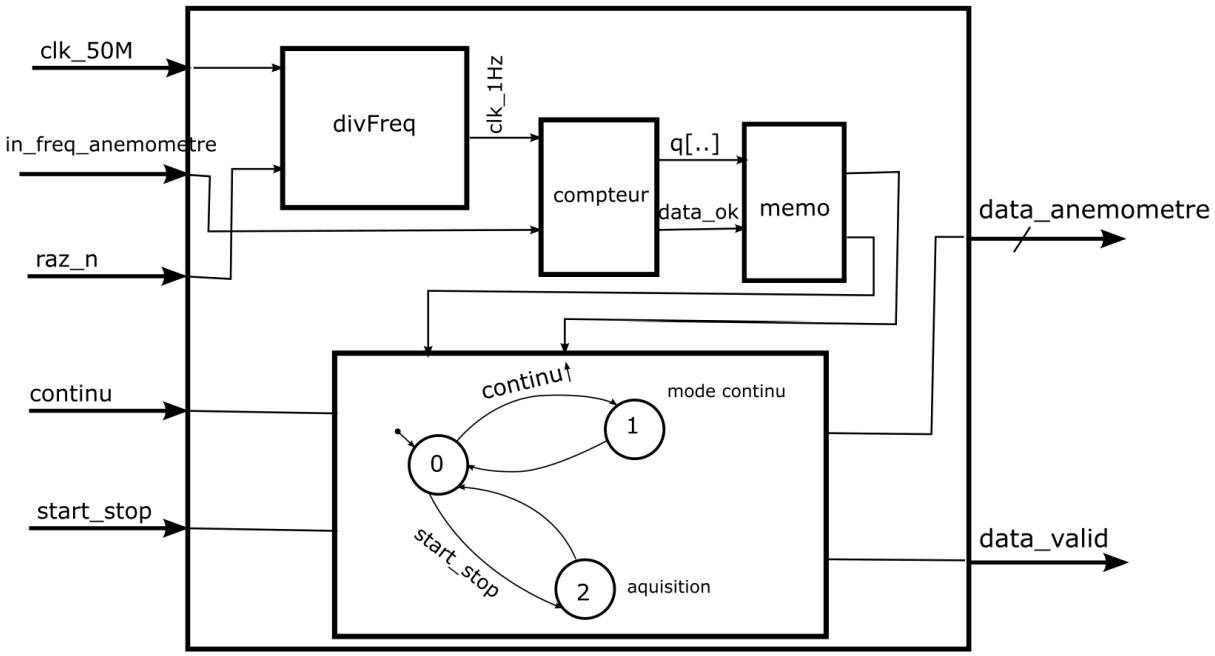

Figure 18: Architecture finale de l'anémomètre

Ci-dessous, on peut avoir le schéma électrique générer par Quartus juste pour la mesure de la fréquence. La machine à état implémenté permet de gérer les différents modes de fonctionnement.

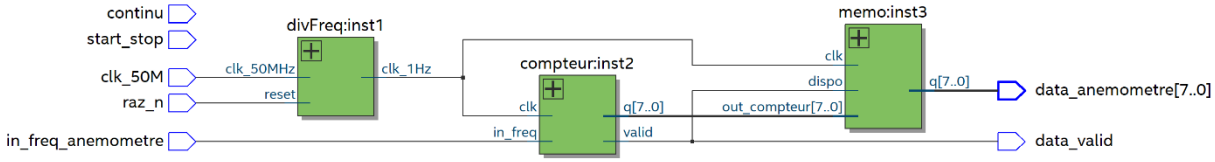

Figure 19: Schéma électrique générer par Quartus

1.4 Description de l'ensemble **avalon_gestion_anemometre**

Cette étape est nécessaire pour intégrer la fonction dans le SOPC. On ajoute à la partie fonctionnelle une l'interface avalon. L'interface avalon est décrite ci-dessous : (voir figure 14)

- Des entrées sorties

```
entity avalon gestion anemometre is
port(
clk 50M,chipselect, write n, reset n : in std logic;
writedata : in std logic vector (31 downto 0);
readdata : out std logic vector (31 downto 0);
address: std_logic_vector (1 downto 0);
in freq anemometre: in std logic
);
end avalon_gestion_anemometre;
```

Certains signaux d'entrée sorties deviennent des signaux internes

- Une fonction d'écriture dans le registre
- Une fonction de lecture

2. Conception d'une fonction compliqué : **gestion_bouton**

2.1. Spécifications de **gestion_bouton**

Le clavier du Tillerpilot a été conçu pour une utilisation aussi simple et intuitive que possible.
Il doit etre capable de passer du mode veille au mode automatique et aussi etre capable de régler le CAP de direction :

1. **Le mode veille**
   - La led STBY clignote
   - Un appui sur Babord (<) ou Tribord (>) modifier le cap de dans la direction indiquée, ce réglage est confirmé par un bip et un éclat de led Babord ou Tribord.
2. **Le mode automatique**
   - Un appui sur STBY verrouille le Tillerpilot pour passer du mode manuel au mode automatique.
   - Un appui 1 fois sur Babord (<) ou Tribord (>) modifier le cap de 1° dans la direction indiquée, ce réglage est confirmé par un bip et un éclat de led Babord ou Tribord.
   - Un appui long sur Babord (<) ou Tribord (>) modifier le cap de 10° dans la direction indiquée, ce réglage est confirmé par un double bip et double clignotement de la led Babord ou Tribord.

2.2. Architecture générale

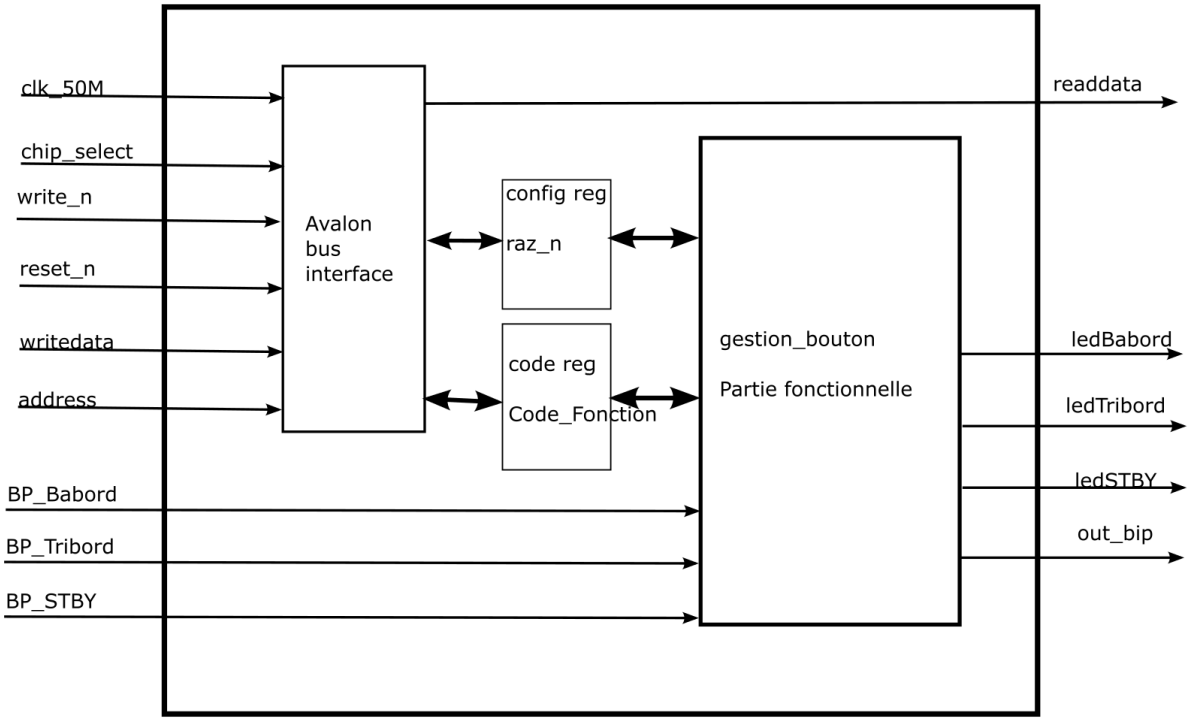

Figure 20: Architecture générale de gestion_bouton

2.3. Description de la partie fonctionnelle

La figure 21 montre schéma fonctionnelle de gestion_bouton.

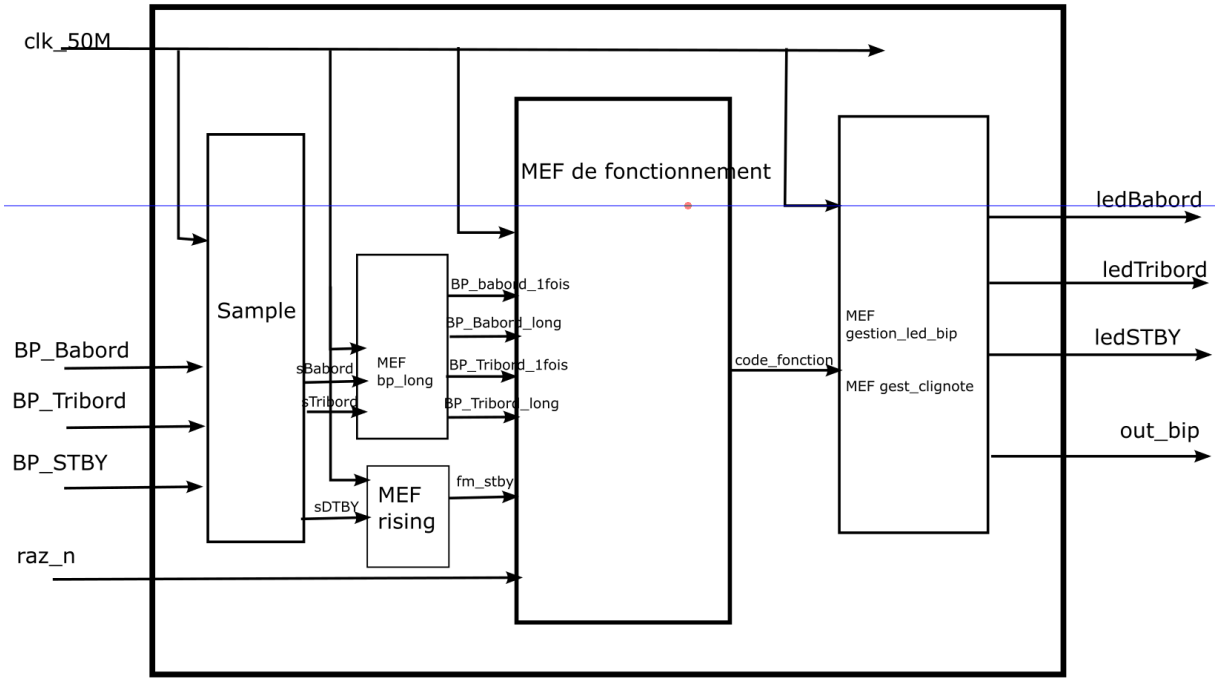

Figure 21: Partie fonctionnelle de gestion_bouton

Description des différentes fonctions
Sample : cette fonction permet d'échantillonner les trois boutons BP_Babord, BP_Tribord et BP_STBY. Elle renvoi l'état du bouton chaque 1ms. Ceci permet d'éviter des bruits parasites sur le bouton.

Bp_long : Cette fonction renvoi des signaux appui court (…_1fois) ou appui long lorsqu'il détecte un appui sur les boutons BP_Babord ou BP_Tribord

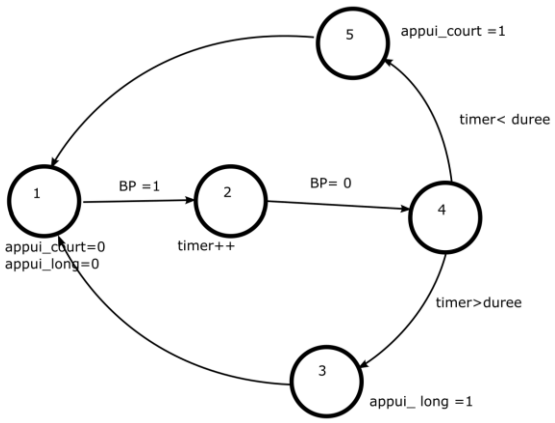

Figure 22: Gestion appui long

MEF rising : Cette fonction permet de détecter le front montant ou le front descendant sur le bouton BP_STBY.

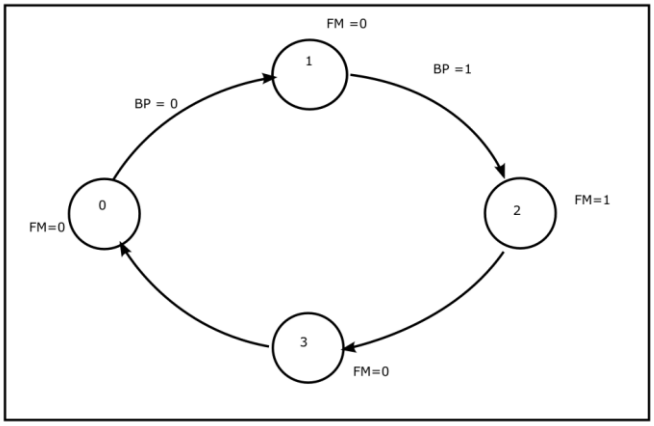

Figure 23: MEF du détecteur de front montant sur BP_STBY

MEF de fonctionnement :
Voir figure 23, 24, 25.

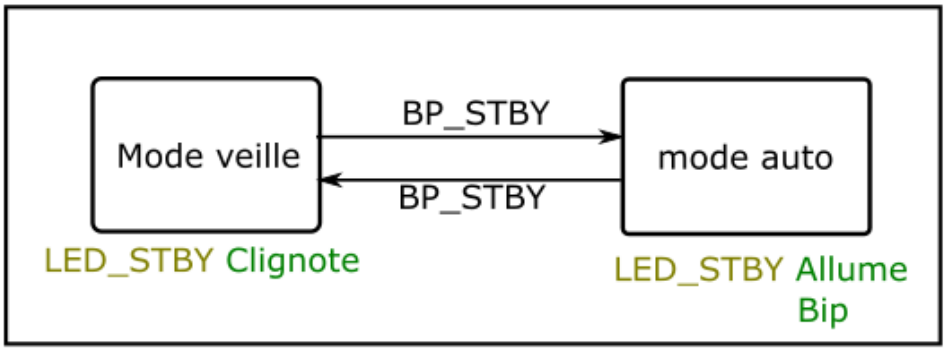

Figure 24: Machine du fonctionnement générale de gestion_bouton

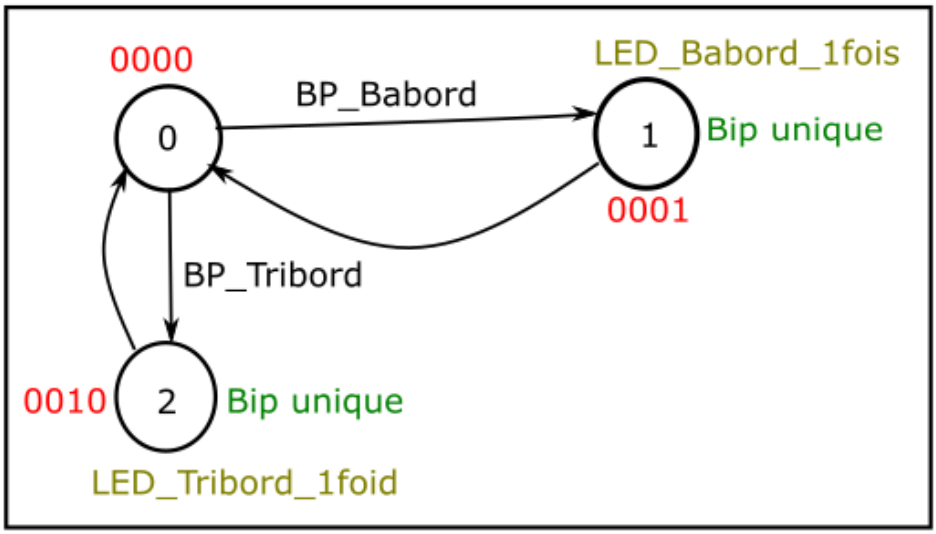

Figure 25: mode veille

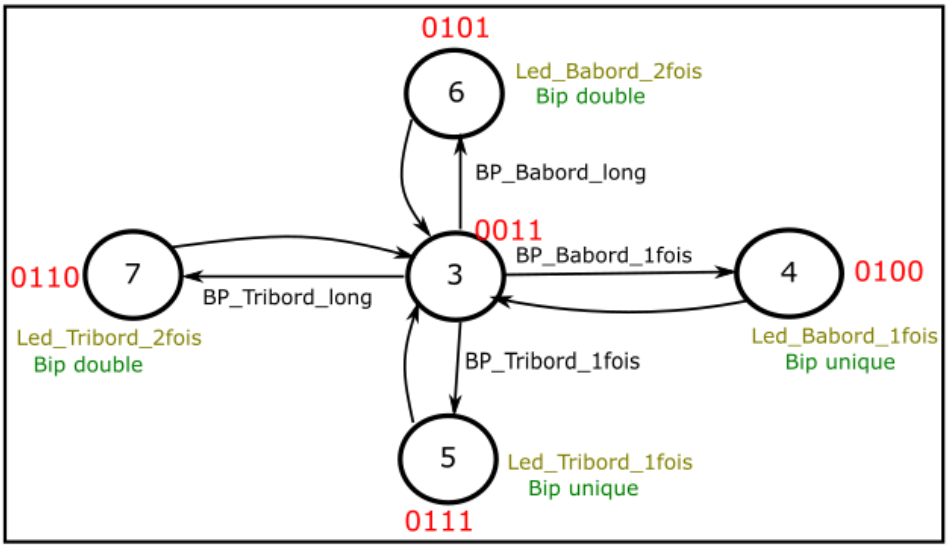

Figure 26: Mode automatique

Gestion_led_bip : Cette fonction gère les LEDs et les bips pour confirmer les fonctions.

Remarque : Sur le prototype de la carte de barre franche disponible en salle de TP, un appui sur les boutons poussoir fait une mise à zéro.

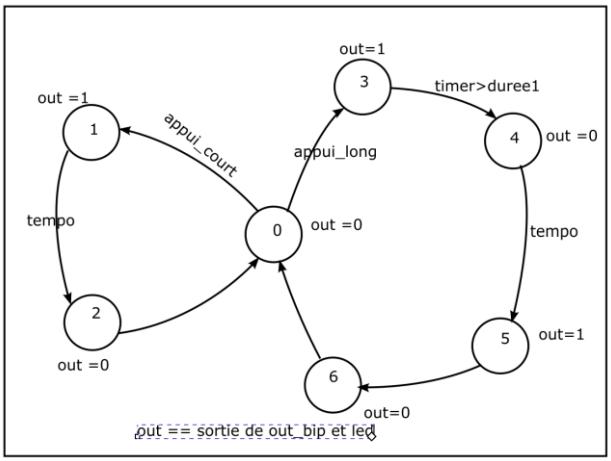

Figure 27: MEF gestion_led_bip

3. Conception de SOPC

Un SOPC est un système sur puce programmable. Pour créer notre SOPC intégrant les fonctions on utilise l'outil de « Platform designer » présent dans Quartus.
Ce SOPC est constitué de :

- CPU basé sur l'architecture NIOS II
- Une mémoire, on_chip_memory
- Un bus
- Des périphériques de bases comme les boutons, les LEDs
- Des fonctions propriétaires (gestion_anemometre et gestion_bouton)

La figure ci-dessous, montre l'architecture finale du système conçu avec toutes les fonctions.

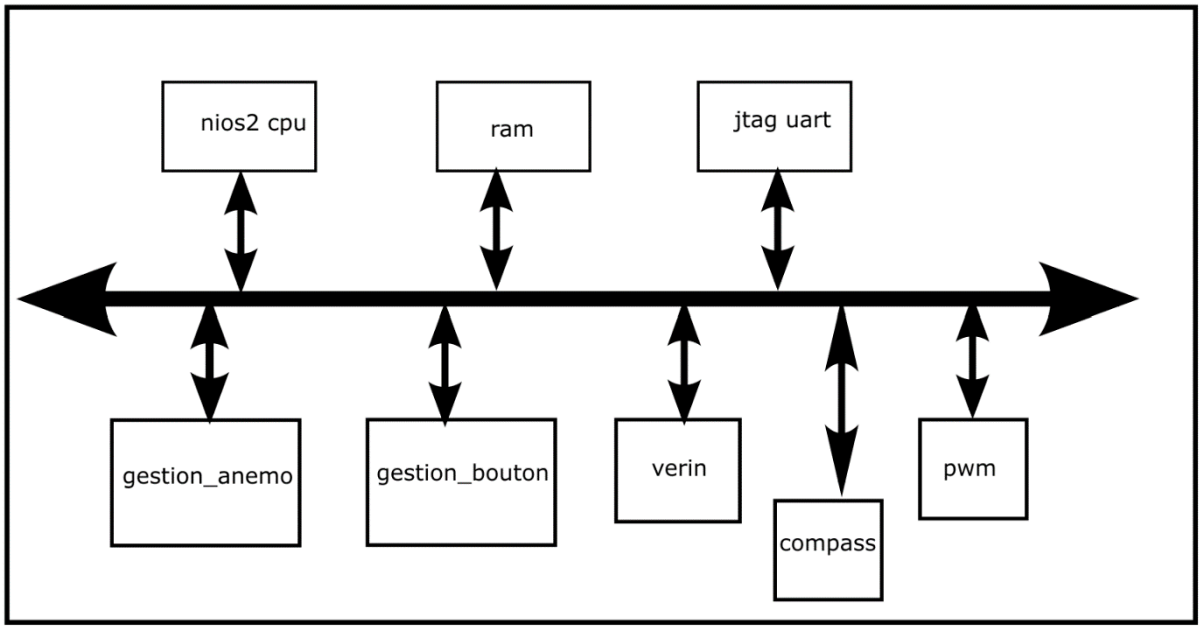

Figure 28: SOPC

4. Programmation du SOPC sous Eclipse™

Conclusion

Ce be VHDL nous a permis dans un premier temps lors des TP de bas de s'initier à l'utilisation d'environnement Quartus II ainsi l'utilisation de la carte DE0 (FPGA), il nous permet aussi de s'initier à l'utiliser de Platform designer pour la conceptions des microcontrôleurs (SOPC) comportent une partie processeur à laquelle on associe des périphériques ou des fonctions.

Dans un second temps ce dernier nous a permis de réaliser une interface pilote d'une barre franche, en réalisant la conception d'une fonction simple (anémomètre) et d'une fonction complique (boutons), tout en utilisant l'interface du bus avalon pour chaque fonction pour l'intégration dans SOPC.

Le déroulement du BE c'est porté sur une validation de chaque étape du TP par l'enseignant, pour notre part on a pu valider tous les Tp de bases et ainsi la fonction simple et la fonction compliqué.

Listes des figures