



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



UNIVERSITÉ PAUL SABATIER

RAPPORT DE Travaux Pratiques

BUREAU D'ÉTUDES

MISE EN OEUVRE D'UN PILOTE DE BARRE

FRANCHE POUR VOILIERS

Auteurs :

Lucky ROBINSON
Hafsa RAOUI

Encadrant :

Mr Thierry PERISSE

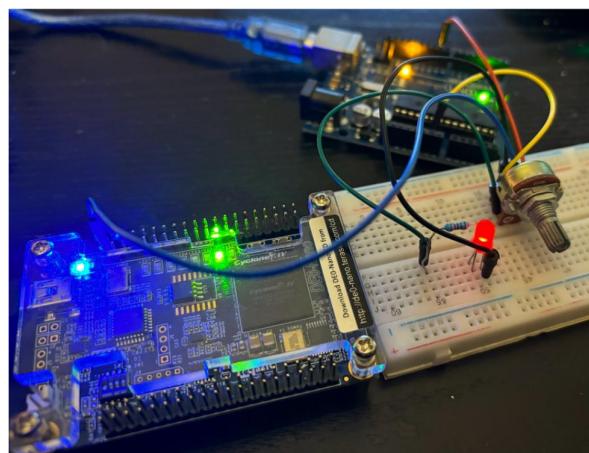


TABLE DES MATIÈRES

Introduction Générale	1
1 Etude du matériel	2
1.1 Carte DE2	2
1.2 Carte DE0 Nano	3
1.3 Générateur Basse Fréquence et Oscilloscope	3
1.4 Générateur signal PWM : Arduino	5
2 TP de base	6
2.1 Introduction	6
2.2 Implémentation d'une porte logique ET	6
2.3 Additionneur de 2 mots de 1 bit	7
2.4 Mini projet	8
2.4.1 Bloc 1 : Diviseur de fréquence	9
2.4.2 Bloc 2 : Compteur/Décompteur BCD	9
2.5 Bloc 3 : Décodeur 7 segments :	11
2.6 Simulation et test sur la carte du mini projet	12
2.7 Génération d'une PWM :	12
3 BE : Pilote de barre franche pour voiliers	14
3.1 Fonction simple : Anémomètre	15
3.1.1 Analyse fonctionnelle	16
3.1.1.1 Génération du signal 1hz	16
3.1.1.2 Détecteur et compteurs de fronts montants de l'horloge 1 Hz :	18
3.1.1.3 Mémorisation d'événement	19
3.1.1.4 Le choix de mode	20
3.1.2 Configuration sur SOPC : System On Programmable Chip	21
3.1.2.1 Implémentation et conception du Plateform Designer sur projet	21
3.1.2.2 Affichage Hello Word et Bouton Poussoir	21
3.1.2.3 Génération du PWM	22
3.1.3 Configuration sur SOPC pour l'avalon Anémomètre	24
3.2 Fonction complexe : le vérin	26
3.2.1 Gestion du PWM	26
3.2.2 Gestion des butées	27
3.2.3 Gestion du convertisseur AN MCP 3201	28
3.2.3.1 Machine à état	29
3.2.3.2 Comptage des fronts d'horloge	29
3.2.3.3 Registre à décalage	29
3.2.3.4 Génération du 1MHz	29
3.2.3.5 Génération périodique	29
Conclusion Générale	31

INTRODUCTION GÉNÉRALE

Dans le domaine de l'électronique, la connaissance des systèmes basés sur le langage de description matérielle ou le VHDL est primordiale pour mettre en oeuvre des projets sur du SOC FPGA. L'intérêt d'étudier et de travailler dans cet environnement nous permet de réaliser un projet suivant les étapes de la conception, la spécification, la modélisation, la simulation et la synthèse du système. Nous avions effectué un Bureau d'étude qui consite à étudier et mettre en oeuvre la gestion automatique d'une barre franche d'un voilier.

Ce module de synthèse et mise en oeuvre de systèmes a pour but de comprendre et de manipuler les circuits logiques programmables et l'environnement d'un FPGA sur les carte Altera DE2 et DE0 Nano. Aussi, l'environnement du quartus Prime 18 nous permet d'interfacer le processeur de la carte avec le FPGA reprogrammable grâce à l'IDE NIOS II basé sur du langage C.

Le choix de ces cartes nous nous facilite le test des fonctions logiques sur l'environnement Quartus Cyclone II et IV. Ces fonctions logiques simples seront assemblées ou instanciées pour former des circuits de plus en plus complexes et on réalise ainsi le système. Ainsi, les démarches du travail se font par décomposition du système en plusieurs blocs et sous blocs fonctionnels.

CHAPITRE 1

ETUDE DU MATÉRIEL

Dans cette première partie, nous allons voir les différents matériels qu'on a utilisé durant les séances de TP de base et le projet en Bureau d'étude.

1.1 Carte DE2

La carte DE2 est basée sur du SOC FPGA Cyclone II 2C35 qui nous a aidé à réaliser des fonctions en VHDL et des circuits logiques. Les composants de la carte sont reliés aux broches des puces du microprocesseur. Cela permet à l'utilisateur d'utiliser les fonctionnalités présentes sur la carte. Durant les séances de TP, nous avions surtout configurés et utilisés les interrupteurs (interrupteurs à bascule et bouton-poussoir), des LEDs et les affichages à 7 segments.

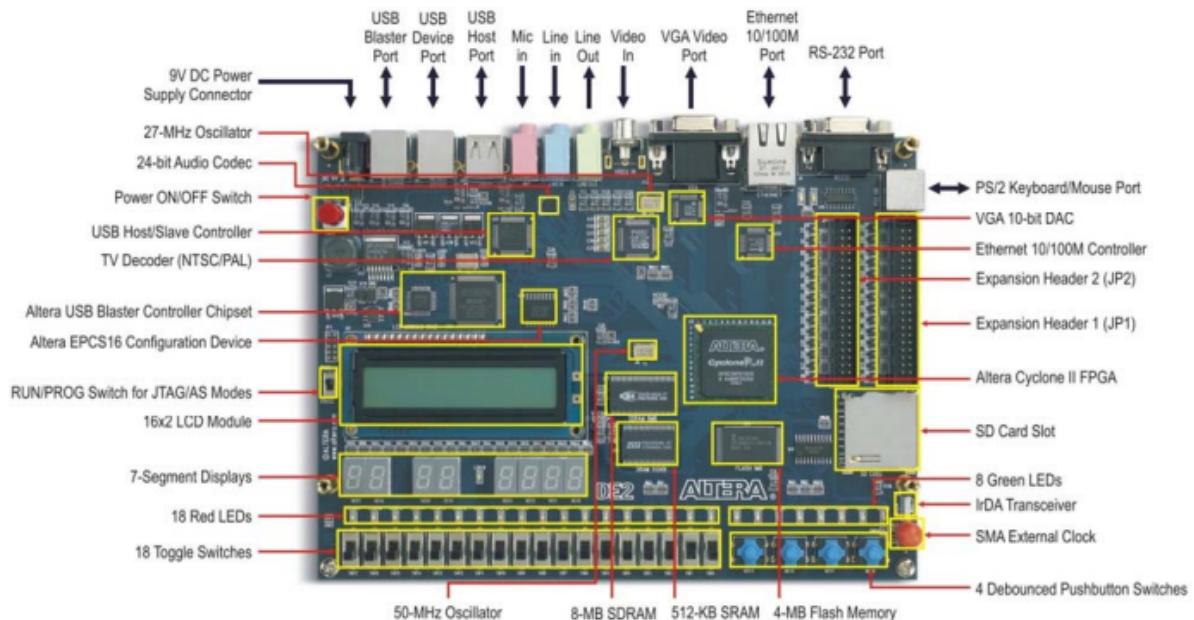


FIGURE 1.1 – Carte DE2

1.2 Carte DE0 Nano

Cette carte de développement de la famille Altera Cyclone IV est adaptée pour travailler sur des projets individuels. Par sa taille et son prix réduits, on a utilisé cette carte pour le bureau d'étude et a pour cible les composants Quartus Cyclone IV. Elle a la possibilité d'être reconfigurable. Les modules ont été intégrées grâce à des outils de développement sur SOC (System On Chip) programmable d'Altera sur une carte DEO NANO. La carte DE0-Nano comprend aussi des LEDs et des deux boutons de poussoirs. Les périphériques sont configuré par les drivers de Altera Cyclone IV. L'alimentation de la carte se fait par l'intermédiaire d'un connecteur USB mini-AB qui permet de relier la carte à l'ordinateur. Cette carte sera utilisée pour réaliser le traitement de données collectées par les capteurs et pour pouvoir commander notre système. La carte est basé sur un oscillateur d'horloge à Quartz fonctionnant avec une fréquence de 50 MHz. Elle est constituée de 80 GPIO répartie où chaque moitié est répartie sur deux groupes : GPIO 0 et GPIO 1. Cela permet de faire transiter les signaux depuis la carte vers un autre dispositif ou l'inverse. Les 8 Leds sur la carte permettent de visualiser les sorties sur 8 bits. Les boutons de poussoir et les switch permettent d'agir sur un signal.

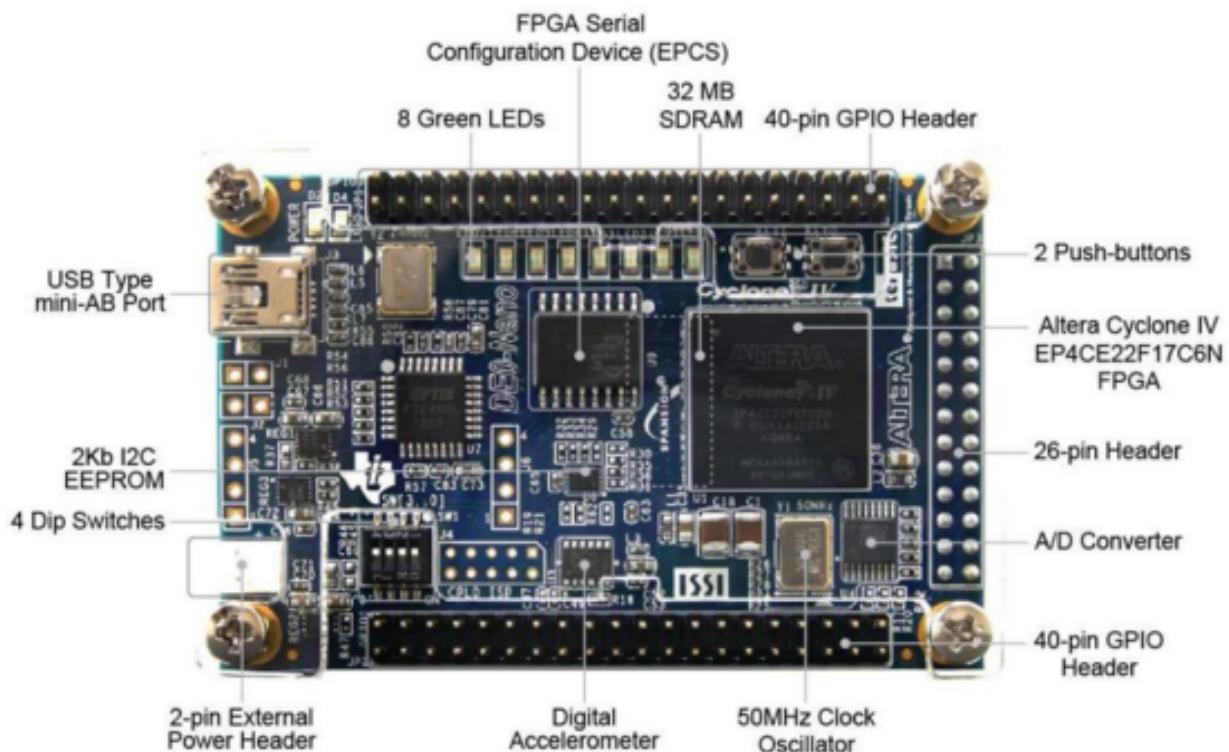


FIGURE 1.2 – Carte DE0 nano

1.3 Générateur Basse Fréquence et Oscilloscope

Durant le test de la fonction anémomètre sur la carte DE0 nano, nous avions connecté le GBF à un GPIO de la carte pour générer un signal carré externe à fréquence configurable. Pour visualiser ce signal d'entrée ou les signaux de sortie, nous avions utilisé l'oscilloscope de la salle de TP. Il s'agit donc de voir des trames de données et des signaux d'horloges et externes.

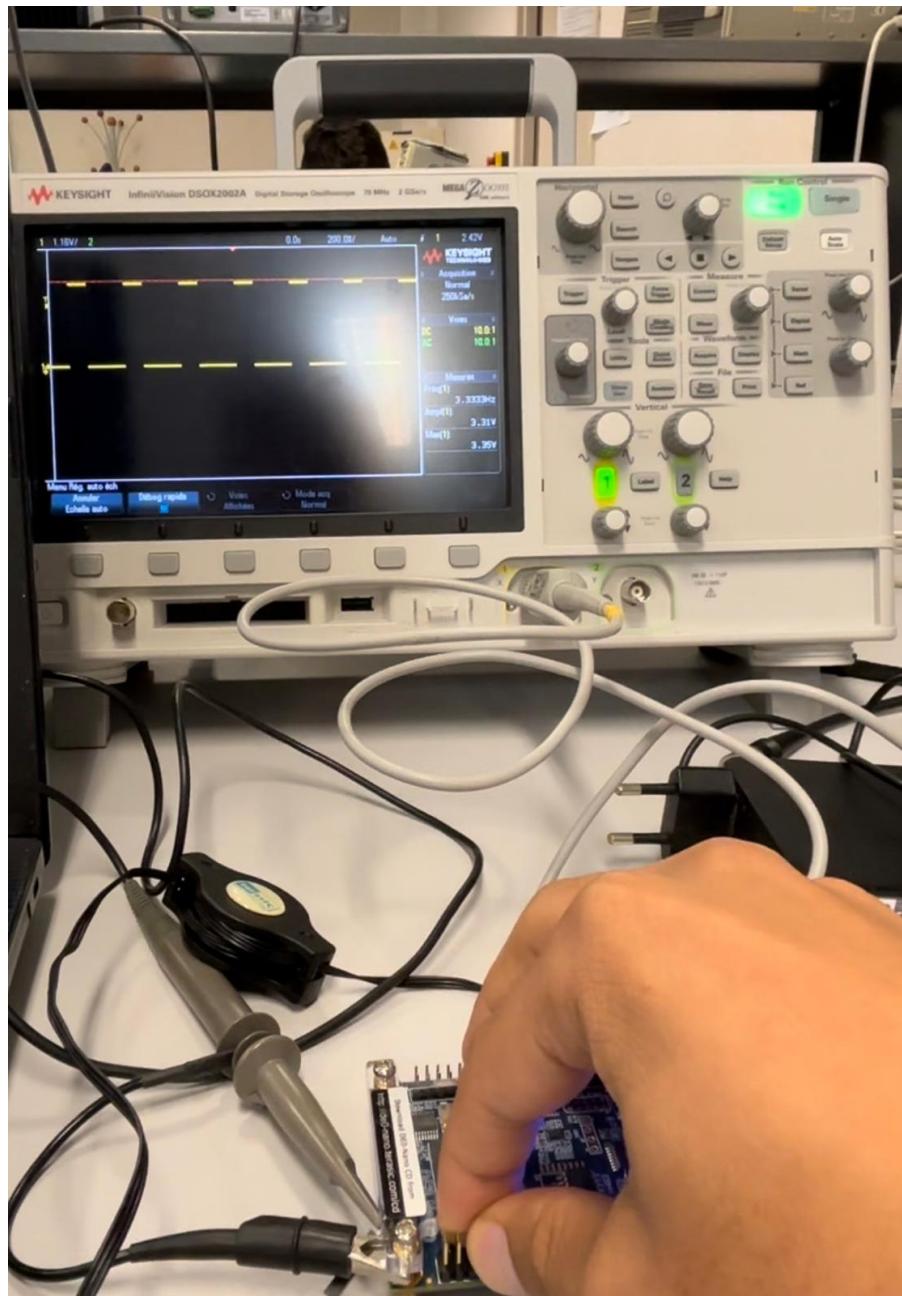


FIGURE 1.3 – Oscilloscope pour voir un signal en sortie d'un GPIO par exemple

Le générateur basse fréquence nous permet d'avoir un signal pwm externe pour faire de l'acquisition en entrée des blocs fonctionnels dans la fonction simple et complexe. En effet, il s'agit d'un signal rectangulaire où l'on fixe la tension à 4 V et la fréquence est variable suivant la plage de fréquence inscrite dans le cahier des charges où l'on a la fonction. On envoie ce signal pwm via une sonde connectée à la carte. Comme chaque poste n'a qu'un seul générateur basse fréquence, un autre moyen de générer un signal PWM à fréquence fixe est l'utilisation de l'arduino.

1.4 Générateur signal PWM : Arduino

Comme nous ne disposions pas d'un matériel comme le GBF chez soi, nous avons utilisé l'arduino qui est une plateforme open source et cela nous a permis de travailler chez soi pour pouvoir avancer dans le projet. Ainsi grâce à un code source Arduino qu'on a mis en annexe, nous avons généré un signal rectangulaire avec la carte arduino. Dans le code arduino , la librairie TimerOne() permet de contrôler la fréquence signal pwm généré par l'arduino sur une de ces broches. Le rapport cyclique est fixé à 50Voici le signal PWM obtenu à la sortie d'une broche de l'arduino.

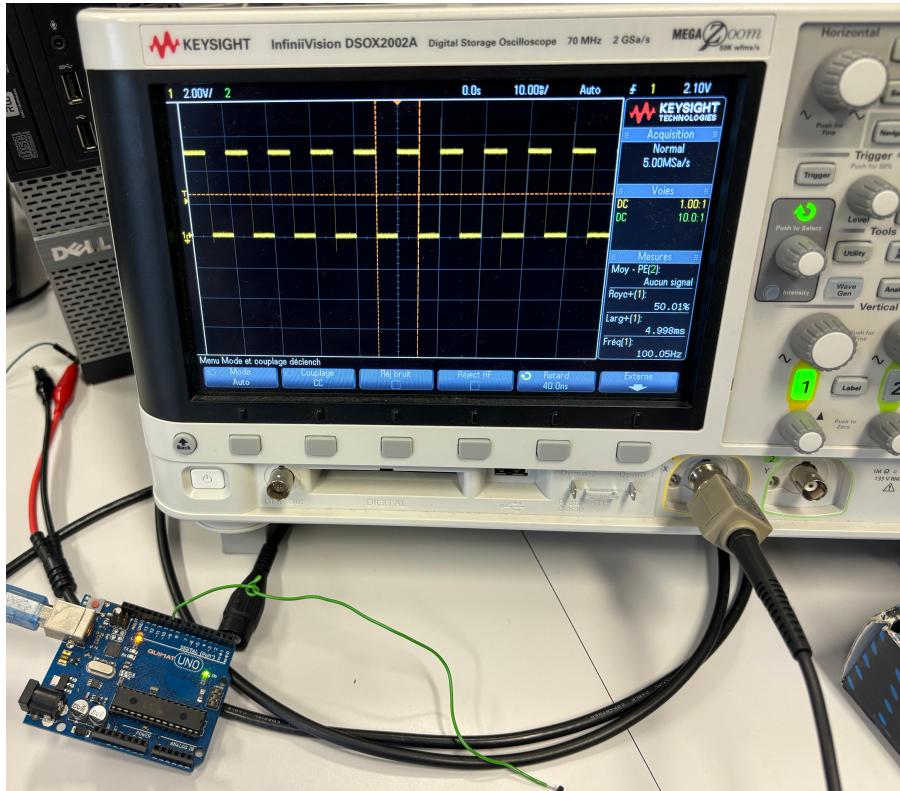


FIGURE 1.4 – Génération du signal PWM avec l'arduino sur l'oscilloscope

```
pwm | Arduino 1.8.19 (Windows Store 1.8.57.0)
Fichier Édition Croquis Outils Aide
pwm
#include <TimerOne.h>

const int pwmPin = 9; // Port de la sortie de signal PWM

void setup() {
  pinMode(pwmPin, OUTPUT);
  // Mise en marche le timer avec les valeurs de fréquence et DUTY
  Timer1.initialize(10000); // en microseconds -> 10000us => 100 Hz
  Timer1.pwm(pwmPin, 512); // 50% DUTY (0 à 1023)
}

void loop() {
  //delay(2000);
  // Your main code goes here
}
```

FIGURE 1.5 – Codant arduino générant le pwm

CHAPITRE 2

TP DE BASE

2.1 Introduction

Les séances de TP de base ont pour but de s'initier à l'utilisation de Quartus II. Il s'agit concrètement de concevoir une implémentation sur FPGA d'une fonction logique par des schémas blocs et des codes VHDL.

2.2 Implémentation d'une porte logique ET

Dans cette partie, nous avons construit un schéma bloc (interface bloc diagramme) pour la fonction logique ET. Dans un premier temps, nous avions effectué une simulation fonctionnelle et temporelle à partir de deux entrées (A et B) et une sortie S. Nous avions remarqué que lors d'une simulation temporelle, la sortie est décalée par rapport à l'entrée après avoir mis à jour la sortie. Ce décalage est généré par le composant de la porte ET. Donc plus on importe de composants dans le schéma, plus on génère des décalages par rapport aux entrées. Pour ne pas avoir ces décalages temporels, il faut effectuer une simulation fonctionnelle. Nous nous sommes basés de la table de vérité de la fonction logique ET pour effectuer la simulation fonctionnelle. Voici le résultat obtenu :

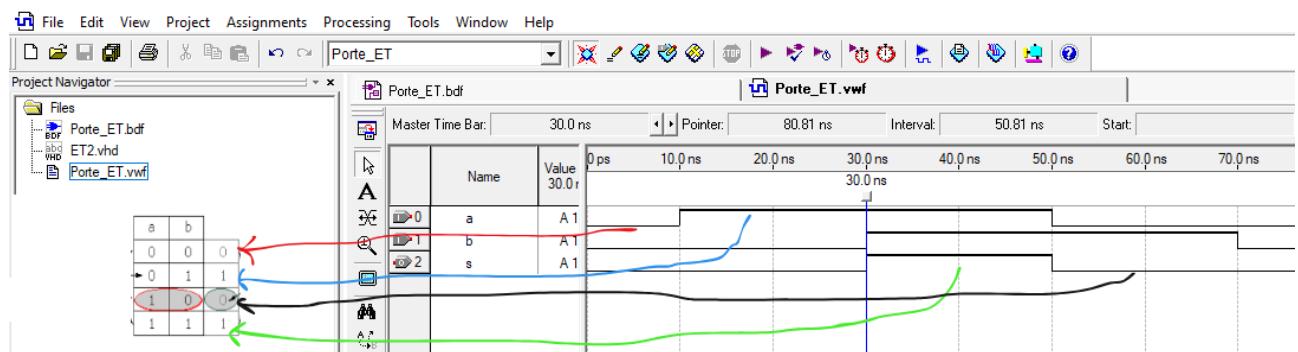


FIGURE 2.1 – Simulation Porte ET

Nous avions ensuite créer la fonction de la porte ET à partir du codage VHDL et le tester sur la carte DE2. En suivant le manuel de Quartus II, nous avions effectué la programmation du code dans le FPGA puis nous avions testé la fonction à partir des interrupteurs et leds de la carte. Pour tester le programme sur la carte DE2, nous avions configurer deux interrupteurs pour piloter les entrées A et B puis une led pour visualiser l'état de la sortie.

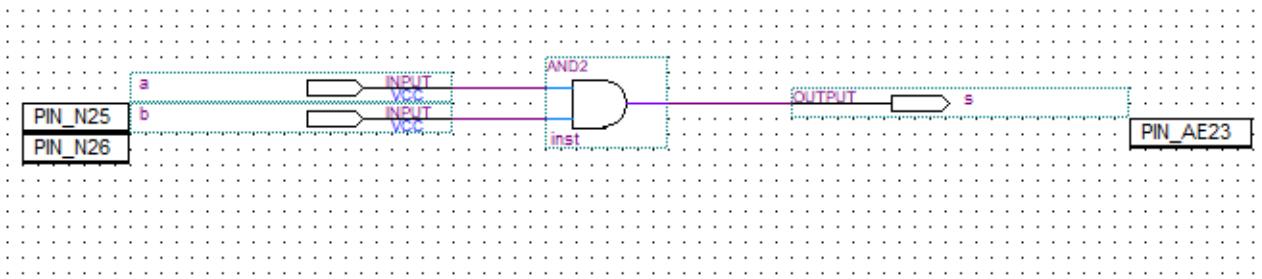


FIGURE 2.2 – Initialisation liaison I2C

2.3 Additionneur de 2 mots de 1 bit

Dans cette partie, nous avions réalisé la fonction additionneur de deux mots de 1 bit : les deux bits d'entrée. Il faudrait rajouter une entrée de retenue Cin, un bit de sortie et une retenue Cout. Il s'agit d'un circuit logique réalisant une addition. Pour avoir l'équation de la sortie, nous nous sommes basés de la table de vérité de l'additionneur et la table de Karnaugh.

Nous aurions pu construire le schéma structurel de la fonction additionneur. Cependant, Quartus II possède un moyen de transformer le code vhdl en schéma structurel. Dans le code VHDL, nous nous sommes basés des équations pour faire évoluer la sortie en fonction des entrées et retenues. Nous avons effectué la simulation en important les bits d'entrée et de sortie ainsi que les bits de retenue Cin et Cout. La simulation s'effectue suivant la table de vérité de l'additionneur.

AB		OO	O1	11	10
Cin	0	0	0	1	0
	1	0	1	1	1

Tableau de Karnaugh pour la sortie de retenue Cout

$$\text{Cout} = (\text{A Xor B}) \text{ Xor Cin}$$

AB		OO	O1	11	10
Cin	0	0	1	0	1
	1	1	0	1	0

Tableau de Karnaugh pour la sortie S

$$S = (\text{A Xor B}) \text{ OR } ((\text{A Xor B}) \text{ AND Cin})$$

FIGURE 2.3 – Tableau de Karnaugh

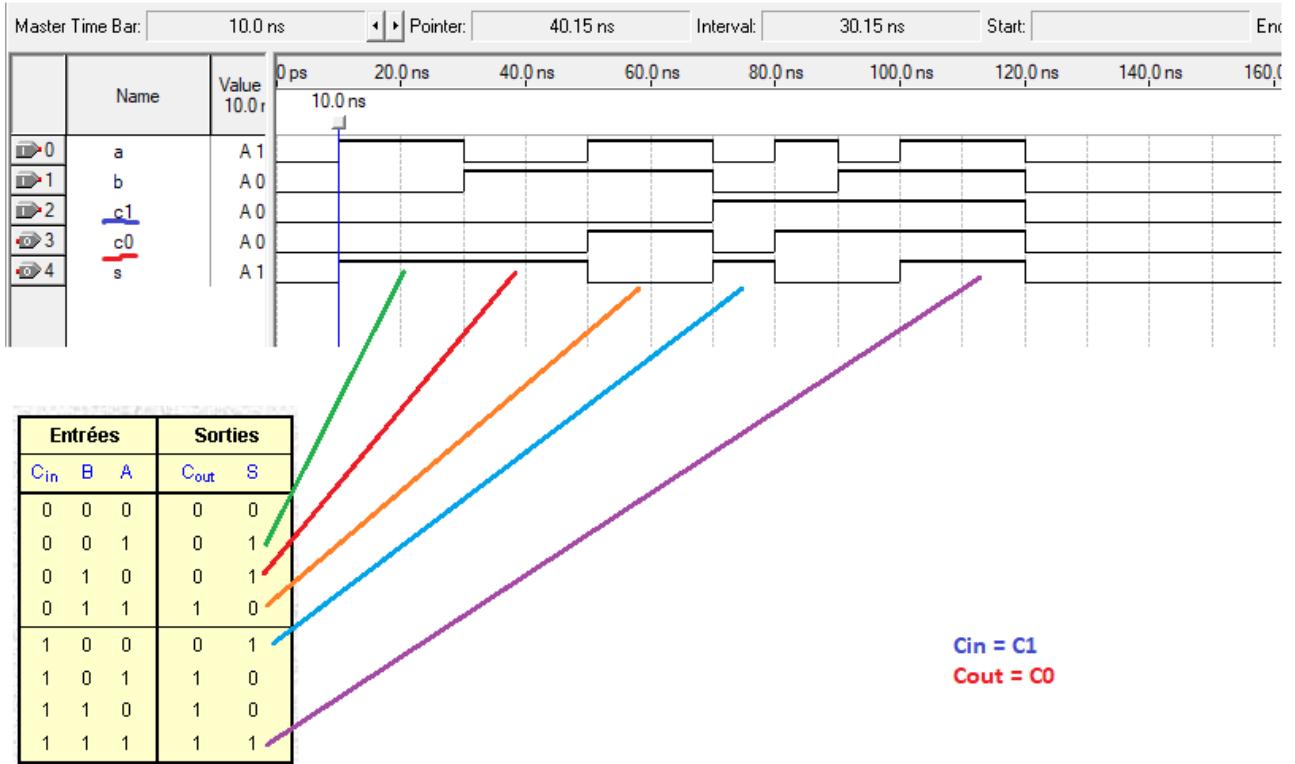


FIGURE 2.4 – Tableau de Karnaugh

Après avoir testé l'additionneur de 2 mots de 1 bit, nous avions implémenté la fonction additionneur de 2 mots de 3 bits. Pour cela, grâce à la fonctionnalité de Quartus pour créer un schéma bloc, nous avions pu former le schéma bloc de l'additionneur de 2 mots de 1 bit et l'instancier en 3 bloc pour avoir 3 bloc en cascade. Ces blocs en cascade forme l'additionneur de 2 mots de 3 bits.

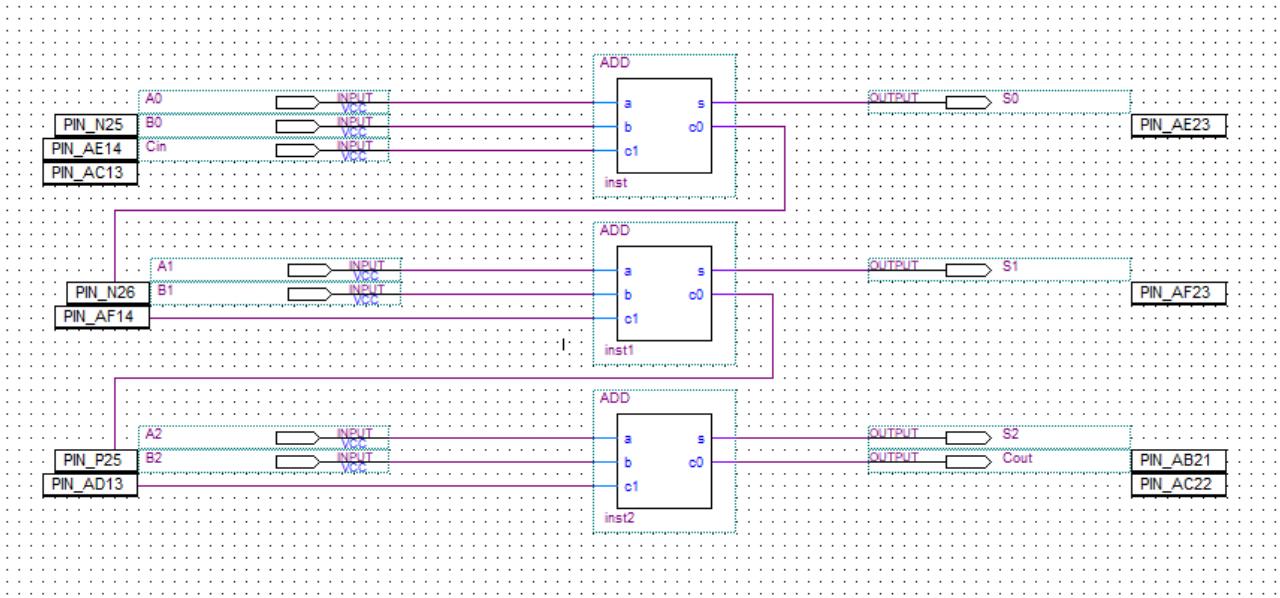


FIGURE 2.5 – Montage du capteur sur stm32

2.4 Mini projet

Dans cette partie, nous avions réalisé un mini projet qui consiste à l'affichage des secondes comptant de 0 à 9 en boucle dans un afficheur 7 segments. Nous avions besoin de la base de temps de la carte pour effectuer le comptage. Il s'agit d'un quartz à 50 MHz. La décomposition du miniprojet s'effectue en trois blocs principaux : - le diviseur de fréquence - Le compteur et décompteur BCD - Le décodeur BCD à 7 segments.

La broche de l'entrée du système est reliée au Quartz à 50 MHz et les broches de sorties sont reliés à l'afficheur 7 segments. Voici le schéma du système.

2.4.1 Bloc 1 : Diviseur de fréquence

Ce mini projet permet d'effectuer un comptage de temps en seconde donc la mise à jour de la valeur affiché se fait toutes les 1 secondes. En prenant la fréquence du Quartz à 50 MHz comme base de temps, nous allons lancer un compteur à chaque front montant de l'horloge de 50 MHz. Pour cela, sur une demi-période du signal 1 Hz on comptera 25 millions de front montant d'horloge à partir d'une incrémentation de compteur. Une fois qu'on atteint cette valeur, on fait évoluer le signal de sortie en basculant l'état du signal à 1 Hz en complémentaire à celui de la première demi-période. Pour des raisons de long délai de lancement de la simulation, nous avions imposé 2500 fronts montant à compter pour avoir une période de 100 microsecondes. Voici la simulation qui valide le résultat précédent ainsi que le diviseur de fréquence.

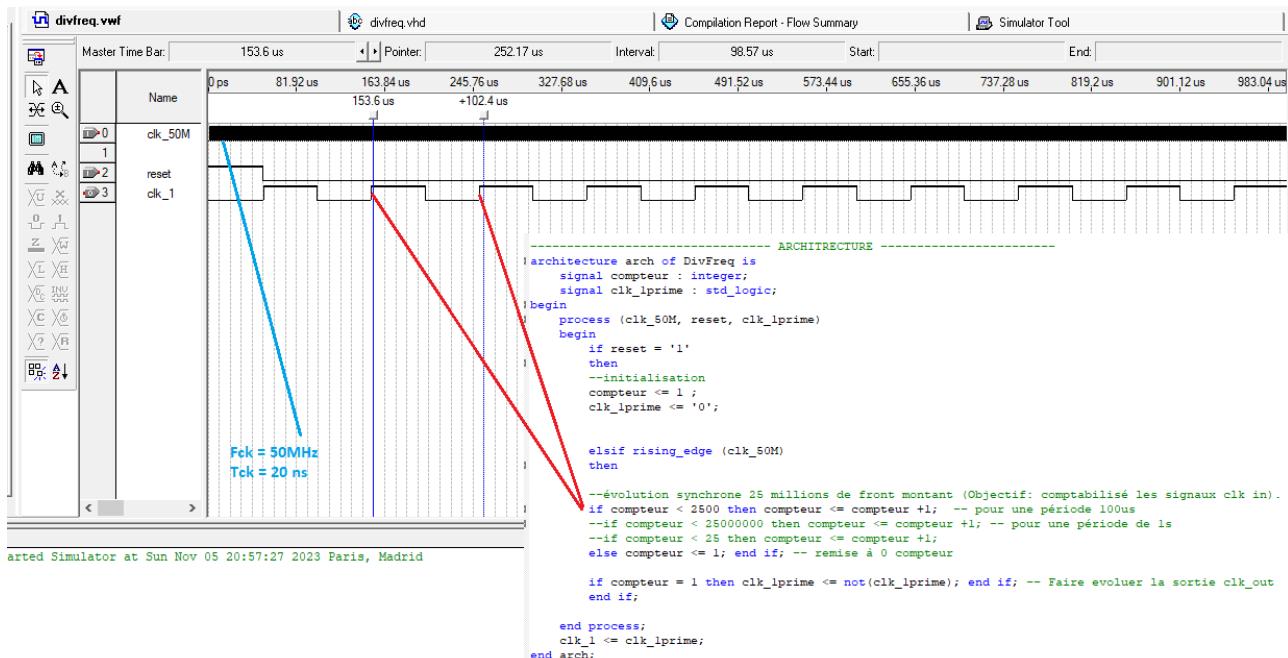


FIGURE 2.6 – Simulation diviseur de fréquence

2.4.2 Bloc 2 : Compteur/Décompteur BCD

Dans cette partie, le système effectue un compteur BCD (ou Décimal codé binaire). Le compteur va compter de 0 à 9 et il revient à la valeur 0 après avoir compté jusqu'à 9 et cela s'effectue en boucle. Le décompteur effectue une décrementation de valeur en chargeant la valeur 9 à l'initialisation et elle décremente jusqu'à 0. Le changement de valeur se fait toutes les 1 seconde donc on utilise le signal 1 Hz en sortie du diviseur de fréquence comme signal d'horloge du compteur décompteur BCD. A chaque front montant de l'horloge, il y a une incrémentation. On réinitialise après 9 seconde.

Pour réaliser ces fonction, nous avions tout d'abord commencer par le compteur bcd simple qui compte de 0 à 9. Puis nous avions réalisé le compteur décompteur bcd où un signal booléen en entrée varie en mode compteur ou décompteur. Enfin, il y a le comteur décompteur avec une entrée chargement de valeur.

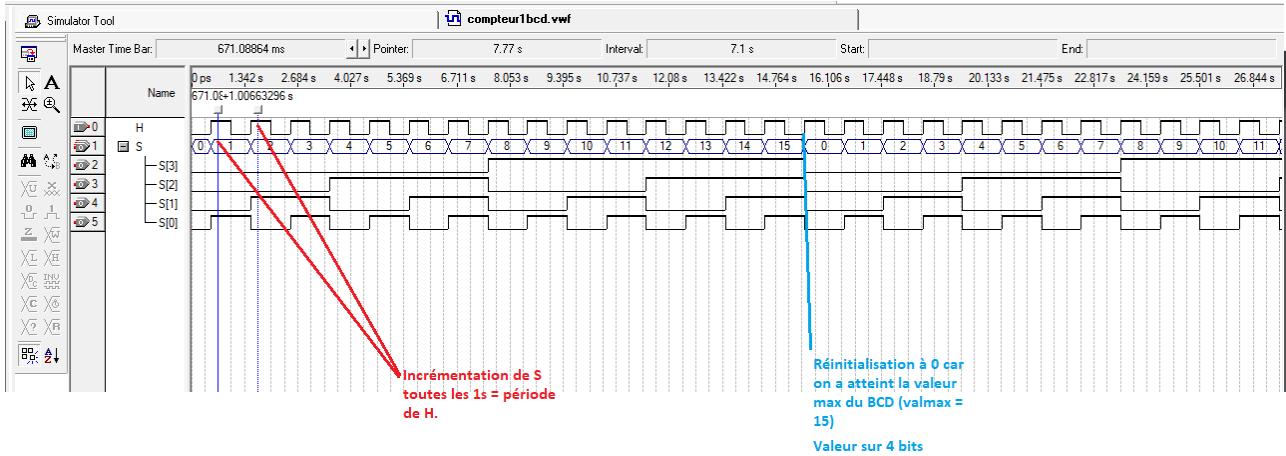


FIGURE 2.7 – Simulation comptage BCD simple

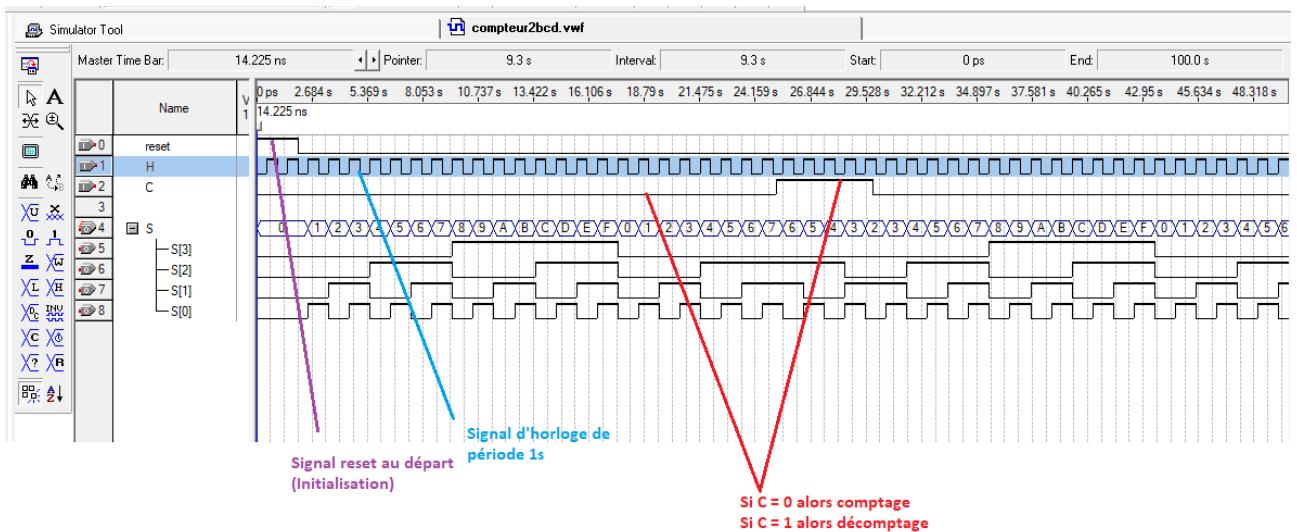


FIGURE 2.8 – Simulation comptage/décomptage BCD

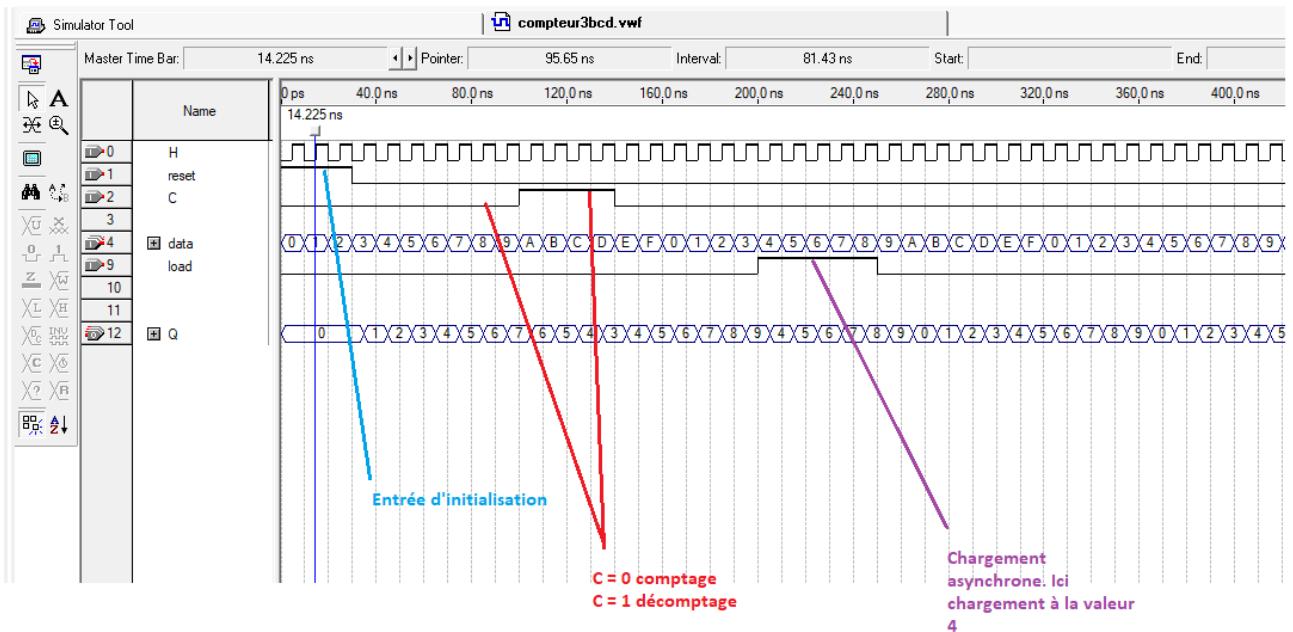


FIGURE 2.9 – Simulation comptage/décomptage BCD avec chargement asynchrone

2.5 Bloc 3 : Décodeur 7 segments :

Pour avoir l'affichage des secondes sur un afficheur 7 segments, nous avons converti ces valeurs par une fonction décodeur 7 segments. En effet, l'afficheur 7 segments affichera une valeur suivant la valeur en sortie du compteur ou décompteur. Pour comprendre le fonctionnement du décodeur 7 segments, l'objectif est d'avoir une combinaison de 7 bits en sortie à partir d'une valeur binaire sur 4 bits en entrée. Chaque segment est piloté par une broche. Chaque valeur en sortir du Compteur BCD est codée sur 4 bits et ces valeurs en binaire sont associées à une combinaison des valeurs des segments A à G en bit. La mise à 1 de ces bits allume donc les leds. Nous pouvons prendre le cas de la valeur 8 qui correspond à "1000" en binaire, on affiche toutes les leds pour avoir 8 en décimal donc la combinaison de bits des segment sera "1111111". Voici un schéma qui récapitule ces combinaisons en sortie.

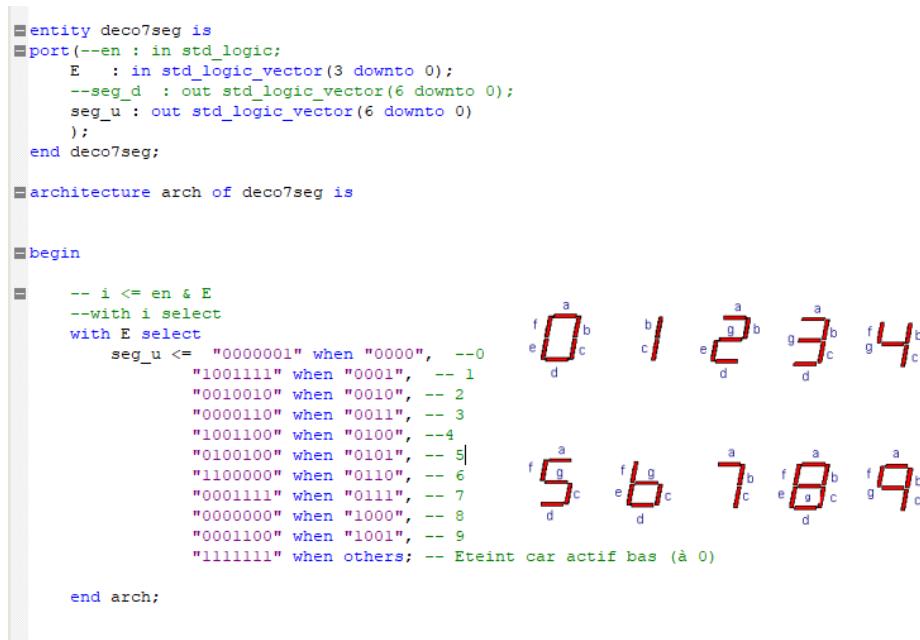


FIGURE 2.10 – Décodage 7 segments

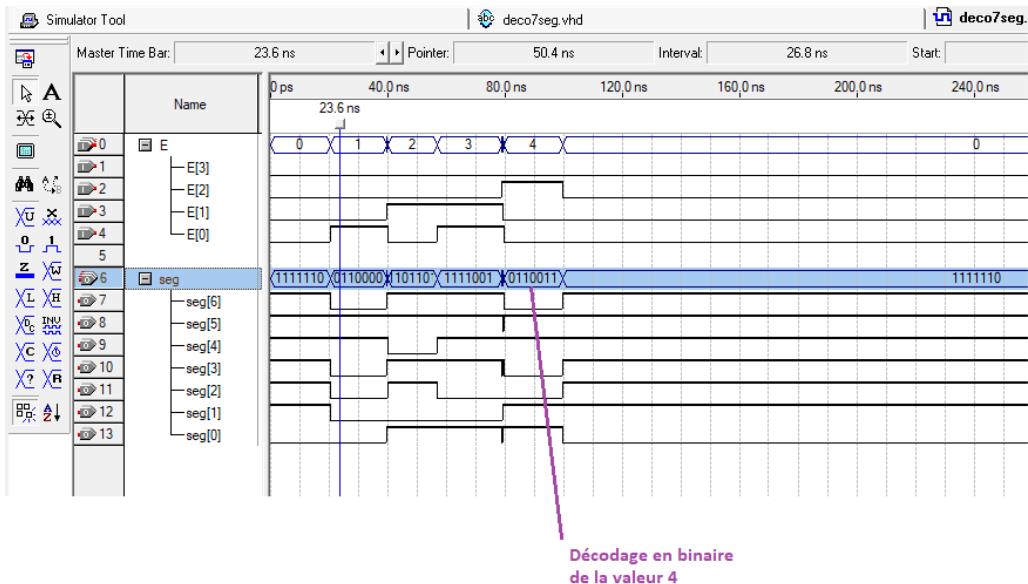


FIGURE 2.11 – Simulation décodage 7 segments

2.6 Simulation et test sur la carte du mini projet

Après avoir testé séparément chaque bloc de fonction du mini projet, nous avons effectué une simulation et un test où l'on relie chaque bloc en cascade suivant le cablage comme suit,

Lors de la simulation, on crée un signal d'horloge à 50 Mhz. Donc les signaux d'entrée seront le signal d'horloge et un signal pour indiquer la réinitialisation. En sortie, nous avions les signaux en sortie du décodeur 7 segments pour piloter un afficheur. Ici, on commande uniquement l'afficheur pour l'unité donc une incrémentation de 0 à 9 puis la réinitialisation. Voici le résultat de la simulation.

Après la simulation, nous avons testé le fonctionnement sur la carte DE2. Lors de l'affectation des broches de la carte avec les signaux d'entrée, nous avons utilisé le signal d'horloge du quartz à 50 MHz et un bouton de poussoir pour la réinitialisation. Les broches de l'afficheur 7 segments sont reliés aux signaux de sortie.

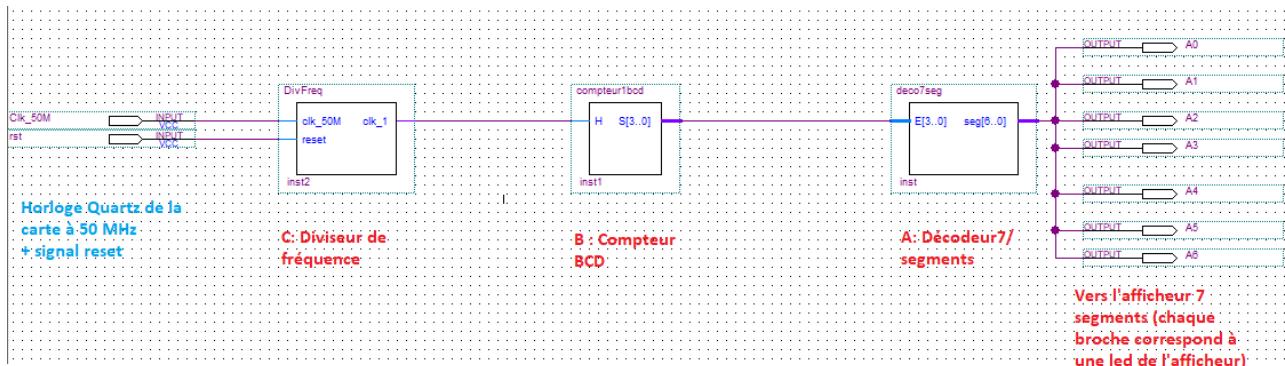


FIGURE 2.12 – Assemblage et cablage des blocs du mini projet

2.7 Génération d'une PWM :

Un signal PWM ou Pulse Width Modulation ou modulation à largeur d'impulsion constitue une fonction qui permet de générer un signal de rapport cyclique variable : la durée de l'état haut du signal sur la période varie : $R = T_{on}/\text{période}$. On utilise la fonction PWM pour commander la vitesse du vent dans la partie BE.

La réalisation du "PWM consiste à utiliser deux fonctions : un compteur libre sur N bits qui compte le temps à partir d'une horloge de référence clk et un comparateur pour déterminer d'une part la fin de la période et d'autre part la fin de la durée de l'état haut. Ce comparateur génère le signal pwmout en fonction de la

comparaison de la sortie du compteur avec le rapport cyclique. La fonction PWM s'effectue donc en deux temps. Voici une description fonctionnelle du PWM :

Une entrée RazN permet une remise à zéro asynchrone.

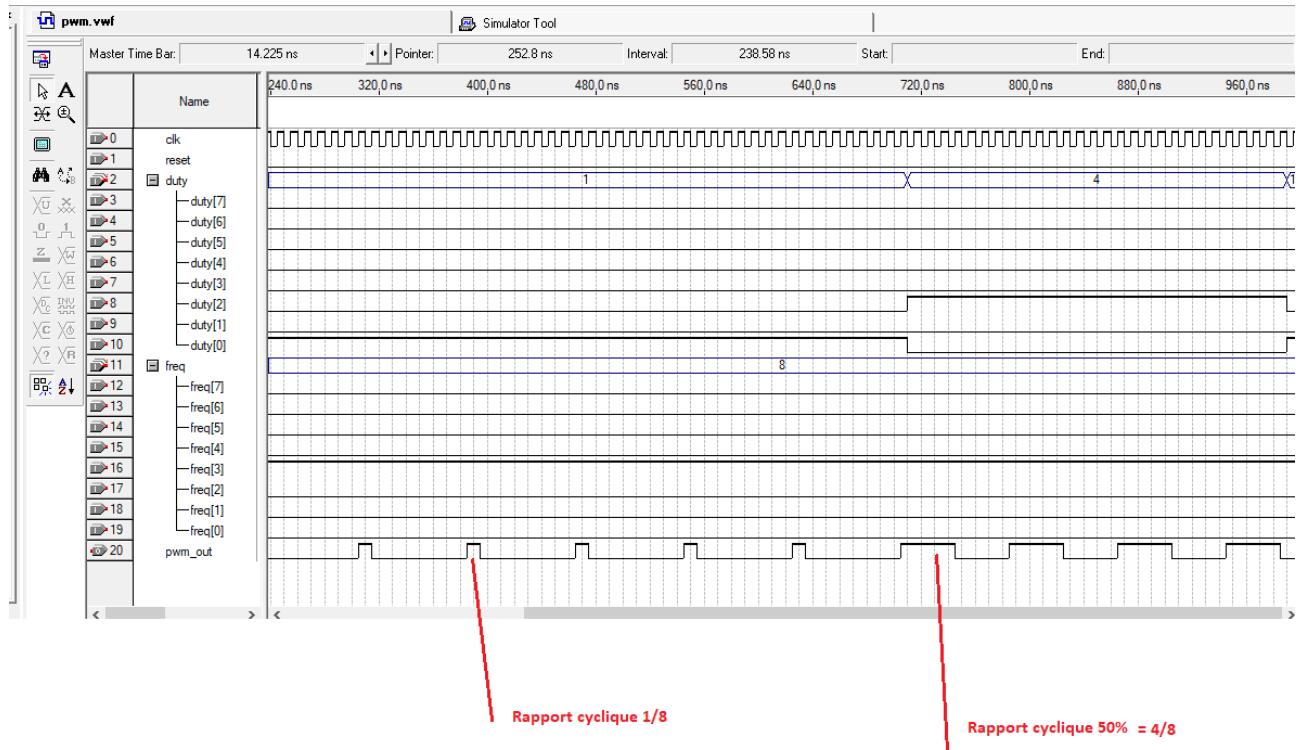


FIGURE 2.13 – Simulation PWM

CHAPITRE 3

BE : PILOTE DE BARRE FRANCHE POUR VOILIERS

Dans ce chapitre, nous allons effectuer la synthèse et mise en oeuvre de système. Il s'agit de manipuler des FPGA en langage VHDL. La mise en oeuvre d'un système qui répond au besoin de contrôle de trajectoire d'un voilier de barre franche nous incite à réaliser ce projet. Le cahier des charges comporte les exigences générales du projet. Ainsi le système doit être capable de :

- obtenir les informations de trajectoire réelles via une boussole et de récupérer les consignes appliquée à la trajectoire à boutons de pousoirs.
- acquérir les informations relatives à la vitesse du vent par l'anémomètre.
- interagir avec l'utilisateur pour voir l'évolution de l'état du système.
- récupérer l'itinéraire indiqué par le GPS et transmettre la position de la barre à travers un terminal NMEA à distance.
- actionner un vérin qui est utilisé pour changer l'angle de la barre franche et donc le change de direction.

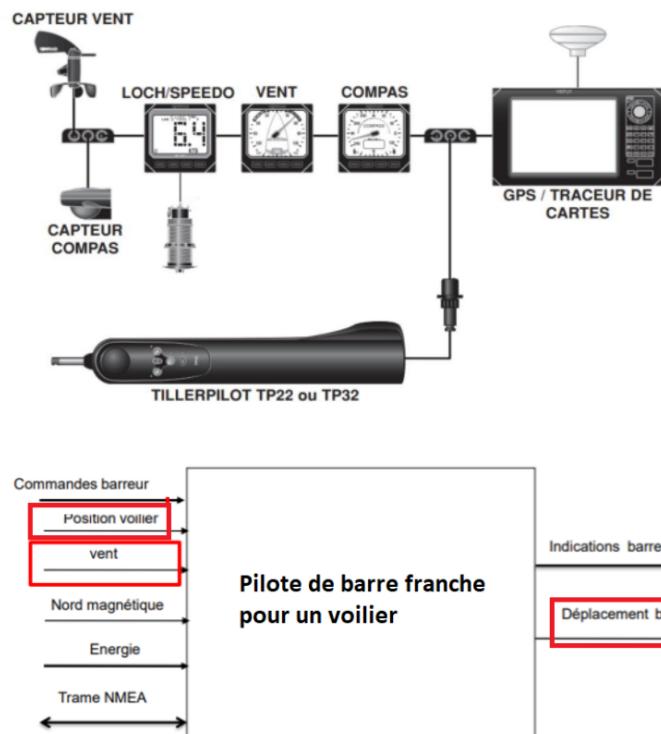


FIGURE 3.1 – Système de pilote de la barre franche

Au vu de la complexité du système et des séances limitées pour ce Bureau d'étude, nous avons choisi d'étudier l'anémomètre (fonction simple) et le vérin (fonction complexe).

Les étapes de développement s'effectuent en deux parties : la première partie concerne le développement matériel avec le quartus 18. Puis, la deuxième partie nous amène au développement logiciel avec le NIOS II en y intégrant les interfaces Avalon SOPC. Dans ce rapport, les étapes d'analyse fonctionnelle et les résultats obtenus sont expliqués.

3.1 Fonction simple : Anémomètre

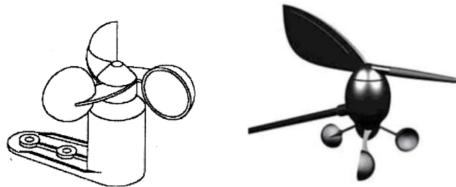


FIGURE 3.2 – Anémomètre coiffé de sa roue à aube

L'anémomètre permet de mesurer et convertir la vitesse du vent en un signal impulsional ayant une fréquence proportionnelle à cette vitesse. Il s'agit ici de traiter le signal issu de l'anémomètre. Pour connaître la valeur de la vitesse du vent, on effectue un comptage du nombre d'impulsion d'un signal d'acquisition (freq in) de l'anémomètre chaque 1 seconde d'horloge. Selon le cahier des charges, la fréquence est comprise entre 0 Hz et 250 Hz. Par exemple, si la fréquence est à 100 Hz, on aura 100 impulsions pendant 1 seconde.

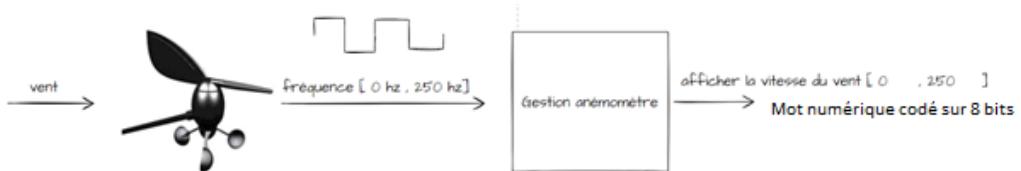


FIGURE 3.3 – Principe de fonctionnement de la gestion anémomètre

On aura en sortie une donnée (data_anemometre) codée sur 8 bits qui est le résultat du comptage de la fréquence du signal d'entrée. Ce module s'opère sur deux modes de fonctionnements : - le mode continu où une donnée sera mise à jour en sortie toutes les 1 secondes. - le mode mono-coup où la donnée peut être activée ou désactivée par un signal start_stop.

Voici la spécification de la gestion anémomètre :

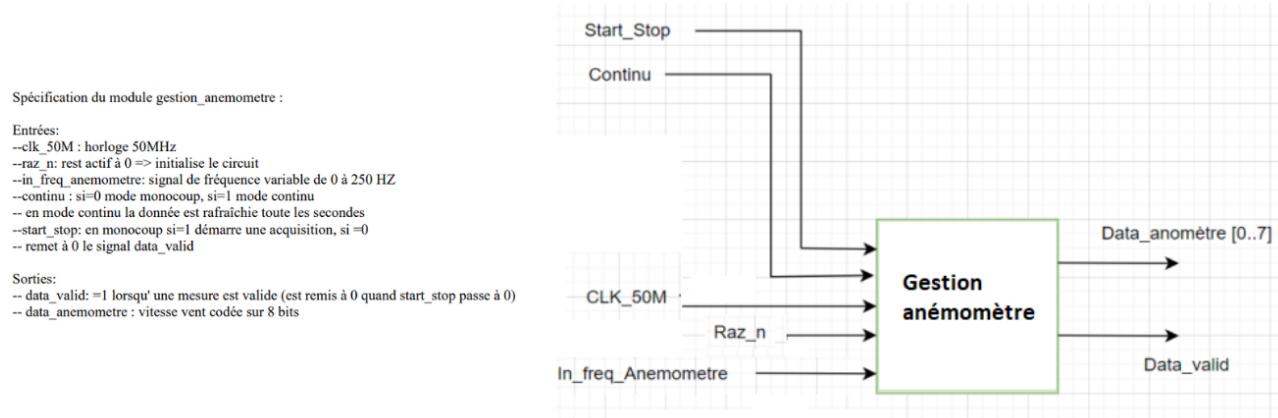


FIGURE 3.4 – Spécification et vue externe de la gestion anémomètre

3.1.1 Analyse fonctionnelle

Pour traiter le composant qui permet de relier les entrées aux sorties, nous avons découpés le composants en plusieurs blocs. Voici une vue simplifiée des blocs fonctionnels :

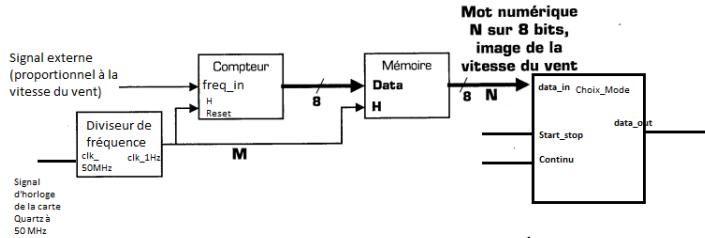


FIGURE 3.5 – Vue externe de l'anémomètre

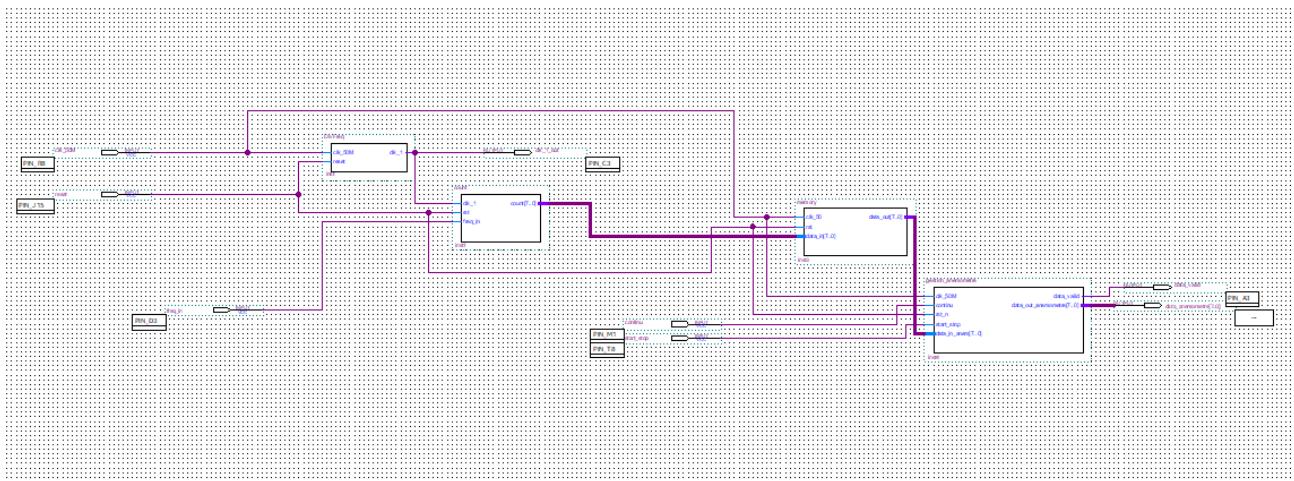


FIGURE 3.6 – Schéma diagramme des blocs fonctionnels de la gestion anémomètre sur Quartus 18

3.1.1.1 Génération du signal 1hz

L'objectif du diviseur de fréquence est de créer un signal d'horloge de 1 Hz à partir du signal d'horloge de 50 MHz fourni par l'horloge FPGA. Ce signal d'horloge de 1 Hz sera ensuite utilisé pour mesurer la fréquence de l'anémomètre une fois par seconde et actualiser les données en sortie de l'anémomètre toutes les secondes. En analysant le code VHDL de la division de fréquence, le signal 50MHz comporte 25000000 fronts montant pendant la demi-période d'un signal 1 Hz. Donc nous avons exploité cette observation pour mettre en place un compteur qui va s'incrémenter. Lorsqu'il atteind la valeur 24999999, il va se réinitialiser et déclenche le basculement ou le changement d'état du signal 1 Hz en sortie : allant de l'état haut à l'état bas ou l'inverse selon l'état précédent.

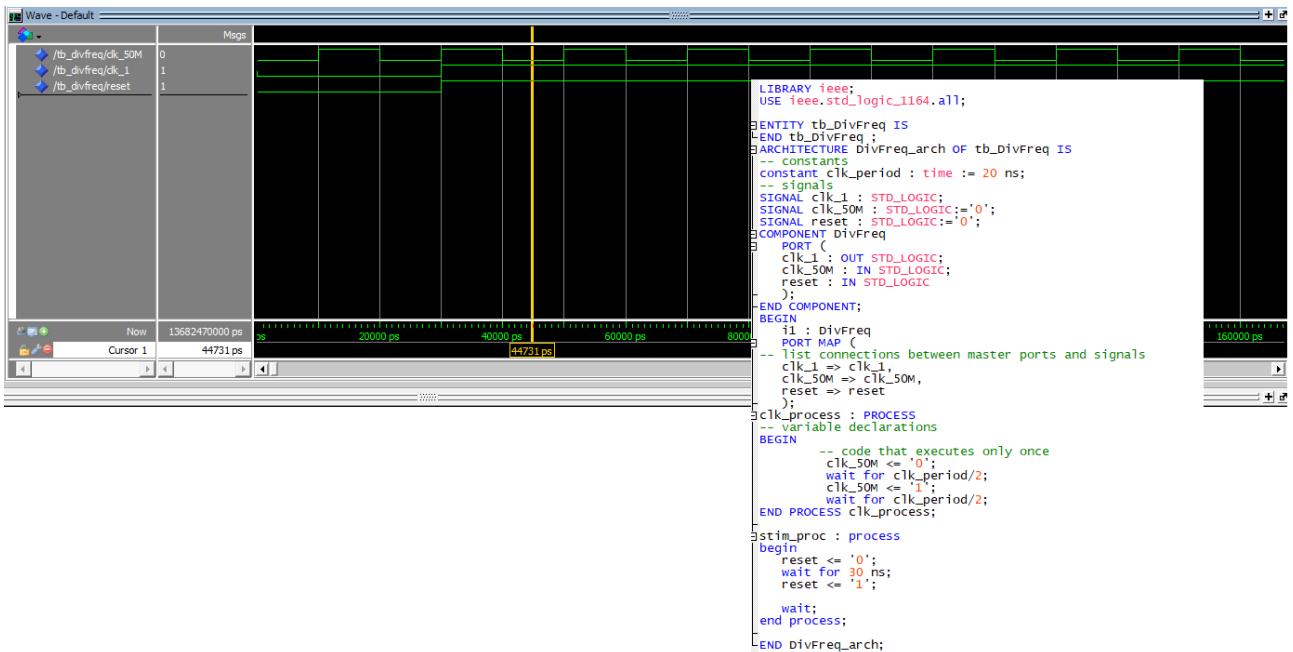


FIGURE 3.7 – Zoom sur le signal d’horloge 50 MHz et le test bench

Voici le résultat obtenu avec l’oscilloscope :

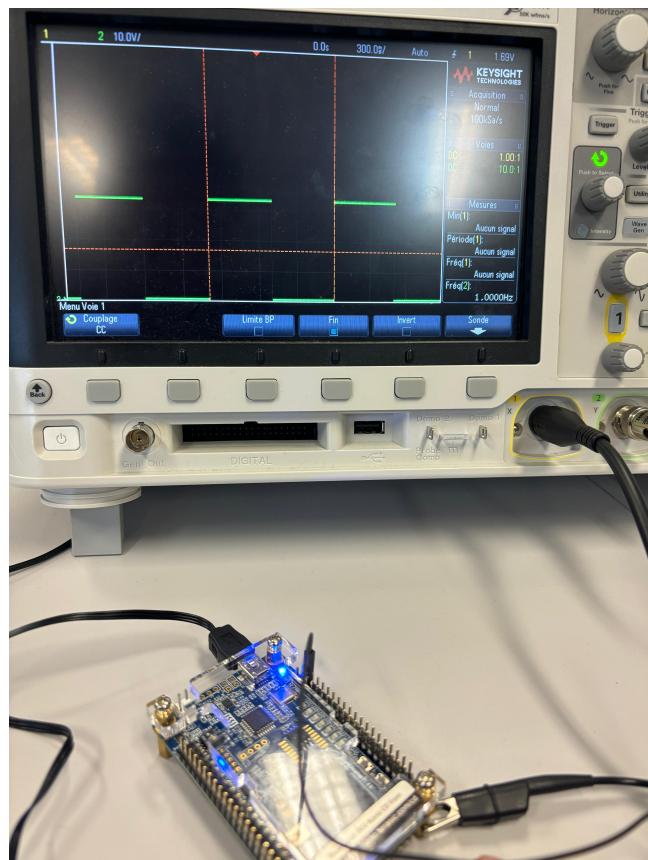


FIGURE 3.8 – Diviseur de fréquence 50 MHz à 1Hz

3.1.1.2 DéTECTEUR et compteURS de frontS montants de l'horloge 1 Hz :

Ce bloc fonctionnel est conçu pour identifier et compter les fronts montants du signal d'entrée de l'anémomètre freq in dans une fenêtre de mesure. En effet, la fenêtre de mesure est le signal à 1Hz donc pendant le temps de montée de ce signal ou sa première demi-période, un compteur va s'incrémenter à chaque front montant du signal freq in. En analysant le code VHDL de cette fonction, nous pouvons partir sur le même principe que le diviseur de fréquence où un compteur s'incrémentera lorsqu'on a un front montant du signal freq in et qu'on est à l'état haut du signal 1 Hz. Ainsi une variable intermédiaire va mémoriser la dernière valeur du compteur lorsque le signal 1 Hz bascule à l'état bas et on réinitialise le compteur en même temps. La valeur mémorisée correspond à la moitié de la valeur de la fréquence du signal freq in. En addition la variable intermédiaire par elle-même, on obtient ainsi la valeur de la fréquence qui correspond à la valeur en sortie de l'anémomètre et donc à l'image de la vitesse anémomètre. En effet, le signal de sortie a un rapport cyclique de 1/2. Si aucun front montant n'est détecté ou on a le signal reset qui est actif, l'état de la sortie de donnée anémomètre est réinitialisé à 0 ainsi que le compteur.

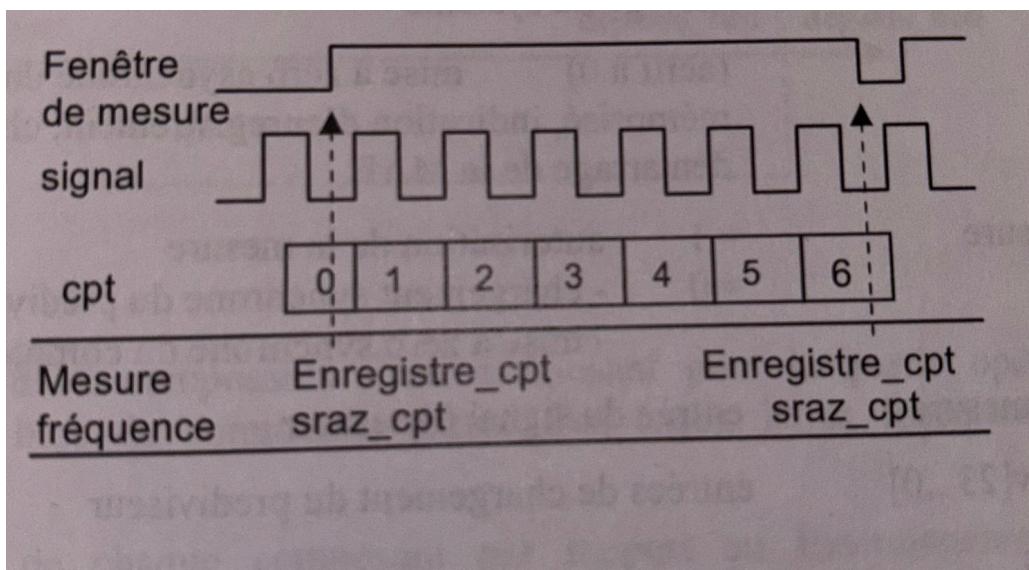


FIGURE 3.9 – Principe du détecteur et compteur de front montant d'un signal d'entrée dans la fenêtre de mesure de l'horloge à 1 Hz dans notre cas

Cette opération est synchronisée avec le signal de 1 Hz du diviseur de fréquence. Par conséquent, durant la fenêtre de mesure de la demi-période du signal à 1 Hz (où la demi-période est à 500 ms). La fin du comptage est fait lorsqu'on détecte un front descendant de l'horloge à 1 Hz. On génère un signal sur un GPIO de la carte où sa fréquence est déterminée par l'utilisation soit par le GBF (Générateur Basse fréquence) ou d'autres outils comme le signal PWM de l'arduino) D'après la simulation qu'on obtient ci-dessous, lorsqu'on compte le nombre total de front montant du signal freq in pendant l'état haut du signal d'horloge 1Hz, cette valeur correspond à la moitié de la valeur en décimal de data anémomètre. Ce résultat confirme ainsi notre analyse par rapport à la synchronisation du signal 1 Hz et le signal freq in.

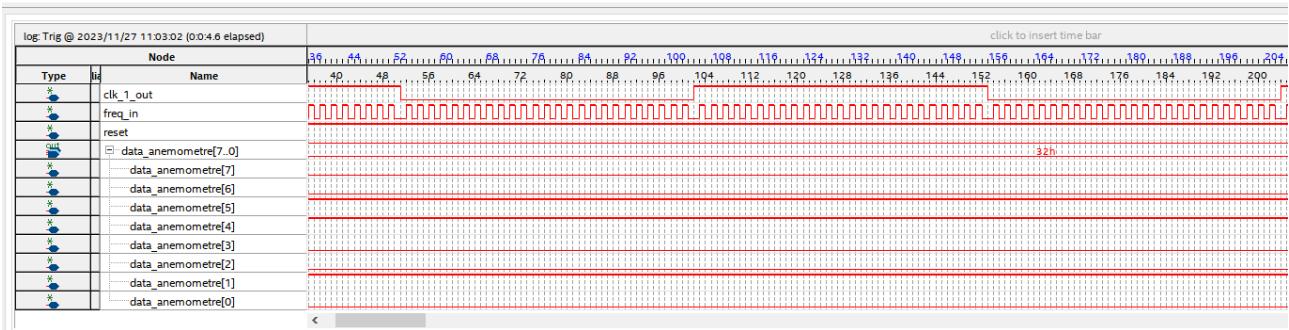


FIGURE 3.10 – Exemple de simulation de comptage sur signal Tap Logic Analyser. On compte 25 fronts montants pendant l'état haut de clk1out et la valeur 32h correspond à 50 en décimal.

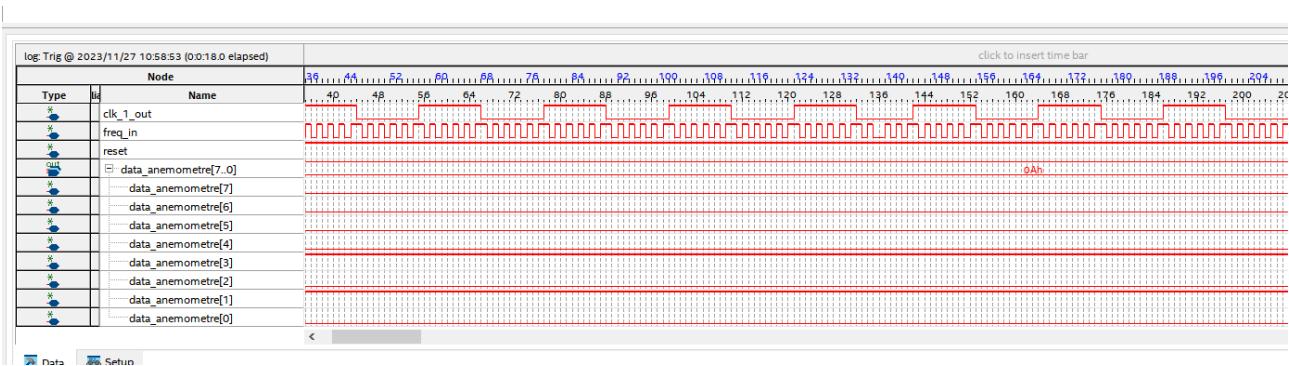


FIGURE 3.11 – Exemple de simulation de comptage sur signal Tap Logic Analyser. On compte 5 fronts montants pendant l'état haut de clk1out et la valeur 0Ah correspond à 10 en décimal

3.1.1.3 Mémorisation d'événement

Il faut mettre à jour le comptage lorsqu'on détecte un front montant de l'horloge 1Hz et mémoriser ainsi la valeur. Nous avons créer un bloc fonctionnel qui assure la mémorisation d'une donnée binaire en entrée. Il s'agit du résultat du comptage. Cela permet d'avoir un rafraîchissement de données toutes les secondes. L'unité de comptage est ici le clk 1Hz donc 1 seconde de période. Après avoir compté le nombre de fronts montant survenu pendant cette unité de temps, nous allons utiliser le principe de registre dans un process pour mémoriser le nombre de fronts comptés à la fin de la mesure, c'est à dire à chaque changement en front montant de clk 1Hz.

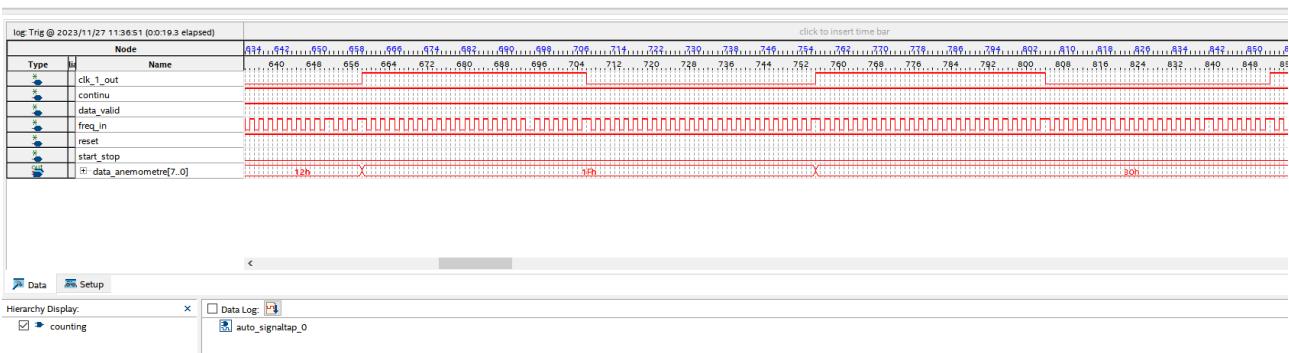


FIGURE 3.12 – Gestion de mémorisation de la valeur binaire de la fréquence en sortie

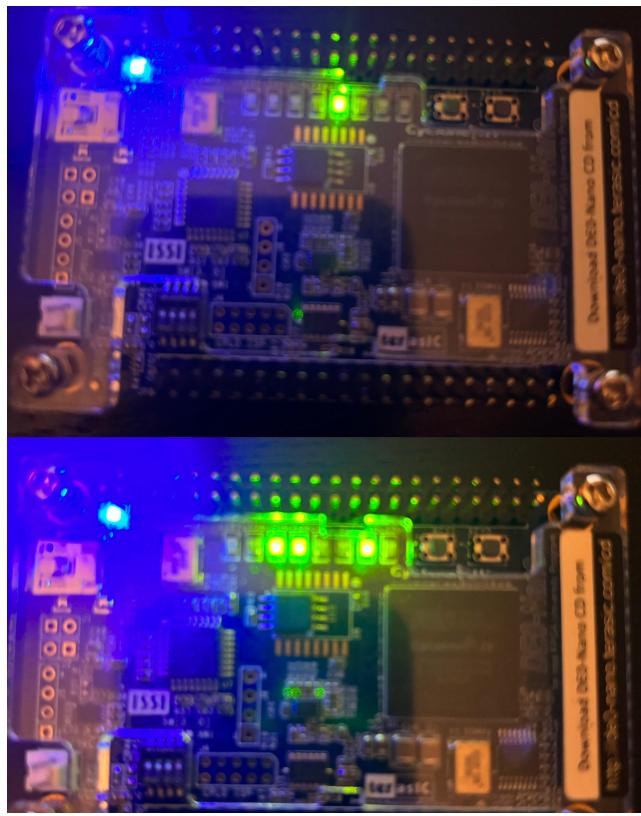


FIGURE 3.13 – Valeurs binaires de la fréquence Freq in du signal d’entrée 8 Hz et 50 Hz

3.1.1.4 Le choix de mode

En mode continu, le module gestion anémomètre met à jour régulièrement les données de vitesse du vent toutes les secondes. En mode monocoupe, le module démarre une acquisition avec le signal start stop et valide la mesure avec data valid. Une fois start stop revenu à '0', data valid est également remis à '0', indiquant la fin de la mesure. Le signal de réinitialisation raz n permet de réinitialiser le circuit à tout moment.

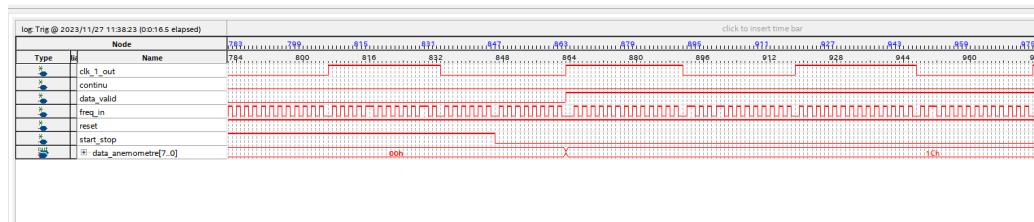


FIGURE 3.14 – Acquisition de donnée anémomètre par l’activation de data valide

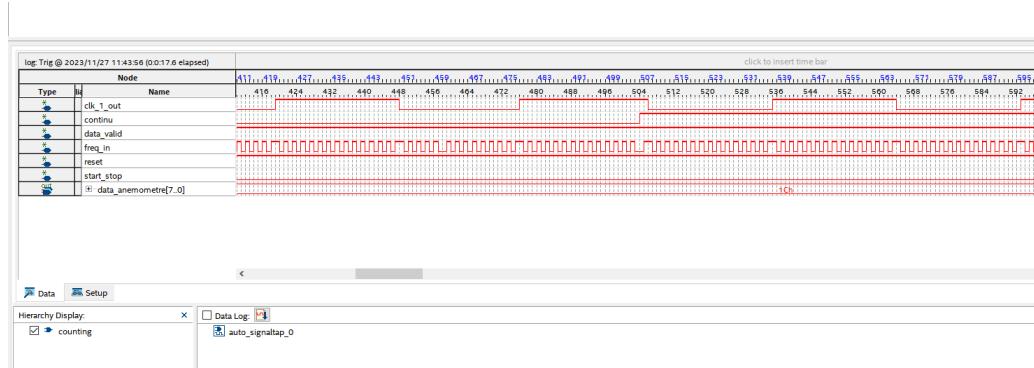


FIGURE 3.15 – Donnée anémomètre en continu

3.1.2 Configuration sur SOPC : System On Programmable Chip

3.1.2.1 Implémentation et conception du Platform Designer sur projet

L'objectif du SOPC ou System On Programmable Chip est de pouvoir intégrer les composants matériels et logiciels sur une puce unique ou un circuit programmable. Cela permet ainsi d'associer les périphériques matériels(PIO, Timers, UART, JTAG...) et la mémoire de la carte FPGA grâce à la partie logiciel intégré dans le processeur de la carte où la plateforme sera le SoftCore NIOS II. Pour cette première partie, le test de cette configuration s'effectue par la mise en place de la commande de bouton de poussoir et des leds qui sont intégrés dans le Platform Designer où l'on alloue une case mémoire pour tous les périphériques exploités dans cette première partie.

3.1.2.2 Affichage Hello Word et Bouton Poussoir

Nous avons donc ajouté des PIO dans le platform Designer pour l'intégration des LED et des boutons de la carte FPGA dans le SOPC. Voici le circuit du plateforme designer par rapport à cela. L'affichage de "Hello World!" sur le terminal du NIOS II est géré par le périphérique JTAG Uart. Après avoir lancé le NIOS II depuis le quartus prime 18 et en se plaçant dans le bon répertoire de travail, nous avons effectuer des configurations pour générer le BSP. (Board Support Package). Le BSP contient tous les drivers et logiciels pour la configuration matériel. On y trouve les informations concernant l'adresse mémoire attribuée aux commandes bouton poussoir, au pwm ou le signal utilisé pour l'anémomètre. Ce BSP fournit donc une couche d'abstraction pour le logiciel d'application où les noms des adresses sont appelés.

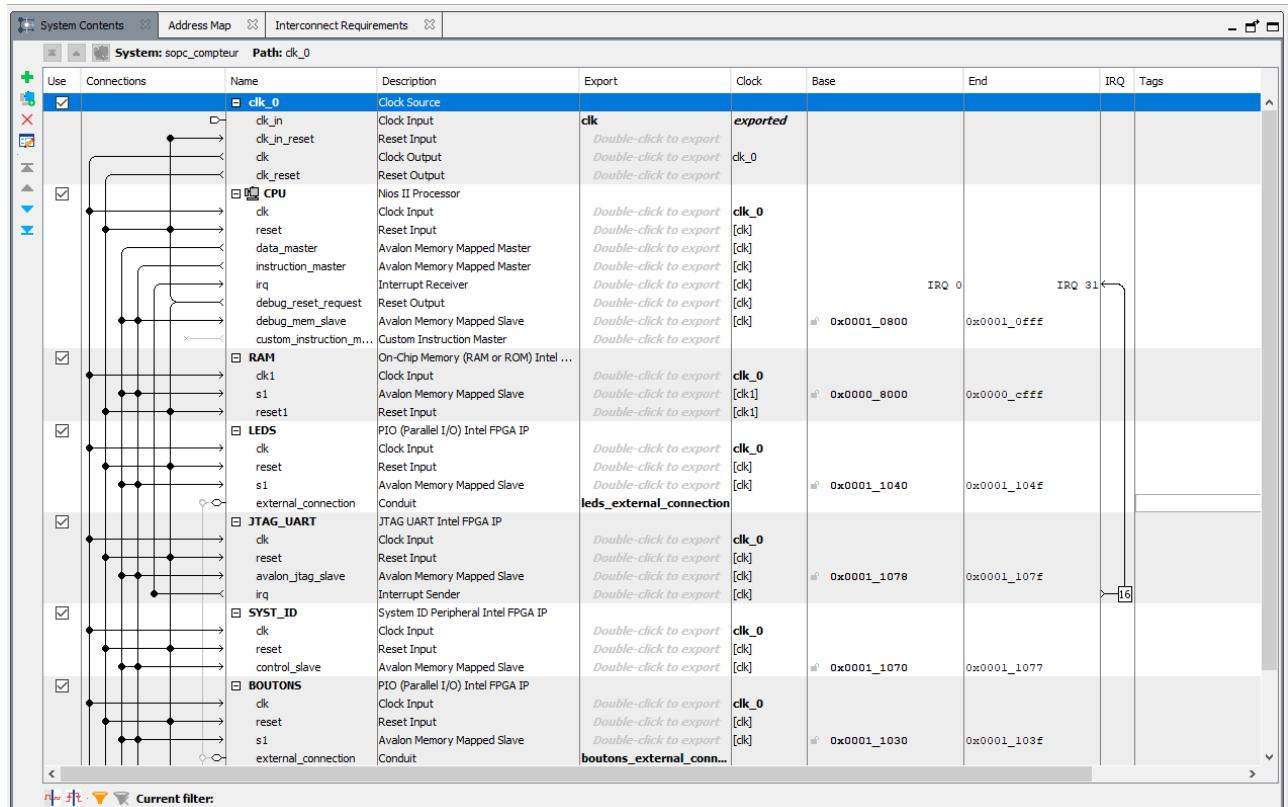


FIGURE 3.16 – Configuration SOPC pour les led, boutons poussoirs

Voici le résultat obtenu sur le terminal du NIOS II. Le terminal affiche en permanence la valeur 3 pour le bouton. Suivant l'appuie de l'utilisateur sur l'un des boutons, la valeur peut changer à 1 ou à 2.

The screenshot shows a software interface with two main windows. The top window is a code editor displaying C code for a project named 'essai projet leds boutons'. The code includes includes for stdio.h, system.h, and altera_avalon_pio_regs.h, defines for button and LED addresses, and a main loop with a delay. The bottom window is a terminal window titled 'Nios II Console' showing the output of the program. The output consists of multiple 'Hello world!' messages and values for 'boutons' (3, 2, 1) repeated in a loop.

```

1 // essai projet leds boutons
2
3 #include "sys/alt_stdio.h"
4 #include <stdio.h>
5 #include "system.h"
6 #include "altera_avalon_pio_regs.h" // pour éviter de renseigner les adresses physiques des périphériques
7 #include "unistd.h" // pour la fonction délai
8
9 #define boutons (volatile char *) BOUTONS_BASE
10 #define leds (unsigned int*) LEDS_BASE
11
12 #define freq (unsigned int *) AVALON_PWM_0_BASE
13 #define duty (unsigned int *) (AVALON_PWM_0_BASE + 4)
14 #define control (unsigned int *) (AVALON_PWM_0_BASE + 8)
15

Hello world!
boutons = 3
Hello world!
boutons = 2
Hello world!
boutons = 3
Hello world!
boutons = 3
Hello world!
boutons = 1
Hello world!
boutons = 1
Hello world!
boutons = 3
Hello world!
boutons = 2
Hello world!
boutons = 2
Hello world!
boutons = 3

```

FIGURE 3.17 – Affichage Hello World et les commandes boutons poussoirs

3.1.2.3 Génération du PWM

Un circuit générateur de modulation de largeur d’impulsion (PWM) utilise une technique qui permet de contrôler la quantité moyenne de puissance fournie à un dispositif électronique en modifiant la proportion du temps pendant lequel un signal est à un niveau élevé (état '1') ou à un niveau bas (état '0'). Selon le cahier des charges pour le Bureau d’études, le pwm est utilisé pour commander le moteur de notre système de pilote de la barre franche. Nous avons donc repris le code PWM dans la partie TP de base et nous avons intégrer ce bloc PWM dans le SOPC. Voici le schéma du plateform Designer mis à jour avec le schéma bloc fonctionnel de SOPC contenant la fonction avalonPWM, bouton de poussoir et leds.

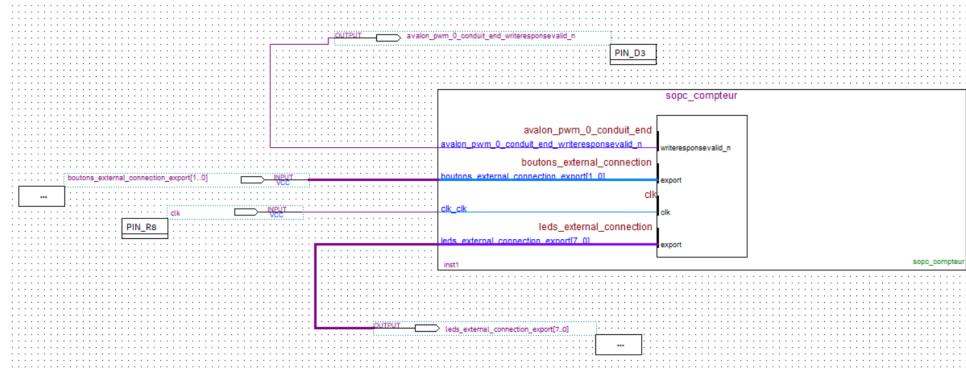
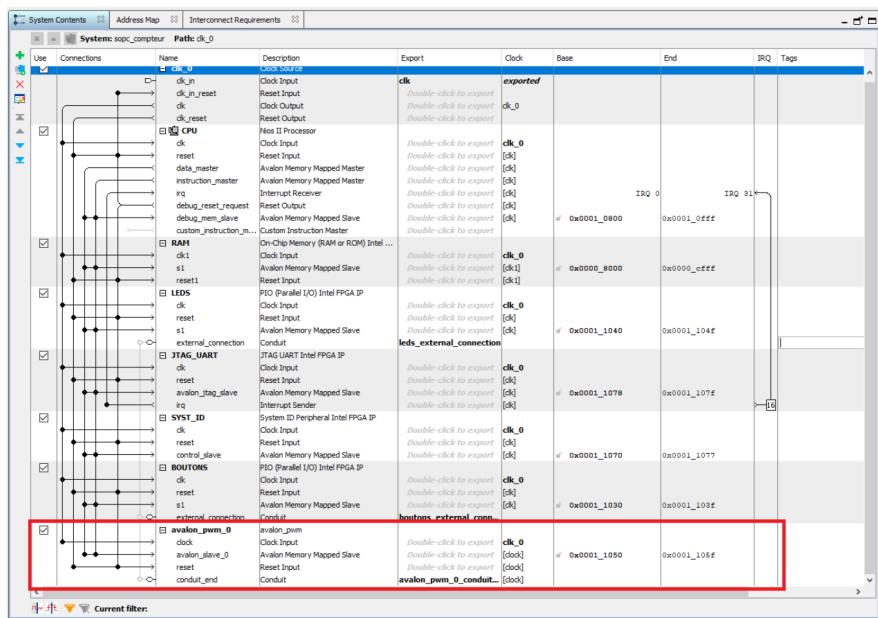


FIGURE 3.18 – Configuration SOPC pour la fonction PWM. La création de conduit end dans le plateforme Designer permet de faire sortir les broches utilisés comme entrées ou sortie du SOPC

Donc les signaux utilisés en entrée ou sortie du SOPC sont la sortie du signal pwm, l'entrée d'horloge 50 MHz, les signaux d'entrée pour la commande des boutons qui sont sur 2 bits et les signaux 8 bits pour la sortie des LEDs.

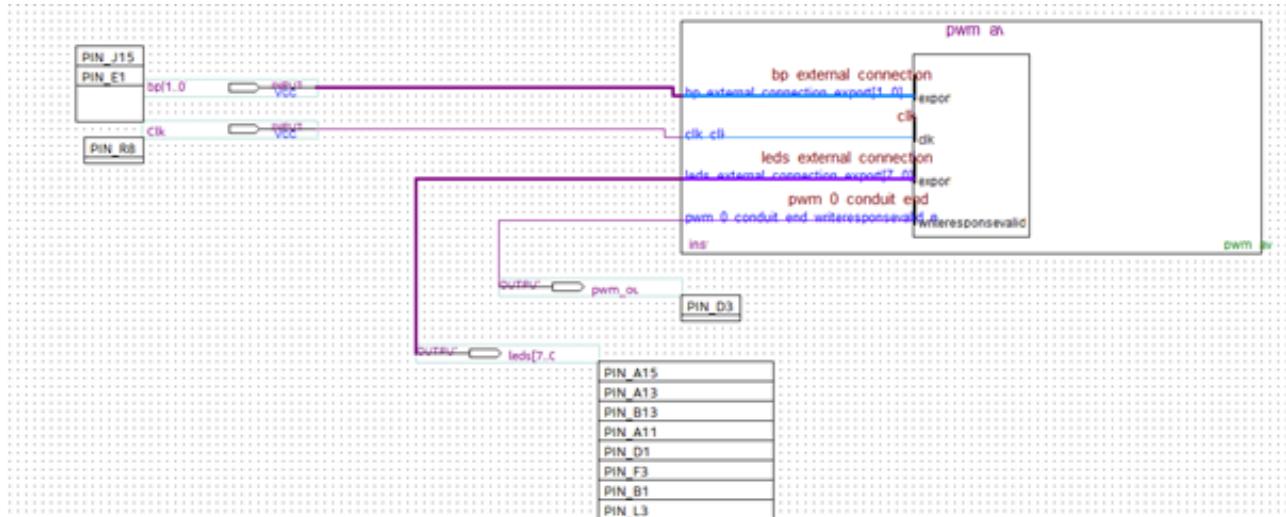


FIGURE 3.19 – Bloc SOPC PWM

3.1.3 Configuration sur SOPC pour l'avalon Anémomètre

Le but de cette partie est de réaliser le SOPC pour l'avalon anémomètre. En effet, dans la partie fonction anémomètre, nous avons pu visualiser la valeur de données en sortie sur les 8 leds de la carte car comme la valeur de la fréquence va de 0 à 250 Hz donc cette plage de valeurs peut être codée sur 8 bits. En intégrant l'anémomètre dans le SOPC, le processeur va se charger de lire et écrire les données de l'anémomètre dans la mémoire. Pour que l'utilisateur puisse voir ces échanges en lecture ou écriture, nous avons développé du code en C pour afficher sur le terminal du NIOS II la valeur de donnée anémomètre en fonction de la fréquence qu'on envoie dans la carte.

Voici le résultat obtenu :

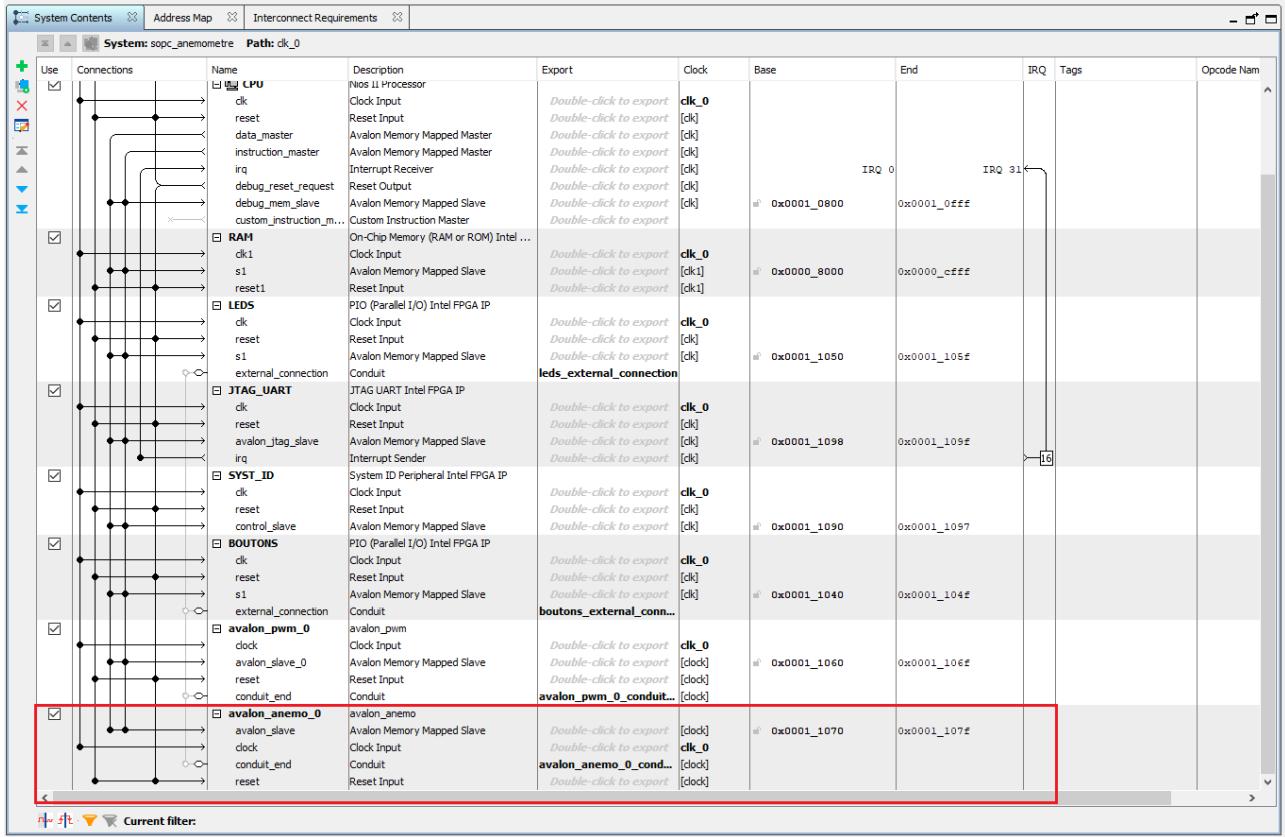


FIGURE 3.20 – Circuit sur platform Designer du SOPC avec l'avalon Anémomètre

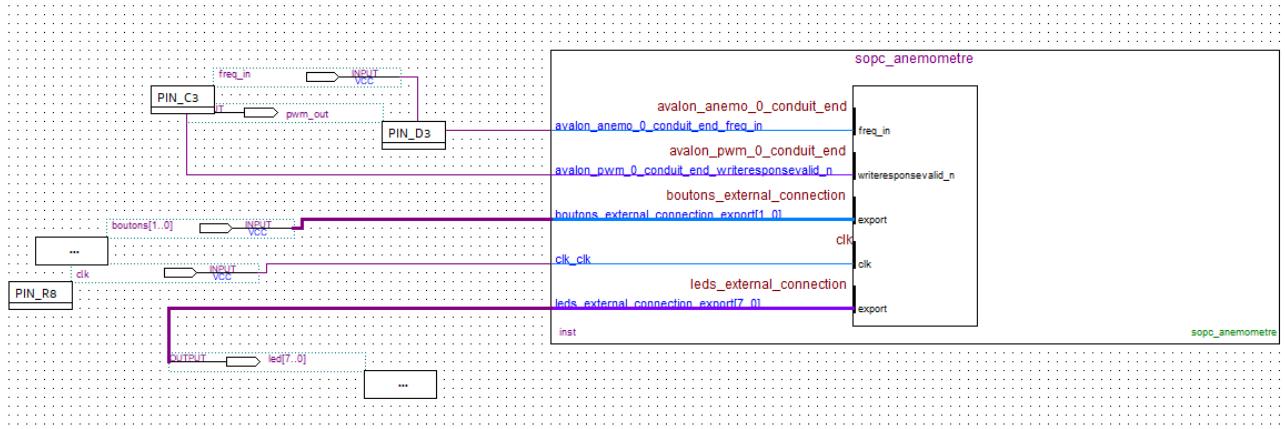


FIGURE 3.21 – Schéma du diagramme de bloc du SOPC intégrant l'avalon Anémomètre. Le signal freq in doit être un signal extérieur du SOPC car lié à une broche de la carte

Nous avons donc programmer la carte et relier le SOPC du plateforme designer à la partie processeur dans le NIOS II. Après avoir régénérer le BSP (Board Support Package) et compléter le programme pour l'affichage des boutons de poussoir et les données du anémomètre, nous avons compilé et lancer le programme dans le

terminal du NIOS. Voici le résultat obtenu en choisissant les fréquences 1 Hz, 100 Hz et 250 Hz comme en entrée du freq in :

FIGURE 3.22 – Affichage données anémomètre sur le terminale NIOS pour différentes valeurs de fréquence

3.2 Fonction complexe : le vérin

Le circuit vérin, utilisé pour actionner la barre franche du voilier, se compose de trois fonctions principales qui ensemble facilitent le pilotage du vérin contrôlant la barre franche .Pour cela nous devons développer quatre fonctions clés qui, combinées, permettront de piloter efficacement le vérin qui gère la barre franche. Ces fonctions sont :

- La fonction de Gestion des butées
- La fonction de Gestion du MCP3201
- La fonction de Gestion PWM
- La fonction d'Interface avec le bus Avalon.

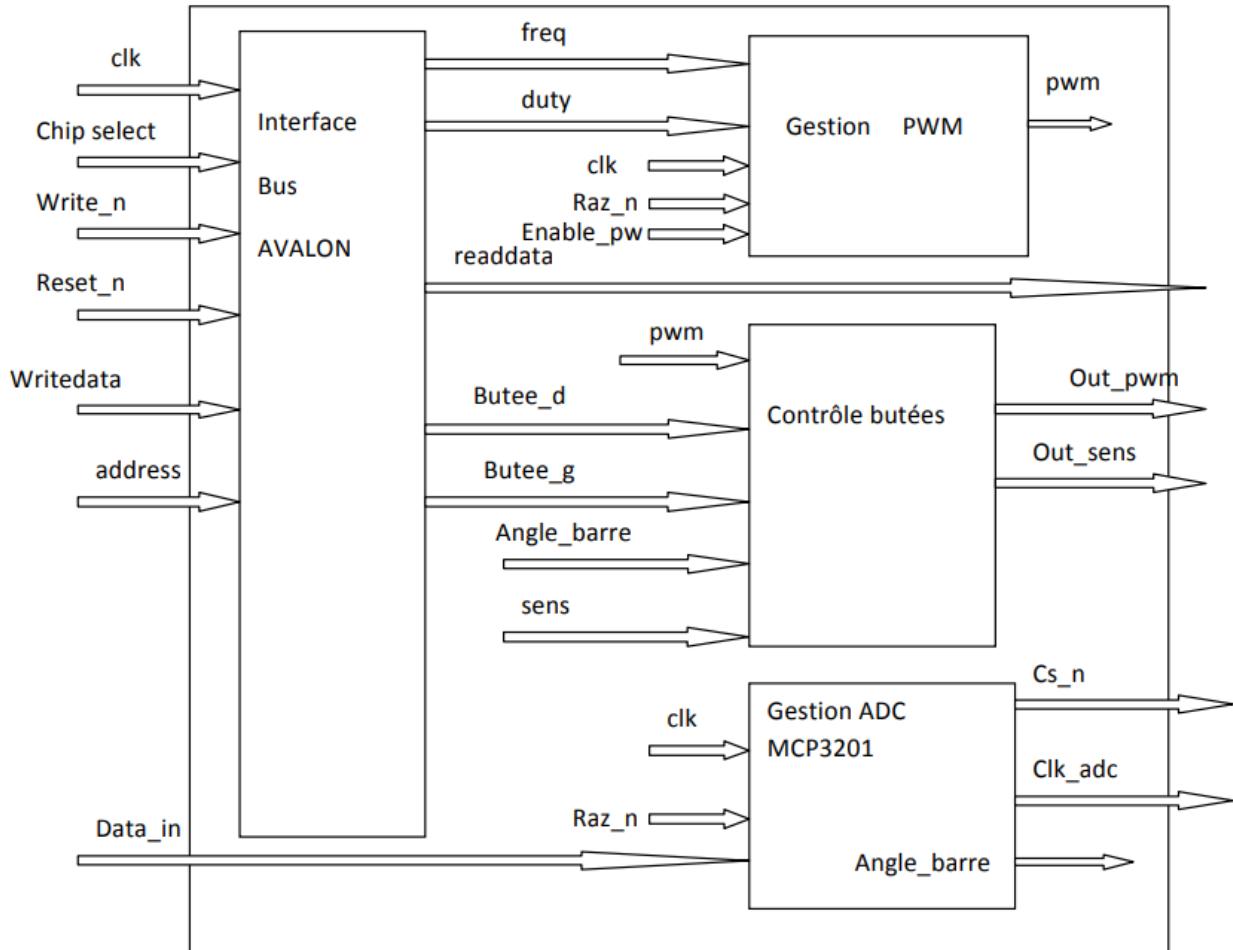


FIGURE 3.23 – Schéma fonctionnel du circuit gestion-verin

3.2.1 Gestion du PWM

L'objectif de cette fonction est de piloter le vérin. Son fonctionnement se déroule de la manière suivante : Un comparateur évalue si la valeur du compteur est inférieure à celle de l'entrée Duty. Si c'est le cas, le signal de sortie PWM est maintenu à '1'. Lorsque le compteur n'est plus inférieur à Duty, le signal PWM passe à '0'. Un autre comparateur a pour rôle de générer un signal de réinitialisation lorsque la valeur du compteur atteint celle du signal « freq ».

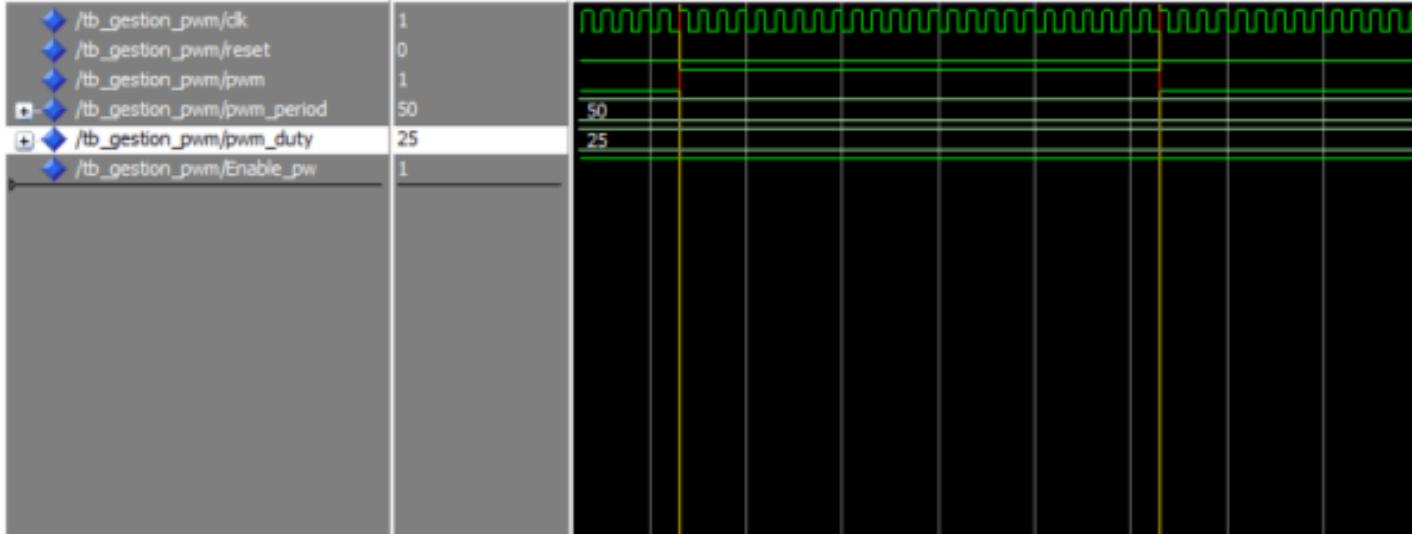


FIGURE 3.24 – Logigramme du PWM avec modelsim

3.2.2 Gestion des butées

La gestion des butées utilise la fonction principale 'controle-butées', qui :

-Positionne le signal 'PWM' à 0 si 'angle-barre' dépasse les limites définies par 'butee-g' et 'butee-d', en tenant compte du sens de rotation du moteur, et active les signaux 'fin-course-d' ou 'fin-course-g' en conséquence. Cela arrête le moteur lorsque l'angle de la barre est hors des limites des butées.

-En revanche, si 'angle-barre' se situe entre les deux butées, le signal 'PWM' est activé pour contrôler la vitesse du moteur, permettant ainsi le mouvement de la barre franche.

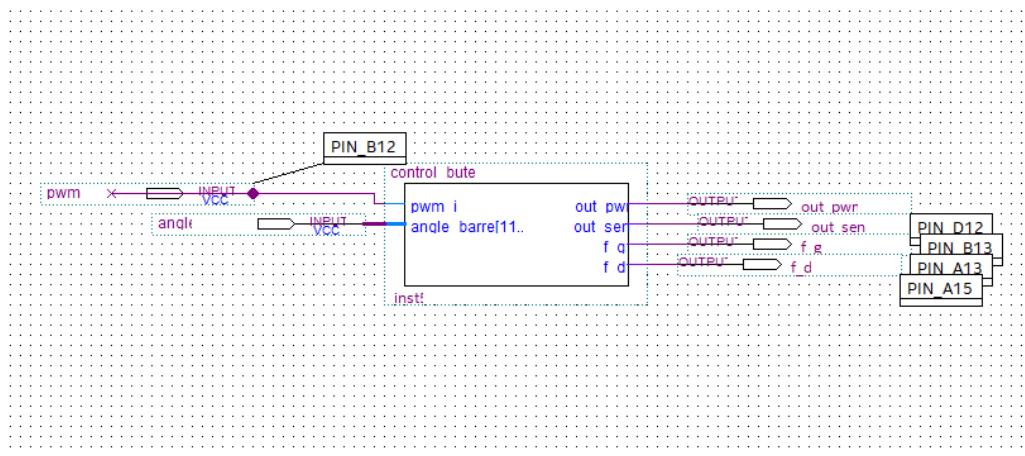


FIGURE 3.25 – schéma bloc du butées

==> Pour tester cette section, nous avons initialisé les valeurs de 'butee-g', 'butee-d' réglables de 0 à 4095 elles fixent les valeurs limites que le vérin (angle-barre) ne doit pas dépasser et le sens . En ajustant le potentiomètre, qui simule la valeur de l'angle, nous observons que si la valeur de l'angle est située entre butee-g et butee-d, alors PWM-out est égal à PWM-IN. Si ce n'est pas le cas, PWM-out est réglé sur 0. Cette configuration reflète le fonctionnement du moteur.



FIGURE 3.26 – Gestion des butées

3.2.3 Gestion du convertisseur AN MCP 3201

L'angle de la barre franche, étant une mesure analogique, nécessite l'intégration d'un convertisseur analogique-numérique (CAN) dans notre système. Nous avons opté pour le MCP3201, un CAN de 12 bits doté d'une interface SPI série. Par conséquent, il est essentiel de mettre en place une interface de communication SPI sur notre composant vénin.

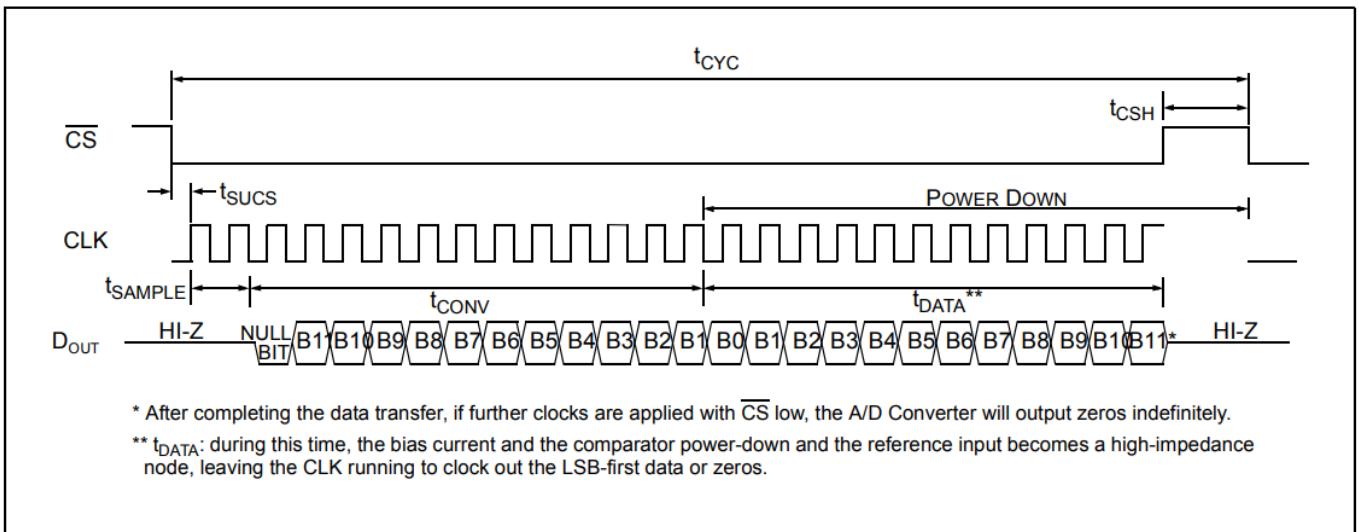


FIGURE 3.27 – Fonctionnement du convertisseur

- CS : un chip select. On doit le mettre à l'état bas pour activer le composant et démarrer une acquisition.
- CLK : l'horloge de référence du convertisseur que l'on doit établir à 1Mhz pour ce BE.
- Dout : la sortie du convertisseur qui sera relié au FPGA pour récupérer les données converties

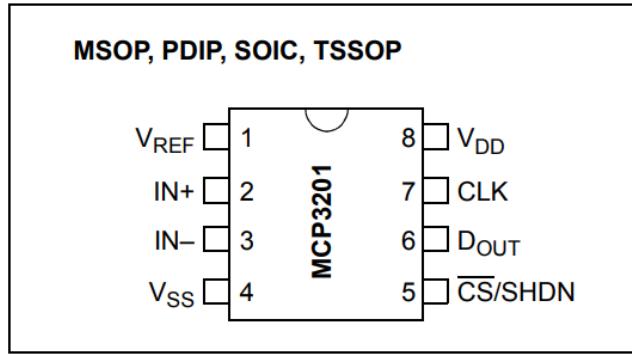


FIGURE 3.28 – Schéma de cablage de MCP3201

Ce convertisseur fait appel à 5 fonctions secondaires :

3.2.3.1 Machine à état

- Machine à état pour piloter l'adc et mémoriser la donnée « angle-barre » (process pilote-adc) : Le pilotage de la communication avec l'ADC, dépend des signaux « Start-conv » pour passer de l'état « IDLE » à l'état « Reading » (lecture des données de l'ADC) et du signal « Conv-state » (Cs-n) qui signale la fin de la conversion et le passage de l'état « Reading » à l'état « IDLE ».

3.2.3.2 Comptage des fronts d'horloge

- Comptage des fronts d'horloge de clk-adc (process compt-fronts)

3.2.3.3 Registre à décalage

- Registre à décalage pour récupérer la donnée du convertisseur (process rec-dec) :

Le MCP3201 envoie ses données en série. Ce processus utilise un registre à décalage pour convertir ces données série en un format parallèle, facilitant ainsi leur traitement par le système.

3.2.3.4 Génération du 1MHz

- Génération du 1MHz pour la machine à état (process gene-1M) :

Ce processus génère une horloge de 1 MHz à partir de notre bloc de diviseur de 50Mhz ,le 1Mhz est utilisé par la machine à état pour contrôler le timing du processus de conversion de l'ADC.

3.2.3.5 Génération périodique

- Génération périodique (toutes les 100ms) du signal « start-conv » (process gene-start-conv) :

Ce processus génère un signal "start-conv" toutes les 100 ms, indiquant au MCP3201 de commencer un nouveau cycle de conversion. Cette périodicité assure que les données de l'angle de la barre franche sont mises à jour régulièrement.

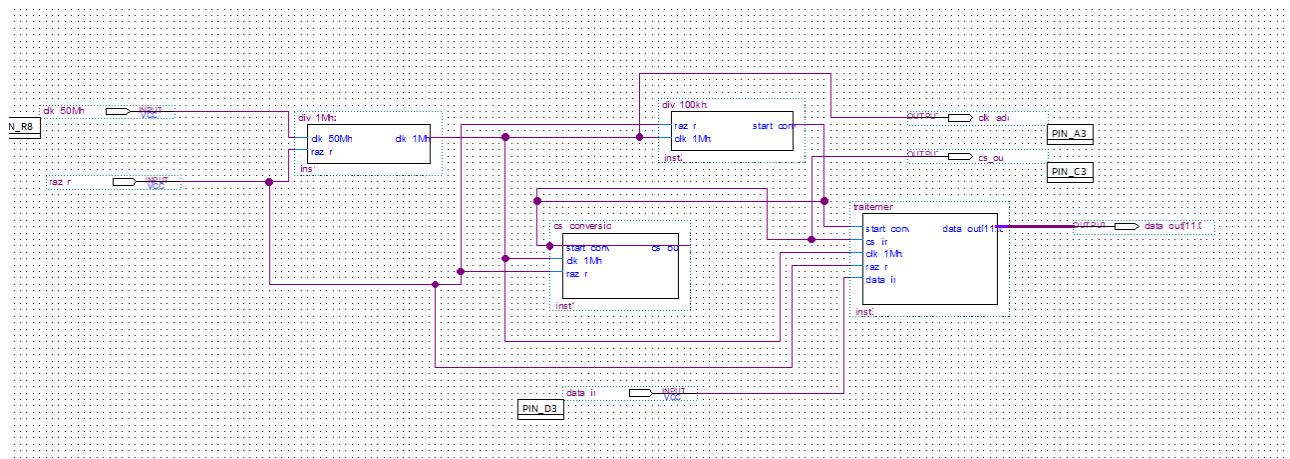


FIGURE 3.29 – Schéma bloc du convertisseur

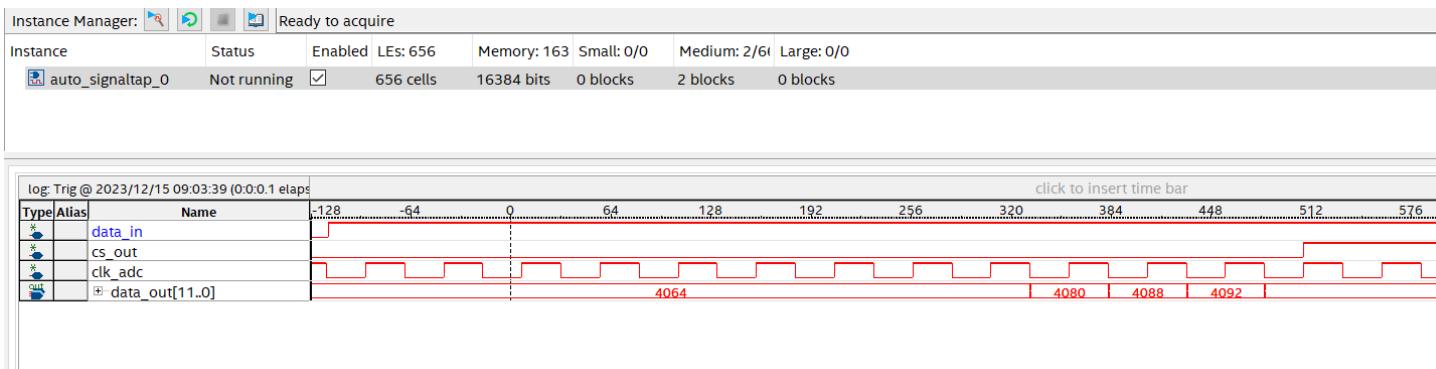


FIGURE 3.30 – La valeur maximal du convertisseur 4095

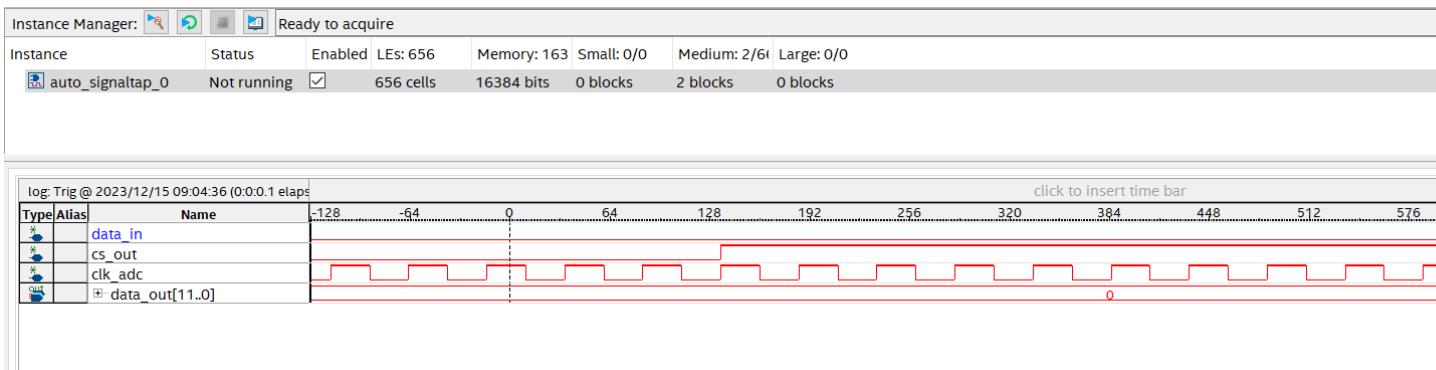


FIGURE 3.31 – La valeur minimal du convertisseur 0

==>Après développement de la fonction, nous avons eu le même comportement que spécifié dans la data-sheet pour la communication avec l'ADC.

CONCLUSION

Toutes ces séances nous ont permis de comprendre la synthèse et la mise en des systèmes. L'analyse fonctionnelle et la simulation nous ont aidé à programmer les fonctions à implémenter en VHDL sur Quartus 18 et corriger les erreurs pendant la phase de debug. Ce projet de Bureau d'études nous a permis de configurer les composants matériels sur une carte FPGA grâce aux fonctions logiques en langage VHDL. Dans la deuxième partie, ce projet nous a permis aussi de distinguer les composants d'un microcontrôleur où il y a une interaction entre les périphérique et la mémoire grâce aux processeur NIOS II. Cela nous a donné l'opportunité de travail avec le langage de description matériel (VHDL) et le langage de programmation C. Nous avions réalisé des vérifications du fonctionnement demandé dans le cahier des charges par la simulation et le test sur la maquette. Et nous avions aussi travaillé sur l'interfaçage des blocs composants avec le bus du processeur (Bus Avalon et le NIOS II) grâce aux outils de développement Quartus Prime et le NIOS II Eclipse.

Ce projet a été bénéfique pour nous en matière de recherche et de la gestion de projet. Aussi, au niveau bilan personnel, les séances de Bureau d'étude nous a permis de renforcer notre travail en équipe en répartissant les tâches pour faciliter la résolution des problèmes et trouver ainsi les solutions adéquates. Il s'agit d'une opportunité pour approfondir nos connaissances sur les cartes FPGA. Nous souhaitons remercier notre encadrant Mr Thierry PERISSE qui nous a aidé durant les séances du BE afin de développer nos compétences dans l'analyse et l'étude des circuits logiques dans une carte FPGA.