Université Paul Sabatier

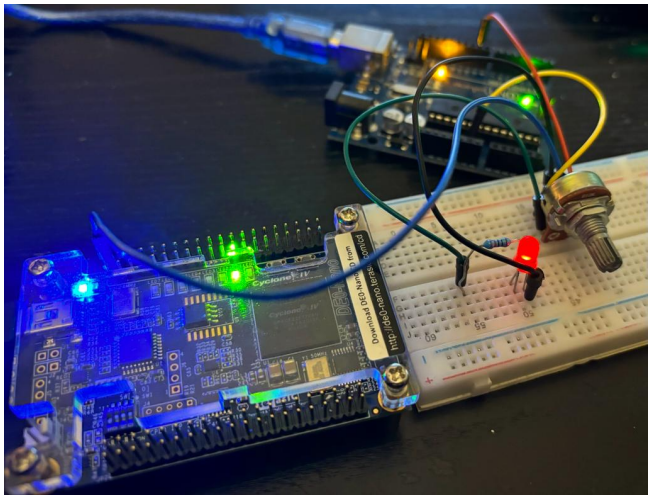
RAPPORT DE Travaux Pratiques

Bureau d'études

Mise en oeuvre d'un pilote de barre franche pour voiliers

Auteurs :
Lucky ROBINSON
Hafsa RAOUI

Encadrant :
PERISSE Thierry



16 novembre 2023

INTRODUCTION GÉNÉRALE

Dans le domaine de l'électronique numérique, la connaissance des systèmes basés sur le langage de description matérielle ou le VHDL est primordial pour mettre en oeuvre des projets sur des systèmes électroniques numériques. 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.

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.

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.

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 basé 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.

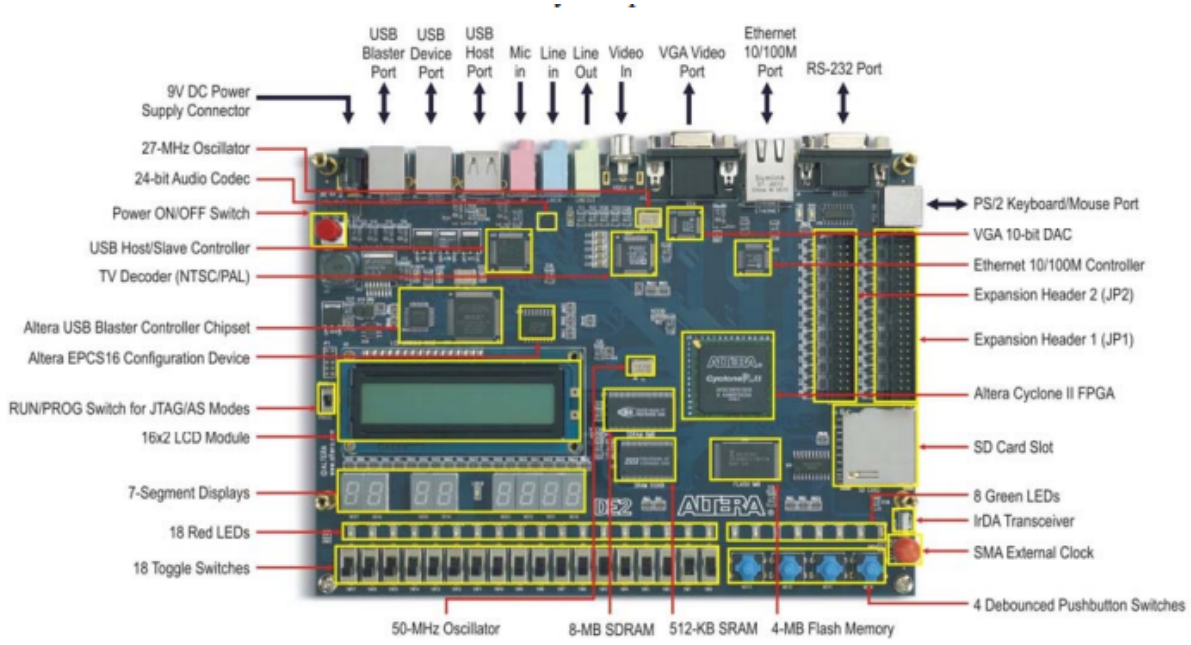


FIGURE 1.1 – Carte DE2

1.2 Carte DE0 Nano

Cette carte de développement de la famille Altera Cyclone IV est adapté 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 possiblité 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-Nan 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.

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 temporelles, il faut effectuer une simulation fonctionnelle. Nous nous sommes basés de la table de vérité de la fonction logique ET pour effectué la simulation fonctionnelle. Voici le résultat obtenu :

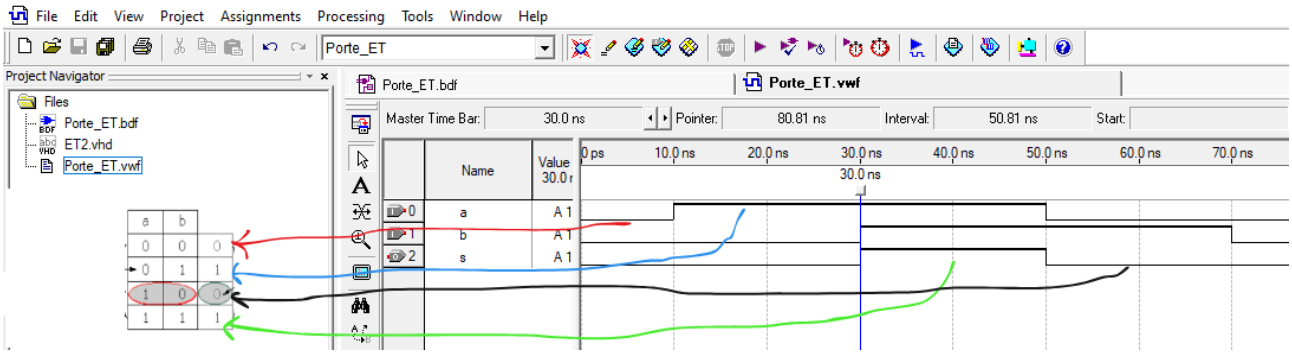


FIGURE 2.1 – Simulation Porte ET

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.

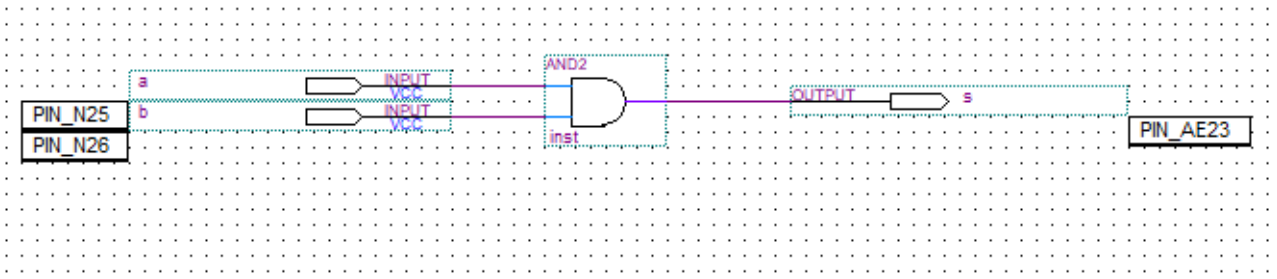


FIGURE 2.2 – Initialisation liaison I2C

2.3 Additionneur de 2 mots de 1 bit

Dans cette partie, nous avions réaliser 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 avions 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.

Cin \ AB	OO	O1	11	1O
0	0	0	1	0
1	0	1	1	1

Tableau de Karnaugh pour la sortie de retenue Cout

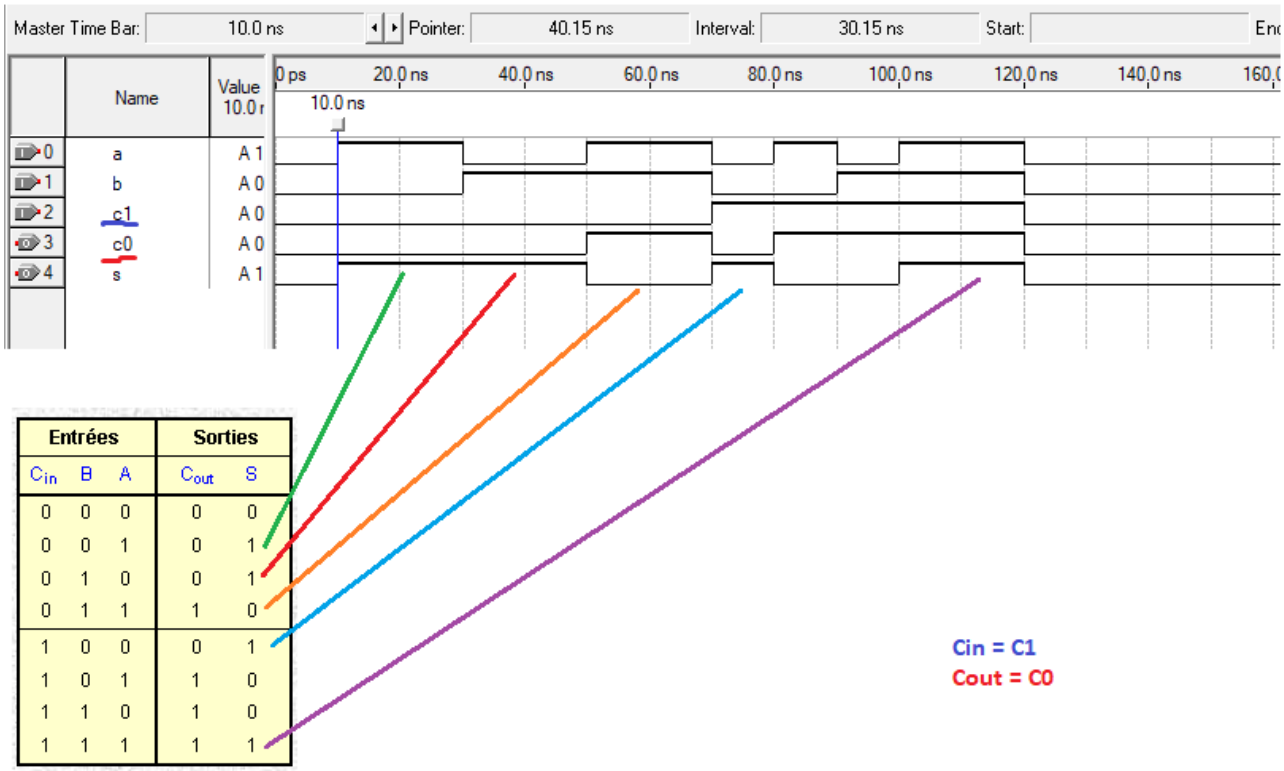
Cout = (A Xor B) Xor Cin

Cin \ AB	OO	O1	11	1O
0	0	1	0	1
1	1	0	1	0

Tableau de Karnaugh pour la sortie S

S = (A Xor B) OR ((A Xor B) AND Cin)

FIGURE 2.3 – Tableau de Karnaugh



Entrées			Sorties	
C_{in}	B	A	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

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.

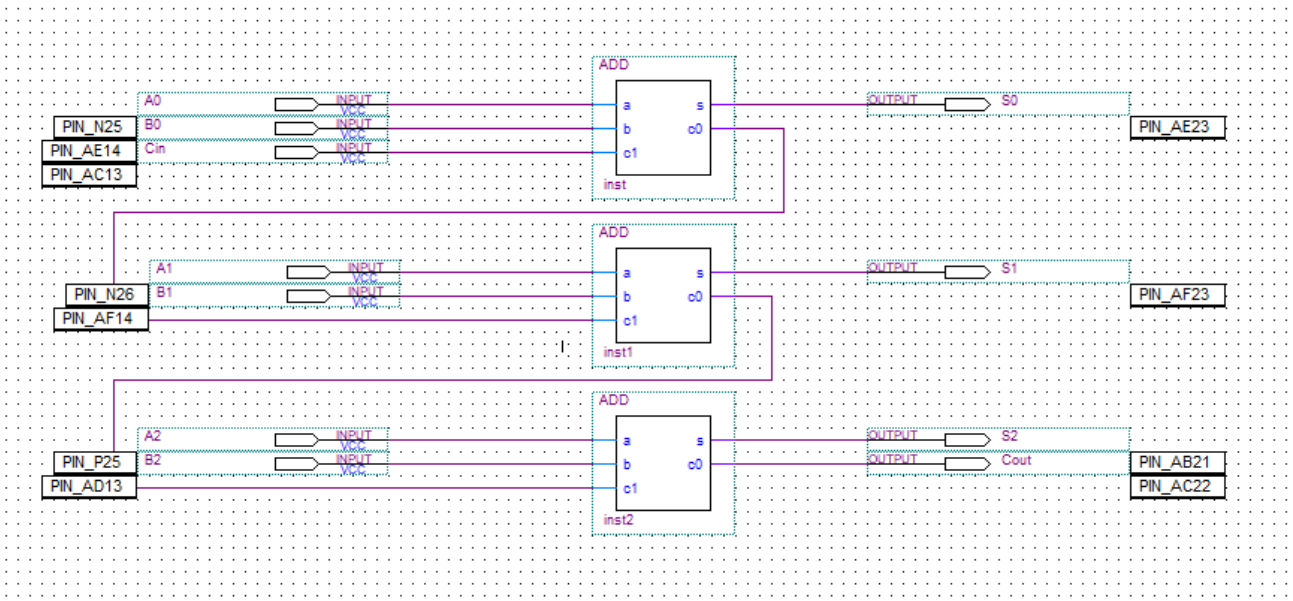


FIGURE 2.5 – Montage du capteur sur stm32

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.

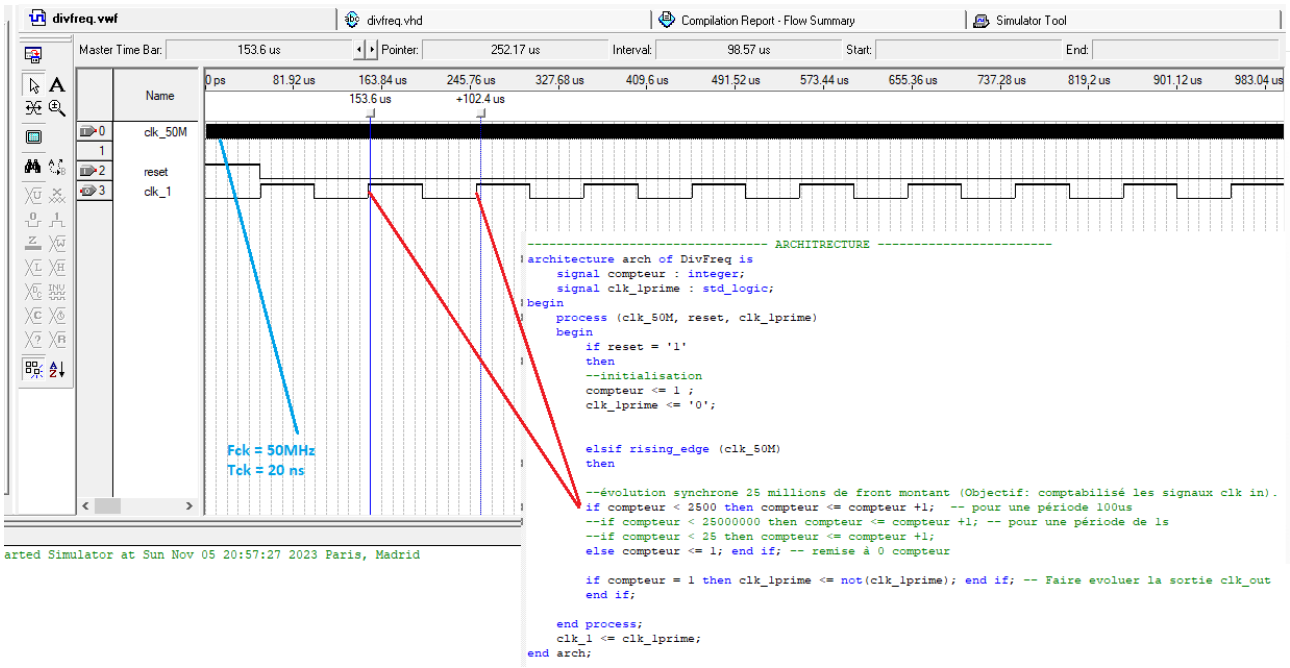


FIGURE 2.6 – Simulation diviseur de fréquence

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 compteur 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.

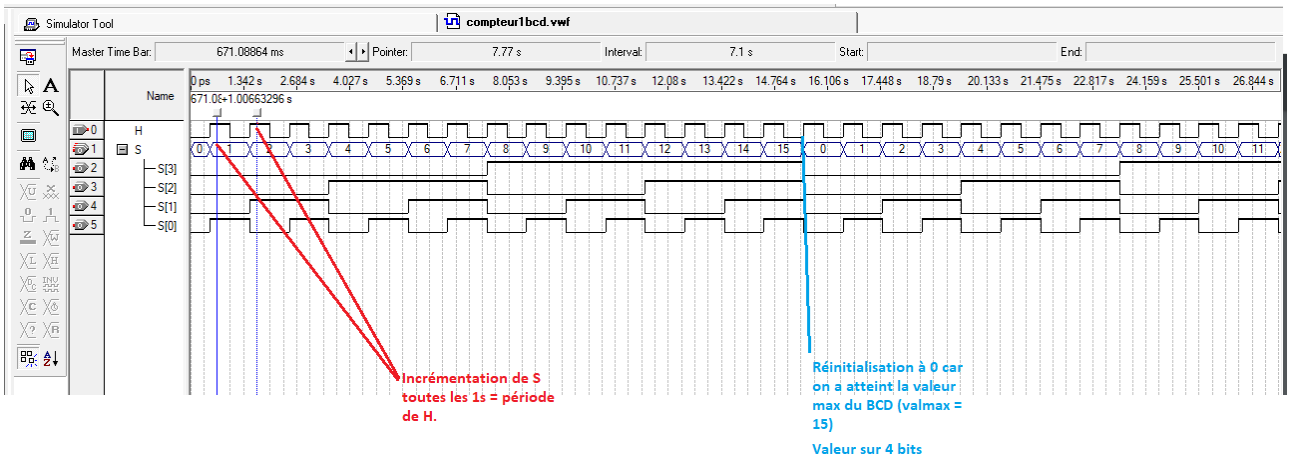


FIGURE 2.7 – Simulation comptage BCD simple

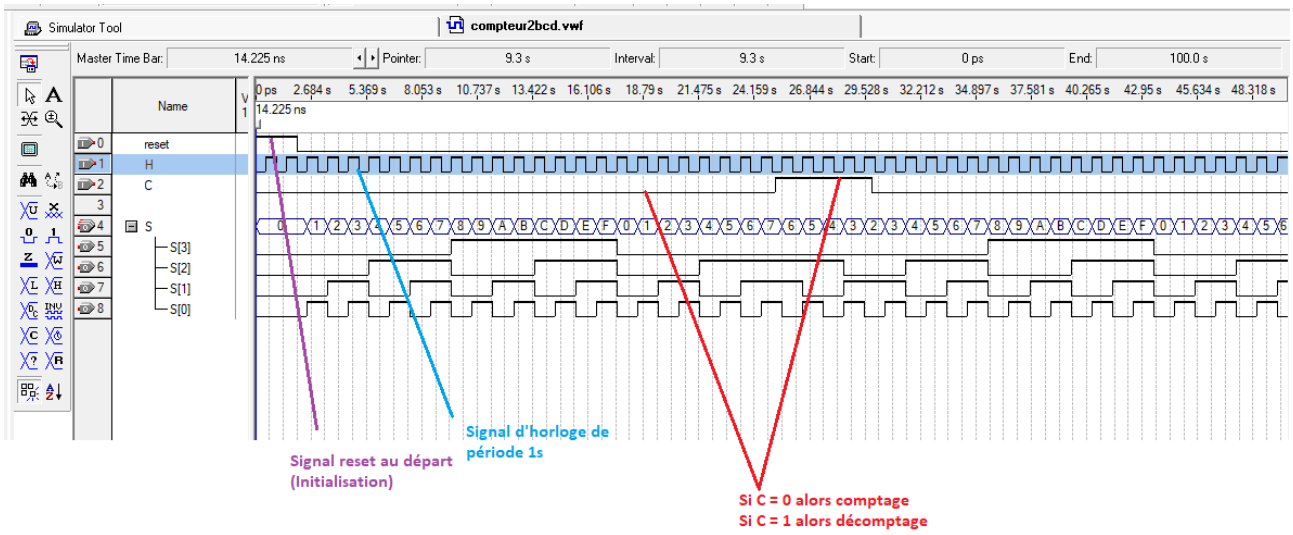


FIGURE 2.8 – Simulation comptage/décomptage BCD

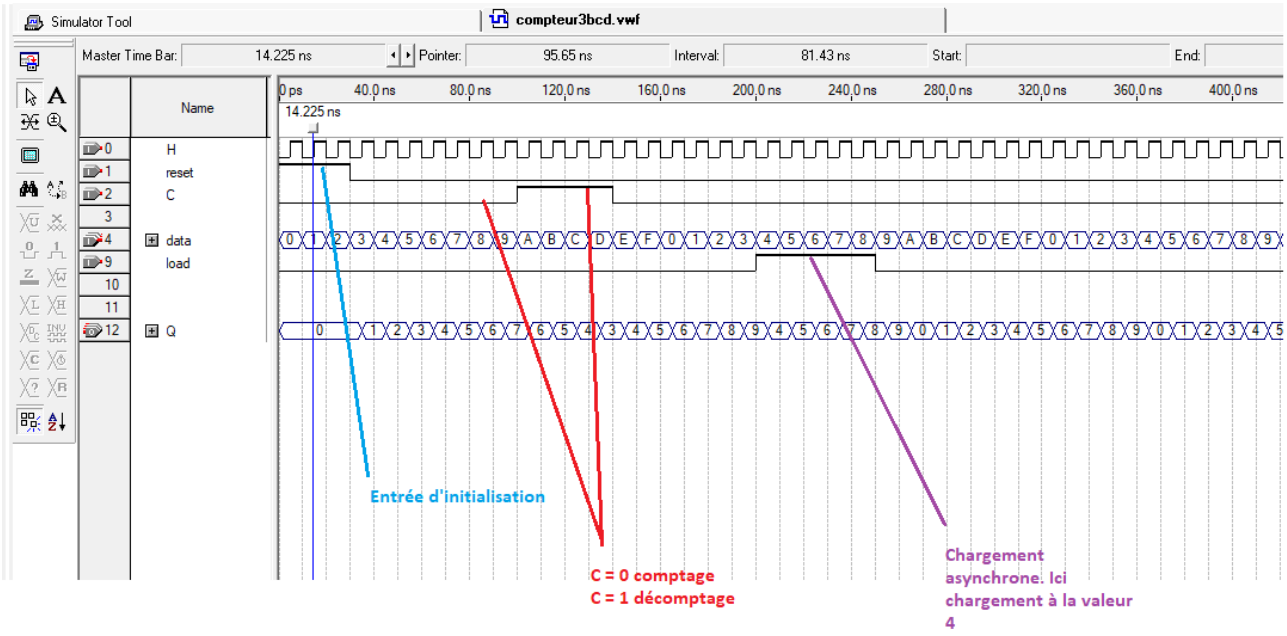


FIGURE 2.9 – Simulation comptage/décomptage BCD avec chargement asynchrone

2.5 Bloc 3 : Décodeur 7 segments :

Pour avoir l'affichage des secondes sur un afficheur 7 segments, nous avions 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.

```
entity deco7seg is
port(--en : in std_logic;
     E    : in std_logic_vector(3 downto 0);
     --seg_d  : out std_logic_vector(6 downto 0);
     seg_u : out std_logic_vector(6 downto 0)
     );
end deco7seg;

architecture arch of deco7seg is


begin

    -- i <= en & E
    --with i select
    with E select
        seg_u <=  "0000001" when "0000",  --0
                  "1001111" when "0001",  -- 1
                  "0010010" when "0010", -- 2
                  "0000110" when "0011", -- 3
                  "1001100" when "0100", --4
                  "0100100" when "0101", -- 5
                  "1100000" when "0110", -- 6
                  "0001111" when "0111", -- 7
                  "0000000" when "1000", -- 8
                  "0001100" when "1001", -- 9
                  "1111111" when others; -- Eteint car actif bas (à 0)

    end arch;
```

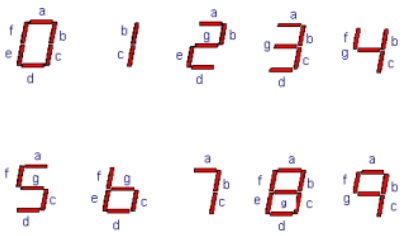


FIGURE 2.10 – Décodage 7 segments

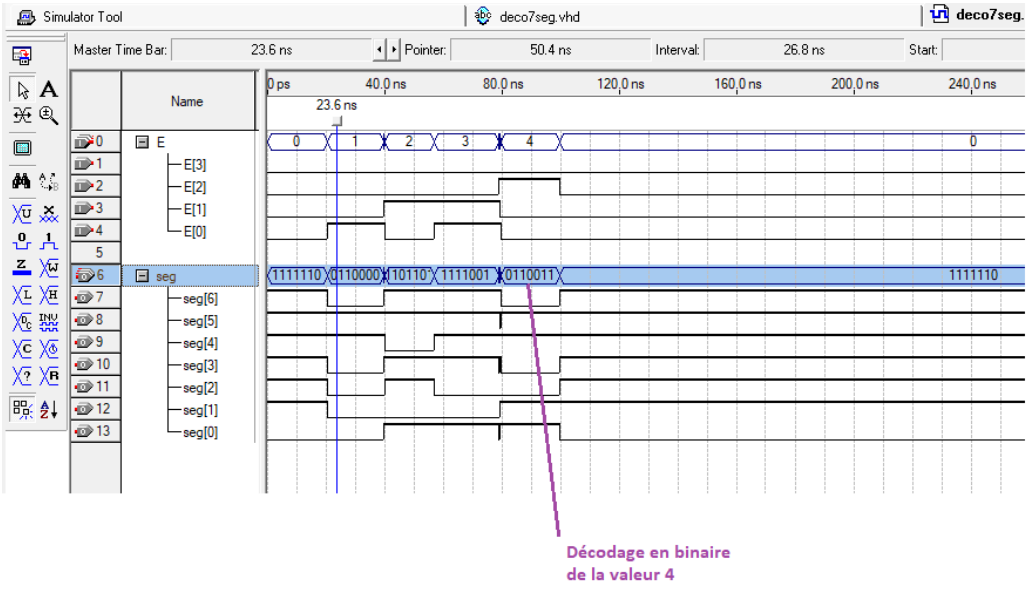


FIGURE 2.11 – Simulation décodage 7 segments

2.6 Simulation et test sur la carte du mini projet

Après avoir testé séparement 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.

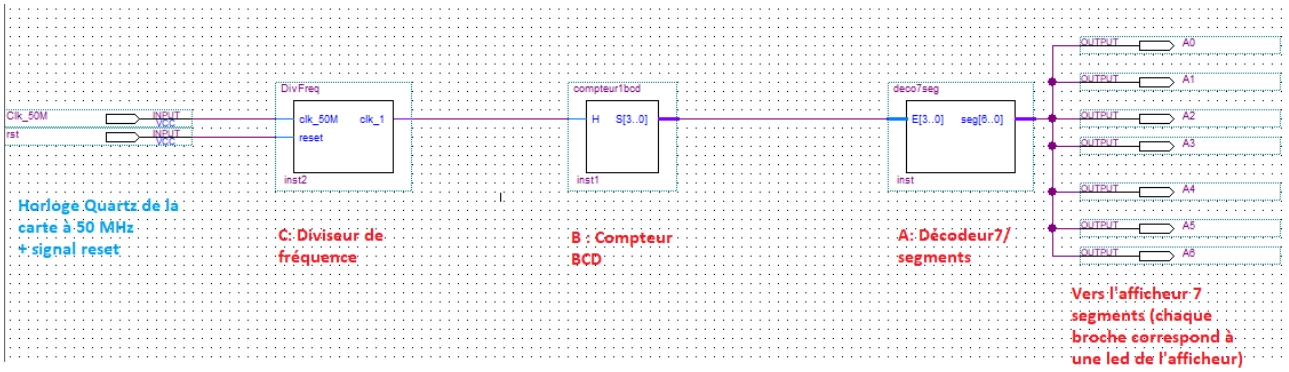


FIGURE 2.12 – Assemblage et cablage des blocs du mini projet

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 = Ton/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.

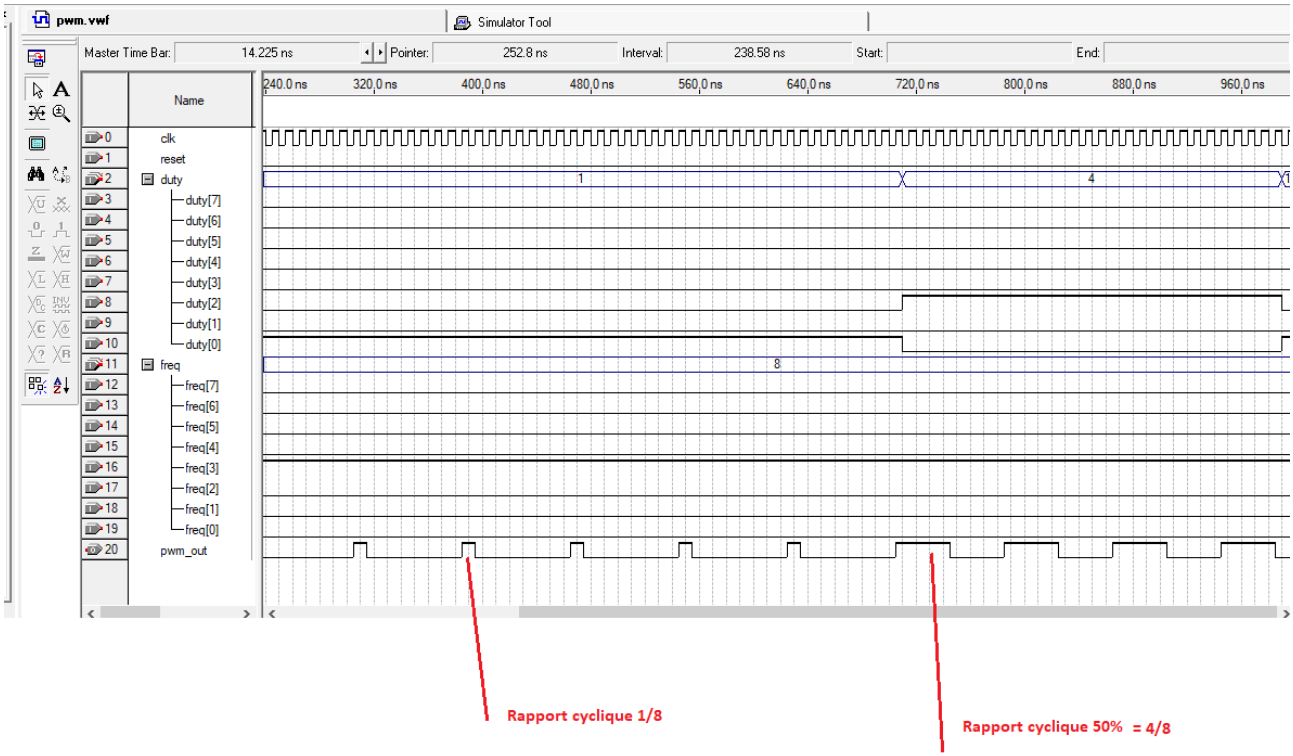


FIGURE 2.13 – Simulation PWM

CHAPITRE 3

BE : PILOTE DE BARRE FRANCHE POUR VOILIERS

Dans ce chapitre, nous allons effectué 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. Les étapes de développement s'effectuent en deux parties : - la première partie consiste à développer des modules et fonctions sur un SOC de la famille Altera avec la carte DE0 Nano. -la deuxième partie nous amène à réaliser des blocs fonctionnels et logiciels entre Quartus et le NIOS II Eclipes en langage C.

3.1 Fonction simple : Anémomètre

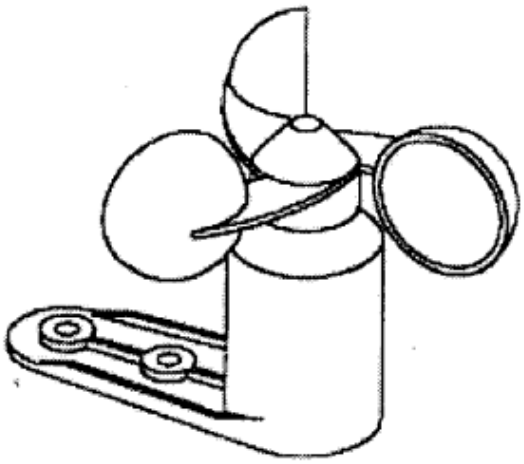
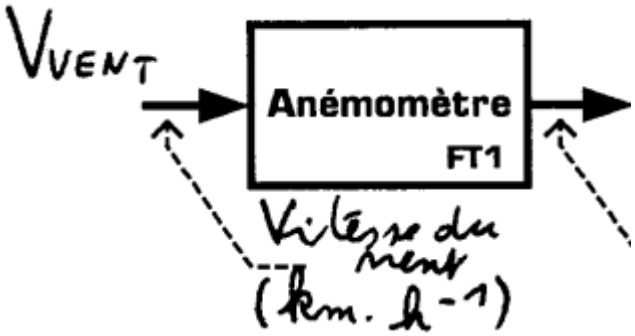


FIGURE 3.1 – Anémomètre coiffé de sa roue à aube

L'anémomètre permet de mesurer et convertir la vitesse du vent en un signal impulsionnel ayant une fréquence proportionnelle à cette vitesse. Il s'agit ici de traiter le signal issu de l'anémomètre. Pour connaitre 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 3.2 – Vue externe de l'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.

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.

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.

FIGURE 3.3 – Zoom sur le signal d'horloge 50 MHz et le test bench

Voici le résultat obtenu par la simulation en sortie :

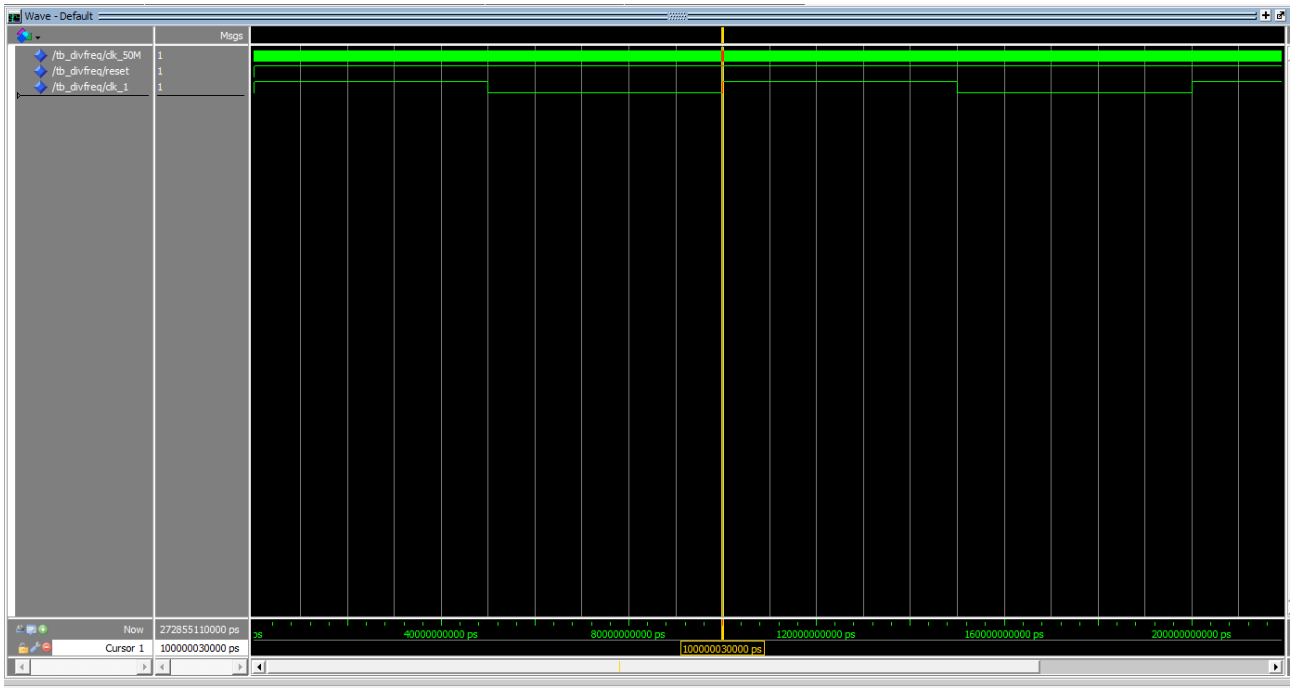


FIGURE 3.4 – Diviseur de fréquence 50 MHz à 1Hz sur ModelSim

3.1.1.2 Détecteur de fronts montants de l'horloge 1 Hz :

Ce composant est conçu pour identifier les fronts montants du signal produit par le diviseur de fréquence. Si aucun front montant n'est détecté, la sortie du composant est réglée sur 0. En revanche, si un front montant est détecté, la sortie est mise à 1.

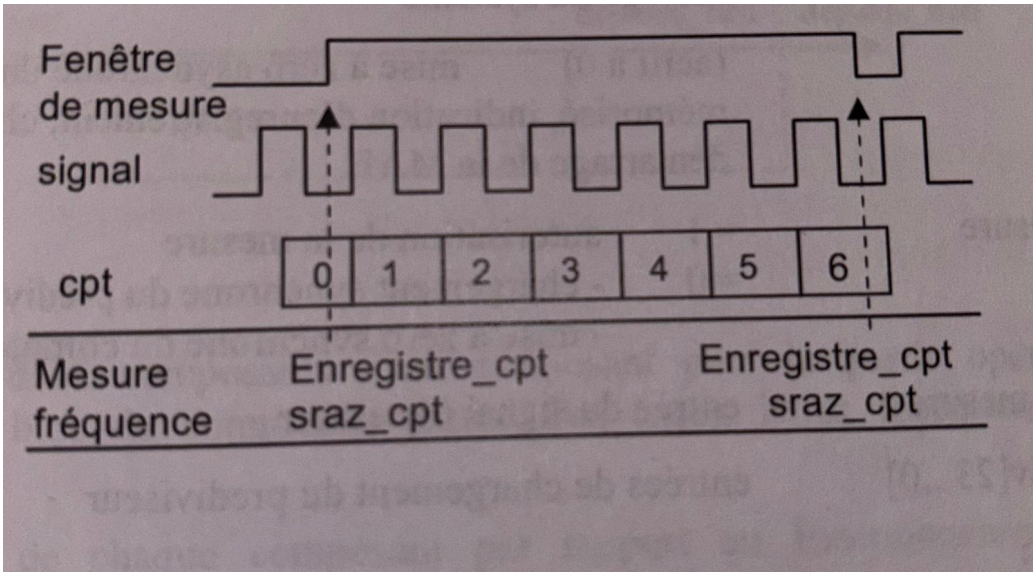


FIGURE 3.5 – Principe du détecteur et compteur de front montant d'un signal d'entrée dans la fenêtre de mesure de l'horloge à 1 Hz dans notre cas

3.1.1.3 Compteur des fronts montants

Le composant "front counter" est configuré pour effectuer le comptage des fronts montants générés par le "détecteur de fronts", et ce, à chaque seconde. Cette opération est synchronisée avec le signal de 1 Hz du diviseur de fréquence. Par conséquent, à chaque intervalle d'une seconde, le "front counter" réinitialisera son comptage en réponse aux fronts montants détectés. On choisit un signal où sa fréquence ou sa période est déterminé. Comme il faut synchroniser l'horloge à 1Hz par rapport au signal d'entrée, il faut effectuer un comptage pendant qu'on est au front montant du signal 1 Hz. SI on effectue la division de la demi période du signal 1 Hz par la période du signal d'entrée, on obtient le nombre de front à l'état haut. Il suffit de multiplier par deux pour avoir le nombre de front pour la période complète de l'horloge clk 1Hz. Ce nombre de période correspond donc à la vitesse. Il faut mettre à jour le comptage lorsqu'on détecte un front montant de l'horloge 1Hz et mémoriser ainsi la valeur.

FIGURE 3.6 – Affichage de la Température et pression sur l'écran LCD

FIGURE 3.7 – Affichage de l'altitude et de la pression sur l'écran LCD

3.1.1.4 Mémorisation d'événement

L'unité de comptage est ici le clk 1Hz donc 1seconde 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 montants comptés à la fin de la mesure, c'est à dire à chaque changement en front montant de clk 1Hz.

3.1.1.5 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.

3.1.2 Implémentation sur Quartus 18

3.1.2.1 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').

3.1.3 Configuration sur SOPC

Le but de cette partie est de réaliser le SOPC de l'avalon pwm pour l'intégrer dans le bloc anémomètre pour générer le signal de la fréquence d'entré, après la création des GPIO, CPU, le signal PWM ,JTAG,Clock,mémoire. On a fait le câblage du circuit pour le simuler avec Eclipse et avoir un composant pour l'intégrer dans notre projet final .

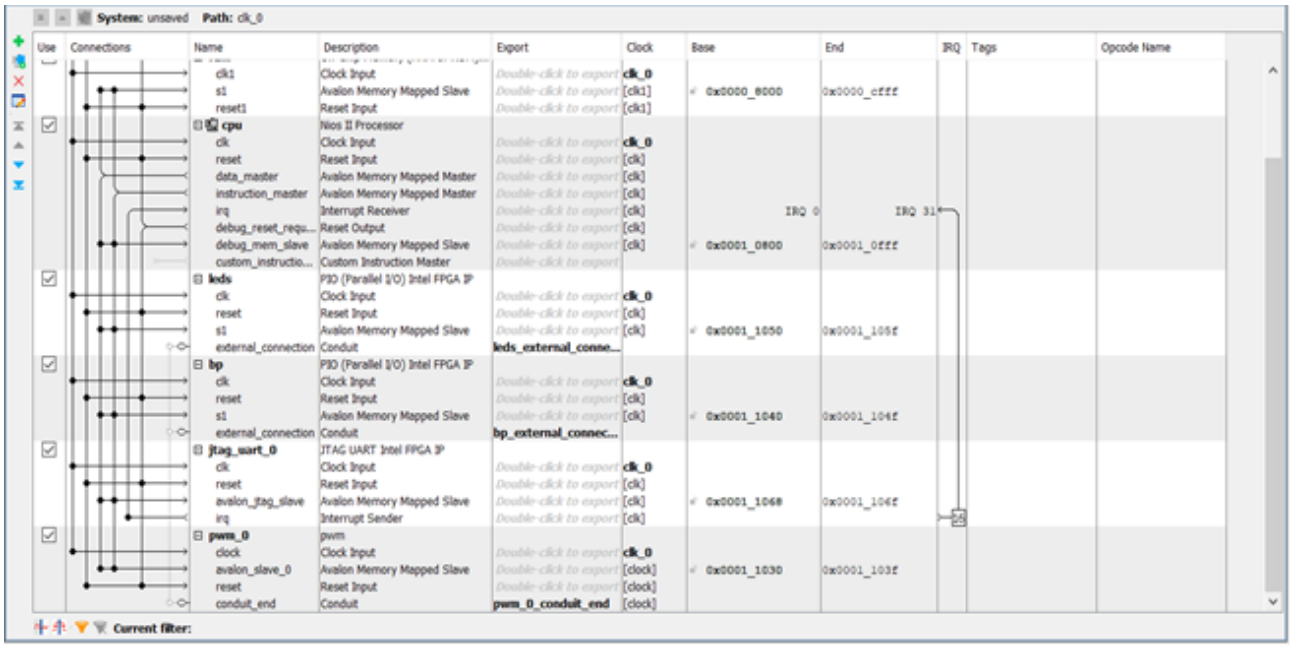


FIGURE 3.8 – Circuit sur platform Designer

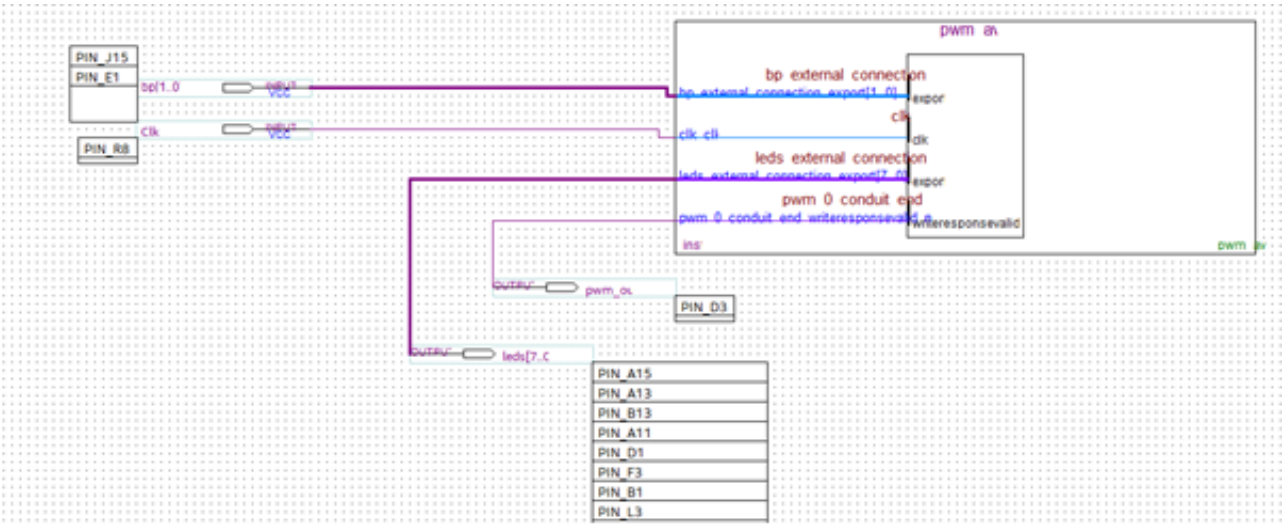


FIGURE 3.9 – Bloc SOPC PWM