



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



Université
de Toulouse

Bureau d'étude : Pilote de barre franche



Réalisé par :

- *Youssef Abbih*
- *Farouk Madouni*

Encadré par :

- *Pr. Thierry Périssé*

Sommaire

1. Introduction
2. Mise en situation
 - 2.1. Architecture générale du pilote barre franche
 - 2.2. Objectifs du bureau d'études
3. Travail réalisé
 - 3.1. Anémomètre
 - 3.1.1. Entrées / sorties bloc anémomètre
 - 3.1.1. Bloc anémomètre
 - 3.1.1. Schémas fonctionnel
 - 3.1.1. Diviseur de fréquence (1Hz)
 - 3.1.1. Compteur nombre de fronts
 - 3.1.1. Machine à états
 - 3.1.1. Simulation Modelsim
 - 3.2. Boussole
 - 3.2.1 Entrées / sorties bloc compas
 - 3.2.2 Bloc Compas
 - 3.2.3 Schémas fonctionnel
 - 3.2.4 Diviseur de fréquence (10kHz)
 - 3.2.5 Compteur durée état haut
 - 3.2.6 Machine à états
 - 3.2.7. Simulation Modelsim
 - 3.3. Gestion des boutons
 - 3.3.1 Entrées / sorties bloc boutons
 - 3.3.2 Bloc boutons
 - 3.3.3 Schémas fonctionnel
 - 3.3.4 Diviseurs de fréquences (100Hz, 50Hz, 1Hz)
 - 3.3.5 Bloc machine à états boutons poussoir
 - 3.3.6 Bloc générateur tempo
 - 3.3.7. Bloc générateur bip
 - 3.3.8 Simulation Modelsim
 - 3.4. Interface Avalon et Qsys
 - 3.5. Validation BE
4. Conclusion
- Annexe
 - A. code VHDL Anémomètre
 - B. code VHDL Compas
 - C. code VHDL Boutons
 - D. code NIOS 2

1. Introduction

Le pilotes de barre franche est une solution technique pour diriger et faire naviguer un voilier de manière autonome. Le principe étant de prendre en compte des grandeurs physique tel que la direction et la force du vent, et le voilier conservera la direction indiquée (le cap) quelques soit les perturbations. Pour réaliser ce système, nous utiliserons une carte DEO de chez Altera. Nous utiliserons les logiciels du constructeur pour implémenter notre code : Quartus 9.0 & Quartus 11.0. L'intégralité du projet est disponible sur l'outil GitHub à l'adresse suivant :

https://github.com/youssef-abbih/tp_VHDL_G2_B5

2. Mise en situation

Le pilote automatique d'un voilier est un système installé en complément du pilotage manuel. Véritable équipier, le pilote automatique permet de diriger automatiquement un voilier afin que ce dernier conserve le cap préalablement défini.

Les pilotes automatiques se rangent en deux grandes catégories.

Catégorie 1 – Pilote automatique de cockpit

Pilote de cockpit : commande directe sur la barre (franche ou à roue). Sur une barre franche, c'est un vérin qui agit tandis que sur une barre à roue c'est une courroie ou une mignonnerie entraînée par un moteur électrique.

Ce pilote automatique de voilier ne peut pas être utilisé de façon fiable par tous les temps. Il est totalement adapté pour une navigation sur des bateaux de déplacement maximum de 6 à 7 tonnes et de longueur maximum de 12 mètres et dans des conditions météo estivales. Par contre, il ne peut pas se substituer au barreur dans des conditions difficiles, par mer formée et vents forts.

Attention, étant situé dans le cockpit, ce pilote automatique voilier est exposé aux intempéries.

L'installation du pilote de cockpit peut être réalisée par le propriétaire du voilier, sans l'intervention de professionnels. Ce pilote automatique de voilier doit être installé entre la coque et la barre du bateau ; il est composé :

- D'un compas intégré (compas fluxgate) : de façon continue, ce compas fournit la direction (cap) suivie par le bateau à l'unité électronique,
- D'une unité électronique : garde en mémoire le cap fourni par le compas et calcule la différence entre le nouveau cap du bateau et la valeur du cap mémorisée afin d'envoyer un ordre à l'unité de puissance,
- D'une unité de puissance : agit sur la barre pour resituer le bateau sur sa route programmée,
- D'un capteur d'angle de barre (feedback) : mesure chaque déplacement du safran de façon permanente afin d'envoyer les informations à l'unité électronique ; ces informations sont inverses à celles du compas afin d'éviter au voilier de faire naître un phénomène de lacet (induit par le roulis, le bateau engage un mouvement de rotation sur lui-même).

Catégorie 2 – Pilote automatique in-board

Pilote in-board : agit sur le circuit de commande mécanique ou de commande hydraulique ou avec prise directe sur l'axe du safran.

Ce pilote automatique de voilier peut barrer même si les conditions de vent et de mer sont délicates. Contrairement au pilote cockpit où les accessoires qui le composent sont regroupés dans un seul appareil, ceux du pilote automatique in-board sont répartis dans le bateau : le compas est à l'intérieur du bateau, proche du centre de gravité, le vérin est dans un coffre arrière au niveau du secteur de barre et l'unité de commande est généralement située près de la barre.

Le pilote automatique de voilier in-board peut s'adjoindre des éléments complémentaires :

- Un gyromètre (gyrocompas) : capteur qui mesure les “accélérations radiales rapides” (vague forte de travers) et transmet la correction nécessaire au calculateur plus rapidement que le compas fluxgate,
- Un capteur d’angle de gîte et une centrale d’attitude : utiles surtout sur les voiliers de vitesse (régates).

Dans un système de pilote automatique in-board, le choix de l’unité de puissance est primordial :

- Sur les bateaux de moins de 15 mètres, un vérin linéaire actionné par une pompe hydraulique ou un moteur électrique agit directement sur l’axe du safran,
- Sur les bateaux équipés d’une commande de barre hydraulique, le vérin est éventuellement remplacé par une pompe montée dans le circuit de commande de barre.

L’installation du pilote automatique de voilier in-board est délicate et l’intervention d’un professionnel est recommandée.

2.1. Architecture générale du pilote de barre franche :

Un pilote de barre franche reçoit et renvoie des données qui permettent de naviguer le bateau et de maintenir le cap

Ce schéma représente une vue globale de l’architecture du pilote de barre franche

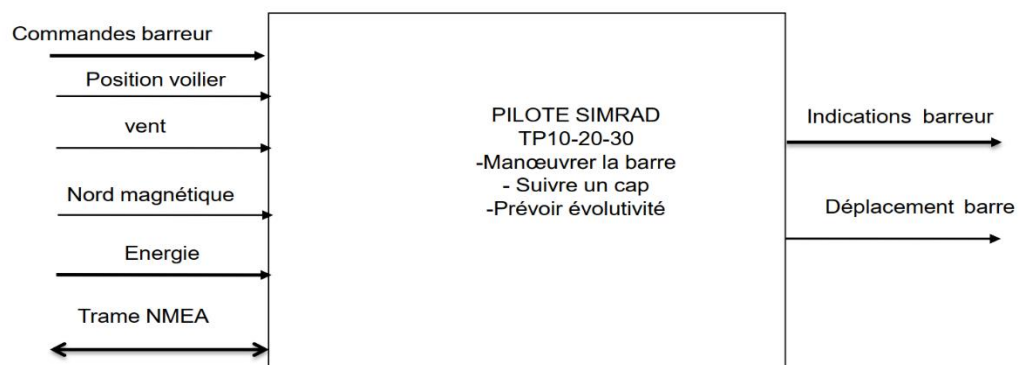


Figure 1 : Architecture pilote

2.2. Objectif du bureau d’étude :

L’objectif de ce bureau d’étude est la conception du pilote de barre franche sous forme d’un SOC, qui sera implémenté sur la maquette mise à notre disposition, contient tous les éléments nécessaires à notre système ainsi qu’une carte FPGA Cyclone IV (DE0 NANO) sur laquelle seront embarqués les différents composants (modules). Pour cela, le pilote a été partagé en cinq modules distincts :

➤ **Module gestion anémomètre** : Ce module récupère et fait le traitement des informations relatives à la force et direction du vent.

➤ **Module gestion compas pour boussole CMPS03 ou CMPS10** : En utilisant la donnée du position du voilier et le nord magnétique cette fonction ressort les informations du cap via la boussole. Cette information sera transmise à la fonction F5 pour le pilotage sécurisé.

➤ **Module gestion des boutons poussoirs** : C’est la fonction prend en entrée les commandes de l’utilisateur via des boutons poussoirs et fournit en sortie des indications « LEDs » visuelles ou sonores « buzzer ».

➤ **Module Gestion interface NMEA** : Fonction de traitement et de gestion des trames spécifiques à la communication inter équipements.

➤ **Module gestion vérin** : C’est la fonction qui gère les déplacements du vérin de la barre et du contrôle des butées.

Cette décomposition modulaire du pilote de barre franche est représentée dans le schéma ci-dessous :

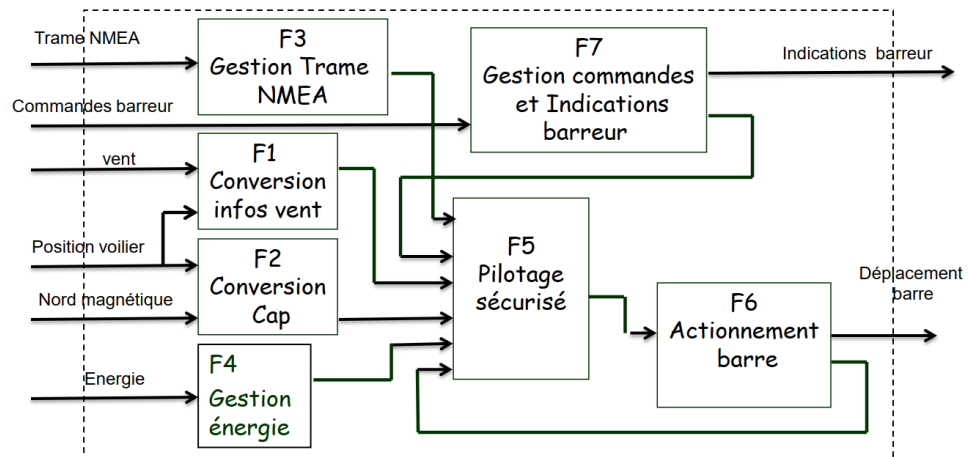


Figure 2 : décomposition modulaire.

3. Travail Réalisé

3.1. Anémomètre

Dans cette partie nous allons traiter le fonctionnement de l'anémomètre. C'est Un appareil qui nous aide à prendre les mesures, la vitesse et la poussée du vent : voilà en gros, **un anémomètre**. Le plus souvent on l'associe à une girouette et fournir à son utilisateur les attitudes précises en ce qui concerne son orientation. Habituellement en l'utilise en météorologie, mais aussi les données apportées peuvent être utilisé pour les transports aérien, en mer ou à reconnaître les effets radioactifs ou les poussières industrielles.

Dans notre projet le vent sera simulé par un GBF générant un signal carré de fréquence de 0-250Hz, et la sortie sera proportionnel au signal d'entrée.



Figure 3 : Anémomètre

3.1.1. Entrées /sorties du bloc anémomètre

```

*****
-- module gestion_anemometre
__*****
--entrées:
--clk_50M : horloge 50MHz
--raz_n: reset actif à 0 => initialise le circuit
--in_freq_anemometre: signal de fréquence variable de 0 à 250 HZ
--continu : si=0 mode monocoup, si=1 mode continu
-- en mode continu la donnée est rafraîchie toute les secondes
--start_stop: en monocoup si=1 démarre une acquisition, si =0
-- remet à 0 le signal data_valid
__*****
  
```

-- sorties:
 -- data_valid: =1 lorsque une mesure est valide
 -- est remis à 0 quand start_stop passe à 0
 -- data_anemometre : vitesse vent codée sur 8 bits
 __*****

3.1.2. Bloc gestion_anemometre

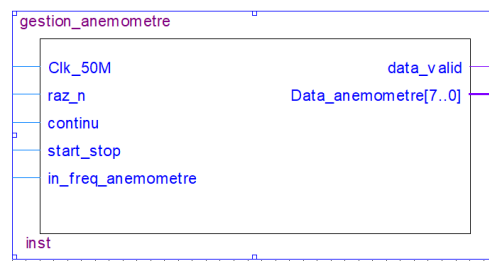


Figure 4 : Bloc gestion_anemometre

3.1.3. Schéma fonctionnel

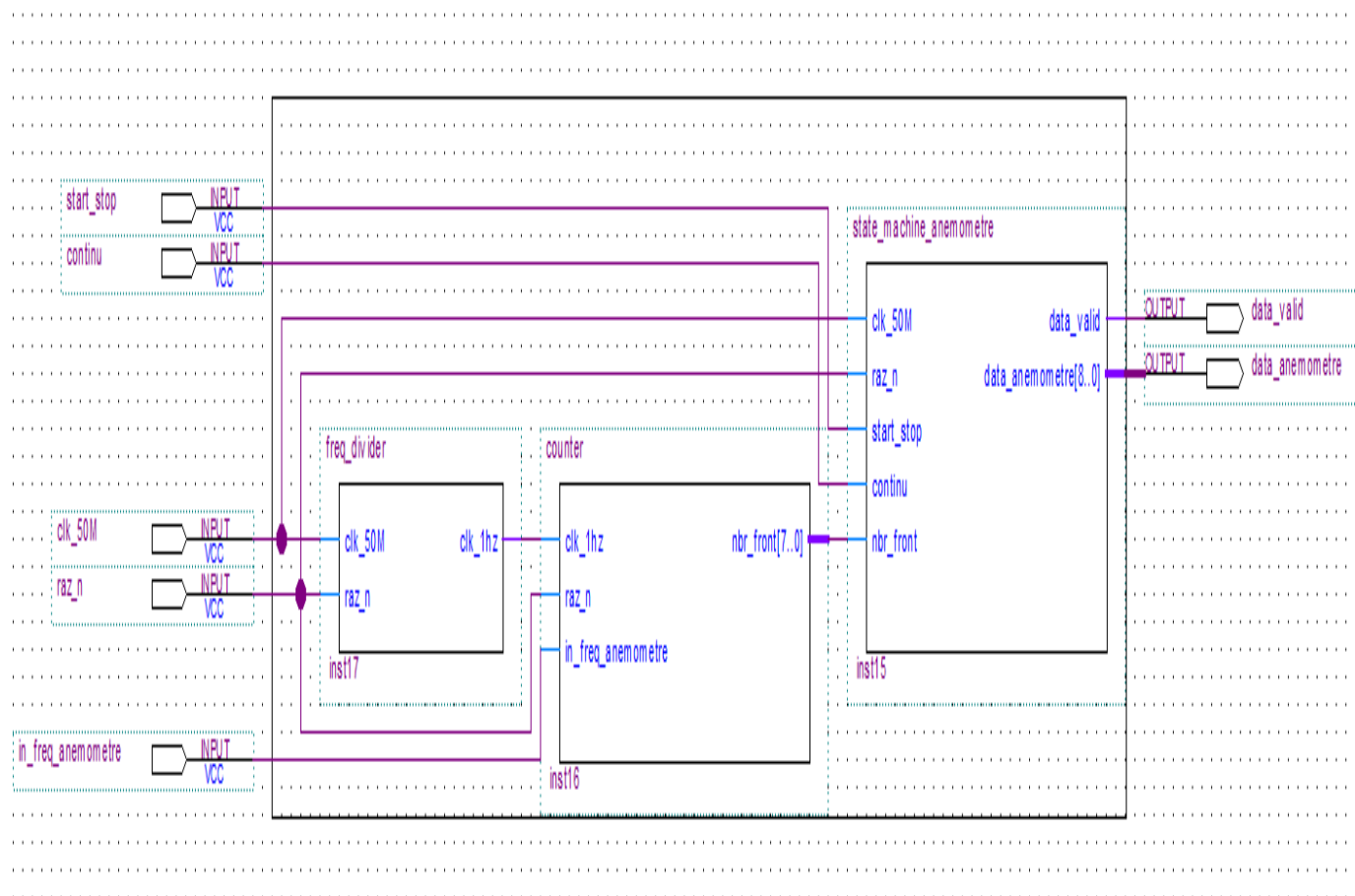


Figure 5 : schéma fonctionnel

3.1.4. Bloc diviseur de fréquence



Figure 6 : Bloc diviseur de fréquence

Ce bloc permet de créer une horloge de 1 Hz qui sera la fenêtre de mesure.

3.1.5. Bloc compteur

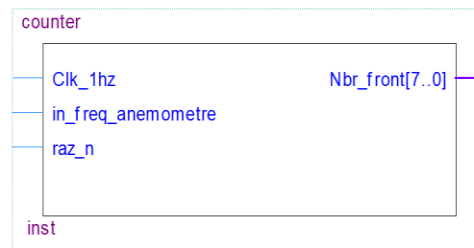


Figure 7 : Bloc compteur

Ce bloc permet de compter le nombre de front montants et descendant du signal in_freq_anemometre pendant une période d'une seconde.

3.1.6. Bloc machine à états

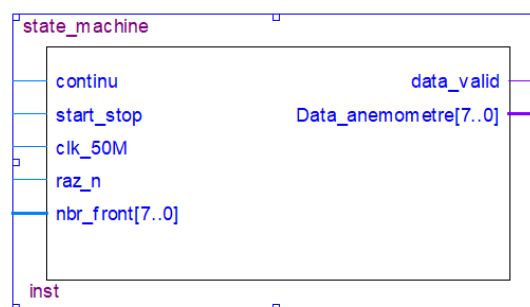


Figure 8 : Bloc machine à états

Ce bloc est une machine à état qui a été mise en lieu pour gérer les deux modes de fonctionnement : mode continu : la mesure s'effectue en continu ; la donnée est rafraîchie toute les secondes et mode monocoup ; l'acquisition de la donnée d'entrée (fréquence) est activée ou désactivée par le signal start_stop.

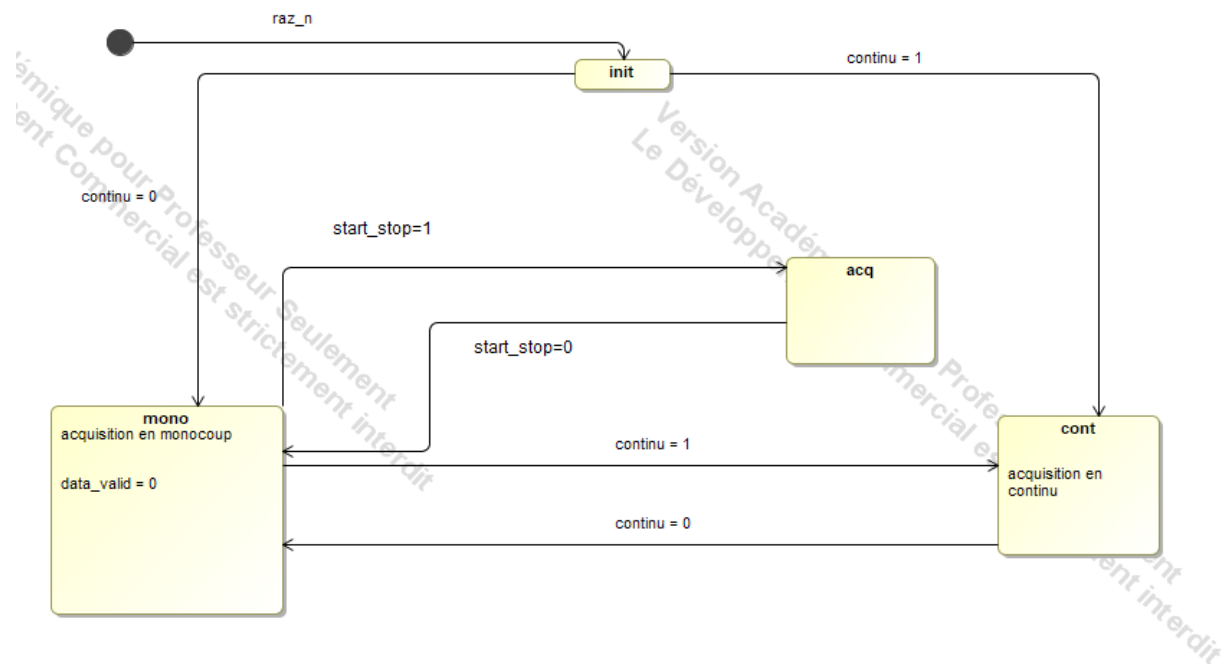


Figure 9 : machine à états

3.1.7. Simulation sur Modelsim

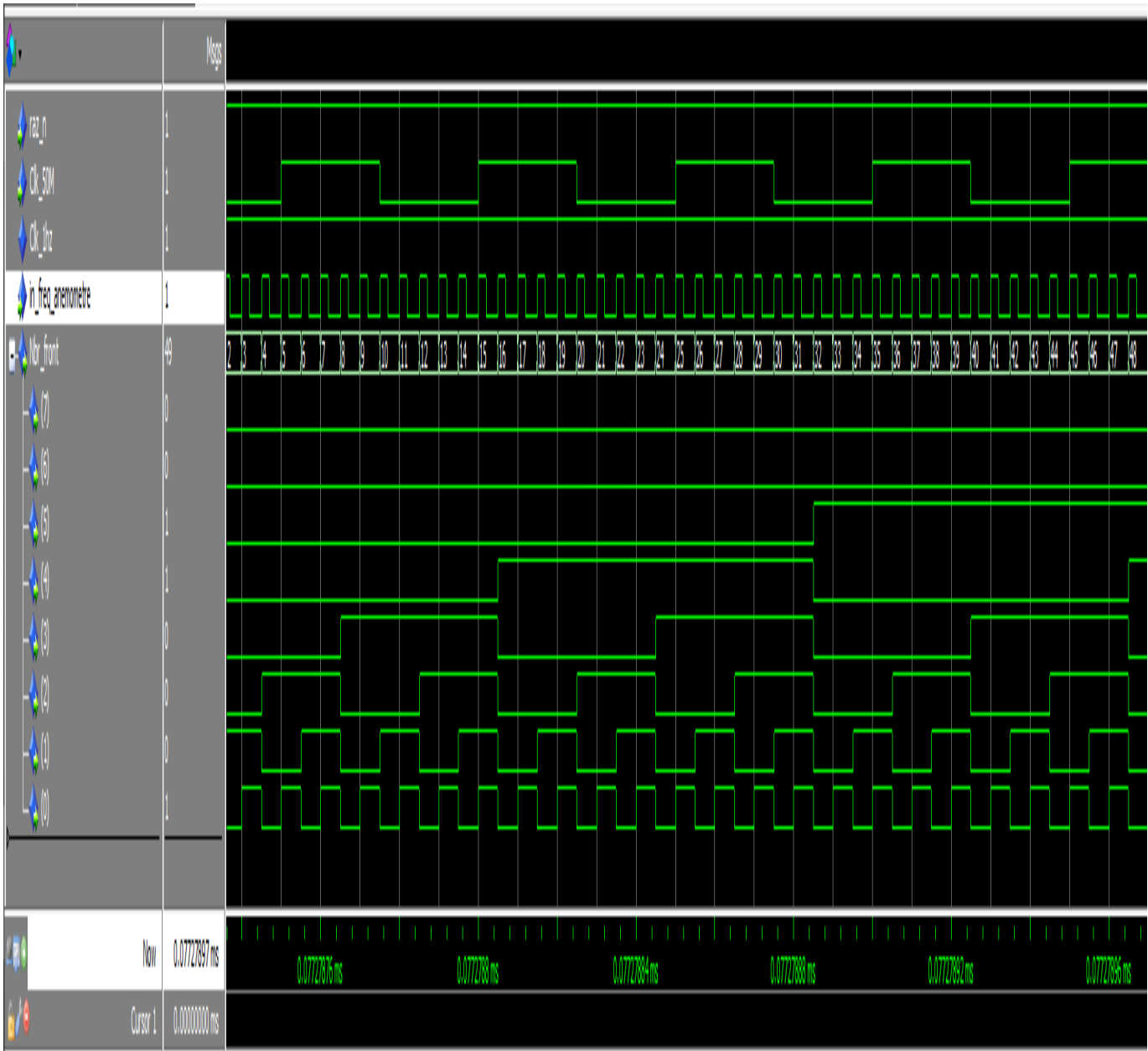


Figure 10 : Simulation

3.2. Compas

Le module gestion de compas a pour but de convertir un signal PWM généré par le compas en une mesure de l'angle correspondante.

• Compas CMPS03

Le compas utilisé dans cette partie est le CMPS03. Ce module est capable de détecter le "nord" grâce à l'emploi de 2 capteurs spécialisés montés en angle à 90° et par déduction indiquer son orientation par le biais d'un échange d'information via un signal PWM. Le signal PWM fournit une impulsion comprise entre 1 ms (0°) et 36,99 ms (359,9°) avec une résolution d'environ 100 us / °).



Figure 11 : Module compas CCMP503

3.2.1. Entrées /sorties du bloc gestion_compas

```
*****
-- module gestion_compas
__*****
--entrées:
--clk_50M : horloge 50MHz
--raz_n: reset actif à 0 => initialise le circuit
--in_pwm_compas: signal PWM de la compas , durée variée de 1 à 36.9ms
--continu : si=0 mode monocoup, si=1 mode continu
-- en mode continu la donnée est rafraîchie toute les secondes
--start_stop: en monocoup si=1 démarre une acquisition, si =0
-- remet à 0 le signal data_valid
__*****
-- sorties:
-- data_valid: =1 lorsque une mesure est valide
-- est remis à 0 quand start_stop passe à 0
-- data_anemometre : vitesse vent codée sur 8 bits
__*****
```

3.2.2. Bloc gestion_Compas

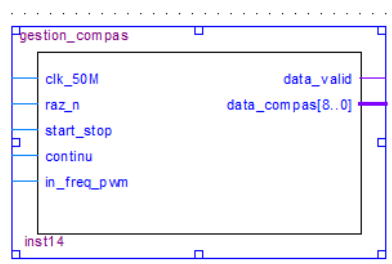


Figure 12 : bloc gestion_compas

3.2.3. Schéma fonctionnel

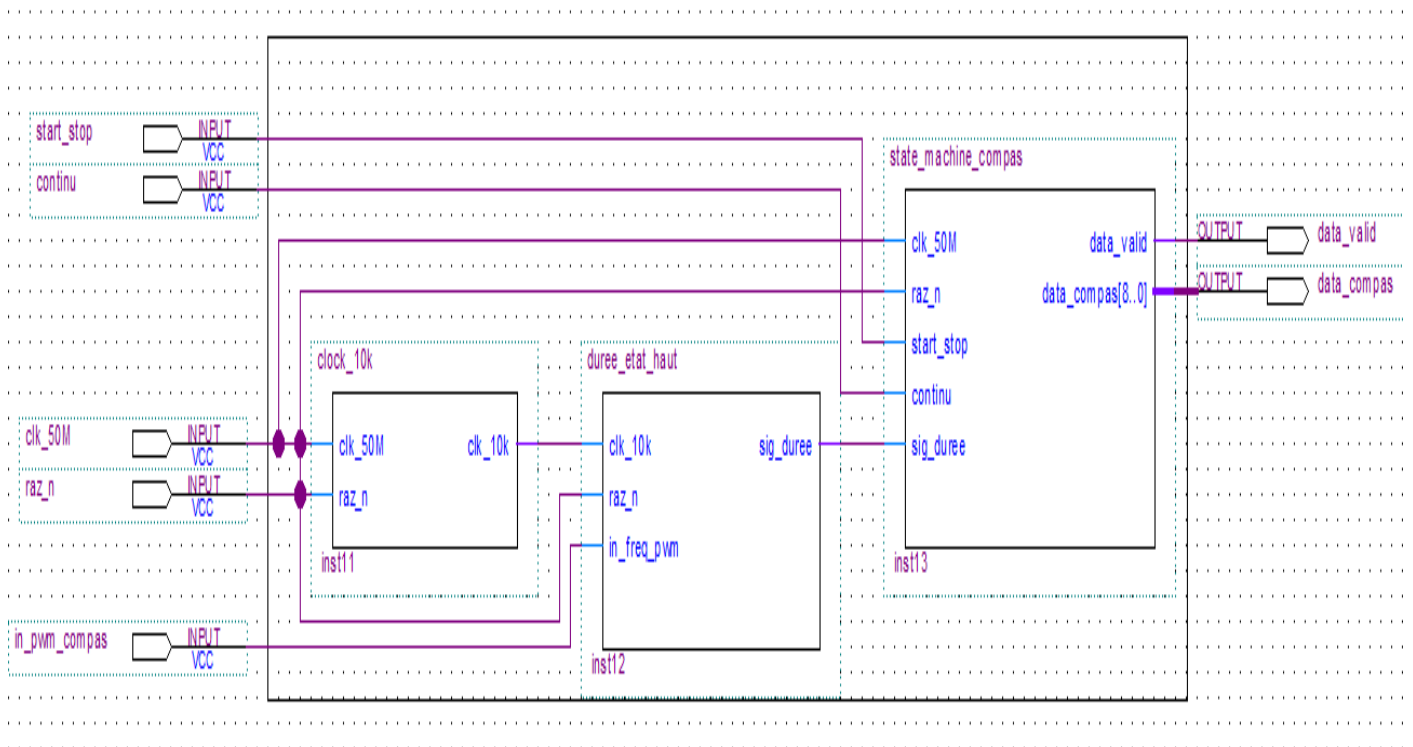


Figure 13 : Schéma fonctionnel module gestion_compas

3.2.4. Bloc diviseur de fréquence

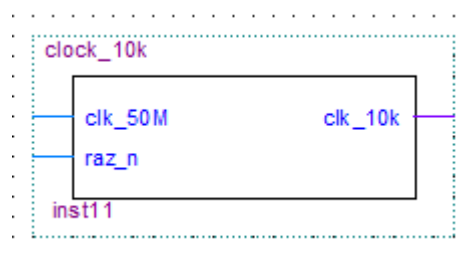


Figure 14 : bloc générateur

Ce bloc permet de créer une horloge de 10k.

3.2.5. Bloc compteur durée état haut

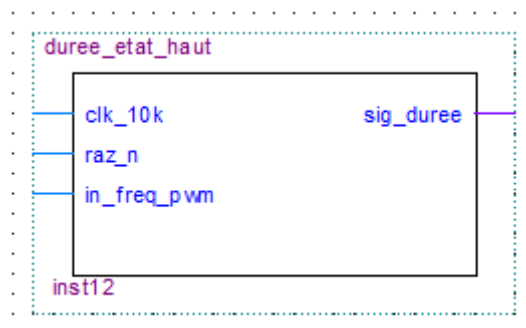


Figure 15 : bloc durée état haut

Ce bloc permet de mesurer la durée de l'état haut du signal in_pwm_compas

3.2.6. Bloc machine à états

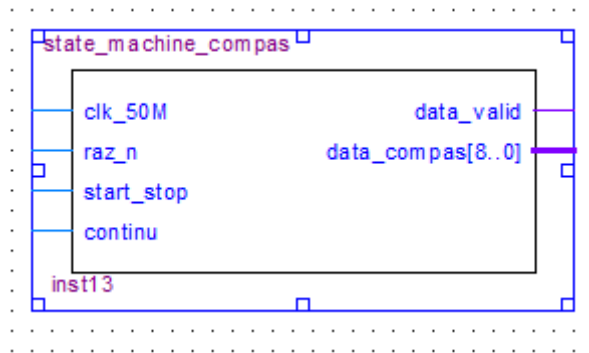


Figure 16 : bloc machine à états

Ce bloc est une machine à état qui a été mise en lieu pour gérer les deux modes de fonctionnement : mode continu : la mesure s'effectue en continu ; la donnée est rafraîchie toute les secondes et mode monocoup ; l'acquisition de la donnée d'entrée (fréquence) est activée ou désactivée par le signal start_stop.

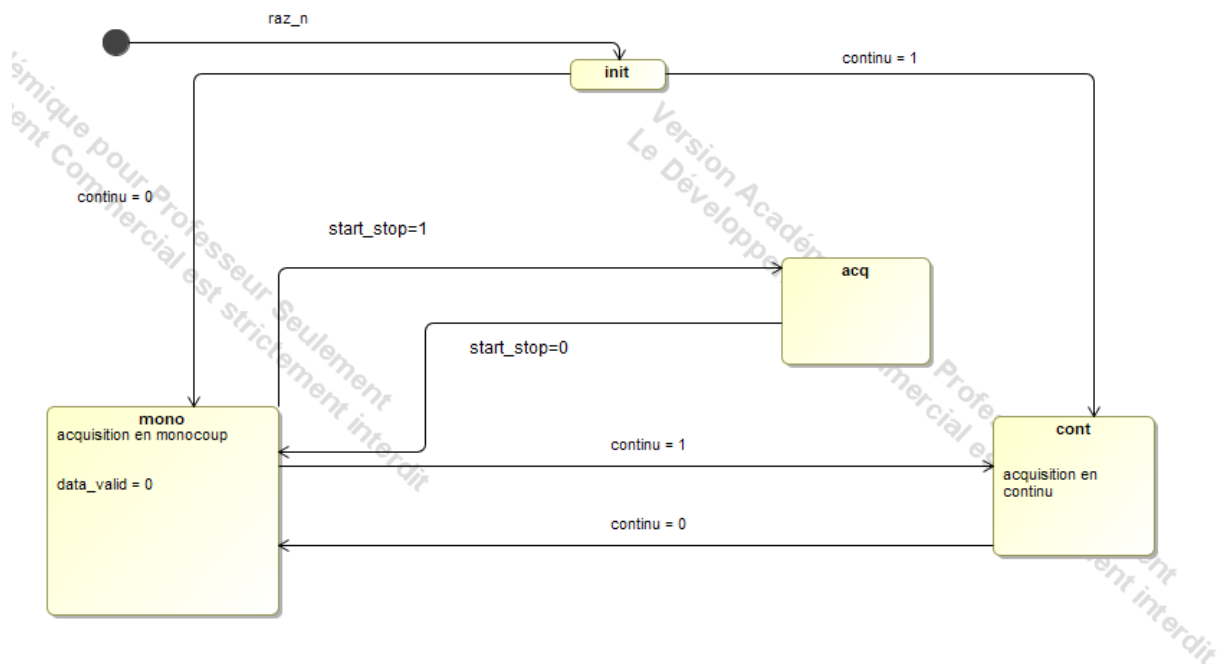


Figure 17 : machine à états

3.2.7. Simulation Modelsim

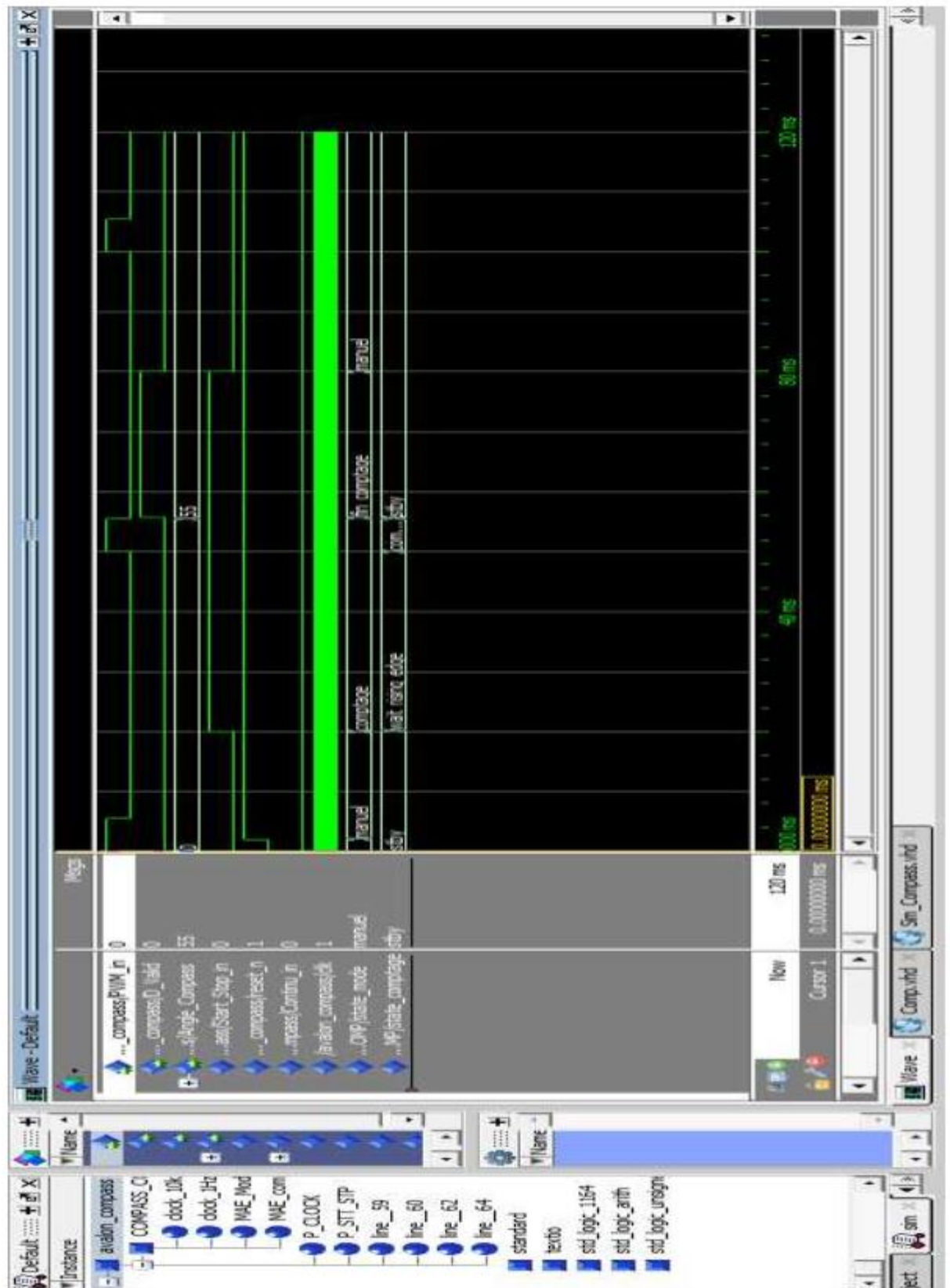


Figure 18 : Simulation Modelsim gestion compas

3.3. Bouton

Cette partie du système a pour but de gérer l'interface des boutons poussoirs du pilote, ainsi que les signalisations qui en résultent, à savoir l'éclairnement des LEDs et le bip sonore.

Les modes de fonctionnement ce sous-système sont :

- Mode sans action : code fonction = '0000'
- Mode manuel bâbord : code fonction = '0001' Rotation à gauche quand le bouton STBY n'est pas appuyé et le bouton bâbord appuyé.

- Mode manuel tribord : code fonction = '0010' Rotation à droite quand le bouton STBY n'est pas appuyé et le bouton tribord appuyé.
- Mode automatique : Le bouton STBY est appuyé
- Mode bâbord 1 : code fonction = '0100' Incrémentation de 1° : Le bouton bâbord appuyé (relâché avant la fin de la temporisation) et le bouton tribord relâché.
- Mode tribord 1 : code fonction = '0111' Décrémentation de 1° : Le bouton tribord appuyé (relâché avant la fin de la temporisation) et le bouton bâbord relâché.
- Mode bâbord 10 : code fonction = '0101' Incrémentation de 10° : Le bouton bâbord appuyé jusqu'à la fin de la temporisation et le bouton tribord relâché.
- Mode tribord 10 : code fonction = '0110' décrémentation de 10° : Le bouton tribord reste appuyé jusqu'à la fin de la temporisation et le bouton bâbord relâché.

3.3.1. Entrées /sorties du bloc gestion_bouton

```

*****
-- module gestion des boutons poussoirs
__*****
-- entrées: BP_Babord,BP_Tribord, BP_STBY, clk_50M, raz_n
-- sorties: codeFonction, ledBabord, ledTribord,ledSTBY, out_bip
__*****
--clk_50M: horloge à 50MHz
-- raz_n: actif à 0 => initialise le circuit
-- valeurs de codeFonction:
-- =0000: pas d'action, le pilote est en veille
-- =0001: mode manuel action vérin babord
-- =0010: mode manuel action vérin tribord
-- =0011: mode pilote automatique/cap
-- =0100: incrément de 1° consigne de cap
-- =0101: incrément de 10° consigne de cap
-- =0111: décrément de 1° consigne de cap
-- =0110: décrément de 10° consigne de cap
__*****

```

3.3.2. Bloc gestion_bouton

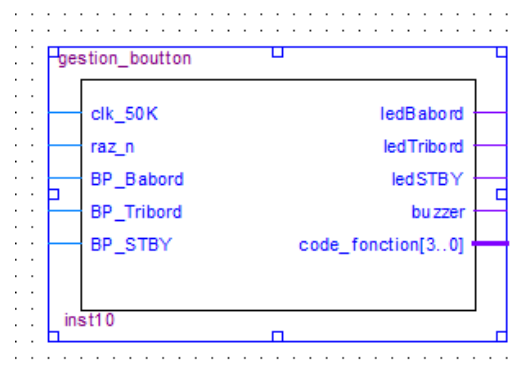


Figure 19 : bloc boutons

3.3.3. Schémas fonctionnel bloc gestion_bouton

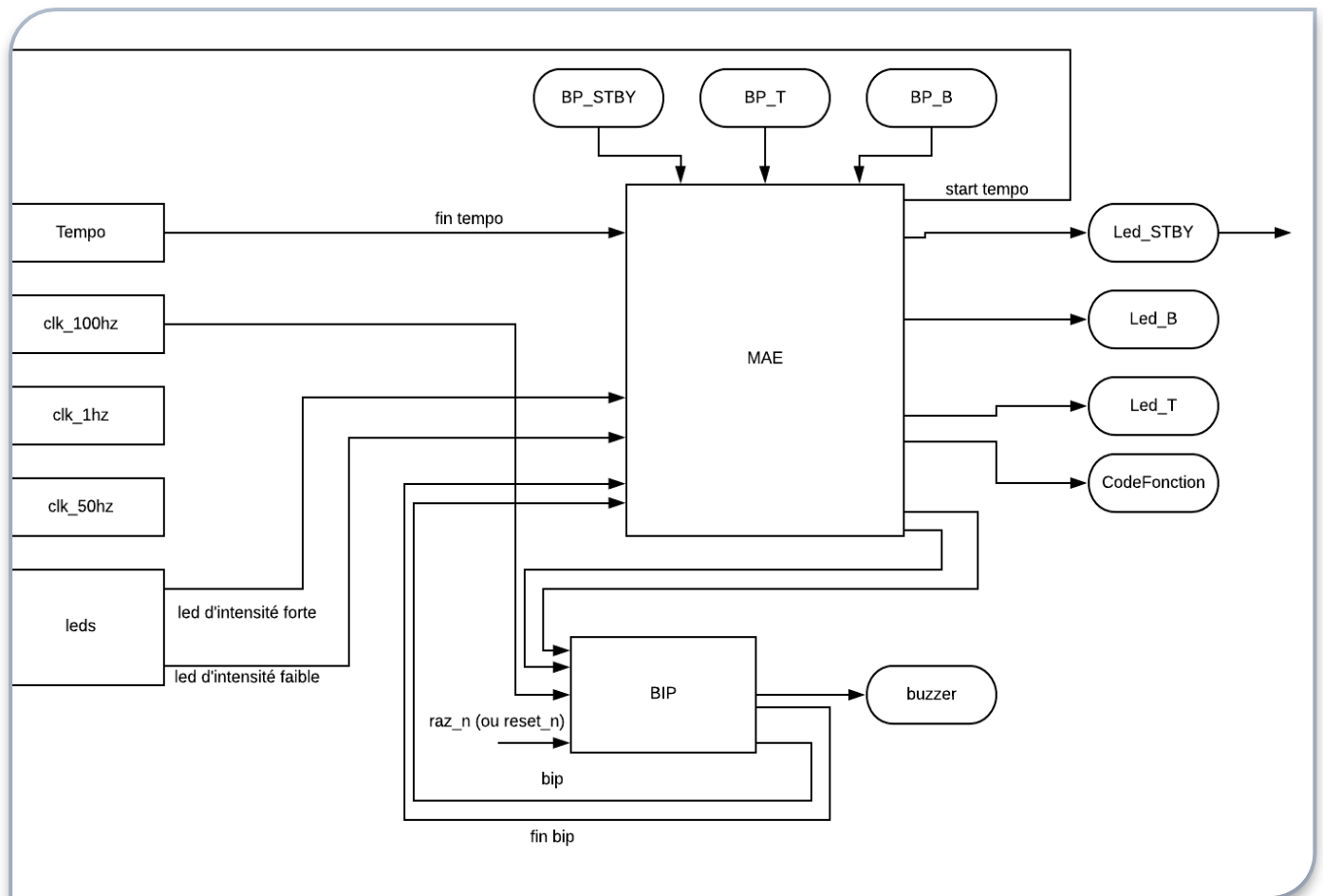


Figure 20 : schéma fonctionnel bloc boutons

3.3.4. Blocs diviseur de fréquence

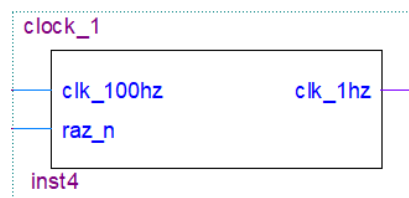


Figure 21 : bloc générateur clock 1hz

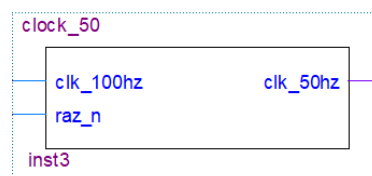


Figure 22 : bloc générateur clock 50hz

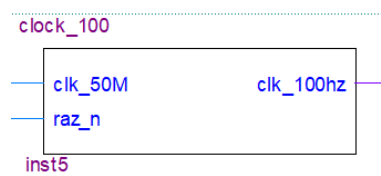


Figure 23 : bloc générateur clock 100hz

- ✓ Bloc clock_1 : Génération de l'horloge 1Hz qui fait clignoter les LEDs.
- ✓ Bloc clock_50 : Génération de l'horloge 50Hz qui allume les LEDs (fréquence de rafraichissement minimum non perceptible par l'œil humain).

- ✓ Bloc clock_100 : Génération de l'horloge 10KHz qui synchronise les différents process.

3.3.5. Bloc machine à états boutons

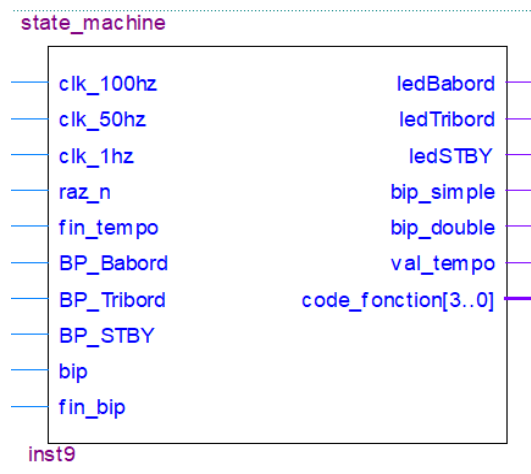


Figure 24 : bloc machine à états boutons poussoir

Ce process assure la transition, synchronisée à 100Hz, entre les différents modes de fonctionnement et états ainsi que la génération du code fonction qui en résulte.

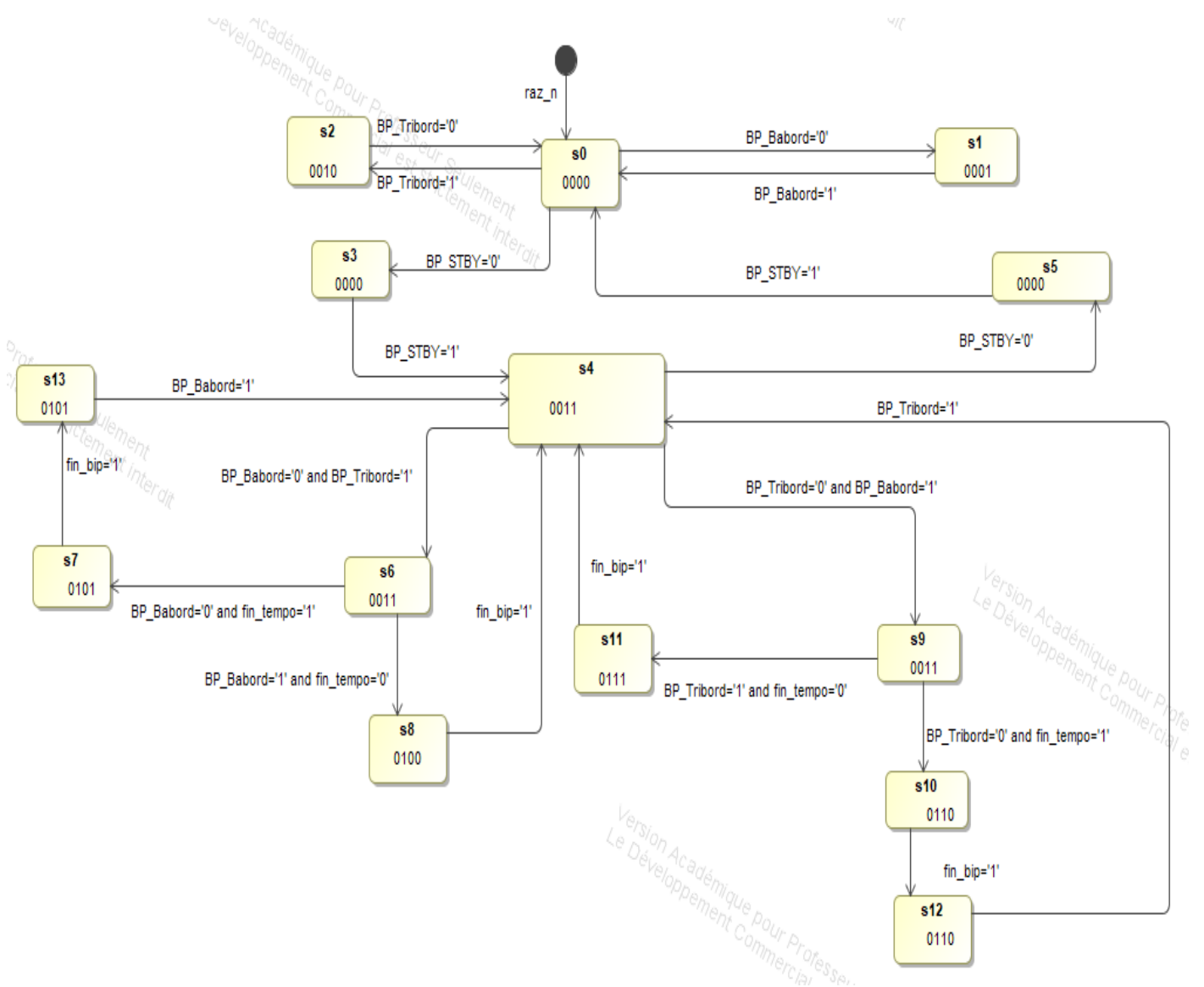


Figure 25 : machine à états boutons poussoir

3.3.6. Bloc génération du tempo

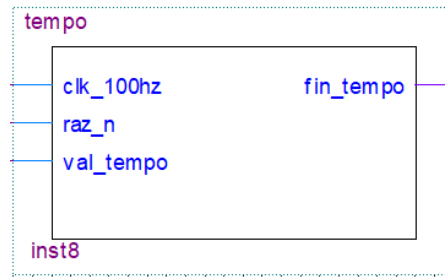


Figure 26 : bloc générateur tempo

Génération de la temporisation de 3 secondes afin de différencier les appuis longs et courts des boutons Bâbord et Tribord.

3.3.7. Bloc génération du bip

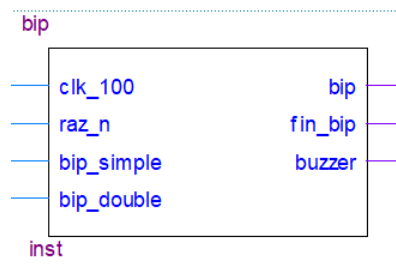


Figure 27 : bloc générateur de bip

Génération du bip court et du bip prolongé en fonction de la durée d'appui sur les boutons.

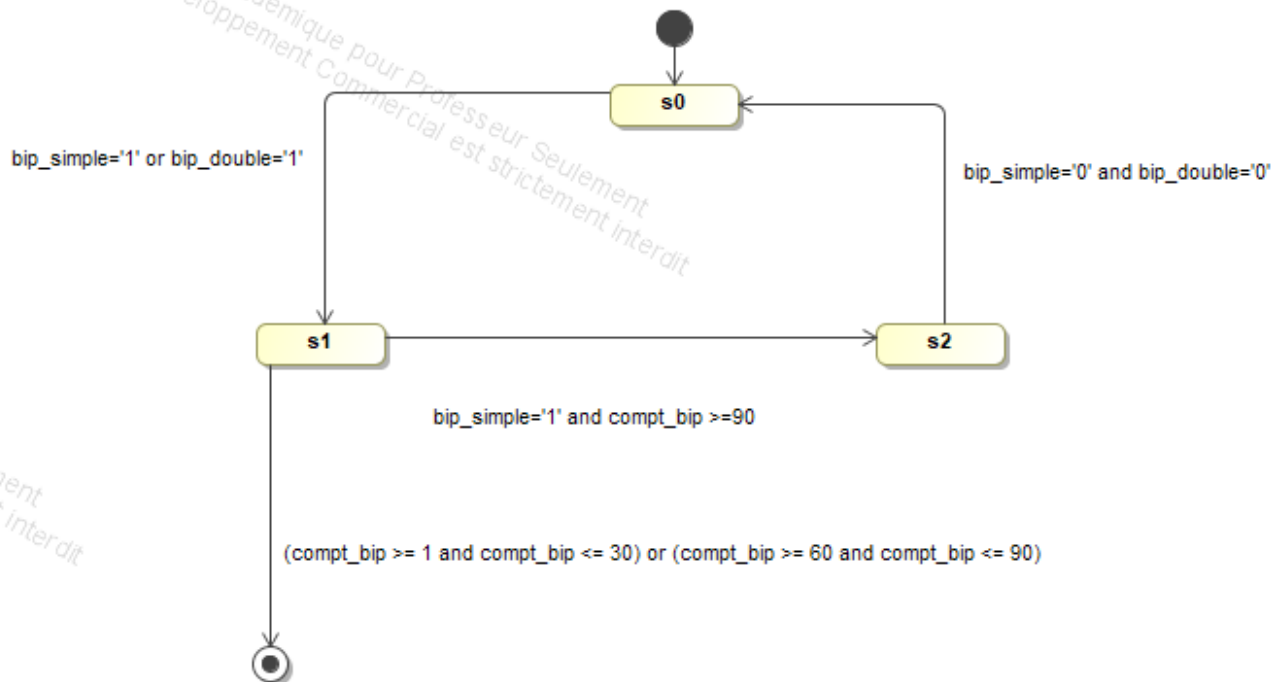


Figure 28 : machine à états génération bip

3.3.8. Simulation Modelsim

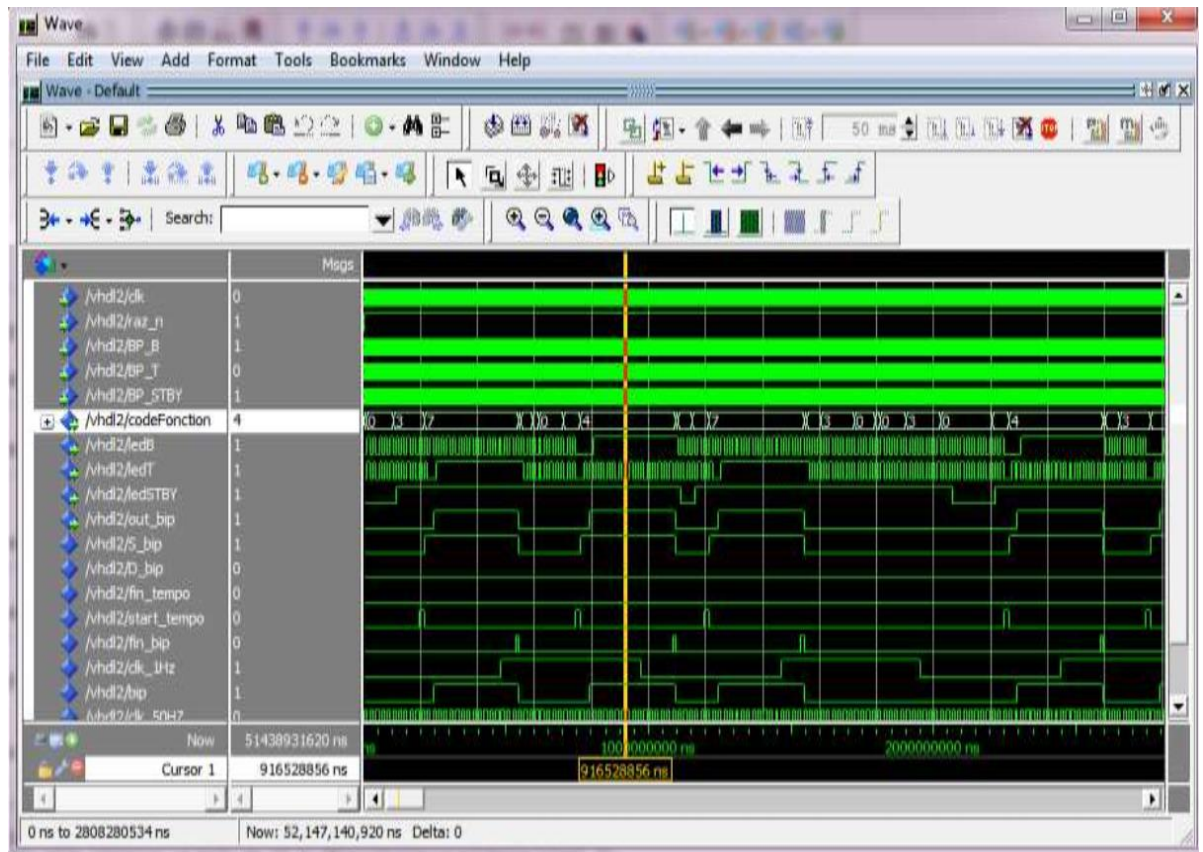


Figure 29 ; Simulation Modelsim gestion boutons

3.4. Interface Avalon et Qsys

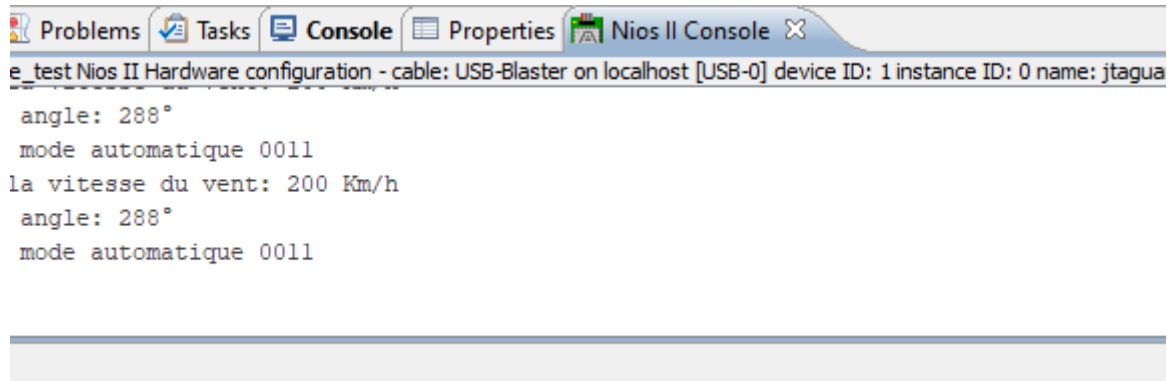
Afin de créer un microprocesseur sur la carte Altera, nous avons utilisé l'outil Qsys intégré dans Quartus 13, cet outil nous permet aussi de créer une mémoire et un Jtag ainsi de créer une version software de nos composants (anémomètre, compas, boutons etc...), ces composants sont connectés entre eux via le bus Avalon ce dernier est un bus de 32 bits développé par Altera.

Dans cette partie nous vous montrons quelques mesures acquises par l'anémomètre et la boussole



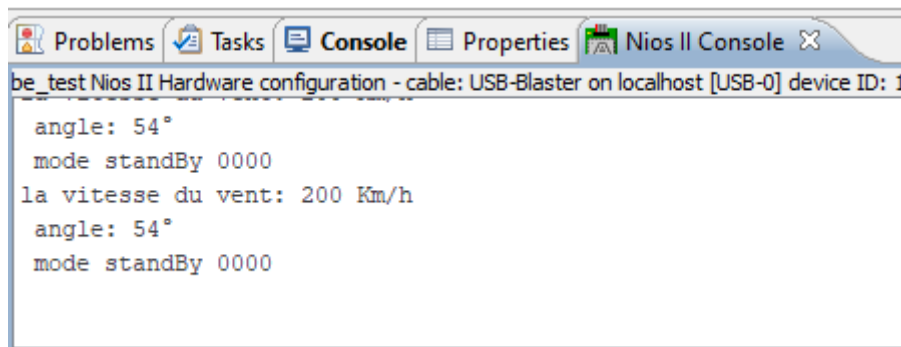
```
be_test Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 i
angle: 92°
mode standBy 0000
la vitesse du vent: 156 Km/h
angle: 92°
mode standBy 0000
```

Figure 32 : Mesures 1



```
be_test Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1 instance ID: 0 name: jtagua
angle: 288°
mode automatique 0011
la vitesse du vent: 200 Km/h
angle: 288°
mode automatique 0011
```

Figure 33 : Mesures 2



```
be_test Nios II Hardware configuration - cable: USB-Blaster on localhost [USB-0] device ID: 1
angle: 54°
mode standBy 0000
la vitesse du vent: 200 Km/h
angle: 54°
mode standBy 0000
```

Figure 34 : Mesures 3

4. Conclusion

Tout au long de ce bureau d'étude, nous avons perfectionné notre capacité d'analyse et d'implémentation de systèmes sur FPGA en suivant le model Top Down, nous avons pu découvrir le Qsys qui est une version puissante que le SOPC ainsi que Modelsim qui est un puissant logiciel de simulation, durant ce Bureau d'étude nous avons réussi à validé les fonctions suivantes : Anémomètre, boussole et boutons poussoir. Par ailleurs on a pas pu réaliser le deuxième niveau implémentation et de test par manque de temps.

Annexe

A. Code VHDL Anémomètre

```
library IEEE;
```

```

use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;
entity gestion_anemometre is
    port (
        clk, chipselect, write_n, reset_n, address : in std_logic;
        in_freq_anemometre : in std_logic;
        writedata : in std_logic_vector (31
downto 0);
        readdata : out std_logic_vector (31
downto 0)
    );
end entity;

architecture arch of gestion_anemometre is
    signal count : integer:=1;
    signal tmp : std_logic := '0';
    signal temp : std_logic_vector ( 7 downto 0);
    signal temp1 : std_logic_vector ( 7 downto 0);
    signal temp2 : std_logic_vector ( 7 downto 0);
    signal clk_1hz : std_logic;
    signal nbr_front : std_logic_vector(7 downto 0);
    TYPE State_type IS (init, cont, mono,acq);
    SIGNAL State : State_Type;
    signal tmp1 : std_logic;
    signal tmp2 : std_logic_vector(7 downto 0);
    signal config : std_logic_vector(2 downto 0);
    --signal code : std_logic_vector(8 downto 0);
    signal start_stop : std_logic;
    signal raz_n : std_logic;
    signal continu : std_logic;
    signal data_anemometre : std_logic_vector(7 downto 0);
    signal data_valid : std_logic;

begin
    -----process diviseur de frequence-----
    -----

    process(clk,raz_n)
    begin
        if(raz_n='0') then
            count<=1;
            tmp<='0';
        elsif(clk'event and clk='1') then
            count <=count+1;
            if (count = 25000000) then
                tmp <= NOT tmp;
                count <= 1;
            end if;
        end if;
    end process;
    clk_1hz <= tmp;

    -----process compteur-----
    -----

    process(Clk_1hz,in_freq_anemometre,raz_n) begin
        if raz_n = '0' then
            temp<= (others=>'0');
        elsif(in_freq_anemometre = '1' and in_freq_anemometre'event) then
            if(Clk_1hz = '1')then
                temp <= temp + 1;
            else temp <= (others => '0');
            end if;
        end if;
    end process;

    process(Clk_1hz,raz_n) begin
        if raz_n = '0' then
            temp2<= (others=>'0');
        elsif(falling_edge(in_freq_anemometre) ) then

```

```

        if(Clk_1hz = '1')then
            temp2 <= temp2 + 1;
        else temp2 <= (others => '0');
        end if;
    end if;
end process;
process(Clk_1hz,raz_n) begin
    if raz_n= '0' then
        temp1<= (others=>'0');
    elsif (falling_edge(clk_1hz))then
        temp1 <= temp+temp2;
    end if;
end process;
Nbr_front <= temp1;
-----process machine à états-----
process(raz_n,clk)
begin
    if (raz_n='0') then
        tmp1 <='0';
        tmp2 <=(others=> '0');
        state <=init;
    elsif rising_edge(clk)then
        case state is
            when init =>
                if (continu='1') then
                    state <= cont;
                else
                    state <= mono;
                end if;
            when cont =>
                if (continu='0') then
                    state <= mono;
                end if;
                tmp1 <= '1';
                tmp2 <= nbr_front;
            when mono =>
                if (continu='1') then
                    state <= cont;
                elsif(start_stop= '1')then
                    state <= acq;
                end if;
                tmp2 <=(others=> '0');
                tmp1 <= '0';
            when acq =>
                if (start_stop= '0')then
                    state <= mono;
                end if;
                tmp1 <= '1';
                tmp2 <= nbr_front;
            when others =>
                state <= init;
        end case;
    end if;
end process;
Data_anemometre <= tmp2;
data_valid <= tmp1;
-----process ecriture avalon-----
raz_n <= config(0);
continu <= config(1);
start_stop <= config(2);

registers: process (clk, reset_n)
begin
    if reset_n = '0' then
        config <= (others => '0');
    end if;
end process;

```

```

elsif clk'event and clk = '1' then
    if chipselect = '1' and write_n = '0' then
        if address = '0' then
            config <= (writedata (2 downto 0));
        end if;
    end if;
end if;
end process registers;
readdata <= X"00000000"&'0'&config when address = '0' else
X"000000"&"000"&data_valid&data_anemometre;

end arch;

```

B. Code VHDL Compas

```

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

entity gestion_compas is
port (
    in_pwm_compas           : in std_logic;
    clk, chipselect, write_n, reset_n, address : in std_logic;
    writedata               : in std_logic_vector (31 downto 0);
    readdata                : out std_logic_vector (31 downto 0)
);
end entity;
architecture arch of gestion_compas is
    signal clk_10K, fin_mesure : std_logic;
    signal sig_duree, data_compas : std_logic_vector(8 downto 0);
    TYPE State_type IS (init, cont, mono, acq);
    SIGNAL State : State_Type;
    signal tmp1 : std_logic;
    signal tmp2 : std_logic_vector(8 downto 0);
    signal config : std_logic_vector(2 downto 0);
    signal start_stop, raz_n, continu : std_logic;
    signal data_valid: std_logic;
begin
    gene_10K: process(clk, raz_n)
        variable count : integer range 0 to 2500;
    BEGIN
        if raz_n= '0' then
            count:=0; clk_10K <= '0';
        elsif rising_edge(clk) then
            count:= count +1;
            if count = 2499 then
                clk_10K <= not clk_10K;
                count:= 0;
            end if;
        end if;
    end process gene_10K;

    duree_etat_haut: process(clk_10K, raz_n)
    BEGIN
        if raz_n= '0' then
            sig_duree <= "0000000000";
        elsif rising_edge(clk_10K) then
            if in_pwm_compas = '1' then
                sig_duree <= sig_duree + "0000000001";
            else sig_duree <= "0000000000";
            end if;
        end if;
    end process duree_etat_haut;

```

```

-----process machine à états-----
process(raz_n,clk)
begin
    if (raz_n='0') then
        tmp1 <='0';
        tmp2 <=(others=> '0');
        state <=init;
    elsif rising_edge(clk) then
        case state is
            when init =>
                if (continu='1') then
                    state <= cont;
                else
                    state <= mono;
                end if;
            when cont =>
                if (continu='0') then
                    state <= mono;
                end if;
                tmp1 <= '1';
                tmp2 <= sig_duree;
            when mono =>
                if (continu='1') then
                    state <= cont;
                elsif(start_stop= '1') then
                    state <= acq;
                end if;
                tmp2 <=(others=> '0');
                tmp1 <= '0';
            when acq =>
                if (start_stop= '0') then
                    state <= mono;
                end if;
                tmp1 <= '1';
                tmp2 <= sig_duree;
            when others =>
                state <= init;
        end case;
    end if;
end process;
data_compas <= tmp2;
data_valid <= tmp1;
-----process ecriture avalon-----
-----
raz_n <= config(0);
continu <= config(1);
start_stop <= config(2);

registers: process (clk, reset_n)
begin
    if reset_n = '0' then
        config <= (others => '0');
    elsif clk'event and clk = '1' then
        if chipselect = '1' and write_n = '0' then
            if address = '0' then
                config <= (writedata (2 downto 0));
            end if;
        end if;
    end if;
end process registers;
readdata <= X"00000000"&'0'&config when address = '0' else
X"000000"&"00"&data_valid&data_compas;
end arch;

```

C. Code VHDL Boutons

```

LIBRARY ieee;

USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;

ENTITY gestion_boutton IS
    PORT (
        clk, chipselect, write_n, reset_n: in std_logic;
        writedata : in std_logic_vector (31 downto 0);
        address : in std_logic;
        readdata : out std_logic_vector (31 downto 0);
        BP_Babord,BP_Tribord, BP_STBY : in std_logic;
        ledBabord, ledTribord,ledSTBY, out_bip : out std_logic
    );
END gestion_boutton ;

ARCHITECTURE arch_gestion_boutton OF gestion_boutton IS
    signal fin_tempo, val_tempo, val_bip, fin_bip, bip_simple, bip_double: std_logic;
    signal codeFonction: std_logic_vector (3 downto 0);
    signal clk_1Hz, bip, clk_50, clk_100, raz_n : std_logic;
    signal compt_bip: integer range 0 to 200;
    signal count_1: integer:=1;
    signal count_50: integer:=1;
    signal count_100: integer:=1;
    signal tmp_1 : std_logic := '0';
    signal tmp_50 : std_logic := '0';
    signal tmp_100 : std_logic := '0';
    signal counter :integer:=1;
    signal led_faible,led_intense : std_logic;
    signal d : std_logic_vector (7 downto 0);
    TYPE State_button IS (s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13);
    TYPE States_bip IS (s0,s1,s2);
begin

--*****
--state machine bouton poussoir
--*****
gestion_bp:process (raz_n, clk_100)
variable State : State_button;
begin
    if raz_n ='0' then
        state:= s0;
        codeFonction <="0000";
    elsif rising_edge(clk_100) then
        case state is
            when s0 =>
                if BP_Babord='0' then
                    state:=s1; codeFonction <="0001";
                end if;
                if BP_Tribord='0' then
                    state:=s2; codeFonction <="0010";
                end if;
                if BP_STBY='0' then
                    state:=s3; codeFonction <="0000";
                end if;
                ledSTBY <= clk_1Hz; ledBabord <= led_faible; ledTribord <= led_faible;
            when s1 =>
                if BP_Babord='1' then
                    state:=s0; codeFonction <="0000";
                end if;
                ledSTBY <= clk_1Hz; ledBabord <= clk_50; ledTribord <= clk_50;
            when s2 =>
                if BP_Tribord='1' then
                    state:=s0; codeFonction <="0000";
                end if;
                ledSTBY <= clk_1Hz; ledBabord <= led_intense; ledTribord <= led_intense;
            when s3 =>
                if BP_STBY='1' then

```



```

state:=s4; codeFonction <="0011";
end if;
ledSTBY <= clk_1Hz; ledBabord <= clk_50; ledTribord <= clk_50;
when s4 =>
if BP_STBY='0' then
state:=s5; codeFonction <="0000";
end if;
if BP_Babord='0' and BP_Tribord='1' then
state:=s6; codeFonction <="0011"; val_tempo <='1';
end if;
if BP_Tribord='0' and BP_Babord='1' then
state:=s9; codeFonction <="0011"; val_tempo <='1';
end if;
ledSTBY <= '1'; ledBabord <= clk_50; ledTribord <= clk_50;
when s5 =>
if BP_STBY='1' then
state:=s0; codeFonction <="0000";
end if;
ledBabord <= clk_50; ledTribord <= clk_50;
when s6 =>
if BP_Babord='0' and fin_tempo='1' then
state:=s7; codeFonction <="0101"; val_tempo <='0'; bip_double<='1';
end if;
if BP_Babord='1' and fin_tempo='0' then
state:=s8; codeFonction <="0100"; val_tempo <='0'; bip_simple<='1';
end if;
ledBabord <= clk_50; ledTribord <= clk_50;
when s7 =>
if fin_bip='1' then
state:=s13; codeFonction <="0101"; bip_double<='0';
end if;
ledBabord <= bip ; ledTribord <= clk_50;
when s8 =>
if fin_bip='1' then
state:=s4; codeFonction <="0011"; bip_simple<='0';
end if;
ledBabord <= bip ; ledTribord <= clk_50;
when s9 =>
if BP_Tribord='0' and fin_tempo='1' then
state:=s10; codeFonction <="0110"; val_tempo <='0'; bip_double<='1';
end if;
if BP_Tribord='1' and fin_tempo='0' then
state:=s11; codeFonction <="0111"; val_tempo <='0'; bip_simple<='1';
end if;
ledBabord <= clk_50; ledTribord <= clk_50;
when s10 =>
if fin_bip='1' then
state:=s12; codeFonction <="0110"; bip_double<='0';
end if;
ledBabord <= clk_50; ledTribord <= bip;
when s11 =>
if fin_bip='1' then
state:=s4; codeFonction <="0011"; bip_simple<='0';
end if;
ledBabord <= clk_50; ledTribord <= bip;
when s12 =>
if BP_Tribord='1' then
state:=s4; codeFonction <="0011"; bip_double<='0';
end if;
ledBabord <= clk_50; ledTribord <= clk_50;
when s13 =>
if BP_Babord='1' then
state:=s4; codeFonction <="0011"; bip_double<='0';
end if;
ledBabord <= clk_50; ledTribord <= clk_50;
end case;
end if;

```

```

    end process gestion_bp;
--*****

--*****
-- génération de la temporisation de 3s
--*****
gen_tempo:process (raz_n, clk_100)
variable duree_tempo : integer range 0 to 300;
begin
    if raz_n ='0' then
        duree_tempo:= 0; fin_tempo <='0';
    elsif rising_edge(clk_100) then
        if val_tempo ='1' then
            duree_tempo:=duree_tempo+1;
            if duree_tempo=300 then duree_tempo:=0;
            fin_tempo <='1';
            end if;
        else duree_tempo:=0;    fin_tempo <='0';
        end if;
    end if;
end process gen_tempo;
--*****

--*****
-- génération 100Hz
--*****
gene_100:    process(clk,raz_n)
BEGIN
    if raz_n= '0' then
        count_100<=1;
        tmp_100<='0';
    elsif rising_edge(clk) then
        count_100<= count_100 +1;
        if (count_100 = 249999 )then
            tmp_100 <= NOT tmp_100;
            count_100 <= 1;
        end if;
    end if;
    clk_100 <= tmp_100;
end process gene_100;
--*****

--génération 1Hz
--*****
gene_1hz : process(Clk_100,raz_n)
begin
    if(raz_n='0') then
        count_1<=1;
        tmp_1<='0';
    elsif rising_edge(clk_100) then
        count_1 <=count_1+1;
        if (count_1 = 49) then
            tmp_1 <= NOT tmp_1;
            count_1 <= 1;
        end if;
    end if;
    clk_1hz <= tmp_1;
end process gene_1hz;
--*****

--génération 50 Hz
--*****
gene_50 : process(Clk_100,raz_n)
begin
    if(raz_n='0') then
        count_50<=1;
        tmp_50<='0';

```

```

    elsif rising_edge(clk_100) then
        count_50 <= count_50 + 1;
        if (count_50 = 2) then
            tmp_50 <= NOT tmp_50;
            count_50 <= 1;
        end if;
    end if;
    clk_50 <= tmp_50;
end process gene_50;

--*****
-- génération du bip et double bip
--*****
double_bip: process (raz_n, bip_simple, bip_double, clk_100)
    variable state_bip : States_bip;
begin
    if raz_n = '0' or (bip_simple = '0' and bip_double = '0') then
        state_bip := s0;
        compt_bip <= 0;
        fin_bip <= '0';
        bip <= '0';
    elsif rising_edge(clk_100) then
        case state_bip is
            when s0 =>
                if bip_simple = '1' or bip_double = '1' then
                    state_bip := s1;
                end if;
            when s1 =>
                compt_bip <= compt_bip + 1;
                if bip_simple = '1' then
                    if compt_bip >= 30 then compt_bip <= 0; fin_bip <= '1';
                    state_bip := s2; bip <= '0';
                end if;
                end if;
                if bip_double = '1' then
                    if compt_bip >= 90 then compt_bip <= 0; fin_bip <= '1';
                    state_bip := s2; bip <= '0';
                end if;
                end if;
                if (compt_bip >= 1 and compt_bip <= 30) or (compt_bip >= 60 and compt_bip <=
90) then
                    bip <= '1';
                    else bip <= '0';
                end if;
            when s2 =>
                if bip_simple = '0' and bip_double = '0' then
                    state_bip := s0;
                end if;
            end case;
        end if;
    end process double_bip;

--*****
-- intensité des leds
--*****
intensite_LED: process (clk_100, raz_n)
begin
    if (raz_n = '0')
        then
            counter <= 0;
            d <= (others => '0');
    elsif rising_edge(clk_100)
        then
            counter <= counter + 1;
            if (counter = 6) then
                counter <= 0;
            end if;
        end if;
    end process;
end process;

```



```

#define freq_compas (unsigned int *)AVALOM_PWM_COMPAS_0_BASE
#define duty_compas (unsigned int *) (AVALOM_PWM_COMPAS_0_BASE+4)
#define control_compas (unsigned int *) (AVALOM_PWM_COMPAS_0_BASE+8)
#define config (unsigned int *)GESTION_ANEMOMETRE_0_BASE
#define code (unsigned int *) (GESTION_ANEMOMETRE_0_BASE+4)
#define cfg (unsigned int*) GESTION_COMPAS_0_BASE
#define data_compas (unsigned int*) (GESTION_COMPAS_0_BASE + 4)
#define buttons ( unsigned int *)GESTION_BOUTTON_0_BASE
#define Code_fonction ( unsigned int *) (GESTION_BOUTTON_0_BASE +4)
volatile unsigned int Code1;

void delay(volatile long unsigned t){
    while(t--);
}
int main()
{
    alt_putstr("Hello from Nios II!\n");
    /*****pwm compas*****/
    *freq_compas=0x1388;// pour avoir une frequence de 10k
    *duty_compas = 0x400;
    *control_compas = 0x3;
    /*****pwm anemometre*****/
    *freq_anemo=0x3D090;// pour avoir une frequence de 200HZ
    *duty_anemo = 0x200;
    *control_anemo = 0x3;
    *config=0x3;
    *cfg=0x3;
    int a,b;
    a = (*code) & 0xFF;
    b = (*data_compas)& 0xFF;
    *buttons=1; //Reset to 1
    /* Event loop never exits. */
    while (1){

        printf ("la vitesse du vent: %d Km/h\n ",a);
        delay(500000);
        *duty_compas=(*duty_compas)+1;
        printf ("angle: %d°\n ",b);
        delay(600000);

        Code1=*Code_fonction;
        if (Code1 == 0)
            printf("mode standBy 0000 \n");
            delay(500000);
        if (Code1 == 1)
            printf("manuel Babord 0001 \n");
            delay(500000);
        if (Code1 == 2)
            printf("manuel tribord 0010 \n");
            delay(500000);
        if (Code1 == 3)
            printf("mode automatique 0011 \n");
            delay(500000);
        if (Code1 == 4)
            printf("increment par 1° babord 0100 \n");
            delay(500000);
        if (Code1 == 5)
            printf("increment par 10° babord 0101 \n");
            delay(500000);
        if (Code1 == 6)
            printf("increment par 1° tribord 0110 \n");
            delay(500000);
        if (Code1 == 7)
            printf("increment par 10° tribord 0111 \n");
            delay(500000);
    }
}

```

```
    return 0;  
}
```