UNIVERSITÉ TOULOUSE III
PAUL SABATIER

Université de Toulouse

Bureau d'étude : Pilote de barre franche

Encadré par :
Mr . *Thierry* PERISSE

Réalisé par:
Aymane CHARRAT
Mouad MOATASSIM BILLAH

M2 SME
2021/2022

Table des matières

Liste des figures

Introduction

L'objectif de ce bureau d'étude est de concevoir le pilote de barre franche sous forme d'un système sur puce programmable SOPC (System On Programmable Chip) décrite à l'aide du langage de description de Hardware VHDL (Very High Speed Hardware Description Langage) en se basant sur l'analyse de spécifications et découpage fonctionnel du système choisi et la conception de circuits d'interfaces numériques en VHDL pour le simuler et le valider sur la maquette, puis faire des interfaçages avec les bus microprocesseur tels que NIOS, Altera, Avalon pour la validation du SOPC en manipulation.

1 Présentation

Un pilote automatique pour voilier est un équipement électrique ou hydraulique destiné à maintenir le cap d'un voilier à la place d'un équipier. Ces pilotes automatiques sont très utiles aux navigateurs solitaires ou en équipage réduit Le pilote est constitué de trois éléments principaux : un compas, une unité électronique et une unité de puissance. Sur tous les pilotes de la nouvelle génération, le compas est électronique, il donne continuellement à l'unité de traitement le cap suivi par le bateau. Cette même unité a comme consigne le cap que l'on souhaite suivre. Elle compare en permanence ces deux caps, s'ils ne sont pas identiques, elle donne l'ordre à l'unité de puissance d'agir sur la barre pour ramener le bateau sur son cap. L'unité de puissance pour les pilotes pour barre franche est un vérin linéaire. Ce vérin a une extrémité fixée sur le banc de cockpit, l'autre sur la barre. Toute variation de cap lui est transmise et il agit en conséquence sur la barre.

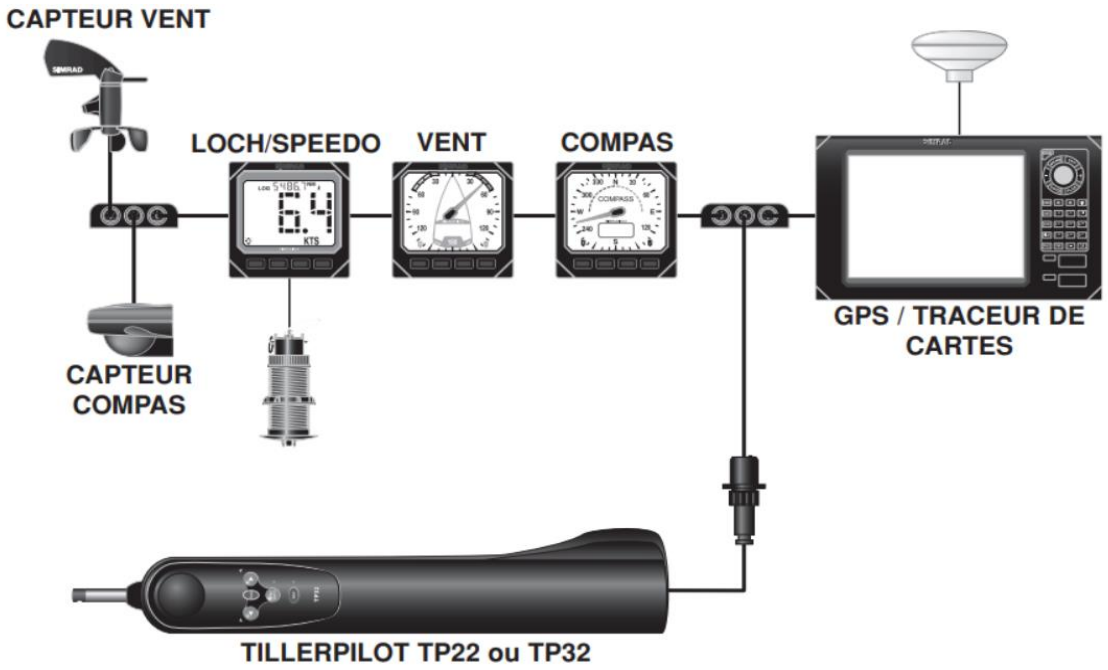


Figure 1 : Présentation de pilote de barre franche.

Le système à réaliser est divisé en sous-systèmes, représenté dans la figure ci-dessous :

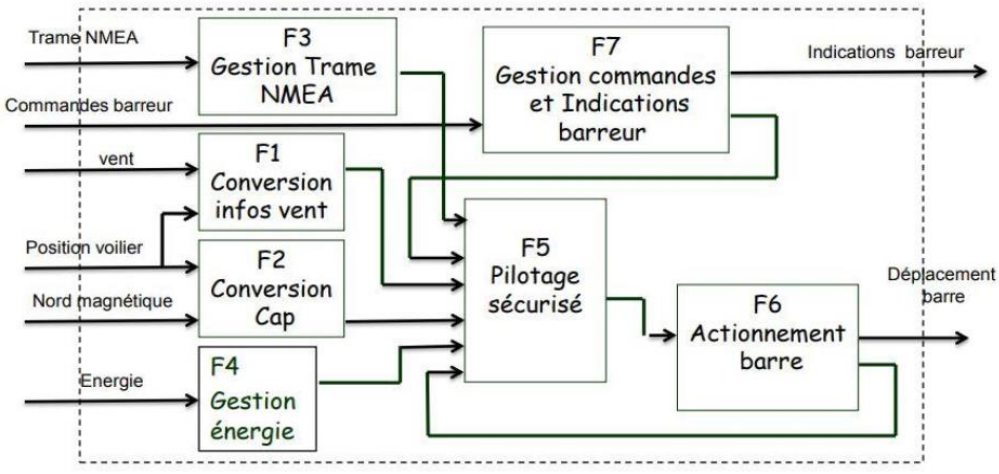


Figure 2 : Les blocs de Pilote de barre franche.

2 Matériels de mise en œuvre

2.1 Carte DE0 – Nano

La carte de développement DE0-Nano et une plateforme de développement FPGA compacte adaptée à la conception de circuits de prototypage tels que les robots et les projets "portables". La carte est conçue pour être utilisée de la façon le plus simple possible ciblant les composants Cyclone IV jusqu'à 22.320 éléments Logiques (LEs). Cette plateforme permet à l'utilisateur d'étendre les conceptions au-delà des cartes DE0-Nano avec deux I/O générales externe (GPIO). L'utilisateur peut gérer un plus grand stockage de données et des tampons grâce à la mémoire internet SDRAM et EEPROM, fournit à l'utilisateur des périphériques améliorés avec des LED et des boutons-poussoirs. La carte est petite et légère, reconfigurable sans nécessiter de matériel superflu, elle est adaptée pour les designs mobiles où la puissance est cruciale, car elle offre trois options de gestion d'alimentation, y compris un USB mini-AB, 2 broches d'alimentation externe et deux broches 5V DC.

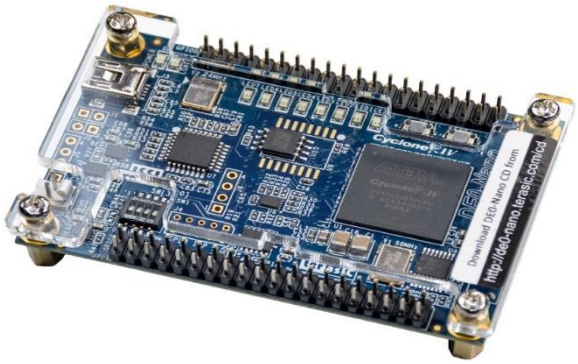


Figure 3 : Carte Altera DE0 – Nano.

2.2 Carte DE2

La carte DE2 comprend un FPGA Cyclone II 2C35 à la pointe de la technologie dans un boîtier à 672 broches. Tout important les composants de la carte sont connectés aux broches de cette puce, permettant à l'utilisateur de contrôler tous les aspects de la carte opération. Pour des expériences simples, la carte DE2 comprend un nombre suffisant d'interrupteurs robustes (des deux à bascule et type à bouton-poussoir), des LED et des affichages à 7 segments. Pour des expériences plus poussées, il existe des SRAM, SDRAM, et des puces de mémoire Flash, ainsi qu'un affichage de 16 x 2 caractères. Pour les expériences qui nécessitent un processeur et simple Interfaces d'E/S, il est facile à instancier Processeur Nios II d'Altera et interface d'utilisation normes telles que RS-232 et PS/2. Pour expériences qui impliquent le son ou la vidéo signaux, il existe des connecteurs standard pour microphone, entrée ligne, sortie ligne (audio 24 bits CODEC), entrée vidéo (décodeur TV) et VGA (DAC 10 bits), Ces fonctionnalités peuvent être utilisé pour créer un son de qualité CD applications et aspect professionnel vidéo. Pour les grands projets de conception, le DE2fournit une connectivité USB 2.0 (à la fois hôte et appareil), Ethernet 10/100, un (IrDA) et une carte mémoire SD connecteur. Enfin, il est possible de connecter d'autres cartes définies par l'utilisateur à la carte DE2. Au moyen de deux en-têtes d'extension.

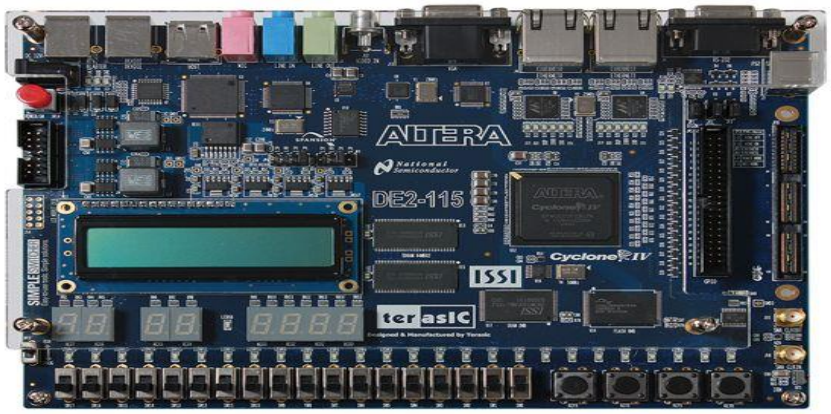


Figure 4 : Carte Altera DE2.

3 Fonctionnement de l'anémomètre

3.1 Présentation de l'anémomètre

L'anémomètre permet une mesure du vent. Le signal de sortie de ce capteur est un signal logique de fréquence variable. Cette fréquence qui varie de 0 à 250 Hz et dont la correspondance de la vitesse est de 0 à 250 Km/h.

Figure 5 : L'anémomètre.

3.2 L'analyse fonctionnelle

Pour répondre bien aux exigences de notre circuit à concevoir, nous avons réalisé la description fonctionnelle suivant :

Diviseur : C'est un bloc qui permet de générer une horloge de 1 khz afin de synchroniser les différents process de notre circuit.

Détecteur des front montants : C'est un bloc qui permet de détecter les fronts montants du signal divisé par le diviseur afin de mettre sa sortie à 0 s'il ne détecte pas des fronts montants, et à 1 dans le cas contraire.

Compteur : C'est un bloc qui permet de compter le nombre des fronts montant du signal numérique dans le but de mesurer la fréquence.

Choix_mode : C'est une fonction qui gère les modes de mesure de la fréquence in_freq, qui sera mémorisée dans une variable de sortie codé de 8 bits nommé data_ anemometre, lorsqu'une mesure est valide, le circuit met sa sortie data_anemo et data valid.

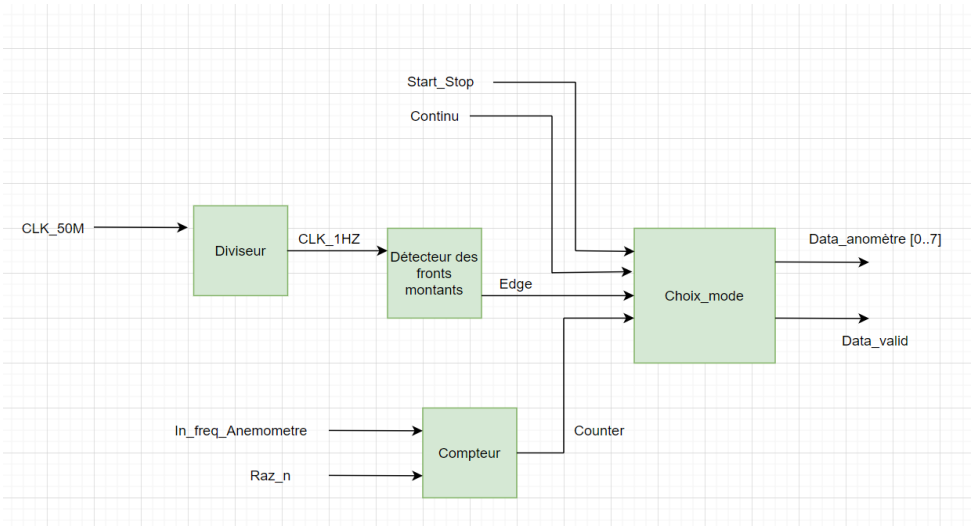


Figure 6 : L'analyse fonctionnelle d'anémomètre.

3.3 Implémentation

Après avoir réalisé l'analyse fonctionnelle de notre circuit nous avons basculé vers Quartus pour implémenter notre système par des blocs en VHDL comme il est indiqué dans la figure ci-dessous :

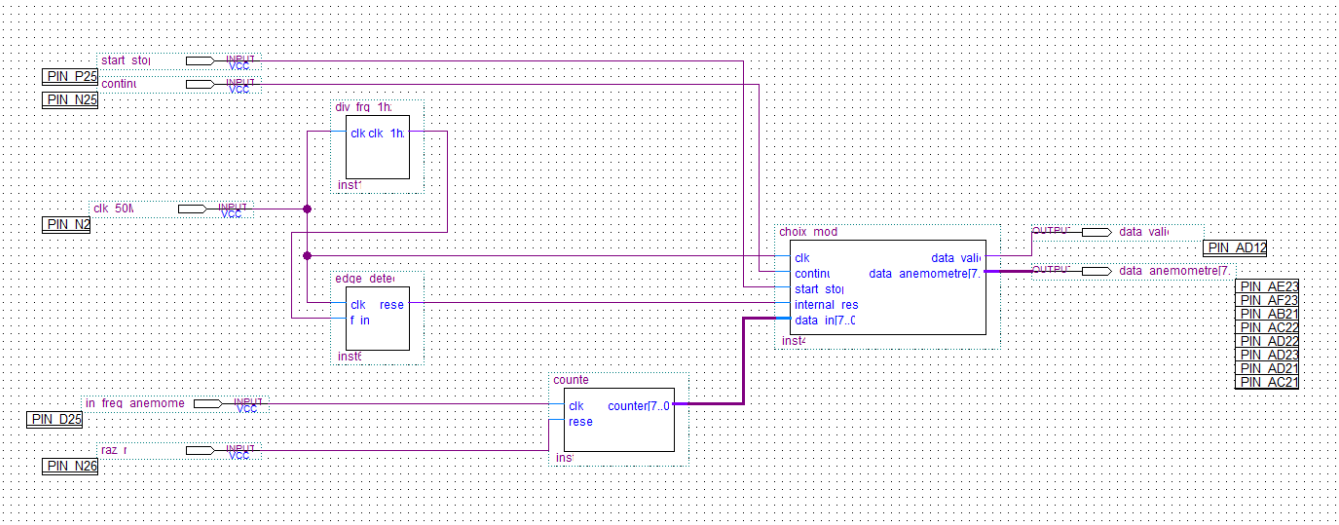


Figure 7 : Implémentation d'anémomètre sur Quartus.

Avant de manipuler le fonctionnement de notre système nous avons lancé une simulation sur Quartus 9 comme il est indiqué dans la figure 8 :

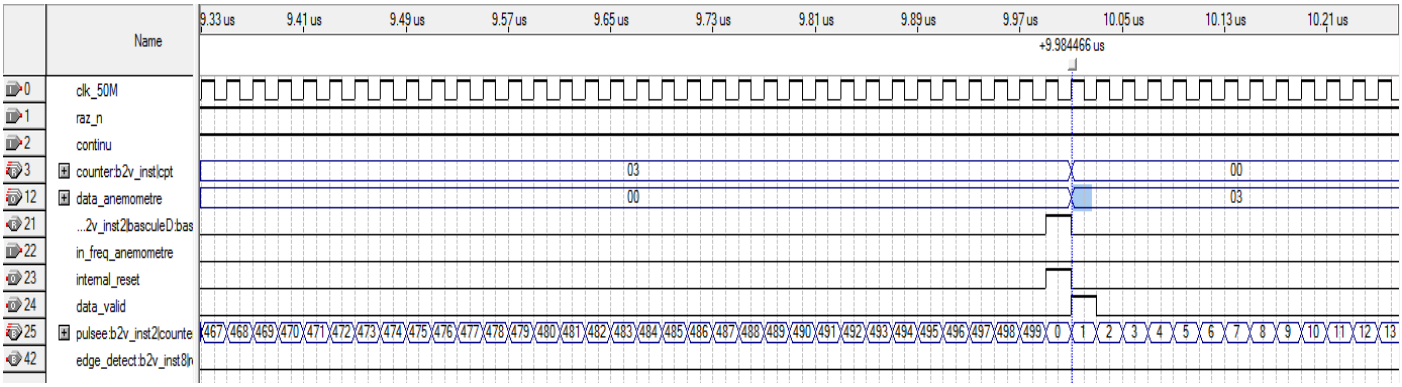


Figure 8 : Simulation d'anémomètre sur Quartus 9.

Pour valider le fonctionnement de code VHDL, nous avons visualisé les signaux sur l'oscilloscope Sur les deux photos ci-dessus, nous poubons voir en jaune l'évolution de DATA_VALID dans le temps. Sur l'image de gauche on voit l'utilisateur en mode monocoup par le biais d'un appui sur un bouton poussoir. Sur l'image de droite, on voit l'utilisation en mode continu avec le signal START_STOP évoluant à une fréquence de 1Hz.

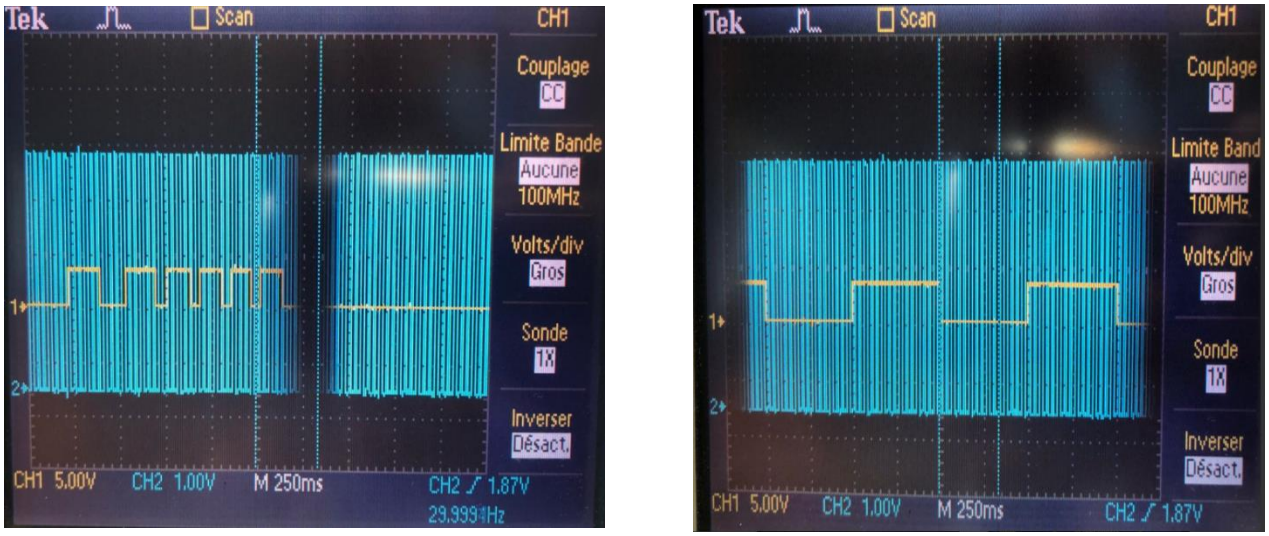
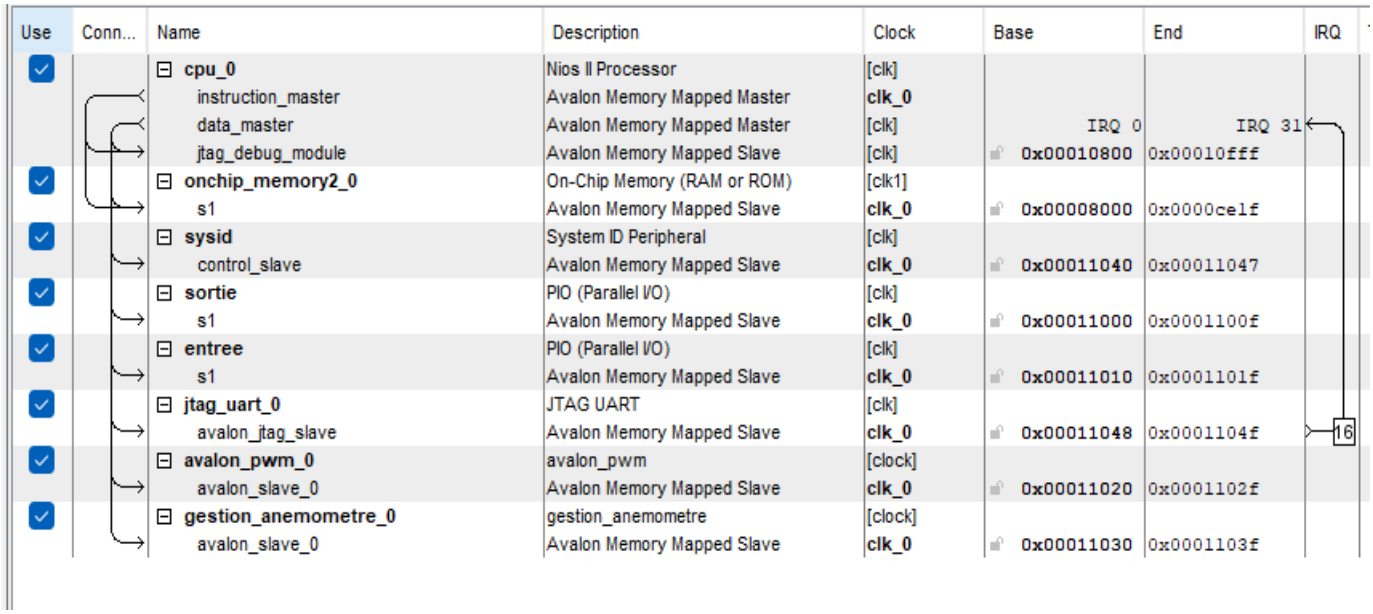


Figure 9 : Validation sur l'oscilloscope

3.4 Conception du SOPC

Afin de télécharger le circuit sur la carte DE0 nous avons utilisé le SOPC Builder afin de créer le microprocesseur et les éléments périphériques en intégrant l'anémomètre et Avalon PWM comme il est indiqué dans les figures ci-dessous :



Use	Conn...	Name	Description	Clock	Base	End	IRQ
☑		⊟ **cpu_0**	Nios II Processor	[clk]			
		instruction_master	Avalon Memory Mapped Master	**clk_0**			
		data_master	Avalon Memory Mapped Master	[clk]	IRQ 0	IRQ 31	
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00010800	0x00010fff	
☑		⊟ **onchip_memory2_0**	On-Chip Memory (RAM or ROM)	[clk1]			
		s1	Avalon Memory Mapped Slave	**clk_0**	0x00008000	0x0000ce1f	
☑		⊟ **sysid**	System ID Peripheral	[clk]			
		control_slave	Avalon Memory Mapped Slave	**clk_0**	0x00011040	0x00011047	
☑		⊟ **sortie**	PIO (Parallel I/O)	[clk]			
		s1	Avalon Memory Mapped Slave	**clk_0**	0x00011000	0x0001100f	
☑		⊟ **entree**	PIO (Parallel I/O)	[clk]			
		s1	Avalon Memory Mapped Slave	**clk_0**	0x00011010	0x0001101f	
☑		⊟ **jtag_uart_0**	JTAG UART	[clk]			
		avalon_jtag_slave	Avalon Memory Mapped Slave	**clk_0**	0x00011048	0x0001104f	16
☑		⊟ **avalon_pwm_0**	avalon_pwm	[clock]			
		avalon_slave_0	Avalon Memory Mapped Slave	**clk_0**	0x00011020	0x0001102f	
☑		⊟ **gestion_anemometre_0**	gestion_anemometre	[clock]			
		avalon_slave_0	Avalon Memory Mapped Slave	**clk_0**	0x00011030	0x0001103f	

Figure 10 : Conception sur SOPC Builder.

3.5 Circuit d'interface Avalon/anémomètre

Après avoir construit notre système en SOPC Builder et intégré l'Avalon PWM et l'anémomètre dedans, nous avons obtenu l'interfaçage qui est indiqué dans la figure ci-dessous :

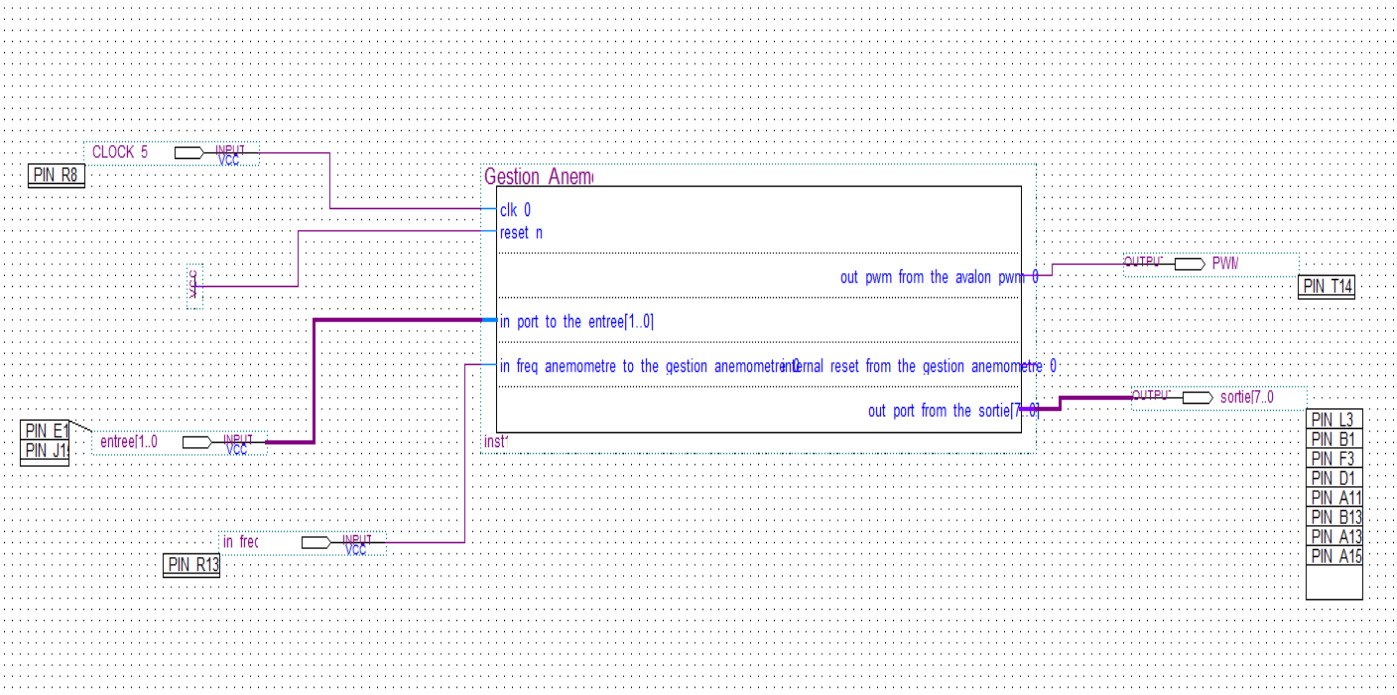


Figure 11 : Circuit d'interface Avalon/anémomètre.

3.6 Validation des résultats obtenus

Pour valider le fonctionnement de notre programme en C nous avons utilisé le terminal. Sur les photos ci-dessus, nous pourrons voir la donnée brute d'un octet correspondant à **donnees_anemo** et la vitesse en km/h et les modes d'envoie des données.

.

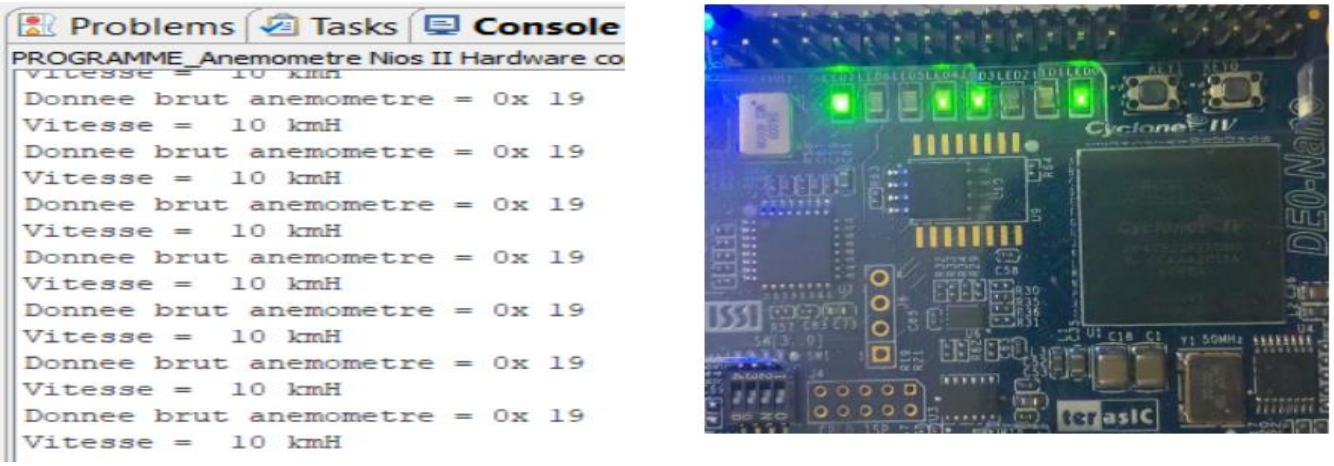


Figure 12 : Manipulation et validation d'anémomètre.

4 Fonctionnement des boutons poussoirs et des leds

4.1 Conception du SOPC

Afin de télécharger le circuit sur la carte DE0 nous avons utilisé le SOPC Builder afin de créer le microprocesseur et les éléments périphériques en intégrant l'Avalon PWM comme il est indiqué dans les figures ci-dessous :

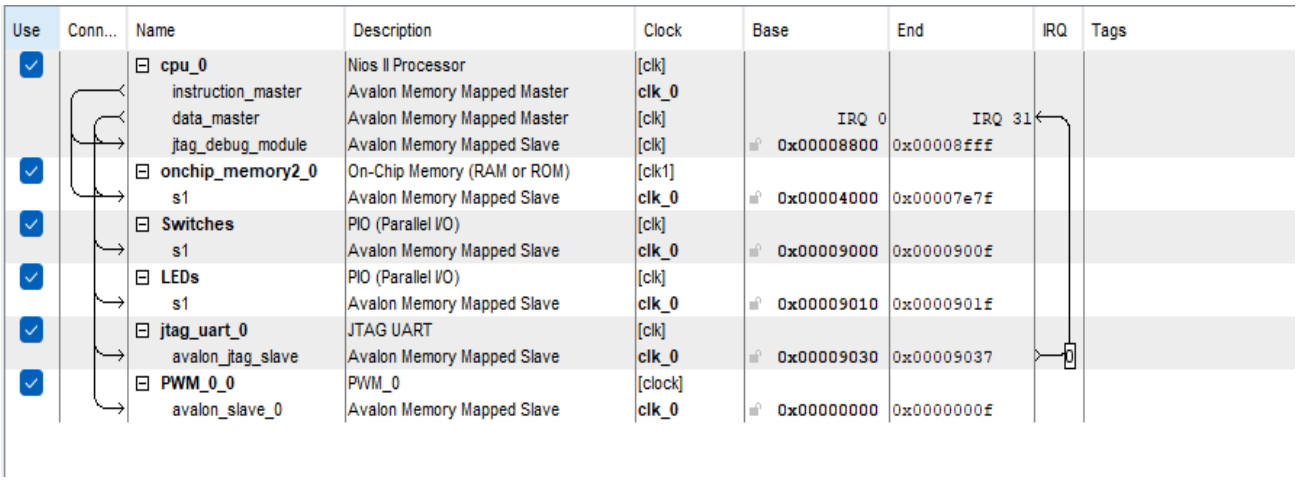


Figure 13 : Conception sur SOPC Builder.

4.2 Circuit d'interface des boutons poussoirs et les leds

Après avoir construit notre système en SOPC Builder et intégré l'Avalon PWM nous avons obtenu l'interfaçage qui est indiqué dans la figure ci-dessous :

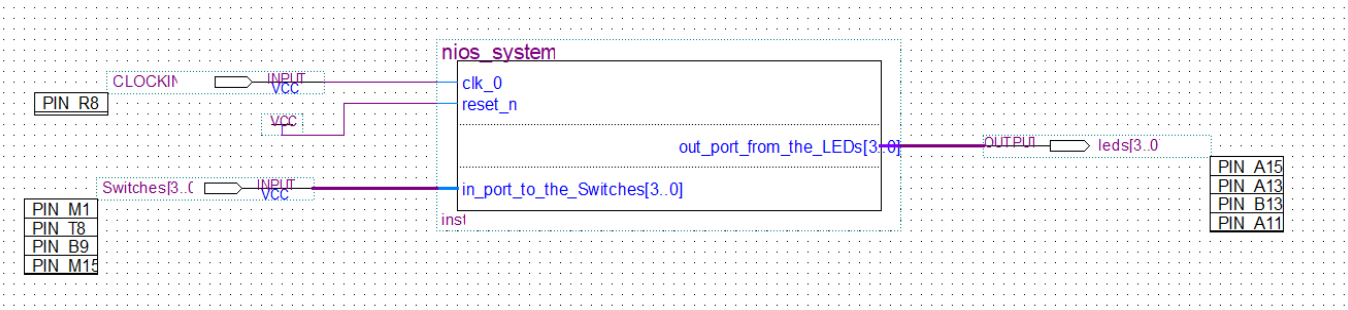


Figure 14 : Circuit d'interface du vérin.

5 Fonctionnement de vérin

5.1 Présentation de vérin

La fonction réalisant le mouvement de la barre franche est réalisée avec le circuit vérin. Ce circuit est composé de 3 fonctions principales qui vont permettre le pilotage du vérin qui contrôle la barre franche du voilier.

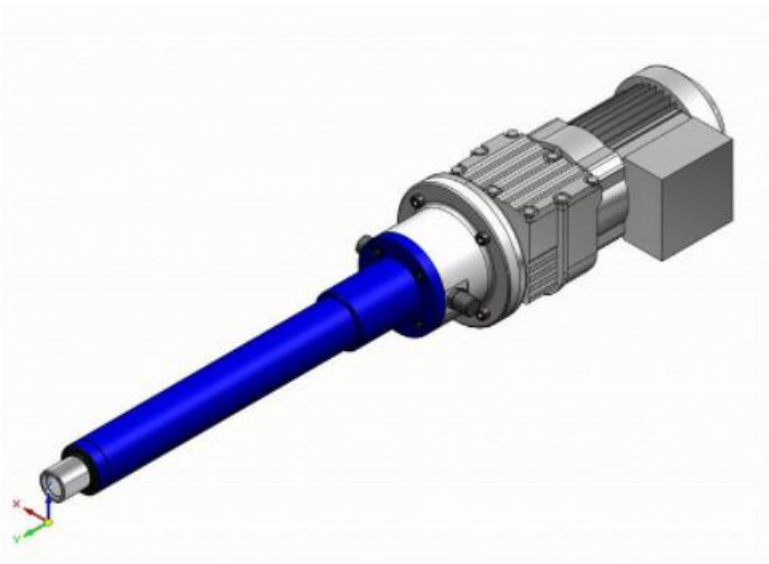


Figure 15 : Vérin.

5.2 L'analyse fonctionnelle

Pour répondre bien aux exigences de notre circuit à concevoir, nous avons réalisé la description fonctionnelle suivant :

Gestion de PWM : C'est un bloc qui fixe la fréquence de la PWM et le rappel cyclique afin de générer un signal PWM en sortie.

Contrôle des butées : C'est un bloc qui permet de mettre le signal PWM à zéro si l'angle de la barre dépasse les valeurs de seuls qui génère des signaux de fin de course.

Gestion du convertisseur AN MCP 3201 : C'est un bloc qui fait appel à cinq fonctions secondaires :

1. ***Machine à état*** : pour pilote l'ADC et mémoriser la donnée de l'angle de la barre.
2. ***Compteur*** : qui compte les fronts d'horloge de clk_adc.
3. ***Registre à décalage*** : qui permet de récupérer la donnée du convertisseur.
4. ***Générateur du 1MHZ*** : qui permet de démarrer la machine à état.
5. ***Générateur périodique*** : qui permet de générer des périodes toutes les 100ms pour commencer la conversion.

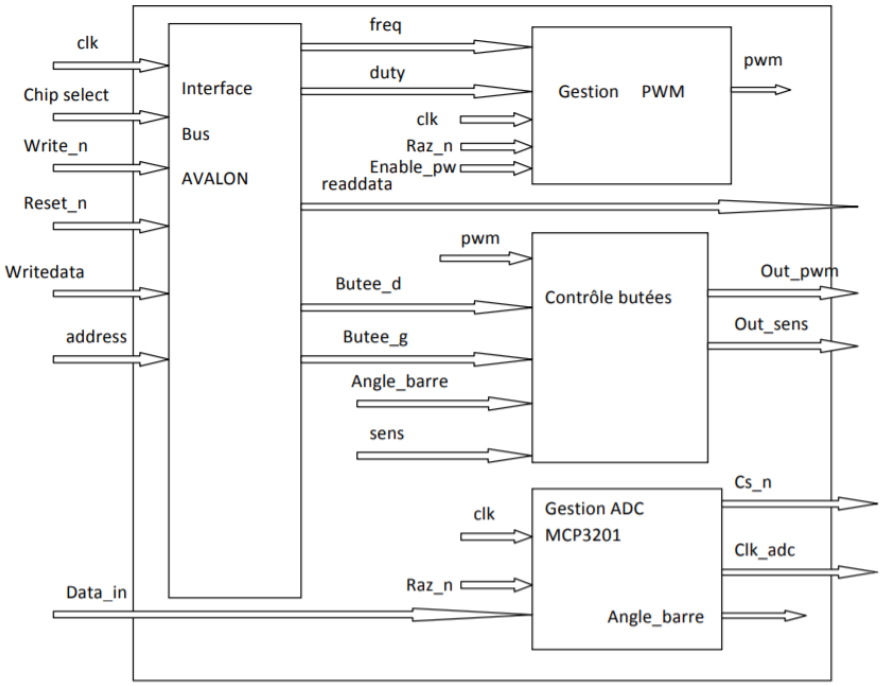


Figure 16 : L'analyse fonctionnelle de vérin.

5.3 Fonctionnement du convertisseur

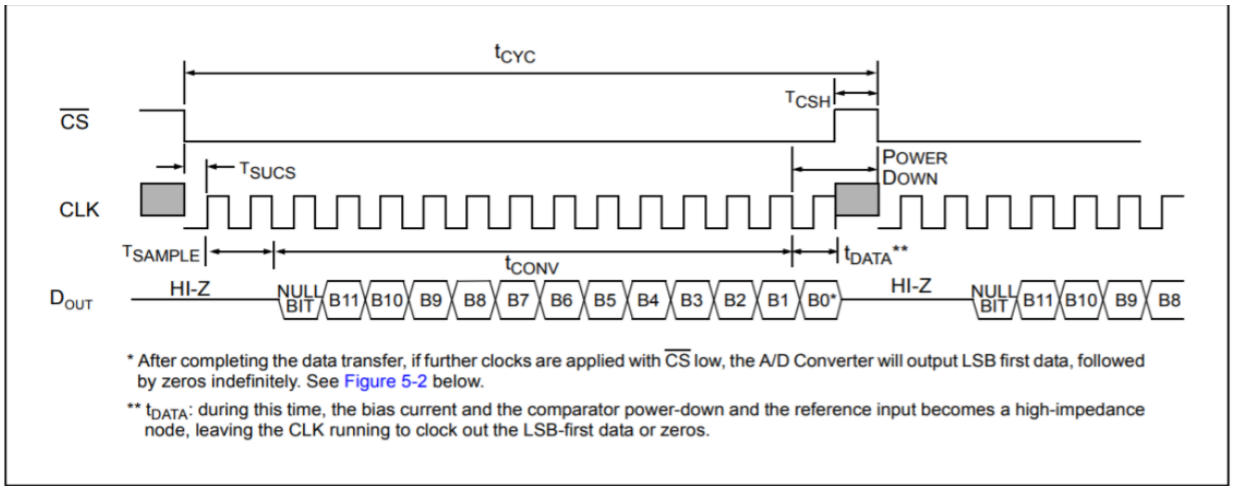


Figure 17 : Extrait de la datasheet du convertisseur

Comme nous peuvons le voir sur la figure ci-dessus, le convertisseur utilise trois signaux pour fonctionner :

- 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

Dès que l'on veut effectuer une acquisition d'une mesure, le FPGA activera le CS du convertisseur en le plaçant à l'état bas. Ensuite le FPGA devra attendre que le convertisseur traite la donnée analogique convertir. Cette durée est égale à 2 coups d'horloges. Ensuite le FPGA pourra lire les 12 bits de données utiles qui sortiront de la broche Dout.

5.4 Conception en SOPC Builder

Afin de télécharger le circuit sur la carte DE0 nous avons utilisé le SOPC Builder que nous avons créé en ajoutant le vérin et le compas et les boutons poussoirs comme il est indiqué dans la figure ci-dessous :

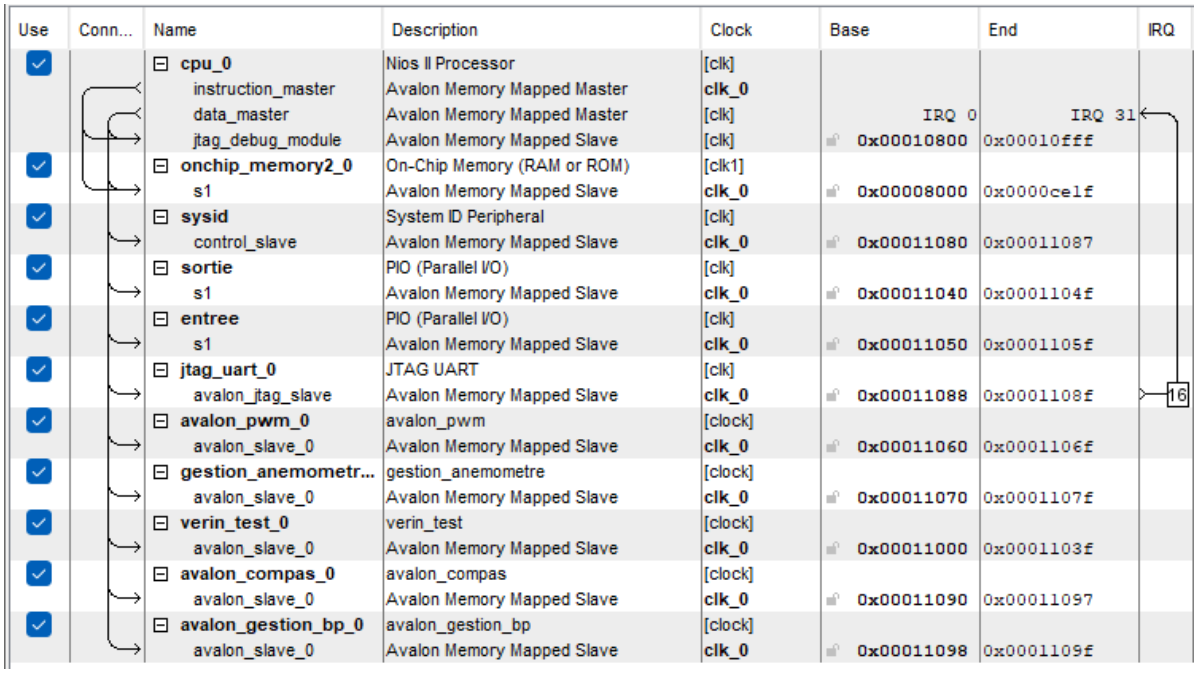


Figure 18 : Conception sur SOPC Builder.

5.5 Circuit d'interface du vérin

Après avoir construit notre système en SOPC Builder et le vérin dedans, nous avons obtenu l'interfaçage qui est indiqué dans la figure ci-dessous

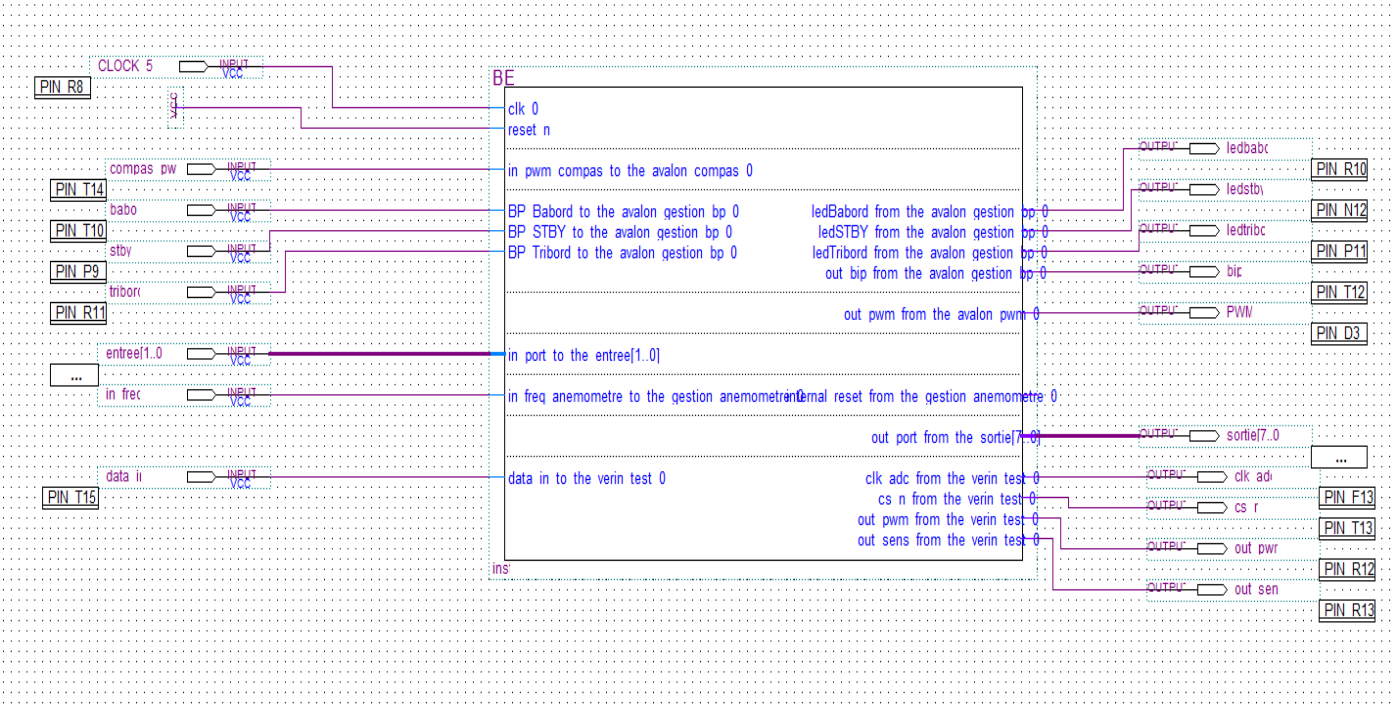


Figure 19 : Circuit d'interface

5.6 Test et validation

Premièrement nous avons visualisé sur l'oscilloscope les signaux CS et Data_adc afin de vérifier si la conversion fonctionne comme il faut.

Figure 20 : Visualisation des signaux CS et Data_ADC

Pour valider le fonctionnement de notre programme en C nous avons utilisé le terminal. Sur les photos ci-dessus. Nous pourrons regarder les résultats obtenus pour le mode automatique quand nous l'activons après avoir changé la position du pilote le vérin doit retourner dans sa position normale, et le mode manuel.

```
Anémo data = 0 | config = 200
freq= 2000 | duty= 1500 | b_g= 1334 | b_d= 2882 | config= 1 | angle_b= 2456
compas= 246 | compass_config = 3
code_fonction= 3 | bp_config = 3
Compass_ref= 248 | angle_barre_ref = 2315 | Angle_corr = 2331 | new_auto = 0
```

```
Anémo data = 0 | config = 200
freq= 2000 | duty= 1500 | b_g= 1334 | b_d= 2882 | config= 1 | angle_b= 2463
compas= 243 | compass_config = 3
code_fonction= 0 | bp_config = 1
Compass_ref= 244 | angle_barre_ref = 2465 | Angle_corr = 2465 | new_auto = 1
```

Figure 21 : Les résultats obtenus

Nous avons réalisé trois tests pour valider le bon fonctionnement du projet. Pour le premier test nous avons utilisé un GBF qui nous a permis d'injecter directement sur le GPIO, du FPGA attribué, une fréquence qu'on peut modifier pour venir simuler la sortie de l'anémomètre. Après observation la fréquence est bien actualisée toutes les secondes comme demandé dans le cahier des charges. Pour le second test nous venons tester la fonction gestion vérin avec la fonction interface Homme système les boutons nous on permit de tester le bon fonctionnement qui contrôle le signal PWM du vérin. Pour la dernière fonction qui est l'asservissement du système complet nous observons un maintien du cap définit dans le logicielle téléversé sur la maquette de développement et que celle-ci respecte bien le cap fixé avant chaque passage en mode automatique.

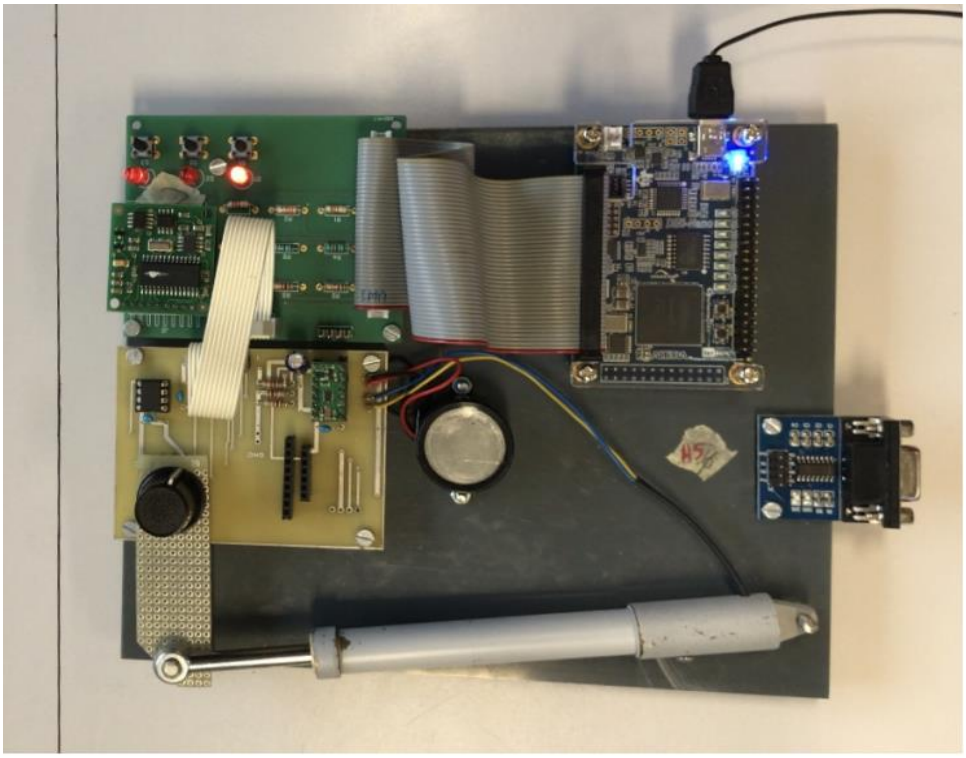


Figure 22 : Validation du projet sur la maquette

6 Annexes :

6.1 Gestion anémomètre

6.1.1 Bloc diviseur de fréquence :

```
1
2  library IEEE;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use IEEE.numeric_std.ALL;
5
6  entity div_frq_1hz is
7  port ( clk: in std_logic;
8  clk_1hz : out std_logic);
9  end div_frq_1hz;
10
11 architecture bhv of div_frq_1hz is
12 begin
13
14 process(clk)
15 variable cpt : integer range 0 to 11;
16 begin
17
18 if(clk'event and clk='1') then
19 cpt := cpt + 1;
20 if cpt < 5 then clk_1hz <= '1';
21                   else if (cpt >= 5 and cpt < 10) then clk_1hz <= '0';
22                             else if cpt = 10 then cpt := 0;
23                                       end if;
24                             end if;
25                   end if;
26 end if;
27 end process;
28 end bhv;
29
```

6.1.2 Bloc de détecteur des fronts

```
1  Library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity edge_detect is
6          port(
7                    clk, f_in : in std_logic;
8                    reset : out std_logic);
9  end edge_detect;
10
11 architecture arc of edge_detect is
12         signal etat : std_logic_vector (1 downto 0) := "00";
13 begin
14         process(clk) is
15         begin
16                   if rising_edge(clk) then
17                             if etat = "00" then
18                                       if f_in = '1' then
19                                                 etat <= "11";
20                                                 reset <= '1';
21                                       end if;
22                             end if;
23                             if etat = "11" then
24                                       etat <= "10";
25                                       reset <= '0';
26                             end if;
27                             if etat = "10" then
28                                       if f_in = '0' then
29                                                 etat <= "00";
30                                                 reset <= '0';
31                                       end if;
32                             end if;
33                   end if;
34         end process;
35
36 end arc;
```

6.1.3 Bloc du choix de mode

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.numeric_std.ALL;

entity choix_mode is
port (   clk, continu, start_stop, internal_reset: in std_logic;
                  data_in : in std_logic_vector (7 downto 0);
                  data_valid : out std_logic;
                  data_anemometre : out std_logic_vector (7 downto 0));
end choix_mode;

architecture bhv of choix_mode is

signal tmp, tmp1 : std_logic_vector (7 downto 0);

begin
process(clk)

begin

if(clk'event and clk='1') then
         if continu = '1' then
                  if internal_reset = '1' then
                           data_anemometre <= tmp;
                           data_valid <= '1';
                  else
                           tmp <= data_in;
                           data_valid <= '1';
                  end if;
         else
                  if start_stop = '1' then
                           data_anemometre <= tmp;
                           data_valid <= '1';
                  else
                           data_anemometre <= "00000000";
                           data_valid <= '0';
                           if internal_reset='0' then
                                    tmp1 <= data_in;
                           else
                                    tmp <= tmp1;
                           end if;

                  end if;
         end if;

end if;
end process;
end bhv;
```

6.1.4 Bloc du compteur

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity counter is
        port(
                clk : in std_logic;
                reset : in std_logic;
                counter : out unsigned (7 downto 0));
end counter;

architecture arc of counter is

begin

        process(clk, reset) is
                variable cpt : unsigned (7 downto 0);
        begin
        if reset = '1' then
     counter <= "00000000";
     cpt := "00000000";
        elsif rising_edge(clk) then
     cpt := cpt + 1;
     counter <= cpt ;

        end if;
        end process p_asynchronous_reset;
end arc;
```

6.1.5 Développement sur NIOS

```
int main()
{
  alt_putstr( "Bonjour de Nios II !\n" );
    *freq= 0x00000400 ;
    *devoir= 0x00000200 ;
    *contrôle = 3 ;
    alt_putstr( " salut 2 !\n" );
    *config = 7 ;
    résultat int = 0 ;
    données int = 0 ;
    int i = 0xFF ;
    while( 1 ){
    resultat = *code;
    données = i & resultat;
    printf( "données %d \n" , données);
    }
  renvoie  0 ;
}
```

6.2 Gestion vérin, compas, bouton poussoir, anémomètre

6.2.1 Bloc du shift registre

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity shift_register is
        Port (
                reg : out std_logic_vector(11 downto 0);
                sig : in std_logic;
                cs_n : in std_logic;
                clk_1mhz : in std_logic
        );
end shift_register;

architecture behavioral of shift_register is
        signal shift_reg, shift_next : std_logic_vector(11 downto 0);
        signal adc_data_reg : std_logic_vector(11 downto 0);


begin

        process(clk_1mhz)
        begin
                if (clk_1mhz'event and clk_1mhz='0') then
                        shift_reg <= shift_next;
                end if;
        end process;
        shift_next <= shift_reg(10 downto 0) & sig;

        process(clk_1mhz, cs_n)
        begin
                if (clk_1mhz'event and clk_1mhz='1') then
                                        if cs_n = '1' then
                                        adc_data_reg <= shift_reg;
                                        end if;
                end if;
        end process;
        reg <= adc_data_reg;
end behavioral;
```

6.2.2 Bloc générateur de CN

```
Library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity cs_n_generator is
        port(
                clk_1mhz : in std_logic;
                cs_n : out std_logic);
end cs_n_generator;

architecture arc of cs_n_generator is
begin

process(clk_1mhz) is
        variable cpt : integer range 0 to 100000;
begin

        if(clk_1mhz'event and clk_1mhz='1') then
                cpt := cpt + 1;
                if cpt < 99995 then
                        cs_n <= '0';
                else
                        cs_n <= '1';
                        cpt := 0;
                        end if;
                end if;
end process;
end arc;
```

6.2.3 Bloc du contrôle des butées

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3  use IEEE.std_logic_unsigned.all;
4
5
6  entity control_butee is
7  port (
8           pwm_in, sens, enable: in std_logic;
9           butee_g, butee_d : in std_logic_vector (15 downto 0);
10          angle_barre : in std_logic_vector (11 downto 0);
11          out_pwm, out_sens, f_g, f_d : out std_logic
12 );
13 end entity;
14
15 ARCHITECTURE arch of control_butee IS
16
17
18 BEGIN
19 process (angle_barre)
20
21 begin
22          if ( (angle_barre > butee_g) and sens = '0' and enable ='1') then
23                    out_pwm <= '0';
24                    f_g <= '1';
25          elsif (angle_barre < butee_d and sens = '1' and enable ='1') then
26                    out_pwm <= '0';
27                    f_d <= '1';
28                    else
29                              out_pwm <= pwm_in;
30                              f_d <= '0';
31                              f_g <= '0';
32          end if;
33 end process;
34          out_sens <= sens;
35 END arch ;
```

6.2.4 Développement sur NIOS

```
  while(1){

          // Anémometre
          resultat = *code;
          data = i & resultat;
          printf("Anémo data = %d | config = %X \n", data, *config);


          //Verin
          f = *v_duty_freq & 0x0000FFFF;
          d = (*v_duty_freq >> 16);
          l = *v_butee_d_g & 0x0000FFFF;
          r = (*v_butee_d_g >> 16);
          printf("freq= %d | duty= %d | b_g= %d | b_d= %d | config= %d | angle_b= %d\n", f, d, l,
r,  *v_config, *v_angle_barre);

          //Compass
          a=((*compas)-10)&0x1ff;
          printf("compas= %d | compass_config = %d \n", a, *compas_config);

          //test bp en mode manuel seul
          b=*code_fonction;
          printf("code_fonction= %d | bp_config = %d\n", b, *BP_config);
          printf("Compass_ref= %d | angle_barre_ref = %d | Angle_corr = %d | new_auto = %d\n", ref,
angle_barre_ref, angle_corr, new_auto);
          switch(b)
          {
          case 0: *v_config=1; new_auto = 1;break;
          case 1: *v_config=7; new_auto = 1;break;
          case 2: *v_config=3; new_auto = 1;break;
        case 3:
                    if (new_auto == 1){
                    ref = a;
                    angle_barre_ref = *v_angle_barre;
                    new_auto = 0;
                    if ((ref > 315 && ref < 360) ||(ref < 45&&ref > 0)){
                              ref = (ref + 180) % 360;
                              boucle_0_360 = 1;
                    }
                    else
                    boucle_0_360 = 0;
                    angle_corr = angle_barre_ref - ((a-ref)*8);
                    }
                    angle_corr = angle_barre_ref - ((a-ref)*8);

                    if (boucle_0_360 == 1)
                    a = (a + 180) % 360;

                     if (a > ref){
                              if(*v_angle_barre > angle_corr)
                              *v_config = 3;
                              else
                              *v_config = 1;
                              }
                              if (a < ref){
                               if(*v_angle_barre > angle_corr)
                                        *v_config = 1;
                              else
                                       *v_config = 7;
                              }
                              ;break;
                     default:*v_config=1;
                    }

          usleep(200000);
    }
  return 0;
}
```

Conclusion

Pour conclure, ce projet fut très enrichissant pour chacun de nous. Il nous as permis de concevoir, valider par la simulation et l'implémentation à partir d'outils appropriés, tout ou une partie des fonctions identifiées lors de la phase d'analyse. Nous avons pu réaliser la synthèse et la réalisation de fonctions logiques en langage VHDL, la vérification du fonctionnement en simulation et sur maquette (carte DE2 d'Altera), l'interfaçage des composants développées avec le bus du processeur (bus Avalon et processeur NIOS 32 bits) et pour finir le test et la validation du système complet.

Ce projet nous as permis d'acquérir des compétences techniques, de mettre en œuvres les différentes compétences acquises lors de nos cours magistraux et de nous confronter aux difficultés techniques entraînées par la réalisation d'un projet. De pouvoir concevoir, tester et observer un produit développé et réalisé de ses propres mains est quelque chose d'important et qui accorde une grande fierté et satisfaction.