



UNIVERSITÉ
TOULOUSE III
PAUL SABATIER



Université
de Toulouse

Bureau d'étude : Pilote de barre franche

Réalisé par : JIDAN Bilal & BERKI Amira.

Encadré par : M. PERISSE Thierry.

Formation : M2 SME

Année universitaire : 2023/2024

SOMMAIRE

Introduction.....	3
Présentation du projet	3
Matériels utilisés	4
Carte DE0-Nano	4
Carte DE2	5
SOPC	6
définition des composant d'un SOPC	6
Test SOPC	7
Bus Avalon	8
Test de l'intégration de la fonction PWM dans notre SOPC	8
Intégration de la fonction gestion anémomètre dans notre SOPC	11
Intégration de la fonction gestion vérin dans notre SOPC.....	14
Fonctionnement de l'anémomètre.....	15
Présentation de l'anémomètre.....	16
L'analyse fonctionnelle	17
Fonctionnement de vérin.....	18
Présentation de vérin	18
Analyse Fonctionnelle	19
Fonctionnement du convertisseur	21
Conclusion.....	23

1. Introduction :

Ce bureau d'études vise à développer un pilote de barre franche sous la forme d'un système sur puce programmable (SOPC), en utilisant le langage de description matériel VHDL (Very High Speed Hardware Description Language). Cette conception se fonde sur l'analyse des spécifications et la décomposition fonctionnelle du système sélectionné, suivie de la création de circuits d'interfaces numériques en VHDL. Le processus inclut la simulation et la validation sur une maquette, ainsi que l'implémentation d'interfaces avec les bus microprocesseurs tels que NIOS, Altera, Avalon pour valider le SOPC par manipulation.

2. Présentation du projet :

Un pilote automatique pour voilier constitue un dispositif électrique ou hydraulique conçu pour maintenir le cap d'un voilier en remplacement d'un équipier. Il s'avère particulièrement utile pour les navigateurs en solitaire ou en équipage restreint.

Nous sommes chargés de développer ce pilote automatique en question. Ce système combine une électronique hautement sophistiquée gérée par un logiciel avancé et une mécanique puissante, offrant des performances élevées en termes de précision de barre dans diverses conditions de navigation.

Les composants clés de ce système comprennent :

- a) Anémomètre : un instrument qui calcule la direction et la vitesse du vent.
- b) Compas : une boussole à compensation d'inclinaison qui acquiert des données sur l'angle d'inclinaison.
- c) Boutons et Leds : une interface de communication entre l'opérateur et le voilier, permettant la surveillance et la configuration en temps réel des différents paramètres.
- d) Vérin : responsable du contrôle de la barre en fonction de la direction souhaitée.

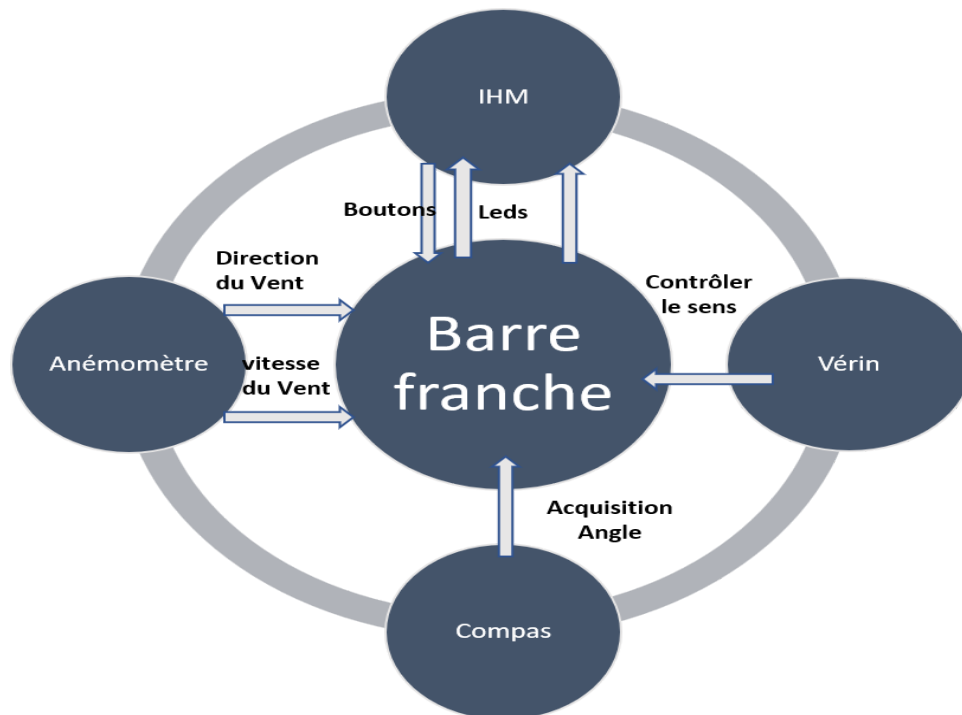


Figure 1 : Schéma illustrant un pilote de barre franche

Notre système principal est divisé en sous systèmes (plusieurs blocs), représenté ci-dessous :

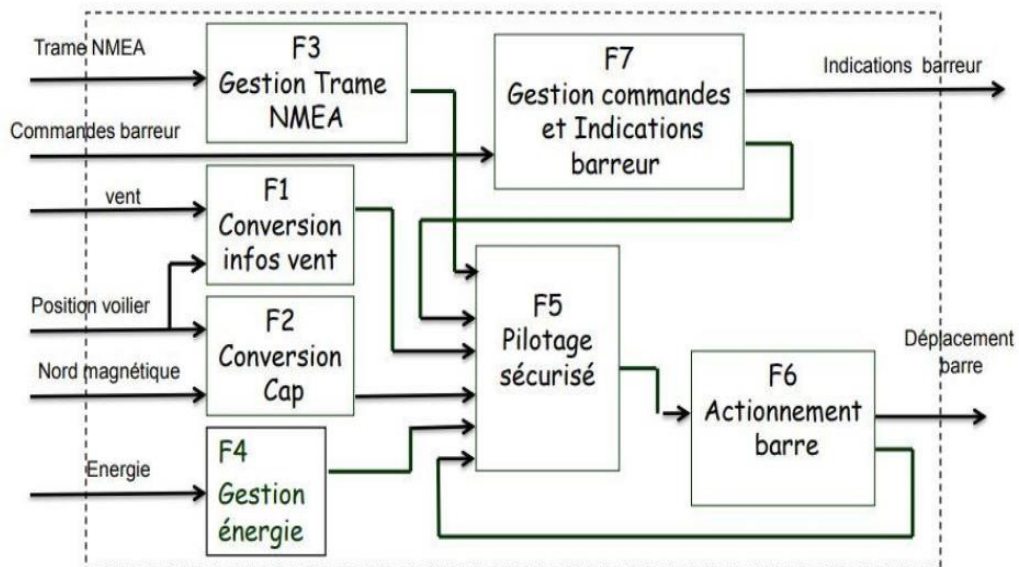


Figure 2 : Schéma illustrant les différents blocs de notre système

3. Matériels utilisés :

3.1 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 la 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 externes (GPIO). L'utilisateur peut gérer un plus grand stockage de données et des tampons grâce à la mémoire internet SDRAM et EEPROM, fournissant 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

3.2 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 utilisés pour créer un son de qualité CD applications et aspect professionnel vidéo. Pour les grands projets de conception, le DE2 fournit 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.

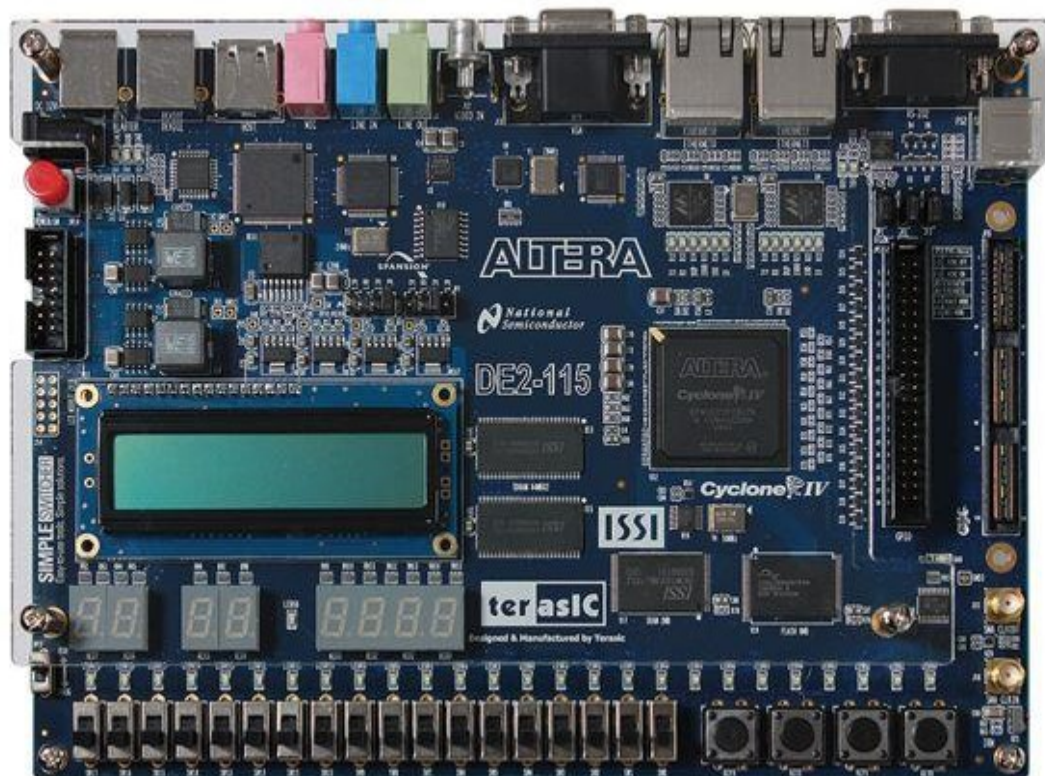


Figure 4 : Carte Altera DE2

4. SOPC

4.1 définition des composants d'un SOPC

CPU : Dans un SOPC (System on a Programmable Chip), le CPU (Central Processing Unit) est le composant central chargé d'exécuter les instructions des logiciels. Il coordonne les opérations du système en traitant les données et en exécutant les programmes, jouant ainsi un rôle crucial dans le fonctionnement global du système embarqué.

MÉMOIRE RAM (ON-chip RAM) : sert à stocker temporairement les données et les instructions nécessaires au fonctionnement du système. Elle offre un accès rapide

au processeur, améliorant les performances en permettant un accès plus rapide aux données par rapport aux mémoires externes, et contribue à la rapidité d'exécution des programmes.

JTAG UART : Joint Test Action Group Universal Asynchronous Receiver/Transmitter est un composant utilisé pour la communication série dans un système électronique, souvent dans le contexte de la programmation, du débogage, et de la configuration des FPGA (Field-Programmable Gate Arrays) ou d'autres dispositifs programmables. Son rôle principal est de fournir une interface série via le protocole JTAG, permettant la transmission de données entre un périphérique (comme un FPGA) et un ordinateur ou un outil de développement. Cela facilite le processus de programmation et de débogage lors du développement de systèmes électroniques.

Sys ID : il attribue un numéro ID unique au système conçu, son objectif principal est souvent de permettre une identification claire et sans ambiguïté, que ce soit pour des raisons de suivi, de gestion, ou d'autres besoins spécifiques liés à la conception et à l'utilisation du système.

Boutons : ce sont des entrées utilisateurs qui peuvent servir d'interfaces utilisateur pour fournir des entrées interactives

LEDS : les leds sont des indicateurs d'état, peuvent être utilisées comme indicateurs visuels pour représenter l'état du système

4.2 Test SOPC

Après avoir implémenté les composants du SOPC via la plateforme designer (voir figure 5) on a généré le code VHDL ainsi que le bloc associé à ce dernier que vous trouverez sur la figure 6

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags
✓		clk_0	Clock Source						
		clk_in	Clock Input	clk	exported				
		clk_in_reset	Reset Input	Double-click to export					
		clk	Clock Output	Double-click to export	clk_0				
		clk_reset	Reset Output	Double-click to export					
✓		RAM	On-Chip Memory (RAM or ROM)...						
		clk1	Clock Input	Double-click to export	clk_0				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	# 0x0000_8000	0x0000_celf		
		reset1	Reset Input	Double-click to export	[clk1]				
✓		BOUTONS	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0001_1030	0x0001_103f		
		external_connection	Conduit	boutons_external_c...					
✓		LEDS	PIO (Parallel I/O) Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0001_1020	0x0001_102f		
		external_connection	Conduit	leds_external_conne...					
✓		JTAG_UART	JTAG UART Intel FPGA IP						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0001_1050	0x0001_1057		
		irq	Interrupt Sender	Double-click to export	[clk]				
✓		SYST_ID	System ID Peripheral Intel FPGA...						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0001_1048	0x0001_104f		
✓		CPU	Nios II Processor						
		clk	Clock Input	Double-click to export	clk_0				
		reset	Reset Input	Double-click to export	[clk]				
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]				
		irq	Interrupt Receiver	Double-click to export	[clk]				
		debug_reset_requ...	Reset Output	Double-click to export	[clk]				
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	# 0x0001_0800	0x0001_0fff		
		custom_instructio...	Custom Instruction Master	Double-click to export					

Figure 5 : conception du sopc depuis plateforme designer

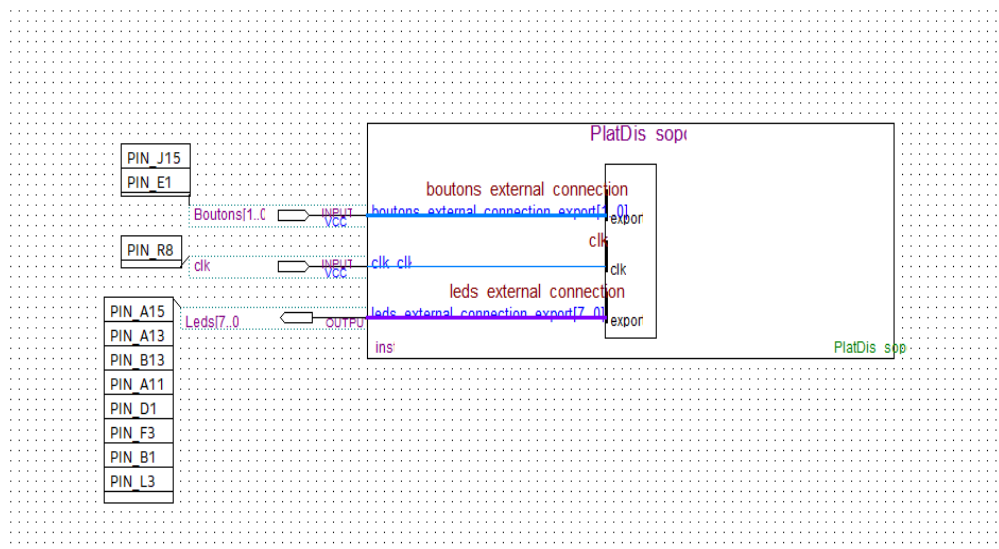


Figure 6 : Schéma bloc SOPC

Ensuite nous avons programmé ce schéma sur notre carte nano. depuis Eclipse nous avons testé le code Hello world ainsi que boutons leds afin de vérifier le bon fonctionnement de notre SOPC.

4.3 Bus Avalon

le bus avalon nous permet de faciliter la communication entre notre SOPC et notre fonction sachant que notre SOPC est relié directement à l'interface avalon qui est relié à son tour aux registres freq, duty et control qui nous permettent soit de lire ou écrire les données de notre fonction

4.4 Test de l'intégration de la fonction PWM dans notre SOPC

Afin de pouvoir associer une nouvelle fonction à notre SOPC, nous devons implémenter son code VHDL sur quartus, on génère le composant associé à ce code qu'on ajoutera par la suite a notre SOPC via plateforme designer

NB : le code VHDL doit comporter des signaux de control clk, chipselect, write_n, reset_n et writedata et readdata, adress afin d'être conforme au protocole Avalon.

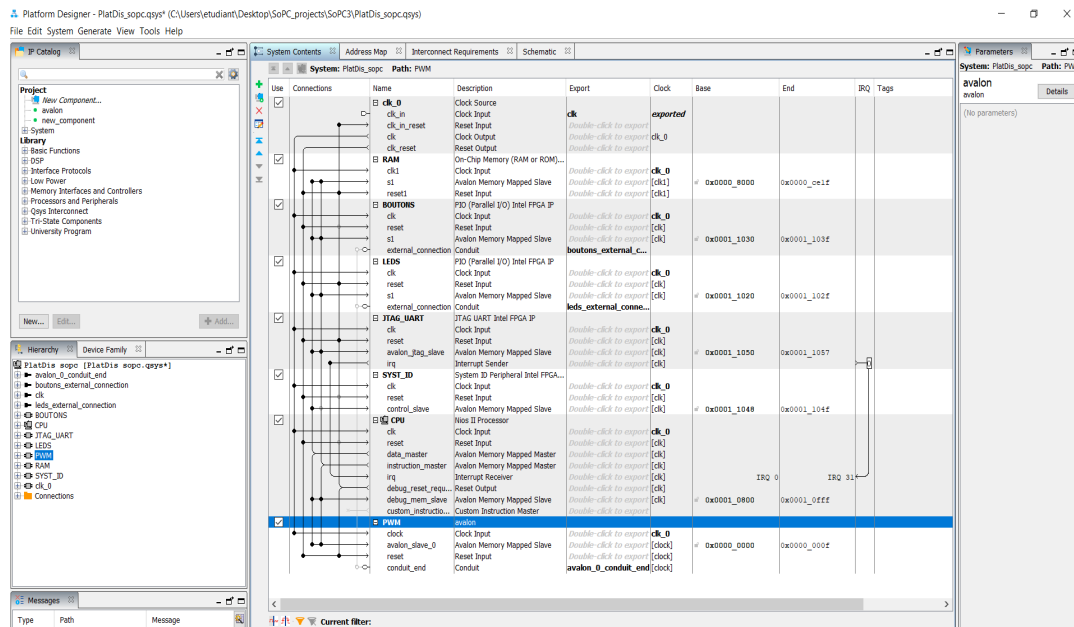


Figure 7: ajoute de la fonction PWM au SOPC via plateforme designer

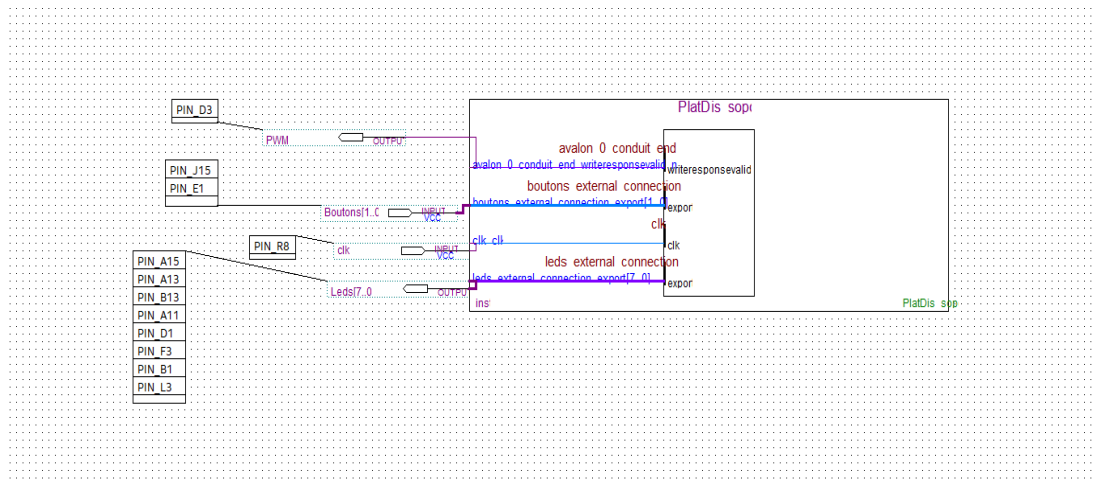


Figure 8 : Schéma bloc de PWM intégré dans SOPC

Sur eclipse nous avons créé un code en C que vous trouverez ci-dessous

```

1 #include "sys/alt_stdio.h"
2 #include <stdio.h>
3 #include "system.h"
4 #include "altera_avalon_pio_regs.h"//pour viter de renseigner les adresses physiques des peripheriques
5 #include "unistd.h" // pour la fonction d lai
6
7 #define freq (unsigned int *) AVALON_0_BASE
8 #define duty (unsigned int *) (AVALON_0_BASE + 4)
9 #define control (unsigned int *) (AVALON_0_BASE + 8)
10 #define boutons (volatile char *) BOUTONS_BASE
11 #define leds (unsigned int*) LEDS_BASE
12 unsigned int a;
13 int main()
14 {
15     *freq = 0x0800; // Frequence de 2048 Hz
16     *duty = 0x0400; // Duty de 1024 alpha de 50 %
17     *control = 0x0003;
18     alt_putstr("Salut ext!\n"); // test si communication OK
19     while (1){
20         alt_putstr("Salut int!\n"); // test si communication OK
21         a = *boutons & 3;
22         printf("boutons = %d \n", a);
23         usleep(1000000);
24         switch(a){
25             case 0 : *leds=0; break;
26             case 1 : *leds=0; break;
27             case 2 : break;
28             case 3 : *leds=*leds + 1; break;
29             default : *leds = 0; break;
30         }
31     }
32     return 0;
33 }

```

Figure 9 : code en c sur NIOS

Après avoir exécuté le code, nous visualisons le résultat sur l'oscilloscope :

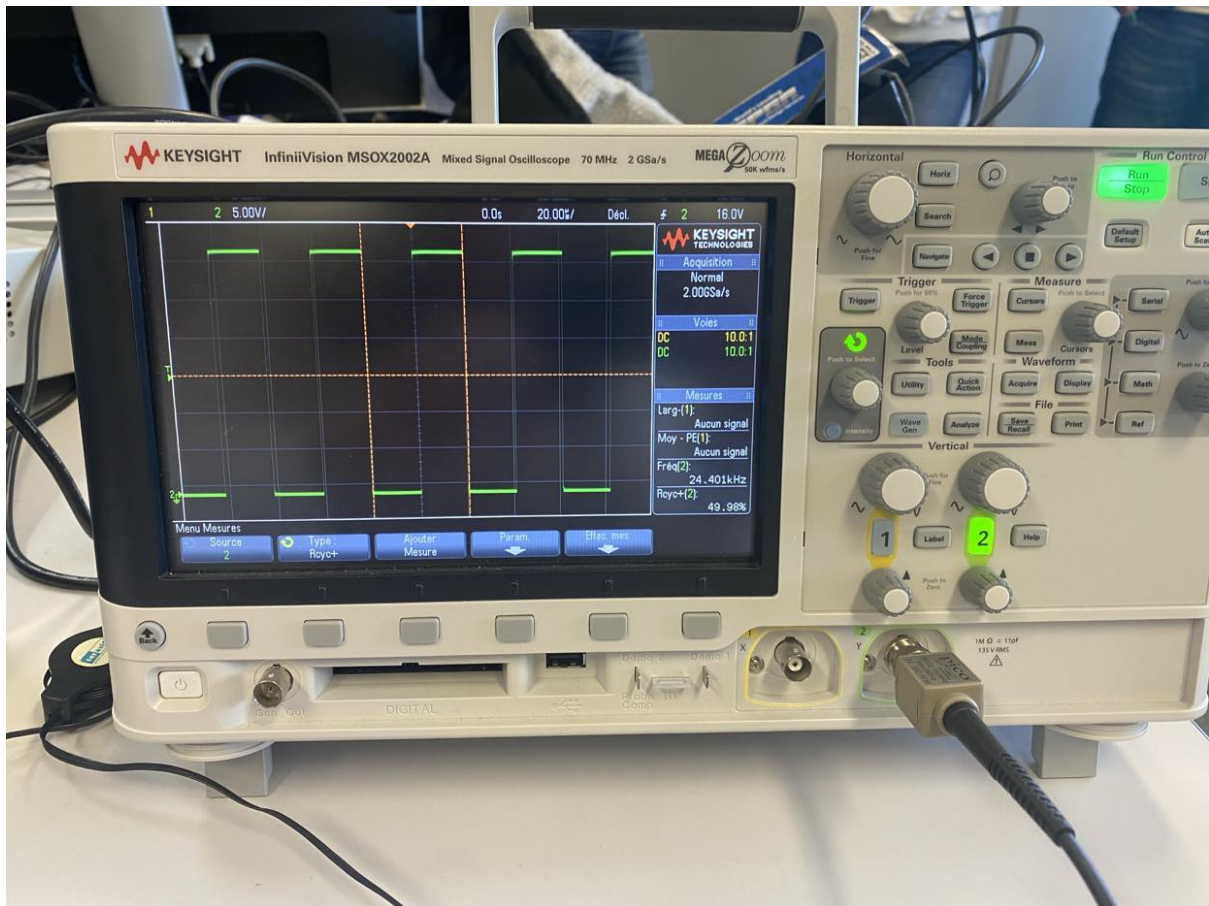


Figure 10 : signal de sortie PWM

D'après le signal affiché sur l'oscilloscope nous obtenons une fréquence de sortie de 24KHz qui est équivalent à notre clock=50Mhz divisé par notre fréquence d'entrée (registre d'écriture) $f=2048\text{hz}$

4.5 Intégration de la fonction gestion anémomètre dans notre SOPC

l'étape de l'intégration de la fonction PWM nous as servi comme test pour apprendre la manière et qu'un SOPC peut réceptionner de nouveaux composants
Nous avons suivi les mêmes étapes que précédemment.

System: PlatDis_soc Path: clk_0							
Use	Connections	Name	Description	Export	Clock	Base	End
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk	exported		
		clk_in	Clock Input	Double-click to export	clk_0		
		clk_in_reset	Reset Input	Double-click to export			
		clk	Clock Output	Double-click to export			
		clk_reset	Reset Output	Double-click to export			
<input checked="" type="checkbox"/>		RAM	On-Chip Memory (RAM or ROM)...				
		clk1	Clock Input	Double-click to export	clk_0		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]	0x0000_8000	0x0000
		reset1	Reset Input	Double-click to export	[clk1]		
<input checked="" type="checkbox"/>		BOUTONS	PIO (Parallel I/O) Intel FPGA IP				
		clk	Clock Input	Double-click to export	clk_0		
		reset	Reset Input	Double-click to export	[clk]		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1030	0x0000
		external_connection	Conduit	boutons_external_c...			
<input checked="" type="checkbox"/>		LEDES	PIO (Parallel I/O) Intel FPGA IP				
		clk	Clock Input	Double-click to export	clk_0		
		reset	Reset Input	Double-click to export	[clk]		
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1020	0x0000
		external_connection	Conduit	leds_external_conne...			
<input checked="" type="checkbox"/>		JTAG_UART	JTAG UART Intel FPGA IP				
		clk	Clock Input	Double-click to export	clk_0		
		reset	Reset Input	Double-click to export	[clk]		
		avalon_jtag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1050	0x0000
		irq	Interrupt Sender	Double-click to export	[clk]		
<input checked="" type="checkbox"/>		SYST_ID	System ID Peripheral Intel FPGA...				
		clk	Clock Input	Double-click to export	clk_0		
		reset	Reset Input	Double-click to export	[clk]		
		control_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]	0x0001_1048	0x0000
<input checked="" type="checkbox"/>		CPU	Nios II Processor				
		clk	Clock Input	Double-click to export	clk_0		
		reset	Reset Input	Double-click to export	[clk]		
		data_master	Avalon Memory Mapped Master	Double-click to export	[clk]		
		instruction_master	Avalon Memory Mapped Master	Double-click to export	[clk]		
		irq	Interrupt Receiver	Double-click to export	[clk]		IRQ 0
		debug_reset_requ...	Reset Output	Double-click to export	[clk]		
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]		
		custom_instructio...	Custom Instruction Master	Double-click to export	[clk]	0x0001_0800	0x0000
<input checked="" type="checkbox"/>		anemometre_0	anemometre				
		reset	Reset Input	Double-click to export			
		avalon_anemometre	Avalon Memory Mapped Slave	Double-click to export	[clock_1]	0x0000_0000	0x0000
		clock_in_freq_ane...	Clock Input	anemometre_0_cloc...	exported		
		clock_1	Clock Input	Double-click to export	clk_0		

Figure 11 : ajout de la fonction gestion anémomètre au SOPC

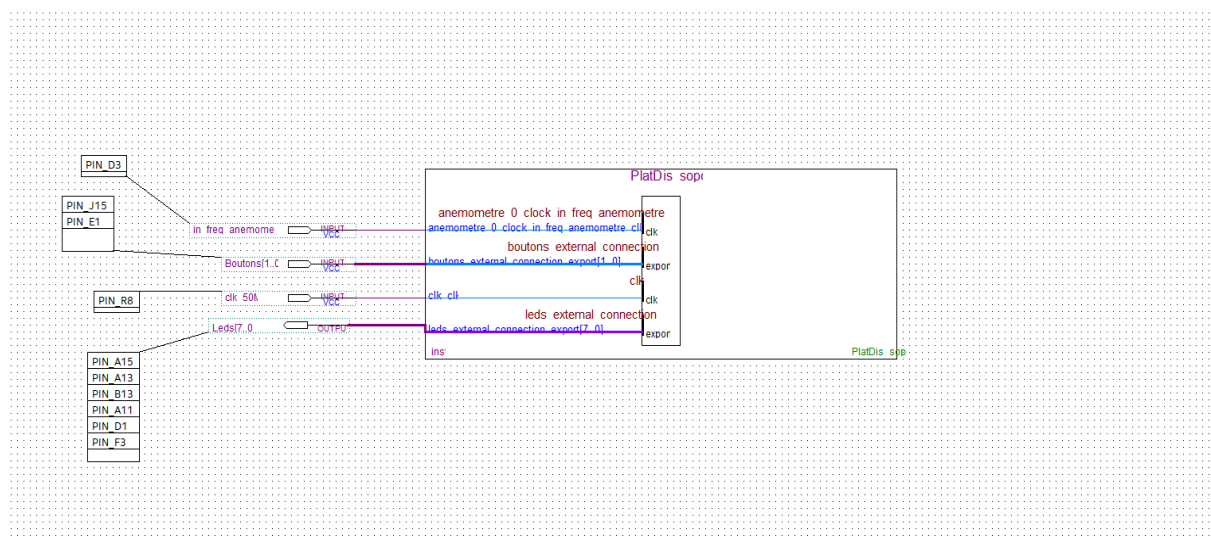


Figure 12 : Schéma bloc de la fct gestion anémomètre intégré dans SOPC

```

1 #include "sys/alt_stdio.h"
2 #include "alt_types.h"
3 #include "system.h"
4 #include "unistd.h"
5 #include "stdio.h"
6 #include "time.h"
7 #include "io.h"
8 // Définition de la macro pour accéder au registre de fréquence
9 #define reg_inFreq (unsigned int *) (ANEMOMETRE_0_BASE + 4)
10 int main()
11 {alt_putstr("Hello from Nios II!\n");
12 // Variable pour stocker la fréquence (initialisée à 25)
13 unsigned int data = 25;
14 unsigned int InFreq = 25;
15 /* Boucle principale du programme (ne se termine jamais) */
16 // Configure le composant anémomètre pour continuer à acquérir la fréquence (continue = 00010)
17 IOWR_32DIRECT(ANEMOMETRE_0_BASE, 0, 0x4);
18 while (1) {
19     printf("----- \n");
20     printf("Freq d'entrée : \n");
21     // Lit la valeur à partir du même registre
22     InFreq = IORD_32DIRECT(ANEMOMETRE_0_BASE + 4, 0);
23 // Masque les bits indésirables et affiche la fréquence sous forme décimale
24     InFreq = 0xFF & InFreq;
25     printf("InFreq = %u Hz\n", InFreq);
26     printf("----- \n");
27 // Lit la fréquence à partir du registre défini par la macro
28     data = IORD_32DIRECT(ANEMOMETRE_0_BASE + 4, 0);
29     printf("vitesse de sortie : \n");
30 // Masque les bits indésirables et affiche la fréquence sous forme décimale
31     data = 0xFF & data;
32     printf("data Vitesse mesurée = %u Km/h\n", data);
33     usleep(500000); // Attend 500 000 microsecondes (0.5 seconde)
34 }
35 return 0;
36 }

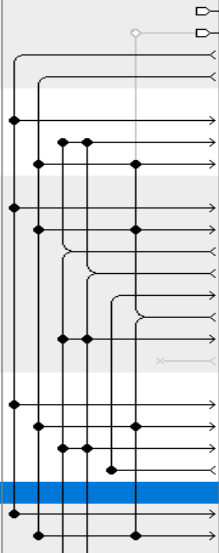
```

Figure 13 : Implémentation du code sur NIOS

```
-----  
Freq d'entrée :  
InFreq = 250 Hz  
-----  
vitesse de sortie :  
data Vitesse mesurée = 250 Km/h  
-----  
Freq d'entrée :  
InFreq = 250 Hz  
-----  
vitesse de sortie :  
data Vitesse mesurée = 250 Km/h  
-----  
Freq d'entrée :  
InFreq = 250 Hz  
-----  
vitesse de sortie :  
data Vitesse mesurée = 250 Km/h  
-----  
Freq d'entrée :  
InFreq = 250 Hz  
-----  
vitesse de sortie :  
data Vitesse mesurée = 250 Km/h  
-----  
Freq d'entrée :  
InFreq = 250 Hz  
-----  
vitesse de sortie :  
data Vitesse mesurée = 250 Km/h  
-----
```

NB: ici data Vitesse mesurée = data anemometre

de la même façon que précédemment nous avons intégré la fonction anémomètre dans notre Sopc

Use	Connections	Name	Description	Export	
<input checked="" type="checkbox"/>		clk_0	Clock Source	clk	
		clk_in	Clock Input	reset	
		clk_in_reset	Reset Input	<i>Double-click to export</i>	
		clk	Clock Output	<i>Double-click to export</i>	
		clk_reset	Reset Output		
<input checked="" type="checkbox"/>		onchip_memory2_0	On-Chip Memory (RAM or ROM) Intel ...		<i>Double-click to export</i>
		clk1	Clock Input	<i>Double-click to export</i>	
		s1	Avalon Memory Mapped Slave	<i>Double-click to export</i>	
		reset1	Reset Input	<i>Double-click to export</i>	
<input checked="" type="checkbox"/>		nios2_gen2_0	Nios II Processor		<i>Double-click to export</i>
	clk	Clock Input	<i>Double-click to export</i>		
	reset	Reset Input	<i>Double-click to export</i>		
	data_master	Avalon Memory Mapped Master	<i>Double-click to export</i>		
	instruction_master	Avalon Memory Mapped Master	<i>Double-click to export</i>		
	irq	Interrupt Receiver	<i>Double-click to export</i>		
	debug_reset_request	Reset Output	<i>Double-click to export</i>		
	debug_mem_slave	Avalon Memory Mapped Slave	<i>Double-click to export</i>		
	custom_instruction_m...	Custom Instruction Master	<i>Double-click to export</i>		
<input checked="" type="checkbox"/>	jtag_uart_0	JTAG UART Intel FPGA IP		<i>Double-click to export</i>	
	clk	Clock Input	<i>Double-click to export</i>		
	reset	Reset Input	<i>Double-click to export</i>		
	avalon_jtag_slave	Avalon Memory Mapped Slave	<i>Double-click to export</i>		
	irq	Interrupt Sender	<i>Double-click to export</i>		
<input checked="" type="checkbox"/>	avalon_verin	new_component		<i>Double-click to export</i>	
	clock	Clock Input	<i>Double-click to export</i>		
	reset	Reset Input	<i>Double-click to export</i>		
	avalon_slave_0	Avalon Memory Mapped Slave	<i>Double-click to export</i>		
	conduit_end	Conduit	new_component_0_conduit		

15

```

6  port (
7      clk, reset_n, data_in : in std_logic;
8      out_pwm, cs, out_sens  : out std_logic
9  );
10 end entity TOP;
11
12 architecture rtl_s of TOP is
13     component avalon_verin is
14     port (
15         clk_clk          : in std_logic := 'x'; -- clk
16         new_component_0_conduit_end_data_in : in std_logic := 'x'; -- data_in
17         new_component_0_conduit_end_cs      : out std_logic;      -- cs
18         new_component_0_conduit_end_out_pwm : out std_logic;      -- out_pwm
19         new_component_0_conduit_end_out_sens : out std_logic;      -- out_sens
20         reset_reset_n      : in std_logic := 'x' -- reset_n
21     );
22     component avalon_verin;
23 begin
24     u0 : component avalon_verin
25     port map (
26         clk_clk          => clk,          --
27         new_component_0_conduit_end_data_in => data_in, -- new_component_0_conduit_end.data_in
28         new_component_0_conduit_end_cs      => cs,      -- .cs
29         new_component_0_conduit_end_out_pwm => out_pwm, -- .out_pwm
30         new_component_0_conduit_end_out_sens => out_sens, -- .out_sens
31         reset_reset_n      => reset_n      --
32     );
33
34 end architecture;
35

```

Figure 16 : Code VHDL du vérin

5. Fonctionnement de l'anémomètre

5.1 Présentation de l'anémomètre

Le mot anémomètre vient du mot grec "anemos" signifiant vent et du suffixe "mètre" signifiant mesure. Un anémomètre est donc un appareil qui existe dans le domaine météorologique et qui sert à mesurer la vitesse du vent.

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 17 : anémomètre

5.2 L'analyse fonctionnelle

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

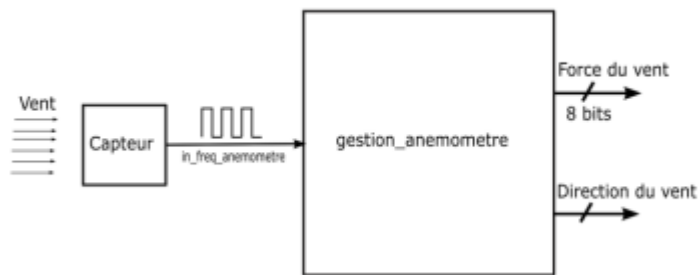


Figure 18: Schéma fonctionnel simplifié de l'anémomètre

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

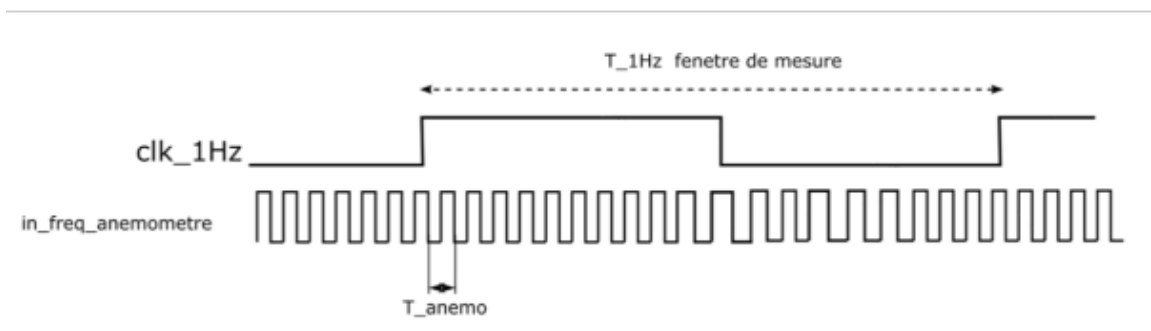


Figure 19: Principe de mesure `in_freq_anemometre`

Description de l'interface

Clk_50M : Horloge de 50MHz présente sur la carte DE0 NANO. Elle permet de piloter tous les éléments du système.

In_freq_anemometre : la fréquence entrante à mesurer

Raz_n : actif à l'état haut. Permet de réinitialiser le système.

Continu : Anémomètre réalise des mesures de façon continue lorsque actif.

Start_stop : Permet de lancer une mesure lorsqu'on n'est pas en mode continu.

Data_anemometre : Contient les données de mesures sur 8bits.

Data_valide : Actif lorsqu'une donnée est disponible et mise à zéro quand les données sont lues.

Memo: Ce bloc permet de stocker et mémoriser la valeur de la fréquence venant du compteur et par la suite l'initialiser le registre pour autoriser la réception de la nouvelle donnée sans écraser l'ancienne. Ainsi que ce registre rendre possible de visualiser les données puisque le calcul se fait à l'échelle d'une seconde.

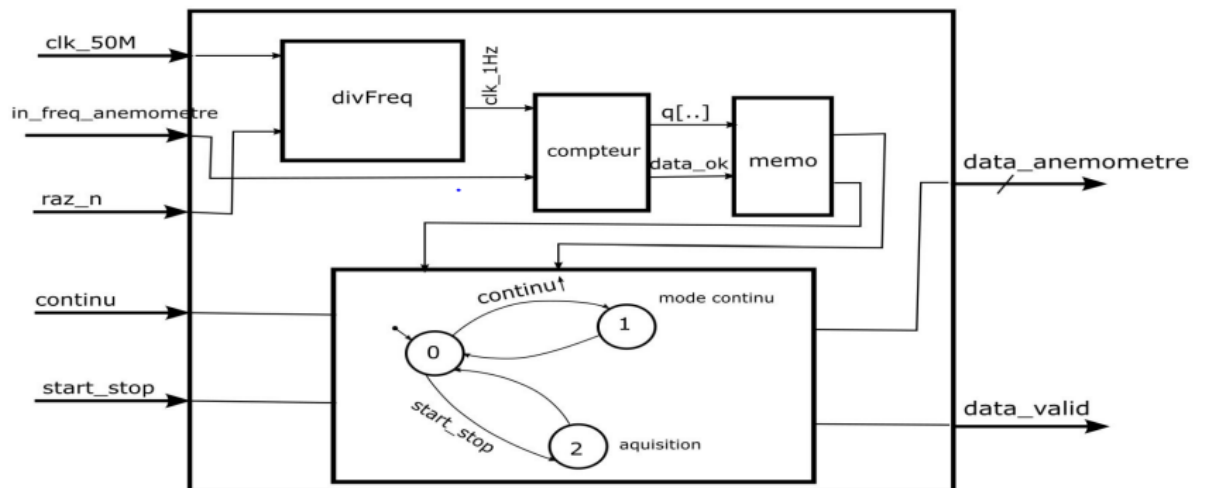


Figure 20: schéma fonctionnel

Ci-dessous, on peut avoir le schéma électrique généré par Quartus à l'aide de port map qui relie toutes les blocs pour avoir ce bloc final.

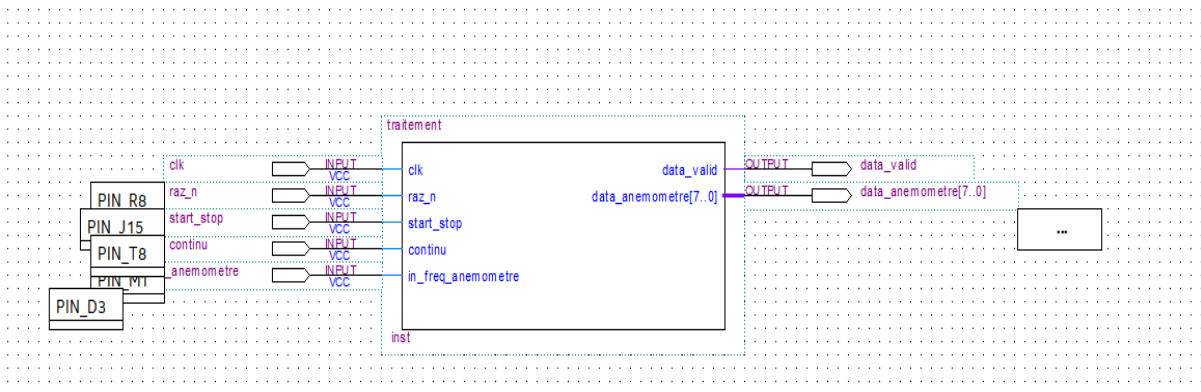


Figure 21: généré par quartus

6. Fonctionnement de vérin

6.1 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.

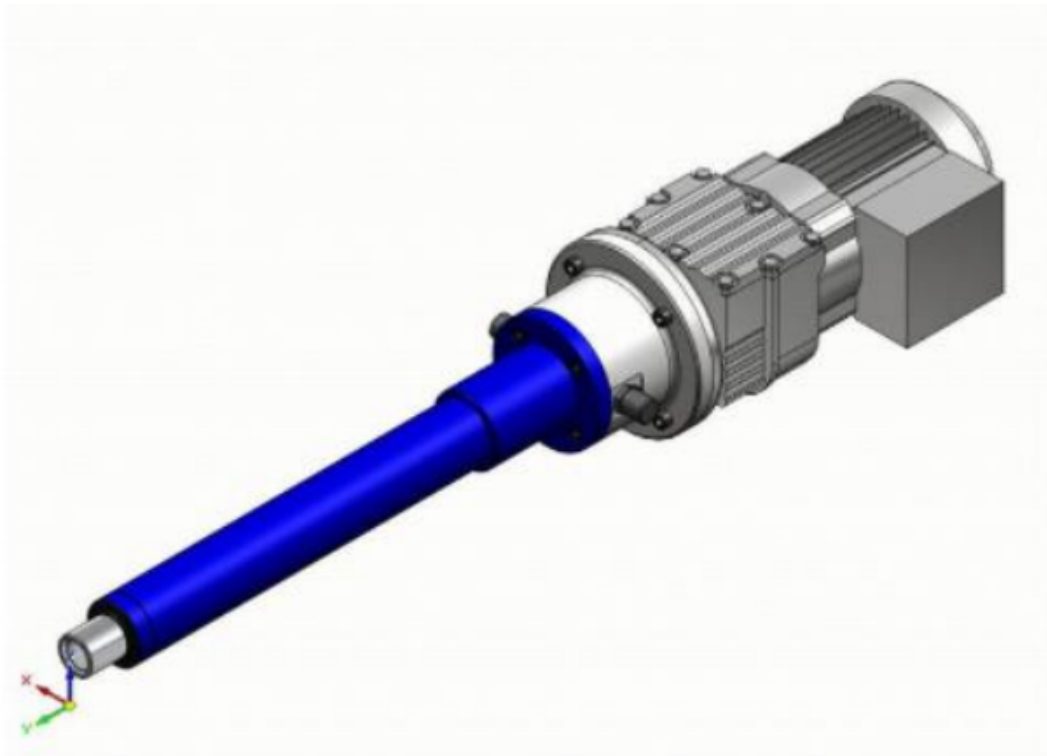


Figure 22 : Vérin

6.2 Analyse Fonctionnelle

Le schéma-bloc ci-dessous définit les différentes qui constituent la fonction “Vérin” ainsi qu'avec l'interface Bus Avalon :

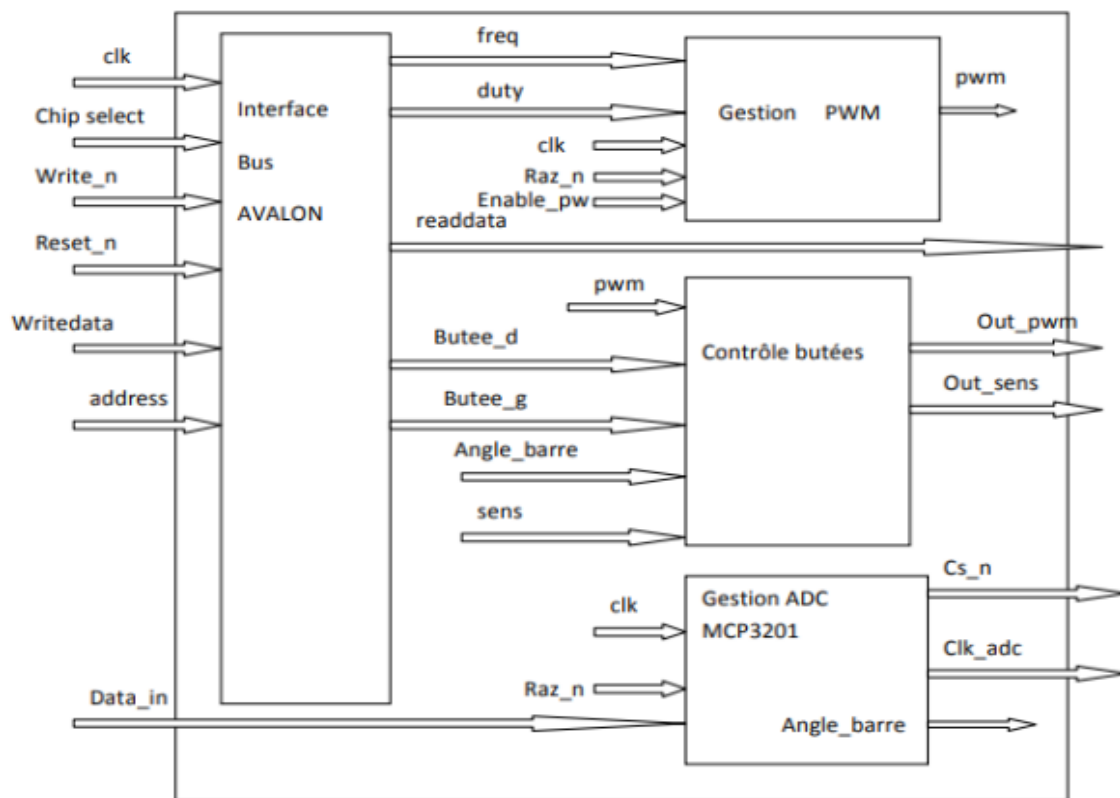


Figure 23 : L'analyse fonctionnelle de vérin.

- ❖ **Fonction Gestion PWM:** Cette fonction a pour but de créer un signal PWM en sortie avec une possibilité de modifier la largeur d'impulsion en changeant la valeur de rapport cyclique et aussi de modifier la période en agissant sur le signal "Freq" plus le rapport cyclique est proche de 0,5 plus le vérin bouge plus vite
- ❖ **Fonction Contrôle Butées :** Cette fonction permet de forcer le signal PWM à 0 si la barre de vérin atteint les valeurs seuil fixé pour le vérin ("Butée_d" - "Butée_g") ça permet aussi de contrôler le sens du déplacement de la barre et de nous indiquer l'arrivée de la tige aux extrémités en les indiquant sur les fins de course droite et gauche.
- ❖ **Fonction de Gestion ADC MCP 3201:** Cette fonction permet de récupérer la donnée de l'angle du vérin à partir du convertisseur AN MCP 3201 chaque 100 ms, puis envoyer cette valeur au bloc de contrôle butées afin de commander le vérin

6.3 Fonctionnement du convertisseur

Pour extraire les données, le convertisseur utilise trois signaux :

- **CS**: un signal qui indique le début et la fin de la conversion. Quand il est à 0 en commence le processus d'acquisition des données et au moment où il passe à 1 ça veut dire que l'acquisition est finie.
- **CLK**: est l'horloge du convertisseur et qui est choisie de telle façon que chaque période soit équivalente à un bit numérique. Sa valeur est de 1 MHz.
- **Dout** : est le signal d'entrée qui contient l'information numérique à propos de l'angle et qui entre dans le processus de traitement pour extraire les données d'angle barre.

Au début le signal CS est en état haut, dès qu'on veut commencer l'acquisition des données, on active le CS en le remettant en état bas.

Comme il est montré dans la figure ci-dessous, en remarque qu'au niveau du signal Dout les bites utiles commencent juste après le bit de sécurité qui est le 4^e bit qui doit être à la zéro logique, aussi en remarque que le début du bit commence avec le front montant et la fin du bit se fait sur le front descendant on sélectionne les 12 bits qui suit les bit de sécurité.

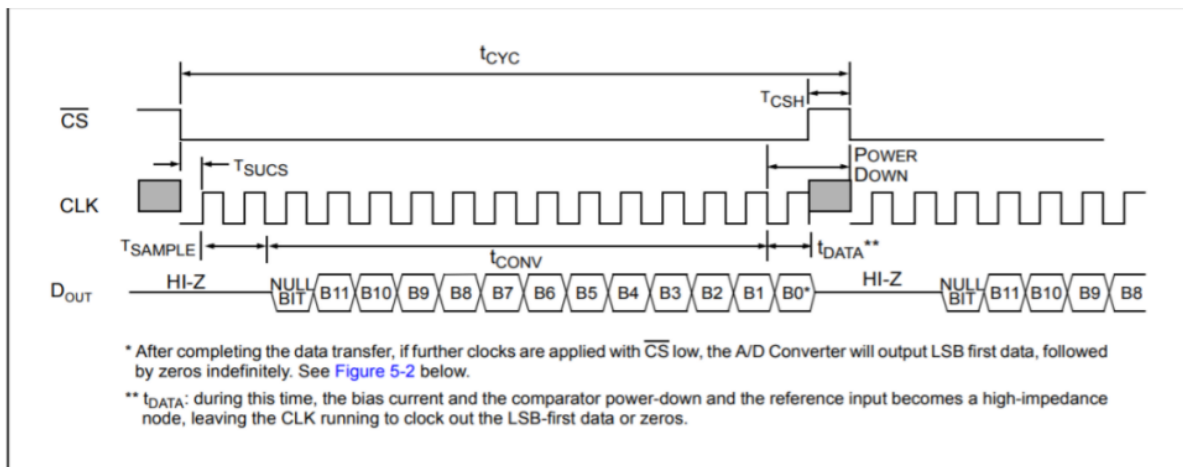


Figure 24 : Extrait de la datasheet du convertisseur

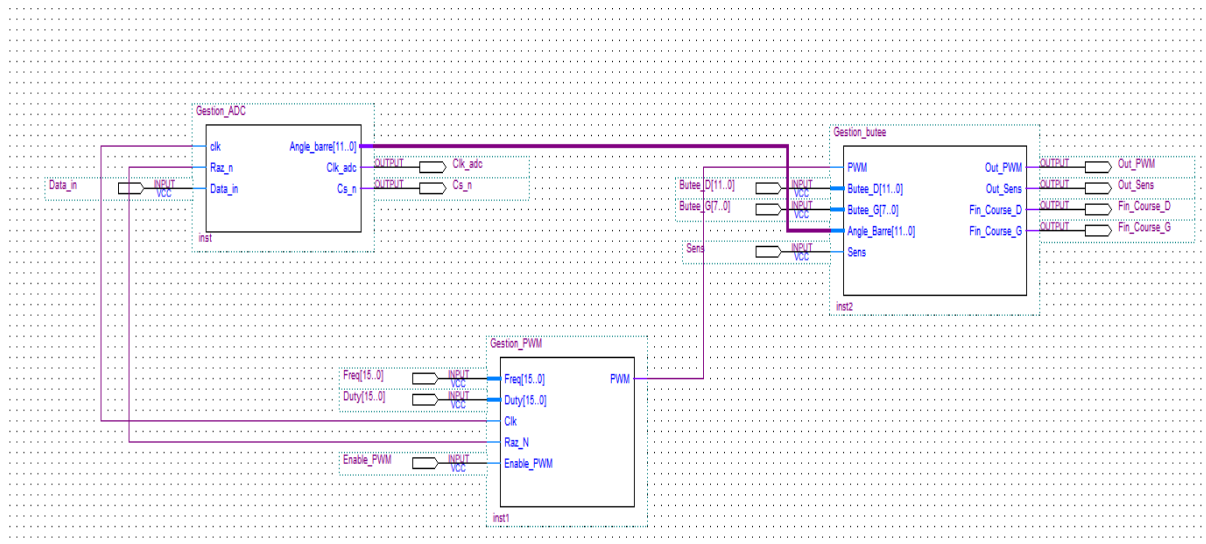


Figure 25 : Schéma fonctionnel de la fonction "gestion du vérin"

- ❖ **Bloc PWM:** son rôle est de générer un signal PWM en sortie pour pouvoir gérer la vitesse du vérin. Pour créer ce signal on doit fixer la fréquence et le rapport cyclique (cela dépendra de la vitesse choisie pour déplacer le vérin). En fixant la fréquence on va pouvoir créer une période, et on fixant le rapport cycle on va pouvoir créer le signal carré PWM(état haut pour un duty cycle inférieur à la valeur fixée
- ❖ **Bloc Contrôle_Butée:** Ce bloc permet de contrôler le mouvement du vérin, pour le réaliser on aura besoin du signal créer dans le bloc précédent "PWM", de la position du vérin "angle barre", des valeurs seuils pour pouvoir comparer à chaque fois la position du vérin avec ces valeurs qui sont "butée_g" et "butée_d" et finalement le sens pour savoir si le vérin est en mouvement vers la 20 gauche ou vers la droite. Si l'angle barre soit supérieur ou égale à la valeur seuil de butée (butée_g si le sens du déplacement du vérin est vers la gauche, idem pour la butée_d) il faut mettre le signal PWM en état bas, puis générer un signal de fin de course "fin_course_d" ou "fin_course_g" (selon le sens de déplacement).
- ❖ **Bloce du convertisseur AN MCP 3201:** Le bloc de gestion du convertisseur AN MCP 3201 coordonne l'acquisition de données analogiques à partir du capteur de position de la barre. Il utilise une machine à état, un compteur d'horloge, un registre à décalage et des générateurs de signaux pour contrôler le convertisseur ADC, synchroniser le processus de conversion, récupérer les données converties, et assurer une mise à jour régulière de la position de la barre dans le système global de contrôle du vérin.

7. Conclusion

Ce bureau d'études nous a offert une expérience enrichissante, notamment en nous permettant d'acquérir des compétences approfondies dans l'utilisation de logiciels tels que Quartus et Eclipse. Nous avons également eu l'occasion de résoudre divers problèmes qui peuvent survenir lors de la manipulation de ces outils. Une partie importante de notre apprentissage a été consacrée à la fonction 1 et à la création d'un SOPC, qui fonctionne comme une puce programmable intégrée à notre carte DE0 nano, constituée de différents composants.

Au cours de ce projet, nous avons mis en œuvre avec succès la fonction de gestion de l'anémomètre en utilisant plusieurs blocs que nous avons intégrés dans notre SOPC. Nous avons pu observer les résultats souhaités et valider ainsi notre approche. De plus, nous avons travaillé sur la fonctionnalité du vérin, bien que nous n'ayons pas complètement finalisé la visualisation des résultats. En résumé, malgré les défis rencontrés lors des travaux pratiques, cette expérience en bureau d'études a été extrêmement bénéfique, car elle nous a permis d'approfondir nos connaissances et de développer des compétences concrètes.