

UNIVERSITÉ DE TOULOUSE 3 - PAUL SABATIER

DÉPARTEMENT ÉLECTRONIQUE ÉNERGIE ÉLECTRIQUE AUTOMATIQUE
Systèmes et Microsystèmes Embarqués

RAPPORT DE BE

SYNTHÈSE ET MISE EN ŒUVRE DES SYSTÈMES

Pilote de barre franche SIMRAD

Réalisé par :
Meriem BOUSLAH
Nouria KACEMI

Encadré par :
Pedro CARVALHO MENDES



Table des matières

1 INTRODUCTION	1
2 L'ANALYSE FONCTIONNELLE	1
2.1 BESOINS EXTERNES	1
2.2 BESOINS INTERNES	1
2.3 MISE EN OEUVRE	2
3 LE VHDL - VHSIC Hardware Description Language	3
4 LOGICIEL QUARTUS	5
5 LE FPGA - FIELD-PROGRAMMABLE GATE ARRAY	5
6 MATERIELS ET LOGICIELS UTILISES DANS CE BE	6
6.1 CARTE DE0	6
7 SOFT CORE NIOS II	8
7.1 REGISTRES GENERAUX DU NIOS II	9
7.2 RESET ET SIGNAUX DE DEBUG	9
8 PILOTE DE BARRE FRANCHE	10
8.1 MODE DE FONCTIONNEMENT	11
8.1.1 MODE DE FONCTIONNEMENT MANUEL	11
8.1.2 MODE DE FONCTIONNEMENT AUTOMATIQUE	12
9 OBJECTIF DU BE - PILOTE BARRE FRANCHE	12
10 Contexte du système	13
11 GENERATION SIGNAL PWM	14
11.1 SIMULATION SUR OSCILLOSCOPE	14
12 ANEMOMETRE	15
12.1 ETAPES DE REALISATION :	15
12.2 MODE MONOCOUP	17
12.3 MODE CONTINU	17
12.4 SIMUMATION ANEMOMETRE	18
13 BUS AVALON DE L'ANEMOMETRE	19
13.1 INTEGRATION DU SOPC :	20
14 COMMUNICATION MAITRE DE0 ET ESCLAVE AN MCP 3201	21
15 VERIN	21
15.1 ARCHITECTURE DE QUELQUES BLOCS	23
15.2 SIMULATION VERIN	26
15.3 SPECIFICATIONS AVALON	26
16 CONCLUSION	27

1 INTRODUCTION

Dans le cadre de la formation Master 2 Systèmes et Microsystèmes Embarqués, de la validation de nos compétences en programmation VHDL ; nous avons mis nos connaissances en pratique au travers d'un projet concret : Pilote de barre franche.

L'objectif de ce bureau est d'acquérir de solides compétences en VHDL, être capable de réaliser des fonctions logiques de différentes façons et d'exécuter des simulations structurelles et comportementales, ainsi que la maîtrise du logiciel QUARTUS.

Durant ce BE, nous avons appris à construire une analyse fonctionnelle et réaliser les synthèses des fonctions en utilisant une description VHDL.

2 L'ANALYSE FONCTIONNELLE

Pour réaliser la conception d'un système, il faut passer par l'étape de la conception et ainsi accomplir l'étude et l'analyse fonctionnelle du système : cette étude doit s'inscrire dans le processus global de développement de conception ; elle permet de rendre le système compréhensible et réalisable.

L'analyse fonctionnelle se subdivise en deux catégories : externe et interne (fig 1)

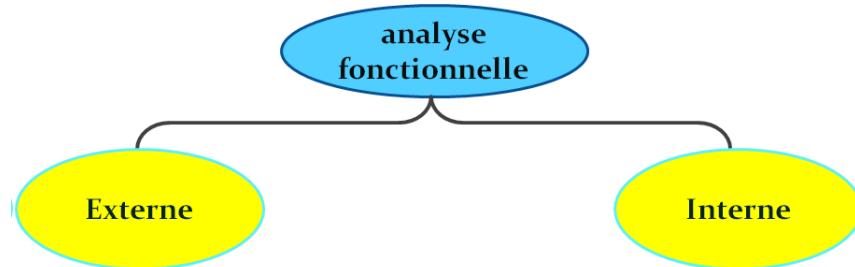


FIGURE 1 – L'Analyse fonctionnelle

2.1 BESOINS EXTERNES

Les besoins externes concernent les fonctions que doit assurer le système d'un point de vue utilisateur c'est à dire l'usage du produit. Ce besoin doit satisfaire les attentes du client et répondre au cahier des charges de l'analyse fonctionnelle externe.

Le produit peut être considéré comme une boîte noire et seules les fonctions qui « sortent » de la boîte vers l'extérieur sont à prendre en considération.

2.2 BESOINS INTERNES

L'analyse interne concerne le produit en lui même, elle consiste à travailler cette fois, l'intérieur de la boîte noir qui explique/ améliore et qui donne des précisions sur les fonctionnalités intérieures du système.

Elle exprime le point de vue du concepteur du système et donc met en évidence les fonctions techniques de ce dernier.

2.3 MISE EN OEUVRE

L'étude de L'analyse fonctionnelle, se résume dans la réalisation des étapes suivantes :

- Définition du système et de ses limites lors d'une analyse au moyen des documents d'exigences correspondants.
- Définition du niveau de détail (profondeur de l'analyse).
- Identification des fonctions du système/modes d'exploitation/scénarios opérationnels.
- Représentation du système par élaboration de la ou des arborescence(s) fonctionnelle(s), matrice(s) fonctionnelle(s), du ou des schéma(s) fonctionnel(s).

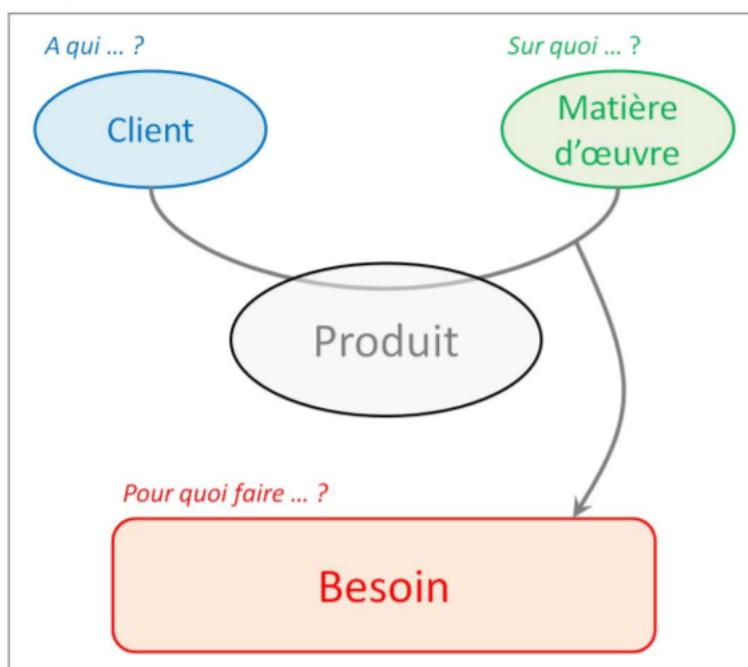


FIGURE 2 – Schémas d'analyse fonctionnelle (conception)

Les réponses à ces trois questions sur la figure 3, aboutissent à un énoncé du besoin, qui doit être rédigé de la façon suivante :

Le produit rend service au client en agissant sur la matière d'oeuvre pour satisfaire le besoin : Cela revient à faire la conception d'un système suivant un cahier des charge particulier sous un environnement précis.

Le processus de conception est représenté dans la figure 3

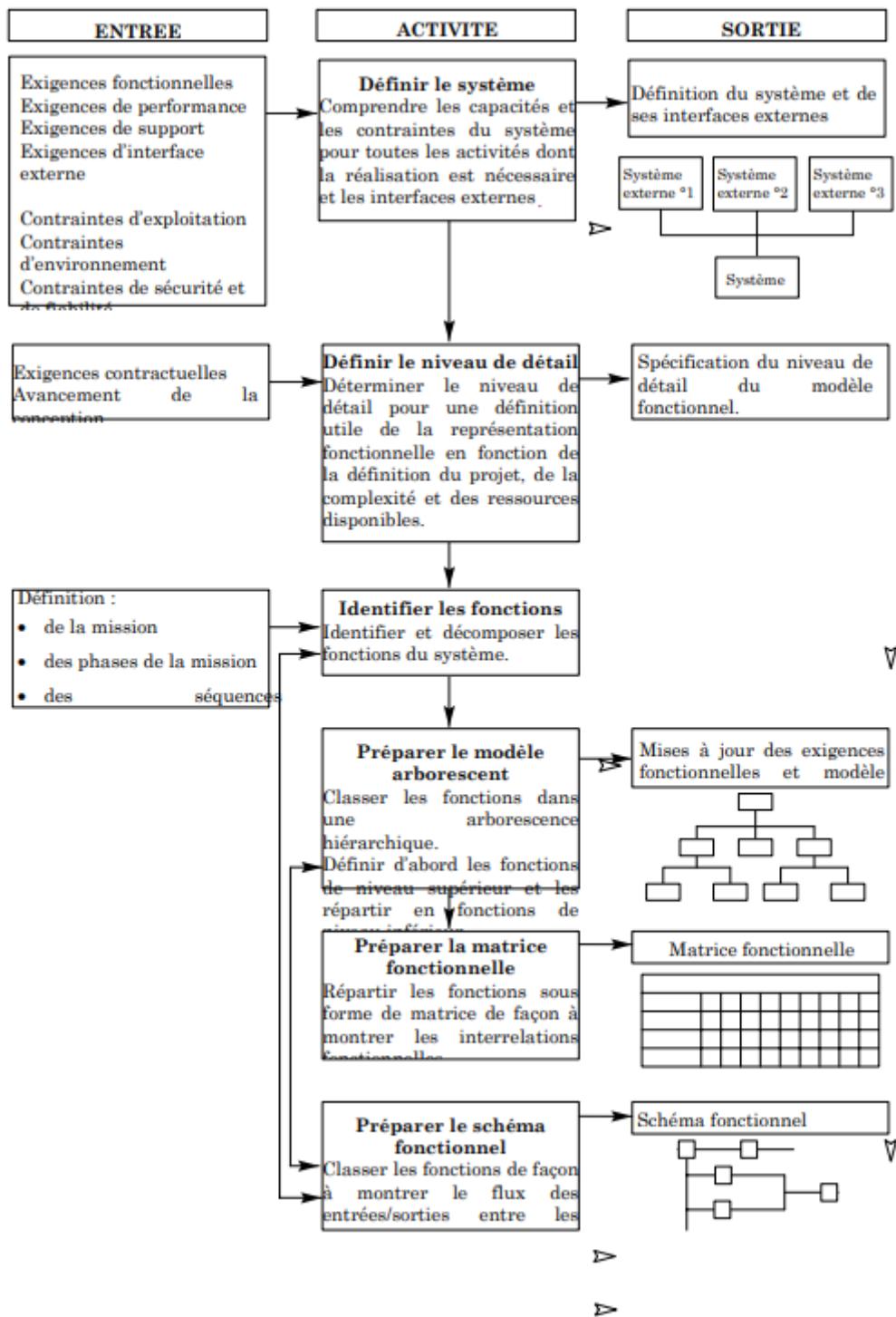


FIGURE 3 – Présentation de la mise en oeuvre de l'analyse fonctionnelle

Chaque itération doit être détaillée dans la conception.

3 LE VHDL - VHSIC Hardware Description Language

Le VHDL (VHSIC Hardware Description Language) est un langage destiné à programmer les circuit numérique et électronique, il est fondé sur la logique combinatoire et séquentielle, il intègre le comportement ainsi que l'architecture d'un système. Le VHDL permet de vérifier les

circuits avant leur conception grâce à la simulation ce qui représente un grand avantage pour les électroniciens (figure 4).

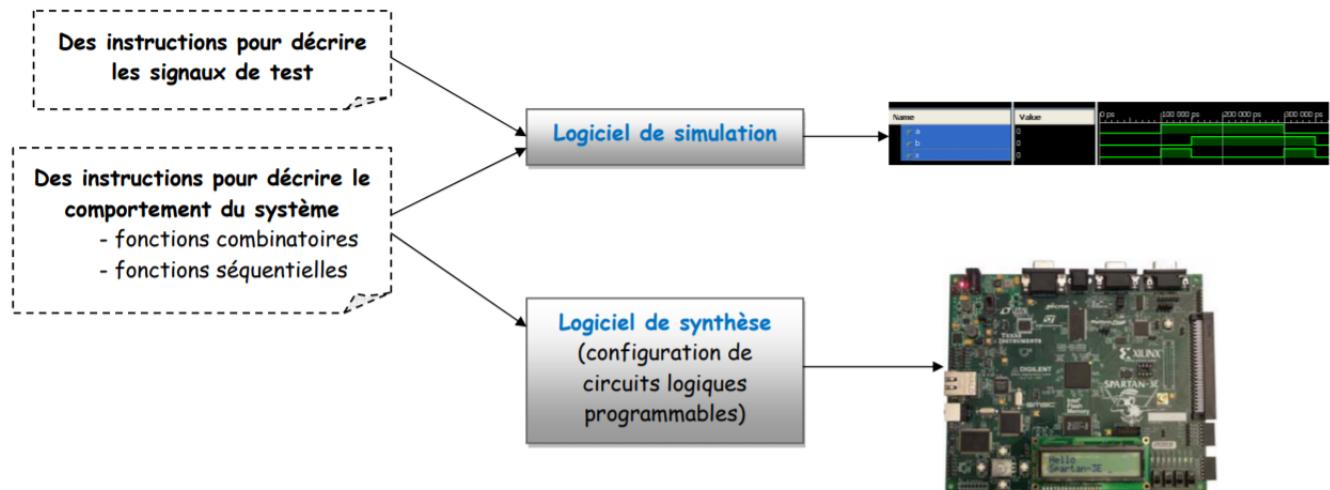


FIGURE 4 – Mise en oeuvre d'un système en VHDL

Nous, nous utilisons le logiciel Quartus 2.

Une description VHDL est composée de 2 parties indivisibles :

- L'entité (ENTITY), elle définit les entrées et sorties.
- L'architecture (ARCHITECTURE), elle contient les instructions VHDL permettant de réaliser le fonctionnement attendu.

Les objectifs de la simulation :

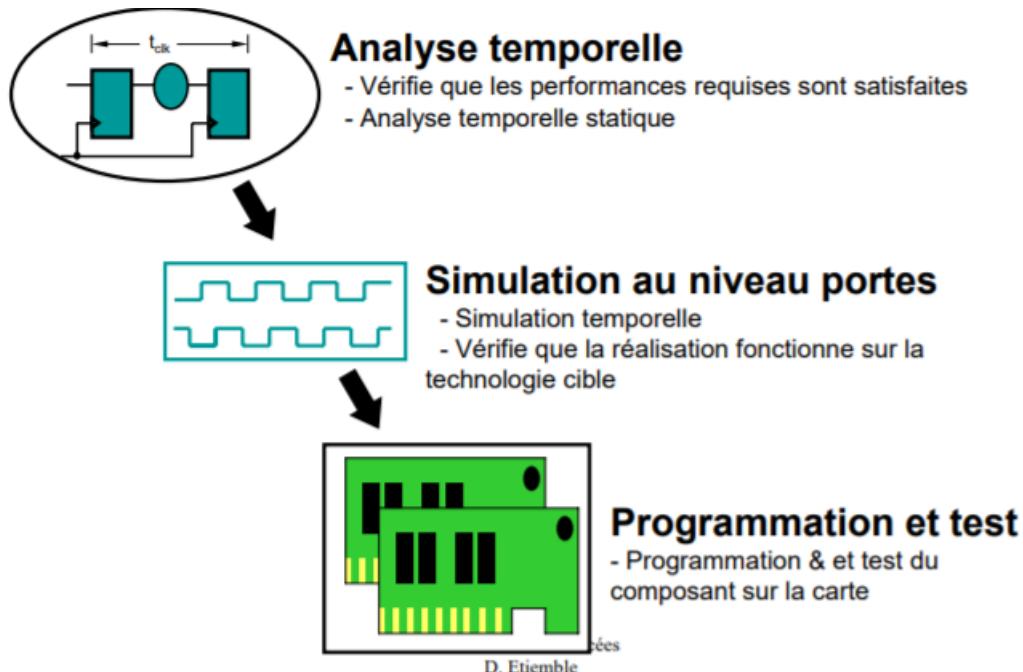


FIGURE 5 – Les objectifs de la simuulation

4 LOGICIEL QUARTUS

Quartus est un logiciel développé par la société Altera, permettant la gestion complète d'un flot de conception CPLD -Complex Programmable Logic Device- ou FPGA -Field Programmable Gate Array-. Ce logiciel permet de faire une saisie graphique ou une description HDL -Hardware Description Language- (VHDL – Very Hardware Description Language- ou AHDL - Altera Hardware Description Language-) d'architecture numérique, d'en réaliser une simulation, une synthèse et une implémentation sur cible reprogrammable.

Il comprend une suite de fonctions de conception au niveau système, permettant d'accéder à la large bibliothèque d'IP d'Altera et un moteur de placement routage intégrant la technologie d'optimisation de la synthèse physique et des solutions de vérification.



FIGURE 6 – Logiciel QUARTUS II

5 LE FPGA - FIELD-PROGRAMMABLE GATE ARRAY

Le FPGA (Field-Programmable Gate Array) est un circuit intégré logique programmable, qui sera par la suite câblé avec d'autres FPGA, dans le but de répondre à des fonctions spécifiques. Par exemple, les circuits FPGA peuvent être utilisés dans la fabrication des microprocesseurs.

Les circuits FPGA réunissent le meilleur des ASIC et des systèmes basés processeur. Les cinq principaux atouts de la technologie FPGA sont :

- Performances
- Temps de mise sur le marché
- Coût
- Fiabilité
- Maintenance à long terme

6 MATERIELS ET LOGICIELS UTILISES DANS CE BE

Nous avons donc programmé des circuits FPGA, pour réaliser un système de barre franche (qui sera expliqué plus tard). Nous avons utilisé les cartes électroniques ALTERA DE2 CYCLONE II et ALTERA DE0 CYCLONE IV.

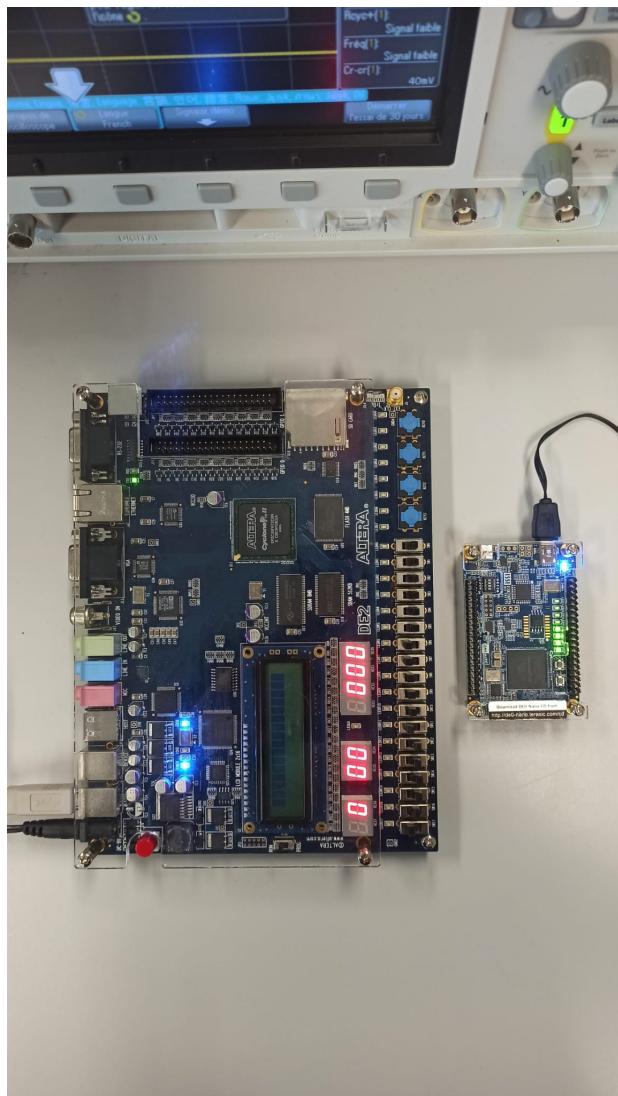


FIGURE 7 – Carte ALTERA DE2 et DE0

Nous avons utilisé les logiciels Quartus 9 et Quartus 18 pour la configuration graphique du FPGA GIO, pour la simulation des signaux et pour la programmation de la carte en VHDL.

6.1 CARTE DE0

La carte DE0-Nano présente une plate-forme de développement FPGA compacte adaptée au prototypage de conceptions de circuits telles que les robots et les projets "portables". Spécifications : Altera Cyclone IV EP4CE22F17C6N.

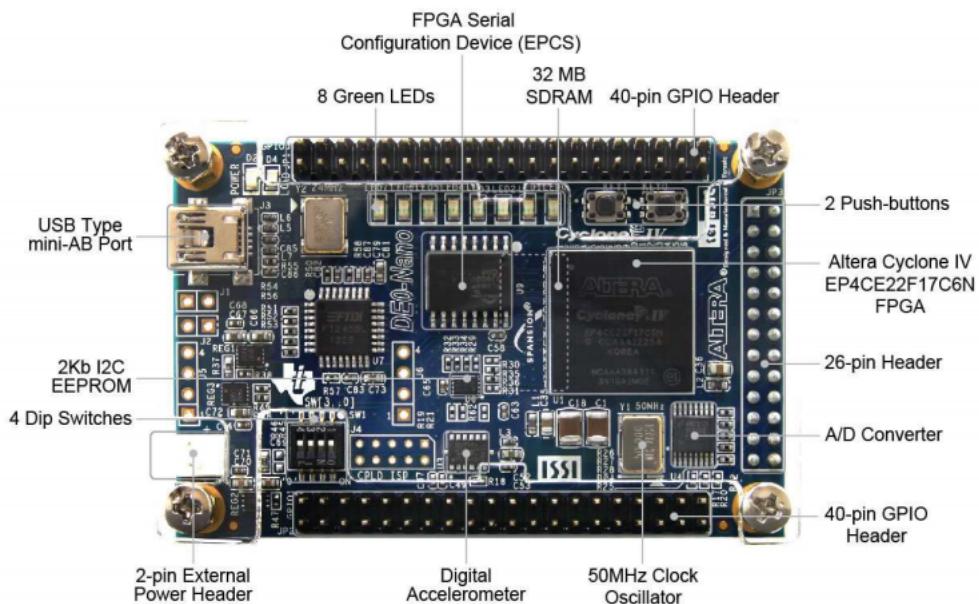


FIGURE 8 – Carte de développement FPGA ALTERA DE0

Caractéristiques techniques :

Altera Cyclone IV EP4CE22F17C6N FPGA, 22 320 éléments logiques, mémoire de 594 Ko, 66 multiplicateurs, 4 PLL, 153 broches E/S maximum

- Mémoire flash de configuration 64 Mo
- Oscillateur d'horloge 50 MHz
- 32 Mo SDRAM
- EEPROM I2C 2 ko
- Accéléromètre ADXL345 3 axes
- ADC128S022, 8 canaux, convertisseur A/N 12 bits
- 8 x LED vertes
- 2 boutons-poussoirs antirebond
- 4 interrupteurs DIP
- Alimentation : connecteur mini-USB (5 V), deux broches d'embase GPIO (5 V), embase d'alimentation externe 2 broches (de +3,6 à +5,7 V)
- Circuit intégré USB-Blaster pour la programmation
- Deux embases 40 broches offrant 72 broches d'E/S numériques
- Une embase 26 broches offrant 16 broches d'E/S numériques et 8 broches d'entrée analogiques

Les GPIO du DE0 :

le DE0 est composé de deux GIOP identiques : GIOP 0 et GPIO 1.

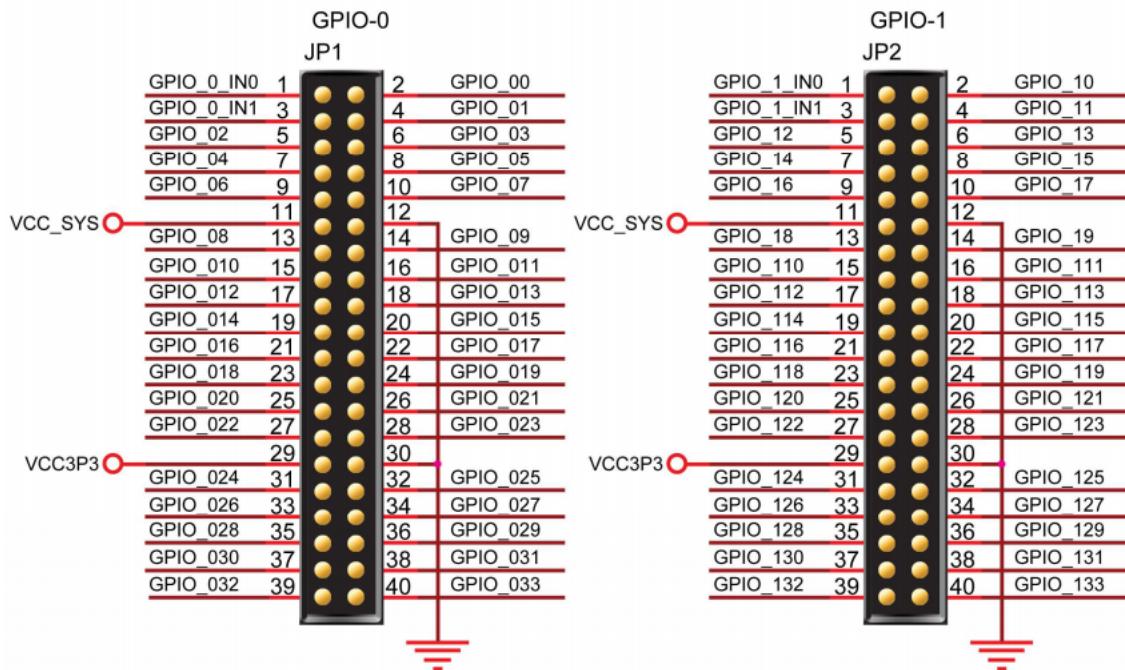


FIGURE 9 – Les GPIO du DE0

Les ports GPIO (General Purpose Input/Output) sont des Entrée sortie à usage général. Elles sont placées sur un circuit électronique afin de communiquer avec des composants électroniques et circuits externes.

7 SOFT CORE NIOS II

Le NIOS est un processeur soft core propriétaire d'Altera. Il est basé sur un cœur 32 bits et doté du bus Avalon. Le Nios-II est configurable et permet d'ajouter des éléments internes en fonction des besoins du processeur et de l'application



FIGURE 10 – Soft core NIOS II

Altera :

- Soft Core Nios, Nios-II
 - Hard Core ARM

Caractéristiques du NIOS II :

- Processor RISC
- Architecture de Harvard :
 - Data Master port
 - Instruction Master port
- Banc de 32 registres de 32 bits
- Chacune de ces entités définit l'architecture du Nios-II, mais rien n'oblige que ces unités soient réalisées en hard
 - Exemple : Unité flottante émulée en SW, Lorsque l'instruction n'est pas implantée en Hw, le processeur génère une exception, et l'exception handler appelle la routine d'émulation Sw (instruction, div,...)
- ALU : Support d'instructions en virgule fixe et virgule flottante (format IEEE 754 simple précision)
- Interface vers des instructions logiques custom
- Contrôleur d'exceptions/interruption
- Memory management/Protection Units (MMU/MPU)
- JTAG (debug)

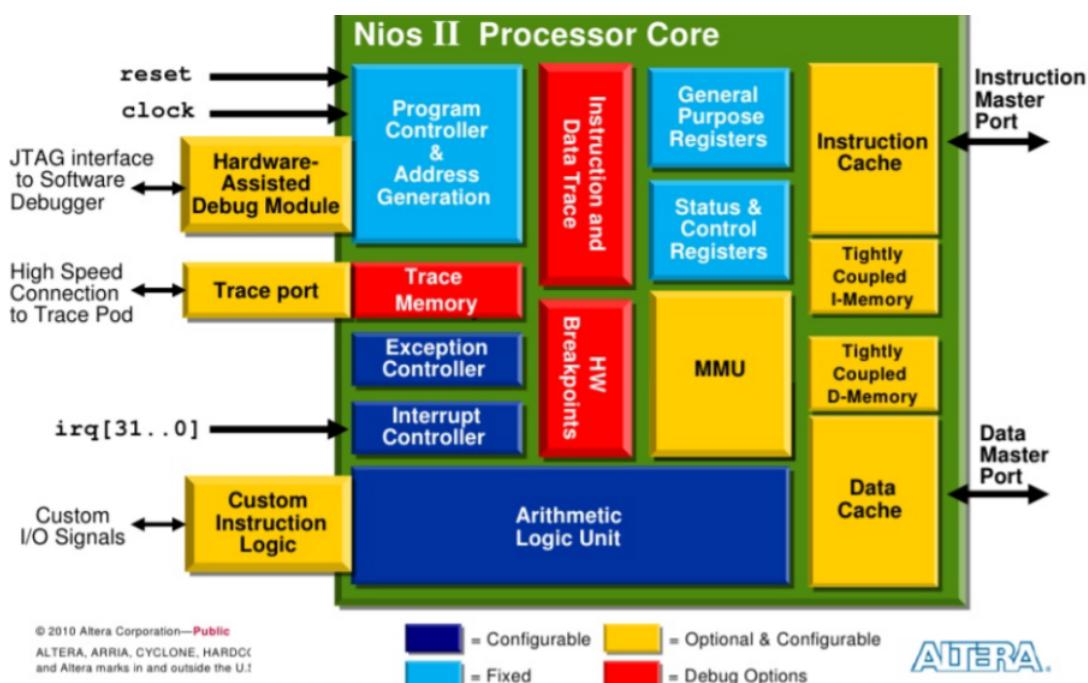


FIGURE 11 – Diagramme de bloc du NIOS II

7.1 REGISTRES GENERAUX DU NIOS II

Le Nios II dispose de 32 Registres 32 bits + 32 registres de contrôle + possibilité de 63 shadow registers permettant l'accélération du changement de contexte (si OS).

7.2 RESET ET SIGNAUX DE DEBUG

Reset :

- Reset global matériel forçant la remise à 0 immédiate du processeur
- `cpu_resetrequest`, remise à 0 du processeur mais pas des autres composants

Debug : `debug_req` suspend l'activité du processeur pour debugger

8 PILOTE DE BARRE FRANCHE

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
- 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.

Le système réel est représenté comme suit :



FIGURE 12 – Pilote automatique pour voilier

8.1 MODE DE FONCTIONNEMENT

Le système est piloté par une barre franche avec deux modes de fonctionnement :



FIGURE 13 – Pilote automatique pour voilier

La barre est constituée de boutons poussoirs pour le pilotage suivant un mode de fonctionnement : Bâbord, Tribord et STBY/AUTO.

8.1.1 MODE DE FONCTIONNEMENT MANUEL

C'est un mode qui permet de positionner le vérin de façon manuelle (le vérin peut être entré et sorti manuellement) en appuyant sur les boutons Bâbord et Tribord . Utilisation du Tiller-pilot comme système de barre motorisée.

A bord d'un bateau, on ne parle de pas de côté gauche ou côté droit, mais de «bâbord» et de «tribord». Bâbord désigne la partie gauche du bateau quand on regarde l'avant et tribord la partie droite.

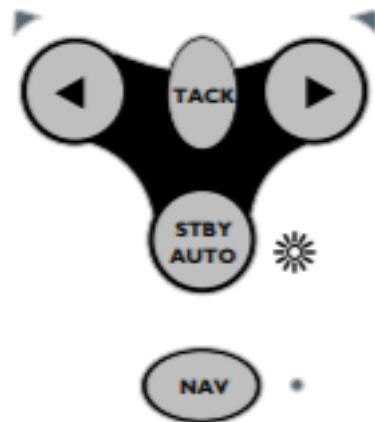


FIGURE 14 – Mode veille

Ce mode est également appelé mode veille, c'est le mode par défaut à la mise en marche de l'appareil.

8.1.2 MODE DE FONCTIONNEMENT AUTOMATIQUE

L'activation du mode automatique se fait par un appui sur la touche STBY/AUTO pour verrouiller le Tillerpilot sur le cap actuel.



FIGURE 15 – Mode veille

Le réglage du cap sur un angle de 1° ou de 10°, est réalisé par des mouvements de la barre vers Babord ou tribord et s'effectue à l'aide des boutons correspondant du clavier.

9 OBJECTIF DU BE - PILOTE BARRE FRANCHE

L'objectif de ce bureau d'étude est de développer le système de barre franche sous forme d'un système sur puce programmable SOPC (System On Programmable Chip) en utilisant le langage VHDL (Very High Speed Hardware Description Langage) sur circuit FPGA DE0.

Etapes de la conception du système barre franche :

- L'analyse de spécifications et le découpage fonctionnel du système en plusieurs sous systèmes.
- La conception de circuits d'interfaces numériques en VHDL (conception, simulation des signaux sur Quartus, vérification du fonctionnement sur maquette).
- Interfaçage avec bus microprocesseur NIOS et Altera Avalon.
- conception d'un SOPC et intégration D'IP (Intellectual Properties) propriétaires et fournisseurs tiers , maîtriser la simulation Hardware In the Loop et la validation du SOPC en simulation.

Le système et ses sous systèmes sont représentés sur la figure suivante :

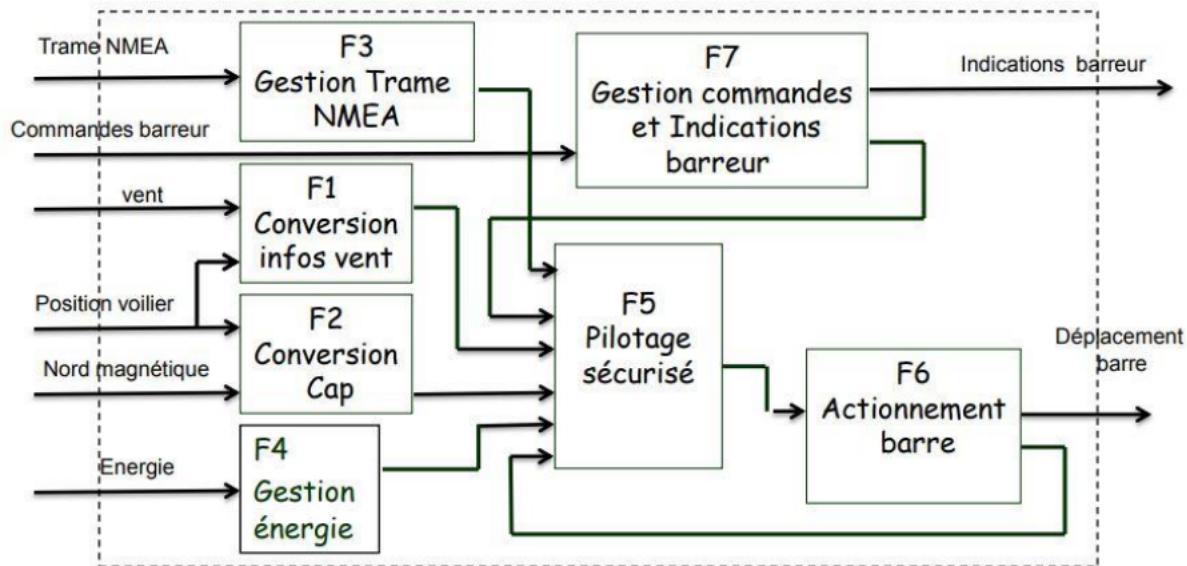


FIGURE 16 – Le système pilote barre franche et ses sous systèmes

10 Contexte du système

Le pilote automatique a pour objectif de gérer les voiliers à barre franche afin d'augmenter les performances et la précision de navigation.

Le système doit disposer de deux modes de fonctionnement : automatique et manuel.

La direction de navigation est en fonction de l'angle de la barre. Celle-ci est contrôlée par un moteur DC. Les divers capteurs (boussole, GPS) et autres informations permettent de donner au système les éléments de trajectoire réel.

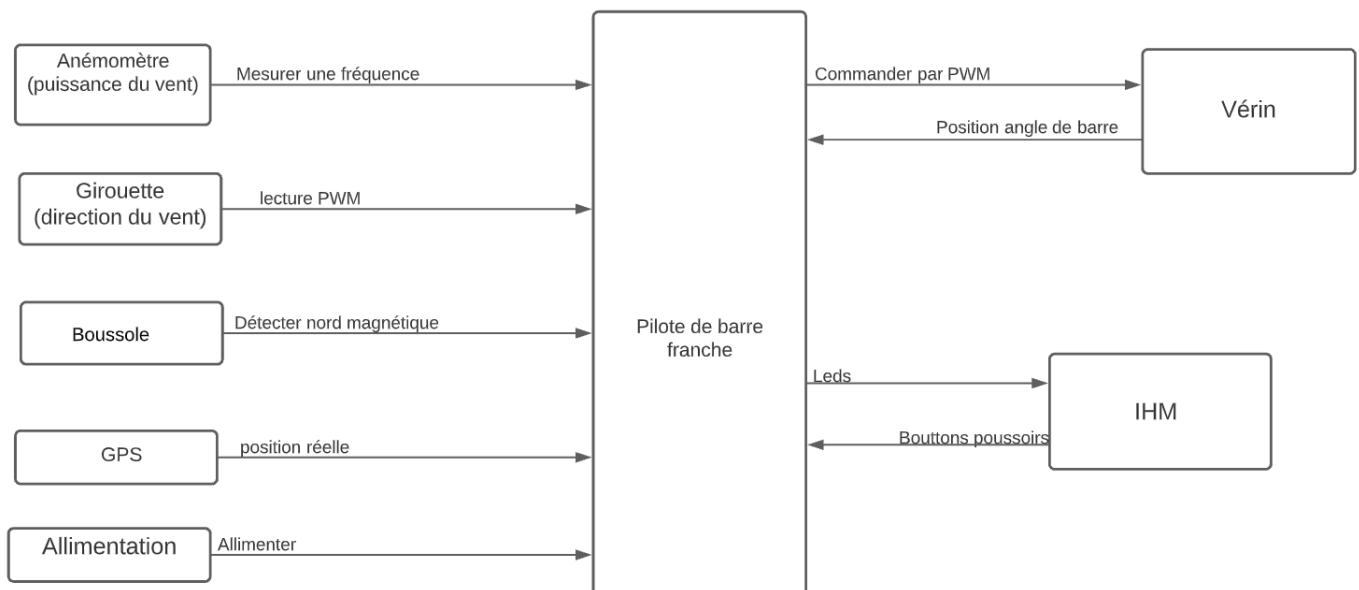


FIGURE 17 – système complet

11 GENERATION SIGNAL PWM

Pour le signal PWM, on veut avoir un signal avec une fréquence fixe et un rapport cyclique (duty) variable. Le signal PWM est généré à l'aide d'un compteur, d'un diviseur et d'un comparateur, les résultats de simulation sont sur la figure suivante :

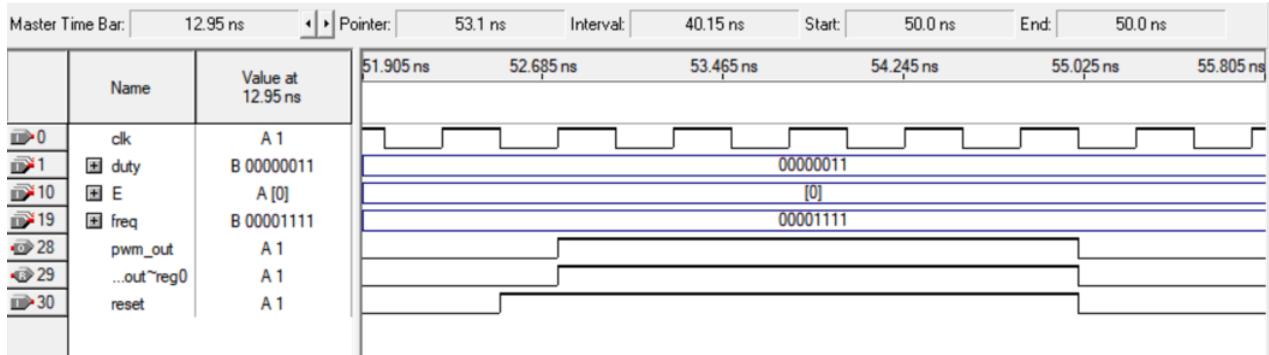


FIGURE 18 – Simultaion signal PWM

Pour générer l'architecture des blocs sur Quartus 18 :

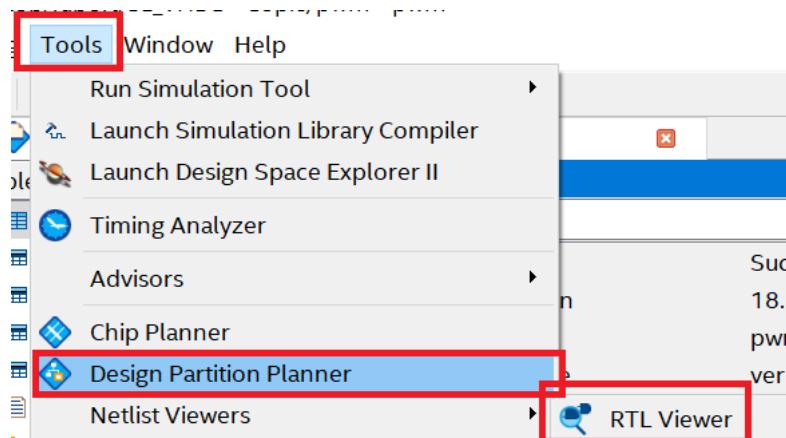


FIGURE 19 – L'architecture des blocs sur Quartus 18

L'architecture de la PWM :

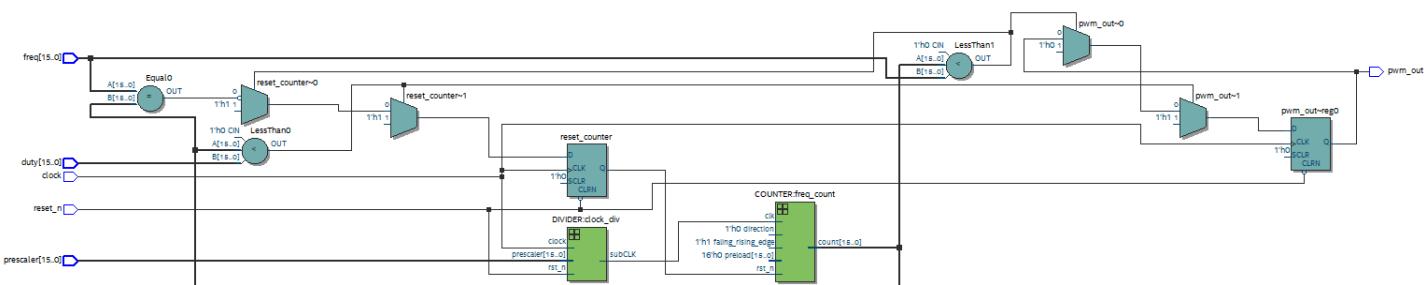


FIGURE 20 – L'architecture du bloc PWM

11.1 SIMULATION SUR OSCILLOSCOPE

Nous avons simulé notre signal PWM sur l'occilloscope après la configuration graphique des pins sur Quartus 9 (carte DE2).

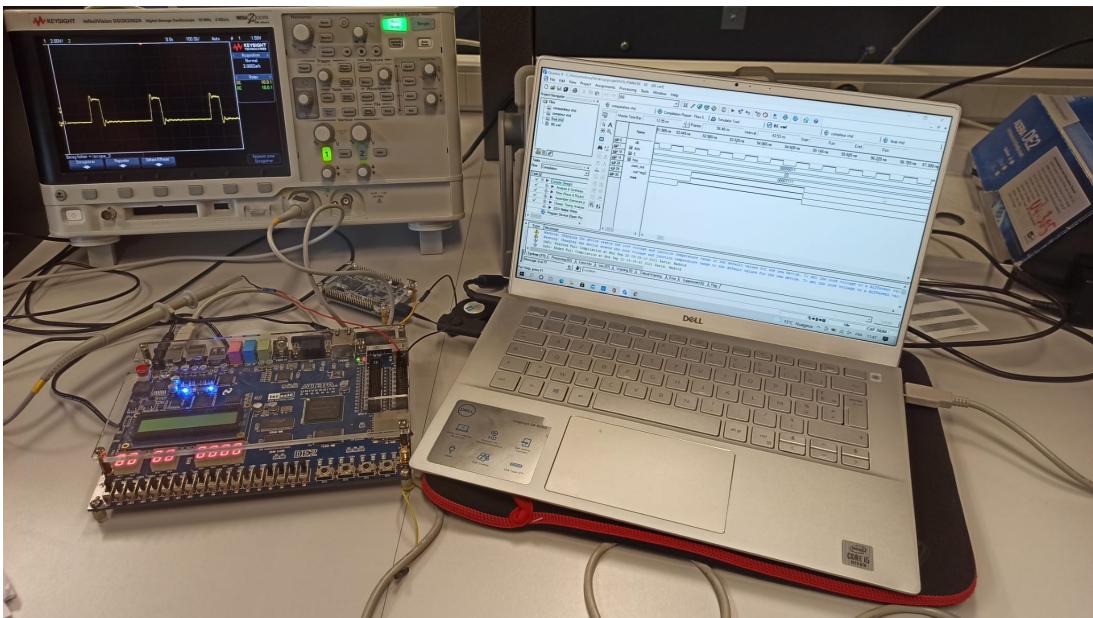


FIGURE 21 – Simultaion signal PWM sur occiloscope

Explication : Nous avons la fréquence en binaire : 00001111 et le rapport cyclique (duty) 00000011 affichés à l'écran. Nous avons programmé 8 Toggle Switch de fréquence et 8 Toggle Switch de rapport cyclique, la clock 50 MHZ (CLOCK_50 PIN_N2 50 MHz clock input), le reset (KEY0 PIN_G26 Pushbutton[0]), le pwm_out est programmé sur le GPIO0 (GPIO_1[0] PIN_K25 GPIO Connection 1[0]) et branché à l'occiloscope.

Pour que le signal de fréquence varie, pour un front montant d'horloge, il faut que la valeur de la fréquence soit supérieur à celle du rapport cyclique (c'est le rôle du comparateur).

12 ANEMOMETRE

Le but de l'anémomètre est de calculer la vitesse du vent, à partir de la fréquence d'entrée. La mesure dure une seconde. Nous obtenons la vitesse du vent, et donc sa fréquence, en comptant le nombre de fronts montants de la freq_anemometre.

Exemple :

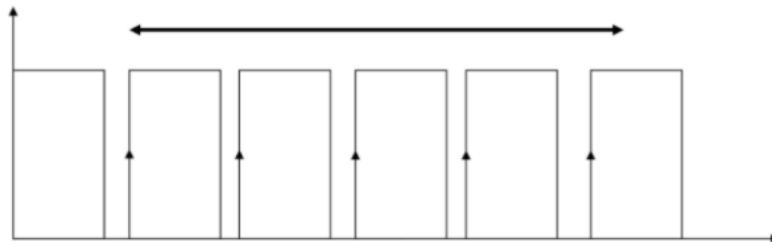


FIGURE 22 – Exemple

Nous remarquons qu'il y'a 5 fronts montants de la fréquence, cela veut dire que la vitesse du vent est égale à 5km/h.

12.1 ETAPES DE REALISATION :

- Répartition de la fonction Anémomètre en blocs internes.
- Comprendre l'enchaînement des blocs (Machine à états)

- Codage et simulation de chaque bloc
- Test du fonctionnement de ma fonction sur la carte DE2
- Implémentation du bus Avalon sur le SOPC
- Développement logiciel sur NIOS II
- Test du fonctionnement de notre fonction sur la carte DE0

La figure suivante représente les entrées et les sorties du bloc "anémomètre"



FIGURE 23 – Entrées/sorties de l'anémomètre

Voici le circuit de l'anémomètre réalisé sur Qaurtus :

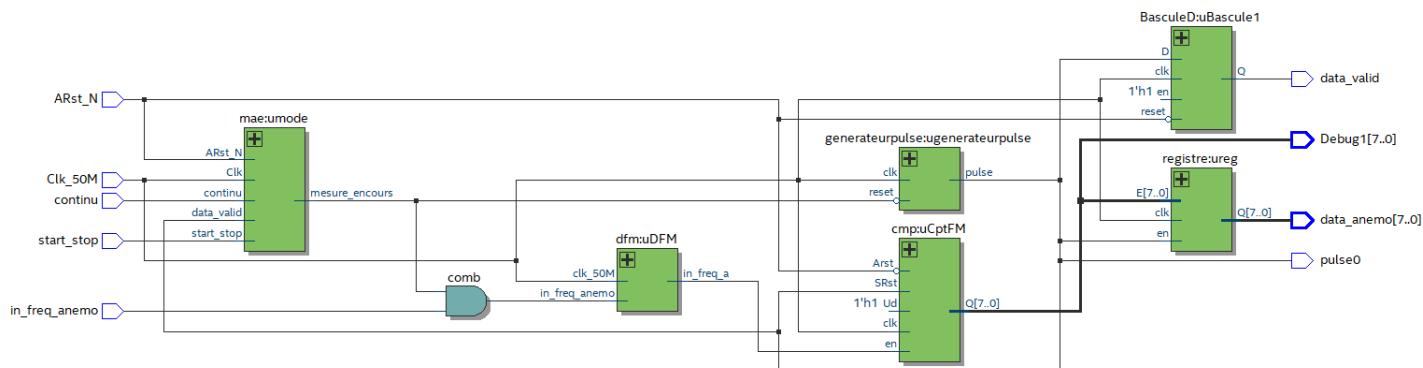


FIGURE 24 – Circuit anémomètre

Ici on doit créer le bloc anémomètre qui reçoit un signal physique à l'entrée qui est la vitesse du vent, le signal sera transformé en signal numérique carré et la fréquence de ce signal doit être en sortie égale à 1Mhz.

Le détecteur de front montant du signal in_freq_anemo qui a comme entrées (Clk 50M et in_freq_anemo) et comme sortie (in_freq_a) qui sera l'entrée du compteur.

Le compteur qui compte le nombre de front montant, permet de mesurer la fréquence du signal numérique in_freq_anemometre..

Le générateur d'impulsion (diviseur) qui dit au compteur : tu arrêtes de compter au bout de 1s puis remet le compteur à 0 au bout de 1s (génère un signal de 1 Mhz chaque 1s).

La bascule sert à mémoriser une donnée.

Le registre reçoit la valeur du compteur chaque 1 s et donne en sortie le signal 1Mhz.

Nous avons créé une machine à état avec 2 états : C'est une fonction qui gère les modes de mesure de la fréquence, qui sera mémorisée dans une variable de sortie codé de 8 bits nommé data_anemo, lorsqu'une mesure est valide, le circuit met sa sortie data_valid (signal logique) à 1 sinon elle est à 0.

Les état : repos et acquisition (acquisition est l'image d'enable et enable vaut 1 que si on est dans l'état acquisition) qui permet la gestion du circuit en assurant le passage entre les deux états du système. Il existe deux modes de fonctionnement : mode continu et mode monocoup.

En mode continu la donnée est rafraîchie toutes les secondes :

Continu=0 et start_stop passe à = 1 alors mesure_encours passe à 0 que lorsque nous avons un front montant de data_valid (=1) parce que nous avons fini notre mesure (nous sommes prêt à faire une autre mesure). Il faut un effet mémoire (Bascule) si nous n'avons pas de mesure en cours et que start_stop passe à 1 alors mesure_encours prend 1, sinon si nous avons data_valid qui passe à 1 alors mesure_encours prend 0.

Nous cherchons à détecter un évènement qui indique la fin de la mesure.

12.2 MODE MONOCOUP

Au début nous sommes dans un état de repos cela signifie qu'une mesure est en cours, data_valid veut dire qu'une mesure s'est achevée : je demande à faire une acquisition quand start_stop = 1.

Nous savons qu'une acquisition est achevée quand nous avons data_valid = 1 donc à partir de là nous repassons en mode repos.

L'état acquisition est un enable qui autorise à notre anémomètre de faire l'acquisition.

12.3 MODE CONTINU

Le monde continu = 1 c'est quand nous restons tout le temps dans l'état acquisition quel que soit l'état de data_valid, nous ajoutons 1 ou avec la transition de repos vers acquisition.

par exemple quand continu = 1 peu importe la valeur de start_stop nous savons que nous allons être en mode continu, le mode va tout le temps rester dans l'état acquisition et pour éviter de sortir de l'état d'acquisition (nous sortons que si continu = 0).

Nous sortons de l'état d'acquisition vers repos si datavalid = 1 et que continu = 0.

12.4 SIMUMATION ANEMOMETRE

Configiration graphique :

	Node Name	Location
1	ARst_N	PIN_G26
2	Clk_50M	PIN_N2
3	data_anemo[7]	PIN_AC21
4	data_anemo[6]	PIN_AD21
5	data_anemo[5]	PIN_AD23
6	data_anemo[4]	PIN_AD22
7	data_anemo[3]	PIN_AC22
8	data_anemo[2]	PIN_AB21
9	data_anemo[1]	PIN_AF23
10	data_anemo[0]	PIN_AE23
11	in_freq_anemo	PIN_J22
12	<<new node>>	

FIGURE 25 – Simulation Anemomètre

Simulation Test Anemo :

Simumation de l'anémomètre pour une fréquence F = 3hz.

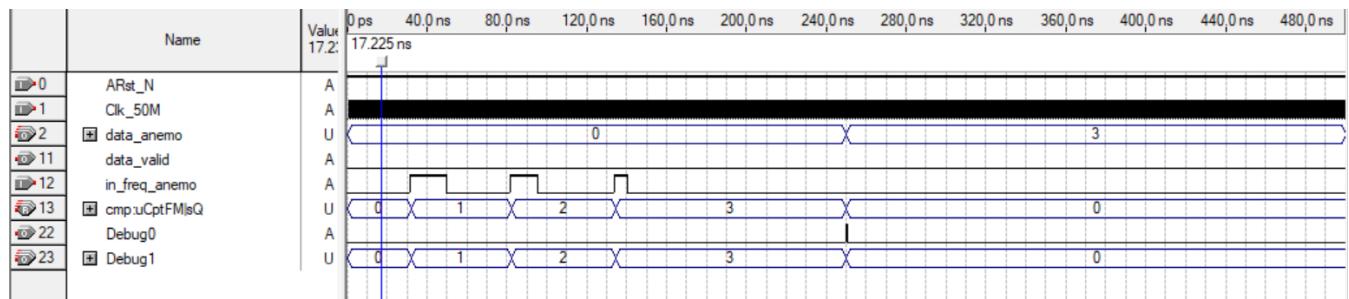


FIGURE 26 – Simulation Anemomètre 3 pulse F = 3hz

Simulation oscilloscope :

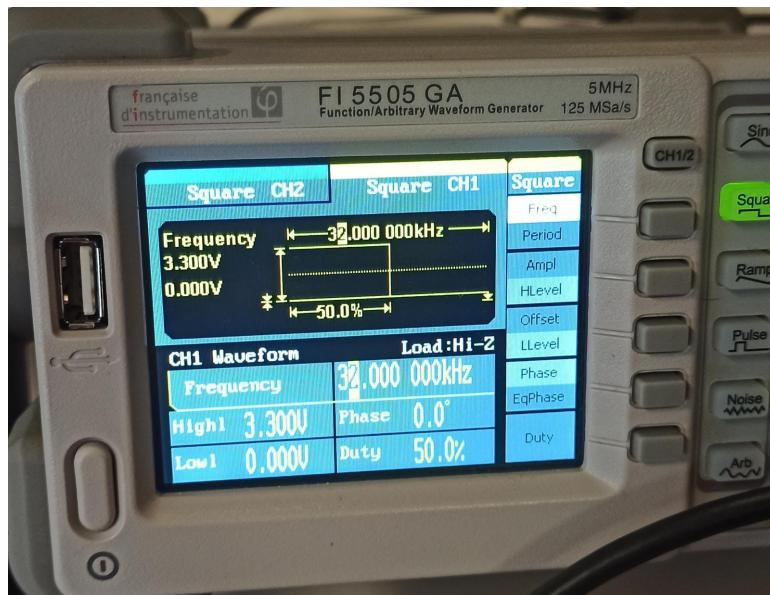


FIGURE 27 – Simulation Anemomètre sur oscilloscope pour F=32KHz

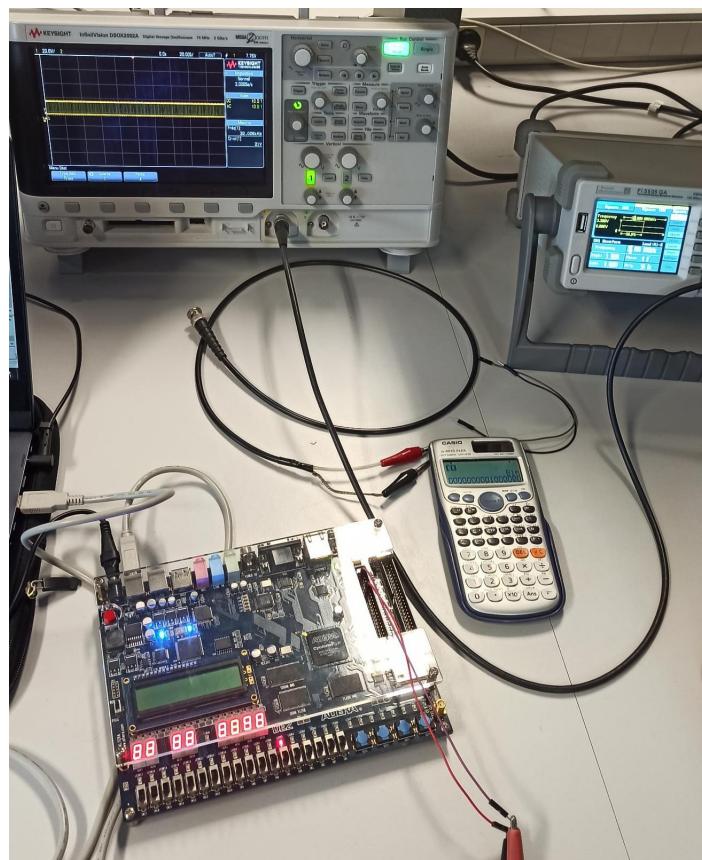


FIGURE 28 – Simulation Anemomètre sur oscilloscope CODE BINAIRE 100000

Donc sur la simulation, nous avons bien une fréquence qui correspond à un nombre binaire de 00100000 qui correspond à 32 en base décimale.

13 BUS AVALON DE L'ANEMOMETRE

L’avalon est un bus informatique développé par la société Altera et destiné à l’implémentation sur du matériel programmable. Son rôle est d’assurer l’interconnexion entre le processeur (NIOS II) et des circuits périphériques. c’est une spécification d’interfaces pour des composants sur puce qui définit les transferts entre un ou plusieurs périphériques et une structure d’interconnect. Avalon est un système de communication maître-esclave.

Les différentes entrées sorties de l’interface Avalon sont les suivantes :



FIGURE 29 – Entrées/sorties du bus avalon de l’anémomètre

l’interface Avalon définit un certain nombre de signaux :

- Lignes séparées pour :
 - Les adresses
 - Les données (entrantes et sortantes si pas tristate)
 - Le contrôle
- Lignes de données de largeur jusqu'à 1024
- Opérations synchrones
- Performances jusqu'à un transfert par cycle
- Actifs à l'état haut sauf si explicitement suivi de '_n' (read_n)

Circuit Anémomètre avalon :

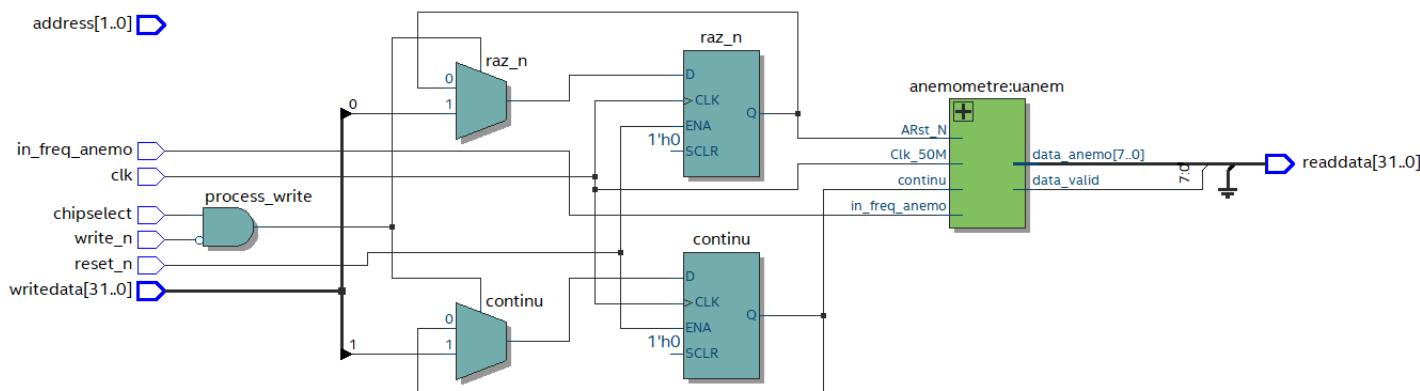


FIGURE 30 – Circuit Anémomètre Avalon

13.1 INTEGRATION DU SOPC :

Pour générer notre système sur puce, nous ajoutons le bus avalon de l'anémomètre en utilisant l'outil SOPC Builder.

Use	Connections	Name	Description	Export	Clock	Base	End	I...	Tags
<input checked="" type="checkbox"/>	clk_in_reset clk clk_reset	cpu	Nios II Processor	reset Double-click to Double-click to					
<input checked="" type="checkbox"/>	clk reset data_master instruction_m... irq debug_reset_r... debug_mem... custom_instru...	ram	On-Chip Memory (RAM o...	Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to Double-click to	clk [clk] [clk] [clk] [clk] [clk] [clk]	IRQ 0 0x8800	IRQ 31 0xffff		
<input checked="" type="checkbox"/>	clk reset avalon_jtag_si... irq	jtag	JTAG UART Intel FPGA IP	Double-click to Double-click to Double-click to Double-click to	clk [clk] [clk] [clk]	0x9020	0x9027		
<input checked="" type="checkbox"/>	clk reset s1 external_conn...	leds	PIO (Parallel I/O) Intel F...	Double-click to Double-click to Double-click to	clk [clk] [clk]	0x9000	0x900f		
<input checked="" type="checkbox"/>	avalon_slave_0 clock reset conduit_end	avalon_anem...	Avalon Memory Mapped ...	Double-click to Double-click to Double-click to	[clock] clk [clock]	0x9010	0x901f		

FIGURE 31 – SOPC

CPU, NIOS II : processeur. RAM : Mémoire du FPGA double port. Jtag : permet d'avoir une interface série et un Jtag pour débuger. PIO : des entrées sorties pour pouvoir contrôler des Leds.

14 COMMUNICATION MAITRE DE0 ET ESCLAVE AN MCP 3201

Un composant Maître initie les transactions sur le bus soit en envoyant directement des données, soit en émettant des requêtes aux composants esclaves, la carte DE0 est le maître.

Un composant esclave ne prend jamais l'initiative d'utiliser le bus (en lecture ou écriture), le convertisseur AN MCP 3201 est l'esclave. Il ne fait que répondre aux requêtes des maîtres.

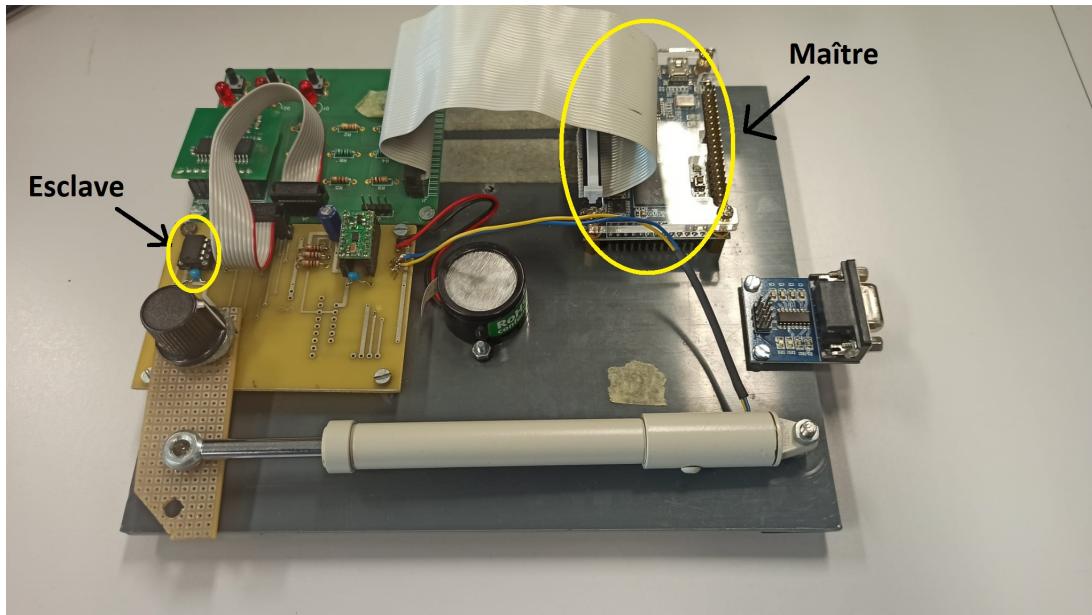


FIGURE 32 – Maître DE0 - esclave convertisseur AN MCP 3201

15 VERIN

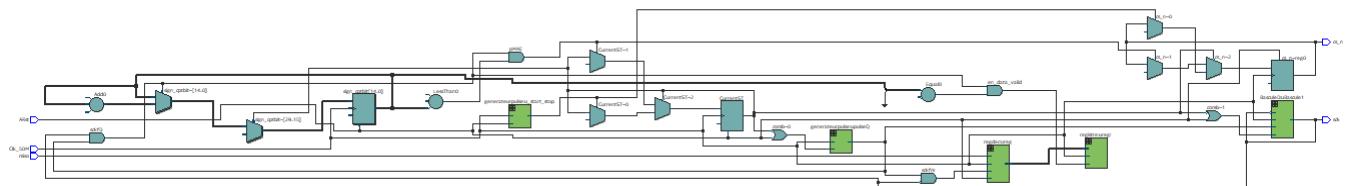


FIGURE 33 – Circuit du vérin

Le but c'est d'avoir une PWM avec une fréquence fixe et un rapport cyclique variable pour faire entrer et faire sortir le vérin, plus le rapport cyclique est grand plus la vitesse du vérin est grande. La signal sens permet de faire entrer et faire sortir le vérin (la direction du vérin). La communication ici se fait en SPI, donc nous avons un signal SCK qui est l'horloge SPI (signaux physiques qui se trouvent entre le convertisseur AN MCP 3201 et le FPGA).

Le Mosi c'est la data du FPGA vers l'ADC, le Miso c'est le data de l'ADC vers le FPGA, le cs_n indique qu'on s'adresse à l'esclave. L'ADC n'a que le miso, le cs_n et le SCK.

```

126 |      -- signal sck --
127 |      sckfm <= '1' when sign_sck = '0' and sign_pulse = '1' else '0'; -- detecter FM
128 |      sckfd <= '1' when sign_sck = '1' and sign_pulse = '1' else '0'; -- detecter FD
129 |
130 |      sck <= sign_sck;
131 |
132 |      -- signal recuper --
133 |      en_data_valid <= '1' when sckfd = '1' and sign_cptbit = 14 else '0';
134 |
135 |
136 |      -- chipeselect etat bas puis etat haut
137 |      -- start_stop : si start_stop = 1 , cs = 0 , et on autorise generateurpulse
138 |      -- on autorise le detecteurFD a cpmter sur chaque fd
139 |      -- on veut etre sur qu on arrive jusquau dernier fd = 15
140

```

FIGURE 34 – SCK FD FM

Pour le convertisseur AN MCP 3201 on récupère les datas sur front montant de SCK, cela indique que CPHA = 0.

Il faut créer un composant qui permet de générer une horloge et qui permet de récupérer les données sur front montant, donc on génère des pulses grâce à notre generateurpulse (dans un process, à chaque fois qu'on est sur un front montant, on récupère la donnée).

Registre à décalage : permet de faire un décalage, avec une autorisation, donc le front montant correspond à un enable pour le registre à décalage. Le compteur cptbit, à chaque front descendant, il compte de 1 (le front descendant correspond à un enable pour le cptbit) donc la donnée est valide quand le cptbit vaut 15 et SCK est sur un front montant (on récupère la donnée valid avec un registre, tel que l'enable de ce registre prend 1 when SCKFM = '1' and cptbit = 15 else 0).

La figure suivante représente les blocs cités ci dessus :

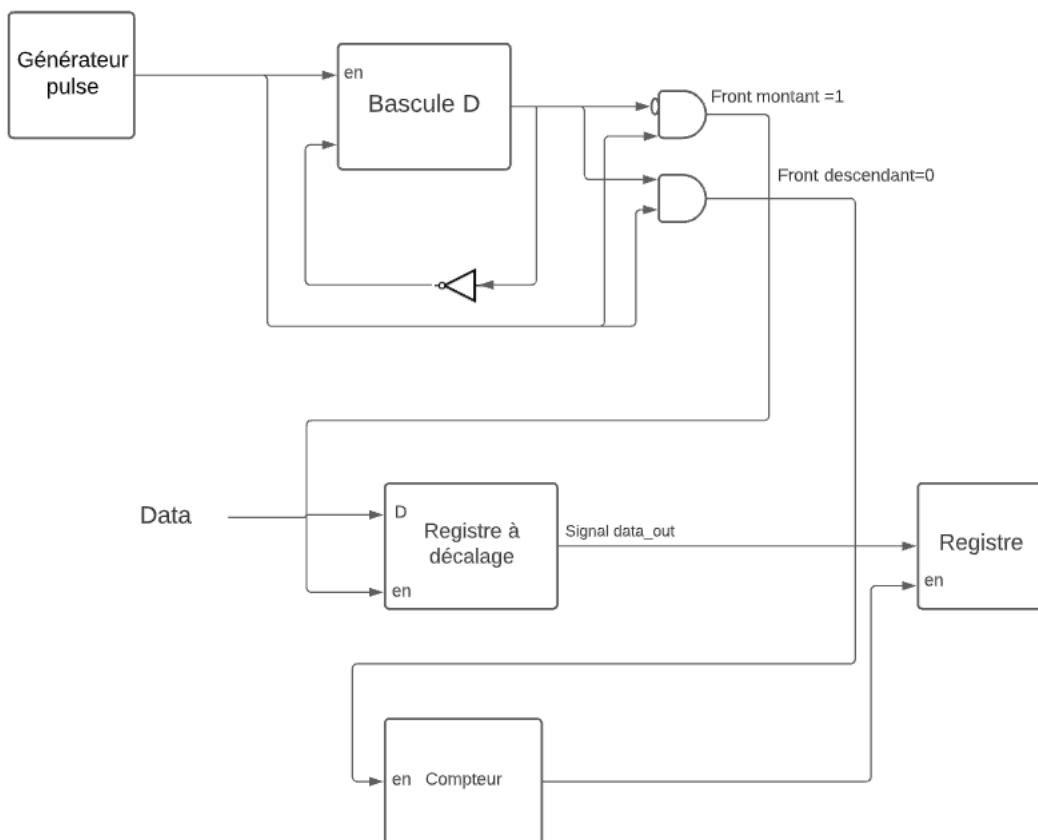


FIGURE 35 – Schéma verin

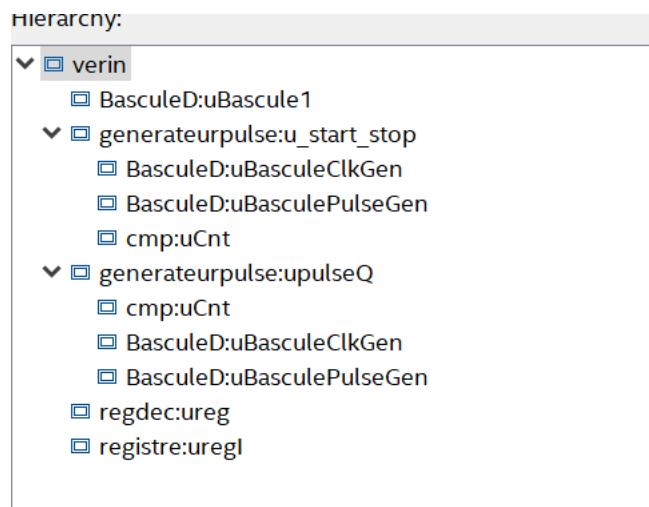


FIGURE 36 – Les blocs du vérin

15.1 ARCHITECTURE DE QUELQUES BLOCS

Architecture du registre :

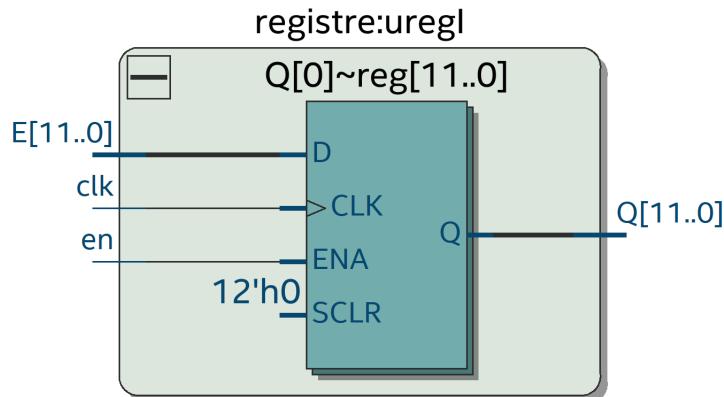


FIGURE 37 – Registre

Architecture du registre à décalage :

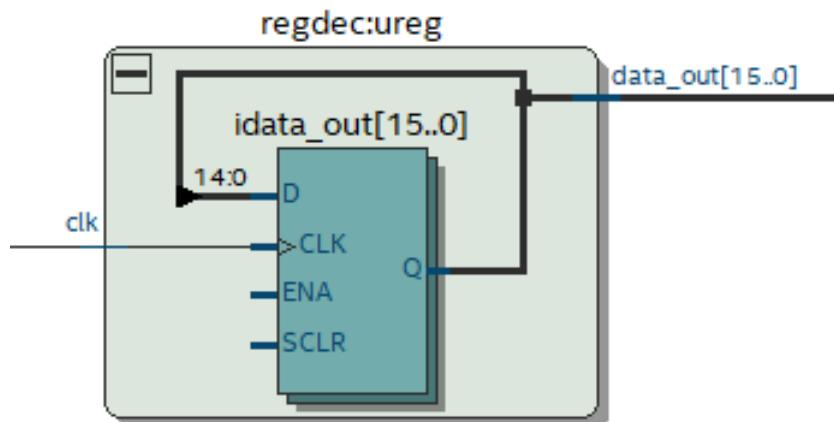


FIGURE 38 – Registre à décalage

Architecture de la basculeD :

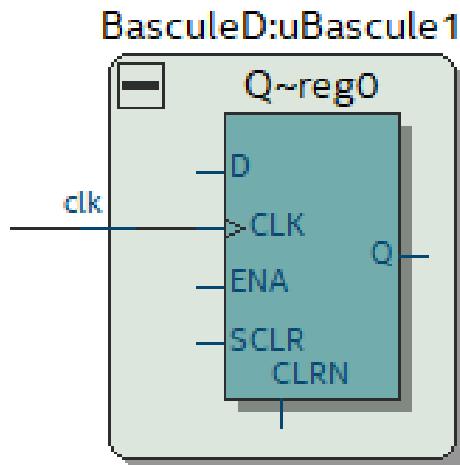


FIGURE 39 – BasculeD

Son rôle est de mémoriser une donnée ou stabiliser un signal.

Architecture de la bascule clkgen :

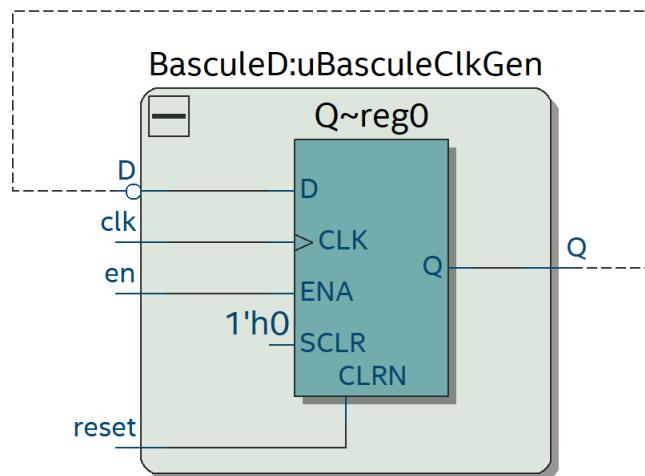


FIGURE 40 – BasculeClkgen

- la bascule D "uBasculeClkGen" permet de générer une horloge de rapport cyclique de 50% dont la fréquence est la suivante :

$$\text{freq_out} = \text{freq_in} / (2 * \text{compare_value})$$

freq_in : fréquence du signal clk

freq_out : fréquence du signal iClkDiv

Architecture de la bascule pulsegen :

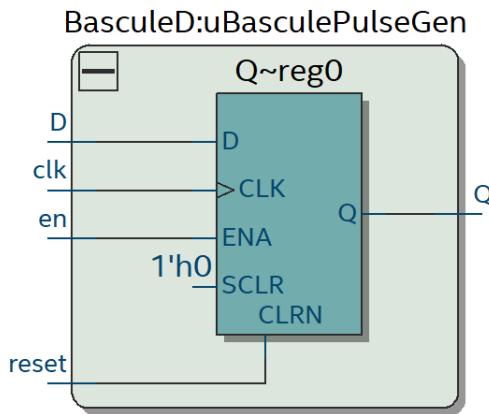


FIGURE 41 – Basculepulsegen

- la bascule D "uBasculePulseGen" permet de générer des impulsions à la fréquence suivante :

$$\text{freq_out} = \text{freq_in} / \text{compare_value}$$

freq_in : fréquence du signal clk

freq_out : fréquence du signal $i\text{ClkDiv}$

Architecture du generateur d'impulsions :

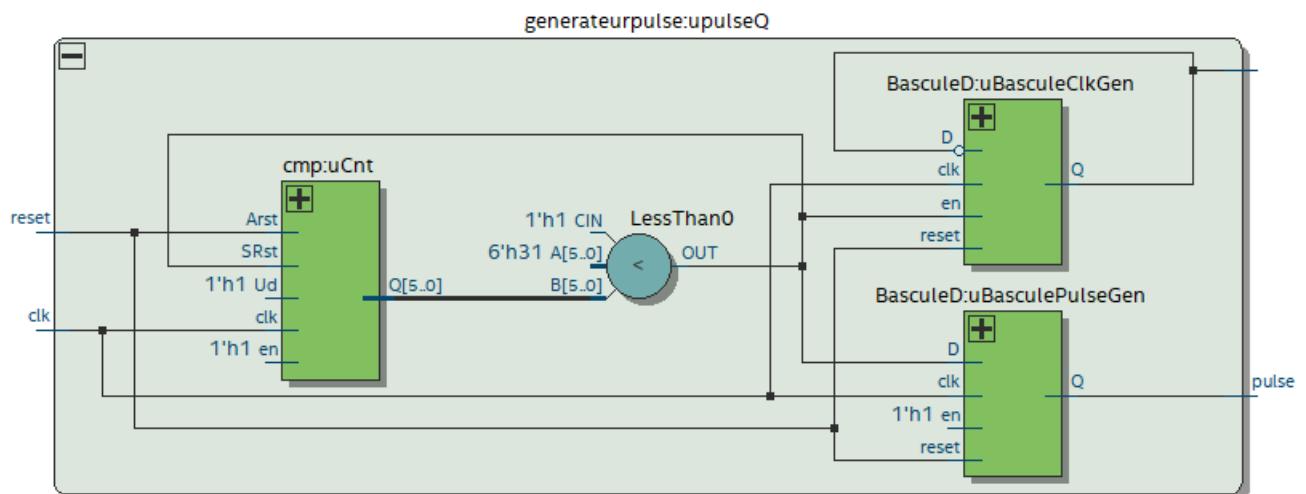


FIGURE 42 – Générateur d'impulsions

Générer des impulsions, c'est un diviseur.

L'architecture du compteur :

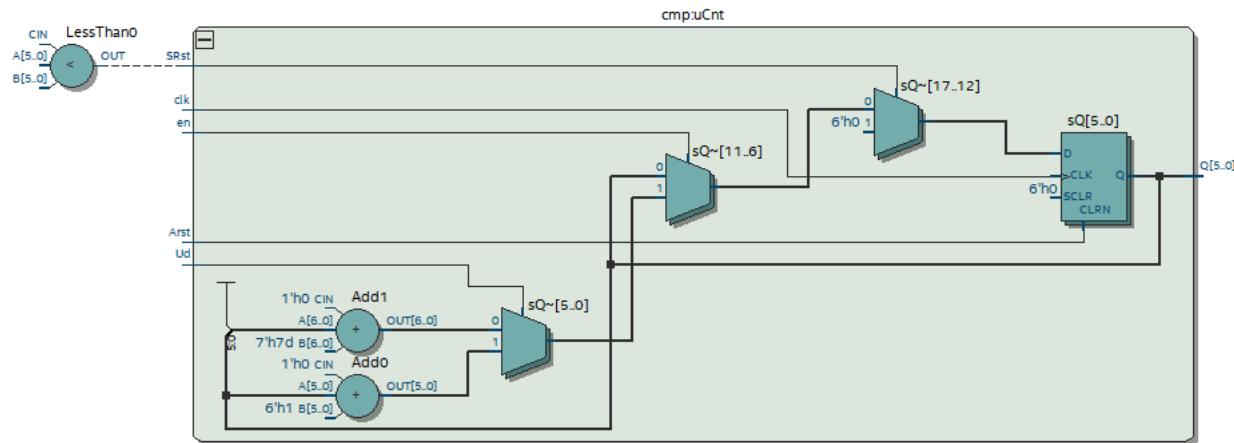


FIGURE 43 – L'architecture du bloc compteur

15.2 SIMULATION VERIN



FIGURE 44 – Simulation Verin

Nous avons bien une remise à 0 de notre compteur Cptbit à 15 et à un front montant de SCK et une mémorisation data.

15.3 SPECIFICATIONS AVALON

Il faut ajouter un contrôleur sur le SoC qui traduit le protocole Avalon dans l'interface de la mémoire choisie : RAM.

Le bus Avalon définit :

- un ensemble de signaux
- le comportement des périphériques
- les types de transfert supportés par ces signaux
- Chaque périphérique est connecté par un (ou +) port(s) M/S

16 CONCLUSION

Grâce à ce bureau d'études, nous avons pu acquérir plusieurs notions de base. Il nous a permis de monter en compétences et ce, en ayant appris une méthode de travail fondée sur la réflexion, l'observation et l'analyse.

Nous avons aussi pris en main plusieurs outils qui nous seront très utiles, nous avons également obtenu une première expérience solide sur une technologie fortement utilisée dans le milieu industriel.

Ce fut aussi l'opportunité de mettre en pratique les bases théoriques que nous avons acquises lors de notre formation, que ce soit en ingénierie système, en gestion de projet, ainsi qu'en codage VHDL.

Au terme de ce projet, les résultats obtenus sont assez satisfaisants à nos yeux, mais notre travail n'est pas terminé car la plus grande difficulté pour nous était de pouvoir avancer rapidement sur le projet. Cela n'empêche que la compétence VHDL est acquise pour nous, l'objectif de cet apprentissage est atteint.

Le bilan est donc favorable.