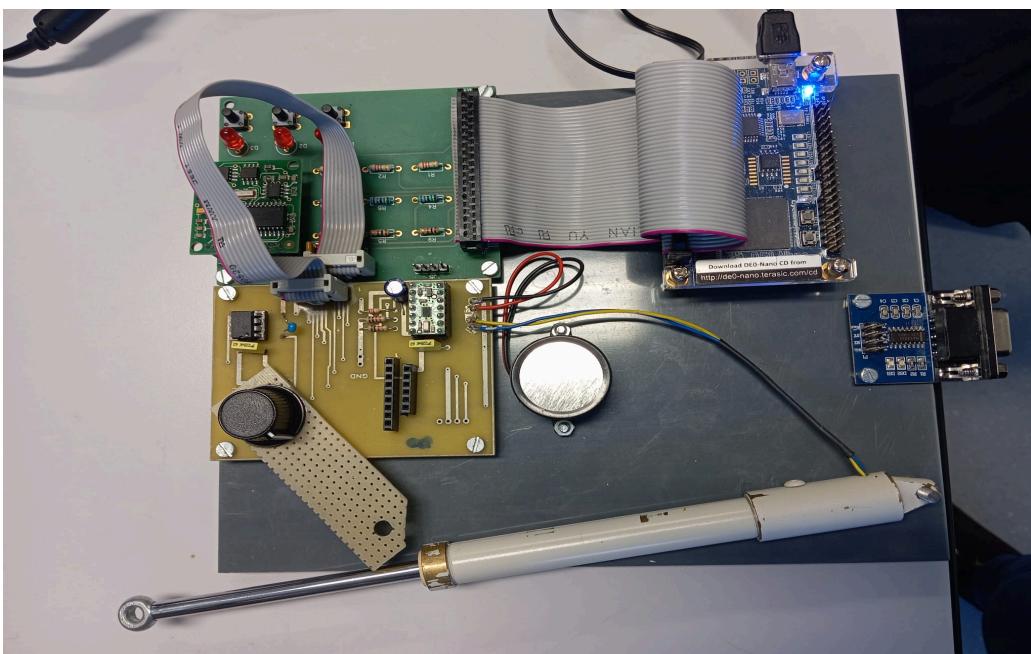


M2 Systèmes Microsystèmes Embarqués  
UE Synthèse et mise en œuvre des systèmes

**Bureau d'Étude:**  
**Pilote de barre franche**



**Rédigé par:**

MESSAOUDI Sara  
NOÉ JONAS HOUNTONDJI Monique

**Encadré par:**

M. PERRISE Thierry

# Table des matières

<u>I. INTRODUCTION</u>	2
<u>II. Présentation du BE</u>	3
1. Analyse des spécifications et découpage fonctionnel	3
2. Matériels et outils	4
2.1. Carte DE0 – Nano	4
2.2. La carte TERASIC DE2-C35	4
2.3. Logiciel QUARTUS	4
2.4. Conception et Configuration du SOPC avec Platform Designer	4
2.5. Partie software	5
<u>III. Conception de la fonction simple: gestion de l'anémomètre</u>	5
1. Analyse fonctionnelle	5
2. Composants et Fonctions Principales	6
3. Test et simulation	7
4. Mise en oeuvre du SOPC “anémomètre”	7
5. Partie software (Nios II)	8
<u>IV. Conception de la fonction complexe: gestion vérin</u>	9
1. Analyse fonctionnelle	9
2. Composants et Fonctions Principales	9
3. Test et simulation	10
3.1. Test de l'ADC seul:	10
3.2. Test de la fonction vérin:	11
4. Mise en oeuvre du SOPC “vérin”	11
<u>IV. Bonus: gestion de boutons poussoirs</u>	13
1. Analyse fonctionnelle	13
Entrées principales :	14
Sorties principales :	14
2. Machine à états des boutons poussoirs	15
3. Simulation	16
<u>V. CONCLUSION</u>	17

## I. INTRODUCTION

Dans le cadre du Bureau d'Études (BE), nous avons travaillé sur le développement d'un système embarqué destiné au pilotage automatique d'une barre franche. Ce projet s'inscrit dans une démarche pédagogique visant à approfondir nos compétences en conception numérique, en interfaçage matériel/logiciel et en intégration de systèmes complexes sur FPGA.

Le principal objectif du BE est de développer une solution technique fonctionnelle en utilisant une approche de co-design. Cela inclut la conception en VHDL de modules numériques, leur simulation et leur validation, ainsi que leur intégration au sein d'un Système sur Puce (System on Programmable Chip: SOPC). Une attention particulière est portée sur l'utilisation de la carte FPGA DE0 Nano, équipée du processeur Altera Cyclone IV, dont les ressources matérielles nous permettent de répondre aux besoins spécifiques de ce projet.

La carte FPGA DE0 Nano est un outil puissant et polyvalent, disposant de 22 320 éléments logiques, 594 Kbits de mémoire SRAM intégrée, et 66 multiplieurs matériels. Ces caractéristiques rendent possible la mise en œuvre de circuits complexes, tout en intégrant des interfaces numériques avancées. De plus, elle est équipée de plusieurs E/S (153), offrant une connectivité suffisante pour piloter la barre franche et communiquer avec des capteurs ou des actionneurs externes.

Ce rapport s'organise en plusieurs parties : nous commencerons par une présentation des objectifs terminaux du BE et des spécifications techniques, suivie d'une explication détaillée des outils et technologies utilisés. Ensuite, nous décrirons la méthodologie de conception employée, en insistant sur les étapes de simulation, d'intégration et de validation. Enfin, nous présenterons les résultats obtenus avant de conclure sur les compétences acquises à travers ce projet, nous visons à maîtriser les fondamentaux de la conception de systèmes embarqués complexes tout en répondant aux défis techniques posés par le pilotage d'une barre franche dans un environnement réaliste.

## II. Présentation du BE

### 1. Analyse des spécifications et découpage fonctionnel

Un pilote de barre franche est un dispositif destiné à maintenir la direction d'un bateau grâce à un compas qui mesure l'écart par rapport à la trajectoire souhaitée. Le système compare en continu la direction réelle et le cap prévu, puis active un vérin pour corriger la déviation. Il intègre plusieurs fonctionnalités clés :

- ★ Mesure de la Fréquence et Détection des Données de Vitesse du Vent : Le système doit pouvoir mesurer la fréquence et détecter les informations relatives à la vitesse du vent, fournies par l'anémomètre.
- ★ Collecte des Informations de Trajectoire Réelle de la Boussole : Il doit récupérer les données de trajectoire réelle qui sont transmises par une boussole.
- ★ Acquisition de la Trajectoire Réelle via GPS : Le système doit être capable de récupérer la trajectoire réelle à partir des données fournies par un GPS.
- ★ Interaction avec l'Utilisateur : Le système doit interagir avec l'utilisateur au moyen de LEDs, de boutons et de signaux sonores.
- ★ Commande du vérin : Il doit pouvoir commander le vérin, qui est un élément clé pour ajuster la direction du bateau.
- ★ Détection de l'Écart d'Angle de Déviation avec la Girouette : Enfin, le système doit être capable de détecter et de récupérer l'écart d'angle de déviation par rapport à la direction indiquée par la girouette.

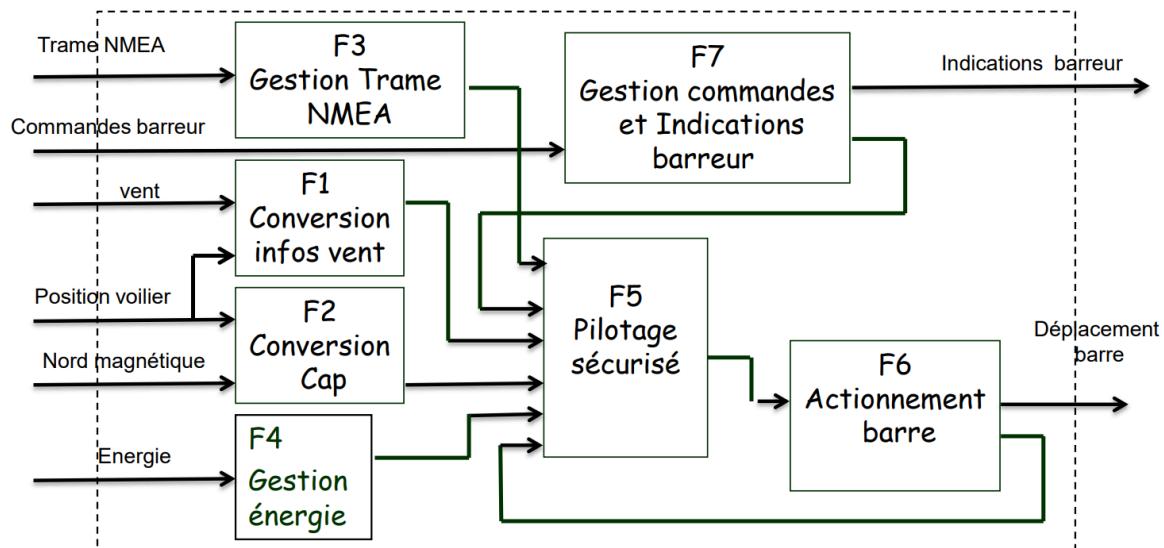


Figure 1 - Découpage en blocs du projet

Nous remarquons que le système à concevoir est un système complexe. Cependant, nous avons opté pour deux fonctions spécifiques. Une fonction qu'on qualifie de fonction simple: Gestion de l'anémomètre, et la deuxième fonction qu'on qualifie de fonction complexe: Gestion du vérin. Nous avons également travaillé comme bonus sur la fonction "Gestion de Boutons Poussoirs".

## 2. Matériels et outils

### 2.1. Carte DE0 – Nano

La DE0-Nano est une plateforme de développement FPGA compacte et performante, spécialement adaptée à la conception de prototypes de circuits, notamment pour les applications robotiques et les projets mobiles. Optimisée pour une mise en œuvre simple, elle repose sur le circuit Cyclone IV, qui comprend jusqu'à 22 320 éléments logiques. Ses principaux atouts résident dans sa taille réduite, son poids léger et sa capacité à être facilement reconfigurée sans nécessiter d'équipement supplémentaire. Ces caractéristiques la distinguent des cartes de développement plus généralistes. De plus, elle propose un accès libre aux fichiers de conception du microcontrôleur Propeller à 8 cœurs de Parallax.

### 2.2. La carte TERASIC DE2-C35

La carte DE2, équipée d'un FPGA Altera Cyclone II 2C35, se distingue par sa polyvalence et sa capacité à prendre en charge une large gamme d'applications. Elle intègre divers composants essentiels, notamment des modules de mémoire, des afficheurs, ainsi que des interfaces Ethernet et USB, ce qui en fait une solution idéale pour le développement et le prototypage de systèmes numériques avancés.

### 2.3. Logiciel QUARTUS

Quartus, développé par Intel, est un logiciel dédié à la conception et au développement de circuits logiques sur FPGA. Il permet de concevoir, simuler et programmer ces circuits grâce à ses outils intégrés de synthèse, compilation, routage, simulation et débogage. Largement utilisé pour des projets électroniques complexes, Quartus se révèle indispensable pour développer des solutions sur mesure adaptées aux besoins spécifiques.

### 2.4. Conception et Configuration du SOPC avec Platform Designer

La mise en œuvre d'un Système sur Puce (SOPC) consiste à concevoir un dispositif programmable intégrant les interconnexions entre divers composants tels que le **CPU**, la **RAM**, le **JTAG**, le **SYS ID** et les **PIO**. Ce système inclut également des modules externes à concevoir, comme un **Avalon PWM**, un **compas** et un **anémomètre**.

La première étape consiste à élaborer un module Avalon PWM pour générer un signal de modulation de largeur d'impulsion (PWM) avec des paramètres ajustables, qui sera intégré au système NIOS parallèlement au développement d'un compteur.

Pour configurer le SOPC dans Quartus, l'outil **Platform Designer** est utilisé. Les principaux composants matériels à intégrer sont:

- **Processeur NIOS II** (mode économique)
- **RAM On-Chip** de 20k
- **JTAG UART** pour la programmation avec une priorité d'interruption de 16
- **System ID Peripheral** pour l'identification
- **PIO** pour la gestion des entrées/sorties (2 boutons en entrée et 8 LEDs en sortie).

L'ajout du module Avalon PWM se fait en important le fichier VHDL associé, en définissant une interface avec le signal "out\_pwm", et en procédant à l'analyse des fichiers. Une fois tous les composants configurés, ils sont interconnectés dans Platform Designer, en ajustant notamment les champs reset vector et exception vector pour associer correctement la RAM au CPU.

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
		clk_0	Clock Source							
		clk_in	Clock Input							
		clk_in_reset	Reset Input							
		dk	Clock Output							
		dk_reset	Reset Output							
		nios2_qsys_0	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_request	Reset Output	Double-click to export	[clk]					
		debug_mem_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		custom_instruction	Custom Instruction Master	Double-click to export	[clk]					
		onchip_memory2_0	On-Chip Memory (RAM or ROM) Intel FPGA IP							
		clk1	Clock Input	Double-click to export	clk_0					
		s1	Avalon Memory Mapped Slave	Double-click to export	[clk1]					
		reset1	Reset Input	Double-click to export	[clk1]					
		Boutons	PID (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]					
		external_connection	Conduit							
		Leds	PID (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]					
		external_connection	Conduit	leds_external_connection						
		JTAG_uart_0	JTAG UART Intel FPGA IP							
		clk	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clk]					
		avalon_tag_slave	Avalon Memory Mapped Slave	Double-click to export	[clk]					
		irq	Interrupt Sender	Double-click to export	[clk]					
		sysid_qsys_0	System ID Peripheral Intel FPGA IP							
		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]					
		pwm_0	pwm							
		clock	Clock Input	Double-click to export	clk_0					
		reset	Reset Input	Double-click to export	[clock]					
		avalon_slave_0_1	Avalon Memory Mapped Slave	Double-click to export	[clock]					
		conduit_end_1	Conduit	pwm_0_conduit_end_1	[clock]					

Figure 2 – Configuration SOPC pour pwm

Enfin, le SOPC est inséré dans un schéma de blocs, avec une assignation des ports d'entrée/sortie, avant la compilation finale du projet:

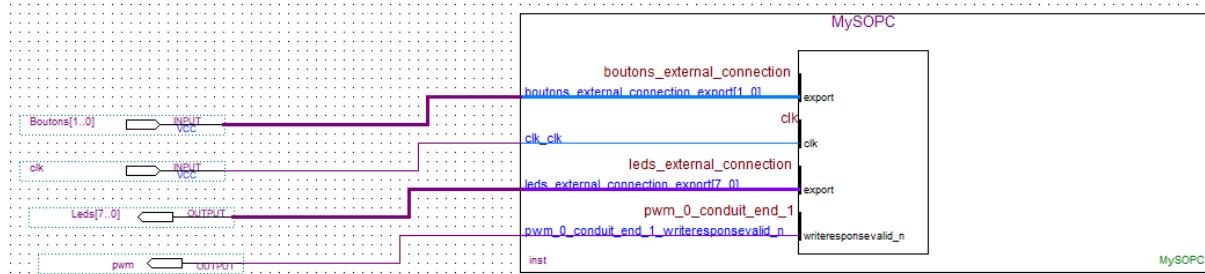


Figure 3 – Bloc SOPC bouton+led+ pwm

## 2.5. Partie software

Dans cette étape, nous utilisons l'outil NIOS II Software Tool ou Eclipse pour initier un nouveau projet dédié à la programmation de notre SOPC. Lors de la création du projet, il est essentiel de sélectionner le fichier "**.sopinfo**", qui servira à l'implémentation et à la programmation. Une fois le projet créé, il doit être compilé puis exécuté à l'aide de l'option "**Run as NIOS Hardware**". Après avoir intégré et exécuté le programme de modulation de largeur d'impulsion (PWM) dans le SOPC, il est indispensable de tester la sortie PWM afin de valider son bon fonctionnement.

### **III. Conception de la fonction simple: gestion de l'anémomètre**

## 1. Analyse fonctionnelle

La mesure de la vitesse du vent, comprise entre 0 et 250 km/h, est réalisée à l'aide d'un anémomètre. La vitesse du vent est ensuite convertie en une fréquence variable, de 0 à 250 Hz.

Pour implémenter la fonction “gestion anémomètre” en respectant le cahier des charges imposé, nous avons élaboré un schéma fonctionnel sous forme de blocs pour bien structurer les différents composants de la fonction.

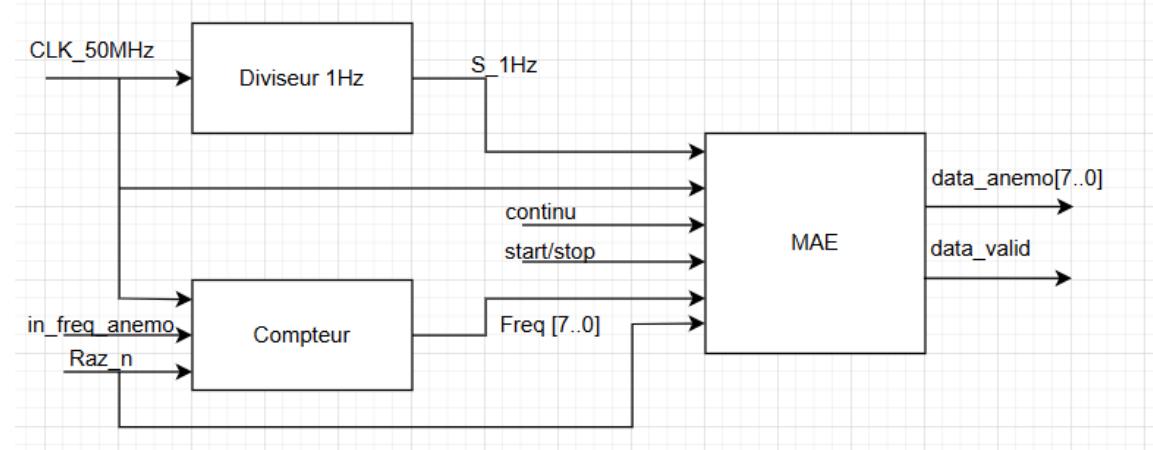


Figure 4 – Schéma bloc pour l'anémomètre

## 2. Composants et Fonctions Principales

### - Diviseur d'Horloge (Diviseur 1Hz)

Ce bloc génère une horloge de 1 Hz, dérivée du signal d'horloge principal (Clk 50MHz) de la carte. Il assure la synchronisation des différents processus du circuit.

### - Compteur

Le compteur mesure la fréquence du signal numérique provenant de l'anémomètre (in\_freq\_anemometre) en comptant les fronts montants. Chaque impulsion correspond à une rotation de l'anémomètre, permettant de calculer la vitesse du vent.

- **MAE:** Synchronisé par une horloge à 1 Hz (*S\_1Hz*), ce module prend en entrée plusieurs signaux : un signal pour le mode continu ou manuel (*continu*), un signal de démarrage/arrêt (*start/stop*), une fréquence d'entrée de l'anémomètre (*freq*), ainsi qu'un signal de remise à zéro (*raz\_n*). Cette MAE nous permet de choisir le mode de fonctionnement de l'anémomètre comme suit :

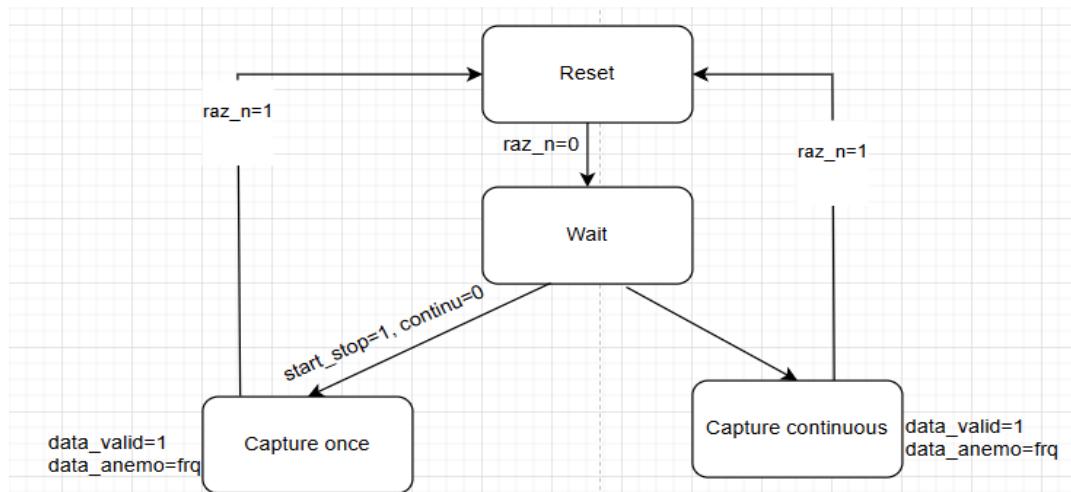


Figure 5 – Machine à état anémomètre

- En **mode manuel** (*continu* = 0), le bloc valide les données uniquement si le signal *start/stop* est activé, permettant ainsi de contrôler manuellement le début et la fin des mesures.
- En **mode continu** (*continu* = 1), les données sont automatiquement validées et mises à jour à chaque cycle de l'horloge 1 Hz.

Les fréquences mesurées sont converties en un vecteur logique de 8 bits (*data\_anemo*), tandis qu'un signal de validation (*data\_valid*) indique si les données sont prêtes à être utilisées. “*data\_anemo*” peut ensuite être affichée ou transmise à d'autres systèmes pour une analyse plus poussée.

### 3. Test et simulation

Dans cette simulation, nous avons testé le mode continu de la fonction anémomètre sur Modelsim. Nous avons fait le test avec 2 fréquences différentes (100hz puis 110hz). On peut constater le bon déroulement de la simulation.

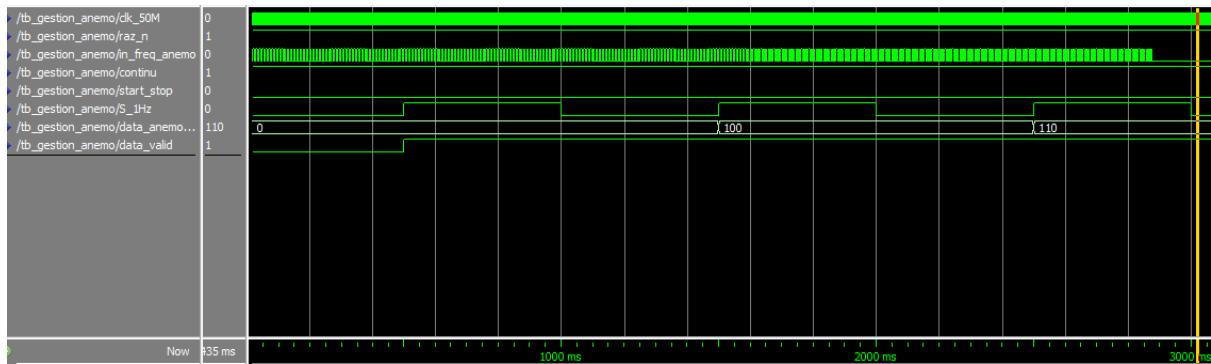


Figure 6 – Résultat de simulation anémomètre

### 4. Mise en oeuvre du SOPC “anémomètre”

Avec Platform Designer, nous assemblons notre composant avec une RAM, un CPU et le bus Avalon afin d'exploiter le processeur et de développer notre système en langage C.

Use	Connections	Name	Description	Export	Clock	Base	End	IRQ	Tags	Opcode Name
		clk_0	Clock Source Clock Input Reset Input Reset Output	clk Double-click to export Double-click to export Double-click to export						
		nios2_qsys_0	Nios II Processor Clock Input Reset Input data_master instruction_master irq debug_reset_request debug_mem_slave custom_instruction_m	clk_0 Double-click to export Double-click to export				IRQ 0	IRQ 31	
		onchip_memory2_0	On-Chip Memory (RAM or ROM) Intel ... Clock Input clk reset1	clk_0 Double-click to export Double-click to export Double-click to export				0x0002_0800	0x0002_0fff	
		Boutons	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input s1 external_connection	clk_0 Double-click to export Double-click to export Double-click to export				0x0001_0000	0x0001_bb7f	
		Leds	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input s1 external_connection	clk_0 Double-click to export Double-click to export Double-click to export				0x0002_1040	0x0002_104f	
		Jtag_uart_0	JTAG UART Intel FPGA IP Clock Input Reset Input avalon_jtag_slave irq	clk_0 Double-click to export Double-click to export Double-click to export Double-click to export				0x0002_1088	0x0002_108f	
		syid_qsys_0	System ID Peripheral Intel FPGA IP Clock Input Reset Input control_slave	clk_0 Double-click to export Double-click to export Double-click to export				0x0002_1080	0x0002_1087	
		pwm_0	jwm clock reset avalon_slave_0_1 conduit_end_1	clk_0 Double-click to export Double-click to export Double-click to export				0x0002_1050	0x0002_105f	
		avalon_anemo_0	avalon_anemo clock reset avalon_slave_0 conduit_end	clk_0 Double-click to export Double-click to export Double-click to export				0x0002_1060	0x0002_106f	

Figure 7 – Configuration SOPC pour anémomètre

Il est donc nécessaire d'intégrer le SOPC dans un schéma bloc, d'assigner les ports d'entrée/sortie du système, puis de compiler le projet.

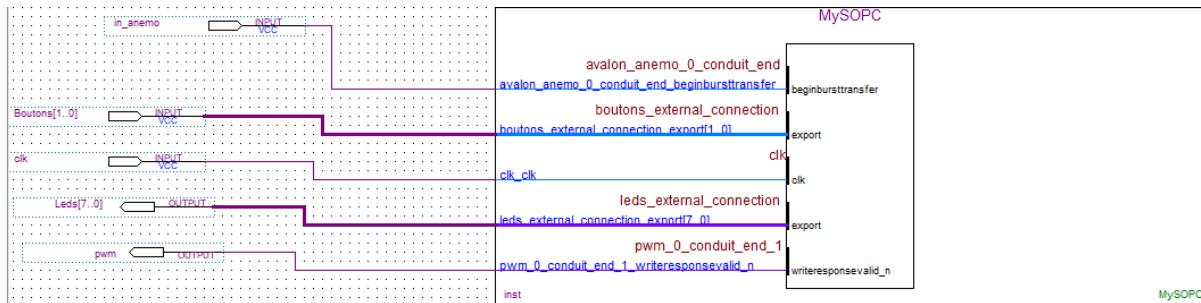


Figure 8 – Bloc SOPC anémomètre

## 5. Partie software (Nios II)

l'interface Avalon\_anemo comporte deux registres principaux :

```
#define config_anemo (unsigned int*) (AVALON_ANEMO_0_BASE)
#define code (unsigned int*) (AVALON_ANEMO_0_BASE + 4)
```

**le registre config:** registre de configuration du circuit ( start/stop, continu et raz\_n)

**le registre code:** il réserve 8 bits pour la valeur de data\_anemometre et un bit pour la vérification de la validité (data\_valid)

Une fois le projet configuré, nous avons développé le code en C pour piloter et tester l'interface Avalon\_anemo. Ce code permet d'interagir avec les registres *config* et *code*, de configurer le circuit, et d'extraire les données de l'anémomètre.

Les résultats suivants montrent les valeurs obtenues pour des fréquences de 9 Hz et 200 Hz et démontrent le fonctionnement global du système.

```

Problems Tasks Console
gestion_anemo Nios II Hardware configuration - cab
Lecture des données
Boutons = 3
data_anemometre = 200
config_anemo = 3
Lecture des données
Boutons = 3
data_anemometre = 200
config_anemo = 3

Problems Tasks Console
gestion_anemo Nios II Hardware configuration - cable: U
Lecture des données
Boutons = 3
data_anemometre = 9
config_anemo = 3
Lecture des données
Boutons = 3
data_anemometre = 9
config_anemo = 3

```

Figure 9 – Affichage données anémomètre sur le terminale NIOS

## IV. Conception de la fonction complexe: gestion vérin

### 1. Analyse fonctionnelle

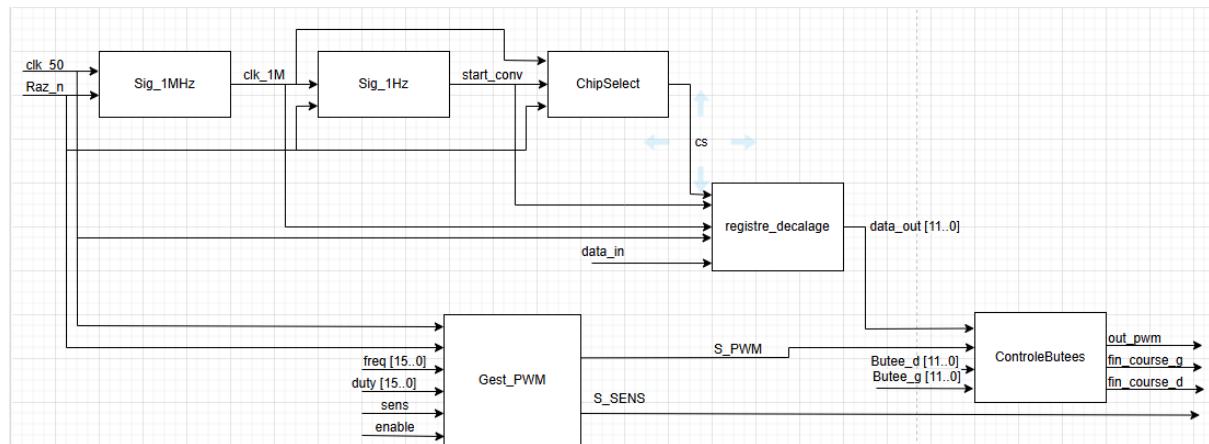


Figure 10 – Schéma bloc pour gestion vérin

Cette fonction vise à générer un mouvement mécanique de la barre du vérin selon le sens souhaité (à gauche ou à droite). En fonction de l'angle de la barre, le moteur du vérin est commandé à l'aide d'un signal PWM pour réguler son déplacement.

### 2. Composants et Fonctions Principales

- **Sig\_1MHz** : Module qui génère une horloge de 1 MHz à partir d'une horloge de 50 MHz. Cette horloge est indispensable pour le fonctionnement du MCP3201 ainsi que pour d'autres blocs du système.

- **Sig\_10Hz** : Génère une horloge de 10 Hz à partir de 50 MHz, utilisée pour produire périodiquement le signal *start\_conv* toutes les 100 ms.
- **ChipSelect** : Produit le signal CS nécessaire au fonctionnement du MCP3201.
- **registre\_decalage**: Registre à décalage permettant de récupérer les données du convertisseur MCP3201. Ces données, correspondant à l'angle de la barre, sont ensuite transmises au bloc suivant du circuit.
- **ControleButees** : Ce module désactive le signal PWM lorsque l'angle de la barre dépasse les limites fixées par *butee\_g* et *butee\_d* (*butee\_d* et *butee\_g* ont été fixées dans le programme), tout en tenant compte du sens de rotation du moteur. Il génère également les signaux *fin\_course\_g* et *fin\_course\_d*.

### 3. Test et simulation

#### 3.1. Test de l'ADC seul:

Pour tester notre programme, nous avons effectué des manœuvres expérimentales.

Tout d'abord, nous avons mesuré la tension maximale délivrée par la maquette. Nous avons relevé une tension

de 4.72V sur la broche VCC (en théorie, la maquette est censée libérer 5V).



En choisissant une position du potentiomètre, nous avons relevé les LED allumées (correspondant aux MSB), converti la trame en décimal et calculé la tension (*Vcalculé*) via un produit en croix avec VCC et la valeur maximale de la trame ( $2^{12}$ ). Nous avons obtenu *Vcalculé* = 2.63V.

En comparant cette tension à la tension mesurée (*Vmesuré* = 2.33V), nous avons constaté que les résultats sont proches, validant ainsi le fonctionnement du programme. Les légères différences sont dues aux 4 bits du LSB ignorés. Ce test a été répété avec différentes positions du potentiomètre, confirmant la fiabilité du code.



### 3.2. Test de la fonction vérin:

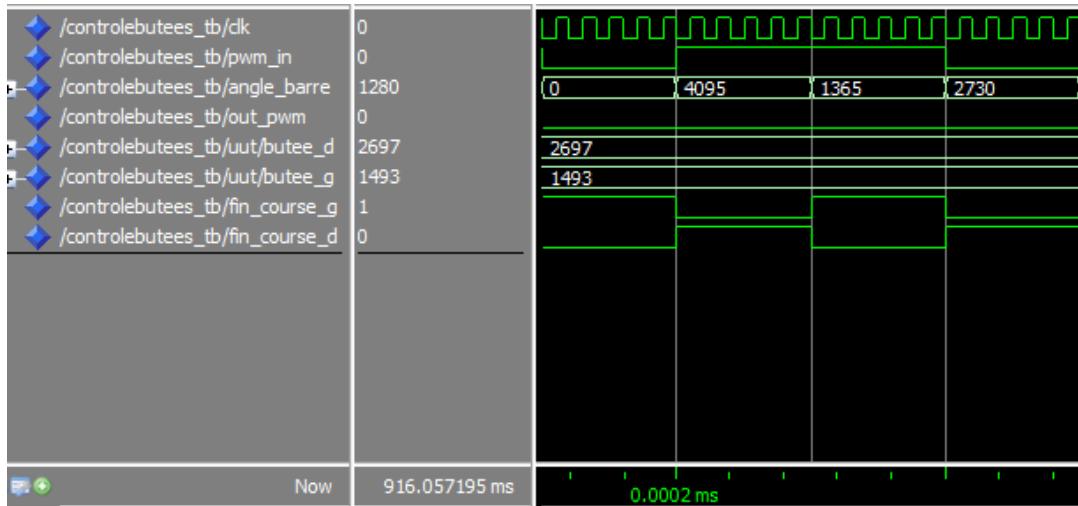


Figure 11 – Résultat de simulation vérin

### 4. Mise en oeuvre du SOPC “vérin”

Avec Platform Designer, nous assemblons notre composant avec une RAM, un CPU et le bus Avalon afin d'exploiter le processeur et de développer notre système en langage C.

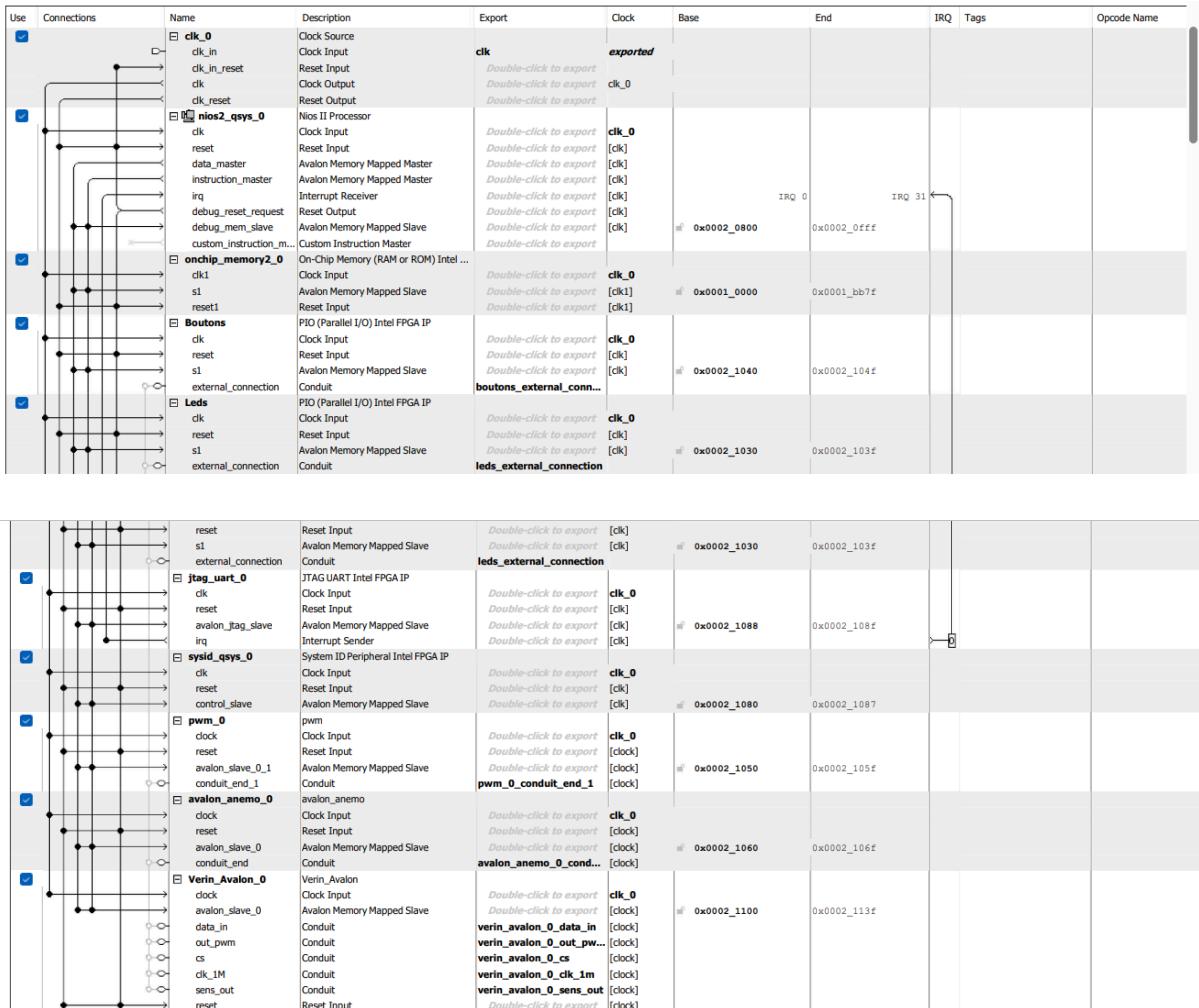


Figure 12 – Configuration SOPC pour vérin

Il est donc nécessaire d'intégrer le SOPC dans un schéma bloc, d'assigner les ports d'entrée/sortie du système, puis de compiler le projet.

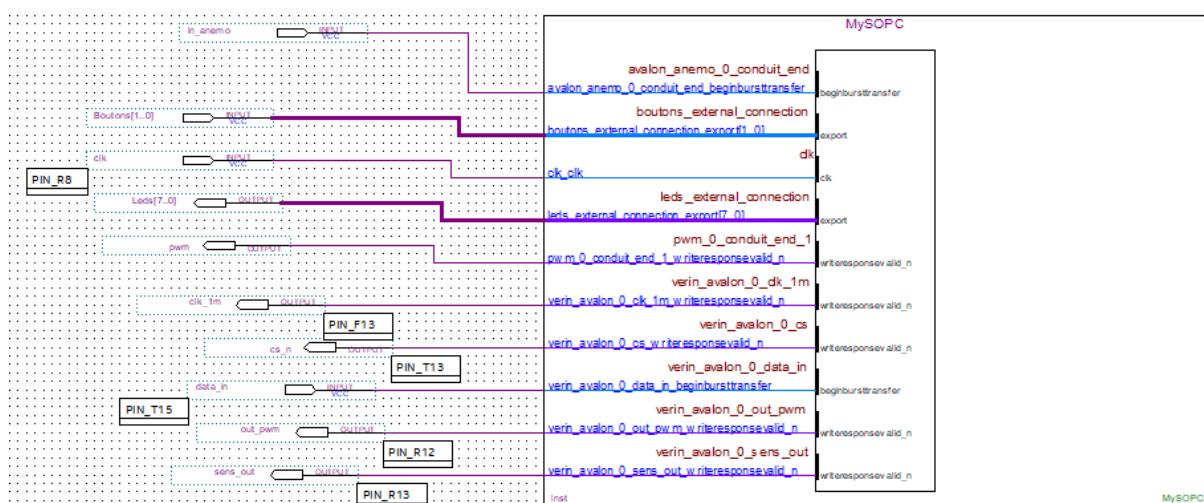


Figure 13 – Bloc SOPC vérin

L'interface verin\_avalon dispose de 6 registres tels que décrits ci-dessous :

```
#define butee_d (int *) (VERIN_AVALON_0_BASE+12)
#define butee_g (int *) (VERIN_AVALON_0_BASE+8)
#define freq (int *) VERIN_AVALON_0_BASE
#define duty (int *) (VERIN_AVALON_0_BASE+4)
#define config (int *) (VERIN_AVALON_0_BASE+16)
#define angle_barre (int *) (VERIN_AVALON_0_BASE+20)
```

**Freq** : Détermine la fréquence du signal PWM du moteur. Par exemple, si *freq* = 2000, alors la fréquence du PWM est calculée comme *clk*/2000, soit 25 kHz avec *clk* = 50 MHz.

**Duty** : Définit le rapport cyclique du signal PWM.

**Butee\_g** et **butee\_d** : Réglables entre 0 et 4095, elles fixent les limites que l'angle de la barre (*angle\_barre*) ne doit pas dépasser. Si ces limites sont franchies, le moteur est automatiquement désactivé. Par défaut, ces valeurs sont initialisées à 0 et doivent être configurées au démarrage du système.

**Registre Config:** registre de configuration du circuit.

**Angle\_barre:** Représente la valeur de l'angle de la barre, codée sur 12 bits, obtenue suite à la conversion analogique-numérique (CAN).

Une fois le projet configuré, nous avons développé le code en C pour piloter et tester l'interface Avalon\_anemo. Ce code permet d'interagir avec les registres *config* et *code*, de configurer le circuit, et d'extraire les données de l'anémomètre.

## IV. Bonus: gestion de boutons poussoirs

Dans cette section, nous avons mis en place une gestion des boutons poussoirs à l'aide d'une machine à états. Chaque bouton déclenche une action distincte, signalée par l'activation des LEDs associées :

- Un appui sur le bouton **Babord** allume la LED correspondante.
- Un appui sur le bouton **Tribord** active la LED Tribord.
- Le bouton **STBY** permet de passer du mode manuel au mode auto après un appui long.

### 1. Analyse fonctionnelle

Le système repose sur une **machine à états** pour gérer les interactions utilisateur via trois boutons poussoirs (**BP\_Babord**, **BP\_Tribord**, **BP\_STBY**) et générer des sorties sous forme de LEDs et d'un signal de commande **codeFonction**.

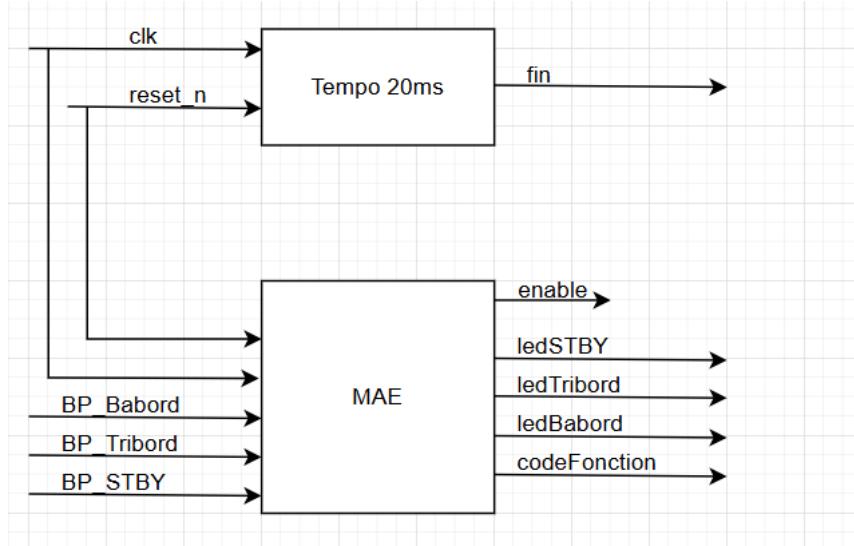


Figure 14 – Schéma bloc pour boutons poussoirs

#### Entrées principales :

- **clk** : Horloge 50 MHz pour la synchronisation du système.
- **reset\_n** : Réinitialisation active bas, ramenant la machine à l'état initial (**IDLE**).
- **BP\_Babord**, **BP\_Tribord**, **BP\_STBY** : Boutons poussoirs pour déclencher des actions spécifiques.

#### Blocs fonctionnels :

- **Génération de temporisation (20 ms)** :  
Ce bloc utilise l'horloge pour générer un signal **fin**, permettant de temporiser les transitions entre les états, pour éviter les rebonds.
- **Machine à états (MAE)** :  
La machine à états gère les différentes actions en fonction des boutons pressés :
  - **BP\_Babord** → Activation de **ledBabord**.
  - **BP\_Tribord** → Activation de **ledTribord**.
  - **BP\_Babord** et **BP\_Tribord** appuyés simultanément → Passage en **MODE\_AUTO**.

#### Sorties principales :

- **ledSTBY** : Active lorsque le système est prêt ou en mode automatique.
- **ledBabord** et **ledTribord** : S'allument selon l'action associée à chaque bouton.
- **codeFonction** : Signal 4 bits indiquant l'état en cours ou l'action réalisée.

## 2. Machine à états des boutons poussoirs

La machine à états contrôle les transitions entre les états comme suit:

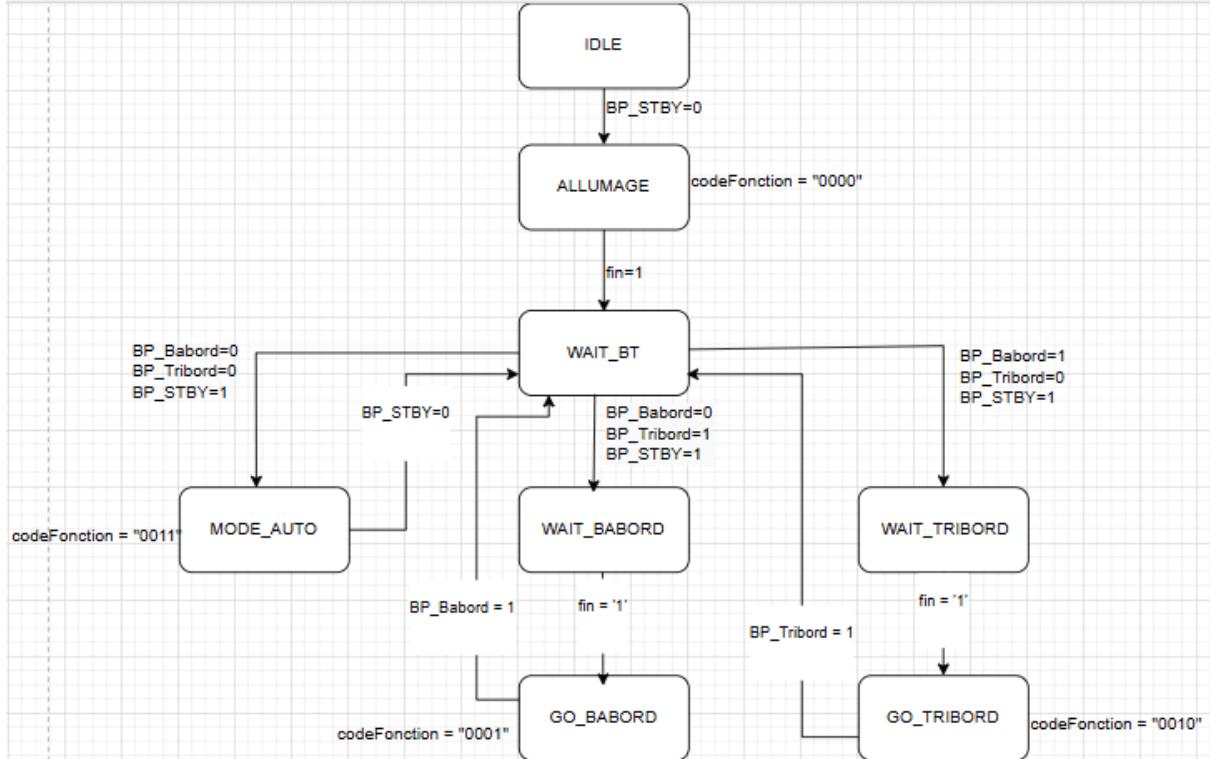


Figure 15 – Machine à état boutons poussoirs

- **IDLE (Repos) :**
  - État initial sans action.
  - LEDs éteintes, **codeFonction = "0000"**.
  - Appui sur **BP\_STBY** → Transition vers **ALLUMAGE**.
- **ALLUMAGE :**
  - Activation de **ledSTBY** pour indiquer que le système est prêt.
  - Temporisation via **fin**, puis passage à **WAIT\_BT**.
- **WAIT\_BT (Attente d'action) :**
  - **BP\_Babord** appuyé → Transition vers **WAIT\_BABORD**.
  - **BP\_Tribord** appuyé → Transition vers **WAIT\_TRIBORD**.
  - **BP\_Babord + BP\_Tribord** appuyés simultanément → Transition vers **MODE\_AUTO**.
- **WAIT\_BABORD / WAIT\_TRIBORD :**
  - Temporisation via **fin** pour stabiliser l'action.
  - Passage respectivement vers **GO\_BABORD** ou **GO\_TRIBORD**.
- **GO\_BABORD / GO\_TRIBORD :**
  - Activation des LEDs associées (**ledBabord** ou **ledTribord**).
  - **codeFonction** mis à jour (**0001** ou **0010**).
  - Retour à **WAIT\_BT** lorsque le bouton est relâché.
- **MODE\_AUTO :**
  - Toutes les LEDs activées (**ledSTBY**, **ledBabord**, **ledTribord**).
  - **codeFonction = "0011"**.

- Appui sur **BP\_STBY** → Retour à **WAIT\_BT**.

### 3. Simulation

La simulation valide le fonctionnement de la machine à états. À l'état initial (**IDLE**), toutes les LEDs sont éteintes et **codeFonction** est “**0000**”. l'image ci-dessous montre le fonctionnement général de la fonction.



Figure 16 – Résultat de simulation de la fonction boutons poussoirs

Un appui sur **BP\_Tribord** (ou **BP\_Babord**) seul active **ledTribord** (ou **ledBabord**) et **codeFonction** = “**0010**”(ou “**0001**”). Lors du relâchement, le système retourne à **WAIT\_BT** avec les LEDs éteintes et **codeFonction** = “**0000**”.

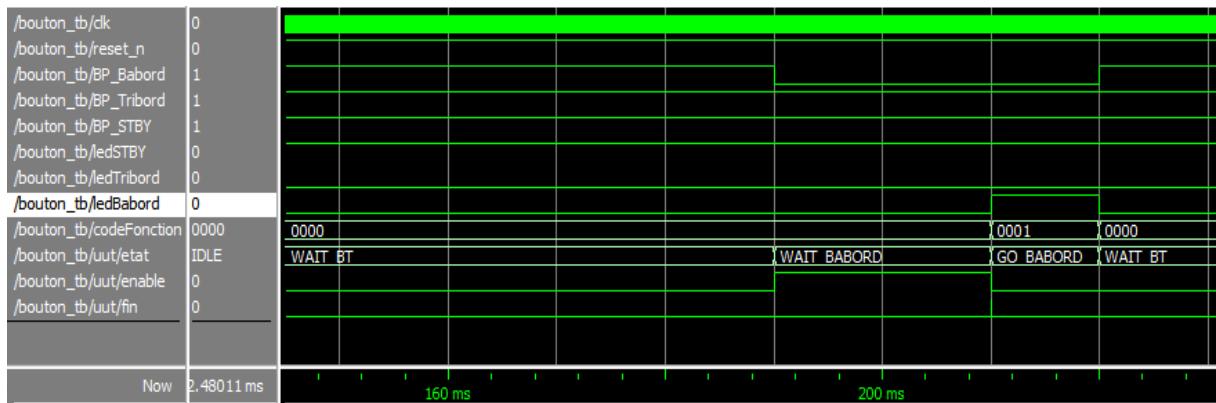


Figure 17 – Résultat de simulation mode manuel “babord”

Lors de l'appui simultané sur **BP\_Babord** et **BP\_Tribord**, la transition vers **MODE\_AUTO** est observée avec **codeFonction** = “**0011**” et toutes les LEDs activées.

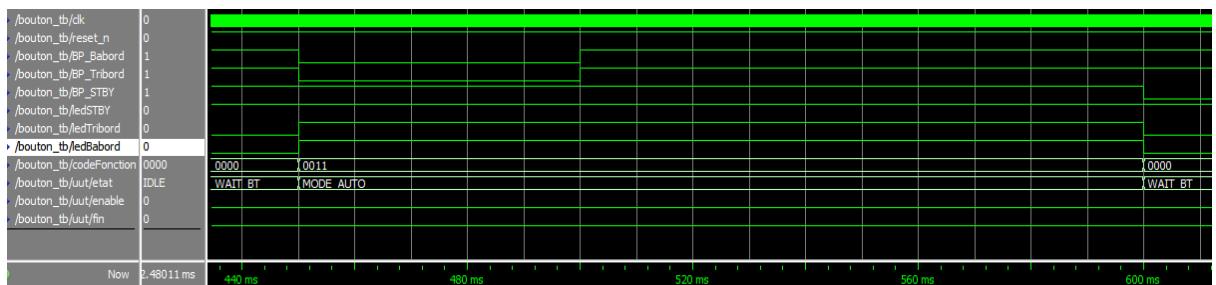


Figure 18 – Résultat de simulation mode auto

## V. CONCLUSION

Le projet *Pilote de Barre Franche* nous a permis d'acquérir des compétences techniques précieuses dans le domaine des systèmes embarqués. Grâce à ce projet, nous avons appris à maîtriser plusieurs outils et concepts essentiels tels que :

- **Le langage VHDL** : pour la conception et la description des fonctionnalités numériques sur une carte FPGA.
- **La carte FPGA** : qui nous a permis d'implémenter concrètement les fonctions développées.
- **Platform Designer** : un outil clé pour intégrer et configurer les différents composants matériels du système.
- **Quartus** : pour la synthèse, la compilation et le déploiement des designs VHDL sur la carte FPGA.
- **Eclipse** : pour l'écriture et l'implémentation du code en langage C, nous permettant d'interagir avec les fonctionnalités matérielles.

Sur le plan pratique, nous avons réussi à implémenter deux types de fonctions :

- Une **fonction simple d'anémomètre**, qui nous a permis de mieux comprendre les bases de la mesure et du traitement de signaux.
- Une **fonction complexe de gestion de vérin**, intégrant une logique plus avancée pour contrôler le mouvement et l'état du vérin.

En conclusion, ce projet nous a offert une vision complète du développement matériel et logiciel pour un système embarqué. Il nous a permis d'acquérir une méthodologie rigoureuse, de renforcer nos compétences techniques, et d'approfondir notre compréhension de l'architecture FPGA et de la conception numérique. Cette expérience enrichissante constitue une base solide pour aborder des projets futurs plus complexes.