RAPPORT DU BE

Gestion d'un pilote de barre franche

Année universitaire 2020 – 2021

M2 - SME

Lamoussa SANOGO : sanogolamouss30@yahoo.fr

Victor SALAZAR : xavicoel@gmail.com

Plan de travail

I. Objectif et présentation du projet

L'objectif de ce projet est l'apprentissage du langage VHDL et l'implémentation de système en ce dernier. Pour cela, nous devons implémenter les fonctions d'une pilote de barre de franche. Ces fonctions récupèrent la position de la barre franche et des informations de l'environnement à partir d'un certains nombres de capteurs et asservissent le positon de la barre franche en fonction des valeurs des données reçues et le mode de pilotage choisi.

Nous allons créer un microcontrôleur sur le FPGA pour y ajouter toutes nos fonctions comme n'importe quel autre interface.

II. Fonctions implémentées

A. Qu'est-ce qu'un SOPC ?

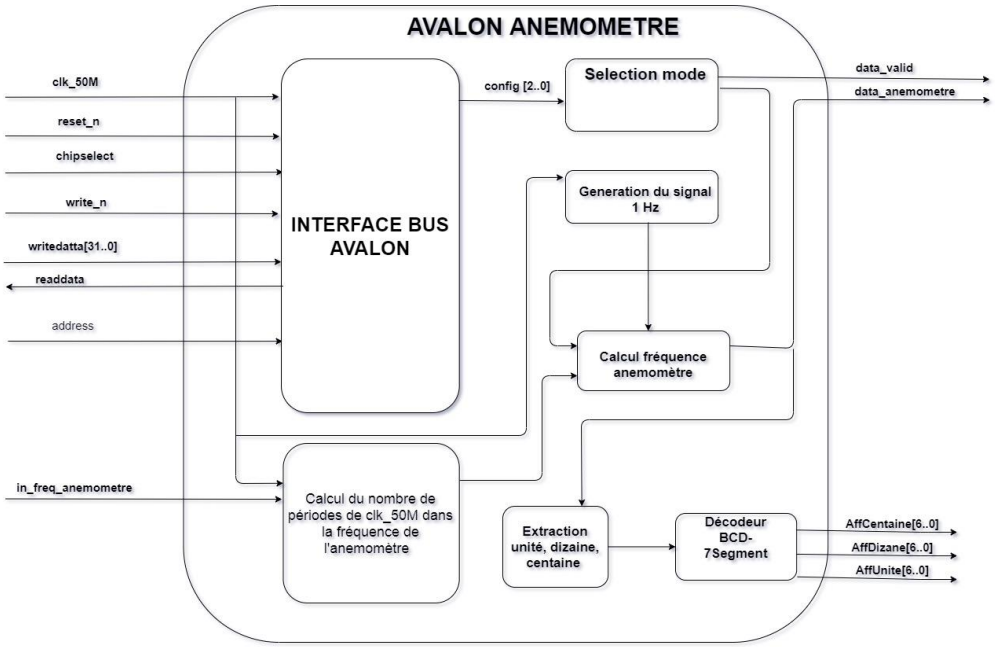
Le SOPC est un microcontrôleur implémenté sur le FPGA. Ce microcontrôleur utilise le processeur NIOS, des entrées sorties, des interfaces de communications. La communication avec un programme C se fait à travers le bus Avalon et est implémentée avec un processus de lecture et un processus d'écriture des registres pour chaque interface du SOPC.

Process d'écriture : permet d'écrire dans les registres des interfaces du SOPC depuis le programme C, idéale pour configurer les interfaces du SOPC. Exemple : écriture dans le registre (variable) config pour configurer l'interface de l'anémomètre.

Process de lecture : permet de lire des registres des interfaces du SOPC depuis le programme C. Par exemple, lire et afficher la vitesse du vent dans la console du NIOS.

A chaque fonction, nous ajouterons donc le processus de lecture et le processus d'écriture pour faire l'interfaçage avec le bus Avalon.

B. Anémomètre



La configuration de l'anémomètre, fait à travers l'interface Avalon, consiste à écrire une valeur dans le registre config. Cette valeur détermine le mode de mesure de la vitesse du vent comme suit :

Config(2) : start_stop

Config(1) : changement de mode

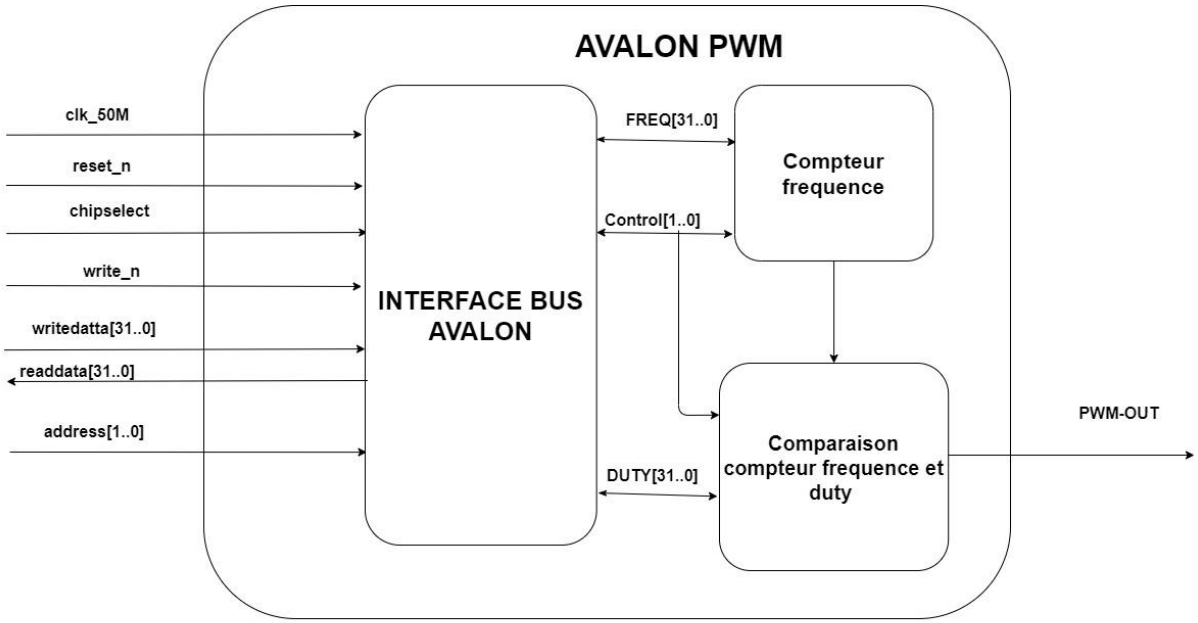
Mode monocoup : config(1) = '0'. Dans ce cas, on utilise l'interrupteur start_stop pour faire la mesure de fréquence. La sortie datavalid, affectée à une LED, passe à 1 à chaque mesure.

Mode continu (nommé multicoup dans notre code) : la mesure de fréquence est faite toutes les secondes. La sortie datavalid, affectée à une LED, passe à 1 à chaque mesure.

Si la fréquence d'entrée est supérieure à 250 Hz, la sortie data_anemomètre prend la valeur 0.

C. Le PWM

En réalité, cette fonction ne fait pas partir du système de gestion de la barre franche, elle est juste utilisée pour valider le bon fonctionnement du programme de l'anémomètre.



Ce programme utilise trois registres accessibles en lecture et écriture depuis le NIOS :

FREQ : pour écrire ou lire la fréquence du signal PWM

DUTY : pour écrire ou lire le rapport cyclique du signal PWM

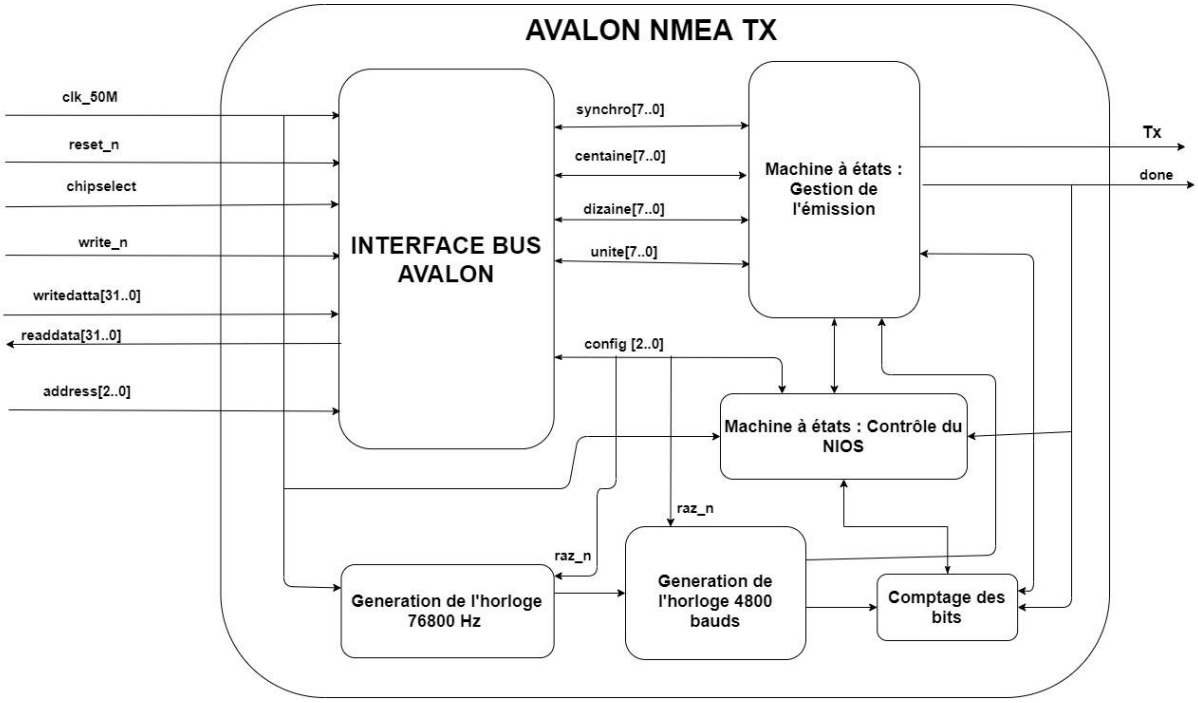
Control : ce registre permet d'activer et de désactiver la sortie PWM. La sortie PWM est activé avec la valeur 0x03 dans ce registre.

Pour valider le bon fonctionnement du programme anémomètre, on connecte en interne ou en externe la sortie PWM-OUT à l'entrée in_freq_anemometre de l'anémomètre, on devrait alors mesurer la même fréquence à la sortie data_anemometre.

Dans la pratique, nous obtenons une valeur à ± 1 près, à une entrée 200Hz sur in_freq_anemometre notre anémomètre mesure 199Hz, cette marge de ± 1 peut provenir du programme générant le PWM ou le programme de l'anémomètre.

D. NMEA_TX

Le protocole de communication NMEA fait partir des plus difficiles à implémenter dans ce projet. Comme tout protocole de communication, la précision temporelle est indispensable et les marges d'erreurs tolérées sur ce point sont très réduites.



Le schéma ci-dessous représente un découpage en blocs de la fonction transmission NMEA :

Raz_n = config(0) : permet de démarrer ou d'arrêter la transmission

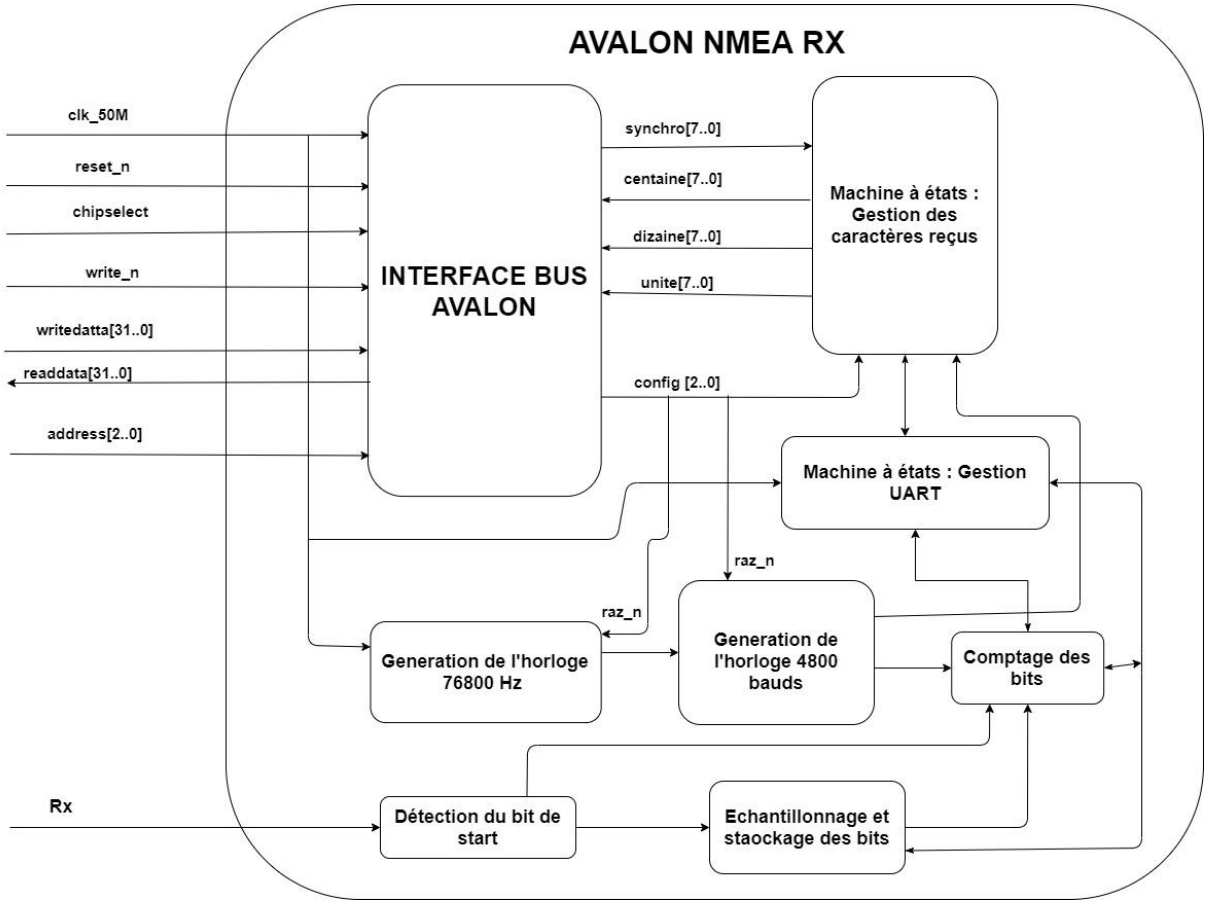
Start_stop = config(1) : permet de contrôler la fréquence de transmission. Par exemple, pour transmettre une fois par seconde, nous mettons config(1) à 1 pendant une seconde, puis à 0 pendant une seconde pour ensuite le remettre à 1, ainsi de suite. Ceci est fait depuis le code C dans le NIOS II Software.

Pour obtenir la vitesse de transmission 4800 bauds à partir de l'horloge 50MHz, nous passons par deux (02) blocs pour plus de précision. En effet, lorsqu'on divise 50MHz par 76800Hz, on obtient 651.041, le taux d'erreur est très faible. Ensuite en divisant 76800Hz par 4800Hz, on obtient exactement 16, donc théoriquement pas d'erreur.

Nous avons utilisé la bibliothèque « softwareSerial.h » de l'Arduino pour créer une interface série logiciel et visualiser ainsi dans le terminal de l'Arduino les données sortants sur la sortie Tx (appélée txd dans notre code), et tout fonctionne correctement.

E. NMEA_RX

Nous avons eu beaucoup de difficultés avec cette fonction. Cela a été encore plus difficile par le fait que nous n'avions pas de moyens simples pour observer ce que reçoit le programme. Nous y sommes finalement arrivés avec l'aide du pico-scope.



Le schéma ci-dessous représente un découpage en blocs de la fonction réception NMEA :

Raz_n = config(0) : permet de démarrer ou d'arrêter la réception.

Ack = config(1) : l'accusé de réception permet de signaler la fin d'une trame NMEA et met le programme de réception en attente d'une nouvelle trame.

Pour obtenir la vitesse de transmission 4800 bauds à partir de l'horloge 50MHz, nous faisons exactement la même chose que dans la transmission.

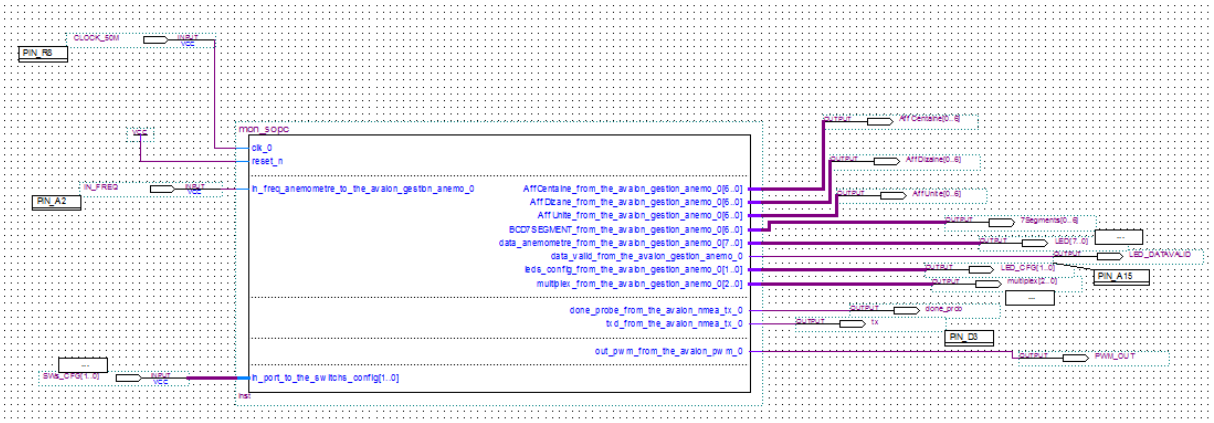
Et pour terminer, nous avons le choix d'afficher les données reçues soit sur les afficheurs 7 segments soit dans le terminal du NIOS II Software, cette dernière option est utilisée dans la démonstration finale du projet puisque nous affichons déjà la vitesse du vent sur les afficheurs 7 segments.

F. LE SOPC

Toutes les fonctions, que nous venons de voir, ont été validées individuellement sans l'implémentation de l'interface Avalon. Ensuite, nous avons ajouté à chaque fonction un process de lecture et un process d'écriture qui gèreront la communication de données entre le NIOS II Software et les registres du SOPC. Et pour terminer, nous avons créé un SOPC avec Quartus 11 et y intégré toutes nos fonctions les unes après les autres.

On peut voir sur la première image ci-dessous, les composants de notre SOPC.

Use	Conn...	Name	Description	Clock	Base	End	IRQ	Tags
☑		cpu_0	Nios II Processor	[clk]				
		instruction_master	Avalon Memory Mapped Master	clk_0				
		data_master	Avalon Memory Mapped Master	[clk]			IRQ 0	IRQ 31
		jtag_debug_module	Avalon Memory Mapped Slave	[clk]	0x00010800	0x00010fff		
☑		sram	On-Chip Memory (RAM or ROM)	[clk1]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00008000	0x0000cfff		
☑		jtag_uart_0	JTAG UART	[clk]				
		avalon_jtag_slave	Avalon Memory Mapped Slave	clk_0	0x00011050	0x00011057	16	
☑		sysid	System ID Peripheral	[clk]				
		control_slave	Avalon Memory Mapped Slave	clk_0	0x00011058	0x0001105f		
☑		avalon_pwm_0	avalon_pwm	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011020	0x0001102f		
☑		switchs_config	PIO (Parallel I/O)	[clk]				
		s1	Avalon Memory Mapped Slave	clk_0	0x00011030	0x0001103f		
☑		avalon_gestion_anemo_0	avalon_gestion_anemo	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011040	0x0001104f		
☑		avalon_nmea_tx_0	avalon_nmea_tx	[clock]				
		avalon_slave_0	Avalon Memory Mapped Slave	clk_0	0x00011000	0x0001101f		



Ensuite, nous utilisons NIOS II Software pour programmer le SOPC, le code C est montré dans les captures suivantes :

```c
#include "sys/alt_stdio.h"
#include "system.h"
#include "stdio.h"
#include "stdlib.h"
#include "unistd.h"


#define   config_nmeatx (unsigned int*) AVALON_NMEA_TX_0_BASE
#define   synchro_nmeatx (unsigned int*) (AVALON_NMEA_TX_0_BASE + 4)
#define   centaine_nmeatx (unsigned int*) (AVALON_NMEA_TX_0_BASE + 8)
#define   dizaine_nmeatx (unsigned int*) (AVALON_NMEA_TX_0_BASE + 12)
#define   unite_nmeatx (unsigned int*) (AVALON_NMEA_TX_0_BASE + 16)

#define   freq  (unsigned int*)    AVALON_PWM_0_BASE
#define   duty  (unsigned int*)    (AVALON_PWM_0_BASE+4)
#define   ctrl  (unsigned int*)    (AVALON_PWM_0_BASE+8)

#define   cfg          (unsigned int*)    AVALON_GESTION_ANEMO_0_BASE
#define   data_anemo   (unsigned int*)    (AVALON_GESTION_ANEMO_0_BASE+4)
#define    sws_cfg  (unsigned int*)  SWITCHS_CONFIG_BASE

int main()
{
  alt_putstr("Hello from Nios II!\n");
```

Tx

PWM

ANEMO

```c
//  PWM
  *freq = 0x3D090;  // 200 Hz
  *duty = 0x1E848;  // 50 %
  *ctrl = 0x03;

  //TRANSIMISION TX
  *synchro_nmeatx =0xAA;
  *centaine_nmeatx =0xBB;
  *dizaine_nmeatx=0xCC;
  *unite_nmeatx=0xDD;

  /* Event loop never exits. */
  while (1){
      // Contrôle de la transmission TX
      *config_nmeatx =0x07;
      usleep(1000000);
      *config_nmeatx =0x05;
      usleep(1000000);

    //Anemomètre
     printf("Vitesse du vent %d \n", *data_anemo);
     switch(*sws_cfg)
     {
     //LECTURE BOUTONS
```

```
    //LECTURE BOUTONS
    case 0x81 : *cfg=0x03; break;  // Configuration de l'anemomètre
    case 0x82 : *cfg=0x05; break;
    default   : *cfg=0x01;
    }

  }
  return 0;
}

//fin
```

Comme on peut le voir, le NIOS II Software est principalement utilisé pour configurer des interfaces du SOPC en écrivant dans des registres et/ou pour lire les contenus des registres.

Une quantité de mémoire est allouée à chaque interface du SOPC. La taille de cette mémoire par interface dépend du nombre de signaux de l'interface transitant entre cette dernière et le NIOS Software à travers le bus Avalon, autrement dit, le nombre de signaux présents dans ces processus lecture et écriture. Chaque signal est stocké sur quatre (04) octets, c'est ainsi que le premier signal, affecté à l'adresse 0 se trouve au début et occupe 4 cases mémoires successives, le deuxième signal à l'adresse 1 commence alors à l'adresse de début + 4, et ainsi de suite.

III. Conclusion

Ce projet nous a permis d'apprendre énormément de choses sur le langage VHDL comme les règles de syntaxes, les comportements des variables et des signaux, le fonctionnement des process, des components, etc.… Nous avons également appris la création de circuits, graphiquement, en utilisant des composants graphiques sans passer par du code VHDL, nous avons appris aussi les différentes structures des fichiers d'un projet VHDL.

Et enfin, nous avons découvert le SOPC qui est un outil très puissant et flexible qui simplifie la création de circuit complexe.

A présent, nous en savons beaucoup sur les différents outils Intel de développement sur FPGA, les différentes versions de Quartus et les simulateurs intégrés et indépendants selon les versions de Quartus, le SOPC Builder, Qsys, le NIOS II Software, etc...

Nous n'avons pas pu implémenter toutes les fonctions par manque de temps et/ou de matériels mais ce projet a été pour nous une expérience TRES enrichissante.

Lien vers le code : https://github.com/xaviSalazar/SALAZAR_SANOGO_BINOME7

Lien vers la vidéo : https://drive.google.com/file/d/1rIXbGJaoVXweyif6qLyptBfT2bsxViHT/