18/12/2019

Pierrick Müller et Tommy Gerardi

HEIG-VD

CSN Laboratoire 4

Acquisition de position

Table des matières

[1 Introduction 2](#_Toc27509916)

[2 Système à réaliser 2](#_Toc27509917)

[2.1 Description du système 2](#_Toc27509918)

[2.2 Entrées/Sorties du système 2](#_Toc27509919)

[2.3 Description du fonctionnement 3](#_Toc27509920)

[3 Décomposition 3](#_Toc27509921)

[3.1 Description générale 3](#_Toc27509922)

[3.2 Bloc 1 : Machine d’état du système 4](#_Toc27509923)

[3.3 Bloc 2 : Compteur 4](#_Toc27509924)

[3.3.1 Schéma 4](#_Toc27509925)

[3.3.2 Table des fonctions synchrone 4](#_Toc27509926)

[3.4 Bloc 3 : Détection des erreurs 4](#_Toc27509927)

[4 VHDL 4](#_Toc27509928)

[5 Synthèse 5](#_Toc27509929)

[5.1 Système complet 5](#_Toc27509930)

[5.2 Compteur 5](#_Toc27509931)

[5.3 Détection des erreurs 6](#_Toc27509932)

[5.4 Machine d’état du système 6](#_Toc27509933)

[6 Vérification du fonctionnement et tests réalisés 6](#_Toc27509934)

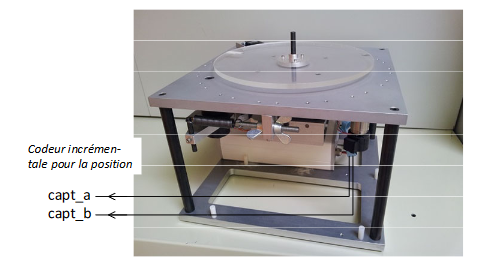
[7 Conclusion 6](#_Toc27509935)

[8 Annexes 6](#_Toc27509936)

# Introduction

Ce laboratoire a pour but de mettre en place un système permettant l’acquisition de la position d’un disque tournant. La plateforme Servo-USB sera utilisée, elle comprend un codeur incrémental entraîné par le disque qui permet de mesurer la position de ce dernier.

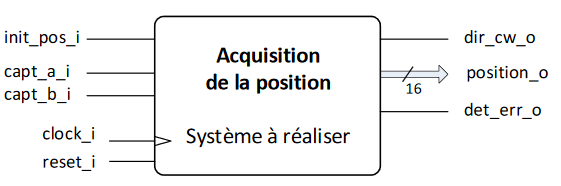
Voici la plateforme en question :



# Système à réaliser

## Description du système

Le système reçoit des informations du codeur incrémental présent sur la plateforme Servo-USB. Il reçoit aussi un signal d’initialisation de la position. Ce schéma montre le système qu’on va réaliser :



## Entrées/Sorties du système

|  |  |  |  |
| --- | --- | --- | --- |
| Signal | Direction | Taille | Description |
| Init\_pos\_i | Entrée | 1 | Signal d’initialisation de la position |
| Capt\_a\_i | Entrée | 1 | Signal A venant du codeur incrémental |
| Capt\_b\_i | Entrée | 1 | Signal B venant du codeur incrémental |
| Dir\_cw\_o | Sortie | 1 | Signal indiquant le sens de rotation du disque |
| Position\_o | Sortie | 16 | Sortie indiquant la position du disque |
| Det\_err\_o | Sortie | 1 | Sortie indiquant qu’une erreur s’est produite sur le codeur (double changement) |
| Clock\_i | Sortie | 1 | Horloge du système synchrone, 1Mhz |
| Reset\_o | Sortie | 1 | Signal de remise à zéro asynchrone |

## Description du fonctionnement

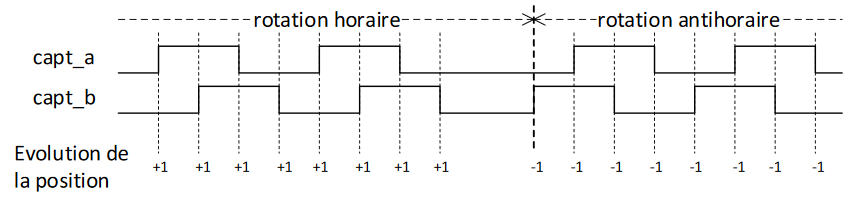
Lorsque le signal init\_pos\_i est actif, la position est initialisée à 0 et le signal det\_err\_o est désactivé.

Si le signal init\_pos\_i n’est pas actif, la position doit être incrémentée ou décrémentée à chaque changement d’état d’un des capteurs du codeur comme suit :

* Le codeur tourne dans le sens horaire -> la position est incrémentée à chaque changement du capteur A ou B.
* Le codeur tourne dans le sens anti-horaire -> la position est décrémentée à chaque changement du capteur A ou B.

Dans le cas où les capteurs A et B changent simultanément (même période d’horloge), le signal det\_err\_o doit être activé. Ce cas de figure rend impossible la détermination du sens dans lequel le codeur a tourné.

Voici un exemple de fonctionnement :



# Décomposition

Le système à été décomposé en 3 blocs principaux comme suit :

INSÉRER DESSIN SWAG QUI RPZ NOTRE TRAVAIL

## Description générale

Le système en lui-même fonctionne de manière assez simple, une machine d’état s’occupe de gérer les signaux nécessaires à un bloc compteur 16 bits qui garde en mémoire la position et à un bloc qui contient un flip flop de type « Inserer type flip-flop » et qui permet de gérer le signal de sortie « det\_err\_o ».

La sortie « dir\_cw\_o » est un signal de sortie de la machine d’état et la sortie « position\_o » correspond à la position stockée et gérée par le bloc compteur.

A la fin du laboratoire, nous avons de plus dû ajouter une synchronisation des entrées « capt\_a\_i » et « capt\_b\_i » afin d’éviter des erreurs provenant d’entrées non synchrones. Cette synchronisation s’effectue en faisant passer ces signaux lors de leurs arrivées dans le système par un flip-flop simple qui permet donc la synchronisation de ces entrées. Ces flip-flop ne figurent pas dans le schéma ci-dessus, mais ont été ajoutés lors de la création du code VHDL, qui vous est fourni en annexe.

## Bloc 1 : Machine d’état du système

Voici le schéma que nous avons créer pour notre machine d’état :

INSÉRER MSS SWAG

## Bloc 2 : Compteur

### Schéma

Voici le schéma représentant notre compteur 16 bits

INSÉRER COMPTEUR SWAG

### Table des fonctions synchrone

Voici la table des fonctions synchrone de notre compteur sur N bits de 1 à 32:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Init\_pos | Compt\_en | Sens\_i | Reg\_pres | Reg\_futur |
| = 1 | - | - | - | 0 |
| = 0 | = 0 | - | - | Reg\_pres |
| = 0 | = 1 | = 1 | - | Reg\_pres + 1 |
| = 0 | = 1 | = 0 | - | Reg\_pres - 1 |

Le signal d’initialisation est le plus prioritaire sur les autres car tant qu’il est actif, le compteur doit être à 0. Ensuite vient l’enable du compteur, qui correspond au maintient de la valeur pour notre compteur. Finalement, le sens définit si le compteur doit incrémenter ou décrémenter la valeur qu’il contient en fonction du sens selon lequel la roue tourne. La valeur du registre présent n’a pas d’importance sur la définition du registre futur, mais il est utilisé afin de calculer la valeur du registre futur (+1, -1, maintient). Ainsi, il nous semblait pertinant de faire apparaitre cette colonne « Reg\_pres » dans notre table des fonctions synchrone.

## Bloc 3 : Détection des erreurs

Ce bloc utilise aussi une table des fonctions synchrone, la voici :

|  |  |  |  |
| --- | --- | --- | --- |
| Init\_pos | err | Reg\_pres | Reg\_futur |
| = 1 | - | - | 0 |
| = 0 | = 1 | - | 1 |
| = 0 | = 0 | - | Reg\_pres |

Cette table est plus simple que la précédente. En effet, ce bloc s’occupe simplement de garder une bascule avec la valeur 1 si une erreur à été détectée à un moment donné par la machine d’état. La bascule restera à 1 tant que le signal d’initialisation n’aura pas été mis à 1, ce qui nous permettra d’avoir notre sortie det\_err\_o à 1 de manière constante si une erreur à été détectée, tant que le système n’a pas été réinitialisé.

# VHDL

Nous avons fait comme choix lors de la création du VHDL d’implémenter un compteur générique pouvant aller de 1 à 32 bits, afin de garder un système pouvant évoluer dans le temps, au cas ou la taille de la position (et donc le nombre de positions possibles) changeait.

Comme dans notre schéma général, nous disposons de trois fichiers VHDL représentant les trois blocs que nous avons décidé d’implémenter et d’un fichier supplémentaire représentant le système d’acquisition de position en lui-même.

Tous les codes ont étés commentés afin de simplifier leur compréhension et peuvent être retrouvés dans les annexes de ce document. Nous passerons donc ici sur les fonctionnements précis des codes et nous tenterons surtout de mettre en lumière les difficultés qui ont été rencontrées lors de l’implémentation de nos codes VHDL.

# Simulation et testbench

# Synthèse

Lors de la synthèse, nous avons pu observer plusieurs points intéressants dans les vues RTL et technologique des différents blocs. Voici ce que nous pouvons dire de ces résultats :

## Système complet

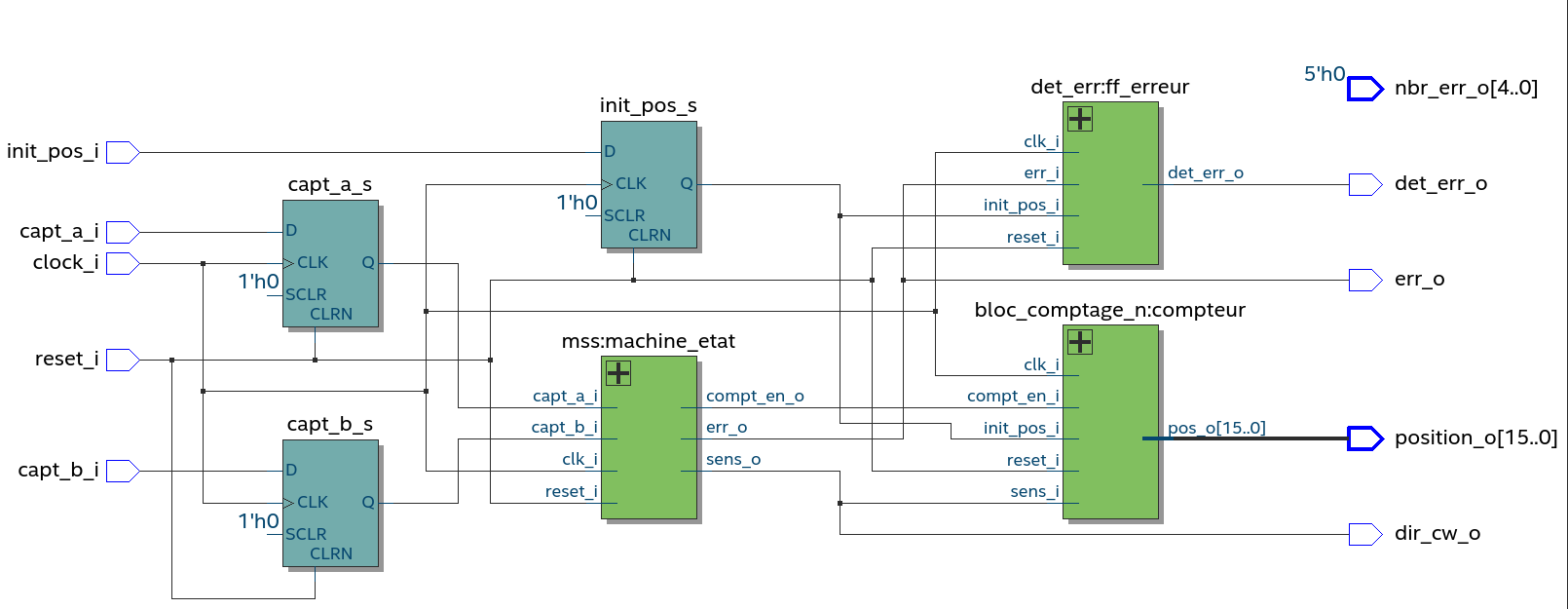


Figure 1 : Vue RTL Système Acquisition

La vue complète correspond à l’idée que nous avions de base de notre système avec la gestion des signaux interne par la machine d’état qui dit au compteur et au module de détection d’erreur ce qu’ils doivent faire. On peut voir l’ajout d’une synchronisation pour les entrées des capteurs et du signal d’initialisation, cela a été ajouté après les tests effectués qui démontrait que ces signaux devaient être synchrone, faute de quoi la machine d’état ne pouvait pas fonctionner, car se basant sur les deux signaux correspondant aux capteurs pour effectuer les changements d’état nécessaires. Pour ce qui est du signal d’initialisation, nous avons préféré le synchroniser aussi, étant donné que rien ne nous était dis sur le fait que notre système devait le traiter comme un signal asynchrone.

## Compteur

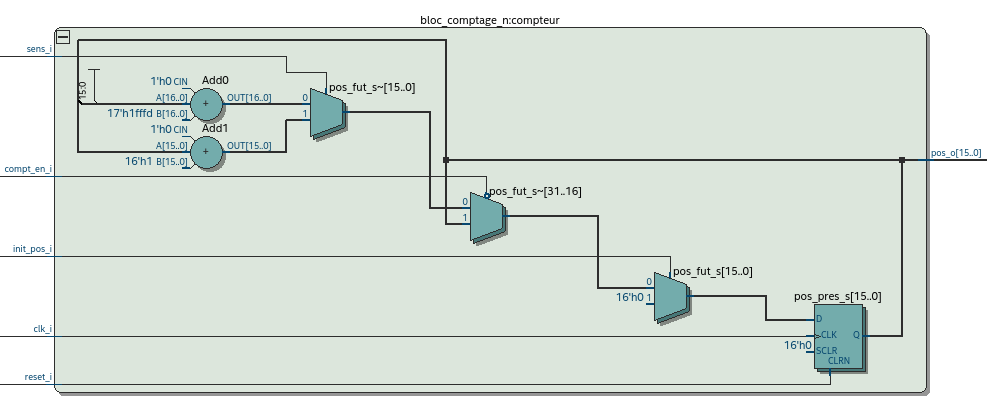


Figure 2 : Vue RTL Compteur

On peut voir que le compteur utilise trois multiplexeurs, un pour chacun des signaux d’entrée ayant un impact décrit dans la table des fonctions synchrone. En prenant dans l’ordre de proximité du registre final, on à d’abord la gestion du signal d’initialisation, qui aura donc la plus grande priorité dans la définition de la valeur du registre, avec soit 0, soit la valeur définit auparavant par les autres multiplexeurs. Ensuite, on à la gestion de l’enable du compteur, qui sert à gérer le maintient du registre dans son état précédent ou son évolution en fonction du sens. Finalement, un multiplexeur géré par le signal de sens permet soit de prendre le résultat de l’addition avec la valeur du registre présent, soit le résultat de la soustraction. Ainsi, la vue RTL correspond exactement à ce qui était attendu lors de la création du compteur.

## Détection des erreurs

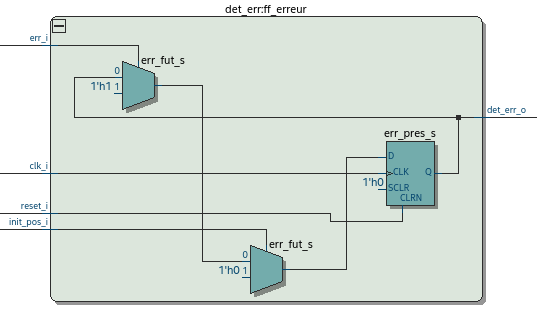


Figure 3 : Vue RTL det\_err

Ici, on peut voir aussi que la vue RTL correspond à ce qui était attendu lors de la création du bloc. En effet, un premier multiplexeur prioritaire correspond à la gestion du signal d’initialisation, qui permet d’initialiser la détection avec comme valeur 0. Ensuite, l’autre multiplexeur permet de gérer l’activation du signal d’erreur en permettant de mettre la valeur du registre à 1. Ainsi, dès qu’une erreur est détectée, le signal de sortie det\_err\_o restera à la valeur 1 tant que le signal d’initialisation n’aura pas été envoyé pour « reset » de manière synchrone le registre à 0.

## Machine d’état du système

Il n’y a pas grand-chose à tirer du résultat de la vue RTL de la machine d’état. Quartus à effectué les optimisations, mais le résultat n’est pas franchement compréhensible, et il n’est pas utile d’essayer de le comprendre dans notre cas, c’est justement le fait que Quartus effectue les optimisations à partir du code VHDL qui nous permet de ne pas avoir à nous soucier du résultat de la vue RTL dans ce cas-là.

A CONTROLLER SI ON VEUT QUAND MEME METTRE LA VUE LA.

# Vérification du fonctionnement et tests réalisés

Après avoir programmer le circuit Max-V de la carte, nous avons put testé directement le résultat à l’aide de la plateforme Servo-USB et de la console qui nous était

# Conclusion

# Annexes