Auteur : Spinelli Isaia et

Lankeu Ngassam Cédric

Prof : [Etienne](https://cyberlearn.hes-so.ch/user/view.php?id=104149&course=1) Messerli

Ing : Sébastien Masle

Date : 28.11.2019

Salle : A09

Classe : CSN

Acquisition de position

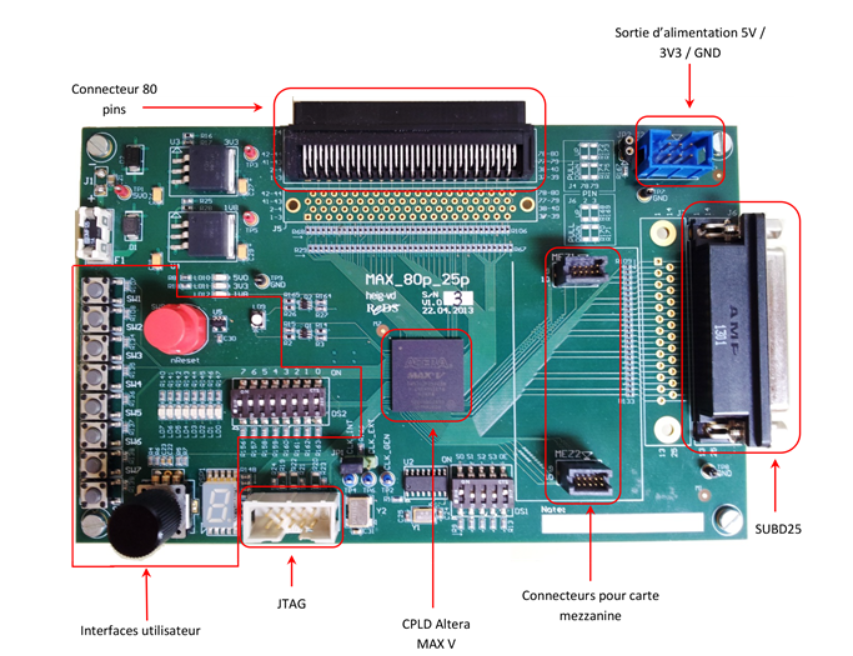
[CONCEPTION DE SYSTÈMES NUMÉRIQUES (CSN)](https://cyberlearn.hes-so.ch/course/view.php?id=14116#section-1) 

Table des matières

[Objectifs - 2 -](#_Toc27582018)

[Présentation du système - 2 -](#_Toc27582019)

[Spécification du système - 3 -](#_Toc27582020)

[Analyse du fonctionnement - 4 -](#_Toc27582021)

[Schéma bloc - 5 -](#_Toc27582022)

[Machine d’état - 6 -](#_Toc27582023)

[Compteur et décodeur d’état futur - 19 -](#_Toc27582024)

[Bascule RS - 23 -](#_Toc27582025)

[Regroupement des blocs - 24 -](#_Toc27582026)

[Synthèse ou quantité logique - 24 -](#_Toc27582027)

[Vérification du fonctionnement - 26 -](#_Toc27582028)

[Test du système - 28 -](#_Toc27582029)

[Conclusion - 29 -](#_Toc27582030)

[Difficultés rencontrées - 29 -](#_Toc27582031)

[Compétences acquises - 29 -](#_Toc27582032)

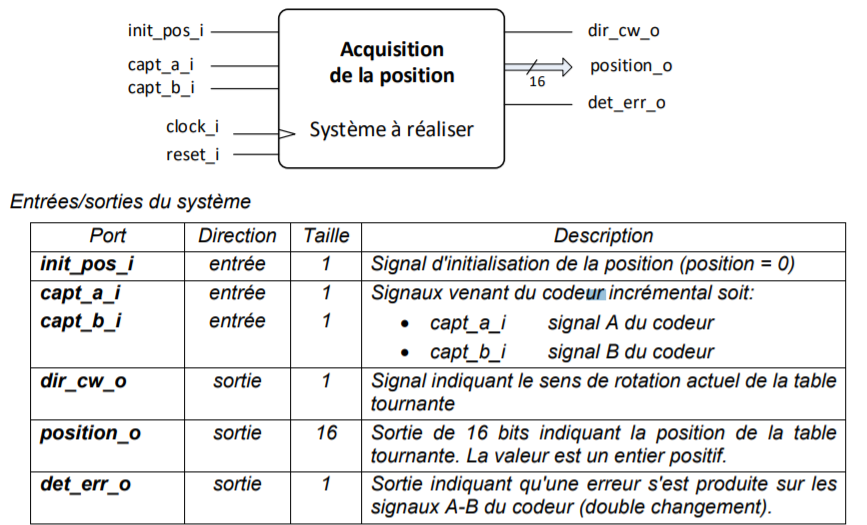
[Résultats obtenus - 29 -](#_Toc27582033)

## Objectifs

Nous souhaitons réaliser un système permettant de réaliser l’acquisition de la position d’un disque tournant. Nous allons utiliser la plateforme Servo-USB qui comprend un codeur incrémental entrainé par le disque qui nous permet de mesurer la position de celui-ci.

## Présentation du système

Le système reçoit les informations du codeur incrémental ainsi qu’un signal d’initialisation de la position. Voici le symbole du système à réaliser :





## Spécification du système

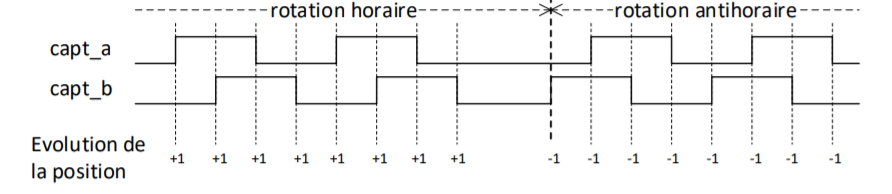
Le système reçoit aussi un signal d’initialisation permettant une remise à zéro de la position. Description du fonctionnement de la position :

* Lorsque le signal init\_pos\_i est actif, la position est initialisée à 0. D'autre part le signal d'erreur det\_err\_o est désactivé.
* Sinon: La position doit être incrémentée ou décrémentée à chaque changement d’état d’un des capteurs du codeur, soit :
  + Si le codeur tourne dans le sens horaire, la position est incrémentée à chaque changement d'état d’un des capteurs (capt\_a\_i ou capt\_b\_i)
  + Si le codeur tourne dans le sens anti-horaire, la position est décrémentée à chaque changement d'état d’un des capteurs (capt\_a\_i ou capt\_b\_i)

Lors d’un changement simultané des signaux capt\_a\_i et capt\_b\_i durant la même période d’horloge, le signal det\_err\_o doit être activé.

* En effet dans ce cas, il est impossible de déterminer dans quel sens le codeur a tourné !

Voici un exemple de fonctionnement :

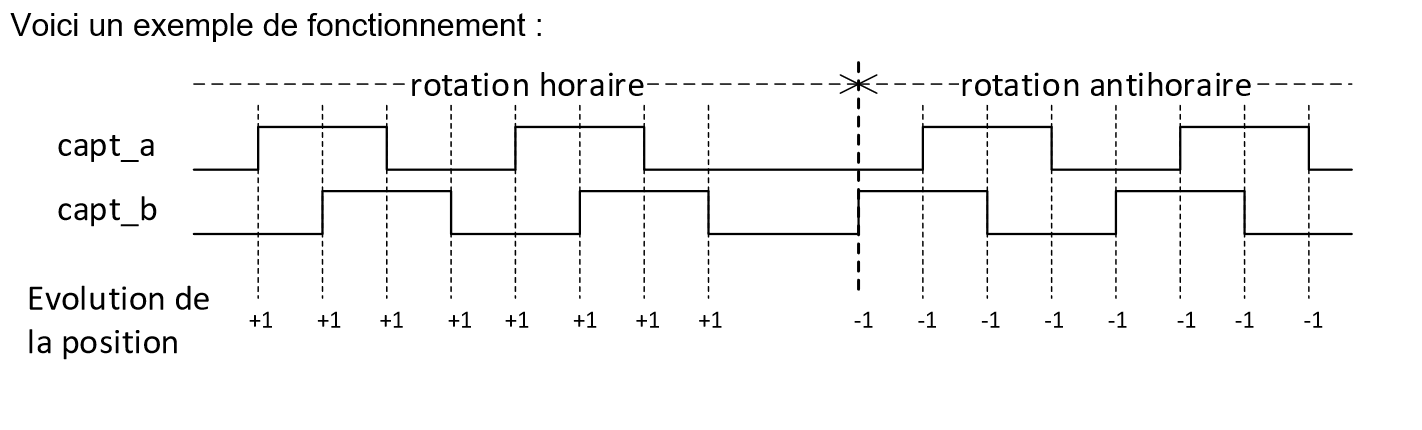


Spécifications de la table tournante :

Le codeur de la table est de type HEDS-5310. Celui-ci a une résolution de 500 traits. Cela permet d'obtenir 2000 incréments par tour du codeur en comptabilisant tous les changements des signaux A et B (multiplication par 4). Un tour de la table tournante correspond à 5 tours du codeur, donc à un total de 10'000 incréments.

## Analyse du fonctionnement

Nous avons commencé par analyser le fonctionnement du système d’acquisition de position et définir une décomposition.



0

1

1

1

1

0

0

1

1

0

0

1

1

0

0

1

1

1

1

0

1

0

0

0

1

1

1

0

0

0

La combinaison des entrées Capt\_a et Capt\_b permet de déterminer le sens de rotation. On peut toutefois constater que lorsque le signal de capt\_a arrive à ’1’ avant celui du capt\_b, on effectue une rotation horaire, au cas contraire, on a une rotation antihoraire. On peut aussi voir que la rotation horaire (dans le cadre vert) a une séquence bien précise et pareil que celle encadrée en rouge(antihoraire).

Nous avons rapidement compris qu’il fallait une machine d’état pour gérer les deux différentes séquences des capteurs ainsi qu’un compteur pour 16 bits afin de suivre la position du disque tournant.

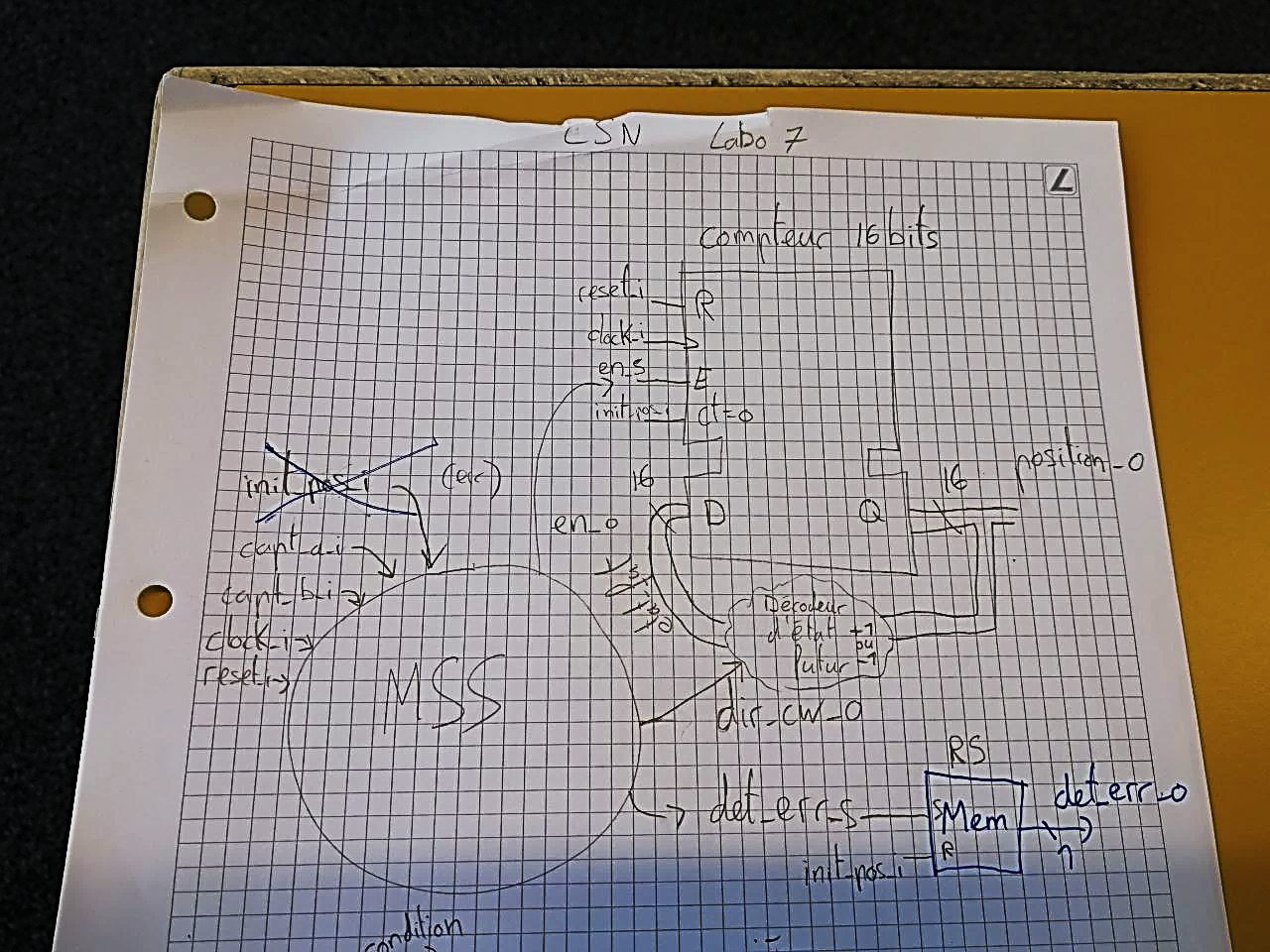
Plusieurs sorties sont demandées pour ce système comme détaillé précédemment. La sorite « **position\_o** » sur 16 bits est la sortie du compteur qui indiquera la position de la table tournante.

La machine d’état s’occupera de fournir les sorties « **dir\_cw\_o** » afin d’indiquer la direction de rotation de la plaque. Cette sortie devra être utiliser par le compteur afin de savoir si la position doit être incrémenté ou décrémente. Lorsque le système détecte une séquence en rotation horaire, le compteur incrémente et dans le cas contraire, il décrémente.

Finalement la sortie « **det\_err\_o** » doit être fourni afin d’indiquer qu'une erreur s'est produite sur les signaux A-B du codeur (double changement).

## Schéma bloc

Afin d’avoir une bonne vue globale du système nous avons décidé de faire un schéma en indiquant les différents points importants pour ce système :



En peut voir en bas à gauche notre machine d’état (MSS) avec comme entrées le reset, clock, capteur\_a et capteur\_b ainsi que comme sorties un enable, dir\_cw\_o (direction de rotation) et le det\_err.

En haut à droite notre compteur qui prend en entrée le reset, le clock, le enable fourni par la MSS, le signal init\_pos\_i afin d’initialiser la position (pos = 0) et D qui est la position future en fonction du signal « dir\_cw\_o » aussi fourni par la MSS. Puis comme sortis, la position actuelle de la plaque.

Nous avons décidé de gérer l’erreur avec une bascule RS (en bas à droite) qui nous permettra de garder l’information qu’une erreur s’est produite tout en continuant à faire fonctionner le système. De plus, comme demander, le signal « init\_pos\_i » doit permettre de reset l’erreur à 0. Il n’est pas indiqué sur notre schéma, mais le reset sera bien évidemment braché sur la bascule RS afin de remettre aussi à 0 une éventuelle erreur.

Comme mentionné plus tôt, la sortie « dir\_cw\_o » permettra d’indiquer au compteur dans quel sens tourne la plaque et donc de savoir s’il faut incrémenter ou décrémenter la position actuelle.

Finalement, il est important d’indiquer au compteur à quel moment la position de la plaque change afin d’effectuer une opération sur la position présente. Pour ceci, nous fournissant un signal « enable » au compteur qui sera actif lors d’un changement d’état d’un des capteurs afin qu’il puisse mettre à jour la position de la plaque.

### Machine d’état

#### Conception

Afin de concevoir cette machine d’état nous nous sommes aidés du principe de la table des états. Afin de la concevoir, il a d’abord fallu définir nos différents états :

1. Etat de départ lors d’un allumage du système ou d’un reset **(Start)**
2. Etat d’une erreur lors d’un double changement sur les capteurs **(Err)**
3. Quatre états pour chaque position du codeur incrémental :
   1. Activation du enable dans le sens horaire **(Exy+)**
   2. Attente d’un changement de position dans le sens horaire **(Exy+w)**
   3. Activation du enable dans le sens anti-horaire **(Exy-)**
   4. Attente d’un changement de position dans le sens anti-horaire **(Exy-w)**

***X = état du capteur a***

***Y = état du capteur b***

En tout, cela nous fait 1 + 1 + (4 x 4) états (**18**). Il est important de faire deux états distincts, lors de la détection d’un changement d’état et lors de l’attente d’un changent de position. Cela nous permet d’activer le signal « enable » un seul coup de clock pour que le compteur effectue qu’une opération, et non pas plusieurs, pour un seul changement de position.

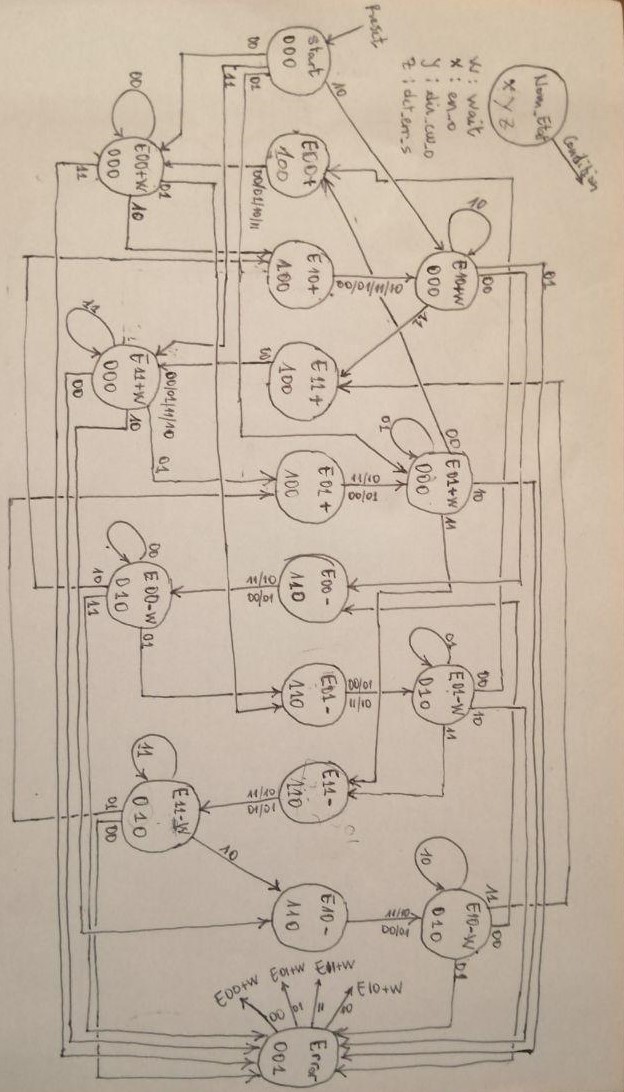
Voici notre table des états :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Etats** | **Entrées f(capt\_a\_i, capt\_b\_i)** | | | | **Sorties** | | |
| **00** | **01** | **11** | **10** | **en\_o** | **dir\_cw\_o** | **det\_err\_s** |
| **Start** | E00+w | E01+w | E11+w | E10+w | 0 | 0 | 0 |
| **E00+** | E00+w | E00+w | E00+w | E00+w | 1 | 0 | 0 |
| **E00+w** | **(E00+w)** | E01- | Err | E10+ | 0 | 0 | 0 |
| **E10+** | E10+w | E10+w | E10+w | E10+w | 1 | 0 | 0 |
| **E10+w** | E00- | Err | E11+ | **(E10+w)** | 0 | 0 | 0 |
| **E11+** | E11+w | E11+w | E11+w | E11+w | 1 | 0 | 0 |
| **E11+w** | Err | E01+ | **(E11+w)** | E10- | 0 | 0 | 0 |
| **E01+** | E01+w | E01+w | E01+w | E01+w | 1 | 0 | 0 |
| **E01+w** | E00+ | **(E01+w)** | E11- | Err | 0 | 0 | 0 |
| **E00-** | E00-w | E00-w | E00-w | E00-w | 1 | 1 | 0 |
| **E00-w** | **(E00-w)** | E01- | Err | E10+ | 0 | 1 | 0 |
| **E01-** | E01-w | E01-w | E01-w | E01-w | 1 | 1 | 0 |
| **E01-w** | E00+ | **(E01-w)** | E11- | Err | 0 | 1 | 0 |
| **E11-** | E11-w | E11-w | E11-w | E11-w | 1 | 1 | 0 |
| **E11-w** | Err | E01+ | **(E11-w)** | E10- | 0 | 1 | 0 |
| **E10-** | E10-w | E10-w | E10-w | E10-w | 1 | 1 | 0 |
| **E10-w** | E00- | Err | E11+ | **(E10-w)** | 0 | 1 | 0 |
| **Err** | E00+w | E01+w | E11+w | E10+w | 0 | 0 | 1 |

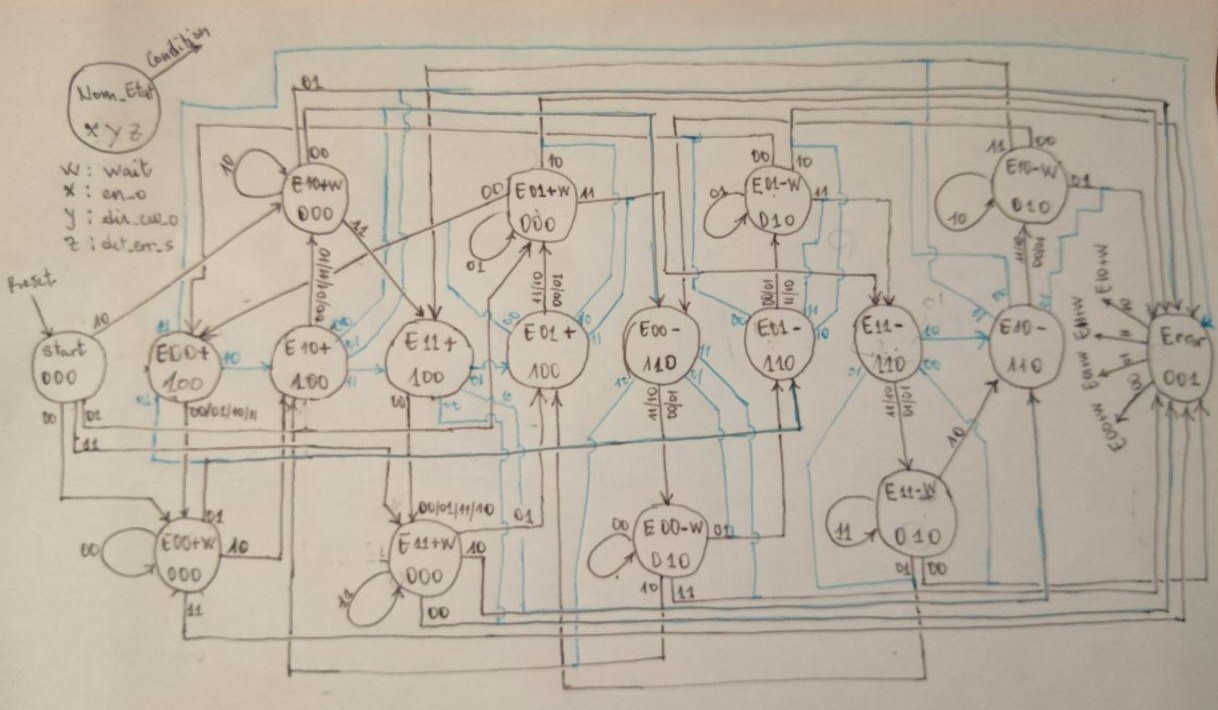
*dir\_cw\_o à 0 = sens de rotation horaire.*

*Remarque :*

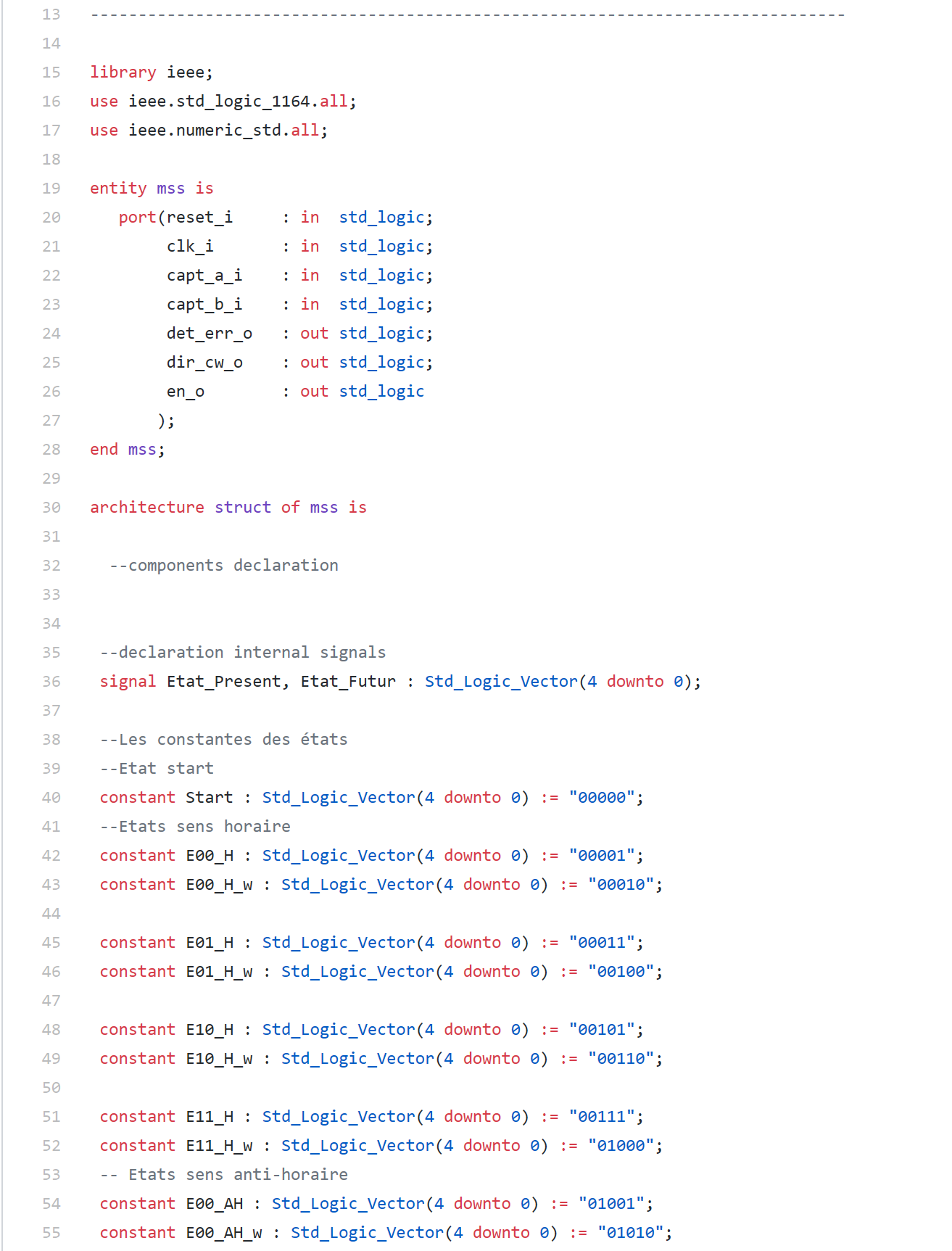
Dans l’état Start et Err, nous ne connaissance momentanément plus le sens de rotation. De ce fait, l’état qui suit un de ces états est directement un état wait (attend un prochain changement) afin de ne pas faire d’opération inconnue sur la position. Dans notre cas, nous avons choisis de se mettre dans les états ou la rotation est sens horaire mais cela n’a pas importante.

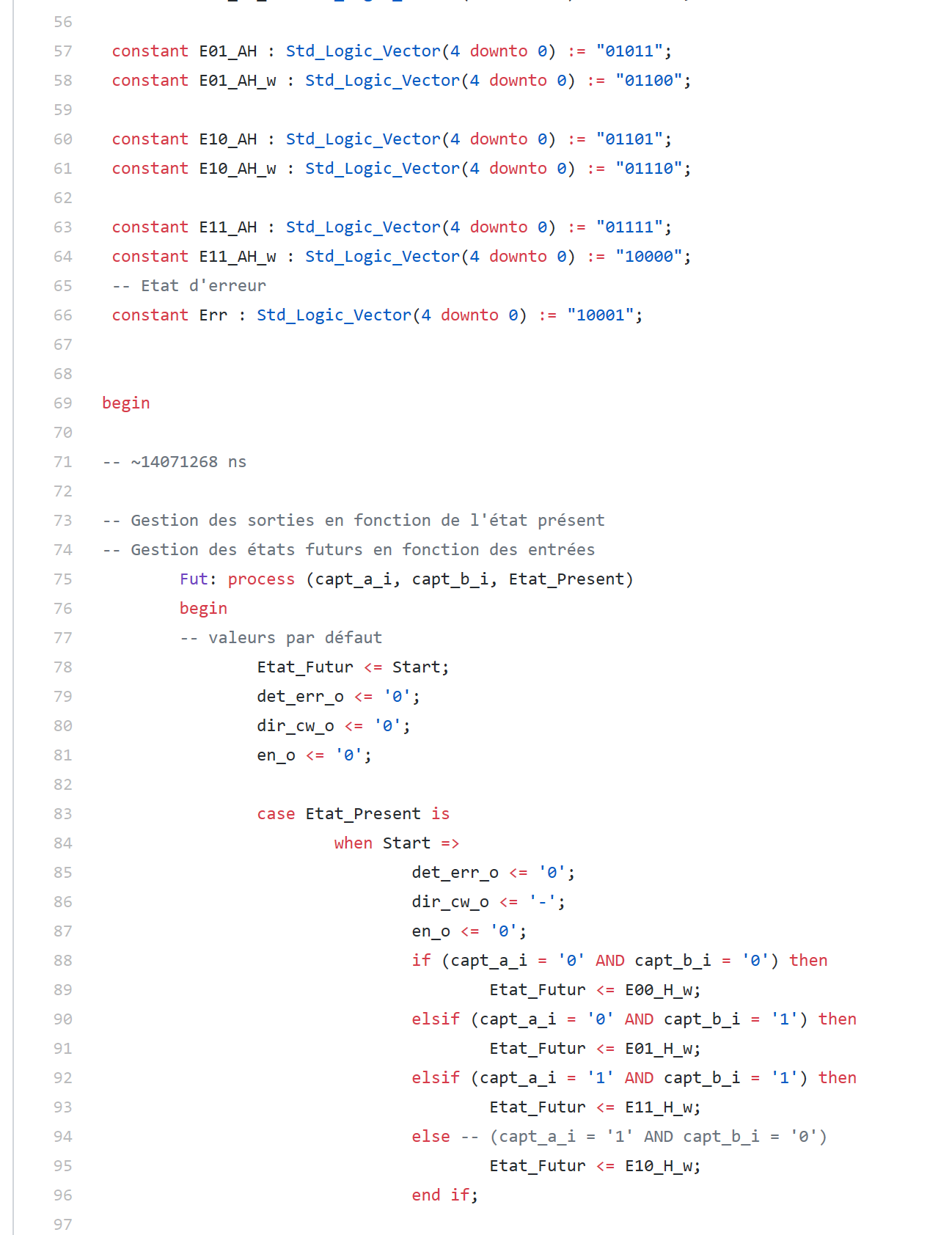
Afin d’avoir une meilleure vision du fonctionnement ains que pour facilement se repérer dans le VHDL, nous avons tout de même fait un graphe d’état :

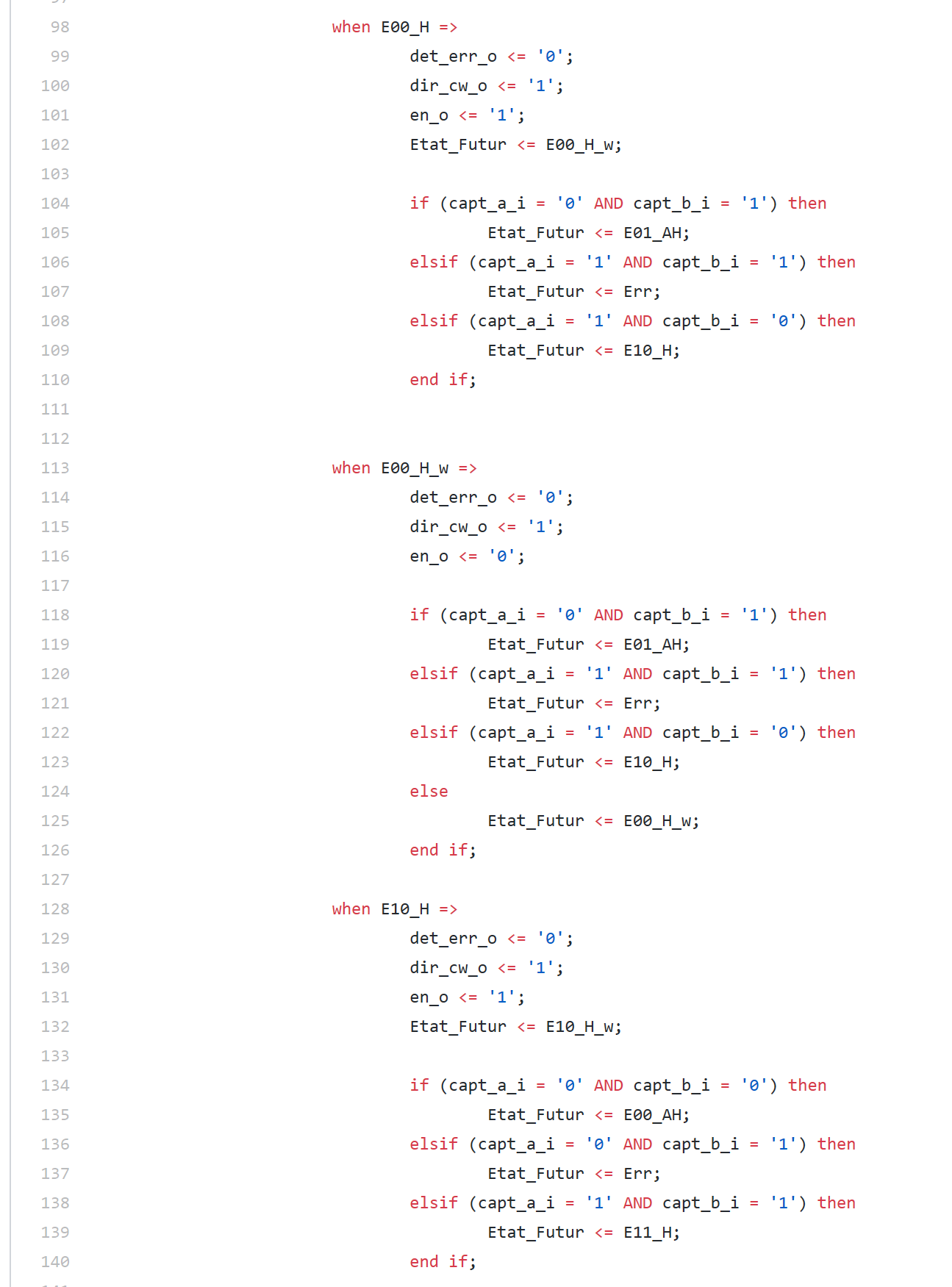
Ce graphe des états fut la première version de notre machine d’état. Une fois que celle-ci fut décrite en VHDL, nous avons mis ensemble tous les éléments du système et effectué une simulation. Nous avons toute de suite constaté qu’il y avait un retard qui se produisait un moment donné dans notre système. Grace l’assistant, nous avons pu trouver cette erreur et résolu le problème. L’erreur venait du fait que lorsque l’état présent est dans l’un des états (E001+, E10+, E11+, E01+, E001-, E10-, E11-, E01-), le système était contraint à avoir un état futur soit (E001+w, E10+w, E11+w, E01+w, E001-w, E10-w, E11-w, E01-w). Alors pour résoudre ce problème, nous avons due réadapter notre graphe d’état en rajoutant la possibilité d’aller vers les états futurs sans toutefois passer obligatoirement par l’état « wait », étant sur l’un des états (E001+, E10+, E11+, E01+, E001-, E10-, E11-, E01-). Le graphe obtenu est le suivant :

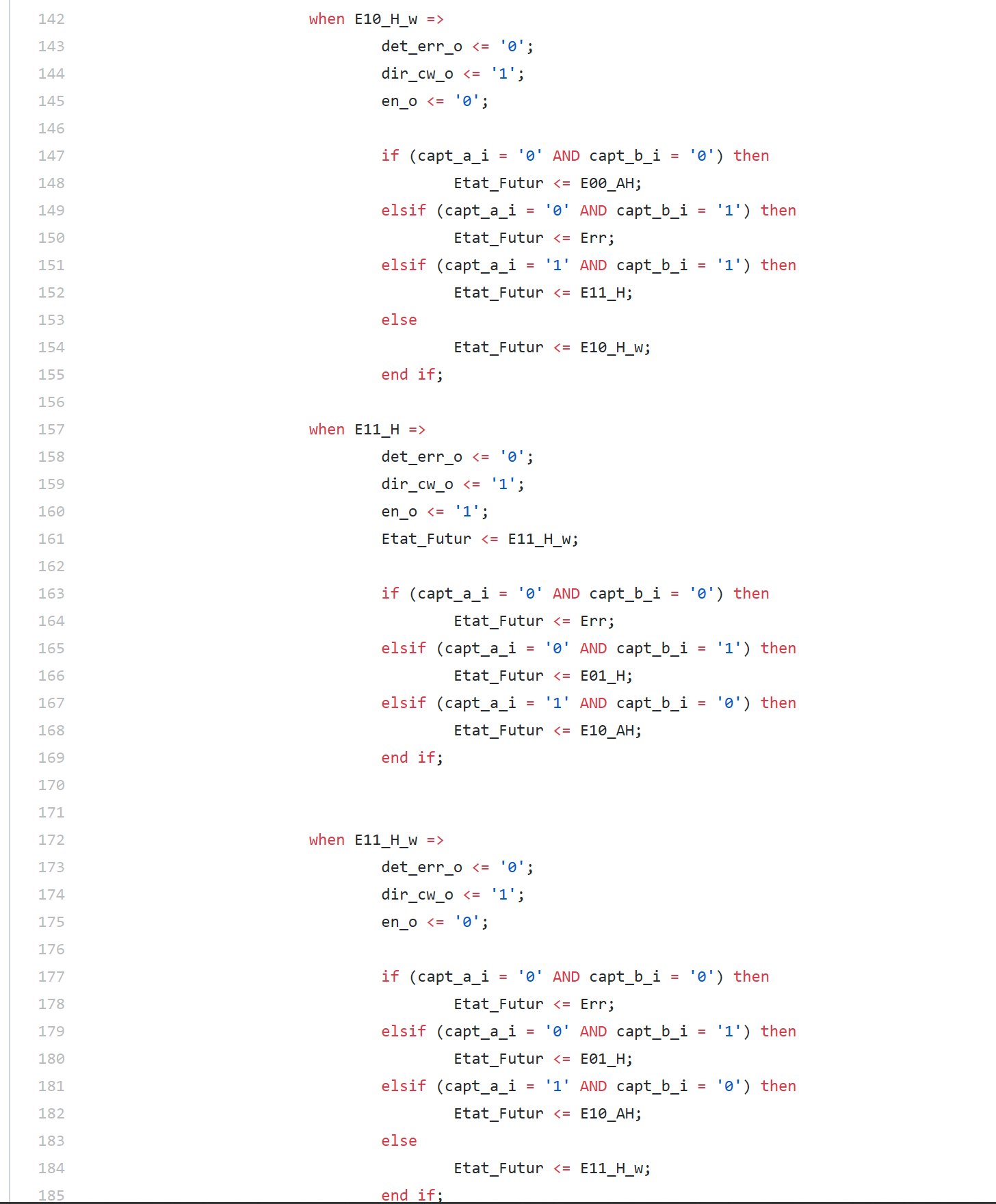


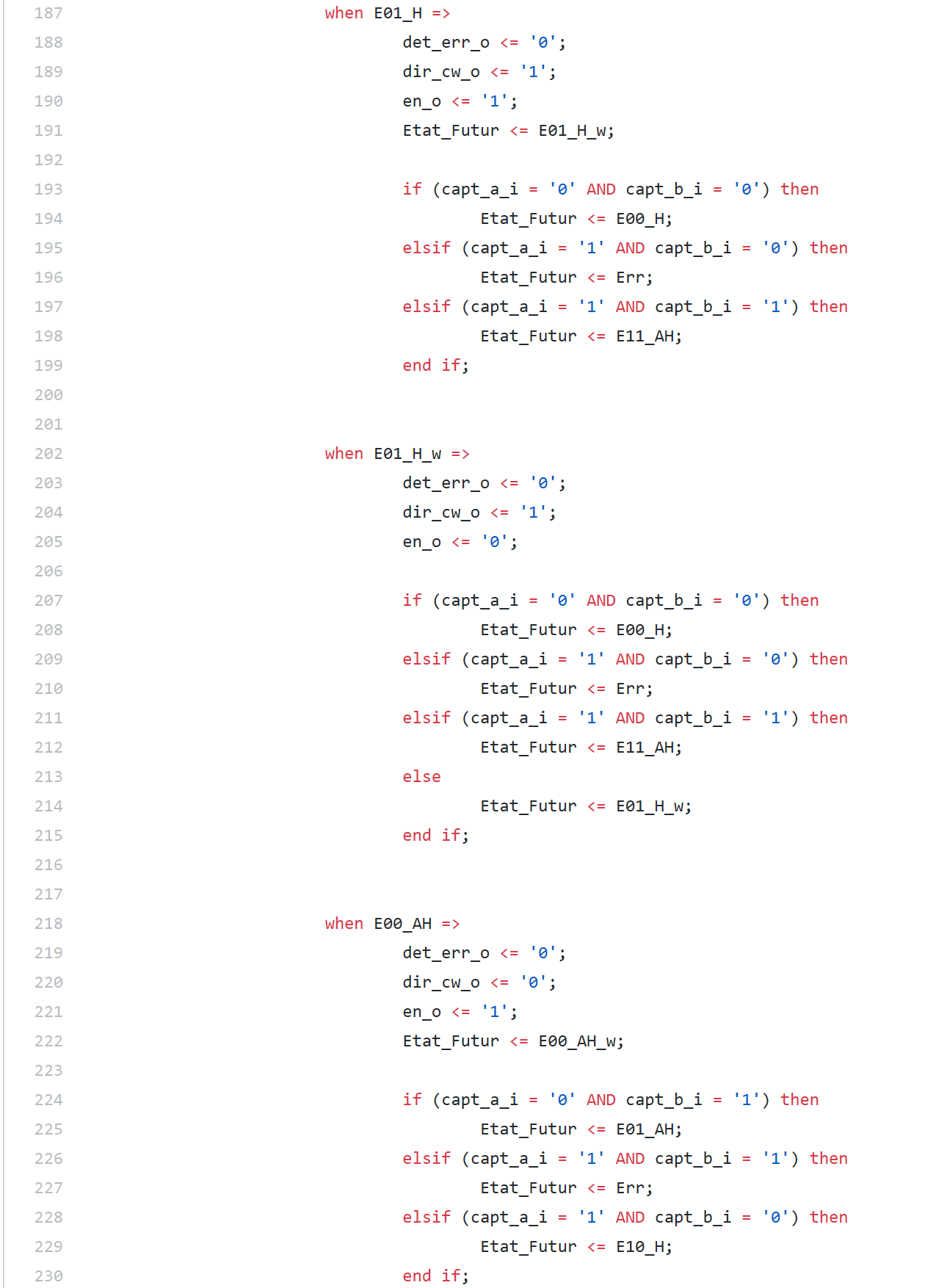
#### Description VHDL

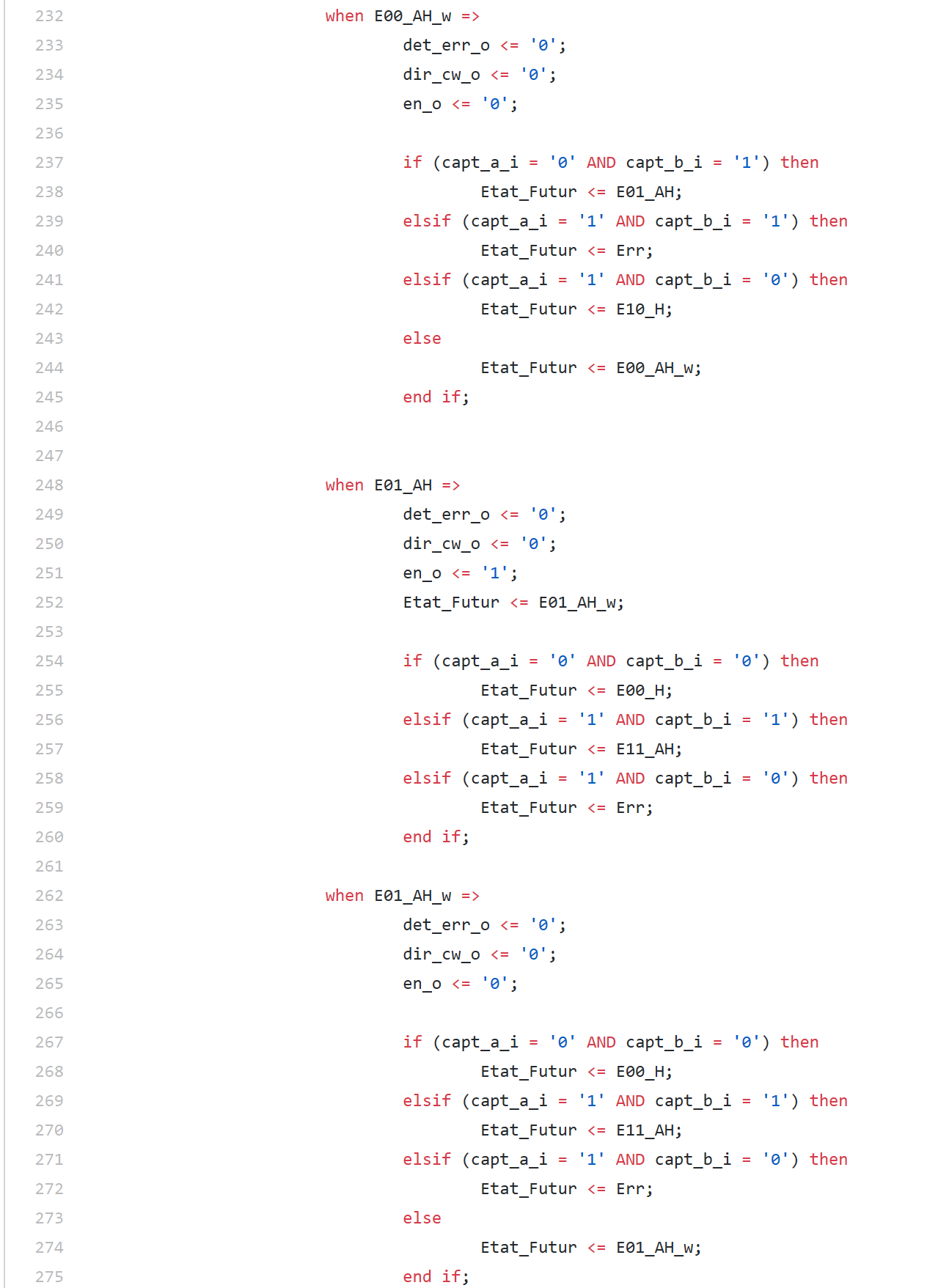


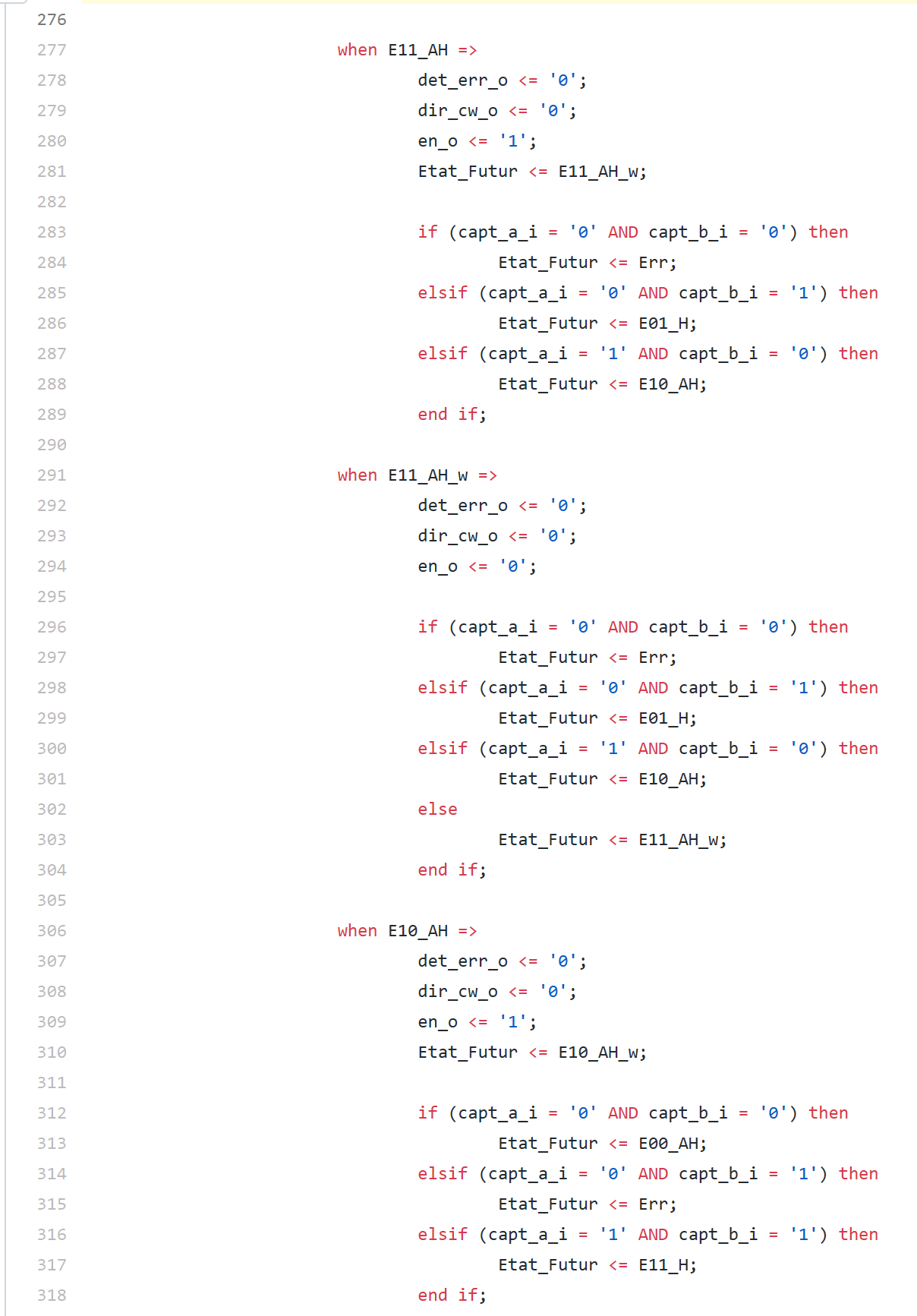


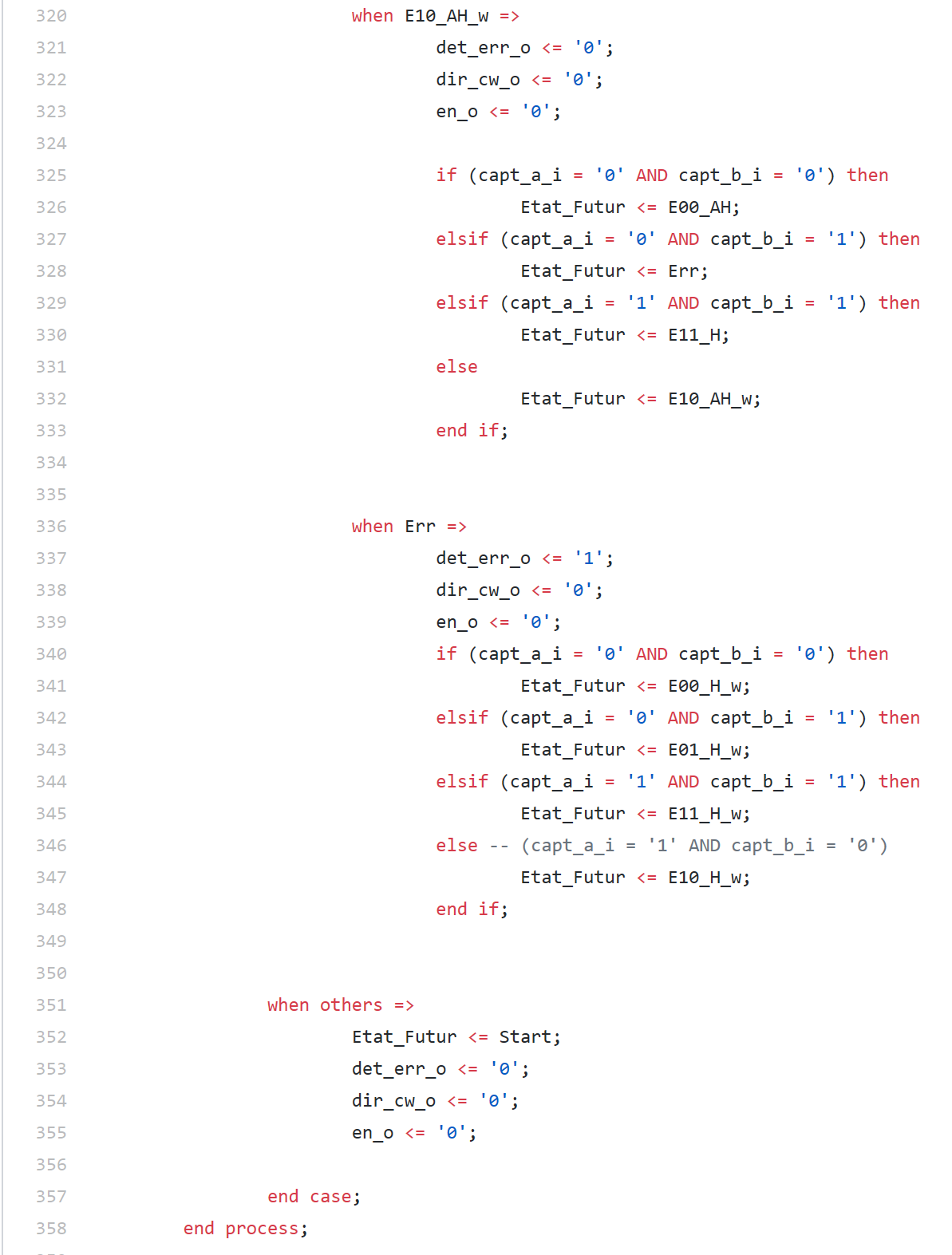


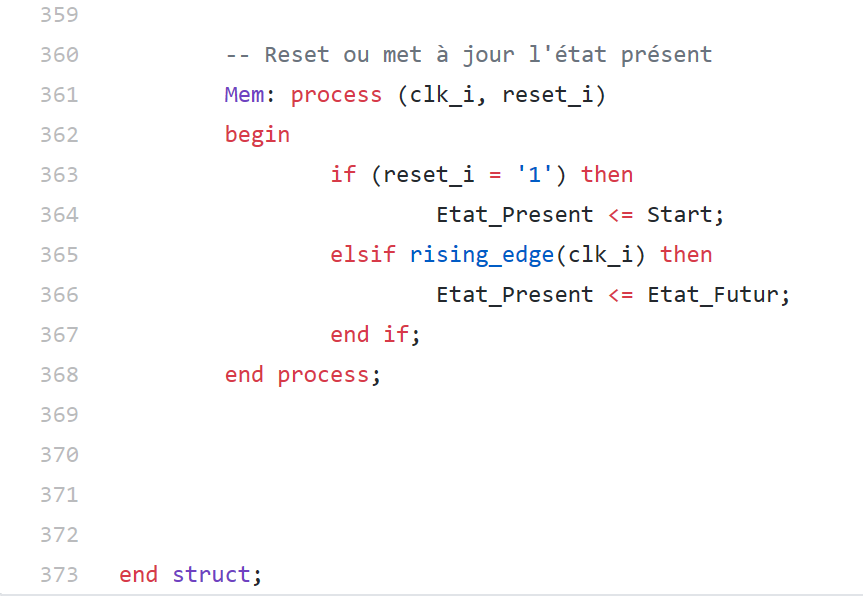








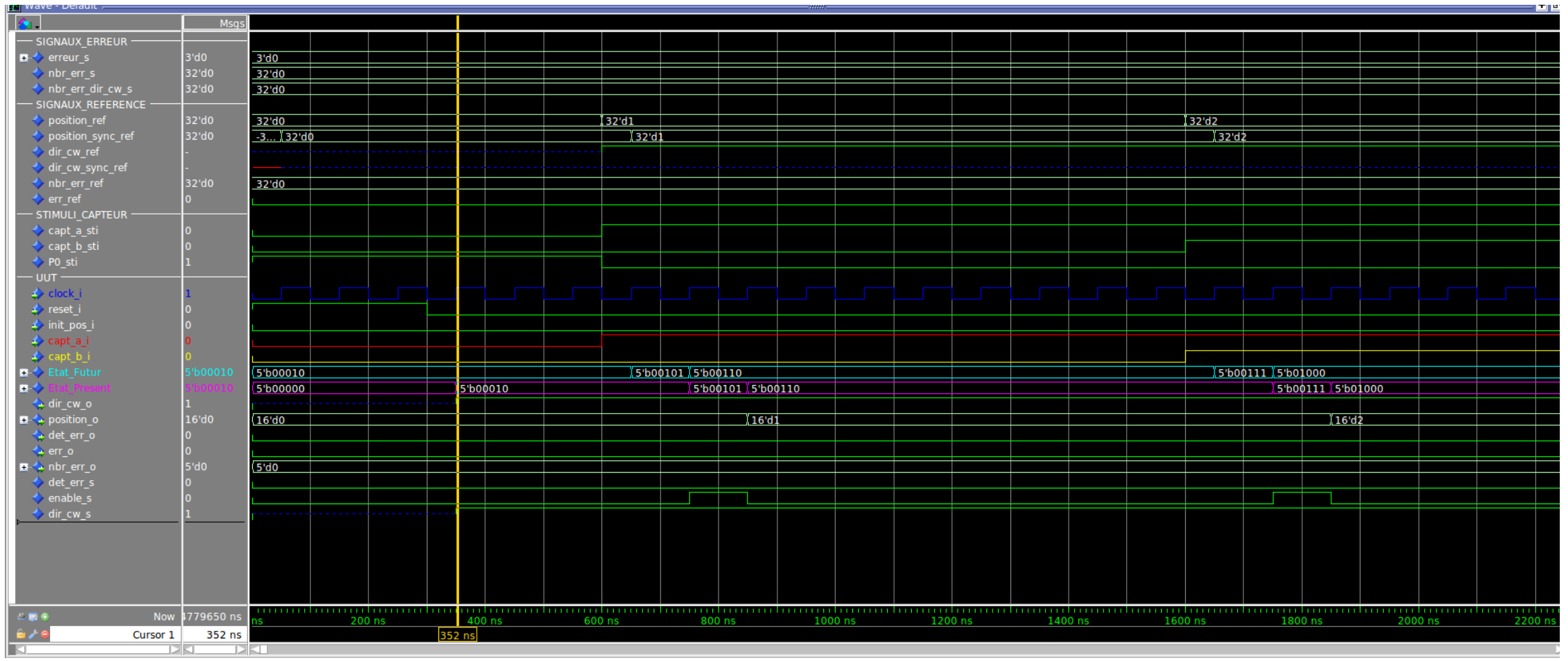




#### Vérification du fonctionnement

Comme nous l’avons décrit ci-dessus, il est question ici de montrer qu’à travers une simulation manuelle que notre machine d’état fonctionne tel que décrit dans notre analyse.

Pour le cas de notre machine d’état, nous allons choisir quelques cas qui démontre les différentes transitions entre les états.

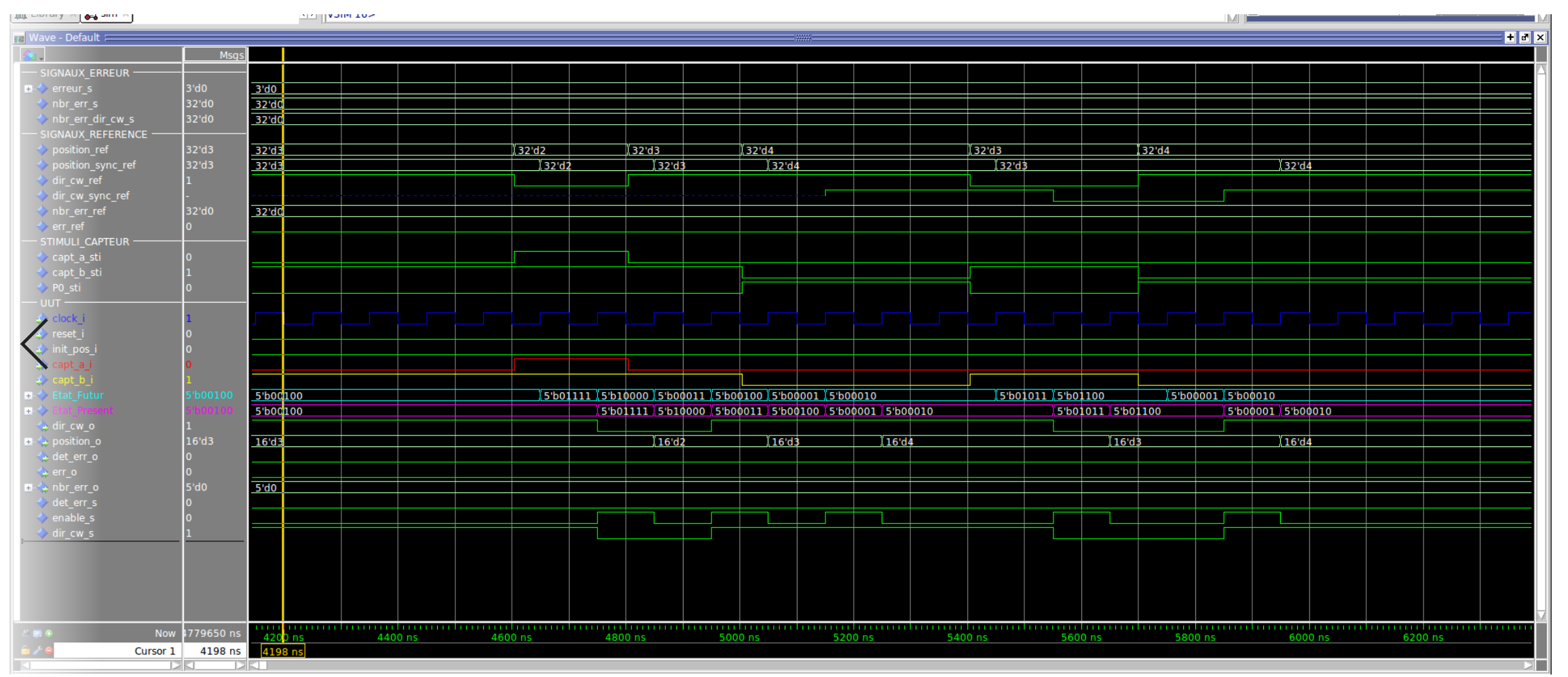


**Le cadre bleu** sur notre chronogramme permet de montrer qu’à l’état\_present = «00000 » => start , et qu’en entrée on a « capt\_a = 0, capt\_b= 0  », l’etat futur est « 00010 » qui correspond bien à notre état E00\_H\_w avec une direction horaire.

**Le cadre vert** sur notre chronogramme permet de montrer qu’à l’état\_present = «00010 » => « E00\_H\_w », et qu’en entrée on a « capt\_a = 1, capt\_b= 0 », l’état futur est « 00101 » qui correspond bien à notre état « E10\_H » avec une direction horaire. On peut constater quelque temps après qu’il y a un changement d’état , le système passe à l’état E10\_H\_w.

**Le cadre jaune** sur notre chronogramme permet de montrer qu’à l’état\_present(E10\_H\_w) = «00110 » , et qu’en entrée on a « capt\_a = 1, capt\_b= 1 », l’état futur est « 00111 » qui correspond bien à notre état « E11\_H» avec une direction horaire. De plus, on peut constater quelque temps après qu’il y a changement d’état, le système passe à l’état E11\_H\_w.

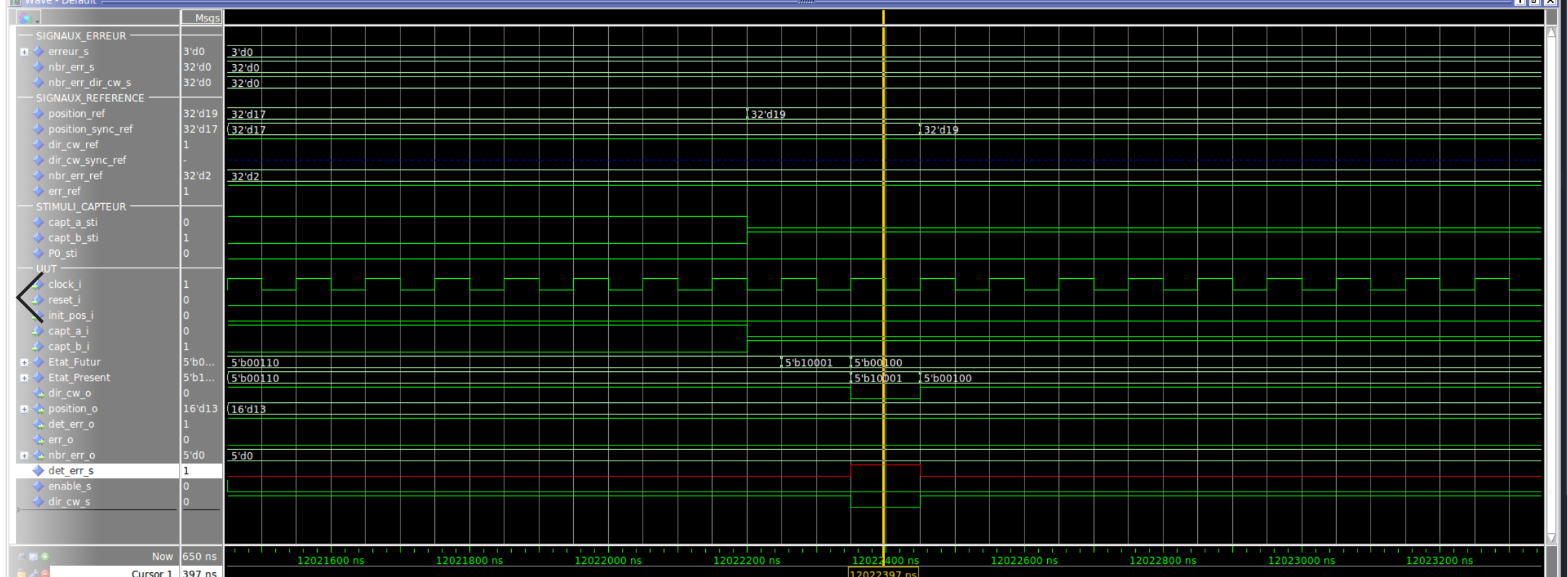
On peut aussi voir que le « enable est actif », l’état présent de notre système passe à l’état « wait ».



**Le cadre rouge** sur notre chronogramme (4600ns – 5200ns) permet de montrer que lorsque l’état\_present est «00100 » => « E01\_H\_w», avec une direction horaire et que en entrée on a « capt\_a = 1, capt\_b= 1», l’état futur est « 01111 » qui correspond bien notre état « E11\_AH» . On peut constater quelque temps après que « capt\_a = 0, capt\_b= 1» et un changement de direction en antihoraire (Entre 4750ns – 4950ns) entraine une décrémentation du compteur, ensuite le système se met dans l’état « wait(E11\_AH\_w) » de l’état présent en cours.

Entre 4950ns-5200ns, on peut voir que quand « capt\_a = 0, capt\_b= 1», avec une direction horaire, l’état présent étant à « 00011 » => E01\_H , l’état futur est « 00100 » => E01\_H\_w(état wait de E01\_H). et par la suite quand l’état présent prend « 00100 » => E01\_H\_w, l’état futur passer à « 00001 » =>E00\_H, avec comme entrée « capt\_a = 0, capt\_b= 0».

On peut bien voir à partir de ces cas cités ci-haut que notre MSS fonctionne comme prévu dans notre analyse. Le compteur incrémente /décrémente comme il faut.



**Le cadre violet** permet de montrer que si en entrée on a « capt\_a = 0, capt\_b= 1», enable = 0, direction = antihoraire, état présent = « 00110 » => E10\_H\_w , une erreur est activé et on passe à l’état erreur et son état futur pour cette même entrée est « 00100 » => E01\_H\_w. Ceci montre un cas de situation d’erreur que notre système gère et respecte bien notre démarche selon le graphe des états.

### Compteur et décodeur d’état futur

Afin d’avoir un suivi de la position, il est indispensable d’avoir un compteur avec un décodeur d’état futur. Il y a donc un registre de 16 bits afin de mémoriser ou initialiser la valeur de la position ainsi qu’un décodeur d’état futur afin de savoir s’il faut incrémenter ou décrémenter la position actuelle.

Pour la réalisation de ce compteur, nous avons pris le compteur réalisé dans le précèdent labo(celui du groupe d’**Isaia**) et apporté quelques modifications.

#### Conception

Voici la table des fonctions synchrone du compteur :

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Init\_pos\_i** | **En\_i** | **dir\_cw\_i** | **Cpt\_pres** | **Cpt\_fut** | **Fonctions** |
| 1 | - | - | - | = 0 | Initialisation de la position à 0 |
| 0 | 0 | - | . | = Cpt\_pres | Maintien de la position |
| 0 | 1 | 0 | - | = Cpt\_pres-1 | Décrémentation de la position |
| 0 | 1 | 1 | - | = Cpt\_pres+1 | Incrémentation de la position |

On peut voir que notre compteur est relativement simple. Il y a une fonction prioritaire d’initialisation (init\_pos\_i) commander par l’utilisateur. De plus, il y a une fonction d’activation (en\_i) fourni par la machine d’état dans le cas où il y a un changement de position. Finalement, si l’enable est actif, en fonction de l’entrée « dir\_cw\_i » aussi fourni par la MSS, une opération d’incrémentation ou de décrémentation en fonction du sens de rotation.

#### Description VHDL





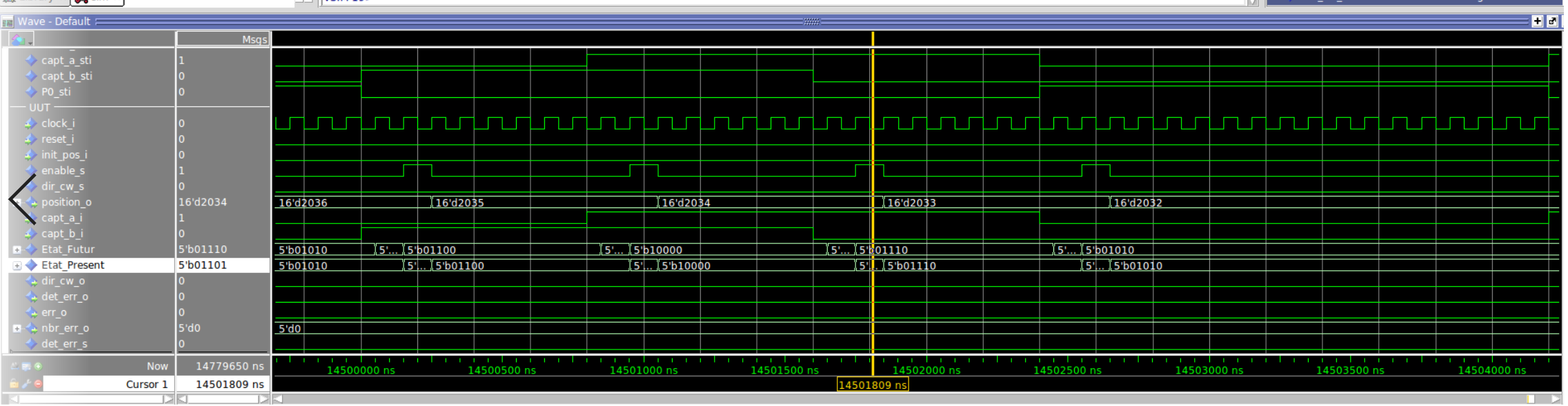
#### Vérification du fonctionnement

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Init\_pos\_i** | **En\_i** | **dir\_cw\_i** | **Cpt\_pres** | **Cpt\_fut** | **Fonctions** |
| 0 | 1 | 1 | - | = Cpt\_pres+1 | Incrémentation de la position |

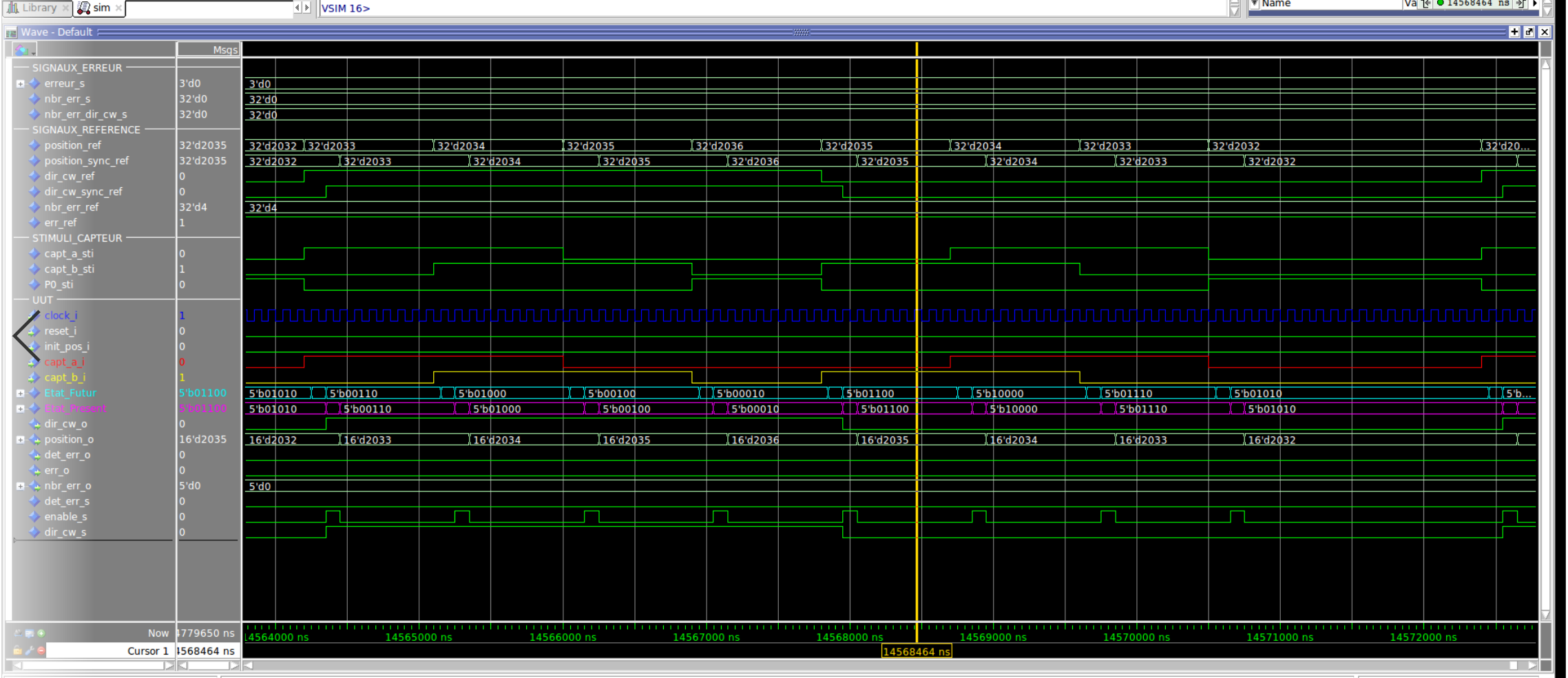


Cette capture du chronogramme de notre simulation nous montre que le compteur incrémente la position comme indiqué dans notre table de fonctionnement de notre compteur.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Init\_pos\_i** | **En\_i** | **dir\_cw\_i** | **Cpt\_pres** | **Cpt\_fut** | **Fonctions** |
| 0 | 1 | 0 | - | = Cpt\_pres-1 | décrémentation de la position |



Cette capture du chronogramme de notre simulation nous montre que le compteur décrémente la position comme indiqué dans notre table de fonctionnement de notre compteur.



On peut bien voir ici que la partie encadrée en rouge montre le changement de direction et ensuite, on voit bien que notre compteur qui vient d’incrémenter la position, se met directement à la décrémenter.

De plus, nous avons vérifié que le compteur n’incrémente pas la position tant que enable n’est pas acitf. Finalement, nous avons aussi vérifié que si le signal init\_pos\_i est actif, la position est mise à 0.

Au vu de toutes ces vérifications, on peut donc affirmer que notre bloc compteur fonctionne comme indiqué dans analyse et les résultats sont corrects.

### Bascule RS

Le dernier bloc est une simple bascule RS permettant d’enregistrer l’arrivé d’une erreur.

#### Conception

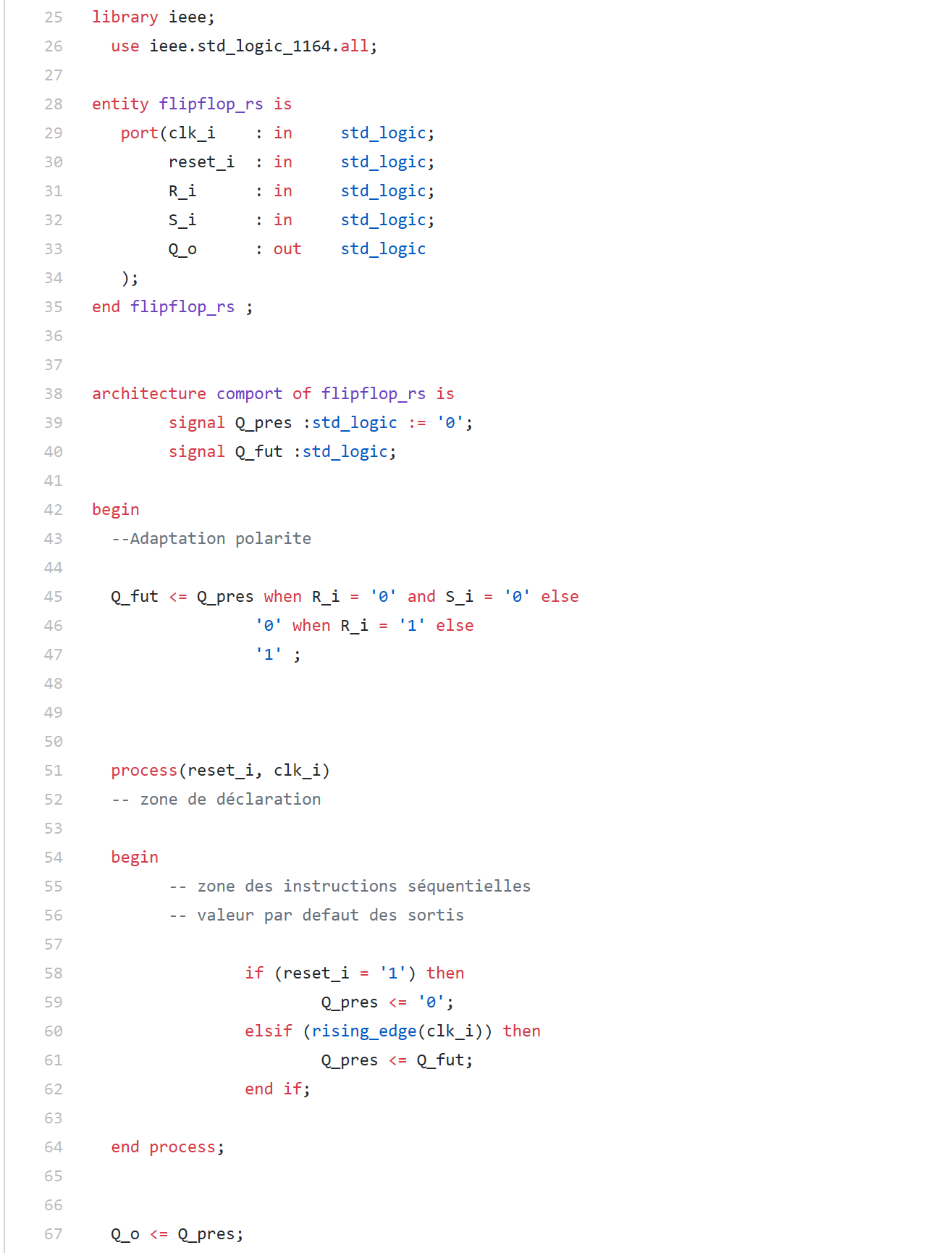
Le signal d’erreur levé par la MSS est directement connecté à l’entrée S de la bascule afin de la mémoriser.

Étant donné que l’erreur doit pouvoir être désactivé avec le signal « init\_pos\_i » celui-ci sera placé sur l’entrée R de la bascule.

Finalement, le signal reset\_i est bien évidemment branché sur le reset de la bascule.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Init\_pos\_i** | **Det\_err** | **Q\_pres** | **Q\_fut** | **Fonction** |
| 0 | 0 | - | Q\_pres | Mémorisation |
| 0 | 1 | - | 1 | Mise à 1 de l’erreur |
| 1 | 0 | - | 0 | Mise à 0 de l’erreur |

#### Description VHDL



Ici on peut voir que nous avons modifié la logique de la bascule RS afin que si R et S soit à ‘1’ la sortie soit à ‘1’ afin de s’assurer de voir qu’une erreur se soit levée.

#### 



#### Vérification du fonctionnement

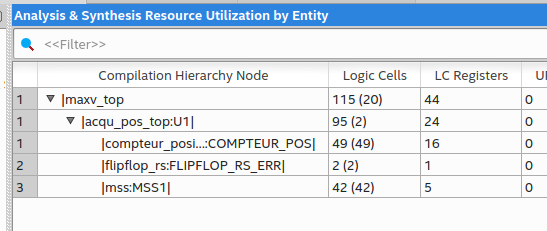
Pour ce bloc, nous avons simplement repris la bascule RS d’un précèdent laboratoire et modifié l’état Q si R et S sont à 1. Nous avons donc testé simplement ce bloc à l’aide de la console en aplatant le fichier de la console sim. Une fois que les 4 états ont été correctement validées, nous avons pu affirmer que ce bloc était fonctionnel.

## Regroupement des blocs

Afin de regrouper tous nos blocs, nous avons compléter le fichier « acqu\_pos\_top.vhd » fourni afin déclarer et d’instancier nos blocs pour les relier entre eux selon la décomposition hiérarchique établie précédemment.

### Synthèse ou quantité logique

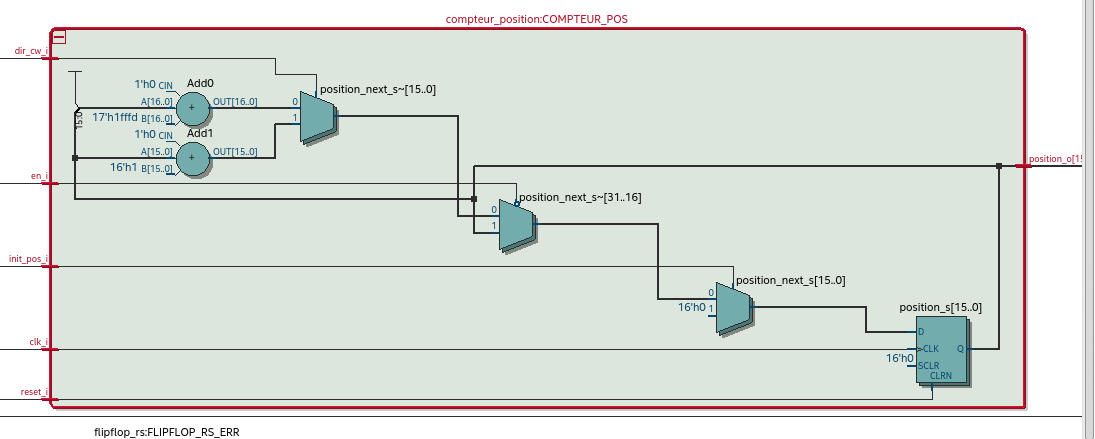
Ensuite, nous avons pu faire la synthèse de notre solution afin de déterminer la quantité de logique utilisé.



On peut voir que sans le maxv\_top, notre solution utilise 95 quantités de logique. Afin d’y voir plus claire, nous avons décider d’analyser les vues RTLs.

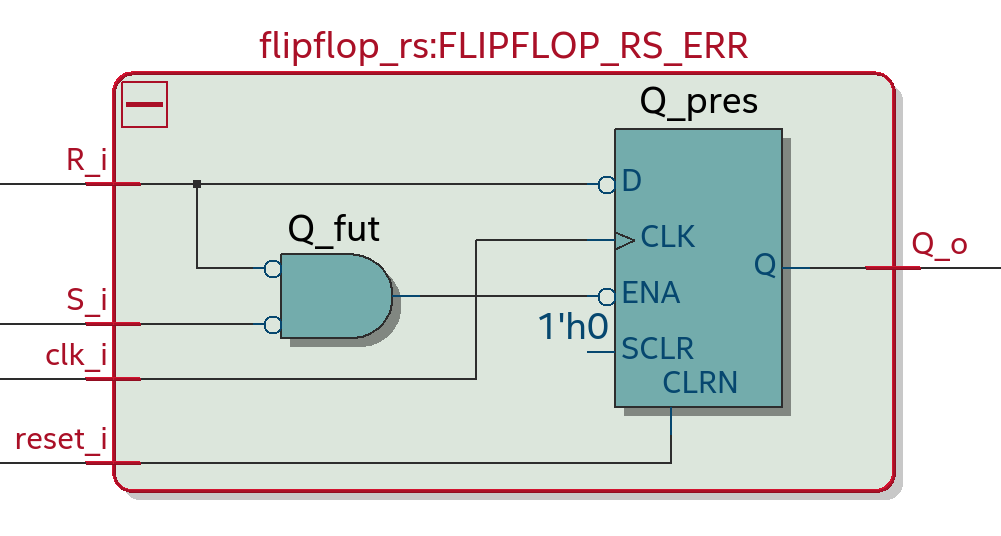
#### Compteur de position

Le compteur de position utilise 49 quantités logiques :



Cela se comprend après avoir observé la vue RTL : 16 -> additionneur / 17 -> soustracteur / 16 flip-flop de mémorisation de la position.

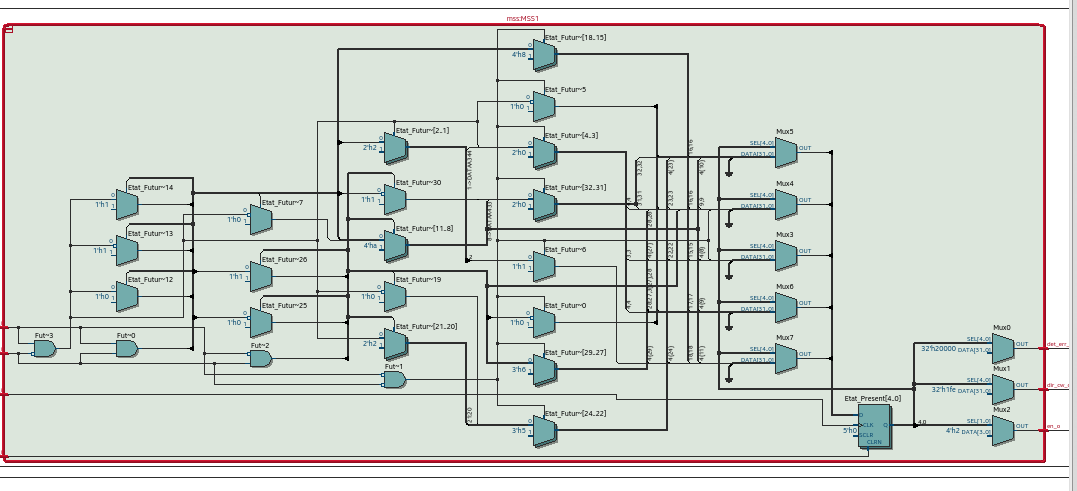
#### Bascule RS



La bascule RS utilise 2 quantité logiques, 1 simplement pour la logique combinatoire et l’autre pour le flip-flop de mémorisation de l’état.

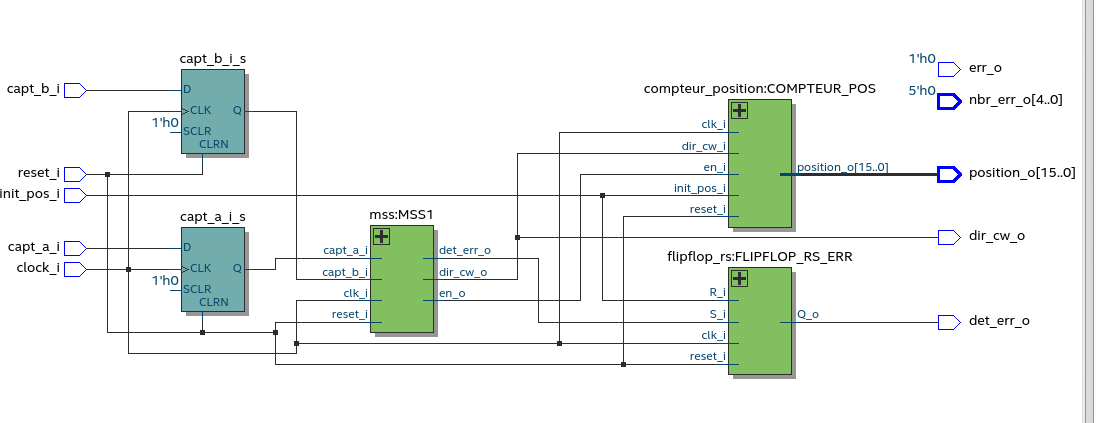
#### MSS

La machine d’état utilise 42 quantités logiques :



Dont 5 flip-flop pour mémoriser l’état présent et 37 pour la logique des 18 différents états, ce qui fait une moyenne d’environ 2 logiques par état.

#### Top (acqu\_pos\_top)



Remarque : Après avoir testé notre système sur le montage réel, nous avons pu constater qu’il fallait synchroniser les entrées des capteurs a et b ce qui explique les 2 flip-flops dans l’acqu\_pos\_top.

### Vérification du fonctionnement

Enfin, nous avons pu vérifier le fonctionnement du système acqu\_pos\_top.vhd à l’aide d’une simulation automatique dans Questasim. Nous avons utilisé le script automatique run\_acqu\_pos\_tb.tcl. De plus, nous avons dû adapter «console\_sim.vhd » et nous avons modifier le script « wave\_acqu\_pos.do » afin d’ajouter l’état présent et futur pour avoir une meilleure vision du système afin de débugger plus facilement.

Temporairement, grâce à l’aide de M.Meserli, nous avons modifier le test bench afin de ne pas tester les signaux que nous devions pas gérer (err\_o et nbr\_err\_o). De plus, une attente à de l’être ajouté après le relâchement du reset.

Finalement, après correction du test bench, notre système générait 0 erreurs :





Maintenant que nous savons que notre système passe le test bench, nous pouvons tester notre solution sur le montage réel.

## Test du système

Afin de tester le montage final, nous avons fait la synthèse et le placement routage de notre solution pour le système d’acquisition de position à l’aide du logiciel Quartus. Puis, nous avons programmer le circuit Max-V de la carte.

Après avoir programmer la carte et fait tous les connections, nous avons suivi les instructions dans la donnée afin de pouvoir utiliser la console pour tester notre système sur un montage réel.

Nous avons donc testé notre système en tourant la plaque de tous les côtés avec plusieurs vitesses différentes. Nous avons aussi testé le bouton d’init ainsi que le bouton de reset. Après avoir fait tous ces tests, nous avons demandé pour faire valider notre solution.

Au final, notre système a pu être validé par M.Meserli le jeudi 12 décembre.

## Conclusion

### Difficultés rencontrées

Au début nous avons eu de la difficulté pour déboguer notre système avec le test bench étant donné qu’il y a énormément de signaux et que nos états présent et futur n’était pas affiché.

### Compétences acquises

Grâce à l’aide de Mike Meury, nous avons pu afficher nos états présents et future. De plus, il nous a montré comment modifier la couleur d’un signal ce qui est vite pratique lors qu’il y a beaucoup de signaux.

De plus, nous nous sommes familiarisés avec la méthodologie de travail ainsi qu’avec la conception de machine d’état via une table d’état et un graphe d’état.

### Résultats obtenus

Evidemment, tout n’a pas fonctionné du premier coup mais nous avons tout de même réussi à parvenir à un système complètement fonctionnel et nous en somme particulièrement fier.

*Date : 18.12.19*

*Nom des étudiants : Lankeu Ngassam Cédric et Spinelli Isaïa*