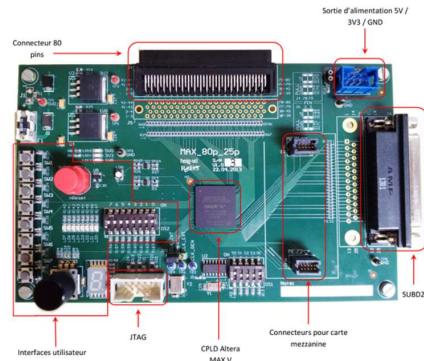


Acquisition de position

CONCEPTION DE SYSTÈMES NUMÉRIQUES (CSN)



Auteur : Spinelli Isaia et
Lankeu Ngassam Cédric
Prof : Etienne Messerli
Ing : Sébastien Masle
Date : 28.11.2019
Salle : A09
Classe : CSN

Table des matières

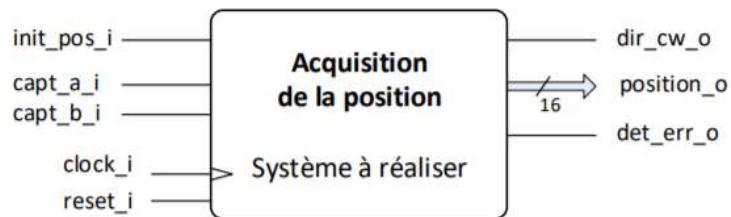
| | |
|---|--------|
| Objectifs | - 2 - |
| Présentation du système..... | - 2 - |
| Spécification du système..... | - 3 - |
| Analyse du fonctionnement..... | - 4 - |
| Schéma bloc | - 5 - |
| Machine d'état | - 6 - |
| Compteur et décodeur d'état futur | - 19 - |
| Bascule RS..... | - 23 - |
| Regroupement des blocs..... | - 24 - |
| Synthèse ou quantité logique..... | - 24 - |
| Vérification du fonctionnement..... | - 26 - |
| Test du système..... | - 28 - |
| Conclusion | - 29 - |
| Difficultés rencontrées | - 29 - |
| Compétences acquises..... | - 29 - |
| Résultats obtenus..... | - 29 - |

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 entraîné 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 :



Entrées/sorties du système

| Port | Direction | Taille | Description |
|-------------------|-----------|--------|--|
| init_pos_i | entrée | 1 | Signal d'initialisation de la position (position = 0) |
| capt_a_i | entrée | 1 | Signaux venant du codeur incrémental soit: |
| capt_b_i | entrée | 1 | <ul style="list-style-type: none"> • capt_a_i signal A du codeur • capt_b_i signal B du codeur |
| dir_cw_o | sortie | 1 | Signal indiquant le sens de rotation actuel de la table tournante |
| position_o | sortie | 16 | Sortie de 16 bits indiquant la position de la table tournante. La valeur est un entier positif. |
| det_err_o | sortie | 1 | Sortie indiquant qu'une erreur s'est produite sur les signaux A-B du codeur (double changement). |
| clock_i | entrée | 1 | Horloge du système synchrone, 1 MHz |
| reset_i | entrée | 1 | Signal de remise à zéro asynchrone |

Spécification du système

Le système reçoit aussi un signal d'initialisation permettant une remise à zéro de la position.

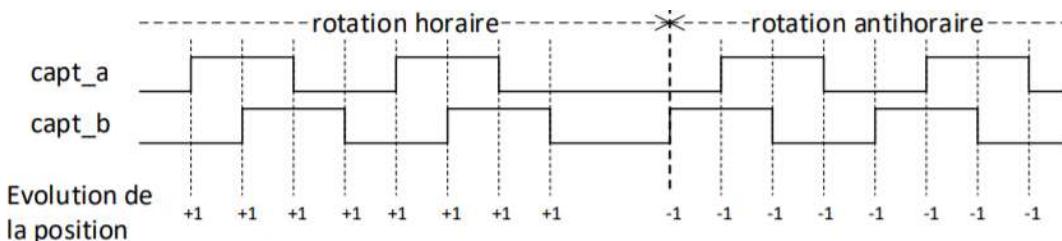
Description du fonctionnement de la position :

- 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 :



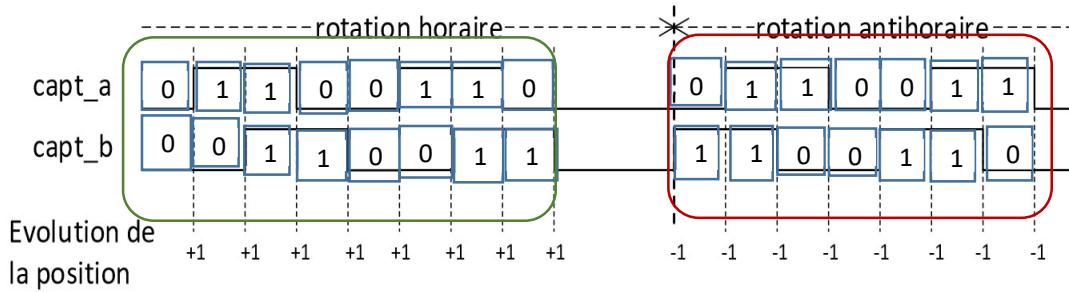
Spécifications de la table tournante :

Le codeur de la table est de type HEDS-5310. Celui-ci a une résolution de 500 traits. Cela permet d'obtenir 2000 incrémentations 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 incrémentations.

Analyse du fonctionnement

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

Voici un exemple de fonctionnement :



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.

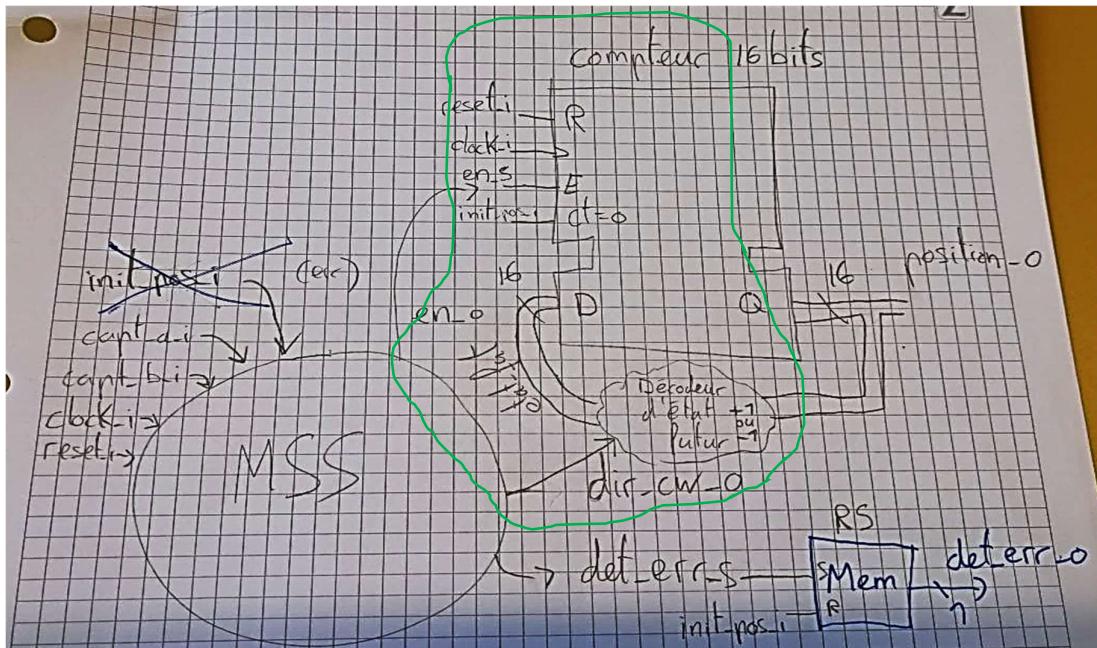
Plusieurs sorties sont demandées pour ce système comme détaillé précédemment. La sortie « **position_o** » sur 16 bits est la sortie du compteur qui indiquera la position de la table tournante.

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 utilisée par le compteur afin de savoir si la position doit être incrémenté ou décrémenté. Lorsque le système détecte une séquence en rotation horaire, le compteur incrémenté 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 :



On 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 :
 - a. Activation du enable dans le sens horaire (**Exy+**)
 - b. Attente d'un changement de position dans le sens horaire (**Exy+w**)
 - c. Activation du enable dans le sens anti-horaire (**Exy-**)
 - d. 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 $1 + 1 + (4 \times 4)$ 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 changement 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 :

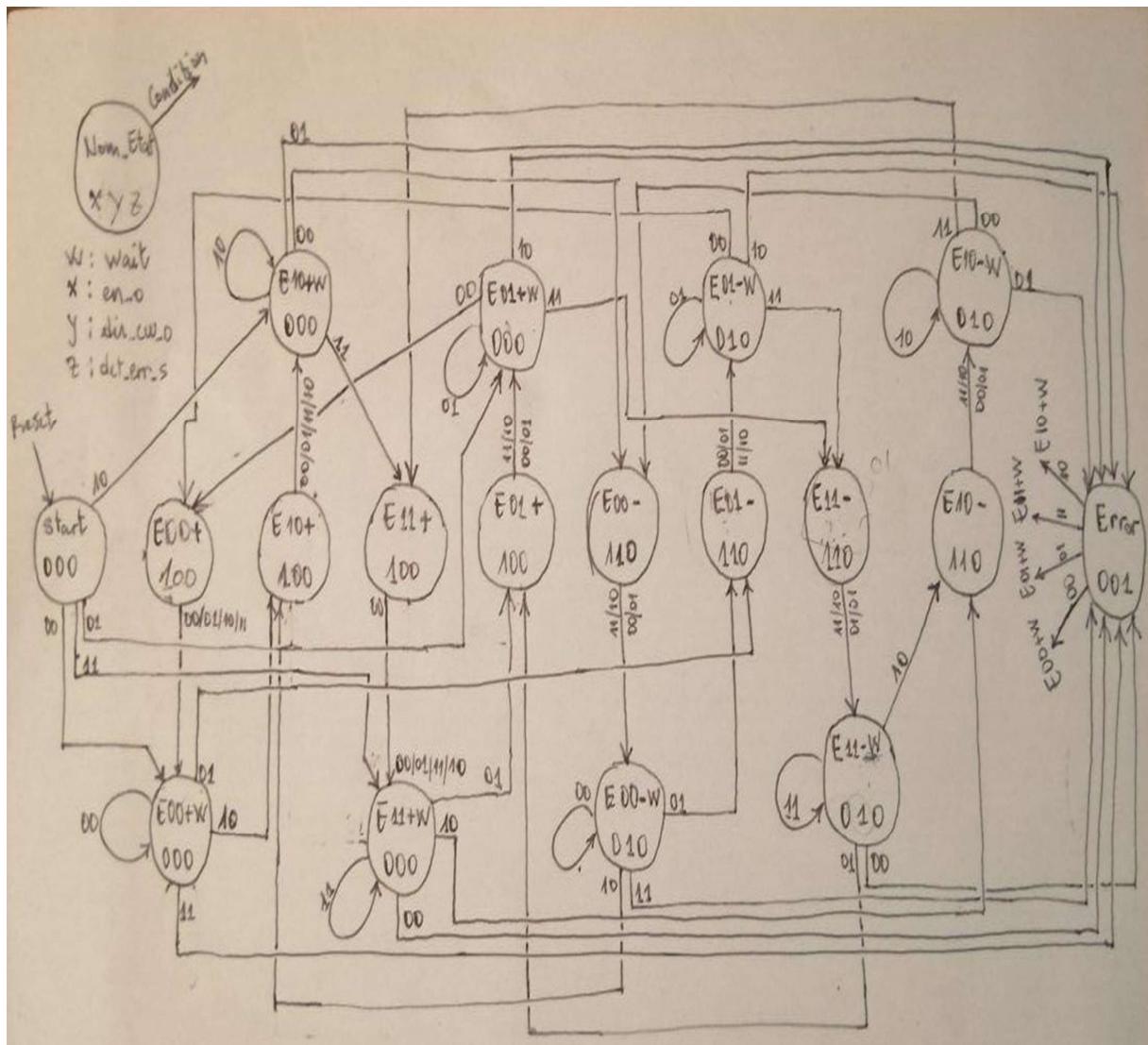
| Etats | Entrées f(capt_a_i, capt_b_i) | | | | Sorties | | |
|--------------|-------------------------------|---------|---------|---------|---------|----------|-----------|
| | 00 | 01 | 11 | 10 | en_o | dir_cw_o | det_err_s |
| Start | E00+w | E01+w | E11+w | E10+w | 0 | 0 | 0 |
| E00+ | E00+w | E00+w | 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 :

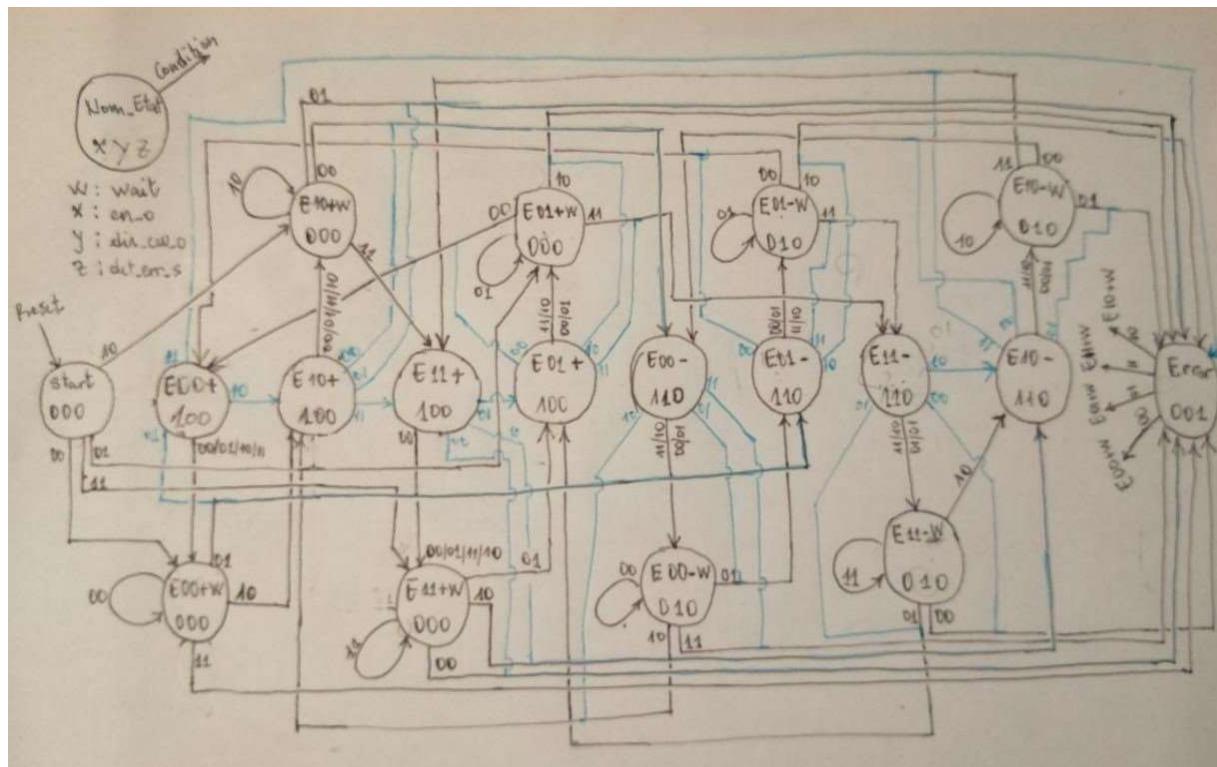
Dans l'état Start et Err, nous ne connaissons momentanément plus le sens de rotation. De ce fait, l'état qui suit un de ces états est directement un état wait (attend un prochain changement) afin de ne pas faire d'opération inconnue sur la position. Dans notre cas, nous avons choisis de se mettre dans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

Afin d'avoir une meilleure vision du fonctionnement ainsi 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. Grâce 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 dû 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

```
13 -----
14
15 library ieee;
16 use ieee.std_logic_1164.all;
17 use ieee.numeric_std.all;
18
19 entity mss is
20     port(reset_i      : in  std_logic;
21           clk_i       : in  std_logic;
22           capt_a_i    : in  std_logic;
23           capt_b_i    : in  std_logic;
24           det_err_o   : out std_logic;
25           dir_cw_o   : out std_logic;
26           en_o        : out std_logic
27       );
28 end mss;
29
30 architecture struct of mss is
31
32     --components declaration
33
34
35     --declaration internal signals
36     signal Etat_Present, Etat_Futur : Std_Logic_Vector(4 downto 0);
37
38     --Les constantes des états
39     --Etat start
40     constant Start : Std_Logic_Vector(4 downto 0) := "00000";
41     --Etats sens horaire
42     constant E00_H : Std_Logic_Vector(4 downto 0) := "00001";
43     constant E00_H_W : Std_Logic_Vector(4 downto 0) := "00010";
44
45     constant E01_H : Std_Logic_Vector(4 downto 0) := "00011";
46     constant E01_H_W : Std_Logic_Vector(4 downto 0) := "00100";
47
48     constant E10_H : Std_Logic_Vector(4 downto 0) := "00101";
49     constant E10_H_W : Std_Logic_Vector(4 downto 0) := "00110";
50
51     constant E11_H : Std_Logic_Vector(4 downto 0) := "00111";
52     constant E11_H_W : Std_Logic_Vector(4 downto 0) := "01000";
53     -- Etats sens anti-horaire
54     constant E00_AH : Std_Logic_Vector(4 downto 0) := "01001";
55     constant E00_AH_W : Std_Logic_Vector(4 downto 0) := "01010";
```

```
-- -- -- -- --  
56  
57  constant E01_AH : Std_Logic_Vector(4 downto 0) := "01011";  
58  constant E01_AH_W : Std_Logic_Vector(4 downto 0) := "01100";  
59  
60  constant E10_AH : Std_Logic_Vector(4 downto 0) := "01101";  
61  constant E10_AH_W : Std_Logic_Vector(4 downto 0) := "01110";  
62  
63  constant E11_AH : Std_Logic_Vector(4 downto 0) := "01111";  
64  constant E11_AH_W : Std_Logic_Vector(4 downto 0) := "10000";  
65  -- Etat d'erreur  
66  constant Err : Std_Logic_Vector(4 downto 0) := "10001";  
67  
68  
69  begin  
70  
71  -- ~14071268 ns  
72  
73  -- Gestion des sorties en fonction de l'état présent  
74  -- Gestion des états futurs en fonction des entrées  
75      Fut: process (capt_a_i, capt_b_i, Etat_Present)  
76      begin  
77          -- valeurs par défaut  
78          Etat_Futur <= Start;  
79          det_err_o <= '0';  
80          dir_cw_o <= '0';  
81          en_o <= '0';  
82  
83          case Etat_Present is  
84              when Start =>  
85                  det_err_o <= '0';  
86                  dir_cw_o <= '-';  
87                  en_o <= '0';  
88                  if (capt_a_i = '0' AND capt_b_i = '0') then  
89                      Etat_Futur <= E00_H_w;  
90                  elsif (capt_a_i = '0' AND capt_b_i = '1') then  
91                      Etat_Futur <= E01_H_w;  
92                  elsif (capt_a_i = '1' AND capt_b_i = '1') then  
93                      Etat_Futur <= E11_H_w;  
94                  else -- (capt_a_i = '1' AND capt_b_i = '0')  
95                      Etat_Futur <= E10_H_w;  
96                  end if;  
97
```

```

-- 
98      when E00_H =>
99          det_err_o <= '0';
100         dir_cw_o <= '1';
101         en_o <= '1';
102         Etat_Futur <= E00_H_w;
103
104         if (capt_a_i = '0' AND capt_b_i = '1') then
105             Etat_Futur <= E01_AH;
106         elsif (capt_a_i = '1' AND capt_b_i = '1') then
107             Etat_Futur <= Err;
108         elsif (capt_a_i = '1' AND capt_b_i = '0') then
109             Etat_Futur <= E10_H;
110         end if;
111
112
113     when E00_H_w =>
114         det_err_o <= '0';
115         dir_cw_o <= '1';
116         en_o <= '0';
117
118         if (capt_a_i = '0' AND capt_b_i = '1') then
119             Etat_Futur <= E01_AH;
120         elsif (capt_a_i = '1' AND capt_b_i = '1') then
121             Etat_Futur <= Err;
122         elsif (capt_a_i = '1' AND capt_b_i = '0') then
123             Etat_Futur <= E10_H;
124         else
125             Etat_Futur <= E00_H_w;
126         end if;
127
128     when E10_H =>
129         det_err_o <= '0';
130         dir_cw_o <= '1';
131         en_o <= '1';
132         Etat_Futur <= E10_H_w;
133
134         if (capt_a_i = '0' AND capt_b_i = '0') then
135             Etat_Futur <= E00_AH;
136         elsif (capt_a_i = '0' AND capt_b_i = '1') then
137             Etat_Futur <= Err;
138         elsif (capt_a_i = '1' AND capt_b_i = '1') then
139             Etat_Futur <= E11_H;
140         end if;
-- 

```

```

142      when E10_H_w =>
143          det_err_o <= '0';
144          dir_cw_o <= '1';
145          en_o <= '0';
146
147          if (capt_a_i = '0' AND capt_b_i = '0') then
148              Etat_Futur <= E00_AH;
149          elsif (capt_a_i = '0' AND capt_b_i = '1') then
150              Etat_Futur <= Err;
151          elsif (capt_a_i = '1' AND capt_b_i = '1') then
152              Etat_Futur <= E11_H;
153          else
154              Etat_Futur <= E10_H_w;
155          end if;
156
157      when E11_H =>
158          det_err_o <= '0';
159          dir_cw_o <= '1';
160          en_o <= '1';
161          Etat_Futur <= E11_H_w;
162
163          if (capt_a_i = '0' AND capt_b_i = '0') then
164              Etat_Futur <= Err;
165          elsif (capt_a_i = '0' AND capt_b_i = '1') then
166              Etat_Futur <= E01_H;
167          elsif (capt_a_i = '1' AND capt_b_i = '0') then
168              Etat_Futur <= E10_AH;
169          end if;
170
171      when E11_H_w =>
172          det_err_o <= '0';
173          dir_cw_o <= '1';
174          en_o <= '0';
175
176          if (capt_a_i = '0' AND capt_b_i = '0') then
177              Etat_Futur <= Err;
178          elsif (capt_a_i = '0' AND capt_b_i = '1') then
179              Etat_Futur <= E01_H;
180          elsif (capt_a_i = '1' AND capt_b_i = '0') then
181              Etat_Futur <= E10_AH;
182          else
183              Etat_Futur <= E11_H_w;
184          end if;
185

```

```

187      when E01_H =>
188          det_err_o <= '0';
189          dir_cw_o <= '1';
190          en_o <= '1';
191          Etat_Futur <= E01_H_W;
192
193          if (capt_a_i = '0' AND capt_b_i = '0') then
194              Etat_Futur <= E00_H;
195          elsif (capt_a_i = '1' AND capt_b_i = '0') then
196              Etat_Futur <= Err;
197          elsif (capt_a_i = '1' AND capt_b_i = '1') then
198              Etat_Futur <= E11_AH;
199          end if;
200
201
202      when E01_H_W =>
203          det_err_o <= '0';
204          dir_cw_o <= '1';
205          en_o <= '0';
206
207          if (capt_a_i = '0' AND capt_b_i = '0') then
208              Etat_Futur <= E00_H;
209          elsif (capt_a_i = '1' AND capt_b_i = '0') then
210              Etat_Futur <= Err;
211          elsif (capt_a_i = '1' AND capt_b_i = '1') then
212              Etat_Futur <= E11_AH;
213          else
214              Etat_Futur <= E01_H_W;
215          end if;
216
217
218      when E00_AH =>
219          det_err_o <= '0';
220          dir_cw_o <= '0';
221          en_o <= '1';
222          Etat_Futur <= E00_AH_W;
223
224          if (capt_a_i = '0' AND capt_b_i = '1') then
225              Etat_Futur <= E01_AH;
226          elsif (capt_a_i = '1' AND capt_b_i = '1') then
227              Etat_Futur <= Err;
228          elsif (capt_a_i = '1' AND capt_b_i = '0') then
229              Etat_Futur <= E10_H;
230          end if;

```

```

232         when E00_AH_W =>
233             det_err_o <= '0';
234             dir_cw_o <= '0';
235             en_o <= '0';
236
237             if (capt_a_i = '0' AND capt_b_i = '1') then
238                 Etat_Futur <= E01_AH;
239             elsif (capt_a_i = '1' AND capt_b_i = '1') then
240                 Etat_Futur <= Err;
241             elsif (capt_a_i = '1' AND capt_b_i = '0') then
242                 Etat_Futur <= E10_H;
243             else
244                 Etat_Futur <= E00_AH_W;
245             end if;
246
247
248         when E01_AH =>
249             det_err_o <= '0';
250             dir_cw_o <= '0';
251             en_o <= '1';
252             Etat_Futur <= E01_AH_W;
253
254             if (capt_a_i = '0' AND capt_b_i = '0') then
255                 Etat_Futur <= E00_H;
256             elsif (capt_a_i = '1' AND capt_b_i = '1') then
257                 Etat_Futur <= E11_AH;
258             elsif (capt_a_i = '1' AND capt_b_i = '0') then
259                 Etat_Futur <= Err;
260             end if;
261
262         when E01_AH_W =>
263             det_err_o <= '0';
264             dir_cw_o <= '0';
265             en_o <= '0';
266
267             if (capt_a_i = '0' AND capt_b_i = '0') then
268                 Etat_Futur <= E00_H;
269             elsif (capt_a_i = '1' AND capt_b_i = '1') then
270                 Etat_Futur <= E11_AH;
271             elsif (capt_a_i = '1' AND capt_b_i = '0') then
272                 Etat_Futur <= Err;
273             else
274                 Etat_Futur <= E01_AH_W;
275             end if;

```

```

276
277         when E11_AH =>
278             det_err_o <= '0';
279             dir_cw_o <= '0';
280             en_o <= '1';
281             Etat_Futur <= E11_AH_w;
282
283             if (capt_a_i = '0' AND capt_b_i = '0') then
284                 Etat_Futur <= Err;
285             elsif (capt_a_i = '0' AND capt_b_i = '1') then
286                 Etat_Futur <= E01_H;
287             elsif (capt_a_i = '1' AND capt_b_i = '0') then
288                 Etat_Futur <= E10_AH;
289             end if;
290
291         when E11_AH_w =>
292             det_err_o <= '0';
293             dir_cw_o <= '0';
294             en_o <= '0';
295
296             if (capt_a_i = '0' AND capt_b_i = '0') then
297                 Etat_Futur <= Err;
298             elsif (capt_a_i = '0' AND capt_b_i = '1') then
299                 Etat_Futur <= E01_H;
300             elsif (capt_a_i = '1' AND capt_b_i = '0') then
301                 Etat_Futur <= E10_AH;
302             else
303                 Etat_Futur <= E11_AH_w;
304             end if;
305
306         when E10_AH =>
307             det_err_o <= '0';
308             dir_cw_o <= '0';
309             en_o <= '1';
310             Etat_Futur <= E10_AH_w;
311
312             if (capt_a_i = '0' AND capt_b_i = '0') then
313                 Etat_Futur <= E00_AH;
314             elsif (capt_a_i = '0' AND capt_b_i = '1') then
315                 Etat_Futur <= Err;
316             elsif (capt_a_i = '1' AND capt_b_i = '1') then
317                 Etat_Futur <= E11_H;
318             end if;

```

```

320           when E10_AH_W =>
321               det_err_o <= '0';
322               dir_cw_o <= '0';
323               en_o <= '0';
324
325               if (capt_a_i = '0' AND capt_b_i = '0') then
326                   Etat_Futur <= E00_AH;
327               elsif (capt_a_i = '0' AND capt_b_i = '1') then
328                   Etat_Futur <= Err;
329               elsif (capt_a_i = '1' AND capt_b_i = '1') then
330                   Etat_Futur <= E11_H;
331               else
332                   Etat_Futur <= E10_AH_W;
333               end if;
334
335
336           when Err =>
337               det_err_o <= '1';
338               dir_cw_o <= '0';
339               en_o <= '0';
340               if (capt_a_i = '0' AND capt_b_i = '0') then
341                   Etat_Futur <= E00_H_W;
342               elsif (capt_a_i = '0' AND capt_b_i = '1') then
343                   Etat_Futur <= E01_H_W;
344               elsif (capt_a_i = '1' AND capt_b_i = '1') then
345                   Etat_Futur <= E11_H_W;
346               else -- (capt_a_i = '1' AND capt_b_i = '0')
347                   Etat_Futur <= E10_H_W;
348               end if;
349
350
351           when others =>
352               Etat_Futur <= Start;
353               det_err_o <= '0';
354               dir_cw_o <= '0';
355               en_o <= '0';
356
357       end case;
358   end process;
---
```

```

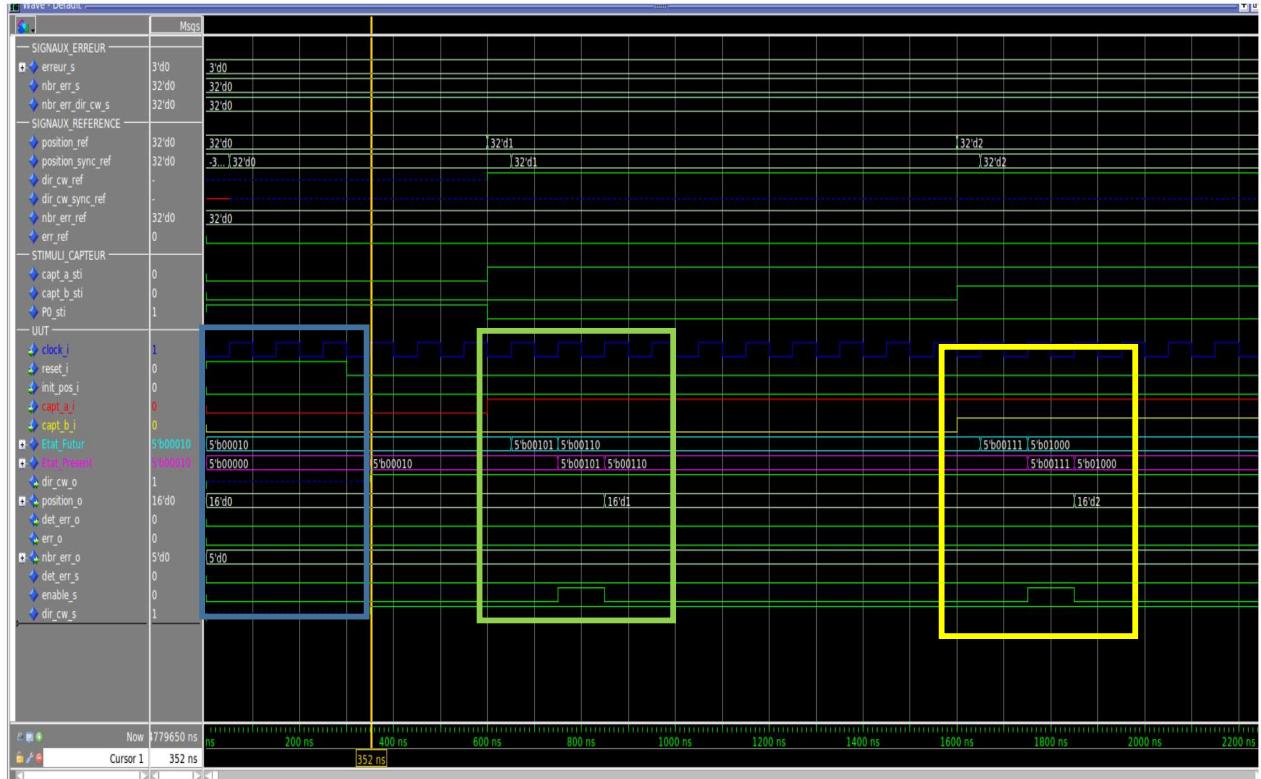
359
360      -- Reset ou met à jour l'état présent
361      Mem: process (clk_i, reset_i)
362      begin
363          if (reset_i = '1') then
364              Etat_Present <= Start;
365          elsif rising_edge(clk_i) then
366              Etat_Present <= Etat_Futur;
367          end if;
368      end process;
369
370
371
372
373  end struct;

```

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.

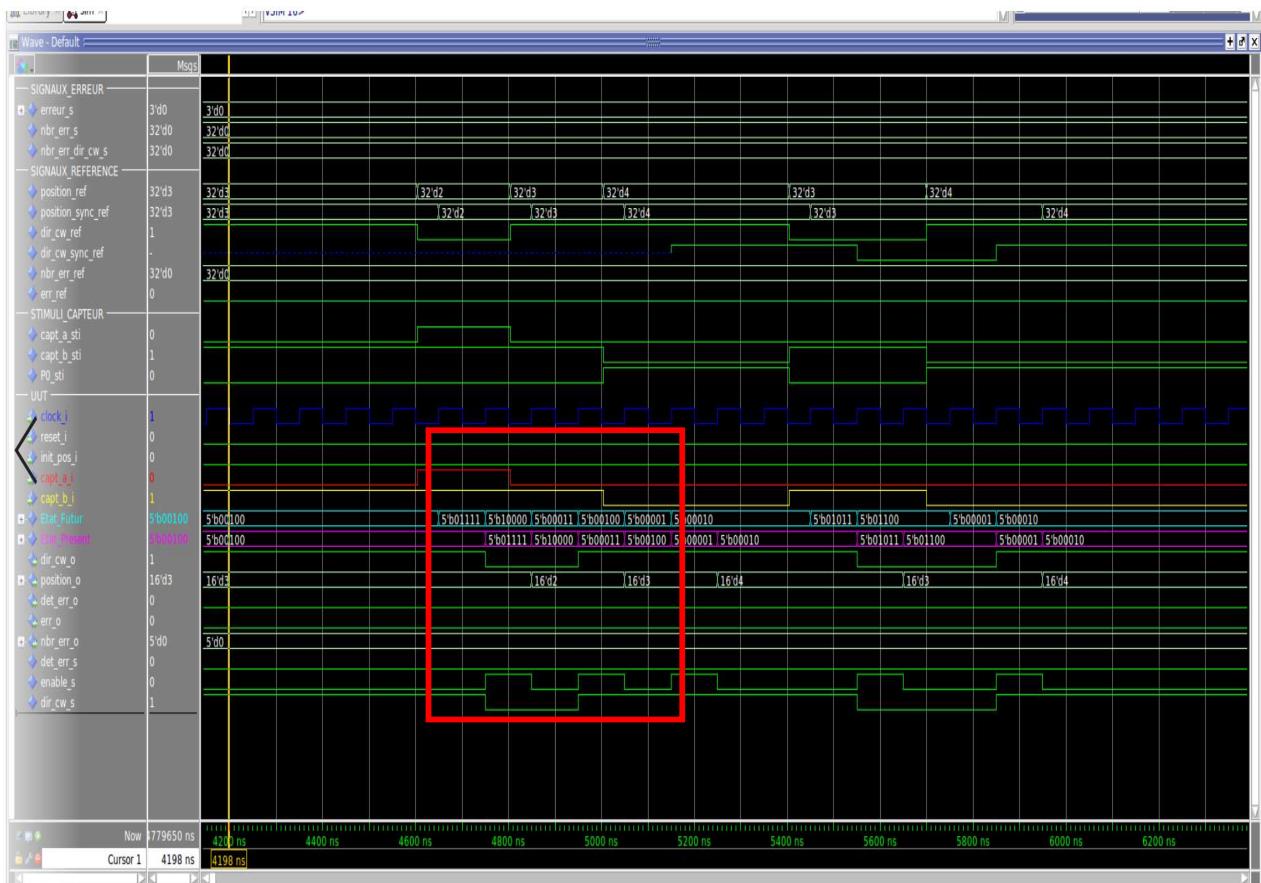


Le cadre bleu sur notre chronogramme permet de montrer qu'à l'état_present = «00000 » => start , et qu'en entrée on a « capt_a = 0, capt_b= 0 », l'état futur est « 00010 » qui correspond bien à notre état E00_H_w avec une direction horaire.

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.

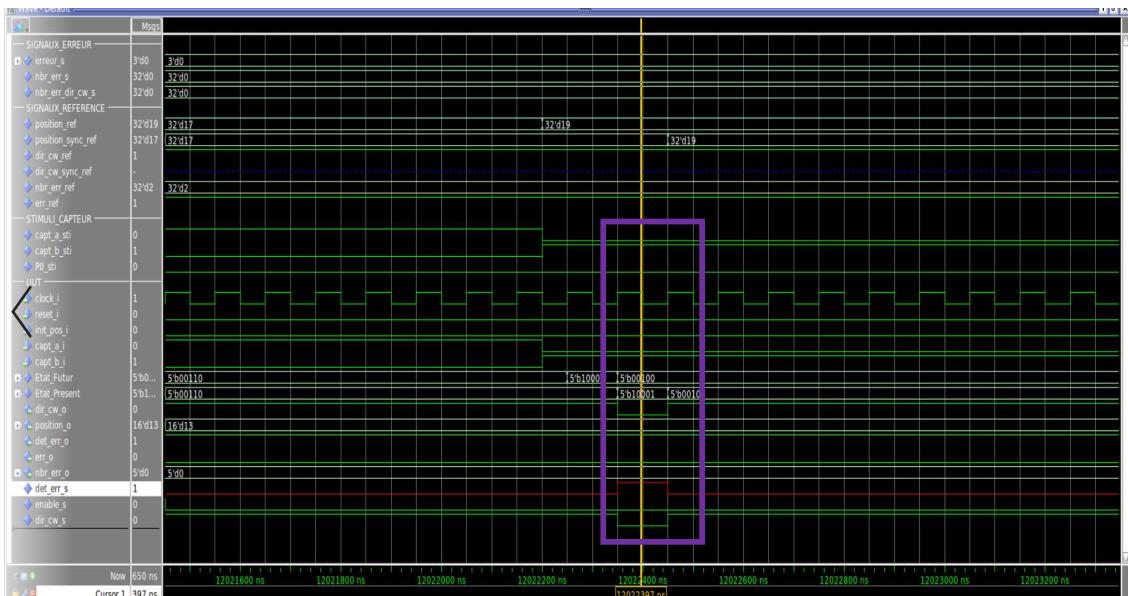
On peut aussi voir que le « enable est actif », l'état présent de notre système passe à l'état « wait ».



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) entraîne 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 précédent 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) commandée 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.

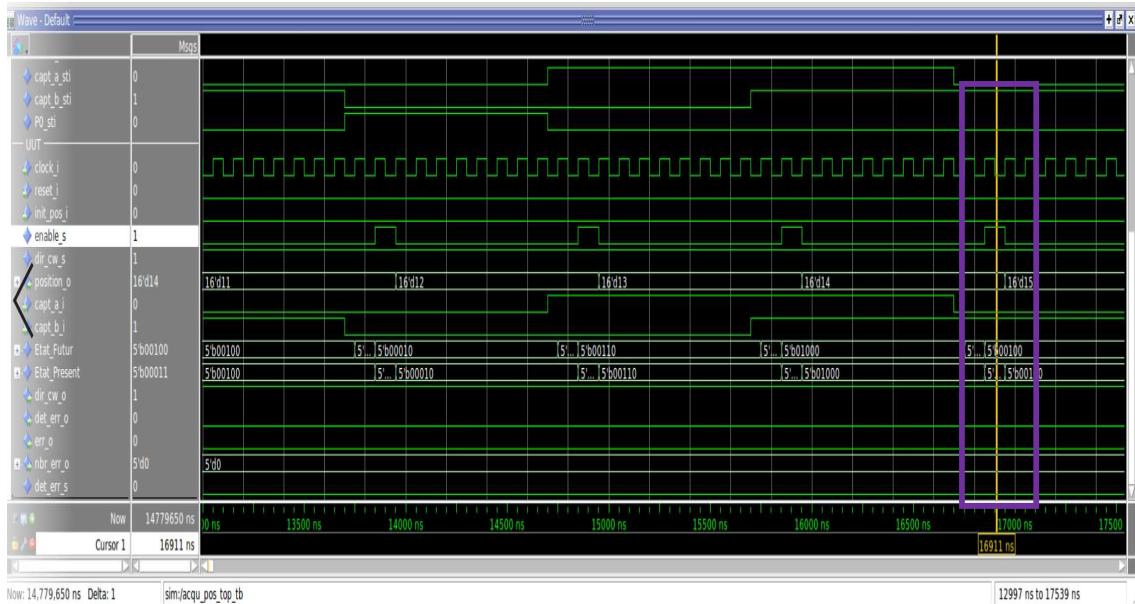
Description VHDL

```

14  library ieee;
15  use ieee.std_logic_1164.all;
16  use ieee.numeric_std.all;
17
18  entity compteur_position is
19      port(reset_i      : in  std_logic;
20            clk_i       : in  std_logic;
21            en_i        : in  std_logic;
22            init_pos_i   : in  std_logic;
23            dir_cw_i    : in  std_logic;
24            position_o  : out  std_logic_vector(15 downto 0)
25        );
26  end compteur_position;
27
28  architecture struct of compteur_position is
29
30      --components declaration
31
32
33      --declaration internal signals
34      signal position_s           : unsigned(15 downto 0) := (others => '0');
35      signal position_next_s      : unsigned(15 downto 0) := (others => '0');
36
37  begin
38      -- Définit la position futur
39      position_next_s <= (others => '0') when init_pos_i = '1' else
40                               position_s when en_i = '0' else
41                               (position_s + 1) when dir_cw_i = '1' else
42                               (position_s - 1) ;
43
44      -- Reset ou met à jour la position
45      process(clk_i, reset_i)
46          begin
47              if(reset_i = '1') then
48                  position_s <= (others => '0');
49              elsif (rising_edge(clk_i)) then
50                  position_s <= position_next_s;
51              end if;
52          end process;
53
54      -- Met à jour la position
55      position_o <= std_logic_vector(position_s(15 downto 0));
56
57
58  end struct;
```

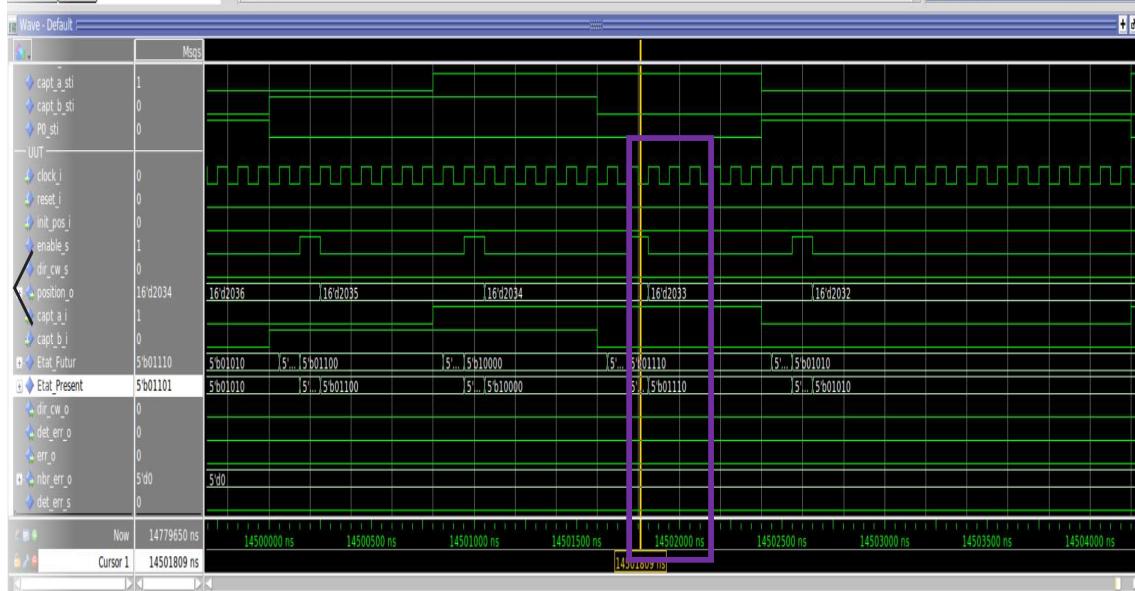
Vérification du fonctionnement

| Init_pos_i | En_i | dir_cw_i | Cpt_pres | Cpt_fut | Fonctions |
|-------------------|-------------|-----------------|-----------------|----------------|-------------------------------|
| 0 | 1 | 1 | - | = Cpt_pres+1 | Incrémentation de la position |

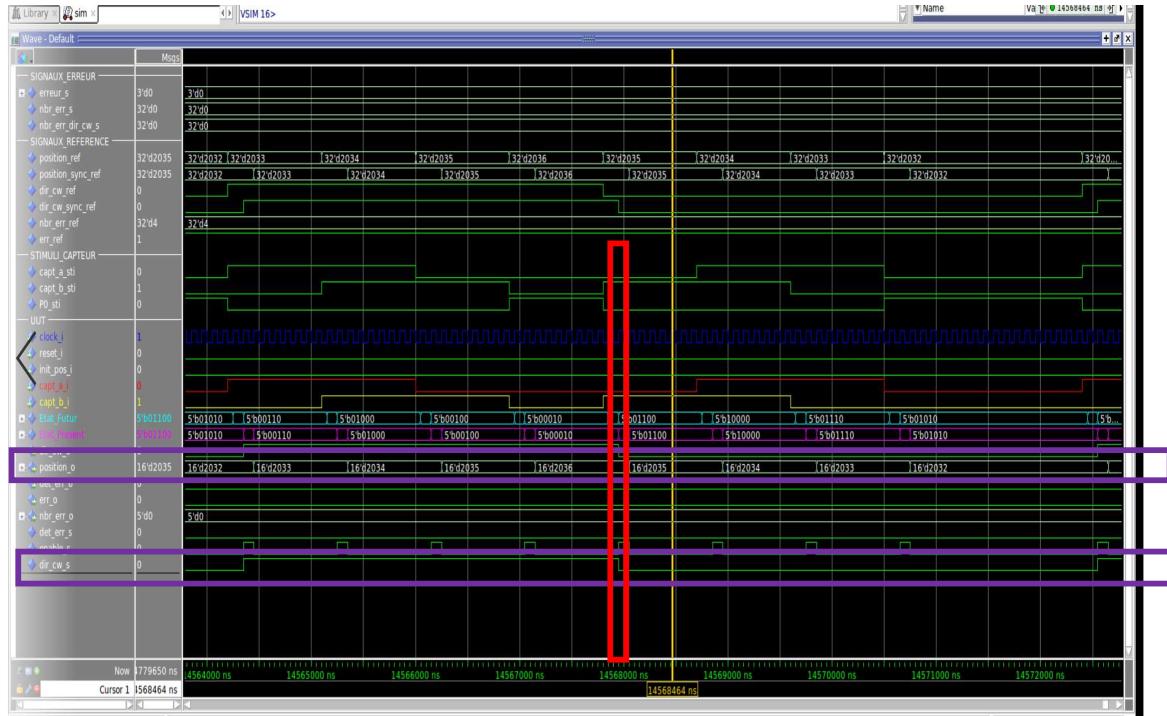


Cette capture du chronogramme de notre simulation nous montre que le compteur incrémente la position comme indiqué dans notre table de fonctionnement de notre compteur.

| 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 actif. 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

```

25  library ieee;
26    use ieee.std_logic_1164.all;
27
28  entity flipflop_rs is
29    port(clk_i      : in    std_logic;
30          reset_i   : in    std_logic;
31          R_i       : in    std_logic;
32          S_i       : in    std_logic;
33          Q_o       : out   std_logic
34      );
35  end flipflop_rs ;
36
37
38  architecture comport of flipflop_rs is
39    signal Q_pres :std_logic := '0';
40    signal Q_fut :std_logic;
41
42  begin
43    --Adaptation polarité
44
45    Q_fut <= Q_pres when R_i = '0' and S_i = '0' else
46      '0' when R_i = '1' else
47      '1' ;
48
49
50    process(reset_i, clk_i)
51    -- zone de déclaration
52
53    begin
54      -- zone des instructions séquentielles
55      -- valeur par défaut des sorties
56
57      if (reset_i = '1') then
58        Q_pres <= '0';
59      elsif (rising_edge(clk_i)) then
60        Q_pres <= Q_fut;
61      end if;
62
63    end process;

```

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 précédent 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 complété 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

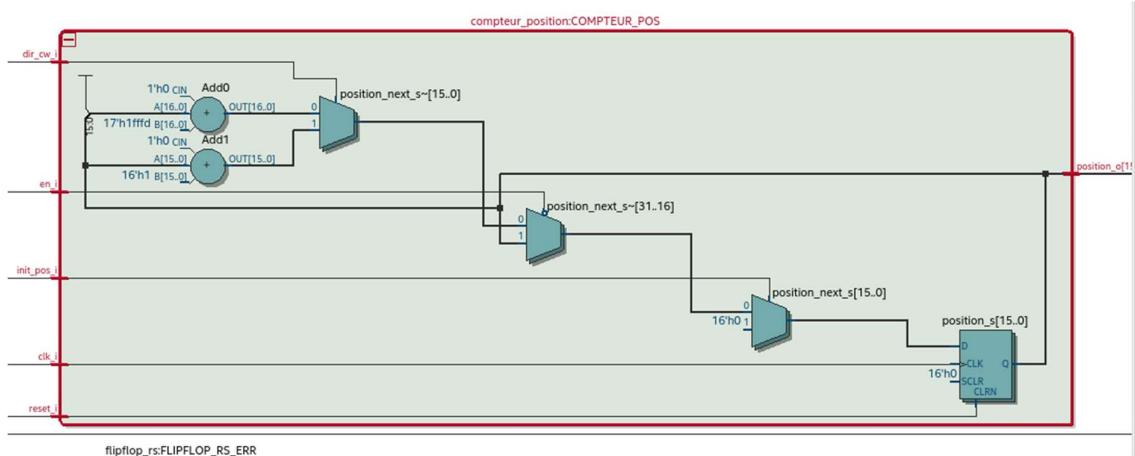
Ensuite, nous avons pu faire la synthèse de notre solution afin de déterminer la quantité de logique utilisé.

| Analysis & Synthesis Resource Utilization by Entity | | | | |
|---|-------------------------------|-------------|--------------|----|
| | Compilation Hierarchy Node | Logic Cells | LC Registers | UI |
| 1 | maxv_top | 115 (20) | 44 | 0 |
| 1 | acqu_pos_top:U1 | 95 (2) | 24 | 0 |
| 1 | compteur_posi...:COMPTEUR_POS | 49 (49) | 16 | 0 |
| 2 | flipflop_rs:FLIPFLOP_RS_ERR | 2 (2) | 1 | 0 |
| 3 | mss:MSS1 | 42 (42) | 5 | 0 |

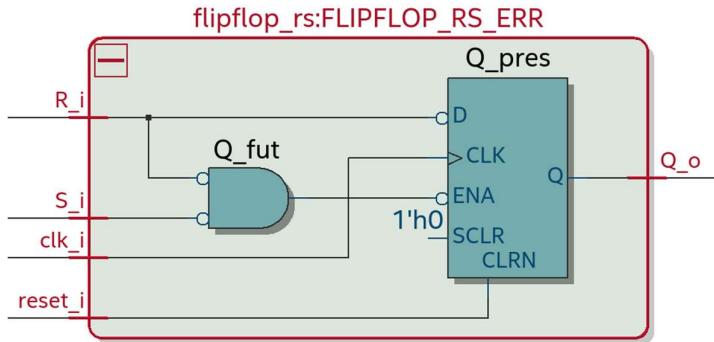
On peut voir que sans le maxv_top, notre solution utilise 95 quantités de logique. Afin d'y voir plus claire, nous avons décidé d'analyser les vues RTLs.

Compteur de position

Le compteur de position utilise 49 quantités logiques :



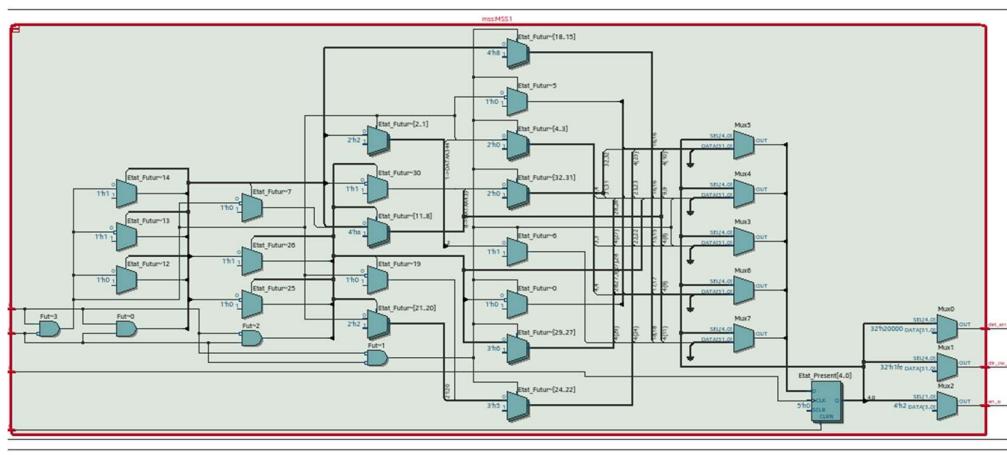
Cela se comprend après avoir observé la vue RTL : 16 -> additionneur / 17 -> soustracteur / 16 flip-flop de mémorisation de la position.

Bascule RS

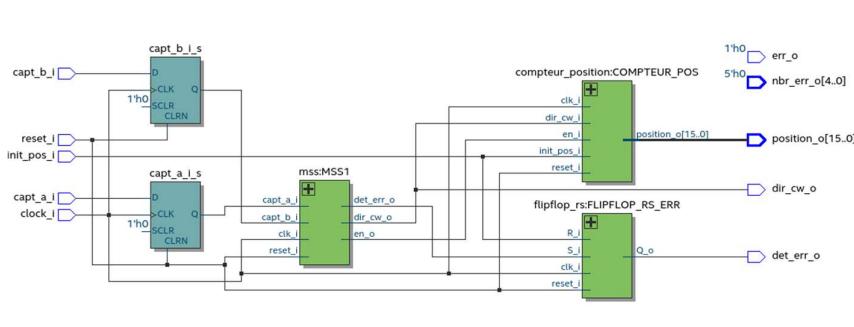
La bascule RS utilise 2 quantité logiques, 1 simplemnet pour la logique combinatoire et l'autre pour le flip-flop de mémorisation de l'état.

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 :

```
# Errors: 0, Warnings: 4
# End time: 17:12:55 on Dec 12,2019, Elapsed time: 0:00:44
# Errors: 0, Warnings: 5
# vsim -novopt work.acqu_pos_top_tb
# Start time: 17:12:55 on Dec 12,2019
# ** Warning: (vsim-8891) All optimizations are turned off because the -novopt switch is in effect. This will cau
se your simulation to run very slowly. If you are using this switch to preserve visibility for Debug or PLI featu
res please see the User's Manual section on Preserving Object Visibility with vopt.
# Loading std.standard
# Refreshing /cours_RED5/CSN/SpinelliIsaia/Labo_7_acqui_pos/acqu_pos/comp/work.acqu_pos_top_tb(testbench)
# Loading std.textio(body)
# Loading ieee.std_logic_1164(body)
# Loading ieee.numeric_std(body)
# Loading ieee.math_real(body)
# Loading work.acqu_pos_top_tb(testbench)
# Refreshing /cours_RED5/CSN/SpinelliIsaia/Labo_7_acqui_pos/acqu_pos/comp/work.acqu_pos_top(struct)
# Loading work.acqu_pos_top(struct)
# Refreshing /cours_RED5/CSN/SpinelliIsaia/Labo_7_acqui_pos/acqu_pos/comp/work.mss(struct)
# Loading work.mss(struct)
# Refreshing /cours_RED5/CSN/SpinelliIsaia/Labo_7_acqui_pos/acqu_pos/comp/work.compteur_position(struct)
# Loading work.compteur_position(struct)
# Refreshing /cours_RED5/CSN/SpinelliIsaia/Labo_7_acqui_pos/acqu_pos/comp/work.flipflop_rs(comport)
# Loading work.flipflop_rs(comport)
# ****
# 0 ns: START: SIMULATION
# *****
```

```

# *****
# 600 ns: Determine le nombre de cycles de latence
# *****
# 4604 ns: Determine le nombre de cycles de latence dir_cw_obs 1 -> 0
# 4696 ns: Nb de cycle de latence du systeme sur dir_cw_obs: 1
# 4704 ns: Determine le nombre de cycles de latence dir_cw_obs 0 -> 1
# 4796 ns: Nb de cycle de latence du systeme sur dir_cw_obs: 1
# 5000 ns: Nb de cycle de latence de position_obs lors de l'incrementation : 2
# 5300 ns: Nb de cycle de latence de position_obs lors de la decrementation : 2
# *****
# 5300 ns: START: Verification des compteurs de position et d'index lors d'un déplacement horaire
# *****
# 6005300 ns: RESULTAT: Verification des compteurs de position et d'index lors d'un déplacement horaire >> Nombre total d'erreurs detectees = 0 <<
# *****
# 6005300 ns: START: Verification des compteurs de position et d'index lors d'un déplacement antihoraire
# *****
# 12005300 ns: RESULTAT: Verification des compteurs de position et d'index lors d'un déplacement antihoraire >> Nombre total d'erreurs detectees = 0 <<
# *****
# 12005300 ns: START: Verification Detection des Erreurs
# *****
# 12039900 ns: RESULTAT: Verification Detection des Erreurs >> Nombre total d'erreurs detectees = 0 <<
# *****
# 12039900 ns: START: Verification des compteurs de position lors d'oscillations autour de l'index
# *****
# ** Note: Valeur position_obs =2032, attendu:2032
#   Time: 14073900 ns Iteration: 0 Instance: /acqu_pos_top_tb
# 14073900 ns: TEST avec Temps par position = 100 ns <<
# 14094900 ns: TEST avec Temps par position = 200 ns <<
# 14126900 ns: TEST avec Temps par position = 300 ns <<
# 14169900 ns: TEST avec Temps par position = 400 ns <<
# 14223900 ns: TEST avec Temps par position = 500 ns <<
# 14288900 ns: TEST avec Temps par position = 600 ns <<
# 14364900 ns: TEST avec Temps par position = 700 ns <<
# 14451900 ns: TEST avec Temps par position = 800 ns <<
# 14549900 ns: TEST avec Temps par position = 900 ns <<
# 14658900 ns: TEST avec Temps par position = 1000 ns <<
# 14778900 ns: RESULTAT: Verification des compteurs de position et d'index lors d'oscillations autour de l'index
>> Nombre total d'erreurs detectees = 0 <<
# *****
# 14778900 ns: DONE: SIMULATION
# *****
# ** Note: >> Nombre total d'erreurs detectees = 0 <<
#   Time: 14778900 ns Iteration: 1 Instance: /acqu_pos_top_tb
# 0 ns
# 15517000 --
```

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 sommes particulièrement fier.

Date : 18.12.19

Nom des étudiants : Lankeu Ngassam Cédric et Spinelli Isaïa