M1 Info – ARC - LAB6

Olivier HUREAU - Groupe 3

21/04/2020

Table des matières

0.1 Conclusion . . . 3

1 Simulation **4**

2 Assertion **6**

3 Synthèse du compteur **7**
3.1 Compteur non limité . . . 7
3.2 Compteur avec valeur borné . . . 7

4 Annexes **7**
4.1 Descprition comportementale du système de contrôle . . . 7
4.1.1 Premier process . . . 7
4.1.1.1 Code VHDL . . . 7
4.1.1.2 Bloc combinatoire ou mémorisant . . . 8
4.1.1.3 Liste de sensibilité . . . 8
4.1.1.4 Composants utilisé . . . 8
4.1.2 Second process . . . 9
4.1.2.1 Code VHDL . . . 9
4.1.2.2 Bloc combinatoire ou mémorisant . . . 9
4.1.2.3 Liste de sensibilité . . . 9
4.1.2.4 Composants utilisé . . . 9
4.1.3 Output process . . . 10
4.1.3.1 Code VHDL . . . 10
4.1.3.2 Bloc combinatoire ou mémorisant . . . 10
4.1.3.3 Liste de sensibilité . . . 10
4.1.3.4 Composants utilisé . . . 10
4.2 Descprition comportementale du compteur . . . 11
4.2.1 Premier process . . . 11
4.2.1.1 Code VHDL . . . 11
4.2.1.2 Bloc combinatoire ou mémorisant . . . 11
4.2.1.3 Liste de senssibilité . . . 11
4.2.1.4 Composants utilisé . . . 11
4.2.2 Deuxième process . . . 12
4.2.2.1 Code VHDL . . . 12
4.2.2.2 Bloc combinatoire ou mémorisant . . . 12
4.2.2.3 Liste de senssibilité . . . 12
4.2.2.4 Composants utilisé . . . 12
4.2.3 Troisième process . . . 13
4.2.3.1 Code VHDL . . . 13
4.2.3.2 Bloc combinatoire ou mémorisant . . . 13
4.2.3.3 Liste de senssibilité . . . 13
4.2.3.4 Composants utilisé . . . 13
4.2.4 Quatrième process . . . 14
4.2.4.1 Code VHDL . . . 14
4.2.4.2 Bloc combinatoire ou mémorisant . . . 14
4.2.4.3 Liste de sensibilité . . . 14
4.2.4.4 Composants utilisé . . . 14
4.3 Synthèse du contrôleur . . . 15
4.3.1 Encodage binaire . . . 15

4.3.1.1 Codage des états . . . 15
4.3.1.2 Report Area . . . 15
4.3.1.3 Report Delay . . . 15
4.3.2 Encodage de Gray . . . 16
4.3.2.1 Codage des états . . . 16
4.3.2.2 Report Area . . . 16
4.3.2.3 Report Delay . . . 16
4.3.3 Encodage OneHot . . . 17
4.3.3.1 Codage des états . . . 17
4.3.3.2 Report Area . . . 17
4.3.3.3 Report Delay . . . 17
4.4 Synthèse du compteur . . . 18
4.4.1 Sans borner C . . . 18
4.4.1.1 Report Area . . . 18
4.4.1.2 Report Delay . . . 18
4.4.2 Avec C borné . . . 18
4.4.2.1 Report Area . . . 18
4.4.2.2 Report Delay . . . 18

0.1 Conclusion

La différence avec un codage de gray ou binaire n'est pas très significative. On as la même nombre de composants mémorisants mais le chemin critique est plus long pour le codage binaire.

Cependant avec un codage OneHot la fréquence est moitié plus grande qu'en binaire ou gray. La surface est un quart plus grande car utilise plus de composants mémorisant (DFC1 et DFP1)

On pouvais s'y attendre car le oneHot nécessite moins d'utilisation de composant mémorisant et c'est ceux la qui prennent le plus de temps.
En effet, plus le délai du chemin critique est faible, plus il sera possible de passer par celui la en une seconde. Donc la fréquence augmente.

1 Simulation

Pour analyser les différentes synthèse et le composant inital on vas alors regarder si la valeur des états est la même pour tout les composants et si les sorties sont toutes équivalentes.
les sorties devant être équivalentes sont alors : search, rest et food.
Ce qu'on vérifie ici :

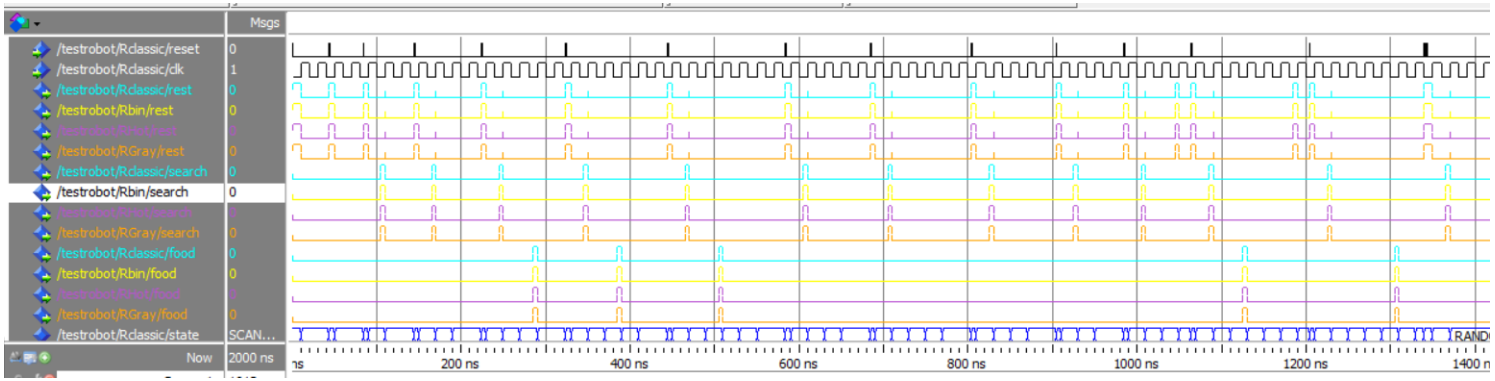

Figure 1: Chronograme avec les différentes valeurs de sorties

On vérifie en suite pour les états. On rappel les valeurs associés à chaque état.

Etats	Binaire	Gray	OneHot
IDLE	0000	0000	000000010
RESTING	-001	0001	000000100
RANDOMWALK	-010	0011	000000100
SCANAREA	-011	0010	000001000
HOMING	-100	0110	000010000
MOVETOFOOD	-101	0111	000100000
MOVETOHOME	-110	0101	001000000
DEPOSIT	-111	0100	010000000
GRABFOOD	1000	1100	100000000

Il est asses simple de retrouver ces valeurs pour la synthèse en Gray et la synthèse en Binaire car il existe les signaux correspondant : state_0; state_1; state_2 ; state_3.
Néanmoins pour la synthèse en OneHot il nous manque les signaux state_6 et state_8. Cela dois surement être du à une optimisation est que celui-ci n'était pas nécessaire.
Cependant si l'on regarde le code VHDL de la synthèse. Chaque signaux correspond à un process.
Par exemple :

```
reg_state_2 : DFC1 port map ( Q=>state_2, QN=>OPEN, C=>clk, D=>nx64, RN=>nx759);
```

Le signal state_2 est modifié à la sortie du flip flop dans le process reg_state_2.
Si on s'intéresse au process reg_state_6 :

```
reg_state_6 : DFC1 port map ( Q=>OPEN, QN=>nx767, C=>clk, D=>nx110, RN=>
```

Rien n'est branché sur Q du flip flop mais on peut récupérer la valeur de state_6 car c'est l'inverse de QN. On s'intéressera alors ç l'inverse du signal nx767.

Pareil pour state_8 :

```
reg_state_8 : DFC1 port map ( Q=>OPEN, QN=>nx772, C=>clk, D=>nx102, RN=>
```

Donc on surveillera nx772.
On vérifie alors que les valeurs des différents signaux coincident bien avec les états courants sur le chronogramme suivant :

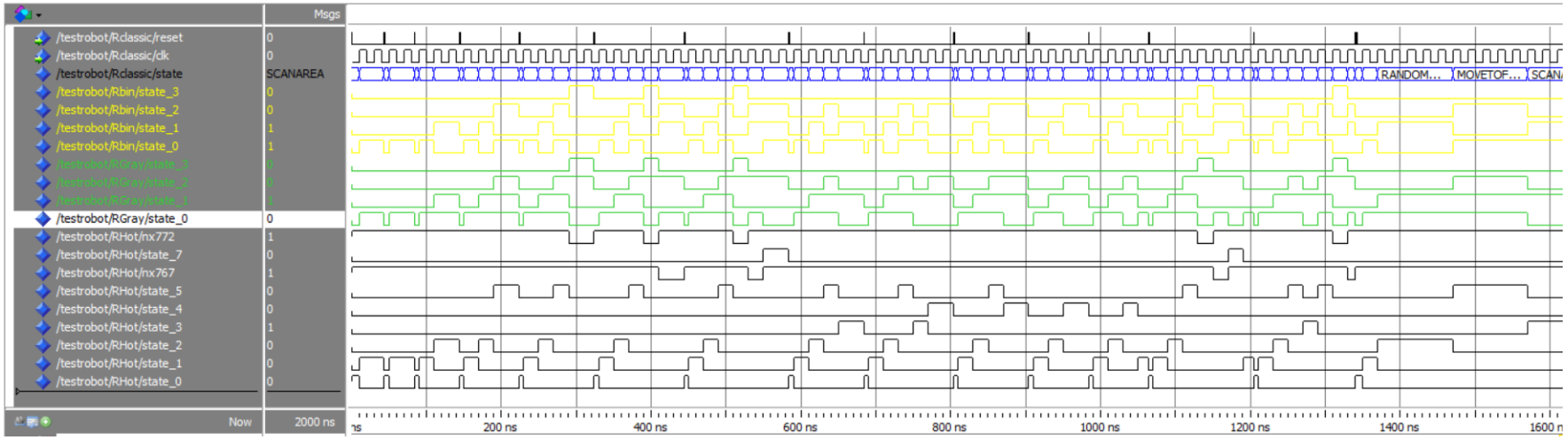

Figure 2: Chronograme avec les signaux d'états

Ce qui correspond bien avec ce qui était attendu.

En faisant un montage on peux alors vérifier plus simplement toutes les valeurs pour chaque états.

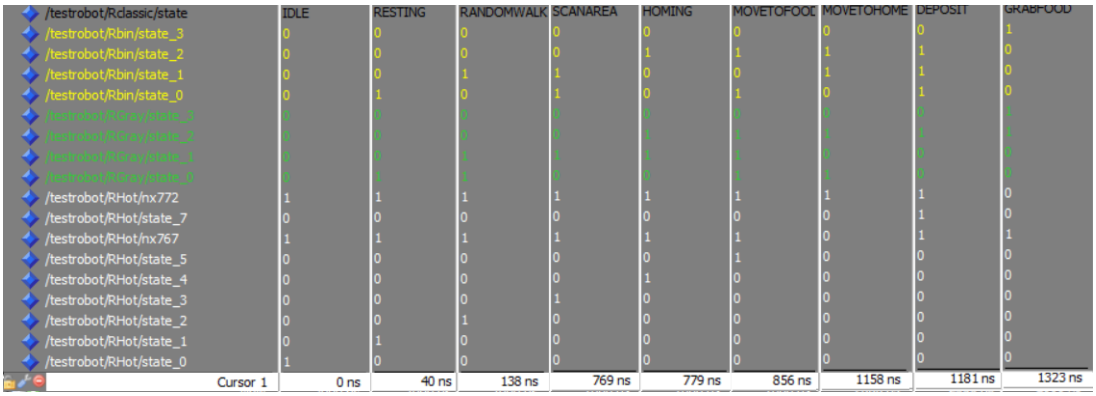

Figure 3: Montage du chronograme avec les différentes valeurs de sorties
Il est aussi possible de créer des propriété PSL verifiant les valeurs des signaux des synthèses en fonction de la valeur de l'état sur le système non synthétisé.

2 Assertion

Pour vérifier que les assertions temporelles définies au TP précédent restent satisfaite après synthèse il faut réecrire les assertions dans les fichiers de synthèse et remplacer les endroits où l'on test la valeur d'un état par son identité en encodage. Voix annexes.
Pour vérifier que les nouvelles assertions soient juste il suffit de les regrouper et de vérifier qu'elles sont identique comme suit :

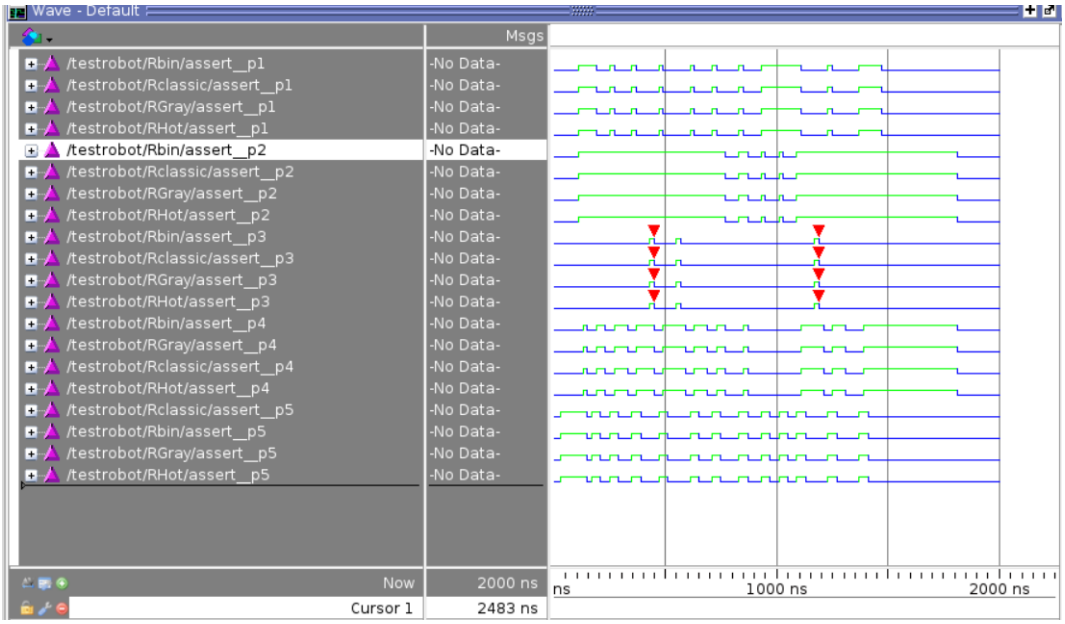

Figure 4: Assertions P1, P2, P3, P4 et P5 sur le chronogramme

Pour vérifier que les sorties des architectures soient toujours identique j'ai crée trois assertions qui vérifie ceux-ci grace à une formule booléenne :

```
-- psl property RestAlwaysTheSame is always ( ((re = '0' or reBin = '0' or reGray = '0' or reHot = '0')
-- AND (re = '0' and reBin = '0' and reGray = '0' and reHot = '0' and testPSL = '0'))
-- OR  ((re = '1' or reBin = '1' or reGray = '1' or reHot = '1') AND (re = '1' and reBin = '1' and reGr
-- psl assert RestAlwaysTheSame;
```

Aucune assertion échoue donc sela semble fonctionner. Pour vérifier on rajoute un signal qui est toujours égale à 1 dans une des clauses et on obtiens :
Les formules sont donc juste.

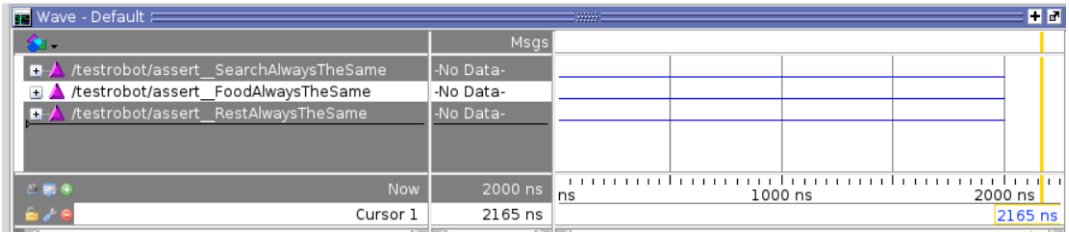

Figure 5: Assertion verifiant l'égalité des sorties

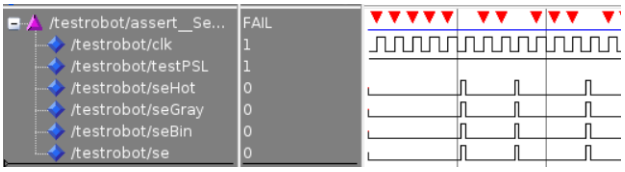

Figure 6: Test de l'assertion verifiant l'égalité des sorties

3 Synthèse du compteur

3.1 Compteur non borné

D'après les résultats obtenus après synthèse (voir annexes). Il y a 31 DFC1 et 1 DFCP1 soit un total de 32 flip flops. Il y a en tout 32 flips flops parce que notre signal peut être codé sur 32 bits. Il faut alors de quoi stocker ses 32 bits.
La surface obtenue est 17963 um2
La fréquence obtenue est 155.2 MHz

3.2 Compteur avec valeur borné

D'après les résultats obtenus après synthèse (voir annexes). Il y a 4 DFC1 et 1 DFCP1 soit un total de 5 flip flops. Il y a en tout 5 flips ce qui parrait étrange car $log_2(10) = 3.3$, 4 flips flops aurais suffit.
La surface obtenue est 2639 um2
La fréquence obtenue est 685.2 MHz

3.3 Conclusion

Nous travaillons ici dans un système où chaque choix de design aura un impact sur la synthèse du système : la surface a était réduit plus de 6 fois est la fréquence à plus que quadruplé. Il faut donc bien faire attention car l'erreur est beaucoup plus vite punitive habituellement. Il font donc faire preuve de rigueur.

4 Annexes

4.1 Descprition comportementale du système de contrôle

4.1.1 Premier process

4.1.1.1 Code VHDL

```
process (state, athome, findfood, lostfood, closetofood, success,
 aboverestth, abovesearchth, scantimeup)
begin
case state is
when IDLE => nextstate <= RESTING;
when RESTING =>
if aboverestth = '1' then nextstate <= RANDOMWALK;
else--elsif aboverestth = '0' then
nextstate <= RESTING;
end if;

when RANDOMWALK =>
if abovesearchth = '1' then nextstate <= HOMING;
else--  abovesearchth = '0' then
if findfood = '1' then nextstate <= MOVETOFOOD;
else--elsif findfood = '0' then
nextstate <= RANDOMWALK;
end if;
end if;

when SCANAREA =>
if abovesearchth = '1' then nextstate <= HOMING;
else--elsif  abovesearchth = '0' then
if findfood = '1' then nextstate <= MOVETOFOOD;
else--elsif findfood = '0' then
if scantimeup = '1' then nextstate <= RANDOMWALK;
else--elsif scantimeup = '0' then
nextstate <= SCANAREA;
end if;
end if;
end if;
when HOMING => if(athome = '1') then nextstate <= RESTING; else nextstate <= HOMING; end if;
when MOVETOFOOD =>
if abovesearchth = '1' then nextstate <= HOMING;
else--elsif  abovesearchth = '0' then
if lostfood = '1' then nextstate <= SCANAREA;
else--elsif lostfood = '0' then
if closetofood = '1' then nextstate <= GRABFOOD;
else--elsif closetofood = '0' then
nextstate <= MOVETOFOOD;
end if;
end if;
end if;
when GRABFOOD =>
if success = '1' then nextstate <= MOVETOHOME;
else--elsif success = '0' then
 nextstate <= GRABFOOD;
end if;
when MOVETOHOME =>
if athome = '1' then nextstate <= DEPOSIT;
else--elsif athome = '0' then
 nextstate <= MOVETOHOME;
end if;
when DEPOSIT =>
if success = '1' then nextstate <= RESTING;
```

```
else--elsif success = '0' then
 nextstate <= DEPOSIT;
end if;
end case;
end process;
```

4.1.1.2 Bloc combinatoire ou mémorisant

"The VHDL process that characterizes the next state generation corresponds to the transition function combinational block:"
Alors c'est un bloc combinatoire

4.1.1.3 Liste de sensibilité

Tout ce qui peux alors modifier la valeur du prochain état Soit tout ce qui se trouve dans les branchements.

4.1.1.4 Composants utilisé

Ce sera alors des composants combinatoires qui seront utilisé

4.1.2 Second process

4.1.2.1 Code VHDL

```
process(reset, clk)
begin
-- RESET : asynchrone haut
if reset = '1' then state <= IDLE;
-- HORLOGE : front montant
elsif (clk'event and clk = '1') then
state <= nextstate;
end if;
end process;
```

4.1.2.2 Bloc combinatoire ou mémorisant

"And the "clocked process" expresses the updating of the flip-flops (registers)"
C'est un bloc mémorisant

4.1.2.3 Liste de sensibilité

L'horloge et le reset

4.1.2.4 Composants utilisé

Des composants mémorisants

4.1.3 Output process

Mon process d'output est en fait plusieurs affectations représentant les process. Cela permet une optimisation car la liste de senssibilité est réduite.

4.1.3.1 Code VHDL

```
rest <= '1' when (( state = DEPOSIT and success = '1' ) OR (state = IDLE) OR (state = HOMING and
     athome = '1') ) else '0';
search <= '1' when (state = RESTING and aboverestth = '1' ) else '0';
food <= '1' when  (state = MOVETOFOOD and abovesearchth = '0' and lostfood = '0' and closetofood ='1')
    else '0';
```

4.1.3.2 Bloc combinatoire ou mémorisant

"The VHDL process that characterizes the output generation corresponds to the output function combinational block:"
Bloc combinatoire

4.1.3.3 Liste de sensibilité

Chaque outputs possède sa propre liste de sensibilité qui est les différents items utilisé dans la formule.

4.1.3.4 Composants utilisé

Ce sera alors des composants combinatoires qui seront utilisé

4.2 Descprition comportementale du compteur

J'ai modifié le compteur pour qu'il ne soit plus générique tel que

```
Signal threshold : natural = 4;
```

4.2.1 Premier process

4.2.1.1 Code VHDL

process (state, start, c) begin case state is when IDLE =¿ if start = '1' then nextstate ¡= COUNTING; elsif start = '0' then nextstate ¡= IDLE; end if; when COUNTING =¿ if c ¡ threshold then nextstate ¡= COUNTING; else nextstate ¡= IDLE; end if; end case; end process;

4.2.1.2 Bloc combinatoire ou mémorisant

Combinatoire

4.2.1.3 Liste de senssibilité

state, start, c

4.2.1.4 Composants utilisé

Combinatoire

4.2.2 Deuxième process

4.2.2.1 Code VHDL

process(reset, clk, start) begin – RESET : asynchrone haut if reset = '1' then state ¡= IDLE; – HORLOGE : front montant elsif ((start = '1') and (state = IDLE))then state ¡= COUNTING;
elsif (clk'event and clk = '1') then state ¡= nextstate; – Detecter un pic sur start
end if;
end process;

4.2.2.2 Bloc combinatoire ou mémorisant

Mémorisant

4.2.2.3 Liste de senssibilité

reset, clk, start

4.2.2.4 Composants utilisé

Mémorisant et combinatoire

4.2.3 Troisième process

4.2.3.1 Code VHDL

process(start, clk, reset) begin if(reset = '1') then c ¡= 0;
else if (clk'event and clk = '1') then if (state = IDLE and start = '0') then C ¡= 0; elsif (state = IDLE and start = '1') then c ¡= c + 1; elsif (state = COUNTING and c ¡ threshold) then c ¡= c + 1; elsif(state = COUNTING and c ¿= threshold) then c ¡= 0; end if; end if; end if; end process;

4.2.3.2 Bloc combinatoire ou mémorisant

Mémorisant (valeur du compteur)

4.2.3.3 Liste de senssibilité

start, clk, reset

4.2.3.4 Composants utilisé

Combinatoire et mémorisant

4.2.4 Quatrième process

4.2.4.1 Code VHDL
process(c) begin if (c ¿= threshold) then aboveth ¡= '1'; else aboveth ¡= '0'; end if; end process;

4.2.4.2 Bloc combinatoire ou mémorisant
Combinatoire

4.2.4.3 Liste de sensibilité
Valeur du compteur

4.2.4.4 Composants utilisé
Combinatoires

4.3 Synthèse du contrôleur

4.3.1 Encodage binaire

4.3.1.1 Codage des états

4.3.1.2 Report Area

4.3.1.3 Report Delay

4.3.2 Encodage de Gray

4.3.2.1 Codage des états

4.3.2.2 Report Area

4.3.2.3 Report Delay

4.3.3 Encodage OneHot

4.3.3.1 Codage des états

4.3.3.2 Report Area

4.3.3.3 Report Delay

4.4 Synthèse du compteur

4.4.1 Sans borner C

4.4.1.1 Report Area

4.4.1.2 Report Delay

4.4.2 Avec C borné

4.4.2.1 Report Area

4.4.2.2 Report Delay