

M1 Info – ARC - LAB4

Olivier HUREAU - Groupe 3

06/04/2020

Table des matières

1	Évaluation de la qualité des tests.	2
1.1	Première analyse	2
1.1.1	Résultats du test de coverage	2
1.1.1.1	Résultats du test de coverage sur le système entier	2
1.1.1.2	Résultats du test de coverage sur le robot	3
1.1.1.3	Interprétation des résultats	4
1.2	Correction du code après une première analyse	4
1.2.1	Correction du testbench du robot	4
1.2.1.1	Les branchements	4
1.2.1.2	Les états et les transitions	4
1.2.1.3	Résultat après correction	4
1.2.1.4	Conclusion de la correction du robot	5
1.2.2	Correction du testbench du système	5
1.2.2.1	Résultat après correction	5
1.2.2.2	Interprétation des résultats	6
2	Vérification d'assertions temporelles	7
2.1	P1	7
2.1.1	Formule PSL	7
2.1.2	Analyse du chronogramme	7
2.2	P2	8
2.2.1	Formule PSL	8
2.2.2	Analyse du chronogramme	8
2.3	P3	9
2.3.1	Formule PSL	9
2.3.2	Analyse du chronogramme	9
2.4	P4	10
2.4.1	Formule PSL	10
2.5	P5	12
2.5.1	Formule PSL	13
2.5.2	Analyse du chronogramme	14
2.5.3	Réaliser une simulation dans laquelle la condition attendue ne se produit pas	14
2.5.3.1	Qu'observez-vous ?	14
2.5.3.2	Que se passe-t-il si vous faites alors "End simulation" dans ModelSim ?	14
2.5.3.3	Comment interprétez-vous ce qui se produit ?	14
3	Conclusion	15
4	Réaction après discussion par mail	15
5	Annexes	16
5.1	Différences de code entre la partie 1 et 2 du projet	16
5.2	Le compteur	17
5.3	Architecture du robot	18
5.4	TestBench du robot	19
5.5	TestBench du système	20

1 Évaluation de la qualité des tests.

Les différents types de couverture nous intéressant sont

- State / Transition coverage : En principe les testbench vérifie que la machine passe dans tout les états. Si nous n'avons pas 100% de couverture ici c'est qu'il y a un problème.
- Branch coverage : Si le teste de couverture remonte des erreurs, il sera alors plus facile de traquer les erreurs de transissions..
- Statement coverage : Les statements influant sur les branchements, ce test de couverture nous permet aussi de vérifier le comportement du robot.

Les tests "line coverage" et donne une indication sur la présentation ou l'optimisation du code, ne nous souciaints uniquement du bon fonctionnement de notre implémentation, il ne nous sont pas utiles.

Le test Toggle coverage donnant une indication sur la robustesse du l'implémentation. Il est bien trop compliqué et trop long d'analyser les résultats de ce test dans le cadre de nos labs.

1.1 Première analyse

1.1.1 Résultats du test de coverage

1.1.1.1 Résultats du test de coverage sur le système entier Ci dessous : S est l'architecture System, C1 et C2 les compteurs et R le robots dans le testbench System.











Instance	Branch %	State %	State graph	Transition %	Transition graph
testsystem	74.7%	92.3%		64.1%	
S	74.7%	92.3%		64.1%	
C1	95%	100%		100%	
C2	90%	100%		100%	
R	61.8%	88.9%		54.8%	

Figure 1: Résultats de couverture Branch, State et Transition sur le système entier

Le graph de couverture du robot dans le testbench system

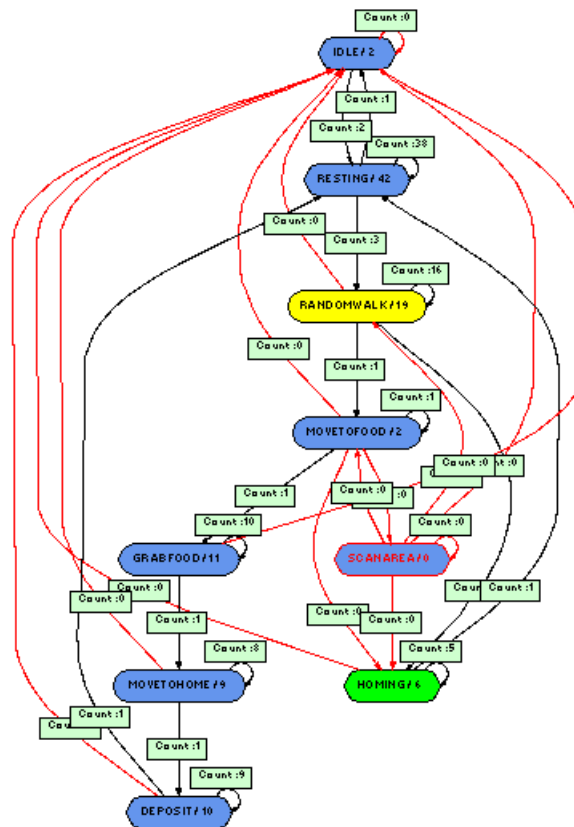


Figure 2: Graph de couverture d'état sur le système entier

1.1.1.2 Résultats du test de couverture sur le robot Ci dessous : S est l'architecture System, C1 et C2 les compteurs et R le robots dans le testbench System.





Instance	Branch %	State %	State graph	Transition %	Transition graph
testrobot	76.4%	100%		90.3%	
A	76.4%	100%		90.3%	

Figure 3: Résultats de couverture Branch, State et Transition sur le robot

Le graph de couverture du robot dans le testbench system

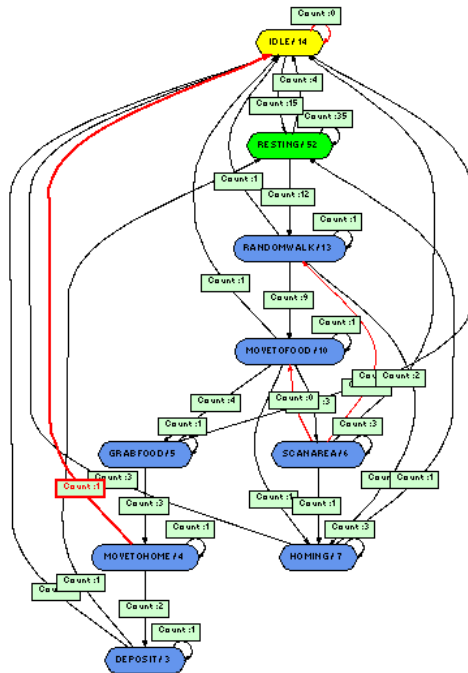


Figure 4: Graph de couverture d'état sur le robot

On observera aussi le code coverage analysis :

```

robot.vhd
23 when IDLE => nextstate <= RESTING;
24 when RESTING =>
25   if aboverestth = '1' then nextstate <= RANDOMWALK;
26   elsif aboverestth = '0' then nextstate <= RESTING;
27   else
31     when RANDOMWALK =>
32       if abovesearchth = '1' then nextstate <= HOMING;
33       elsif abovesearchth = '0' then
34         if findfood = '1' then nextstate <= MOVETOFOOD;
35         elsif findfood = '0' then
40           when SCANAREA =>
41             if abovesearchth = '1' then nextstate <= HOMING;
42             elsif abovesearchth = '0' then
43               if findfood = '1' then nextstate <= MOVETOFOOD;
44               elsif findfood = '0' then
45                 if scantimeup = '1' then nextstate <= RANDOMWALK;
46                 elsif scantimeup = '0' then nextstate <= SCANAREA;
50             when HOMING => if(athome = '1') then nextstate <= RESTING; else nex...
51             when MOVETOFOOD =>
52               if abovesearchth = '1' then nextstate <= HOMING;
53               elsif abovesearchth = '0' then
54                 if lostfood = '1' then nextstate <= SCANAREA;
55                 elsif lostfood = '0' then
56                 if closetofood = '1' then nextstate <= GRABFOOD;

```

Figure 5: Graph de couverture d'état sur le robot

1.1.1.3 Interprétation des résultats Les tests de couverture sont très probant (78.5% pour le système entier). Cependant, on voit que dans le système entier, l'état SCANAREA n'est jamais atteint bien que les tests de système (se reporter au tableau du précédent lab) aient prévu un chemin pour arriver jusqu'à cet état ainsi que d'utiliser ses transitions.

L'analyse du robot avec un testbench différent nous donne alors des indications suivantes : certaines transitions ne sont jamais atteintes, l'état SCANAREA est bien accessible pour l'automate.

Deux hypothèses se posent alors : Soit le système est mal conçu, soit c'est les testbenches. Avant de se replonger dans une éventuelle réécriture des architectures et/ou entité (la couverture des branches pourrait nous aider pour cela). Il faut être certains que c'est les testbenches qui ne sont pas faux.

Après interprétation des résultats, on crée des séquences de signaux qui nous permettront d'être sûr que les testbenches du robot ne sont pas faux.

1.2 Correction du code après une première analyse

1.2.1 Correction du testbench du robot

1.2.1.1 Les branchements Le code coverage analysis nous indique que certains branchements ne sont pas exécutés. En effet, j'avais mis des sécurités sur mes conditions. Prenons l'exemple suivant :

```
if aboverestth = '1' then nextstate <= RANDOMWALK;
elsif aboverestth = '0' then nextstate <= RESTING;
```

Je voulais être sûr que le deuxième statement ne s'exécute uniquement si aboverestth vaut '0' et non pas si aboverestth est différent de '1'.

Je ne sais pas si je dois garder cette sécurité.. Pour les tests de couverture, je l'enlève donc.

1.2.1.2 Les états et les transitions Grâce au graph de couverture (figure 4) il faut tester les transitions suivantes :

- Reset depuis MOVETOHOME
- Aller de SCANAREA à MOVETOFOOD (rappel de contraintes de transition : AboveSearch = '0' & findfood = '0' & scantimeup = '1')
- Aller de SCANAREA à RANDOMWALK (rappel de contraintes de transition : AboveSearch = '0' & findfood = '1')

1.2.1.3 Résultat après correction Après correction, on obtient donc les résultats suivants :

Coverage Report Summary Data by file

```
=====
=== File: /tp/xm1iarc/xm1iarc003/projet_asic/ARC/lab4/robot.vhd
=====
```

Enabled Coverage	Active	Hits	Misses	% Covered
-----	-----	----	-----	-----
Stmts	28	28	0	100.0
Branches	43	43	0	100.0
FEC Condition Terms	11	11	0	100.0
FSMs				100.0
States	9	9	0	100.0
Transitions	31	31	0	100.0

```
=====
=== File: /tp/xm1iarc/xm1iarc003/projet_asic/ARC/lab4/testRobot.vhd
=====
```

Enabled Coverage	Active	Hits	Misses	% Covered
-----	-----	----	-----	-----
Stmts	14	14	0	100.0

Total Coverage By File (code coverage only, filtered view): 100.0%


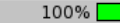


instance	branch %	state %	state graph	transition %	transition graph
testrobot	100%	100%		100%	
A	100%	100%		100%	

Figure 6: Rapport de couverture

1.2.1.4 Conclusion de la correction du robot Avec ces tests de couvertures qui sont concluants, on peut donc affirmer que le robot se comporte correctement grâce aux modifications effectuées sur notre code. Voir annexes :

1.2.2 Correction du testbench du system

On sait ici que plusieurs problèmes sont à résoudre pour obtenir un test de couverture concluant. Il faut donc :

- Passer dans l'état SCANAREA
- Utiliser les différentes transitions :
 - MOVETOFOOD vers SCANAREA (Contraintes : abovesearchth = '0' & losftood = '1')
 - SCANAREA vers RANDOMWALK (Contraintes: abovesearchth = '0' & findfood = '0' & scantimeup = '1')
 - SCANAREA vers MOVETOFOOD (Contraintes : abovesearchth = '0' & findfood = '1')
 - SCANAREA vers HOMING (Contraintes : abovesearchth = '1')
- Utiliser le reset dans pour tout les états

1.2.2.1 Résultat après correction Après correction, on obtiens donc les résultats suivants :

Coverage Report Summary Data by file

=====				
=== File: /tp/xm1iarc/xm1iarc003/projet_asic/ARC/lab4/count.vhd				
=====				
Enabled Coverage	Active	Hits	Misses	% Covered
-----	----	----	-----	-----
Stmts	17	17	0	100.0
Branches	20	19	1	95.0
FEC Condition Terms	10	6	4	60.0
FSMs				100.0
States	2	2	0	100.0
Transitions	4	4	0	100.0
=====				
=== File: /tp/xm1iarc/xm1iarc003/projet_asic/ARC/lab4/robot.vhd				
=====				
Enabled Coverage	Active	Hits	Misses	% Covered
-----	----	----	-----	-----
Stmts	28	28	0	100.0
Branches	43	43	0	100.0
FEC Condition Terms	11	11	0	100.0
FSMs				87.0
States	9	9	0	100.0
Transitions	31	23	8	74.1
=====				
=== File: /tp/xm1iarc/xm1iarc003/projet_asic/ARC/lab4/testSystep.vhd				
=====				
Enabled Coverage	Active	Hits	Misses	% Covered
-----	----	----	-----	-----
Stmts	12	12	0	100.0

Total Coverage By File (code coverage only, filtered view): 91.9%

Instance	Branch %	State %	Transition %	Stmt %	Stmt graph	Branch graph	State graph	Transition graph
testsystem	96.4%	100%	79.5%	98.7%	<div></div>	<div></div>	<div></div>	<div></div>
line_18								
line_20								
line_27								
line_30								
line_34								
line_36								
line_38								
line_41								
S	96.4%	100%	79.5%	98.4%	<div></div>	<div></div>	<div></div>	<div></div>
C1	95%	100%	100%	100%	<div></div>	<div></div>	<div></div>	<div></div>
C2	90%	100%	100%	94.1%	<div></div>	<div></div>	<div></div>	<div></div>
line_35								
R	100%	100%	74.2%	100%	<div></div>	<div></div>	<div></div>	<div></div>

Figure 7: Rapport de couverture

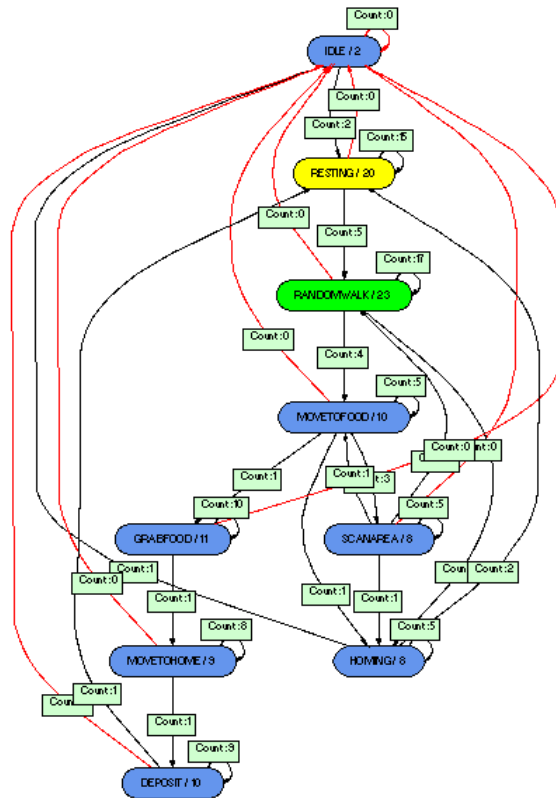


Figure 8: Graph de couverture des états du robot

1.2.2.2 Interprétation des résultats La couverture n'est pas totale car je n'ai pas testé si le reset marche pour tout les états. De plus, les counters ne rentre pas dans les configurations permettant d'avoir un branching parfait.

Néanmoins, je ne pense pas qu'il est nécessaire d'aller plus loin. Les tests de couvertures sont passé haut la main avec un Total Coverage by file the 91,9%.

2 Vérification d'assertions temporelles

Ici, le code de P1 et P2 est faux, il est corrigé en section 4. La démonstration que l'assertion est juste reste néanmoins la même.

2.1 P1

Chaque fois que le robot commence sa recherche, il ne pourra pas atteindre l'état grabfood tant qu'il n'aura pas trouvé de nourriture (entrée findfood)

2.1.1 Formule PSL

```
-- psl property p1 is always (search = '1' -> (eventually!(state=GRABFOOD)) and ( not(state = GRABFOOD) until findfood = 1))
-- psl assert p1;
```

Figure 9: PSL property P1

2.1.2 Analyse du chronogramme

Dans un premier temps, on veut vérifier que l'assertion démarre lorsque search vaud 1. (Entouré en bleu ci dessous). En suite, on veut vérifier que l'assertion se finissent avec succès lorsqu'avant de passer dans l'état grabfood on détecte une findfood = '1' (entouré en rouge)

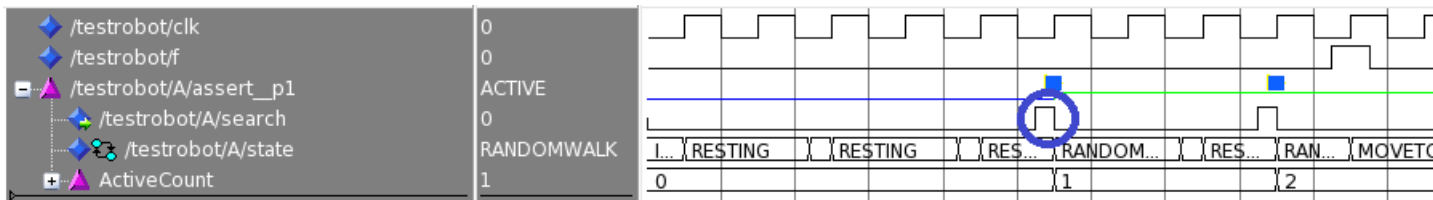


Figure 10: Assertion P1 sur le chronogramme

On voit alors bien ici que l'assertion se déclenche au bon moment et se termine avec succès comme souhaité.

2.2 P2

Chaque fois que le robot commence sa recherche, il ne pourra pas atteindre l'état homing tant qu'il n'aura pas dépassé le seuil du temps de recherche

2.2.1 Formule PSL

```
-- psl property p2 is always (search = '1' -> (eventually!(state=HOMING)) and ( not(state = HOMING) until (
-- psl assert p2;
```

Figure 11: PSL property P2

2.2.2 Analyse du chronogramme

Ici on veut vérifier que l'assertion démarre en même temps que la recherche (losange rouge), qu'aboveresearchth soit bien à '1' (rond rouge) avant de rentrer dans l'état homing (rectangle bleu).

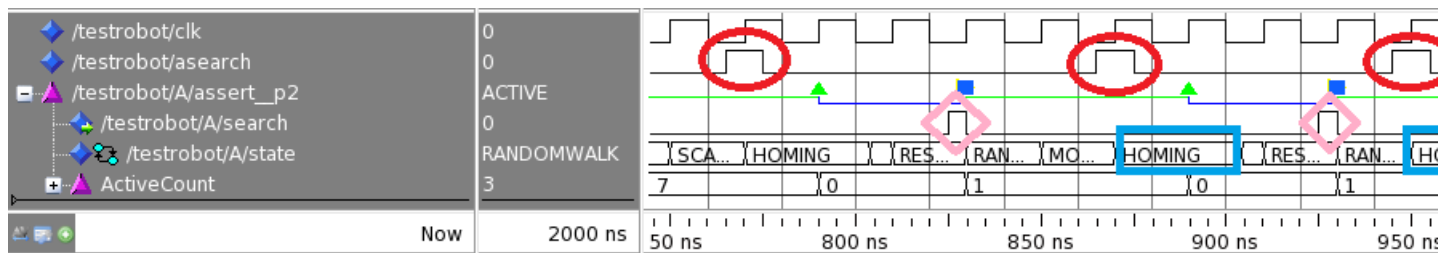


Figure 12: Assertion P2 sur le chronogramme

Tout ce passe comme prévu, la PSL property vérifie bien le comportement énoncé.

2.3 P3

Chaque fois que le robot ramène de la nourriture, il devra passer par l'état deposit avant de positionner la sortie rest à '1'

2.3.1 Formule PSL

```
-- psl property p3 is always (state = MOVETOHOME -> state=DEPOSIT before! rest = '1');  
-- psl assert p3;
```

Figure 13: PSL property P3

2.3.2 Analyse du chronogramme

Ce qui est intéressant ici est que lorsqu'on fait un reset : lors de la transition de IDLE à RESTING, on met aussi en sortie rest à 1.

J'interprète ici le fait que le robot ramène de la nourriture lorsque qu'il rentre dans l'état MOVETOHOME.

Il faut donc vérifier que :

- L'assertion se déclenche bien à l'entrée de MOVETOHOME (rectangle bleu)
- L'assertion s'arrête avec succès lorsque l'on rentre dans l'état deposit (rectangle vert)
- L'assertion s'arrête avec un échec lorsqu'on effectue un reset (rond rouge)

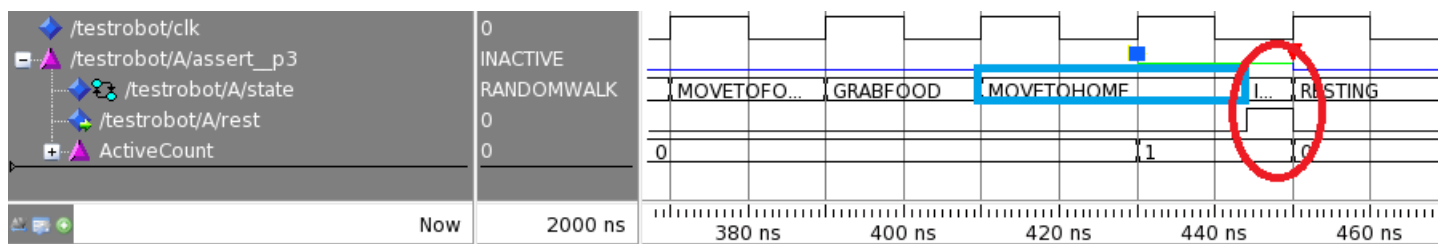


Figure 14: Assertion P3 sur le chronogramme

Les résultats sont concluants, la PSL property vérifie bien le comportement énoncé.

2.4 P4

A partir du moment où le robot commence sa recherche, si `abovesearchth` n'est pas atteint et qu'il trouve de la nourriture puis la perd puis il a dépassé le délai pendant lequel il pouvait tenter à nouveau de l'approcher, alors il repasse dans l'état de recherche.

2.4.1 Formule PSL

```
-- psl property p4 is
-- always { search = '1';
-- (abovesearchth = '0' and findfood = '0' and not(state = IDLE) )[*];
-- (abovesearchth = '0' and findfood = '1' and not(state = IDLE)) ;
-- (abovesearchth = '0' and lostfood = '0' and closetofood = '0' and not(state = IDLE))[*];
-- (abovesearchth = '0' and lostfood = '1' and not(state = IDLE)) ;
-- (abovesearchth = '0' and findfood = '0' and scantimeup = '0' and not(state = IDLE))[*];
-- (abovesearchth = '0' and findfood = '0' and scantimeup = '1' and not(state = IDLE)) } | => {st
-- psl assert p4;
```

Figure 15: PSL property P4

Pour cette propriété, j'ai jugé bon de rajouter le `not(state = IDLE)` car sinon la lecture des assertions n'était pas cohérente. Cela permet alors que l'assertion ne reste pas valide lors d'un reset. De plus, j'ai rajouté une contrainte `closetofood = '0'` par rapport à l'énoncé du tp pour éviter que lorsque le robot prend la transition de `MOVETOFOOD` vers `GRABFOOD`, l'assertion reste valide.

De plus, je me posais la question si ça devait être l'état `RANDOMWALK` ou le passage de `rest` à `'1'` qui déclenche l'assertion. Après quelques tests, si c'est l'état `RANDOMWALK`, alors plusieurs assertions peuvent se déclencher en même temps. C'est donc `rest = '1'` qui déclenchera l'assertion.

On obtiendras alors le chronogramme ci dessous.

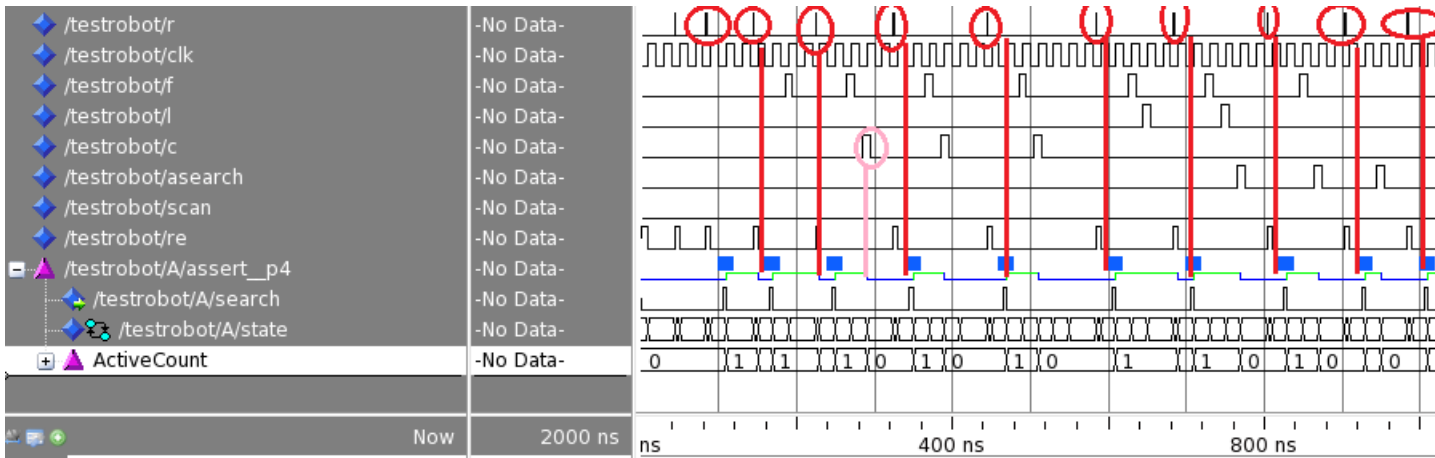


Figure 16: Assertion P4 sur le chronogramme

Comme on ne s'intéresse pas aux assertions avec reset (rond rouge) et que le rond rose représente le chemin où `closetofood = '1'`, je zoom donc sur la seule partie qui nous intéresse.

On peut alors apercevoir les différents morceaux de la phrase qui fait la description de l'assertion tel que :

- Label A : Chaque fois que le robot commence sa recherche
- Label B : Puis `abovesearchth` n'est pas atteint et la nourriture n'est pas trouvée (pendant un nombre quelconque de cycles)
- Label C : Puis `abovesearchth` n'est pas atteint et la nourriture est trouvée
- Label D : Puis `abovesearchth` n'est pas atteint et la nourriture n'est pas perdue (pendant un nombre quelconque de cycles)
- Label E : Puis `abovesearchth` n'est pas atteint et la nourriture est perdue

- Label F : Puis abovesearchth n'est pas atteint et la nourriture n'est pas trouvée et le délai de récupération n'est pas dépassé i.e., scantimeup n'est pas à '1' (pendant un nombre quelconque de cycles)
- Label G : Puis abovesearchth n'est pas atteint et la nourriture n'est pas trouvée et le délai est dépassé
- Label H : Alors il doit repasser dans l'état de recherche

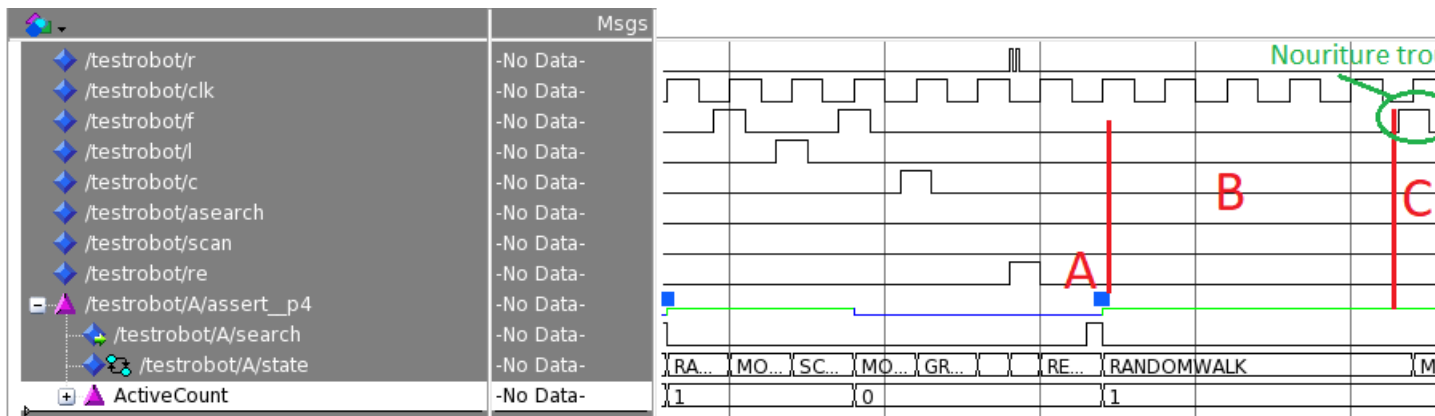
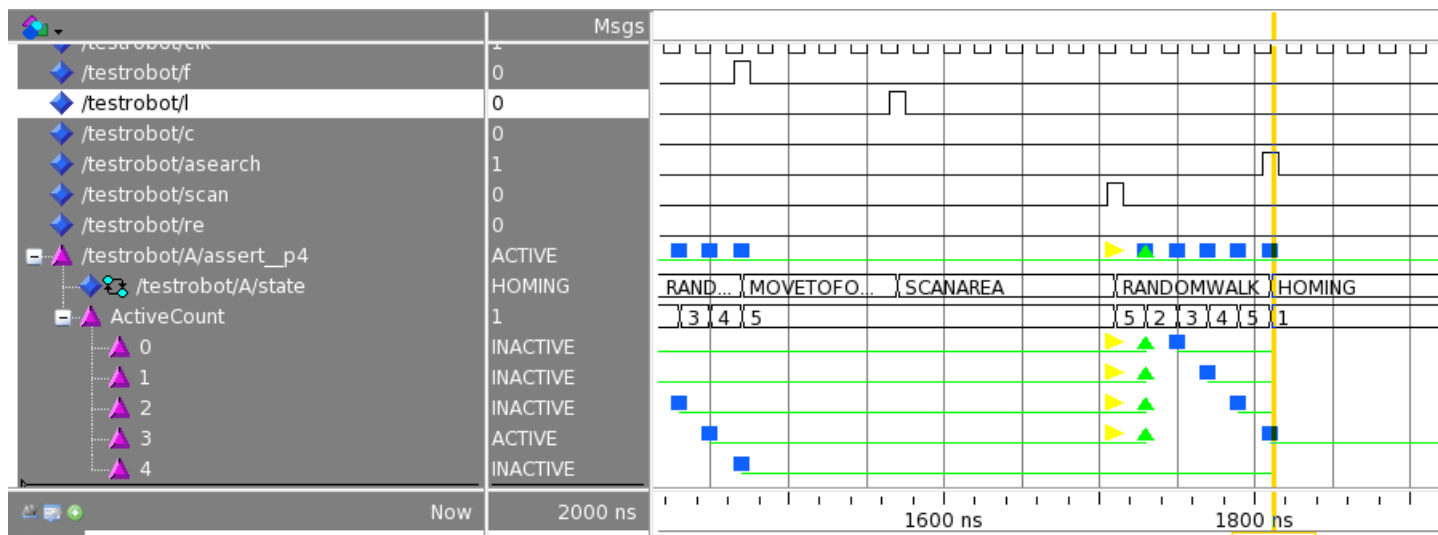


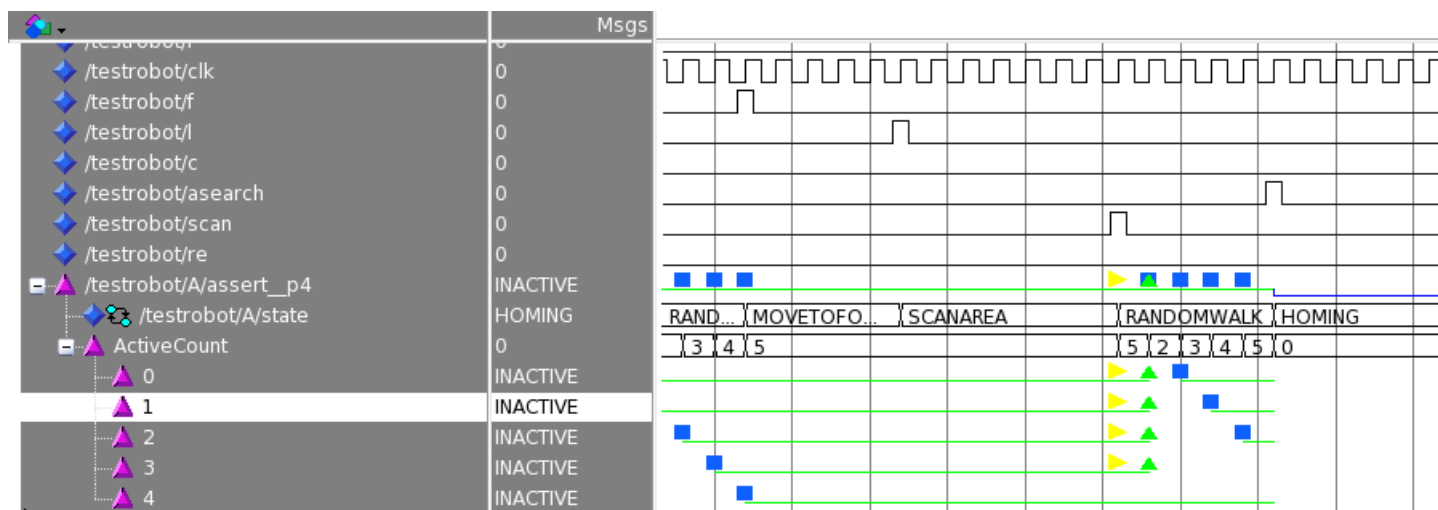
Figure 17: Assertion P4 sur le chronogramme

Les résultats sont concluants, la PSL property vérifie bien le comportement énoncé une première fois mais une assertion suivante n'est pas levée. J'avais donc tort. Il faut bien que ce soit l'état RANDOMWALK qui déclenche l'assertion.

Cependant, toutes les assertions se terminent quand abovesearch vaut '1' mais une assertion se lève aussi pendant le début de ce cycle et ne verras pas le signal d'arrêt :



Pour régler le problème il faut vérifier que la première expression régulière vérifie que `aboveresearchth` soit bien à '0'.



C'est bien ce que l'on attend.

```
-- psl property p4 is
--  always { state = RANDOMWALK and abovesearchth = '0';
--  (abovesearchth = '0' and findfood = '0' and not(state = IDLE) )[*];
--  (abovesearchth = '0' and findfood = '1' and not(state = IDLE)) ;
--  (abovesearchth = '0' and lostfood = '0' and closetofood = '0' and not(state = IDLE))[*];
--  (abovesearchth = '0' and lostfood = '1' and not(state = IDLE)) ;
--  (abovesearchth = '0' and findfood = '0' and scantimeup = '0' and not(state = IDLE))[*];
--  (abovesearchth = '0' and findfood = '0' and scantimeup = '1' and not(state = IDLE)) } | => {state = RANDOMWALK
-- psl assert p4;
```

2.5 P5

2.5.1 Formule PSL

```
-- psl property p5 is always ( {state = RESTING } | => {[*] ; state = RANDOMWALK }));
-- psl assert p5;
```

Figure 21: PSL property P5

2.5.2 Analyse du chronogramme

Comme précédemment, on regarde quand est-ce que ça se déclenche et quand est-ce que ça s'arrête en succès.

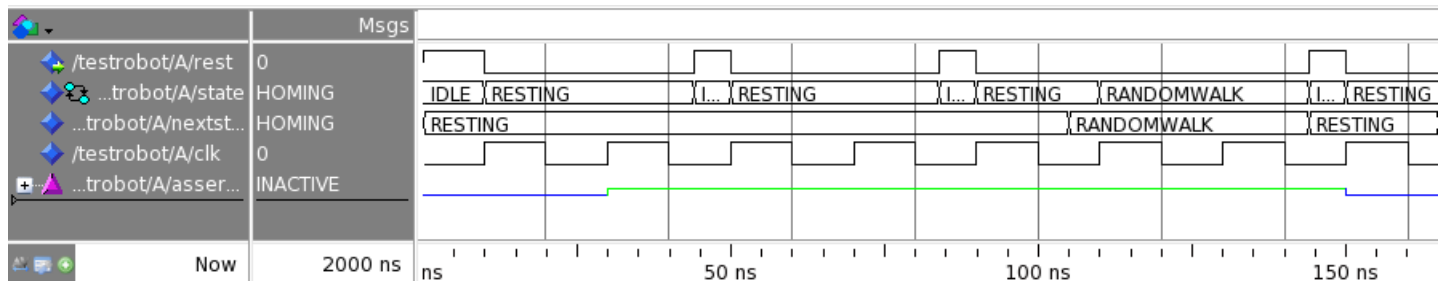


Figure 22: Assertion P5 sur le chronogramme

2.5.3 Réaliser une simulation dans laquelle la condition attendue ne se produit pas

Il suffit de faire de ne jamais mettre aboverestth à '1' dans le testbench.

2.5.3.1 Qu'observez-vous ?

L'assertion reste valide tout le temps.

2.5.3.2 Que se passe-t-il si vous faites alors "End simulation" dans ModelSim ?

Une fenêtre nous demande si l'on est sûr de bien vouloir quitter la simulation puis il est écrit en sur la sortie stdout :

```
# End time: 20:52:45 on Apr 05,2020, Elapsed time: 0:13:14
# Errors: 21, Warnings: 0
```

2.5.3.3 Comment interprétez-vous ce qui se produit ?

Que je ne pense pas qu'il s'est passé ce que le vous attendiez...

3 Conclusion

Tout est cohérent par rapport au design. Les assertions sont alors complémentaire à la couverture de code. La couverture de code permet de vérifier que tout les états du robot sont atteignable quant au assertions, elles permettent de prouver que le robots réagis correctement.

J'ai effectivement fait des corrections sur le counter (voir annexes). J'ai très peu modifié mes testbenchs.

4 Réaction après discussion par mail

Soit la formule suivante :

(eventually!(φ_1)) and (φ_2 until b)

Or :

- eventually! $\varphi_1 \Leftrightarrow [*] \models \varphi_1$
- φ_2 untill b $\Leftrightarrow \{[*]; \varphi_2[+]\} \models b$

Alors comme $\{[*]; \varphi_2[+]\} \subseteq [*]$ on peut dire que (eventually!(φ_1)) and (φ_2 until b) $\Rightarrow \{[*]; \varphi_2[+]\} \models b$ and φ_1

Dans notre cas avec l'assertion $P1\varphi_1 = (\text{state} = \text{GRABFOOD})$, $\varphi_2 = \text{not}(\text{state} = \text{GRABFOOD})$, b = (findfood = 1) et quand l'état passe à GRABFOOD, findfood vaux encore 1

Donc l'assertion vérifie au moins que la phrase Chaque fois que le robot commence sa recherche, il ne pourra pas atteindre l'état grabfood tant qu'il n'aura pas trouvé de nourriture (entrée findfood) est vraie.

Alors certes elle ne vérifie pas que ça mais la démonstration sur le chronogramme reste vraie.

On peut néanmoins remplacer le code de P1 par :

```
-- psl property p1 is always (search = '1' -> (findfood = '1') before! (state = GRABFOOD) );
-- psl assert p1;
```

Figure 23: Assertion P1 révisé

et obtenir le chronogramme suivant :

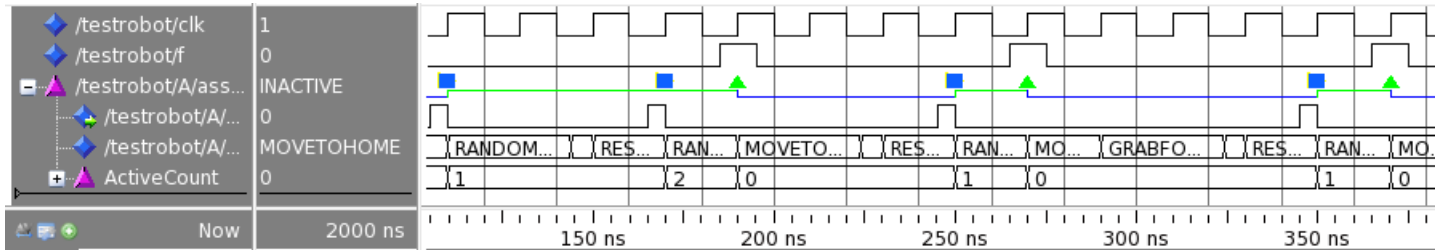


Figure 24: Assertion de P1 révisé sur le chronogramme

Même si au final notre formule était juste (P1 était un sous ensemble de notre assertion). Le code a été corrigé.

On peut faire de même avec P2 tel que :

```
-- psl property p2 is always (search = '1' -> (abovesearchth = '1') before! (state = HOMING) );
-- psl assert p2;
```

Figure 25: Assertion P2 révisé

5 Annexes

5.1 Différences de code entre la partie 1 et 2 du projet

36	process(reset, clk, start)	36	process(reset, clk, start)
37	begin	37	begin
38	-- RESET : asynchrone haut	38	-- RESET : asynchrone haut
39	if reset = '1' then	39	if reset = '1' then
40	state <= IDLE;	40	state <= IDLE;
41	-- HORLOGE : front montant	41	-- HORLOGE : front montant
		42	+
		43	+
		44	+
42	elsif (clk'event and clk = '1') then	45	elsif (clk'event and clk = '1') then
43	state <= nextstate;	46	state <= nextstate;
44	-- Detecter un pic sur start	47	-- Detecter un pic sur start
45	- elsif ((start = '1') and (state = IDLE))then	48	+
46	- state <= COUNTING;		
47	-		
48	end if;	49	end if;
49		50	
50		51	
51	end process;	52	end process;

Figure 26: Différence avec la première partie du projet de count.vhd

5.2 Le compteur

```

C:\Users\atati\AppData\Roaming\Notepad++\plugins\config\NppFTP\Cache\m1arc003@cinetd105.cine.inpg.fr\ip\m1arc\m1arc003\projet_asic\ARC\lablundi 6 avril 2020 00:04
-- ROBOT.VHD --

library ieee;
use ieee.std_logic_1164.all;

entity Robot is
    port(reset, clk, athome, findfood, lostfood, closetofood,
          success, aboverestth, abovesearchth, scantimeup: in std_logic;
          rest, search, food: out std_logic);
end Robot;

architecture automate_robot of Robot is

    type States is (IDLE, RESTING, RANDOMWALK, SCANAREA, HOMING, MOVETOFOOD, MOVETOHOM,
                    DEPOSIT, GRABFOOD);
    Signal state, nextstate : States := IDLE;
    -- psl default clock is rising edge(clk);
    -- psl property p1 is always (search = '1' -> (findfood = '1') before! (state =
    GRABFOOD));
    -- psl assert p1;

    -- psl property p2 is always (search = '1' -> (abovesearchth = '1') before! (state =
    HOMING));
    -- psl assert p2;

    -- psl property p3 is always (state = MOVETOHOM -> state=DEPOSIT before! rest = '1');
    -- psl assert p3;

    -- psl property p4 is
    -- always (state = RANDOMWALK and abovesearchth = '0');
    -- (abovesearchth = '0' and findfood = '0' and not(state = IDLE)) [*];
    -- (abovesearchth = '0' and findfood = '1' and not(state = IDLE));
    -- (abovesearchth = '0' and lostfood = '0' and closetofood = '0' and not(state =
    IDLE)) [*];
    -- (abovesearchth = '0' and lostfood = '1' and not(state = IDLE));
    -- (abovesearchth = '0' and findfood = '0' and scantimeup = '0' and not(state = IDLE)) [*];
    -- (abovesearchth = '0' and findfood = '0' and scantimeup = '1' and not(state = IDLE))
    |>= (state = RANDOMWALK);
    -- psl assert p4;

    -- psl property p5 is always ( (state = RESTING ) |>= ( [*] ; state = RANDOMWALK ));
    -- psl assert p5;

begin

    -- Calcul de l'etat suivant
    -- Comme on est en std_logic, "elsif = '0'" et non "else", car le signal peut avoir
    d'autre valeur
    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 <= MOVETOHOM;
                else--elsif success = '0' then
                    nextstate <= GRABFOOD;
                end if;
            when MOVETOHOM =>
                if athome = '1' then nextstate <= DEPOSIT;
                else--elsif athome = '0' then
                    nextstate <= MOVETOHOM;
                end if;
            when DEPOSIT =>
                if success = '1' then nextstate <= RESTING;
                else--elsif success = '0' then
                    nextstate <= DEPOSIT;
                end if;
        end case;
    end process;

    -- MISE A JOUR DU REGISTRE D'ETAT

    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;

    -- MISE A JOUR DES OUTPUTS
    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';

end automate_robot;

```

Figure 27: count.vhd

5.3 Architecture du robot

```
C:\Users\atab\AppData\Roaming\Notepad++\plugins\config\NppFTP\Cache\um1arc003@cineldf105.cime.inpg.fr\tpum1arc\um1arc003\projet_asic\ARC\lablundi 6 avril 2020 00:04
-- ROBOT.VHD --

library ieee;
use ieee.std_logic_1164.all;

entity Robot is
    port(reset, clk, athome, findfood, lostfood, closetofood,
    success, aboverestth, abovesearchth, scantimeup: in std_logic;
    rest, search, food: out std_logic);
end Robot;

architecture automate_robot of Robot is

    type States is (IDLE, RESTING, RANDOMWALK, SCANAREA, HOMING, MOVETOFOOD, MOVETOHOME,
    DEPOSIT, GRABFOOD);
    Signal state, nextstate : States := IDLE;
    -- psl default clock is rising_edge(clk);
    -- psl property p1 is always (search = '1' -> (findfood = '1') before! (state =
    GRABFOOD));
    -- psl assert p1;

    -- psl property p2 is always (search = '1' -> (abovesearchth = '1') before! (state =
    HOMING));
    -- psl assert p2;

    -- psl property p3 is always (state = MOVETOHOME -> state=DEPOSIT before! rest = '1');
    -- psl assert p3;

    -- psl property p4 is
    -- always ( state = RANDOMWALK and abovesearchth = '0';
    -- (abovesearchth = '0' and findfood = '0' and not(state = IDLE)) [*];
    -- (abovesearchth = '0' and findfood = '1' and not(state = IDLE));
    -- (abovesearchth = '0' and lostfood = '0' and closetofood = '0' and not(state =
    IDLE)) [*];
    -- (abovesearchth = '0' and lostfood = '1' and not(state = IDLE));
    -- (abovesearchth = '0' and findfood = '0' and scantimeup = '0' and not(state = IDLE)) [*];
    -- (abovesearchth = '0' and findfood = '0' and scantimeup = '1' and not(state = IDLE)) }
    | => (state = RANDOMWALK);
    -- psl assert p4;

    -- psl property p5 is always ( (state = RESTING) | => {[*]; state = RANDOMWALK});
    -- psl assert p5;

begin

    -- Calcul de l'état suivant
    -- Comme on est en std_logic, "elsif = '0'" et non "else", car le signal peut avoir
    d'autre valeur
    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 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;

    -- MISE A JOUR DU REGISTRE D'ETAT
    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;

    -- MISE A JOUR DES OUTPUTS
    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';

end automate_robot;
```

```
C:\Users\atab\AppData\Roaming\Notepad++\plugins\config\NppFTP\Cache\um1arc003@cineldf105.cime.inpg.fr\tpum1arc\um1arc003\projet_asic\ARC\lablundi 6 avril 2020 00:04
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;
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;

-- MISE A JOUR DU REGISTRE D'ETAT
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;

-- MISE A JOUR DES OUTPUTS
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';

end automate_robot;
```

Figure 28: robot.vhd

5.4 TestBench du robot

```
C:\Users\atb\AppData\Roaming\Notepad++\plugins\Config\NppFTP\Cache\m1arc003@cimeid105.cime.inpg.#tptm1arc003\project_asic\ARC\laund 6 avril 2020 00:28
-- TESTROBOT.VHD --

library ieee;
use ieee.std_logic_1164.all;

entity testRobot is
end testRobot;

architecture test1 of testRobot is
    component Robot is
        port(reset, clk, athome, findfood, lostfood, closetofood,
            success, aboverestth, abovesearchth, scantimeup: in std_logic;
            rest, search, food: out std_logic);
    end component;
    signal r, clk, ah, f, l, c, s, arest, asearch, scan, re, se, fo : std_logic := '0';
begin
    A: Robot port map(r, clk, ah, f, l, c, s, arest, asearch, scan, re, se, fo);
    -- manage reset
    r <= '0', '1' after 44 ns, '0' after 45 ns, '1' after 84 ns, '0' after 85 ns, '1' after
    144 ns, '0' after 145 ns,
    '1' after 224 ns, '0' after 225 ns, '1' after 324 ns, '0' after 325 ns, '1' after
    444 ns, '0' after 445 ns, '1' after 584 ns, '0' after 585 ns,
    '1' after 684 ns, '0' after 685 ns, '1' after 804 ns, '0' after 805 ns, '1' after
    904 ns, '0' after 905 ns, '1' after 984 ns, '0' after 985 ns,
    '1' after 1064 ns, '0' after 1065 ns, '1' after 1204 ns, '0' after 1205 ns, '1'
    after 1340 ns, '0' after 1341 ns
    , '1' after 1342 ns, '0' after 1343 ns;
    -- manage clock
    process
    begin
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
        wait for 10 ns;
    end process;
    -- manage athome
    ah <= '0', '1' after 545 ns, '0' after 555 ns, '1' after 1045 ns, '0' after 1055 ns, '1'
    after 1165 ns, '0' after 1175 ns;
    -- manage findfood
    f <= '0', '1' after 185 ns, '0' after 195 ns, '1' after 265 ns, '0' after 275 ns, '1'
    after 365 ns, '0' after 375 ns,
    '1' after 485 ns, '0' after 495 ns, '1' after 625 ns, '0' after 635 ns, '1' after 725
    ns, '0' after 735 ns,
    '1' after 845 ns, '0' after 855 ns, '1' after 1105 ns, '0' after 1115 ns, '1' after 1245
    ns, '0' after 1255 ns,
    '1' after 1285 ns, '0' after 1295 ns, '1' after 1465 ns, '0' after 1475 ns ;
    -- manage lostfood
    l <= '0', '1' after 645 ns, '0' after 655 ns, '1' after 745 ns, '0' after 755 ns, '1'
    after 1265 ns, '0' after 1275 ns,
    '1' after 1565 ns, '0' after 1575 ns;
    --manage closetofood
    c <= '0', '1' after 285 ns, '0' after 295 ns, '1' after 385 ns, '0' after 395 ns, '1'
    after 505 ns, '0' after 515 ns,
    '1' after 1125 ns, '0' after 1135 ns , '1' after 1305 ns, '0' after 1315 ns;
    --manage success
    s <= '0', '1' after 405 ns, '0' after 415 ns, '1' after 525 ns, '0' after 535 ns, '1'
    after 1145 ns, '0' after 1155 ns,
    '1' after 1185 ns, '0' after 1195 ns, '1' after 1325 ns, '0' after 1335 ns;

    -- manage aboverestth
    arest <= '0', '1' after 105 ns, '0' after 115 ns, '1' after 165 ns, '0' after 175 ns,
    '1' after 245 ns, '0' after 255 ns , '1' after 345 ns, '0' after 355 ns, '1' after 465
    ns, '0' after 475 ns,
    '1' after 605 ns, '0' after 615 ns, '1' after 705 ns, '0' after 715 ns, '1' after 825
    ns, '0' after 835 ns,
    '1' after 925 ns, '0' after 935 ns, '1' after 1005 ns, '0' after 1015 ns, '1' after 1085
    ns, '0' after 1095 ns,
    '1' after 1225 ns, '0' after 1235 ns, '1' after 1365 ns, '0' after 1375 ns;

    -- manage abovesearchth
```

```
C:\Users\atb\AppData\Roaming\Notepad++\plugins\Config\NppFTP\Cache\m1arc003@cimeid105.cime.inpg.#tptm1arc003\project_asic\ARC\laund 6 avril 2020 00:28
    asearch <= '0', '1' after 765 ns, '0' after 775 ns, '1' after 865 ns, '0' after 875 ns,
    '1' after 945 ns, '0' after 955 ns,
    '1' after 1025 ns, '0' after 1035 ns, '1' after 1805 ns, '0' after 1815 ns ;

    -- manage scantimeup
    scan <= '0', '1' after 1705 ns, '0' after 1715 ns;

end test1;

library work;
configuration config1 of work.testRobot is
    for test1
        for A:Robot use entity work.Robot(automate_robot);
        end for;
    end for;
end config1;
```

Figure 29: testRobot.vhd

5.5 TestBench du système

```
C:\Users\ratab\AppData\Roaming\Wotepad++\plugins\Config\NppFTP\Cache\xm1iarc003@cimeld105.cime.inpg.fr\tp\xm1iarc\xm1iarc003\projet_asic\ARC\lab\undi 6 avril 2020 00:30
-- TESTSYSTEM.VHD --

library ieee;
use ieee.std_logic_1164.all;

entity testSystem is
end testSystem;

architecture test3 of testSystem is
    component System is
        port(reset, clk, athome, findfood, lostfood, closetofood, success,
              scantimeup: in std_logic;
              food: out std_logic);
    end component;
    Signal reset, clk, athome, findfood, lostfood, closetofood, success, scantimeup, food :
    std_logic := '0';
begin
    S: System port map(reset, clk, athome, findfood, lostfood, closetofood, success,
                       scantimeup, food);
    reset <= '1', '0' after 5 ns, '1' after 1830 ns, '0' after 1850 ns;
    process
    begin
        clk <= '0';
        wait for 10 ns;
        clk <= '1';
        wait for 10 ns;
    end process;

    athome <= '0', '1' after 400 ns, '0' after 440 ns, '1' after 1030 ns, '0' after 1050 ns,
    '1' after 1530 ns, '0' after 1550 ns;

    findfood <= '0', '1' after 590 ns, '0' after 610 ns, '1' after 1330 ns, '0' after 1350 ns,
    '1' after 1430 ns, '0' after 1450 ns, '1' after 1650 ns, '0' after 1670 ns,
    '1' after 1690 ns, '0' after 1710 ns;

    closetofood <= '0', '1' after 630 ns, '0' after 650 ns;

    success <= '0', '1' after 850 ns, '0' after 870 ns, '1' after 1230 ns, '0' after 1250 ns;

    lostfood <= '0', '1' after 1370 ns, '0' after 1390 ns, '1' after 1670 ns, '0' after
    1690 ns,
    '1' after 1710 ns, '0' after 1730 ns;

    scantimeup <= '0', '1' after 1410 ns, '0' after 1430 ns;

end test3;
```

Figure 30: testSystem.vhd