

```
-- 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 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;
```