UNIVERSITÉ PARIS 12 VAL de MARNE — FACULTÉ DES SCIENCES ET TECHNOLOGIE

Introduction au VHDL

Philippe Meyne – philippe.meyne@univ-paris12.fr

Structure du langage VHDL

- Langage parallèle / concurrent
- Langage de description d'application séquentielle
- Plusieurs niveaux de description

2

Architecture générale d'un fichier de description VHDL

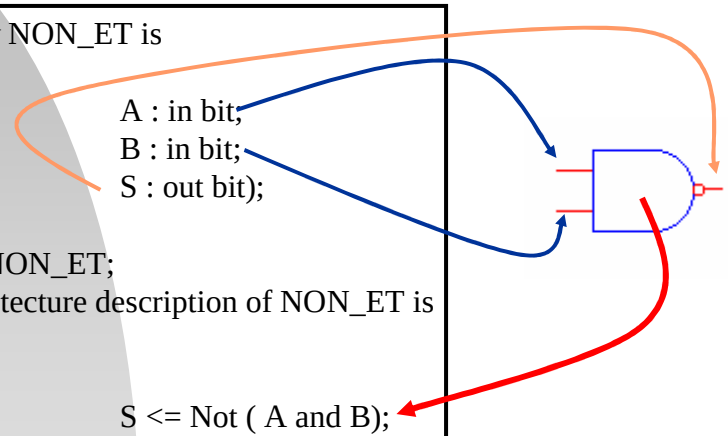
- Déclaration : `entity`
- Déclaration : `architecture`

3

Architecture générale d'un fichier de description VHDL (suite)

- Exemple : mise en œuvre d'un Nand

```
entity NON_ET is
port(
        A : in bit;
        B : in bit;
        S : out bit);

end NON_ET;
Architecture description of NON_ET is
begin

        S <= Not ( A and B);

end description;
```



4

Les types de variables

- Type entiers :

```
D : in integer range 0 to 256;
```

- Type flottants :

```
E : in real;
```

- Type record (enregistrement) :

```
type node is record
     capa : real;
     sortance : integer range 1 to 20;
end record;
…
     process(clk)
     variable x : node;
     begin
```

5

Les types de variables (suite)

- Type physique :

```
A : in bit;

D1 : out bit_vector(0 to 7);

C : in std_logic;

D1 : out std_logic_vector(0 to 7);
```

6

Les conversions de type

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity conversion is
port(
            sortie : out integer range 0 to 255;
            entree : in std_logic_vector(0 to 7);
            A : in bit;
            E: out real);

end conversion;
Architecture description of conversion is
begin
            E <= real(A);
            sortie <= integer(entree);
end description;
```

7

Les conversions de type (suite)

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity conversion is

port(
            entree : in integer range 0 to 255;
            sortie : out std_logic_vector(0 to 7));

end conversion;
Architecture description of conversion is
begin
            sortie <= std_logic_vector(entree);
end description;
```

8

Les signaux

- variables intermédiaires

```
entity bascule_RS is
port(
            reset : in bit;
            set : in bit;
            Q     : out bit;
            QL    : out bit);

end bascule_RS;
Architecture description of bascule_RS is
signal inter1 : bit;
signal inter2 : bit;
begin
            inter1 <= not ( not set and inter2 );
            inter2 <= not( inter1 and not reset);
            QL <= inter2;
            Q <= inter1;
end description;
```

9

Les opérateurs

- Opérateurs logiques parallèles permettant de décrire une application
- Opérateurs arithmétiques

```
+ - * / mod abs
```

- Opérateurs logiques

```
and or not xor nand nor
```

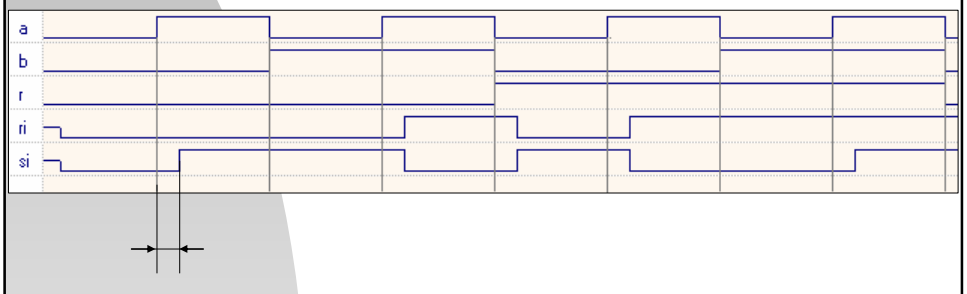
10

Les opérateurs
exemple mise en œuvre d'opérateurs

```
entity demi_adder is
port(           A : in bit;
                B : in bit;
                R : in bit;
                Si : out bit;
                Ri : out bit);
end demi_adder;

architecture descript of demi_adder is
begin
        Si <= A XOR B XOR R;
        Ri <= (A and B) OR (R AND (A OR B) );
end descript;
```

11

Les opérateurs
simulation mise en œuvre d'opérateurs



Retard du au
temps de propagation

12

Les opérateurs (suite)

- Opérateur d'affectation conditionnel

Sortie <= expression when condition else expression

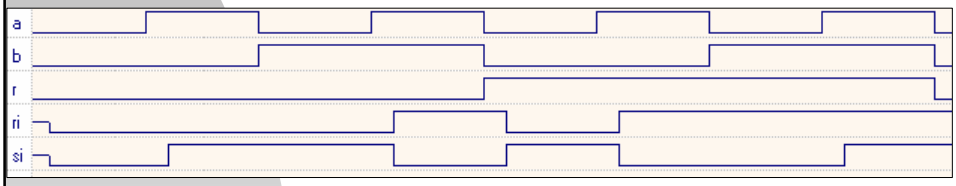
13

Les opérateurs
exemple assignation conditionnelle

```
entity demi_adder is
port(       A : in bit;
            B : in bit;
            R : in bit;
            Si : out bit;
            Ri : out bit);
end demi_adder;
architecture descript of demi_adder is
begin
      Si <= '1' when (A='1' xor B='1' xor R='1')
            else '0';

      Ri <= '1' when (A='1' and B ='1') or
                        ((A='1' or B='1') and R='1')
            else '0';
end descript;
```

14

Les opérateurs
simulation assignation conditionnelle



15

Les opérateurs (suite)

- Opérateur d'affectation sélective

```
with sélecteur select
sortie <=       expression_1 when valeur sélecteur 1
                expression_2 when valeur sélecteur 2
                …
                expression_N when others;
```

16

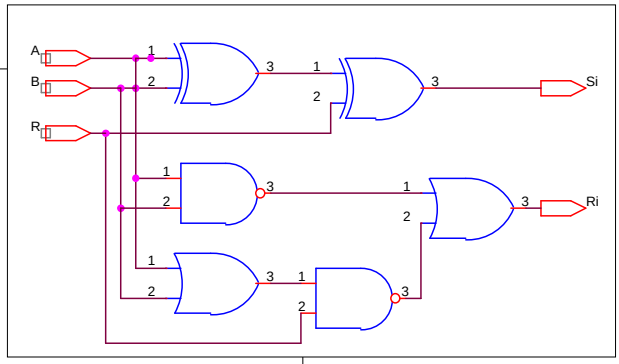
Les opérateurs
exemple

```
entity sept_segment is
port(
                code : in integer range 0 to 9;
                sortie : out bit_vector(0 to 6));
end sept_segment;
architecture description of sept_segment is
begin
        with code select
        sortie <=     "1111110" when 0,
                              "0110000" when 1,
                              "1101101" when 2,
                              "1111001" when 3,
                              "0010111" when 4,
                              "1011011" when 5,
                              "1011111" when 6,
                              "1110000" when 7,
                              "1111111" when 8,
                              "1111011" when 9,
                              "0011101" when others;
end description;
```

17

L'instanciation de composants

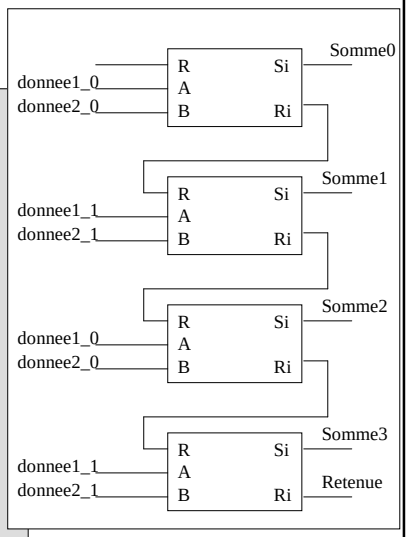
```
entity demi_adder is
port(   A : in bit;
        B : in bit;
        R : in bit;
        Si : out bit;
        Ri : out bit);
end demi_adder;

architecture descript of demi_adder is
begin
        Si <= A xor B xorR R;
        Ri <= (A and B) or (R and (A or B) );
end descript;
```



18

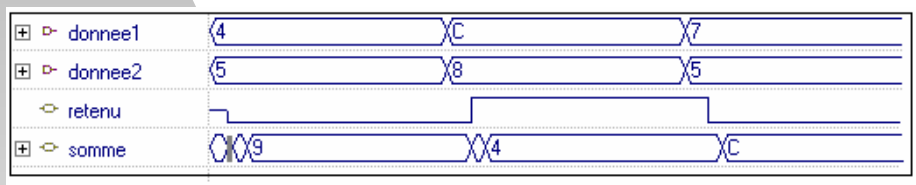
L'instanciation de composants
suite

```
entity full_adder is
port(
        donnee1 : in bit_vector(3 downto 0);
        donnee2 : in bit_vector(3 downto 0);
        somme   : out bit_vector(3 downto 0);
        retenu  : out bit);
end full_adder;

architecture descript of full_adder is
signal ri0,ri1,ri2 : bit;
begin
add_1:  demi_adder
        port map(donnee1(0),donnee2(0),'0',somme(0),ri0);
add_2: demi_adder
        port map(donnee1(1),donnee2(1),ri0,somme(1),ri1);
add_3: demi_adder
        port map(donnee1(2),donnee2(2),ri1,somme(2),ri2);
add_4: demi_adder
        port map(donnee1(3),donnee2(3),ri2,somme(3),retenu);
end descript;
```



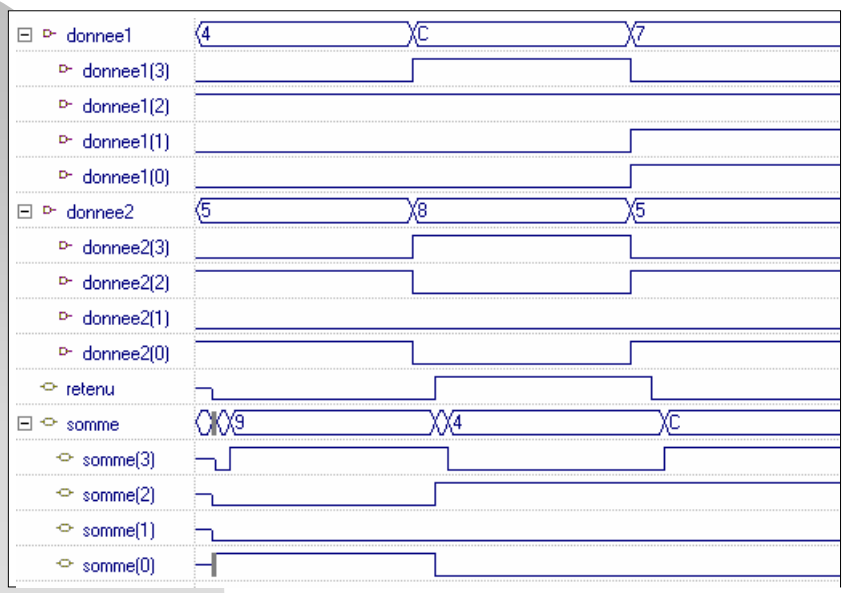
19

L'instanciation de composants
simulation



20

L'instanciation de composants
simulation



L'opérateurs process

- Opérateur permettant d'introduire des instructions séquentielles

process(*liste de signaux***)**
déclaration de variables
begin
…
instructions séquentielles
…
end process;

22

Le process et les instructions

- Assignation

  `<=` `:=`

- test

```
if condition   then instructions
          else instructions
end if
```

23

Le process et les instructions (suite)

- structure selon

```
case variable

   when valeur1   => instructions
   when valeur2   => instructions
   …
   when others    => instructions

end case;
```

24

Le process et les instructions (suite)

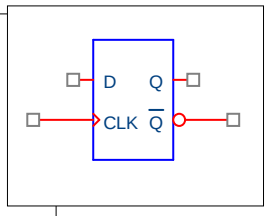
- structures de boucle

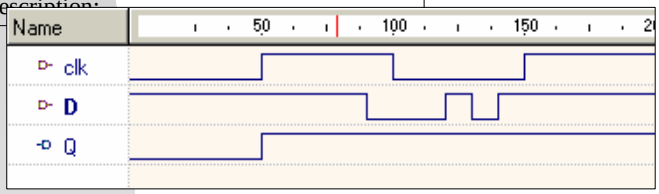
```
étiquette for variable in intervalle loop
          instructions
          end loop étiquette

étiquette while condition loop
          instructions
          end loop étiquette

          next

          exit
```

25

Le process et les instructions exemple d'une bascule D

```
entity basculeD is
port(     clk : in bit;
          D : in bit;
          Q : out bit);
end basculeD;
architecture description of basculeD is
begin
process(clk)
begin
          if(clk'event and clk = '1') then Q <= D;
          end if;
end process;
end description;
```





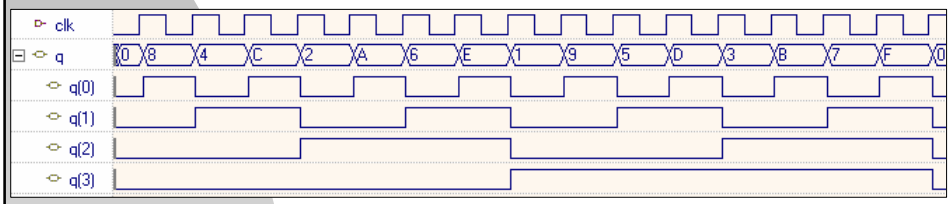
26

Le process et les instructions
exemple d'un compteur

```
library ieee;
use ieee.std_logic_1164.all;
entity compteur is
port( clk : in bit;
            Q : out std_logic_vector(0 to 3));
end compteur;
architecture descrip of compteur is
signal cpt : integer range 0 to 15;
begin
      process(clk)
      begin
            if (clk'event and clk='1') then
                              cpt <= cpt + 1;
            end if;
      end process;
      Q <= std_logic_vector(cpt);
end;
```

27

Le process et les instructions
exemple d'un compteur - simulation



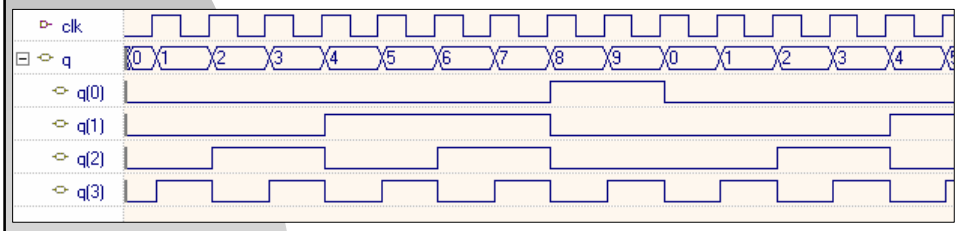
28

Le process et les instructions
compteur modulo 9

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity compteur is
port( clk : in bit;
            Q : out std_logic_vector(0 to 3));
end compteur;
architecture descrip of compteur is
begin
      process(clk)
      begin
            if (clk'event and clk='1') then
                        Q <= Q + 1;
                        if(Q = "1001") then Q <= "0000";
                        end if;
            end if;
      end process;
end descrip;
```

29

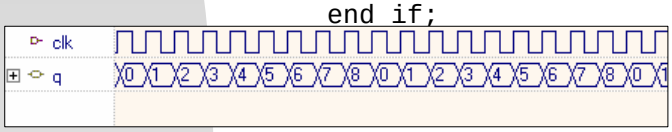
Le process et les instructions
compteur modulo 9 - simulation



30

Le process et les instructions
compteur modulo 9 - correction

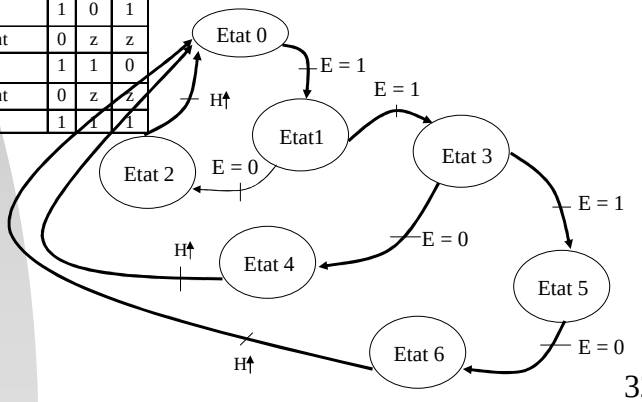
```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity compteur is
port( clk : in bit;
            Q : out std_logic_vector(0 to 3));
end compteur;
architecture descrip of compteur is
begin
      process(clk)
      begin
            if (clk'event and clk='1') then
                        Q <= Q + 1;
                        if(Q = "1000") then Q <= "0000";
                        end if;
end
```



31

Implantation de machine d'état
conception

N° Etat	Action	V	L1	L0
0	Attente	0	z	z
1	Détection d'un front montant	0	z	z
2	Affichage court	1	0	1
3	Détection d'un front	0	z	z
4	Affichage moyen	1	1	0
5	Détection d'un front	0	z	z
6	Affichage long	1	1	1



32

Implantation de machine d'état
implantation - 1

```
library ieee;
use ieee.std_logic_1164.all;
use work.std_arith.all;
entity detection is
port( clk   : in bit;
      E     : in bit;
      L     : out std_logic_vector(1 downto 0);
      V     : out std_logic);
end detection;
architecture descript of detection is
signal etat : std_logic_vector(0 to 2);
begin
process(clk)
begin
```

33

Implantation de machine d'état
implantation - 2

```
    if(clk'event and clk='1') then
          case etat is
          when "000" => if(E='1')    then etat <= "001";
                                     else etat <= "000";
                                       end if;
          when "001" => if(E='1')    then etat <= "011";
                                     else etat <= "010";
                                       end if;
          when "010" =>  etat <= "000";
          when "011" => if(E='1')    then etat <= "101";
                                     else etat <= "100";
                                       end if;
          when "100" => etat <= "000";
          when "101" => if(E='1')    then etat <= "000";
                                     else etat <= "110";
                                       end if;
          when others => etat <= "000";
          end case;
    end if;
end process;
```

34

Implantation de machine d'état
implantation - 3

```
with etat select
L <=  "01"  when  "010",
      "10"  when  "100",
      "11"  when  "110",
      "ZZ"  when  "011",
      "ZZ"  when  others;
with etat select
V <=  '1'         when  "010",
      '1'         when  "100",
      '1'         when  "110",
      '0'         when  others;
end descript;
```

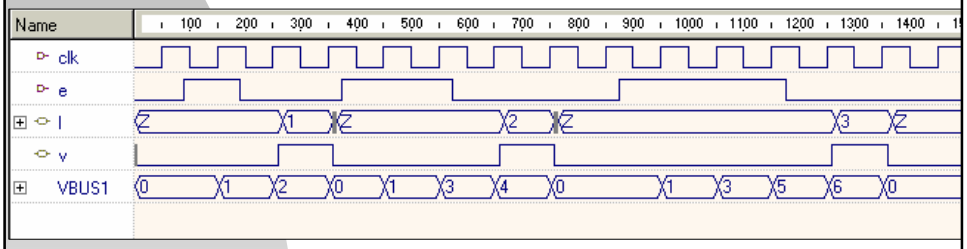
35

Implantation de machine d'état
implantation - 3

```
with etat select
L <=  "01"  when  "010",
      "10"  when  "100",
      "11"  when  "110",
      "ZZ"  when  "011",
      "ZZ"  when  others;
with etat select
V <=  '1'         when  "010",
      '1'         when  "100",
      '1'         when  "110",
      '0'         when  others;
end descript;
```

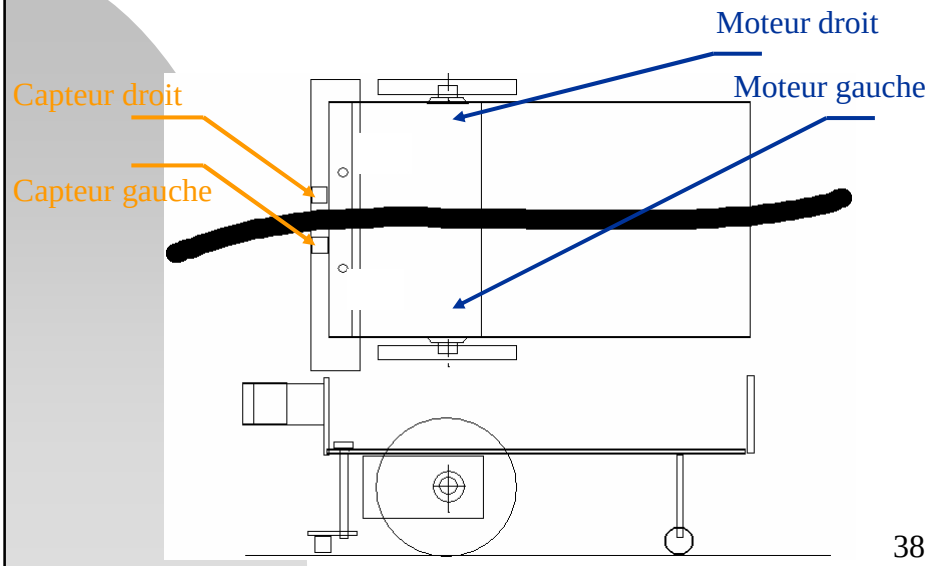
36

Implantation de machine d'état
simulation



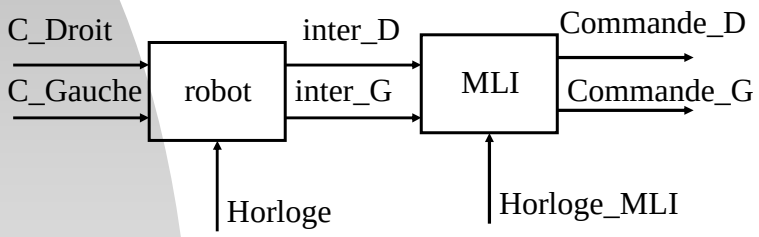
37

Conception modulaire
Package - exemple



38

Conception modulaire
Package – exemple (suite)



39

Conception modulaire
Package – Code MLI

```
library ieee;
use ieee.std_logic_1164.all;
package MLI_pkg is
        component MLI
        port(   clk : in bit;                           -- horloge 128*Tclk=50us
                entree_G : in integer(0 to 127);        -- vitesse moteur G
                entree_D : in integer(0 to 127);        -- vitesse moteur D
                Droit : out bit;                        -- sortie moteur droit
                Gauche: out bit);                               -- sortie moteur
gauche
        end component;
end MLI_pkg;
entity MLI is
port(       clk : in bit; -- horloge 128*Tclk=50us
        entree_G : in integer(0 to 127); -- vitesse moteur G
        entree_D : in integer(0 to 127); -- vitesse moteur D
        Droit : out bit; -- sortie moteur droit
        Gauche: out bit); -- sortie moteur gauche end MLI;
End MLI;
```

40

Conception modulaire
Package – Code MLI (suite)

```
architecture des of MLI is
signal cpt : integer(0 to 127);                    -- compteur interne
begin
process(clk)
begin
        if (clk'event and clk = '1') then          cpt <= cpt + 1;
        if(cpt>entree_D)  then Droit <= '1';
                else Droit <= '0;
        end if;
        if(cpt>entree_G)  then Gauche <= '1';
                               else Gauche <= '0'
        end if;
        end if;
end process;
end des;
```

41

Conception modulaire
Package – Code robot

```
library ieee;
use ieee.std_logic_1164.all;
package gestion_pkg is
        component robot
        port(   clk : in bit;                      -- horloge
                marche : in bit;                   -- mise en marche
                capteur_droit : in bit;            -- capteur droit
                capteur_gauche : in bit;           -- capteur gauche
                M_gauche : out integer(0 to 127);  -- sortie moteur G
                M_droit : out integer(0 to 127));  -- sortie moteur D
        end component;
end gestion_pkg;
entity robot is
port(   clk : in bit;                              -- horloge
        marche : in bit;                           -- mise en marche
        capteur_droit : in bit;                    -- capteur droit
        capteur_gauche : in bit;                   -- capteur gauche
        M_gauche : out integer(0 to 127);          -- sortie moteur G
        M_droit : out integer(0 to 127));          -- sortie moteur D
        end component;
end robot;
```

42

Conception modulaire
Package – Code robot (suite)

```
architecture etat of robot is
signal etat : integer(0 to 3);    -- variable d'état
begin
process(clk)
begin
        if (clk'event and clk = '1') then
        case etat is
        -- état 0 : attente mise en marche
        when 0 =>          if( marche = '1')    then etat <= 1;
                                                else etat <= 0;
                           end if;
```

43

Conception modulaire
Package – Code robot (suite)

```
-- état 1 : suivi de ligne
when 1 => if marche ='1' then
                  if (capteur_droit = '0' and capteur_gauche = '1') then etat <= 2;
                        else if (capteur_droit = '1' and capteur_gauche = '0') then etat <= 3;
                        else if(capteur_droit = '1' and capteur_gauche = '0') then etat <= 0;
                        else      M_gauche <= 64;
                                  M_droit <= 64;
                                  etat <= 1;
                        end if;
                        end if;
                        end if;
        else etat <=0;
        end if;
```

44

Conception modulaire
Package – Code robot (suite)

```
-- état 2 : visage à droite
when 2 => if marche = '1' then
                    if (capteur_droit = '1' and capteur_gauche = '1') then etat <= 1;
                            else if(capteur_droit = '1' and capteur_gauche = '0') then etat <= 3;
                            else    M_droit <= M_droit + 1;
                                    M_gauche <= M_gauche - 1;
                            etat <= 2;
                     end if;
                     end if;
            else etat <=0;
          end if;
```

45

Conception modulaire
Package – Code robot (suite)

```
-- état 3 : visage à gauche
when 3 => if marche = '1' then
                    if (capteur_droit = '1' and capteur_gauche = '1') then etat <= 1;
                            else if(capteur_droit = '0' and capteur_gauche = '1') then etat <= 2;
                            else    M_droit <= M_droit - 1;
                                    M_gauche <= M_gauche + 1;
                                    etat <= 3;
                            end if;
                    end if;
          else etat <=0;
          end if;
-- état 3 : visage à gauche
          when others => etat <= 0;
end case;
end if;
end process;
end etat;
```

46

Conception modulaire
Package – Code commande

```
library ieee;
use ieee.std_logic_1164.all;
use work.gestion_pkg.all;
use work.mli_pkg.all;
entity commande is
port(
-- partie Gestion
        horloge : in bit;              -- horloge gestion capteur
        bouton_marche : in bit;        -- mise en marche
        C_droit  : in bit;             -- capteur droit (suivit de ligne)
        C_gauche  : in bit;            -- capteur gauche (suivit de ligne)

-- partie MLI
        horloge_MLI : in bit;          -- horloge 3MHz (signal de sortie 20KHz)
        Commande_Droit : out bit;      -- sortie moteur droit
        Commande_Gauche : out bit); -- sortie moteur gauche
end commande;
```

47

Conception modulaire
Package – Code commande

```
architecture des of commande is

-- déclaration des signaux internes
signal inter_G      : integer(0 to 127);  -- commande du moteur gauche
signal inter_D      : integer(0 to 127);  -- commande du moteur droit
begin

-- gestion des capteurs
F1: robot
port map(horloge,bouton_marche,C_droit,C_gauche,inter_G,inter_D);

-- génération de la MLI
F2: MLI
port map( horloge_MLI, inter_G, inter_D,Commande_Droit,Commande_Gauche);

end des;
```

48