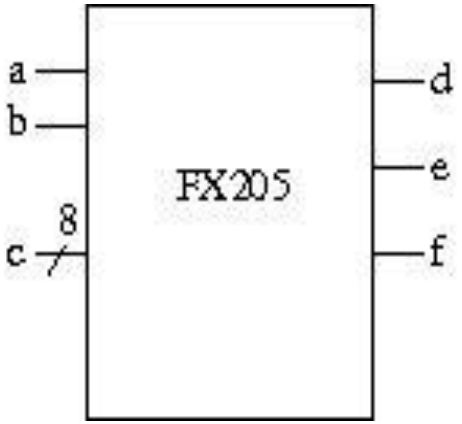
VHDL
EISTI – ING1 2008-2009

- **V**ery High Speed Integrated Circuit **H**ardware **D**escription **L**anguage
- langage de haut niveau
- réalisation matérielle de circuit numérique
  - spécification
  - conception
  - simulation
- modulaire

Syntaxe VHDL (1)

- Interface d'un circuit : entity




```
entity FX205 is
port (
        a : in bit;
        b : in bit;
        c : in bit_vector(0 to 7);
        d : out bit;
        e : out bit;
        f : out bit
);
end FX205;
```

Syntaxe VHDL (2)

- réalisation du circuit : architecture

```
architecture arch_FX205 of FX205 is
-- déclarations types, constantes, signaux
begin
        -- domaine concurrent
        f <= a xor b;

        process(a,c)
        -- déclarations types, constantes,
        -- signaux et variables
        begin
                -- domaine séquentiel
        end process;
end arch FX205;
```

Données

- entrées/sorties définies dans l'entité
  - 3 modes d'accès : in, out, inout
- constantes
  - constant memory_size : integer := 64;
- signaux : fils internes/réalité physique
  - signal Q : bit; -- bit mémoire d'une bascule
- variables : alias
  - variable i : integer;

Types scalaires (1)

- bit : '0', '1'
- std_logic :
  - '0', '1', 'X', 'U', 'Z'
  - librairie ieee.std_logic_1164
- boolean : true, false
- integer
  - a : integer range 0 to 15; -- entier sur 4 bits
  - 679, 2#101#, B"101", 8#672#, O"672", X"FFA0"

Types scalaires (2)

- real : flottants
- type énuméré de n valeurs :
  - sur N bits tel que $2^{N-1} < n <= 2^{N}$
  - exemple :

```
type t_direction is (GAUCHE, DROITE);
signal dir : t_direction;
```

Types complexes (1)

- vecteurs :
  - bit_vector
    - tab1 : bit_vector(0 to 7);
    - tab2 : bit_vector(7 downto 0);
    - "01111010"
  - std_logic_vector (idem bit_vector)
  - array

    type t_memory is array(0 to 63) of std_logic_vector(7 downto 0);

    signal mem : t_memory;

Types complexes (2)

- enregistrements :

```
-- déclaration du type tag de bloc pour mémoire cache
type t_block_tag is record
                        modified : boolean;
                        valid : boolean;
                        address : std_logic_vector(4 downto 0);
                end record;

-- déclaration d'un tag de bloc
signal unTag : t_block_tag;
```

Opérateurs (1)

- logiques :
  - and, or, xor, not, nand, nor
  - >, <, =, /=
- arithmétiques :
  - *, /, +, -, mod, rem, **, abs

Opérateurs (2)

- vecteurs :
  - concaténation : &
    - "01"&"11001" --> "0111001"
  - accès :
    - 1 élément : tab(i)
    - n éléments : tab(4 downto 2)

Instructions concurrentes

- affectation de signal
  - c <= a xor b;
- affectation conditionelle
  - c <= '0' when a = b else '1';
- process
- instance de composant

Instructions séquentielles (1)

- affectation de signal : **<=**
- affectation de variable : **:=**

  v := une_valeur;
- conditionelle : **if**

```
if (a<b) then
        -- bloc 1

elsif (a=b) then
        -- bloc 2

else
        -- bloc 3

end if;
```

Instructions séquentielles (2)

- boucles :

```
for i in 0 to 255 loop
        -- bloc --

end loop;
```

```
while (i < 255) loop
        -- bloc --

end loop;
```

  - attention à l'aspect physique
- delai (simulation) :
  - wait for 20 ns;
- null; -- instruction qui ne fait rien

Instructions séquentielles (3)

- case (bit, bit_vector, integer) :

```
case tab is
        when "000" =>
                -- bloc 1

        when "001" =>
                -- bloc 2

        when "010" =>
                -- bloc 3

        when others =>
                -- bloc 4
end case;
```

Programme OR_ent.vhdl

```
library ieee;
use ieee.std_logic_1164.all;
-------------------------------------
entity OR_ent is
port(
        x: in std_logic;
        y: in std_logic;
        F: out std_logic
);
end OR_ent;
```

1 entité et 1 architecture (au choix)

```
architecture OR_arch1 of OR_ent is
begin

    process(x, y)
    begin
        -- compare to truth table
        if ((x='0') and (y='0')) then
            F <= '0';
        else
            F <= '1';
        end if;
    end process;

end OR_arch1;
```

```
architecture OR_arch2 of OR_ent is
begin

    F <= x or y;

end OR_arch2;
```

Composant de test : tb_or_ent.vhdl

```
library ieee;
use ieee.std_logic_1164.all;

entity tb_or_ent is                -- entity declaration
end tb_or_ent;

architecture arch_tb_or_ent of tb_or_ent is
    -- signaux necessaires a la simulation
    -- pour connecter les composants
    signal x_in, y_in, f_out : std_logic;

    -- rappel de la définition du composant
    component OR_ent is
            port(    x: in std_logic;
            y: in std_logic;
            F: out std_logic
            );
    end component;
begin
    -- instanciation d'une porte OR_ent
    u_or : OR_ent port map (
            x => x_in,
            y => y_in,
            F => f_out);

    process     -- simulation
    begin
            x_in <= '0';
            y_in <= '0';
            wait for 20 ns;
            x_in <= '0';
            y_in <= '1';
            wait for 20 ns;
            x_in <= '1';
            y_in <= '0';
            wait for 20 ns;
            x_in <= '1';
            y_in <= '1';
            wait for 20 ns;
    end process;
end arch_tb_or_ent;
```

Un process permet d'affecter des signaux de manière complexe :

- en décomposant le calcul
- en prenant en compte plusieurs cas, l'état du système
- en utilisant des boucles finies pour des affectations multiples

Un process s'évalue :

- en un temps infiniment petit
  - hors utilisation de wait (séparateur de temps élémentaires)
  - signaux affectés **une seule fois** en fin de process
- en boucle

Un process est paramétré par une liste de sensibilités :

- changement sur **un** signal de la liste => évaluation du process
- important : liste de sensibilités **≠** liste de paramètres
- un process peut utiliser un signal n'appartenant pas à la liste

```
signal a, b, c, d, e : std_logic;
```

```
process(a, b)
begin
        if a=b then
                c <= '1'
        else
                c <= '0';
        end if;
end;
```

- c n'est affecté qu'une seule fois par éxécution du process
- le process est évalué sur un changement de a et/ou b
- à la fin de son éxécution, le process se met en attente :
    - à l'écoute de sa liste de sensibilité
    - au début de son code

```
process(a)
        variable v1, v2 : std_logic;
begin
        b <= a;
        c <= b;
        --
        v1 := a;
        d <= v1;
        v1 := not(v1);
        e <= v1;
end;
```

- b et c ne seront affectés qu'à la fin du process
    - b avec la valeur de a au début du process
    - c avec l'ancienne valeur de b (pas celle de a)
- les variables sont évaluées séquentiellement
    - v1 reçoit la valeur du signal a
    - d recevra (à la fin du process) la valeur de v1 (i.e. a)
    - v1 reçoit son complément (i.e. not(a))
    - e recevra la valeur de v1 (i.e. not(a))
- application avec (a,b) = (1,1) et a passe de 1 à 0
    - (a,b,c,d,e) = (0, 0, 1, 0, 1)

Un process permet la description d'un composant synchrone.

La liste de sensibilité ne contient que :

- le signal d'horloge
- les signaux asynchrones d'initialisation (set, reset)

Vous devez préciser sur quel front d'horloge se passe le traitement.

Canevas général d'un **composant synchrone** :

```
process(clock, set, reset)
-- le process est déclenché sur un changement sur clock ou set ou reset
begin
        if (set='1') then
                -- traitement du set asynchrone actif sur niveau haut
                -- le set est prioritaire sur le reset et l'horloge
        elsif (reset='1') then
                -- traitement du reset asynchrone actif sur niveau haut
                -- le reset est prioritaire sur l'horloge
        elsif (clock'event and clock='1') then
                -- traitement du front montant de l'horloge
        end if;
end process;
```

Réalisation d'une bascule

Un circuit séquentiel a des sorties qui dépendent :

- de ses entrées
- de son état précédent

On ne peut pas lire une sortie d'un circuit
=> nécessité d'utiliser un signal interne

```
entity FX40 is port map (
        E : in std_logic;
        S : out std_logic);
end FX40;
```

```
architecture archi_comp of comp is
begin
        S <= f(S, E);
end archi_comp;
```

=>

```
architecture archi_comp of comp is
        signal mem : std_logic;
begin
        mem <= f(mem, E);
        S <= mem;
end archi_comp;
```

```
entity JK_flipflop is
port (
        -- entrées de commande
        J : in std_logic;
        K : in std_logic;
        -- horloge sur front descendant
        clock : in std_logic;
        -- initialisation active à niveau haut
        set : in std_logic;

        -- sorties de la bascule
        Q : out std_logic;
        notQ : out std_logic
);
end JK_flipflop;
```

```
architecture archi_JK_flipflop of JK_flipflop is
        signal mem : std_logic; -- mémoire interne
begin
        process (clock, set)
        begin
                if (set='1') then
                        mem <= '1';
                elsif (clock'event      and  clock='0') then
                        if (J='0' and K='1') then
                                mem <= '0';
                        ...
                        if (J='1' and K='1') then
                                mem <= not(mem);
                        ...
                end if;
        end process;

        -- branchement des sorties /mémoire
        Q  <=  mem;
        notQ <=  not(mem);

end archi_JK_flipflop;
```

Test de la bascule JK :

1 – signal d'horloge (process dédié)

```
process
begin
        clock <= '0';
        wait for 10 ns;
        clock <= '1';
        wait for 10 ns;
end process;
```

2 – jeu d'essai pour les autres entrées (2ème process dans la même architecture)

```
process
-- changer les valeurs aux temps 10, 30, 50, 70, etc ...
begin
        J <= '0';
        ...
end process;
```

VHDL offre plusieurs niveaux de description :

- **structurelle** : vue d'un composant comme la composition de sous-éléments. On obtient une architecture avec des instances de composants connectés entre eux.

- **comportementale** : à travers 1 ou plusieurs process, on définit les sorties en fonction des entrées suivant un algorithme.

- **flots de données** : sorties exprimées en fonction des entrées grâce à des affectations de signaux dans le domaine concurrent.

En VHDL, il est possible de faire une description d'un composant plus ou moins proche de l'aspect matériel :

- **spécification d'un composant** : permet de simuler le comportement général du circuit
- **composant synthétisable** : description d'un composant réel :
  - piloté par une horloge et un nombre limité de signaux asynchrones (set, reset)
  - boucles dépliables : bornes calculables

Pour basculer entre plusieurs architectures, on utilise une configuration :

```
configuration tb_or_ent_conf1 of tb_or_ent is
        for arch_tb_or_ent
                for u_or : OR_ent
                        use entity monWorkspace.OR_ent (OR_arch1);
                end for;
        end for;
end tb_or_ent_conf1;

configuration tb_or_ent_conf2 of tb_or_ent is
        for arch_tb_or_ent
                for u_or : OR_ent
                        use entity monWorkspace.OR_ent (OR_arch2);
                end for;
        end for;
end tb_or_ent_conf2;
```

Définition de package :

```
package pack_Von_Neumann is

-- type des mots utilisés par la machine de Von Neumann
subtype t_mot is std_logic_vector(3 downto 0);

-- type des opérations de l'ALU
type t_operation is (ADD, ADC, MULT, DIV, SUB, MODULO);

end pack_Von_Neumann;
```

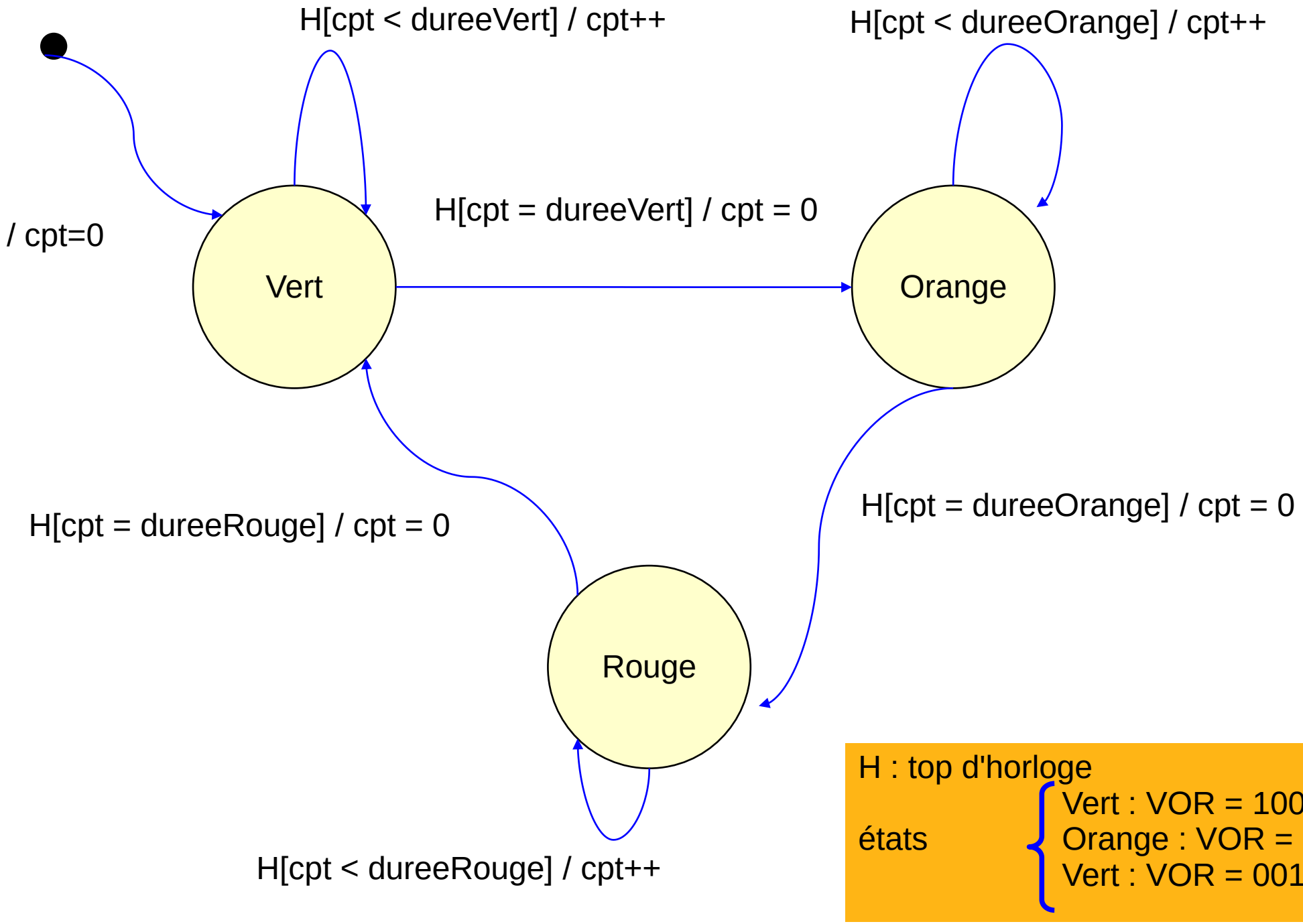
Utilisation du package :

```
use monWorkspace.pack_Von_Neumann.all;

....
signal op : t_operation;
```

Composants à états : exemple des feux tricolores

```
entity feuxTricolore is
port (
H : in bit; -- horloge
reset : in bit; -- signal réinitialisation (actif niveau bas)
-- sorties : lampes à allumer ('1') / éteindre ('0')
lampeVerte : out bit;
lampeOrange : out bit;
lampeRouge : out bit
);
end feuxTricolore;
```

Les lampes sont allumées suivant l'état interne du feu. Le feu reste dans chaque état un certain nombre de cycles d'horloge. On représente le fonctionnement du feu avec un diagramme d'état.


H[cpt < dureeVert] / cpt++
H[cpt < dureeOrange] / cpt++
/ cpt=0
Vert
H[cpt = dureeVert] / cpt = 0
Orange
H[cpt = dureeRouge] / cpt = 0
H[cpt = dureeOrange] / cpt = 0
Rouge
H[cpt < dureeRouge] / cpt++
H : top d'horloge
états
Vert : VOR = 100
Orange : VOR = 010
Vert : VOR = 001

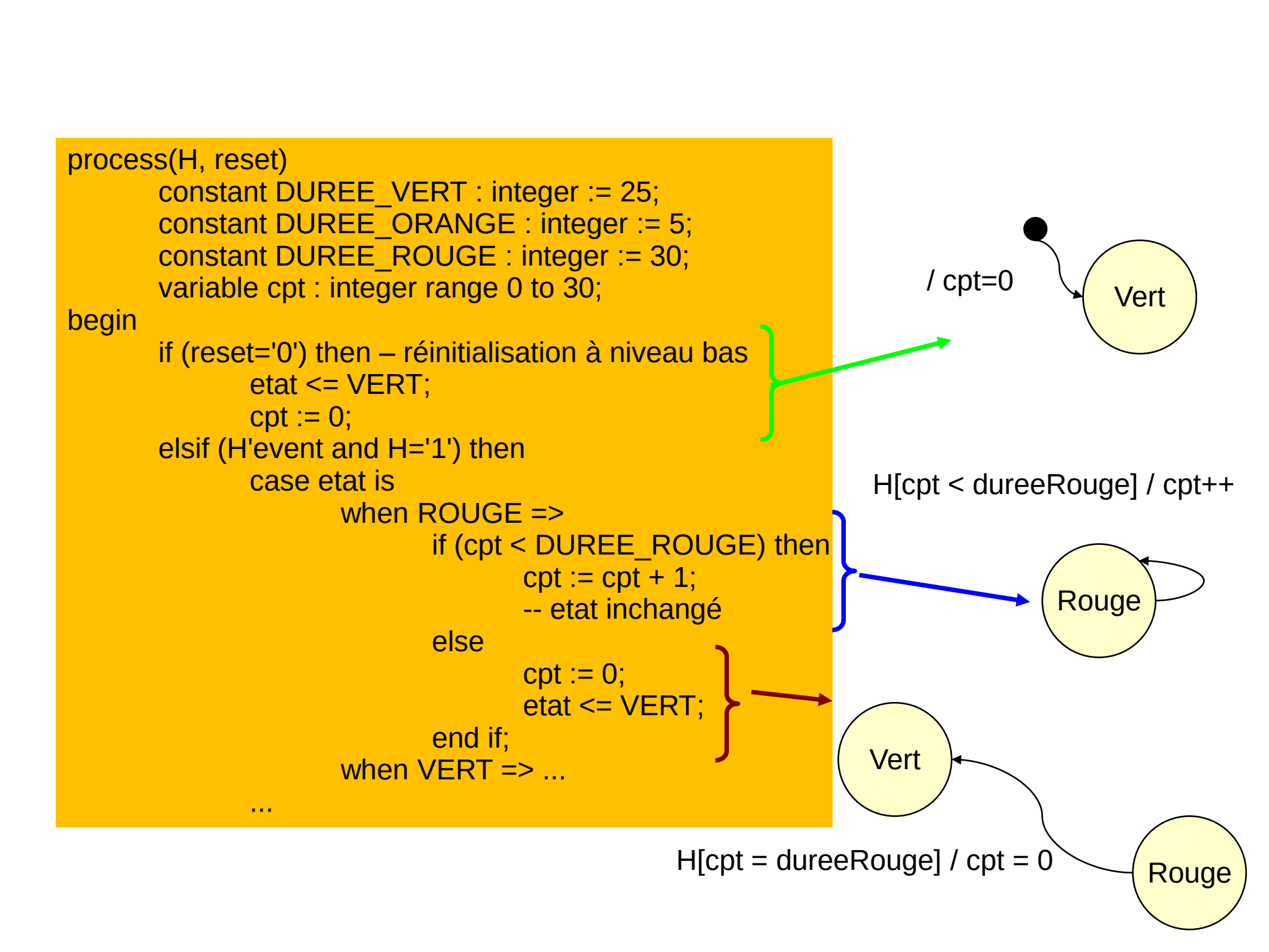
Un feu tricolore se comporte différemment suivant son état : ROUGE, VERT, ORANGE

```
type t_etat is (ROUGE, VERT, ORANGE);

signal etat : t_etat;

process(H, reset)
begin
        if (reset='0') then – réinitialisation à niveau bas
                etat <= VERT;
        elsif (H'event and H='1') then
                case etat is
                        when ROUGE => ...
                        when VERT => ...
                        when ORANGE => ..
                end case;
        end if;
end;
```

type énuméré pour représenter les différents états et un signal pour mémoriser l'état courant entre 2 tops d'horloges

A chaque top d'horloge,on regarde dans quel état est le composant pour appliquer le traitement adéquat

```
process(H, reset)
        constant DUREE_VERT : integer := 25;
        constant DUREE_ORANGE : integer := 5;
        constant DUREE_ROUGE : integer := 30;
        variable cpt : integer range 0 to 30;
begin
        if (reset='0') then – réinitialisation à niveau bas
                etat <= VERT;
                cpt := 0;
        elsif (H'event and H='1') then
                case etat is
                        when ROUGE =>
                                if (cpt < DUREE_ROUGE) then
                                        cpt := cpt + 1;
                                        -- etat inchangé
                                else
                                        cpt := 0;
                                        etat <= VERT;
                                end if;
                        when VERT => ...
                ...
```



```
process(H, reset)
begin
        if (reset='0') then – réinitialisation à niveau bas
                etat <= VERT;
                cpt := 0;
        elsif (H'event and H='1') then
                case etat is
                        when ROUGE =>
                                if (cpt < DUREE_ROUGE) then
                                        cpt := cpt + 1;
                                        -- etat inchangé
                                else
                                        cpt := 0;
                                        etat <= VERT;
                                end if;
                        when VERT => ...
                ...
end process;

lampeVerte <= '1' when (etat = VERT) else '0';
...
```

lampes allumées
en fonction de l'état