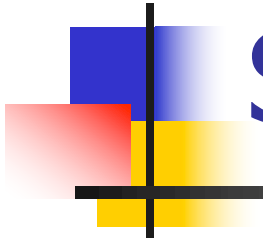
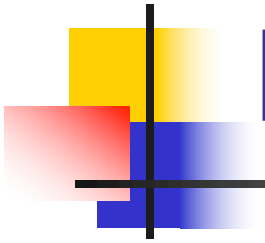


Synthèse Logique

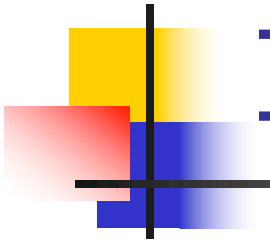
Chapitre 3

Codage de FSM



Plan

- **Introduction**
- Moore v.s. Mealy
- Exercices

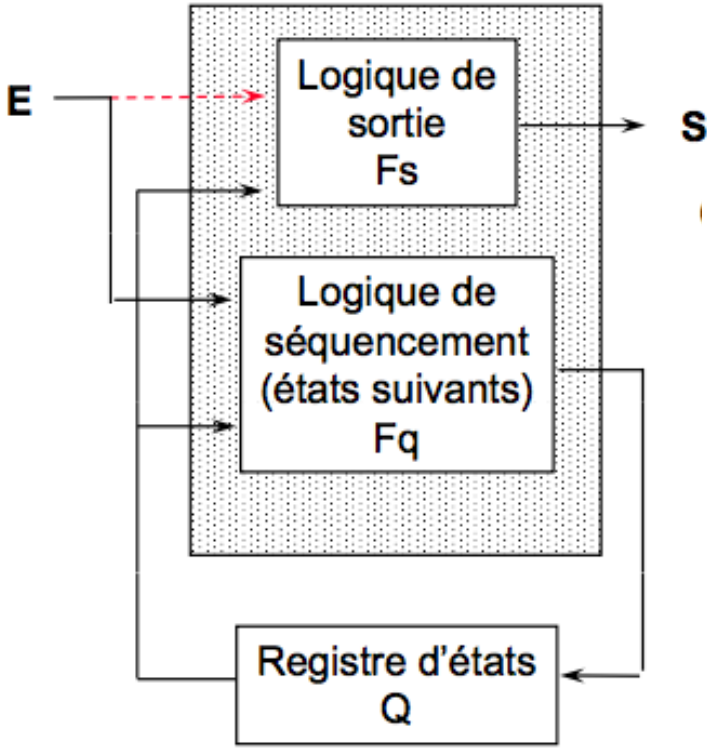


Introduction

- Les machines à états (FSM) sont des composants couramment utilisés, nous allons voir comment :
  - Les décrire
  - En faire la synthèse pour atteindre les performances optimales (notion de compromis Temps/Fréquence)
  - Les utiliser dans un système complexe (avec une hiérarchie)
  - Nous terminerons ce chapitre et le cours par l'étude d'exemples de conception

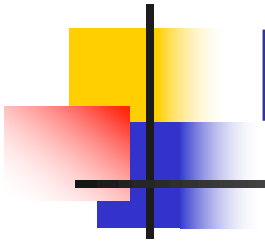
Moore v.s. Mealy

- Deux blocs combinatoires (calcul de l'état suivant et des sorties)
- Un registre d'état à N entrées parallèles
- Etats codés sur N bits, N = $\log_2$ (Q)



Quintuplet : (E, S, Q, Fs, Fq)

E: Ensemble des entrées primaires
S : Ensemble des sorties primaires
Q : Ensemble des états
Fs : Fonction de sortie (Moore / Mealy)
Fs = f (Q) => Moore
Fs = f (Q, E) => Mealy
Fq : Fonction de transition d'états



Méthode d'Huffman-Mealy

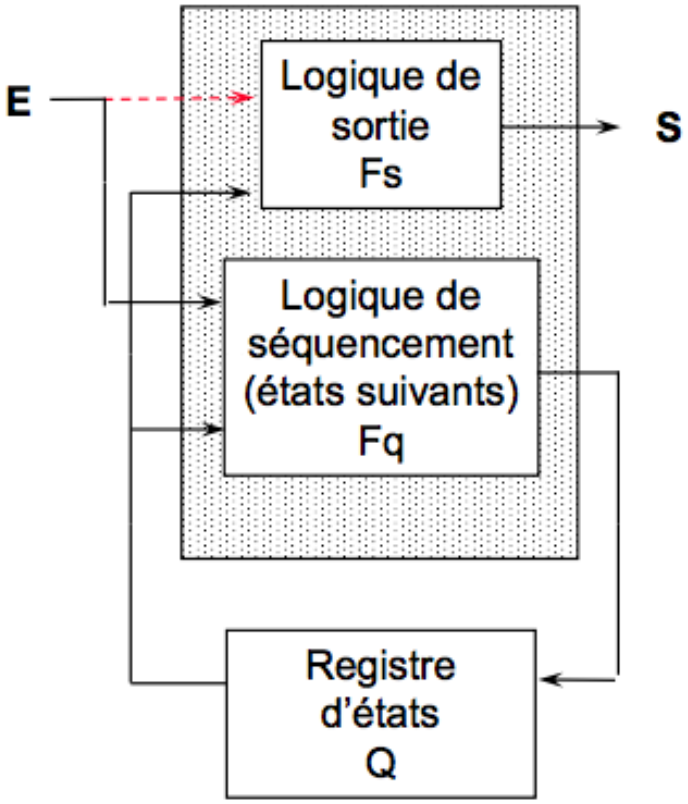
- Modélisation du cahier des charges
  - Graphe d'état
  - Table d'état
- Minimisation du nombre d'états
  - Règles de minimisation
  - Détermination du nombre de bascules minimum
- Codage des états
  - Codage des états
  - Codage des entrées de bascules
- Synthèse
  - Synthèse des entrées de bascules
  - Synthèse des sorties

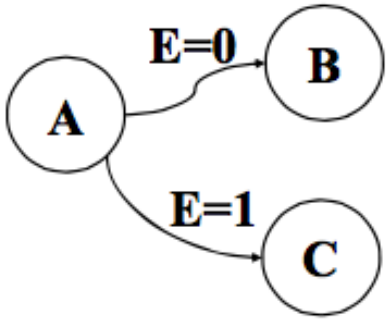
Méthode d'Huffman-Mealy

- Modélisation du cahier des charges
  - Graphe d'état
  - Table d'état
- ~~Minimisation du nombre d'états~~
  - ~~Règles de minimisation~~
  - ~~Détermination du nombre de bascules minimum~~
- ~~Codage des états~~
  - ~~Codage des états~~
  - ~~Codage des entrées de bascules~~
- ~~Synthèse~~
  - ~~Synthèse des entrées de bascules~~
  - ~~Synthèse des sorties~~

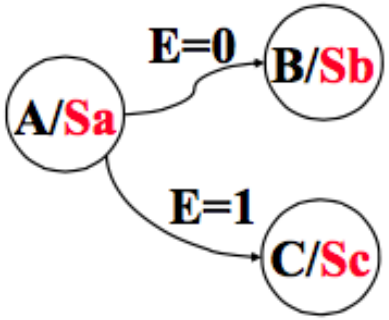
Codage VHDL

Graphe d'Etat

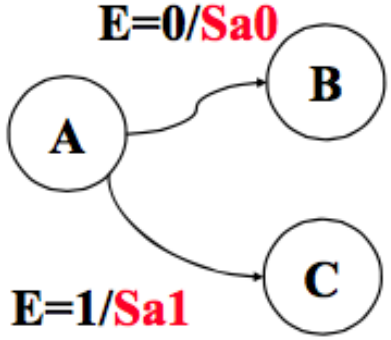




Machine de Moore



Machine de Mealy



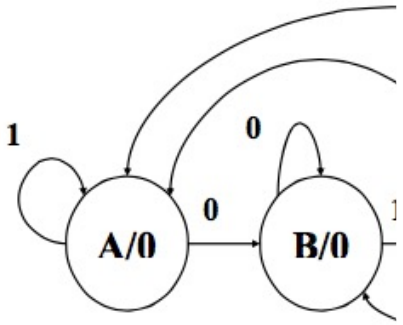
Exemple : Modélisation

- Cahier des charges : Le système considéré a une entrée (E) et une sortie (S). Il reçoit sur son entrée des bits arrivant en série. La sortie (S) doit passer à 1 chaque fois qu'une séquence 010 apparait sur l'entrée (E) puis repasser à 0 sur le bit suivant quel que soit sa valeur.

Exemple : Modélisation

- Cahier des charges : Le système considéré a une entrée (E) et une sortie (S). Il reçoit sur son entrée des bits arrivant en série. La sortie (S) doit passer à 1 chaque fois qu'une séquence 010 apparait sur l'entrée (E) puis repasser à 0 sur le bit suivant quel que soit sa valeur.

Graphe d'état (Moore)



Exemple : Modélisation

- Cahier des charges : Le système considéré a une entrée (E) et une sortie (S). Il reçoit sur son entrée des bits arrivant en série. La sortie (S) doit passer à 1 chaque fois qu'une séquence 010 apparait sur l'entrée (E) puis repasser à 0 sur le bit suivant quel que soit sa valeur.

Graphe d'état (Moore)

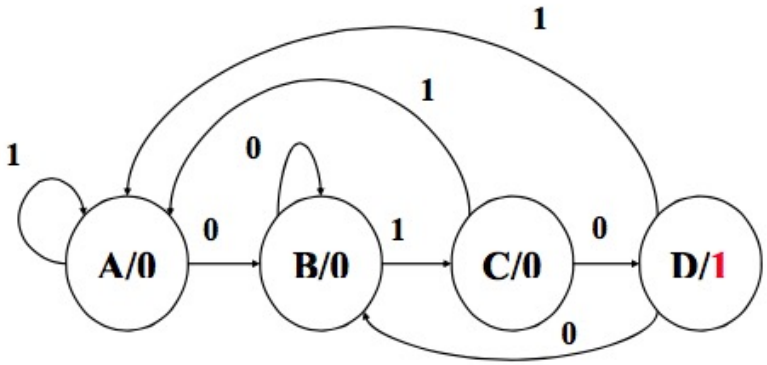
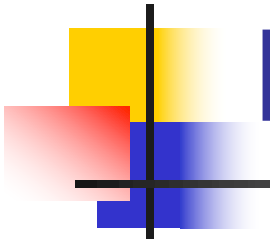


Table d'état

Etats	Etats Suivants		Sortie
	0	1	
A	B	A	0
B	B	C	0
C	D	A	0
D	B	A	1



Exemple : Codage VHDL

- FSM à deux process

```
-- déclaration des ports d'entrées/sorties de la FSM
entity example is port (
    h, e    : in bit;
    s       : out bit);
end example;

architecture state_machine of example is
-- Déclaration d'un type énuméré composé des noms des états
    type StateType is (A, B, C, D);
-- Déclaration de deux signaux du type précédent
    signal present_state, next_state : StateType;
begin
```

Exemple : Codage VHDL

```
state_comb : process(present_state, E)
begin
      case present_state is
            when A => S <= '0' ;
                        if E = '0' then next_state <= B ;
                        else next_state <= A ;
                        end if ;
            when B => S <= '0' ;
                        if E = '1' then next_state <= C ;
                        else next_state <= B ;
                        end if ;
            when C => S <= '0' ;
                        if E = '0' then next_state <= D ;
                        else next_state <= A ;
                        end if ;
            when D => S <= '1' ;
                        if E = '0' then next_state <= B ;
                        else next_state <= A ;
                        end if ;
      end case;
end process state_comb;
```

On définit quand on est dans un état **toutes les sorties** et **l'état suivant** en fonction des entrées

Exemple : Codage VHDL

```
state_clocked : process(h)
begin
    if (h'event and h= '1') then
            present_state <= next_state ;
    end if;
end process state_clocked;

end architecture state_machine;
```

Second process qui décrit le passage d'un état à l'autre sur les fronts montants de l'horloge

- Remarque :
  - Cette description correspond à une machine de Moore car les sorties dépendent uniquement de l'état de la machine et pas des entrées.
  - Pour une machine de Mealy, on aurait des if…then ou des case… pour affecter les sorties en fonction des entrées dans le premier process

Comparaison Moore / Mealy

Graphe d'état (Moore)

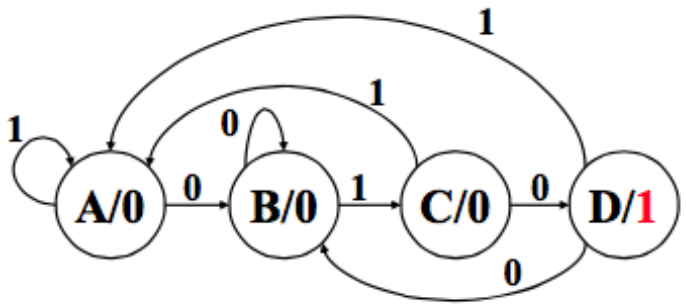


Table d'état

Etats	Etats Suivants		Sortie
	0	1	
A	B	A	0
B	B	C	0
C	D	A	0
D	B	A	1

Graphe d'état (Mealy)

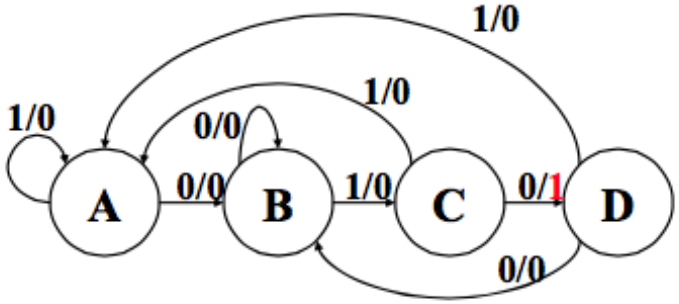


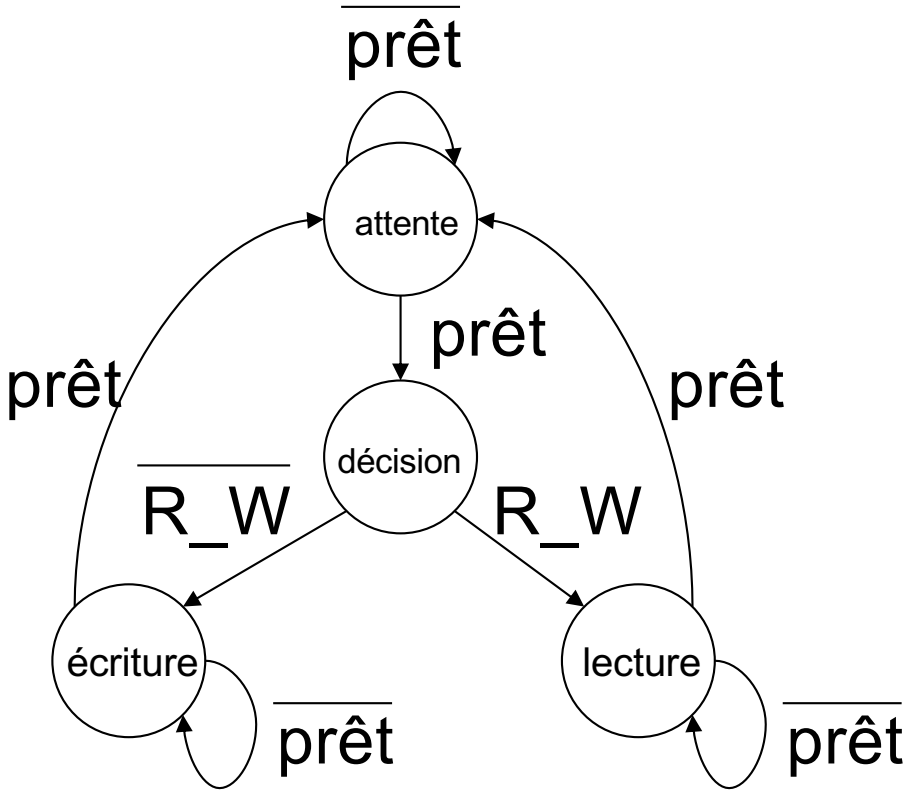
Table d'état

Etats	Etats Suivants		Sortie	
	0	1	0	1
A	B	A	0	0
B	B	C	0	0
C	D	A	1	0
D	B	A	0	0

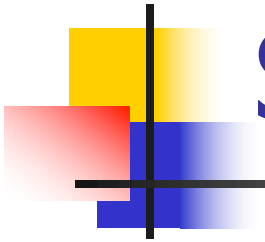
Comparaison Moore / Mealy

```
state_comb : process(present_state, E)
begin
      case present_state is
            when A => if E = '0' then S <= '0' ; next_state <= B ;
                      else S <= '0' ; next_state <= A ;
                      end if ;
            when B => if E = '0' then S <= '0' ; next_state <= B ;
                      else S <= '0' ; next_state <= C ;
                      end if ;
            when C => if E = '0' then S <= '1' ; next_state <= D ;
                      else S <= '0' ; next_state <= A ;
                      end if ;
            when D => if E = '0' then S <= '0' ; next_state <= B ;
                      else S <= '0' ; next_state <= A ;
                      end if ;
      end case;
end process state_comb;
```

Exercice 1



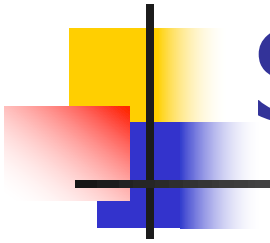
États	Sorties OE	WE
Attente	0	0
Décision	0	0
Écriture	0	1
Lecture	1	0



Solution

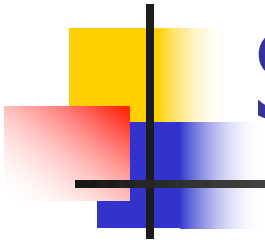
```
-- déclaration des ports d'entrées/sorties de la FSM
entity example is port (
     read_write, ready, clk      : in bit;
     oe, we                      : out bit);
end example;

architecture state_machine of example is
-- Déclaration d'un type énuméré composé des noms des états
     type StateType is (attente, decision, read, write);
-- Déclaration de deux signaux du type précédent
     signal present_state, next_state : StateType;
begin
```



Solution

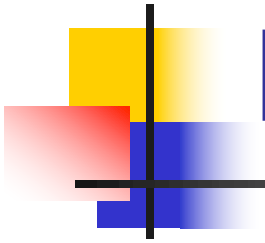
```
state_comb:process(present_state, read_write, ready)
begin
      case present_state is
            when attente=> oe <= '0'; we <= '0';
                        if ready = '1' then next_state <= decision;
                        else next_state <= attente;
                        end if;
            when decision => oe <= '0'; we <= '0';
                        if (read_write = '1') then next_state <= read;
                        else next_state <= write;
                        end if;
            when read => oe <= `1'; we <= `0';
                        if (ready = '1') then next_state <= attente;
                        else next_state <= read;
                        end if;
            when write => oe <= '0'; we <= `1';
                        if (ready = '1') then next_state <= attente;
                        else next_state <= write;
                        end if;
      end case;
end process state_comb;
```



Solution

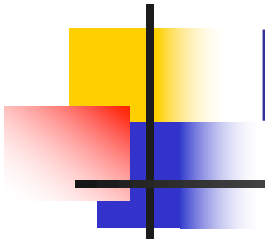
```
state_clocked:process(clk) begin
    if (clk'event and clk = '1') then
            present_state <= next_state;
    end if;
end process state_clocked;

end architecture state_machine;
```



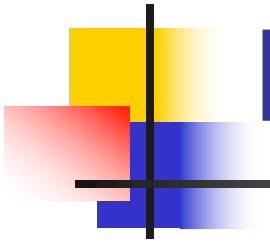
Remarque

- Noter que l'utilisation de **if**…**then** au lieu de **case**… est aussi possible dans le premier **process** pour tester le *present_state*.
- Cependant, le **if**…**then** ajoute une notion de priorité et donc de la logique supplémentaire…



Description de FSM

- Problème de la mémorisation implicite :
  - Dans un process « non clocké » (combinatoire), si dans un case… un signal est spécifié seulement sous certaines conditions, alors **la synthèse produit des bascules (non désirées!) pour mémoriser l'état du signal dans tous les cas non spécifiés**
  - Dans le process state_comb (« non clocké »), **il faut affecter toutes les sorties dans toutes les conditions**



Description de FSM

- Problème du reset :
  - La simulation utilise comme valeur initiale les valeurs à gauche des énumérés : ici, attenteest l'état initial de la machine
  - Après synthèse, l'état initial peut être n'importe quel état (et dans certains cas, cet état peut ne pas être atteignable par la machine), aussi il est recommandé **d'ajouter un reset dans l'un ou l'autre des deux process**

Description de FSM

- Solution Synchrone

```
…
state_comb:process(reset, present_state, read_write, ready)
        begin
        if (reset='1') then
                oe <= '0'; we <= '0';
                next_state <= attente;
        else
                case present_state is
                        when attente=> oe <= '0'; we <= '0';
                                if ready = '1' then
                        ….
                end case;
        end if;
end process state_comb;
…
```

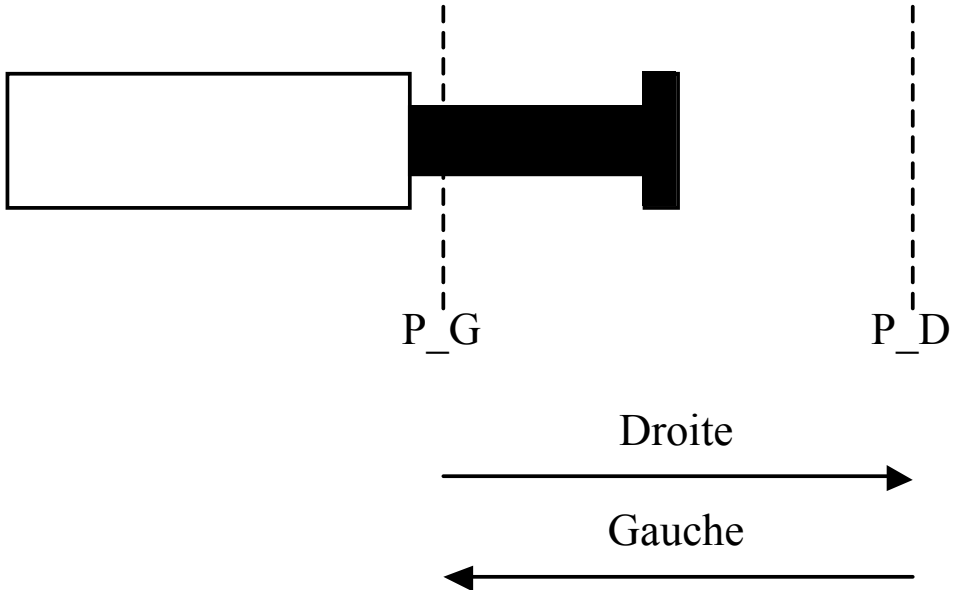
Description de FSM

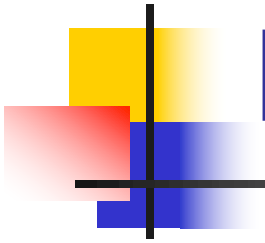
- Solution Asynchrone

```
…
state_clocked:process(clk, reset)
begin
        if (reset=‘1’) then
                present_state <= attente;
        elsif (clk'event and clk='1') then
                present_state <=
next_state;
        end if;
end process state_clocked;
…
```

Exercice 2

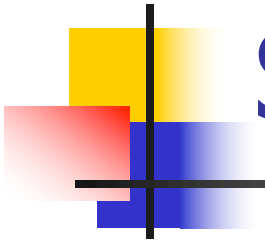
- Considérer le vérin de la figure ci-dessous. Les capteurs P_D et P_G sont à 1 quand le vérin est respectivement à droite et à gauche. Les ordres Droite et Gauche font sortir et rentrer le vérin respectivement. La commande Start lance le système.





Exercice

- Cahier des charges
  - Le vérin sort jusqu'à la position P_D.
  - Il attend 3 cycles d'horloge
  - Puis rentre pour atteindre la position P_G
  - Attend encore 3 cycles d'horloge
  - La commande Start relance le même cycle.
- Exercice :
  - Représenter le fonctionnement du vérin sous forme de graphe d'états
  - Donner le code VHDL du système.



Solution

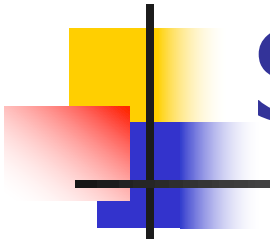
```
entity verin is
port (      clk, reset, start, PosD, PosG : in  bit;
            Droite, Gauche : out  bit);
end verin;



architecture bhv of verin is

type state is (A, D, AD, G, AG);
signal present_state, next_state : state;
signal cpt : bit_vector (1 downto 0);

begin
```



Solution

```
process (start, PosD, PosG, present_state, cpt)
begin
        case present_state is
                when A => Droite <= '0'; Gauche <= '0';
                        if start = '1' then next_state <= D;
                        else next_state <= A;
                        end if;
                when D => Droite <= '1'; Gauche <= '0';
                        if PosD = '1' then next_state <= AD; cpt <= "00";
                        else next_state <= D;
                        end if;
                when AD => Droite <= '0'; Gauche <= '0';
                        if cpt = "11" then next_state <= G;
                        else next_state <= AD;
                        end if;
                when G => Droite <= '0'; Gauche <= '1';
                        if PosG = '1' then next_state <= AG; cpt <= "00";
                        else next_state <= G;
                        end if;
                when AG => Droite <= '0'; Gauche <= '0';
                        if cpt = "11" then next_state <= A;
                        else next_state <= AG;
                        end if;
        end case;
end process;
```

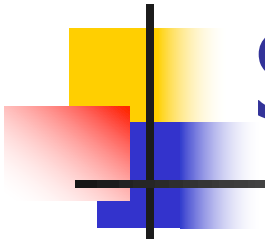
Solution

```
        process (clk, reset)
        begin
                if reset = '1' then
                        present_state <= A;
                        cpt <= "00";
                elsif clk'event and clk = '1' then
                        present_state <= next_state;
                        cpt <= cpt + 1;
                end if;
        end process;

end bhv;
```

Solution correcte

```
process (start, PosD, PosG, present_state, cpt)
begin
        case present_state is
                when A => Droite <= '0'; Gauche <= '0';
                        if start = '1' then next_state <= D;
                        else next_state <= A;
                        end if;
                when D => Droite <= '1'; Gauche <= '0';
                        if PosD = '1' then next_state <= AD;
                        else next_state <= D;
                        end if;
                when AD => Droite <= '0'; Gauche <= '0';
                        if cpt = "11" then next_state <= G;
                        else next_state <= AD;
                        end if;
                when G => Droite <= '0'; Gauche <= '1';
                        if PosG = '1' then next_state <= AG;
                        else next_state <= G;
                        end if;
                when AG => Droite <= '0'; Gauche <= '0';
                        if cpt = "11" then next_state <= A;
                        else next_state <= AG;
                        end if;
        end case;
end process;
```



Solution correcte

```
process (clk, reset)
begin
        if reset = '1' then
                present_state <= A;
                cpt <= "00";
        elsif clk'event and clk = '1' then
                present_state <= next_state;
                        -- reset du compteur si changement d'état
                if present_state /= next_state then cpt <= "00";
                        -- incrémentation du compteur
                else cpt <= cpt + 1;
                end if;
        end if;
end process;

end bhv;
```