

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

Université de Carthage

Ecole Nationale d'Ingénieurs de Carthage



المدرسة الوطنية للمهندسين بقرطاج

Ecole Nationale d'Ingénieurs de Carthage

وزارة التعليم العالي و البحث العلمي

جامعة قرطاج

المدرسة الوطنية للمهندسين بقرطاج

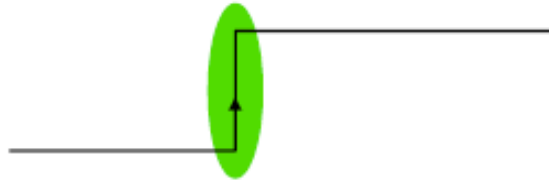
Systèmes embarqués

Niveau: 2 ING INFO

Année Universitaire
2020/2021

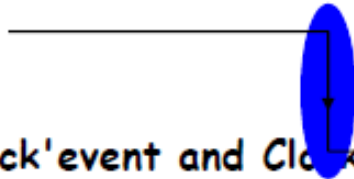
Attribut 'event

- détection d'un front montant :



– if (Clock'event and Clock = '1') then

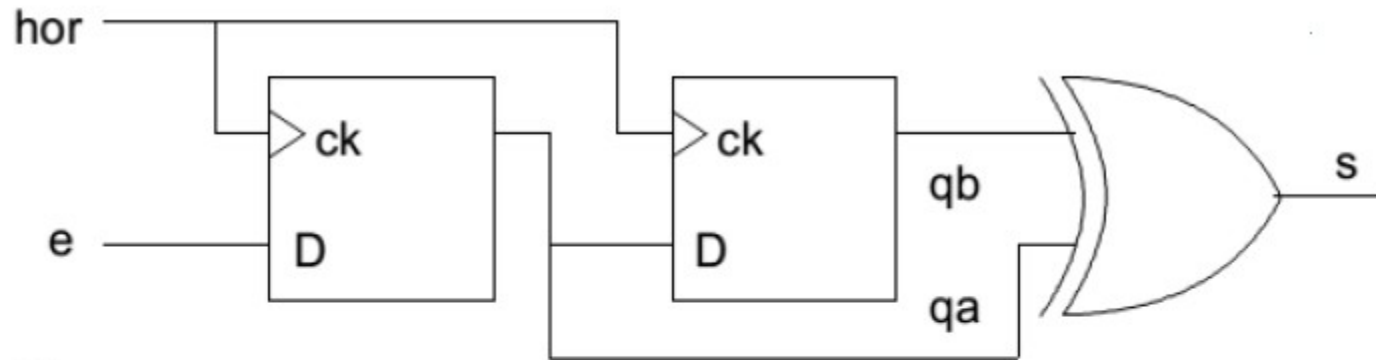
- détection d'un front descendant :



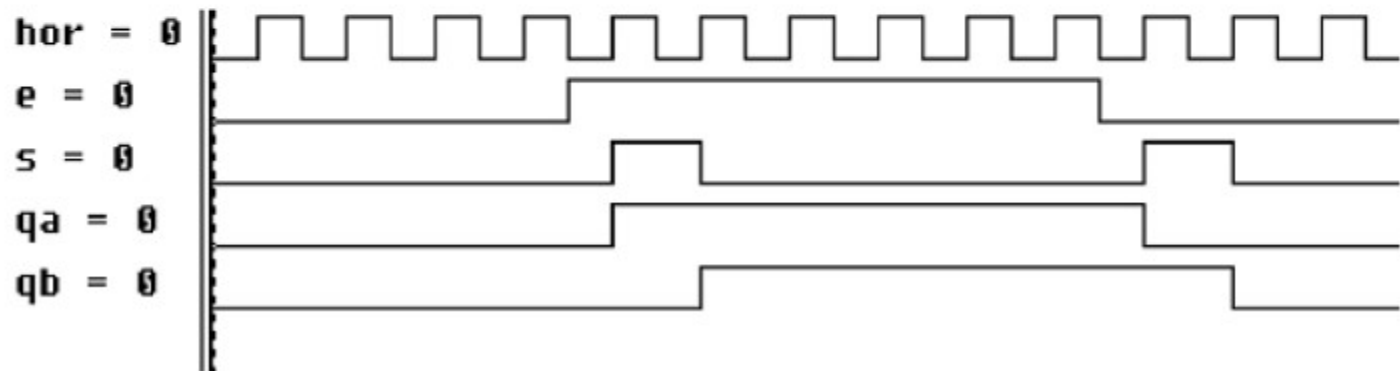
– if (Clock'event and Clock = '0') then

Logique synchrone

Exemple:



Chronogramme :



```

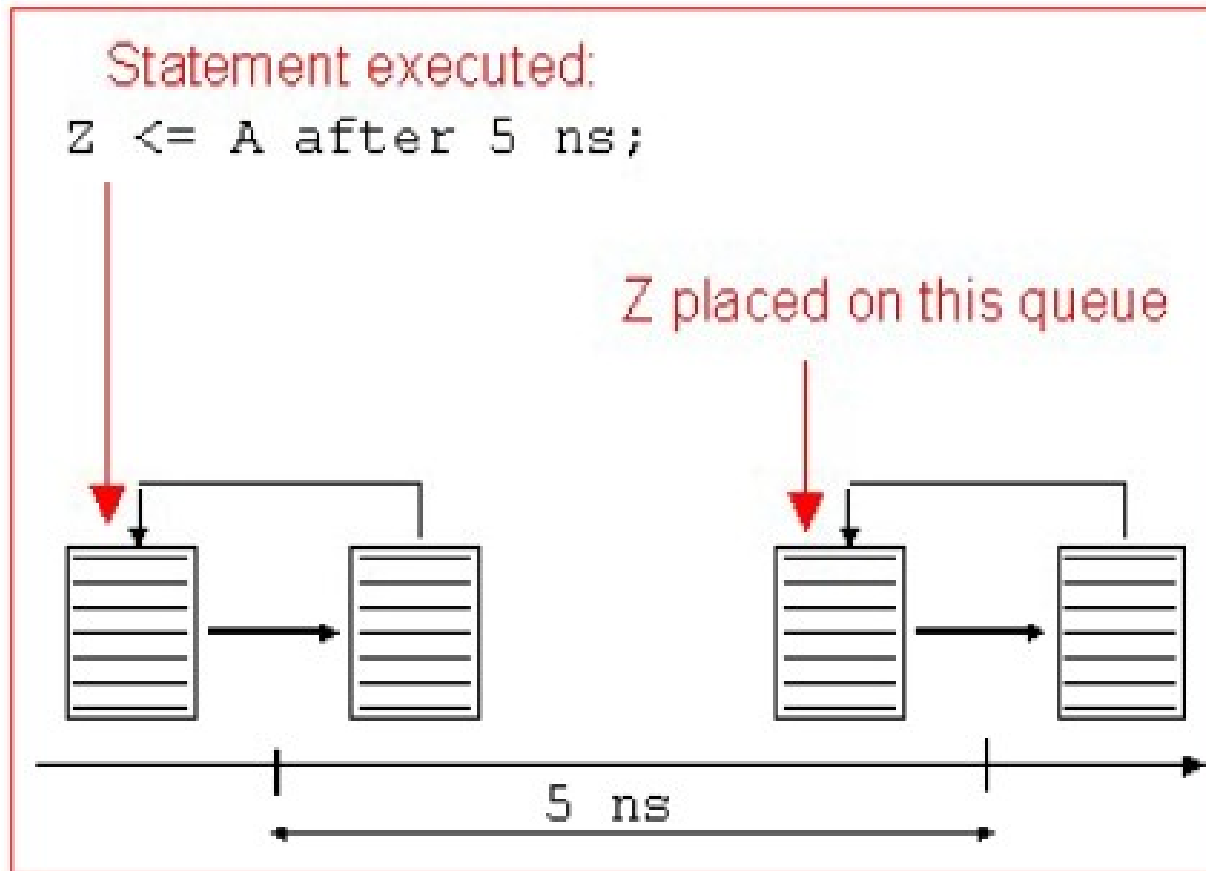
entity transitm is
    port (
        hor, e      : in bit ;
        s           : out bit );
end transitm ;

architecture quasi_struct of transitm is
    signal qa, qb : bit ;
begin
    s <= qa xor qb ;

    schem : process (hor)
    begin
        if(hor'event and hor = '1')then
            qa <= e ;
            qb <= qa ;
        end if;
    end process schem ;
end quasi_struct ;


```

AFTER permet d'affecter la mise à jour du signal dans le futur au point de simulation spécifié par AFTER.



```
process (A,B)
begin
  if (A='1' or B='1') then
    Z <= '1';
  else
    Z <= '0';
  end if;
end process;
```

Suspends
at bottom



Suspends
at "wait"



```
process
begin
  if (A='1' or B='1') then
    Z <= '1';
  else
    Z <= '0';
  end if;
  wait on A, B;
end process;
```

L'instruction **wait on A, B** remplace la liste de sensibilité (paramètres d'entrée du process)

WAIT ON cause la réexécution du processus après un événement sur les signaux spécifiés

- ❑ **WAIT FOR** cause la réexécution du processus après la période passée à cette instruction.
- ❑ Cette forme est utilisée en **TESTBENCHS**.

```
wait for <specific time>;
```

```
STIMULUS: process  
begin  
    SEL <= '0';  
    BUS_B <= "0000";  
    BUS_A <= "1111";  
    wait for 10 ns;  
    SEL <= '1';  
    wait for 10 ns;  
    -- etc, etc  
end process STIMULUS;
```

WAIT UNTIL cause la réexécution du processus après un événement conditionnel sur un signal

```
wait until <condition>;
```

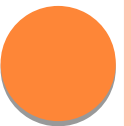
```
process  
begin  
    wait until CLK='1';  
    Q <= D;  
end process;
```


WAIT suspend le processus pour toujours et stoppe la simulation dans les testbenchs

```
wait;
```

```
STIMULUS: process  
begin  
    SEL <= '0';  
    BUS_B <= "0000";  
    BUS_A <= "1111";  
    wait for 10 ns;  
    SEL <= '1';  
    wait;  
end process STIMULUS;
```

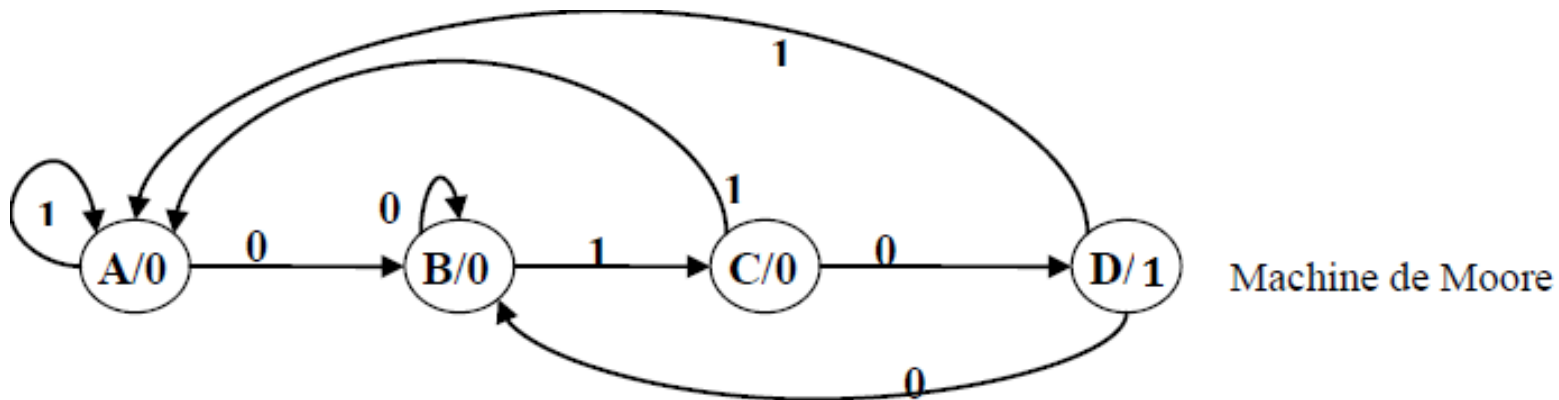
MACHINES À ÉTATS FINIS

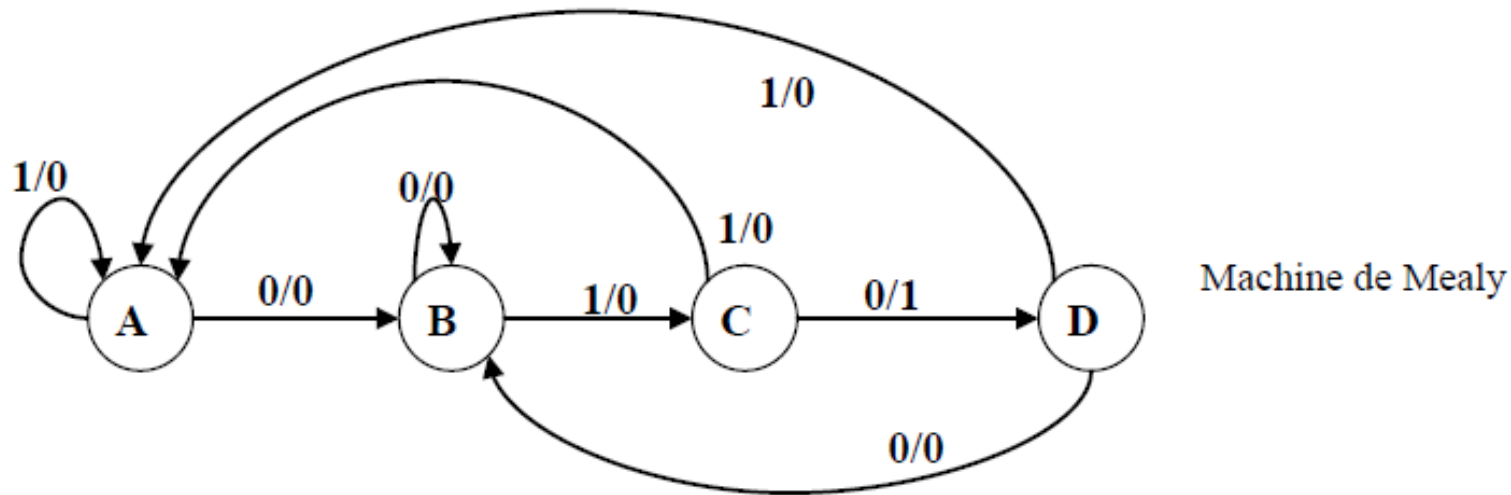


EXEMPLE: LE SÉQUENCEUR

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 apparaît sur l'entrée (E) puis repasser à 0 sur le bit suivant quel que soit sa valeur.

Pour faire la synthèse d'un tel cahier des charges, la première étape est de le modéliser: **graphe d'états**





La structure des deux graphes paraît identique. En fait, il est toujours possible de passer d'un graphe de Moore à un graphe de Mealy. Pour cela il, suffit de reporter les sorties associées à chaque état (Moore) sur les arcs arrivant à chacun de ces états.

Passer d'un graphe de Mealy à un graphe de Moore n'est pas toujours possible.

Table d'états

Etats	Etats suivants		Sortie
	E=0	E=1	
A	B	A	0
B	B	C	0
C	D	A	0
D	B	A	1


Machine de Moore

Etats	Etats suivants		Sortie	
	E=0	E=1	E=0	E=1
A	B	A	0	0
B	B	C	0	0
C	D	A	1	0
D	B	A	0	0

Machine de Mealy

Règle de minimisation:

A la table d'état de la machine de Mealy on remarque que: A et D ont mêmes sorties et mêmes états suivants, ils sont donc équivalents. L'état D peut par exemple être éliminé; on peut remplacer D par A, la table d'état devient :

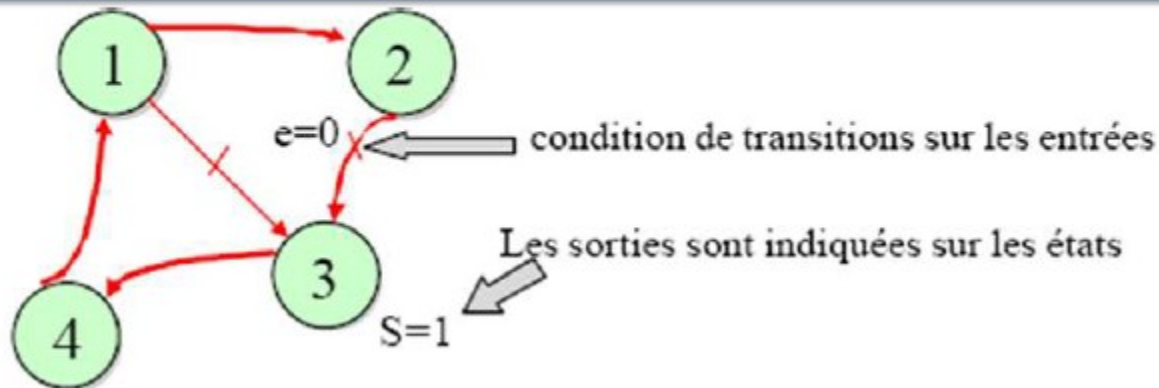


Etats	Etats suivants		Sortie	
	E=0	E=1	E=0	E=1
A	B	A	0	0
B	B	C	0	0
C	A	A	1	0

DÉFINITION D'UNE MACHINE À ÉTATS FINIS

Une machine d'états est un système dynamique (i.e. évolutif) qui peut se trouver, à chaque instant, dans une position parmi un nombre fini de positions possibles. Elle parcourt des cycles, en changeant éventuellement d'état lors des transitions actives de l'horloge, dans l'ordre qui dépend des entrées externes, de façon à fixer sur ses sorties des séquences déterminées par l'application à contrôler.

Spécification d'une machine à états par graphe

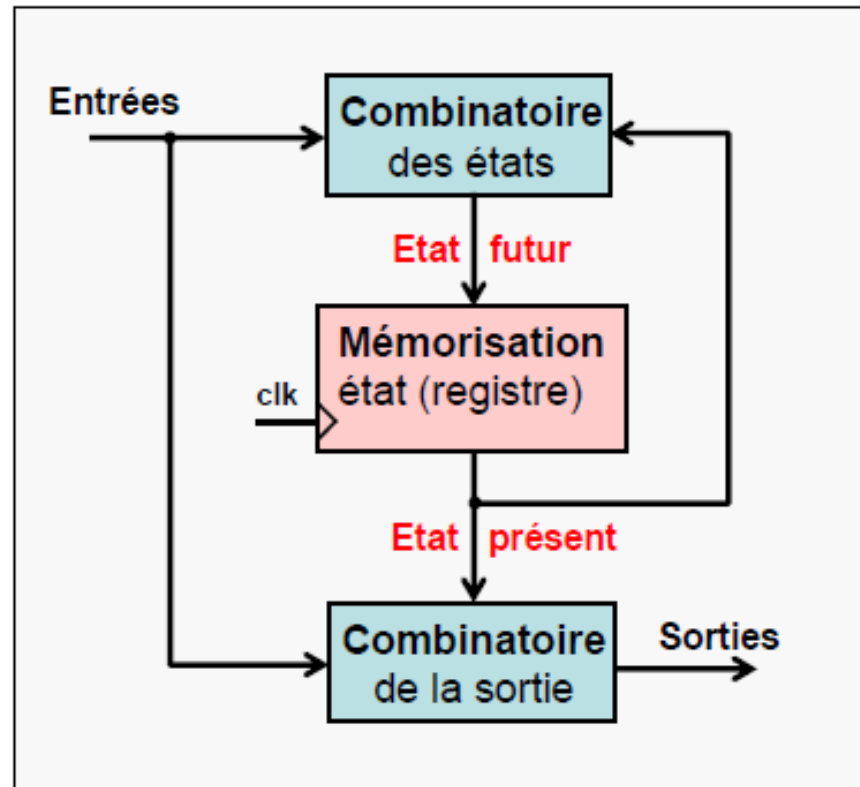


DÉFINITION FORMELLE

- I est l'alphabet d'entrée
- O est l'alphabet de sortie
- S est un ensemble non vide d'états
- Θ est la fonction de transition entre états
 $\Theta : I \times S \rightarrow S$
- Γ est la fonction de sortie
 $\Gamma : S \rightarrow O$ pour les machines de *Moore*
 $\Gamma : I \times S \rightarrow O$ pour les machines de *Mealy*

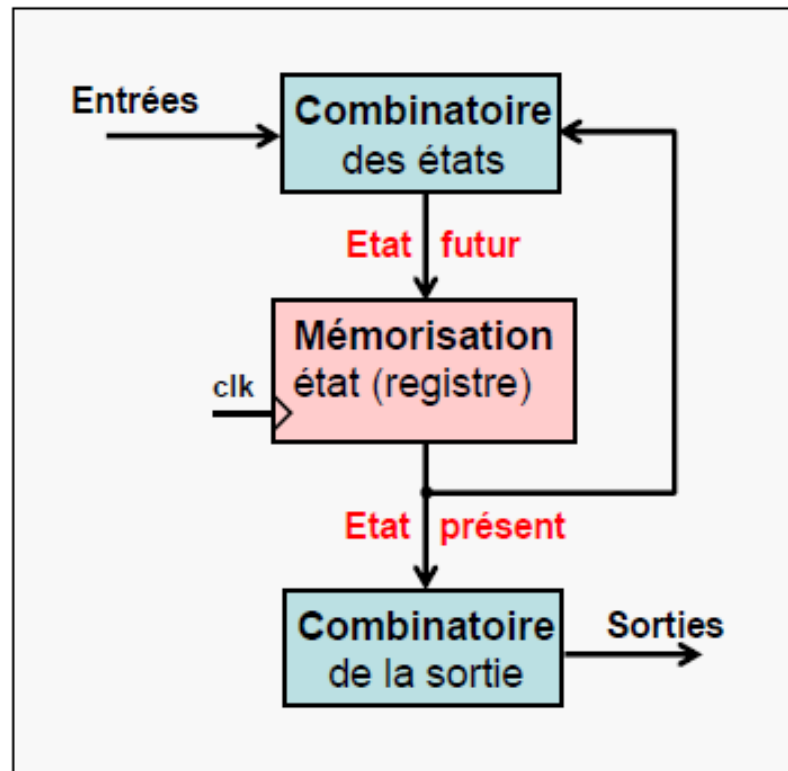
Machine de Mealy.

- L'état futur est calculé à partir des entrées et de l'état présent.
 - Les sorties d'une machine de Mealy dépendent de l'état présent et des entrées.
 - Mémorisation synchrone des états (càd sur un front d'horloge).
 - La sortie dépend directement de l'entrée et ceci indépendamment de l'horloge (clk).
- ⇒ Sortie asynchrone.
- Nombre d'états plus réduit que pour une machine de Moore.
 - Il est possible de resynchroniser la sortie au besoin en ajoutant des bascules D.



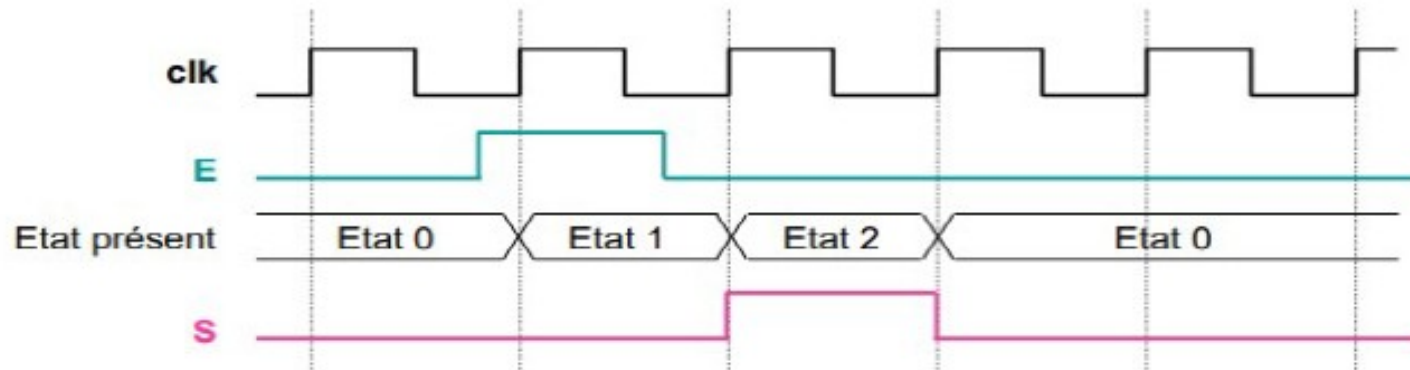
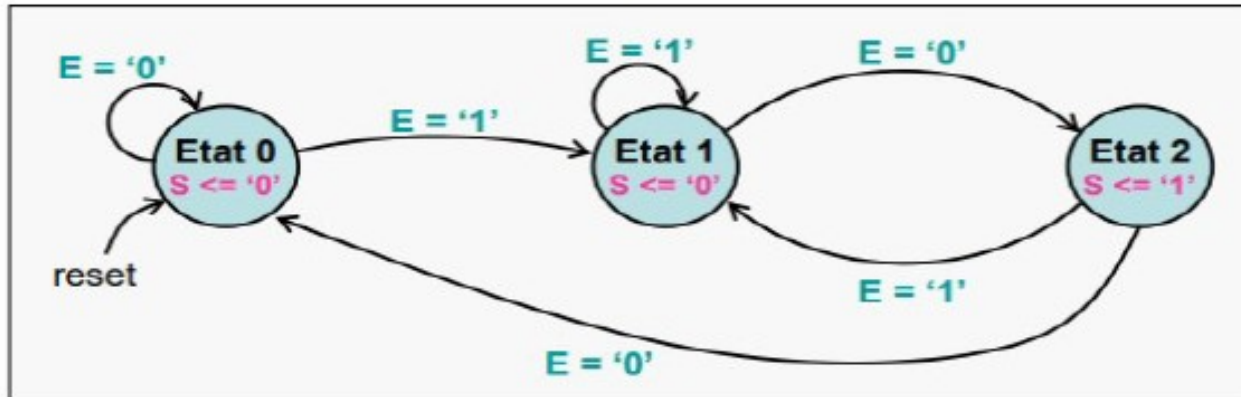
Machine de Moore.

- Les sorties d'une machine de Moore dépendent de l'état présent (synchrones, elles changent sur un front d'horloge).
- L'état futur est calculé à partir des entrées et de l'état présent.



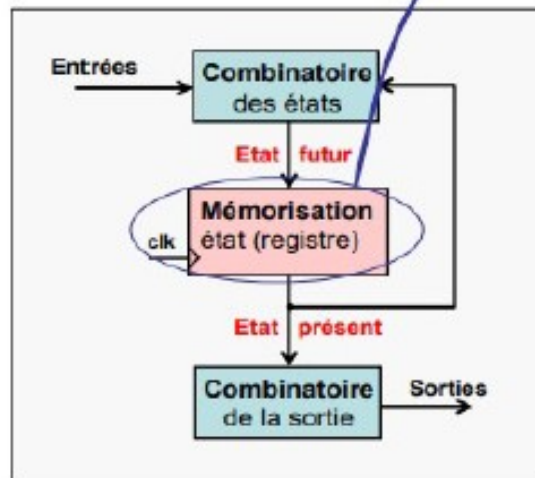
FSM EN VHDL

Exemple : Machine de Moore reconnaissant la séquence 10



Ecriture VHDL machine de Moore

Description avec 3 process



- Un process séquentiel de mise à jour de l'état présent par l'état futur sur les fronts montant d'horloge (reset asynchrone inclus) :

```
type Etat is (Etat0, Etat1, Etat2);  
Signal Etat_present, Etat_futur : Etat := Etat0;
```

```
Sequentiel_maj_etat : process (clk, reset)  
begin
```

```
    if reset = '0' then
```

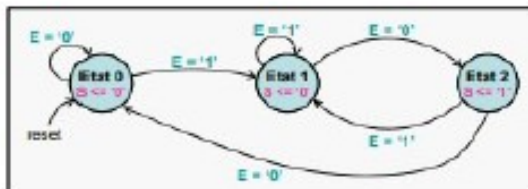
```
        Etat_present <= Etat0;
```

```
    elsif clk'event and clk = '1' then
```

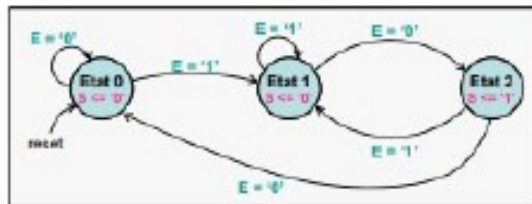
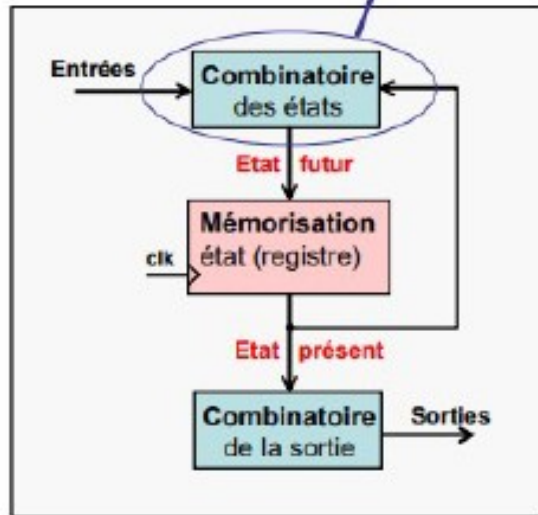
```
        Etat_present <= Etat_futur;
```

```
    end if;
```

```
end process Sequentiel_maj_etat;
```



Description avec 3 process



- Un process combinatoire de calcul de l'état futur à partir des entrées et de l'état présent :

```
Combinatoire_etats : process (E, Etat_present)
begin
  case Etat_present is

    when Etat0 => if E = '1' then
                    Etat_futur <= Etat1;
                  else
                    Etat_futur <= Etat0;
                  end if;

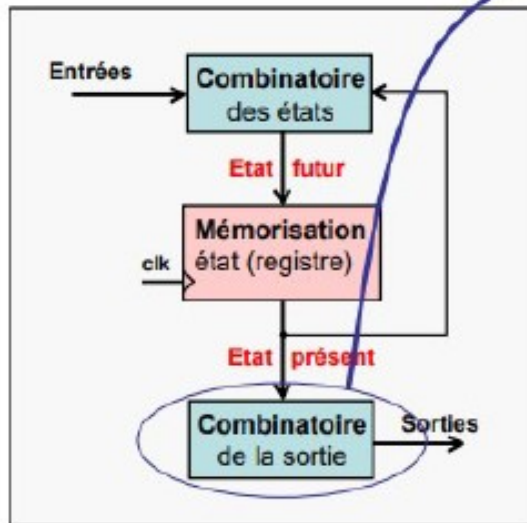
    when Etat1 => if E = '0' then
                    Etat_futur <= Etat2;
                  else
                    Etat_futur <= Etat1;
                  end if;

    when Etat2 => if E = '1' then
                    Etat_futur <= Etat1;
                  else
                    Etat_futur <= Etat0;
                  end if;

  end case;
end process Combinatoire_etats;
```

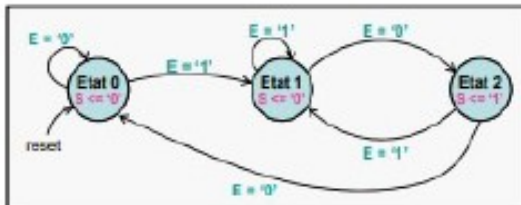
Activer V

Description avec 3 process

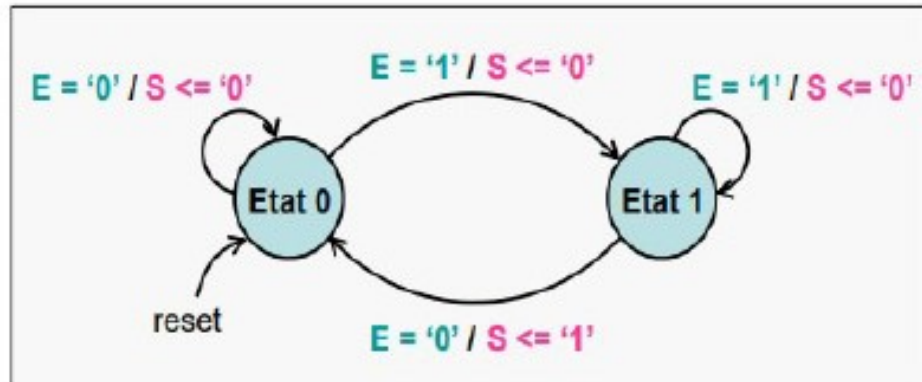


- Un process combinatoire de calcul des sorties à partir de l'état présent :

```
Combinatoire_sorties : process (Etat_present)
begin
    case Etat_present is
        when Etat0 => S <= '0';
        when Etat1 => S <= '0';
        when Etat2 => S <= '1';
    end case;
end process Combinatoire_sorties;
```

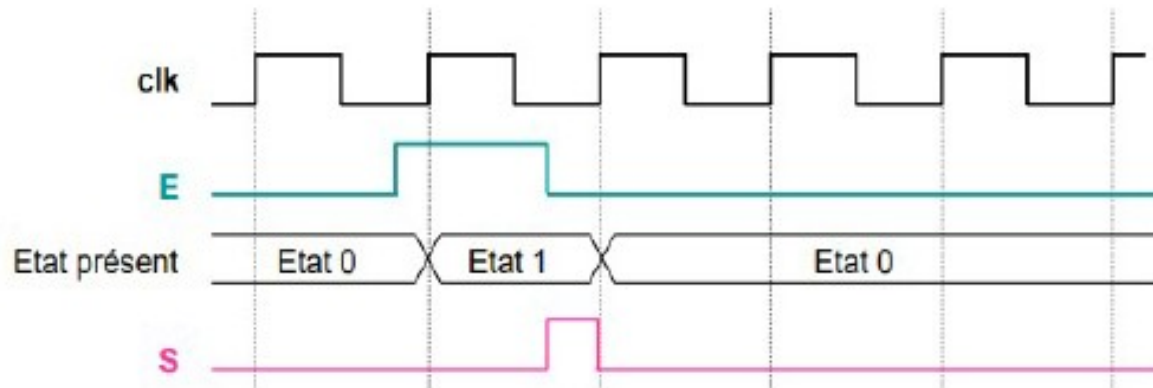


Exemple : Machine de Mealy reconnaissant la séquence 10



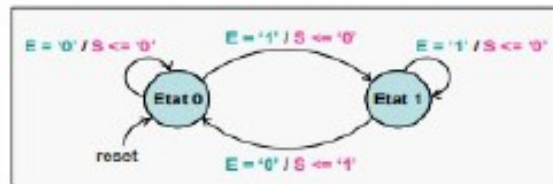
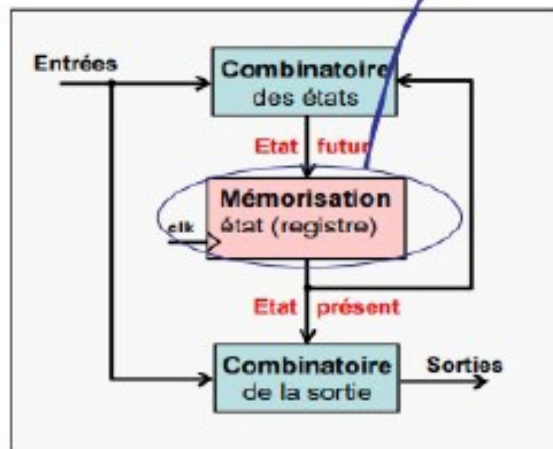
$E = '1' / S \leq '0'$

Condition de validation de la transition Affectation de la valeur '0' à la sortie



Ecriture VHDL machine de Mealy

Description avec 3 process



- Un process séquentiel de mise à jour de l'état présent par l'état futur sur les fronts montant d'horloge (reset asynchrone inclus) :

```
type Etat is (Etat0, Etat1);  
Signal Etat_present, Etat_futur : Etat := Etat0;
```

```
Sequential_maj_etat : process (clk, reset)  
begin
```

```
    if reset = '0' then
```

```
        Etat_present <= Etat0;
```

```
    elsif clk'event and clk = '1' then
```

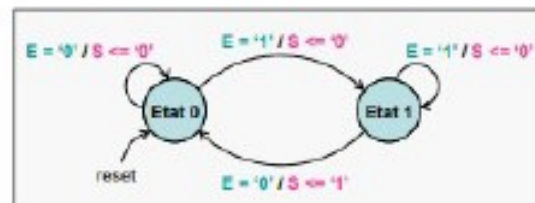
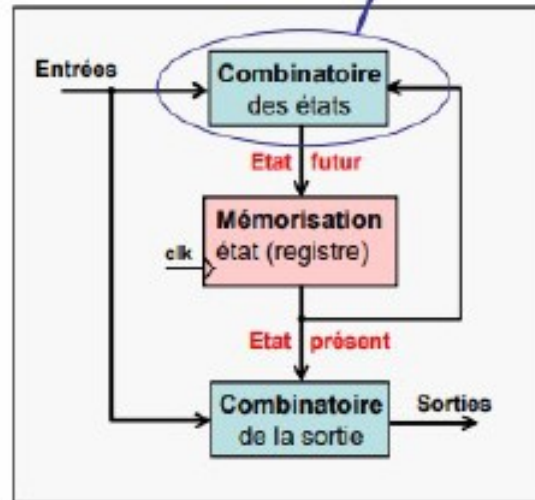
```
        Etat_present <= Etat_futur;
```

```
    end if;
```

```
end process Sequential_maj_etat;
```

Description avec 3 process

- Un process combinatoire de calcul de l'état futur à partir des entrées et de l'état présent :



```
Combinatoire_etats : process (E, Etat_present)
begin
```

```
  case Etat_present is
```

```
    when Etat0 => if E = '1' then
                     Etat_futur <= Etat1;
                   else
                     Etat_futur <= Etat0;
                   end if;
```

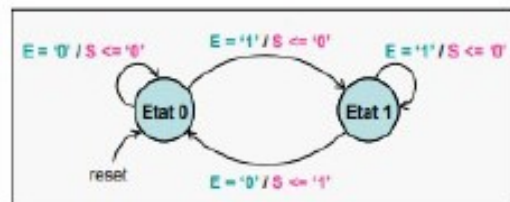
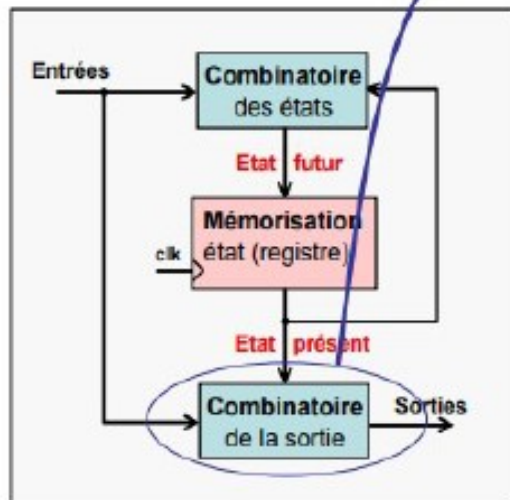
```
    when Etat1 => if E = '1' then
                     Etat_futur <= Etat1;
                   else
                     Etat_futur <= Etat0;
                   end if;
```

```
  end case;
```

```
end process Combinatoire_etats;
```


Description avec 3 process

- Un process combinatoire de calcul des sorties à partir des entrées et de l'état présent :



```
Combinatoire_sorties : process (E, Etat_present)
begin
```

```
  case Etat_present is
```

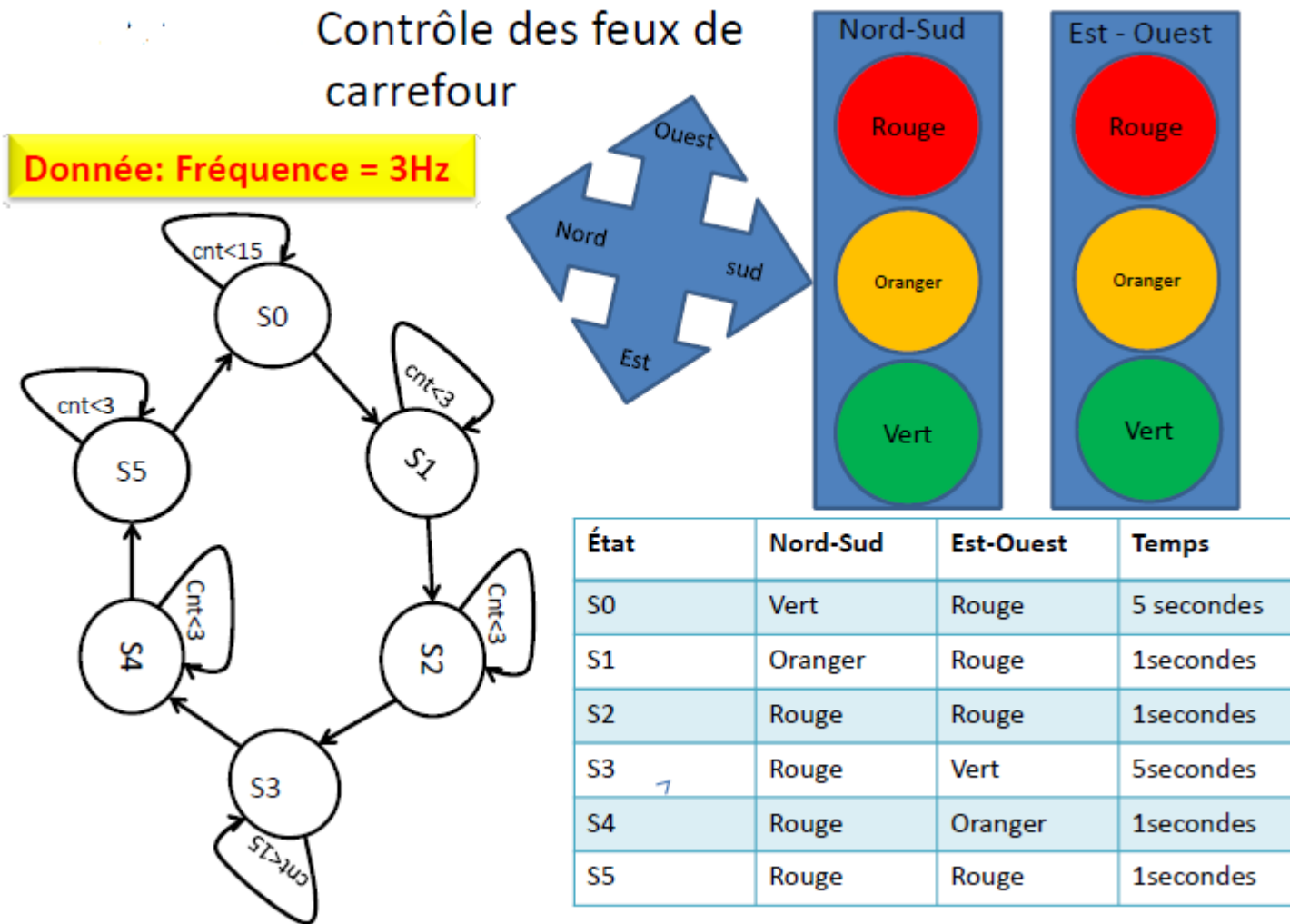
```
    when Etat0 => if E = '1' then
                      S <= '0';
                    else
                      S <= '0';
                    end if;
```

```
    when Etat1 => if E = '0' then
                      S <= '1';
                    else
                      S <= '0';
                    end if;
```

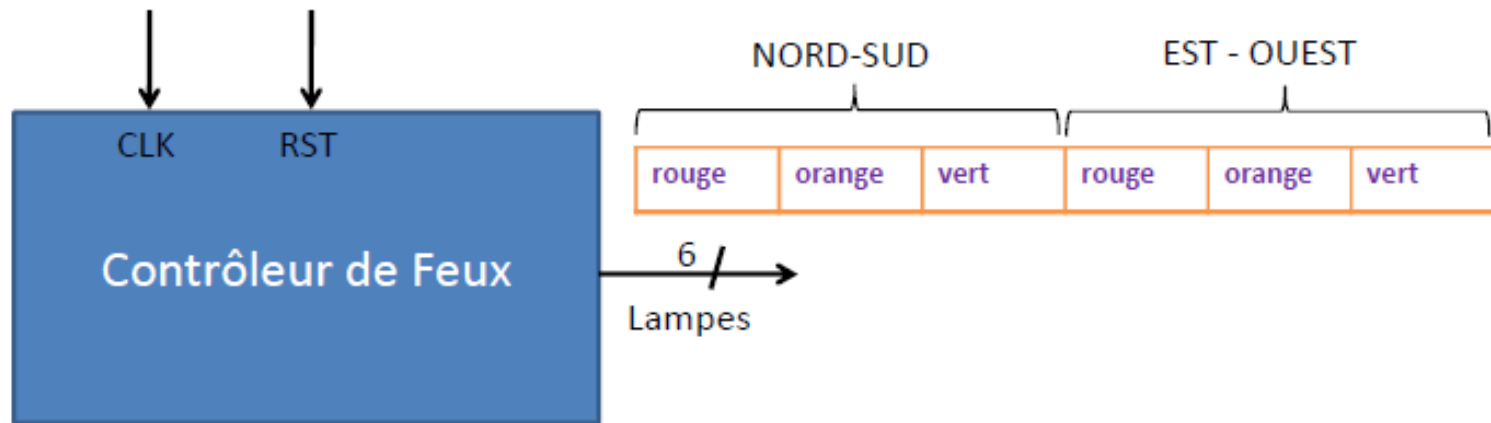
```
  end case;
```

```
end process Combinatoire_sorties;
```

Exemple avec comptage



Soit l'interface Suivante



- Définir un signal interne qui s'appelle CNT. Il permet de compter le nombre de cycles écoulés.

Solution

```
entity cont_feux is
    port(clk, RST: in std_logic;
          lampes:out std_logic_vector(5 downto 0));
end cont_feux ;

architecture FSM of cont_feux is
    Signal count: std_logic_vector(3 downto 0);
    constant sec5:std_logic_vector(3 downto 0) := "1111";
    constant sec1:std_logic_vector(3 downto 0) := "0011";
    type Tstate is (S0, S1, S2,S3,S4,S5);
    signal state: Tstate ;
begin
```

Suite ...

```
carrefour: process(clk, RST)
begin
  if(RST = '1') then
    state <= S0;
    count <= "0000";
  elsif(clk'event and clk ='1') then
    case state is
      when S0 =>
        Lampes <= "001100";
        if count < sec5 then
          state <= S0;
          count <= count +1;
        else
          state <= S1;
          count <= "0000";
        end if;
```

```
      when S1 =>
        Lampes <= "010100";
        if count < sec1 then
          state <= S1;
          count <= count +1;
        else
          state <= S2;
          count <= "0000";
        end if;
      when S2 =>
        Lampes <= "100100";
        if count < sec1 then
          state <= S2;
          count <= count +1;
        else
          state <= S3;
          count <= "0000";
        end if;
```

Suite ...

```
when S3 =>
    Lampes <= "100001";
    if count < sec5 then
        state <= S3;
        count <= count +1;
    else
        state <= S4;
        count <= "0000";
    end if;
when S4 =>
    Lampes <= "100010";
    if count < sec3 then
        state <= S4;
        count <= count +1;
    else
        state <= S5;
        count <= "0000";
    end if;
when S5 =>
    Lampes <= "100001";
    if count < sec3 then
        state <= S5;
        count <= count +1;
    else
        state <= S0;
        count <= "0000";
    end if;
when others =>
    Lampes <= "000000";
    count <= "0000";
end case;
end process carrefour;
end FSM;
```