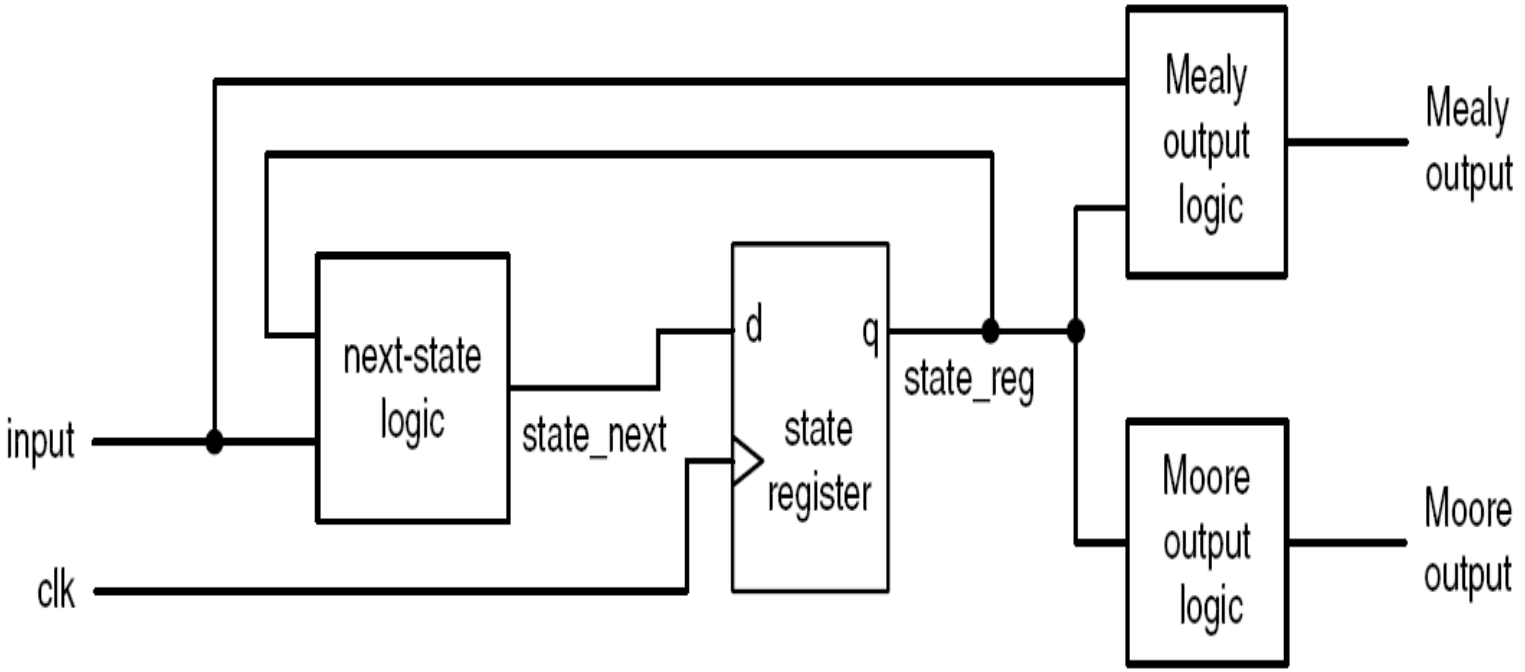
Sommaire

5 Machines à état fini (Finite State Machines)
- Machines à état fini (Finite State Machines)

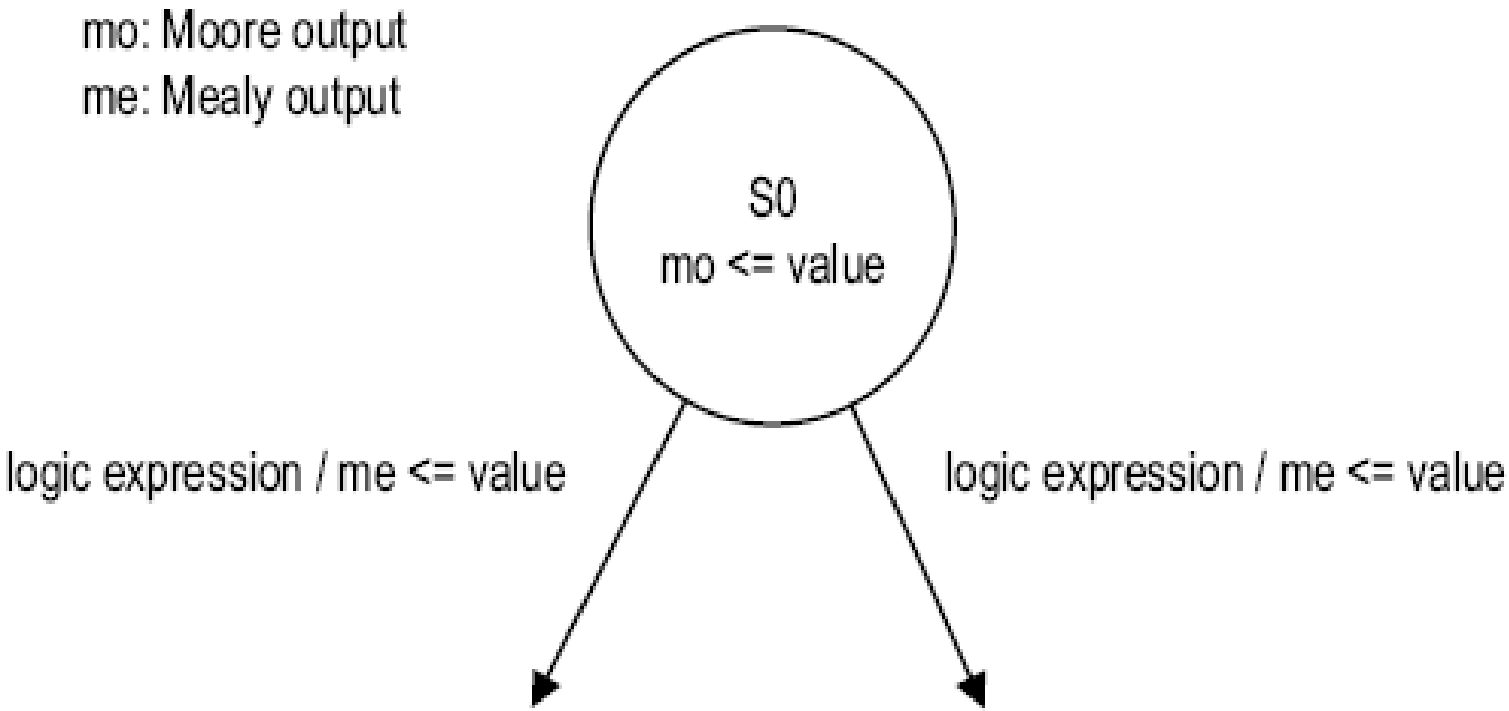
Machines d'états

- Pour la réalisation de contrôleur dans un circuit complexe
- Deux types de machines : Mealy et Moore



Machines d'états

Diagramme d'états



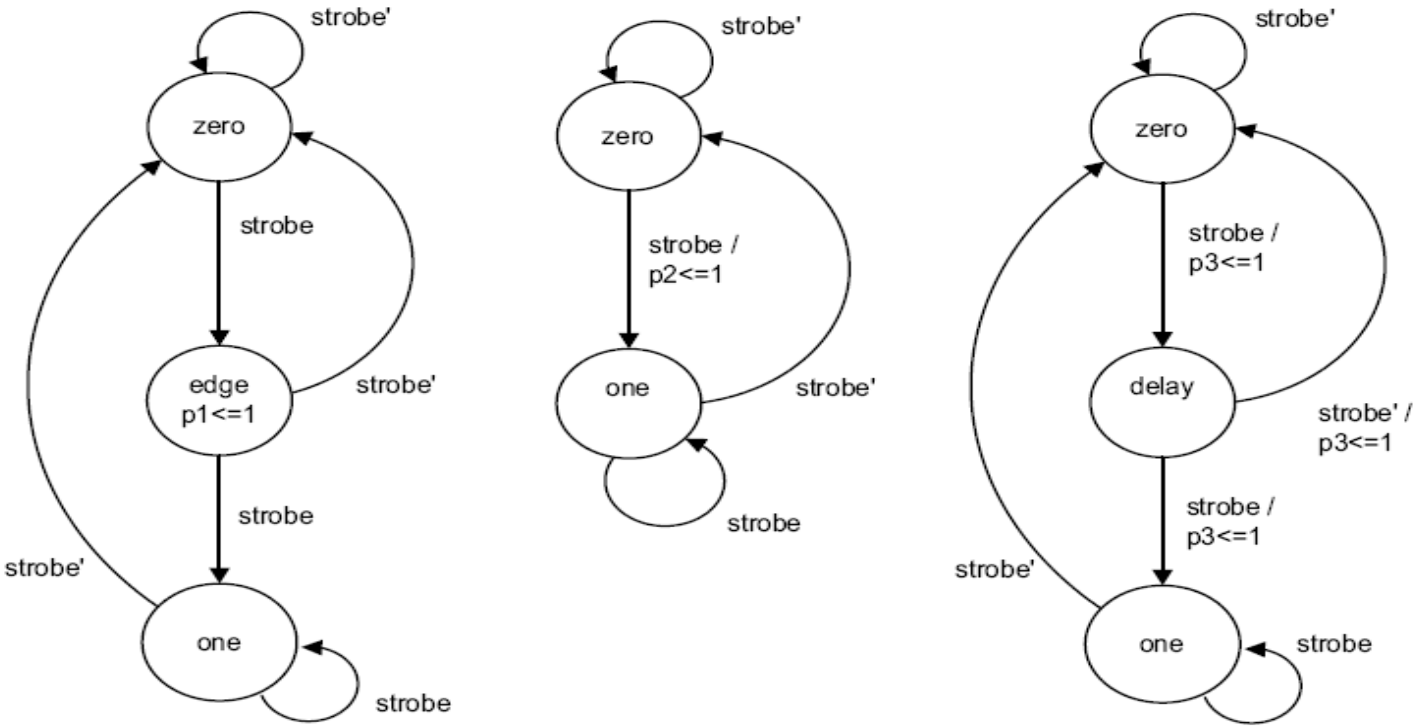
Moore vs Mealy

- Machine de Moore
  - la sortie est uniquement la fonction de l'état de la machine
- Machine de Mealy
  - la sortie est une fonction de l'état de la machine et des entrées
- Comment choisir le type de machine d'états ?

Moore vs Mealy

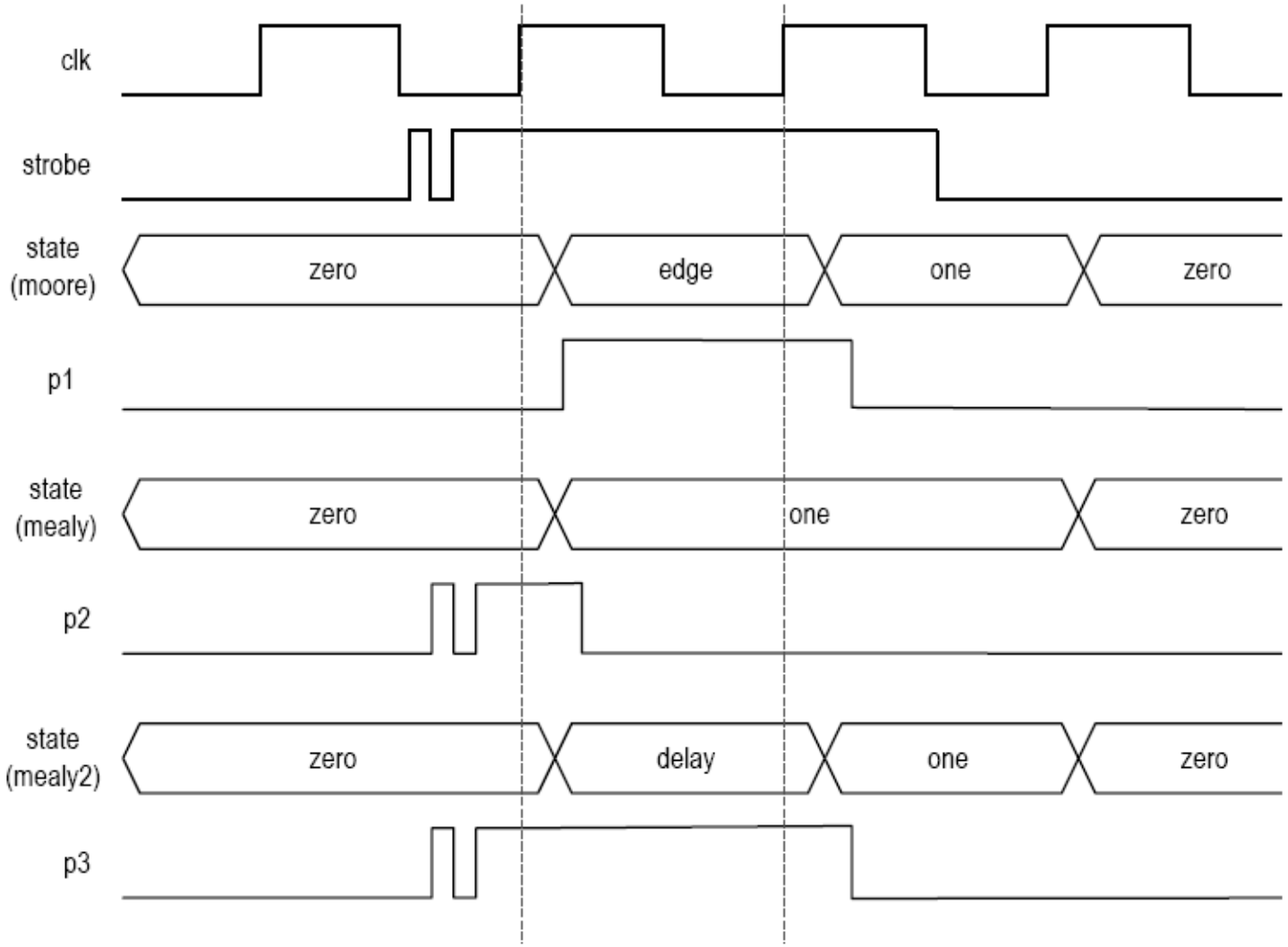
Exemple

- Détecteur de front montant d'un signal d'entrée



Moore vs Mealy

Exemple



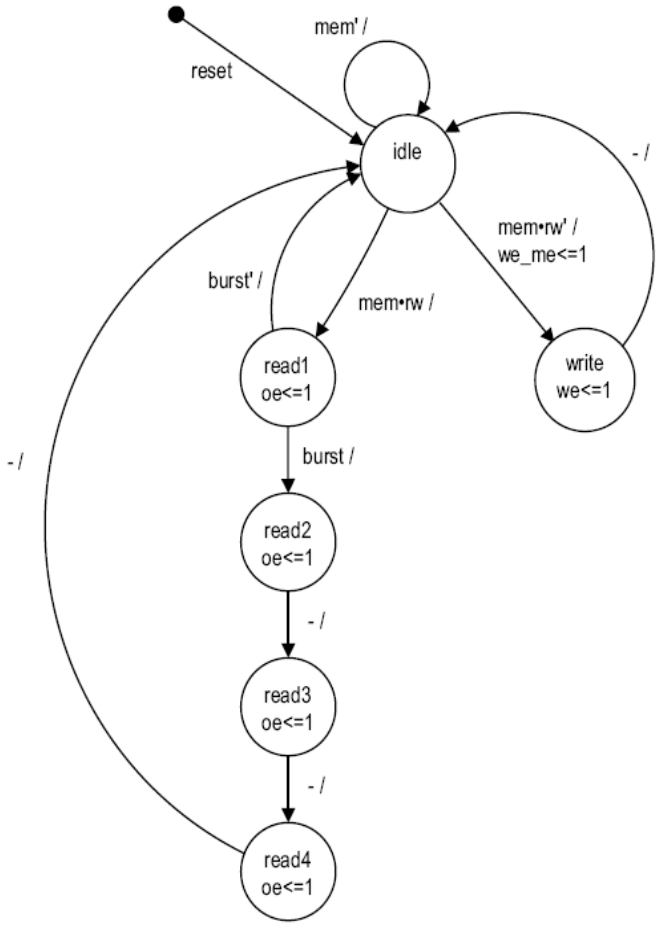
Moore vs Mealy

Exemple

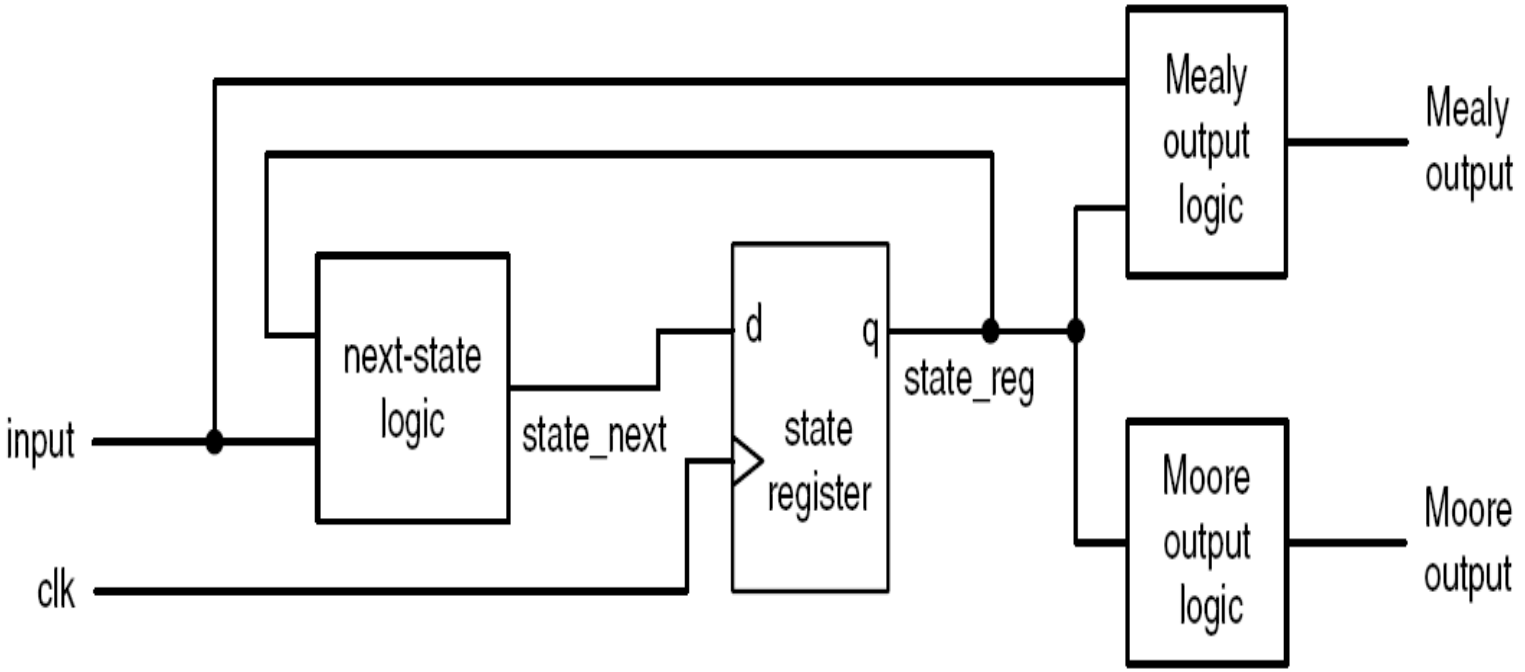
- Machine de Mealy :
  - utilise en général moins d'états
  - plus rapide
  - transparente aux signaux transitoires (*glitches*)
- Laquelle des deux est meilleure ?
- La réponse dépend du type de signal de contrôle à réaliser
- Un signal sensible aux fronts montant ou descendant (*edge sensitive*)
  - Exemple : le signal d'activation d'un compteur
  - Les deux peuvent être utilisées mais la machine de Mealy est plus rapide
- Un signal sensible au niveau (*level sensitive*)
  - Exemple : un signal d'écriture d'une SRAM
  - Pour ce cas de figure, à préférer une machine de Moore

Machine d'états en VHDL

- Utilisation de types énumérés pour coder les états d'une FSM et pour rendre le code plus lisible



Machine d'états en VHDL



- Entité

Machine d'états en VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
   port(
      clk, reset: in std_logic;
      mem, rw, burst: in std_logic;
      oe, we, we_me: out std_logic
   );
end mem_ctrl ;
```

Machine d'états en VHDL

- Type énuméré pour décrire les états

```
architecture mult_seg_arch of mem_ctrl is
   type mc_state_type is
      (idle, read1, read2, read3, read4, write);
   signal state_reg, state_next: mc_state_type;
begin
```

- Registre d'états

```
   -- state register
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= idle;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
   -- next-state logic
```

Machine d'états en VHDL

```
-- next-state logic
process(state_reg,mem,rw,burst)
begin
   case state_reg is
      when idle =>
         if mem='1' then
            if rw='1' then
               state_next <= read1;
            else
               state_next <= write;
            end if;
         else
            state_next <= idle;
         end if;
      when write =>
         state_next <= idle;
      when read1 =>
         if (burst='1') then
            state_next <= read2;
         else
```

Machine d'états en VHDL

```
            state_next <= idle;
         end if;
      when read2 =>
         state_next <= read3;
      when read3 =>
         state_next <= read4;
      when read4 =>
         state_next <= idle;
   end case;
end process;
```

Machine d'états en VHDL

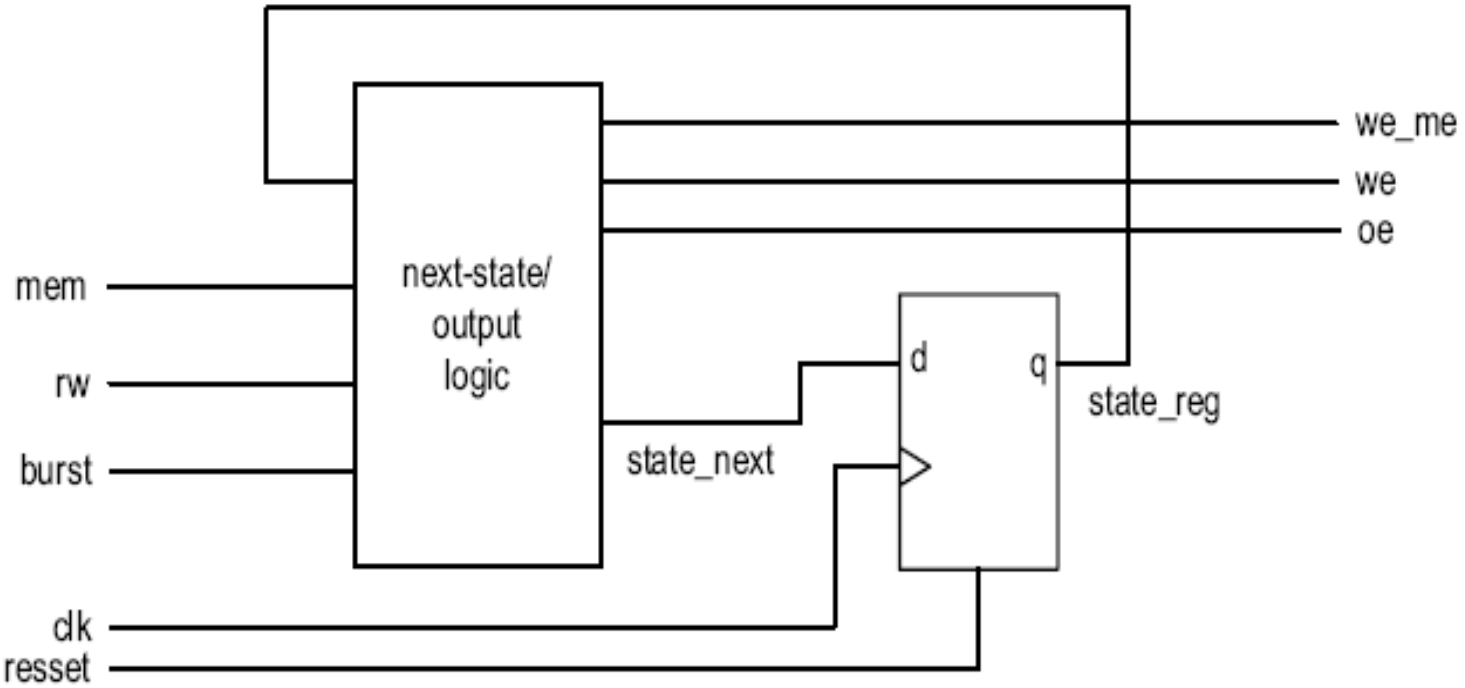
```
-- Moore output logic
process(state_reg)
begin
   we <= '0'; -- default value
   oe <= '0';  -- default value
   case state_reg is
      when idle =>
      when write =>
         we <= '1';
      when read1 =>
         oe <= '1';
      when read2 =>
         oe <= '1';
      when read3 =>
         oe <= '1';
      when read4 =>
         oe <= '1';
   end case;
end process;
```

Machine d'états en VHDL

```
   -- Mealy output logic
   process(state_reg,mem,rw)
   begin
      we_me <= '0'; -- default value
      case state_reg is
         when idle =>
            if (mem='1') and (rw='0') then
              we_me <= '1';
            end if;
         when write =>
         when read1 =>
         when read2 =>
         when read3 =>
         when read4 =>
      end case;
   end process;
   we_me <= '1' when ((state_reg=idle) and (mem='1') and (rw='0')) else
     '0';
end mult_seg_arch;
```

Machine d'états en VHDL

- Combiner la logique de sortie avec la logique de l'état futur



Machine d'états en VHDL

```
-- next-state logic and output logic
process(state_reg,mem,rw,burst)
begin
   oe <= '0';     -- default values
   we <= '0';
   we_me <= '0';
   case state_reg is
      when idle =>
         if mem='1' then
            if rw='1' then
               state_next <= read1;
            else
               state_next <= write;
               we_me <= '1';
            end if;
         else
            state_next <= idle;
         end if;
      when write =>
         state_next <= idle;
```

Machine d'états en VHDL

```
            we <= '1';
         when read1 =>
            if (burst='1') then
               state_next <= read2;
            else
               state_next <= idle;
            end if;
            oe <= '1';
         when read2 =>
            state_next <= read3;
            oe <= '1';
         when read3 =>
            state_next <= read4;
            oe <= '1';
         when read4 =>
            state_next <= idle;
            oe <= '1';
      end case;
   end process;
```

Machine d'états en VHDL

- Une machine peut être décrite en un seul segment

```
architecture one_seg_wrong_arch of mem_ctrl is
   type mc_state_type is
        (idle, read1, read2, read3, read4, write);
   signal state_reg: mc_state_type;
begin
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= idle;
      elsif (clk'event and clk='1') then
         oe <= '0';     -- default values
         we <= '0';
         we_me <= '0';
         case state_reg is
            when idle =>
               if mem='1' then
                  if rw='1' then
                     state_reg <= read1;
```

Machine d'états en VHDL

```
            else
               state_reg <= write;
               we_me <= '1';
            end if;
         else
            state_reg <= idle;
         end if;
      when write =>
         state_reg <= idle;
         we <= '1';
      when read1 =>
         if (burst='1') then
            state_reg <= read2;
         else
            state_reg <= idle;
         end if;
         oe <= '1';
      when read2 =>
         state_reg <= read3;
         oe <= '1';
```

Machine d'états en VHDL

```
            when read3 =>
               state_reg <= read4;
               oe <= '1';
            when read4 =>
               state_reg <= idle;
               oe <= '1';
         end case;
      end if;
   end process;
end one_seg_wrong_arch;
```

Machine d'états en VHDL

- L'encodage des états d'une machine d'états peut être effectué à plusieurs façons :
  - binaire, binaire réfléchi, *one-hot* ou presque *one-hot*
- Le type d'encodage choisi ne joue pas sur la fonctionnalité de la FSM mais surtout sur sa complexité (partie l'état futur et la logique de sortie)

Table 10.1 State assignment example

	Binary assignment	Gray code assignment	One-hot assignment	Almost one-hot assignment
idle	000	000	000001	00000
read1	001	001	000010	00001
read2	010	011	000100	00010
read3	011	010	001000	00100
read4	100	110	010000	01000
write	101	111	100000	10000

Machine d'états en VHDL

- Utilisation de l'attribut `enum_encoding`

```
   type mc_state_type is (idle,write,read1,
read2,read3,read4);
   attribute enum_encoding: string;
   attribute enum_encoding of mc_state_type:
   type is "0000 0100 1000 1001 1010 1011";
```

- Utilisation explicite de constantes

Machine d'états en VHDL

```
architecture state_assign_arch of mem_ctrl is
   constant idle:  std_logic_vector(3 downto 0):="0000";
   constant write: std_logic_vector(3 downto 0):="0100";
   constant read1: std_logic_vector(3 downto 0):="1000";
   constant read2: std_logic_vector(3 downto 0):="1001";
   constant read3: std_logic_vector(3 downto 0):="1010";
   constant read4: std_logic_vector(3 downto 0):="1011";
   signal state_reg,state_next: std_logic_vector(3 downto 0);
begin
   -- state register
   process(clk,reset)
   begin
      if (reset='1') then
         state_reg <= idle;
      elsif (clk'event and clk='1') then
         state_reg <= state_next;
      end if;
   end process;
   -- next-state logic
   process(state_reg,mem,rw,burst)
```

Machine d'états en VHDL

```
begin
   case state_reg is
      when idle =>
         if mem='1' then
            if rw='1' then
               state_next <= read1;
            else
               state_next <= write;
            end if;
         else
            state_next <= idle;
         end if;
      when write =>
         state_next <= idle;
      when read1 =>
         if (burst='1') then
            state_next <= read2;
         else
            state_next <= idle;
         end if;
```

Machine d'états en VHDL

```
        when read2 =>
           state_next <= read3;
        when read3 =>
           state_next <= read4;
        when read4 =>
           state_next <= idle;
        when others =>
           state_next <= idle;
    end case;
end process;
```

- Parfois, un signal de sortie sans *glitch* (aléas) est nécessaire
- Pour l'obtenir, il faut rajouter une bascule en sortie introduisant un cycle de retard supplémentaire
- Une solution nommée *look-ahead* permet de réduire ce cycle d'horloge tout en veillant à ce que le signal de sortie soit sans *glitches*

Machine d'états en VHDL

```
-- output buffer
process(clk,reset)
begin
   if (reset='1') then
      oe_buf_reg <= '0';
      we_buf_reg <= '0';
   elsif (clk'event and clk='1') then
      oe_buf_reg <= oe_next;
      we_buf_reg <= we_next;
   end if;
end process;
```

Machine d'états en VHDL

```
-- look-ahead output logic
process(state_next)
begin
   we_next <= '0'; -- default value
   oe_next <= '0'; -- default value
   case state_next is
      when idle =>
      when write =>
         we_next <= '1';
      when read1 =>
         oe_next <= '1';
      when read2 =>
         oe_next <= '1';
      when read3 =>
         oe_next <= '1';
      when read4 =>
         oe_next <= '1';
   end case;
end process;
```

Machine d'états

Exemples

- Coder en VHDL un détecteur de front montant en utilisant :
  - une machine de Moore
  - une machine de Mealy