

# VHDL 3: Composants de base

## combinatoires et séquentiels

Andres Upegui







Laboratoire de Systèmes Numériques  
hepia

- 1 Composants de base combinatoires
- 2 Bascules
- 3 Composants Séquentiels
- 4 Machines d'états
- 5 Machine de Mealy: Exemple
- 6 Machine de Moore: Exemple






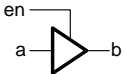
# Composants de base

- Composants de base des designs à réaliser au laboratoire:
- Combinatoires
  - Portes logiques: and, or, xor, nand, ...
  - Porte tri-state
  - Multiplexeurs
  - fonctions de type additionneur, soustracteur, comparateur
- Séquentiels
  - Bascules D (avec ou sans load)
  - Registres

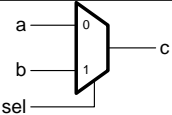
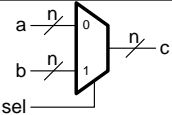
# Composants combinatoires

Not		$b \leq \text{not } a$
And		$c \leq a \text{ and } b$
Or		$c \leq a \text{ or } b$
Xor		$c \leq a \text{ xor } b$
Nand		$c \leq a \text{ nand } b$
...	...	...
Tri-state		

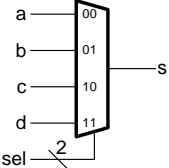
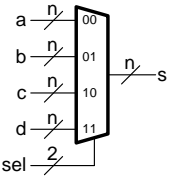
# Composants combinatoires

Not		$b \leq \text{not } a$
And		$c \leq a \text{ and } b$
Or		$c \leq a \text{ or } b$
Xor		$c \leq a \text{ xor } b$
Nand		$c \leq a \text{ nand } b$
...	...	...
Tri-state		$c \leq a \text{ when } en = '1' \text{ else } 'Z'$

# Composants combinatoires: multiplexeurs (1)




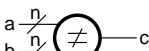
mux2		<pre>c &lt;= a when sel='0' else b</pre>
mux2_n		<pre>c &lt;= a when sel='0' else b</pre>

# Composants combinatoires: multiplexeurs (2)

mux4		<pre>c &lt;= a when sel="00" else b  when sel="01" else c  when sel="10" else d;</pre>
mux4_n		<pre>c &lt;= a when sel="00" else b  when sel="01" else c  when sel="10" else d;</pre>

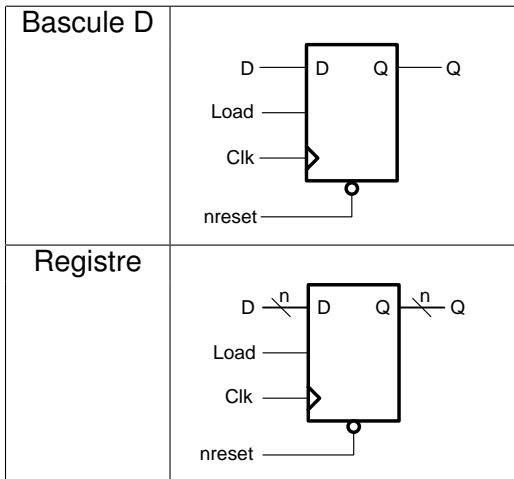
- ... mux8, mux16, ...

# Composants combinatoires: opérations

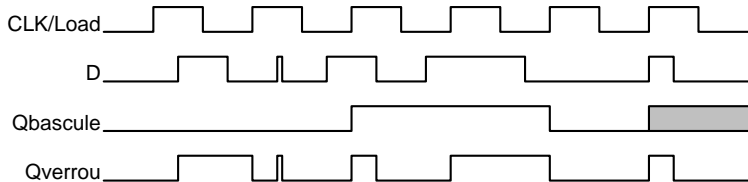
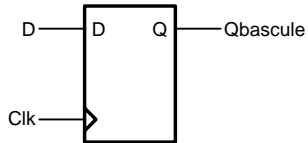
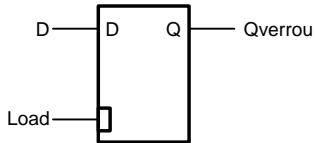
+		$c \leq a + b$
-		$c \leq a - b$
=		$c \leq '1' \text{ when } a = b \text{ else } '0'$
$\neq$		$c \leq '0' \text{ when } a = b \text{ else } '1'$
...	...	...



# Composants séquentiels



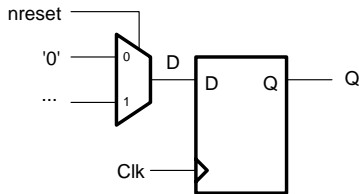
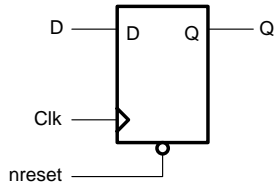
# Bascules et verrous



# Bascule D: Reset

## Entité

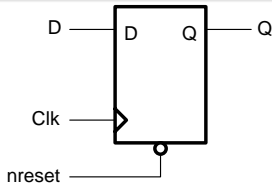
```
entity basculeD is
  port( clk,nreset,d: in std_logic;
        q: out std_logic);
end basculeD;
```



# Bascule D: Reset

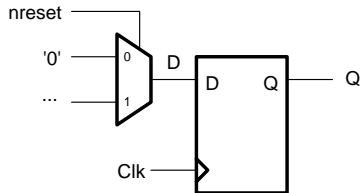
## Reset asynchrone

```
architecture asynch of basculeD is
begin
  process(nreset,clk)
  begin
    if nreset='0' then
      q<='0';
    elsif rising_edge(clk) then
      q<=d;
    end if;
  end process;
end asynch;
```

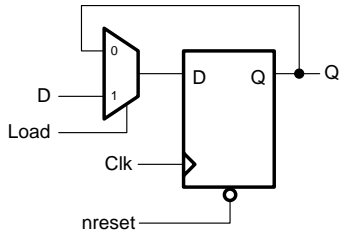


## Reset synchrone

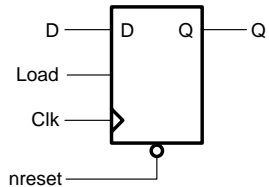
```
architecture synch of basculeD is
begin
  process(clk)
  begin
    if rising_edge(clk) then
      if nreset='0' then
        q<='0';
      else
        q<=d;
      end if;
    end if;
  end process;
end synch;
```



# Bascules D: Load



≡



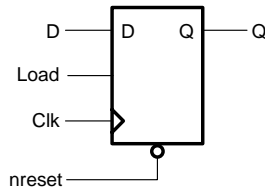
# Bascules D: Load

## Bascule D avec Load

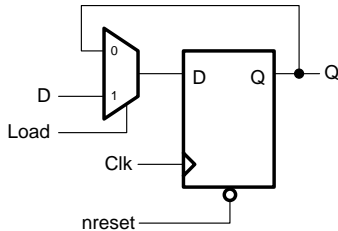
```

architecture comp of basculeD is
begin
    process(nreset,clk)
    begin
        if nreset='0' then
            q<='0';
        elsif rising_edge(clk) then
            if load='1' then
                q<=d;
            end if;
        end if;
    end process;
end comp;
  
```

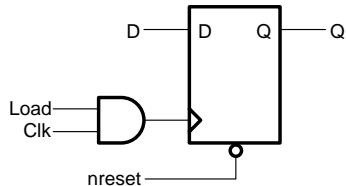
≡



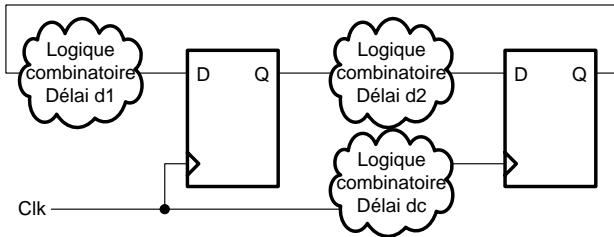
# Bascules D: Load



Pourquoi pas ça?



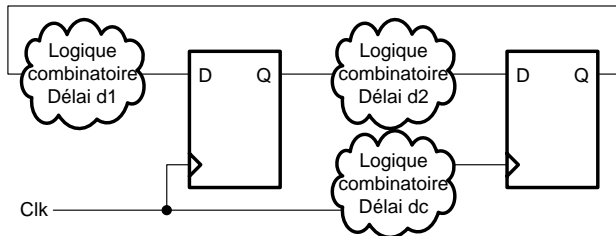
# Bascules D: Mauvais design



- Pourquoi?



# Bascules D: Mauvais design



- Pourquoi?
- Si  $dc$  est trop grand par rapport à  $d2$ , la valeur à l'entrée de la deuxième bascule n'est pas correcte.

# Composants Séquentiels: Registre à décalage (1)

```
process (clk)
begin
    if rising_edge (clk) then
        if decalage_g = '1' then
            reg <= reg(size-2 downto 0) & '0';
        elsif decalage_r = '1' then
            reg <= '0' & reg(size-1 downto 1);
        end if
    end if;
end process;
```

## Registre à décalage (2)

Avec signaux de reset et chargement d'une valeur en parallel.

```

process (clk, rst)
begin
    if rst=RST_ACTIVE then
        reg <= valeurs initiales;
    elsif rising_edge(clk) then
        if load = '1' then
            reg <= bus_entree_parallel;
        elsif decalage_g = '1' then
            reg <= reg(size-2 downto 0) & '0';
        elsif decalage_r = '1' then
            reg <= '0' & reg(size-1 downto 1);
        end if
    end if;
end process;

```

## Registre à décalage (3)

```
process (clk, rst)
begin
  if rst=RST_ACTIVE then
    reg <= valeurs initiales;
  elsif rising_edge(clk) then
    if load = '1' then
      reg <= bus_entree_parallel;
    elsif decalage_g = '1' then
      reg <= reg(size-2 downto 0) & entree_d;
    elsif decalage_r = '1' then
      reg <= entree_g & reg(size-1 downto 1);
    end if
  end if;
end process;
```

# Composants Séquentiels: Compteur (1)

```
...  
-- déclaration du signal compteur sur 8 bits  
signal compteur: std_logic_vector(size-1 downto 0);  
...  
process(clk)  
  begin  
    if rising_edge(clk) then  
      compteur <= compteur + 1;  
    end if;  
end process;
```

## Compteur (2)

Avec reset et enable:

```
process(clk,rst)
begin
  if rst=RST_ACTIVE then
    reg <= (others => '0');
  elsif rising_edge(clk) then
    if enable = '1' then
      compteur <= compteur + 1;
    end if
  end if;
end process;
```

## Compteur (3)

Avec chargement d'une valeur en parallél et un signal de contrôle pour incrémenter ou décrémenter.

```

process (clk, rst)
begin
  if rst=RST_ACTIVE then
    reg <= (others => '0');
  elsif rising_edge(clk) then
    if load = '1' then
      compteur <= bus_entree_parallel;
    elsif enable = '1' then
      if up_down='1' then
        compteur <= compteur + 1;
      else
        compteur <= compteur - 1;
      end if
    end if
  end if;
end process;

```

# Types de machines d'états

- Posons:
  - X: le vecteur d'entrée
  - S: le vecteur d'état
  - Y: le vecteur de sortie
- On définit les types de machines suivants:



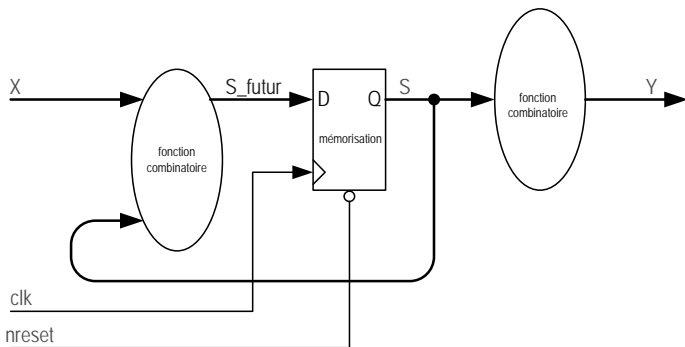
# Types de machines d'états

- Posons:
  - X: le vecteur d'entrée
  - S: le vecteur d'état
  - Y: le vecteur de sortie
- On définit les types de machines suivants:
  - Moore
    - Le vecteur de sortie Y est fonction du vecteur d'état S
    - $Y=f(S)$

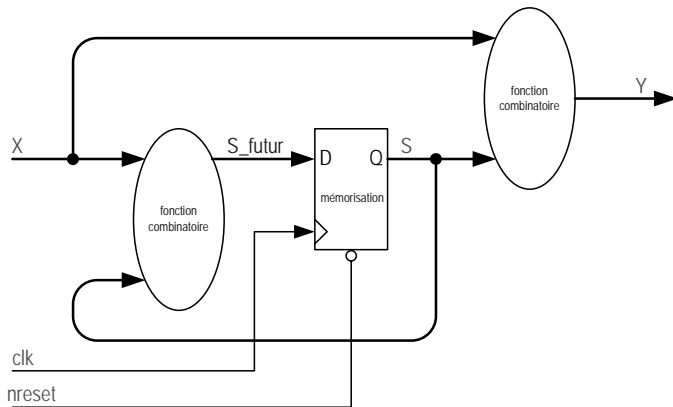
# Types de machines d'états

- Posons:
  - X: le vecteur d'entrée
  - S: le vecteur d'état
  - Y: le vecteur de sortie
- On définit les types de machines suivants:
  - Moore
    - Le vecteur de sortie Y est fonction du vecteur d'état S
    - $Y=f(S)$
  - Mealy
    - Le vecteur de sortie Y est fonction du vecteur d'état S ET du vecteur d'entrée X
    - $Y=f(S,X)$

# Machine de Moore: $Y=f(S)$

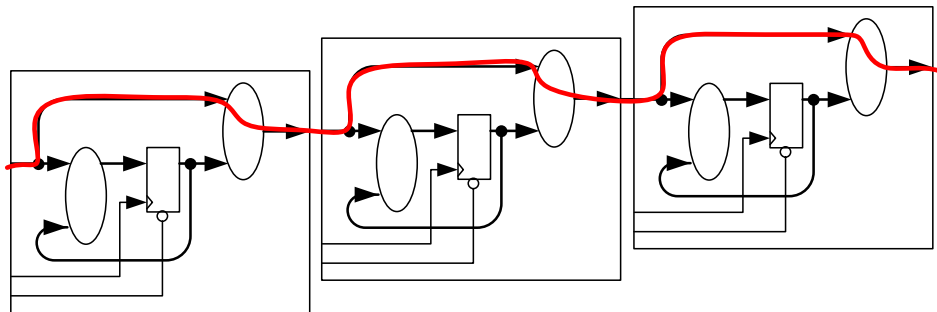


# Machine de Mealy: $Y=f(X,S)$

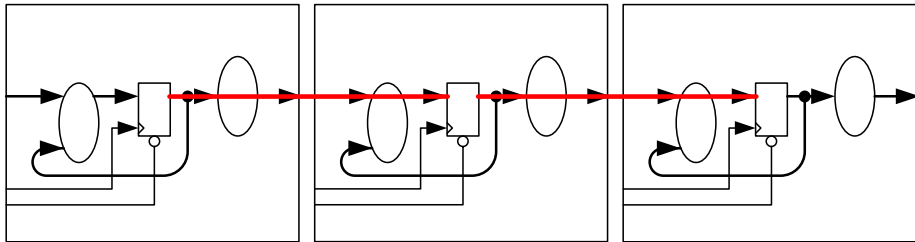


# Machine de Mealy: Problème

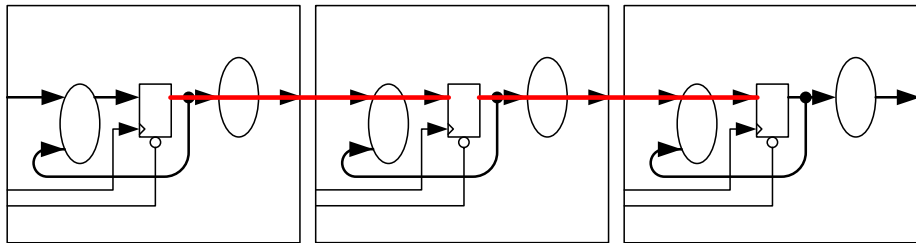
- En chaînant des machines de Mealy, il faut éviter de propager les signaux combinatoires!!!!!!



# Machine de Moore: Pas de problème, mais...

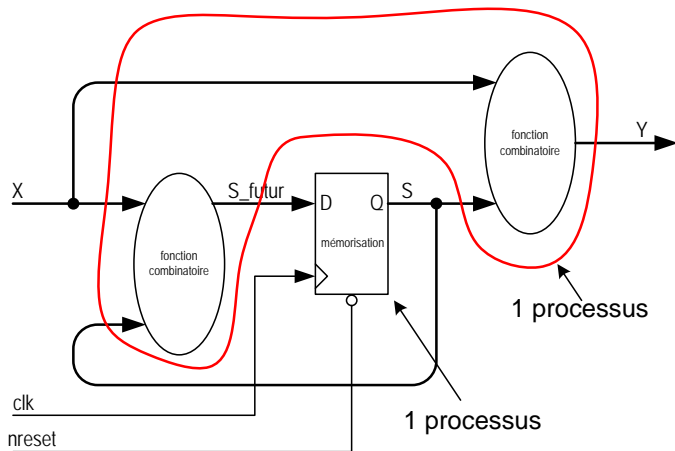


# Machine de Moore: Pas de problème, mais...



- Il y a forcément un délai d'un coup d'horloge pour modifier les sorties.

# Machine de Mealy: Implémentation



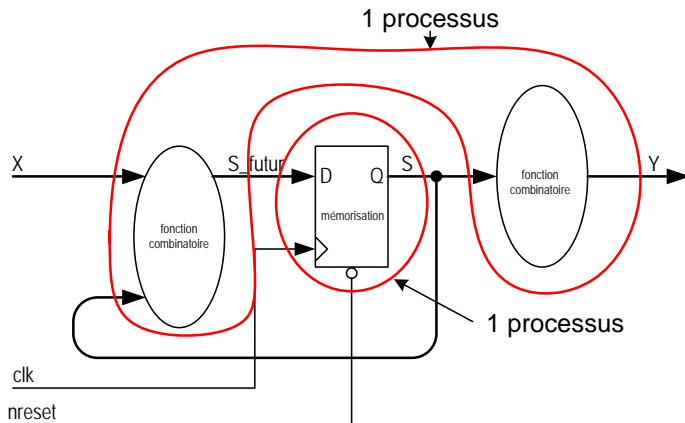


# Machine de Mealy: 2 processus

```
sorties_n_etats: process(etat,entrees)
begin
    -- définition des sorties en fonction de l'état et des entrées
    ...
    -- définition de l'état suivant en fonction de l'état et des entrées
    etat_futur<= ...;
end process;

etats: process(clk,rst)
begin
    if rst=RST_ACTIVE then
        etat <= valeurs initiales
    elsif rising_edge(clk) then
        etat <= etat_futur;
    end if;
end process;
```

# Machine de Moore: Implémentation à 2 processus (a)



# Machine de Moore: Implémentation à 2 processus (a)

```
sorties_etatfutur: process(etat,entrées) begin
    -- définition des sorties en fonction de l'état
    ...
    -- définition de l'état suivant en fonction de l'état et des entrées
    etat_futur <= ...;
end process;

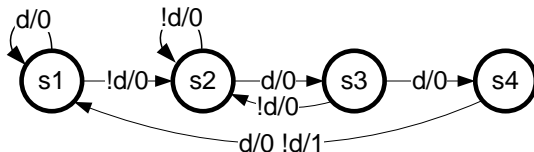
etats: process(clk,rst)
begin
    if rst=RST_ACTIVE then
        etat <= valeurs initiales
    elsif rising_edge(clk) then
        -- affectation des registres avec l'état futur
        etat <= etat_futur;
    end if;
end process;
```

# Machine de Mealy: Exemple

- Un détecteur de séquence 0110.
- Le signal de détection passe à '1' de manière asynchrone lorsque la séquence est détectée.

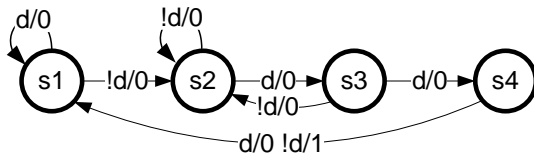
# Machine de Mealy: Exemple

- Un détecteur de séquence 0110.
- Le signal de détection passe à '1' de manière asynchrone lorsque la séquence est détectée.



# Machine de Mealy: Exemple

- Un détecteur de séquence 0110.
- Le signal de détection passe à '1' de manière asynchrone lorsque la séquence est détectée.



état	!d	d
s1	s2/0	s1/0
s2	s2/0	s3/0
s3	s2/0	s4/0
s4	s1/1	s1/0

# Machine de Mealy: codage One hot

état	!d	d
s1	s2/0	s1/0
s2	s2/0	s3/0
s3	s2/0	s4/0
s4	s1/1	s1/0

$$s1^+ = s1d + s4$$

$$s2^+ = s1\bar{d} + s2\bar{d} + s3\bar{d}$$

$$s3^+ = s2d$$

$$s4^+ = s3d$$

$$detected = s4\bar{d}$$

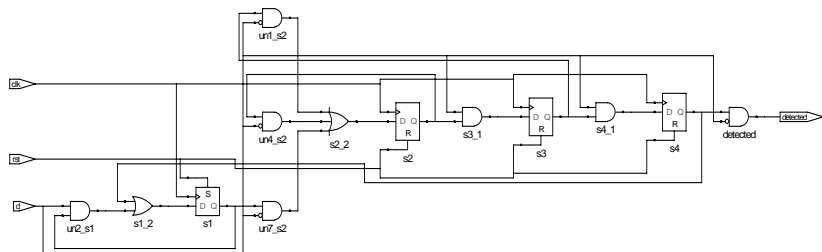
# Machine de Mealy: One Hot

## Codage 1 parmi M (one hot)

```
architecture flot_one_hot of seqdetect is
  signal s1, s2, s3, s4: std_logic;
begin
  detected<=s4 and not d;
  process(rst,clk)
  begin
    if rst='1' then
      s1<='1';
      s2<='0';
      s3<='0';
      s4<='0';
    elsif rising_edge(clk) then
      s1<=(s1 and d) or s4;
      s2<=(s1 and not d) or (s2 and not d) or (s3 and not d);
      s3<=s2 and d;
      s4<=s3 and d;
    end if;
  end process;
end flot_one_hot;
```



# Machine de Mealy: Synthèse



# Machine de Mealy: Sous forme de table d'états

```
architecture state_table of seqdetect is

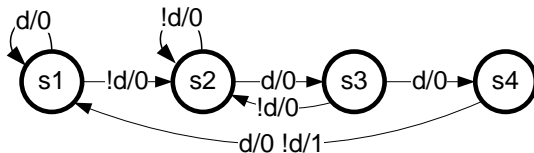
    type state_type is (s1, s2, s3, s4);
    signal state: state_type;
    type state_table_type is array(state_type,0 to 1) of state_type;
    type out_table_type is array(state_type,0 to 1) of std_logic;
    constant state_table:state_table_type:=((s2,s1),(s2,s3),(s2,s4),(s1,s1));
    constant output_table:out_table_type:=(('0','0'),('0','0'),('0','0'),('1','1'),

begin

    process(clk,rst)
    begin
        if rst='1' then
            state<=s1;
        elsif rising_edge(clk) then
            state<=state_table(state,conv_integer(d));
        end if;
    end process;

    detected<=output_table(state,conv_integer(d));
end state_table;
```

# Machine de Mealy: Sous forme de graphe d'états



```

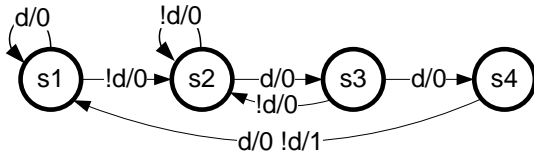
library ieee;
use ieee.std_logic_1164.all;

entity seqdetect is
port(
    clk: in std_logic;
    rst: in std_logic;
    d: in std_logic;
    detected: out std_logic);
end seqdetect;
  
```

```

architecture comp of seqdetect is
type state_type is (s1,s2,s3,s4);
signal state: state_type;
signal n_state: state_type;
begin
    p2: process(rst,clk)
    begin
        if rst='1' then
            state<=s1;
        elsif rising_edge(clk) then
            state<=n_state;
        end if;
    end process;
  
```

# Machine de Mealy: Sous forme de graphe d'états



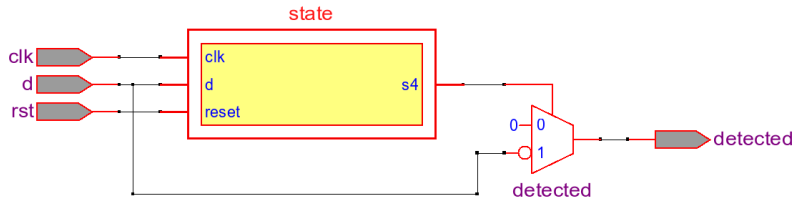
```

p1: process (state,d)
begin
    n_state<=state;
    detected<='0';
    case state is
    when s1=>
        if d='0' then
            n_state<=s2;
        end if;
    when s2=>
        if d='1' then
            n_state<=s3;
        end if;
  
```

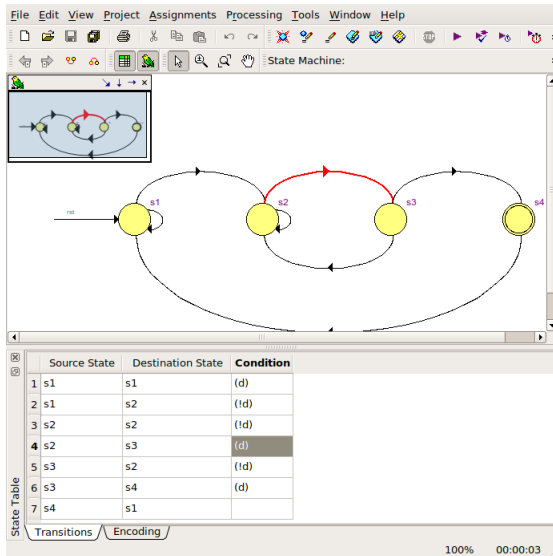
```

    when s3=>
        if d='1' then
            n_state<=s4;
        else
            n_state<=s2;
        end if;
    when s4=>
        n_state<=s1;
        if d='0' then
            detected<='1';
        end if;
    end case;
end process;
end comp;
  
```

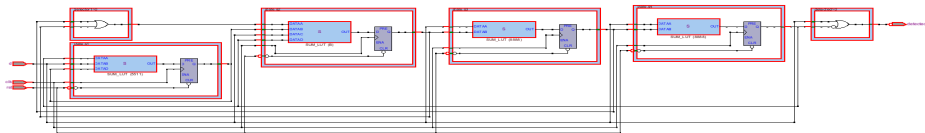
# Machine de Mealy: Synthèse



# Machine de Mealy: Synthèse



# Synthèse: Technology map view



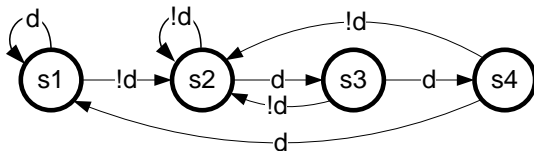
# Machine de Moore: Exemple

- Un détecteur de séquence 011.
- Le signal de détection passe à '1' de manière synchrone lorsque la séquence est détectée.



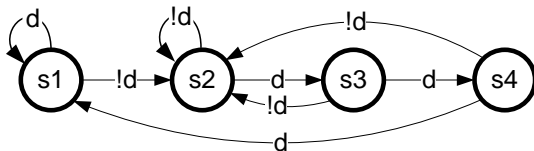
# Machine de Moore: Exemple

- Un détecteur de séquence 011.
- Le signal de détection passe à '1' de manière synchrone lorsque la séquence est détectée.



# Machine de Moore: Exemple

- Un détecteur de séquence 011.
- Le signal de détection passe à '1' de manière synchrone lorsque la séquence est détectée.



état	!d	d	detected
s1	s2	s1	0
s2	s2	s3	0
s3	s2	s4	0
s4	s2	s1	1

# Machine de Moore: One Hot

## Codage 1 parmi M (one hot)

```
architecture flot_one_hot of seqdetect is
  signal s1, s2, s3, s4: std_logic;
begin
  detected<=s4;
  process(rst,clk)
  begin
    if rst='1' then
      s1<='1';
      s2<='0';
      s3<='0';
      s4<='0';
    elsif rising_edge(clk) then
      s1<=(s1 and d) or (s3 and (not d)) or (s4 and d);
      s2<=not d;
      s3<=s2 and d;
      s4<=s3 and d;
    end if;
  end process;
end flot_one_hot;
```

# Machine de Moore: Sous forme de table d'états

```
architecture state_table of seqdetect is

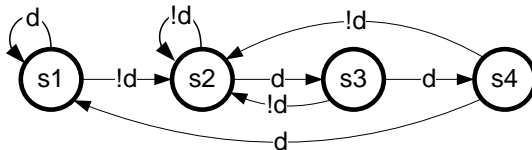
    type state_type is (s1, s2, s3, s4);
    signal state: state_type;
    type state_table_type is array(state_type, 0 to 1) of state_type;
    type out_table_type is array(state_type) of std_logic;
    constant state_table: state_table_type := ((s2, s1), (s2, s3), (s2, s4), (s2, s1));
    constant output_table: out_table_type := ('0', '0', '0', '1');

begin

    process (clk, rst)
    begin
        if rst = '1' then
            state <= s1;
        elsif rising_edge(clk) then
            state <= state_table(state, conv_integer(d));
        end if;
    end process;

    detected <= output_table(state);
end state_table;
```

# Machine de Moore: Sous forme de graphe d'états



```

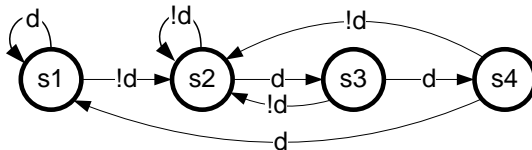
library ieee;
use ieee.std_logic_1164.all;

entity seqdetect is
port(  clk: in std_logic;
      rst: in std_logic;
      d: in std_logic;
      detected: out std_logic);
end seqdetect;
  
```

```

architecture comp of seqdetect is
type state_type is (s1, s2, s3, s4);
signal state: state_type;
signal n_state: state_type;
begin
  p2: process(rst,clk)
  begin
    if rst='1' then
      state<=s1;
    elsif rising_edge(clk) then
      state<=n_state;
    end if;
  end process;
  detected<='1' when state=s4 else '0'
end architecture;
  
```

# Machine de Moore: Exemple



```

p1: process (state,d)
begin
  n_state<=state;
  case state is
  when s1=>
    if d='0' then
      n_state<=s2;
    end if;
  when s2=>
    if d='1' then
      n_state<=s3;
    end if;
  
```

```

    when s3=>
      if d='1' then
        n_state<=s4;
      else
        n_state<=s2;
      end if;
    when s4=>
      if d='0' then
        n_state<=s2;
      else
        n_state<=s1;
      end if;
    end case;
  end process;
end comp;

```

# Machine de Moore: Synthèse

