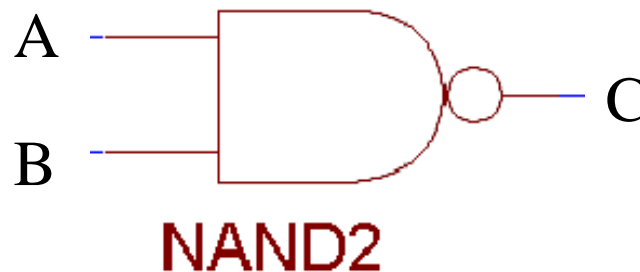


VHDL et la synthèse de circuits séquentiels



Qu'est-ce que la Synthèse?

- Générer du hardware depuis HDL
- Eg. **C** <= **A nand B** donne



- Pas si simple!
- Nécessite un processus complexe afin d'optimiser un arrangement entre le temps et l'espace.

Règles de conception

- Systèmes synchrones(simu fonctionnelle simplifiée et puissante, analyse temporelle statique possible,) :
- 1 SEULE horloge (broche globale du composant)
- Si plusieurs domaines d horloges, il faut des FIFOs tampons
- resynchroniser tous les signaux asynchrones pour éviter la métastabilité entre 1 et 0

Règles de conception

- Adopter une démarche qualité (dénomination, hierarchisation, testbench...)
- Attention aux assignations incomplètes (mémorisation implicite)
- Attention à l'utilisation des variables
- Penser implantation (orienter le compilateur) : il faut connaître l'architecture du composant
- Trouver le bon compromis entre ressources et vitesses en choisissant le bon degré de parallélisation (pas compact mais rapide) ou de sérialisation (compact mais lent)

Danger!

- Time expressions – after
- Assert, Text I/O
- Configuration declarations
- Dynamic Loops
- Initial values
- Sensitivity lists



Simulation
seulement



Modifie les
résultats de
synthèse

Eléments séquentiels de base

- **Latches**

- Moins complexe que FFs
- Perturbe les synchronizations et l'analyse temporelle
- Des latches sont parfois générés sans le savoir!!

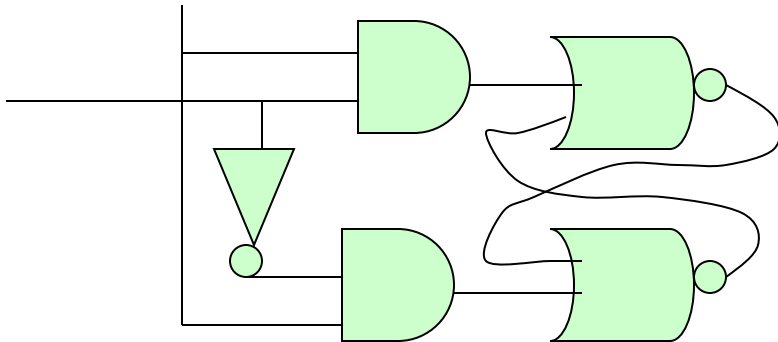
- **Flip-flops**

- Edge sensitive , front montant ou descendant
- Asynchronous et synchronous set and reset

Description with Latches

- Data input: D; enable: LE; output: Q.

```
signal D, LE, Q: bit ;  
...  
b1 : process (D, LE)  
begin  
  if (LE = '1') then  
    Q <= D;  
  end if ;  
end process ;
```



Observe that this syntax example involves storing the signal in a memory

Modeling Latches

- **when:**

```
Q <= D when En='1' else Q;
```

```
...
```

```
Q <= D when En='1' else  
unaffected;
```

```
(VHDL 93)
```

- if controlled by Enable

```
library IEEE;
```

```
    use IEEE.Std_Logic_1164.all;
```

```
entity Latch is
```

```
    port (D, En : in Std_Logic;
```

```
          Q  : out Std_Logic);
```

```
end Latch;
```

```
architecture RTL of Latch is
```

```
begin
```

```
    L1: process (D, En)
```

```
    begin
```

```
        if En='1' then Q <= D;
```

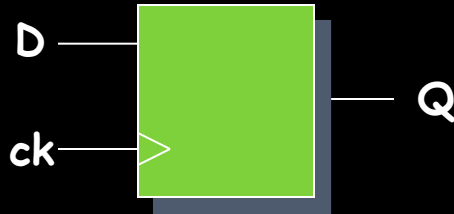
```
        end if;
```

```
    end process;
```

```
end RTL;
```


LOGIQUE SEQUENTIELLE / bascules

bascule D « edge-triggered »



1^{ère} possibilité de description:

utilisation de l'instruction **WAIT**

- le PROCESS n'est exécuté que lors du passage de 0 à 1 du signal d'horloge
- le PROCESS ne comporte pas de liste de sensibilité
- il est nécessaire de passer par un signal interne (« memo » dans l'exemple)

```
ENTITY DFF IS
  PORT (D,ck : IN BIT;
        Q : OUT BIT);
END DFF ;
```

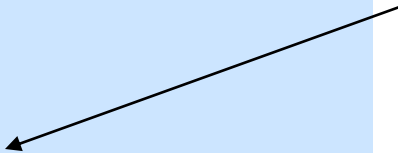
```
ARCHITECTURE arch1 OF DFF IS
  SIGNAL memo : BIT;
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL ck='1';
    memo <= D;
  END PROCESS;
  Q <= memo;
END arch1;
```

Description using Edge Triggered Flip Flops

- Flip Flop is based on a clock signal.
- Rising/Falling edge triggered.

```
signal D, Q, clk : bit ;  
....  
process (clk)  
begin  
  if (clk'event and clk='1') then  
    Q <= D;  
  end if ;  
end process ;
```

Rising edge



Descriptions of Synchronous Set/Reset Flip Flops

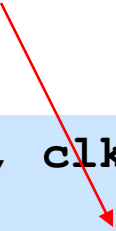
- Data and reset is NOT on sensitivity list.

```
signal D, Q, clk, reset : bit ;  
...  
process (clk)  
begin  
    if (clk'event and clk = '1') then  
        if reset = '1' then  
            D <= '0' ;  
        else  
            Q <= D ;  
        end if ;  
    end if ;  
end process ;
```

D is symbol
of next state,
not current
state

Asynchronous Set/Reset Flip Flops

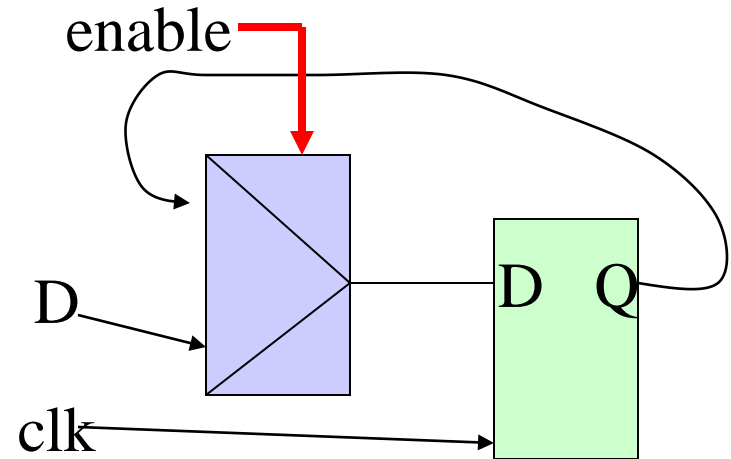
- Reset is ON sensitivity list.



```
signal D, Q, clk, reset : bit ;
...
process (clk, reset)
begin
    if (reset = '1') then
        Q <= '0' ;
    elsif (clk'event and clk = '1') then
        Q <= D ;
    end if ;
end process ;
```

Clock Enable Flip Flops

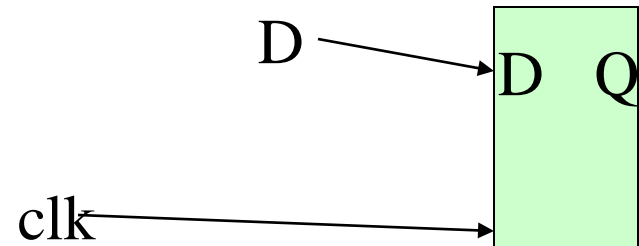
```
signal D, Q, enable, clk : bit ;  
...  
process (clk)  
begin  
  if (clk'event and clk='1') then  
    if (enable='1') then  
      Q <= D ;  
    end if ;  
  end if ;  
end process ;
```



Flip Flops inferred by Wait Until

```
signal D, Q, clk : bit ;  
...  
process  
begin  
    wait until clk'event and clk='1' ;  
    Q <= D ;  
end process ;
```

- No sensitivity lists
- No asynchronous reset



Why so many FF syntax?

- Highlight the importance of writing good HDL code
- Coding style affects synthesized structure directly
- Internal working of synthesizers

Descriptions of Tristate Buffers

- 3 output states: 0, 1, Z (high impedance)

```
entity tristate is
  port ( D, en: in std_logic ;
         Q : out std_logic);
end tristate ;

architecture EG of tristate is
begin
  Q <= D when en = '1' else 'Z' ;
end EG ;
```

```
p1 : process (en, D)
begin
  if (en='1') then
    Q <= D ;
  else
    Q <= 'Z' ;
  end if ;
end process ;
```

Observe that because I used symbol Z the system inferred tri-state circuit by itself

Description of wired circuit using Tristate Buffers

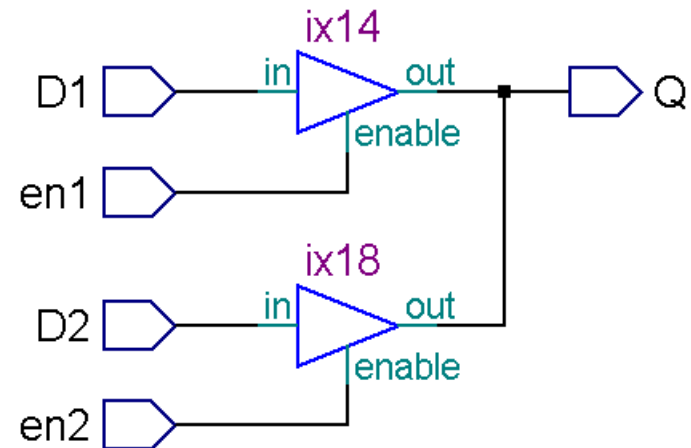
- Simultaneous assignment to 1 signal
- Does not verify exclusivity

We do not specify if this is wired OR or wired AND or what

```
library IEEE;
use IEEE.std_logic_1164.all;

entity tristate is
port ( D1, D2 , en1, en2 : in std_logic ;
      Q : out std_logic);
end tristate ;

architecture EG of tristate is
begin
  Q <= D1 when en1 = '1' else 'Z' ;
  Q <= D2 when en2 = '1' else 'Z' ;
end EG ;
```



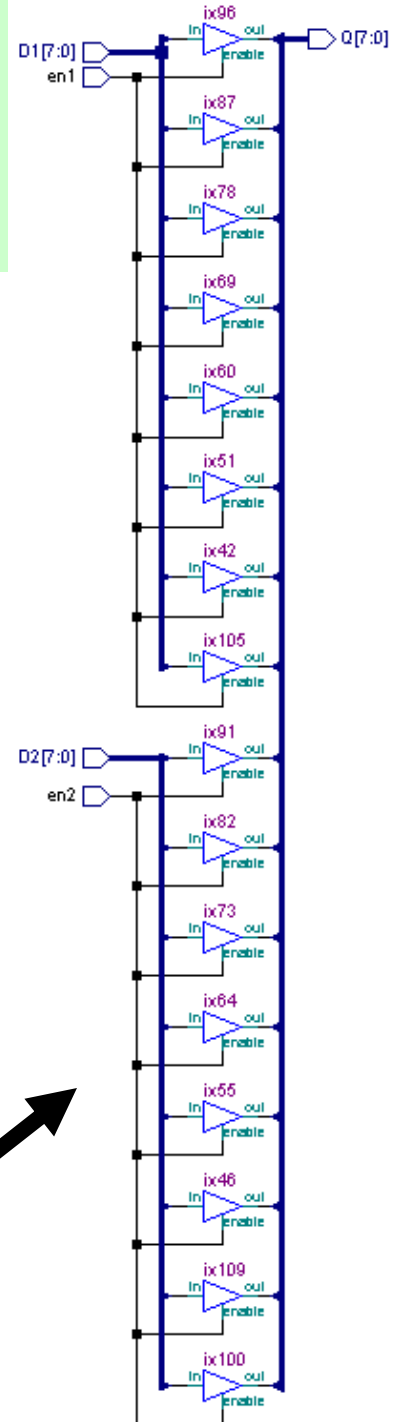
Busses with tri-state buffers

- Use arrays in the declarations

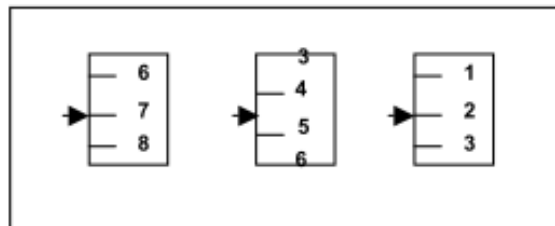
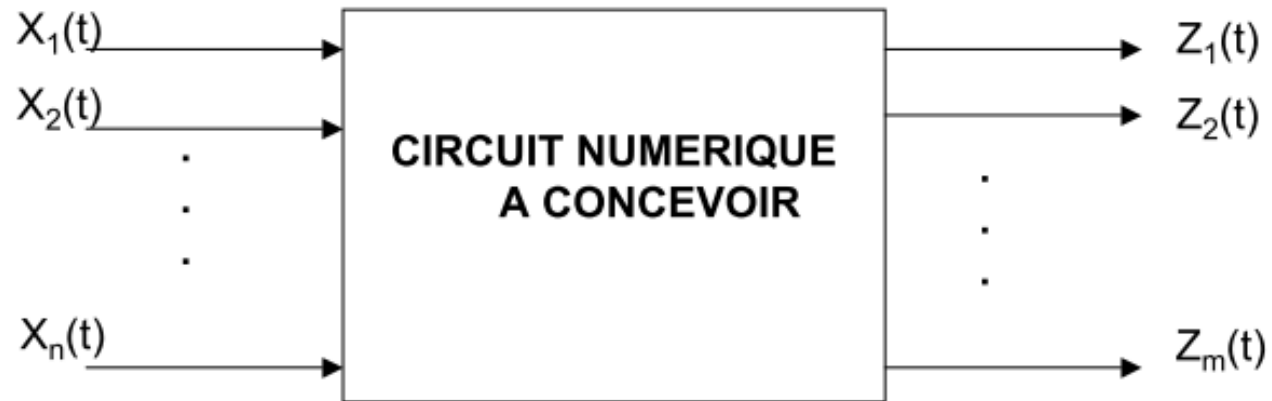
```
entity tristate is
port ( D1, D2: in std_logic_vector (7 downto 0);
      en1, en2 : in std_logic;
      Q : out std_logic_vector (7 downto 0) );
end tristate ;

architecture EG of tristate is
begin
  Q <= D1 when en1 = '1' else "ZZZZZZZZ" ;
  Q <= D2 when en2 = '1' else "ZZZZZZZZ" ;
end EG ;
```

This description and circuit describes selector circuit realized with tri-state buffers on a bus – this is an important circuit

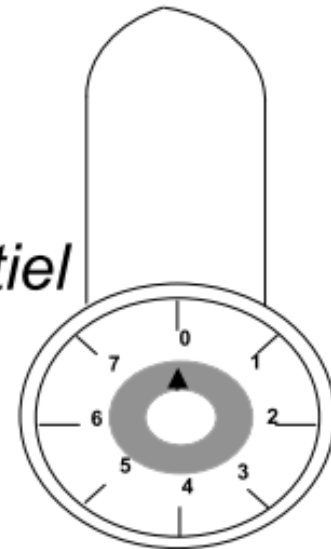


Automatisation des procédés



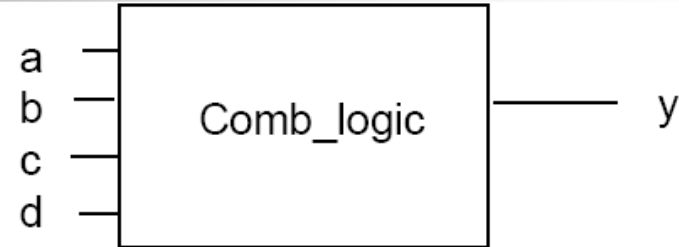
Combinatoire

Séquentiel



Combinational Logic

Combinational Logic



Models representing a combinational block described by the truth table are given below:

AB		"00"	"01"	"11"	"10"
CD	"00"	'0'	'1'	'1'	'0'
	"01"	'1'	'1'	'1'	'0'
	"11"	'0'	'1'	'1'	'1'
	"10"	'0'	'0'	'0'	'0'

$$y = C'b + db + cda + c'da'$$

Combinational Logic

```
entity COMB_LOGIC is    --  $y = c'd + db + cda + c'da'$ 
    generic (P_DELAY : time);
    port (A, B, C, D : in Bit;
          Y : out Bit);
end COMB_LOGIC;        -- behavior from the truth table
                        -- process statement and sequential statements

architecture ARCH2 of COMB_LOGIC is
begin
    process (A, B, C, D)
        variable TEMP : Bit_Vector (3 DOWNT0 0);
    begin
        TEMP := A&B&C&D;
        case TEMP is
            when b"0000" | b"1000" | b"1001" | b"0011" |
                 b"0010" | b"0110" | b"1110" | b"1010" =>
                y <= '0';
            when OTHERS => y <= '1';
        end case;
    end process;
end ARCH2;
```

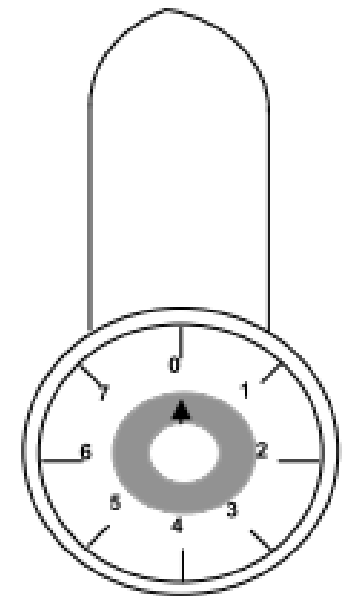
Catégories de machines séquentielles



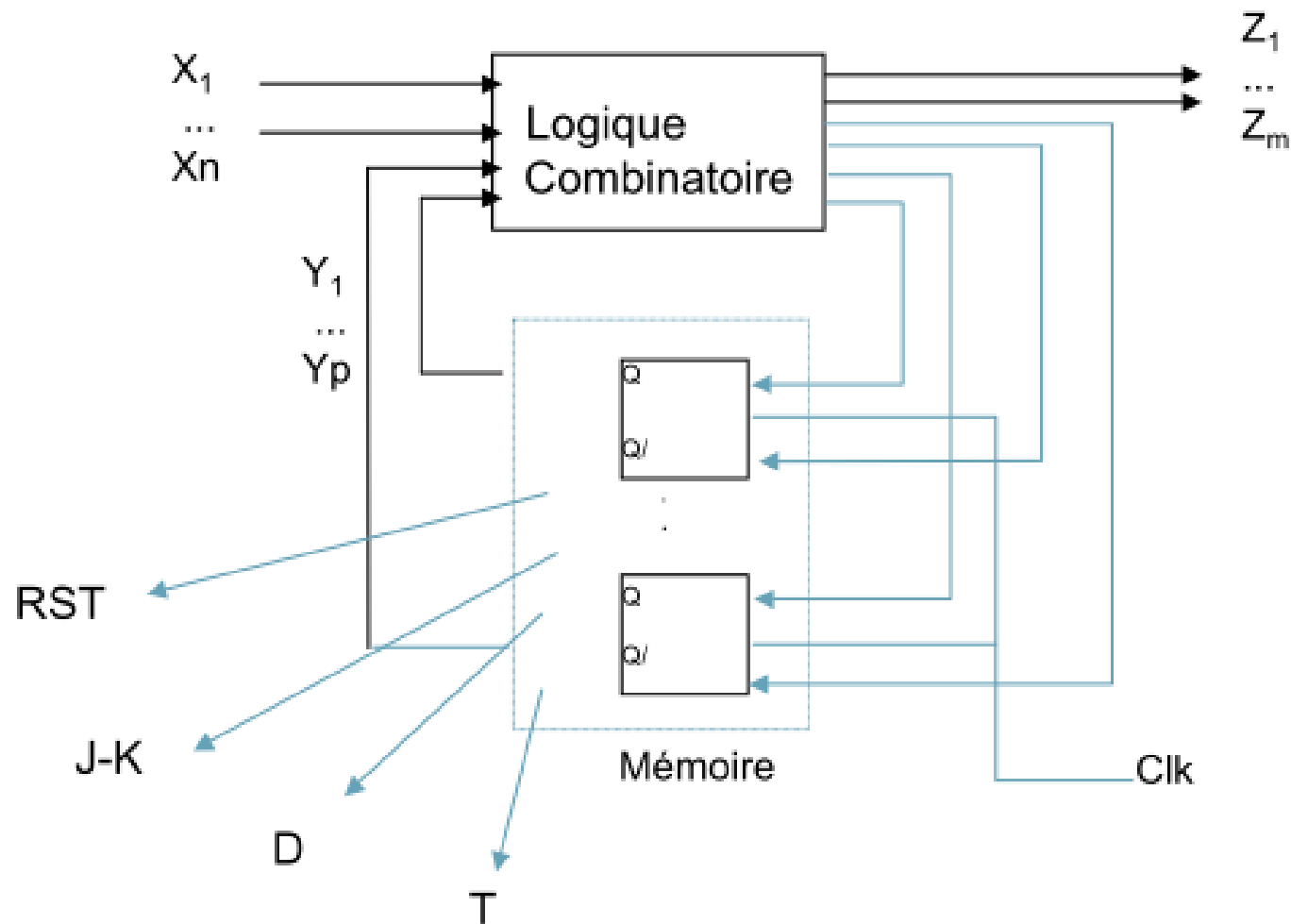
• Synchrones

• Asynchrone pulsé *ou* fondamental

Y'_1
 \dots
 Y'_p

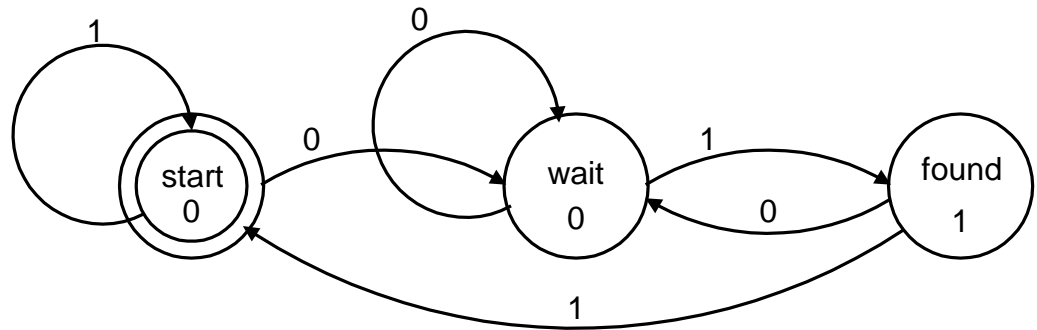


Modèle générale (machines synchrones)

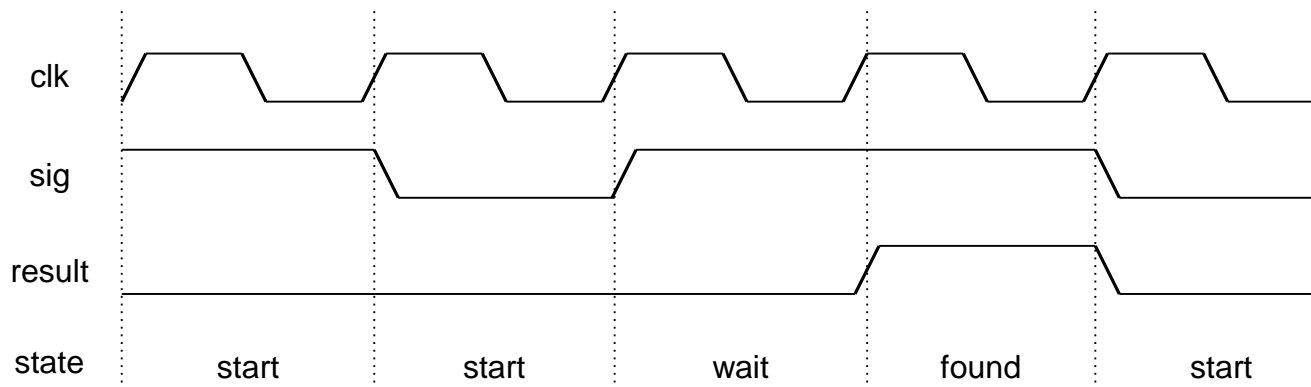


State Machines

- Finite State Machines (FSM) are a key tool of logic design
- A synthesizer can perform state optimization on FSMs to minimize the circuit area / delay
- This optimization is only available if the FSM model fits templates

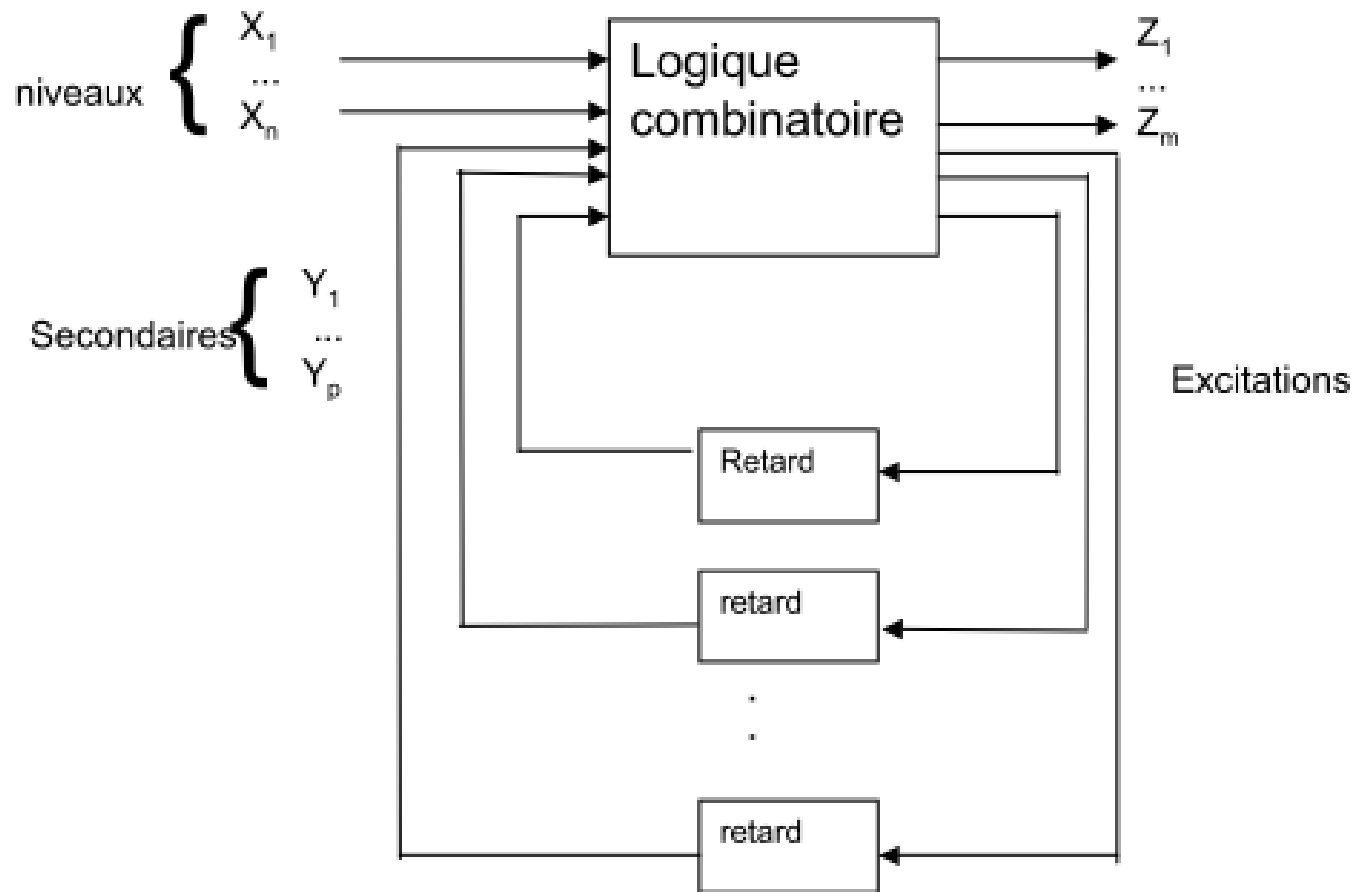


State Diagram for Pulse Generator



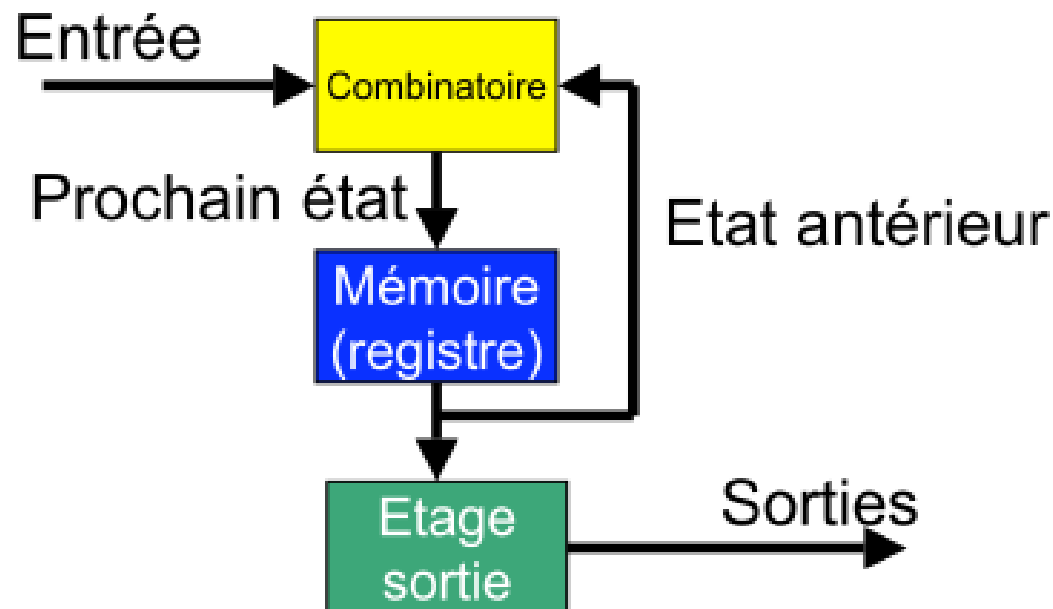
Timing Diagram for Pulse Generator

Modèle général (machine asynchrone)



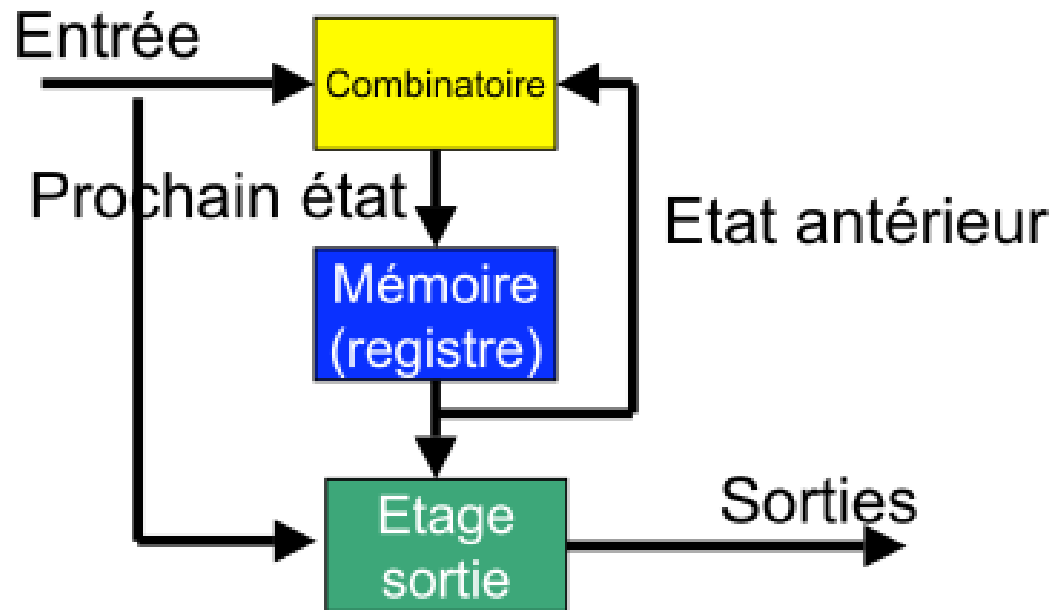
Machine de Moore

- Les sorties dépendent des états présents. Un bloc de logique combinatoire traite les entrées et l'état antérieur des sorties et alimente un bloc de bascules en sortie. Ainsi, les sorties changent de manière synchrone sur un front d'horloge.



Machine de Mearly

- A la différence de la machine de Moore, la sortie peut changer lors d'un changement de l'entrée et ce indépendamment de l'horloge.
- Une machine de Mealy est donc asynchrone.
- Une machine de Moore changera seulement sur des transitions du signal d'horloge : son fonctionnement synchrone facilite son utilisation dans tout circuit synchrone, ce qui en fait un avantage sur la machine de Mealy. Il existe cependant une version synchrone de la machine de Mealy.

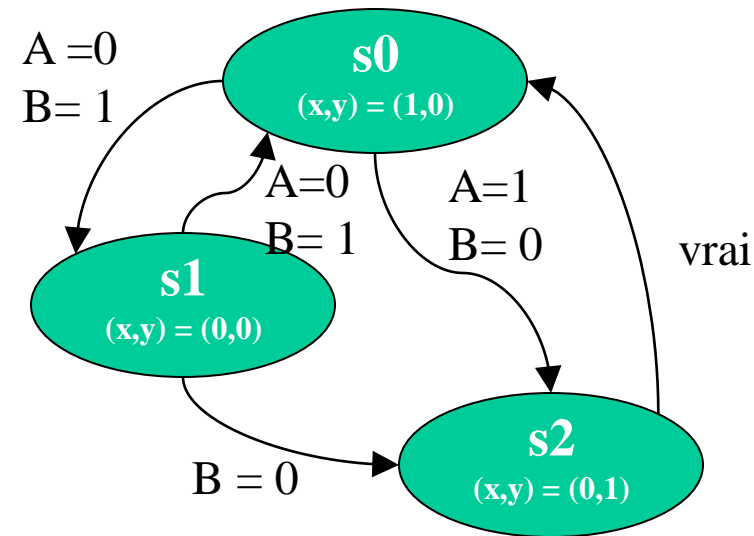


MACHINES d' ETAT (exemple)

```

ENTITY machine IS
  PORT (a,b,clk : IN BIT;
        x,y : OUT BIT);
END machine ;

ARCHITECTURE moore OF machine IS
  TYPE STATE_TYPE IS (s0,s1,s2);
  SIGNAL etat : STATE-TYPE;
BEGIN
  PROCESS
  BEGIN
    WAIT UNTIL clk = '1';
    CASE etat IS
      WHEN s0 => IF a='0' AND b='1' THEN etat <= s1;
                  ELIF a='1' AND b='0' THEN etat <= s2;
                  ELSE etat <= s0;
                  END IF;
      WHEN s1 => IF a='0' AND b='1' THEN etat <= s0;
                  ELIF b='0' THEN etat <= s2;
                  ELSE etat <= s1;
                  END IF;
      WHEN s2 => etat <= s0;
      WHEN OTHERS => etat <= s0;
    
```

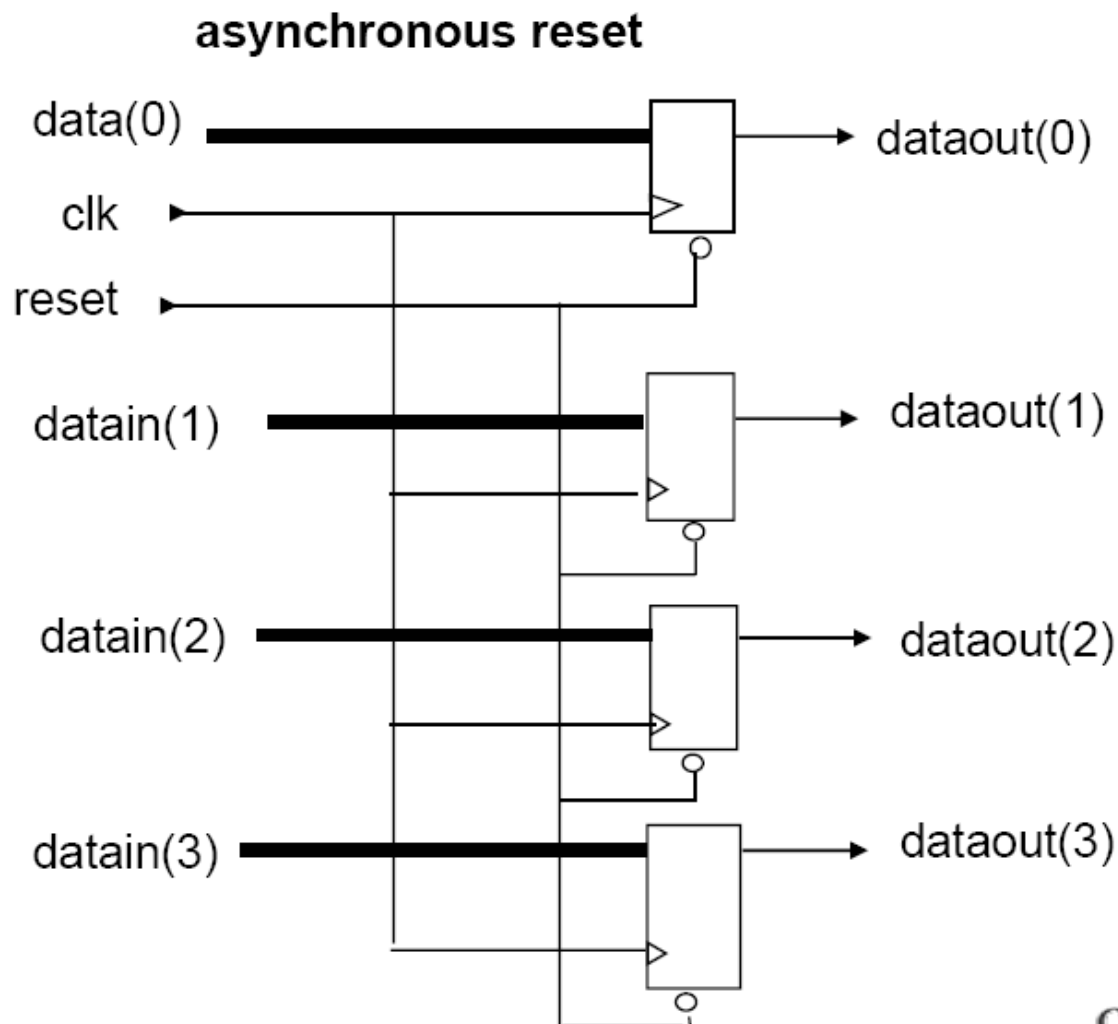


```

END CASE;
END PROCESS;
x <= '1' WHEN etat = s0 ELSE '0';
y <= '1' WHEN etat = s2 ELSE '0';
END moore;

```

VHDL Register





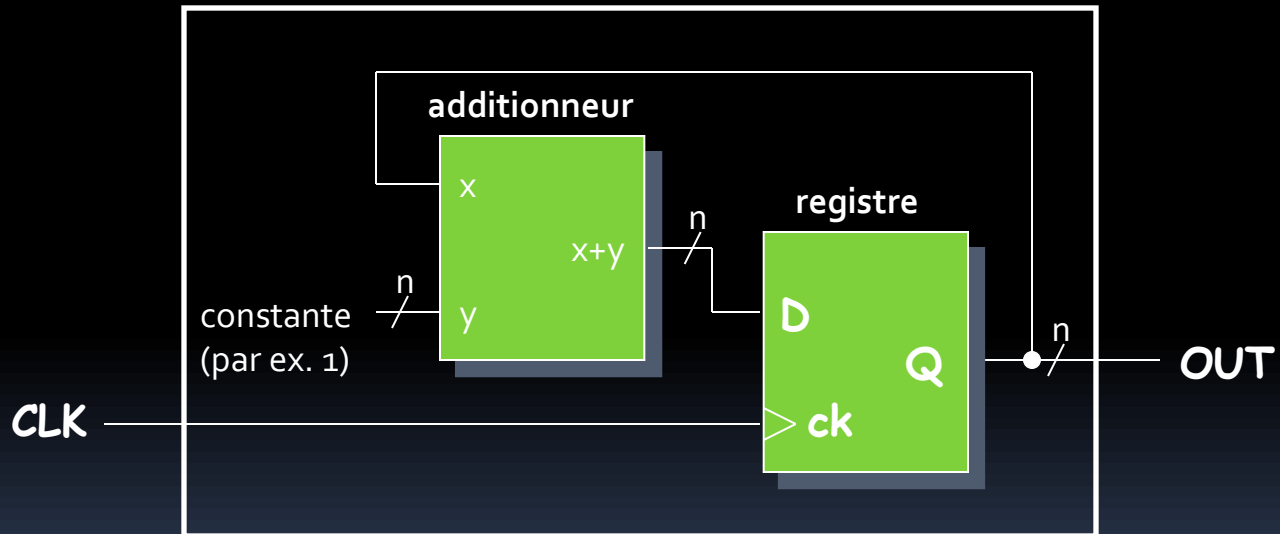
VHDL Register

```
process (RESET, CLK)
begin
    if RESET = '0' then
        DATAOUT <= "0000";
    elsif CLK'event and CLK = '1' then -- rising edge
        DATAOUT <= DATAIN; -- of clock
    end if;
end process;
```

Asynchronous reset does not depend on clock.

LOGIQUE SEQUENTIELLE / compteurs

compteur: association d'un registre et d'un additionneur



LOGIQUE SEQUENTIELLE / compteurs (suite)

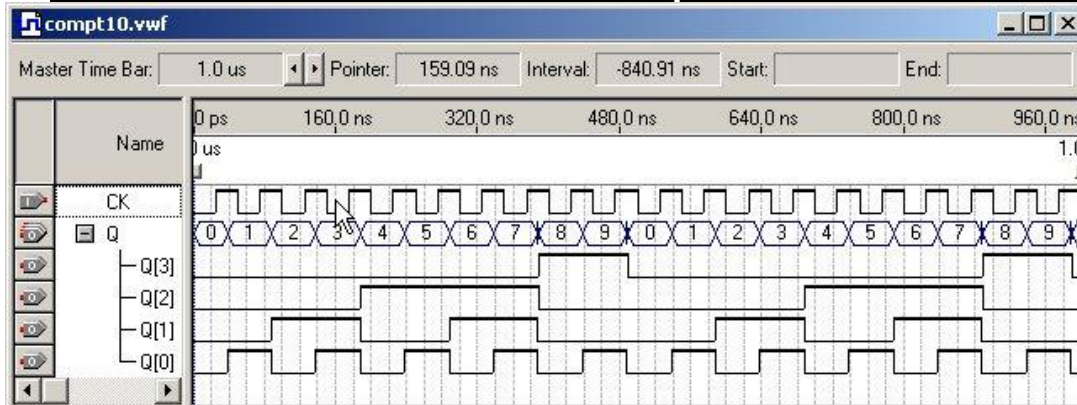
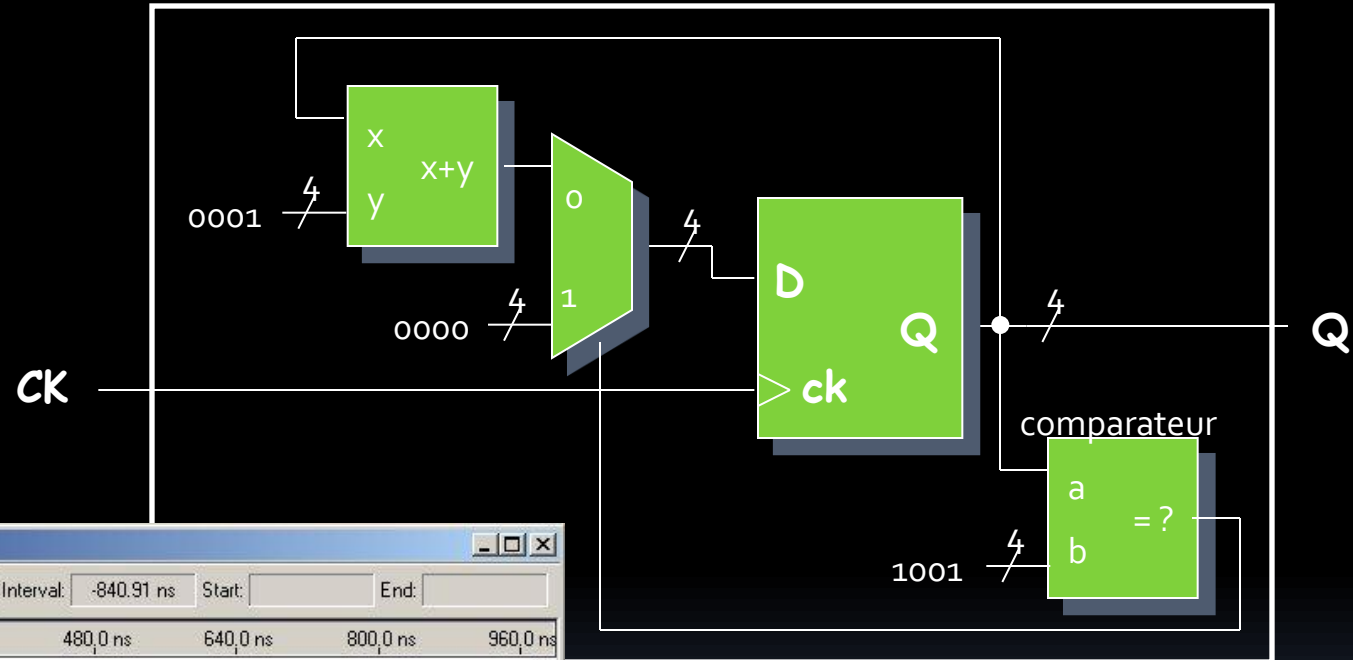
exemple: compteur 4 bits modulo 10

```
ENTITY compt10 IS
    PORT (CK : IN BIT;
          Q : OUT INTEGER RANGE 0 TO 9 );
END compt10;
ARCHITECTURE mod10 OF compt10 IS
    SIGNAL valeur : INTEGER RANGE 0 TO 9;
BEGIN
    PROCESS (CK)
    BEGIN
        IF CK'EVENT AND CK='1' THEN
            IF valeur = 9 THEN valeur <= valeur + 1;
            ELSE valeur <= 0;
            END IF;
        ELSE valeur <= valeur;
        END IF;
    END PROCESS;
    Q <= valeur;
END mod10;
```


LOGIQUE SEQUENTIELLE / compteurs (suite)

pour info: structure physique du compteur 4 bits modulo 10

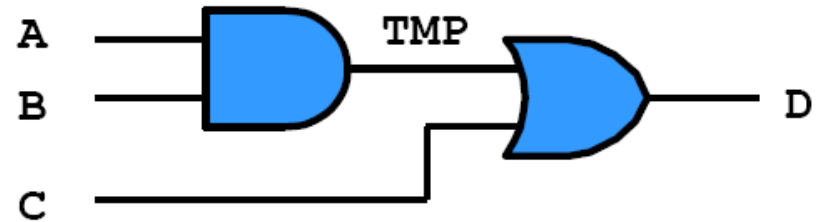
- synthétisé par le compilateur
- transparent pour le concepteur



A word of caution.

Hardware realization of VHDL objects

```
signal A,B,C,D: bit
...
NO_MEMORY: process (A,B,C)
variable TMP: bit;
begin
    TMP := A and B;
    D <= TMP or C;
end process;
```



**Implementation of TMP :
wire**

```
signal A,B,C: bit
...
IS_IT_LATCH: process (A,B,C)
variable TMP: bit;
begin
    C <= TMP and B;
    TMP := A or C;
end process;
```

**Implementation of TMP :
latch. Do you real want it?**

Séquentialiser si besoin!

- Interdire un feedback zéro délai :
$$OP \leq OP + Y ;$$
- syntaxiquement correcte, refusée par la plupart des synthétiseurs
- On utilise un process :
Process (OP, Y)
Begin
$$OP \leq OP + Y ;$$

End process ;

VHDL!!!

- C'est la pratique qui fait le designer !