

# TP d'Electronique Numérique

---

Langage VHDL et logiciel Quartus

# Déroulement des TP

- Pas de comptes-rendus
- CC de TP en séance 3 et en séance 5
- Examen de TP individuel de deux heures en séance 6

# Séance 1

# VHDL

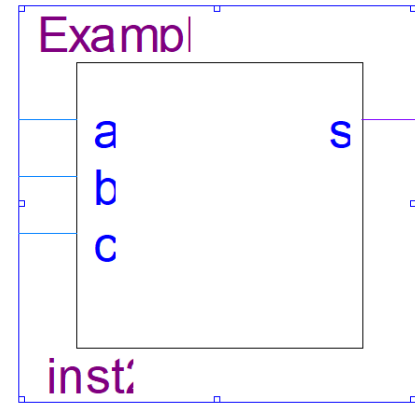
- VHSIC Hardware Description Language
  - Very High Speed Integrated Circuits
- Le mot important est : Description
- Le mot à bannir est : Programmation
- Utilisation depuis les années 1980

# Conception d'une description VHDL

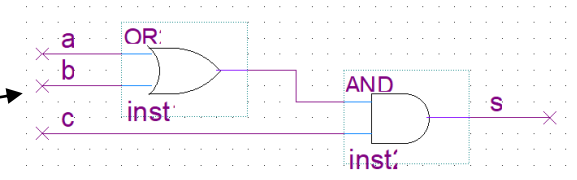
```
1  LIBRARY ieee;  
2  USE ieee.std_logic_1164.all;  
3  USE ieee.numeric_std.all;  
4  
5  ENTITY Example IS  
6  PORT (  
7      a : IN STD_LOGIC;  
8      b : IN STD_LOGIC;  
9      c : IN STD_LOGIC;  
10     s : OUT STD_LOGIC  
11 );  
12 END Example;  
13  
14 ARCHITECTURE archi OF Example IS  
15 BEGIN  
16     PROCESS (a,b,c)  
17     BEGIN  
18  
19         s <= (a OR b) AND c;  
20  
21     END PROCESS;  
22 END archi;
```

Utilisation de bibliothèques

ENTITY  
Création d'un « bloc »



ARCHITECTURE  
« Câblage » de portes logiques



## Autres descriptions possibles

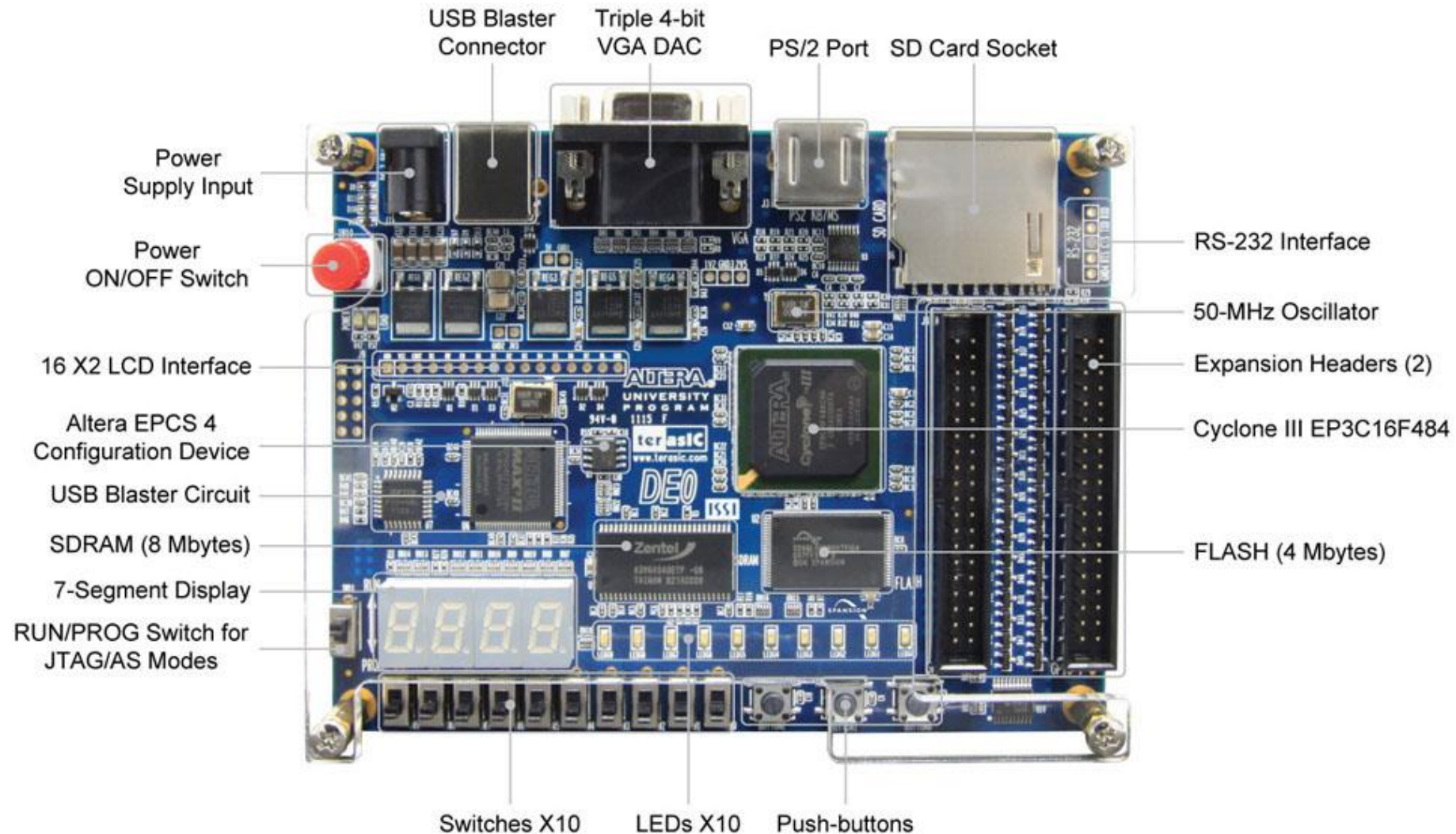
```
14 ARCHITECTURE comportementale OF Example IS
15   L
16 BEGIN
17   P1: PROCESS(a,b,c) IS
18     BEGIN
19       IF C='0' OR ((a='0') AND (b='0')) THEN
20         s <= '0';
21       ELSE
22         s <= '1';
23       END IF;
24     END PROCESS P1;
25 END ARCHITECTURE comportementale;
```

Description comportementale

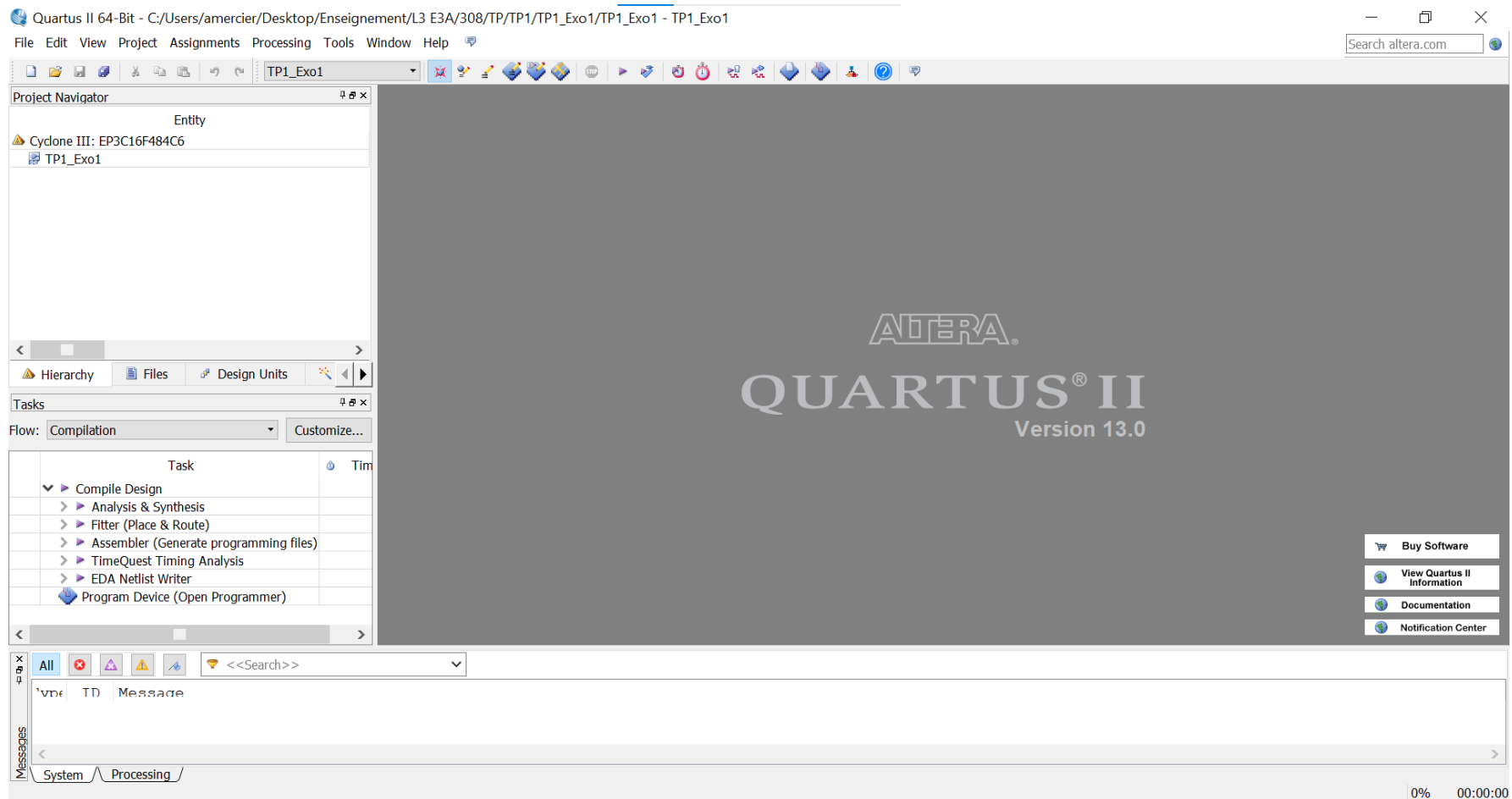
```
28 ARCHITECTURE table_de_verite OF Example IS
29   L
30   signal entrees : STD_LOGIC_VECTOR(2 downto 0);
31 BEGIN
32   P1: PROCESS(entrees) IS
33     BEGIN
34       CASE entrees is
35         WHEN "000" => S <= '0';
36         WHEN "001" => S <= '0';
37         WHEN "010" => S <= '0';
38         WHEN "011" => S <= '1';
39         WHEN "100" => S <= '0';
40         WHEN "101" => S <= '1';
41         WHEN "110" => S <= '0';
42         WHEN "111" => S <= '1';
43         WHEN OTHERS => S <= '0';
44       END CASE;
45     END PROCESS P1;
46
47     entrees <= a & b & c;
48
49 END ARCHITECTURE table_de_verite;
```

Description table de vérité

# La carte d'essai DE0 – FPGA Altera



# Le logiciel Quartus - A vous de jouer !





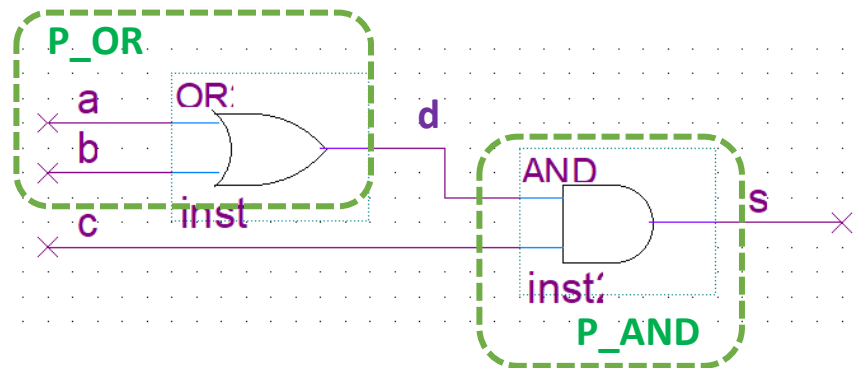
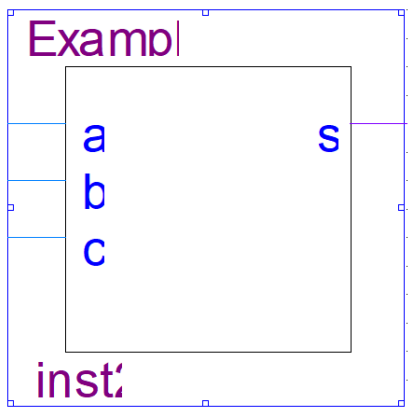
# Séance 2

# Processus (Suite) – Description comportementale

```

1  LIBRARY ieee;
2  USE ieee.std_logic_1164.all;
3  USE ieee.numeric_std.all;
4
5  ENTITY Example IS
6  PORT (
7      a : IN STD_LOGIC;
8      b : IN STD_LOGIC;
9      c : IN STD_LOGIC;
10     s : OUT STD_LOGIC
11 );
12 END Example;

```



```

12  L
13  ARCHITECTURE archi2 OF Example IS
14      SIGNAL d : STD_LOGIC;
15
16
17  BEGIN
18
19      P_AND: PROCESS (c,d) IS
20      BEGIN
21          IF c='1' AND d='1' THEN
22              s <= '1';
23          ELSE
24              s <= '0';
25          END IF;
26      END PROCESS P_AND;
27
28      P_OR: PROCESS (a,b) IS
29      BEGIN
30          IF a='1' OR b='1' THEN
31              d <= '1';
32          ELSE
33              d <= '0';
34          END IF;
35      END PROCESS P_OR;
36  END archi2;

```

# Nombres entiers en VHDL

Types arithmétiques entiers présents dans VHDL

```
type integer is range -2147483648 to 2147483647 ; -- 32 bits signés
type natural is range 0 to 2147483647; -- 31 bits non-signés
```

Types arithmétiques entiers à base de tableaux de bits (ajouter paquetage **NUMERIC\_STD**)

```
type unsigned is array ( natural range <>) of std_logic;
type signed is array ( natural range <>) of std_logic;
```

À utiliser de préférence car permet de modéliser n'importe quel nombre entier (64 bits ...)

**Aucune opération arithmétique n'est possible avec des std\_logic\_vector**

## Conversions de type

Si les deux types sont des tableaux de bits, un simple cast suffit

```
A_std_logic_vector_16b <= std_logic_vector(A_unsigned_16b);
```

```
A_unsigned_16b <= unsigned(A_std_logic_vector_16b);
```

Sinon (integer vs tableau de bits), → fonction de conversion

```
A_int <= to_integer(A_unsigned_16b);
```

```
A_unsigned_16b <= to_unsigned(A_int, 16);
```

## Opérateurs disponibles

Opérateurs logiques : not, or, and, nor, nand, xor, xnor

Opérateurs de décalage : sll, srl, sla, sra, rol, ror

Opérateurs relationnels : =, /=, <, >, >=, <=

Opérateurs arithmétiques : +, -, \*, /, \*\*, mod, rem

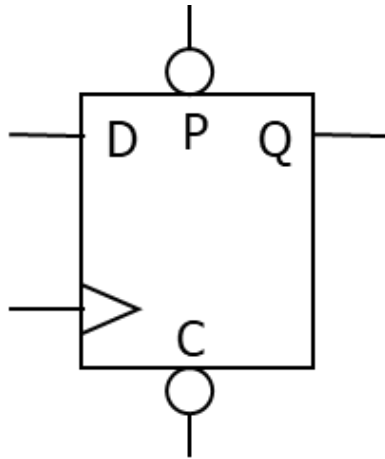
Autres Opérateurs : abs, &

# Séance 3

# Circuits séquentiels en VHDL

## Bascule D :

**Circuit synchrone : la sortie n'évolue que sur un front d'horloge**

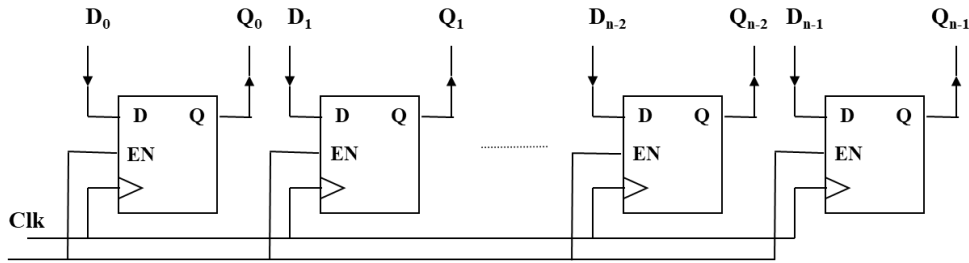


Avec entrée preset synchrone et  
clear asynchrone

```
entity dff is
port (CLK, P, C: in STD_LOGIC; -- commandes
      D:          in STD_LOGIC; -- entrée
      Q:          out STD_LOGIC); -- sortie
end dff;

architecture bhv of dff is
begin
  process (CLK, C)
  begin
    if C = '0' then
      Q <= '0';
    elsif CLK'EVENT AND CLK = '1' then
      if P = '0' then
        Q <= '1';
      else
        Q <= D;
      end if;
    end if;
  end process;
end bhv;
```

# Registres en VHDL



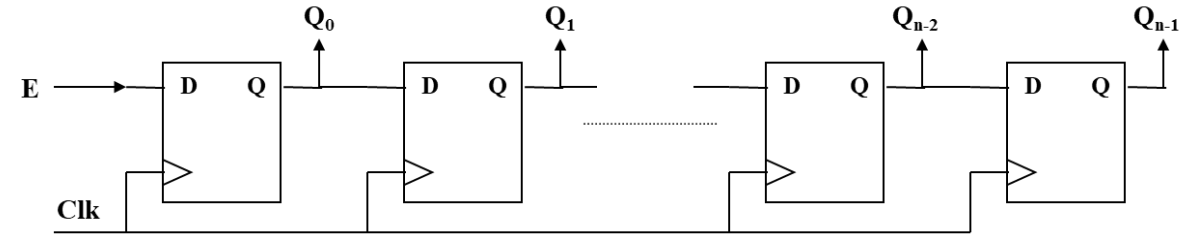
**Registre en VHDL :**

```
entity registre is
port (
    CLK, load : in std_logic;
    D : in std_logic_vector(15 downto 0);
    Q : out std_logic_vector(15 downto 0)
);
end registre;

architecture bhv of registre is
begin

    process (CLK)
    begin
        if CLK'EVENT AND CLK = '1' then
            if load = '1' then
                Q <= D;
            end if;
        end if;
    end process;

end bhv;
```



**Registre à décalage en VHDL :**

```
entity regdec is
port (
    CLK, E : in std_logic;
    Q : out std_logic_vector(15 downto 0)
);
end regdec;

architecture bhv of regdec is

    signal Qint : std_logic_vector(15 downto 0);

begin

    Q <= Qint;

    process (CLK)
    begin
        if CLK'EVENT AND CLK = '1' then
            Qint <= Qint(14 downto 0) & E;
        end if;
    end process;

end bhv;
```

# Compteur en VHDL

**Compteur  
modulo 7 en  
VHDL avec  
remise à zéro  
asynchrone :**

```
entity compteur_mod7 is
port (CLK, raz : in std_logic;
      cpt : out std_logic_vector(2 downto 0));
end compteur_mod7 ;

architecture bhv of compteur_mod7 is
  signal cpt_int : unsigned(2 downto 0);
begin

  cpt <= std_logic_vector(cpt_int);
  process (CLK,raz)
  begin
    if raz = '1' then
      cpt_int <= (others => '0'); -- met à 0 tous les bits
    elsif CLK'EVENT AND CLK = '1' then
      if cpt_int = to_unsigned(6,3) then
        cpt_int <= to_unsigned(0,3); -- transforme l'entier 6 en unsigned sur 3 bits
      else
        cpt_int <= cpt_int + 1 ;
      end if;
    end if;
  end process;

end bhv;
```

# Séance 4



# Machines à états finis en VHDL

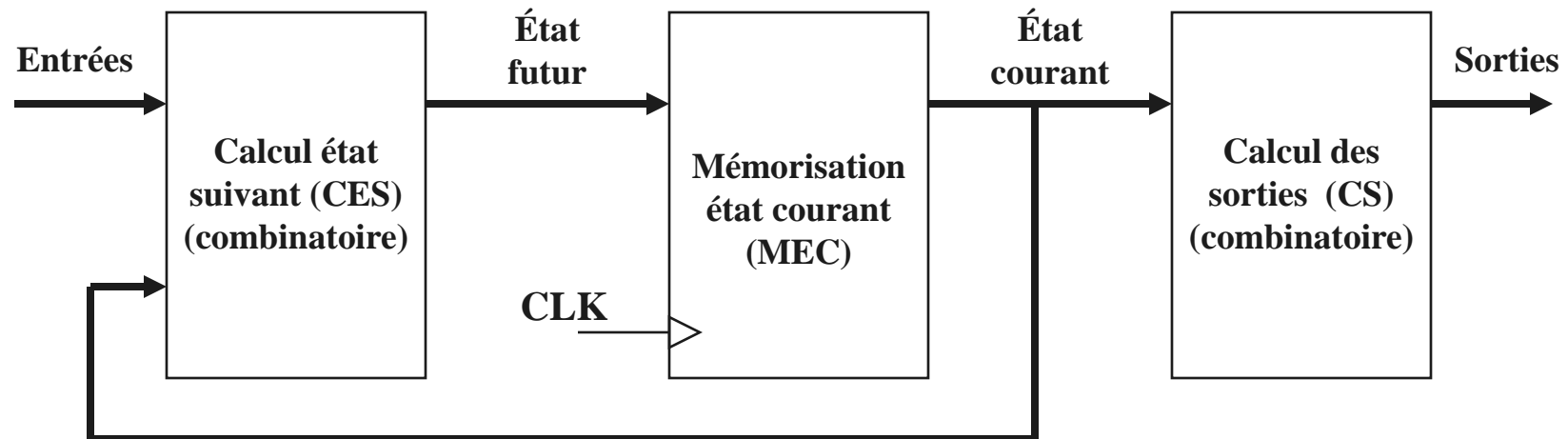
## Définition :

Système à base de bascules synchrones permettant de réaliser n'importe quel processus séquentiel (compteur, contrôle ...).

Les bascules permettent de mémoriser l'état courant du processus

Des parties combinatoires permettent d'en déduire l'état futur et la valeur des sorties

**Machine de Moore :** Les sorties ne dépendent que de l'état courant  
L'état futur dépend de l'état courant et des entrées



# Machines à états finis en VHDL

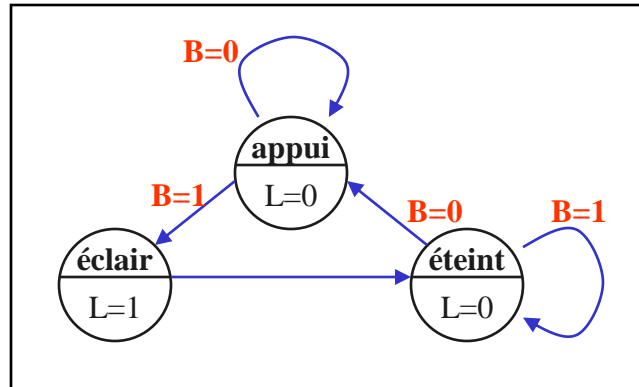
**Exemple :** fonction flash

Un appui sur B provoque un éclair pendant un coup d'horloge après relâchement de B

**Réalisation sur machine de Moore :**

- **Spécification :**
  - une entrée B qui vaut 0 quand on appuie, 1 au repos
  - une sortie L (=1 quand allumée)

- **Diagramme d'états :**



- **Codage des états :** détermine le nombre de bascules nécessaires

éteint  $\Leftrightarrow$  "00" ; appui  $\Leftrightarrow$  "01" ; éclair  $\Leftrightarrow$  "10"    2 bascules ( $Q_1Q_0$ )

# Machines à états finis en VHDL

## Machines de Moore : fonction flash Représentation VHDL :

```
entity flash is
port (CLK, RST, B: in std_logic;
      L: out std_logic);
end flash;

architecture fsm_moore of flash is
type état is (éteint,appui,éclair);
signal ecur, esuv: état;
begin
CES: process(ecur,B) -- combinatoire
begin
case ecur is
when éteint =>
if B = '0' then esuv <= appui; else
esuv <= éteint; end if;

when appui =>
if B = '1' then esuv <= éclair; else
esuv <= appui; end if;

when éclair => esuv <= éteint;
when others => esuv <= éteint;
end case;
end process CES;
```

## Utilisation de 3 process pour représenter les trois blocs (CES,MEC,CS)

```
MEC: process(RST,CLK)
begin
if RST = '1' then
ecur <= éteint;
elsif CLK'event AND CLK='1' then
ecur <= esuv;
end if;
end process MEC;

CS: process (ecur) -- combinatoire
begin
case ecur is
when éteint => L <= '0';
when appui => L <= '0';
when éclair => L <= '1';
when others => L <= '0';
end case;
end process CS;

end fsm_moore;
```