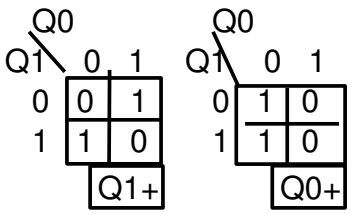
Les représentations graphiques du séquentiel

I) Le séquentiel simple (diagramme d'évolution) avec équations de récurrence

État présent		État futur	
Q1	Q0	$Q1^+$	$Q0^+$
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

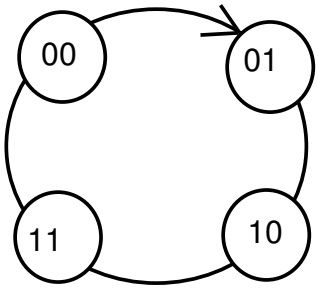


Équations de récurrence :

$Q_1^+ = Q_1$ €XOR˜ Q_0

$Q_0^+ = /Q_0$

```
ENTITY cmpt IS PORT (
clk: IN BIT;
q0,q1: INOUT BIT);
END cmpt;
ARCHITECTURE acmpt OF cmpt IS
BEGIN
  cmpt1 : PROCESS (clk) BEGIN
  IF (clk'EVENT AND clk='1') THEN
      q0 <= NOT q0;
      q1 <= q0 XOR q1;
  END IF;
  END PROCESS;
END acmpt;
```



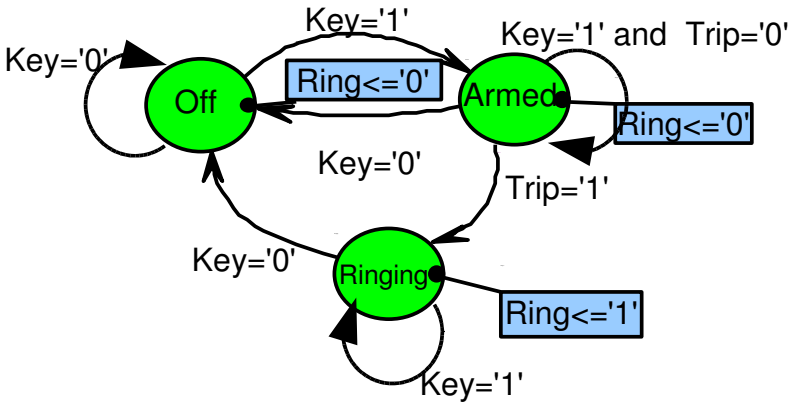
II) Le séquentiel simple (diagramme d'évolution) sans équations de récurrence

```
ENTITY demo IS PORT(
  clock : IN BIT;
  q : INOUT BIT_VECTOR(0 TO 1));
END demo;

ARCHITECTURE mydemo OF demo IS
     BEGIN
       PROCESS(clock) BEGIN
         IF clock'EVENT AND clock='1' THEN
           CASE q IS  --style case when
             WHEN "00" => q <="01";
             WHEN "01" => q <="10";
             WHEN "10" => q <="11";
             WHEN OTHERS => q <="00" ;
           END CASE;
         END IF;
       END PROCESS;
     END mydemo;
```

En cours : montrer comment on programme un compteur en VHDL

III) Le séquentiel plus complexe : laisser le compilateur faire le travail du codage

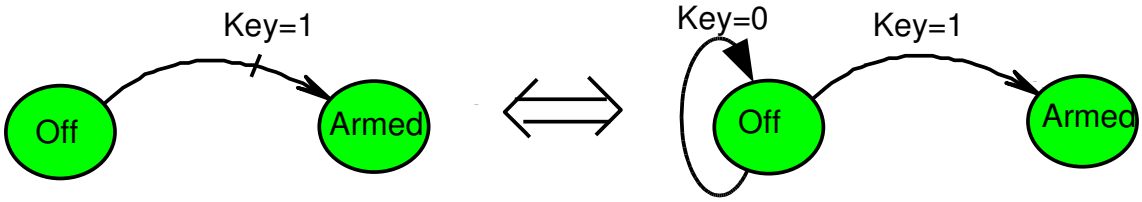


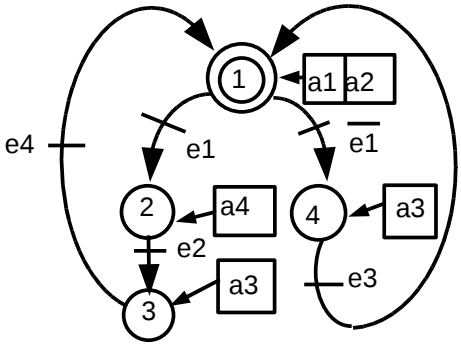

```
ENTITY Alarm IS
       PORT(
             clock,Key,Trip :IN BIT;
             Ring :OUT BIT
             );
     END Alarm;

     ARCHITECTURE ar  OF Alarm IS
       TYPE typetat IS (Armed, Off, Ringing);
       SIGNAL etat : typetat;
       BEGIN
         PROCESS (clock,etat) BEGIN  -- partie séquentielle
           IF Clock ='1' AND Clock'EVENT THEN
             CASE etat IS
               WHEN Off => IF key ='1' THEN etat <= Armed;
                           ELSE etat <= Off;
                         END IF;
               WHEN Armed => IF Key = '0' THEN
                             etat <= Off;
                           ELSIF Trip ='1' THEN
                               etat <= Ringing;
                             ELSE etat <= Armed;
                           END IF;
               WHEN Ringing => IF Key ='0' THEN
                                 etat <= Off;
                               ELSE etat <= Ringing;
                           END IF;
             END CASE;
           END IF;
           IF etat=Ringing THEN  -- partie combinatoire
             Ring<='1';
           ELSE Ring <='0';
           ENDIF
         END PROCESS;
     END ar;
```

En cours : montrer comment programmer ce réveil avec équations de récurrences en VHDL

IV) Les graphes d'états





AC1 = x3.e4+x4.e3
AC2 = x1.e1 AC3 = x2.e2
AC4 = x1./e1
D1 = e1+/e1=1
D2 = e2
D3 = e4
D4 = e3

$x1^+$ = x3.e4+x4.e3 + Init
$x2^+$ = (x1.e1+x2./e2)./Init
$x3^+$ = (x2.e2+x3./e4)./Init
$x4^+$ = (x1./e1+x4./e3)./Init
Équations de sorties
a1 = x1 a2 = x1
a3 = x3 + x4
a4 = x2

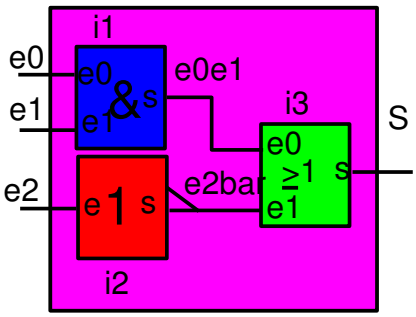
On supprime les oreilles de Mickey

```
-- sans initialisation
BEGIN
  PROCESS (clock) BEGIN
    IF clock'EVENT AND clock='1' THEN
      CASE etat IS
      WHEN Off => IF key ='1' THEN etat <= Armed;
                  ELSE etat <= Off;
                  END IF;
                  ....
      END CASE;
    END IF;
  END PROCESS;
  ....

-- avec initialisation synchrone
BEGIN
  PROCESS (clock) BEGIN
    IF clock'EVENT AND clock='1' THEN
      IF Init='1' THEN etat  <=Off; --initialisation synchrone
      ELSE
        CASE etat IS
        WHEN Off => IF key ='1' THEN etat <= Armed;
                    ELSE etat <= Off;
                    END IF;
                    ....
        END CASE;
      END IF
    END IF;
  END PROCESS;
```

V) Assembler des composants

On utilise un seul fichier avec des components et des port map.

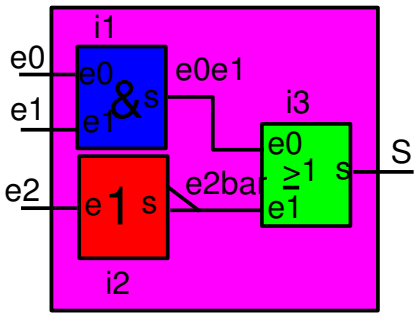



```
ENTITY Fct IS
PORT(e0,e1,e2 : IN BIT;
     s : OUT BIT);
END Fct;

ARCHITECTURE truc OF Fct IS
COMPONENT et
PORT(e0,e1 : IN BIT;
     s : OUT BIT);
END COMPONENT;
COMPONENT ou
PORT(e0,e1 : IN BIT;
     s : OUT BIT);
END COMPONENT;
COMPONENT inverseur
PORT(e : IN BIT;
     s : OUT BIT);
END COMPONENT;
SIGNAL e0e1,e2bar : BIT;
BEGIN
i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
i2:inverseur PORT MAP(e=>e2,s=>e2bar);
i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>s);
END truc;

ENTITY et IS
PORT(e0,e1 : IN BIT;
     s : OUT BIT);
END et;
ARCHITECTURE aet OF et IS
BEGIN
  s<=e0 AND e1;
END aet;
ENTITY ou IS
PORT(e0,e1 : IN BIT;
     s : OUT BIT);
END ou;
ARCHITECTURE aou OF ou IS
BEGIN
  s<=e0 OR e1;
END aou;
ENTITY inverseur IS
PORT(e : IN BIT;
     s : OUT BIT);
END inverseur;
ARCHITECTURE ainv OF inverseur IS
BEGIN
  s<= NOT e;
END ainv;
```

On utilise deux fichiers ou plus avec des components et des port map. Le fichier décrivant les composants est appelé librairie.



top.vhd

```
LIBRARY portes; --portes.vhd
USE work.mesportes.ALL;

ENTITY Fct IS
PORT(e0,e1,e2 : IN BIT;
      s : OUT BIT);
END Fct;

ARCHITECTURE truc OF Fct IS
SIGNAL e0e1,e2bar : BIT;
BEGIN
i1:et PORT MAP(e0=>e0,e1=>e1,s=>e0e1);
i2:inverseur PORT MAP(e=>e2,s=>e2bar);
i3:ou PORT MAP(e0=>e0e1,e1=>e2bar,s=>s);
END truc;
```

portes.vhd

```
PACKAGE mesportes IS
COMPONENT et
PORT(e0,e1 : IN BIT;
      s : OUT BIT);
END COMPONENT;
COMPONENT ou
PORT(e0,e1 : IN BIT;
      s : OUT BIT);
END COMPONENT;
COMPONENT inverseur
PORT(e : IN BIT;
      s : OUT BIT);
END COMPONENT;
end mesportes;
ENTITY et IS
PORT(e0,e1 : IN BIT;
      s : OUT BIT);
END et;
ARCHITECTURE aet OF et IS
BEGIN
  s<=e0 AND e1;
END aet;
ENTITY ou IS
PORT(e0,e1 : IN BIT;
      s : OUT BIT);
END ou;
ARCHITECTURE aou OF ou IS
BEGIN
  s<=e0 OR e1;
END aou;
ENTITY inverseur IS
PORT(e : IN BIT;
      s : OUT BIT);
END inverseur;
ARCHITECTURE ainv OF inverseur IS
BEGIN
  s<= NOT e;
END ainv;
```

VI) Le codage des états

La programmation des états nécessite une déclaration symbolique comme on l'a vu dans le cas du réveil :

```
TYPE typetat IS (Armed, Off, Ringing); -- dans architecture
SIGNAL etat : typetat;
```

Quand la synthèse sera demandée plusieurs solutions peuvent se présenter suivant le codage des états. Une telle déclaration débouchera sur un codage Armed=00, Off=01 et Ringing=10.

On peut modifier ce codage à l'aide de deux attributs différents : enum_encoding et state_encoding. Enum_encoding est normalisé par le standard IEEE 1076.6.

```
type state is (s0,s1,s2,s3);
attribute enum_encoding of state:type is "00 01 10 11";
```

La directive state_encoding spécifie la nature du code interne pour les valeurs d'un type énuméré.

```
attribute state_encoding of type-name:type is value;
```

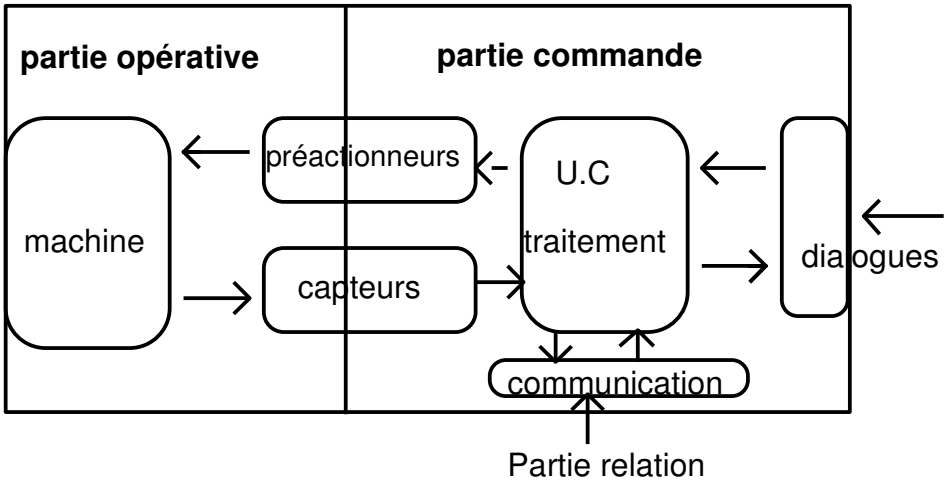
Les valeurs légales de la directive state_encoding sont sequential, one_hot_zero, one_hot_one, and gray.

sequential: on code en binaire au fur et à mesure de l'énumération avec autant de bits que nécessaire.
one_hot_zero: on code la première valeur par zéro, puis le reste en utilisant à chaque fois un seul un : N états nécessiteront donc N-1 bits.
one_hot_one: idem à one_hot_zero sauf que l'on n'utilise pas le code zéro. N états nécessiteront donc N bits.
Gray: les états suivent un code GRAY.

Exemples:

```
type state is (s0,s1,s2,s3);
attribute state_encoding of state:type
is one_hot_zero;
type s is (s0,s1,s2,s3);
attribute state_encoding of s:type is gray;
```

VII) Le GRAFCET NF EN 60848 de AOUT 2002.



Les règles de syntaxe