Le langage VHDL

F. Senny Bureau R140

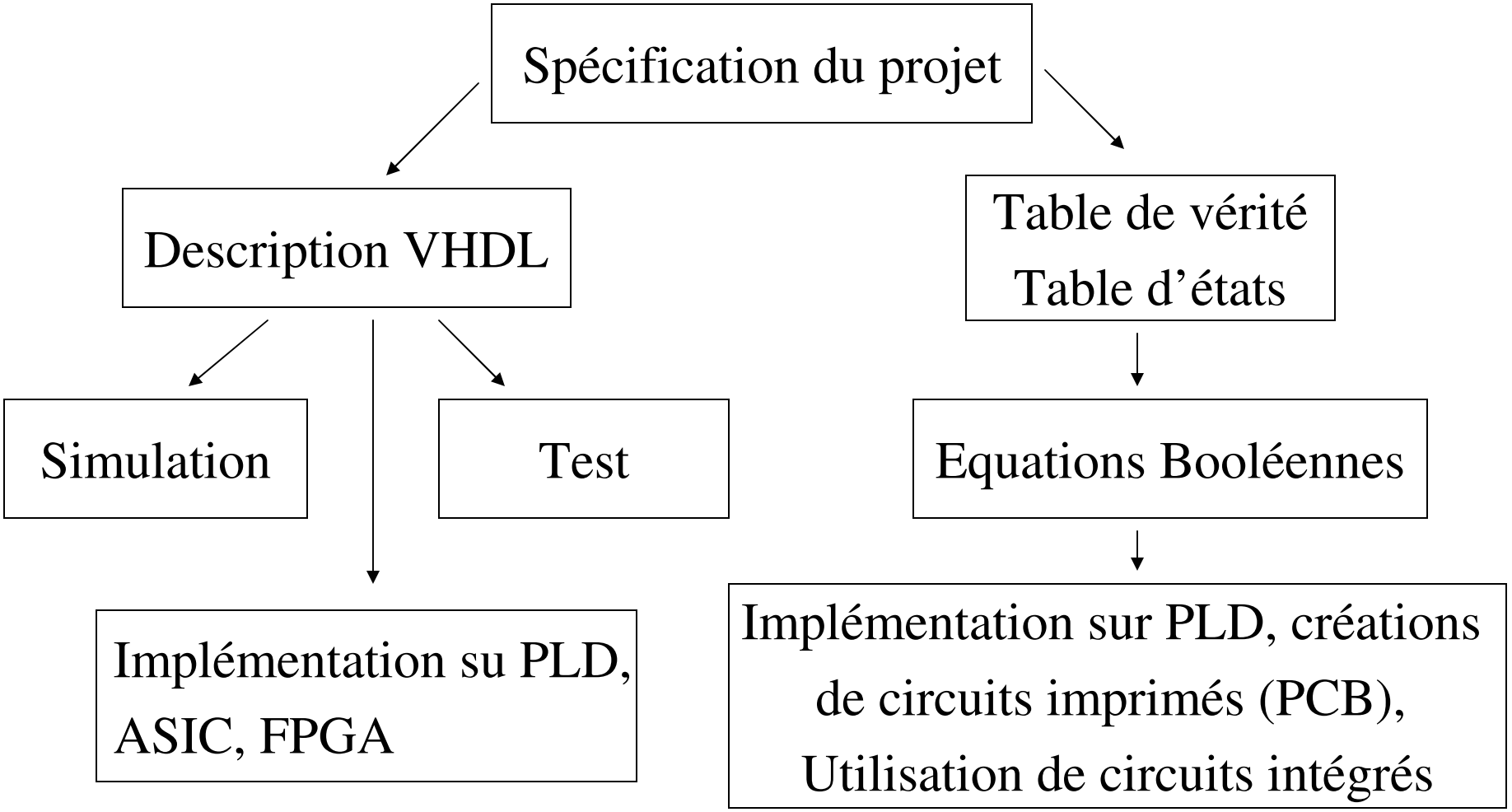
Tel: 04/366.26.80

Email: senny@montefiore.ulg.ac.be

Le VHDL

- VHDL:
  - VHSIC Hardware Description Langage
  - Existe depuis fin des années 80
  - Permet de créer des design de toutes tailles
- Caractéristiques :
  - Standard (indépendant du logiciel $\Rightarrow$ échange facile)
  - Méthodologies de conception diverses
  - Outil complet (design, simulation, synthèse)
  - Haut niveau d'abstraction (indépendant de la technologie)

Réalisation d'un projet



Généralités

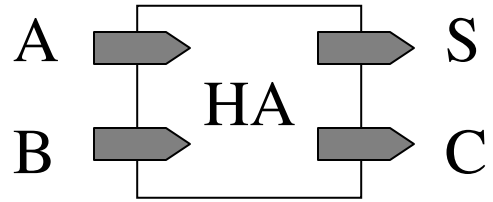
Paire *entité/architecture*

- Tout programme VHDL contient au moins une paire indissociable *entité/architecture*.

  Exemple : le half adder

```
-- ceci est un commentaire
-- déclaration des librairies
library ieee;
use ieee.std_logic_1164.all;
-- déclaration de l'entité
entity HA is
port(A,B : in std_logic; S,R : out std_logic);
end HA;
-- architecture de type flot de données
architecture arch_HA_flot of HA is
begin
   S <= A xor B;
   C <= A and B;
end arch_HA_flot;
```

Entité = vue externe



Architecture = vue interne

$S = A \oplus B$

$C = A \bullet B$

Librairies

- Tous programmes VHDL nécessitent la déclaration d'une ou de plusieurs bibliothèques qui incluent la définition des types de signaux et des opérations que l'on peut effectuer entre ceux-ci.

- Typiquement, l'en-tête est celle-ci :

```
   library ieee;
   use ieee.std_logic_1164.all; -- définition du type bit, vecteur de bit,etc
   use ieee.numeric_std.all; -- operations signées ou non signées
   use ieee.std_logic_arith.all; -- operations signées ou non signées
-- use work.std_arith.all; -- similaire à numeric_std=operations entre vecteurs
                           -- = operations entre vecteurs de std_logic avec
                           -- (vecteurs de) std_logic, (vecteurs d')entier(s)
```

Entité, entrées/sorties (I/O)

- L'entité définit le nom de la description VHDL ainsi que les I/O. La syntaxe se formule comme suit

```
entity NOM_DE_L_ENTITE is
     port(déclaration des I/O);
end NOM_DE_L_ENTITE;
```

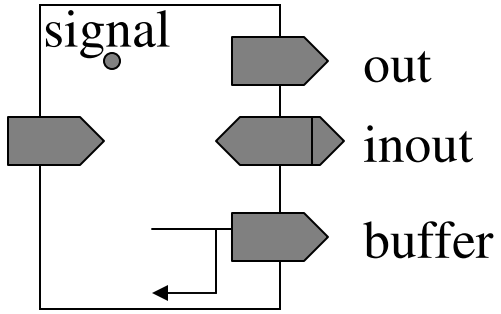
- L'instruction `port` définit les I/O par la syntaxe

```
NOM_DU_SIGNAL : MODE TYPE;
```

  - `NOM_DU_SIGNAL` est composé de caractère commençant par une lettre.
  - Le `MODE` du signal peut-être
    - *`in`*, un port d'entrée
    - *`out`*, un port de sortie

Entité, entrées/sorties (I/O)

- *inout*, un port bidirectionnel
- *signal + out (= buffer)*, un port de sortie mais aussi utilisé en interne dans la description
- *signal*, un point interne
- *variable*, **aucun sens physique**



– Les TYPE fort utilisés pour un signal sont

- *std_logic* : '0' ,'1', 'Z' , 'L, 'H', '-',…
- *std_logic_vector (X downto 0)* ou *(0 to X)* : « 0011»
- *integer* → compteur, indice de boucle,…
- Remarque: on peut définir un nouveau type. Par exemple,

```
type etat4 is (e0,e1,e2,e3);
```

Architecture

- Elle décrit le fonctionnement du système, les relations entre les entrées et les sorties.
- Si un système comprend plusieurs sous-systèmes, chacun sera décrit par une paire entité/architecture.
- Le fonctionnement peut être combinatoire et/ou séquentielle.
- On recense trois grands formalismes d'architecture
  - Flot de données
  - Structurel
  - Comportemental

Architecture

COMPORTEMENTAL

- Description condition-nelle
- Nécessite un `process`
- Circuits de toutes tailles

FLOT DE DONNEES

- Description des équations logiques
- Petits circuits

STRUCTUREL

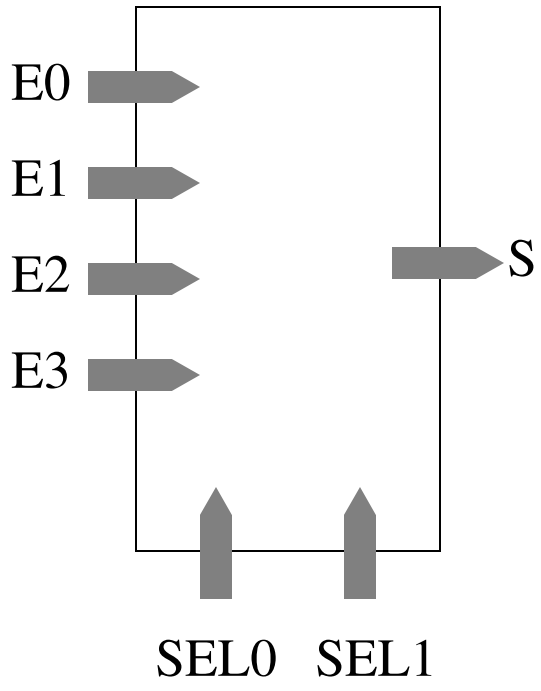
- Description du circuit sous forme de boîtes noires interconnectées
- Circuits de taille moyenne à grande

VHDL en Logique Combinatoire

Flot de données

- Description fonctionnelle, des équations booléennes
- Exemple: MUX 4-1

```
entity MUX is port(
      E0,E1,E2,E3,SEL0,SEL1 : in std_logic;
      S : out std_logic);
end MUX;
architecture FLOT_MUX of MUX is
begin
      S <= ((not SEL0) and (not SEL1) and E0) or
            (SEL0 and (not SEL1) and E1) or
            ((not SEL0) and SEL1 and E2) or
            (SEL0 and SEL1 and E3);
end FLOT_MUX;
```



Flot de données

- Usage de forme conditionnelles lors des affectations dans le cas du MUX 4-1 :

when expression **else** — **with** expression **select**

```
entity MUX is port(
   E0,E1,E2,E3,SEL0,SEL1 : in std_logic;
   S : out std_logic);
end MUX;
architecture FLOT_MUX of MUX is
begin
```

```
S <= E0 when (SEL0='0' and
              SEL1='0') else
     E1 when (SEL0='1' and
              SEL1='0') else
     ...              else
     '-';
```

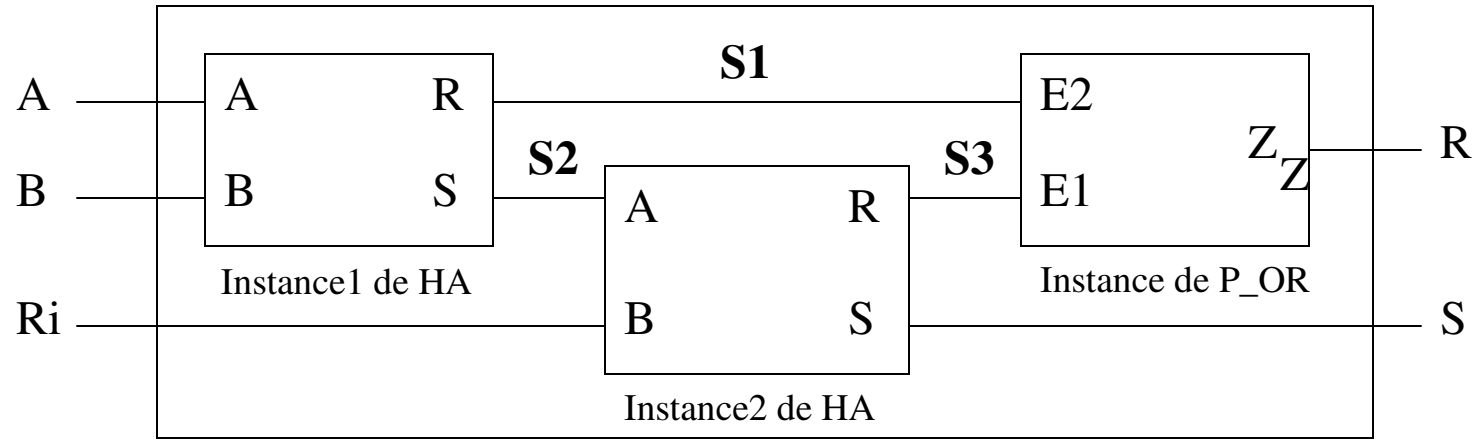
```
with SEL1&SEL0  select
  S <= E0 when «00»,
       E1 when «01»,
       E2 when «10»,
       E3 when «11»,
       '-' when others;
```

```
end FLOT_MUX;
```

Structurel

- Il s'agit de l'assemblage de composants déjà décrits
- On utilise des *instances* de ces boîtes noires.

  Instancier = placer le composant dans son environnement.
- Exemple : Full adder = association de 2 HA et 1 OR



Structurel

```
-- entité HA
entity HA is port(
      A,B : in std_logic;
      R,S : out std_logic);
end HA;
architecture FLOT of HA is
begin
      S <= A xor B;
      C <= A and B;
end FLOT;
-- entité porte OR
entity P_OR is port(
      E1,E2 : in std_logic;
      Z: out std_logic);
end HA;
architecture FLOT of P_OR is
begin
      Z <= E1 or E2;
end FLOT;
```

```
-- entité full adder
entity FA is port(
A,B,Ri : in std_logic;
          R,S : out std_logic);
end FA;
architecture STRCT_FA of FA is
          component HA
          port (A,B : in std_logic;
                    R,S : out std_logic);
          end component;
          component P_OR port(E1,E2 : in
          std_logic; Z: out std_logic);
          end component;
signal S1,S2,S3: std_logic;
begin
          instance1 : HA port map(A,B,S1,S2);
          instance2 : HA port map(S2,Ri,S3,S);
          instance : P_OR port map(S3,S1,R);
end STRCT_FA;
```

Comportemental

- Similaire à un programme informatique, emploi d'instructions séquentielles conditionnelles
- Fait appel à un `process` :

```
[NOM_DU_PROCESS :] process(liste de sensibilité)
begin
     -- instructions séquentielles !!
end process [NOM_DU_PROCESS];
```

  - la liste de sensibilité est la liste des variables auxquelles le process est sensible : un changement d'une de ces variables «réexécute» (!☠!) le process.
  - Les instructions sont exécutées **séquentiellement !!**
  - Les **changements** d'états des signaux sont pris en compte à la **fin** du process.

IF … THEN … ELSE

- Syntaxe :

```
if cond_bool_1 then
      instructions_1;
elsif cond_bool_2 then
       instructions_2;
[else instructions_3;]
end if;
```

- Exemple : MUX

```
entity MUX is port(
      E0,E1,E2,E3 : in std_logic;
      SEL0,SEL1 : in std_logic;
      S : out std_logic;);
end MUX;
architecture CMP1_MUX of MUX is
begin
process(E0,E1,E2,E3,SEL0,SEL1)
      begin
      if (SEL0='0' and SEL1='0') then
                  S <= E0;
      elsif (SEL0='1' and
                  SEL1='0') then
                  S <= E1;
      elsif (SEL0='0' and
                  SEL1='1') then
                  S <= E2;

      elsif (SEL0='0' and
                  SEL1='1') then
                  S <= E3;
      else S <= '-';
      end if;
end process;
end CMP1_MUX;
```

CASE …IS WHEN …

- Syntaxe :

```
case expression is
      when valeur_1 =>
        instructions_1;
      when valeur_2 =>
        instructions_2;
      [when others =>
        instructions_3;]
end case;
```

- Exemple : MUX

```
entity MUX is port(
      E0,E1,E2,E3 : in std_logic;
      SEL : in
         std_logic_vector(1 downto 0);
      S : out std_logic;);
end MUX;
architecture CMP2_MUX of MUX is
begin
   process -- pas de liste MAIS
   begin
      case SEL is
         when «00» => S <= E0;
         when «01» => S <= E1;
         when «10» => S <= E2;
         when «11» => S <= E3;
         when others => null;
      end case;

   wait on E0,E1,E2,E3,SEL; -- !!
   end process;
end CMP2_MUX;
```

Les boucles

- Boucle simple :

```
[label :] loop
        instructions;
end loop[ label];
```

- Boucle while :

```
[label :] while  expression  loop
        instructions;
end loop[ label];
```

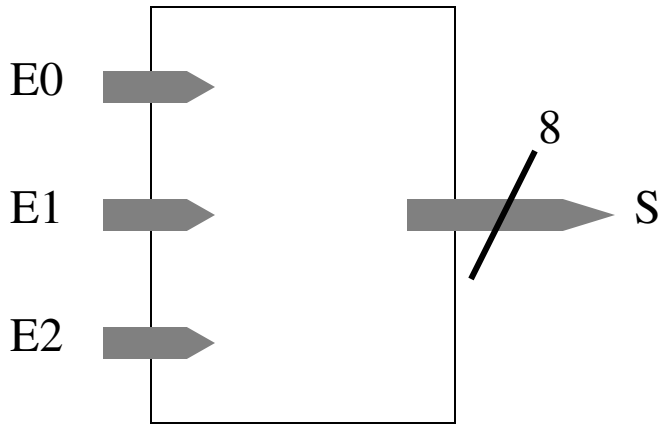
- Boucle for :

```
[label :] for  var  in  exp1 to exp2 loop
        instructions;
end loop[ label];
```

Les boucles

- Exemple : le décodeur 3-8

```
entity DEC3_8 is port(
  E0,E1,E2 : in std_logic;
  S : out std_logic_vector(7 downto 0));
end DEC3_8;
architecture CMP_DEC of DEC3_8 is
begin
  process(E0,E1,E2)
  variable N := integer;
  begin
      N:=0;
      if E0='1' then N:=N+1;end if;
      if E1='1' then N:=N+2;end if;
      if E2='1' then N:=N+4;end if;
      S <= «00000000»;
      for I in 0 to 7 loop
          if (I=N) then
             S(I) <= '1';
          end if;
      end loop;
  end process;
end CMP_DEC;
```



Fonction et procedure

- Fonction. Exemple : le maximum de deux nombres

```
max2 := MAX(N1,N2);
```

  - exécute une suite d'instructions séquentielles et retourne *une et une seule* valeur.
  - L'affectation n'est pas contenue dans la fonction ⇒ l'affectation successive de deux fonctions y1:= f1(x1) et y2:= f2(x2) ont lieu en même temps (hors process !!).

```
function MAX(A,B: integer) return integer is
begin
  if  A > B then
    return A;
  else
    return B;
  end if;
end MAX;
```

Fonction et procedure

- Procédure. Exemple : le minimum et le maximum de trois nombres `MIN_MAX(N3,N4,N5,MINI,MAXI);`
  - exécute une suite d'instructions séquentielles et retourne *une ou plusieurs* valeur(s).
  - L'affectation est contenue dans la procédure

```
procedure MIN_MAX(A,B,C: in integer
                  MINI,MAXI : out integer) is
variable AUX1, AUX2 : integer;
begin
  if A > B then
    AUX1 := A;AUX2 := B;
  else
    AUX1 := B;AUX2 := A;
  end if;
  if C > AUX1 then
    AUX1 := C;
  elsif C < AUX2 then
    AUX2 := C;
  end if;
  MIN := AUX2;
  MAX := AUX1;
end MIN_MAX;
```

Tableau des Opérations

Opérateurs logiques	and, or, nand, nor, xor, xnor, not
Opérateurs additifs	+, -, &
Opérateurs multiplicatifs	*, /, mod, rem
Opérateurs divers	Abs, **
Opérateurs d'assignation	<=, :=
Opérateurs d'association	=>
Opérateurs de décalage	sll, srl, sla, sra, rol, ror

Remarques importantes

Signal, variable et constante

- Le type `signal` est utilisé pour des signaux intermédiaires. A définir dans l'en-tête de l'architecture.

```
signal NOM_DU_SIGNAL : mode type; -- type = std_logic ou std_logic_vector
```

- Une `variable` est déclarée et uniquement valide *dans un process* (avant le `begin` du process). Affectation immédiate lors de :=

```
variable NOM_DE_VARIABLE : type [:= exp]; -- type = integer(souvent),...
```

- Une `constante` se déclare dans l'en-tête de l'architecture.

```
constant NOM_DE_CONSTANTE : type := exp; -- type = integer, boolean,...
```

Ordre des affectations

- *Hors d'un process*, les affectations sont *concurrentes* et *immédiates*. Ainsi, deux *process* peuvent être exécutés en même temps.
- *Dans un process*, les instructions sont exécutées *séquentiellement* et les *affectations* ont lieu à la *fin du process*.

```
  signal  S : std_logic;
begin
process(A)
  begin
    S <= A;
    B <= S;
end process;
```

≠

```
  signal  S : std_logic;
begin
process(A,S)
  begin
    S <= A;
    B <= S;
end process;
```

Attention aux *else*

- La présence ou l'absence de la condition «dans les autres cas» (if..then…else, case…is…when…when others,…) conditionnera le circuit synthétisé.

```
begin
process(EN,D)
  begin
    if (EN = '1') then
      Q <= D;
    end if;
end process;
```

≠

```
begin
process(EN,D)
  begin
    if (EN = '1') then
      Q <= D;
    else Q <= 0;
    end if;
end process;
```

Logique Séquentielle

Fonctions séquentielles

- Attributs pour un signal *CLK*
  - `CLK'event` : fonction de type booléenne, vraie si un changement et apparu sur CLK.
  - `CLK'stable(T)` : fonction de type booléenne, vraie si CLK n'a eu aucun changement pendant le temps T.
  - `CLK'transaction` : signal de type bit qui change de valeur pour tout calcul de CLK.
- Détection d'un flanc montant (resp. descendant) de CLK :

```
CLK'event and CLK='1' (resp. CLK='0')
```

Modélisation d'une bascule D

- Descritpion d'un FF-D déclenchant sur flanc montant,

avec reset asynchrone

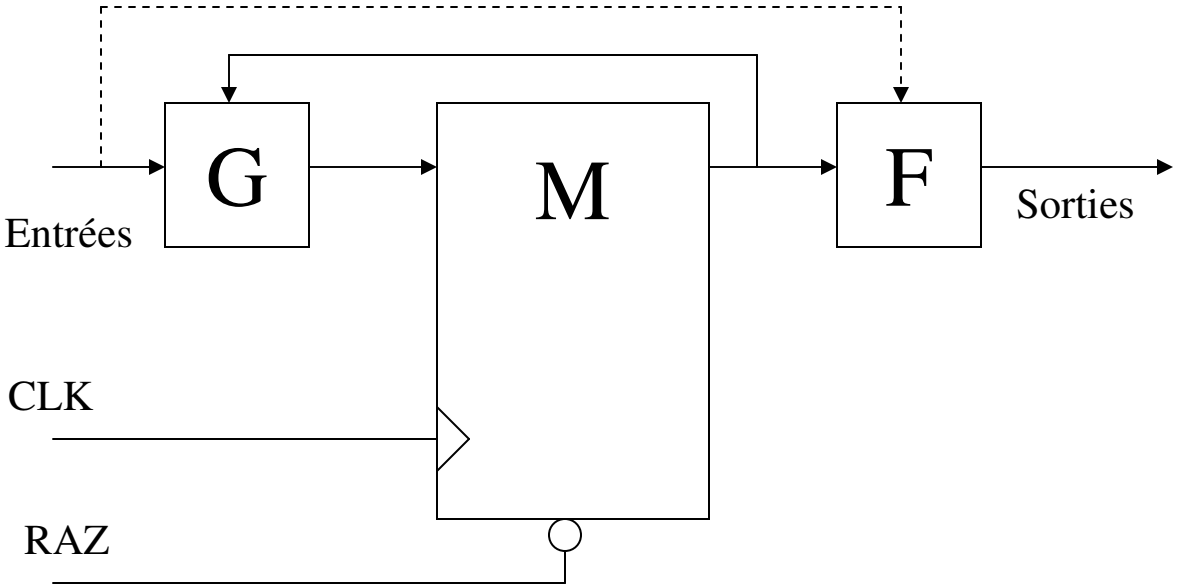
```
entity FFDRA is port(
  D,RAZ,CLK : in bit; Q : out bit);
end FFDRA;
architecture ARCH_FFRA of FFRA is
begin
  process(CLK,RAZ) begin
    if RAZ = '0' then Q <= 0;
    elsif (CLK'event and CLK = '1')
      then Q <= D;
    end if;
  end process;
end ARCH_FFRA;
```

avec reset synchrone

```
entity FFDRS is port(
  D,RAZ,CLK : in bit; Q : out bit);
end FFDRS;
architecture ARCH_FFRS of FFRS is
begin
  process(CLK,RAZ) begin
    if (CLK'event and CLK = '1') then
      if RAZ = '1' then Q <= 0;
      else Q <= D;
      end if;
    end if;
  end process;
end ARCH_FFRS;
```

Machine d'états

- Conception possible pour une machine de Moore ou de Mealy
- Schéma classique d'une machine d'états :



- G → état suivant
- F → sorties
- M → bloc mémoire

Solutions envisagées :

1 process (G + M + F)

2 process (G + M) + F

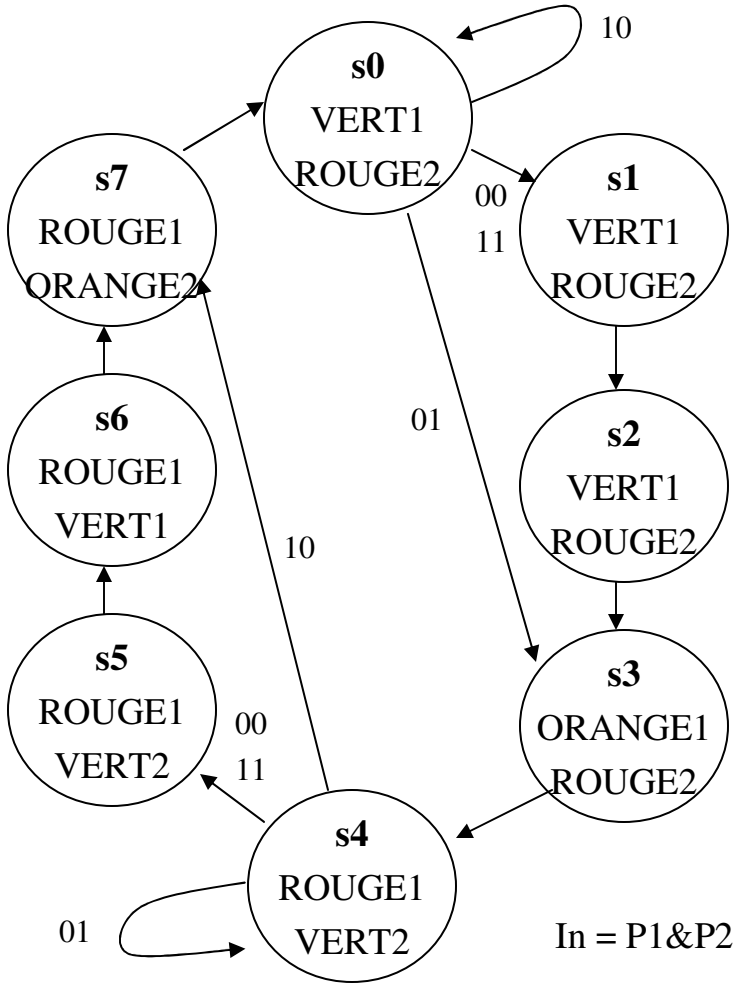
3 process G + M + F

Exemple de machine d'états

- Gestion d'un feu rouge :

  P1 et P2 : capteurs sur voie 1 et 2

  Si une voiture se présente sur la voie 2, P2 = 1 et le feu 1 devient O → R et le feu 2 R → V.
  Si P1 = P2, les feux suivent la séquence
  V ↔ V ↔ V ↔ O ↔ R ↔ R ↔ R.



2 process (G + M) + F

```
entity FEUX1 is port(
  RAZ, CLK : in std_logic;
  V1,R1,O1,V2,R2,O2 : out std_logic);
end FEUX1;
architecture ARCH_FEUX1 of FEUX1 is
type ETAT8 is (s0,s1,s2,...,s7);
signal ETAT : ETAT8:=s0;
begin
  CALCUL_ETAT : process(CLK,RAZ)
  begin
    if RAZ='0' then
      ETAT <= s0;
    elsif (CLK='1' and CLK'event) then
      case ETAT is
        when s0 =>
          case P1&P2 is
            when «10» => ETAT <= s0;
            when «01» => ETAT <= s3;
            when others => ETAT <= s1;
          end case;
        when s1 => ETAT <= s2;
        when s2 => ETAT <= s3;
        when s3 => ETAT <= s4;
        when s4 =>
          case P1&P2 is
            when «01» => ETAT <= s4;
            when «10» => ETAT <= s7;
            when others => ETAT <= s5;
          end case;
        when s5 => ETAT <= s6;
        when s6 => ETAT <= s7;
        when s7 => ETAT <= s0;
      end case;
    end if;
  end process CALCUL_ETAT;
  -- suite à la page suivante
```

2 process (G + M) + F

```
-- suite
CALCUL_SORTIES : process(ETAT)
begin
  case ETAT is
    when s0 =>
      V1<='1';R1<='0';O1<='0';
      V2<='0';R2<='1';O2<='0';
    when s1 =>
      V1<='1';R1<='0';O1<='0';
      V2<='0';R2<='1';O2<='0';
    when s2 =>
      V1<='1';R1<='0';O1<='0';
      V2<='0';R2<='1';O2<='0';
    when s3 =>
      V1<='0';R1<='0';O1<='1';
      V2<='0';R2<='1';O2<='0';
    when s4 =>
      V1<='0';R1<='1';O1<='0';
      V2<='1';R2<='0';O2<='0';
    when s5 =>
      V1<='0';R1<='1';O1<='0';
      V2<='1';R2<='0';O2<='0';
    when s6 =>
      V1<='0';R1<='1';O1<='0';
      V2<='1';R2<='0';O2<='0';
    when s7 =>
      V1<='0';R1<='1';O1<='0';
      V2<='0';R2<='0';O2<='1';
  end case;
 end process CALCUL_SORTIES;
end ARCH_FEUX1;
```

2 process (G + F) + M

```
entity FEUX2 is port(
  RAZ, CLK : in std_logic;
  V1,R1,O1,V2,R2,O2 : out std_logic);
end FEUX2;
architecture ARCH_FEUX2 of FEUX2 is
type ETAT8 is (s0,s1,s2,...,s7);
signal ETAT_PRES,ETAT_SUIV : ETAT8;
begin
 SYNCHRO : process(CLK,RAZ)
  begin
    if RAZ='0' then
      ETAT_PRES <= s0;
    elsif (CLK='1' and CLK'event) then
      ETAT_PRES <= ETAT_SUIV;
    end if;
  end process SYNCHRO;
  FCT_G_F process(ETAT_PRES,P1,P2)
  begin
    case ETAT_PRES is
      when s0 =>
        V1<='1';R1<='0';O1<='0';
        V2<='0';R2<='1';O2<='0';
        if (P1 and not P2) then
          ETAT_SUIV <= s0;
        elsif (not P1 and P2) then
          ETAT_SUIV <= s3;
        else ETAT_SUIVANT <= s1;
      when s1 =>
        V1<='1';R1<='0';O1<='0';
        V2<='0';R2<='1';O2<='0';
        ETAT_SUIV <= s2;
      when s2 =>
      ...
    end case;
  end process FCT_G_F;
end ARCH_FEUX2;
```

3 process G + F + M

```
entity FEUX3 is port(
  RAZ, CLK : in std_logic;
  V1,R1,O1,V2,R2,O2 : out std_logic);
end FEUX3;
architecture ARCH_FEUX3 of FEUX3 is
type ETAT8 is range 0 to 7;
signal ETAT_PRES,ETAT_SUIV : ETAT8;
begin
 SYNCHRO : process(CLK,RAZ)
  begin
    if RAZ='0' then
      ETAT_PRES <= 0;
    elsif (CLK='1' and CLK'event) then
      ETAT_PRES <= ETAT_SUIV;
    end if;
  end process SYNCHRO;
  FCT_G process(ETAT_PRES,P1,P2)
  begin
    case ETAT_PRES is
      when 0 =>
        if (P1 and not P2) then
          ETAT_SUIV <= 0;
        elsif (not P1 and P2) then
          ETAT_SUIV <= 3;
        else ETAT_SUIVANT <= 1;
      when 1 =>
        ETAT_SUIV <= 2;
      when 2 =>
      ...
      when 7 =>
        ETAT_SUIV <= 0;
    end case;
  end process FCT_G;
  FCT_F : process(ETAT_PRES)
  begin
    case ETAT_PRES is
```

3 process G + F + M

```
      when 0 =>
        V1<='1';R1<='0';O1<='0';
        V2<='0';R2<='1';O2<='0';
      when 1 =>
        V1<='1';R1<='0';O1<='0';
        V2<='0';R2<='1';O2<='0';
      when 2 =>
        V1<='1';R1<='0';O1<='0';
        V2<='0';R2<='1';O2<='0';
      ...
      when 7 =>
        V1<='1';R1<='1';O1<='0';
        V2<='0';R2<='0';O2<='1';
    end case;
  end process FCT_F;
end ARCH_FEUX3;
```

Annexe 1 : liste des types

Types		Prédéfinis	Utilisateurs

SCALAIRE	énuméré	`bit,std_logic`
		`boolean`
		`character`
		`deverity_level`
			COULEUR

	entier	`integer`	INDEX
	flottant	`real`	COSINUS
	physique	`time`	CAPACITE
COMPOSITE	array	`bit_vector` `string`	
	record		
access			
file			

Annexe 1 : exemples

```
type bit is ('0','1');
type boolean is (false,true);
type character is ('A','B',...,'a',...,'0',...,'*',...);
type severity_level is (note,warning,error,failure);
type integer is range -2 147 483 648 to 2 147 483 648;
type real is range -16#0,7FFFFF8#E+32 to 16#0,7FFFFF8#E+32;
type time is range -9_223_372_036_854_775_808 to
   9_223_372_036_854_775_808;
units : fs; ps=1000 fs; ns=1000 ps;...;min=60 sec;hr=60 min;
type bit_vector is array (natural range <>) of bit;
type string is array (natural range <>) of character;
type COULEUR is (R,V,O);
type INDEX is range 0 to 100;
```

Annexe 1 : exemples

```
type ANNEE is array (0 to 3) of integer;
```

Indice	0	1	2	3
	2	0	0	3

```
type TABLE is array (0 to 7,1 downto 0) of std_logic;
constant TV : TABLE :=(('0','0'),('1','0'),...,('1','1'));
```

E2	E1	E0		S1	S0
0	0	0		0	0
0	0	1		1	0
0	1	0		1	1
0	1	1		0	0
1	0	0		1	1
1	0	1		0	1
1	1	0		0	0
1	1	1		1	1

≡

Indice 2 →	1	0
Indice 1	S1	S0
0	0	0
1	1	0
2	1	1
3	0	0
4	1	1
5	0	1
6	0	0
7	1	1

Annexe 2 : Attributions des pins

- L'attribution est déclarée dans l'entité.
- Il n'est pas nécessaire d'attribuer tous les signaux déclarés dans `port` aux broches du composant PLD.

```
entity NOM_ENTITE is port(
   CLK, RST,DATA : in std_logic;
   S : buffer std_logic_vector(4 downto 0));

attribute pin_numbers of NOM_ENTITE : entity is
   «  S(4):23 S(3):22 S(2):21 S(1):20 S(0):19 »;
end NOM_ENTITE;
```

Annexe 3 : Package et librairies

- Package : ensemble de déclarations, fonctions et sou-programmes souvent utilisés par plusieurs personnes.
- 2 packages prédéfinis : TEXTIO et STANDARD
- D'autres packages peuvent être définis par l'utilisateur
- Les packages sont stockés dans des librairies
  - `work` (accès en lecture et écriture)
  - Librairie de ressources (accès en lecture)

```
library  lib; -- déclaration de la librairie LIB
use lib.elem; -- accéder à l'élément ELEM de LIB
use lib.pack.all; -- accéder à TOUS les éléments du package PACK de LIB
```

- Les librairies `work` et `std` ne doivent pas être déclarées

Annexe 3 : Package et librairies

- Exemple du package PACK :

```
entity ADDI is port(
  A,B,Ri : in bit;
  S,R : out bit);
end ADDI;
package  PACK is
 component DEMI_ADDI
 port(A,B : in bit; S,R : out bit);
 end component;
 component P_OR
 port(E1,E2 : in bit; Z : out bit);
 end component;
 signal S1,S2,S3 : bit;
end PACK;
```

```
use work.PACK.all;
architecture STRCT_ADDI of ADDI is
-- déclaration des composants et
-- des signaux
-- spécification de configuration
for all : DEMI_ADDI use entity
work.DEMI_ADD(ARCH) port map (A,B,S,R);
begin
  I1:DEMI_ADDI port map (A,B,S1,S2);
  I2:DEMI_ADDI port map (S1,Ri,S,S3);
  I: P_OR port map (S3,S2,R);
end STRCT_ADDI;
```

Annexe 4 : mots réservés

abs	disconnect	label	package	then
acces	downto	library	port	to
after		linkage	procedure	transport
alias	else	loop	process	type
all	elsif			
and	end	map	range	units
architecture	entity		record	until
array	exit	nand	register	use
assert		new	rem	
attribute	file	next	report	variable
	for	nor	return	
begin	function	not		wait
block		null	select	when
body	generate		severity	with
buffer	generic	of	signal	while
bus	guarded	on	subtype	with
		open		
case	if	or		
component	in	others		
configuration	inout	out		
constant	is			

Références utiles

[1] Warp, référence manual (CYPRESS)

[2] VHDL for Programmable Logic (CYPRESS)

[3] VHDL, introduction à la synthèse logique, Philippe Larcher, Eyrolles.

[4] Web…