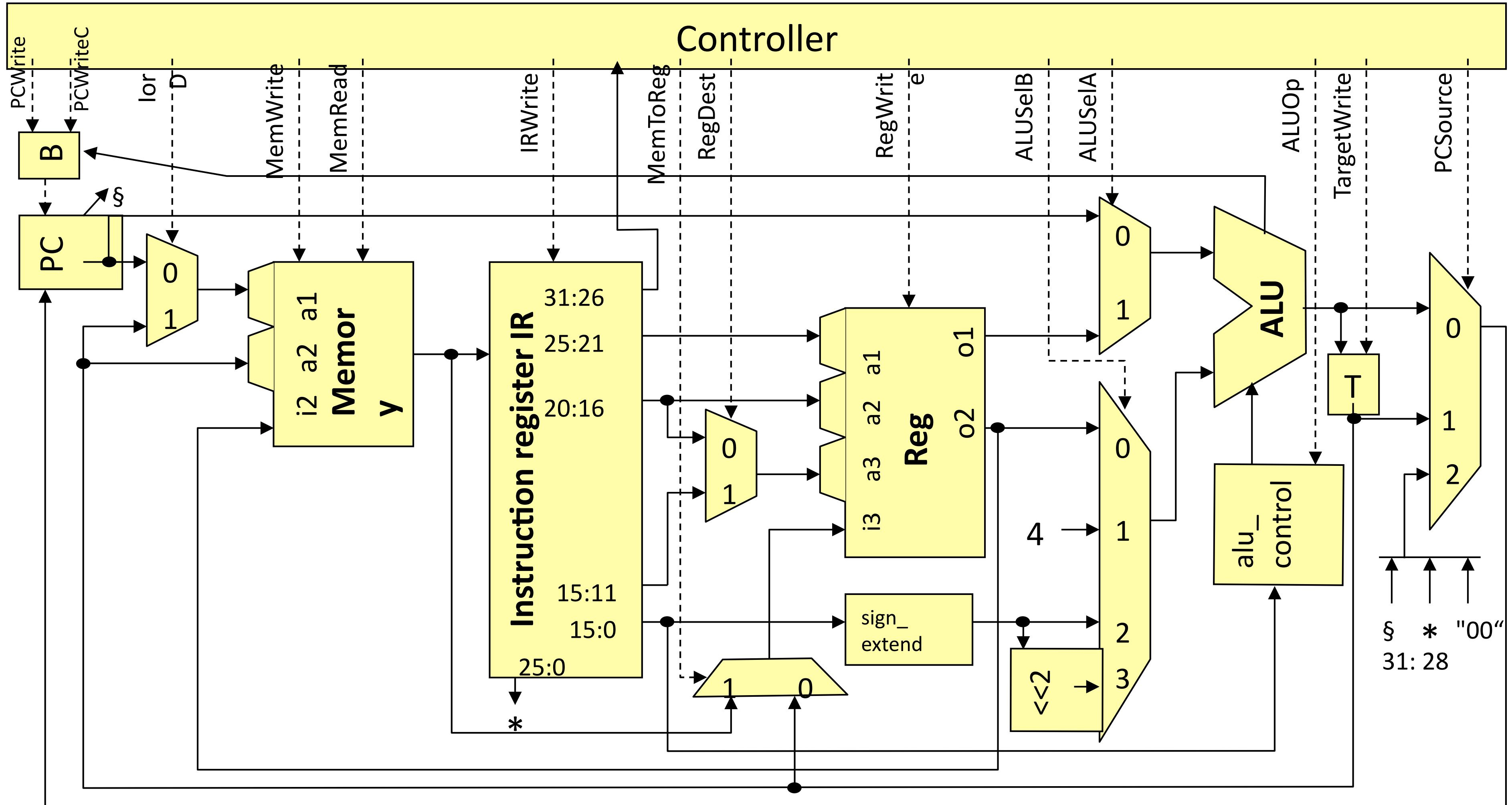
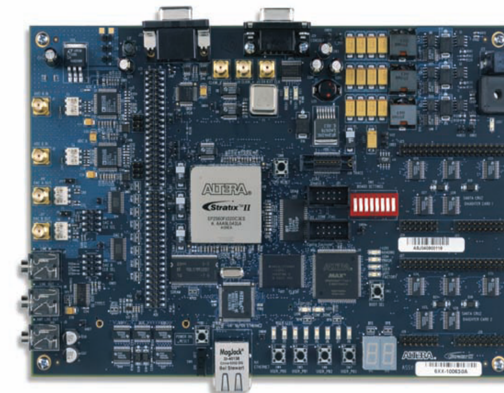
Niveau RTL



RTL implementation

- Hardware description language
  - VHDL
  - Verilog

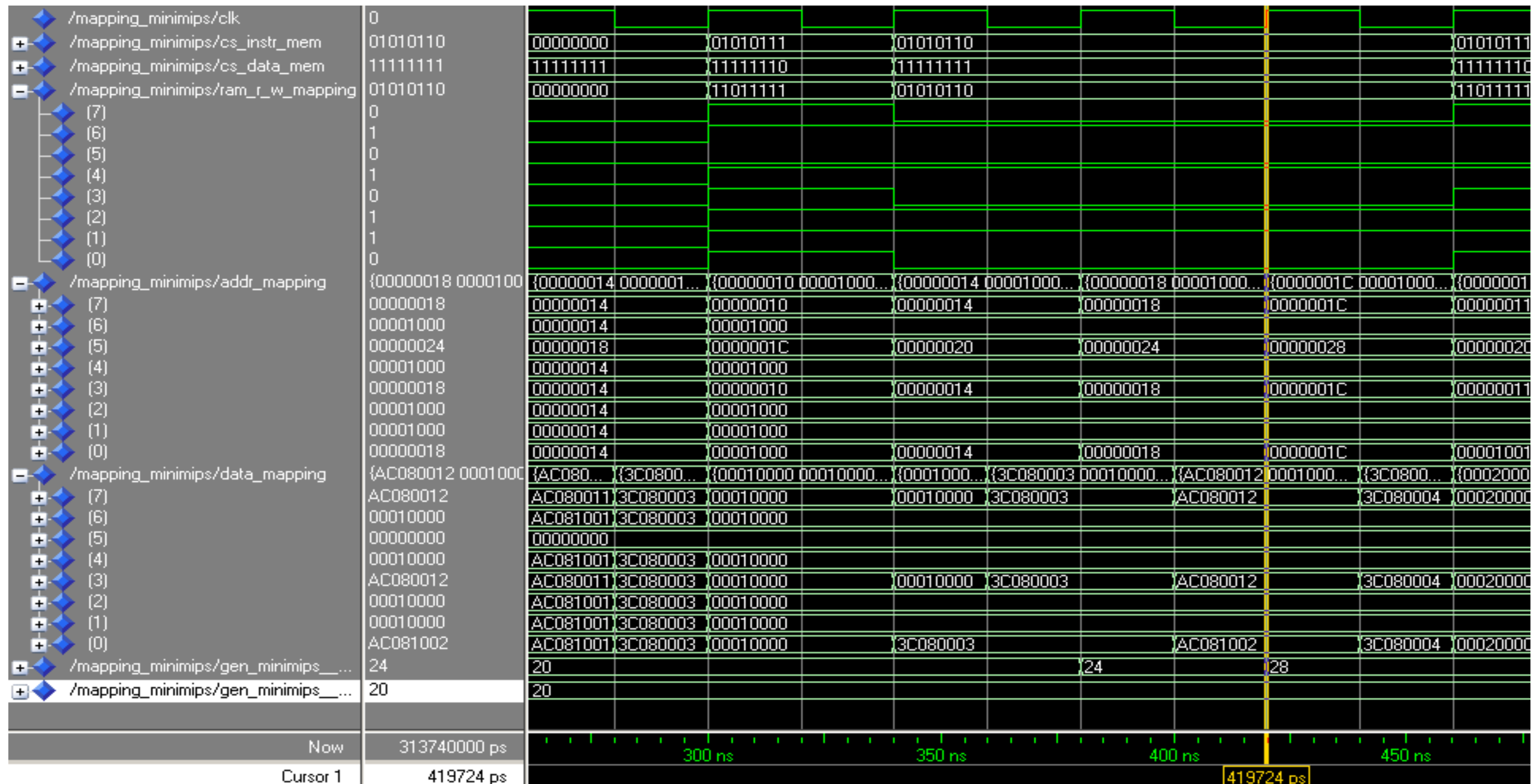
- Target platforms
  - ALTERA : Apex, Stratix…
    - Quartus II
    - Model sim
  - XILINX : Spartron, Virtex…



StratixII EP2S60

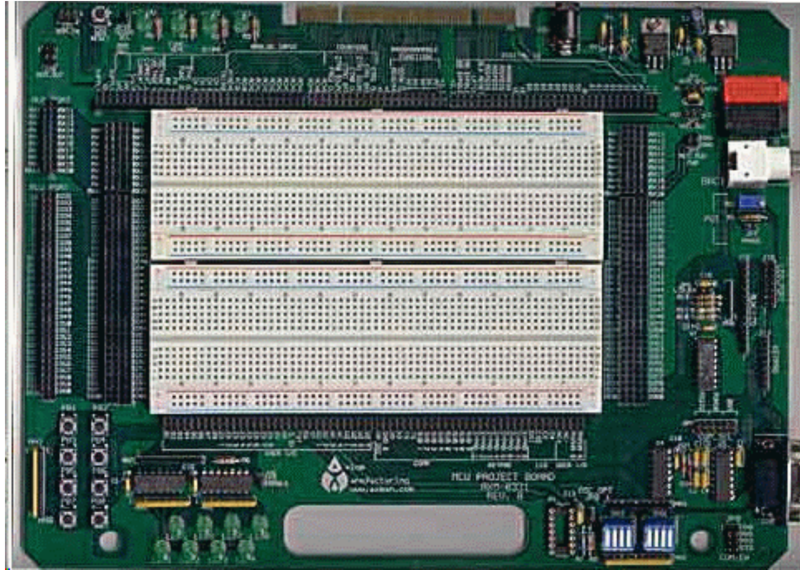
- 48.352 LC
- 493 Pins
- 288 DSP block
- 2.5 Mb Memory

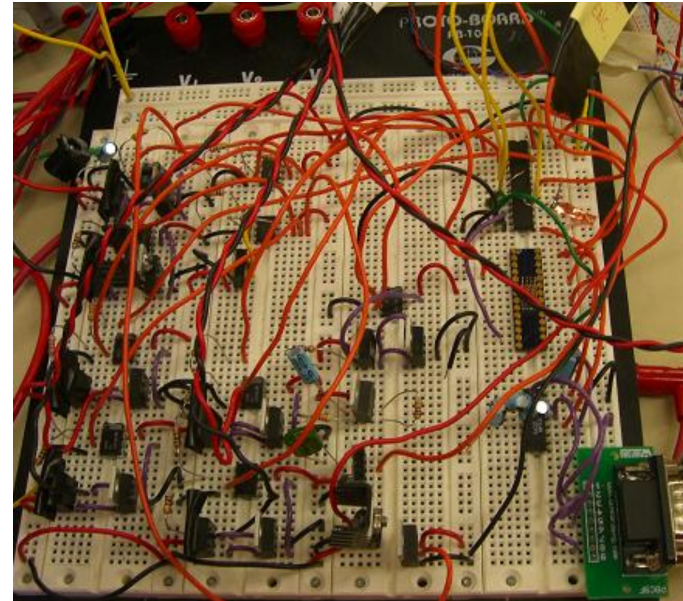
Exemple de simulation RTL

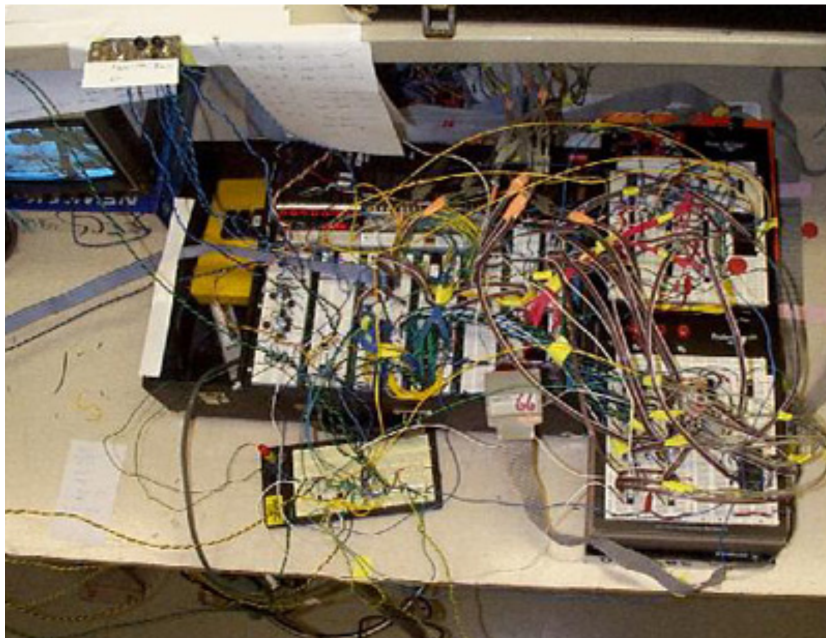


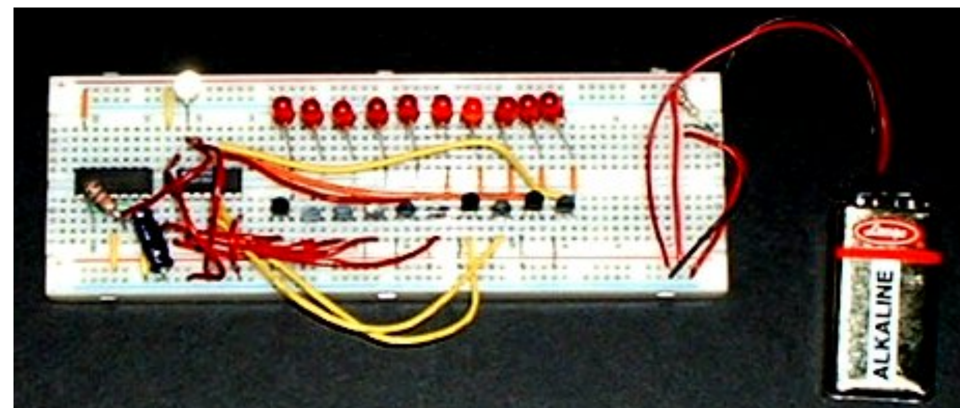
Le prototypage traditionnel

Le prototypage traditionnel









FPGA: Field Programmable Gate Array

Avantages :

=> technologie « facile » à maîtriser
=> temps de développement réduit
=> reprogrammable pour certains (idéal pour le prototypage)
=> coût peu élevé

Inconvénients :

=> performances non optimisées
=> architecture interne entièrement figée
=> système numérique seul (avec quelques exceptions)

Technologie des FPGA

Plusieurs types :

=> programmés par RAM (XILINX et ALTERA)

=> programmés par EEPROM ou FLASH (LATTICE et ACTEL)

=> programmés par antifusible (ACTEL)

Les premiers se prêtent particulièrement bien au prototypage des Systèmes sur puce programmable (SOPC)

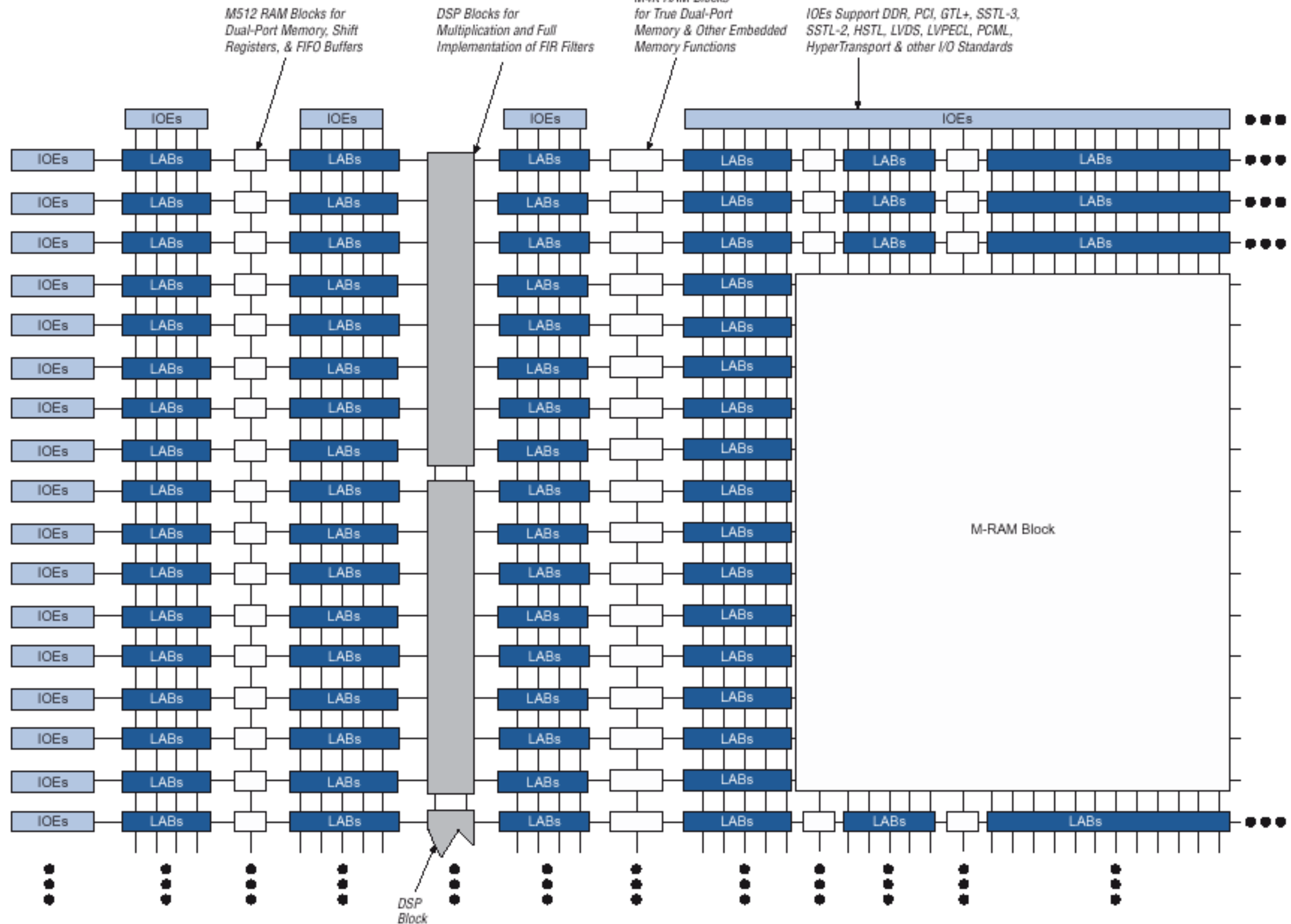
Les FPGA de type SRAM

Caractéristiques d'un circuit récent :

Table 1. Stratix Device Features (Part 1 of 2)

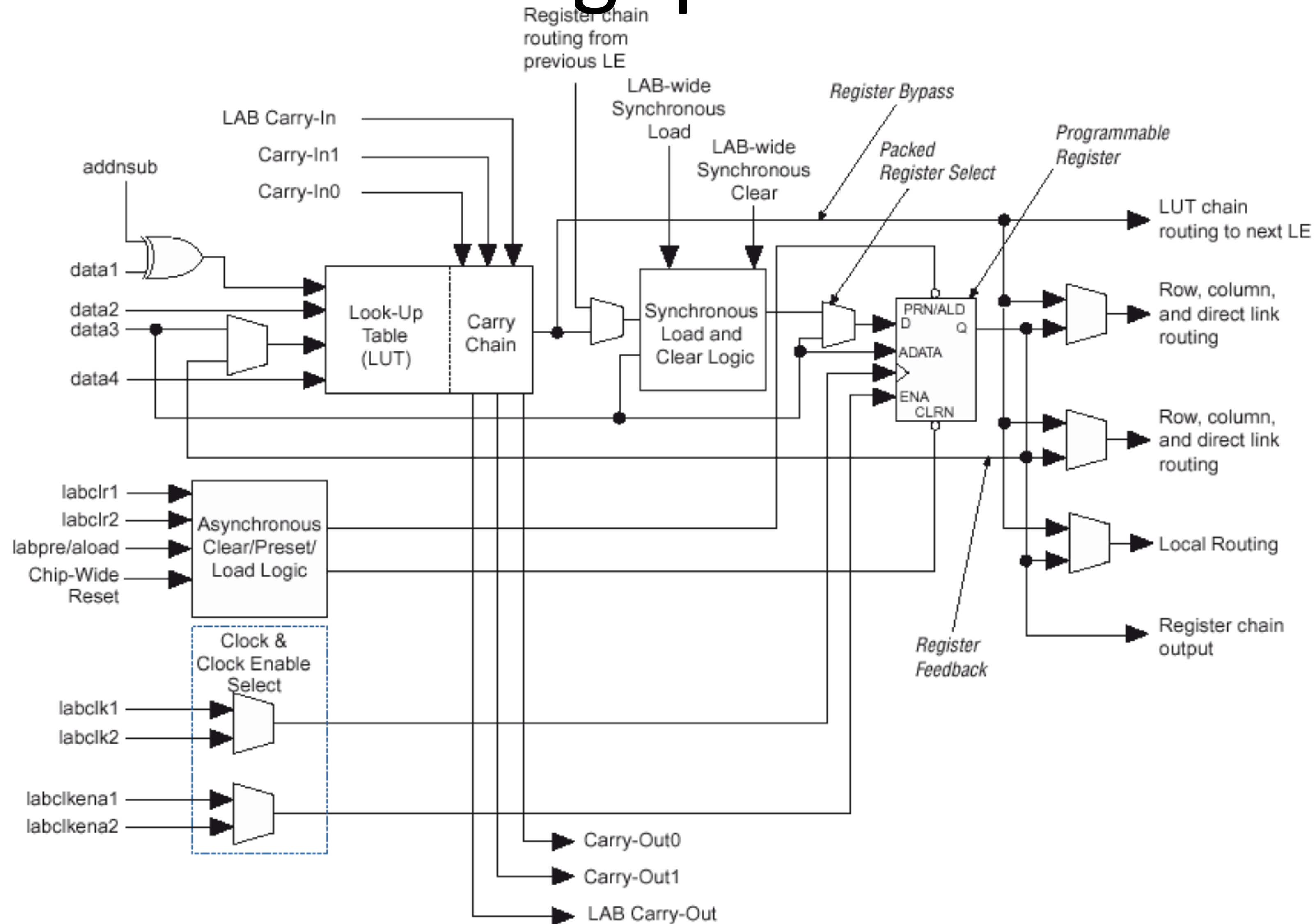
Feature	EP1S10	EP1S20	EP1S25	EP1S30
LEs	10,570	18,460	25,660	32,470
M512 RAM blocks (32 × 18 bits)	94	194	224	295
M4K RAM blocks (128 × 36 bits)	60	82	138	171
M-RAM blocks (4K × 144 bits)	1	2	2	4
Total RAM bits	920,448	1,669,248	1,944,576	3,317,184
DSP blocks	6	10	10	12
Embedded multipliers *(1)*	48	80	80	96
PLLs	6	6	6	10
Maximum user I/O pins	426	586	706	726

Architecture interne d’un FPGA



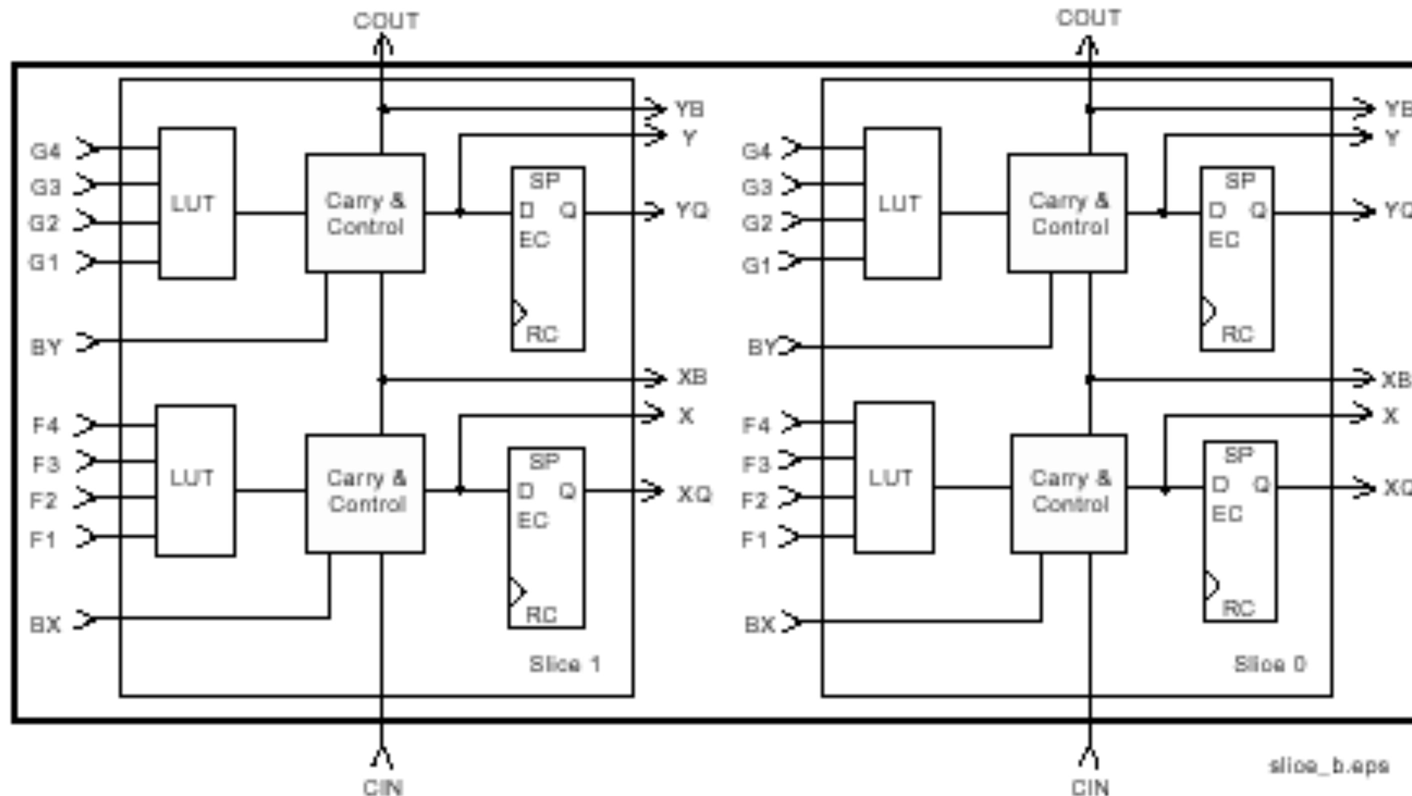
Stratix d’ALTERA

Cellule logique de base



Stratix d'ALTERA

Cellule logique de base



Virtex de XILINX

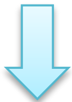
Les FPGA pour les SOPC

- Grand nombre de cellules logiques
- Quantité de mémoire configurable importante
- Entrées sorties compatibles avec de multiples normes
- Blocs spécialisés (multiplieurs, PLLs)
- Reprogrammable

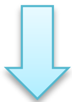
Pour réussir à implanter un système dans un FPGA de manière efficace, il est indispensable de bien connaître sa structure interne et ses limites du point de vue des Performances.

Différence entre le langauge C et VHDL

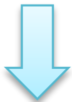
VHDL

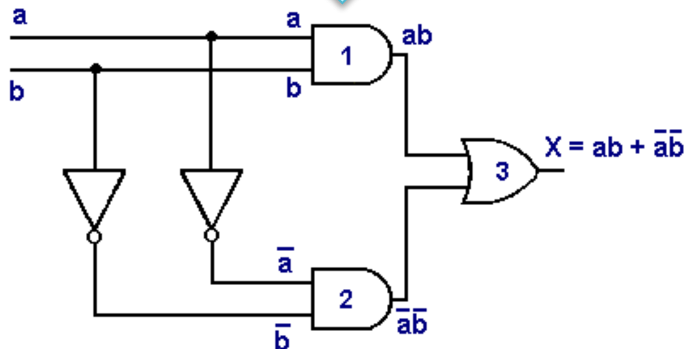


Equation



Synthése



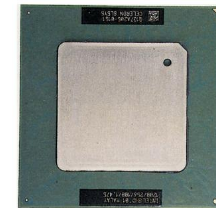


C/C++/pascal

compilation

assembleur

1. mov al,a
2. Mov bl,b
3. Xor al,bl
4. Not al



Plan

1. Introduction aux systèmes embarqués (SE)
2. Flots de conception
3. Outils pour la simulation, la modélisation et l'évaluation des performances
4. Langage VHDL
5. Langage SystemC

Objectifs et caractéristiques du VHDL

- Langage unique pour
  - la spécification, la documentation
  - la vérification (preuve formelle), la simulation
  - la **synthèse** : le langage sert d'entrée à des outils intelligents qui permettent la réalisation de circuits intégrés (customs, ASICs) ou programmables (PALs, PLDs, CPLDs, FPGAs)

$\Rightarrow$ conception, simulation, synthèse de circuits numériques

(du + simple au plus compliqué)

- Interprétable par l'homme et la machine
- Indépendent des process et des technologies, indépendent des systèmes-hôtes

Qu’est-ce qu’un langage de description?

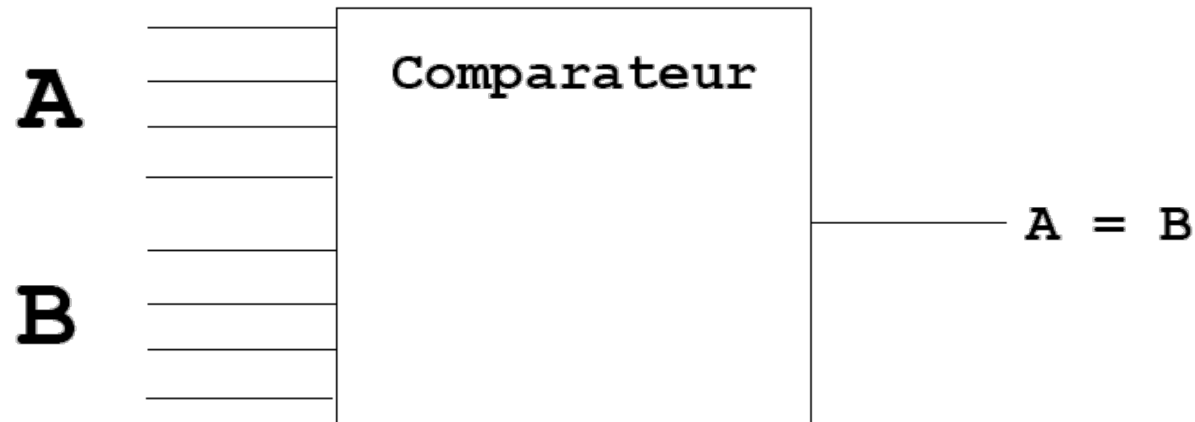
Ce n’est pas un langage informatique classique: il sert à décrire un matériel et à créer un circuit.

- blocs d’instructions exécutés simultanément
- prise en compte de la réalité : possibilité de spécifier des retards, de préciser ce qui se passe en cas de conflits de bus, …
- l’information est associée à des signaux qui modélisent des équipotentielles; elle peut être datée et permettre de construire des chronogrammes

LE VHDL PAR L'EXEMPLE

OPERATIONS COMBINATOIRES.

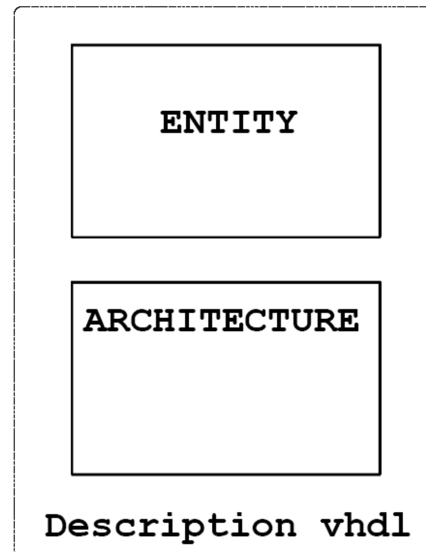
- ***Etude d'un comparateur de deux fois 4 bits.***
- Etudions un comparateur chargé de comparer deux nombres de
- quatre éléments binaires.



- Nous pouvons y distinguer les différentes parties le constituant, à savoir :
- Les entrées A et B de quatre bits chacunes, la sortie A = B.
- Le corps du design délimite la frontière entre la structure interne du comparateur et le monde extérieur.

LE VHDL PAR L'EXEMPLE

- ***Organisation simple d'une description vhdl.***
- Une description vhdl est composée de deux parties: une entité et une architecture.



- L'entité décrit l'interface entre le design et le monde extérieur.
- L'architecture décrit la structure interne du composant. Il y a plusieurs façons de décrire ce fonctionnement.

LE VHDL PAR L'EXEMPLE

La description vhdl du comparateur est donnée ci-dessous :

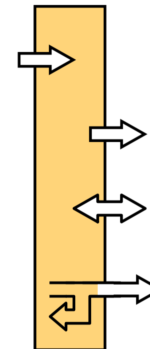
```
-- comparateur de deux fois quatre bits
ENTITY eqcomp4 IS
     PORT (    a0,a1,a2,a3 : IN BIT;
          b0,b1,b2,b3 : IN BIT;
          aeqb : OUT BIT);
END eqcomp4;
ARCHITECTURE logique OF eqcomp4 IS
     BEGIN
     aeqb <= '1'    WHEN (
                    (a0=b0) and
                    (a1=b1) and
                    (a2=b2) and
                    (a3=b3)
                    )
          ELSE '0';
END logique;
```

L'entité est décrite par l'intermédiaire de l'instruction PORT.Celle-ci liste les différentes entrées et sorties du design. Pour chaque donnée transférée à travers un PORT on définit son mode et son type.

L'entité

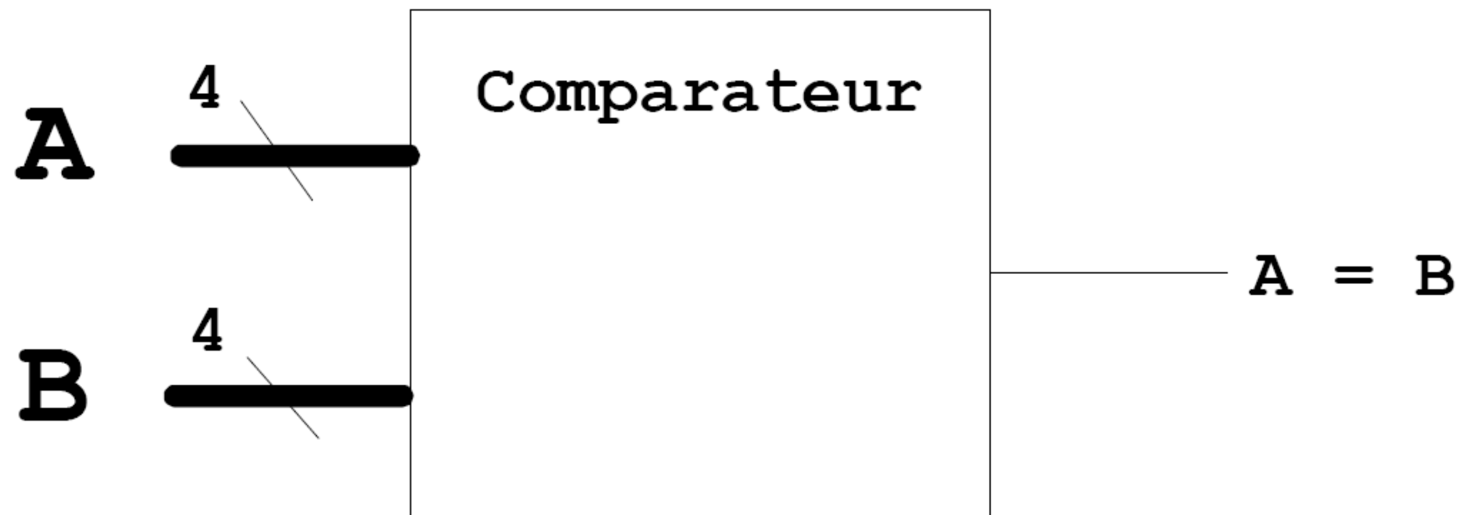
- Les signaux d'entrée/sortie de l'entité sont des PORTS
- un PORT est défini par :
  - un nom
  - un mode (sens)
  - un type

- le MODE correspond au sens de transfert
  - IN entrée (monodirectionnelle)
  - OUT sortie (monodirectionnelle)
  - INOUT entrée/sortie (bidirectionnelle)
  - BUFFER sortie rebouclée



LE VHDL PAR L'EXEMPLE

- Le ***type peut être un type bit, bit_vector, std_logic,*** std_logic_vector, un entier ou encore un type défini par l'utilisateur. Certains type nécessitent, pour être employés, l'utilisation de bibliothèque particulière.
- Utilisons un autre type en regroupant nos entrées en deux vecteurs de quatre bits chacun :



Les types

- Tout objet VHDL doit être associé à un type (objet = signal, constante ou variable)

- Un type définit:
  - L'ensemble des valeurs possibles
  - L'ensemble des opérateurs disponibles

- Organisation des types en VHDL:
  - Types prédéfinis *(integer, bit, bit_vector, boolean,etc…)*
  - Types complémentaires
    - IEEE1164 *(std_logic, td_logic_vector)*
    - Types spécifiques définis par les outils des fournisseurs
    - Types utilisateur (type énuméré, sous-type)

Types et opérateurs prédéfinis

TYPE		*CLASSE*
boolean	→	type énuméré
bit	→	type énuméré
character	→	type énuméré
integer	→	type numérique
natural	→	sous-type numérique
positive	→	sous-type numérique
string	→	chaîne de caractères
bit_vector	→	array of bit
time	→	physique

Tout type appartient à une classe.

OPERATEURS	
Booléens (logiques)	not, and, or, nand, nor, xor, xnor
De comparaison	=, /=, <, <=, >, >=
De décalage	sll, srl, sla, sra, rol, ror
Arithmétiques	sign +, sign -, abs, +, -, *
De concaténation	&

Le type std_logic (IEEE1164)

- Le type *bit* de VHDL peut prendre les valeurs '0' et '1'. Ce type est insuffisant pour décrire des signaux logiques réels (haute-impédance, forçage, etc.)
- Le standard IEEE1164 définit des signaux multi-valeurs répondant aux besoins de systèmes réels, et facilitant la simulation
- Le type *std_logic* (et *std_logic_vector*) possède 9 valeurs :

  '0', '1', 'X', 'L', 'H', 'W', 'Z', 'U', '-'
- L'emploi du type *std_logic* est possible via la library IEEE1164 (cf. déclaration d'une Library) :

```
library ieee;
use ieee.std_logic_1164.all;
```

Les types utilisateurs (exemples)

Syntaxe d'une déclaration de type:

- **type** nom_type_entier **is range** start to end;
- **type** nom_type_énuméré **is** (liste des valeurs);
- **type** nom_type_tableau **is array** (start **to** end) **of** element_type;

Exemples:

- **type** memory_size **is range** 1 to 1024; -- intervalle montant
- **type** etat **is** (reset, stop, wait, go);
- **type** my_word **is array** (0 **to** 31) **of** bit;
- **type** table_vérité **is array** (bit, bit) **of** bit;

Les objets

Objet = élément nommé ayant des valeurs d'un type donné

4 classes d'objets : - contantes
- variables
- signaux
- fichiers

La <u>constante</u> possède une valeur fixe pendant la simulation.

Déclaration d'une constante :

constant nom_constant **:** type [**:=** expression] ;

exemple :
```
constant PERIODE : time :=20 ns;
constant BUS : std_logic :='1';
```

Les variables

C'est un simple "conteneur" dont la valeur peut changer en cours de simulation.

Déclaration d'une variable

variable nom_variable **:** type [**:=** expression];

La variable a une valeur initiale explicite (si elle est précisée dans la déclaration) ou implicite. Dans ce cas elle dépend du type.

Exple : '0' pour le type std_logic

false pour le type boolean

de manière générale, la valeur initiale est la valeur la plus à gauche des valeurs prises par le type

Les signaux

Les signaux représentent des formes d'ondes logiques sous forme de paires temps/valeur. Ils permettent de modéliser les caractéristiques temporelles des signaux réels et de prendre en compte les différents délais.

Ils sont essentiels pour la modélisation de comportements concurrents.

Un signal peut être de n'importe quel type.

Déclaration d'un signal

signal nom_signal**:** type [**:=** expression] **;**

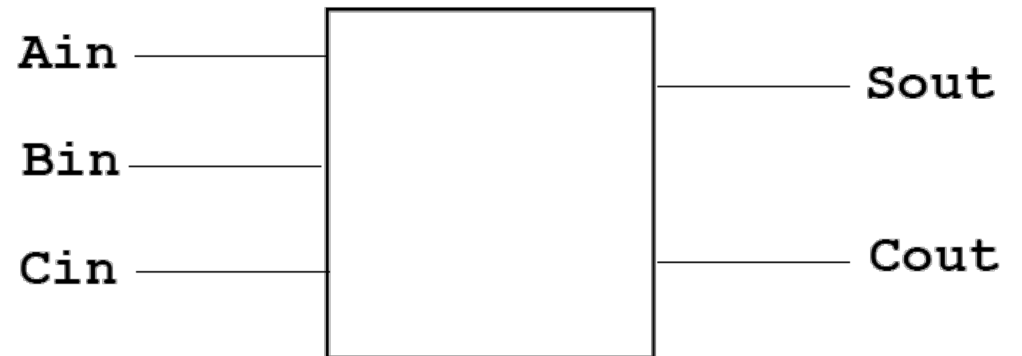
valeur initiale : même chose que pour les variables.

LE VHDL PAR L'EXEMPLE

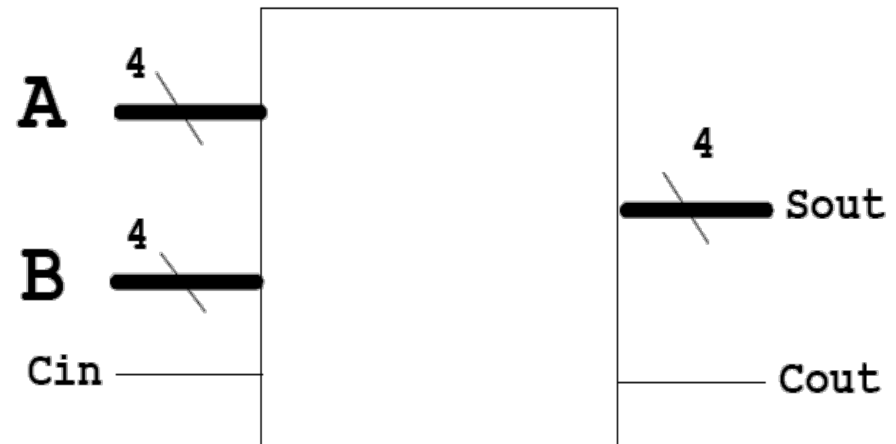
```
-- comparateur de deux fois quatre bits
--
-- utilisation de la bibliothèque nécéssaire au type std_logic
library ieee;
use ieee.std_logic_1164.all;
ENTITY eqcomp4 IS
   PORT (  a : IN STD_LOGIC_VECTOR(3 downto 0);
         b : IN STD_LOGIC_VECTOR(3 downto 0);
         aeqb : OUT STD_LOGIC);
END eqcomp4;
ARCHITECTURE logique OF eqcomp4 IS
BEGIN
aeqb <= '1' WHEN ( a = b ) ELSE '0';
END logique;
```

Exercices

- **Exercice 1 : Définir** l'entité décrivant un additionneur de deux fois un bit.



- **Exercice 2 : Définir** l'entité décrivant un additionneur de deux mots de quatre bits.



Déclaration d'entité

Syntaxe d'une déclaration d'entité:

```
entity nom_entité is
  [generic(liste_generique)]
  [port(liste_port)]
end [entity] nom_entité;
```

EXEMPLES

```
entity full_adder is
        port(      X,Y,Cin: in bit;
           Cout, Sum: out bit);
end full_adder;
```

```
entity comparateur is
  port(
  signal a : in bit_vector(7 downto 0);
  signal b : in bit_vector(7 downto 0);
  signal egal : out bit);
end comparateur ;
```

Déclaration d’un corps d’architecture

<u>Syntaxe d’une architecture</u>:

architecture nom_architecture **of** nom_entité **is**
{signal_declaration} | {constant_declaration}
{type_declaration} | {component_declaration}
begin

{ instruction_concurrente}

end [**architecture**] nom_architecture;

<u>EXEMPLES</u> :

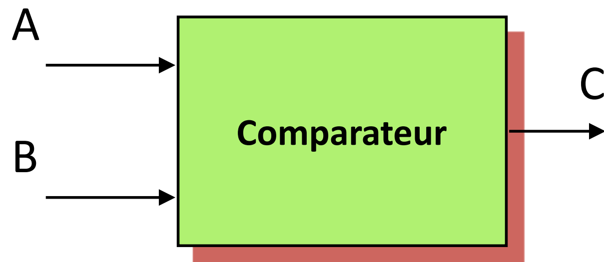
```
architecture simple of comparateur is
-- zone de déclaration vide
begin

    egal <= '1' when a = b else '0';

end simple ;
```

```
architecture dataflow of full_adder is
    signal A, B: bit;
begin
    A<=X xor Y;
    B<=A xor Cin;
    Cout<=B xor (X and Y);
    Sum<=A xor Cin;
end architecture dataflow;
```

L’architecture

- L'**ARCHITECTURE** décrit le fonctionnement de la boîte noire déclarée dans l'ENTITY
- VHDL permet différents niveaux de description :
  - **Haut niveau** (comportemental) : description de la fonctionnalité, sans référence au 'hardware' sous-jacent
  - **Bas niveau** (structurel) : description par utilisation et interconnexion de '*components*' (par ex. portes logiques), comme pour un schéma
  - D'une manière générale les descriptions de haut niveau favorisent la portabilité, les descriptions de bas niveau l'efficacité
- A tout moment (y compris à l'intérieur d'une même architecture), c'est le concepteur qui choisit le niveau de description adéquat

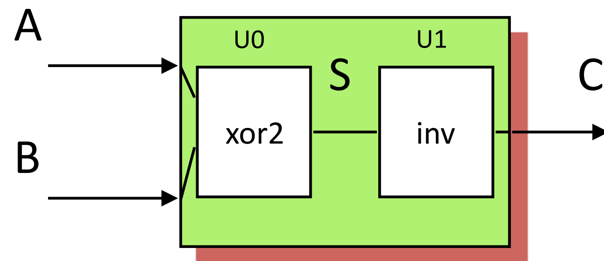
Architecture : description comportementale




```
architecture ARCH1 of COMPARATEUR is
begin
        C <= '1' when (A=B) else '0';
end ARCH1;
```

```
architecture ARCH2 of COMPARATEUR is
begin
        C <= not(A xor B);
end ARCH2;
```

Architecture : description structurelle




```
use work.gatespkg.all;
architecture ARCH3 of COMPARATEUR is
signal S : bit;
begin
    U0 : xor2 port map (A,B,S);
    U1 : inv port map (S,C);
end ARCH3;
```

Les signaux internes sont déclarés en tête de l'architecture (même déclaration que les signaux d'entité, sans 'mode')

Architecture : choix de la description

Les descriptions comportementales sont recommandées.

en *comportemental* :

```
aeqb <= '1' WHEN a = b ELSE '0';
```

en *structurel bas niveau* :

```
u0: xnor2 PORT MAP (a(0), b(0), xnr(0));
u1: xnor2 PORT MAP (a(1), b(1), xnr(1));
u2: xnor2 PORT MAP (a(2), b(2), xnr(2));
u3: xnor2 PORT MAP (a(3), b(3), xnr(3));
u4: and4 PORT MAP (xnr(0), xnr(1), xnr(2), xnr
(3), aeqb);
```

$\Rightarrow$ meilleure compacité $\Rightarrow$ meilleure lisibilité

$\Rightarrow$ meilleure portabilité

Parallèle / Séquentiel

- VHDL permet de décrire des fonctionnements parallèles (*combinatoires*) et séquentiels (*synchrones*)
- Tout repose sur le PROCESS
  - à l'extérieur d'un PROCESS les opérations s'exécutent simultanément, et en permanence
  - à l'intérieur d'un PROCESS les instructions s'exécutent séquentiellement
    - un PROCESS s'exécute à chaque changement d'état des signaux auxquels il est sensible ('sensitivity list')
    - Les signaux modifiés par le process sont mis à jour à la fin de celui-ci
    - les variables déclarées conservent leur valeur d'une activation à une autre (ce qui n'est pas le cas des sous-programmes)

Les instructions concourantes

Rappels :

- elles se trouvent entre les mots réservés **begin** et **end architecture**
- elles sont évaluées indépendamment de leur ordre d'écriture
- elles sont à l'extérieur du PROCESS

Liste d'instructions :

- Assignation (<=)
- assignation conditionnelle (WHEN)
- assignation sélective (WITH)

Assignation simple de signal

Syntaxe: [label:] signal<= expression;

- « signal » ne peut pas être une entrée (mode in).
- « expression » ne peut pas utiliser de signaux de sortie (mode out)
- « signal » et « expression » doivent être de même type.

Exemples

```
C <= not(A xor B);
```

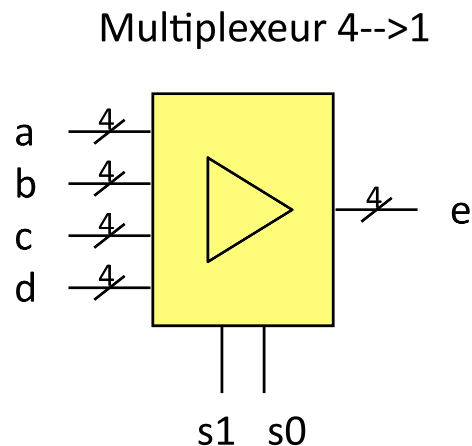
Assignation conditionnelle

Syntaxe: [label:] signal <= {expression **when** condition **else**}
expression [**when** condition];

- Une condition a forcément une valeur booléenne (true, false).
- Le résultat d'un test d'égalité ou d'une comparaison en général est de type booléen

Exemple:

Multiplexeur 4-->1




```
library ieee;
use ieee.std_logic_1164.all;
entity MUX4 is
port( a,b,c,d : in std_logic_vector(3 downto 0);
        e : out std_logic_vector(3 downto 0);
        s1,s0 : in std_logic );
end MUX4;
architecture ARCHMUX4 of MUX4 is
begin
e <=      a when (s1 & s0) = "00" else
          b when (s1 & s0) = "01" else
          c when (s1 & s0) = "10" else
          d;
end ARCHMUX4;
```

Assignation sélective

- Comme l'assignation conditionnelle, l'assignation sélective de signal permet de sélectionner une expression à affecter à un signal, sur la base d'une condition

- Différences:
  - Une seule condition
  - La condition porte sur les valeurs possibles d'une expression ou d'un signal (appelé sélecteur)

    $\Rightarrow$ valeurs testées pas forcément booléennes (plus de deux valeurs possibles)

Assignation sélective

Syntaxe: [label:] **with** expression **select**

signal <= {expression **when** valeur,}
expression when valeur;

- Les conditions sont mutuellement exclusives (une condition ne peut être un sous-cas d'une autre condition)

Exemple:

```
with sel select
  s <= E0 when "00",
       E1 when "01",
       E2 when "10",
       E3 when "11";
```

Le Processus

Syntaxe: [label :] **process** [liste_de_sensibilité)] [**is**]

type_declaration | subtype_declaration | constant_declaration | variable_declaration | function_declaration | procedure_declaration

begin

{instruction_séquentielle}

end process [label];

Instructions séquentielles:

- sont évaluées dans l'ordre dans lequel elles sont écrites
- analogues avec les instructions des langage de programmation conventionnels

Le Processus

- Liste de sensibilité
  - Liste des signaux (séparés par des virgules) dont un événement (changement de valeur) entraîne l'exécution du processus
  - A chaque exécution, l'ensemble des instructions du processus sont évaluées, puis le processus s'arrête en attendant un nouvel événement
  - Cette liste est optionnelle. Dans le cas où elle est omise (et seulement dans ce cas), on doit trouver une instruction **wait** dans la zone des instructions séquentielles.

- Zone de déclaration
  - Les objets déclarés dans cette zone sont locaux au processus.

L'instruction WAIT

Instruction séquentielle, plusieurs instructions possibles dans un même process.

Exemples :

```
-- liste de sensibilité
wait on S1, S2, S3;

-- délai (de type time)
wait for 10 ns;

-- condition
wait until clk = '1';
wait on clk until clk = '1';      -- équiv. au précédant
wait on reset until clk = '1';  -- plus sensible sur clk!
```

```
-- forme générale
wait on S1, S2 until en = '1' for 15 ns;

-- stop définitif (pas de réactivation possible)
wait;
wait until next_event;
    -- si: variable next_event: boolean;
```

Le Processus

- Instructions séquentielles
  - assignation simple (de signal ou de variable)
  - if
  - case
  - for

Instruction IF

- C 'est l 'équivalent séquentiel de l 'assignation conditionnelle

Syntaxe : [label :] **if** condition **then**
{instruction_sequentielle}
{**elsif** condition **then**
{instruction_sequentielle} }
[**else** {instruction_sequentielle}]
end if [label];

Exemples

```
process (a,b)
begin
 if a=b then
       equal <='1';
 else
       equal<='0';
 end if;
end process;
```

```
process (a,b,sel1,sel2)
begin
 if sel1='1' then     z<=a;
 elsif sel2='1' then     z<=b;
 else    z<=c;
 end if;
end process;
```

Instruction CASE

- C 'est l 'équivalent séquentiel de l'assignation sélective de signal.
  (Permet l 'execution d 'une opération en fonction de la valeur prise par une expression)

Syntaxe :

```
[label :] case expression is
        when valeur  => {instruction_sequentielle}
        {when valeur => {instruction_sequentielle}}
      end case [label];
```

Exemple :

```
process(light)
begin
  case light is
        when red =>        next_light <= green;
        when amber =>      next_light <= red;
        when green =>      next_light <= amber;
  end case;
end process;
```

Attention : valeur et expression doivent être du même type

Instruction de boucle (1/3)

Instruction FOR

Syntaxe :

[etiquette :] **for** identifiant **in** discrete_range **loop**
{instruction_sequentielle}
end loop [etiquette];

Exemple :

```
-- a, b et s sont des signaux de type
-- std_logic_vector(7 downto 0)

process(a,b)
begin
for i in 7 downto 0 loop
    s(i) <= not(a(i) xor b(i));
  end loop;
end process;
```

Instruction de boucle (2/3)

Instruction WHILE

Syntaxe :

[etiquette :] **while** condition **loop**
{instruction_sequentielle}
end loop [etiquette];

Exemple :

```
process(a,b)
begin
while i < 10  loop
        i<=i+1;
end loop;
end process;
```

Instruction de boucle (3/3)

Boucle générale

Syntaxe :

```
[etiquette :] loop
        {instruction_sequentielle}
        end loop [etiquette];
```

Exemples :

L'instruction **exit** stoppe l'itération et sort de la boucle.

L'instruction **next** stoppe l'itération courante et execute la suivante.

Boucle infinie

```
loop
    wait until clk = ' 1 ';
    q<=d after 5 ns;
end loop;
```

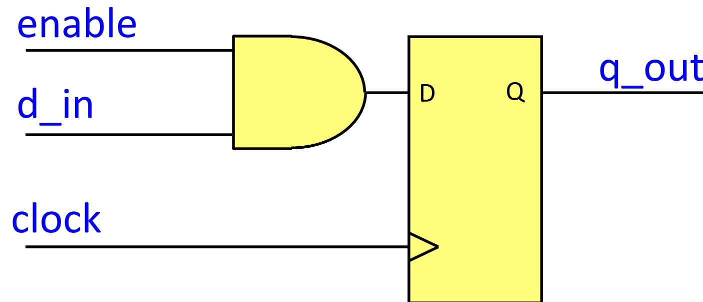
Boucle avec sortie

```
L:loop
    exit L when value = 0;
    valeur := valeur /2;
end loop;
```

Synchronisation par une horloge

Les 2 descriptions suivantes sont équivalentes :

```
PROCESS
BEGIN
        WAIT UNTIL clock='1';
        IF enable='1' THEN
                q_out <= d_in;
        ELSE
                q_out <= '0';
        END IF;
END PROCESS;
```




```
PROCESS (clock)
BEGIN
        IF (clock'EVENT and clock='1');
                IF enable='1' THEN
                        q_out <= d_in;
                ELSE
                        q_out <= '0';
                END IF;
        END IF;
END PROCESS;
```

Rmq : L’instruction WAIT UNTIL peut se substituer à la liste de sensitivité d’un PROCESS. Elle s’utilise essentiellement lorsque le process n’est activé que par un seul signal.

Déclaration du reset

Reset synchrone

```
PROCESS (clk)
  BEGIN
     IF clk'EVENT AND clk= '1' THEN
        IF reset = '1' THEN  count <= "0000";
        ELSE      count <= count + 1;
        END IF;
     END IF;
     END PROCESS;
```

Le process n'est sensible qu'au signal clk

Reset asynchrone

```
PROCESS (clk, rst)
  BEGIN
     IF rst = '1' THEN       count <= x"0";
     ELSIF (clk'EVENT AND clk = '1') THEN
          count <= count + 1;
     END IF;
  END PROCESS;
```

Le process est sensible aux signaux CLK et RST. Le reset est prioritaire.

Remarque : la mémorisation implicite

- En VHDL, les signaux ont une valeur courante, et une valeur suivante.
- Si la valeur suivante d'un signal n'est pas spécifiée, elle est supposée identique à la valeur courante.

Avantage : la description d'éléments mémoire est simplifiée

```
if (clk'event and clk='1') then Q<= D;
end if;
```

```
if (clk'event and clk='1') then Q<= D;
        else Q <= Q;
end if;
```

Inconvénient : la logique générée peut devenir beaucoup plus compliquée

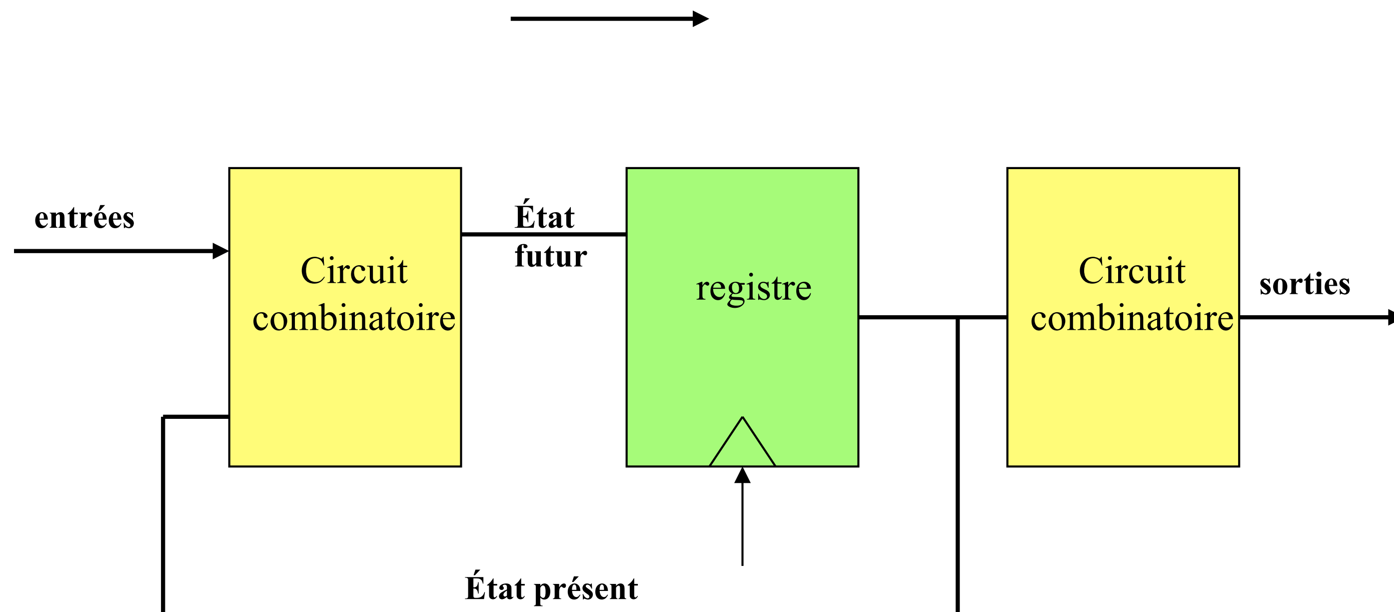
Description de machines d'état

<u>Machine de Moore et de Mealy</u>

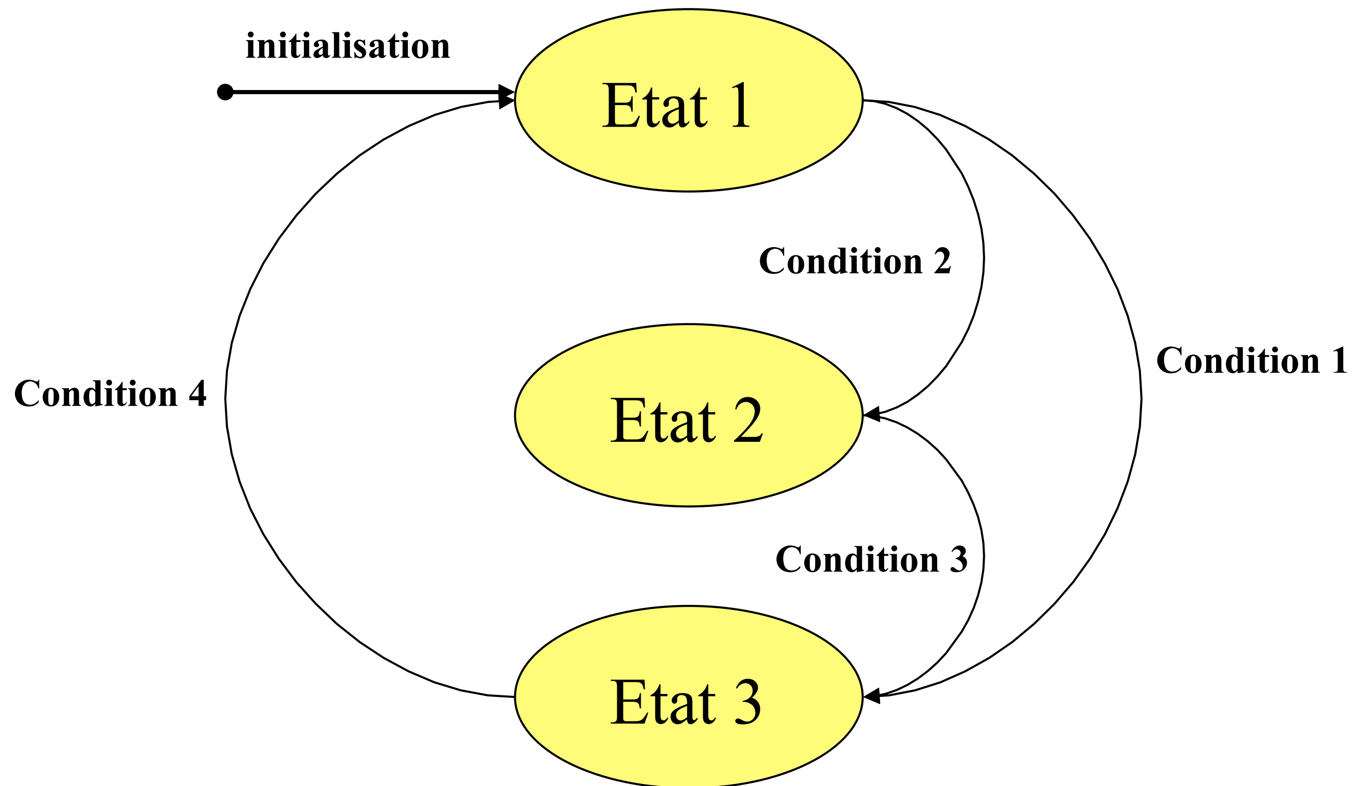
description de circuits séquentiels, où on raisonne en

état présent *état futur*

avec des conditions de transitions



Exemple (1/2)



Exemple (2/2)

```
Architecture machine_arch of machine is
signal etat : integer range 1 to 3;
begin
process (clk)
begin
if clk'event and clk='1' then
    if initialisation = '1' then etat <= 1;
        else
          case  etat is
         when 1=> if condition 1 then etat <=3;
                   elsif condition 2 then etat <=2;
                 end if;
         when 2 => if condition 3 then etat <= 3; end if;
         when 3 => if condition 4 then etat <=1; end if;
          end case;
      end if;
end process;
end machine-arch;
```

Remarque :on peut définir un type état

architecture ……is

type etat_type is (etat1, etat2, etat3);

signal etat : etat_type;

begin

………

Le type état est un type énuméré, le nom des état peut être quelconque (début, fin, vitesse, marche, arrêt,…..