Syst de Télécom Embarqués
Chapitre III: Le flot de conception d'un circuit FPGA - VHDL

Introduction au VHDL

- Very high speed integrated circuit Hardware Description Language
- Langage de description de structures électroniques numériques

Historique

- Définition au milieu des années 80 par le Département de la Défense américain
- Standardisation du langage VHDL en 1987 (norme IEEE 1076.1987)
- Evolution du langage VHDL en 1993 (norme IEEE 1076.1993)
- Extension (IEEE 1076.2008)

Introduction au VHDL

- Very high speed integrated circuit Hardware Description Language
- Langage de description de structures électroniques numériques

Historique

- Définition au milieu des années 80 par le Département de la Défense américain
- Standardisation du langage VHDL en 1987 (norme IEEE 1076.1987)
- Evolution du langage VHDL en 1993 (norme IEEE 1076.1993)
- Extension (IEEE 1076.2008)

Introduction au VHDL

Un modèle VHDL peut être :

- Comportemental: décrit la fonctionnalité d'un objet par un algorithme séquentiel, ou table de vérité, sans référence à une structure d'implémentation
  - Autrement: Flux de données ou RTL: décrit le flux entre entrée et sortie au niveau bit, par des équations élémentaires
- Structurel: Décrit la constitution de l'objet en un ensemble d'objets élémentaires interconnectés (proche du schéma)

Deux jeux d'instructions en VHDL :

- Instructions séquentielles: elles s'exécutent les unes après les autres
- Instructions concurrentes: elles s'exécutent en même temps

Introduction au VHDL

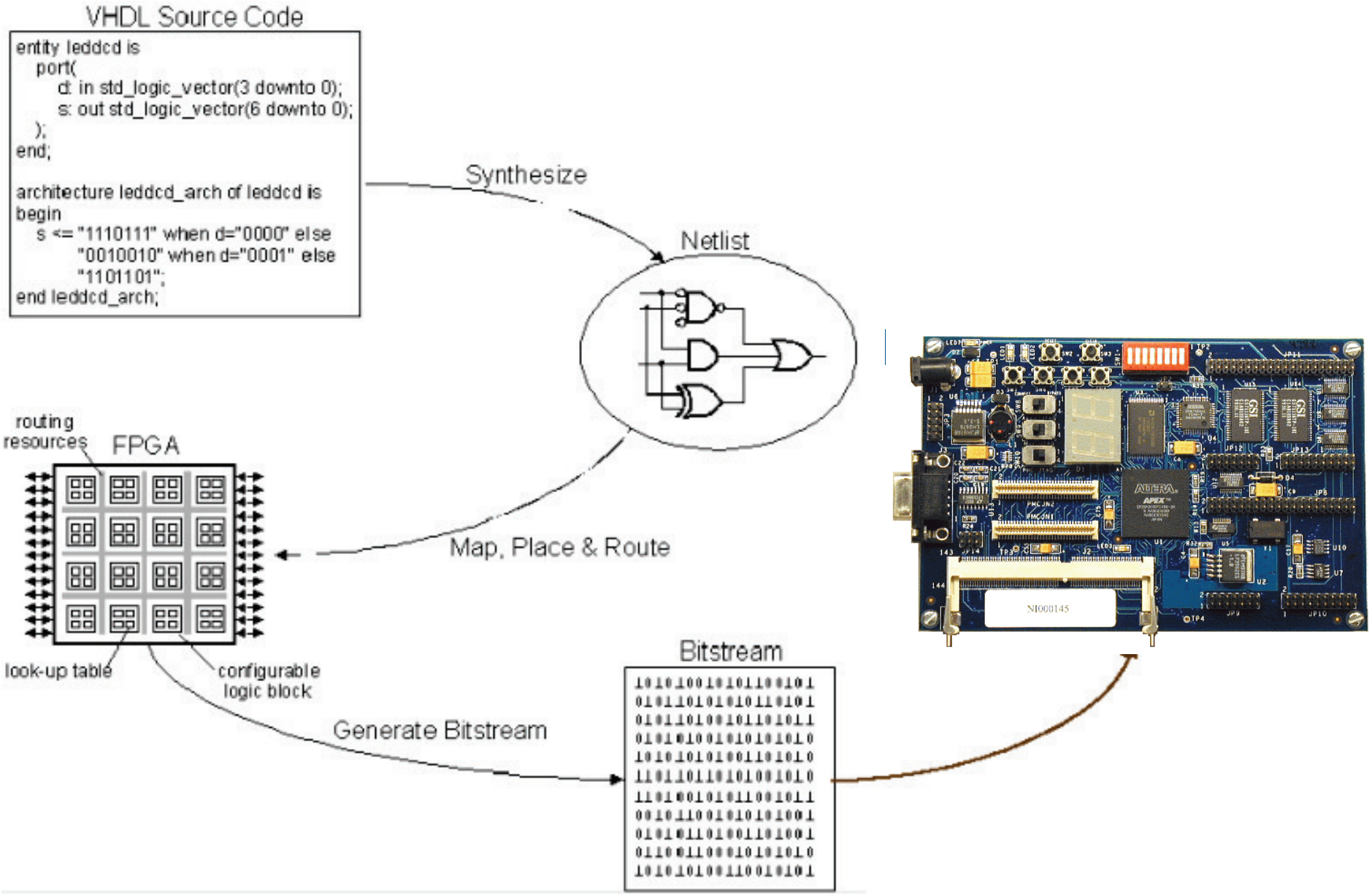
Un modèle VHDL peut être :

- Comportemental: décrit la fonctionnalité d'un objet par un algorithme séquentiel, ou table de vérité, sans référence à une structure d'implémentation
  - Autrement: Flux de données ou RTL: décrit le flux entre entrée et sortie au niveau bit, par des équations élémentaires
- Structurel: Décrit la constitution de l'objet en un ensemble d'objets élémentaires interconnectés (proche du schéma)

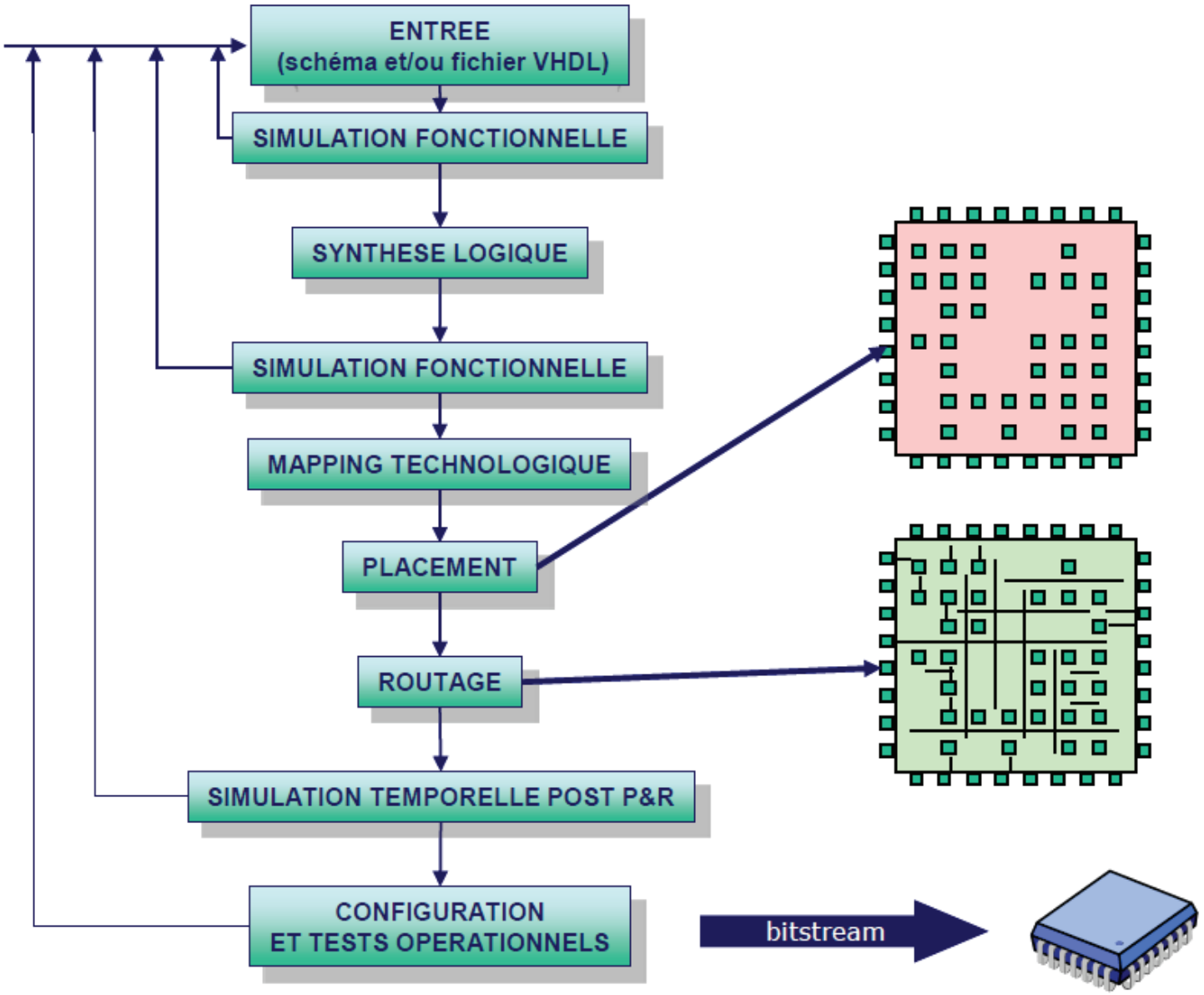
Deux jeux d'instructions en VHDL :

- Instructions séquentielles: elles s'exécutent les unes après les autres
- Instructions concurrentes: elles s'exécutent en même temps

Flot de synthèse:



Flot détaillé de Chez Xilinx



Les outils de synthèse ciblant les FPGA

Les fabricants de FPGA proposent des outils de CAO pour la programmation de leurs circuits

Xilinx

- ISE-Foundation

Altera

- Quartus

- Implémentation (placement-routage & configuration)
- Synthèse logique et de simulation VHDL

Partie I: Programmation des portes logiques

Entités et architectures

- Description d'une porte AND à deux entrées:

```
entity And2 is
    port (x, y : in BIT; z: out BIT);
end entity And2;
architecture ex1 of And2 is
begin
    z ⇐ x and y;
end architecture ex1;
```

- La partie "entity" décrit une boite noire
- Ce qui est vu: Entrées et sorties de la boite noire et leur type
- Aucune idée sur cette boite dedans
- L'architecture décrit la structure des fonctions And/Or du circuit

Conventions lexicales

- En général, les instructions se terminent par ";"
- 26 lettres de l'alphabet, les 10 chiffres et " _ "
- Le premier caractère est une lettre
- Il ne peut y avoir 2 " _ " de suite
- L'identifieur ne peut se terminer par " _ "
- Majuscule / Minuscule: Pareils
- Identificateurs: Noms de variables, de signaux, de fonctions ... etc
  - Ils ne peuvent pas contenir d'espace
- Littéraux : Ce sont des valeurs explicites
  - 67 est un littéral pour le type entier
  - '0' est un littéral pour un bit
  - "001" O"562" X"FF1" sont des littéraux pour les vecteurs de bits
  - "chaine" est un littéral de type chaine de caractères
- Commentaires: Ils commencent par 2 tirets "- -"

Netlists

A	B	C	Z
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

- $Z=\overline{A}.B+A.C$
- Code VHDL:

```
entity comb_function is
  port (a, b, c : in BIT; z: out BIT);
end entity comb_function;
architecture expression of
comb_function is
begin
  z⇐ (not a and b) or (a and c);
end architecture expression;
```

- Bien que ce programme décrit parfaitement la fonction logique, il ne peut être implémenté sur un circuit
- Réécriture du programme en terme portes logiques

Netlists

- En plus du And2, nous avons:
- Il n'y a auncun conflit en appelant chaque architecture ex1 du moment qu'elles sont liées à deux "entity" différentes

```
entity Or2 is
  port (x, y : in BIT; z: out BIT);
end entity Or2;
architecture ex1 of Or2 is
begin
  z ⇐ x or y;
end architecture ex1;
entity Not1 is
  port (x : in BIT; z: out BIT);
end entity Not1;
architecture ex1 of Not1 is
begin
  z ⇐ not x;
end architecture ex1;
```

Netlists

La description de la fonction logique peut s'écrire alors sous la forme:

```
architecture netlist of comb_function is
signal p, q, r : BIT;
begin
g1: entity WORK.Not1(ex1) port map (a, p);
g2: entity WORK.And2(ex1) port map (p, b, q);
g3: entity WORK.And2(ex1) port map (a, c, r);
g4: entity WORK.Or2(ex1) port map (q, r, z);
end architecture netlist;
```

Netlists

- Dans cette description VHDL nous avons simplifié l'écriture
- La déclaration signal rassemble toute la liste des signaux avec leur type
- Instantiation des portes
- Consiste à introduire une référence entre ”begin” et ”end”
- Non seulement nous déclarons chaque porte explicitement
- On défini aussi l'emplacement où se trouvent les modèles et leurs architectures
- WORK signifie la librairie courante
- Car chaque ”entity” et architecture, une fois compilée, est stockée dans un répértoire
- L'instance a un nom ”g1”
- ”port map” pour montrer comment sont liés les entrées et les sorties des portes (mapping)
- L'ordre de signaux est très important:
  g2: entity WORK.And2(ex1) port map (z $\Rightarrow$ q, x $\Rightarrow$ p, y $\Rightarrow$b);

Netlists

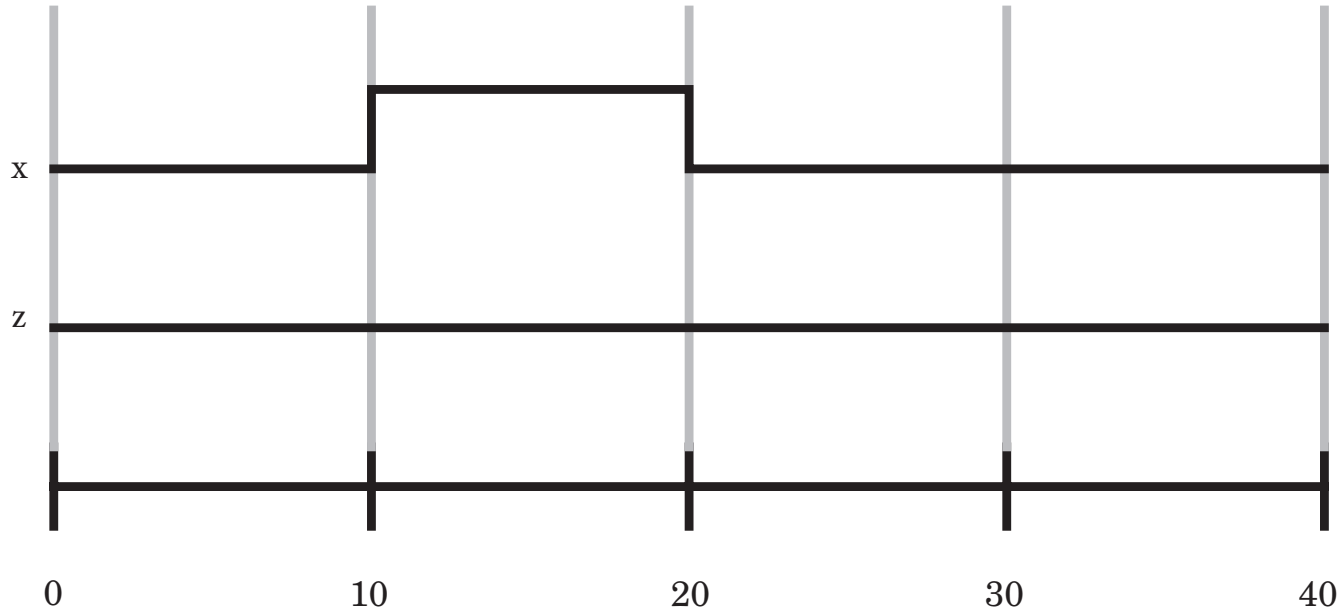
```
architecture netlist2 of comb_function is
component And2 is
  port (x, y : in BIT; z: out BIT);
end component And2;
component Or2 is
  port (x, y : in BIT; z: out BIT);
end component Or2;
component Not1 is
  port (x : in BIT; z: out BIT);
end component Not1;
  signal p, q, r : BIT;
begin
  g1: Not1 port map (a, p);
  g2: And2 port map (p, b, q);
  g3: And2 port map (a, c, r);
  g4: Or2 port map (q, r, z);   end architecture netlist2;
```

Netlists

- La déclaration ”component” informe le compilateur à quoi ressemble chaque porte
- Identique à la déclaration ”entity” $\simeq$
- Si les noms des ports sont les mêmes
- Si chaque porte a une seule architecture
- Si plusieurs architectures, la dernière compilée est celle utilisée
- Connu par ”configuration par défaut”

Assignement des signaux

- La porte And2 a l'assignement z $\Leftarrow$ x and y;
- Ceci peut-être utilisé pour une opération logique plus complexe: z $\Leftarrow$ not ((x and y) or (a and b));
- Retard: z $\Leftarrow$ x after 20 NS;
- Cela signifie que chaque entrée est affectée après 20 NS à z

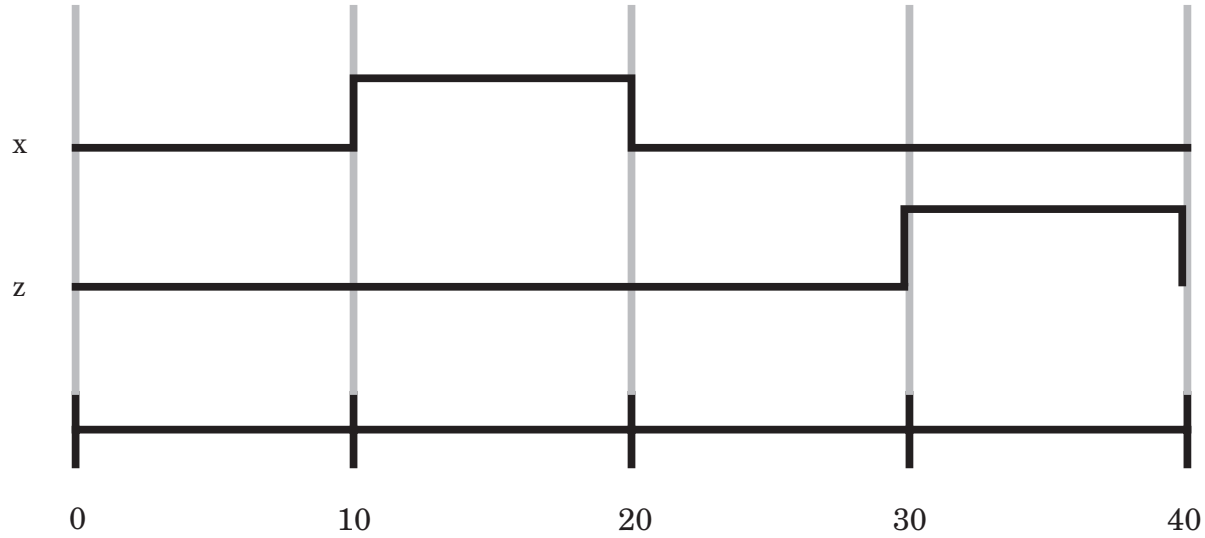


- Ne pas oublier l'espace entre 20 et NS

Assignement des signaux

Cela peut s'appliquer à n'importe quelle expression logique

- z $\Leftarrow$ x and y after 5 NS;
- Utilisation aussi de "transport": z $\Leftarrow$ transport x after 20 NS;



- z $\Leftarrow$ transport not ((x and y) or (a and b)) after 8 NS;

Résumé

La fonction logique décrite par $Z=\overline{A}.B+A.C$ peut-être implémentée avec différentes façons

Exercice

Soit une porte logique universelle, admettant 3 entrèes (x, y, Invert) et 2 sorties (1 correspond à AND, 1 correspond à OR)

- Extraire les fonctions logiques correspondantes aux deux sorties
- Donner un code VHDL décrivant cette porte universelle (entité et architecture)

Réponse

```
entity universal is
  port (x, y, invert : in BIT; a, o : out BIT);
end entity universal;
architecture univ of universal is
  begin
  a ⇐ (y and (x xor invert)) or (invert and not y);
  o ⇐(not x and (y xor invert)) or (x and not invert);
end architecture univ;
```

Constantes et ports ouverts

- Pour que la porte universelle vue dans l'exemple se comporte comme AND, il faut que "Inverter" soit à "0"
- Laisser la sortie "o" non-connectée en utilisant le mot "open"

```
u0 : entity WORK.universal(univ) port map (x, y, '0', a, open);
```

- Les sorties peuvent-être laisser non-connectées
- Mais les entrées ne peuvent-être non-connectées que s'il y a une valeur de sortie par défaut

```
entity universal is
  port (x, y : in BIT;
        invert : in BIT := '0';
        a, o : out BIT);
end entity universal;
```

- L'instanciation peut s'écrire maintenant comme:

```
u0 : entity WORK.universal port map (x, y, open, a, open);
```

Testbenches

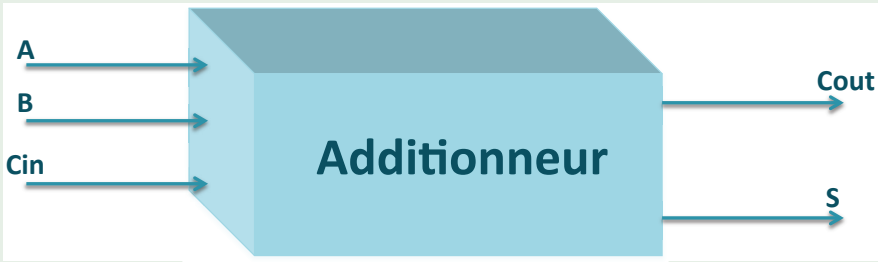
- Si nous voulons simulé notre circuit pour savoir s'il fonctionne réellement comme voulu
- Nous aurons besoin de faire quelques tests: Testbench
- Pour la porte AND à deux entrées, nous écrivons:

```
entity TestAnd2 is
end entity TestAnd2;
architecture io of TestAnd2 is
  signal a,b,c : BIT;
begin
  g1: entity WORK.And2(ex2) port map (x⇒a, y⇒b, z⇒c);
  a⇐ '0', '1' after 100 NS;
  b⇐ '0', '1' after 150 NS;
end architecture io;
```

Partie II: Programmation des Blocs logiques

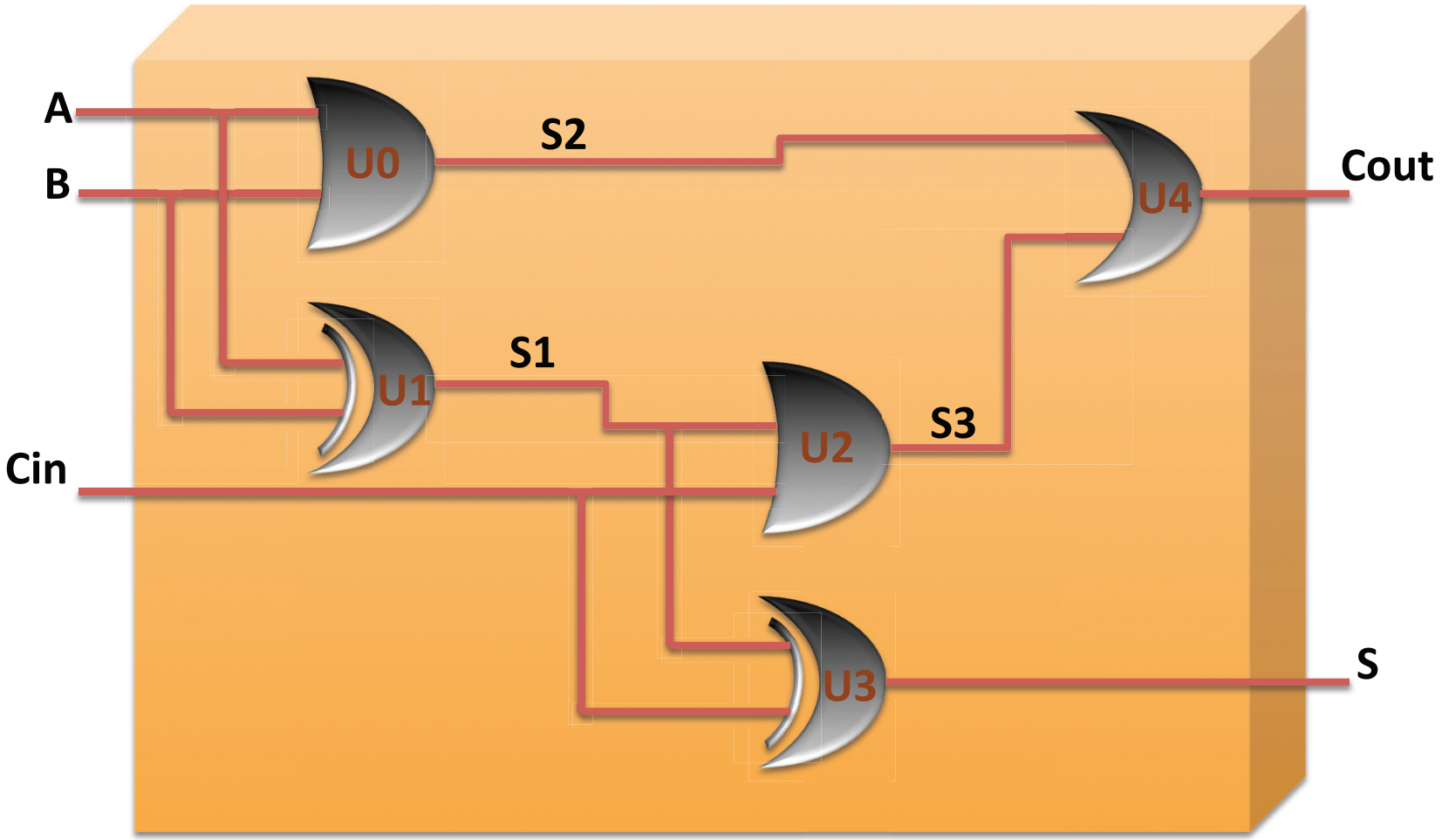
Exercice

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1



- Donner en utilisant ”architecture” et ”component” puis l'instanciation une description structurelle de cet additionneur

Solution



Solution

```
architecture structurelle1 of Additionneur is
  component porteOU
  port ( e1 : in bit;
         e2 : in bit;
         s : out bit );
  end component;
  component porteET
  port ( e1 : in bit;
         e2 : in bit;
         s : out bit );
  end component;
  component porteXOR
  port ( e1 : in bit;
         e2 : in bit;
         s : out bit );
end component;
```

Solution

```
  signal S1, S2, S3 : bit;
begin
  u0 : porteET
       port map ( A, B, S2);
  u1 : porteXOR
       port map ( A, B, S1);
  u2 : porteET
       port map ( S1, Cin, S3);
  u3 : porteXOR
       port map ( S1, Cin, S);
  u4 : porteOU
       port map ( S2, S3, Cout);
end structurelle1;
```

Librairies

- En réalité chaque modèle VHDL est préfixé par ces lignes:

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

- C'est un ensemble de "Packages" contenant des fonctions compilées est prédéfinies
- Dans le package std_logic_1164 :
- Définition des types:
  - std_logic
  - std_logic_vector
- Définition des fonctions logiques de base sur ces types:
  - AND, OR, NOR, XOR, etc

Exemple d'utilisation:


```
ENTITY black_box IS PORT (
    clk, rst: IN std_logic;
    d: IN std_logic_vector(7 DOWNTO 0);
    q: OUT std_logic_vector(7 DOWNTO 0);
    co: OUT std_logic);
END black_box;
```

Autres éléments du langage

- Instructions conditionnelles:
  - Structure if then else end if:

```
if condition_boolenne then
        sequence d'instructions 1 ;
else
        sequence d'instructions 2 ;
end if;
```

  - Structure if then elseif else end if:

```
if condition_1 then
        sequence d'instructions 1 ;
elseif condition_2 then
        sequence d'instructions 2 ;
elseif condition_3 then
        sequence d'instructions 3 ;
else
        sequence d'instructions 4 ;
end if;
```

Autres éléments du langage

- Instructions conditionnelles:
  - Structure case:

```
architecture case expression is
        when valeur1              ⇒ sequence d'instructions ;
        when valeur2 | valeur3    ⇒ sequence d'instructions ;
        when valeur4 to valeur5   ⇒ sequence d'instructions ;
        when others               ⇒ sequence d'instructions ;
end case;
```

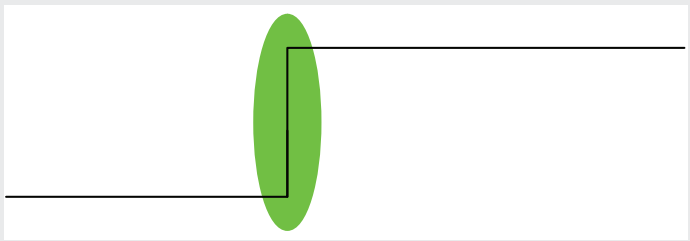
  - Structure when_else:

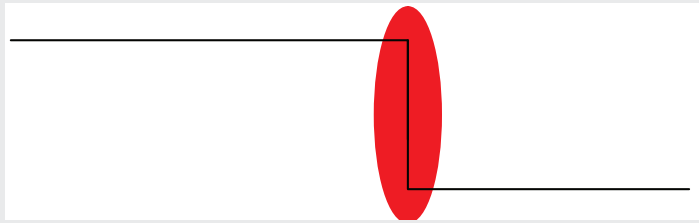
```
architecture when_else of boite_noire
begin
    z ⇐ "sortie_logique1" when entrée = "entrée_logique1" else
        "sortie_logique2" when entrée = "entrée_logique2" else
        "XXXX...X" - - length(sortie_logique" )
end when_else
```

else "XXXX...X" : Sinon en VHDL z continuera à prendre la dernière valeur

Autres éléments du langage

- Instructions de boucles:
  - Boucle while:
    while condition loop
      séquence d'instructions ;
    end loop ;
  - Boucle for:
    for i in 1 to 10 loop
      séquence d'instructions ;
    end loop;
- Utilisation des attributs de signaux:





if (Clock'event and Clock = '1') then if (Clock'event and Clock = '0') then

Exemples de blocs

Soit le décodeurs 2-4 dont la table de vérité est:

Entrèes		Sorties			
A_1	A_0	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Donner le code VHDL en utilisant when ... else

Exemples de blocs

Réponse

```
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder is
  port (a : in std_logic_vector(1 downto 0);
        z : out std_logic_vector(3 downto 0));
end entity decoder;
architecture when_else of decoder is
begin
      z <= "0001" when a = "00" else
           "0010" when a = "01" else
           "0100" when a = "10" else
           "1000" when a = "11" else
           "XXXX";
end architecture when_else;
```

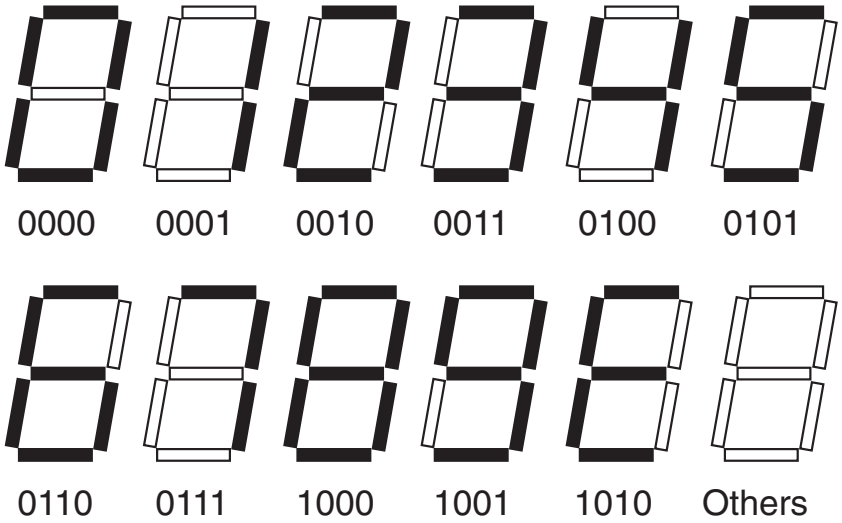
Exemples de blocs

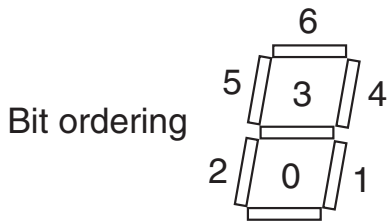
Réponse: Une déclaration alternative à when ... else

```
library IEEE;
use IEEE.std_logic_1164.all;
entity decoder is
  port (a : in std_logic_vector(1 downto 0);
        z : out std_logic_vector(3 downto 0));
end entity decoder;
architecture with_select of decoder is
begin
  with a select
    z ⇐ "0001" when "00",
        "0010" when "01",
        "0100" when "10",
        "1000" when "11",
        "XXXX" when others;
end architecture with_select;
```

Exemples de blocs

Soit l'afficheur 7 ségments (4 entrées) suivant: (led on "1" , led off "0")





- Donner le code VHDL en utilisant with ... select

Exemples de blocs

Réponse:

```
library IEEE;
use IEEE.std_logic_1164.all;
entity seven_seg is
   port (a : in std_logic_vector(3 downto 0);
         z : out std_logic_vector(6 downto 0));
end entity seven_seg;
```

Exemples de blocs

```
architecture with_select of seven_seg is
begin
  with a select
     z ⇐ "1110111" when "0000",
      "0010010" when "0001",
      "1011101" when "0010",
      "1011011" when "0011",
      "0111010" when "0100",
      "1101011" when "0101",
      "1101111" when "0110",
      "1010010" when "0111",
      "1111111" when "1000",
      "1111011" when "1001",
      "1101101" when "1010" | "1011" | "1100" |"1101" | "1110" | "1111",
      "0000000" when others;
end architecture with_select;
```

Exemples de blocs

Entrèes				Sorties		
A_3	A_2	A_1	A_0	Y_1	Y_0	valid
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	-	0	1	1
0	1	-	-	1	0	1
1	-	-	-	1	1	1

Donner le code VHDL respectant la table de vérité ci-dessus en utilisant:

- with ... select
- when ... else

Exemples de blocs

```
library IEEE;
use IEEE.std_logic_1164.all;
entity priority is
port (a: in std_logic_vector(3 downto 0);
y: out std_logic_vector(1 downto 0);
valid: out std_logic);
end entity priority;
```

Exemples de blocs

- with ... select

```
architecture DontCare of priority is
begin
with a select
y ⇐ ”00” when ”0001”,
”01” when ”001-”,
”10” when ”01- -”,
”11” when ”1- - -”,
”00” when others;
valid ⇐ ’1’ when a(0) = ’1’ or a(1) = ’1’ or a(2) = ’1’
or a(3) = ’1’ else ’0’;
end architecture DontCare;
```

Exemples de blocs

- when ... else

```
architecture Ordered of priority is
begin
y ⇐ ”11” when a(3) = ’1’ else
”10” when a(2) = ’1’ else
”01” when a(1) = ’1’ else
”00” when a(0) = ’1’ else
”00”;
valid ⇐ ’1’ when a(0) = ’1’ or a(1) = ’1’ or a(2) = ’1’ or a(3) = ’1’ else
’0’;
end architecture Ordered;
```

Exemples de blocs

- VHDL séquentielle

```
architecture Sequential of priority is
begin
process (a) is
begin
if a(3) = '1' then
y ⇐ "11";
valid ⇐ '1';
elseif a(2) = '1' then
y ⇐ "10";
valid ⇐ '1';
elseif a(1) = '1' then
y ⇐ "01";
valid ⇐ '1';
elseif a(0) = '1' then
y ⇐ "00";
valid ⇐ '1';
else
y ⇐ "00";
valid ⇐ '0';
end if;
end process;
end architecture Sequential;
```

Exemples de blocs

- Développer un modèle VHDL pour la conversion du code Gray (4bits) au binaire normal (4bits)