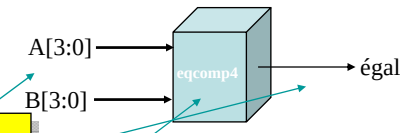
Introduction à VHDL

Daniel Etiemble
de@lri.fr

Introduction à VHDL

- Exemple de conception
  - Code VHDL pour réaliser un comparateur 4 bits

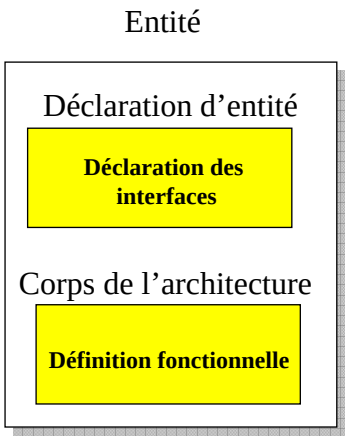



```
-- eqcomp4 est un comparateur 4 bits
entity eqcomp4 is
  port (a, b: in bit_vector(3 downto 0);
        equals : out bit);
end eqcomp4;

architecture dataflow of eqcomp4 is
begin
   equals <= '1' when (a = b) else '0';
end dataflow;
```

NOTE:
1. -- indique un commentaire
2. Deux parties dans le code

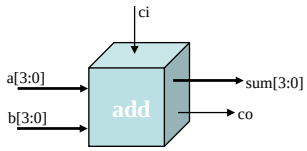
Format VHDL

- Le code VHDL décrit une entité en deux parties
  - Déclaration d'entité
  - Corps de l'architecture
- Il y a aussi d'autres aspects
  - package
  - configuration



Déclaration d'entité

- Description des entrées sorties
  - Comme le schéma d'un composant
  - Description des "ports" d'un composant (les "pattes" d'E/S du schéma)
  - **entity** <name> **is** . . . **end** <name>;
  - EXEMPLE : un additionneur 4 bits



Schéma

```
entity add4 is port(
   a, b:  in std_logic_vector(3 downto 0);
   ci:    in std_logic;
   sum: out std_logic_vector(3 downto 0);
   co:    out std_logic);
end add4;
```

PORTS

- Chaque signal d'E/S d'une entité est appelé port
  - Un port est un objet de type données qui peut recevoir des valeurs et être utilisé dans des expressions
  - Chaque port a un nom, une direction (mode) et un type de données
- Nom de port
  - Majuscules/minuscules sont équivalents
  - Le premier caractère doit être une lettre
  - Le dernier caractère ne peut être un "underscore"
  - Deux "underscores" successifs sont interdits

Modes et types

- Un mode décrit la direction du transfert d'une donnée à travers un port
  - in
  - out
  - inout (signaux bidirectionnels)
  - buffer (un noeud interne utilisé pour feedback)
- Il y a plusieurs types de données disponibles
  - Boolean, bit, bit_vector, integer
  - La bibliothèque standard IEEE fournit également std_ulogic et std_logic et des tableaux de ces deux types
    - On doit signaler à VHDL d'utiliser cette bibliothèque

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

Architecture

- Une architecture décrit le contenu d'une entité
- VHDL a trois styles d'architecture qui peuvent être combinée dans le corps d'une architecture
  - Comportemental
  - Flot de données
  - Structurel
- Le même circuit peut être décrit en utilisant n'importe lequel des trois styles

Description comportementale

```
library ieee;
use ieee.std_logic_1164.all;
entity eqcomp4 is port(
  a, b:      in std_logic_vector(3 downto 0)
  equals:   out std_logic);
end eqcomp4;

architecture behavioral of eqcomp4 is
begin
  comp:  process (a, b)
           begin
             if a = b then
                equals <= '1';
             else
                equals <= '0';
             end if;
           end process comp;
end behavioral;
```

- Une descripton comportementale fournit un algorithme qui modélise le fonctionnement du circuit
- Une déclaration de processus contient un algorithme
  - Elle commence par une étiquette (optionnel), le mot clé "process" et une liste des signaux actifs (*sensivity list*)
  - La liste des signaux actifs indique quels signaux provoqueront l'exécution du processus

Description flot de données

- Une description flot de données spécifie comment la donnée est transférée de signal à signal sans utiliser d'affectations séquentielles (comme dans la description comportementale)
- Cette description ne nécessite pas de déclaration de processus
- Toutes les déclarations s'exécutent en même temps ("*concurrent signal assignment*")

```
architecture dataflow of eqcomp4 is
begin
    equals <= '1' when (a = b) else '0';
end dataflow;
```


Descriptions structurelles

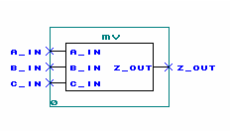
```
library ieee;
use ieee.std_logic_1164.all;
entity eqcomp4 is port(
  a, b:      in std_logic_vector(3 downto 0)
  equals:   out std_logic);
end eqcomp4;

use work.gatespkg.all;
architecture struct of eqcomp4 is
   signal x : std_logic_vector(0 to 3)
begin
   u0:  xnor2 port map (a(0),b(0),x(0));
   u1:  xnor2 port map (a(1),b(1),x(1));
   u2:  xnor2 port map (a(2),b(2),x(2));
   u3:  xnor2 port map (a(3),b(3),x(3));
   u4:  and4 port map (x(0),x(1),x(2),x(3),equals);
end struct;
```

- Les composants sont instanciés et connectés.
  - Ils doivent être définis dans un package et compilés dans une bibliothèque
  - Les bibliogthèques sont attachées par une déclaration

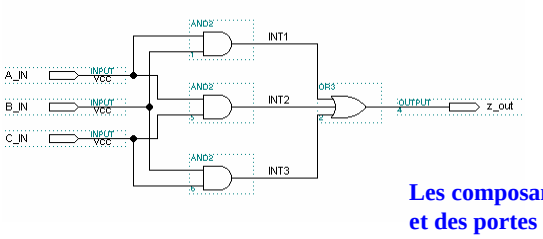
xnor2 and and4 sont dans la bibliothèque work.gatespkg.

Modélisation structurelle en VHDL

- Concevoir une fonction majorité à trois entrées
  - La déclaration d'entité est équivalente à



  - La déclaration d'architecture est équivalente à



Les composants sont des portes ET et des portes OU

VHDL Code

- Aspect du code VHDL :
  - L'architecture a trois déclarations
    - Deux composants
    - Un signal
  - Les déclarations de composants correspondent à leur entité
  - Les déclarations de signaux créent un ou plusieurs signaux internes

```
entity MAJORITY is port(
   A_IN, B_IN, C_IN: in BIT;
   Z_OUT             : out BIT);
end MAJORITY;


architecture struct of MAJORITY is
--Declare logic operators
component AND2_OP
  port (A, B : in BIT; z : out BIT);
end component;
component OR3_OP
  port (A, B, C : in BIT; z : out BIT);
end component;
--Declare signals to interconnect logic operators
 signal INT1, INT2, INT3 : BIT;
begin
    A1:  AND2_OP port map (A_IN,B_IN,INT1);
    A2:  AND2_OP port map (A_IN,C_IN,INT2);
    A3:  AND2_OP port map (B_IN,C_IN,INT3);
    A4:  OR3_OP port map (INT1,INT2,INT3,Z_OUT);
end struct;
```

Déclaration de composants

- Les composants nécessaires sont aussi décrits avec un style VHDL

```
entity AND2_OP is
  port
    (A, B : in BIT;
     Z    : out BIT);
end AND2_OP;

architecture MODEL of AND2_OP is
begin
  Z <= A and B;
end MODEL;
```

```
entity OR3_OP is
  port
    (A, B, C : in BIT;
     Z       : out BIT);
end OR3_OP;

architecture MODEL of OR3_OP is
begin
  Z <= A or B or C;
end MODEL;
```

Ici modélisation "flot de données"

Instantiation des composants

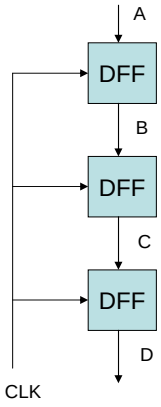
- Le circuit Majorité contient quatre déclarations d'instantiation de composants
  - Elles commencent par une étiquette suivie de :
  - Puis un nom de composant
  - "port map" décrit comment chaque composant est connecté au reste du système.
- Ce sont des déclarations concurrentes : l'ordre n'a pas d'importance
  - Elles sont toutes exécutées en même temps

```
begin
   A1:  AND2_OP port map (A_IN,B_IN,INT1);
   A2:  AND2_OP port map (A_IN,C_IN,INT2);
   A3:  AND2_OP port map (B_IN,C_IN,INT3);
   A4:  OR3_OP port map (INT1,INT2,INT3,Z_OUT);
end struct;
```

Logique séquentielle

- Passage d'un état à l'état suivant sur une transition d'horloge

```
label: process (CLK)
        begin
                if(rising_edge(CLK)) then
                        D       <=      C;
                        B       <=      A;
                        C       <=      B;
                end if;--CLK
        end process;--main
```



Structures hiérarchiques

- Il n'y a pas de restrictions sur la hiérarchisation des modèles structurels
  - Les entités d'une réalisation peuvent utiliser des composants dont les entités à leur tour peuvent utiliser des composants

EXEMPLE : un circuit avec deux circuits majorité qui détecte quand deux groupes de trois entrées ont tous deux un signal de sortie haut.

```
entity MAJORITY_2x3 is port(
   A1,B1, C1    : in BIT;
   A2, B2, C2   : in Bit;
   Z_OUT        : out BIT);
end MAJORITY_2x3;

architecture struct of MAJORITY_2x3 is
component AND2_OP
  port (A, B : in BIT; z : out BIT);
end component;
component MAJORITY
  port (A_IN,B_IN,C_IN : in BIT
        Z_OUT          : out BIT);
end component;
 signal INT1, INT2: BIT;
begin
   M1:  MAJORITY port map (A1,B1,C1,INT1);
   M2:  MAJORITY port map (A2,B2,C2,INT2);
   A1:  AND2_OP port map (INT1,INT2,Z_OUT);
end struct;
```

PACKAGES

```
package LOGIC_OPS is
component AND2_OP
  port (A, B : in BIT; z : out BIT);
end component;
component OR32_OP
  port (A, B, C : in BIT; z : out BIT);
end component;
component NOT_OP
  port (A, : in BIT; A_BAR : out BIT);
end component;
end LOGIC_OPS;
```

- Un package sert à réutiliser un ensemble de déclarations de composants
  - Un package commence par le mot clé *package* et se termine par le mot clé *end*
  - Il contient les définitions de composants et de signaux (et plus…)
- Ce package pourrait être sauvegardé dans la bibliothèque WORK et appelé par
  - uses WORK.LOGIC_OPS.all
- Ou il pourrait être inclus dans le fichier source qui contient toutes les entités pour un problème

Bibliothèques

- Une bibliothèque contient du code compilé d'entités de conception
  - Il y a une bibliothèque standard VHDL
  - Lors de la compilation d'un projet, il est rangé par défaut dans la bibliothèque WORK
- L'utilisation d'une bibliothèque implique la déclaration

  library <nom_bibliothèque>;
  - Pour accéder à ses packages:

    use <nom_bibliothèque>.<nom_bibliothèque>.*all*;

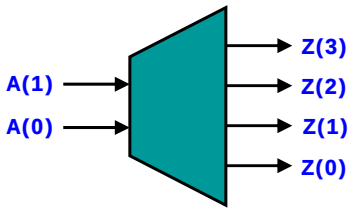
Exemple : décodeur

Exemple: décodeur 2 : 4

```
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;
```

Interface

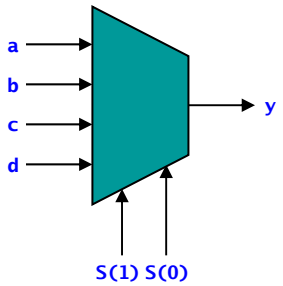
Fonction



A(1..0)		Z(3..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

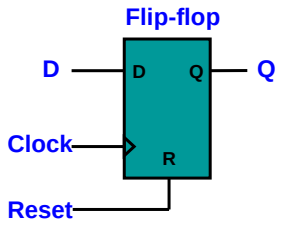
Exemple : multiplexeur 4:1

```
entity mux is
  port (
    a, b, c, d: in std_logic;
    s: in  std_logic_vector(1 downto
   0);
    y: out std_logic);
end entity mux;


architecture mux1 of mux is
begin
   process (a, b, c, d, s)
   begin
     case s is
       when "00“ => y <= a;
       when "01" => y <= b;
       when "10" => y <= c;
       when "11" => y <= d;
     end case;
   end process;
end architecture mux1;
```



Bascule D

```
architecture rtl of D_FF is
begin
  process (Clock, Reset) is
  begin
    if Reset = '1' then
      Q <= '0';
    if rising_edge(Clock)
   then
      Q <= D;
    end if;
  end process;
end architecture rtl;
```



Compteur binaire

```
entity counter is
  generic (n : integer := 4);
  port (
    clk  : in  std_logic;
    reset: in  std_logic;
    count: out std_logic_vector(n-1 downto
   0)
  );
end entity counter;
```

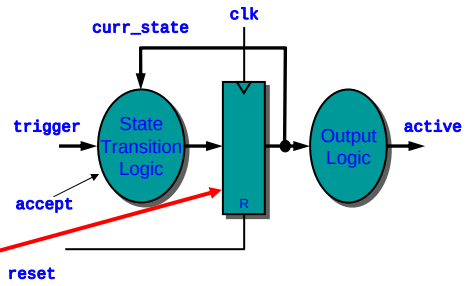
- Pas de détails sur la manière de réaliser l'opérateur.
- Le + indique l'opération d'incrémentation.

```
use ieee.numeric_std.all;

architecture binary of counter is
begin
  process (clk, reset)
    variable cnt : unsigned(n-1 downto
   0);
  begin
    if reset = '1' then   -- async reset
      cnt := (others => '0');
    elsif rising_edge(clk) then
      cnt := cnt + 1;
    end if;
    count <= std_logic_vector(cnt);
  end process;
end architecture binary;
```

Automate (FSM)

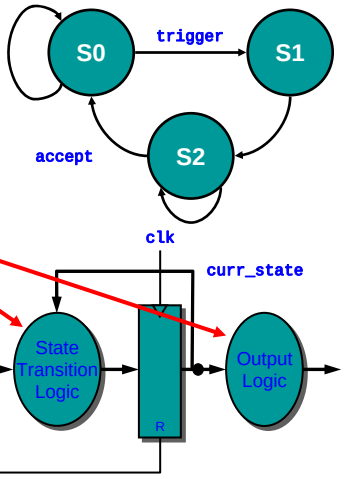
Sur la réception d'un signal (trigger), génère un signal active de 2 cycles et attend un signal accept

```
entity trigger is
  port (
    clk, reset:       in
   std_logic;
    trigger, accept : in
   std_logic;
    active:           out
   std_logic);
end entity trigger;

architecture rtl of trigger is
  type state_type is (s0, s1, s2);
  signal cur_state,
         next_state: state_type;
begin
  registers: process (clk, reset)
  begin
    if (reset='1') then
      cur_state <= s0;
    elsif rising_edge(clk) then
      cur_state <= next_state;
    end if;
  end process;
```



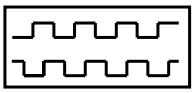
Automate (suite)

```
process (cur_state, trigger, accept) is
begin
  case cur_state is
    when s0 =>
      active <= '0';
      if (trigger = '1') then
        next_state <= s1;
      else
        next_state <= s0;
      end if;
    when s1 =>
      active <= '1';
      next_state <= s2;
    when s2 =>
      active <= '1';
      if (accept = '1') then
        next_state <= s0;
      else
        next_state <= s2;
      end if;
  end case;
end process;
```



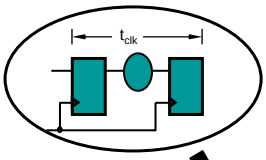
Flot de conception FPGA (1)

Réalisation
Description comportementale ou structurelle

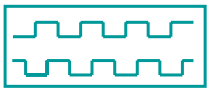


Simulation RTL
• Simulation fonctionnelle
•Vérifie le fonctionnement logique et le flot de données (pas d'analyse temporelle)

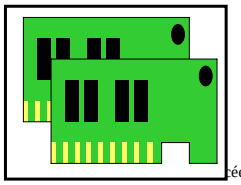
Flot de conception FPGA (2)



Analyse temporelle
- Vérifie que les performances requises sont satisfaites
- Analyse temporelle statique



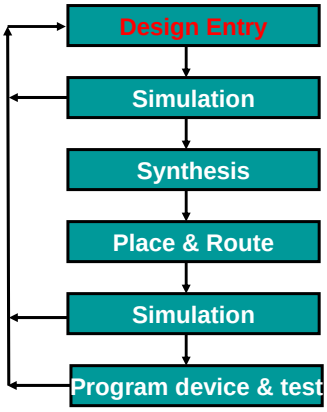
Simulation au niveau portes
- Simulation temporelle
- Vérifie que la réalisation fonctionne sur la technologie cible



Programmation et test
- Programmation & et test du composant sur la carte

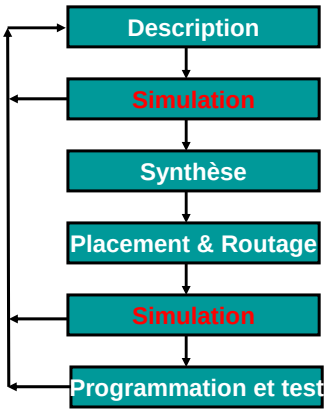
Cycle de conception : description

- Description du système
  - Textuelle (HDL)
    - VHDL, Verilog…
  - Graphique
    - Schéma bloc
    - Graphe d'états
    - Table de vérité
    - …



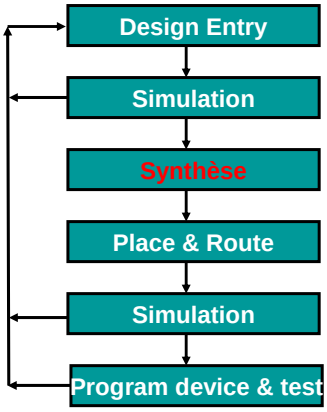
Cycle de conception : simulation

- Simulation fonctionnelle:
  - Indépendante du type de FPGA
  - Pas d'information temporelle
- Simulation temporelle
  - Simulation après placement et routage
  - "timing" détaillé



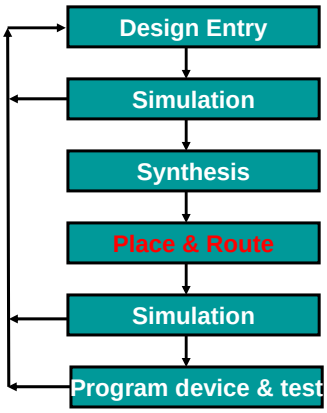
Synthèse RTL

- L'entrée est une description HDL
- Compilation & traduction
  - Génère la liste de noeuds indépendante de la technologie
  - Schéma TRL (analyse du code HDL)
- Mappage sur la technologie
  - Mappage sur les structures spécifiques de la technologie
    - LUTs
    - Registres
    - TAM/ROM
    - Blocs DSP
    - Autres composants
- Optimisation logique
  - Analyse de l'implantation



Placement et routage

- "fitter" sur le FPGA
  - Logiciel fourni par le vendeur du FPGA, spécifique de l'architecture de chaque composant FPGA
- Fonctions
  - Placement et routage
  - Edition des contraintes
  - Annotation de la liste de noeuds avec les informations temporelles
  - Fichier de configuration



Exemple : Quartus II d'Altera

- Logiciel de conception intégré
  - Plusieurs techniques d'entrée
    - textuelles: VHDL, Verilog, AHDL
    - Editeur de schéma
  - Synthèse logique
  - Placement et routage
  - Simulation
  - Analyse temporelle et puissance dissipée
  - Création de la "netlist" pour l'analyse temporelle
  - Programmation du composant
- ISE (Xilinx) a des caractéristiques semblables