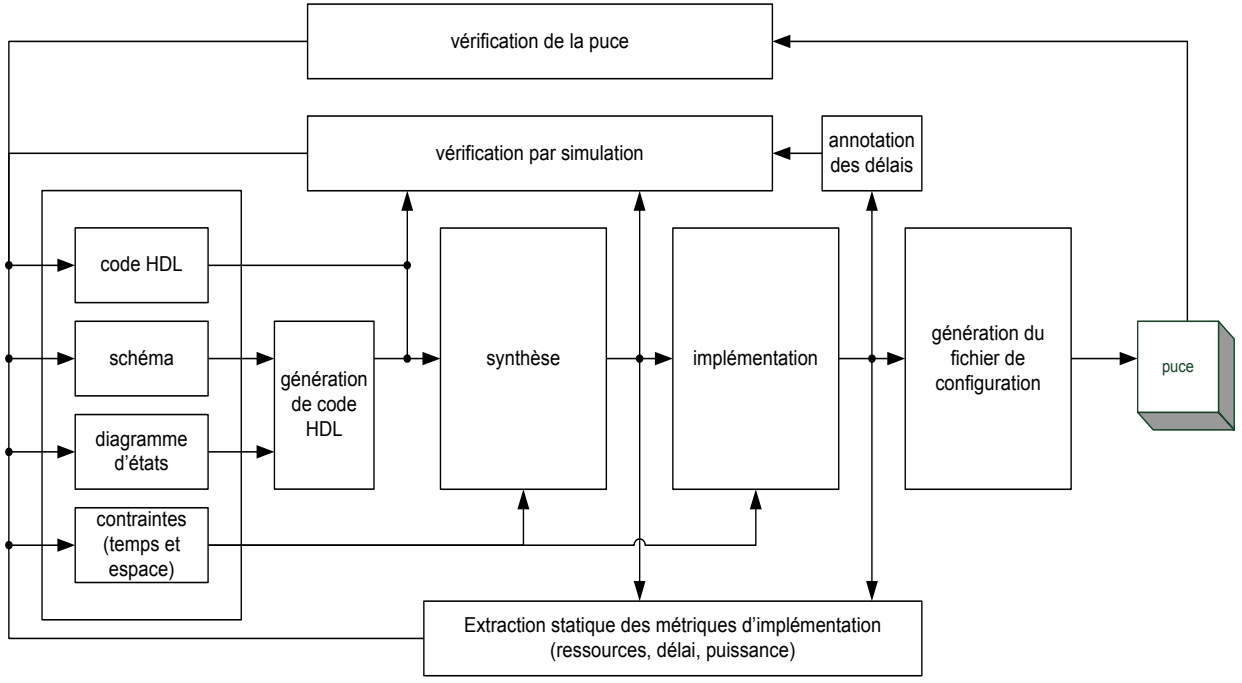
Synthèse d'un modèle VHDL

Simulation d’un modèle VHDL
Sujets de ce thème

- Le processus de synthèse
- Norme 1076.6 et documentation
- Types, opérateurs, boucles
- Éléments à mémoire
- Exemples
- Recommandations

Synthèse de code VHDL

- La synthèse du code VHDL est effectuée par un synthétiseur.
- Le processus de synthèse peut être décomposé et effectué en plusieurs passes. Ce processus est très complexe sauf pour les circuits les plus simples.
- Le produit du synthétiseur est communément appelé «liste des interconnexions» (*netlist*): les composantes de base et les liens qui les relient.
- Après le processus de synthèse, il est possible d'obtenir un estimé de la performance et des coûts du circuit.



La norme IEEE 1076.6-2004

- IEEE Standard for VHDL Register Transfer Level (RTL) Synthesis
- *"This document specifies a standard for use of VHDL to model synthesizable register-transfer level digital logic.*
  A standard syntax and semantics for VHDL register-transfer level synthesis is defined.
  The subset of the VHDL language, which is synthesizable, is described, and nonsynthesizable VHDL constructs are identified that should be ignored or flagged as errors."
- http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1342563&isnumber=29580

6.1.3.1 Edge-sensitive storage from a process with sensitivity list and one clock

Edge-sensitive storage shall be modeled for a signal or variable assigned inside a process with sensitivity list when all of the following apply:

a) The signal or variable has a <sync_assignment>.

b) There is no execution path in which the value update from a <sync_assignment> overrides the value update from an <async_assignment> unless the <async_assignment> is an assignment to itself.

c) It is possible to statically enumerate all execution paths to the signal or variable assignments.

d) The process sensitivity list includes the clock and any signal controlling an <async_assignment>.

e) The <clock_edge> is present in the conditions only, and the <clock_edge> always expresses the same edge of the same clock signal.

f) For a variable, the value written by a given clock edge is read during a subsequent clock edge.

Example 1: Storage may be assigned in multiple statements in a process.

```
TwoReg : process(clk)
  begin
  if rising_edge(clk) then
    Q1 <= D1;
    Q2 <= D2;
  end if;
end process;
```

La documentation de XST

- « The *XST User Guide:*
  - Describes Xilinx® Synthesis Technology (XST) support for Hardware Description Language (HDL), Xilinx devices, and design constraints for the Xilinx ISE® Design Suite software
  - Discusses FPGA and CPLD optimization and coding techniques when creating designs for use with XST
- Le document fournit beaucoup d'exemples sur la façon de coder différents modules matériels.

About Accumulators

An *accumulator* differs from a *counter* in the nature of the operands of the **add** and **subtract** operation.

- In a *counter*:
  - The destination and first operand is a signal or variable
  - The second operand is a constant equal to **1**:

```
A <= A + 1
```

- In an *accumulator*:
  - The destination and first operand is a signal or variable
  - The second operand is either:
    - A signal or variable:

```
A <= A + B
```

    - A constant not equal to **1**:

```
A <= A + Constant
```

An inferred accumulator can be **up**, **down**, or **updown**. For an **updown** accumulator, the accumulated data may differ between the **up** and **down** mode:

```
...
if updown = '1' then
  a <= a + b;
else
  a <= a - c;
...
```

XST can infer an accumulator with the same set of control signals available for counters.

La documentation de XST

- « The *XST User Guide:*
  - Describes Xilinx® Synthesis Technology (XST) support for Hardware Description Language (HDL), Xilinx devices, and design constraints for the Xilinx ISE® Design Suite software
  - Discusses FPGA and CPLD optimization and coding techniques when creating designs for use with XST »
- Le document fournit beaucoup d'exemples sur la façon de coder différents modules matériels.

```
-- 4-bit Unsigned Up Accumulator with Asynchronous Reset
--
library ieee;
use ieee.std_logic_1164.all; use ieee.std_logic_unsigned.all;

entity accumulators_1 is
   port(C, CLR : in std_logic;
   D : in std_logic_vector(3 downto 0);
   Q : out std_logic_vector(3 downto 0));
end accumulators_1;

architecture archi of accumulators_1 is
signal tmp: std_logic_vector(3 downto 0);
begin
   process (C, CLR)
   begin
      if (CLR='1') then
            tmp <= "0000";
      elsif (C'event and C='1') then
            tmp <= tmp + D;
      end if;
   end process;
   Q <= tmp;
end archi;
```

Types utilisés pour la synthèse

- Les types `std_logic` ou `std_logic_vector` ne devraient être utilisés que pour des valeurs logiques.
- Pour représenter des nombres pour les ports d'une entité, les types préférés sont tirés du package `numeric_std` :
  - `unsigned` pour une interprétation non signée
  - `signed` pour une interprétation signée en complément à deux.

- Pour représenter des signaux internes d'un module, on a avantage à utiliser les types plus abstraits comme `integer`, `character` ou `string`, même s'il est plus difficile de retrouver ces signaux pour débogage dans les produits de la synthèse ou de l'implémentation.
- Pour les constantes et les paramètres d'un module, on peut en général utiliser tous les types abstraits supportés par VHDL, comme `real`, `integer`, ou `string`.

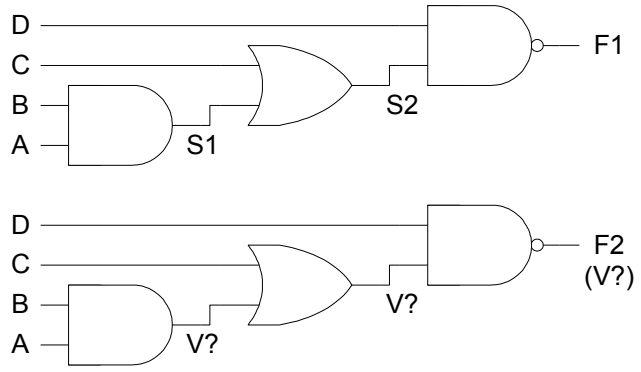
Exemple: code utilisant les types `character` et `string`

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

entity banderolev2 is
 port (
  reset, clk : in  STD_LOGIC;
  charout : out unsigned(7 downto 0)
 );
end banderolev2;
architecture arch of banderolev2 is
constant message : string := "bonjour et bienvenue!";
begin
process (clk, reset)
variable index : integer range 1 to message'length;
begin
 if (rising_edge(clk)) then
  if reset = '1' then
   index := 1;
  else
   charout <= to_unsigned(character'pos(message(index)), 8);
   if index = message'length then
    index := 1;
   else
    index := index + 1;
   end if;
  end if;
 end if;
end process;
end arch;
```

```
=========================================================================
Advanced HDL Synthesis Report

Macro Statistics
# ROMs                                                  : 1
 21x8-bit ROM                                          : 1
# Counters                                              : 1
 5-bit up counter                                      : 1
# Registers                                             : 8
 Flip-Flops                                            : 8

=========================================================================

=========================================================================
*                            Low Level Synthesis                        *
=========================================================================
WARNING:Xst:1710 - FF/Latch <charout_7> (without init value) has a constant
value of 0 in block <banderolev2>. This FF/Latch will be trimmed during the
optimization process.

Optimizing unit <banderolev2> ...

Mapping all equations...
Building and optimizing final netlist ...
Found area constraint ratio of 100 (+ 5) on block banderolev2, actual ratio is
0.

Final Macro Processing ...

=========================================================================
Final Register Report

Macro Statistics
# Registers                                             : 12
 Flip-Flops                                            : 12
```

Fils modélisés par les catégories signal et variable

- En général, l'utilisation de la catégorie signal résulte en un fil concret dans un module.
- C'est moins souvent le cas pour la catégorie variable, à cause du traitement différent de ces deux catégories.




```
library ieee;
use IEEE.STD_LOGIC_1164.ALL;
entity demoSignalVariable is
  port (
    A, B, C, D: in std_logic;
    F1, F2 : out std_logic
  );
end demoSignalVariable;
architecture demo of demoSignalVariable is
signal S1, S2 : std_logic;
begin

  S1 <= A and B;
  S2 <= S1 or C;
  F1 <= S2 nand D;

  process(A, B, C, D)
  variable V : std_logic;
  begin
    V := A and B;
    V := V or C;
    V := V nand D;
    F2 <= V;
  end process;

end demo;
```

Opérateurs synthétisables

- La plupart des synthétiseurs peuvent synthétiser des circuits matériels qui réalisent les opérateurs suivants:
  - logique: and, or, nand, nor, xor, xnor, not
  - relation: =, /=, <, <=, >, >=
  - concaténation: &
  - arithmétique et décalage:
    - +, - ,*, abs
    - /, rem et mod, si l'opérande de droite est une constante égale à une puissance de 2
    - sll, srl, sla, sra, rol, ror
- Les opérateurs ne sont pas définis pour tous les types, il faut utiliser les librairies correspondantes.

Synthèse: boucles et conditions

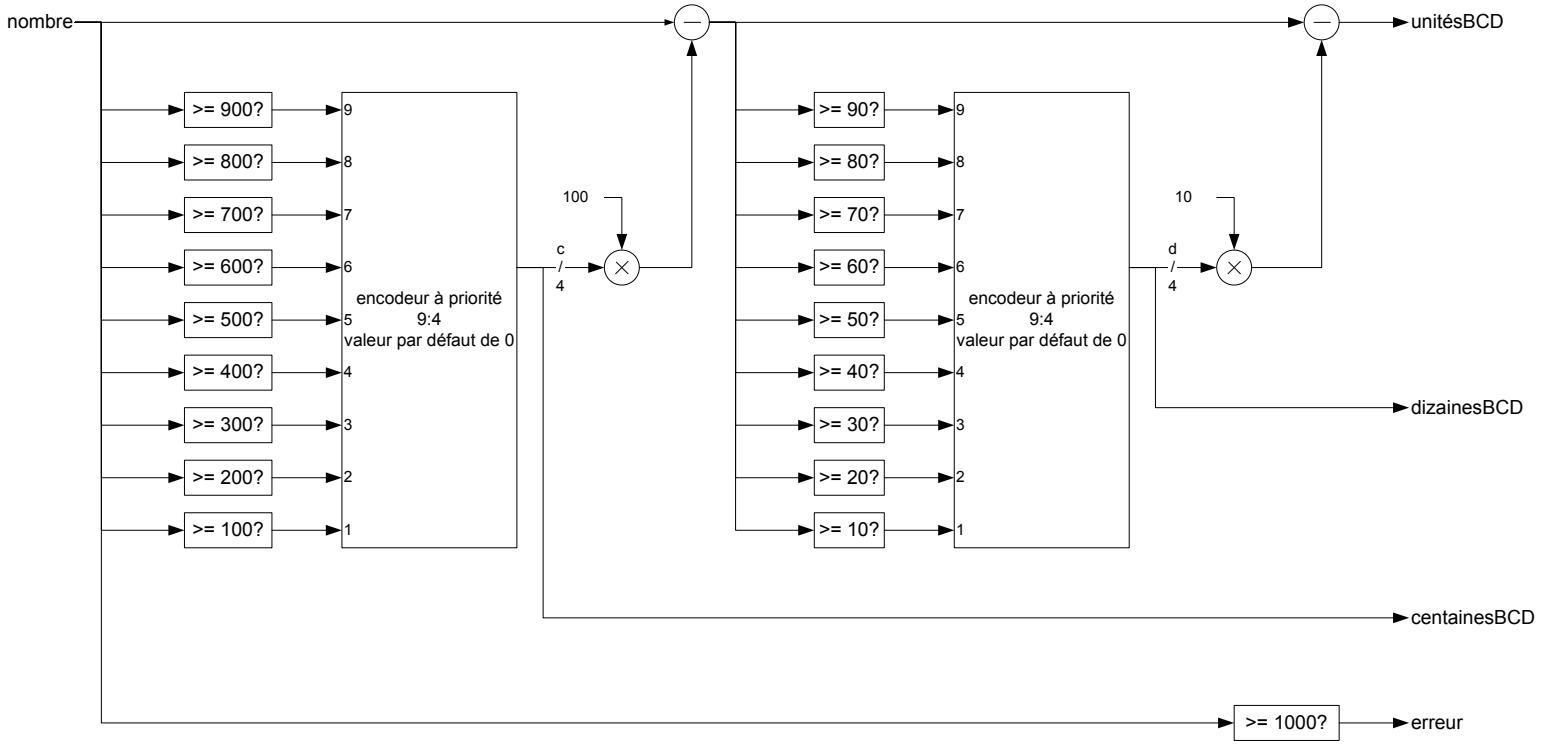
- À l'intérieur d'un processus, on peut utiliser des boucles et des conditions pour modéliser le comportement d'un circuit.
- Les boucles sont une manière compacte de représenter plusieurs énoncés reliés logiquement entre eux.
- Les paramètres d'exécution de la boucle doivent prendre des valeurs statiques au moment de la synthèse.
- Les boucles sont implémentées en les déroulant: les énoncés d'assignation qu'elles contiennent sont répliqués, un pour chaque itération de la boucle.

- Pour les circuits combinatoires, les conditions permettent d'effectuer un choix.
  - L'énoncé `case` a l'avantage de représenter des choix qui sont mutuellement exclusifs et qui ont la même préséance. Il correspond assez exactement à l'action d'un multiplexeur.
  - L'énoncé `if`, est plus général avec les clauses `elsif` ainsi qu'une clause `else`. Il est possible de l'utiliser pour donner préséance à certaines conditions par rapport à d'autres. Cela peut résulter en un circuit plus complexe que nécessaire, parce que le comportement décrit peut être plus restrictif que ce que le concepteur a en tête.

`Case` est un meilleur choix quand on n'a pas besoin de priorité!

Exemple de code synthétisable utilisant des boucles et des conditions avec priorité

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

entity unsigned2dec is
    port(
        nombre : in unsigned(9 downto 0);
        centainesBCD, dizainesBCD, unitesBCD : out unsigned(3 downto 0);
        erreur : out std_logic
    );
end unsigned2dec;

architecture arch of unsigned2dec is
begin
    erreur <= '1' when nombre >= 1000 else '0';
    process(nombre)
    variable n, c, d, u : natural := 0;
    begin
        n := to_integer(nombre);
        c := 0;
        for centaines in 9 downto 1 loop
            if n >= centaines * 100 then
                c := centaines;
                exit;
            end if;
        end loop;
        n := n - c * 100;
        d := 0;
        for dizaines in 9 downto 1 loop
            if n >= dizaines * 10 then
                d := dizaines;
                exit;
            end if;
        end loop;
        u := n - d * 10;
        centainesBCD <= to_unsigned(c, 4);
        dizainesBCD <= to_unsigned(d, 4);
        unitesBCD <= to_unsigned(u, 4);
    end process;
end arch;
```



Inférence d'éléments à mémoire

- Le processus de synthèse repose sur le principe de l'inférence de composantes matérielles à partir d'une description en code.
- Il est important de vérifier la documentation d'un outil de synthèse pour savoir quelle structure de langage utiliser pour obtenir le circuit désiré.
- À l'intérieur d'un processus, un élément à mémoire est inféré en VHDL si un objet des catégories signal ou variable se voit assigner une valeur dans un énoncé `if-else`, et que certains cas ne sont pas couverts.

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

entity mystere1 is
   port (a, b, c: in std_logic;
      s : in std_logic_vector (1 downto 0);
      o : out std_logic);
end mystere1;

architecture archi of mystere1 is
begin
   process (a, b, c, s)
   begin
      if (s = "00") then o <= a;
      elsif (s = "01") then o <= b;
      elsif (s = "10") then o <= c;
      end if;
   end process;
end archi;
```

Exemples de code ambigu ou non synthétisable - 1

```
entity boucledynamique is
   port (
   x0, xmax : in integer range 0 to 255;
   somme : out integer range 0 to 65535
);
end boucledynamique;

architecture arch of boucledynamique is

begin

   process(x0, xmax)
   variable sommet : integer;
   begin
      sommet := 0;
      for k in x0 to xmax loop
            sommet := sommet + k;
      end loop;
      somme <= sommet;
   end process;

end arch;
```

Boucle à bornes indéfinies.

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

entity registremanque1 is
   port (
   clk, D : in std_logic;
   Q : out std_logic
);
end registremanque1;

architecture arch of registremanque1 is
begin
   process(clk)
   begin
      if (rising_edge(CLK) or falling_edge(CLK)) then
            Q <= D;
      end if;
   end process;
end arch;
```

Ne respecte pas les patrons pour la description de registres.

Exemples de code ambigu ou non synthétisable - 2

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

entity deuxsourcesetpire is
   port (
   A, B, C, D, CLK : in std_logic;
   F : out std_logic
);
end deuxsourcesetpire;

architecture arch of deuxsourcesetpire is
begin
   F <= CLK and (C or D);
   process(CLK)
   begin
      if (rising_edge(CLK)) then
            F <= A or B;
      end if;
   end process;
end arch;
```

Deux sources pour le signal F
Utilisation du signal d'horloge dans une expression (synthétisable, mais …)

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

entity registremanque is
   port (
   clk, reset, enable, D : in std_logic;
   Q : out std_logic
);
end registremanque;

architecture arch of registremanque is
begin
   process(clk, reset)
   begin
      if (rising_edge(CLK) and reset = '0' and enable = '1') then
            Q <= D;
      end if;
   end process;
end arch;
```

Ne respecte pas les patrons pour la description de registres.

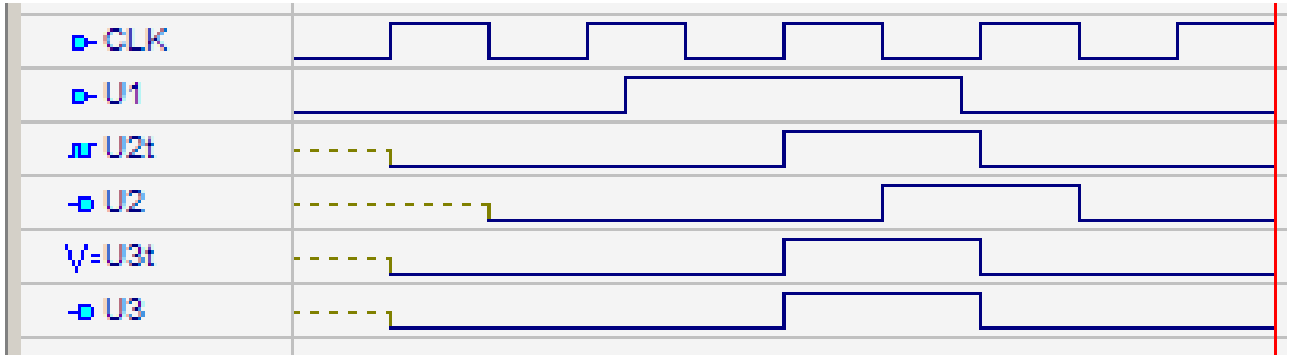
Résultats différents pour la synthèse et la simulation
Comment les éviter

- Un circuit décrit en VHDL peut être parfaitement simulable et pas du tout synthétisable.
- Parfois, des erreurs dans la description du circuit ne sont découvertes que lors du processus de synthèse.
- Attention à la liste de sensitivité
  - l'absence d'un signal dans la liste de sensitivité est importante pour le simulateur
  - le synthétiseur suppose (en général) que le concepteur a fait une erreur:
    "*Declare asynchronous signals in the sensitivity list. Otherwise, XST issues a warning and adds them to the sensitivity list. In this case, the behavior of the synthesis result may be different from the initial specification.*" – XST User Guide, v. 11.1.0, Apr. 2009

- Quoi faire?
  - Surveiller les avertissements du synthétiseur.
  - Pour les processus décrivant des bascules: placer seulement `clk` et `reset` dans la liste de sensitivité
  - Pour les processus décrivant de la logique combinatoire: placer tous les signaux faisant partie d'expressions dans la liste de sensitivité

Résultats différents pour la synthèse et la simulation
Exemple

```
   signal U2t : std_logic;
-- ...

   process (clk)
   begin
      if rising_edge(clk) then
         U2t <= U1;
      end if;
      U2 <= U2t;
   end process;

   process (clk)
   variable U3t : std_logic;
   begin
      if rising_edge(clk) then
         U3t := U1;
      end if;
      U3 <= U3t;
   end process;
```



Pour le signal U2, lors de la simulation, l'assignation de valeur se fait sur un front descendant d'horloge, ce qui *ne* correspond *absolument pas* au produit de la synthèse.

Synthèse: mot de la fin

- Le synthétiseur respecte un contrat, p. ex. la norme IEEE 1076.6-2004.
- Les concepteurs de synthétiseurs sont des gens prudents. Les synthétiseurs n'infèrent que des modules qui peuvent être décrits sans ambiguïté.
  - le code HDL doit être non ambigu;
  - le code HDL doit correspondre à une structure disponible sur la technologie ciblée, exemples de cas à considérer:
    - registre avec reset synchrone/asynchrone?
    - registre actif sur un ou deux fronts d'horloge?
    - type d'additionneur?
    - division, reste, autres fonctions (sinus, log, tanh, médiane, etc.)?
- Pour écrire du « bon code » synthétisable, il faut bien connaître:
  - le contrat respecté par le synthétiseur utilisé; et,
  - la technologie ciblée.
- Le défi du concepteur de circuits numériques est de décrire ses intentions d'une façon non ambigüe pour le synthétiseur.
- Une approche qui fonctionne consiste à tout d'abord décomposer le circuit en blocs de base correspondant à des composantes logiques connues, puis à produire une description en fonction de ces blocs.

Vous devriez maintenant être capable de …

- Expliquer comment fonctionne le processus de synthèse d'un modèle. (B2)
- Expliquer dans quels cas un modèle VHDL n'est pas synthétisable. (B2)
- Expliquer la relation entre la norme IEEE 1076.6, les structures de langage supportées par les synthétiseurs, et l'ensemble du langage VHDL. (B2)
- Analyser un modèle VHDL et déterminer s'il est synthétisable ou non. (B3)
- Écrire du code synthétisable en exploitant correctement les types, opérateurs et structures de contrôle de VHDL. (B3)

Code	Niveau (http://fr.wikipedia.org/wiki/Taxonomie_de_Bloom)
B1	Connaissance – mémoriser de l'information.
B2	Compréhension – interpréter l'information.
B3	Application – confronter les connaissances à des cas pratiques simples.
B4	Analyse – décomposer un problème, cas pratiques plus complexes.
B5	Synthèse – expression personnelle, cas pratiques plus complexes.