Design re-use

Etienne Messerli
Professeur institut REDS, HEIG-VD
etienne.messerli@heig-vd.ch
http://www.reds.ch

21 septembre 2009

Pourquoi « Design re-use » ?

- Explosion des FPGAs de dernières générations
- Forte demande pour le traitement rapide d'informations
  - ✓Quantité de données de plus en plus important (Go)
  - ✓Débit de transfert de plus en plus rapide (Gbits/s!)
  - ✓Applications temps réel:
    Voice over IP, vidéo, télévision, ...
  - ✓Utilisation de réseau public : Internet
    →Confidentialité des données => cryptographie
  - ✓...

D'où la nécessité de:

- Concevoir des systèmes de plus complexes
- Intégrer ceux-ci dans des PLDs
- Garantir le fonctionnement correct du système
- Garantir l'évolution des systèmes réalisés
- Réduire le temps de la conception à la réalisation : *Time to market*
- Réutilisation indispensable : *Design re-use*
- Nouveau défi : *Vérification*

Design re-use

- Description lisible (viser la simplicité)
- Description facilement ré-utilisable (modifiable)
- Conception et réalisation bien documentée (important : description commentée)
- Fonctionnement fiable
- Synthèse automatique doit être garantie pour tous les outils EDA du marché

Contenu de la présentation ...

- Evolution des méthodologies de conception
- Rappel sur le VHDL pour la synthèse automatique
  - ✓Méthodologie pour description fiable
- Descriptions paramétrables
  - ✓taille définie par les déclarations dans l'entité
  - ✓taille des vecteurs définie dans un paquetage
  - ✓utilisation de constantes génériques (*generic*)
  - ✓utilisation de vecteurs non contraints
- Annexes

Module Master ReCo

Design re-use

Évolution des méthodologies de conception

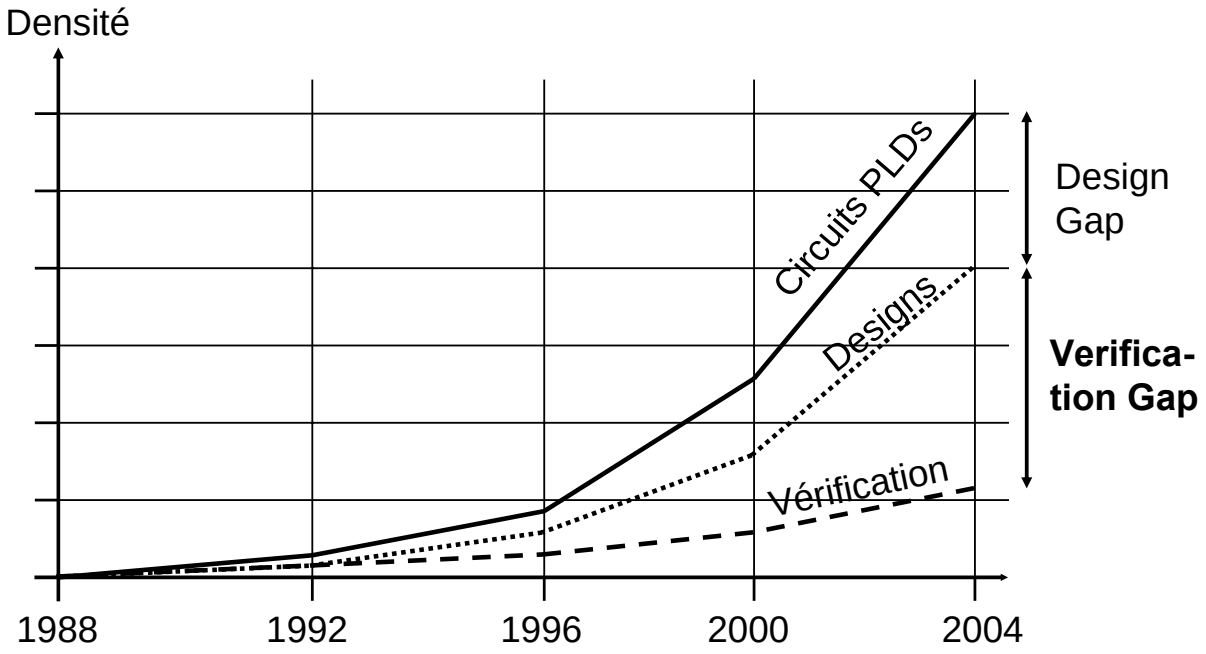
Complexité et coûts...

- Nouveau développement:
  - ✓PLD indispensable (CPLD, FPGA, HardCopy, ASIC)
- Niveau de complexité augmente rapidement :
  - ✓dizaines vers centaines de milliers de portes, voir millions de portes
- Développement "à la main" trop difficile et trop coûteux :
  outils informatiques performants et bibliothèques de "pièces" sont indispensables

Méthodologie de conception

- Conception *Top - Down*
- Décomposition du système
  - ✓choisir l'architecture
  - ✓modules de bases doivent être maîtrisable
  - ✓réutilisation des descriptions
- Conception *full synchrone*
  - ✓fiabilité, testabilité
- Vérification de chaque module

Défi des PLDs …



… défi des PLDs

- La capacité disponible des circuits progresse plus vite que les designs réalisés.
- Raccourcir les temps de développement :
  - ✓langage de haut niveau : description fiable et efficace
  - ✓outils moderne et performant
  - ✓réutiliser les descriptions
  - ✓Utilisation de bloc IP (Intellectual Property)
- Garantir la vérification des designs

Vérification : le nouveau défi

- Dépannage d'un PLD programmé :
  très coûteux voir impossible
- Simulation indispensable,
  nécessite : outils performants
  fichiers de simulation

Thème de la seconde partie : la vérification

Besoin représentations standardisées pour:

- Communiquer, spécifier
- Utiliser des outils de simulation
- Disposer de descriptions :
  lisibles, performantes et hiérarchiques
- Assurer la pérennité des descriptions
- Utiliser des outils d'aide à la réalisation : synthèse (génération automatique d'un schéma), placement (des composants) et routage (des fils)

La portabilité

La portabilité est le maître mot. Dans le monde des circuits ce mot a un double sens :

- portabilité vis-à-vis des circuits.
- portabilité vis-à-vis des outils.

Citation de [2] .

Solution :

Nouvelles méthodes de conception associées avec un langage de haut niveau

soit :

VHDL (préféré en Europe)

ou Verilog (préféré aux USA)

\+ Outils de développement modernes

Bricolages interdits !!

Etablissement d'un standard

- 1993 révision norme 1076, VHDL93
- 1993 paquetage Std_Logic_1164
- 1997 paquetages arithmétiques
  Numeric_bit & Numeric_Std
- 1999 sous-ensemble synthétisable,
  norme 1076.6

Synthèse : standard établit dès 2000

Norme IEEE-1076 : 2008

- Evolution importante du langage
  - ✓ Norme publiée
  - ✓ Livre "VHDL 2008"
- Pas d'outils EDA mis à jour avec cette nouvelle norme
- Arrivée trop tardive de la norme IEEE-1076 2008

Pourquoi utiliser le VHDL ?

- Normalisé (pérennité)
- Moderne, lisible et puissant
- Permet la réutilisation de bloc (IP)
- Disponible avec beaucoup d'outils
  - ✓outils à tous les prix de 0.- à ….
- Compatible avec le domaine ASIC

Langage incontournable en synthèse

Les avantages du langage VHDL

- Langage unique : description, simulation, ...
- Indépendant vis-à-vis du circuit cible
- Indépendant vis-à-vis des outils
  - ⚠sous certaines conditions pour ces 2 points
- Langage hiérarchique : *Top - Down*
- Raccourci le temps de conception :
  - ✓ *time to market*
- Augmente la productivité (description)

Niveaux de description en VHDL

- Haut niveau comportemental
  - ✓non synthétisable actuellement
- RTL Register Transfert Level
  - ✓synthétisable
- Netlist de primitive
  - ✓résultat obtenu après synthèse
- niveau porte
  - ✓résultat obtenu après placement-routage

Module Master ReCo

Design re-use

Rappel sur le VHDL
pour la synthèse automatique

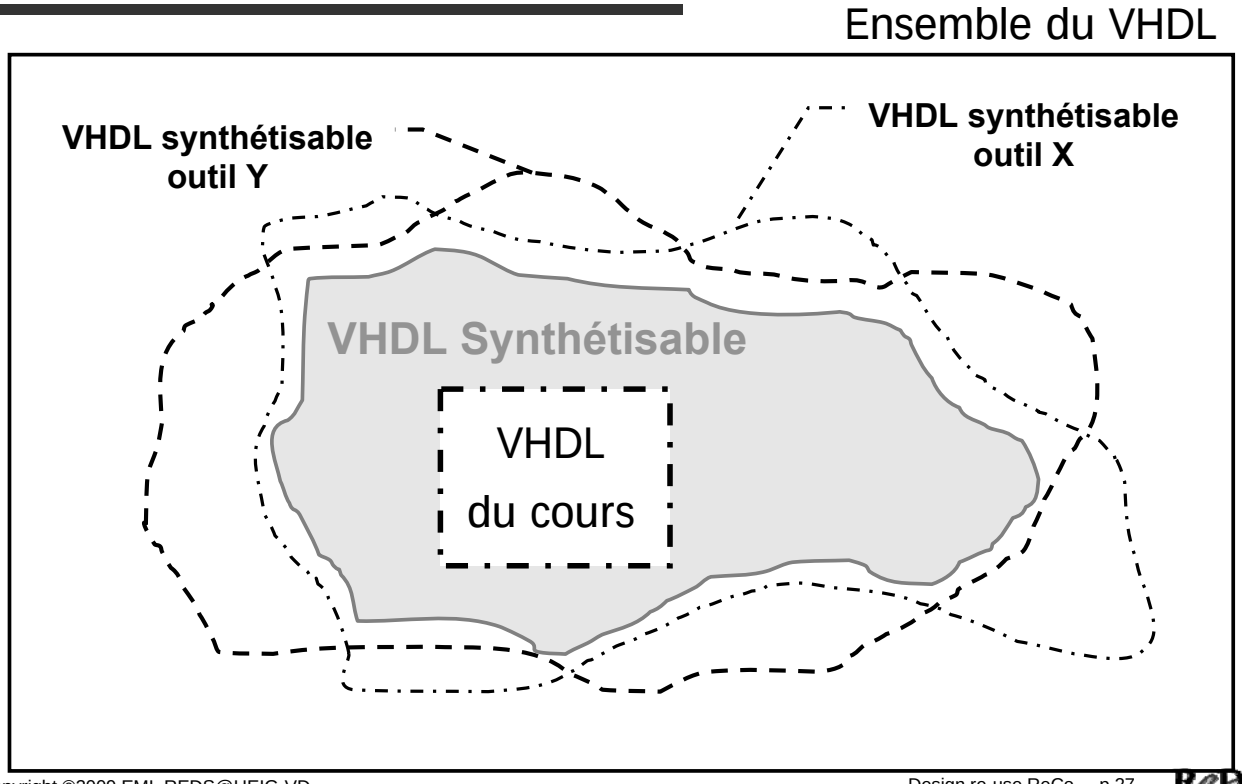
Fausses idées sur le langage VHDL

Le langage en tant que tel ne garantit pas :

- ✓qualité des descriptions
- ✓portabilité des descriptions
- ✓descriptions soient synthétisables
- ✓matériel optimum

Mauvais concepteur + VHDL = **catastrophe**

Ensemble synthétisable du VHDL



Conception avec le VHDL

- Il faut penser CIRCUIT.
- Une bonne conception commence par une décomposition du système (hiérarchie).
- Il faut imaginer l'architecture physique du système.
- Le VHDL n'est pas un outil de CONCEPTION.
- Le VHDL est un outil de DESCRIPTION.

Apprentissage du langage

- Connaissance des instructions VHDL seules est insuffisante
- Ensemble du langage VHDL pas nécessaire
- Structures des descriptions pour la synthèse
- Différence entre simulation et synthèse
- Méthodologie

Formation sur le langage INDISPENSABLE

La portabilité des descriptions

- Afin de garantir une bonne portabilité des descriptions :
  - ✓méthodologie indispensable
  - ✓une seule fonction par module VHDL
  - ✓faire des descriptions simples et lisibles
  - ✓expliciter les éléments mémoires
  - ✓utiliser uniquement les bibliothèques standardisées IEEE et ses propres bibliothèques

Les bibliothèques recommandées

- Portabilité des descriptions assurées en utilisant les bibliothèques IEEE :
  - ✓`Std_Logic_1164` types et fonctions de bases
  - ✓`Numeric_Std` opérations arithmétiques
- Bibliothèques propres (générale, projet):
  - ✓disposer des sources
  - ✓maîtriser le contenu
  - ✓pièces et boite à outils personnel

Le type Std_uLogic (type énuméré)

Défini par le paquetage IEEE.Std_Logic_1164

```
type Std_uLogic is (
      'U',-- état non initialisé
      'X',-- état inconnu fort
      '0',-- état logique 0 fort
      '1',-- état logique 1 fort
      'Z',-- état haute impédance
      'W',-- état inconnu faible
      'L',-- état logique 0 faible
      'H',-- état logique 1 faible
      '-' -- état indifférent, don't care );
```


… type Std_uLogic …

Etats utilisables pour l'affectation d'un signal en synthèse

```
type Std_uLogic is (
       …
      'X',-- pour cas de simulation
      '0',-- état 0
      '1',-- état 1
      'Z',-- état haute impédance
       …
       …
       …
      '-' -- état indifférent, don't care );
```

… type Std_uLogic

Etats utilisables pour tester l'état d'un signal en synthèse

```
type Std_uLogic is (
        …
        …
       '0',-- état 0
       '1',-- état 1
        …
        …
        …
        …
        …                                        );
```


… type Std_uLogic …

Etats utilisables pour l'affectation d'un signal en spécification et en simulation

```
type Std_uLogic is (
        …
       'X',-- pour cas de simulation
       '0',-- état 0
       '1',-- état 1
       'Z',-- état haute impédance
        …
       'L',-- état 0 faible => pull-down
       'H',-- état 1 faible => pull-up
       '-' -- état indifférent, don't care );
```

Déroulement concurrent et séquentiel

- Dans un langage informatique :
  - ✓les instructions ont un déroulement séquentiel
- Dans un circuit :
  - ✓toutes les portes fonctionnent simultanément
  - ✓tous les signaux évoluent de manière concurrente
- Le langage VHDL dispose d'instructions concurrentes pour la description de circuits (matériel)

L'instruction *process* ...

- Le langage VHDL dispose d'une instruction *process* dont la syntaxe est :

```
process (Liste_De_Sensibilité)
  --zone de déclaration
begin

  --Zone pour instructions
  --              séquentielles ....

end process;
```

Elément puissant du langage VHDL

L'instruction *process*

- Exécution séquentielle à l'intérieur
  Utilisation des instructions séquentielles !
- Le temps **ne progresse pas** durant l' exécution
- Activé uniquement lorsqu'un signal de la liste de sensibilité change
- Variables utilisables à l'intérieur d'un process
  => description algorithmique possible avec un process

Processus et liste de sensibilité …

- Avec liste de sensibilité :
  - ✓instruction *wait* **interdite**
  - ✓utilisé pour les descriptions synthétisables
  - ✓processus activé **uniquement** lorsqu'un signal de la liste de sensibilité change
  - ✓processus endormi à la fin (*end process*)
  - ✓permet de décrire avec un algorithme :
    - →système combinatoire ou
    - →système séquentiel ou
    - →model, spécification.

Processus et liste de sensibilité ...

- Sans liste de sensibilité :
  - ✓utilisation de l'instruction *wait*
  - ✓utilisé pour fichiers de simulation & spécification
  - ✓processus activé à l'instant 0 ns
  - ✓processus endormi à chaque *wait*
  - ✓réactivé automatiquement à la fin (end process)

  Attention :
  - ✓processus sans liste de sensibilité et sans *wait* à une durée d'exécution nul !

Bloque le simulateur

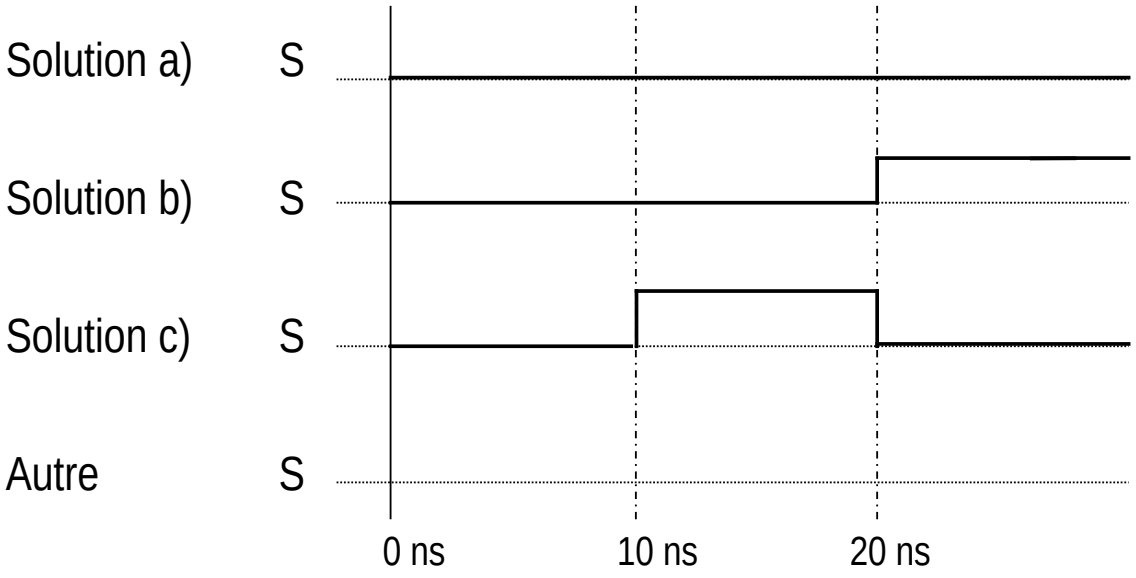
Exemple de processus

Comportement séquentielle à l'intérieur d'un process

```
process
begin
  S <= '0';
  S <= '1' after 20 ns;
  S <= '1' after 10 ns, '0' after 20 ns;
  wait ;
end process;
```

Complétez le chronogramme ci-après

Quel chronogramme est correct ?



Variables et process

- Indispensable pour description avec un algorithme :
  - ✓Signal n'évolue pas durant l'évaluation d'un *process*
  - ✓Variable évolue durant l'évaluation d'un *process*
- Mais, lors de la synthèse, il est possible que :
  - ✓la variable n'existe pas
  - ✓la variable corresponde à un signal
  - ✓la variable corresponde à un élément mémoire
- En synthèse, important de penser MATERIEL

Déclaration d'une variable …

- Syntaxe de la déclaration

```
process (…              )
  variable V1 : Std_Logic;
begin
  …
```

✓ la variable est crée à l'instant t=0ns, elle aura l'état non initialisé 'U' (*Uninitialized*)

✓ lorsque le processus est endormi, la variable **conserve** son état

Exemple de processus avec une variable …

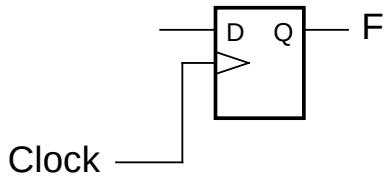
```
  signal F : Std_logic;
  signal A : Std_logic_Vector(3 downto 0);

-------------------------------------------
  process(Clock)
    variable V : Std_Logic;
  begin
    if Rising_Edge(Clock) then
      for I in 0 to 3 loop
         V := V and A(I);
      end loop;
      F <= V;
    end if;
  end process;
```

[a]

- Quelle logique est générée ?

… processus avec variable …

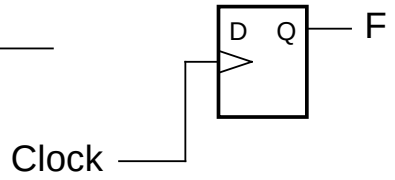
```
process(Clock)
  variable V : Std_Logic;
begin
  if Rising_Edge(Clock) then
    V := V and A(0);
    V := V and A(1);
    V := V and A(2);
    V := V and A(3);
    F <= V;
  end if;
end process;
```



… processus avec variable …

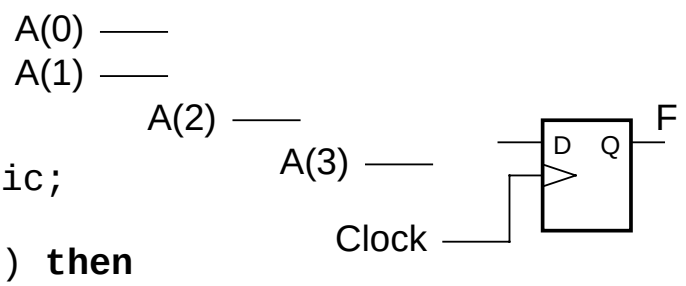
Solution sans bascule pour la variable : version 1

```
process(Clock)
  variable V : Std_Logic;
begin
  if Rising_Edge(Clock) then
    V := '1';
    for I in 0 to 3 loop
       V := V and A(I);
    end loop;
    F <= V;
  end if;
end process;
```



… processus avec variable.

Solution sans bascule pour la variable : version 2




```
process(Clock)
  variable V : Std_Logic;
begin
  if Rising_Edge(Clock) then
    V := A(0);
    for I in 1 to 3 loop
       V := V and A(I);
    end loop;
    F <= V;
  end if;
end process;
```


Description SLC avec un process

IMPORTANT (SLC : Système Logique Combinatoire)

- Toutes les entrées du SLC **doivent** être dans la liste de sensibilité.
- Toutes les sorties du SLC doivent être initialisées au **début** du processus.
  ou
  *Toutes les sorties du SLC doivent êtres affectées dans **toutes** les branches des instructions séquentielles.*
- **Interdit** d'utiliser, dans le process, les sorties générée par celui-ci ! (Si nécessaire utiliser une variable dans le process)

Description algorithmique …

- Description du comportement du démultiplexeur :
  - Si Enable est actif alors
    - la sortie correspondante à Sel est activée
    - toutes les autre sorties sont inactives
- autre variante :
  - Toutes les sorties sont inactives
  - Si Enable est actif alors
    - la sortie correspondante à Sel est activée

Description algorithmique démultiplexeur …

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity DMUX1_4 is
  port(Sel_i : in  Std_Logic_Vector(1 downto 0);
       En_i  : in  Std_Logic;
       Y_o   : out Std_Logic_Vector(3 downto 0)
      );
end DMUX1_4;
```

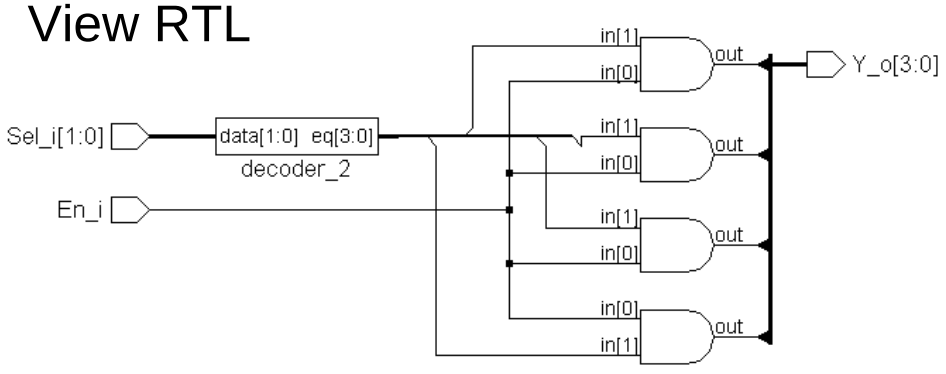
ATENTION : description non synthétisable

… description algorithmique démultiplexeur …

```
architecture Comport of DMUX1_4 is
begin

  process(Sel_i ,En_i)
  begin
    Y_o <= (others => '0'); -- Valeur par défaut

    Y_o(To_Integer(Unsigned(Sel_i))) <= En_i;

  end process;
end Comport;
```

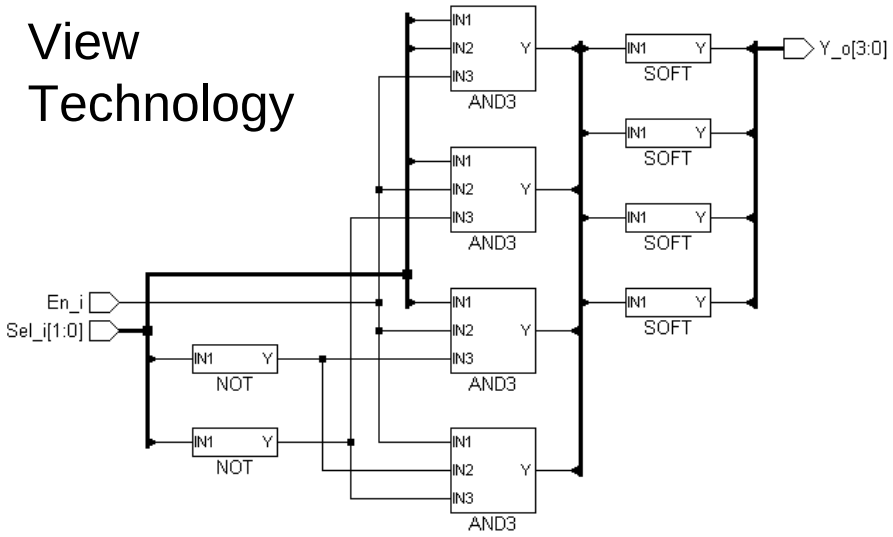

Synthèse avec Leonardo …

View RTL



… synthèse avec Leonardo

View
Technology



Synthèse DMUX avec Max+plus II (ALTERA)

Equations fournies dans fichier *.rpt

```
Y_o0    = LCELL( _EQ001 $  GND);
 _EQ001 =  En_i & !Sel_i0 & !Sel_i1
        #  Sel_i0 &  Y_o0
        #  Sel_i1 &  Y_o0
        #  En_i & !Sel_i1 &  Y_o0
        #  En_i & !Sel_i0 &  Y_o0;

Y_o1    = LCELL( _EQ002 $  GND);
 _EQ002 =  En_i &  Sel_i0 & !Sel_i1
        #  Sel_i1 &  Y_o1
        # !Sel_i0 &  Y_o1
        #  En_i &  Sel_i0 &  Y_o1
        #  En_i & !Sel_i1 &  Y_o1;
```

← **terme correct**

termes inutiles !

← **terme correct**

termes inutiles !

Description algorithmique pas supportée par Max+plus II

```
process(Sel_i ,En_i)
begin
  Y_o <= (others => '0'); -- Valeur par défaut
  Y_o(To_Integer(Unsigned(Sel_i))) <= En_i;
end process;
```

Indice variable

donc :

description NON SYNTHETISABLE !

Description moins complexe du démultiplexeur

```
architecture Comport of DMUX1_4 is
begin

  process(Sel_i ,En_i)
  begin
    Y_o <= (others => '0'); -- Valeur par défaut
    for I in 0 to Y_o'length-1 loop
      --la boucle for permet d'expliciter les
      --comparaisons (test =)
      if I = To_Integer(Unsigned(Sel_i)) then
        Y_o(I) <= En_i;
      end if;
    end loop;
  end process;

end Comport;
```

pas d'indice variable

Synthèse avec Max+plus II (ALTERA)

Equations fournies dans fichier *.rpt

```
Y_o0    = LCELL( _EQ001 $  GND);
 _EQ001 =  En_i & !Sel_i0 & !Sel_i1;

Y_o1    = LCELL( _EQ002 $  GND);
 _EQ002 =  En_i &  Sel_i0 & !Sel_i1;

Y_o2    = LCELL( _EQ003 $  GND);
 _EQ003 =  En_i & !Sel_i0 &  Sel_i1;

Y_o3    = LCELL( _EQ004 $  GND);
 _EQ004 =  En_i &  Sel_i0 &  Sel_i1;
```

Equations correctes !

Description d'éléments mémoires

- Pas de déclaration explicite en VHDL
- Description indique au synthétiseur :
  signal correspond à un élément mémoire

La description doit :

Diriger correctement le synthétiseur

afin d'obtenir l'élément mémoire désiré

Comportement par défaut du langage VHDL

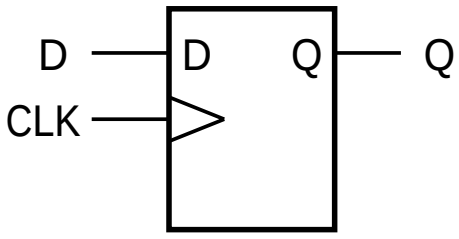
Lors de l'utilisation d'instructions séquentielles :

Cas non traité $\Rightarrow$ VHDL maintien l'état du signal

- Cette fonctionnalité servira de base pour écrire les éléments mémoires
- Cette fonctionnalité sera aussi un piège par l'obtention de *latchs* non désirés

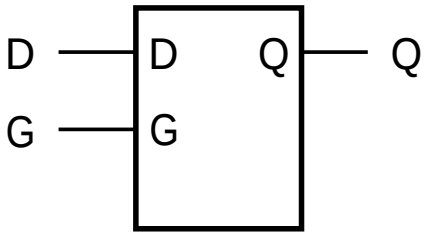
Description d 'un flip-flop

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Flip_Flop is
   port (D       : in Std_Logic;
         Clock   : in Std_Logic;
         Q       : out Std_Logic);
end Flip_Flop;

architecture Comport of Flip_Flop is
begin
  process(Clock)
  begin
    if Rising_Edge(Clock) then
       Q <= D;
    end if;
  end process;
end Comport;
```



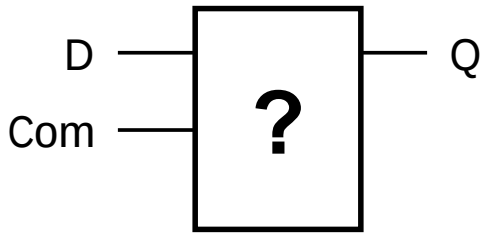
Description d'un latch

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Latch is
   port (D      : in Std_Logic;
         G      : in Std_Logic;
         Q      : out Std_Logic);
end Latch;

architecture Comport of Latch is
begin
  process(G, D)
  begin
    if G = '1' then
       Q <= D;
    end if;
  end process;
end Comport;
```



Type de bascule ?

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Bascule is
   port (D   : in Std_Logic;
         Com : in Std_Logic;
         Q   : out Std_Logic);
end Bascule;

architecture Comport of Bascule is
begin
  process(Com)
  begin
    if Com = '1' then
       Q <= D;
    end if;
  end process;
end Comport;
```



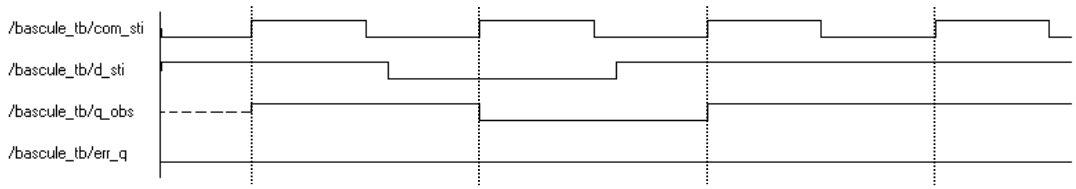
Synthèse de cette bascule

- Cette description est ambiguë.
- Deux informations se contredissent :
  - ✓le processus ne réagit que sur Com
    => action dynamique => flip-flop
  - ✓la condition du test spécifie un niveau (=)
    => action sur un niveau => latch

Type de bascule ?

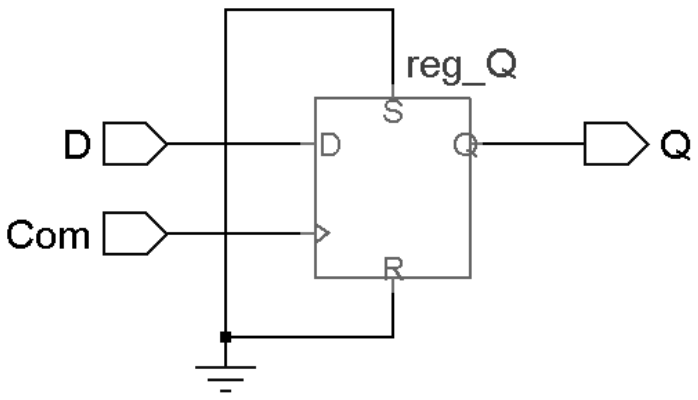
- Compréhension par le simulateur, inter- prétation stricte de la description VHDL:
  - ✓une seule réponse : .....
- Compréhension par le synthétiseur qui va traduire la description VHDL en logique:
  - ✓priorité à la liste de sensibilité : .....
  - ✓priorité à la condition de test : .....

Résultat de la simulation (ModelSim)



Le comportement correspond bien a celui d'un flip-flop D

Synthèse avec Léonardo

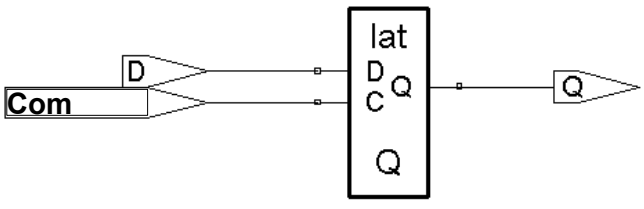


Le résultat de la synthèse est un **flip-flop**

Synthèse avec Synplify : 3 *WARNING*

```
Synthesizing work.bascule.comport
@W:"f:\en_cours\synplify\bascule.vhd":30:2:30:8
  Incomplete sensitivity list - assuming
  completeness
@W:"f:\en_cours\synplify\bascule.vhd":33:10:33:10
  Referenced variable d is not in sensitivity list
Post processing for work.bascule.comport
@W:"f:\en_cours\synplify\bascule.vhd":32:4:32:5
  Latch generated from process for signal q,
  probably caused by a missing assignment in an if
  or case stmt
@END
```


Synthèse avec Synplify



Le résultat de la synthèse est un LATCH

Conclusion sur cette description

- Le comportement en simulation peut-être différent du matériel obtenu
- Matériel obtenu dépend du synthétiseur, résultat pas identique

Cette description est **NON SYNTHETISABLE**

Interdit par la norme IEEE 1076.6-1999

Etat initial d'une bascule

- A l'instant t = 0 ns :
  Tous les signaux sont à l'état 'U'
- Une initialisation est indispensable
- Comment forcer l'état initial d'une bascule en VHDL ?

Proposition I

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Flip_Flop is
   port (D     : in Std_Logic;
         Clock : in Std_Logic;
         Q     : out Std_Logic := '0' );
end Flip_Flop;

architecture Comport of Flip_Flop is
begin
  process(Clock)
  begin
    if Rising_Edge(Clock) then
      Q <= D;
    end if;
  end process;
end Comport;
```


Proposition II

```
Library IEEE;
use IEEE.Std_Logic_1164.all;

entity  Flip_Flop is
   port (D       : in Std_Logic;
         Clock   : in Std_Logic;
         Reset   : in Std_Logic;
         Q       : out Std_Logic);
end Flip_Flop;

architecture Comport of Flip_Flop is
begin
  process(Clock, Reset)
  begin
    if Reset = '1' then
      Q <= '0';
    elsif Rising_Edge(Clock) then
      Q <= D;
    end if;
  end process;
end Comport;
```

Conclusion initialisation bascule

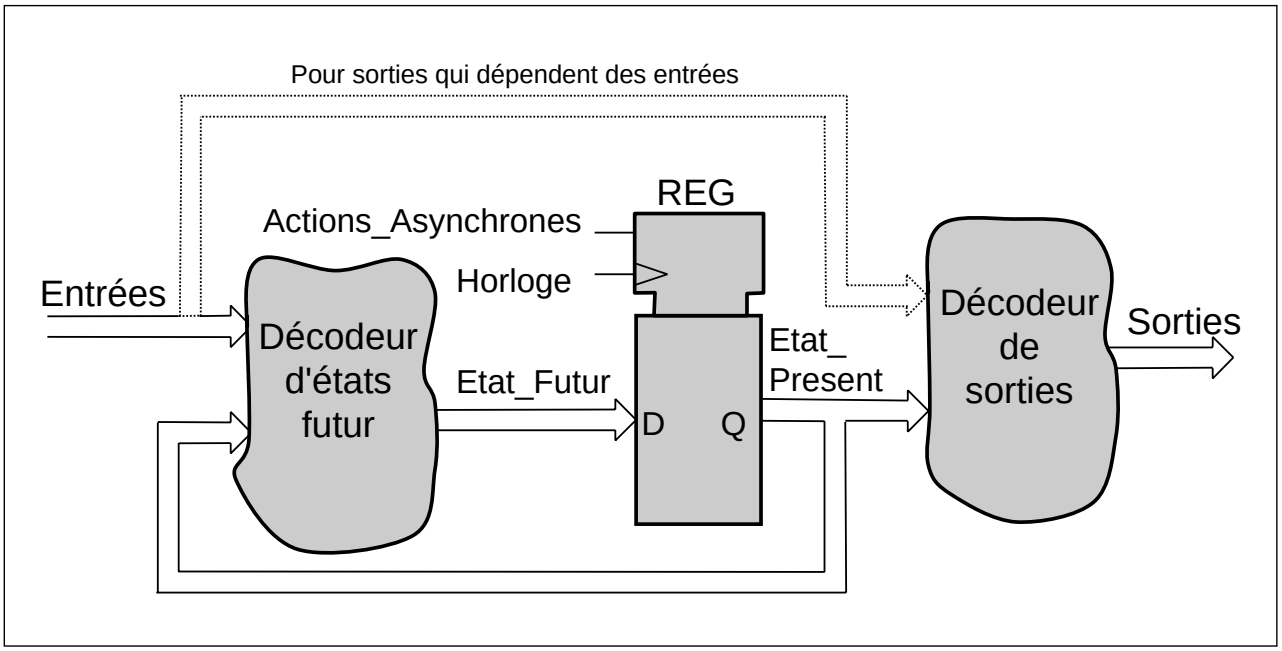
- Proposition I
  - ✓fonctionne en simulation VHDL
  - ✓ne sera pas synthétisé : "soft"
- Proposition II
  - ✓fonctionne en simulation VHDL
  - ✓sera correctement synthétisé : "hard"
  - ✓fonctionne en simulation après synthèse

Description systèmes séquentiels

Principe :

- Suivre la décomposition d'un système séquentiel
- Décomposition visible dans la description
- Avantages :
  - ✓ description fiable (synthèse)
  - ✓ description lisible
- Inconvénients :
  - ✓ description un peu plus longue (plus de lignes)

Décomposition système séquentiel



Description système séquentiel

Description décomposée en trois parties :

- Décodeur d'états futur, combinatoire :
  - ✓ description concurrente ou séquentielles (*process*)
- Elément mémoire, séquentiel :
  - ✓ description avec un processus
- Décodeur de sortie, combinatoire :
  - ✓ description concurrente ou séquentielles (*process*)

Syntaxe description système séquentiel

```
architecture Comport of Sys_Seq is
begin
  -- Description concurrente du décodeur d'état futur
     . . .
   Mem: process (Horloge, Entree_Action_Asynch)
  begin
    if (Entree_Action_Asynch = '1') then
      Etat_Present <= Etat_Initial;
    elsif Rising_Edge(Horloge) then
      Etat_Present <= Etat_Futur;
    end if;
  end process;
  -- Description concurrente du décodeur de sorties
     . . .
end Comport;
```


Description d'un compteur …

- Compteur 4 bits
- Reset asynchrone actif haut
- Actions synchrones :

Charge	Compte	Description	
'1'	-	charge	Cpt = Valeur
'0'	'1'	compte	Cpt = Cpt + 1
'0'	'0'	maintien	Cpt = Cpt

Ordre de priorité : charge, compte, maintien

… description d'un compteur …

Traduction "textuelle" du fonctionnement synchrones

Si Charge actif alors	Cpt = Valeur
sinon Compte actif alors	Cpt = Cpt + 1
sinon	Cpt = Cpt

description aisée avec une instruction when…else

```
Cpt_Futur <=
  Valeur          when Charge = '1' else
  Cpt_Present +1  when Compte = '1' else
  Cpt_Present;
```


… description compteur …

```
library IEEE;
use IEEE.Std_Logic_1164.all;
--definit les operations arithmetiques
use IEEE.Numeric_Std.all;

entity Compteur is
   port(Charge, Compte : in Std_Logic;
        Horloge, Reset : in Std_Logic;
        Valeur : in  Std_Logic_vector(3 downto 0);
        Cpt    : out Std_Logic_Vector(3 downto 0)
        );
end Compteur;
```

… description compteur …

- Description concurrente du décodeur d'états futur

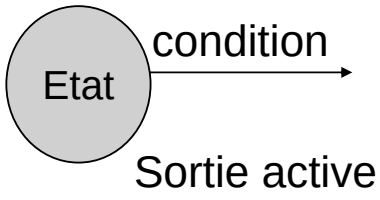
```
architecture Comport of Compteur is
 signal Cpt_futur, Cpt_Present : unsigned(3 downto 0);
begin
  --Description concurrente du décodeur d'états futur
  --ordre de priorite : charge, compte, maintien
  Cpt_Futur <=
        Unsigned(Valeur) when (Charge = '1') else
        Cpt_Present +1   when (Compte = '1') else
        Cpt_Present; -- maintien
```


… description compteur

```
  Mem: process (Horloge, Reset)
  begin
    if (Reset = '1') then
      Cpt_Present <= (others => '0');
    elsif Rising_Edge(Horloge) then
      Cpt_Present <= Cpt_Futur;
    end if;
  end process;

  --Mise a jour de l'etat du compteur
  Cpt <= Std_Logic_Vector(Cpt_present);

end Comport;
```

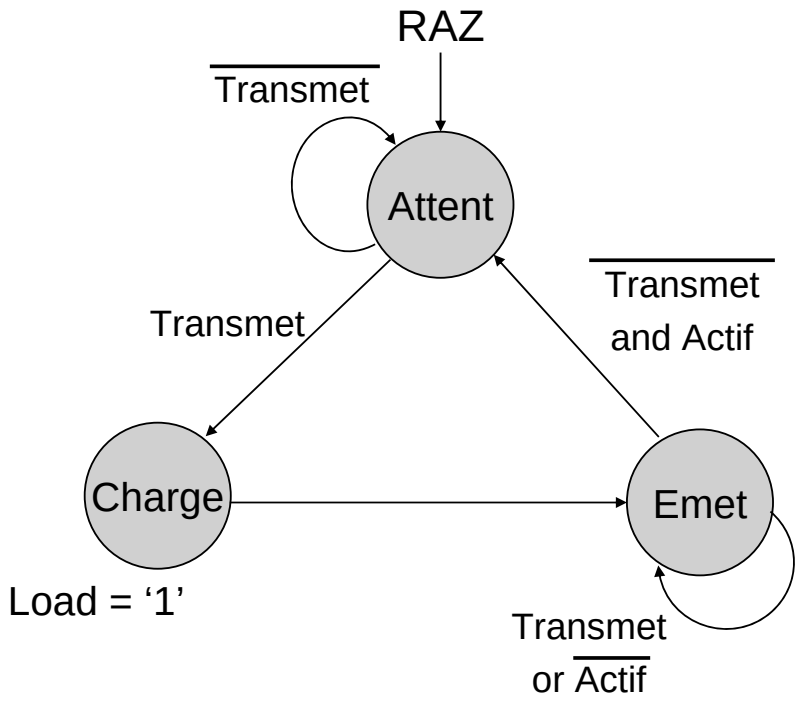
Exemple de machine d'état …

Convention graphe



Entrées:
- Transmet, Actif
- Load

Sortie:
- Load



… exemple de machine d'état …

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity UC_Emetteur is
  port ( Transmet, Actif : in  Std_Logic;
         Clock, Reset :    in  Std_Logic;
         Load :            out Std_Logic);
end UC_Emetteur;

architecture M_Etat of UC_Emetteur is

  signal Etat_Pres, Etat_Fut : Std_Logic_Vector(1 downto 0);

--Les constantes permettent de fixer le codage
--Si modification : seul les constantes sont a corriger.
  constant Attent : Std_Logic_Vector(1 downto 0) := "00";
  constant Charge : Std_Logic_Vector(1 downto 0) := "01";
  constant Emet   : Std_Logic_Vector(1 downto 0) := "10";
begin
```

```
--process combinatoire, calcul etat futur
Fut:process (Transmet, Actif, Etat_Present)
begin
  Etat_Futur <= Attent; -- valeur par defaut
  case Etat_Present is
    when Attent =>
    if Transmet = '1' then
      Etat_Futur <= Charge;
    else
      Etat_Futur <= Attent;
    end if;
    when Charge =>
    Etat_Futur <= Emet;
    when Emet =>
    if Transmet = '0' and Actif = '0' then
      Etat_Futur <= Attent;
    else
      Etat_Futur <= Emet;
    end if;
    when others =>
    Etat_Futur <= Attent;
  end case;
end process;
```


… exemple de machine d'état

```
--processus de memorisation
Mem:process (Clock, Reset)
begin
  if Reset = '1' then
    -- reset asynchrone prioritaire
    Etat_Present <= Attent;
  elsif Rising_Edge(Clock) then
    Etat_Present <= Etat_Futur;
  end if;
end process;

-- equations combinatoire de sortie
Load <= '1' when Etat_Present = Charge else '0';

end M_Etat;
```

Design re-use

Descriptions paramétrables

Design re-use ...

- Objectifs :
  - ✓augmenter la productivité de descriptions
  - ✓obtenir des descriptions lisibles et fiables
  - ✓disposer de description portables
  - ✓gain de temps : *Time to Market*
- Comment :
  - ✓améliorer l'écriture des descriptions
  - ✓bénéficier des performances du VHDL

… design re-use

- Ecrire des descriptions paramétrables
  - ✓Paramètres modifiés pour la synthèse
  - ✓Module synthétisé aura une taille fixe
- Taille des vecteurs doit être modifiable
- Utilisation des attributs indispensables
- Plusieurs possibilités de descriptions paramétrables

Défi

- Etre capable de réaliser un système répondant aux caractéristiques (vitesse-performance) dans le temps impartit
  - ✓Travail 100% à la main impossible
  - ✓Sans réutilisation impossible d'utiliser toutes les capacités des FPAGs !
  - ✓Nécessite de réutiliser des composants
  - ✓Nouvelle méthodologie indispensable

Descriptions réutilsables

- Respecter des règles de méthodologies
  => thème a été traité dans la présentation précédente
  - ✓Identificateurs, mots réservés, ...
  - ✓Structure description système combinatoire avec process
  - ✓Structure description système séquentiel
- Description simple et lisible
- Commentaires dans la description
- Une seule fonction par module VHDL
  - ✓Compteur Up/Dn, Registre à décalage, ...

Descriptions paramétrables

- Solutions possibles :
  - ✓taille des vecteurs définie par les déclarations dans l'entité
  - ✓taille des vecteurs définie dans un paquetage
    - →sous-type, constante
  - ✓Utilisation de constantes génériques (*generic*)
  - ✓Utilisation de vecteurs non contraints

=> utilisation des attributs indispensable

Design re-use et Outils

- Certains synthétiseurs ne supportent pas le VHDL utilisé pour les descriptions paramétrables.
- Génériques et non contraint
  - ✓utilisable uniquement avec des outils performants !
- Solution portable sur tous les outils :
  - ✓taille définie par les déclarations dans l'entité
  - ✓définir des constantes ou sous-types dans un paquetage

Descriptions paramétrables, taille définie dans l'entité …

- Signaux internes basés sur la taille des vecteurs déclarés dans l'entité

```
entity Exemple is
  port(…
       Vect_o : out Std_logic_Vector(9 downto 0));
end Exemple;
architecture Comport of Exemple is
  signal Vect_s : Std_logic_Vector(Vect_o'range);
begin
  …
end Comport;
```

… taille définie dans l'entité …

- Exemple compteur 4 bits

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity Cpt4_e is
  port(Reset_i   : in  Std_logic;
       Clock_i   : in  Std_logic;
       Cpt_o     : out Std_logic_Vector(3 downto 0);
       Cpt_Max_o : out Std_logic  );
end Cpt4_e;
architecture Comport of Cpt4_e is
  signal Cpt_s : Unsigned(Cpt_o'range);
…
```


… taille définie dans l'entité …

```
…
begin
  process(Reset_i, Clock_i)
  begin
    if Reset_i = '1' then
      Cpt_s <= (others => '0');
    elsif Rising_Edge(Clock_i) then
      Cpt_s <= Cpt_s + 1;
    end if;
  end process;
  --Affectation des sorties
  Cpt_o     <= Std_logic_Vector(Cpt_s);
  Cpt_Max_o <= '1' when Cpt_s = (2**Cpt_o'length)-1 else
               '0';
end Comport;
```

Descriptions paramétrables avec un paquetage

- Déclarer un sous-type dans un paquetage
- Définir la taille des vecteurs en déclarant une constante dans un paquetage
- Même constante pour toutes les descriptions
- Sous-type et constante visible partout

Déclaration d'un paquetage …

- Syntaxe :

```
package My_Tools is
  --zone de déclaration du paquetage
end My_Tools;

package body My_Tools is
  --zone de déclaration du corps du paquetage
end My_Tools;
```

- Par défaut :
  - ✓paquetage placé dans la bibliothèque *work*

Les types

- Tout objet manipulé doit avoir un type
- Définir des objets personnalisés
- Utile pour paquetages, spécifications et test benchs
- Présentation de types pour les signaux et variables uniquement

Le sous-type (subtype)

- Restriction des valeurs d'un type
- Hérite des opérations prédéfinies pour le type
- Exemple :

– limitation des valeurs du type *integer*

```
subtype T_Integer_0a15 is integer range 0 to 15;
```

– Explicite un sous-type pour un bus

```
subtype T_Bus_Adr is Std_Logic_Vector(9 downto 0);
```

Type énuméré

- Type pour une machine d'états :

```
type Type_Etat is (Init, Run, Stop, Att, Fin)
signal Etat_Pres, Etat_Fut : Type_Etat;
```

Remarque :

Le codage de la machine d'état sera défini lors de la synthèse. Il faudra vérifier les options du synthétiseur et faire les choix souhaités.

Type de tableau de vecteur

- Construction d'un tableau de vecteur

```
type Type_Tab_Vect is array (natural range <> )
                  of Std_logic_Vector(7 downto 0);
```

Tableau non contraint (taille non définie)

- Exemple : chaîne de caractères

```
--ce type est définit dans le paquetage standard
type string is array (positive range <> )
                                  of charactere;
```

Déclaration dans le paquetage

- Déclaration de sous-type et/ou de constante :

```
package My_Define is
  subtype T_Reg is Std_Logic_Vector(15 downto 0);
  constant Taille_Vect : Natural := 16;
end My_Define;
```


Description paramétrable avec paquetage

```
library IEEE;
use IEEE.Std_Logic_1164.all;My_Tools is

--Appel au paquetage personnel
use Work.My_Define.all;

entity Nom_Entite is
  port (Signal_i : in  Std_Logic;
        Vect_i   : in  Std_Logic_Vector
                     (Taille_Vect-1 downto 0);
            …
        Reg_o    : out T_Reg
        );
end Nom_Entite;
```

Descriptions paramétrables avec constante générique

- Déclarer des constantes *generic* dans l'entité
- Valeurs des constantes définies lors de chaque instanciation du composant
- La même description peut-être utilisée avec des tailles différentes

Les constantes génériques ...

- Déclaration de constantes génériques dans l'entité :

```
entity Nom_Entite is
  generic (Const0_g : Type_0 := Val_Defaut_0;
             …
           Constm_g : Type_m := Val_Defaut_m);
  port (Nom_Port_0 : mode Type_A;
             …
        Nom_Port_n : mode Type_Z);
end Nom_Entite;
```

… constantes génériques …

- Valeurs des constantes fixées lors de chaque instanciation du composant :

```
label: Nom_Composant
  generic map (Const_0_g =>Valeur_0,
                  ...
               Const_m_g =>Valeur_m)
  port map (port1=>signal1,
                ...
            portn=>signaln);
```


Exemple description générique …

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity RegN_en is
  generic( N_g : Positive range 1 to 31 := 4);
  port(Horloge_i : in  Std_Logic;
       Reset_i   : in  Std_Logic;
       Enable_i  : in  Std_Logic;
       Data_i : in  Std_Logic_Vector(N_g-1 downto 0);
       Reg_o  : out Std_Logic_Vector(N_g-1 downto 0)
       );
end RegN_en;
```

```
architecture Comport of RegN_en is
  signal Reg_s : Std_Logic_Vector(Reg_o'range);
                          -- ou (N_g-1 downto 0)
begin

  process(Reset_i, Horloge_i)
  begin
    if Reset_i = '1' then
      Reg_s <= (others => '0');
    elsif Rising_Edge(Horloge_i) then
      if Enable_i = '1' then
         Reg_s <= Data_i;
      end if;
    end if;
  end process;

 --Affectation de la sortie
  Reg_o <= Reg_s;

end Comport;
```


… exemple description générique …

```
entity Exemple is
   ...
end Exemple;

architecture Struct of Exemple is

  component RegN_en is
    generic( N_g : Positive range 1 to 16 := 4);
    port(Horloge_i : in  Std_Logic;
         Reset_i   : in  Std_Logic;
         Enable_i  : in  Std_Logic;
         Data_i : in  Std_Logic_Vector(N_g-1 downto 0);
         Reg_o  : out Std_Logic_Vector(N_g-1 downto 0)
         );
  end component;

...
```

… exemple description générique

```
 signal En_s     : Std_Logic;
 signal Valeur_s : Std_Logic_Vector(7 downto 0);
 signal Sortie_s : Std_Logic_Vector(7 downto 0);
...
begin
...
  --Intanciation d'un registre 8 bits
  Reg8: RegN_en
    generic map ( N_g => 8            )
    port map(Horloge_i => Horloge_i,
             Reset_i   => Reset_i,
             Enable_i  => En_s;
             Data_i    => Valeur_s,
             Reg_o     => Sortie_s      );

...
```


Descriptions paramétrables avec des vecteurs non contraint

- Il est possible de déclarer un vecteur sans dimension (*unconstrained*)
- La taille du vecteur sera définie lors de l'instanciation du composant
- Il n'est pas nécessaire de déclarer un générique
- La description **seule** n'est pas synthétisable

Vecteur non contraint

- La déclaration du type `Sdt_Logic _Vector` est par définition non contraint :

```
type Std_Logic_Vector is
    array(natural range<>) of Std_Logic;
```

- Au lieu de spécifié la taille lors de la définition, nous le ferons lors de l'instanciation du composant.

Exemple description paramétrable avec non contraint...

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Reg_en is
  port(Horloge_i : in  Std_Logic;
       Reset_i   : in  Std_Logic;
       Enable_i  : in  Std_Logic;
       Data_i    : in  Std_Logic_Vector;
       Reg_o     : out Std_Logic_Vector
       );
end Reg_en;
```

```
architecture Comport of Reg_en is
  signal Reg_s : Std_Logic_Vector(Reg_o'range);
begin

  process(Reset_i, Horloge_i)
  begin
    if Reset_i = '1' then
      Reg_s <= (others => '0');
    elsif Rising_Edge(Horloge_i) then
      if Enable_i = '1' then
         Reg_s <= Data_i;
      end if;
    end if;
  end process;

 --Affectation de la sortie
  Reg_o <= Reg_s;

end Comport;
```


… exemple avec non contraint …

```
entity Exemple is
   ...
end Exemple;

architecture Struct of Exemple is

  component Reg_en is
    port(Horloge_i : in  Std_Logic;
         Reset_i   : in  Std_Logic;
         Enable_i  : in  Std_Logic;
         Data_i    : in  Std_Logic_Vector;
         Reg_o     : out Std_Logic_Vector   );
  end component;

...
```

… exemple avec non contraint

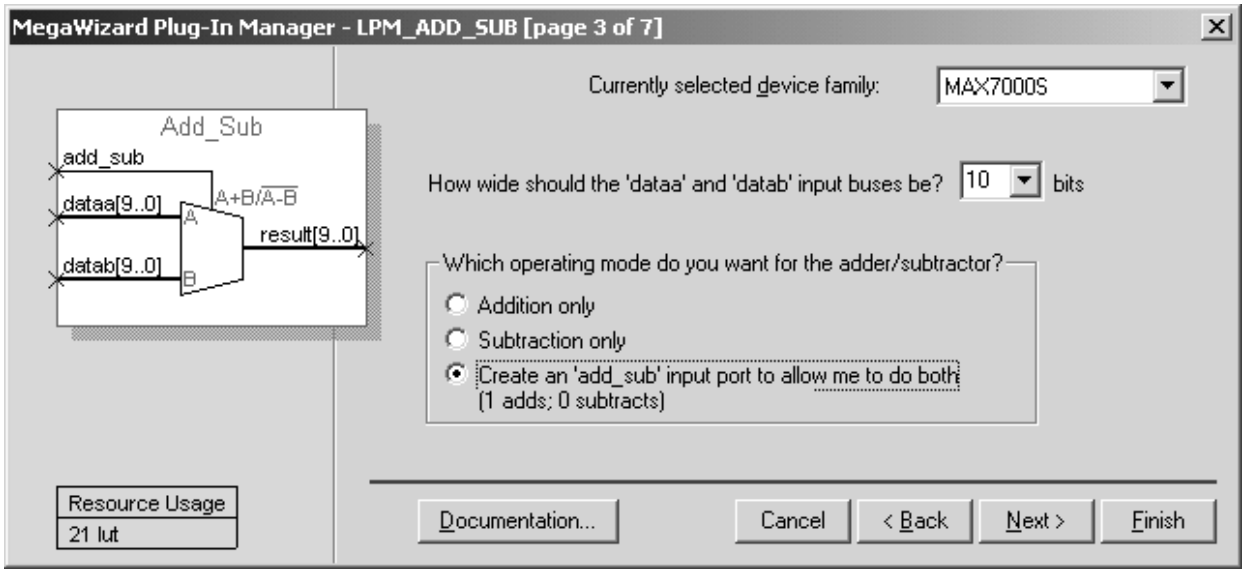
```
  --Déclaration de signaux internes
  signal En_s      : Std_Logic;
  signal Val_s     : Std_Logic_Vector(7 downto 0);
  signal Val_Mem_s : Std_Logic_Vector(7 downto 0);

begin
  --Intanciation d'un registre 8 bits
  Reg8: Reg_en
    port map(Horloge_i => Horloge_i,
             Reset_i   => Reset_i,
             Enable_i  => En_s;
             Data_i    => Val_s,
             Reg_o     => Mem_s       );

...
```


Autres possibilités :

- Utiliser des IPs du marché
  - ✓Full VHDL : modifiable, mais cher
  - ✓Netlist : plus modifiable, valable pour une technologie
  - ✓Important : disposer d'un modèle et d'un banc de test
- Utiliser les mega-fonctions proposées par les fabricants de FPGAs
  - ✓Attention: solution non portable

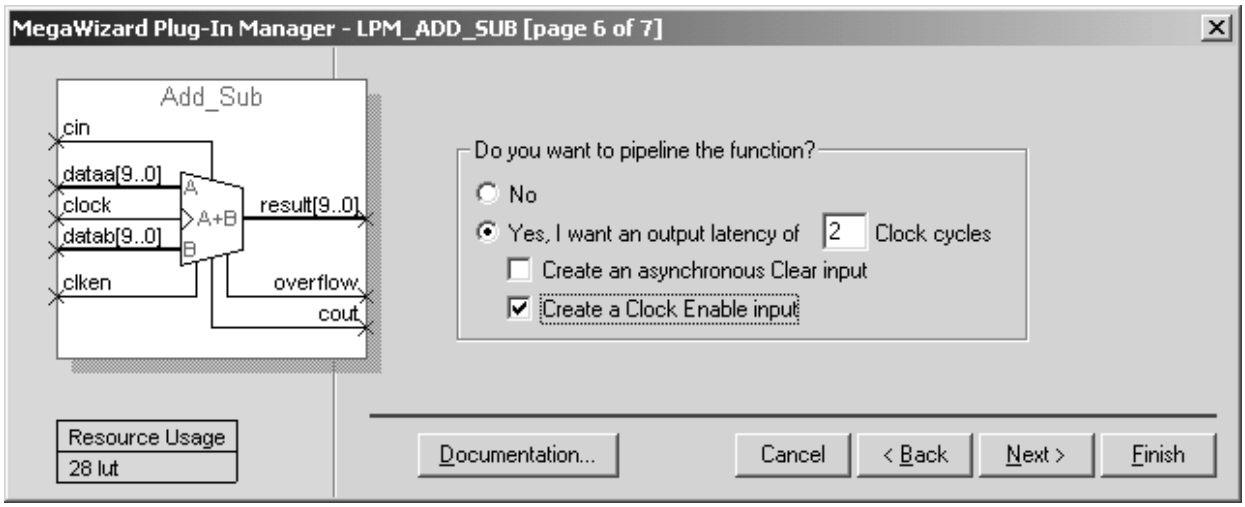
MegaWizard de Altera …

- Génération automatique de mega-fonction
  - ✓Attention : utilisable uniquement avec FPGAs Altera



.. MegaWizard de Altera

- Possibilité de choisir un pipeline



Instructions avancées du VHDL

- Instruction concurrente
  - ✓ for … generate et if … generate

    doit être utilisée dans la zone :

    architecture – **begin** - <ICI> - **end**
- Instruction séquentielle
  - ✓ for … loop

    doit être utilisée dans les zones :

    process – **begin** - <ICI> - **end**

    function – **begin** - <ICI> - **end**

    procedure – **begin** - <ICI> - **end**

Instruction for … generate

- Instruction concurrente, syntaxe :

```
--Le label est obligatoire
Label: for I in Domaine_de_Variation generate
  [ Zone de déclaration ..
begin]
  -- instructions concurentes
end generate;
```

- Cas avec domaine de variation croissant :

```
Label: for I in Entier_A to Entier_B generate
   -- instructions concurentes
end generate;
```

Exemple instruction for ... generate ...

```
library IEEE;
use IEEE.Std_Logic_1164.all;
use IEEE.Numeric_Std.all;

entity Bin_Lin is
  port (Bin_i : in  Std_Logic_Vector(2 downto 0);
        Lin_o : out Std_Logic_Vector(7 downto 0) );
end Bin_Lin;

architecture Flot_Don_Gen of Bin_Lin is
begin
  Boucle: for I in 0 to 7 generate
    Lin_o(I) <= '1' when Unsigned(Bin_i) >= I else 0';
  end generate Boucle;
end Flot_Don_Gen;
```


... exemple instruction for ... generate ...

Ce qui correspond à :

```
architecture Flot_Don of Bin_Lin is
begin
  Lin_o(0) <= '1' when Bin_i >= "000" else '0';
  Lin_o(1) <= '1' when Bin_i >= "001" else '0';
  Lin_o(2) <= '1' when Bin_i >= "010" else '0';
  Lin_o(3) <= '1' when Bin_i >= "011" else '0';
  Lin_o(4) <= '1' when Bin_i >= "100" else '0';
  Lin_o(5) <= '1' when Bin_i >= "101" else '0';
  Lin_o(6) <= '1' when Bin_i >= "110" else '0';
  Lin_o(7) <= '1' when Bin_i >= "111" else '0';
end Flot_Don;
```

Instruction if … generate

- Instruction concurrente, syntaxe :

```
--Le label est obligatoire
Label: if Condition generate
  [ Zone de déclaration ..
begin]
  -- instructions concurentes
end generate;
```


Exemple : for..generate & if…generate

```
-- Description d'un additionneur 4 bits ¦------------
library IEEE;
  use IEEE.Std_Logic_1164.all;

entity Add4 is
  port (Nbr_A_i, Nbr_B_i :
                    in  Std_Logic_Vector(3 downto 0);
        Carry_o  : out Std_Logic;
        Somme_o  : out Std_Logic_Vector(3 downto 0));
end Add4;

architecture Struct of Add4 is
  component Add1
    port (A_i, B_i, C_i : in  Std_Logic;
          S_o, C_o      : out Std_Logic );
  end component;
  for all : Add1 use entity work.Add1(Logique);

  signal Vect_C_s : Std_logic_Vector(3 downto 0);
```

… exemple : for..generate & if…generate

```
begin
  StrucAdd: for I in 0 to 3 generate
    --Premier addtionneur : pas de C_i, add simplifié
    Add1er: if I = 0 generate
      Somme_o(I)  <= Nbr_A_i(I) xor Nbr_B_i(I);
      Vect_C_s(I) <= Nbr_A_i(I) and Nbr_B_i(I);
    end generate;

    Add_N: if I > 0 generate
      I_Add: Add1 port map (A_i => Nbr_A_i(I),
                            B_i => Nbr_B_i(I),
                            C_i => Vect_C_s(I-1),
                            S_o => Somme_o(I),
                            C_o => Vect_C_s(I)  );
    end generate;
  end generate;

  --affectation du Carry de sortie
  Carry_o <= Vect_C_s(3);

end Struct;
```


Instruction for … loop …

- Instruction séquentielle, syntaxe :

```
[Label:] for I in Domaine_de_variation loop
   --zone pour instructions séquentielles
end loop;
```

- Cas avec domaine de variation croissant :

```
[Label:] for I in Entier_A to Entier_B loop
   --zone pour instructions séquentielles
end loop;
```

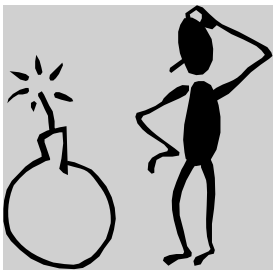
… instruction for … loop …

- Exemple d'utilisation :

```
library IEEE;
use IEEE.Std_Logic_1164.all;

entity Parite is
  port (Vect_i : in  Std_Logic_Vector(7 downto 0);
        Par_o  : out Std_Logic
        );
end Parite;
```


… instruction for … loop …

```
architecture Comport of Parite is   --[a]
begin
 process(Vect_i)
    variable Par_v : Std_logic := '0';
  begin
    for I in 0 to Vect_i'length-1 loop
       if Vect_i(I) = '1' then
           Par_v  := not Par_v;
       end if;
    end loop;
    Par_o <= Par_v;
  end process;
end Comport;
```



ERREUR

… instruction for … loop …

- Ce qui correspond à :

```
architecture Equation of Parite is
begin
  --La parite est calculee avec une equation
  Par_o <= Vect_i(7) xor Vect_i(6) xor
           Vect_i(5) xor Vect_i(4) xor
           Vect_i(3) xor Vect_i(2) xor
           Vect_i(1) xor Vect_i(0);
end Equation;
```

Rem : La variable n'apparaît plus !

Module Master ReCo

Design re-use

Annexes

- Informations pratiques sur le VHDL avec exemples

  http://www.alse-fr.com/tech_corner.html
- Normes IEEE

  http://www.ieee.com/
- Informations des vendeurs d'outils EDA

  http://www.vhdl.org/

Les mots réservés du VHDL93 …

abs
access
after
alias
all
and
architecture
array
assert
attribute

begin
block
body
buffer
bus

case
component
configuration
constant

disconnect
downto

else
elsif
end
entity
exit

file
for
function

generate
generic
group
guarded

… mots réservés du VHDL93 …

if
impure
in
inertial
inout
is

label
library
linkage
loop

map

mod

nand
new
next
nor
not
null

of
on
open
or

others
out

package
port
postponed
procedure
process
protected
pure

range
record

… mots réservés du VHDL93

register
reject
rem
report
return
rol
ror

select
severity
signal
shared
sla

sll
sra
srl
subtype

then
to
transport
type

unaffected
units
until

use
variable

wait
when
while
with

xnor
xor

Historique normes du langage VHDL …

- 1980 : Début du projet financé par le DoD
- 1987 : Standard IEEE Std 1076-1987 (VHDL 87)
- 1993 : Standard IEEE Std 1164-1993 (Std_Logic_1164)
- 1993 : Standard IEEE Std 1076-1993 (VHDL 93)
- 2000 : Standard IEEE Std 1076-2000 (VHDL 2000)
- 2002 : Standard IEEE Std 1076-2002 (VHDL 2002)
- 2008 : Standard IEEE Std 1076-2008 (VHDL 2008)
  - ✓ actuellement pas accepté par les outils EDA!

… historique normes du langage VHDL …

- 1995 : Standard IEEE Std 1076.4
  - ✓ Vital_Primitive et Vital_Timing pour la simulation après P-R
- 1997 : Standard IEEE Std 1076.3
  - ✓ Normalisation des paquetages Numeric_Bit et Numeric_Std
- 1999 : Standard IEEE Std 1076.6
  - ✓ Normalisation de la syntaxe pour la synthèse RTL: Standard for VHDL Register Transfer Level Synthesis
- 2004 : Standard IEEE Std 1076.6 -2004
  - ✓ Evolution de la syntaxe pour la synthèse RTL, pas de changement significatif.

… historique normes du langage VHDL

Autres étapes de l'historique

- 1994 : Approbation ANSI (ANSI/IEEE Std 1076-1993)
- 1996 : Standard IEEE Std 1076.2 (Mathematical Packages)

Autres normalisations

- 1999 : Standard IEEE-1076.1, VHDL-AMS (modélisation mixte)

Références EIVD

- Présentations des collègues
  - ✓Serge Boada
  - ✓Maurice Gaumain
- Manuels VHDL
  - ✓Darryl Gauthey
  - ✓Claude Guex
  - ✓Michel Salamin
  - ✓Yves Sonnay

Références bibliographiques …

[1] Manuel VHDL, synthèse et simulation, Etienne Messerli. HEIG-VD, 2007

[2] VHDL, Introduction à la synthèse logique, Philippe Larcher, Eyrolles, 1997 (Livre simple et facile d'accès, très bien pour les étudiants)

[3] Le langage VHDL, J. Weber & M. Meaudre, Dunod, 2001
(Bon livre pour débuter en VHDL)

[4] VHDL. Méthodologie de design et techniques avancées. Thierry Schneider, Dunod, 2001

[5] VHDL for Engineers, Kenneth L. Short, 2008, Pearson Internationnal (très bien)

[6] VHDL-2008 Just the new stuff, Peter L. Ashenden, Jim Lewis, MK publishers, 2008

[7] Digital System Design with VHDL, 2000, Mark Zwolinski, Prentice Hall

[8] VHDL du langage à la modélisation, Airiau & Bergé & Olive & Rouillard, édition 1990 et 1996, PPUR (référence pour les instructions VHDL)

[9] VHDL Made Easy !, D. Pellerin et D. Taylor, Hardcover, 1996

… références bibliographiques

Normes IEEE:

[10] IEEE Standard VHDL Language Reference Manual (VHDL-1993),
IEEE 1076-1993

[11] IEEE Standard Multivalue Logic System for VHDL Model Interoperability,
IEEE Std 1164-1993

[12] IEEE Standard VHDL Synthesis Packages,
IEEE-1076.3, 1997
Définit en autre le paquetage Numeric_Std, avec les types Unsigned et Signed.

[13] IEEE Standard for VHDL Register Transfer Level Synthesis, IEEE 1076.6-1999

Guides de référence et articles

Guides de références:

[14] The VHDL Golden Reference Guide, compatible IEEE std 1076-2002 disponible chez : Doulos, http://www.doulos.com/

[15] ACTEL HDL Coding, Style guide, ACTEL, Edition 2003 disponible en PDF sur le site http://www.actel.com/

Articles:

[16] Circuits programmables et langages de conception, une évolution en parallèle, C. Guex & E. Messerli, Revue Vision 1998, EIVD

[17] Conception numérique: Description VHDL et synthèse, D. Gauthey & E. Messerli, Revue Vision 2000, EIVD

Références de cours VHDL

[a] Cours Expert VHDL, design & verification, Doulos présenté par Bertrand Cuzeau, septembre 2001, 2002

[b] Le Langage VHDL pour la Conception et le Test des Circuits Logiques Programmables, A.L.S.E., Paris, Bertrand Cuzeau, juin 1999.

[c] Cours : VHDL tout simplement …., Multi Video Designs, M. E. Garcia, mars 2000

[d] Conception numérique : méthode et langage VHDL, Etienne Messerli, HEIG-VD, édition 2000 à 2005

[e] Cours VHDL avancé, Etienne Messerli, HEIG-VD, 2004 à 2007

[f] Cours VHDL avancé INSA, Rennes, France, Etienne Messerli, 2007 à 2009

Liens internet

- Informations pratiques sur le VHDL avec exemples

  http://www.alse-fr.com/tech_corner.html
- Normes IEEE

  http://www.ieee.com/
- Informations des vendeurs d'outils EDA

  http://www.vhdl.org/

FIN présentation VHDL !

Questions