

Design re-use

heig-vd

Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

ReDS
Reconfigurable & Embedded
Digital Systems

Etienne Messerli

Mise à jour le 4 avril 2012

Copyright ©2012 EMI, REDS@HEIG-VD

ReUse, p 1

Défi

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

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 :
 - ✓ structurer l'écriture des descriptions
 - ✓ profiter des performances du langage VHDL
 - ✓ obtenir des descriptions réutilisables

... 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

Descriptions ré-utilisables

- Respecter des règles de méthodologies
 - ✓ Identificateurs, mots réservés, ...
 - ✓ Structure description système combinatoire avec process
 - ✓ Structure description système séquentiel
- Description simple et lisible
- Synthèse automatique doit être garantie
- Commentaires dans la description
- Une seule fonction par module VHDL
 - ✓ Compteur Up/Dn, Registre à décalage, ...

Rappel : Attributs prédéfinis pour tableaux (*array*) ...

Ces attributs sont très utiles pour rendre les descriptions paramétrables:

- ✓ permettent de manipuler des *array*
(exemple : Std_logic_Vector(7 downto 0))
- ✓ nécessaires pour rendre les descriptions paramétrables
(indépendantes de la taille des tableaux)
- ✓ à utiliser avec l'opérateur de concaténation & et la notation par agrégat
- ✓ utiles pour la synthèse, les spécifications et les test benches

... les attributs pour les *array*...

Voici les principaux attributs pour les *array*:

✓'left	: indice de gauche
✓'right	: indice de droite
✓'high	: indice supérieur (MSB)
✓'low	: indice inférieur (LSB)
✓'length	: longueur du tableau (array)
✓'range	: intervalle des indices
✓'reverse_range	: intervalle inverse des indices

... les attributs pour les *array*

- Exemple d'utilisation des attributs :

```
Data : Std_Logic_Vector(7 downto 0);
```

Dans ce cas :	correspond à
Data'left = Data'high	7
Data'right = Data'low	0
Data'length	8
Data'range	7 downto 0
Data'reverse_range	0 to 7

Exemples utilisation attributs *array* ...

- Déclaration d'un signal interne :

```
signal Compteur: Std_Logic_Vector(7 downto 0);  
signal Cpt_Int: Unsigned(Compteur'range);
```

- Décalage à droite :

```
Vect_SHR <= '0' & Vecteur(vecteur'high downto 1);
```

- Rotation à gauche :

```
Vect_ROL <= Vect(Vect'high-1 downto 0) &  
                Vect(Vect'high);
```

- Initialiser un vecteur signé à la valeur min

```
Vect <= (Vect'high => '1', others => '0');
```

Descriptions paramétrables

- Solutions possibles :

- ✓ Taille des vecteurs définie dans les déclarations de l'entité
→description adaptable dans l'architecture
- ✓ 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 les fonctionnalités VHDL utilisées 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);
  ...
end architecture;
```

... 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 benches
- 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 caractere;
```

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 instantiation 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 instantiation 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 contraints

- 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 `Std_Logic_Vector` est par définition non contraint :

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

- Au lieu de spécifier 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      ) ;

  ...
```

Unité SSA: Systèmes séquentiels avancés

Design re-use VHDL avancé

Instructions avancées du VHDL

Instructions avancées du VHDL

- Instruction concurrente
 - ✓ for ... generate
 - ✓ 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 concurrentes
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 addtioneur : pas de C_i, add simplifié
  Addler: 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 StrucAdd;
```

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 ...

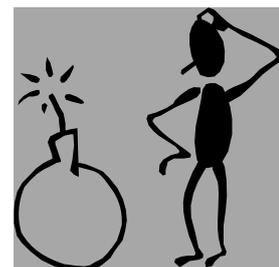
- Exemple d'utilisation :

```
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 !

Questions !

