Unités : CSN & SysLog1

Design re-use

HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Etienne Messerli

Octobre 2021

Défi

- Etre capable de réaliser un système répondant aux caractéristiques (vitesse-performance) dans le temps impartit
  - Travail 100% à la main impossible
  - Nécessite de réutiliser des composants
  - IP souvent nécessaire pour pouvoir utiliser toutes les capacités des FPAGs !
  - Nouvelle méthodologie indispensable

Design re-use ...

- Objectifs :
  - améliorer la productivité de descriptions
  - garantir des descriptions lisibles et fiables
  - disposer de description portables
  - d'où un gain de temps : *Time to Market*
- Comment :
  - structurer l'écriture des descriptions
  - profiter des performances du langage VHDL
    - disposer de descriptions réutilisables

… design re-use

- Ecrire des descriptions paramétrables
  - Paramètres modifiés pour la synthèse
  - Module synthétisé aura une taille fixe
- Il est nécessaire d'avoir:
  - Taille des vecteurs modifiables
  - Utilisation des attributs indispensables
    - Voir présentation "visseries & astuces"
- Plusieurs possibilités de descriptions paramétrables

Descriptions ré-utilsables

- 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 indispensables
- 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 benchs

… 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 cpt_s : unsigned(7 downto 0);
signal val_s : std_logic_vector(cpt_s'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_s <= (vect_s'high => '1', others => '0');
```

Descriptions paramétrables

- Solutions possibles :
  - Taille des vecteurs définie dans les déclarations de l'entité
    - description adaptable dans l'architecture
  - Utilisation de constantes génériques (*generic*)
  - Taille des vecteurs définie dans un paquetage
    - sous-type, constante
  - Utilisation de vecteurs non contraints

=> utilisation des attributs indispensable

Design re-use et Outils

- Certains synthétiseurs ne supportent pas toutes les possibilités de descriptions paramétrables en VHDL.
  - vecteur non contraint : parfois non supporté !
- Solution portable sur tous les outils :
  - taille définie par les déclarations dans l'entité
  - taille définie par une constante générique
  - utilisation de 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 constante générique

- Déclarer des constantes *generic* dans l'entité
- Valeurs des constantes définies lors de chaque instanciation du composant

- Avantage:
  - 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 : positive range 1 to 31 := 4);
  port(clock_i   : in  std_logic;
       reset_i   : in  std_logic;
       enable_i  : in  std_logic;
       data_i    : in  std_logic_vector(N-1 downto 0);
       reg_o     : out std_logic_vector(N-1 downto 0)
       );
end regn_en;
```

```
architecture comport of regn_en is
  signal reg_s : std_logic_vector(reg_o'range);
                                 -- ou (N-1 downto 0)
begin

  process(reset_i, clock_i)
  begin
    if reset_i = '1' then
      reg_s <= (others => '0');
    elsif rising_edge(clock_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 : Positive range 1 to 16 := 4);
    port(clock_i    : in  std_logic;
         reset_i    : in  std_logic;
         enable_i   : in  std_logic;
         data_i     : in  std_logic_vector(N-1 downto 0);
         reg_o      : out std_logic_vector(N-1 downto 0)
         );
  end component;
  for all : regn_en use entity work.regn_en(comport);

...
```

… 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 => 8 )
    port map(clock_i   => clock_i,
             reset_i   => reset_i,
             enable_i  => en_s;
             data_i    => valeur_s,
             reg_o     => sortie_s      );

...
```

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

- Avantages:
  - Sous-type et constante visible partout
  - Structure la description du design

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 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

- Avantage:
  - Il n'est pas nécessaire de déclarer un générique

- Inconvénient:
  - 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(clock_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, clock_i)
  begin
    if reset_i = '1' then
      reg_s <= (others => '0');
    elsif rising_edge(clock_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(clock_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;
  for all : reg_en use entity work. reg_en(comport);

...
```

… 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(clock_i   => clock_i,
              reset_i   => reset_i,
              enable_i  => en_s;
              data_i    => val_s,
              reg_o     => mem_s        );

...
```

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

    pas présentée dans les unités CSN/SysLog2

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;
```

Questions !