26 février 2020
Jean-Noël Bazin, Erwan Libessart (CentraleSupelec)

Aide mémoire VHDL

Exemples et bonnes pratiques pour la conception VHDL orienté FPGA

1.2

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

Table des matières

Listings

1 Avant propos

Les pages qui suivent donnent quelques exemples de descriptions VHDL pour des fonctions logiques élémentaires en vue de la synthèse logique. Les solutions proposées ne sont pas exhaustives mais elles permettent d'assurer un circuit électronique correct à la synthèse.

Avant de se lancer dans l'écriture VHDL, il faut :

- décomposer le circuit en un ensemble de blocs élémentaires les plus simples possibles et avoir un schéma bloc complet et précis du circuit avec pour chaque bloc le nom des entrées-sorties ;
- avoir une idée assez précise de la nature des opérateurs qui seront synthétisés : combinatoire ou synchrone par rapport à une horloge.

On pensera à utiliser des paquetages standards qui permettent d'assurer la portabilité du VHDL d'un outil à l'autre tel que numeric_std.

2 Introduction

Un circuit intégré est réalisé à partir de sa description en termes d'opérateurs logiques, pris dans une bibliothèque, reliés par des connexions ou équipotentielles. Cette description s'appelle la netlist. Cette netlist peut être obtenue, soit par saisie de schéma, soit à partir d'une description en langage de haut niveau, par exemple VHDL.

Le rôle d'un circuit électronique est de traiter des signaux électriques présents sur ses entrées et de générer des signaux cohérents sur ses sorties en fonction du cahier des charges. Le signal est l'objet central du langage VHDL :

- sa déclaration se traduit en synthèse par une équipotentielle ;
- son état est défini par le symbole d'affectation (connexion au sens électronique) " <= ".

ex : C <= A and B ; (se lit C reçoit A et B) sera synthétisé par :

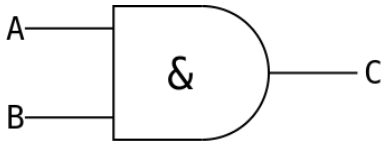

FIGURE 1 – Porte ET

Un signal doit être déclaré et typé avant d'être affecté.

3 À savoir

Le VHDL n'est pas sensible à la casse (minuscule/majuscule).

4 Le flow de conception FPGA

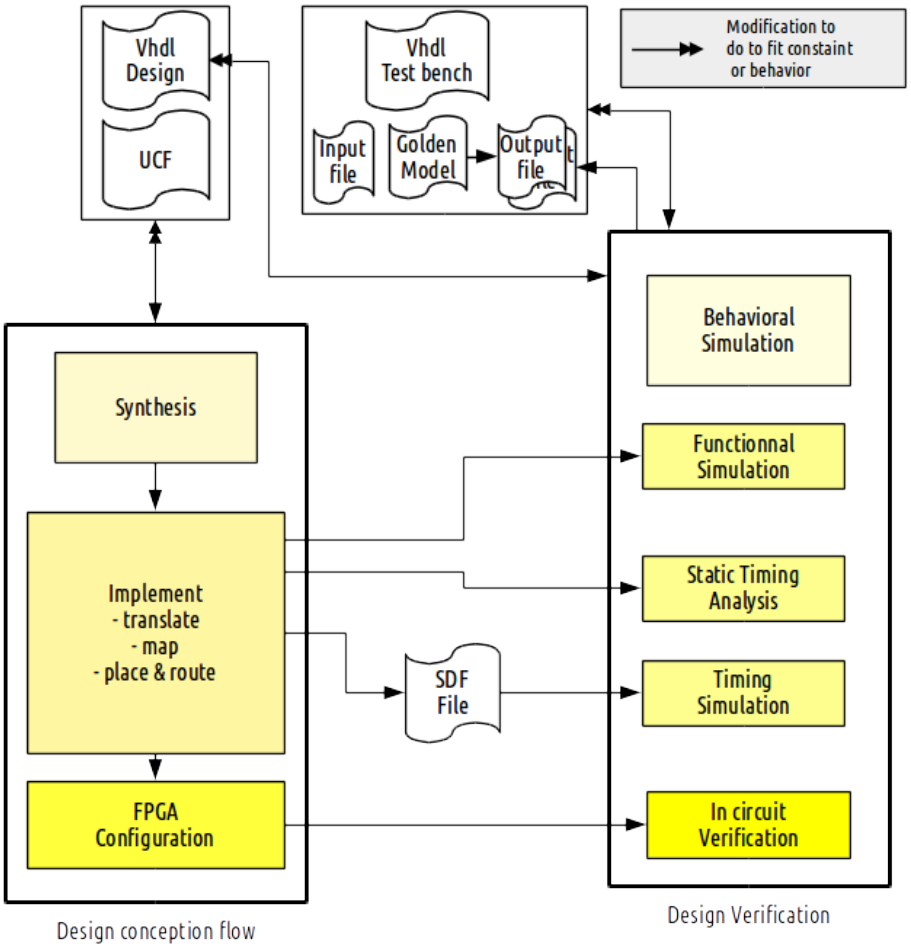

FIGURE 2 – Flow de conception FPGA (source : `https://www.xilinx.com/itp/xilinx10/isehelp/ise_c_fpga_design_flow_overview.htm`)

La conception d'un circuit, et de ses sous module, suis en général la procédure suivante quand on cible un FPGA :

- Description d'un golden model (VHDL non synthétisable, C, Python, Matlab) de référence ;
- Conception sur papier d'une architecture ;
- Description de cette architecture en VHDL synthétisable ;
- Écriture d'une procédure de test (test bench, fichiers d'entrée, script d'automatisation ...) ;
- Simulation comportementale
- Si la simulation comportementale se déroule correctement, on procède à la synthèse
- Simulation post synthèse
- Si la simulation post synthèse se déroule correctement, on procède à l'implémentation (translate, map, place and route)
- Simulation post place and route
- Si la simulation post place and route se déroule correctement, on procède au passage sur FPGA
- Vérification du bon fonctionnement du circuit sur FPGA

Si une des Vérification retourne de mauvais résultats, il faut procéder la plupart du temps à des modifications dans la description VHDL du circuit.

5 Les types en VHDL

Il existe différents types de signaux en fonction des besoins. Chaque type est défini dans une bibliothèque.

5.1 std_logic

Le type std_logic est utilisé pour déclarer des signaux de 1 bit. Il fait partie de la bibliothèque standard std_logic_1164. Ce bit peut prendre 9 états différents :

- '0' : état logic bas ;
- '1' : état logic haut ;
- 'Z' : haute impédance ;
- 'U' : non définie (Undefined) ;
- 'X' : conflit (drivers multiples) ;
- 'W' : Inconnu (Weak) ;
- 'H' : Inconnu, probablement haut (High) ;
- 'L' : Inconnu, probablement bas (Low) ;
- '-' : Don't care

Seulement les trois premiers états sont cependant synthétisables. Donc si en simulation un des autres états est détecté, le design risque de ne pas se comporter comme prévu sur FPGA. C'est l'avantage de ce type, il est ainsi plus facile de débugger un circuit lors des simulations.

Déclaration :

Listing 1 – Type std_logic

```
1 signal C : std_logic;
```

5.2 std_logic_vector

Ce type est utilisé pour représenter des signaux composés de plusieurs bits, par exemple un bus, ou un signal de donnée. Il fait partie de la bibliothèque standard std_logic_1164. Chacun des bits composant ce vecteur possède les mêmes propriétés qu'un std_logic. Il faut impérativement indiquer une taille à ce vecteur lors de sa déclaration :

Listing 2 – Type std_logic_vector

```
1 signal C : std_logic_vector(7 downto 0);  -- ici le signal est compose de 8 bits
```

Comme c'est un tableau, il est possible d'accéder à chaque élément qui le compose, en lecture comme en écriture, via des indices de type integer :

Listing 3 – Accès aux bits d'un std_logic_vector

```
1 signal D : std_logic;
2 signal E : std_logic_vector(3 downto 0);
3 .
4 .
5 .
6 C <= "11001001";
7 D <= C(3); -- 1
8 E <= C(4 downto 1); --"0100"
```

5.3 enumerate

Ce type est utilisé notamment pour décrire les différents états que prendre un automate (machine d'état). La déclaration est de haut niveau, c'est à dire qu'elle ne fait pas intervenir de taille de vecteur etc. C'est le synthétiseur qui se charge de convertir les états en grandeur physique (ie. signaux électriques). Déclaration :

Listing 4 – Type enumerate

```
type T_State is(
                ST_Reset    ,
                ST_Wait     ,
                ST_Read
                );
```

5.4 unsigned/signed

Ces types sont utilisés pour les opérations arithmétiques. Ils sont définis dans la librairie standard numeric_std. Les signaux *signed* sont représentés en complément à 2. De la même manière que pour les vecteurs std_logic_vector, il faut impérativement indiquer une taille à ces vecteurs lors de leur déclaration :

Listing 5 – Types unsigned et signed

```
signal U : unsigned(7 downto 0);
signal S : signed(7 downto 0);
```

Il faut utiliser ces signaux en cas d'opérations arithmétiques, sinon il est préférable d'utiliser des std_logic_vector. De la même manière que pour les std_logic_vector il est possible d'accéder aux bits qui les composent via un indice de type integer.

5.5 natural/integer

Ces types de signaux peuvent aussi être utilisés pour des opérations arithmétiques, notamment pour la réalisation de compteur/décompteur. Le type natural est utilisé pour des signaux de données positives, integer pour des données positives et/ou négatives. Cette fois il n'est pas la peine de spécifier une taille de vecteur en nombre de bit, mais une plage de valeur lors de la déclaration. C'est le type utilisé en temps qu'index de vecteurs et de tableaux :

Listing 6 – Types natural et integer

```
signal N : natural range 0 to 17;
signal I : integer range -4 to 29;
```

Si on indique pas de plage de valeur, par défaut la plage est de $0 \rightarrow 2^{32}-1$ pour un *naturel* et $-2^{31} \rightarrow 2^{31}-1$ pour un *integer*. Il est donc important, pour éviter d'utiliser plus de ressource que nécessaire de contraindre correctement la plage de valeur possible.

5.6 array

Le type array sert à construire des tableaux de signaux, ou de tableau (tableau à 2 dimensions et plus). Le plus souvent, ce type sert à décrire des mémoires : tableau à 1 dimension de signaux. Dans le cas de tableaux de dimension supérieure ou égale à 2 se pose le problème de l'utilisation des ressources. En effet pour ce type de tableau un nombre très grand de multiplexeur est nécessaire. Exemple de déclaration :

Listing 7 – Types array

```
type ram_type is array (5 downto 0) of std_logic_vector(7 downto 0);
```

On définit un type ram_type composé de 6 signaux de 8 bits. Un signal de ce type est alors adressable aussi bien en lecture qu'en écriture. L'index utilisé doit être de type *integer*.

Listing 8 – Accès aux élément d'un tableau

```
signal Tableau          : ram_type;
signal indice           : integer range 0 to 5;
signal SC_sortieTableau : std_logic_vector(5 downto 0);
.
.
.
Tableau(indice)  <= "100011";
SC_sortieTableau <= Tableau(indice);
```

5.7 record

Le type record sert à construire des structures de données pouvant être hétérogènes. C'est l'équivalent des *struct* en C.

Listing 9 – Record

```
type my_record is record
    value_1 : std_logic_vector(7 downto 0);
    value_2 : std_logic;
    value_3 : integer range -12 to 14;
    value_4 : my_type; -- on peut inclure ses propres types dans les records
end record
  .
  .
  .
signal SC_dataInput : my_record;
  .
  .
  .
SC_dataInput.value_1 <= "01001101";
SC_dataInput.value_3 <= 7;
O_output <= SC_dataInput.value_2;
```

Il est possible d'utiliser des record dans des tableaux :

Listing 10 – Record dans un tableau

```
type my_array is array (12 downto 0) of my_record;
  .
  .
  .
signal SR_Data : my_array;
  .
  .
  .
SR_Data(4).value_1 <= "01001101";
```

5.8 line

Il est possible de manipuler des fichiers textes en VHDL pour faciliter les simulations. La manipulation de fichiers n'est évidement pas synthétisable, ça n'a pas sens sur FPGA ou ASIC. Le type line est le type qui permet comme son nom l'indique de manipuler des lignes.

L'utilisation de ce type est expliquée dans la section 9.2

6 Les opérateurs

6.1 Opérateur d'affectation

L'opérateur d'affection est : <= :

Listing 11 – Affectation

```
1 b <= a; -- a et b doivent etre de meme type
```

Il s'utilise pour connecter un signal avec un autre signal de même type ou avec une opération dont le résultat est de même type.

6.2 Opérateurs logiques

Les opérateurs logiques AND OR NOT NAND NOR XOR XNOR permettent de faire des opérations entre signaux de type std_logic :

Listing 12 – Opérateurs logiques

```
1 c <= a and b;
2 d <= a or b;
3 e <= not a;
4 f <= a nand b;
5 g <= a nor b;
6 h <= a xor b;
7 i <= a xnor b;
```

6.3 Opérateurs relationnels

Les opérateurs relationnels servent à comparer les valeurs de signaux de mêmes type. La valeur retournée est un booléen, servant notamment pour les structures conditionnelles.

=	égal
/=	différent
<	inférieur
>	supérieur
<=	inférieur ou égal
>=	supérieur ou égal

TABLE 1 – Opérateurs relationnels

6.4 Opérateurs arithmétiques

+	addition	synthétisable
-	soustraction	synthétisable
*	multiplication	synthétisable
/	division	synthétisable (mais peu optimisée)
mod	modulo	non synthétisable
rem	reste	non synthétisable
**	exponentiel	synthétisable dans certains cas

TABLE 2 – Opérateurs arithmmétiques

L'opérateur exponentiel est synthétisable notamment lors de la définition de taille de signaux ou de tableaux lors de leur déclaration :

Listing 13 – Cas de l'opérateur exponentiel synthétisable

```
1 type ram_type is array (2**Addr-1 downto 0) of std_logic_vector(7 downto 0);
```

6.5 Opérateurs de concaténation

L'opérateur de concaténation & permet de construire un vecteur de bits en concaténant plusieurs sous vecteurs :

Listing 14 – Concaténation

```
1 signal a : std_logic_vector(3 downto 0); -- 4 bits
2 signal b : std_logic_vector(3 downto 0); -- 5 bits
3 signal c : std_logic_vector(3 downto 0); -- 9 bits
4 --
5 a <= "0000";
6 b <= "11111";
7 c <= a & b; -- c = "000011111"
```

6.6 Les opérateurs en fonction du type

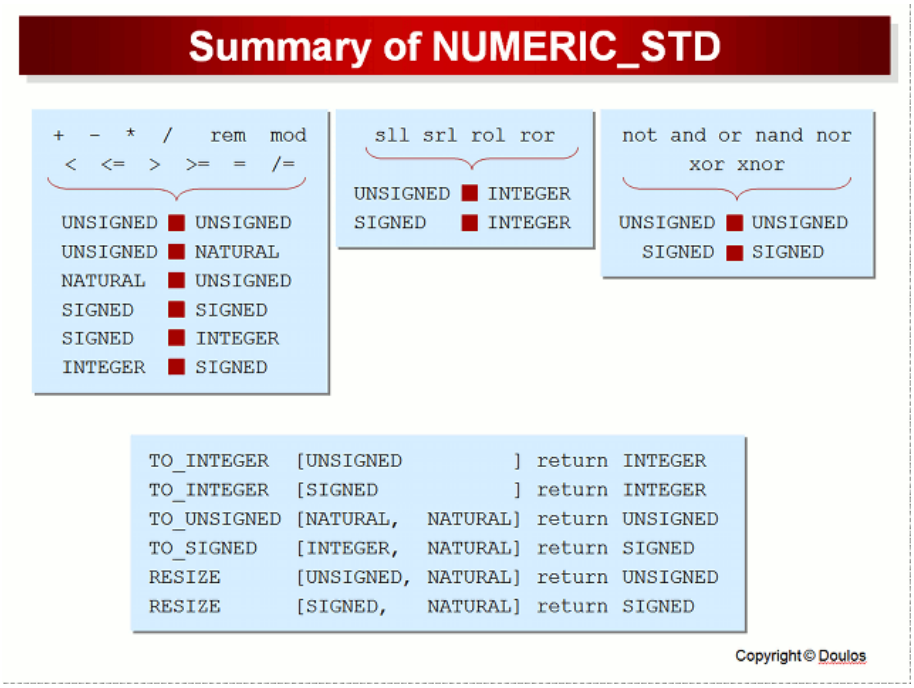

FIGURE 3 – Les opérateurs autorisés en fonction du type(source : Doulos)

7 Constitution d'un fichier VHDL

7.1 Les commentaires et entête

Les commentaires sont essentiels à la bonne compréhension d'un code. Le délimiteur des commentaire est "–" (deux tirets classiques à la suite). Pour commenter plusieurs lignes, il faut placer – à chaque ligne. Il est possible de placer un commentaire à la fin d'une ligne de code.

Exemple :

Listing 15 – Commentaires dans le code

```
1 -- ceci est la premiere ligne d'un commentaire en VHDL
2 -- ceci est la deuxieme ligne d'un commentaire en VHDL
3 signal U : unsigned(7 downto 0);  -- ici le signal est compose de 8 bits
```

Il est aussi important dans l'entête d'un fichier de donner des renseignements sur l'auteur du code, la version, la date de création/modification etc.

7.2 Les bibliothèques

Les bibliothèques contiennent principalement les définitions des types et des opérateurs arithmétiques qui le sont associés. Les bibliothèques à privilégier sont les bibliothèques standards de l'IEEE. Declaration :

Listing 16 – Les bibliothèques de base

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 use ieee.numeric_std.all;
```

7.3 Entity

La partie "entity" est la partie dans laquelle on va décrire les entrées sorties du composant. On peut aussi, via les "generic" définir des paramètres de configuration du composant. Par exemple la taille de certains signaux internes ou entrées sorties. Exemple d'entité :

Listing 17 – Entity

```
1  entity EntityExample is
2    generic(
3      G_Size1 : integer := 4;
4      G_Size2 : integer := 6
5    );
6    port(
7      I_Input1   : in  std_logic;
8      I_Input2   : in  std_logic;
9      I_Input3   : in  std_logic_vector(G_Size1 -1 downto 0);
10     I_Input4   : in  std_logic_vector(G_Size2 -1 downto 0);
11     O_Outpout1 : out std_logic_vector(G_Size1 -1 downto 0);
12     O_Outpout2 : out std_logic_vector(G_Size2 -1 downto 0)
13   );
14 end entity EntityExample;
```

L'entity sert à décrire les interfaces d'un composant :

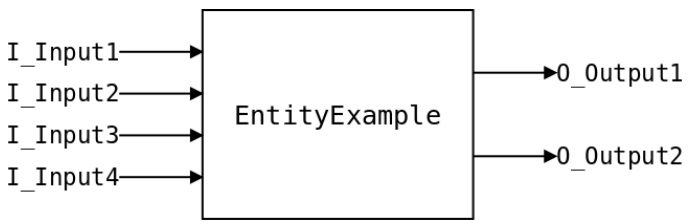

FIGURE 4 – entity

7.4 Architecture

La partie architecture est dédiée à la description du composant en lui même, c'est à dire son fonctionnement, sa constitution.

Listing 18 – Architecture

```
architecture archi_EntityExample of EntityExample is
    -- signal/constant/component declaration
begin
     -- componant structure / behavior
end; -- end of file
```

7.5 Exemple d'un fichier complet

Listing 19 – Un fichier VHDL complet

```
--------------------------------------------------------------------
-- Title              : VHDLExample
-- Project            : Project example
-- creation           : 2014-04-01
-- File               : VHDLExample.vhd
-- Author             : name @mail
-- Company            : Telecom Bretagne
-- Last update        : 2014-04-01
-- Platform           : constructor ; model (Xilinx, altera...)
-- brief Description  : This module does/performs/computes stuffs
--------------------------------------------------------------------

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

entity VHDLExample is
  generic(
    G_AddWidth : integer := 4;
    G_WordSize : integer := 6
  );
  port(
    I_sys_clk : in  std_logic;
    I_we      : in  std_logic;
    I_adr     : in  std_logic_vector(G_AddWidth -1 downto 0);
    I_di      : in  std_logic_vector(G_WordSize -1 downto 0);
    O_do      : out std_logic_vector(G_WordSize -1 downto 0)
  );
end VHDLExample;

architecture archi_VHDLExample of VHDLExample is

  type ram_type is array(2**G_AddWidth-1 downto 0)of std_logic_vector(G_WordSize-1 downto 0);
  signal RAM   :     ram_type;
  signal SR_do : out std_logic_vector(G_WordSize -1 downto 0);

begin

  process (I_sys_clk)
  begin
    if (I_sys_clk'event and I_sys_clk = '1') then
      if (I_we = '1') then
        RAM(to_unsigned(I_adr)) <= I_di;
      end if;
      -- synchronous output RAM : uses block RAM in FPGA instead of LUT :
      SR_do <= RAM(to_unsigned(I_adr));
    end if;
  end process;

  O_do <= SR_do;

end archi_VHDLExample;
```

8 Exemples de VHDL synthétisable

Les exemples présentés dans ce document sont tous inférables sur les ressources dédiées des FPGA (adder, counter, RAM...). Il s'agit de codes VHDL synthétisables donc utilisable pour décrire un circuit que l'on veut implanter sur FPGA ou ASIC.

8.1 Les process

Les process en VHDL servent à décrire le fonctionnement d'une partie d'un circuit, notamment les circuits synchrones (utilisant une horloge) ou combinatoires complexes. Dans les process il est possible d'utiliser des structures conditionnelles telles que les classiques *if/elsif/else* et les *case*.

Un process possède une liste de sensibilité, c'est à dire une liste de signaux d'entrée. Le process réagit aux changement d'état des signaux qui sont présent dans la liste de sensibilité. Il est crucial de renseigner correctement cette liste de sensibilité.

Par contre un signal uniquement mis à jour dans le process n'a pas besoin d'être présent dans liste de sensibilité.

Process synchrone

Un process synchrone a comme son nom l'indique un fonctionnement synchronisé par les événements affectant un signal. On appel ce signal l'horloge, et le plus souvent les process de ce type sont synchronisés sur le front montant de l'horloge, c'est à dire le passage le l'état 0 à l'état 1. Le process met à jour les signaux qu'il pilote uniquement au moment du front montant, quelque soit l'évolution des signaux utilisés dans ce process. De ce fait la liste de sensibilité d'un process synchrone est un peu particulière. Elle ne contient que l'horloge et le reset dans le cas ou ce dernier est asynchrone. Même si des signaux sont lus ou utilisés dans la partie synchrone, il n'est pas nécessaire de les mettre dans la liste de sensibilité.

Listing 20 – Process synchrone

```
1 process(I_Clock,I_Reset)--liste de sensibilite entre parentheses, composee d'une horloge et d'un reset
2 begin
3   if I_Reset = '1' then -- reset asynchrone actif a l'etat haut
4     -- partie asynchrone : ce qui se passe si le reset est actif
5   elsif(rising_edge(I_Clock)) then
6     -- partie synchrone : ce qui se passe au moment d'un front montant d'horloge
7   end if;
8 end process;
```

Process asynchrone

Le process asynchrone, étant par définition combinatoire, doit réagir immédiatement au changement d'état d'un de ses signaux. Il est donc impératif de renseigner exhaustivement la liste de sensibilité.

Listing 21 – Process asynchrone

```
1  process(S1,S2,S3,S3,S5,S6)
2  begin
3    -- description du circuit utilisant les signaux present dans la liste de sensibilite. Par exemple :
4    if(S1 = '1')then
5      S7 <= S5 xor S6; -- S7 n'a pas a etre dans la liste de sensibilite
6    elsif(S2 = '1')then
7      S7 <= S3;
8    else
9      S7 <= S4;
10   end if;
11 end process;
```

Process implicite

Il est possible de décrire le fonctionnement d'un circuit en dehors d'un process. On appel cela un process implicite. On utilise les process implicite quand il n'est pas nécessaire d'utiliser un process en temps que tel. Mettre une connexion (affectation) permanente entre deux signaux dans un process explicite ne présente aucun intérêt, on la fait en dehors :

Listing 22 – Process implicite

```
process(...)
begin
    ...
end process;

A <= B;

process(...)
begin
    ...
end process;
```

8.2 Transcription d'une table de vérité

On doit toujours s'assurer que toutes les branches soient affectées. On sait ce qu'on doit faire quelque soit la valeur du signal d'entrée.

A(2)	A(1)	A(0)	S(2)	S(1)	S(0)
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	1
0	1	1	0	1	0
1	0	0	1	1	0
1	0	1	1	1	1
1	1	0	1	0	1
1	1	1	1	0	0

Processus explicite "if, elsif, else"

Listing 23 – Process et structures conditionnelles *if elsif else*

```
signal A : std_logic_vector(2 downto 0);
signal S : std_logic_vector(2 downto 0);
--
process(A)
begin
    if (A = "000") then
        S <= "000";
    elsif (A = "001") then
        S <= "001";
    elsif (A = "001") then
        S <= "011";
    elsif (A = "010") then
        S <= "011";
    elsif (A = "011") then
        S <= "010";
    elsif (A = "100") then
        S <= "110";
    elsif (A = "101") then
        S <= "111";
    elsif (A = "110") then
        S <= "110";
    elsif (A = "111") then
        S <= "100";
    else
        S<="000";
    end if;
end process;
```

Processus explicite : avec "case, when, when others"

Listing 24 – Process et structure conditionnelle *case*

```
Process(A)
Begin
```

```
    Case A is
        when "000"  => S <= "000";
        when "001"  => S <= "001";
        when "001"  => S <= "011";
        when "010"  => S <= "011";
        when "011"  => S <= "010";
        when "100"  => S <= "110";
        when "101"  => S <= "111";
        when "110"  => S <= "110";
        when "111"  => S <= "100";
        when others => S <= "000";
    end case;
end process ;
```

Processus implicite : défini dans le domaine concurrent

Affectation non conditionnelle :

Listing 25 – Équivalent en équations logiques

```
S(2) <=    (A(2) and not A(1) and not A(0))
        or (A(2) and not A(1) and     A(0))
        or (A(2) and     A(1) and not A(0))
           (A(2) and     A(1) and     A(0));

S(1) <=    (not A(2) and     A(1) and not A(0))
        or (not A(2) and     A(1) and     A(0))
            or (A(2) and not A(1) and not A(0))
            or (A(2) and not A(1) and     A(0)) ;

S(0) <=    (not A(2) and not A(1) and     A(0))
        or (not A(2) and     A(1) and not A(0))
            or (A(2) and not A(1) and     A(0))
            or (A(2) and     A(1) not     A(0));
```

Affectation conditionnelle :

Listing 26 – Process implicite et structure conditionnelle *when else*

```
S <= "000" when(A = "000")else
     "001" when(A = "001")else
     "011" when(A = "010")else
     "010" when(A = "011")else
     "110" when(A = "100")else
     "111" when(A = "101")else
     "110" when(A = "110")else
     "100" when(A = "111")else
     "000";
```

Ce type d'affectation est très pratique et lisible pour l'affectation d'un signal binaire.

Listing 27 – Structure conditionnelle *when else* avec condition plus complexe

```
S <= '1' when(A = "01110" or B = '1' and C = '1')else '0';
```

Affectation sélective :

Listing 28 – Process implicite et structure conditionnelle *with select*

```
with A select
S <= "000" when "000",
     "001" when "001",
     "011" when "010",
     "010" when "011",
     "110" when "100",
     "111" when "101",
     "110" when "110",
     "100" when "111",
     "000" when others;
```

NB : toujours identifier un cas par défaut à l'aide de :
ELSE
Ou
WHEN OTHERS

8.3 Conversion de type avec la bibliothèque numeric_std

Il est nécessaire de faire appel à un paquetage extérieur (de préférence normalisé) qui définit les opérateurs. C'est le rôle de la bibliothèque *numeric_std*. On pourra alors facilement passer d'un format à un autre.

- ***to_integer :*** Transforme un vecteur de type signed ou unsigned en integer ;
- ***to_unsigned :*** Transforme un vecteur de type signed ou un signal de type integer, en un vecteur de type unsigned. Permet également d'étendre le format d'un vecteur de type unsigned ;
- ***to_signed :*** Même propriété que la fonction to_unsigned pour les nombres signés.

SYNTAXE :

Listing 29 – conversions de types grâce aux fonctions de la bibliothèque numeric_std

```
interger_Signal <= to_integer(unsigned_Signal);
interger_Signal <= to_integer(signed_Signal);
unsigned_Signal <= to_unsigned(natural , length_of_unsigned_Signal);
unsigned_Signal <= to_unsigned(integer , length_of_unsigned_Signal);
signed_Signal   <= to_signed(natural   , length_of_signed_Signal);
signed_Signal   <= to_signed(integer   , length_of_signed_Signal);

signal B : signed(3 downto 0) ; -- taille de 4 bits
A < = to_unsigned(B , 5) ;  -- 5 est la taille du vecteur A
```

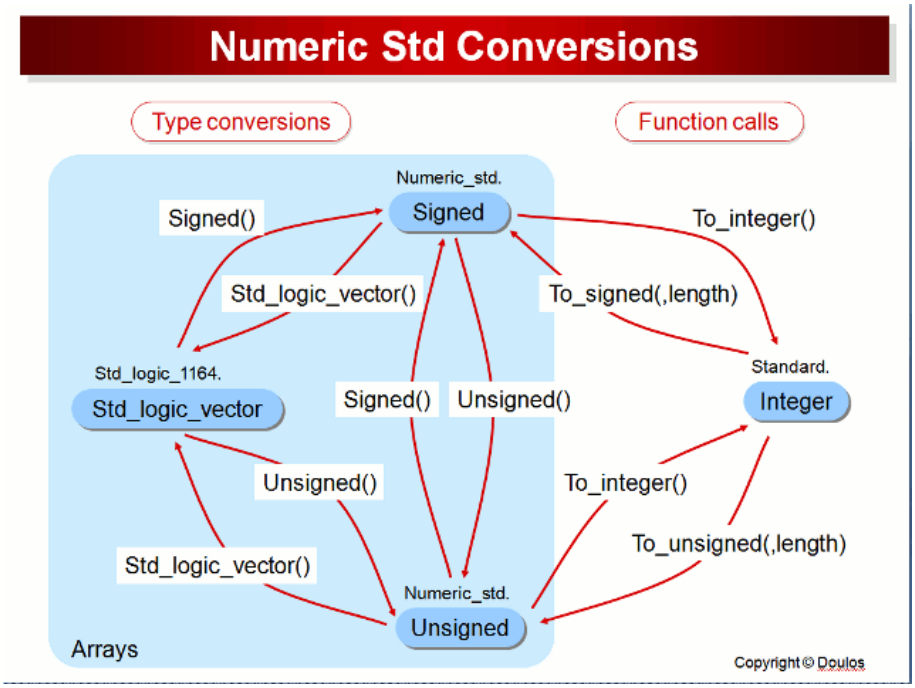

FIGURE 5 – Conversion avec les fonctions du package numeric_std (source : Doulos

8.4 Affectation de valeurs avec la librairie numeric_std

Listing 30 – Affectation de valeurs décimales à des signaux binaire grâce à numeric_std

```
signal AU : unsigned(3 downto 0); -- 4 bits
signal BU : unsigned(3 downto 0); -- 4 bits
signal SU : unsigned(7 downto 0); -- 8 bits
signal AS : signed(3 downto 0);
signal BS : signed(3 downto 0);
signal SS : signed(7 downto 0);
--
AU <= to_unsigned(15, 4);
BU <= to_unsigned(15, 4);
AS <= to_signed(7, 4);
BS <= to_signed(-8, 4);
```

```
12 AI <= -16;
13 BI <= -16;
```

8.5 Multiplexeur

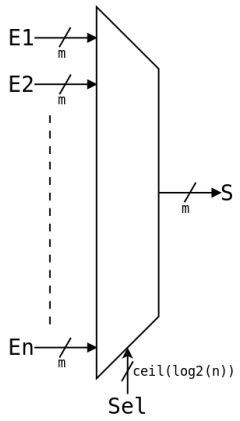

FIGURE 6 – Multiplexeur

Il existe beaucoup de manières différentes de décrire un multiplexeur. En voici quelques-unes.

Listing 31 – Multiplexeur décrit avec un case

```
1  process(E1,E2,...,En,Sel)
2  begin
3      case Sel is
4          when "0...00" =>
5              S <= E1;
6          when "0...01" =>
7              S <= E2;
8          .
9          .
10         .
11         when others =>
12             S <= En;
13     end case;
14 end process;
```

Listing 32 – Multiplexeur décrit la structure when else

```
1
2  S <= E1 when(Sel = "0...00")else
3       E2 when(Sel = "0...01")else
4       .
5       .
6       En-1 when (Sel = "......") else
7       En;
```

8.6 Additionneur

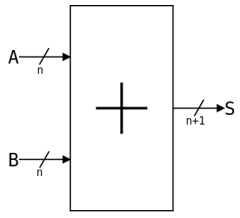

FIGURE 7 – Additionneur

Sans carry-out :

Listing 33 – Additionneur sans carry out

```
signal A : signed (3 downto 0);
signal B : signed (3 downto 0);
signal S : signed (3 downto 0);
--
S <= A + B ;
```

Avec carry-out :

Listing 34 – Additionneur avec carry out

```
signal A : signed (3 downto 0);
signal B : signed (3 downto 0);
signal S : signed (4 downto 0);
--
S <= to_signed (to_integer(A) , 5) + to_signed (to_integer(B) , 5);
-- OR, with sign bit extension
S <= (A(3)&A)+(B(3)&B);
```

8.7 Multiplieur

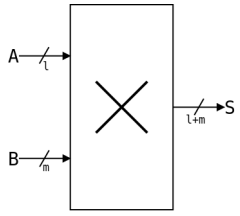

FIGURE 8 – Multiplieur

Listing 35 – Multiplieur

```
signal AU : unsigned(3 downto 0); -- 4 bits
signal BU : unsigned(3 downto 0); -- 4 bits
signal SU : unsigned(7 downto 0); -- 8 bits
signal AS : signed(3 downto 0);
signal BS : signed(3 downto 0);
signal SS : signed(7 downto 0);
signal AI : integer range -16 to 15;
signal BI : integer range -16 to 15;
signal SI : integer range -512 to 511;
--
SU <= AU * BU;
SS <= AS * BS;
SI <= AI * BI;
```

8.8 Registre simple

Exemple de registre à reset asynchrone et enable synchrone.

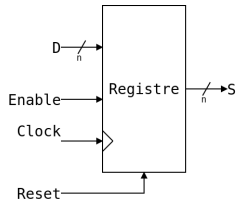

FIGURE 9 – Registre

Listing 36 – Registre

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

```
3
4  entity Registre is
5    port(
6      I_Clock  : in  std_logic;
7      I_Reset  : in  std_logic;
8      I_Enable : in  std_logic;
9      I_D      : in  std_logic_vector(7 downto 0);
10     O_S      : out std_logic_vector(7 downto 0)
11   );
12 end Registre;
13
14 architecture archi of Registre is
15 begin
16
17   process(I_Clock, I_Reset)
18   begin
19     if(I_Reset = '1')then
20       O_S <= (others => '0');  -- notation par agregat
21     elsif(rising_edge(I_Clock))then
22       if(I_Enable = '1')then
23         O_S <= I_D;
24       end if;
25     end if;
26   end process;
27
28 end archi;
```

Une bascule est un registre dont les entrées/sorties sont des *std_logic*.

8.9 Registre à décalage

Exemple de registre à décalage avec chargement parallèle.

Listing 37 – Registre à décalage

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity RegistreDecalage is
5    port(
6      I_Clock  : in  std_logic;
7      I_Reset  : in  std_logic;
8      I_Enable : in  std_logic;
9      I_Load   : in  std_logic;
10     I_D      : in  std_logic_vector(7 downto 0);
11     O_S      : out std_logic
12   );
13 end RegistreDecalage;
14
15 architecture archi of RegistreDecalage is
16
17   signal SR_Reg : std_logic_vector(7 downto 0);
18
19 begin
20
21   process(I_Clock, I_Reset)
22   begin
23     if(I_Reset = '1')then
24       SR_Reg <= (others => '0');
25     elsif(rising_edge(I_Clock))then
26       if(I_Enable = '1')then
27         if(I_Load = '1')then
28           SR_Reg <= I_D;
29         else
30           SR_Reg(6 downto 0) <= SR_Reg(7 downto 1);
31           SR_Reg(7)          <= '0';
32         end if;
33       end if;
34     end if;
35   end process;
36
37   O_S <= SR_Reg(0);
38
```

```
39 end archi;
```

8.10 Compteurs

Exemple de compteur modulo 112 avec reset asynchrone, enable et load.

Listing 38 – Compteur avec le type unsigned

```
1
2  ---- EN UTILISANT LE TYPE UNSIGNED
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.NUMERIC_STD.all;
7
8  entity Counter is
9
10   port(
11     I_Clock  : in  std_logic;
12     I_Reset  : in  std_logic;
13     I_Load   : in  std_logic;
14     I_DLoad  : in  std_logic_vector(7 downto 0);
15     I_Enable : in  std_logic;
16     O_Count  : out std_logic_vector(7 downto 0)
17   );
18
19 end entity Counter;
20
21 architecture archi_Counter of Counter is
22
23   signal SR_Counter : unsigned(7 downto 0);
24
25 begin
26
27   process (I_Clock, I_Reset) is
28   begin
29     if(I_Reset = '1')then
30       SR_Counter <= (others => '0');
31     elsif(rising_edge(I_Clock))then
32       if(I_Enable = '1')then
33         if(I_Load = '1')then
34           SR_Counter <= unsigned(I_DLoad);
35         elsif(SR_Counter >= to_unsigned(111 ,8))then
36           SR_Counter <= (others => '0');
37         else
38           SR_Counter <= SR_Counter + 1;
39         end if;
40       end if;
41     end if;
42   end process;
43
44   O_Count <= std_logic_vector(SR_Counter);
45
46 end architecture archi_Counter;
```

Listing 39 – Compteur avec le type integer

```
1
2  ---- EN UTILISANT LE TYPE INTEGER
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.NUMERIC_STD.all;
7
8  entity Counter is
9
10   port(
11     I_Clock  : in  std_logic;
12     I_Reset  : in  std_logic;
13     I_Load   : in  std_logic;
14     I_DLoad  : in  std_logic_vector(7 downto 0);
15     I_Enable : in  std_logic;
```

```
16      O_Count  : out std_logic_vector(7 downto 0)
17    );
18
19  end entity Counter;
20
21  architecture archi_Counter of Counter is
22
23    signal SR_Counter : integer range 0 to 111;
24
25  begin
26
27    process (I_Clock, I_Reset)
28    begin
29      if(I_Reset = '1')then
30        SR_Counter <= 0;
31      elsif (rising_edge(I_Clock))then
32        if(I_Enable = '1')then            -- Enable counter
33          if(I_Load = '1')then            -- Load counter with input data
34            SR_Counter <= to_integer(unsigned(I_DLoad));
35          elsif(SR_Counter >= 111)then  -- Reset counter value at
36            SR_Counter <= 0;
37          else
38            SR_Counter <= SR_Counter + 1;
39          end if;
40        end if;
41      end if;
42    end process;
43
44    O_Count <= std_logic_vector(to_unsigned(SR_Counter , 8));
45
46  end architecture archi_Counter;
```

8.11 RAM

La plupart des FPGA intègrent des blocs spécifiques dédiés à implémenter des RAM de manière optimale. Il ne s'agit pas de LUT spéciales, mais directement de RAM qui peuvent être utilisées si le VHDL est écrit correctement. Un exemple est donné dans la section 7.5.

8.12 Machine à états finis (FSM)

À partir d'une complexité même relativement faible, un circuit est souvent découpé en plusieurs blocs principaux. La plupart du temps une partie opérative, contenant tous les opérateurs arithmmétiques, les unités de traitement etc, et une partie de contrôle, réalisée par une machine à états.

Une machine à états permet de :

- dire dans quel état se trouve le système
- dire dans quel état se trouve le système
- piloter les signaux de contrôle de la partie opérative et le cas échéant des sorties en fontion de l'état présent et potentiellement d'entrées.

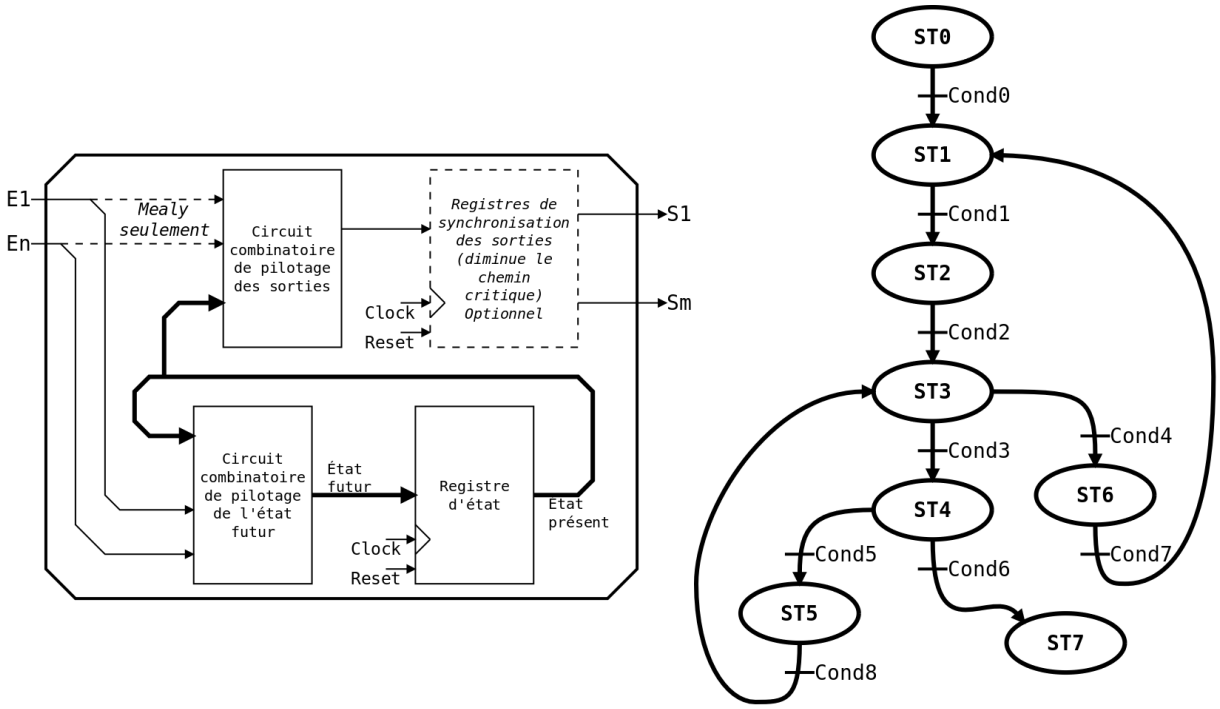

FIGURE 10 – FSM

Listing 40 – Exemple de FSM

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity fsm is
6    port (
7      I_Clock : in  std_logic;
8      I_reset : in  std_logic;
9      I_1     : in  std_logic;
10     I_2     : in  std_logic;
11     I_3     : in  std_logic;
12     I_4     : in  std_logic;
13     I_5     : in  std_logic;
14     O_1     : out std_logic;
15     O_2     : out std_logic;
16     O_3     : out std_logic;
17     O_4     : out std_logic);
18 end entity fsm;
19
20 architecture a_fsm of fsm is
21
22   type T_State is (ST0, ST1, ST2, ST3, ST4, ST5, ST6, ST7);
23   signal SR_present : T_State;
24   signal SC_futur   : T_State;
25
26 begin
27
28   process (I_Clock, I_reset) is
29   begin
30     if I_reset = '1' then
31       SR_present <= ST0;
32     elsif(rising_edge(I_Clock)) then
33       SR_present <= SC_futur;
34     end if;
35   end process;
36
37   process (I_1, I_2, I_3, I_4, I_5, SR_present) is
38   begin
39     O_1 <= '0';                          --default output values
40     O_2 <= '0';
```

```
41      O_3 <= '0';
42      O_4 <= '0';
43      case SR_present is
44        when ST0 =>
45          SC_futur <= ST1;
46
47        when ST1 =>
48          if(I_1 = '1')then               --Cond1
49            SC_futur <= ST2;
50          else
51            SC_futur <= ST1;
52          end if;
53
54        when ST2 =>
55          O_1 <= '1';         --moore output (depending on state only)
56          if(I_2 = '1' and I_4 = '0')then  --Cond2
57            SC_futur <= ST3;
58            O_2      <= '1';  --mealy output (depending on state and input)
59          else
60            SC_futur <= ST2;
61            O_2      <= '0';  --mealy output (depending on state and input)
62          end if;
63
64        when ST3 =>
65          O_1 <= '1';
66          O_3 <= '1';
67          if(I_3 = '1')then               --Cond3
68            SC_futur <= ST4;
69          elsif(I_2 = '1')then            --Cond4
70            SC_futur <= ST6;
71          else
72            SC_futur <= ST3;
73          end if;
74
75        when ST4 =>
76          O_2 <= '1';
77          O_3 <= '1';
78          if(I_1 = '0' and I_2 = '1')then  --Cond5
79            SC_futur <= ST5;
80            O_4      <= '1';
81          elsif(I_5 = '1')then
82            SC_futur <= ST7;
83            O_4      <= '0';
84          else
85            SC_futur <= ST4;
86            O_4      <= '0';
87          end if;
88
89        when ST5 =>
90          O_1 <= '1';
91          if(I_5 = '1')then               --Cond8
92            SC_futur <= ST3;
93          else
94            SC_futur <= ST5;
95          end if;
96
97        when ST6 =>
98          O_1 <= '1';
99          if(I_4 = '1' and not (I_2 = '1' xor I_3 = '0'))then  --Cond7
100           SC_futur <= ST1;
101           O_2      <= '1';
102         else
103           SC_futur <= ST6;
104           O_2      <= '0';
105         end if;
106
107       when ST7 =>
108         SC_futur <= ST7;
109         O_1      <= '1';
110         O_2      <= '1';
111         O_3      <= '1';
112
113       when others => null;
```

```
114     end case;
115   end process;
116
117 end architecture a_fsm;
```

Un process synchrone met l'état à jour. Un process combinatoire determine l'état futur. Un autre process combinatoire determine les sorties. Attention, les deux process combinatoires ne doivent pas créé de latches.

8.13 Instanciation de composants

Pour créer un composant, il est souvent très intéressant de le découper en sous-composant. Une fois les sous-composants créés, il faut les assembler. L'utilisation d'un sous-composant dans le composant courant est appeler une instanciation, on fait appel à une instance du sous-composant. Une fois les sous-composant instanciés, il faut les connecter aux entrées/sorties/signaux du composant courant. On parle dans ce cas de description structurelle.

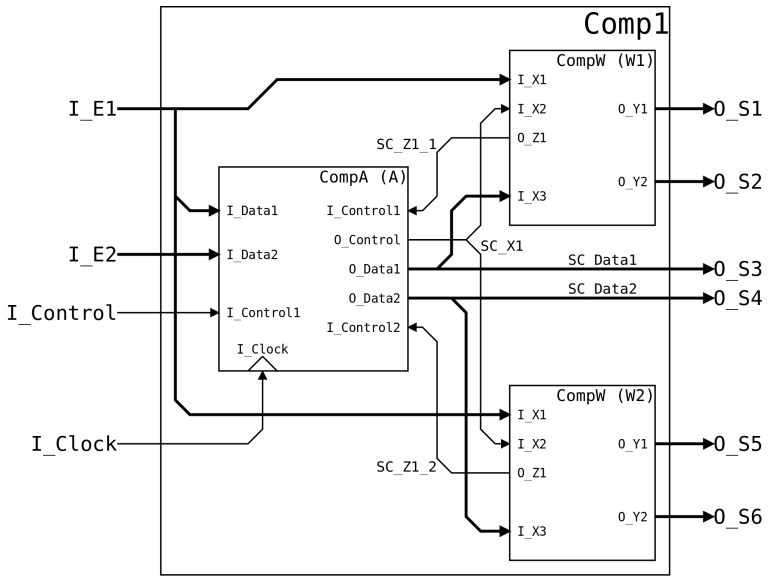

FIGURE 11 – Exemple

Listing 41 – Instanciation de composants

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity Comp1 is
6    generic(
7      G_SizeInput : natural;
8      G_SizeS1    : natural;
9      G_SizeS2    : natural;
10     G_SizeS3    : natural);
11   port(
12     I_Clock   : in  std_logic;
13     I_E1      : in  std_logic_vector(G_SizeInput-1 downto 0);
14     I_E2      : in  std_logic_vector(G_SizeInput-1 downto 0);
15     I_Control : in  std_logic;
16     O_S1      : out std_logic_vector(G_SizeS1-1 downto 0);
17     O_S2      : out std_logic_vector(G_SizeS2-1 downto 0);
18     O_S3      : out std_logic_vector(G_SizeS3-1 downto 0);
19     O_S4      : out std_logic_vector(G_SizeS3-1 downto 0);
20     O_S5      : out std_logic_vector(G_SizeS1-1 downto 0);
21     O_S6      : out std_logic_vector(G_SizeS2-1 downto 0));
22 end entity Comp1;
23
24 architecture archi_Comp1 of Comp1 is
25
```

```
26    component CompA is
27      generic(
28        G_Data : natural;
29        G_Out  : natural);
30      port(
31        I_Clock    : in  std_logic;
32        I_Data1    : in  std_logic_vector(G_Data-1 downto 0);
33        I_Data2    : in  std_logic_vector(G_Data-1 downto 0);
34        I_Control1 : in  std_logic;
35        I_Control2 : in  std_logic;
36        O_Control  : out std_logic;
37        O_Data1    : out std_logic_vector(G_Out-1 downto 0);
38        O_Data2    : out std_logic_vector(G_Out-1 downto 0));
39    end component;
40
41    component CompW is
42      generic(
43        G_X1 : natural;
44        G_X3 : natural;
45        G_Y1 : natural;
46        G_Y2 : natural);
47      port(
48        I_X1 : in  std_logic_vector(G_X1-1 downto 0);
49        I_X2 : in  std_logic;
50        I_X3 : in  std_logic_vector(G_X3-1 downto 0);
51        O_Z1 : out std_logic;
52        O_Y1 : out std_logic_vector(G_Y1-1 downto 0);
53        O_Y2 : out std_logic_vector(G_Y2 downto 0));
54    end component;
55
56    signal SC_Z1_1  : std_logic;
57    signal SC_Z1_2  : std_logic;
58    signal SC_Data1 : std_logic_vector(G_SizeS3-1 downto 0);
59    signal SC_Data2 : std_logic_vector(G_SizeS3-1 downto 0);
60    signal SC_X1    : std_logic;
61
62  begin
63
64    A : CompA
65      generic map (
66        G_Data => G_SizeInput,
67        G_Out  => G_SizeS3)
68      port map (
69        I_Clock    => I_Clock,
70        I_Data1    => I_E1,
71        I_Data2    => I_E2,
72        I_Control1 => SC_Z1_1,
73        I_Control2 => SC_Z1_2,
74        O_Control  => SC_X1,
75        O_Data1    => SC_Data1,
76        O_Data2    => SC_Data2);
77
78
79    W1 : CompW
80      generic map (
81        G_X1 => G_SizeInput,
82        G_X3 => G_SizeS3,
83        G_Y1 => G_SizeS1,
84        G_Y2 => G_SizeS2)
85      port map (
86        I_X1 => I_E1,
87        I_X2 => SC_X1,
88        I_X3 => SC_Data1,
89        O_Z1 => SC_Z1_1,
90        O_Y1 => O_S1,
91        O_Y2 => O_S2);
92
93    W2 : CompW
94      generic map (
95        G_X1 => G_SizeInput,
96        G_X3 => G_SizeS3,
97        G_Y1 => G_SizeS1,
98        G_Y2 => G_SizeS2)
```

```
 99      port map (
100        I_X1 => I_E1,
101        I_X2 => SC_X1,
102        I_X3 => SC_Data2,
103        O_Z1 => SC_Z1_2,
104        O_Y1 => O_S5,
105        O_Y2 => O_S6);
106
107  -- Il est possible d'ajouter des process autour de ces instanciations
108  -- pour manipuler des signaux internes si besoin
109    --process(...)
110    --begin
111    --  .
112    --  .
113    --  .
114    --end process;
115
116  end architecture;
```

9 Exemples VHDL non synthétisable

Il existe en VHDL des instructions qui ne sont pas synthétisable, c'est à dire qu'elles ne sont pas destinées à être utilisées pour décrire un circuit électronique. L'utilité de ces instructions est qu'elles facilitent la simulation et la création rapide de golden modèles (modèle de référence fonctionnel de comparaison par rapport au circuit que l'on souhaite concevoir).

Une simulation se grâce à un test bench et un simulateur (voir section 10).

9.1 Création d'un fichier de test bench

Un test bench est un fichier VHDL qui ne servira que pour tester un ou plusieurs composant. Sa description VHDL n'est pas nécessairement synthétisable, elle l'est même rarement. En effet on utilise souvent des fonctionnalité non synthétisable telle que les instructions *wait* et *after*.

Un test bench est un module qui n'a pas d'entrées/sorties, il ne fait qu'inclure un ou plusieurs composants, piloter leurs entrées et éventuellement surveiller leurs sorties.

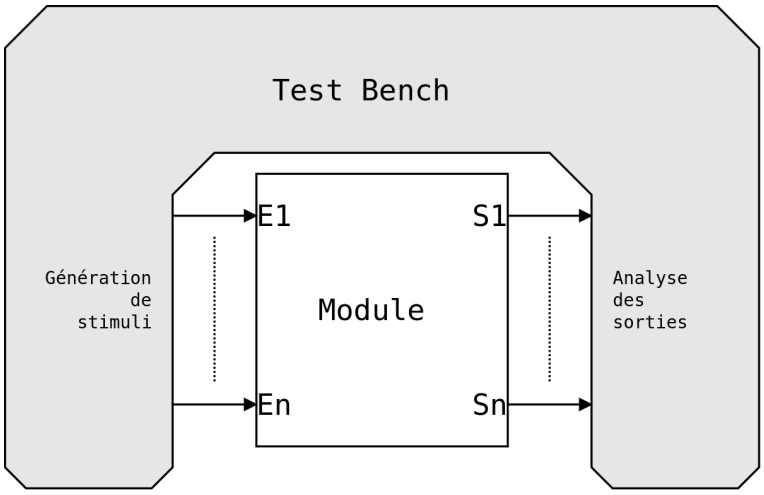

FIGURE 12 – FSM

Listing 42 – Test bench

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

-- exemple de test bench

entity tb_Module is
end tb_Module;

architecture archi_tb_Module of tb_Module is

component Module is     -- module que l'on souhaite tester
    generic(
        G_GenericA  : integer;
        G_GenericB  : integer
    );
    port(
        I_Clock         : in  std_logic;
        I_Reset         : in  std_logic;
        I_Enable        : in  std_logic;
        I_Data          : in  std_logic_vector(G_GenericA-1 downto 0);
        I_Address       : in  std_logic_vector(G_GenericB-1 downto 0);
        O_Ready         : out std_logic;
        O_ResultValid   : out std_logic;
        O_Result        : out std_logic_vector(G_GenericA+G_GenericB-1 downto 0)
    );
end component;
```

```
28
29 constant CST_GENERIC_A  : integer := 12;
30 constant CST_GENERIC_B  : integer := 5;
31
32 signal SR_Clock         : std_logic := '0';
33 signal SR_Reset         : std_logic;
34 signal SR_Enable        : std_logic;
35 signal SR_Data          : std_logic_vector(CST_GENERIC_A-1 downto 0);
36 signal SR_Address       : std_logic_vector(CST_GENERIC_B-1 downto 0);
37 signal SC_Ready         : std_logic;
38 signal SC_ResultValid   : std_logic;
39 signal SC_Result        : std_logic_vector(CST_GENERIC_A+CST_GENERIC_B-1 downto 0);
40
41 begin
42
43     SR_Clock <= not SR_Clock after 7 ns;
44
45     SR_Reset <= '1' , '0' after 59 ns , '1' after 2313 ns , '0' after 2350 ns;
46
47     instance1_Module : Module
48         generic map(
49             G_GenericA  => CST_GENERIC_A    ,
50             G_GenericB  => CST_GENERIC_B
51         )
52         port map(
53             I_Clock         => SR_Clock         ,
54             I_Reset         => SR_Reset         ,
55             I_Enable        => SR_Enable        ,
56             I_Data          => SR_Data          ,
57             I_Address       => SR_Address       ,
58             O_Ready         => SC_Ready         ,
59             O_ResultValid   => SC_ResultValid   ,
60             O_Result        => SC_Result
61         );
62
63     process     -- process de pilotage des entrees du composant a tester
64     begin
65         SR_Enable   <= '0';
66         SR_Data     <= (others => '0');
67         SR_Address  <= (others => '0');
68         wait for 61 ns;                     -- wait : instruction non synthetisable
69         wait until rising_edge(SR_Clock);
70         SR_Enable   <= '1';
71         SR_Data     <= std_logic_vector(to_unsigned (111 , CST_GENERIC_A));
72         SR_Address  <= std_logic_vector(to_unsigned (15 , CST_GENERIC_B));
73         wait until rising_edge(SR_Clock);
74         SR_Enable   <= '1';
75         SR_Data     <= std_logic_vector(to_unsigned (1 , CST_GENERIC_A));
76         SR_Address  <= std_logic_vector(to_unsigned (7 , CST_GENERIC_B));
77
78         -- ... --
79
80     end process;
81
82 end archi_tb_Module;
```

9.2 Manipulation de fichier texte

Pour simuler un circuit il est parfois très utile de pouvoir manipuler des fichiers textes dans un test bench ou même dans un circuit (temporairement avant synthèse) On peut utiliser un fichier texte pour :

- stocker des valeurs à appliquer aux entrées ;
- stocker des valeurs de sorties attendues et les comparer aux valeurs présentes en sortie du circuit ;
- faire la même chose avec des signaux internes ;
- enregistrer des signaux intermédiaires ou des sorties et les utiliser ailleurs.

Fichier lu :

12 13

```
15 14
11 2
1234 678
345 234
12 987
-12   -87
  -3124 9786
```

Listing 43 – Manipulation de fichier texte en simulation

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4  use STD.textio.all;
5
6  entity testStdLogicTextio is
7  end entity testStdLogicTextio;
8
9  architecture arch of testStdLogicTextio is
10   file F_input      : text open read_mode is "../input.txt";
11   file F_output     : text open write_mode is "../output.txt";
12   signal SR_Clock   : std_logic := '0';
13   signal SR_reset   : std_logic;
14   signal SR_Value1  : integer;
15   signal SR_Value2  : integer;
16   signal SR_Value3  : integer;
17   signal SR_Value4  : integer;
18   signal SR_reading : std_logic;
19
20 begin
21
22   SR_reset <= '1', '0'     after 3 ns;
23   SR_Clock <= not SR_Clock after 10 ns;
24
25   process (SR_Clock, SR_reset) is
26     variable theline  : line;
27     variable V_Value1 : integer;
28     variable V_Value2 : integer;
29   begin
30     if(SR_reset = '1')then
31       SR_Value1  <= 0;
32       SR_Value2  <= 0;
33       SR_reading <= '0';
34     elsif rising_edge(SR_Clock) then
35       if(not endfile(F_input))then
36         readline(F_input, theline);
37         read(theline, V_Value1);
38         SR_Value1  <= V_Value1;
39         read(theline, V_Value2);
40         SR_Value2  <= V_Value2;
41         SR_reading <= '1';
42       else
43         SR_reading <= '0';
44       end if;
45     end if;
46   end process;
47
48   SR_Value3 <= SR_Value1*SR_Value2;
49   SR_Value4 <= SR_Value2+SR_Value1;
50
51   process (SR_Clock, SR_reset) is
52     variable theline : line;
53   begin
54     if SR_reset = '1' then
55     elsif rising_edge(SR_Clock) then
56       if(SR_reading = '1')then
57         write(theline, string'("Op1 = "));
58         write(theline, SR_Value1, left, 8); --alignement a gauche, taille minimum de 8 caracteres
59         write(theline, string'("Op2 = "));
60         write(theline, SR_Value2, left, 8); --alignement a gauche, taille minimum de 8 caracteres
61         write(theline, string'("ResMult = "));
62         write(theline, SR_Value3, right, 4); --alignement a droite, taille minimum de 4 caracteres
```

```
63          write(theline, string'("   ResAdd = "));
64          write(theline, SR_Value4, left, 9); --alignement a gauche, taille minimum de 9 caracteres
65          writeline(F_output, theline);
66          -- les deux lignes ci-dessous sont expliquees dans la section suivante
67          assert SR_Value1 > SR_Value2 report "SR_Value1 = " &integer'image(SR_Value1) severity warning;
68          assert SR_Value1 > SR_Value2 report "SR_Value2 = " &integer'image(SR_Value2) severity warning;
69        end if;
70      end if;
71    end process;
72
73  end architecture arch;
```

Fichier de sortie :

```
Op1 = 12      Op2 = 13      ResMult =  156   ResAdd = 25
Op1 = 15      Op2 = 14      ResMult =  210   ResAdd = 29
Op1 = 11      Op2 = 2       ResMult =   22   ResAdd = 13
Op1 = 1234    Op2 = 678     ResMult = 836652   ResAdd = 1912
Op1 = 345     Op2 = 234     ResMult = 80730   ResAdd = 579
Op1 = 12      Op2 = 987     ResMult = 11844   ResAdd = 999
Op1 = -12     Op2 = -87     ResMult = 1044   ResAdd = -99
Op1 = -3124   Op2 = 9786    ResMult = -30571464   ResAdd = 6662
```

9.3 assert

Il existe une instruction qui affichent des messages dans la fenêtre de dialogue du simulateur. Il s'agit de *assert*. Cette instruction permet plusieurs niveaux de criticité :

- note
- warning
- error
- failure (arrêt de la simulation)

Listing 44 – instruction assert

```
1  assert SR_Value1 > SR_Value2 report "SR_Value1 = " &integer'image(SR_Value1) severity warning;
2  assert SR_Value1 > SR_Value2 report "SR_Value2 = " &integer'image(SR_Value2) severity warning;
```

Le message est affiché si la condition booléenne est fausse.

10 Simulation et Script Modelsim

Pour simuler un circuit, il faut un test bench (9.1) et utiliser un simulateur.

Il est souvent plus rapide de lancer ses simulations sous Modelsim avec des scripts. Voici un exemple de script permettant de compiler les fichier VHDL, lancer et parametrer la simulation ainsi que de choisir les signaux que l'on souhaite afficher :

Listing 45 – Script modelsim pour automatiser une simulation

```
1  #
2  # Create work library
3  #
4  vlib work
5  #
6  # Compile sources
7  #
8  vcom -explicit  -93 "Test.vhd"
9  vcom -explicit  -93 "Registre.vhd"
10 vcom -explicit  -93 "RegistreDecalage.vhd"
11 vcom -explicit  -93 "Counter.vhd"
12 vcom -explicit  -93 "FSM.vhd"
13 vcom -explicit  -93 "tb_Test.vhd"
14 #
15 # Call vsim to invoke simulator
16 #
```

```
vsim -novopt -t 1ns -lib work work.tb_Test
#
# Source the wave do file
#
do wave.do # appel du script qui affiche les signaux
#
# Set the window types
#
view wave
view structure
view signals
#
# Run simulation for this time
#
run 1000ns # il est possible de changer la duree
#
# End
#
```

Le script wave.do est de la forme suivante :

```
onerror {resume}
quietly WaveActivateNextPane {} 0
add wave -noupdate /tb_test/I_Clock
add wave -noupdate /tb_test/I_Reset
add wave -noupdate /tb_test/I_Load
add wave -noupdate -radix unsigned /tb_test/I_DLoad
add wave -noupdate /tb_test/I_Enable
add wave -noupdate -radix unsigned /tb_test/O_Count
add wave -noupdate -divider -height 50 <NULL>
TreeUpdate [SetDefaultTree]
WaveRestoreCursors {{Cursor 1} {233 ns} 0}
quietly wave cursor active 1
configure wave -namecolwidth 150
configure wave -valuecolwidth 100
configure wave -justifyvalue left
configure wave -signalnamewidth 1
configure wave -snapdistance 10
configure wave -datasetprefix 0
configure wave -rowmargin 4
configure wave -childrowmargin 2
configure wave -gridoffset 0
configure wave -gridperiod 1
configure wave -griddelta 40
configure wave -timeline 0
configure wave -timelineunits ps
update
WaveRestoreZoom {0 ns} {1050 ns}
```

Cependant, il est beaucoup plus simple de le construire avec l'interface graphique de Modelsim, et de le sauvegarder (En se plaçant dans la fenêtre wave pour la rendre active, *file -> Save Format*.

11 Fichier de contrainte

Le fichier de contrainte contient entre autre la correspondance entre les entrées sorties du circuit que l'on conçoit et les pins du FPGA choisit pour implémenter ce circuit. Il est aussi possible de contraindre la période d'horloge que l'on souhaite atteindre. L'outil de synthèse cherche alors le meilleur compromis timing/surface pour minimiser la surface tout en étant capable de respecter le timing choisit.

11.1 Pour ISE

C'est un fichier portant l'extension .ucf (user constraint file), ce n'est pas du VHDL. Dans sa forme la plus basique, il ressemble à ça :

Listing 46 – User Constraint File

```
NET "I_DataIn[0]" loc = P25;
NET "I_DataIn[1]" loc = H14;
NET "I_DataIn[2]" loc = N25;
```

```
4  NET "I_DataIn[3]" loc = H15;
5  NET "I_DataIn[4]" loc = N24;
6  NET "I_DataIn[5]" loc = J15;
7  NET "I_DataIn[6]" loc = H16;
8  NET "I_DataIn[7]" loc = M24;
9
10 NET "I_DataInWrite" loc = N18;
11
12 NET "O_DataOut[0]" loc = G14;
13 NET "O_DataOut[1]" loc = J28;
14 NET "O_DataOut[2]" loc = G16;
15 NET "O_DataOut[3]" loc = K24;
16 NET "O_DataOut[4]" loc = M18;
17 NET "O_DataOut[5]" loc = H30;
18 NET "O_DataOut[6]" loc = H29;
19 NET "O_DataOut[7]" loc = K19;
20
21 NET "O_DataOutWrite" loc = K25
22
23 NET "I_Clock" LOC = M16
24
25
26 ########## Clock Period Constraints #############
27
28 TIMESPEC "TS_CLK48N" = PERIOD "I_Clock" 20 ns HIGH 50%;
```

Il est possible de l'écrire soit même, ou de passer par une interface graphique, PlanAHead pour Xilinx ISE.

11.2 Pour Vivado et pour Intel Quartus Prime

Coming soon !

12 Nommage et convention de codage

Tous les noms doivent être de préférence en anglais, ou composés de mots en anglais. Utilisation d'un préfixe en majuscule suivit d'un "_" en fonction du genre :

Listing 47 – Convention de codage

```
1     I_NameOfInput               -- pour une entree
2     O_NameOfOutput              -- pour une sortie
3     IO_NameOfInputOutput        -- pour une entree/sortie
4     SC_NameOfCombinatorialSignal -- pour un signal interne combinatoire
5     SR_NameOfRegisteredSignal   -- pour un signal interne de type bascule
6     V_NameOfVariable            -- pour une variable interne a un process, une fonction
7     CST_NAME_OF_CONSTANT        -- pour une constante
8     A_NameOfArray               -- pour un tableau
9     G_NameOfGeneric             -- pour un generic
10    T_NameOfType                -- pour un type
11    R_NameOfRecord              -- pour un record
```

Utilisation d'un suffixe précédé d'un "_" en fonction du signal :

```
1     I_Reset_n   -- pour un signal actif au niveau bas
2     I_Clock_g   -- pour signal global (horloge, reset ...) present dans tout le systeme
```

Entity : Elle doit porter le nom du fichier (sans .vhd évidement), certains synthétiseurs bug sinon

13 Les 10 commandements de la conception de circuits intégrés numériques

(source : ENSSAT)

1. Une seule horloge maîtresse tu auras et tu ne contruiras point de fausses horloges à partir de circuits astables.
2. L'horloge tu ne manipuleras point et tu ne transmettras point à travers une porte logique, car cela causerait des aléas, des fausses transitions et des montées lentes.
3. Tu concevras tous les circuits selon les méthodes de conception synchrones à moins de pouvoir convaincre celui qui paie ton salaire, ou qui attribue ta note que, pour des raisons de rapidité, consommation de puissance ou publication d'articles scientifiques, les circuits synchrones ne peuvent faire l'affaire.
4. Tu ne t'associeras point avec des indésirables tels que les compteurs en cascade ("ripple counter") et les multivibrateurs un-coup ("oneshot" ou multivibrateur monostable), mais tu cultiveras des amitiés avec les compteurs Johnson, les compteurs pseudo-aléatoires et les bascules avec entrée d'activation ("enable").
5. Tous les éléments séquentiels tu raccorderas au RESET global afin que le circuit démarre toujours dans un état connu et défini et que tes simulations ne demeurent point indéfinies pour l'éternité.
6. Tu ne mélangeras point les mises à terre analogique et numérique, car une telle union ne peut mener qu'au désastre.
7. Il ne restera point impuni, celui qui laisse flotter des entrées CMOS.
8. Un reset asynchrone n'est pas prévu pour des tâches telles que remettre un compteur à zéro. Vraiment je te le dis, 6 circuits crées de cette manière se réinitialiseront correctement et sembleront t'apporter gloire et honneur, mais le septième échouera lamentablement et te précipitera dans la honte et la disgrâce.
9. Les entrées asynchrones impropres tu purgeras en les passant dans au moins une bascule D avant de leur donner accès aux pures variables d'états.
10. Quiconque comprend parfaitement les motivations des précédents commandements saura aussi quelle libertés peuvent être tolérées avec eux. Que celui qui les viole dans l'ignorance prenne garde !

14 Édition VHDL

Les fichiers VHDL comme la plupart des codes source ne sont finalement que de simples fichiers textes. N'importe quel éditeur de texte permet de faire du VHDL. Cependant certains sont plus adaptés que d'autres. Emacs (Windows, GNU/Linux,MacOS) fait partie de ces derniers, notamment grâce à son mode majeur VHDL et au mode mineur electric. Emacs se place automatiquement en mode VHDL à l'ouverture ou la création d'un fichier dont l'extension est .vhd. Notepad++ sous windows est aussi un outil intéressant pour éditer en VHDL.

14.1 Emacs

Nous n'allons pas faire ici un manuel d'emacs, d'autres l'on fait sûrement beaucoup mieux que nous le pourrions (mais en 611 pages) : `https://www.gnu.org/software/emacs/manual/pdf/emacs.pdf`

Ce lien pointe vers un document pdf de deux pages donnant les principaux raccourcis clavier d'emacs : `https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf`

L'avantage d'emacs en plus d'être très complet est d'être multi-plates-formes.

Note : la notation des raccourcis ci-dessous est celle adopter dans quasiment toutes les documentation sur Emacs. Par exemple le raccourcis noté C-c C-t si veut dire :

1. Enfoncer la touche *Ctrl*
2. Enfoncer et relâcher la touche *c*
3. Enfoncer et relâcher la touche *t*
4. Relâcher la touche *Ctrl*
5. Enfoncer et relacher la touche *s*
6. Enfoncer et relacher la touche *i*

Dans un raccourci, le M- désigne la touche méta. Il s'agit sur la plupart des clavier de la touche ALT (à gauche de la touche espace).

Les raccourcis généraux du mode VHDL d'emacs

C-c C-b : Beautify, remet au propre un code VHDL, à user sans modération ! ! !

Les raccourcis claviers pour insérer des Template

Insertion de Template		
C-c C-t	C-h	insert header
	C-p (s,n,t)	insert package (1164,numeric_std,textio)
	en	insert entity
	ar	insert architecture
	pc	insert process (comb)
	ps	insert process (seq)
	si	insert signal

Les raccourcis claviers pour manipuler les ports d'entrée/sortie

Manipulation des ports d'entité		
C-c C-p	C-w	copie les ports
	C-e	colle en temps qu'entity
	C-c	colle en temps que component
	C-i	colle en temps qu'instance
	C-s	colle en temps que signal
	M-c	colle en temps que constant
	C-g	colle en temps que generic map
	C-z	colle en temps qu'initialisation
	C-t	colle en temps que testbench

L'édition rectangulaire

Il est possible d'éditer plusieurs lignes en même temps. Pour commencer il faut activer l'option *Use CUA keys (cut/paste with C-x/C-c/C-v)*. Ensuite il suffit de se placer là où l'on souhaite commencer l'édition rectangulaire puis appuyer sur C-RET (RET pour désigne la touche entrée : carriage return) en déplacer le point grâce aux flèche du clavier. Il est possible de faire des copier/coller rectangulaire.

14.2 Notepad++

L'éditeur Notepad++ est libre et gratuit, disponible sous windows seulement. Son principal atout est son édition rectangulaire très efficace. Il suffit de maintenir la touche ALT enfoncer et de sélectionner à la souris. Il existe un plugin VHDL qui propose quelques-unes des fonctions du mode VHDL d'emacs.

15 Sources

- XST User Guide (chapter 7 : HDL coding techniques) : `https://www.xilinx.com/support/documentation/sw_manuals/xilinx14_7/xst_v6s6.pdf`
- ModelSim SE User's Manual Software Version 10.4d : `https://intranet.telecom-bretagne.eu/data/elec/INTRA_ELEC/modelsim_se_user_manual_10_4d.pdf`
- Syntaxe VHDL : `http://amouf.chez.com/syntaxe.htm`
- Doulos : numeric_std : `https://www.doulos.com/knowhow/vhdl_designers_guide/numeric_std/`
- GNU Emacs Reference Card : `https://www.gnu.org/software/emacs/refcards/pdf/refcard.pdf`
- GNU Emacs Manual : `https://www.gnu.org/software/emacs/manual/pdf/emacs.pdf`

IMT Atlantique
Bretagne-Pays de la Loire
École Mines-Télécom

IMT Atlantique Bretagne - Pays de la Loire - www.imt-atlantique.fr

Campus de Brest
Technopôle Brest-Iroise
CS 83818
29238 Brest Cedex 03
T +33 (0)2 29 00 11 11
F +33 (0)2 29 00 10 00

Campus de Nantes
4, rue Alfred Kastler - La Chantrerie
CS 20722
44307 Nantes Cedex 03
T +33 (0)2 51 85 81 00
F +33 (0)2 51 85 81 99

Campus de Rennes
2, rue de la Châtaigneraie
CS 17607
35576 Cesson Sévigné Cedex
T +33 (0)2 99 12 70 00
F +33 (0)2 99 12 70 08