Cours VHDL - IV

L3-S6 - Université de Cergy-Pontoise

Laurent Rodriguez – Benoît Miramond

Plan du cours

I – Historique de conception des circuits intégrés
- **HDL**
  - Modèles de conceptions
- **VHDL**
  - Les modèles de conceptions en VHDL
  - Les 5 briques de base

II – VHDL et FPGA
- **VHDL**
  - **Syntaxe et typage**
  - **Retour sur les briques de bases**
  - **Retour sur la conception structurelle et comportementale en VHDL**
    Port map, Equations logiques, Tables de vérités (With ... Select)
- **FPGA**
  - **Qu'est ce qu'un FPGA**
  - **Flot de conception**
  - **Carte de développement et environnement de TP**

Plan du cours

III – Modélisation

- Compléments sur la description Structurelle

- Description comportementale approfondie

Circuits combinatoires

- Styles de description

Flots de données

Instructions concurrentes

Table de vérité

Fonctions

- Circuits standards

- Comparateur
- Multiplexeurs
- Encodeurs

Plan du cours

IV – Processus et gestion du temps
- **Syntaxe et sémantique**
- **Portée - signaux & variables**
- **Annotations temporelles et délais**

V – Simulation
- **TestBench**
- **GHDL & GTK-Waves**
- **Quartus & ModelSIM**

VI – Modèles génériques
- **Paramètres génériques**
- **« Generate »**
- **Boucles**
- **Exemples**

Plan du cours

VII – VHDL pour la synthèse

- **Les types**
- **Processus**
  - **Initialisation et RESET**
  - **Variables et signaux**
  - **Synchronisation**
  - **Instructions conditionnelles**
  - **Boucles**
    - **Synthèse série vs parallèle**
- **Instructions concurrentes**
- **Sous-programmes**
- **Opérateurs : l'inférence des opérateurs**
  - **Groupement**
  - **Partage**
  - **Parallèlisme**

VII - VHDL POUR LA SYNTHESE

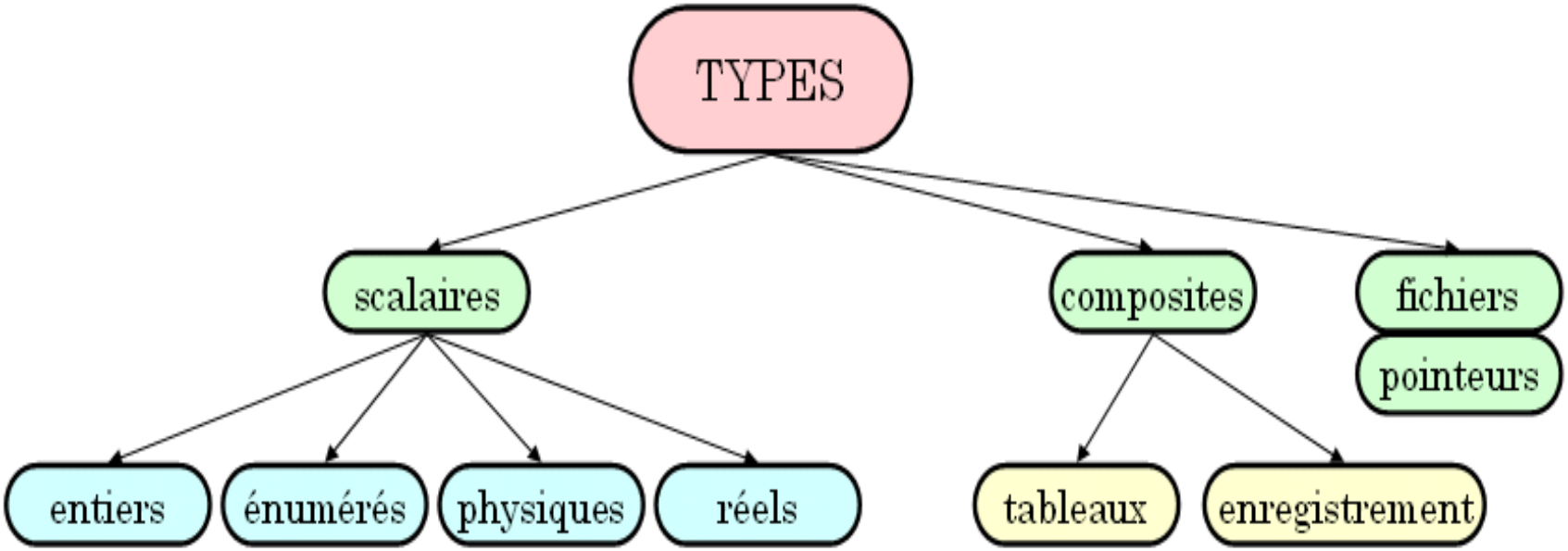
Les types

IV - Sous-ensemble VHDL synthétisable : restrictions

Les restrictions imposées par la synthèse sur le langage VHDL sont principalement les suivantes:

- Les informations de délai (clause « **after** », « **wait for** ») **sont ignorées.**
- Un **processus** doit être écrit selon certaines règles pour être correctement traduit en éléments matériels (**circuits combinatoires VS circuits séquentiels**).
- **Seuls certains types sont supportés.**

IV - Sous-ensemble VHDL synthétisable : **Types**



IV - Sous-ensemble VHDL synthétisable : **Types : Entiers**

Le type entier prédéfini dans le package STD est le type **INTEGER** définit sur **32 bits** $[-2^{31}, 2^{31}-1]$

2 sous-types sont définis :

```
SUBTYPE natural IS integer RANGE 0 TO integer'high;
SUBTYPE positive IS integer RANGE 1 TO integer'high;
```

Ils sont réalisés en synthèse comme des **vecteurs de 32 bits** codés en **non signés ou en complément à 2**

Par contre :

```
SUBTYPE my_byte IS INTEGER RANGE -128 TO 127;
-- codage sur 8 bits
```

IV - Sous-ensemble VHDL synthétisable : **Types : Entiers**

A noter pour l'utilisation d'opérateurs :

```
SUBTYPE UN_A_DIX IS natural RANGE (1 to 10);
SUBTYPE DIX_A_UN IS natural RANGE (10 downto 1);
```

Sont compatibles étant tout deux un sous-type du même type ***natural*** alors que :

```
TYPE UN_A_DIX IS natural RANGE (1 TO 10);
TYPE DIX_A_UN IS natural RANGE (10 DOWNTO 1);
```

Sont considérés comme deux types indépendants et incompatibles.

IV - Sous-ensemble VHDL synthétisable : **Types : Enumérés**

C'est un type défini par une **énumération exaustive de ses valeures** :

```
TYPE NomDuType IS (V0, V1, ... Vn) ;
```

L'ordre d'initialisation **est important** ! **A l'initialisation** d'un signal d'un type énuméré **T**, le signal prend la valeur **T'LEFT**.

Ex :

```
TYPE state IS ( idle, init, shift, add, check );
SIGNAL cs : state ; -- CS = idle par défaut !
```

IV - Sous-ensemble VHDL synthétisable :
Types : Enumérés

Les valeurs sont encodées automatiquement suivant l'ordre :

idle, init, shift, add, check => "000" "001" "010'' ...

Pour les surcharger *(notamment pour les FSM...)* :

```
TYPE state IS ( idle, init, shift, add, check );
ATTRIBUTE enum_encoding: STRING;
attribute enum_encoding of state: type is
                "00001 00010 00100 01000 1000000 11100"
```

IV - Sous-ensemble VHDL synthétisable : **Types : Enumérés**

Dans le paquetage STANDARD de la bibliothèque STD, plusieurs types énumérés sont définis :

```
TYPE boolean IS (FALSE, TRUE);
TYPE bit IS ('0', '1');
TYPE severity_level IS (NOTE, WARNING, ERROR, FAILURE);
TYPE character IS ( NUL, SOH, STX, ETX,..., '0','1', ...);
...
```

A noter que la valeur d'un bit est équivalente à celle d'un caractère et toujours entre quote : '0' et '1' contrairement aux entiers 0 et 1

IV - Sous-ensemble VHDL synthétisable : **Types : Enumérés**

Dans le paquetage IEEE.STD_LOGIC_1164 le type STD_LOGIC étend le type STD_ULOGIC qui est est défini par :

```
TYPE std_ulogic IS ('U', 'X', 'O', '1', 'Z', 'W', 'L', 'H', '-');
```

- **'U'** : Etat Undefined pour un signal non initialisé au démarage.
- **'X'** : Conflit, le signal est affecté d'un côté à '1' et d'un autre à '0'
- **'0'** et **'1'** : Valeurs booléennes du signal.
- **'Z'** : Etat haute 'impédance".
- **'W'** : Signal relié à 2 résistances de tirage, une tirant à 0 et l'autre à 1.
- **'H'** et **'L'** : sont des valeurs d'un signal relié respectivement à une résistance de tirage à 1 et à 0.
- **'-'** : est un état indifférent. Utile pour décrire les tables de vérité.

IV - Sous-ensemble VHDL synthétisable : **Types : Tableaux**

Les types **TABLEAU** ou **ARRAY** sont des collections d'objets de même type, indexés par des entiers ou des énumérés.

```
TYPE word IS array (31 downto 0) OF bit; -- intervalle descendant
TYPE memory IS array (0 to 255) OF word; -- intervalle montant
TYPE truth_table IS array (bit, bit) OF bit;
```

Un tableau peut avoir une taille inconnue donc non contraint, par exemple le type BIT_VECTOR de la bibliothèque STD est un tableau de dimension 1 (vecteur) de taille quelconque :

```
TYPE bit_vector IS array (natural RANGE <>) OF bit;
```

La taille est précisée à l'utilisation :

```
SIGNAL toto : bit_vector(31 DOWNTO 0);
```

IV - Sous-ensemble VHDL synthétisable : **Types : Tableaux**

Il faut noter que l'indexation peut être **MSB en tête (31 downto 0)** , la plus courante, ou **LSB en tête (0 to 31).**

! une affectation de signaux vectoriels ayant des indexations différente provoque une **inversion des composantes du signal.**

```
SIGNAL a : STD_LOGIC_VECTOR(3 DOWNTO 0) ;
SIGNAL b : STD_LOGIC_VECTOR(0 to 3) ;
...
b <= a ; -- b(0) = a(3) ;
```

IV - Sous-ensemble VHDL synthétisable : **Types : Tableaux**

Il peut y avoir des tableaux de tableaux ou des tableaux à plusieurs dimensions; Les affectations diffèrent quelque peu comme illustré dans l'exemple suivant :

```
TYPE TAB1 is array(0 to 2) of bit_vector(7 downto 0);
TYPE TAB2 is array(0 to 3, 1 to 8) of bit;
SIGNAL A : TAB1;
SIGNAL B : TAB2;

BEGIN
 --tableau de tableau
 A(0) <="01001111";
 A(2)(5) <= '1';

 -- tableau à 2 dimensions ( ! pas forcément synthétisables)
 B(3,5) <= '0';

END exemple;
```

IV - Sous-ensemble VHDL synthétisable : **Types : Attributs**

Il s'agit de caractéristiques de types ou d'objet. Syntaxe :

<OBJET>'<ATTRIBUT>

Il existe des attributs sur les types, sur les objets de type tableau et sur les signaux. On peut définir nos propres attributs.

Il est possible de créer ces propres attributs. Certains outils de synthèse en tirent profit pour passer des arguments de synthèse.

Les principaux attributs de type :

```
TYPE COULEUR IS (BLEU, ROUGE, VERT);
```

- COULEUR'left : renvoie BLEU
- COULEUR'right : renvoie VERT
- COULEUR'pos(BLEU) : renvoie 0
- COULEUR'val(0) : renvoie BLEU
- COULEUR'succ(BLEU) : renvoie ROUGE
- COULEUR'pred(ROUGE) : renvoie BLEU

IV - Sous-ensemble VHDL synthétisable : **Types : Attributs**

Les principaux attributs pour les tableaux :

```
TYPE MOT IS bit_vector(7 DOWNTO 0);
TYPE TAB IS array (4 DOWNTO 0) OF MOT;
SIGNAL NOM : MOT;
SIGNAL TABLEAU : TAB;
```

- **NOM'LEFT** : renvoie 7;
- **NOM'LENGTH** : renvoie 8;
- **TABLEAU'RIGHT** : renvoie 0;
- **TABLEAU'RANGE** : renvoie 4 downto 0;

Sur les signaux :

- **CLK'EVENT :** renvoie un **BOOLEAN** indiquant **si le signal CLK a changé**.

IV - Sous-ensemble VHDL synthétisable :
Types : Constantes

Une constante possède une valeur fixe durant la simulation.
Syntaxe : CONSTANT nom-const {, ... } : soustype [:= expression] ;

Exemples :

```
CONSTANT MAX_COUNT: positive := 255;
```

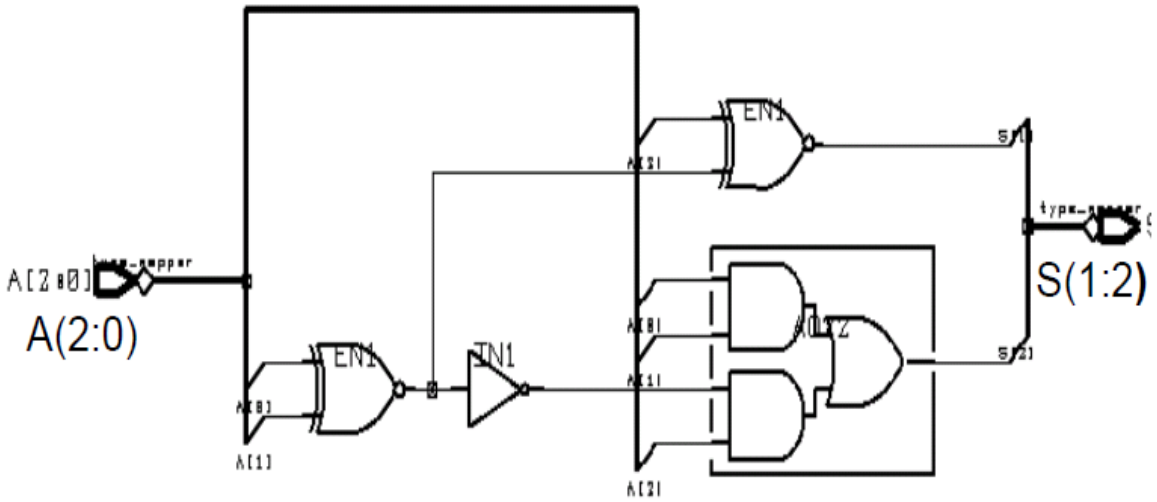
Un autre exemple typique de l'utilisation de constante en synthèse est la déclaration d'une table de vérité ou de contenu d'une mémoire ROM.

Le Code suivant est un exemple d'utilisation de la table de vérité du modèle comportemental d'un additionneur 1 bit complet.

On tire avantage de l'utilisation du type UNSIGNED ou SIGNED pour disposer d'une représentation d'une valeur entière au niveau du bit et manipuler les objets de ce type comme des valeurs entières.

Code d'un additionneur

```
package rom_pkg is
    subtype t_word is BIT_VECTOR(1 to 2);
    subtype t_address is NATURAL range 0 to 7;
    type t_rom is array (t_address) of t_word;
end package rom_pkg;

use WORK.rom_pkg.all;
entity add1b is
    port (A: in t_address; S: out t_word);
end;

architecture tt of add1b is
    constant add1b_tt: t_rom := (
        0          => "00",
        1 | 2      => "10",
        3 | 5 | 6  => "01",
        4          => "10",
        7          => "11");
begin
    S <= add1b_tt(A);
end tt;
```



VII - VHDL POUR LA SYNTHESE

Processus et synchronisation

IV - Sous-ensemble VHDL synthétisable : **Processus combinatoires ou séquentiels**

L'usage de processus permet de contrôler le type de circuit inféré par la synthèse, à savoir un circuit combinatoire ou séquentiel. Le premier type de circuit est asynchrone et son comportement est uniquement dirigé par des événements sur des signaux. Le second type de circuit est synchronisé sur un signal d'horloge et implique des registres (latches ou flip-flops).

Il existe quatre conditions à remplir pour assurer qu'un processus va inférer un circuit purement combinatoire:

1) Le processus possède une liste de sensibilité ou une seule instruction **wait** équivalente.
2) Le processus ne déclare pas de variable locale ou ses variables locales sont toutes affectées avant d'être lues.
3) Tous les signaux lus dans le processus font partie de la liste de sensibilité ou de l'instruction **wait** équivalente.
4) Tous les signaux affectés dans le processus le sont dans chaque branche impliquée par une instruction conditionnelle (instruction **if**) ou de sélection (instruction **case**).

Si l'une quelconque de ces règles n'est pas satisfaite, un ou plusieurs éléments de mémoire sont inférés. Le circuit inféré est dans ce cas séquentiel synchrone. Le signal d'horloge est identifié par des instructions de formes particulières:

```
wait until clk = '1';
wait until clk ='1' and clk'EVENT;
wait until clk = '1' and not clk'STABLE;
```

IV - Sous-ensemble VHDL synthétisable :
Instruction séquentielles

Affectation de signal

Une clause de délai dans une affectation de signal est ignorée pour la synthèse, mais pas considérée comme un erreur:

```
S <= '0' after 10 ns;  -- délai ignoré
```

L'affectation d'une forme d'onde à éléments multiples est par contre considérée comme une erreur:

```
S <= '1', '0' after 20 ns, '1' after 30 ns;  -- erreur
```

IV - Sous-ensemble VHDL synthétisable :
Initialisation : SET/RESET Synchrone/Asynchrone

Tout objet VHDL a normalement une valeur initiale soit par son type, soit lors de la déclaration.
Ces hypothèses ne sont pas supportées lors de la synthèse
Il faut systématiquement prévoir un **set/reset**

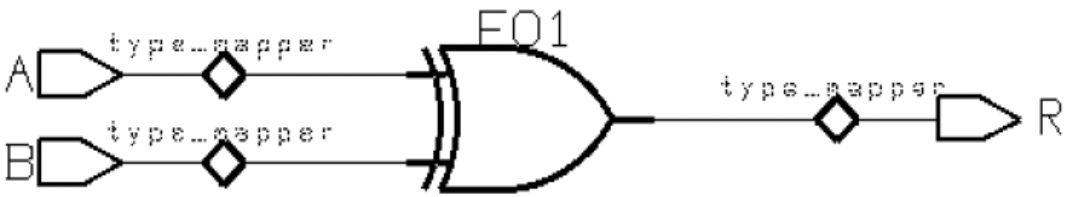
```
entity E is
    port ( ...; rst: in BIT; ... );
end E;

architecture sync of E is
    signal S: BIT_VECTOR(15 downto 0);
begin
    process
    begin
        wait until clk = '1';
        if rst = '1' then -- reset synchrone
            S <= (others => '0');
            -- + autres initialisations
        else
            -- comportement normal...
        end if;
    end process;
end sync;
```

```
architecture async of E is
    signal S: BIT_VECTOR(15 downto 0);
begin
    process (clk, rst)
    begin
        if rst = '1' then -- reset asynchrone
            S <= (others => '0');
            -- + autres initialisations
        elsif clk = '1' and clk'EVENT then
            -- comportement normal synchrone...
        end if;
    end process;
end async;
```

IV - Sous-ensemble VHDL synthétisable : **Variables et signaux**

Exemple de résultat de l'optimisation du synthétiseur :

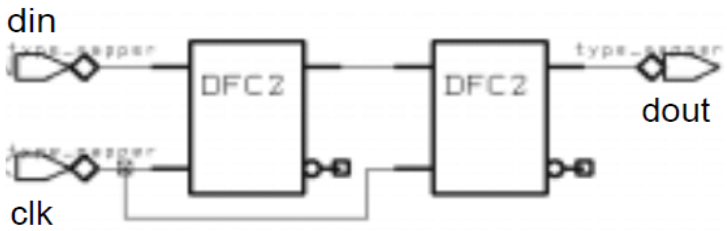
```
entity var is
    port (A, B: in BIT; R: out BIT);
end entity var;
;
architecture a of var is
begin
    process
    begin
        variable V: BIT;
    begin
        V := A xor B;
        R <= V;
    end process;
end architecture a;
```



```
entity shiftreg is
    port (clk, din: in BIT; dout: out BIT);
end shiftreg;
```

-- modèles corrects

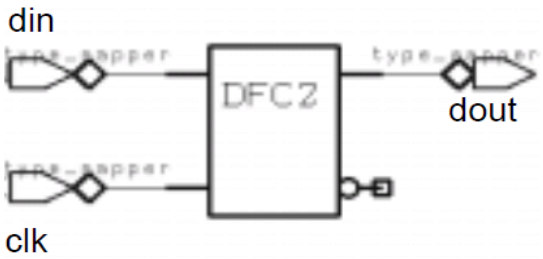
```
architecture good of shiftreg is
    signal sint: BIT;
begin
    process
    begin
        wait until clk = '1';
        sint <= din;
        dout <= sint;
    end process;
end good;

architecture good2 of shiftreg is
begin
    process
        variable vint: BIT;
    begin
        wait until clk = '1';
        dout <= vint;
        vint := din;
    end process;
end good2;
```



-- modèle incorrect

```
architecture bad of shiftreg is
begin
    process
        variable vint: BIT;
    begin
        wait until clk = '1';
        vint := din;
        dout <= vint;
    end process;
end bad;
```

L'affectation des variables est immédiate !



IV - Sous-ensemble VHDL synthétisable : **Variables et signaux**

Affectation de signal

Une clause de délai dans une affectation de signal est ignorée pour la synthèse, mais pas considérée comme un erreur:

```
S <= '0' after 10 ns;  -- délai ignoré
```

L'affectation d'une forme d'onde à éléments multiples est par contre considérée comme une erreur:

```
S <= '1', '0' after 20 ns, '1' after 30 ns;  -- erreur
```

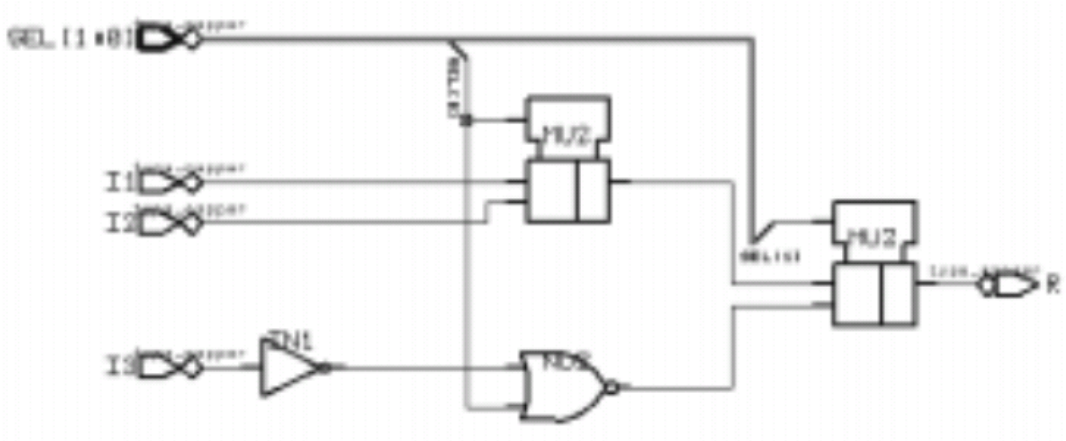
IV - Sous-ensemble VHDL synthétisable :
Structures conditionnelles : IF

Instruction conditionnelle :

L’instruction **IF** suppose une priorité dans les traitements.
Cette priorité est implémentée par des multiplexeurs en série.
Dans le premier cas, le **else est précisé**, le circuit est **combinatoire**
Dans le second cas, **il est absent** le circuit est **séquentiel** pour mémoriser le signal R

IV - Sous-ensemble VHDL synthétisable : **Structures conditionnelles : CASE**

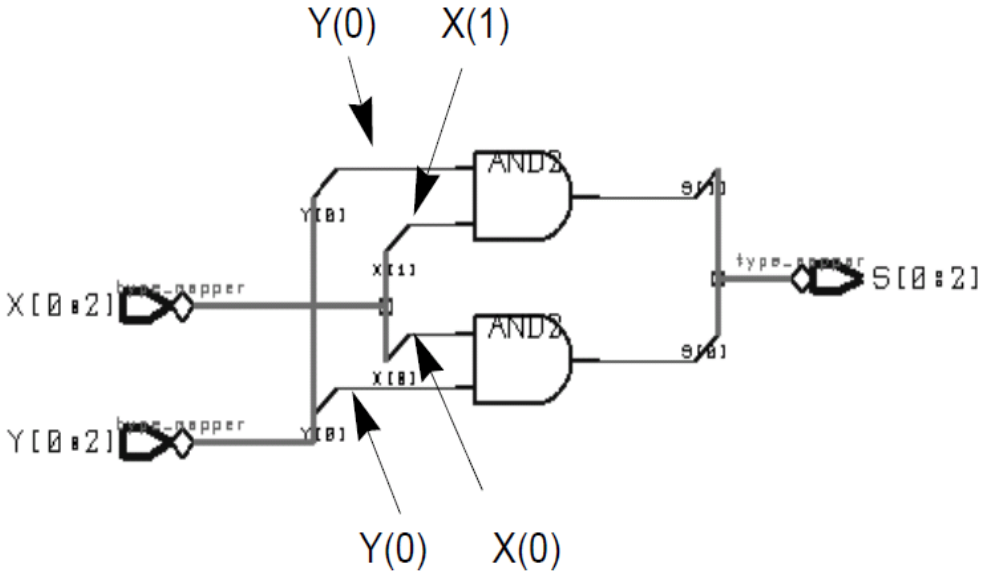
```
package casestmt_pkg is
    type sel_cmd is (S1, S2, S3, S4);
end casestmt_pkg;

use WORK.casestmt_pkg.all;
entity casestmt is
    port (sel: in sel_cmd;
          I1, I2, I3: in BIT;
          R: out BIT);
end;

architecture comb of casestmt is
begin
    process (sel, I1, I2, I3)
    begin
        case sel is
            when S1     => R <= I1;
            when S2     => R <= I2;
            when S3     => R <= I3;
            when others => R <= '0';
        end case;
    end process;
end comb;
```



IV - Sous-ensemble VHDL synthétisable : **Les boucles**

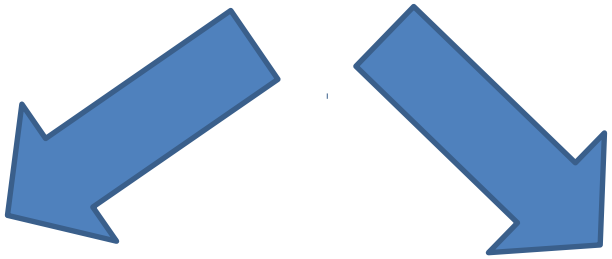
Les boucles For, loop et while sont très pratique pour simplifier le code.
Le nombre d'itération doit être statique !

```
-- X, Y: in BIT_VECTOR(0 to 2);
-- S: out BIT_VECTOR(0 to 2);
architecture a of e is
    subtype nat is NATURAL range 0 to 2;
    constant N: nat := 1;
begin
    process (X, Y)
    begin
        for I in X'RANGE loop
            S(I) <= X(I) and Y((I + 1) mod N);
            exit when I = N;
        end loop;
    end process;
end a;
```



IV - Sous-ensemble VHDL synthétisable : **Les boucles**

```
signal S1, S2: BIT_VECTOR(1 to 10);
```



```
for i in S1'RANGE loop
    S2(i) <= S1(i);
end loop;
```

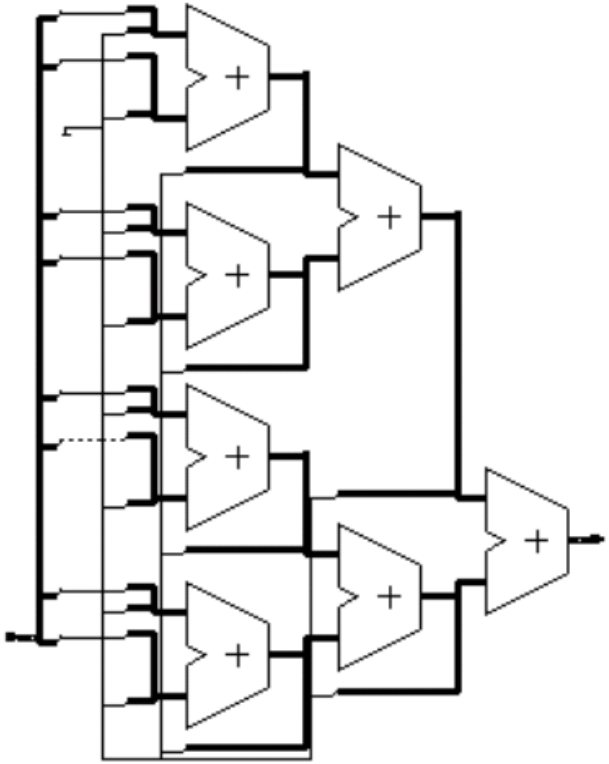
```
S2 <= S1;
```

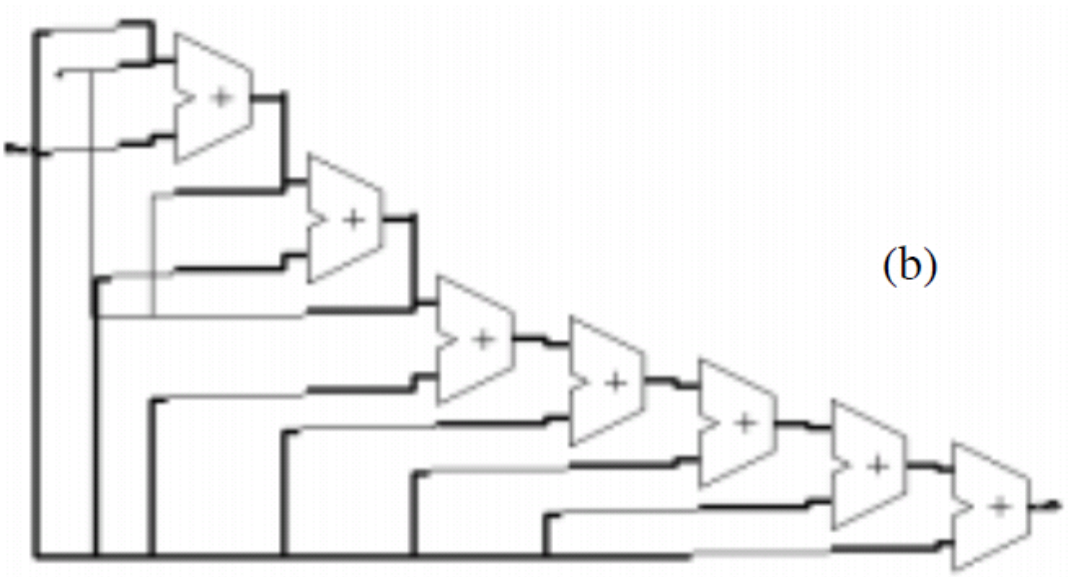
Equivalents !

IV - Sous-ensemble VHDL synthétisable :
Les boucles : Synthèse série vs parallèle

```
Count := 8;
for Level in 0 to 2 loop
    Count := Count/2;
    for K := 0 to Count-1 loop
        Acc(K) := Acc(K*2) + Acc((K*2)+1);
    end loop;
end loop;
Sum <= Acc(0);
```

(a)





(b)

```
Acc := 0;
for K in 0 to 7 loop
    Acc := Acc + A(K);
end loop;
Sum <= Acc;
```

VII - VHDL POUR LA SYNTHESE

Instructions concurrentes

IV - Sous-ensemble VHDL synthétisable :
Les Instructions concurrentes

Sont supportées par la synthèse :

- Les affectations de signaux conditionnelles
- La hiérarchie des composants
- Seulement la configuration des entités par défaut
- L’instruction generate
- Et les paramètres génériques de type entier

VII - VHDL POUR LA SYNTHESE

Sous-programmes

IV - Sous-ensemble VHDL synthétisable : **Les sous-programmes**

Un appel de sous-fonction n'implique pas un nouveau niveau hiérarchique.
Il est remplacé par un circuit combinatoire si :

- ses arguments sont de mode **in** ou **out**
- il n'y a pas d'instruction **wait** dans le corps de procédure
- la procédure ne manipule que des objets locaux et ses arguments (pas d'effet de bord).

Autrement, l'appel infère un circuit séquentiel

IV - Sous-ensemble VHDL synthétisable :
Exemple de sous-programme

```
-- type data is INTEGER range 0 to 3;
-- type darray is array (1 to 3) of data;
-- ports: inar: in darray; outar: out darray
architecture a of sort is
begin
    process (inar)
        procedure swap (d: inout darray; l, h: in INTEGER) is
            variable tmp: data;
        begin
            if d(l) > d(h) then
                tmp := d(l);
                d(l) := d(h);
                d(h) := tmp;
            end if;
        end swap;
        variable tmpar: darray;
    begin
        tmpar := inar;
        swap(tmpar,1,2);
        swap(tmpar,2,3);
        swap(tmpar,1,2);
        outar <= tmpar;
    end process;
end a;
```

VII - VHDL POUR LA SYNTHESE

Opérateurs : l'inférence des opérateurs

IV - Sous-ensemble VHDL synthétisable : **Les opérateurs**

Classe	Symbole	Fonction	Défini pour
Opérateurs divers	Not ** abs	Complément Exponentiel Valeur absolue	Bit, booléen, entier, réel, numérique
Opérateurs multiplicatifs	* / Mod rem	Multiplication Division Modulo Reste	Numérique Entier
Signe (unaire)	+ -	Positif Négatif	Numérique
Opérateurs additifs (binaire)	+ - &	Addition Soustraction Concaténation	Numérique 1 dimension
Opérateurs relationnels	= /= < > <= >=	Égal Différent Inférieur Supérieur Inférieur ou égal Supérieur ou égal	Tous les types Retourne un booléen
Opérateurs logiques (binaire)	And Or Nand Nor Xor	ET OU NON ET NON OU OU exclusif	Bit Booléen vecteur

IV - Sous-ensemble VHDL synthétisable : **Les opérateurs**

Numérique : « Type identifier IS RANGE implementation_defined ; »

Ex : TYPE byte IS RANGE 0 to 255; -- byte varie de 0 à 255
TYPE index IS RANGE 7 DOWNTO 0; -- index varie de 7 à 0
SUBTYPE byte IS INTEGER RANGE 0 TO 255; -- byte est un entier de 0 à 255

Classe d'opérateurs **SIGNE (Unaire):**

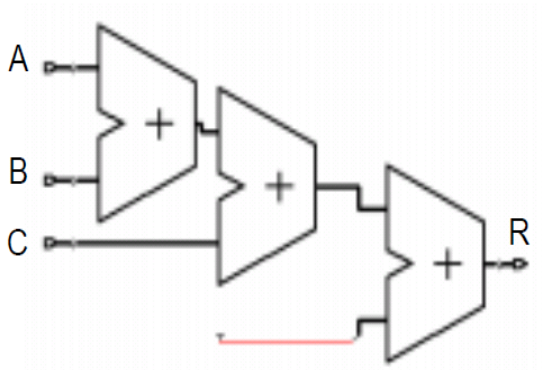
A / -B ; -- illégal
A / (-B) ; -- OK

Classe d'opérateurs **MULTIPLICATIFS :** opérandes de types différents possible !

9 ns * 8; -- = 72ns : le résultat est du type physique
9 ns / 3; -- = 3 : le résultat est du type numérique
9 ns * 8 ns -> l'opération provoque une erreur de compilation

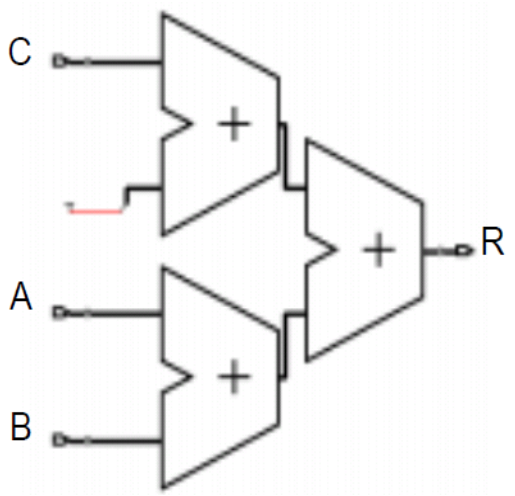
IV - Sous-ensemble VHDL synthétisable :
Les opérateurs : Importance du groupement

R <= A + B + C + 1



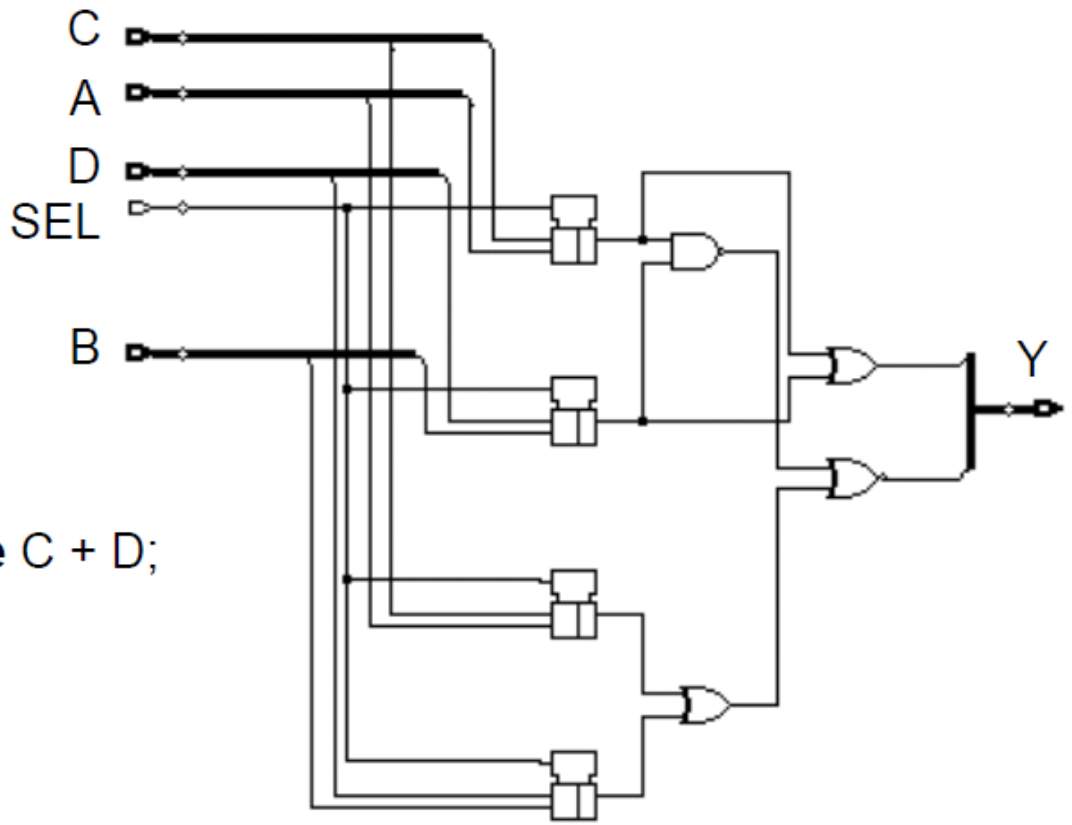
R <= (A + B) + (C + 1)

R <= ((A + B) + C) + 1)



IV - Sous-ensemble VHDL synthétisable :
Les opérateurs : partage des opérateurs

Ici, un seul additionneur avec entrées multiplexées (A, B, C et D sont sur 1 bit) :

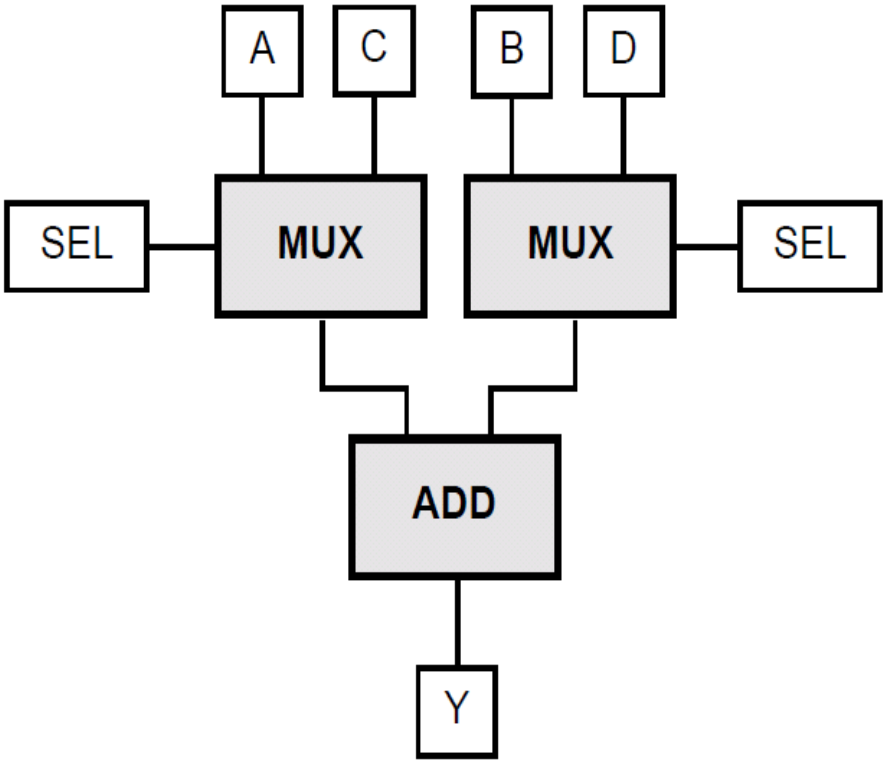



```
Y <= A + B when SEL = '1' else C + D;
```

IV - Sous-ensemble VHDL synthétisable : **Les opérateurs : partage des opérateurs**

Exemples équivalents, soit en séquentiel, soit en concurrent :

```
-- instructions concurrentes
MUX1 <= A when SEL = '1' else C;
MUX2 <= B when SEL = '1' else D;
Y <= MUX1 + MUX2;

-- instructions séquentielles
if SEL = '1' then
    MUX1 := A;
    MUX2 := B;
else
    MUX1 := C;
    MUX2 := D;
end if;
Y <= MUX1 + MUX2;
```



Biblio

Documents de cours

- Ce cours en ligne à :

Http://perso-etis.ensea.fr/rodriguez/

- Un très bon support de cours

http://hdl.telecom-paristech.fr/index.html

- Le cours de Licence 2 (en particulier pour ceux qui ne l'ont pas suivi):

http://perso-etis.ensea.fr/miramond/Enseignement/L2/circuits_numeriques.html

- Le cours de Licence 3 de 2013 :

http://perso-etis.ensea.fr/miramond/Enseignement/L3/Cours_VHDL_2011.html

Documents de développement

- Quartus

http://quartushelp.altera.com/current/

- Documentation Altera sur les Cyclones II et IV (entre autre...)

http://www.altera.com/literature/lit-index.html