Le langage VHDL

VHDL:

VHSIC (Very High Speed Integrated Circuit)

Hardware

Description

Language

HISTORIQUE:

1980 => Projet du DoD

1987 => VHDL adopté par IEEE (norme IEEE-l076)

1993 => enrichissement de la norme (IEEE-1164)

Le langage VHDL

VHDL:

=> Logiciel de programmation permettant la description d'ensembles et/ou sous-ensembles matériels.

=> permet des descriptions comportementales, structurelles ou mixtes.

Comparaison avec autres standards:

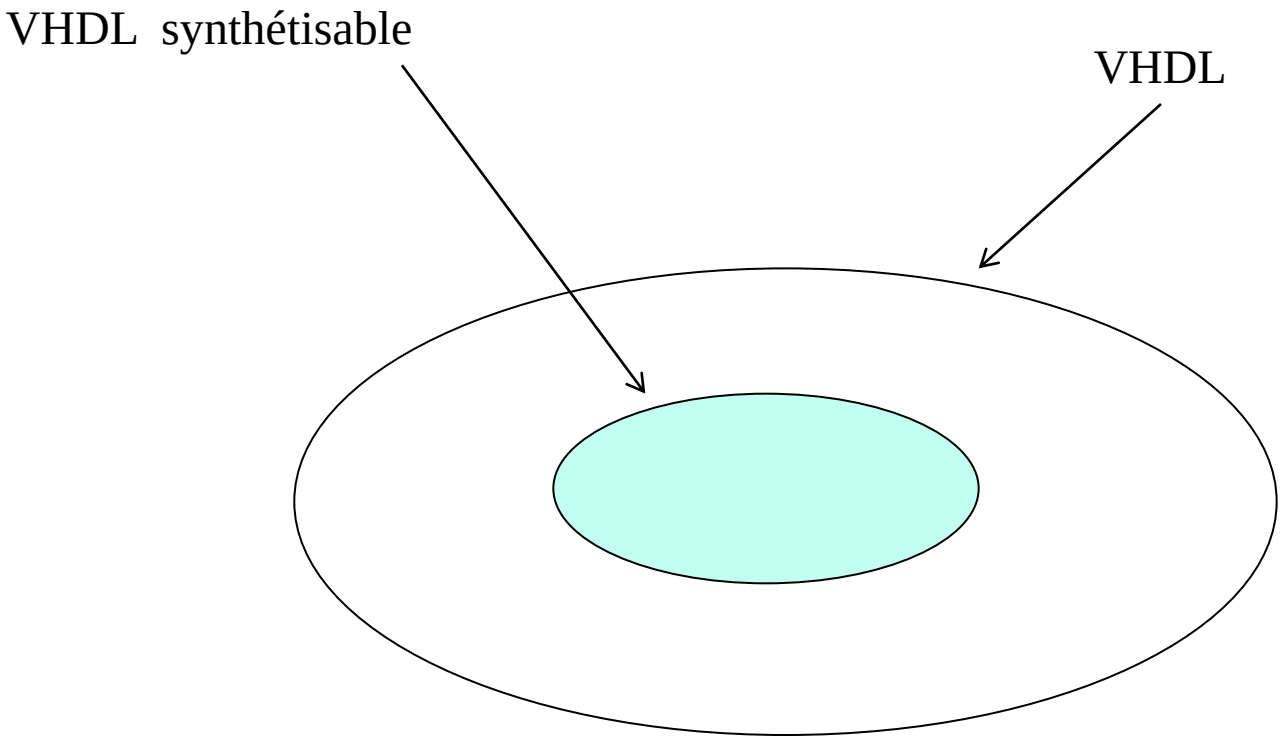
VHDL:

" Dites moi comment doit se comporter votre application et je vous fournirai les circuits correspondants".

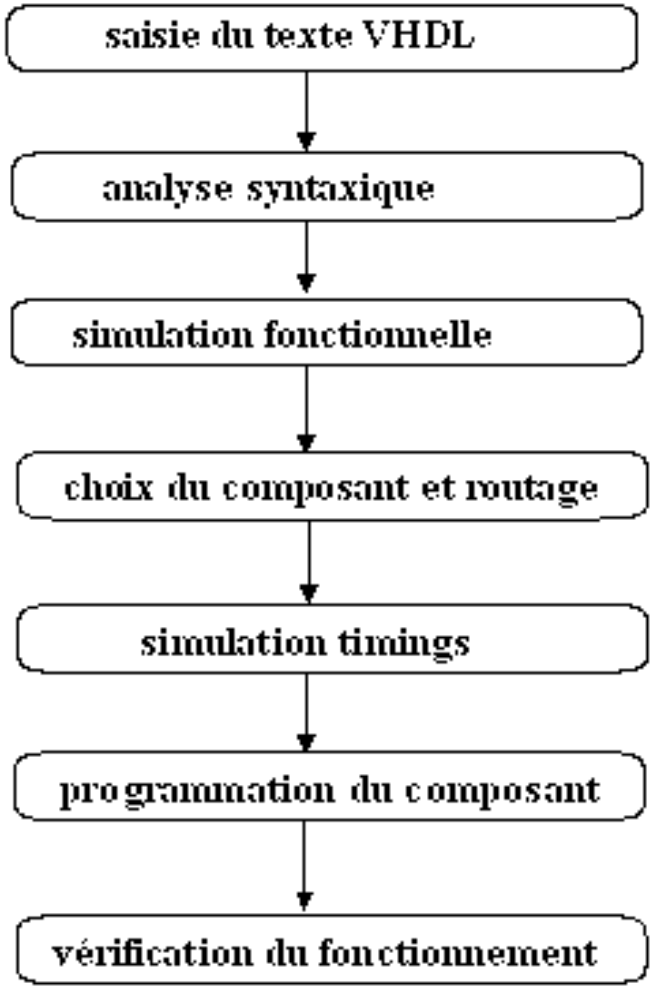
ABEL, PALASM, AHDL:

" Dites moi quels circuits vous voulez et je vous les fournirai".

Le langage VHDL



ETAPES DE DEVELOPPEMENT



CONCEPTS DE BASE:

- **ENTITY**
- **ARCHITECTURE**
- **CONFIGURATION**
- **PACKAGE**

⇒ les instructions sont terminées par;
⇒ les "blancs" n'ont pas d'effets sur le programme
⇒ les commentaires sont précédés de --
⇒ on peut utiliser indifféremment majuscules ou minuscules

LES LIBRAIRIES:

Quelques librairies utilisées:

- **IEEE.std_logic_1164.all**
- **IEEE.std_logic_arith.all**
- **IEEE.std_logic_signed.all**
- **IEEE.std_logic_unsigned.all**

La librairie IEEE.std_logic_1164.all:

⇒ issue de la révision de 1993.
⇒ permet de travailler sur:
- **signaux**
- **variables**
- **entrées-sorties**

⇒ utilise les types: BIT, BOOLEEN, ENTIER, REEL
⇒ opérateurs associés:
pour les entiers: +, -, <, >, >=, >=, *, /
pour les bits: AND, OR, NOT, XOR, &

Librairie lpm.lpm_components.all:

⇒ librairie de macro-fonctions
Les librairies utilisées sont à mettre en début du fichier VHDL:
⇒ Library IEEE;

USE IEEE.std_Iogic_1164.all;

l'entité: ENTITY

⇒ **s'apparente à un symbole de composant**
⇒ **vue extérieure du modèle VHDL réalisé**

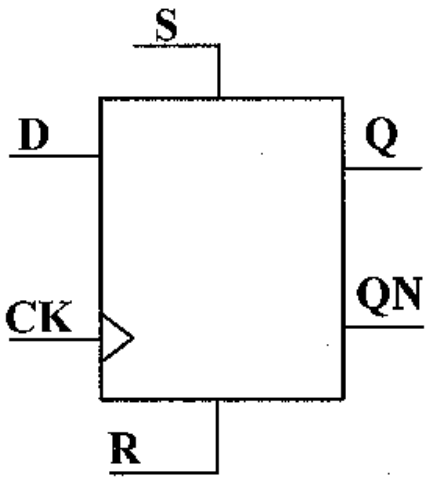
SYNTAXE:

ENTITY <nom entité> IS
déclarations génériques;
déclaration des ports;
END <nom entité>;

EXEMPLE:

bascule D type CD4013A

```
ENTITY CD4013A IS
        port (CK, D, R, S : in std_Iogic ;
        Q, QN : out std_Iogic
        );
END CD4013A;
```



l'architecture: ARCHITECTURE

⇒ décrit le comportement du composant VHDL
⇒ les instructions sont exécutées simultanément à l'exception de celles incluses dans les PROCESS.
⇒ les descriptions peuvent être structurelles, comportementales ou mixtes.

SYNTAXE:

```
ARCHITECTURE <nom arch> OF <nom entité> IS
        Zone de déclaration
BEGIN
        Description de la structure logique
END <nom_arch>;
```

EXEMPLE:

```
-- multiplexeur 3 entrées (a, b, sel), 1 sortie (s)

Library IEEE;
USE IEEE.std_logic_1164.all;

ENTITY multi_2 IS
        Port (a, b, sel: in std_Iogic;
        s : out std_Iogic
        );
END multi_2;

ARCHITECTURE arch_multi_2 OF multi_2 IS
BEGIN
s <= (a AND NOT sel) OR (b AND sel);
END arch_multi_2 ;
```

MODELISATION DES ARCHITECTURES: LES SIGNAUX

⇒ **équipotentielles enterrées** reliant différentes fonctions
⇒ déclarés à l'intérieur de l'architecture

exemple de déclaration:

SIGNAL sig_Drampe : std _logic_vector (5 downto 0) ;

Initialisation du signal

- **sig_Drampe <= "100111";** (doubles cotes si plusieurs bits)
- **sig_Drampe <= X "27";**

Pour initialiser un bit (le 3 par ex.):

- **sig_Drampe(3) <= '0';** (simples cotes si un bit)

Affectation conditionnelle des signaux

```
nom_signal  <=    valeur_signal     WHEN condition1 ELSE
                  valeur_signal     WHEN condition2 ELSE
                  .....
                  valeur_signal ;
```

Affectation selective des signaux

```
WITH <expression> SELECT
nom_signal  <=    <valeur_signal> WHEN conditionl,
                  <valeur_signal> WHEN condition2,
                  ••••
                  <valeur_signal> WHEN OTHERS ;
```

Modélisation des architectures: **LE PROCESS**

⇒ décrit de **manière séquentielle** le comportement de fonctions
⇒ ne **s'exécute** que si un **évènement** se produit
⇒ une architecture peut avoir **plusieurs PROCESS**.
Ceux-ci s'exécutent **simultanément**.
⇒ les **liens** entre process sont effectués par les **signaux**

Syntaxe:

```
[nom_process]:   PROCESS (a,b)
                 BEGIN
                        liste des instructions
                 END PROCESS [nom_process];
```

⇒ ne s'exécute que si un évènement se produit sur **a** ou **b**.
⇒ on appelle (a, b) la liste de sensibilité du process.

Affectation des signaux:

⇒ peut se faire à l'intérieur ou à l'extérieur du process (signal <= valeur)
⇒ ne prend effet qu'à la fin de celui-ci.

Instructions utilisables:

- **IF ...THEN...ELSE**
- **FOR** paramètre **IN** intervalle **LOOP**
- **CASE** signal **IS**

  WHEN valeur1 => instructions séquentielles;

  WHEN valeur2 => instructions séquentielles;

  …..

  END CASE;

MODELISATION DES ARCHITECTURES: LES VARIABLES

⇒ correspondent à un **stockage temporaire**
⇒ déclarées à l'intérieur d'un **PROCESS**
⇒ sont **mises à jour immédiatement**

exemple de déclaration:

VARIABLE var_Drampe: std _logic_vector (5 downto 0) ;

Initialisation de la variable

- var_Drampe := "100111"; (doubles cotes si plusieurs bits)
- var_Drampe := X "27";

Pour initialiser un bit (le 3 par ex.):

- var_Drampe(3):= '0'; (simples cotes si un bit)

Affectation conditionnelle des variables:

```
nom variable:=   valeur variable WHEN conditionl  ELSE
                 valeur variable WHEN condition2  ELSE
                 …..
                 valeur_variable;
```

Affectation selective des variables:

```
WITH <param> SELECT
nom_variable :=  <valeur_variable> WHEN <valeur-param>,
                 <valeur_variable> WHEN <valeur-param>,
                 ....
                 <valeur_variable> WHEN OTHERS ;
```

Modélisation des architectures: LES INSTRUCTIONS SÉQUENTIELLES

⇒ facilitent la description d'un comportement
⇒ obligatoirement à l'intérieur d'un **PROCESS**

Instructions utilisables:

⇒ **IF...THEN...ELSE**

⇒ **FOR** paramètre **IN** intervalle **LOOP**

⇒ **CASE** signal **IS**

```
WHEN valeur1 => instructions séquentielles;
WHEN valeur2 => instructions séquentielles;
….
END CASE;
```

Exemple: INSTRUCTION FOR…LOOP: Compteur CD 4040

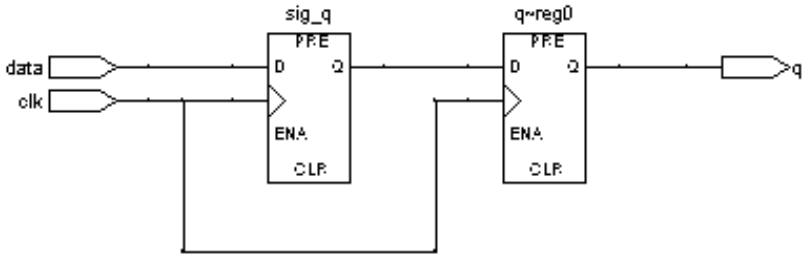
```
library IEEE;
        use IEEE.std_logic_1164.all;
-- description des entités du CD4040 (compteur 12 bits)
ENTITY CD4040 IS
        port        (clock, reset : in std_logic ;
                     q : out std_logic_vector (11 downto 0)
                     );
END CD4040;
-- description de l'architecture
ARCHITECTURE arch_CD4040 of CD4040 IS
SIGNAL sig_q : std_logic_vector (11 downto 0) ;
BEGIN
        PROCESS (clock, reset)
        VARIABLE var_q : std_logic_vector (11 downto 0) ;
        VARIABLE carry : std_logic ;
        BEGIN
                IF reset = '1' THEN
                        sig_q <= "000000000000" ;
                ELSIF clock 'event and clock = '0' THEN
                carry := '1' ;
                        FOR i in 0 to 11 LOOP
                                var_q (i) := sig_q (i) xor carry ;
                                carry := sig_q (i) and carry ;
                        END LOOP;
                END IF;
                sig_q <= var_q ;
        END PROCESS;
        q <= sig_q ;
END arch_CD4040 ;
```

Exemple: INSTRUCTION **IF ..THEN..ELSE** : bascule D type CD4013

```
library IEEE;
        use IEEE.std_logic_1164.all;
 -- description des entités du CD4013
 ENTITY CD4013 IS
        port        (ck, d, s, r : in std_logic ;
                     q, qn : out std_logic
                     );
END CD4013;
 -- description de l'architecture
 ARCHITECTURE arch_CD4013 of CD4013 IS
SIGNAL sig_q : std_logic ;
SIGNAL sig_qn : std_logic ;
BEGIN
        PROCESS (ck, r, s)
        BEGIN
                IF r= '1' THEN
                        sig_q <= '0' ;
                ELSIF s = '1' THEN
                         sig_q <= '1' ;
                ELSIF ck 'event and ck = '1' THEN
                sig_q <= d ;
                END IF;
                sig_qn <= NOT sig_q ;
        END PROCESS;
        q <= sig_q ;
        qn <= sig_qn ;
END arch_CD4013 ;
```

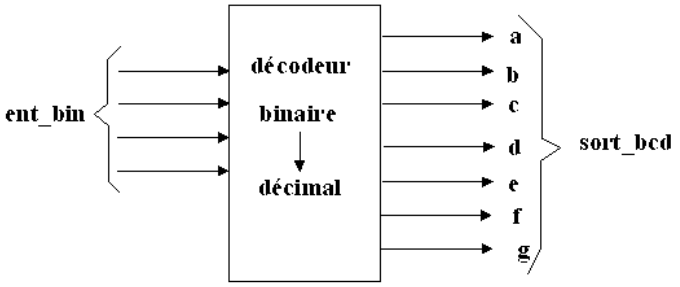
Bascule cd4013 : MAUVAISE REPRESENTATION

```
ENTITY cd4013b is
        port        (clk, data : in std_logic;
                     q : out std_logic);
END cd4013b;
ARCHITECTURE arch_cd4013b of cd4013b is
signal sig_q : std_logic;
begin
        process (clk)
        begin
                if clk 'event and clk = '1' then
                sig_q <= data ;          -- un registre utilisé (affectation conditionnée par clk)
                q <= sig_q;              -- un autre registre utilisé  (affectation conditionnée par clk)
                End if ;
        end process;
END arch_cd4013b;
```



Le langage VHDL

```
--          Exemple utilisation de CASE: décodeur BCD - 7 segments
--          a       b       c       d       e       f       g
            0       1       2       3       4       5       6

library IEEE;
            use IEEE.std_logic_1164.all;
                        --          description des entités du décodeur
ENTITY decodeur IS
            port        (ent_bin : in std_logic_vector (3 downto 0) ;
                        sort_bcd : out std_logic_vector (6 downto 0)
                        );
END decodeur;
                    --          description de l'architecture (sorties actives à 1)
 ARCHITECTURE arch_decodeur of decodeur IS
BEGIN
            PROCESS (ent_bin)
            BEGIN
                        CASE ent_bin IS
                                    WHEN "0000" =>      sort_bcd <= "0111111";  -- 0
                                    WHEN "0001" =>      sort_bcd <= "0000110";  -- 1
                                    WHEN "0010" =>      sort_bcd <= "1011011";  -- 2
                                    WHEN "0011" =>      sort_bcd <= "1001111";  -- 3
                                    WHEN "0100" =>      sort_bcd <= "1100110";  -- 4
                                    WHEN "0101" =>      sort_bcd <= "1101101";  -- 5
                                    WHEN "0110" =>      sort_bcd <= "1111101";  -- 6
                                    WHEN "0111" =>      sort_bcd <= "0000111";  -- 7
                                    WHEN "1000" =>      sort_bcd <= "1111111";  -- 8
                                    WHEN "1001" =>      sort_bcd <= "1101111";  -- 9
                                    WHEN others =>      sort_bcd <= "0000000";  --
éteint
                        END CASE;
            END PROCESS;
END arch_decodeur ;
```



Exemple: Additionneur 4 bits - description structurelle-

```
library IEEE;
use IEEE.std_logic_1164.all;
-- description des entrées / sorties
ENTITY add_4 IS
        port    (cin, a0,a1,a2,a3, b0,b1,b2,b3 : in std_logic;
                s0,s1,s2,s3, cout : out std_logic
                );
END add_4;
-- description de l'architecture
ARCHITECTURE arch_add_4 of add_4 IS
        signal  r1, r2, r3 : std_logic;
BEGIN
        s0 <= a0 XOR b0 XOR cin ;
        r1 <= (a0 and b0) or (cin and (a0 xor b0));
        s1 <= a1 XOR b1 XOR r1 ;
        r2 <= (a1 and b1) or (r1 and (a1 xor b1));
        s2 <= a2 XOR b2 XOR r2 ;
        r3 <= (a2 and b2) or (r2 and (a2 xor b2));
        s3 <= a3 XOR b3 XOR r3 ;
        cout <= (a3 and b3) or (r3 and (a3 xor b3));
END arch_add_4 ;
```

Exemple: Additionneur n bits – description par un process

```
 library IEEE;
use IEEE.std_logic_1164.all;
ENTITY add_4_it IS
generic (n : integer :=8);
                port          (a, b : in std_logic_vector ((n-1) downto 0);
                               cin            : in std_logic;
                               cout: out std_logic;
                               s : out std_logic_vector ((n-1) downto 0)
                               );
END add_4_it;
ARCHITECTURE arch_add_4_it of add_4_it IS
                BEGIN
                process (a,b, cin)
                variable var_s : std_logic_vector ((n-1) downto 0);
                variable var_r : std_logic;
                begin
                var_r := cin;
                               for i in 0 to (n-1) loop
                               var_s(i) := a(i) XOR b(i) XOR var_r ;
                               var_r := ((a(i) and b(i)) or ((a(i) xor b(i)) and var_r));
                               end loop;
                s <= var_s ;
                cout <= var_r ;
                end process;
                END arch_add_4_it ;
```

Exemple: Additionneur n bits – Représentation comportementale

```
library IEEE;
        use IEEE.std_logic_1164.all;
        use IEEE.std_logic_arith.all;

                -- description des entités de l'additionneur 2n entrées

ENTITY adder_simple IS
generic (n: integer := 255 );
        port        (a,b : in integer range 0 to n ;
                     s : out integer range 0 to 2*n
                     );
END adder_simple;

                -- description de l'architecture

ARCHITECTURE arch_adder_simple of adder_simple IS
BEGIN
s <= a + b ;
END arch_adder_simple;
```

Utilisation d'un paramètre générique:

⇒ Le paramètre générique permet de disposer d'un seul modèle pour un type de composant.

Exemple : compteur modulo « **modulo** »

```
library IEEE;
        use IEEE.std_logic_1164.all;

            -- description des entités du CD40xx (compteur n bits)

ENTITY CD40xx IS
        generic (modulo : integer := 255) ;
        port    (clock, reset : in std_logic ;
                 q : out integer range 0 to modulo
                 );
END CD40xx;
```

Le langage VHDL- la conception hiérarchisée

⇒ Pour la conception de circuits complexes.

⇒ Décomposition du problème en fonctions.

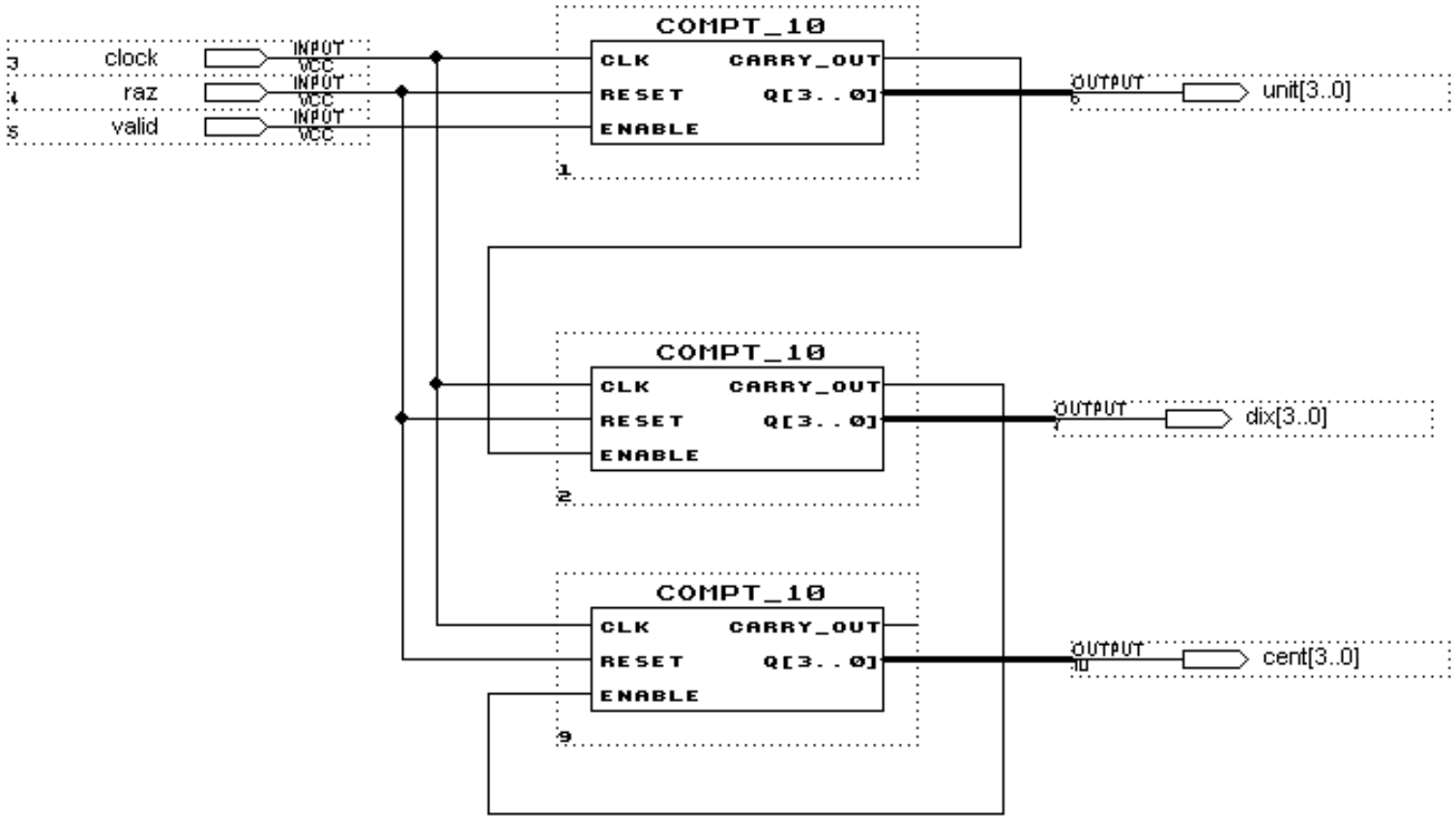
⇒ Création d'une entité pour chacune des fonctions identifiées (si elles sont différentes).

⇒ Interconnexion des fonctions par l'instruction **PORT MAP**.

Si une fonction est utilisée plusieurs fois, on crée des instances de cette fonction.

Exemple :

Conception d'un circuit de comptage **compt_1000** (de 0 à 999) à partir de trois compteurs base 10 :



Le langage VHDL- la conception hiérarchisée

⇒ Création d'un compteur base 10 (**compt_10**) qui sera mis en cascade.

Entité du fichier de description du circuit **compt_10** :

```
ENTITY compt_10 IS
        PORT (clk, reset, enable: in  std_logic ;
              carry_out : inout std_logic;
              q : inout integer range 0 to 9);
END compt_10;
```

⇒ **inout** permet la relecture des ports .
(carry_out et q sont exploités à l'extérieur et à l'intérieur du circuit).

Le langage VHDL- la conception hiérarchisée

Description du circuit **compt_1000**:

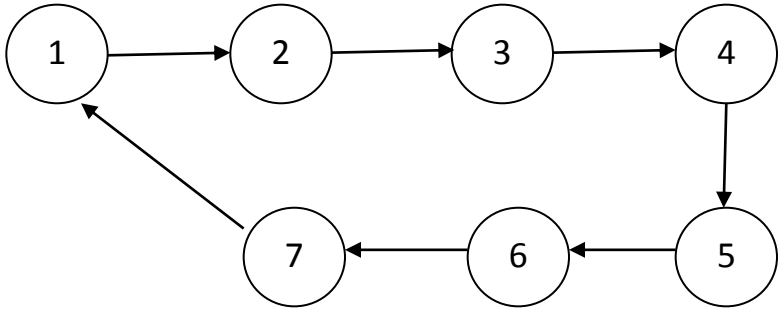
```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
 ENTITY compt_1000 IS
                PORT (clock, raz, valid: IN  std_logic ;
                unit, dix, cent : inout integer range 0 to 9);
END compt_1000;
ARCHITECTURE arch_compt_1000 OF compt_1000 IS
signal cr_dix, cr_cent, cr_mille : std_logic;
                                -- déclaration du ou des composants utilisés
                component compt_10
                                PORT (clk, reset, enable: IN  std_logic ;
                                carry_out : inout std_logic;
                                  q : inout integer range 0 to 9);
                end component;
BEGIN
                U1: compt_10                    -- U1 est une instance de compt_10
                port map  (clk => clock, reset=>raz, enable=>valid, carry_out=>cr_dix, q=>unit);

                U2: compt_10                    -- U2 est une 2ème instance de compt_10
                port map (clk => clock, reset=>raz, enable=>cr_dix, carry_out=>cr_cent, q=>dix);

                U3: compt_10                    -- U3 est une 3ème instance de compt_10
                port map (clk => clock, reset=>raz, enable=>cr_cent, carry_out=>cr_mille, q=>cent);

END arch_compt_1000;
```

⇒ **port map permet le câblage des signaux .**

Synthèse des machines à états

⇒ Un seul état actif à tout instant
⇒ On peut utiliser une variable ou un signal de type **vector** ou **integer**.
⇒ Si N est le nombre d'états de la machine, alors la taille du type **integer** sera N
⇒ Si c'est un type **vector** sa taille sera n de telle sorte que $N \leq 2^n$

Exemple : compteur synchrone modulo 7 (compte de 0 à 6 donc 3 sorties Q2 Q1 Q0)
Graphe d'état (chaque transition est validée par un front d'horloge non représenté).
Un signal de reset remet le circuit dans l'état 1 (non représenté).



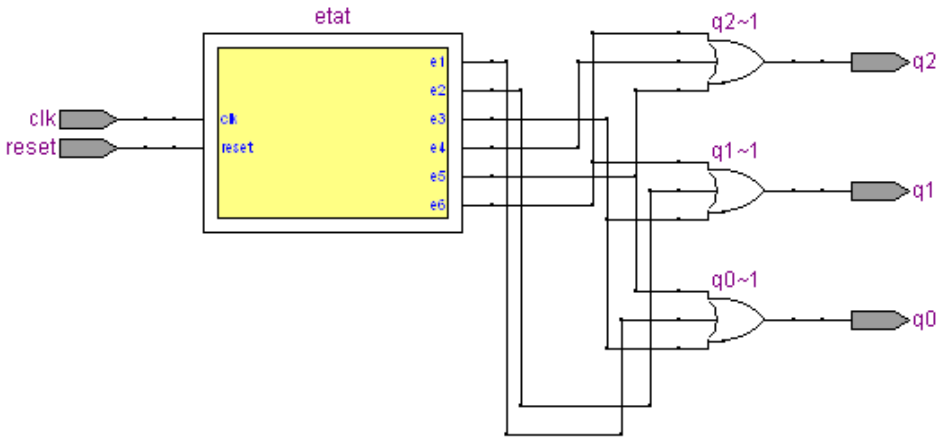
Etat	Q2 Q1 Q0
1	000
2	001
3	010
4	011
5	100
6	101
7	110

Le langage VHDL- machines à états

```
 LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY compt_etat_7 IS
        PORT (clk, reset: IN  std_logic ;
        q0, q1, q2     : OUT std_logic);
END compt_etat_7;

ARCHITECTURE arch_compt_etat_7 OF compt_etat_7 IS
BEGIN
        process (reset, clk)
        variable etat : integer range 1 to 7;
        begin
        if reset ='1' then
        etat:= 1;
        q0 <= '0'; q1 <= '0'; q2 <= '0';
        elsif clk'event and clk='1' then
        case etat is
        when 1 =>
        etat:=2;
        q0<='1'; q1<='0'; q2<='0';
        when 2 =>
        etat:=3;
        q0<='0'; q1<='1'; q2<='0';
        when 3 =>
        etat:=4;
        q0<='1'; q1<='1'; q2<='0';
        when 4 =>
        etat:=5;
        q0<='0'; q1<='0'; q2<='1';
        when 5 =>
        etat:=6;
        q0<='1'; q1<='0'; q2<='1';
        when 6 =>
        etat:=7;
        q0<='0'; q1<='1'; q2<='1';
        when others =>
        etat :=1;
        q0<='0'; q1<='0'; q2<='0';
        end case;
        end if;
        end process;
END arch_compt_etat_7;
```

Le langage VHDL- machines à états

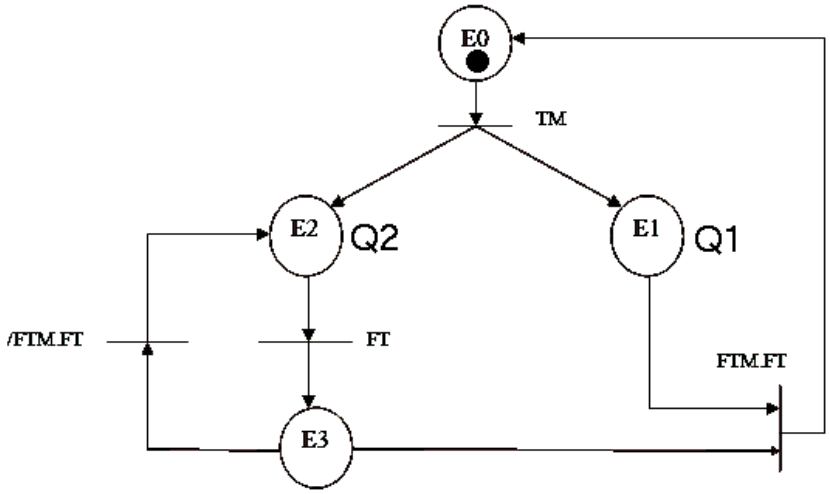
```
 LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY compt_etat_7a IS
        PORT (clk, reset:       IN  std_logic ;
                q0, q1, q2:     OUT std_logic);
END compt_etat_7a;

ARCHITECTURE arch_compt_etat_7a OF compt_etat_7a IS
TYPE STATE_TYPE IS (e0,e1,e2,e3,e4,e5,e6);
signal etat : state_type;
BEGIN
        process (reset, clk)
        begin
        if reset ='1' then
        etat<= e0;
        elsif clk'event and clk='1' then
        case etat is
        when e0 =>
        etat<=e1;
        when e1 =>
        etat<=e2;
        when e2 =>
        etat<=e3;
        when e3 =>
        etat<=e4;
        when e4 =>
        etat<=e5;
        when e5 =>
        etat<=e6;
        when others =>
        etat <=e0;
        end case;
        end if;
        end process;
        q0 <= '1' when etat=e1 or etat=e3 or etat=e5 else '0';
        q1 <= '1' when etat=e2 or etat=e3 or etat=e6 else '0';
        q2 <= '1' when etat=e4 or etat=e5 or etat=e6 else '0';
END arch_compt_etat_7a;
```



Le langage VHDL- les réseaux de Petri

Synthèse des Réseaux de Petri :

⇒Plusieurs places marquées à tout instant
⇒ Associer un signal à chaque place
⇒ Les signaux seront nécessairement de type bit.




```
ENTITY petri_1 IS
        PORT (reset, clk, tm, ftm, ft : IN  std_logic;
        q1, q2 : out std_logic);
END petri_1;
```

```
ARCHITECTURE arch_petri_1 OF petri_1 IS
signal      e0, e1, e2, e3 : std_logic ;
Begin
        process (clk, reset)
        begin
        if reset='0' then
                e0 <='1'; e1 <='0'; e2 <='0'; e3 <='0';
                elsif clk'event and clk ='1' then
                if e0 = '1'  and tm = '1' then
                e0 <='0'; e1 <='1'; e2 <='1';
                end if;
                if e2 = '1'  and ft = '1' then
                e3 <='1'; e2 <='0';
                end if;
                if (e1 and e3 and ftm and ft) ='1' then
                e0 <='1'; e1 <='0'; e3 <='0';
                end if;
                if (e3 and ft and not ftm) = '1' then
                e3 <='0'; e2 <='1';
                end if;
                end if;
        end process;
                q1 <= e1;
                q2 <= e2;
END arch_petri_1;
```

<u>Nota</u> : pour ne pas surcharger le réseau, les signaux de reset et clk n'apparaissent pas. « reset » remet le réseau dans sa configuration initiale : place E0 marquée

Quelques règles pour une bonne conception VHDL

- Avoir une vision « circuit » de son projet
- décomposer le projet en fonctions simples simulables indépendamment
- utiliser une seule horloge de base
- ne coder en VHDL que lorsque la fonction à réaliser a été parfaitement définie
- se souvenir que le VHDL n'est qu'un mode de représentation d'un circuit
- n'utiliser que des bascules type « flip-flop » (pas de Latchs)
- prévoir un signal externe de « reset » qui remet le circuit dans une configuration connue
- synchroniser les signaux d'entrée sur l'horloge de base
- …..