

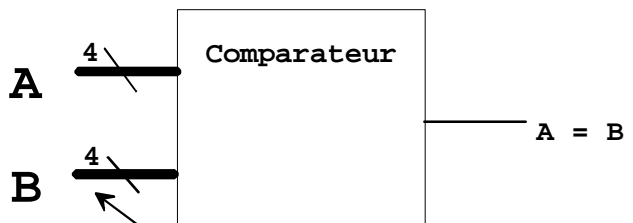
Le VHDL par l'exemple

Etiquette

Liste de sensibilité

```
p1: PROCESS (clk)
BEGIN
    IF (clk'event AND clk='1') THEN
        q <= d;
    END IF;
END PROCESS;
```

Attribut du signal clk



Un Bus de quatre fils

Description vhdl

BIBLIOTHEQUE UTILISEE

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

ENTITY

```
ENTITY eqcomp4 IS
    PORT ( a : IN STD_LOGIC_VECTOR(3 downto 0);
          b : IN STD_LOGIC_VECTOR(3 downto 0);
          aeqb : OUT STD_LOGIC);
END eqcomp4;
```

ARCHITECTURE

```
ARCHITECTURE logique OF eqcomp4 IS
BEGIN
    aeqb <= '1' WHEN ( a = b ) ELSE '0';
END logique;
```

LE VHDL PAR L'EXEMPLE

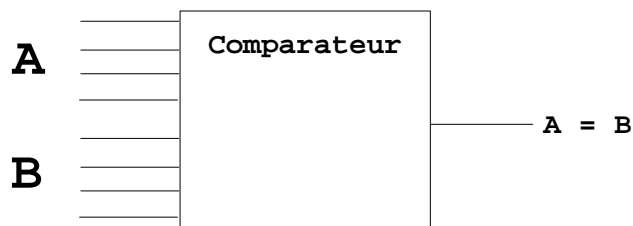
(A) OPERATIONS COMBINATOIRES.	3
(a-1) Etude d'un comparateur de deux fois 4 bits.	3
(a-2) Organisation simple d'une description vhdl.	3
(a-3) Exercices :	5
(a-4) Différentes architectures.	6
(a-5) Description modulaire.	7
(a-6) Exercices.	10
(a-7) le multiplexeur.	11
(B) LES OPERATIONS SEQUENTIELLES ET LE PARALLELISME.	12
(b-1) La logique synchrone, la bascule D.	12
(b-2) La bascule D avec un reset asynchrone.	13
(b-3) Description d'un compteur de quatre bits.	14
(b-4) Exercices.	14
(C) LES MACHINES D'ETATS.	15
(c-1) Détection de début et de fin d'impulsion.	15
(c-2) Machine de Moore.	16
(c-3) Codage des états :	17
(c-4) Machine de Mealy.	18
(c-5) Exercices.	19
(D) DIRECTIVES DE SYNTHÈSE.	21
(d-1) Imposer les broches utilisées.	21
(d-2) Choix du composant.	21
(d-3) Exemple d'utilisation des attributs de synthèse.	21
(d-4) Exercice.	21
Annexe 1 : bibliographie	23

LE VHDL PAR L'EXEMPLE

(A) OPERATIONS COMBINATOIRES.

(a-1) Etude d'un comparateur de deux fois 4 bits.

Etudions un comparateur chargé de comparer deux nombres de quatre éléments binaires. Il est représenté sur le schéma ci-dessous :

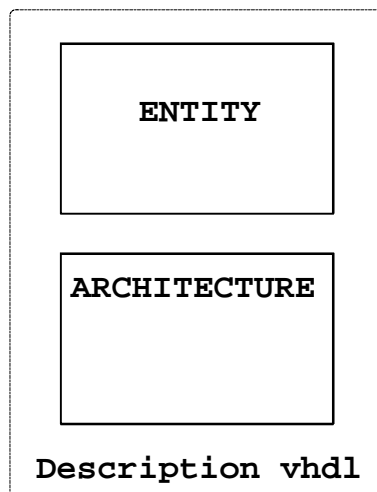


Nous pouvons y distinguer les différentes parties le constituant, à savoir :

Les entrées A et B de quatre bits chacune, la sortie A = B. Le corps du design délimite la frontière entre la structure interne du comparateur et le monde extérieur.

(a-2) Organisation simple d'une description vhdl.

Une description vhdl est composée de deux parties: une entité et une architecture.



L'entité décrit l'interface entre le design et le monde extérieur.

L'architecture décrit la structure interne du composant. Il y a plusieurs façons de décrire ce fonctionnement.

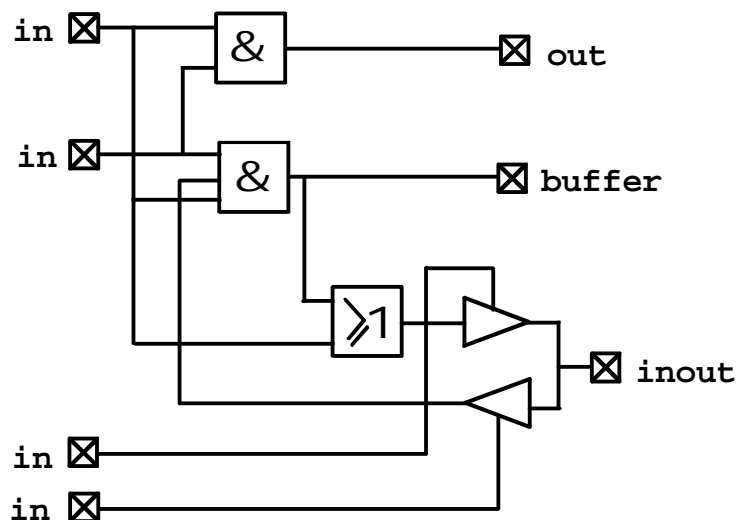
La description vhdl du comparateur est donnée ci-dessous :

```
-- comparateur de deux fois quatre bits
ENTITY eqcomp4 IS
  PORT ( a0,a1,a2,a3 : IN BIT;
         b0,b1,b2,b3 : IN BIT;
         aeqb : OUT BIT);
END eqcomp4;

ARCHITECTURE logique OF eqcomp4 IS
BEGIN
  aeqb <= '1' WHEN (
    (a0=b0) and
    (a1=b1) and
    (a2=b2) and
    (a3=b3)
  )
  ELSE '0';
END logique;
```

L'entité est décrite par l'intermédiaire de l'instruction PORT. Celle-ci liste les différentes entrées et sorties du design. Pour chaque donnée transférée à travers un PORT on définit son **mode** et son **type**.

Le **mode** peut être de quatre valeurs différentes : IN, OUT, BUFFER, INOUT.



Un port de mode IN décrit un port qui entre dans l'entité, le pilote (driver) est extérieur à l'entité.

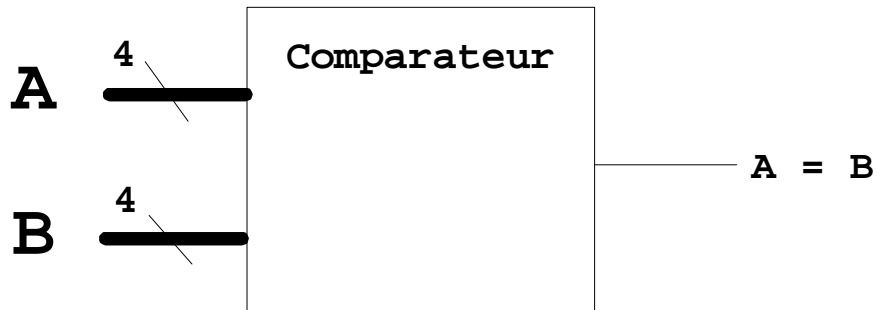
Un port de mode OUT décrit un port qui sort de l'entité, le pilote associé est à l'intérieur de l'entité.

Un port de mode INOUT décrit un port bidirectionnel, le pilote peut être à l'intérieur ou à l'extérieur de l'entité.

Un port de mode BUFFER décrit un port qui sort de l'entité mais qui est utilisé en interne en entrée. Le pilote est à l'intérieur de l'entité.

Le **type** peut être un type bit, bit_vector, std_logic, std_logic_vector, un entier ou encore un type défini par l'utilisateur. Certains type nécessitent, pour être employés, l'utilisation de bibliothèque particulière.

Utilisons un autre type en regroupant nos entrées en deux vecteurs de quatre bits chacun :



```

-- comparateur de deux fois quatre bits
--

-- utilisation de la bibliothèque nécessaire au type std_logic
library ieee;
use ieee.std_logic_1164.all;

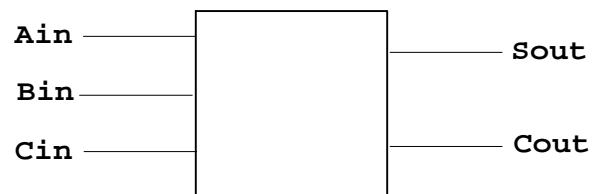
ENTITY eqcomp4 IS
    PORT ( a : IN STD_LOGIC_VECTOR(3 downto 0);
          b : IN STD_LOGIC_VECTOR(3 downto 0);
          aeqb : OUT STD_LOGIC);
END eqcomp4;

ARCHITECTURE logique OF eqcomp4 IS
BEGIN
    aeqb <= '1' WHEN ( a = b ) ELSE '0';
END logique;

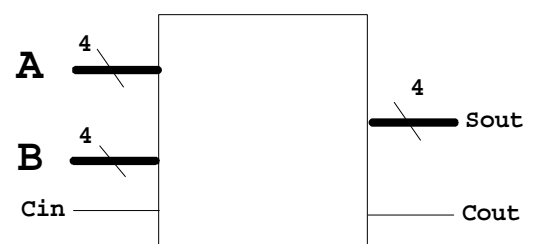
```

(a-3) Exercices :

Exercice 1 : Définir l'entité décrivant un additionneur de deux fois un bit.



Exercice 2 : Définir l'entité décrivant un additionneur de deux mots de quatre bits.



(a-4) Différentes architectures.

Différentes architectures sont possibles pour décrire le comportement attendu d'un design. Les voici présentées avec l'exemple d'un comparateur de deux bits .

Comparateur



Utilisation d'une assignation conditionnelle **when** :

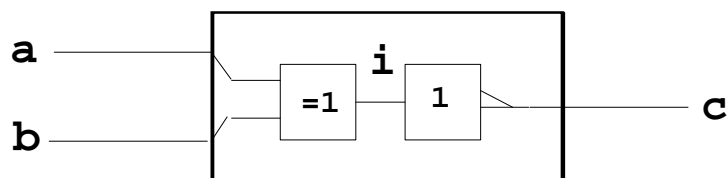
```
ARCHITECTURE logique OF comparateur IS
BEGIN
    c <= '1' WHEN ( a = b ) ELSE '0';
END logique;
```

Utilisation de l'équation logique de la comparaison :

```
ARCHITECTURE operateur OF comparateur IS
BEGIN
    c <= not ( a xor b )
END operateur;
```

Réalisation « matérielle » de l'équation logique avec des composants xor et inv définis par ailleurs dans une bibliothèque :

Comparateur



```
ARCHITECTURE composant OF comparateur IS
SIGNAL i : BIT;
BEGIN
    comp1 : xor2 PORT MAP (a,b,i);
    comp2 : inv PORT MAP (i,c);
END composant;
```

A noter que la liaison à l'intérieur de l'architecture doit être réalisée par un **signal** , dans cet exemple le signal est de type **bit**.

(a-5) Description modulaire.

Il n'est pas nécessaire de définir tous les composants ou architectures utilisées dans un design vhdl dans le même fichier que le fichier principal. Il est possible de découper le projet en plusieurs morceaux. Ces morceaux sont rangés dans des **paquetages**. Ils peuvent être alors utilisés par tout autre projet décrit en vhdl et qui fera appel à ce paquetage, l'appel est précisé dans une déclaration **use** .

Pour illustrer cette description modulaire nous allons à partir du comparateur de deux fois quatre bits, réaliser un découpage entre un paquetage de portes logiques élémentaires, et le corps principal du projet.

Voici tout d'abord la description complète non découpée du comparateur de deux mots de quatre bits :

```
-----
--                               description d'un opérateur non xor
library ieee;
use ieee.std_logic_1164.all;

ENTITY nonxor IS
    PORT ( x , y : IN STD_LOGIC; xn : OUT STD_LOGIC);
END nonxor;

ARCHITECTURE operateur OF nonxor IS
BEGIN
    xn <= not ( x xor y );
END operateur;
-----
--                               description d'un opérateur et à quatre entrées
library ieee;
use ieee.std_logic_1164.all;

ENTITY and4 IS
    PORT ( a , b , c , d : IN STD_LOGIC; e : out STD_LOGIC );
END and4;

ARCHITECTURE operateur OF and4 IS
BEGIN
    e <= ((a and b) and ( c and d));
END operateur;
-----
-- description du comparateur de deux fois quatre bits
-----
library ieee;
use ieee.std_logic_1164.all;

ENTITY cmp4 IS
    PORT ( a : IN STD_LOGIC_VECTOR(3 downto 0);
          b : IN STD_LOGIC_VECTOR(3 downto 0);
          aeqb : OUT STD_LOGIC);
END cmp4;

ARCHITECTURE composant OF cmp4 IS
SIGNAL i : STD_LOGIC_VECTOR(3 downto 0);
BEGIN
    ic1: nonxor PORT MAP (a(0),b(0),i(0));
    ic2: nonxor PORT MAP (a(1),b(1),i(1));
    ic3: nonxor PORT MAP (a(2),b(2),i(2));
    ic4: nonxor PORT MAP (a(3),b(3),i(3));
    ic5: and4 PORT MAP(i(0),i(1),i(2),i(3),aeqb);
END composant;
```

La même description mais en préparant la coupure en deux parties du fichier :

```
-- fichier cmp41.vhd
-----
-- description d'un opérateur non xor
library ieee;
use ieee.std_logic_1164.all;

ENTITY nonxor IS
    PORT ( x , y : IN STD_LOGIC; xn : OUT STD_LOGIC);
END nonxor;

ARCHITECTURE operateur OF nonxor IS
BEGIN
    xn <= not ( x xor y );
END operateur;
-----
-- description d'un opérateur et à quatre entrées
library ieee;
use ieee.std_logic_1164.all;

ENTITY and4 IS
    PORT ( a , b , c , d : IN STD_LOGIC; e : out STD_LOGIC );
END and4;

ARCHITECTURE operateur OF and4 IS
BEGIN
    e <= ((a and b) and ( c and d));
END operateur;

-----
-- description du comparateur de deux fois quatre bits
-----
library ieee;
use ieee.std_logic_1164.all;

ENTITY cmp4 IS
    PORT ( a : IN STD_LOGIC_VECTOR(3 downto 0);
          b : IN STD_LOGIC_VECTOR(3 downto 0);
          aeqb : OUT STD_LOGIC);
END cmp4;

ARCHITECTURE composant OF cmp4 IS

-- on précise ici la déclaration des composants que l'on souhaite utiliser dans
la description principale.
--
COMPONENT nonxor PORT ( x , y : IN STD_LOGIC; xn : OUT STD_LOGIC);
END COMPONENT;

COMPONENT and4 PORT ( a , b , c , d : IN STD_LOGIC; e : out STD_LOGIC );
END COMPONENT;
-----

SIGNAL i : STD_LOGIC_VECTOR(3 downto 0);

BEGIN
    ic1: nonxor PORT MAP (a(0),b(0),i(0));
    ic2: nonxor PORT MAP (a(1),b(1),i(1));
    ic3: nonxor PORT MAP (a(2),b(2),i(2));
    ic4: nonxor PORT MAP (a(3),b(3),i(3));
    ic5: and4 PORT MAP(i(0),i(1),i(2),i(3),aeqb);
END composant;
```

Coupure en deux parties et réalisation du **paquetage** portes :

```
-- paquetage de portes élémentaires
--
-- fichier portes.vhd

library ieee;
use ieee.std_logic_1164.all;

PACKAGE portes IS

    COMPONENT nonxor PORT ( x , y : IN STD_LOGIC; xn : OUT STD_LOGIC);
    END COMPONENT;

    COMPONENT and4 PORT ( a , b , c , d : IN STD_LOGIC; e : out STD_LOGIC );
    END COMPONENT;

END portes;

-----
-- description d'un opérateur non xor
--
library ieee;
use ieee.std_logic_1164.all;

ENTITY nonxor IS
    PORT ( x , y : IN STD_LOGIC; xn : OUT STD_LOGIC);
END nonxor;

ARCHITECTURE operateur OF nonxor IS
BEGIN
    xn <= not ( x xor y );
END operateur;

-----
-- description d'un opérateur et à quatre entrées
--
library ieee;
use ieee.std_logic_1164.all;

ENTITY and4 IS
    PORT ( a , b , c , d : IN STD_LOGIC; e : out STD_LOGIC );
END and4;

ARCHITECTURE operateur OF and4 IS
BEGIN
    e <= ((a and b) and ( c and d));
END operateur;
```

Le paquetage déclare les composants contenus à l'intérieur. Ces composants sont utilisables par tous les projets l'indiquant par une instruction **use**.

Voici le projet du comparateur quatre bits utilisant le paquetage portes :

```
-- comparateur de deux fois quatre bits
--
-- utilisation d'un paquetage de composants extérieurs
-- nommé portes
--
-- fichier cmp42.vhd
-----

library ieee;
use ieee.std_logic_1164.all;

use work.portes.all;

ENTITY cmp4 IS
    PORT ( a : IN STD_LOGIC_VECTOR(3 downto 0);
          b : IN STD_LOGIC_VECTOR(3 downto 0);
          aeqb : OUT STD_LOGIC);
END cmp4;

ARCHITECTURE composant OF cmp4 IS

SIGNAL i : STD_LOGIC_VECTOR(3 downto 0);

BEGIN
    ic1: nonxor PORT MAP (a(0),b(0),i(0));
    ic2: nonxor PORT MAP (a(1),b(1),i(1));
    ic3: nonxor PORT MAP (a(2),b(2),i(2));
    ic4: nonxor PORT MAP (a(3),b(3),i(3));
    ic5: and4 PORT MAP(i(0),i(1),i(2),i(3),aeqb);
END composant;
```

Cette description est la plus concise. Elle permet de plus de construire son projet avec des blocs compilés à part ce qui facilite la mise au point.

Ces paquetages peuvent être partagés avec d'autres projets, le travail en équipe de développement est possible avec le VHDL.

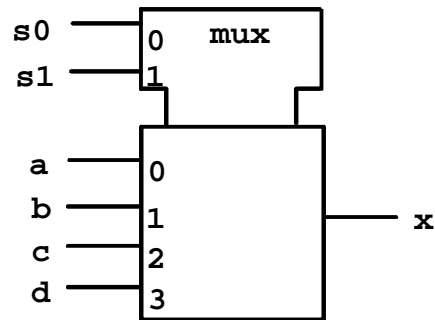
(a-6) Exercices.

[1] Reprendre les exemples ci-dessus et réaliser la synthèse d'un additionneur de deux fois 1 bit avec retenue en entrée.

[2] Mettre l'additionneur élémentaire dans le paquetage portes puis réaliser la synthèse d'un additionneur de deux mots de quatre bits.

(a-7) le multiplexeur.

Utilisons un multiplexeur pour illustrer les assignations conditionnelle **when** et sélective **with**.



Utilisation de l'assignation conditionnelle **when** :

```
-- multiplexeur
--
-- fichier mux1.vhd
-----

library ieee;
use ieee.std_logic_1164.all;

ENTITY mux1 IS
    PORT ( a,b,c,d : IN STD_LOGIC;
          s : IN STD_LOGIC_VECTOR(1 downto 0);
          x : OUT STD_LOGIC);
END mux1;

ARCHITECTURE archmux OF mux1 IS
BEGIN
    x <=  a WHEN ( s="00" ) ELSE
          b WHEN ( s="01" ) ELSE
          c WHEN ( s="10" ) ELSE
          d;
END archmux;
```

Utilisation de l'assignation sélective **with** :

```
-- multiplexeur
--
-- fichier mux2.vhd
-----

library ieee;
use ieee.std_logic_1164.all;

ENTITY mux2 IS
    PORT ( a,b,c,d : IN STD_LOGIC;
          s : IN STD_LOGIC_VECTOR(1 downto 0);
          x : OUT STD_LOGIC);
END mux2;

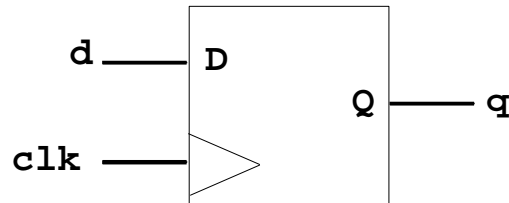
ARCHITECTURE archmux OF mux2 IS
BEGIN
    WITH s SELECT
    x <=  a WHEN "00" ,
          b WHEN "01" ,
          c WHEN "10" ,
          d WHEN OTHERS;
END archmux;
```

(B) LES OPERATIONS SEQUENTIELLES ET LE PARALLELISME.

(b-1) La logique synchrone, la bascule D.

Le vhdl permet de décrire des fonctionnements parallèles. La 'brique' de base de ces descriptions est le process.

Le process est une construction qui ne s'exécute que sur un changement détecté dans l'un des signaux auxquels il est sensible.



```
-- bascule D active sur front
--
-- fichier dff1.vhd
-----

library ieee;
use ieee.std_logic_1164.all;

ENTITY dff1 IS
    PORT ( d      : IN STD_LOGIC;
          clk    : IN STD_LOGIC;
          q      : OUT STD_LOGIC);
END dff1;

ARCHITECTURE archdff OF dff1 IS
BEGIN
    P1: PROCESS (clk)
    BEGIN
        IF (clk'event AND clk='1') THEN
            q <= d;
        END IF;
    END PROCESS p1;
END archdff;
```

La description de la bascule D repose sur le process ici repéré par l'étiquette P1.

Ce process est sensible à l'entrée clk. Il ne s'exécute que si il y a un changement sur l'entrée clk.

Lors de l'exécution les instructions composant le process s'exécutent séquentiellement.

Les sorties du process ne sont affectées qu'à la fin du déroulement de celui-ci.

Nous voyons ici l'utilisation d'un attribut du signal clk. Il s'agit de l'attribut **'event'**, celui-ci est vrai lorsqu'un changement sur le signal clk vient de se produire.

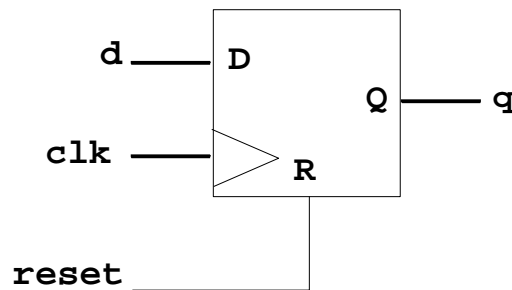
Ne pas oublier que les valeurs possibles du signal clk sont: 'U' 'X' '0' '1' 'Z' 'W' 'L' 'H' '-' si il est du type std_logic.

La condition logique clk'event and clk='1' détecte donc un front montant sur clk.

D'autres attributs prédéfinis sont disponibles, il est aussi possible de créer des attributs personnalisés.

(b-2) La bascule D avec un reset asynchrone.

Complétons la description de la bascule précédente pour y ajouter un reset asynchrone.



```
-- bascule D active sur front
--
-- fichier dff2.vhd
-----

library ieee;
use ieee.std_logic_1164.all;

ENTITY dff2 IS
    PORT (
        d      : IN STD_LOGIC;
        reset  : IN STD_LOGIC;
        clk    : IN STD_LOGIC;
        q      : OUT STD_LOGIC);
END dff2;

ARCHITECTURE archdff OF dff2 IS
BEGIN
    p1: PROCESS (clk,reset)
    BEGIN
        IF (reset='1') THEN
            q <= '0';
        ELSIF (clk'event AND clk='1') THEN
            q <= d;
        END IF;
    END PROCESS p1;
END archdff;
```

(b-3) Description d'un compteur de quatre bits.

Nous décrivons ci-dessous un compteur de quatre bits. Le comptage s'effectue directement par l'addition

```
count <= count + 1;
```

Cette opération sur la variable de type `std_logic_vector` est possible par l'utilisation du paquetage `std_arith`.

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

entity counter is port(
    clk, load: in std_logic;
    data:      in std_logic_vector(3 downto 0);
    count:     buffer std_logic_vector(3 downto 0));
end counter;

use work.std_arith.all;

architecture archcounter of counter is
begin
    upcount: process (clk)
        begin
            if (clk'event and clk= '1') then
                if load = '1' then
                    count <= data;
                else
                    count <= count + 1;
                end if;
            end if;
        end process upcount;
    end archcounter;
```

(b-4) Exercices.

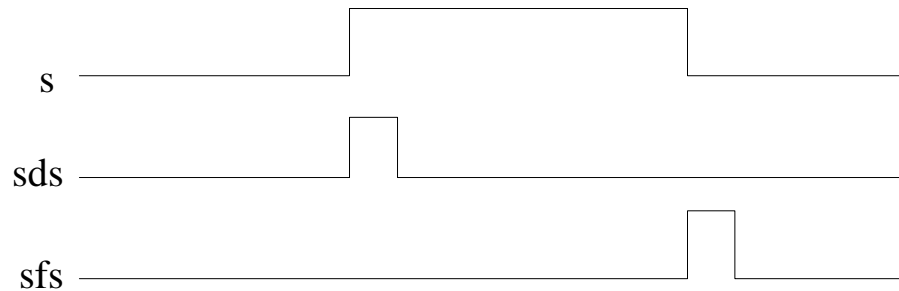
[1] Compléter la description ci-dessus pour ajouter la possibilité d'un reset et d'un preset asynchrones.

[2] Décrire un compteur de huit bits avec chargement synchrone et sorties trois états.

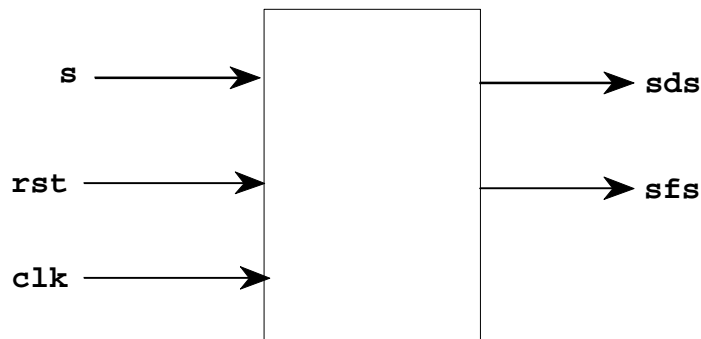
(C) LES MACHINES D'ETATS.

(c-1) Détection de début et de fin d'impulsion.

Réalisons un système séquentiel détectant le début et la fin d'une impulsion. Le chronogramme désiré est le suivant :



Le bilan des entrées et sorties est décrit ci-dessous, on y a ajouté la possibilité d'un reset avec l'entrée rst.



Synthèse de la machine d'état (voir texte ci-dessous) :

L'état est mémorisé dans un signal state de type states. Le type states précise la liste des valeurs possibles, c'est un type énuméré.

Un process sensible aux signaux clk et rst gère les évolutions de la machine d'état.

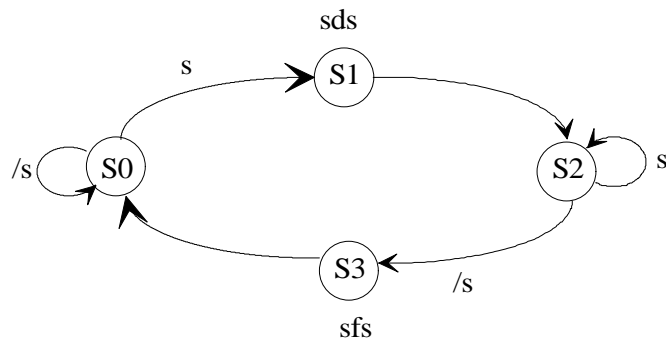
Les sorties sont calculées d'après la valeur de l'état dans des assignations concourantes de type <= ;

Les assignations <= et le process sont simultanément activées et chacun s'exécute dès que sa condition d'activation est réalisée à savoir :

- un changement de rst ou clk pour le process.
- un changement de la variable state pour sds et sfs.

(c-2) Machine de Moore.

Voici la machine de Moore répondant au cahier des charges ci-dessus :



Et la description vhdl correspondante :

```
--
-- Machine de Moore avec decodage combinatoire des sorties
--
-- P.Guérangé

library ieee;
use ieee.std_logic_1164.all;

entity moore1 is port(
    clk,rst : in std_logic;
    s : in std_logic;
    sds,sfs : out std_logic);
end moore1;

library ieee;
use ieee.std_logic_1164.all;

architecture archmoore1 of moore1 is
    type states is (s0,s1,s2,s3);
    signal state : states:=s0;
begin

    moore:process(rst,clk)
    begin
        if (rst='1') then
            state <= s0;
        elsif ((clk'event) and (clk='1')) then
            case state is
                when s0 => if (s='1') then state <= s1;
                            else state <= s0; end if;
                when s1 => state <= s2;
                when s2 => if (s='0') then state <= s3;
                            else state <= s2; end if;
                when s3 => state <= s0;
            end case;
        end if;
    end process;

    sds <= '1' WHEN (state=s1) else '0';
    sfs <= '1' WHEN (state=s3) else '0';

end archmoore1;
```

(c-3) Codage des états :

Le codage des états est réalisé de façon transparente par le compilateur. Il est néanmoins possible d'imposer un codage particulier en utilisant des directives de synthèse.

Les directives de synthèse sont écrites sous la forme d'attribut attaché à différentes parties du code vhdl.

Ici pour le codage des états nous utilisons l'attribut **enum_encoding** si nous voulons coder nous-mêmes les états:

syntaxe : *attribute enum_encoding of type_name:type is "string"*

```
type states is (s0,s1,s2,s3);  
attribute enum_encoding of states:type is " 00 01 10 11 ";
```

D'autres codages sont possibles avec l'attribut **state_encoding** comme par exemple le codage *one_hot_zero*, *one_hot_one*, *sequential*, ou *gray*. Décrivons ces codes :

Sequential : le codage est binaire normal.

Codage one_hot_zero : dans ce codage chaque code comporte un seul bit à 1, le code tous les bits nuls est utilisé.

Appliqué au cas précédent le codage donne (s0: 000) (s1: 001)
(s2: 010) (s3: 100)

Codage one_hot_one : dans ce codage chaque code comporte un seul bit à 1, le code tous les bits nuls n'est pas utilisé.

Appliqué au cas précédent le codage donne (s0: 0001) (s1: 0010)
(s2: 0100) (s3: 1000)

Syntaxe : *attribute state_encoding of type-name is value;*

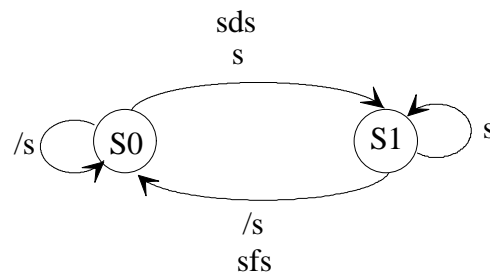
Avec value choisit parmi *sequential*, *one_hot_zero*, *one_hot_one*, *gray*.

```
type states is (s0,s1,s2,s3);  
attribute state_encoding of states:type is gray;
```

Les codages de type *one_hot_xxx* consomment plus de bascules qu'un codage binaire. Les équations sont cependant plus simples et le gain global est plus intéressant dans le cas de composants programmables possédant un grand nombre de registre internes.

(c-4) Machine de Mealy.

Reprenons le même problème en le décrivant avec une machine de Mealy. Le graphe de cette machine est décrit ci-dessous :



Voici la description vhdl correspondante :

```
-- Machine de Mealy
--
-- P.Guérangé
-----

library ieee;
use ieee.std_logic_1164.all;

entity mealy1 is port(
    clk,rst : in std_logic;
    s : in std_logic;
    sds,sfs : out std_logic);
end mealy1;

library ieee;
use ieee.std_logic_1164.all;

architecture archmealy1 of mealy1 is
    type states is (s0,s1);
begin

    mealy:process(rst,clk,s)
        variable state : states:=s0 ;
        begin
            if (rst='1') then
                state := s0;
            elsif ((clk'event) and (clk='1')) then
                case state is
                    when s0 => if (s='1')
                                then state := s1;
                                else state := s0;
                            end if;
                    when s1 => if (s='0')
                                then state := s0;
                                else state := s1;
                            end if;
                end case;
            end if;
            -- envoi des sorties
            if (state=s0) and (s='1')
                then sds <= '1'; else sds <= '0';
            end if;
            if (state=s1) and (s='0')
                then sfs <= '1'; else sfs <= '0';
            end if;
        end process;

end archmealy1;
```

La machine d'état présentée a une structure différente de la machine de Moore. Ici l'état est mémorisé dans une variable appartenant au process. (Et non plus à l'architecture comme dans le cas précédent.)

Les évolutions de la machine d'état sont prises en compte en donnant une valeur à la variable state par l'opérateur d'affectation := .

Les sorties sont donc gérées à l'intérieur du process en fonction de la valeur de la variable etat.

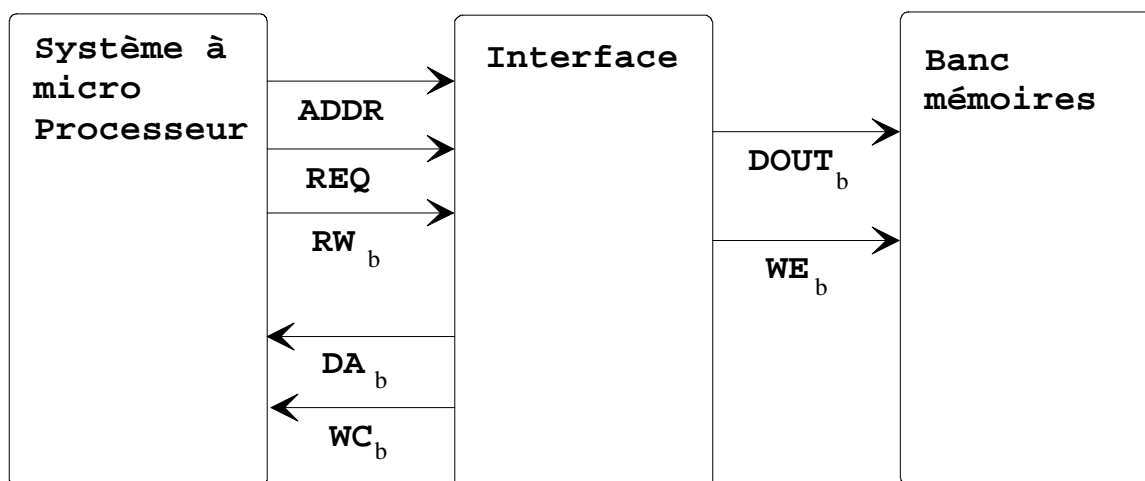
(c-5) Exercices.

[1] Modifier les machines d'états précédentes pour rendre la détection de début et de fin d'impulsion insensible à un parasite sur l'entrée s. Ce parasite ne dure pas plus d'un top d'horloge clk.

[2] Synthèse d'un banc mémoire.

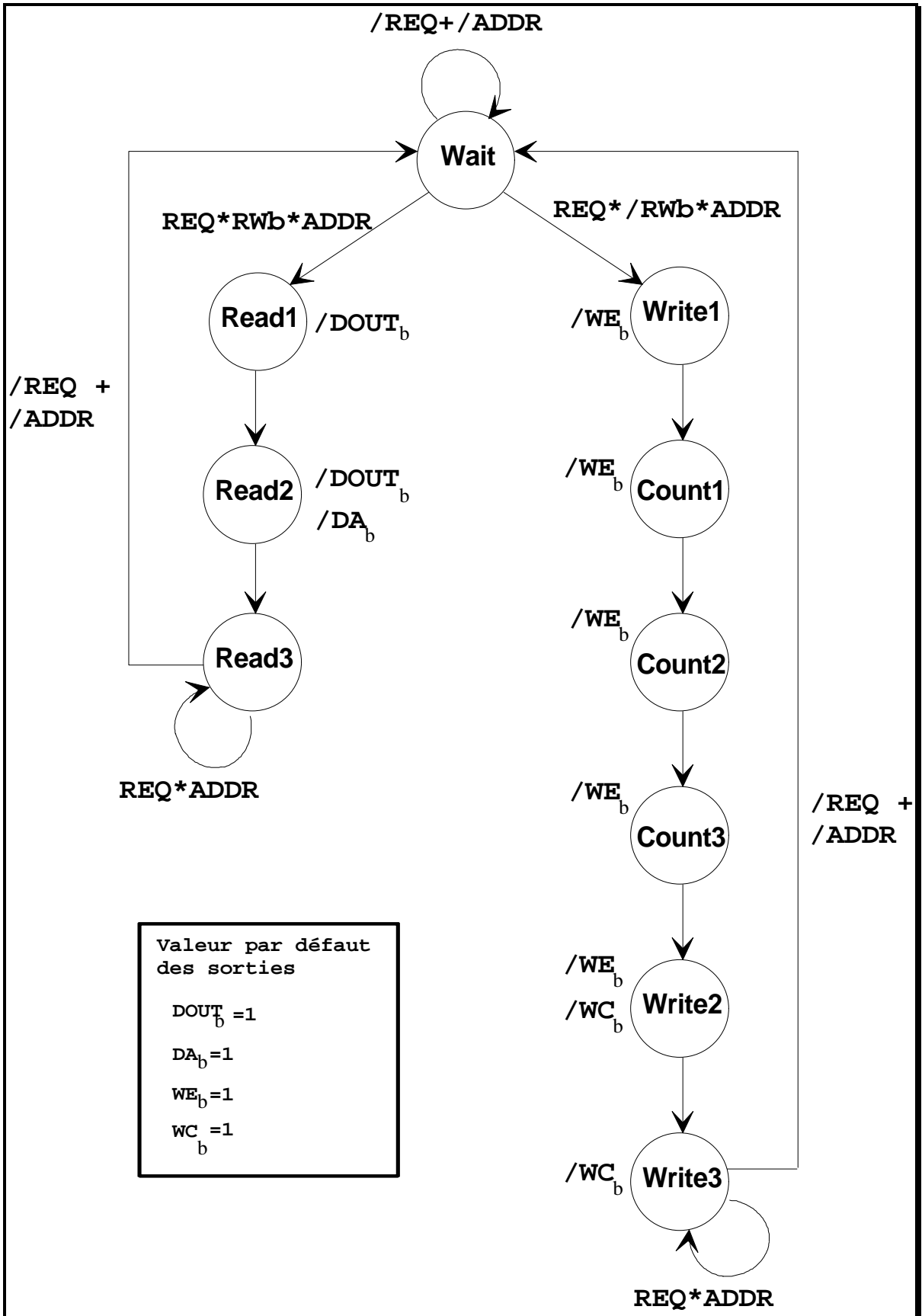
Une interface entre un calculateur et un banc mémoire répond à la machine d'état présentée à la page ci_après.

Le bilan des entrées sorties est le suivant :



Les signaux actifs bas sont repérés avec la lettre b à la fin.

Réaliser la synthèse en vhdl de cette machine.



(D) DIRECTIVES DE SYNTHÈSE.

(d-1) Imposer les broches utilisées.

Il est souvent nécessaire d'imposer le brochage lors d'une synthèse. L'attribut **pin_numbers** permet de donner les numéros de broche aux signaux d'entrées et sorties du composant. Cet attribut est attaché à l'entité.

syntaxe : *attribute pin_numbers of my_design:entity is "string"*
"string" = "sig1:1 " & "sig2:3 " & "sig3:5"

(d-2) Choix du composant.

L'attribut **part_name** permet d'imposer le composant à utiliser.

syntaxe : *attribute part_name of my_design:entity is "C371"*

(d-3) Exemple d'utilisation des attributs de synthèse.

```
entity seq_lum3 is port(
  RAZb      : in std_logic;
  DEPART    : in std_logic;
  ARRET     : in std_logic;
  FIN_COMPTAGE : in std_logic;
  INTER     : in std_logic_vector(6 downto 0);
  CLK       : in std_logic;
  LED_ROUGE : out std_logic;
  LOAD_C    : out std_logic;
  VALID_C   : out std_logic;
  CLOCK_C   : out std_logic;
  PROGRAMME : out std_logic_vector(2 downto 0));

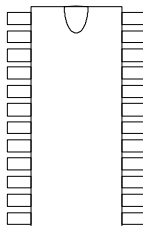
  attribute part_name of seq_lum3: entity is "C22V10";
  attribute pin_numbers of seq_lum3: entity is
    " clk:1 " &
    " depart:2 " &
    " fin_comptage:3 " &
    " razb:4 " &
    " inter(0):5 " &
    " inter(1):6 " &
    " inter(2):7 " &
    " inter(3):8 " &
    " inter(4):9 " &
    " inter(5):10 " &
    " inter(6):11 " &
    " clock_c:22 " &
    " load_c:21 " &
    " programme(1):20 " &
    " led_rouge:19 " &
    " programme(2):18 " &
    " programme(0):17 " &
    " arret:13 " &
    " valid_c:16 ";
end seq_lum3;
```

(d-4) Exercice.

Reprendre la description de la machine d'état de la gestion du banc mémoire en imposant le choix du Pal20R4 et le brochage ci-dessous :

Brochage retenu :

TYPE	NOM	
Horloge	CLK	1
Entrée	R/Wb	2
Entrée	REQ	3
Entrée	ADDR	4
		5
		6
		7
		8
		9
		10
		11
		12



	NOM	TYPE
24		
23		
22	WCb	Sortie
21	WEb	Sortie
20		Etat
19		Etat
18		Etat
17		Etat
16	DAb	Sortie
15	DOUTb	Sortie
14		
13		

Annexe 1 : bibliographie

[1] Manuels livrés avec le logiciel Warp 2

-> le manuel de référence pour le langage et des exemples.

-> le guide de l'utilisateur pour les tutoriaux.

[2] VHDL Initiation , application aux composants programmables.

Polycopié de formation Cypress.

[3] Circuits numériques et synthèse logique un outil le vhdl.

Collection Masson Technologies J.Weber et M.Meaudre

[4] VHDL For Programmable Logic. par Kevin Skahill

Edition Cypress.

[5] VHDL du langage à la modélisation.

Collection informatique, CNET ENST, R.Airiau, J-M Bergé,
V.Olive, J.Rouillard.

Ouvrage général sur le vhdl, n'est pas directement
orienté vers la synthèse de composant programmable.