

Le langage Vhdl

Pierre Leray

Patrice Quinton

Olivier Sentieys

Lucien Ungaro

Le langage VHDL

Modélisation de composants digitaux

MODÈLE DE COMPOSANT

```
-- usage de la bibliothèque standard

LIBRARY IEEE; USE IEEE.std_logic_1164.all

ENTITY NonEt IS -- broches d'entrée et de sortie

    PORT (x,y: IN std_logic; s: OUT std_logic);

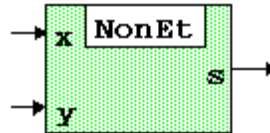
END;

ARCHITECTURE comportement OF NonEt IS

    -- description de la fonction réalisée

    s <= not(x and y);

END comportement;
```



3

Un modèle de composant comporte essentiellement deux parties :

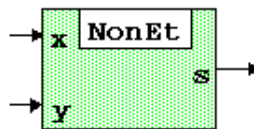
- une interface d'utilisation (ENTITY) : on y trouve, dans la rubrique PORT, la définition des broches d'entrée et de sortie du composant,
- une description de la fonction réalisée (ARCHITECTURE).

L'exemple montre la modélisation d'une porte "non-et". Ce composant possède deux entrées x et y et une sortie s. Ces broches sont de type std_logic qui représente les valeurs binaires usuelles.

Ce type est défini dans la bibliothèque standard IEEE.

Description comportementale

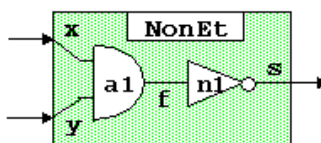
$s \leq \text{not}(x \text{ and } y);$



Description structurelle

a1: et PORT MAP (x,y,f);

n1: inv PORT MAP (f,s);



4

On distingue 2 grandes classes de descriptions :

- **les descriptions comportementales** : Elles spécifient le comportement au moyen de formules ou de programmes d'actions.

L'exemple montre la description comportementale d'une porte NonEt : Cette formule signifie que la sortie s vaut en permanence l'inverse de la fonction et appliquée aux entrées x et y . Une telle description n'utilise aucun composant interne : on considère que le composant est atomique, entièrement défini par sa spécification logique.

- **les descriptions structurelles** : Elles énoncent une interconnexion de composants. Une description structurelle est la transcription directe d'un schéma.

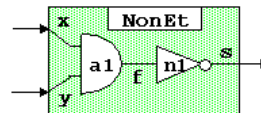
L'exemple montre une description structurelle d'une porte NonEt : une porte NonEt peut être réalisée en connectant un composant a1 de type et sur la broche x , la broche y et un fil interne f et un composant n1 de type inv entre le fil interne f et la broche s . Le comportement résulte du comportement des composants internes et du câblage réalisé, selon les règles usuelles de causalité physique.

Description structurelle

```
ARCHITECTURE structure OF NonEt IS
-- déclarations de modèles de composants
COMPONENT et
  PORT(a,b:IN std_logic; s:OUT std_logic);
END COMPONENT;

COMPONENT inv
  PORT(e:IN std_logic; s:OUT std_logic);
END COMPONENT;

-- déclarations de signaux internes (fils)
SIGNAL f: std_logic;
BEGIN -- instanciation et câblage des composants
(schéma)
  a1: et PORT MAP (x,y,f)
  n1: inv PORT MAP (f,s);
END structure;
```



5

Une description structurelle se compose de trois rubriques :

- Des déclarations de modèles des composants utilisés : c'est un rappel de ce que l'on trouve dans les entités qui définissent les composants.
- Des déclarations de signaux internes, destinés à interconnecter les composants.
- Les composants, avec pour chacun d'eux le nom de son modèle et son câblage (PORT MAP) qui indique les fils internes ou les broches qui sont connectés sur les broches de ce composant.

L'exemple montre un modèle structurel complet d'une porte NonEt. Le modèle nécessite la déclaration d'un signal interne f, analogue d'un fil pour connecter la sortie de la porte et à l'entrée de l'inverseur.

Affectation concurrente de signal

forme 1: $s \leq e;$

forme 2: $s \leq e1 \text{ WHEN } c1 \text{ ELSE } e2 \text{ WHEN } c2 \text{ ELSE } \dots ;$

Ces affectations sont faites en parallèle.

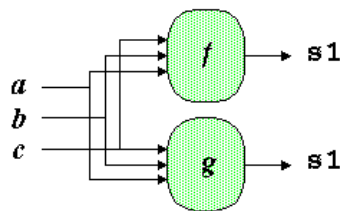
BEGIN

$s1 \leq f(a,b,c...);$

$s2 \leq g(a,b,c...);$

END

<-- formulations équivalentes -->



BEGIN

$s2 \leq g(a,b,c...);$

$s1 \leq f(a,b,c...);$

END

6

Les descriptions comportementales utilisées précédemment sont des affectations concurrentes de signal. Il en existe 2 formes :

- La première signifie l'affectation continue du signal s à la valeur de l'expression e . Plus précisément, l'affectation a lieu chaque fois qu'un signal mentionné dans l'expression change de valeur.
- La seconde, signifie l'affectation du signal s à la valeur de l'expression $e1$ lorsque la condition $c1$ est vraie sinon à la valeur de $e2$ lorsque la condition $c2$ est vraie...

Il peut y avoir plusieurs affectations concurrentes dans un modèle. Leur ordre de présentation est sans importance, car elles représentent des affectations en parallèle (potentiellement simultanées).

Types logique standard

type std_logic

valeurs : '0', '1', 'Z', 'X'

-- exemple de déclarations de broches et de signaux

```
PORT(... e:IN std_logic;... s:OUT std_logic);  
SIGNAL s1, s2 : std_logic;
```

type std_logic_vector

-- exemple de déclarations de broches et de signaux

```
PORT(... A:IN std_logic_vector(7 DOWNT0 0);...);  
SIGNAL reg : std_logic_vector(3 DOWNT0 0);
```

```
accès    reg<=('0','0','0','1');  
          ou encore reg<="0001"; reg(2)<='0'; reg(3)<=A(6);
```

7

Les valeurs logiques portées par les signaux (broches ou fils internes) sont définies par les types logiques standard de la bibliothèque IEEE :

std_logic : bit simple, qui peut prendre 4 valeurs notées '0', '1', 'Z' et 'X' qui représentent respectivement le bit 0, le bit 1, la déconnexion (signal non forcé, ou encore laissé en "haute impédance" selon la terminologie électronique), et une valeur indéterminée.

std_logic_vector : vecteur de bits. Chaque élément est de type std_logic. Les bornes d'indice d'un vecteur sont indiquées à sa déclaration, sous la forme (7 DOWNT0 0) ou encore (0 TO 7) pour un vecteur de 8 bits indicé de 0 à 7. La différence entre les deux formes ne concerne que les notations littérales de valeurs, la première plaçant le bit 7 à gauche et la deuxième à droite.

L'accès à l'élément d'indice i d'un tableau tab se note : tab(i)

On peut dénoter une valeur de tableau en énumérant ses composante :
(v1,v2,v3,v4)

Une notation littérale de la forme "0001" est acceptée pour le type std_logic_vector.

Quelques opérateurs

opérations sur bits et vecteur de bits : types `std_logic` et `std_logic_vector`
opérations logiques : `and`, `or`, `not`, `xor`
concaténation : `&`

exemple : `"1101"&"011"` vaut `"1101011"`

opérations booléennes : type `boolean`
valeurs : `false`, `true`
opérations logiques : `and`, `or`, `not`, `xor`

opérations arithmétiques : type `integer`
opérations arithmétiques : `+`, `-`, `*`, `/`

comparaisons : opérandes `std_logic`, `integer`...
résultat `boolean`

comparaisons : `=`, `/=`, `<`, `<=`, `>`, `>=`

8

Les types `std_logic` et `std_logic_vector` disposent des opérations usuelles sur bits : `and` (et), `or` (ou), `not` (non), `xor` (ou exclusif).

Pour des vecteurs de bits elles signifient les opérations bit à bit.

L'opération de concaténation, notée `&`, permet de construire des vecteurs à partir de vecteurs plus petits ou de simples bits.

Pour exprimer des conditions logiques, le langage possède le type `boolean` (booléen).

Ses valeurs sont `false` (faux) et `true` (vrai). Le type booléen dispose des opérations logiques usuelles : `and`, `or`, `not`, `xor`.

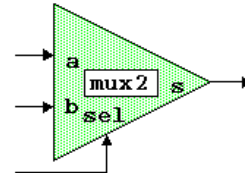
Les nombres entiers sont offerts par le type `integer`, doté des opérations arithmétiques usuelles.

Les opérations de comparaison, notées `=`, `/=`, `<`, `<=`, `>`, `>=`, rendent un résultat booléen. Elles sont définies pour tous les types scalaires (`std_logic`, entiers, caractères, flottant...).

Exemple 1a: Multiplexeur 2 voies

Définition d'entité

```
ENTITY mux2 IS  
  PORT (a,b,sel: IN std_logic; s: OUT std_logic);  
END;
```



Description comportementale

```
ARCHITECTURE comportement OF mux2 IS  
  BEGIN  
    s <= a WHEN sel='0' ELSE b WHEN sel='1' ELSE 'X';  
  END comportement;
```

9

Ce composant est un multiplexeur de 2 voies de 1 bit. Pour sel=0, la sortie s vaut l'entrée a et pour sel=1, elle vaut l'entrée b.

La description comportementale indique que, en permanence, s vaut a si sel=0, elle vaut b si sel=1 et elle est indéterminée (notation 'X') si sel est indéterminée.

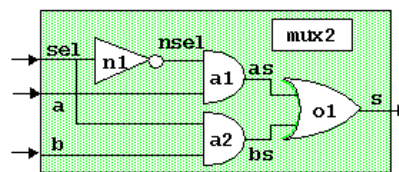
Exemple 1b: Multiplexeur 2 voies

Description structurelle

```
ARCHITECTURE structure OF mux2 IS
-- déclarations de modèles de composants
COMPONENT inv PORT(e:IN std_logic; s:OUT std_logic);
END COMPONENT;
COMPONENT et PORT(e1,e2:IN std_logic; s:OUT std_logic);
END COMPONENT;
COMPONENT ou PORT(e1,e2:IN std_logic; s:OUT std_logic);
END COMPONENT;

-- déclarations de signaux internes (fils)
SIGNAL nsel,as,bs: std_logic;

-- instanciation et câblage des composants (schéma)
BEGIN
  n1: inv PORT MAP (sel,nsel);
  a1: et PORT MAP (nsel,a);
  a2: et PORT MAP (sel,b);
  o1: ou PORT MAP (as,bs);
END structure;
```



10

Cette description structurelle du multiplexeur utilise un composant `inv`, deux portes `et` et une porte `ou` connectées comme le montre le schéma.

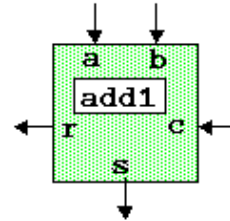
Exemple 2a: Additionneur 1 bit

Définition d'entité

```
ENTITY add1 IS  
  PORT(a,b,c: IN std_logic; r,s: OUT std_logic);  
END;
```

Description comportementale

```
ARCHITECTURE comportement OF add1 IS  
BEGIN  
  r <= (a and b) or (a and c) or (b and c);  
  s <= a xor b xor c ;  
END comportement;
```



11

Cet exemple décrit un additionneur pour données de 1 bit : a, b et c sont les entrées de données et de report. r et s sont les sorties de report et de somme. La description comportementale utilise les fonction logiques and et or pour exprimer le report , et xor (ou-exclusif) pour exprimer la somme.

Exemple 2b: Additionneur 1 bit

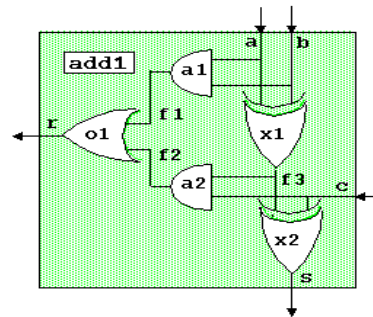
Description structurelle

```
ARCHITECTURE structure OF add1 IS  
  COMPONENT et PORT(e1,e2:IN std_logic; s:OUT std_logic);  
  END COMPONENT;  
  COMPONENT ou PORT(e1,e2:IN std_logic;s:OUT std_logic);  
  END COMPONENT;  
  COMPONENT oux PORT(e:IN std_logic; s:OUT std_logic);  
  END COMPONENT;
```

```
SIGNAL f1,f2,f3: 3std_logic;
```

```
BEGIN
```

```
  a1: et PORT MAP (a,b,f1);  
  a2: et PORT MAP (f3,c,f2);  
  o1: ou PORT MAP (f1,f2,r);  
  x1: oux PORT MAP (a,b,f3);  
  x2: oux PORT MAP (f3,c,s);  
END structure;
```



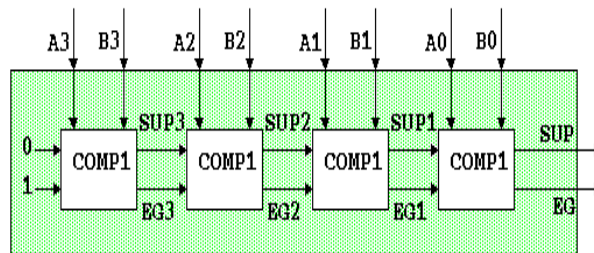
12

Cette description structurelle utilise des portes et, ou et oux (ou-exclusif).

Exercice 1

Comparateur numérique binaire

SUP : (A>B), EG : (A=B)



13

Exercice 1

On se propose de réaliser un comparateur pour des nombres entiers positifs représentés en binaire sur n bits. Ce circuit reçoit en entrée deux nombres A et B et délivre 2 sorties : SUP qui indique si $A > B$ et EG qui indique si $A = B$.

On décide d'utiliser un algorithme de comparaison bit à bit à partir des poids forts. L'étage de comparaison, appelé COMP1, reçoit les bits de rang i , A_i et B_i , ainsi que le résultat de la comparaison des nombres représentés par les bits de rangs supérieurs à i , SUP_{i+1} et EG_{i+1} . Il délivre le résultat de la comparaison jusqu'aux bits de rang i , selon la règle suivante :

si $SUP_{i+1}=1$, $SUP_i=1$, sinon, si $EG_{i+1}=1$, SUP_i est déterminé par la comparaison de A_i et B_i , sinon $SUP_i=0$.

Rédiger :

- la définition d'entité du comparateur COMP1,
- une description comportementale du comparateur COMP1,
- la définition d'entité d'un comparateur COMP4 pour données de 4 bits,
- une description structurelle de COMP4 qui utilise des instances de COMP1 interconnectées comme le montre le schéma.

Processus et instructions séquentielles

par un processus

```
ARCHITECTURE comportement
  OF NonEt IS
BEGIN
  Evaluation: PROCESS(x,y)
  s <= not(x and y);
END PROCESS;
END comportement;
```

par affectation concurrente

```
ARCHITECTURE comportement
  OF NonEt IS
BEGIN
  s <= not(x and y);
END comportement;
```

Forme générale

```
NomDeProcessus: PROCESS(signaux déclencheurs)
  déclarations
BEGIN
  instructions séquentielles
END PROCESS;
```

14

Un composant peut être décrit à l'aide d'un ou plusieurs processus. Chaque processus est défini par un programme introduit par le mot `PROCESS`. Les instructions d'un programme de processus sont similaires à celles d'un langage de programmation impératif classique. Elles sont exécutées séquentiellement.

Un processus possède généralement d'une liste de signaux déclencheurs. Son programme est exécuté en début de simulation et chaque fois qu'un signal déclencheur change de valeur.

L'exemple proposé montre la description d'une porte NonEt par un processus. Les signaux déclencheurs sont les entrées `x` et `y` afin de réévaluer la sortie `s` chaque fois qu'une entrée change.

Cette forme est plus lourde que l'affectation concurrente de signal, mais elle est plus générale.

L'affectation concurrente est un cas particulier simplifié de processus, réduit à l'affectation du signal et déclenché chaque fois qu'un des signaux de l'expression en partie gauche change. Cette forme est bien adaptée à la spécification de composants combinatoires et elle est préférable dans ce cas.

Quelques instructions séquentielles

Conditionnelle booléenne

```
IF cond1 THEN I1
ELSIF cond2 THEN I2
...
ELSE In
END IF;
```

Conditionnelle par cas

```
CASE expr IS
WHEN cas1 => I1
WHEN cas2 => I2
...
WHEN OTHERS => In
END CASE;
```

Boucle tant que

```
WHILE condition LOOP instructions END LOOP;
```

Boucle pour

```
FOR i IN a TO b LOOP instructions END LOOP
FOR i IN a DOWNTO b LOOP instructions END LOOP
```

15

Le langage possède les structures de contrôle usuelles :

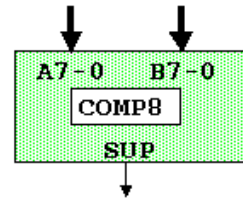
- Conditionnelle booléenne (IF) : exécute la première séquence d'instructions I_k dont la condition booléenne associée $cond_k$ est vraie. La branche ELSE, optionnelle, est exécutée si aucune condition n'est vraie.
- Conditionnelle par cas (CASE) : $expr$ est une expression de type simple ou tableau, cas_1, cas_2, \dots sont des valeurs ou listes de valeurs (séparées par le symbole |). Cette construction exécute la première séquence d'instructions I_k dont la liste de valeurs cas_k contient la valeur de $expr$. La branche OTHERS, optionnelle, est exécutée si aucun cas n'est satisfait.
- Boucle "tant que" (WHILE) : exécute les instructions tant que la condition est vraie.
- Boucle "pour" (FOR) : exécute les instructions pour i variant de a à b .

Exemple 3: Comparateur de supériorité

```
ENTITY COMP8 IS
  PORT(A,B:IN std_logic_vector(7 DOWNT0 0); SUP:OUT std_logic)
END;
```

```
ARCHITECTURE comportement OF COMP8 IS
BEGIN
```

```
  evalSUP : PROCESS(A,B)
    VARIABLE VA,VB : integer;
  BEGIN
    VA:=0; VB:=0;
    FOR i IN 7 DOWNT0 0 LOOP -- interprétation numérique de A et B
      IF A(i)='1' THEN VA:=2*VA+1; ELSE VA:=2*VA; END IF;
      IF B(i)='1' THEN VB:=2*VB+1; ELSE VB:=2*VB; END IF;
    END LOOP;
    IF VA>VB THEN SUP<='1'; ELSE SUP <='0'; END IF;
  END PROCESS;
END comportement;
```



16

Cet exemple est un comparateur pour nombres entiers positifs représentés en binaire sur 8 bits. La sortie SUP indique si $A > B$.

Le composant est modélisé par un processus qui interprète les entrées A et B en binaire pour calculer deux nombres entiers VA et VB. Le résultat est alors celui de la comparaison arithmétique de VA et VB. Remarquer le rôle abstrait que joue le type integer : il sert juste d'intermédiaire pour spécifier la fonction réalisée. Les signaux physiques ne sont jamais de type integer (ni flottant, ni booléen, ni caractère...), mais de type std_logic.

Pour VA et VB on a utilisé des variables (VARIABLE) et non pas des signaux. On aurait pu utiliser des signaux, mais les variables, d'usage plus restrictif que les signaux, sont justement destinées à conserver des résultats intermédiaires pour calculer des fonctions par des moyens algorithmiques, ce qui est bien le cas ici. L'affectation de variable se note :=.

Composants séquentiels synchrones

ARCHITECTURE comportement OF xxx IS

signal etat : ...;

BEGIN

changementEtat: PROCESS(ck)

BEGIN

IF ck='1' THEN

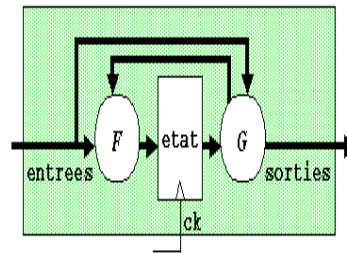
etat <= F(entrees,etat);

END IF;

END PROCESS;

sorties <= G(entrees,etat);

END comportement;



17

Un composant séquentiel synchrone possède un état interne qui change uniquement au front montant de l'horloge ck. Pour modéliser un tel circuit, le changement d'état peut être représenté par un processus changementEtat déclenché par l'horloge ck. Le programme de ce processus doit tester ck='1' pour ne changer l'état que lors du front montant (passage de 0 à 1 de ck).

Les sorties dépendent de l'état mais peuvent aussi dépendre des entrées de façon continue. Elles doivent donc être modélisées par des processus séparés ou des affectations concurrentes.

L'exemple illustre un composant synchrone quelconque, défini par une fonction de transition F et une fonction de sortie G.

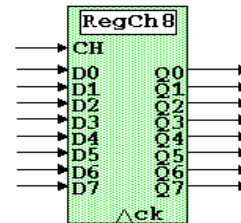
Exemple 4: Registre à chargement commandé

```
ENTITY RegCh8 IS
  PORT (ck,CH: IN std_logic;
        D: IN std_logic_vector(7 DOWNTO 0);
        Q: OUT std_logic_vector(7 DOWNTO 0));
END;
```

```
ARCHITECTURE comportement OF RegCh8 IS
BEGIN
  changementEtat: PROCESS(ck)
  BEGIN
    IF ck='1' THEN IF CH='1' THEN Q <= D; END IF;
    END IF;
  END PROCESS;

  -- les sorties sont directement l'état

END comportement;
```



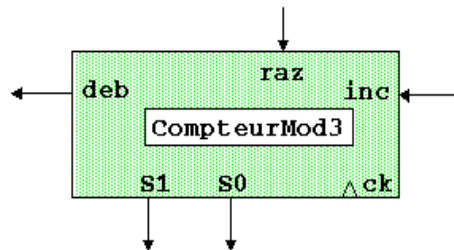
18

Ce premier exemple de composant séquentiel synchrone est un registre 8 bits, doté d'une commande de chargement CH. Au front montant de l'horloge, si CH=1 l'état Q prend la valeur de l'entrée D, si CH=0, l'état reste inchangé.

Les sorties Q7-0 sont directement l'état. Il n'y a donc pas besoin d'exprimer les sorties. Pour la même raison, l'état est défini en tant que broches dans cet exemple, ce qui dispense de le déclarer en tant que signal interne.

Exemple 5a: Compteur modulo 3

```
ENTITY CompteurMod3 IS
  PORT (ck,raz,inc: IN std_logic;
        deb : OUT std_logic;
        S: OUT std_logic_vector(1 DOWNTO 0));
END;
```



19

Cet exemple un peu plus général de composant synchrone est un compteur modulo 3.

Il est mis à 0 par la commande raz et il s'incrémente modulo 3 (0, 1, 2, 0, 1, 2, 0...) lorsque la commande inc est active.

Il délivre sur S1 et S0 la représentation binaire de l'état (00 pour 0, 01 pour 1 et 10 pour 2). La sortiedeb (débordement) est à 1 lorsque le compteur est incrémenté dans l'état 2 (cela permet par exemple de mettre plusieurs compteurs modulo 3 en cascade pour réaliser un compteur de plusieurs chiffres en base 3).

Exemple 5b: Compateur de sup riorit 

```
ARCHITECTURE comportement OF CompteurMod3 IS
  SIGNAL compte : integer;
BEGIN
  changementEtat: PROCESS(ck)
  BEGIN
    IF ck='1' THEN
      IF raz='1' THEN compte <= 0;
      ELSIF inc='1' THEN
        CASE compte IS
          WHEN 0 => compte <= 1;
          WHEN 1 => compte <= 2;
          WHEN 2 => compte <= 0;
        END CASE;
      END IF;
    END IF;
  END PROCESS;

  S <= "00" WHEN compte=0 ELSE "01" WHEN compte=1 ELSE "10" WHEN compte=2
    ELSE "XX"
END comportement;
```

20

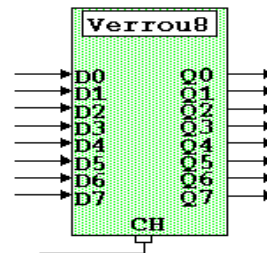
Le type choisi pour l' tat compte est ici un entier (integer). C'est un bon choix car le comptage est bien une op ration du domaine des entiers, et non des vecteur de bits. Ceci est une r gle g n rale : choisir pour l' tat le type abstrait en accord avec la logique r alis e. On peut se le permettre, car l' tat n'est pas directement visible dans un composant s quentiel, seules les broches de sortie le sont, en g n ral d finies par une fonction de sortie.

Les sorties S et deb sont d finies par des affectations concurrentes.

Changement d'état asynchrone

```
ENTITY Verrou8 IS
PORT (CH: IN std_logic;
      D: IN std_logic_vector(7 DOWNTO 0);
      Q: OUT std_logic_vector(7 DOWNTO 0));
END;
```

```
ARCHITECTURE comportement OF Verrou8 IS
BEGIN
  changementEtat: PROCESS(CH,D)
  BEGIN
    IF CH='1' THEN Q <= D;
    END IF;
  END PROCESS;
END comportement;
```



21

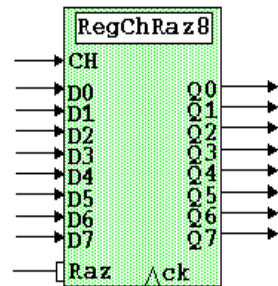
Un changement d'état asynchrone est un changement d'état qui a lieu tant qu'une condition est vérifiée, et non pas sur un front d'horloge comme pour un changement synchrone.

L'exemple montre un verrou 8 bits (latch), qui enregistre les données présentes sur D tant que la commande de chargement CH vaut 1. Un tel composant se modélise par un processus sensible non seulement aux signaux qui provoquent le changement d'état, CH ici, mais également aux signaux dont dépendent la valeur de l'état, D ici.

Changement d'état mixte

```
ENTITY RegChRaz8 IS
  PORT    (ck,Raz,CH: IN std_logic;
           D: IN std_logic_vector(7 DOWNTO 0);
           Q: OUT std_logic_vector(7 DOWNTO 0));
END;
```

```
ARCHITECTURE comportement OF RegChRaz8 IS
BEGIN
  changementEtat: PROCESS(ck,Raz)
  BEGIN
    IF Raz='1' THEN Q <= "00000000";
    ELSIF ck'EVENT and ck='1' THEN
      IF CH='1' THEN Q <= D; END IF;
    END IF;
  END PROCESS;
END comportement;
```



22

Certains composants utilisent les deux modes de changement d'état, synchrone (sur front d'horloge) pour certaines commandes et asynchrone (sur niveau de certains signaux) pour d'autres commandes.

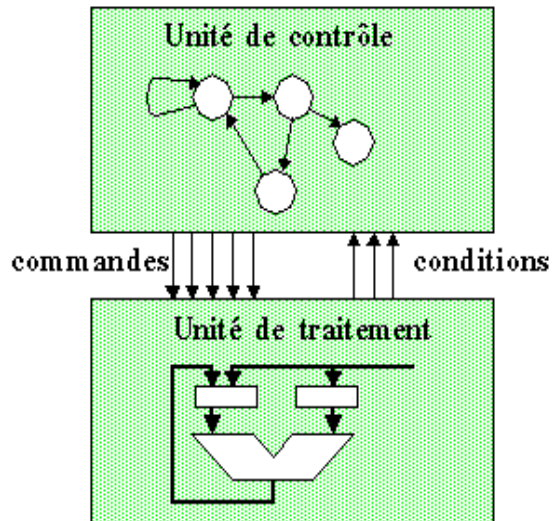
On peut prendre pour exemple un registre 8 bits doté d'une commande de chargement synchrone, CH, et d'une commande de mise à 0 asynchrone, Raz, qui provoque la mise à 0 tant que Raz=1, indépendamment de l'horloge.

Cet exemple nécessite deux remarques :

- Par nature, les changements d'état asynchrones sont prioritaires sur les changements d'état synchrones : ceci se traduit par un test préliminaire des commandes asynchrones, IF Raz='1' dans cet exemple.
- Le processus de changement d'état n'est plus déclenché seulement par l'horloge, mais aussi par les commandes asynchrones : la détection du front montant nécessite donc une condition supplémentaire notée ck'EVENT qui indique que l'horloge vient juste de changer de valeur.

Plus généralement, s'EVENT est une expression booléenne qui indique que le signal s change de valeur.

Automates de contrôle 1



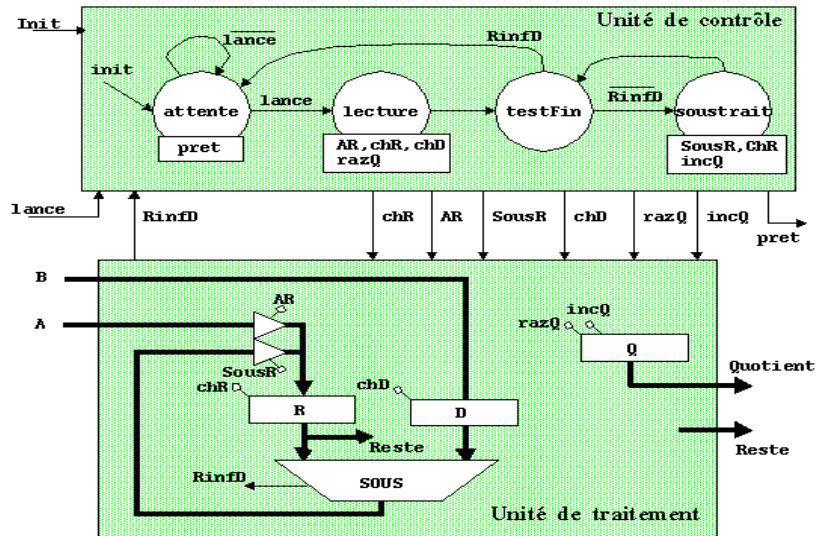
23

On décompose généralement la conception d'un composant complexe en deux parties :

- une unité de traitement, qui regroupe des mémoires, des opérateurs et des chemins de données,
- une unité de contrôle qui séquence les opérations effectuées par l'unité de traitement en générant des commandes et en testant des conditions.

Dans les cas usuels, unité de traitement et unité de contrôle sont des composants synchrones fonctionnant sur le même horloge. L'unité de contrôle est définie au moyen d'un diagramme d'états dont les flèches sont validées par les conditions. Les commandes sont généralement fonction uniquement de l'état.

Automates de contrôle 2



24

Comme exemple d'une telle décomposition, on peut considérer une machine qui effectue la division entière d'un nombre A par un nombre B. Un calcul est démarré par un signal lance maintenu à 1 pendant un cycle. La machine effectue la division par soustractions successives.

R est initialisé à A et B est soustrait de R tant que R est supérieur ou égal à B. Le nombre de soustraction est comptabilisé dans un registre Q pour fournir le quotient. En fin de calcul la machine délivre les résultats sur Reste et Quotient, accompagné d'un signal pret=1 maintenu jusqu'au prochain lancement de calcul.

Automates de contrôle 3

```
ENTITY DiventUC IS
  PORT (ck,Init,lance,RinfD: IN std_logic;
        chR,AR,SousR,chD,razQ,incQ,pret: OUT std_logic);
END;
ARCHITECTURE comportement OF DiventUC IS
  TYPE valeurEtat IS (attente,lecture,testFin,soustrait);
  SIGNAL etat : valeurEtat;
BEGIN
  changementEtat: PROCESS(ck) BEGIN
    IF ck='1' THEN
      IF Init='1' THEN etat<=attente; ELSE
        CASE etat IS
          WHEN attente => IF lance='1' THEN etat<=lecture; END IF;
          WHEN lecture => etat<=testFin;
          WHEN testFin => IF RinfD='1' THEN etat<=attente;
            ELSE etat<=soustrait; END IF;
          WHEN soustrait => etat<=testFin;
        END CASE;
      END IF;
    END IF;
  END PROCESS;
  ...
```

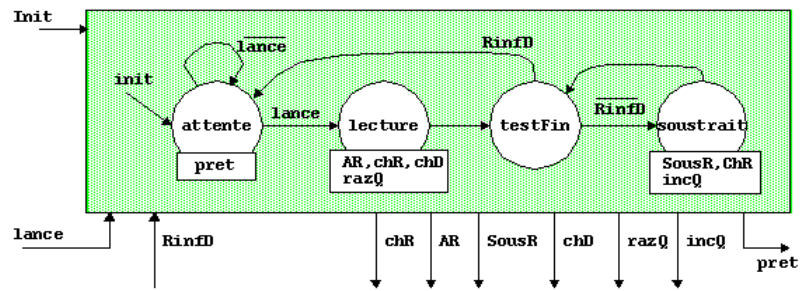
25

Pour représenter les valeurs de l'état de l'unité de contrôle, on utilise un type énuméré. Ici on a déclaré le type valeurEtat, composé des valeurs possibles pour l'état, chaque valeur étant définie par un identificateur : attente, lecture, testFin, soustrait.

L'évolution de l'état, représenté par le signal etat, est réalisé par un processus changementEtat, qui à chaque top d'horloge change l'état en fonction de sa valeur présente et des conditions.

Une bonne méthode consiste à utiliser un CASE selon la valeur présente de l'état, ainsi le texte est plus facile à lire car il présente la même structure que le diagramme des états.

Automates de contrôle 4



```

...
pret <= '1' WHEN etat=attente ELSE '0';
AR <= '1' WHEN etat=lecture ELSE '0';
chR <= '1' WHEN (etat=lecture) or (etat=soustrait)
      ELSE '0';
chD <= '1' WHEN etat=lecture ELSE '0';
razQ <= '1' WHEN etat=lecture ELSE '0';
SousR <= '1' WHEN etat=soustrait ELSE '0';
incQ <= '1' WHEN etat=soustrait ELSE '0';
  
```

END comportement;

26

La génération des commandes est faite au moyen d'assignations concurrentes.