

Concurrent et Séquentiel

- ▶ ENTITY comb IS

▶ PORT (e1, e2 : IN BIT ;

▶ s1, s2 : OUT BIT ) ;

▶ END comb ;

▶ ARCHITECTURE combinatoire OF

comb IS

▶ BEGIN

▶ S1<= e1 AND e2 ;

▶ S2 <= e1 OR e2;

▶ END combinatoire ;
- ENTITY comb IS

■ PORT (e1, e2 : IN BIT ;

■ s1, s2 : OUT BIT ) ;

■ END comb ;

■ ARCHITECTURE combinatoire

OF comb IS

■ BEGIN

■ process

□ S1<= e1 AND e2 ;

□ S2 <= e1 OR e2;

■ End process;

■ END combinatoire ;

▶ 1

Processus et instructions séquentielles

par un processus	par affectation concurrente
ARCHITECTURE comportement OF NonEt IS BEGIN Evaluation: PROCESS(x,y) s <= not(x and y); END PROCESS; END comportement;	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;	

Composant séquentiel synchrone

- ▶ état interne change uniquement à un front d'horloge ck

(montant ou descendant)

▶ modéliser par processus

▶ changement état => déclencheur

▶ 3

VHDL séquentiel

- ▶ Un composant peut être décrit à l'aide d'un ou plusieurs

processus.

▶ Chaque processus est défini par une liste de constructions

introduites par le mot PROCESS

▶ Les constructions d'une modélisation d'un processus se

"déroulent" séquentiellement.

▶ 4

Séquentiel et Combinatoire

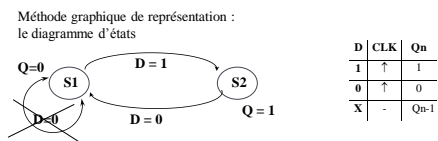
```
ENTITY comb_seq IS
PORT (e1, e2 : IN BIT ;
      s : OUT BIT ) ;
END comb_seq ;
ARCHITECTURE combinatoire OF comb_seq IS
BEGIN
p1 : PROCESS (e1,e2)
BEGIN
IF (e1 = '1') THEN
s <= e2 ;
ELSE
s <= '0' ;
ENDIF ;
END PROCESS ;
END combinatoire ;
```

un fonctionnement combinatoire  
peut-être spécifié dans un PROCESS  
aussi

De quelle fonction s'agit-il ?

▶ 5

Modélisation séquentielle synchrone



- \* En synchrone, toutes les transitions ont lieu aux fronts actifs de l'horloge (signal

non-représenté sur le diagramme).

\* Conditions de maintien non plus => tant que la condition de transition n'est pas satisfaite

pas de changement d'état.

Comment le transcrire en VHDL ?

▶ 6

## Composants séquentiels synchrones

```

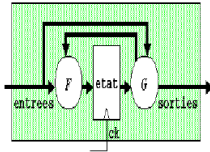
-- ARCHITECTURE comportement OF xxx IS
  signal etat : ...;
BEGIN

```

```

  changementEtat: PROCESS(ck)
  BEGIN
    IF ck='1' THEN
      etat <= F(entrees,etat);
    END IF;
  END PROCESS;

```



```

  sorties <= G(entrees,etat);

```

```

END comportement;

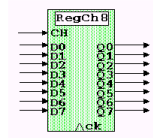
```

## 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;

```

## Exemple 3: Comparateur de supériorité

```

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

```

```

ARCHITECTURE comportement OF COMP8 IS
  BEGIN

```

```

    evalSUP: PROCESS(A,B)
    VARIABLE VA,VB: integer;
  BEGIN

```

```

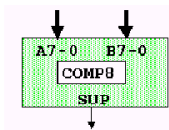
    VA:=0; VB:=0;
    FOR i IN 7 DOWNTO 0 LOOP -- interprétation numérique de A et B
      IF A(i)='1' THEN VA:=2*VA+1; ELSE VA:=2*VA; END IF;
      IF B(i)='1' THEN VB:=2*VB+1; ELSE VB:=2*VB; END IF;
    END LOOP;
    IF VA>VB THEN SUP<='1'; ELSE SUP <='0'; END IF;
  END PROCESS;

```

```

END comportement;

```



## Remarques diverses

On parle de moins en moins de combinatoire et séquentiel, mais de commandes concurrentes ou séquentielles.

\* Si plusieurs PROCESS comprennent le même signal dans leur liste de sensibilité, ils sont tous déclenchés au même moment.

\* Le nom d'un PROCESS est optionnel, et sert à en avoir plusieurs dans la même ARCHITECTURE

## Conclusions combinatoire séquentiel

Les constructions autorisées hors PROCESS

- Affectations simples  
s <= '1'; a <= b;
- Affectations conditionnelles  
A <= '1' WHEN (c = b) ELSE '0'

Si nécessaire, ne pas hésiter à utiliser un PROCESS pour disposer de constructions plus élaborées

```

CASE ...
  WHEN
END CASE
IF THEN
ELSE
ENDIF

```

## 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;

```

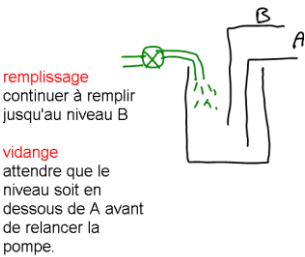
### Boucle pour

```

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

```

Nécessité du séquentiel



Quel circuit Combinatoire pourrait faire ceci ?

18

Nécessité du séquentiel

- Feux de circulation
  - Séquence dépendante du temps
  - Et/Ou avec d'autres entrées externes (bouton piéton)

19

Constructions plus élaborées

```
CASE expression IS
  WHEN valeur1 => commande1 ;
  ....
  ....
  WHEN valeur2 => commande1 ;
  ....
  OTHERS => commande1 ;
END CASE ;
```

Exemple :

```
CASE bit_vec IS
  WHEN "00" => s <= '1' ;
  WHEN "01" => s <= '0' ;
  OTHERS => s <= '0' ;
END CASE ;
```

21

Construction loop

Exemple 1 :

```
FOR i IN 1 TO 10 LOOP
  i_carre ( i ) := i * i
END LOOP ;
```

-- i\_carre est une variable de type tableau

- Il n'est pas nécessaire de déclarer i
- la valeur de i ne peut être changée à l'intérieur de LOOP

Exemple 2 :

```
PROCESS (i)
BEGIN
  x <= i + 1 ;
  FOR i IN 1 to a/2 LOOP
    q ( i ) := a ;
  END LOOP ;
END PROCESS ;
```

-- x est un signal

- la visibilité de i est très locale
- le premier i n'a rien à voir avec le deuxième

22

Construction loop

Exemple 1 :

```
WHILE ( i < 10 ) LOOP
  i_carre ( i ) := i * i
  i := i + 1 ;
END LOOP ;
```

-- i\_carre est une variable tableau de type entier

- La condition est vérifiée à chaque itération
- Dans ce cas, i peut-être modifié

23

NEXT dans LOOP

```
PROCESS (A, B)
  CONSTANT max_limit : INTEGER := 255 ;
BEGIN
  FOR i IN 0 TO max_limit LOOP
    IF (done (i) = TRUE) THEN
      NEXT ;
    ELSE
      done (i) := TRUE ;
    ENDIF ;
    q (i) <= a (i) AND b (i) ;
  END LOOP ;
END PROCESS ;
```

Pour donner une valeur initiale au contenant, meilleur endroit à la déclaration

Si la condition est vraie, i sera incrémenté, et le traitement reprendra à partir de LOOP

24

EXIT dans LOOP

```
PROCESS (a)
  variable int_a : integer ;
  BEGIN
    int_a := a ;
    FOR i IN 0 TO max_limit LOOP
      IF (int_a <= 0) THEN -- plus petit ou égal
        EXIT ;
      ELSE
        int_a := int_a - 1 ;
        q ( i ) <= 3.1416 / REAL ( a * i ) ;
      ENDIF;
    END LOOP ;
    y <= q ;
  END PROCESS ;
```

*Exit de la boucle si par exemple une valeur dépasse une limite*

*Le fonctionnement reprend après END LOOP*

*Conversion de type avec REAL*

*Dans le cas de plusieurs LOOP imbriquées, EXIT sort du LOOP concernée*

WAIT

```
WAIT ON signal -- attend le changement de niveau du signal

WAIT UNTIL expression_booleenne -- attend jusqu'à ce que expression soit vraie

WAIT FOR expression_temporelle -- attend n unités de temps

PROCESS
  BEGIN
    WAIT ON a ;
    WAIT ON b ;
  END PROCESS ;
```

*le fonctionnement s'arrêtera ici jusqu'au changement du niveau du signal a*

WAIT

```
WAIT UNTIL (( x * 10 ) < 100 ) ;

PROCESS
  BEGIN
    x <= a + b ;
    WAIT UNTIL (( x * 10 ) < 100 ) ;
  END PROCESS ;
```

*Tant que le signal x est inférieur à 10, attendre*

*x ne sera pas évalué en fonction de cette expression*

*Pour débloquer la situation, x doit être modifié dans un autre PROCESS*

WAIT

```
WAIT FOR 10 ns ;

WAIT FOR ( a * ( b + c ) ) ;

WAIT ON nmi, interrupt UNTIL (nmi = TRUE)
OR (interrupt = TRUE)) ;
```

*le symbole de l'unité peut varier*

*après évaluation, l'unité est celle définie par défaut*

*attendre un événement sur nmi OU interrupt, ET que l'un des deux soit à 1*

WAIT

Liste de sensibilité d'un PROCESS	WAIT ON a, b ;
PROCESS ( a, b ) BEGIN	PROCESS BEGIN
END PROCESS ;	WAIT ON a, b ;  END PROCESS ;
le PROCESS n'est pas déclenché en attendant un changement sur a ou b	Conclusion ?

WAIT versus Liste de sensibilité

Liste de sensibilité d'un PROCESS	WAIT ON a, b ;
PROCESS ( clk ) VARIABLE last_clk : t_wlogic := U ; BEGIN IF (clk /= last_clk) AND (clk = F1) THEN q <= din ; ENDIF ; last_clk := clk ; END PROCESS ;	PROCESS VARIABLE last_clk : t_wlogic := U ; BEGIN IF (clk /= last_clk) AND (clk = F1) THEN q <= din ; ENDIF ; last_clk := clk ; WAIT ON clk ; END PROCESS ;

Fonctionnement concurrentiel ; différence variable et signal

non-optimisé	optimisé
<pre>PROCESS variable A : integer := 0 ; variable B : integer := 1 ; variable C : integer := 2 ; variable X : integer := 3 ;  BEGIN A := B + C ; X := A + C ; A := X + C ; END PROCESS ; (A := 7)</pre>	<pre>PROCESS signal A : integer &lt;= 0 ; signal B : integer &lt;= 1 ; signal C : integer &lt;= 2 ; signal X : integer &lt;= 3 ;  BEGIN A &lt;= B + C ; X &lt;= A + C ; A &lt;= X + C ; END PROCESS ; (A &lt;= 5)</pre> <p><i>Affectation à un signal est planifiée pendant le parcours du process, et effectué à la fin du process</i></p>
<p><i>Affectation à une variable est instantanée. variable : plus proche du sens informatique habituel</i></p>	


31

Optimisation de style

non-optimisé	optimisé
<pre>IF (bit_1 =0) THEN bit_2 &lt;= '1' ; ELSE bit_2 &lt;= '0' ; ENDIF</pre>	<pre>bit_2 &lt;= NOT bit_1 ;</pre>
<pre>IF (boolean) THEN boolean &lt;= FALSE ; ELSE boolean &lt;= TRUE ; ENDIF</pre>	<pre>boolean &lt;= NOT boolean</pre>

32

Optimisation de style

non-optimisé	optimisé
<p>Rotation à gauche</p> <pre>flag (carry) &lt;= accumulateur(7) ; FOR i IN 7 DOWNT0 0 LOOP accumulateur (i) &lt;= accumulateur (i-1) ; END LOOP ; accumulateur (0) &lt;= flag (carry) ;</pre> 	<pre>flag (carry) &lt;= accumulateur(7) ; accumulateur (7 DOWNT0 1) &lt;= accumulateur (6 DOWNT0 0) ; accumulateur (0) &lt;= flag (carry) ;</pre>

33

Ecrire pour simuler ou pour synthétiser

non-optimisé	optimisé
<pre>IF (bit_1 =0) THEN bit_2 &lt;= '1' ; ELSE bit_2 &lt;= '0' ; ENDIF</pre> <p><i>Synthétiseur &lt; 1000 F synthétisé en un multiplexeur à cause du IF plus sophistiqué multi-driver (3 états) ou égalité (équipotentiel)</i></p>	<pre>bit_2 &lt;= NOT bit_1 ;</pre> <p><i>Directement en équipotentiel quelque soit le moteur de synthèse</i></p>

34

Ecrire pour simuler ou pour synthétiser

non-optimisé	optimisé
<pre>IF (boolean) THEN boolean &lt;= FALSE ; ELSE boolean &lt;= TRUE ; ENDIF</pre> <p><i>Synthèse Multiplexeur ou multi-driver (3 états)</i></p>	<pre>boolean &lt;= NOT boolean</pre> <p><i>Synthèse inverseur probablement pour tout synthétiseur</i></p>

35

Ecrire pour simuler ou pour synthétiser

non-optimisé	optimisé
<p>Rotation à gauche</p> <pre>flag (carry) &lt;= accumulateur(7) ; FOR i IN 7 DOWNT0 0 LOOP accumulateur (i) &lt;= accumulateur (i-1) ; END LOOP ; accumulateur (0) &lt;= flag (carry) ;</pre> <p><i>Synthèse souvent LOOP en compteur + décodage pour agir à une valeur donnée</i></p>	<pre>flag (carry) &lt;= accumulateur(7) ; accumulateur (7 DOWNT0 1) &lt;= accumulateur (6 DOWNT0 0) ; accumulateur (0) &lt;= flag (carry) ;</pre> <p><i>Synthèse équipotentiels ?? Exemple du multiplexeur</i></p>

36

Ecrire pour simuler ou pour synthétiser

- En ce qui nous concerne, on écrira tantôt pour simuler tantôt pour synthétiser ,
- Il est possible, pour rendre la simulation plus réaliste, de rajouter des attributs dans dans le source VHDL pour se rapprocher de la réalité.
- Les coder en VHDL utilisant les attributs et les commandes temporels

37

Commandes temporelles

```
AFTER
USE STD.std_logic.ALL -- bibliothèque de logique standard
ENTITY buf IS
  PORT (e1, e2 : IN t_wlogic ;
        s1, s2, s3 : OUT t_wlogic) ;
END buf ;
ARCHITECTURE buf OF buf IS
BEGIN
  s1 <= e1 AFTER 20 ns ; -- le délai est par rapport au begin pas de cumul donc
  s2 <= e1 AND e2 AFTER 30 ns ;
  s3 <= NOT e1 AFTER 5 ns ; -- le premier à être affecté
END buf ;
```



- AFTER n'a aucune incidence sur la synthèse, juste pour la simulation
- D'autres commande (configuration) permettent de sélectionner une technologie particulière (avec des délais de propa bien définis)

39

Aspects temporels

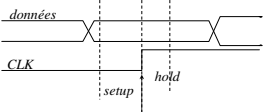
- Rendre la modélisation
    - plus réaliste
    - plus proche des contraintes technologiques
- Définition des différentes caractéristiques temporels des composants :
- Temps de propagation
    - Délai de traversée d'une porte simple (un seul niveau)
- En séquentiel :
- Temps d'établissement
    - Délai de stabilité nécessaire, avant le front actif, pour qu'une entrée soit prise en compte
  - Temps de maintien
    - Délai de stabilité nécessaire, après le front, pour que l'entrée soit prise en compte

38

Attributs Temporels

```
Attribut 'LAST_EVENT
USE STD.std_logic.ALL ;
ENTITY dff IS
  GENERIC ( setup_time, hold_time : TIME ) ;
  PORT ( d, clk : IN t_wlogic ;
        q : OUT t_wlogic ) ;
BEGIN
  setup_check : PROCESS (CLK)
  BEGIN
    IF (CLK = F1 ) and (CLK'event) THEN
      ASSERT ( d'LAST_EVENT >= setup_time )
        REPORT "setup violation"
          SEVERITY ERROR ;
    END IF ;
  END PROCESS setup_check ;
END dff ;
ARCHITECTURE bd OF dff IS
```

Retourne le temps passé depuis le précédent événement arrivé au signal



- La vérification du setup peut-être faite dans l'ENTITY ou l'ARCHITECTURE
- ASSERT exécute REPORT si la condition est fausse ; analogie avec #define ou #if du C
- 4 niveaux de SEVERITY
- note, warning, error, failure

40

Attributs Temporels

```
'ACTIVE
  retourne vrai si une transaction ou un événement a eu lieu sur le signal

  transaction : affectation (type INOUT)

'LAST_ACTIVE
  retourne le temps depuis le dernier événement ou transaction

Analogie avec 'EVENT et 'LAST_EVENT

Attributs de type signal
  un nouveau signal est créé, basé sur le signal de référence (attaché à l'attribut)

s'DELAYED [( temps )]
  crée un signal du même type que s, et qui suit exactement s mais avec un retard de 'temps'
```

41