

LICENCE
EEA

3 - VHDL

SYSTEMES NUMERIQUES & PROCESSEURS EMBARQUES

Plan de Cours

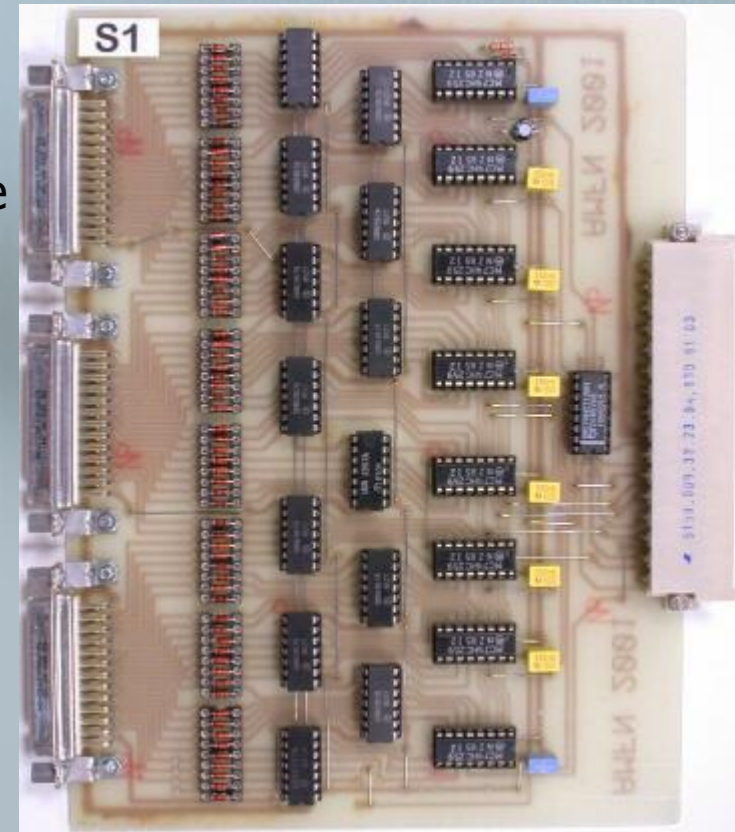
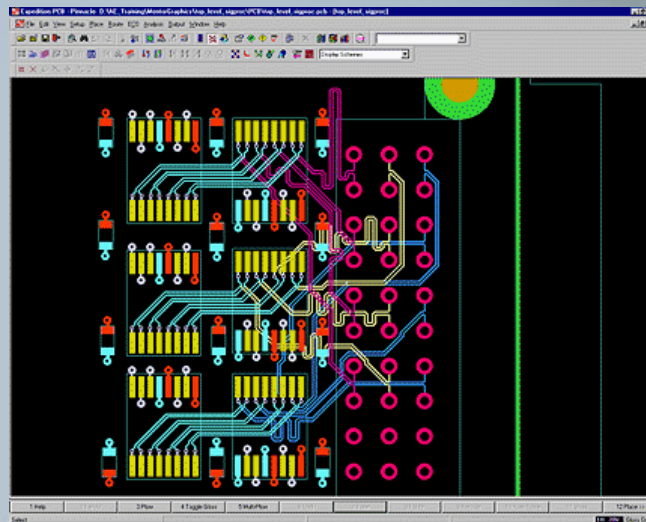
- Intro
- Concepts de Base
- Niveaux de Description – VHDL concurrent
- Process – VHDL séquentiel
- Comment bien coder en VHDL
- Testbench - Simulation
- Annexes: Codes VHDL de modules de base

C3

2

Réalisation de Système Num.

- A (not so) very long time ago...
 - Logique câblée
 - Associations de composants discrets
- Outils de conception:
 - Logiciel de routage de carte

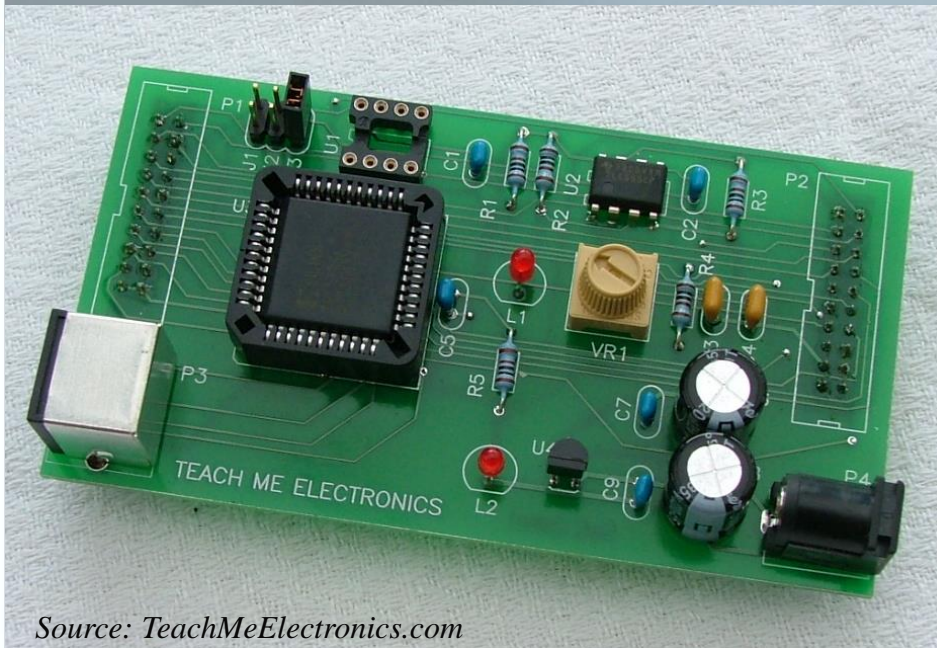


C3

3

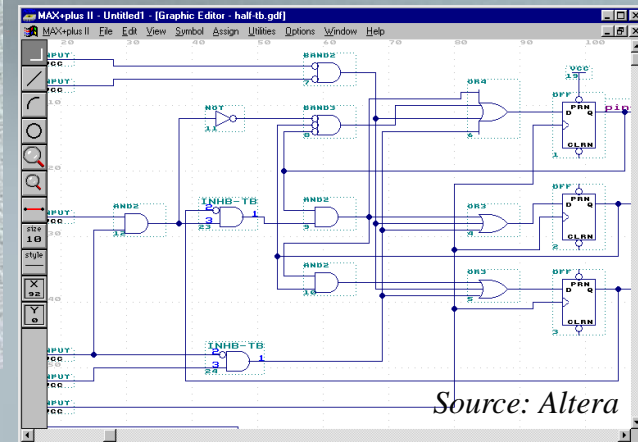
Réalisation de Système Num.

- Some time ago...
 - Logique programmable
 - Apparition de composants programmables de faible ou moyenne densité (PAL, CPLD)



Source: TeachMeElectronics.com

- Outils de conception
 - Saisie de schémas
- Langages de description bas niveau (KARL, ABEL)



Source: Altera

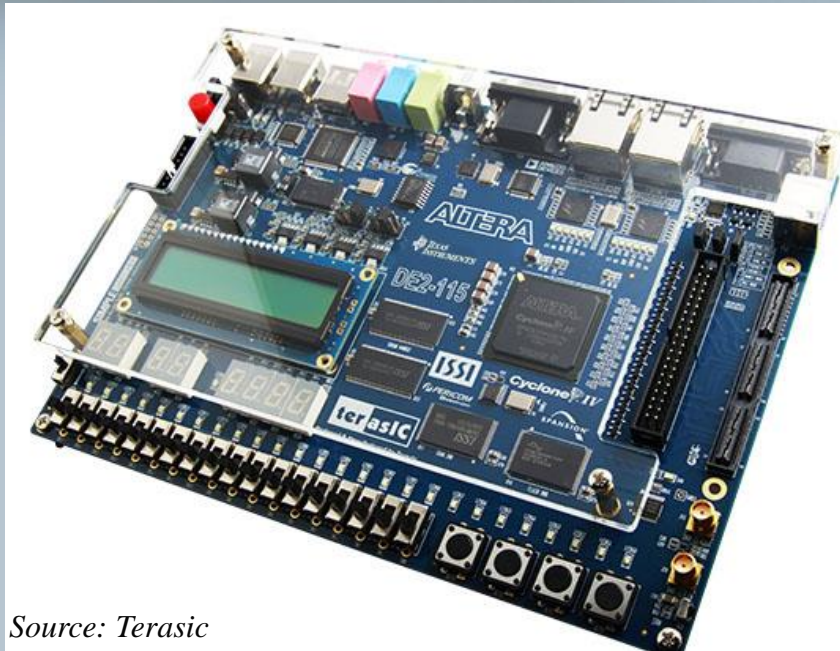
C3

4

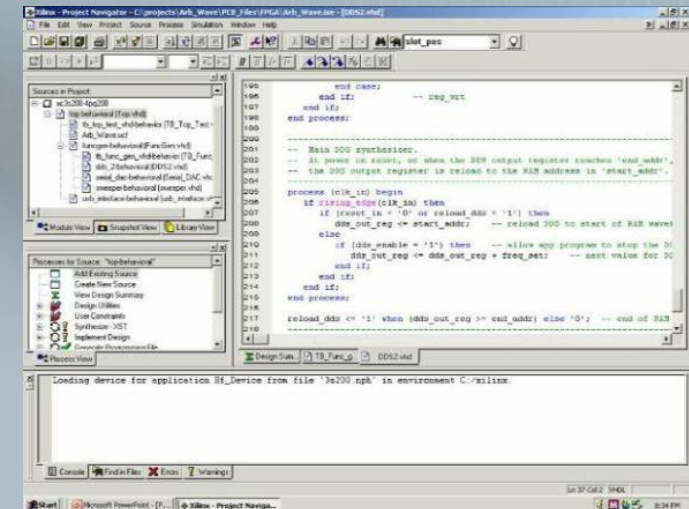
Réalisation de Système Num.

■ Now...

- Augmentation exponentielle de la complexité des circuits
- Outils de conception
 - Logiciels de CAO complexes
 - Langages de description de haut niveau (**VHDL**, **VERILOG**)



Source: Terasic



C3

5

Logiciels

- Xilinx Vivado Design Suite – HLx Editions
 - Chaîne de conception et d'implémentation
 - Webpack: Version gratuite

- Version 2016.4 Utilisé en TP

<https://www.xilinx.com/support/download/index.html/content/xilinx/en/downloadNav/vivado-design-tools/2016-4.html>



C3

6

Installation Vivado WebPack

- Création d'un compte (gratuit) sur le site de Xilinx
- Download Self Extracting Web Installer
- Run Xilinx_Vivado_SDK_2016.4_0124_1_Win64.exe
 - Paramétrage de l'installation
 - Sélectionner l'édition Vivado WebPack
 - Connection au site de Xilinx
 - Obtention d'une licence PERMANENTE*

**Mais pas d'exploitation commerciale des systèmes développés avec le WebPack*

Pour plus d'infos:

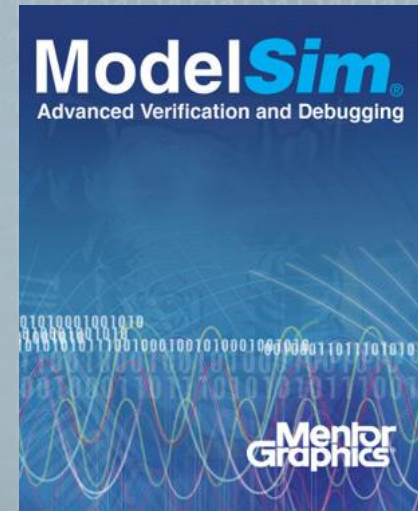
https://www.xilinx.com/support/documentation/sw_manuals/xilinx2016_4/ug973-vivado-release-notes-install-license.pdf

C3

7

Logiciels

- Modelsim – Student Edition
 - Compilation / Simulation de VHDL




- Utilisé en TD et TP

www.mentor.com/company/higher_ed/modelsim-student-edition

- Télécharger puis installer
- A la fin de l'installation, un formulaire d'obtention de licence vous est proposé
 - Remplir le formulaire
 - La licence vous sera envoyée par e-mail

C3

8




```
architecture behave of mug is
    signal sig : std_logic_vector(7 downto 0);
begin

    process (sig)
    begin
        for i in 0 to 2 loop
            sig(i) <= '0';
        end loop;
    end process;

    sig(3) <= '1';

end behave;
```



What is the value of sig after 10 ns?

A: 00001000

B: UUUU1000

C: UUUUU000

D: LA REPONSE D

Source: ht-lab.com

VHDL

Concepts de Base

Langage de Description

Very high speed integrated circuit

Hardware

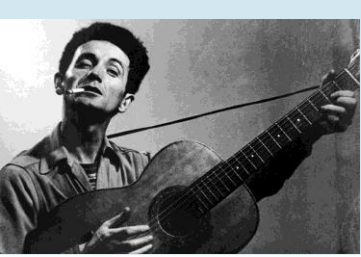
Description

Language

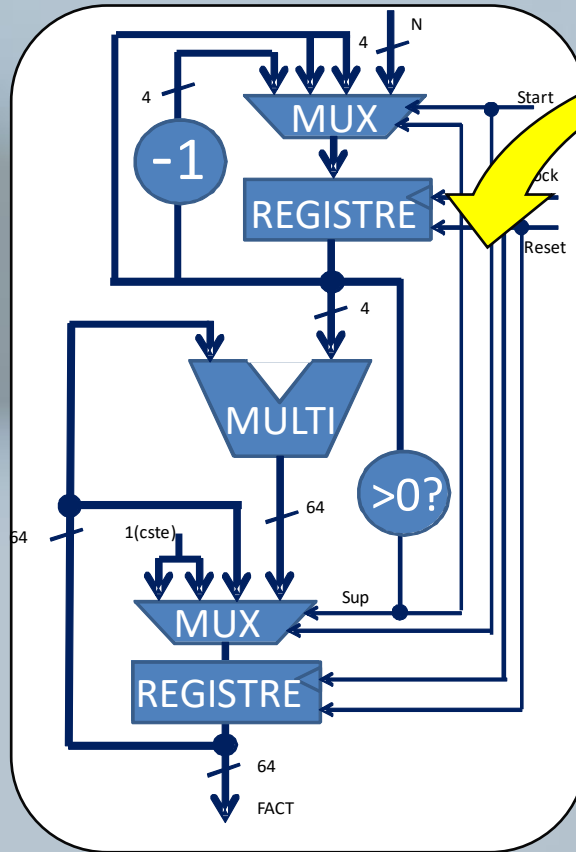
Langage
de
Description
de
Matériel

C3

10



"All you can write is what you see..."



Si vous savez dessiner ça...

Vous saurez écrire ça...

C3

11

```

process(clk,reset)
begin
    -- Reset Asynchrone
    if reset='0' then reg<=(others =>'0'); cpt <= 0;

    elsif rising_edge(clk) then

        -- Gestion du Compteur avec Modulo
        if cpt=29 then cpt<=0;
        else cpt<=cpt+1;
        end if;

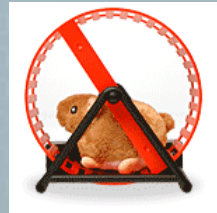
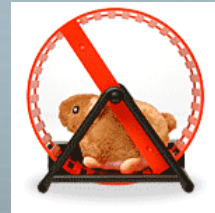
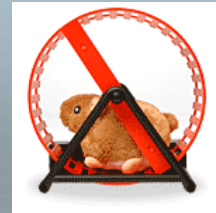
        -- Chargement Operande
        if com="01" then reg<=load;
        -- Envoi Serie
        if com="10" then reg<=
    end if;
end process;
end entity;

```

Langage HDL

- Son but est... de décrire du matériel et d'aboutir à la réalisation d'un circuit
- Spécificités d'un circuit matériel

- Parallélisme



- Permanence des ressources utilisées



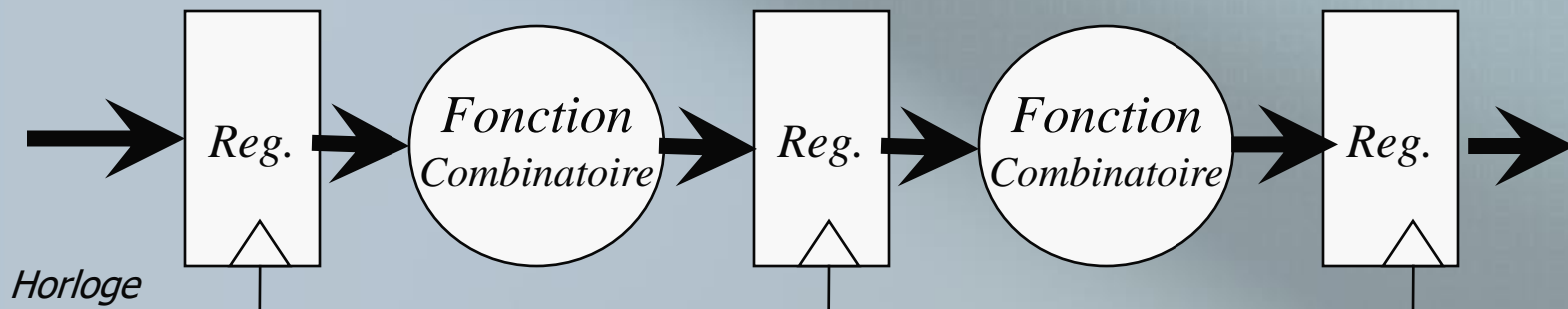
Source: pontdugard.fr

C3

12

VHDL is...

- Apparition en 1987
- Mise à jour du standard en 1993, 2002 puis 2008
- But: Synthèse des circuits numériques
 - Description synthétisable
 - Plusieurs niveaux de description
 - RTL: Register Transfer Level



- Point d'entrée du flot de conception

VHDL is not...

- Approche différente d'un langage informatique "classique" (C)
 - C: langage séquentiel
 - Une instruction s'exécute après une autre
 - VHDL: langage concurrent
 - Dans un système matériel, tous les blocs travaillent en parallèle
 - Les instructions d'un langage VHDL s'exécutent en parallèle

C3

14

Description synthétisable

- Un code VHDL peut compiler correctement
 - Car il est syntaxiquement juste
 - Mais ne pas fonctionner en simulation

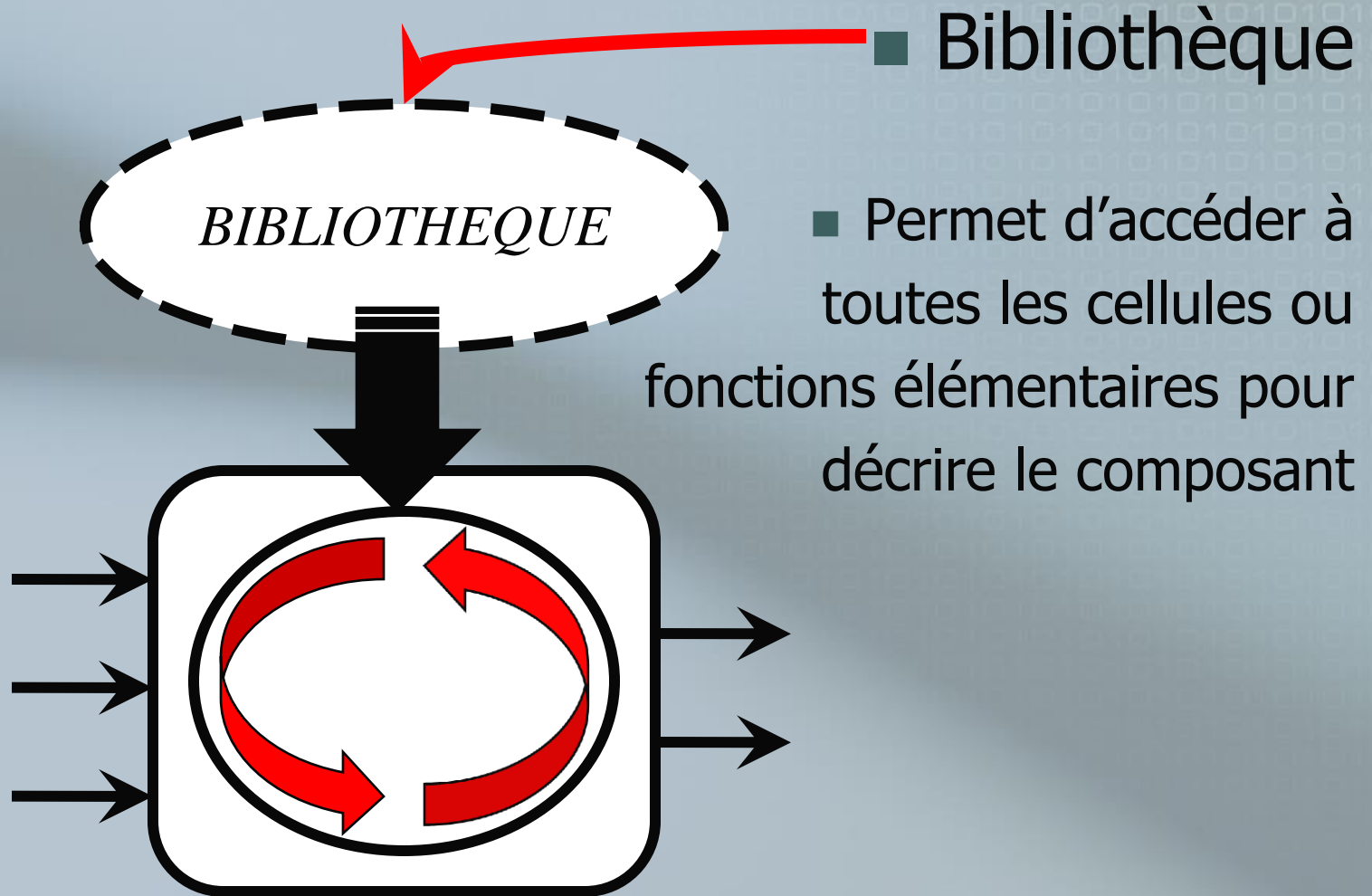
(Comme en C...)

- Un code VHDL peut correctement fonctionner en simulation
 - Mais être rejeté par un outil de synthèse
 - Car la description n'est pas synthétisable

C3

15

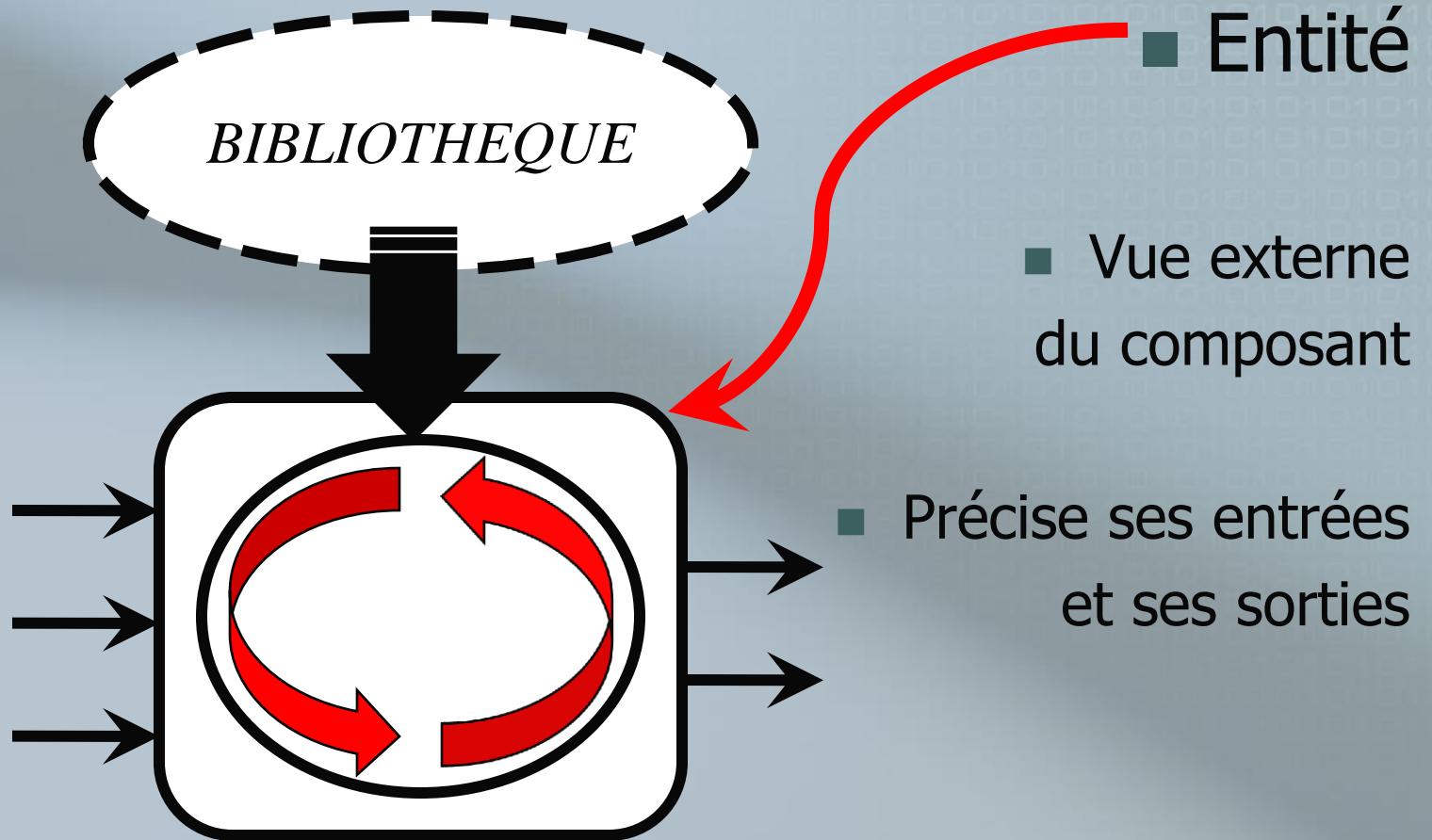
Blocs de base VHDL



C3

16

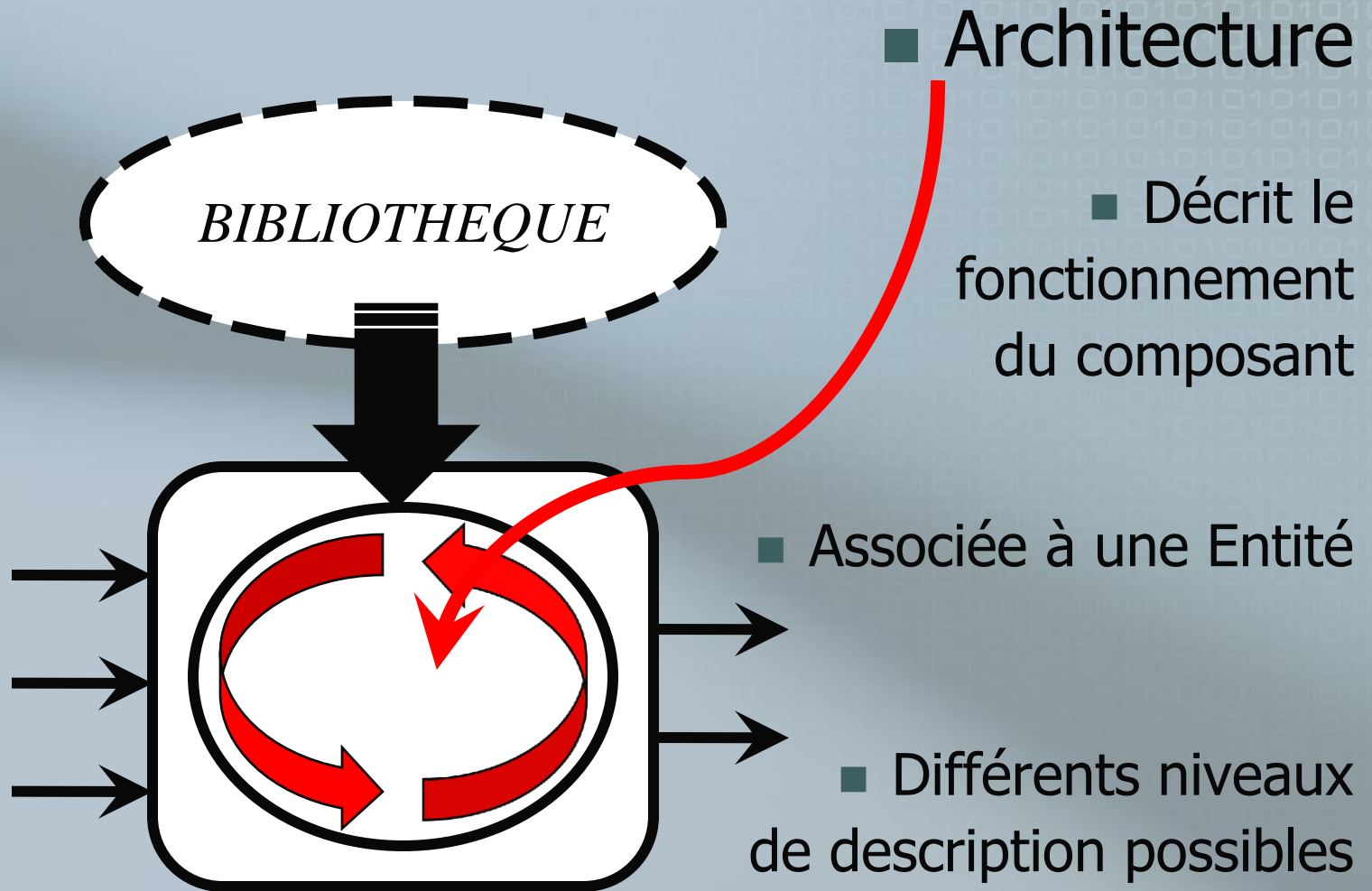
Blocs de Base VHDL



C3

17

Blocs de Base VHDL



C3

18

Exemple

■ Déclaration d'une Bibliothèque

- En tête de programme
- Utilisation de certains packages
 - Un package regroupe des fonctions ou types de données

```
library ieee;
```

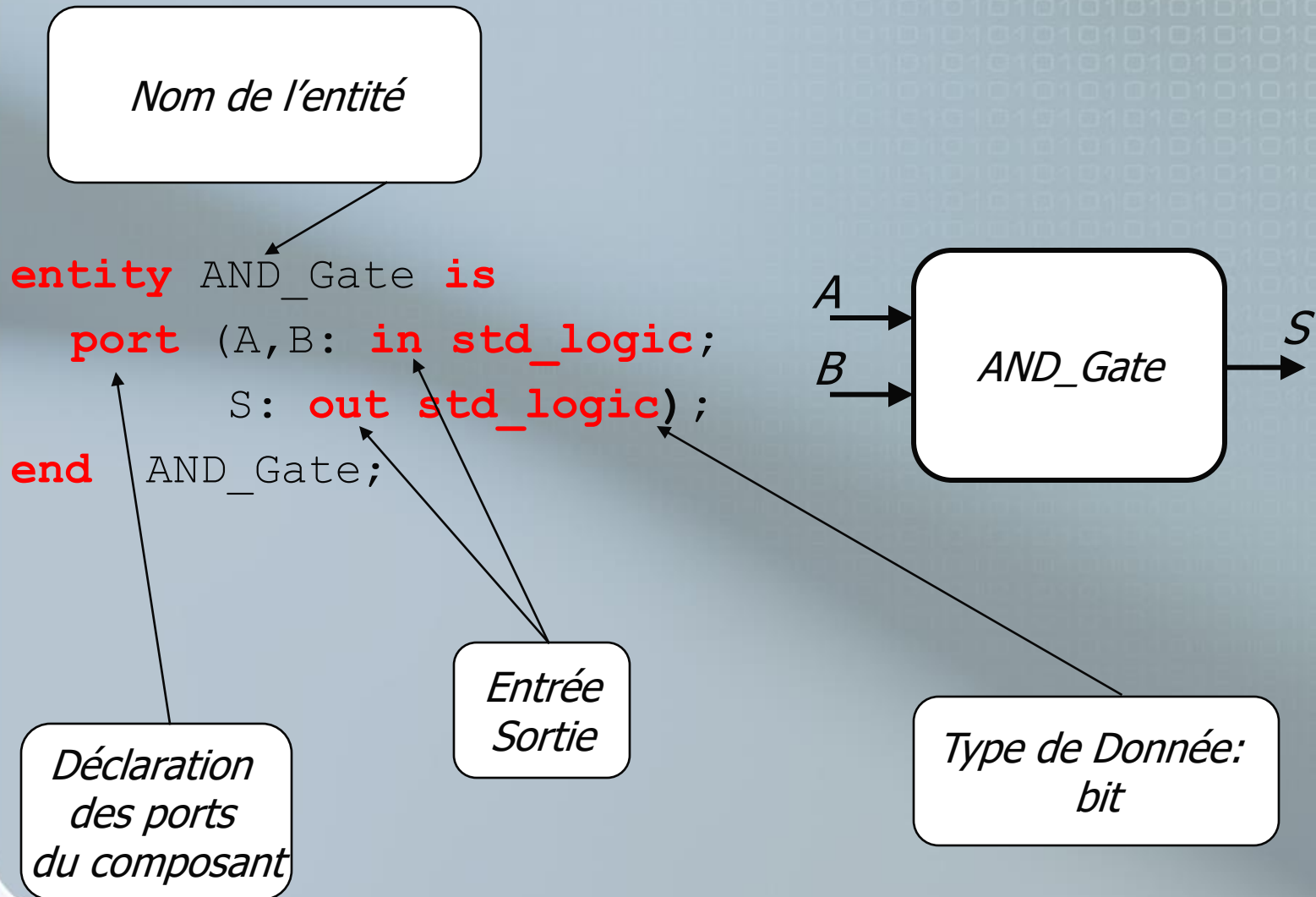
```
use ieee.std_logic_1164.all;
```

```
use ieee.std_logic_unsigned.all;
```

*Utilisation de deux
packages de la
librairie ieee*

*Utilisation de
toutes les fonctions
du package*

Entité



C3

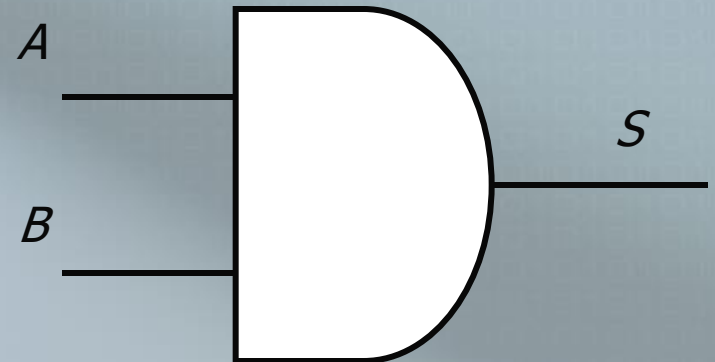
20

Architecture

C3

21

```
architecture Mon_Archi of AND_Gate is  
begin  
    S <= A and B;  
end Mon_Archi;
```



Ports

- Le port permet de rentrer ou de sortir une donnée dans le module que l'on décrit
- Le type de la donnée est précisé lors de la déclaration du port
- 3 types de ports
 - Entrée : TOTO: `in std_logic;`
 - Sortie : TITI: `out std_logic;`
 - Bidirectionnel : TUTU: `inout std_logic;`
 - A n'utiliser que si on veut VRAIMENT faire du bidirectionnel (description de bus, etc...)

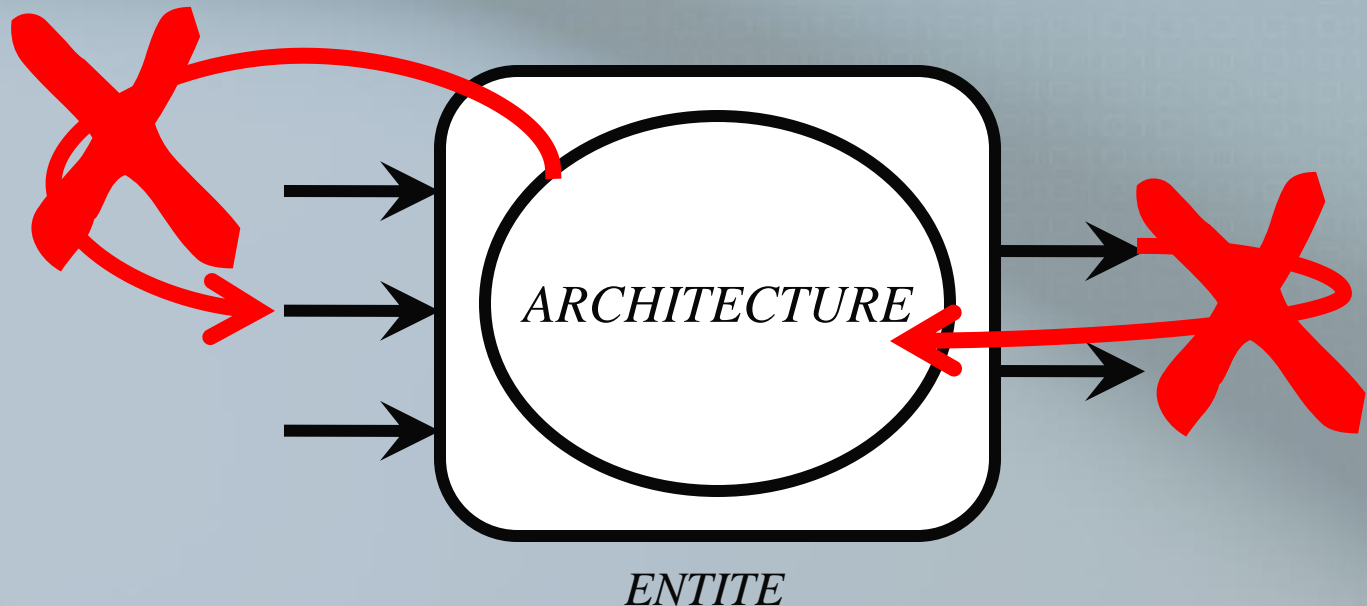
C3

22

Affectation de ports

■ QU'EST-CE QU'ON NE PEUT PAS FAIRE?

- On ne peut pas affecter (écrire) une valeur sur un port d'entrée
- On ne peut pas lire la valeur d'un port de sortie pour l'affecter à un signal ou un autre port



C3

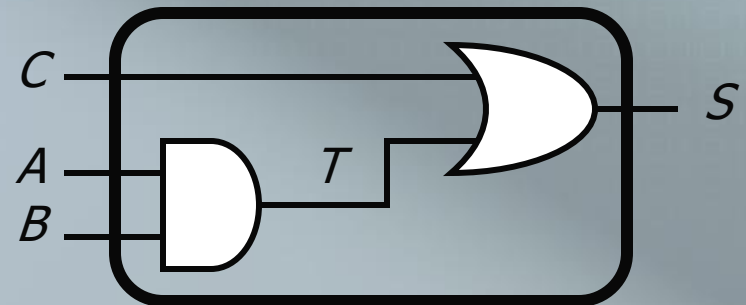
23

Signaux

- Un signal peut être vu comme un fil connectant deux parties d'un circuit
 - Il peut être de n'importe quel type (précisé lors de la déclaration)
 - Il se déclare dans l'architecture (avant le begin)
 - L'affectation d'un signal se fait après le begin

```
entity exemple is
  port (A,B,C: in std_logic;
        S: out std_logic);
end exemple;

architecture archi of exemple is
  signal T: std_logic;
begin
  T <= A and B;
  S <= C or T;
end exemple;
```



C3

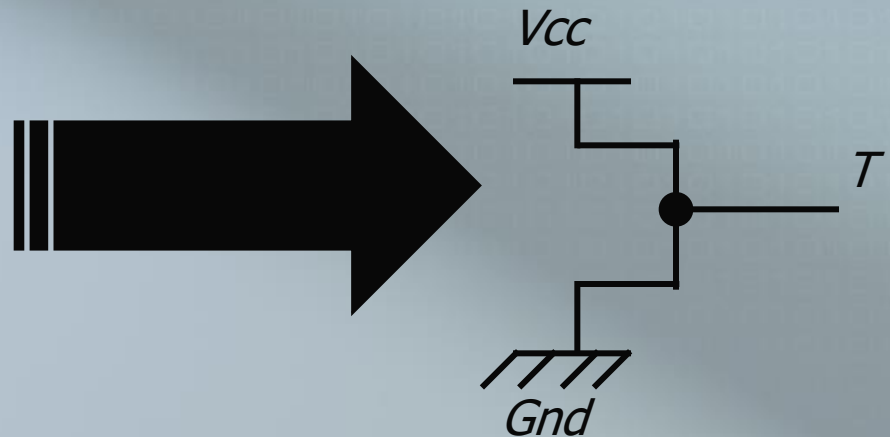
24

Signaux

■ REGLE FONDAMENTALE

- Un signal (ou un port de sortie) ne peut être affecté que par UNE SEULE instruction

```
architecture bad of exemple is  
  signal T: std_logic;  
begin  
    T <= '0';  
    T <= '1';  
end bad;
```



C3

25

Types de Données

- Bit ('0', '1')
 - Bit_vector
 - Boolean (false, true)
 - Integer
 - Natural (entiers ≥ 0)
 - Positive (entiers > 0)
 - Real
 - Character
 - String
 - Time
- Tous ces types ne sont pas synthétisables
(et servent juste à de la modélisation)

C3

26

Types IEEE

- Types de référence pour les bits/vecteurs de bits
- `std_logic`
 - 9 états possibles
 - `'U'`: Non défini
 - `'X'`: Forçage Indéterminé (Conflit)
 - `'0'`: Forçage au 0 logique
 - `'1'`: Forçage au 1 logique
 - `'Z'`: Haute impédance
 - `'W'`: Niveau faible - Indéterminé
 - `'H'`: Niveau 1 faible
 - `'L'`: Niveau 0 faible
 - `'-'`: Peu importe
- `std_logic_vector`
 - Vecteur de `std_logic`
 - `signal A: std_logic_vector(3 downto 0); -- 4 bits`

C3

27

Affectations

- Opérateur d'affectation pour les signaux et les ports

<=

- Affectation d'un entier : `A <= 12;`
- Affectation d'un bit : `B <= '0';`
- Affectation d'un vecteur de bits: `C <= "001";`

C3

28

Opérateurs en VHDL

■ Opérateurs logiques (bit à bit)

- **not**
- **and**
- **or**
- **nand**
- **nor**
- **xor**
- **xnor**
- **&** (Concaténation)

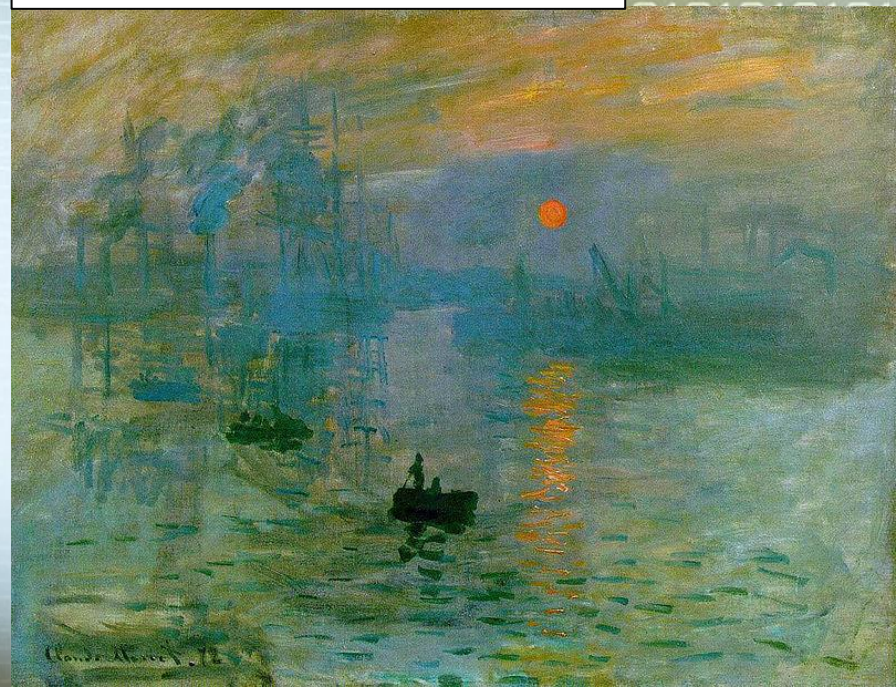
■ Opérateurs arithmétiques

- **+**
- **-**
- ***** (synthétisable mais quelle architecture?)
- **/** (peu synthétisable)
- **mod** (pas synthétisable)
- **exp** (pas synthétisable)

Aivazovsky, *Lever de soleil à Théodosie* (1855)



Monet, *Impression soleil levant* (1872)



VHDL

Niveaux de Description



Langage Concurrent

- Par défaut, VHDL est un langage concurrent
- Cela signifie que toutes les instructions s'exécutent en parallèle
- Cela signifie aussi que l'ordre des instructions n'a pas d'importance

C3

31

Niveaux de Description

- 3 Niveaux sont proposés en VHDL
 - Flot de Données
 - Description bas-niveau
 - Écriture des équations logiques des sorties en fonction des entrées
 - Structurelle
 - Utilisation de composants pré-définis et référencés en bibliothèque
 - Comportementale
 - Description de plus haut niveau
 - Vision "algorithmique" ou fonctionnelle du système

C3

32

Niveaux de Description

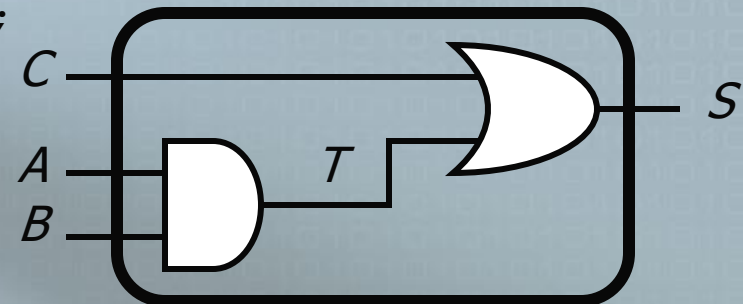
■ Description flot de données

- Rappel: Toutes les instructions s'exécutent EN MÊME TEMPS

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

```
entity exemple is  
    port (A,B,C: in std_logic;  
          S: out std_logic);  
end exemple;
```

```
architecture dataflow of exemple is  
    signal T: std_logic; -- SIGNAL INTERNE  
begin  
    S <= C or T;  
    T <= A and B;  
end dataflow;
```



*L'ordre des
instructions
est indifférent*

C3

33

Niveaux de Description

■ Description structurelle

■ 1 - Description de modules VHDL

- Après compilation, ces modules sont disponibles dans votre bibliothèque de travail

```
library ieee, work;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;
```

```
entity AND_Gate is  
    port (A,B: in std_logic;  
          S: out std_logic);  
end AND_Gate;
```

```
architecture dataflow of AND_Gate  
begin  
    S <= A and B;  
end dataflow;
```

```
library ieee, work;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;
```

```
entity OR_Gate is  
    port (A,B: in std_logic;  
          S: out std_logic);  
end OR_Gate;
```

```
architecture dataflow of OR_Gate  
begin  
    S <= A or B;  
end dataflow;
```

C3

34

Niveaux de Description

■ Description structurelle

- 2 – Réutilisation (Instanciation) des modules dans un autre composant

```
library ieee, work;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;
```

```
entity exemple is  
    port (A,B,C: in std_logic;  
          S: out std_logic);  
end exemple;
```

```
architecture struct of exemple is  
    signal T: std_logic;  
begin  
    label0: entity work.OR_Gate(dataflow)  
        port map (C,T,S);  
    label1: entity work.AND_Gate(dataflow)  
        port map (A,B,T);  
end struct;
```

*Utilisation d'un
composant déjà inclus
dans la bibliothèque*

*Connections des ports
du composant à
ceux du module*

*Instanciation
~
Instruction
concurrente*

C3

35

Niveaux de Description

■ Description comportementale

- Le système n'est plus vu comme une succession de portes
- Vision plus fonctionnelle et algorithmique
 - Permet de décrire plus efficacement des systèmes complexes

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity exemple is  
    port (A,B,C: in std_logic;  
          S: out std_logic);  
end exemple;  
  
architecture behavioral of exemple is  
    signal T: std_logic;  
begin  
    S <= T when C='0'  
    else '1';  
    T <= A when B='1'  
    else '0';  
end behavioral;
```

*L'ordre des
instructions
est indifférent*

C3

36

Syntaxe VHDL Concurrent

■ Affectation conditionnelle

■ **when else**

```
■ S <= "000" when E = "00"  
      else "101" when E = "01"  
      else "011" ;
```

■ **with select**

```
■ with E select  
  S <= "000" when "00",  
      "101" when "01",  
      "011" when others ;
```

C3

37

Syntaxe VHDL Concurrent

■ Duplication d'instructions

■ **generate**

```
entity AND_Gate_4 is
    port ( a,b : in std_logic_vector(3 downto 0);
          s   : out std_logic_vector(3 downto 0);
    end AND_Gate_4;
```

```
architecture arc of AND_Gate_4 is
```

```
begin
```

```
    LABEL_1: for i in 0 to 3 generate
        s(i) <= a(i) and b(3-i);
```

```
    end generate;
```

```
end arc;
```

C3

38

Syntaxe VHDL Concurrent

■ Duplication d'instanciations

■ **generate**

```
entity AND_Gate_4 is
    port ( a,b : in std_logic_vector(3 downto 0);
          s   : out std_logic_vector(3 downto 0);
    end AND_Gate_4;
```

```
architecture arc of AND_Gate_4 is
```

```
begin
```

```
    LABEL_1: for i in 0 to 3 generate
        LABEL_2: entity work.AND_Gate_2
            port map (a(i),b(3-i),s(i));
    end generate;
```

```
end arc;
```

C3

39

Généricité

- VHDL permet de paramétrer les tailles des données d'un composant
- Intérêt: Réutilisation d'une même description pour plusieurs modules
- Instruction **generic**

C3

40

Généricité

■ Exemple

```
entity Module is
    generic( N: natural := 4 );
    port (a,b : in std_logic_vector(N-1 downto 0);
          s : out std_logic_vector(N-1 downto 0));
end Module;

architecture arc of Module is
    signal T: std_logic_vector(N-1 downto 0);
    begin

        T <= not A;
        S <= A and B;

    end arc;
```

*Valeur par défaut
du paramètre*

C3

41

Généricité

■ Généricité et instantiation de composant

```
entity Module2 is
    port (a,b : in std_logic_vector(31 downto 0);
          s : out std_logic_vector(31 downto 0));
end Module2;

architecture arc of Module2 is
    signal T: std_logic_vector(31 downto 0);
begin
    LABEL_2: entity work.Module
        generic map(32)
        port map (a,b,t);

    S <= not T;
end arc;
```

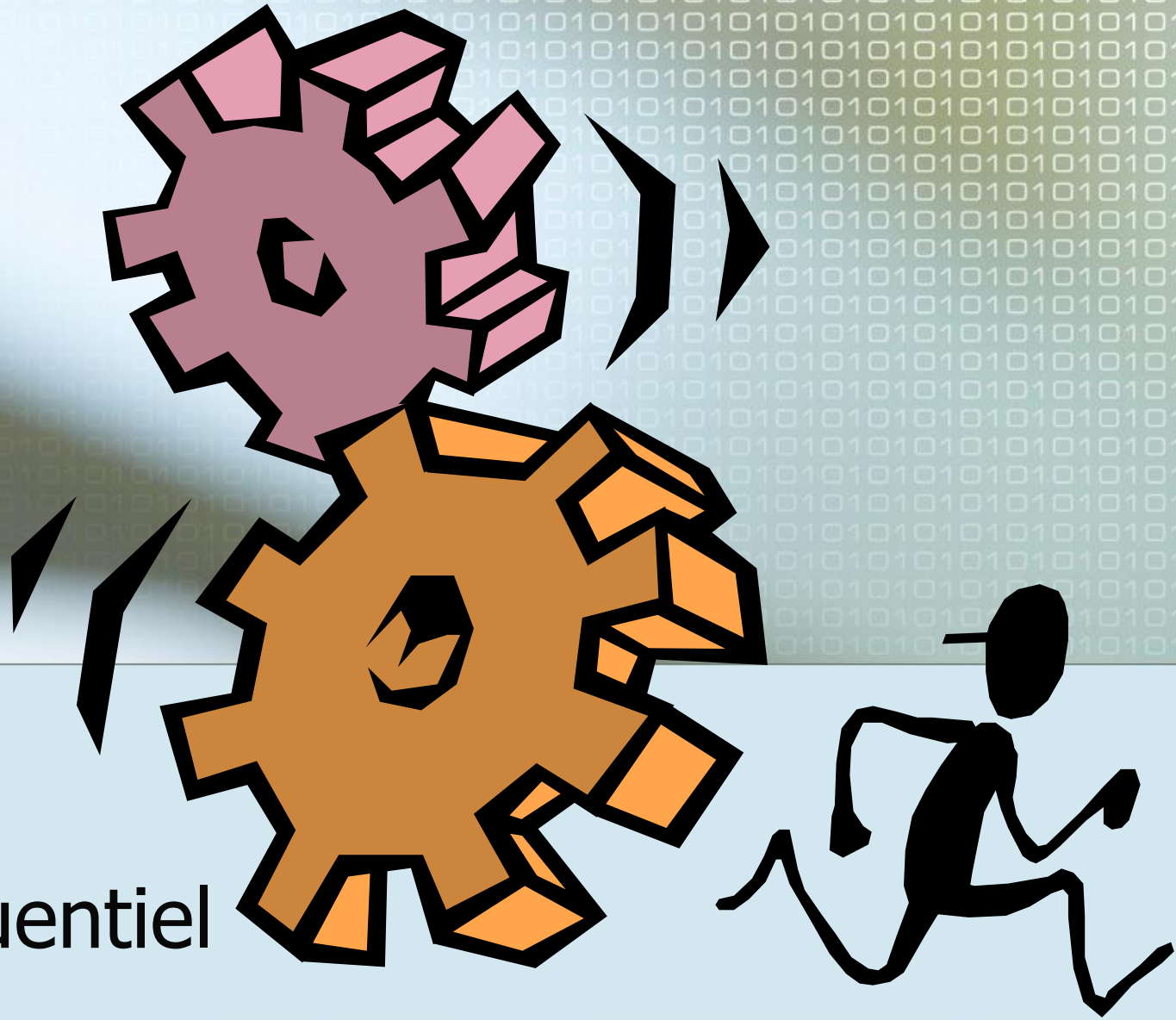
*Changement
de la valeur
du paramètre N*

C3

42



Process
VHDL Séquentiel



VHDL Concurrent/Séquentiel

- Les descriptions vues précédemment sont appelées **VHDL concurrent**
 - Pour certains composants, cette méthode de description n'est pas toujours la plus agréable à utiliser
- VHDL offre la possibilité de décrire séquentiellement le comportement d'un système
- Cela se fait dans une structure de programmation appelée **process**
- On parle alors de **VHDL séquentiel**

C3

44

VHDL Concurrent/Séquentiel



- Ne pas confondre:
 - Électronique combinatoire/séquentielle
 - Description VHDL concurrente/séquentielle

- On peut décrire du combinatoire en VHDL séquentiel
- On peut décrire du séquentiel en VHDL concurrent
- Et vice et versa...

- Seuls la syntaxe et le style de programmation changent...

C3

45

Process

- Un process peut être vu comme une fonction dont l'exécution dépend de la variation de signaux/ports à observer
- Il permet une description plus algorithmique du système.
- Les signaux/ports que l'on souhaite observer font partie de la **liste de sensibilité** du process
 - Si un des signaux de la liste est modifié, le process s'exécute
 - Si aucun des signaux n'est modifié, le process reste en veille

C3

46

Liste de sensibilité

■ REGLE FONDAMENTALE

- Lorsque l'on décrit dans un process un système électronique combinatoire, la liste de sensibilité doit comporter TOUTES LES ENTREES du système
- Lorsque l'on décrit dans un process un système électronique séquentiel, la liste de sensibilité doit comporter L'HORLOGE et les ENTREES DE FORCAGE ASYNCHRONE (reset, preset)

C3

47

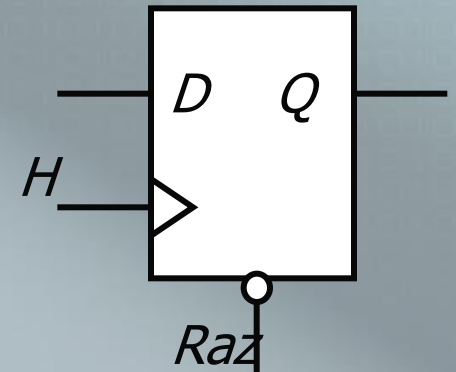
Exemple

■ Description d'une bascule D

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity Bascule is  
port(h, raz, d: in std_logic;  
      q: out std_logic);  
end Bascule;  
  
architecture archi of Bascule is  
begin  
    process(h, raz)  
    begin  
        if raz='0' then q<='0';  
        elsif rising_edge(h) then q<=d;  
        end if;  
    end process;  
end archi;
```

*Liste de sensibilité
process s'exécute si
évolution des signaux*

Test du front montant



C3

48

Simulateur VHDL

- Simulation à Événements Discrets
 - A chaque pas de simulation
 - Evolution des valeurs des entrées
 - Ces changements activent les listes de sensibilités de certains process
 - Exécution de ces process
 - Mise à jour des sorties des process et des instructions concurrentes du programme

C3

49

Process et Simulation

- Pour garantir l'aspect concurrent d'un programme VHDL et la bonne synchronisation des instructions, un process possède les caractéristiques suivantes:
 - Pour le simulateur, le temps d'exécution d'un process est nul (sauf si un délai y est explicitement indiqué)
 - Lors de la simulation du programme, l'ensemble des instructions d'un process est vu comme une grosse instruction concurrente
 - Pour le simulateur, deux process ayant la même liste de sensibilité seront activés aux mêmes pas de simulation.
 - Un signal/port dont la valeur est modifiée au cours d'un process n'est remis à jour QU'APRES L'EXECUTION DE TOUS LES PROCESS

C3

50

Exemple

```
entity Exemple is
port (clk, e: in std_logic;
      s: out std_logic);
end Exemple;
```

```
architecture archi of Exemple is
signal t: std_logic;
```

```
begin
```

```
  process (clk)
```

```
  begin
```

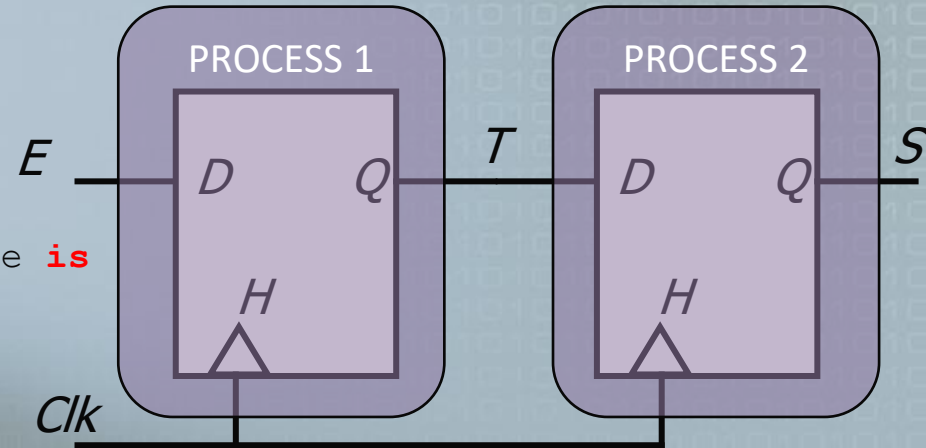
```
    if rising_edge(clk) then t<=e; end if;
  end process;
```

```
  process (clk)
```

```
  begin
```

```
    if rising_edge(clk) then s<=t; end if;
  end process;
```

```
end archi;
```



C3

51

Quel est le comportement des signaux T et S?

Autre Exemple

- Affectation et mise à jour d'un signal dans un process

```
entity Bascule is
port(h, raz: in std_logic;
      sortie: out std_logic);
end Bascule;

architecture archi of Bascule is
signal cpt: std_logic_vector(2 downto 0);
begin
  process(h, raz)
  begin
    if raz='0' then cpt<="000"; sortie<='0';
    elsif rising_edge(h) then
      cpt<=cpt+1;
      if cpt="110" then sortie<='1'; else sortie<='0'; end if;
    end if;
  end process;
end archi;
```

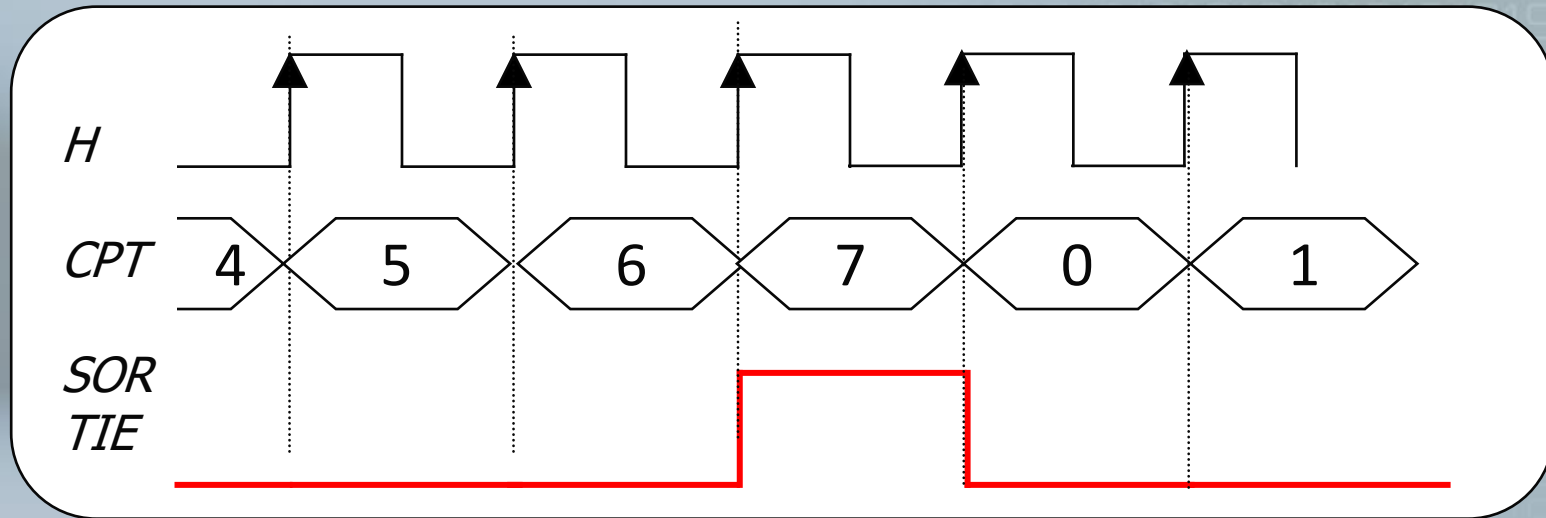
*Le test se fait avec
la valeur qu'a le signal cpt
en DEBUT de process*

*Mise à jour de cpt
et de sortie*

C3

52

Simulation

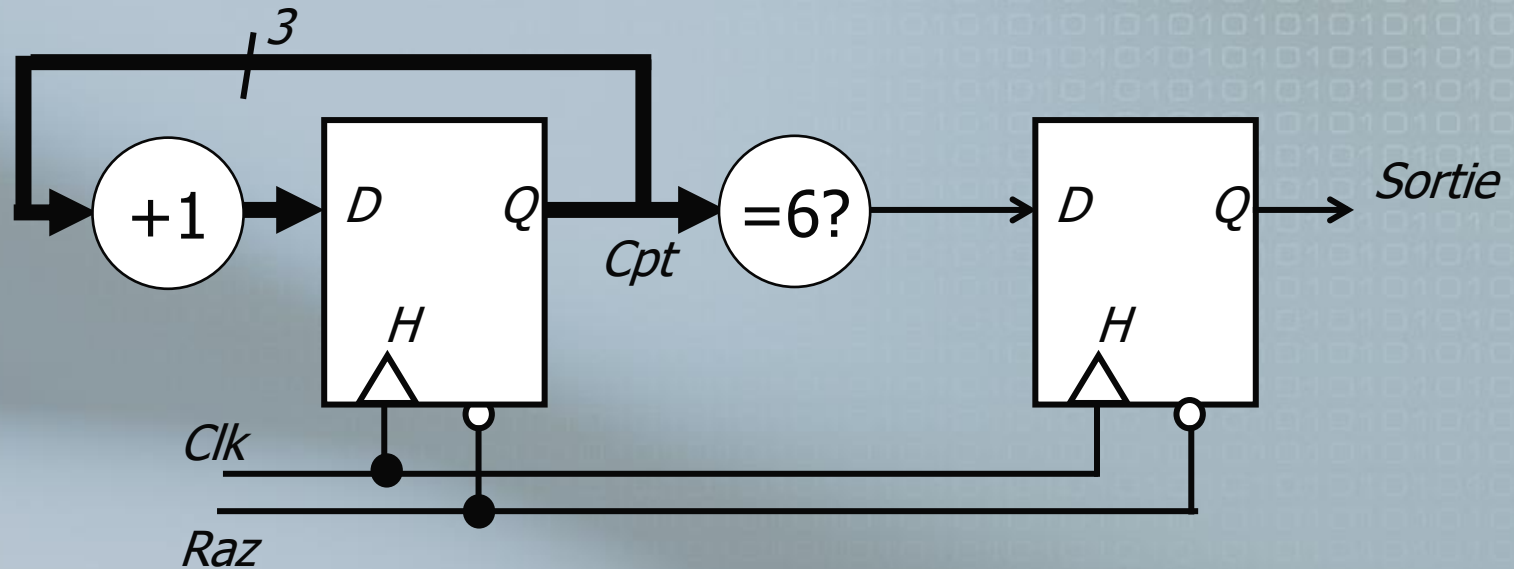


C3

```
process (h, raz)
begin
    if raz='0' then cpt<="000"; sortie<='0';
    elsif rising_edge(h) then
        cpt<=cpt+1;
        if cpt="110" then sortie<='1';
        else sortie<='0';
        end if;
    end if;
end process;
```

53

Architecture synthétisée



C3

- "Sortie" est synthétisé sous la forme d'une bascule D

- **REGLE:** Tous les signaux/ports qui sont affectés dans un process sensible sur une horloge seront synthétisés sous forme de bascules/registres

54

Variables

- Le process dispose d'une structure de données propre: la **variable**
- Une variable sert à stocker un résultat intermédiaire dans le déroulement d'un algo
- Contrairement au signal, une variable dont la valeur est modifiée est IMMEDIATEMENT remise à jour
- Il n'est pas possible d'utiliser une variable en dehors d'un process

C3

55

Variables

■ Exemple – Détection de parité

```
entity Parity is
port(x: in std_logic_vector(7 downto 0);
      p: out std_logic);
end Parity;
```

*Déclaration
d'une variable*

```
architecture archi of Parity is
begin
```

```
  process (x)
```

```
    variable tmp : std_logic := '0';
```

```
  begin
```

```
    for i in 0 to 7 loop -- boucle for
```

```
      if x(i) = '1' then tmp := not tmp; end if;
```

```
    end loop; -- fin de la boucle for
```

```
    p <= tmp;
```

```
  end process;
```

```
end archi;
```

*Opérateur d'affectation
différent de
celui des signaux*

*Affectation
de la variable dans
un signal/port de sortie*

C3

56

Signal vs Variable

SIGNAL

- Portée:
 - Tout le programme
- Opérateur d'affectation:
 \leq
- Mise à jour
 - VHDL concurrent:
 - Immédiate
 - VHDL séquentiel (process)
 - A la sortie du process
- INTERET
 - Description de matériel

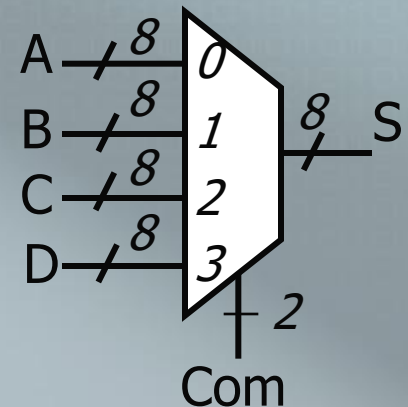
VARIABLE

- Portée:
 - Locale à un process
- Opérateur d'affectation:
 $:=$
- Mise à jour
 - VHDL concurrent:
 - Impossible
 - VHDL séquentiel (process)
 - Immédiate
- INTERET
 - Stockage d'un résultat intermédiaire d'un algo

Instructions VHDL Séquentiel

■ IF – THEN – ELSIF – ELSE

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity Mux41 is  
port (a,b,c,d: in std_logic_vector(7 downto 0);  
      com: in std_logic_vector(1 downto 0)  
      s: std_logic_vector(7 downto 0));  
end Mux41;  
  
architecture archi of Mux41 is  
begin  
  process (a,b,c,d,com)  
  begin  
    if com="00" then s<=a;  
    elsif com="01" then s<=b;  
    elsif com="10" then s<=c;  
    else s<=d;  
    end if;  
  end process;  
end archi;
```



C3

58

Instructions VHDL Séquentiel

■ CASE – WHEN

```
entity Mux41 is
port (a,b,c,d: in std_logic_vector(7 downto 0);
      com: in std_logic_vector(1 downto 0)
      s: out std_logic_vector(7 downto 0));
end Mux41;
```

```
architecture archi of Mux41 is
begin
```

```
    process (a,b,c,d,com)
```

```
    begin
```

```
        case (com) is
```

```
            when "00" => s<=a;
```

```
            when "01" => s<=b;
```

```
            when "10" => s<=c;
```

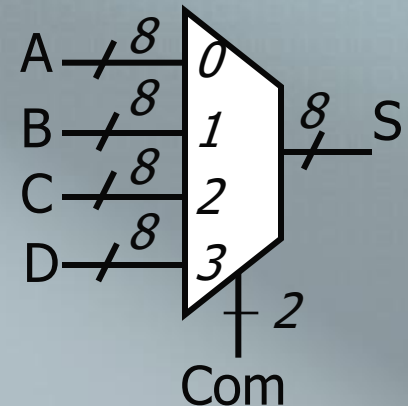
```
            when "11" => s<=d;
```

```
            when others => NULL;
```

```
        end case;
```

```
    end process;
```

```
end archi;
```



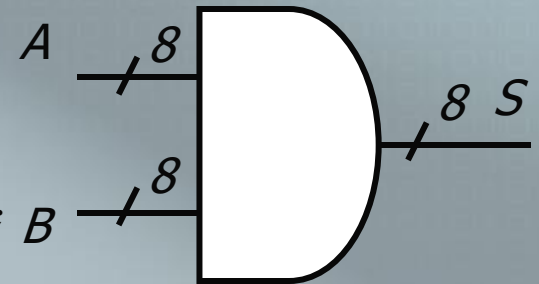
C3

59

Instructions VHDL Séquentiel

■ FOR – LOOP

```
library ieee;  
use ieee.std_logic_1164.all;  
use ieee.std_logic_unsigned.all;  
  
entity exemple_for is  
port(a,b: in std_logic_vector(7 downto 0);  
      s: std_logic_vector(7 downto 0));  
end exemple_for;  
  
architecture archi of exemple_for is  
begin  
  process(a,b)  
  begin  
    for i in 0 to 7 loop  
      s(i) <= a(7-i) and b(i);  
    end loop;  
  end process;  
end archi;
```



C3

60



Bien programmer en VHDL

Ce qu'on doit faire, ce qu'il vaut mieux éviter...

Règles de syntaxe

- VHDL n'est pas case sensitive
 - Pas de différence entre minuscules et majuscules
- Commentaires
 - --
 - La suite de la ligne est en commentaires

C3

62

Règles de syntaxe

■ Initialisation de signaux

```
signal toto: std_logic_vector(2 downto 0) := "101";
```

- La valeur initiale sera prise en compte en simulation
- Ce ne sera pas forcément le cas en synthèse
 - Cela dépend de l'outil
- Une initialisation "propre" d'une information dans signal ou d'un port de sortie se fait à l'aide d'un reset asynchrone...

Règles de syntaxe

- Pour affecter une partie d'un vecteur à 1 et le reste à 0

```
signal titi: std_logic_vector(7 downto 0);
```

```
titi <= (7 => '1', 4 downto 2 => '1', others => '0');  
-- idem titi <= "10011100";
```

*Pas de parenthèses
dans ce cas...*

- Pour tout affecter à 0

```
titi <= (others => '0'); -- idem titi <= "00000000"
```

C3

64

Règles de syntaxe

■ Affectation de valeurs hexadécimales

```
signal toto: std_logic_vector(7 downto 0);  
toto <= x"3C"; -- idem toto <="00111100";
```

■ Attention: Le nombre de bits du vecteur doit être un multiples de 4

```
signal titi: std_logic_vector(6 downto 0);  
titi <= x"3C"; -- Erreur à la Compilation!
```

C3

65

Règles de syntaxe

■ Instanciation

```
entity module is
    port (H, Raz: in std_logic;
          E: in std_logic;
          S: out std_logic);
end module;
```

```
entity Top_Level is
    port (Clock, Reset, E1, E2: in std_logic;
          S1, S2: out std_logic);
end Top_Level;
```

```
architecture archi of Top_Level is
```

```
begin
```

```
    Inst_1: entity work.module
```

```
    port map (Clock, Reset, E1, S1);
```

```
    -- Instanciation par Position: même ordre que l'entité module
```

```
    Inst_2: entity work.module
```

```
    port map (Raz => Reset, S => S2, H => Clock, E => E2);
```

```
    -- Instanciation par Nom: l'ordre est indifférent
```

```
end archi;
```

C3

66

Règles de syntaxe

■ Attributs

- Permet d'extraire une caractéristique d'un signal/port/variable X
- Syntaxe: X' **attribut**

■ Exemples:

- **signal** toto: **std_logic_vector**(N-1 **downto** 0);
 -- N paramètre générique
 if (toto'**high** = 4) **then** ...
 -- Est-ce que ma donnée est sur 5 bits?
- **signal** horloge: **std_logic**;

 if (horloge'**event and** horloge = '1') **then** ...
 -- Idem rising_edge(horloge)

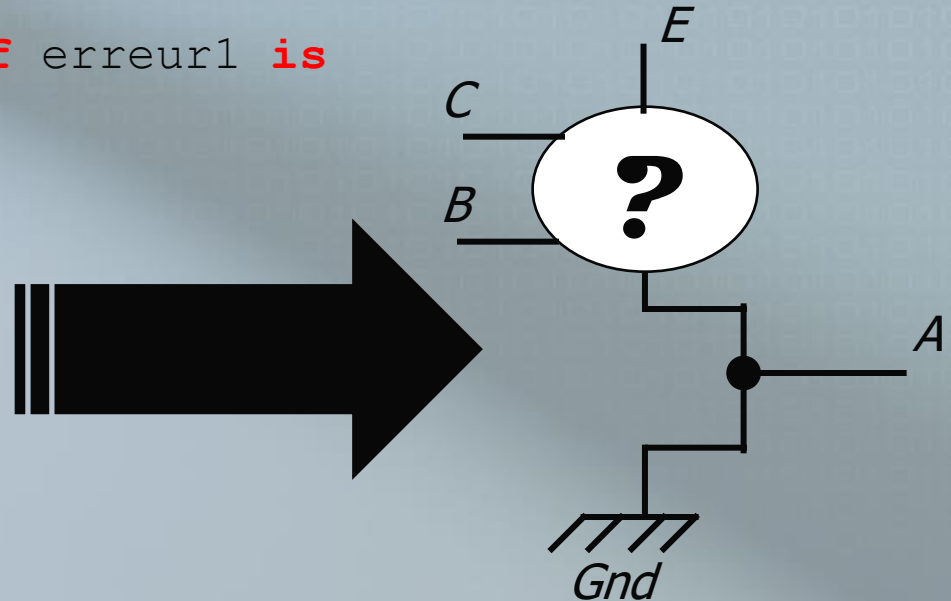
Règles générales

- Un signal/port doit être décrit à UN et UN SEUL endroit
- Pourquoi?
 - Car TOUTES les instructions concurrentes s'exécutent en parallèle

```
architecture archi of erreur1 is  
begin
```

```
  a  <= '0' ;  
  s1 <= c and d ;  
  s2 <= e or c ;  
  
  a  <= b when e='1'  
  else c ;
```

```
end archi;
```



C3

68

Règles générales

- Un signal/port est décrit soit dans UN SEUL process, soit dans UNE SEULE instruction concurrente
- Pourquoi?
 - Car un process est vu comme une grosse instruction concurrente

```
architecture archi of erreur1 is  
begin
```

```
    a <= '0';
```

```
    process (b, c, e)  
    begin
```

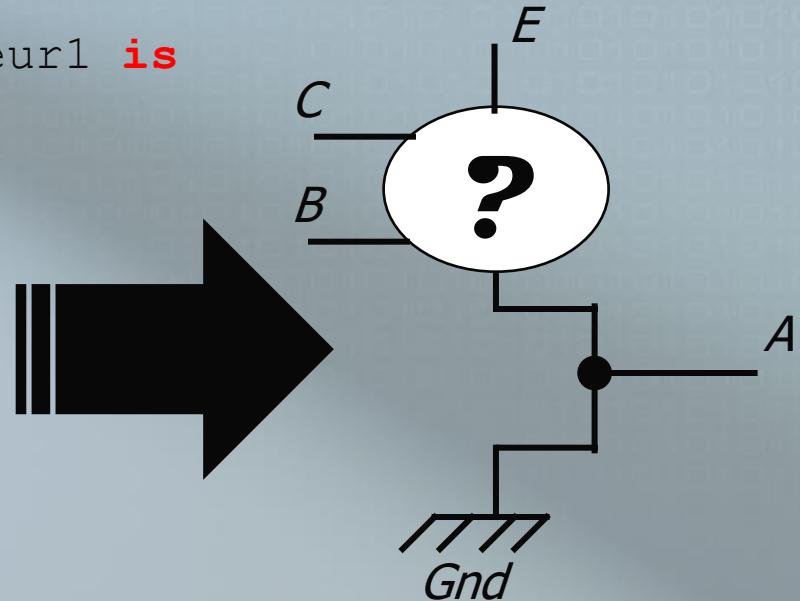
```
        if e='1' then a<=b;
```

```
        else a<=c;
```

```
        end if;
```

```
    end process;
```

```
end archi;
```

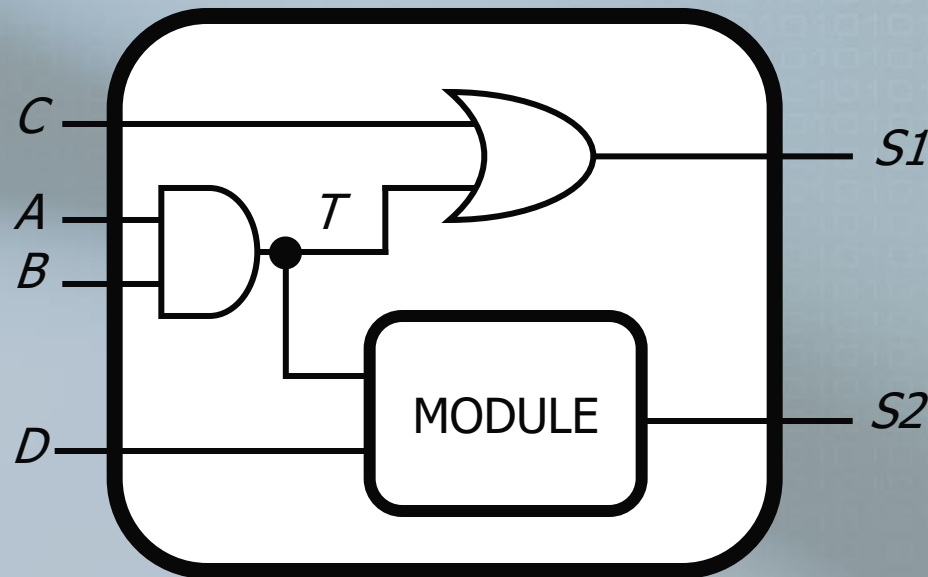


C3

69

Règles générales

- De même, une instantiation de module est considérée comme une instruction concurrente



*S2 est affecté à la sortie du bloc MODULE,
on ne peut donc pas ajouter d'instruction type $S2 \leq \text{toto};$*

Règles générales

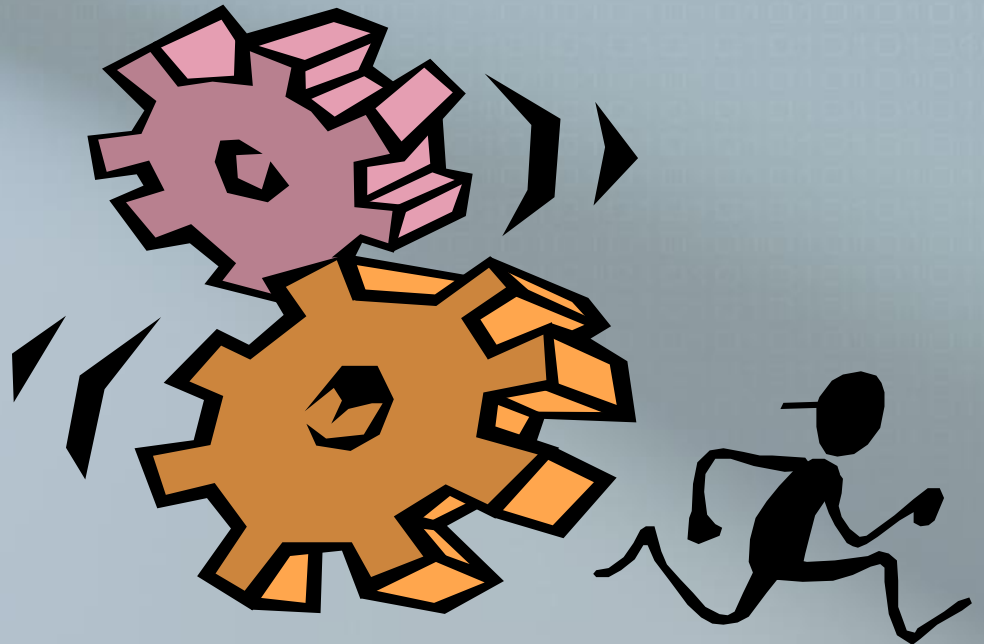
- Listes de sensibilité d'un process:
 - Système combinatoire:
TOUTES LES ENTREES du combinatoire
 - Système séquentiel:
HORLOGE + Reset/Preset ASYNCHRONE
- Ne pas tenter de mélanges:
 - Ne pas décrire un système séquentiel dans un process qui n'est pas déclenché par l'horloge
 - Et inversement...

C3

71

Règles générales

- Dans un process
 - Une variable est mise à jour IMMEDIATEMENT
 - Un signal est mis à jour EN SORTIE DU PROCESS

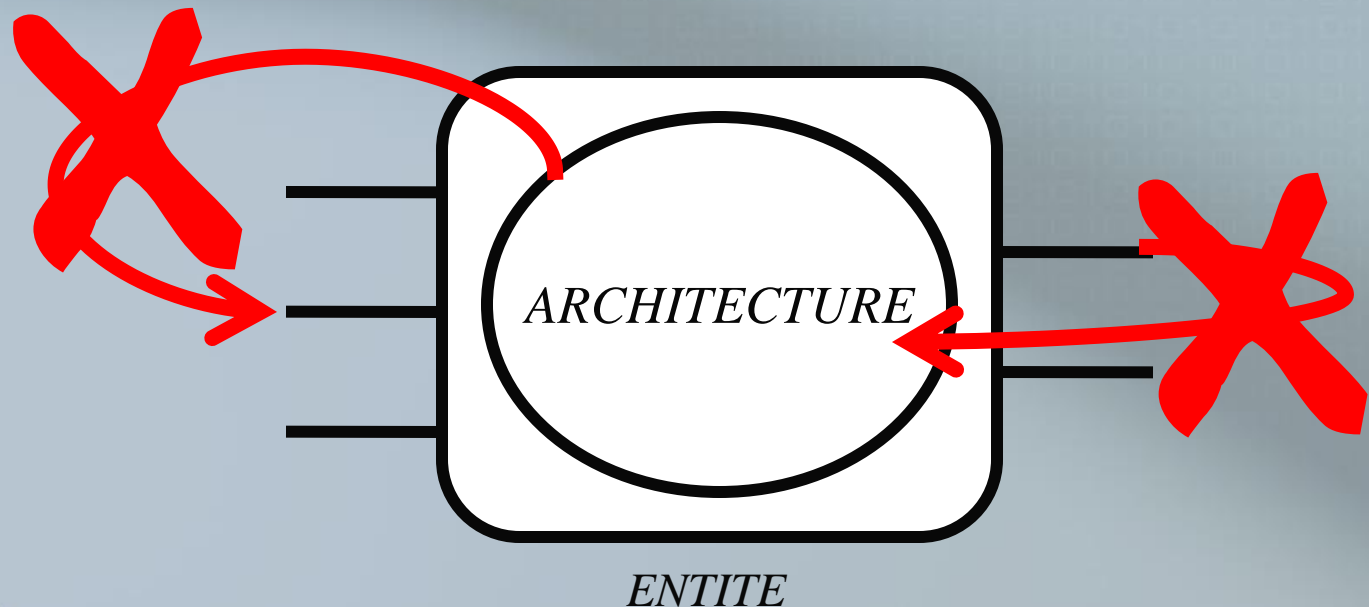


C3

72

Règles générales

- Ports d'entrée/sortie
 - On ne peut pas affecter (écrire) une valeur sur un port d'entrée
 - On ne peut pas lire la valeur d'un port de sortie pour l'affecter à un signal ou un autre port



C3

73

Bien décrire du combinatoire

- Les valeurs des sorties doivent être définies pour toutes les combinaisons possibles des entrées
- Pourquoi?
 - Sinon, l'outil de synthèse insère un élément de mémorisation (latch) qui souvent n'est pas synthétisable (notamment sur FPGA)

```
entity mux_4_1 is
port (a,b,c: in std_logic;
      com: in std_logic_vector(1 downto 0);
      s: out std_logic);
end mux_4_1;

architecture archi of mux_4_1 is
begin
    s <= a when com="00"
    else b when com="01"
    else c when com="10";
end archi;
```

*Que se passe-t-il
si com="11"?*

C3

74

Bien décrire du combinatoire

- Si le combinatoire est décrit dans un process, mettre toutes les entrées dans la liste de sensibilité
- Pourquoi?
 - La sortie d'un combinatoire est directement dépendante de la combinaison des entrées
 - Tout changement sur une entrée entraîne potentiellement un changement de la sortie
 - Si une entrée ne figure pas dans la liste du process, un changement de valeur sur cette entrée ne sera pas immédiatement détecté
-> Mauvaise description du comportement de la sortie

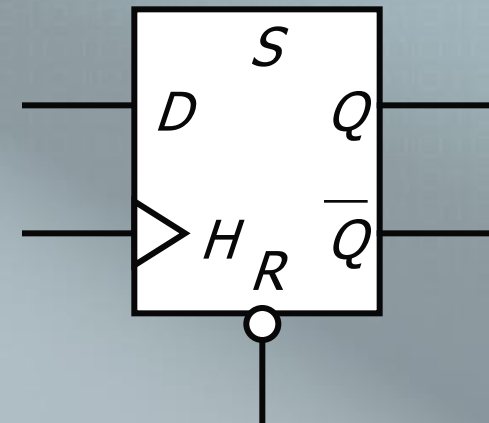
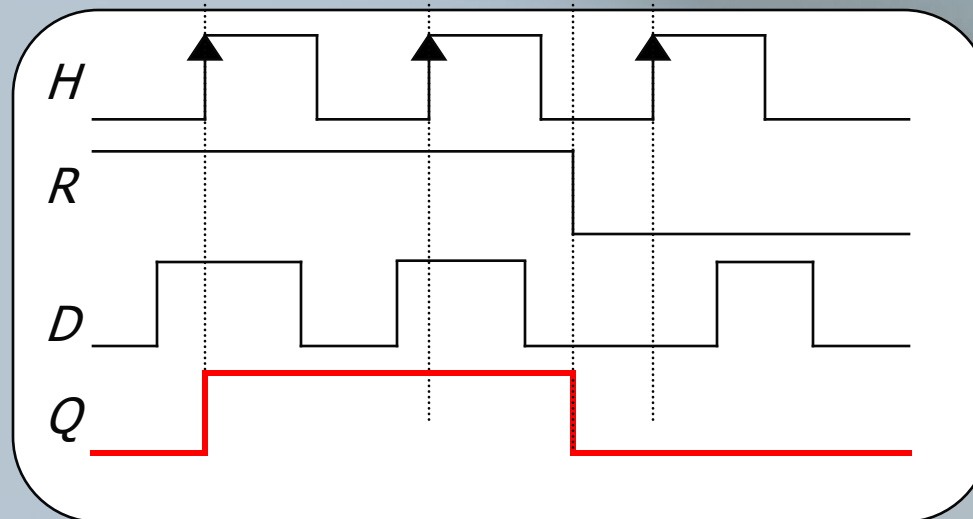


C3

75

Bien décrire du séquentiel

- La liste de sensibilité comprend **UNIQUEMENT** l'horloge et le reset asynchrone
- Pourquoi?
 - La sortie d'une bascule est modifiée lors d'un front d'horloge ou lors de l'activation du reset asynchrone
 - Le changement de son entrée n'entraîne pas **IMMEDIATEMENT** un changement en sortie



C3

76

Bien décrire du séquentiel

- A quoi sert le reset ASYNCHRONE?
 - A donner des valeurs aux bascules lors de l'init. du système
 - Il n'y a ainsi pas de valeurs non définies au démarrage
- Quelle différence avec un reset SYNCHRONE?
 - Reset asynchrone:
 - Initialisation globale du système
 - Gérée par l'utilisateur (ON/OFF)
 - Reset synchrone:
 - Réinitialisation d'une partie du système
 - Auto-gérée par le système



Bien décrire du séquentiel

- Quel est l'ordre des tests dans une description d'un système séquentiel?
 - Le RAZ asynchrone est prioritaire sur l'horloge
 - On le teste donc en premier
 - On teste ensuite le front d'horloge
 - On teste enfin les autres entrées de commande

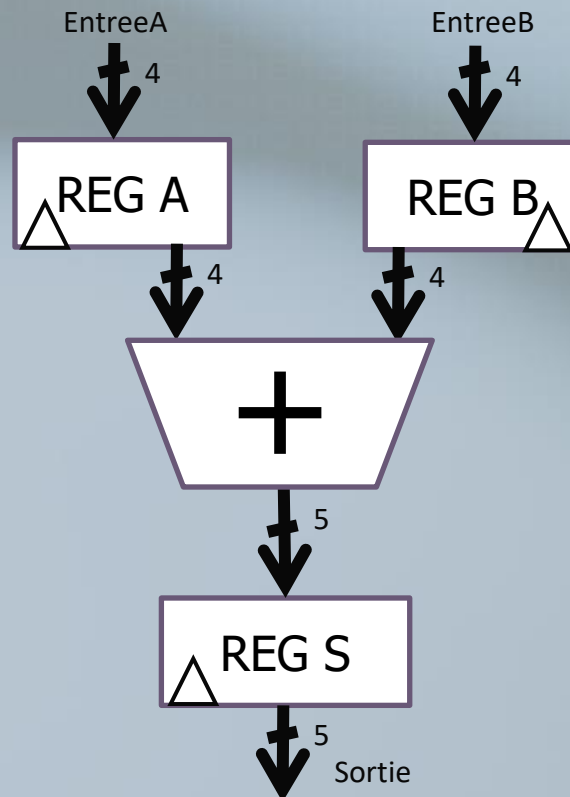
```
process (h, raz)
begin
    if raz='0' then registre<="0000";
    elsif rising_edge(h) then
        if load='1' then registre<=entree;
        else registre<="0011"; end if;
    end if;
end process;
```

C3

78

Bien combiner combinatoire et séquentiel

- Exemple d'un système mêlant combinatoire et séquentiel
 - Additionneur avec registres d'entrée et de sortie

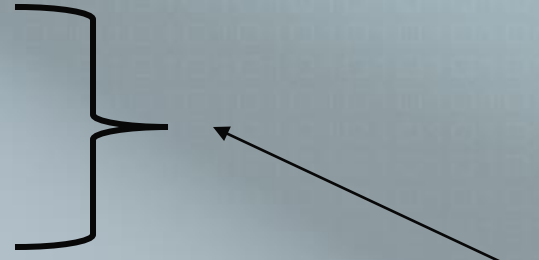


Bien combiner combinatoire et séquentiel

■ Ce qu'il ne faut pas faire

Décrire l'additionneur dans le même process que les registres (avec l'horloge comme liste de sensibilité)

```
process (h, raz)
begin
    if raz='0' then RegA<="0000"; RegB<="0000"; RegS<="00000";
    elsif rising_edge(h) then
        RegA <= EntreeA;
        RegB <= EntreeB;
        Adder <= '0' & RegA + RegB;
        RegS <= Adder;
    end if;
end process;
```



*A l'intérieur du test de l'horloge
tous les signaux à gauche d'une flèche
d'affectation sont des bascules/registres*

C3

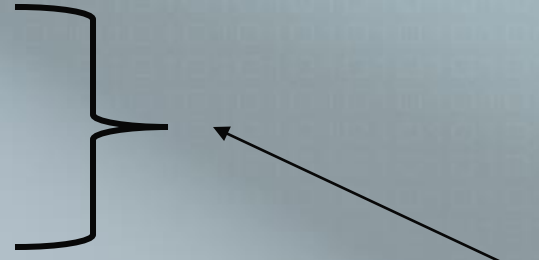
80

Bien combiner combinatoire et séquentiel

■ Ce qu'il ne faut pas faire

Décrire l'additionneur dans le même process que les registres (avec l'horloge comme liste de sensibilité)

```
process (h, raz)
begin
    if raz='0' then RegA<="0000"; RegB<="0000"; RegS<="00000";
    elsif rising_edge(h) then
        RegA <= EntreeA;
        RegB <= EntreeB;
        Adder <= '0' & RegA + RegB;
        RegS <= Adder;
    end if;
end process;
```



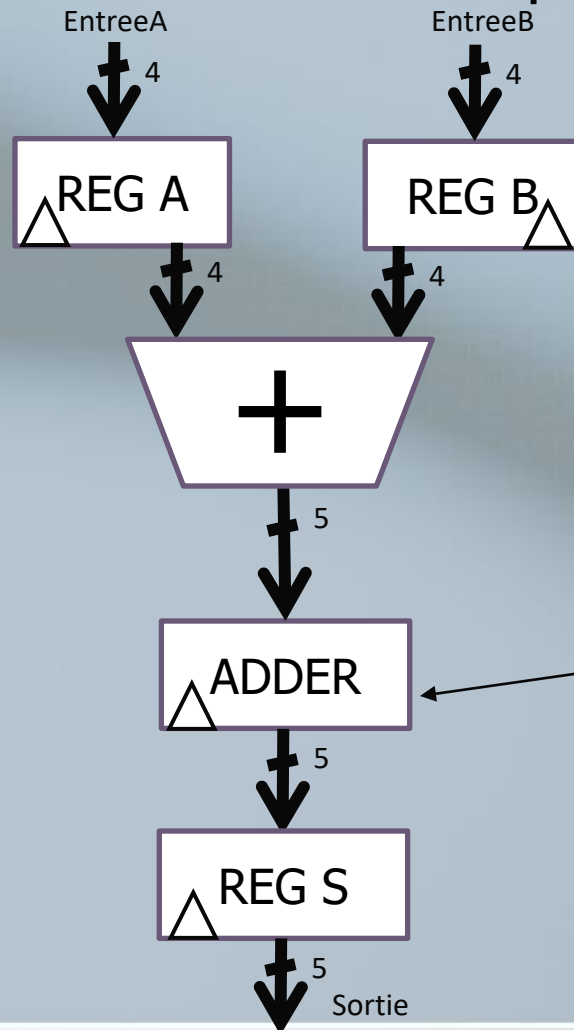
*On insère donc un registre supplémentaire en sortie de l'additionneur.
L'additionneur n'est plus combinatoire*

C3

81

Bien combiner combinatoire et séquentiel

- Architecture réalisée par le programme précédent



*Registre supplémentaire
à cause de la mauvaise description*

C3

82

Bien combiner combinatoire et séquentiel

- Ce qu'il faut faire
 - Décrire les parties séquentielles et les parties combinatoires dans des zones différentes du programme
 - Par exemple, décrire le séquentiel dans un ou plusieurs process sensible(s) sur l'horloge et le reset
 - Et décrire le combinatoire à l'aide d'instructions concurrentes

C3

83

Bien combiner combinatoire et séquentiel

■ Programme correct

```
-- ZONE COMBINATOIRE DU PROGRAMME
```

```
Adder <= '0' & RegA + RegB;
```

```
-- ZONE SEQUENTIELLE DU PROGRAMME (PROCESS)
```

```
process (h, raz)
```

```
  begin
```

```
    if raz='0' then RegA<="0000"; RegB<="0000";  
                      RegS<="00000";
```

```
    elsif rising_edge (h) then
```

```
      RegA <= EntreeA;
```

```
      RegB <= EntreeB;
```

```
      RegS <= Adder;
```

```
    end if;
```

```
  end process;
```

C3

84

Bien combiner combinatoire et séquentiel

■ Autre exemple: compteur

Mauvaise description d'un compteur (module séquentiel) dans un process combinatoire

```
process (start, entree)
  begin
    if start='0' then compteur<=entree;
    else compteur <= compteur + 1;
    end if;
  end process;
```

C3

85

Boucle combinatoire: l'incrémentation se répète indéfiniment tant que le test sur start ='1' est validé.
NON SYNTHETISABLE!!

Bien combiner combinatoire et séquentiel

■ Programme correct

Description dans un process sensible sur l'horloge

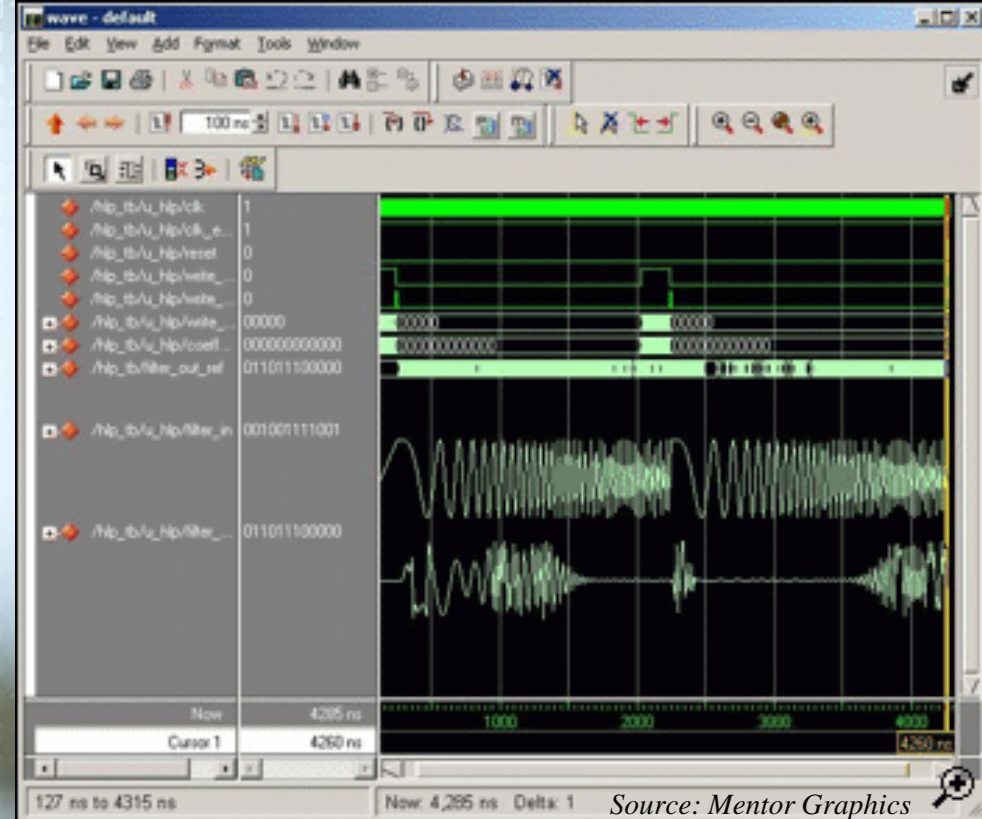
```
process (reset, h)
begin
    if reset='0' then compteur <= (others => '0');
    elsif rising_edge(h) then
        if start='0' then compteur <= entree;
        else compteur <= compteur+1;
        end if;
    end if;
end process;
```

Permet d'affecter tous les bits d'un vecteur à '0'

Grâce au test du front d'horloge, l'instruction d'incrément ne s'exécutera qu'UNE SEULE FOIS par cycle d'horloge

C3

86



Testbenchs, Simulation

Comment simuler efficacement son code VHDL

Simulation Manuelle

- La plupart des outils VHDL permet de simuler « à la main » votre code compilé
- Chaque étape de simulation s'effectue en trois temps
 - On donne une valeur aux entrées
 - On fait avancer le temps de simulation
 - On observe les valeurs des sorties
- Puis on itère ces trois étapes pour toutes les combinaisons possibles...

C3

88

Inconvénients Simu. Manuelle

- Si le système possède 32 bits en entrée
 - 4 294 967 296 combinaisons possibles à tester
 - Sans compter les séquences de valeurs...
- Conséquence
 - La simulation manuelle va être TRES longue
 - La probabilité que vous fassiez une erreur est TRES importante
- Conséquence d'une erreur
 - Il faut tout recommencer depuis le début...

C3

89

Comment échapper à cela?

■ Faire un Testbench

- Fichier dans lequel on va écrire les valeurs successives que vont prendre les entrées au cours de la simulation

■ Testbench en VHDL

- C'est un programme VHDL
- On y instancie le module à simuler
- Puis on y décrit le comportement des entrées

C3

95

Exemple Fichier Testbench

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

```
entity TestBench is  
end TestBench;
```

```
architecture simu of TestBench is
```

```
signal entree, sortie: std_logic_vector(3 downto 0);  
signal commande: std_logic;
```

```
begin
```

```
  L0: entity work.Module  
  port map (entree, commande, sortie);
```

```
  entree<="0100", "1001" after 100 ns, "1111" after 200 ns;  
  commande<='0', '1' after 40 ns, '0' after 120 ns,
```

```
end simu;
```

*Instanciation du composant
Module correspond à un
bloc VHDL décrit auparavant*

*Evolution des
signaux d'entrée*

C3

96

Autre Exemple

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

entity TestBench is
end TestBench;

architecture simu of TestBench is

    signal entree, sortie: std_logic_vector(3 downto 0);
    signal clk: std_logic := '0';

begin

    L0: entity work.Module
        port map (entree, clk, sortie);

    entree <= "0100", "1001" after 100 ns, "1111" after 200 ns;
    clk <= not clk after 100 ns;

end simu;
```

Valeur initiale
donnée au
signal d'horloge

Evolution
de l'horloge

C3

97

Utilisation du Testbench

- Compilation du module à tester
- Compilation du module testbench
- Lancement du simulateur
- On fait avancer le temps
 - Les entrées évoluent automatiquement
 - On observe le comportement des sorties
- Si on s'aperçoit qu'on a fait une erreur
 - On modifie la mauvaise instruction du fichier testbench
 - On recompile et on relance la simulation
- Gain de temps considérable

C3

98

Annexes

Codes VHDL COMPORTEMENTAL
de modules de base

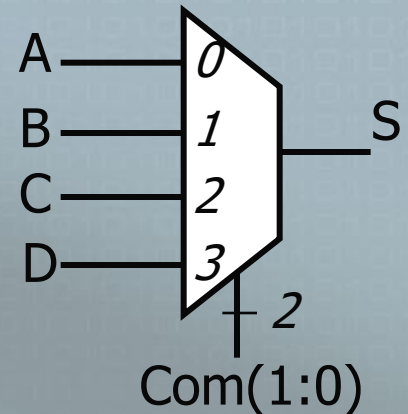


Multiplexeur

- Exemple de multiplexeur 4 → 1

```
entity mux_4_1 is
port (a,b,c,d: in std_logic;
      com: in std_logic_vector(1 downto 0);
      s: out std_logic);
end mux_4_1;
```

```
architecture archi of mux_4_1 is
begin
    s <= a when com="00"
    else b when com="01"
    else c when com="10"
    else d;
end archi;
```



C3

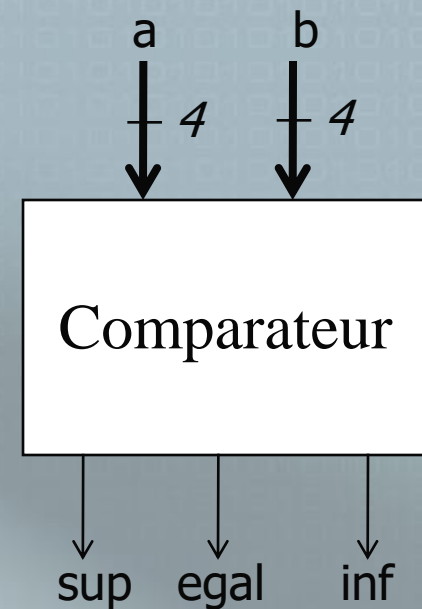
100

Comparateur

■ Exemple avec entrées sur 4 bits

```
entity compareur is
port (a,b: in std_logic_vector(3 downto 0);
      egal,sup,inf: out std_logic);
end compareur;
```

```
architecture archi of compareur is
begin
    egal <= '1' when a=b else '0';
    sup  <= '1' when a>b else '0';
    inf  <= '1' when a<b else '0';
end archi;
```



Voir les détails de l'architecture dans le poly précédent...

C3

101

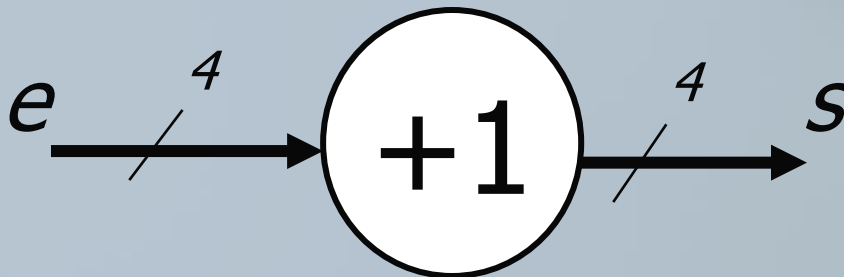
Incrémenteur

```
entity incrementeur is
port(e: in std_logic_vector(3 downto 0);
      s: out std_logic_vector(3 downto 0));
end incrementeur;
```

```
architecture archi of incrementeur is
begin
```

```
    s <= e+1;
```

```
end archi;
```



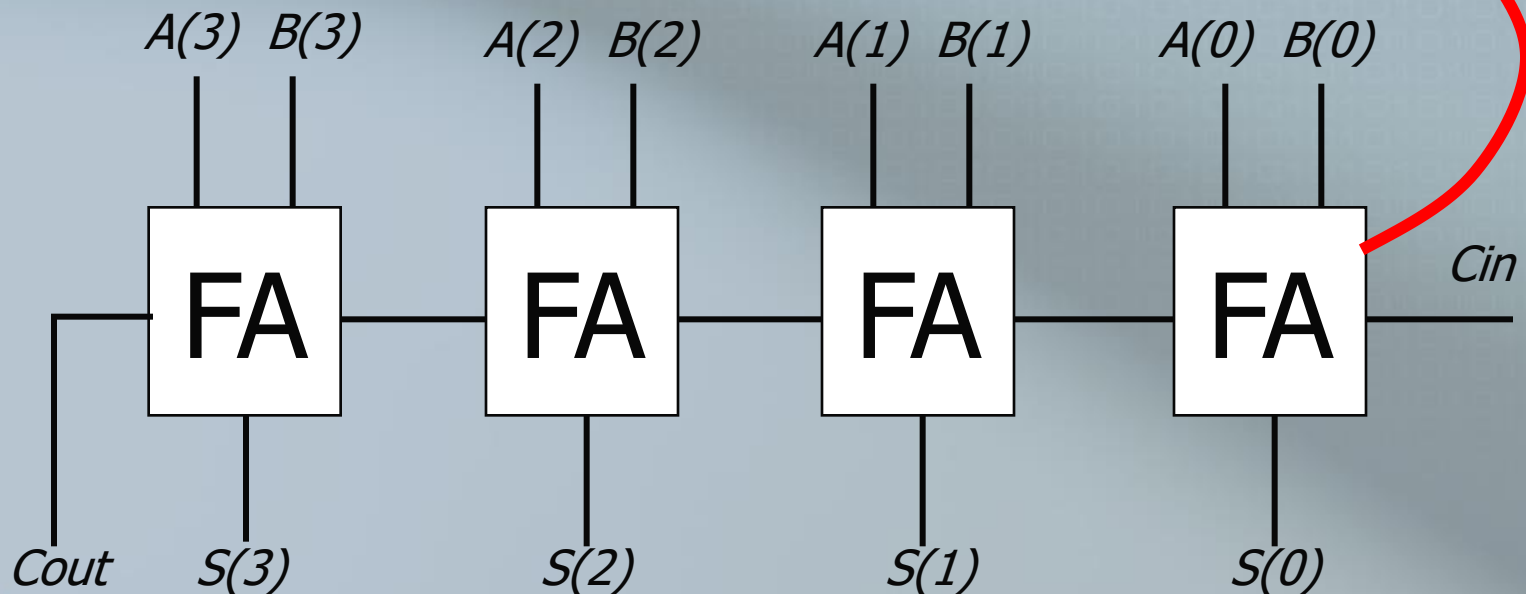
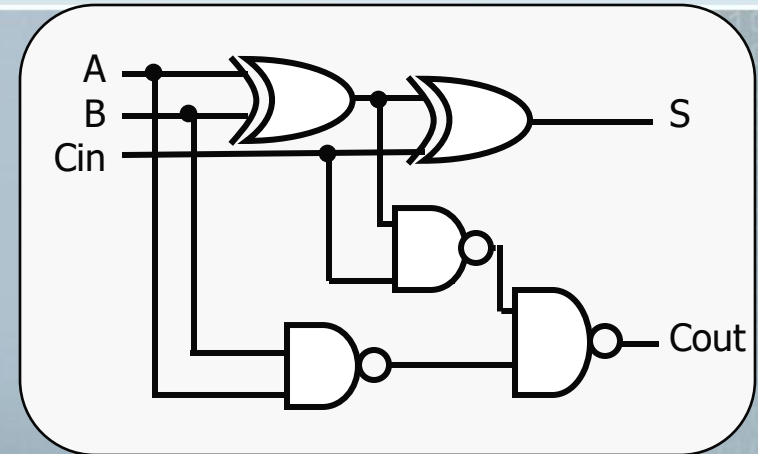
*Expression utilisable
grâce au package
std_logic_unsigned*

C3

102

Additionneur

■ Schéma portes



C3

103

Additionneur

■ Avec ou sans retenue sortante

```
entity adder is
port (
  a,b: in std_logic_vector(3 downto 0);
  cin: in std_logic;
  s1: out std_logic_vector(3 downto 0);  -- sans retenue
  s2: out std_logic_vector(4 downto 0)); -- avec retenue
end adder;

architecture archi of adder is
begin
  s1 <= a+b+cin;      -- Addition sans retenue sortante
  s2 <= '0'&a+b+cin;-- Addition avec retenue sortante
end archi;
```


Registre simple

```
entity Registre is
port( h,raz: in std_logic;
      entree: in std_logic_vector(3 downto 0);
      sortie: out std_logic_vector(3 downto 0));
end Registre;
```

```
architecture archi of Registre is
begin
```

```
  process(h,raz)
```

```
  begin
```

```
    if raz='0' then
```

```
      sortie<=(others => '0');
```

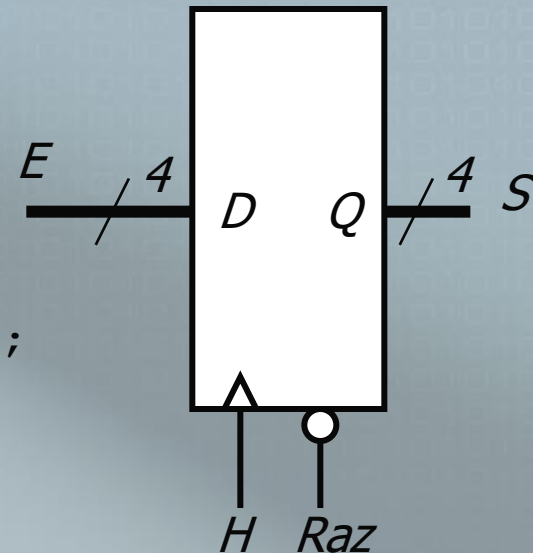
```
    elsif rising_edge(h) then
```

```
      sortie <= entree;
```

```
    end if;
```

```
  end process;
```

```
end archi;
```

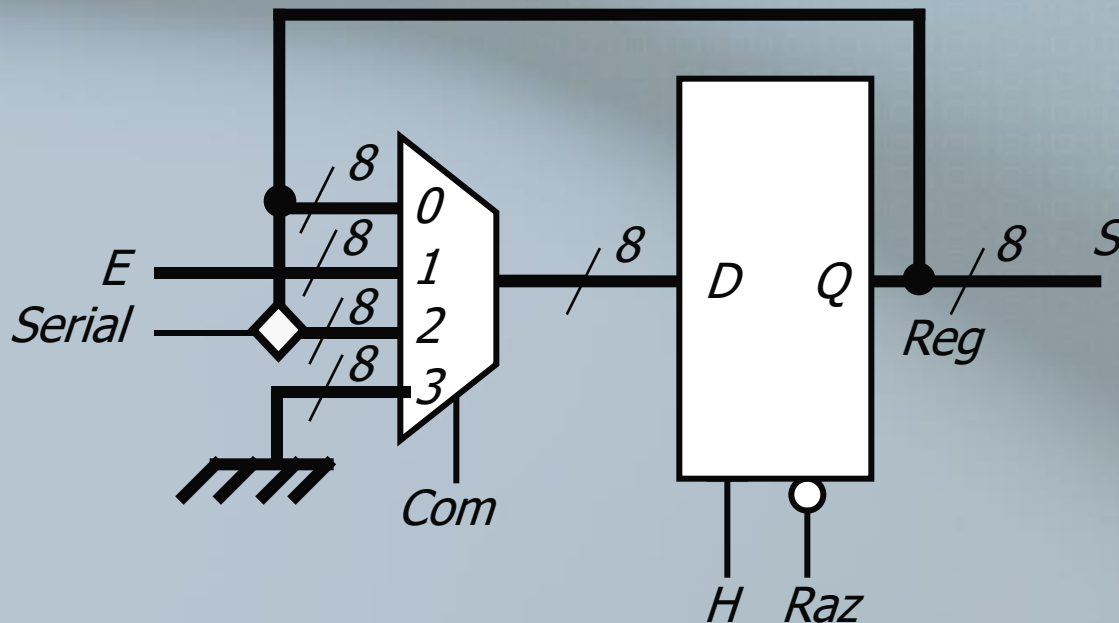


C3

105

Registre multifonctions (1/2)

```
entity Registre is
port(  h,raz,serial: in std_logic;
      com: in std_logic_vector(1 downto 0);
      E: in std_logic_vector(7 downto 0);
      S: out std_logic_vector(7 downto 0));
end Registre;
```



Fonction	Com
Mémo	0
Chgt //	1
Decal →	2
RAZ Sync	3

C3

106

Registre multifonctions (2/2)

```
architecture archi of Registre is
  signal reg: std_logic_vector(7 downto 0);
begin

  S <= reg; -- CONNECTION DU REGISTRE AU PORT DE SORTIE

  process(h,raz)
  begin

    if raz='0' then reg<=(others => '0'); -- RAZ ASYNCHRONE
    elsif rising_edge(h) then

      case(com) is

        when "01" => reg<=E; -- Chgt //
        when "10" => reg<=serial & reg(7 downto 1); -- DECALAGE
        when "11" => reg<=(others => '0'); -- RAZ SYNCHRONE

      end case;
    end if;
  end process;
end archi;
```

C3

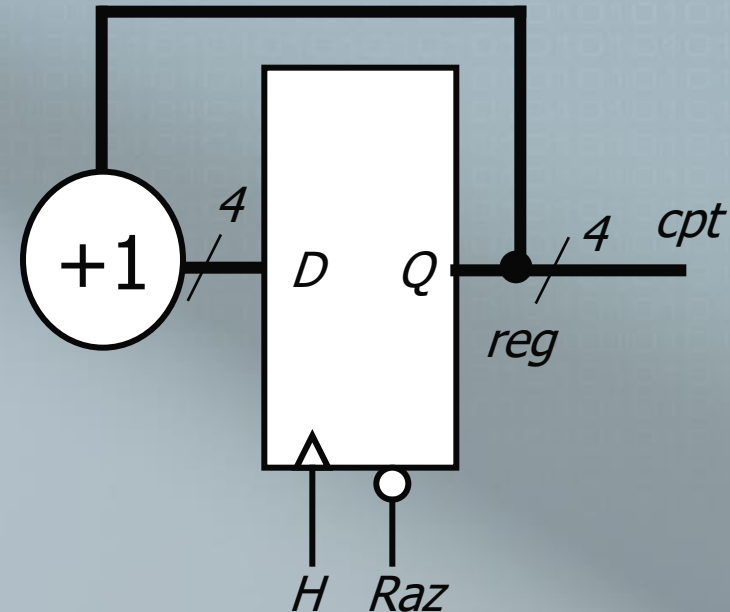
107

Compteur simple

```
entity Compteur is
port(  h,raz: in std_logic;
      cpt: out std_logic_vector(3 downto 0));
end Compteur;
```

```
architecture archi of Compteur is
signal reg: std_logic_vector(3 downto 0);
```

```
begin
  cpt <= reg;
  process(h,raz)
  begin
    if raz='0' then
      reg <= (others => '0');
    elsif rising_edge(h) then
      reg <= reg+1;
    end if;
  end process;
end archi;
```

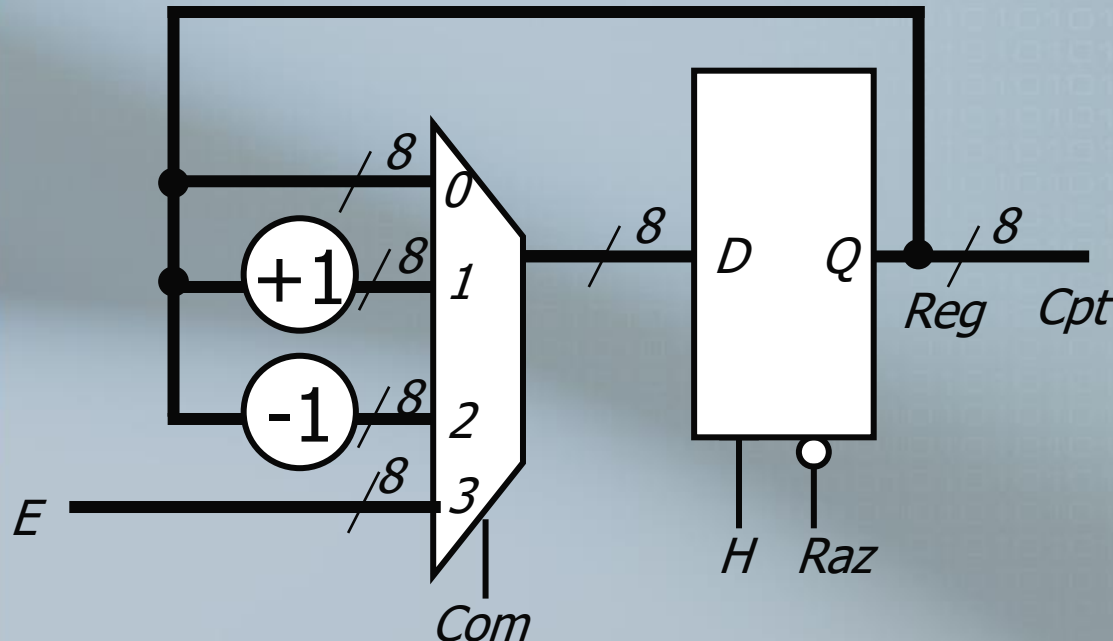


C3

108

Compteur multifonctions (1/2)

■ Schéma Bloc



Fonction	Com
Mémo	0
Increment	1
Decrement	2
Chargement	3

C3

```
entity Compteur is
port(  h,raz:  in std_logic;
      com:    in std_logic_vector(1 downto 0);
      E:      in std_logic_vector(7 downto 0));
      cpt: out std_logic_vector(7 downto 0));
end Compteur;
```

109

Compteur multifonctions (2/2)

```
architecture archi of Compteur is
    signal reg: std_logic_vector(7 downto 0);

begin
    cpt <= reg;
    process(h, raz)
    begin
        if raz='0' then reg <= (others => '0');
        elsif rising_edge(h) then

            case(com) is

                when "01" => reg<=reg+1; -- Incrementation
                when "10" => reg<=reg-1; -- Decrementation
                when "11" => reg<=E;      -- Chargement //

            end case;
        end if;
    end process;
end archi;
```

C3

110