Cours VHDL – I

L3-S6 - Université de Cergy-Pontoise

Laurent Rodriguez – Benoît Miramond

Plan du cours

I – Historique de conception des circuits intégrés

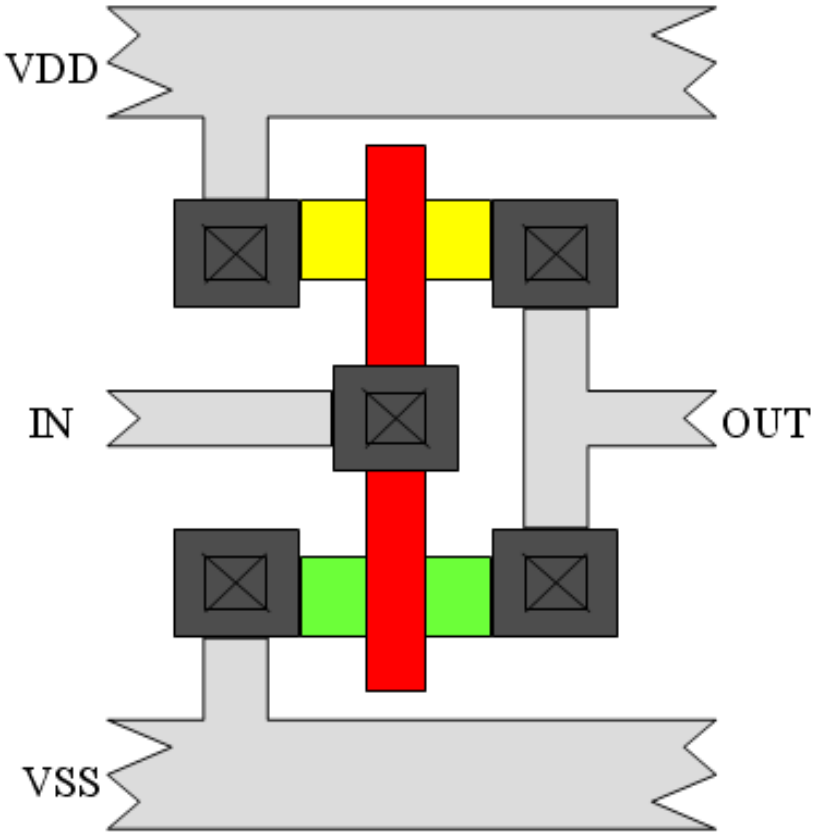
- **HDL**
  - Modèles de conceptions
- **VHDL**
  - Les modèles de conceptions en VHDL
  - Les 5 briques de base

II – VHDL et FPGA

- **VHDL**
  - Syntaxe et typage
  - Retour sur les briques de bases
  - Retour sur la conception structurelle et comportementale en VHDL
    Port map, Equations logiques, Tables de vérités (With ... Select)
- **FPGA**
  - Situer le FPGA
  - Qu'est ce qu'un FPGA
  - Flot de conception
  - Carte de développement et environnement de TP

I – HDL : Hardware Description Language - Historique

A) Dessin au micron



A la main,
au niveau transistor,
avec des surfaces rectangulaires !

=> Complexité des circuits limitée
+ technologie des fondeurs

I – HDL : Hardware Description Language - Historique

B) Les langages de descriptions

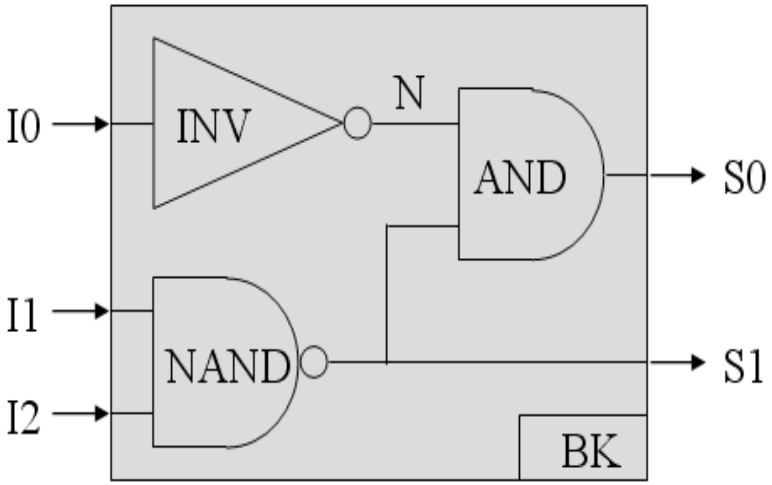
```
bloc BK(I0, I1, I2 : in;
        S0, S1 : out)

  noeud N;

  {
  inv(I0, N);
  nand(I1, I2, S1);
  and(N, S1, S0);
  }
```

Modélisation/Simulation/Conception

Outils placeurs-routeurs :
Desc. Textuelle (Structure)
+ bibliothèque de composants
=> Dessin au micron ou simulation

I – HDL : Hardware Description Language - Historique

C) Descriptions schématique



Arrivée des interfaces graphiques
=> éditeurs de schémas

+ ergonomie
- portable
- maintenable
- archivable

I – HDL : Hardware Description Language - Historique

D) Abstraction fonctionnelle (ou comportementale)

Synthétiseur logique : le « Quoi » plutôt que le « comment »
La fonction et non plus (seulement) la structure

```
bloc BK(I0, I1, I2 : in;
        S0, S1 : out)

  {
  S0 = non(I0 ou (I1 et I2));
  S1 = non(I1) ou non(I2);
  }
```

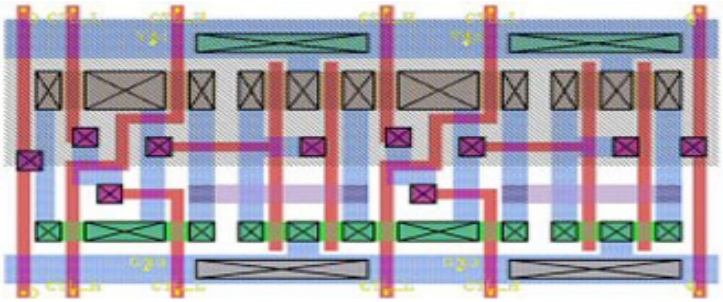
Niveau d'abstraction plus élevé :

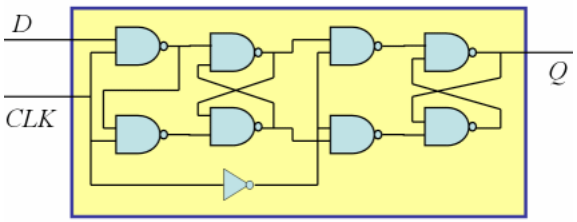
- Réduction du temps de conception
- Efficacité des simulations
- Normalisation des échanges
- Anticipation
- Fiabilité
- Portabilité
- Maintenabilité & Réutilisabilité

I – HDL : Hardware Description Language – Modèles de descriptions

On distingue donc plusieurs niveaux de modélisation (modèles ou encore vues) :

- Physiques
  - Dimensions
  - Matériaux
  - Transistors
  - Masques, ...
- Structurelles
  - Assemblage de composants
  - Hiérarchie d'interconnexions de différents sous-ensemble dont le plus bas niveau est le transistor
- Comportementales
  - Fonction réalisée
  - Le « Quoi » et non le « Comment »

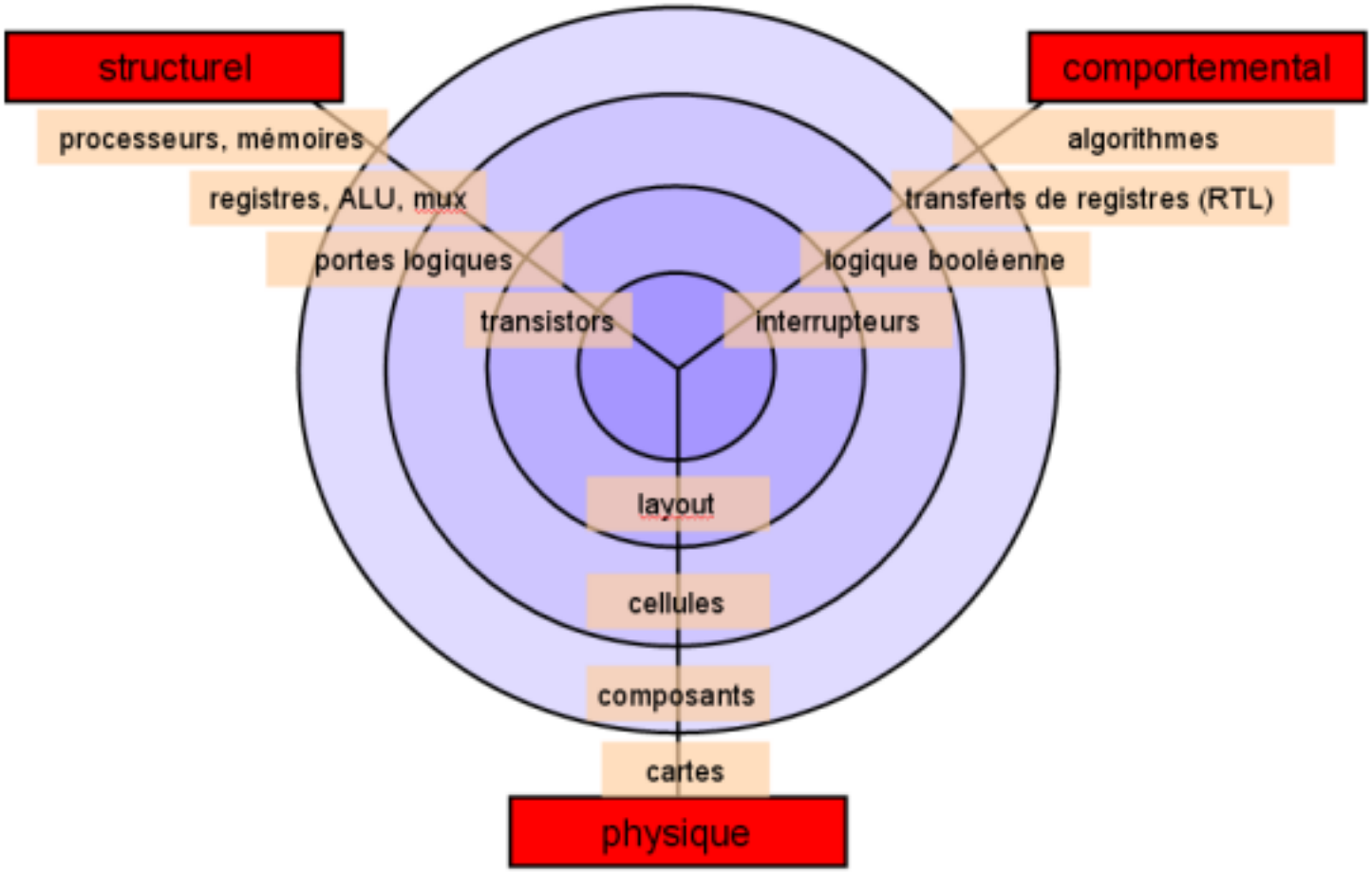





```
bloc BK(I0, I1, I2 : in;
        S0, S1 : out)

  {
  S0 = non(I0 ou (I1 et I2));
  S1 = non(I1) ou non(I2);
  }
```

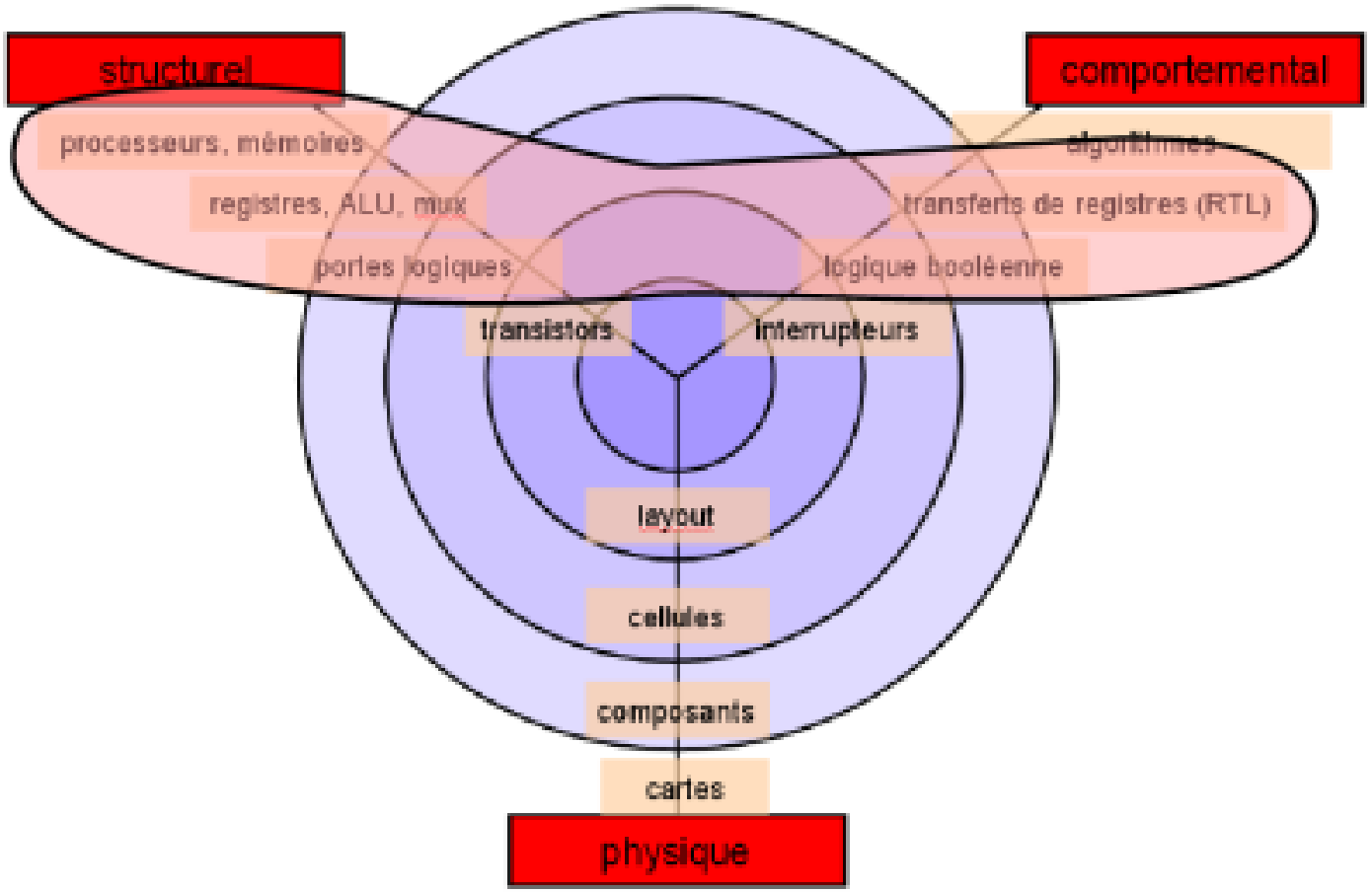
I – HDL : Hardware Description Language – Modèles de descriptions

Degrés (finesse) de modélisation



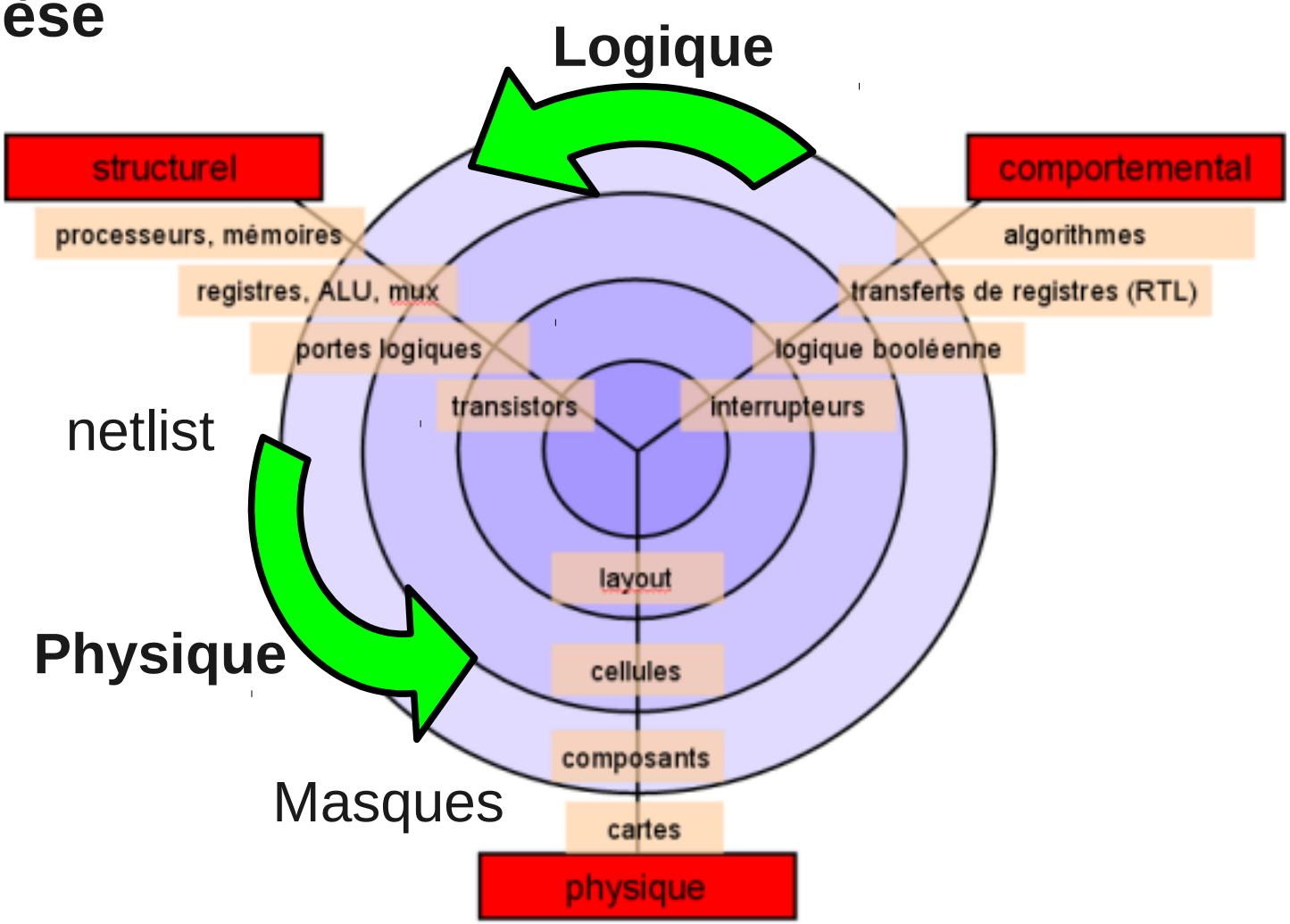
I – HDL : Hardware Description Language – Modèles de descriptions

Degrés (finesse) de modélisation du VHDL



I – HDL : Hardware Description Language – Modèles de descriptions et Synthèse

Synthèse



I – HDL : Hardware Description Language – Synthèse

La synthèse logique

Transforme le code en une représentation structurelle de bas niveau (netList) utilisant les cellules de la bibliothèque de la technologie visée (fondeur ou FPGA).
Ces modèles sont souvent au format VITAL pour VHDL (VHDL Initiative Towards VHDL Libraries) pour permettre la rétroannotation des délais (des portes).

La synthèse physique

Transforme une représentation structurelle de bas niveau en une description physique du circuit (layout).
Elle nécessite une étape supplémentaire de placement et de routage des portes.
Le format d'entrée des outils diffère du VHDL.
Les formats EDIF ou XNF sont souvent utilisés.
On peut alors connaître les délais complets du aux interconnexions, et les stockés dans un fichier au format SDF (Standard Delay Format) pour rétroannoté le code VHDL.

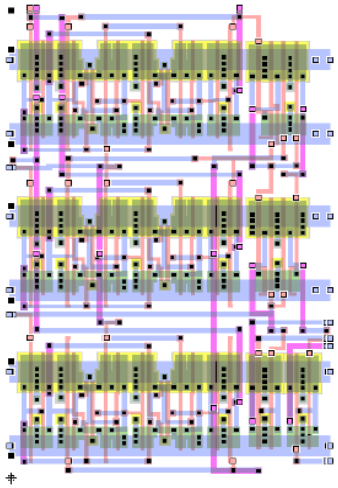
I – HDL : Hardware Description Language

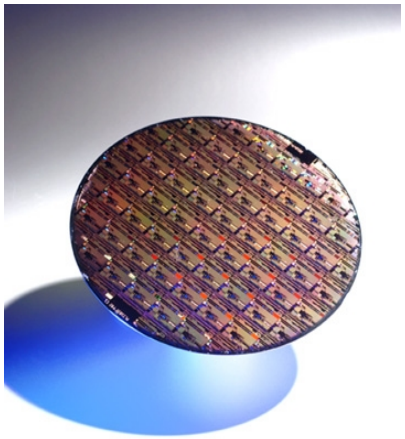
Simulation et/ou preuve formelle

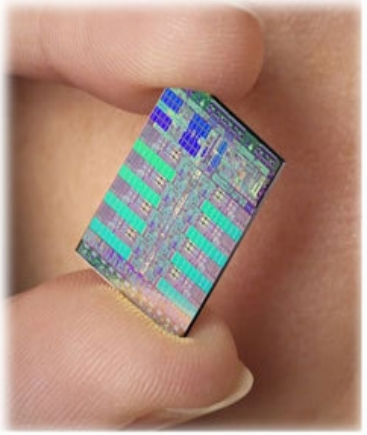
L'utilisation d'un HDL permet de décrire un système matériel et de le simuler => un modèle VHDL est un modèle exécutable.
Il est possible de lui appliquer des stimuli et d'observer l'évolution des signaux dans le temps grâce à un simulateur discret évènementiel

Réalisation du circuit

Le langage est aussi utilisé comme format de description d'entrée de la « synthèse »
La synthèse est l'étape de traduction préalable à la réalisation physique du circuit







I – HDL : Hardware Description Language

- Description formelle d'un système électronique.
- Fonctionnement d'un circuit
- Structure d'un circuit
- Vérification -> Simulation, vérification formelle

Un grand nombre de HDLs :

- Verilog
- VHDL - VHDL-AMS
- SystemC/C++ - SystemC-AMS
- Confluence (C => Verilog ou VHDL)
- AHDL (Altera)
- ...

Texte => Comportement temporel ou structurel d'un circuit

Langage matériel Vs Langages de programmation classiques => Notions de **concurrence** et de **temps**

Le but des HDL : **Simulation** et **Synthèse**

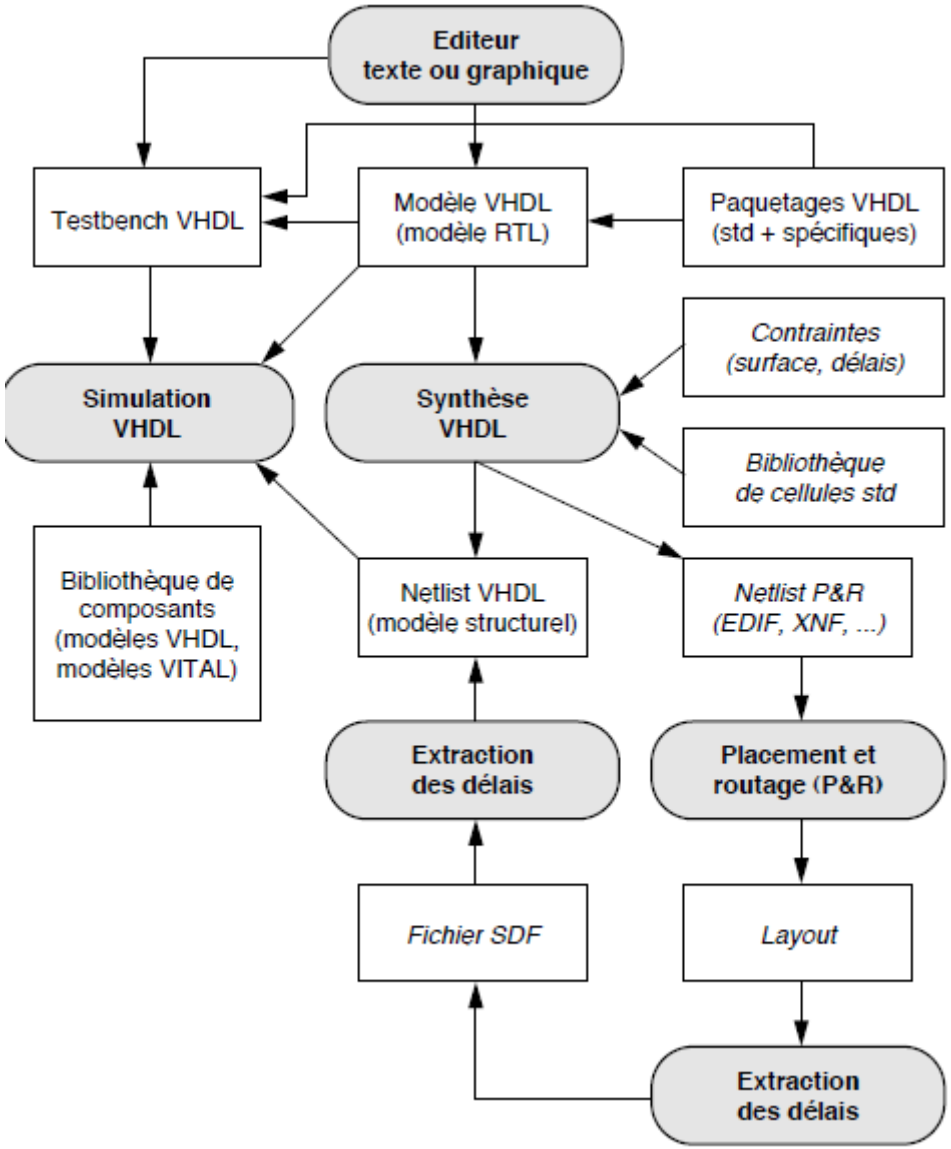
Synthèse Vs Compilation => Transformer le code HDL en un langage décrivant le circuit (portes logiques interconnectées) : **Netlist**

-> utilisation d'un sous-langage de l'HDL utilisé dit synthétisable (typiquement ce qui n'a pas trait aux notions de temps).

VHDL

(Very High Speed Integrated Circuit - Hardware Description Language)

II – VHDL : Flot de conception V(HDL)



II – VHDL : Structure du langage

Unités de conception

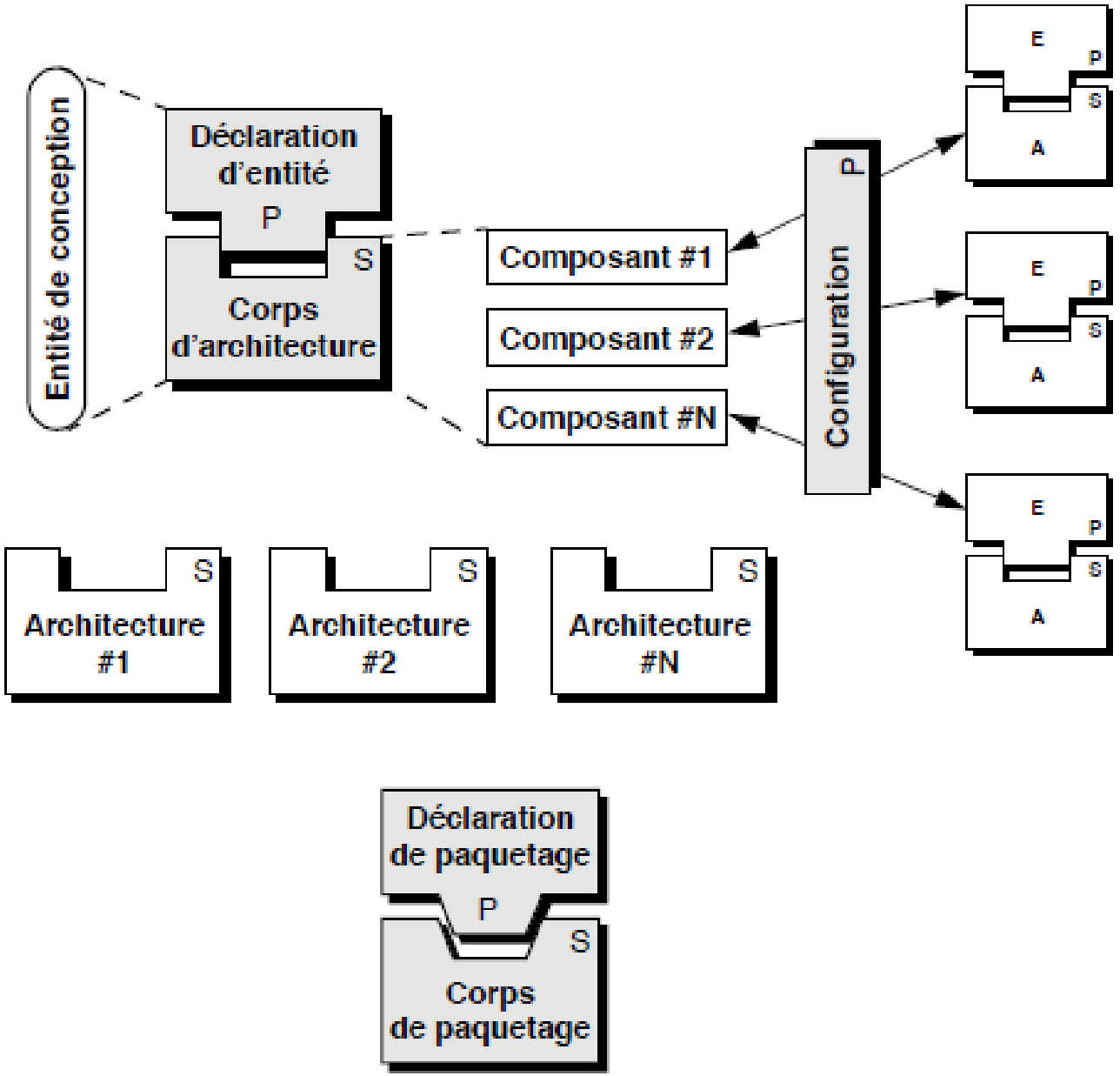
Description d’un modèle

- Déclaration d’entités, interface / conteneur
- Corps d’architecture, contenu
- Déclaration de configuration, contenu

Description d’une librairie

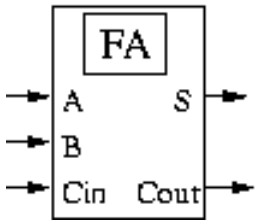
- Déclaration de paquetage
- Corps de paquetage

II – VHDL : Unités de conception



II – VHDL : L'entité

L'entité est la description de l'interface du circuit . Elle correspond au symbole dans les représentations schématiques :



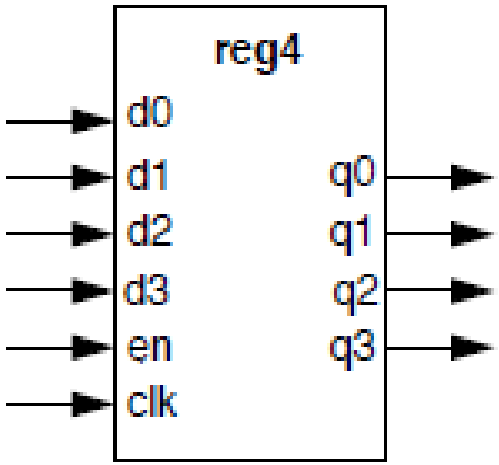

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fa is
 port (
 a, b, cin : in std_logic;
 s, cout : out std_logic
 );
end entity;
```

L'entité précise :

- le nom du circuit
- Les ports d'entrée-sortie :
  - Leur nom
  - Leur direction (in, out, inout,...)
  - Leur type (bit, bit_vector, integer, std_logic,...)
- Les paramètres éventuels pour les modèles génériques

II – VHDL : L'entité – exemple : registre 4 bits

```
Entity reg4 is
    port (
        d0, d1, d2, d3 : in BIT; -- données entrantes
        en, clk : in BIT; -- signaux entrants
        q0, q1, q2, q3 : out BIT; -- données sortantes
    );
End entity reg4;
```



II – VHDL : Architecture – description des modules

L'architecture est la description interne du circuit.

- Elle est toujours associée à une entité.
- Une même entité peut avoir plusieurs architecture.

Le mécanisme de **configuration** (décrit dans le VHDL structurel) permet d'indiquer l'architecture rattachée à une entité.

L'exemple suivant montre un schéma de l'additionneur 1 bit fa et 2 architectures possibles écrites en VHDL :

```
architecture arc1 of fa is
 signal resultat : unsigned(1 downto 0);
begin
 resultat <= ('0' & a) + ('0' & b) + ('0' & cin);
 s <= resultat(0);
 cout <= resultat(1);
end arc1;
```

```
architecture arc2 of fa is
begin
 s <= a xor b xor cin;
 cout <= (a and b) or ((a xor b) and cin);
end arc1;
```

II – VHDL : Architecture – Type de descriptions

Comportementale

Ce type correspond à expliciter le comportement d'un modèle par ses équations

Structurelle

Ce type correspond à l'instanciation hiérarchique d'autres composants

II – VHDL : Architecture – Exemple d'architecture comportementale

```
architecture bhv of reg4 is
begin
    process is
        variable d0_reg, d1_reg, d2_reg, d3_reg: BIT;
    begin
        if en = '1' and clk = '1' then
            -- mémorisation des entrées
            d0_reg := d0;
            d1_reg := d1;
            d2_reg := d2;
            d3_reg := d3;
        end if;
        -- modification des signaux de sortie
        q0 <= d0_reg after 5 ns;
        q1 <= d1_reg after 5 ns;
        q2 <= d2_reg after 5 ns;
        q3 <= d3_reg after 5 ns;
        -- attente du prochain événement sur l'un des signaux d'entrée
        wait on d0, d1, d2, d3, en, clk;
    end process;
end architecture bhv;
```

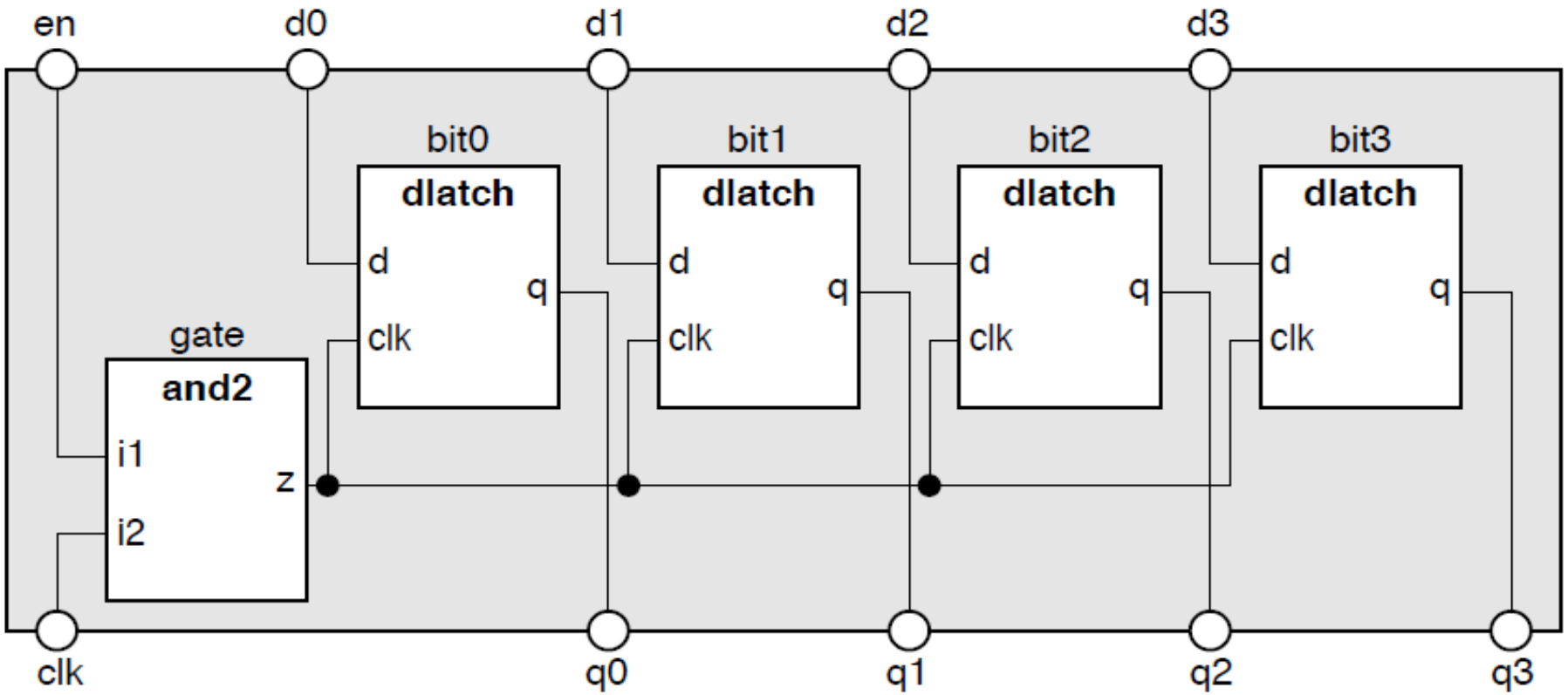
II – VHDL : Architecture – Exemple d'architecture comportementale

- Tous les objets déclarés dans l’entité sont visibles dans l’architecture
- Le processus définit une séquence d’instructions qui, dans ce cas
  - Mémorisent les signaux d’entrée de manière conditionnelle
  - Modifient systématiquement les signaux de sortie
  - Mettent le processus en veille jusqu’à un nouvel évènement
- Les variables (typées) sont initialisées par défaut à zéro et conservent leur valeur d’une activation à l’autre

II – VHDL : Architecture – Architecture structurelle

- L'architecture str de l'exemple fait référence à la même déclaration d'entité que le modèle comportemental précédent
- Elle est composée de 2 parties :
  - La déclaration des composants utilisés
  - L'instanciation des composants et leur interconnexion
- On peut utiliser une connexion
  - explicite par nom (latch)
  - Ou implicite par position (and2)
- Chaque instance a sa propre étiquette unique
- Une étape supplémentaire de configuration sera nécessaire pour associer une entité à chaque instance de composant utilisé dans le modèle structurel

II – VHDL : Architecture –
Exemple d'architecture structurelle



II – VHDL : Architecture – Exemple d'architecture structurelle

```
architecture str of reg4 is

    -- déclaration des composants utilisés
    component and2 is
        port (i1, i2: in BIT; z: out BIT);
    end component and2;

    component dlatch is
        port (d, clk: in BIT; q: out BIT);
    end component dlatch;

    signal int_clk: BIT;   -- horloge interne

begin
    -- instances de latches
    bit0: dlatch port map (d => d0, clk => int_clk, q => q0);
    bit1: dlatch port map (d => d1, clk => int_clk, q => q1);
    bit2: dlatch port map (d => d2, clk => int_clk, q => q2);
    bit3: dlatch port map (d => d3, clk => int_clk, q => q3);
    -- génération de l'horloge interne
    gate: and2 port map (en, clk, int_clk);
end architecture str;
```

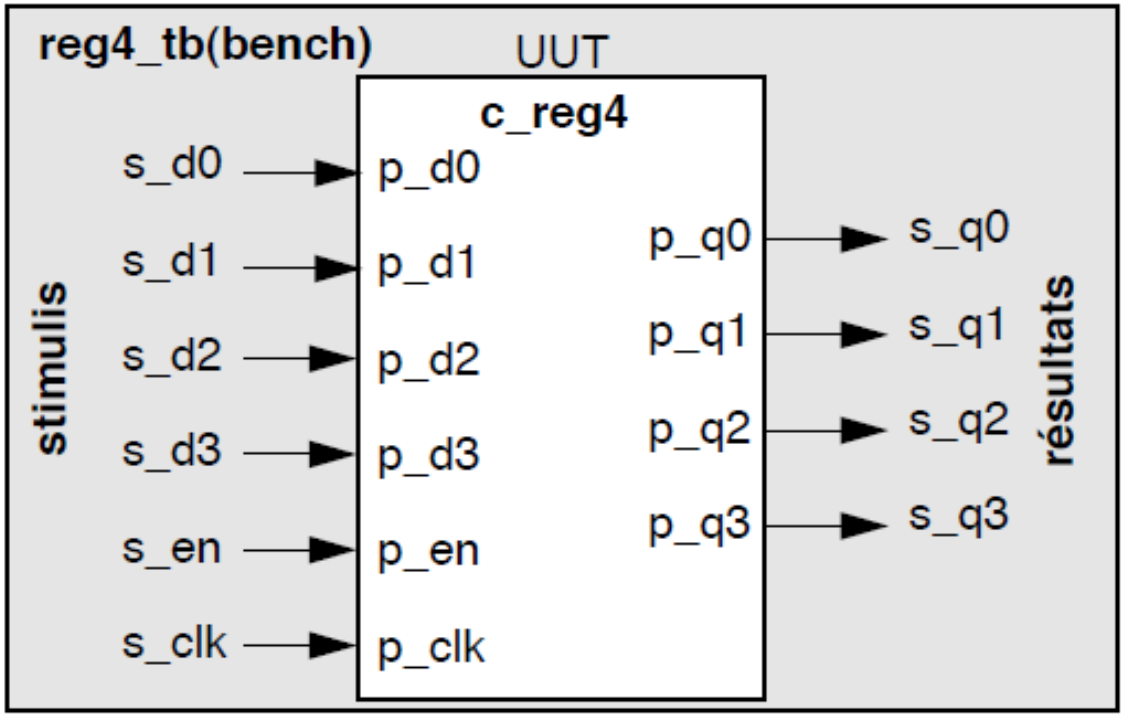
II – VHDL : Architecture –
Les composants de base de la bibliothèque GATES

```
entity dlatch is
    port (d, clk: in  BIT;
          q     : out BIT);
end entity dlatch;

architecture bhv of dlatch is
begin
    process is
    begin
        if clk = '1' then
            q <= d after 2 ns;
        end if;
        wait on clk, d;
    end process;
end architecture bhv;
```

```
entity and2 is
    port (a, b: in   BIT;
          z    : out BIT);
end entity and2;

architecture bhv of and2 is
begin
    process is
    begin
        z <= a and b after 2 ns;
        wait on a, b;
    end process;
end architecture bhv;
```

II – VHDL : Environnement de test – Test Bench

- Il faut instancier le composant à tester
- Puis lui envoyer un certain nombre de stimulis qui valident le fonctionnement
- Le modèle VHDL du testbench est donc à la fois comportemental et structurel



II – VHDL : Environnement de test – Test Bench Code

```
entity reg4_tb is
end entity reg4_tb;

architecture bench of reg4_tb is

    component c_reg4 is
        port (p_d0, p_d1, p_d2, p_d3, p_en, p_clk: in BIT;
              p_q0, p_q1, p_q2, p_q3: out BIT);
    end component c_reg4;

    signal s_d0, s_d1, s_d2, s_d3, s_en, s_clk, s_q0, s_q1, s_q2, s_q3: BIT;
begin
    -- composant à tester
    UUT: c_reg4 port map (p_d0 => s_d0, p_d1 => s_d1, p_d2 => s_d2, p_d3 => s_d3,
                          p_en => s_en, p_clk => s_clk,
                          p_q0 => s_q0, p_q1 => s_q1, p_q2 => s_q2, p_q3 => s_q3);
    -- stimulis
    s_clk <= not s_clk after 20 ns;  -- période de 40 ns
    process
    begin
        s_en <= '0';
        s_d0 <= '1'; s_d1 <= '1'; s_d2 <= '1'; s_d3 <= '1';
        wait for 40 ns;
        s_en <= '1';
        wait for 40 ns;
        s_d0 <= '0'; s_d2 <= '0';
        wait for 40 ns;
        s_en <= '0'; s_d1 <= '0'; s_d3 <= '0';
        wait for 40 ns;
        s_en <= '1';
        wait; -- stop définitif
    end process;
end architecture bench;
```

II – VHDL : Configuration d'un modèle

- Elle définit les associations entre les composants instanciés dans un modèle et les entités de conception disponibles dans la bibliothèque
- La partie 'port map' associe les noms de ports de l'entité à ceux utilisés dans la déclaration du composant s'ils sont différents, sinon elle est inutile.

```
library GATES;
configuration reg4_tb_cfg_str of reg4_tb is
    for test
        for UUT: c_reg4 use entity WORK.reg4(str)
            port map (p_d0, p_d1, p_d2, p_d3, p_en, p_clk,
                      p_q0, p_q1, p_q2, p_q3);
            for str
                for all: dlatch use entity GATES.dlatch(bhv); end for;
                for all: and2   use entity GATES.and2(bhv)
                         port map (a => i1, b => i2, z => z);
                end for; -- and2
            end for; -- str
        end for; -- UUT
    end for; -- test
end configuration reg4_tb_cfg_str;
```

II – VHDL : Configuration de l'environnement de test

Utilisation du reg4 structurel

```
architecture bench of reg4_tb is
    component c_reg4 is
        port (p_d0, p_d1, p_d2, p_d3, p_en, p_clk: in BIT;
              p_q0, p_q1, p_q2, p_q3: out BIT);
    end component c_reg4;
    ...
begin
    -- composant à tester
    UUT: c_reg4 port map (p_d0 => s_d0, p_d1 => s_d1, p_d2 => s_d2, p_d3 => s_d3,
                          p_en => s_en, p_clk => s_clk,
                          p_q0 => s_q0, p_q1 => s_q1, p_q2 => s_q2, p_q3 => s_q3);
    ...
end architecture bench;

configuration reg4_tb_cfg_bhv of reg4_tb is
    for test
        for UUT: c_reg4 use entity WORK.reg4(bhv)
            port map (d0 => p_d0, d1 => p_d1, d2 => p_d2, d3 => p_d3,
                      en => p_en, clk => p_clk,
                      q0 => p_q0, q1 => p_q1, q2 => p_q2, q3 => p_q3);
        end for; -- UUT
    end for; -- test
end configuration reg4_tb_cfg_bhv;
```

```
library GATES;
configuration reg4_tb_cfg_str of reg4_tb is
    for test
        for UUT: c_reg4 use entity WORK.reg4(str)
            port map (p_d0, p_d1, p_d2, p_d3, p_en, p_clk,
                      p_q0, p_q1, p_q2, p_q3);
            for str
                for all: dlatch use entity GATES.dlatch(bhv); end for;
                for all: and2   use entity GATES.and2(bhv)
                         port map (a => i1, b => i2, z => z);
                end for; -- and2
            end for; -- str
        end for; -- UUT
    end for; -- test
end configuration reg4_tb_cfg_str;
```

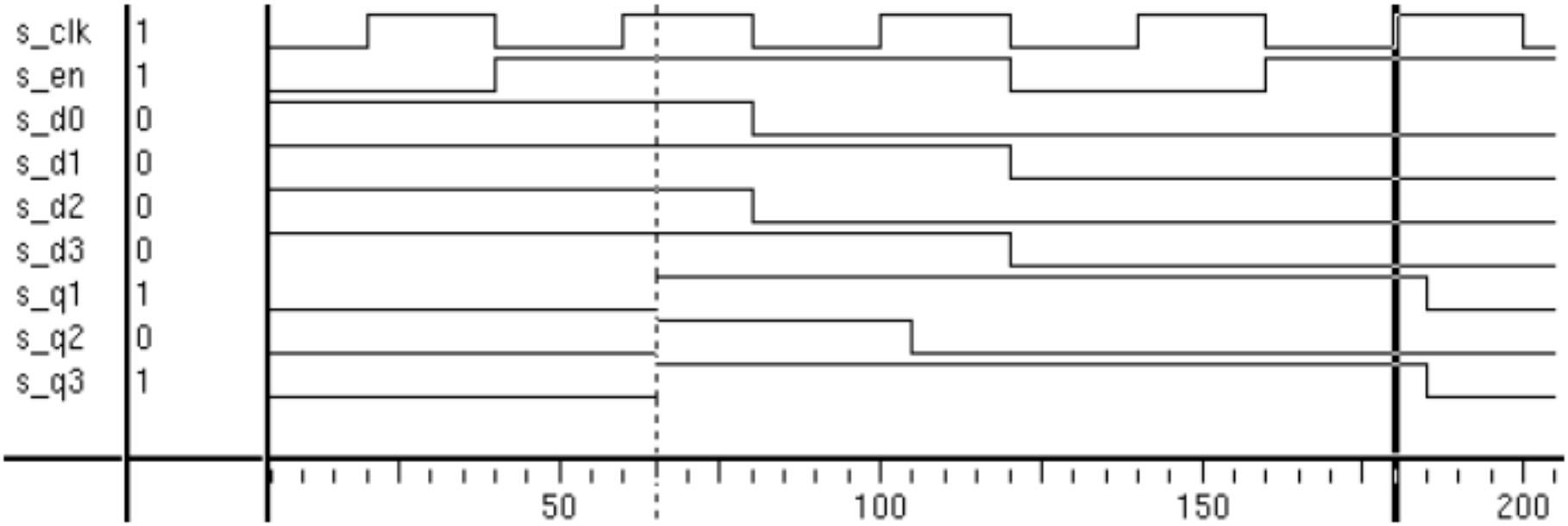
Utilisation du reg4 comportemental

II – VHDL : Configuration directe

- Il existe un mode plus simple qui n'utilise pas de déclaration de configuration
- Il utilise alors une instanciation directe, comme dans le fichier initial du reg4
- Mais on ne peut plus utiliser la structure **for all…**
- Il est enfin possible d'utiliser la configuration par défaut (sans avoir à la spécifier donc), s'il existe une entité d'interface identique dans la bibliothèque de travail

```
library GATES;
architecture str2 of reg4 is
    signal int_clk: BIT;  -- signal interne
begin
    -- instanciations directes des latches
    bit0: entity GATES.dlatch(bhv)
          port map (d => d0, clk => int_clk, q => q0);
    bit1: entity GATES.dlatch(bhv)
          port map (d => d1, clk => int_clk, q => q1);
    bit2: entity GATES.dlatch(bhv)
          port map (d => d2, clk => int_clk, q => q2);
    bit3: entity GATES.dlatch(bhv)
          port map (d => d3, clk => int_clk, q => q3);
    -- instanciation directe de la porte and2
    gate: entity GATES.and2(bhv)
          port map (en, clk, int_clk);
end architecture str2;
```

II – VHDL : Résultat de la simulation



Organisation Cours/TPs/Projet

- 8 séances de CM – dont 1-2 séances finales dédiées au Projet et préparation de l'Exam
- 8 ou 7 séances de Tps
  - Préparation des différents bloques logiques (ALU, Registres, ...)
  - Contrôleurs – FSM
  - Projet (assemblage et extension des bloques précédents) + définition d'un ISA + instanciation mémoire

Biblio

- http://hdl.telecom-paristech.fr/index.html
- Le cours de Licence 2:

http://perso-etis.ensea.fr/miramond/Enseignement/L2/circuits_numeriques.html

- Le cours de Licence 3:

http://perso-etis.ensea.fr/miramond/Enseignement/L3/Cours_VHDL_2011.html