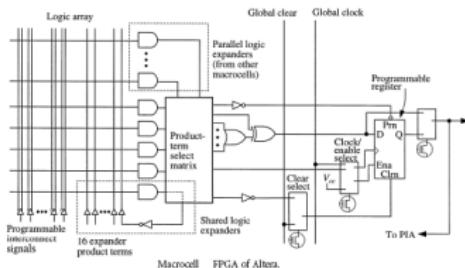
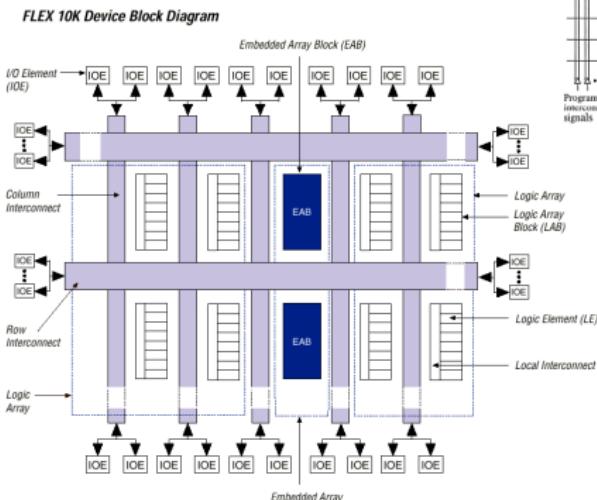


CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : FPGA



le bloc logique reconfigurable
CLB

matrice d'un FPGA

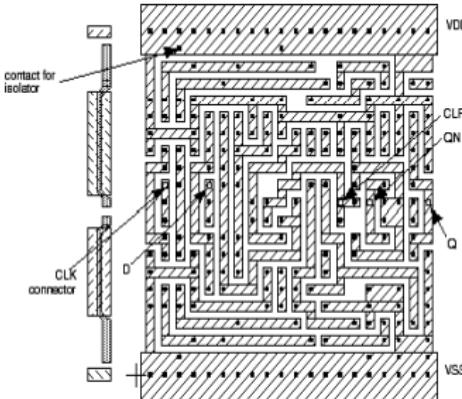
CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : *Sea of Gates*

Puce en partie réalisée

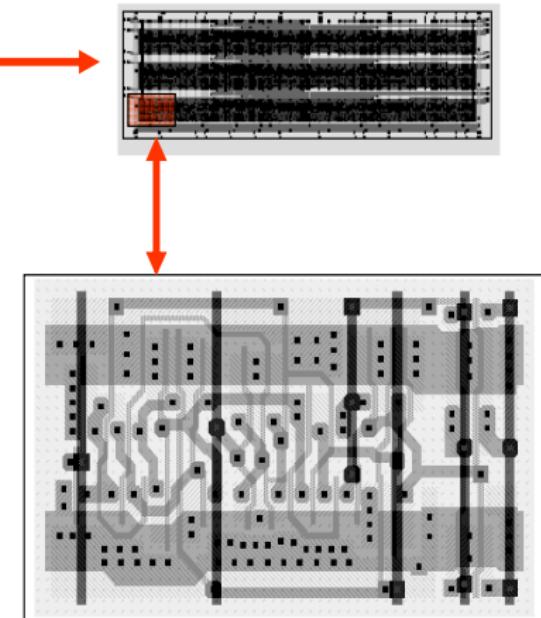
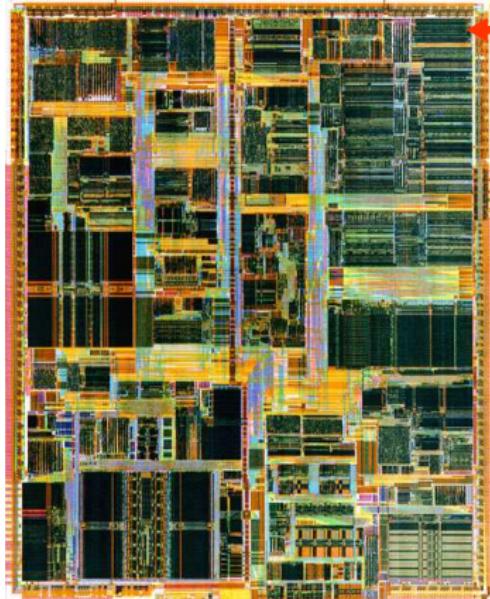


Connexions fournies
par le concepteur



CIRCUITS INTÉGRÉS

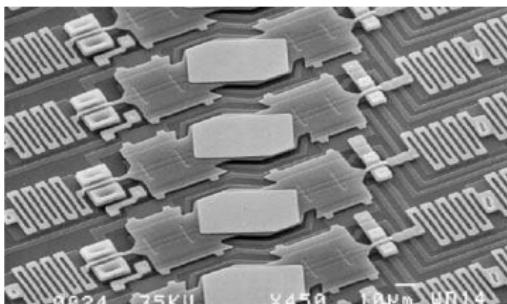
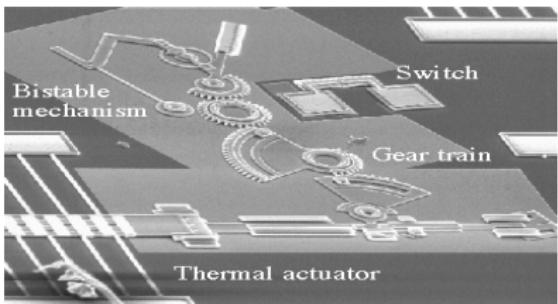
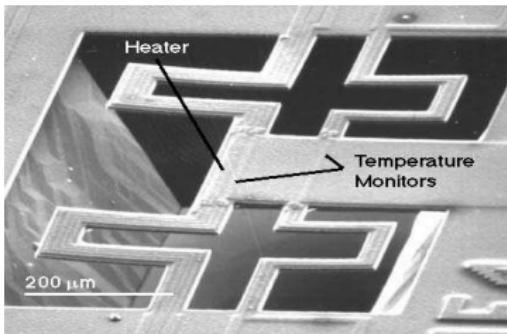
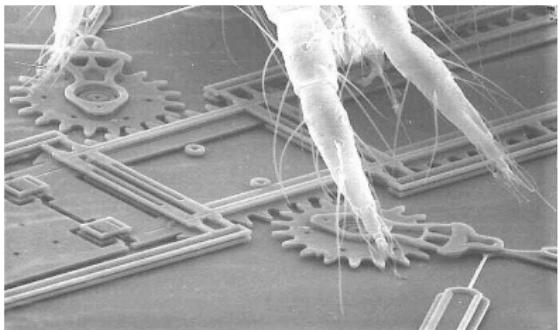
LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : *Standard Cells*



CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : MEMS

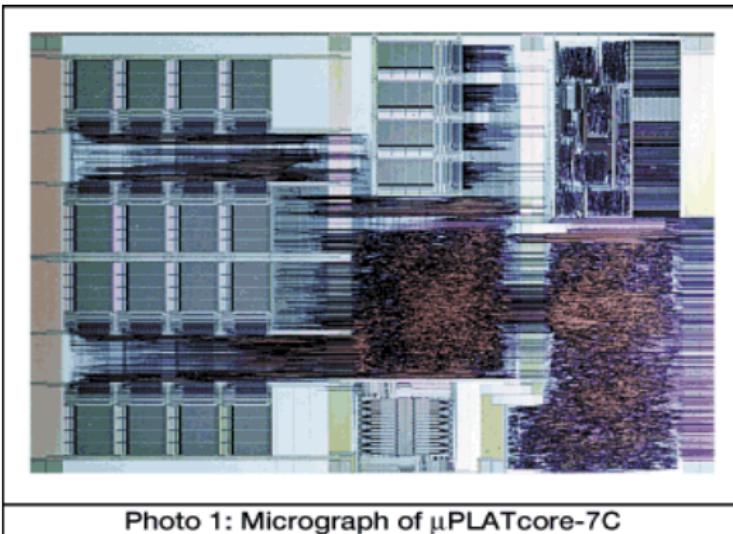
- MEMS : *Micro Electro-Mechanical Systems*



CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : IP

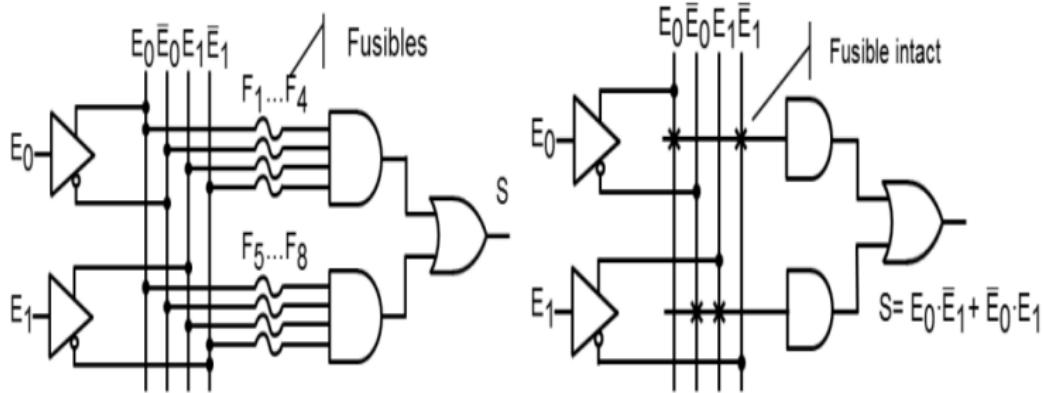
- IP : *Intellectual Property*



CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

- Les premiers circuits configurables : PAL (*Programmable Array Logic*) - une technologie à fusible
- Leur structure déduite de la forme canonique des équations logiques issues d'une table de vérité (somme de produits).

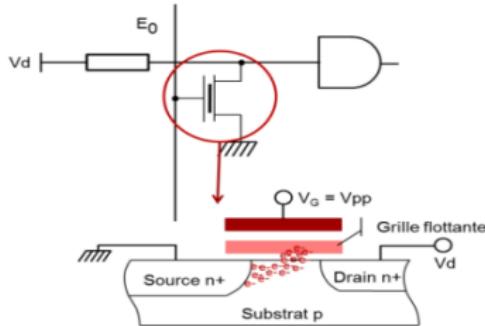


CIRCUITS INTÉGRÉS

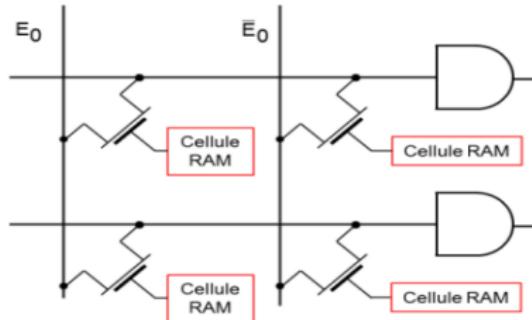
LES CIRCUITS PROGRAMMABLES

PLD et CPLD

- PLD - *Programmable Logic Device*
- trois grandes familles :
 - ▷ les anti-fusibles (programmable une seule fois)
 - ▷ le flash (reprogrammable)
 - ▷ la SRAM (reprogrammable)



a. PLD flash et transistor à grille isolée

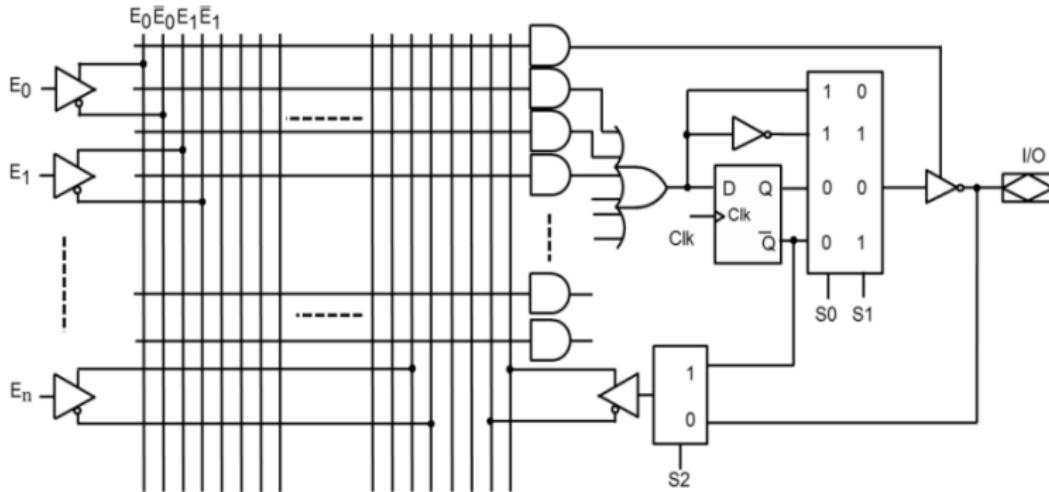


b. PLD SRAM

CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

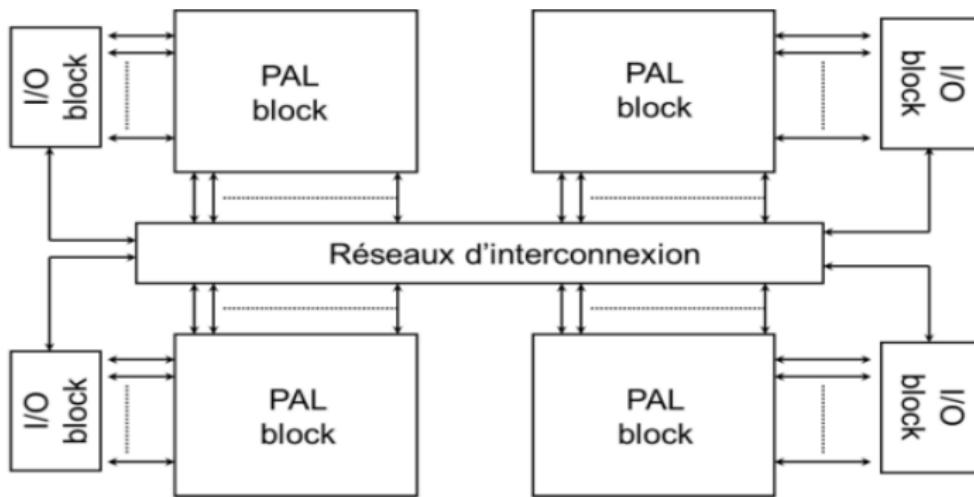
- Pour les circuits séquentielles, une bascule a été rajoutée à la sortie de chaque PAL



CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

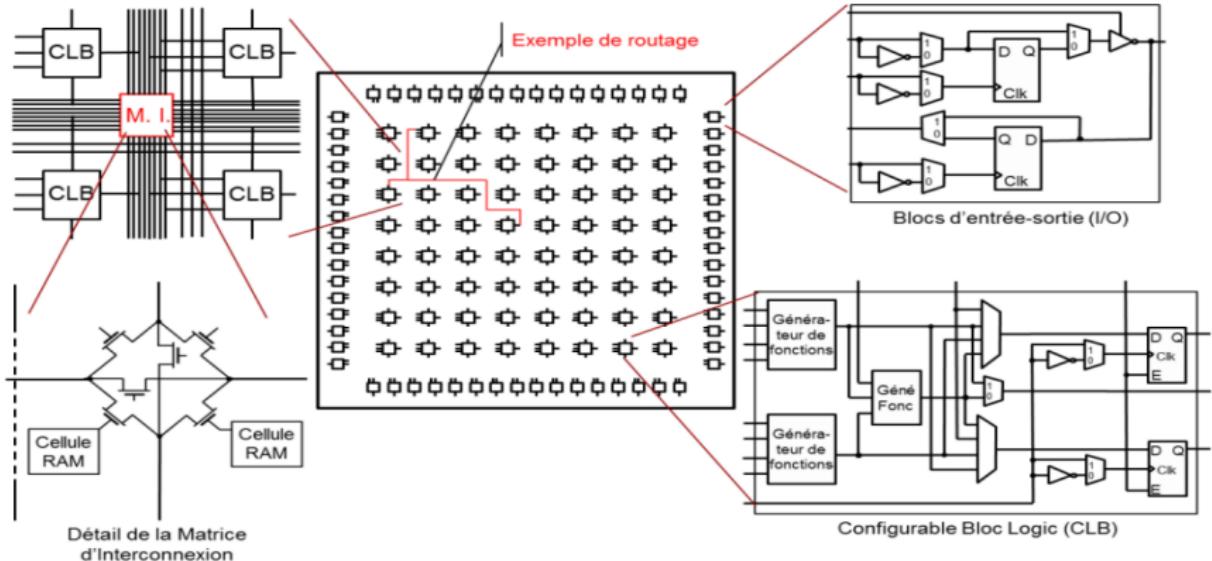
- CPLD - *Complex PLD* sont le regroupement de plusieurs PAL-registre avec un réseau programmable



CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

FPGA - *Field Programmable Gate Array*



SOMMAIRE

1 Introduction

- Motivations
- Histoire des semiconducteurs
- Circuits intégrés
- Conception de circuits intégrés numériques

CONCEPTION DE CI NUMÉRIQUES

POURQUOI LES CIRCUITS NUMÉRIQUES ?

Avantages des circuits numériques :

- Reproductibilité de l'information
- Flexibilité et fonctionnalité : l'information plus simple à stocker, transmettre et manipuler
- Plus économique : les circuits sont moins chers et plus simple à concevoir

Applications

- Numérisation de tous les domaines d'applications (télécommunications, santé, agroalimentaire, traitement d'information, transport, énergie, ...)
- Chaque application est spécifique et exige un certain degré de "customisation"

CONCEPTION DE CI NUMÉRIQUES

POURQUOI LES CIRCUITS NUMÉRIQUES ?

Les méthodes principales de "customisation"

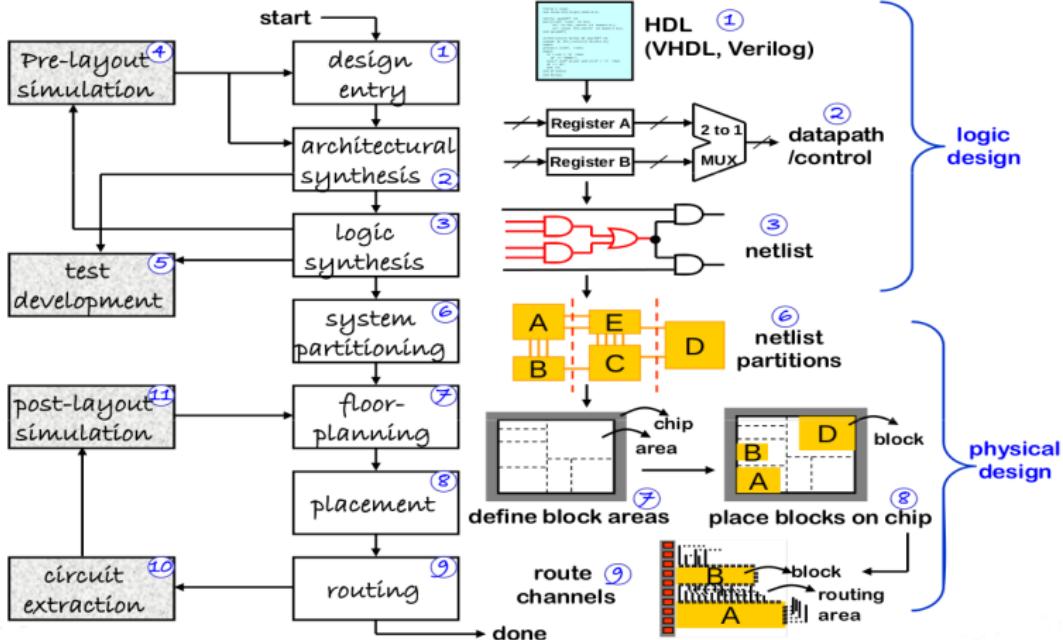
- Un HW "General-purpose" avec un SW adapté
 - ▷ processeurs à usage général : performants (e.g., Pentium) ou à bas cout (e.g., PIC microcontrôleurs)
 - ▷ processeur à usage spécifique : DSP (multiplication-addition), *network* processeurs (pour le buffering et routage), GPU (3D rendering)
- Custom HW
- Custom SW + custom HW (HW-SW co-design)

Une solution adoptée est toujours :

- Compromis entre programmabilité, coût, performances et consommation
- Une application complexe peut utiliser plusieurs méthodes de "customisation"

FLOT DE CONCEPTION

Design Flow



FLOT DE CONCEPTION

- ① Saisir le design schématiquement ou le décrire dans un langage de description HDL
- ② Utiliser HDL/schéma pour produire une description au niveau architectural
- ③ Générer une netlist (au niveau des portes logiques)
- ④ Vérifier/valider le design (vérification logique)
- ⑤ Développer une stratégie de test (génération de vecteurs de test)
- ⑥ Diviser un grand système en sous blocs de netlist
- ⑦ Faire le *floorplan* du circuit - arranger les blocs de base constituant le système
- ⑧ Décider l'emplacement exact de cellules standard dans un bloc du système
- ⑨ Effectuer les connexions entre les cellules et les blocs (routage global et local)
- ⑩ Déterminer les paramètres d'interconnexions (résistances/capacités)
- ⑪ Vérifier le design complet avec les charges supplémentaires liées aux interconnexions (post-routing simulation)



VUES D'UN SYSTÈME

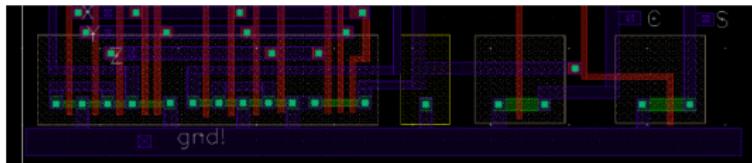
MODÈLES DE DESCRIPTION

Un système peut être représenté en utilisant des modèles de description :

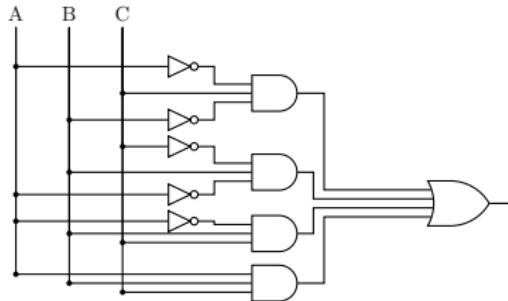
- Comportemental (*behavioral*)
 - ▷ Décrit les fonctionnalités d'un système et ses entrées/sorties (comportement d'un système)
 - ▷ Le système est considéré comme une boîte noire (le « quoi » et non le « comment »)
- Structurel
 - ▷ Décrit l'implémentation interne d'un système (les composants constitutifs et leurs interconnexions)
 - ▷ Le système est représenté sous forme d'un schéma bloc
- Physique :
 - ▷ La vue structurelle avec plus de détails : la taille des composants, leur placement, les chemins d'interconnexion, les matériaux utilisés, les masques de dessin, ...
 - ▷ Le layout d'un CI

VUES D'UN SYSTÈME

□ Physique 



□ Structurel 

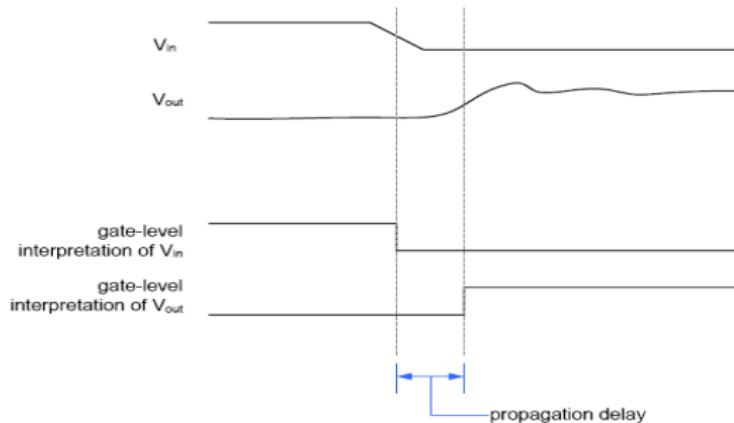


□ Comportemental 

```
entity d_ff is
  port( d  : in std_logic;
        q  : out std_logic;
        clk : in std_logic);
end entity;
```

NIVEAUX D'ABSTRACTION

- Comment représente-t-on une puce comprenant 10 millions de transistors ?
- Utilisation de niveaux d'abstraction
 - ▷ une représentation simplifiée d'un système
 - ▷ un certain nombre de propriétés est présenté
 - ▷ **Exemple :** la réponse d'une porte inverseuse



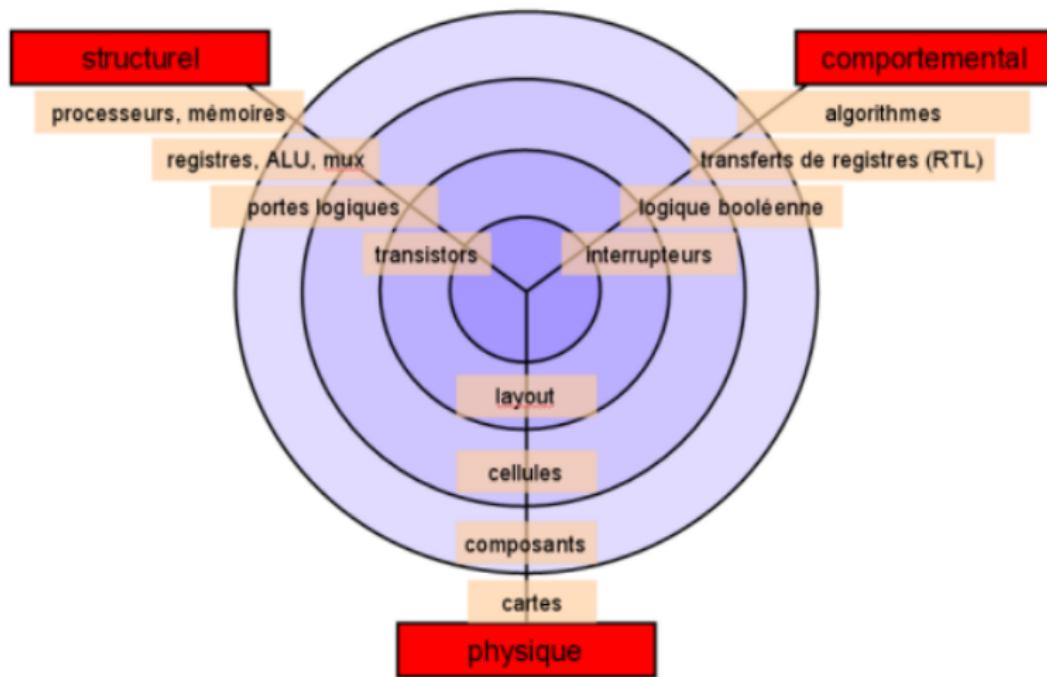
NIVEAUX D'ABSTRACTION

- Un niveau d'abstraction : niveau de description d'un système apportant un nombre de détails donnés. Un flot de conception est une succession de niveau d'abstractions
- Différents niveaux d'abstractions :
 - ▷ le niveau des transistors (*transistor level*)
 - ▷ le niveau des portes logiques (*gate level*)
 - ▷ le niveau des transferts de "registres" (*register transfer level*)
 - ▷ le niveau des processeurs (*processor level*)
- Caractéristiques de chaque niveau d'abstraction :
 - ▷ éléments constitutifs de base
 - ▷ représentation des signaux
 - ▷ représentation du temps
 - ▷ représentation comportementale
 - ▷ représentation physique

NIVEAUX D'ABSTRACTION

	typical blocks	signal representation	time representation	behavioral description	physical description
transistor	transistor, resistor	voltage	continuous function	differential equation	transistor layout
gate	and, or, xor, flip-flop	logic 0 or 1	propagation delay	Boolean equation	cell layout
RT	adder, mux, register	integer, system state	clock tick	extended FSM	RT level floor plan
processor	processor, memory	abstract data type	event sequence	algorithm in C	IP level floor plan

NIVEAUX D'ABSTRACTION



LES ÉTAPES DE DÉVELOPPEMENT

La conception d'un système numérique passe par un certain nombre d'étapes :

- Synthèse logique
- Synthèse physique
- Vérification et
- Test

LES ÉTAPES DE DÉVELOPPEMENT

SYNTHÈSE LOGIQUE

- Un processus de transformation d'un système numérique en une représentation de bas niveau en utilisant les composants prédéfinis (cellules) des bibliothèques de la technologie visée (fondeur ou FPGA)
- Le résultat de cette opération est un modèle de description structurel dans un niveau d'abstraction plus bas
- Les principaux types de synthèse :
 - ▷ La synthèse de haut niveau (*High-level synthesis*)
 - ▷ La synthèse RT
 - ▷ La synthèse au niveau porte
 - ▷ La correspondance technologique (mapping)

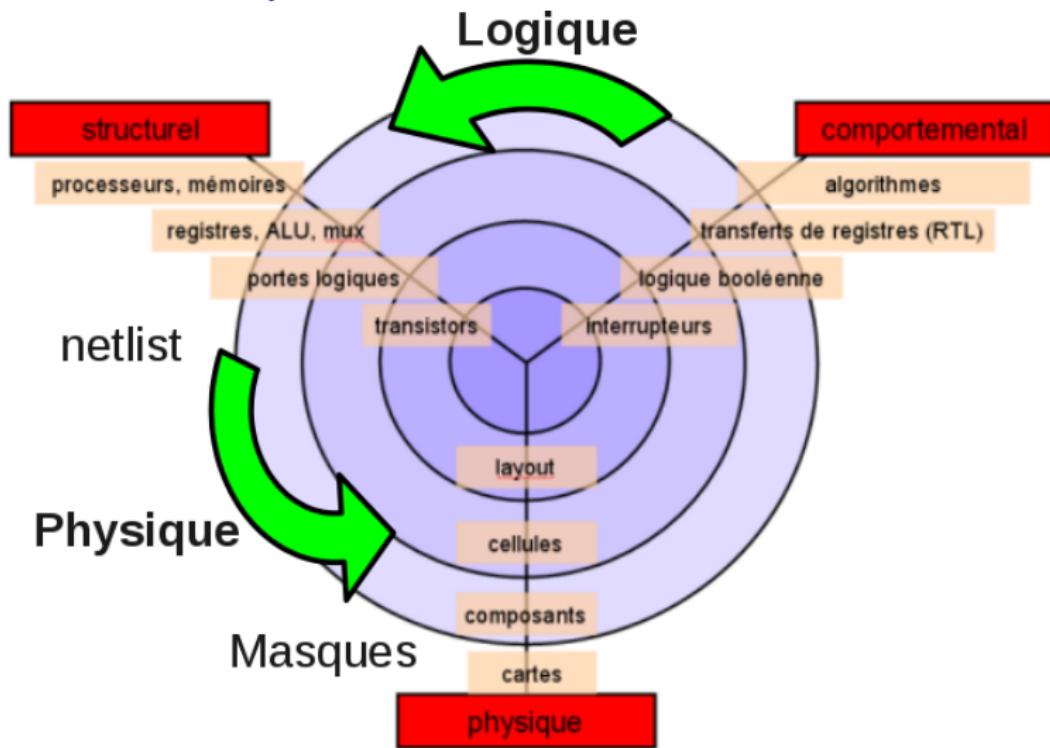
LES ÉTAPES DE DÉVELOPPEMENT

SYNTHÈSE PHYSIQUE

- La synthèse physique
 - ▷ Un processus de transformation d'une représentation structurelle de bas niveau (le résultat d'une synthèse logique) en une description physique du circuit (*layout*)
 - ▷ Les étapes principales :
 - Placement
 - Routage
 - ▷ Les informations sur les délais introduits par les pistes de routage peuvent être extraites (format Standard Delay File) → utile pour évaluer le respect des contraintes temporelles du système
 - ▷ A ce niveau, les rails d'alimentation et l'arbre de l'horloge sont également définis

LES ÉTAPES DE DÉVELOPPEMENT

SYNTHÈSE PHYSIQUE



LES ÉTAPES DE DÉVELOPPEMENT

VÉRIFICATION

□ Vérification

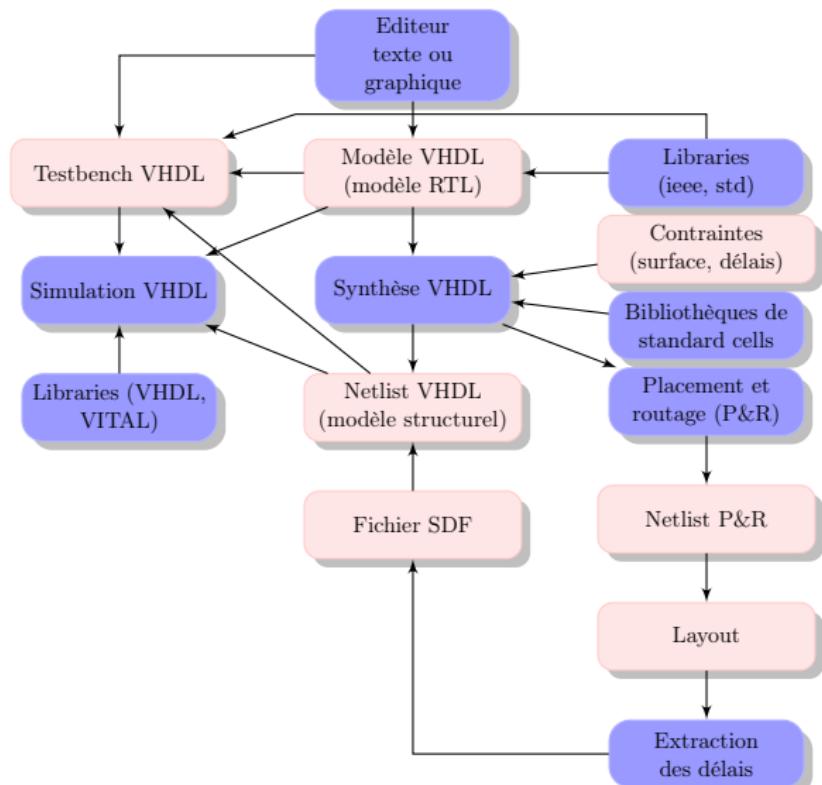
- ▷ Vérifier si le système conçu respecte le cahier des charges initial
- ▷ Deux aspects principaux à vérifier :
 - La fonctionnalité et
 - Les performances obtenues : timing (et parfois surface ou énergie)
- ▷ La vérification est effectuée en appliquant des stimuli aux entrées du système et en observant l'évolution des signaux dans le temps (utilisation des simulateurs évènementiels discrets). Ces simulations peuvent être à forte intensité de calcul
- ▷ Cette vérification peut être effectuée à plusieurs niveaux d'abstraction (avec les mêmes stimuli)
- ▷ Certaines vérifications peuvent être effectuées par une émulation sur matériel

LES ÉTAPES DE DÉVELOPPEMENT

FABRICATION ET TEST

- Réalisation du circuit et test
 - ▷ Une fois le circuit réalisé, les tests de chaque composant réalisé doivent être effectués
 - ▷ L'étape de test nécessite souvent (notamment pour des grands designs) la conception des circuits de test
 - ▷ Parfois ces circuits font même partie du design initial : *Built-in self test (BIST), scan chains, ...*
- Les outils de CAO (*Electronics Design Automation - EDA*) peuvent automatiser un certain nombre de tâches
- Il ne faut pas espérer qu'un mauvais design va devenir bon grâce aux outils de synthèse

FLOT DE CONCEPTION VHDL





SOMMAIRE

- 2 Hardware Description Language
 - HDL vs PL
 - HDL





HDL vs PL

LANGAGE DE DESCRIPTION DE MATÉRIEL VS LANGAGE DE PROGRAMMATION

- HDL = Hardware Description Language
- Peut-on utiliser C, Java ou un autre langage comme un HDL ?
- Caractéristiques du matériel :
 - ▷ Interconnexion des composants
 - ▷ Opérations concurrentes
 - ▷ Concept de délai de propagation et de synchronisation
- Un langage traditionnel de programmation :
 - ▷ séquentiel
 - ▷ pas de notion de parallélisme
 - ▷ pas de notion de temps
- La réponse : **NON** → HDL





SOMMAIRE

- 2 Hardware Description Language
 - HDL vs PL
 - HDL



LANGAGE DE DESCRIPTION DE MATÉRIEL

- Un langage de description de matériel moderne permet de représenter les caractéristiques principales d'un circuit numérique :
 - ▷ entité (notion de composant)
 - ▷ connectivité (interconnexion avec d'autres composants)
 - ▷ concurrence
 - ▷ timing
- Permet des descriptions au niveau portes logiques et RT
- Permet des descriptions structurelles et comportementales
- Deux principaux HDL :
 - ▷ VHDL et
 - ▷ Verilog
- Les syntaxes complètement différentes
- Les possibilités très proches
- Les deux sont des standards industriels et supportés par la plupart des outils CAO





LE LANGAGE VHDL

- VHDL : VHSIC (*Very High Speed Integrated Circuit*) HDL
- Une initiative de DoD dans les années 80
- Reconnu comme standard IEEE en 1987 (VHDL-87)
- Une modification majeure en 1993 (VHDL-93)
- Revu de manière continue
- Des modifications mineures en 2002 et 2008 (VHDL-2002 et VHDL-2008)





EXTENSIONS IEEE

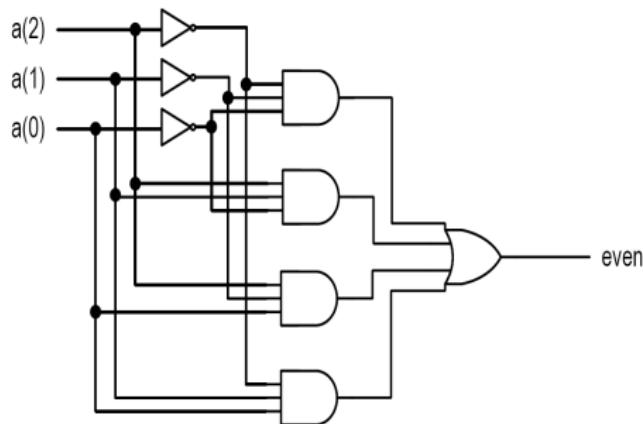
- IEEE standard 1076.1 Analog and Mixed Signal Extensions (VHDL-AMS)
- IEEE standard 1076.2 VHDL Mathematical Packages
- IEEE standard 1076.3 Synthesis Packages
- IEEE standard 1076.4 VHDL Initiative Towards ASIC Libraries (VITAL)
- IEEE standard 1076.6 VHDL Register Transfer Level (RTL) Synthesis
- IEEE standard 1164 Multivalue Logic System for VHDL Model Interoperability
- IEEE standard 1029.1-1991 VHDL Waveform and Vector Exchange (WAVES)



VHDL

EXEMPLE : CIRCUIT DE DÉTECTION DE PARITÉ

- Entrées : $a(2)$, $a(1)$ et $a(0)$
- Sortie : even



$$\text{even} = \overline{a(2)} \cdot \overline{a(1)} \cdot \overline{a(0)} + \overline{a(2)} \cdot a(1) \cdot a(0) + a(2) \cdot \overline{a(1)} \cdot a(0) + a(2) \cdot a(1) \cdot \overline{a(0)}$$

a(2)	a(1)	a(0)	even
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

VHDL

EXEMPLE : CIRCUIT DE DÉTECTION DE PARITÉ

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

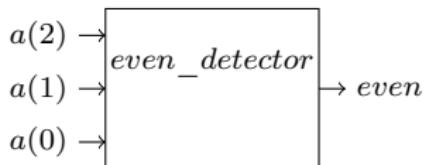
-- déclaration de l'entité
entity even_detector is
    port(
        a: in std_logic_vector(2 downto 0);
        even: out std_logic
    );
end even_detector;

-- corps de l'architecture
architecture sop_arch of even_detector is
    signal p1, p2, p3, p4 : std_logic;
begin
    even <= (p1 or p2) or (p3 or p4) after 20 ns;
    p1 <= (not a(2)) and (not a(1)) and (not a(0)) after 15 ns;
    p2 <= (not a(2)) and a(1) and a(0) after 12 ns;
    p3 <= a(2) and (not a(1)) and a(0) after 12 ns;
    p4 <= a(2) and a(1) and (not a(0)) after 12 ns;
end sop_arch ;
```

VHDL

ENTITÉ

- L'entité représente la description de l'interface du circuit
- Si on fait une analogie avec les représentations schématiques → le symbole du circuit
- L'entité précise :
 - ▷ le nom du circuit
 - ▷ les ports d'entrée/sortie
 - leur nom,
 - leur direction (in, out ou inout)
 - leur type (bit, bit_vector, integer, std_logic,...)
 - ▷ Les paramètres génériques (optionnel)



```
entity even_detector is
  port(
    a: in std_logic_vector(2 downto 0);
    even: out std_logic
  );
end even_detector;
```

VHDL

L'ENTITÉ : EXEMPLES

```
1 entity even_detector is
2     port(
3         a: in std_logic_vector(2 downto 0);
4         even: out std_logic
5     );
6 end even_detector;
```

```
-- Nom de l'entité
-- port déclaration début
-- a en entrée
-- even en sortie
-- port déclaration fin
-- entité fin
```

```
1 entity xor2 is
2     port(
3         i1, i2: in std_logic;
4         o1: out std_logic
5     );
6 end xor2;
```

```
-- Nom de l'entité
-- port déclaration début
-- i1,i2 en entrée
-- o1 en sortie
-- port déclaration fin
-- entité fin
```



VHDL

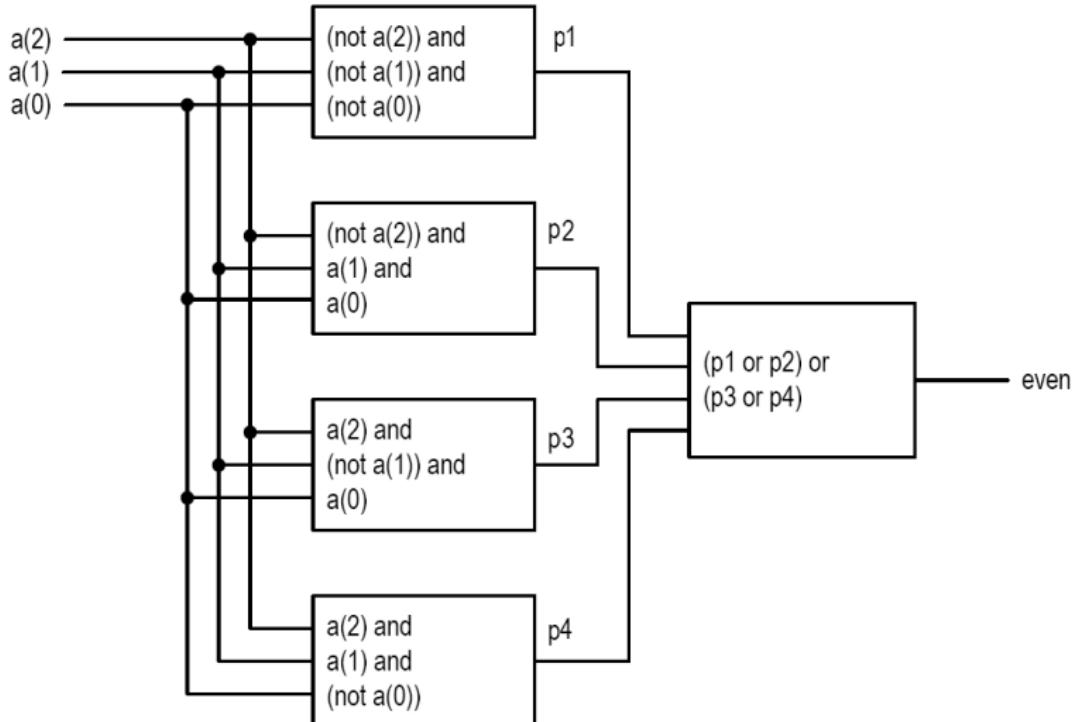
ARCHITECTURE

- L'architecture est la description interne du circuit
- Toujours associée à une entité
- Une entité peut avoir plusieurs architectures
- Le mécanisme de configuration permet d'associer l'architecture rattachée à une entité

```
1 architecture sop_arch of even_detector is
2   signal p1, p2, p3, p4 : std_logic;
3 begin
4   even <= (p1 or p2) or (p3 or p4) after 20 ns;
5   p1 <= (not a(2)) and (not a(1)) and (not a(0)) after 15 ns;
6   p2 <= (not a(2)) and a(1) and a(0) after 12 ns;
7   p3 <= a(2) and (not a(1)) and a(0) after 12 ns;
8   p4 <= a(2) and a(1) and (not a(0)) after 12 ns;
9 end sop_arch ;
```

VHDL

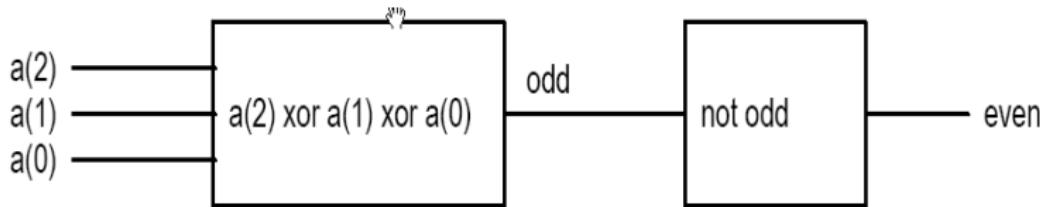
ARCHITECTURE : INTERPRÉTATION



VHDL

ARCHITECTURE

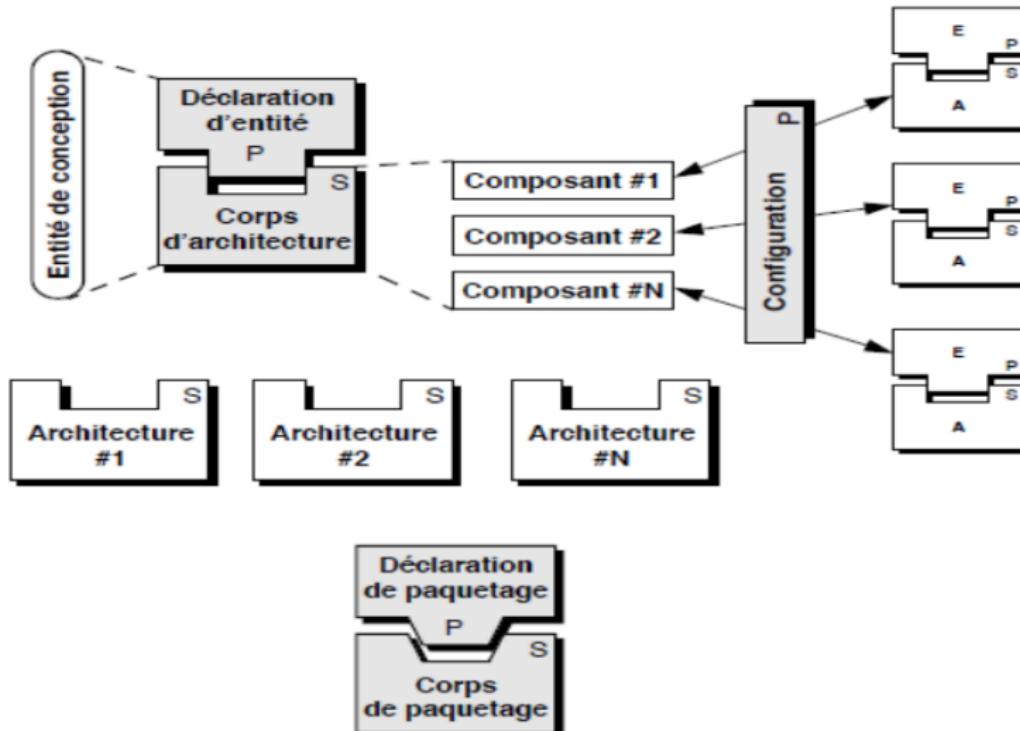
```
1 architecture xor_arch of even_detector is
2     signal odd: std_logic;
3 begin
4     even <= not odd;
5     odd <= a(2) xor a(1) xor a(0);
6 end xor_arch;
```



- Une autre architecture pour le même entité
- Le temps delta délai δ

VHDL

UNITÉS DE CONCEPTION





VHDL

ARCHITECTURE

Deux types de descriptions :

- Comportementale
 - ▷ Correspond à expliciter le comportement d'un modèle par ses équations
 - ▷ Pas de définition formelle d'une description comportementale en VHDL
 - ▷ Tous les objets déclarés dans l'entité sont visibles dans l'architecture
 - ▷ Utilisation du **process** : la construction permettant d'encapsuler la sémantique séquentielle
 - Les **process** s'exécutent de manière concurrente entre-eux

VHDL

ARCHITECTURE

Syntaxe d'un processus :

```
process(liste_de_sensibilite)
    variable declaration;
begin
    instructions sequentielles;
end process;
```

Exemple d'utilisation :

```
1 architecture beh1_arch of even_detector is
2 signal odd: std_logic;
3 begin
4     -- inverseur
5     even <= not odd;
6     -- réseau xor pour la parité impaire
7     process(a)
8         variable tmp: std_logic;
9     begin
10        tmp := '0';
```



VHDL

ARCHITECTURE

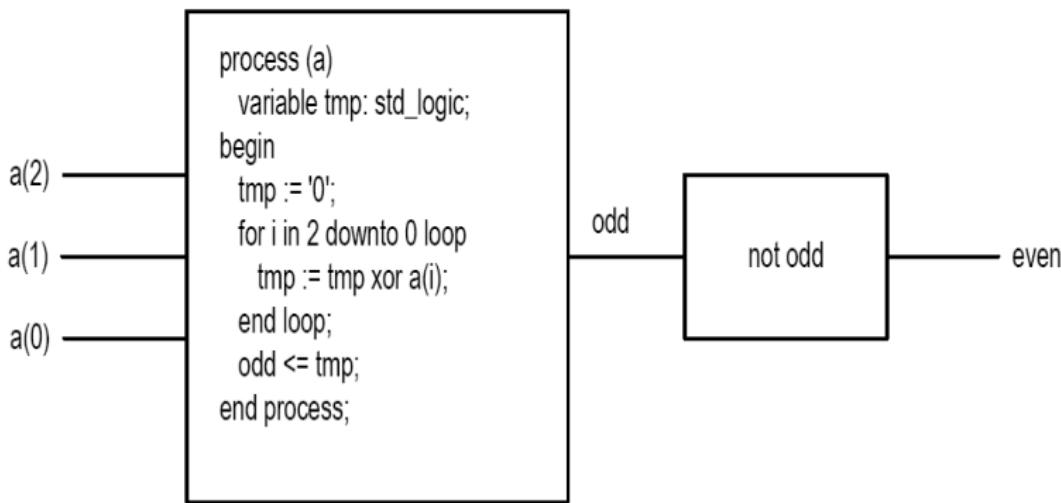
```
11      for i in 2 downto 0 loop
12          tmp := tmp xor a(i);
13      end loop;
14      odd <= tmp;
15  end process;
16 end beh1_arch;
```



VHDL

ARCHITECTURE

Représentation schématique d'un processus :



VHDL

ARCHITECTURE

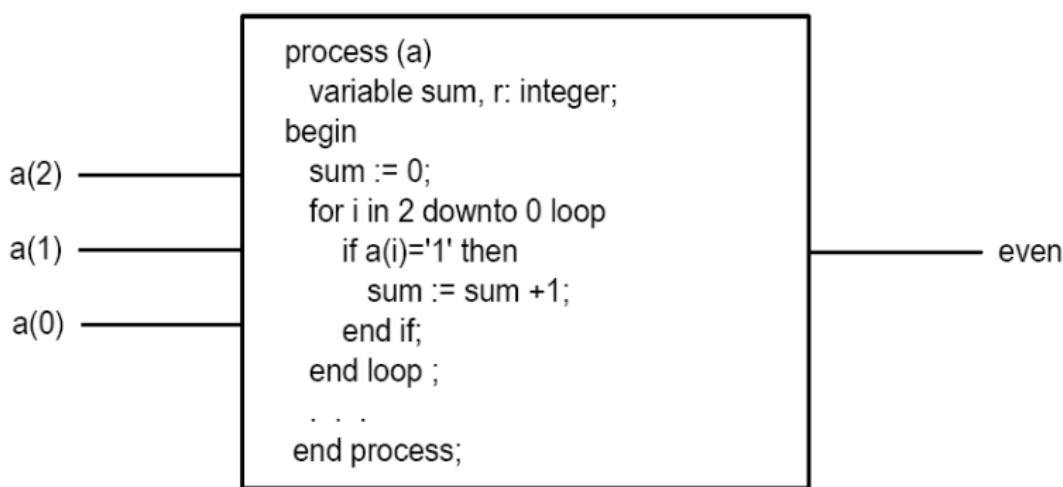
Entité comportementale :

```
1 architecture beh2_arch of even_detector is
2 begin
3     process(a)
4         variable sum, r: integer;
5     begin
6         sum := 0;
7         for i in 2 downto 0 loop
8             if a(i)='1' then
9                 sum := sum +1;
10            end if;
11        end loop ;
12        r := sum mod 2;
13        if (r=0) then
14            even <= '1';
15        else
16            even <='0';
17        end if;
18    end process;
19 end beh2_arch;
```

VHDL

ARCHITECTURE

Représentation schématique d'une entité comportementale :



VHDL

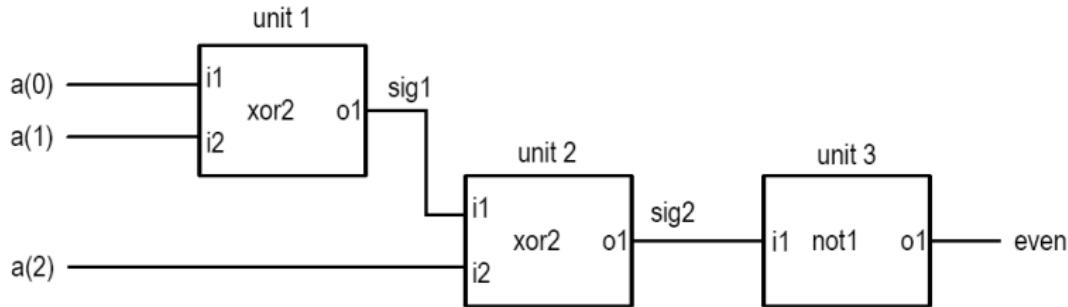
ARCHITECTURE

□ Structurelle

- ▷ Correspond à l'instanciation (utilisation) hiérarchique d'autres composants
- ▷ La description structurelle spécifie les types de composants à utiliser et leurs interconnexions
- ▷ Utilisation du mot clé **component**
 - Le composant à utiliser doit être déclaré au préalable (partie déclarative de l'architecture)
 - et ensuite instancié (utilisé) dans le corps de l'architecture
 - L'architecture du composant peut-être décrite dans le même fichier ou dans un autre fichier incorporé au projet (bibliothèque **work**)

VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE



- L'exemple du détecteur de parité à base de portes XOR à deux entrées

VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : TOP-LEVEL

```
1 architecture str_arch of even_detector is
2     -- déclaration de la porte xor
3     component xor2
4         port(
5             i1, i2: in std_logic;
6             o1: out std_logic
7         );
8     end component;
9     -- déclaration de l'inverseur
10    component not1
11        port(
12            i1: in std_logic;
13            o1: out std_logic
14        );
15    end component;
16    signal sig1,sig2: std_logic;
17
18 begin
19     -- instanciation de la 1ere porte xor
```



VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : TOP-LEVEL

```
20      unit1: xor2
21          port map (i1 => a(0), i2 => a(1), o1 => sig1);
22          -- instantiation de la 2eme porte xor
23      unit2: xor2
24          port map (i1 => a(2), i2 => sig1, o1 => sig2);
25          -- instantiation de l'inverseur
26      unit3: not1
27          port map (i1 => sig2, o1 => even);
28 end str_arch;
```

VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : COMPOSANTS

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3 entity xor2 is
4     port(
5         i1, i2: in std_logic;
6         o1: out std_logic
7     );
8 end xor2;
9
10 architecture beh_arch of xor2 is
11 begin
12     o1 <= i1 xor i2;
13 end beh_arch;
14
15 -- inverseur
16 library ieee;
17 use ieee.std_logic_1164.all;
18 entity not1 is
19     port(
```



VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : COMPOSANTS

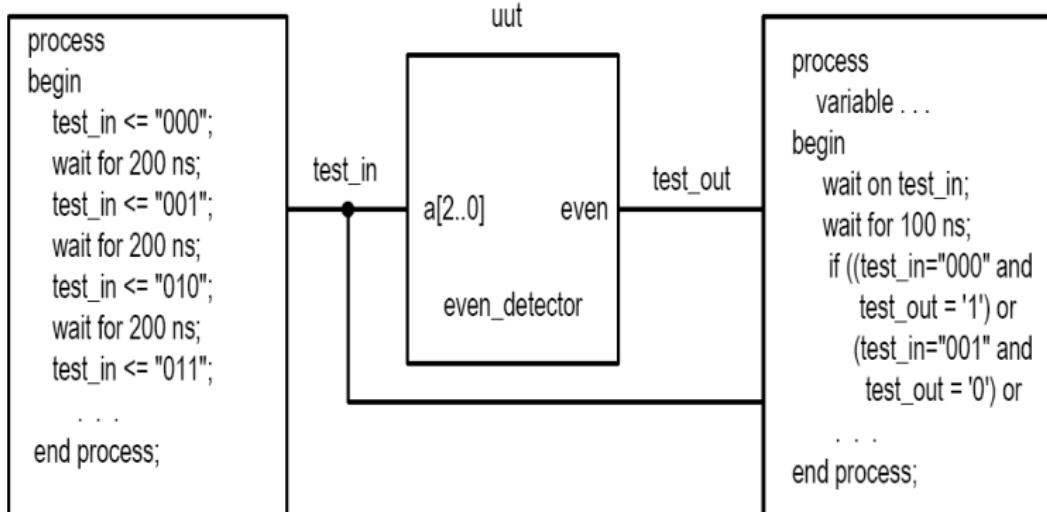
```
20      i1: in std_logic;
21      o1: out std_logic
22  );
23 end not1;
24 architecture beh_arch of not1 is
25 begin
26     o1 <= not i1;
27 end beh_arch;
```



VHDL

TESTBENCH

- Tester le composant décrit
- Décrire un certain nombre de stimuli pour valider le fonctionnement
- Un testbench est à la fois comportemental et structurel



VHDL

TESTBENCH : EXEMPLE

```
1 library ieee;
2 use ieee.std_logic_1164.all;
3
4 entity even_detector_testbench is
5 end even_detector_testbench;
6
7 architecture tb_arch of even_detector_testbench is
8     component even_detector
9         port(
10             a: in std_logic_vector(2 downto 0);
11             even: out std_logic
12         );
13     end component;
14     signal test_in: std_logic_vector(2 downto 0);
15     signal test_out: std_logic;
16
17 begin
18     -- instancier le circuit à vérifier
19     uut: even_detector
```

VHDL

TESTBENCH : EXEMPLE

```
20      port map( a=>test_in, even=>test_out);
21      -- générateur de vecteurs de test
22      process
23      begin
24          test_in <= "000";
25          wait for 200 ns;
26          test_in <= "001";
27          wait for 200 ns;
28          test_in <= "010";
29          wait for 200 ns;
30          test_in <= "011";
31          wait for 200 ns;
32          test_in <= "100";
33          wait for 200 ns;
34          test_in <= "101";
35          wait for 200 ns;
36          test_in <= "110";
37          wait for 200 ns;
38          test_in <= "111";
39          wait for 200 ns;
```

VHDL

TESTBENCH : EXEMPLE

```
40      end process;
41      -- vérificateur
42  process
43      variable error_status: boolean;
44  begin
45      wait on test_in;
46      wait for 100 ns;
47      if ((test_in="000" and test_out = '1') or
48          (test_in="001" and test_out = '0') or
49          (test_in="010" and test_out = '0') or
50          (test_in="011" and test_out = '1') or
51          (test_in="100" and test_out = '0') or
52          (test_in="101" and test_out = '1') or
53          (test_in="110" and test_out = '1') or
54          (test_in="111" and test_out = '0'))
55      then
56          error_status := false;
57      else
58          error_status := true;
59      end if;
```



VHDL

TESTBENCH : EXEMPLE

```
60      -- rapport d'erreurs
61      assert not error_status
62          report "test failed!"
63          severity error;
64      end process;
65  end tb_arch;
```

