Conception des systèmes numériques

1. Codage (Rappel)

Base de Numération

Dans un système de base *B*, sur *n* chiffres, un nombre s'écrit

$$b_{n-1}b_{n-2}\dots b_2\,b_1b_0$$

La valeur décimale de ce nombre est égale à : $P = \sum_{i=0}^{n-1} b_i . B^i$

avec *B* qui exprime la base.

Bases	Systèmes	Vocabulaires
2	Binaire	0,1
8	Octal	0,1,2,3,4,5,6,7
10	Décimal	0,1,2,3,4,5,6,7,8,9
16	Hexadécimal	0,1,2,3,4,5,6,7,8,9,A,B, C,D,E,F

Base de Numération

- Conversion décimal-binaire
  - La conversion décimal-binaire peut s'effectuer en utilisant la méthode inverse de celle énoncée précédemment. Fastidieux pour de grand nombre.
    - Réalise une division par 2

```
29  2
1   14   2
    0  7   2
       1  3  2
          1  1 2
             1  0
```

Nombre binaire = 11101

Base de Numération

Conversion héxadécimale

- Travail avec des quartets binaires : 1010
- La taille du mot binaire de base est l'octet
- Un octet = Deux Quartets

- La base du système Hexadécimal est la base 16
- Il faut donc 16 symboles : 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Hexa	Décimal	Binaire	Hexa	Décimal	Binaire
0	0	0000	8	8	1000
1	1	0001	9	9	1001
2	2	0010	A	10	1010
3	3	0011	B	11	1011
4	4	0100	C	12	1100
5	5	0101	D	13	1101
6	6	0110	E	14	1110
7	7	0111	F	15	1111

Base de Numération
Représentation des entiers

Exemples d'opérations binaires

Addition	Soustraction	Multiplication	Division
10101111 +00000001. ——————— 10110000	10101111 -00000001. ——————— 10101110	101 110 ——————— 000 101 . 101…. ——————— 11110	110101 1001 ——————— 101

Base de Numération
Représentation des entiers

Nombre non signé: $b_{n-1}b_{n-2}\dots b_1 b_0$ avec $b_i \in \{0,1\}$

$$P = \sum_{i=0}^{n-1} b_i . 2^i \quad \text{avec } P \in [0, 2^n - 1] \quad \text{et } P \geq 0$$

Nombre signé: Introduire un bit de signe → bit de poids fort

- Exemple de nombre sur 4 bits

signe

b_3	b_2	b_1	b_0	signe	valeur décimale
0	1	0	0	+	4
1	1	0	0	-	-4

Valeur absolue

Inconvénients: l'addition ne donne pas naturellement zéro et nécessite trop de logique pour réaliser des opérateurs arithmétiques

Base de Numération
Représentation des entiers

Nombres signés: Complément à 2

- Utilisation d'un codage qui permet de limiter les opérateurs
- Complément à 2 :
  - Bit de signe : bit de poids fort
  - Si bit de signe = 0 : Le nombre est codé
  - Si bit de signe = 1 : Complément à 2 pour avoir la valeur

Principe : Pour un nombre de n bits complémenter le nombre pour arriver à 2^n

Codage de 7 :

b_3	b_2	b_1	b_0	signe	valeur décimale
0	1	1	1	+	7

Codage de -7

b_3	b_2	b_1	b_0	signe	valeur décimale
1	0	0	1	-	-7

Base de Numération
Représentation des entiers

Nombres signés : Complément à 2

- Étapes pour complémenter à 2
  - Faire le complément à 1 du nombre : complémentation bit à bit
  - Ajouter 1 au nombre

Exemple : codage de -5

b_3	b_2	b_1	b_0	Commentaires
0	1	0	1	Valeur Absolue
1	0	1	0	Complément à 1
+			1	Ajout de 1
1	0	1	1	Complément à 2

- Avantages :
  - Unicité du 0
  - Utilisation du même opérateur pour l'addition et la soustraction

Base de Numération
Représentation des entiers

Si P positif on le code:

$$P = \sum_{i=0}^{i=N-1} b_i * 2^i$$
$$P = b_{N-1} * 2^{N-1} + \sum_{i=0}^{i=N-2} b_i * 2^i \text{ avec } b_{N-1} = 0$$

Si P négatif on le code:

$$P = -(2^N - \sum_{i=0}^{i=N-1} b_i * 2^i)$$
$$P = -(2^N - b_{N-1} * 2^{N-1} - \sum_{i=0}^{i=N-2} b_i * 2^i) \text{ avec } b_{N-1} = 1$$
$$P = -(2^N - 2^{N-1} - \sum_{i=0}^{i=N-2} b_i * 2^i)$$
$$P = -(2^{N-1}(2-1) - \sum_{i=0}^{i=N-2} b_i * 2^i)$$
$$P = -(2^{N-1} - \sum_{i=0}^{i=N-2} b_i * 2^i)$$
$$P = -b_{N-1} * 2^{N-1} + \sum_{i=0}^{i=N-2} b_i * 2^i \text{ avec } b_{N-1} = 1$$

Nombre en complément à 2

$$P = -b_{N-1} * 2^{N-1} + \sum_{i=0}^{i=N-2} b_i * 2^i$$

Échelle des nombres entiers signés représentables

$-2^{N-1} < P < 2^{N-1}-1$

Base de Numération
Codage des nombres à virgule fixe

$$P = -b_{n-1}.2^{n-1} + \sum_{i=0}^{n-2} b_i.2^i$$

Par convention, on place la virgule et on interprète

2^{n-1} ---------- 2^0 avant de placer la virgule

MSB xxxxxx , xxxx LSB

2^{n-1-k} 2^0,2^{-1} 2^{-k} avec la virgule au rang k

Dynamique : 2^{n-1-k}

Résolution : 2^{-k} # 0

Nombre fractionnaire codé sur n bits au format dit Q_k:

$$P = -b_{n-1}.2^{n-1-k} + \sum_{i=0}^{n-2-k} b_i.2^i + \sum_{i=-k}^{-1} b_i.2^i$$

Base de Numération
Codage des nombres à virgule fixe

Nombre fractionnaire codé sur n bits au format dit Q_k:

$$P = -b_{n-1}.2^{n-1-k} + \sum_{i=0}^{n-2-k} b_i.2^i + \sum_{i=-k}^{-1} b_i.2^i$$

Exemples avec n = 8 et k = 5

1/3	000 01011	0,34375
$\sqrt{2}$	001 01101	1,40625

Échelle des nombres entiers signés représentables au format Q_k

$-2^{n-k-1} < P < (2^{n-k-1}-1)+q.(2^k-1)$ Avec $q = 2^{-k}$

Propriétés :

- le produit de deux nombres codés au format Q_x et Q_y est au format Q_{x+y}
- la somme de deux nombres codés au format Q_x est au format Q_x

Base de Numération
Codage des nombres à virgule flottante

$N = M.b^E$

M = mantisse en 2* de forme 0,xxx
b = base de l'exponentiation (2 ou 16)
E = exposant en binaire décalé (exposant –décalage)

On stocke la chaîne de bits ME dans le calculateur

Exemple : codage de π sur 5 chiffres de mantisse
et 2 chiffres d'exposant (en décimal)

$\rightarrow 0{,}3141.10^1 = 0{,}0003.10^4$

On dit qu'un flottant est normalisé quand le premier chiffre significatif est juste derrière la virgule (précision maximum)

Base de Numération
Codage des nombres à virgule flottante
Binaire décalé

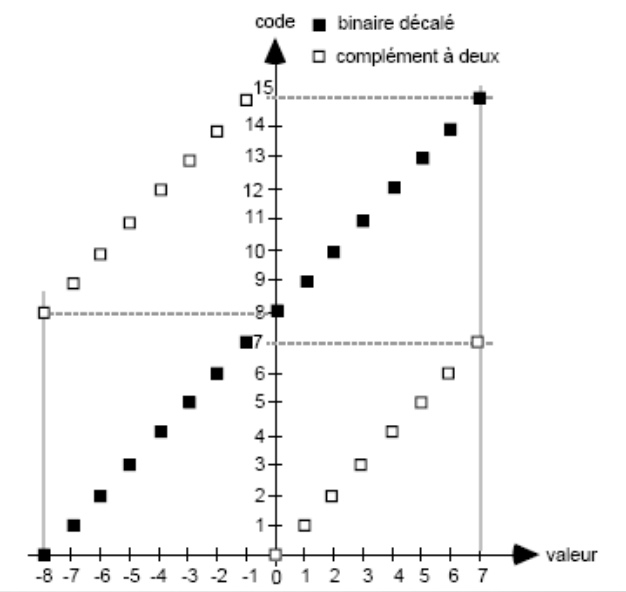
Le binaire décalé est un codage dérivé du complément à deux. Il est utilisé pour pouvoir comparer facilement des entiers signés. Sa principale application est le codage de l'exposant des nombres réels.

Valeur relative exprimée en décimal	valeur exprimée en C2	équivalent décimal du codage binaire
+7	0111	7
+6	0110	6
+5	0101	5
+4	0100	4
+3	0011	3
+2	0010	2
+1	0001	1
0	0000	0
-1	1111	15
-2	1110	14
-3	1101	13
-4	1100	12
-5	1011	11
-6	1010	10
-7	1001	9
-8	1000	8

Valeur relative exprimée en décimal	Valeur exprimée en binaire décalé	Equivalent décimal du codage binaire
+7	1111	15
+6	1110	14
+5	1101	13
+4	1100	12
+3	1011	11
+2	1010	10
+1	1001	9
0	1000	8
-1	0111	7
-2	0110	6
-3	0101	5
-4	0100	4
-5	0011	3
-6	0010	2
-7	0001	1
-8	0000	0

Base de Numération
Codage des nombres à virgule flottante
Binaire décalé

Dans le codage en complément à 2, C2, les codes des nombres positifs sont plus petits que ceux des nombres négatifs.



Par contre, les codes « binaire décalé » des nombres positifs sont supérieurs aux codes des nombres négatifs.
Cette propriété du binaire décalé permet de **comparer plus facilement les codes quand il faut comparer des nombres.**

Base de Numération
Codage des nombres à virgule flottante

Norme internationale : IEEE 754 flottant sur 32 bits

b_{31} .... b_0
signe mantisse, exposant, mantisse
1 bit 8 bit 23 bits

Le bit de signe est 1 pour négatif et 0 pour positif
La mantisse vaut toujours 1,xxxx et on ne stocke que xxxx
L'exposant est en excédent à 127
La valeur 0 correspond à des 0 partout (en fait $1,0.2^{-127}$)

	Encodage	**S**igne	**E**xposant	**M**antisse	Valeur d'un nombre	Précision	Chiffres significatifs
Simple précision	32 bits	1 bit	8 bits	23 bits	$(-1)^S.M.2^{(E-127)}$	24 bits	7
Double précision	64 bits	1 bit	11 bits	52 bits	$(-1)^S.M.2^{(E-1023)}$	53 bits	16

Base de Numération
Codage des nombres à virgule flottante

Codage des nombres à virgule flottante de simple précision

Représentation de l'exposant (IEEE 754, représentation normalisée)

- L'opération la plus fréquente est l'addition/soustraction → on n'utilise pas le complément à 2. Pour la simple précision, on utilise une représentation « excédent 127 » .

Exemple : 0 01111111 00000000000000000000000 = $1,0.2^0 = 1$

- La représentation est le nombre (positif) obtenu en additionnant 127 à la mantisse.
- De plus, les représentations 00000000 et 11111111 sont particulières.
- La représentation 11111110 indique un exposant de $254 - 127 = 127$
- La représentation 00000001 indique donc un exposant de $1 - 127 = -126$
- Le plus petit nombre représentable est donc $1 * 2^{-126}$
- Le plus grand nombre représentable est $1,11111111111111111111111 * 2^{127}$

Conception des systèmes numériques

2. Modélisation Numérique

Introduction au VHDL

Du cahier des charges à la réalisation

Problème
(Cahier des charges)

→ Conception Fonctions logiques

↓

Fonctions logiques simplifiées
coût / vitesse / encombrement / fiabilité ?

← Attention : critères pas toujours compatibles

↓

Réalisation Technologique

Attention: Méthodes «classiques» de simplifications :

- pas de solution unique
- indépendant de la technologie
- le temps n'est pas pris en compte

Contexte technologique

L'élément de base dans un processeur est la porte élémentaire. Plusieurs transistors sont associés avec notamment une capacité pour former cette porte élémentaire. La puissance consommée par un processeur peut être modélisée par l'équation définie dans [Mud01] par

$$P = A.C.V^2.f + t.A.V.I_{cc}.f + V.I_{fuite}$$

où :

– A : activité du processeur (nombre de portes élémentaires modifiées lors d'un coup d'horloge),

– C : capacité des portes élémentaires,

– V : tension d'alimentation du processeur,

– *f* : fréquence de cadencement du processeur,

– *t* : temps d'utilisation du processeur,

– I_{cc} *: courant de court-circuit (lors du basculement d'une porte élémentaire),*

– I_{fuite} *: courant de fuite (consommation du composant lorsque l'horloge est désactivée).*

Contexte technologique

$$P = A.C.V^2.f + t.A.V.I_{cc}.f + V.I_{fuite}$$

Le premier terme est appelé ***puissance dynamique et dépend*** de la fréquence d'utilisation ***f*** et **de la tension d'alimentation V du processeur.**

Le second terme est appelé ***puissance statique***

Le dernier terme est appelé ***puissance de fuite.***

Cette équation reste valable pour toutes les technologies CMOS utilisées pour la fabrication des processeurs. Les technologies CMOS sont actuellement encore les plus intéressantes en termes d'intégration.

Diminuer la tension d'alimentation V implique des améliorations sur la technologie des composants car il est impossible de diminuer V sans diminuer la fréquence

$$f_{max} = \frac{(V - V_{threshold})^2}{V},$$

où $V_{threshold}$ *est la tension de seuil entre le niveau haut (1 logique) et le niveau logique bas* (0 logique).

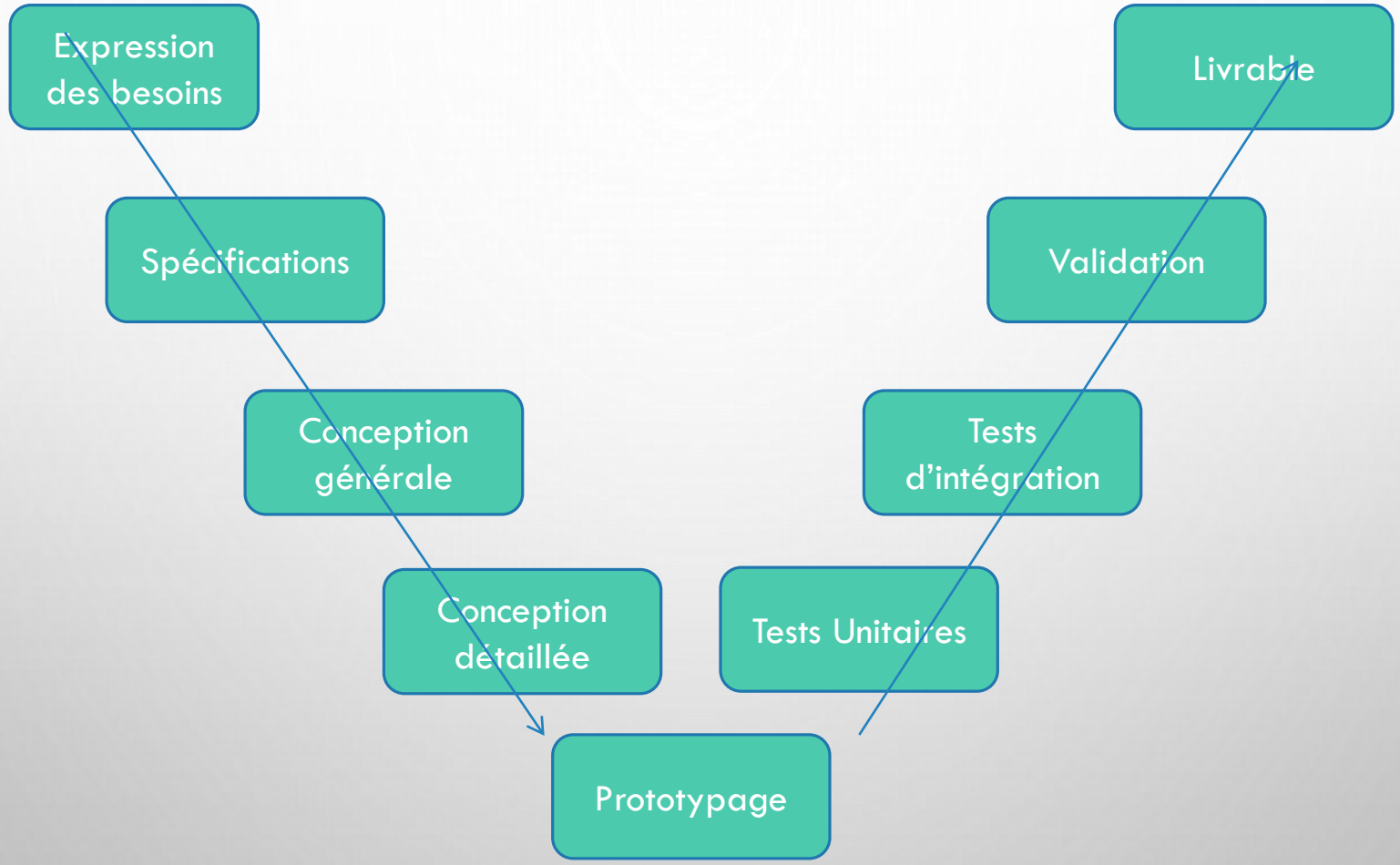
Contexte technologique

- Quels sont les leviers technologiques qui permettent d'accroitre la puissance de calcul dans un processeur ?

Pour respecter les limitations en consommation, il est préférable d'utiliser plusieurs cœurs cadencés à des fréquences moindres qu'un unique cœur cadencé à une forte fréquence.

→ Système multi-coeurs : il est beaucoup plus complexe de programmer plusieurs cœurs qu'un seul.
→ La technologie des composants évolue plus rapidement que les méthodes de conception.

Du cahier des charges à la réalisation



Du cahier des charges à la réalisation

Objectif d'une description logique:

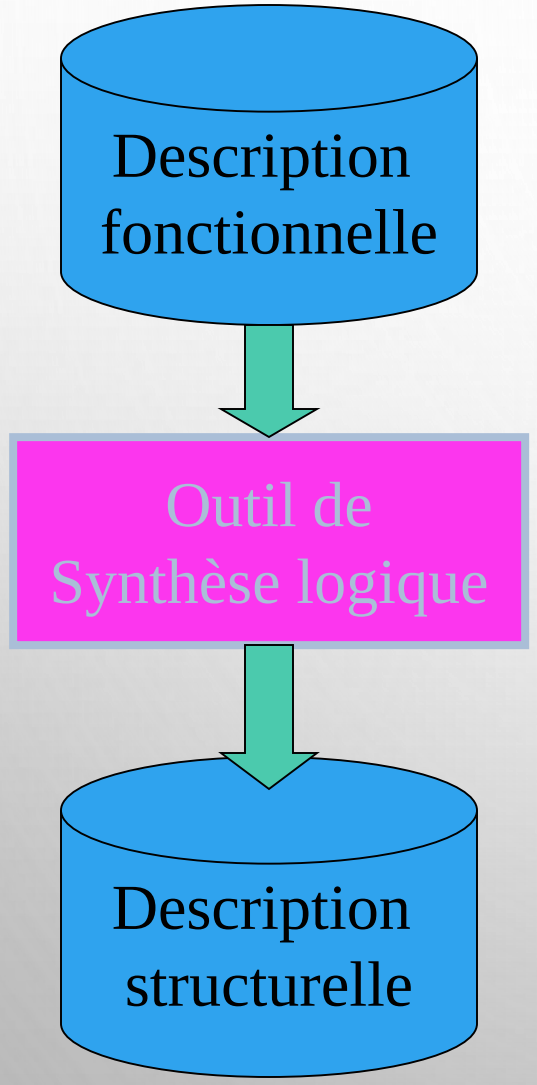
→ Saisir un circuit logique en vue de la simulation

→ Décrire un circuit pour documentation

→ Décrire un circuit en vue de synthèse logique

- Description structurelle :
  - ✓Objet décrit par ses composants
  - ✓Besoin : fabrication
- Description fonctionnelle :
  - ✓Objet décrit par le service qu'il rend
  - ✓Besoin : utilisateur
- La description peut aussi être mixte

Introduction aux langages de description : Outil de synthèse logique



Outil de synthèse logique :

Logiciel générant la description structurelle (**netlist**) à partir d'une description fonctionnelle

VHDL

- Initiateur : Department of Defense **(**DoD **– USA)**

→ Utilisateur de circuits provenant de nombreux fournisseurs

→ Longue durée de vie des produits (souvent > 40 ans)

- Financement: projet V.H.S.I.C.
- Résultat: langage V.H.D.L.
- VHDL est l'abréviation de (Very Hardware Description Language) → initialement VHSIC (Very High Speed Integrated Circuit)
- Un HDL est un langage de description matériel (Hardware Description Language)

Introduction aux langages de description : conception

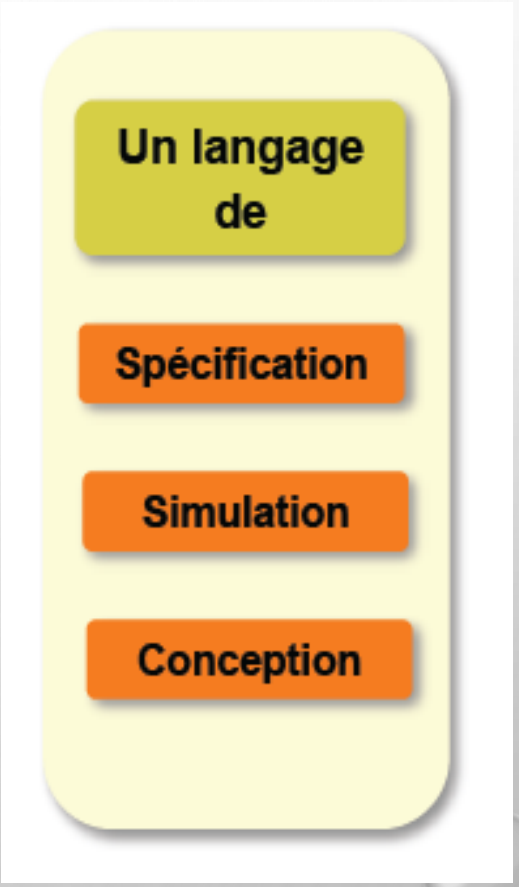
- C’est un standard dès l’origine : Initiative « ouverte »
- Concurrent Verilog (Société Cadence, System C, produit « propriétaire », mis dans le domaine publique)
- On distingue :

  → une partie numérique pure (VHDL’93)

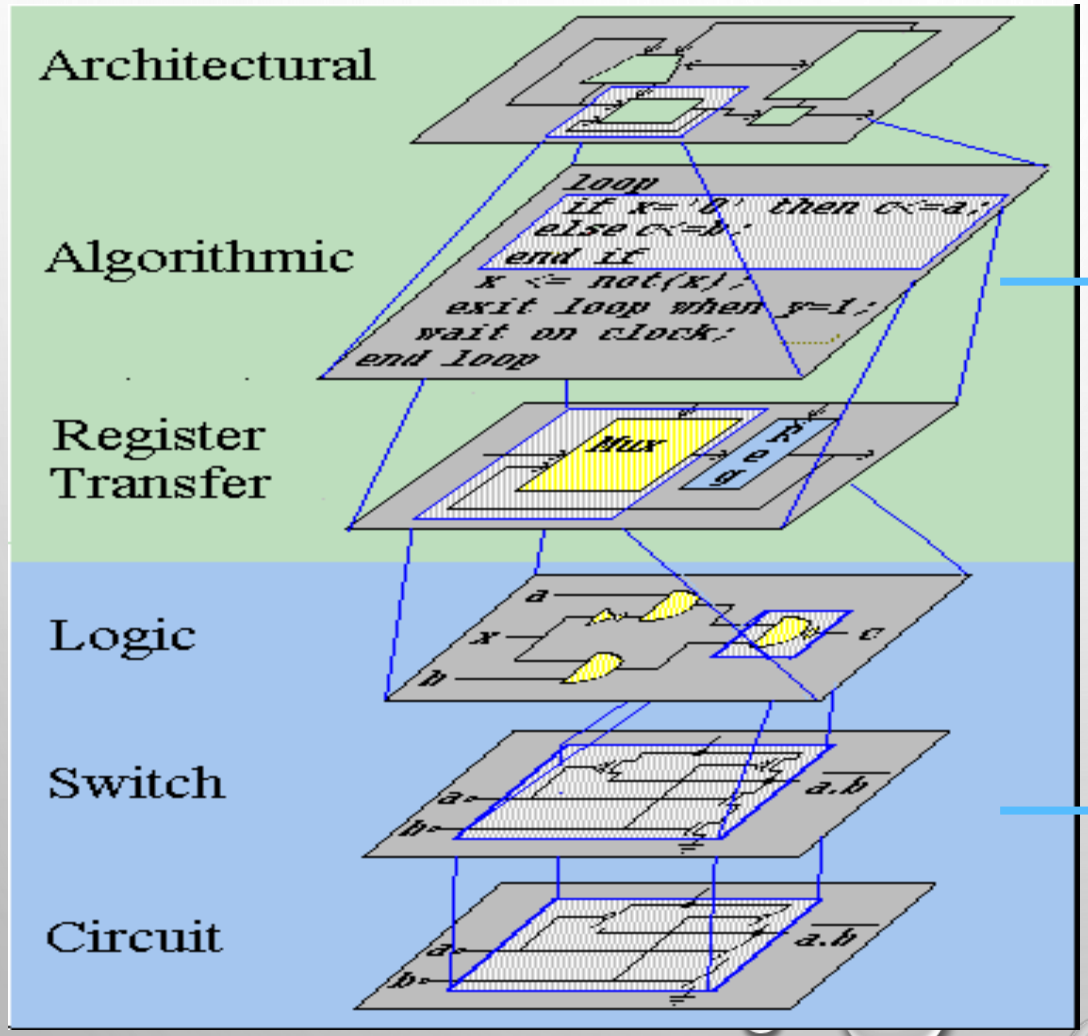
  → une extension analogique (VHDL-AMS : IEEE 1076.1-1999) (compatible VHDL’93)

Introduction à VHDL

- **Avantages**
  - **Simulation de systèmes complexes** :
    - Macroscopiques ou microscopiques
    - Modélisation d'ensembles de circuits
  - **Description structurée :**
    - Travail en équipe, séparation des tâches
    - Rapidité de conception
  - **Adaptation aux projets Multi-entreprise :**
    - Indépendance vis à vis de la technologie
    - Sécurité grâce à des modèles compilés
- **Inconvénients**
  - **Description complexe**
  - **Tout n'est pas synthétisable** (analogique)
    Prise en compte avec le VHDL-AMS (1997)

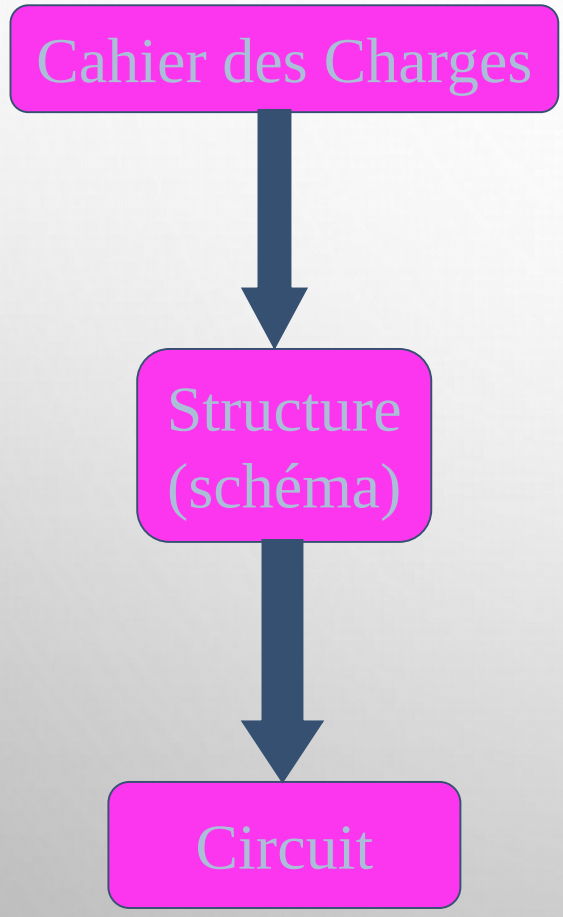


Introduction aux langages de description : Intérêts



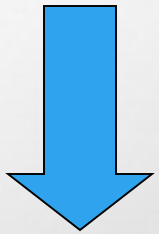
Niveaux principalement couverts par les H.D.L.

Introduction aux langages de description: Outil de synthèse logique



Comment s'y prendre?

HDL pour la synthèse

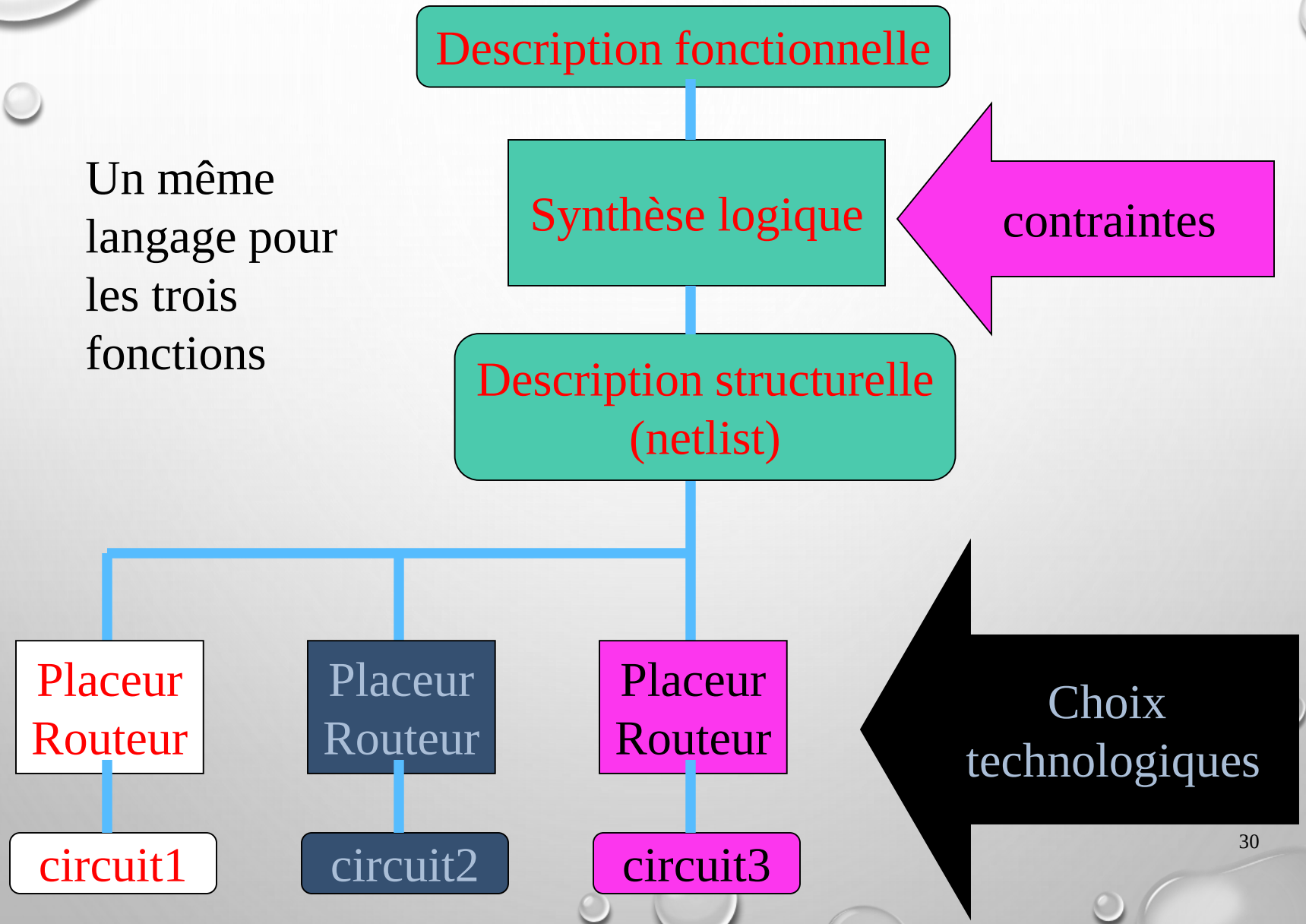


Cela va-t-il fonctionner?

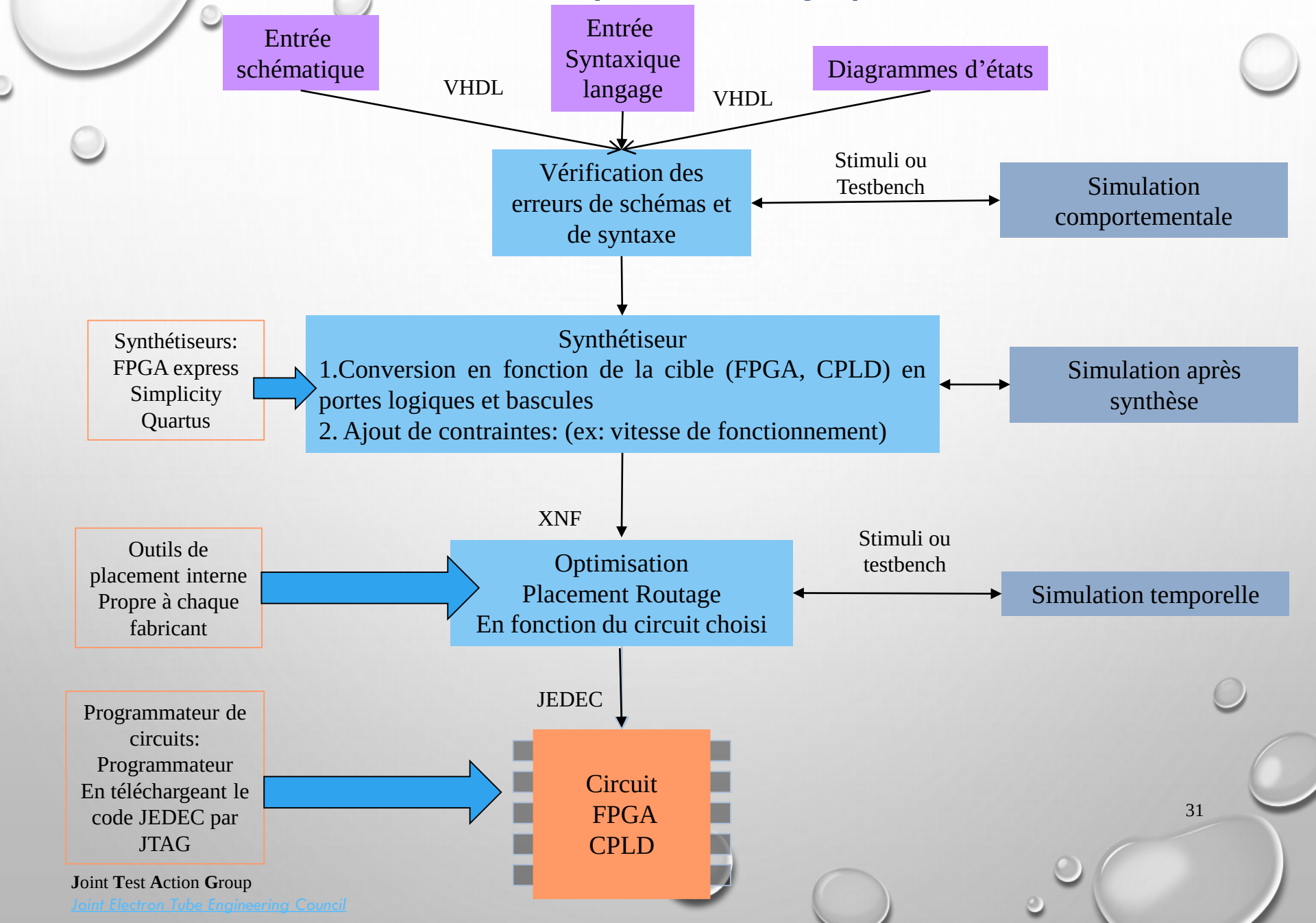
HDL pour la simulation

Conception indépendante de la technologie

Un même langage pour les trois fonctions



Outil de synthèse logique



Joint **T**est **A**ction **G**roup

Joint Electron Tube Engineering Council

Outil de synthèse logique : Liaison JTAG

L'interface JTAG vu depuis le composant permet de programmer et debugger, elle est :

- Transparente
- Asynchrone → tester le composant en cours de fonctionnement

4 Signaux :
TCK = 1 horloge TCK
TMS = 1 commande de contrôleur JTAG
TDI = charger une donnée dans un registre
TDO = lecture du contenu d'un registre

Introduction HDL
Présent et Futur

- Modélisation environnementale : processus autour du système
- Plusieurs disciplines (thermique, électrique, logique, …)
- Conception mixte

  → VHDL AMS (analogique, numérique)

  → Verilog Mixed Signal
- Co-Design (Matériel/Logiciel)
- Représentation hiérarchique
- Preuve formelle
- Travail grande équipe, optimisation (Time-to-Market)

Introduction VHDL
Architecture du langage

Une description **VHDL** est composée de 3 parties **indissociables** à savoir :

-***Library*** : déclare les bibliothèques

-***L'entité*** (***ENTITY***) : définit les entrées et les sorties

-***L'architecture*** (***ARCHITECTURE***) : contient les instructions **VHDL** permettant de réaliser le fonctionnement attendu.

Introduction à VHDL

Trois blocs de base

✓Les bibliothèques :

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

✓L'entité : décrit l'interfaçage du composant

```
entity MON-ET is
port( A : in std_logic;
 B : in std_logic;
 S : out std_logic);
end entity MON-ET;
```

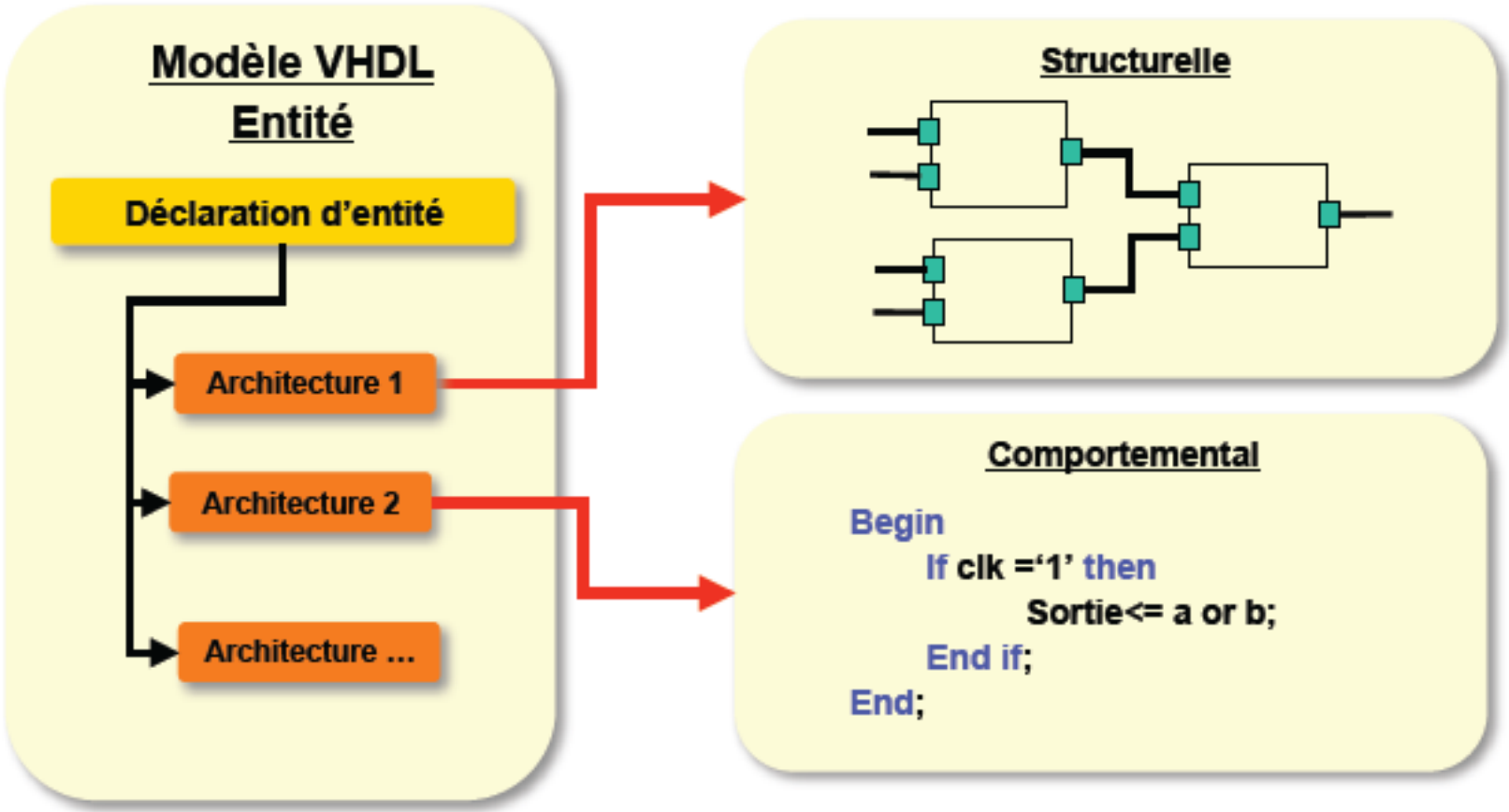
✓L'architecture : décrit le fonctionnement du composant

```
architecture comportement of MON-ET is
begin
S <= A and B;
end architecture comportement;
```

Introduction à VHDL

Structure d'un modèle VHDL

Le modèle peut avoir plusieurs descriptions adaptables à chaque projet



Introduction VHDL
Architecture du langage

✓Toute description **VHDL** a besoin de bibliothèques.

✓L'**IEEE** (**I**nstitut of **E**lectrical and **E**lectronics **E**ngineers) les a normalisées et plus particulièrement la bibliothèque **IEEE1164**.

✓Elles contiennent les définitions des types de signaux électroniques, des fonctions et sous programmes permettant de réaliser des opérations arithmétiques et logiques,...

Library ieee;
Use ieee.std_logic_1164.all;
Use ieee.numeric_std.all;
Use ieee.std_logic_unsigned.all;

La directive ***Use*** permet de sélectionner les bibliothèques à utiliser.

Architecture du langage

Déclaration de l'entité et des entrées / sorties

Elle permet de définir le ***NOM*** de la description **VHDL** ainsi que les entrées et sorties utilisées, ***Syntaxe:***

```
entity NOM_DE_L_ENTITE is
port ( Description des signaux d'entrées /sorties …);
end NOM_DE_L_ENTITE;
```

Exemple :

```
entity SEQUENCEMENT is
port (
CLOCK : in std_logic;
RESET : in std_logic;
Q : out std_logic_vector(1 downto 0) );
end SEQUENCEMENT;
```

Syntaxe: *NOM_DU_SIGNAL* **:** ***sens type;***

Exemple: *CLOCK*: ***in std_logic;***

BUS **:** ***out*** **std_logic_vector(7** ***downto*** **0);**

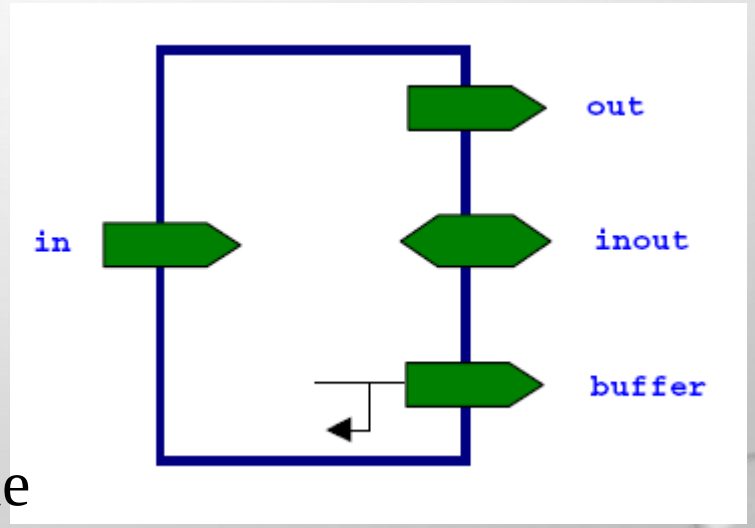
→ On doit définir pour chaque signal : le ***NOM_DU_SIGNAL,*** le ***sens*** et le ***type***.

Architecture du langage

Déclaration de l'entité et des entrées / sorties

Le *NOM* du signal.

Il est composé de caractères, le premier caractère doit être une lettre, sa longueur est quelconque, mais elle ne doit pas dépasser une ligne de code. **VHDL** n'est pas sensible à la « casse » (pas de distinction entre les majuscules et les minuscules.



Le *SENS* du signal.

- ***in*** : pour un signal en entrée.
- ***out*** : pour un signal en sortie.
- ***inout*** : pour un signal en entrée sortie
- ***buffer*** : pour un signal en sortie mais utilisé comme entrée dans la description.

Architecture du langage

L'architecture décrit le fonctionnement souhaité pour un circuit.
Le fonctionnement d'un circuit est généralement décrit par plusieurs modules **VHDL**. Il faut comprendre par module le couple ***ENTITE/ARCHITECTURE***.
L'architecture établit à travers les instructions les relations entre les entrées et les sorties. On peut avoir un fonctionnement purement combinatoire, séquentiel voire les deux.

Exemples :

```
-- Opérateurs logiques de base
entity PORTES is
port (A,B :in std_logic;
Y1,Y2,Y3,Y4,Y5,Y6,Y7:out std_logic);
end PORTES;
architecture DESCRIPTION of PORTES
is
begin
Y1 <= A and B;
Y2 <= A or B;
Y3 <= A xor B;
Y4 <= not A;
Y5 <= A nand B;
Y6 <= A nor B;
Y7 <= not(A xor B);
end DESCRIPTION;
```

Structure du modèle

Nom_du_modèle (signaux d'entrée, signaux de sortie)

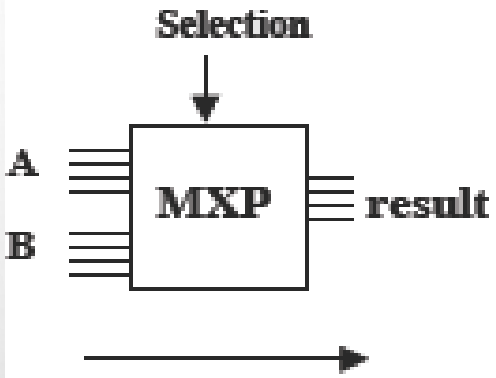
Architecture du modèle
- définition des signaux internes
- blocs de description

- Plusieurs architectures peuvent être décrites → choix à la compilation
- Lien symbolique avec les éléments de la bibliothèque redirection de connexions à la compilation
- 3 Niveaux de descriptions peuvent être utilisés
  - Comportemental
  - Flot de données
  - Structurel

Ces niveaux sont concourants

Les descriptions

Descriptions comportementales Algorithmes séquentiels




```
architecture nico of microP  is
---------
    signal A,B,result : bit_vector (1 to 4);
    selection : bit;
---------
MXP : process (Selection)
begin
    if (selection = '0') then
        result <= A after 10 ns;
    else
        result <= B after 15 ns;
    endif;
end process MXP;
--------
```

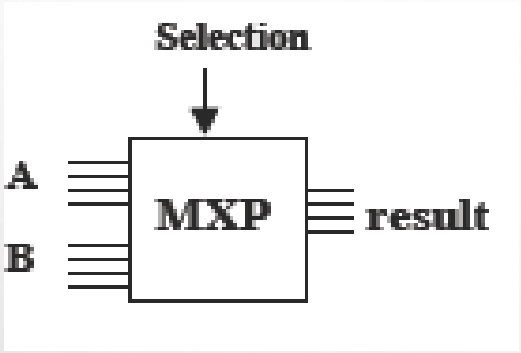
VHDL autorise :

- Structure de boucle (loop, endloop, while)
- Branchement (if then else, case endcase) .
- On utilise CASE pour permettre le choix entre plusieurs actions.
- Appel aux procédures et fonctions

Les descriptions

- Descriptions flots de données

  Équations concourantes




```
architecture nico2 of microP  is
.....
   signal A,B,result : bit_vector (1 to 4);
   selection : bit;
.....
result <= A after 10 ns when selection = '0' else
          B after 15 ns;
```

  Descriptions structurelles

Création d'un lien avec un autre modèle
Le modèle peut être
Soit un élément de la bibliothèque
Soit un élément de l'utilisateur

```
architecture nico3 of microP  is
.....
   signal A,B,result : bit_vector (1 to 4);
   selection : bit;
.....
G1 : MULTI4 port map ( A,B, selection, result);
```

Architecture du langage

Mode « concurrent », logique combinatoire.

Qu'est ce que le mode ***« concurrent »*** ?

- Pour une description **VHDL,** toutes les instructions sont évaluées et affectent les signaux de sortie en même temps.

→ L'ordre d'écriture n'a aucune importance.

- En effet, la description génère des structures électroniques, c'est la grande différence entre une description **VHDL** et un langage informatique classique.

- Dans un système à microprocesseur, les instructions sont exécutées les unes à la suite des autres

- Avec **VHDL,** il faut essayer de penser à la structure qui va être générée par le synthétiseur pour écrire une description

Architecture du langage

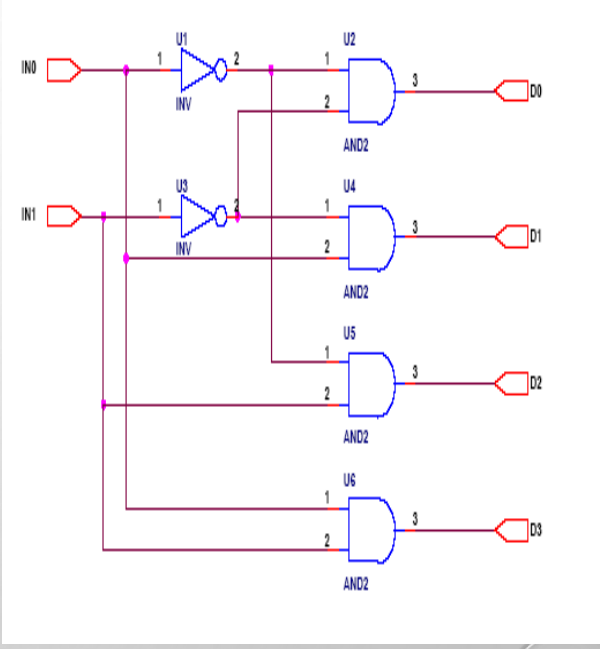
Mode « concurrent », logique combinatoire.

Exemple : Décodeur 2 vers 4 l'ordre dans lequel seront écrites les instructions n'a aucune importance.

```
architecture DESCRIPTION of DEMUX2_4 is
begin
D0 <= (not(IN1) and not(IN0)); -- première instruction
D1 <= (not(IN1) and IN0); -- deuxième instruction
D2 <= (IN1 and not(IN0)); -- troisième instruction
D3 <= (IN1 and IN0); -- quatrième instruction
end DESCRIPTION;
```

L'architecture ci dessous est équivalente :

```
architecture DESCRIPTION of DEMUX2_4 is
begin
D1 <= (not(IN1) and IN0); -- deuxième instruction
D2 <= (IN1 and not(IN0)); -- troisième instruction
D0 <= (not(IN1) and not(IN0)); -- première instruction
D3 <= (IN1 and IN0); -- quatrième instruction
end DESCRIPTION;
```



Architecture du langage

L'affectation simple : <=

Il permet de modifier l'état d'un signal en fonction d'autres signaux et/ou d'autres opérateurs.

Exemple avec des portes logiques : ***S1 <= E2 and E1 ;***

Les valeurs numériques que l'on peut affecter à un signal sont les suivantes :

- **'1'** ou **'H'** pour un niveau **haut** avec un signal de **1** bit.
- **'0'** ou **'L'** pour un niveau **bas** avec un signal de **1** bit.
- **'Z'** pour un état **haute impédance** avec un signal de **1** bit.
- **'-'** pour un état quelconque, c'est à dire **'0'** ou **'1'**. Cette valeur est très utilisée avec les instructions : ***when … else*** et ***with …. Select …***.
- Pour les **signaux** composés de plusieurs bits, on utilise les guillemets **" … "**
- Les bases numériques utilisées pour les bus peuvent être :

BINAIRE, exemple : ***BUS <= "1001" ; -- BUS = 9 en décimal***

HEXA, exemple : ***BUS <= X"9" ; -- BUS = 9 en décimal***

Architecture du langage

L'affectation simple : <=

Exemple:

```
Library ieee;
Use ieee.std_logic_1164.all;
entity AFFEC is
port (
E1,E2 : in std_logic;
BUS1,BUS2,BUS3 : out std_logic_vector(3 downto 0);
S1,S2,S3,S4 : out std_logic);
end AFFEC;
architecture DESCRIPTION of AFFEC is
begin
S1 <= '1'; -- S1 = 1
S2 <= '0'; -- S2 = 0
S3 <= E1; -- S3 = E1
S4 <= '1' when (E2 ='1') else 'Z'; -- S4 = 1 si E1=1 sinon S4 prend la valeur haute impédance
BUS1 <= "1000"; -- BUS1 = "1000"
BUS2 <= E1 & E2 & "10"; -- BUS2 = E1 & E2 & 10
BUS3 <= X"A"; -- valeur en HEXA -> BUS3 = A(déc)
end DESCRIPTION;
```

Architecture du langage

Définition d'un PROCESS.

Un **process** est une partie de la description d'un circuit dans laquelle les instructions sont exécutées séquentiellement. Il permet d'effectuer des opérations sur les signaux en utilisant les instructions standard de la programmation structurée comme dans les systèmes à microprocesseurs.

L'exécution d'un **process** est déclenchée par un ou des changements d'états de signaux logiques. Le nom de ces signaux est défini dans **la liste de sensibilité** lors de la déclaration du **process**.

[Nom_du_process :]
process(Liste_de_sensibilité_nom_des_signaux)
Begin
-- instructions du process
end process [Nom_du_process] ;

Architecture du langage

Définition d'un PROCESS.

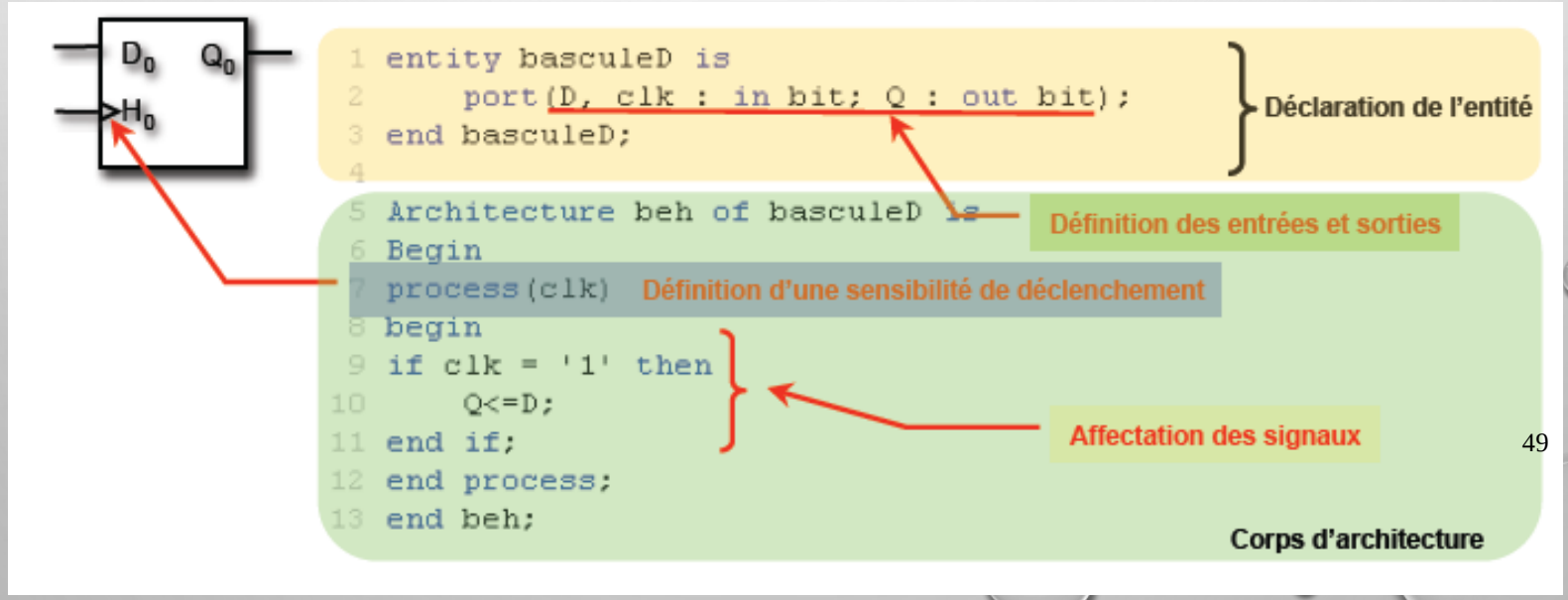
Règles de fonctionnement d'un process :

1) L'exécution d'un ***process*** a lieu à chaque changement d'état d'un signal de la liste de sensibilité.

2) Les instructions du ***process*** s'exécutent séquentiellement.

3) Les changements d'état des signaux par les instructions du ***process*** sont pris en compte à la **fin** du ***process***.

Exemple : bascule D - description comportementale



Architecture du langage

Les processus: exemple d'un compteur simple de 8 bits

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
ENTITY compt IS PORT (
h: IN STD_LOGIC;
q: OUT INTEGER RANGE 0 TO 7);
END compt;

ARCHITECTURE archi OF compt IS
SIGNAL test: INTEGER RANGE 0 TO 7;
BEGIN
PROCESS (h)
BEGIN
WAIT UNTIL h='1';
test <= test+ 1;
END PROCESS;
 q <= test;
END archi;
```

Architecture du langage

Les deux principales structures utilisées dans un process

L'assignation conditionnelle

```
if condition then
instructions
[elsif condition then instructions]
[else instructions]
end if ;
```

Exemple:

```
if (RESET='1') then SORTIE <=
"0000";
```

L'assignation sélective

```
case signal_de_selection is
when valeur_de_sélection => instructions
[when others => instructions]
end case;
```

Exemple:

```
case SEL is
when "000" => S1 <= E1;
when "001" => S1 <= '0';
when "010" | "011" => S1 <='1';
-- La barre | permet de réaliser un 'ou'
--logique entre les deux valeurs "010" et
--"011"
when others => S1 <= '0';
end case;
```

Remarque: ne pas confondre **=>** (implique) et **<=** (affecte).

VHDL : Les Littéraux

Valeurs décimales

Entiers : 12, 0 , 1E6,

Réels : 12.0 , 0.0, 0.456

Réels avec exposant: 1.23E-12 , 1.0e+6

Notation basée

format : base#valeur# base comprise entre 2 et 16

2#11111111#

16#FF# Entiers de valeur 255

Valeurs physiques

100 ps 3 ns 5 v

Il faut toujours un espace entre la valeur et l'unité

VHDL : Les Littéraux

Chaines de bits

Utilisées pour affecter des signaux de type bit_vector

X"FFF" -- base hexadécimale, longueur 12 bits

B"1111_1111_1111" -- base binaire, longueur 12 bits

Les objets et leurs types

• VHDL permet de manipuler des objets typés

• un objet est le contenant d'une valeur d'un type donné

• 4 classes d'objets :

– CONSTANT : objet possédant une valeur fixe

– VARIABLE : peut évoluer pendant la simulation

– SIGNAL : variable + notion temporelle (valeurs datées)

– FILE : ensemble de valeurs qui peuvent être lues ou écrites

Scalaires : entiers, réels...

– Composites : tableaux, articles, Pointeurs, Fichiers

VHDL : Les Littéraux

Les objets et leurs types

• Les types scalaires

• Types entiers

type integer is range -2_147_483_648 to 2_147_483_647; -- machine 32 bits

type Index is range (0 to 15);

• Types flottants

type real is range : -lim_inf to +lim_sup; -- prédéfini

type mon_real is range (0.0 to 13.8);

• Types énumérés

type bit is ('0','1'); -- prédéfini

type boolean is (false,true); -- prédéfini

type feu_tricolore is (vert,orange,rouge,panne);

type LOGIC4 is ('X','0','1','Z');

VHDL : Les Littéraux

Les objets et leurs types

• exemple

```
constant PI : real := 3.141592;

variable A,B,C : Integer;
variable DATA : integer := 0;
variable grand_rond : feu_tricolore := rouge;

signal enable : Logic4 := 'Z';
signal feu1,feu2 : feu_tricolore := panne;
```

Quand les objets ne sont pas initialisés, ils prennent la valeur la plus basse du type

VHDL : Les Tableaux

Les types tableaux (array)

- Le type Array réunit des objets de même type
- Un tableau se caractérise par :
  - sa dimension
  - le type de l'indice dans chacune de ses dimensions
  - le type de ses éléments
- Chaque élément est accessible par sa position (indice)
- On peut définir un type de tableaux non contraints, mais la dimension doit être spécifiée lors de la déclaration d'un objet de ce type

```
Type string is array (Positive range <>) of character; -- prédéfini
Type bit_vector is array (Natural range <>) of bit; -- prédéfini
Type Compte_feu is array (Feu_tricolore) of integer;
Type Matrice2D is array (Positive range <>,Positive range <>) of real;
Signal mat3*4 : Matrice2D ( 1 to 3, 1 to 4);
Signal mot16 : bit_vector (15 downto 0);
```

VHDL : Les records

- Le type RECORD réunit des objets de types différents
- Les éléments (champs) sont accessibles par un nom
- La déclaration d'un article inclut la déclaration de chacun de ses champs

```
Type type_mois is (jan,fev,mars,avr,mai,juin,juil,aout,sept,oct,nov,dec);
Type date is record
                    mois : type_mois;
                    annee : natural;
End record;
Type personne is record
                    nom : string;
                    prenom : string;
                    age : natural;
                    arrivee : date;
End record;
```

VHDL : Les records

Les sous types

Les sous-types

- Association d'une contrainte à un type
- La contrainte est optionnelle
- Le simulateur vérifie dynamiquement la valeur de l'objet
- Les opérateurs définis pour le type sont utilisables pour le sous-type

```
Subtype printemps is type_mois range mars to juin ;
Subtype valeur is bit;
Subtype octet is bit_vector ( 7 downto 0);
Subtype byte is bit_vector (7 downto 0);
```

VHDL : les opérateurs logiques

AND, NAND, OR, NOR, XOR et NOT

- utilisés pour des objets de type bit, booléen ou tableaux unidimensionnels de ces types
- utilisation des opérateurs optimisée par le simulateur

```
signal A,B,C,S : bit;
variable raz, init, marche : boolean;
S <= A or ( B nand C ) after 10 ns;
raz := init and not marche ;
```

VHDL : les opérateurs logiques

Les opérateurs arithmétiques

- +, -, /, MOD,, ABS, ** (MOD est le reste de la division euclidienne)
- addition et soustraction définies pour tous les types numériques
- multiplication et division s'appliquent à 2 réels ou entiers ou 1 objet physique et un réel entier
- MOD défini pour le type entier
- ** élève entier ou réel à une puissance entière

```
signal A,B,C: real;
signal I,J,K : integer;
A <= B + C;
I <= J + K;
A <= B + real(J);
 I <= 5; J <=- 2;
K <= I/J; -- K = -2
```

Opérateurs relationnels

=	égal	tous les types retournent un booléen
/=	différent	
<	inférieur	
>	supérieur	
<=	inférieur ou égal	Scalaire retourne un booléen
>=	supérieur ou égal	

VHDL : les opérateurs logiques

La concaténation

- **Utilise l'opérateur & et s'applique aux vecteurs de taille quelconque**
- **concaténation de 2 vecteurs, d'un élément et d'un vecteur ou de 2 éléments**

```
"CONCA"&"TENATION" -> "CONCATENATION"
'A' & 'B' -> "AB"
'0' & "1011" & '0' -> "010110"
REG ( 31 downto 0 ) <= GAUCHE & REG (31 downto 1);
```

VHDL : Instructions concurrentes

• Le VHDL est un langage concurrent ➠ le système à modéliser peut être divisé en plusieurs blocs agissant en parallèle

• L'instruction concurrente est exécutée quand un évènement sur certains de ses signaux apparait

➠ **Instanciation de composants**
➠ **Process**
➠ **Affectation de signaux**
➠ **Appel de procédure**
➠ **Assertion**
➠ **Bloc**
➠ **Génération**

VHDL : *Les processus*

Les processus constituent les éléments calculatoires de base du simulateur.

Au niveau descriptif, ils peuvent être explicites (PROCESS) ou implicites (instructions concurrentes).

Un processus tourne toujours depuis le chargement du code exécutable dans le simulateur. Il ne peut prendre fin qu'avec la simulation mais peut par contre être endormi pour une durée plus ou moins longue.

L'instruction WAIT, synchronise le processus.

```
SIGNAL h : bit;
BEGIN
horloge : PROCESS
BEGIN
h <= '0', '1' AFTER 75 ns;
WAIT FOR 100 ns;
END PROCESS;
```

Ce processus produit un signal répétitif de période 100 ns avec 75ns pour le niveau bas et 25 ns pour le niveau haut.

VHDL : ***Les processus***

Une variable ne peut exister que dans un contexte séquentiel, elle est affectée immédiatement. Elle n'est pas visible à l'extérieur d'un processus

Le signal est le seul objet qui peut être affecté soit de façon concurrente, soit de façon séquentielle selon le contexte.

L'affectation du signal est différée à cause de son pilote.

Lors de l'affectation séquentielle, le ou les couples *valeur_future: heure_de_simulation sont placés dans le pilote. La valeur sera passée au signal au moment de la suspension du Process par une instruction WAIT.*

C'est le pilote du signal qui est affecté et non le signal lui-même »

VHDL : *Les processus*

Exemple:

```
ENTITY varsig IS
END;
ARCHITECTURE exercise OF varsig IS
SIGNAL aa, : INTEGER;:=1
SIGNAL bb: INTEGER; :=2
BEGIN
P1: PROCESS
VARIABLE a: INTEGER :=7; VARIABLE b: INTEGER :=6;
BEGIN
WAIT FOR 10 ns;
                    a := 1; --- a est égal à 1
                    b := a + 8 ; --- b est égal à 9
                    a := b - 2 ; --- a est égal à 7
                    aa <= a; -- 7 dans pilote de aa
                    bb <= b ; -- 9 dans pilote de bb
END PROCESS;
```

De deux affectations successives d'un même signal, seule la deuxième compte. **Le pilote du signal constitue une mémoire associée au signal.**

à l'heure H = 10 ns , aa prend la valeur 7, bb prend la valeur 9 entre 0 et 10 ns aa vaut 1 et bb vaut 2

VHDL : *Les processus*

D’un **point de vue interne**, un processus est constitué d'**instructions séquentielles**, à l'instar des instructions d'un langage impératif classique (C, …)
Il faut raisonner comme si le processus correspondait à un programme d'instructions séquentielles exécutées par un simulateur, mais penser qu’en synthèse le processus génère des connexions matérielles

VHDL : *Les processus*

Un processus est **activé** à chaque **changement d'état** de l'un quelconque des signaux auxquels il est déclaré sensible ; en conséquence, toutes ses instructions sont scrutées

Une **liste de sensibilité** est constituée

- pour les fonctions combinatoires : de **tous les signaux lus par le processus**
- pour les fonctions séquentielles synchrones : de **l'horloge** et des **signaux asynchrones** (set, reset)

Au cours de l'activation du processus, chaque signal, référencé dans la partie droite d'une instruction d'assignation, prend une **valeur courante** qu'il conservera tout au long du déroulement du processus

VHDL : *Les processus*

- Toute instruction d'assignation de signal porte sur la **valeur courante** des signaux qui se situent à droite de l'opérateur d'assignation <=
- Les instructions déterminent les signaux à modifier et **planifient** leur prochaine valeur
- Les nouvelles valeurs sont calculées au fur et à mesure des assignations, et rangées dans une zone temporaire
- L'attribution définitive des nouvelles valeurs est faite **à la fin du processus**, et au même moment pour tous les signaux modifiés par les instructions d'assignation

Assignation de tous les signaux, en même temps, à la fin du processus.

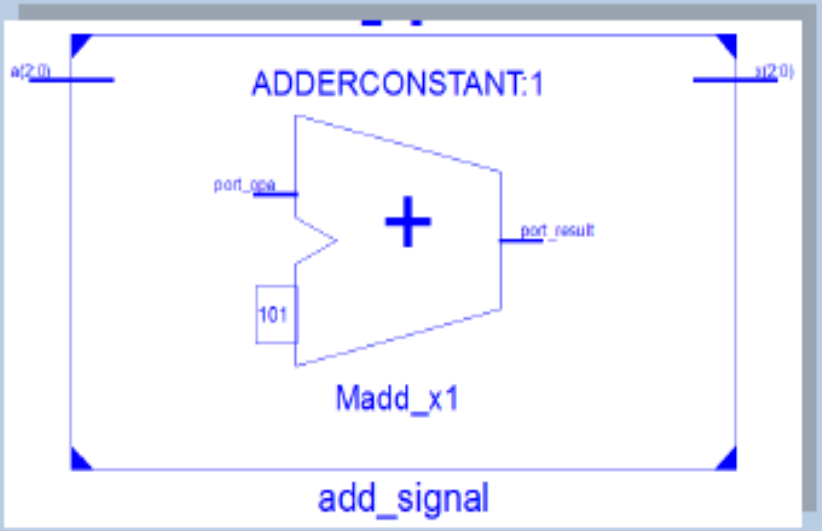
Traduit une simultanéité d'exécution; donc adapté à une description de matériel où tout est parallélisme

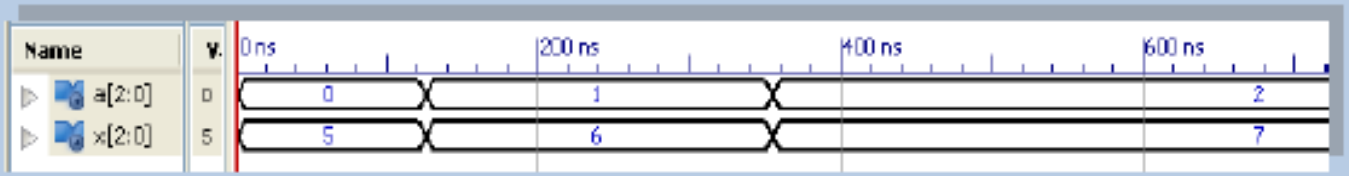
```
shift_proc : process (clk)
begin
…
x(0) <= serial_input;
x(1) <= x(0);
x(2) <= x(1);
end process;
```

Les valeurs de ces signaux sont celles prises à l'activation du processus, cela pour toutes leurs occurrences.
Au moment du traitement de la 2e ligne, x(0) n'a pas encore pris la valeur de serial_input.

VHDL : *Les processus*

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

entity add_signal is
    port ( a : in  std_logic_vector (2 downto 0);
           x : out  std_logic_vector (2 downto 0));
end add_signal;

architecture behavioral of add_signal is
begin

process (a,b)
begin
      x <= a + 1;
      x <= a + 5;
end process;

end behavioral;
```





Dans cet exemple **la dernière assignation l'emporte**, la séquence ne génère aucune erreur, **mais n'a aucune utilité** !

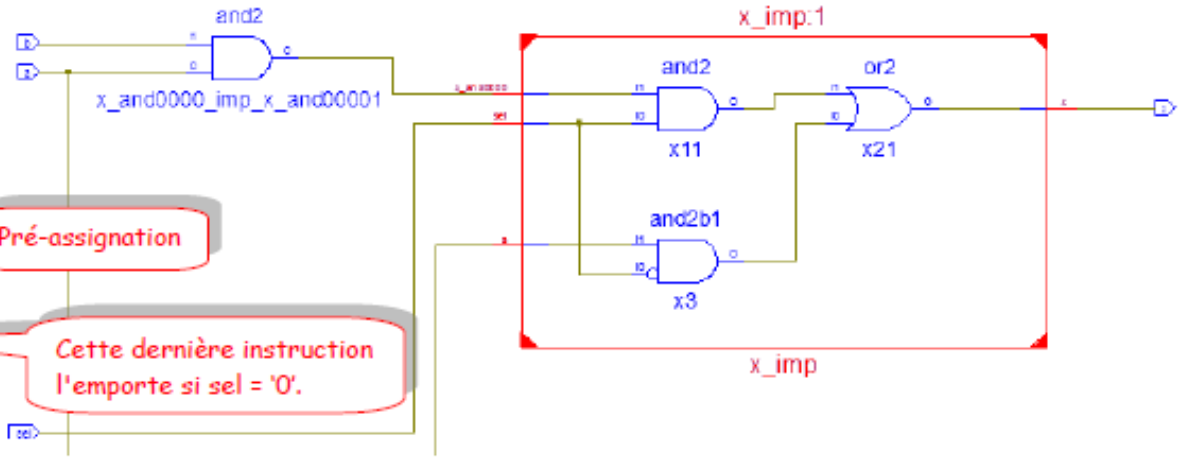
VHDL : *Les processus*

- **Influence de l'ordre des instructions**

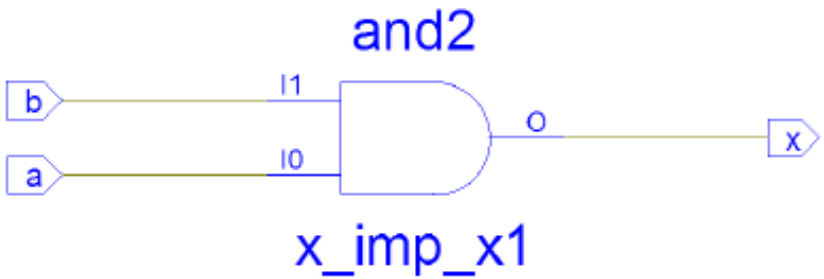
```
process(a,b,sel)

begin
        x <= a and b;
        if sel = '0' then
                x <= a;
        end if;
end process;
```

Pré-assignation

Cette dernière instruction l'emporte si sel = '0'.




```
process(a,b,sel)

begin
        if sel = '0' then
                x <= a;
        end if;
        x <= a and b;
end process;
```



Cette dernière instruction l'emporte systématiquement, donc l'instruction if ne sert à rien !!

VHDL : *Les processus*

Comportement d'un processus du point de vue des variables

Contrairement aux signaux, les variables sont mises à jour immédiatement, au fur et à mesure de leur assignation par l'opérateur := (Rappel : pour un signal, c'est à la fin du processus)

Les modifications des variables se propagent immédiatement vers les instructions suivantes comme dans un langage impératif classique

Règles pour les variables

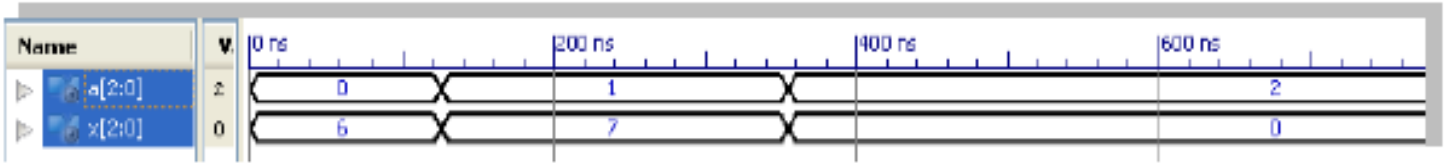
Les variables sont déclarées **uniquement à l'intérieur** du processus

Avant de lire une variable pour la première fois, il faut préalablement l'**assigner**

VHDL : *Les processus*

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

entity add_variable is
    Port ( a : in  std_logic_vector (2 downto 0);
           x : out  std_logic_vector (2 downto 0));
end add_variable;

architecture Behavioral of add_variable is
begin
process (a)
      variable v : std_logic_vector(2 downto 0) := "000";
begin
      v := a + 1;
      v := v + 5;
      x <= v;
end process;
end Behavioral;
```

Les modifications de la variable v se propagent d'instruction en instruction



Affectations concurrentes de signaux

Ces affectations sont des instructions concurrentes comme les processus et les instanciations de composants.
Le codage comportemental d'une architecture repose sur l'utilisation de ces 3 types d'instructions concurrentes qui peuvent apparaître dans n'importe quel ordre.

Architecture avec processus

```
architecture arc of adder is begin
process(A,B,Cin)
begin S <= A xor B xor Cin;
end process
```

Architecture avec affectation concurrente

```
architecture arc of adder is begin
S <= A xor B xor Cin;
end arc;
```

Affectations concurrentes de signaux

Affectation concurrente conditionnelle

Il existe également des instructions concurrentes conditionnelles permettant d'effectuer des raccourcis d'écriture pour remplacer des processus simples à base de IF et CASE

Affectations concurrentes de signaux

Architecture avec processus

```
architecture arc of adder is
Begin
 process(A,B,Cin)
Begin
   if A = '0' then S <= B xor Cin;
   elsif B = '0' then S <= not(Cin);
   else S <= Cin;
  end if;
end process;
 end arc;
```

```
architecture arc of MUX is begin process(SEL)
 begin
case SEL is
when 0 => sortie <= A; when 1 => sortie
<= B; when 2 => sortie <= C;
when others => sortie <= D; end case; end
process; end arc;
```

Architecture avec affectation concurrente

```
architecture arc of adder is begin
S <= B xor Cin when A = '0'
else not Cin when B = '0'
else Cin;
```

```
architecture arc of MUX is
Begin
 with SEL select
sortie <= A when 0,
B when 1, C when 2,
D when others; end arc;
```

VHDL : *Les processus*

Ce qu'il faut retenir :

	Le signal	La variable
Symbole d'assignation	<=	:=
Champ d'action	Global (dans l'ensemble du code VHDL).	Local ; utilisée uniquement à l'intérieur d'un processus, d'une fonction ou d'une procédure.
Fonction	Représente une interconnexion physique de circuits.	Représente une information utile pour l'exécution d'un algorithme (n'induit pas nécessairement une interconnexion physique).
Comportement	S'il se trouve dans le code séquentiel d'un processus, sa mise à jour a lieu à la fin.	La mise à jour est immédiate (la nouvelle valeur est disponible dans la ligne suivante de code).

Logiciel : Quartus

Quartus est un logiciel développé par la société Altera

→ la gestion complète d'un flot de conception CPLD ou FPGA.

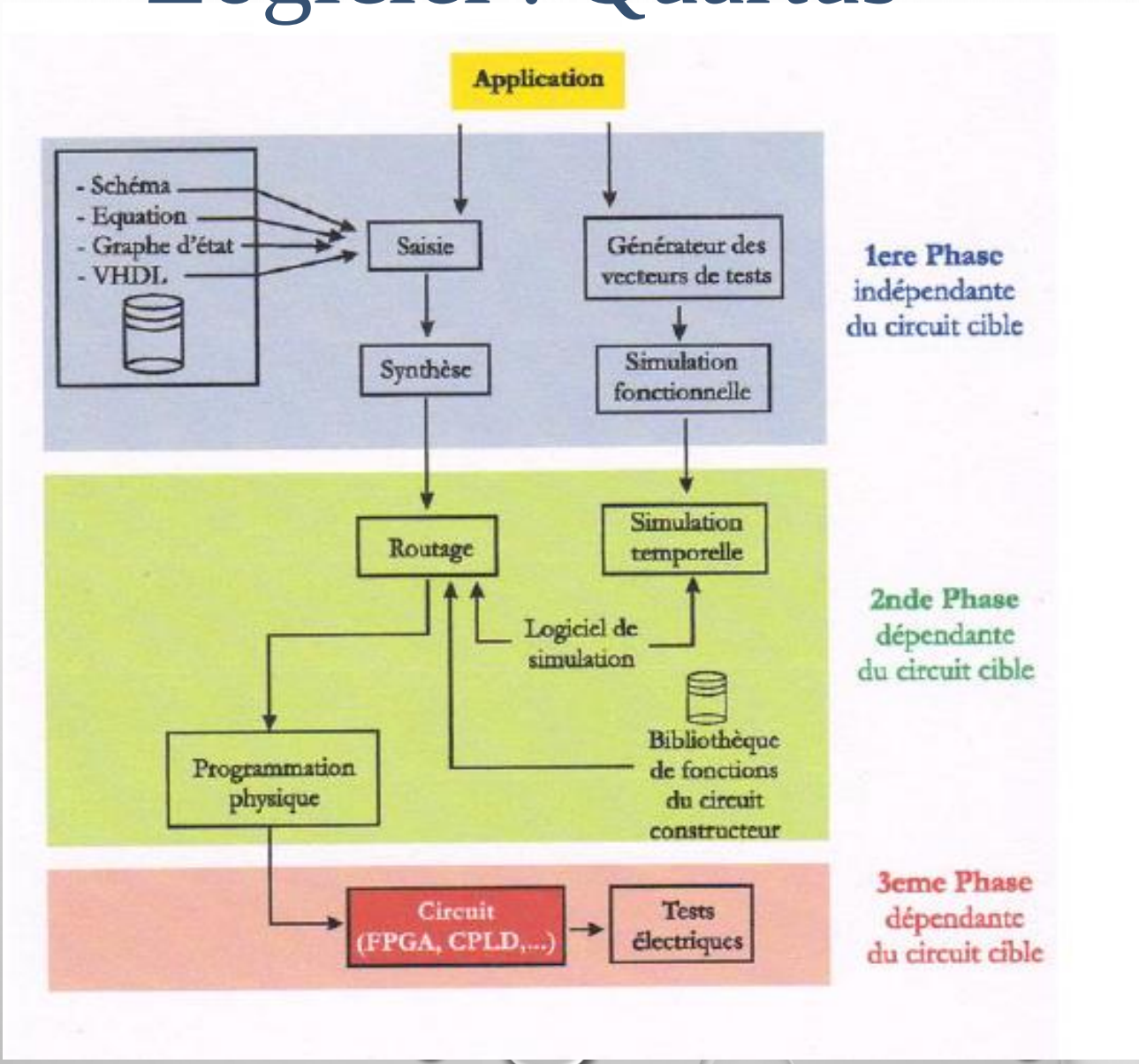
→ saisie graphique ou une description HDL (VHDL ou verilog) d'architecture numérique

→ Réalisation d'une simulation, une synthèse et une implémentation sur cible reprogrammable.

Il comprend :

- une suite de fonctions de conception au niveau système, permettant d'accéder à la large bibliothèque d'IP d'Altera
- un moteur de placement-routage intégrant la technologie d'optimisation de la synthèse physique et des solutions de vérification.

Logiciel : Quartus



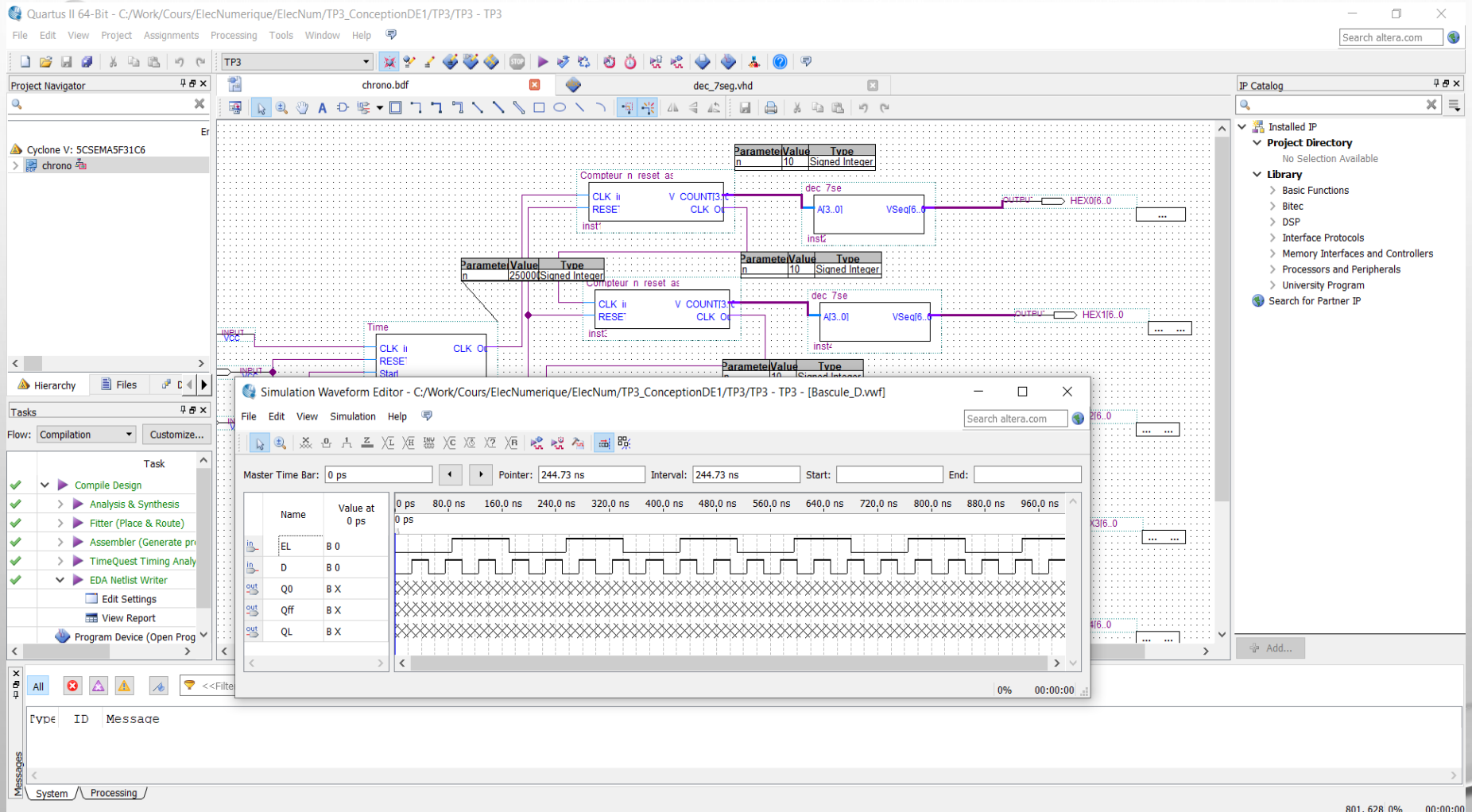
Logiciel : Quartus

Quartus est un logiciel qui travaille sous forme de projets c'est à dire qu'il gère un système avec un ‘design’ sous forme d'entités hiérarchiques.

Un projet est l'ensemble des fichiers d'un système que ce soit

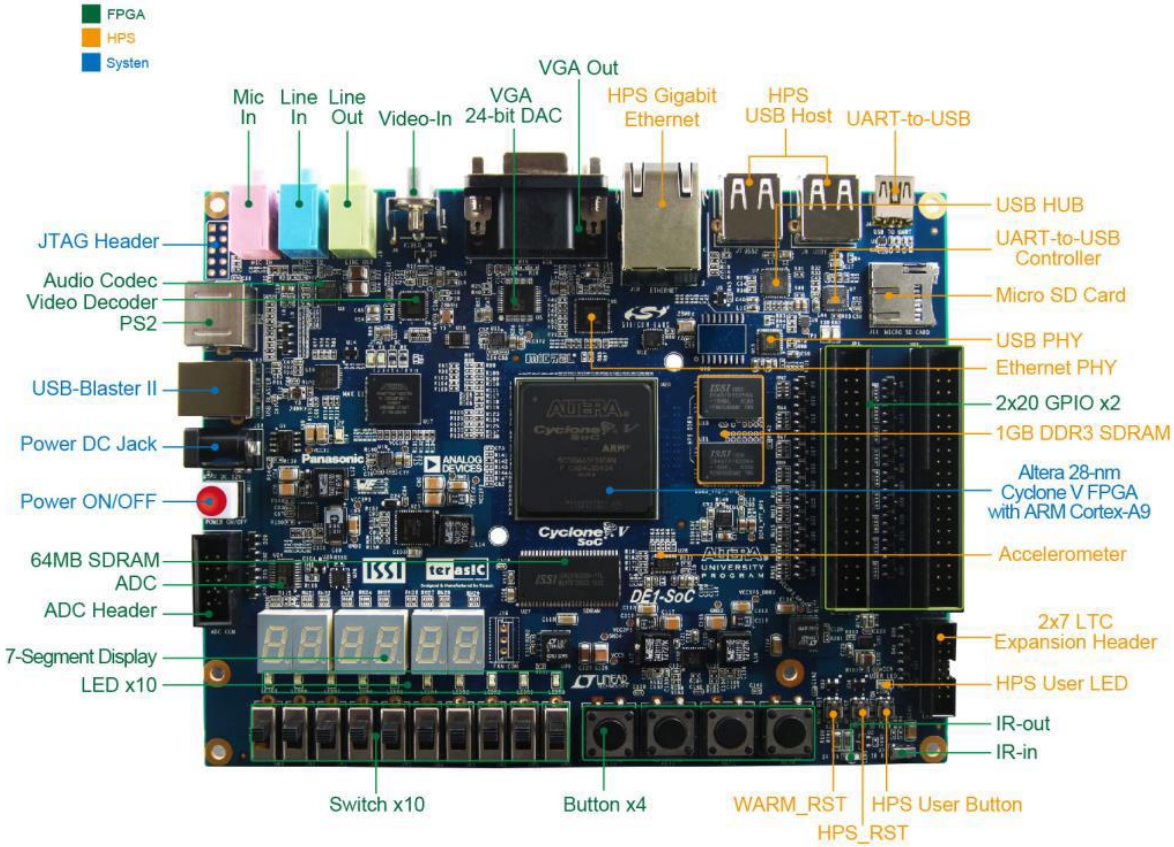
- des saisies graphiques
- des fichiers VHDL
- ou bien encore des configurations de composants (affectation de pins par exemple).

Logiciel: Quartus



CARTE FPGA

LA CARTE DE1-SoC (UTILISÉ EN TP)



Specifications

FPGA

• Cyclone V SoC 5CSEMA5F31 FPGA

I/O Devices

• Two port USB 2.0 Host
• 10/100/1000 Ethernet
• PS/2 mouse or keyboard port

Memory

• 64MB (32Mx16) SDRAM
• Micro SD card socket on HPS

Switches, LEDs, Displays, and Clocks

• 10 toggle switches
• 4 pushbutton switches
• 10 LEDs
• Six 7-segment displays
• Four 50-MHz clock sources from clock generator