La République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université Ammar Telidji de Laghouat
Faculté de Technologie
Département d'Electronique

Support de cours

FPGA et VHDL

1ére année Master
Electronique des systèmes embarqués
Systèmes de télécommunication

Réalisé par : Dr. Lahcene Merah

Année universitaire 2022/2023

Introduction

Aujourd'hui, les circuits FPGA devenus véritablement des dispositifs révolutionnaires qui combinent les avantages du hardware et logiciel. Ils implémentent des circuits comme faire le hardware, offrant des avantages considérables en termes de puissance, de complexité et de performance par rapport aux logiciels, mais peuvent être reprogrammés à moindre coût et facilement pour mettre en œuvre un large éventail de tâches.

Ce support de cours est destiné aux étudiants de Master en électronique et télécommunication. Il contient une bonne introduction aux dispositifs numériques programmables et plus spécifiquement les circuits FPGA. Le support commence par donner une idée sur la constitution des portes logiques à base des transistors. Ensuite, présenter l'évolution des circuits intégrés, commençant par les circuits standards, les circuits ASIC, et finalement l'origine des FPGAs (circuits logiques programmables).

Une partie de ce support de cours sera consacrée à donner une vue générale sur les circuits FPGA. Un FPGA (Field Programmable Gate Array, qu'on peut traduire par matrice de portes programmables sur site) est, pour aller vite, un circuit logique programmable, ou configurable, contenant quelques milliers à quelques dizaines de milliers de blocs logiques. Chaque bloc logique peut contenir une fonction combinatoire de quelques entrées (4 à 8) et de 1 à 4 bascules D, selon les générations.

La dernière partie de ce support sera consacré à présenter le langage VHDL utilisé pour la conception, le développement, et la configuration des circuits FPGA. Le VHDL est une langage de description matèriel le plus utilisé pour le développement des FPGA. VHDL signifie Very high speed integrated circuit Hardware Description Language.

Chapitre 1 : Notions sur les circuits numériques

1. Technologies d'implémentation

L'élément de base dans l'électronique numérique c'est le bit. Un bit est représenté par un signal électrique. Ces signaux électriques en effet sont des niveaux de tension. Les deux valeurs logiques 0 et 1 sont représentées comme des niveaux de tension différents.

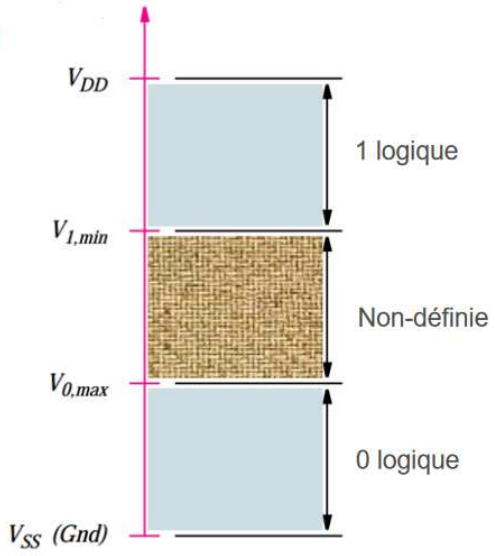


Fig.1. Les plages des niveaux de tension pour chaque état logique.

Les niveaux de tension associés à chaque variable logique dépendent du la technologie utilisée. La figure suivante illustre les plages des niveaux de tension pour chaque technologie :

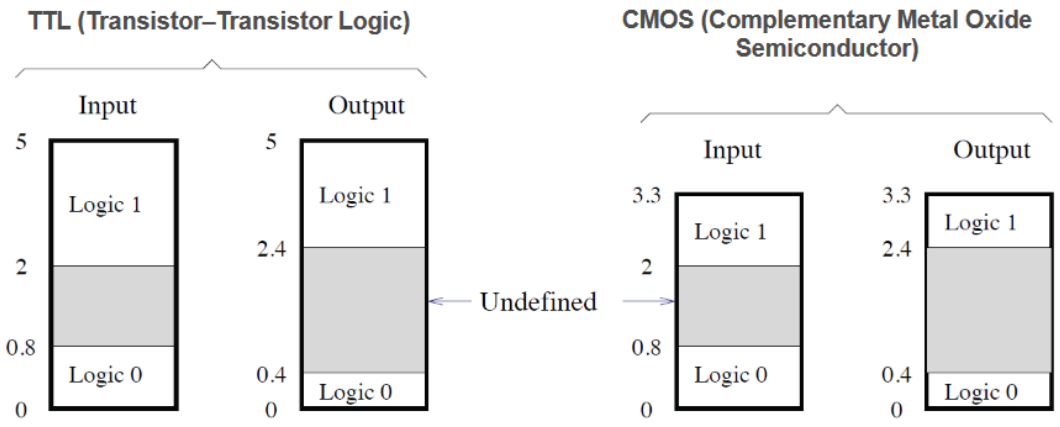
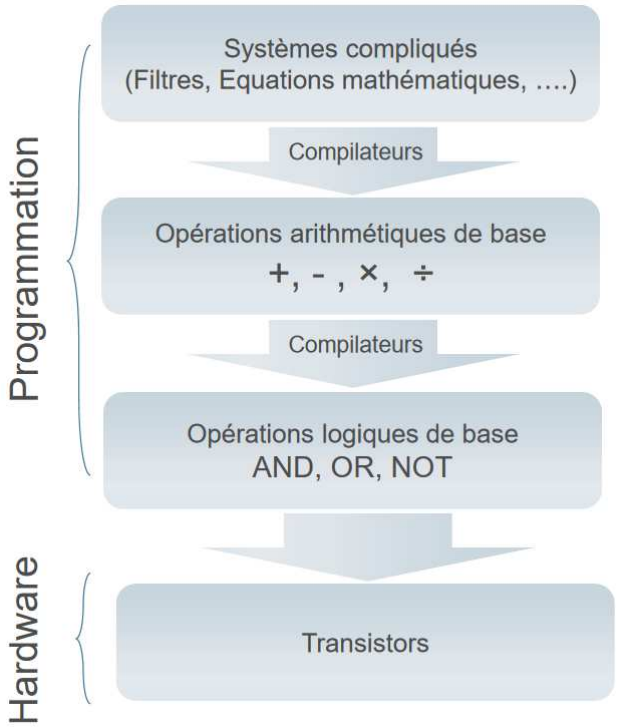


Fig.2. Les niveaux de tension selon la technologie utilisée.

2. Implémentation des fonctions numériques

N'importe quel système numérique et que ce soit sa degré de complexité, Il est essentiellement composé d'un ensemble des fonctions mathématiques. Par la suite, d'un point de vue d'implémentation, n'importe quelle fonction mathématique, elle est essentiellement composée d'un ensemble des opérations arithmétiques de base. Les opérations arithmétiques par la suite sont composées d'un ensemble des fonctions logiques de base. Finalement, et d'un point de vue hardware, les opérateurs logiques de base sont implémentées à l'aide des transistors, et ces derniers sont l'élément de base pour l'implémentation d'un système numérique. L'implémentation des systèmes numérique se fait à l'aide de programmation, et les compilateurs facilitent le passage d'un niveau haut (niveau algorithmique) à un niveau bas (langage machine).



Les compilateurs contient des *packages* (bibliothèques), permettant de d'analyser les programmes et convertir les

A titre d'exemple, les fonctions mathématiques connues peuvent être analysées et transformées aux opérations arithmétiques (développent de Taylor comme exemple) :

$$\exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \cdots + \frac{x^n}{n!} + o(x^n) .$$

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} + \cdots + \frac{(-1)^n x^{2n+1}}{(2n+1)!} + o(x^{2n+1}) .$$

$$\cos(x) = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} + \cdots + \frac{(-1)^n x^{2n}}{(2n)!} + o(x^{2n}) .$$

$$\frac{1}{1-x} = 1 + x + x^2 + \cdots + x^n + o(x^n) .$$

$$(1+x)^\alpha = 1 + \alpha x + \frac{\alpha(\alpha-1)}{2!} x^2 + \cdots + \frac{\alpha(\alpha-1)\cdots(\alpha-n+1)}{n!} x^n + o(x^n) .$$

Encore, les opérations arithmétiques peuvent être analysées et transformées aux opérations logiques, à titre d'exemple, l'addition peut être implémentée à les des opérations logiques suivantes :

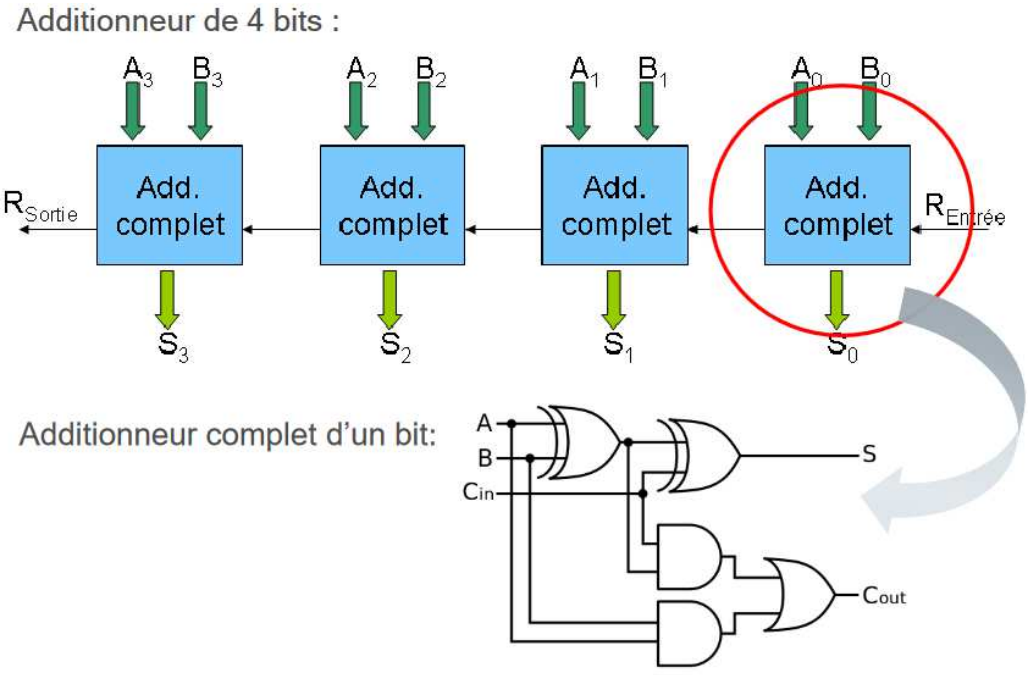


Fig.4. Circuit logique d'un additionneur de 4 bits.

L'image suivante représente le circuit logique d'un multiplier à 3 bits :

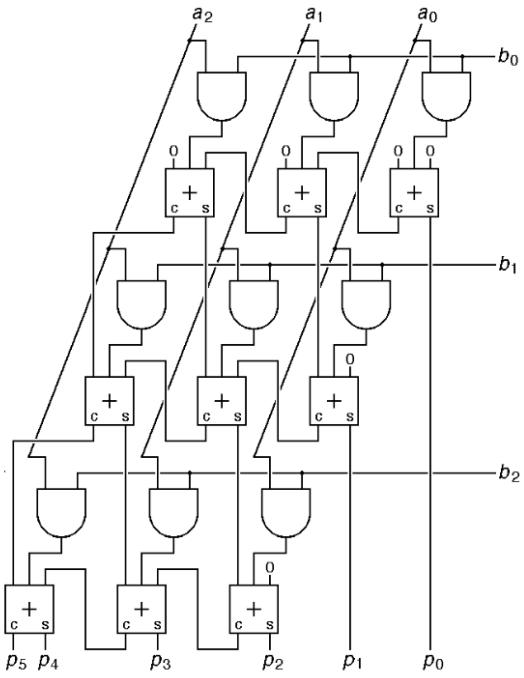


Fig.5. Circuit logique implémentant un multiplieur de 3bits.

3. Constitution des portes logiques à base des transistors

D'un point de vue général, un transistor peut agir comme un commutateur. Le transistor le plus populaire c'est : Le MOSFET (Metal Oxide Semiconductor Field-Effect Transistor). Il existe deux type de transistor MOSFET; nMOS (n-channel) et pMOS (p channel). nMOS : à 4 connections; source, drain, grille (gate), et substrat. Généralement le substrat est connecté à la masse.

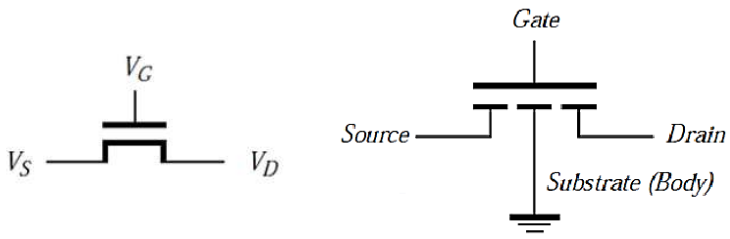


Fig.6. Symboles simplifiés d'un transistor de type nMOS.

En bref, le nMOS est commandé par la tension V_G à la borne de la grille. Si V_G est bas, alors il n'y a pas de connexion entre la source et le drain, et on dit que le transistor est éteint.

Un transistor pMOS a 4 connections; source, drain, grille (gate), et substrat. Généralement le substrat est connecté à la masse.

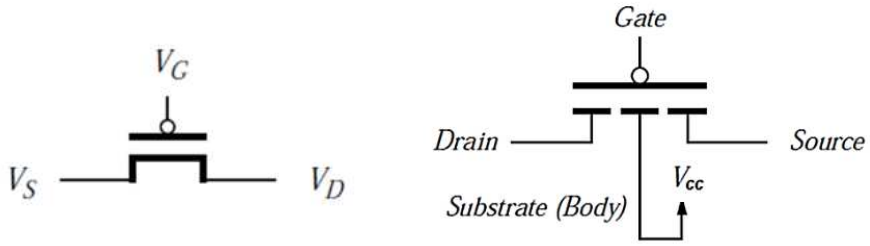


Fig.7. Symboles simplifiés d'un transistor de type pMOS.

En bref, le pMOS travail de façon opposée de nMOS. Si VG est haut, alors il n'y a pas de connexion entre la source et le drain, et on dit que le transistor est éteint.

3.1. Les portes logiques à base des transistors nMOS

1970 c'était l'année de la fabrication des portes logiques à base des transistors MOSFET soit par nMOS ou bien pMOS mais pas avec les deux en même temps. En 1980 c'était l'année de la combinaison des deux transistors (nMOS et pMOS) pour la fabrication des portes logiques. La première approche de la construction d'une porte logique en utilisant la technologie nMOS est représentée sur la figure suivante :

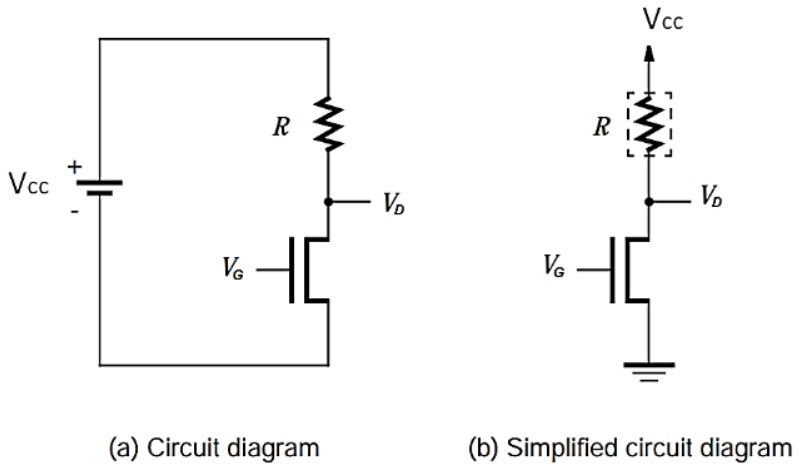


Fig.8. Première approche de la construction d'une porte logique avec nMOS.

- Si V_G = 1 logique (= Vcc = 3.3 v), le transistor nMOS est atteint, donc la tension V_D = Vcc. Cette tension est considérée 1 logique.
- Si V_G = 0 logique, le transistor nMOS est fermé, donc la tension V_D = tension de jonction (de l'ordre de 0.2 V). Cette tension est considérée 0 logique.

Alors le circuit est une implémentation nMOS d'une porte NOT.

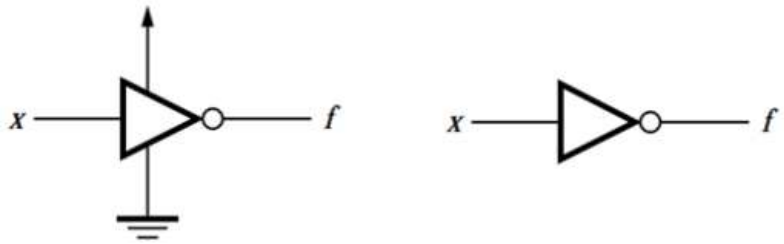
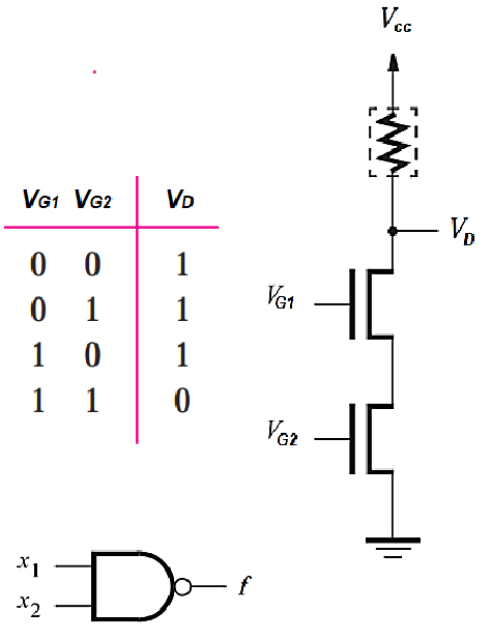


Fig.9. Symbole de porte logique NOT.

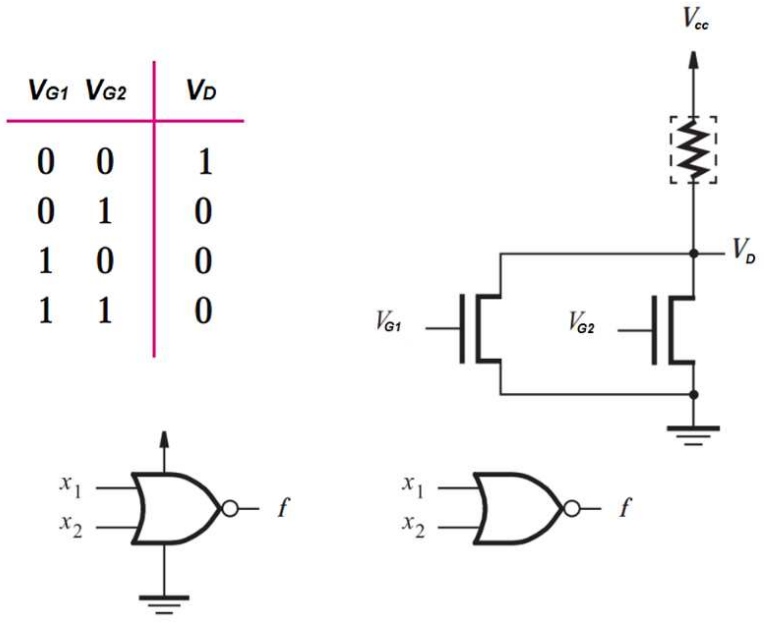
On peut enchainer deux transistors nMOS comme suite pour construire une porte logique NAND :



V_{G1}	V_{G2}	V_D
0	0	1
0	1	1
1	0	1
1	1	0

Fig.10. Construction d'une porte logique NAND à base des transistors nMOS.

La configuration suivante permet de réaliser une fonction logique NOR à base de la technologie nMOS :



V_{G1}	V_{G2}	V_D
0	0	1
0	1	0
1	0	0
1	1	0

Fig.11. Construction d'une porte logique NOR à base des transistors nMOS.

La configuration suivante permet de réaliser une fonction logique AND à base de la technologie nMOS :

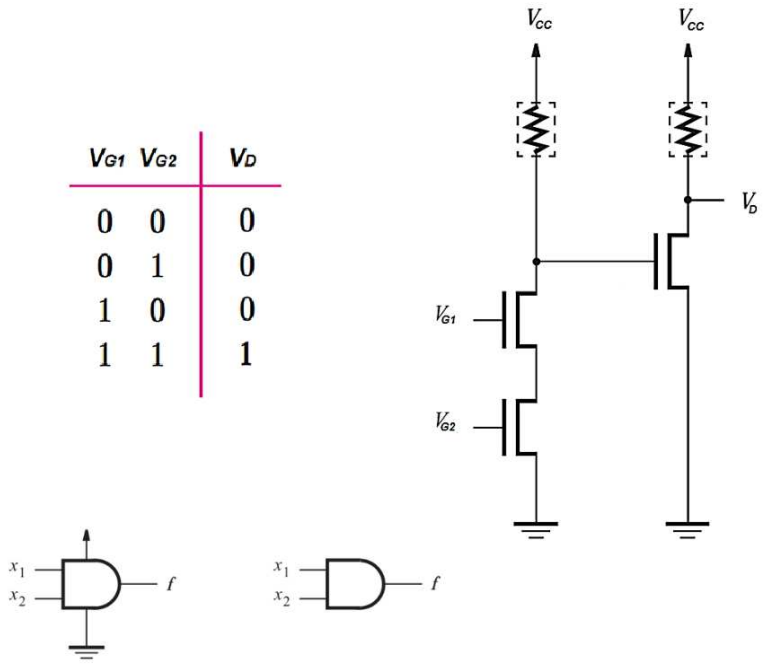


Fig.12. Construction d'une porte logique AND à base des transistors nMOS.

3.2. Portes logiques avec la technologie CMOS

La technologie CMOS (Complementary MOS), est la combinaison des transistors NMOS avec les PMOS pour réaliser les fonctions logiques. La technologie CMOS offre des avantages pratiques intéressants par rapport à la technologie NMOS:

- Consommation réduite d'énergie.
- Moins de bruit durant son fonctionnement

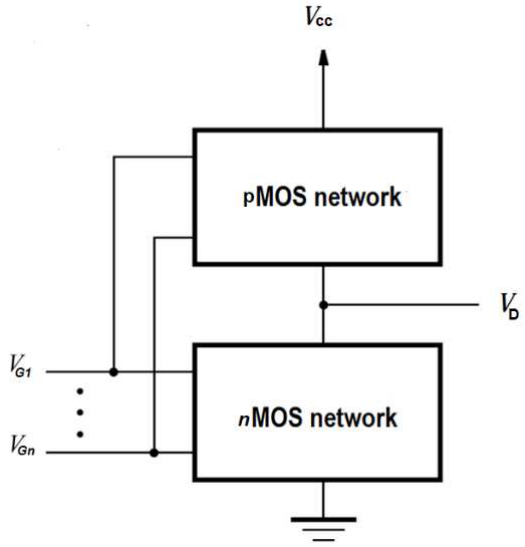
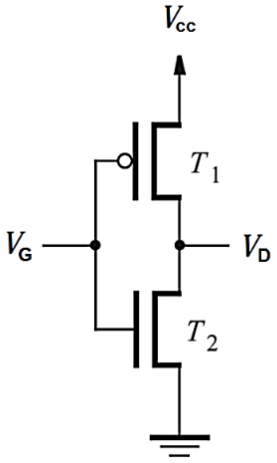


Fig.13. Portes logiques à base de la technologie CMOS.

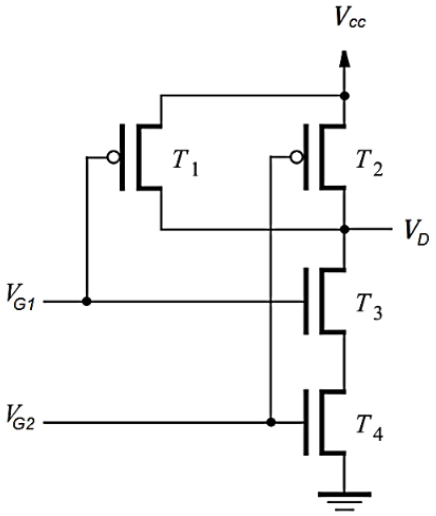
La figue suivante présente la constitution d'une porte logique NOT à base de la technologie CMOS:



V_G	T_1	T_2	V_D
0	on	off	1
1	off	on	0

Fig.14. Porte logique NOT à base de la technologie CMOS.

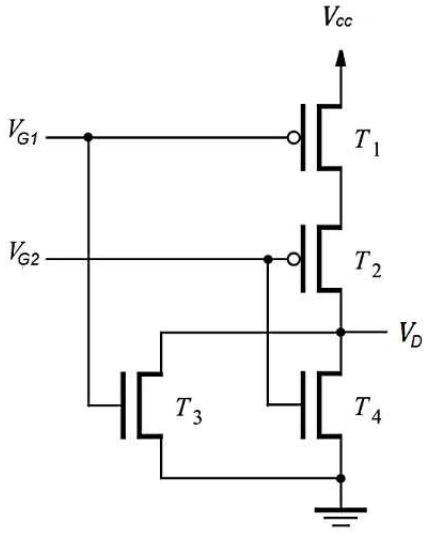
La figue suivante présente la constitution d'une porte logique NAND à base de la technologie CMOS:



V_{G1}	V_{G2}	T_1	T_2	T_3	T_4	V_D
0	0	on	on	off	off	1
0	1	on	off	off	on	1
1	0	off	on	on	off	1
1	1	off	off	on	on	0

Fig.15. Porte logique NAND à base de la technologie CMOS.

La figue suivante présente la constitution d'une porte logique NOR à base de la technologie CMOS:



V_{G1}	V_{G2}	T_1	T_2	T_3	T_4	V_D
0	0	on	on	off	off	1
0	1	on	off	off	on	0
1	0	off	on	on	off	0
1	1	off	off	on	on	0

Fig.16. Porte logique NOR à base de la technologie CMOS.

Chapitre 2 : Circuits logiques programmables

Chapitre 3 : Les FPGAs

1. Introduction

Vers le début des années 1980, il est devenu évident qu'il y avait un gap dans le continuum des circuits intégrés numériques. À une extrémité, il y a des dispositifs programmables comme SPLD et CPLD qui ont hautement configurables, avait une conception rapide et modifiable, mais qui ne pouvaient pas faire des fonctions importantes ou complexes. À l'autre extrémité nous avons les ASICs, ces dispositifs sont capables de réaliser des fonctions extrêmement importantes et complexes, mais ils étaient douloureusement cher et prend du temps à concevoir. En outre, une fois la conception est mis en œuvre, elle a effectivement gelée dans le silicium et ne serait plus modifiable.

Afin de combler la lacune entre les PLDs et les ASICs, Xilinx a développé une nouvelle classe de circuits intégrés appelé FPGA (Field Programmable Gate Array). Ross Freeman, le co-fondateur de la société Xilinx, a inventé le premier FPGA en 1985. Il à noter que les premiers FPGAs étaient basés sur la technologie CMOS et utilisent les cellules SRAM pour des raisons de configuration.

Si les FPGA rencontrent un tel succès dans tous les secteurs, c'est parce qu'ils combinent les meilleures caractéristiques des ASIC (Application-Specific Integrated Circuits) et des systèmes basés sur processeurs (logiciels). Les FPGAs disposent la même souplesse des logiciels, mais ils ne sont pas limités par le nombre de cœurs de traitement disponibles. Contrairement aux processeurs, les FPGAs sont vraiment parallèles par nature, de sorte que plusieurs opérations de traitement différentes ne se trouvent pas en concurrence lors de l'utilisation des ressources. Chaque tâche de traitement indépendante est affectée à une section spécifique du circuit, et peut donc s'exécuter en toute autonomie sans dépendre aucunement des autres blocs logiques. En conséquence, nous pouvons accroître le volume de traitement effectué sans que les performances d'une partie de l'application n'en soient affectées pour autant.

Les FPGA se placent à mi-chemin entre les PLD et les ASIC. Les avantages par rapport aux ASIC sont:

- Coût de développement plus bas: les dépenses en NRE (non-recurring engineering) sont moins importantes.
- Modifications plus simples à réaliser.
- Time-to-market plus court.

2. Architecture interne d'un FPGA

Chaque FPGA est constitué d'un nombre déterminé de ressources, Il se compose de trois parties principales:

1- Blocs logiques configurables (CLB) - qui implémentent les fonctions logiques.
2- Interconnexions programmables - qui implémentent le routage (interconnections entre les différents ressources).
3- Blocs d'entrées/sorties programmables - qui se connectent à des composants externes (pour que le circuit puisse accéder au monde extérieur).

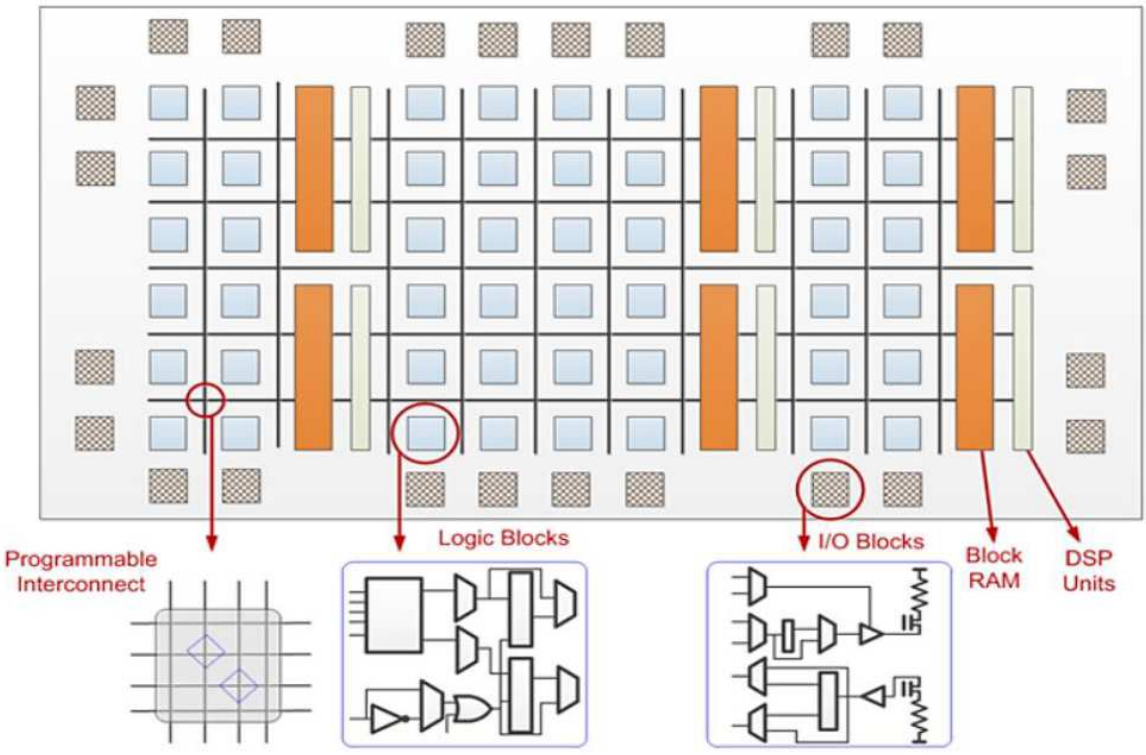


Fig.1. Architecture interne simplifié d'un FPGA.

2.1. Bloc logique configurable (CLB)

Les premiers FPGAs de Xilinx étaient basés sur le concept de de *Configurable Logic Block* (CLB), ce dernier contient une LUT de 3 entrées (Look Up table), une bascule, un multiplexeur et plusieurs autres éléments, mais ceux-ci sont typiquement les plus importants.

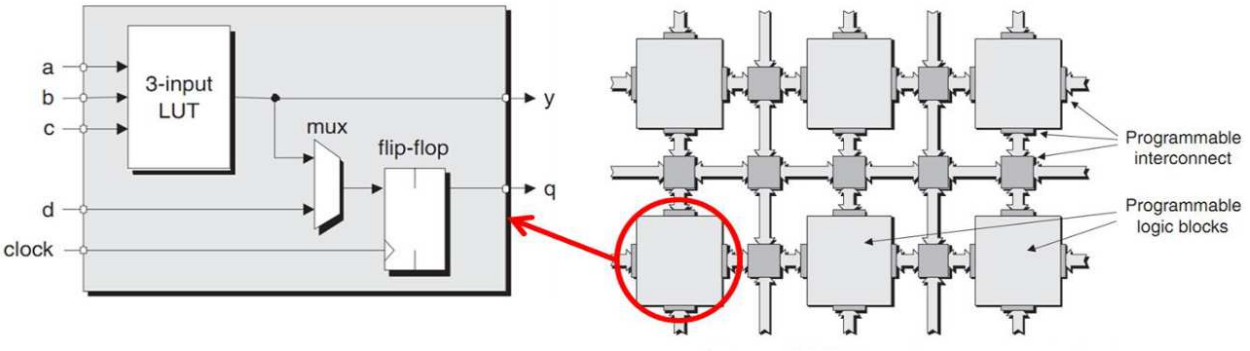


Fig.2. Vue générique d'un block CLB dans un FPGA.

Chaque FPGA contint un grand nombre des CLBs (iles) intégrés dans une mer d'interconnexion programmable. La fonction de la LUT est de stocker la table de vérité de la fonction combinatoire à implémenter dans les cellules mémoires.

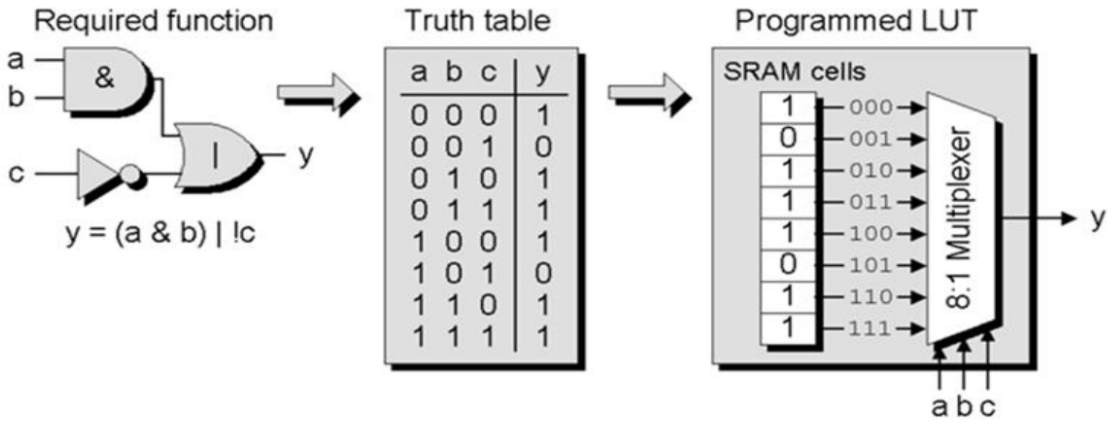


Fig.3. Rôle d'un LUT dans un FPGA.

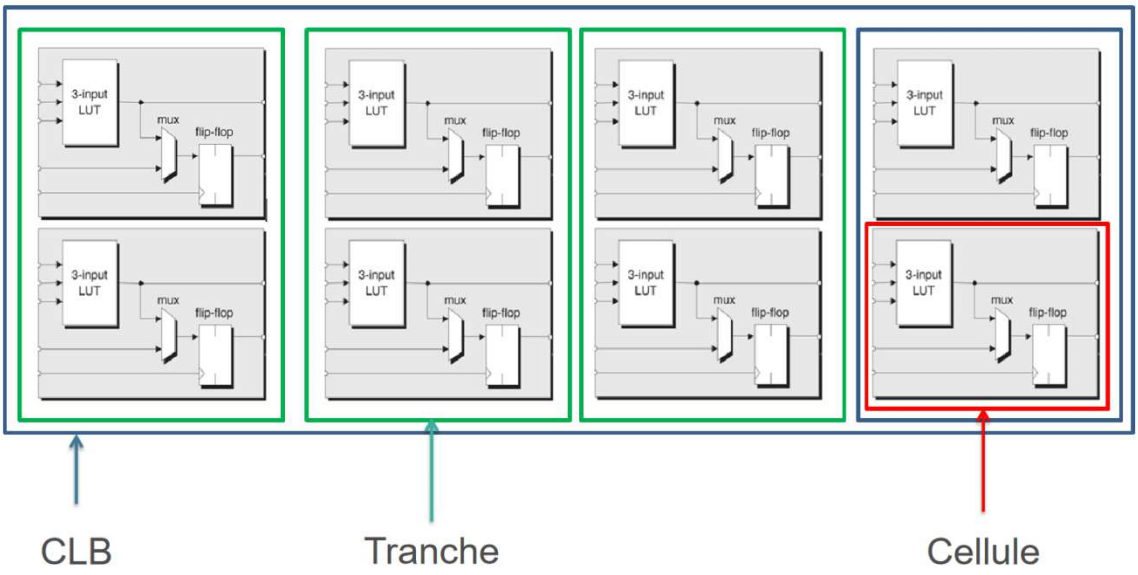


Fig.4. CLB d'un FPGA Spartan 3E.

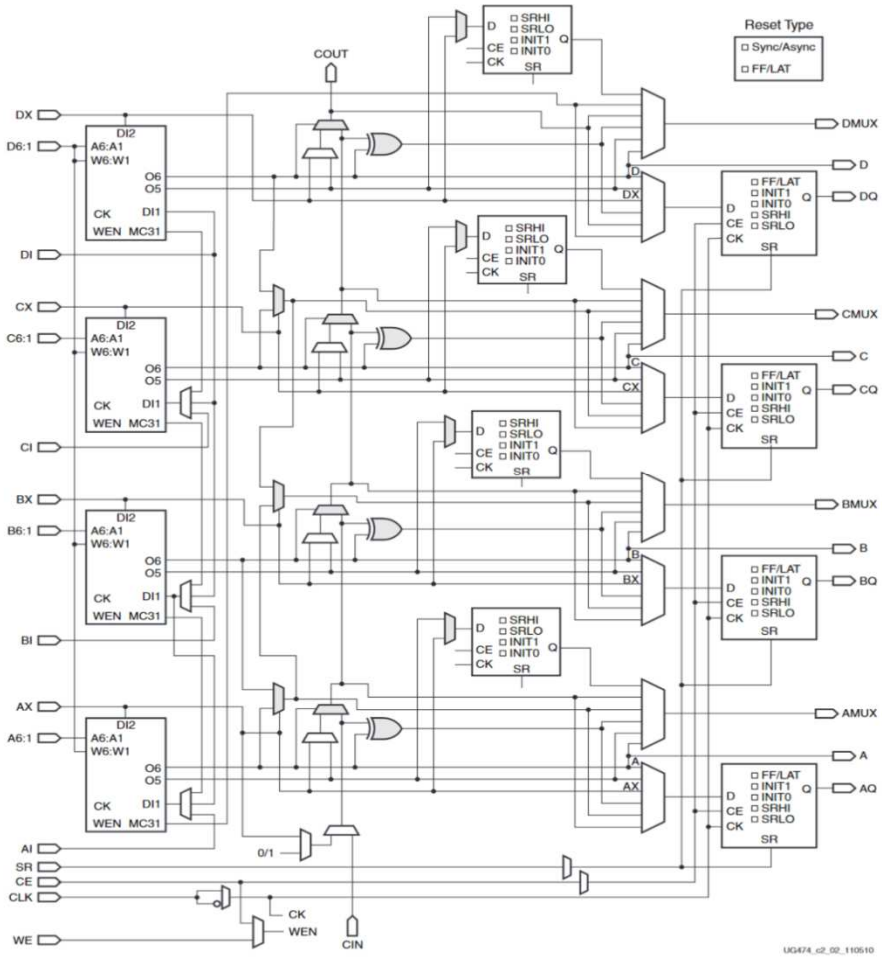


Fig.5. Tranche d'un FPGA ARTIX 7.

Une LUT à 4 entrées peut être utilisé comme DRAM (*Distributed* RAM) 16x1 bit. *Distributed* RAM car elle est distribuée sur toute la surface de la puce, ce qui la distinguer de la Block-RAM. Toutes les cellules configurables du FPGA, y compris ceux qui formant les LUTs sont effectivement attachés en ensemble pour former une chaîne longue.

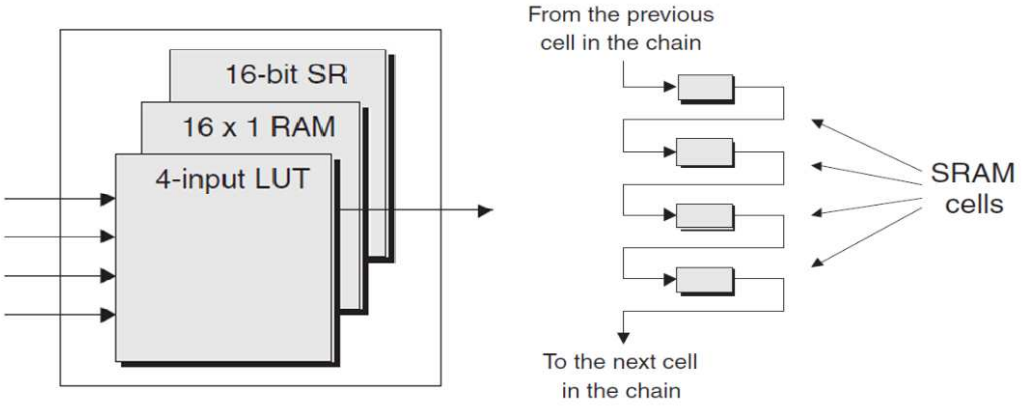


Fig.6. Utilisation des LUTs comme mémoires distribuées.

2.2. Réseau de routage programmable

En cascadant plusieurs CLB, de grandes conceptions peuvent être réalisées facilement. La connexion entre différents CLB et même les bus d'entrée-sortie (IOB) ou d'autres blocs personnalisés comme les RAM de bloc peut être établie à l'aide des *Programmable Switch Matrixes* (PSM) réparties sur la puce FPGA. Le PSM consiste en un réseau de points d'interconnexion programmables (*Programmable Interconnect Points* ou PIP), chaque PIP (voir Fig.7) est en fait une intersection d'une ligne de ligne avec une ligne de colonne, un PIP a six possibilités de connexions. Les FPGA modernes sont reconfigurables à base de SRAM ; ainsi, la connexion entre les lignes d'horizontales et de verticales dans un PIP est assurée par des transistors, chaque base de transistor est connectée à une cellule SRAM. Si la SRAM contient 1 logique, la connexion sera établie et vice versa.

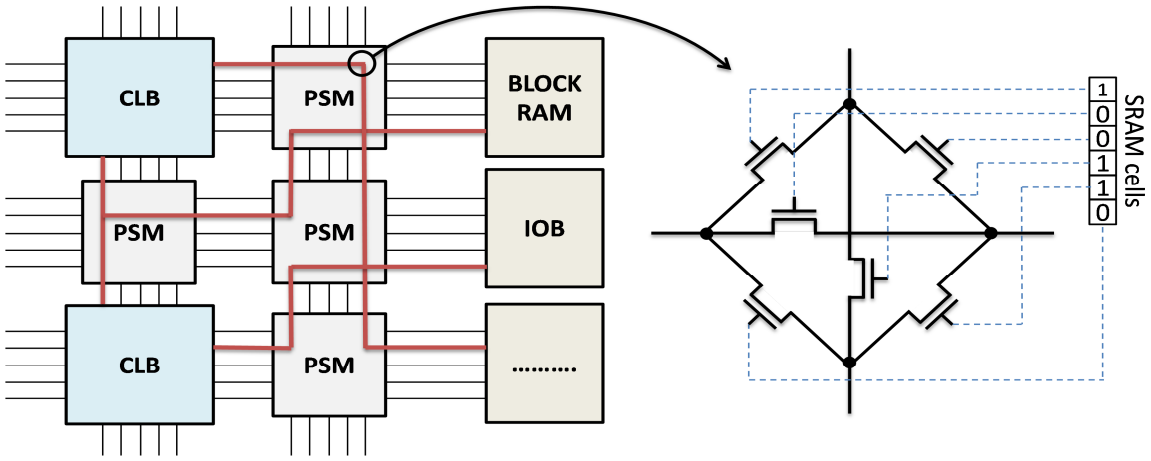


Fig.7. Programmable Switch Matrixes

Le fichier binaire de configuration ou *bitstream* permet de configurer toutes les cellules SRAM programmables que ce soit sur les LUTs (pour les fonctions logiques) ou le PSM pour la connexion.

2.3. Entrées/sorties programmables (General I/O)

Les boitiers des FPGA d'aujourd'hui peuvent avoir plus de 1000 broches, qui sont disposées en réseau sur la base du boitier. De même, en ce qui concerne la puce de silicium à l'intérieur du boîtier, les stratégies d'emballage flip-chip permettent de présenter les broches d'alimentation, de masse, d'horloge et d'E/S sur toute la surface de la puce.

Les blocs des E/S contrôlent le flux de données entre les broches d'E/S et la logique interne de l'appareil. Chaque E/S prend en charge le flux de données bidirectionnel et le fonctionnement à 3 états. Il prend en charge une variété de normes de signal, y compris quatre normes différentielles hautes performances. Des registres à double débit de données (DDR) sont inclus.

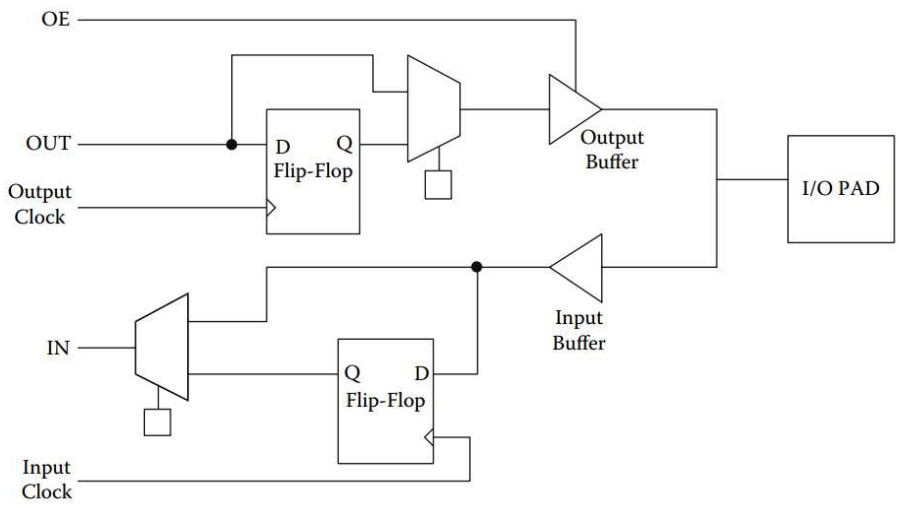


Fig.8. Un bloc d'entrée/sortie simplifié.

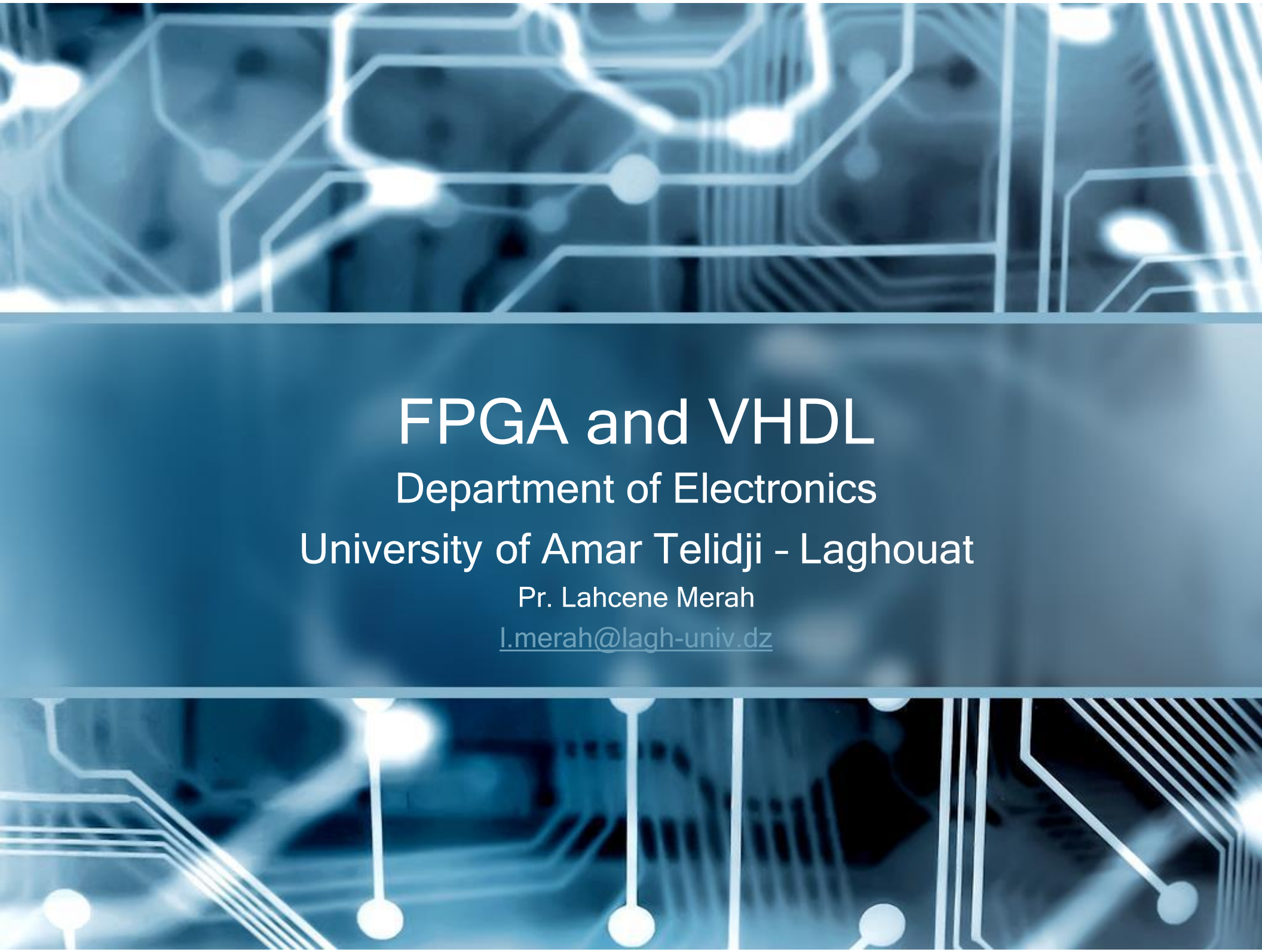
2.4. Autres ressources FPGA

Nous passons brièvement en revue quelques autres composants internes d'un FPGA :

- Block RAM : En plus de la RAM distribuée, les FPGA modernes ont également un bloc RAM (BRAM) intégré qui a des tailles différentes selon la famille de FPGA. La BRAM stocke de manière synchrone de grandes quantités de données par rapport à la RAM distribuée.
- Processeurs de signaux numériques (DSP) : ces blocs ont pour rôle d'effectuer des opérations DSP au lieu d'utiliser des CLB pour créer de tels opérateurs ; il est évident que de nombreuses opérations arithmétiques sont nécessaires dans toute application DSP telle que les multiplicateurs. Pour cette raison, les constructeurs de FPGA ont leurs blocs DSP personnalisés intégrés sur les FPGA.
- Gestionnaires d'horloge numérique (*Digital Clock Managers* ou DCM) : il s'agit d'une fonction câblée (bloc) qui génère un certain nombre d'horloges filles à partir de l'horloge système d'origine. Un DCM a de nombreuses fonctions différentes telles que la suppression de la gigue, la synthèse de fréquence, le déphasage et la correction automatique du biais.

- Émetteurs-récepteurs d'E/S haute vitesse : les FPGA haut de gamme d'aujourd'hui incluent des blocs d'émetteurs-récepteurs Gigabit câblés spéciaux. Ces blocs utilisent une paire de signaux différentiels (c'est-à-dire une paire de signaux qui portent toujours des valeurs logiques opposées) pour transmettre (TX) des données et une autre paire pour recevoir (RX) des données. À titre d'exemple, le débit de données réalisable des émetteurs-récepteurs Xilinx Ultra-Scale varie de 500 Mb/s à 32,75 Gb/s.
- *Hard IP cores* (Propriété Intellectuelle) : Un cœur IP se présente sous la forme de blocs pré-implémentés tels que des cœurs de microprocesseur, des interfaces gigabit, des multiplicateurs, des additionneurs, des fonctions MAC, etc. Ces blocs sont conçus pour être aussi efficaces que possible en termes de consommation d'énergie, d'espace en silicium et de performances. Chaque famille de FPGA comportera différentes combinaisons de ces blocs. Avec diverses quantités de blocs logiques programmables.

Chapitre 4 : Le VHDL



FPGA and VHDL

Department of Electronics

University of Amar Telidji - Laghouat

Pr. Lahcene Merah

l.merah@lagh-univ.dz

Programmation en VHDL

Programmation des circuits

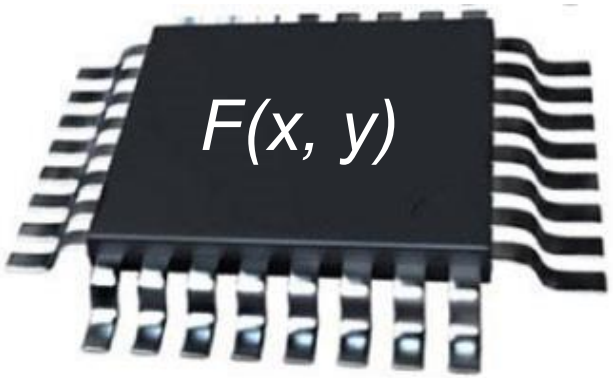
Programmer un circuit pour faire quelque chose!

Programmé
(préfabriqué)
à l'usine

Programmé par
l'utilisateur, FPGA
CPLD, DSP

*La programmation
en utilisant certains langages de
programmation*

x, y



z

FPGA (Field Programmable Gate Array) : est un circuit intégré conçu pour être configuré par un concepteur après fabrication.

Configuration des FPGAs

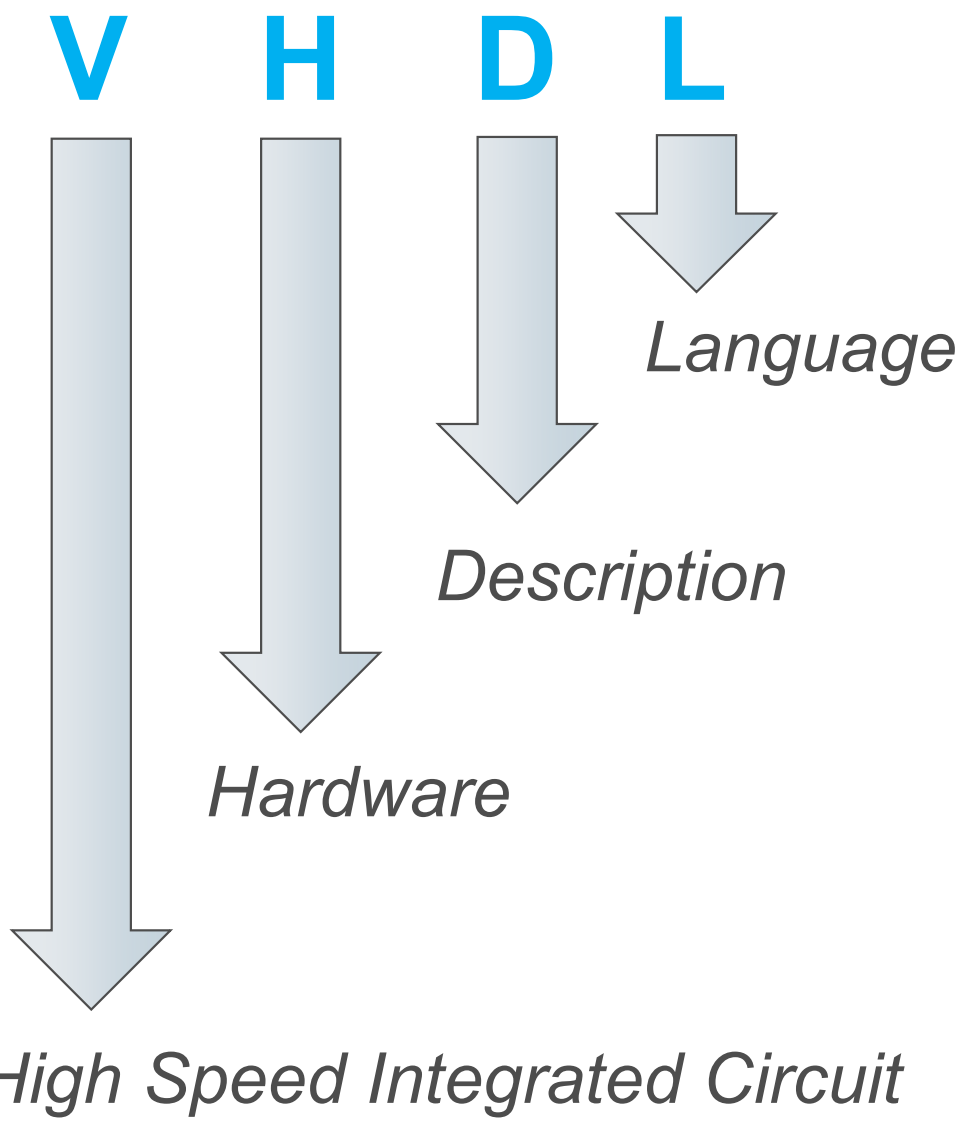
Hardware Description Languages

VHDL

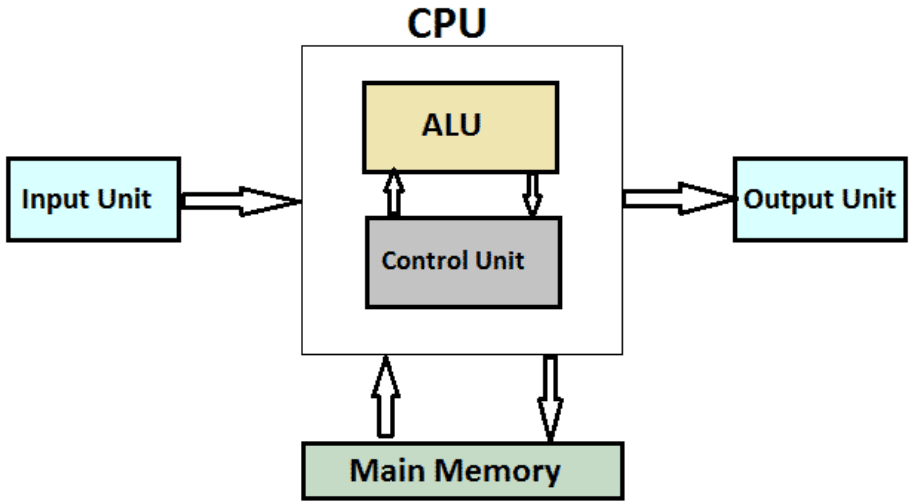
Verilog

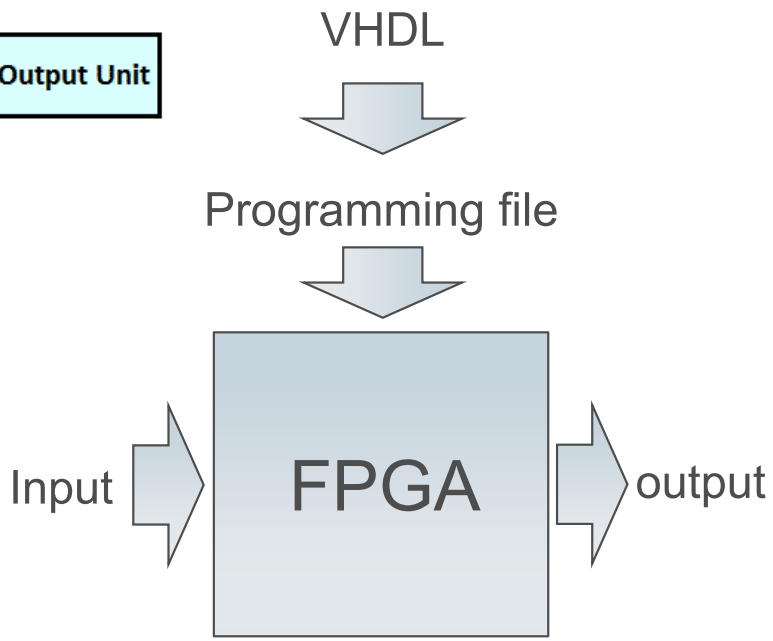
VHDL


```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity B_BOX is
    Port ( A, B, C : in  STD_LOGIC;
           F : out  STD_LOGIC);
end B_BOX;
```


V H D L
Language
Description
Hardware
Very High Speed Integrated Circuit

- Normalisé en 1987 sous IEEE 1076-87.
- Similaire aux langages de programmation mais pas exactement les mêmes.
- Utilisé pour décrire ou exprimer le comportement des circuits logiques numériques (alors que les langages de programmation pour créer des logiciels).





- VHDL permet de vérifier et de modéliser le comportement du système avant la traduction des outils de synthèse de la conception en portes et fils réels (matériel).

- VHDL est un langage de flux de données, ce qui signifie qu'il peut examiner simultanément chaque instruction pour exécution.

- Permet de passer d'un très haut niveau d'abstraction, à un niveau proche du matériel (ensembles de portes logiques et d'interconnexion (gate level)).

RTL (Register Transfer Level)

Fonctionnalité de la conception

au-lieu

Sa mise en œuvre

laissant décrire ce que fait le design

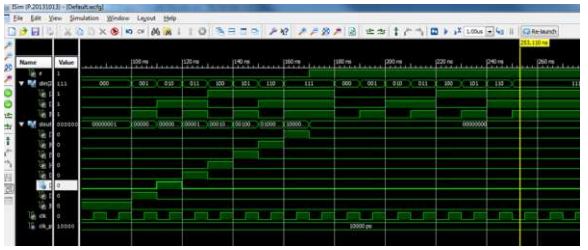
Le rôle de l'outil de synthèse est de mettre en œuvre la conception

Test Bench

Le code pour créer des vecteurs d'entrée de simulation et pour tester le comportement des vecteurs de sortie de simulation (vérification de fonctionnalité)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity B_BOX is
    Port ( A, B, C : in  STD_LOGIC;
           F : out  STD_LOGIC);
end B_BOX;
```

Testbench



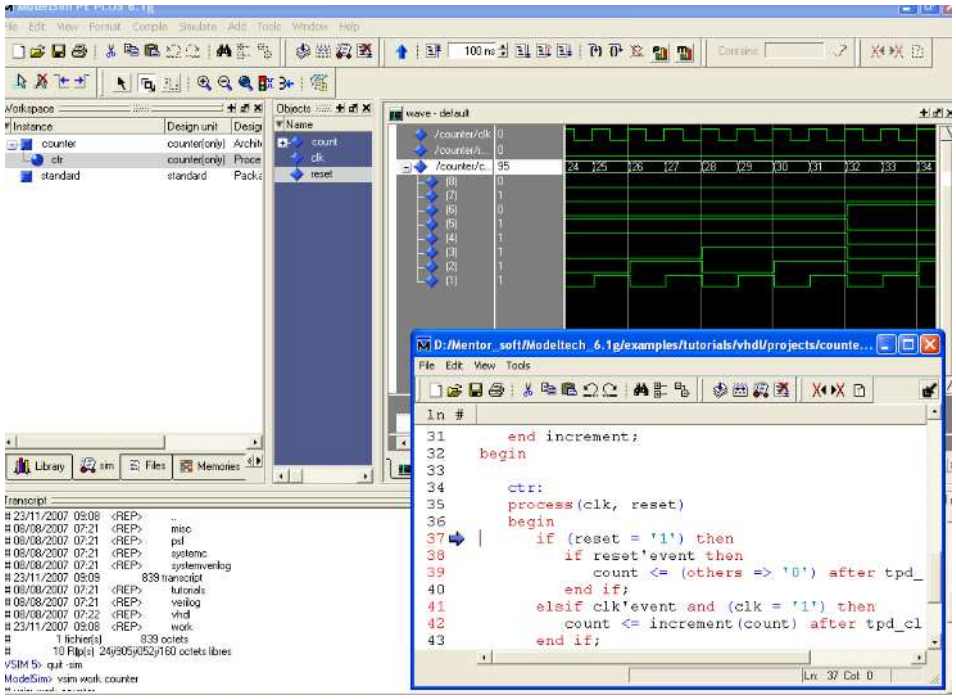
Flot de conception

1- Synthèse Logique,
   (HDL, Schématique)
2- Simulation fonctionnelle,
3- L'optimisation (Mappage, package,
   placement et routage),
4- Simulation temporelle,
5- La génération de fichier de
   programmation (*bitstream*).

Simulation fonctionnelle

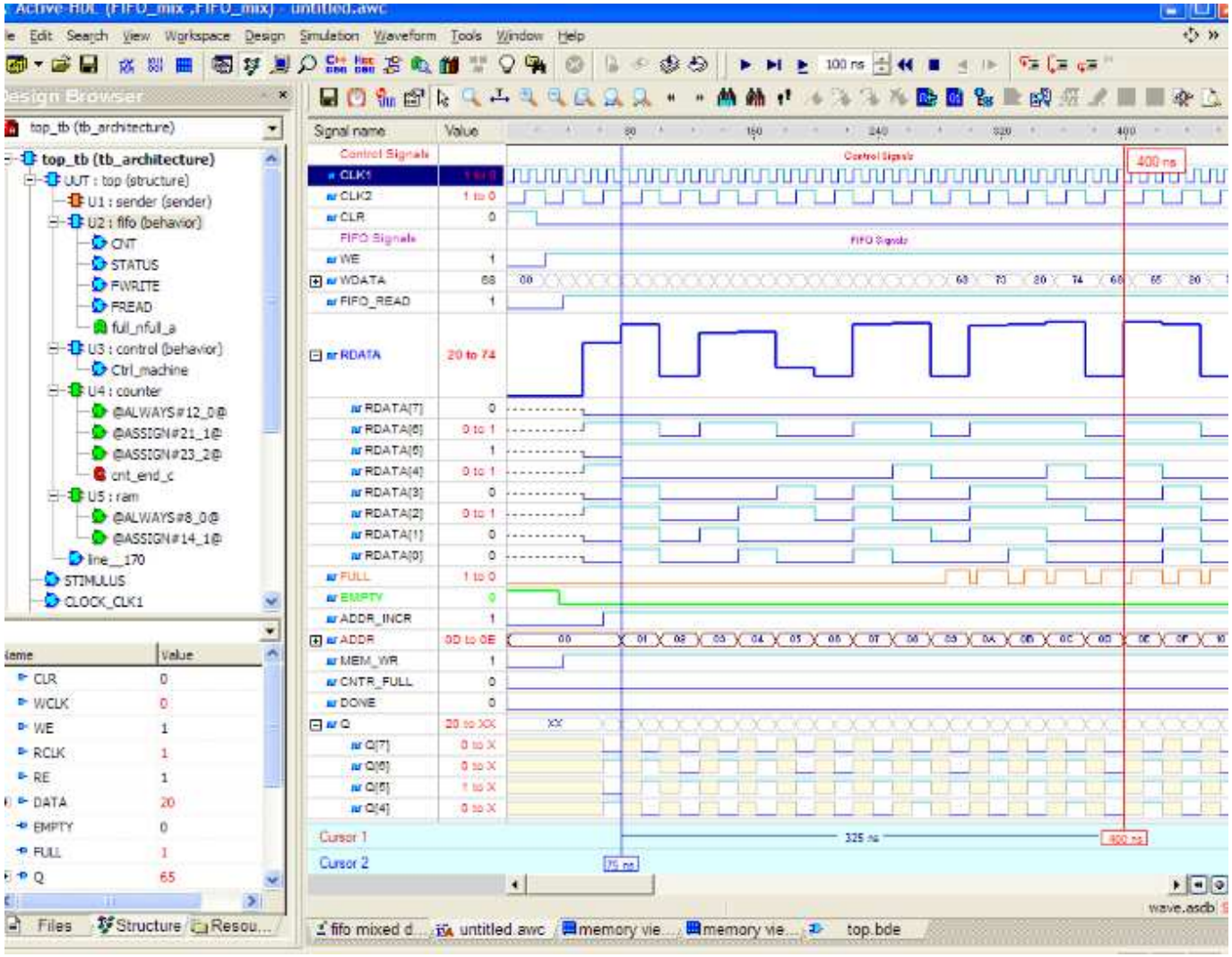
La simulation fonctionnelle peut être réalisée par l'intermédiaire d'un outil spécifique à cette tâche comme :

ModelSim
Advanced Verification and Debugging



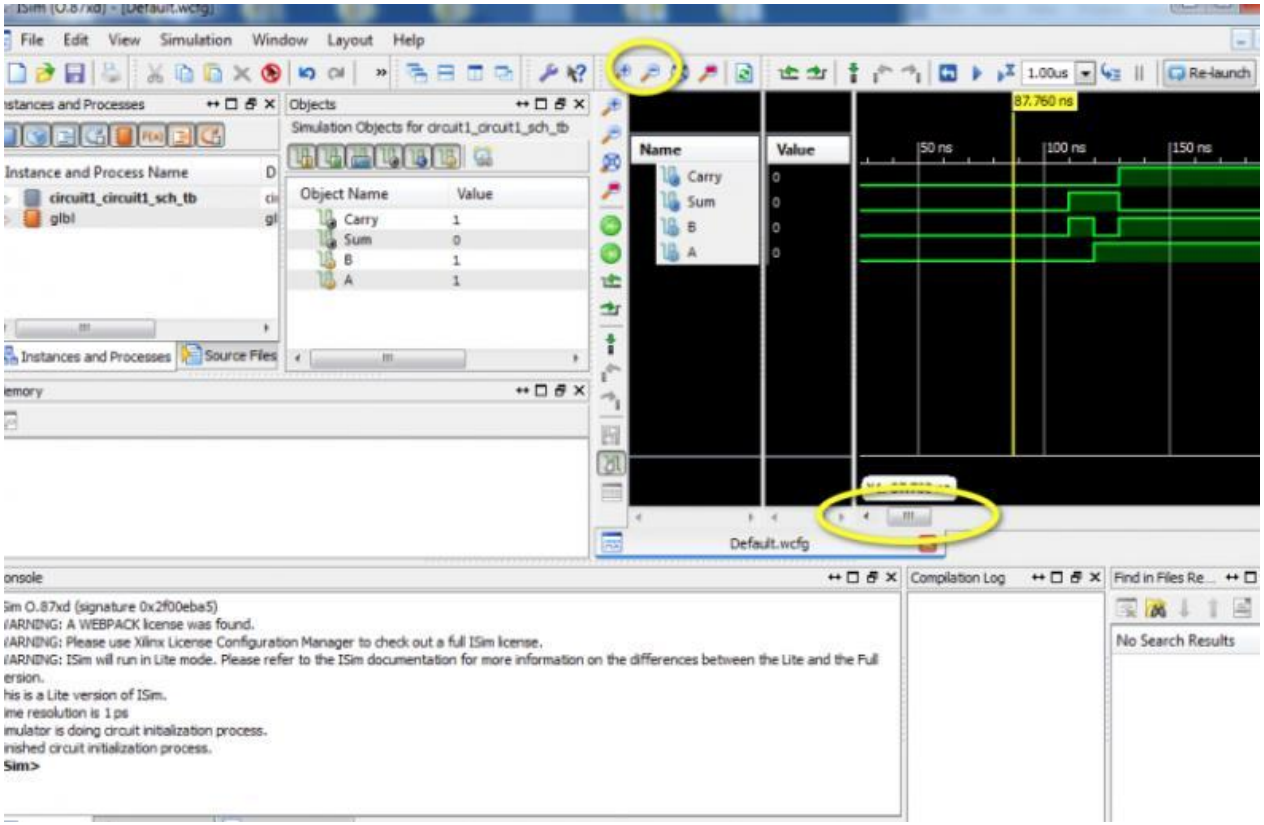
ALDEC

THE DESIGN VERIFICATION COMPANY



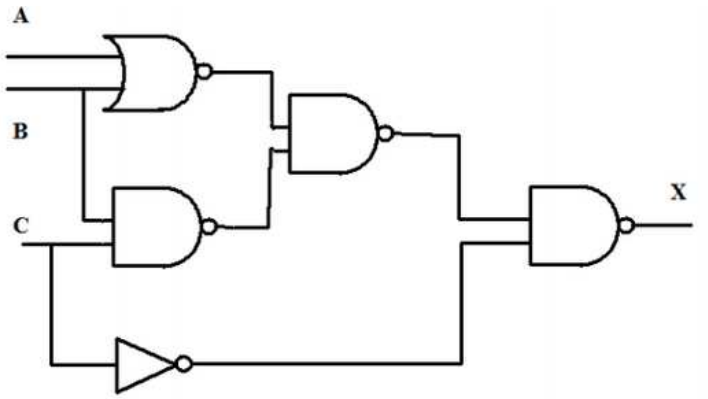
XILINX®


Vivado 2016.3


File Edit View Simulation Window Layout Help
Re-launch
Instances and Processes
Objects
Simulation Objects for circuit1_circuit1_sch_tb
Instance and Process Name
circuit1_circuit1_sch_tb
glbl
Object Name
Value
Carry 1
Sum 0
B 1
A 1
Instances and Processes
Source Files
Name
Value
Carry
Sum
B
A
87.760 ns
50 ns
100 ns
150 ns
Default.wcfg
Console
Compilation Log
Find in Files Re...
No Search Results
Sim O.87xd (signature 0x2f00eba5)
WARNING: A WEBPACK license was found.
WARNING: Please use Xilinx License Configuration Manager to check out a full ISim license.
WARNING: ISim will run in Lite mode. Please refer to the ISim documentation for more information on the differences between the Lite and the Full
ersion.
his is a Lite version of ISim.
ime resolution is 1 ps
mulator is doing circuit initialization process.
inished circuit initialization process.
Sim>

Conception avec le VHDL

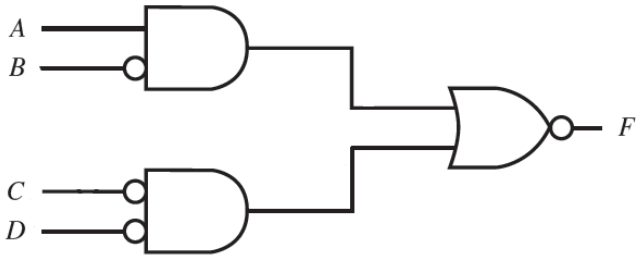
Circuits très simples



Systèmes très compliqués

Conception avec le VHDL

Exemple :



1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

```
-- Mettre vos comentaires ici
```

2

```
entity simple_circuit is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           F : out  STD_LOGIC);
end simple_circuit;
```

3

```
architecture Behavioral of simple_circuit is

begin

F <= (A and not B) nor (not C and not D);

end Behavioral;
```

Déclaration de la bibliothèque

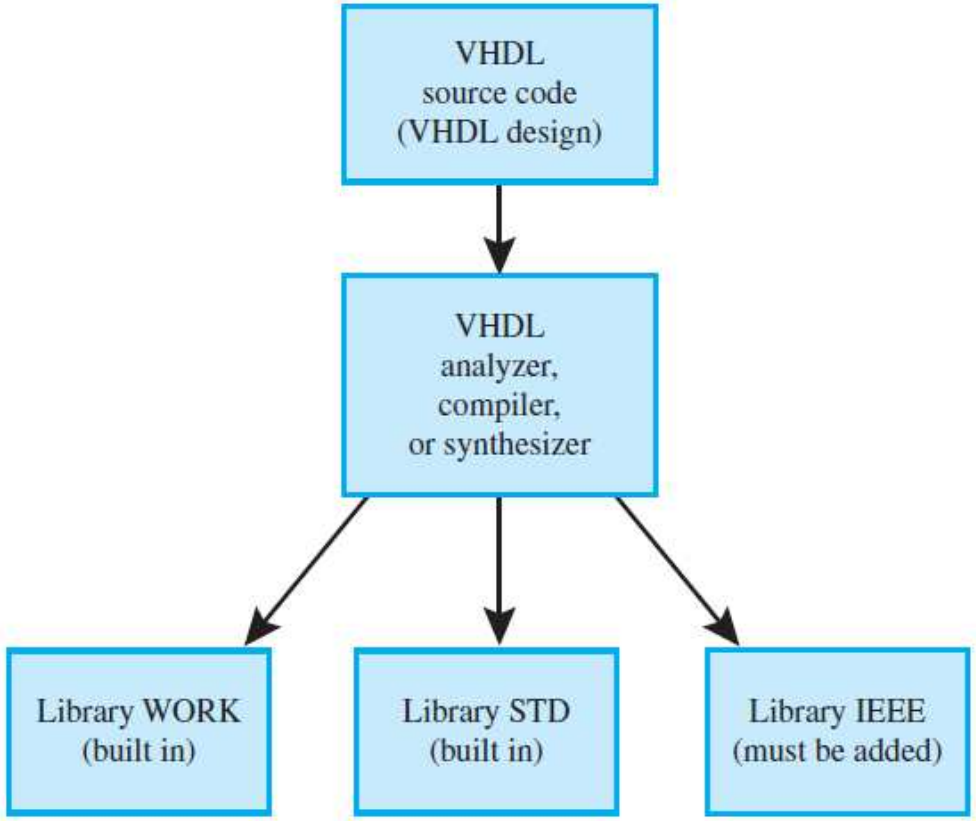
- Pour réaliser un code VHDL complet, nous avons besoin de placer la partie bibliothèque qui indiquée par les mots ***Library*** et ***use*** dans chaque entité de conception.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

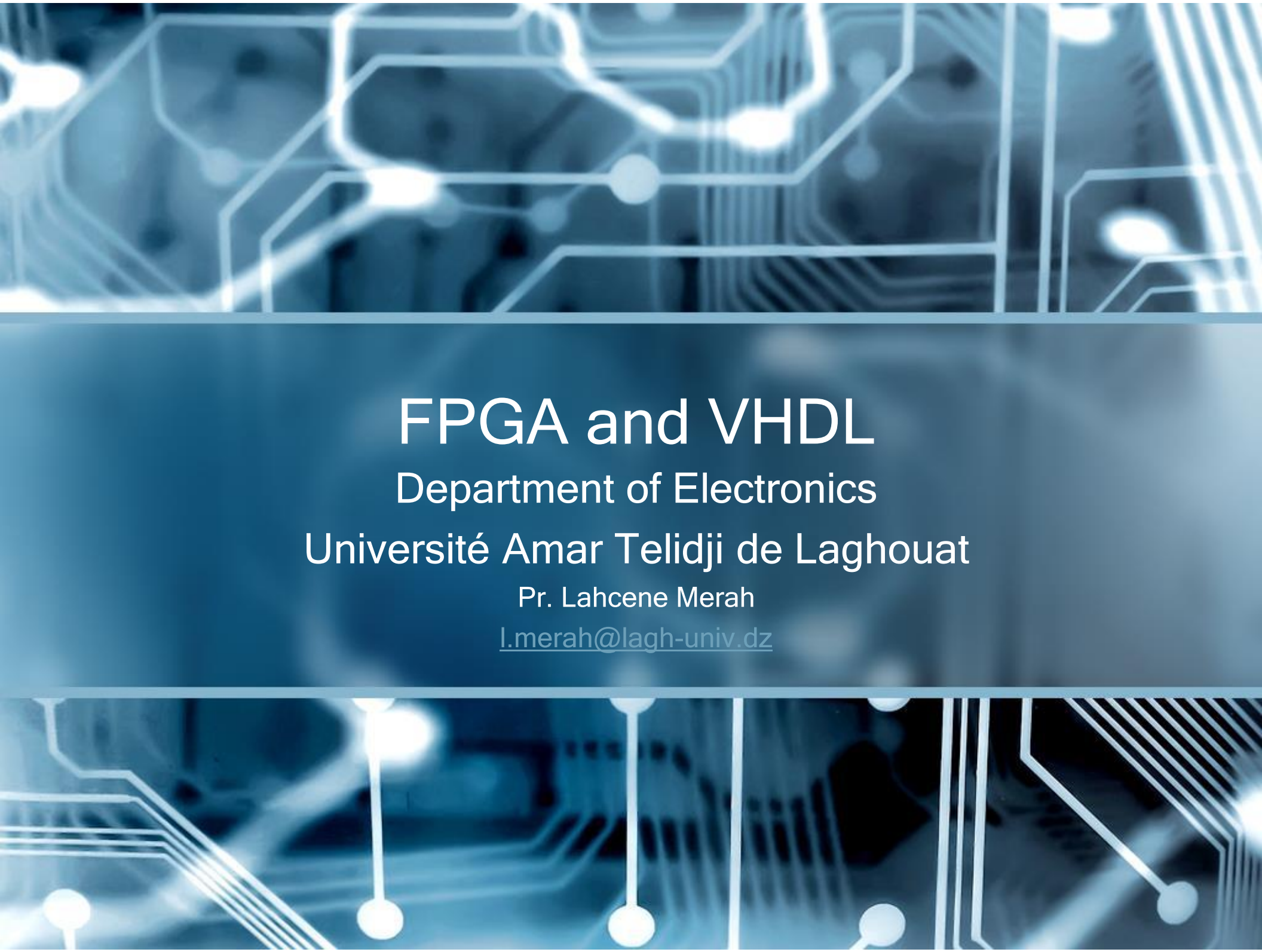
- La bibliothèque est un lieu de stockage qui contient des paquetages qui fournissent des informations pour la conception.
- La bibliothèque est aussi le lieu de stockage du code VHDL compilé.
- L'écriture indique le répertoire ou se trouve le package de la bibliothèque, exemple de Xilinx :

  C:\Xilinx\Vivado\2016.3\ids_lite\ISE\vhdl\src\ieee\ieee.std_logic_1164.vhd

Déclaration de la bibliothèque



Les bibliothèques WORK et STD sont implicitement déclaré. mais la bibliothèque IEEE doit être ajouté à une conception VHDL via une clause ***library*** pour le rendre visible à la conception.



FPGA and VHDL

Department of Electronics

Université Amar Telidji de Laghouat

Pr. Lahcene Merah

l.merah@lagh-univ.dz

Le VHDL

Entité, architecture, opérateurs,

styles de conceptions

Syntaxe

Prenons l'exemple du circuit logique suivant :

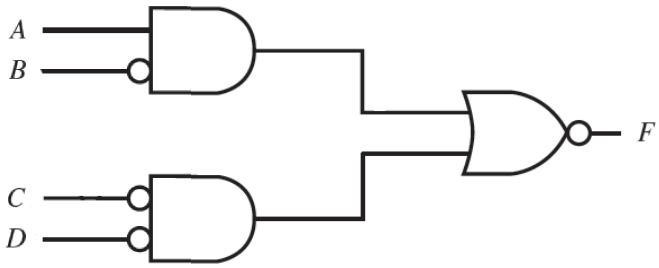
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
```

1

```
-- Mettre vos comentaires ici
```

```
entity simple_circuit is
    Port ( A : in  STD_LOGIC;
           B : in  STD_LOGIC;
           C : in  STD_LOGIC;
           D : in  STD_LOGIC;
           F : out  STD_LOGIC);
end simple_circuit;
```

2



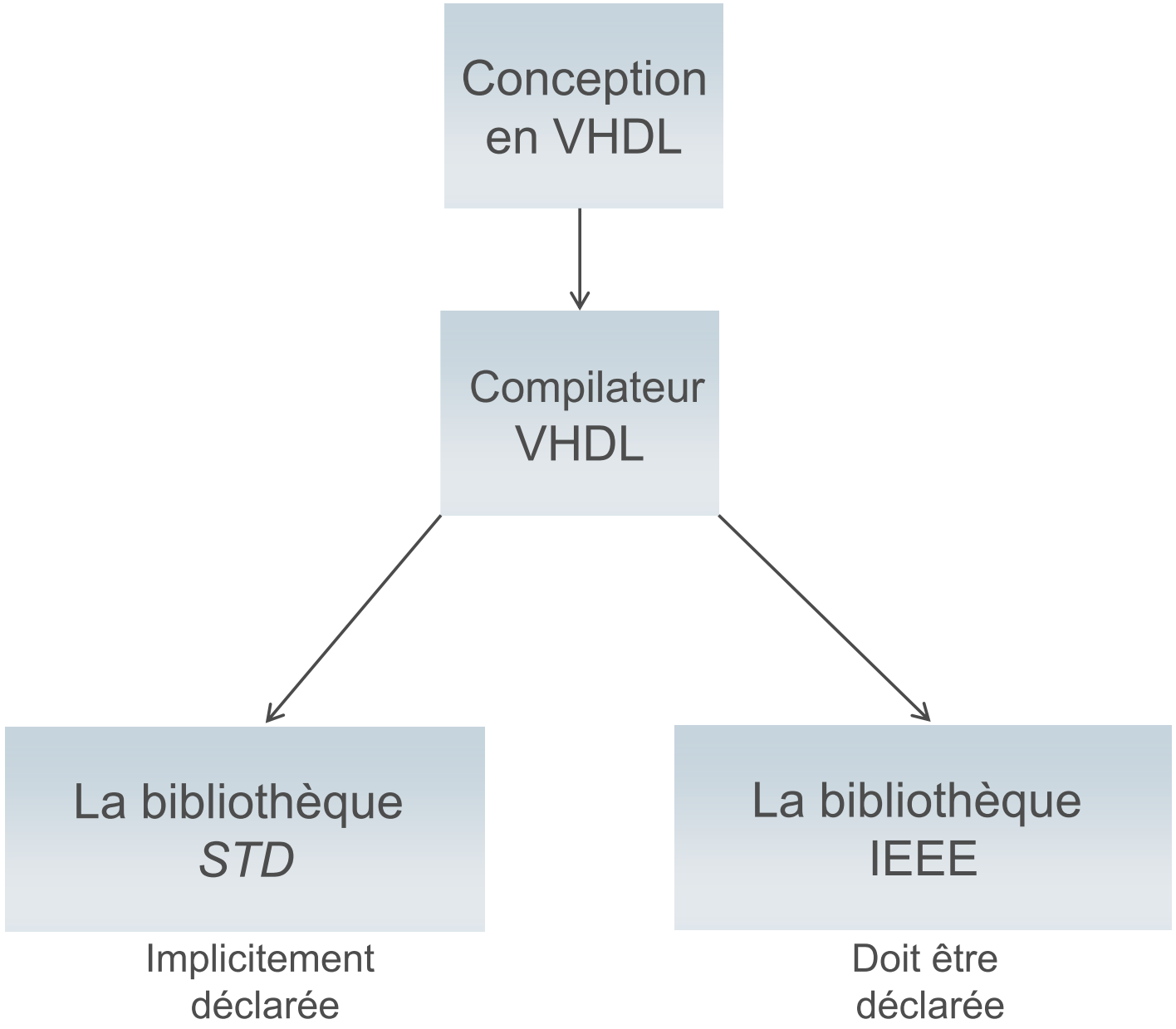

```
architecture Behavioral of simple_circuit is

begin

F <= (A and not B) nor (not C and not D);

end Behavioral;
```

3

La bibliothèque

- **Pour réaliser un code VHDL complet, nous avons besoin de placer la partie bibliothèque qui indiquée par les mots *Library* et *use* dans chaque entité de conception afin que nous puissions utiliser une bibliothèque logique standard avec ses définitions de types de données, fonctions et procédures.**
- **La bibliothèque est un lieu de stockage qui contient des paquetages qui fournissent des informations pour la conception.**

La bibliothèque



La bibliothèque standard (STD)

Cette bibliothèque est implicitement déclarée, elle définir les données de type **BIT** ou **BIT_VECTOR**.

Si on travail avec ce type de données, on ait pas besoin de déclarer ça bibliothèque.

```
entity my_circuit_name is
  port( inp1: in  bit;
        inp2: in  bit;
        outp1: out  bit;
        outp2: out  bit;
        outp3: out  bit);
end my_circuit_name;
```

La bibliothèque *STD* est le lieu de stockage pour les opérateurs logique divers tels que NOT, AND, OR, NAND, XOR…etc. La bibliothèque *STD* peut aussi contenir les opérations rationnelles talques : >, < , =, la figure suivante présente la liste des instructions et opérateurs supportés :

A	**abs, all, alias, and, architecture, array, attribute**
B	**begin, block, body, buffer**
C	**case, component, configuration, constant**
D	**downto**
E	**else, elsif, end, entity, exit**
F	**for, function**
G	**generate, generic, group**
I	**if, in, inout, is**
L	**library, literal, loop**
M	**map, mod**
N	**nand, next, nor, not, null**
O	**of, or, others, out**
P	**package, port, procedure, process**
R	**range, record, rem, return, rol, ror**
S	**select, signal, sla, sll, sra, srl, subtype**
T	**then, to, type**
U	**until, use**
V	**variable**
W	**wait, when, while, with**
X	**xnor, xor**

La bibliothèque IEEE

- Le groupe de travail IEEE responsable du Manuel de référence du langage VHDL avait proposé des packages standardisés.
- L'écriture indique le répertoire ou se trouve le package de la bibliothèque, exemple de Xilinx :

C:\Xilinx\Vivado\2016.3\ids_lite\ISE\vhdl\src\ieee

- C'est le package le plus utilisé, il contient les sous-bibliothèques :

```
library IEEE;
        use IEEE.std_logic_1164.all;
        use IEEE.std_logic_textio.all;
        use IEEE.std_logic_arith.all;
        use use IEEE.numeric_std.all;
        use IEEE.std_logic_signed.all;
        use IEEE.std_logic_unsigned.all;
        use IEEE.math_real.all;
        use IEEE.math_complex.all;
```

Le package STD vs IEEE

La bibliothèque STD

BIT/BIT_VECTOR

↓

0,1

Nous n'avons pas besoin d'inclure un package supplémentaire en tête du programme VHDL

STD_LOGIC (IEEE - 1164)

STD_ULOGIC/
STD_LOGIC

↓

9 digital states

Le package IEEE doit être inclut.
(ieee.std_logic_1164)

STD_LOGIC/ STD_LOGIC_VECTOR

Des types importants prédéfinis dans le package std_logic_1164 sont les types std_ulogic (Unresolved) et std_logic (Resolved). Ces deux types peuvent représenter en commun les valeurs suivants:

'U'	Uninitialized
'X'	Forcing unknown
'0'	Forcing 0
'1'	Forcing 1
'Z'	High impedance
'W'	Weak unknown
'L'	Weak 0
'H'	Weak 1
'–'	Don't care

La différence entre eux est la suivante: std_ulogic est un type non résolu. C'est une erreur pour un signal std_ulogic d'être entraîné par deux sources de signal (comme le court-circuit de 2 fils ensemble.

L’état U : Uninitialized (Non initialisé)

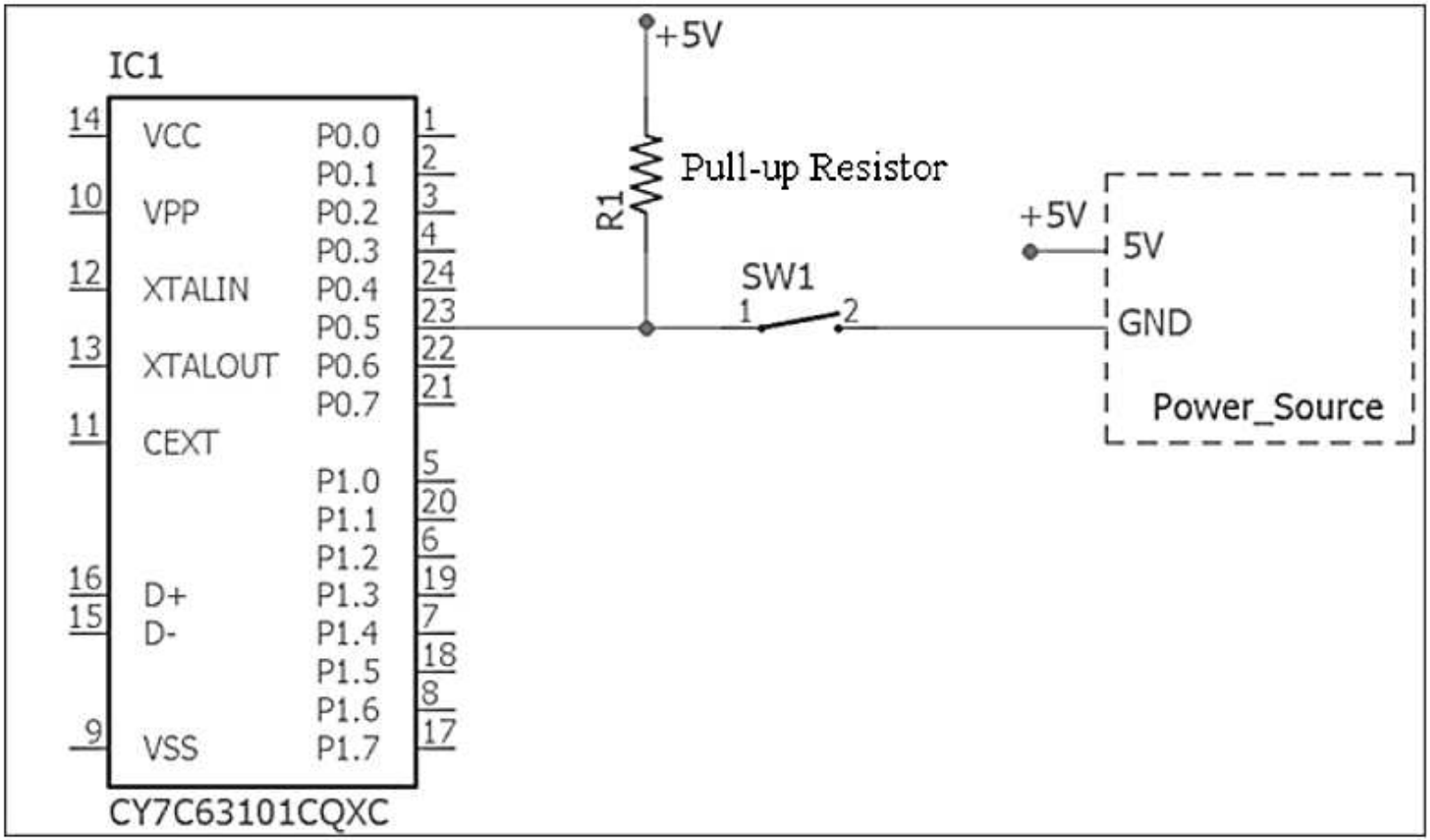
- La valeur **U** représente un signal non initialisé.
- La valeur non initialisée est importante lors du démarrage du simulateur et est la valeur initiale de tous les signaux. En observant quels signaux restent non initialisés, le concepteur peut identifier les problèmes au démarrage du circuit.

Les états logiques ‘1’, ‘H’, ‘0’ , ’L’

Les états logiques ‘1’, ‘H’, ‘0’ , ’L’ de type STD_ULOGIC sont interprétées comme représentant l'un des deux niveaux logiques, où chaque niveau logique représente l'une des deux plages de tension distinctes dans le circuit à synthétiser.

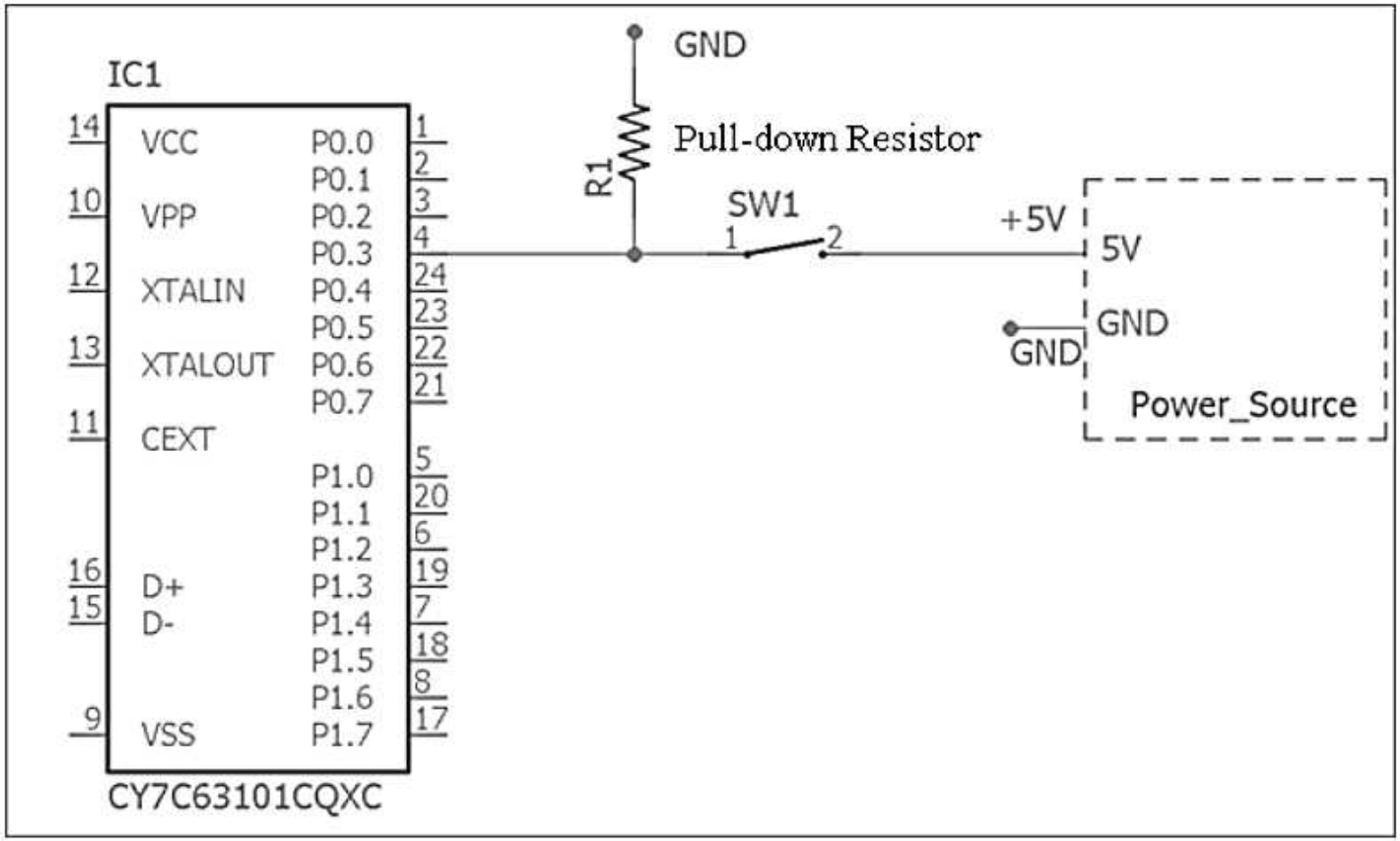
Les états logiques ‘H’, ’L’

H : signifie que le signal est logiquement haut, mais qu'il y a une chute de tension (issue d’une résistance de tirage (pull-up)).



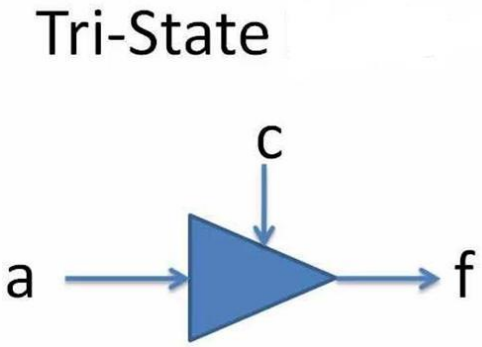
Les états logiques ‘H’, ’L’

L : Un «0» logique fort représente une masse parfaite, tandis qu'un «0 faible» représente un signal qui est logiquement «0», mais pas exactement la tension de masse (signal issu d’une résistance de rappel (pull-down)).

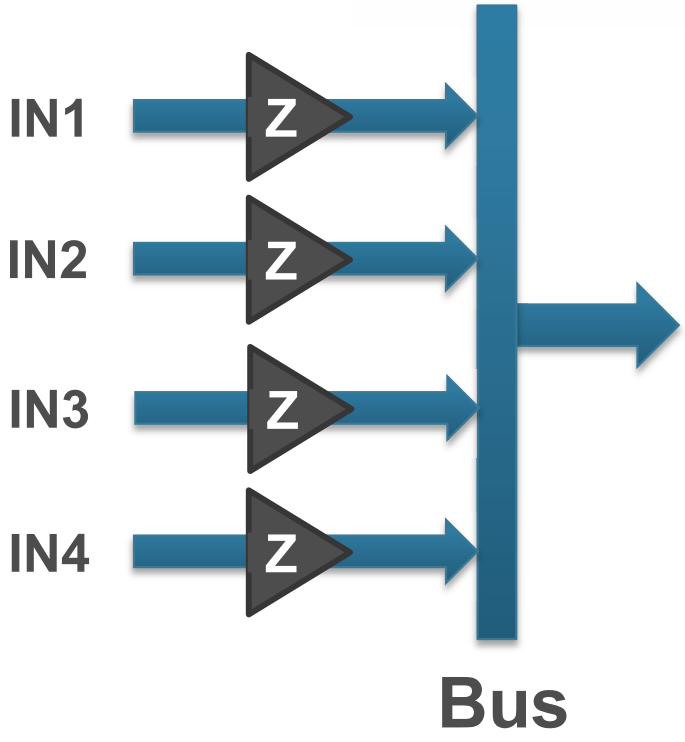


La haute impédance (Hi-Z, tri-state (3éme état)):

L'objet de la sortie à trois états (Hi-Z) est de supprimer l'influence d'un circuit sur ceux qui lui sont reliés. Si plus d'un appareil est connecté électriquement, mettre une sortie dans l'état Hi-Z évite les conflits.



c	a	f
0	0	Z
0	1	Z
1	0	0
1	1	1



OUT (l'un des entrées a chaque instant t, a cette instant les restes des entrées doivent êtres libérés).

L’état : don’t care state (-)

L'état **don't care (-)** est souvent introduit par des outils de synthèse, il fournit un moyen de spécifier que le concepteur ne se soucie littéralement pas de la valeur de certaines entrées d'un bloc. Les valeurs de sortie peuvent être spécifiées ou calculées sans connaître les valeurs de ces entrées.

Les états X et W :

C’est l’état d’un signal inconnu : Si une entrée reçois deux états logique 0 et 1 à la fois, le système ne peut pas confirmer la valeur, alors on parle à l’état X.

De la même manière, si une entrée reçois deux états logiques L et H à la fois, le système ne peut pas confirmer la valeur, alors on parle à l’état W.

STD_LOGIC (Résolu):

std_logic est un type résolu. Il est légal d'avoir deux conducteurs ou plus sur un signal. Le résultat est déterminé par une "fonction de résolution" qui examine toutes les valeurs de conduite et les combine dans la valeur que vous verrez sur le signal.

```
CONSTANT resolution_table : stdlogic_table := (
-- ---------------------------------------------------------------------
--   |   U    X    0    1    Z    W    L    H    -                |     |
-- ---------------------------------------------------------------------
    (  'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U'  ),  -- |  U  |
    (  'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'  ),  -- |  X  |
    (  'U', 'X', '0', 'X', '0', '0', '0', '0', 'X'  ),  -- |  0  |
    (  'U', 'X', 'X', '1', '1', '1', '1', '1', 'X'  ),  -- |  1  |
    (  'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X'  ),  -- |  Z  |
    (  'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X'  ),  -- |  W  |
    (  'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X'  ),  -- |  L  |
    (  'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X'  ),  -- |  H  |
    (  'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X'  )   -- |  -  |
  );
```

L'entité

Une déclaration d'entité définie les ports d'interface entre une entité de conception et son environnement.

```
entity my_circuit_name is
  port( inp1: in  std_logic;
        inp2: in  std_logic;
        outp1: out  std_logic;
        outp2: out  std_logic;
        outp3: out  std_logic );
end my_circuit_name;
```

```
entity my_circuit_name is
  port( inp1: in  std_logic;
        inp2: in  std_logic;
        outp1: out  std_logic;
        outp2: out  std_logic;
        outp3: out  std_logic );
end entity;
```

```
entity my_circuit_name is
  port( inp1: in  std_logic;
        inp2: in  std_logic;
        outp1: out  std_logic;
        outp2: out  std_logic;
        outp3: out  std_logic );
end;
```

Les choses que vous devriez remarquer concernant la déclaration d'entité VHDL :

- Les commentaires commencent par --.
- Le premier caractère doit être une lettre
- L'identificateur ne doit pas se terminer par un souligné _.
- Dans un identificateur deux soulignés successives (__) sont interdits.
- Le langage n'est pas sensible à la casse des identificateurs. Il n'y a pas de différence entre deux identificateurs écrits l'un en majuscule et l'autre en minuscule.

L'architecture

Le corps d'une architecture définit le corps d'une entité de conception. Il spécifie les relations entre les entrées et les sorties d'une entité de conception. Cette spécification peut être exprimée sous forme **comportementale**, **structurelle**, **flot de données**, les trois formes peuvent coexister à l'intérieur d'un même corps d'architecture.

Toutes les instructions qui décrivent le corps de l'architecture sont des **instructions concurrentes** qui s'exécutent de façon asynchrone les unes par rapport aux autres.

L’architecture

La partie architecture d'un programme VHDL décrit le comportement interne du circuit électronique. Les données reçues des ports du circuit électronique sont traitées à l'intérieur de la partie architecture et les données de sortie sont obtenues. Les données de sortie produites sont envoyées aux ports de sortie du circuit électronique. La syntaxe générale de la partie architecture est décrite comme :

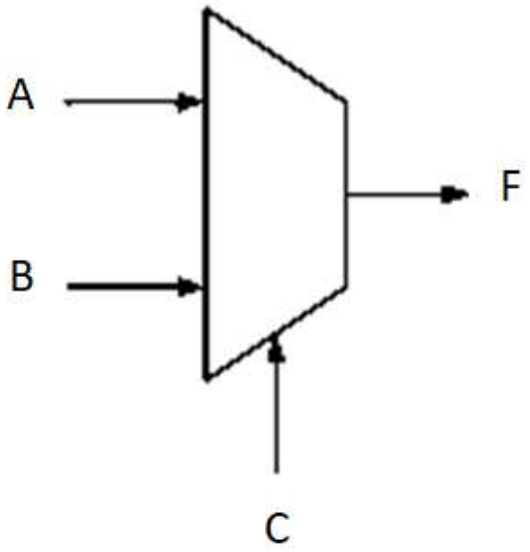
```
architecture nomArchitecture of nomEntité is
[déclarations sous-programmes, types, constantes,
signaux, composants…]
begin
instructions concurrentes
end [architecture] [nomArchitecture];
```

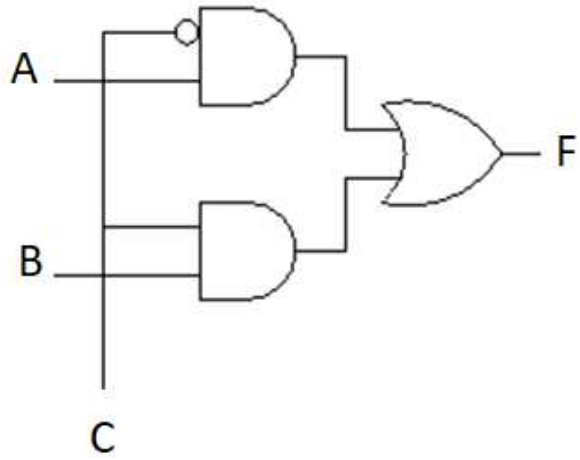
L’architecture : Styles de conception

Une architecture peut être écrite dans l'un des trois styles de conception de base:

1- Flot de données (datafellow) :
2- Comportementale (Behavioral).
3- Structurelle (Structural)

Soit l’exemple d’un multiplexeur suivant :





Flot de données (conception concurrente)

- Utilisation des équations booléennes

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           F : out STD_LOGIC);
end mux;

architecture Behavioral of mux is

begin
F <= (A and not C) or (C and B);
end Behavioral;
```

Flot de données (conception concurrente)

- Utilisation des instructions *when* et *else*

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           F : out STD_LOGIC);
end mux;

architecture Behavioral of mux is

begin
F <= A when C ='1'  else
     B;
end Behavioral;
```

- Le signal ou l'expression booléenne doit être comparé avec le bit par l'opérateur relationnel (=).
- Une demi-colonne est nécessaire le dernier résultat.

Flot de données (conception concurrente)

➢ Utilisation des instructions *with* et *select*

- Le signal ou l'expression booléenne doit être comparé avec le bit par l'opérateur relationnel (=).
- Une demi-colonne est nécessaire le dernier résultat.
- *When others*; est nécessaire dans le cas de la conception par l'affectation des signaux sélectionnés, pour indiquer que toutes les possibilités sont disponibles.
- L'ordre n'est pas important, et la déclaration de sortie exécutée est seulement celle qui remplit la condition.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           F : out STD_LOGIC);
end mux;

architecture Behavioral of mux is

begin
with C select F <=
       A when '0',
       B when '1',
       A when others ;
end Behavioral;
```

Conception comportementale (Behavioral).

- Généralement on utilise un ***processus*** pour cette méthode de conception, les instructions doivent être placées à l'intérieure d'un processus.
- On utilise généralement des instructions comme if, et déclarations de cas (case).
- Le ***processus*** complet est une instruction concurrente.
- Mais il faut noter que les instructions à l'intérieure d'un ***processus*** sont exécutés séquentiellement par le compilateur.
- La liste de sensibilité d'un ***processus*** contienne les signaux contrôlant le fonctionnement de la tache assurée par cette ***processus*** .
- Un processus est une boucle infinie , lorsqu'il arrive à la fin du code, il reprend automatiquement au début

Conception comportementale (Behavioral).

- Utilisation des instructions *if* et *else*

- La déclaration de processus est fait après *begin*.
- La liste de sensibilité contient tous les signaux influençant l'état du circuit.
- Tous les signaux d'entrée doivent être déclarés par la liste de sensibilité.
- Pour coder un processus combinatoire, l'utilisation du ***else*** est obligatoire de façon à traiter toutes les combinaisons (sinon il y a mémorisation donc c'est de la logique séquentielle)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           F : out STD_LOGIC);
end mux;
architecture Behavioral of mux is

begin
process(C)
   begin
   if c ='1' then
   F <= B; else
   F <= A;
   end if;
end process;
end Behavioral;
```

Conception comportementale (Behavioral).

- Utilisation des instructions *if* et *else*
  - Priorité :

```
if x = '1' then f <= a;
else if y = '1' then f <= b;
     else f <= c;
     end if;
end if;
```

X	Y	F	
1	1	A	Highest priority
1	0	A	↓
0	1	B	
0	0	C	Lowest priority

  - Il est parfois utile d'utiliser *if, elsif :*

```
if x = '1' then f <= a;
elsif y = '1' then f <= b;
else f <= c;
end if;
```

Conception comportementale (Behavioral).

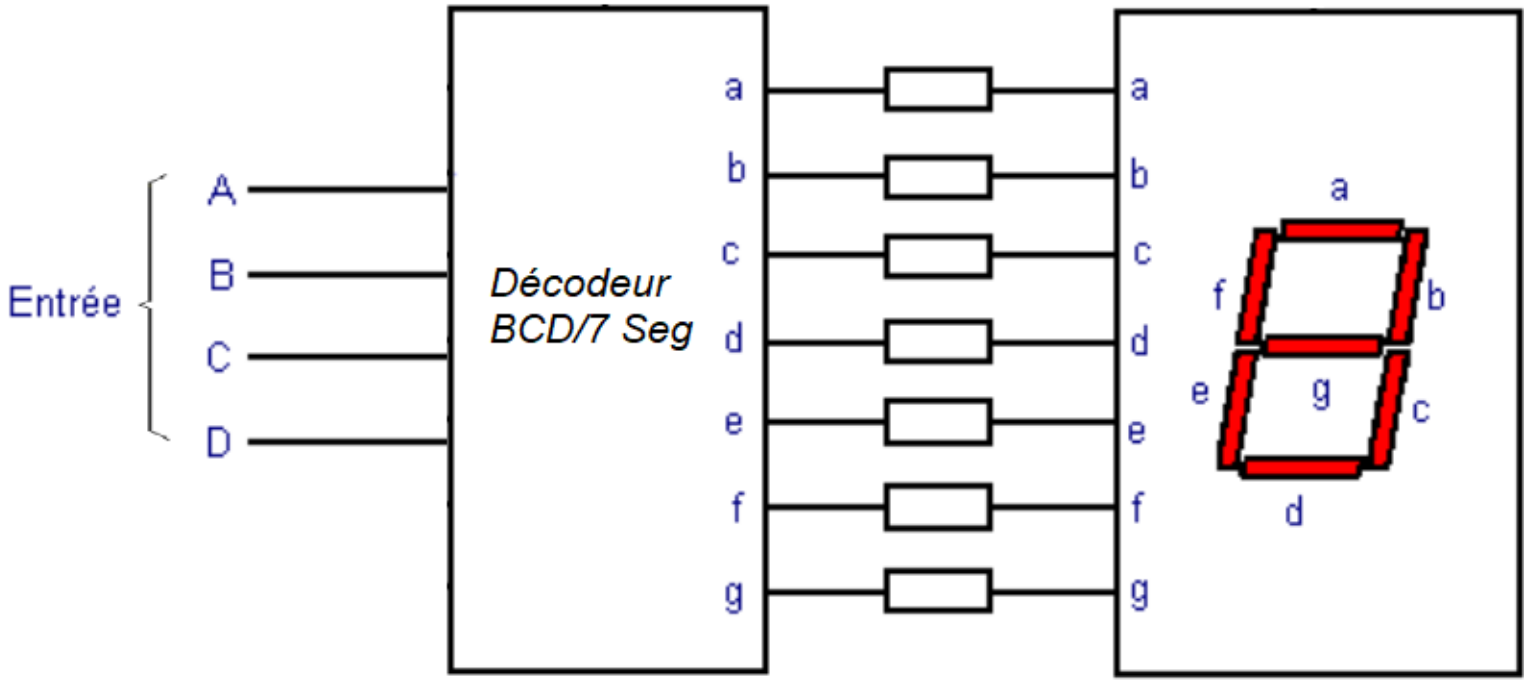
- Utilisation de l'instruction: *CASE*

- L' instructions *CASE* reposent sur le test d'un signal ou d'une variable. En fonction de la valeur, une instruction spécifique est exécutée.
- Pour coder un processus combinatoire, l'utilisation du ***when others*** est obligatoire de façon à traiter toutes les combinaisons (sinon il y a mémorisation donc c'est da la logique séquentielle)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity mux is
    Port ( A : in STD_LOGIC;
           B : in STD_LOGIC;
           C : in STD_LOGIC;
           F : out STD_LOGIC);
end mux;
architecture Behavioral of mux is

begin
process(C)
   begin
   case C is
   when '0' => F <= A;
   when '1' => F <= B;
   when others => F <= A;
   end case;
end process;
end Behavioral;
```

Exemple : control d'un afficheur à 7 segments

L'utilisation de l'instruction *case* est mieux adapté aux plusieurs situations, un simple exemple est le décodeur BCD (pour contrôler un afficheur à 7 segments) :

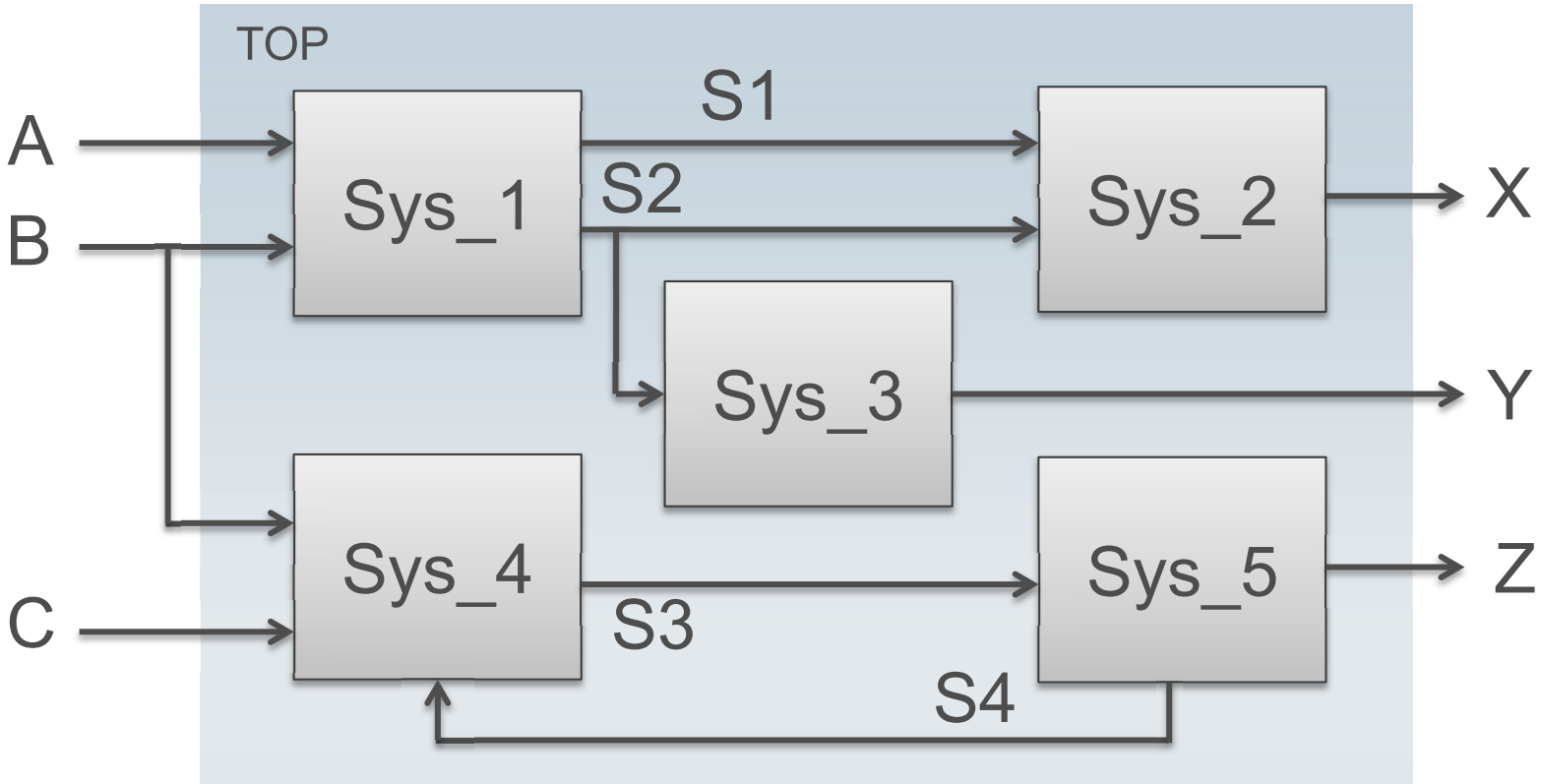


Exemple : control d'un afficheur à 7 segments

```
begin
process(C)
begin
case C is
   when "0000" => seg <= "1111110";
   when "0001" => seg <= "0110000";
   when "0010" => seg <= "1101101";
   when "0011" => seg <= "1111001";
   when "0100" => seg <= "0110011";
   when "0101" => seg <= "1011011";
   when "0110" => seg <= "1011111";
   when "0111" => seg <= "1110000";
   when "1000" => seg <= "1111111";
   when "1001" => seg <= "1111011";
   when others => seg <= "0000000";
end case;
end process;
end Behavioral;
```

Conception structurelle (structural):

Cette méthode de conception est un mélange de toutes les méthodes de conception vues auparavant, elle est utile pour la conception des systèmes compliqués. Elle est basée sur la division des taches compliquées en plusieurs taches simples.



Conception structurelle (structural):

La conception structurelle d'un système complexe en VHDL présente de nombreux avantages :

- Une architecture hiérarchique compréhensible : il est plus simple de séparer un circuit en un ensemble de blocs plus petits, ayant des fonctions bien identifiées. Ces blocs pourront alors être décrits sous forme comportementale, ou bien à leur tour être séparés en blocs encore plus simples.

- Une synthèse logique efficace : la synthèse est un processus lent (en terme de temps de calcul). Plus un bloc est gros et complexe, plus sa synthèse prendra du temps. Il vaut donc mieux travailler sur des blocs plus petits, plus simples à synthétiser, et rassembler le tout à la fin.

Conception structurelle (structural):

Les étapes d'une description structurelle:

1- La conception de chaque bloc séparément.

2- Création d'une entité (ex : top.vhd) qui rassemble tous les blocs créés.

3- La déclaration de ces blocs dans la l'entité 'top' comme composants (directive component). Ce composant peut être appelé plusieurs fois dans un même circuit.

4- Pour différencier ces mêmes composants, il est nécessaire de leur donner un nom d'"instance". L'appel d'un composant se dit aussi "instanciation" .

Exemple : prenant l'exemple vu auparavant

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SYS_1 is
  Port ( A : in STD_LOGIC;
         B : in STD_LOGIC;
         S1 : out STD_LOGIC;
         S2 : out STD_LOGIC);
end SYS_1;
architecture Behavioral of SYS_1 is

begin
S1 <= A xor B;
S2 <= A and B;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SYS_2 is
  Port ( S1 : in STD_LOGIC;
         S2 : in STD_LOGIC;
         X : out STD_LOGIC);
end SYS_2;
architecture Behavioral of SYS_2 is

begin
X <= S1 xor S2;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SYS_3 is
  Port ( S2 : in STD_LOGIC;
         Y : out STD_LOGIC);
end SYS_3;
architecture Behavioral of SYS_3 is

begin
Y <= not S2;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SYS_4 is
  Port ( B,C, S4 : in STD_LOGIC;
         S3 : out STD_LOGIC);
end SYS_4;
architecture Behavioral of SYS_4 is

begin
S3 <= (not B and C) xor S4 ;
end Behavioral;
```

Exemple : prenant l'exemple vu auparavant

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity SYS_5 is
  Port ( S3 : in STD_LOGIC;
         S4, Z : out STD_LOGIC);
end SYS_5;
architecture Behavioral of SYS_5 is

begin
S4 <= not S3 ;
Z <= S3;
end Behavioral;
```

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity top is
  Port ( A, B, C : in STD_LOGIC;
         X, Y ,Z : out STD_LOGIC);
end top;
architecture Behavioral of top is

begin

end Behavioral;
```

Exemple : prenant l'exemple vu auparavant

1- Déclaration de chaque composants dans le module 'top', Le mot clé component sert à déclarer le prototype d'interconnexion. La syntaxe est presque identique à celle de l'entité :

```
entity top is
  Port ( A, B, C : in STD_LOGIC;
         X, Y ,Z : out STD_LOGIC);
end top;
architecture Behavioral of top is
component SYS_1 is
port ( A, B : in std_logic;
       S1, S2 : out std_logic);
end component;
-----------------------------------------
component SYS_2 is
port ( S1, S2 : in std_logic;
       X : out std_logic);
end component;
-----------------------------------------
```

```
component SYS_3 is
port ( S2 : in std_logic;
       Y : out std_logic);
end component;
-----------------------------------------
component SYS_4 is
port ( B, C, S4 : in std_logic;
       S3 : out std_logic);
end component;
-----------------------------------------
component SYS_5 is
port ( S3 : in std_logic;
       S4, Z : out std_logic);
end component;
-----------------------------------------
```

Exemple : prenant l'exemple vu auparavant

2- Déclaration des signaux internes:

```
-----------------------------------------------
component SYS_5 is
port ( S3 : in std_logic;
       S4, Z : out std_logic);
end component;
-----------------------------------------------
signal S1, S2, S3, S4 : std_logic;

begin

end Behavioral;
```

Exemple : prenant l’exemple vu auparavant

3- Instanciation : L'instanciation d'un composant se fait dans le corps de l'architecture de cette façon :

<NOM_INSTANCE>:<NOM_COMPOSANT> port map (LISTE DES CONNEXIONS);

```
-----------------------------------------------
signal S1, S2, S3, S4 : std_logic;

begin
PC1 : SYS_1 port map (A => A, B => B, S1 => S1, S2 => S2);
PC2 : SYS_2 port map (S1 => S1, S2 => S2, X => X);
PC3 : SYS_3 port map (S2 => S2, Y => Y);
PC4 : SYS_4 port map (C => C, B => B, S4 => S4, S3 => S3);
PC5 : SYS_5 port map (S3 => S3, S4 => S4, Z => Z);
end Behavioral;
```

Références

[1] Barr, Keith (2007). ASIC Design in the Silicon Sandbox: A Complete Guide to Building Mixed-signal Integrated Circuits. New York: McGraw-Hill. ISBN 978-0-07-148161-8.

[2] Smith, Michael John Sebastian (1997). Application-Specific Integrated Circuits. Addison-Wesley Professional. ISBN 978-0-201-50022-6.

[3] Maxfield, Clive. The design warrior's guide to FPGAs: devices, tools and flows. Elsevier, 2004.

[4] Place and Route for FPGAs, Western University, CANADA, http://www.eng.uwo.ca/

[5] Richard S. Sandige, Michael L. Sandige , Fundamentals of Digital and Computer Design with VHDL, McGraw-Hill, ISBN-10: 0-07-338069-5.

[6] William Kafig, VHDL 101, Everything you need to know to get started, Elservier, 2011, ISBN: 978-1-85617-704-7