



SPÉCIFICATION DES CIRCUITS INTÉGRÉS

TÉLÉCOM NANCY

Slaviša Jovanović et Yves Berviller

{slavisa.jovanovic,yves.berviller}@univ-lorraine.fr
<http://www.ijl.univ-lorraine.fr/>

13 janvier 2020



UNIVERSITÉ
DE LORRAINE



Institut Jean Lamour



OBJECTIFS ET CONTENU DU MODULE

- L'objectif principal de ce cours est de, à partir d'un cahier des charges initial, concevoir un circuit numérique en VHDL synthétisable et le tester sur une plate-forme FPGA
- Développer, simuler, réaliser et programmer un microcontrôleur RISC *ex-nihilo*.

Pour ce faire, vous allez apprendre :

- Les bases de la conception d'un circuit numérique (combinatoire et/ou séquentiel)
- Le langage de description de matériel VHDL
- Altera Quartus FPGA software development tools
- spécifier des composants en langage VHDL
- simuler des composants en VHDL



OBJECTIFS ET CONTENU DU MODULE

- optimiser la synthèse
- maîtriser les architecture et fonctionnement des FPGA
- prédire performances et ressources nécessaires
- produire les fichiers de configuration par synthèse automatique

Contenu du cours :

- Introduction aux circuits intégrés
- Circuits intégrés programmables : FPGA, CPLD
- Processus de développement
- Le langage de spécification de matériel VHDL.
- Spécification des composants en VHDL :
 - ▷ multiplexeur, additionneur, registres, compteur, bloc de registres ;

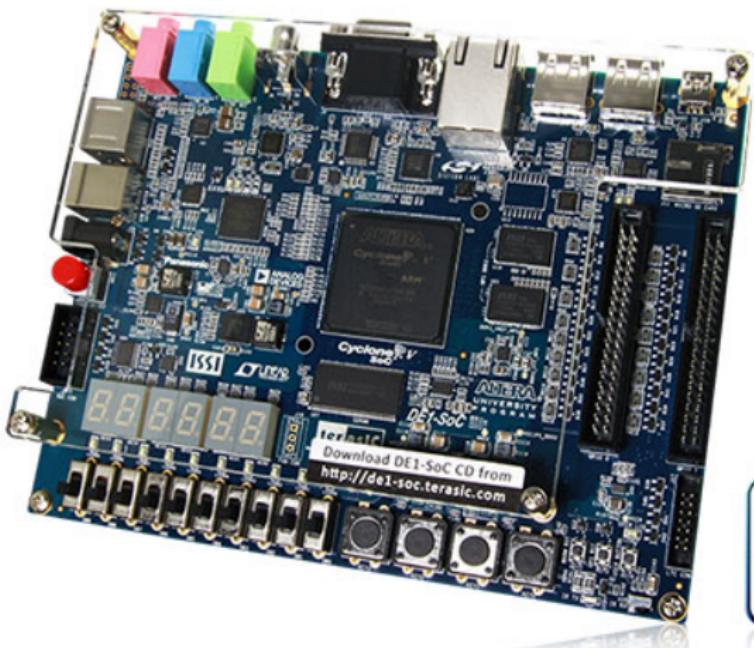


OBJECTIFS ET CONTENU DU MODULE

- L'objectif principal :
 - ▷ être capable de réaliser un circuit numérique de faible complexité en technologie FPGA et de l'optimiser par rapport aux différentes contraintes : taille (coût) et vitesse
- Prérequis :
 - ▷ Logique booléenne, électronique numérique de base

OBJECTIFS ET CONTENU DU MODULE

Plateforme de test : Altera DE1-SoC + Altera Quartus 13.0



INFORMATIONS SUR LE COURS

- Cours/TD/TP : 48h (24 séances)

- Cours/TD/TP : S. Jovanović et Y. Berviller
slavisa.jovanovic@univ-lorraine.fr
yves.berviller@univ-lorraine.fr
Institut Jean Lamour, N2EV - MAE

- Salle cours & TD : Salles Télécom

- Adresse web interne : ARCHE Spécification des Circuits Intégrés

- Manuels : les diapos du cours



RÉFÉRENCES

- Pong P Chu, *RTL hardware design using VHDL : coding for efficiency, portability, and scalability*, John Wiley & Sons, 2006.
- Jacques Weber and Sébastien Moutault, *Le langage VHDL : du langage au circuit, du circuit au langage-4e édition : Cours et exercices corrigés*, Dunod, 2011.



Contrôle des connaissances

- Évaluation terminale = 75%
 - Examen final, au mois de mai
- Évaluation continue = 25%
 - devoirs à rendre
- Note finale :
 - $N = N_E * 0.75 + N_D * 0.25$

SOMMAIRE

1 Introduction

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

MOTIVATIONS

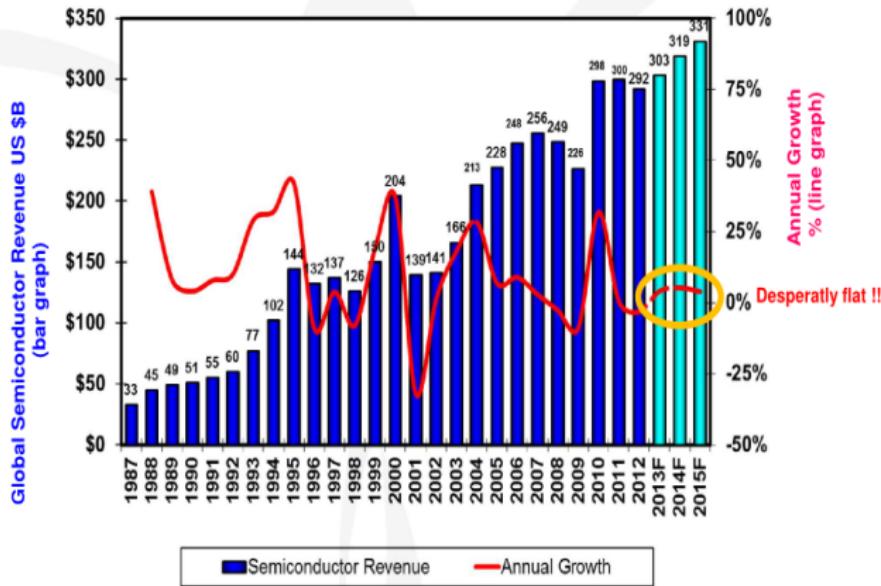
MARCHÉ MONDIAL DE L'ÉLECTRONIQUE PAR SEGMENTS



Source : DECISION March 2013 (Embedded Systems)

MOTIVATIONS

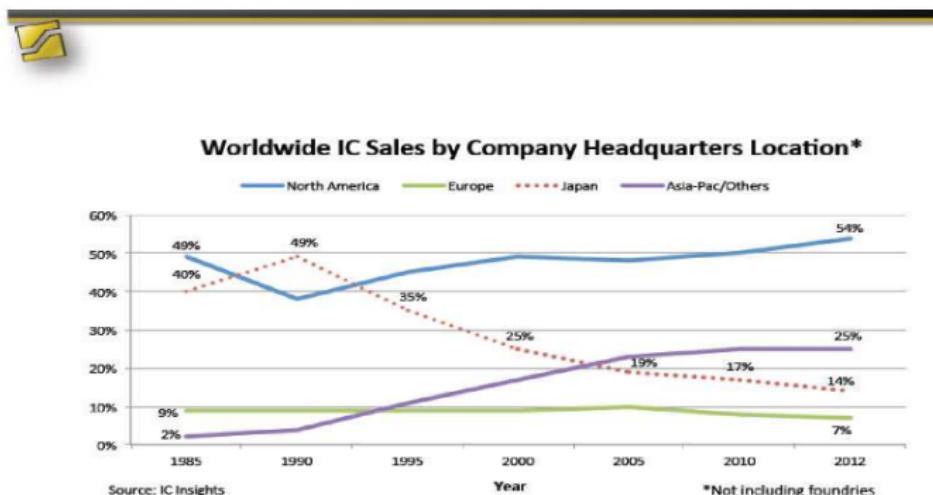
CYCLES DE L'INDUSTRIE DE SEMICONDUCTEURS



Source: SEMI 2013 : SIA/WSTS historical year end reports, WSTS

MOTIVATIONS

VENTES DE SEMICONDUCTEURS DANS LE MONDE



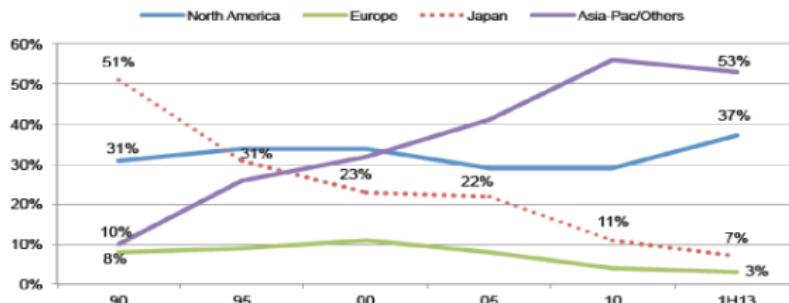
- L'Europe et le Japon en baisse

MOTIVATIONS

LA PRODUCTION DE SEMICONDUCTEURS DANS LE MONDE



Semiconductor Capital Expenditures by Region



Source: IC Insights

- L'Europe et le Japon à la peine ...

MOTIVATIONS

CES 2014 - *chips for everything*



Infomotions



Reebok



Sports



Sony Tennis



ibitz



Fitbit, bitfit, fitfit ...



Instabeats



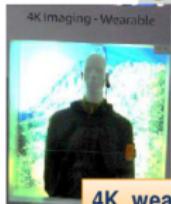
Interchangeable ehealth



UV jewel

MOTIVATIONS

CES 2014 - *Internet of things*



MOTIVATIONS

CES 2014 - other applications



Robots



SmartKeys



Brain sensing



AirDrones



Immersive Gaming



3D scanners



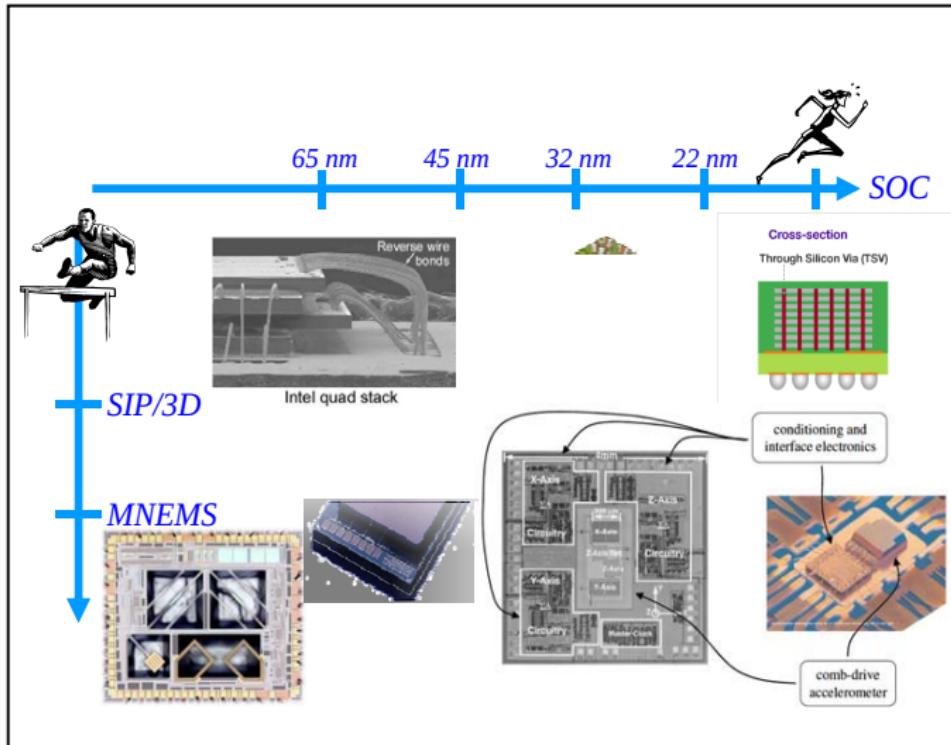
3D printers

MOTIVATIONS

DÉFIS À RELEVER

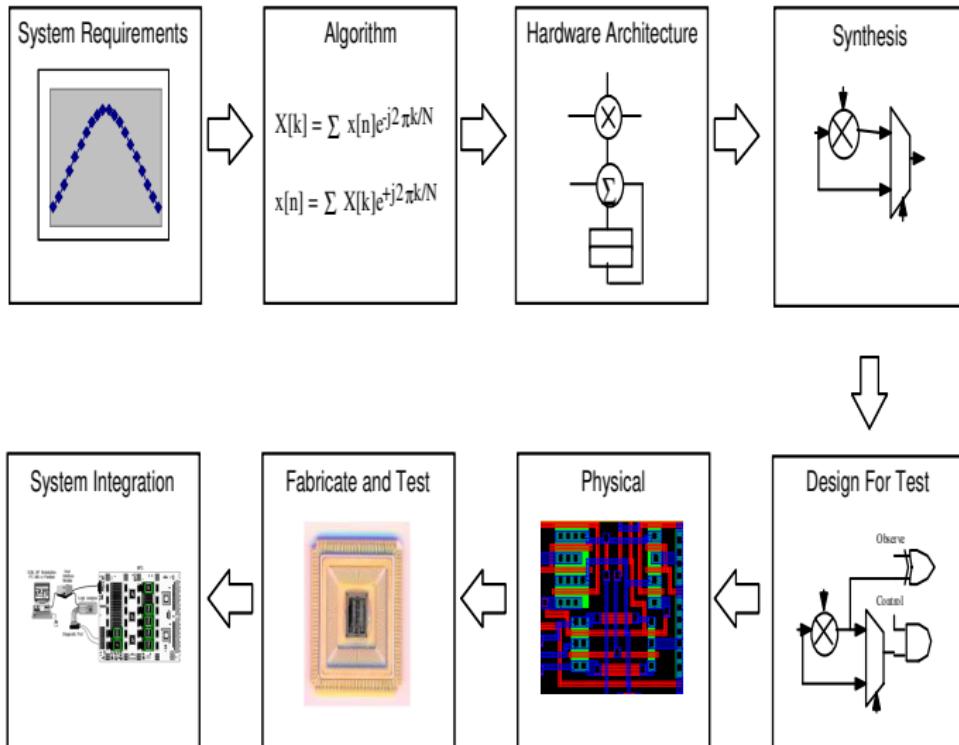
- De nouvelles puces plus performantes pour irriguer les nouveaux marchés de croissance
 - ▷ santé, sécurité, contrôle de l'énergie, objets communications, ...
- Ruptures au niveau technologique au plan système
- Technologies *More than Moore* et *More Moore*
- marché européen : analogique et MEMS, RF, capteurs, FDSOI, ...
- Les CI : plus petit, moins cher, plus mobile, plus performant, plus fonctionnel, ...
- Les CI : plus fiable et plus autonome (consommation)
→ *energy harvesting*
- Va-t-on vers des circuits auto-alimentés ?
- La réponse dans quelques années !

MOTIVATIONS



MOTIVATIONS

DU CAHIER DES CHARGES À LA PUCE



SOMMAIRE

1 Introduction

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

PREMIER ORDINATEUR

ORDINATEUR MÉCANIQUE

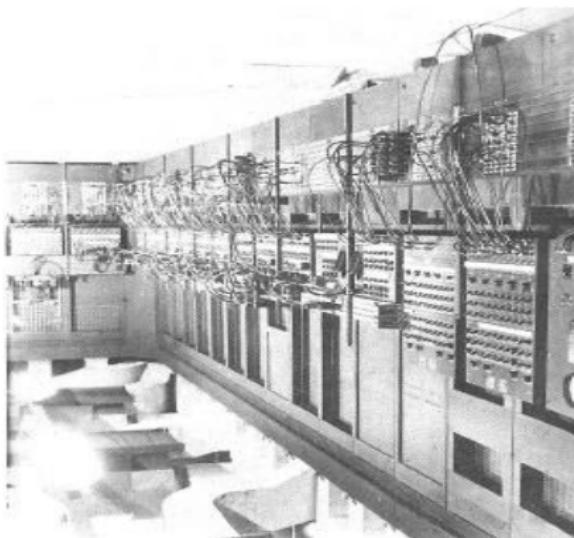


- 1832 : « *The Babbage Difference Engine* »
- 25,000 pièces
- coût 17,470 livres sterling

PREMIER ORDINATEUR

ORDINATEUR ÉLECTRONIQUE À TUBES À VIDE

- 1943 : ENIAC
 - ▷ 30 tonnes
 - ▷ 42 armoires de 3m de haut
 - ▷ $167 m^2$
 - ▷ 50 000 résistances
 - ▷ 10 000 condensateurs
 - ▷ 6 000 commutateurs
 - ▷ 17 468 tubes à vide
 - ▷ consommation : 174 KW
 - ▷ une famille/an \approx 7000 KWh
 - ▷ ENIAC en 40h de fonctionnement



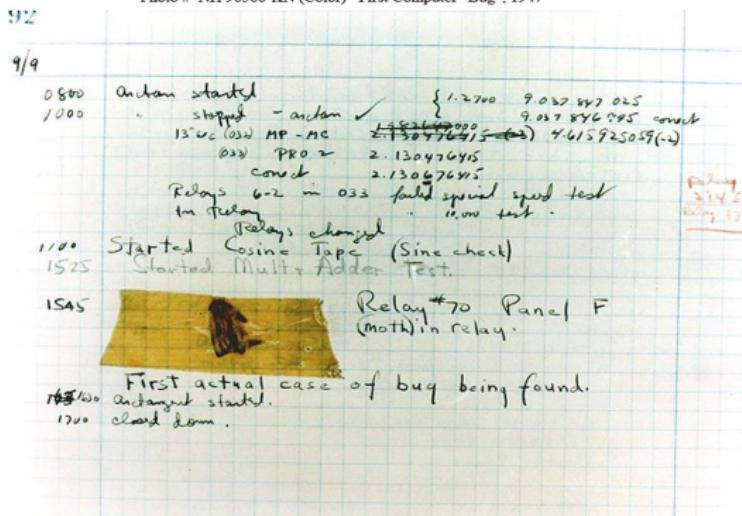


PREMIER ORDINATEUR

ORDINATEUR ÉLECTRONIQUE À L'ORIGINE DU *bug*

- 1945 : Harvard Mark II
 - ▷ Premier BUG

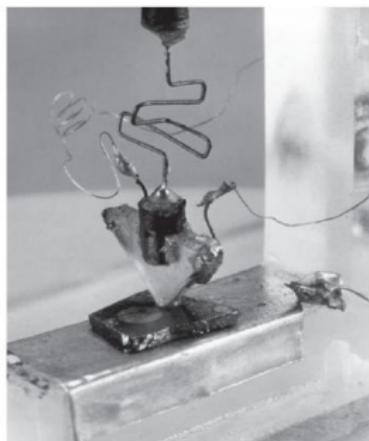
Photo # NH 96566-KN (Color) First Computer "Bug", 1947



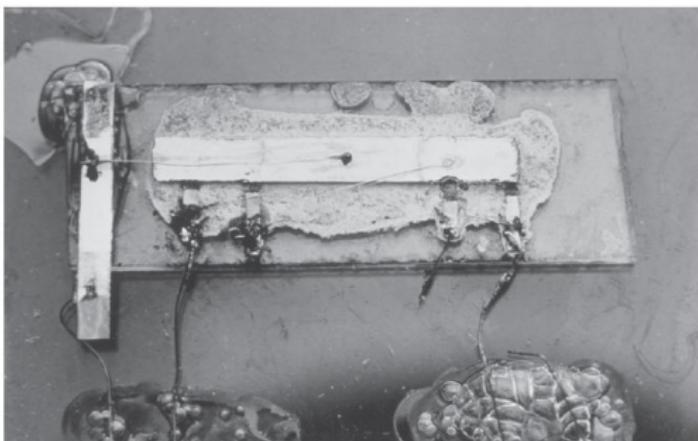
TRANSISTOR BIPOLAIRE

INTRODUCTION

- Invention du transistor en 1947 - *Bell labs*
(Bardeen, Brattain et Shockley)
 - ▷ fiable
 - ▷ moins susceptible au bruit
 - ▷ faible consommation par rapport aux tubes à vide
- Invention du premier circuit intégré (IC)



a)

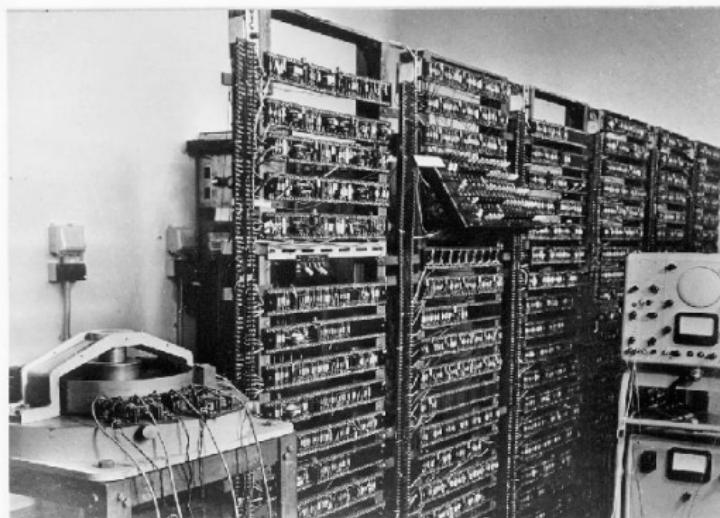


(b)

TRANSISTOR BIPOLAIRE

ORDINATEUR À TRANSISTORS BIPOLAIRES

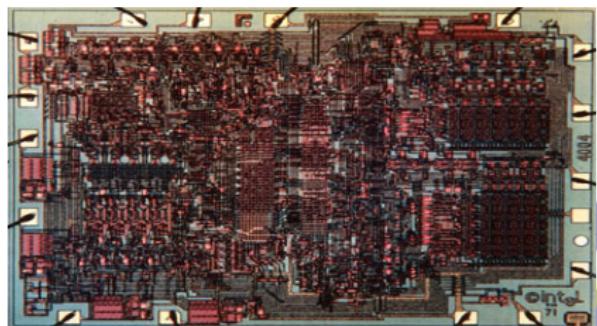
- 1953 : Ordinateur à transistors
 - ▷ 92 transistors
 - ▷ 550 diodes



TRANSISTOR À EFFET DE CHAMP

INTRODUCTION

- Réalisation du transistor à effet de champ (*MOS*) en 1963
 - ▷ moins rapide que le transistor bipolaire
 - ▷ plus efficient en consommation
- Deux types : NMOS et PMOS
- Au début PMOS était dominant, NMOS depuis 1970
- 1971 : INTEL 4004
 - ▷ Premier microprocesseur commercialisé
 - ▷ Fonctionne sur 4 bits
 - ▷ 2 250 NMOS transistors
 - ▷ technologie 10 μm
 - ▷ puissance d'un ENIAC
 - ▷ 740 KHz
 - ▷ alimentation 15V
 - ▷ 90 000 instructions / sec



TRANSISTOR CMOS

INTRODUCTION ET PROPRIÉTÉS

- CMOS (*Complementary Metal-Oxide Semiconductor*)
- MOS transistor inventé en 1935 par Oskar Heil
- Avancées technologiques permettent sa fabrication dans les années 80
- composé d'un NMOS + PMOS
- consommation statique presque 0
- consommation dynamique lors des transitions
 $0 \rightarrow 1$ et $1 \rightarrow 0$
- faible coût
- toujours moins rapide que la technologie TTL
- se prête bien à l'intégration à très grande échelle (*VLSI*)

TRANSISTOR CMOS

RÉVOLUTION MICROÉLECTRONIQUE

- Petit transistor
 - ▷ faible consommation
 - ▷ faible vitesse de propagation de signal → plus rapide
 - ▷ faible coût de fabrication
 - ▷ isolation naturelle
- coût de fabrication par cm^2
- plus le transistor est petit → plus de transistors au cm^2
 - plus le transistor est moins cher
- technologie de fabrication du processeur Intel 4004 : $10 \mu m$
- technologie actuelle $14 nm$
- $\frac{10\mu m}{14nm} = 714.28$
- conséquence : démocratisation totale des circuits VLSI et leur omniprésence dans tous les domaines d'applications

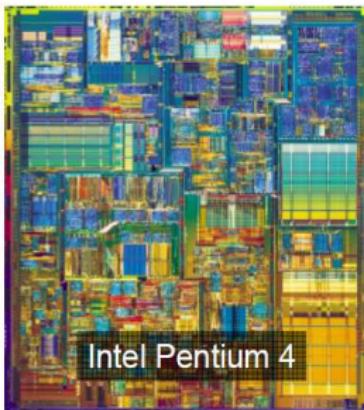
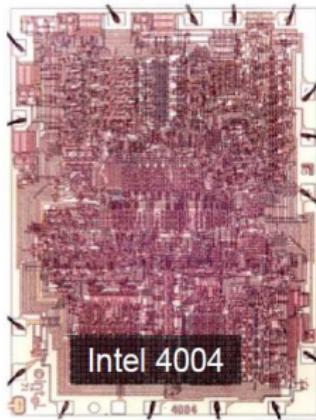
TRANSISTOR CMOS

RÉVOLUTION MICROÉLECTRONIQUE

- Intel annonçait les circuits en technologie 14nm pour le début de l'année 2014 → novembre 2014 processeur 5Y30 à base de FinFET de 2nde génération
- Chenming Hu, le co-inventeur du transistor FinFET :
« Nobody knows anymore what 16nm means or what 14nm means. »
- Certains considèrent que dernièrement ces chiffres ont été détournés à des fins commerciales
- Ces chiffres cachent les écarts au niveau des procédés technologiques des principaux fabricants de circuits
- Par exemple, pour la technologie 130nm, la longueur du canal des circuits Intel était de l'ordre de 70nm
 - ▷ *strain engineering*
 - ▷ nouveaux types d'isolants pour la réalisation de la grille
 - ▷ nouvelles structures de transistor (*FinFET, tri-gates,...*)
 - ▷ etc.

TRANSISTOR CMOS

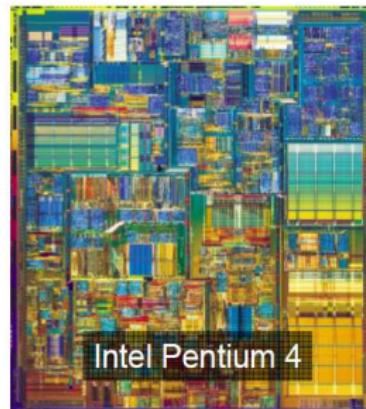
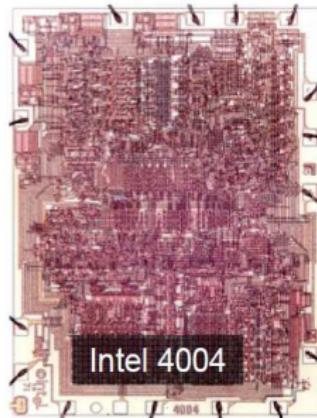
ÉVOLUTION



| Year | 1971 | 2001 |
|----------------------|-------|------------|
| Transistors | 2,300 | 42,000,000 |
| Speed (kHz) | 108 | 2,000,000 |
| CD (μm) | 10.00 | 0.13 |

TRANSISTOR CMOS

ÉVOLUTION



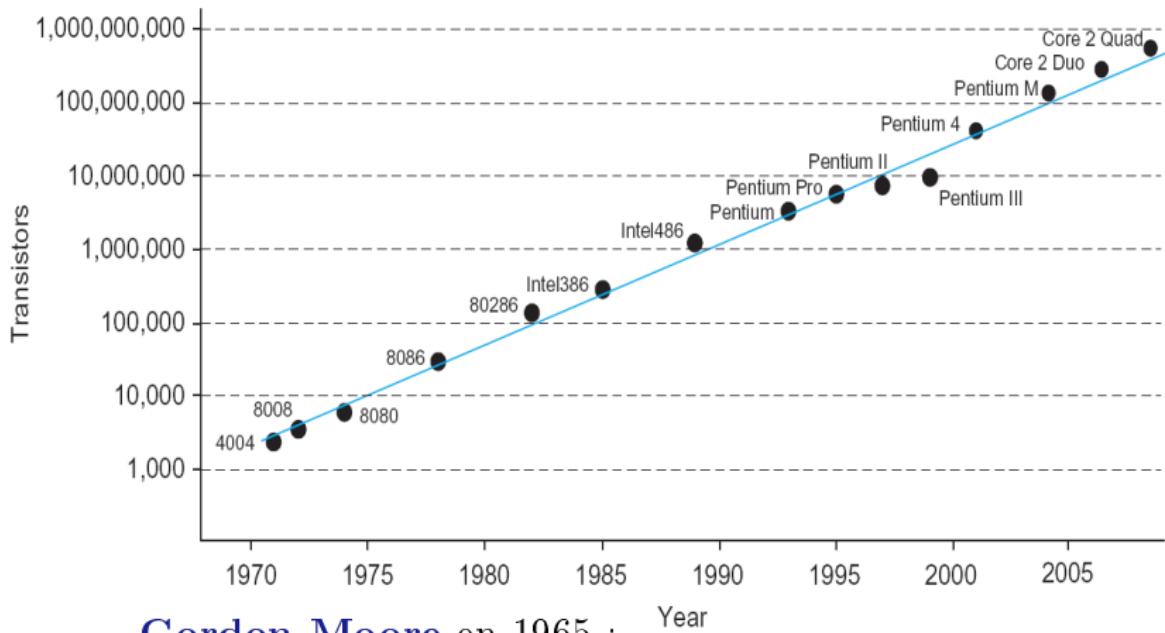
□ Intel i7 Sandy Bridge-E :

- ▷ 32 nm process technology
- ▷ 8 coeurs physiques
- ▷ 2270 million de transistors
- ▷ paru en novembre 2011

□ Intel i7 Ivy Bridge :

- ▷ 22 nm
- ▷ *Tri-gate* transistors
- ▷ courants de fuite plus faibles
→ gain en consommation

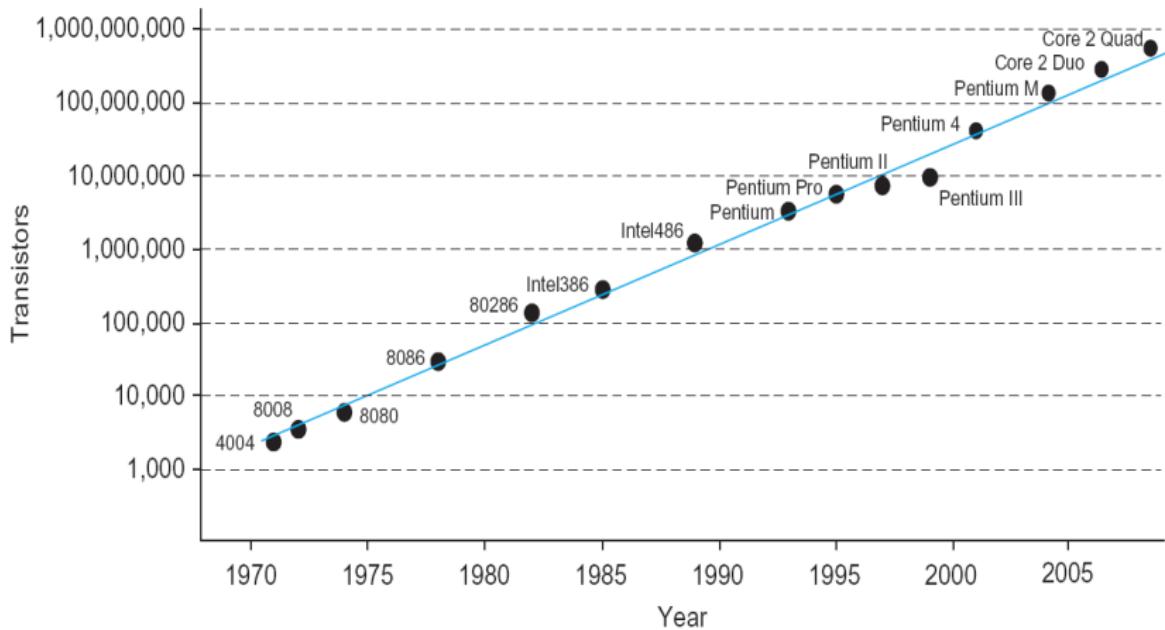
LOI DE MOORE



Gordon Moore en 1965 :

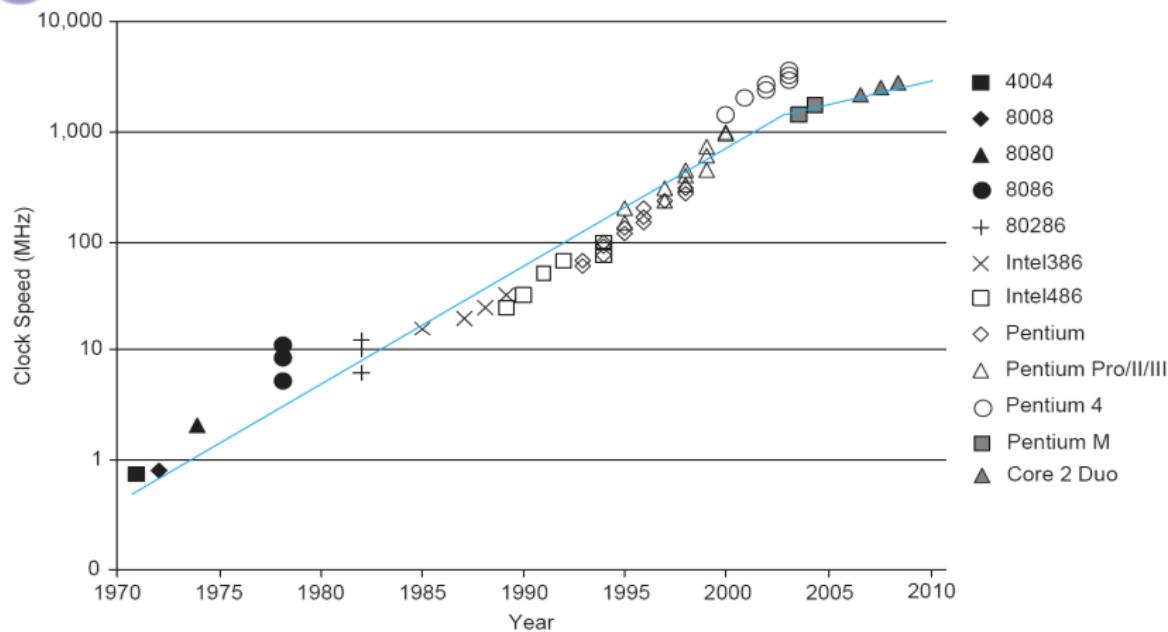
- Le nombre de transistors sur une puce double tous les 2 ans

LOI DE MOORE



- Le nombre de transistors sur des processeurs Intel double tous les 26 mois

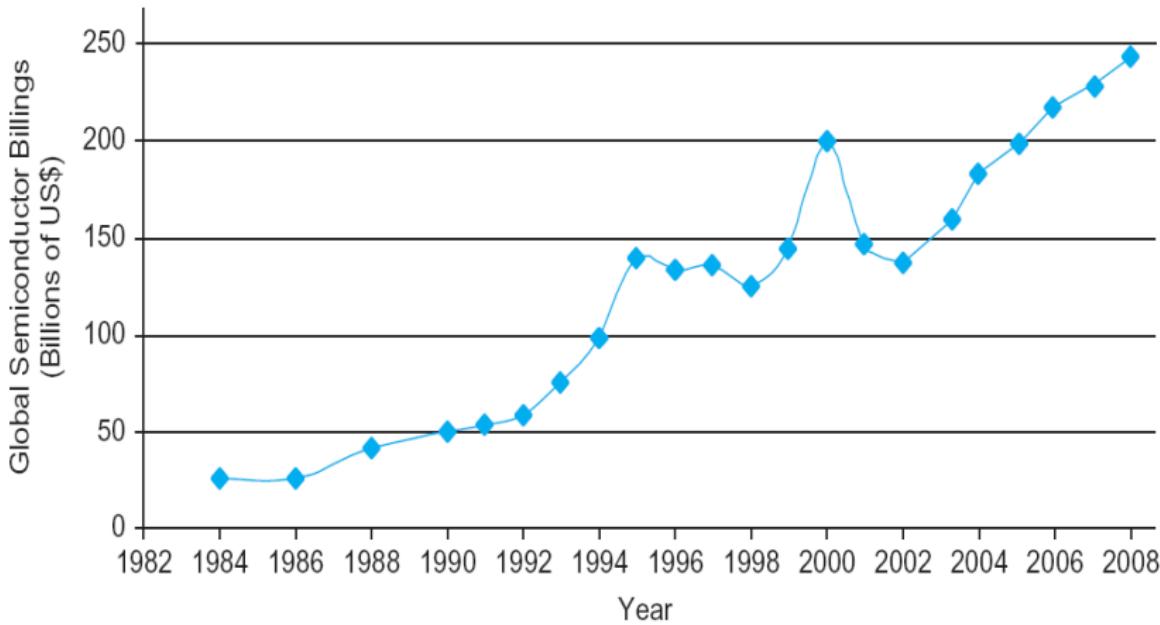
LOI DE MOORE II



- La fréquence de fonctionnement des processeurs Intel double tous les 36 mois

ENJEUX ÉCONOMIQUES

MARCHÉ DES SEMICONDUCTEURS



- un marché très volumineux
- revenu total annuel de quelques centaines de milliards

SOMMAIRE

1 Introduction

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

INTRODUCTION

DOMAINES D'APPLICATIONS

Digital

CPU,
Mémoire,
...

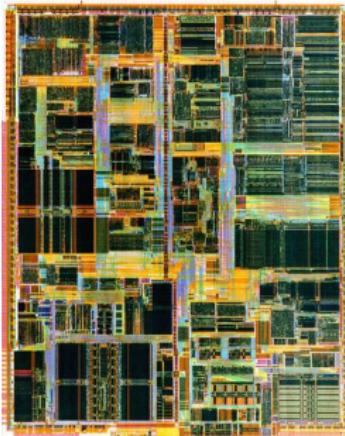
DSP

Audio/video
Mpeg
...

Application

multimédia
Communication,
Calcul ...

systems VLSI / SoC



MEMS

μcapteurs CCD ..
μtransformateurs
μrésonateurs ..

RF/ analogique

Filtres LNA,
mélangeurs,
VCO ...

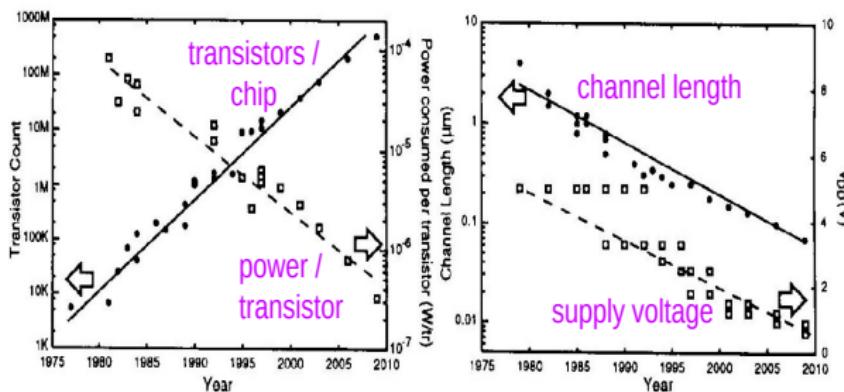
Gestion
d'énergie

Convertisseurs,
régulateurs ..

TECHNOLOGIE CMOS

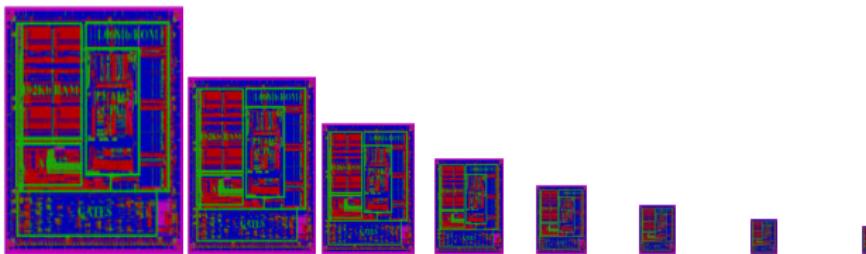
TENDANCES

- le nombre de transistors par chip en augmentation avec le temps
- puissance consommée/transistor en baisse avec le temps
- la longueur du canal diminue avec le temps
- la tension d'alimentation diminue avec le temps



TECHNOLOGIE CMOS

EXEMPLE D'ÉVOLUTION DANS LE DOMAINE DE LA TÉLÉPHONIE

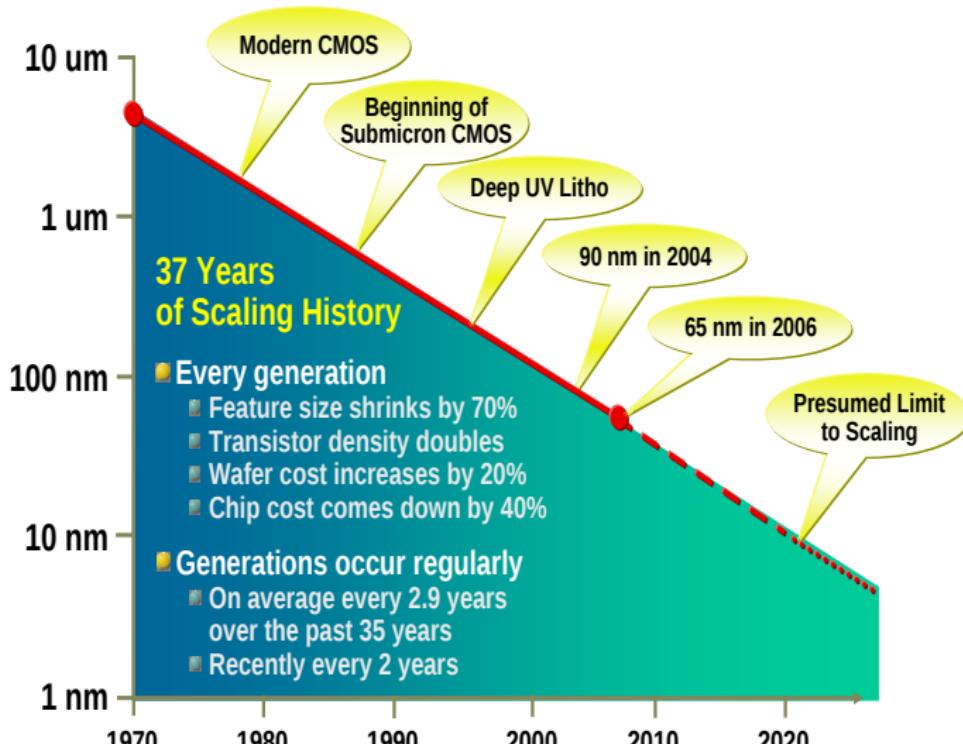


| Year | 1994 | 1997 | 1999 | 2000 | 2002 | 2004 | 2006 | 2008 |
|-----------------------------|-------|-------|-------|-------|--------|--------|--------|--------|
| Nano-meter | 500nm | 350nm | 250nm | 180nm | 130nm | 90nm | 65nm | 45nm |
| Wafer size | 6" | 8" | 8" | 8" | 12" | 12" | 12" | 12" |
| Die size (mm ²) | 80.7 | 46.6 | 19.2 | 10.7 | 6.7 | 4.2 | 2.4 | 1.4 |
| Dies per wafer | 310 | 950 | 2550 | 4700 | 12,200 | 18,700 | 26,500 | 46,500 |

→ 150X increase in die per wafer →

TECHNOLOGIE CMOS

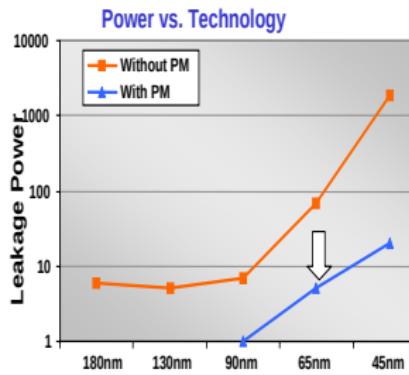
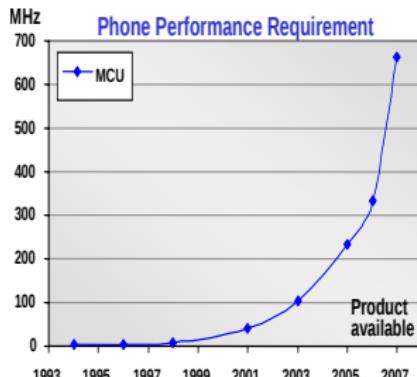
Technology Scaling



TECHNOLOGIE CMOS

CONSOMMATION

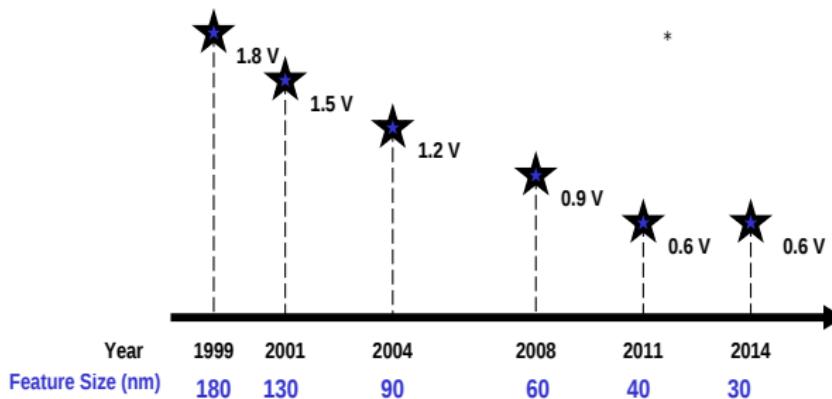
- $P = f(C, V^2, F, \text{Leakage})$
- C diminue avec les avancées technologiques
- V pratiquement constant, au niveau le plus bas possible (?)
- F augmente avec les avancées technologiques
- *Leakage* (courants de fuite) augmente avec les avancées technologiques et la température (qui augmente avec la puissance)



TECHNOLOGIE CMOS

TENSION D'ALIMENTATION

- Pronostics en 2000 par l'ITRS
- La technologie 65nm atteinte en 2006
- La technologie 45nm atteinte en 2008

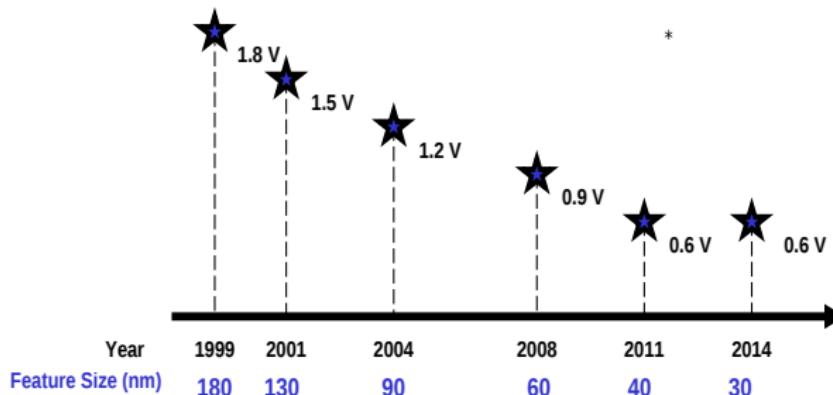


TECHNOLOGIE CMOS

TENSION D'ALIMENTATION

Robert X. Cringely

If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost \$100, get one million miles to the gallon, and explode once a year . . .



TECHNOLOGIE CMOS

TRANSISTOR

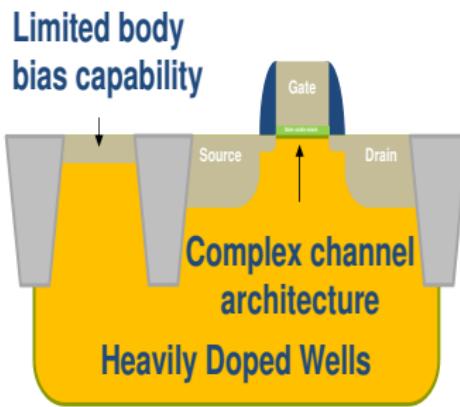
- fin du transistor MOS ?
- moins d'atomes de Si dans le canal
- beaucoup moins de porteurs de courant dans le canal
- une variabilité plus accentuée (V_{TH})
- ratio $\frac{I_{ON}}{I_{OFF}}$ dégradé

Deux solutions actuelles :

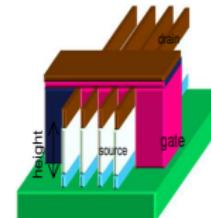
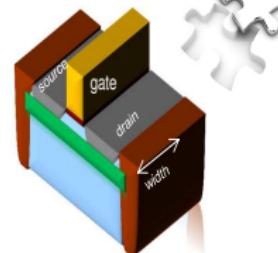
- FDSOI et
- FinFET

TECHNOLOGIE CMOS

TRANSISTOR



FD-SOI = 2D



FinFET = 3D

CLASSIFICATION DE CI

TAILLE ET DENSITÉ D'INTÉGRATION

- SSI** (*Small-Scale Integration*) < 12 portes logiques dans un boîtier
- MSI** (*Medium-Scale Integration*) entre 13 et 99 portes logiques
- LSI** (*Large-Scale Integration*) > 100 portes logiques
- VLSI** (*Very Large-Scale Integration*) > 100,000 de transistors
- ULSI** (*Ultra Large-Scale Integration*) plusieurs millions de portes logiques sur une puce

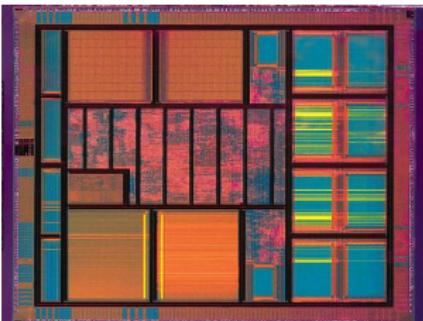
Exemple : Réalisation d'un système de comptage comprenant 3 compteurs de 4 bits

- 1963** : 36 transistors et 244 diodes
- 1966** : 13 circuits SSI en technologie RTL (*Resistor-Transistor Logic*)
- 1969** : 3 circuits LSI en technologie TTL (*Transistor-Transistor Logic*)
- Aujourd'hui** : une cellule élémentaire d'un circuit spécifique ou programmable

CIRCUITS INTÉGRÉS VLSI

INTRODUCTION

- **VLSI** : *Very Large Scale Integration* - Intégration à Très Grande Échelle (ITGE)
La densité d'intégration > 100,000
- Réalisée pour la première fois en **1980**
- **Exemple** : un microprocesseur est un circuit VLSI
- **VLSI** désigne :
 - ▷ les puces elles-mêmes,
 - ▷ les techniques de conception,
 - ▷ la science relative aux CI à haute densité



CIRCUITS INTÉGRÉS VLSI

INNOVATIONS TECHNOLOGIQUES

- Progrès conjoint du développement VLSI et innovation technologique
 - ▷ Progrès en techniques de lithographie, métallisation, ... → nouveaux produits
 - ▷ Forte demande du marché de produits particuliers → impact sur la recherche technologique
- Produits **VLSI** émergents :
 - ▷ Intégration de divers systèmes à technologie spécifique au sein d'une même puce (*SoC*)
- **ULSI** (*Ultra Large Scale Integration*) :
 - ▷ **SoC** : *System on a Chip*
 - ▷ **3D-IC** : *3D Integrated Circuits*
- Le procédé utilisant le Si est à l'origine de la plupart des progrès en terme de forte intégration de circuits et de systèmes

CIRCUITS INTÉGRÉS VLSI

CLASSIFICATION PAR LA NATURE DE SIGNAL D'ENTRÉE

Types de circuits VLSI :

- Analogique,
 - ▷ Circuits d'instrumentation,
 - ▷ Systèmes HF (plages de fréquence différentes et variées),
 - ▷ Circuits de puissance
 - faiblement intégrés
 - forts courants
 - hautes températures
 - technologie hybride (\sim KW)
- Numérique et
 - ▷ Composé essentiellement de circuits logiques et de mémoires
 - ▷ Peuvent contenir des parties mixtes : les interfaces avec le monde réel CAN (ADC) et CNA (DAC)
- Mixte
- Domaines d'applications :
 - + de 90 % numérique /- de 10 % analogique (interface avec monde réel)



CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS

Classification selon l'usage :

- **Circuits Standard** : fonctions produites en grande série et disponible sur catalogue
- **ASIC (*Application Specific Integrated Circuits*)** : Circuits intégrés plus complexes et spécifiques à une application
 - ▷ regroupent sur une même puce l'équivalent de plusieurs circuits standard
 - ▷ Réduction de la taille du circuit
 - ▷ Gain en fiabilité
 - ▷ Gain en vitesse



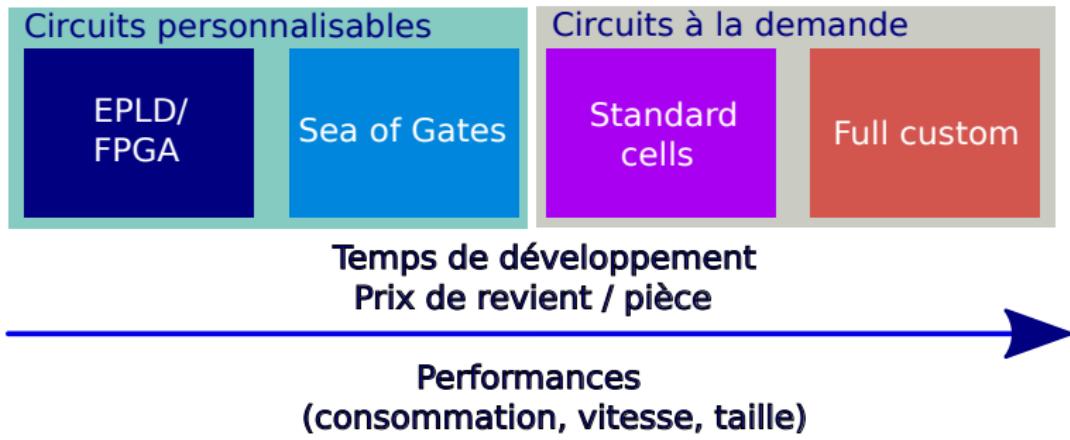
CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS

- Circuits personnalisables (semi-spécifiques) :
 - ▷ programmables : **EPLD / FPGA** - architecture figée → le concepteur active les connexions et programme la logique qui l'intéresse
 - ▷ prédiffusés : **Sea of Gates** - les niveaux d'interconnexions restent à définir et sont envoyés au fabricant
- Circuits à la demande (spécifiques) :
 - ▷ précaractérisés : **Standard Cells**
 - Le circuit est construit à partir d'éléments de bibliothèques ou *IP (Intellectual Property)* dont les caractéristiques sont connues
 - Le concepteur décide de l'architecture et contrôle l'assemblage
 - Le fabricant réalise tous les niveaux de masque (*layout*)
 - ▷ *full custom* :
 - Tous les éléments utilisés sont développés par le concepteur
 - Le fabricant réalise tous les niveaux de masque (*layout*)

CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS



- Les CI les plus couramment utilisés sont de type standard, programmable ou pré-caractérisé

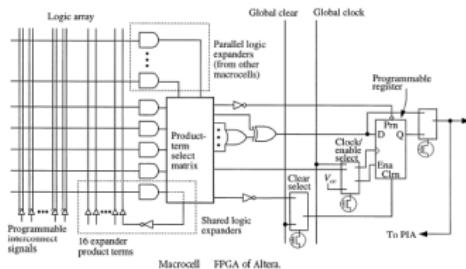
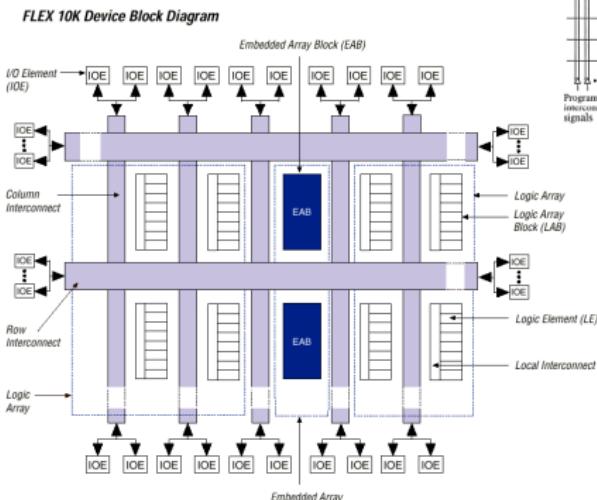
CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS

- Intérêt du *Full Custom* ?
- Création de :
 - circuits standard
 - circuits programmables
 - bibliothèques de cellules standard ou d'*IP*
- Utilisation des technologies les plus récentes
- Contraintes de performances trop fortes pour les composants existants
- Conception de circuits intégrés analogiques, mixtes ou de *MEMS*
- Rentable si grande série ou pas d'autre solution

CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : FPGA



le bloc logique reconfigurable
CLB

matrice d'un FPGA

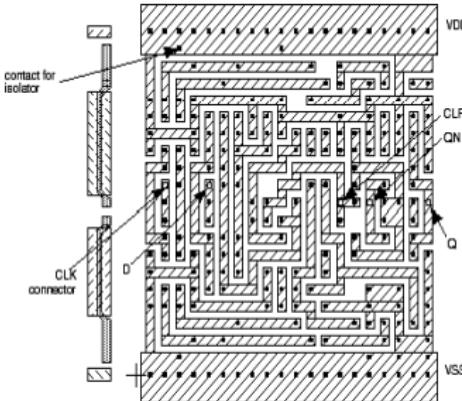
CIRCUITS INTÉGRÉS

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

Puce en partie réalisée

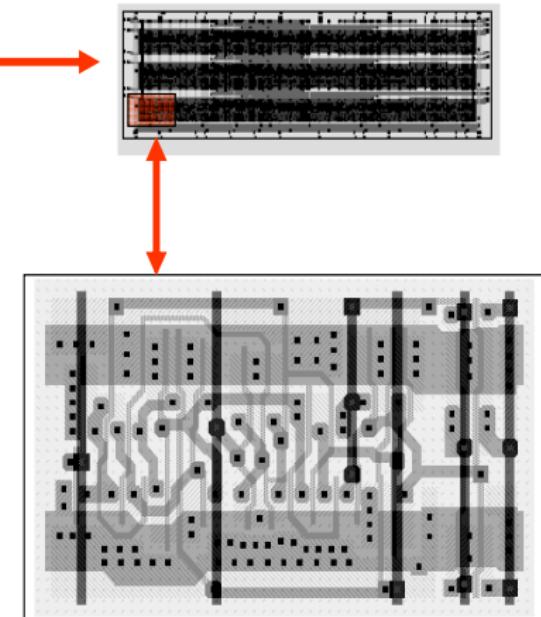
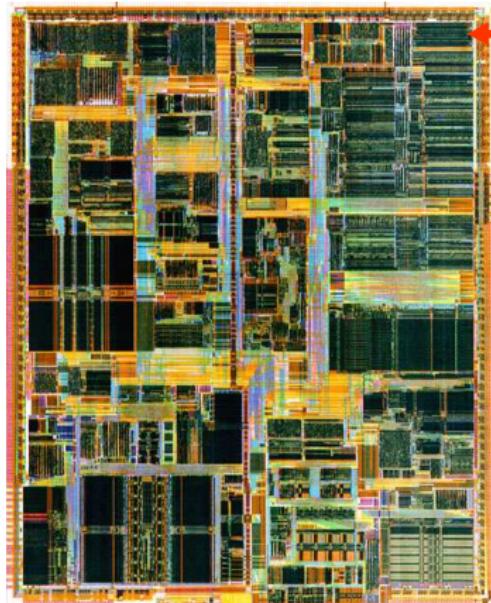


Connexions fournies
par le concepteur



CIRCUITS INTÉGRÉS

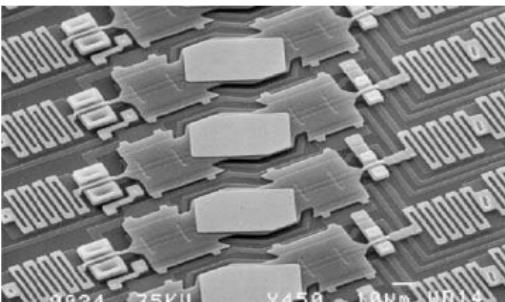
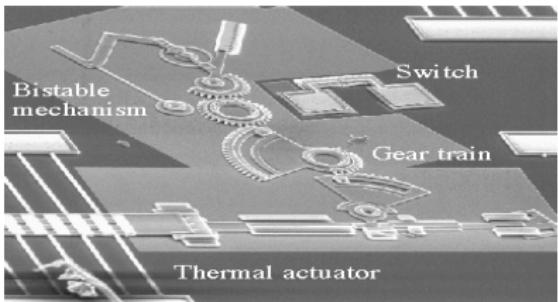
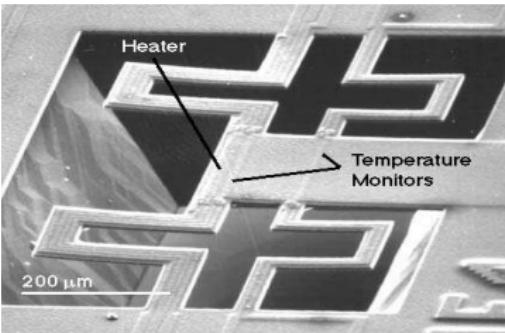
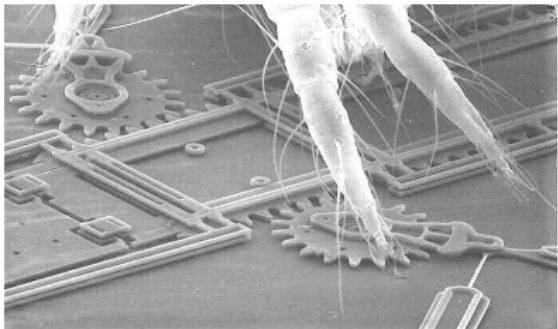
LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : *Standard Cells*



CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : MEMS

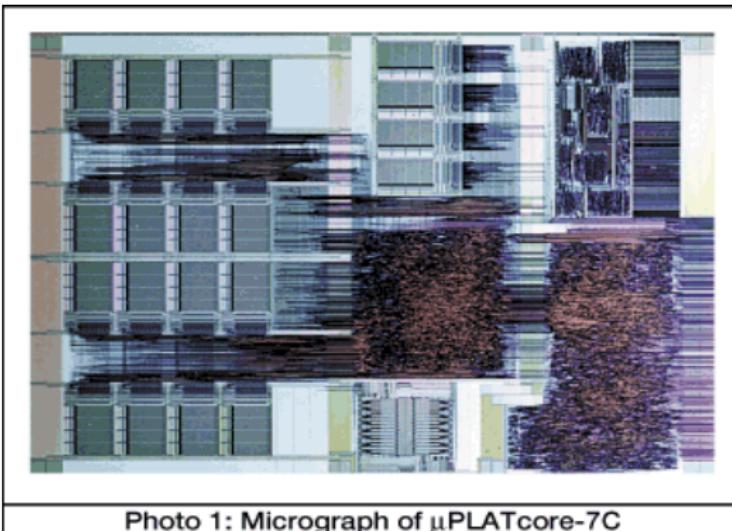
- MEMS : *Micro Electro-Mechanical Systems*



CIRCUITS INTÉGRÉS

LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : IP

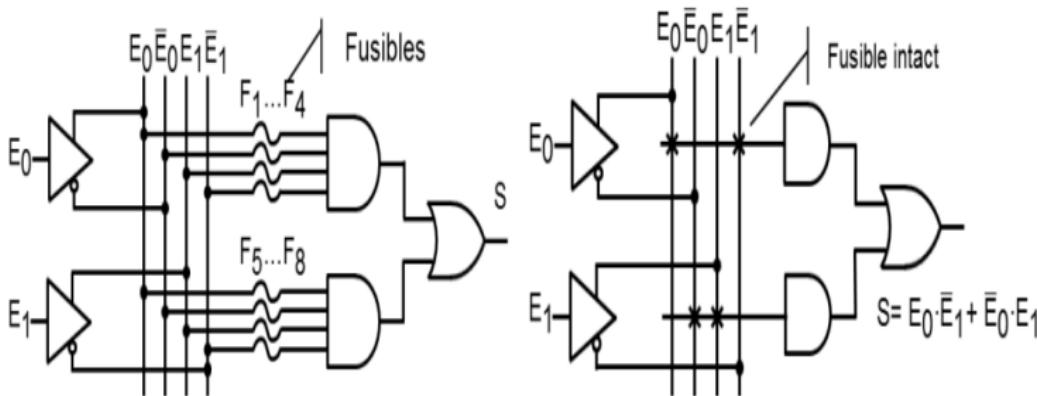
- IP : *Intellectual Property*



CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

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

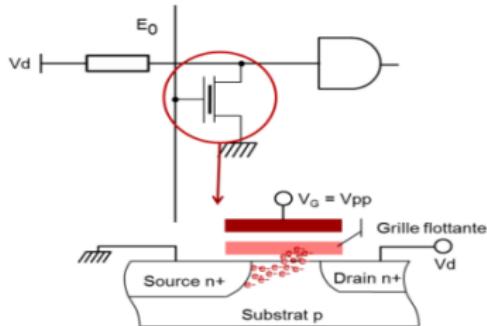


CIRCUITS INTÉGRÉS

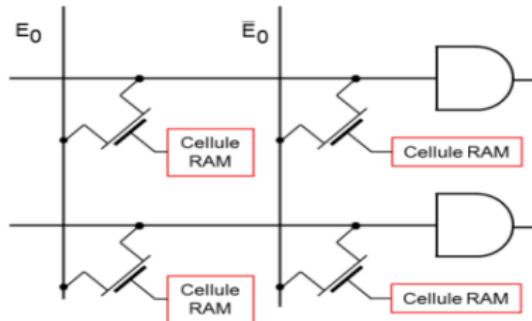
LES CIRCUITS PROGRAMMABLES

PLD et CPLD

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



a. PLD flash et transistor à grille isolée

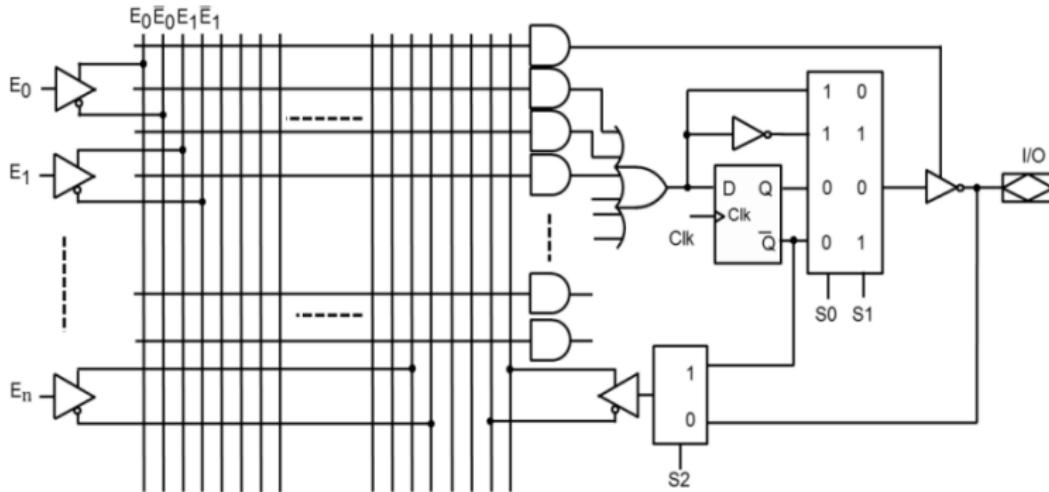


b. PLD SRAM

CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

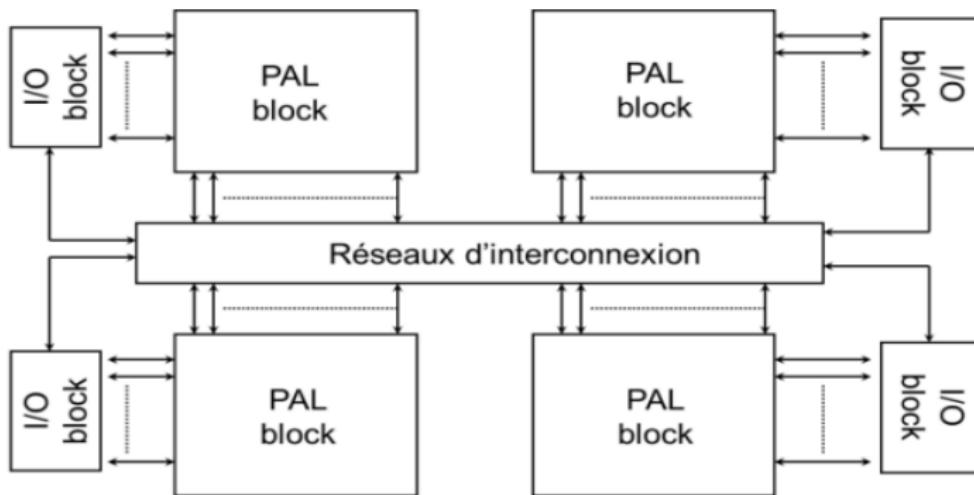
- Pour les circuits séquentielles, une bascule a été rajoutée à la sortie de chaque PAL



CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

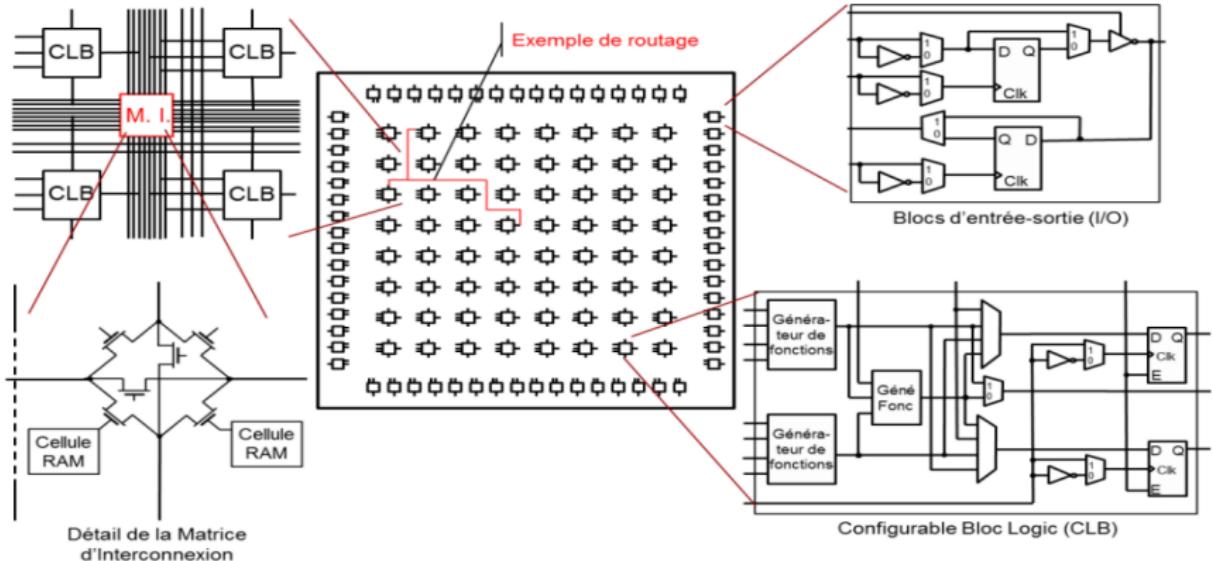
- CPLD - *Complex PLD* sont le regroupement de plusieurs PAL-registre avec un réseau programmable



CIRCUITS INTÉGRÉS

LES CIRCUITS PROGRAMMABLES

FPGA - *Field Programmable Gate Array*



SOMMAIRE

1 Introduction

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

CONCEPTION DE CI NUMÉRIQUES

POURQUOI LES CIRCUITS NUMÉRIQUES ?

Avantages des circuits numériques :

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

Applications

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

CONCEPTION DE CI NUMÉRIQUES

POURQUOI LES CIRCUITS NUMÉRIQUES ?

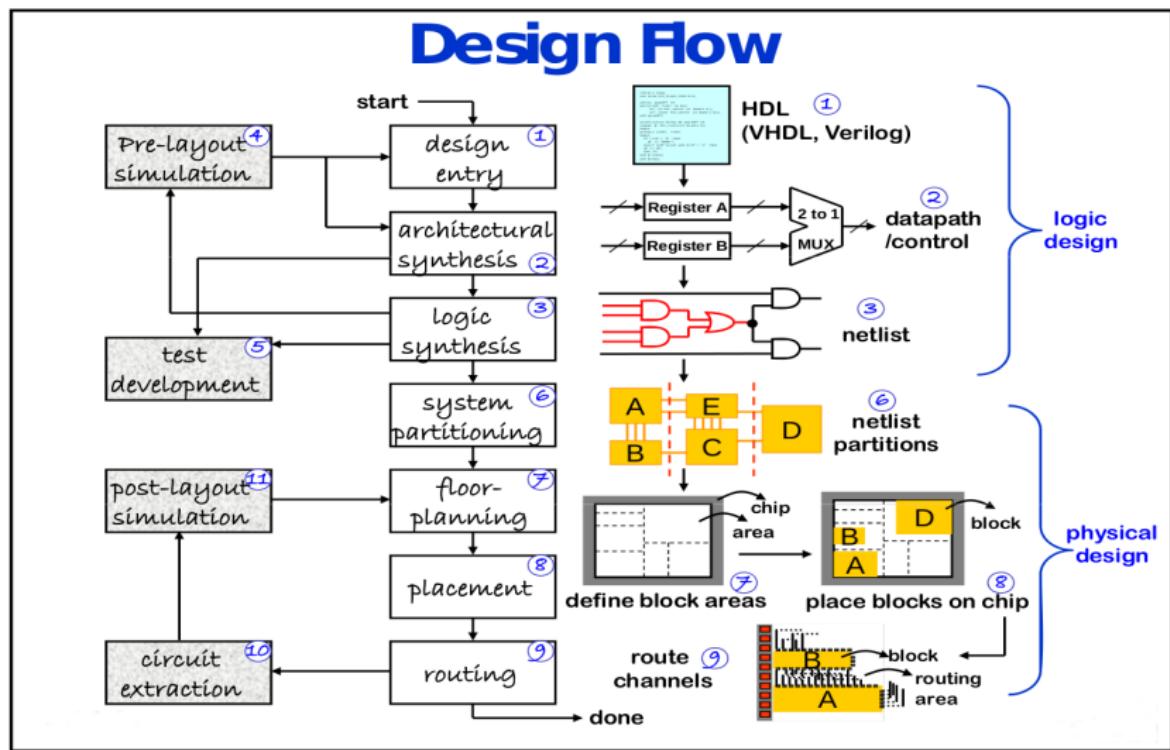
Les méthodes principales de "customisation"

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

Une solution adoptée est toujours :

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

FLOT DE CONCEPTION



FLOT DE CONCEPTION

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



VUES D'UN SYSTÈME

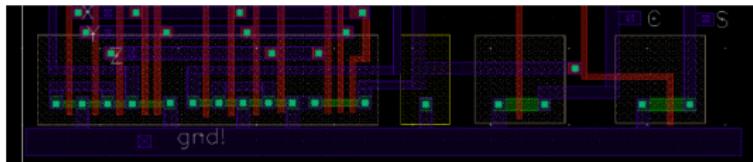
MODÈLES DE DESCRIPTION

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

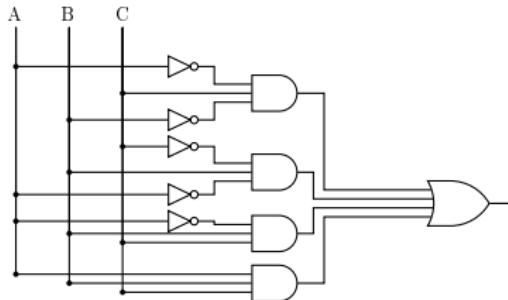
- Comportemental (*behavioral*)
 - ▷ Décrit les fonctionnalités d'un système et ses entrées/sorties (comportement d'un système)
 - ▷ Le système est considéré comme une boîte noire (le « quoi » et non le « comment »)
- Structurel
 - ▷ Décrit l'implémentation interne d'un système (les composants constitutifs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

VUES D'UN SYSTÈME

□ Physique 



□ Structuruel 

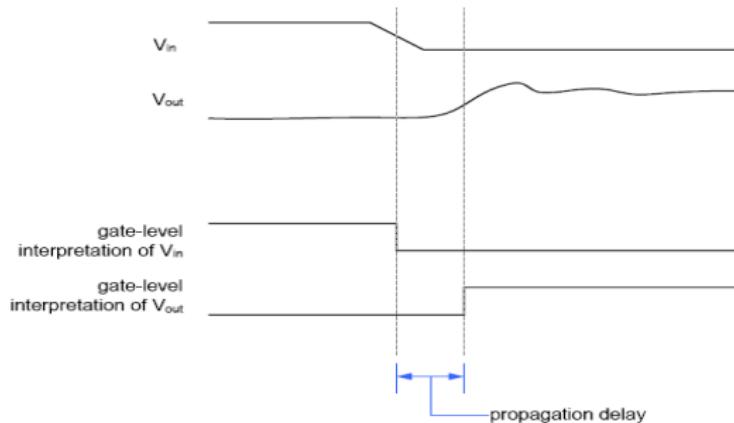


□ Comportemental 

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

NIVEAUX D'ABSTRACTION

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



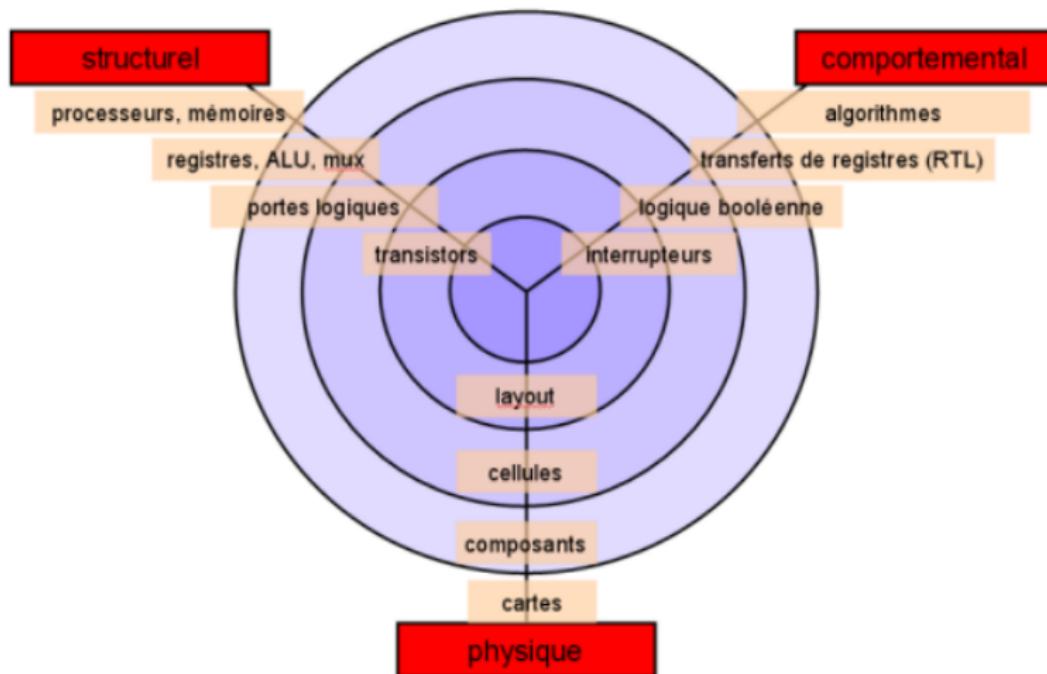
NIVEAUX D'ABSTRACTION

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

NIVEAUX D'ABSTRACTION

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

NIVEAUX D'ABSTRACTION



LES ÉTAPES DE DÉVELOPPEMENT

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

- Synthèse logique
- Synthèse physique
- Vérification et
- Test

LES ÉTAPES DE DÉVELOPPEMENT

SYNTHÈSE LOGIQUE

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

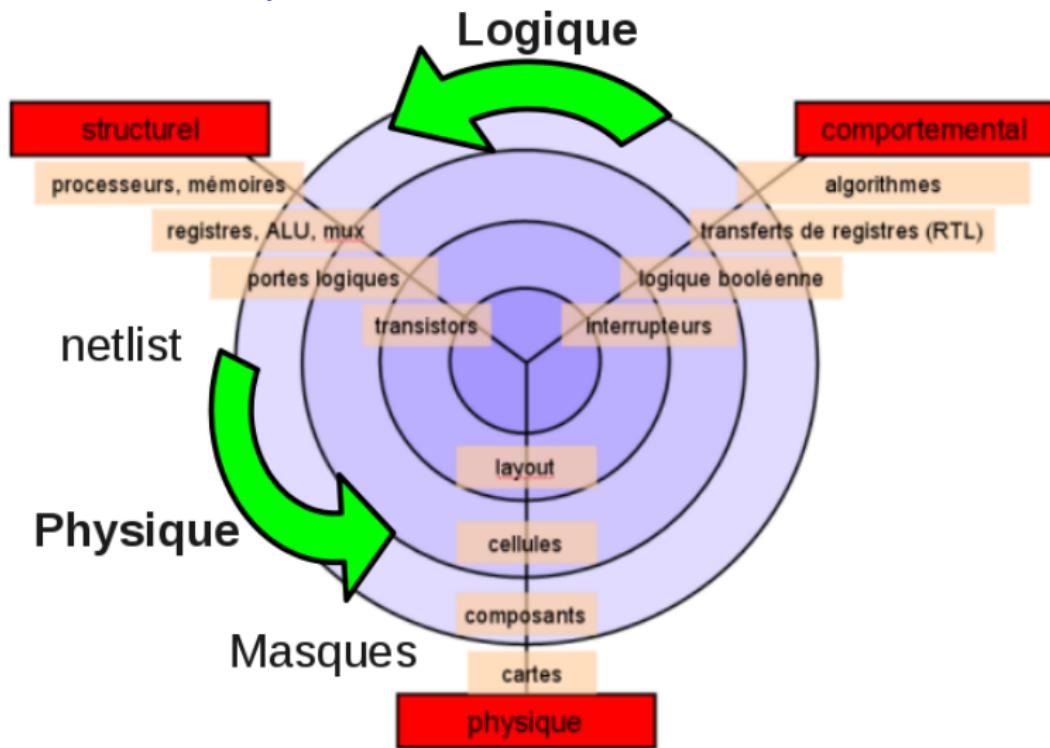
LES ÉTAPES DE DÉVELOPPEMENT

SYNTHÈSE PHYSIQUE

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

LES ÉTAPES DE DÉVELOPPEMENT

SYNTHÈSE PHYSIQUE



LES ÉTAPES DE DÉVELOPPEMENT

VÉRIFICATION

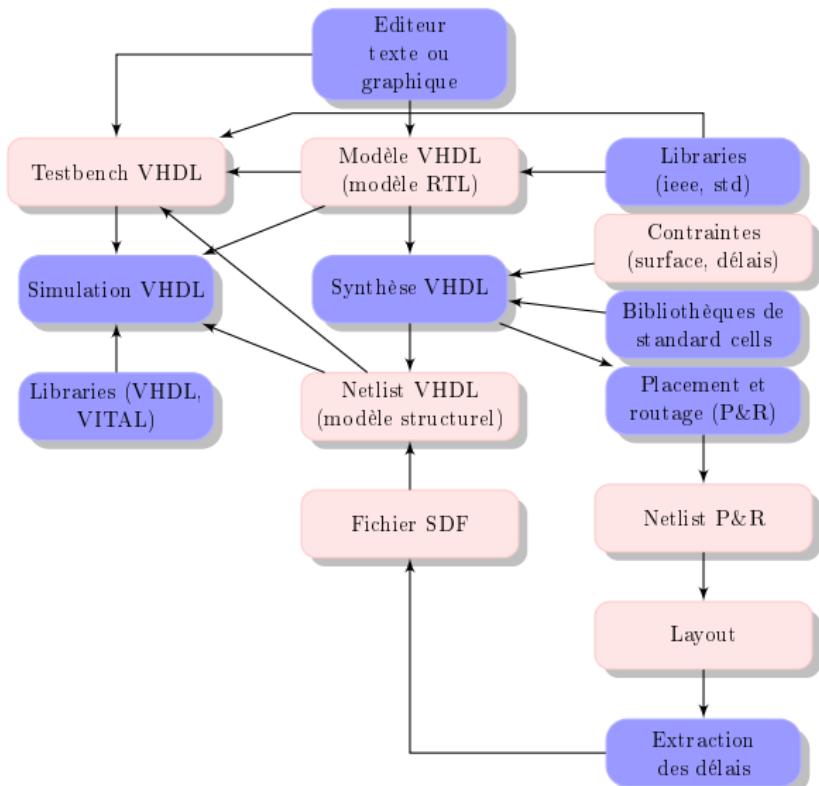
- Vérification
 - ▷ Vérifier si le système conçu respecte le cahier des charges initial
 - ▷ Deux aspects principaux à vérifier :
 - La fonctionnalité et
 - Les performances obtenues : timing (et parfois surface ou énergie)
 - ▷ La vérification est effectuée en appliquant des stimuli aux entrées du système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

LES ÉTAPES DE DÉVELOPPEMENT

FABRICATION ET TEST

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

FLOT DE CONCEPTION VHDL





SOMMAIRE

- 2 Hardware Description Language
 - HDL vs PL
 - HDL





HDL vs PL

LANGAGE DE DESCRIPTION DE MATÉRIEL VS LANGAGE DE PROGRAMMATION

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





SOMMAIRE

- 2 Hardware Description Language
 - HDL vs PL
 - HDL





LANGAGE DE DESCRIPTION DE MATÉRIEL

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





LE LANGAGE VHDL

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





EXTENSIONS IEEE

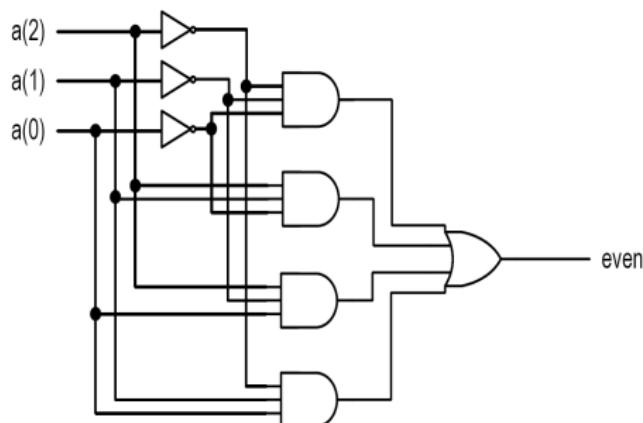
- IEEE standard 1076.1 Analog and Mixed Signal Extensions (VHDL-AMS)
- IEEE standard 1076.2 VHDL Mathematical Packages
- IEEE standard 1076.3 Synthesis Packages
- IEEE standard 1076.4 VHDL Initiative Towards ASIC Libraries (VITAL)
- IEEE standard 1076.6 VHDL Register Transfer Level (RTL) Synthesis
-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



VHDL

EXEMPLE : CIRCUIT DE DÉTECTION DE PARITÉ

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



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

| $a(2)$ | $a(1)$ | $a(0)$ | even |
|--------|--------|--------|------|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

VHDL

EXEMPLE : CIRCUIT DE DÉTECTION DE PARITÉ

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

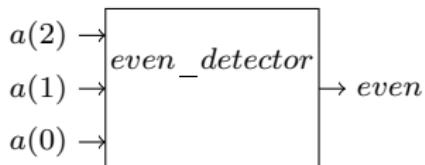
-- déclaration de l'entité
entity even_detector is
    port(
        a: in std_logic_vector(2 downto 0);
        even: out std_logic
    );
end even_detector;

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

VHDL

ENTITÉ

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



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

VHDL

L'ENTITÉ : EXEMPLES

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

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

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

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

VHDL

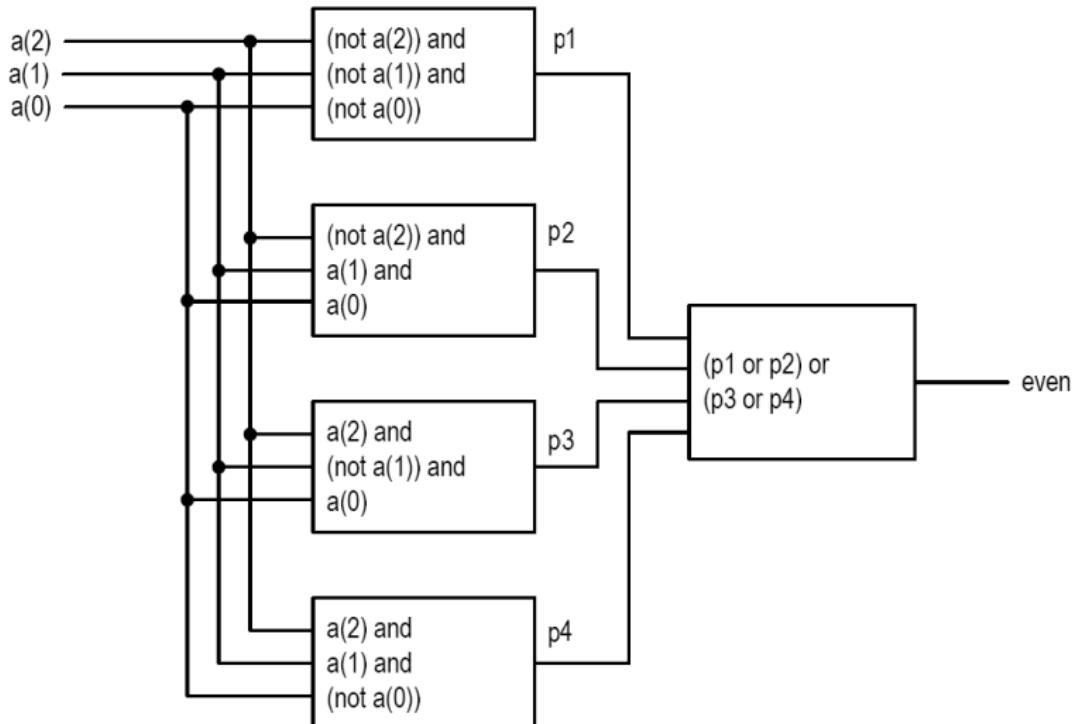
ARCHITECTURE

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

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

VHDL

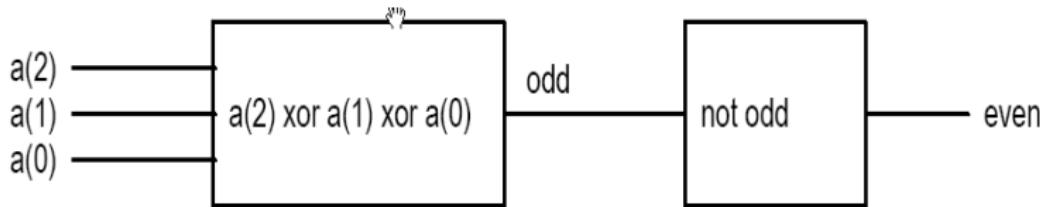
ARCHITECTURE : INTERPRÉTATION



VHDL

ARCHITECTURE

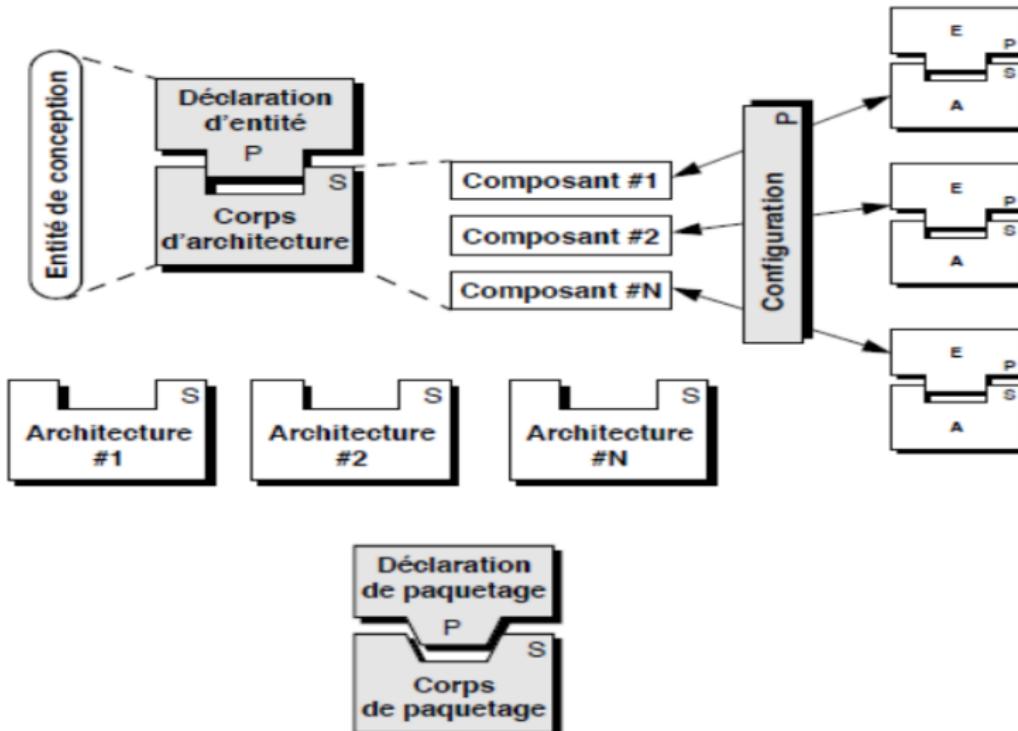
```
1 architecture xor_arch of even_detector is
2     signal odd: std_logic;
3 begin
4     even <= not odd;
5     odd <= a(2) xor a(1) xor a(0);
6 end xor_arch;
```



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

VHDL

UNITÉS DE CONCEPTION





VHDL

ARCHITECTURE

Deux types de descriptions :

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



VHDL

ARCHITECTURE

Syntaxe d'un processus :

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

Exemple d'utilisation :

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



VHDL

ARCHITECTURE

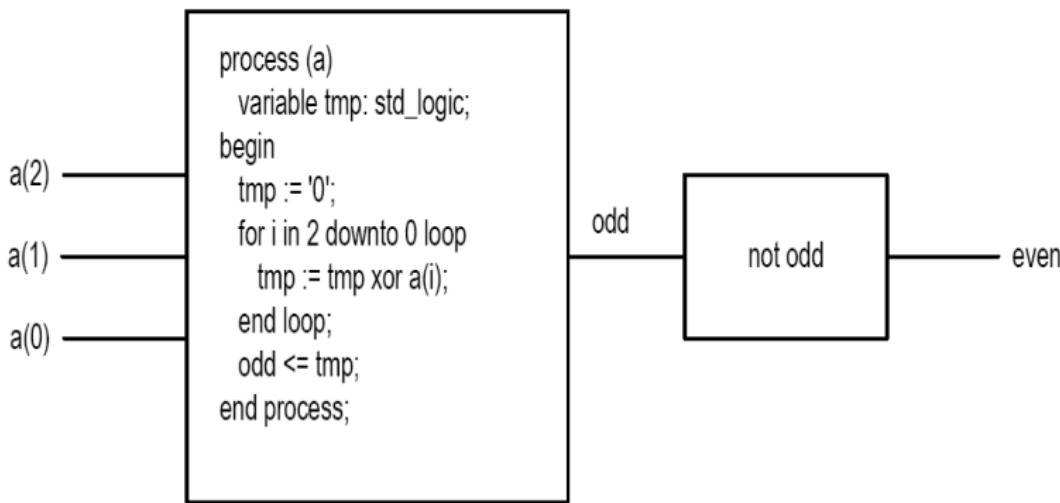
```
11      for i in 2 downto 0 loop
12          tmp := tmp xor a(i);
13      end loop;
14      odd <= tmp;
15  end process;
16 end beh1_arch;
```



VHDL

ARCHITECTURE

Représentation schématique d'un processus :



VHDL

ARCHITECTURE

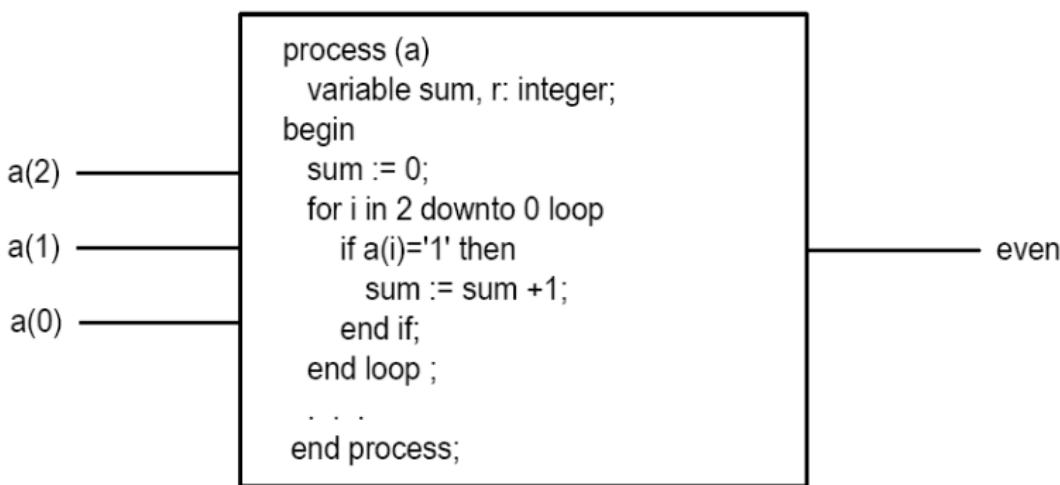
Entité comportementale :

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

VHDL

ARCHITECTURE

Représentation schématique d'une entité comportementale :



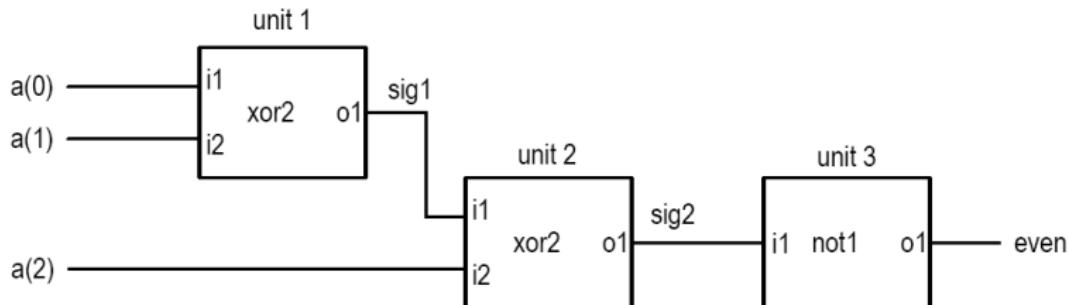
VHDL

ARCHITECTURE

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

VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE



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

VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : TOP-LEVEL

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



VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : TOP-LEVEL

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

VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : COMPOSANTS

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



VHDL

ARCHITECTURE : DESCRIPTION STRUCTURELLE : COMPOSANTS

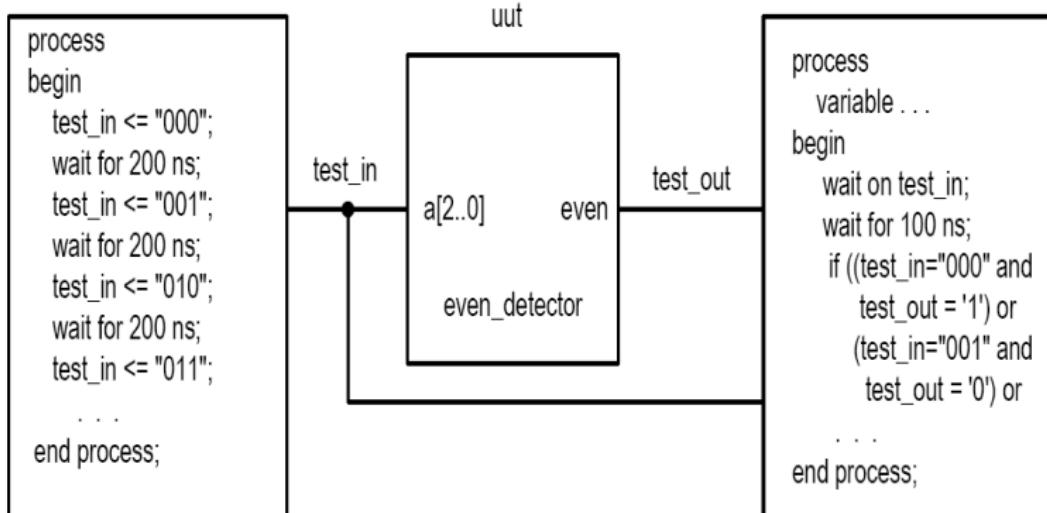
```
20      i1: in std_logic;
21      o1: out std_logic
22  );
23 end not1;
24 architecture beh_arch of not1 is
25 begin
26     o1 <= not i1;
27 end beh_arch;
```



VHDL

TESTBENCH

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



VHDL

TESTBENCH : EXEMPLE

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

VHDL

TESTBENCH : EXEMPLE

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

VHDL

TESTBENCH : EXEMPLE

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

VHDL

TESTBENCH : EXEMPLE

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



VHDL

CONFIGURATION

- Plusieurs architectures peuvent être associées à une entité
- Le rôle principal d'une **configuration** est de préciser cette association

```
configuration demo_config of even_detector_tb is
  for tb_arch
    for uut: even_detector
      use entity work.even_detector(sop_archi);
    end for;
  end for;
end demo_config;
```

- Une configuration directe dans le corps de l'architecture (en précisant la **library.composant**)



SOMMAIRE

- 3 Utilisation de Quartus
 - Création du projet Quartus
 - Utilisation du flot de conception





CRÉATION DU PROJET QUARTUS

UTILISATION DU PROJECT WIZARD

- page 1 : Donner un nom au projet et choisir un dossier
- page 2 : Ne rien changer
- page 3 : Spécifions le FPGA de la carte de développement :
 - ▷ Family Cyclone V
 - ▷ Devices Cyclone V SE Mainstream
 - ▷ Target device ⇒ Specific device
 - ▷ Sélectionner 5CSEMA5F31C6
- page 4 : Ne rien changer
- page 5 : Cliquer sur Finish

Family & Device Settings [page 3 of 5]

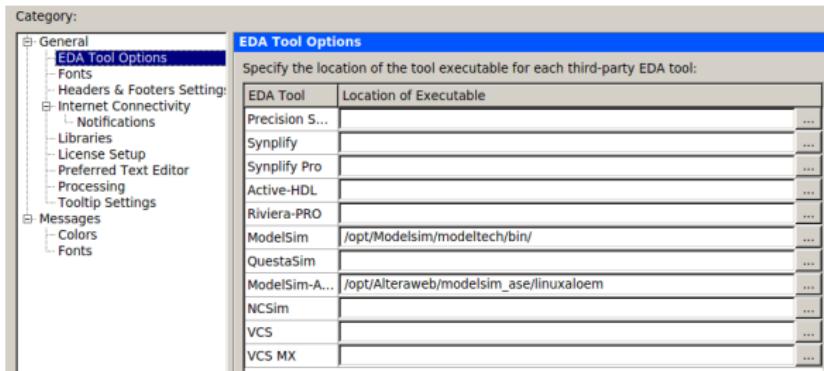
Select the family and device you want to target for compilation.
You can install additional device support with the Install Devices command on the Tools menu.

| Device family | Show in 'Available devices' list | | | |
|---|---|--------------|------------|-----------------|
| Family: Cyclone V (E/GX/GT/SX/SE/ST) | Package: Any | | | |
| Devices: Cyclone V SE Mainstream | Pin count: Any | | | |
| Target device | | | | |
| <input type="radio"/> Auto device selected by the Filter | Speed grade: Any | | | |
| <input checked="" type="radio"/> Specific device selected in 'Available devices' list | Name filter: | | | |
| <input type="radio"/> Other: n/a | <input checked="" type="checkbox"/> Show advanced devices | | | |
| Available devices: | | | | |
| Name | Core Voltage | ALMs | User I/Os | GXB Channel PMs |
| 5CSEMA4U23I7 (Advanced) | 1.1V | 15880 | 314 | 0 |
| 5CSEMA5F31A7 | 1.1V | 32070 | 457 | 0 |
| 5CSEMA5F31C6 | 1.1V | 32070 | 457 | 0 |

CONFIGURATION DE QUARTUS

CHOIX DE LA VERSION DE MODELSIM

- Si l'on ne possède pas de licence pour modelsim :
- Menu **tools** ⇒ **Options** ⇒ pointer vers la version Altera Starter Edition

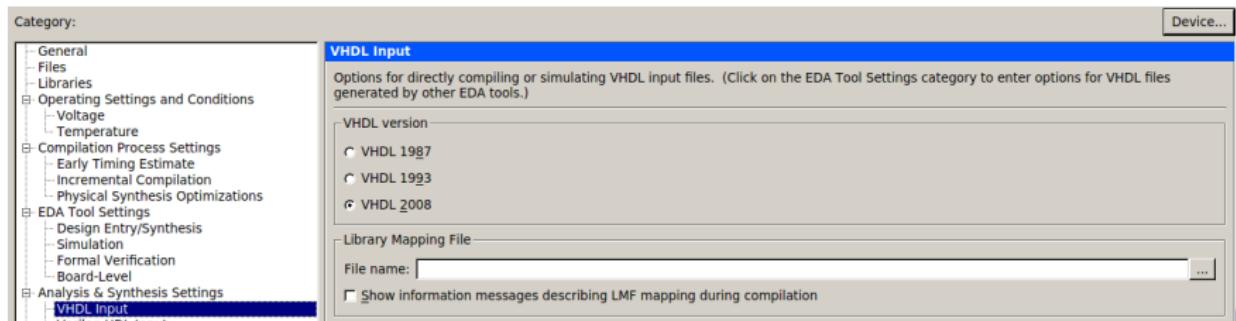




CONFIGURATION DE QUARTUS

CHOIX DE LA VERSION DE VHDL

- Pour bénéficier des facilités offertes par VHDL-2008
 - ▷ Clic droit sur device dans Hierarchy ⇒ Settings
 - ▷ VHDL input ⇒ cocher VHDL-2008
 - ▷ Ok





EDITION DU CODE VHDL

UTILISATION DES MODÈLES

- Quartus offre de nombreux modèles de codage VHDL
 - ▷ Edit ⇒ Insert Template
 - ▷ Choisir le modèle souhaité
 - ▷ Insert

```
-- Quartus II VHDL Template
-- Single port RAM with single read/write address

library ieee;
use ieee.std_logic_1164.all;

entity single_port_ram is
    generic
    (
        DATA_WIDTH : natural := 8;
        ADDR_WIDTH : natural := 6
    );
    port
    (
        clk      : in std_logic;
        addr    : in natural range 0 to 2**ADDR_WIDTH - 1;
        data   : in std_logic_vector((DATA_WIDTH-1) downto 0);
        we     : in std_logic := '1';
        q      : out std_logic_vector((DATA_WIDTH -1) downto 0)
    );
end entity;

architecture rtl of single_port_ram is
    -- Build a 2-D array type for the RAM

```

Save Insert Close



SOMMAIRE

3 Utilisation de Quartus

- Création du projet Quartus
- Utilisation du flot de conception



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF CODE VHDL

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

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

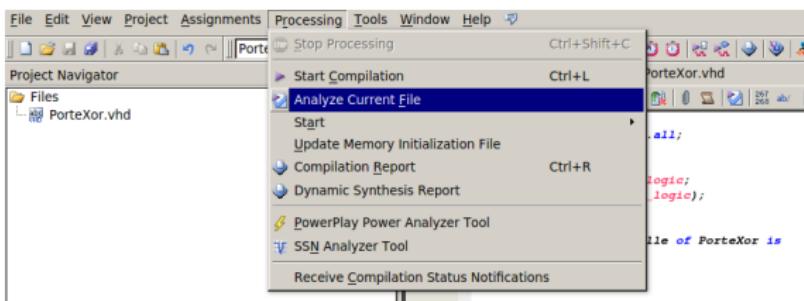
architecture fonctionnelle of PorteXor is
begin
    S <= A xor B;
end fonctionnelle;
```



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

FLOT DE CONCEPTION

- Vérifier que le code ne comporte pas d'erreurs de syntaxe
 - ▷ Processing ⇒ Analyze current file



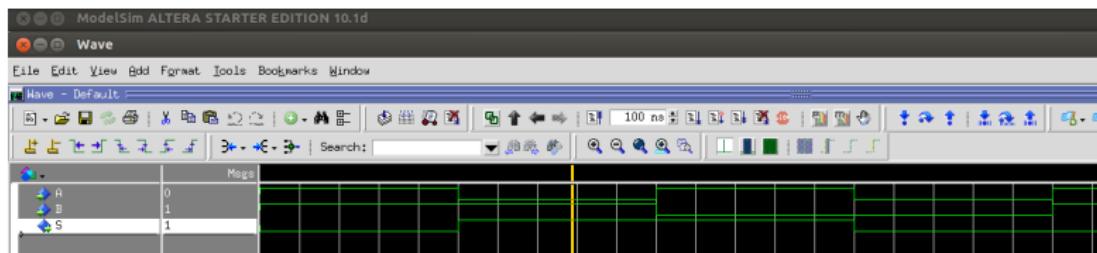
- Pas d'erreurs ⇒ Compiler (Ctrl-L) puis démarrer le simulateur
 - ▷ Tools ⇒ Run Simulation Tool ⇒ RTL Simulation



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

MODELSIM

- Sélectionner l'entité **portexor** de la bibliothèque **work** (double clic)
- Dans la fenêtre **Objects** sélectionner les signaux intéressants
- Les ajouter aux chronogrammes (clic droit \Rightarrow **add wave**)
- Les modifier (clic droit \Rightarrow **modify** \Rightarrow **apply clock** ou **apply wave**)
- Démarrer la simulation : **run 300 ns**





EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

MODELSIM

- Pour ne pas devoir refaire la configuration graphique de modelsim à chaque itération, enregistrer la configuration :
 - ▷ Soit l'ensemble par la commande : `write transcript msimxor.do`
 - ▷ Soit la configuration de la fenêtre wave uniquement : `file save format` qui créera un fichier `wave.do`
- Lors d'une nouvelle ouverture de modelsim taper l'une des commandes
 - ▷ `do msimxor.do`
 - ▷ `do wave.do`
- Démarrer la simulation : `run 300 ns`
- En pratique on préfère utiliser un `testbench` écrit directement en VHDL



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

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

entity PorteXor_tb is
end PorteXor_tb;

architecture tb of PorteXor_tb is
    --passage de l'entité PorteXor au testbench comme
    --composant
    component PorteXor is
        port( A, B : in std_logic;
              S : out std_logic);
    end component;
```



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

```
    signal  inA, inB, outS : std_logic;
begin
    --relier les signaux du testbench aux ports de
    PorteXor
    mapping: PorteXor port map(inA, inB, outS);

process
    --variable pour les erreurs
    variable errCnt : integer := 0;
begin
    --TEST 1
    inA <= '0';
    inB <= '0';
    wait for 15 ns;
```





EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

```
assert(outS = '0') report "Error 1" severity
error;
if(outS /= '0') then
    errCnt := errCnt + 1;
end if;

--TEST 2
inA <= '0';
inB <= '1';
wait for 15 ns;
assert(outS = '1') report "Error 2" severity
error;
if(outS /= '1') then
    errCnt := errCnt + 1;
```





EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

```
end if;

--TEST 3
inA <= '1';
inB <= '1';
wait for 15 ns;
assert(outS = '0') report "Error 3" severity
error;
if(outS /= '0') then
  errCnt := errCnt + 1;
end if;

----- RESUME -----
if(errCnt = 0) then
```





EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

```
        assert false report "OK!"    severity note;
else
    assert true report "KO!"     severity error;
end if;

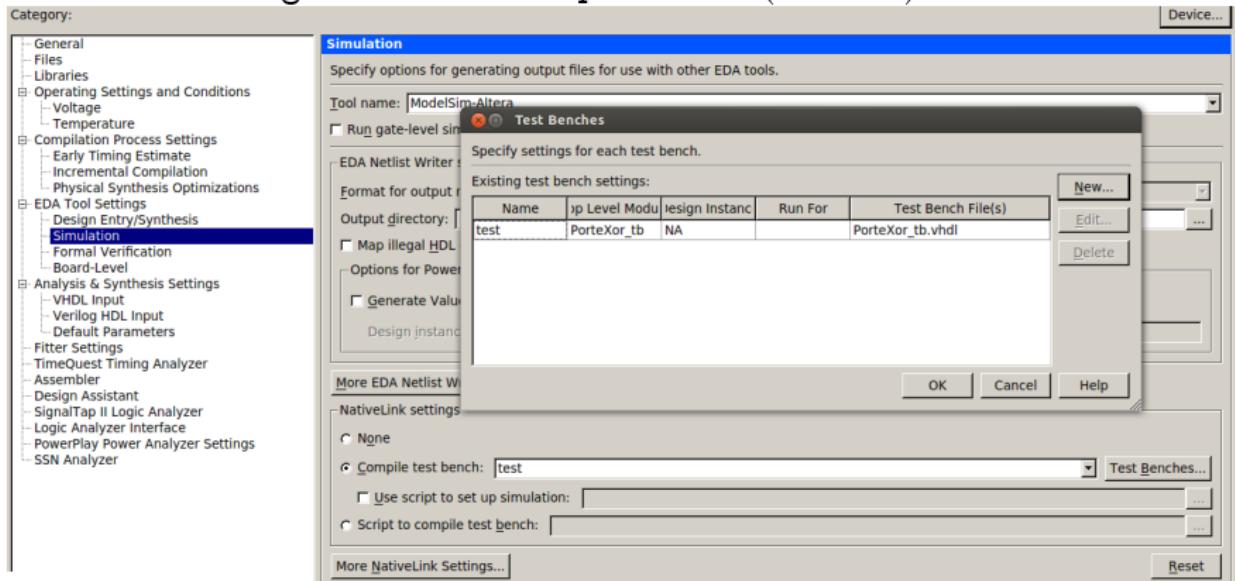
end process;
end tb;
```



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

- Spécification du fichier testbench dans les settings du projet Quartus
- Puis compilation de l'ensemble système + testbench :
Processing ⇒ Start compilation (Ctrl-L)



EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

TESTBENCH

- Quartus peut aussi générer un modèle de testbench :
Processing ⇒ Start ⇒ Start Testbench Template Writer
- Cela crée un fichier .vht (dans le dossier modelsim) qu'il faut compléter puis spécifier en tant que **testbench** comme vu précédemment
- Dans tous les cas on doit obtenir un résultat de simulation du type suivant :

```
run 100 ns
# ** Note: OK!
#      Time: 45 ns  Iteration: 0  Instance: /portexor_tb
# ** Note: OK!
#      Time: 90 ns  Iteration: 0  Instance: /portexor_tb
```



SOMMAIRE

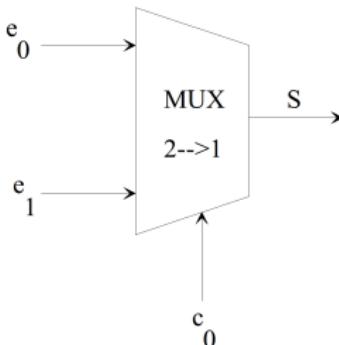
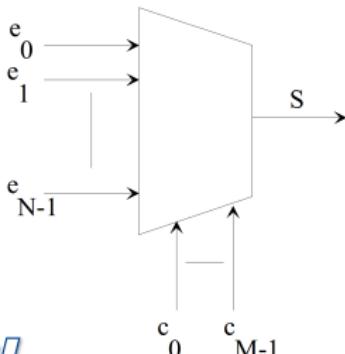
- 4 VHDL : utilisation du langage
 - Description des fonctions combinatoires usuelles
 - Types de données



LE MULTIPLEXEUR

FONCTION ET SPÉCIFICATIONS

- Choix «d'affectation» en fonction de la valeur d'un sélecteur (mot binaire)
- Équivaut à structure conditionnelle (ou choix multiples)
- Sortie = valeur (entrée dont le numéro est le sélecteur)
- $\{entree_0 ; entree_1 ; \dots ; entree_{N-1}\}$: entiers d'entrée
- sélecteur : entier (nb bits = Sup[$\log_2(N)$])
- Sortie = $entree_{selecteur}$



$$S = \bar{c}_0 \cdot e_0 + c_0 \cdot e_1$$



LE MULTIPLEXEUR

DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity Multiplexeur is
port(
    IN0,IN1,IN2,IN3: in  std_logic_vector(7 downto
0);
    Selecteur         : in  std_logic_vector(1 downto
0);
    Sortie           : out std_logic_vector(7 downto
0));
end Multiplexeur;

architecture ConcSelect of Multiplexeur is
begin
```





LE MULTIPLEXEUR

DESCRIPTION VHDL

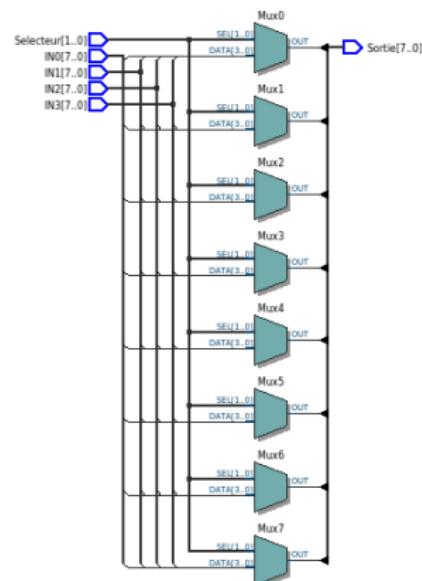
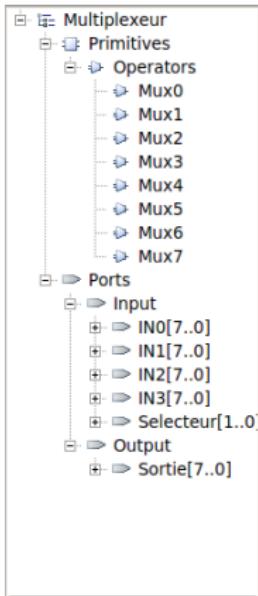
```
with Selecteur select
Sortie <= IN0 when "00",
    IN1 when "01",
    IN2 when "10",
    IN3 when "11",
    IN0 when others;
end ConcSelect;
```

- le cas **Others** n'est pas nécessaire ici pour la synthèse

LE MULTIPLEXEUR

VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)

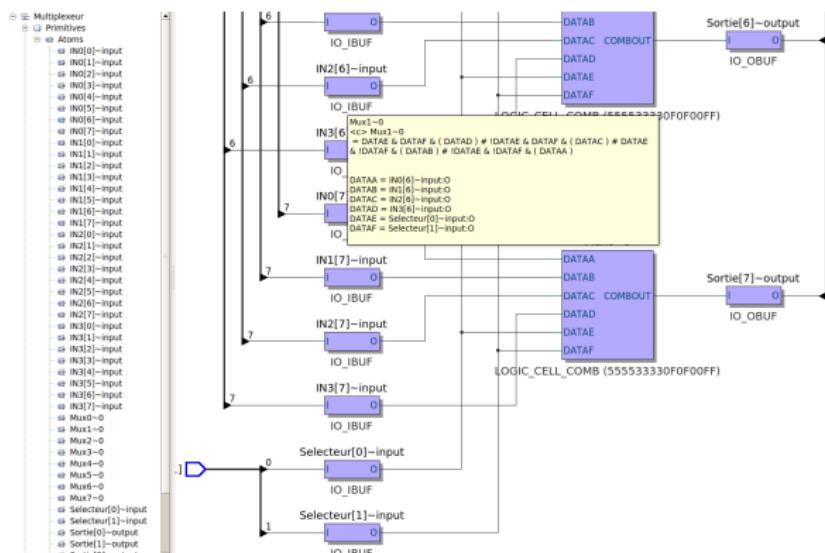
- Après compilation :
Processing \Rightarrow
Start compilation
(Ctrl-L)
- Affichage de la vue
RTL : Tools \Rightarrow
Netlist Viewers
 \Rightarrow RTL viewer
- On note l'utilisation
d'une primitive
multiplexeur 1 bit



LE MULTIPLEXEUR

VUE DU NIVEAU CELLULES FPGA

- Généralement pas utilisé, ici but pédagogique
- Affichage de la vue FPGA : Tools ⇒ Netlist Viewers ⇒ Technology Map Viewer (Post-Mapping)



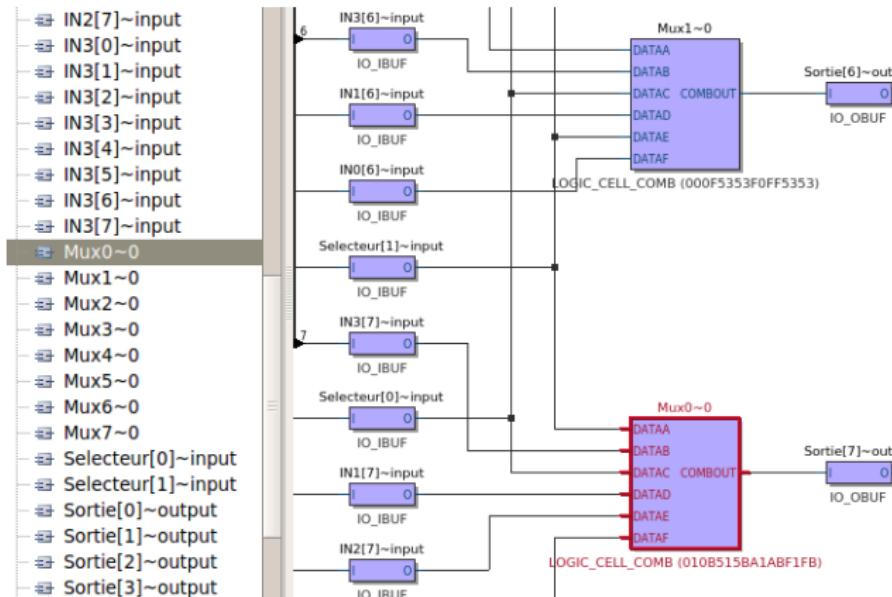
LE MULTIPLEXEUR

VUE DU NIVEAU CELLULES FPGA

- Généralement pas utilisé, ici but pédagogique
- Affichage de la vue FPGA : Tools ⇒ Netlist Viewers ⇒ Technology Map Viewer (Post-Fitting)
- On note le placement différent des entrées sur chaque cellule (différence avec Post-Mapping)

```

    ┌─────────────────────────────────────────────────────────────────────────┐
    | IN2[7]~input          | IN3[0]~input          | IN3[1]~input          |
    | IN3[2]~input          | IN3[3]~input          | IN3[4]~input          |
    | IN3[5]~input          | IN3[6]~input          | IN3[7]~input          |
    | Mux0~0                | Selecteur[1]~input   | Selecteur[0]~input   |
    | Mux1~0                | IN3[7]~input          | IN1[7]~input          |
    | Mux2~0                | IN2[7]~input          | IN2[7]~input          |
    | Mux3~0                |                         |                         |
    | Mux4~0                |                         |                         |
    | Mux5~0                |                         |                         |
    | Mux6~0                |                         |                         |
    | Mux7~0                |                         |                         |
    | Selecteur[0]~input   |                         |                         |
    | Sortie[0]~output      |                         |                         |
    | Sortie[1]~output      |                         |                         |
    | Sortie[2]~output      |                         |                         |
    | Sortie[3]~output      |                         |                         |
    └─────────────────────────────────────────────────────────────────┘
  
```





LE MULTIPLEXEUR

INSTRUCTION case

```
library ieee;
use ieee.std_logic_1164.all;
entity Multiplexeur is port(
    IN0,IN1,IN2,IN3: in std_logic_vector(7 downto 0);
    Selecteur :         in std_logic_vector(1 downto 0);
    Sortie :           out std_logic_vector(7 downto
0));
end Multiplexeur;
architecture ArchCase of Multiplexeur is begin
process(Selecteur,IN0,IN1,IN2,IN3)
--process(all) -VHDL 2008
begin
    case Selecteur is
```





LE MULTIPLEXEUR

INSTRUCTION case

```
when "00" => Sortie <= IN0;
when "01" => Sortie <= IN1;
when "10" => Sortie <= IN2;
when "11" => Sortie <= IN3;
when others => Sortie <= (others => '0');
end case;
end process;
end ArchCase;
```



LE MULTIPLEXEUR

INSTRUCTION if-end if

```
library ieee;
use ieee.std_logic_1164.all;
entity Multiplexeur is port(
    IN0,IN1,IN2,IN3: in std_logic_vector(7 downto 0);
    Selecteur :      in std_logic_vector(1 downto 0);
    Sortie :         out std_logic_vector(7 downto
0));
end Multiplexeur;
architecture ArchIf of Multiplexeur is begin
process(Selecteur,IN0,IN1,IN2,IN3)
--process(all) -VHDL 2008
begin
    if Selecteur = "00" then
```



LE MULTIPLEXEUR

INSTRUCTION if-end if

```
Sortie <= IN0;  
elsif Selecteur = "01" then  
    Sortie <= IN1;  
elsif Selecteur = "10" then  
    Sortie <= IN2;  
elsif Selecteur = "11" then  
    Sortie <= IN3;  
else  
    Sortie <= (others => '0');  
end if;  
end process;  
end ArchIf;
```

- Recompile le design





LE MULTIPLEXEUR

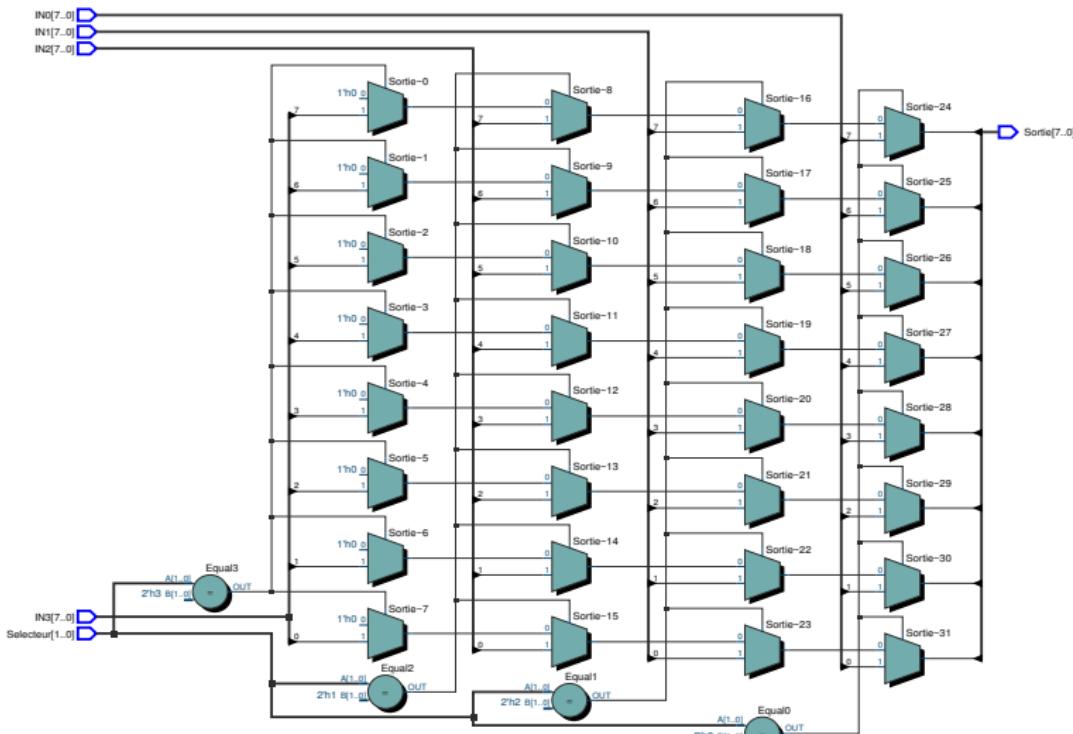
INSTRUCTION if-end if

- Visualiser la netlist RTL
- Quelle est la différence par rapport à l'architecture utilisant `with-select` ou `case` ?



LE MULTIPLEXEUR

VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)





RAPPEL

ENTITÉ

```
entity nom is
    -- les ports d'entrées/sorties
    port (in1 : in std_logic;
          in2 : in std_logic;
          -- port_nom: sens(in/out/inout) type;
          ...
          out1 : out std_logic;
          out2 : out std_logic;
          ...
          outn : out std_logic
        );
end;
-- end entity;
-- end nom;
```



RAPPEL

ARCHITECTURE

```
architecture arch_nom of nom is
    -- partie déclarative de l'architecture
    signal sig1, sig2, ..., sign : std_logic;
    -- signal nom : type;

    -- utilisation de composants
    component comp_nom
        ...
    end component;

    begin
        ...
    end;
    -- end nom;
```



RAPPEL

INSTRUCTIONS

- concurrentes
 - en dehors d'un process
- séquentielles
 - à l'intérieur d'un process

```
architecture arch_nom of nom is
    signal sig1, sig2: std_logic;
begin
    -- instruction concurrente
    sig1 <= a xor b;
    --instruction séquentielle
    process(a,b) --process(all)
begin
    sig2 <= a xor b;
end process;
end;
```

RAPPEL

INSTRUCTIONS

concurrent

```
architecture arch_nom of nom is
begin
with sel select
    sortie <= in1 when "00",
                in2 when "01",
                in3 when "10",
                ...
                in1 when others;
end;
```

séquentiel

```
architecture arch_nom of nom is
begin
    process(all)
    begin
        case sel is
            when "00" => sortie <= in1;
            when "01" => sortie <= in2;
            when "10" => sortie <= in3;
            ...
            when others => sortie <= in1;
        end case;
    end process;
end;
```



RAPPEL

INSTRUCTIONS

concurrent

```
architecture arch_nom of nom is
begin
    sortie <= in1 when sig1 = '0'
        else in2;
end;
```

séquentiel

```
architecture arch_nom of nom is
begin
    process(all)
    begin
        if sig1='0' then
            sortie <= in1;
        else
            sortie <= in2;
        end if;
    end process;
end;
```





PARAMÈTRES GÉNÉRIQUES

```
entity nom is
    -- paramètres génériques
    generic (param1: integer:=10;
              param2: integer:=10;
              ...
              paramn: integer:=10
            );
    -- les ports d'entrées/sorties
    port (in1 : in std_logic_vector(param1-1 downto 0);
          out1 : out std_logic_vector(param2-1 downto 0);
          ...
          outn : out std_logic
            );
end;
```





LE MULTIPLEXEUR

EXEMPLE DE DESCRIPTION GÉNÉRIQUE

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

entity MuxGenerique is
generic ( TAILLE : integer := 8 );
port (
    in0, in1, in2, in3 : in
    std_logic_vector(TAILLE-1 downto 0);
    Selecteur : in std_logic_vector(1 downto 0);
    Sortie : out std_logic_vector(TAILLE-1 downto
    0));
end MuxGenerique;
```





LE MULTIPLEXEUR

EXEMPLE DE DESCRIPTION GÉNÉRIQUE

```
architecture CaseProcess of MuxGenerique is
begin
process(all)
begin
case Selecteur is
when "00" => Sortie <= in0;
when "01" => Sortie <= in1;
when "10" => Sortie <= in2;
when others => Sortie <= in3;
end case;
end process;
end CaseProcess;
```





LE MULTIPLEXEUR

EXEMPLE DE DESCRIPTION GÉNÉRIQUE

Création d'un paquetage pour déclarer une constante :

```
package Config is
    constant TAILLEDATA : integer;
end package Config;
package body Config is
    constant TAILLEDATA: integer := 16;
end package body Config;
```

Il est aussi possible de déclarer une constante dans la partie déclarative du fichier, dans l'entité ou dans un **process** selon la visibilité souhaitée.

Instanciation et paramétrage du multiplexeur générique :





LE MULTIPLEXEUR

EXEMPLE DE DESCRIPTION GÉNÉRIQUE

```
library work;
-- appel du package Config
use work.Config.all;
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Multiplexeur16bits is port (
    ina, inb, inc, ind : in std_logic_vector(TAILLEDATA-1 downto 0);
    Sel : in std_logic_vector(1 downto 0);
    S : out std_logic_vector(TAILLEDATA-1 downto 0));
end Multiplexeur16bits;

architecture instance of Multiplexeur16bits is
-- partie déclarative de l'archi
component MuxGenerique is
generic ( TAILLE : integer := 8 );
port ( in0, in1, in2, in3 : in std_logic_vector(TAILLE-1 downto 0);
       Selecteur : in std_logic_vector(1 downto 0);
       Sortie : out std_logic_vector(TAILLE-1 downto 0));
```





LE MULTIPLEXEUR

EXEMPLE DE DESCRIPTION GÉNÉRIQUE

```
end component;

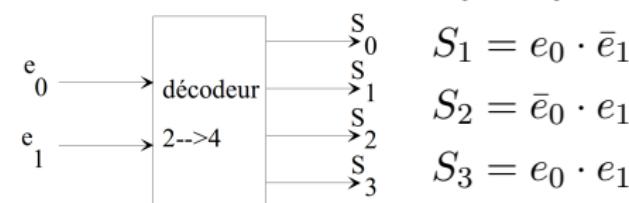
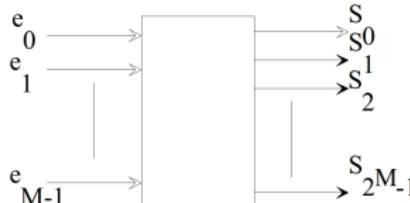
begin
-- instanciation du composant
MonMux: MuxGenerique
generic map(TAILLE=>TAILLEDATA)
port map(in0=>ina,in1=>inb,in2=>inc,in3=>ind,
         Selecteur=>Sel,Sortie=>S);
end instance;
```



LE DÉCODEUR

FONCTION ET SPÉCIFICATIONS

- Optimisation d'un cas particulier du multiplexeur
- Très utilisé pour l'adressage
- Sortie active = sortie dont le numéro correspond à la valeur d'entrée
- *entree* : entier d'entrée sur N bits
- $\{Sortie_0; Sortie_1; \dots; Sortie_{2^N-1}\}$: sorties sur 1 bit
- Sortie active = $Sortie_{entree}$ (\forall autres inactives)



$$S_0 = \bar{e}_0 \cdot \bar{e}_1$$

$$S_1 = e_0 \cdot \bar{e}_1$$

$$S_2 = \bar{e}_0 \cdot e_1$$

$$S_3 = e_0 \cdot e_1$$



LE DÉCODEUR

DESCRIPTION VHDL

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

entity Decodeur is
port(
    Entree:  in  std_logic_vector(1 downto 0);
    Sortie : out std_logic_vector(3 downto 0));
end Decodeur;

architecture ConcSelect of Decodeur is
begin
with Entree select
Sortie <= "0001" when "00",
    "0010" when "01",
```





LE DÉCODEUR

DESCRIPTION VHDL

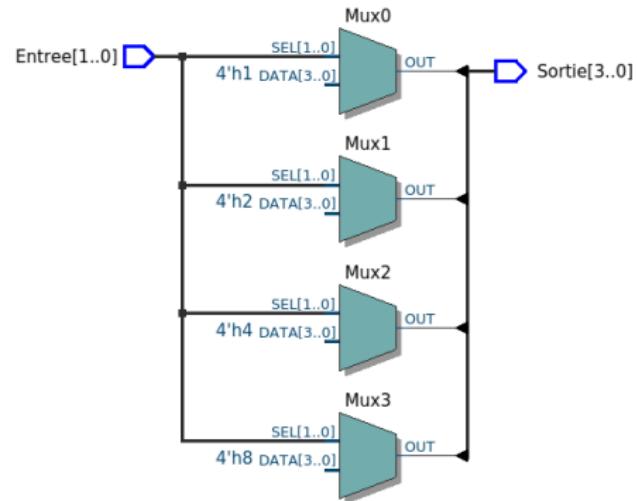
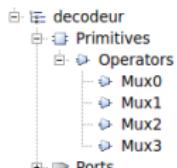
```
"0100" when "10",
"1000" when "11",
"0000" when others;
end ConcSelect;
```

- le cas **Others** n'est pas nécessaire ici pour la synthèse

LE DÉCODEUR

VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)

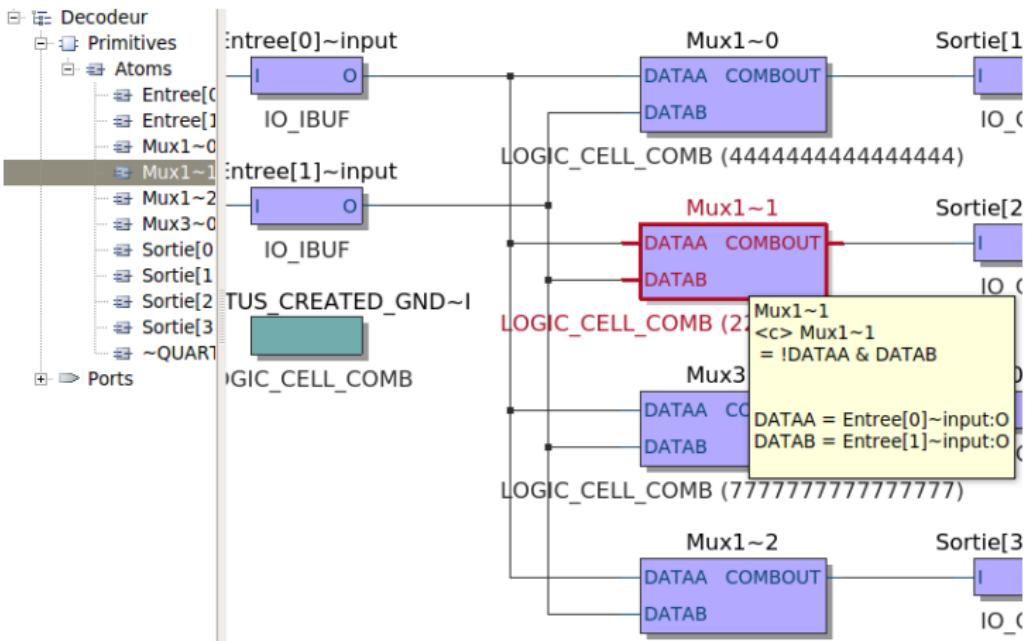
- On constate qu'il y a bien utilisation de multiplexeurs
- On retrouve l'entrée du décodeur comme sélecteur
- L'entrée du multiplexeur devient une constante



LE DÉCODEUR

VUE DU NIVEAU CELLULES FPGA

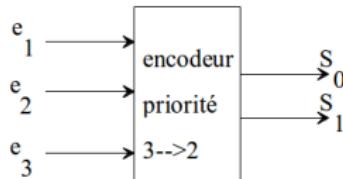
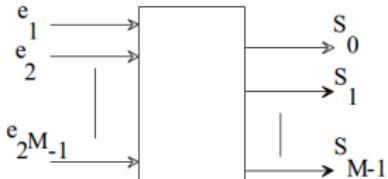
- On voit l'optimisation par rapport à la vue RTL



L'ENCODEUR

FONCTION ET SPÉCIFICATIONS

- Donne l'indice le plus élevé parmi les entrées actives
- Presque fonction «réciproque» du décodeur
- Priorité d'évènements (interruptions)
- $\{E_1; E_2; \dots; E_N\}$: entrées binaires
- Sortie : entier (nb bits $\geq \log_2(N)$)
- Sortie = N si E_N active sinon N-1 si E_{N-1} active sinon... 1 si E_1 active sinon 0



$$S_0 = e_3 + \bar{e}_3 \cdot \bar{e}_2 \cdot e_1$$

$$S_1 = e_3 + e_2$$



L'ENCODEUR

DESCRIPTION VHDL

- Nous utilisons un **process** pour bénéficier des indéterminations dans les choix

```
entity Encodeur is
port(
    Entree : in std_logic_vector(2 downto 0);
    Sortie : out std_logic_vector(1 downto 0));
end Encodeur;

architecture ProCase of Encodeur is begin
process (Entree)
begin
case? Entree is
    when "1--" => Sortie <= "11" ;
```



L'ENCODEUR

DESCRIPTION VHDL

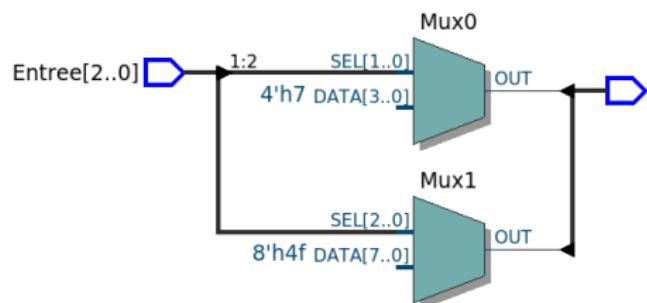
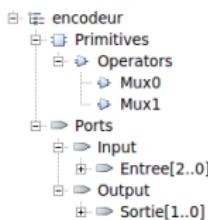
```
        when "01-" => Sortie <= "10" ;
        when "001"  => Sortie <= "01" ;
        when others => Sortie <= "00" ;
    end case? ;
end process;
end ProCase;
```



L'ENCODEUR

VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)

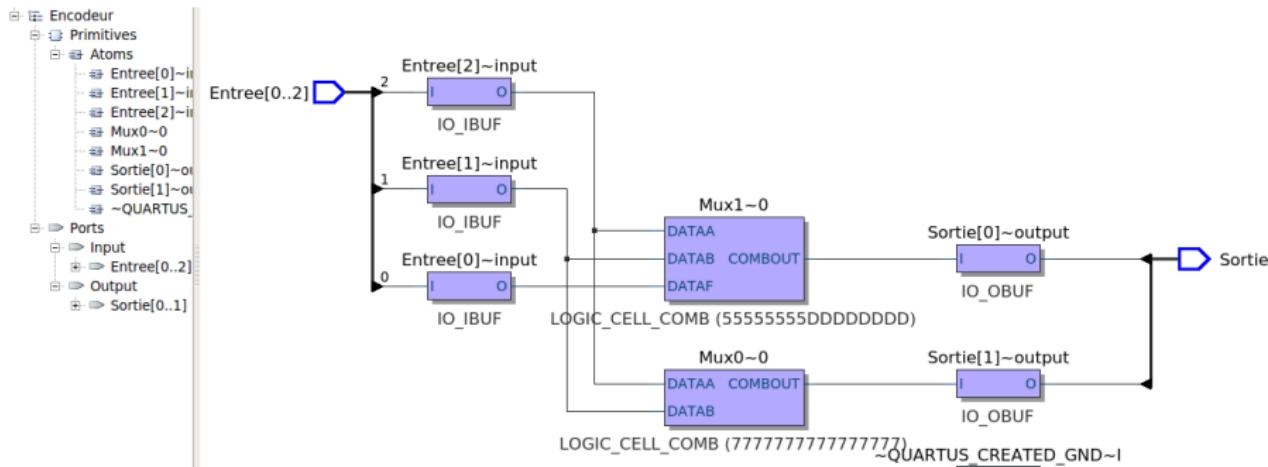
- On constate à nouveau l'utilisation de multiplexeurs
- On retrouve l'entrée de l'encodeur comme sélecteur
- L'entrée du multiplexeur devient une constante



L'ENCODEUR

VUE DU NIVEAU CELLULES FPGA

- On voit à nouveau l'optimisation par rapport à la vue RTL





EXERCICES

- ➊ Reprendre le multiplexeur et le modifier pour qu'il soit de type $5 \rightarrow 1$. Utiliser un **process**
- ➋ Que se passe-t-il si l'on n'utilise pas le cas **others** ?
- ➌ Reprendre le décodeur et le décrire en utilisant un **process**
- ➍ Reprendre l'encodeur et le décrire en utilisant uniquement des instructions **concurrentes**

