



# SPÉCIFICATION DES CIRCUITS INTÉGRÉS

## TÉLÉCOM NANCY

Slaviša Jovanović et Yves Berviller

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

10 janvier 2022

# 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
- optimiser la synthèse

# OBJECTIFS ET CONTENU DU MODULE

- 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 ;
- L'objectif principal :

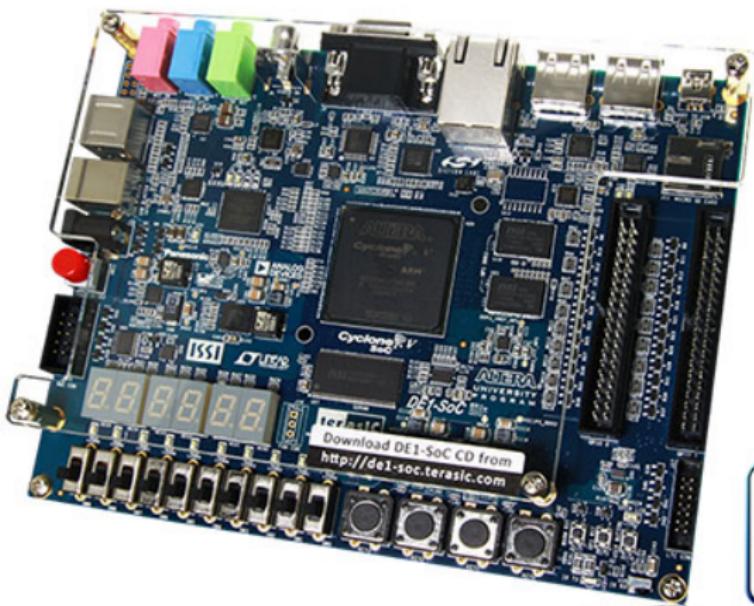
# OBJECTIFS ET CONTENU DU MODULE

- ▷ 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 21.0



# 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

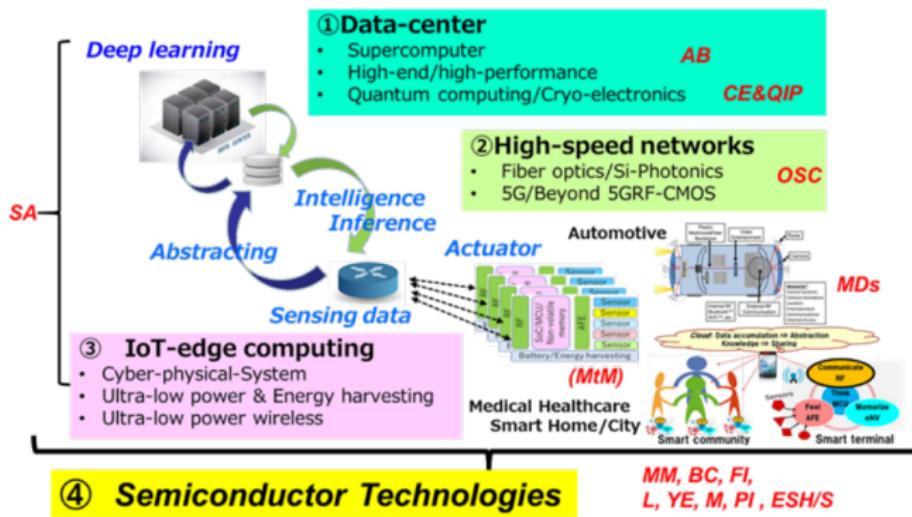
## MARCHÉ MONDIAL DE L'ÉLECTRONIQUE PAR SEGMENTS



. Source : DECISION March 2013 (Embedded Systems)

# MOTIVATIONS

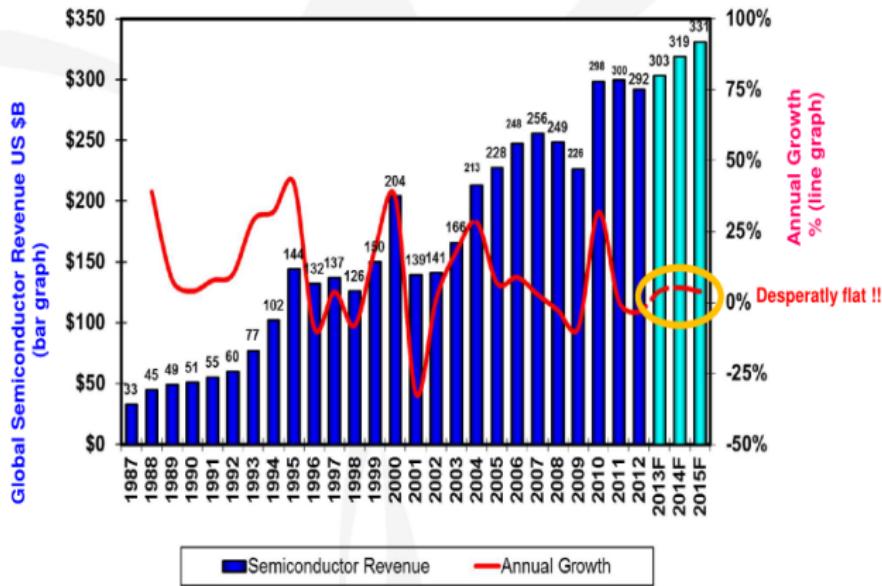
## DOMAINES D'APPLICATIONS



AB: Applications Benchmarking, SA: Systems and Architecture, OSC: Outside system Connectivity, MM: More Moore, BC: Beyond CMOS, CE&QIP: Cryogenics Electronics and Quantum Information Processing, PI: Packaging Integration, FI: Factory Integration, L: Lithography, YE: Yield Enhancement, M: Metrology, ESH/S: Environment, Safety, Health, and Sustainability, MtM: More than Moore, MDs: Market drivers (automobile, medical devices).

# MOTIVATIONS

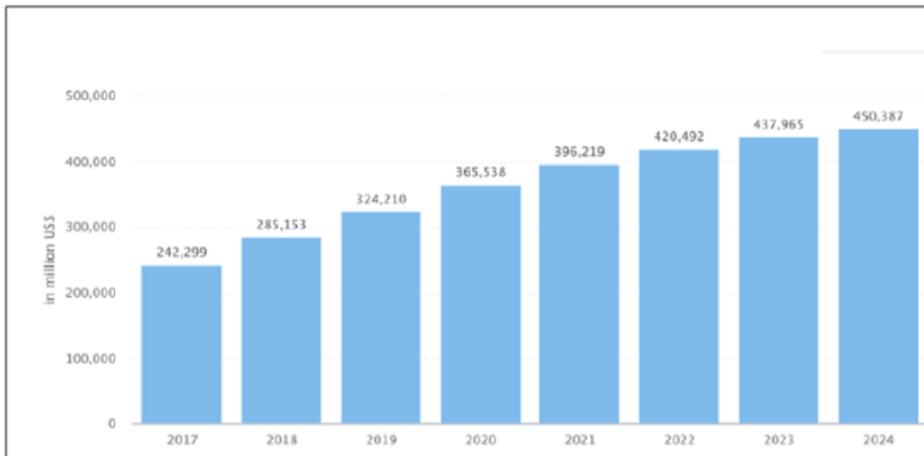
## CYCLES DE L'INDUSTRIE DE SEMICONDUCTEURS



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

# MOTIVATIONS

## CYCLES DE L'INDUSTRIE DE SEMICONDUCTEURS

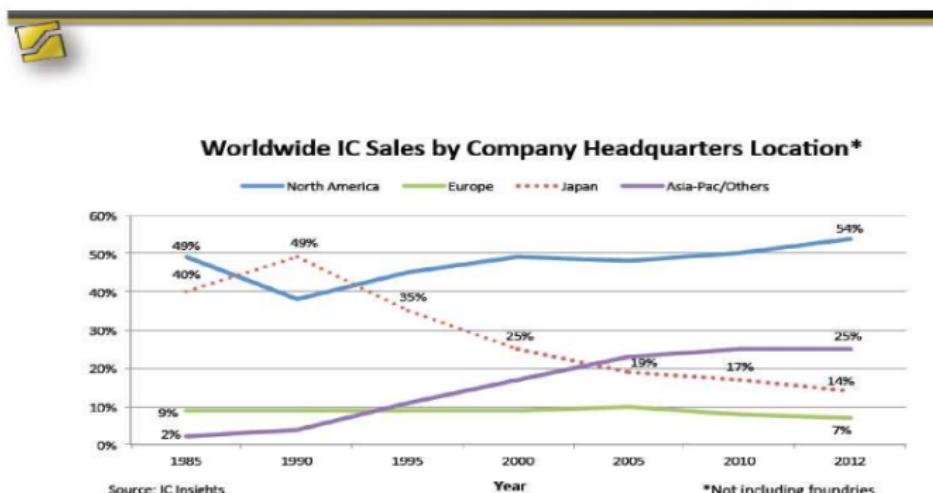


Source: Statista

Source : IRDS 2020

# MOTIVATIONS

## VENTES DE SEMICONDUCTEURS DANS LE MONDE



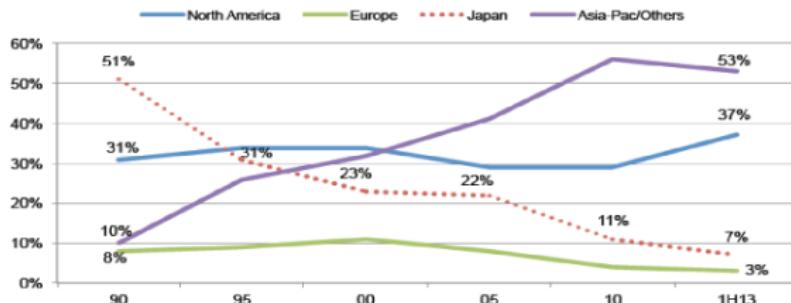
- L'Europe et le Japon en baisse

# MOTIVATIONS

## LA PRODUCTION DE SEMICONDUCTEURS DANS LE MONDE



Semiconductor Capital Expenditures by Region



Source: IC Insights

IC Insights

September 26, 2013

European Microelectronics Summit

- L'Europe et le Japon à la peine ...

# MOTIVATIONS

CES - *chips for everything*



Infomotions



Reebok



Sports



Sony Tennis



ibitz

wearable devices



Fitbit, bitfit, fitfit ...



Instabeats



UV jewel



Interchangeable ehealth

# MOTIVATIONS

## CES - Internet of things



smartglasses



Fitbit for dogs



IoT Cisco



Smartwatches

Bed measure....



Mother Sen.se



4K wearable



Smarthome



# MOTIVATIONS

## CES - other applications



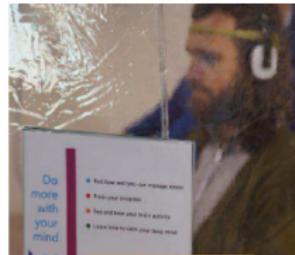
Robots



AirDrones



SmartKeys



Brain sensing



Immersive Gaming



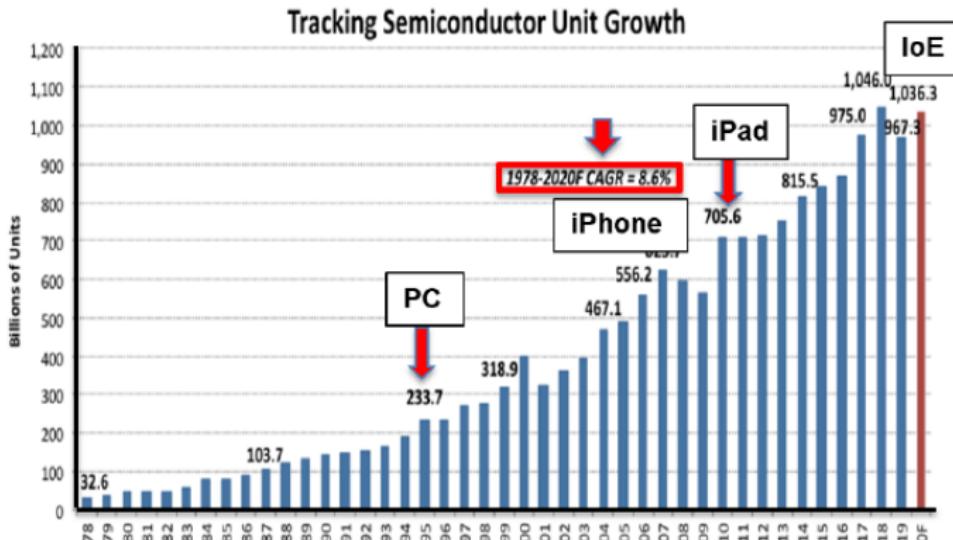
3D scanners



3D printers

# MOTIVATIONS

## PRODUCTION DES SEMICONDUCTEURS

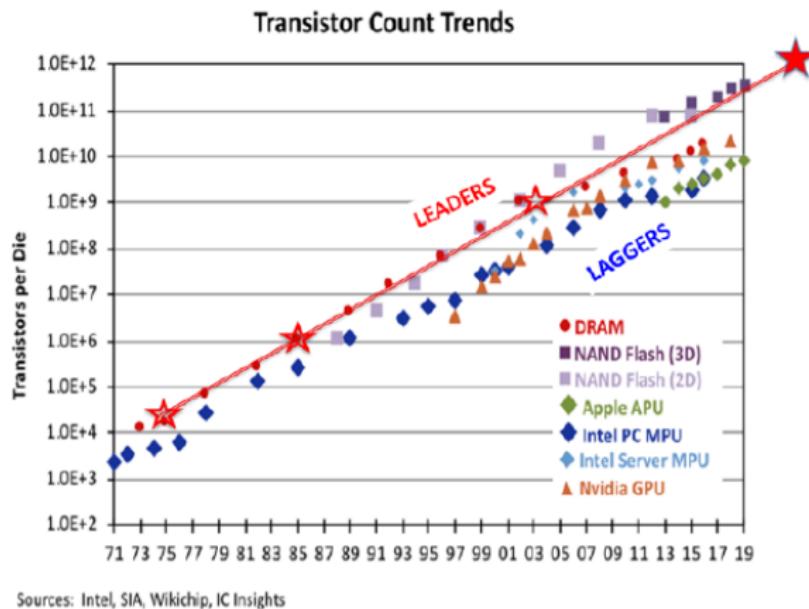


Source: IC Insights

Source : IRDS 2020

# MOTIVATIONS

## PRODUCTION DES SEMICONDUCTEURS



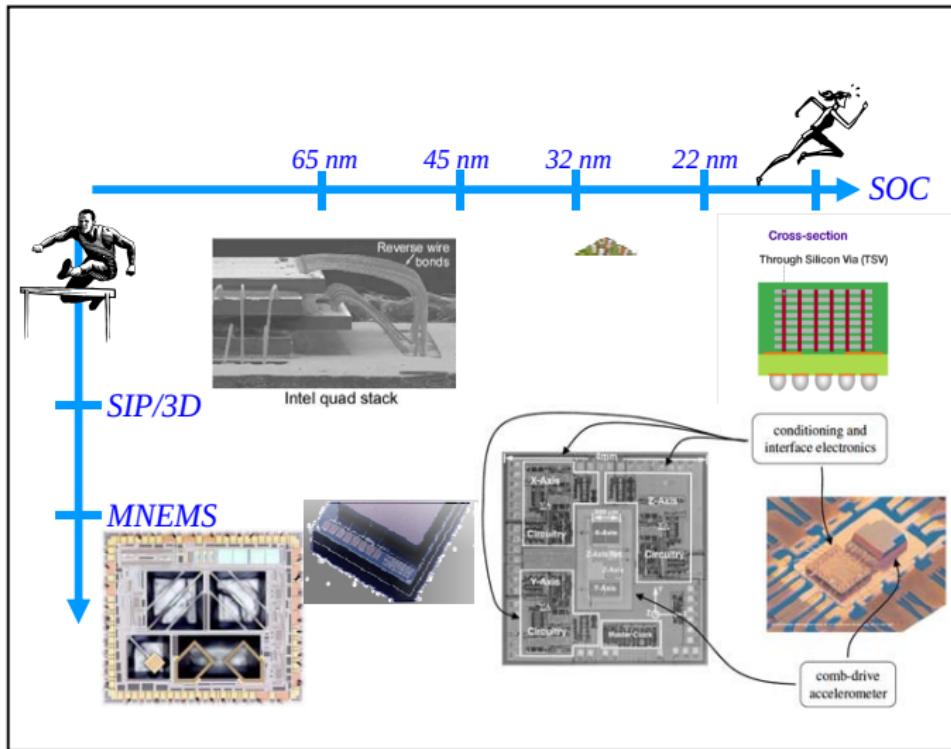
Source : IRDS 2020

# MOTIVATIONS

## DÉFIS À RELEVER

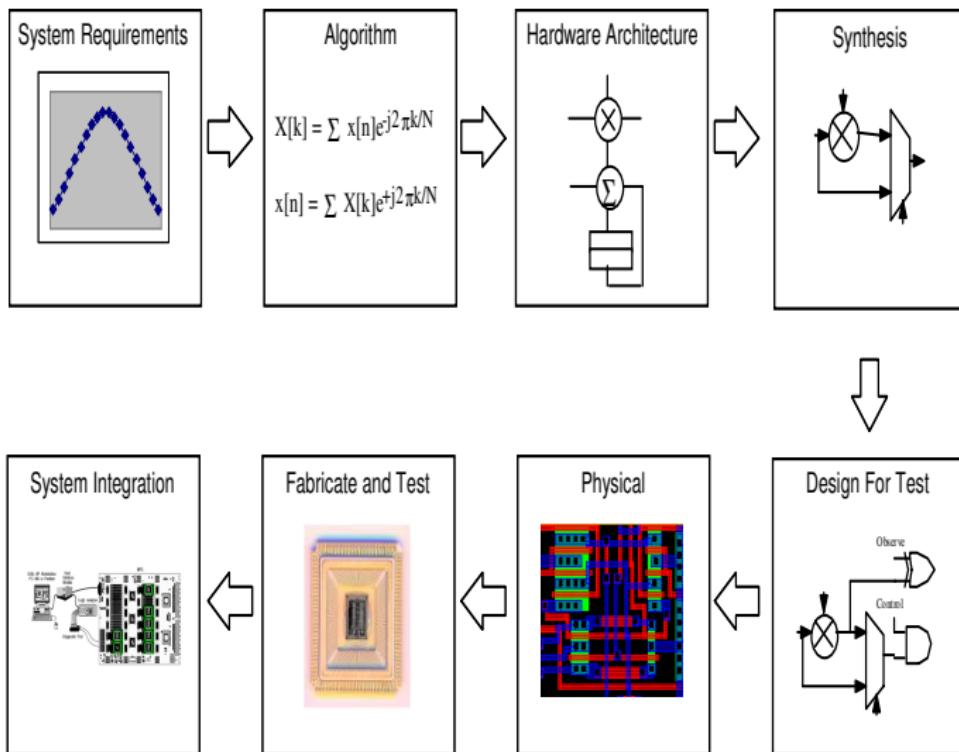
- De nouvelles puces plus performantes pour irriguer les nouveaux marchés de croissance
  - ▷ santé, sécurité, contrôle de l'énergie, objets communicants, ...
- Ruptures au niveau technologique au plan système
- Technologies *More than Moore* et *More Moore*
- marché européen : analogique et MEMS, RF, 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

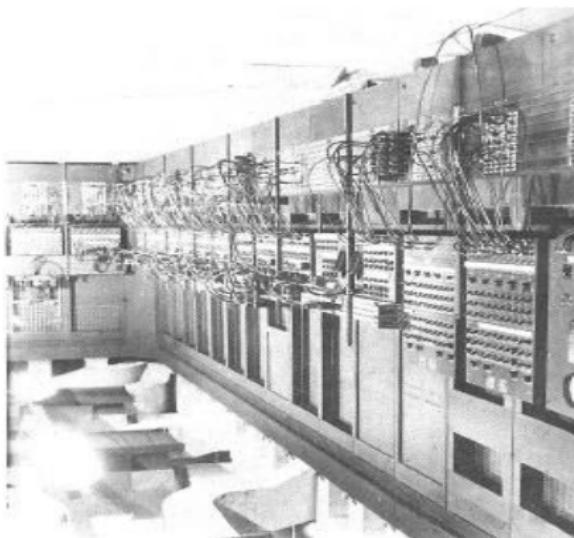


- 1832 : « *The Babbage Difference Engine* »
- 25,000 pièces
- coût 17,470 livres sterling

# 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



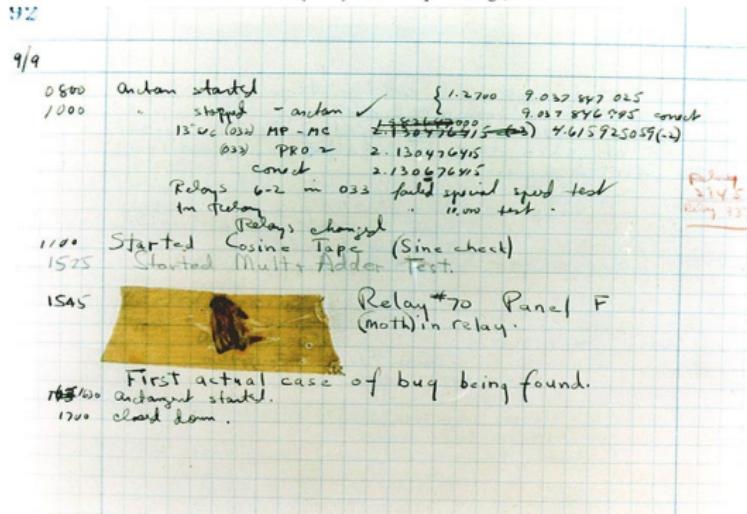


# PREMIER ORDINATEUR

## ORDINATEUR ÉLECTRONIQUE À L'ORIGINE DU *bug*

- 1945 : Harvard Mark II
  - ▷ Premier BUG

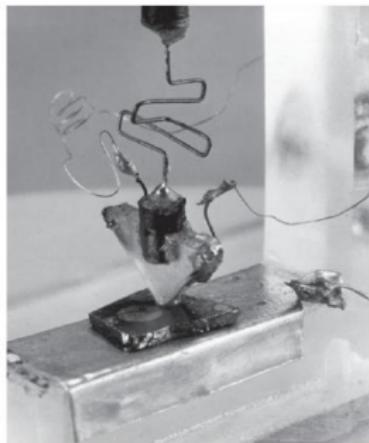
Photo # NH 96566-KN (Color) First Computer "Bug", 1947



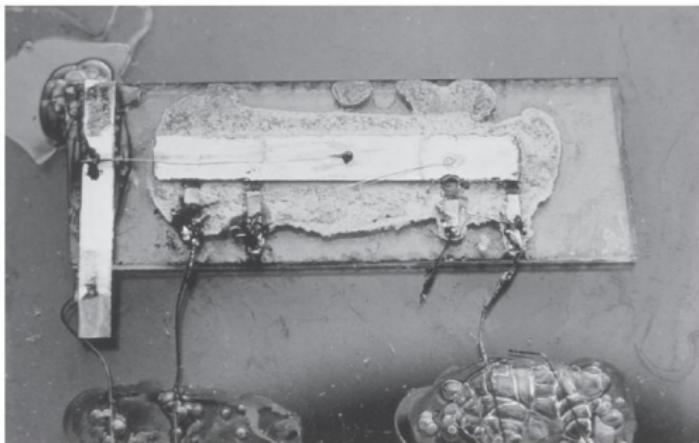
# TRANSISTOR BIPOLAIRE

## 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)

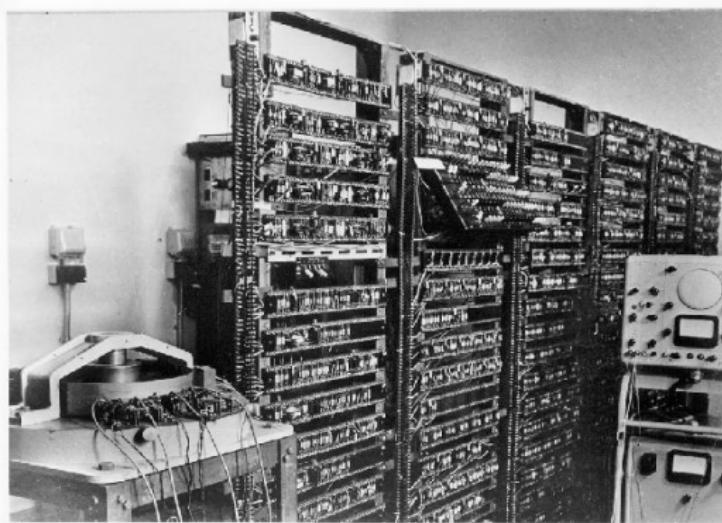


(b)

# TRANSISTOR BIPOLAIRE

## ORDINATEUR À TRANSISTORS BIPOLAIRES

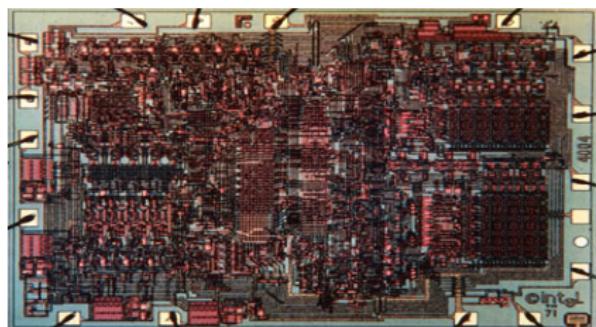
- 1953 : Ordinateur à transistors
  - ▷ 92 transistors
  - ▷ 550 diodes



# 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  $\mu\text{m}$
  - ▷ 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  $7$  et  $5 nm$  → annonces  $3 nm$  pour 2022
- $\frac{10\mu m}{5nm} = 2000$
- conséquence : démocratisation totale des circuits VLSI et leur omniprésence dans tous les domaines d'applications

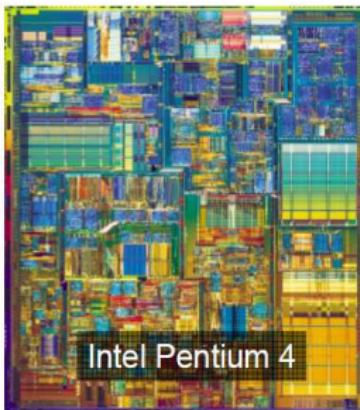
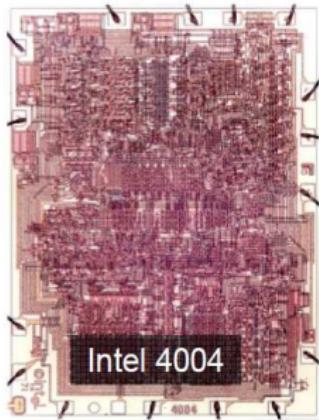
# TRANSISTOR CMOS

## RÉVOLUTION MICROÉLECTRONIQUE

- Intel a produit les circuits en technologie 14nm en début de 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

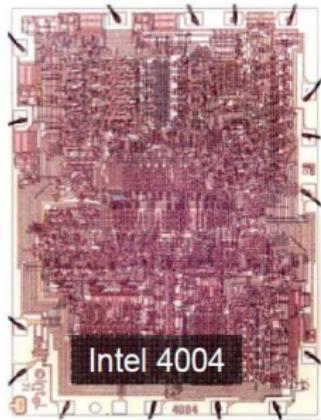
## ÉVOLUTION



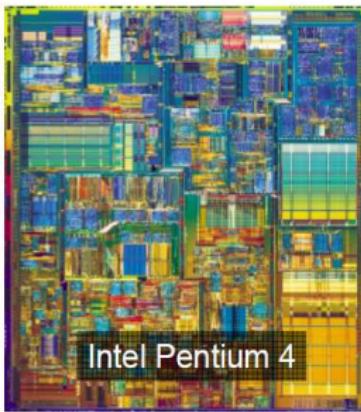
Year	1971	2001
Transistors	2,300	42,000,000
Speed (kHz)	108	2,000,000
CD ( $\mu\text{m}$ )	10.00	0.13

# TRANSISTOR CMOS

## ÉVOLUTION



Intel 4004

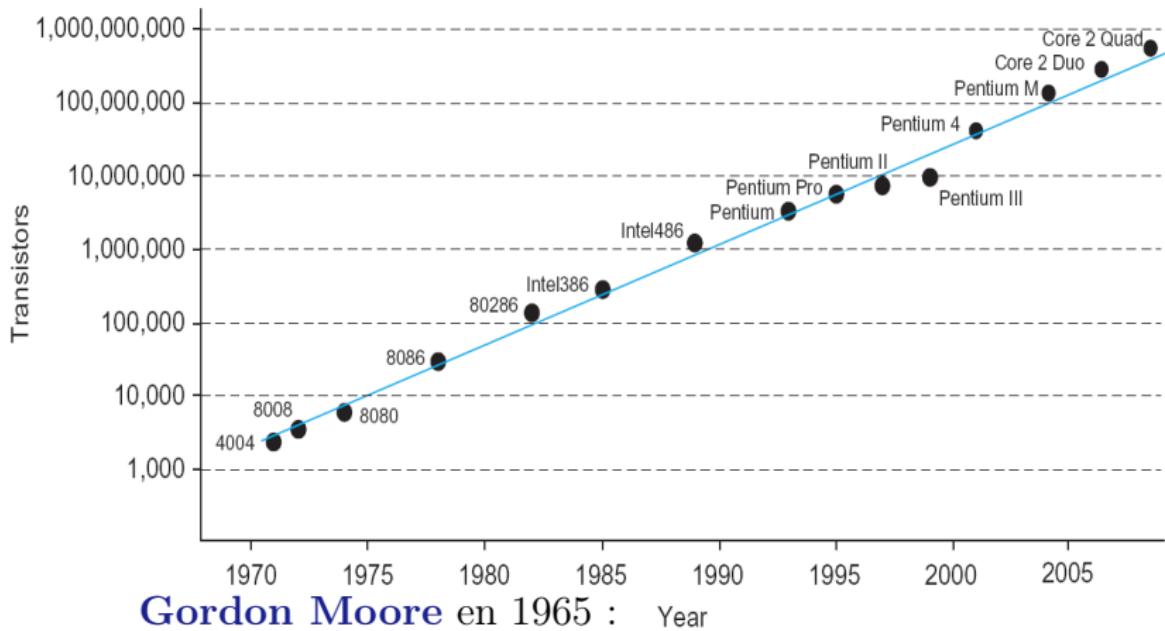


Intel Pentium 4

- 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

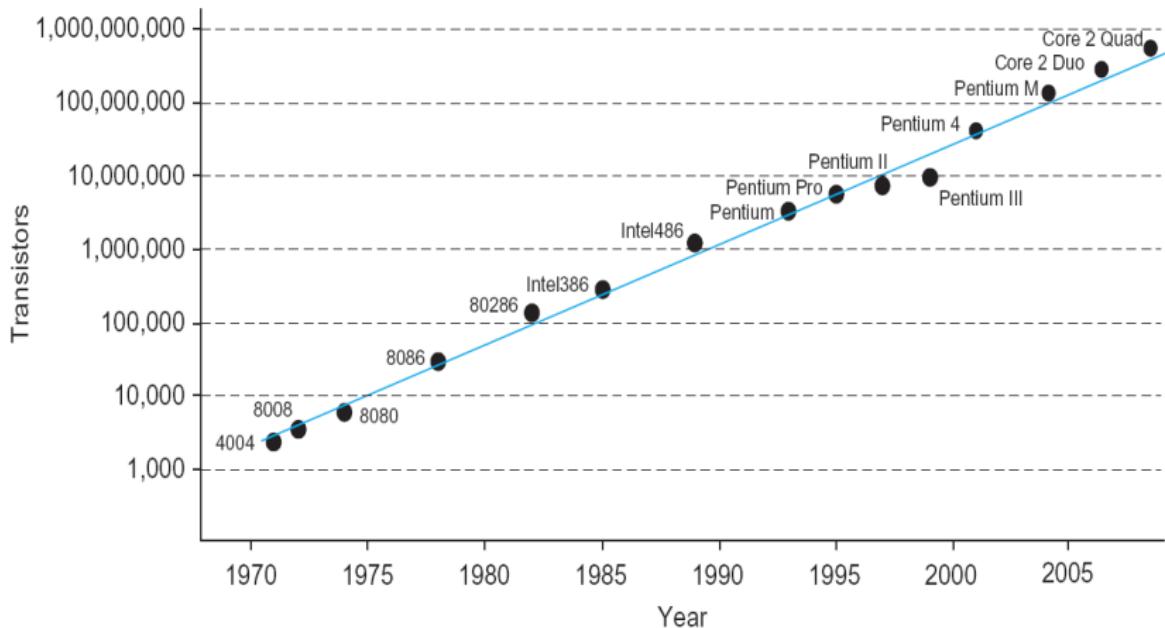
# LOI DE MOORE



Gordon Moore en 1965 : Year

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

# LOI DE MOORE



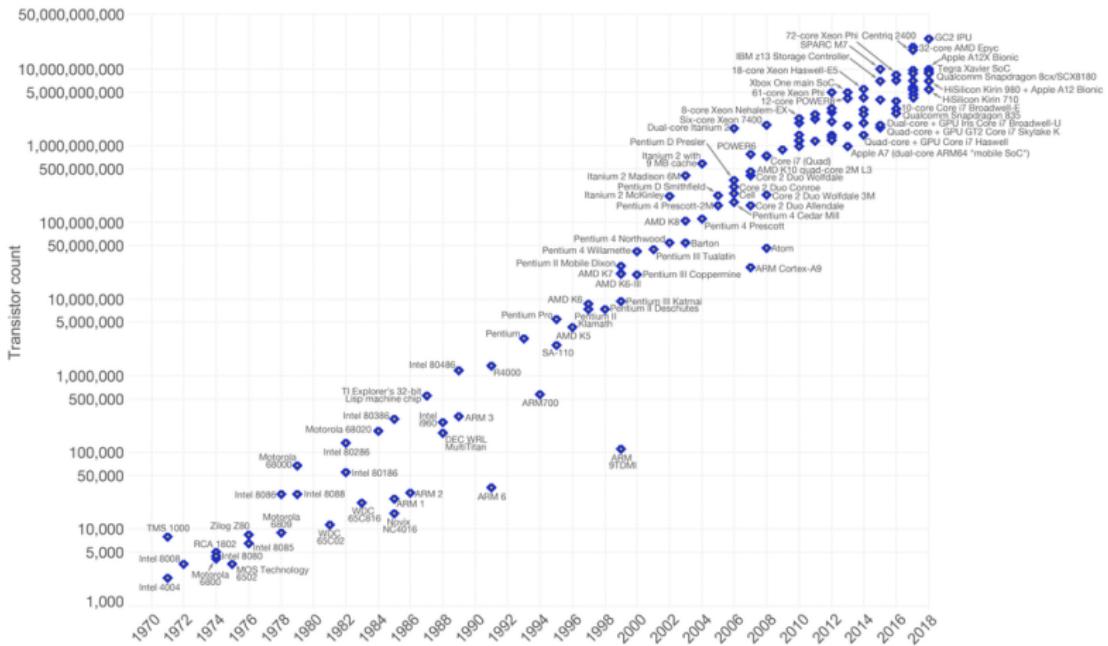
- Le nombre de transistors sur des processeurs Intel double tous les 26 mois

# LOI DE MOORE

## Moore's Law – The number of transistors on integrated circuit chips (1971-2018)

Moore's law describes the empirical regularity that the number of transistors on integrated circuits doubles approximately every two years. This advancement is important as other aspects of technological progress – such as processing speed or the price of electronic products – are linked to Moore's law.

OurWorld  
in Data

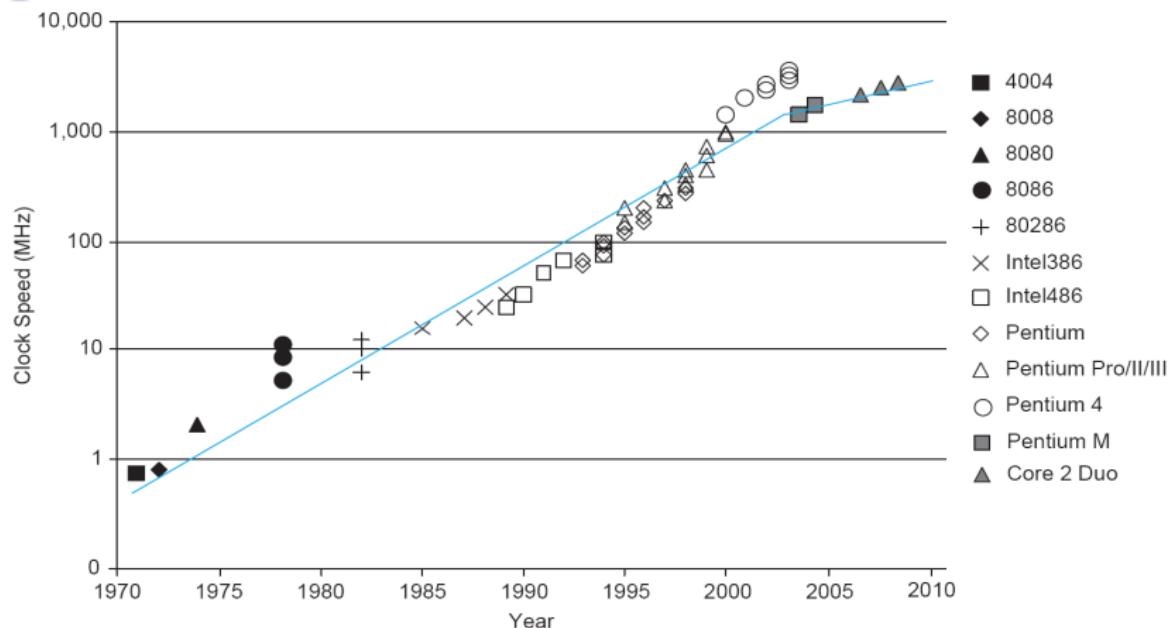


Data source: Wikipedia ([https://en.wikipedia.org/wiki/Transistor\\_count](https://en.wikipedia.org/wiki/Transistor_count))

The data visualization is available at OurWorldInData.org. There you find more visualizations and research on this topic.

Licensed under CC-BY-SA by the author Max Roser.

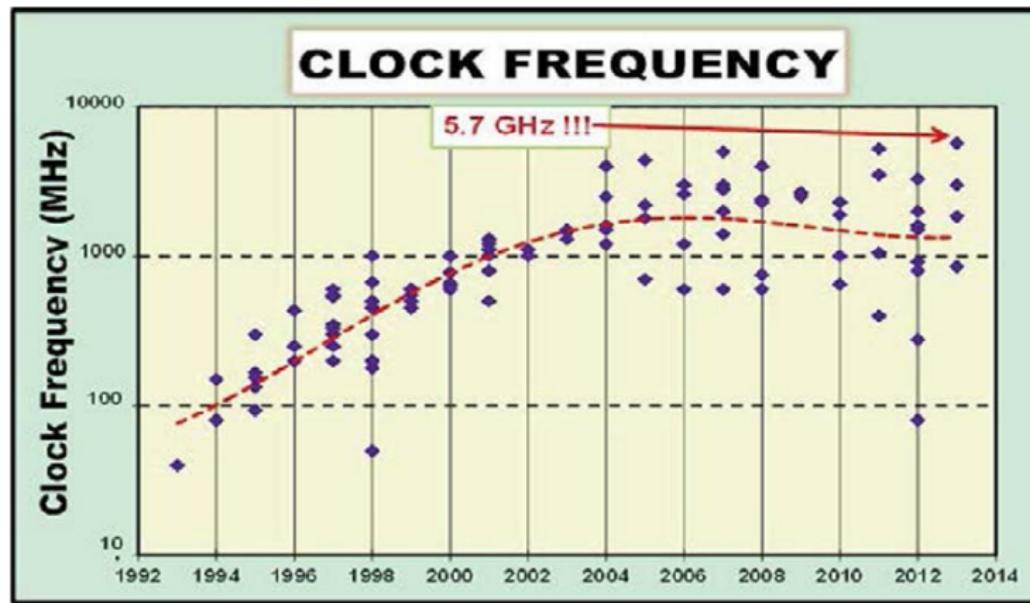
## LOI DE MOORE II



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

# LOI DE MOORE II

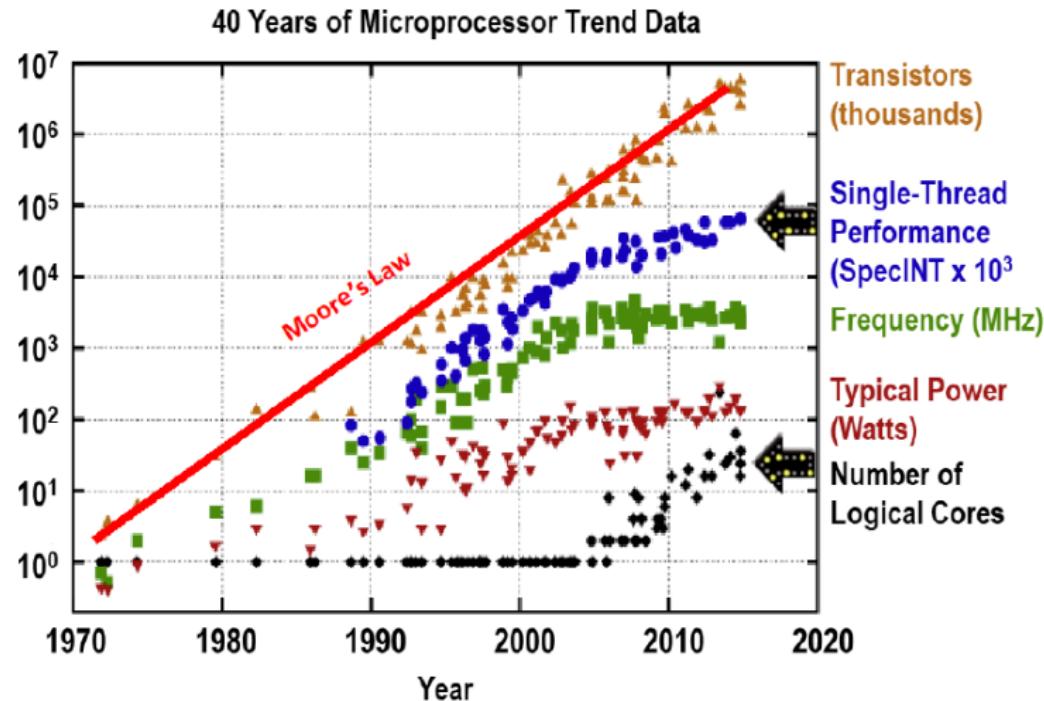
## FRÉQUENCE



IEEE, ISSCC: Transistor's 60<sup>th</sup> year commemorative supplement

Source : IRDS 2020

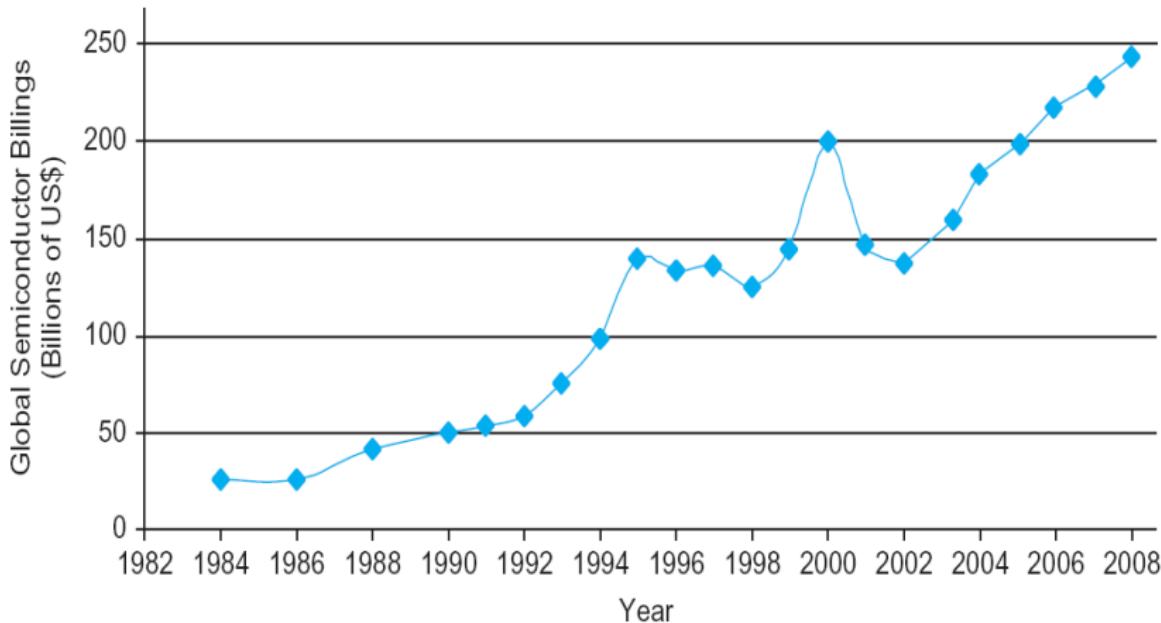
## LOT DE MOORE II



. Source : IRDS 2020

# ENJEUX ÉCONOMIQUES

## MARCHÉ DES SEMICONDUCTEURS



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



# SOMMAIRE

## 1 Introduction

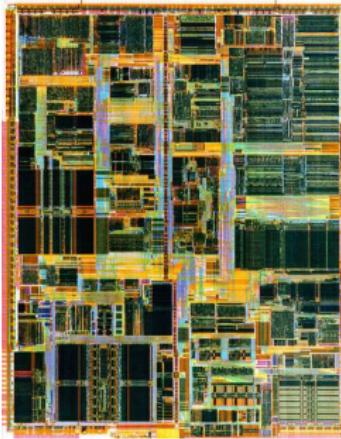
- Motivations
- Histoire des semiconducteurs
- Circuits intégrés
- Conception de circuits intégrés numériques

# INTRODUCTION

## DOMAINES D'APPLICATIONS

Digital  
CPU,  
Mémoire,  
...

**systèmes VLSI / SoC**



DSP  
Audio/video  
Mpeg  
...

Application  
multimédia  
Communication,  
Calcul ...

MEMS  
μcapteurs CCD ..  
μtransformateurs ..  
μrésonateurs ..

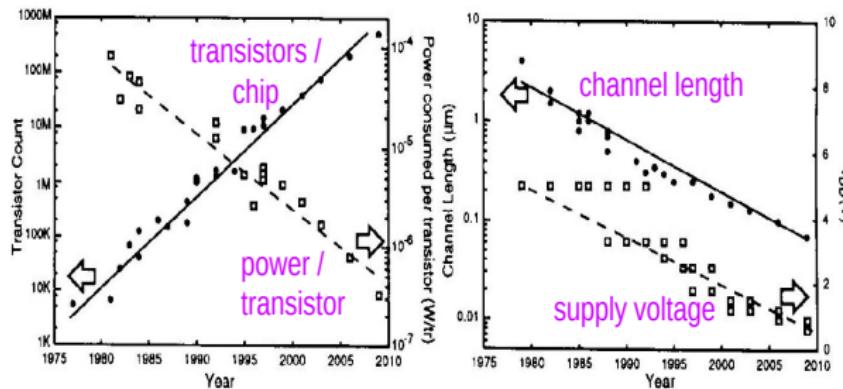
RF/ analogique  
Filtres LNA,  
mélangeurs,  
VCO ...

Gestion  
d'énergie  
Convertisseurs,  
régulateurs ..

# TECHNOLOGIE CMOS

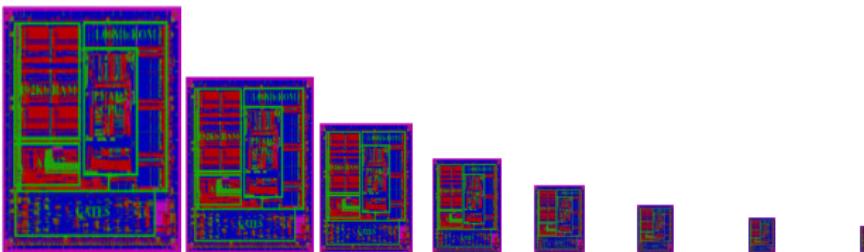
## TENDANCES

- 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

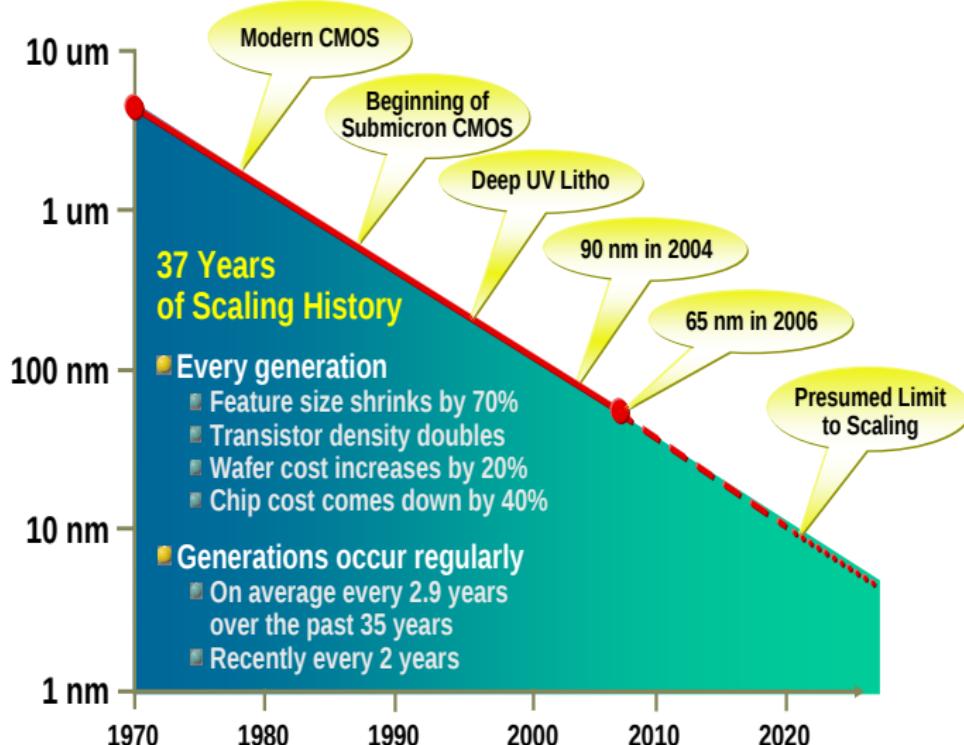


Year	1994	1997	1999	2000	2002	2004	2006	2008
Nano-meter	500nm	350nm	250nm	180nm	130nm	90nm	65nm	45nm
Wafer size	6"	8"	8"	8"	12"	12"	12"	12"
Die size (mm <sup>2</sup> )	80.7	46.6	19.2	10.7	6.7	4.2	2.4	1.4
Dies per wafer	310	950	2550	4700	12,200	18,700	26,500	46,500

→ 150X increase in die per wafer →

# TECHNOLOGIE CMOS

## Technology Scaling



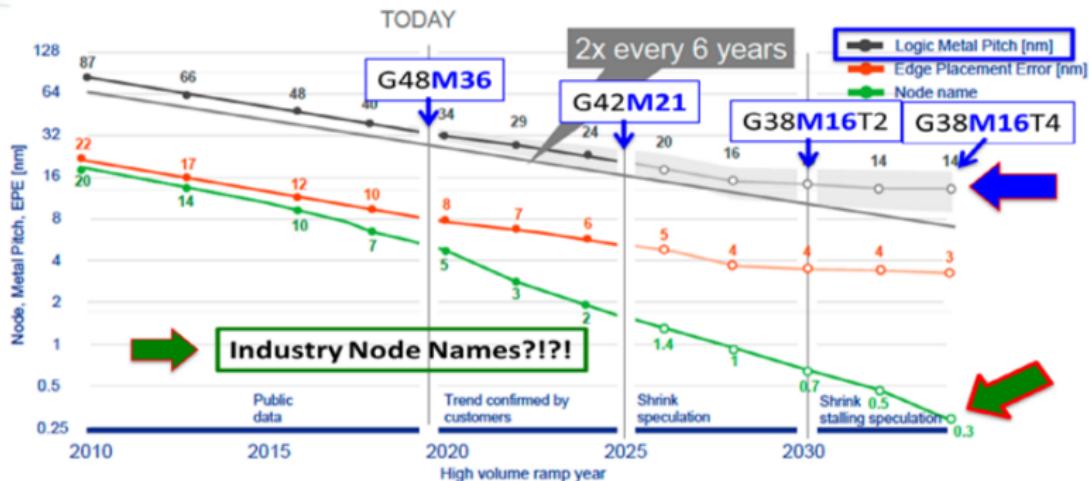
# TECHNOLOGIE CMOS

## Technology Scaling

Dimensional scaling continues another decade  
Edge Placement Error reduction accordingly

ASML

Slide 14  
IEDM Dec 2019



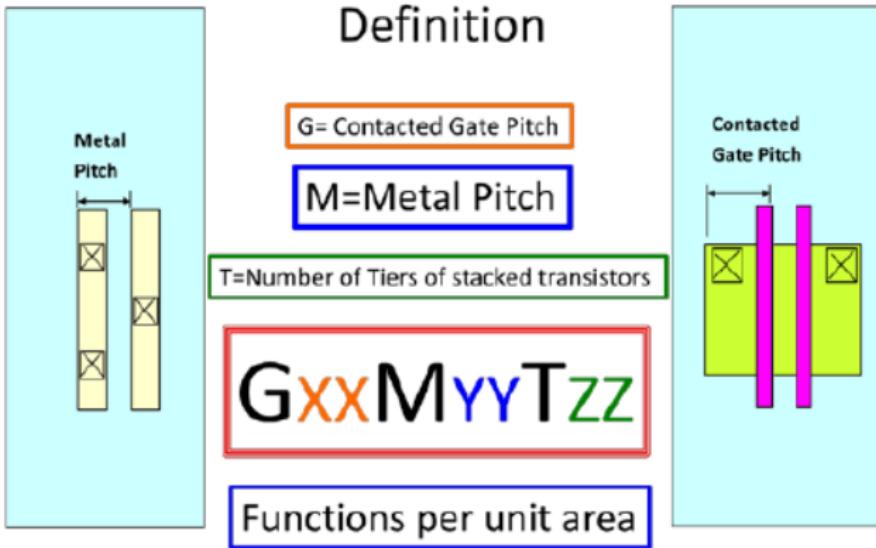
Source: Average customer roadmap 2019 extended by ASML extrapolation averaged with IDRS roadmap, IRDS, Mustafa Badaroglu, More Moore out brief, Nov. 2018 (International Roadmap for Devices and Systems)

Source : IRDS 2020

# TECHNOLOGIE CMOS

## Technology Scaling

### NTRS/ITRS->IRDS Technology Node Definition



Source : IRDS 2020

# TECHNOLOGIE CMOS

## Technology Scaling

Logic/Foundry Process Roadmaps (for Volume Production)

	2015	2016	2017	2018	2019	2020	2021
Intel		14nm+	10nm (initial) 14nm++		10nm	10nm+	7nm EUV 10nm++
Samsung		28nm FDSOI	10nm	8nm	7nm EUV 6nm EUV	18nm FDSOI 5nm	4nm
TSMC	16nm+ finFET	10nm	7nm 12nm	7nm+ EUV	5nm 6nm	5nm+	
GlobalFoundries	14nm finFET		22nm FDSOI 12nm finFET		12nm FDSOI	12nm+ finFET	
SMIC	28nm			14nm finFET	12nm finFET		
UMC		14nm finFET			22nm planar		

Note: What defines a process "generation" and the start of "volume" production varies from company to company, and may be influenced by marketing embellishments, so these points of transition should only be seen as very general guidelines.

Sources: Companies, conference reports, IC Insights

Source : IRDS 2020

# TECHNOLOGIE CMOS

## Technology Scaling

YEAR OF PRODUCTION	2020	2022	2025	2028	2031	2034
Logic industry "Node Range" labeling (nm)	G48M36	G46M24	G42M20	G40M16	G38M16T2	G38M16T4
IDM-Foundry node labeling	"5"	"3"	"2.1"	"1.5"	"1.0 eq"	"0.7 eq"
Logic device structure options	I7-f5	I5-f3	I3-f2.1	I2.1-f1.5	I1.5e-f1.0e	I1.0e-f0.7e
Mainstream device for logic	FinFET	FinFET LGAA	LGAA	LGAA	LGAA-3D	LGAA-3D
LOGIC TECHNOLOGY ANCHORS						
Patterning technology inflection for Mx interconnect	193i, EUV DP	193i, EUV DP	193i, EUV DP	193i, High-NA EUV	193i, High-NA EUV	193i, High-NA EUV
Beyond CMOS as complimentary to mainstream CMOS	-	-	-	2D Device, FeFET	2D Device, FeFET	2D Device, FeFET
Channel material technology inflection	SiGe25%	SiGe50%	SiGe50%	Ge, 2D Mat	Ge, 2D Mat	Ge, 2D Mat
Process, technology inflection	Conformal doping, Contact	Channel RMG	Lateral/Atomic Etch	Non-Cu Mx	3D VLSI	3D VLSI
Stacking generation inflection	2D	3D stacking: W2W, D2W Mem-on-Logic	3D stacking: W2W, D2W Mem-on-Logic	3D stacking, Fine-pitch stacking, P-over-N, Mem-on-Logic	3D stacking, 3D VLSI: Mem-on-Logic with Interconnect	3D stacking, 3D VLSI: Logic-on-Logic

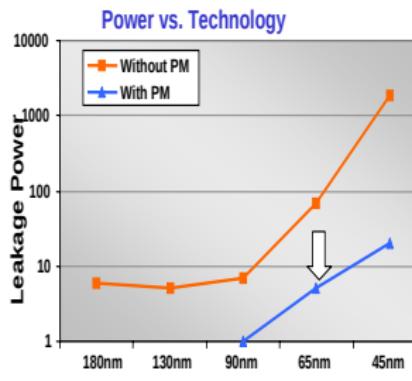
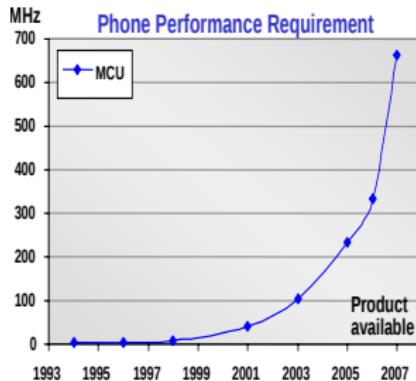
Note: Mx—Tight-pitch routing metal interconnect. IDM—Independent device manufacturer. FinFET—fin field-effect transistor. LGAA—lateral gate all around. EUV—extreme ultraviolet. NA—numerical aperture. Ge—germanium. SiGe—silicon germanium. RMG—replacement metal gate. VLSI—very large scale integration. W2W—wafer to wafer. D2W—die to wafer. Mem-on-Logic—memory on logic

Source : IRDS 2020

# TECHNOLOGIE CMOS

## CONSOMMATION

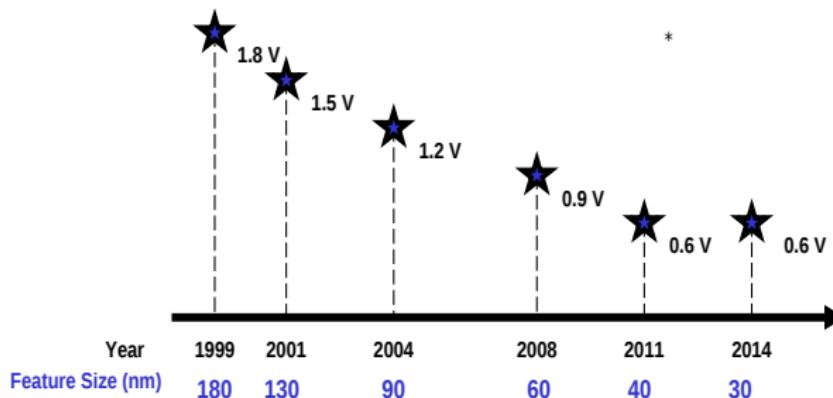
- $P = f(C, V^2, f, \text{Leakage})$
- $C$  diminue avec les avancées technologiques
- $V$  pratiquement constant, au niveau le plus bas possible (?)
- $f$  augmente avec les avancées technologiques
- *Leakage* (courants de fuite) augmente avec les avancées technologiques et la température (qui augmente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

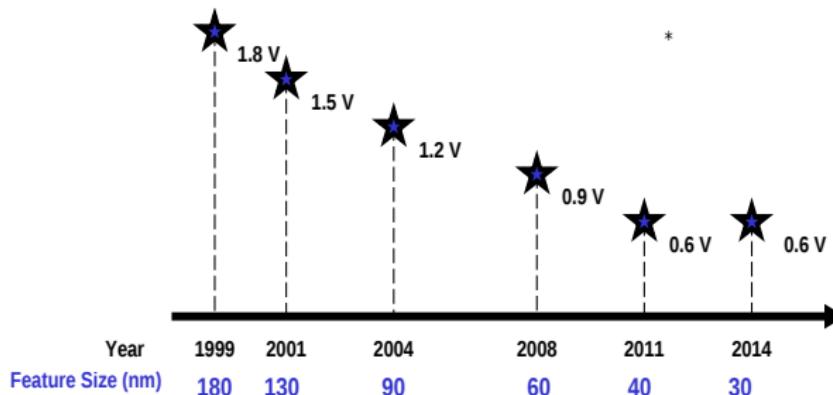


# TECHNOLOGIE CMOS

## 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

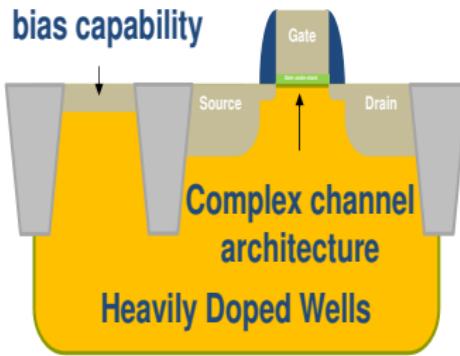
Deux solutions actuelles :

- FDSOI et
- FinFET

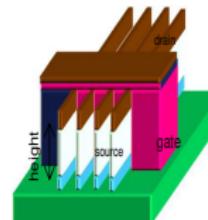
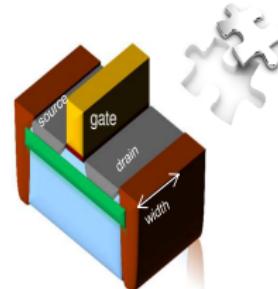
# TECHNOLOGIE CMOS

## TRANSISTOR

Limited body bias capability



FD-SOI = 2D



FinFET = 3D

. Source STMicroelectronics

# 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

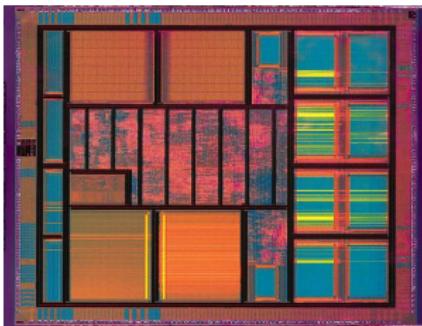
**Exemple :** Réalisation d'un système de comptage comprenant 3 compteurs de 4 bits

- 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 prouesses 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

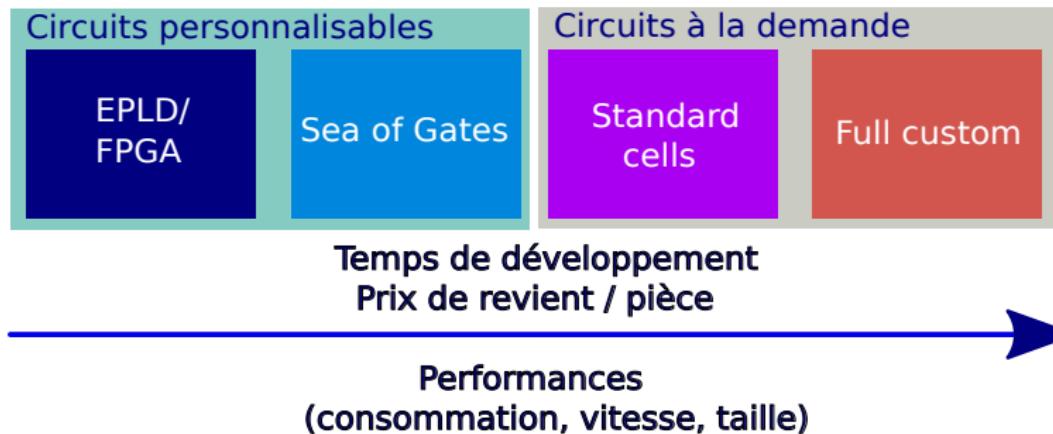
# CIRCUITS INTÉGRÉS

## 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

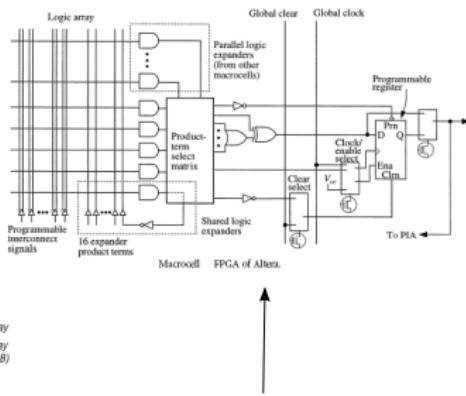
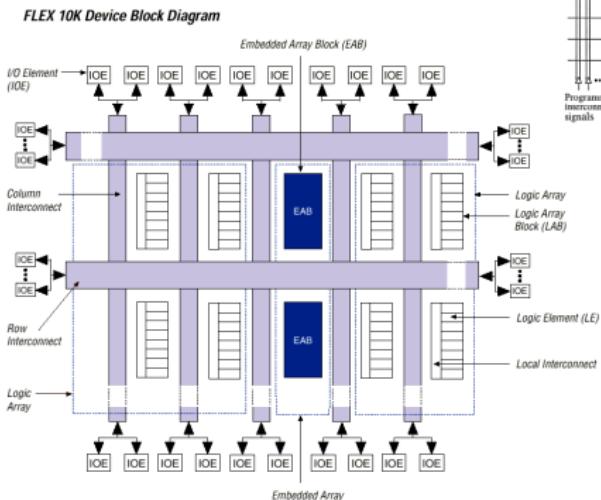
# CIRCUITS INTÉGRÉS

## 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

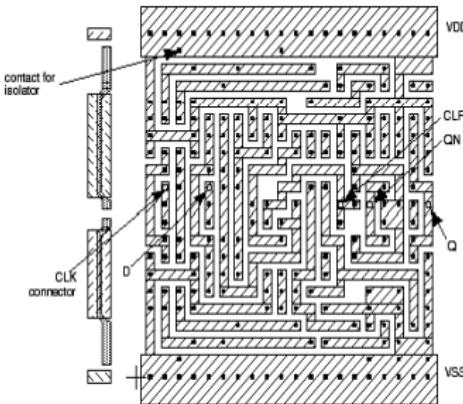
# CIRCUITS INTÉGRÉS

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

Puce en partie réalisée

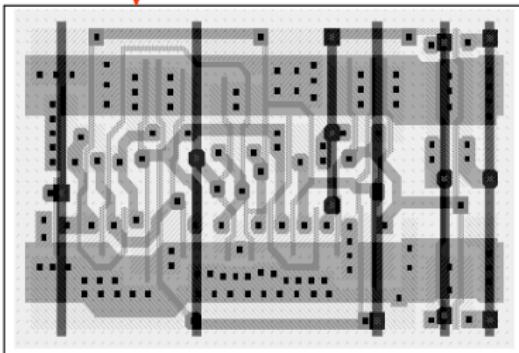
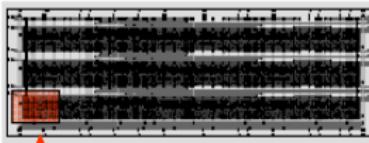
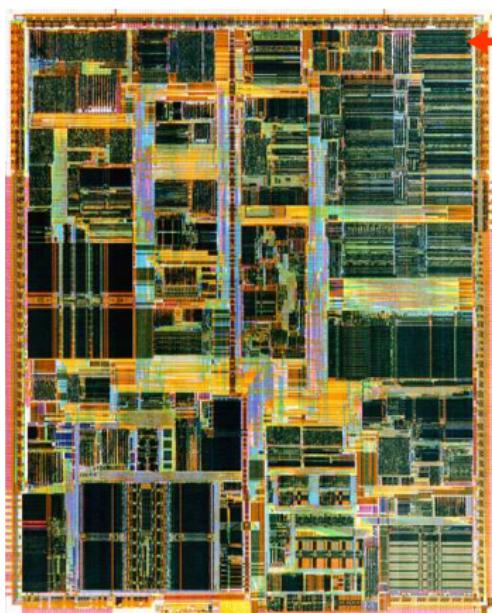


Connexions fournies  
par le concepteur



# CIRCUITS INTÉGRÉS

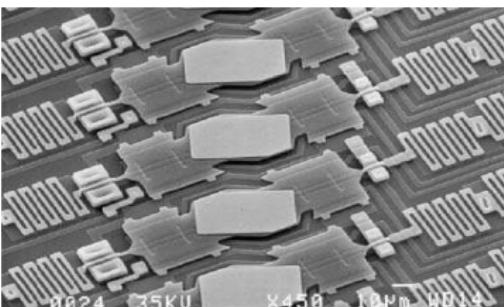
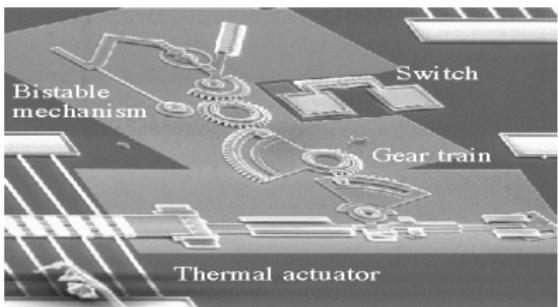
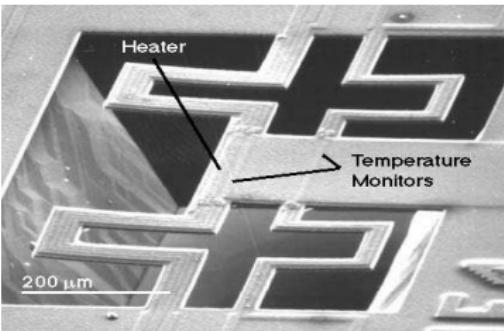
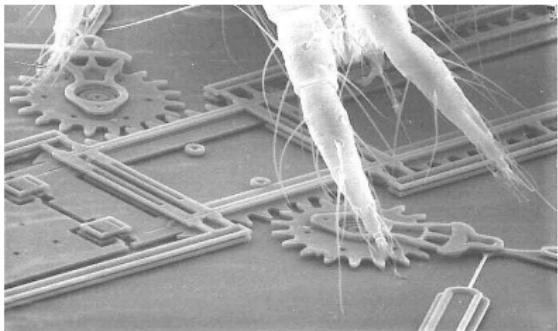
LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : *Standard Cells*



# CIRCUITS INTÉGRÉS

## LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : MEMS

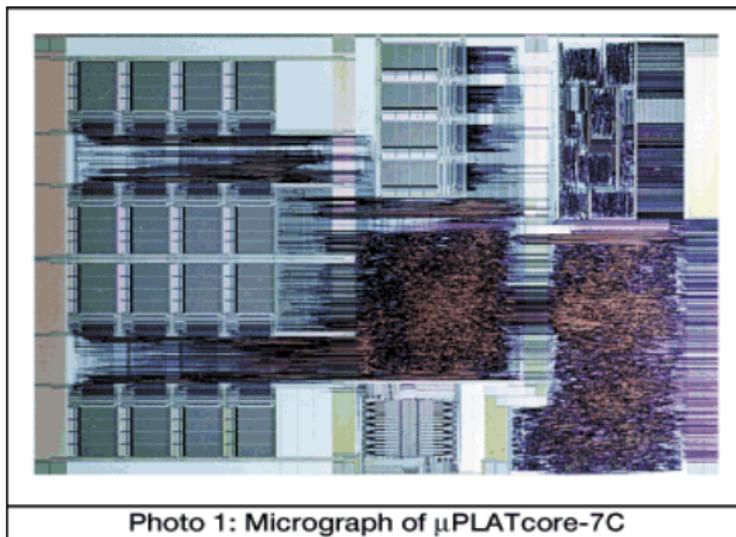
- MEMS : *Micro Electro-Mechanical Systems*



# CIRCUITS INTÉGRÉS

## LES DIFFÉRENTS TYPES DE CIRCUITS INTÉGRÉS : IP

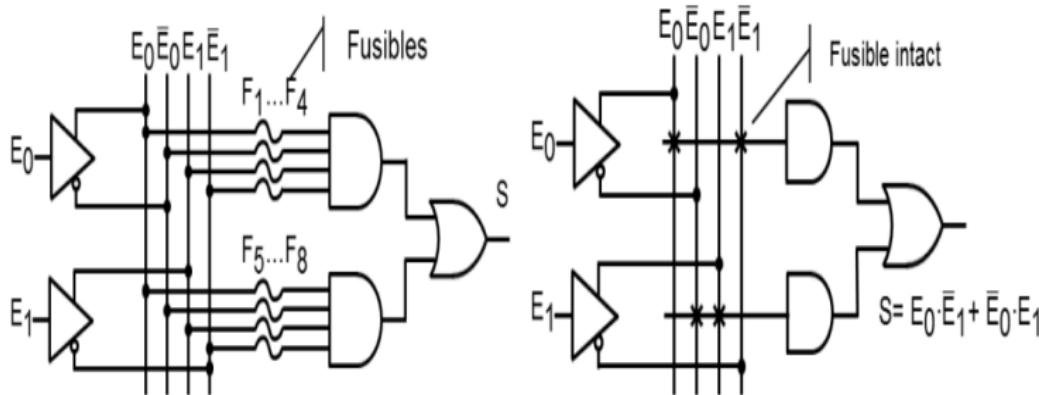
- IP : *Intellectual Property*



# 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).

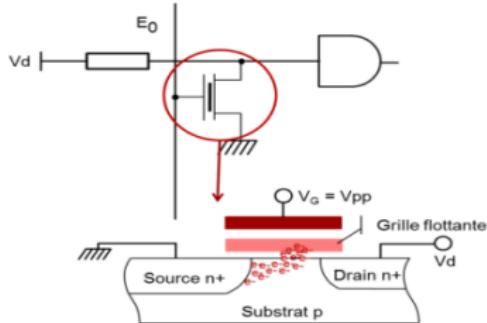


# CIRCUITS INTÉGRÉS

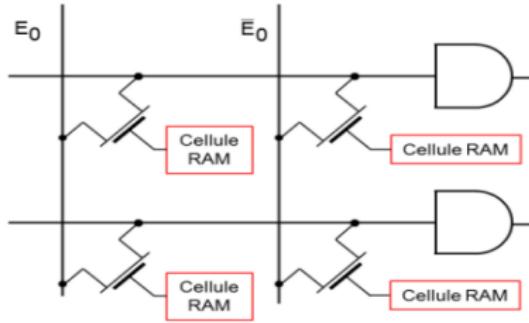
## LES CIRCUITS PROGRAMMABLES

### 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

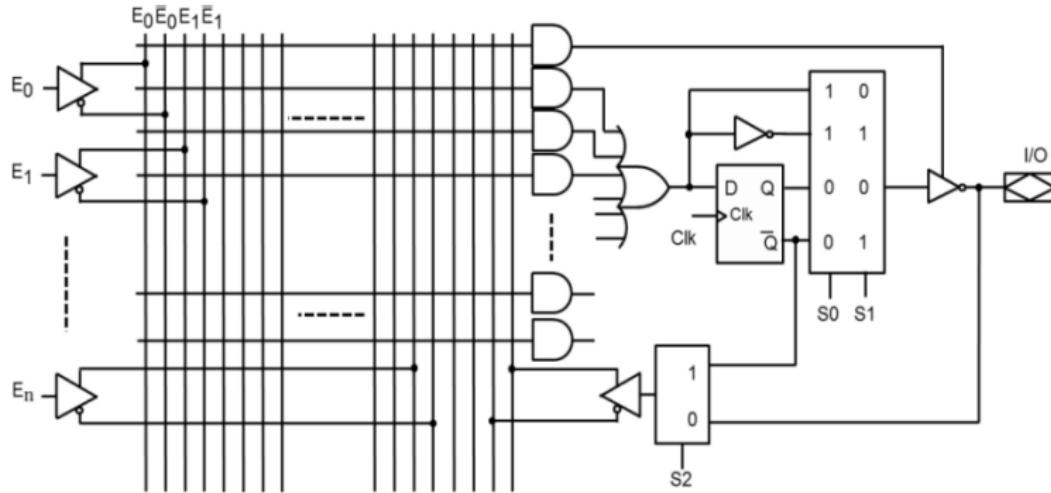


b. PLD SRAM

# CIRCUITS INTÉGRÉS

## LES CIRCUITS PROGRAMMABLES

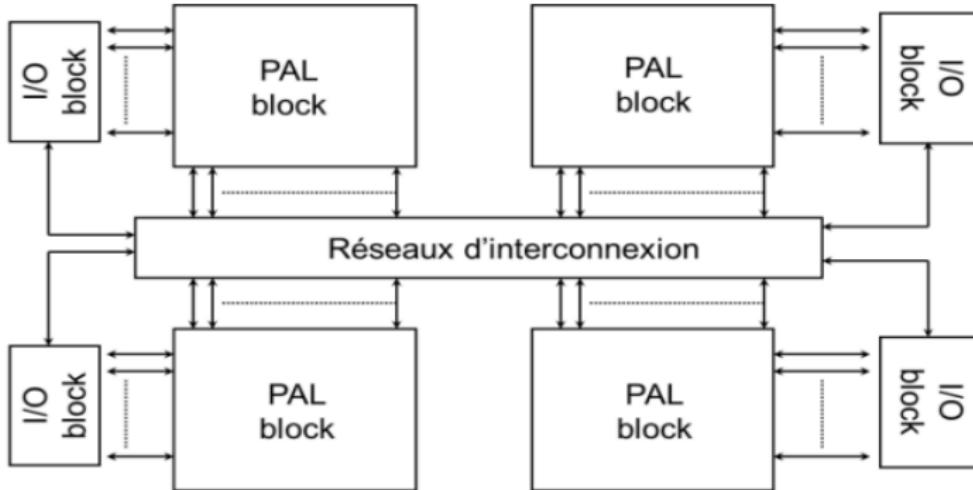
- Pour les circuits séquentiels, une bascule a été rajoutée à la sortie de chaque PAL



# CIRCUITS INTÉGRÉS

## LES CIRCUITS PROGRAMMABLES

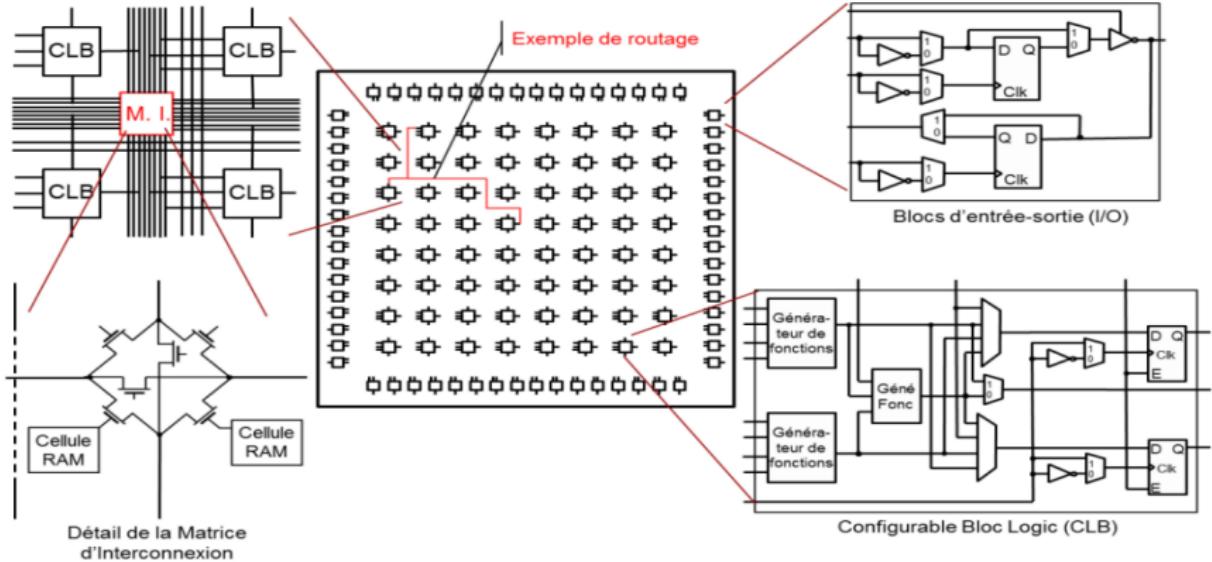
- CPLD - *Complex PLD* sont le regroupement de plusieurs PAL-registre avec un réseau programmable



# 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 simples 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 ?

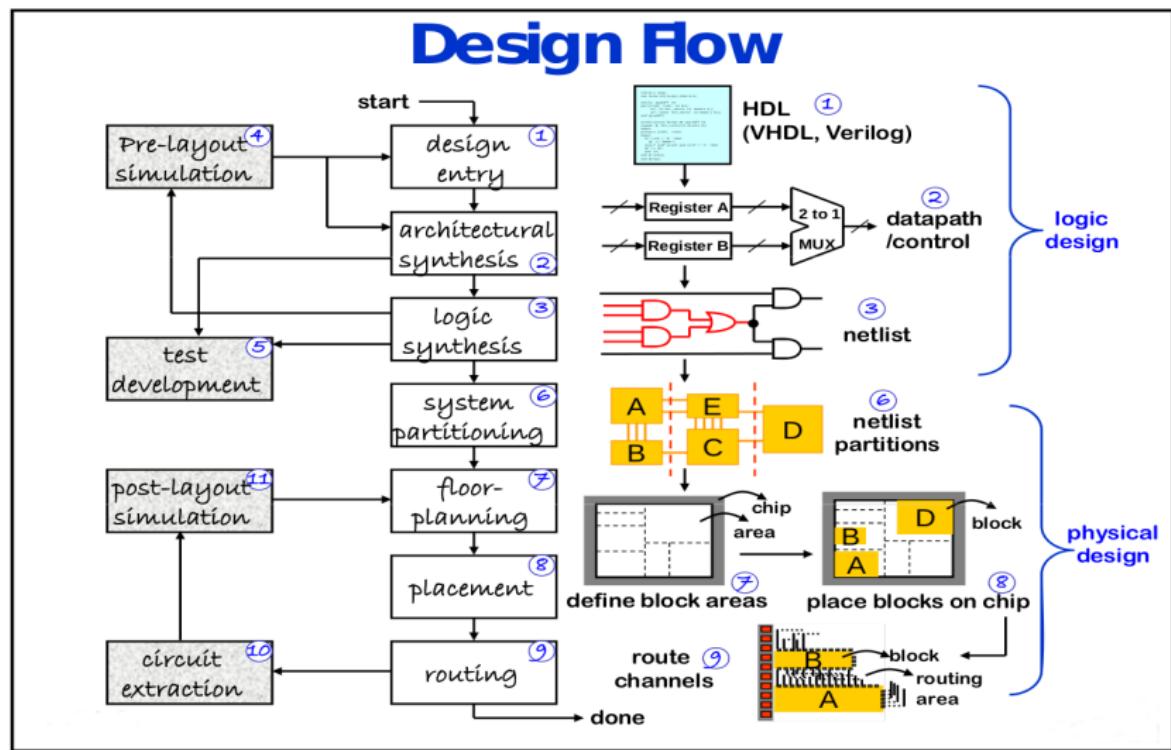
Les méthodes principales de "customisation"

- 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

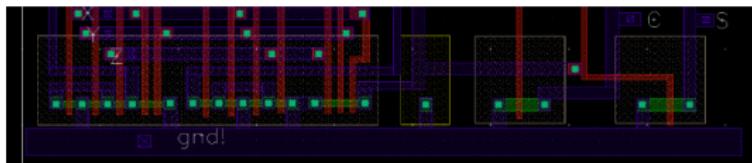
## MODÈLES DE DESCRIPTION

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

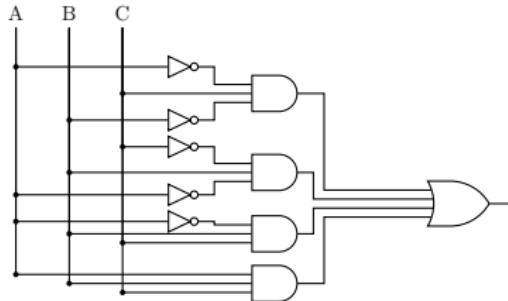
- Comportemental (*behavioral*)
  - ▷ Décrit les fonctionnalités d'un système et ses entrées/sorties (comportement d'un système)
  - ▷ Le système est considéré comme une boîte noire (le « quoi » et non le « comment »)
- Structurel
  - ▷ Décrit l'implémentation interne d'un système (les composants constitutifs 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 



□ Structurel 

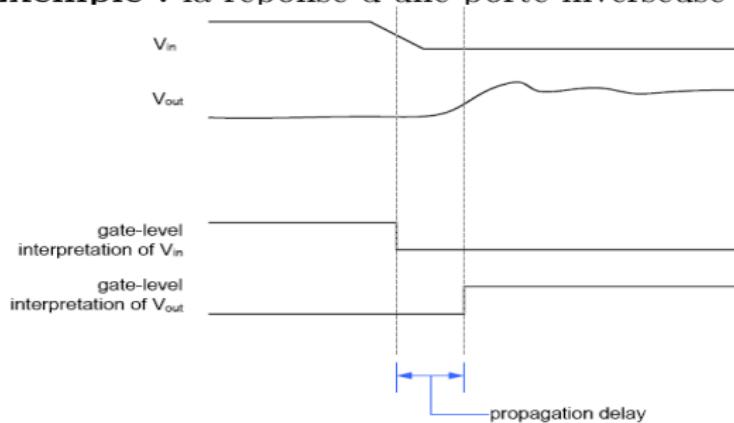


□ Comportemental 

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

# NIVEAUX D'ABSTRACTION

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



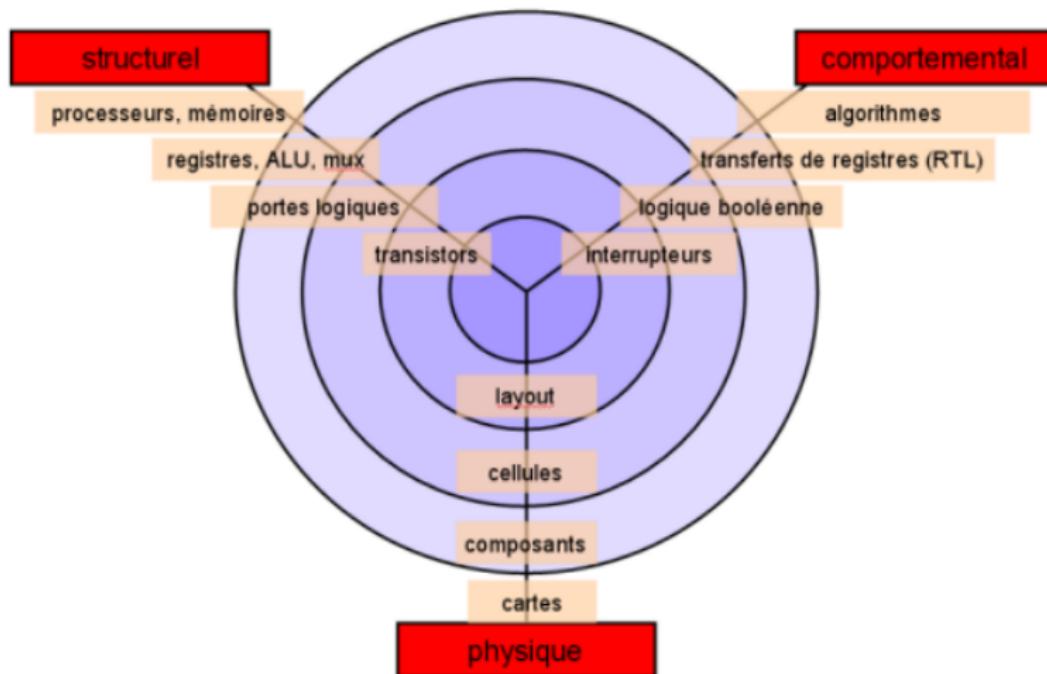
# NIVEAUX D'ABSTRACTION

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

# NIVEAUX D'ABSTRACTION

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

# NIVEAUX D'ABSTRACTION



# LES ÉTAPES DE DÉVELOPPEMENT

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

- Synthèse logique
- Synthèse physique
- Vérification et
- Test

# LES ÉTAPES DE DÉVELOPPEMENT

## SYNTHÈSE LOGIQUE

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

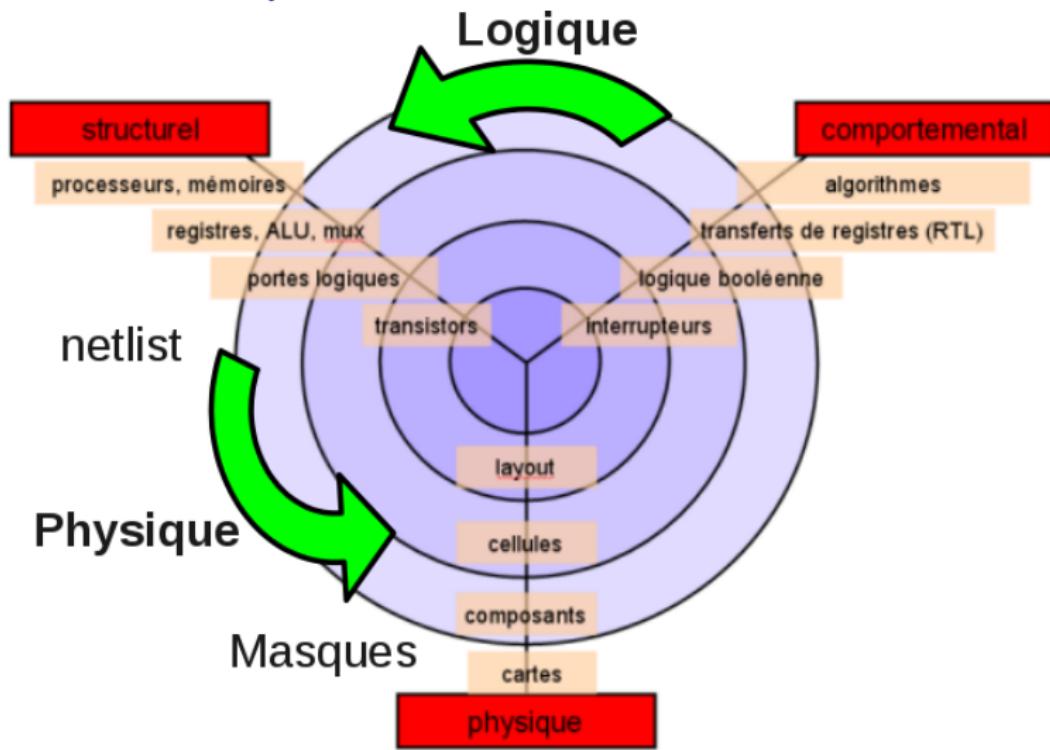
# LES ÉTAPES DE DÉVELOPPEMENT

## SYNTHÈSE PHYSIQUE

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

# LES ÉTAPES DE DÉVELOPPEMENT

## SYNTHÈSE PHYSIQUE



# LES ÉTAPES DE DÉVELOPPEMENT

## VÉRIFICATION

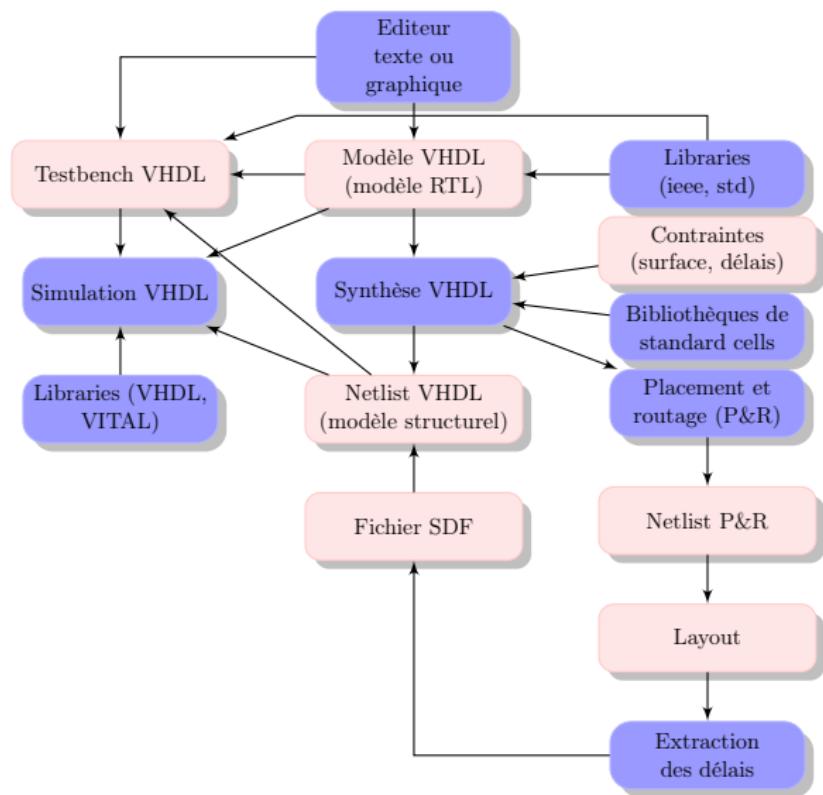
- Vérification
  - ▷ Vérifier si le système conçu respecte le cahier des charges initial
  - ▷ Deux aspects principaux à vérifier :
    - La fonctionnalité et
    - Les performances obtenues : timing (et parfois surface ou énergie)
  - ▷ La vérification est effectuée en appliquant des stimuli aux entrées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

# LES ÉTAPES DE DÉVELOPPEMENT

## FABRICATION ET TEST

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

# FLOT DE CONCEPTION VHDL





# SOMMAIRE

- 2 Hardware Description Language
  - HDL vs PL
  - HDL



# HDL vs PL

## LANGAGE DE DESCRIPTION DE MATÉRIEL VS LANGAGE DE PROGRAMMATION

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



# SOMMAIRE

- 2 Hardware Description Language
  - HDL vs PL
  - HDL

# LANGAGE DE DESCRIPTION DE MATÉRIEL

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



# LE LANGAGE VHDL

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



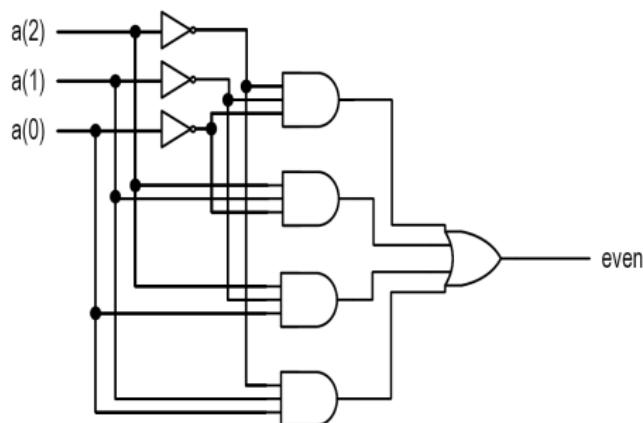
## EXTENSIONS IEEE

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

# VHDL

## EXEMPLE : CIRCUIT DE DÉTECTION DE PARITÉ

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



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

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;

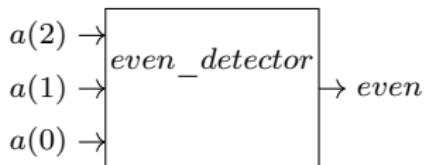
-- déclaration de l'entité
entity even_detector is
    port(
        a: in std_logic_vector(2 downto 0);
        even: out std_logic
    );
end even_detector;

-- 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

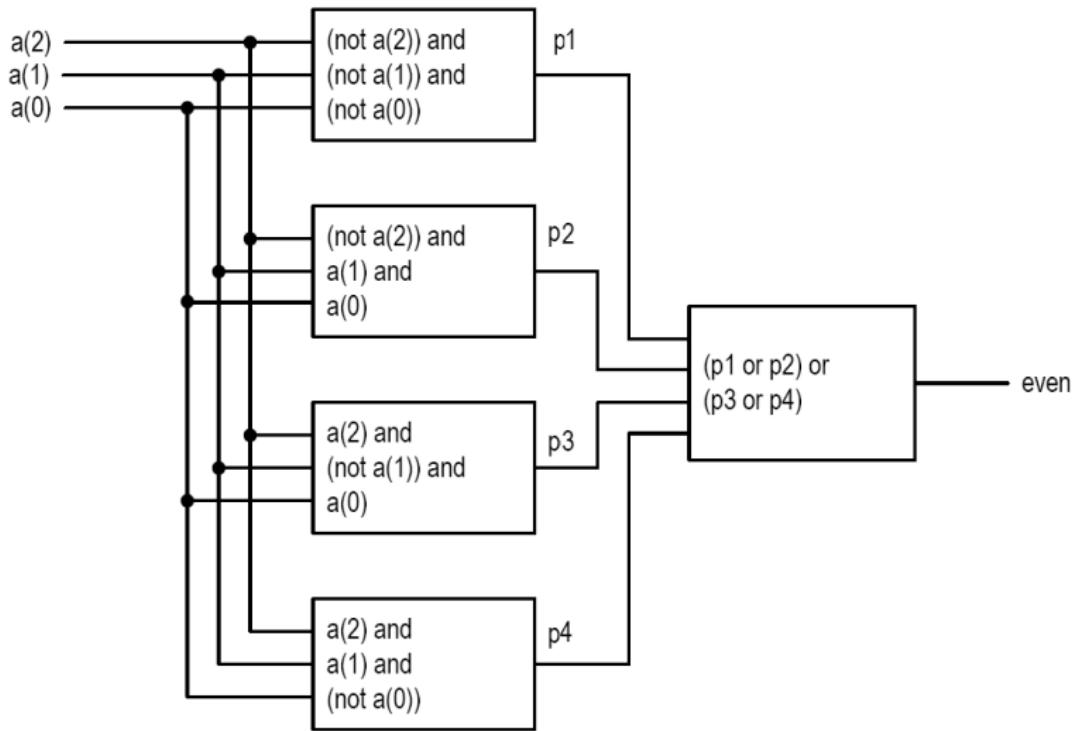
## ARCHITECTURE

- 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

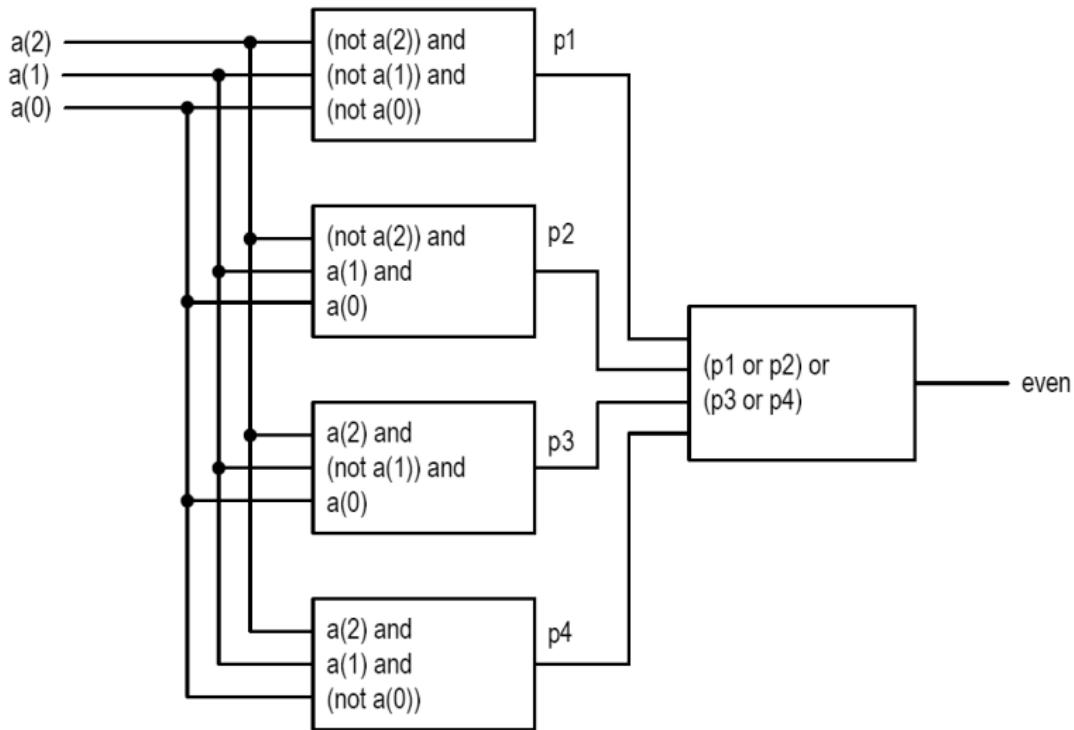
## ARCHITECTURE

- 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

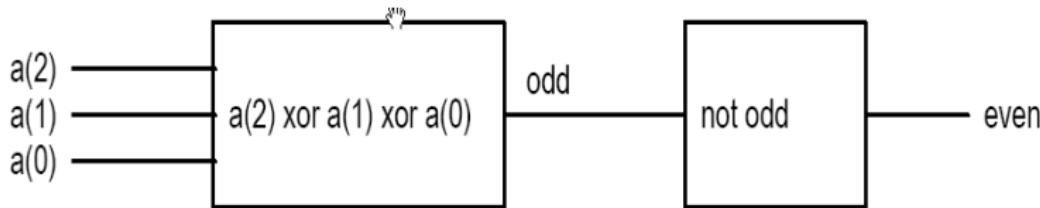
## ARCHITECTURE : INTERPRÉTATION



# VHDL

## ARCHITECTURE

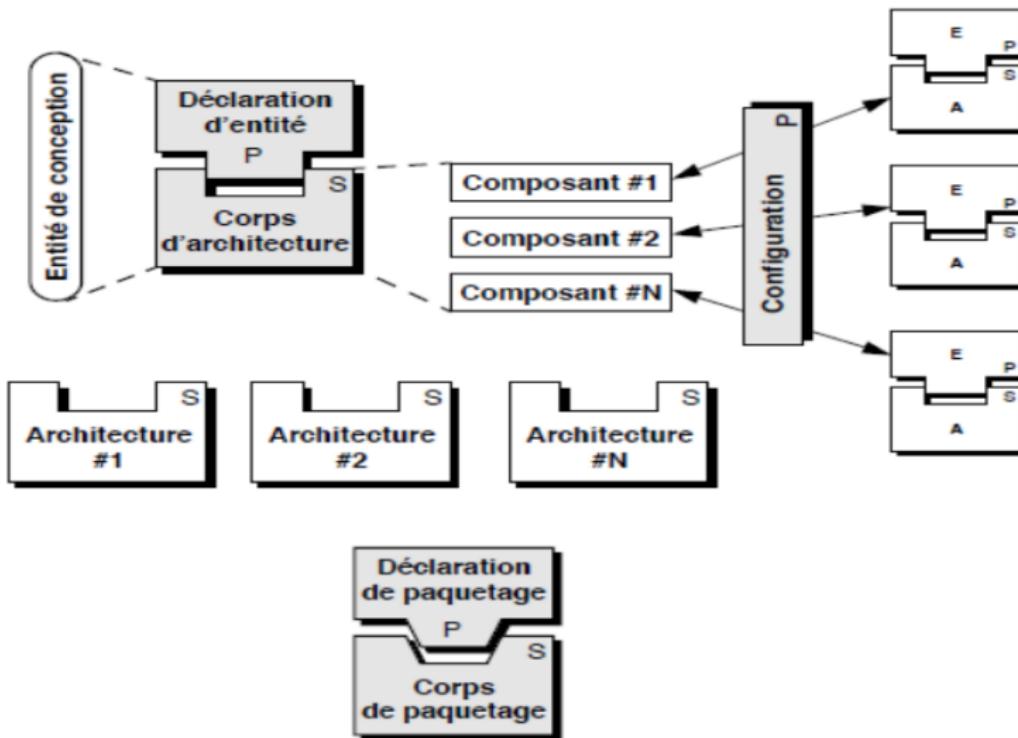
```
1 architecture xor_arch of even_detector is
2     signal odd: std_logic;
3 begin
4     even <= not odd;
5     odd <= a(2) xor a(1) xor a(0);
6 end xor_arch;
```



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

# 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
```





# VHDL

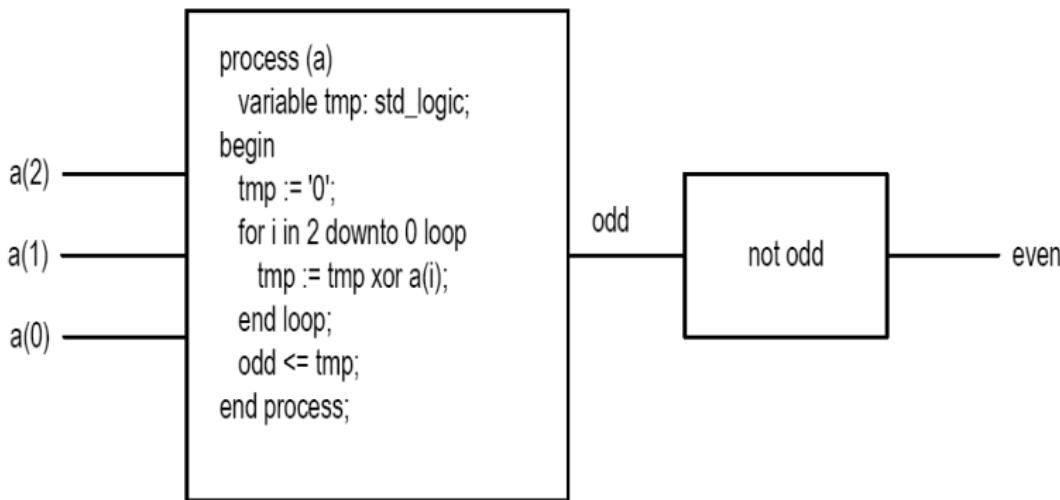
## ARCHITECTURE

```
10      tmp := '0';
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

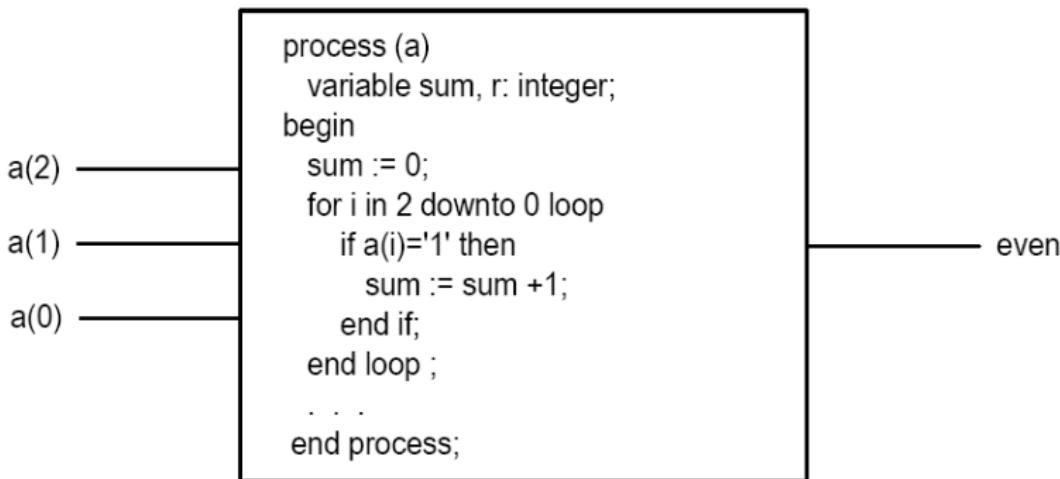
Entité comportementale :

```
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 :





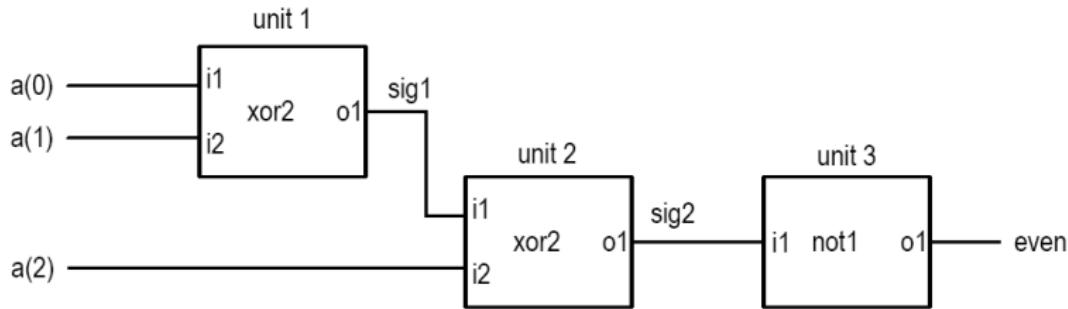
# VHDL

## 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

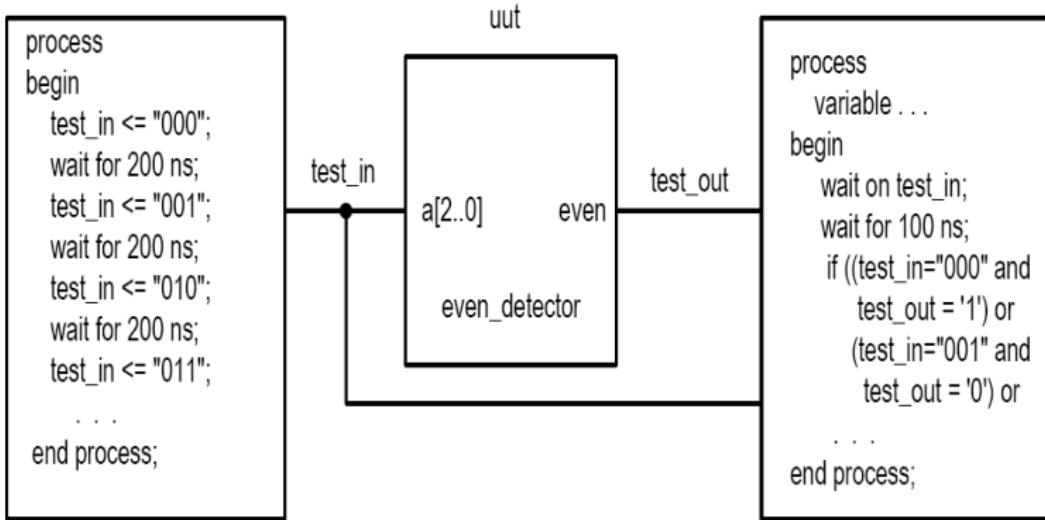
```
20      i1: in std_logic;
21      o1: out std_logic
22  );
23 end not1;
24 architecture beh_arch of not1 is
25 begin
26     o1 <= not i1;
27 end beh_arch;
```



# 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";
```

# VHDL

## TESTBENCH : EXEMPLE

```
39      wait for 200 ns;
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
```



# VHDL

## TESTBENCH : EXEMPLE

```
58      error_status := true;
59  end if;
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

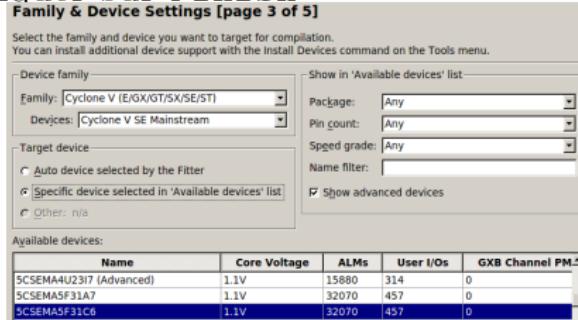
- 3 Utilisation de Quartus
  - Création du projet Quartus
  - Utilisation du flot de conception



# 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

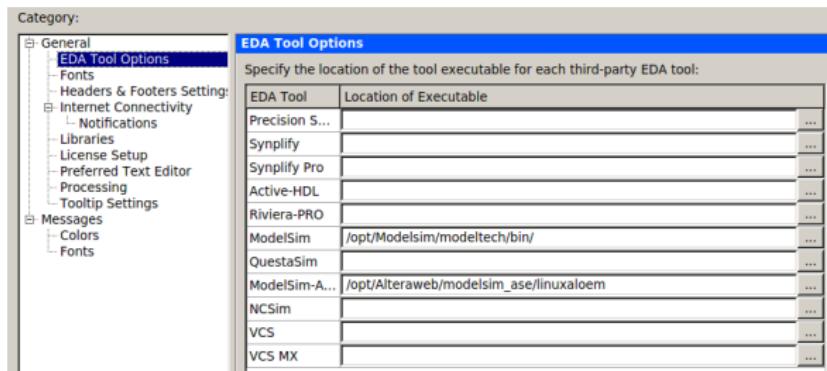




# CONFIGURATION DE QUARTUS

## 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

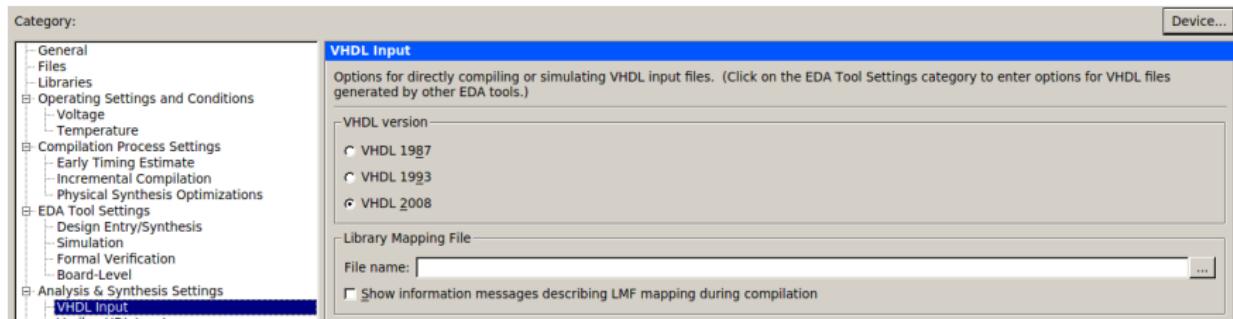




# CONFIGURATION DE QUARTUS

## 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

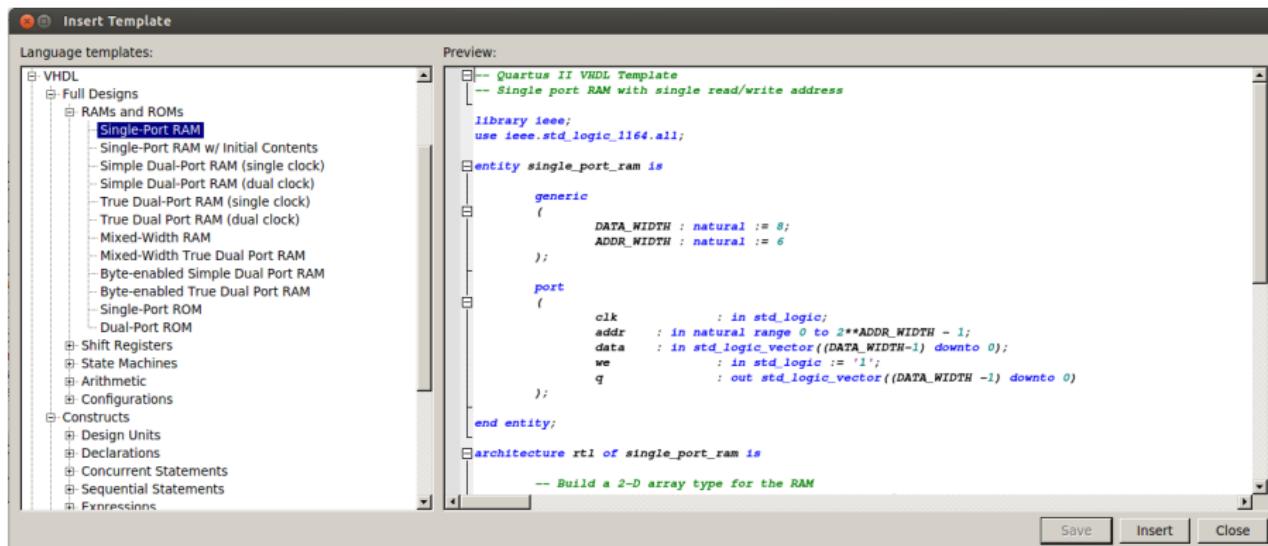




# EDITION DU CODE VHDL

## UTILISATION DES MODÈLES

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





# 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;
```

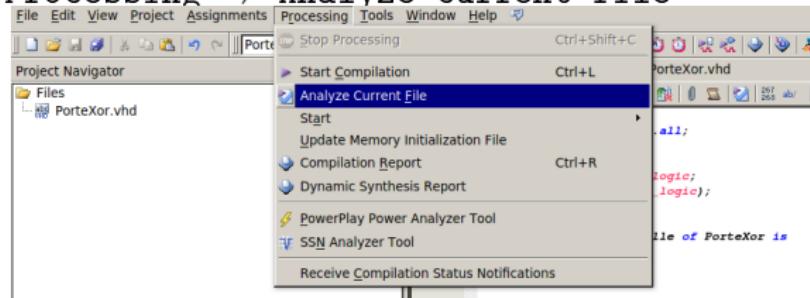


# EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

## FLOT DE CONCEPTION

- Vérifier que le code ne comporte pas d'erreurs de syntaxe

- ▷ Processing ⇒ Analyze current file



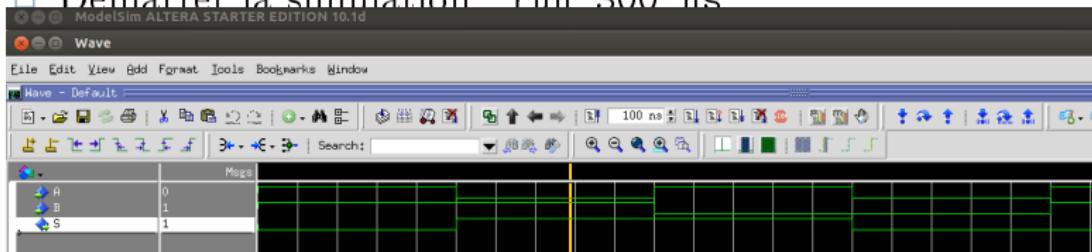
- Pas d'erreurs ⇒ Compiler (Ctrl-L) puis démarrer le simulateur
  - ▷ Tools ⇒ Run Simulation Tool ⇒ RTL Simulation



# 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;

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

    process
```





# EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

## TESTBENCH

```
--variable pour les erreurs
variable errCnt : integer := 0;
begin
    --TEST 1
    inA <= '0';
    inB <= '0';
    wait for 15 ns;
    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
    assert false report "OK!" severity note;
else
    assert true report "KO!" severity error;
end if;

end process;
```



# EXEMPLE TRIVIAL : PORTE OU-EXCLUSIF

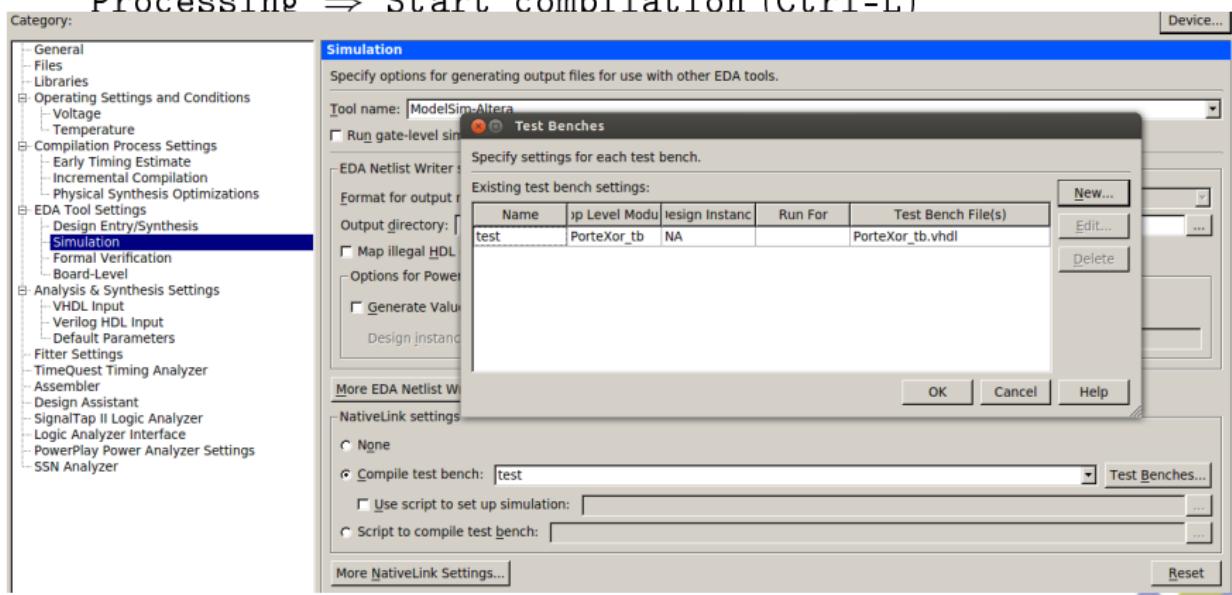
## TESTBENCH

```
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
```



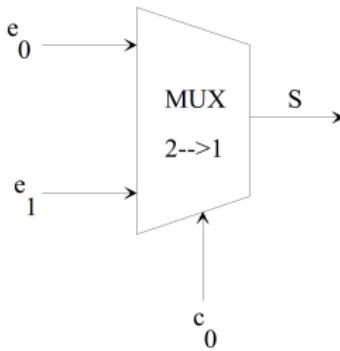
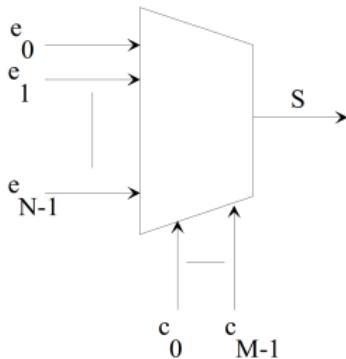
# SOMMAIRE

- 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;

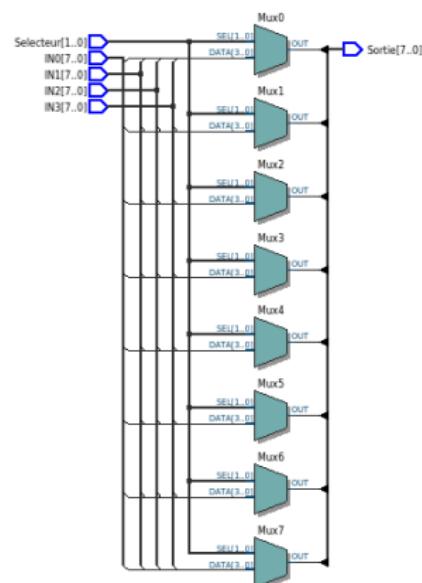
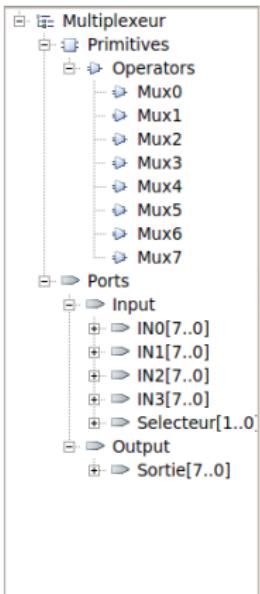
architecture ConcSelect of Multiplexeur is
begin
with Selecteur select
Sortie <= IN0 when "00",
    IN1 when "01",
    IN2 when "10",
    IN3 when "11",
    IN0 when others;
end ConcSelect;
```

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

# LE MULTIPLEXEUR

## VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)

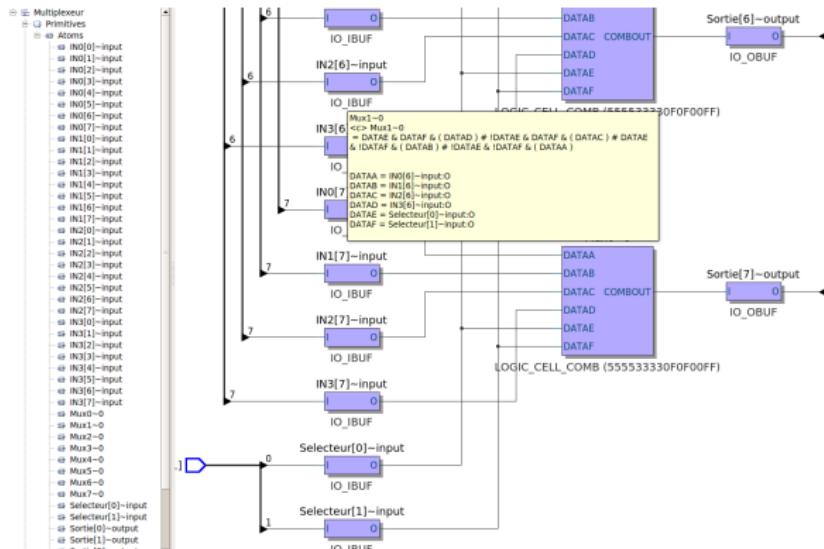
- Après compilation :  
Processing  $\Rightarrow$   
Start compilation  
(Ctrl-L)
- Affichage de la vue  
RTL : Tools  $\Rightarrow$   
Netlist Viewers  
 $\Rightarrow$  RTL viewer
- On note l'utilisation  
d'une primitive  
multiplexeur 1 bit



# 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

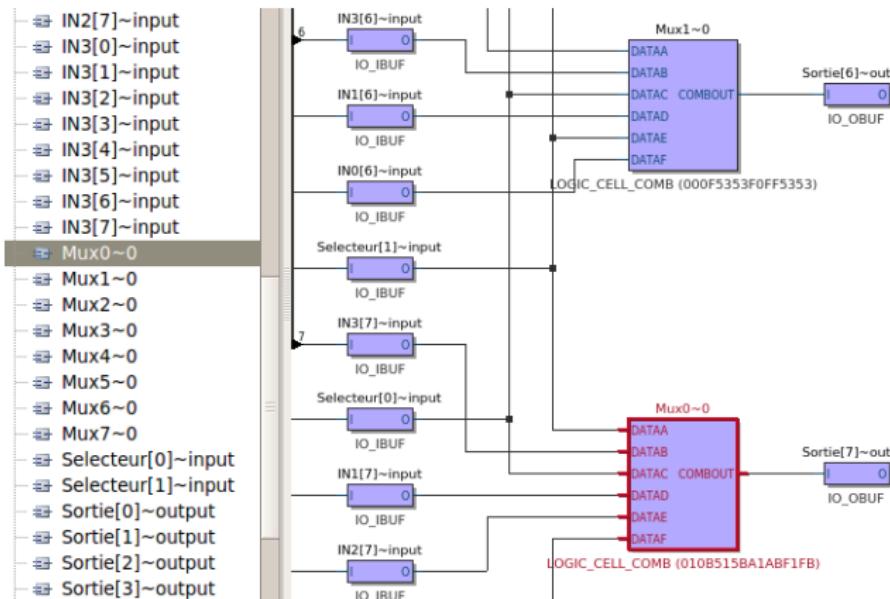
## VUE DU NIVEAU CELLULES FPGA

- 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
            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;
```



# LE MULTIPLEXEUR

## INSTRUCTION case

```
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
        Sortie <= IN0;
    elsif Selecteur = "01" then
        Sortie <= IN1;
    elsif Selecteur = "10" then
        Sortie <= IN2;
    elsif Selecteur = "11" then
        Sortie <= IN3;
```



# LE MULTIPLEXEUR

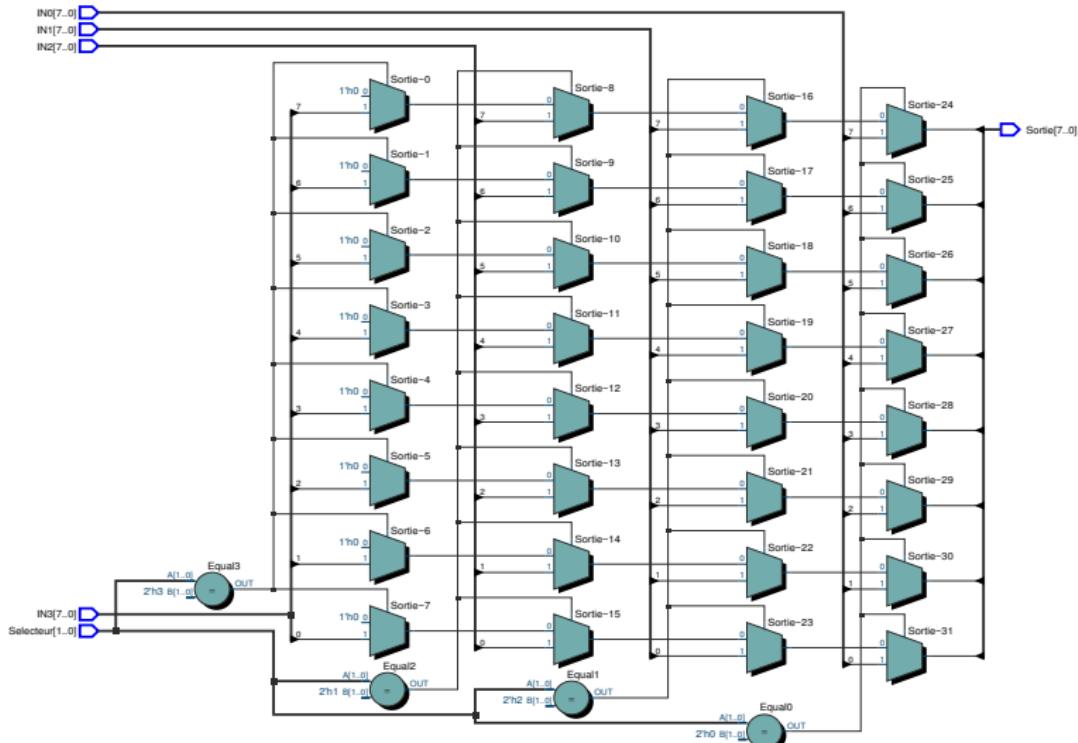
## INSTRUCTION if-end if

```
else
    Sortie <= (others => '0');
end if;
end process;
end ArchIf;
```

- Recompile le design
- 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;

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



# LE MULTIPLEXEUR

## EXEMPLE DE DESCRIPTION GÉNÉRIQUE

```
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);
```



# LE MULTIPLEXEUR

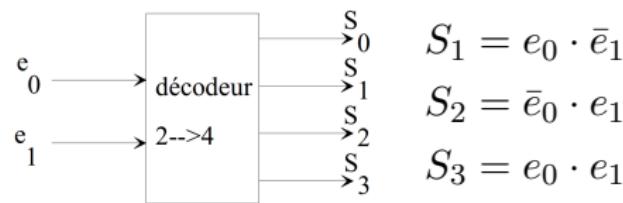
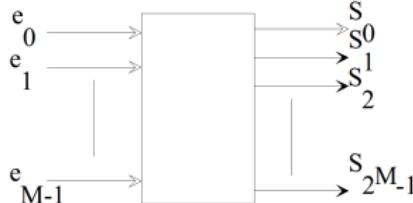
## EXEMPLE DE DESCRIPTION GÉNÉRIQUE

```
Sortie          : out std_logic_vector(TAILLE-1 downto 0));  
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

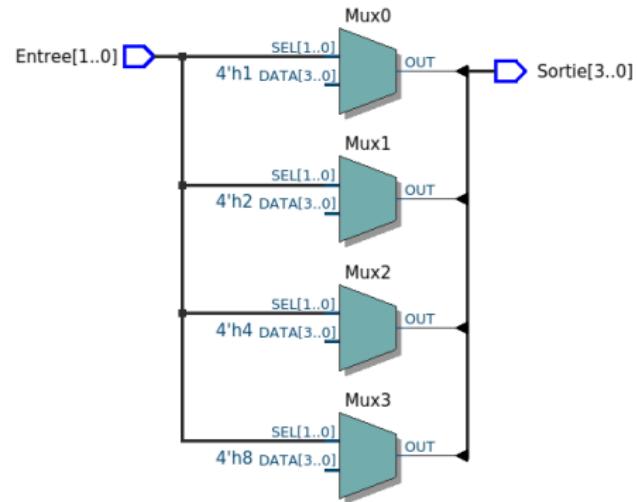
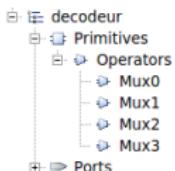
```
"0100" when "10",
"1000" when "11",
"0000" when others;
end ConcSelect;
```

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

# LE DÉCODEUR

## VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)

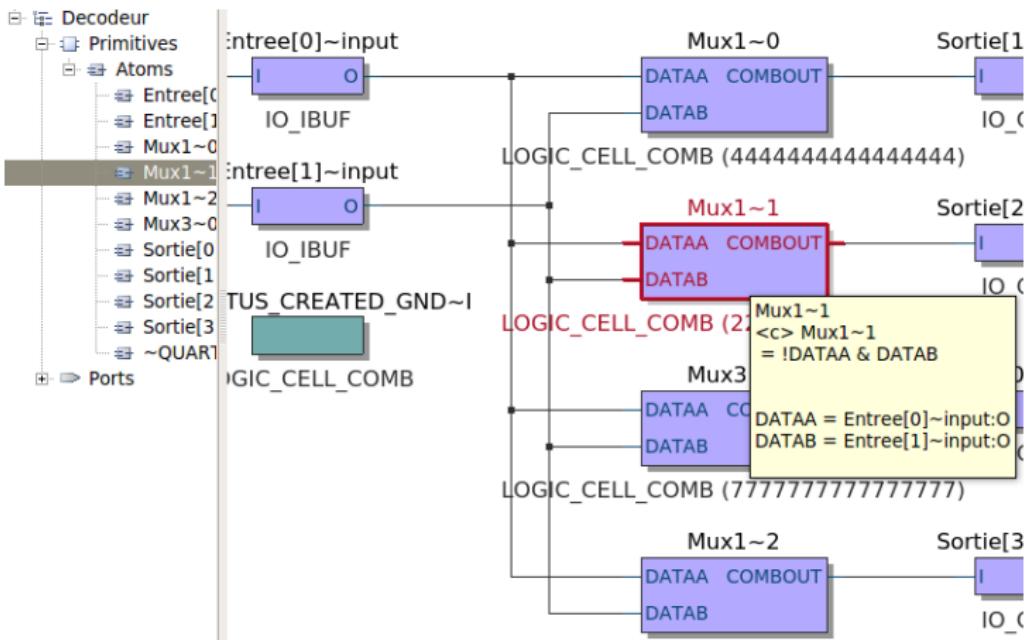
- On constate qu'il y a bien utilisation de multiplexeurs
- On retrouve l'entrée du décodeur comme sélecteur
- L'entrée du multiplexeur devient une constante



# LE DÉCODEUR

## VUE DU NIVEAU CELLULES FPGA

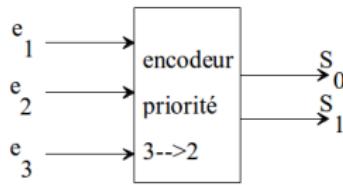
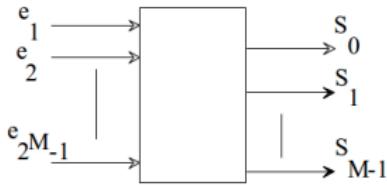
- On voit l'optimisation par rapport à la vue RTL



# 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" ;
    when "01-" => Sortie <= "10" ;
```



# L'ENCODEUR

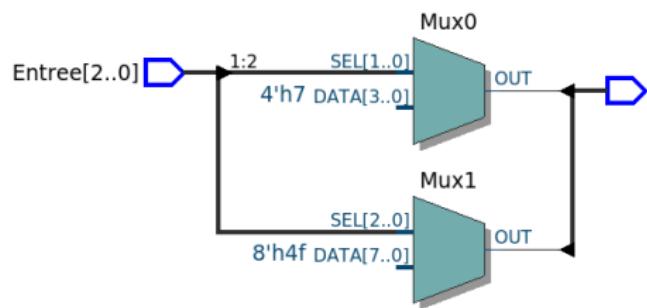
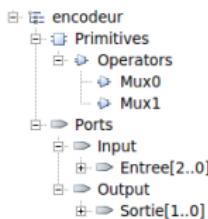
## DESCRIPTION VHDL

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

# L'ENCODEUR

## VUE DU NIVEAU TRANSFERT DE REGISTRES (RTL)

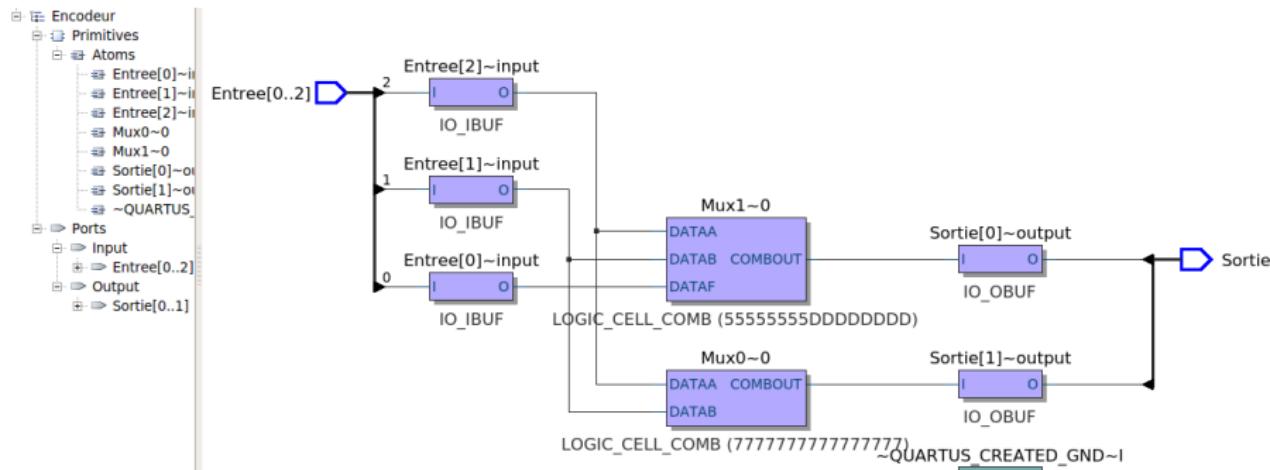
- On constate à nouveau l'utilisation de multiplexeurs
- On retrouve l'entrée de l'encodeur comme sélecteur
- L'entrée du multiplexeur devient une constante



# L'ENCODEUR

## VUE DU NIVEAU CELLULES FPGA

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





# EXERCICES

- ① Reprendre le multiplexeur et le modifier pour qu'il soit de type  $5 \rightarrow 1$ . Utiliser un **process**
- ② Que se passe-t-il si l'on n'utilise pas le cas **others** ?
- ③ Reprendre le décodeur et le décrire en utilisant un **process**
- ④ Reprendre l'encodeur et le décrire en utilisant uniquement des instructions **concurrentes**



# 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**



# EXERCICES

## SOLUTION QUESTION 1

```
entity Multiplexeur is
port (
    in0, in1, in2, in3, in4 : in std_logic_vector(7
downto 0);
    Selecteur : in std_logic_vector(2
downto 0);
    Sortie : out std_logic_vector(7
downto 0));
end Multiplexeur;

architecture CaseProcess of Multiplexeur is
begin
process(all)
```



# EXERCICES

## SOLUTION QUESTION 1

```
begin
    case Selecteur is
        when "000" => Sortie <= in0;
        when "001" => Sortie <= in1;
        when "010" => Sortie <= in2;
        when "011" => Sortie <= in3;
        when "100" => Sortie <= in4;
        when others => Sortie <= (others => '0');
    end case;
end process;
end CaseProcess;
```



# EXERCICES

## SOLUTION QUESTION 3

```
entity Decodeur is
port(
    Entrée : in std_logic_vector(1 downto 0);
    Sortie : out std_logic_vector(3 downto 0));
end Decodeur;
architecture ConcSelect of Decodeur is
begin
process(all)
begin
    case Entrée is
        when "00"      => Sortie <= "0001";
        when "01"      => Sortie <= "0010";
        when "10"      => Sortie <= "0100";
        when "11"      => Sortie <= "1000";
```



# EXERCICES

## SOLUTION QUESTION 3

```
    when others  => Sortie <= "0000";
end process;
end ConcSelect;
```



# EXERCICES

## SOLUTION QUESTION 4

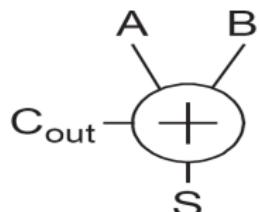
```
entity Encodeur is
port(
    Entree : in  std_logic_vector(2 downto 0);
    Sortie : out std_logic_vector(1 downto 0));
end Encodeur;

architecture ProConc of Encodeur is begin
    Sortie(0) <= Entree(2) or (not Entree(2) and not
    Entree(1) and Entree(0));
    Sortie(1) <= Entree(2) or Entree(1);
end ProConc;
```

# ADDITIONNEUR

## FONCTION ET SPÉCIFICATIONS

### Half adder



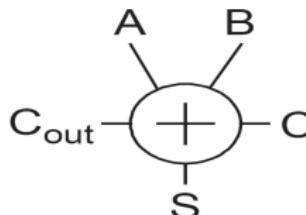
A	B	$C_{OUT}$	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

$$G = A \cdot B$$

$$P = A \oplus B$$

$$K = \overline{A} \cdot \overline{B}$$

### Full adder



A	B	C	G	P	K	$C_{OUT}$	S
0	0	0	0	0	1	0	0
		1				0	1
0	1	0	0	1	0	0	1
		1				1	0
1	0	0	0	1	0	0	1
		1				1	0
1	1	0	1	0	0	1	0
		1				1	1



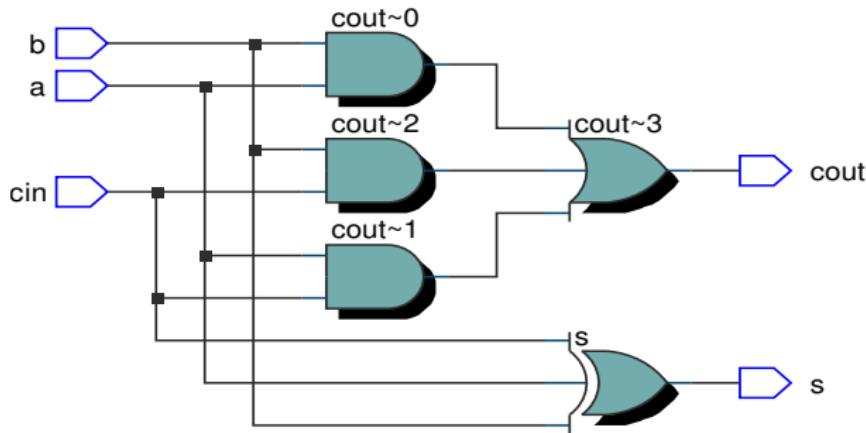
# ADDITIONNEUR

## DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity additionneur is
port(
    a,b,cin  : in  std_logic;
    s,cout    : out std_logic);
end additionneur;
--end;
architecture archConc of additionneur is
begin
    s    <= a xor b xor cin;
    cout <= (a and b) or (a and cin) or (b and cin);
end archConc;
```

# ADDITIONNEUR

## VUE DU NIVEAU TRANSFERT DE REGISTRE (RTL)



# SOUSTRACTEUR

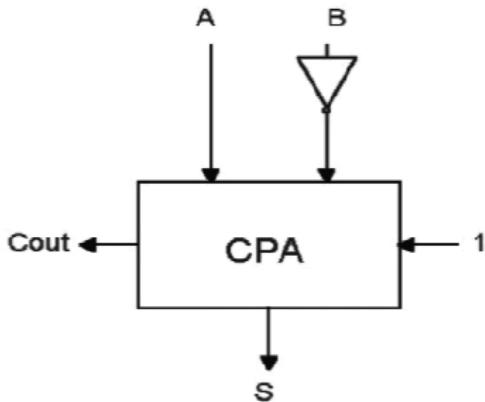
Principe :

- Complémenter à 2 l'opérande à soustraire et réaliser une addition du résultat avec l'autre opérande

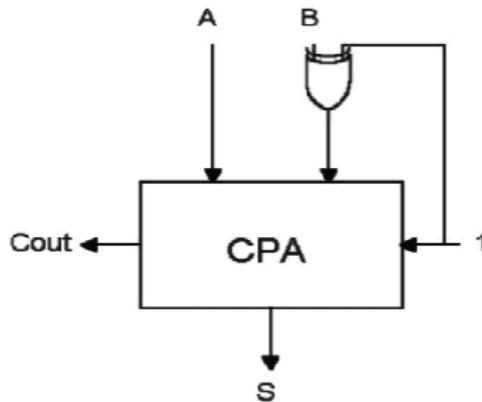
Exemple :

Soustracteur

$$A - B = A + (-B) = A + \overline{B} + 1$$



Additionneur/soustracteur  
mode addition :  $sub = 0$   
mode soustraction :  $sub = 1$   
 $A \pm B = A + (B \oplus sub) + sub$





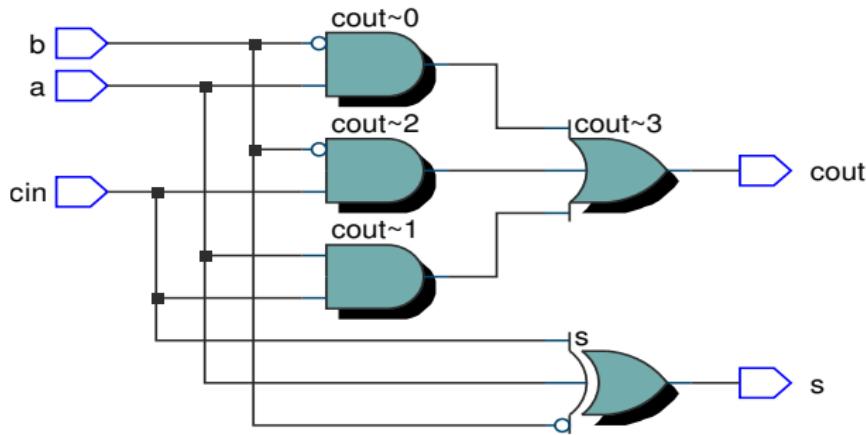
# SOUSTRACTEUR

## DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity soustracteur is
port(
    a,b,cin  : in  std_logic;
    s,cout    : out std_logic);
end soustracteur;
--end;
architecture archConc of soustracteur is
    signal bn :std_logic;
begin
    bn  <= not b;
    s   <= a xor bn xor cin;
    cout <= (a and bn) or (a and cin) or (bn and cin);
end archConc;
```

# SOUSTRACTEUR

## VUE DU NIVEAU DE TRANSFERT DE REGISTRES (RTL)



# COMPARATEUR

## FONCTIONALITÉ ET SPÉCIFICATIONS

### Principe :

- Calculer si  $A = B$  ou si  $A \geq B$  permet de déterminer les autres résultats de comparaison

$$EQ = (A = B) \quad NE = (A \neq B) = \overline{EQ} \quad GT = (A > B) = \overline{EQ} \cdot GE$$

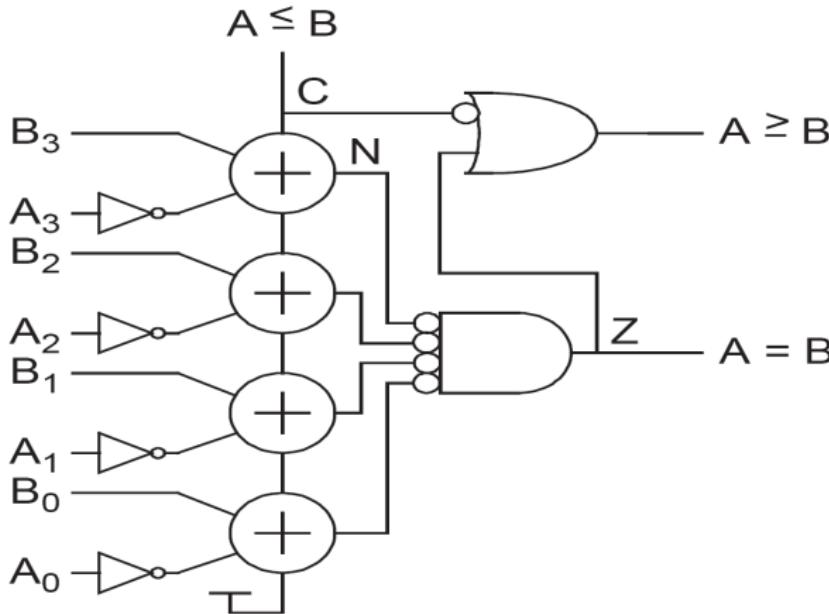
$$GE = (A \geq B) \quad LT = (A < B) = \overline{GE} \quad LE = (A \leq B) = \overline{GE} + EQ$$

- $EQ = (A = B) = (B - A = 0)$
- $EQ_{i+1} = (A_i = B_i) \cdot EQ_i = \overline{(A_i \oplus B_i)} \cdot EQ_i$
- $EQ_0 = 1; EQ_n = EQ$  ou utiliser un soustracteur optimisé (sans sortie  $S_i$ )
- $GE = (A \geq B) = (A - B \geq 0)$
- $GE_{i+1} = (A_i > B_i) + (A_i = B_i) \cdot GE_i = A_i \cdot \overline{B_i} + \overline{(A_i \oplus B_i)} \cdot GE_i$
- $GE_0 = 1; GE_n = GE$  ou utiliser un soustracteur

# COMPARATEUR

## FONCTIONALITÉ ET SPÉCIFICATIONS

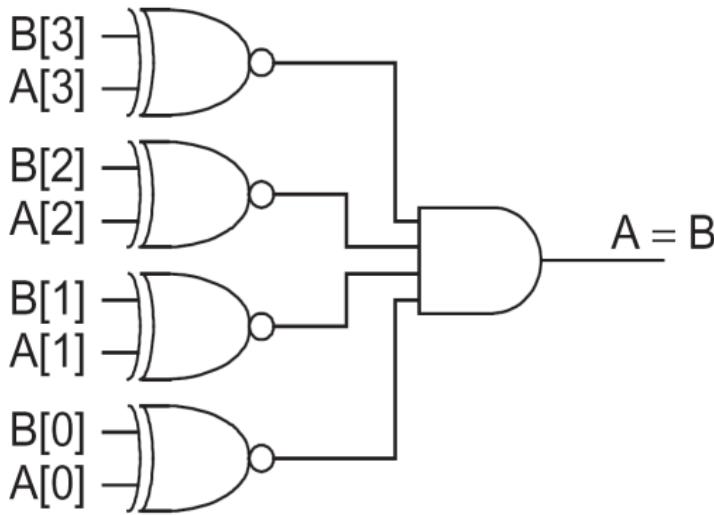
- Utilisation d'un soustracteur modifié



# COMPARATEUR

## FONCTIONALITÉ ET SPÉCIFICATIONS

- Si on veut calculer uniquement si  $A=B$ , le circuit est plus simple





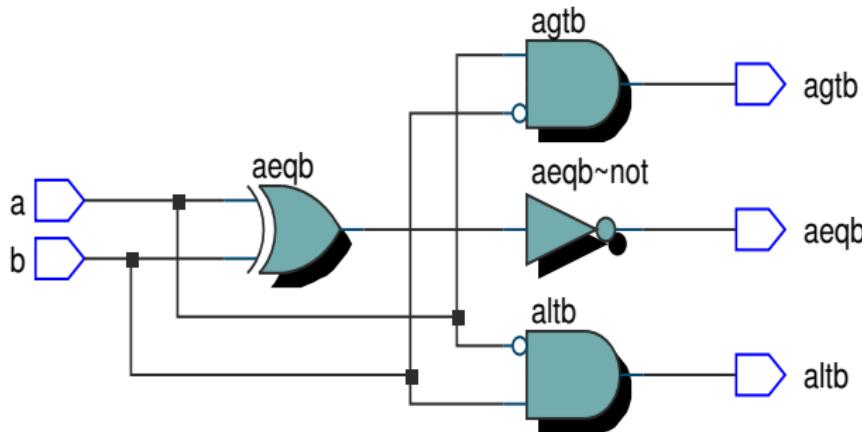
# COMPARATEUR

## DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity Comparateur is
port(
    a,b           : in std_logic;
    agtb, altb, aeqb : out std_logic);
end Comparateur;
architecture archConc of Comparateur is
begin
    agtb <= '1' when a > b else '0';
    altb <= '1' when a < b else '0';
    aeqb <= '1' when a = b else '0';
end archConc;
```

# COMPARATEUR

## DESCRIPTION VHDL





# COMPARATEUR

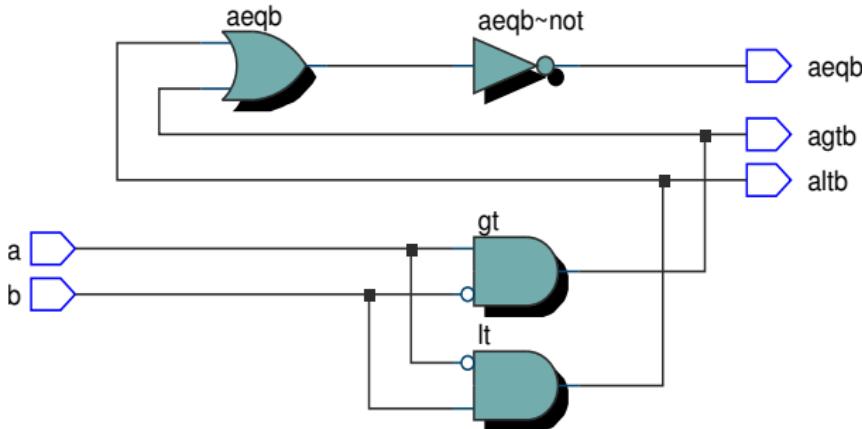
## DESCRIPTION VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity Comparateur is
port(
    a,b           : in std_logic;
    agtb, altb, aeqb : out std_logic);
end Comparateur;
architecture archConc1 of Comparateur is
    signal gt, lt :std_logic;
begin
    gt <= '1' when a > b else '0';
    lt <= '1' when a < b else '0';
    agtb <= gt;
```

# COMPARATEUR

## DESCRIPTION VHDL

```
altb <= lt;  
aeqb <= not (gt or lt);  
end archConc1;
```





# COMPARATEUR

## DESCRIPTION VHDL

- Quelles sont les différences au niveau RTL entre ces deux descriptions VHDL ?
- A votre avis, laquelle des deux est la plus rapide ?



# SOMMAIRE

## 4 VHDL : utilisation du langage

- Description des fonctions combinatoires usuelles
- Types de données

# TYPES DE DONNÉES

- Définition d'un type de données
  - ▷ un ensemble de valeurs pouvant être affectées à un objet
  - ▷ un ensemble d'opérations pouvant être appliquées sur les objets d'un même type
- VHDL est un langage très typé
- un objet peut être affecté uniquement avec la valeur du même type
- uniquement les opérations définies pour un type de données peuvent être appliquées sur un objet de même type

Types de données standard :

- entier (**integer**) :
  - ▷ de  $-(2^{31} - 1)$  à  $2^{31} - 1$  par défaut (au maximum)
  - ▷ deux sous-types : **natural** et **positive**



# TYPES DE DONNÉES

- ▷ possibilité de spécifier un domaine avec `range` :  
`variable UnEntier : integer range 0 to 511;`
- ▷ possibilité d'accéder aux limites du domaine avec des attributs :  
`signal limB,limH : natural;`  
`begin limB <= UnEntier'Left; limH <=`  
`UnEntier'Right;`  
⇒ limB vaut 0 et limH vaut 511
- booléen (`boolean`) : (`false`, `true`)
- `bit` : ('0', '1')
- `bit_vector` : un tableau 1D de bits

# TYPES DE DONNÉES

operator	description	data type of operand a	data type of operand b	data type of result
$a^{**} b$	exponentiation	integer	integer	integer
$\text{abs } a$	absolute value	integer		integer
$\text{not } a$	negation	boolean, bit, bit_vector		boolean, bit, bit_vector
$a * b$	multiplication	integer	integer	integer
$a / b$	division			
$a \bmod b$	modulo			
$a \bmod b$	remainder			
$+ a$	identity	integer		integer
$- a$	negation			
$a + b$	addition	integer	integer	integer
$a - b$	subtraction			
$a \& b$	concatenation	1-D array, element	1-D array, element	1-D array

# TYPES DE DONNÉES

a <b>sll</b> b	shift left logical	bit_vector	integer	bit_vector
a <b>srl</b> b	shift right logical			
a <b>sla</b> b	shift left arithmetic			
a <b>sra</b> b	shift right arithmetic			
a <b>rol</b> b	rotate left			
a <b>ror</b> b	rotate right			
<hr/>				
a = b	equal to	any	same as a	boolean
a /= b	not equal to			
a < b	less than	scalar or 1-D array	same as a	boolean
a <= b	less than or equal to			
a > b	greater than			
a >= b	greater than or equal to			
<hr/>				
a <b>and</b> b	and	boolean, bit,	same as a	boolean, bit,
a <b>or</b> b	or	bit_vector		bit_vector
a <b>xor</b> b	xor			
a <b>nand</b> b	nand			
a <b>nor</b> b	nor			
a <b>xnor</b> b	xnor			

# TYPES DE DONNÉES

## std\_logic

- Pourquoi le type **bit** n'est pas suffisant ?
- le package **std\_logic\_1164**
- **std\_logic** :
- 9 valeurs possibles :
  - ▷ '0' ou '1'
  - ▷ 'Z' : l'état de haute impédance
  - ▷ 'L' ou 'H' : un faible '0' ou '1'
  - ▷ 'X', 'W' : inconnu ou un faible inconnu
  - ▷ 'U' : non initialisé
  - ▷ '-' : peu importe (indéfini)
- **std\_logic\_vector** : vecteur de **std\_logic**  
→ **std\_logic\_vector(7 downto 0)**



# TYPES DE DONNÉES

`std_logic`

- utilisation :

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

Opérateurs utilisés avec le type `std_logic`

# TYPES DE DONNÉES

std\_logic

<b>overloaded operator</b>	<b>data type of operand a</b>	<b>data type of operand b</b>	<b>data type of result</b>
<b>not a</b>	std_logic_vector std_logic		same as a
<b>a and b</b>			
<b>a or b</b>			
<b>a xor b</b>	std_logic_vector	same as a	same as a
<b>a nand b</b>	std_logic		
<b>a nor b</b>			
<b>a xnor b</b>			

Les fonctions de conversion disponibles dans le package :

# TYPES DE DONNÉES

std\_logic

---

function 	data type of operand a	data type of result
to_bit(a)	std_logic	bit
to_stdulogic(a)	bit	std_logic
to_bit_vector(a)	std_logic_vector	bit_vector
to_stdlogicvector(a)	bit_vector	std_logic_vector

---

Exemple :



# TYPES DE DONNÉES

std\_logic

```
signal s1, s2, s3: std_logic_vector(7 downto 0);
signal b1, b2      : bit_vector(7 downto 0);

--KO
s1 <= b1;
b2 <= s1 and s2;
s3 <= b1 or s2;
--OK
s1 <= to_stdlogicvector(b1);
b2 <= to_bitvector(s1 and s2);
s3 <= to_stdlogicvector(b1) or s2;
-- ou
s3 <= to_stdlogicvector(b1 or to_bitvector(s2));
```



# TYPES DE DONNÉES

std\_logic

# OPÉRATEURS SUR LES TABLEAUX D'ÉLÉMENTS

- Les opérandes n'ont pas toujours la même taille
- Lors de la comparaison de deux tableaux n'ayant pas la même taille, tous les éléments sont comparés un par un en partant de gauche (MSB)
- Exemple :

"011"="011", "011">>"010", "011">>"00010", "0110">>"011"

Tous les exemples précédents sont vrais

- Opérateur de concaténation

```
y<= "00" & a(7 downto 2);
```

```
y<= a(7) & a(7) & a(7 downto 2);
```

```
y<= a(1 downto 0) & a(7 downto 2);
```



# OPÉRATEURS SUR LES TABLEAUX D'ÉLÉMENTS

## □ Opération d'agrégation

```
a <= "10100000";
a <= (7=> '1', 6=> '0', 0=> '0', 1=> '0', 5=> '1',
      4=> '0', 3=> '0', 2=> '1');
a <= (7|5=> '1', 6|4|3|2|1|0 => '0');
a <= (7|5=> '1', others => '0');
a <= "00000000";
a <= (others => '0');
```

# TYPES DE DONNÉES

## PACKAGE numeric\_std

- Comment réaliser les opérations arithmétiques avec les std\_logic ?
- la solution : le package numeric\_std
- définit un entier comme un tableau d'éléments de type std\_logic
- Deux types : unsigned et signed
- utilisation :

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

# TYPES DE DONNÉES

## PACKAGE numeric\_std

Les opérateurs définis dans le package :

overloaded operator	description	data type of operand a	data type of operand b	data type of result
<b>abs</b> a	absolute value	signed		signed
- a	negation			
a * b				
a / b		unsigned	unsigned, natural	unsigned
a <b>mod</b> b	arithmetic operation	unsigned, natural	unsigned	unsigned
a <b>rem</b> b		signed	signed, integer	signed
a + b		signed, integer	signed	signed
a - b				
a = b				
a /= b		unsigned	unsigned, natural	boolean
a < b	relational	unsigned, natural	unsigned	boolean
a <= b		signed	signed, integer	boolean
a > b		signed, integer	signed	boolean
a >= b				



# TYPES DE DONNÉES

PACKAGE numeric\_std

```
signal a,b,c,d: unsigned (7 downto 0);
...
a <= b + c;
d <= b + 1;
e <= (5 + a + b) - c;
```

# TYPES DE DONNÉES

## PACKAGE numeric\_std

function	description	data type of operand a	data type of operand b	data type of result
shift_left(a,b)	shift left	unsigned, signed	natural	same as a
shift_right(a,b)	shift right			
rotate_left(a,b)	rotate left			
rotate_right(a,b)	rotate right			
resize(a,b)	resize array	unsigned, signed	natural	same as a
std_match(a,b)	compare '-'	unsigned, signed std_logic_vector, std_logic	same as a	boolean
to_integer(a)	data type conversion	unsigned, signed		integer
to_unsigned(a,b)		natural	natural	unsigned
to_signed(a,b)		integer	natural	signed



# TYPES DE DONNÉES

## PACKAGE numeric\_std

- Les types de données `std_logic_vector`, `unsigned` ou `signed` sont définis comme des tableaux d'éléments `std_logic`
- Ces trois types sont considérés comme des types différents
- Utilisation de fonctions de conversion pour passer d'un type à un autre

# TYPES DE DONNÉES

## PACKAGE numeric\_std

data type of a	to data type	conversion function / type casting
unsigned, signed	std_logic_vector	std_logic_vector(a)
unsigned, std_logic_vector	unsigned	unsigned(a)
unsigned, signed	std_logic_vector	std_logic_vector(a)
unsigned, signed	integer	to_integer(a)
natural	unsigned	to_unsigned(a, size)
integer	signed	to_signed(a, size)

# TYPES DE DONNÉES

## PACKAGE numeric\_std

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
.
.
signal s1, s2, s3, s4, s5, s6: std_logic_vector(3
downto 0);
signal u1, u2, u3, u4, u6, u7: unsigned(3 downto 0);
signal sg: signed(3 downto 0);

-- OK
u3 <= u2 + u1;    --- ok, operandes non-signés
u4 <= u2 + 1;    --- ok, operandes non-signé et
natural

--KO
```

# TYPES DE DONNÉES

## PACKAGE numeric\_std

```
u5 <= sg;    -- type mismatch
u6 <= 5;     -- type mismatch
```

--Solution

```
u5 <= unsigned(sg);      -- type casting
u6 <= to_unsigned(5,4);  -- fonction de conversion
```

--KO

```
s3 <= u3;    -- type mismatch
s4 <= 5;     -- type mismatch
```

--Solution

```
s3 <= std_logic_vector(u3); -- type casting
s4 <= std_logic_vector(to_unsigned(5,4));
```



# TYPES DE DONNÉES

## PACKAGE numeric\_std

```
--KO
s5 <= s2 + s1; -- addition indefinie
s6 <= s2 + 1; -- addition indefinie

-- Solution
s5 <= std_logic_vector(unsigned(s2) + unsigned(s1));
s6 <= std_logic_vector(unsigned(s2) + 1);
```



# TYPES DE DONNÉES

## PACKAGE std\_logic\_arith

- package développé par *Synopsys* avant le standard IEEE numeric\_std
- presque similaire à numeric\_std
- deux nouveaux types : unsigned et signed
- les détails d'implémentation sont différents
- manipule les std\_logic\_vector comme des nombres signés ou non-signés
- Dans les outils de simulation, on le trouve souvent dans la library ieee (même s'il n'en fait pas partie)

# TYPES DE DONNÉES

PACKAGE std\_logic\_arith

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_arith_unsigned.all;
...
signal s1,s2,s3,s4,s5,s6: std_logic_vector(3
downto 0);
...
s5<=s2+s1; -- ok, l'opérateur + surchargé
s6<= s2+1; -- ok, l'opérateur + surchargé
```

- un seul des deux packages peut être utilisé à la fois



# TYPES DE DONNÉES

PACKAGE std\_logic\_arith

- leur utilisation remet en cause la réputation du langage VHDL comme un langage très typé
- En conclusion : préférer le package **numeric\_std**





# EXERCICES

## UTILISATION DU PACKAGE numeric\_std

- ① Réaliser un additionneur 4 bits en instanciant l'additionneur 1 bit déjà présenté
- ② Décrire un additionneur 4 bits en utilisant le package `numeric_std`
- ③ Décrire un soustracteur 4 bits en utilisant le package `numeric_std`
- ④ Décrire un comparateur 8 bits en utilisant le package `numeric_std`
- ⑤ Pour le comparateur 8 bits, est-il nécessaire de faire des conversions en `unsigned` ?

# EXERCICES

## UTILISATION DU PACKAGE numeric\_std

- ① Réaliser un additionneur 4 bits en instanciant l'additionneur 1 bit déjà présenté
- ② Décrire un additionneur 4 bits en utilisant le package `numeric_std`
- ③ Décrire un soustracteur 4 bits en utilisant le package `numeric_std`
- ④ Décrire un comparateur 8 bits en utilisant le package `numeric_std`
- ⑤ Pour le comparateur 8 bits, est-il nécessaire de faire des conversions en `unsigned` ?



# EXERCICE 1

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
entity additionneur4bit is
port(
    a,b      : in  std_logic_vector(3 downto 0);
    cin     : in  std_logic;
    s       : out std_logic_vector(3 downto 0);
    cout    : out std_logic);
end;

architecture archConc of additionneur4bit is
component additionneur
port(
    a,b,cin  : in  std_logic;
    s,cout   : out std_logic);
end component;
signal c : std_logic_vector(2 downto 0);
begin
    add_1: additionneur port map (a(0),b(0),cin,s(0),c(0));
```





# EXERCICE 1

## SOLUTION

```
add_2: additionneur port map (a(1),b(1),c(0),s(1),c(1));
add_3: additionneur port map (a(2),b(2),c(1),s(2),c(2));
add_4: additionneur port map (a(3),b(3),c(2),s(3),cout);
end archConc;
```



# EXERCICE 2

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity additionneur4bit_1 is
port(
    a,b      : in  std_logic_vector(3 downto 0);
    cin      : in  std_logic;
    s        : out std_logic_vector(3 downto 0);
    cout     : out std_logic);
end;

architecture archConc of additionneur4bit_1 is
    signal s_int : unsigned (4 downto 0);
    signal cin_nat: natural range 0 to 1;
    signal t : std_logic_vector (4 downto 0);
begin
    cin_nat <= 1 when cin='1' else 0;
    s_int <= unsigned('0' & a) + unsigned('0' & b) + cin_nat;
    s      <= std_logic_vector(s_int(3 downto 0));

```

# EXERCICE 2

## SOLUTION

```
    cout  <= s_int(4);
end archConc;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity additionneur4bit_1a is
port(
    a,b      : in  std_logic_vector(3 downto 0);
    cin      : in  std_logic;
    s        : out std_logic_vector(3 downto 0);
    cout     : out std_logic);
end;

architecture archConc of additionneur4bit_1a is
    signal s_int : unsigned (4 downto 0);
begin
    s_int <= unsigned('0' & a) + unsigned('0' & b) +
        unsigned(std_logic_vector('0'&cin));
```



# EXERCICE 2

## SOLUTION

```
s      <= std_logic_vector(s_int(3 downto 0));
cout  <= s_int(4);
end archConc;
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity additionneur4bit_2 is
port(
    a,b      : in  std_logic_vector(3 downto 0);
    cin      : in  std_logic;
    s        : out std_logic_vector(3 downto 0);
    cout     : out std_logic);
end;

architecture archConc of additionneur4bit_2 is
    signal s_int : unsigned (4 downto 0);
    signal cin_vector: std_logic_vector (0 downto 0);
begin
```





## EXERCICE 2

### SOLUTION

```
    cin_vector(0) <= cin;
    s_int <= unsigned('0' & a) + unsigned('0' & b) + unsigned("0000" &
      cin_vector);
    --s_int <= resize(unsigned(a),5) + resize(unsigned(b),5) +
      resize(unsigned(cin_vector),5);
    s     <= std_logic_vector(s_int(3 downto 0));
    cout  <= s_int(4);
end archConc;
```





# EXERCICE 3

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity soustracteur4bit is
port(
    a,b      : in  std_logic_vector(3 downto 0);
    cin      : in  std_logic;
    s        : out std_logic_vector(3 downto 0);
    cout     : out std_logic);
end;

architecture archConc of soustracteur4bit is
    signal s_int : unsigned (4 downto 0);
    signal cin_vector: std_logic_vector (0 downto 0);
begin
    cin_vector(0) <= cin;
    s_int <= unsigned('0' & a) + unsigned(not('0' & b)) +
    unsigned("0000" & cin_vector);
```





# EXERCICE 3

## SOLUTION

```
--s_int <= resize(unsigned(a),5) + resize(unsigned(b),5) +
  resize(unsigned(cin_vector),5);
s      <= std_logic_vector(s_int(3 downto 0));
cout   <= s_int(4);
end archConc;
```



# EXERCICE 4

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity comparateur8bit_1 is
port(
    a,b      : in  std_logic_vector(7 downto 0);
    eq,gt,lt : out  std_logic);
end;

architecture archConc of comparateur8bit_1 is
    signal diff: unsigned (7 downto 0);
begin
    diff <= unsigned(a)-unsigned(b);
    eq <= '1' when diff = 0 else '0';
    gt <= '1' when diff > 0 else '0';
    lt <= '1' when diff < 0 else '0';
end archConc;
```



# EXERCICE 4

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity comparateur8bit is
port(
    a,b      : in  std_logic_vector(7 downto 0);
    eq,gt,lt : out  std_logic);
end;

architecture archConc of comparateur8bit is
begin
    eq <= '1' when a = b else '0';
    gt <= '1' when a > b else '0';
    lt <= '1' when a < b else '0';
end archConc;
```

# EXERCICES DE SYNTHÈSE

## CODAGE D'UNE ALU SIMPLE

- ➊ Donnez un code permettant de synthétiser une ALU opérant sur des données de 8 bits et possédant les 5 modes suivants : ADD; SUB; AND; OR; XOR
- ➋ Modifiez le code pour que la taille des données soit générique
- ➌ Observez le résultat de synthèse dans la vue RTL
- ➍ Écrivez un testbench permettant de vérifier automatiquement et exhaustivement les 5 modes avec des données codées sur 5 bits.

# EXERCICES DE SYNTHÈSE

## CODAGE D'UNE ALU SIMPLE

- ➊ Donnez un code permettant de synthétiser une ALU opérant sur des données de 8 bits et possédant les 5 modes suivants : ADD; SUB; AND; OR; XOR
- ➋ Modifiez le code pour que la taille des données soit générique
- ➌ Observez le résultat de synthèse dans la vue RTL
- ➍ Écrivez un testbench permettant de vérifier automatiquement et exhaustivement les 5 modes avec des données codées sur 5 bits.

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

- Le code VHDL de l'ALU

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu is
generic (N: integer:=8);
port(
    a,b      : in  std_logic_vector(N-1 downto 0);
    operation: in  std_logic_vector(2 downto 0);
    res      : out std_logic_vector(N downto 0));
end;

architecture archConc of alu is
begin
with operation select res <=
    std_logic_vector(unsigned('0' & a)+unsigned('0' & b)) when "000",
    std_logic_vector(unsigned('0' & a)-unsigned('0' & b)) when "001",
    '0' & a and '0' & b when "010",
```



# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
'0' & a or '0' & b when "011",
'0' & a xor '0' & b when "100",
(others => '0') when others;
end archConc;
```



# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

- Le code d'un testbench de l'ALU (non-exhaustif)

```
library ieee;
use ieee.std_logic_1164.all;
entity alu_tb is
end;

architecture archi_alu_tb of alu_tb is
component alu
generic (N: integer:=8);
port(
    a,b      : in  std_logic_vector(N-1 downto 0);
    operation: in  std_logic_vector(2 downto 0);
    res      : out std_logic_vector(N downto 0));
end component;
signal a_t,b_t: std_logic_vector(7 downto 0);
signal res_t : std_logic_vector(8 downto 0);
signal op_t  : std_logic_vector(2 downto 0);
begin
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
alu_1: alu generic map(8) port map(a_t,b_t,op_t,res_t);

op_t <= "000", "001" after 40 ns, "010" after 80 ns, "011" after 120
      ns, "100" after 160 ns;
a_t  <= x"01", x"02" after 10 ns, x"03" after 20 ns, x"04" after 30 ns,
      x"05" after 40 ns, x"06" after 50 ns, x"07" after 60 ns, x"08"
      after 70 ns,
      x"09" after 80 ns, x"0A" after 90 ns, x"0B" after 100 ns, x"0C"
      after 110 ns,
      x"0D" after 120 ns, x"0E" after 130 ns, x"0F" after 140 ns,
      x"10" after 150 ns;

b_t  <= x"10", x"0F" after 10 ns, x"0E" after 20 ns, x"0D" after 30 ns,
      x"0C" after 40 ns, x"0B" after 50 ns, x"0A" after 60 ns, x"09"
      after 70 ns,
      x"08" after 80 ns, x"07" after 90 ns, x"06" after 100 ns, x"05"
      after 110 ns,
      x"04" after 120 ns, x"03" after 130 ns, x"02" after 140 ns,
      x"01" after 150 ns;
```



# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
end;
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

- Le code d'un testbench de l'ALU exhaustif sans assertions

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu_tb_full is
end;

architecture archi_alu_tb_full of alu_tb_full is
component alu
generic (N: integer:=5);
port(
    a,b      : in  std_logic_vector(N-1 downto 0);
    operation: in  std_logic_vector(2 downto 0);
    res      : out std_logic_vector(N downto 0));
end component;
signal a_t,b_t: std_logic_vector(4 downto 0);
signal res_t : std_logic_vector(5 downto 0);
signal op_t   : std_logic_vector(2 downto 0);
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
signal cntA,cntB: integer range 0 to 32;
signal cntOp: integer range 0 to 7;
begin
alu_1: alu generic map(5) port map(a_t,b_t,op_t,res_t);

process
begin
    wait for 10 ns;
    cntA <= cntA+1;
    if cntA=31 then
        cntA<=0;
        cntB <= CntB+1;
        if cntB=31 then
            cntB<=0;
            cntOp <= cntOp+1;
            if cntOp=4 then
                cntOp <= 0;
            end if;
        end if;
```



# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
    end if;
end process;

a_t <= std_logic_vector(to_unsigned(cntA,5));
b_t <= std_logic_vector(to_unsigned(cntB,5));
op_t <= std_logic_vector(to_unsigned(cntOp,3));

end;
```



# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

- Le code d'un testbench de l'ALU exhaustif (avec `for loop`) sans assertions

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu_tb_full is
end;

architecture archi_alu_tb_full of alu_tb_full is
component alu
generic (N: integer:=5);
port(
    a,b      : in  std_logic_vector(N-1 downto 0);
    operation: in  std_logic_vector(2 downto 0);
    res      : out  std_logic_vector(N downto 0));
end component;
signal a_t,b_t: std_logic_vector(4 downto 0);
signal res_t : std_logic_vector(5 downto 0);
signal op_t   : std_logic_vector(2 downto 0);
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
begin
alu_1: alu generic map(5) port map(a_t,b_t,op_t,res_t);
process
    variable cntA,cntB: integer range 0 to 2**5-1;
    variable cntOp: integer range 0 to 4;
begin
    for cntOp in 0 to 4 loop
        op_t <= std_logic_vector(to_unsigned(cntOp,3));
        for cntA in 0 to 31 loop
            a_t <= std_logic_vector(to_unsigned(cntA,5));
            for cntB in 0 to 31 loop
                b_t <= std_logic_vector(to_unsigned(cntB,5));
                wait for 10 ns;
            end loop;
        end loop;
    end loop;
end process;
end;
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

- Le code d'un testbench de l'ALU exhaustif (avec `for loop`) avec assertions

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity alu_tb_full is
end;

architecture archi_alu_tb_full of alu_tb_full is
component alu
generic (N: integer:=5);
port(
    a,b      : in  std_logic_vector(N-1 downto 0);
    operation: in  std_logic_vector(2 downto 0);
    res      : out  std_logic_vector(N downto 0));
end component;
signal a_t,b_t: std_logic_vector(4 downto 0);
signal res_t : std_logic_vector(5 downto 0);
signal op_t   : std_logic_vector(2 downto 0);
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
begin
alu_1: alu generic map(5) port map(a_t,b_t,op_t,res_t);
process
    variable cntA,cntB: integer range 0 to 2**5-1;
    variable cntOp: integer range 0 to 4;
begin
    for cntOp in 0 to 4 loop
        op_t <= std_logic_vector(to_unsigned(cntOp,3));
        for cntA in 0 to 31 loop
            a_t <= std_logic_vector(to_unsigned(cntA,5));
            for cntB in 0 to 31 loop
                b_t <= std_logic_vector(to_unsigned(cntB,5));
                wait for 10 ns;
                case (cntOp) is
                    when 0 =>
                        assert
res_t=std_logic_vector(unsigned('0'&a_t)+unsigned('0'&b_t)) report
"Erreur: A=& integer'image(cntA) &
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
        " B= " & integer'image(cntB) & " Operation= " &
integer'image(cntOp) &
        " Resultat: " & integer'image(cntA+cntB) severity
failure;
        when 1 =>
        assert
res_t=std_logic_vector(unsigned('0'&a_t)-unsigned('0'&b_t)) report
"Erreur: A=& integer'image(cntA) &
        " B= " & integer'image(cntB) & " Operation= " &
integer'image(cntOp) &
        " Resultat: " & integer'image(cntA-cntB) severity
failure;
        when 2 =>
        assert res_t=((‘0’&a_t) and (‘0’&b_t)) report
"Erreur: A=& integer'image(cntA) &
        " B= " & integer'image(cntB) & " Operation= " &
integer'image(cntOp) &
```

# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
" Resultat: " &
integer'image(to_integer(to_unsigned(cntA,5) and
to_unsigned(cntB,5))) severity failure;
when 3 =>
    assert res_t=((‘0’&a_t) or (‘0’&b_t)) report "Erreur:
A=& integer'image(cntA) &
    " B= " & integer'image(cntB) & " Operation= " &
integer'image(cntOp) &
    " Resultat: " &
integer'image(to_integer(to_unsigned(cntA,5) or
to_unsigned(cntB,5))) severity failure;
when 4 =>
    assert res_t=((‘0’&a_t) xor (‘0’&b_t)) report
"Erreur: A=& integer'image(cntA) &
    " B= " & integer'image(cntB) & " Operation= " &
integer'image(cntOp) &
    " Resultat: " &
integer'image(to_integer(to_unsigned(cntA,5) xor
to_unsigned(cntB,5))) severity failure;
```



# EXERCICES DE SYNTHÈSE

## SOLUTION ALU

```
        end case;
      end loop;
    end loop;
  end loop;
end process;

end;
```



# SOMMAIRE

- Types d'objets
- Process
- Circuits séquentiels
- Utilisation de la carte FPGA

# TYPES D'OBJETS EN VHDL

- Un objet en VHDL est une structure pouvant garder la valeur d'un type de donnée

4 types d'objets :

- signal
- variable
- constante
- et fichier (non-synthétisable)

# SIGNAL

- déclaré dans la partie déclarative d'une architecture
- `signal signal_name, signal_name, ... : data_type;`
- affectation d'un signal :  
`signal_name <= new_value;`
- tous les ports d'une entité sont considérés comme signaux
- interprétation physique : peuvent être considérés comme des fils conducteurs ou des "fils avec mémoire" (cas des sorties de registres)

# VARIABLE

- déclarée et utilisée au sein d'un **process**

- déclaration d'une variable :

```
variable variable_name, ... : data_type;
```

- affectation d'une variable :

```
variable_name := new_value;
```

- ne contient pas d'information temporelle

- utilisée comme dans un langage de programmation traditionnel

- pas d'équivalent en HW

# CONSTANTE

- Valeur d'une constante ne peut pas être changée
- déclaration d'une constante :  
`constant const_name, ... : data_type:=value;`
- utilisée pour une meilleure lisibilité du code

```
constant BUS_WIDTH: integer:= 32;  
constant BUS_BYTES: integer:= BUS_WIDTH/8;
```

# ALIAS

- n'est pas un objet en sens propre du terme
- un nom alternatif pour un objet
- utilisé pour une meilleure lisibilité du code

```
signal word: std_logic(15 downto 0);
alias op: std_logic(6 downto 0) is word(15 downto 9);
alias reg1: std_logic(2 downto 0) is word(8 downto 6);
alias reg2: std_logic(2 downto 0) is word(5 downto 3);
alias reg3: std_logic(2 downto 0) is word(2 downto 0);
```

# SOMMAIRE

- Types d'objets
- Process
- Circuits séquentiels
- Utilisation de la carte FPGA



# PROCESS

- un ensemble d'instructions qui seront exécutées de manière séquentielle
- le **process** lui-même est concurrent
- peut être interprété comme une boîte noire dont les entrées/sorties sont connues
- peut être synthétisé en circuit (mais pas forcément)
- deux types de **process** :
  - ▷ avec une liste de sensibilité
  - ▷ sans liste de sensibilité (avec un **wait**)



# PROCESS

**process** avec une liste de sensibilité :

```
process(liste_de_sensibilite)
declarations;
begin
instruction sequentielle;
instruction sequentielle;
...
end process;
```

- un **process** est actif lorsqu'un des signaux présents sur sa liste de sensibilité change de valeur
- si un **process** est actif, son contenu sera exécuté de manière séquentielle jusqu'à la fin

# PROCESS

- exemple :

```
signal a,b,c,y : std_logic;  
process(a,b,c)  
begin  
y <= a and b and c;  
end process;
```

- Le comportement du modèle suivant est-il le même ?

# PROCESS

```
signal a,b,c,y : std_logic;  
process(a)  
begin  
y <= a and b and c;  
end process;
```

- Pour un circuit combinatoire, toutes les entrées doivent être présentes dans la liste de sensibilité
- Le synthétiseur peut l'interpréter en émettant un avertissement



# PROCESS

**process** sans liste de sensibilité (avec l'instruction **wait**)

- **process** poursuit son exécution jusqu'à l'instruction **wait** où il est suspendu
- Utilisations de l'instruction **wait**
  - ▷ **wait on signals;**
  - ▷ **wait until boolean\_expression;**
  - ▷ **wait for time\_expression;**
- exemple :





# PROCESS

```
signal a,b,c,y : std_logic;  
process  
begin  
y <= a and b and c;  
wait on a,b,c;  
end process;
```

- un **process** peut avoir plusieurs instructions **wait**
- un **process** avec une liste de sensibilité est préféré par les outils de synthèse
- Comment l'affectation des signaux/variables est effectuée au sein d'un **process** ?



# PROCESS

- Pour un signal :

```
signal_name <= new_value;
```

- Pas de différence par rapport à l'affectation en dehors d'un **process**

- **Attention** :

→ Au sein d'un **process**, un signal peut être affecté plusieurs fois ; seule la dernière affectation sera prise en compte

- Exemple :

# PROCESS

```
signal a,b,c,d,y : std_logic;
process(a,b,c,d)
begin
    --y_entry:=y
    y <= a or c;
    --y_exit :=a or c;
    y <= a and b;
    --y_exit :=a and b;
    y <= c and d;
    --y_exit :=c and d;
end process;
--y <= y_exit;
```

- Cette description est identique à :



# PROCESS

```
signal a,b,c,d,y : std_logic;  
process(a,b,c,d)  
begin  
y <= c and d;  
end process;
```

- Si les trois instructions étaient en dehors d'un **process** ?
- Affectation d'une variable
  - `variable_name:=new_value;`
- L'affectation prend effet immédiatement ;
- Exemple :



# PROCESS

```
signal a,b,c: std_logic;
process(a,b,c)
variable tmp: std_logic;
begin
tmp:='0';
tmp:= tmp or a;
tmp:= tmp or b;
y <= tmp;
end process;
```

- Si pour le même exemple, un signal au lieu d'une variable est utilisé ?

# PROCESS

```
signal a,b,c,tmp: std_logic;  
process(a,b,c)  
begin  
tmp<='0';  
tmp<= tmp or a;  
tmp<= tmp or b;  
y <= tmp;  
end process;
```

- Cette description est identique à :

# PROCESS

```
signal a,b,c,tmp: std_logic;
process(a,b,c,tmp)
begin
tmp<= tmp or b;
y <= tmp;
end process;
```

- A l'intérieur d'un **process**, on peut utiliser l'instruction **if-then-else**



# PROCESS

```
if expr_1 then  
seq_instruction;  
elsif expr2 then  
seq_instruction;  
elsif expr3 then  
seq_instruction;  
...  
else  
seq_instruction;  
end if;
```



# PROCESS

- Tous les exemples utilisant les instructions concurrentes **when-else** vus précédemment peuvent être décrits dans un **process** en utilisant **if-then-else**
- Pour obtenir le même circuit avec **when-else** et **if-then-else**, les deux structures doivent agir sur le même(s) signal(aux) de sortie
- L'instruction **if-then-else** est plus flexible puisqu'elle permet des structures imbriquées
- Exemple **when-else** :



# PROCESS

```
sig <= value_1 when expr_1 else  
value_2 when expr_2 else  
value_3 when expr_3 else  
...  
value_n;
```

- Exemple **if-then-else** :



# PROCESS

```
process(...)  
begin  
if expr_1 then  
sig <= value_1;  
elsif expr_2 then  
sig <= value_2;  
elsif expr_3 then  
sig <= value_3;  
...  
else  
sig <= value_n;  
end if;  
end process;
```



# PROCESS

- Exemple du calcul de `max(a,b,c)`

En utilisant `if-then-else`

```
process(a,b,c)
begin
  if (a>b) then
    if (a>c) then
      max <= a;
    else
      max <= c;
  end if;
  else
    if (b>c) then
```



# PROCESS

```
max <= b;  
else  
max <= c;  
end if;  
end if;  
end proces;
```

En utilisant **when-else**

# PROCESS

```
signal ac_max, bc_max, max: std_logic;  
...  
ac_max <= a when (a>c) else c;  
bc_max <= b when (b>c) else c;  
max    <= ac_max when (a>b) else bc_max;  
-- ou  
max  <= a when ((a>b) and (a>c)) else  
c when (a>b) else  
b when (b>c) else  
c;
```

- Dans l'instruction **if-then-else**, seule la partie **then** est nécessaire → les instructions incomplètes sont autorisées

# PROCESS

- Si un signal n'est pas affecté par omission, il garde sa valeur précédente → des *latches* sont inférés

```
process(a,b)
begin
if(a=b) then
eq <= '1';
end if;
end process;
```

```
process(a,b)
begin
if(a=b) then
eq <= '1';
else
eq <= eq;
end if;
end process;
```

Solution :



# PROCESS

```
process(a,b)
begin
if(a=b) then
eq <= '1';
else
eq <= '0';
end if;
end process;
```

# PROCESS

Un autre exemple : comparaison de deux valeurs

```
process(a,b)
begin
if (a>b) then
gt <= '1';
elsif (a=b) then
eq <= '1';
else
lt <= '1';
end if;
end process;
```



# PROCESS

Solution 1 :

```
process(a,b)
begin
if (a>b) then
gt <= '1';
eq <= '0';
lt <= '0';
elsif (a=b) then
gt <= '0';
eq <= '1';
lt <= '0';
else
gt <= '0';
eq <= '0';
lt <= '1';
end if;
end process;
```

Solution 2 :

```
process(a,b)
begin
gt <= '0';
eq <= '0';
lt <= '0';
if (a>b) then
gt <= '1';
elsif (a=b) then
eq <= '1';
else
lt <= '1';
end if;
end process;
```

# PROCESS

- Dans un **process**, on utilise l'instruction **case** à la place de **with-select**
- Tout ce qui est valable pour les instructions séquentielles incomplètes de type **if-then-else** est également valable pour **case**

Instructions de boucle

```
for index in loop_range loop
seq_instruction;
end loop;
```

- l'intervalle **loop\_range** doit être statique
- Les indices doivent appartenir à l'intervalle (de gauche à droite)



# PROCESS

```
library ieee;
use ieee.std_logic_1164.all;
entity bit_xor is
    port(
        a, b: in std_logic_vector(3 downto 0);
        y: out std_logic_vector(3 downto 0)
    );
end bit_xor;

architecture demo_arch of bit_xor is
    constant WIDTH: integer := 4;
begin
    process(a,b)
    begin
```



# PROCESS

```
for i in (WIDTH-1) downto 0 loop
    y(i) <= a(i) xor b(i);
end loop;
end process;
end demo_arch;
```

Un autre exemple d'opération OU exclusif entre bits d'un vecteur :

```
library ieee;
use ieee.std_logic_1164.all;
entity reduced_xor_demo is
port(
    a: in std_logic_vector(3 downto 0);
    y: out std_logic
```



# PROCESS

```
 );
end reduced_xor_demo;

architecture demo_arch of reduced_xor_demo is
  constant WIDTH: integer := 4;
  signal tmp: std_logic_vector(WIDTH-1 downto 0);
begin
  process(a,tmp)
  begin
    tmp(0) <= a(0);    -- boundary bit
    for i in 1 to (WIDTH-1) loop
      tmp(i) <= a(i) xor tmp(i-1);
    end loop;
  end process;
```

# PROCESS

```
y <= tmp(WIDTH-1);  
end demo_arch;
```

- "Dérouler" la boucle à réaliser
- A utiliser avec modération pour des instructions répétitives
- Exemple : opérations sur des bits d'un vecteurs

```
y(3) <= a(3) xor b(3);  
y(2) <= a(2) xor b(2);  
y(1) <= a(1) xor b(1);  
y(0) <= a(0) xor b(0);
```

```
tmp(0) <= a(0);  
tmp(1) <= a(1) xor tmp(0);  
tmp(2) <= a(2) xor tmp(1);  
tmp(3) <= a(3) xor tmp(2);  
y      <= tmp(3);
```



# PROCESS

- Les instructions concurrentes
  - ▷ s'inspirent du matériel
  - ▷ sont claires et simple d'utilisation et peuvent être plaquées sur une structure matérielle
- Les instructions séquentielles
  - ▷ l'objectif est de décrire le comportement d'une structure matérielle
  - ▷ plus flexibles
  - ▷ peuvent être difficilement synthétisable
  - ▷ risque d'utilisation incorrecte pour la synthèse
- Penser matériel (*Think HW!*)
- Conception matérielle : différent d'une simple conversion d'un programme C vers VHDL

# SOMMAIRE

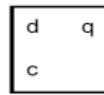
- Types d'objets
- Process
- Circuits séquentiels
- Utilisation de la carte FPGA

# CIRCUITS SÉQUENTIELS

- Combinatoire vs séquentiel
  - ▷ Dans un circuit séquentiel, la ou les sorties sont une fonction des entrées et des états précédents des sorties
- Les principaux circuits séquentiels :
  - ▷ un verrou D (*D latch*)
  - ▷ une bascule D (*D flip-flop*)
  - ▷ une mémoire

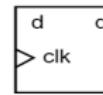
# CIRCUITS SÉQUENTIELS

- Un verrou D est sensible au niveau de l'horloge
- Une bascule D est sensible au front (montant ou descendant) d'horloge



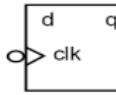
c	$q^*$
0	q
1	d

(a) D latch



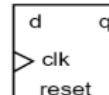
clk	$q^*$
0	q
1	q
↓	d

(b) pos-edge triggered D FF



clk	$q^*$
0	q
1	q
↑	d

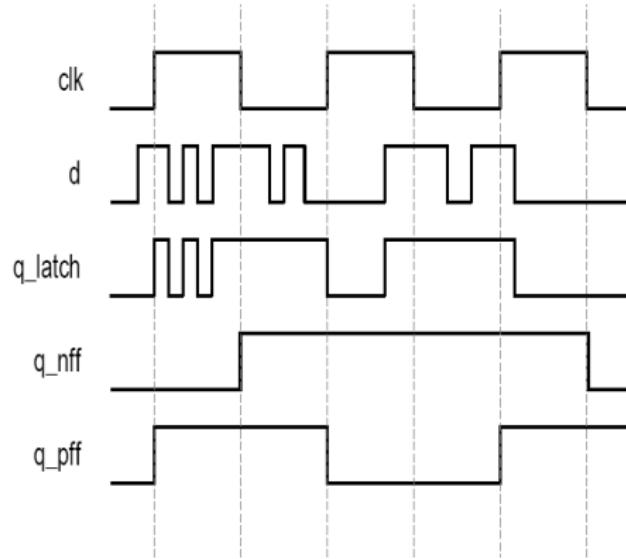
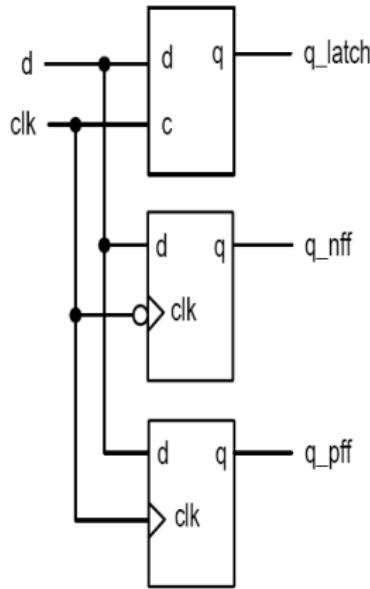
(c) neg-edge triggered D FF



reset	clk	$q^*$
1	-	0
0	0	q
0	1	q
0	↑	d

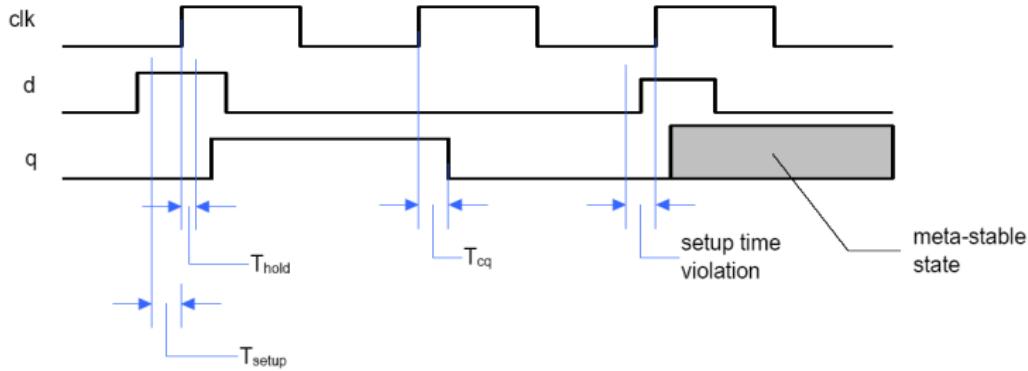
(d) D FF with asynchronous reset

# CIRCUITS SÉQUENTIELS



# CIRCUITS SÉQUENTIELS

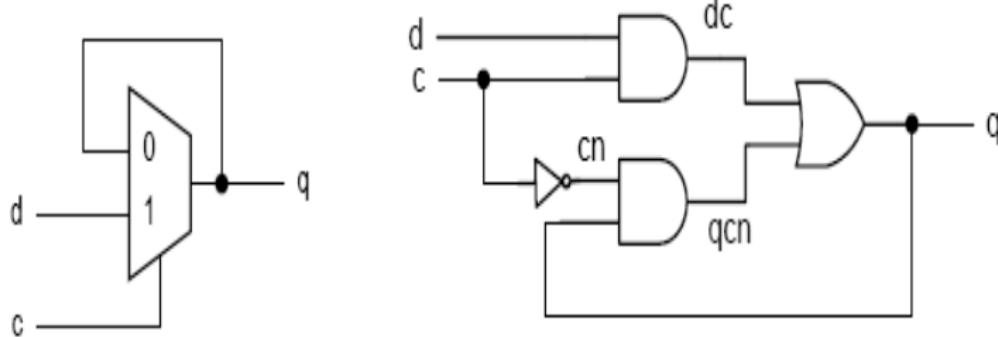
- Timing d'une bascule D
- *setup, hold, clock-to-q*



# CIRCUITS SÉQUENTIELS

- Une bascule D ou un verrou D peuvent être réalisé à partir de portes logiques élémentaires
- Il s'agit de circuits combinatoires bouclés dont la conception est délicate (ils sont sensibles aux délais)
- On ne devrait pas les synthétiser à bas niveau *from scratch*
- On préférera les cellules prédéfinies ou inférées automatiquement

# CIRCUITS SÉQUENTIELS



Un verrou D sensible à l'état haut

# CIRCUITS SÉQUENTIELS

```
library ieee;
use ieee.std_logic_1164.all;
entity dlatch is
    port(
        c: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end dlatch;

architecture arch of dlatch is
begin
    process(c,d)
    begin
```

# CIRCUITS SÉQUENTIELS

```
if (c='1') then
    q <= d;
end if;
end process;
end arch;
```

Une bascule D sensible au front montant de l'horloge

```
library ieee;
use ieee.std_logic_1164.all;
entity dff_1 is
port(
    clk: in std_logic;
    d: in std_logic;
    q: out std_logic
```

# CIRCUITS SÉQUENTIELS

```
 );
end dff_1;

architecture arch of dff_1 is
begin
  process(clk)
  begin
    if (clk'event and clk='1') then
      -- if rising_edge(clk) then
        q <= d;
    end if;
  end process;
end arch;
```

# CIRCUITS SÉQUENTIELS

## EXERCICES

- ➊ Décrire une bascule D avec une remise à zéro (*reset*) asynchrone.  
Décrire un fichier testbench permettant de tester cette bascule D.
- ➋ Rajouter à la bascule D précédente une remise à 1 (*preset*) asynchrone. Décrire un fichier testbench permettant de tester cette bascule D.
- ➌ Rajouter à la bascule D précédente un signal d'activation (*enable*) synchrone. Décrire un fichier testbench permettant de tester cette bascule D.
- ➍ Décrire en VHDL un registre 8 bits avec un *reset* asynchrone.  
Décrire un fichier testbench permettant de tester le registre.
- ➎ Décrire en VHDL un compteur générique (8 bits par défaut) avec chargement parallèle et *reset* synchrone. Décrire un fichier testbench permettant de tester le compteur 8 bits.

# CIRCUITS SÉQUENTIELS

## EXERCICES

- 6 Décrire en VHDL un compteur/décompteur générique (8 bits par défaut) avec chargement parallèle, *reset* asynchrone, signal *enable* et un signal de sens (*count/decount*). Décrire également un fichier testbench permettant de tester le circuit.

# EXERCICE 1

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
entity dffr is
    port(
        clk: in std_logic;
        reset: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end dffr;

architecture arch of dffr is
begin
```

# EXERCICE 1

## SOLUTION

```
process(clk,reset)
begin
    if (reset='1') then
        q <='0';
    elsif (clk'event and clk='1') then
        q <= d;
    end if;
end process;
end arch;
```

# EXERCICE 2

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
entity dffrp is
    port(
        clk: in std_logic;
        reset, preset: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end dffrp;

architecture arch of dffrp is
begin
```

# EXERCICE 2

## SOLUTION

```
process(clk,reset,preset)
begin
    if (reset='1') then
        q <='0';
    elsif (preset='1') then
        q <= '1';
    elsif (clk'event and clk='1') then
        q <= d;
    end if;
end process;
end arch;
```

# EXERCICE 3

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
entity dff_en is
    port(
        clk: in std_logic;
        reset: in std_logic;
        en: in std_logic;
        d: in std_logic;
        q: out std_logic
    );
end dff_en;

architecture two_seg_arch of dff_en is
```

# EXERCICE 3

## SOLUTION

```
signal q_reg: std_logic;
signal q_next: std_logic;
begin
    -- D FF
    process(clk,reset)
    begin
        if (reset='1') then
            q_reg <= '0';
        elsif (clk'event and clk='1') then
            q_reg <= q_next;
        end if;
    end process;
    -- next-state logic
    q_next <= d when en ='1' else
```

# EXERCICE 3

## SOLUTION

```
        q_reg;
-- output logic
q <= q_reg;
end two_seg_arch;
architecture one_seg_arch of dff_en is
begin
process(clk,reset)
begin
if (reset='1') then
    q <='0';
elsif (clk'event and clk='1') then
    if (en='1') then
        q <= d;
    end if;
```

# EXERCICE 3

## SOLUTION

```
    end if;  
end process;  
end one_seg_arch;
```

# EXERCICE 4

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
entity reg8 is
    port(
        clk: in std_logic;
        reset: in std_logic;
        d: in std_logic_vector(7 downto 0);
        q: out std_logic_vector(7 downto 0)
    );
end reg8;

architecture arch of reg8 is
begin
```

# EXERCICE 4

## SOLUTION

```
process(clk,reset)
begin
    if (reset='1') then
        q <=(others=>'0');
    elsif (clk'event and clk='1') then
        q <= d;
    end if;
end process;
end arch;
```

# EXERCICE 5

## SOLUTION

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

entity compteur is
    generic (TAILLE : integer := 8 );
    port(
        din   : in  std_logic_vector(TAILLE-1 downto 0);
        clk   : in  std_logic;
        load  : in  std_logic;
        reset : in  std_logic;
        dout  : out std_logic_vector(TAILLE-1 downto 0));
end;
architecture behavior of compteur is begin
clk_proc:process(clk)
```

# EXERCICE 5

## SOLUTION

```
variable count:unsigned(TAILLE-1 downto 0) := (others
    => '0');
begin
    if rising_edge(clk) then
        if reset = '1' then
            count := (others => '0');
        elsif load = '1' then
            count := unsigned(din);
        else
            count := count + 1;
        end if;
    end if;
    dout <= std_logic_vector(count);
end process clk_proc;
end behavior;
```

# EXERCICE 6

## SOLUTION

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

entity compteur_decompteur is
generic (TAILLE : integer := 16 );
port(
    din   : in  std_logic_vector(TAILLE-1 downto 0);
    clk   : in  std_logic;
    load  : in  std_logic;
    reset : in  std_logic;
    updown : in  std_logic;
    dout  : out std_logic_vector(TAILLE-1 downto 0));
end;
architecture behavior of compteur_decompteur is begin
```

# EXERCICE 6

## SOLUTION

```
clk_proc:process(clk)
    variable count:unsigned(TAILLE-1 downto 0) := (others
        => '0');
    begin
        if rising_edge(clk) then
            if reset = '1' then
                count := (others => '0');
            elsif load = '1' then
                count := unsigned(din);
            elsif updown = '0' then
                if count = to_unsigned(2**TAILLE-1, TAILLE) then
                    count:=(others=> '0');
                else
                    count := count + 1;
                end if;
            end if;
        end if;
    end process;
```

# EXERCICE 6

## SOLUTION

```
else
    if count = 0 then
        count:= to_unsigned(2**TAILLE-1, TAILLE);
    else
        count := count - 1;
    end if;
end if;
dout <= std_logic_vector(count);
end process clk_proc;
end behavior;
```

# SOMMAIRE

- Types d'objets
- Process
- Circuits séquentiels
- Utilisation de la carte FPGA

# EXEMPLES SIMPLES

## CIRCUIT COMBINATOIRE

- ➊ Décrire en VHDL un circuit qui réalise différentes opérations arithmétiques et logiques sur des opérandes 4 bits en fonction d'un sélecteur sur 2 bits.
- ➋ Tester le fonctionnement sur la carte
  - On utilisera les interrupteurs **SW[9]** et **SW[8]** pour la sélection des fonctions  
 $(00 \rightarrow ET; 01 \rightarrow OU; 10 \rightarrow OU-Excl; 11 \rightarrow +)$  et les opérandes seront codés par **SW[7..4]** et **SW[3..0]** respectivement. Les résultats seront visualisés sur les LEDS **LEDR[3..0]**
  - Pour les contraintes de placement, importer le fichier **qsf** disponible sur Arche ou en créer un nouveau.

# EXEMPLES SIMPLES

## CIRCUIT COMBINATOIRE

```
set_location_assignment PIN_AE26 -to HEX0[0]
set_location_assignment PIN_AE27 -to HEX0[1]
set_instance_assignment -name IO_STANDARD LVTTL
    -to HEX0[0]
set_instance_assignment -name IO_STANDARD LVTTL
    -to HEX0[1]
```

- Après compilation vérifier qu'il n'y a pas d'avertissements critiques (problèmes de placement).
- Pour la programmation, choisir le circuit de type 5CSEMA puis lui associer le fichier sof du projet.

## SOLUTION A)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity BOUTONSLEDS is
    port(
        SW    : in  std_logic_vector(9 downto 0);
        LEDR : out std_logic_vector(9 downto 0));
end BOUTONSLEDS;
architecture behavior of BOUTONSLEDS is
begin
    WITH SW(9 downto 8) SELECT
        LEDR(3 downto 0) <= SW(3 downto 0)and SW(7 downto 4) when "00",
        SW(3 downto 0)or SW(7 downto 4) when "01",
        SW(3 downto 0)xor SW(7 downto 4) when "10",
        std_logic_vector(unsigned(SW(3 downto 0)) + unsigned(SW(7 downto
4))) when "11",
        "0000" when others;
        LEDR(9 downto 8) <= SW(9 downto 8);
        LEDR(7 downto 4) <= "0000";
end behavior;
```

## SOLUTION B)

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity boutons_leds is
port(
    sel: in std_logic_vector(1 downto 0);
    a : in std_logic_vector(3 downto 0);
    b : in std_logic_vector(3 downto 0);
    c : out std_logic_vector(3 downto 0);
    s : out std_logic_vector(1 downto 0));
end;
architecture behavior of boutons_leds is
begin
    WITH sel SELECT
        c <= a and b when "00",
        a or b when "01",
        a xor b when "10",
        std_logic_vector(unsigned(a) + unsigned(b)) when "11",
        "0000" when others;
    s <= sel;
```

## SOLUTION B)

```
end behavior;
```

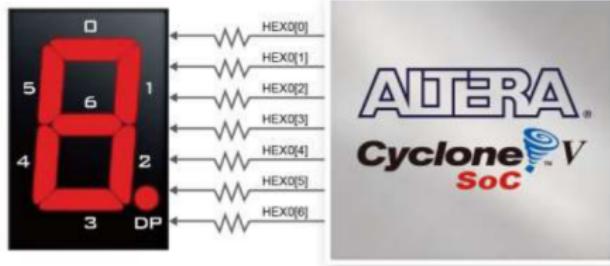
```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
entity boutons_leds_wrapper is
port(
    SW : in std_logic_vector(9 downto 0);
    LEDR : out std_logic_vector(9 downto 0));
end;
architecture behavior of boutons_leds_wrapper is
component boutons_leds
port(
    sel: in std_logic_vector(1 downto 0);
    a : in std_logic_vector(3 downto 0);
    b : in std_logic_vector(3 downto 0);
    c : out std_logic_vector(3 downto 0);
    s : out std_logic_vector(1 downto 0));
end component;
```

## SOLUTION B)

```
begin  
  
    uut: boutons_leds port map (SW(9 downto 8),SW(7 downto 4),SW(3 downto  
        0),  
                                LEDR(3 downto 0),LEDR(9 downto 8));  
  
end behavior;
```

# EXEMPLES SIMPLES

## CIRCUIT COMBINATOIRE MODIFIÉ



- 1 Reprendre le circuit précédent et lui ajouter des décodeurs Hexadécimal → 7-segments afin d'afficher les valeurs numériques des entrées et des sorties.
- 2 Tester le fonctionnement sur la carte (Les afficheurs se nomment `HEX0[6..0]` à `HEX5[6..0]` dans le fichier de contraintes)

# SOLUTION POUR LE DÉCODEUR 7 SEGMENTS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity dec_7seg is
    port(hex_in : in  std_logic_vector(3 downto 0);
         segs      : out  std_logic_vector(6 downto 0));
end dec_7seg;
architecture rtl of dec_7seg is
    signal segs_int   : std_logic_vector(6 downto 0);
begin
    process(hex_in)
    begin
        case hex_in is
            when "0000" => segs_int <= "0111111";
            when "0001" => segs_int <= "0000110";
            when "0010" => segs_int <= "1011011";
            when "0011" => segs_int <= "1001111";
            when "0100" => segs_int <= "1110110";
            when "0101" => segs_int <= "1011011";
            when "0110" => segs_int <= "1111101";
```

# SOLUTION POUR LE DÉCODEUR 7 SEGMENTS

```
when "0111" => segs_int <= "0000111";
when "1000" => segs_int <= "1111111";
when "1001" => segs_int <= "1101111";
when "1010" => segs_int <= "1110111";
when "1011" => segs_int <= "0111001";
when "1100" => segs_int <= "0011001";
when "1101" => segs_int <= "1011110";
when "1110" => segs_int <= "1111001";
when "1111" => segs_int <= "1110001";
when others => segs_int <= "0101010";
end case;
end process;
segs <= not segs_int;
end rtl;
```

# EXEMPLES SIMPLES

## CIRCUIT SÉQUENTIEL

- ➊ Modifier le compteur de l'exercice 5 des circuits séquentiels pour en faire un compteur modulo 10 et lui ajouter une entrée de validation et une sortie de retenue.
- ➋ Décrire en VHDL un circuit synchrone qui compte en décimal le nombre d'appuis (au moins jusqu'à 99) sur un bouton poussoir depuis la remise à zéro effectuée avec un autre bouton poussoir.
- ➌ Quelle précaution faut-il prendre (circuit à ajouter) pour que ce circuit fonctionne correctement ?

# EXEMPLES SIMPLES

## CIRCUIT SÉQUENTIEL

- ④ Tester le fonctionnement sur la carte. On utilisera les boutons poussoirs parmi KEY[3..0]. Les résultats seront visualisés sur les afficheurs 7 segments parmi HEX0[6..0] à HEX5[6..0]. Pour l'horloge choisir CLOCK\_50. Ajouter en plus des contraintes de placement, une contrainte de temps sur l'horloge du système (fichier .sdc).

# SOLUTION POUR LE DÉCODEUR 7 SEGMENTS

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity dec_7seg is
    port(hex_in : in  std_logic_vector(3 downto 0);
         segs      : out  std_logic_vector(6 downto 0));
end dec_7seg;
architecture rtl of dec_7seg is
    signal segs_int   : std_logic_vector(6 downto 0);
begin
    process(hex_in)
    begin
        case hex_in is
            when "0000" => segs_int <= "0111111";
            when "0001" => segs_int <= "0000110";
            when "0010" => segs_int <= "1011011";
            when "0011" => segs_int <= "1001111";
            when "0100" => segs_int <= "1110110";
            when "0101" => segs_int <= "1011011";
            when "0110" => segs_int <= "1111101";
```

# SOLUTION POUR LE DÉCODEUR 7 SEGMENTS

```
when "0111" => segs_int <= "0000111";
when "1000" => segs_int <= "1111111";
when "1001" => segs_int <= "1101111";
when "1010" => segs_int <= "1110111";
when "1011" => segs_int <= "0111001";
when "1100" => segs_int <= "0011001";
when "1101" => segs_int <= "1011110";
when "1110" => segs_int <= "1111001";
when "1111" => segs_int <= "1110001";
when others => segs_int <= "0101010";
end case;
end process;
segs <= not segs_int;
end rtl;
```

# EXEMPLES SIMPLES

## CIRCUIT SÉQUENTIEL

- ➊ Modifier le compteur de l'exercice 5 des circuits séquentiels pour en faire un compteur modulo 10 et lui ajouter une entrée de validation et une sortie de retenue.
- ➋ Décrire en VHDL un circuit synchrone qui compte en décimal le nombre d'appuis (au moins jusqu'à 99) sur un bouton poussoir depuis la remise à zéro effectuée avec un autre bouton poussoir.
- ➌ Quelle précaution faut-il prendre (circuit à ajouter) pour que ce circuit fonctionne correctement ?

# EXEMPLES SIMPLES

## CIRCUIT SÉQUENTIEL

- ➄ Tester le fonctionnement sur la carte. On utilisera les boutons poussoirs parmi KEY[3..0]. Les résultats seront visualisés sur les afficheurs 7 segments parmi HEX0[6..0] à HEX5[6..0]. Pour l'horloge choisir CLOCK\_50. Ajouter en plus des contraintes de placement, une contrainte de temps sur l'horloge du système (fichier .sdc).

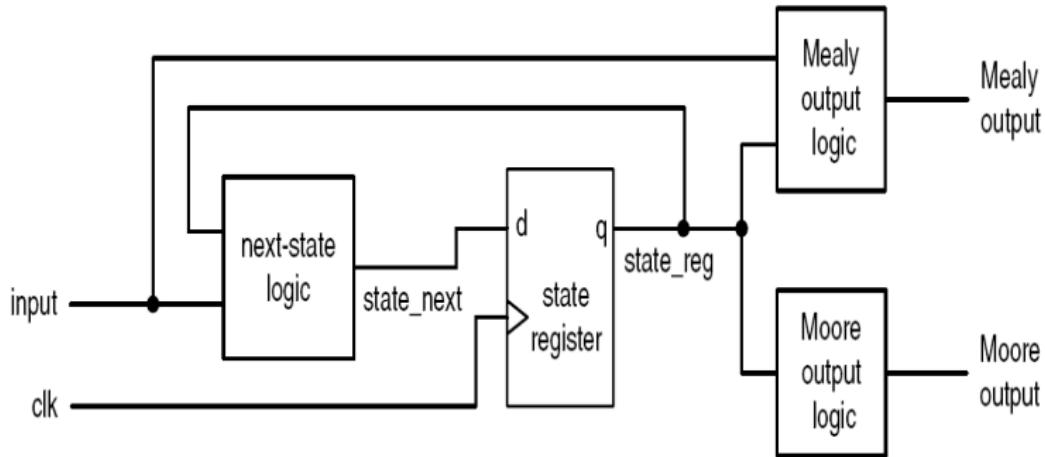


# SOMMAIRE

- 5 Machines à état fini (Finite State Machines)
  - Machines à état fini (Finite State Machines)

# MACHINES D'ÉTATS

- Pour la réalisation de contrôleur dans un circuit complexe
- Deux types de machines : Mealy et Moore

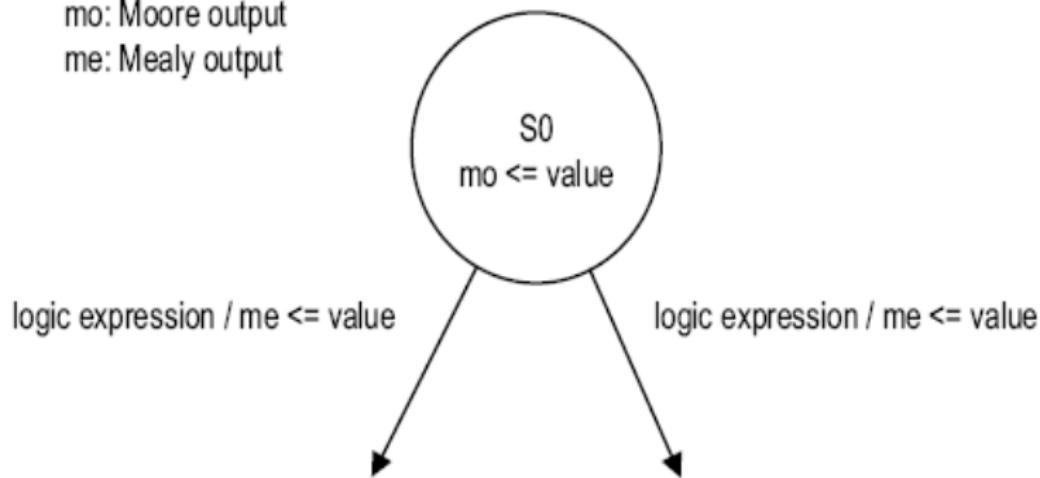


# MACHINES D'ÉTATS

## DIAGRAMME D'ÉTATS

mo: Moore output

me: Mealy output





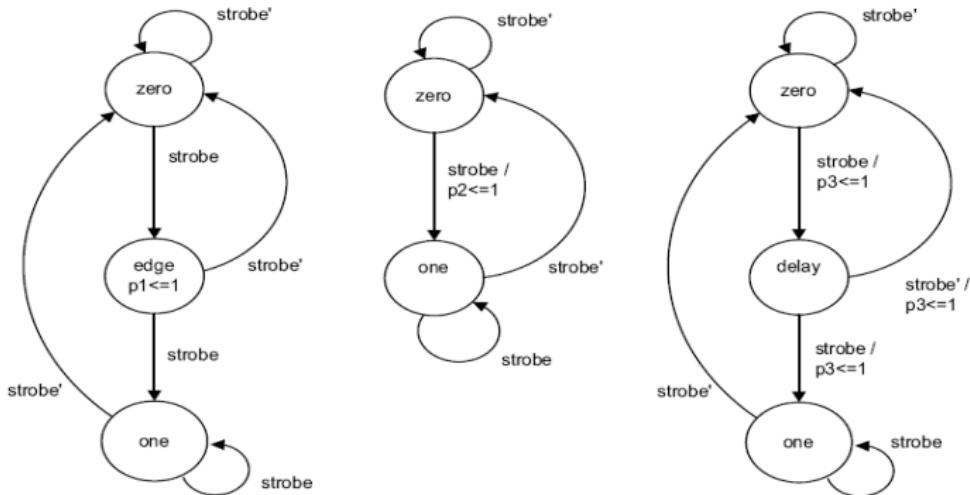
# MOORE VS MEALY

- Machine de Moore
  - ▷ la sortie est uniquement la fonction de l'état de la machine
- Machine de Mealy
  - ▷ la sortie est une fonction de l'état de la machine et des entrées
- Comment choisir le type de machine d'états ?

# MOORE VS MEALY

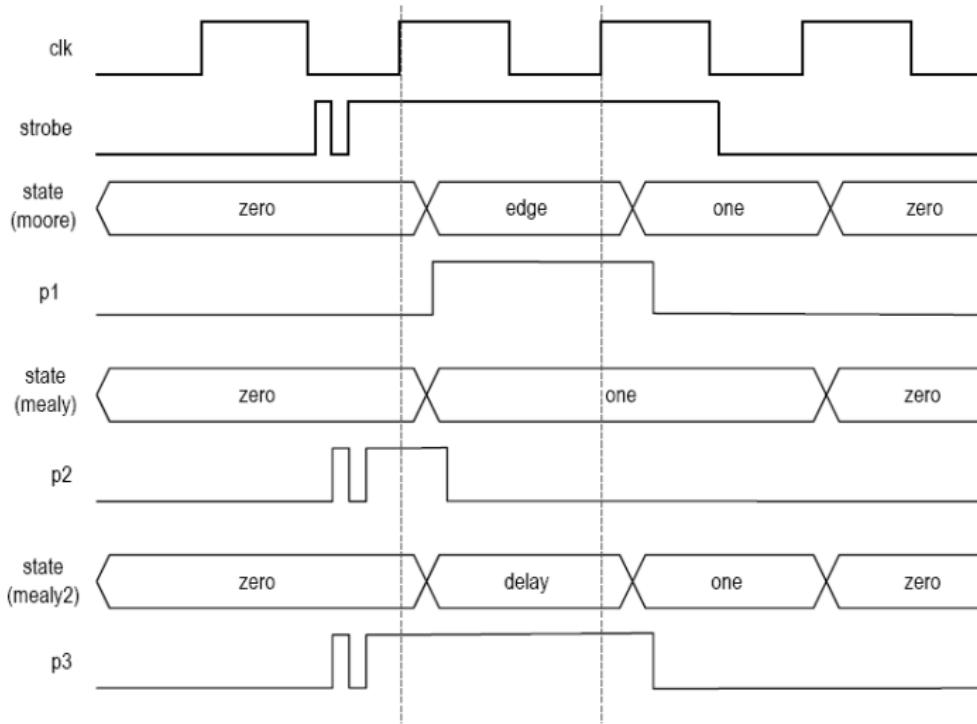
## EXEMPLE

- Détecteur de front montant d'un signal d'entrée



# MOORE VS MEALY

## EXEMPLE



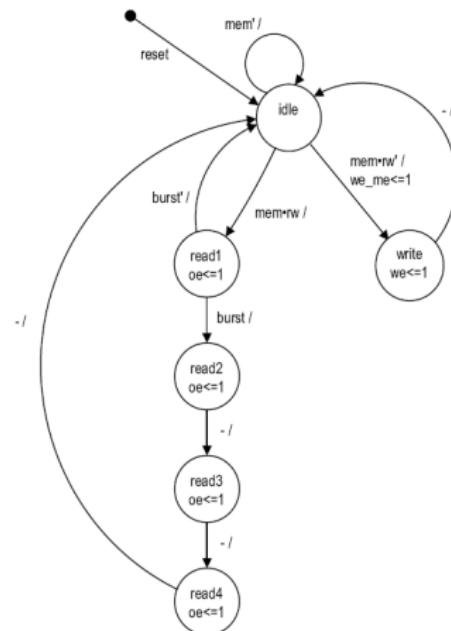
# MOORE VS MEALY

## EXEMPLE

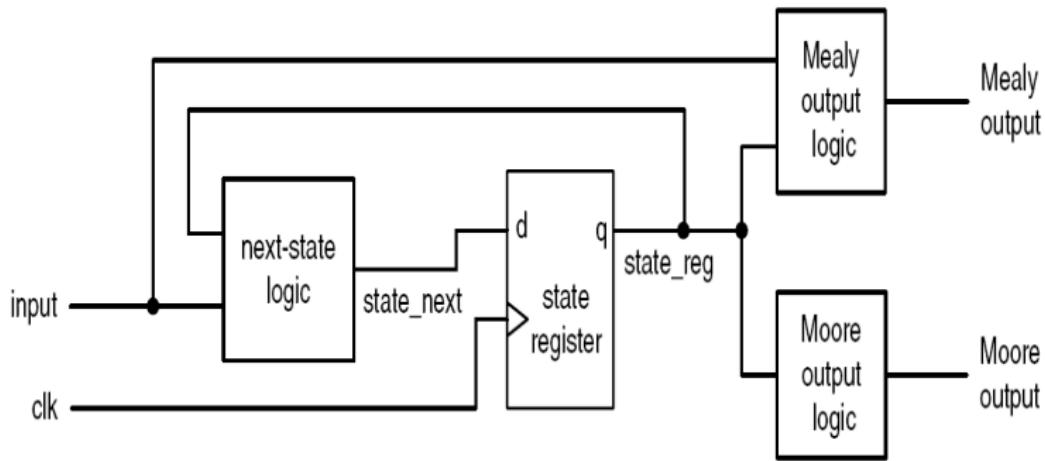
- Machine de Mealy :
  - ▷ utilise en général moins d'états
  - ▷ plus rapide
  - ▷ transparente aux signaux transitoires (*glitches*)
- Laquelle des deux est meilleure ?
- La réponse dépend du type de signal de contrôle à réaliser
- Un signal sensible aux fronts montant ou descendant (*edge sensitive*)
  - ▷ Exemple : le signal d'activation d'un compteur
  - ▷ Les deux peuvent être utilisées mais la machine de Mealy est plus rapide
- Un signal sensible au niveau (*level sensitive*)
  - ▷ Exemple : un signal d'écriture d'une SRAM
  - ▷ Pour ce cas de figure, à préférer une machine de Moore

# MACHINE D'ÉTATS EN VHDL

- Utilisation de types énumérés pour coder les états d'une FSM et pour rendre le code plus lisible



# MACHINE D'ÉTATS EN VHDL



□ Entité

# MACHINE D'ÉTATS EN VHDL

```
library ieee;
use ieee.std_logic_1164.all;
entity mem_ctrl is
    port(
        clk, reset: in std_logic;
        mem, rw, burst: in std_logic;
        oe, we, we_me: out std_logic
    );
end mem_ctrl ;
```

# MACHINE D'ÉTATS EN VHDL

- Type énuméré pour décrire les états

```
architecture mult_seg_arch of mem_ctrl is
    type mc_state_type is
        (idle, read1, read2, read3, read4, write);
    signal state_reg, state_next: mc_state_type;
begin
```

- Registre d'états

```
-- state register
process(clk,reset)
begin
    if (reset='1') then
        state_reg <= idle;
    elsif (clk'event and clk='1') then
        state_reg <= state_next;
    end if;
end process;
-- next-state logic
```

# MACHINE D'ÉTATS EN VHDL

```
-- next-state logic
process(state_reg,mem,rw,burst)
begin
    case state_reg is
        when idle =>
            if mem='1' then
                if rw='1' then
                    state_next <= read1;
                else
                    state_next <= write;
                end if;
            else
                state_next <= idle;
            end if;
        when write =>
            state_next <= idle;
        when read1 =>
            if (burst='1') then
                state_next <= read2;
            else
```

# MACHINE D'ÉTATS EN VHDL

```
        state_next <= idle;
    end if;
when read2 =>
    state_next <= read3;
when read3 =>
    state_next <= read4;
when read4 =>
    state_next <= idle;
end case;
end process;
```

# MACHINE D'ÉTATS EN VHDL

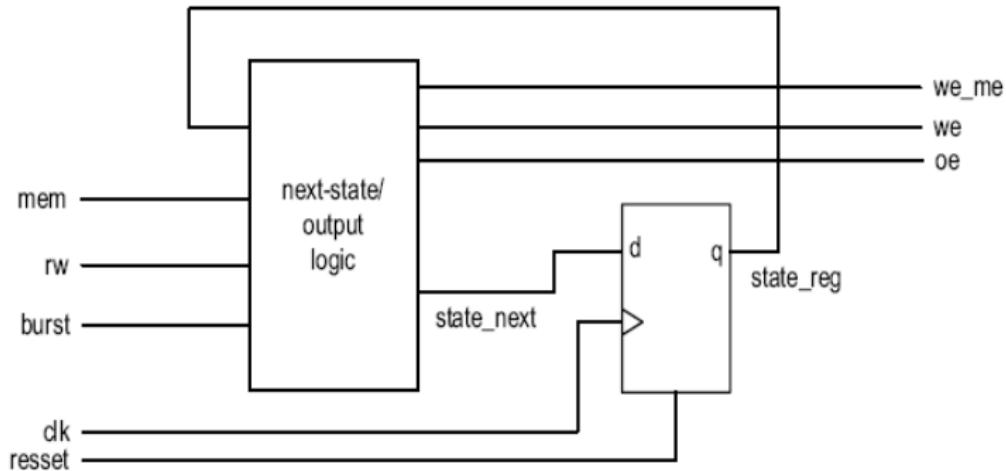
```
-- Moore output logic
process(state_reg)
begin
    we <= '0'; -- default value
    oe <= '0'; -- default value
    case state_reg is
        when idle =>
        when write =>
            we <= '1';
        when read1 =>
            oe <= '1';
        when read2 =>
            oe <= '1';
        when read3 =>
            oe <= '1';
        when read4 =>
            oe <= '1';
    end case;
end process;
```

# MACHINE D'ÉTATS EN VHDL

```
-- Mealy output logic
process(state_reg,mem,rw)
begin
    we_me <= '0'; -- default value
    case state_reg is
        when idle =>
            if (mem='1') and (rw='0') then
                we_me <= '1';
            end if;
        when write =>
        when read1 =>
        when read2 =>
        when read3 =>
        when read4 =>
    end case;
end process;
we_me <= '1' when ((state_reg=idle) and (mem='1') and (rw='0')) else
'0';
end mult_seg_arch;
```

# MACHINE D'ÉTATS EN VHDL

- Combinez la logique de sortie avec la logique de l'état futur



# MACHINE D'ÉTATS EN VHDL

```
-- next-state logic and output logic
process(state_reg,mem,rw,burst)
begin
    oe <= '0';      -- default values
    we <= '0';
    we_me <= '0';
    case state_reg is
        when idle =>
            if mem='1' then
                if rw='1' then
                    state_next <= read1;
                else
                    state_next <= write;
                    we_me <= '1';
                end if;
            else
                state_next <= idle;
            end if;
        when write =>
            state_next <= idle;
```

# MACHINE D'ÉTATS EN VHDL

```
    we <= '1';
when read1 =>
  if (burst='1') then
    state_next <= read2;
  else
    state_next <= idle;
  end if;
  oe <= '1';
when read2 =>
  state_next <= read3;
  oe <= '1';
when read3 =>
  state_next <= read4;
  oe <= '1';
when read4 =>
  state_next <= idle;
  oe <= '1';
end case;
end process;
```

# MACHINE D'ÉTATS EN VHDL

- Une machine peut être décrite en un seul segment

```
architecture one_seg_wrong_arch of mem_ctrl is
  type mc_state_type is
    (idle, read1, read2, read3, read4, write);
  signal state_reg: mc_state_type;
begin
  process(clk,reset)
  begin
    if (reset='1') then
      state_reg <= idle;
    elsif (clk'event and clk='1') then
      oe <= '0';      -- default values
      we <= '0';
      we_me <= '0';
      case state_reg is
        when idle =>
          if mem='1' then
            if rw='1' then
              state_reg <= read1;
```

# MACHINE D'ÉTATS EN VHDL

```
        else
            state_reg <= write;
            we_me <= '1';
        end if;
    else
        state_reg <= idle;
    end if;
when write =>
    state_reg <= idle;
    we <= '1';
when read1 =>
    if (burst='1') then
        state_reg <= read2;
    else
        state_reg <= idle;
    end if;
    oe <= '1';
when read2 =>
    state_reg <= read3;
    oe <= '1';
```

# MACHINE D'ÉTATS EN VHDL

```
when read3 =>
    state_reg <= read4;
    oe <= '1';
when read4 =>
    state_reg <= idle;
    oe <= '1';
end case;
end if;
end process;
end one_seg_wrong_arch;
```

# MACHINE D'ÉTATS EN VHDL

- L'encodage des états d'une machine d'états peut être effectué à plusieurs façons :
  - ▷ binaire, binaire réfléchi, *one-hot* ou presque *one-hot*
- Le type d'encodage choisi ne joue pas sur la fonctionnalité de la FSM mais surtout sur sa complexité (partie l'état futur et la logique de sortie)

**Table 10.1** State assignment example

	<b>Binary assignment</b>	<b>Gray code assignment</b>	<b>One-hot assignment</b>	<b>Almost one-hot assignment</b>
idle	000	000	000001	00000
read1	001	001	000010	00001
read2	010	011	000100	00010
read3	011	010	001000	00100
read4	100	110	010000	01000
write	101	111	100000	10000

# MACHINE D'ÉTATS EN VHDL

- Utilisation de l'attribut `enum_encoding`

```
type mc_state_type is (idle,write,read1,  
read2,read3,read4);  
attribute enum_encoding: string;  
attribute enum_encoding of mc_state_type:  
type is "0000 0100 1000 1001 1010 1011";
```

- Utilisation explicite de constantes

# MACHINE D'ÉTATS EN VHDL

```
architecture state_assign_arch of mem_ctrl is
    constant idle: std_logic_vector(3 downto 0):="0000";
    constant write: std_logic_vector(3 downto 0):="0100";
    constant read1: std_logic_vector(3 downto 0):="1000";
    constant read2: std_logic_vector(3 downto 0):="1001";
    constant read3: std_logic_vector(3 downto 0):="1010";
    constant read4: std_logic_vector(3 downto 0):="1011";
    signal state_reg,state_next: std_logic_vector(3 downto 0);
begin
    -- state register
    process(clk,reset)
    begin
        if (reset='1') then
            state_reg <= idle;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
        end if;
    end process;
    -- next-state logic
    process(state_reg,mem,rw,burst)
```

# MACHINE D'ÉTATS EN VHDL

```
begin
    case state_reg is
        when idle =>
            if mem='1' then
                if rw='1' then
                    state_next <= read1;
                else
                    state_next <= write;
                end if;
            else
                state_next <= idle;
            end if;
        when write =>
            state_next <= idle;
        when read1 =>
            if (burst='1') then
                state_next <= read2;
            else
                state_next <= idle;
            end if;
```

# MACHINE D'ÉTATS EN VHDL

```
when read2 =>
    state_next <= read3;
when read3 =>
    state_next <= read4;
when read4 =>
    state_next <= idle;
when others =>
    state_next <= idle;
end case;
end process;
```

- Parfois, un signal de sortie sans *glitch* (aléas) est nécessaire
- Pour l'obtenir, il faut rajouter une bascule en sortie introduisant un cycle de retard supplémentaire
- Une solution nommée *look-ahead* permet de réduire ce cycle d'horloge tout en veillant à ce que le signal de sortie soit sans *glitches*

# MACHINE D'ÉTATS EN VHDL

```
-- output buffer
process(clk,reset)
begin
    if (reset='1') then
        oe_buf_reg <= '0';
        we_buf_reg <= '0';
    elsif (clk'event and clk='1') then
        oe_buf_reg <= oe_next;
        we_buf_reg <= we_next;
    end if;
end process;
```

# MACHINE D'ÉTATS EN VHDL

```
-- look-ahead output logic
process(state_next)
begin
    we_next <= '0'; -- default value
    oe_next <= '0'; -- default value
    case state_next is
        when idle =>
        when write =>
            we_next <= '1';
        when read1 =>
            oe_next <= '1';
        when read2 =>
            oe_next <= '1';
        when read3 =>
            oe_next <= '1';
        when read4 =>
            oe_next <= '1';
    end case;
end process;
```

# MACHINE D'ÉTATS

## EXEMPLES

- Coder en VHDL un détecteur de front montant en utilisant :
  - ▷ une machine de Moore
  - ▷ une machine de Mealy

# Edge detector

## SOLUTION

```
library ieee;
use ieee.std_logic_1164.all;
entity edge_detector1 is
    port(
        clk, reset: in std_logic;
        strobe: in std_logic;
        p1: out std_logic
    );
end edge_detector1;

architecture moore_arch of edge_detector1 is
    type state_type is (zero, edge, one);
    signal state_reg, state_next: state_type;
begin
    -- state register
    process(clk,reset)
    begin
        if (reset='1') then
            state_reg <= zero;
```

# Edge detector

## SOLUTION

```
elsif (clk'event and clk='1') then
    state_reg <= state_next;
end if;
end process;
-- next-state logic
process(state_reg,strobe)
begin
    case state_reg is
        when zero=>
            if strobe= '1' then
                state_next <= edge;
            else
                state_next <= zero;
            end if;
        when edge =>
            if strobe= '1' then
                state_next <= one;
            else
                state_next <= zero;
```

# Edge detector

## SOLUTION

```
        end if;
when one =>
    if strobe= '1' then
        state_next <= one;
    else
        state_next <= zero;
    end if;
end case;
end process;
-- Moore output logic
p1 <= '1' when state_reg=edge else
    '0';
end moore_arch;
```

# Edge detector

## SOLUTION

- Machine d'états Moore sans la logique de sortie

```
architecture clever_assign_buf_arch of edge_detector1 is
    constant zero: std_logic_vector(1 downto 0):= "00";
    constant edge: std_logic_vector(1 downto 0):= "10";
    constant one: std_logic_vector(1 downto 0) := "01";
    signal state_reg,state_next: std_logic_vector(1 downto 0);
begin
    -- state register
    process(clk,reset)
    begin
        if (reset='1') then
            state_reg <= zero;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
        end if;
    end process;
    -- next-state logic
    process(state_reg,strobe)
```

# Edge detector

## SOLUTION

```
begin
    case state_reg is
        when zero=>
            if strobe= '1' then
                state_next <= edge;
            else
                state_next <= zero;
            end if;
        when edge =>
            if strobe= '1' then
                state_next <= one;
            else
                state_next <= zero;
            end if;
        when others =>
            if strobe= '1' then
                state_next <= one;
            else
                state_next <= zero;
```

# Edge detector

## SOLUTION

```
        end if;
    end case;
end process;
-- Moore output logic
p1 <= state_reg(1);
end clever_assign_buf_arch;
```

# Edge detector

## SOLUTION

- Machine d'états Moore avec la logique *look-ahead*

```
architecture look_ahead_arch of edge_detector1 is
    type state_type is (zero, edge, one);
    signal state_reg, state_next: state_type;
    signal p1_reg, p1_next: std_logic;
begin
    -- state register
    process(clk,reset)
    begin
        if (reset='1') then
            state_reg <= zero;
        elsif (clk'event and clk='1') then
            state_reg <= state_next;
        end if;
    end process;
    -- output buffer
    process(clk,reset)
    begin
```

# Edge detector

## SOLUTION

```
if (reset='1') then
    p1_reg <= '0';
elsif (clk'event and clk='1') then
    p1_reg <= p1_next;
end if;
end process;
-- next-state logic
process(state_reg,strobe)
begin
    case state_reg is
        when zero=>
            if strobe= '1' then
                state_next <= edge;
            else
                state_next <= zero;
            end if;
        when edge =>
            if strobe= '1' then
                state_next <= one;
```

# Edge detector

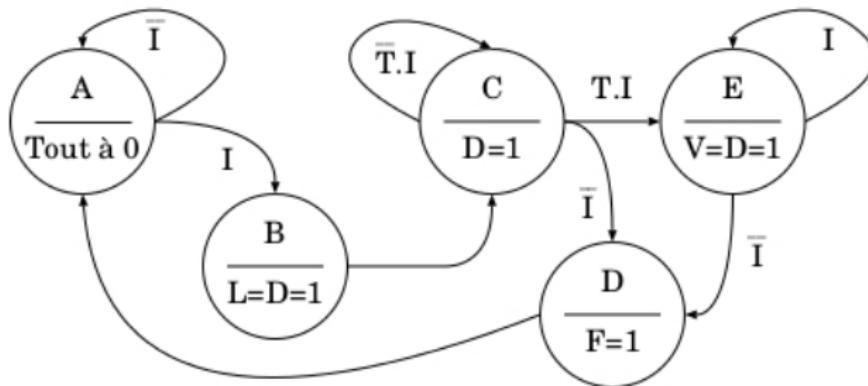
## SOLUTION

```
        else
            state_next <= zero;
        end if;
    when one =>
        if strobe= '1' then
            state_next <= one;
        else
            state_next <= zero;
        end if;
    end case;
end process;
-- look-ahead output logic
p1_next <= '1' when state_next=edge else
    '0';
-- output
p1 <= p1_reg;
end look_ahead_arch;
```

# EXEMPLE D'UTILISATION SUR LA CARTE

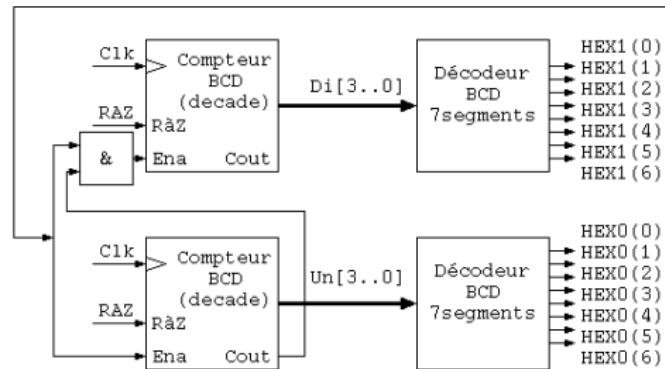
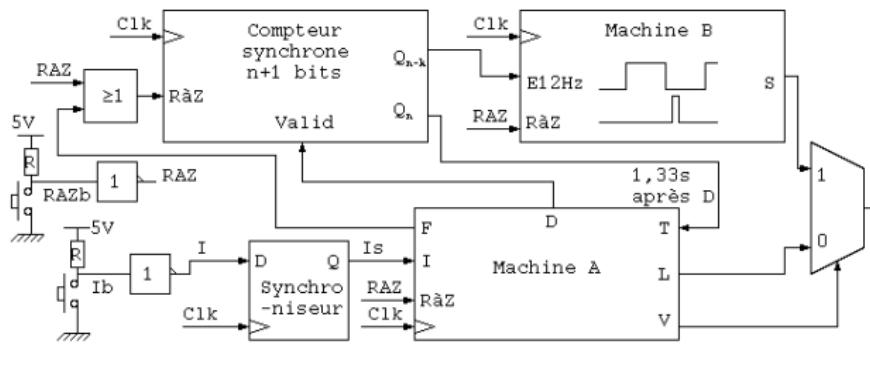
## BOUTON POUSSOIR AUTOMATIQUE

On souhaite réaliser un système de comptage par bouton poussoir qui devient automatique (à 24Hz : division par  $2^{21}$  de l'horloge à 50MHz) si le bouton est maintenu plus de 1,34s (division par  $2^{26}$  de l'horloge à 50MHz)



# EXEMPLE D'UTILISATION SUR LA CARTE

## BOUTON POUSSOIR AUTOMATIQUE



# COMPTEUR MODULO

## SOLUTION

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

entity compteur_modulo is
    generic (TAILLE : integer := 8;
              MODULO : integer := 10 );
    port(
        CLK      : in  std_logic;
        RESET    : in  std_logic;
        ENABLE   : in  std_logic;
        DOUT    : out std_logic_vector(TAILLE-1 downto 0);
        TC       : out std_logic);
end;
architecture behavior of compteur_modulo is begin
clk_proc:process(CLK)
    variable COUNT:unsigned(TAILLE-1 downto 0) := (others => '0');
begin
    if rising_edge(CLK) then
```

# COMPTEUR MODULO

## SOLUTION

```
if RESET = '1' then
    COUNT := (others=>'0');
else
    TC    <= '1';
    if ENABLE = '0' then
        COUNT := COUNT + 1;
        if COUNT = MODULO then
            COUNT := (others => '0');
            TC    <= '0';
        end if;
    end if;
end if;
DOUT <= std_logic_vector(COUNT);
end process clk_proc;
end behavior;
```

# COMPTEUR MODULO

## SOLUTION

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

entity compteurs is
    generic (TAILLE : integer := 8;
              MODULO : integer := 10 );
    port(
        CLK      : in  std_logic;
        RESET   : in  std_logic;
        ENABLE  : in  std_logic;
        DOUT1   : buffer std_logic_vector(TAILLE-1 downto 0);
        DOUT2   : buffer std_logic_vector(TAILLE-1 downto 0);
        SEG1    : out std_logic_vector(6 downto 0);
        SEG2    : out std_logic_vector(6 downto 0);
        TC      : out std_logic);
end;
```

# COMPTEUR MODULO

## SOLUTION

```
architecture arch_compteurs of compteurs is
    component compteur_modulo
        generic (TAILLE : integer := 8;
                  MODULO : integer := 10 );
        port(
            CLK      : in  std_logic;
            RESET   : in  std_logic;
            ENABLE  : in  std_logic;
            DOUT    : out std_logic_vector(TAILLE-1 downto 0);
            TC      : out std_logic);
    end component;
    component dec_7seg is
        port(hex_in : in  std_logic_vector(3 downto 0);
             segs     : out   std_logic_vector(6 downto 0));
    end component;
    signal TC1, TC2  : std_logic;
begin
    unit_1: compteur_modulo generic map (4,10) port
        map(CLK,RESET,ENABLE,DOUT1,TC1);
```

# COMPTEUR MODULO

## SOLUTION

```
unit_2: compteur_modulo generic map (4,10) port
  map(CLK,RESET,TC1,DOUT2,TC2);
seg_1: dec_7seg port map(DOUT1,SEG1);
seg_2: dec_7seg port map(DOUT2,SEG2);
  TC <= TC2;
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity top_compteurs is
  port (KEY : in std_logic_vector(3 downto 0);
        SW : in std_logic_vector(9 downto 0);
        LEDR : out std_logic_vector(9 downto 0);
        HEX0 : out std_logic_vector(0 to 6);
        HEX1 : out std_logic_vector(0 to 6);
        CLOCK_50 : in std_logic);
end;

architecture arch_top_compteurs of top_compteurs is
```

# COMPTEUR MODULO

## SOLUTION

```
component compteurs
generic (TAILLE : integer := 8;
          MODULO : integer := 10 );
port(
    CLK      : in  std_logic;
    RESET    : in  std_logic;
    ENABLE   : in  std_logic;
    DOUT1   : out std_logic_vector(TAILLE-1 downto 0);
    DOUT2   : out std_logic_vector(TAILLE-1 downto 0);
    SEG1    : out std_logic_vector(6 downto 0);
    SEG2    : out std_logic_vector(6 downto 0);
    TC      : out std_logic);
end component;
begin
    compteurs_1: compteurs generic map (4,10) port
        map(CLOCK_50,SW(0),KEY(0),
             LEDR(3 downto 0), LEDR(7 downto 4),
             HEX0,HEX1,LEDR(8));
    LEDR(9) <= '0';
```

# COMPTEUR MODULO

## SOLUTION

```
end;
```

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

entity compteurs6 is
generic (TAILLE : integer := 8;
         MODULO : integer := 10 );
port(
    CLK      : in  std_logic;
    RESET    : in  std_logic;
    ENABLE   : in  std_logic;
    SEG1    : out std_logic_vector(6 downto 0);
    SEG2    : out std_logic_vector(6 downto 0);
    SEG3    : out std_logic_vector(6 downto 0);
    SEG4    : out std_logic_vector(6 downto 0);
    SEG5    : out std_logic_vector(6 downto 0);
```

# COMPTEUR MODULO

## SOLUTION

```
SEG6 : out std_logic_vector(6 downto 0);
TC      : out std_logic);
end;

architecture arch_compteurs6 of compteurs6 is
component compteur_modulo
generic (TAILLE : integer := 8;
         MODULO : integer := 10 );
port(
    CLK      : in  std_logic;
    RESET    : in  std_logic;
    ENABLE   : in  std_logic;
    DOUT    : out std_logic_vector(TAILLE-1 downto 0);
    TC      : out std_logic);
end component;
component dec_7seg is
port(hex_in : in  std_logic_vector(3 downto 0);
      segs    : out   std_logic_vector(6 downto 0));
end component;
```

# COMPTEUR MODULO

## SOLUTION

```
signal TC1, TC2, TC3, TC4, TC5, TC6 : std_logic;
signal DOUT1,DOUT2,DOUT3,DOUT4,DOUT5,DOUT6 :
    std_logic_vector(TAILLE-1 downto 0);
begin
    unit_1: compteur_modulo generic map (4,10) port
        map(CLK,RESET,ENABLE,DOUT1,TC1);
    unit_2: compteur_modulo generic map (4,10) port
        map(CLK,RESET,TC1,DOUT2,TC2);
    unit_3: compteur_modulo generic map (4,10) port
        map(CLK,RESET,TC2,DOUT3,TC3);
    unit_4: compteur_modulo generic map (4,10) port
        map(CLK,RESET,TC3,DOUT4,TC4);
    unit_5: compteur_modulo generic map (4,10) port
        map(CLK,RESET,TC4,DOUT5,TC5);
    unit_6: compteur_modulo generic map (4,10) port
        map(CLK,RESET,TC5,DOUT6,TC6);
    seg_1: dec_7seg port map(DOUT1,SEG1);
    seg_2: dec_7seg port map(DOUT2,SEG2);
    seg_3: dec_7seg port map(DOUT3,SEG3);
```

# COMPTEUR MODULO

## SOLUTION

```
seg_4: dec_7seg port map(DOUT4,SEG4);
seg_5: dec_7seg port map(DOUT5,SEG5);
seg_6: dec_7seg port map(DOUT6,SEG6);
TC <= TC6;
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity top_compteurs6 is
    port (KEY : in std_logic_vector(3 downto 0);
          SW : in std_logic_vector(9 downto 0);
          HEX0 : out std_logic_vector(0 to 6);
          HEX1 : out std_logic_vector(0 to 6);
          HEX2 : out std_logic_vector(0 to 6);
          HEX3 : out std_logic_vector(0 to 6);
          HEX4 : out std_logic_vector(0 to 6);
          HEX5 : out std_logic_vector(0 to 6);
          CLOCK_50 : in std_logic);
end;
```

# COMPTEUR MODULO

## SOLUTION

```
architecture arch_top_compteurs6 of top_compteurs6 is
component compteurs6
generic (TAILLE : integer := 8;
         MODULO : integer := 10 );
port(
    CLK      : in  std_logic;
    RESET   : in  std_logic;
    ENABLE   : in  std_logic;
    SEG1 : out std_logic_vector(6 downto 0);
    SEG2 : out std_logic_vector(6 downto 0);
    SEG3 : out std_logic_vector(6 downto 0);
    SEG4 : out std_logic_vector(6 downto 0);
    SEG5 : out std_logic_vector(6 downto 0);
    SEG6 : out std_logic_vector(6 downto 0);
    TC      : out std_logic);
end component;
begin
```



# COMPTEUR MODULO

## SOLUTION

```
compteurs_1: compteurs6 generic map (4,10) port
  map(CLOCK_50,SW(0),KEY(0),
       HEX0,HEX1,HEX2,HEX3,HEX4,HEX5);
end;
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

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

entity compteur_modulo_mod is
    generic (TAILLE : integer := 8;
              MODULO : integer := 10 );
    port(
        CLK      : in  std_logic;
        RESET   : in  std_logic;
        ENABLE  : in  std_logic;
        DOUT    : out std_logic_vector(TAILLE-1 downto 0);
        TC      : out std_logic);
end;
architecture behavior of compteur_modulo_mod is
signal enable_del, enable_edge : std_logic;
begin

clk_proc:process(CLK)
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
variable COUNT:unsigned(TAILLE-1 downto 0) := (others => '0');
begin
    if rising_edge(CLK) then
        if RESET = '1' then
            COUNT := (others=>'0');
            enable_del <= '0';
        else
            enable_del <= ENABLE;
            TC      <= '1';
            if enable_edge = '0' then
                COUNT := COUNT + 1;
                if COUNT = MODULO then
                    COUNT := (others => '0');
                    TC      <= '0';
                end if;
            end if;
        end if;
    end if;
    DOUT <= std_logic_vector(COUNT);
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
end process clk_proc;  
enable_edge <= not(not ENABLE and enable_del);  
end behavior;
```

```
library IEEE;  
use IEEE.std_logic_1164.all;  
use IEEE.numeric_std.all;  
  
entity compteurs6_mod is  
    generic (TAILLE : integer := 8;  
             MODULO : integer := 10 );  
    port(  
        CLK      : in  std_logic;  
        RESET   : in  std_logic;  
        ENABLE  : in  std_logic;  
        SEG1    : out std_logic_vector(6 downto 0);  
        SEG2    : out std_logic_vector(6 downto 0);  
        SEG3    : out std_logic_vector(6 downto 0);
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
SEG4 : out std_logic_vector(6 downto 0);
SEG5 : out std_logic_vector(6 downto 0);
SEG6 : out std_logic_vector(6 downto 0);
TC      : out std_logic);
end;

architecture arch_compteurs6_mod of compteurs6_mod is
component compteur_modulo_mod
generic (TAILLE : integer := 8;
         MODULO : integer := 10 );
port(
    CLK      : in  std_logic;
    RESET   : in  std_logic;
    ENABLE  : in  std_logic;
    DOUT    : out std_logic_vector(TAILLE-1 downto 0);
    TC      : out std_logic);
end component;
component dec_7seg is
port(hex_in : in  std_logic_vector(3 downto 0);
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
    segs      : out    std_logic_vector(6 downto 0));
end component;
signal TC1, TC2, TC3, TC4, TC5, TC6 :  std_logic;
signal DOUT1,DOUT2,DOUT3,DOUT4,DOUT5,DOUT6 :
    std_logic_vector(TAILLE-1 downto 0);
begin
    unit_1: compteur_modulo_mod generic map (4,10) port
        map(CLK,RESET,ENABLE,DOUT1,TC1);
    unit_2: compteur_modulo_mod generic map (4,10) port
        map(CLK,RESET,TC1,DOUT2,TC2);
    unit_3: compteur_modulo_mod generic map (4,10) port
        map(CLK,RESET,TC2,DOUT3,TC3);
    unit_4: compteur_modulo_mod generic map (4,10) port
        map(CLK,RESET,TC3,DOUT4,TC4);
    unit_5: compteur_modulo_mod generic map (4,10) port
        map(CLK,RESET,TC4,DOUT5,TC5);
    unit_6: compteur_modulo_mod generic map (4,10) port
        map(CLK,RESET,TC5,DOUT6,TC6);
    seg_1: dec_7seg port map(DOUT1,SEG1);
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
seg_2: dec_7seg port map(DOUT2,SEG2);
seg_3: dec_7seg port map(DOUT3,SEG3);
seg_4: dec_7seg port map(DOUT4,SEG4);
seg_5: dec_7seg port map(DOUT5,SEG5);
seg_6: dec_7seg port map(DOUT6,SEG6);
TC <= TC6;
end;
```

```
library ieee;
use ieee.std_logic_1164.all;
entity top_compteurs6_mod is
    port (KEY : in std_logic_vector(3 downto 0);
          SW   : in std_logic_vector(9 downto 0);
          HEX0 : out std_logic_vector(0 to 6);
          HEX1 : out std_logic_vector(0 to 6);
          HEX2 : out std_logic_vector(0 to 6);
          HEX3 : out std_logic_vector(0 to 6);
          HEX4 : out std_logic_vector(0 to 6);
          HEX5 : out std_logic_vector(0 to 6);
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
CLOCK_50 : in std_logic);  
end;  
  
architecture arch_top_compteurs6_mod of top_compteurs6_mod is  
component compteurs6_mod  
generic (TAILLE : integer := 8;  
        MODULO : integer := 10 );  
port(  
    CLK      : in  std_logic;  
    RESET   : in  std_logic;  
    ENABLE   : in  std_logic;  
    SEG1 : out std_logic_vector(6 downto 0);  
    SEG2 : out std_logic_vector(6 downto 0);  
    SEG3 : out std_logic_vector(6 downto 0);  
    SEG4 : out std_logic_vector(6 downto 0);  
    SEG5 : out std_logic_vector(6 downto 0);  
    SEG6 : out std_logic_vector(6 downto 0);  
    TC      : out std_logic);  
end component;
```

# COMPTEUR MODULO « VISIBLE »

## SOLUTION

```
begin
    compteurs_1: compteurs6_mod generic map (4,10) port
        map(CLOCK_50,SW(0),KEY(0),
              HEX0,HEX1,HEX2,HEX3,HEX4,HEX5);
end;
```



# SOMMAIRE

- 6 Mémoires internes au FPGA
  - Mémoires internes au FPGA

# ROM

- Les FPGA des familles Cyclone (I) étaient les derniers à inclure des ROM totalement asynchrones (adresses et données)
- Depuis les Cyclones II, les adresses sont nécessairement synchrones
- Les données peuvent être synchrones ou asynchrones
- Le contenu est précisé par un fichier .mif ou .hex explicite ou généré par une fonction VHDL
- Il existe une version double port, permettant de lire simultanément le contenu à deux adresses distinctes



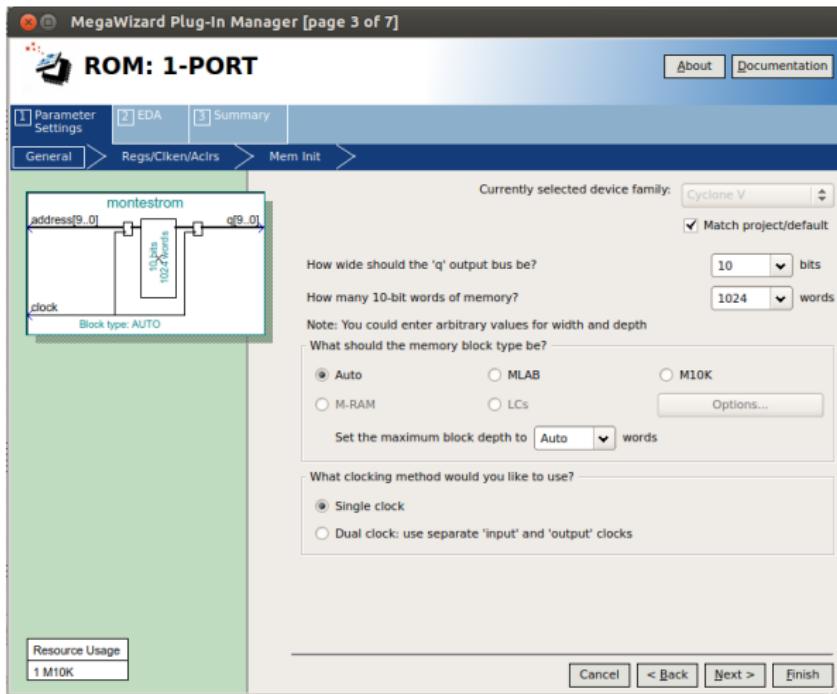
# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

- Pour pouvoir créer une ROM de façon graphique il faut au préalable créer un fichier d'initialisation du contenu (requis par l'outil)
- Créer un nouveau fichier d'initialisation **File** → **New** → **Memory Initialization File**
- Choisir 1024 mots de 10 bits
- Enregistrer sous **montestrom.mif**
- Clic droit dans le tableau crée **Custom Fill Cells**
- Remplir par exemple avec un compteur partant de 0
- On peut alors démarrer l'assistant de création de composants **Tools** → **IP Catalog**
- Choisir une mémoire de type **ROM: 1-PORT** et créer un nouveau composant VHDL nommé **montestrom.vhd**



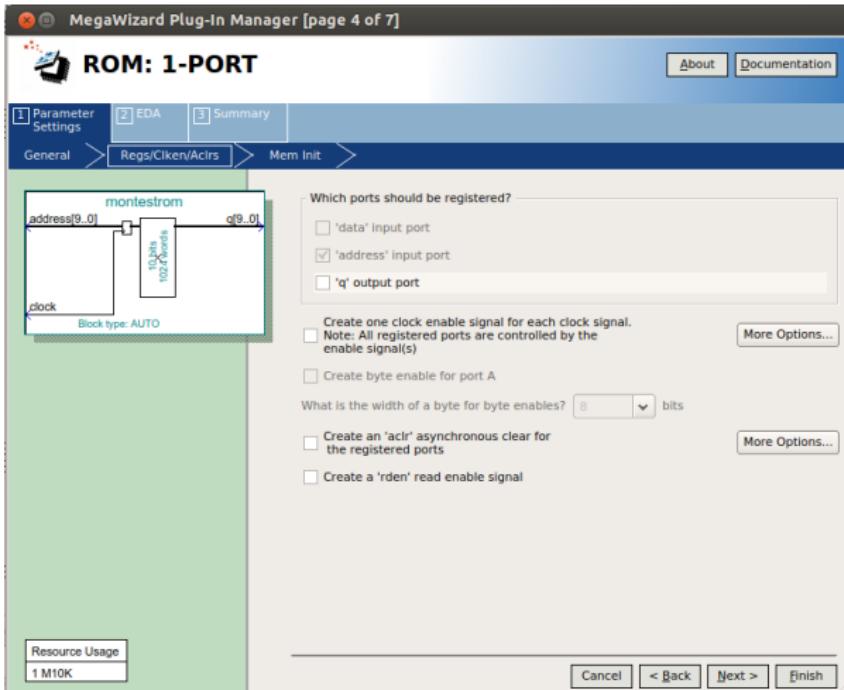
# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

## CONFIGURATION GÉNÉRALE





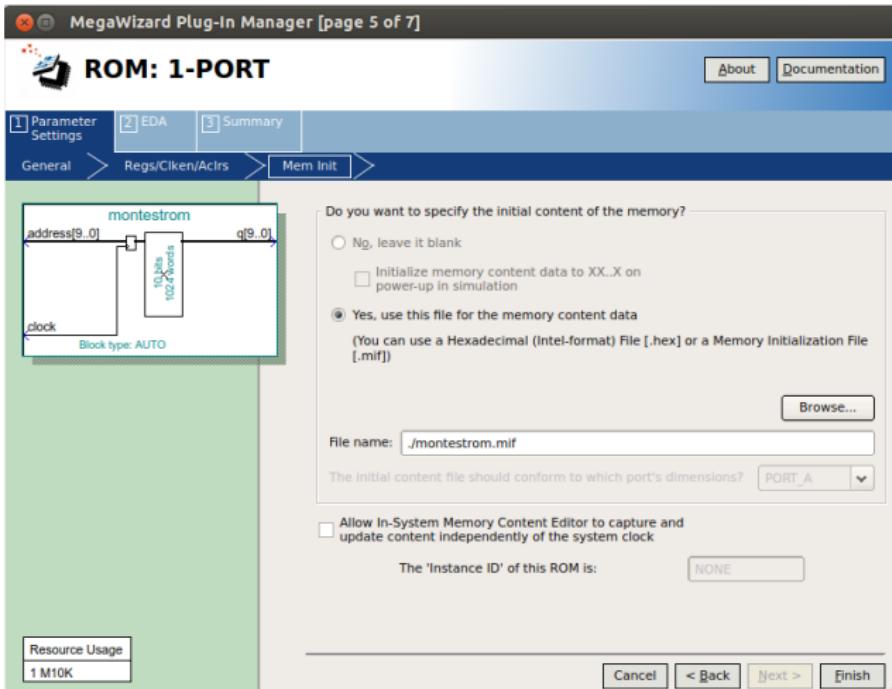
# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE CONFIGURATION





# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE

## CONFIGURATION





# ROM : CRÉATION PAR ASSISTANT GRAPHIQUE CONFIGURATION

**MegaWizard Plug-In Manager [page 7 of 7]**

## ROM: 1-PORT

Parameter Settings EDA Summary

Turn on the files you wish to generate. A gray checkmark indicates a file that is automatically generated, and a green checkmark indicates an optional file. Click Finish to generate the selected files. The state of each checkbox is maintained in subsequent MegaWizard Plug-In Manager sessions.

The MegaWizard Plug-In Manager creates the selected files in the following directory:  
`/home/berville5/tmp/Quartus/TelecomNancy/exemple1/`

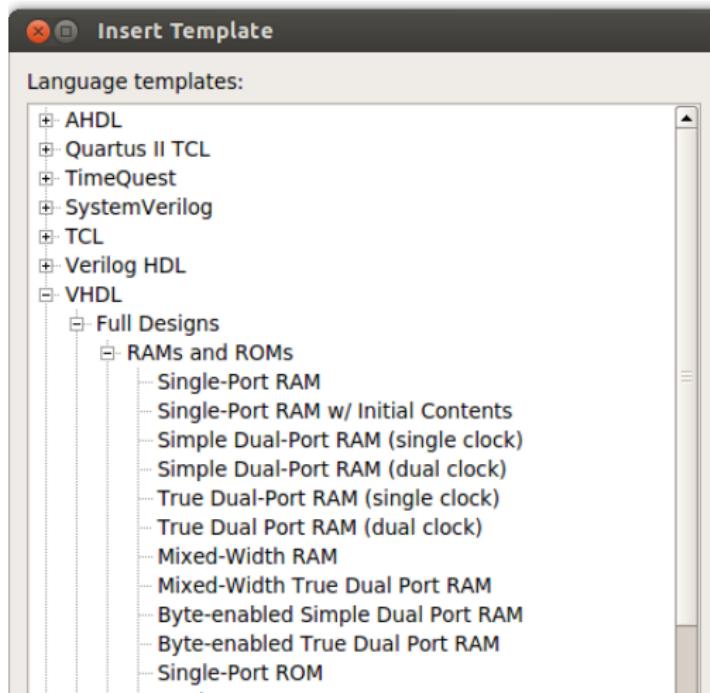
File	Description
<input checked="" type="checkbox"/> montestrom.vhd	Variation file
<input type="checkbox"/> montestrom.inc	AHDL Include file
<input type="checkbox"/> montestrom.cmp	VHDL component declaration file
<input type="checkbox"/> montestrom.bsf	Quartus II symbol file
<input checked="" type="checkbox"/> montestrom_inst.vhd	Instantiation template file

Resource Usage  
1 M10K

Cancel Back Next Finish

```
graph LR; subgraph montestrom [montestrom]; direction TB; A[address[9..0]] --> B[10 bits<br/>1024 Words]; B --> C[q[9..0]]; end; D[clock]
```

# ROM : CRÉATION PAR MODÈLE VHDL



# ROM : CRÉATION PAR MODÈLE VHDL

```
entity single_port_rom is
  generic
  (
    DATA_WIDTH : natural := 8;
    ADDR_WIDTH : natural := 8
  );
  port
  (
    clk      : in std_logic;
    addr    : in natural range 0 to 2**ADDR_WIDTH - 1;
    q       : out std_logic_vector((DATA_WIDTH -1) downto 0)
  );
end entity;
architecture rtl of single_port_rom is
  -- Build a 2-D array type for the RoM
  subtype word_t is std_logic_vector((DATA_WIDTH-1) downto 0);
  type memory_t is array(2**ADDR_WIDTH-1 downto 0) of word_t;

  function init_rom
    return memory_t is
```

# ROM : CRÉATION PAR MODÈLE VHDL

```
variable tmp : memory_t := (others => (others => '0'));
begin
    for addr_pos in 0 to 2**ADDR_WIDTH - 1 loop
        -- Initialize each address with the address itself
        tmp(addr_pos) := std_logic_vector(to_unsigned(addr_pos,
DATA_WIDTH));
    end loop;
    return tmp;
end init_rom;
-- Declare the ROM signal and specify a default value. Quartus II
-- will create a memory initialization file (.mif) based on the
-- default value.
signal rom : memory_t := init_rom;
begin
process(clk)
begin
if(rising_edge(clk)) then
    q <= rom(addr);
end if;
end process;
```



# ROM : CRÉATION PAR MODÈLE VHDL

```
end rtl;
```



## A FAIRE

- ➊ Instancier la ROM précédemment créée dans une entité VHDL
- ➋ Tester son fonctionnement, par simulation
- ➌ Tester son fonctionnement en reliant les adresses à un compteur modulo 1024 et en affichant les adresses et les données en hexadécimal sur les afficheurs 7 segments de la carte (3 pour les adresses et 3 pour les données)



## A FAIRE

- ➊ Instancier la ROM précédemment créée dans une entité VHDL
- ➋ Tester son fonctionnement, par simulation
- ➌ Tester son fonctionnement en reliant les adresses à un compteur modulo 1024 et en affichant les adresses et les données en hexadécimal sur les afficheurs 7 segments de la carte (3 pour les adresses et 3 pour les données)

# SOMMAIRE

## 7 Processeur

- Processeur

# PROCESSEUR

## INTRODUCTION

Un processeur simple composé de :

- Une unité arithmétique et logique (ALU) d'une largeur de 4 bits (addition, OU exclusif et ET logique)
- Les instructions sont codées sur 8 bits et leurs adresses sur 7 bits.
- Les données ont un format de 4 bits et occupent 16 adresses.

Le jeu d'instruction :

# PROCESSEUR

## INTRODUCTION

Mnémonique	Opcode	Description
load Rd	0100d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	charge l'accumulateur avec le mot d'adresse d[3..0]
load #d	0001d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	charge l'accumulateur avec d[3..0]
store Rd	0011d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	stocke le contenu de l'accumulateur à l'adresse d[3..0]
add Rd	0101d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	additionne l'accumulateur avec la retenue (C) et le mot d'adresse d[3..0]. Résultat dans l'accumulateur et C
add #d	0010d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	additionne l'accumulateur avec la retenue (C) et le mot d[3..0]. Résultat dans l'accumulateur et C
xor Rd	0110d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	effectue un OU exclusif entre le contenu de l'accumulateur et le mot d'adresse d[3..0]. Résultat dans l'accumulateur
test Rd	0111d <sub>3</sub> d <sub>2</sub> d <sub>1</sub> d <sub>0</sub>	effectue un ET logique entre le contenu de l'accumulateur et le mot d'adresse d[3..0]. Résultat dans l'indicateur de nullité (Z)
clear_c	00000000	met C à 0
set_c	00000001	met C à 1
skip_c	00000010	saute la prochaine instruction (en incrémentant le compteur d'instruction (PC)) si C=1
skip_z	00000011	saute la prochaine instruction (en incrémentant le compteur d'instruction (PC)) si Z=1
jump #a	1a <sub>6</sub> a <sub>5</sub> a <sub>4</sub> a <sub>3</sub> a <sub>2</sub> a <sub>1</sub> a <sub>0</sub>	charge PC avec a[7..0] et exécute à partir de cette adresse

# PROCESSEUR

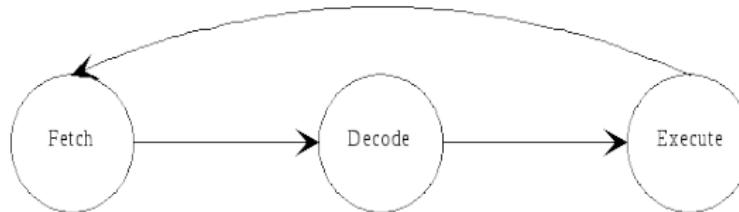
## INTRODUCTION

Chaque instruction se décompose en 3 phases (trois cycles machine) :

- **Recherche d'instructions (Fetch)** : lecture de l'instruction pointée par PC et stockage dans le registre d'instructions (IR).
- **Décodage d'instructions (Decode)** : détermination du type d'instruction et recherche des opérandes.
- **Execution (Execute)** : réalisation des opérations et stockage éventuel des résultats.

# PROCESSEUR

## INTRODUCTION



- Signaux positionnés (mis à 1) lors des différentes phases d'exécution :
  - ▷ **Fetch** : `ld_ir, inc_pc`.
  - ▷ **Decode** : Si (`[IR7..IR4]=load_Rd ou add_Rd ou xor_Rd ou test_Rd`) alors `ld_ir_lsn`, sinon rien.
  - ▷ **Execute** : dépend des instructions, par exemple : Affecter `[aop2..aop0]` en fonction de l'instruction en cours.

# PROCESSEUR

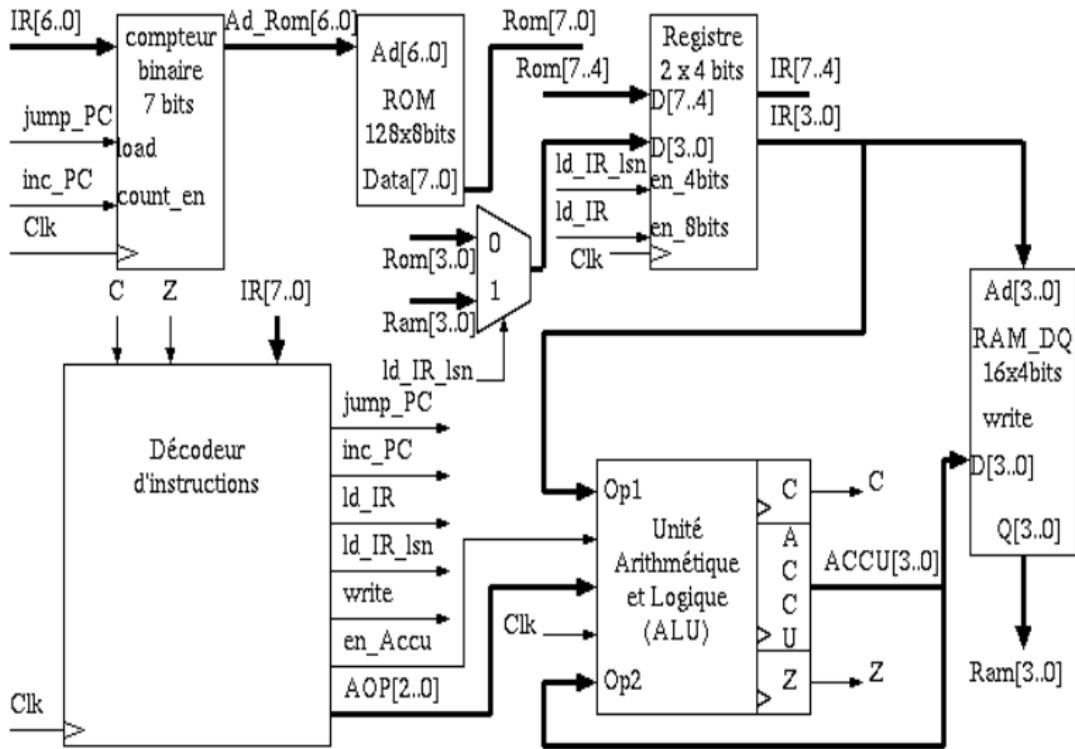
## INTRODUCTION

Si ( $IR=skip\_c$ ) alors  $inc\_pc=C$ , si ( $IR=jump$ ) alors  $jump\_pc=1$ , si ( $IR=store$ ) alors  $write=1$ . . . etc. Il faut analyser la phase « execute » de chaque instruction pour être sûr de ne rien oublier.

Instructions	Fetch	Decode	Execute
<code>load #d</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$ACC \leftarrow [IR3..IR0]$
<code>load Rd</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$	$[IR3..IR0] \leftarrow ([IR3..IR0])$	$ACC \leftarrow [IR3..IR0]$
<code>store Rd</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$([IR3..IR0]) \leftarrow ACC$
<code>add #d</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$ACC \leftarrow ACC + [IR3..IR0] + C$
<code>add Rd</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$	$[IR3..IR0] \leftarrow ([IR3..IR0])$	$ACC \leftarrow ACC + [IR3..IR0] + C$
<code>xor Rd</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$	$[IR3..IR0] \leftarrow ([IR3..IR0])$	$ACC \leftarrow ACC \oplus [IR3..IR0]$
<code>test Rd</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$	$[IR3..IR0] \leftarrow ([IR3..IR0])$	$Z \leftarrow ACC \& [IR3..IR0]$
<code>clear_c</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$C \leftarrow 0$
<code>set_c</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$C \leftarrow 1$
<code>skip_c</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$PC \leftarrow PC+C$
<code>skip_z</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$PC \leftarrow PC+Z$
<code>jump #a</code>	$IR \leftarrow (PC)$ ; $PC \leftarrow PC+1$		$PC \leftarrow [IR6..IR0]$

# PROCESSEUR

## INTRODUCTION



# PROCESSEUR

## INTRODUCTION

Ressources nécessaires :

- La mémoire programme (ROM) comporte 128 mots de 8 bits.
- La mémoire de données (RAM) comporte 16 mots de 4 bits.
- Le compteur programme (PC) 7 bits stocke l'adresse de l'instruction en cours.
- Le registre d'instructions (IR) 8 bits stocke l'instruction et l'opérande en cours.
- La retenue C et l'indicateur de nullité Z sont stockés dans 2 bascules.
- Une unité arithmétique et logique (ALU) traite 2 mots de 4 bits et fourni un résultat sur 4 bits (+éventuelle retenue).

# PROCESSEUR

## INTRODUCTION

- Un accumulateur (registre 4 bits) qui stocke les résultats de l'ALU.
- L'ALU opère sur le contenu de l'accumulateur et les 4 bits de poids faible (LSN) de IR.
- La seule source d'écriture des données est l'accumulateur.
- Le PC est un registre (ou compteur) qui peut être incrémenté ou chargé avec [IR6..IR0].

Le décodeur d'instructions :

- A partir des opcodes [IR7..IR4] et de C et Z (et de la phase d'exécution) il fournit les signaux de contrôle permettant aux ressources d'exécuter l'instruction.
- Les signaux sont :

# PROCESSEUR

## INTRODUCTION

- ▷ **jump\_pc** : contrôle le multiplexeur alimentant le PC. La nouvelle valeur du PC est [IR6..IR0] s'il est à 1, sinon c'est PC+1.
- ▷ **inc\_pc** : contrôle l'incrémenteur du PC. S'il est à 1, on incrémente le PC, sinon le PC est inchangé.
- ▷ **ld\_ir** : contrôle le chargement de IR. IR est chargé avec le contenu de la mémoire programme (pointée par PC) s'il est à 1, sinon inchangé.
- ▷ **ld\_ir\_lsn** : contrôle le chargement des 4 bits de poids faible de IR. [IR3..IR0] est chargé avec le contenu de la mémoire de données pointée par [IR3..IR0] s'il est à 1, sinon inchangé.
- ▷ **en\_Accu** : valide le stockage de la sortie de l'ALU dans l'accumulateur, la retenue ou Z
- ▷ **[aop2..aop0]** : contrôlent l'ALU, l'accumulateur et C :

# PROCESSEUR

## INTRODUCTION

- 001 : les données [IR3..IR0] passent directement dans l'accumulateur.
  - 010 : l'ALU additionne.
  - 011 : l'ALU effectue un OU exclusif.
  - 100 : l'ALU effectue un ET logique.
  - 101 : mise à 1 de C.
  - 110 : mise à 0 de C.
- ▷ **write** : contrôle l'écriture dans la mémoire de données. Les données de l'accumulateur peuvent être écrites dans la RAM s'il est à 1, sinon les données de la RAM peuvent être lues.

# EXERCICE

## DÉCODEUR D'INSTRUCTION

- ➊ Réaliser le décodeur d'instruction présenté suivant le cahier des charges énoncé
- ➋ Tout d'abord, déterminer le type de la machine d'états à utiliser. Justifier votre réponse
- ➌ Tester votre décodeur d'instruction pour plusieurs types d'instructions.
- ➍ Vérifier en simulation le bon fonctionnement du décodeur d'instruction

# ROM

## CONTENU

```
tmp(0) := x"00";      -- clear_c
tmp(1) := x"40";      -- charge R0
tmp(2) := x"52";      -- add R2
tmp(3) := x"34";      -- store R4
tmp(4) := x"41";      -- load R1
tmp(5) := x"53";      -- add R3
tmp(6) := x"35";      -- store R5
tmp(7) := x"87";      -- halt: jump halt
```

# RAM

## CONTENU

```
tmp(0) := x"F";
tmp(1) := x"8";
tmp(2) := x"2";
tmp(3) := x"7";
tmp(4) := x"0";
tmp(5) := x"0";
tmp(6) := x"0";
tmp(7) := x"0";
```