LSI2

Méthodologie de conception des circuits intégrés VLSI

De l'idée à l'ASIC

Introduction

I. Technologie des CIs

II. Méthodologie de conception des ASIC

III. Conception synchrone des circuits

IV. Synthèse logique à partir de VHDL

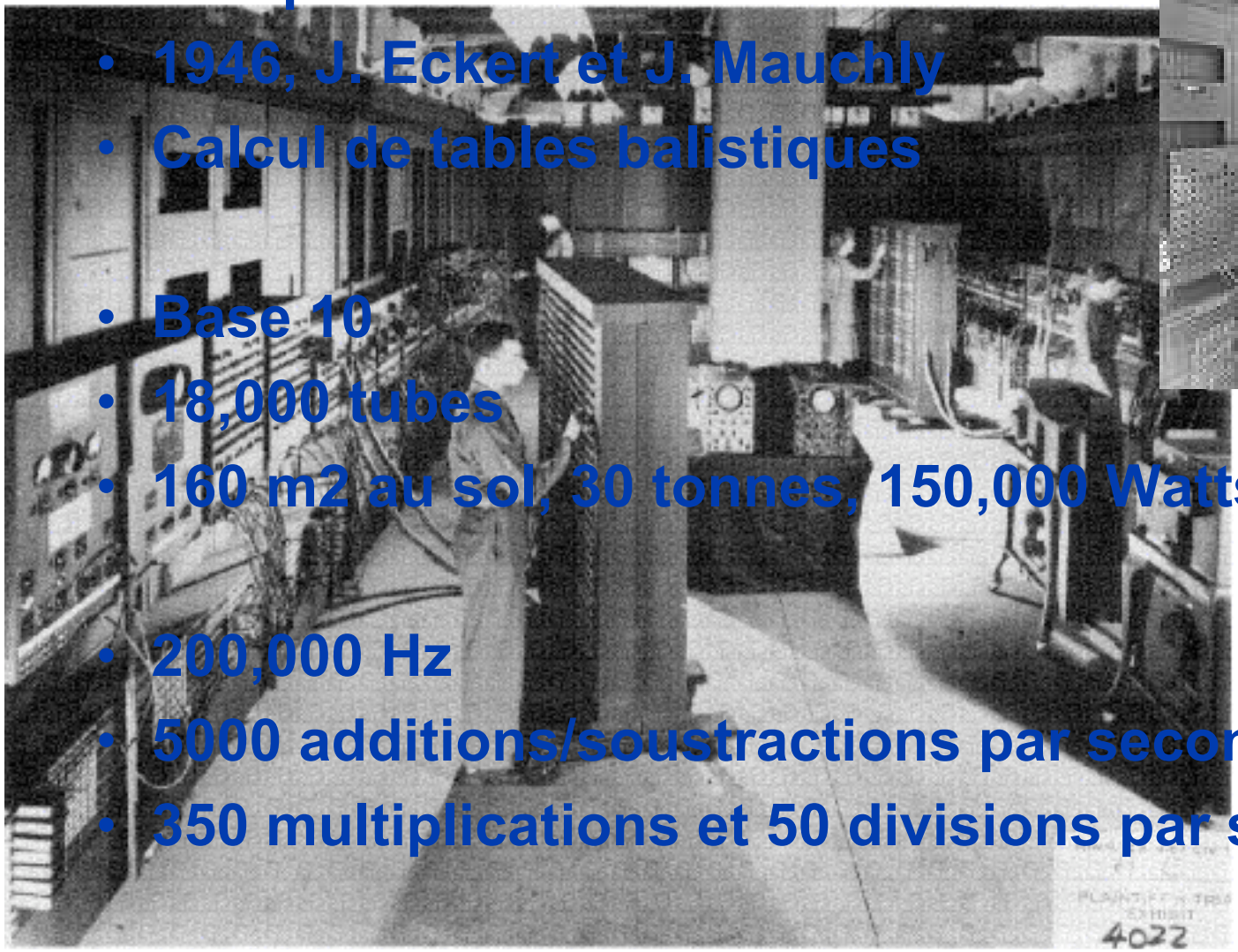
V. Projet de réalisation d'un ASIC

Thomas GUIHAL
thomas.guihal@irisa.fr
http://r2d2.enssat.fr/

UNIVERSITE DE RENNES 1

ENIAC - premier ordinateur *électronique*

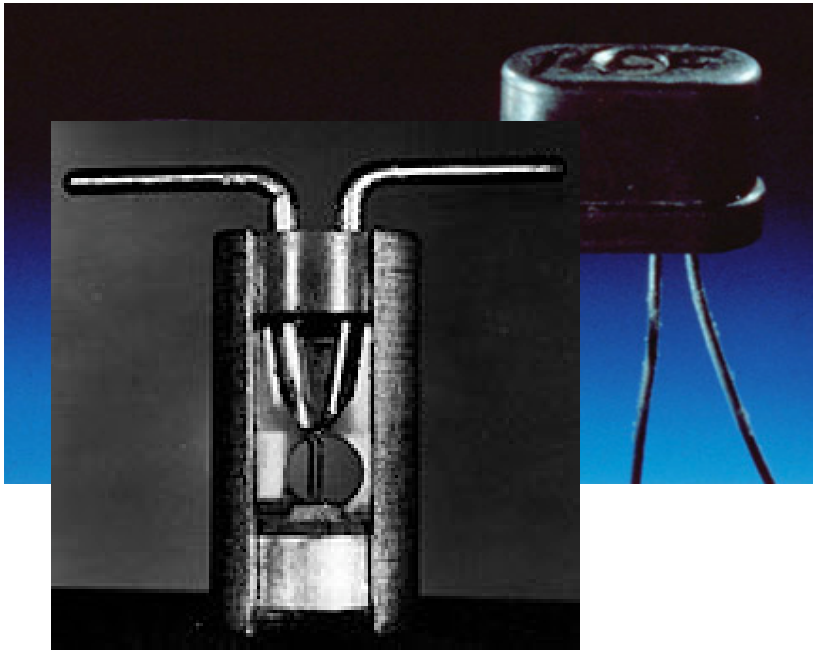
- ENIAC : Electronic Numerical Integrator and Computer
- 1946, J. Eckert et J. Mauchly
- Calcul de tables balistiques
- Base 10
- 18,000 tubes
- 160 m2 au sol, 30 tonnes, 150,000 Watts
- 200,000 Hz
- 5000 additions/soustractions par seconde
- 350 multiplications et 50 divisions par seconde

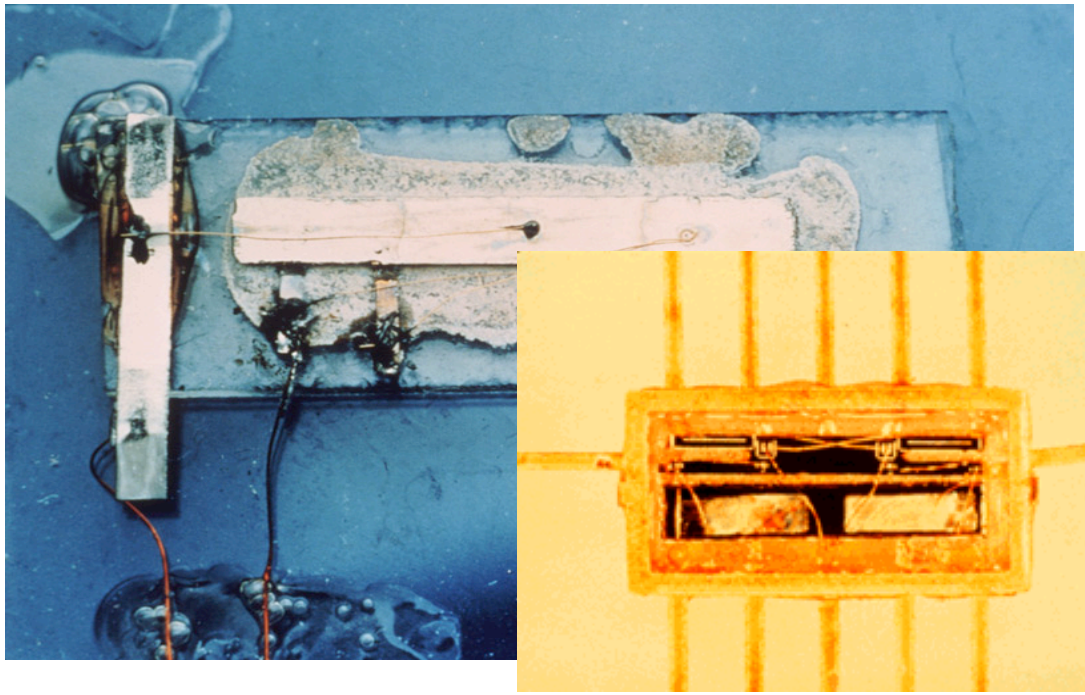


1958 Premier circuit intégré

- **1947 : W. Schockley (Bell Labs) invente le transistor**

  (prix nobel de physique 1956)





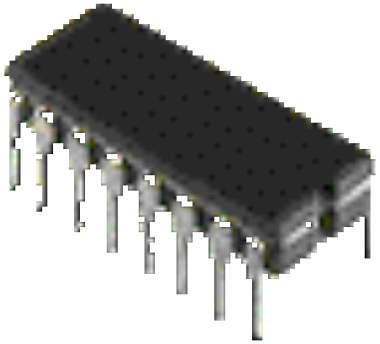
- **1958 : J. Kilby (Texas Inst.) conçoit le premier C.I.**
  - Transistors, diodes, condensateurs, fils, … agencés sur une mince plaque de silicium

"I perceived that a method for low-cost production of electronic circuits was in hand.... that instead of merely being able to build things smaller, we could fabricate entire networks in one sequence, and that we had extended the transistor's capability as a fundamental electronics tool."

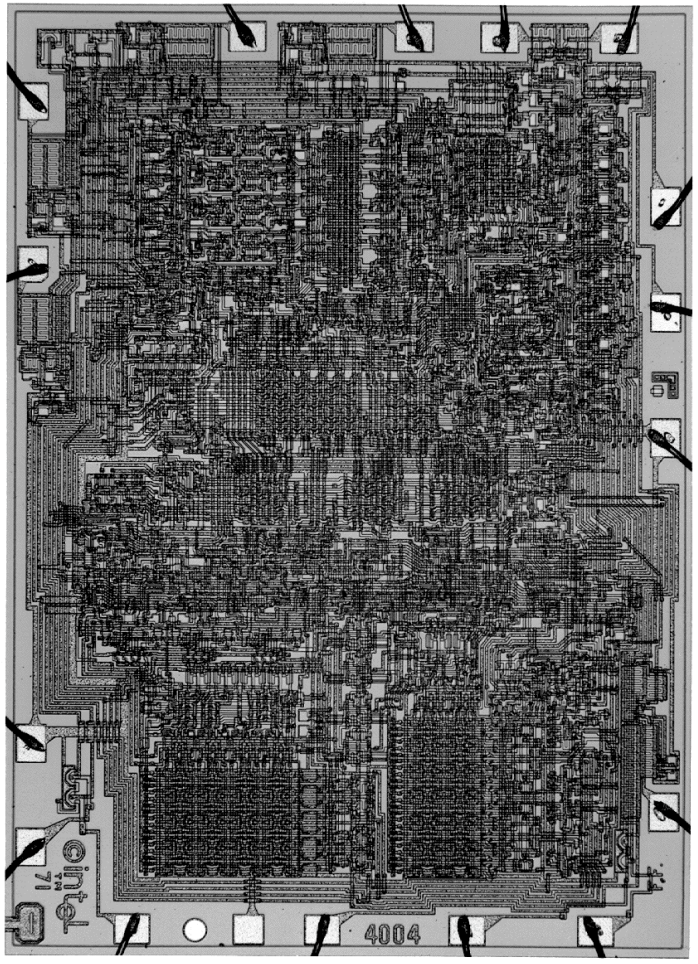
Jack Kilby, 1958, prix nobel de physique 2000

Le premier microprocesseur

- Intel 4004
- 1971
- 108 kHz
- 4 bits
- 1200 FF
- 0,06 MOPS



- 10 microns
- 2300 transistors
- 640 adressable bytes



Puis tout s'accélère

- 1970 Mémoire 4Kbits MOS
- 1972 1er processeur : 4004 (Intel), techno. NMOS
- 1977 16K DRAM et 4K SRAM en production
- 1979 64K DRAM en production
- 1980 Intel® Processeur x86
- 1984 Intel® Processeur 80286 (PC AT)
- 1986 1 mégabit DRAM
- 1988 TI/Hitachi 16-megabit DRAM
- 1990 Intel® Processeur 80286 (fonctions multimédia)
- 1990 Wafer de 20cm en production
- 1991 4 mégabit DRAM en production
- 1993 Intel® Processeur Pentium
- 1997 Intel® Pentium® II Processor
- 1999 Intel® Pentium® III Processor
- 2000 Intel® Pentium® 4 Processor
- 2002 Intel® Itanium™ 2 Processor
- 2003 Intel® Pentium® M Processor, 1 gigabit DRAM

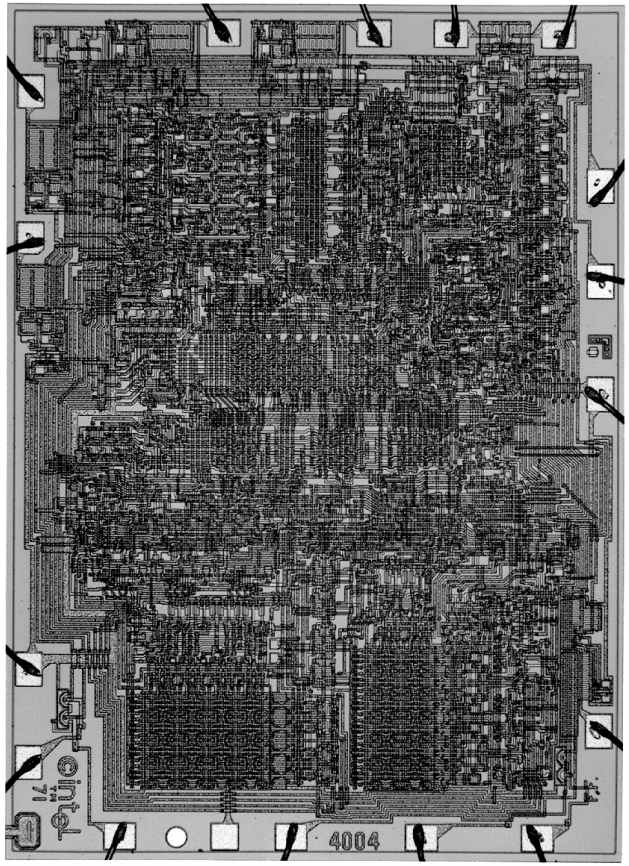
25 ans d'évolutions

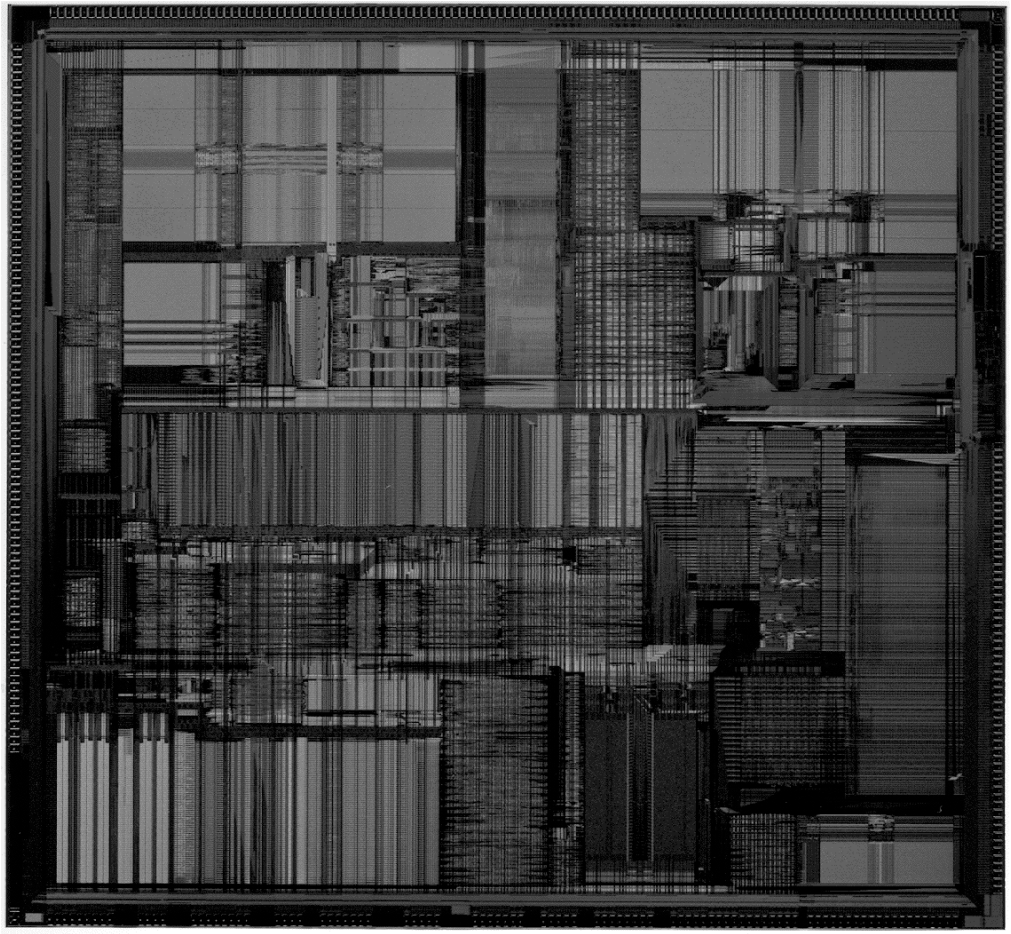
INTEL 4004 (1971)

Données sur 4 bits

2300 transistors, 10 microns

0,06 MOPS, 108 kHz





INTEL Pentium II (1996)

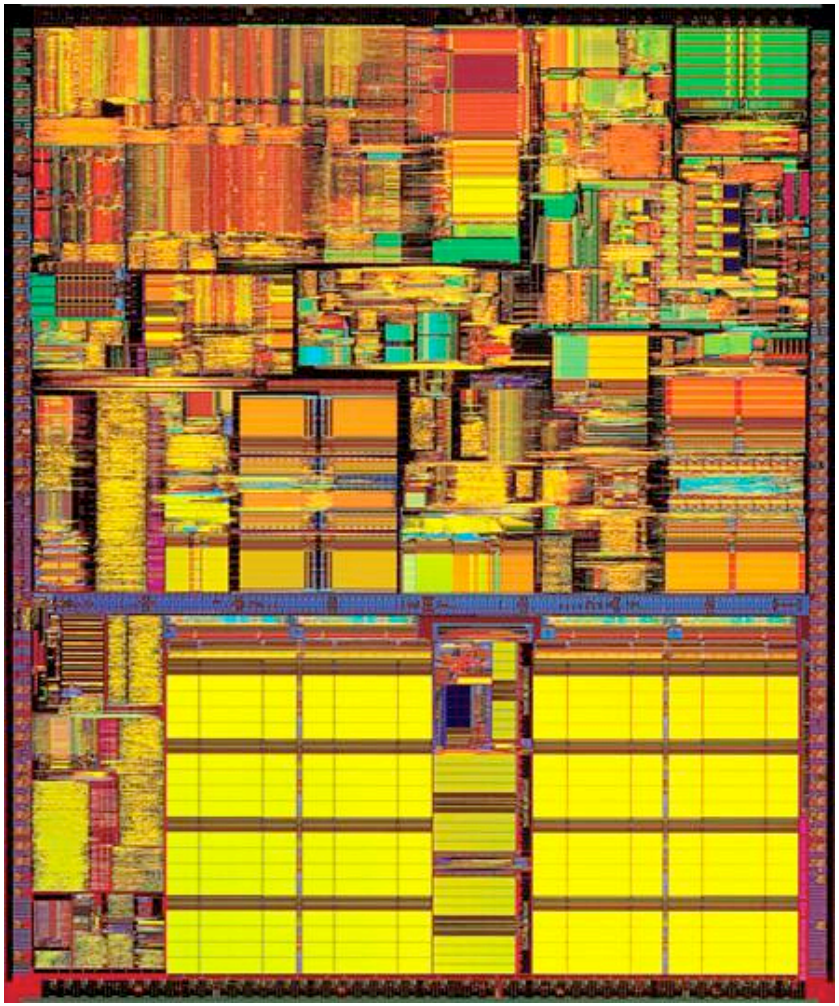
Données sur 32 bits

5.5M de transistors, 0.35µ, 2 cm^2

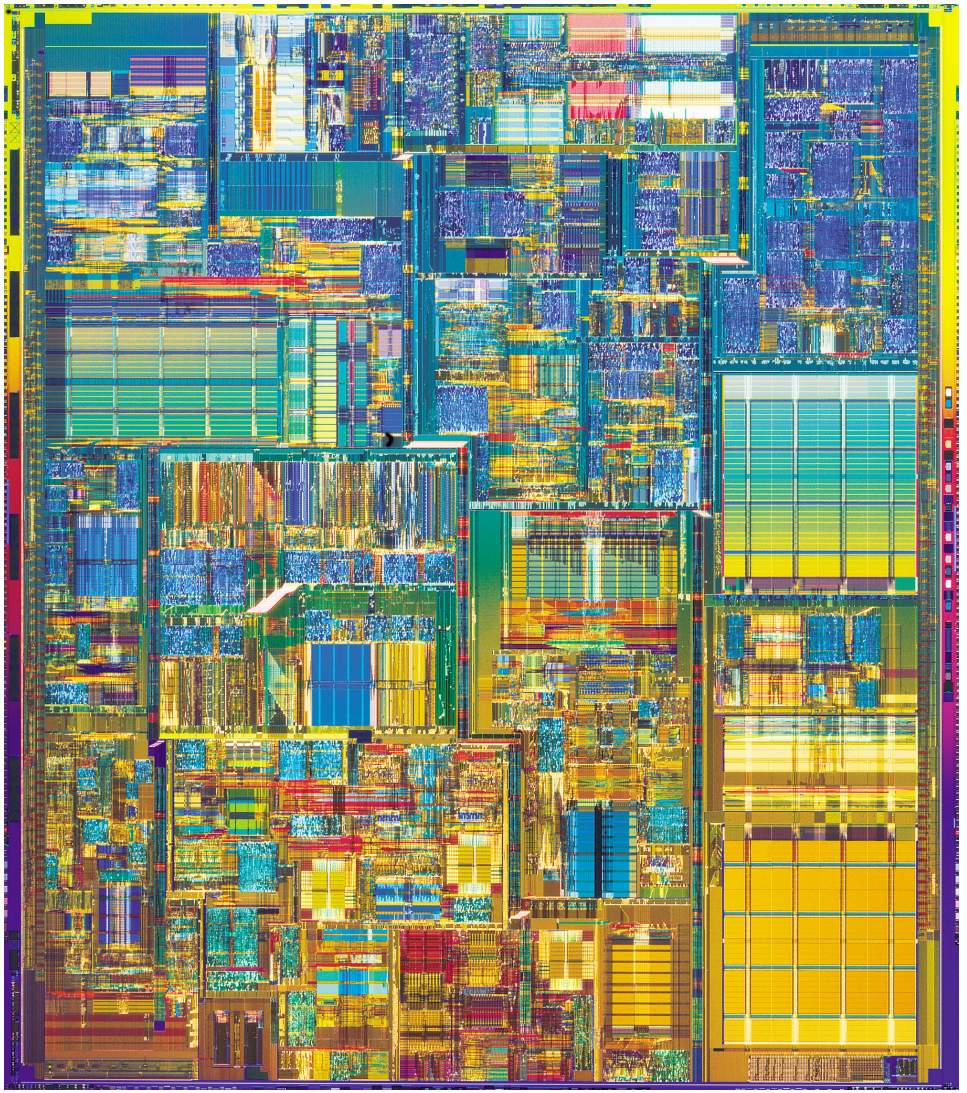
200 MHz, 200 MOPS, 3.3V, 35W

Intel' Microprocessor Gallery

1999 : Intel® Pentium® III Processor
9.5M Tr, 0.25um, 450MHz – 1GHz

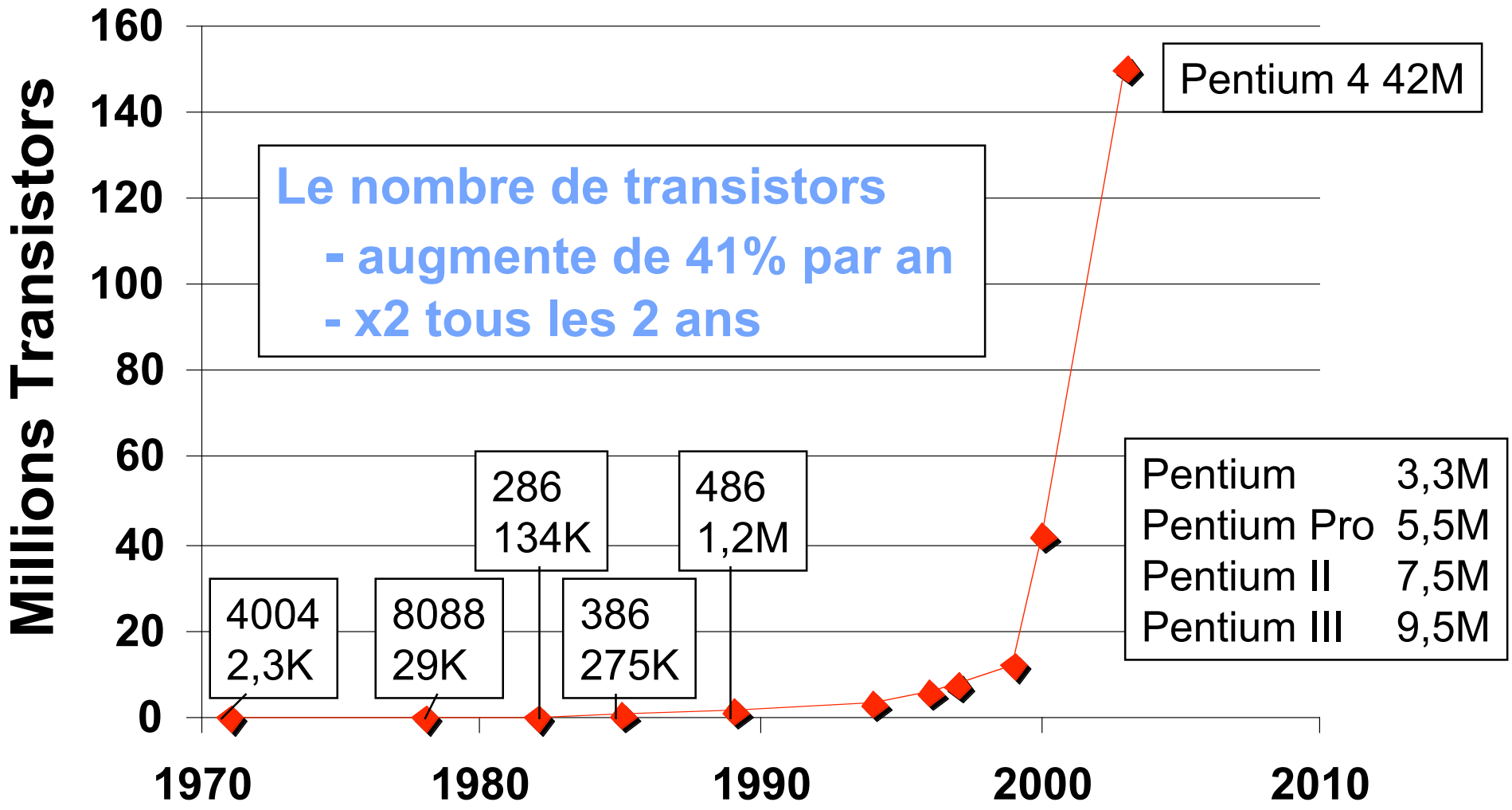


2000 : Intel® Pentium® 4 Processor
42M Tr, 0.18um, 1.5GHz – 3.6GHz



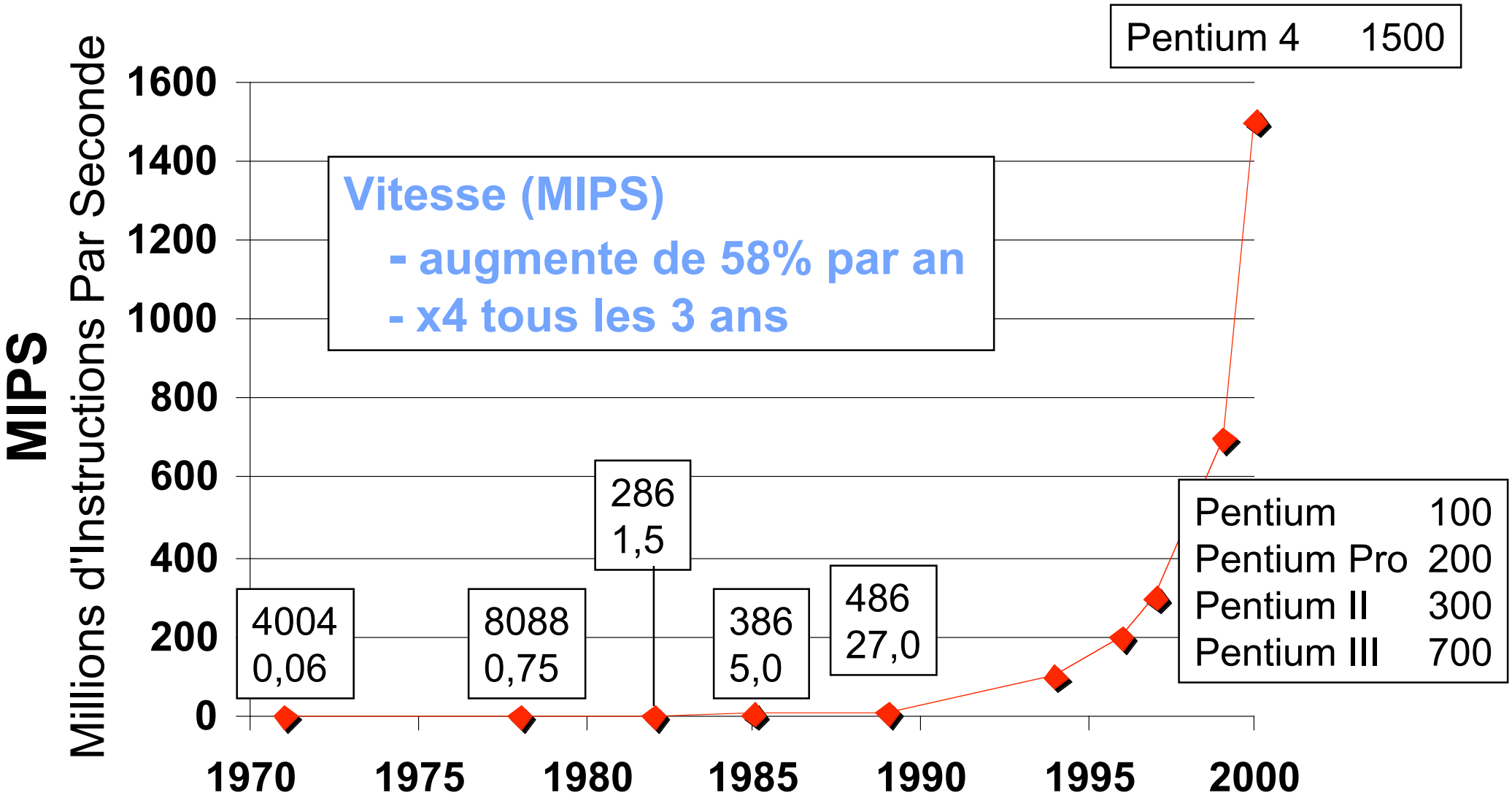
Nombre de transistors

- **Loi de G. Moore (INTEL corp.)**



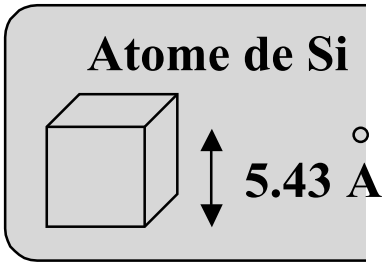
Évolution de la vitesse

- **Loi de G. Moore (INTEL corp.)**



Evolutions Technologiques

- **Microélectronique sur Silicium**
  - Evolution des techniques
    augmentation des vitesses
    réduction des dimensions
    augmentation de la surface des puces (15% / an)
    complexité des CI (Nb de Tr) (1,50 / an)
  - Silicium couvre 99% du marché
  - CMOS couvre 80% du marché des CI
  - Nombre de bits des DRAM : 4x tous les 3 ans



- **Aujourd'hui**
  - Mémoires DRAM 1 Gbits
  - Transistor MOS longueur de canal 0.18µ
  - CI comportant 20.10^6 transistors (µP)
  - Dispositifs d'une vitesse de 100 ps (NAND2 # 280 ps) => **VLSI**
  - Wafer de Si de 20 cm de diamètre
  - Taille de la puce (3-6 cm^2) limitée par ***la consommation / fiabilité***

- **En 2010**
  - Perspectives pour pour la première décennie du 21ème siècle
    CMOS, 600M transistors, wafer de 40 cm, puces de 6 à 14 cm^2
    Canal de 700 Ang. (0.07 µm), vitesse de 10 ps, traitement au GHz, 0.6 Volt => **ULSI**
    DRAM 64 GBits à 10ns, SRAM (cache) de 1 GBits à 1.5ns
    Usine de fabrication de plusieurs milliards de $US

Perspectives : RoadMap SIA

- **0.35 µm en 1995**
- **0.25 µm en 1997**
- **0.18 µm en 1999, 0.15 µm en 2001, 0.13 µm en 2003, 0.07 µm en 2009, 0.05 µm en 2012**

Année d'introduction	**1997**	**1999**	**2001**	**2003**	**2006**	**2009**	**2012**
Technologie (µm)	**0,25**	**0,18**	**0,15**	**0,13**	**0,1**	**0,07**	**0,05**
Nb de niveaux de métal	6	6-7	7	7	7-8	8-9	9
Tension d'alimentation (V)	1,8-2,5	1,5-1,8	1,2-1,5	1,2-1,5	0,9-1,2	0,6-0,9	0,5-0,6
Puissance sous radiateur (W)	70	90	110	130	160	170	175
Fréquence de fonctionnement (MHz)							
Horloge locale On-chip µP	750	1250	1500	2100	3500	6000	10000
Horloge locale On-chip ASIC	300	500	600	700	900	1200	1500

[RoadMap SIA]

Perspectives : RoadMap SIA

Année d'introduction	**1997**	**1999**	**2001**	**2003**	**2006**	**2009**	**2012**
Technologie (µm)	**0,25**	**0,18**	**0,15**	**0,13**	**0,1**	**0,07**	**0,05**
Taille du circuit (mm^2)							
DRAM	280	400	450	560	790	1120	1580
Microprocesseur	300	340	385	430	520	620	750
ASIC	480	800	850	900	1000	1100	1300
Mémoire							
Taille	256M	1G	*	4G	16G	64G	256G
Bits/cm^2	96M	270M	380M	770M	2,2B	6,1B	17B
Coût/MBit (Centimes)	20	10	5	2,5	0,9	0,32	0,11
Microprocesseur							
Logique Tr/cm^2	3,7M	6,2M	10M	18M	39M	84M	180M
Coût/MTr (Centimes)	500	290	166	97	42	18	8
ASIC							
Logique Tr/cm^2	8M	14M	16M	24M	40M	64M	100M
Coût/MTr (Centimes)	8,3	4,1	3,3	2,5	1,6	0,8	0,4

Marché des semi-conducteurs

- **Un marché mondial de 205 milliards de US dollars en 2000, pour une croissance de 37%**
- **En 2001, 249 B US$ (+22%)**
- **En 2004, 218.5 B US$ (+23,4%)**

- **DSPs : 6 B US$, +40% en 2000**
- **DRAM, 23.1 B US$ en 1999, 76 B US$ en 2002**

LES DIX PREMIERS FABRICANTS MONDIAUX DE SEMICONDUCTEURS EN 2004(*)

Rang 2004	Rang 2003	Sociétés	Ventes 2004 (M$)	Variation 2004/2003	Part de marché 2004
1	1	Intel	30509	+12,6%	13,7%
2	2	Samsung	15640	+48,9%	7,0%
3	4	Texas Instruments	9714	+31,1%	4,4%
4	7	Infineon	8903	+29,7%	4,0%
5	3	Renesas Technology	8849	+11,5%	4,0%
6	5	Toshiba	8849	+20,3%	4,0%
7	6	STMicroelectronics	8752	+21,9%	3,9%
8	8	Nec Electronics	6750	+15,5%	3,0%
9	10	Philips	5720	+26,8%	2,6%
10	9	Freescale	5697	+23,1%	2,6%

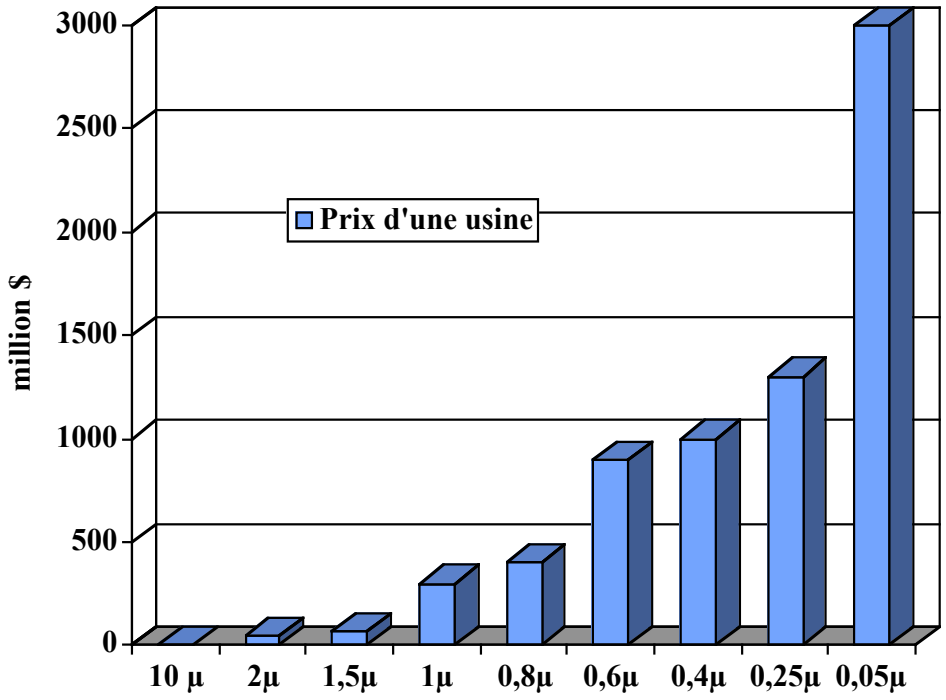
(*) Marché total 218,5 milliards de dollars (+23,4%). *Source: Gartner Dataquest décembre 2004*

Les mémoires et les radiotéléphones sont responsables des croissances supérieures à la moyenne enregistrées par Samsung, Texas Instruments et Infineon.

- **4 milliards de microprocesseurs**
  - 50 millions de microprocesseurs pour PC
  - Le marché est dans les µP *embarqués*

Semi-conducteurs et économie

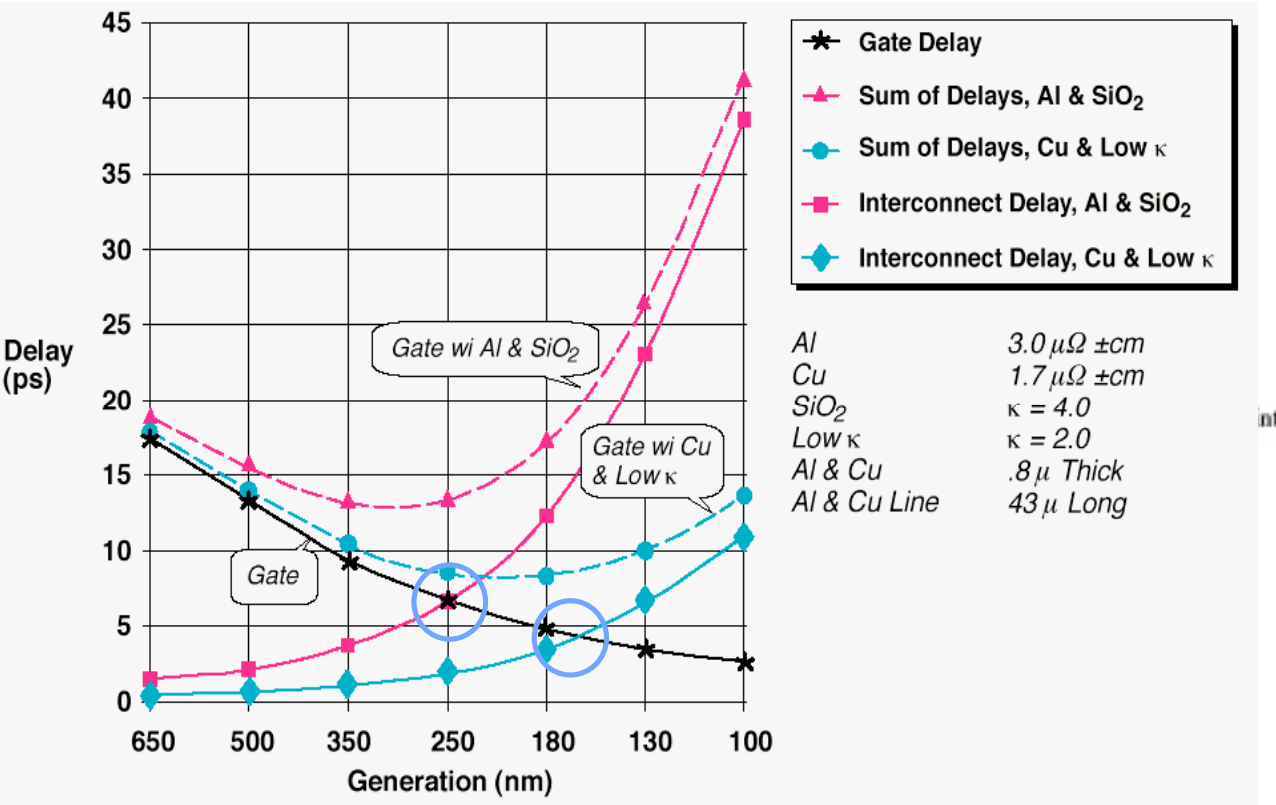
- Usine, lithographie : investissement important
- Loi de Rock : croissance des investissements (19% / an)
- Plus d'hommes pour concevoir les circuits
- Le coût par porte équivalente en baisse

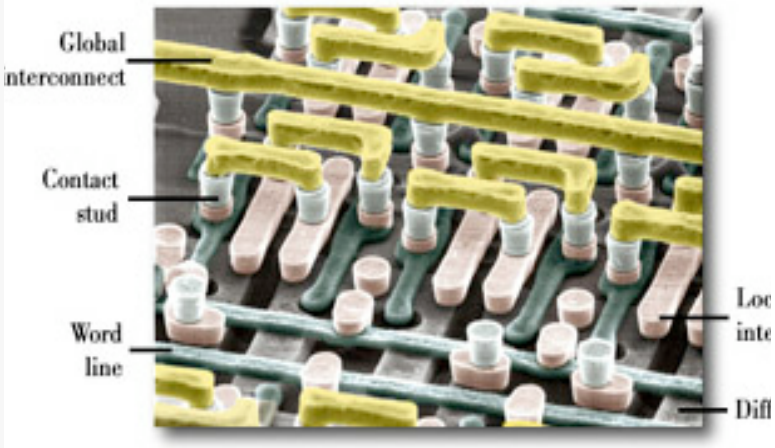


Prix des usines de fabrication en fonction de la technologie

Submicronique profond (DSM)

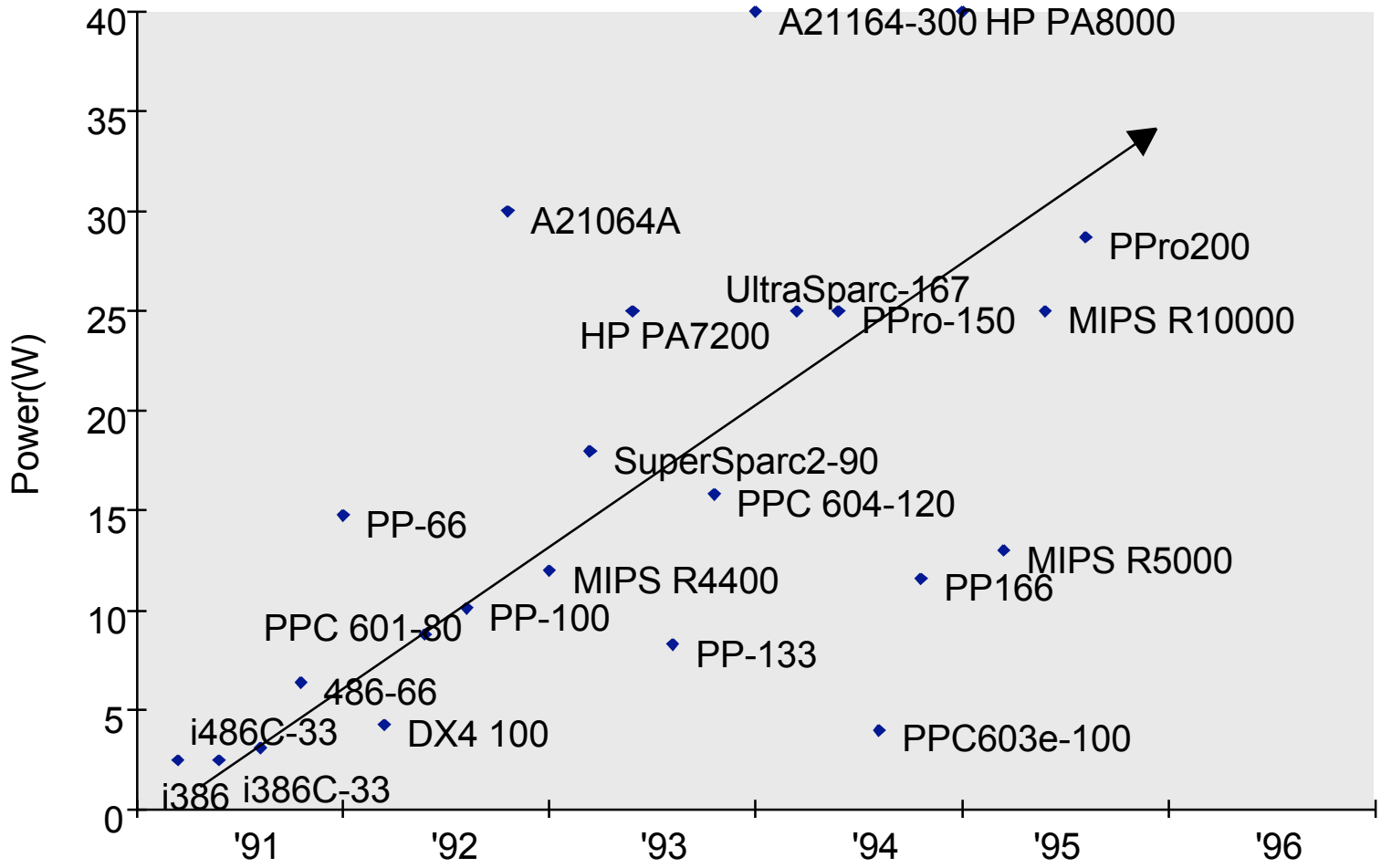
- **Changement fondamental dans le délai des composants**
  - Les interconnexions dominent le temps de fonctionnement et la consommation
    Jusqu'à 60% du chemin critique du aux interconnexions
  - Problème pour prédire de manière précise le routage
    Exemple: temps de propagation d'une NAND 2 entrées :
    - connecté à 2mm de métal : 280 ps
    - connecté à 0.5mm de métal : 119 ps
  - DSM implique un comportement non linéaire des portes
  - Cuivre, augmentation du nombre de niveaux de métal





Pourquoi s'occuper de la puissance ?

- **Evolution de la consommation des microprocesseurs**
  - x4 tous les 3 ans



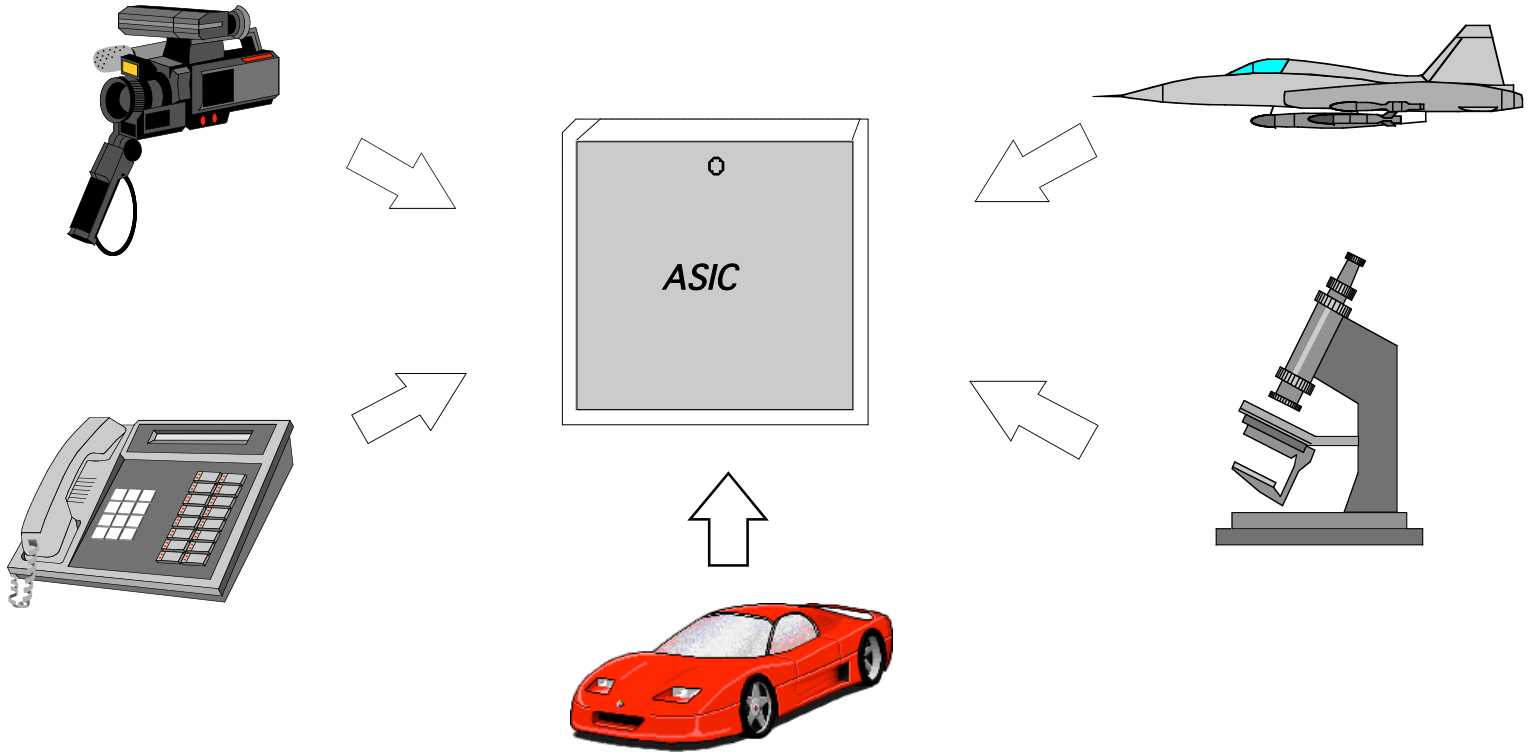
[Source: Microprocessor Report]

Qu'est ce qu'un circuit ASIC

ASIC : acronyme pour Applications Specific Integrated Circuit

"circuit à la demande" ou "circuit spécifique"

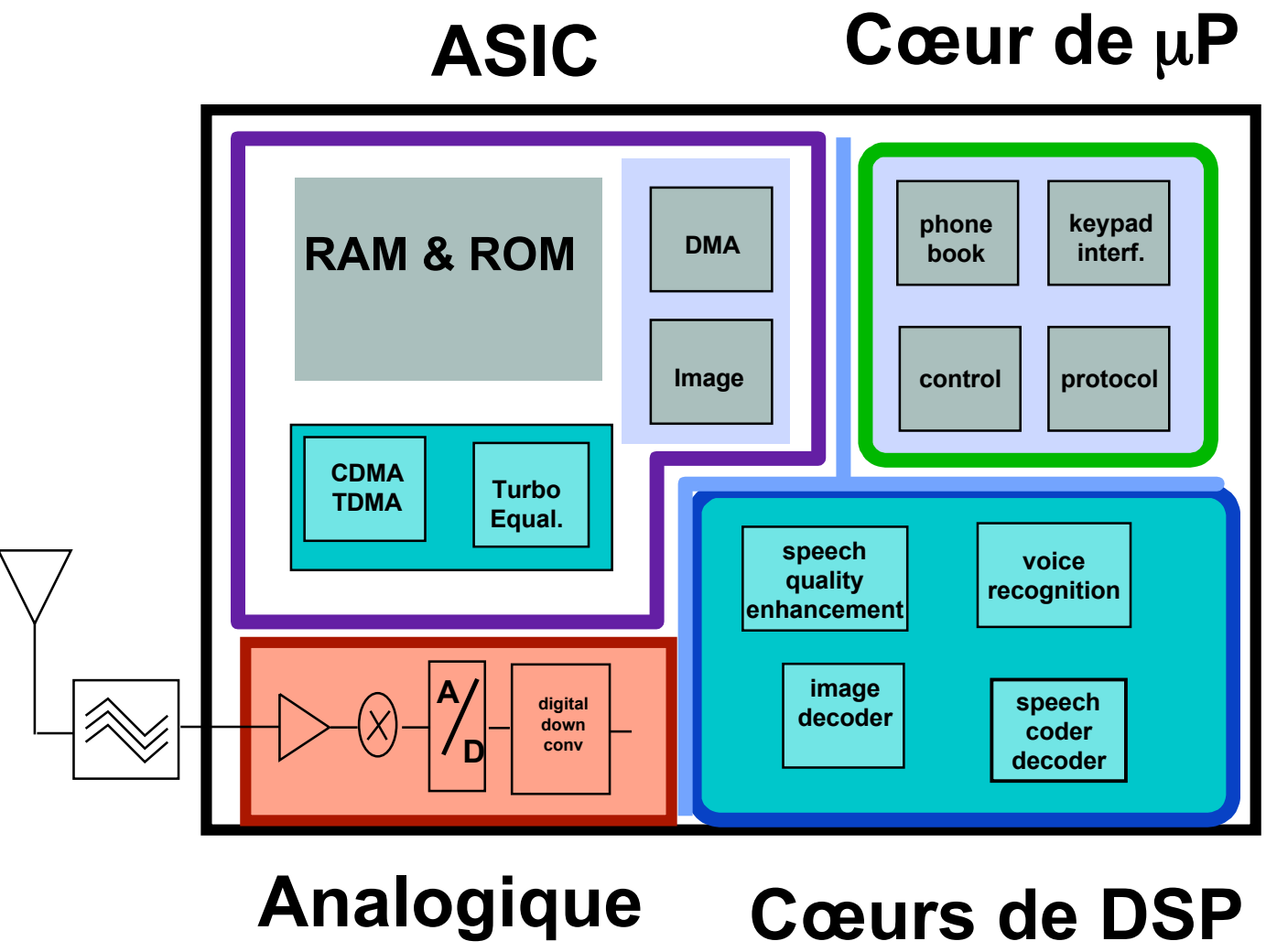
Circuit intégré ou FPGA conçu exclusivement pour le projet ou l'application qui l'utilise. Cela permet de le distinguer d'un composant standard pouvant être utilisé dans un large éventail d'application



Pourquoi un ASIC ?

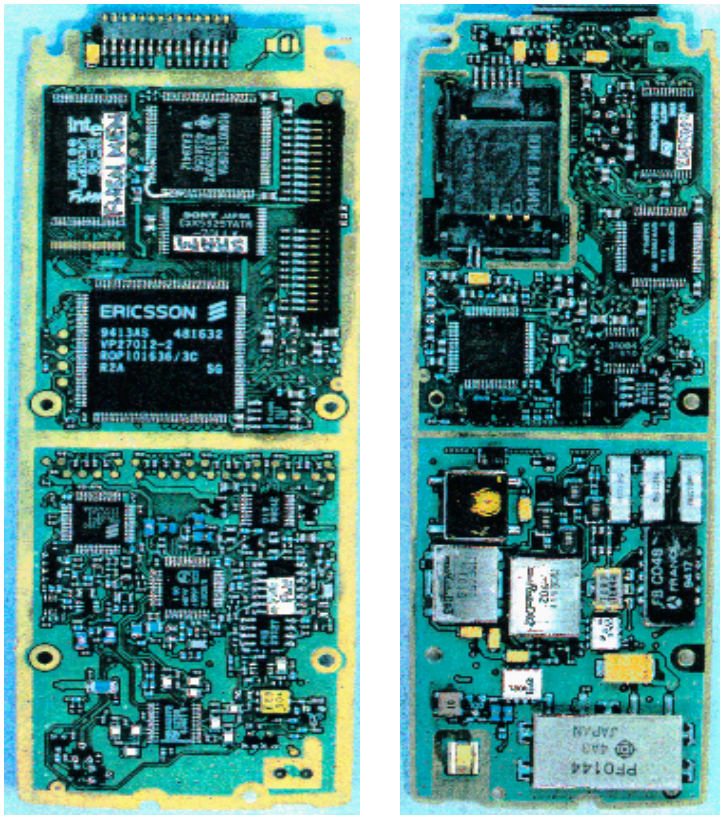
- **Diversité des applications justifiant le développement d'un ASIC**
  - Transformations de solutions existantes : opération d'intégration de solutions utilisant l'électronique analogique ou numérique. Augmentation des performances. Réduction du coût de production. Augmentation de la confidentialité.
  - Développement d'interfaces électroniques spécifiques : environnement physique (moteur, clavier, écran, ...), couplage entre tout type d'entité ou procédé.
  - Conception de fonctionnalités nouvelles : la technologie VLSI permet de considérer de nouvelles fonctionnalités spécifiques.
  - Développement d'architectures générales : µP, DSP, FPGA conduisent à accroître les possibilités de développement et la diversité des applications, à réduire le temps de développement ainsi que le coût par réutilisation d'un même type de composants.

Systèmes sur Silicium (SOC)

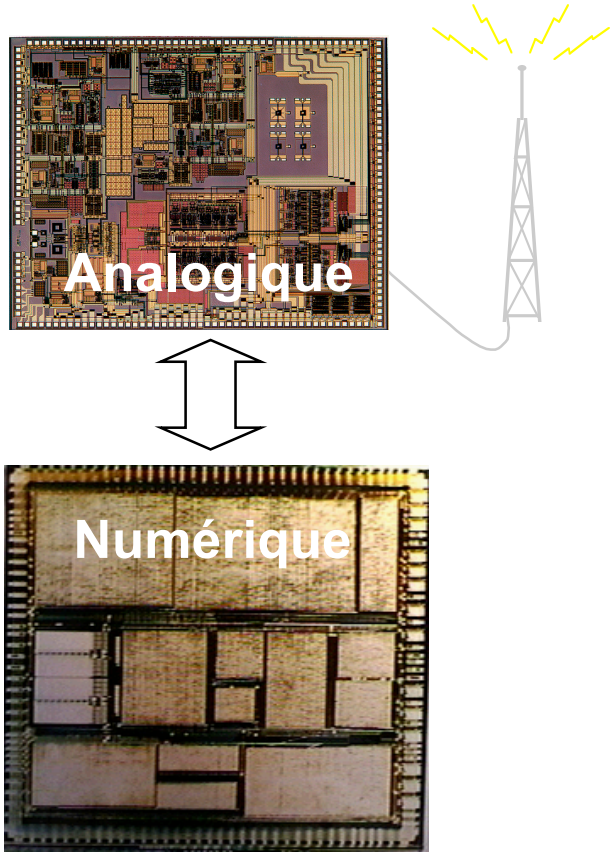


- Analogique
  - A/D
  - RF, modulation
- Cœur de µP/µC
  - protocole et contrôle
  - interface utilisateur
- Cœur de DSP
  - calculs lents
  - flexibilité
- ASIC
  - accélérateurs
- Mémoire
- Bus internes

Ex. 1 : terminal 2G

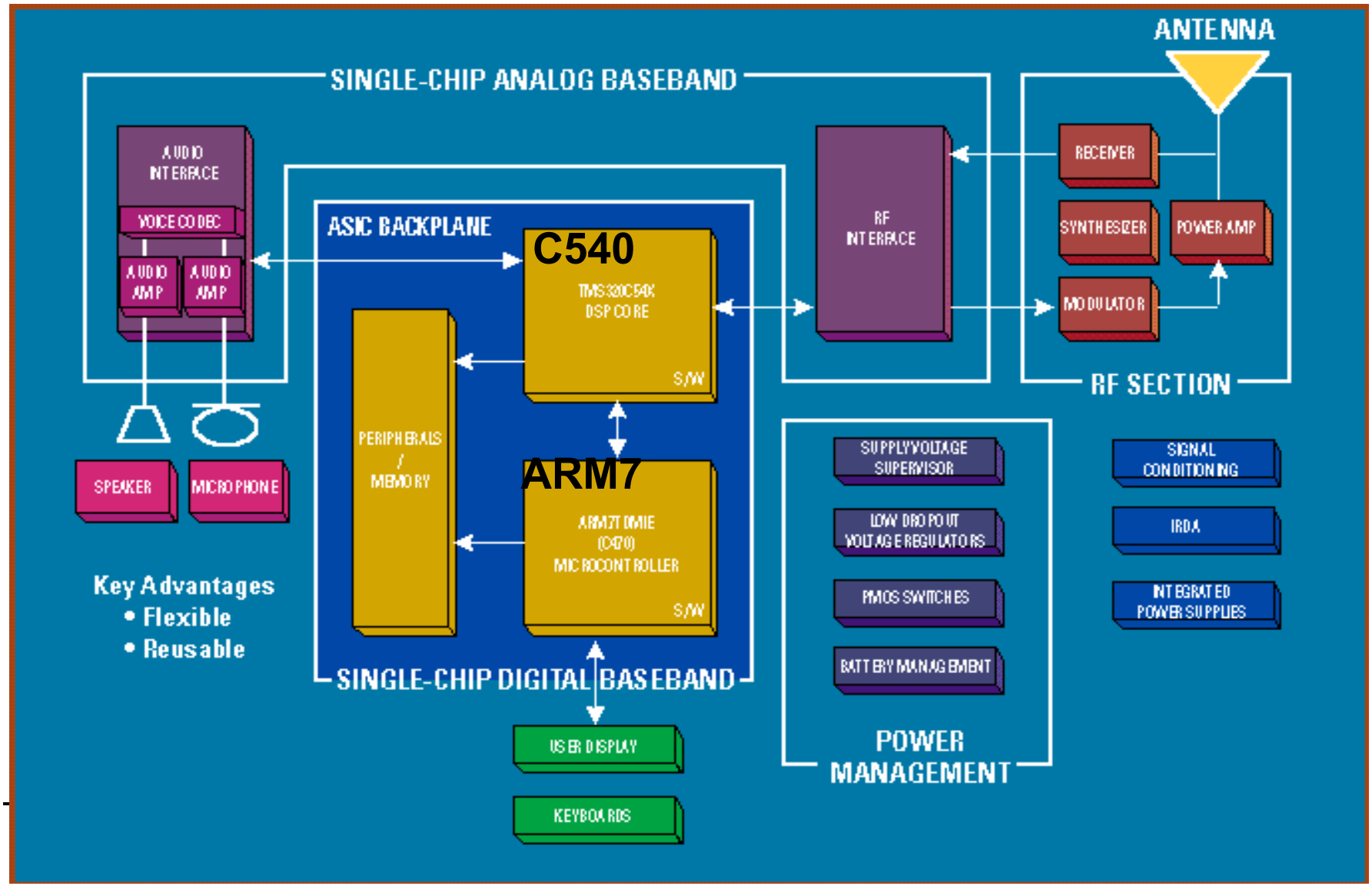


GSM standard



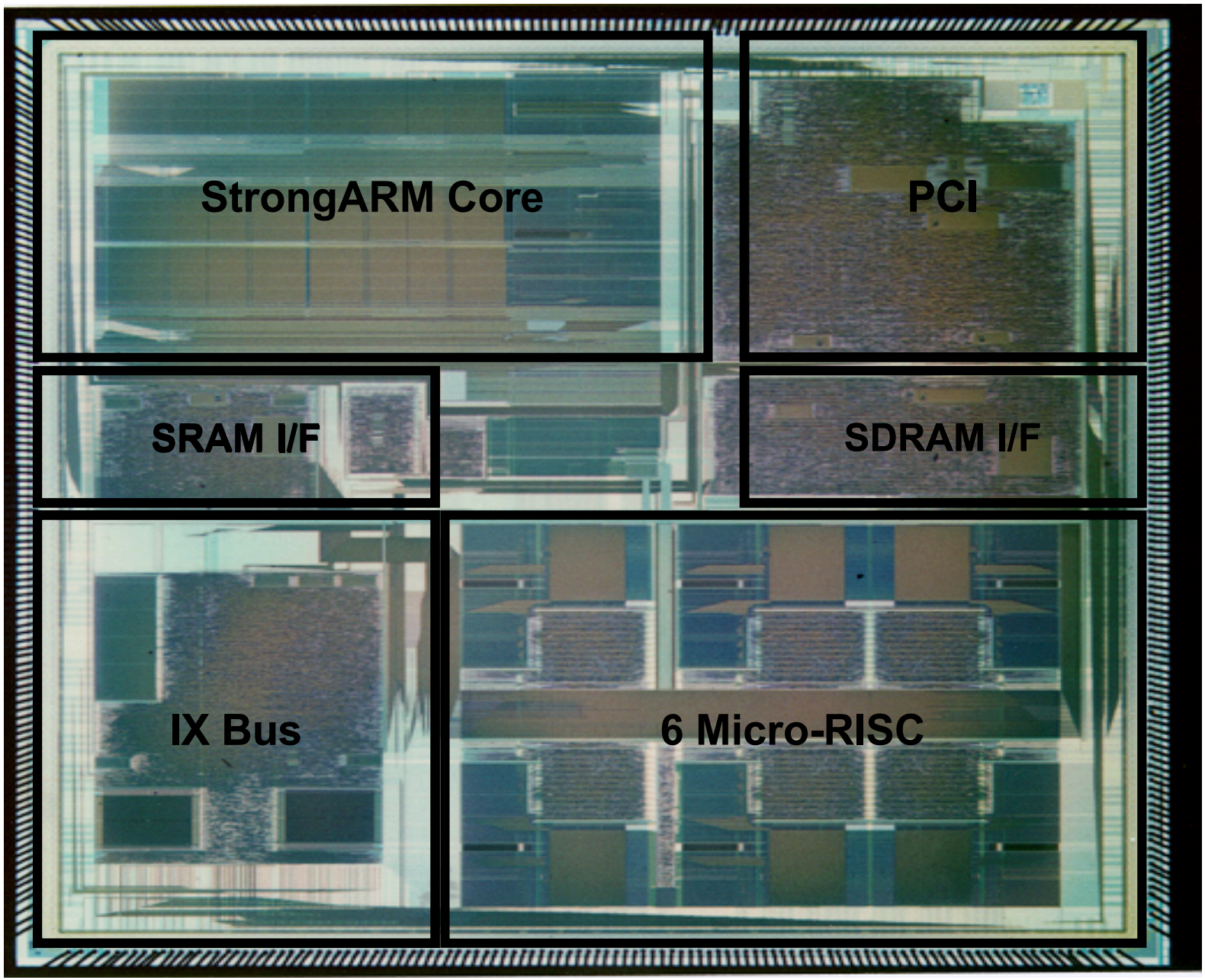
Future génération

Ex. 1 : terminal 2G



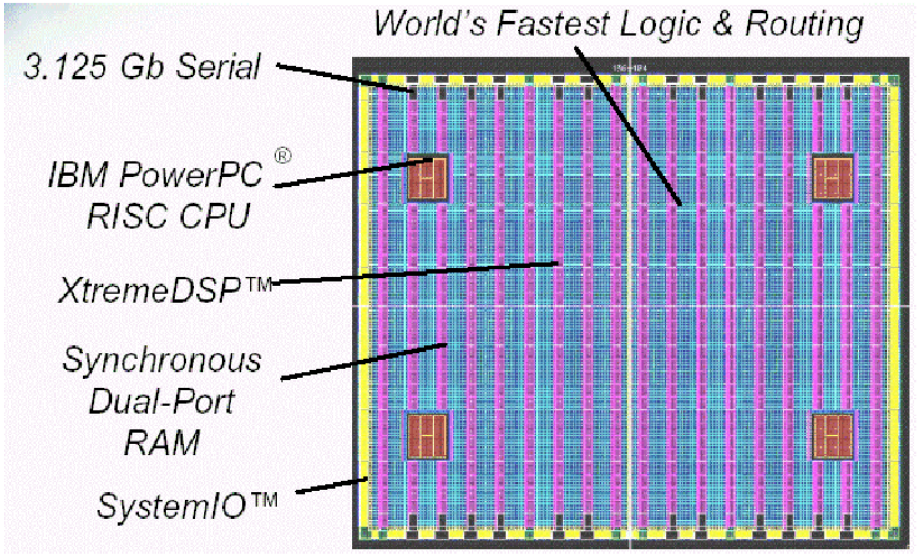
Ex. 2 : IXP1200 Intel NPU

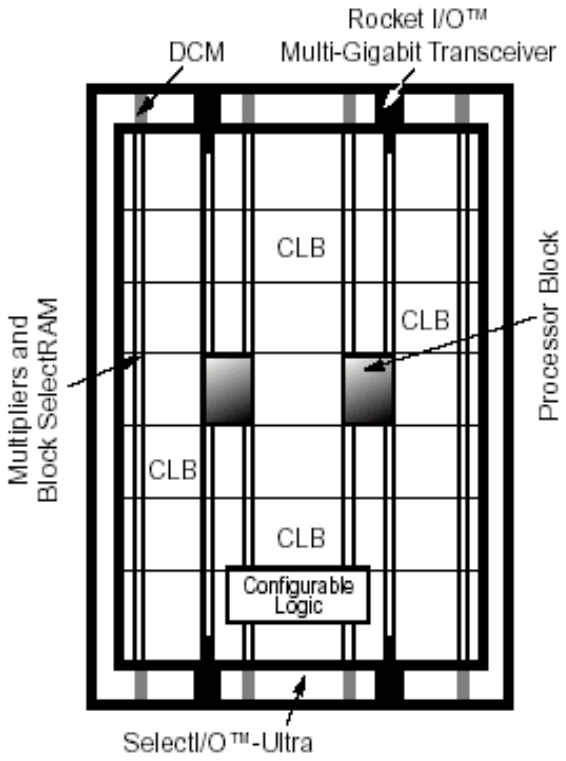
6.5M Transistors



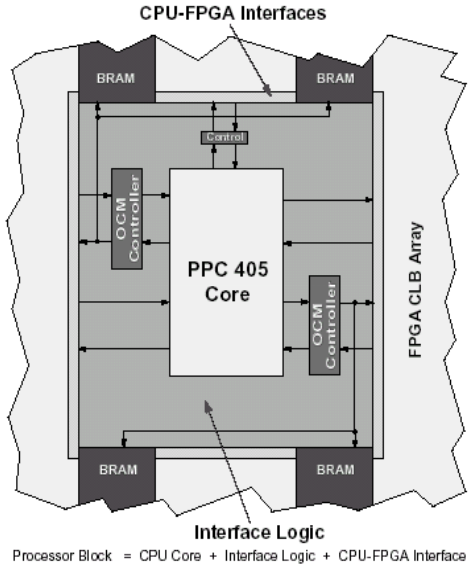
Ex. 3 : FPGA Xilinx VIRTEX II Pro

- Jusqu'à 4 blocs processeurs IBM PowerPC 405.
- Fréquence de travail jusqu'à 300MHz.
- Cœurs Rocket I/O Multi-Gigaset Tranceiver (transformation des données série-parallèle)





Architecture des Virtex II Pro



Bloc processeur des Virtex II Pro

Ex. 3 : FPGA Xilinx VIRTEX II Pro

Caractéristiques des Virtex II Pro

Feature/Product	XC2VP2	XC2VP4	XC2VP7	XC2VP20	XC2VP50
CLB Slices	1,408	3,008	4,928	9,280	22,592
Logic Cells	3,168	6,768	11,088	20,880	50,832
Block RAM (Kbits)	216	504	792	1,584	3,888
PowerPC Processors	0	1	1	2	4
Rocket I/O Multi-Gigabit Transceivers	4	4	8	8	16
18x18 Multipliers	12	28	44	88	216
Digital Clock Management Blocks	4	4	4	8	8
Max Available User I/O	204	348	396	564	852
Package	**User I/O**	**User I/O**	**User I/O**	**User I/O**	**User I/O**
FG256	140	140			
FG456	156	248	248		
FF672	204	348	396		
FF896			396	556	
FF1152				564	692
FF1517					852
BF957				564	584

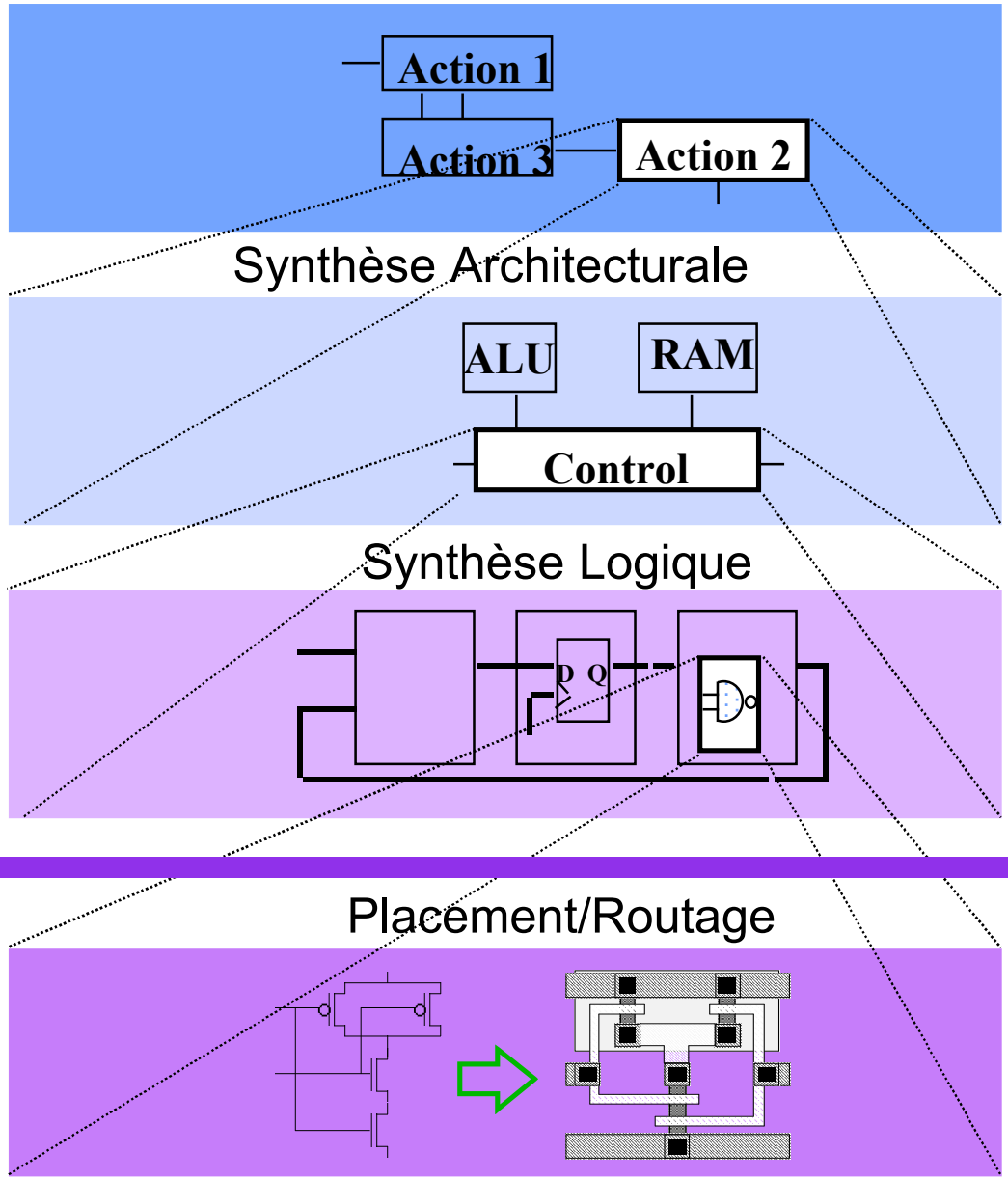
Avantages et inconvénients

- Avantages offerts par les circuits spécifiques
  - **Réduction de la taille** des produits finis.
  - **Consommation électrique diminuée** : les transistors internes ne commandent que les charges internes et non les pistes du circuit imprimé.
  - **Augmentation de la vitesse** : les signaux internes circulent plus rapidement dans une puce qu'entre plusieurs puces.
  - Confidentialité des projets.
  - Augmentation du nombre de fonctions proposées.
  - Coût d'assemblage réduit et fiabilité accrue du dispositif final.
  - Supériorité technologique d'une société : prolonger la durée de vie des produits.

- Inconvénients de la solution ASIC
  - L'utilisation des ASIC implique un niveau d'engagement élevé (financier et technique).
  - Perte de contrôle partielle au niveau des coûts et des délais en phase de fabrication.
  - Complexité de conception (systèmes sur puces)
  - Risques encourus en cas de non-fonctionnement ou de retard de fabrication
  - Coût du premier circuit

Description hiérarchique d'un circuit

Spécification fonctionnelle de l'ASIC



Plan Général

I Technologie des CIs
1 Classification des CI
2 Technologie MOS

II Méthodologie de conception
1 Démarche de conception
2 Spécification d'un ASIC
3 Outils de CAO

III Conception synchrone des C.I.
1 Règles de conception synchrone
2 Machine UT/UC câblée
3 Conception pour rapidité

IV Synthèse Logique à partir de VHDL
1 Conception des ASICs : Méthodes et Outils
2 Synthèse logique à partir de VHDL
3 Mécanismes de synthèse logique

V Projet de réalisation d'un ASIC

A vous de jouer!

I Technologie des CI

1. Fabrication des C.I.

- Processus de fabrication
- Phénomènes physiques et chimiques
- Exemple d'une diode et d ’un transistor

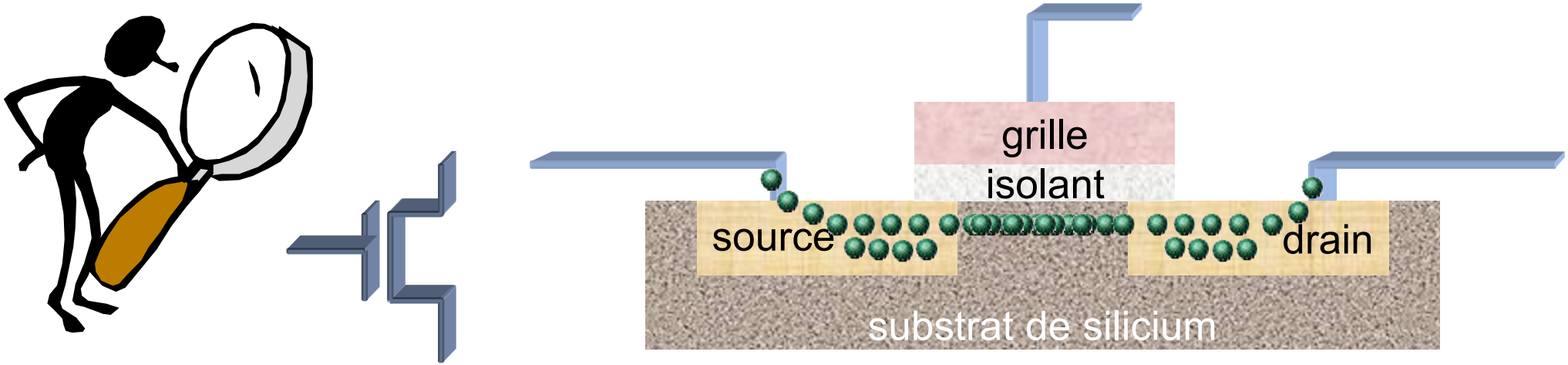
2. Classification des C.I.

3. Technologie MOS

- *Le transistor MOS : présentation générale*
- *Modèle et performances*
- *Technologies MOS (nMOS, pMOS, CMOS) et évolutions*

Le composant de base

- **Le transistor : un robinet à électrons**

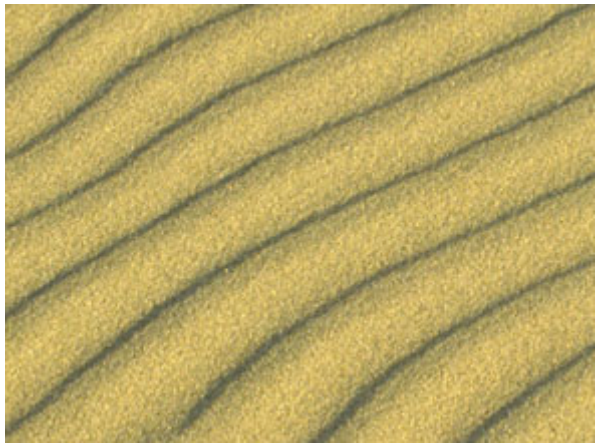


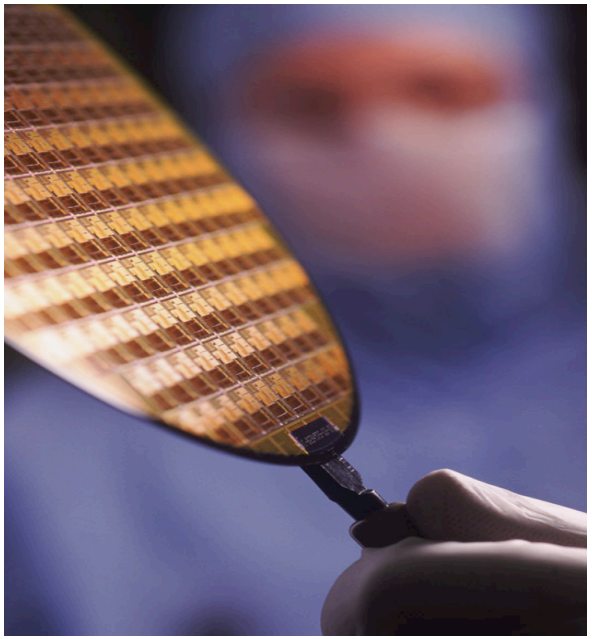
- **Permet de réaliser des fonctions logiques, arithmétiques, ...**

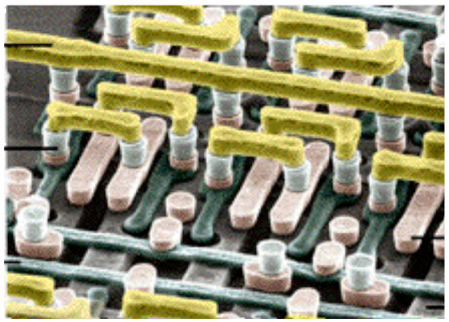
Comment fabrique t'on un CI ?

Du sable au silicium

Du silicium au circuit intégré

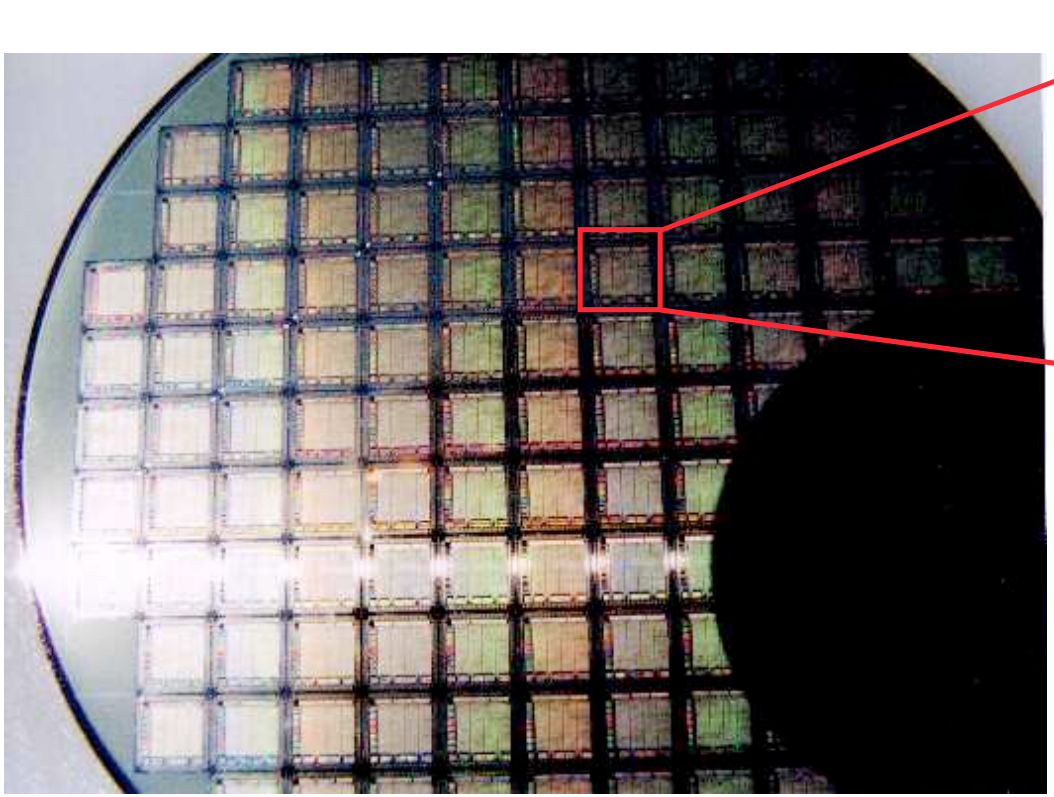


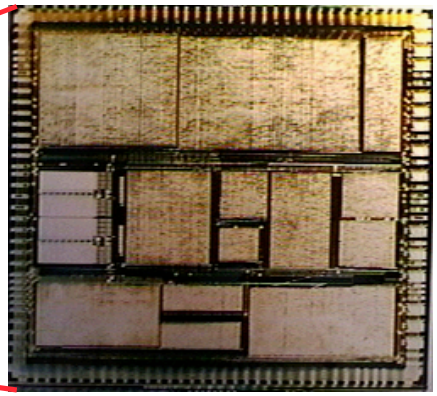




I.1 Fabrication des circuits intégrés

- **Disque de Silicium pur à 99,9999999%**



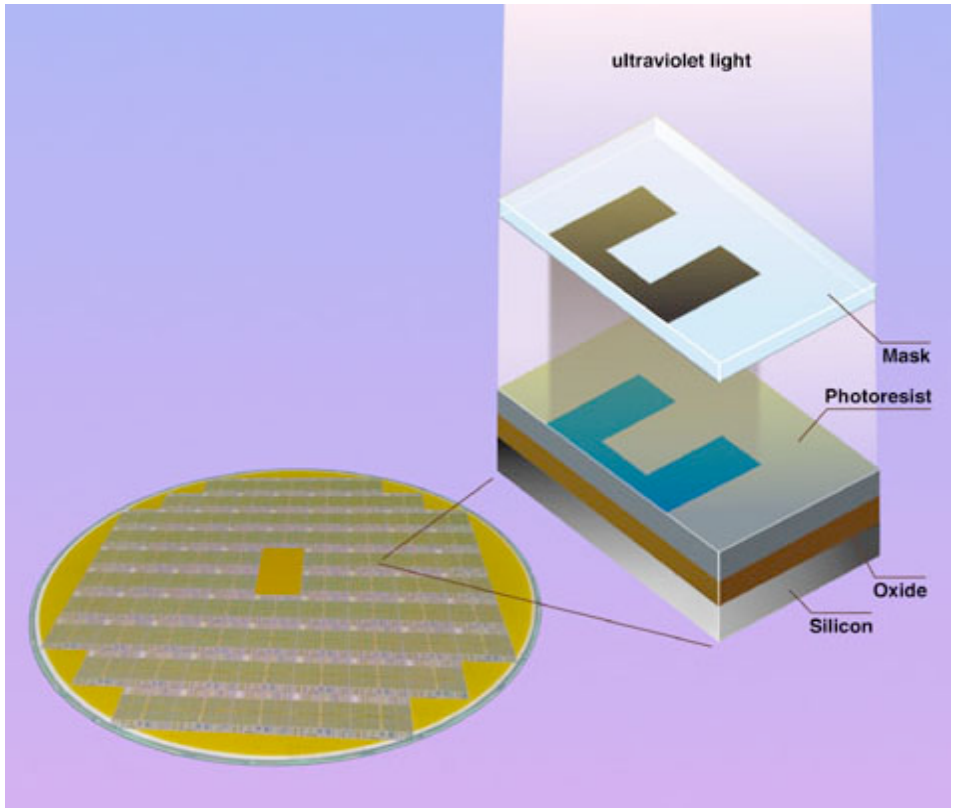


Wafer

Contrôle sous pointes des puces

Impression en 3D

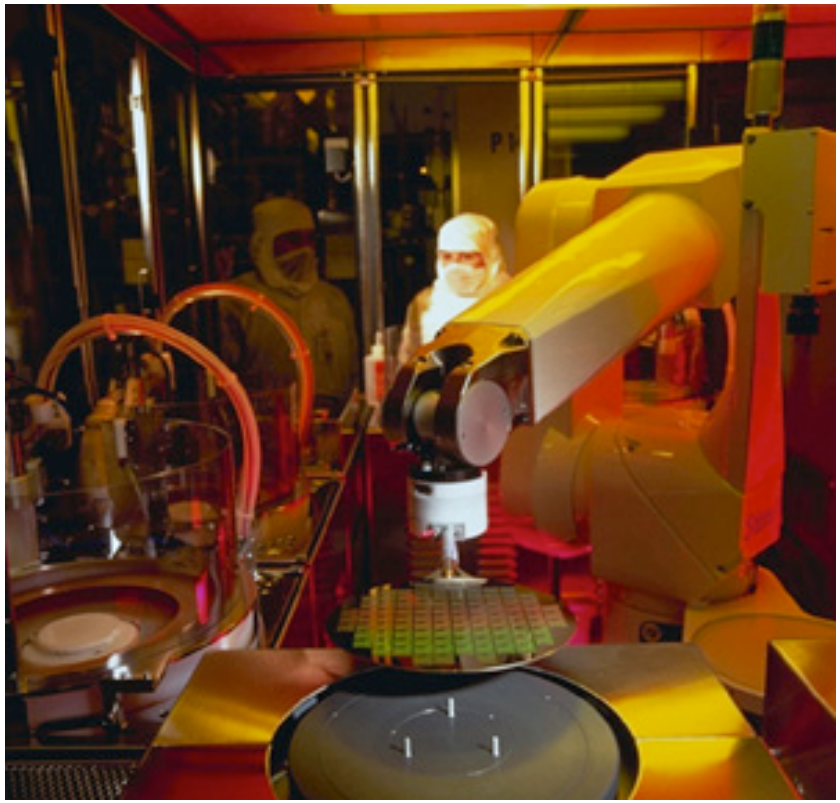
- **Comme l'impression d'un livre miniature**



Fabrication des circuits intégrés

Cybertour of a
chip manufacturing
clean room

click to start/pause



I Technologie des CI

1. Fabrication des C.I.

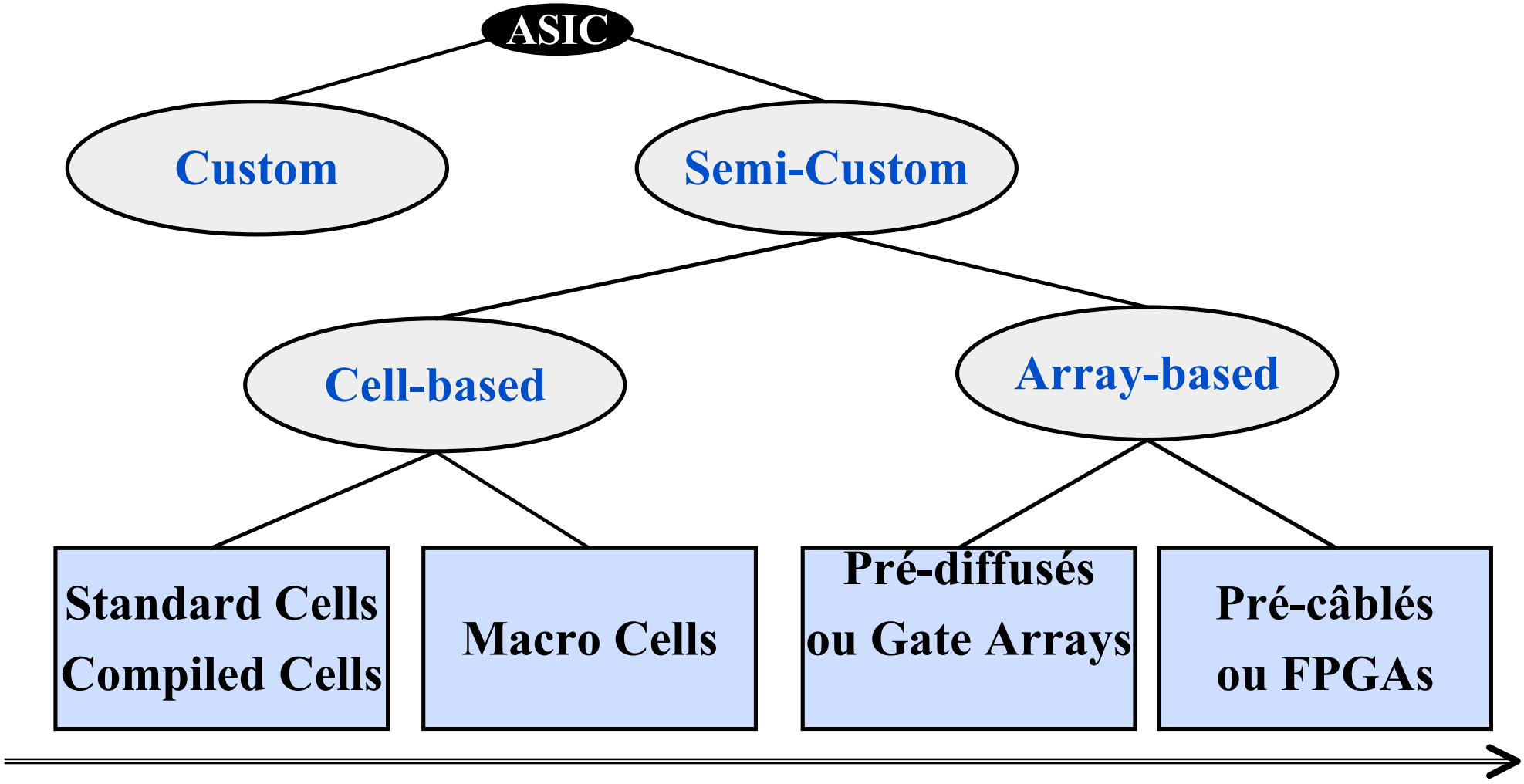
2. Classification des C.I.
   - Circuits Customs
   - Circuits Prédiffusés
   - Circuits Précaractérisés
   - Circuits FPGA
   - Critères de choix

3. Technologie bipolaire

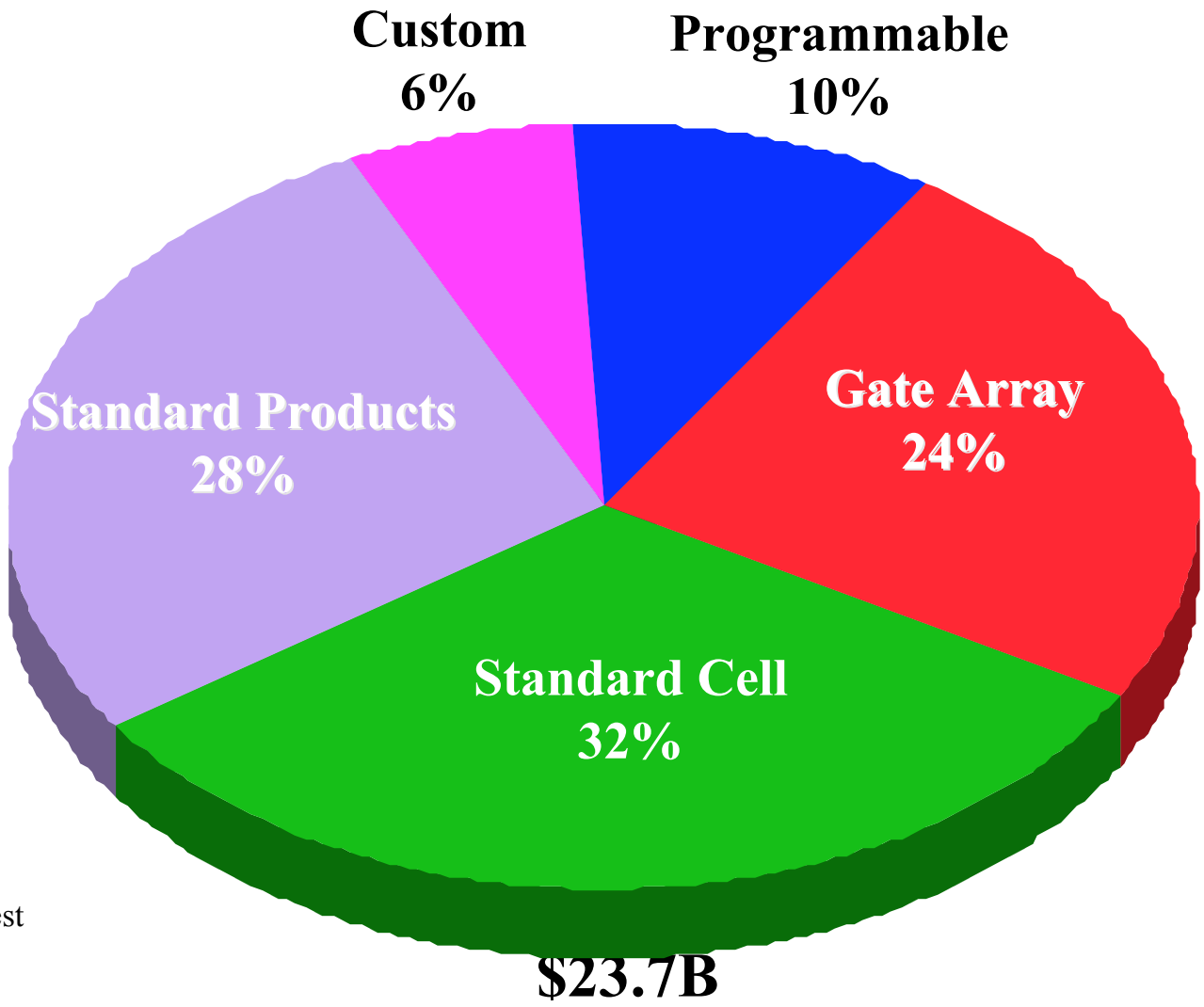
4. Technologie MOS

Classification des ASICs

- **ASIC : optimisations vs flexibilité, coût, temps de développement, prototypage, fondeurs, outils**

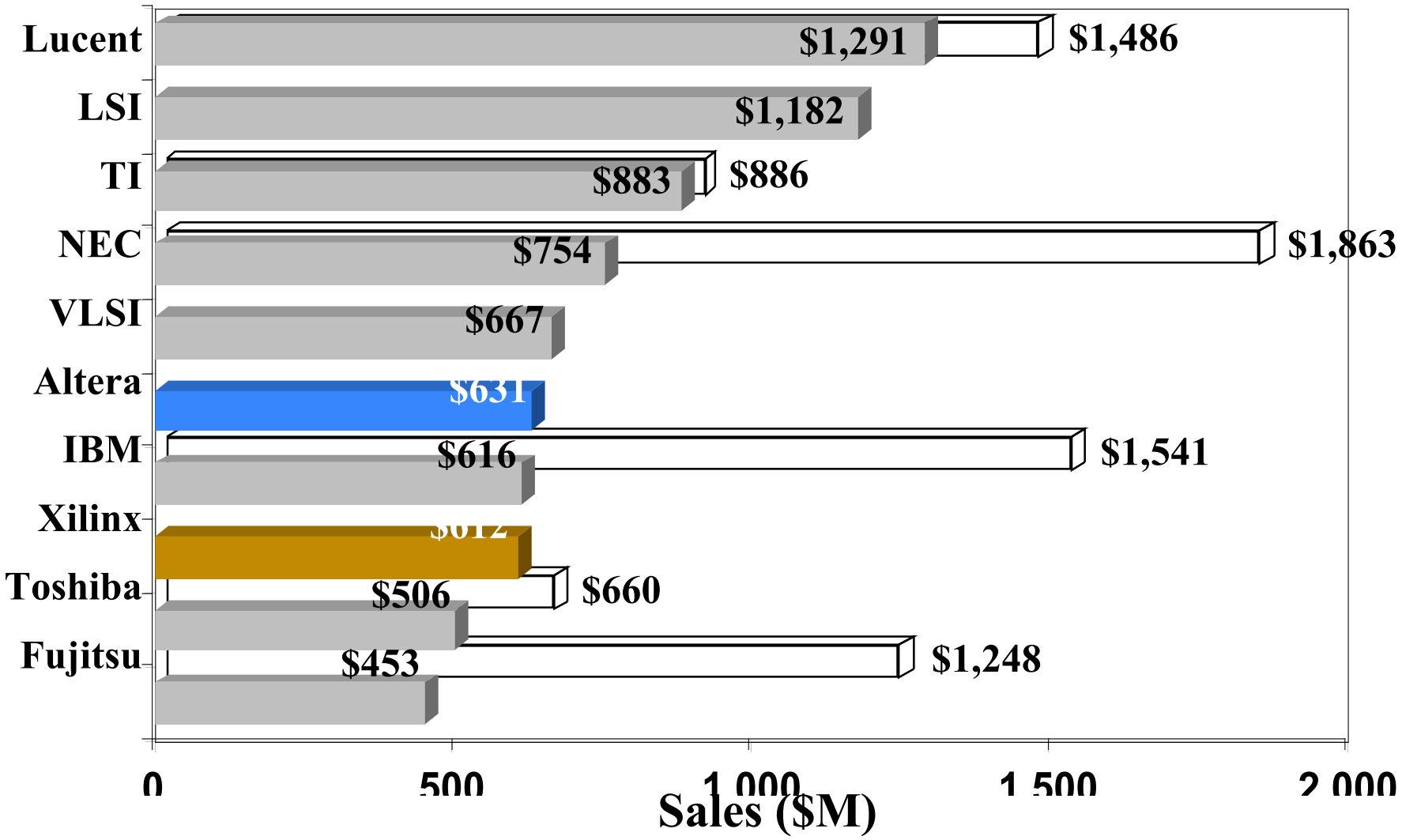


1997 CMOS Logic Market



Source: Dataquest

Top 10 ASIC/PLD Suppliers: 1997

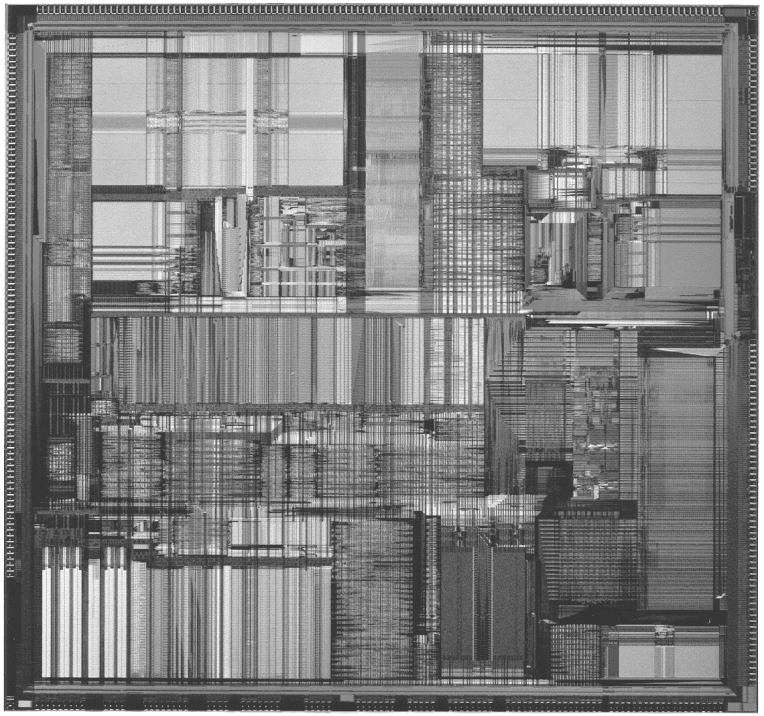


In Bold : Does Not Include Intracompany Sales
Source: Dataquest

Circuits custom

- **Transistors *taillés* sur mesure**
- **Avantages : optimisation densité, performance, consommation**
- **Inconvénients : prix très élevé, temps de développement (6/17 trans./jour), pas de *re-engineering***
- **UTILISATION : processeurs, domaine spatial, mémoires, modules performants.**

  (Exemple DEC Alpha : semi-custom excepté pour les unités arithmétiques)



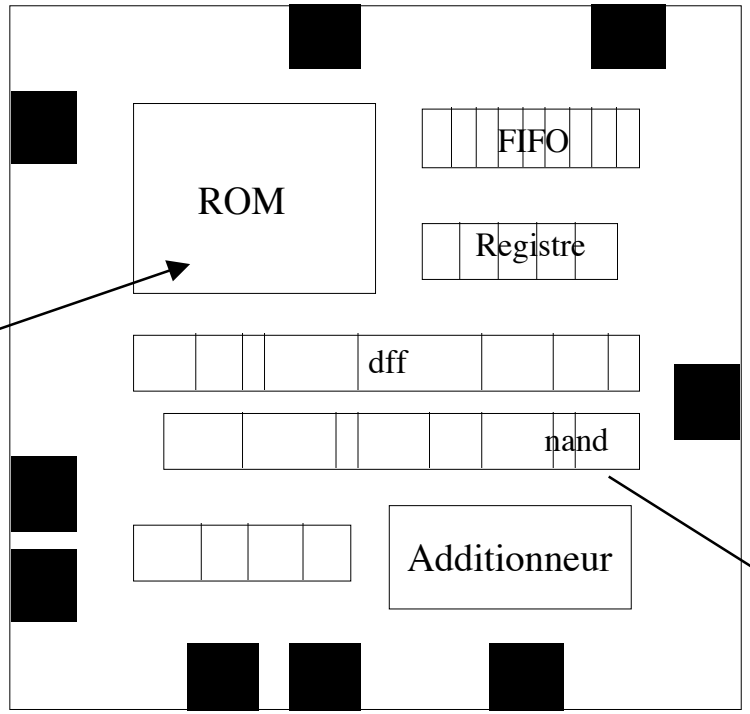
Pentium II (1998)

Data 32 bits, 7.5 M Transistors, 300 MHz, 3.3V, 0.35µ, 43W, 2 cm²

Circuits à base de Cellules

(Standard Cells, Précaractérisés, Cell based, ...)

Plots (pads)



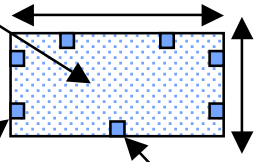
Blocs compilés
ROM, RAM
multiplieur
DataPath

- paramétrables
- génériques

description mono-bit
de la cellule puis
extension par compilation
ou juxtaposition sur N bits

Bibliothèque de cellules élémentaires définies au niveau physique
portes logiques
bascules, latches
buffers, mux, décodeurs
Full Adder/Substracter
Pad, compteurs, level shifter...
+ CAN, CNA, PLA, UART, µP

largeur variable
ex. NAND : 14.2µ
DFF : 73,6µ



hauteur fixe
ex. 43µ pour
techno. 1µ

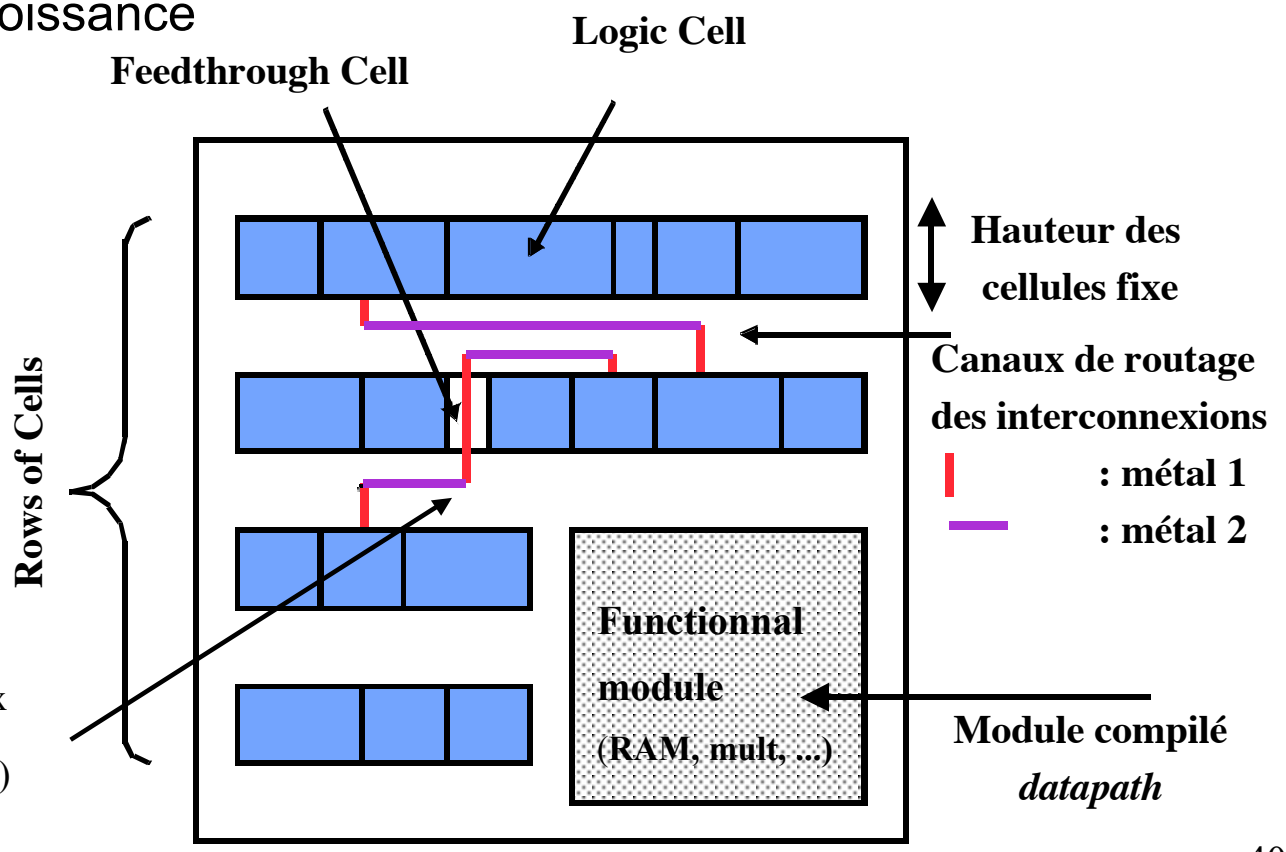
plots Vcc/Vdd pour mise en série

connecteurs d'E/S de la cellule

-> Compilation de silicium

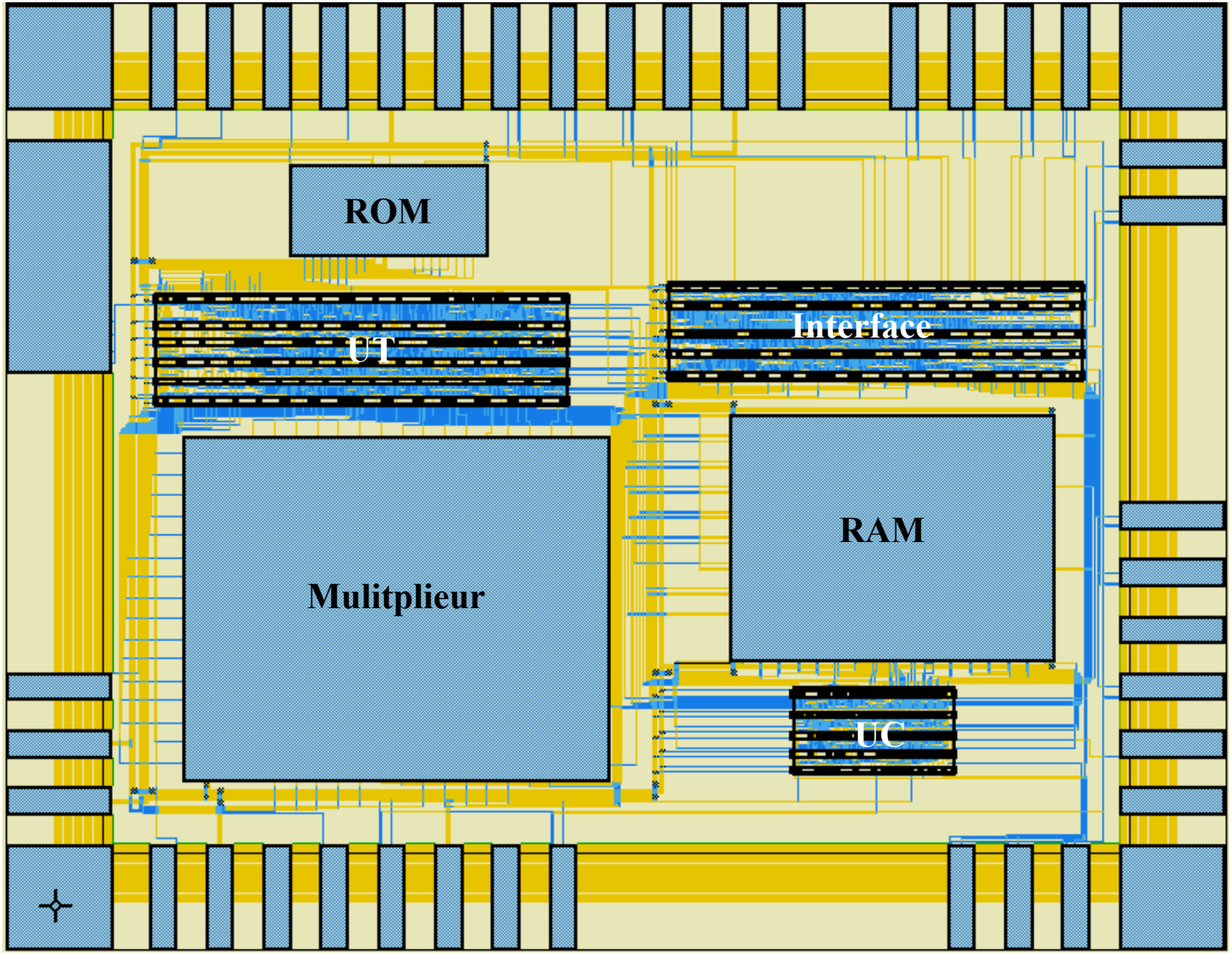
Circuits à base de Cellules (suite)

- **Avantages**
  - Conception hiérarchique de modules réutilisables
  - Réduction du temps de conception, coûts moindres
  - Perte de place pour les Connexions entre modules irréguliers.
  - Challenge : placement routage
  - Réutilisation des cellules
  - Librairies riches
  - Méthode en pleine croissance



Dans les techno. à 3 et plus niveaux de métal, les *feedthrough* (connexions au dessus des cellules) disparaissent

Standard Cells



Circuit FIR16

1995

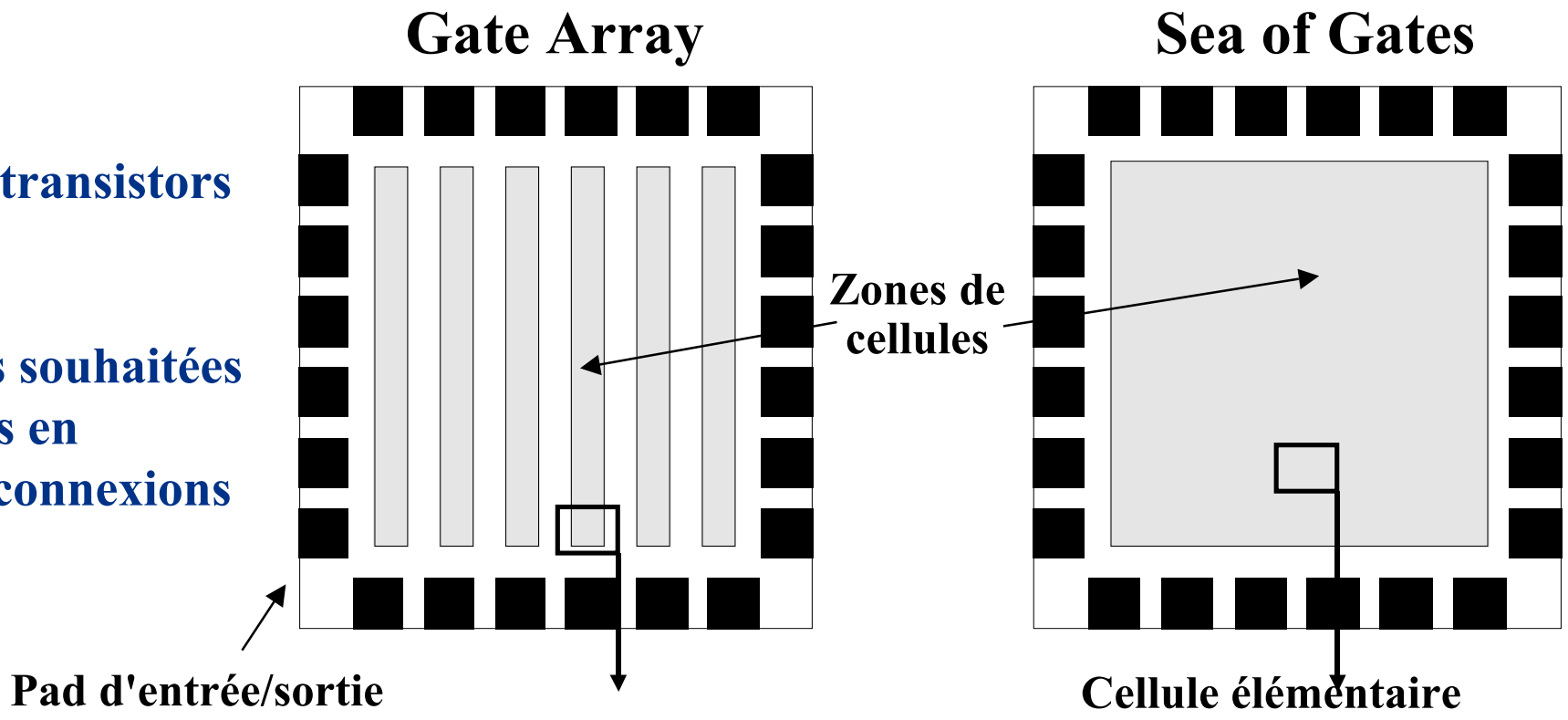
ES2 1µ

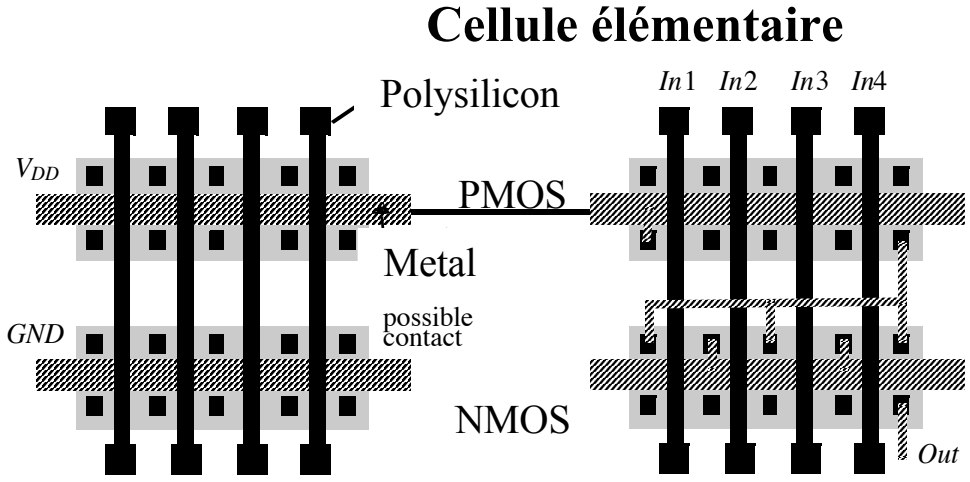
24 sqmm

Circuits Prédiffusés

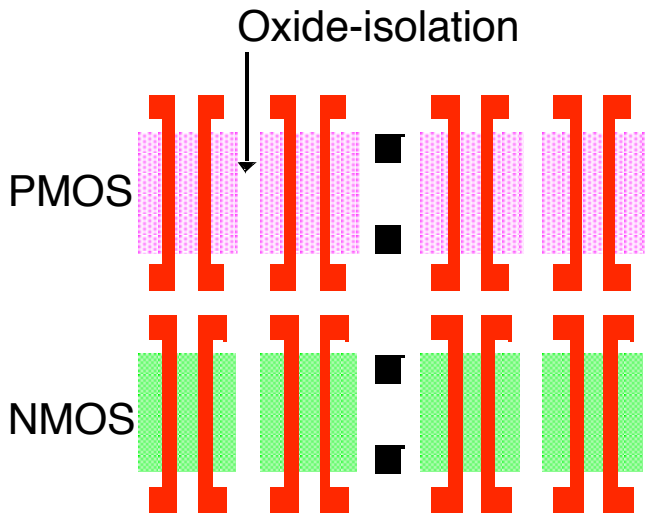
Tableaux de transistors pré-intégrés.

Les fonctions souhaitées sont obtenues en réalisant les connexions adéquates.





Non-Caractérisée → 4-input NOR



II Méthodologie de conception

1. Spécification d'un ASIC

1 Nature de la spécification d'un ASIC
2 Spécification détaillée
3 Plan méthodologique type de réalisation

2. Démarche de conception

1 Méthodologies ascendante et descendante
2 Cas de la conception de circuit

3. Outils de CAO

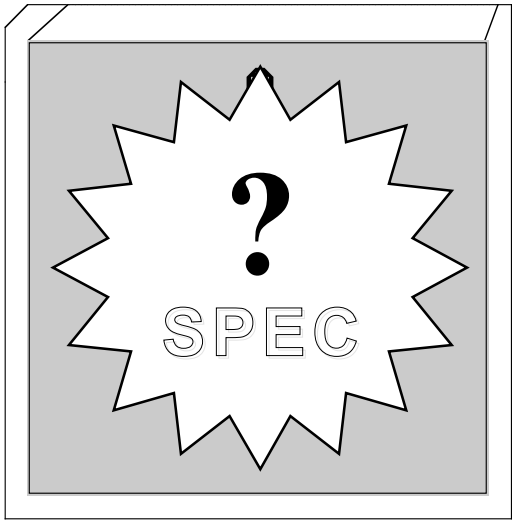
1 Evolution des outils
2 Simulation
3 Placement et routage
4 Synthèse logique

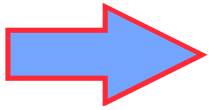
II.1 Spécification d'un ASIC

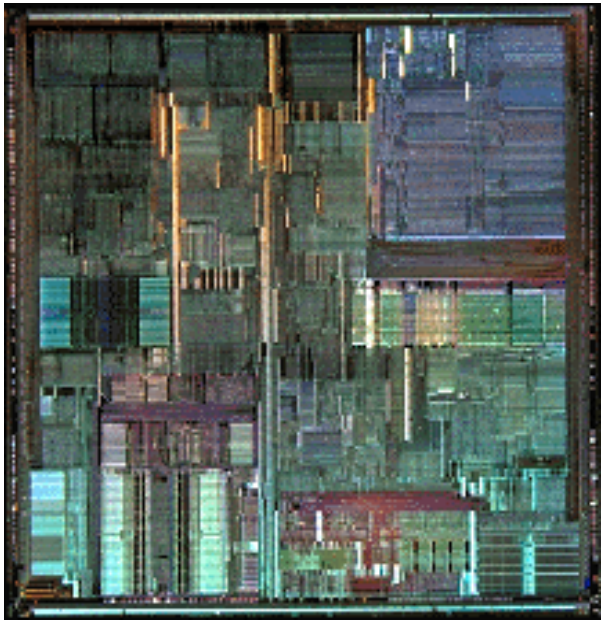
Le terme spécification regroupe toutes les informations qui caractérisent de l'extérieur le composant à réaliser. Les "spéc" sont indépendantes de l'utilisation qui est faite du circuit. Elles ont pour but de décrire ce que doit faire le composant (le QUOI) et pas du tout comment il le fait (le COMMENT).

- **Fonctionnelle, Opératoire, Technologique**
- **Sous forme papier ou informatique**

-> Vérification des spécifications







II.1.1 Nature de la spécification d'un circuit

- **Spécifications fonctionnelles**
  Description des fonctions que doit assurer le circuit (ou le bloc)
  - » Equation logique, Table de vérité, Chronogramme, Table de transition (équation d'état)
  - » Diagramme état/transition, Grafcet (divergence/convergence ET/OU)
  - » Statechart (extension du diagramme d'état au parallélisme)
  - » Réseau de Petri (comportement temporel)
  - » Diagramme des activités (diagramme flots de données et flots de contrôle : Statamate de iLogix)
  - » Modèle mathématique (TNS)
  - » Description algorithmique

- **Spécifications opératoires**
  Manière dont une fonction doit opérer, conditions et domaines de fonctionnement
  - » Renseignements sur les grandeurs ou données utilisées dans les spécifications fonctionnelles
  - » (type, domaine de définition, précision)
  - » Informations pour guider les concepteurs dans le choix des solutions à mettre en œuvre
  - » (expériences précédentes dans l'entreprise ou ailleurs)
  - » Test à opérer sur le circuit

- **Spécifications technologiques**
  Renseignements en rapport avec la réalisation matérielle
  - » définition électrique des E/S (DC, type de driver,...)
  - » performances / contraintes
  - » spéc de réalisation (taille, coût, technologie, type de boîtier,...)
  - » contraintes de l'environnement
  - » qualité de test (AQL)

II.1.2 Spécification détaillée d'un ASIC

Une spécification détaillée (ASIC Detailed Design Specification) d'ASIC est un document écrit qui regroupe

- La spécification de la définition
- La notice descriptive de fonctionnement

Elle doit fournir toutes les informations utiles aux concepteurs des cartes utilisatrices, ainsi qu'aux concepteurs de logiciels.

[Alcatel]

Plan type de spécification détaillée

1 Introduction

- Rappelle l'utilisation du circuit sur la carte
- Rôle principal et fonctions

2 Environnement du circuit

- Présentation générale de l'environnement du circuit sur la carte

3 Organisation générale du circuit

- Interfaces externes
- Présentation générale des fonctions
  - Synoptique générale du circuit faisant apparaître les blocs fonctionnels
  - Description des liaisons inter-blocs

4 Fonctions réalisées

- Description détaillée des fonctions réalisées par le circuit :
  - fonctions d'entrées-sorties
  - fonction d'initialisation
  - fonctions pour le test du composant
  - fonctions pour le test en sortie de fabrication des cartes utilisant le circuit
  - etc.

Plan type ••• (suite)

5 Caractéristiques matérielles

- Interface physique
  - » Packaging
  - » Brochage. Description des signaux d'E/S et des alimentations
  - » Technologie des E/S
- Chronogramme et Timings
  - » Chronogrammes théoriques associés aux diverses fonctions
  - » Timings à respecter
- Caractéristiques électriques
  - » Limites électriques, conditions transitoires et opérationnelles
  - » Caractéristiques statiques et dynamiques des E/S (alimentations, tension, courant, capacités de charge, *Slew Rate Control*,...)
  - » Découplage d'alimentation à prévoir sur la carte

6 Interface Logiciel

- Synthèse des informations relatives à l'interface matériel/logiciel
- Description des registres et compteurs accessibles et de leur mode d'accès
- L'utilisateur logiciel doit pouvoir se limiter à ce paragraphe pour le développement des logiciels

II.1.3 Spécification de réalisation

But de la spécification de réalisation est :

- Dans la phase de conception des blocs, de préciser aux concepteurs les directives de réalisation
- De décrire l'architecture interne et de fournir une description détaillée de chacun des blocs constituant l'ASIC

Ce document est nécessaire pour les ASICs dont la complexité nécessite plusieurs concepteurs. En fin de conception, la spécification fait l'objet d'une mise à jour.

Plan type

- **Présentation générale de la découpe en blocs**
  - Diagramme général du circuit découpé
  - Interfaces inter-blocs ou avec l'extérieur
    - » Les bus et liaisons de contrôle principaux
    - » La distribution d'horloge interne
    - » Les cycles d'échanges entre blocs
  - Pour chaque bloc :
    - » Une description succincte de la fonction réalisée
    - » Une estimation de la complexité
    - » Les mémoires, macrocellules nécessaires (capacité, temps de cycle, simple/multi port...)
    - » La fréquence moyenne de fonctionnement et le taux d'activité

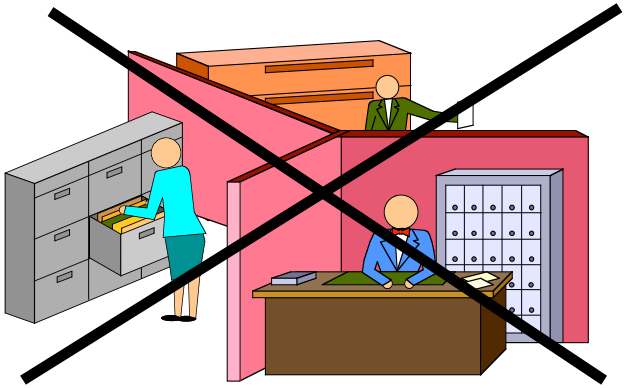
Plan type ••• (suite)

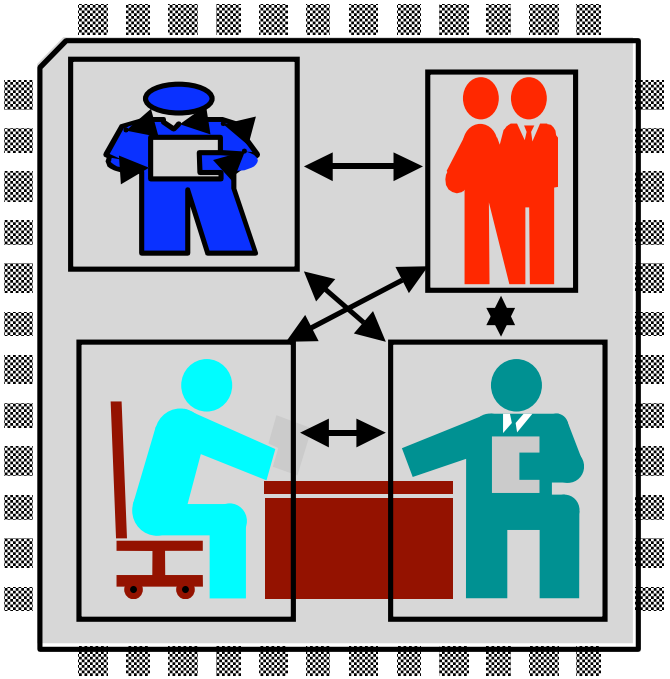
- **Les Contraintes de réalisation**
  - Boîtier : type de montage sur carte (soudé, support)
  - Nombre et type d'E/S
  - Fréquence aux accès
  - Contraintes principales de timing
  - Bilan des mémoires et/ou macrocellules nécessaires
  - Technologie (CMOS, bipolaire, ECL,...)
  - Type de "Matrice" et référence fondeur du circuit
- **Description détaillée des interfaces inter-blocs**
  - Fonctions et chronogrammes de chaque bus ou liaison de contrôle interne
- **Description détaillée de chaque bloc**
  - Fonctions réalisées
  - Accès E/S
  - Structure interne du bloc
  - Complexité du bloc (nombre de portes)

Un travail d'équipe avant tout

- **Complexité croissante des ASIC et délais de + en + courts -> équipe de plusieurs personnes pour un même CI**

- **Découpe précise du travail**
- **Distribution des tâches**
  - **Front End, Back End, Verification**
- **Méthode de travail**
- **Gestion informatique**





- *Une personne : Une fonction*
- *Chaque fonction communique avec d'autres*
- *Réunions fréquentes pour préciser les interfaces et les échanges de données*

Découpe du circuit en blocs fonctionnels

- **Cahier des charges :**
  - -> Recensement des fonctions à implanter dans l'ASIC
  - -> Regroupement en blocs fonctionnels
  - -> Définir l'ordre des traitements à appliquer aux données
  - -> Synoptique global du circuit
- **Organisation du circuit :**
  - ☯ Que fait ce bloc ?
  - ☯ Quelle est la place la plus adaptée pour lui dans la chaîne de traitement des données ?
  - ☯ Quel est le format des données qu'il reçoit ?
  - ☯ Quel est le format des données qu'il transmet ?

II Méthodologie de conception

1. Spécification d'un ASIC

1 Nature de la spécification d'un ASIC
2 Spécification détaillée
3 Plan méthodologique type de réalisation

2. Démarche de conception

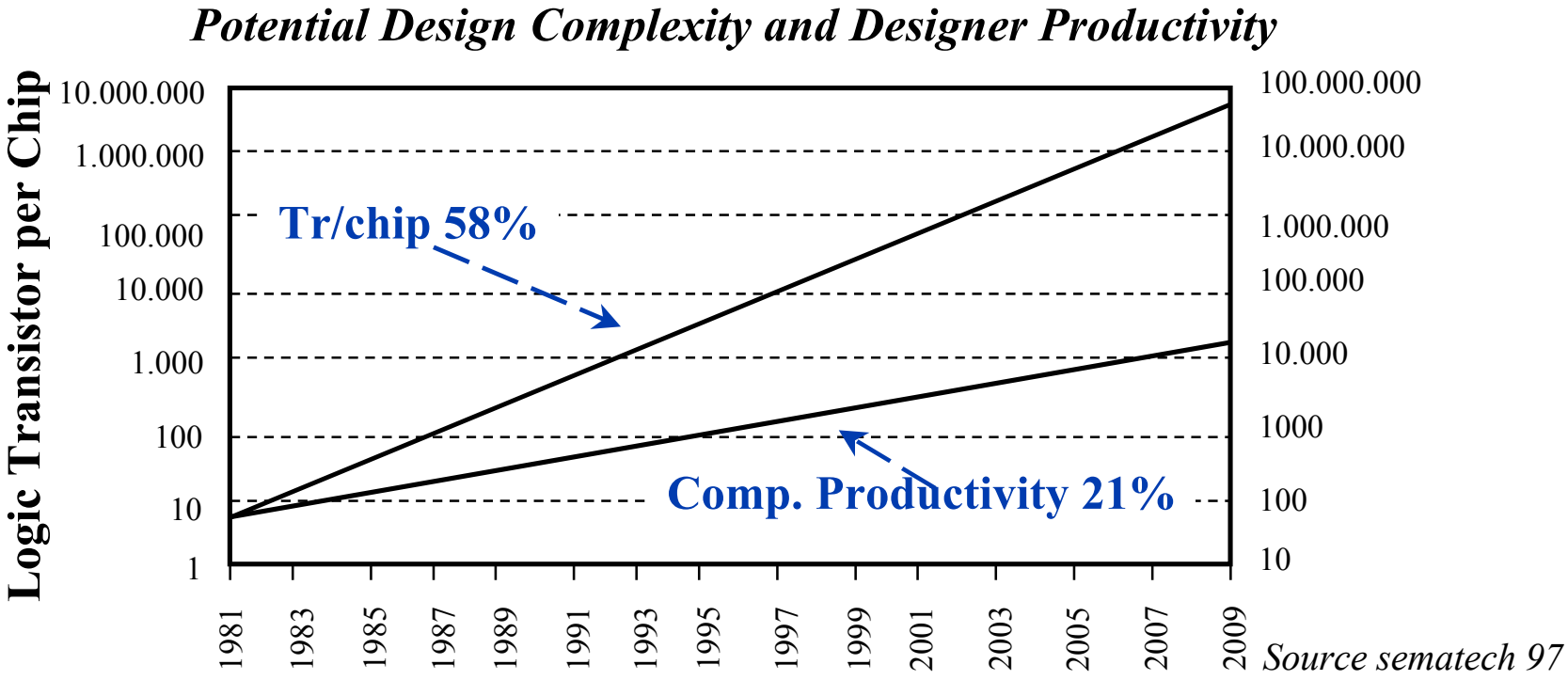
1 Méthodologies ascendante et descendante
2 Cas de la conception de circuit

3. Outils de CAO

1 Evolution des outils
2 Simulation
3 Placement et routage
4 Synthèse logique

Le problème clef du design

- **Fossé croissant entre la complexité des systèmes et la productivité des concepteurs de circuits**



II.2 Conception d'un système

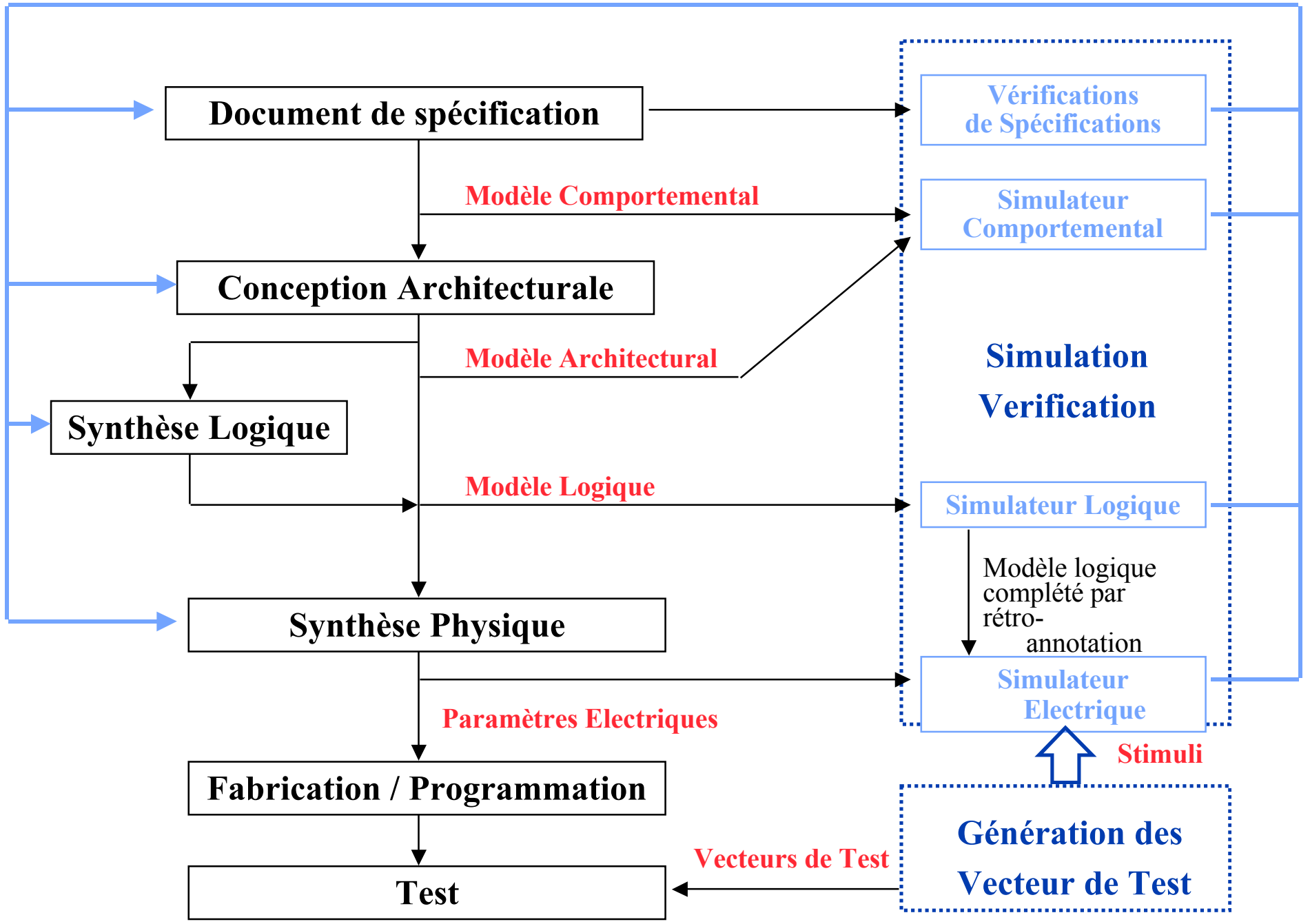
- **La conception d'un circuit ou d'un système consiste à passer d'un cahier des charges à une réalisation.**

- **On distingue quatre grands niveaux de conception :**

- *Niveau **Spécification** (ou système) : Définition du problème*
- *Niveau **Architectural**: Agencement général de la réalisation*
- *Niveau **Logique** ou logiciel: Conception détaillée*
- *Niveau **Implantation**: Réalisation physique*

La réalisation peut être matérielle ou logicielle.

Boucles de Verification

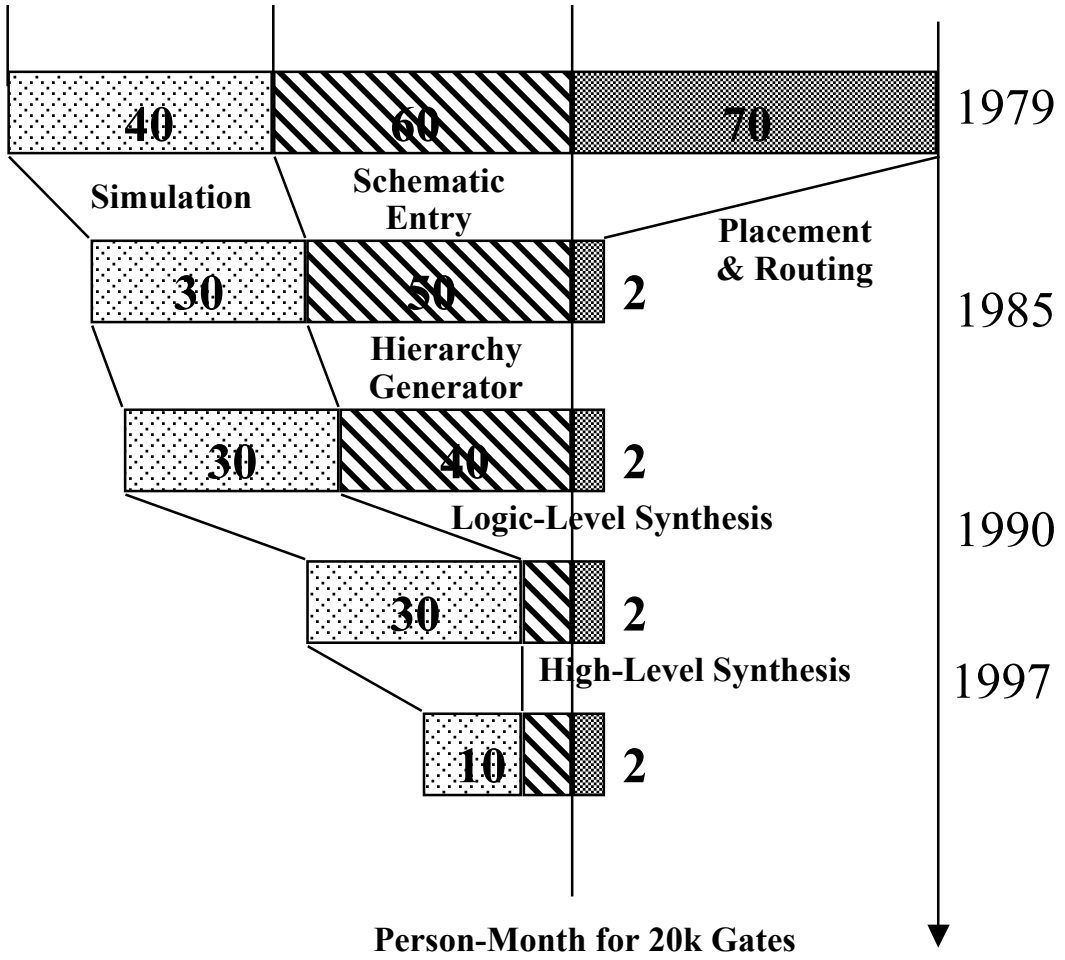


Outils de CAO de CI VLSI

Par jour un ingénieur - écrit 10 lignes de code **corrects**
- dessine 10 portes **corrects**
- place 10 cellules **corrects**

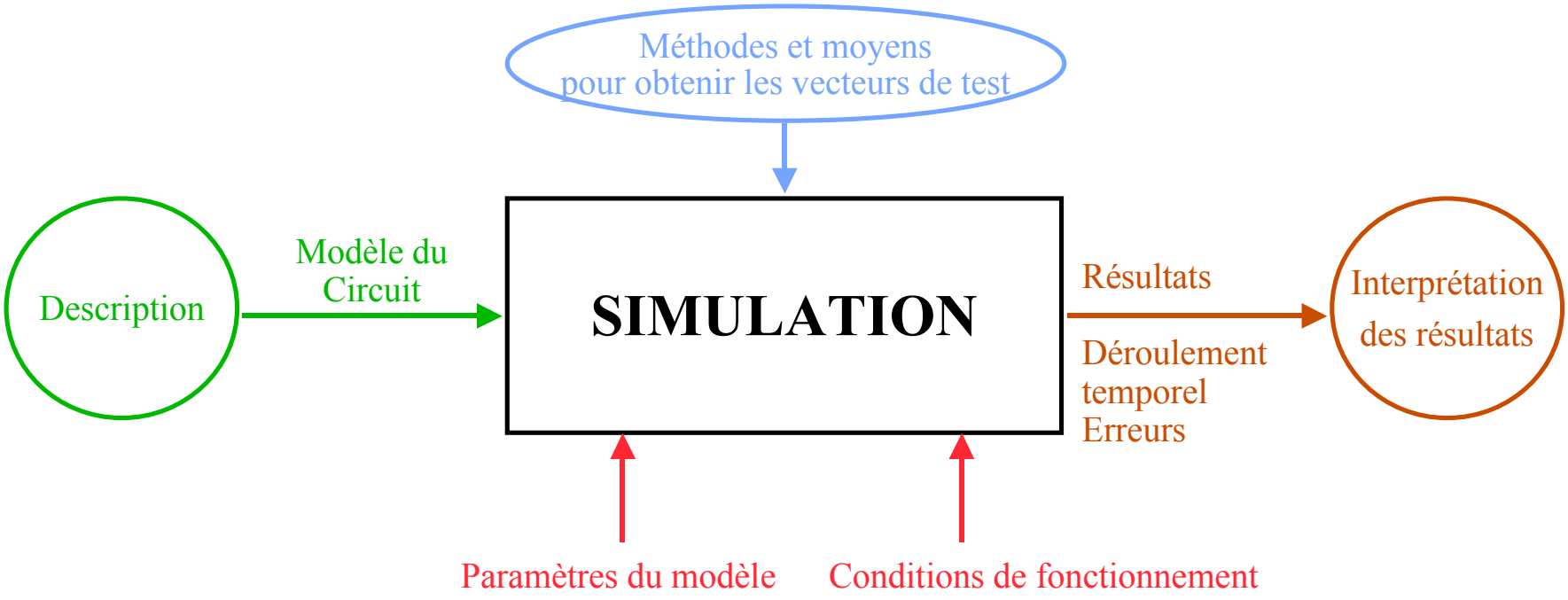
Actuellement :

- **Recherche**
  - Synthèse système
  - Preuve formelle
- **Disponible**
  - Synthèse architecturale
- **Outils du commerce**
  - Synthèse logique et RTL
  - CAO (saisie de schéma, simulation, P&R,...)



Simulation du circuit

- **Comportement du circuit une fois réalisé**
- **Simulation logique**
  - Simulateurs VHDL/Verilog : ModelSim, VSS, …
  - Modes minimum, nominal, maximum
- **Simulation électrique**
  - SPICE, ...
- **Vérifications temporelles et électriques**



II.3.3 Placement et Routage

Le **Placement Routage** consiste d'une part à trouver l'arrangement optimal des modules ou cellules composant le CI et d'autre part à effectuer l'interconnexion des signaux entre les modules, cellules et Plots d'E/S.
Avant cette opération représentait 30 à 60% du temps de conception d'un circuit. L'automatisation apporte gain de temps, souplesse de construction et sécurité du résultat obtenu.

Placement-routage : passage d'une description structurelle à une description physique

- L'entrée du processus est une liste des éléments qui constituent le circuit, et une liste de signaux qui donnent les interconnexions.
- La sortie est la liste des éléments (cellules de base, modules, plots d'E/S) et des interconnexions. Pour chacun des constituants, on connaît coordonnées, surface, orientation.
- Les deux processus sont intimement liés et **interdépendants**. Du placement dépend la longueur des connexions et la faisabilité du routage.
- En pratique, les algorithmes de placement et de routage sont adressés séparément et séquentiellement (grande complexité de l'approche globale). Cependant des rétroactions sont toujours nécessaires.
- Problème du P&R des FPGA largement aussi complexe que pour les Standard Cells.

Placement d'un CI

- **Placement**
  - Gate Array : Place et dessine la topologie des transistors associés aux différentes cellules dans la matrice du circuit
  - Standard Cell : Place les rangés/colonnes de cellules sur la puce. Toutes les cellules ont la même hauteur (ou largeur), il faut dimensionner le canal de routage. Dans une approche sur mesure, les cellules peuvent être de tailles différentes.
  - Place éventuellement d'autres blocs précompilés (ROM, CNA, ...)
  - **Doit minimiser la complexité du routage**
- **Techniques**
  - placement constructif : production d'un placement initial complet. Techniques de partitionnement, approche globale, technique de branch&bound et de cluster.
  - placement itératif : modification d'un placement pour en obtenir un meilleur. Ce processus est réitéré jusqu'à atteinte d'un critère d'arrêt (amélioration relative ou absolue, temps d'exécution). Techniques d'échanges de paires de composants, force-directed, ensembles non interconnectés, simulation de monte-carlo.
  - complexité en $O(n^2)$ ou plus : il faut donc hiérarchiser ou partitionner un gros circuit en sous problèmes solvables.
- **Intervention du concepteur**
  - partitionnement ou hiérarchisation manuel.
  - préciser les critères (surface, vitesse) à optimiser
  - contraintes sur certains nœuds (horloge, contrôle,...) par des poids de routage
  - outils d'évaluation à priori des long. des interconnexions pour influencer le placement.

Routage

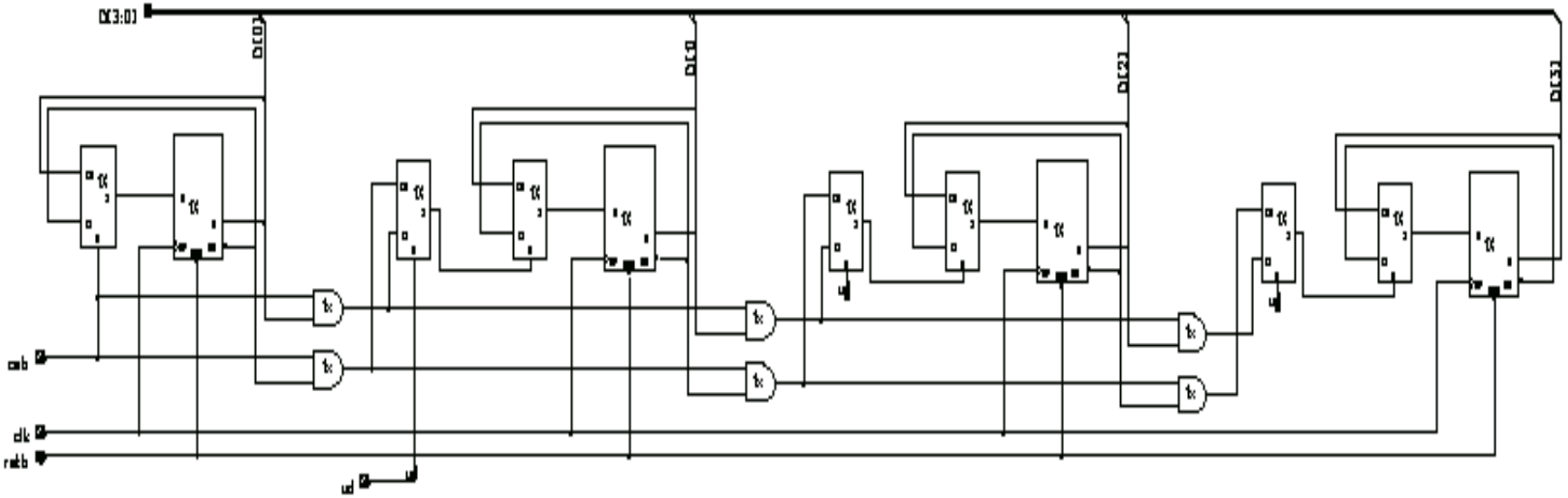
- **Routage : génération des dessins de masque réalisant les interconnexions électriques entre les composants du CI.**
  - Routage à l'intérieur d'une cellule en bibliothèque
  - Routage inter-cellules à la charge de l'outil de CAO
  - 4 niveaux de masques sur une technologie bi-metal (metal1, metal2, contact, via)
  - Distribution des alimentations, Routage de E/S, de l'horloge
  - Format **JDEC** pour les composants programmables
  - Format **CIF** pour les ASIC masqués
- **Techniques**
  - Optimisations du routage : certaines connexions trop longues causent des délais de propagation importants -> optimisation des signaux critiques (par poids de routage).

Décomposition en un ensemble de sous problèmes :

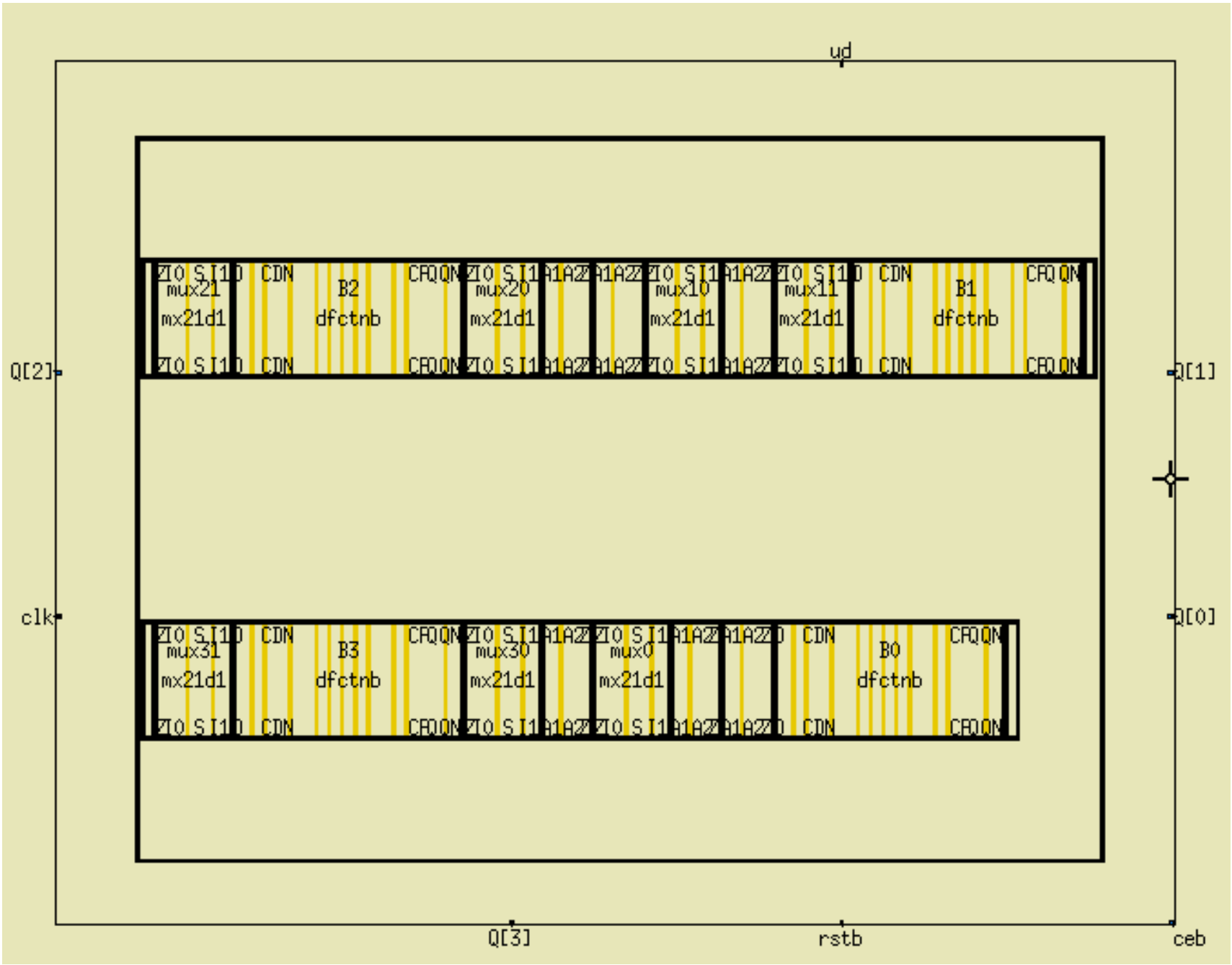
- La génération des canaux de routage divise la surface de routage en zones.
  - Cette génération est simple pour des cellules standard (CI précaractérisé).
  - On peut privilégier un découpage horizontal ou vertical.
- Le routage global assigne à chaque nœud un ensemble de canaux de routage. Cette étape définit la topologie du routage et un ensemble de problèmes restreints.
  - Prédiction du routage, réduction de la surface, minimiser la longueur
  - Améliorer la faisabilité et répartir équitablement le routage
  - Techniques : trouver un chemin pour chaque connexion à travers un graphe dont les arcs sont valués.
- Le routage détaillé résout chaque problème restreint et donne en résultat le dessin de masques des couches d'interconnexions et les contacts entre elles.
  - Certains routeurs privilégient une couche de métal pour une direction donnée.
- Un routage manuel est souvent utilisé
  - Masse, alimentations, horloge, signaux non-routés.

Placement / Routage sur Standard Cells

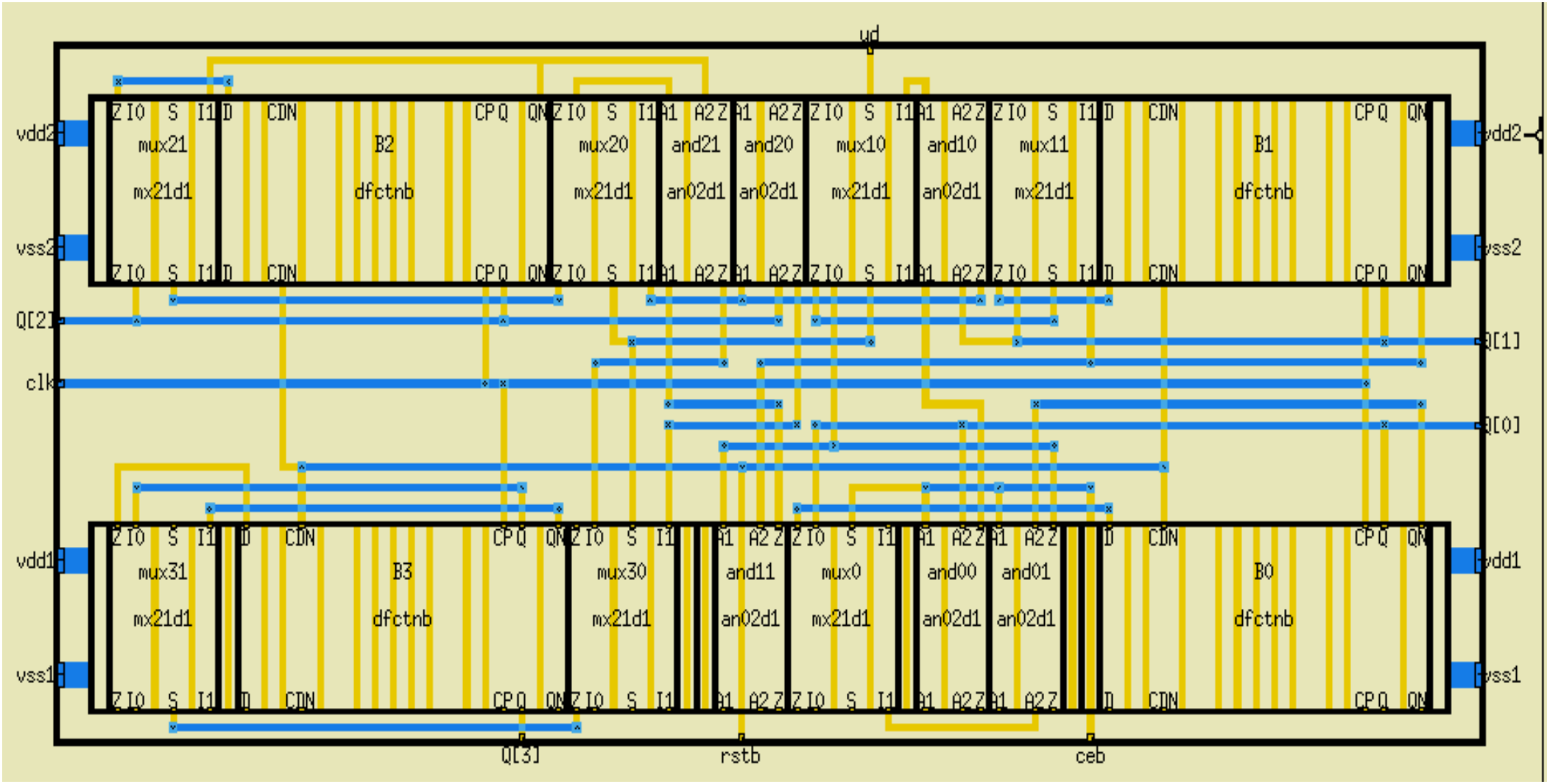
Compteur/Décompteur 4 bits avec enable



Placement du compteur



Routage du compteur

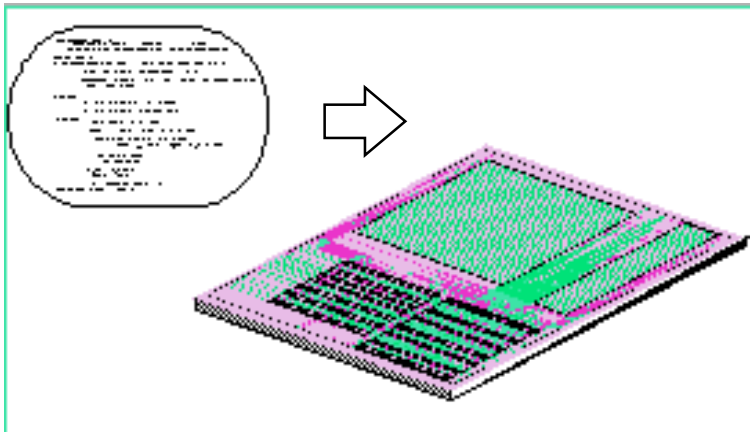


II.3.4 Synthèse logique : le challenge

- **La problématique**
  - A gain d'intégration => gain de productivité en conception
  - A accroissement de complexité => gain en sécurité
  - A demande forte du marché => rapidité de prototypage
- **Les origines de la synthèse : une rencontre**
  - Techniques de base connues depuis longtemps
  - Définition de langages pour la description de systèmes matériels (VHDL, Verilog, …)

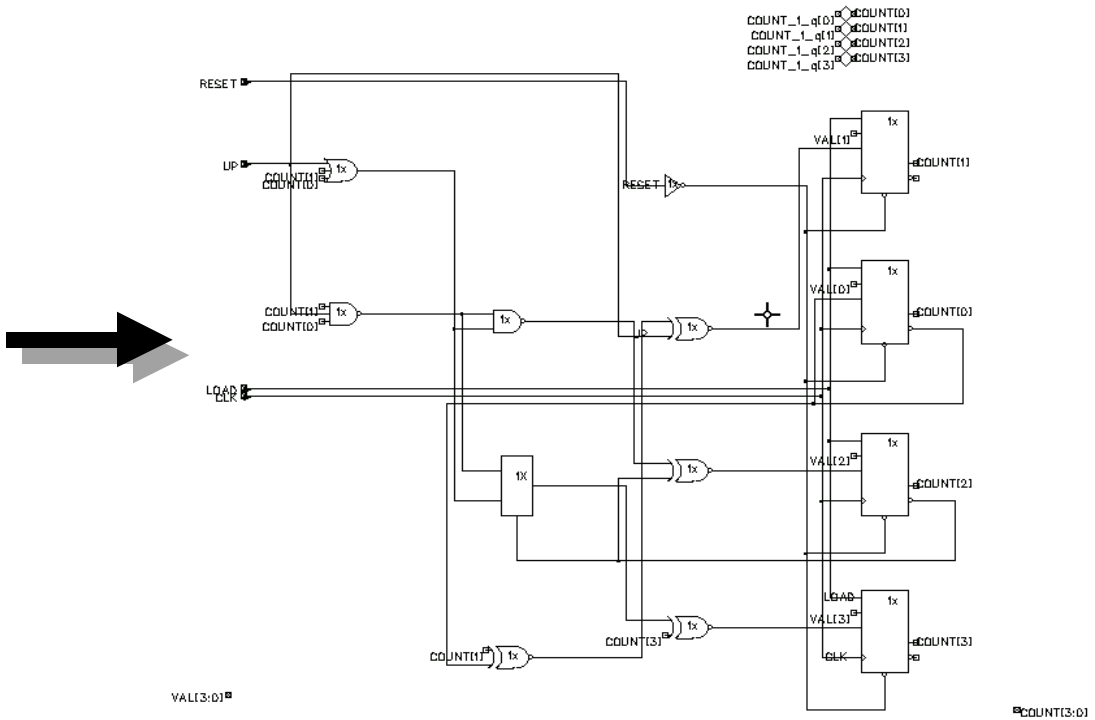
=> La conception

***devient* modélisation**

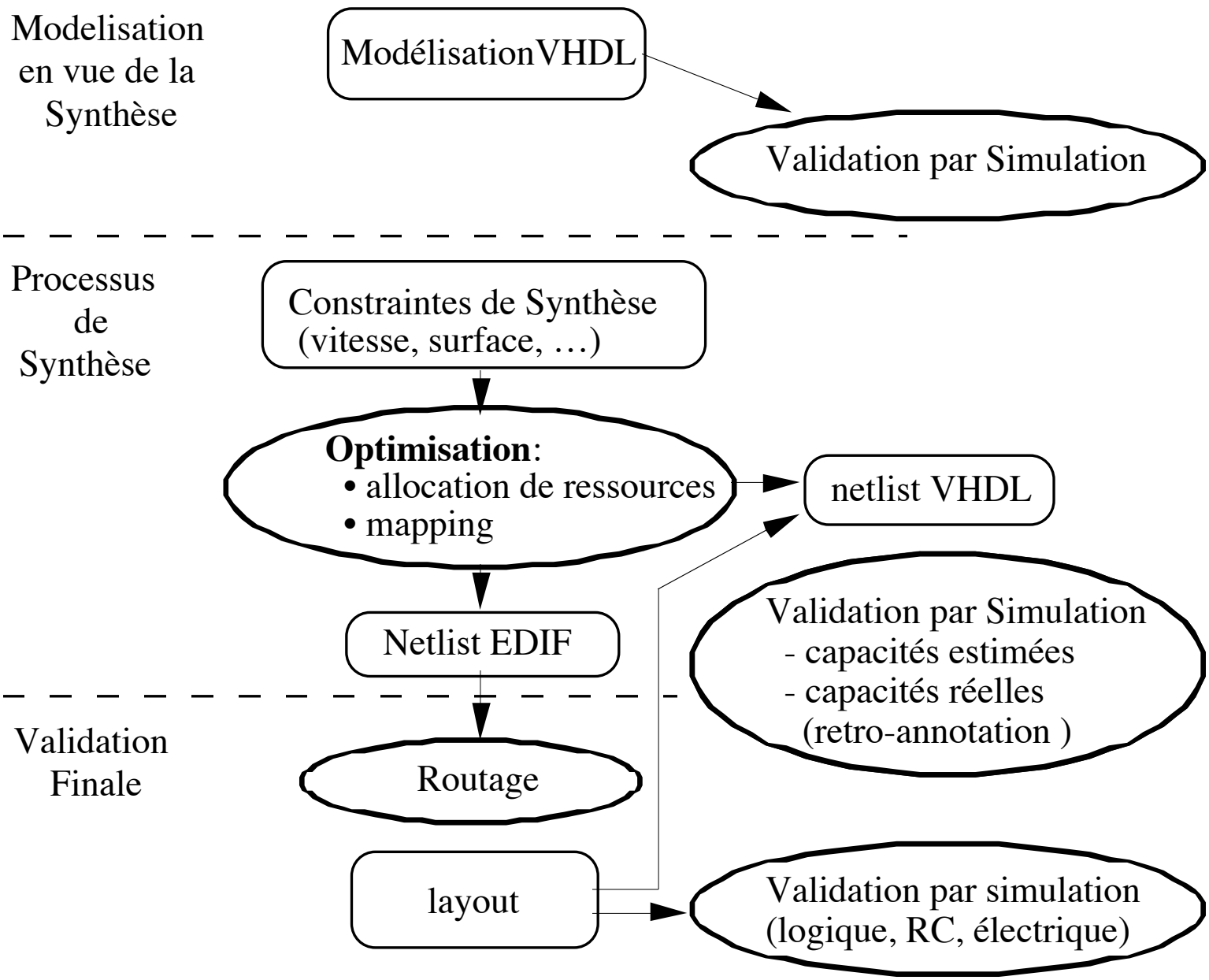


Synthèse logique

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

entity compteur is
   port (     reset, clk, load, up : in Std_Logic;
              val : in Std_Logic_Vector(3 downto 0);
              count : buffer Std_Logic_Vector(3 downto 0) );
end compteur;

architecture comportementale of compteur is
              begin
                 synchrone : process(reset,clk)
                             begin
                             if reset='1' then
                                 count <= "0000";
                             elsif clk'event and clk='1' then
                                 if load = '1' then
                                             count <= val;
                                 elsif up = '1' then
                                             count <= count + "0001";
                                 else
                                             count <= count - "0001";
                                 end if;
                             end if;
                 end process;
end comportementale ;
```



Cycle de conception avec HDL



III Conception synchrone des ASICs

1. Règles de conception

- Problèmes et paramètres généraux
- Méthodes synchrones
- Circuits recommandés
- Circuits non recommandés

2. Machine UT/UC

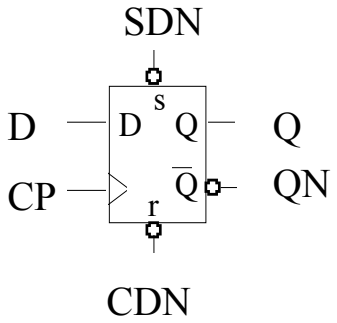
- Modèle synchrone
- Diagramme d'états
- Machine Moore/Mealy

3. Cellules arithmétiques et de mémoire

- Additionneurs, Soustracteurs
- Multiplieurs
- Mémoire

Paramètres Temporels

Bascule D



Temps de Setup
Temps de Hold
Temps de propagation
Largeur minimum d'impulsion

sur clock
reset ...

DFBTNB

D FLIP-FLOPS WITH SET AND CLEAR

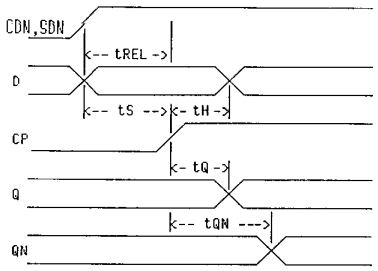
PERFORMANCE EQUATIONS

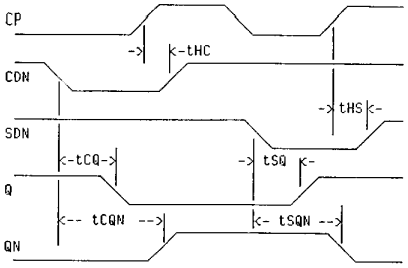
		RISE	FALL
tQ	CP->Q	2.00 + 0.22 + 2.77 * Cld	2.20 + 0.19 + 2.42 * Cld
tQN	CP->QN	2.58 + 0.23 + 3.05 * Cld	2.48 + 0.18 + 2.42 * Cld
tSQ/tCQ	SDN/CDN->Q	2.03 + 0.22 + 2.77 * Cld	0.88 + 0.19 + 2.41 * Cld
tCQN/tSQN	CDN/SDN->QN	1.69 + 0.22 + 3.03 * Cld	1.04 + 0.18 + 2.41 * Cld

Width	CP (min)	1.21 ns	Width	CDN (min)	1.36 ns
Width	SDN (min)	1.36 ns	tS,	D->CP (min)	1.13 ns
tH,	CP->D (min)	0.60 ns	tRelS,	SDN->CP (min)	0.31 ns
tHS,	CP->SDN (min)	1.00 ns	tRelC,	CDN->CP (min)	0.00 ns
tHC,	CP->CDN (min)	1.93 ns			

PROPAGATION DELAYS (ns) FOR SAMPLE LOADS, std load = 0.057 pF

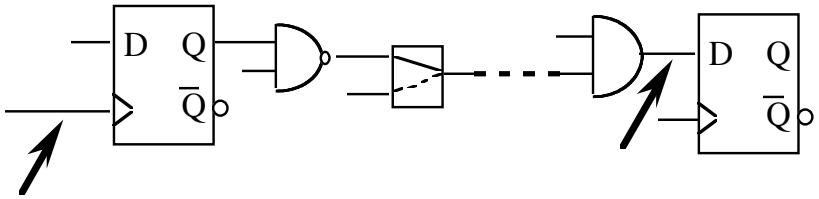
Std Load	2		4		8		16		32	
	RISE	FALL	RISE	FALL	RISE	FALL	RISE	FALL	RISE	FALL
tQ	2.54	2.67	2.85	2.95	3.48	3.50	4.75	4.60	7.27	6.81
tQN	3.15	2.93	3.50	3.21	4.20	3.76	5.59	4.87	8.37	7.07
tSQ/tCQ	2.57	1.35	2.88	1.62	3.51	2.17	4.78	3.27	7.30	5.47
tCQN/tSQN	2.26	1.49	2.61	1.77	3.30	2.32	4.68	3.42	7.44	5.61



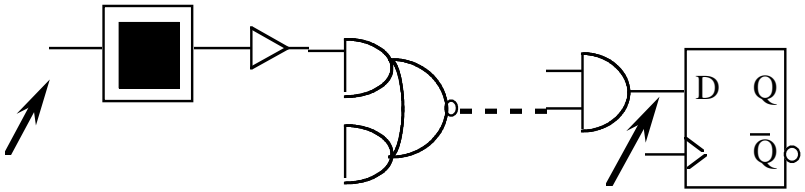


Chemin Critique

- **Principe**
  - Tout circuit possède une fréquence limite de fonctionnement.
  - La fréquence limite est liée à un seul retard : celui du chemin critique.
  - Les données doivent être prêtes (stables) avant l ’arrivée de l ’horloge, le délai de transmission maximal entre deux composant synchrones représente le chemin critique.
  - Ce délai est dû à la logique combinatoire qui limite le fonctionnement ou la vitesse d'un circuit.
- **Type de chemin critique**
  - Pad d'entrée → Combinatoire → pad de sortie
  - Pad d'entrée → Combinatoire → patte d'entrée d'un élément de mémorisation (a)
  - Patte d'horloge d'un élément de mémorisation → Combinatoire → pad de sortie
  - Patte d'horloge d'un élément de mémorisation → Combinatoire → patte d'entrée d'un autre élément de mémorisation (chemin interne) (b)
  - Ce peut aussi être un temps d ’accès mémoire, une distribution d’horloge, une opération arithmétique.



(a)



(b)

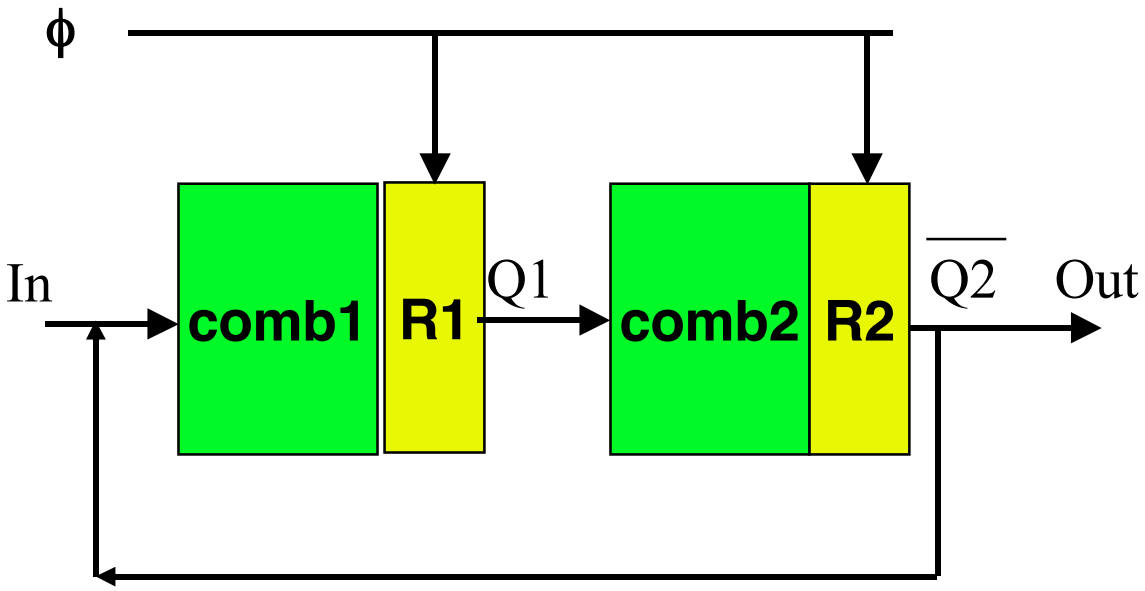
Chemin Critique

- **Exemple :**
  - Deux registres (identiques) R1 et R2 séparés par de la logique combinatoire comb2.
  - Rebouclage de la sortie Out sur l'entrée In.
  - Le chemin critique est le maximum du temps de propagation de 1 vers 2 et du temps de rebouclage 2 vers 1.

$$T_{CC} = \mathrm{Max}(T_{12}, T_{21})$$

$$T_{12} = (T_{DQ} + T_{comb2} + T_{setup})$$

$$T_{21} = (T_{D\overline{Q}} + T_{comb1} + T_{setup})$$



Meta-stabilité

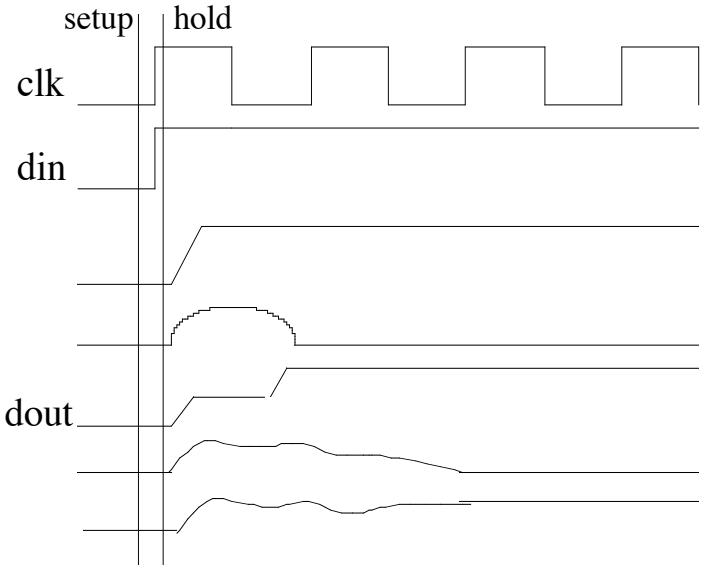
Etat analogique entre deux états logiques

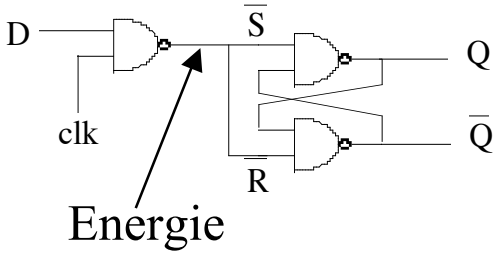
-> non-observation des temps setup et hold

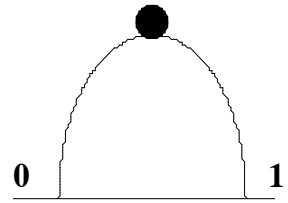
-> violation de largeur minimum d'impulsion

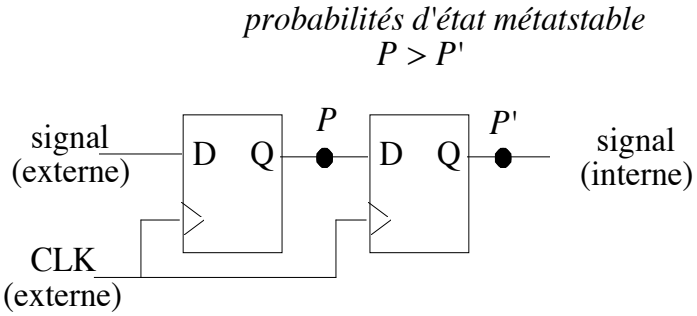
-> remise à zéro asynchrone

-> fronts lents





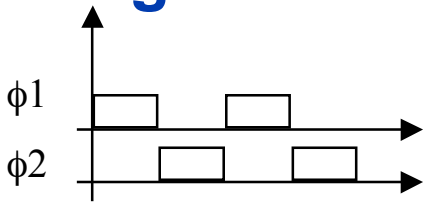


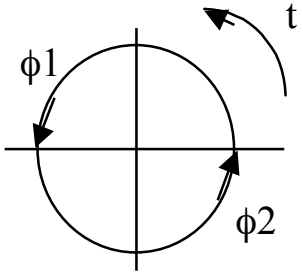


cascadage de bascules réduit la probabilité de méta-stabilité

Clock Skew

- **Décalage entre les transitions de la même horloge à différents points d'un circuit**
- **Horloge à deux phases idéale**

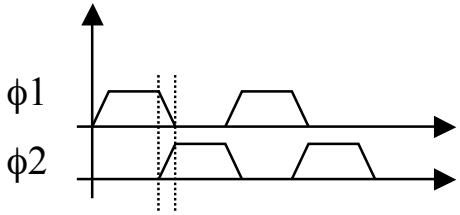


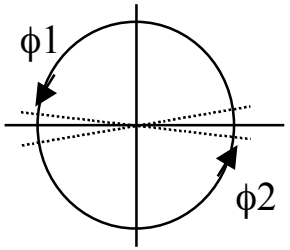


$\phi1(t) \cdot \phi2(t) = 0 \quad \forall \quad t$

Timing Circle

- **Avec fronts lents**

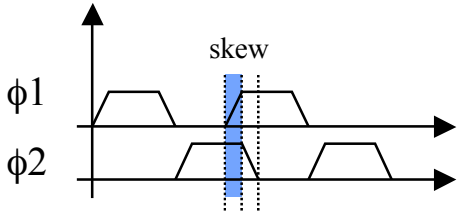


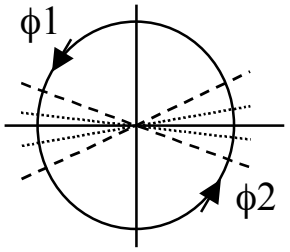


$\phi1(t) \cdot \phi2(t) = 0$ sur les niveaux

$\phi1(t) \cdot \phi2(t) \neq 0$ sur les transitions

- **Avec skew**





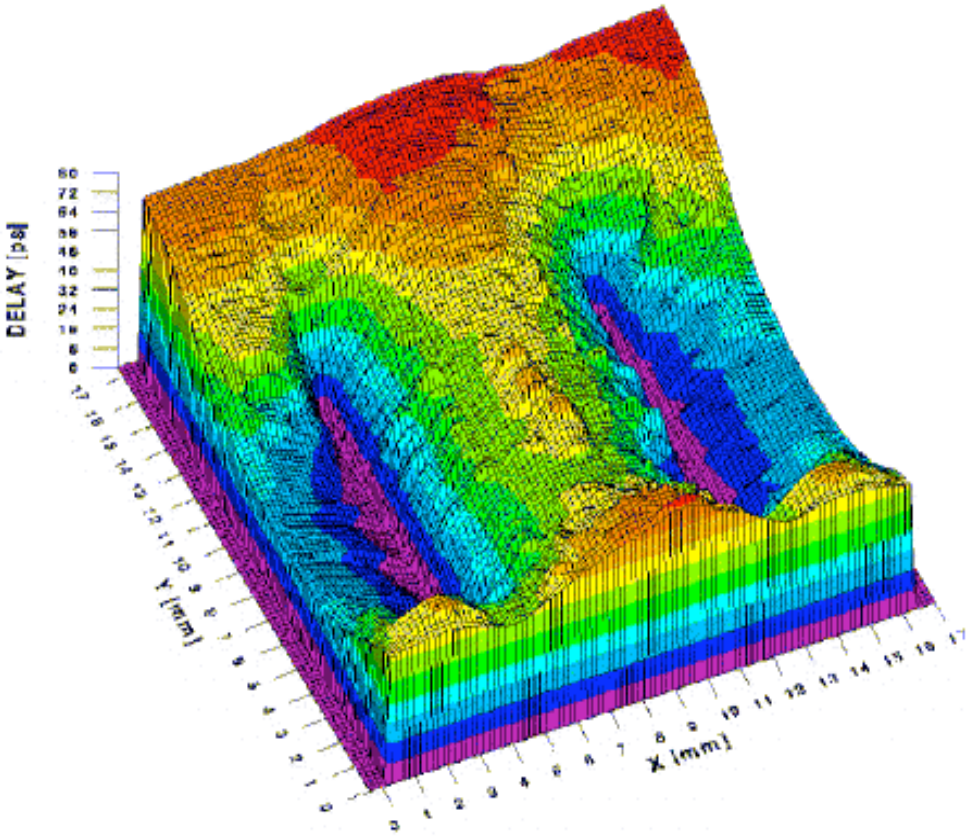
skew : temps pendant lequel $\phi1(t). \phi2(t) = 1$

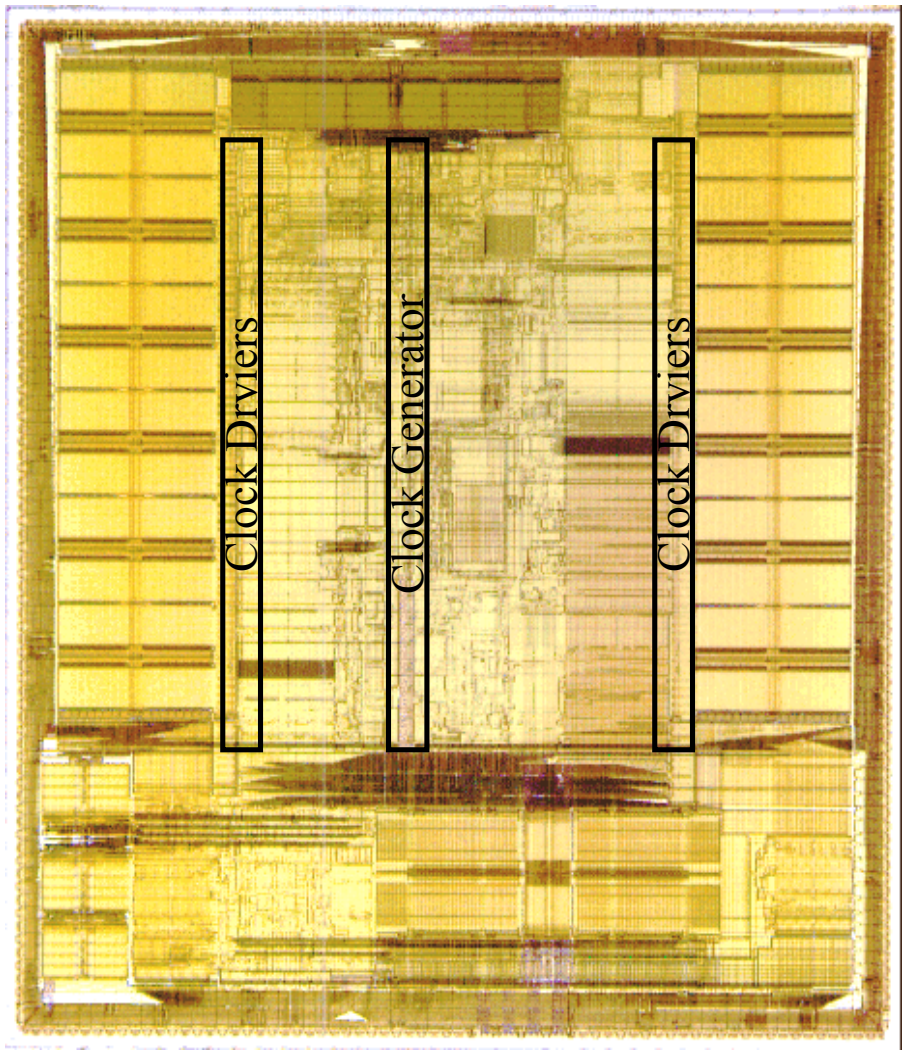
Clock Skew : Exemple du µP α

- **DEC Alpha 21164 (1995)**

  9.3M Transistors 0.5µ

  300 MHz 64/128 Bits

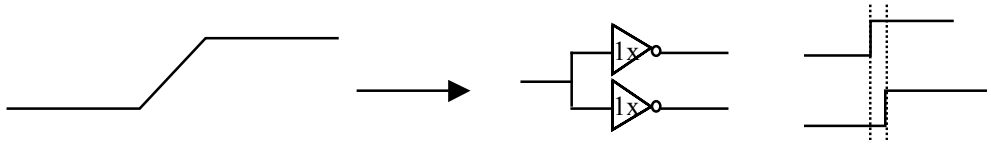




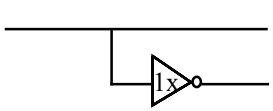
Clock Skew : causes

- **Fronts lents**

Tensions de seuil ≠



- **Génération d'horloges**

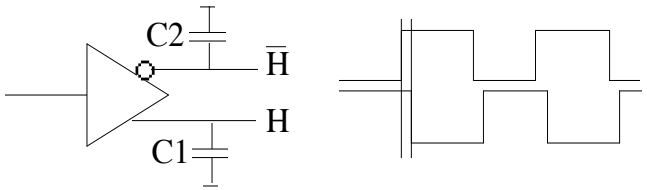


ϕ1

ϕ2

Tskew = Tinv

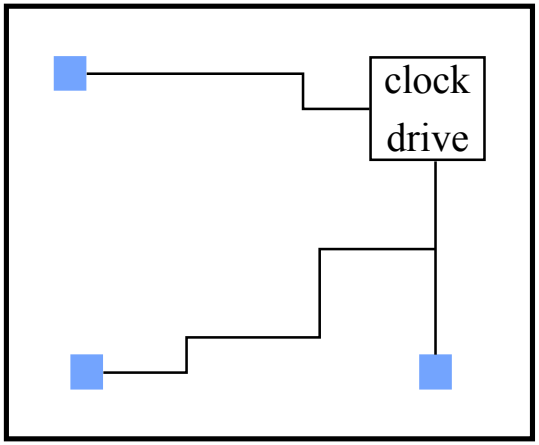
Charges différentes



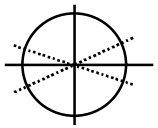
- **Routage d'horloge**

Floorplan

small skew

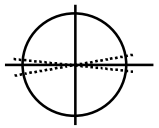


zero skew



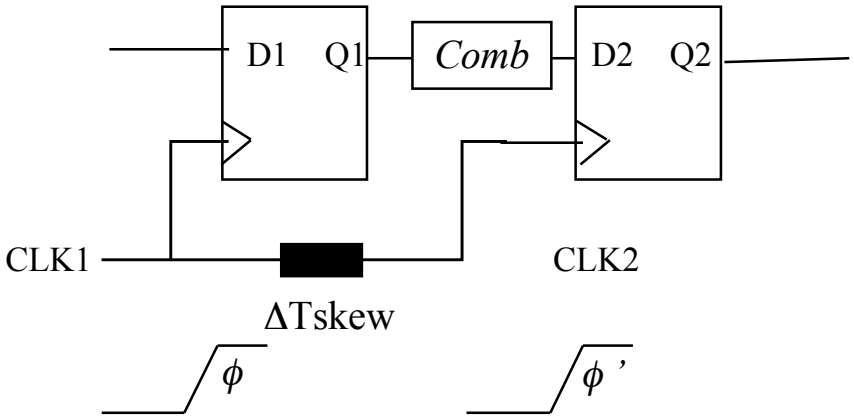
large skew

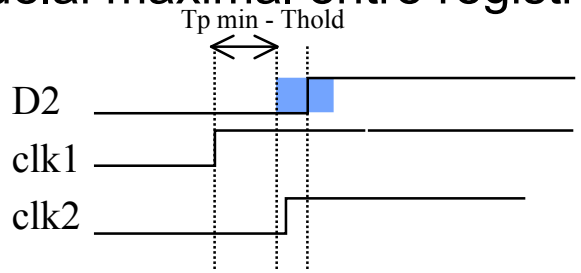
small skew



Clock Skew : problèmes et remèdes

- **Problèmes**
  - Décalage (skew) maximal déterminé par le délai minimal entre registres
  - Période d 'horloge minimale déduite du délai maximal entre registres





$$t_{\phi'} < t_{\phi} + t_{R1} + t_{itx} + t_{comb} - t_{hold}$$

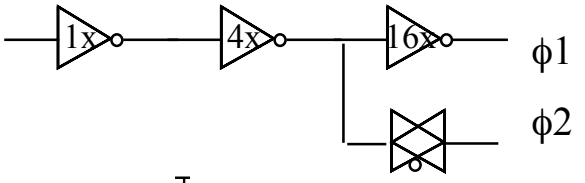
$$t_{\phi'} - t_{\phi} = \Delta Tskew < Tp\ min - Thold$$

$$avec\ Tp\ min = MIN\ (t_{R1} + t_{itx} + t_{comb})$$

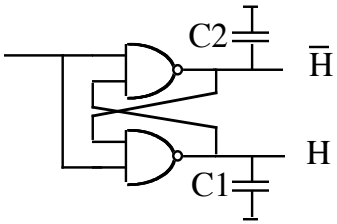
Skew min : le retard sur l'horloge ϕ ' ne doit pas être tel que le signal issu de l 'horloge ϕ soit pris en compte par la bascule suivante dans le même cycle.

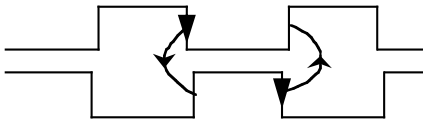
- **Remèdes**

équilibrage



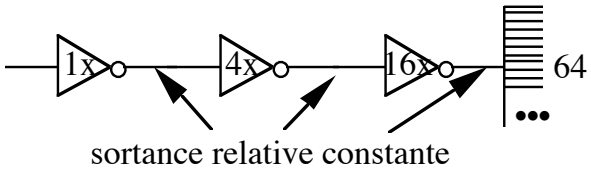
horloges non recouvrantes

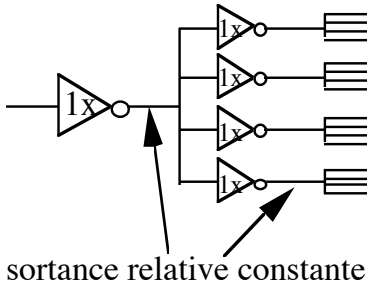


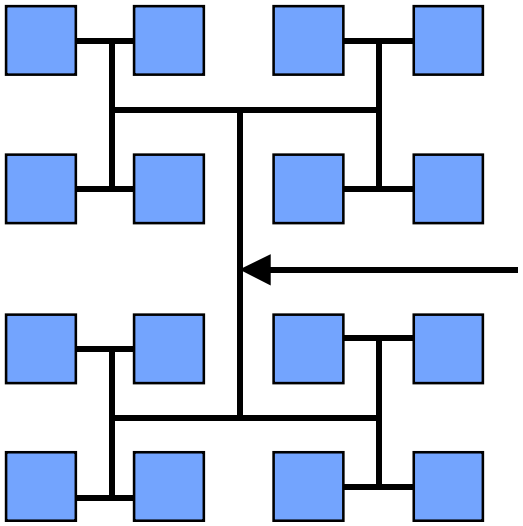


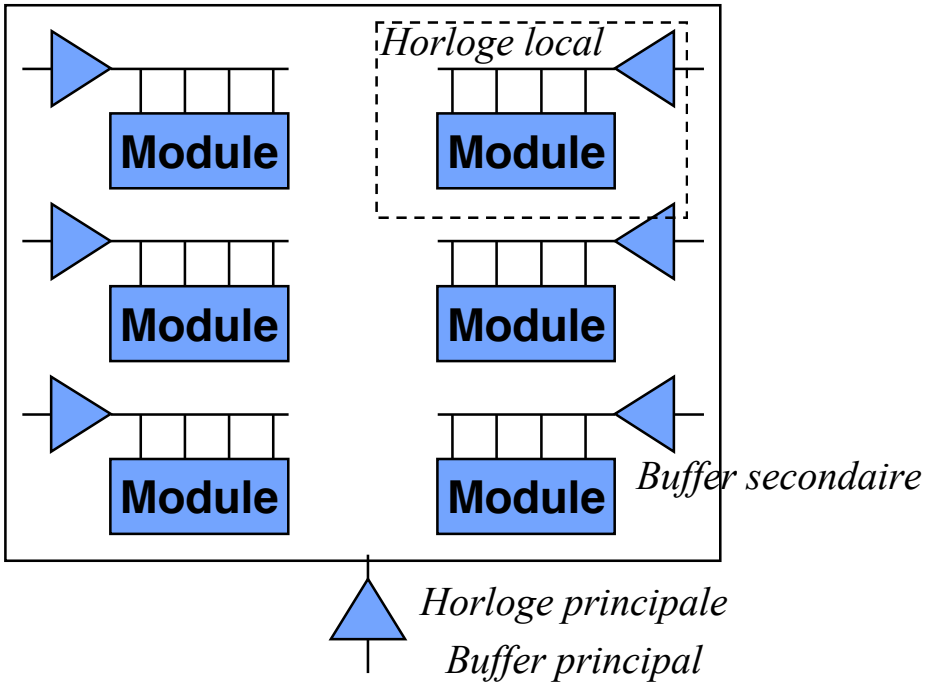
Distribution des signaux

- **Distribution de l'Horloge**
  - Géométrique : accroissement constant de la taille des buffers (délai min)
  - Arborescent
  - Géométrique et Arborescent
  - Exemples : Htree, ...









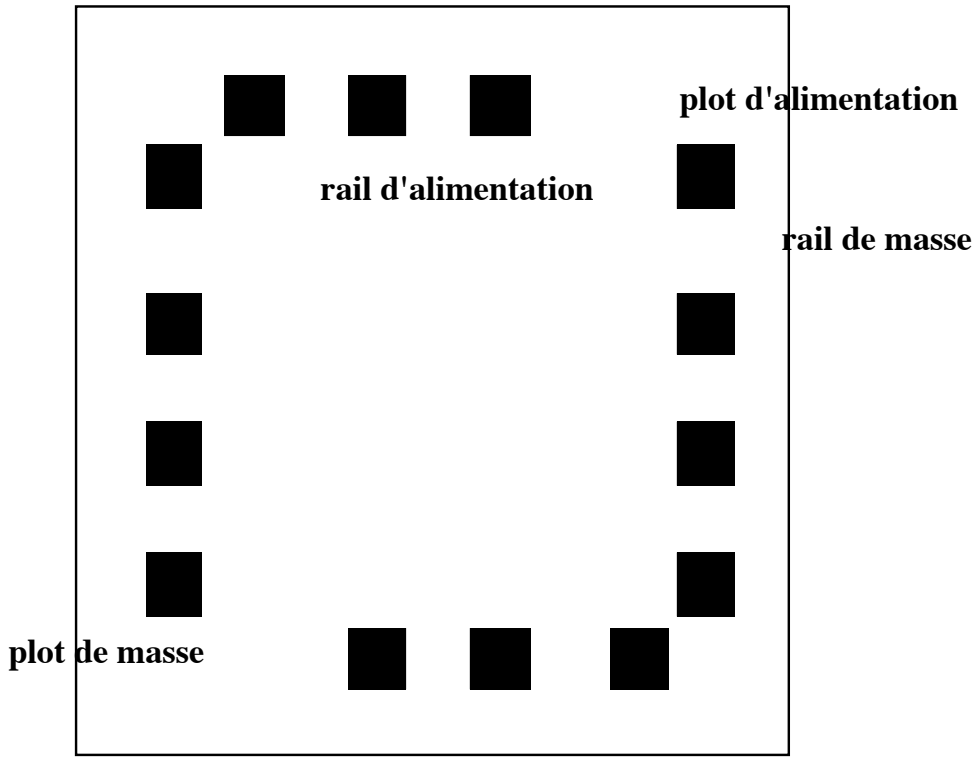
Seul le skew relatif est important.

Arbre en H : skew constant dans chaque bloc de taille égale. Bloc de taille suffisamment petite pour que le skew interne soit faible

Buffer et Arbre : réduction du délai absolu et maîtrise possible de la consommation (alim. éteinte pour les modules non utilisés)

Distribution des signaux

- **Distribution de l ’alimentation :**
  - Séparation du cœur de la puce (forte activité) des buffers d’entrées/sorties (fort courant)
  - Accès au VDD et GND par l ’intermédiaire de rails
  - Capa. de découplage alim/puce
  - Package à faible inductance et résistance



Règles de conception

Méthodes Synchrones : règles de base

« Bonne » horloge (buffer, distribution, fronts rapides,...)

Phase d'horloge connectée à toutes les bascules

Reset asynchrone global connecté à toutes les bascules

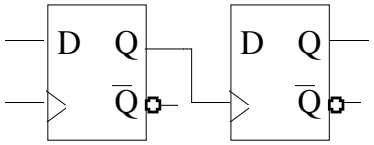
Quelques règles supplémentaires :

Eviter les rebouclages combinatoires
Minimiser le nombre de portes de transmission
Contrôle centralisé des module trois-états
Eviter les fronts lents

Circuits de base

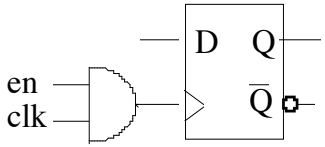
Circuits non recommandés

ripple clock



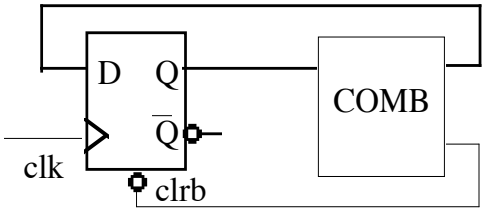
retard d'activation

gated clock



désynchronisation
pics sur enable -> clock

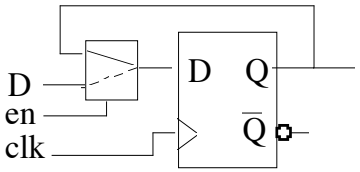
reset asynchrone



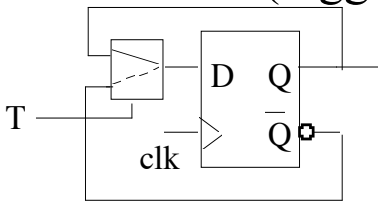
état reset transitoire
-> Tp bascules °
passage trop bref du reset
-> Treset bascules °

Circuits recommandés

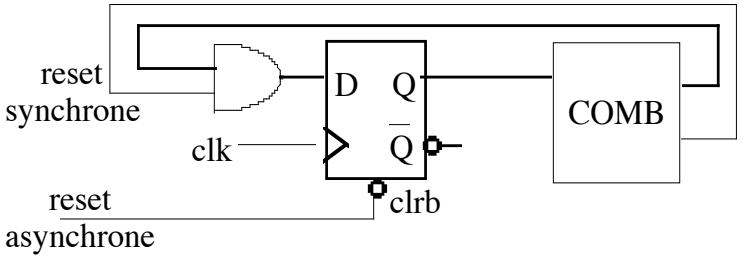
bascule E



bascule T (toggle)



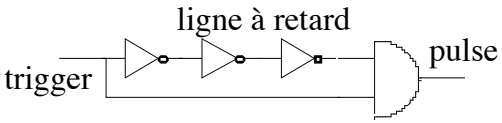
reset synchrone
reset asynchrone global



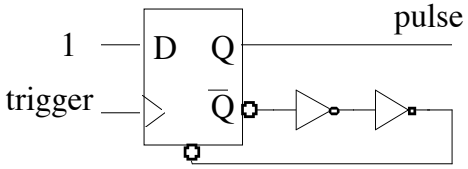
Circuits de base (suite)

Circuits non recommandés

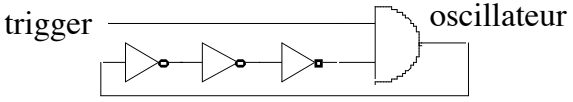
monostables



générateur d'impulsion

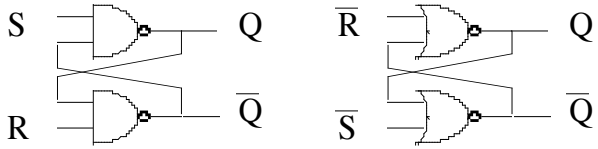


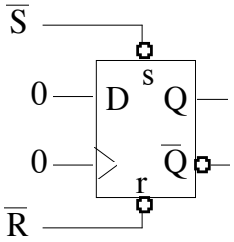
oscillateurs



lignes à retard

bistables RS
(mémorisation)

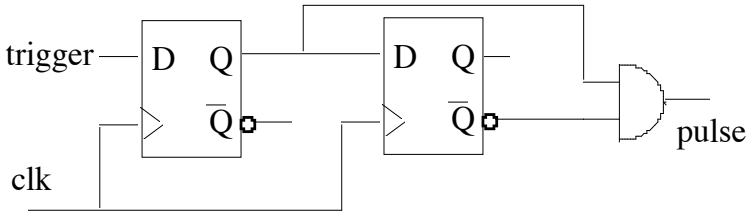




états inconnus
rebouclage combinatoire

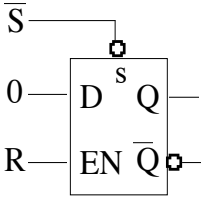
Circuits recommandés

générateur d'impulsion



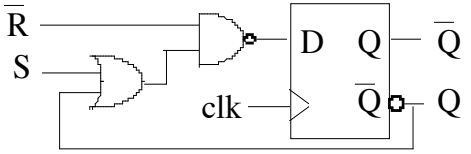
synchrone

latch en RS



états toujours connus

RS synchrone



reset prioritaire

III Conception synchrone des ASICs

1. Règles de conception

- Problèmes et paramètres généraux
- Méthodes synchrones
- Circuits recommandés
- Circuits non recommandés

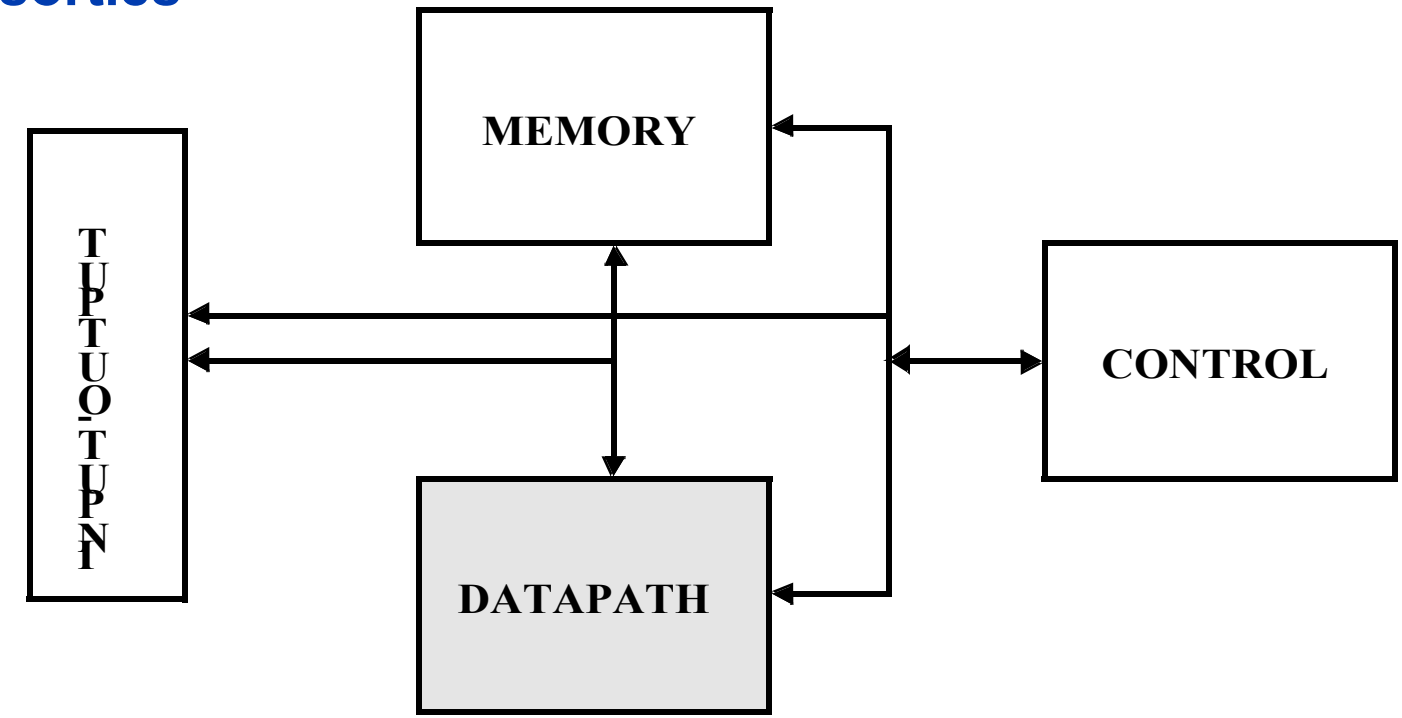
2. Machine UT/UC

- Modèle synchrone
- Diagramme d'états
- Machine Moore/Mealy

3. Cellules arithmétiques et mémoire

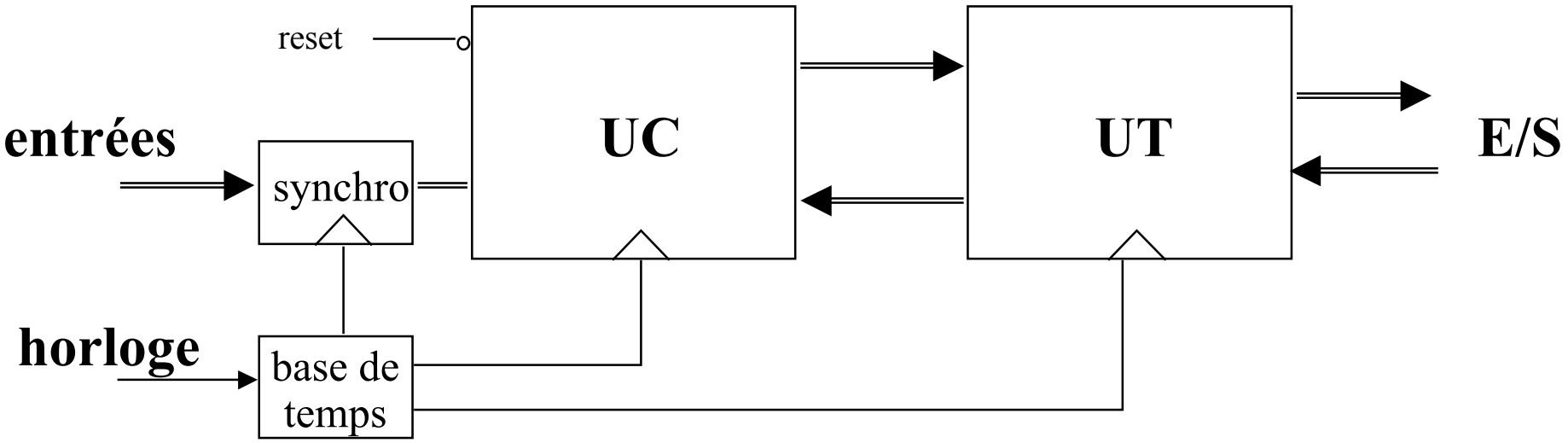
Architecture générale

- **Unité arithmétiques : *datapath***
  - Additionneur, multiplieur, décaleur, comparateur, …
- **Mémoire**
  - RAM, ROM, registre, banc de registres, buffers, registres à décalage
- **Contrôle**
  - Machine d'états (PLA, logique combinatoire, registre), compteurs, ROM
- **Interconnexions**
  - Bus, arbitres, trois-états, multiplexeurs, …
- **Entrées-sorties**



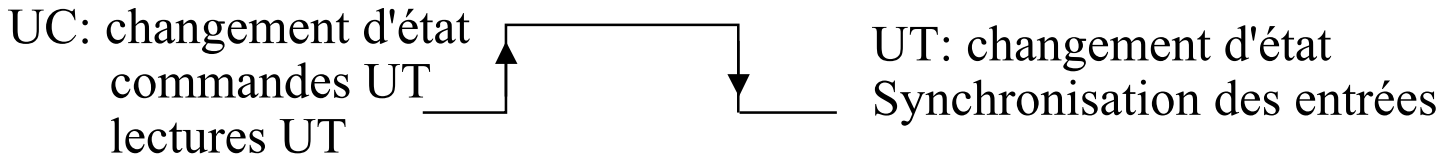
Machine UT/UC câblée

Modèle synchrone



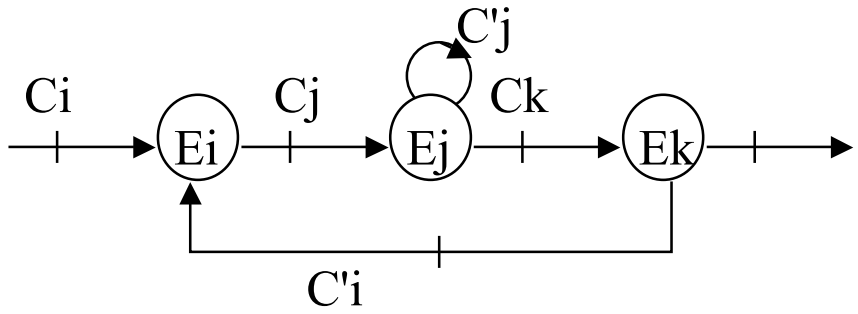
UC : Machine MOORE
Entrées Synchronisées (méta-stabilité)
RAZ global du circuit -> connaître l'état initial

***Base de temps** : données stables pendant leur échantillonnage*



Unité de contrôle : spécifications

Diagramme d'état



Transitions : conditions booléennes des entrées
horloge implicite
conditions de sortie disjointes
rebouclage sur un état implicite

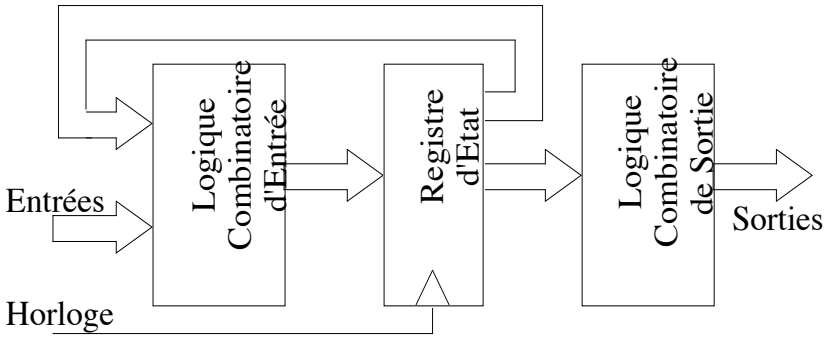
Nœuds : états stables de l'UC
nombre d'états -> taille du registre d'état

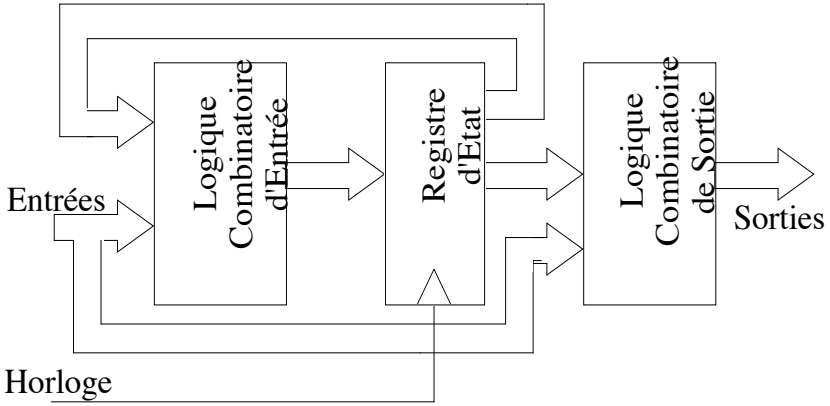
Codage des états : valeur numérique du registre d'état (Ei -> b0001)
types de codages (binaire, gray, johnson,...)

Sorties de l'UC : décodage de l'état

Précautions : Machine de Moore plus sûre
Forcer le codage des états non-atteints (struck states)
Entrées et états stables lors de leur échantillonnage
Le codage influe sur les parasites de sortie et sur la fréquence MAX

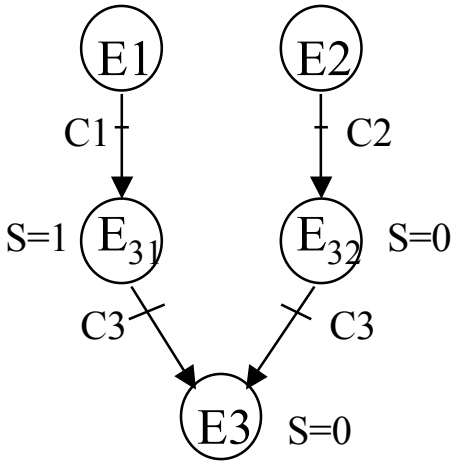
Machine de Moore / Machine de Mealy





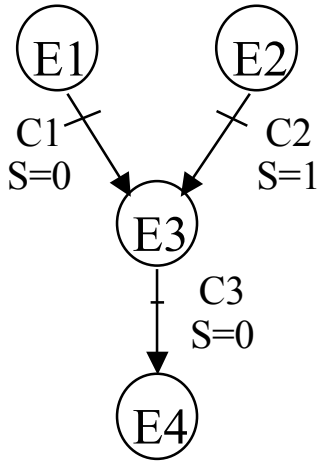
Sorties = F(EtatCourant)
EtatSuivant = F'(EtatCourant, Entrées)

Sorties = F(EtatCourant, Entrées)
EtatSuivant = F'(EtatCourant, Entrées)



transformabilité

Moore<-> Mealy

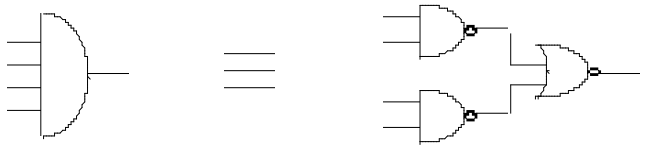


Synchrone
Plus sûre de fonctionnement
Plus d'états

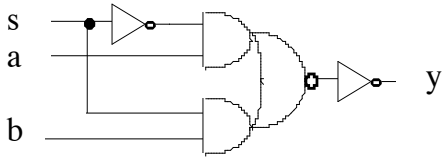
Asynchrone
Plus rapide
Aléas de fonctionnement

Vitesse

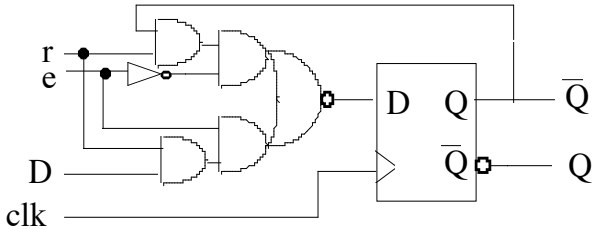
Réduire le nombre d'entrées



Utilisation de la logique AND/OR/INV (AOI) ou OR/AND/INV (OAI)



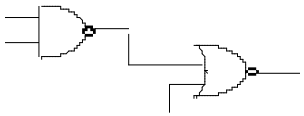
Multiplexeur en AOI



Bascule E avec reset en AOI

Réduire la taille des chemins critiques
Distribution des signaux sur le chemin critique optimisée

signaux non critiques
(early-changing inputs)

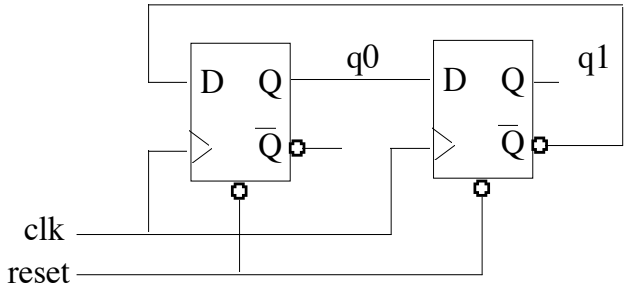


signaux critiques
(late-changing inputs)

Vitesse (suite)

Compteur de Johnson

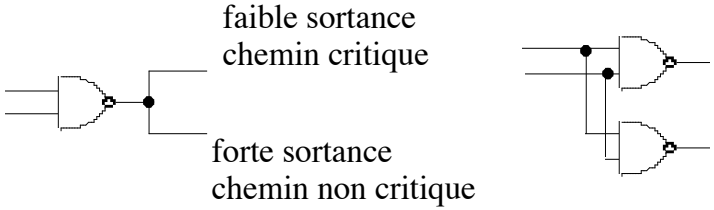
vitesse maximum
plus coûteux : M/2 > log2(M)



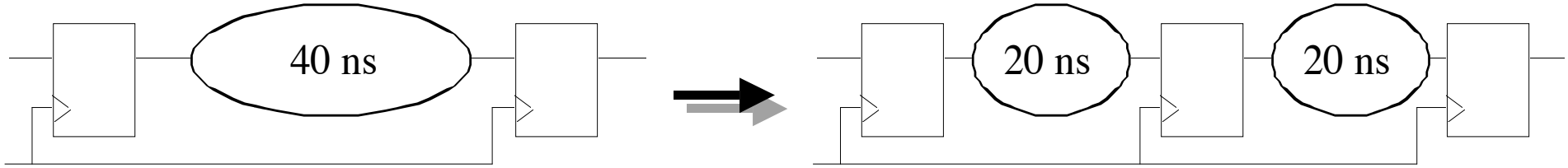
q1	q0
0	0
0	1
1	1
1	0

M bascules
2M états

Duplication pour réduire la sortance



Utilisation des cellules rapides de la librairies



fmax = 25 MHz

fmax = 50 MHz

Pipeline

V Synthèse logique

1. Synthèse logique à partir de VHDL

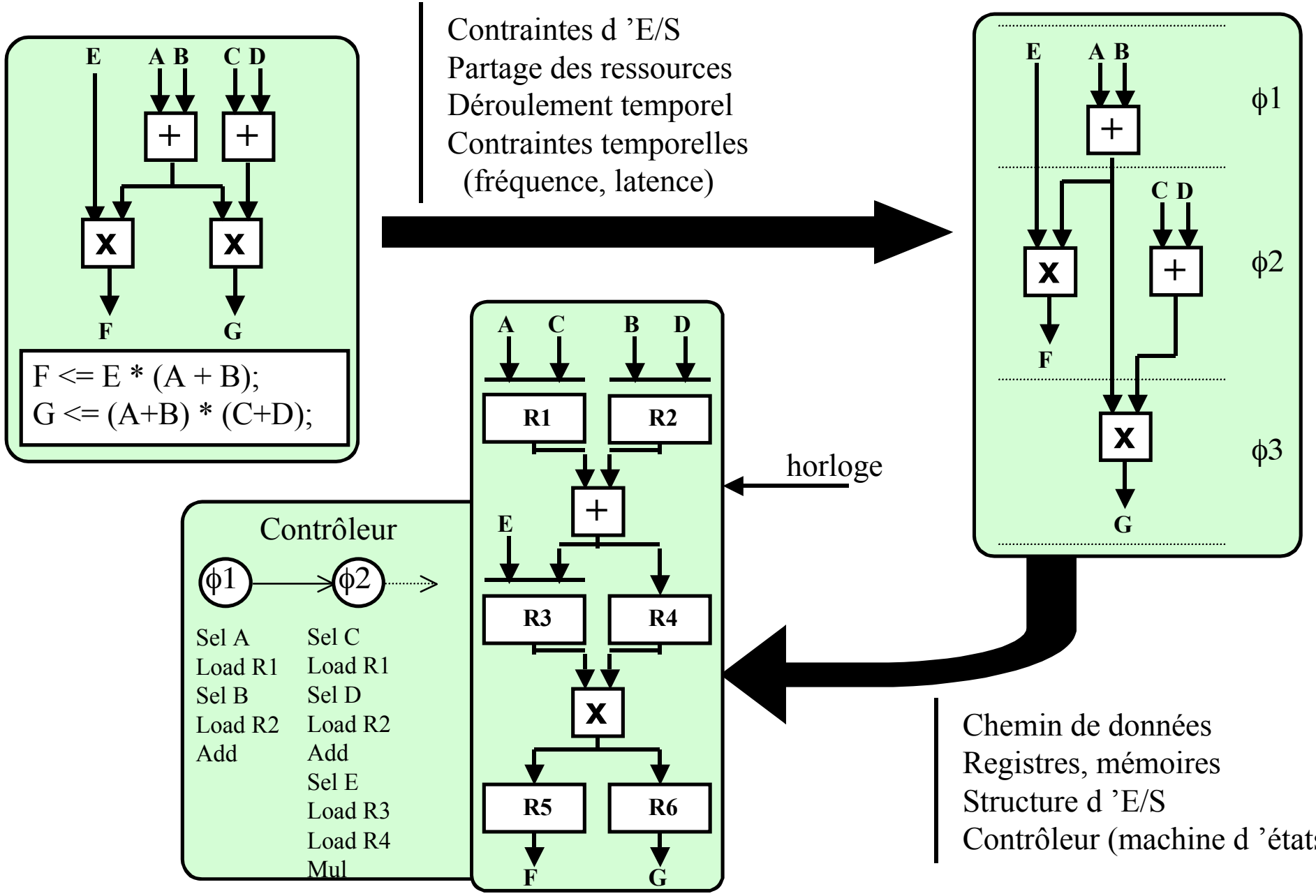
- Principe et Flot de conception
- VHDL pour la synthèse
- Styles de description
- Logique combinatoire
- Logique séquentielle

2. Mécanismes de synthèse logique

- *Optimisations*
- *Structuring / Flattening / Mapping*

3. Conclusions générales

Synthèse comportementale



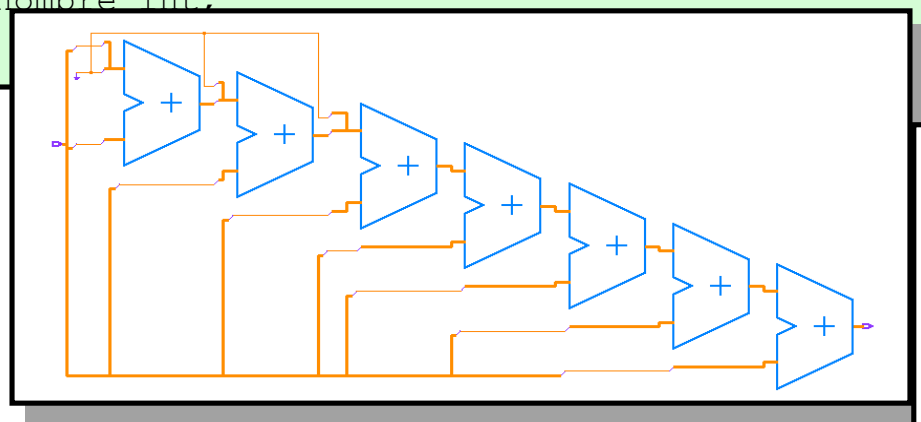
Synthèse logique

Exemple Fonction combinatoire : en entrée : un mot de 8 bits
en sortie : le nombre de ‘ 1 ’ présent dans ce mot.

Description algorithmique

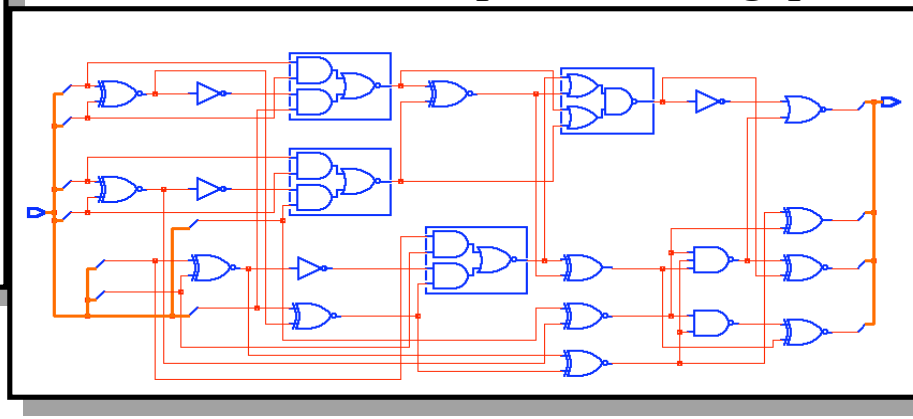
```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
entity nb2 is       port(DIN : in std_logic_vector(7 downto 0);
                    nombre : out integer range 0 to 8);
end nb2;
architecture arch of nb2 is
begin
process(DIN)
          variable nombre_int : integer range 0 to 8;
begin
    nombre_int := 0;
    for i in 0 to 7 loop
        nombre_int := nombre_int + conv_integer(DIN(i));
    end loop;
    nombre <= nombre_int;
end process;
end arch;
```

$$nombre = \sum_{i=0}^{7} DIN(i)$$



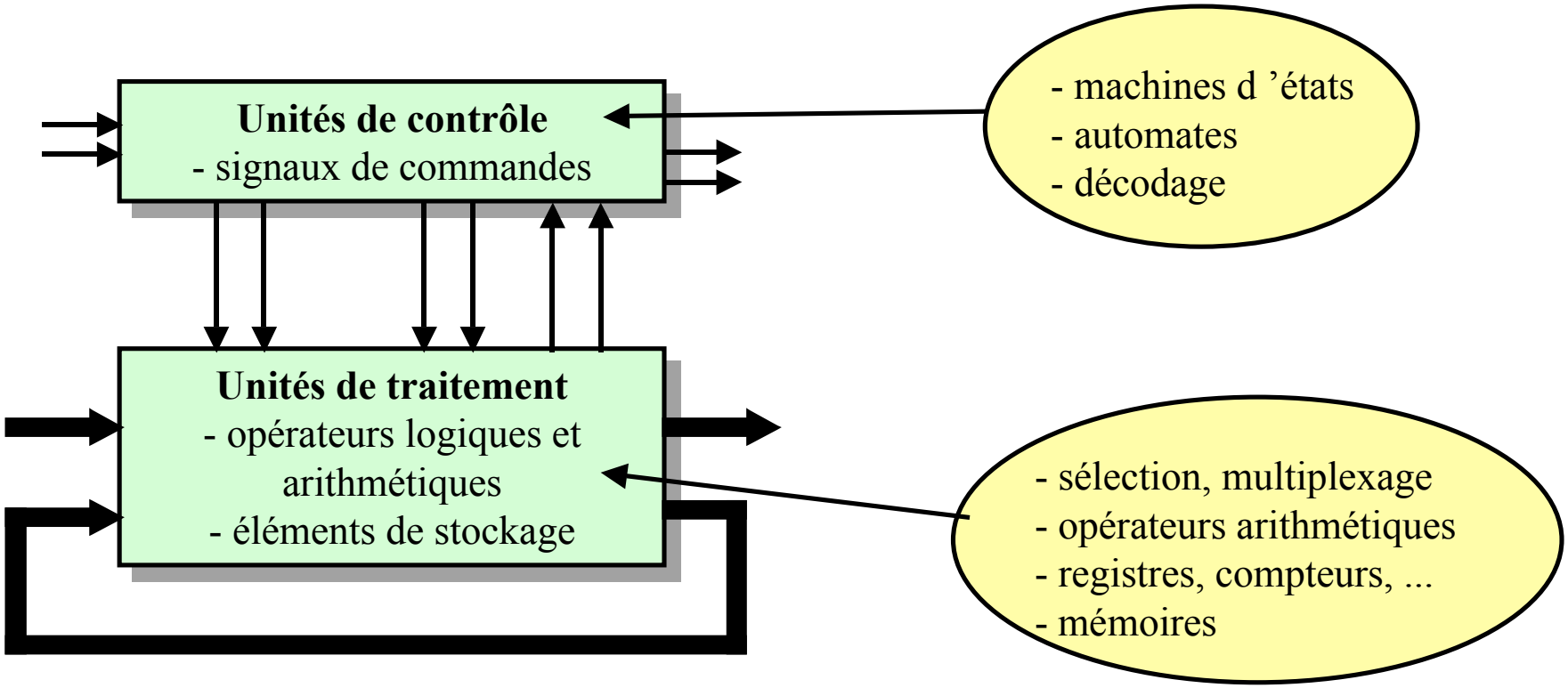
Synthèse : 7 additionneurs en cascade

Optimisation logique



Synthèse logique

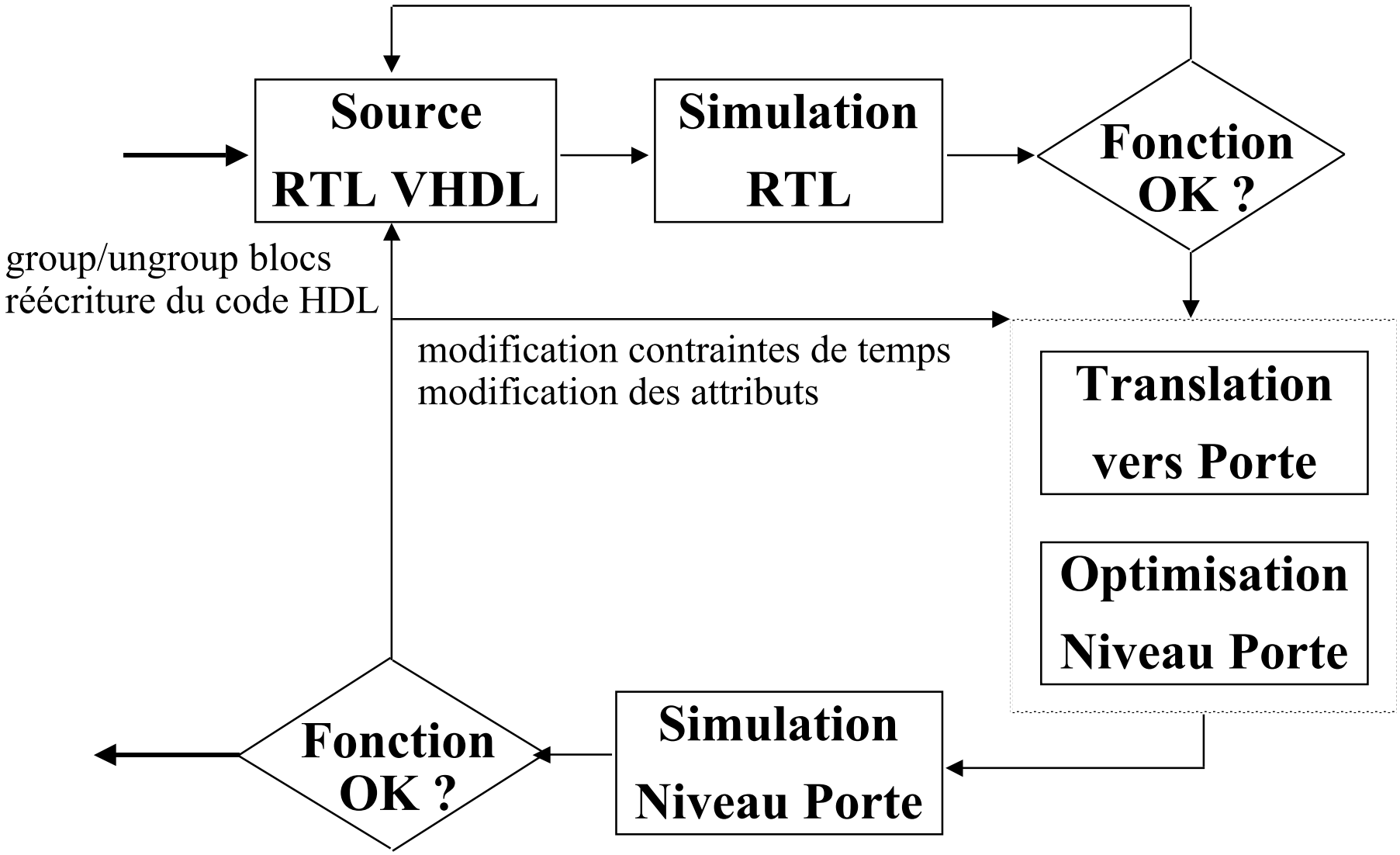
- **Synthèses des fonctions logiques**
  - Architecture d ’un système numérique



Synthèse Logique : avantages

- **Automatisation du flot de design**
- **Permet de concevoir des circuits plus complexes**
- **Conception indépendante de la technologie**
- **Domaine d'exploration des solutions plus important**
- **HDL plus flexible, plus lisible**
  - Nouveau niveau d'abstraction plus élevé
- **Différentes descriptions**
  - (équations, diagramme d'état, netlist, description RTL)
- **Dirigé par des directives et des buts**
- **Résultats optimisés et analysés**

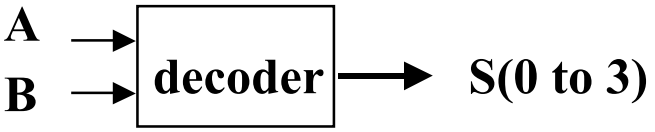
Flot de Synthèse



VHDL : rappels

Vue extérieure : ENTITY

```
entity decoder is
      port ( A, B :  in bit;
                S      :  out bit vector (0 to 3) );
end decoder ;
```



Description interne : ARCHITECTURE

```
architecture comportementale
   of decoder is
begin
          process (A,B)
          begin
                    if A='0' then
                      if B='0' then
                        S<= "0001"
                      else
                        S<= "0010"
                      end if;
                    else
                      if B='0' then
                        S<= "0100"
                      else
                        S<= "1000"
                      end if;
                    end if;
          end process ;
end comportementale ;
```

```
architecture flot_données
   of decoder is
begin
          S(0) <= not(A) and not(B) ;
          S(1) <= not(A) and B ;
          S(2) <= A and not(B) ;
          S(3) <= A and B ;
end flot_données ;
```

```
architecture structurelle
   of decoder is
component DEC24
   port ( I1, I2 :                  in   bit;
             O1, O2, O3, O4  :  out bit  );
end component ;
begin
   CELL : DEC24
   port map
        (A,B,S(0),S(1),S(2),S(3));
end structurelle ;
```

Niveau RTL

```
entity compteur is
    port ( reset : in bit;
           S     : out integer );
end compteur;

architecture behavioral OF compteur is
begin

    process
        variable count : integer;
    begin
        wait until reset = '1' for 20 ns;
        if reset = '1' or count = 15
            then count :=0;
        else count := count + 1 after 10 ns;
        end if;
        S <= count;
   end process;
end behavioral;
```

Non Synthétisable

```
entity compteur is
    port ( reset : in bit;
           clk : in bit;
           S : out integer range 0 to 15 );
end compteur;

architecture RTL OF compteur is
  signal count : integer range 0 to 15;
begin

    process(reset,clk)
    begin
        if reset='1' then count <=0;
        elsif clk'event and clk='1' then
            if count=15 then count <= 0;
            else count <= count + 1;
            end if;
        end if;
   end process;
   S <= count;
end behavioral;
```

Synthétisable

Sous ensemble de VHDL

Entity : Types des ports d'Entrées / Sorties de taille bornée
(*in*, *out*, *inout*, *buffer*). Valeurs par défauts ignorées

Architecture : Une seule architecture

Configuration : Component : *for all <nom> use entity <package>*

Librairies : Compilation séparée : non supportée

Packages : Collection de ressources
(types, constantes, fonctions, procédures, composants)
Packages Standards (STD_LOGIC...) ou CAO (Compass...)

Types : Integer avec range
Enumération, Record, Subtype
Tableaux 1D de dimension finie
Bit, Bit_Vector ou STD_LOGIC, STD_LOGIC_VECTOR
Physique, Réel, Access, File : ignorés

Déclarations : *Constante*, *Signal*, *Variable*, *Component* acceptés
Register, Bus, Linkage, Alias : ignorés

Opérateurs : Logiques (and, nand, or, nor, xor, not)
Relationnels (=, /=, <, >, <=, >=)
Arithmétiques (+, -, *, signe, abs)
(/, mod, rem,** pour puissance de 2)

Attributs : 'length, 'event, 'left, 'right, 'high, 'low, 'range, 'reverse_range

Sous ensemble de VHDL (2)

Instructions Séquentielles :

Wait : Supporté en première ligne d'un PROCESS synchrone :
wait until clock = value;
wait until clock'event and clock = value;
wait until not clock'stable and clock = value;

Affectations : Affectation de variables et de signaux supportée
Fonctions et Procédures : supportées
Transport et after : ignorés, Assertion : ignorée

If/Case : Supportés

Boucles : Boucle *for* de taille statique : supportée
Boucle *while* : non supportée

Instructions Concurrentes :

Process : Liste de sensibilité ou wait => combinatoire ou synchrone

Affectations : Affectation conditionnée de signaux supportée (*when, select)*

Block : Guards, ports et génériques : non supportés

Instanciation : Port map, Generic map, generate : supportée

Sous Prog. : Procedures et Fonctions supportées

Package IEEE Standard Logic 1164

```
PACKAGE std_logic_1164 IS
   TYPE std_ulogic IS ( 'U',  -- Uninitialized
                        'X',  -- Forcing  Unknown
                        '0',  -- Forcing  0
                        '1',  -- Forcing  1
                        'Z',  -- High Impedance
                        'W',  -- Weak     Unknown
                        'L',  -- Weak     0
                        'H',  -- Weak     1
                        '-'   -- Don't care  );

   TYPE std_ulogic_vector IS ARRAY ( NATURAL RANGE <> ) OF
    std_ulogic;

   FUNCTION resolved ( s : std_ulogic_vector ) RETURN std_ulogic;

   SUBTYPE std_logic IS resolved std_ulogic;
   TYPE std_logic_vector IS ARRAY ( NATURAL RANGE <>) OF std_logic;


   SUBTYPE X01     IS resolved std_ulogic RANGE 'X' TO '1'; -- ('X','0','1')
```

2.3 Différents modes de description

Assignation Concurrente de signaux

```
S <= a+b;

S <= a and b       when  a < "010"
     else a xor b  when  a < "101"
     else a or b;

with a select
   S <= a and b  when  2 downto 0,
        a xor b  when  3 to 4,
        a or b   when others;
```

Process Combinatoire

```
add : process (a,b,c)
      begin
        if c='1' then  res <= a + b;
        else           res <= a - b;
        end if;
      end process;
```

- Tous les signaux lus doivent être en liste de sensibilité du Process
- Les sorties doivent être affectées quelles que soient les valeurs de la condition

Process Séquentiel

```
process (signal, clock) is begin
  if signal = '1'
      -- actions asynchrone
  elsif -- condition sur l'horloge
      -- actions synchrone
  -- pas de else possible
  end if;
  -- aucune autre action n'est possible
end process;
```

- La liste de sensibilité du Process doit contenir obligatoirement l'horloge, plus un éventuel signal asynchrone (reset)
- On peut remplacer la liste de sensibilité par un wait
- Les sorties doivent être affectées de manière synchrone et asynchrone

Logique combinatoire

Portes logiques

Multiplexage, trois-états

Fonctions arithmétiques

Comparaison

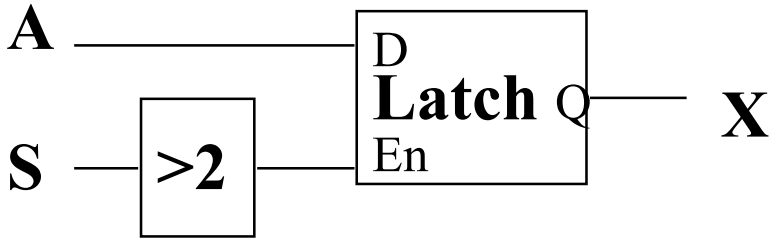
Décodage

Latch ou porte logique

Latch

```
process (A,S)
begin
   if S > 2  then
      X  <= A;
   end if;
end process;
```

=>

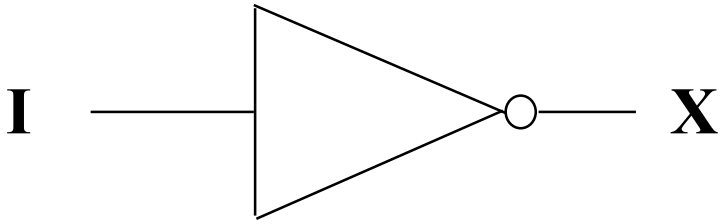


Porte Logique

X <= not I

```
process (I)
begin
   if I = '0'  then
      X  <= '1';
   else
      X  <= '0';
   end if;
end process;
```

=>

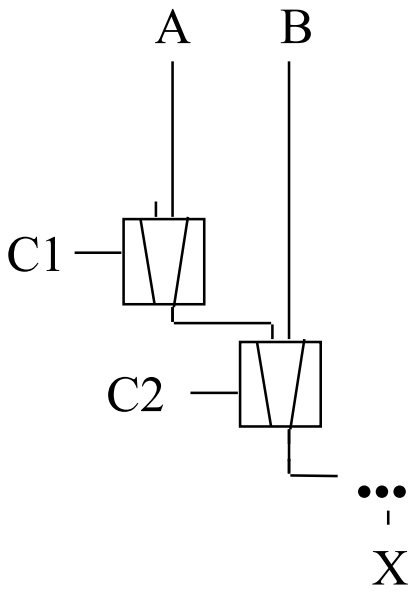


-> spécification complète des équations logiques pouvant aboutir à une table de vérité

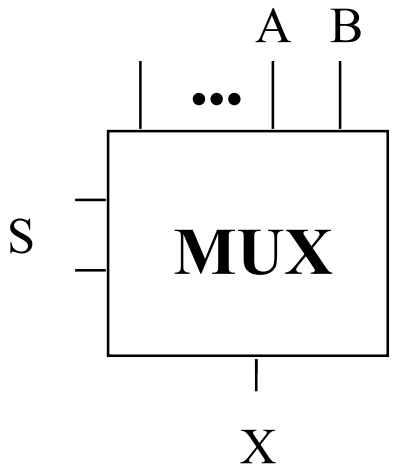
Multiplexage

Avec priorité

```
process (A,B,...C1,...)
begin
   if C1  then
      X  <= A;
   elsif C2  then
      X  <= B;
   •••
   else
   end if;
end process;
```



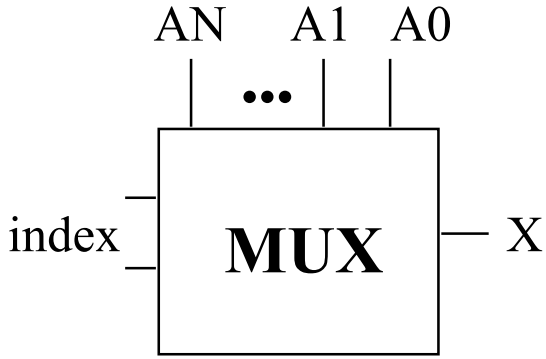
Sans priorité




```
process (A,B,...,S)
begin
   case (S) is
    when C1 =>
      X  <= A;
   when C2 =>
      X  <= B;
   •••
end process;
```

Index dans un tableau

```
X <= A(index);
```

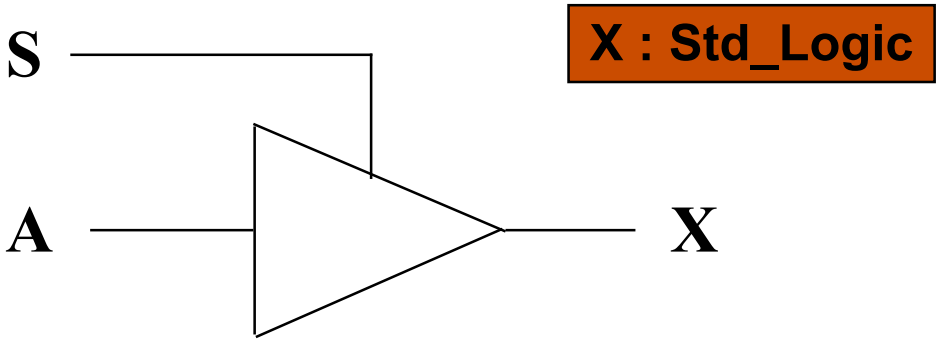


Logique câblée

Tri-State

```
process (A,S)
begin
   if S = '1'  then
      X  <= A;
   else
      X  <= 'Z';
   end if;
end process;
```

=>

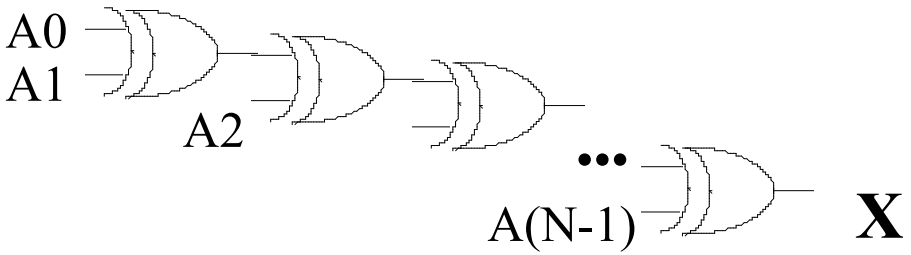


- **Logique cablée entre plusieurs process => Tri-State**
- **Fonctions de résolution ignorées**

For ••• Loop / Generate

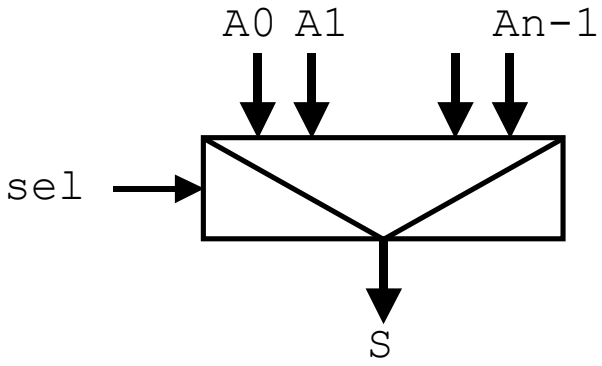
```
parité : process (A)
   variable result : bit;
begin
   result := '0';
   for i in 0 to N-1 loop
      result  := result xor A(i);
   end loop;
   X  <= result;
end process;
```

=>



Exemples

Fonctions de sélection

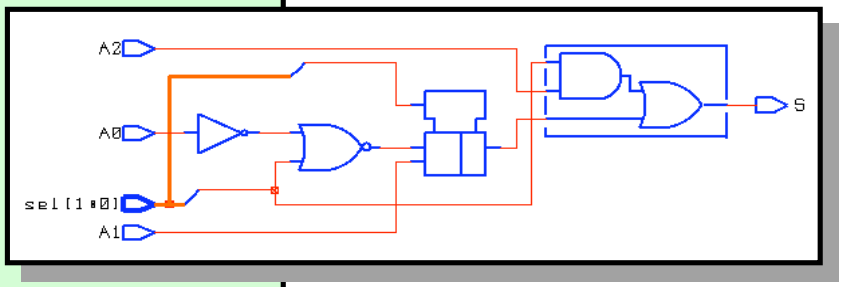


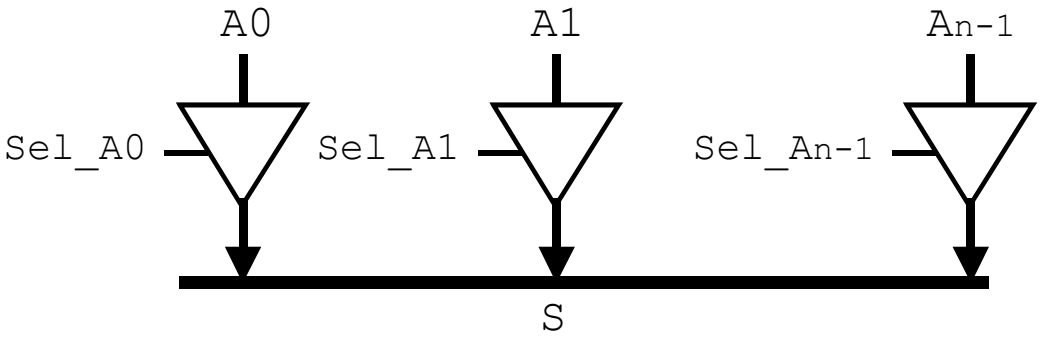
Logique de multiplexage

```
S <= A0 when sel=0 else
     A1 when sel=1 else
     A2 when sel=2 else
     'X';
```

ou

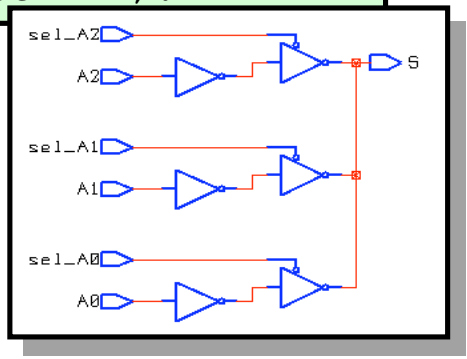
```
process(sel, A0,A1,A2)
begin
  case sel is
    when 0 =>
      S <= A0;
    when 1 =>
      S <= A1;
    when 2 =>
      S <= A2;
  end case;
end process;
```





Logique à trois-états

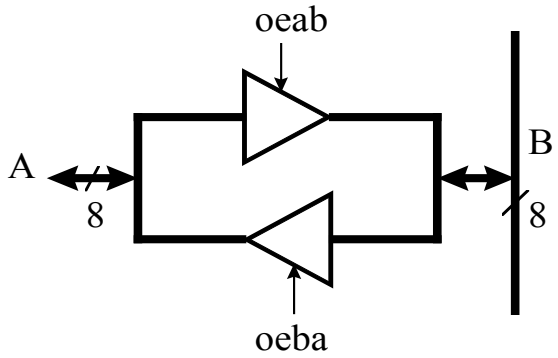
```
signal S, A0, A1,A2 : std_logic ;
signal Sel_A0,Sel_A1,Sel_A2 : std_logic;
….
S <= A0 when sel_A0='1' else 'Z';
S <= A1 when sel_A1='1' else 'Z';
S <= A2 when sel_A2='1' else 'Z';.
```



Exercice

Exercice

Ecrire le fichier VHDL décrivant la fonction Tranceive permettant une connexion bidirectionnelle à un bus 8 bits.




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

entity  transceive is
        port(   A,B : inout std_logic_vector(7 downto 0);
                oeab, oeba : in std_logic);
end entity

architecture transceive of transceive
begin
        B <= A when oeab = '1' else "ZZZZZZZZ";
        A <= B when oeba = '1' else (others => 'Z');
end transceive;
```

Commentaires :

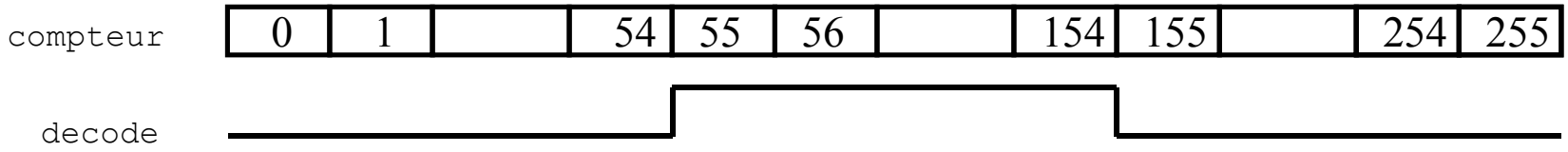
utilisation des types std_logic et std_logic_vector déclarés dans le package std_logic_1164

Exemples

Fonctions de comparaison/décodage

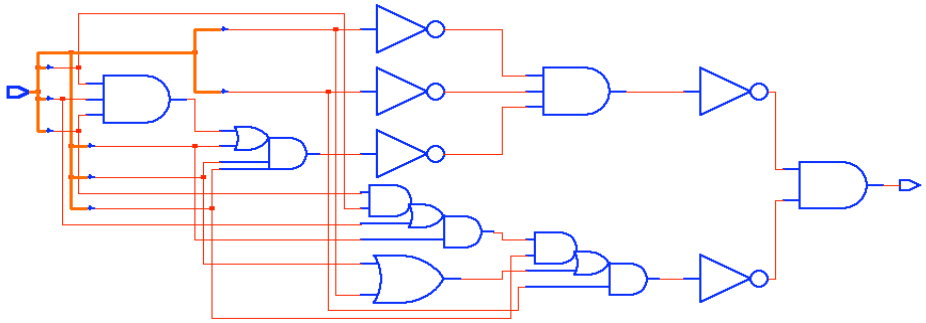
Comparer la valeur de signaux : opérateurs de comparaison,
instructions when, if , case

Exemple : `signal compteur : integer range 0 to 255;`

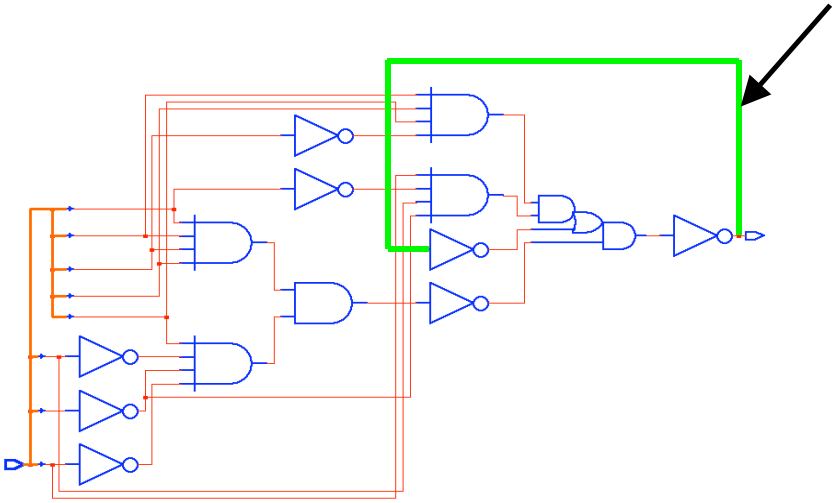


Deux descriptions :

décodage de tous les états : fonction combinatoire



décodage des états de transition : fonction latch



Exercice

Décrivez la fonction précédente dans un process, en utilisant les instructions **if** et **case** ou en flots de données

```
process(compteur)
begin
decode <= ' 0 ';
if (compteur >= 55 and compteur < 155) then
  decode <= ' 1 ';
end if;
end process;
```

```
process(compteur)
begin
if (compteur = 55) then
  decode <= ' 1 ';
elsif compteur = 155) then
  decode <= ' 0 ';
end if;
end process;
```

```
decode <= '1' when (compteur >= 55 and compteur < 155)
          else '0';
```

```
process(compteur)
begin
case compteur is
  when 55 =>
    decode <= ' 1 ';
  when 155 =>
    decode <= ' 0 ';
  when others =>
    null;
end case;
end process;
```

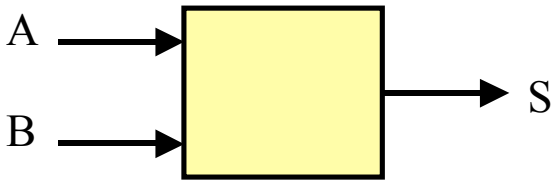
```
decode <= '1' when compteur=55 else
          '0' when compteur=155 else
          decode ;
```

Exemples

Fonctions arithmétiques

Utilisation des opérateurs arithmétiques : **+ - * /**

S <= A **op** B;



avec des signaux/variables de type arithmétique : integer

`integer range 0 to 2^n-1;` → Opérateurs non signés

`integer range -2^n-1 to 2^n-1-1;` → Opérateurs signés

Division

Souvent restreinte à une division par une constante = 2^n → Décalage câblé de *n* bits à droite

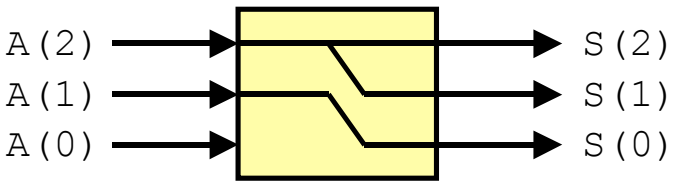
Entiers signés:

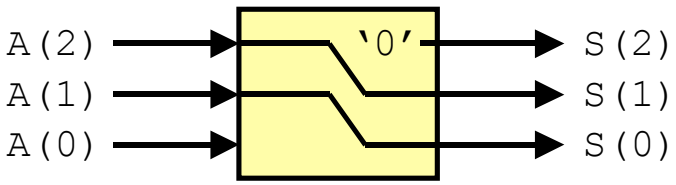
```
signal A, S : integer range -4 to 3;
```

Entiers non signés:

```
signal A, S : integer range 0 to 7 ;
```

```
S <= A/2 ;
```



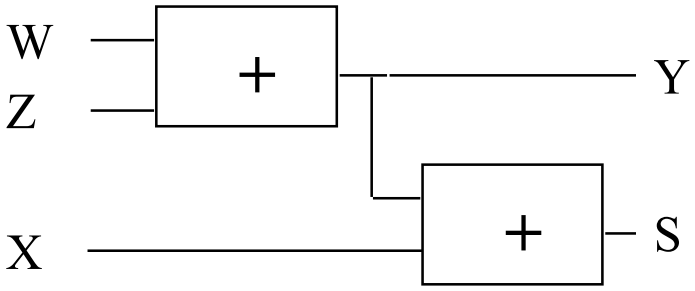


Description VHDL

Assignation Concurrente

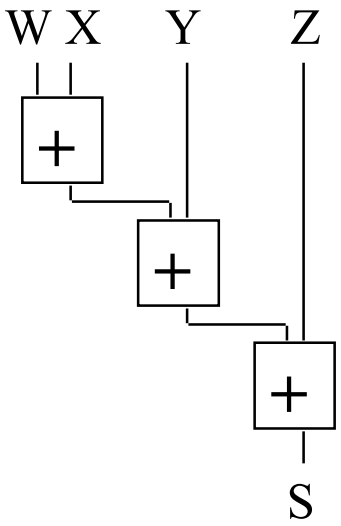
- Ordre des équations sans importance
- Flots de données
- Circuits combinatoires

```
architecture A of E is
begin
   S  <= X + Y;
   Y <= Z + W;
end A;
```



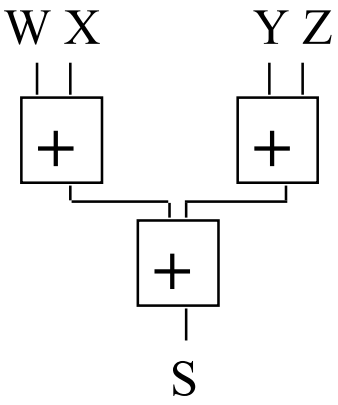
`S <= W + X + Y + Z;`

Evaluation de gauche à droite



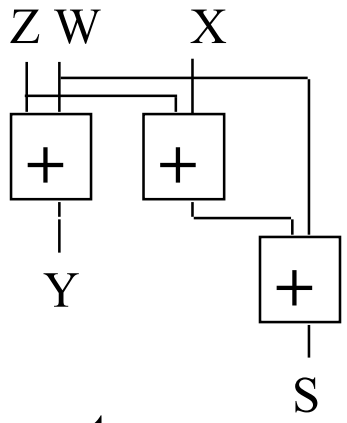
`S <= (W + X) + (Y + Z);`

Effet du parenthésage




```
Y <= Z + W;
S <= X + Z + W;
```

Termes intermédiaires



Rem. Tous les signaux peuvent être des vecteurs

Logique séquentielle

Bascules

Latches

Registres

Machines d'états

Mémoires

Logique séquentielle

Éléments de mémorisation : bascules, registres

L'état d 'un signal, d'une variable doit être mémorisé :

les fonctions logiques utilisées sont des bascules (1 élément binaire) ou des registres (mot de n bits).

En entrée :
- la valeur à mémoriser,
- l 'ordre de mémorisation : un niveau ou un front.

```
if ordre_de_mémorisation vrai  then
    contenu_registre <= valeur_in ;
end if;
```

Niveau *(latch)*

```
process(enable, valeur_in)
    if enable = '1' then
        contenu_registre <= valeur_in;
    end if;
end process;
```

Front *(flip-flop)*

```
process(horl)
    if (horl'event and horl = '1') then
        contenu_registre <= valeur_in;
    end if;
end process;
```

Bascule DFF (D Flip-Flop)

DFF avec front montant

```
PROCESS(clock)  BEGIN
   if clock'event and clock='1' then
      Q  <= D;
   end if;
END PROCESS
```

ou

```
PROCESS BEGIN
   wait until clock'event and  clock='1';
   Q  <= D;
END PROCESS
```

DFF avec reset asynchrone

```
PROCESS(clock, reset)  BEGIN
   if reset = '1' then
      Q  <= '0';
   elsif clock'event and clock='1' then
      Q  <= D;
   end if;
END PROCESS
```

DFF avec front descendant

```
if clock'event and clock='0' then
    Q  <= D;
end if;
```

! *A éviter*

DFF avec reset synchrone

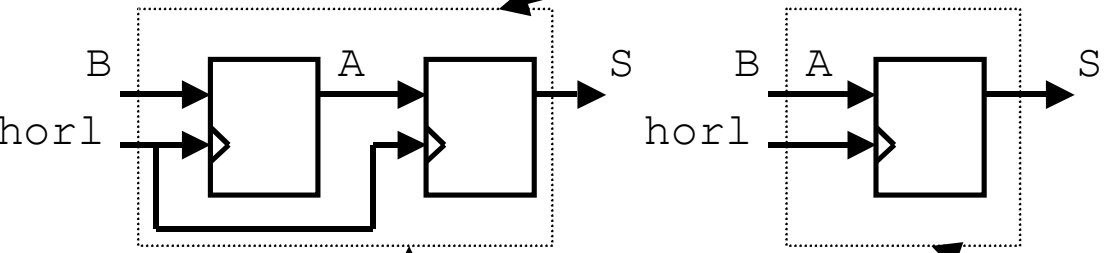
```
wait until clock'event and clock='1';
if reset = '1' then   Q  <= '0';
else                  Q  <= D;
end if;
```

Exemples

Éléments de mémorisation

- S'il s'agit d'un signal, il y a toujours génération d 'une bascule/registre.
- S'il s'agit d'une variable, ça dépend.

Exemples :




```
…….
signal horl, A, B, S : std_logic;
…….
begin
….
process(horl)
begin
  if (horl'event and horl='1') then
    A <= B;
    S <= A;
  end if;
end process;
……..
```

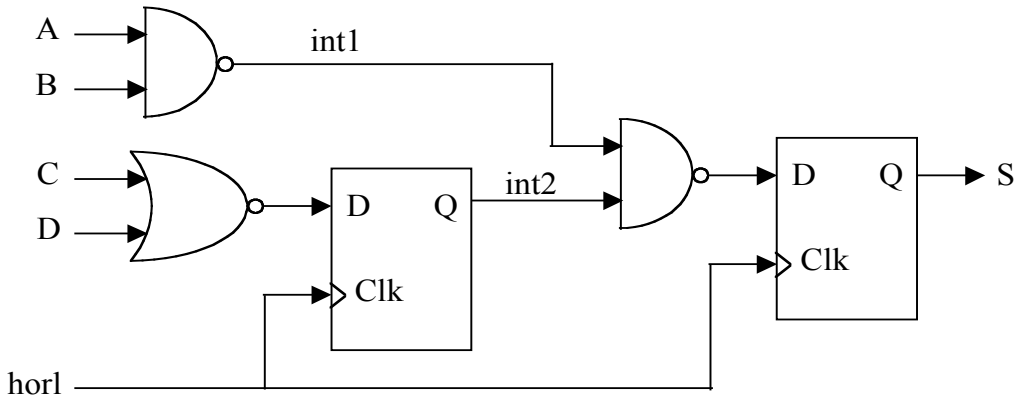
```
…….
signal horl, B, S : std_logic;
…….
begin
….
process(horl)
  variable A : std_logic;
begin
  if (horl'event and horl='1') then
    S <= A;
    A := B;
  end if;
end process;
……..
```

```
…….
signal horl, B, S : std_logic;
…….
begin
….
process(horl)
  variable A : std_logic;
begin
  if (horl'event and horl='1') then
    A := B;
    S <= A;
  end if;
end process;
……..
```

Exercice

Dessiner le schéma logique obtenu par synthèse de la description suivante :

```
process(horl)
    variable int1 : std_logic ;
begin
    if (horl'event and horl='1') then
        int1 := A nand B ;
        int2 <= C nor D ;
        S <= int1 nand int2 ;
    end if ;
end process ;
```



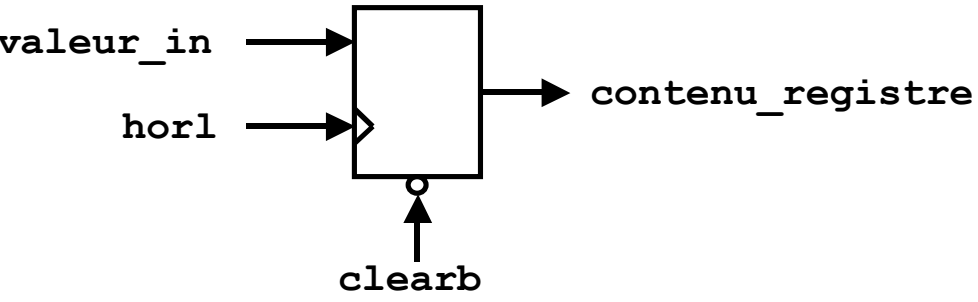
Exemple

Éléments de mémorisation : bascules, registres

Avec initialisation asynchrone ou synchrone

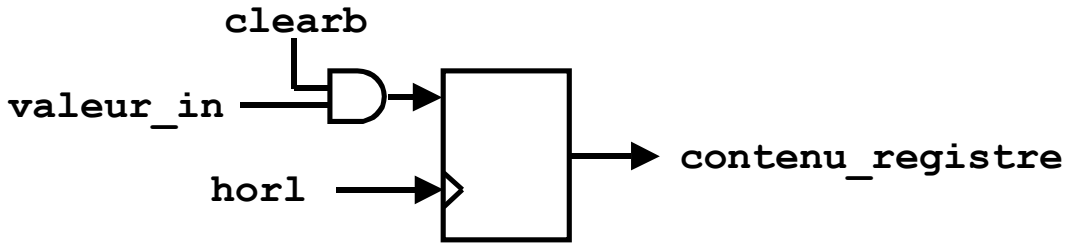
asynchrone

```
process(clearb,horl)
  if clearb = '0' then
    contenu_registre <= 0;
  elsif (horl'event and horl = '1') then
    contenu_registre <= valeur_in;
  end if;
end process;
```



synchrone

```
process(horl)
  if (horl'event and horl = '1') then
    if clearb = '0' then
      contenu_registre <= 0;
    else
      contenu_registre <= valeur_in;
    end if;
  end if;
end process;
```



Règles à respecter

Chargement sur front :

Un seul front
Une seule horloge

```
process(horl)
    if (horl'event and horl = '1') then
        contenu_registre <= valeur_in;
    end if;
    if (horl'event and horl = '0') then
        contenu_registre <= valeur_in;
    end if;
end process;
```

Pas de combinatoire avec l 'horloge

```
process(horl)
    if (horl'event and horl = '1'  and  ena = ' 1 ') then
        contenu_registre <= valeur_in;
    end if;
end process;
```

mais :

```
process(horl)
    if (horl'event and horl = '1') then
        if (ena = ' 1 ') then
            contenu_registre <= valeur_in;
        end if;
    end if;
end process;
```

Fonctions de comptage, de décalage

Compteur binaire n bits : de 0 à 2^n-1

Utilisation de l 'opérateur **+** => le signal/variable doit être de type numérique : conversion

compteur <= compteur +1;

Compteur modulo N :

if (compteur = N-1) then compteur <= 0;

Registre à décalage

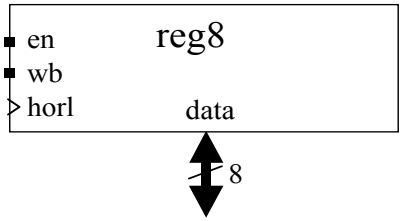
regdec(7 downto 1) <= regdec(6 downto 0);
regdec(0) <= '0';

Exercice

Exercice

Décrire une fonction registre 8 bits à chargement sur front montant de l'horloge **horl**,
Avec un bus de données **data** bidirectionnel

horl : horloge active sur front montant,
en : accès en lecture ou en écriture si **en** = '1' ; **data** = Z si **en** = '0'
wb : accès en écriture si **wb** = '0'
data : bus de données 8 bits bidirectionnel

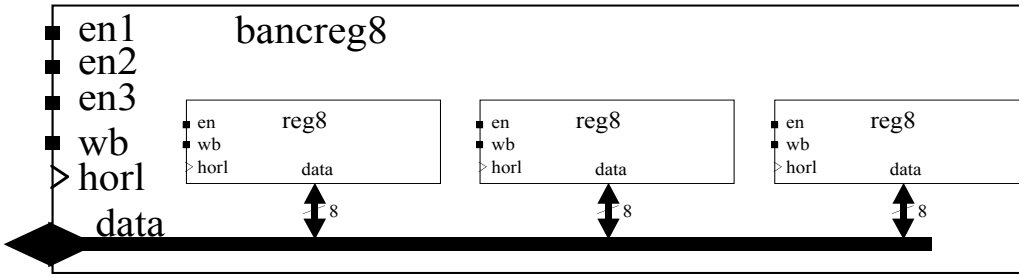



```
library ieee;
use ieee.std_logic_1164.all;

entity reg8 is
        port (horl, en, wb : in std_logic;
                data : inout std_logic_vector(7 downto 0));
end;

architecture A of reg8 is
signal reg :  std_logic_vector(7 downto 0);
begin

process(en,reg,wb)
begin
        if (en = '0') then
                data <= (others => 'Z');
        elsif wb = '0' then
                data <= (others => 'Z');
        else
                data <= reg;
        end if;
end process;
```

```
process(horl)
begin
        if (horl'event and horl='1') then
                if (en = '1' and wb = '0') then
                        reg <= data;
                end if;
        end if;
end process;

end A;
```

Exercice

Description d'un banc de registres

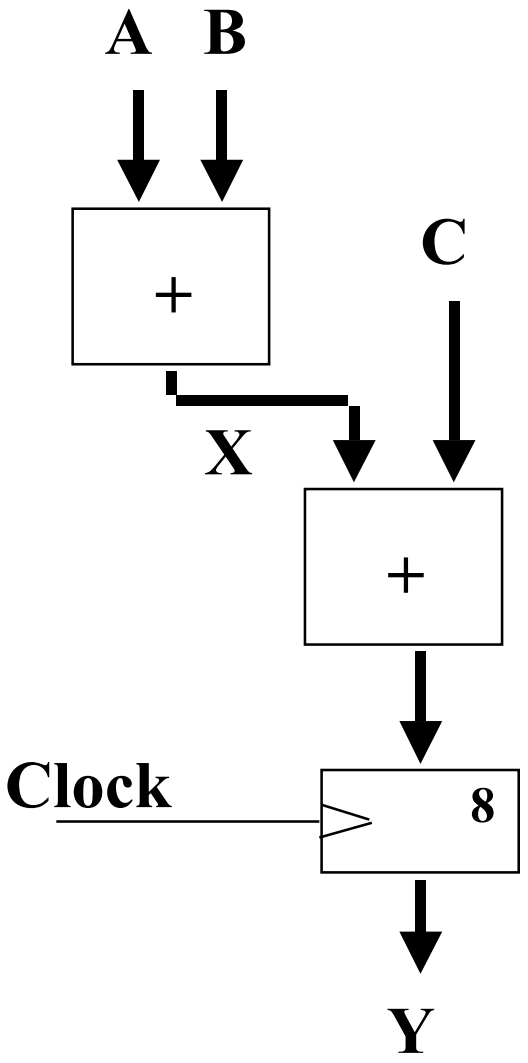



```
library ieee;
use ieee.std_logic_1164.all;

entity bancreg8 is
    port(horl, en1, en2, en3, wb : in std_logic;
         data : inout std_logic_vector(7 downto 0));
end bancreg8;
---------
architecture arch of bancreg8 is

component reg8
        port (horl, en, wb : in std_logic;
                  data : inout std_logic_vector(7 downto 0));
end component;

begin

U1 : reg8 port map (horl,en1,wb,data);
U2 : reg8 port map (horl,en2,wb,data);
U3 : reg8 port map (horl,en3,wb,data);

end bancreg8;
```

Différences entre Variables •••

- **Dans process ou sous-programme**
- **Assignation immédiate**

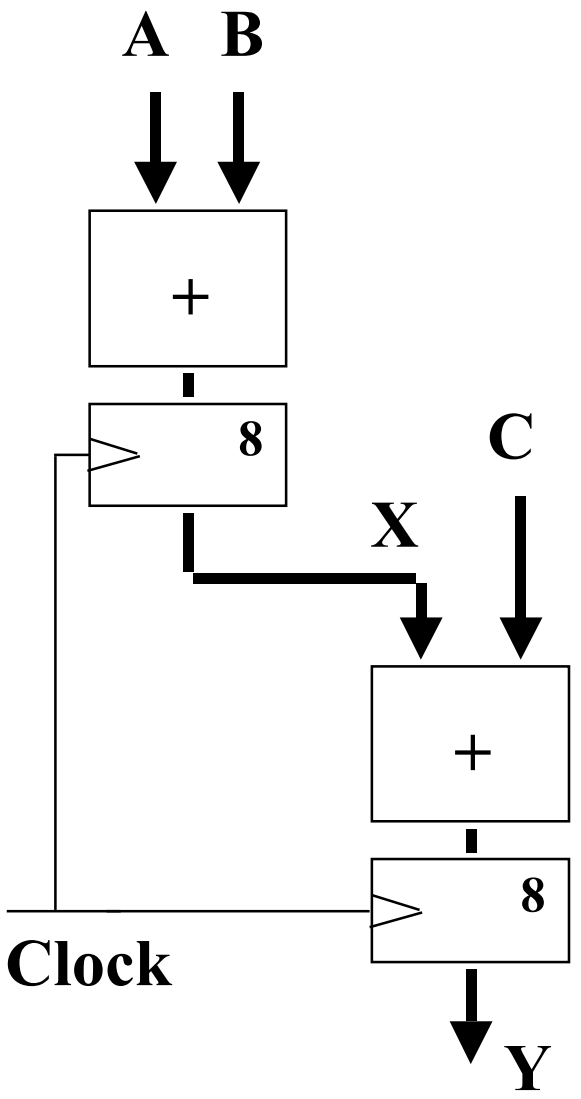
```
entity  add is
   port (A,B,C : in integer range 0 to 255;
           clock   : in bit;
            Y       : out integer range 0 to 255 );
end add;

architecture variable  of add is
process
    variable X : integer range 0 to 255;
begin
    wait until clock='1' and clock'event;
    X  := A + B;
    Y  <= X + C;
end process;
```



••• et Signaux

- **Chaque signal est une sortie de registre**
- **Assignation après _t**

```
entity  add is
   port (A,B,C : in integer range 0 to 255;
         clock  : in bit;
          Y      : out integer range 0 to 255 );
end add;

architecture signal  of add is
    signal X : integer range 0 to 255;
process
begin
    wait until clock='1' and clock'event;
    X  <= A + B;
    Y  <= X + C;
end process;
```



Éléments mémoires

- **Une mémoire est un tableau de mots .**
- **Ses caractéristiques :**
  - sa capacité : le nombre de mots
  - la taille d ’un mot (son type) : nombre de bits

- **Pour lire un mot en mémoire, il faut désigner son emplacement = son adresse.**

- **Exemple : mémoire de 256 octets contient 256 mots de 8 bits.**

Description de la fonction mémoire

A) - déclarer un type tableau :

```
type t_mem is array(0 to nb_mots-1) of std_logic_vector(nb_bits-1 downto 0);
```

B) déclarer un signal/variable

```
signal/variable memoire : t_mem;
```

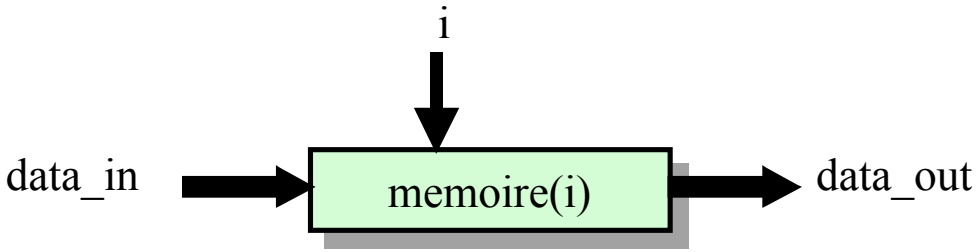
`memoire` représente un tableau de `nb_mots` de `nb_bits`.

`memoire(i)` = i^{ème} mot de la mémoire

C) accéder au mot mémoire d 'adresse `i`

Lecture : `data_out <= memoire(i);`

Écriture : `memoire(i) <= data_in;`

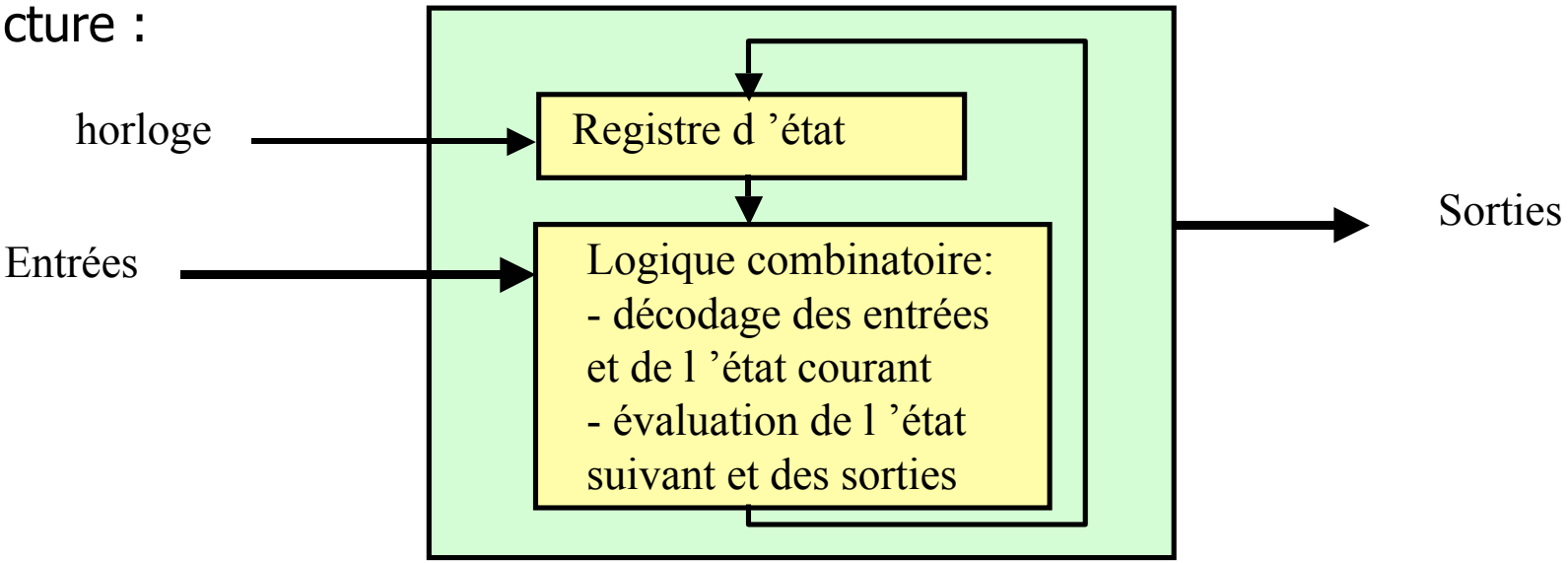


Machine d'états

Fonction logique séquentielle caractérisée par :
- l'énumération d'un ensemble d 'états en nombre fini,
- des conditions de transition entre les états,
- la valeur des sorties dépendant de l'état courant et de l'état des entrées.

Applications :
unité de contrôle, séquenceur,

Structure :



Machine d'états : déclaration et codage des états

Création d 'un type énuméré décrivant la liste des états :

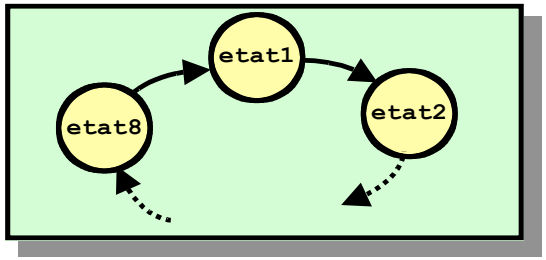
```
type liste_etats is (etat1, etat2, etat3, …..);
signal/variable etat_courant is liste_etats;
```

Déclaration du registre d 'états

Registre d 'état

Codage des *N* états

registre d 'états de *n* bits, tel que $N=2^n$

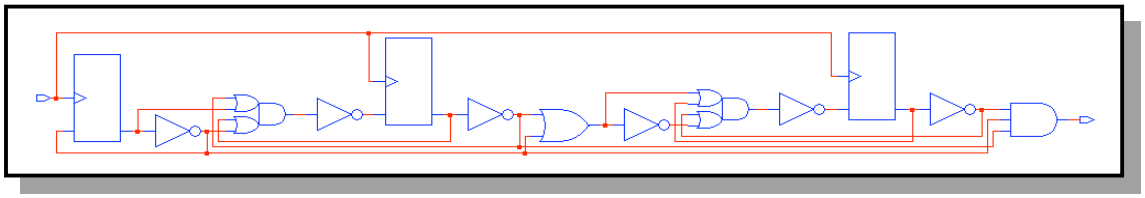


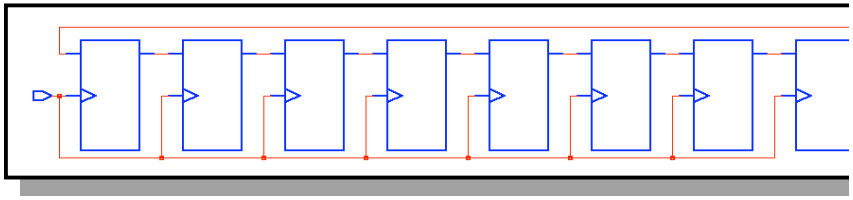
registre d 'états de *N* bits (1 bit par état)

Codage binaire

Exemple : $N = 8$

Codage « one hot »





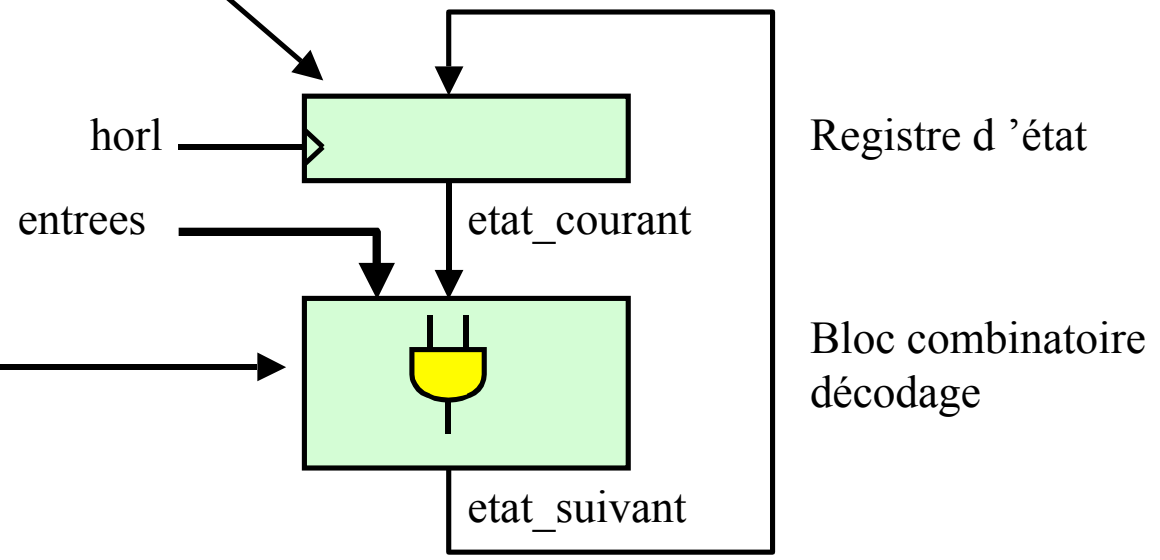
Machine d'états

Machine d'états : Description de la machine d'états

Deux process :
- un process synchrone : registre d'état **etat_courant**
- un process asynchrone : logique combinatoire ,
  etat_suivant = fonction(**etat_courant, entrees**) ;
  utilisation de l'instruction **`case`**

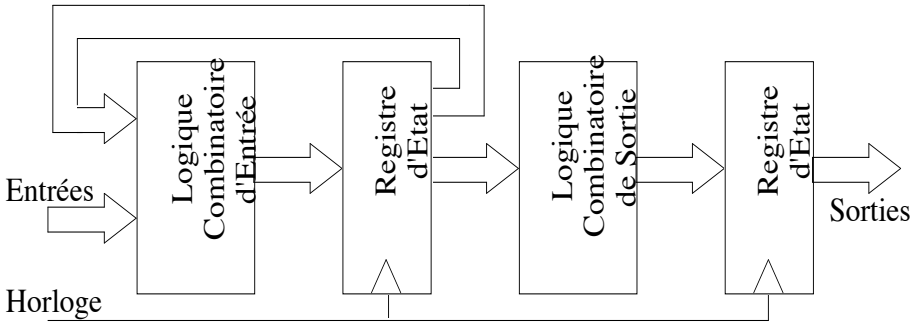
```
process(horl)
begin
  if (horl 'event and horl='1') then
    etat_courant <= etat_suivant;
  end if;
end process;
```

```
process(etat_courant,entrees)
begin
  case etat_courant is
    when etat1 =>
      ….
      etat_suivant <= etat2;

    when etat2 =>
        ….
        etat_suivant <= etat3;
    ….
   end case;
end process;
```



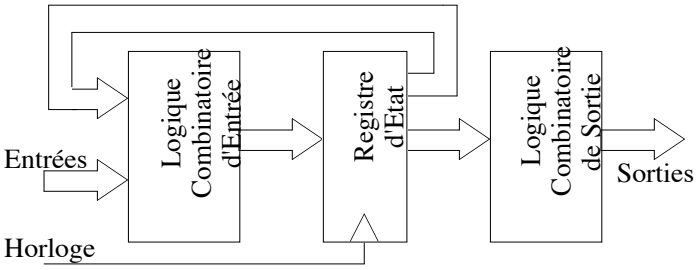
Machine de Moore

Un seul process




```
process
begin
  wait until clk'event and clk='1';
    case EtatCourant  is
       when etat1 =>
          Sortie <= ...;
          if Entree = ...
             then  EtatCourant <= etat2;
             else  EtatCourant <= etat1;
          end if; •••
    end case;
end process;
```

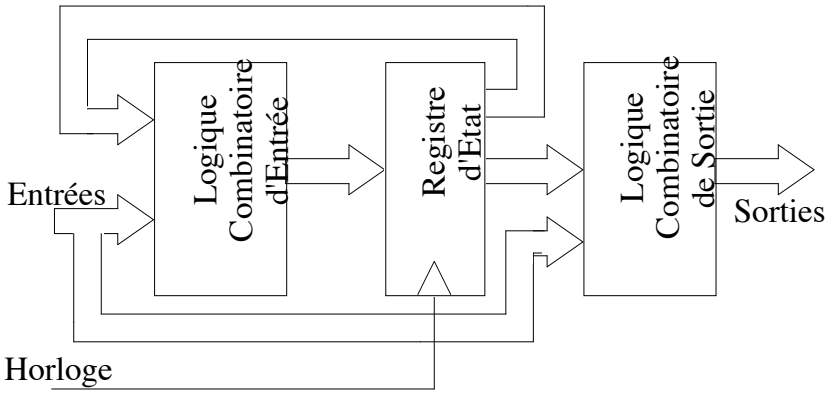
Deux process




```
synchrone : process
begin
  wait until clk'event and clk='1';
     EtatCourant <= EtatSuivant;
end process;
```

```
combinatoire : process (Entree, EtatCourant)
begin
   case EtatCourant  is
      when etat1 =>
         Sortie <= ...;
         if Entree = ...
            then  EtatSuivant <= etat2;
            else  EtatSuivant <= etat1;
         end if;  •••
   end case;
end process;
```

Machine de Mealy



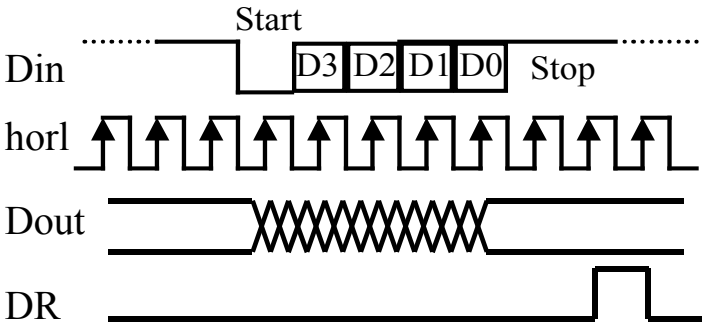

```
synchrone : process
begin
  if reset = '0' then
      EtatCourant <= EtatInitial;
  elsif clk'event and clk='1';
      EtatCourant <= EtatSuivant;
end process synchrone;
```

```
combinatoire : process (Entree, EtatCourant)
begin
   Sortie <= ...;
   case EtatCourant  is
      when etat1 =>
         if Entree = ...  then
            Sortie <= ...;
            EtatSuivant <= etat2;
         else
            Sortie <= ...;
            EtatSuivant <= etat1;
         end if;   •••
   end case;
end process combinatoire;
```

Deux process

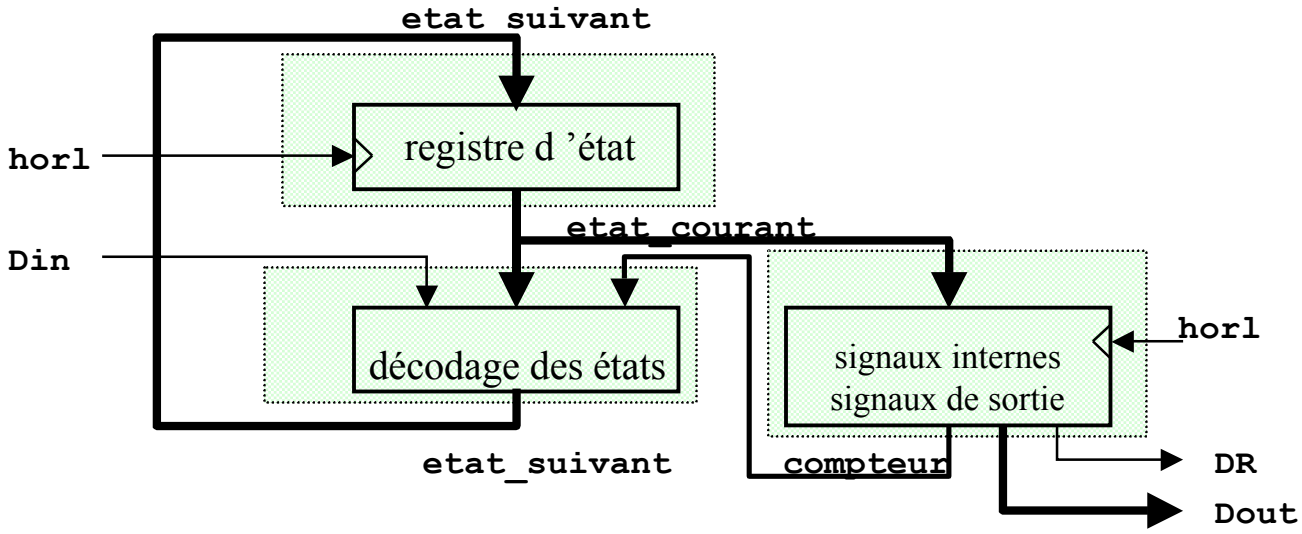
Exercice

Décrire la fonction “ récepteur série asynchrone ” de mots binaires de 4 bits



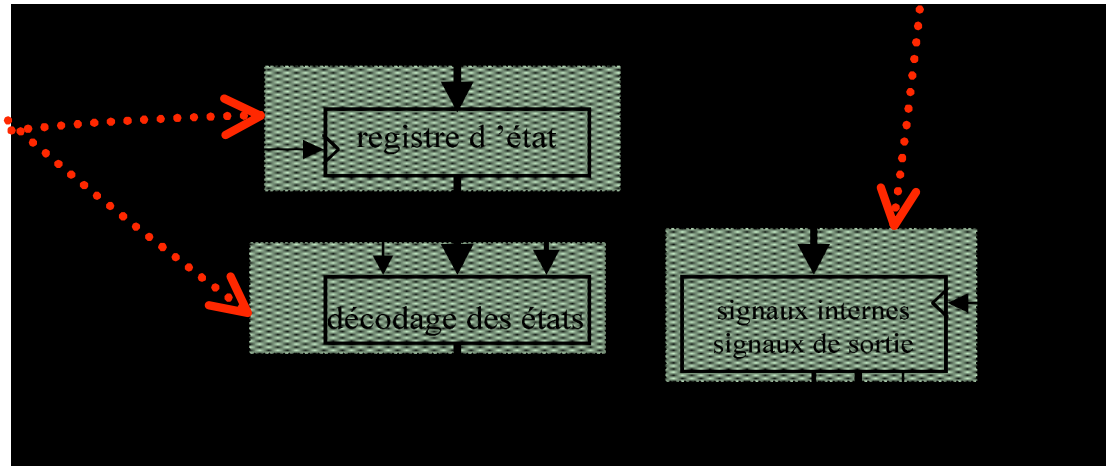
Le message binaire débute par un start bit (Din=’0’) suivi des 4 bits d’information et se termine par 2 stop bits (Din=’1’).
Le circuit reconstitue le mot de 4 bits et le présentera après réception sur le bus de sortie Dout en activant
le signal DR (Data Ready).

Définir les différentes phases du traitement , attente du start bit, réception des 4 bits, …..
, et décrire sous forme de machine d’états le contrôleur.



```
library IEEE;
use IEEE.STD_LOGIC_1164.all;

entity rec1 is
  port(reset,horl, Din: in STD_LOGIC;
  Dout : out STD_LOGIC_VECTOR(3 downto 0);
  DR : out STD_LOGIC);
end rec1;

architecture A of rec1 is
  type etats is (phase_0, phase_1, phase_2, phase_3) ;
  signal etat_courant, etat_suivant : etats;
  signal compteur : integer range 0 to 3;
  signal Dout_int : STD_LOGIC_VECTOR(3 downto 0);
begin
-------------------------------------------------------
process(etat_courant,Din, compteur)
begin
  case etat_courant is
    when phase_0 =>    --  attente start bit
      etat_suivant <= phase_0;
      if (Din= '0') then
        etat_suivant <= phase_1;
      end if;
    when phase_1 => --reception des 4 bits d'information
      etat_suivant <= phase_1;
      if (compteur = 3) then
        etat_suivant <= phase_2;
      end if;
    when phase_2 =>    -- 1er stop bit
      etat_suivant <= phase_3;
    when phase_3 =>     -- 2eme stop bit
      etat_suivant <= phase_0;
  end case;
end process;
-----------------------------------------
process(horl,reset)
begin
  if (reset='1') then
    etat_courant <= phase_0;
  elsif (horl'event and horl='1') then
    etat_courant <= etat_suivant;
  end if;
end process;
-------------------------------------------------------
```

```
process(horl,reset)
begin
  if (reset='1') then
    compteur <= 0;
    DR <= '0';
  elsif (horl'event and horl='1') then
    case etat_courant is
      when phase_1 => --reception des 4 bits d'information
        compteur <= compteur + 1;
        DR <= '0';
        Dout_int(3 downto 1) <= Dout_int(2 downto 0);
        Dout_int(0) <= Din;
      when phase_2 =>
        compteur <= 0;
        DR <= '1';
      when others =>
        compteur <= 0;
        DR <= '0';
    end case;
  end if;
end process;
-------------------------------------------------------

Dout <= Dout_int;

end A;
```



V Synthèse logique

1. Synthèse logique à partir de VHDL

- Principe et Flot de conception
- VHDL pour la synthèse
- Styles de description
- Logique combinatoire
- Logique séquentielle

2. Mécanismes de synthèse logique

- Optimisations
- Structuring / Flattening / Mapping

3. Conclusions générales

Mécanismes de Synthèse Logique

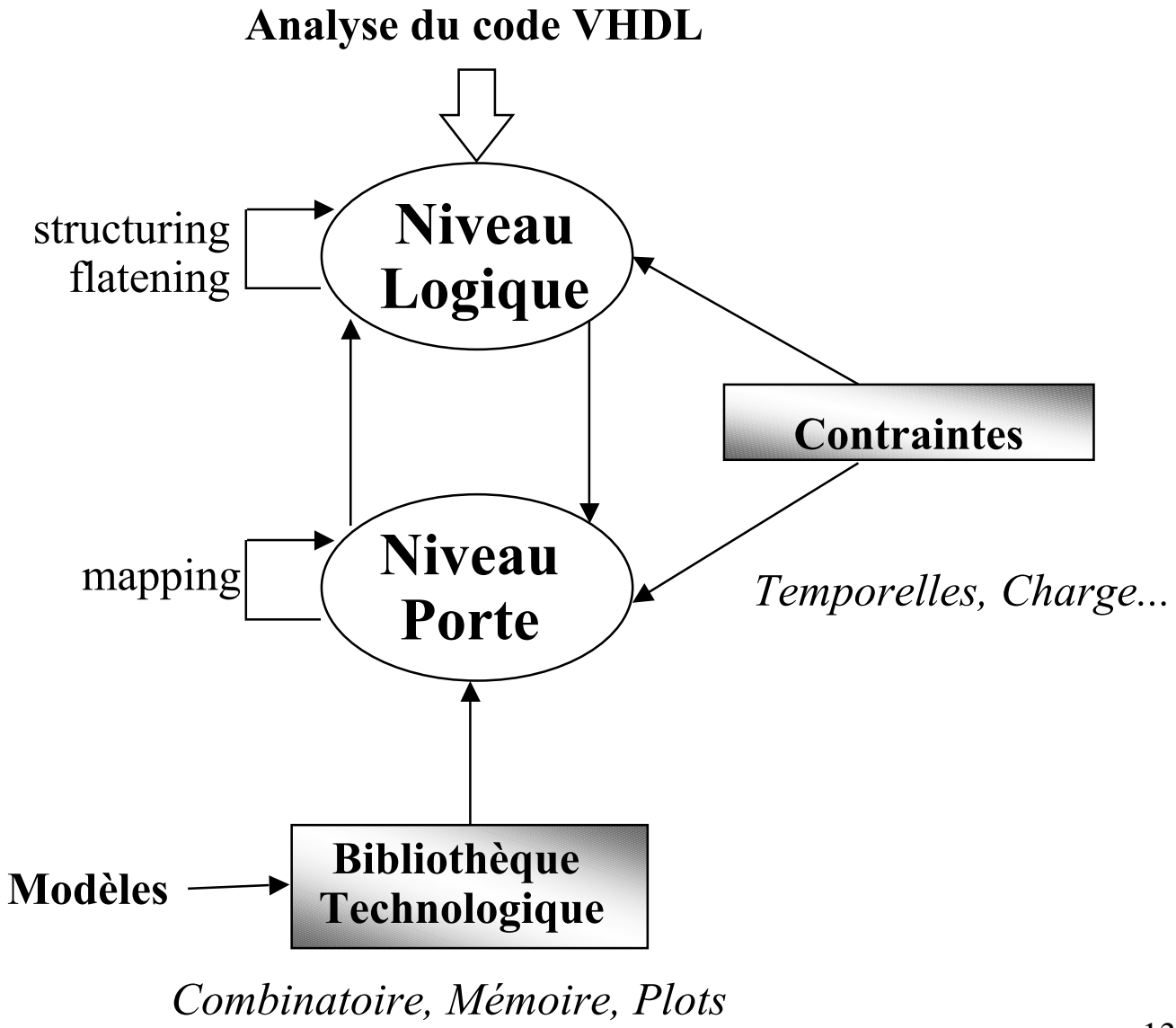
Synthèse logique = Translation + Optimsations

Description VHDL -> Translation

- Le code VHDL est translaté vers une représentation logique générique.

Contraintes -> Optimisations

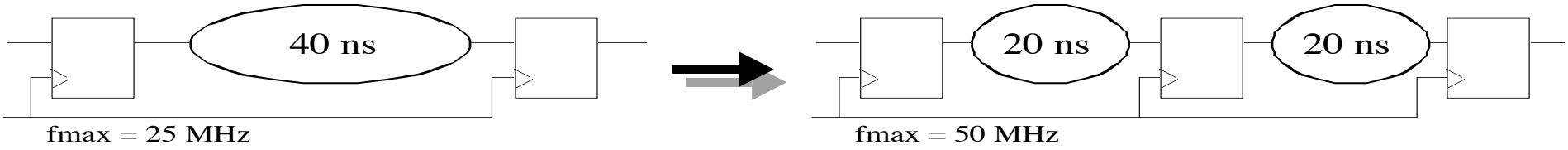
- La logique générique est mapée sur les portes logiques de la bibliothèque



Optimisations •••

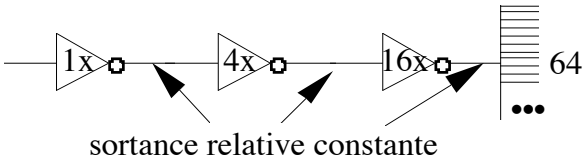
- **Structuring (orienté surface)**
  - Minimisation, factorisation booléenne
- **Flattening (orienté vitesse)**
  - Equations de type PLD
- **Mapping**
  - Spécification des contraintes de fonctionnement et d'environnement
  - Sélection de composants dans la bibliothèque
  - Génération de plusieurs solutions

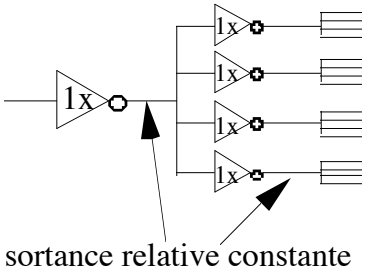
- **Optimisation spécifiques aux machines d'états**
- **Pipeline**
  - Equilibrage de logique combinatoire entre registres



••• Optimisations •••

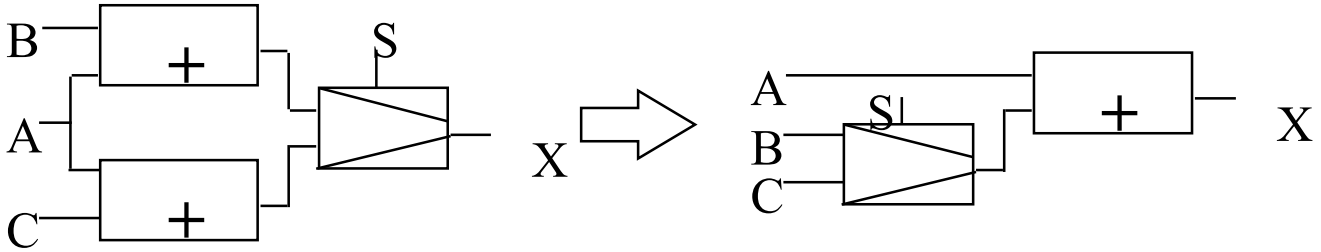
- **Génération d'arbres d'amplification**





- **Partage de ressources**
  - Diminuer le nombre de ressources arithmétiques sans ajouter de structures de contrôle

```
process (A,B,C)
begin
   if S = '1'  then   X  <= A + B;
   else               X <= A + C;
   end if;
end process;
```



Optimisations •••

Structuring (orienté surface)

Le circuit est représenté avec des équations booléennes
Les facteurs communs sont construits et évalués en taille
Les facteurs qui réduisent l'architecture deviennent des variables intermédiaires
Le nombre d'étages logiques des chemins critiques temporels est minimisé
Les termes sont partagés

14 AND
2 NOT
4 OR

$x = a.b.c + a.b.!c + a.d.e$
$y = a.b.c.d + a.b.c.!d$
$z = a.b + c.d$

Optimisation

7 AND
2 OR

$x = a.b + a.d.e$
$y = a.b.c$
$z = a.b + c.d$

Structuring

Mais Temps de traversée augmenté !

$t = a.b$
$u = c.d$
$x = t + a.d.e$
$y = t.c$
$z = t + u$

5 AND
2 OR

Flattening (orienté vitesse)

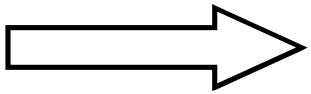
Les équations sont développées pour obtenir des équations PAL
Les facteurs communs sont distribués
Optimisation temporelle
Elimine toute logique structurelle
Nombre d'entrées pas trop important

Mais Temps de traversée diminué !

4 AND
2 OR
1 NOT

$t = d + e$
$x = a.b.t$
$y = a + t$
$z = b.c.!t$

Flattening



$x = a.b.d + a.b.e$
$y = a + d + e$
$z = b.c.!d.!e$

7 AND
3 OR
2 NOT

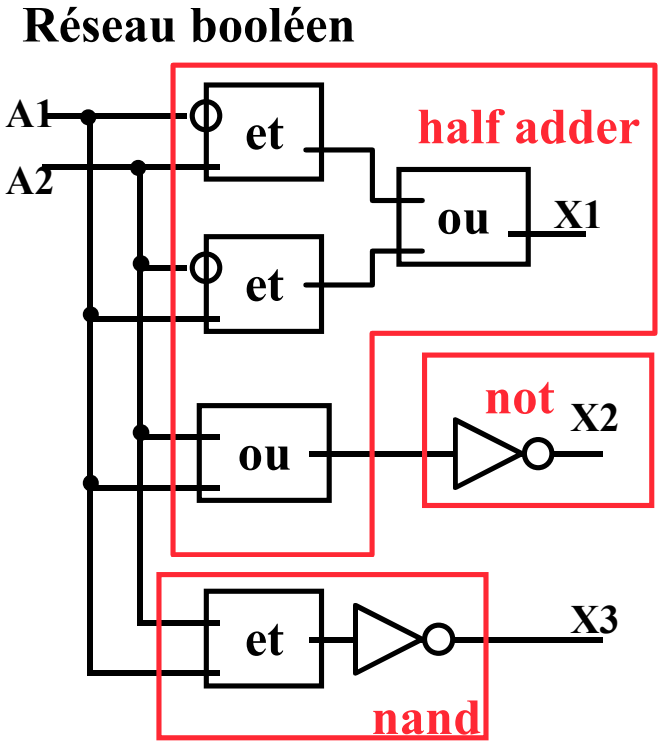
••• Optimisations •••

Mapping : Sélection de composants dans une librairie technologique

A partir d'une librairie de cellules, d'un jeu de contraintes et d'une description logique indépendante de la technologie

-> Passer de la description logique à une netlist de cellules

-> Dépendant de la technologie

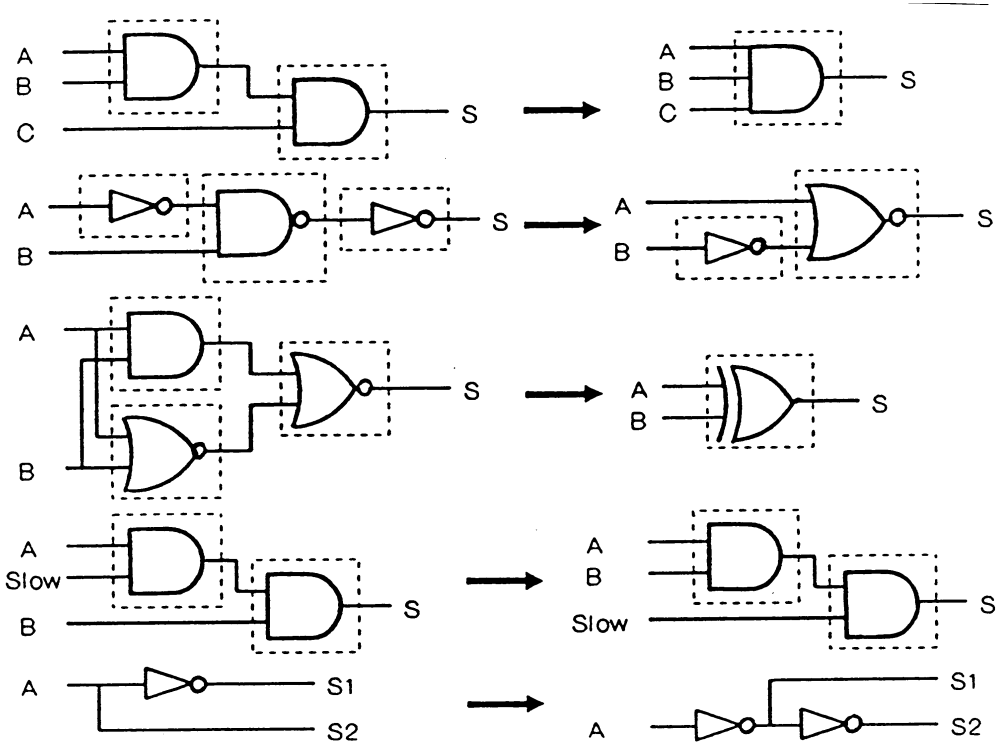


- **On cherche des sous graphes implémentés par des cellules de la librairie couvrant tout le réseau booléen**
- **On évalue les temps / coût**
- **On retient la solution optimale vis à vis des contraintes**

Librairie de composants

cellule	temps	coût	fonction
nand	2 ns	3	!(AB)
half adder	4 ns	14	!A.B + A.!B
••••			

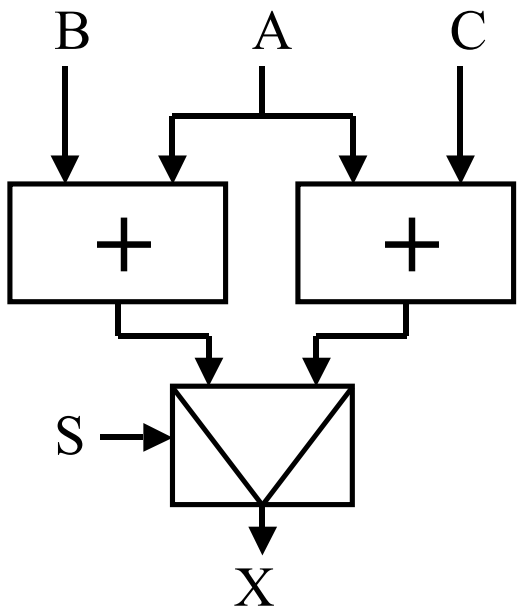
- **Règles de correspondance**



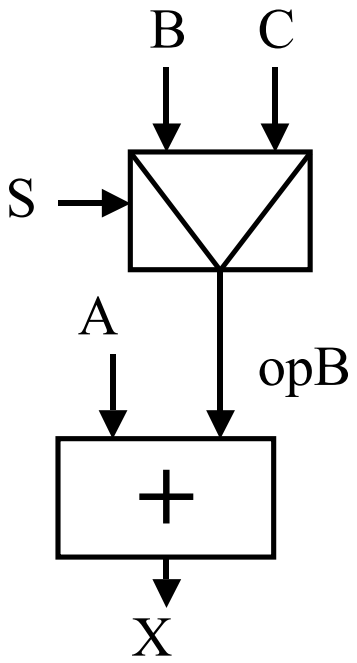
Partage de ressources

Partage de ressources

```
process (A,B,C,S)
begin
    if (S = ' 1 ') then
        X <= A + B ;
    else
        X <= A + C ;
    end if;
end process;
```




```
process (A,B,C,S)
    variable opB : integer;
begin
    if (S = ' 1 ') then
        opB :=  B ;
    else
        opB := C ;
    end if;
    X <=  A + opB;
end process;
```

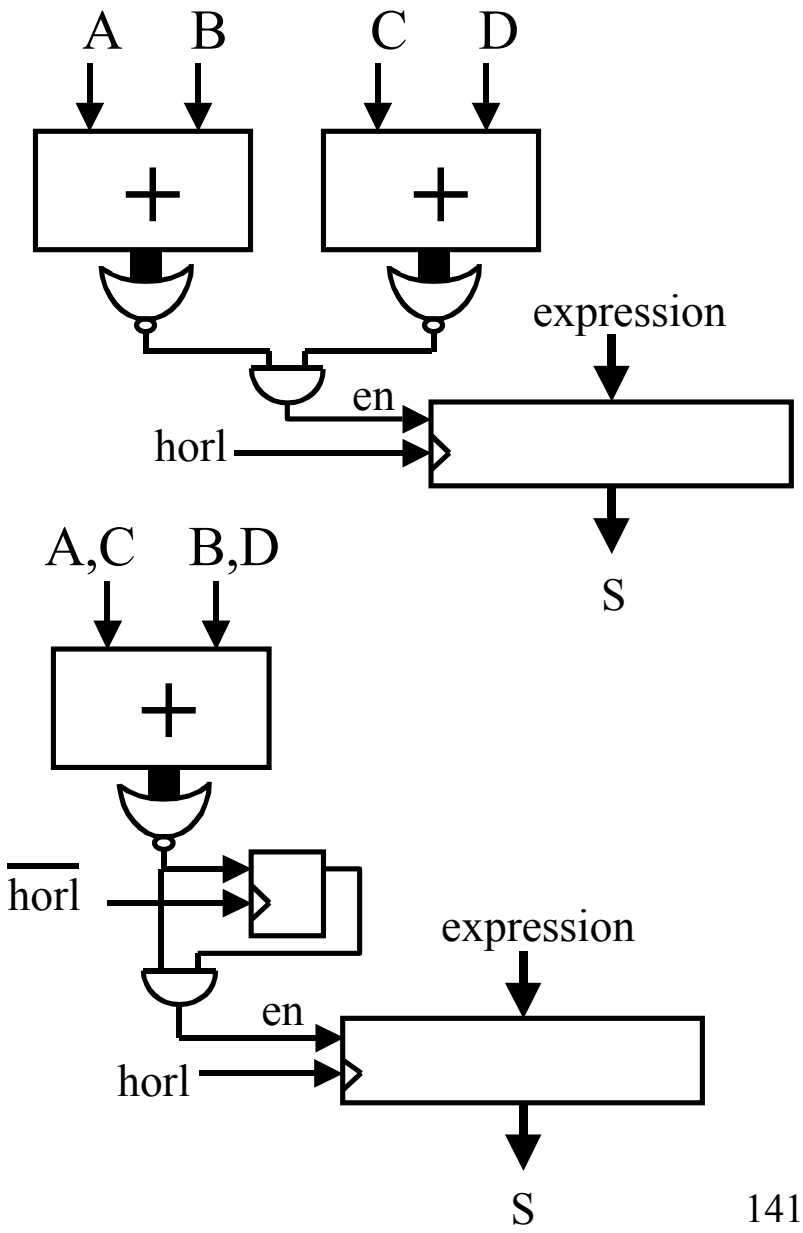


Structure parallèle-série

Structure parallèle ⟷ Structure série

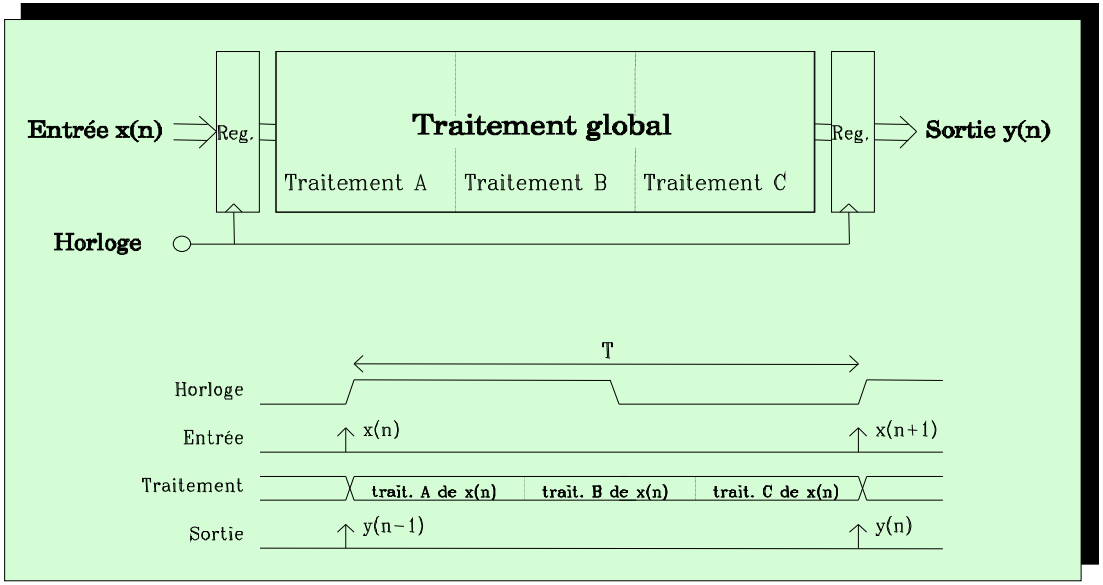
```
if (horl 'event and horl = ' 1 ')then
  if (A + B =0) and (C + D = 0)then
    S <= expression ;
  end if;
end if;
```

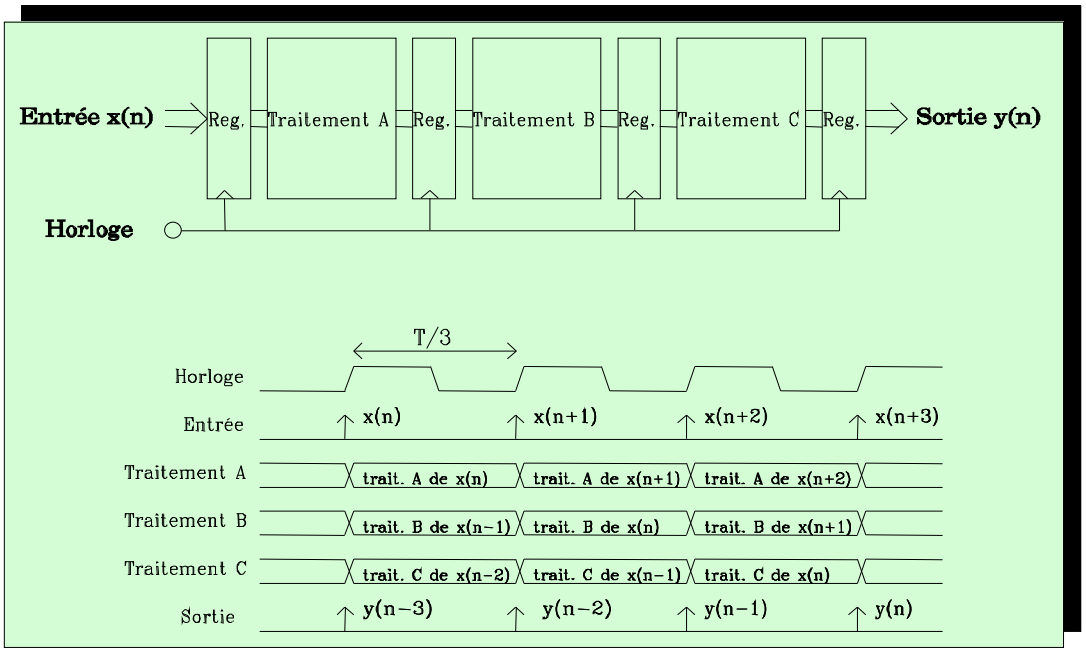
```
if (horl 'event and horl = ' 0 ') then
  if (opa + opb =0) then
    test1 <= '1 ' ;
  else
    test1 <= '0 ';
  end if;
end if;
....
....
 if (horl 'event and horl = ' 1 ') then
  if (opa + opb =0) and (test1 = ' 1 ')then
    S <= expression ;
  end if;
end if;
```



Pipeline

Structure pipeline

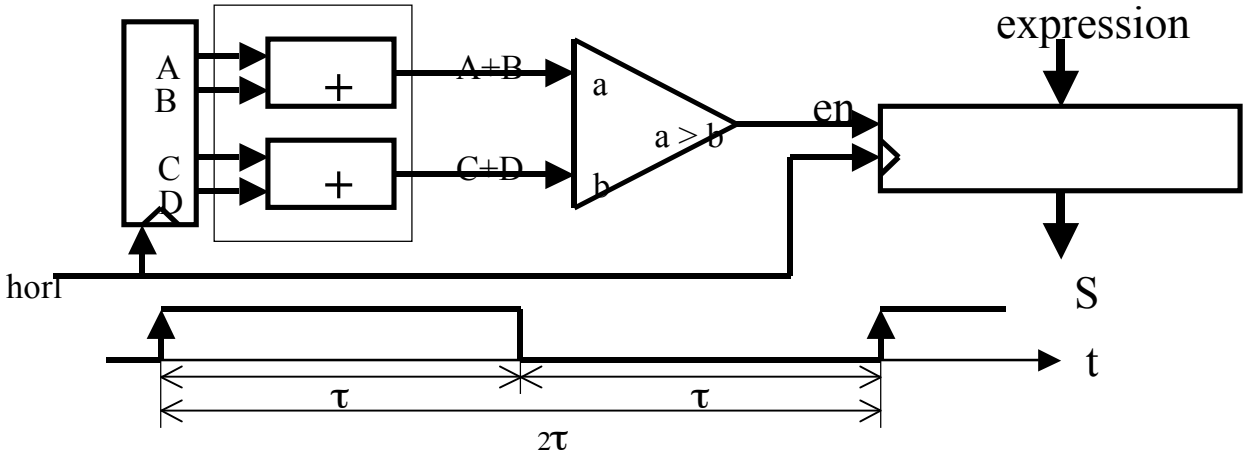




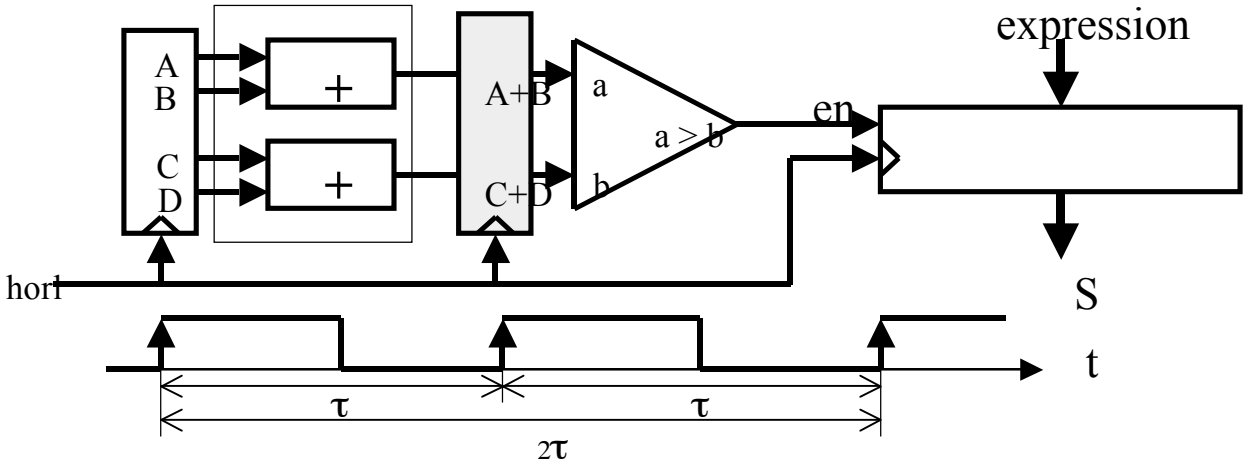
Pipeline

Structure pipeline

```
if (horl 'event and horl = ' 1 ') then
    if (A + B) > (C + D) then
        S <= expression ;
    end if;
end if;
```




```
signal AplusB, CplusD : integer;
…
…
if (horl 'event and horl = ' 1 ') then
    AplusB <= A + B;
    CplusD <= C + D;
    if AplusB > CplusD then
        S <= expression ;
    end if;
end if;
```



3. Conclusions générales

- De plus en plus d'applications justifient le développement d'un circuit spécifique.
- Un ASIC permet d'explorer beaucoup plus de solutions architecturales qu'un processeur généralisé.
- Avantages nombreux, inconvénients réduits grâce aux outils.
- Méthodologie : Spécification détaillée, découpe fonctionnelle, conception descendante.
- Apport des outils d'aide, de synthèse et de CAO dans le flot de conception d'un circuit.
- Apport des langages de description (HDL).
- Comparaison FPGA et ASIC masqués.
- Prototypage rapide par la conception d'un FPGA dédié

Synthèse : Evidences

- Techniques de conception entrées dans les mœurs
- Tous (ou presque) les circuits numériques sont faits par les techniques de synthèse logique à partir d'un HDL
- Plus de temps est passé sur la phase de modélisation, donc plus de compréhension du système complet, travail sur l'architecture favorisé
- Rapprochement avec les équipes systèmes, élévation du niveau d'abstraction des descriptions de circuits
- Très grande réactivité en cas d'une évolution du cahier des charges en dernière minute…
- Fiabilité, sûreté de la production
- Flexibilité du flot (interfaces bien identifiées)

Synthèse : Evidences

- **Ré-utilisation => bibliothèques d'éléments génériques**
- **Migration technologique (ex: 0.5µ -> 0.25µ -> 0.18µ)**
- **Introduction des techniques de test in-situ**
- **Double compétence: informatique-électronique**
  - formation

Mais …

- **Il faut garder un œil critique «expert électronicien»**
- **Besoin d'outils pour gérer les gros projets**
  - cohérence de l'ensemble, mis à jour automatique, …
- **Evolution des outils**
  - garantir la stabilité de la chaîne de conception

Evolutions

Le monde continue de bouger

- **Langage VHDL**
  - extension analogique (bientôt en option)
  - extension objet (à l'étude)
  - meilleure admissibilité du langage VHDL vis-à-vis de la synthèse
    - > paramètres génériques
    - > instructions "generate" (macro-génération d'instructions)
    - > types composites (record, structures complexes)
- **Synthèse**
  - prise en compte du *floorplan* (plan de masse)
  - synthèse architecturale (existe au niveau commercial)
    - => partitionnement et allocation des ressources dans le temps
  - cible d'outils de plus haut niveau (générateurs, compilateurs, …)

Evolutions

- **Routage**
  - prise en compte des contraintes temporelles (placement)
  - génération des arbres d'horloge
  - prise en compte des contraintes temporelles
    => placement inter et intra bloc
  - génération des arbres d'horloge
  - lien avec la synthèse (pour une nouvelle optimisation)
- **Technologies cibles**
  - prépondérance des circuits programmables (FPGA, ...)
  - disponibilité des modélisations temporisées des éléments de base
    => VITAL
  - submicronique: quelles conséquences ?
    > modèle linéaire de timing plus suffisant
    => modèle tabulé: tpd = f(capacité, slope)
    > prise en compte des résistances (réseau d'horloges, ...)
    > flot de conception modifié: prise en compte du placement des cellules le plus tôt possible pour la synthèse (interconnexions)

Evolutions

- **Flot de conception**
  - FPGA : les structures se complexifient

    => le flot de conception se rapproche de celui des Asics
    - arbre d'horloge au niveau du placement-routage
    - détermination de régions (équivalent blocs Asic)
  - Quel flot ?

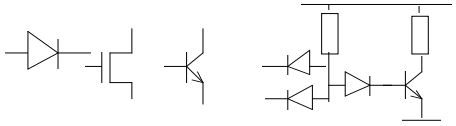
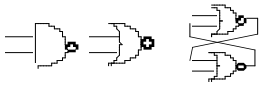
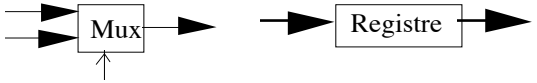
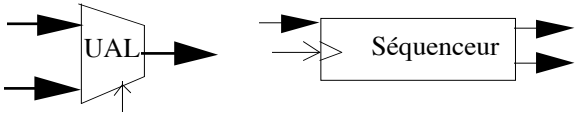
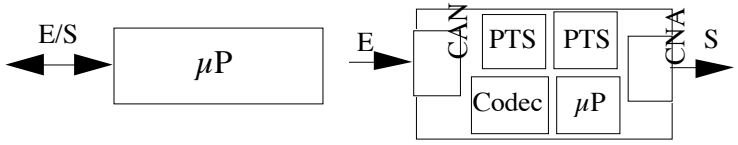
    > pré-placement des blocs (plan de masse)

    > synthèse par bloc avec pré-placement estimées des cellules

    > placement des cellules, puis routage final

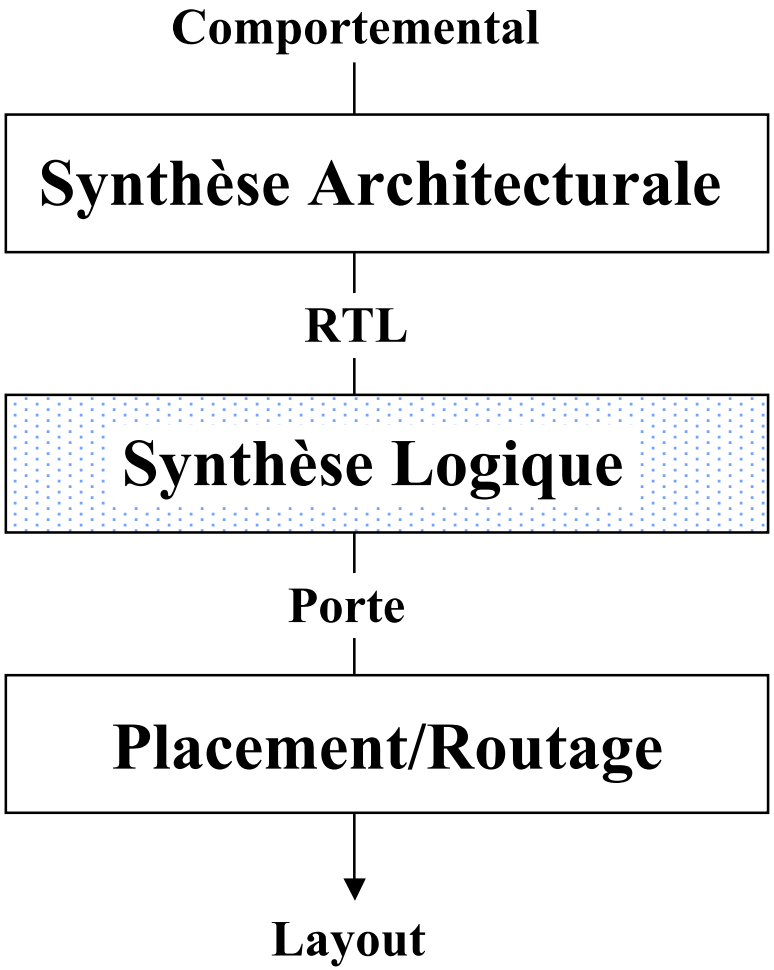
    > placement des blocs, puis routage final

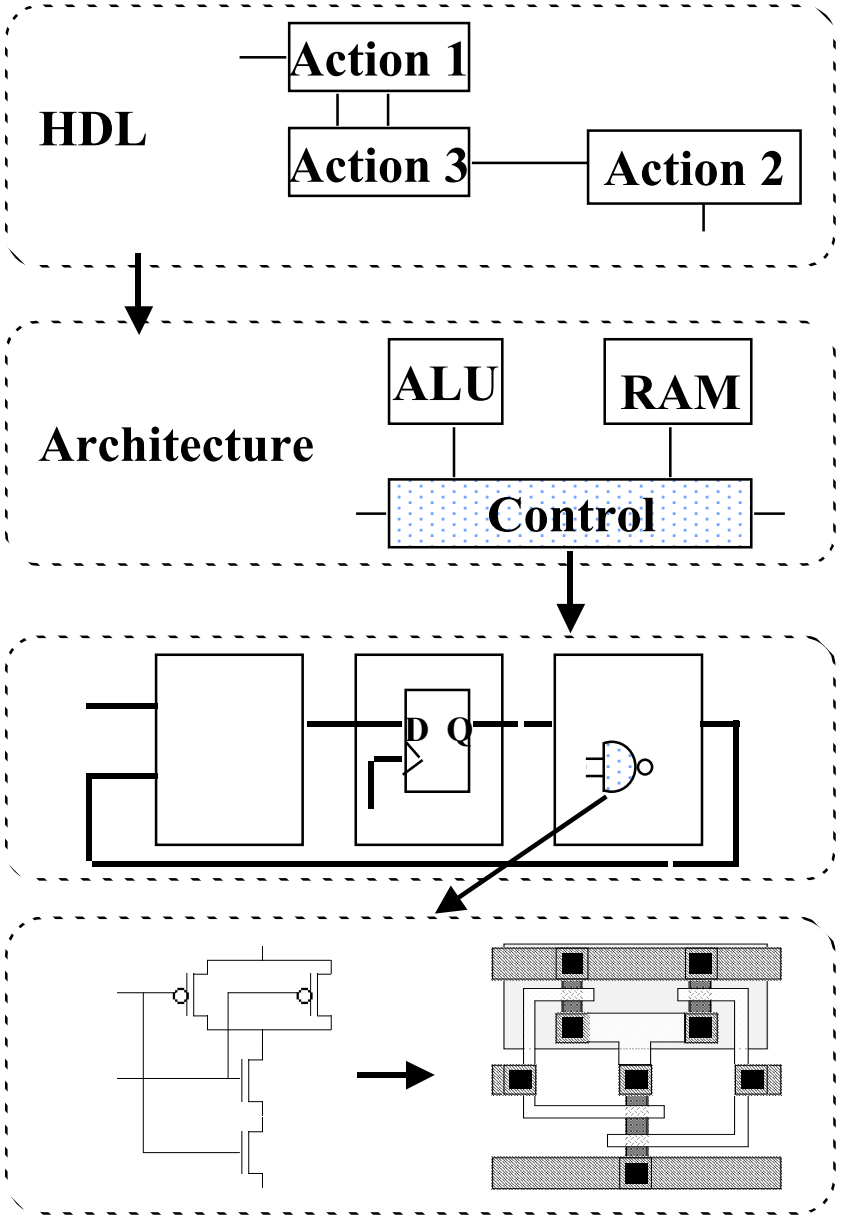
Niveaux technologiques

Type	Constituants de base	Conception
Composants discrets		**Conception électrique** Optimisation des caractéristiques électriques, structurelles,…
SSI		**Conception logique** Optimisation des équations logiques
MSI		**Conception numérique** Optimisation des traitements
LSI		**Conception architecturale** Choix des fonctionnalités
VLSI		**Conception mixte** Optimisation des implantations matérielles et logicielles

ULSI : “System on a single chip”

Conception des ASICs





Validation

La description comportementale demeure la référence.

- mise au point du modèle => définition de l'environnement
- validation après synthèse par simulation
  (délais estimés par l'outil de synthèse)
- validation après routage par simulation
  (délais calculés par l'outil de routage)

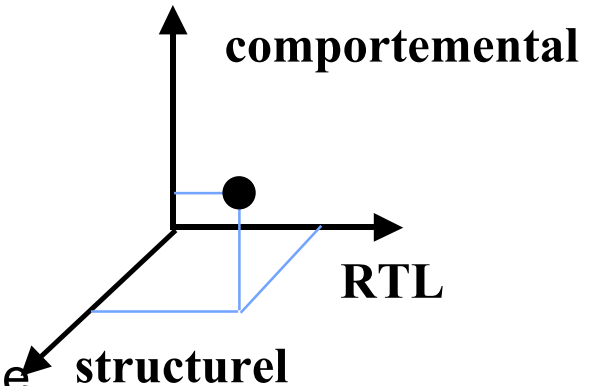
L'environnement doit être le plus précis et le plus complet possible (un investissement à multi-usage).

Autres méthodes, autres outils:

- preuve (en test prochainement)
- simulation électriques fines (bien, mais coûteux)
- analyse de chemins critiques (de façon statique)
- vérificateur électrique (ERC)

VHDL et la synthèse

- **Nécessité d'un formalisme commun : VHDL**
- **VHDL explore l'espace 3D des niveaux d'abstraction**
  - large spectre d'abstractions
  - hiérarchique modulaire et non ambigu
  - supporte les types de données abstrait -> abstraction
  - idéale pour définir une spécification exécutable
  - richesse du jeu d'instruction
  - modèles réutilisables (généricité, type non contraints)



- **Nécessité de définir un sous-ensemble**
  - Ne pas faire un sous-langage propre à chaque outil
    => définir le sous-ensemble le plus largement acceptable
  - Types de données
  - Opérations (avec contraintes d'utilisation)
    logiques, arithmétiques, comparaisons, décalages
  - Instructions de synchronisation (wait), conditionnelles, de boucle, d'affectation
  - Constructions pour la hiérarchie et la structuration du code ou pour le style