

# Cours d'électronique numérique

Aurélie Gensbittel  
Enseignante-Chercheuse  
Bertrand Granado  
Enseignant-Chercheur

Université Pierre et Marie Curie  
Mel : Aurelie.Gensbittel@upmc.fr  
Mel : Bertrand.Granado@upmc.fr

Hiver 2007

# Plan du Cours

- Introduction
- Algèbre de Boole et Codage
- Introduction au vhdl
- Fonctions Combinatoire Simples
- Fonctions Combinatoires Complexes
- Fonctions Séquentielles Simples
- Fonctions Séquentielles Complexes
- Pipeline
- Technologie des circuits numériques
- Les Convertisseurs Analogiques-Numériques et Numériques-Analogiques

## Les enseignants de l'UE201 ...

Cours	A. Gensbittel
TD	P. Ravary - H. Kokabi
TP	P. Ravary - H. Kokabi

## Evaluation des connaissances ...

- Le Contrôle Continu : Type QCM, Questions de Cours
- L'examen Final : Réflexion (Les TD ne servent pas à Bachoter !)
- Le Mini-Projet

## Evaluation des connaissances: Le Mini-Projet

- Rapport Final à la fin de toutes les séances de TP
- Evaluation durant le TP
- Soutenance de Mini-Projet

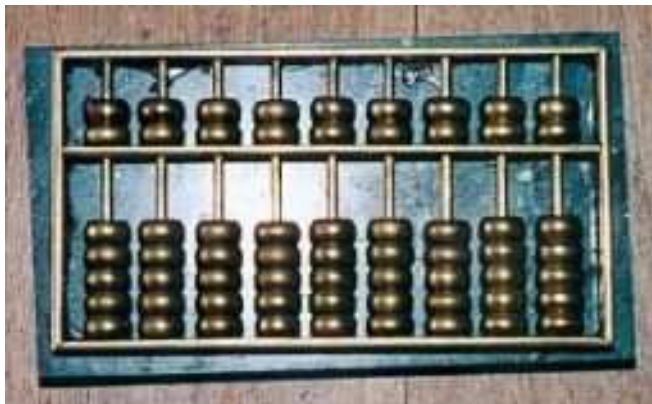
## Evaluation des connaissances : Coefficients

- Note de l'écrit  $E$  :  $E = \sup((0,2CC + 0,8Ex), Ex)$  où  $CC$  est la note du contrôle continu et  $Ex$  la note de l'examen final
- Note Finale  $F$  :  $F = 0,65 * E + 0,25 * TP + 0,1 * CC$  où  $TP$  est la note des travaux pratiques.

## Polycopié de TD

- Disponible à la reprographie la semaine
- 2ième étage - bâtiment Esclangon
- Horaires : 12 h 45 - 14 h 15 - Mardi - Mercredi - Vendredi
- **A avoir avant le TD !**

## Un peu d'histoire ... Le boulier chinois

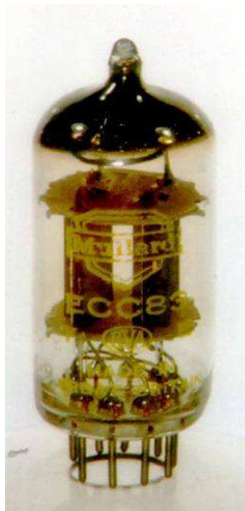




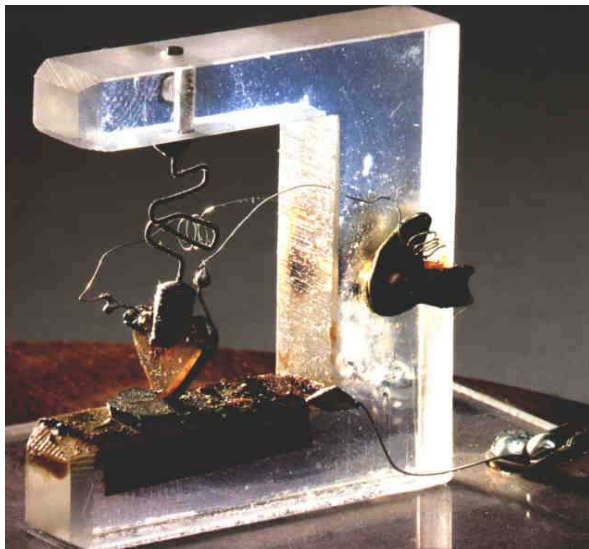
## Un peu d'histoire ... La machine de Babbage



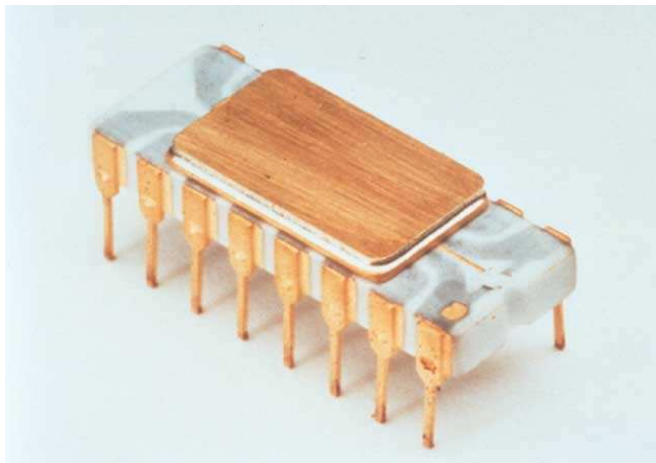
## Un peu d'histoire ... Une Lampe Triode



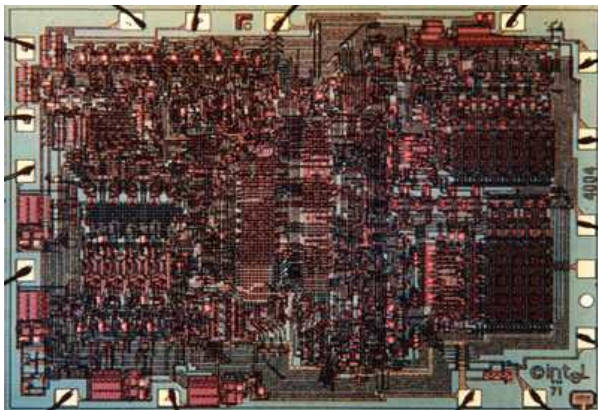
## Un peu d'histoire ... Le transistor de 1947



## Un peu d'histoire ... Le 4004 d'Intel



## Un peu d'histoire ... Le 4004 d'Intel



# Aujourd'hui : Un Monde Numérique ?

- Le monde numérique est un monde discrétisé
- Le monde réel est un monde continu
- Le monde n'est pas numérique
- Alors pourquoi l'électronique numérique ?
- Discrétisation permet de se prémunir du bruit
- Il est possible de quantifier les erreurs
- Possibilité de mise en oeuvre d'une arithmétique basée sur la logique

# Plan du Cours

- Introduction
- Algèbre de Boole et Logique Combinatoire

# Références

- **Architecture des Ordinateurs** - *Philippe Darche* - **Vuibert**,  
**Collection Passeport pour l'informatique** - 77.8 DAR - MIE
- **Circuits Numériques** - *Ronald J. Tocci* - **Dunod** - PE

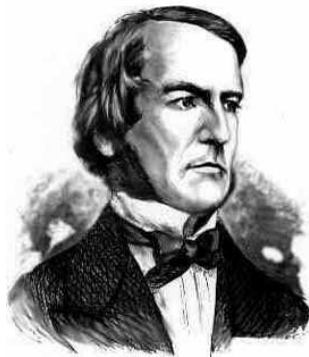


# Algèbre de Boole

Algèbre de Boole

# Mister G. Boole

- Mathématicien Anglais du 19<sup>ième</sup> siècle.



1815 – 1864

# La génèse

- Georges Boole introduit un formalisme mathématique de la logique  
*The Calculus of Logic*  
Cambridge and Dublin Mathematical Journal  
Vol. III (1848), pp. 183–9
- (3) *That those laws are capable of mathematical expression, and that they thus constitute the basis of an interpretable calculus.*
- Au départ beaucoup utilisé dans les jeux de salons
- Mais à l'arrivée : Véritable révolution qui est devenue le fondement de l'électronique numérique

# L'algèbre - Les bases - 1

- L'algèbre de Boole manipule des variables qui ne peuvent prendre que deux états : *Vrai* ou *Faux*
- Une telle variable est appelée variable *Booléenne*
- Il est possible aussi d'associer le chiffre 1 à la valeur *Vrai* et le chiffre 0 à la valeur *Faux*
- Les variables Booléennes dans ce cas sont des variables *Binaires*

# L'algèbre - Les bases

- exemples

# Algèbre de Boole - Définitions

- Algèbre de Boole  $B$ 
  - $B = \langle E, +, \cdot, ^-, 0, 1 \rangle$
  - $+$ ,  $\cdot$  sont des lois de composition interne
  - $^-$  est la loi de complémentation

# Algèbre de Boole - Lois de Composition

- loi de composition  $\cdot$

- |         |   |   |
|---------|---|---|
| $\cdot$ | 0 | 1 |
| 0       | 0 | 0 |
| 1       | 0 | 1 |

- loi de composition  $+$

- |     |   |   |
|-----|---|---|
| $+$ | 0 | 1 |
| 0   | 0 | 1 |
| 1   | 1 | 1 |

# Algèbre de Boole - Loi de complémentation

- Le *complément*  $\bar{a}$  d'une variable  $a$  est défini par :
  - si  $a = 1 \rightarrow \bar{a} = 0$
  - si  $a = 0 \rightarrow \bar{a} = 1$
- La variable  $a$ , lorsqu'elle est notée  $a$ , est dite sous sa forme normale
- La variable  $a$ , lorsqu'elle est notée  $\bar{a}$ , est dite sous sa forme complétementée



# Axiomes de bases - 1

## ■ Commutativité

- $\forall (a, b) \in E^2$
- $a + b = b + a$
- $a.b = b.a$

## ■ Distributivité

- $\forall (a, b, c) \in E^3$
- $a + (b.c) = (a + b).(a + c)$
- $a.(b + c) = (a.b) + (a.c)$

# Axiomes de bases - 2

## ■ Éléments Neutre

- $\forall a \in E$
- $a + 0 = a$
- $a \cdot 1 = a$

## ■ Complémentation

- $\forall a \in E$
- $a + \bar{a} = 1$
- $a \cdot \bar{a} = 0$

# Propriétés - 1

- A partir des axiomes de base des propriétés fondamentales sont déduites.
- Éléments Absorbants
  - $\forall a \in E$
  - $a + 1 = 1$
  - $a \cdot 0 = 0$
- Loi d'idempotence
  - $\forall a \in E$
  - $a + a = a$
  - $a \cdot a = a$

## Propriétés - 2

- Loi d'involution

- $\forall a \in E$

- $\overline{\overline{a}} = a$

- Loi d'absorption

- $\forall (a, b) \in E^2$

- $a + (a.b) = a$

- $a.(a + b) = a$

# Propriétés - 5

## ■ Loi d'associativité

- $\forall (a, b, c) \in E^3$

- $a + (b + c) = (a + b) + c$

- $a.(b.c) = (a.b).c$

## ■ Loi de De Morgan

- $\overline{\overline{(a, b)}} \in E^2$

- $\overline{a + b} = \overline{a}.\overline{b}$

- $\overline{a.b} = \overline{a} + \overline{b}$

# L'algèbre - Ordre et Fonction

- Relation d'ordre :
  - Ordre Total :  $0 < 1$
  - Ordre Lexicographique :  $00 < 01 < 10 < 11$  *Utile pour les tables de vérité*
- Définition d'une fonction logique :
  - $f(x_{n-1}, x_{n-2}, \dots, x_1, x_0) : 0, 1^n \rightarrow 0, 1, n \in \mathbb{N}^*$

# Fonctions Logiques à une variable $a$

- 1 variable soit 4 fonctions possibles :
  - $f = 0$  : fonction constante nulle
  - $f = 1$  : fonction constante à un
  - $f = a$  : fonction identité
  - $f = \bar{a}$  : fonction complément ou fonction *NON*

# Fonctions Logiques à deux variables $a$ et $b$

- 2 variables soit 16 fonctions possibles
  - $f = a.b$  : fonction *ET*
  - $f = a + b$  : fonction *OU*
  - $f = a \oplus b$  : fonction *OU-EXCLUSIF*
  - $f = \overline{a.b}$  : fonction *NON-ET*
  - $f = \overline{a + b}$  : fonction *NON-OU*
  - $f = \overline{a \oplus b}$  : fonction *NON-OU-EXCLUSIF*
  - etc...



# Fonctions Logiques à $n$ variables

- $n$  variables soit  $2^{2^n}$  fonctions possibles
  - 3 variables  $\rightarrow$  256 fonctions possibles
  - 4 variables  $\rightarrow$  65536 fonctions possibles
  - etc ...

# Représentation des fonctions logiques

- La Table de Vérité
- Représentation sous forme de tableau des valeurs de la fonction logique pour toutes les combinaisons de ses variables

a	b	f
0	0	$f_0$
0	1	$f_1$
1	0	$f_2$
1	1	$f_3$

# Représentation des fonctions logiques

- Le Tableau de Karnaugh
  - Représentation sous forme de matrice des valeurs de la fonction logique pour toutes les combinaisons de ses variables en exploitant la propriété d'adjacence

	b	0	1
a	c		
0	0	$f_0$	$f_1$
0	1	$f_2$	$f_3$
1	1	$f_6$	$f_7$
1	0	$f_4$	$f_5$

# Représentation des fonctions logiques

- Diagramme de Veitch
- Diagramme de Venn
- Arbre de décision binaire
- Logigramme *Partie technologie*
- Représentation algébrique *Ecriture logique*

# Écriture Algébrique

- La représentation sous forme de tableau ou de matrice est limitée  $\sim 5$  variables.
- Nécessité d'utiliser une écriture algébrique
- La fonction logique s'exprime alors sous la forme de variables booléennes reliées entre elles par des opérateurs de l'algèbre de Boole
- $f(a) = \bar{a}$  Fonction NON
- $f(a, b, c) = \bar{c}b + a\bar{b}$

## Écriture Algébrique - Minterme et Maxterme

- Un produit booléen de variables booléennes est appelé *p-terme*
- Une somme booléenne de variables booléennes est appelée *s-terme*
- Un *Minterme* est un p-terme de degré  $n$

$$m_j = \prod_{i=0}^{n-1} \tilde{a}_i, \tilde{a}_i \in (\bar{a}_i, a_i)$$

- Un *Maxterme* est un s-terme de degré  $n$

$$M_j = \sum_{i=0}^{n-1} \tilde{a}_i, \tilde{a}_i \in (\bar{a}_i, a_i)$$

## Écriture Algébrique - Minterme et Maxterme

- La somme logique de tous les Mintermes est égale à 1 si la fonction réalisée est différente de la fonction constante 0

$$\sum_{j=0}^{p-1} m_j = 1$$

- Le produit logique de tous les Maxtermes est égal à 0 si la fonction réalisée est différente de la fonction constante 1

$$\prod_{j=0}^{p-1} M_j = 0$$

- Relation entre Minterme et Maxterme

$$\overline{m_j} = M_j$$

# Écriture Algébrique - Minterme et Maxterme

- Exemples



# Écriture Algébrique - Forme Canonique

- Écriture algébrique d'une fonction logique n'utilisant que des Mintermes ou des Maxtermes.
- Il existe deux possibilités d'écriture :
  - *Forme Canonique Disjonctive ou première forme canonique :*  
*Elle s'exprime sous forme d'une somme de Mintermes*
  - *Forme Canonique Conjonctive ou seconde forme canonique :*  
*Elle s'exprime sous forme d'un produit de Maxtermes*

## Écriture Algébrique - Forme Canonique

- Fonction Ou-exclusif  $\oplus$  : la valeur de la fonction est un si une et une seule des deux variables a la valeur un.

a	b	f
0	0	0
0	1	1
1	0	1
1	1	0

- Forme Canonique Disjonctive :  
 $f(a, b) = \bar{a}b + a\bar{b} \rightarrow$  Somme des Mintermes tel que  $f(a,b)=1$ , lu directement de la table
- Forme Canonique Conjonctive :  
 $f(a, b) = (a + b).(\bar{a} + \bar{b}) \rightarrow$  Produit des Maxtermes tel que  $f(a,b)=1$ , cherche les mintermes pour lesquels  $f(a,b)=0$  et on détermine les valeurs de  $a$  et de  $b$  liées à ce minterme qui nie  $f(a,b)=0$

# Écriture Algébrique - Forme Canonique

- Exemples

## Domaine de définition des fonctions

- Une fonction logique peut-être soit *complètement* soit *incomplètement* définie
- Une fonction est complètement définie lorsque pour toutes les combinaisons de ses variables la valeur de la fonction est définie
- Une fonction est complètement définie lorsque pour toutes les combinaisons de ses variables la valeur de la fonction est définie

a	b	f
0	0	0
0	1	0
1	0	0
1	1	1

## Domaine de définition des fonctions

- Une fonction est incomplètement définie lorsque pour toutes les combinaisons de ses variables la valeur de la fonction n'est pas définie
- Une fonction est incomplètement définie lorsque pour toutes les combinaisons de ses variables la valeur de la fonction n'est pas définie

a	b	f
0	0	1
0	1	X
1	0	X
1	1	1

## Minimisation de Fonctions

- Utilisation des axiomes de base et des Propriétés qui en découlent
- $f(a, b, c) = ab + bc + c$  en utilisant la loi d'absorption  $bc + c = c$  on obtient  $f(a, b, c) = ab + c$
- $f(a, b) = a.(\bar{a} + b)$  en utilisant l'axiome de la complémentation  $a.\bar{a} = 0$  on obtient  $f(a, b) = ab$ .
- $f(a, b, c) = (a + bc)ab = aab + abbc = ab + abc = ab$  en utilisant successivement la loi d'idempotence et la loi d'absorption.

# Minimisation de Fonctions

- Exemples

# Minimisation de Fonctions

- Une méthode graphique : Les Tableaux de Karnaugh
- Les variables sont présentées de façon à faire apparaître la loi d'absorption
- $a.b + a.\bar{b} = a$
- Pour ce faire le code binaire réfléchi ou code de Gray est utilisé



# Minimisation de Fonctions

- Les Tableaux de Karnaugh : étapes
- Regroupement d'ensembles de  $2^i$  cases de même valeur (en général de valeur 1) en maximisant  $i$  à chaque fois. Possibilité de regrouper les cases extrêmes
- Regrouper les cases de même valeur restantes avec des cases d'ensembles déjà établis pour avoir  $2^j$  cases en maximisant  $j$
- Ecrire l'équation booléenne algébrique.

# Minimisation de Fonctions

- Exemples

# Minimisation de Fonctions

- Les Tableaux de Karnaugh : remarques
- Dans le cas de fonctions incomplètement définies, considérer  $X$  comme un 1 afin de maximiser les ensembles
- Méthode limitée à  $\sim 5$  variables.

# Minimisation de Fonctions

- Exemples

# Codage

Codage

# Codage

- Système de base : codage décimal
- Conversion décimal-binaire et binaire-décimal
- $\nexists n \Rightarrow 2^n = 10$ , nécessité codage octal ou hexadécimal
- Codage DCB : Décimal Codé Binaire
- Code de Gray ou binaire réfléchi
- Code ASCII

# Conversion binaire-décimal

- La conversion binaire-décimal s'effectue simplement en réalisant la somme des bits pondérés par leur position
- $\sum_{i=0}^{n-1} b_i * 2^i$  où  $b_i$  est la valeur du bit de position  $i$

Nombre Binaire	1	0	1	1	0
Position du bit	4	3	2	1	0
Nombre Décimal	$2^4$	0	$2^2$	$2^1$	0
$= 16 + 0 + 4 + 2 + 0 = 22$					

## Conversion décimal-binaire

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

$$\begin{array}{r}
 29 \quad 2 \\
 1 \quad \frac{14}{2} \\
 \quad 0 \quad \frac{7}{2} \\
 \quad \quad 1 \quad \frac{3}{2} \\
 \quad \quad \quad 1 \quad \frac{1}{2} \\
 \quad \quad \quad \quad 1 \quad \frac{0}{2}
 \end{array}$$

Nombre binaire = 11101



# Codage Hexadécimal

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

# Codage Hexadécimal

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

# Codage Hexadécimal

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

## Conversion Hexadécimal-Décimal

- De même que pour la conversion binaire-décimal, il s'agit ici de faire une sommation pondéré des symboles

Nombre Hexadécimal	A	2	E
Position du symbole	2	1	0
Puissance associée	$16^2$	$16^1$	$16^0$
Nombre Décimal	$10 * 16^2$	$+2 * 16^1$	$+14 * 16^0$
	$= 2606$		

# Conversion Décimal-Hexadécimal

- De même que pour la conversion décimal-binaire on a recourt à la division

$$\begin{array}{r}
 311 \quad 16 \\
 7 \quad \frac{19}{16} \\
 \quad 3 \quad \frac{1}{16} \\
 \quad \quad 1 \quad \frac{0}{16}
 \end{array}$$

Nombre Héxadécimal = 137

# Conversion Hexadécimal-Binaire

- Le nombre binaire est déduit en remplaçant chaque chiffre hexadécimal par son quartet binaire

Nombre Hexadécimal	E	3	B	1
Nombre Binaire	1110	0011	1011	0001

## Conversion Binaire-Hexadécimal

- La méthode est l'inverse de la précédente
- on regroupe les bits par quartet et on remplace les quartets par leur équivalent hexadécimal.

Nombre Binaire	0101	1010	1100	1011
Nombre Hexadécimal	5	A	C	B

# Notations

- Les symboles 0,1 appartiennent au code binaire, décimal et hexadécimal
- les symboles 0,1,2,3,4,5,6,7,8 et 9 appartiennent au code décimal et hexadécimal
- Nécessité d'une convention d'écriture pour différencier

Binaire             $100_B$

Décimal            $100$

Hexadécimal     $100_H$



## Décimal Codé Binaire : DCB

- Remplacer chaque chiffre d'un nombre décimal par son équivalent binaire
- Faire une correspondance directe entre binaire et décimal

Nombre Décimal	5	3	7	1
Nombre Binaire	0101	0011	0111	0001
Nombre Binaire	0101	1001	1000	0011
Nombre Décimal	5	9	8	3

## Décimal Codé Binaire

- Sous-Utilisation de l'espace de représentation binaire
- 6 représentations interdites
- $1010_B, 1011_B, 1100_B, 1101_B, 1110_B, 1111_B$
- Différence entre codage binaire et DCB
- $231 = 11100111_B$  en binaire
- $231 = 001000110001_B$  en DCB

# Code de Gray

- Une représentation ne diffère de la précédente que d'un bit

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

# Code ASCII

- Besoin de traiter de l'information non numérique
- Information Alphanumérique : , ? R t j
- Mise en place d'un codage sur 7 bits : l'ASCII
- American Standard Code for Information Interchange
- 7 bits : 26 lettres minuscules, 26 lettres majuscules, 10 chiffres, 7 signes de ponctuation soit 69 signes à coder. Le reste sert pour des caractères spéciaux
- ASCII étendu : 8 bits

# ASCII

Caractère	Code Hexadécimal
A	41 <sub>H</sub>
E	45 <sub>H</sub>
I	49 <sub>H</sub>
M	4D <sub>H</sub>
N	4E <sub>H</sub>
Q	51 <sub>H</sub>
R	52 <sub>H</sub>
U	55 <sub>H</sub>

- 4E<sub>H</sub>55<sub>H</sub>4D<sub>H</sub>45<sub>H</sub>52<sub>H</sub>49<sub>H</sub>51<sub>H</sub>55<sub>H</sub>45<sub>H</sub>
- NUMERIQUE

Conception

Conception

# Comment Concevoir un Circuit ?

- Cela dépend du circuit.
- petits circuits : A la main, en schématique
- circuits moyens : A la main à l'aide de composants discrets
- gros circuits : A l'aide de langage de Description de circuits numériques

# Les Langages de description

- Langage de type HDL : Hardware Description Language
- VHDL : Volonté d'Industriels et de Chercheurs de définir un langage HDL
- Verilog : Issu de la société Cadence Inc.
- System C : Mettre au même niveau Logiciel et Matériel



# VHDL

- Existe depuis 1987 date de la première norme. 1993 seconde norme.
- Langage Mûr et couramment utilisé
- 3 Niveaux :
  - Niveau Structurel
  - Niveau Flot de Données
  - Niveau Comportemental

# VHDL - RTL

- RTL : Register Transfert Level
- Description Synthétisable
- Utilisable pour fondre un circuit
- Utilisée dans ce cours

# VHDL - Base

- 3 blocs de base:
  - Les bibliothèques
  - L'entité : Décrit l'interfaçage du composant
  - L'architecture : Décrit le fonctionnement du composant

# VHDL - Bibliothèque

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

# VHDL - Entité

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

# VHDL - L'architecture

$S = A \text{ et } B$

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

# flot de conception

- De la description au circuit
- Décrit en VHDL le circuit
- Simule le circuit
- Synthétise le circuit
- Placement-Routage du circuit
- Réalise un masque
- Cuisson du circuit

## Langage HDL, a quoi ça sert ?

- A la conception d'ASIC
- A la programmation de FPGA (cours SSN)
- A la vérification fonctionnelle de circuits numériques.



# Logique Combinatoire

Logique Combinatoire

# Logique Combinatoire

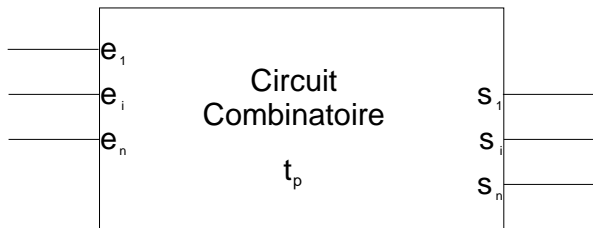
- Transposition de l'algèbre de Boole à *l'électronique*
- Rendu possible grâce au composant tel que le transistor commandé en tension
- Le domaine de validité de ce qui suit est l'électronique numérique

# Logique Combinatoire

## Définition :

Un circuit électronique est dit combinatoire si ses **sorties** sont déterminées par la **combinaison** de ses **variables d'entrées** et ceci après **un temps fini**. L'état d'un système est donc défini par la combinaison des variables  $e_1, \dots, e_i, \dots, e_n$ .

# Logique Combinatoire

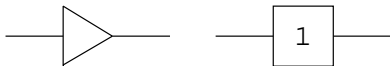


# Les Aléas Temporels

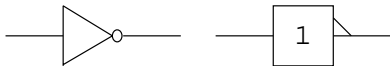
- $a + \bar{a} = 0$  ?

# Logique Combinatoire - Opérateurs de base

- Fonctions à une variable
- Buffer (identité) :  $s = a$



- Inverseur :  $s = \bar{a}$



# Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables
- ET (AND) :  $s = a.b$



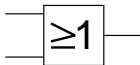
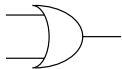
- NON-ET (NAND) :  $s = \overline{a.b}$



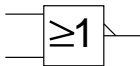
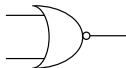
# Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables

- OU (OR) :  $s = a + b$



- NON-OU (NOR) :  $s = \overline{a + b}$

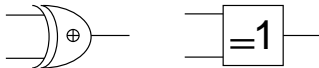




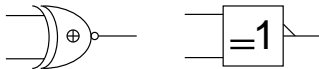
# Logique Combinatoire - Opérateurs de base

- Fonctions à deux variables

- OU-EXCLUSIF :  $s = a \oplus b = a\bar{b} + \bar{a}b$



- NON-OU-EXCLUSIF :  $s = \overline{a \oplus b} = \overline{a\bar{b} + \bar{a}b} = ab + \bar{a}\bar{b}$



# Plan du Cours

- Introduction
- Algèbre de Boole et Logique Combinatoire
- Fonctions Combinatoires Complexes

# Fonctions Combinatoires Complexes

Fonctions Combinatoires Complexes

# introduction

- Composées à partir des opérateurs de base
- Conditionnement de données
- Contrôle de données
- Définies par leur table de vérité

# Fonction Egalité

## ■ Egalité 2 bits

a	b	s
0	0	1
0	1	0
1	0	0
1	1	1

$$s = \overline{a \oplus b}$$

# Egalité - VHDL

```
entity egalite is
port( a,b : in std_logic;
      s : out std_logic);
end entity egalite;
```

```
architecture flot of egalite is
begin
    s <= not(a xor b);
end architecture flot;
```

# Fonction Égalité

- Égalité 2 mots de 2 bits
- $a = a_1, a_0$  et  $b = b_1, b_0$

$a_1$	0	0	1	1
$a_0$	0	1	1	0
$b_1 b_0$				
00	1			
01		1		
11			1	
10				1

- $s = \overline{(a_1 \oplus b_1)} \overline{(a_0 \oplus b_0)}$
- Égalité de 2 mots de n bits :  

$$s = \overline{(a_{n-1} \oplus b_{n-1})} \overline{(a_{n-2} \oplus b_{n-2})} \dots \overline{(a_1 \oplus b_1)} \overline{(a_0 \oplus b_0)}$$

# Egalité - VHDL

```
entity egalite is
port( a,b : in std_logic_vector(1 downto 0);
      s : out std_logic);
end entity egalite;
```

```
architecture flot of egalite is
begin
  s <= not(a(1) xor b(1)) and not(a(0) xor b(0));
end architecture flot;
```



# Egalité - VHDL

## VHDL apres l'UE LE201

Ce que peut faire VHDL  
Hors du cadre du cours LE201

```
entity egalite is
port( a,b : in std_logic_vector(1 downto 0);
      s : out std_logic);
end entity egalite;
```

```
architecture comp of egalite is
begin
  process(a,b) is
  begin
    if a= b then
      s<='1';
    else
      s<='0';
    end if;
  end process;
end architecture comp;
```

# Multiplexeurs

- Multiplexeur = Aiguillage
- Une commande choisie l'entrée
- Entrée choisie recopiée sur la sortie
- Partie Commande :  $p$  bits
- Partie Donnée :  $2^p = n$  entrées, 1 sortie

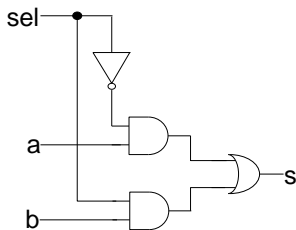
# Multiplexeurs 2 vers 1 - Table de vérité

sel	a	0	0	1	1
	b	0	1	1	0
0				1	1
1			1	1	

■  $s = \overline{sel}.a + sel.b$

# Multiplexeurs 2 vers 1

## ■ Schéma



# VHDL - mux2v1

```
entity m2v1 is
port(a,b,sel : in std_logic;
      s: out std_logic);
end entity m2v1;
```

```
architecture flot of m2v1 is
begin
    s <= (a and not(sel)) or (b and sel);
end architecture flot;
```

# VHDL - mux2v1

## VHDL un petit cran en plus

Ou l'on utilise des constructions VHDL un peu plus expressives

```
entity m2v1 is
port(a,b,sel : in std_logic;
      s: out std_logic);
end entity m2v1;

architecture flot of m2v1 is
begin
    s <= a when sel='0' else b; -- s prend la valeur de a s
                               -- sinon si sel <> '0' s pr
end architecture flot;
```

# Multiplexeurs 4 vers 1

- 4 données et 2 commandes
- $2^6 = 64$  lignes dans la table de vérité
- Toutes les lignes ne sont pas intéressantes
- Une commande  $\Rightarrow$  Une variable pertinente

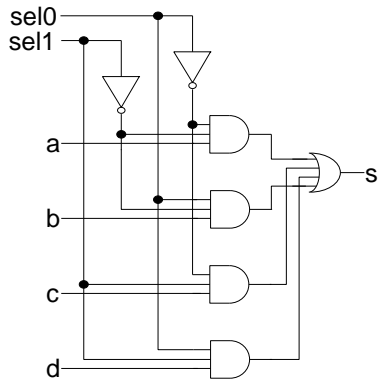
# Multiplexeurs 4 vers 1

sel1	sel0	s
0	0	a
0	1	b
1	0	c
1	1	d

$$s = \overline{\text{sel1}}.\overline{\text{sel0}}.a + \overline{\text{sel1}}.\text{sel0}.b + \text{sel1}.\overline{\text{sel0}}.c + \text{sel1}.\text{sel0}.d$$



# Multiplexeurs 4 vers 1



# Multiplexeurs

- Permet de réaliser des fonctions logiques
- Un Multiplexeur  $n$  vers 1 réalise  $2^n$  fonctions
- Valeurs des entrées = valeurs de la fonction
- Un Multiplexeur 4 vers 1

x	y	s	entrée mux
0	0	0	a = 0
0	1	0	b = 0
1	0	0	c = 0
1	1	1	d = 1

- x et y commandes du multiplexeur

# Démultiplexeurs

- Inverse du Multiplexeurs
- 1 données,  $p$  commandes,  $2^p = n$  sorties
- Démultiplexeur 1 vers 2

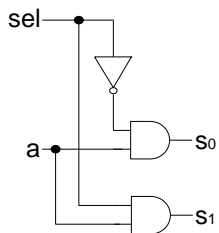
sel	a	s1	s0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

# Démultiplexeurs

sel	a	s1	s0
0	0	0	0
0	1	0	1
1	0	0	0
1	1	1	0

- $s0 = \overline{sel}.a$  et  $s1 = sel.a$

# Démultiplexeurs



# Démultiplexeurs - VHDL

```
entity demux is
port ( sel,a : in std_logic;
      s0,s1 : out std_logic);
end entity demux;

architecture flot of demux is
begin
    s0 <= a when sel = '0' else '0';
    s1 <= a when sel = '1' else '0';
end architecture flot;
```

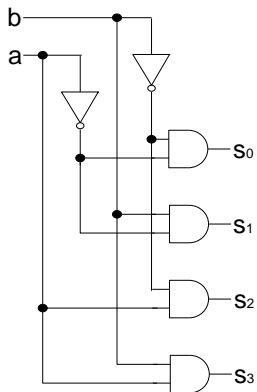
# Décodeurs

- Décodage Binaire  $\rightarrow$  Codage 1 parmi n
- n entrées,  $2^n$  sorties

a	b	s3	s2	s1	s0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

- Autant de tableaux de Karnaugh que de sorties !
- Trivial :
  - $s0 = \bar{a}.\bar{b}$ ,
  - $s1 = \bar{a}.b$ ,
  - $s2 = a.\bar{b}$ ,
  - $s3 = a.b$

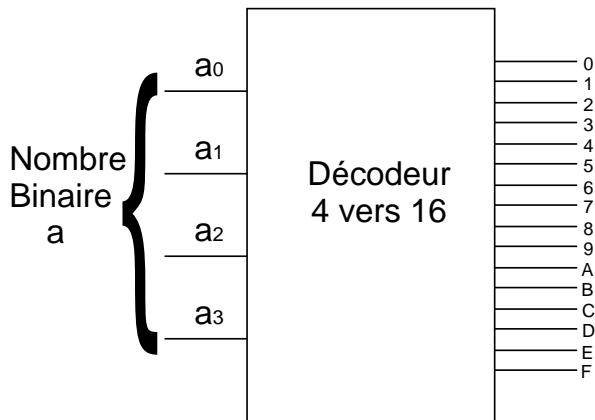
# Décodeur





# Décodeur

## ■ Décodeur Binaire Base n



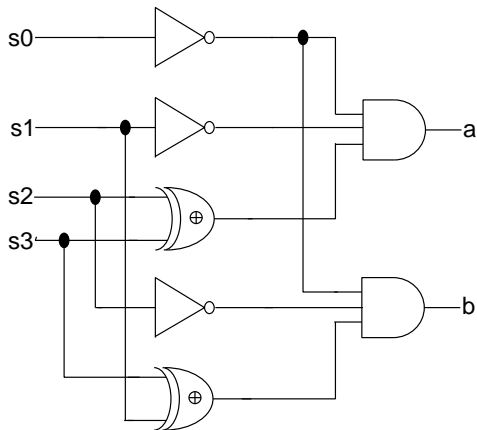
# Encodeurs

- inverseur des décodeurs : codeurs
- $2^n$  entrées, n sorties

s3	s2	s1	s0	a	b
0	0	0	1	0	0
0	0	1	0	0	1
0	1	0	0	1	0
1	0	0	0	1	1

- $b = \overline{s3}.\overline{s2}.s1.\overline{s0} + s3.\overline{s2}.\overline{s1}.\overline{s0} = \overline{s2}.\overline{s0}(s3 \oplus s1)$
- $a = \overline{s3}.s2.\overline{s1}.\overline{s0} + s3.\overline{s2}.\overline{s1}.\overline{s0} = \overline{s1}.\overline{s0}.(s3 \oplus s2)$

# Encodeurs



# Fonctions Combinatoires Complexes

Fonctions Combinatoires Arithmétiques

# Nombres Signés

- Comment Coder les Nombres Signés en Binaire ?
- Introduire un bit de signe : bit de poids fort
- Nombre sur 4 bits

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

- Codage Signe + Valeur Absolue
- Nécessite trop de logique pour réaliser des opérateurs arithmétiques

## Complément à 2

- Utilisation d'un codage qui permet de limiter les opérateurs
- Complément à 2 :
  - Bit de signe : bit de poids fort
  - Si bit de signe = 0 : Le nombre est codé
  - Si bit de signe = 1 : Complément à 2 pour avoir la valeur
- Principe : Pour un nombre de  $n$  bits complémenter le nombre pour arriver à  $2^n$

## Complément à 2

- Codage de 7 :

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

- Codage de -7 :

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

## Complément à 2

- Etapes pour complémenter à 2
- Faire le complément à 1 du nombre : complémentation bit à bit
- Ajouter 1 au nombre
- Exemple : codage de -5

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



## Complément à 2

- Avantage :
- Unicité du 0
- Utilisation du même opérateur pour l'addition et la soustraction
- Modulo :  $9_H - 4_H = (9_H + C_H) \text{ modulo } (10_H) = 5_H$
- Exemples en binaire.

## Complément à 2

- Codage sur  $N$  bits,  $N$  fini
- On veut coder un nombre négatif  $-P$  sur  $N$  bits,  $P \in [0, 2^N]$
- On sait que  $2^N = CP + P$
- On pose  $-P = CP \bmod 2^N$
- Ce qui donne  $-P = (2^N - P) \bmod 2^N$
- On sait que  $P \in [0, 2^N]$  donc on a bien  $(2^N - P) \bmod 2^N = -P$

## Complément à 2

Si  $P$  positif on le code

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

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

Si  $P$  négatif on le code

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

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

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

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

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

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

Nombre en complément à 2

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

# Demi-Additionneur

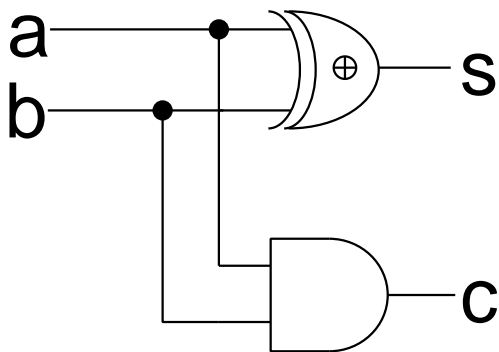
- Réalisation d'un demi-additionneur

a	b	s	r
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

- $s = a \oplus b$

- $r = a.b$

# Demi-Additionneur



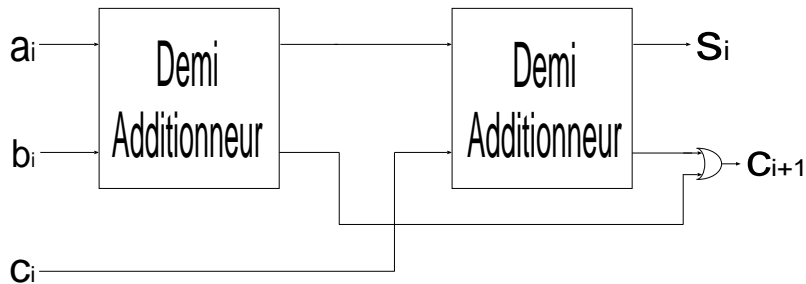
# Demi-Additionneur

```
entity demi-add is
port( a,b : in std_logic;
      s,c : out std_logic);
end entity demi-add;
architecture flot of demi-add is
begin
    s<= a xor b;
    c<= a and b;
end architecture flot;
```

# Additionneur 1 bit

- Introduction d'une retenue d'entrée
- Trois variables d'entrées, deux de sorties
- $a_i, b_i, c_i$  et  $s_i, c_{i+1}$
- $s_i = a_i \oplus b_i \oplus c_i$
- $c_{i+1} = a_i \cdot b_i + a_i \cdot c_i + b_i \cdot c_i$
- Utilisation de deux demi-additionneurs

# Additionneur 1 bit





## Additionneur 1 bit

```
entity add1 is
port( a,b,cin : in std_logic;
      s,cout : out std_logic);
end entity add1;
architecture struct of add1 is
signal stemp,ctemp1,ctemp2 : std_logic;
begin
  demi-add1 : entity work.demi-add(flot)
    port map(a,b,stemp,ctemp1);
  demi-add2 : entity work.demi-add(flot)
    port map(stemp,cin,s,ctemp2);
  cout <= ctemp1 or ctemp2
end architecture struct;
```

## Additionneur 1 bit

```
entity add1 is
port( a,b,cin : in std_logic;
      s,cout : out std_logic);
end entity add1;
architecture flot of add1 is
begin
    s<= a xor b xor cin;
    cout<= (a and b) or (a and cin)
           or (b and cin);
end architecture flot;
```

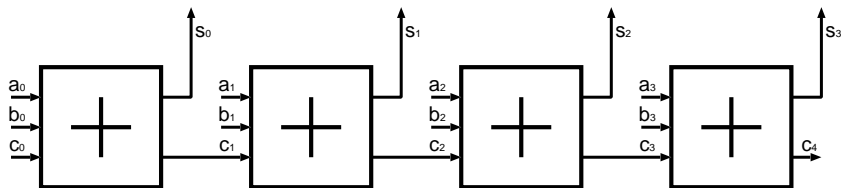
# Additionneur 1 bit

## VHDL apres l'UE LE201

Ce que peut faire VHDL  
Hors du cadre du cours LE201

```
entity add1 is
port( a,b  : in std_logic;
      s  : out std_logic);
end entity add1;
architecture comport of add1 is
begin
    s<= a + b;
end architecture comport;
```

# Additionneur 4 bits



## Additionneur 4 bits : Entité

```
ENTITY add4 IS
  port (a,b : in std_logic_vector(3 downto 0);
        cin : in std_logic;
        s : out std_logic_vector(3 downto 0);
        cout : out std_logic);
END ENTITY add4;
```

# Additionneur 4 bits : Architecture Simple

```
ARCHITECTURE struct_simple OF add4 IS
signal c : std_logic_vector(4 downto 0);
BEGIN

    c(0) <= cin;
    cout <= c(4);

    add1_0 : entity work.add1(flot)
        port map (a(0),b(0),c(0),s(0),c(1));

    add1_1 : entity work.add1(flot)
        port map (a(1),b(1),c(1),s(1),c(2));

    add1_2 : entity work.add1(flot)
        port map (a(2),b(2),c(2),s(2),c(3));

    add1_3 : entity work.add1(flot)
        port map (a(3),b(3),c(3),s(3),c(4));
END ARCHITECTURE struct_simple;
```

## Additionneur 4 bits : Architecture avec Génération

```
ARCHITECTURE struct_generate OF add4 IS
signal c : std_logic_vector(4 downto 0);
BEGIN

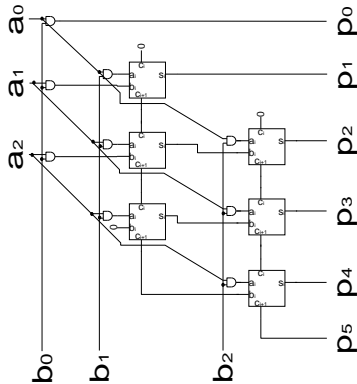
    c(0) <= cin;
    cout <= c(4);

    instance : for i in 0 to 3 generate
    add1_i : entity work.add1(flot)
        port map (a(i),b(i),c(i),s(i),c(i+1));
    end generate;

END ARCHITECTURE struct_generate;
```

# Multiplieur

- Utilisation de l'algorithme de Multiplication
- $n * m$  additions de  $n$





## Complément VHDL : Modélisation du temps

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

entity mon-et is
port(a,b : in std_logic;
      s : out std_logic);
end entity mon-et;

architecture flot of mon-et is
begin
s <= a and b after 25 ns;
end architecture flot;
```

## Complément VHDL : Décalage et mise à l'échelle

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

entity conversion is
port(a : in std_logic_vector(5 downto 0);
      s,s2,s3 : out std_logic_vector(11 downto 0));
end entity conversion;

architecture flot of conversion is
begin
a <= "010101";
s <= "0000" & a & "00"; -- s = "000001010100"
s2 <= "00000" & a & '0'; -- s2 = "000000101010"
s3 <= a & "000000"; -- s3 = "010101000000"
end architecture flot;
```

# Plan du Cours

- Introduction
- Algèbre de Boole et Logique Combinatoire
- Fonctions Combinatoires Complexes
- **Éléments séquentiels de base : Les Bascules**

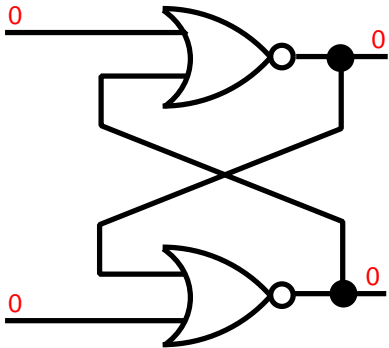
# Les éléments séquentiels de base

- Eléments de base
- Régulation du flux des données
- Fonction Mémoire
- Eléments Asynchrones
- Eléments Synchrones

# Éléments Asynchrones

## Les Bascules Asynchrones

# Le bascule RS



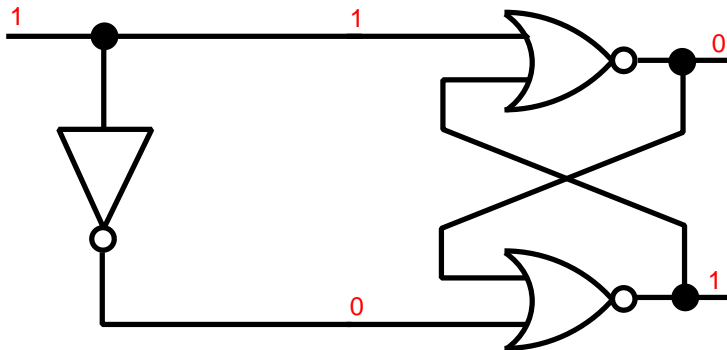
# Le bascule RS

- Table de Vérité

$R$	$S$	$Q$	$\overline{Q}$
0	0	$Q$	$\overline{Q}$
0	1	1	0
1	0	0	1
1	1	Etat	Interdit

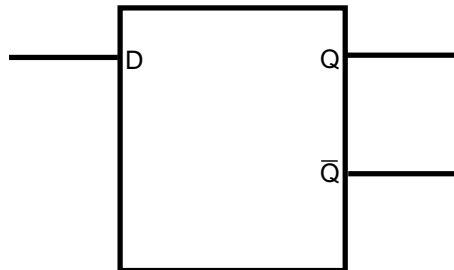
- Élément Asynchrone
- Base de toutes les bascules

## La bascule D





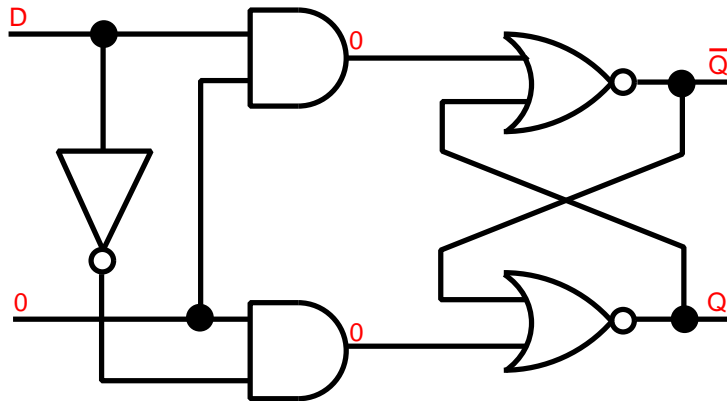
# La bascule D



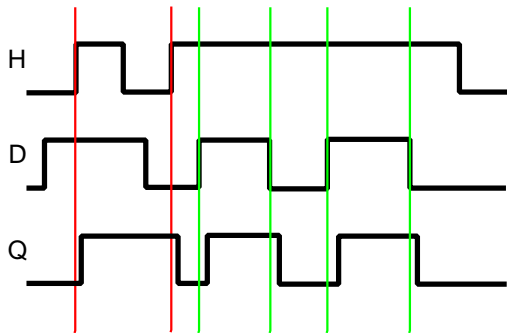
# Eléments Synchrones

## **Bascules Synchrones**

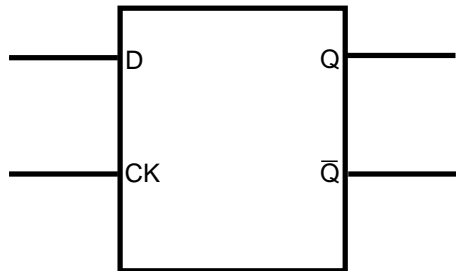
# La bascule D active sur niveau



# La bascule D active sur niveau



# La bascule D active sur niveau



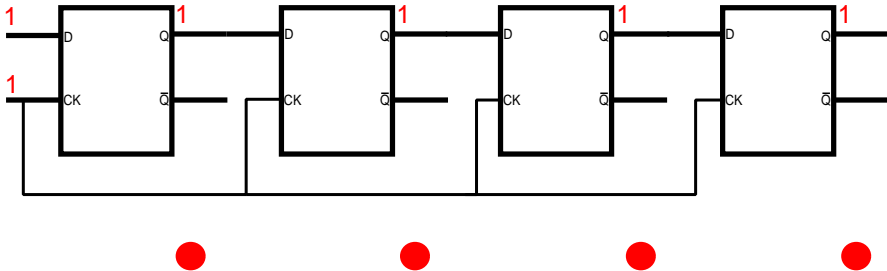
## La bascule D active sur niveau

## ■ Table de Vérité

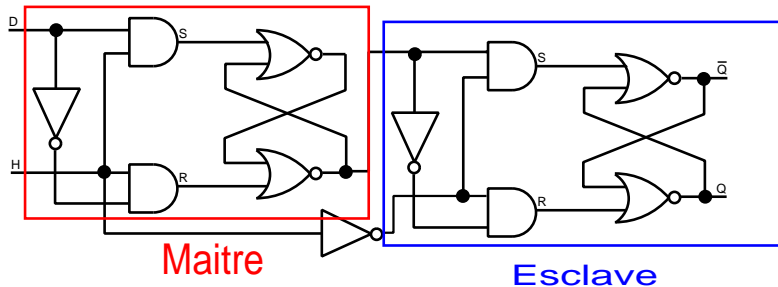
$D$	$H$	$Q_{n+1}$	$\overline{Q_{n+1}}$
0	0	$Q_n$	$\overline{Q_n}$
0	1	0	1
1	0	$Q_n$	$\overline{Q_n}$
1	1	1	0

$D$	$H$	$Q_{n+1}$	$\overline{Q_{n+1}}$
X	0	$Q_n$	$\overline{Q_n}$
0	1	0	1
1	1	1	0

# La bascule D active sur niveau : chenillar



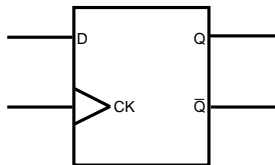
# La bascule D active sur front



Architecture Maître-Esclave



# La bascule D active sur front



# La bascule D active sur front

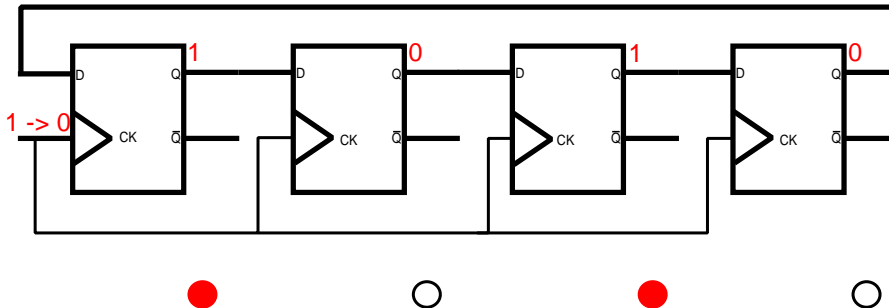
- Table de Vérité

- Front Montant

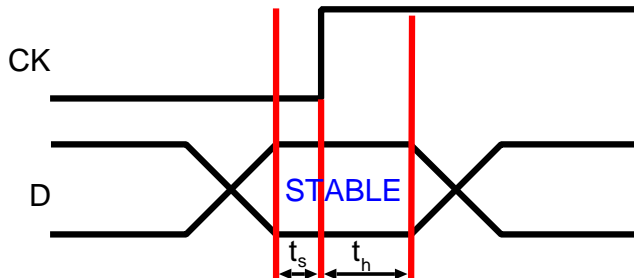
- Front Descendant

$D$	$H$	$Q$	$\overline{Q}$
X	0	$Q$	$\overline{Q}$
0	↓	0	1
1	↓	1	0

# La bascule D active sur front : chenillar



# La bascule D active sur front : Considérations Temporelles



- $t_s$  est le temps de prépositionnement (*setup en anglais*)
- $t_h$  est le temps de maintien (*hold en anglais*)

## Vhdl : Bascule D active sur front

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

architecture comport of bascule is
begin
  stockage : process(d,clk) is
  begin
    if clk='1' and clk'event then
      q <= d;
    end if;
  end process stockage;
end architecture comport;
```

## La bascule JK

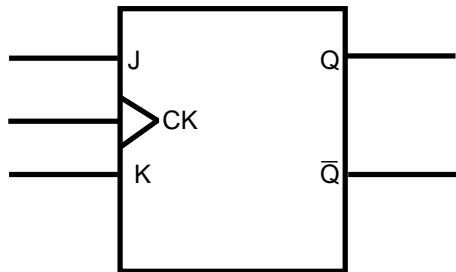
## ■ Front Montant

$J$	$K$	$H$	$Q_{n+1}$	$\overline{Q_{n+1}}$
0	0	↑	$Q_n$	$\overline{Q_n}$
0	1	↑	0	1
1	0	↑	1	0
1	1	↑	$\overline{Q_n}$	$Q_n$

## ■ Front Descendant

$J$	$K$	$H$	$Q_{n+1}$	$\overline{Q_{n+1}}$
0	0	↓	$Q_n$	$\overline{Q_n}$
0	1	↓	0	1
1	0	↓	1	0
1	1	↓	$\overline{Q_n}$	$Q_n$

# Symbole Bascule JK



## La bascule T

- Front Montant

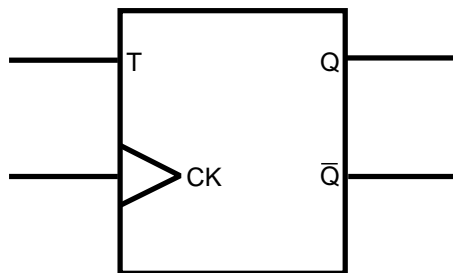
$T$	$H$	$Q_{n+1}$	$\overline{Q_{n+1}}$
0	↑	$Q_n$	$\overline{Q_n}$
1	↑	$\overline{Q_n}$	$Q_n$

- Front Descendant

$T$	$H$	$Q_{n+1}$	$\overline{Q_{n+1}}$
0	↓	$Q_n$	$\overline{Q_n}$
1	↓	$\overline{Q_n}$	$Q_n$



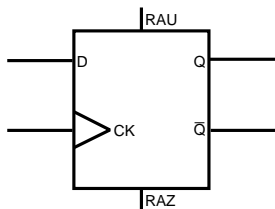
# Symbole Bascule JK



# Entrées Asynchrones

- Nécessité de forcer les sorties
- Ajout d'entrées asynchrones
- RAU : Remise à Un ou entrée *Set* en anglais
- RAZ : Remise à Zéro ou entrée *Reset* en anglais

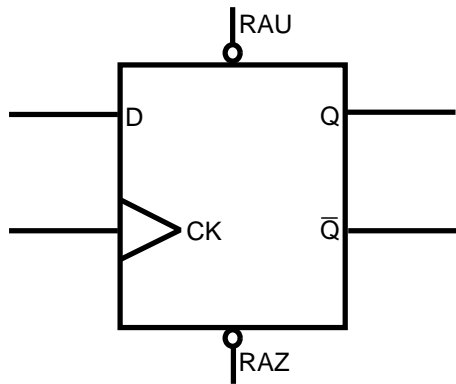
# Entrées Asynchrones



$D$	$H$	$RAU$	$RAZ$	$Q_{n+1}$	$\overline{Q_{n+1}}$
X	X	1	0	1	0
X	X	0	1	0	1
0	$\uparrow$	0	0	0	1
1	$\uparrow$	0	0	1	0

- Combinaison  $RAU=RAZ=1$  interdite
- Entrées RAU et RAZ souvent actives à 0

# Entrées Asynchrones



# Et après ?

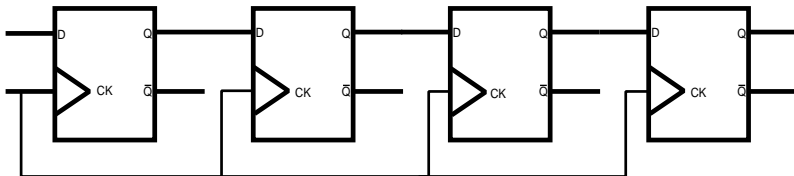
## Les registres

# Les registres

- Taille des données 1 bits
- Associer des Bascules pour augmenter la taille
- Éléments importants dans les micro-processeurs : permet de réaliser un pipeline
- Réalise des barrières de synchronisation

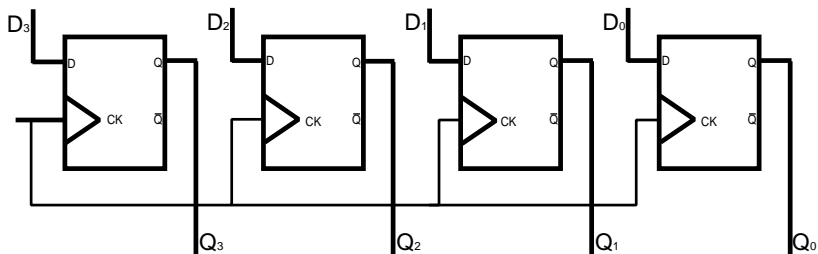
# Registre bascules D

## Registre à Décalage



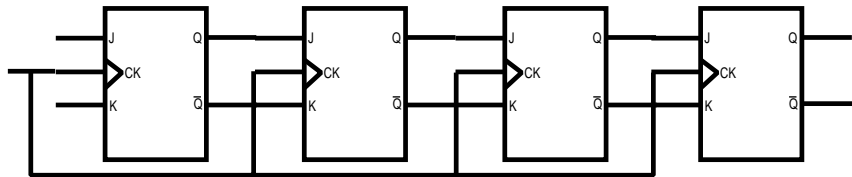
# Registre bascules D

## Registre à chargement parallèle





# Registre bascules JK



## Vhdl : registre

```
entity reg8generic is
generic (N : natural := 8);
port ( d :in std_logic_vector(N-1 downto 0);
      clk : in std_logic;
      q: out std_logic_vector(N-1 downto 0) );
end entity reg8generic;
```

```
architecture comport of reg8generic is
begin
  stockage : process(clk,d) is
  begin
    if (clk='1' and clk'event) then
      q <= d;
    end if;
  end process stockage;
end architecture comport;
```

## Les monostables

# Monostable simple

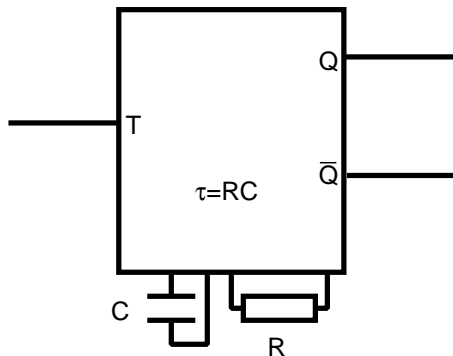
- Circuit ne possédant qu'un état stable

- Souvent :

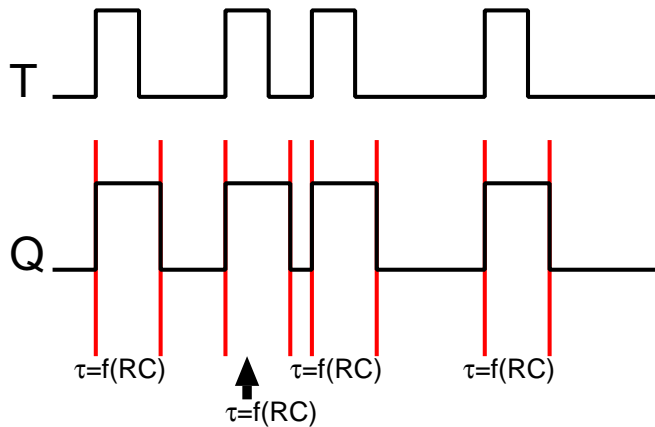
$Q$	$\overline{Q}$	Etat
0	1	Stable
1	0	Quasi Stable

- Etat Quasi Stable est momentané
- Durée Quasi Stable fixée par circuit RC

# Monostable simple

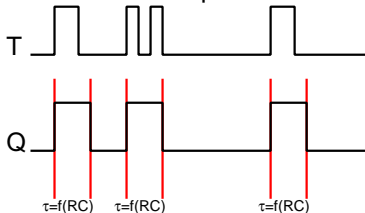


# Monostable simple

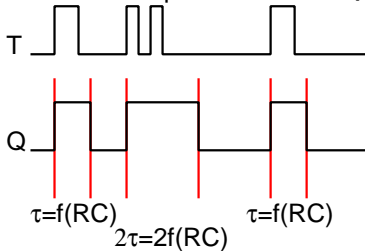


# Monostable redéclencheable

- Monostable Simple : Durée Etat Quasi Stable Fixe



- Nécessité de pouvoir rester Quasi Stable



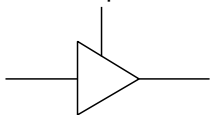
## Les portes 3 états

- Boole = 2 états : 0 et 1
- Connexion de 2 composants sur le même fil ?
- Première Solution : Court Circuit **PERDU**
- Seconde Solution : multiplexeur **encombrant**
- Troisième Solution : Composant d'interface La porte 3 états **GAGNE**



# Les portes 3 états

- Introduction d'un Etat Z
- Etat Haute Impédance
- Tout ce passe comme si le fil n'était pas connecté



# Vhdl : registre

```
entity reg8generic is
generic (N : natural := 8);
port ( d :in std_logic_vector(N-1 downto 0);
      en, clk : in std_logic;
      q: out std_logic_vector(N-1 downto 0) );
end entity reg8generic;
```

```
architecture comport of reg8generic is
begin
  stockage : process(d,en,clk) is
  begin
    if (clk='1' and clk'event) then
      if en = '1' then
        q <= d;
      else
        q<= (others=>'Z');
      end if;
    end if;
  end process stockage;
end architecture comport;
```

# Systèmes Séquentiels Complexes

## Systèmes Séquentiels

# Systèmes Séquentiels

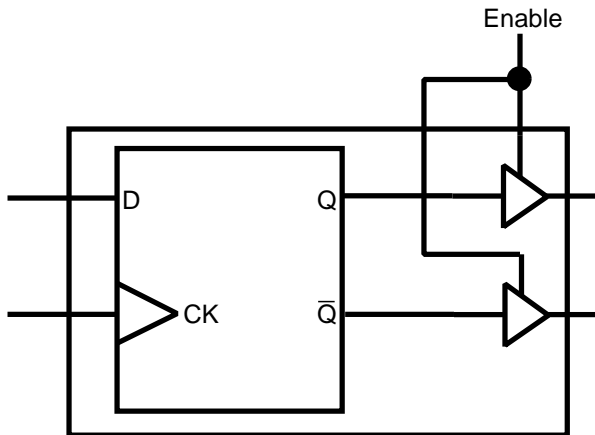
- Les registres (Déjà vus)
- Les mémoires
- Les compteurs
- Le contrôle (les pipelines)
- Les Machines à Etats (Cours Synthèse des Systèmes Numériques)
- What Else ?

# Les mémoires

- Association de plusieurs Registres
- Utilisation de Bascules D souvent
- Mémoires Asynchrones  $\Rightarrow$  Bascules Asynchrones
- Mémoires Synchrones  $\Rightarrow$  Bascules Synchrones
- Ajout d'une entrée de sélection

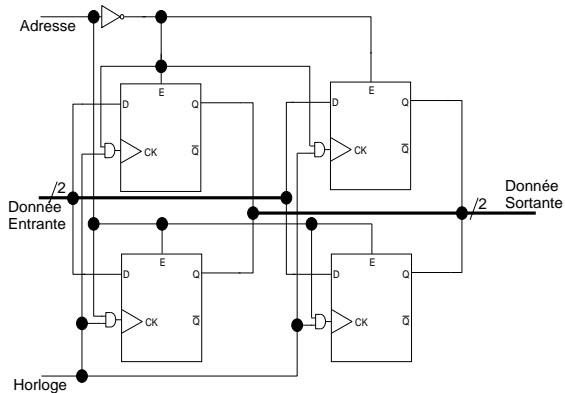
# Les mémoires

- Sélection  $\Rightarrow$  sortie 3 états



# Les mémoires

## ■ Mémoire 2 mots de 2 bits



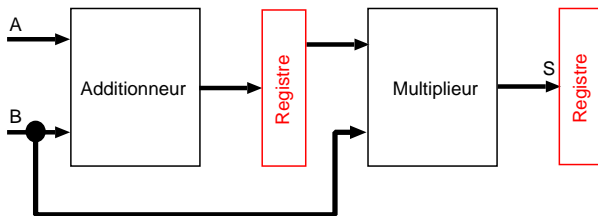
# Les Compteurs

- Système séquentiel dont les sorties sont une suite pré-déterminée d'états
- Bascules D ou JK
- Plus ou Moins Complexes
  - Simple (Bête) compte de 0 à  $N - 1$  en boucle
  - $\log_2(N)$  bascules
  - Complexes
    - Initialisation
    - Arrêt - Reprise
    - Compteur - Décompteur
    - Fonctions nécessaires à l'application



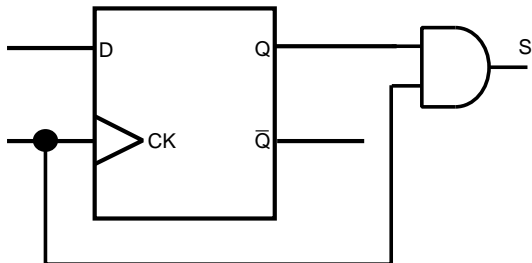
# Le contrôle

- Contrôle du flux des données
- Un registre entre 2 opérateurs
- Réalisation d'un pipeline

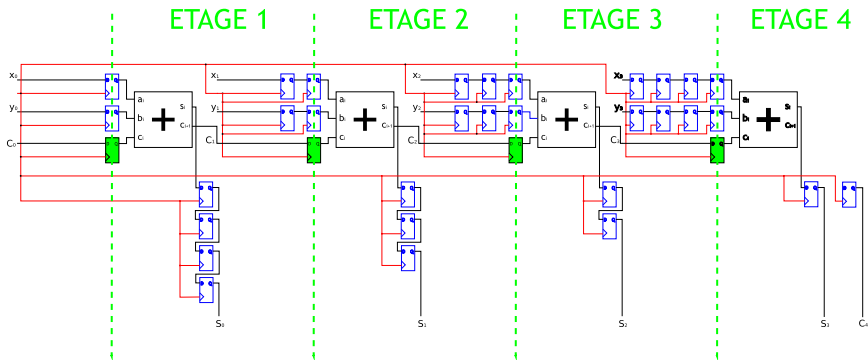


- $S_n = (A_{n-1} + B_{n-1}) * B_n$

# Les Aléas



# Pipeline



# Pipeline

## Principe :

Découper l'opérateur en plusieurs étages isolés les uns des autres. C'est le signal d'horloge qui autorise les transferts des données d'un étage à l'autre. Cette technique permet d'accélérer la cadence de production des résultats.

## Latence

La latence (durée d'exécution totale) de l'opérateur est supérieure d'au moins  $\epsilon$  à celle d'un opérateur non pipeliné. Si la durée maximale d'un étage de l'opérateur pipeliné est de  $t_{max}$  et qu'il y a  $n$  étages, alors la latence de l'opérateur est  $T = n * t_C LK$  avec  $t_C / k > t_{max} + t_h + t_s$  avec  $t_s$  et  $t_h$  les temps de prépositionnement et de maintien des bascules.

## Références

- **Conception des ASICS** - *P. Naish et P. Bishop* - **Masson**
- **Logic Reference Guide 4Q2001** - *Texas Instrument* -  
<http://www.ti.com>
- **Des Liens Utiles pour tous les Electroniciens** -  
<http://perso.wanadoo.fr/xcotton/electron/index.htm>

# Vocabulaire

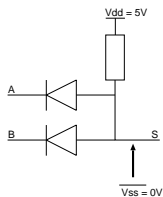
- Quelques grandeurs caractéristiques
- $F$  Fréquence de Fonctionnement
- $P$  Puissance Dissipée
- $T_p$  Temps de Propagation
- Sortance : nombre maximal d'entrées qui peuvent être pilotées par une sortie.

# Intégration ?

- Plusieurs niveaux d'intégration
- SSI: Small Scale Integration  $< 12$  portes logiques
- MSI: Mean Scale Integration de 12 à 99 portes logiques
- LSI: Large Scale Integration de 100 à 9999 portes logiques
- VLSI: Very Large Scale Integration de 10000 à 99999 portes logiques
- ULSI: Ultra Large Scale Integration 100000 et plus de portes logiques

# Technologie - 1

- Comment réaliser des fonctions logiques ?
- Nécessité de composants commandés électriquement
- Les Diodes (Famille DTL)



- Les Transistors

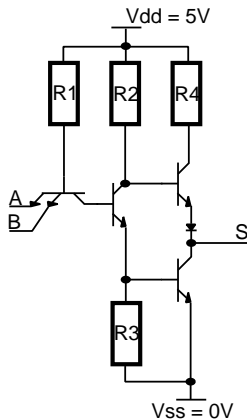


## Technologie - 2

- La logique Bi-polaire
- Famille originelle des transistors
- Bipolaire=Rapidité
- Bipolaire=Consommation

# Bipolaire

## ■ La Famille TTL : Transistor - Transistor - Logic



# T.T.L 1

- Historique : première famille : 1964 - Texas Instrument - Famille 54/74
- Famille 74 : Famille de référence
- Tous les fabricants utilisent cette référence
- Compatibilité des circuits entre eux
- Tension d'alimentation : 0-5V
- 1 logique :  $V_{ih} > 2V$
- 0 logique :  $V_{il} < 0,8V$

# T.T.L 2

- Plusieurs Variantes de la famille 74
- 74L : Famille Faible consommation
- 74H : Famille rapide
- 74S : Famille rapide basée sur des Diodes Schottky
- 74LS : Famille Schottky Faible consommation
- 74AS et 74 ALS : Famille Avancée Schottky

## T.T.L 3

	74	74L	74S	74LS	74ALS
$t_p$ (ns)	9	33	3	9,5	4
$P$ (mW)	10	1	20	2	1,2
$F_{max}$ (MHz)	35	3	125	45	70 (200 pour AS)
Sortance	10	20	20	20	20 (40 pour AS)

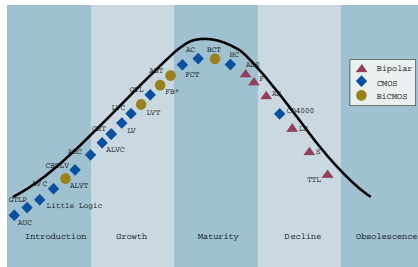
# Bipolaire

- Il existe d'autres technologies bipolaire pour le numérique
- ECL : Emitter Coupled Logic - Plus Rapide et Plus chère
- I<sup>2</sup>L : Utilisé dans des circuits logiques complexes

# Maintenant et Demain

- Quelle technologie aujourd'hui et demain ?
- Hasardeux de prédire l'avenir
- Mais ...
- Prévisions Texas Instruments

PRODUCT LIFE CYCLE



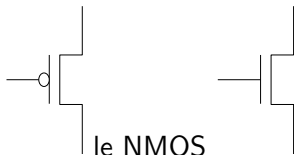
# CMOS

- Avantage du CMOS
- Faible Coût
- Facilité de fabrication
- Petite Dimension, jusqu'à un rapport 50 par rapport au bipolaire
- Pas d'élément résistif
- Faible consommation originale



# Principe de Base du CMOS - 1

- Il existe deux types de transistors dans le CMOS



- le PMOS

le NMOS

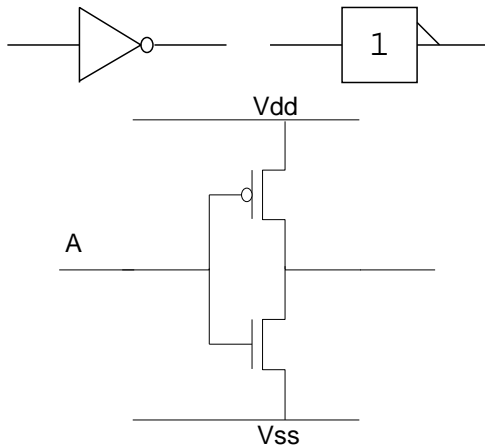
- Mise en oeuvre de charges mobiles de polarité complémentaire  
⇒ ComplementaryMOS
- NMOS conduit si  $V_g > V_s$  de plus de  $V_T$
- PMOS conduit si  $V_g < V_s$  de plus de  $V_T$

## Principe de Base du CMOS - 2

- Numérique  $\implies$  Fonctionnement en mode interrupteur
- NMOS  $V_g = 0 \implies$  transistor bloqué, interrupteur ouvert
- NMOS  $V_g = V_{dd} \implies$  transistor passant, interrupteur fermé
- PMOS  $V_g = 0 \implies$  transistor passant, interrupteur fermé
- PMOS  $V_g = V_{dd} \implies$  transistor bloqué, interrupteur ouvert

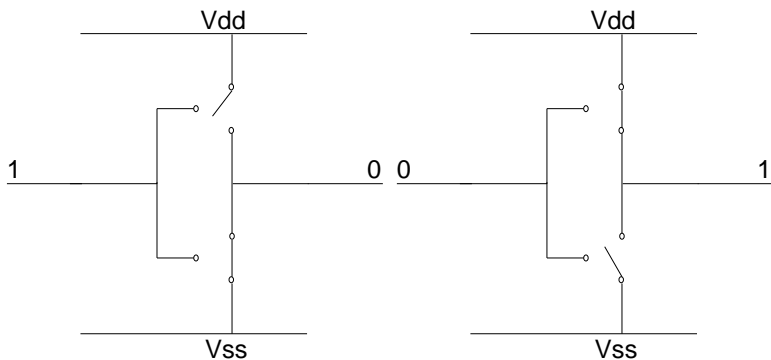
# Utilisation du CMOS - l'Inverseur

## Réalisation



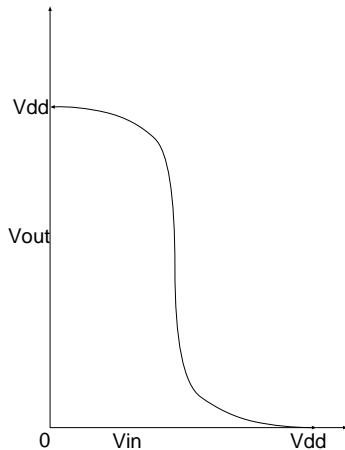
# Utilisation du CMOS - l'Inverseur

## Modélisation



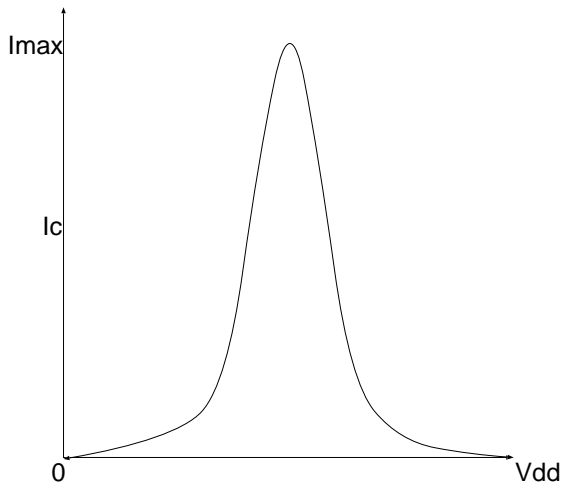
# Caractéristique d'un inverseur CMOS

## Courbe de Transfert

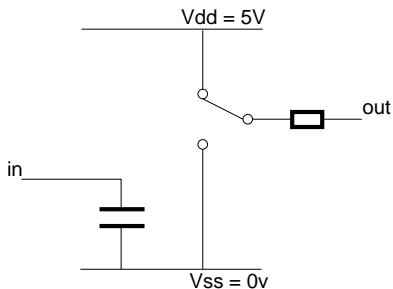


# Caractéristique d'un inverseur CMOS

Courant Consommé

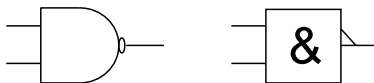


# Schéma Equivalent d'un inverseur CMOS



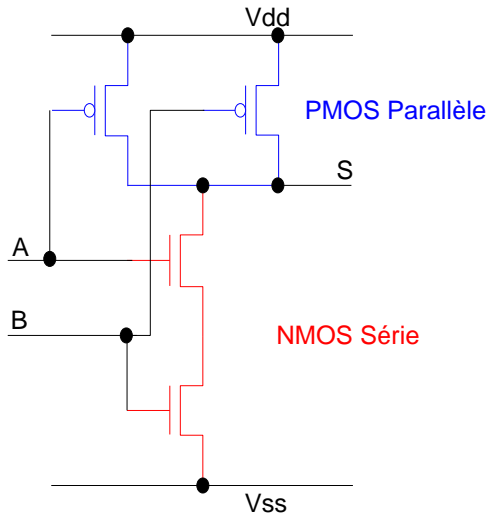
- Modèle cohérent jusqu'à 20 MHz
- Permet de modéliser la sortance d'une porte

# La complémentarité - La porte Non-ET

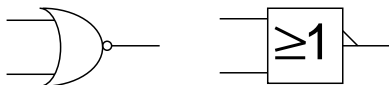




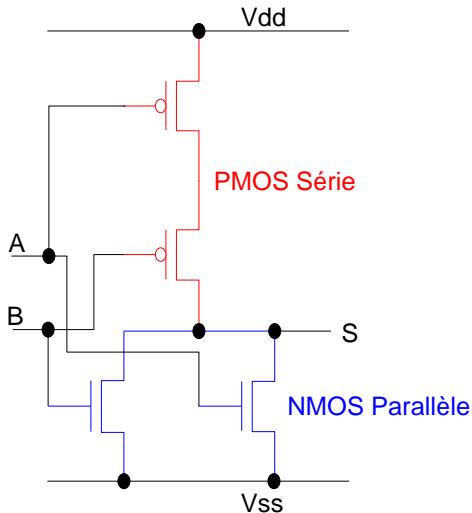
# La complémentarité - La porte Non-ET



# La complémentarité - La porte Non-OU



# La complémentarité - La porte Non-OU



# CMOS - limites ?

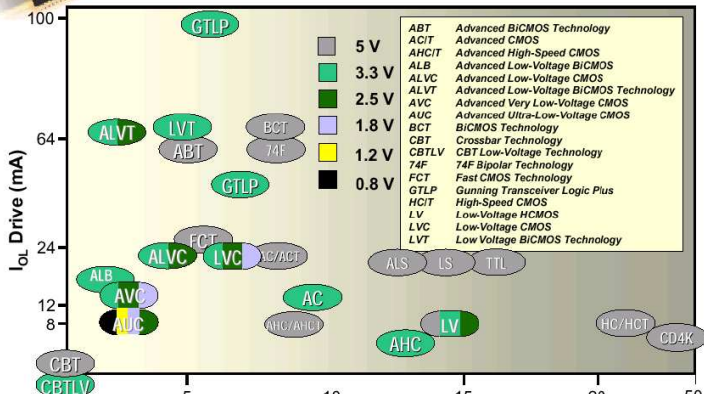
- $P \equiv V_{dd}^2 * C * F$  où  $C$  est la capacité totale du circuit
- Du à la commutation
- A technologie constante, le CMOS finit par consommer plus que le bipolaire si  $F$  ou  $C$  augmente
- Baisser la tension d'alimentation
- Modifier les procédés technologiques de fabrication de CI.

# Famille CMOS

- Plusieurs familles CMOS
- 74HC : Compatibilité TTL



## Family Performance Positioning

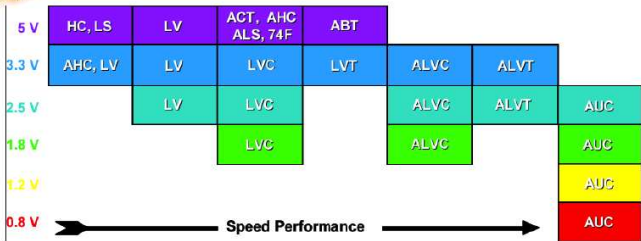


## Famille CMOS



# Low-Voltage Market

## Coverage and Standardization

**AHC**

- ✓ 8.5-ns  $t_{pd}$  (5 V)
- ✓ 13.5-ns  $t_{pd}$  (3.3 V)
- ✓ -8/8 mA (5 V)
- ✓ -4/4 mA (3.3 V)
- ✓ 5-V or 3.3-V  $V_{CC}$
- ✓ 5-V input tolerant
- ✓ 2 WW sources

**LV**

- ✓ 6.5-ns  $t_{pd}$  (5 V)
- ✓ 10-ns  $t_{pd}$  (3.3 V)
- ✓ -16/16 mA (5 V)
- ✓ -8/8 mA (3.3 V)
- ✓ 5-V input tolerant
- ✓ 3 WW sources
- ✓ Partial power down†

**LVC**

- ✓ 6.5-ns  $t_{pd}$
- ✓ -24/24 mA
- ✓ Ultra-low (20  $\mu$ A) standby power
- ✓ 4 WW sources
- ✓ Bus-hold option
- ✓ 5-V tolerant
- ✓ Gate functions

**LVT**

- ✓ 4-ns  $t_{pd}$
- ✓ -32/64 mA
- ✓ Low (90  $\mu$ A) standby power
- ✓ 4 WW sources
- ✓ Bus-hold option
- ✓ 5-V tolerant
- ✓ Hot insertion†

**ALVC**

- ✓ 3-ns  $t_{pd}$
- ✓ -24/24 mA
- ✓ Very low (40  $\mu$ A) standby power
- ✓ 5 WW sources
- ✓ Bus hold

**ALVT**

- ✓ 2.4-ns  $t_{pd}$
- ✓ -32/64 mA
- ✓ Low (90  $\mu$ A) standby power
- ✓ 3 WW sources
- ✓ Bus hold
- ✓ 5-V tolerant
- ✓ Hot insertion†

**AUC**

- ✓ 2-ns speed
- ✓ -8/8-mA drive
- ✓ Ultra-low (10  $\mu$ A) standby power
- ✓ 3 WW sources
- ✓ Bus-hold option
- ✓ 3.6-V tolerant

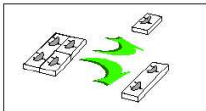
# Famille CMOS



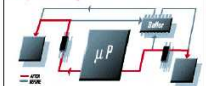
## Little Logic

Provide Leadership Portfolio and Packaging Solutions

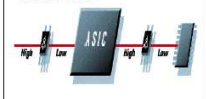
### The Principle



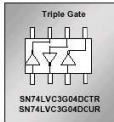
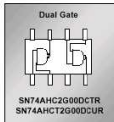
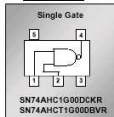
### Simplified routing



### Quick fixes for ASICs



### Example



### Easy Naming from TI

SN74	LVC	1G	00	YEA	R
------	-----	----	----	-----	---

SN74

Standard prefix  
74 = Commercial

LVC

Product Family  
AHC, AHCT, LVC, CBT, AUC

1G

1G – Single Gate  
2G – Dual Gate  
3G – Triple Gate

00

Logic Function

YEA

Package Type  
YEA = Nanostar™  
YZA = NanoFree™  
DCK = SC-70  
DBV = SOT-23  
DCU = US-8  
DCT = SM-8

R

Tape &amp; Reel



## Références

- Michel Hubin -  
[http://perso.wanadoo.fr/michel.hubin/physique/elec/chap\\_car](http://perso.wanadoo.fr/michel.hubin/physique/elec/chap_car)



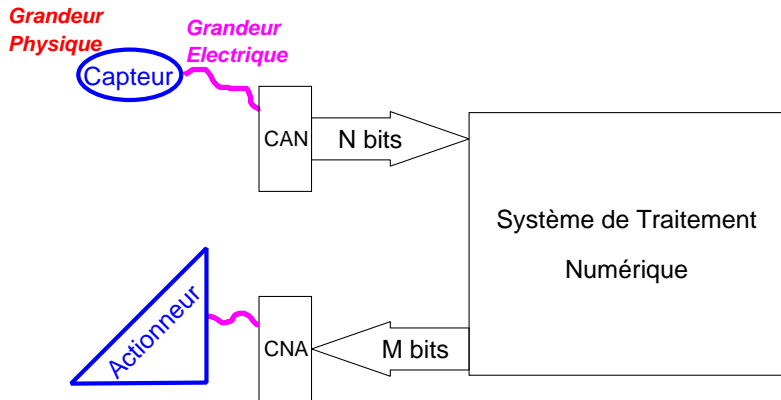
# Communication

- Le monde numérique est un monde discrétisé
- Le monde réel est un monde continu
- Le monde n'est pas numérique
- Comment Interfacer les 2 mondes ?
- A travers des Capteurs

# Communication

- Capteur = Fonction de conversion du *Monde* en grandeur électrique
- Capteur renvoie une Valeur Analogique
- Nécessité d'une fonction de conversion Analogique - Numérique
- Conversion Analogique Numérique : CAN
- Conversion Numérique Analogique : CNA

# Communication



# CAN : Définitions

La conversion analogique numérique consiste à transformer une grandeur électrique représentée par un signal en une grandeur numérique exprimée sur N bits après **échantillonnage** et **quantification** du signal. Cette valeur est une valeur **codée** représentant un nombre proportionnel à la grandeur électrique.

# CAN : Définitions

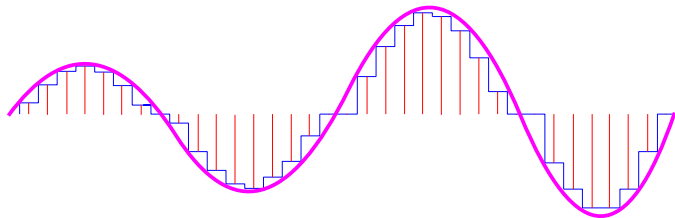
- Echantillonnage : prise périodique de valeur du signal, attention à Shannon  $F_e > 2 * F_{signal}$
- Quantification : association d'une mesure à la valeur échantillonnée, c'est une fonction de mémorisation.
- Pour le traitement Echantillonnage/Quantification on parle aussi d'échantillonnage-blocage ou d'échantillonnage-mémorisation.
- Codage : représentation de la valeur quantifiée dans un alphabet interprétable par un circuit numérique

# CAN : Définitions

*Signal continu*

*Signal échantillonné*

*Signal quantifié*



# CAN : Caractéristiques

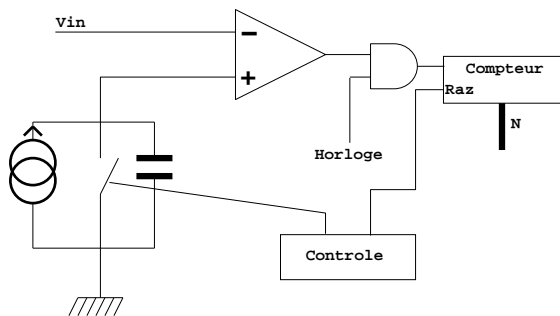
- Résolution : Amplitude de la plus petite variation. Correspond au LSB (Least Significant Bit)
- Temps de conversion : Temps de stabilisation de la donnée en sortie
- Erreur de Quantification : Incertitude du à la conversion
- Pleine Echelle : Etendue de la grandeur Analogique d'entrée

# CAN : Types

- Il existe différents types de conversion
- La conversion à rampe
- La conversion à double rampe
- La conversion à approximation successive
- La conversion Flash
- La conversion Sigma-Delta



# La conversion à rampe

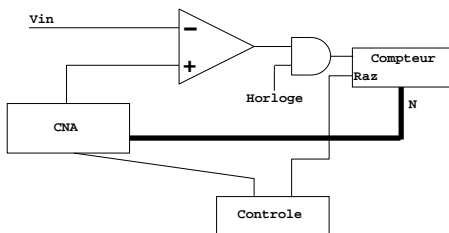


## La conversion à rampe

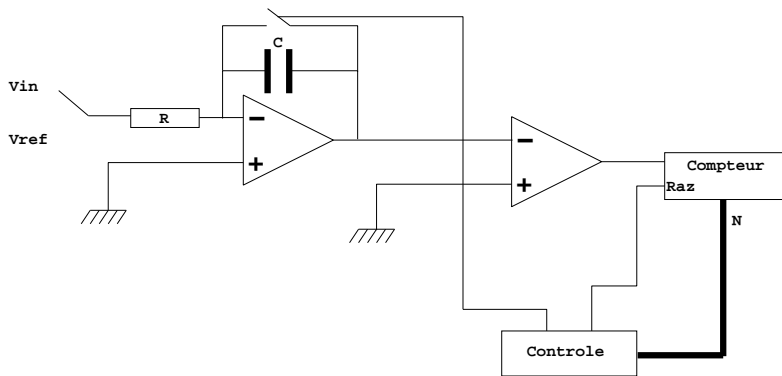
- Phase 1 :  $V_c$ , tension aux bornes de  $C$  mis à 0 ainsi que  $N$
- Phase 2 : Intégration aux bornes de  $C$ ,  $V_c = \frac{1}{C} \sum I dt = \frac{I}{C} t$   
tant que  $V_{in} > V_c$  le compteur est incrémenté
- $V_{in} = V_c$  le comparateur passe de 1 à 0 et bloque le compteur sur la valeur  $N$  correspondant au nombre binaire recherché

# La conversion à rampe numérique

Utilisation d'un CNA pour générer une rampe numérique.



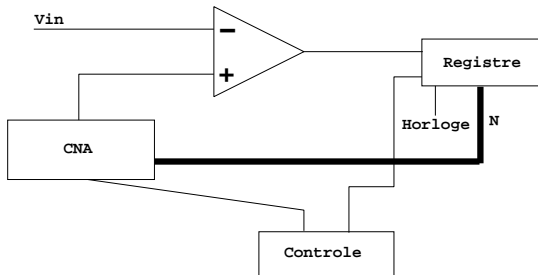
# La conversion double rampe



## La conversion double rampe

- Phase 1 : Connexion du montage sur  $V_{in}$ . Chargement de  $C$  pendant un temps fixe  $T_0$
- Phase 2 : Connexion du montage sur  $V_{ref}$ , de polarité inverse à  $V_{in}$ . Déchargement de  $C$  jusqu'à 0. Durant ce temps on incrémente le compteur jusqu'à  $N$ .  $N$  est la valeur binaire recherchée.

# La conversion à approximations successives



## La conversion à approximations successives

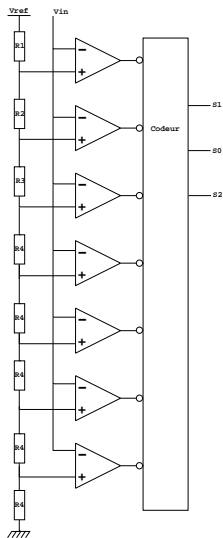
- Détermination des valeurs de bits de  $N$  les uns après les autres en commençant par le bit de poids fort
- On fixe le bit de poids fort à 1 et les autres à 0. Conversion NA du registre et comparaison à  $V_{in}$
- Si  $V_{in}$  est plus grand alors le bit reste à 1 sinon il passe à 0.
- On garde la valeur du bit de poids fort et on passe au bit suivant
- On répète le même traitement que précédemment pour ce bit et ainsi de suite jusqu'au bit de poids faible.

## La conversion à approximations successives

- Exemple : Convertisseur 8 bits,  $V_{ref}=10\text{ V}$
- Tension à convertir 6,92 V
- $10000000 = 5\text{V} < 6,92 \rightarrow B_7 = 1$
- $11000000 = 7,5\text{V} > 6,92 \rightarrow B_6 = 0$
- $10100000 = 6,25\text{V} < 6,92 \rightarrow B_5 = 1$
- $10110000 = 6,675\text{V} < 6,92 \rightarrow B_4 = 1$
- $10111000 = 7,1875\text{V} > 6,92 \rightarrow B_3 = 0$
- $10110100 = 7,03125\text{V} > 6,92 \rightarrow B_2 = 0$
- $10110010 = 6,95312\text{V} > 6,92 \rightarrow B_1 = 0$
- $10110001 = 6,91406\text{V} < 6,92 \rightarrow B_0 = 1$
- Valeur Numérique :10110001



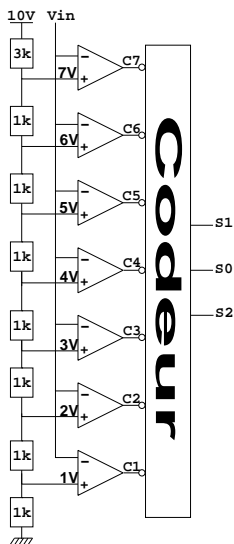
# La conversion Flash



# La conversion Flash

- Flash = Parallèle
- Principe : Comparer  $V_{in}$  à un ensemble de tensions prédéfinie
- Utiliser un codeur pour générer le nombre binaire

# La conversion Flash : Exemple



## La conversion Flash : Exemple

$V_{in}$	$C_1$	$C_2$	$C_3$	$C_4$	$C_5$	$C_6$	$C_7$	$S_2$	$S_1$	$S_0$
$< 1$	1	1	1	1	1	1	1	0	0	0
$>1, <2$	0	1	1	1	1	1	1	0	0	1
$>2, <3$	0	0	1	1	1	1	1	0	1	0
$>3, <4$	0	0	0	1	1	1	1	0	1	1
$>4, <5$	0	0	0	0	1	1	1	1	0	0
$>5, <6$	0	0	0	0	0	1	1	1	0	1
$>6, <7$	0	0	0	0	0	0	1	1	1	0
$>7$	0	0	0	0	0	0	0	1	1	1

# La conversion Sigma-Delta

Convertisseur Sigma-Delta : peut être vu comme un convertisseur double rampe en commutation continue pour maintenir la charge intégrée nulle en moyenne.

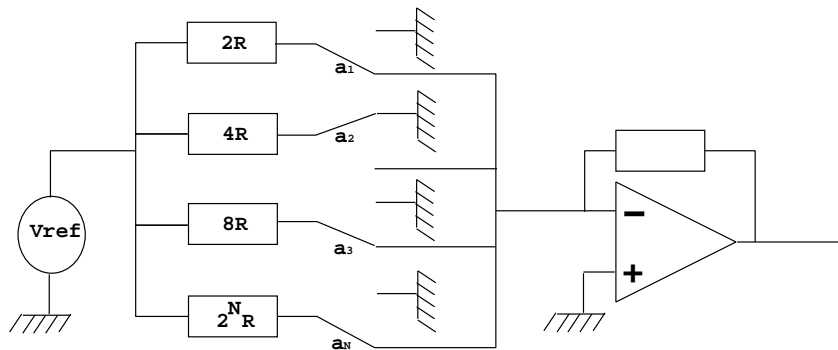
## CAN : Comparaison

Type	Vitesse	Erreur	Résolution
Simple Rampe	Faible (ms)	Elevée	Moyenne à élevée (7 à 14 bits)
Double Rampe	Faible (ms)	Faible	Elevée (10 à 18 bits)
Appro- -ximation	Moyenne ( $\approx 10\mu s$ )	Moyenne	Moyenne à élevée (8 à 6 bits)
Flash	Elevée (ns, $\mu s$ )	Moyenne	Faible à Moyenne (4 à 10 bits)

## CNA : Types

- Il existe différents type de Conversion Numérique Analogique
- Résistances Poids Proportionnels
- Réseau R2R

# CNA : Résistances Poids Proportionnels







# CNA : Comparaison

Type	Vitesse	Erreur	Résolution
Poids Pondérés	Elevée ( $\mu s$ )	Elevée	Faible
R2R	Elevée ( $\mu s$ )	Faible	Elevée