DIRO
IFT 1215

INTRODUCTION AUX SYSTÈMES INFORMATIQUES

CIRCUITS LOGIQUES

Max Mignotte

Département d'Informatique et de Recherche Opérationnelle
Http: //www.iro.umontreal.ca/~mignotte/
E-mail: mignotte@iro.umontreal.ca

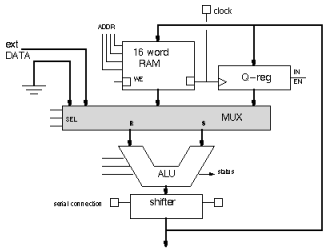
CIRCUITS LOGIQUES

SOMMAIRE

CIRCUITS LOGIQUES
INTRODUCTION

Introduction

• L'ordinateur est composé de circuits intégrés qui ont tous une fonction spécialisée (ex: ALU, mémoire, circuit décodant les instructions, etc.)



• Ces circuits sont fait à partir de circuits logiques dont le but est d'**exécuter** des opérations (fonctions) sur des variables logiques (i.e., binaires)

- 3 Types de Circuits Logiques -

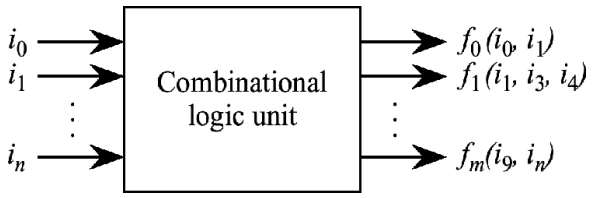
• Les circuits **combinatoires** ([*combinatorial logic*]) où les signaux de sortie ne dépendent que des signaux d'entrée (ex: additionneur, décaleur, etc.)

• Les circuits **séquentiels** ([*sequential logic*]) où les signaux de sortie dépendent des signaux d'entrée et des signaux d'entrée appliqués antérieurement (ex: mémoire, compteur, etc.)

• Les circuit à **état fini** ([*finite state machine*]) où les signaux de sortie dépendent des signaux d'entrée et de l'état interne de la machine (ex: distributeur automatique, etc.)

CIRCUITS LOGIQUES
LES CIRCUITS COMBINATOIRES

Les Circuits Combinatoires



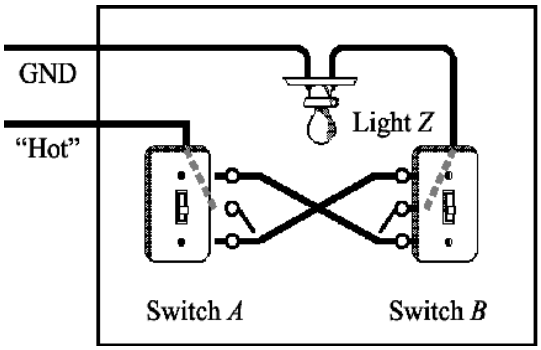
• **Entrées** et **sorties** d'un CLU ont deux valeurs possibles (1 et 0) (*tech. parlant*: 0 Volt et 5 Volts)

• Circuit idéalisé où le temps de propagation des signaux n'est pas pris en compte

Table de Vérité

• Développé par G. Boole en 1854 puis repris par C. Shannon (Bell Labs)

• La **Fonction Logique** d'un CLU peut se définir par le tableau de correspondance (ou Table de Vérité) entre les états d'entrée et les états de sortie



Inputs		Output
A	*B*	*Z*
0	0	0
0	1	1
1	0	1
1	1	0

CIRCUITS LOGIQUES
LES CIRCUITS COMBINATOIRES

• On pourrait construire un autre schéma électrique qui obéisse à une autre table de vérité comme celle-ci dessous

Inputs		Output
A	B	Z
0	0	1
0	1	0
1	0	0
1	1	1

• La table de vérité d'une fonction de n variables a autant de lignes que d'états d'entrée, soit 2^n

Combien de Table de Vérité possibles ?
Pour chacun des ces états, la sortie peut prendre deux valeurs possibles (i.e, "0" ou "1")

Ainsi pour n variables, on peut avoir 2^{2^n} fonctions possibles

- Pour 1 variable d'entrée, 4 fonctions possibles
- Pour 2 variable d'entrée, 16 fonctions possibles
- Pour 3 variable d'entrée, 256 fonctions possibles
- ...

• Toute fonction logique peut être réalisé à l'aide d'un petit nombre de fonctions logiques de base appelé **opérateur logique** ou **portes** [*gates*]

CIRCUITS LOGIQUES
LES CIRCUITS COMBINATOIRES

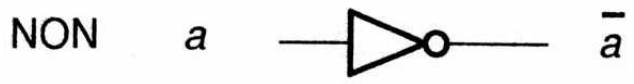
Fonction d'une Variable

- 1 variable ▷ 2 états
- 2^2 Fonctions d'une variable

Fonctions logiques d'une variable a

a	Z_0	Z_1	Z_2	Z_3		
					$Z_0 = 0$	constante
					$Z_1 = a$	identité
0	0	0	1	1	$Z_2 = \bar{a}$	complémentation
1	0	1	0	1	$Z_3 = 1$	constante

- La seule fonction non triviale est la fonction Z_2, dite de complémentation, qui est réalisée par l'opérateur **NON** ou **inverseur** ($Z = \overline{a}$)



CIRCUITS LOGIQUES
LES CIRCUITS COMBINATOIRES

Fonction d'une Variable

• 2 variables ▷ 4 états et 2^4 Fonctions

Fonctions logiques de 2 variables a et b

00	01	10	11	ab	
0	0	0	0	$F_0 = 0$	(constante nulle)
0	0	0	1	$F_1 = ab$	(fonction ET)
0	0	1	0	$F_2 = a\bar{b}$	
0	0	1	1	$F_3 = a$	
0	1	0	0	$F_4 = \bar{a}b$	
0	1	0	1	$F_5 = b$	
0	1	1	0	$F_6 = a \oplus b$	(fonction XOR)
0	1	1	1	$F_7 = a + b$	(fonction OU)
1	0	0	0	$F_8 = \overline{a+b} = \bar{a}\ \bar{b}$	(fonction NOR)
1	0	0	1	$F_9 = \overline{a \oplus b}$	
1	0	1	0	$F_{10} = \bar{b}$	
1	0	1	1	$F_{11} = a + \bar{b}$	
1	1	0	0	$F_{12} = \bar{a}$	
1	1	0	1	$F_{13} = \bar{a} + b$	
1	1	1	0	$F_{14} = \overline{ab} = \bar{a} + \bar{b}$	(fonction NAND)
1	1	1	1	$F_{15} = 1$	(constante 1)

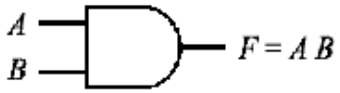
• Quatre de ces fonctions jouent un rôle important, ce sont les fonctions

ET [*AND*]
NON-ET [*NAND*]
OU [*OR*]
NON-OU [*NOR*]

CIRCUITS LOGIQUES
LES PORTES LOGIQUES

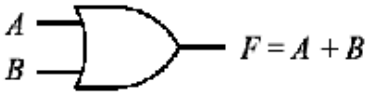
Porte Logique et leur Symbole (1)

A	B	F
0	0	0
0	1	0
1	0	0
1	1	1



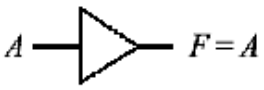
AND

A	B	F
0	0	0
0	1	1
1	0	1
1	1	1



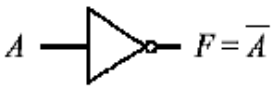
OR

A	F
0	0
1	1



Buffer

A	F
0	1
1	0

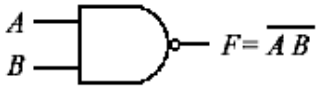


NOT (Inverter)

CIRCUITS LOGIQUES
LES PORTES LOGIQUES

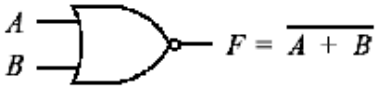
Porte Logique et leur Symbole (2)

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



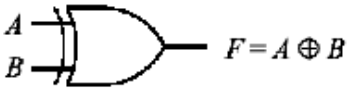
NAND

A	B	F
0	0	1
0	1	0
1	0	0
1	1	0



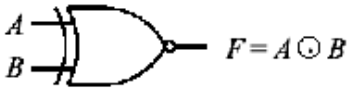
NOR

A	B	F
0	0	0
0	1	1
1	0	1
1	1	0



Exclusive-OR (XOR)

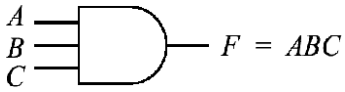
A	B	F
0	0	1
0	1	0
1	0	0
1	1	1



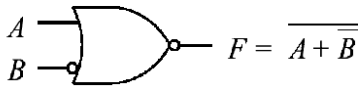
Exclusive-NOR (XNOR)

CIRCUITS LOGIQUES
LES PORTES LOGIQUES

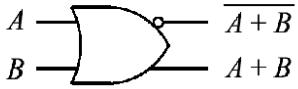
Porte Logique et leur Symbole -Variantes-



(a)



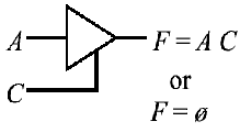
(b)



(c)

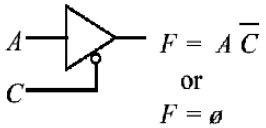
(a) 3 inputs **(b) A Negated input** **(c) Complementary outputs**

C	A	F
0	0	ø
0	1	ø
1	0	0
1	1	1



Tri-state buffer

C	A	F
0	0	0
0	1	1
1	0	ø
1	1	ø



Tri-state buffer, inverted control

CIRCUITS LOGIQUES
THÉORÈME DE L'ALGÈBRE DE BOOLE

Théorèmes Fondamentaux de L'Algèbre de Boole

	Relationship	Dual	Property
Postulates	$A\,B = B\,A$	$A + B = B + A$	Commutative
	$A\,(B + C) = A\,B + A\,C$	$A + B\,C = (A + B)\,(A + C)$	Distributive
	$1\,A = A$	$0 + A = A$	Identity
	$A\,\overline{A} = 0$	$A + \overline{A} = 1$	Complement
Theorems	$0\,A = 0$	$1 + A = 1$	Zero and one theorems
	$A\,A = A$	$A + A = A$	Idempotence
	$A\,(B\,C) = (A\,B)\,C$	$A + (B + C) = (A + B) + C$	Associative
	$\overline{\overline{A}} = A$		Involution
	$\overline{A\,B} = \overline{A} + \overline{B}$	$\overline{A + B} = \overline{A}\,\overline{B}$	DeMorgan's Theorem
	$AB + \overline{A}C + BC = AB + \overline{A}C$	$(A + B)(\overline{A} + C)(B + C) = (A + B)(\overline{A} + C)$	Consensus Theorem
	$A\,(A + B) = A$	$A + A\,B = A$	Absorption Theorem

Principe de Dualité : *Le* dual *d'une fonction Booléenne est obtenue en remplaçant AND (.) avec OU (+) et inversement et (***1***) avec (***0***) et inversement*

CIRCUITS LOGIQUES
L'OPÉRATEUR OU EXCLUSIF

L'Opérateur OU Exclusif

Table de vérité du XOR ($Z = a \oplus b$)

a	b	$a \oplus b$
0	0	0
0	1	1
1	0	1
1	1	0

$a \oplus b = \bar{a}b + a\bar{b}$

$\overline{a \oplus b} = ab + \bar{a}\,\bar{b}$

$a \oplus 0 = a$

$a \oplus a = 0$

$a \oplus 1 = \bar{a}$

$a \oplus \bar{a} = 1$

$a \oplus b = b \oplus a$

$(a \oplus b) \oplus c = a \oplus (b \oplus c)$

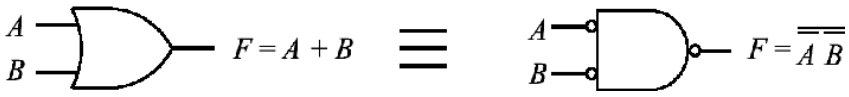
$a \oplus b = \overline{\bar{a}\,\bar{b} + ab} = (a + b)\,(\bar{a} + \bar{b}) = (a + b)\overline{ab}$

CIRCUITS LOGIQUES
THÉORÈME DE DeMORGAN

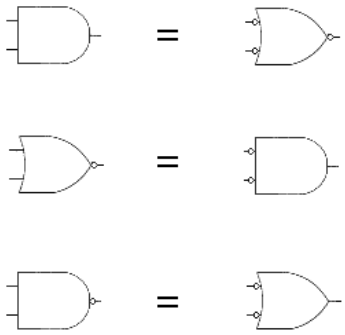
Théorème de DeMorgan

A B	$\overline{A\,B} = \overline{A} + \overline{B}$		$\overline{A + B} = \overline{A}\,\overline{B}$	
0 0	1	1	1	1
0 1	1	1	0	0
1 0	1	1	0	0
1 1	0	0	0	0

DeMorgan's theorem: $A + B = \overline{\overline{A + B}} = \overline{\overline{A}\,\overline{B}}$



De même

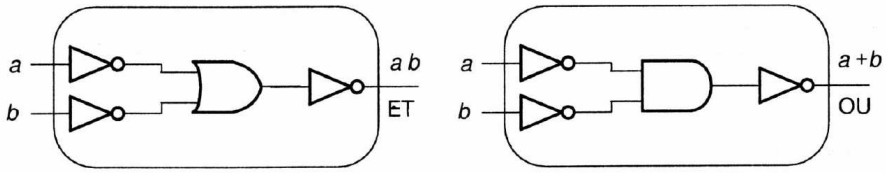


CIRCUITS LOGIQUES
OPÉRATEURS COMPLETS

Opérateurs Complets

L'ensemble [**ET, OU, NON**] est complet
car il permet de synthétiser toute fonction logique

L'ensemble [**ET, OU, NON**] n'est pas minimal
car il est possible de réaliser la fonction ET
avec des OU et des NON



Exemple de réalisation des opérateurs ET et OU

CIRCUITS LOGIQUES
OPÉRATEURS COMPLETS

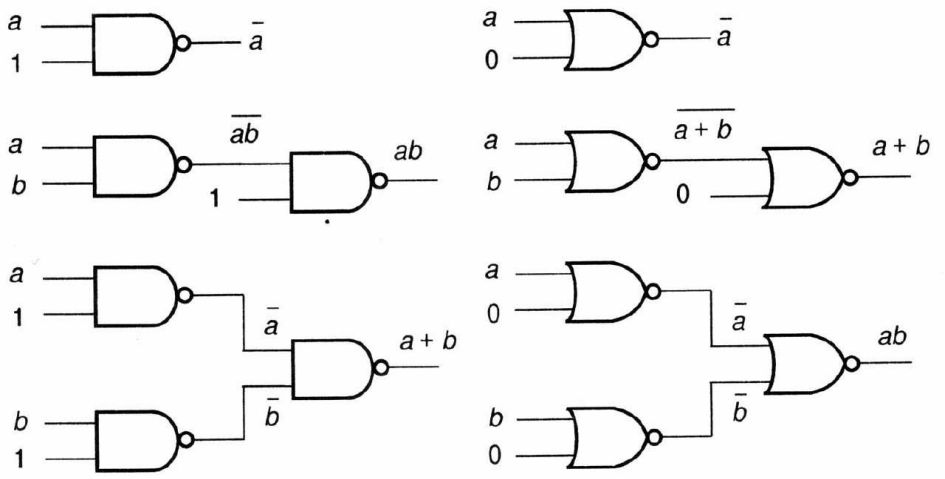
Opérateurs Complets

L'ensemble [**NON-ET (NAND)**] ou [**NON-OU (NOR)**]
est complet et minimal

Table de vérité du NAND et du NOR

NAND ($Z = \overline{ab} = \bar{a} + \bar{b}$) **NOR** ($Z = \overline{a+b} = \bar{a}\ \bar{b}$)

a	b	a NAND b	a NOR b
0	0	1	1
0	1	1	0
1	0	1	0
1	1	0	0



Réalisation de ET, OU, NON avec des NAND et des NOR

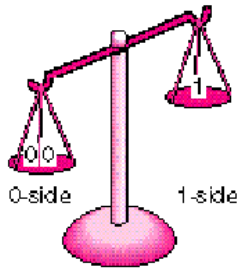
CIRCUITS LOGIQUES
SYNTHÈSE D'UN CIRCUIT COMBINATOIRE (SOP)

Exemple: Fonction Majorité

But: Synthèse d'un circuit combinatoire réalisant une fonction logique particulière

1– Réaliser le tableau de vérité de la fonction logique

Minterm Index	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1



A balance tips to the left or right depending on whether there are more 0's or 1's.

2– En dériver une expression algébrique

$$\begin{aligned} F = M &= CB\overline{A} + C\overline{B}A + \overline{C}BA + CBA \\ &= m_3 + m_5 + m_6 + m_7 \\ &= \sum(3,5,6,7) \end{aligned}$$

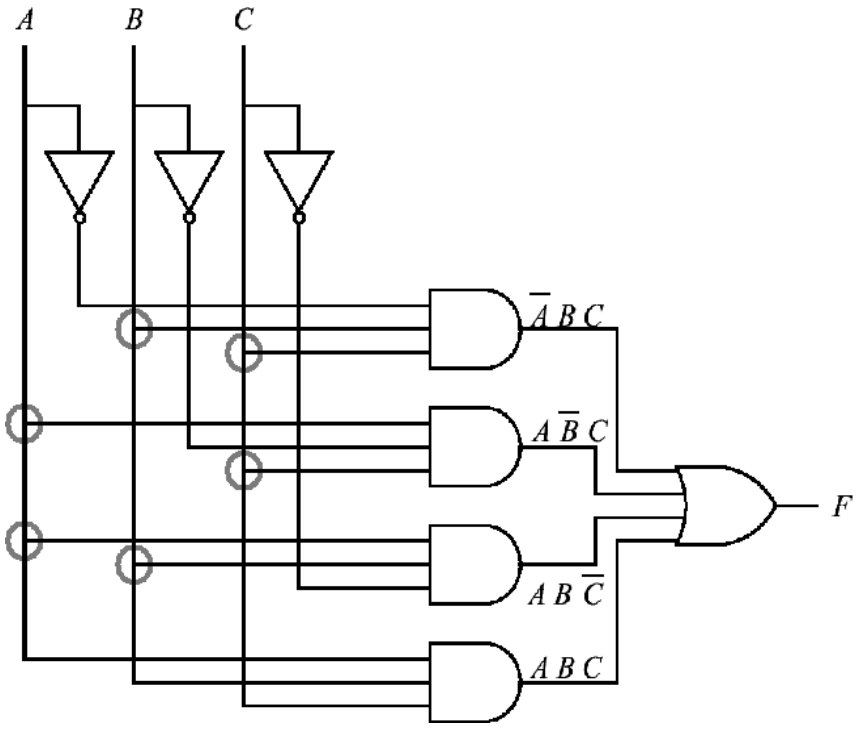
• On réalise une Somme de Produit [*Sum Of Product*] ou **SOP** composée de *minterms*, allant de 0 à $2^{n=3} - 1$ (exprimant sa valeur Booléenne)

CIRCUITS LOGIQUES
SYNTHÈSE D'UN CIRCUIT COMBINATOIRE (SOP)

3– Simplifier cette expression en la transformant en une expression équivalente plus simple (i.e., contenant un nombre minimal de terme)

▷ Tableau de Karnaugh, théo. de l'algèbre de Boole

4– Réaliser la fonction logique à l'aide d'opérateur. Dans le cas d'un **SOP** ; à l'aide de [**AND**,**OR**,**NOT**]

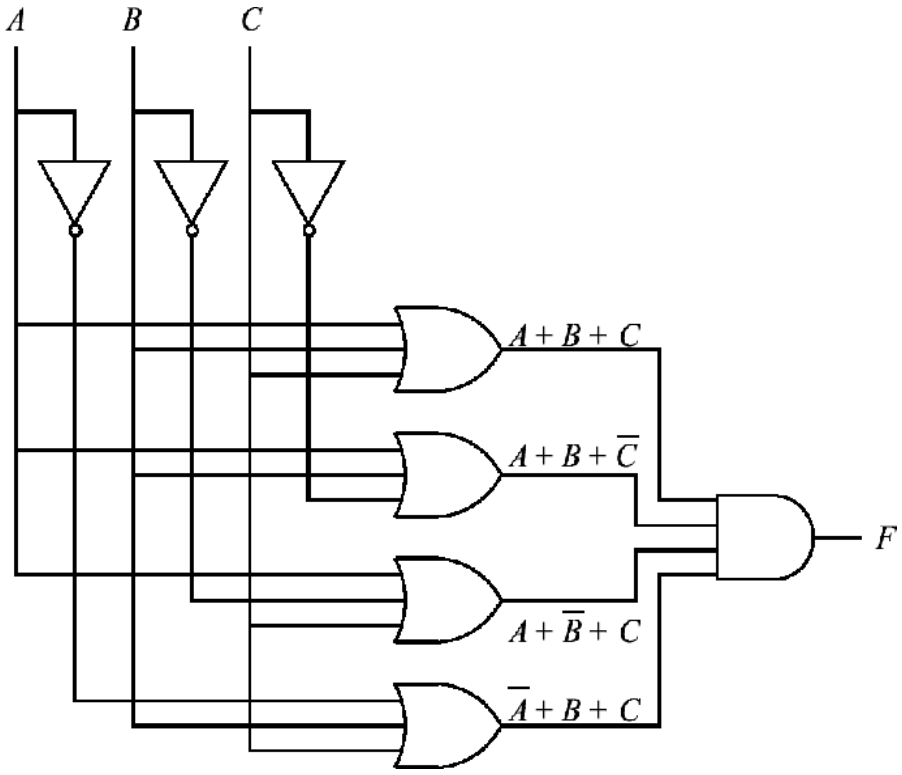


▷ Réalisation de la Fonction Majorité à l'aide de 8 portes et 19 entrés

CIRCUITS LOGIQUES
SYNTHÈSE D'UN CIRCUIT COMBINATOIRE (POS)

On peut réaliser un produit de somme [*Product Of Sum*] ou **POS** et utiliser des *maxterms* en écrivant

$$\begin{aligned}
\overline{F} = \overline{M} &= \overline{C}\,\overline{B}\,\overline{A} + C\overline{B}\,\overline{A} + \overline{C}B\overline{A} + \overline{C}\,\overline{B}A \\
M &= \overline{\overline{C}\,\overline{B}\,\overline{A} + C\overline{B}\,\overline{A} + \overline{C}B\overline{A} + \overline{C}\,\overline{B}A} \\
&= (\overline{\overline{C}\,\overline{B}\,\overline{A}}) \,.\, \overline{(C\overline{B}\,\overline{A})} \,.\, \overline{(\overline{C}B\overline{A})} \,.\, \overline{(\overline{C}\,\overline{B}A)} \\
&= (C + B + A) \,.\, (\overline{C} + B + A) \,.\, (C + \overline{B} + A) \,.\, (C + B + \overline{A})
\end{aligned}$$



▷ Réalisation de la Fonction Majorité à l'aide de 8 portes et 19 entrés

CIRCUITS LOGIQUES
SYNTHÈSE D'UN CIRCUIT COMBINATOIRE

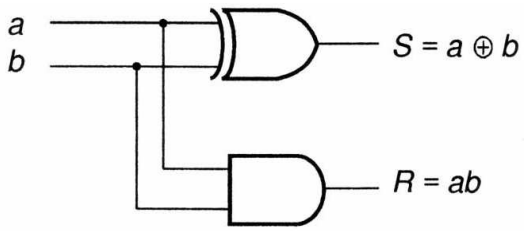
Exemple: Additionneur Binaire

Table de vérité du demi-additionneur

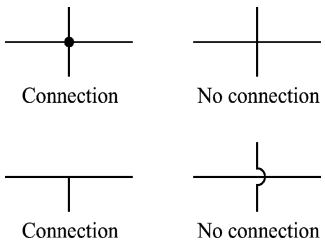
a	b	S	R	
0	0	0	0	S = Somme
0	1	1	0	R = Retenue
1	0	1	0	
1	1	0	1	

$$S = B\overline{A} + A\overline{B} = A \oplus B$$
$$R = AB$$

Logigramme



Nota



CIRCUITS LOGIQUES
SIMPLIFICATION D'EXPRESSION BOOLÉENNE

Simplification par Méthode Algébrique

On peut appliquer la méthode Algébrique pour réduire la fonction majorité à son expression minimal

$F = \overline{A}BC + A\overline{B}C + AB\overline{C} + ABC$

$F = \overline{A}BC + A\overline{B}C + AB(\overline{C} + C)$ Distributive Property

$F = \overline{A}BC + A\overline{B}C + AB(1)$ Complement Property

$F = \overline{A}BC + A\overline{B}C + AB$ Identity Property

$F = \overline{A}BC + A\overline{B}C + AB + ABC$ Idempotence

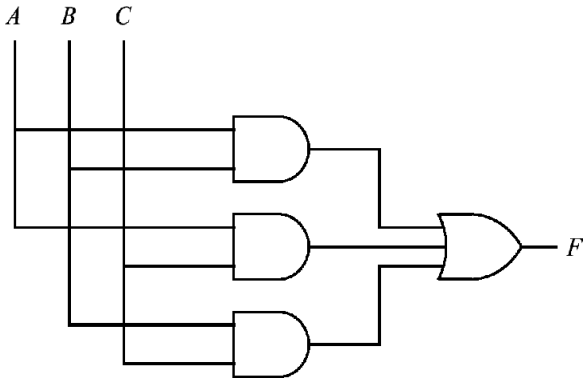
$F = \overline{A}BC + AC(\overline{B} + B) + AB$ Identity Property

$F = \overline{A}BC + AC + AB$ Complement and Identity

$F = \overline{A}BC + AC + AB + ABC$ Idempotence

$F = BC(\overline{A} + A) + AC + AB$ Distributive

$F = BC + AC + AB$ Complement and Identity



CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 2 VARIABLES

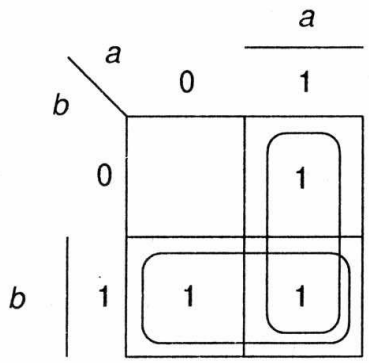
Simplification par Tableau de Karnaugh (2 Variables)

Fonction Z

$$Z = B\overline{A} + A\overline{B} + BA$$

• Pour remplir la table de Karnaugh à partir de la table de vérité, on attribut la valeur "1" aux cases correspondantes aux états d'entrée où la fonction est vraie

a	b	Z
0	0	0
0	1	1
1	0	1
1	1	1



• La méthode de simplification consiste à encercler tout ensemble de cases occupées adjacentes sur la même ligne ou sur la même colonne. Les recouvrements sont permis

Simplification

$$Z = A + B$$

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 3 VARIABLES

Simplification par Tableau de Karnaugh (3 Variables)

Fonction Majorité

$$M = CB\overline{A} + C\overline{B}A + \overline{C}BA + CBA$$

Place un "1" dans la cellule associé au minterme

Minterm Index	A	B	C	F
0	0	0	0	0
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	0
5	1	0	1	1
6	1	1	0	1
7	1	1	1	1

C \ AB	00	01	11	10
0			1	
1		1	1	1

Simplification

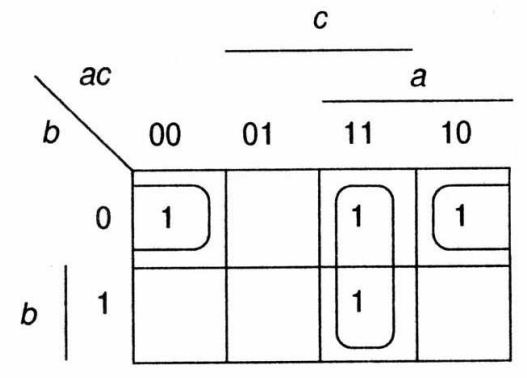
C \ AB	00	01	11	10
0			1	
1	1	1	1	

$$M = CB + CA + BA$$

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 3 VARIABLES

Fonction Z

$$Z = \overline{C}\,\overline{B}\,\overline{A} + C\overline{B}A + \overline{C}\,\overline{B}A + CBA$$



Simplification

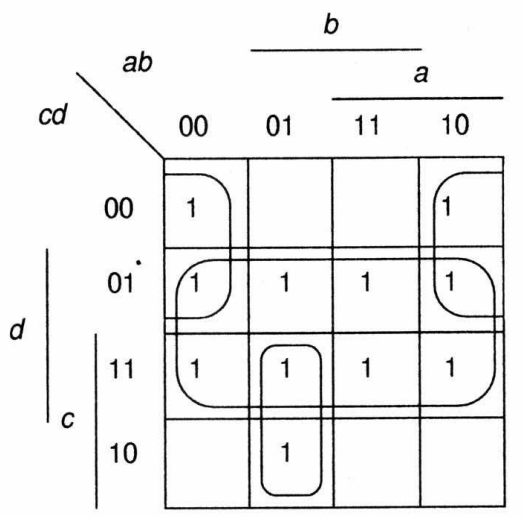
$$Z = \ldots$$

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 4 VARIABLES

Simplification par Tableau de Karnaugh (4 Variables)

Fonction Z

$Z(a,b,c,d) = \bar{a}\ \bar{b}\ \bar{c}\ \bar{d} + \bar{a}\ \bar{b}\ \bar{c}\ d + \bar{a}\ \bar{b}\ c\ d + \bar{a}\ b\ \bar{c}\ d + a\ b\ \bar{c}\ d + a\ \bar{b}\ \bar{c}\ d + a\ \bar{b}\ \bar{c}\ \bar{d} + \bar{a}\ b\ c\ d + a\ b\ c\ d + a\ \bar{b}\ c\ d + \bar{a}\ b\ c\ \bar{d}.$



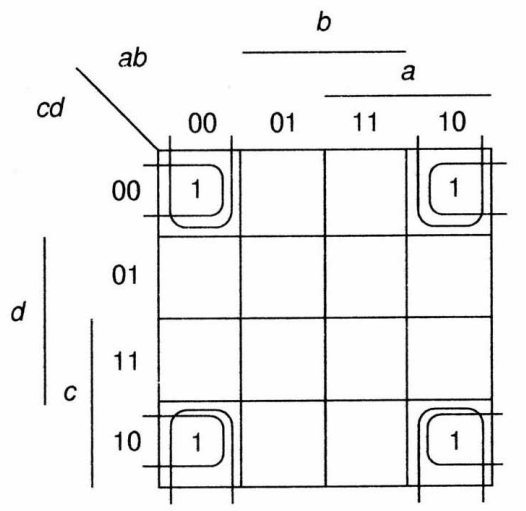
Simplification

$Z \ =$

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 4 VARIABLES

Fonction Z

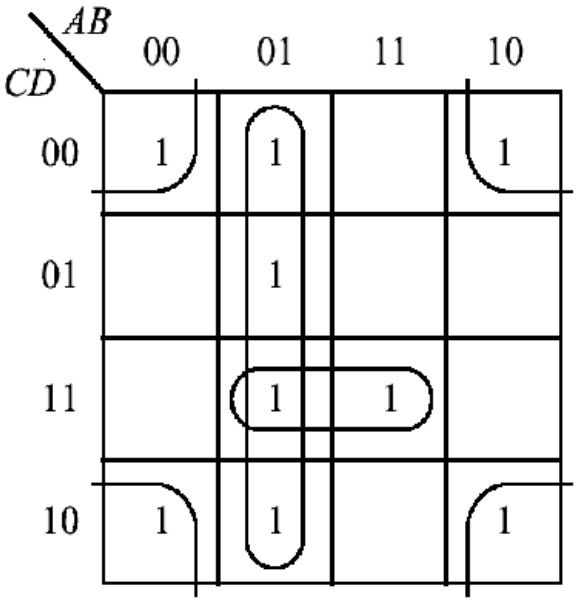
$$Z(a,b,c,d) = \overline{a}\ \overline{b}\ \overline{c}\ \overline{d} + a\ \overline{b}\ \overline{c}\ \overline{d} + \overline{a}\ \overline{b}\ c\ \overline{d} + a\ \overline{b}\ c\ \overline{d}$$



Simplification

$$Z = \overline{b}\,\overline{d}$$

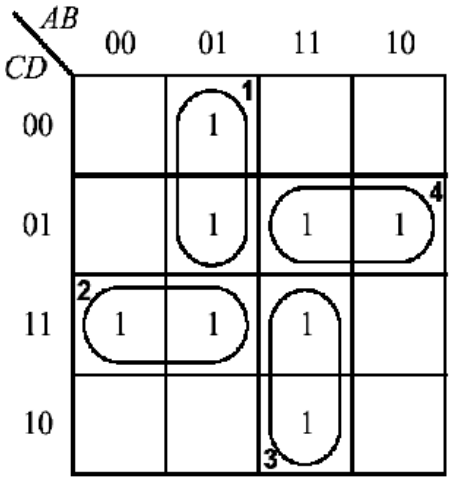
CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 4 VARIABLES



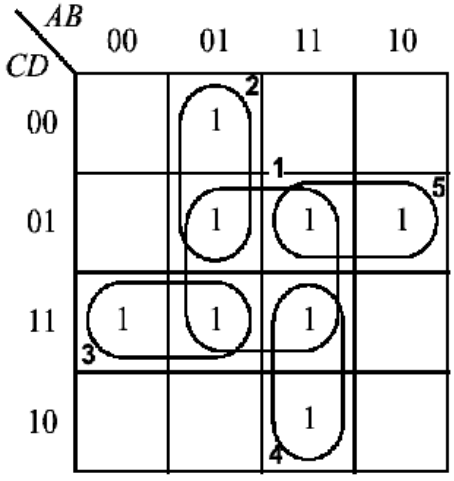
$$F = B\,C\,D + \overline{B}\,\overline{D} + \overline{A}\,B$$

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 4 VARIABLES

Groupement minimal et non-minimal



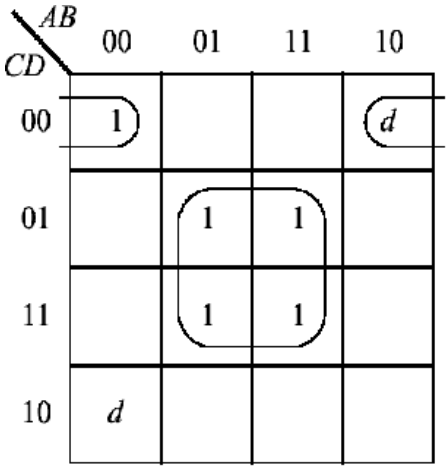
$$F = \overline{A}\,B\,\overline{C} + \overline{A}\,C\,D + A\,B\,C + A\,\overline{C}\,D$$



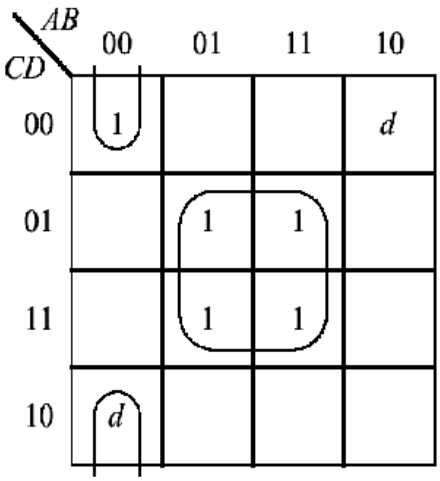
$$F = B\,D + \overline{A}\,B\,\overline{C} + \overline{A}\,C\,D + A\,B\,C + A\,\overline{C}\,D$$

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH ET D(ON'T CARE)

Tableau de Karnaugh et D(on't care)



$F = \overline{B}\,\overline{C}\,\overline{D} + B\,D$

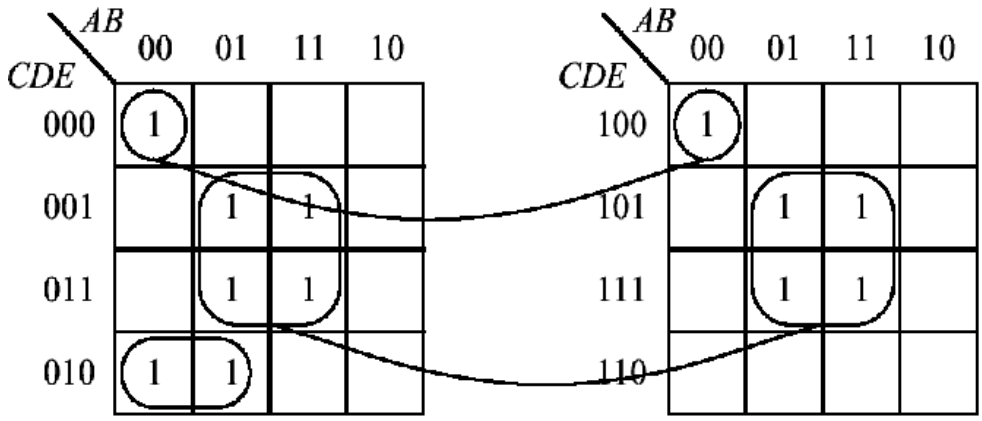


$F = \overline{A}\,\overline{B}\,\overline{D} + B\,D$

• Lorsqu'une variable peut être indifféremment un ¨1" ou un "0" symbolisé par un d (*"don't care"*), il peut y avoir plus d'un groupement minimal

CIRCUITS LOGIQUES
TABLEAU DE KARNAUGH 5 VARIABLES

Simplification par Tableau de Karnaugh (5 Variables)



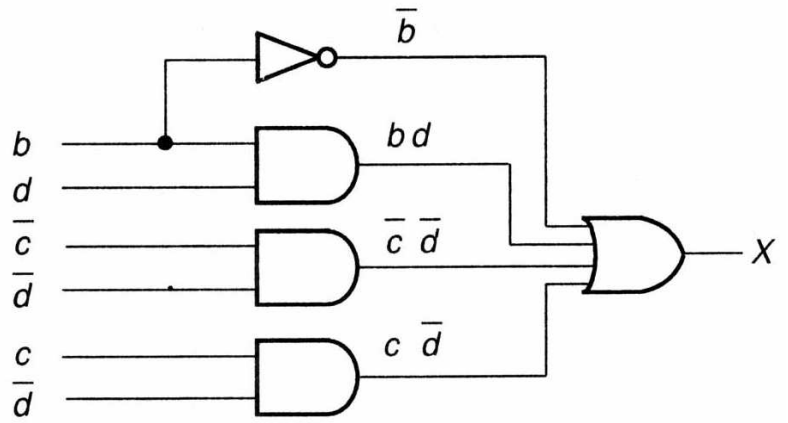
$$F = \overline{A}\,\overline{C}\,D\overline{E} + \overline{A}\,\overline{B}\,\overline{D}\overline{E} + B\,E$$

CIRCUITS LOGIQUES
ANALYSE D'UN CIRCUIT COMBINATOIRE

Analyse d'un circuit combinatoire

But: Consiste à retrouver la fonction d'un circuit dont on connaît uniquement le logigramme
Cette fonction est unique

1– En procédant des entrées vers les sorties, donner, pour chaque opérateur l'expression de sa sortie en fonction des entrées, jusqu'à obtention d'une expression pour chaque fonction (sortie) réalisée par le circuit



Expression de la Fonction X

$$X = \overline{b} + bd + \overline{c}\,\overline{d} + c\,\overline{d}$$

On peut simplifier

$$X = \underbrace{\overline{b} + bd}_{\overline{b}+d} + \overline{d}(c + \overline{c})$$

$$X = 1$$

CIRCUITS LOGIQUES
ANALYSE D'UN CIRCUIT COMBINATOIRE

2– Donner la table de vérité correspondante

b	d	$\overline{b}$	bd	$\overline{d}$	X
0	0	1	0	1	1
0	1	1	0	0	1
1	0	0	0	1	1
1	1	0	1	0	1

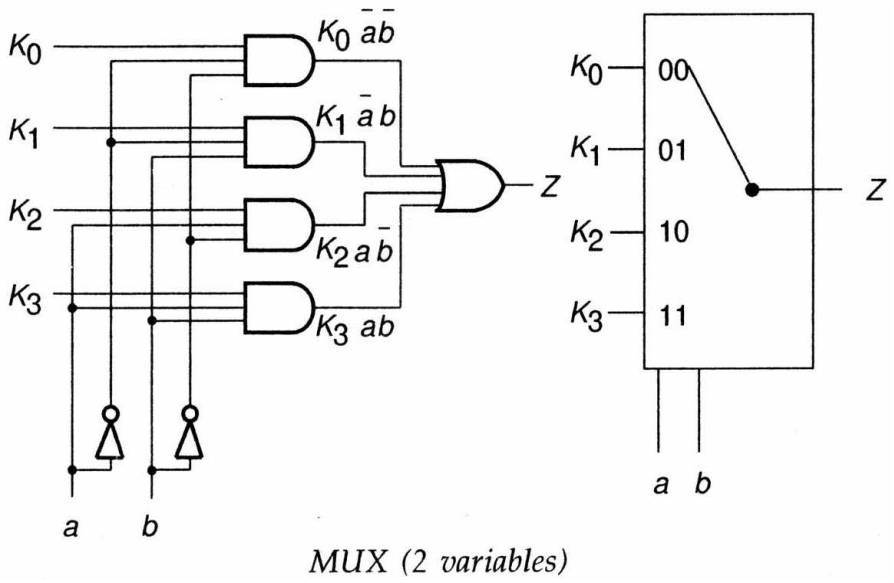
3– En déduire le rôle du circuit

Dans notre exemple, c'est un circuit pour lequel il y a toujours "X=1" à la sortie

CIRCUITS LOGIQUES
MULTIPLEXEURS ET DÉMULTIPLEXEURS

Multiplexeurs

On appelle Multiplexeur (MUX) tout système combinatoire regroupant en série sur une voie les signaux venant de n voies en parallèle



MUX (2 variables)

$$Z(a,b) = K_0 \bar{a}\ \bar{b} + K_1 \bar{a}\ b + K_2 a\ \bar{b} + K_3 a\ b$$

Table de Vérité

CIRCUITS LOGIQUES
MULTIPLEXEURS ET DÉMULTIPLEXEURS

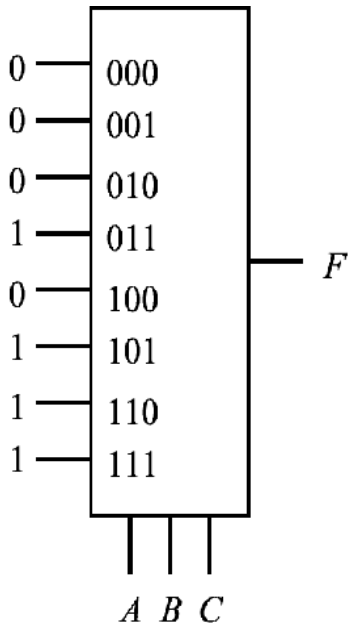
Implémentation de la fonction Majorité avec un $MUX_{8\times1}$

• Le multiplexeur peut générer une fonction booléenne si on utilise ses entrées de contrôle pour sélectionner (une à la fois) les 8 données d'entrée

Fonction Majorité

$$F = M \quad = \quad CB\overline{A} + C\overline{B}A + \overline{C}BA + CBA$$

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

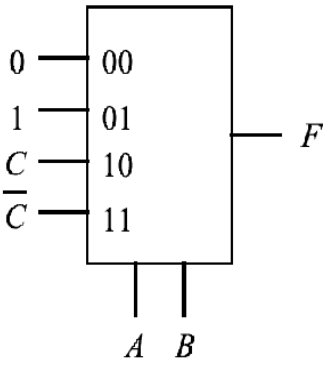


CIRCUITS LOGIQUES
MULTIPLEXEURS ET DÉMULTIPLEXEURS

Implémentation de la fonction F avec un $MUX_{4\times1}$

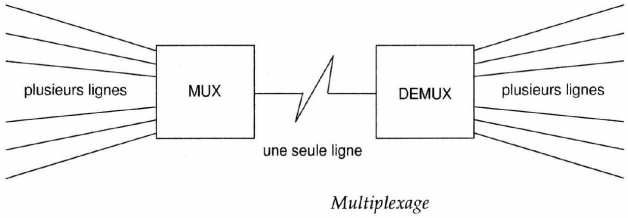
• **Principe** : On utilise les 2 entrées de contrôles de ce $MUX_{4\times1}$ pour sélectionner une paire de minterm. Les valeurs appliquées aux entrées du MUX sont soit $[0, 1, C, \overline{C}]$ pour réaliser le comportement désiré de chaque pair de minterm

A	B	C	F	
0	0	0	0	0
0	0	1	0	
0	1	0	1	1
0	1	1	1	
1	0	0	0	C
1	0	1	1	
1	1	0	1	$\overline{C}$
1	1	1	0	



Nota

Utilisé pour la transmission/conversion parallèle-série

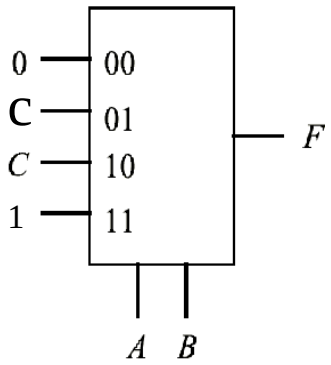


Multiplexage

CIRCUITS LOGIQUES
MULTIPLEXEURS ET DÉMULTIPLEXEURS

Implémentation de la fonction Majorité avec un $MUX_{4\times1}$

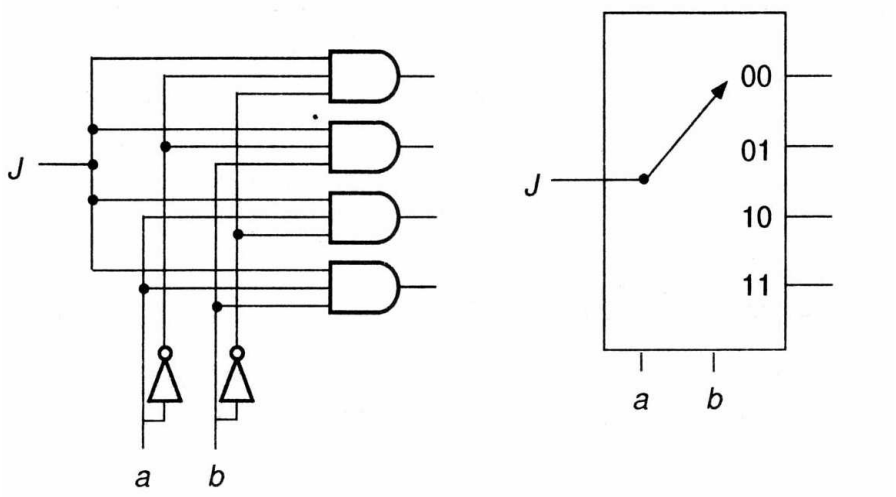
A	B	C	F	
0	0	0	0	0
0	0	1	0	
0	1	0	0	C
0	1	1	1	
1	0	0	0	C
1	0	1	1	
1	1	0	1	1
1	1	1	1	



CIRCUITS LOGIQUES
MULTIPLEXEURS ET DÉMULTIPLEXEURS

DeMultiplexeurs

On appelle DeMultiplexeur (DEMUX) tout système combinatoire regroupant en parallèle sur plusieurs voies les signaux venant d'une voie en série



DEMUX (2 variables)

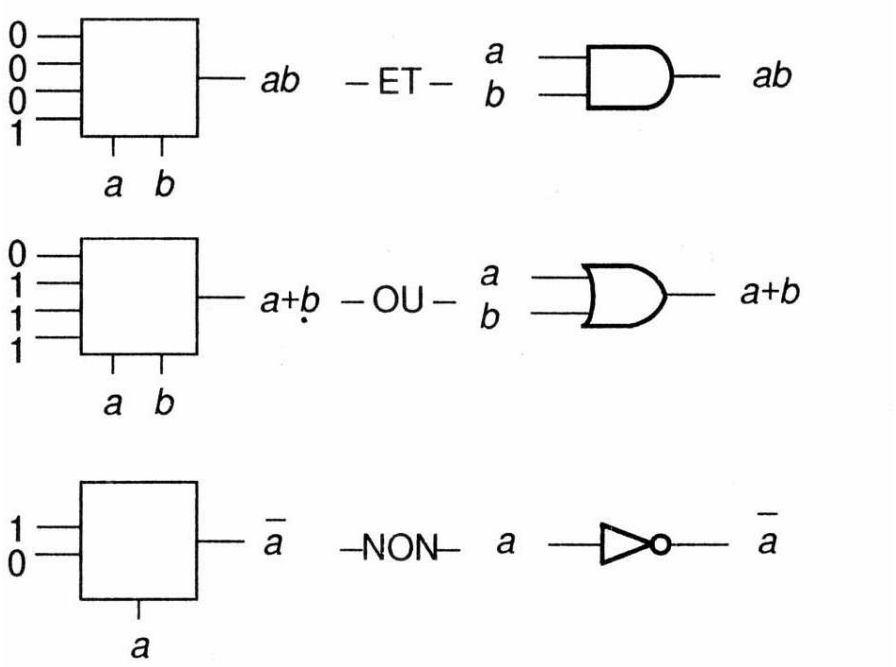
$$F_0 = J\,\overline{A}\,\overline{B} \qquad F_1 = J\,\overline{A}\,B$$

$$F_2 = J\,A\,\overline{B} \qquad F_3 = J\,A\,B$$

Table de Vérité

J	A	B	$\|\|$	F_0	F_1	F_2	F_3

CIRCUITS LOGIQUES
MULTIPLEXEURS ET DEMULTIPLEXEURS

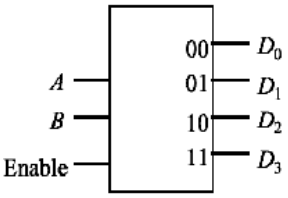


Circuits combinatoires (opérateurs de base)

CIRCUITS LOGIQUES
DECODEUR, ENCODEUR, TRANSCODEUR

DeCodeur

On appelle DeCodeur tout système qui fait correspondre à un code en entrée (sur n lignes) une seule sortie active (= 1) parmi les 2^n sorties possibles

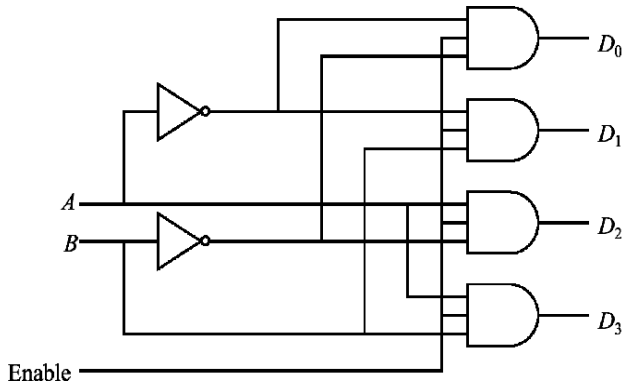


Enable = 1						
A	B	D_0	D_1	D_2	D_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Enable = 0						
A	B	D_0	D_1	D_2	D_3
0	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0
1	1	0	0	0	0

$D_0 = \overline{A}\,\overline{B}$ $D_1 = \overline{A}\,B$ $D_2 = A\,\overline{B}$ $D_3 = A\,B$

Le DeMultiplexeur est un décodeur avec une entrée validation [*enable*]



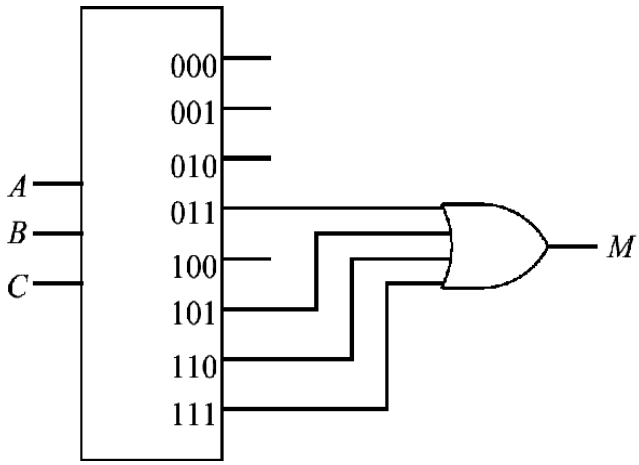
CIRCUITS LOGIQUES
DÉCODEUR, ENCODEUR, TRANSCODEUR

Implémentation de la fonction Majorité avec un Décodeur

Fonction Majorité

$$M = CB\overline{A} + C\overline{B}A + \overline{C}BA + CBA$$

A	B	C	M
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1



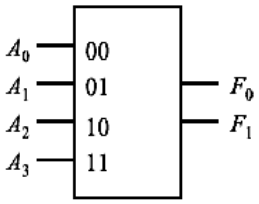
CIRCUITS LOGIQUES
DÉCODEUR, ENCODEUR, TRANSCODEUR

EnCodeur

On appelle EnCodeur tout système qui traduit une series d'entrées en un code binaire (peut être vu comme l'inverse du DéCodeur)

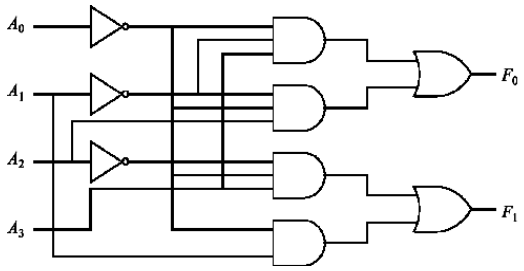
• Un EnCodeur prioritaire est un type d'EnCodeur qui va imposé un ordre sur les entrées. Dans notre exemple l'ordre de priorité est

A_i a une plus haute priorité que A_{i+1}



$$F_0 = \overline{A_0}\,\overline{A_1}\,A_3 + \overline{A_0}\,\overline{A_1}\,A_2$$
$$F_1 = \overline{A_0}\,\overline{A_2}\,A_3 + \overline{A_0}\,A_1$$

A_0	A_1	A_2	A_3	F_0	F_1
0	0	0	0	0	0
0	0	0	1	1	1
0	0	1	0	1	0
0	0	1	1	1	0
0	1	0	0	0	1
0	1	0	1	0	1
0	1	1	0	0	1
0	1	1	1	0	1
1	0	0	0	0	0
1	0	0	1	0	0
1	0	1	0	0	0
1	0	1	1	0	0
1	1	0	0	0	0
1	1	0	1	0	0
1	1	1	0	0	0
1	1	1	1	0	0



CIRCUITS LOGIQUES
DECODEUR, ENCODEUR, TRANSCODEUR

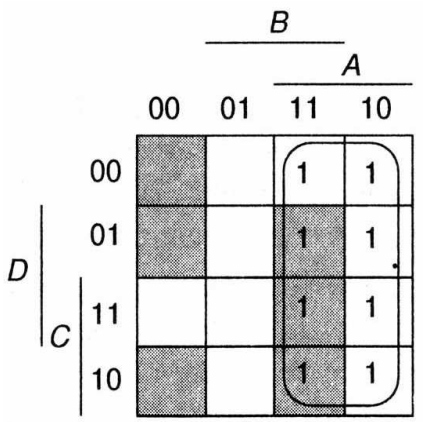
TransCodeur

On appelle Transcodeur tout système qui fait correspondre à un code A en entrée (sur n lignes), un code B en sortie (sur m lignes)

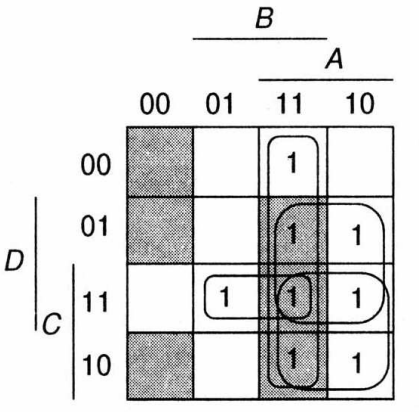
Code excédent 3 en un code 2-4-2-1

	Entrées				Sorties			
	A	*B*	*C*	*D*	*X*	*Y*	*Z*	*T*
0	0	0	1	1	0	0	0	0
1	0	1	0	0	0	0	0	1
2	0	1	0	1	0	0	1	0
3	0	1	1	0	0	0	1	1
4	0	1	1	1	0	1	0	0
5	1	0	0	0	1	0	1	1
6	1	0	0	1	1	1	0	0
7	1	0	1	0	1	1	0	1
8	1	0	1	1	1	1	1	0
9	1	1	0	0	1	1	1	1

CIRCUITS LOGIQUES
DÉCODEUR, ENCODEUR, TRANSCODEUR

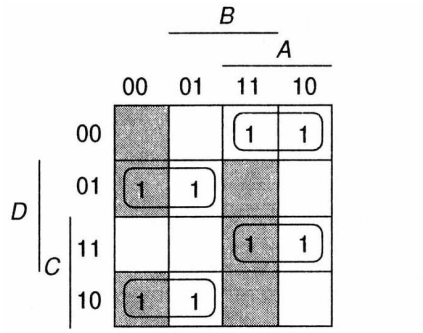


$X = A$



$Y = AB + AD + AC + BCD$

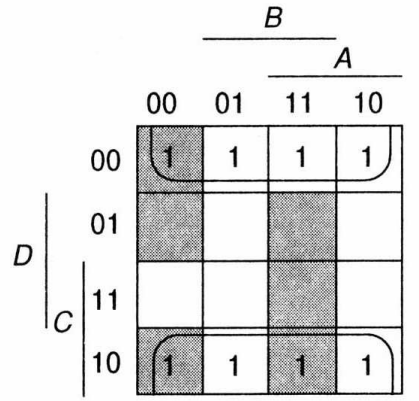
$= A(B + C + D) + BCD$



$Z = A\bar{C}\bar{D} + ACD + \bar{A}\bar{C}D + \bar{A}C\bar{D}$

$= A(\bar{C}\bar{D} + CD) + \bar{A}(\bar{C}D + C\bar{D})$

$= A\overline{(C \oplus D)} + \bar{A}(C \oplus D) = A \oplus C \oplus D$

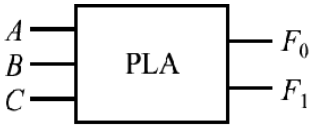


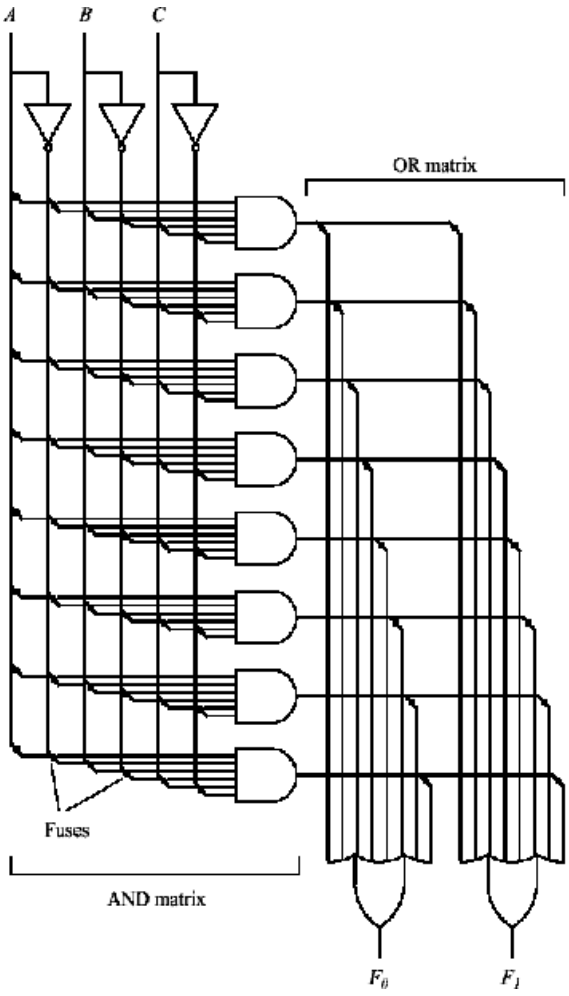
$T = \bar{D}$

CIRCUITS LOGIQUES
PLA

PLAs [*Programmable Logic Arrays*]

• On appelle PLA un composant qui consiste en une matrice de ET (AND) suivie d'une matrice de OU (OR)

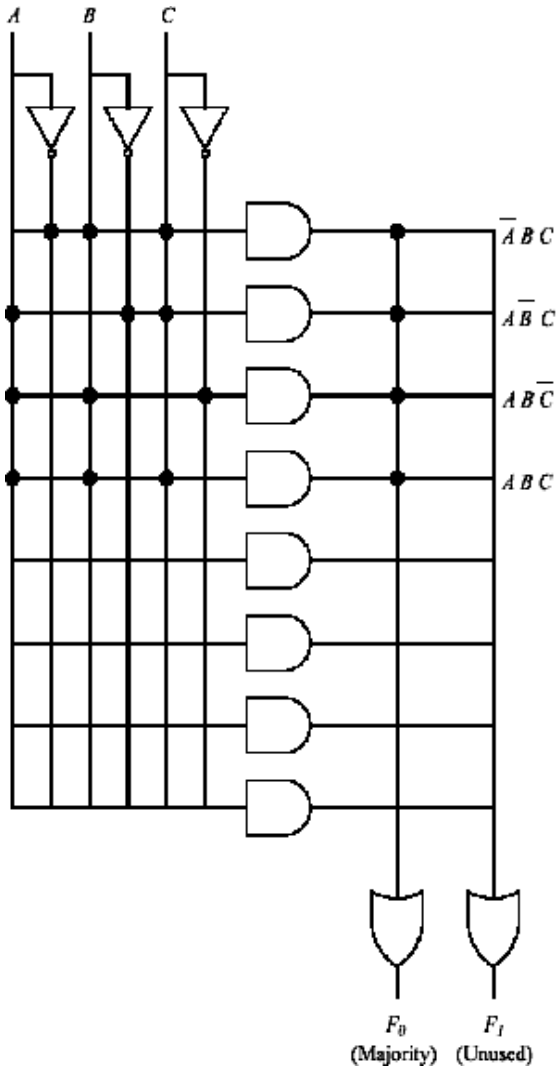




CIRCUITS LOGIQUES
PLA

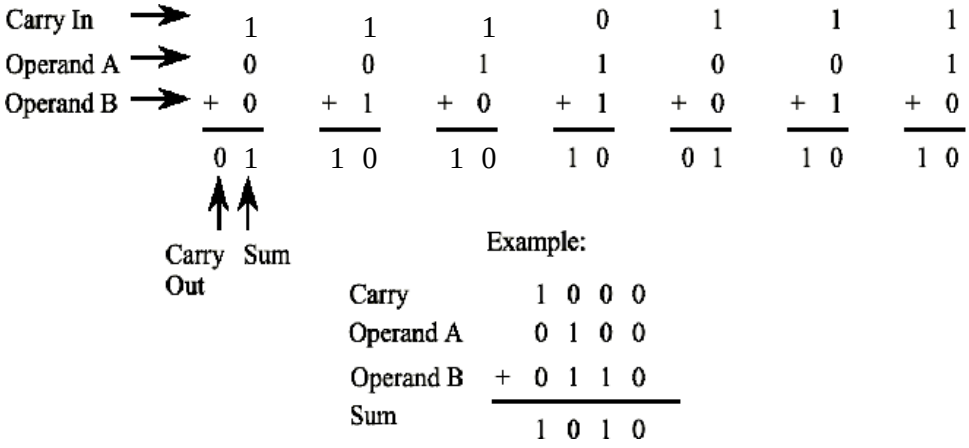
Exemple: Fonction Majorité

$$M = CB\overline{A} + C\overline{B}A + \overline{C}BA + CBA$$

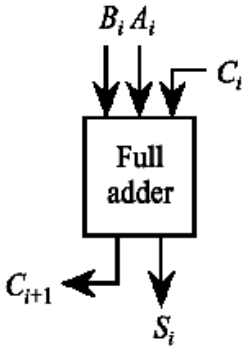


CIRCUITS LOGIQUES
PLA: EXEMPLE DE L'ADDITION AVEC RETENUE

L' Addition avec Retenue [*Ripple-Carry Addition*]



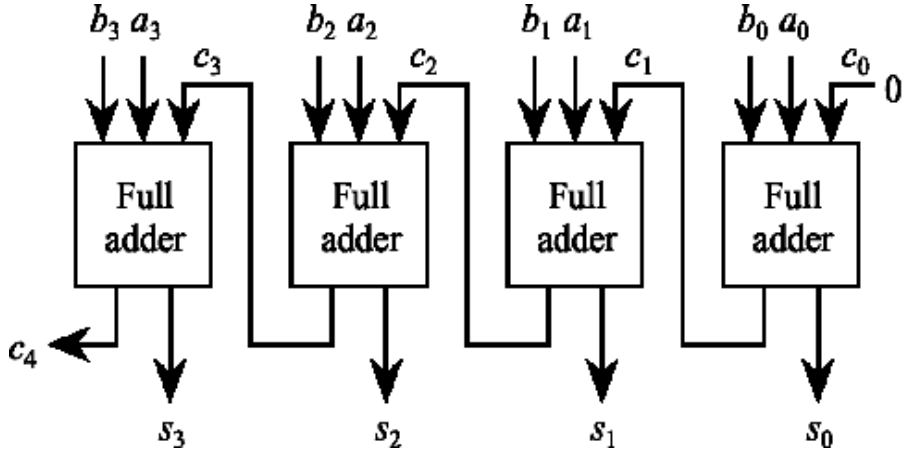
A_i	B_i	C_i	S_i	C_{i+1}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

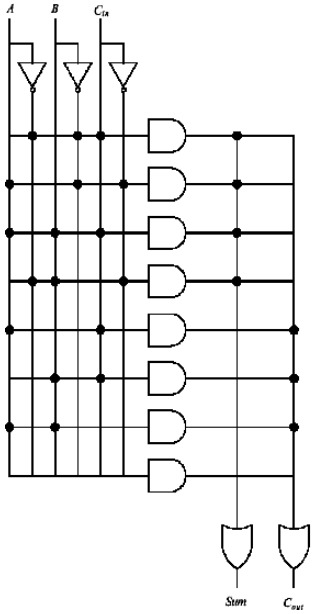


CIRCUITS LOGIQUES
PLA: EXEMPLE DE L'ADDITION AVEC RETENUE

Additionneur 4 Bits

- 4 [*full adders]* connectés dans une [*ripple-carry*] chaîne



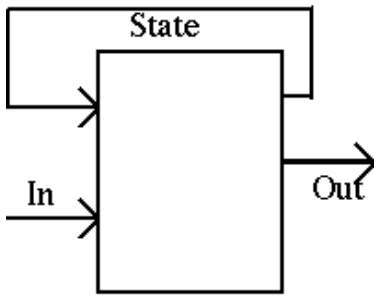


CIRCUITS LOGIQUES
CIRCUITS SÉQUENTIELS & FSM

Circuits Séquentiels

• Précédemment, les circuits combinatoires (*idéaux*) n'avaient aucune mémoire (sorties=f[entrées])

• Les circuits séquentiels possèdent des rétroactions : les signaux de sortie ne dépendent pas uniquement des entrées mais aussi de leur séquence (*mémoire du passé*)

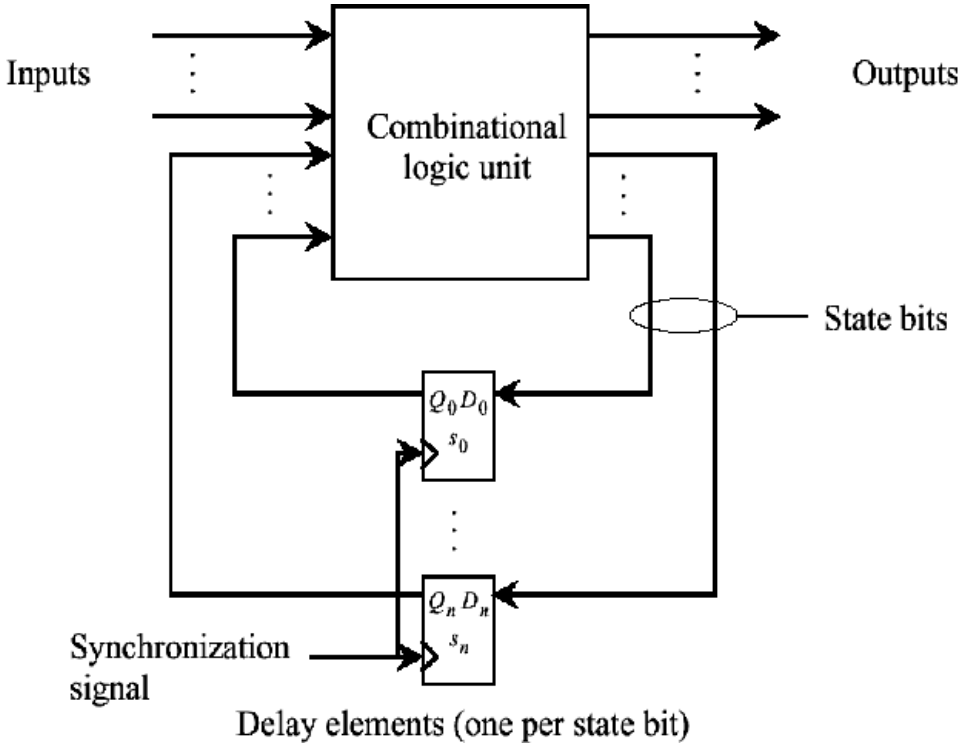


• Exemple : Distributeur automatique

• On les appelle les machines à états finis ([*Finite State Machine*]) parce qu'il ne peut prendre qu'un nombre fini d'états internes

• Si l'étude des circuits combinatoires repose sur l'algèbre de Boole, celle des circuits séquentiels repose sur la théorie des automates finis

CIRCUITS LOGIQUES
CIRCUITS SÉQUENTIELS & FSM

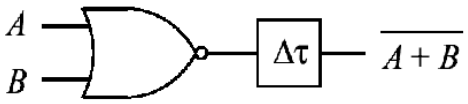


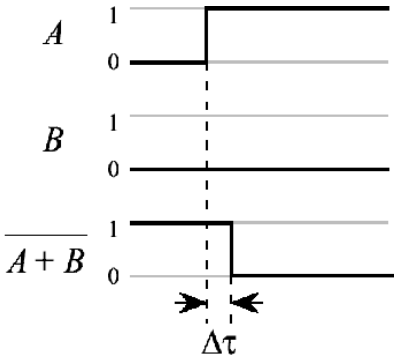
• Un **FSM** est composé d'une unité de logique combinatoire (circuit combinatoire) et d'un nombre fini de mémoires dans une boucle de rétroaction qui mémorise ses états internes

$$S(t+1) = f[E(t), Q(t)]$$

• Q(t) : État interne du FSM à l'instant t

Porte logique NON-OU Réelle





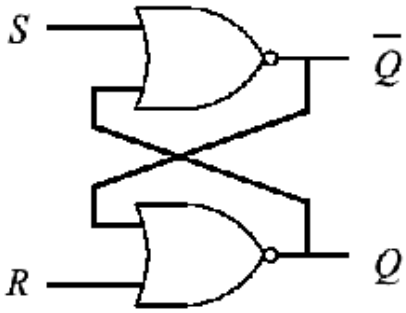
Timing Behavior

- Dans un circuit logique réelle, le temps de propagation entre l'entrée et la sortie n'est pas instantanée et est égale à $\Delta\tau$

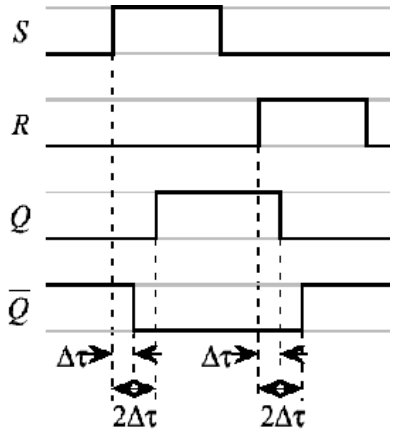
- Cette propriété est utilisé dans le fonctionnement d'un composant essentiel utilisé comme élément de mémoire dans un **FSM** que l'on appelle la Bascule (ou Bistables) flip-flop [*flip-flop*]

Bascule S-R Flip-Flop

Arrangement de portes logiques qui va maintenir un état stable de sortie même après que les entrées ne soient plus actives

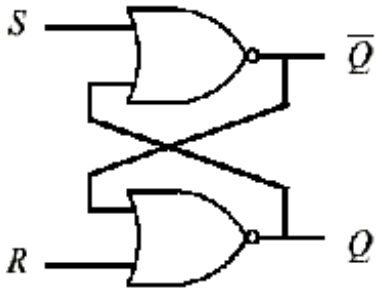


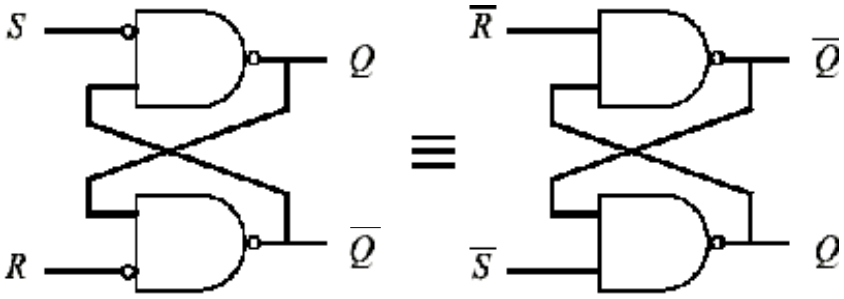
Q_t	S_t	R_t	Q_{t+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	(disallowed)
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	(disallowed)



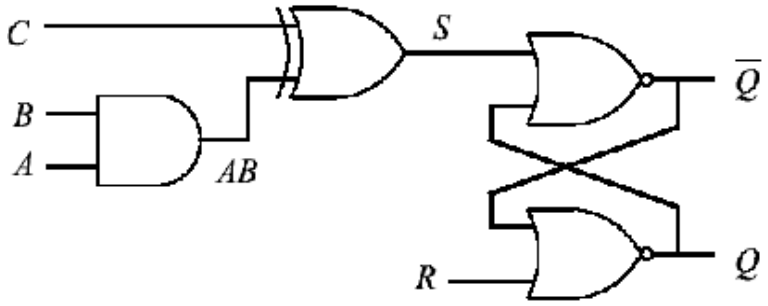
Timing Behavior

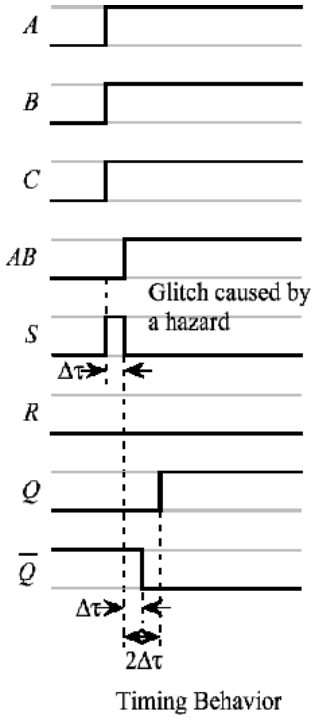
Design de Bascules S-R Flip-Flop possibles





Exemple de problèmes avec les circuits non synchrones

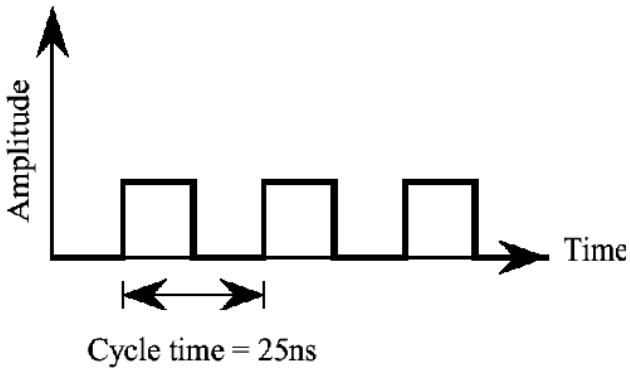




Timing Behavior

CIRCUITS LOGIQUES
CIRCUITS SÉQUENTIELS & FSM

Horloges



• En logique positive, l'*Action* se passe quand l'horloge est haute ou positive.

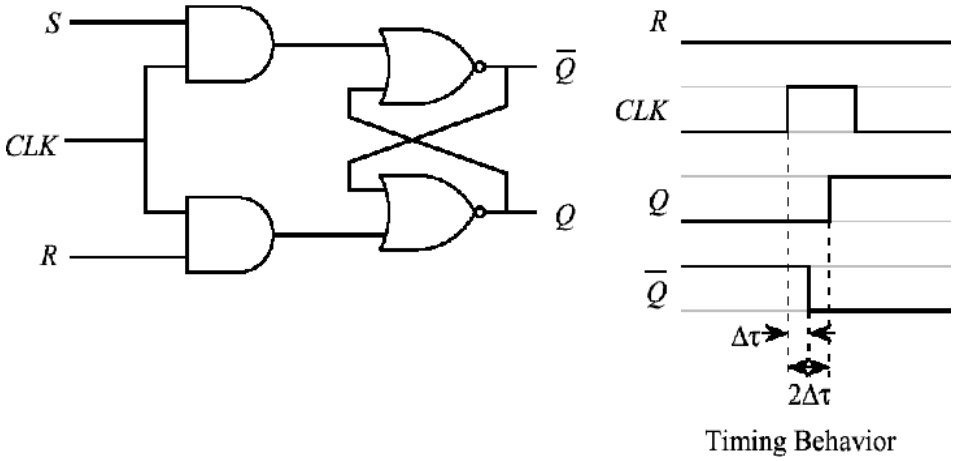
• Dans les circuits **asynchrones**, la réponse (sortie) est évaluée dès qu'il y a un signal d'entrée. Dans les circuits **synchrones**, on définit un signal d'entrée spécial C [*clock*] qui est le signal d'horloge. La sortie du circuit ne peut changer que si l'entrée *clock* est activée

• Les vitesse d'horloges sont exprimée en Kilo Hertz ou Khz (10^3 KHz), Mega Hertz ou MHz (10^6 Hz), Giga Hertz ou GHz (10^9 Hz)

A ne pas confondre avec les Kilo, Mega,
Giga utilisés pour la mémoire de l'ordinateur !!!

Kilo Octet $= 2^{10} = 1024$ Octets
Mega Octet $= (2^{20} = 1024K)$
Giga Octet $= (2^{30} = 1024M)$

Bascule S-R Flip-Flop synchrone

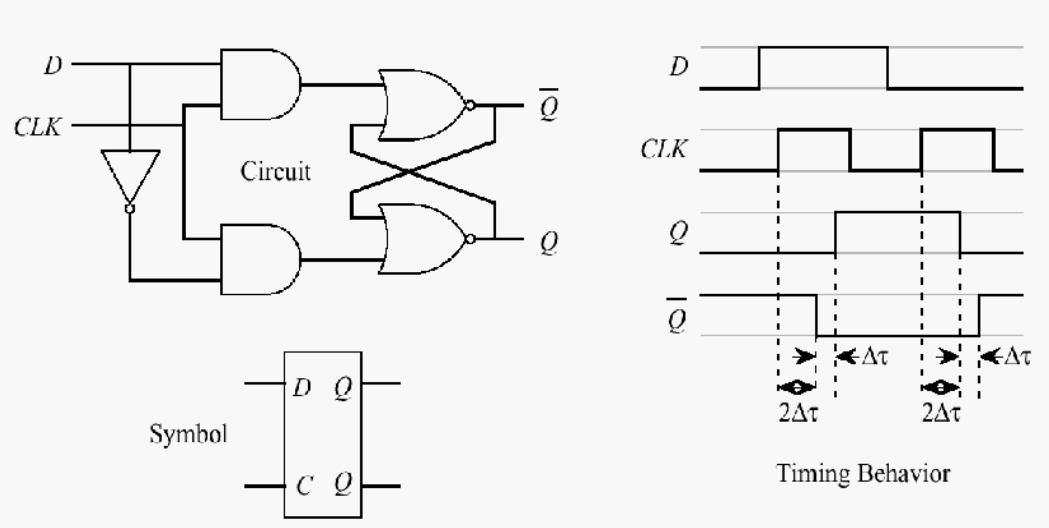


• Le signal d'horloge, CLK, sert à valider l'entrée de la flip-flop

• Avantages de cette synchronisation, entre autre, 1- Valider R et S qu'au moment souhaité, 2- protéger la mémoire des parasites survenant en R et S tant que T=0 (T=1 durée minimale), 3- Préparer les commandes R et S sans perturber la mémoire (T=0), etc.

•Combinaison R=S=1 interdite et lorsque R et S varie en même temps que CLK ▷ problème

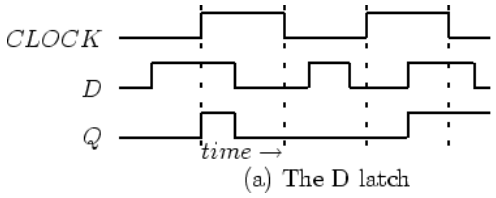
Bascule D Flip-Flop synchrone



• La bascule D [*Delay*] recopie, sur sa sortie Q, l'unique signal appliqué à son entrée D, avec un retard d' un cycle d'horloge

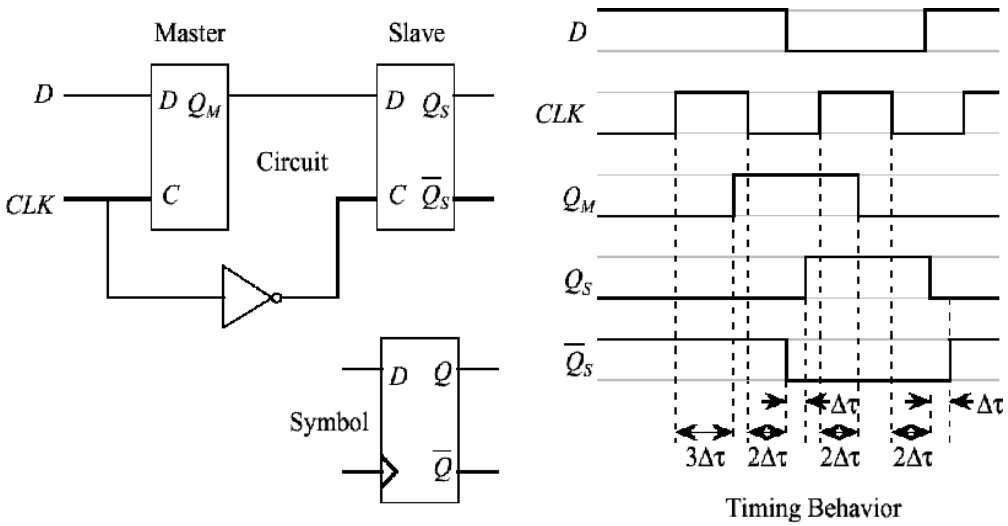
• Facilement construite à partir d'une bascule S-R et ne nécessitant pas la gestion de différentes entrées

• Le bistable D (appelé aussi [*latch*]) a un problème potentiel : Si D change quand CLK=1 logique, la sortie changera. La Bascule Maitre-Esclave [*Master-Slave*] résout ce problème



(a) The D latch

Bascule Flip-Flop Maitre esclave

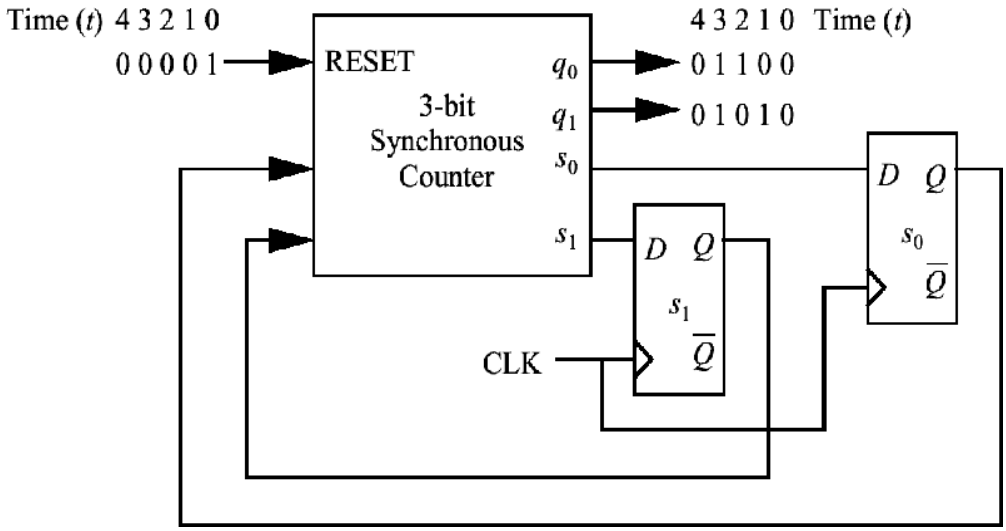


• Le front montant du signal d'horloge permet de charger de nouvelles données dans la bascule maître, pendant que la bascule esclave maintient la données précédente. Le front descendant du signal d'horloge charge la nouvelle donnée du maître dans la bascule esclave

Bascule Protégé contre les parasites

CIRCUITS LOGIQUES
EXEMPLE : COMPTEUR MODULO 4

Exemple: Compteur Modulo 4



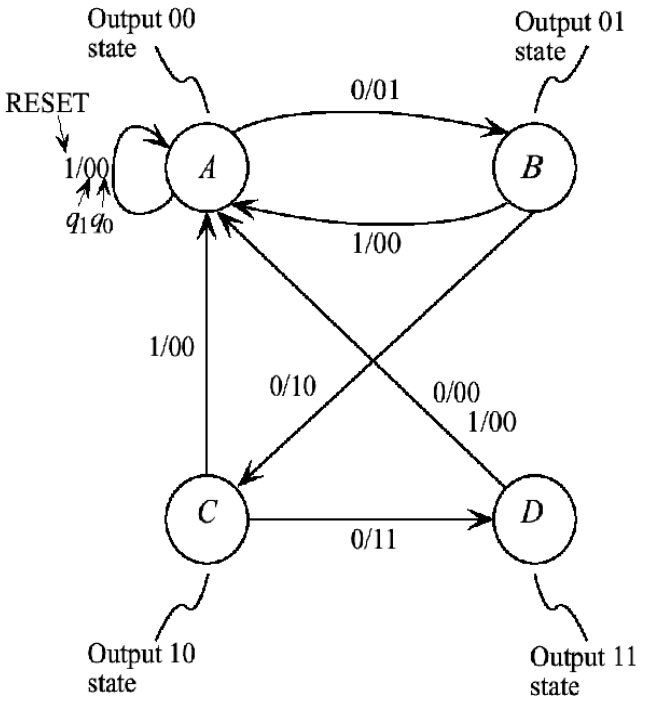
• Le compteur a un signal d'horloge et une entrée Reset (opérant de fonction synchrone avec l'horloge)

• La sortie du compteur est sur q_0 et q_1 et prend les valeurs successives, à la cadence de l'horloge, 00, 01, 10, 11, 00, 01, etc.

• 2 flip-flop permettront de mémoriser 2 bits, soit 2^2 états possibles

CIRCUITS LOGIQUES
EXEMPLE : COMPTEUR MODULO 4

1 Diagramme de Transition d'États



Present state \ Input	*RESET* 0	*RESET* 1
A	*B*/01	*A*/00
B	*C*/10	*A*/00
C	*D*/11	*A*/00
D	*A*/00	*A*/00

Next state — Output

CIRCUITS LOGIQUES
EXEMPLE : COMPTEUR MODULO 4

2 Table d'États

On doit encoder les 4 états possibles (codable sur 2 bits) en binaires en choisissant arbitrairement le codage suivant $A = 00$, $B = 01$, $C = 10$, $D = 11$

Present state (S_t) \ Input	*RESET* 0	*RESET* 1
A:00	01/01	00/00
B:01	10/10	00/00
C:10	11/11	00/00
D:11	00/00	00/00

3 Table de Vérité

RESET $r(t)$	$s_1(t)$	$s_0(t)$	$s_1s_0(t+1)$	$q_1q_0(t+1)$
0	0	0	01	01
0	0	1	10	10
0	1	0	11	11
0	1	1	00	00
1	0	0	00	00
1	0	1	00	00
1	1	0	00	00
1	1	1	00	00

$$s_0(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,\overline{s_0(t)} + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$
$$s_1(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,s_0(t) + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$
$$q_0(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,\overline{s_0(t)} + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$
$$q_1(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,s_0(t) + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$

CIRCUITS LOGIQUES
EXEMPLE : COMPTEUR MODULO 4

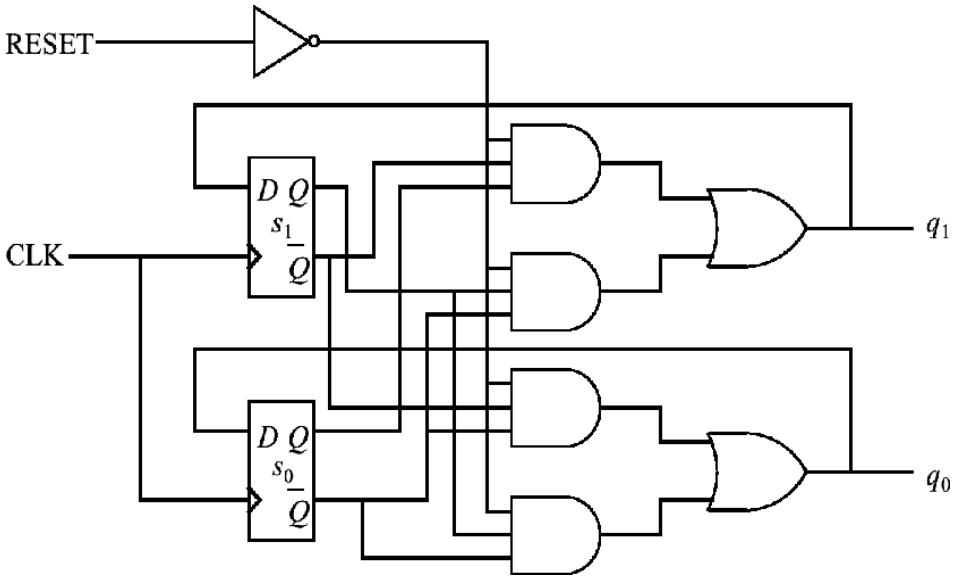
4 Logigramme

$$s_0(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,\overline{s_0(t)} + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$

$$s_1(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,s_0(t) + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$

$$q_0(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,\overline{s_0(t)} + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$

$$q_1(t+1) = \overline{r(t)}\,\overline{s_1(t)}\,s_0(t) + \overline{r(t)}\,s_1(t)\,\overline{s_0(t)}$$



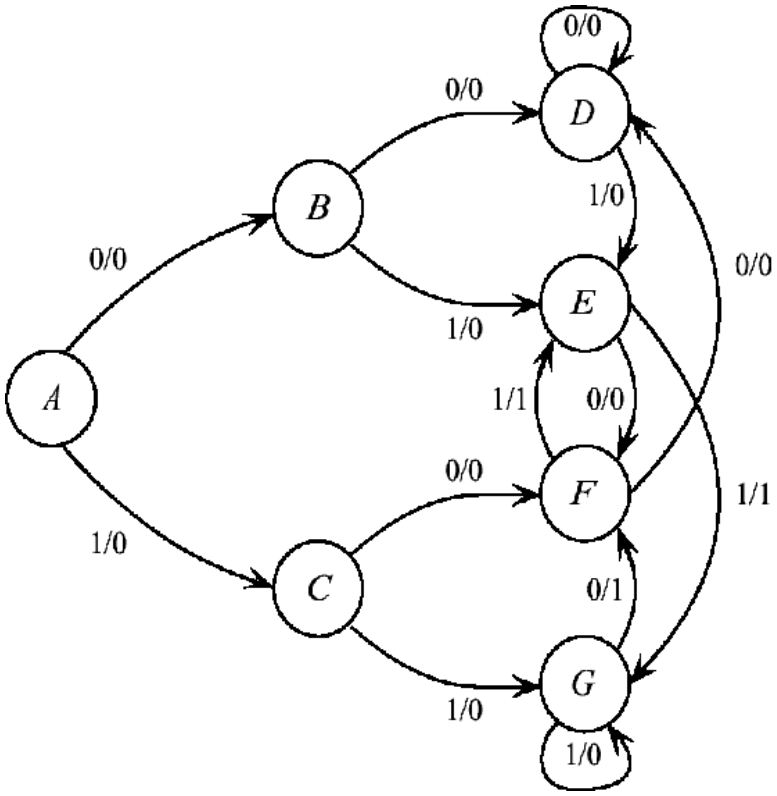
CIRCUITS LOGIQUES
EXEMPLE : DÉTECTEUR DE SÉQUENCE

Exemple: Détecteur de Séquence

• Automate qui mette une sortie à "1" lorsque exactement deux de ses trois entrés (1-bit série) sont à "1"

Exemple : Une séquence d'entrée 0 1 1 0 1 1 1 0 0
produira une séquence de sortie 0 0 1 1 1 1 0 1 0

1 Diagramme de Transition d'États



CIRCUITS LOGIQUES
EXEMPLE : DÉTECTEUR DE SÉQUENCE

Present state \ Input	X = 0	X = 1
A	B/0	C/0
B	D/0	E/0
C	F/0	G/0
D	D/0	E/0
E	F/0	G/1
F	D/0	E/1
G	F/1	G/0

2-3 Table d'États et de Vérité

Present state \ Input	X = 0	X = 1
A: 000	001/0	010/0
B: 001	011/0	100/0
C: 010	101/0	110/0
D: 011	011/0	100/0
E: 100	101/0	110/1
F: 101	011/0	100/1
G: 110	101/1	110/0

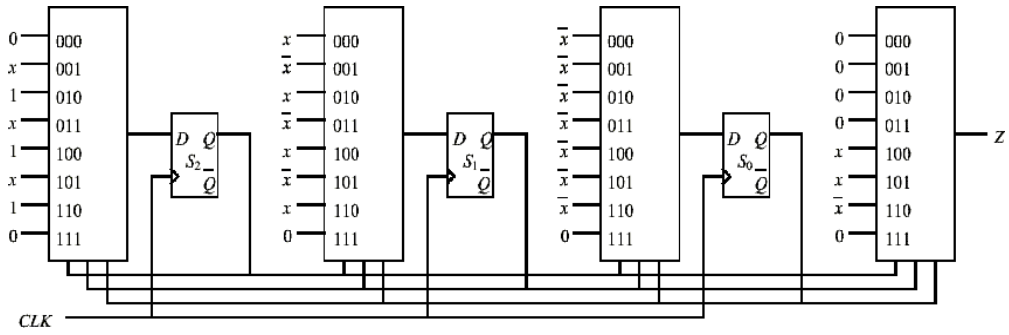
(a)

Input and state at time t				Next state and output at time t+1			
s_2	s_1	s_0	x	s_2	s_1	s_0	z
0	0	0	0	0	0	1	0
0	0	0	1	0	1	0	0
0	0	1	0	0	1	1	0
0	0	1	1	1	0	0	0
0	1	0	0	1	0	1	0
0	1	0	1	1	1	0	0
0	1	1	0	0	1	1	0
0	1	1	1	1	0	0	0
1	0	0	0	1	0	1	0
1	0	0	1	1	1	0	1
1	0	1	0	0	1	1	0
1	0	1	1	1	0	0	1
1	1	0	0	1	0	1	1
1	1	0	1	1	1	0	0
1	1	1	0	d	d	d	d
1	1	1	1	d	d	d	d

(b)

CIRCUITS LOGIQUES
EXEMPLE : DÉTECTEUR DE SÉQUENCE

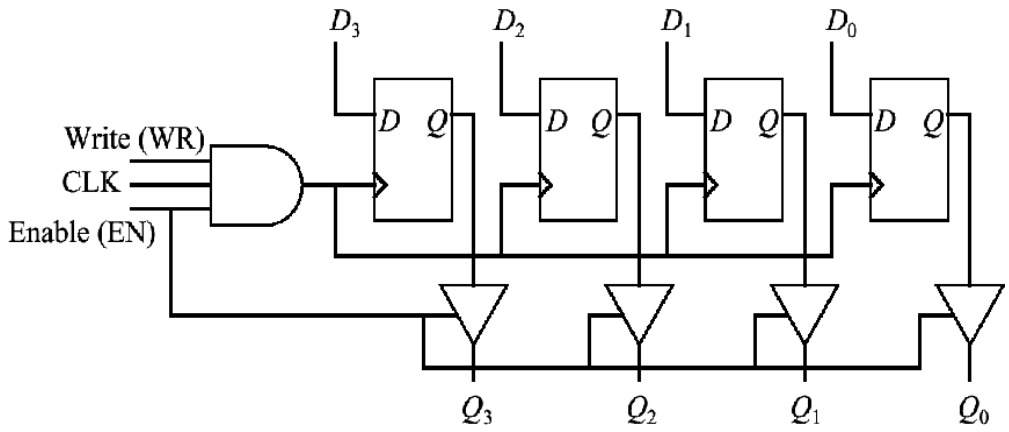
4 Logigramme



• Une flip flop pour chaque bit encodant les différents états et un MUX pour chaque fonction

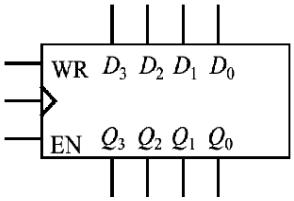
CIRCUITS LOGIQUES
EXEMPLE : REGISTRE 4 BITS

Registre 4 Bits



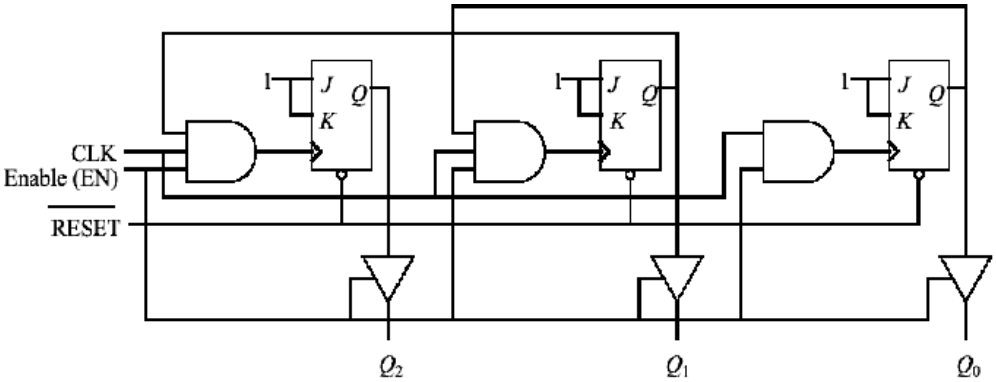
• Un bit d'information est stocké dans une seule bascule D flip-flop

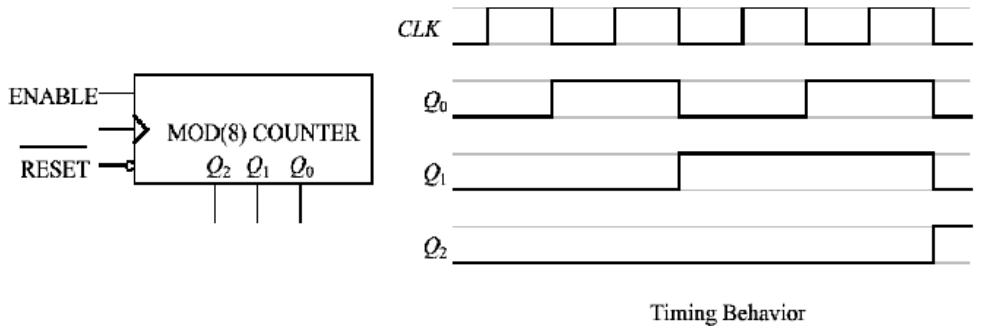
• Les données sont chargées dans le registre lorsque la ligne Write and Enable sont à "1" et synchrone avec l'horloge



CIRCUITS LOGIQUES
ANNEXE A

Compteur Modulo 8





Timing Behavior

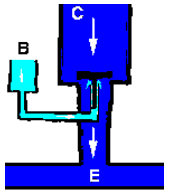
• La bascule J-K se comporte comme la bascule S-R excepté qu'elle change son état Q quand les deux entrés sont à "1"

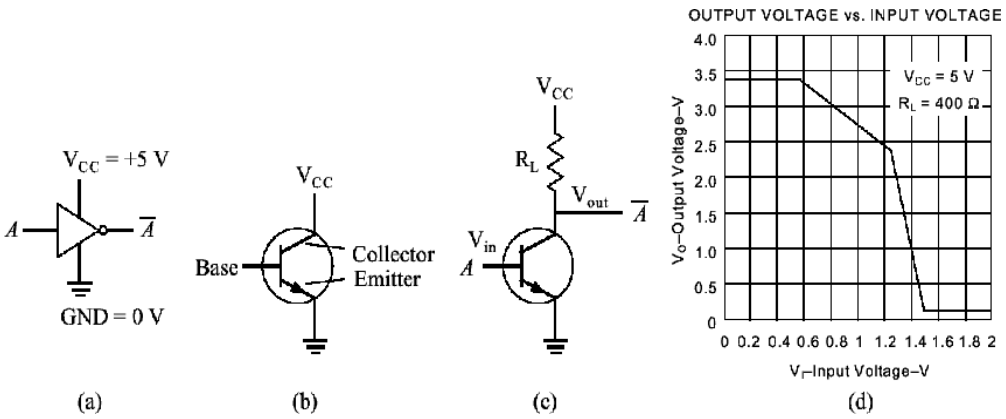
• Quand ses deux entrés sont reliés, la bascule J-K se comporte comme la bascule T

CIRCUITS LOGIQUES
ANNEXE B

Les fonctions logiques élémentaires (NON, NAND, NOR)
sont composés à partir de **transistors**
jouant le rôle de Switch et d'amplificateur

Transistor & Porte Inverseur (NON)



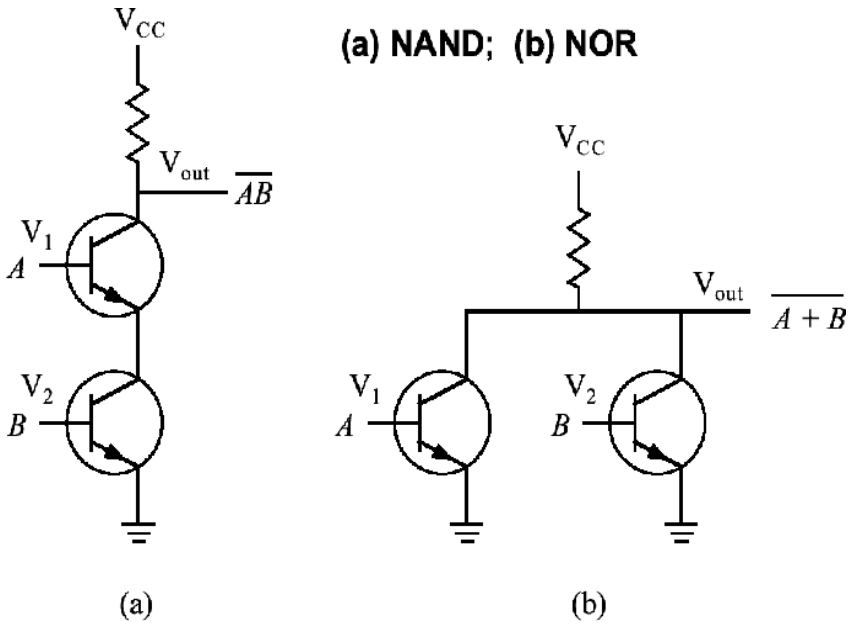


• Quand $V_{in} = 0$ Volts (A= 0 ou entrée=0), transistor bloqué, aucun courant dans R_l $V_{out} = Vcc$ (Sortie=$\overline{A}$)

• Quand $V_{in} > 1.5$ Volts (A= 1 ou entrée=1) (Cf. Diagramme d), transistor passant, un courant circule dans R_l et $V_{out} \approx 0$ (Sortie=$\overline{A}$)

CIRCUITS LOGIQUES
ANNEXE B

Transistor & Porte NAND et NOR

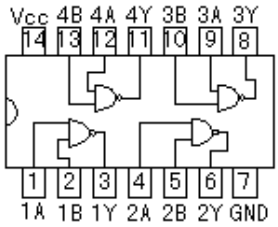


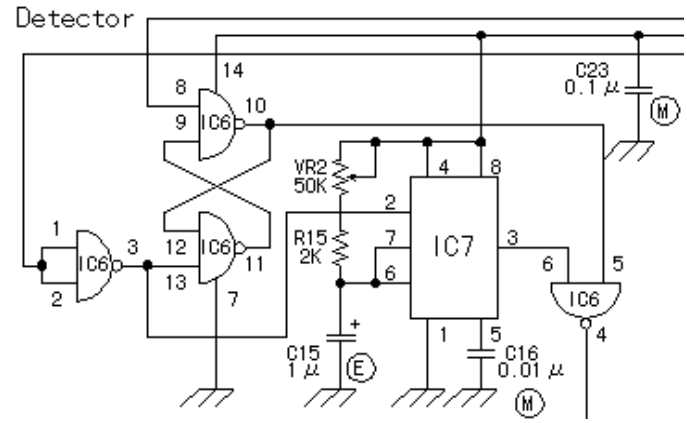
• **NAND** les deux transistors doivent être passant (A=1 & B=1) pour qu'un un courant circule dans R et $V_{out} \approx 0$ (Sortie=$\overline{AB}$)

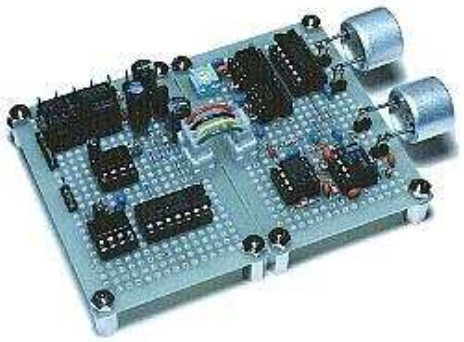
• **NOR** Un des deux transistors doivent être passant (A=1 || B=1) pour qu'un un courant circule dans R et $V_{out} \approx 0$ (Sortie=$\overline{A+B}$)

CIRCUITS LOGIQUES
ANNEXE C

Circuits Intégrés







CIRCUITS LOGIQUES
ANNEXE C

Circuits Intégrés

• Les circuits intégrés moderne réalisent une fonction particuliere et intègrent un grand nombre de portes logiques différentes

Niveau d'intégration (nombre de portes logiques) dans un circuit intégré [*IC*]

- ▷ SSI [*Small Scale Integration* 1966]
  10-100 portes logiques
- ▷ MSI [*Medium Scale Integration* 1969]
  100-1000 portes logiques
- ▷ LSI [*Large Scale Integration* 1975]
  1000-10000 portes logiques
- ▷ VSI [*Very Large Scale Integ.* 1980]
  10K-100K portes logiques
- ▷ ULSI [*Ultra Large Scale Integration* 1985]
  100K- . . . portes logiques

CIRCUITS LOGIQUES
ANNEXE D

Logique Positive vs. Logique Négative

• Logique positive
Assertion vraie = "1" logique = haut voltage (5 Volts)
Assertion fausse = "0" logique = faible voltage (0 Volts)
[*Active high*] [*Positive Logic*]

• Logique négative
Assertion vraie = "0" logique = faible voltage (0 Volts)
Assertion fausse = "1" logique = haut voltage (5 Volts)
[*Active low*] [*Negative Logic*]