Université Blaise Pascal

UFR Sciences et Technologies

L1 : Découverte de l'informatique
1ère partie : de l'électronique à l'informatique

(4) Circuits logiques

1. Algèbre de Boole

Eléments indispensables pour la conception et l'analyse de circuits logiques

► **Algèbre de Boole =**
- **Un ensemble B={0,1} (*faux, vrai*)**
- **Plus un opérateur unaire : – (non)**
- **Plus deux opérateurs binaires de base : + (ou) , ∗ (et)**

► **Tables de calcul (tables de vérité) :**

non	
0	1
1	0

ou	0	1
0	0	1
1	1	1

et	0	1
0	0	0
1	0	1

→ Arithmétique modulo 2 (sans retenue)

→ Théorie des ensembles (union, intersection)

► **Priorité d'opérateurs (parenthésage) : – ➢ + ➢ ∗**

► **Notations différentes selon les domaines**

Algèbre de Boole = Arithmétique binaire = Logique booléenne = Logique propositionnelle

NOTATIONS		Logique	Acronyme	Maths
constantes	vrai	T	TRUE	1
	faux	F	FALSE	0
opérateur unaire	négation	¬	NOT	– (1)
principaux opérateurs binaires	et	∧	AND	∗ (1)
	ou inclusif	∨	OR	+
	ou exclusif (ou bien)	⊕	XOR	
	implique	⇒	IMP	→ ou parfois >
	équivalent	⇔	EQU	≡ ou =
	non-et	Δ	NAND	
	non-ou	∇	NOR	

(1) On utilise souvent un NOT suscrit et un AND implicite : $\overline{A}B$ pour (−A)∗B

!! Distinguer $\overline{A\,B}$ et $\overline{A}\,\overline{B}$

► **Propriétés fondamentales**

Involution : –(–x)=x
Eléments neutres/absorbants : x+0=x x∗1=x x+1=1 x∗0=0 *(théorème des constantes)*
Complémentation : (–x)+x=1 (–x)∗x=0
Commutativité : x+y=y+x x∗y=y∗x
Distributivités : x∗(y+z)=x∗y+x∗z x+(y∗z)=(x+y)∗(x+z)

► **Lois de De Morgan**

–(x+y)=(–x)∗(–y) –(x∗y)=(–x)+(–y) ✍ **Prouver ces lois avec des tables de vérité**

► **Complétude et minimalité**

x NOR y = –(x+y)
x NAND y = –(xy)
x EQU y = (x IMP y)∗(y IMP x)
x IMP y = –x+y
x XOR y = (x+y)∗(–x+–y) = (x∗–y)+(–x∗y)
x AND y = –(–x+–y)
x OR y = –(–x∗–y)

→ NOT + un opérateur binaire suffisent

► Fonctions booléennes

- Fonctions d'une variable et opérateurs unaires : identité, négation (–)
- Fonctions de deux variables et opérateurs binaires

valeurs de x, y: 0 0	0 1	1 0	1 1	fonction F(x,y)	opérateur associé
0	0	0	0	Falsification	...
0	0	0	1	**conjonction**	**AND**
0	0	1	0	soustraction	...
0	0	1	1	1ère projection	...
0	1	0	0	...	...
0	1	0	1	2ème projection	...
0	1	1	0	**exclusion**	**XOR**
0	1	1	1	**disjonction**	**OR**
1	0	0	0	**non-disjonction**	**NOR**
1	0	0	1	**équivalence**	**EQU**
1	0	1	0	...	...
1	0	1	1	...	...
1	1	0	0	...	...
1	1	0	1	implication	IMP
1	1	1	0	**non-conjonction**	**NAND**
1	1	1	1	Vérification	...

valeurs de F(x,y)

- Fonctions de trois variables
- ...

► Théorème de l'expressivité

Toute fonction booléenne peut être définie par une formule booléenne où apparaissent ses variables et les seuls opérateurs NOT et OR (ou bien NOT et AND, ou bien NOR et NAND).

► vocabulaire

- Littéral : variable x ou son complément –x.
- Clause : disjonction (OR) de littéraux.
  exp. –x+y+z+t+–u
- Forme normale* conjonctive : conjonction de clauses.
  exp. (–x+y)∗(z+t+–u)∗(x+–t)
- Forme normale disjonctive : disjonction de conjonctions de littéraux.
  exp. –x∗y+z∗t∗–u+x∗–t

* une clause n'apparaît jamais plus d'une fois, et une clause ne peut contenir à la fois une variable et sa négation

- minterm(a,b)={ab, (-a)b, a(-b), (-a)(-b)}, maxterm(a,b)={a+b, (-a)+b, a+(-b), (-a)+(-b)}

► Théorèmes de simplification

1°) *Toute fonction booléenne peut être exprimée sous forme normale disjonctive.*
2°) *Pour toute forme disjonctive, il existe au moins* une formule disjonctive équivalente minimisant le nombre de disjonction.*

** il peut y en avoir plusieurs => pas de minimisation canonique*

N.B. Ajouter une variable multiplie par deux la complexité de la détermination de la satisfiabilité d'une fonction booléenne.

► Simplification : méthode de Karnaugh pour trois variables - exemple

Table de vérité de
F(x,y,z)=x(-y)z+yz+(-x)z

n° ligne	variables: x	y	z	calculs: x(-y)z	yz	(-x)z	F
0	0	0	0	0	0	0	0
1	0	0	1	0	0	1	1
2	0	1	0	0	0	0	1
3	0	1	1	0	1	1	1
4	1	0	0	0	0	0	0
5	1	0	1	1	0	0	1
6	1	1	0	0	0	0	0
7	1	1	1	0	1	0	1

$7=111_2$

Tableau de Karnaugh
pour 3 variables

x\ yz	0 0	0 1	1 1	1 0
0	n°0	n°1	n°3	n°2
1	n°4	n°5	n°7	n°6

code de Gray

une case du tableau ⇔ une ligne de la table de vérité

1°) Construire le tableau
de Karnaugh pour F

y=1

x\ yz	0 0	0 1	1 1	1 0
0	0	1*	1	1
1	0	1	1	0

z=1

2°) Regrouper les 1 par rectangles maximaux*
(4x4 > 4x2 ou 2x4 > 2x2 > 1x2 ou 2x1 > 1x1)
et en déduire une conjonction pour chaque

(-x)y (1x2 : 2 littéraux)
z (2x2 : 1 littéral)
F(x,y,z)=(-x)y+z

* recouvrement possible des rectangles, regroupement circulaire des 1, commencer par les 1 qui n'entrent que dans un seul rectangle.

Situations typiques

Un seul rectangle ☺

x\ yz	0 0	0 1	1 1	1 0
0	1	0	0	1
1	1	0	0	1

Tous les uns,
dans l'ordre ☺

x\ yz	0 0	0 1	1 1	1 0
0	1	1	0	1
1	0	1	0	0

Oups ☹

x\ yz	0 0	0 1	1 1	1 0
0	1	1	0	1
1	0	1	0	0

Taille incorrecte ☹

x\ yz	0 0	0 1	1 1	1 0
0	0	1	1	1
1	0	0	0	0

Désordre ☹

x\ yz	0 0	0 1	1 1	1 0
0	1	1	0	1
1	0	1	0	0

non maximal ☹

x\ yz	0 0	0 1	1 1	1 0
0	0	1	1	0
1	0	1	1	0

non maximal ☹

x\ yz	0 0	0 1	1 1	1 0
0	0	1	1	0
1	0	1	0	0

N.B. Certaines conjonctions peuvent être précisées "vides"
(en électronique : situations ne pouvant arriver).
Dans ce cas, les cases du tableau sont mises à 1 ou 0
selon les besoins de construction de rectangles
(établir un nombre minimal de rectangles maximaux).

► **Simplification : méthode de Karnaugh pour quatre variables**

Table de vérité pour 4 variables

lignes	x	y	z	t	F
0	0	0	0	0	
1	0	0	0	1	
2	0	0	1	0	
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
6	0	1	1	0	
7	0	1	1	1	
8	1	0	0	0	
9	1	0	0	1	
10	1	0	1	0	
11	1	0	1	1	
12	1	1	0	0	
13	1	1	0	1	
14	1	1	1	0	
15	1	1	1	1	

Tableau de Karnaugh pour 4 variables

xy \ zt	0 0	0 1	1 1	1 0
0 0	n°0	n°1	n°3	n°2
0 1	n°4	n°5	n°7	n°6
1 1	n°12	n°13	n°15	n°14
1 0	n°8	n°9	n°11	n°10

Exemple : G(x,y,z,t)=ℜ(0,2,5,7,8,9,13,15)

N.B. on indique ainsi les lignes de la table de vérité pour lesquelles G=1, on aurait utilisé ℑ(...) ou *Im*(...) pour indiquer les lignes pour lesquelles G=0. Cette écriture évite d'avoir à construire la table de vérité.

1°)Tableau de Karnaugh de G

xy \ zt	0 0	0 1	1 1	1 0
0 0	1			1
0 1		1	1	
1 1		1	1	
1 0	1			1

y=1

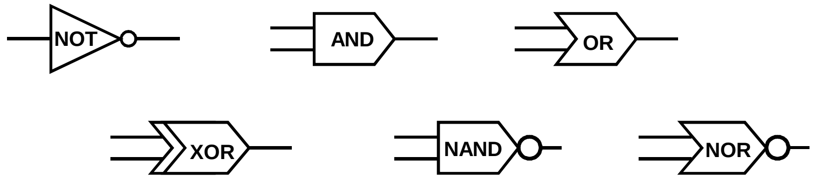
t=1

2°) Forme simplifiée :

G=yt+(-y)(-t)

2. Circuits combinatoires

► Porte logique : composant électronique réalisant un opérateur logique



► Circuit booléen : interconnexion de portes logiques

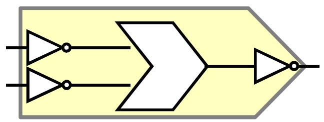
→ **à chaque circuit est associé une fonction booléenne**

+ Circuits booléens <u>combinatoires</u> : pas de contraintes de propagation du signal (<> Circuits séquentiels)

Algèbre booléenne ⇔ Circuits combinatoires

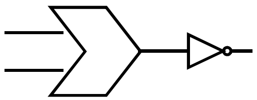
► Exemple : circuits prouvant la complétude de NOT + OR

AND



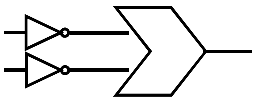
x AND y = NOT (NOT x OR NOT y)

NOR



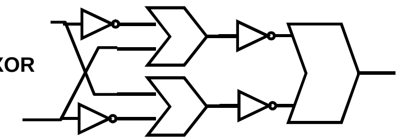
x NOR y = NOT (x OR y)

NAND



x NAND y = NOT x OR NOT y

XOR

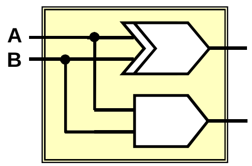


x XOR y = (x AND NOT y) OR (y AND NOT x)

✍ **Construire les circuits montrant que les opérateurs NAND et NOR suffisent en logique booléenne**

2.1. Synthèse d'un circuit

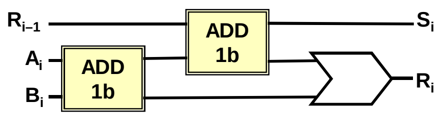
► Exemple : logigramme d 'un additionneur binaire



Additionneur 1 bit

0⊕0=0 ; 1⊕0=0⊕1=1 ; 1⊕1=0 (et 1 de retenue)

1+1=10 : retenue => A+B=RS



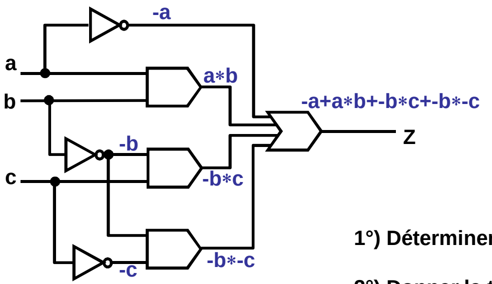
$S_i=R_{i-1}\oplus A_i$

$R_i=(R_{i-1}*(A_i\oplus B_i))+(A_i*B_i)$

Additionneur n bit : i^ème étage

2.2. Analyse d'un circuit

► Exemple d'analyse



1°) Déterminer l'expression de Z(a,b,c)

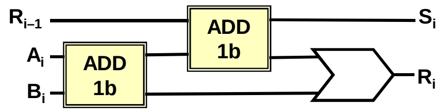
2°) Donner la table de vérité de Z

[expression simplifiée]

Z(a,b,c)=(-a+a∗b)+-b∗(c+-c)=-a+b+-b=1

3°) En déduire le rôle du circuit

► Application : vérifier la conception de l'additionneur binaire



Additionneur n bit : i^{ème} étage

Table de vérité du circuit

R_{i-1}	A_i	B_i	$A_i \oplus B_i$	$R_{i-1}*(A_i \oplus B_i)$	A_i*B_i	$R_i=(R_{i-1}*(A_i \oplus B_i))+(A_i*B_i)$	$S_i=R_{i-1} \oplus A_i \oplus B_i$
0	0	0	0	0	0	0	0
0	0	1	1	0	0	0	1
0	1	0	1	0	0	0	1
0	1	1	0	0	0	1	0
1	0	0	0	0	0	0	1
1	0	1	1	1	1	1	0
1	1	0	1	1	1	1	0
1	1	1	0	0	0	1	1

2.3. Circuits de référence

► Questions :

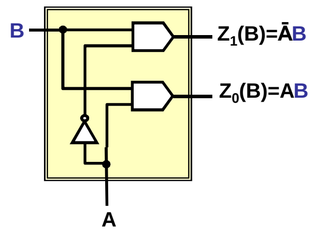
- Comment stocker un bit dans le i^{ème} case mémoire ? Comment le récupérer ?
→ démultiplexeur / multiplexeur

- Comment paralléliser ou sérialiser la transmission des données ?
→ démultiplexeur / multiplexeur (+temps)

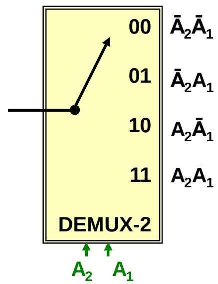
- A quel caractère correspond une touche du clavier ?
- Comment l'écran affiche un caractère ?
→ circuits de codage

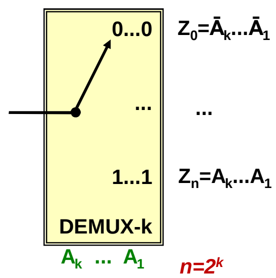
2.3.1. Démultiplexeur

► Circuit transmettant l'entrée vers une des sorties



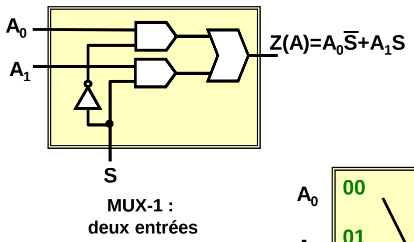
DEMUX-1 :
deux cases mémoires

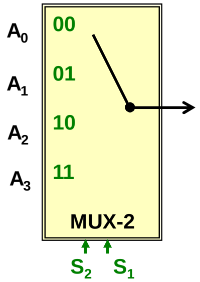


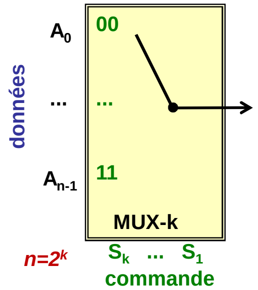


2.3.2. Multiplexeur

► Circuit transmettant une des entrées vers la sortie unique





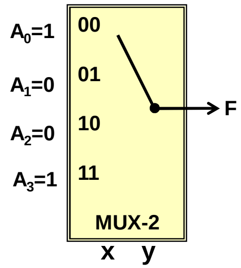


✍ **UART** (Universal Receiver Transmitter)

► Implémentation d'une fonction booléenne par multiplexeur

► $F(x,y)=\overline{x \oplus y} = 1\,\bar{x}\,\bar{y} + 0\,x\,\bar{y} + 0\,x\,\bar{y} + 1\,x\,y$

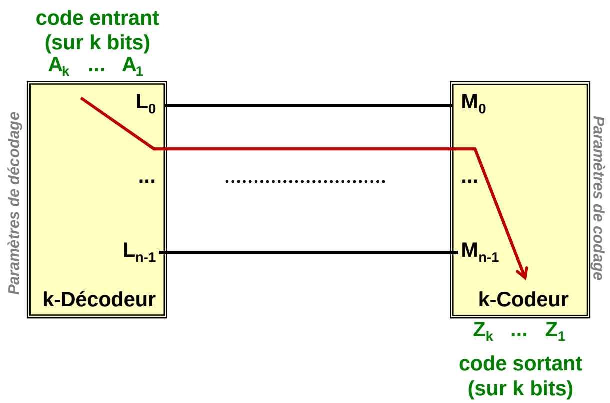
→ n'importe quelle fonction de 2 variables par MUX-2 (+ inverseur si besoin)



✍ **Exercices 2 et 3 du TD 4**

2.3.3. Application : circuits de codage

► Décodeur + Codeur = Transcodeur



► Exemple de transcodeur

- **Code* excédent 3 des chiffres décimaux → code de Haiken**
- **Ajouter une 5ème sortie T pour prendre en compte les codes interdits en excédent 3**

**** Code excédent 3 : ajouter 3 au code binaire usuel***
Code Haiken : complément à 15 des chiffres au delà de 4 (=>code(5)+code(4)=1111)

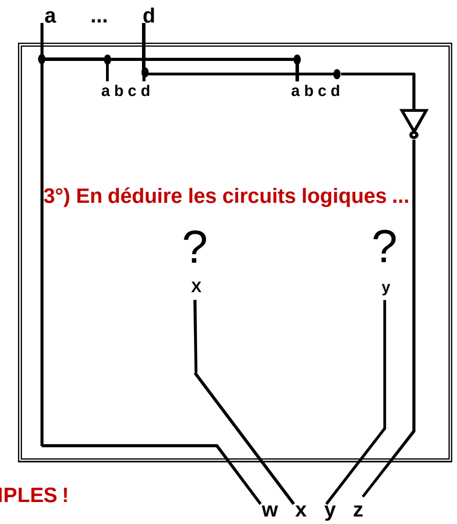
1°) construire la table de trancodage

chiffre	Exc. 3	Haiken	
0	0011	0000	Haiken=Exc−3
1	0100	0001	
2	0101	0010	
3	0110	0011	
4	0111	0100	
5	1000	1011	Haiken=Exc+3
6	1001	1100	
7	1010	1101	
8	1011	1110	
9	1100	1111	
	abcd	wxyz	

► Exemple de transcodeur

- **Code* excédent 3 des chiffres décimaux → code de Haiken**
- **Ajouter une 5ème sortie T pour prendre en compte les codes interdits en excédent 3**

chiffre	*Exc. 3*	*Haiken*
0	*0011*	*0000*
1	*0100*	*0001*
2	*0101*	*0010*
3	*0110*	*0011*
4	*0111*	*0100*
5	*1000*	*1011*
6	*1001*	*1100*
7	*1010*	*1101*
8	*1011*	*1110*
9	*1100*	*1111*
	abcd	*wxyz*



2°) Exprimer chaque variable de sortie en fonction des variables d'entrées

w=a
x=ab+bcd+ac+ad
y=a(-c)(-d)+b(-c)d+acd+bc(-d)
z=–d

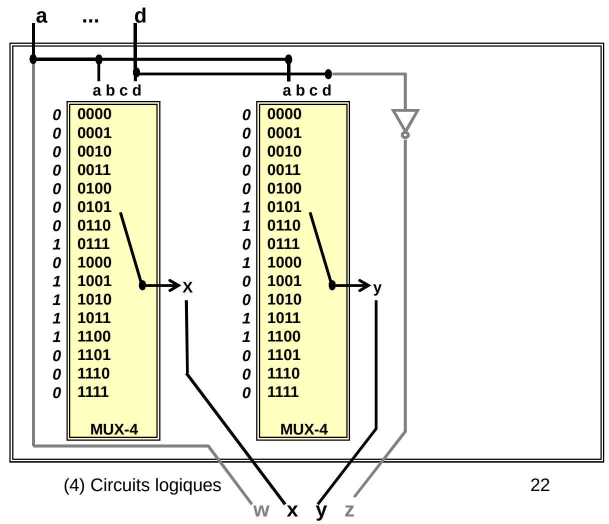
w et z : SIMPLES !

► Exemple de transcodeur

- **Code* excédent 3 des chiffres décimaux → code de Haiken**
- **Ajouter une 5ème sortie T pour prendre en compte les codes interdits en excédent 3**

3°) ... pour les fonctions x(a,b,c,d) et y(a,b,c,d) :

chiffre	*Exc. 3*	*Haiken*
0	*0011*	*0000*
1	*0100*	*0001*
2	*0101*	*0010*
3	*0110*	*0011*
4	*0111*	*0100*
5	*1000*	*1011*
6	*1001*	*1100*
7	*1010*	*1101*
8	*1011*	*1110*
9	*1100*	*1111*
	abcd	*wxyz*

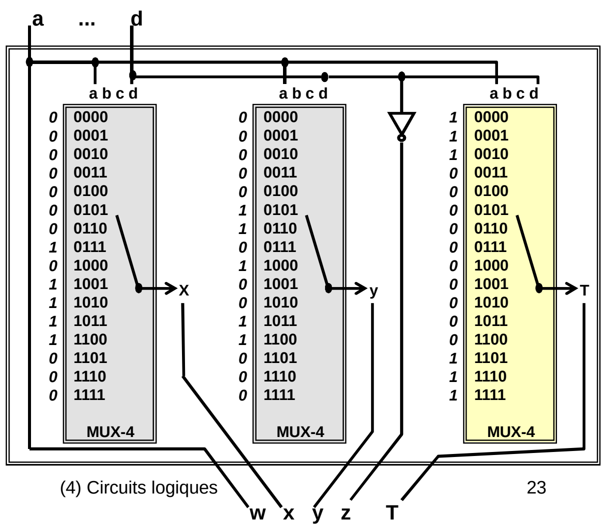


► Exemple de transcodeur

- **Code* excédent 3 des chiffres décimaux → code de Haiken**
- **Ajouter une 5ème sortie T pour prendre en compte les codes interdits en excédent 3**

4°) Fonction supplémentaire T

chiffre	*Exc. 3*	*Haiken*
0	*0011*	*0000*
1	*0100*	*0001*
2	*0101*	*0010*
3	*0110*	*0011*
4	*0111*	*0100*
5	*1000*	*1011*
6	*1001*	*1100*
7	*1010*	*1101*
8	*1011*	*1110*
9	*1100*	*1111*
	abcd	*wxyz*



TD n°4 : A rendre au plus tard le lundi 15 novembre 2010 à midi
(casier A. Sigayret à l'ISIMA)

► Exercice 2* du TD4
* arrivée 1/0 pour allumer/éteindre chaque "diode"

► Exercice 3 du TD4 : seulement 3.a et 3.b

► Exercice 4 (cf 4.a du TD4) :
- **Etudier un circuit réalisant le transcodage Binaire→Gray* (codage sur 2 bits)**
  ** Code de Gray : deux codes successifs ne varient que par une seule valeur (voir tables de Karnaugh)*

► Exercice 5 :
- **Etudier un circuit réalisant la multiplication par 2 d'un naturel codé en BCD (4 bits)**

► Exercice 6 :
a) Etudier un circuit vérifiant si deux entiers naturels binaires (2 bits) sont égaux
b) Même question pour comparer (en sortie, 00 : égal, 01 : supérieur, 10 : inférieur)