Représentation des données	Chap 02	Portes logiques et circuits combinatoires

2.1 Des formules aux circuits

De la logique à l'électronique

Dans le chapitre précédent, nous avons manipulé les valeurs booléennes **Vrai/Faux** et **1/0** comme des objets purement logiques. Pourtant, dans une machine, ces valeurs ont une **réalité physique très concrète**.

Un bit (contraction de *binary digit*) est la plus petite unité d'information qu'un ordinateur peut stocker ou traiter. Il ne peut prendre que deux valeurs :

$$0 \quad \text{ou} \quad 1$$

Dans l'électronique d'un processeur, ces deux valeurs correspondent à deux **niveaux de tension** :

0 = tension basse (absence de courant) $\qquad$ 1 = tension haute (présence de courant)

Porte logique

Une **porte logique** est un petit circuit électronique capable de réaliser une opération de l'algèbre de Boole sur un ou plusieurs bits. Elle reçoit en entrée des signaux électriques correspondant à 0 ou 1, et produit en sortie un nouveau signal également égal à 0 ou 1.
Chaque porte correspond à une opération logique précise :

$$\text{NON}, \quad \text{ET}, \quad \text{OU}, \quad \text{XOR}, \text{ etc.}$$

Circuit combinatoire

En combinant des portes logiques, on peut créer des **circuits combinaoitres** qui permettent de récréer n'importe quelle fonction booléenne. Grâce à ces circuits, on peut introduire des notiions comme l'addition, la comparaison etc. et donc générer un **processeur**.

2.2 Symboles officiels des portes logiques

Deux normes de représentation

Les portes logiques s'écrivent selon deux familles de symboles normalisés :

— la norme **ANSI** (américaine) : symboles arrondis ;

— la norme **IEC** (européenne) : symboles géométriques.

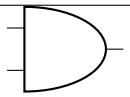
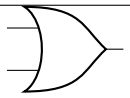
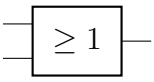
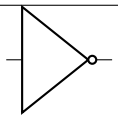
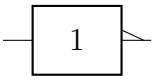
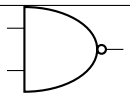
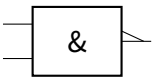
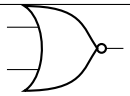
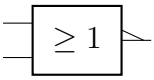
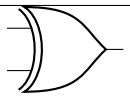
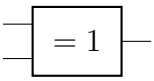
Opérateur	Symbole ANSI	Symbole IEC
ET ($a \cdot b$)		
OU ($a + b$)		
NON ($\overline{a}$)		
NAND ($\overline{a \cdot b}$)		
NOR ($\overline{a + b}$)		
XOR ($a \oplus b$)		

TABLE 1 – Table représentant les portes logiques

2.3 Passer d'une représentation à l'autre

2.3.1 De fonction booléenne à circuit combinatoire

Réaliser un circuit combinaoitre

Pour traduire une fonction booléenne en circuit combinatoire, on doit respecter l'ordre des opérations et créer les portes logiques petit à petit.

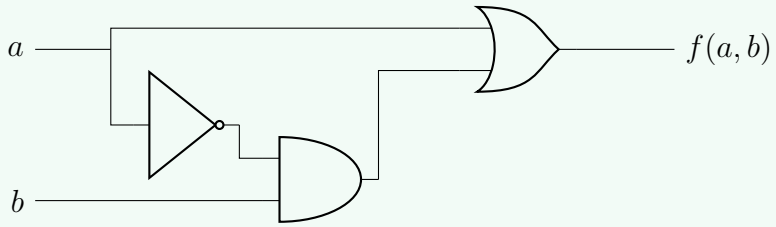
Exemple 1 — Une première fonction

On considère la fonction :

$$f(a, b) = a + \overline{a} \cdot b.$$

Pour la construire :

- une porte NON calcule $\overline{a}$;
- une porte ET calcule $\overline{a} \cdot b$;
- une porte OU combine a avec le résultat précédent.



Ce circuit réalise exactement la fonction $f(a, b)$: pour chaque couple (a, b), la sortie $f(a, b)$ est identique au résultat de l'expression algébrique.

2.3.2 De table de vérité à fonction booléenne

Méthode somme de produits

Une manière systématique de traduire une table de vérité en fonction booléenne est la suivante :

- on construit un produit logique (ET) décrivant cette ligne ;
- puis on fait l'addition logique (OU) de tous ces produits ;
- enfin on peut simplifier.

C'est la méthode dite **somme de produits**.

Exercice 1 — De la table au circuit

Table :

a	b	$f(a,b)$
0	0	0
0	1	1
1	0	1
1	1	0

1) Donner l'expression somme de produits.

---

2) Que reconnait-on ?

---

3) Dessiner le circuit correspondant.

Exercice 2 — On s'entraîne

1) Pour chacune des tables de vérité ci-dessous, proposer un circuit combinatoire (en utilisant les portes étudiées : NON, ET, OU, XOR, NAND, NOR…) qui réalise la fonction indiquée.

Table 1

a	b	$f(a,b)$
0	0	1
0	1	0
1	0	0
1	1	1

Table 2

a	b	c	$f(a,b,c)$
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	0

Table 3

a	b	c	$f(a,b,c)$
0	0	0	1
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

2) (Facultatif) Tester vos circuits sur un simulateur de portes logiques (par exemple `logic.ly/demo`) et vérifier que la sortie correspond bien aux tables de vérité données.

Exercice 3 — Bonus - Simplification et implémentation

Reprendre la fiche d'exercices du chapitre précédent sur l'algèbre de Boole.

1) Pour les fonctions des questions 3) et 4), construire les circuits combinatoires **avant** simplification, puis **après** simplification.

2) Comparer les circuits obtenus : nombre de portes, complexité des connexions… Expliquer en quoi la simplification des expressions booléennes est importante lorsqu'on passe à l'implémentation physique (coût, taille, consommation, fiabilité, etc.).

2.4 [BONUS] Vers une mini-calculatrice binaire

Construire une calculatrice en portes logiques

Une calculatrice, un ordinateur ou même un smartphone ne manipulent ni nombres décimaux, ni symboles " + ". À l'intérieur, tout est traduit en **bits** (0 ou 1) et en opérations très simples réalisées par des **portes logiques**.
Construire une mini-calculatrice en binaire permet de comprendre :

- comment une machine effectue réellement une addition ;
- comment des opérations complexes peuvent être obtenues à partir de quelques portes (NON, ET, OU, XOR…) ;
- comment se construit une **UAL** (Unité Arithmétique et Logique), le cœur de tout processeur.

En apprenant à additionner deux bits, puis deux nombres sur plusieurs bits, on reconstruit pas à pas le fonctionnement interne d'une véritable calculatrice numérique.

2.4.1 Demi-additionneur

Exercice 4 — Additionner deux bits : construire un demi-additionneur

Pour comprendre comment une calculatrice effectue une addition, commençons par le cas le plus simple : **additionner deux bits** a et b.
Lorsque l'on ajoute deux bits, on peut obtenir :

- une **somme** S ;
- une éventuelle **retenue** R qui vaut 1 en cas de dépassement.

Par exemple :

$$0+0=0, \qquad 1+0=1, \qquad 1+1=0 \text{ avec retenue } 1.$$

1) Compléter la table de vérité de l'addition binaire :

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

2) Déduire une expression logique pour la somme S.

3) Déduire une expression logique pour la retenue R.

4) À partir des expressions trouvées, citer les portes logiques nécessaires pour construire un circuit réalisant (S, R).

5) Dessiner le schéma du demi-additionneur en reliant correctement les portes.

2.4.2 Additionneur complet

Exercice 5 — Additionner deux bits avec une retenue

Lorsque l'on additionne des nombres binaires de plusieurs bits, la retenue obtenue sur le bit de poids faible doit être ajoutée au bit suivant. Pour cela, on utilise un **additionneur complet**.
Il possède trois entrées :

- a : premier bit à additionner ;
- b : deuxième bit ;
- R_{in} : la **retenue entrante** provenant du bit précédent.

Et deux sorties :

- S : la somme ;
- R_{out} : la **retenue sortante**.

1) Compléter la table de vérité du full adder. Pour chaque combinaison (a, b, R_{in}), déterminer la somme binaire correspondante (par exemple : $1 + 0 + 1 = 0$ avec retenue 1).

a	b	R_{in}	S	R_{out}
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

2) Déduire une expression logique pour la somme S.

3) Déduire une expression logique pour la retenue R_{out}.

4) Montrer qu'un additionneur complet peut être construit à partir de :

- deux demi-additionneurs ;
- une porte OU.

2.4.3 Additionneur 2 bits, puis n bits

Exercice 6 — Construire une mini-calculatrice binaire

On souhaite maintenant additionner non plus un seul bit, mais des **nombres binaires** sur plusieurs bits. Pour cela, il faut combiner les circuits étudiés précédemment (demi-additionneur et additionneur complet).

1) Expliquer pourquoi, lorsque l'on additionne deux nombres binaires $A = a_1a_0$ et $B = b_1b_0$, il est nécessaire de commencer par le bit de poids faible $a_0 + b_0$.

2) Proposer une manière de construire un **additionneur 2 bits** en utilisant :

- un demi-additionneur ;
- un additionneur complet.

Indiquer quel circuit traite le bit de poids faible, lequel traite le bit suivant, et où circule la retenue.

3) Généraliser la construction pour obtenir un **additionneur sur n bits**. Expliquer :

- combien d'additionneurs complets sont nécessaires ;
- comment se propage la retenue d'un étage à l'autre.